GH20-1292-0

Systems

IBM System/360 and System/370 ASP Version 3 Asymmetric Multiprocessing System System Programmer's Manual

Program Number 360A-CX-15X

The ASP system is a multiprocessing operating system that provides a compatible extension to the Operating System (OS). Designed for the user with a large computer job-shop environment, ASP provides increased automation of the computing operation. The ASP system functions as a programmed operator of OS. It provides advanced scheduling facilities for optimizing total installation production.

This manual contains information on how to generate the ASP system from the distributed tape and how to customize the ASP system to fill the needs of each specific installation. It also provides a functional description of the ASP program and its many parts.

IBM

PREFACE

This manual is intended for the use of the IBM System Engineer, the IBM Program Support Representative, or the system programmer at a current or prospective ASP installation. It contains information on how to generate the ASP system from the distributed tape and how to customize the ASP system to fill the needs of each specific installation. In addition it provides a functional description of the ASP program and its many parts. User modification and Dynamic Support Program guidelines are also provided. The material in this manual has been prepared on the assumption that the reader is thoroughly familiar with the IBM Operating System (OS).

Other publications currently available for ASP Version 3 are:

- ASP Version 3, Application Programmer's Manual GH20-1291
- ASP Version 3, General Information Manual GH20-1173
- ASP Version 3, Messages and Codes Manual GH20-1290
- ASP Version 3, Operator's Manual GH20-1289
- ASP Version 3, Reference Card GX20-1927

Availability of the ASP Version 3 Logic Manual will be announced in a Publications Release Letter.

First Edition (March 1973)

This is a new manual replacing the earlier System Programmer's Manual, GH20-0323. It applies to Version 3 of ASP (360A-CX-15X) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest System/360 and /370 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604. Comments become the property of IBM.

CONTENTS

ILLUSTRATIONS

CHAPTER 1. INTRODUCTION

This manual is designed to provide the necessary insight to install, customize, service, and otherwise use the IBM System/360 and System/370 Asymmetric Multiprocessing System known as ASP.

If the reader is unfamiliar with ASP it is highly recommended that he read the ASP Version 3 General Information Manual, GH20-1173, ASP Version 3 Operator's Manual, GH20-1289, and ASP Version 3 Application Programmer's Manual, GH20-1291, prior to using this manual. These manuals provide a broad view of the ASP system and serve as excellent prerequisite reading to the more detailed information contained herein.

ASP has been written to satisfy the requirements of a wide spectrum of large data processing users. Recognizing the need of each user to initialize his ASP system to meet his own specific needs this manual attempts to provide the tools necessary for such a task. Total system performance is greatly dependent on many factors over which the user has ultimate control. These factors include:

- Quality coding techniques in user modifications
- Selection of ASP Initialization control card parameters
- User job mix
- Amount of available storage
- System hardware configuration
- OS Tuning

ASP provides great flexibility in tuning the system for maximum performance. A thorough understanding of the material contained in this manual will increase the probability of a very successful ASP system.

CHAPTER 2. SYSTEM DISTRIBUTION AND MAINTENANCE

SYSTEM DISTRIBUTION

The ASP system is distributed by the IBM Program Information Department (PID) as a nonlabeled, multifile tape. This tape contains:

1. ASP PREGEN and POSTGEN jobs for OS SYSGEN.
2. A source library containing programs and macros which are OS release dependent. Object decks for these programs are in the PREGEN and POSTGEN jobs.
3. An object deck library containing assembled ASP modules suitable for link editing.
4. A source library containing all ASP programs and macros.
5. A sample program library containing maintenance decks and procedures.

The ASP Program Directory accompanies the distribution tape with a description of the tape's contents, data retrieval methods and instructions for adding the ASP programs to the system.

When applicable, a PTF tape will be sent with the distribution tape, which will contain updated maintenance changes to the ASP programs and new versions of the ASP PREGEN and POSTGEN jobs to support additional OS releases. ASP Version 3 supports OS MVT release 20.7 through release 21 only.

SYSTEM MAINTENANCE

When it is necessary to make a correction to an ASP module, the normal practice is to issue the fix as a source change which the user applies to his system. These changes are broadcast by the Field Engineering Retain system to all FE offices after the development group and the initial requestor have validated the fix.

When a number of changes to an ASP release has accumulated, these changes are packaged as a PTF which is shipped automatically by PID to ASP users of record.

The user may elect to apply fixes selectively as they appear in the Retain system or wait for a cumulative PTF before making a full update to the system.

There is a standard method of applying APAR fixes which the user is urged to follow. To implement the maintenance scheme (described below) a set of maintenance procedures and decks (called delta decks) is supplied on the ASP distribution tape.

There is a delta deck supplied for every module in the ASP system. The delta deck is a job which executes a special procedure consisting of two steps, an update step (using IEBUPDTE) and an assembly step.

THE METHOD OF USING THE DELTA DECK

The user punches the cards specified by the Retain fix for his problem. The change cards are placed into the proper module's delta deck in sequence number order. The delta deck is then placed in the users job stream for execution. The procedure executed by the delta deck performs an update against the ASP source library and the updated source is placed in a temporary data set, for use by the assembly step. The distributed source library should never be modified. Once the assembly is completed successfully the user has an object deck which is then linkage edited into a library of his choosing. It is advisable to maintain a separate maintenance JOBLIB data set, to receive updated ASP modules, to which the basic ASP JOBLIB is concatenated at execution time. Using the delta deck method of applying changes maintains the integrity of the ASP symbolic base because it is never altered directly. This ensures a consistent base against which changes may be planned by the ASP maintenance group.

The complete set of delta decks is maintained by the user and the change cards are accumulated in the individual delta decks. This also provides a convenient way to keep user modifications separate from the ASP symbolic base. Thus, all module changes, both user modifications and maintenance changes, are kept in one place where possible conflicts between user and maintenance changes may be easily found.

Experience has shown that any maintenance procedure which does not ensure the integrity of the ASP symbolic base causes unnecessary problems whenever an error occurs in updating that base.

CHAPTER 3. OS SYSTEM GENERATION

SYSGEN PREPARATION

The generation of an Operating System (OS) to be used in an ASP environment requires additional steps in the SYSGEN process to create the interfaces which ASP needs to manage the resources of the system.

There are four basic steps to the ASP/OS SYSGEN: ASP PREGEN job, OS Stage 1, OS Stage 2, and ASP POSTGEN job. Each one of these steps will require some special consideration by the ASP user and each step is discussed in detail later in this section.

In planning an ASP/OS SYSGEN the user should be aware of certain basic concepts of the ASP system which will help in understanding the detailed discussion of SYSGEN which follows. There are two types of ASP systems and certain commonly used terms which need to be understood.

1. Support Processor - CPU in which the ASP program resides

   Note: When using mixed release levels of OS, the highest level must be on the Support system.

2. Main Processor - CPU used to run user programs

   a. Real Main Processor - separate CPU from Support Processor

   b. Local Main Processor - same CPU as Support Processor

The ASP program communicates with the OS system on a real or local Main Processor via OS messages. These messages are sent over a channel-to-channel adapter (CTC), which connects the Support Processor to the Main Processor. In the case of a local Main the CTC connection is simulated, but this does not make a significant difference in the SYSGEN process. In the discussion that follows the terms real CTC, pseudo CTC, and CTC dummy tape devices will be used and must be understood by the user.

1. Real CTC - Physically present hardware CTC adapter. Generated as a 1052.

2. Pseudo CTC - Nonexistent CTC Adapter - represented by a 1052 console UCB in the OS system and used to simulate the message communication interface between the Support Processor and a local Main Processor.

3. CTC Dummy Tape Device - A nonexistent device represented by a UCB with a tape device type. Many of these dummy devices are needed on each Main Processor for allocation of input and output data sets used by jobs running on that processor.

Before starting the SYSGEN process the user must make certain decisions about the type of processor on which he intends to run. If he expects to produce a multipurpose system or one which may grow, then careful planning at the beginning may save work in the long run. The decisions basically concern the type and number of CTC devices to be generated for a given machine configuration.

Case 1. Single CPU - will be run as Support Processor and a local Main Processor, requires one pseudo CTC 1052 and a number of

CTC tape devices (see Stage 1 details below for calculating the number). All CTC devices may be generated on a nonexistent channel.

Case 2. Two CPU's - one Support Processor with a local Main and one real Main. The Support Processor needs one pseudo CTC console to support the local Main, one real CTC console device to support the real Main, and a number of dummy CTC tapes for the local Main. The real CTC must be SYSGENed on a real channel and all others may be on a nonexistent channel. The real Main Processor requires one real CTC console device for communications to support and a number of dummy tape devices. The real CTC must be on a real channel and the dummy devices may be on a nonexistent channel. At IPL time all CTC tape devices will automatically be placed on the same logical channel as the console CTC.

Case 3. Multiple CPUs. This case is an extension of Case 2. For every real Main added to the system, a new real CTC device must be generated on the Support Processor in addition to the one pseudo CTC console used by the local Main. Each real Main Processor must have at least one real CTC and a set of dummy tape devices as noted under Case 2 above.

In systems using symmetrical configurations (CPUs of the same model), it is possible to produce one SYSGEN which will run on any machine of that type either as a real Main or as a Support/local Main by generating all the necessary CTC devices to meet the conditions and then allowing ASP to establish the proper communications interfaces at initialization time. This type of system was not possible under previous ASP releases, without a great deal of work by the user, but changes in Version 3 have greatly simplified this process.

ASP PREGEN

The ASP PREGEN job is applicable to both Support and Main Processor SYSGENS. It is supplied as a complete job with JCL and is OS release-dependent. Always make sure that the PREGEN job is the proper one for the OS release being generated. A PREGEN job is supplied with the ASP system tape from PID but is current only for the OS release being used when the ASP system was released. New PREGEN jobs will be made available via PTFs as new OS releases are issued. These PTF's will generally be automatically shipped to ASP users and will contain specific instructions for running the PREGEN job. If there is any doubt concerning the applicability of a PREGEN job, contact the IBM FE program representative for clarification.

PREGEN allocates and cataloges two data sets: SYS1.ASP and SYS1.ASPMOD. The user needs to indicate the volume serial and unit type through symbolic parameters. The method of supplying this information is contained in the Program Directory supplied with the system from PID.

The PREGEN job is used to modify OS SYSGEN macros from SYS1.GENLIB and SYS1.MODGEN. The modified macros are placed in a new data set (SYS1.ASPMOD) which is used later by Stage I of SYSGEN.

PREGEN also linkedits a Type I SVC module, (ASPSVC) into a new data set (SYS1.ASP) for later inclusion in the OS nucleus, as SVC 246.

The main purpose of the PREGEN job is to include ASP modifications necessary to support the CTC device under the OS I/O supervisor. These changes are made to the NIP, IODEVICE, SGIEC202, IEAQNU and IOS macros before Stage I so that they are automatically included by SYSGEN.

OS RELEASE 20 AND 21 MODULES AND MACROS MODIFIED BY ASP PREGEN

ASP PREGEN MODIFICATIONS

SYSGEN Stage I

IODEVICE. Has been modified to permit FEATURE=CTC to be specified for tape and 1052 devices. A bit is set to indicate the presence of the CTC features on the device.

SGIEC202. Has been modified to add a CTC bit to each UCB requiring it. (The '04' bit is always set ON in the UCBTYP + 1 byte.) A switch is set to indicate the presence of a CTC device. After the IECIOS and IECXTCH macros are punched, the IECCTC macro is punched if the CTC switch is set. Also, CTC is added to the devices in IECTBL.

SYSGEN Stage II

IOS: Module IOS is assembled from several macros. ASP modifies several of these macros and includes one new macro: IECCTC.

IECCTC: This macro contains CTC device-dependent start I/O and trap code routines.

IECINT: Macro modified for special CTC interrupt handling of attention interrupts.

IECIOS: Macro modified to provide posting of ASP on local Main Processor and provide clearing of UCB flags in master CTC UCB at completion of I/O operation.

IECIOSB: Macro modified to provide special processing of Halt I/O on CTC.

IECTBL: Macro modified to provide unique device table entries for CTC.

IECXTCH: Code added to prevent doing test channel on CTC I/O request for local Main Processor.

IEAANIP0: Module NIP is assembled at SYSGEN from the IEAANIP macro. ASP modifies NIP to provide special CTC console processing in determining the master console of an ASP real Main Processor.

IEAQNU: A change is made to the OS program check first level interrupt handler to distinguish interrupts which occur in supervisor state and protect key of zero.

STAGE I OS SYSGEN

ASP requires that certain items be included in Stage I of SYSGEN. They are discussed below by topic.

JCL

One addition is made to JCL. The SYSLIB card for the assembly must place the data set (SYS1.ASPMOD) first in order of concatenation:

```
//SYSLIB   DD  DSN=SYS1.ASPMOD,DISP=SHR
//         DD  DSN=SYS1.GENLIB,DISP=SHR
```

It is also recommended that the SYSPUNCH output of Stage I be assigned to a tape.

IODEVICE MACRO

An I/O device macro must be included for each real or pseudo CTC device. All such devices must be generated as a 1052 with FEATURE=CTC. This is an important change from previous releases.

```
Example:  IODEVICE UNIT=1052,ADDRESS=270,MODEL=7,FEATURE=CTC
```

A set of dummy tape devices must be defined by using the IODEVICE and IOCONTRL macros and should be designated as 2400 tape drives with FEATURE=CTC. These devices may be on a nonexistent channel; if so, a CHANNEL macro must be included for that channel. See example below. There are two restrictions concerning the assigning of dummy tape devices:

1. All dummy tape devices must be on the same channel. This channel may be different than the one used for the CTC console, however.

2. The addresses used for dummy tape devices must not duplicate the second and third digits of the device address used for the CTC console device, even though these devices may be on different channels. For example, a system may be generated with CTC console devices at addresses 270, 280, and 760 (these may be a combination of real and pseudo CTC's) and all dummy tape devices generated on nonexistant Channel 7. In this case no dummy tape device may use address 760, 770, or 780. Addresses 781 and higher may be used for tape device addresses. The second and third digits of the unit address are used by ASP to identify the source of the data being transferred across the CTC.

To determine the number of dummy tape devices required, take the number of initiators which will normally run on that Main Processor and multiply by the average number of SYSIN and SYSOUT data sets per job step. If, for instance, there will be six initiators being scheduled by ASP on a Main and an average of five SYSIN or SYSOUT data sets per job step, then 30 devices is a reasonable number to generate for that Main. If the number is under estimated, the system performance will be affected. OS allocation recovery will cause the job to be canceled.

The following example shows how to generate the CTC devices for a local Main Processor which will also support a real Main.

Examples:

```
IODEVICE    UNIT=1052,ADDRESS=370,MODEL=7,FEATURE=CTC  -  Real CTC
CHANNEL     ADDRESS=7,TYPE=SELECTOR  nonexistent channel
IODEVICE    UNIT=1052,ADDRESS=780,MODEL=7,FEATURE=CTC
IOCONTRL    UNIT=2803,ADDRESS=78,MODEL=1,FEATURE=16-drive
IODEVICE    UNIT=2401,MODEL=3,ADDRESS=(781,15),FEATURE=(9-TRACK,CTC)
IOCONTRL    UNIT=2803,ADDRESS=79,MODEL=1,FEATURE=16-drive
IODEVICE    UNIT=2401,MODEL=3,ADDRESS=(790,16),FEATURE=(9-TRACK,CTC)
```

Note: ASP does not support priority queuing on CTC dummy tapes.

UNITNAME MACRO

The user must code a UNITNAME macro to include the name CTC as a unit type for allocation purposes. The devices specified as CTC should include all CTC dummy tape devices, but not those CTC devices defined as consoles.

```
Example:    UNITNAME UNIT=((371,31)),NAME=CTC
```

Also the user should supply an alternate unit name for 2400 tape drives so that unit type 2400 may be avoided in JCL usage. The reason for this is that CTC dummy tape drives are defined as 2400 tapes and problems in allocation can occur if dummy tapes and real tapes are not distinguishable by separate unit names.

```
Example:    UNITNAME UNIT=(180,4),NAME=TAPE9
```

SCHEDULR MACRO - CONSOLE DEFINITION

ASP requires that the CTC (either real or pseudo) be designated the alternate console for OS. If a system includes both a pseudo and real CTC, the pseudo CTC must be the alternate console. If such a system is IPLed as a real Main Processor (master console out of ready at IPL) the real CTC will automatically be made the master console by ASP modifications in NIP.

If the user wishes to use the ASP facility provided to write the ASP MLOG to disk (DLOG=YES on the STANDARDS initialization card) the OS LOG facility must be included in the system. The OS LOG is a default option on the SCHEDULR macro and will be generated if not specifically excluded.

It is recommended that the MCS option not be used in an ASP system unless it is needed to support a graphic console as a master OS console. If MCS is required then certain rules must be followed:

1. The CTC is to be designated as the alternate console.

2. Take default route codes on SCHEDULR macro.

3. A SECONSLE macro must be coded for the CTC with no route codes (default value) and ALTERNATE=MASTER (this is the default).

4. Do not code SECONSLE macros for consoles to be controlled by the ASP program unless those consoles will be varied offline to MCS before starting ASP.

5. Be aware that an MCS system with a graphic console requires a hard copy log on a console or SYSLOG and adds overhead which is redundant in an ASP system. Also, MCS may cause operational problems in an ASP system because of operator capability to vary and switch consoles at will.

Below are examples of SCHEDULR and SECONSLE macros coded for an MCS/ASP system, showing only the parameters important to ASP.

```
SCHEDULR    CONSOLE=1E0,ALTCONS=780,CONOPTS=MCS

            (Omit ROUTCDE parm and VLMOUNT=AVR)

SECONSLE    CONSOLE=780,ALTCONS=1E0  (Omit ROUTCDE parm)
```

SUPRVSOR MACRO

The DDR parameter of the SUPRVSOR macro may be coded. However, to avoid possible interlock conditions between ASP and DDR, it is advisable to make the ASP queue volumes and the ASP JOBLIB volume permanently resident.

SVCTABLE MACRO

ASP requires one type I SVC, number 246, in the SVC table.

Example: SVCTABLE SVC-246-T1-S0

RESMODS MACRO

The following example must be coded to include SVC 246 into the OS nucleus.

Example: RESMODS PDS=SYS1.ASP,MEMBERS=(ASPSVC)

REPRO CARDS

Place the following cards near the end of the Stage I deck, just before the GENERATE (or GENTSO) macro:

```
          REPRO
//SYSLIB  DD   DSN=SYS1.ASPMOD,DISP=SHR
          REPRO
//        DD   DSN=SYS1.MODGEN,DISP=(SHR,PASS)
```

These are the DD cards which will be needed in Stage II assemblies of NIP and IOS, and must be on the Stage I output tape if the user elects to use the SYSGEN tape convert program. (See next section STAGE II OF SYSGEN for details of tape convert program.)

STAGE II OF SYSGEN

A few changes to the Stage II job stream are necessary for ASP, but because of the difficulty of handling this job stream in card form it has become advisable to make available some assistance in this area. A sample program is supplied on the ASP distribution tape, which will produce a modified job stream on TAPE, from an input tape produced by Stage I.

The program, called CONVERTT, is supplied in source form with instructions for its use as comments in the source deck. It is supplied as a sample program and although it has several optional functions such as an assembler H option and a multiple job stream option, the user may wish to add more capabilities for special needs. It is recommended that the user punch the program from his distribution tape and examine the instructions for its use even before doing the Stage I assembly. To take advantage of the programs capability requires that repro cards be added to the end of the Stage I deck, as noted in the discussion of Stage I.

For those users who wish to modify their Stage II job stream manually, the following list of changes is provided:

1. Alter JCL for NIP program check FLIH and IOS assemblies.

a. Find the assembly steps for NIP program check FLIH and IOS. Their OBJMOD member names are: IEAANIP0, IEAQNU00 and IEAASU00 respectively.

b. In the JCL for the above three assemblies, replace the //SYSLIB DD cards with the following two cards:

```
//SYSLIB    DD  DSN=SYS1.ASPMOD,DISP=SHR
//          DD  DSN=SYS1.MODGEN,DISP=(SHR,PASS)
```

SPECIAL CONSIDERATIONS OF ASP SYSGEN

1. OS Maintenance

   OS PTF's which affect ASP modified OS modules create some special conditions for the ASP user. It is sometimes necessary to reapply selected portions of the POSTGEN job following the application of an OS PTF. Field Engineering will be notified via the Retain system when these special cases occur. It is a safe practice to always ask for verification from the IBM FE program support representative before applying new OS PTF's to an ASP system.

2. Adding CPU's

   When adding additional CPU's to an ASP complex always be sure to provide the necessary CTC devices on the Support Processor to handle the number of real Main Processors to be driven.

3. MP65 Considerations

   a. When an MP65 CPU is used as a real Main, the CTC device must be physically cabled to the CPU which is IPLed. It is possible to attach the CTC through a switching device such as the 2914 to the MP65 so that more flexibility is provided in the use of the MP65. This allows IPL from either half of the MP65 and/or partitioning either half of the MP65 to remove it from the ASP complex.

   b. The MP65 is not supported as a Support Processor with local Main capability.

   c. In a full MP65 system the CTC device is nonsymmetrical which, as noted in the OS Planning Guide for MP65 systems, may result in some degradation of performance.

4. CTC Lock-out condition.

   Any control program requires unrestricted access to its vital resources to avoid performance degradation or absolute lock-out. Load module libraries or queue data sets are examples of critical resources requiring frequent accesses.

   In configuring a multiple-CPU environment, particular consideration must therefore be given to the resources which are shared between CPU's. In an ASP configuration, where commonly used disk files are shared between the Support and Main Processors, a potential for this performance or lock-out problems exist.

   These problems may be avoided by:

a. Ensuring that the channel used to access the CTC from the Main Processor is not the same Main Processor channel used to access the shared DASD control unit on which reside any data sets referenced by the ASP region. In addition to the normal data sets, such as JOBLIB, queue packs, and CHKPNT, the data sets referenced by user DSP's must be examined.

b. Ensuring that no program which runs on a real Main Processor reserves a shared disk pack on which reside any data sets referenced by the ASP region. This can be avoided, for example, by routing any program such as a SUPERZAP job which is modifying ASP's JOBLIB, to the local Main Processor.

While it is not necessary, both of these conditions may be satisfied by insuring that the ASP referenced data sets are not shared with any other processor.

5. ASPDRDS - ASP DR Data Set

The ASP Disk Reader (DR) is another primary way of introducing job streams into the ASP system, along with the card and tape readers. This function uses as an input file an operator designated member of a partitioned data set. This file (member) may consist of any job stream material that is readable by CR (except for card-image input, which is CR exclusive).

ASP does not require the existence of this PDS, but a write-to-operator message is issued at ASP initialization time if this data set has not been described in the initialization deck. Normal OS BPAM macros are used to access this data set. The OPEN occurs during ASP initialization; DR itself issues the FIND, READ, and CHECK macros.

A member (file) may be a job, a series of jobs, or a job network. The data set and its members may be created and maintained using an OS utility: IEBGENER, IEBUPDTE, etc. Actually, multiple data sets may exist, but they all must be concatenated using only one ddname card in the initialization deck.

The following DD cards are exemplary:

```
1.  //ASPDRDS   DD  DISP=OLD,DSN=ASP.ALWAYS

2.  //          DD  DISP=OLD,DSN=REPORTS.MONTHEND

3.  //          DD  DISP=SHR,DSN=ACCOUNTING,UNIT=SYSDA,  X
    //              VOL=SER=ASPINP

4.  //          DD  DISP=SHR,DSN=ASP.TESTING
```

As can be seen, the described data sets may be cataloged (1, 2, 4) or uncataloged (3). The data sets may be part of an index structure (1, 2, 4) or they may not (3). If a data set is to be modifiable <u>while ASP is running</u>, by a job that is either under the control of ASP or a "hot job", the DD card must have DISP=SHR (3, 4); stable data sets should have DISP=OLD (1, 2). Care should be exercised in order to ensure that DR is not reading a data set at the same time that maintenance is being performed on one of its members. If a data set is forced into secondary allocation, the overflow material will not be accessible.

The DD statement used to create a DR data set must include DCB=(RECFM=F,LRECL=80,BLKSIZE=80) for an unblocked data set, or DCB=(RECFM=FB,LRECL=80,BLKSIZE=multiple-of-80). All data sets in a concatenation must have identical DCB characteristics. An easy

way to ensure this, after the first one has been created, is always to reference the "standard" one. Using the example above:

```
//SYSUT2    DD  DCB=ASP.ALWAYS,DSN=A.NEW.DR.INPUT,...
```

(See the OS Data Management Services Guide manual for a discussion of the concatenation of data sets.)

A large blocking factor makes efficient use of DASD space, but may require that the ASP region size be increased to provide sufficient main storage without impacting the rest of ASP. Deblocking is done by ASP, directly from one input area (GET-LOCATE type logic).

The data set is OPENed by INITGEN. The DCB is in RESPARAM, and includes the following:

```
DCB DDNAME=ASPDRDS,KEYLEN=0,LRECL=80,RECFM=FB
```

ASP POSTGEN JOB

Once Stage II of SYSGEN is complete the new system is available as an OS system but does not yet have all the necessary modifications for ASP to function.

The ASP POSTGEN job is supplied to complete the modification of the OS system. Primarily, modifications are made to modules which are not assembled by SYSGEN. Modifications are made using the service aid IMASPZAP (Superzap) and in some cases program DELINK1 is also used.

The ASP POSTGEN job is supplied as an in-stream procedure which may be tailored to a users system through EXEC card parameters. The job is OS release dependent and all of the comments above concerning the applicability and availability of the ASP PREGEN job also apply to POSTGEN. Because of the release dependent nature of the POSTGEN job, the instructions for its use are included with the job itself.

The basic functions of the POSTGEN job are:

1. Linkedit to a temporary data set two programs:

   ASPPROGM (a small driver program used by POSTGEN)

   DELINK1 (service aid delinkedit program)

2. Execute ASPPROGM to perform necessary renaming of programs and generate condition codes for succeeding steps.

3. Linkedit the ASP CTCOPEN module to SVCLIB.

4. Delinkedit and relinkedit OS programs which need to be expanded to provide superzap patch areas.

5. Superzap modules in SYS1.SVCLIB and SYS1.LINKLIB which require ASP modifications.

6. Add the program name MAINTASK to the table of system programs in SYS1.LINKLIB.

7. Marks the OS Reader/Interpreter modules with a REENTRANT attribute so they can be made resident in Main storage. See "Main Storage Considerations" in the "OS Reader/Interpreter Under ASP" section of Chapter 6 for a discussion of how and what to make resident.

OS RELEASE 20 AND 21 MODULES MODIFIED BY ASP POSTGEN

POSTGEN MODIFICATIONS

All ASP POSTGEN modifications are made by linkedit or Superzap of OS modules.

IFG0192Z: ASP supplies its own CTC OPEN module which becomes part of OS tape OPEN and provides special CTC OPEN handling. Release 21 only.

IFG0193A: Tape OPEN modified to cause linkage to the CTC OPEN module IFG0192Z.

IFG0194H: Tape OPEN modified to prevent sense commands being issued to CTC devices during tape open.

IFG0552X: End of volume modified to prevent CTC concatenation to unlike devices unless unlike attribute bit set in DCB.

IFFSD061: CSECT IEFW42SD modified to issue SVC246 for ASP unit isolation at step termination.

IEFW21SD: CSECT IEFW21SD modified to issue SVC246 for ASP unit isolation at step allocation time.

IEEVLNKT: Modified to add name 'MAINTASK' to OS table of system tasks.

IGC0107B: Modified to remove the backspace character check.

IGC0801C: Modified to change ABEND DCB BLKSIZE to 764.

IGC0L05A: Modified to change ABDUMP maximum block size to 764.

IGC5403D: Modified to bypass the second translate when a successful quote is found.

IGG0190A: is an OS open module, renamed to IGG0190$. Release 20 only.

IGG0190A: is an ASP module used to issue a WTOR to give unit assignment, jobname, stepname and ddname for CTC devices. Other open processing is bypassed in the area of label checking. Control is passed to IGG0190$ for non-CTC tape devices. Release 20 only.

IGG0550N: is an OS module. It is modified to treat CTC devices as a unique device type. Release 20 only.

IGG0199C: is an OS module. It is modified to avoid issuing a sense to CTC devices. Release 20 only.

IGC0201C: is an OS module. It is modified to make BLKSIZE=764 as in IGC0L05A. Release 20 only.

IGC0003E: Is an OS module for WTO. It is modified to place jobnames in WTO/WTOR messages, and to pass reply ID's directly to IJP.

GENERATING HASP REMOTE TERMINAL PROGRAMS (RMTGEN)

Intelligent (programmable) remote terminals, such as System/360 (including the Model 20), System/3, 2922, and the 1130, are supported by ASP with the use of the HASP Remote Terminal Programs. Generation of the HASP Remote Terminal Program package is described in the

documentation supplied with Houston Automatic SPOOLing Priority System II (HASP II), Program #360D-05.1.014.

In general, the user is required to allocate and catalog HASP data sets, execute the HASPGEN job, and then execute RMTGEN job(s). See the section Remote Generation For Non-HASP Users in the HASP documentation supplied with the system.

CHAPTER 4. FUNCTIONAL DESCRIPTION

The program that implements the ASP operating system resides in the Support Processor. It is written in OS Macro Assembler Language, using the execute channel program (EXCP) level of OS to perform its input/output operations. The ASP program is added to a private Job Library, and is loaded and executed as an OS job on the Support Processor.

Once ASP has been loaded and execution has been initiated, ASP assumes control of its task on the Support Processor. From this time until ASP is purposely quiesced or abnormally terminated, ASP assumes control of all devices and consoles assigned to it via ASP initialization control cards, and it becomes the primary job scheduler for all Main Processors assigned to it. All OS control program services and facilities remain effective and many are utilized by ASP, however, the use of some of these facilities by ASP is restricted. These restrictions are discussed in this manual in the chapter on writing Dynamic Support Programs (DSP's).

The ASP program consists of the following major components:

- ASP Resident Programs (ASP Nucleus)
- ASP Dynamic Allocated Resources (Buffers, Main Storage, Devices)
- ASP Dynamic Support Programs (DSP's).

The resident portion of ASP is composed of the service programs that are common to the ASP dynamic support functions. The following are the major components of the resident portion:

- ASP Disk Input/Output program
- ASP Console Service program
- Job Control Routines
- ASP Multifunction Monitor
- ASP Resident Parameter Table
- Call DSP Driver program
- ASP DSP LOAD and DELETE programs
- ASP GETMAIN and PUTMAIN programs
- ASP Unit Record and Tape Device programs (ASPEXCP, ASPOPEN)
- Channel-to-Channel Adapter Interrupt Handler
- ASP Unit Allocation programs (GETUNIT and PUTUNIT)
- ASP Initialization driver

The above programs are linkedited together as the ASP nucleus (ASPNUC) and are loaded by OS when ASP is scheduled for execution. (For a detailed description of all the modules that comprise the ASP Nucleus, refer to the ASP Logic Manual.) The Initialization program receives control and performs the system initialization by reading the

ASP Initialization control cards and performing the required tasks. After ASP has been initialized, the ASP Initialization program releases the storage that it occupied, and the system is ready for execution. All ASP modules may optionally be made resident by use of the RESIDENT Initialization control card.

The basic ASP buffer pool is built by the Initialization program. It consists of a number of fixed-length buffers, which reside in a contiguous area of storage. The number and size of the buffers should be determined by the type of direct access storage devices being used for ASP queue's, by the size of the Support Processor, and by the number and types of support functions that may be active at one time. The size of the buffer pool should be set to accommodate all primary functions with their minimum requirements. At peak processing times, the ASP system will construct temporary buffers in free storage areas to satisfy any additional buffer requirements.

The remainder of storage is allocated to the Dynamic Support Programs and modules loaded by initialization. DSP's are resident on the direct access storage device that is allocated for ASP system residence DSP's are loaded into storage by the ASP Job Segment Scheduler as they are required and are scheduled on the basis of jobs to be processed and available devices. Once a DSP is scheduled and obtains the devices assigned to it, it becomes a specific support function. For example, when Print Service is loaded and is assigned printer number 1, it becomes the printer 1 (PR1) support function.

DSP's are loaded via the ASP load function (ALOAD), which verifies that sufficient storage is available before OS is called to perform the actual loading and relocation. A DSP may, in turn, call subprogram modules via ALOAD , provided that the DSP also deletes those modules via ADELETE prior to terminating its execution. If a nonresident ASP module is to be loaded into hierarchy 1 storage , a linkage editor SETSSI control card must be added to the linkedit step for that module. ASP modules that are to be made resident in hierarchy one core, which are loaded via OS LOAD at initialization time, must be linkedited with the hierarchy attribute. The SETSSI control card format is:

```
SETSSI   Fxxxxxxx
```

where the x's represent normal SETSSI information. Scatter load modules with CSECT's of differing hierarchy attributes are loaded within a single hierarchy.

A DSP may be callable from an operator console, as well as from the job stream. If the DSP is callable, it must be so identified in the ASP DSP Dictionary . Further, DSP's may be constructed to support multiple devices simultaneously. For example, Print Service can process output for jobs on as many printers as are attached to the Support Processor. To support multiple devices, a DSP must be programmed to be reentrant and to have a control section identified for loading each time that it is scheduled. Reentrant DSP's are scheduled as long as there are jobs to process and devices on which to process them. Multiple copies of a nonreentrant DSP may also be active (in storage) if so specified for that DSP in the ASP DSP Dictionary (MLOAD=YES).

ASP INITIALIZATION

The ASP initialization deck is actually an OS job that is read into the system as unblocked (i.e., BLKSIZE=80) via an OS reader and enters the OS job queue. When OS schedules ASP the ASP nucleus (ASPNUC) is loaded into storage. At this point the INITIATE module is the first module of ASP to gain control. INITIATE's function is to load other modules of the ASP initialization program. The first modules loaded are INITDATA,

a CSECT used by the initialization modules, and INITRTNS, a module containing common initialization routines.

The module INITIATE then loads INITIOCD. INITIOCD processes the ASP initialization cards until it encounters the ENDASPIO card. Control then returns to INITIATE which deletes INITIOCD and loads INITIO.

INITIOCD constructs the following:

- ASP save area pool
- ASP track table

INITIOCD issues the OS OPEN macros for the ASPABEND and ASPSADMP data sets.

INITIO constructs the following:

- ASP Buffer Pool
- ASPIO IOBs
- All ASPIO Tables (AIOPARMS - TATPARMS)
- File Directory
- Track Allocation Table
- Formatted Queue Packs (when required)
- Single-Track Table
- DCB's for ASP queue packs

INITIO issues OS OPEN macros for the ASP queue packs and then returns control to INITIATE. INITIATE deletes INITIO and loads INITREST.

INITREST opens the ASP checkpoint data set and builds a job number table. If initialization is being performed because of a restart of ASP, INITREST rebuilds track allocation and job number table to reflect jobs still present in the ASP job queue. INITREST is deleted and INITIATE loads INITCARD.

INITCARD processes the remaining initialization cards until it encounters the ENDINISH card. All cards read by INITCARD are spooled to the ASP job queue by ASPIO routines. The name of each card determines which multirecord file that card is spooled into. After initialization card processing is complete INITIATE loads the module INITGEN.

INITGEN analyzes the initialization cards spooled to the ASP queue pack and builds the following resident tables from information contained in the cards:

- Preallocated Control Blocks

  RESQUEUE

  FCT's

- Assignment Table
- SYSUNITS Table
- SUPUNITS Table

- Printer Resource Table
- Deadline Scheduling Table
- SYSOUT Class Table
- NJP Terminal Table

INITGEN loads the appropriate modules to create the following tables:

- Main Processor Control Table(s)
- MDS Table(s) (SETUNITS)
- RJP Table(s)
- Reader/Interpreter Table(s)

INITGEN also loads the module RIATTACH which loads the Reader/Interpreter resident modules, and attaches the OS R/I as a subtask of ASP.

INITGEN OPENS the partitioned data set containing jobs to be input via DR, as defined on the //ASPDRDS DD statement.

INITGEN gives up control to INITCNS who initializes the ASP console requirements:

- Load Attention Interrupt handler
- Console buffer pool
- Console Control Tables (CONSDATA, CONSUNIT,CONTVTBL)
- Device Dependent Parameters
- Dummy and RJP Entries
- Switch Master Console with Dummy CTC (local main)
- Program Function Key Table

RESIDENT ASP PROGRAMS (ASP NUCLEUS)

The resident ASP programs may be grouped into the following major categories:

- ASP DSP Dispatching and Job Segment Scheduling
- ASP Tables and Control Blocks, and other routines for inter-system communication including Resource Management, Work-To-Do and Unit Record I/O routines.
- ASP Console Service (CONSOLE SERVICE)
- ASP Disk I/O Services (ASPIO)
- ASP DSP Failsoft (AFS)

The section that follows describes the above resident programs of ASP. These programs are those that are always resident. It should be noted however that most of the above programs consists of several modules and

many of the modules do reside on disk. Among the functions of the resident modules is the loading of associated nonresident modules as required. ASP initialization provides the programmer with the facility to make any module permanently resident. All ASP modules are discussed in detail in the ASP Program Logic Manual.

ASP DSP DISPATCHING

ASP DSP Dispatching is accomplished initially by the ASP Job Segment Scheduler (JSS) and after DSP initialization by the ASP Multifunction Monitor (MFM).

JOB SEGMENT SCHEDULER (JSS)

The Job Segment Scheduler (JSS) initiates the processing of a job segment by a DSP. Job segments are defined as units of work accomplished by DSPs, such as print, punch, or Main Processor execution. Segments of the same job (for example, print and punch), if scheduled in parallel, are considered to be non-sequence-dependent or asynchronous. Other segments (for example, computing on the Main Processor) require scheduling in specific order and, therefore, are sequence-dependent and are considered to be synchronous. The Scheduler selects its work from the ASP job queue , which is ordered by length of time in the system within priority. The Scheduler begins its search for job segments with the job at the top of the queue. If all segments in a job are already active or complete, or may not be scheduled, the next job in the queue is examined. The search continues until all resources are scheduled or until there are no more jobs to schedule. If the examined segment is not active, the basic criteria for scheduling it are:

- Availability of all required devices
- Satisfaction of any sequence-dependent or synchronous processing requirements for the job
- Availability of sufficient storage in the Support Processor for the required program and table entries

The requirement for sequence-dependent processing is also associated with the DSP and is indicated by the presence of a flag in the DSP Dictionary entry. This flag indicates that:

- All segments preceding this one must be completed prior to initiation of this segment
- No segment following this one may be initiated until this one is complete.

An example of the use of this feature is the assurance that all preprocessing segments are completed prior to the scheduling of the Main Service function. This feature also assures that the Main Service function is completed prior to the scheduling of the postprocessing segments.

When device requirements and synchronous processing requirements are satisfied, ASP control blocks necessary for DSP processing are constructed and the DSP programs required are ALOADed into storage. At this point JSS initial DSP scheduling is complete. Further DSP scheduling during the DSP's program execution is accomplished by the ASP Multifunction Monitor (MFM) until the DSP processing is complete, at which time it returns to JSS for DSP termination processing. JSS DSP termination processing is essentially the reverse of JSS DSP initiation processing in that obtained devices and control blocks are

returned to system availability, loaded programs are deleted from the system, and the ASP Job Control Table (JCT) entry for this job is updated to reflect the completion of the DSP and the job segment.

MULTIFUNCTION MONITOR (MFM)

The Multifunction Monitor (MFM), which is contained in the module ASPCONTL , allows the routines within the Support Processor to share processing time. Each routine operating in this time-sharing environment is called a function. Each function within the Support Processor reaches a point beyond which it cannot proceed until the occurrence of an event. The event may be the completion of a print cycle, the return of a buffer to the buffer pool, or some similar event. When a function must wait for an event, control reverts to the Multifunction Monitor via an AWAITOFF or AWAIT macro. The Multifunction Monitor then scans the Function Control Table for the function with the highest priority whose AWAIT event is complete; control is given to that function.

Function Control Table (FCT)

Each active function in the system has an associated entry in the FCT . The function's entry in the FCT is ordered in an ASP assigned priority. The FCT priority is a one-byte field with X'00' being lowest and X'FF' being highest. FCT priority is relative to ASP functions in handling jobs and is specified in the functions associated DSP dictionary entry. It is in no way related to OS job priority. For example, the function, CONSOLES, has the highest FCT priority (255). This allows the console operator highest priority in system control.

The FCT chain is expanded or contracted as functions become active or terminate; thus, the FCT reflects the active components of the system at any given time. The FCT entry, which is used for a register save area when the function is not in control, contains pointers and flags associated with the function. Register 11 points to the FCT entry for the function in control, thereby providing access to the pointers and flags.

AWAIT Macro-Instruction

By issuing the AWAIT macro-instruction, a function notifies the Multifunction Monitor that it is unable to proceed. The operands of this macro are a return point address, a mask containing a condition code, an Event Completion Flag (ECF) address, or a list containing multiple event completion flag addresses and their associated condition masks. An ECF is a one-byte field. The mask indicates which bits in the ECF are to be tested. The return point address is the location within the function to which control passes when the event has occurred. The condition code defines a satisfied condition.

Register 11 points to the FCT entry for the relinquishing function. The contents of the FCT entry include the address of the ECF, the ECF mask or the address of the ECF list that contains multiple ECF addresses and ECF mask entries, and the return address.

MFM Function

The MFM searches for a ready function (that is, a function whose event is complete). Starting with the highest priority entry in the FCT, the MFM picks up the address of the ECF. Using the mask supplied, it examines the ECF. If the result of this test does not satisfy the

requirement, then the next entry in the FCT is tested. If an ECF list is used, each entry in the list is tested to satisfy the ready function requirement. If any entry in the list is satisfied, the function ready requirement is satisfied. If the result of this test satisfies the requirement, the registers are reloaded, and control returns to this function.

ASP SYSTEM CONTROL BLOCKS AND TABLES FOR ASP SYSTEM COMMUNICATION

The primary means of intercommunication with the Multifunction Monitor, the Job Segment Scheduler, and the Dynamic Support Programs is via a set of interrelated tables and associated data sets. Figure 1 illustrates the relationship between the basic tables, blocks, and files of the ASP Supervisor. They are presented here for conceptual purposes only; the reader should refer to the ASP Logic Manual for complete definitions of the tables and blocks.

The disk-resident Job Control Table (JCT) defines all jobs in the system and, in this manner, represents the ASP job queue. The JCT is ordered by job priority (15 through 0), and jobs are entered on a first-in-first-out basis within each priority. Job priorities 0 through 14 are programmer-assigned; priority 15 is reserved for operator use. As a job enters the system, a JCT entry is constructed and is placed in the table. The Job Segment Scheduler scans the JCT and schedules work on the basis of the jobs in the system.

The Function Control Table (FCT) is used by the Multifunction Monitor for sharing time between functions. Certain functions such as CONSOLES, MDS, JSS, etc., are permanently resident. Their FCT entries are contained in the Resident Parameter Table (RESPARAM). FCT's of non-resident ASP functions are created dynamically when the function becomes active. These FCT's are chained to the FCT list. The DSP associated with the active function keeps a pointer to that entry in general register 11. By using register 11, the DSP can access any portion of its FCT entry.

One of the fields in the FCT entry is a pointer to an entry in the Active Job Description/Accounting Block Table. This table identifies all active DSP's except those with permanently resident FCT entries. The entry contains the File Description Block (FDB) for the single record data set containing the Job Description/Accounting Block (JDAB) for the job processed. The DSP may access this JDAB by pointing to the FDB and using an AREAD macro-instruction.

In the case of the Main Service function, two FCT entries are constructed at initialization for each Main Processor defined. These entries work in conjunction with the Resident Job Queue Table (RESQUEUE) to control scheduling and execution of the Main Service job segment of every job that calls for Main Processor execution. The Main Device Scheduler (MDS) FCT controls the pre-execution device allocation for jobs requiring setup.

The FCT entry also contains a pointer to a GETUNIT list, which describes the devices required by the DSP by type and may contain null fields for undefined device requirements. After devices have been allocated to the DSP, each GETUNIT list entry contains a pointer to the Support Units Table (SUPUNITS) entry for that device. The SUPUNITS entry contains such information as the type of device, its device name, device address, and UCB address , and a pointer to the appropriate System Units Table (SYSUNITS) entry. The SYSUNITS Table provides further information about the device and indicates whether the device is allocated, assigned, or offline.

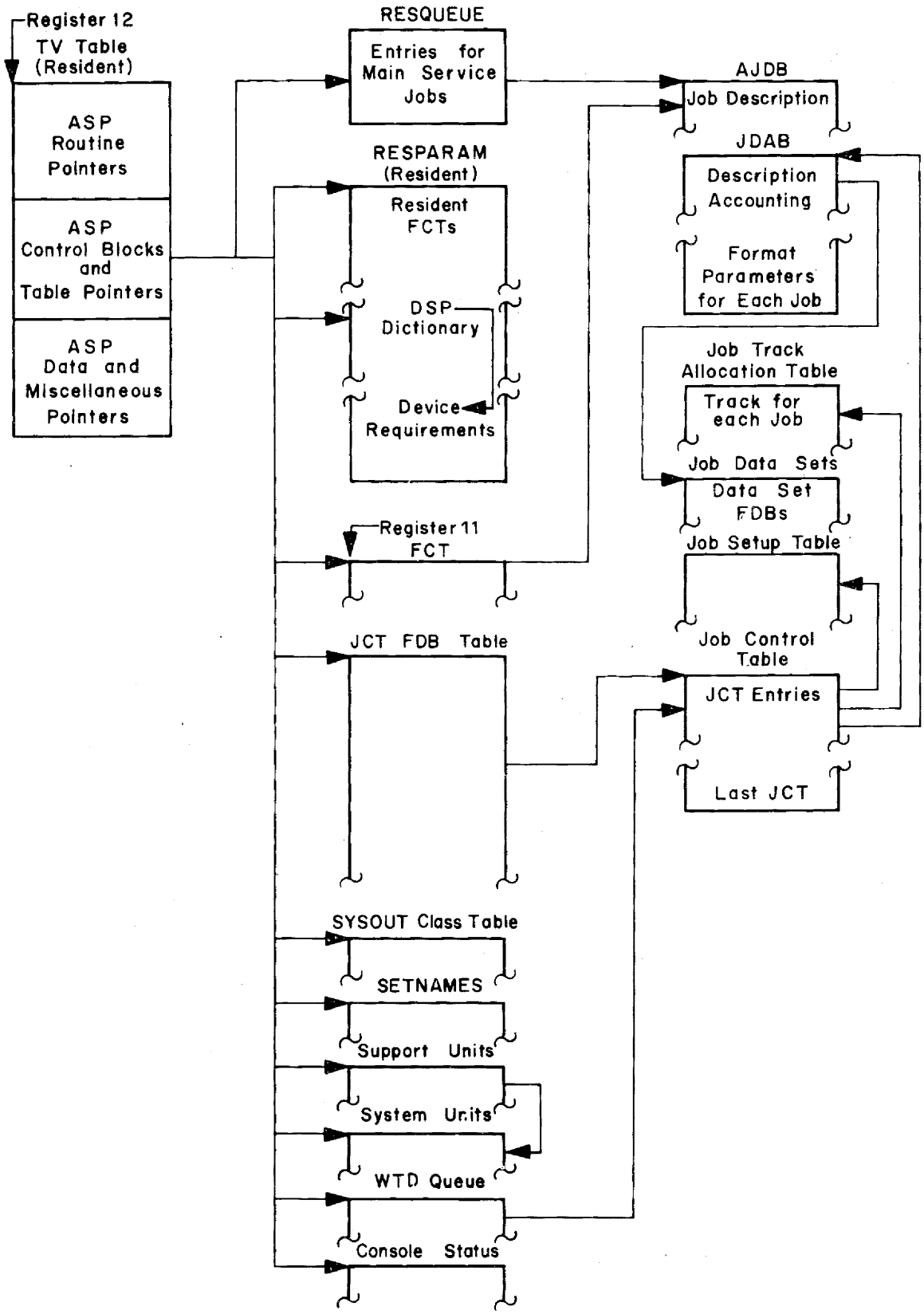


Figure 1. Relationship between the basic Tables, Blocks and Files of the ASP Supervisor.

Another field in the FCT entry is a pointer to the entry in the DSP Dictionary for this function. The DSP may access this information but must not change it.

The main control table is the TVT. The TVT is always pointed to by register 12 and contains pointers to the resident routines and system related information.

The Scheduler Elements, which are located in the JCT, define the processing tasks for the job. The Scheduler Element sequence number, which is found in the JCT entry for the job, is copied into the FCT from the JCT at the time this function is initiated. The DSP may use this number to locate data associated with this function in the Job Description/Accounting Block (JDAB). (Scheduler Elements are numbered sequentially within a job, beginning with 1.)

The Job Description and Accounting Block (JDAB) is constructed for a job at the time the job enters the system. It defines the job, its data sets, and associated processing parameters. The JDAB contains its track allocation table, accounting information for the job, and a pointer to the Job Data Sets Block (JDS), where File Description Blocks are placed for all data sets associated with the job. Entries are generated for the system message data set (SYSMSG), JCL input (JCLIN), and each data set defined by DD * or DD DATA.

The File Description Block associated with the Scheduler Elements in the JDAB is used to access single record parameter buffers. These parameter buffers contain any information required by the DSP. They contain any data cards, following a //*PROCESS card, that establish the associated Scheduler Element. The parameter buffers for Print Service and Punch Service are generated through the use of //*FORMAT control cards and are in a specialized format. They may be accessed and altered by a DSP. The Scheduler Elements and associated FDB's are not in any fixed location in the JDAB relative to the DSP. The proper ones may be located easily, however, by comparing the Scheduler Element sequence number in the FCT entry with the sequence number portion of the Scheduler Elements in the JDAB. In the same manner, the proper entries for Job Accounting may be located to log time-on and time-off for a portion of a job.

The DSP Dictionary entry contains a pointer to an entry in the Device Requirements Table and a count of the number of requirements associated with this program. The Device Requirements Table entries indicate the types of devices required by the DSP.

The Job Control Table, Job Description Accounting Block, and Job Data Sets block, are located on direct access storage devices and are brought into storage as they are needed. The Function Control Table and the GETUNIT list are established by the Job Segment Scheduler, as are the Active JDAB Table and the Resident Job Queue Table. The Support Units and System Units Tables are created during ASP Initialization. The DSP Dictionary and the Device Requirements Table are located in the ASP Resident Parameter (RESPARAM) module and are constructed when that table is assembled.

JOB CONTROL TABLE ROUTINES

The Job Control Table (JCT) routines are used to access and alter the JCT. For each routine, there is a macro-instruction creating a calling sequence that branches to the corresponding JCT routine. These macro-instructions are described in Appendix A. The JCT macros and routines are:

MACRO	JCT ROUTINE
TAADD	Table Add
TADEL	Table Delete
TAFIND	Table Find
TAGET	Table Get
TAPUT	Table Put
TARESET	Table Reset

The Job Control Table is a queue of work for the ASP system, ordered by length of time in the queue within job priority. To allow for an unlimited number of entries, this table is stored on a direct access storage device and is accessed by the ASP Input/Output routines. Each of the 16 priority levels is treated as a separate file of chained, single buffer records (that is, the File Description Block for record n + 1 is found in record n).

The JCT single buffer record contains:

- The track address for the single record file
- The FDB for the next record in the chain
- The displacement of the terminator (X'FFFFFFFF') within the record
- JCT entries

Associated with the JCT routines is a table in the CKPTAREA called the JCTFDB Table . This table contains a 32-byte word alligned entry for each of the 16 priority levels. It is ordered with the entries for priority 0 first and the entry for priority 15 last. The contents of each entry in this table are:

- The FDB for the first record in the chain of this priority
- The FDB for the last record in the chain of this priority
- The priority of this level

Because several users could be attempting to update the same record, access to this table must be synchronized. This is accomplished by use of the AENQ and ADEQ routines discussed under Generalized Resource Management later in this chapter.

Access to the JCT tables is made for a specific priority level only. The AENQ must specify the priority level desired. Only one user (FCT-DSP) may access a priority level at any given time. The priority level is not available until an ADEQ for that priority is issued.

Example:

```
LA      R0,15
AENQ    NAME=JCT,PRTY=(R0)
LA      R0,15
TAGET   PRTY=(R0)
LA      R0,15
ADEQ    NAME=JCT,PRTY=(R0)
```

The following list describes the purpose of each JCT routine:

- Table Add routine. The Table Add routine is used to insert an entry in the JCT. The AENQ routine must be used to gain access to the JCT priority prior to using the Table Add routine. Table Add uses the priority field of the entry to be added and the JCTFDB Table to access the last record in the chain for the required priority. If the last record of the chain contains sufficient room, the new entry is inserted. If there is insufficient room or if the chain contains no record, a new record is created. The newly created record is attached to the end of the chain. Note that when the first record is created, it is also the end of the chain. The ADEQ routine must be used following Table Add to make the JCT priority available for other users.

- Table Get routine. The Table Get routine accesses the next entry in the JCT. Prior to the first use of the Table Get routine, the AENQ routine must be used to ensure that the table is available. The first use of Table Get returns the address and length of the first entry in the JCT of the priority specified. Each successive use of the Table Get routine (with no intervening use of the Table Reset routine) returns the address of the next entry. When there is no next entry, an end-of-file return is made.

- Table Put routine. The Table Put routine writes the current record in the chain on the direct access storage device. This routine is used when a table entry has been altered. Note that the Table Get routine must be used prior to the use of this routine.

- Table Find routine. The Table Find routine locates and returns the address of a specific job's entry or the first entry in a given priority as specified by the user. This routine allows the user to access JCT entries within a given priority (using Table Get). The first use of the Table Get routine following the use of the Table Find routine returns the address and length of the next entry in the specified priority level. If used, Table Find must be executed as the first JCT routine following JCT table acquisition via the ASP AENQ macro.

- Table Delete routine. The Table Delete routine deletes a completed entry from the JCT. A completed JCT entry has the event completion flag in each Scheduler Element of the entry equal to 1. As an entry is removed, the record is compressed. When a record is empty it is deleted.

- Table Reset routine. The Table Reset routine is used to reset the JCT scan pointers to the top of the priority level currently being accessed.

RESIDENT JOB QUEUE TABLE (RESQUEUE)

When the job segment to be scheduled is for Main Service, no FCT entry is created by the Job Segment Scheduler. Two FCT entries, MAIN and MAINIO are created during system initialization for each Main Processor defined. The MAIN FCT controls the system IPL, job scheduling, execution monitoring, and job termination. The other MAINIO controls all input/output between the Main Processor and the Support Processor. For each job segment scheduled for Main Processor service, an entry is created by JSS in the Resident Job Queue Table (RESQUEUE) . Each RESQUEUE entry contains the File Description Block (FDB) for the Job Description Accounting Block (JDAB) and the Job Setup Table (JST), the ASP job number, job priority, and other fields used by Main Service in controlling the execution of the job on the Main Processor.

RESQUEUE ORGANIZATION AND ACCESS

RESQUEUE is chained from RQTOP in the TVTABLE through the RQNEXT field, however this chain will not be in priority order. It will act as a collector only. Both Main Service and MDS utilize subchains through the RQGRPCHN field in priority order and based either in MDSDATA or a Main Group Table. The chain in which a RQ entry exists is indicated by the RQINDEX field in the RESQUEUE entry.

This chaining is controlled by three existing RQ macros RQTAADD, RQTADEL, and RQTAPUT. RQTAADD and RQTADEL are used to enter/delete a RQ entry with the addition that chaining and dechaining through RQNEXT is implicit. RQTAPUT is used to move a RQ entry from one chain to another, or change the priority within a chain. The RQTAADD and RQTAPUT macros have an INDEX= operand to indicate into what chain the RQ is to be entered. RQTADEL and RQTAPUT use the existing RQINDEX in the RQ entry to determine from what chain to remove the entry. INDEX= may be omitted with the following results:

RQTAADD will use the index in the entry (JSS for example builds the RQ with either an index for MDS or Main therefore he might omit the INDEX rather than having two RQTAADD macros).

RQTAPUT will use the index in the entry for both the dechaining and rechaining resulting in performing aging for Main and MDS and a priority change from a MODIFY command.

The macros are presented in Appendix A.

GENERALIZED RESOURCE MANAGEMENT

ASP resources such as the Job Control Table (JCT) and the Resident Job Queue Table (RESQUEUE) must only be accessed by one DSP at a time. To protect against concurrent use of these system resources ASP provides the resource management routines, AENQ and ADEQ. These routines, located in the ASP nucleus, must be used to access the ASP resources. ASP resources that are used serially are defined in the TVTABLE through use of the RESOURCE macro.

The following macro-instructions access the resource management routines which perform the function indicated:

- AENQ - Obtains exclusive use of the resource named in the macro parameter. The macro will wait for the resource to become available or return to a specified address.
- ADEQ - Releases control of the named resource.
- ATEST - Interrogates the status of the resource.

Additional information on the macro-instruction mentioned above is contained in Appendix A.

WORK-TO-DO DRIVER (WTDDRVR)

The Work-To-Do Driver (WTDDRVR) periodically causes the JCT records on the ASP queue to be compressed and also through the scan module processes a created queue of JCT accesses.

Work-To-Do Driver issues a TACMPR macro every 30 minutes and will cause JCT records in the ASP queue to be compressed, i.e., get as many JCT entries into a single ASP buffer as possible.

In addition, the Work-To-Do Driver issues an ATIME macro of five seconds to initiate processing of the WTD queue. The WTD queue is created by Inquiry, Modify and any other system function that requests asynchronous JCT processing. A scan module, WTD-JCT , is ALOADED, given control and processes the WTD queue. The WTD queue is deleted and work-to-do driver issues another ATIME and allows five seconds for another WTD queue to accumulate. If, at the end of five seconds a WTD queue does not exist, WTDDRVR issues an AWAIT and waits for a WTD queue to accumulate.

TAPE AND UNIT RECORD I/O ROUTINE

A resident module, ASPOPENX, is used by ASP for all tape and unit record handling. All reading and writing to Support Processor devices, both directly attached (local) and those devices attached via Remote Job Processing (remote), is accomplished by use of the ASPOPENX routine.

Entry to ASPOPENX from a DSP is accomplished by issuing one of the ASPOPENX related macro-instructions. The I/O device affected is identified to the macro. In order to designate an RJP (remote) device in the ASPOPENX macro RJP must be active and the desired terminal must be signed on.

The macro-instruction and the function ASPOPENX performs when they are issued by the DSP are as follows:

- ASPOPEN - Initializes the DCB and DEB (created by the ASPDCB macro) If a remote device is specified in the macro parameter, ASPOPENX branches to the ROPEN routine of RJP. If a local device is specified in the macro parameter, ASPOPENX inserts an abnormal end appendage.

- ASPEXCP - An OS EXCP is issued for local devices (this normally requires an ASPOPEN macro to be previously issued). Remote devices will cause ASPOPENX to branch to either an RREAD or RWRITE routine of RJP.

- ASPEOV - For RJP devices ASPOPENX sets a unique op code and then branches to RWRITE of RJP. ASPEOV is essentially a NOP when specified for a local device.

- ASPCLOSE - For local devices any outstanding I/O requests will be purged and the DCB is marked closed. For RJP devices, ASPCLOSE marks the DCB closed and branches to the RJP routine RCLOSE.

CONSOLE SERVICE

Console Service, a resident group of modules of the ASP system, provides communication between the operator and the ASP system. The two classes of communication are:

- Input Messages, initiated by the operator, consist of command or action responses directed to various DSP's within the ASP system. Input messages may also be entered from the input stream from one of the readers; CR, DR, or TR.

- Output Messages, initiated by the ASP DSP's or any Main Processor, consist of job status messages, replies to operator inquiries, or operator action required messages.

The Console Service functions are:

- Reading messages and responses into the system from the consoles

- Checking validity of messages entered from RJP workstations according to the level specified on the appropriate RJPTERM card
- Interpreting the verbs in the messages and responses
- Routing the messages and responses to the system functions that perform the required actions
- Queuing messages and responses that are to be sent to the consoles from the functions
- Writing messages and responses to the consoles
- Automatic switching in the event of console failure, if desired.

DSP-to-Operator Communication

DSP to operator communication is accomplished via a MESSAGE macro used by the DSP. The MESSAGE macro allows the DSP to send a message to a class of consoles. Each console is defined as receiving certain class messages. This class is defined in each console's CONSOLE card The MESSAGE macro also allows DSP communication to be directed to a specific console.

The text format for messages includes a message identifier unique to the DSP that issued it. For example, 'MSV01 identifies the first message from Main Service. The message text, including the message number , may contain a maximum of 70 bytes. The format of the MESSAGE macro and its options are discussed in Appendix A, Macro Instructions.

Operator-to-DSP Communication

The operator communicates with the DSP via the operator command language. The link between the operator and the DSP is by device name or the unit address of a device assigned to the DSP. In addition, an operator may use the DSP name, provided that there is only one copy of the DSP active at the time. When a DSP initiates execution, it must identify itself to Console Service via the LOGIN macro . This macro (defined in Appendix A) specifies the entry point of an asynchronous program to which control should be passed for message processing. A module within Console Service called INTERCOM allows the programmer to simulate operator input messages from a DSP.

The asynchronous message processing program of a DSP receives control directly from Console Service whenever a message is received for the DSP. This message processing program must perform the following tasks:

1. Evaluate whether to accept, reject, or request that Console Service queue the message on the FCT to be retrieved later.
2. Identify the command code and set the necessary flags, internal to the DSP, which will cause the message to be acted upon when the DSP is next entered from the Multifunction Monitor.
3. If necessary, satisfy any AWAIT condition to ensure entry from the Multifunction Monitor.
4. Extract any pertinent data from the message for future processing. This usually entails moving the entire message into a temporary work area.
5. Return control to Console Service with an indication of whether or not the message has been accepted, or if it should be queued.

6. Keep to a minimum the DSP processing within the message appendage.

Console Service creates the following conditions upon entry to a DSP asynchronous message processing program:

- Register 1 points to the message preamble. A DSECT for this preamble can /be generated with the CONDSECT macro
- Register 14 establishes the return point to Console Service.
- Register 15 establishes the entry point of the message processing program. Returning to Console Service Register 15 will contain the following code to indicate the action taken:

  0 normal return

  4 message is to be queued

  8 message is to be rejected

The DSP asynchronous message processing program is entered via the ASP Save Routine which saves the working registers. It must return control to Console Services via the ARETURN macro

The preamble for an incoming message is illustrated in Figure 2.

+0	+1	+2	+4	+5	+6	+7	+8
Unused	Authority Level	Console Number	Buffer Control Flag	Char Count	Action Code	Scan Displ	Text
1 Byte	1 Byte	2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	

Figure 2. Arrangement of Fields in Input Message Preamble. The preamble and flags are equated in CONDSECT.

The entries are defined as follows:

- Character count. A binary number that specifies the number of characters in the message text, plus eight (for the preamble).
- Action code. A binary number that identifies the verb in the message. The codes are:

VERB	CODE
START	01
RESTART	02
CANCEL	03
SEND	05
FAIL	11
FREE	21

CALL	41
VARY	42
INQUIRY	43
MODIFY	44
DELAY	81
ERASE	82
MESSAGE	A1
SWITCH	C1
DUMP	E1
RETURN	E2
DISABLE	F1
ENABLE	F2

- Scan displacement. A binary number which, when added to the contents of register 1+8 (the beginning of the text), points to the character in the message that follows the last character examined by Console Service (that is, the first character of the first parameter).

- Console number. The number of the console from which the message was transmitted. This console number is the number used with the MESSAGE macro to route a reply back to the console of origin.

  Note: Consoles on RJP workstations have the high order bit on plus the remote number in the two-byte console number.

- Message. The message text, a variable-length entry which may contain a maximum of 128 bytes.

Special Operator Functions

The verbs START, RESTART, CANCEL, communicate with active DSP's. Console Service interprets these commands and takes one of the following actions:

- If the requested program is in storage, Console Service branches directly to it.

- If the requested program is not in storage, Console Service will reject the message, and send an indicative message to the operator.

(The remaining commands are processed internally by Console Service or invoke programs, either resident or nonresident, to provide special operator functions.)

Console Message Buffer Pool

A preallocated buffer pool, 96 bytes per buffer, is used for both input and output console messages. Each time a buffer is required for an operator message, Console Service will obtain it from this buffer pool.

When the buffer is no longer needed it will be placed back in the pool. If there are no available buffers or a message is too large to fit in one buffer, another buffer will be obtained via an AGETMAIN. Each buffer obtained via an AGETMAIN will be returned via an APUTMAIN.

When an output message requires an action or reply, the message will be placed in the outstanding reply queue. When the reply is received the DSP issues a macro, DEQMSG, to dequeue the outstanding reply buffer.

The number of buffers allocated to Console Service is defined in the STANDARDS card . Buffer depth , the number of messages that may be queued by DSP's for a specific console, is defined in the CONSOLE card .

Console Errors

Console Service will display the sense bytes, status bytes, and operation code on all console error conditions. In the event that a console is out of ready and buffer depth is reached, or a permanent I/O error, Console Service will switch automatically to the alternate console specified in the CONSOLE card, or to the first available console in the console status table if an alternate was not specified. If there is no alternate or available console in the console status table, ASP will ABEND.

Console Service Message Processing

Console Service uses the Function Control Table (FCT) to route the messages and responses to the appropriate functions. The appendage entries are specified by each function through the use of the LOGIN macro -instruction. Each message and response received by Console Service from the operator contains a verb and either a communication noun or a keyword. Console Service scans the FCT for a match between the communication noun and the function names for currently active functions. Upon encountering a match, Console Service branches to the entry point specified in the FCT.

Console Service has an entry in the Function Control Table. While Console Service is not processing messages, it is AWAITing. The event completion flag (ECF), pointed to by the AWAIT macro of Console Service and by the address portion of Register 0 of the FCT entry, is posted when a message is ready to be sent either by the operator or by a system function.

Figure 3 illustrates the flow of an input console message or response (that is, from the operator to the system). The steps in the flow are:

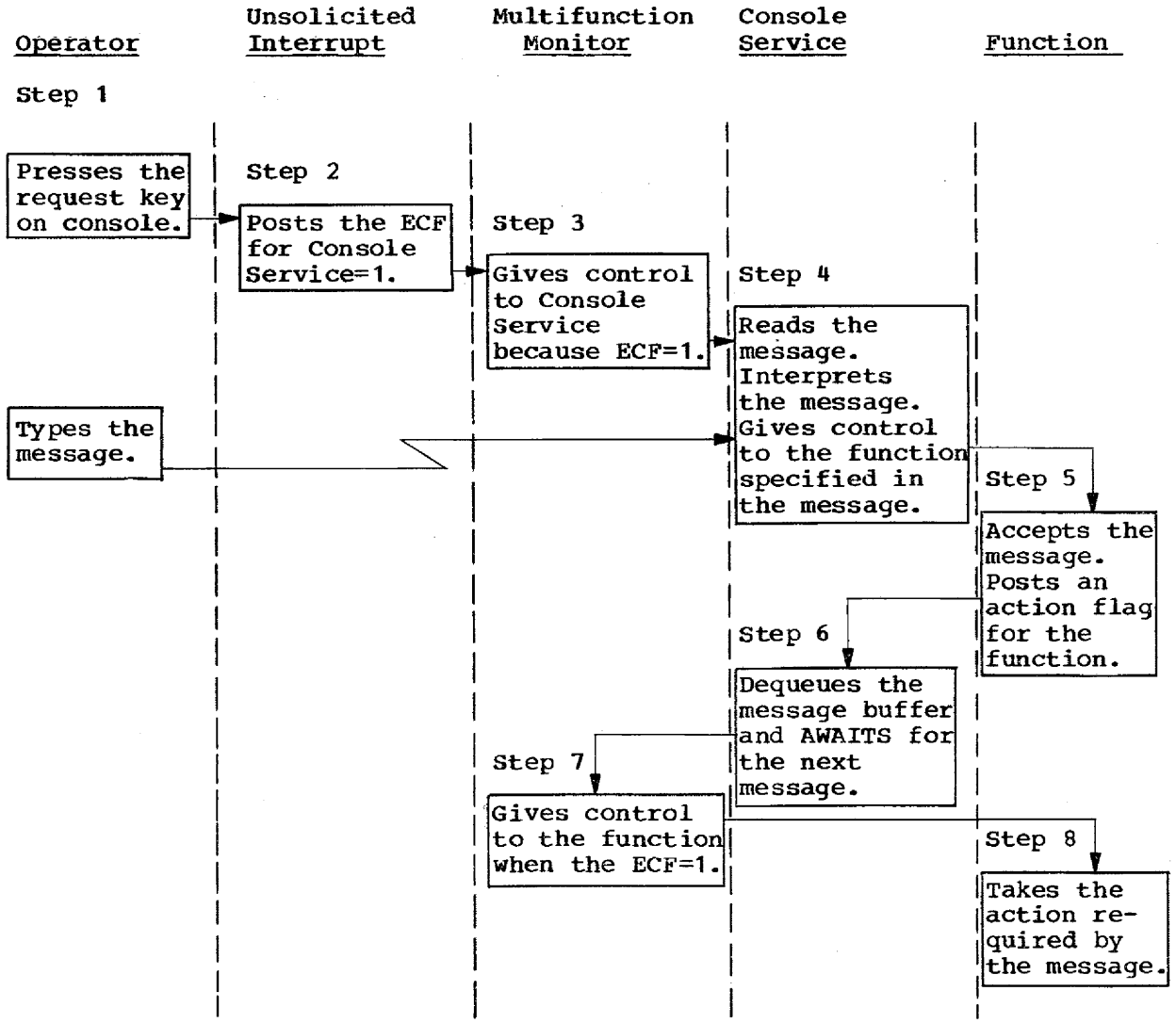


Figure 3. Flow of an Input Message.

1. The operator presses the request key (or equivalent) on the console, causing an unsolicited input/output interrupt.

2. The processing of this interrupt causes the ECF for Console Service to be posted. After the ECF is posted, control returns to the function that was in execution when the unsolicited interrupt occurred. This function eventually returns control to the Multifunction Monitor via an AWAIT.

3. The Multifunction Monitor then passes control to Console Service.

4. When Console Service has control it looks sequentially at each entry in the Console Status Table. The Console Status Table contains information on each console based on initialization parameters. An attention bit set, indicates a specific console requires service. Console Service branches to the appropriate device-dependent routine as directed by the Console Status Table. Each console has a channel-end appendage as part of its device-dependent routine . The appendage is entered after each I/O channel end is received to perform error checking and to post the Console Service ECF .

Console Service reads the message into storage and analyzes it for proper authority level as defined in its CONSOLE card The module, CONSAUTH, is used for local consoles and in the distributed system will treat any non-designated authority as a level 15. A user-written CONSAUTH may be implemented to meet special console authority requirements. Refer to "Writing a CONSAUTH Module" in this chapter. For RJP consoles the module, CONSANAL, is used to analyze the authority specified in the RJPTERM card.

The message, if allowed, is then interrogated as to command and is processed if possible.

When commands are entered through the input stream, the command is queued on the console that called the particular reader. When it is queued it looks the same as if it was entered from that console.

5. When Console Service passes control to a function, the function either accepts or rejects the message, or requests that the message be queued by Console Service off the functions FCT The latter may be done if the function is currently processing another message.

   The function sets its own action flag. This action flag indicates to the function (when it later receives control from the Multifunction Monitor) that it has received a message upon which action must be taken. The function posts its ECF only if the event upon which the function is AWAITing is the receipt of a message or response. For example, in the event that a function has sent a message to the operator requesting a response, the function AWAITs until the response has been accepted by the function. Then, once the ECF has been posted, control can again be passed to the function through the Multifunction Monitor. The function returns control to Console Service. (Note that the action required by the message response is not taken at the time Console Service passes control to an asynchronous message processing program of the function.)

6. Console Service either dequeues the message buffer or queues it for the DSP to process at a later time, and AWAITs for the next message or response. With the AWAIT , control passes to the Multifunction Monitor.

7. The Multifunction Monitor scans the FCT entries for an ECF that has been posted. After the ECF has been posted for the function that accepted the message or response, the Multifunction Monitor passes control to the function.

8. The function checks its action flag (the flag set in Step 5) and detects the receipt of a message or response that requires action. The required action is taken by the function at this time. Input console messages from RJP terminals are preprocessed by CONSANAL to determine validity and to apply defaults before being passed to CONSINPT for processing. Local consoles may be preprocessed by CONSAUTH.

Figure 4 illustrates the flow of an output console message (that is, from the system to the operator). The steps in the flow are:

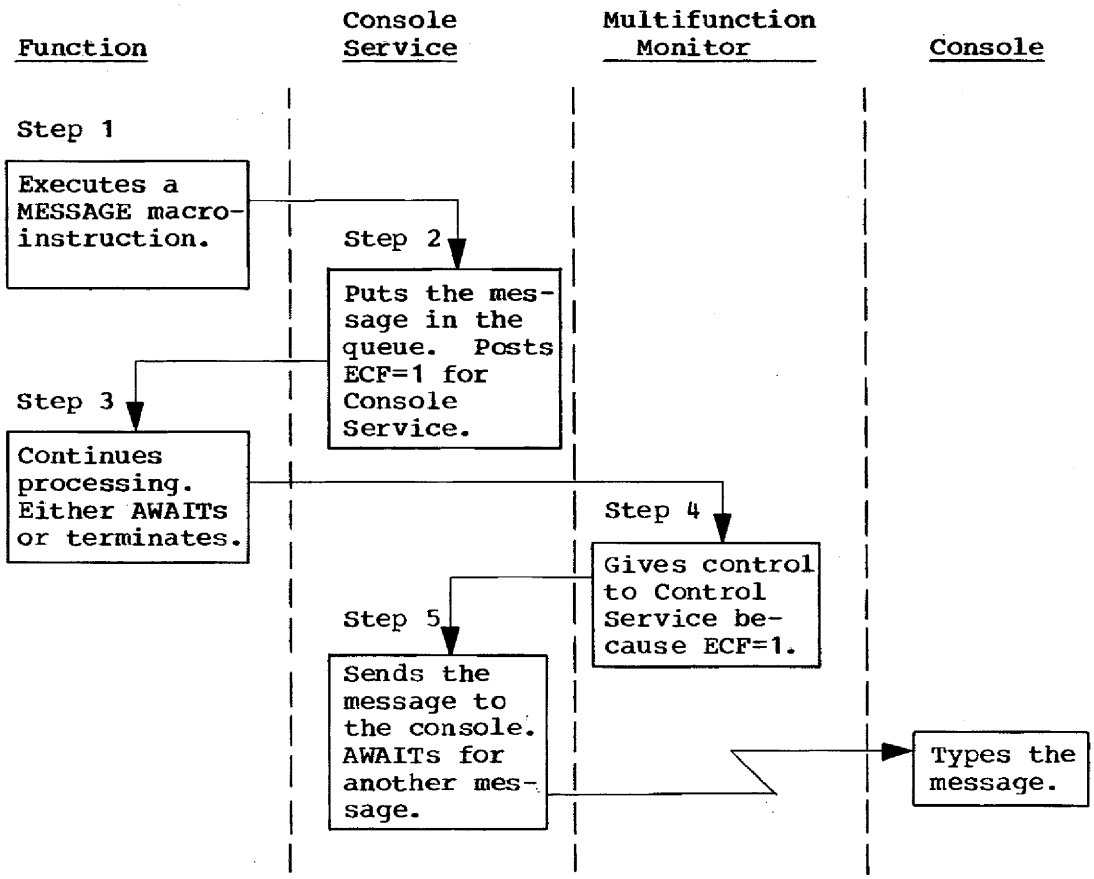


Figure 4. Flow of an Output Message.

1. The flow of an output message is initiated when a function executes a MESSAGE macro-instruction. The expansion of the MESSAGE macro includes a branch to a portion of Console Service called the CONSQMGR routine.

2. The CONSQMGR routine places the message from the function into an output message queue for the appropriate console or consoles as specified by the destination code. The CONSQMGR routine returns control to the function that executed the MESSAGE macro-instruction. A DEQMSG macro dequeues console and action messages in the buffer pool when they are no longer required.

3. The function continues processing until it eventually AWAITs on some event or terminates. In either case, control returns to the Multifunction Monitor.

4. The Multifunction Monitor passes control to Console Service.

5. Console Service transmits the message to the console. Console Service AWAITs for the next message.

There are two other conditions not mentioned above that can cause the Console Service ECF to be posted and thereby gain control. They are:

1. A console timer interrupt causes the ECF to be posted. Time intervals are issued for 2740 Prepare commands and for user-specified graphic console message delay intervals. Output

messages to the consoles are inhibited during the specified interval. When the timer interrupt occurs, the Console Service ECF is posted to allow any pending output activity.

2. When Console Service issues a message, the MESSAGE macro obtains buffers from the buffer pool, or, if there are no buffers available, an AGETMAIN is issued. In the event the AGETMAIN cannot obtain a buffer the BUSY exit is taken. When storage becomes available the Console Service ECF is posted.

Writing a CONSAUTH Module

This module is provided as a user exit to perform local console authority checking on input messages before they are processed.

CONSAUTH is called from CONSINPT. The registers are saved before entry and therefore, must be restored before return. This is accomplished by using the ARETURN macro to return control to CONSINPT.

Upon entry to CONSAUTH Register 0 points to the beginning of the message prefix (mapped with CONDSECT TYPE=INPUT macro), and Register 9 points to the Console Status Table entry for the input console. Register 1 points to a word (the address of CONSAUTH) in the Console Service data area. The zeroing of that word prohibits entry into the CONSAUTH module.

There are two valid return codes from CONSAUTH. A return code of 0 is the normal return, indicating that message processing is to continue. A return code of 4 indicates that the message is not allowable from that console and results in message processing termination for that message and the printing of the following:

CNS03 'message-text' REJECTED, UNAUTHORIZED REQUEST

The above mentioned return codes are the only two that may be used. A further restriction is that a MESSAGE macro may not be issued from CONSAUTH.

ASP DISK INPUT/OUTPUT PROGRAM (ASPIO)

The modules, IONUC , IODATA , IORTNS , and TRACKS are resident in the ASP nucleus and together constitute the ASP Input/Output routines called ASPIO. These routines manage the flow of data between the Support Processor and the direct access devices assigned to the ASP system.

The purpose of ASPIO is to block, deblock and transmit data in the most expeditious manner. This program provides the only available means of accessing the direct access storage devices allocated to ASP on the Support Processor. The program is organized to optimize the transfer of data to and from a uniform pool of direct access storage devices. The data sets managed by the Input/Output program are unique to ASP and cannot be accessed by OS Data Management access methods.

ASPIO has the capabilities to perform the following functions:

- Block data records into a data set
- Deblock data records from a previously blocked data set.
- Provide pool buffering of all output data sets and schedule input/output activity in accordance with the priorities of all active data sets.

- Control buffer allocation.
- Allocate tracks from a pool of direct access storage device tracks.
- Read or write a single buffer of data.
- Provide dedicated buffers for input files as an option.

The devices supported by ASPIO routines are the 2314 or the 3330. The RPS feature of the 3330 is fully supported. The ability to mix 3330 and 2314 queues is not supported. A queue pack may not be varied offline to the ASP system. Additional queue modules may be added on an ASP restart. Any additional modules (DD names) specified on a restart must be of a higher collating sequence than those DD names that already comprise the ASP system.

ASPIO uses the EXCP access method for all I/O transmissions.

The use of dedicated buffers is provided as an option for input files opened by Print Service. This provides a new facility to ensure that once an input file has been opened to be read, three buffers will be used to read all the data records. These buffers will not be available to other active functions until the open input file is closed.

Every ASP DSP communicates with ASPIO by means of a set of macro-instructions. These macro-instructions enable the user to pass to the Input/Output program parameters that identify the data set to be referenced and the action to be performed. The user is subsequently signaled when the requested action has been completed. If the requested action is not performed immediately, the user is automatically placed into an AWAIT status by the Input/Output program until the action can be completed. ASPIO works with two kinds of data sets, single-record and multiple record (multiple file). A single-record data set resides in a single buffer and is written onto the ASP job queue as a physical record. This type of data set is used to record small quantities of data on direct access storage such as a parameter buffer or the Job Description Accounting Block . A six-byte File Description Block (FDB) describes a single-record data set. A multiple record data set resides in one or more buffers on one or more tracks of a direct access storage device. This type of data is used to record large data sets on direct access storage. A 20-byte FDB is required to describe this data set. A File Description Block (FDB) is a work area that is used by ASPIO. Whenever a DSP issues an ASPIO request (macro) it must first set up a pointer to its FDB. The first four bytes of the FDB contain either the buffer address or track address of the file. The next two bytes of the FDB contain flag bits. When a DSP creates a new file the flag bits must be set to zero. From then on ASPIO will maintain the FDB for the file.

Single-Record Data Set

A single-record data set can be read, written, released, or purged. An ASPIO request to write a single-record file will create an entry in a File Directory (FD) if the allocated I/O device is not immediately available. Entries in the FD are ordered by the priorities of the data sets they represent.

When the data set is written, the buffer address in the FDB is changed to a track address and record address. If the data set is being written for the first time, a track is allocated for it. If the data set has been written before, the ASPIO routines extract the address of the track on which the data set resides from the buffer designated in the FDB. In this manner, single-record data sets are automatically

rewritten in place. The DSP must clear the FDB when a data set is being originated.

A request to purge a single-record data set causes the track associated with the data set to be returned to the collection of available tracks. Only single record files that obtained their tracks from the Single Track Table (STT) may be purged. Once a single record data set is read or written, any File Directory entry created is removed. Hence, the Input/Output program is aware of a single-record data set only when a request is made to read, write, or purge such a data set. Once the request is satisfied, the Input/Output program has no further cognizance of the data set.

All single-record data sets that are read must be either rewritten or, if the data they contain has not been changed, released. Releasing a single-record data set restores the FDB for the data set from a buffer address to a track address in the first four bytes. The buffer associated with the file is returned by the ASPIO routines.

The macros associated with single-record data sets, and their functions, are as follows:

MACRO	FUNCTION
AREAD	Reads a single-record data set
AWRITE	Writes a single-record data set
ARELEASE	Releases a single-record data set that has been read
APURGE	Purges a single record file whose track allocation was obtained via the STT, also purges JBTAT's.
AGETBUF	Obtains a buffer from the ASP buffer pool
APUTBUF	Returns a buffer to the ASP buffer pool

The precise definitions of the above macros are found in Appendix A.

Multiple Record Data Set

A multiple record data set contains many logical records, which may, in turn, be organized into logical files. After a data set is opened, data records may be put into the data set (blocking) or extracted from the data set (deblocking). After processing, the data set may be retained (by closing). A request to open a multiple record data set causes the Input/Output program to create an entry in the File Directory and to set the open/closed indicator in the FDB to 0. The I/O required bit is set to 1 whenever transmission is required on an input or output file. The File Directory enables the Input/Output routines to remember which data sets are active and which of these require data transmission. Opening either an input or an output data set causes several pointers, required only by ASPIO routines themselves, to be initialized. These are the current buffer pointer and remaining space left in the buffer.

The closing of a multiple record data set causes the setting of the open/closed indicator in the FDB to 1 and the deletion of the File Directory entry for the data set. When a data set has been closed, the Input/Output program has no knowledge of it.

Data records may be blocked into an output data set (a data set going to direct access storage) or deblocked from an input data set (a data set coming from direct access storage). All blocking and deblocking takes

place in the locate mode; the responsibility for moving data to and from the buffer remains with the user of the program.

Two calls to the Input/Output program are required to block a single data record. The first of these calls (ALOCATE) requests room for n characters, where n may not be larger than the buffer size minus 24. The Input/Output program responds to this call by pointing to a location within a buffer into which the data can be placed. If no room and no buffers are available, the Input/Output program AWAITs until the location is available before returning to the user. The second call (ABLOCK) indicates that the data has been placed in the buffer previously indicated and that m characters ($m \leq n$) should be blocked. The user may locate room for n characters; he may move or read them into a buffer; and he may then block all of them, block some of them, or block none of them.

A single call (ADEBLOCK) is sufficient to deblock a data record. The user indicates a File Description Block, and the Input/Output program returns a pointer to the start of the next record and a count of the number of characters in the record. A request to deblock a record terminates the availability in storage of any previous record. For example, if a user deblocks record 1 and record 2 of a data set, he cannot be certain that the deblocking of record 2 leaves record 1 intact.

ASP Disk Input/Output Macro-Instructions

The macros associated with multiple record data sets and the functions of the macros are as follows:

MACRO	FUNCTION
ABACKR	Backspaces a logical record
ABLOCK	Blocks a logical record into a data set being written
ABLOCKS	Blocks a logical record into a data set being written. However if the record spans across a buffer boundary, the user is required to move the data into two buffers.
ACLOSE	Closes a data set that is in use
ADEBS	Deblocks a logical record and returns two pointers to the data when the record spans a buffer (used by Print Service).
ADEBLOCK	Deblocks a logical record and if the data spans a buffer the data is combined and one pointer is returned.
ALOCATE	Locates space for a logical record in a data set being written
ALOCATES	Locates space for a logical record that spans across a buffer boundary and returns two pointers to the user (used by Main Service)
ANOTE	Notes the current position of a data set so that it may later be repositioned via APOINT
AOPEN	Opens a data set for multiple record input/output
AOPEND	Opens a previously written data set at the end of the data set for subsequent writing

APOINT Repositions a data set that was previously noted (via ANOTE)

AWEOF Records a logical file mark

The detailed definitions of the above macros can be found in Appendix A.

ASPIO Track Allocation

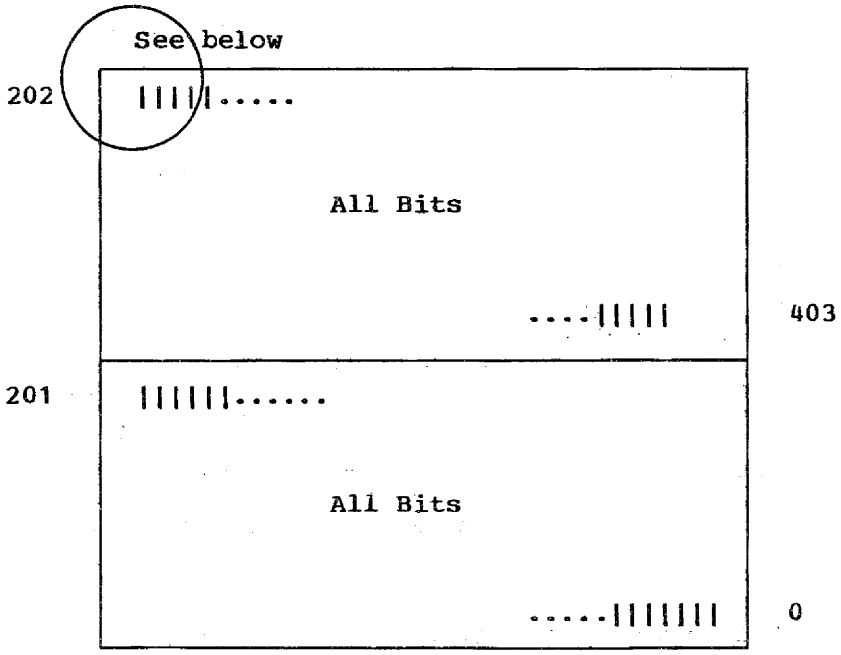
Track allocation is designed to minimize arm motion of the queue devices. The track allocator routine is designed to provide the tracks that are close to the center of a queue device. The data for any particular job in the system is thus spread over all the queue devices in the most general case. Track allocation will be in logical groups of tracks rather than a single-record allocated at each request for a track. These groups are assigned to a job as needed, always with the logical group closest to the center of a device being chosen. So, the particular queue device chosen is the one that has a track group closest to the center of that device.

Note: The size of a logical track group can be either one cylinder or one half cylinder as optioned at initialization in the BUFFER card

During ASP initialization, the module INITIO will construct the track allocator table with each bit representing a logical track group. The new table will be built into two logical parts. The first half of the table contains all cylinders from the center cylinder to the highest cylinder number on the queue device. The second or lower half represents the center cylinder minus 1 to cylinder 0 in descending order.

The logical track group bits are set on with the first bit representing the track group for the first module, and next bit representing the track group for the second module and so forth until the number of modules are reached. The same pattern is repeated until all logical track groups are allocated in the upper and lower parts of the table. This occurs within the extents that are allocated for each module.

Figures 5, 6, and 7 illustrate a created Track Allocation Table (TAT) based on varying initialization conditions. Because the number of bits in the TAT will vary, the logical size of the TAT will also vary. Maximum TAT size is limited to the size of an ASP buffer minus 24 bytes.



Given:

Number of Queue Modules = Two
Queue device type = 3330
Number of cylinders/heads= 404(0-403)/19(0-18)
ASP Initialization Track Group = one half cylinder (first bit represents head 0-9; second bit represents head 10-18)

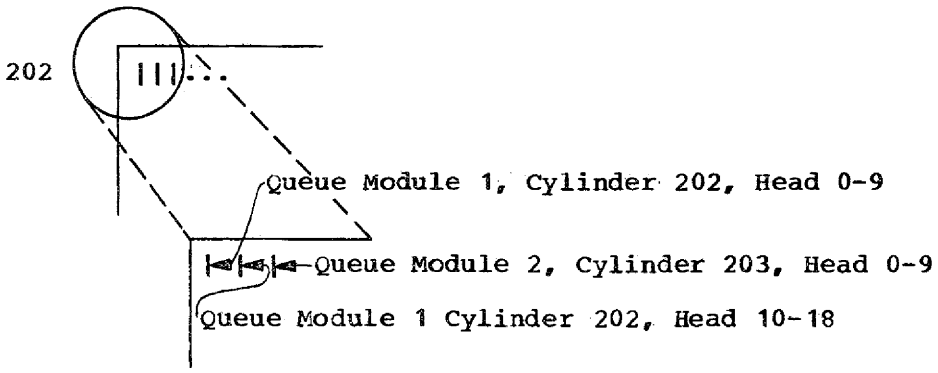
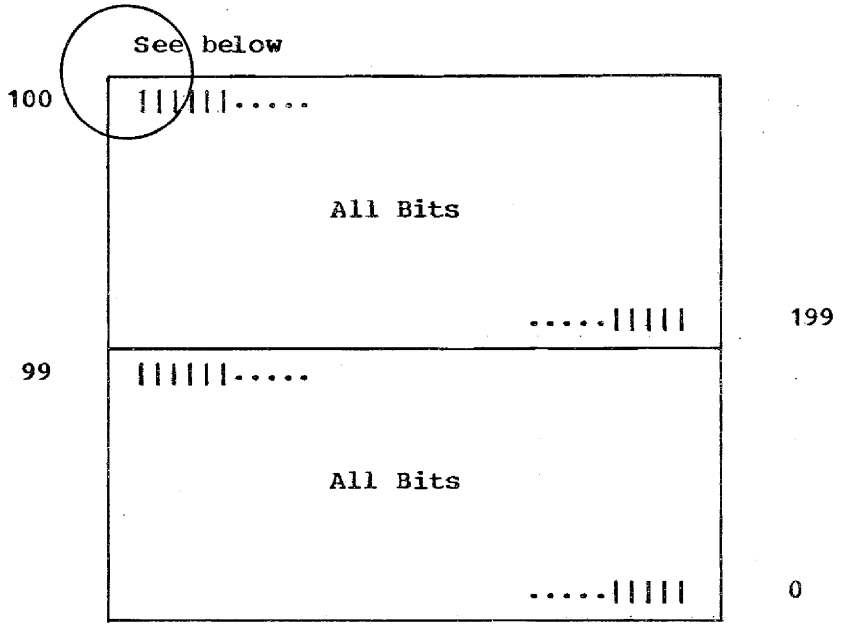


Figure 5. Track Allocator Table - 3330.



Given:

Number of Queue Modules Assigned = Three
Queue device type = 2314
Number of cylinders/heads = 200(0-199)/20(0-19)
ASP Initialization Track group = one cylinder (each bit represents heads 0-19)

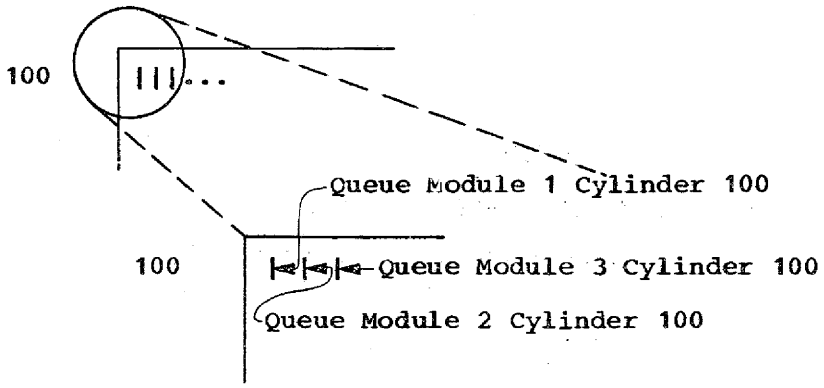
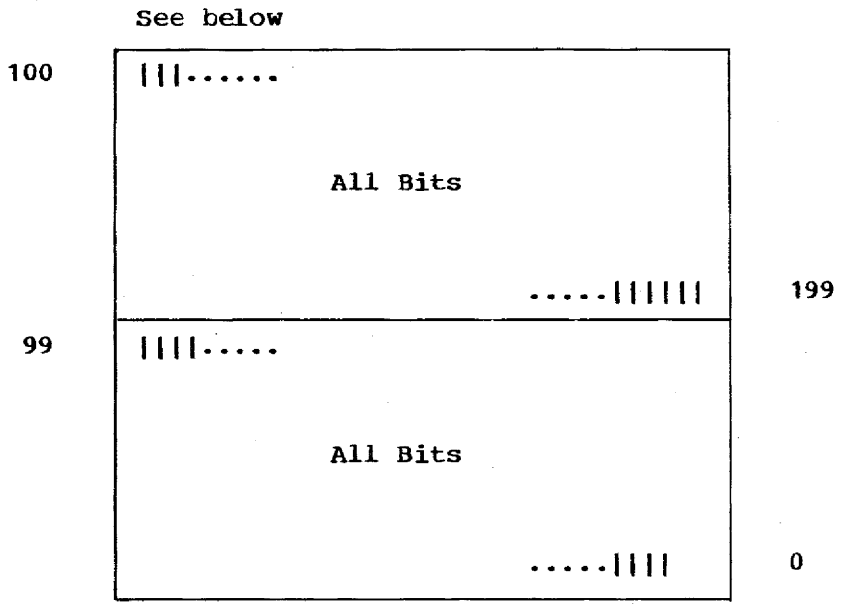


Figure 6. Track Allocator Table - 2314.



Given:

Number of Queue Modules Assigned - Two
Queue device type = 2314
Number of cylinders/heads = 200(0-199)/20(0-19)
ASP Initialization Track Group = one half cylinder (first bit represents heads 0-9, second bit represents heads 10-19)

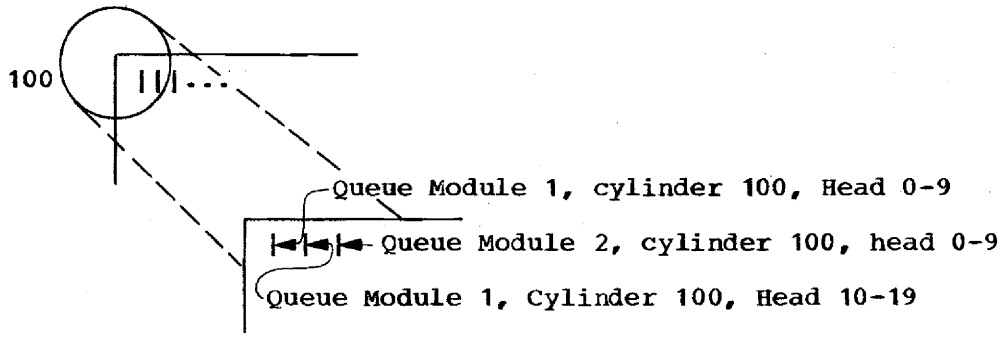


Figure 7. Track Allocation Table - 2314.

After initialization, the track allocator routine, TRACKS, begins allocating logical track groups to each job as required. The first four bytes in the upper and lower half of the TAT are loaded into two registers, R0 and R1. Bits are tested and shifted until the first available bit (or track module) is located. The bit is turned off in the TAT and turned on in the job's track allocation table, JBTAT. The job's JBTAT is created at Input Service time by ASPIO. The FDB for this JBTAT becomes part of that job's JCT.

The JBTAT contains an area which is a "mirror image" table of the TAT. Bits of the JBTAT turned on represent track groups allocated to that job. These same bits are zeroed in the TAT. In addition, the JBTAT contains control information such as the track address of the track group assigned to the job. The first track address of the track group is the actual track address for that job's JBTAT. Each subsequent track used by the job will create a "next available track address" entry

in the JBTAT. Account field is decremented and when zero is reached a new track group must be allocated to the job. The FDB for a job's JBTAT is contained in that job's JCT.

AOPEN , AWRITE , AOPEND , and APOINT are macros available to DSPs for creating files. They all have a TATPTR parameter which may be used to supply the address of the six-byte FDB (newly created files must provide an FDB with all flag bits off). If TATPTR is not specified, ASPIO assumes the pointer AJDTRFDB in the AJDB for the active function.

Note: It is important that any permanent file associated with a job (or DSP) in the system use the JBTAT belonging to the job (or DSP) when the file is created.

When a job has finished all of its processing in the ASP system, all tracks allocated to that job are purged or returned to TAT (the JBTAT is OR'd with the TAT) via the APURGE macro issued by the DSP, Purge.

The Single Track Table

There are many cases where certain DSP's and control blocks of short duration require only two or three single-record files, that is, need only two or three buffers (or sectors) of a track. To eliminate wasted space by allocating each of these files a track group which may be an entire cylinder, a separate JBTAT is created for a table that will allocate space from a track group to such single-record file requests. This table is called the Single Track Table (STT). The six-byte FDB for the STT resides in the TVTABLE and is labeled MNTRKFDB .

The STT is an area of storage defined during initialization. It is 500 bytes if queue devices are 2314s or 1012 bytes if the queue devices are 3330s.

In order to use the STT, the user issues the AWRITE macro with the TATPTR=MNTRKFDB specified. ASPIO will allocate a track group if this is the initial STT request. ASPIO will also create an entry in the STT that will identify the module, beginning head number, and remaining space for additional single records (buffers) that can be contained in the track group. Another field in the STT entry contains bits that represent available record space remaining in the track group. As each additional single-record is written to the file the number field is reduced by one and the corresponding bit for that record is zeroed. When the count reaches zero ASPIO will allocate another track group, update the STT JBTAT, and create a second entry in the STT. The first four bytes of each STT contains a pointer to the next table when the originally allocated space is used up. The four bytes of the first entry will be zero until the second table is created. Figure 8 is an example of a Single Track Table entry.

Single-records purged from the file will cause the corresponding bits and count in the STT to be reversed. This action makes the space just purged available to subsequent single file requests. Track groups allocated to the Single Track Table are not returned to the TAT by space within each group, being of short duration and purged upon completion means that single file requests normally find space in existing track groups and STT track group buildup is minimized.

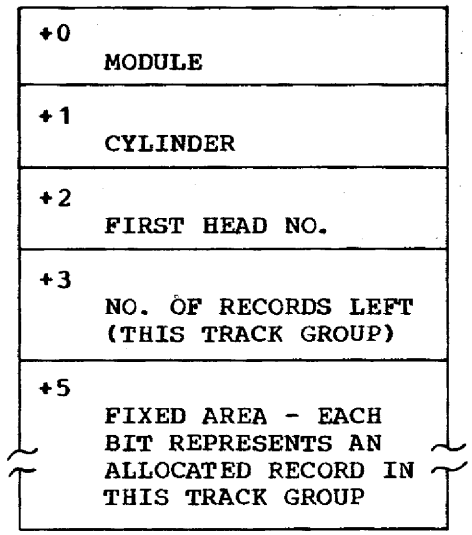


Figure 8. Single Track Table Entry.

Error Recovery

The error recovery routine provided by the OS routines are used to attempt error recovery. If the OS error recovery routines determine that the error is permanent and uncorrectable, ASPIO will then provide its own error recovery procedures. For an uncorrectable write error, where the data is a single-record file and is being written for the first time, a new track will be assigned to write the data. This procedure is used also when writing data in a multiple record data set. A permanent read error will result in a DM760 ABEND.

All accesses to the ASP job queue are done via EXCP by the ASPIO routines. All errors and error recovery are handled by the IOS routines. When the error is determined as permanent, the abnormal end appendage in the module IONUC places the IOB containing the error into an error IOB pool. The module IONUC tests the post of an entry in the error pool. Following the job FCT in RESPARAM is an FCT for ASPIO error recovery called IOERREC . This FCT is chained to the JSS FCT by IONUC and a JSSDR entry point is placed into register 0 and is posted to allow control from the MFM.

The satisfied AWAIT gives JSSDR control and the module IOERREC is ALOADed and control is passed to IOERREC via JSSDR.

The ASPIO error recovery module IOERREC analyzes the I/O error and attempts recovery.

When all errors have been processed by IOERREC, control is returned to JSSDR, who dechains IOERREC FCT and deletes IOERREC.

All write errors are corrected by obtaining a new track and re-issuing the I/O. The bad track is placed into the BADTRACK table.

Read and write errors that are due to overrun and seek checks are reissued until corrected. No upper limit on the number of retries.

Permanent FCT's on RESPARAM FCT chain:

CONSOLES
ASPIO

CALLDRVR
SETUP
WTD
JSS
IOERREC - Chained to JSS only when errors need correcting.

ASP FAILSOFT FACILITY

ASP Failsoft support greatly improves total system reliability by reducing system restart time through Automatic Job Recovery based on installation- and/or programmer-supplied restart parameters. This is accomplished via the ASPCKPT module in the ASP nucleus which checkpoints the necessary tables and parameters onto a direct access storage device. A checkpoint is taken whenever a Job Control Table entry is added or deleted. In addition, Hot Jobs will remain active during an ASP system restart. For ASP-initiated jobs, a checkpoint of ASP data sets is available. In case of system failure, all output up to and including the last complete job step can be recovered, however the job must be restarted from the beginning if restart is specified.

The following table shows the system failure action:

Type of Failure	Hot Jobs on the Local Main	Hot Jobs on the Real Main(s)	ASP Jobs on the Local Main	ASP Jobs on the Real Main(s)
System Failure of Support (Local Main) Processor	Must be Restarted	Remain Active	AJR	AJR
ASP Program Failure on the Support Processor	Remain Active	Remain Active	AJR	AJR
System Failure a Real Main Processor	Remain Active	AJR	Remain Active	AJR

AJR is Automatic Job Recovery. Action taken is dependent upon the FAILURE option specified in the //*MAIN supplied by the programmer, CLASS or STANDARDS cards in the initialization deck.

ASP, itself, is extremely reliable for those installations with a minimum number of user-modifications. Additions of user-modifications and user-written DSPs can impact the reliability to varying degrees. ASP DSP Failsoft support minimizes the impact of ABENDs that occur as a result of DSP program checks.

Functionally, the ASP Failsoft facility can be discussed in three general catagories:

1. ASP program failure on the Support Processor

   a. An ASPABEND dump is taken.

   b. Control is returned to OS. ASP may be restarted.

2. IPL of a main after an ASP RESTART.

   a. All ASP jobs not specified as Hot Jobs that were active on the Main Processors (local and real) at the time of a Support Processor failure are processed according to the FAILURE option. This option can be defined in these ways:

      1) On the //*MAIN control card. This FAILURE option applies only to a single ASP job.

      2) The FAILURE option on the CLASS control card is used to override the FAILURE option on the STANDARDS control card. The CLASS card will affect all jobs in that class and can be overriden by the //*MAIN card for an individual job within a class.

      3) On the STANDARDS control card. This FAILURE option applies to all ASP jobs. It may be overridden by the CLASS control card and //* MAIN card FAILURE options. The FAILURE options,available are:

         a) RESTART the job on Main. This is the default option.

         b) CANCEL the job from Main. PRINT and PUNCH will follow.

         c) Put the job in HOLD for restart on Main.

         d) Print the job and then put the job in the HOLD for restart on Main.

   b. An operator message is issued for each ASP job that was active on Main which indicates the FAILURE action taken.

   c. No special action is taken to support the OS Advanced Checkpoint/Restart facility.

   d. Whenever a Main Processor failure occurs, all active jobs are processed according to their FAILURE options. This processing takes place when the Main Processor is re-IPLed. During Main Processor IPL, all SYSMSG output from incomplete ASP jobs will be recovered.

3. ASP DSP Failsoft

   a. DSP Failsoft isolates a failing DSP and where applicable attempts intelligent recovery of the DSP through system programmer problem correction, operator interaction, or reinitialization of the DSP. This action allows continued ASP operation following abending of a noncritical ASP resource and minimizes the occurrence of situations requiring an ASP system restart.

   b. A failing DSP is intercepted via a STAE or SPIE . The INITIATE routine of ASP contains the SPIE and STAE macros and exit to the ABENDMON module. ABENDMON will give control to AFSDRVR if recovery is possible. AFSDRVR ALOADs appropriate recovery modules. Recovery modules for user-written DSP's may be provided by the user.

c. Operator intervention is required to continue. AFSINIT issues operator messages identifying the failure. The operator has the following options:

   1) WAIT - Message AFS03 is issued every five minutes requesting action unless the WAIT reply is given. The WAIT reply indicates the failure was noted and appropriate action is being taken.

   2) CANCEL - This message requests DSP Failsoft to attempt to return the failing DSP's resources and place the DSP in a permanent AWAIT.

   3) BYPASS - This response will cause the user-written recovery routine to be bypassed and prepares for the diagnosing of the problem using the Dump Core utility DSP.

   4) RETRY - This response causes DSP Failsoft to schedule the user-written retry module.

d. If recovery isn't possible or fails, DSP Failsoft will quiesce the failing DSP. This results in the return of system resources (JCT Queue, RESQUEUE, etc.), the return of all units on the FCT GETUNIT list and placing of the DSP in a permanent AWAIT.

MAIN DEVICE SCHEDULER (MDS)

The Main Device Scheduler (MDS) controls the allocation, mounting, and deallocation, also referred to as setup, of I/O devices associated with job execution on a Main Processor. MDS runs under its own FCT servicing jobs in RESQUEUE that require setup. MDS services are included as part of the Main scheduler element in normal job processing. A job is selected from RESQUEUE for a Main Processor, based on scheduling constraints specified in the SELECT initialization card, and device allocation is attempted. Following allocation of devices to a job, mount messages are directed to the operator indicating the volume(s) to mount on specified device(s). The operator's actions are monitored and a VERIFY is done each time an MDS managed device becomes ready. This VERIFY indicates the volume serial, label status, and other pertinent information to MDS. When all devices have been properly mounted, the job becomes a candidate for Main Processor execution. Following its execution the job is again processed by MDS to return devices and/or volumes to the available status.

The devices to be managed by MDS are either tape or direct access and are defined to the system by means of DEVICE cards in the initialization deck. A volume on a managed device is either considered permanently resident or removable. A volume is considered permanently resident if indicated by the UCB status at the time the processor is IPLed or the operator may request that a volume be "mounted" until he "unloads" it. MDS will recognize requests for volumes that are permanently resident and while no mount will be required, the job will be routed to the Main Processor on which that volume is mounted.

MDS is posted by JSS when jobs requiring setup are added to RESQUEUE . A job's setup requirements are indicated in the Job Setup Table (JST) created by the ASP R/I DSP. Setup allocation makes use of the System Units (SYSUNITS), Setup Units (SETUNITS), and Setup Names (SETNAMES) tables. The SYSUNITS table contains the current allocation status of devices, volume serial numbers of the last volume mounted, and the Main Processor(s) to which the device is attached. The SETUNITS table for each Main Processor

has an entry for each managed device attached to that Main and contains the RESQUEUE address of the first job to which the device is assigned and the console destination to which messages pertaining to this device should be directed. The SETNAMES table correlates the device types specified in the JCL UNIT parameter and the devices satisfying that designation. The SETNAMES entries must therefore include all OS SYSGENed generic names referencing MDS managed devices.

Every device in the total system has a unique SYSUNITS table entry in which allocation status is kept. Therefore devices may be shared between two or more Main Processors or between the Support Processor and one or more Main Processors. The allocation algorithm considers volume serial numbers as well as device type when allocating. A Volume Unavailable Table (VUT) , maintained by the operator, is searched and if a job requires a volume that is in the table its setup is delayed until that volume is available. If the SYSUNITS entry shows a volume mounted but not in use that is required by a job in setup, MDS will utilize the volume if possible on the device where it is presently mounted. If the volume is direct access and referenced with a SHR disposition, more than one job specifying SHR may be allocated to that volume. Units are selected for mounting with a consideration of dismounting requirements.

When allocation is successful, the volume mount messages will be directed to the console class destination indicated in the DEVICE card for the unit selected. Messages encountered in execution of the job, such as OS RETAIN, MOUNT and KEEP will also be directed to the destination requested for that device. The serial number information in these messages will be used to update the SYSUNITS volume serial to indicate the change in volume to MDS. Any errors detected by MDS in setup processing will result in a diagnostic in the SYSMSG data set for the job and the canceling with print of the job.

MDS is divided into one resident, five transient, and one data CSECT modules. The resident module (MDSDRIVR) contains the wait-post logic for MDS processing as well as the message processing analysis.
MDSALLOC provides device allocation and the issuance of mount commands to the operator. MDSVERFY validates the operator mounting and writes checkpoint information for use in an ASP restart.
MDSBRKDN unallocates devices used by a job and completes setup processing for the job. MODMDS processes the operator MODIFY commands relating to MDS. MDSREST ensures the same jobs will be in setup following a system failure.

DYNAMIC SUPPORT PROGRAMS (DSP's) - BASIC

The Dynamic Support Programs (DSP's) perform the support functions of the ASP system. These programs, which are stored on the ASP JOBLIB , operate as transient subroutines of the ASP Supervisor.

Each DSP has an entry in the Dynamic Support Program Dictionary. This entry contains the program name, internally assigned program number, control section name if the DSP is reentrant, priority, sequence-dependent operation indicator, reentrant or reusable indicator, callable or not callable indicator, and a pointer to the Device Requirements Table , which defines the device types required by this program.

As a job is introduced into the ASP system, an entry is created for it in the Job Control Table (JCT). Within the JCT entry, a Scheduler Element is created for each job segment. This Scheduler Element contains, among other things, a DSP number. The program indicated by this number is brought into Support Processor storage from the JOBLIB when the job segment is initiated by the Job Segment Scheduler (JSS).

When a job segment is scheduled, the DSP and the associated data sets become known to the ASP system as a function. The DSP must be written to operate in the environment dictated by the Multifunction Monitor.

The basic ASP DSP's are:

- CR, TR and DR. Spools the input stream on the Support Processor; passes control to Input Service which recognizes control cards and takes the appropriate action; and creates a Job Control Table entry, Job Data Sets Block, Job Description and Accounting Block for each job.

- Reader/Interpreter. Interprets the OS job control language to determine system resources required by a job before being sent to a Main Processor.

- Main Service. Manages the flow of data (for example, system input, system print, and system punch data sets) via the Channel-to-Channel Adapter to and from the Main Processor.

- Print Service. Prints data sets.

- Punch Service. Punches data sets.

- Purge. Removes the job from the system, returning to the system all direct access storage device tracks allocated to this job.

Other DSP's have been implemented to support special operator functions, such as background utility functions. The maximum number of DSPs that may exist under control of the Multifunction Monitor is 256.

DSP's are multiprogrammed components operating in a time-shared (non-time-sliced) environment. Since many DSPs may be active simultaneously, and all are competing for available CPU time and storage, it is important that they be written to permit the sharing of these resources. Most DSP's achieve this sharing naturally, since they release control at many logical points, consequently permitting other DSPs to execute. DSP's without logical points at which they may release control should be assigned very low priorities and should be programmed to release control artificially (by issuing an AWAIT to a satisfied event) to permit higher priority DSP's to be executed.

Normally, DSP's are transient programs loaded via ALOAD The main program (or driver) for a DSP may optionally be made resident by

including it in the ASP RESIDENT card . DSP's communicate with the ASP resident programs by means of the Transfer Vector Table (TVTABLE). This table is a series of entries that define the location (or entry points) of all resident programs and tables of the ASP system. The TVTABLE is established by means of a macro that appears as a dummy section (DSECT) in all DSP's and as a control section (CSECT) in the TVTABLE program in the ASP Nucleus.

As stated earlier, DSP's are scheduled for execution and, if necessary, loaded by the ASP Job Segment Scheduler. Entry is made via a direct branch to the entry point of the DSP, with the return point in register 14. When a DSP terminates execution, it should return to the Job Segment Scheduler via register 14, with the appropriate Job Segment Scheduler completion code in register 15. For an additional discussion on DSP action after it has been scheduled refer to Chapter 7, Writing Dynamic Support Programs.

DSP's may be defined as either reentrant or reusable. Reusable DSP's are, in general, restricted to a single copy active at a time. However, single-module reusable programs may be allowed to have more than one copy active concurrently through the use of the MLOAD option of the DSPDC macro -instruction. For example, multiple copies of Card-to-Printer may execute concurrently if the required devices are available. Reentrant DSP's may actively process several jobs simultaneously, provided that sufficient devices are available. Each reentrant DSP must have a data control section, a copy of which will be loaded for each function for which the DSP is scheduled. All modules of a reentrant DSP must be reentrant and must use the loaded data control section. Input Service, Punch Service, and Print Service are reentrant DSP's, as are Main Service and the background utility Tape-to-Printer program. A functional description of the primary DSP's in the ASP system follows.

INPUT SERVICE (CR, TR, and DR)

Input Service is the support function that accepts and queues all jobs entering the ASP system.

Input Service operates on the split-spool concept. (Split-spooling is closely related to the OS Automatic SYSIN Batching (ASB) Readers.) Cards are read from an input device, blocked, and placed on an intermediate DASD storage device; when physical input has finished, the accumulated card images are read from DASD and processed at high speed. The main storage requirements are low during physical input, which takes a relatively long time. Interpretation and analysis of cards has a greater main storage requirement, but is accomplished in a relatively short time. Utilization of resources is improved through the time-separation of the reading and interpreting functions. Input Service consists of two phases, each consisting of several nonresident modules that process the incoming jobs.

The first function is the Reader. The Reader transfers the job's card images from an external device to an ASP DASD device. A Reader device may be an IBM card reader, a tape unit, or a DASD device supported by OS BPAM . Jobs being submitted from RJP terminals , either programmable or nonprogrammable, are processed by Input Service as though they were coming from a local card reader.

An Input Service Reader will transfer a predetermined number of jobs to the "spool DASD". This becomes a job "batch". As each job batch is read in, Input Service proceeds to the second phase, the ASP control card processing phase which analyzes the ASP control cards and writes the jobs to the ASP queue, along with the ASP control blocks necessary to process the job.

Input Service Reader Function

The Reader function is comprised of six nonresident modules. Three of these modules are callable DSP's, invoked via the operator CALL command.

The Input Service callable DSPs are directly related to the input medium through which jobs are to be introduced. They are:

- Card Reader (CR)
- Tape Reader (TR)
- Disk Reader (DR)

These DSP's when invoked (called) act as the drivers for the other three modules of the Reader function. CR and TR use OS EXCP level programming via the ASPEXCP macro -instruction. DR uses standard OS BPAM . Device errors and error messages are also handled by these three DSP's.

A Reader function module, RDLOGIC , deblocks any blocked input records, examines the 'card' as to type, that is, JOB, DD, etc., and transfers the cards as a multirecord file (MRF) to an ASP DASD via ASPIO. The incoming cards are examined at this time only for the purpose of separating OS jobs, and of properly grouping data sets, that is, scanning for a data delimiter. A separate MRF is written for each OS job. Each MRF is represented by an entry in an ASP Job Data Sets (JDS) table. The JDS entry contains the MRF FDB for each job written to DASD via a Reader. A JDS table is created for each job 'batch' and is passed to the second phase of Input Service, the control card processing phase. The JDS table is placed on disk and located via an FDB in the Job Description and Accounting Block (JDAB) created by RDLOGIC for the interpreter function. RDLOGIC also creates the JCT entry for the interpreter of Input Service.

Before RDLOGIC gains control, RDINISH , another module of the Reader function of Input Service, checks disk resident control blocks to see if the reader was called as the result of an ASP restart or if this is an initial call, sets certain standard (default) operating parameters, acquires main storage for an input area, and allocates the input devices for CR or TR. For DR, RDINISH controls the FIND for the designated PDS member.

Whenever the operator includes operands with a command to a reader, RDOPRMS is loaded into main storage to analyze the message.

The six modules of the Reader function, CR, TR, DR, RDLOGIC, RDINISH, and RDOPRMS comprise the reading portion of Input Service. They are all fully reentrant; there need be only one copy in storage to handle multiple/simultaneous readers.

The Readers, specifically RDLOGIC, recognize the DD * or DD DATA[,DLM=xx] and stops scanning for a JOB card until the /* or specified delimiter is read. RDLOGIC also examines the input stream for the ASP //* DATASET card . If one is found, the input stream is scanned for the //* ENDDATASET card (and for a JOB card if J=NO; if J=YES an ENDDATASET card is the only recognized termination).

CR normally uses a chain of five CCWs to read in a "block" of 400 bytes. However, an *X CR,C causes the reader to be set up for an unchained CCW. This is a prerequisite for any job stream that includes card-image (column binary) input since the location of data mode 2 cards is unpredictable. When a DATASET card with the MODE=C parameter is encountered, the CCW for that reader is modified for card image (mode 2) reading. The CR module then scans for the ENDDATASET card, which is the

only acceptable terminator; when it is found, the CCW is reconverted for normal EBCDIC reading. All card-image input to ASP-spooled Main Processor jobs must be delimited by the //*DATASET MODE=C - //*ENDDATASET cards.

TR accepts labeled and unlabeled tapes containing blocked or unblocked card images.

DR uses BPAM to read one block at a time from an OS partitioned data set. Deblocking is done by ASP.

The nature and extent of error recovery depend upon the device being used. CR offers the greatest opportunity for operator-initiated correction; TR uses OS error recovery; and DR uses a SYNAD exit to intercept permanent errors that would otherwise terminate ASP. When a device is not ready, reading will restart automatically when it is made ready.

DR is a high-performance reader intended for low-volatility (infrequently changed) job streams. The DCB for the PDS is located in the RESPARAM module in the ASP nucleus and is OS OPENed at ASP initialization time by the INITGEN module. Because there is only one copy of the OS control blocks available, only one DR is permitted in the system.

Input Service ASP Control Card Processor

After the Reader function of Input Service has placed the input jobs in the ASP queue in uninterpreted form the ASP control card processing phase must now interpret and place them in the ASP queue as ASP jobs along with the control blocks necessary to process the jobs. As described previously the Reader function creates the JCT , JDAB , and JDS for this ASP 'job'.

The ASP control card processing phase consists of several modules where ISDRVR is the name of the DSP, and the name of the module first invoked by JSS .

The control card processing function, through appropriate modules, reads the input JDS created for each 'batch' by the Reader. For each job, it creates a Job Control Table (JCT), a Job Data Set (JDS) and a Job Description and Accounting Block (JDAB).

The control card processing phase analyzes the //* ASP control cards. Each ASP control card is processed by its own module.

The absence of //* PROCESS cards in a job's JCL indicates a standard OS job and this job will be scheduled by ASP in a Standard Scheduler Element (SE) sequence. The sequence of SE's in the created JCT for each job is ordered as follows: ASP Reader/Interpreter, Main Service, Print, Punch, and Purge.

The //*PROCESS cards are used in the job's JCL to alter the ordering of ASP scheduler elements. Created JCT SEs reflect the order of //*PROCESS cards.

If a //* FORMAT card is included in the job's JCL the Translator function of Input Service will create a Format Parameter (FRP) table for that job.

READER/INTERPRETER SERVICE (ASP R/I)

After Input Service has processed a job the next function scheduled is the Reader/Interpreter (ASP R/I) DSP. ASP R/I working in conjunction with the OS Reader/Interpreter, resident in the Support Processor, performs Reader/Interpreter functions for all Main Processors. During execution of the OS Reader/Interpreter several exit routines are used as an interface between the ASP R/I DSP and the unmodified OS R/I. These exit routines are:

- Access Method Exit - Passes a JCL statement from the ASP queue (JCLIN) to the OS R/I
- Accounting Routine Exit - Converts JCL for use by ASP, that is, SYSOUT=X changed to UNIT=(CTC,,DEFER). This change does not appear in the SYSMSG data set.
- Find Exit - Locates a procedure within SYS1.PROCLIB on the Support Processor when referenced by an EXEC statement
- Queue Manager Exit - Simulates the OS Job Queue Manager by reading OS control blocks from, or writing them to, the ASP queue. "Enqueuing" a job is simulated.
- Return Exit - Intercepts an OS return as a result of an unusual OS Reader/Interpreter event.

Figures 9 through 13 show the relationship in the Support Processor between ASP R/I DSP, ASP Exit routines, and the unmodified OS R/I.

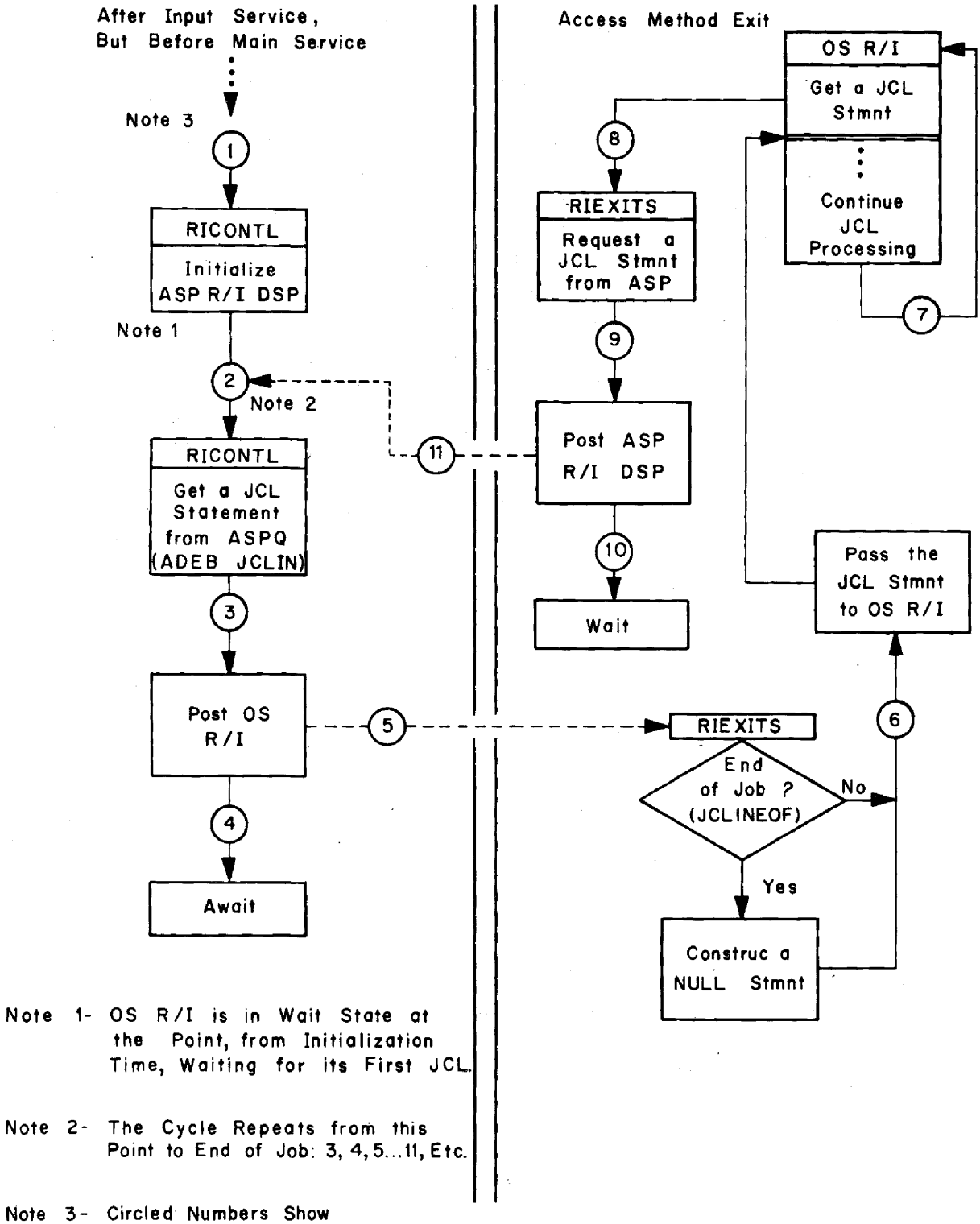
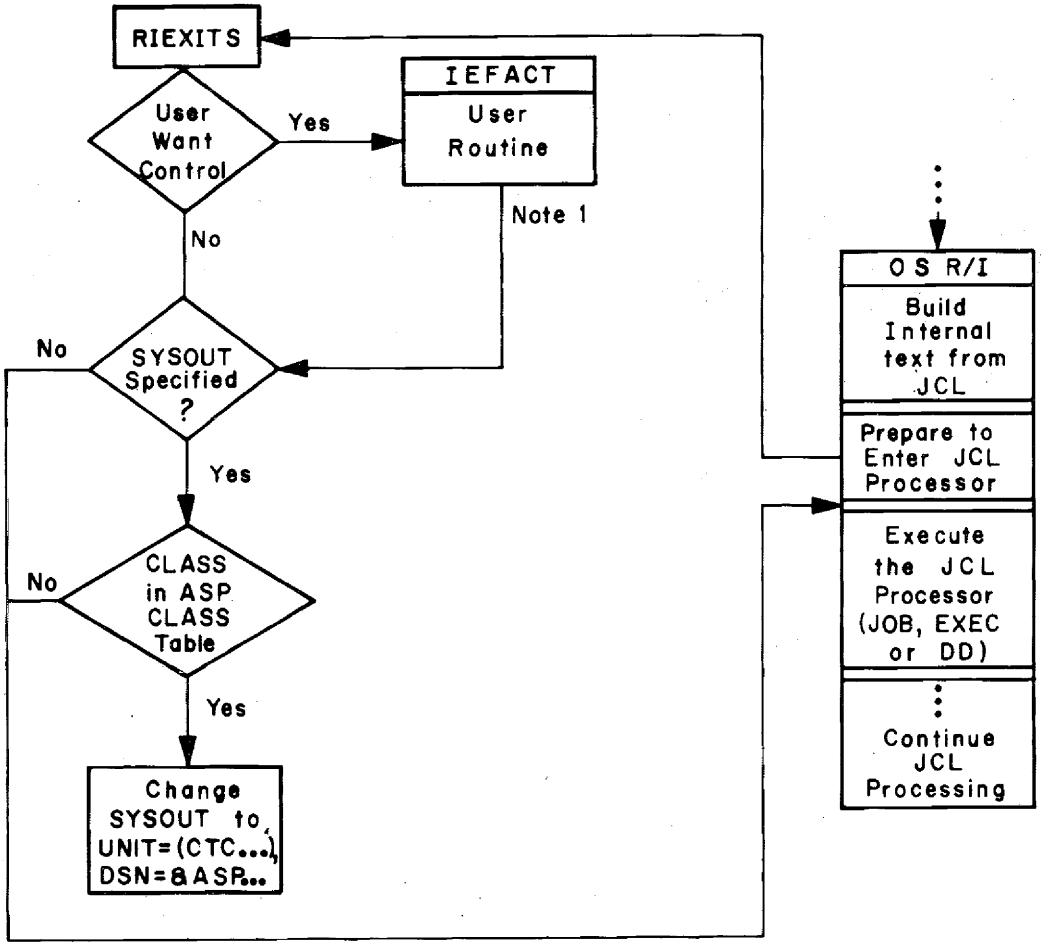


Figure 9. Access Method Exit.



Note 1 : The USER Routine, Named IEFACT, is Linkedited With RIEXITS in Order to Become Part of the System.

Figure 10. Accounting Routine Exit.

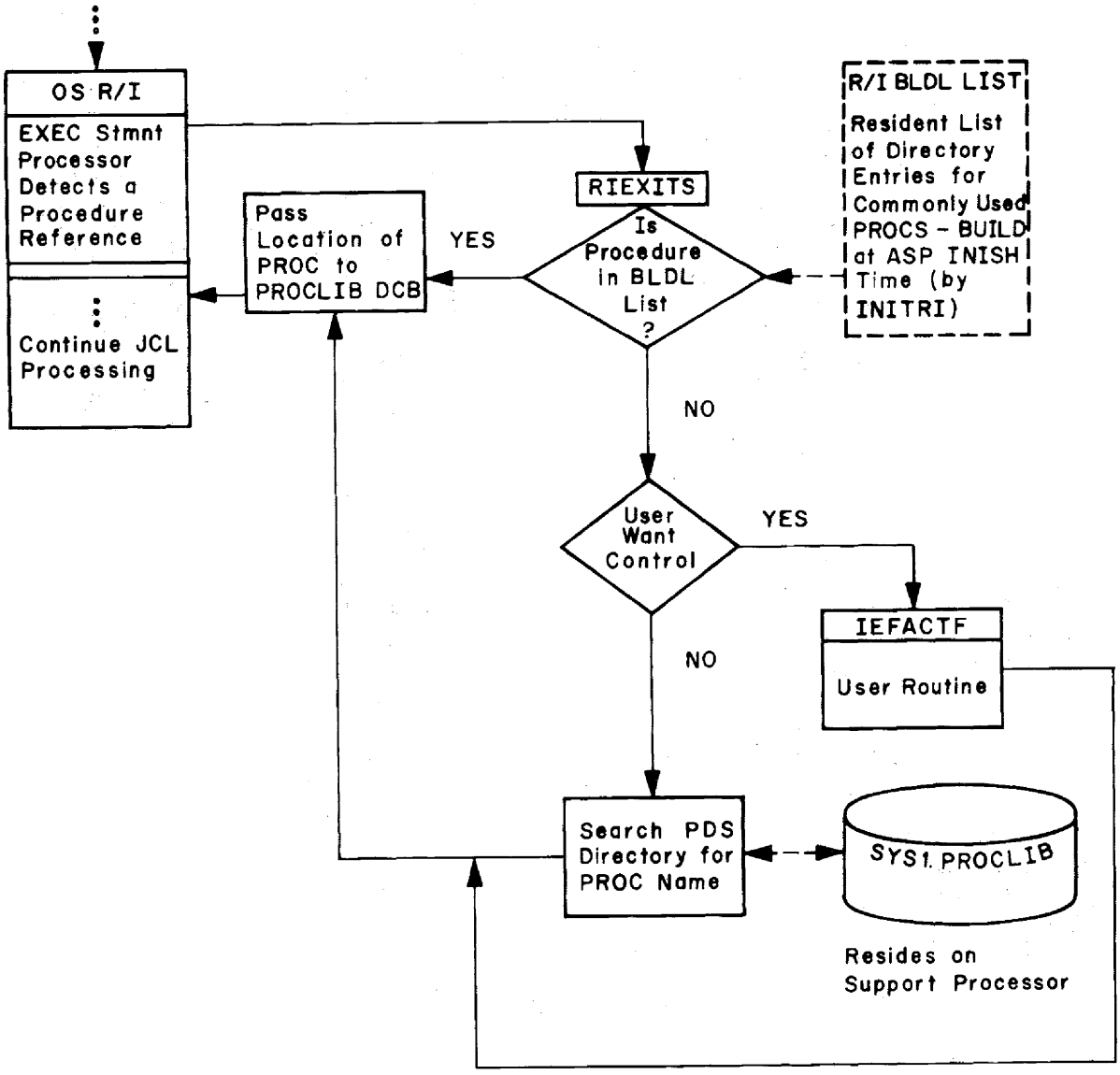
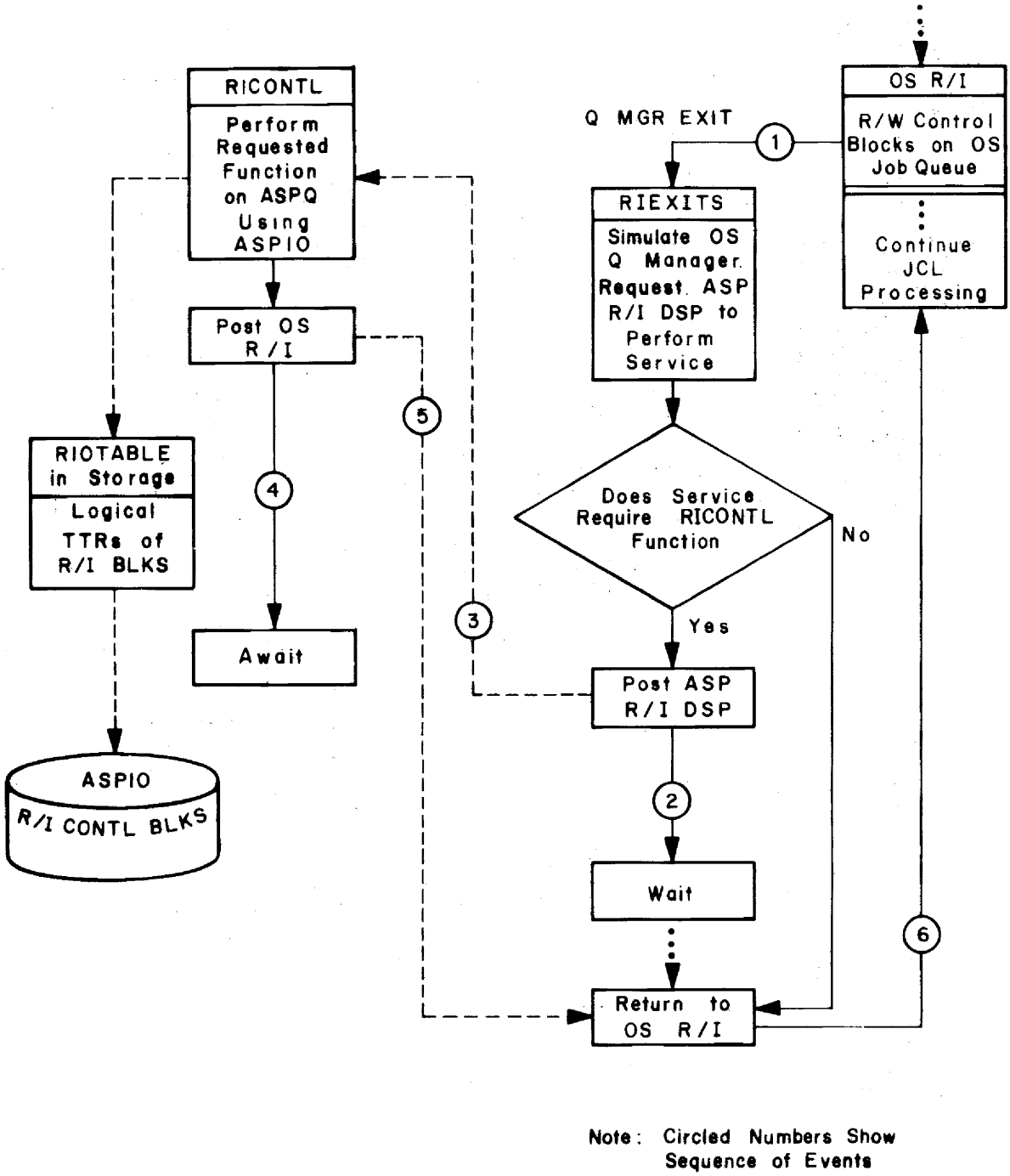


Figure 11. Find Exit.



Note: Circled Numbers Show Sequence of Events

Figure 12. Queue Manager Exit.

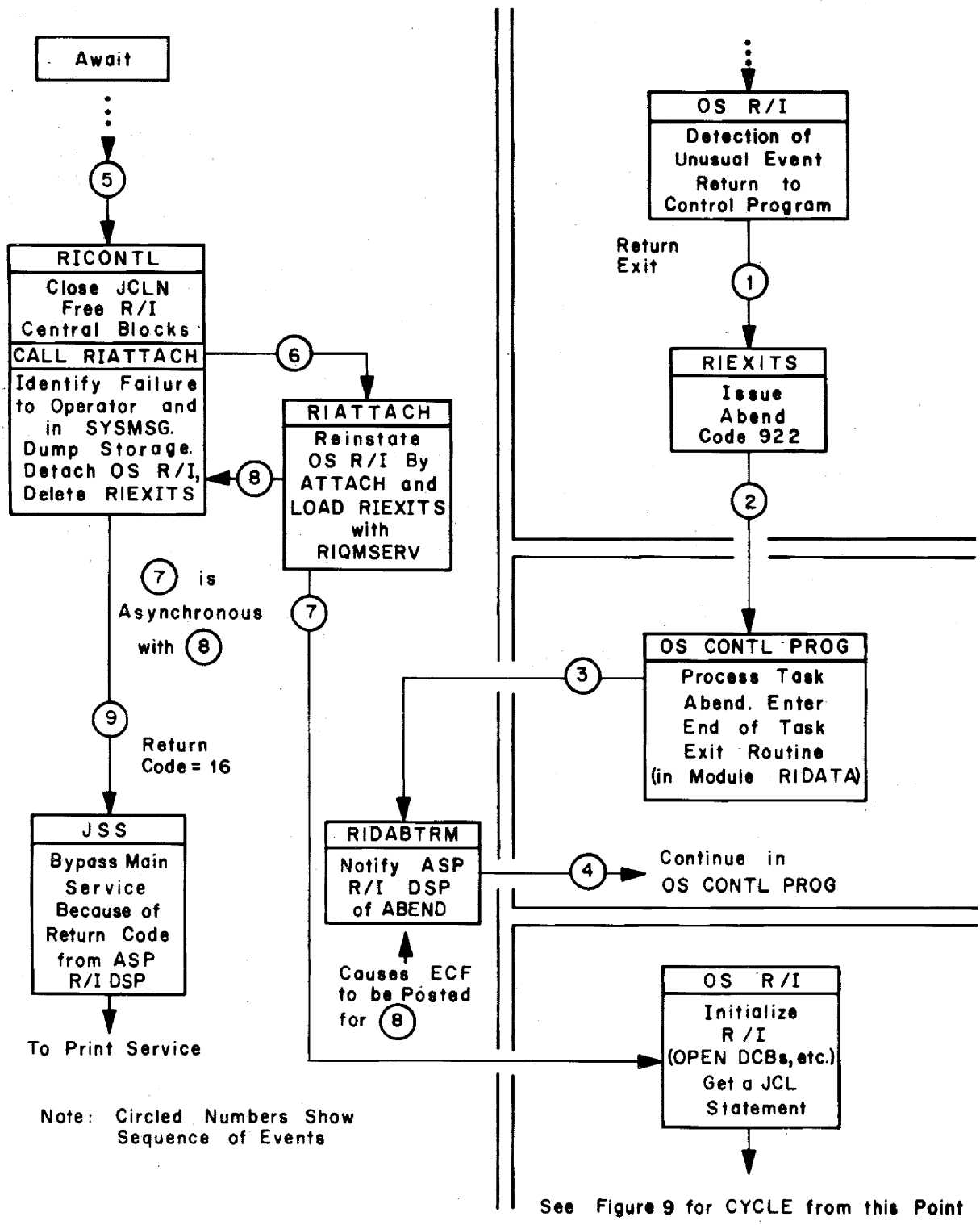


Figure 13. Return Exit.

ASP R/I performs several functions for the exit routines, such as AGETBUF, ADEBLOCK, etc. Besides performing functions for the OS R/I exits, the ASP R/I performs the following functions:

- General housekeeping that allows ASP to control job flow through the OS R/I
- Flush a job with JCL errors prior to entering Main Service in ASP
- Automatic determination of job setup requirements based on interpreted JCL
- Issue fetch messages for necessary volumes
- Handle OS messages (in SMB's) so they can be printed by ASP when an error condition exists
- Issue error messages for unexpected conditions detected by ASP R/I
- Allows operator to cancel a job being processed by ASP R/I

ASP Reader/Interpreter (R/I) Job Flow

After a job has been processed by ASP R/I and OS R/I, that is, entirely interpreted and "enqueued" on the ASP queue, the OS R/I goes into the "wait" state until the next job enters the ASP R/I. If the job had any JCL errors, ASP control statement errors (detected by Input Service), or unusual conditions such as using an ASP-reserved ddname, the job is flushed and its interpreted JCL is written along with an appropriate error message to the ASP SYSMSG print data set. If there were no errors, processing of the job continues with construction of a Job Setup Table (JST).

Job Setup processing is based upon information from each DD statement of the job, stored in OS control blocks (SIOT and JFCB). If a DD statement specified a tape or direct access device, that was specified in the ASP SETNAMES table, it is eligible for setup. Volume information is also extracted from the DD statements. When volume information is absent, it is obtained from the catalog of a Main Processor; the selected Main is either from a //* MAIN statement submitted with the job or to the first Main defined at ASP initialization time (MAINPROC statement). When connecting volumes (CVOL) are used, they must be mounted prior to being referenced. Generic names used (UNIT=2400) must have also been defined to ASP through a SETNAME statement. Demand allocation addresses (UNIT=181) must be known to ASP from the MAIN parameter of a DEVICE statement.

If any EXEC step of the job specifies a program name of JCLTEST, the ASP R/I will flush the entire job with its interpreted JCL written to the ASP SYSMSG print data set accompanied by a JCLTEST message. This facility allows a programmer to test only JCL, without using a Main Processor or any devices eligible for setup.

If any EXEC step of the job specifies a program name of JSTTEST, the ASP R/I will calculate the setup requirements and will print the results to SYSMSG. After this is performed, the job will be flushed with the interpreted JCL written also to SYSMSG.

Fetch messages are not issued for volumes noted as permanently mounted. It should be noted that fetch messages for volumes listed in the Volume Unavailable Table (VUT) will be issued in a different format than normal.

Finally the ASP R/I finishes processing the job by issuing fetch messages to a tape-fetch console or disk-fetch console (if any specified on the R/I Initialization control card). The message(s) lists any volume(s) needed by the job. Fetch messages are not issued for volumes listed in the Volume Unavailable Table (VUT) , or for volumes noted as permanently resident.

The ASP R/I exits to the Job Segment Scheduler (JSS) with a return code indicating a job either is to continue ASP processing, be canceled with Print Service only, or to be rescheduled for another pass through RICONTL because no main was available when attempting to make a catalog reference.

If a Main Processor is not available for catalog references during ASP R/I processing, the job will be rescheduled for processing within ASP R/I again (when a Main becomes available).

If the OS R/I abnormally terminates, it will be reinstated again automatically as a subtask along with a message to the operator. If it abnormally terminates twice in succession, it is marked offline to the system.

Post ASP Reader/Interpreter (R/I) Processing

Control blocks created by the OS R/I must be updated at Main Service time to reflect actual TTR assignments in the now selected Main Processors SYS1.SYSJOBQE data set. The Support Processor and Main Processor are in communication during this process until all control blocks have been "mapped".

The selected Main Processor obtains its input job stream in the form of queue record control blocks stored in ASP Q (JCBIN). The queue reader on Main places the control blocks into locations of the OS job queue assigned during the previous mapping phase, and then enqueues the job on the appropriate input queue to inform an initiator that work is waiting.

Input Requirements

ASP R/I expects a JDS table with entries for JCLIN , SYSMSG , and JCBTAB . The JCLIN data set in ASPQ is the primary input and consists of user-submitted JCL and ASP control statements. Other control blocks and tables used as input to ASP R/I are:

- TVT
- FCT
- AJDB
- JDAB
- JST

Only a table called the R/I I/O Table (RIOT) is created by ASP R/I for use outside the DSP. It contains information necessary to retrieve any OS R/I control block being maintained in the ASPQ. It is maintained in ASPQ (JCBTAB).

The JST is modified when job step entries and device entries are created after scanning the JCL. Other tables are used solely for internal DSP use.

MAIN SERVICE

The Main Service support function controls job processing on the Main Processor(s). The Main Service DSP interfaces directly with each Main Processor via the Channel-to-Channel Adapter (CTC). This program performs the following functions:

- Establishes the system initialization of OS at IPL time
- Controls the loading of the appropriate control program
- Initiates Main Processor execution
- Provides input/output support for data sets transmitted across the CTC or within the Support Processor
- Monitors job execution by analysis of Main Processor console messages and performance statistics (line and card output volume) to detect abnormal performance and to take a user-specified action when difficulty is detected
- Provides operator communication to, and control of, the Main Processor

To provide these functions, the Main Processor system is generated with the CTC assigned as the operator console. Consequently, all operator messages are transmitted across the CTC, thereby providing Main Service with all of the operational controls that the operator has. Main Service scans incoming messages and generates the appropriate messages to initiate and terminate job execution. The Main Processor initiates all data transmissions to and from the Support Processor.

The Input/Output Supervisor (IOS) of the Main Processor provides the necessary interface between the user and the CTC. The adapter appears to the Main Processor programmer as a series of nine-track tape units, with data sets assigned to separate units. All execute channel program (EXCP) calls for these units are intercepted by ASP modifications to IOS and are directed toward the channel-to-channel adapter. When ASP is running in the local mode, the CTC is simulated. If a transmission request is for a data set other than the previous data set, IOS prefixes the transmission with a Data Set Title Label (DSTL). On the other hand, if a transmission is for the same data set as was the previous transmission, no DSTL is sent. DSTL's are indicated to Main Service by special WRITE commands for ease in distinguishing between the DSTL and data.

To achieve maximum performance with the ASP system, Main Service handles CTC Input/Output calls directly rather than via interface with IOS. Main Service interprets input/output requests from the Main Processor and responds with the appropriate read or write for the requested data set. If the data set does not have a JDS entry and the request is for a write, Main Service creates a data set and processes the request. If the request is a read for a nonexistent data set (no JDS entry), or if, in reading, the Main Processor tries to go beyond the end of a data set, Main Service automatically terminates processing for the job, with an appropriate diagnostic message and the job is canceled.

When ASP is running in the local mode, IOS notifies the Main Service interrupt handler that this operation is to be simulated. At this point, IOS has been modified to bypass the actual SIO and proceeds as if the operation had been started normally. After Main Service has simulated the operation of data movement, it then simulates input/output interrupts.

Programmer estimates for lines and cards of output are compared to actual values in order to detect abnormal performance of a job. Actual lines and cards are tabulated for system output and system punch data. If either of these estimates is exceeded, Main Service takes the action dictated by the user; that is, it issues a warning to the operator, cancels the job, or cancels the job with dump.

Main Service is comprised of both resident and nonresident modules. Those made resident during initialization are: MAIN and MSVMVT . A real or local Main uses MAINIO and MPCDATA. A dummy Main (used with POLYASP) uses MSVDUMMY and MPDDATA. A local Main (combined Support/Main Processor) uses MSVLOCAL for CTC simulation. Other modules are ALOADED during Main Service execution.

Generalized Main Scheduling

During ASP initialization, four initialization control cards are used to establish the parameters used by Main Service. These are the MAINPROC, SELECT, CLASS, and GROUP cards. Initialization creates an MPCDATA CSECT for each Main. SELECT card parameters are stored in the MPCDATA area for each Main. INITMN1 and INITMN2 are the initialization routines that process these control cards.

Job class and group control cards are read into permanent class and group tables. The size of these tables is dependent upon the number of control cards and class constraints defined. The tables are defined in the MPCLSTAB and MPGRPTAB macros. There must be a group entry for every group name defined on a CLASS control card. For every CLASS MLIMIT or TLIMIT constraint the corresponding Main Processor or job class must be defined. Figure 15 shows the initialization cards and the control block chaining for the MAINPROC, SELECT, CLASS, and GROUP cards.

Figure 14 illustrates the scheduling algorithms of just one job, JOBA. This example has been simplified for illustrative purposes and assumes selection mode SHIFT1 is the only currently active scheduling mode. Note that JOBA must fit a SELECT, MAINPROC, GROUP and CLASS combination in order to qualify for scheduling on a Main Processor. The specification of SY2 on the //*MAIN card limits scheduling to SY2 even though the selection mode SHIFT1 is also active on SY1. JOBA's specifications of CLASS=A causes GROUP selection of TSO12. The example shows three selection modes; SHIFT1, SHIFT2, and SHIFT3 defined during initialization. Reinitialization is not necessary to alter the selection mode on a Main as it can be done dynamically via the operator *MODIFY command.

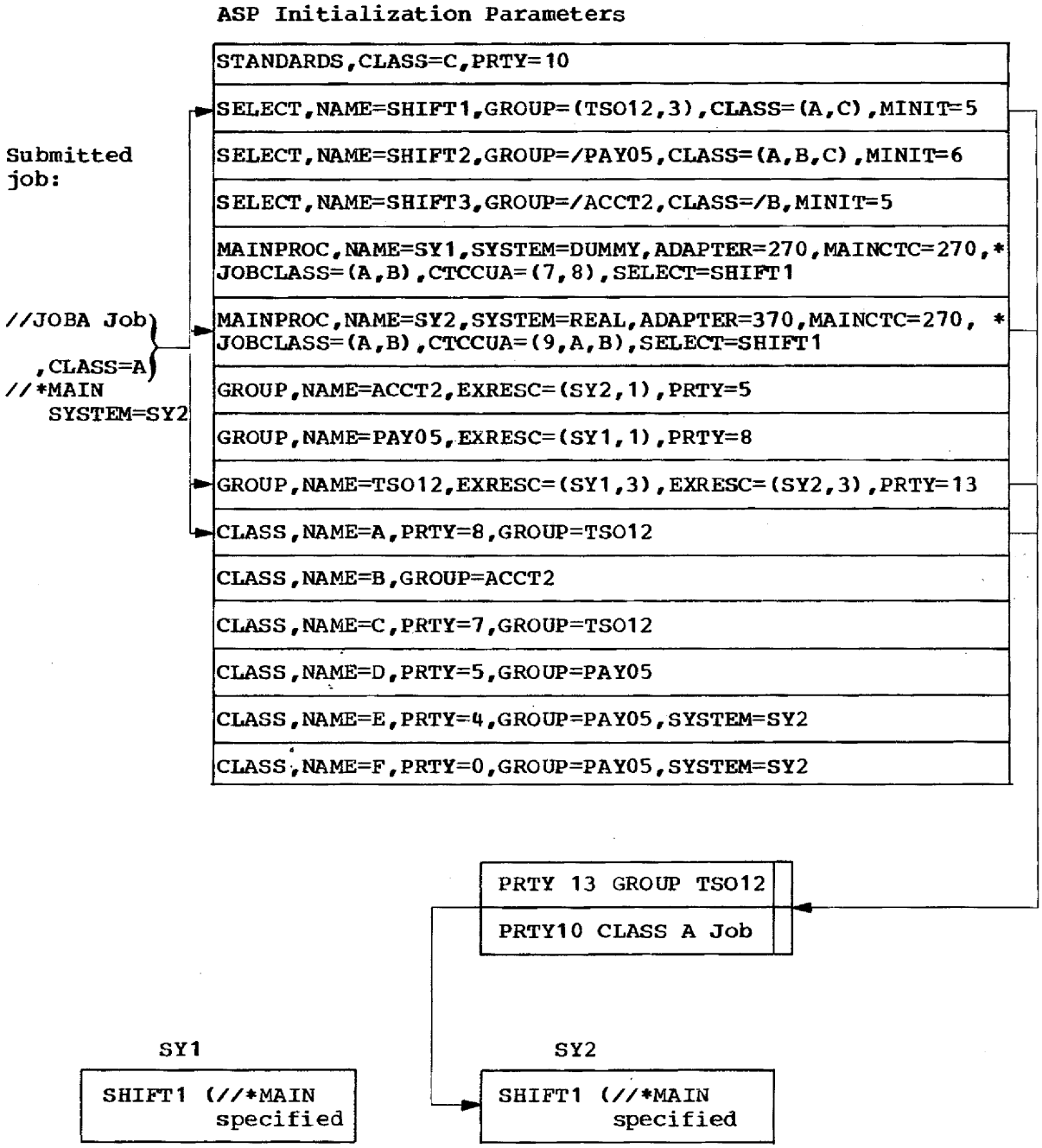


Figure 14. Generalized Main Scheduling - Example 1.

The module MAIN controls job scheduling and execution resource allocation. Processing begins when JSS queues the first job for Main into the RESQUEUE table. MAIN ALOADs MSVIPL to initialize the Main Processor for scheduling. MSVIPL returns to MAIN to begin job selection. Permanent execution resources are allocated to the necessary job class groups. Enough initiators (MINIT parameter on the SELECT card) and OS job classes (JOBCLASS parameter on the MAINPROC card) must be defined to accomplish this allocation. One OS job class is assigned to each group requiring permanent resources. Therefore, all initiators allocated to a group will execute under the same OS job class. If

insufficient resources are defined, an error message will be issued for each group that could not complete its resource allocation.

MAIN scans RESQUEUE for jobs that are eligible for execution. Since MAIN runs off a different FCT for each Main Processor in the system, it schedules jobs for only one Processor at a time. Therefore RESQUEUE is reduced to the subset of jobs that are awaiting Main execution and can run on the Main Processor currently being scheduled. MAIN operates under the SELECT mode constraints currently in effect as well as any job class constraints (that is, TDEPTH, MDEPTH, TLIMIT, or MLIMIT), that may apply to the current job mix on Main. In each group table entry there is a pointer to the first RESQUEUE entry for that group. A subchain in RESQUEUE chains together all jobs of a job class group. MAIN then scans this chain trying to select a job for execution. Each eligible job is examined according to the current scheduling CHOICE, (CHOICE parameter on the SELECT card). The selection scan continues until either a job is accepted for scheduling or the scan ends based on SELECT mode criteria. If a job from a group was not selected, the next available group is tried. The process then repeats itself until all groups are exhausted, at which time the scheduling pass ends. When a job is selected, MAIN ALOADs MSVINIT to initiatize the job for execution on a Main Processor. MSVINIT updates the job counters in both MPCDATA and the Job Class Table. Other job classes whose TLIMITs or MLIMITs refer to this class are checked to see if their scheduling limits are now exceeded. If so, those class entries are flagged as disabled from scheduling. MSVINIT then ALOADs a MSVDATA CSECT, initializes it with job related information, and enqueues it on the active job queue in MPCDATA. MSVINIT then calls MSVQMAP to transmit the OS job queue records to Main. When the job starts execution on a Main Processor, the module MAIN will attempt to schedule another job. Throughout job execution all CTC I/O (for the SYSIN and SYSOUT data sets) is processed by MAINIO, MSVMVT, and for a local Main, MSVLOCAL. Dataset Header information will be created by IOS on the Main Processor and passed to Main Service. When a data set is opened the IECASPO message is issued via a WTOR. This message is used to specify the data set information and the allocated CTC device address. This information is used to find the proper JDS entry or to create a new JDS entry. Thereafter a Data Set Title Label (DSTL) is used to correlate the data set and the JDS entry. The CTC device address is stored in the JDS entry.

The DSTL will contain the logical CTC device address, logical record length (LRECL), physical blocksize (BLKSIZE), record format (RECFM), and a Mode byte. The Mode byte specifies card-image (column binary) or EBCDIC information. The DSTL is created for every OPEN or when the passing of information for this data set is resumed after being interrupted by a different I/O operation.

The CCW that points to the DSTL has a command code of FD and is command chained to the next CCW for the actual data. If no intervening data set has been written across the CTC, the DSTL CCW will not be used.

The data set information from the DSTL is temporarily saved by MAINIO to be used to count records and create an ASP Header Record (ASPHDR). The ASPHDR will be written into the first buffer assigned to the data set. The first six characters of the record are ASPHDR and will be used by Print and Punch Service to determine the LRECL, BLKSIZE, and RECFM.

When a job ends, MSVMVT updates the MAIN Service tables and posts Main to begin another job selection pass. The job counters in both MPCDATA and the Job Class Table are reduced. Other job classes whose constraints refer to this class are checked for disabled scheduling. If the limits are no longer exceeded, the class is enabled for scheduling. In addition, the MSVDATA CSECT for the job is removed from the active job queue and placed in the queue awaiting WTR SMB output. MSVMVT sends a QWTR command to Main to start this process. A QWTR WAITING command

signals the end of SMBs and the MSVDATA is then enqueued for job termination. MAIN is posted for job termination and will ALOAD MSVTERM to remove the job from Main Service. MSVTERM terminates job tables created by MSVINIT and ADELETEs the MSVDATA CSECT. MSVTERM will terminate all jobs in its queue unless MAIN is posted for another scheduling pass. If this post occurs, MSVTERM returns to MAIN for job selection and will be ALOADed again when the scheduling pass is complete.

When ASP is restarted the initialization module INITREST will analyze the JCTs and flag those jobs that were active at the time of the failure. The sequence number of the Main Processor that a job was executing on will be put in the job's JCT. JSS will do a first time pass to verify that all required RESQUEUE entries are built before posting MAIN for a restart.

MSVIPL will determine which jobs were active on a Main Processor via the FENCE, LIST and DISPLAY, N= SOUT commands. MSVIPL will pass these jobs to MSVINIT to get a MSVDATA entry and have it initialized and then process the MSVDATA in the following manner:

Hot Jobs:

Active - fill in initiator ID and place on active queue

SOUTQ - place on WTR queue

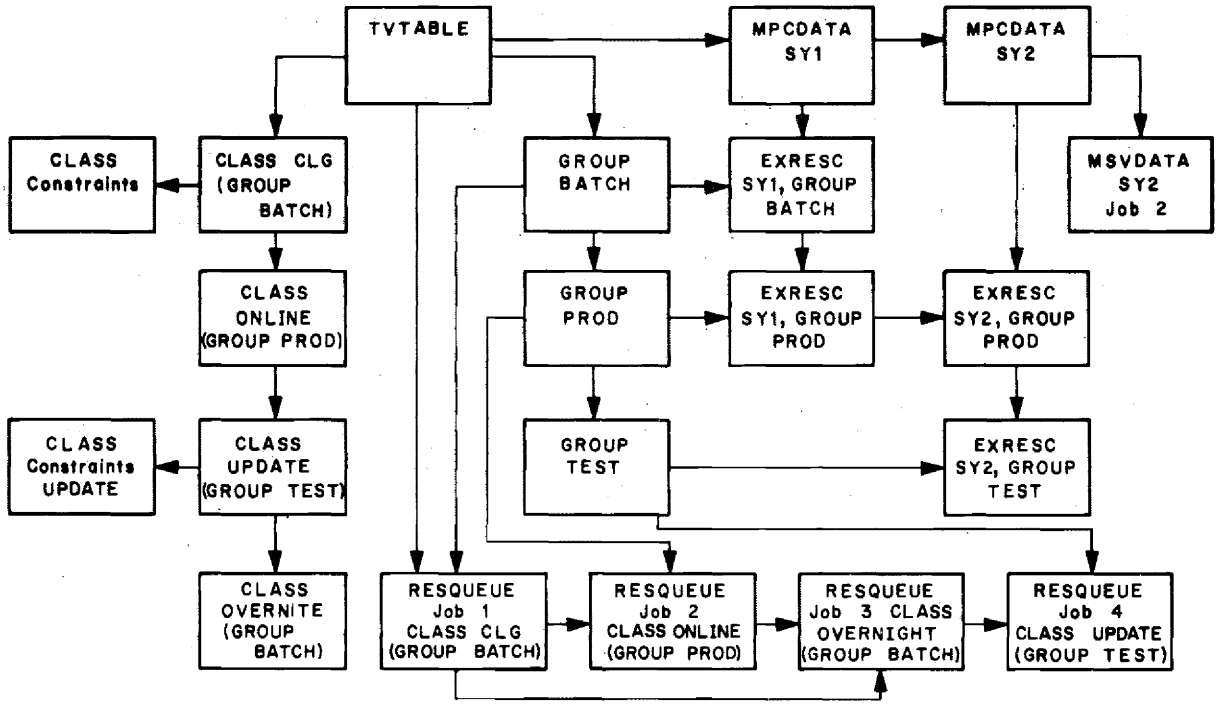
Non Hot Jobs:

Active - fill in initiator ID, place on active queue, flag IPLed off, and CANCEL

SOUTQ - place on WTR queue, flag IPL'ed off

Restart Opionts - processed by MSVTERM

The operator may communicate with Main Service using the START, RESTART, CANCEL, or MODIFY ASP console verbs. Upon receipt of an operator message MAINIO will ALOAD MSVOPER2 for a MODIFY command or MSVOPER1 for START, RESTART or CANCEL command to process the messages. Job selection modes can be switched dynamically by referencing a SELECT mode in a MODIFY command. MSVOPER2 will then search the SELECT records. An error message is issued if the SELECT cannot be found. Otherwise, the new selection mode parameters are moved into the select card position of MPCDATA. Main is posted if the number of initiators is altered causing resource reallocation to be necessary. Resources may be released by an ASP MODIFY command. All resources are allocated and unallocated in Main. JSS posts MAIN for allocation when the first job is enqueued for a Job Class Group that requires resources. MAIN is posted for unallocation by MSVMVT when the last job in this Job Class Group ends on a Main Processor. MSVOPER2 also posts Main for unallocation when resources are released. MAIN will then scan the Group Table in priority order performing the necessary allocation or unallocation.



MAIN SERVICE CONTROL TABLES

```
MAINPROC,NAME=SY1,ID=SY1,MDEST=M1,SYSTEM=LOCAL,CTCCUA=(7,8)          X
ADAPTER=270,MAINCTC=270,JOBCLASS=(A,B),SELECT=SHIFT1
MAINPROC,NAME=SY2,ID=SY2,MDEST=M2,SYSTEM=REAL,CTCCUA=(9,A,B)         X
ADAPTER=370,MAINCTC=270,JOBCLASS=(A,B),SELECT=SHIFT1
SELECT,NAME=SHIFT1,MINIT=5,MBAR=10,CLASS=/OVERNITE
SELECT,NAME=SHIFT2,MINIT=7,GROUP=(BATCH,2,TEST,3)
CLASS,NAME=CLG,GROUP=BATCH,TLIMIT=(ONLINE,0)
CLASS,NAME=ONLINE,GROUP=PROD,FAILURE=PRINT
CLASS,NAME=UPDATE,GROUP=TEST,MLIMIT=(SY2,ONLINE,1)
CLASS,NAME=OVERNITE,GROUP=BATCH,IORATE=HIGH,PRTY=6
GROUP,NAME=BATCH,EXRESC=(SY1,2,,IPL,MANUAL)
GROUP,NAME=PROD,EXRESC=(SY1,2,400K,IPL,MANUAL),EXRESC=(SY2,2)
GROUP,NAME=TEST,EXRESC=(SY2,3)
```

Figure 15. Generalized Main Scheduling - Example 2.

Main Storage Fencing

Main Storage Fencing enables the ASP user to structure Main Storage into predefined areas called Fences . Each Fenced area is associated with a group of ASP job classes. When a Fenced job is scheduled on an ASP Main Processor its region size is allocated from the Fenced area. A Fenced job cannot obtain Main Storage outside of its Fenced area and likewise Unfenced jobs cannot obtain Main Storage from within a Fenced area. As a result, once a Fenced area is constructed the jobs running within the Fence are isolated from other jobs on that Main Processor. In effect, a Fenced area is like an OS MFT partition.

Fencing a long running job solves a number of Main Storage management problems. First, storage fragmentation can be eliminated by allocating the Fenced area at the top of available Main Storage. Also, if the job has multiple steps with large region sizes the Fence prevents other jobs from taking this job's region space during a step change. And finally, if the job abends and must be restarted Fencing will reserve the Job's region until the restart can be accomplished.

A Fenced area is built whenever an ASP Job Class Group requiring a Fence becomes enabled for scheduling. This can occur in three ways, IPL time, Dynamically, or manually (operator). A Fence will be built for every permanently enabled Job Class Group whenever the corresponding Main Processor is IPLed. A Job Class Group is defined as permanently enabled on the GROUP control card. A Fence can also be dynamically built whenever a disabled Job Class Group becomes enabled. A group can become enabled for scheduling either by an ASP operator command or when a job of this group enters the ASP job queue. When a Fence is built, Main Storage is allocated below and adjacent to the MAINTASK region or a previously created fence. If Unfenced jobs are running in this area, their region space will be absorbed into the Fence upon job termination. Once the entire Fenced area is available, ASP will begin scheduling jobs in that Fenced area.

Fences remain in effect until the corresponding Job Class Group is disabled from scheduling. This can be done through an operator command or when the last job in a Fenced group ends on Main. When a Fence is reset the storage space is returned to the OS dynamic area and becomes available for allocation to Unfenced jobs.

Main storage fences cannot be built in LCS (hierarchy 1) storage. Jobs requiring region space both in hierarchy 0 and hierarchy 1 cannot obtain their hierarchy 1 region from a fenced area. Since main storage fence space is allocated directly below the MAINTASK region, MAINTASK must be started in hierarchy 0 if the main storage fencing feature is used.

As in the case of starting OS system tasks, care must be exercised to avoid both storage fragmentation and storage allocation delays when starting main storage fences. The fence space allocation technique is designed specifically to prevent storage fragmentation, in that fence space is always allocated contiguously starting either from the bottom of the MAINTASK region when no other fences are built, or from the bottom of the highest fence in storage below which enough unfenced space exists to hold the fence about to be allocated. If unfenced jobs are found to be executing in the storage area allocated to a fence, jobs from that fenced job class group will be prevented from scheduling until the unfenced jobs terminate.

The examples below represent OS dynamic area configurations prior to allocating a 200K fence:

1.	2.	3.	4.
MAINTASK no jobs or fences	MAINTASK 300K unused Fence B	MAINTASK Job XYZ (100K) Job ABC (200K)	MAINTASK Fence A

The new 200K fence will be allocated directly below MAINTASK in the first three examples. In the last example the new fence will be allocated directly below Fence A. Note that in example 2 100K of storage will be fragmented after the new fence is allocated. In this example the 300K unused space was a fenced area when Fence B was allocated. In example 3 the new fence cannot be used until both jobs terminate.

PRINT SERVICE

The Print Service DSP processes the system output for a job on printer (local or remote RJP) attached to the Support Processor. This DSP provides the necessary carriage control translation and deblocking of print lines, as well as handling the necessary error recovery.

Print Service expects data to be in EBCDIC format, ready for printing with either OS channel command forms control or the extended USASI channel command code supported by OS, which includes those codes defined by USASI FORTRAN as a subset. OS output may be variable, variable blocked, fixed, fixed blocked, or undefined. An ASPHDR record is written by Main Service as the first record in each output file. Print Service will get record format (RECFM), logical record length (LRECL), and block size (BLKSIZE) for each file from the ASP header record. Output to the printer is command chained and unprintable characters are translated to blanks prior to printing to reduce printer overhead. A full ASP buffer is output for each EXCP call to IOS.

Print Service has the ability to request the mounting of special forms, carriage tapes or print trains; to load UCS buffers for printers with UCS; to load forms control buffers (FCB) for 3211 printers; to vary the mode of forms control; and to print additional original copies. The primary vehicle for communicating such requirements is the SYSOUT card in the ASP Initialization deck. The SYSOUT card allows the system programmer, by keyword option, to indicate his special requirements for a SYSOUT class and hence for the printer(s) which support that output class. These requirements can be temporarily overriden for a data set via a //*FORMAT card for the referenced data set. The printer configuration is retained in the Printer Resources Table and updated as printer characteristics (train, forms, and carriage tape) are changed. The SYSOUT class requirements are retained in the SYSOUT class table (SCT) built from SYSOUT cards at initialization time.

Print Service processes the SYSMSG data set and any existing data sets for which SYSOUT classes were defined in the SYSOUT class table (via ASP initialization SYSOUT cards). Additional data sets may be specified by //*FORMAT cards.

The method by which output is allocated among the printers in an installation (printer resource scheduling) is controlled by parameters in the ASP initialization standards card, which establishes the basic installation guidelines; the ASP initialization printer card for each printer on the Support Processor, which provides the initial status of the printer and its setup status (that is, whether forms and/or the print train may be changed to another configuration); the ASP

initialization SYSOUT card for each supported SYSOUT class; and the //*FORMAT card for each data set.

The STANDARDS card names the installation-standard forms, carriage tape (forms control buffer module for 3211), and print train. These are the names used whenever the corresponding parameter in the printer cards, SYSOUT card, or //* FORMAT card is omitted or specifies STANDARD. Two additional standards card parameters affect printer allocation. The FLOCATE parameter indicates whether the printer resources table is to be searched to locate a printer which is already setup as required (minimizing operator intervention but possibly scattering data sets for a job between two or more printers) or whether the first printer available which may be set up to be used (minimizing the number of printers to be used by a job but requiring operator intervention to change forms, trains, etc). The parameter DLOCATE=NO may be used to override the DEST parameter from the SYSOUT class table or the //*FORMAT card, in the event that a disproportionate number of data sets are assigned to a given printer, in order to allocate processing more evenly among the available printers.

If the combination of DLOCATE=YES in the standards card and use of the DEST parameter in a SYSOUT card or //*FORMAT card causes a printer to be assigned for which setup has been barred by the printer card for the device, Print Service overrides the printer card parameter and requests a configuration change if required by the data set.

To assist the operator in collating printer output, a header page precedes the output for each data set, identifying the output by job name, job number, and data set name in large block letters. The output for the last data set on each printer used by the job is followed by a trailer page, the trailer page on the last printer used by the job included a tabulation of all data sets printed on all printers and job statistics. Header and/or trailer pages may be eliminated by specifying HEADER=NO and/or BURST=NO on each PRINTER initialization card where this option is requested. This option may be desirable when slow printers are being used on remote RJP workstations.

Figure 16 is a Print Service parameter cross-reference chart. In the lefthand column are listed those parameters that affect the printers output. The next columns to the right are the defaults assumed if no specific parameters are entered in those functions represented by the remaining columns. An (X) in a column indicates the corresponding parameter may be specified by that function. For example, the HEADER default is YES and may only be changed by an entry on the PRINTER card. Where two or more functions specify the same keyword parameter but of different values the keyword of the function to the extreme right is used. For example, if the TRAIN parameter is HN on the PRINTER card for printer PR1 and PN on the //*FORMAT card in a job whose printed output is destined for PR1 PN is assumed and the program scans the Printer Resource Table to check for the train mounted. A message to the operator informs him to mount the PN train if it is not mounted.

If the SYSOUT parameter is specified on a DD card that is referenced by a //*FORMAT card, DDNAME=, the SYSOUT parameter will be ignored.

Print Service Keyword Parameters	ASSEMBLED DEFAULTS	STANDARDS CARD	PRINTER CARD	SYSOUT CARD	//*FORMAT CARD	OPERATOR COMMANDS	PRUT
			Mutually Exclusive				
CARRIAGE	6 # Inch	X	X	X	X		X
CONTROL	Program			X	X	X	
COPIES	1			X	X		
DESTINATION	Group Original Name			X	X	X	
DLOCATE	Yes	X					
FORMS	Yes, 1 Part	X	X X	X	X		X X
FLOCATE	Yes	X					
HEADER	Yes		X				X
TRAIN	Yes, PN	X	X X	X	X	X	X X
UCS	Yes		X				
TYPE	Prt						
BURST	Yes		X				X
OVFL	ON			X	X		

Figure 16. Print Service Parameter Cross Reference Chart.

Train and Forms Modules

Print Service ALOADs the appropriate ASP print train module and, for 3211 printers, the appropriate Forms Control Buffer (FCB) module. The 1403 print train modules consist of the 240 byte UCS Buffer array, followed by a 256 byte translate table for the array. If the universal character set (UCS) feature is available, the translate table is used to translate to blanks all unprintable

characters in a print line in order to maximize printer performance. The 3211 print train modules consist of three fields: the first 432 bytes are the train-image, the next 15 bytes (433 through 447) and the last byte (512) are reserved for future use, and the remaining 64 bytes (448 through 511) are the associative field. Each character present in the train-image field has the appropriate bit turned on in the associative field. The Forms Control Buffer (FCB) modules contains one byte for each line on the form, to a maximum of 180 lines at eight lines per inch. Bits 4-7 at each position can contain one of twelve channel codes, hex 0 through C corresponding to the channel codes in carriage skip commands. Bit 3 is used as a flag bit in the first and last FCB position. Bit 3 on at position 1 causes six lines per inch spacing. Bit 3 off at position 1 causes eight lines per inch spacing. Bit 3 on at any position other than 1 identifies the corresponding line as the last line of the form. Unused bits should be set to zero. The FCB load module contains two bytes in front of the image. The first byte is X'80' and the second byte is the length of the image.

If the UCS FCB module specified does not exist in the ASP library, Print Service notifies the operator, who may respond with a message specifying the proper name. Additional UCS and FCB modules may be added to the system by assemblying the desired module and linkediting it into the ASP library with the print train name or FCB2 and the first four characters of the carriage tape name as the module name.

Operator Control

The operator commands *START, *RESTART, and *CANCEL are supported by Print Service. *START is used to notify Print Service that an operator action, such as forms mounting, has been completed and that processing may continue. The operator may also force a data set to be printed under single-space carriage control. *CANCEL causes the printing of a data set to be terminated and printing of the next data set to begin. Optionally, this command causes job processing to terminate immediately.

Under normal operation, *RESTART causes reprinting of an entire data set. Optionally, the entire job may be restarted (on the same printer or on a different one, depending on whether the current printer is online) at the beginning or at a position approximately two pages back from the current print line. In addition, the operator may request that the Universal Character Set Buffer be loaded on a local printer with a restart approximately two pages back from the current data position.

In addition to noting the current printer position for restart purposes, Print Service checkpoints the current print position approximately every 20 pages. In the event that a catastrophic error occurs in processing on the Support Processor and an ASP restart is made, Print Service automatically starts processing at the point of the last checkpoint. Assuming 65 lines to a page, this restart point will be no more than 20 pages from the point of abnormal termination. In addition printing will take place in the same printer(s) as were active before the ASP failure.

All equipment failures detected by Print Service result in an intervention required message with an appropriate diagnostic message. Printing resumes when the printer in question is readied or when an operator command is issued.

ASPNEWS Facility

The ASPNEWS facility of Print Service provides a DSP that will create an output data set that is printed after every job. This facility can be used to broadcast general information to the ASP system users. Print Service prints the ASPNEWS prior to the final burst page.

ASPNEWS is invoked as a normally submitted OS job by the following JCL input:

```
//ASPNEWS   JOB...
//*PROCESS  ASPNEWS
//*DATASET  DDNAME=ASPNEWS
        DATA CARDS
    (each data card produces one line of print)
//*ENDDATASET
```

The ASPNEWS output data set can be terminated by resubmitting the ASPNEWS job without data cards.

PUNCH SERVICE

The Punch Service DSP processes any output data sets for which SYSOUT classes were defined in ASP initialization SYSOUT cards for TYPE=PUNCH (SYSOUT, CLASS=B, TYPE=PUNCH) and any additional data sets described by //*FORMAT PU control cards, on the punches (local or remote) attached to the Support Processor. Processing is on a card-image basis; USASI stacker selection characters are ignored if they are present.

Each data set that is punched is identified by a header and a trailer card containing the ASP job number, job name, and data set name. This separator card also appears between multiple copies of a given data set's output. If special card forms are required, or if standard forms are required following the use of special card forms on the punch unit, the operator is notified to place the required forms in the punch unit.

The operator commands *START, *RESTART, and *CANCEL are supported by Punch Service:

- *START,punch-name — A requested forms change has been made, and processing may commence.
- *RESTART,punch-name — When the job is restarted, only those data sets not already completed will be processed.
- *RESTART,punch-name,J — At restart time, the job will be restarted from the beginning; all data sets already processed will be repeated.
- *RESTART,punch-name, P=punch-name — The job will be restarted with the current data set. Output for the current data set will be switched to the named punch.
- *CANCEL,punch-name — Processing of the current data set is canceled; processing resumes with the next data set (if any).
- *CANCEL,punch-name,J — The remaining PUNCH data processing for the entire job is canceled.

PURGE

The Purge function is the last processing step for a job in the ASP system. It accesses the JDAB and the JDS for a job and, through ASPIO, releases all disk tracks assigned to the job. The APURGE macro returns the tracks of a job by oring the bits of the job's Job Allocation Table back into the system Track Allocation Table. This effectively makes these tracks available for allocation to subsequent jobs. (Refer to ASPIO discussed earlier in this chapter). A message is issued to the operator indicating that the job has been purged from the system and upon return to JSS the JCT entry for the job is deleted.

Since Purge is the last processing step for a job, it is the point at which the installation accounting program can be placed to summarize the accounting data for a job.

The ASP installation accounting routine punches master and detail cards as specified in the ASP initialization ACCOUNT card . An installation may use these cards or may modify the program to produce cards tailored to its needs.

An Accounting Print and Summary Routine (ACCPR) is distributed as a Dynamic Support Program with the ASP system to process the output of the Purge accounting routine. Any modifications to the Purge accounting routine output should be reflected in the DSP ACCPR.

The accounting routine may be removed entirely from Purge and, if desired, replaced with an installation-written routine. Purge is scheduled and determines during execution whether it requires punching to be performed. Only in this event does Purge attempt to acquire a punch unit. If an installation-written accounting routine requiring additional units is substituted, or if it is more desirable to ensure the availability of the required device prior to entering Purge, the device requirements entry, as defined by the DSPDC macro in the RESPARAM module, should be altered accordingly.

DEPENDENT JOB CONTROL (DJC)

Characteristically, large commercial data processing applications are complex systems composed of many interactive dependent jobs. Dependence within these systems requires a series of jobs to be executed in a specific order. Operations personnel are generally extensively involved in supervising the execution of these systems to ensure that they are executed in the proper order.

Dependent Job Control (DJC) is a function within ASP that relieves this situation by managing and supervising jobs that are dependent upon one another. A collection of dependent jobs is called a job network. Figure 17 describes a simple job network where Job C is dependent upon the completion of Job B and Job B is dependent upon completion of Job A.

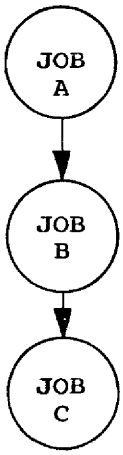


Figure 17. Sample Job Network.

Job dependencies of a more complex nature are typical. DJC manages both simple and complex job networks. Figure 18 describes a more complex network.

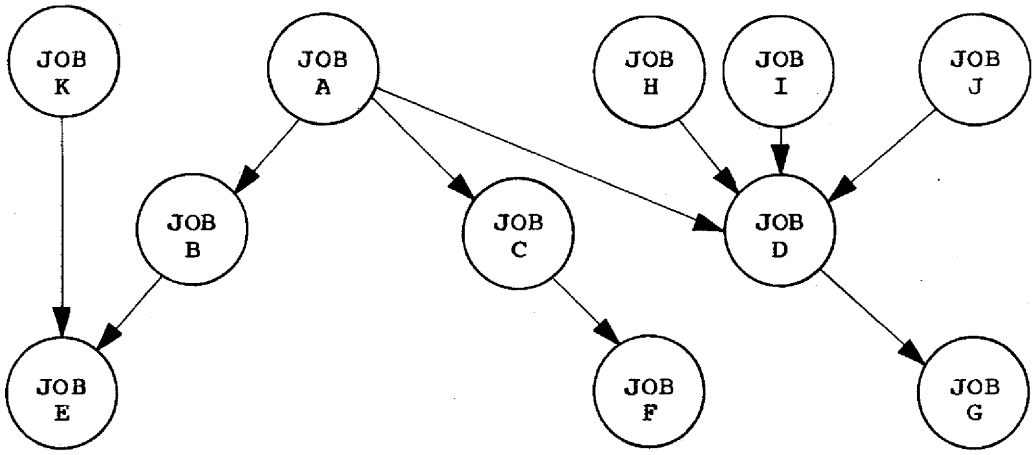
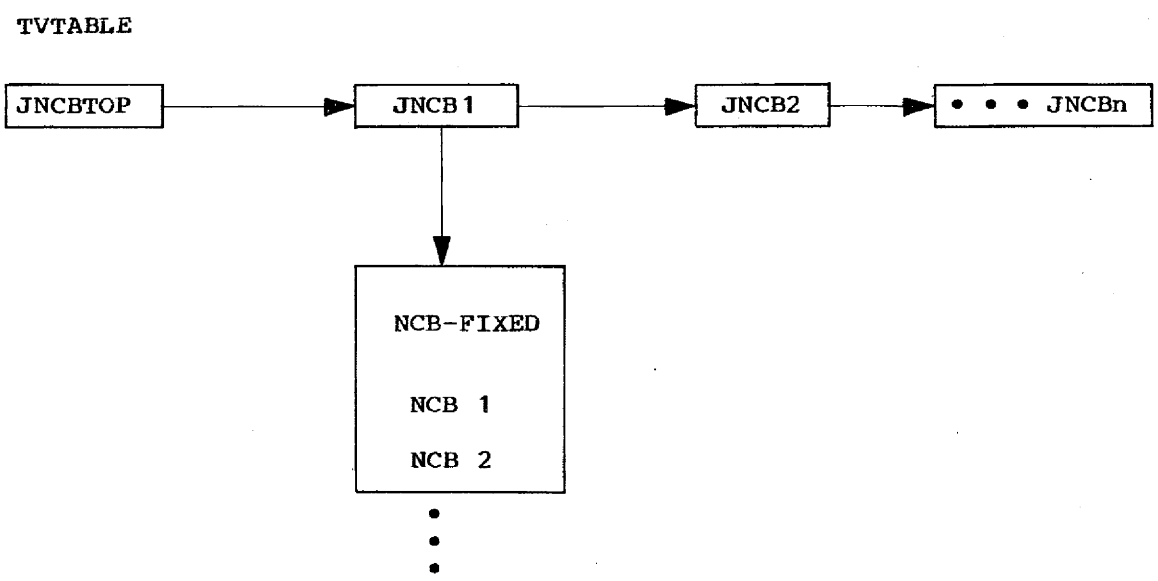


Figure 18. Sample Job Network.

Job dependent networks are defined by the application programmer's specifications in the job's JCL. Normally, no operator action is required to invoke DJC, however, *INQUIRY and *MODIFY commands are provided to interrogate and alter DJC specifications. Additional information about Dependent Job Control is provided in both the ASP Operator's Manual and the ASP Application Programmer's Manual. It is recommended that the reader refer to DJC in the above manuals as the following discussion assumes knowledge of information contained therein.

Dependent Job Control (DJC) Control Blocks

There are two control blocks pertinent to DJC. The Job Net Control Block (JOBNET) reflecting total net information and the Net Control Block (NCB) reflecting specific job information. The control block structure is as follows:



The JNCB's are core-resident areas chained off the TVTABLE. They reflect such information as total number of jobs in the network, number of jobs completed processing, network-id, and NCBFDB pointer. The NCB's are packed into chained single-record files. An NCB is constructed for each job of a given network. It contains job pertinent information such as: jobname, jobnumber, successor jobnames, and //*NET parameter information.

Internally, Dependent Job Control consists of three phases of DJC network-management:

- Initialization of a job-network
- Scheduling/Supervision of a job-network
- Termination of a job-network

Initialization of Job Network

A job network is defined to the ASP system during Input Service processing of JCL with detection of a //*NET control card. A search is made to determine if the job-network defined already exists. If it does not, Input Service constructs a Job Net Control Block (JNCB) for the newly defined job network. The JNCB is added to an existing JNCB chain. An ASP buffer is obtained and its FDB is placed in the respective JNCB. Next, a Net Control Block (NCB) is created and moved into the NCB buffer.

Nonstandard DJC jobs are defined with the inclusion of a //*PROCESS DJC control card. In this case a scheduler element (SE) is created to invoke the DJC process routine when the SE is scheduled by JSS. The

location of the //* PROCESS DJC card within a given set of //*PROCESS cards will determine at what point in the job DJC processing should be invoked, so that any dependent successor jobs can then be scheduled by DJC.

In the event that a DJC designated job fails during Input Service, its NCB is preserved and marked resubmitable provided the job did not fail as a result of a syntax error. In this case no NCB is constructed. When it is reentered, the NCB for that job is updated to reflect the change in status unless the job's NCB had been updated by a predecessor or had previously completed, in which case the original NCB is preserved. A restriction imposed upon the resubmitted job is that the //* NET card must contain the same number of successor jobs as the originally submitted //*NET card.

The last phase of initializing a DJC job is flagging its JCT entry to indicate it as a DJC job. If NHOLD was specified in the //*NET control card then, the JCT constructed by Input Service for this job is placed in DJC hold status. DJC hold occurs prior to the scheduling of the Reader/Interpreter scheduler element.

Scheduling/Supervision of a Job Network

DJC uses four major functions within ASP. They are: Input Service, JSS, Main Service and the DSP, DJCUPDAT. Figure 19, shows the relationship between these major functions. Input Service was discussed in the previous section.

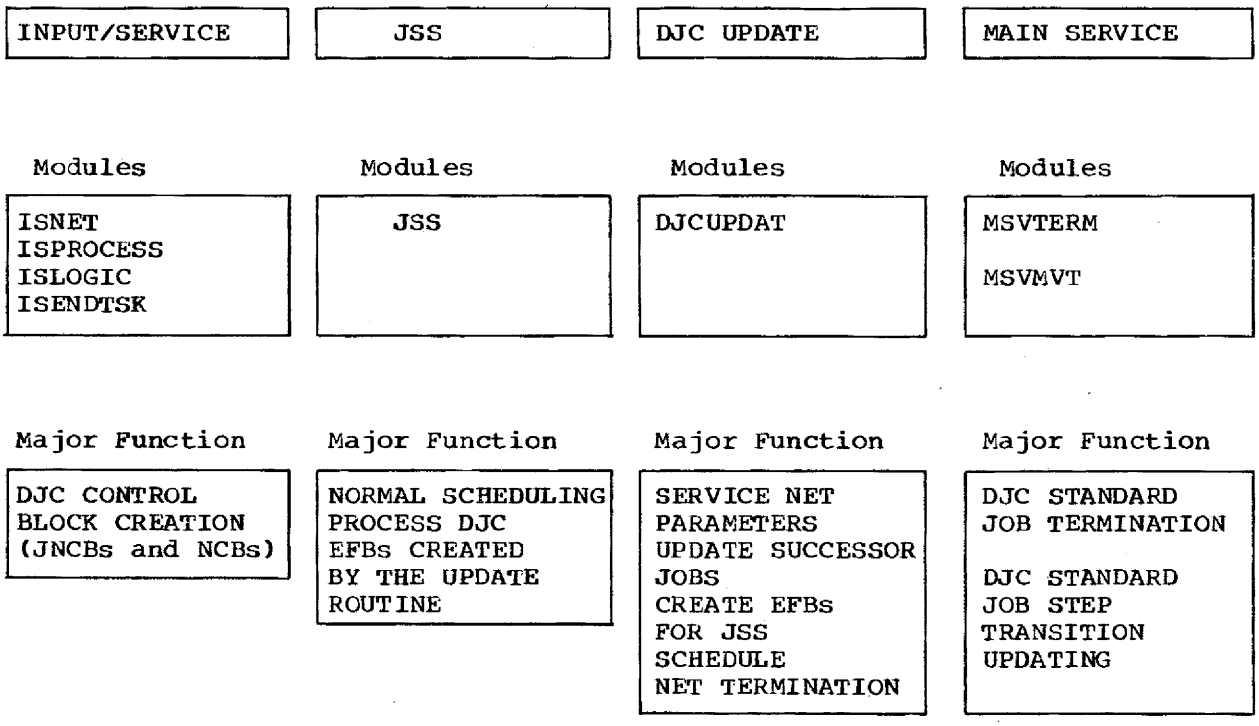


Figure 19. DJC Functional Relationship.

JSS schedules the next segment of a job. For a DJC job, scheduling is bypassed as long as the job is in a DJC-hold or DJC-operator-hold state. If a job is submitted without an NHOLD count specified it is considered as an origin node (no predecessor jobs) and is eligible for immediate scheduling by JSS. Origin nodes initiate net processing.

When a DJC job completes, it must update the NHOLD count of any dependent successor jobs. A job's NCB contains a NHOLD count specifying the number of predecessor jobs that must complete before it is made eligible for scheduling. For a standard job (no //*PROCESS cards) DJC completion is at the end of Main Service processing of that job. For a nonstandard DJC job, the update occurs when the //*PROCESS DJC Scheduler Element is scheduled by JSS. If the special WTO scheduling option is used, as described in the Applications Programmers Manual, the update occurs when a DJC WTO is received from a problem program.

JOB completion of a DJC job invokes the DJC update routine. DJCUPDAT decrements the NHOLD count of each successor job. When the NHOLD count of a job is decremented to zero all predecessor jobs have completed. In this case an Ending Function Block (EFB) is created by DJCUPDAT and placed on an EFB queue chained off of the TVTABLE, JSS is then posted. A function of JSS is to scan for EFBs and take appropriate action when one is encountered. A DJC EFB causes JSS to search for the corresponding job's JCT and release it from DJC hold.

Instead of decrementing the NHOLD count two other options, Flush or Retain, are available when DJCUPDAT services the NCB of a DJC job. Flush causes a Work-To-Do (WTD) cancel entry to be created and the job and its successor jobs are flushed (reference Work-to-do Driver routine). The Retain option means that no action will be taken. This suspends the job and its successors until positive action is taken by the operator or by resubmitting its predecessor job. When a predecessor job is resubmitted, only successors that specified the retain option will be updated.

Specifying the RELSCHCT parameter on the //* NET card allows the job to be released at a NHOLD count other than zero. In this case the job is scheduled up to, but not including, Main Service and then placed in DJC-hold in RESQUEUE. When the NHOLD of this job decrements to zero, DJCUPDAT accesses RESQUEUE and releases the job for Main scheduling.

The optional NETREL parameter of the //*NET card is also acted upon by the DJCUPDAT routine. The NCB for the jobname specified in the NETREL parameter is accessed and updated via the JNCB of the net specified in the same manner as a successor of the major job network.

In the case of a predecessor job abnormally ending, an ABNORMAL parameter may specify Retain, Flush, or Decrement in which case action taken is as described above. If a predecessor is terminated because of a JCL error, it must be corrected and resubmitted. No action is taken toward successors in this case.

Abnormal completion is assumed when a message is received for a job with the prefix "IEF45". This applies to standard jobs only. Nonstandard jobs are considered to always complete normally.

Termination of a Job-Net

Each time a DJC job completes, the DJCUPDAT routine accesses the associated JNCB. The JNCB job-completed count is incremented and when the count equals the JNCB total-job-count the net is purged. Situations can arise where all jobs have been submitted in a network and have been purged, but the network has not completed. This happens when there are

missing successors, and/or when some net jobs failed at Input Service time. If a system failure occurs when a network is in this state, the network will be lost over a system restart. A checkpoint of the net can be preserved in this situation if a pseudo job is included in the network in net-hold status. When all jobs of the net have completed, the pseudo job can be canceled by the operator or flushed by the last real job of the net. Net termination will then occur.

DJC ASP Interfaces

The DJCUPDAT module serves as both a driver module and an update module. It is composed of three major areas; (1) initialization, (2) update logic, (3) console message appendage. Its interface within the ASP system is via utilization of the INTERCOM function. There are seven modules that interface with DJCUPDAT via INTERCOM. Refer to Figure 20.

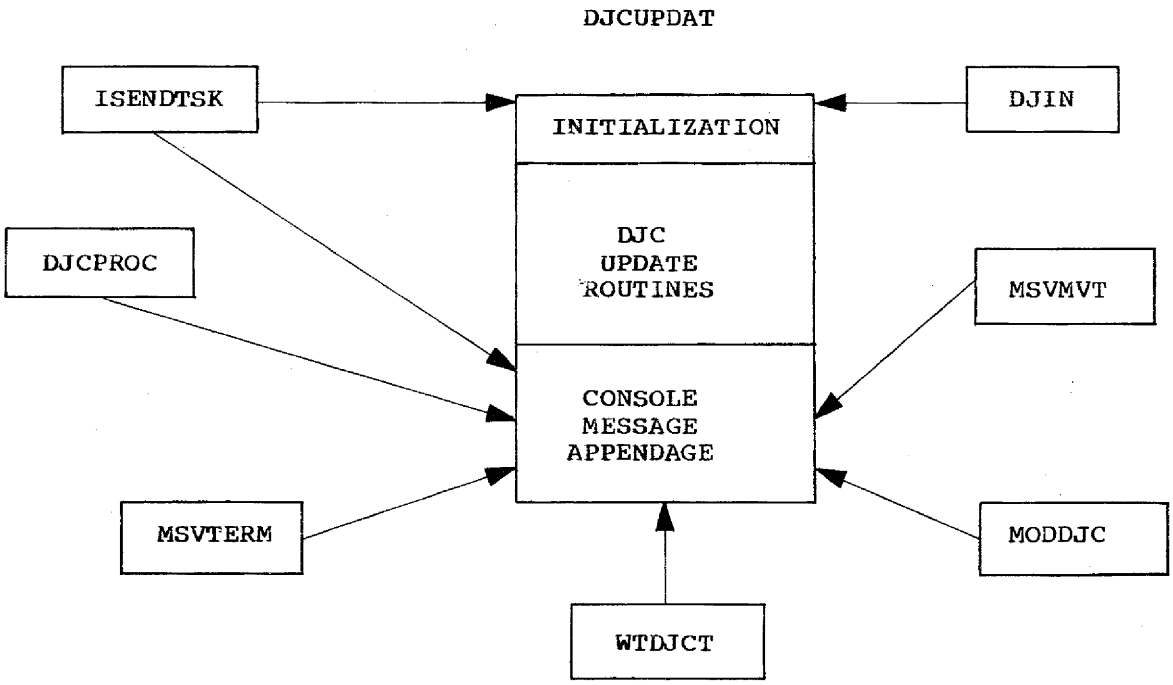


Figure 20. DJCUPDAT ASP Interfaces.

DJC INTERCOM MESSAGE FORMAT

Issuing Module	Command	Module	,	Specify Jobname	,	Specify NETID	,	Flag1	Flag2
1. ISENDTSK	*X	DJCUPDAT							
2. ISENDTSK	*S	DJCUPDAT	,	NO	,	NO	,	F6	40
3. DJCPROC	*S	DJCUPDAT	,	YES	,	YES	,	F2	40
4. MVTERM	*S	DJCUPDAT	,	YES	,	YES	,	F1	40/F0 *
5. WTDJCT	*S	DJCUPDAT	,	NO	,	YES	,	F4	40
6. MODDJC	*S	DJCUPDAT	,	YES	,	YES	,	F5	40
7. MSVMVT	*S	DJCUPDAT	,	YES	,	YES	,	F3	40/F0 *
8. DJIN	*X	DJCUPDAT							

* A X'F0' in FLAG2 denotes abnormal termination.

DJC INTERCOM MESSAGE FUNCTION

1. ISENDTSK on first encountering of a DJC job, calls DJCUPDAT.
2. ISENDTSK when adding a DJC job to the system, checks Network status for missing successor jobs. If this situation exists, ISENDTSK intercoms a START command to DJCUPDAT.
3. DJCPROC intercoms to DJCUPDAT for nonstandard DJC job updating.
4. MSVTERM intercoms to DJCUPDAT as a result of a DJC job completing Main Processor execution.
5. WTDJCT intercoms to DJCUPDAT as a result of a network being canceled.
6. MODDJC intercoms to DJCUPDAT for DJC job scheduling as a result of a modify network command.
7. MSVMVT INTERCOMs to DJCUPDAT for DJC updating at problem program time as a result of receiving a DJC WTO message.
8. DJIN intercoms to DJCUPDAT if a network is dumped back into the system when DJC is not active.

DEPENDENT JOB CONTROL - ACCESS ROUTINES

The DJC table routines are used to access and manipulate the DJC control blocks. For each routine, there is a macro-instruction which generates a calling sequence that branches to the corresponding DJC routine via the TVTABLE. The following are the DJC routines that comprise the module NETCONTL:

Macro	DJC Routine
JNADD	JNCB ADD
JNDEL	JNCB DELETE

JNGET	JNCB GET
JNCBHLD	JNCB HOLD
JNCBREL	JNCB RELEASE
NCBTAFND	NCB FIND
NCBTAADD	NCB ADD
NCBTAGET	NCB GET
NCBTAREL	NCB RELEASE
NCBTAPUT	NCB WRITE

Use of the DJC Access Routines

The JNCB is a core-resident chain of control blocks representing DJC job-nets within the ASP system. These control blocks are accessed synchronously via utilization of the DJC access macros.

The following represents the functions of the JNCB access routines. The AENQ and ADEQ functions reference JNCB's via the parameter NAME=JNCBCTL.

To add a JNCB to the JNCB chain requires the following:

1. Use the ASP AENQ function to provide synchronous access to the JNCB's.

2. Use the JNADD routine to insert the JNCB entry.

3. Use the ASP ADEQ function to make the JNCB's available to other functions.

To delete a JNCB from the JNCB chain requires the following:

1. Use the ASP AENQ function to provide synchronous access to the JNCB's.

2. Use the JNDEL routine to remove the entry from the JNCB chain and make it available to other routines.

3. Use the ASP ADEQ function to release the JNCB chain.

To get the next entry in the JNCB chain requires the following:

1. Use the ASP AENQ function to access the JNCB's.

2. Use JNGET to get the first entry.

3. Use successive JNGETs to get successive entries. The EOF exit must be taken before dequeuing the JNCB chain.

4. Use the ASP ADEQ function to make the JNCB's available to other routines.

The NCBs are defined in ASP as chained single-record files. NCBs are sequentially added to a given buffer. Once the buffer is saturated, a new one is gotten and chained to the previous.

The following represents the functions of the NCB access routines:

To add an NCB to the applicable buffer requires the following:

1. Use the JNCBHLD routine to provide synchronous access to the applicable JNCB.

2. Use the NCBTAADD routine to add the specified entry.

3. Use the JNCBREL routine to make the JNCB available to other routines.

To get the next NCB entry from a buffer requires the following:

1. Use the JNCBHLD routine to gain access to the applicable JNCB.

2. Use the NCBTAGET routine to get the first NCB.

3. Use successive NCBTAGET's to get successive entries.

4. Use the NCBTAREL routine to release the NCB buffer if unchanged or the NCBTAPUT routine to write the NCB to disk if it was changed.

5. Use the JNCBREL routine to make the JNCB available to other routines.

To locate a specific NCB requires the following:

1. Use the JNCBHLD routine to gain access to the applicable JNCB.

2. Use the NCBTAFND routine to locate requested entry.

3. Use the NCBTAREL routine if the requested entry was found.

4. Use the JNCBREL routine to make the JNCB available to other routines.

CALLABLE DYNAMIC SUPPORT PROGRAMS

In addition to those routines that are an integral part of every ASP nucleus and the basic DSP's (for example, ASP R/I, Main Service, etc.), there exists a group of callable DSPs that meet specific needs of many ASP installations. The modules of these optional DSPs are normally nonresident until invoked via operator request through the resident DSP, CALLDRVR. A DSP may be invoked either by an operator *CALL or an internally issued INTERCOM macro -instruction which simulates an operator message.

These callable DSP's are provided to all ASP users on the distributed tape and are designed to meet general needs of most installations. The performance of the callable DSP's can be increased by specifying that frequently called modules be made resident. This is accomplished during ASP initialization via the RESIDENT card

Callable DSP's include the Input Service DSP's; CR, TR, and DR discussed in Chapter 4, Background Utilities such as Card-to-Tape (CT), Tape-to-Print (TP), etc..

This section describes those callable DSP's not of a basic or utility nature, but providing important ASP support. They are:

- Deadline Scheduling - Invoked via a //* MAIN card parameter with a specified deadline type that was defined during ASP initialization. Once invoked, it can be canceled and recalled from the operator console. This ASP function considerably increases the job's chances of completion on time by dynamically increasing the job priority.

- Remote Job Processing - Invoked via an operator *CALL. This ASP function allows jobs to be submitted from remote ASP terminals.

- Network Job Processing - Invoked via an operator *CALL or a //*PROCESS card. This function allows work to be transmitted between ASP Support Processors, allowing work load balancing or taking advantage of specific system capabilities or configuration.

- Internal Job Processing - Invoked via an operator *CALL. This function allows a program executing on a Main Processor to submit work to Print and/or Punch Service to be outputted before the programs termination, or the program may create a job to be sent to Input Service, to be placed in the ASP queue.

- ASP Created Data Sets (ACDS - TSO Support) - Invoked via a //*FORMAT or //* PROCESS card , this function allows the TSO terminal user to have his job processed and scheduled by ASP.

DEADLINE SCHEDULING

For the majority of data processing installations, there exists a class of jobs that must be processed within a specified timeframe in order to meet required schedules. The callable DSP, DEADLINE, increases the ASP priority of a job based on a Deadline type assigned to the specified job. Deadline types are defined during initialization in the DEADLINE card . Deadline Scheduling normally functions without operator action, however, console commands INQUIRY, MODIFY, CALL, and START are provided and are described in the ASP Operator's Manual. The MODIFY command can override the initialization parameters.

Any job entering the ASP system may be specified for Deadline Scheduling via a //* MAIN card with a DEADLINE parameter. Input Service, recognizing the DEADLINE parameter will call the nonresident module DEADLINE if it is not in the system. Input Service places an entry in a Deadline queue to enable Deadline Scheduling to control the job.

Deadline examines the Deadline queue and issues an ATIME macro for the shortest time interval until a job priority change is required. Unless another job with a shorter time interval is entered by Input Service, Deadline will remain in an AWAIT until the time expires at which time the appropriate priority change is made. If there are no jobs in the Deadline queue, an ATIME is set to expire at midnight and Deadline will AWAIT until it is operator canceled or Input Service enters another Deadline job. Deadline Scheduling, canceled via a *CANCEL, DEADLINE, PURGE command will not be automatically called by Input Service. In this case, DEADLINE will not resume unless ASP is reinitialized or an operator *CALL, DEADLINE is issued.

REMOTE JOB PROCESSING

The ASP system uses the Remote Job Processing (RJP) facility for support of batch processing requests from remote terminals. The mode of transmission used is Binary Synchronous Communication (BSC). RJP is written at an EXCP level and provides the control interface between the Dynamic Support Programs (DSP's) and the terminals.

The structure of RJP is comprised of two logical sections. There is the RJP line manager which controls all the line activities, and the Remote Terminal Access Method (RTAM) which is the DSP interface for blocking and deblocking data into the appropriate transmittal buffers.

ASP RJP provides two data formats and control philosophies. The first format, is the intelligent terminal support. This is compatible with the ASP/HASP Remote Workstation Packages for the System/360, System/3, 1130, Model 20, and the 2922. With this format, interleaving and the ASP operator console with limited- or full-function are supported.

The second data format is compatible with hardware terminals such as the 2770, 2780, and the 3780. This format is unidirectional and provides no console support.

All error checking and error recovery is automatic and does not require any local operator intervention. Line error statistics are accumulated automatically and may be printed on a console via the RJP INQUIRY line statistics operator command.

Functional Description

At DSP Initialization, a resident RJP table is built from the RJPTERM and RJPLINE initialization control cards. An entry is created for each defined line and terminal. Within each entry is a pointer to a file which contains all the required preformatted control blocks for normal RJP operations. This includes the Line/Remote Device Control Table(s), SUPUNIT(S), and buffer requirements.

To initiate the RJP mode of operation, the local operator will invoke a call for RJP. This will give the RJP driver module control. From the driver, the RJPMAIN module is loaded, followed by a RJP active message to the console. RJP is now ready to service the lines. RJPMAIN load module consists of six modules linked together. They are: RJPMAIN1, RJPMAIN2, RJPMAIN3, RJPMAIN4, RJPMAIN5, and RJPMAIN6.

RJPMAIN1 is the RJP line manager, which handles DSP dispatching along with all non-data line communications. The RJPMAIN2 module is the Remote Terminal Access Method (RTAM) which provides the access to the RJP facility (that is, OPEN, GET, PUT, and CLOSE routines). Both the RJP Line Manager and RTAM, call subroutines in the RJPMAIN3 and RJPMAIN4 modules. RJPMAIN5 contains an overlay for the RJP TP buffer and is used to initialize the transmittal buffers. RJPMAIN6 is a data CSECT for RJP.

Only one copy of RJP may be active at any one time and the line manager runs off the RJP Function Control Table (FCT) entry. During the initialization of the RJP DSP, entries in the Transfer Vector Table (TVT) are updated to point to the corresponding access routines in the RJPMAIN2 module. The ASPOPEN, ASPEXCP, and ASPCLOSE nucleus routines can now make the appropriate entry into the RTAM access routines. When RTAM facilities are used, the caller's FCT will be used for any AWAITs. Dispatching of these FCTs will be controlled by the Line Manager.

Multileaving Line Manager

This processor controls all line activity with remote terminals, including line initiation/termination, remote terminal synchronization, line error recovery, and sign-on/sign-off processing. It interfaces very closely with the Remote Terminal Access Method described below.

When this processor receives control from the dispatcher, it first determines whether an I/O operation has completed. If not, it scans each line (via the Line Device Control Tables) to check for requested processing. When all processing has been completed, the processor returns control to the Multifunction Monitor (AWAITs) until more work becomes available.

When a channel end is detected, the channel end routine determines the sequence type of the Channel Command Word Chain and branches to the appropriate section to analyze the channel end and initiate any error recovery procedures required.

The Line Device Control Tables (LDCT's) are scanned. When one is found to be available, the line initiation routine is entered to acquire the DCT and a TP buffer, construct an initial CCW chain, and initiate I/O on the line.

A single timer queue element is maintained by the Line Manager to initiate delays in line processing in order to delay a null response to a remote terminal and decrease the associated degradation.

Remote Terminal Access Method (RTAM)

The Remote Terminal Access Method provides an interface between the ASP DSP and the remote terminal. RTAM provides blocking/deblocking, compression/decompression, and synchronization with the remote terminal in such a way that the DSP need not be concerned with the characteristics of the remote terminal with which it is communicating. The Multileaving Line Manager synchronizes very closely with RTAM through a series of subroutines.

RTAM consists of four main sections and some miscellaneous subroutines, which are described in the ASP Logic Manual. The four main sections are:

- The OPEN routines, which convert the line from an idling mode of operation to a transmit and/or receive mode of operation. In the case of the multileaving interface, this routine also generates the request or permission to begin a new function.
- The GET routines, which convert data received from the line into EBCDIC images suitable for processing by the ASP DSP's. This conversion includes deblocking, decompression, and conversion from line code to EBCDIC.
- The PUT routines, which convert data from EBCDIC into a form ready to be transmitted to the remote terminal. This conversion includes compression, blocking, and conversion from EBCDIC to line code.
- The CLOSE routines, which convert the line from a transmit or receive mode of operation to an idling mode of operation.

RJP OPERATING ENVIRONMENT

Data transmission to a remote terminal will start after the terminal has signed on to the local ASP system. The following sequence of events are required for signing on:

1. With RJP active, a line can be activated by a start line command. This will dynamically build the line DCT (LDCT), SUPUNITS, and the signon buffer. The line adapter will now be conditioned to service calls from a remote terminal.
2. The line manager will monitor for a terminal /*SIGNON card, after the phone connection has been established.
3. If a SIGNON request is invalid, a message is issued and is followed by the line being automatically canceled. This will cause the phone to disconnect.
4. With a valid request, the required SUPUNITS, Remote DCT's (RDCT's), and terminal buffer will be dynamically built along with any printer resource entries when required.

5. From this point, the terminal and RJP are ready to accept data transmissions.

Some of the available facilities in RJP are:

- After signon, the remote terminal console will be available to the user.
- After signon, the terminal device will be available to the ASP system. Use of these devices will be the same as operating with any local device.
- After signon, line error statistics and total number of transmissions are accumulated automatically. INQUIRY command can be invoked for displaying of these statistics.
- Line error recovery is automatic and appropriate error messages are issued.
- A RJP line CANCEL command is available with two options. The line (and terminal) can be canceled immediately, which implies critical DSP's (that is, Print and Punch Service) will be specialized rescheduled. The normal line cancel, which allows currently active DSPs to end, prior to canceling the line. This command can be invoked anytime after the line has been started.
- After signon, the remote terminal may terminate transmissions by submitting a signoff card. This will operate the same as normal line cancel, followed by the line being restarted automatically.

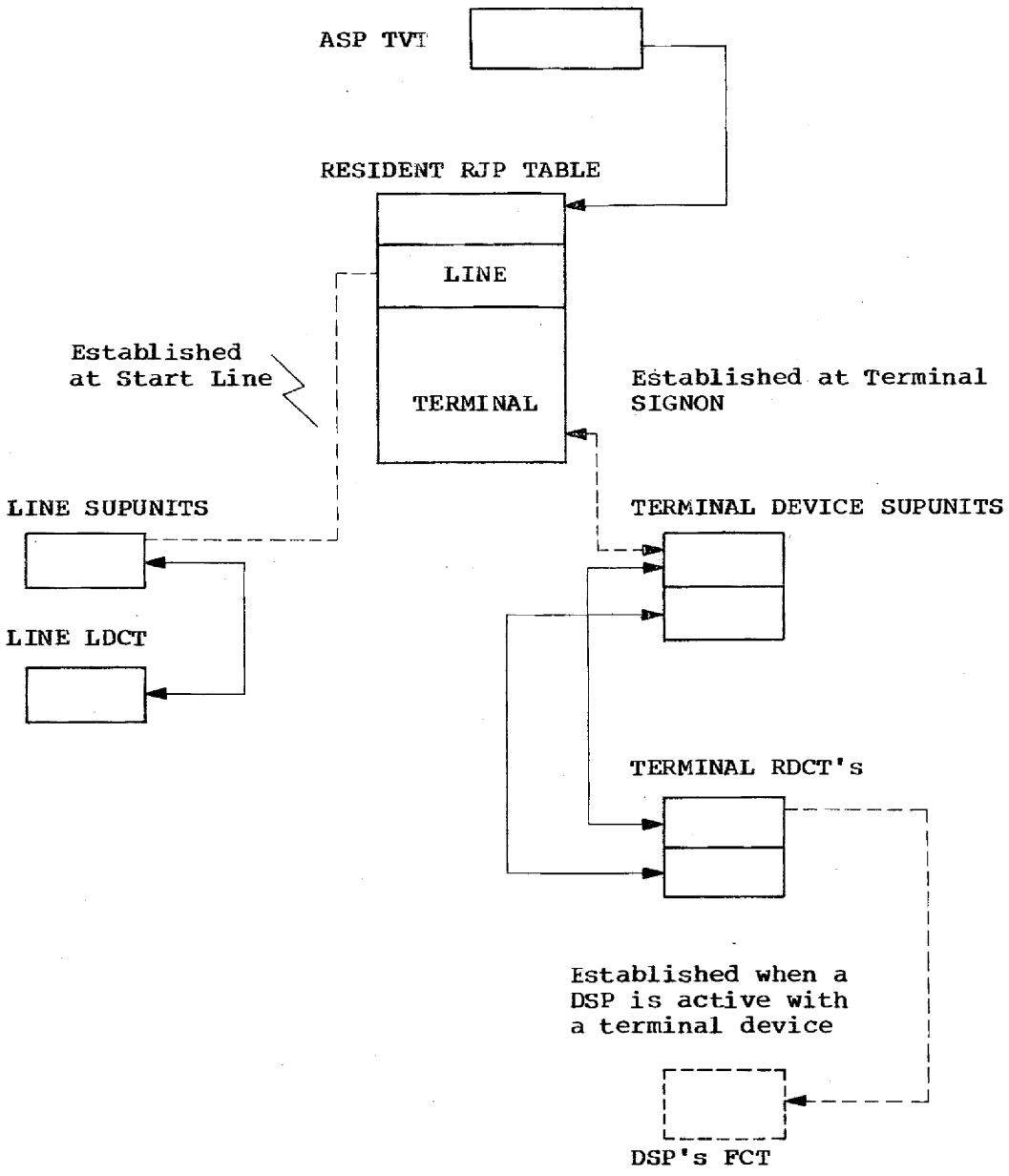


Figure 21. Interrelationship of ASP RJP Control Blocks.

REQUIREMENTS FOR REMOTE RJP TERMINAL PROGRAMMING

ASP Programmable Terminal Teleprocessing Compressed Data Format

Data transmitted to or from programmable terminals is first compressed (or pressed into blocked transmission groups, with all strings of three or more (up to 31) duplicate characters reduced to two-byte control groups. Duplicate blanks are reduced to one character. This compressed format allows improved transmission speed by not transmitting redundant information. Print records have the first character reserved for

carriage control, the extended USASI code supported by OS which includes those codes defined by USASI FORTRAN. Punch records are 80-character EBCDIC card images for punching in data mode 1. Appendix E of this manual provides a detailed description of this compressed format.

The RJPMAIN2 module contains the Remote Terminal Access Method (RTAM) routines which blocks and compresses data transmitted to and from the programmable terminals. Blocking, but not compression is supported from the nonprogrammable RJP terminals. The RJPGET routine handles decompression and deblocking of data and the RJPPUT routine services all the output requirements.

Requirements For Remote RJP Terminals

ASP RJP is designed to support the HASP II remote terminal packages. These are presently available for 1130, System/360, System/3, 2922, and Model 20 operating in Binary Synchronous Communication transmission mode. The HASP II STR terminal packages are not supported by ASP RJP.

Care must be taken in generating the HASP II remote terminal packages to assure compatibility in specifications of devices and buffer sizes with those specified at ASP Initialization time.

The remote workstation terminal name must be exactly five characters in length. It can be composed of any combination of valid alphanumeric characters. Device names on the terminals have a fixed format. The first five characters are the terminal name, followed by a PRn for a printer, PUn for a punch, and RDn for a reader. When a terminal is defined with a remote console, the user is not required to specify it with a unique device name. Examples of device names are:

```
TERM1PR1     PRINTER 1 ON TERMINAL TERM1

T0003PR2     PRINTER 2 ON TERMINAL T0003

T2780PU1     PUNCH 1 ON TERMINAL T2780

CALIFRD2     READER 2 ON TERMINAL CALIF
```

Devices supported on the various terminals supported are described in the ASP General Information Manual (GH20-1173).

NETWORK JOB PROCESSING (NJP)

Network Job Processing permits two or more ASP Support Processors to schedule and route ASP jobs from one Support Processor to another via communication lines.

ASP NJP support uses the OS Basic Telecommunications Access Method (BTAM) for all input/output between NJP ASP Support Processor terminals. Data transmission is in an EBCDIC data mode with full transparency.

NJP supports only dedicated lines. The lines are opened by the ASP operator and are immediately placed into listen mode, waiting to receive data. The operator may close a line only when it is in the listen mode. Canceling a line when it is active cancels only the job active on the line.

Transmission of a job via NJP is accomplished by transmitting all ASP control blocks and data sets to the remote ASP system. The first transmission sent to an NJP terminal is an 18-byte record containing initial information.

The initial write sequence, initiating communication, is a write initial for 18 characters:

SOH-(8-character jobname)-DLE-STX-INISH-DLE-ETB

The first character, the SOH, is required to be the first character of every transmittal block. It is followed in every transmittal block by the jobname of the job being transmitted. The next two characters, the data-link-escape (DLE) and start-of-text (STX), are also required in every transmittal block. The next portion begins the data area and in the initial transmission block, the word INISH is used to signal the first transmission. It is followed by the characters X'1002' (DLE-ETB), which are used as an end-of-block terminator for all transmissions by NJP.

All succeeding transmissions are of the form:

SOH-(8-character jobname)-DLE-STX-data-DLE-ETB

The data area varies with every transmission, so the writes are done for a count of the data area size +13 bytes.

At the receiving end, the initial read is for 20 bytes, and all succeeding reads are for the size of the transmittal block.

Succeeding transmissions begin by transmitting all of the single-record file belonging to the job being transmitted. Each single-record file is moved into a transmittal buffer, and the buffer is transmitted. The transmittal buffer size must be greater than the ASP buffer size by at least 13 bytes. For example, if the ASP buffer size is 1000 bytes, the length of the transmittal blocks must be at least 1013 bytes. The ASP data sets in the JDS single-record file are then transmitted. Each data set is partially pressed and blocked into the transmittal buffer. Partial pressing of the data sets is accomplished by compressing blank characters when four or more blanks appear consecutively. Since ASP data sets reside in the job queue in record formats, each record is pressed if possible and then blocked into the transmittal buffer. When there is no room in the transmittal buffer for another record, it is transmitted. This continues until all data sets have been transmitted. An EOT is then transmitted, signaling the end of the job. The line sending and receiving is placed into listen status at both locations and is available to send or receive at each location.

Functional Description

The module NJP is the driver module that is loaded by CALLDRVR when the operator issues the *CALL,NJP command. A console LOGIN is performed and a signon message is issued. When the operator responds the module NJPCOMM is loaded and control is passed to it to process the operator request.

NJPCOMM scans the input message for a communication verb. The SEND verb is used to schedule jobs in the system for NJP processing at a remote NJP location. The remainder of the input message is scanned to determine the type, amount, time requirements, and priority of the jobs to be rescheduled. Each job to be rescheduled is checked for functions currently active, for current setup status, and for current RESQUEUE status. When it is determined a job can be processed, the JCT and JDAB for the job are rebuilt to include the NJP Scheduler Elements. The old JCT entry is deleted and the new rebuilt JCT is added to the Job Control Tables. When the entire request is satisfied, control is returned to the module NJP.

The OPEN verb is used to open the NJP communication lines. The input message is scanned to determine which lines are to be opened. A job called NJPOPEN is created for each line to be opened. The NJPOPEN job has two Scheduler Elements, NJPOPEN and PURGE. JDAB and JCT are created and the line ddname is placed into the JDAB parameter buffer for the NJPOPEN Scheduler Element. The job is added to the Job Control Tables at priority 15. Control is then returned to the NJP module.

The Q verb is used to request job queue status from a remote NJP terminal. The input message is scanned to determine the terminal location and what information is to be requested. A job called NJPQUEUE is created by building a JCT and JDAB. The job has two Scheduler Elements, NJPIO and PURGE. The input parameters are placed into the parameter buffer for the NJPIO Scheduler Element. This job, called NJPQUEUE, is added to the Job Control Tables at priority 15. Control is returned to the NJP module.

The validity of all parameters is checked, and appropriate messages are issued for all bad parameters. The success or failure of each input message is indicated by an appropriate console message.

The NJPOPEN module performs the opening and closing of all communications lines for Network Job Processing. The module handles all operator communication for canceling or restarting lines.

A CSECT, NJODATA is created for each open line. This NJODATA contains the BTAM control blocks, the DCB, the DECB and the LERB. NJPOPEN does a console LOGIN to provide operator communication with each active line. The line is opened and placed into listen status. An AWAIT macro is issued to wait for activity on the line.

When data is received on the line, NJPOPEN uses the macro-instruction INTERCOM to call the module NJPIO to process all activity on the line. The NJPOPEN module then awaits for either an operator message or for the activity on the line to complete. When an operator request to close the line is received, NJPOPEN issues a BTAM CLOSE and returns to JSS.

The NJPIO module first determines whether it is to receive or send data. When data is to be sent, the JCT entry for the job to be transmitted is obtained from the JCT and saved in the NJPDATA CSECT for the line. The parameter buffer for the NJPIO Scheduler Element is read to obtain the terminal destination. The NJP terminal table is scanned to verify that JSS has scheduled the correct line connecting the two terminals. A reschedule buffer is built if the wrong line has been assigned and control is returned to JSS.

The NJODATA CSECT pointer for the line is obtained from the terminal table. An AGETMAIN is issued for a transmittal buffer. A load module, NJPDJ, is loaded. This module places all data to be transmitted into the transmittal buffer. A HALT I/O is then issued to the line to halt the current read initial listen status.

A write initial is then issued to initiate transmission on the line. Control then passes alternately between NJPDJ and NJPIO. NJPDJ fills the transmittal buffer and NJPIO issues write continues until all data is transmitted. An EOT is issued and the NJPOPEN module is posted to reestablish the listen status. The transmittal buffer is freed via APUTMAIN, the NJPDJ module is deleted, and control is returned to JSS.

When data is to be received, the parameter buffer for the NJPIO Scheduler Element is read. The message in the parameter buffer is the message built by the INTERCOM macro-instruction in NJPOPEN when the read initial was satisfied. The terminal table is scanned to obtain the line entry and the NJODATA CSECT pointer. A transmittal buffer is obtained via AGETMAIN. The initial read information is examined and the NJPDJ

module is ALOADed. A read continue is issued, and control passes alternately between NJPDJ and NJPIO until an EOT is received. The NJPOPEN module is then posted to reestablish listen status. The transmittal buffer is returned and the module NJPDJ is deleted. Control is then returned to JSS.

The NJPQUEUE job, when scheduled by JSS via the NJPIO Scheduler Element, processes through NJPIO in the manner described above for receiving and sending queue information. Instead of the NJPDJ module being loaded, the module NJPINQ is loaded to place the data into the transmittal buffer.

The NJPDJ module places into the transmittal buffer, all ASP control blocks and all ASP data sets for the job being transmitted to the NJP terminal. In receive mode, it also writes the received ASP control blocks and data sets to the ASP job queue.

NJP Terminal Compatabilities

The NJP network must have the following characteristics:

- ASP buffer size must be the same for all Support Processors connected via an NJP line
- The transmittal block size must be the same for all Support Processors connected via the NJP line
- The transmittal block size must be greater than the ASP buffer size by at least 13 bytes
- Jobs transmitted to remote ASP systems may contain setup instructions provided that any requested volumes needed are at the remote location. This information about requested volumes should be known before an attempt is made to transmit such a job.
- The DSP Dictionary entries in RESPARAM must have matching DSP numbers in all systems

Terminal Transmittal Block Size

Transmittal block size must be the same at the receiving and sending locations, and must be greater than the ASP buffer size by at least 13 bytes. Size consideration is based upon line speed and ASP transient area storage. Larger sizes are desirable for higher speed lines. Block size can be changed at ASP coldstart time.

NJP Restriction

When transmitting jobs via NJP, which uses BTAM, the 270X being used must have power on. A permanent lockout condition will occur if transmission is done to a 270X that is offline or powered down.

Error Recovery

Standard OS BTAM error recovery procedures are used. BTAM attempts to send or receive seven times before posting ASP with a permanent error. The operator has at that time an option of restarting or canceling the line. If he chooses to restart the line and this results in a permanent error with no new data transmitted, NJP automatically cancels all activity on the line. The job that was being transmitted goes back into the queue and is scheduled again. Canceling the line places it into

listen mode, and any active job on the line at that time goes back into the job queue to await scheduling at a later time.

If an error occurs on the initial write sequence, NJP attempts to send three times at one-minute intervals. If this fails, the line is varied offline to all jobs that could schedule to use the line, assuming that the receiving location is not currently open to receive data. The operator may initiate activity for the line again by varying the line online.

INTERNAL JOB PROCESSING (IJP)

The IJP routines fall into two distinct classes:

- ASP routines that follow all normal ASP programming and operating conventions
- The interface routine, which is "attached" from the task desiring to make use of the ASP system capabilities. This routine follows standard OS programming conventions.

The interface is accomplished by passing the data requiring ASP processing from the requesting task to ASP via the CTC adapter. The interface routine establishes communication with the ASP IJP routines via the WTO/WTOR macro . All messages to the operator go across the CTC adapter , which has been defined at SYSGEN time as the alternate console. When a WTOR message of prescribed format is received by the ASP IJP DSP from the IJP interface subtask, it performs the necessary preparation for Main Service to receive data from the user and indicates this through a reply to the user message. The data that requires processing is then written across the CTC, and the user task issues another WTOR message which indicates that all data has been sent. The ASP IJP routine then replies that it has received the data and proceeds to input the data to the ASP DSP that was requested to process it. The ASP side of the IJP interface is started by calling the IJP DSP for the logical Main Processor that will be submitting data. The user side of the interface is started by "attaching" the IJPWTR interface subtask with appropriate parameters and JCL statements describing the data to be processed. The requirement is:

> The OS assembler macro ATTACH is used to create a subtask which may execute independently of the originating or ATTACHing task. A parameter list is passed via the ATTACH macro ; the list contains:
>
> - The address (first word in the parameter list) of the six-character volume serial of the disk or tape containing the data set to be processed
> - The address (second word in the parameter list) of the 44-character name of the data set to be processed
> - The address (third word in the parameter list) of the eight-character member name if the data set is partitioned; otherwise, the address of an area containing eight blanks
> - The address (fourth word of the parameter list) of the eight-character name of the ASP DSP that is to process the data set
> - The address (fifth word of the parameter list) of a four-character processing options indicator area. The first of these four characters indicates that the parameters are to be added to a checkpoint data set (C), after which the ATTACHing task may resume execution while the subtask processes the

parameters from the checkpoint data set; or (S) indicates serial processing, where the ATTACHing task waits for all subtask services to complete before continuing. The second character indicates that the input is on direct access (D) or on tape (T). The third character, if (S), indicates that a direct access data set to be sent to ASP is to be scratched after being sent to ASP. The fourth character is presently unused. The example options are C, D, blank, blank (checkpoint, direct access, no-scratch, null).

- The address (sixth word of the parameter list) of a fullword ECB, which is posted by the subtask at the end of serial processing or checkpointing of parameters. (Initially hexadecimal zeros.)

- The address (seventh word of the parameter list) of an 8-byte word aligned work area (initially hexadecimal zero) used by the IJPWTR subtask to determine whether to continue processing or to cease processing and return. The ATTACHing task indicates cease processing to the IJPWTR subtask by moving hexadecimal '0F' to the first byte of the work area. The ATTACHing task may then determine when the IJPWTR subtask has completed by examining the TCBLTC field in the ATTACHing task's TCB. When this field is zero, no subtasks are outstanding.

For further information refer to the ASP Application Programmer's Manual.

Figure 22 provides an overview of the IJP/ASP interface.

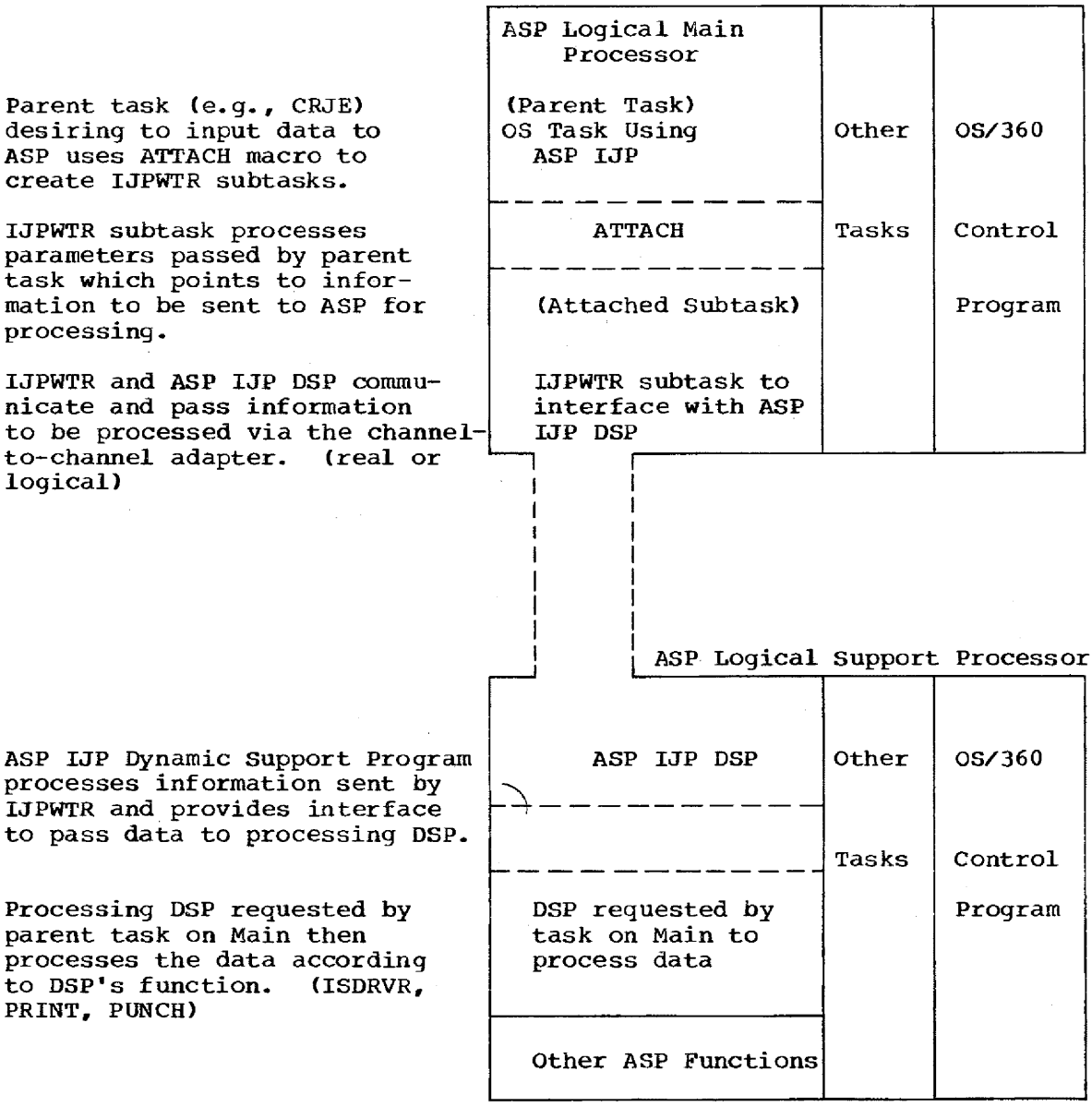


Figure 22. IJP ASP Overview.

IJP is invoked via an operator *CALL command. The IJP driver module is loaded and it in turn loads and calls the necessary modules to interface with Main Service to handle data from the CTC and to interface with the user-requested DSP. IJPINISH is loaded, called and deleted to establish operator communication.

IJP issues a RECEIVE macro for the expected WTO/WTOR message from the IJPWTR module, and IJP then waits for the IJPWTR message or a cancel from the operator. If the RECEIVE is satisfied, the IJPSTART module is loaded, called, and deleted to establish the Main Service interface required to accept the input data from the CTC. When all the data has

been received and another WTO/WTOR message is received indicating end-of-data, the IJPEND module is loaded, called, and deleted to remove the IJP interface to Main Service and to give the JDS containing the data set across the CTC to the user-requested ASP DSP. The routine then AWAITs further processing messages or a cancel from the operator, at which time a return is made to JSS.

ASP CREATED DATA SETS (ACDS - TSO SUPPORT)

Background jobs from a terminal, submitted to TSO for execution on a Main Processor, are temporarily placed in the OS job queue. The job is then submitted to ASP via the ASUBMIT function of MAINTASK and ASP IJP DSP on the support processor. The job is read in via the ASP interface, ISDRVR , and placed in the ASP queue for scheduling. At this point the job will be scheduled per the TSO user's request or by system default values.

The TSO job will complete the normal Input Service (ISDRVR), ASP R/I, MAIN, PRINT, PUNCH, and PURGE scheduler elements. (All ASP control cards may be included in TSO submitted jobs.)

The TSO terminal user may direct output data sets of his job to any ASP defined local or remote printer via the //*FORMAT card. In addition, the TSO user may direct output data sets to his own or some other TSO terminal via a //*FORMAT card with the AC parameter specified.

The ACDS DSP is a callable DSP invoked as a result of Input Service processing of a //*FORMAT AC ASP control card or a SYSOUT data set DD card, the class of which is specified in an initialization deck SYSOUT, TYPE=TSO card. A scheduler element, AC, is inserted and executed immediately following the Main Service SE. The ACDS DSP's function is to provide the necessary interfaces in the ASP system for the ADSGEN subtask of MAINTASK to read user-supplied job output across the CTC adapter to the Main Processor where ADSGEN resides. This is accomplished in a manner similar to the IJP DSP interface with the IJPWTR job subtask or the ASUBMIT subtask of MAINTASK. Essentially this interface for either IJP or ACDS involves the construction of a dummy job and related ASP control blocks in order to allow ASP Main Service CTC I/O routines to be used for data transfer on the CTC.

The ACDS DSP is invoked via the //*PROCESS ASP control card or by inclusion of //*FORMAT AC ASP control cards in a job submitted through ASP IJP processing or through specification of SYSOUT = a class of type = TSO. When the ASP ISFORMAT Input Service module encounters //*FORMAT AC cards it constructs a parameter buffer for the ACDS DSP. This parameter buffer contains the ddname of the ASP JDS entry which the user desires to send across the CTC to the ADSGEN module for processing.

The DDNAME parameter specifies the ddname of the ASP JDS entry to be ACDS/ADSGEN processed. The optional ERDEST parameter indicates the name of an ASP-defined local or remote printer where the data specified by the DDNAME parameter is to be printed in the event of unrecoverable errors encountered by ADSGEN. The USER parameter specifies the user identification of the TSO user who is to receive notification of the creation of a TSO EDIT command-accessible data set by the ADSGEN module. The USER parameter is mandatory for non-IJP submitted jobs (that is, through local or RJP readers). For IJP submitted jobs it may be used to indicate that a user other than the job submitter is to receive the data.

The PRINT parameter indicates to the ACDS DSP whether or not the DDNAME= output is to be printed after successful creation of the TSO EDIT accessible data set. If this parameter is not specified or PRINT=NO is specified, the ACDS DSP will ensure that the effected data

set is not processed by ASP Print Service after completion of ACDS processing. If PRINT=YES is specified this action is not taken.

The output of ACDS processing is a TSO EDIT command-accessible data set. This data set is constructed by the ADSGEN subtask of MAINTASK which interacts with the ACDS DSP to effect the transfer of the user-specified job output across the CTC adapter to the ASP Main Processor where ADSGEN resides.

Further information and/or requirements for TSO/ASP may be obtained as follows:

- DEVICE initialization card required for IJP/TSO:
  See ASP System Programmer's Manual - DEVICE card

- JCL requirements for AOUTPUT/ADSGEN/ASUBMIT subtasks of MAINTASK:
  See ASP System Programmer's Manual - MAINTASK Execution.

- Use of IJP DSP (necessary for TSO JOB submission):
  See ASP Console Operator's Manual - IJP DSP

- Operator communication with MAINTASK:
  See ASP Console Operator's Manual - MAINTASK

- Operator communication with ACDS DSP:
  See ASP Console Operator's Manual - ASP-Created Data Sets

- ASP control cards:
  See ASP Application Programmer's Manual

- Using ASP/TSO support:
  See Application Programmer's Manual - Using ASP/TSO support

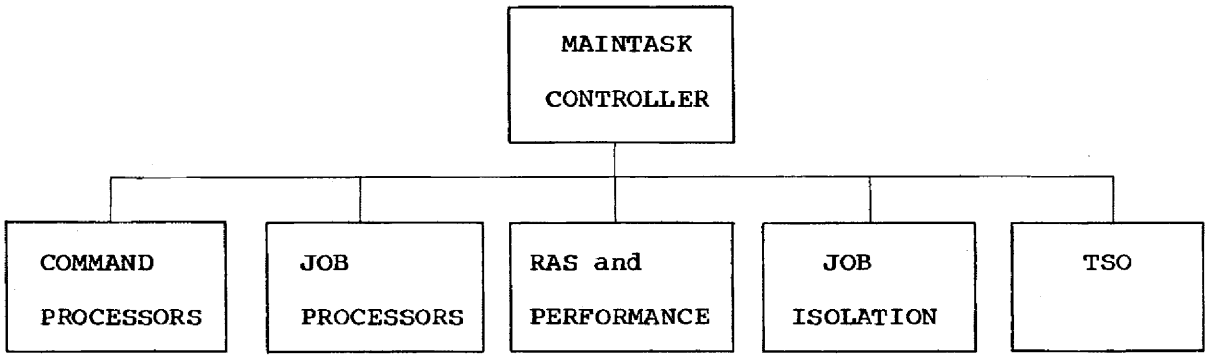
MAINTASK

Those functions which were required on an ASP Main Processor, in previous versions, were implemented through changes to existing OS modules or by extending the function of OS modules. These modifications were performed during the pre- and post-generation steps. With ASP Version 3, many of these required functions, as well as those required for TSO, Hot Jobs, RAS, R/I, and performance feature, have been brought together under control of a single monitor which operates as a system task on the ASP real Main or local Main Processor. This task, called MAINTASK, will monitor and control job processing through the use of the OS MODIFY command and the OS timer. Each time a MAINTASK service is required, the Support Processor will transmit a MODIFY command over the channel-to-channel to MAINTASK. MAINTASK will then analyze the text of the MODIFY command and route the request to the appropriate subtask for action.

Since MAINTASK is an OS system task, its name must be a member of the system task table. This is accomplished by a post-generation step of the ASP generation. A sample JCL procedure is included in Chapter 5 in this manual and will be added to the system procedure library by another post-generation step of ASP generation.

The MAINTASK region is made up of the MAINTASK controls and its subtasks. Some subtasks are required for the normal operation of ASP Version 3 while others are only required if the special support such as TSO or Hot Job is being used. The RAS and performance features are also optional, but their use is recommended.

The diagram below describes the relationship of the subtasks to the MAINTASK controller.



Maintask Controller (MAINTASK)

This module is the command router for the MAINTASK region and will be started as a system task by ASP initialization. The S MT command is entered into the ASP initialization deck as an IPL text card for a real Main Processor. For a local Main Processor MAINTASK is started automatically. The format and options of this IPL text card are covered in a later section of this manual. Once started the MAINTASK Controller will load those other modules which were called for as parameters of the START command. The region size required will vary according to those features requested. The region size is also a possible parameter of the start command.

If no override parameters were specified, the PARM= and REGION= specified in the MT (MAINTASK) procedure will be used. If no PARM= or REGION= is in the start command or procedure only those subtasks required for normal ASP execution will be loaded into a default region size.

Careful attention should be given to creating the MT procedure and the IPL text card for starting MAINTASK. This is described in detail in a later section of this manual.

Once initialized the MAINTASK controller will receive the MODIFY command from the Support Processor and pass it on to the task which will process the request.

Command Processor (ASPVER, ASPLOC, ASPFENCE)

These are three modules which make up the command processors. These three modules are required for the normal operation of ASP Version 3. The commands which are processed by these modules are:

Module	Command(s)
ASPVER	VERIFY and ISOLATE
ASPLOC	LOCATE
ASPFENCE	SIZE and FENCE

These commands are sent by the Support Processor to a Main Processor during setup and job scheduling.

ASPLOC: LOCATE command processor receives requests from ASP to locate cataloged data sets. Uses OS catalog management to locate the data sets and returns the name and type of volume containing the specified data set.

ASPVER: Performs volume mount verification and unit isolation function for ASP Main Device Scheduling. Issues messages pertaining to mount status of ASP SETUP devices.

ASPFENCE: The SIZE command is used by Main Service to determine the size of the largest region currently available on Main. A job is then selected from the resident job queue that will fit in this region. The response from a SIZE command has the message ID MTSZ001 and is described in the ASP Messages and Codes Manual. This message will be displayed on the MDEST console if a size display is specified on the SELECT card or the operator issued the MODIFY command, *f main-name, D=SIZE.

The format of the SIZE command is:

```
F MT,SIZE[,OS-job-class]
```

where OS-class is the OS Job Class assigned to Main Storage Fence. If OS-class is omitted, the largest region available in the OS dynamic area will be given. If OS-class is used, the largest region available within the corresponding Main Storage Fence will be given.

The FENCE command is used by Main Service to reserve Main Storage for an ASP Job Class Group (see Main Storage Fencing description in Chapter 4). Three types of FENCE commands are used; the Fence build, the Fence reset, and the Fence list. The responses to these commands have the message ID MTFNxxx and are described in the ASP Messages and Codes Manual.

The format of the FENCE command is:

```
F MT,FENCE T=task-name,{nnnnnk|RESET|LIST}[,OS-job-class]
```

Examples:

To build a 500K fence for OS Class A:

```
F MT,FENCE 500K,A
```

To determine the largest available region in this fence:

```
F MT,SIZE A
```

To reset that fence:

```
F MT,FENCE RESET,A
```

To fence a system task such as TSO:

```
F MT,FENCE T=TSO,200K
```

To reset that fence:

```
F MT,FENCE T=TSO,RESET
```

To list the status of all fences:

```
F MT,FENCE LIST
```

Job Processors (ASPWRITR, ASPQALL, ASPQRDR)

An ASPWRITR is included as a subtask of MAINTASK and eliminates the need for the OS writer. This writer will be started when MAINTASK is initialized and will wait until it receives a MODIFY command requesting

a job (SMB's) be dequeued from a particular job output class and sent to the Support Processor to be printed with the job's output. Once a job has been transmitted the "WTR WAITING FOR WORK" message will be sent to the Support Processor. This will be the signal that a job has been dequeued and another job's SMB's can be requested via a MODIFY command.

In addition to the ASPWRITR, there are two other modules required for job processing. The first (ASPQALL) is used by the Reader/Interpreter DSP and allocates space and actual TTR's on the OS job queue on the Main Processor for the queue records which are built by the Reader/Interpreter DSP on the support. The ASPQRDR module is the queue record reader which reads the preformatted job queue records from the Support Processor. Once this reader has placed all the control blocks onto the OS job queue, the complete job will be enqueued and the job will be immediately scheduled for execution. The queue record reader replaces the normal OS Reader on a real Main or local Main Processor.

These three modules are required for normal ASP Version 3 operations. These modules replace the standard OS modules at a substantial reduction in the amount of main storage required.

RAS and Performance (ASPCTCM, DYNDISP)

Two modules operate asynchronously, from MAINTASK and are dispatched by time intervals rather than MODIFY commands. These modules are designed to increase the reliability of an ASP complex by keeping a real Main Processor active, even if the Support Processor has terminated, and improve the throughput of jobs operating on any Main Processor.

A Channel-to-Channel Adapter Monitor (ASPCTCM) module is provided as an optional feature, but its use is recommended. This module will monitor the adapter, at the interval selected by the user at initialization, to ensure it is functioning. If it is determined that the adapter is permanently busy, which means the main has tried to initiate a request but the support does not respond, the request will be halted thereby freeing that channel for other activity. The console, which is also the adapter, will be switched back to the primary console on the main. This will allow those jobs which are isolated from the adapter, such as Hot Jobs, to continue execution. When the support machine is restarted the CTC monitor will recognize the restart condition and again switch the primary console back to the adapter. This will allow the support machine to again communicate with the main.

The ASPCTCM module is also used to normally detach a real main processor and return it to OS standalone status. This function is performed for real main systems only and is activated when a P MT command is received by MAINTASK. Before MAINTASK terminates, it posts the CTC monitor to perform a console switch from the CTC to the primary OS console. Logically this should only be invoked by the operator when all ASP scheduled jobs, on that main, have been quiesced and the main has been varied offline to ASP.

Caution: When using the CTC Monitor and it becomes necessary to hit STOP on the support machine, the operator must hit STOP on the main also.

Also ASP dumps on support, written to the printer, can cause ASP to be nondispatchable long enough to create a time-out of the CTC MONITOR and the temporary loss of a real main processor. It is advisable when in a testing mode to run without the CTC MONITOR and when in production to write ASP dumps to tape or disk.

The throughput of ASP scheduled jobs on a Main Processor can be increased by another optional module, the Dynamic Dispatcher (DYNDISP). This module is also time driven with the interval set each time the module is initialized. Each time the Dynamic Dispatcher gets control it will check the time used by each job since the last time interval and reorder the TCB chain in an attempt to give a higher priority to I/O-bound jobs to improve system throughput. A MAINTASK restriction is that Dynamic Dispatching is mutually exclusive with OS Time Slicing for the same priority levels.

Job Isolation (AOUTPUT)

This feature allows installations to isolate important high-priority jobs from failures on the Support system. By executing jobs defined as " Hot Jobs " and assigning the SYSOUT data sets to classes not intercepted by ASP (that is, output classes not defined in the SYSOUT initialization card), the jobs can continue to execute in the event of a Support failure. The subtask AOUTPUT may then be directed to retrieve these data sets via the operator MODIFY MT command, sending them, via the IJP DSP on Support, to the proper DSP; Input, Print or Punch Service.

TSO Support (ADSGEN1, ADSGEN, ASUBMIT)

ASP TSO support extends the advantages of ASP multiprocessing to the TSO user. This support encompasses the uniprocessor environment as well as the multisystem environment.

Main features are:

1. To allow the TSO terminal user to submit jobs to the centralized ASP job queue for processing.

2. To allow a TSO submitted job to run on any ASP Main Processor in the ASP complex for which it is qualified.

3. To allow the TSO user to inquire as to STATUS or cancel his job.

4. To allow the TSO terminal user to obtain selected output back at his terminal even though his job ran on another processor in the ASP complex.

5. To allow the continuing execution of either TSO or ASP in the event of failure of either.

Background jobs submitted to TSO for execution by a terminal user will be temporarily placed in the OS job queue data set. The job will then be read by an ASP interface routine (ASUBMIT) and sent to the ASP IJP DSP for execution scheduling. The processing of jobs submitted to TSO and scheduled by ASP can be transparent to the user or he may take advantage of ASP's unique capabilities.

Subsequent to execution, user-specified data sets are read by an ASP DSP (ACDS) and sent to an interface routine (ADSGEN). The terminal user is notified and he may then access the data set using standard TSO commands.

CHAPTER 5. ASP SYSTEM INITIALIZATION

The initialization of ASP defines the system configuration and processing options. The initialization process is controlled by OS JCL and ASP initialization control cards. This process, being controlled by user-supplied control cards, allows the greatest amount of flexibility to the user to create or modify the ASP system operation. The initialization options can be quickly and easily changed without having to make alterations to the established OS system. Initialization consists of:

- Loading the resident portion of the ASP system
- Loading optional modules
- Formatting the ASP queue devices if necessary
- Allocating disk space for ASP data sets
- Defining job scheduling algorithms
- Defining installation standards
- Defining console message routing
- Defining the number of Main Processors and the operating characteristics of each
- Establishing Support Processor device allocation tables
- Starting of procedures on selected Main Processors

Initialization can be broken into three sections:

1. Starting ASP as a problem program to OS
2. Defining the ASP configuration and processing options
3. IPLing of each Main Processor

OS CONTROL CARDS FOR ASP EXECUTION

ASP executes as a problem program with a nonzero protect key in supervisor state. When required ASP will get a zero protect key for short intervals. To initiate the execution of the ASP system normal OS JCL is required, for which an example and explanation follows:

```
//EXASP    JOB 836,SUPPORT,REGION=250K,PRTY=13
//JOBLIB   DD  DSN=ASPMNT,VOL=SER=111111,UNIT=2314,DISP=OLD
//         DD  DSNAME=ASP,UNIT=2314,DISP=OLD,VOLUME=SER=111111
//STEP     EXEC PGM=ASPNUC,TIME=1440
//CHKPNT   DD  DSNAME=ASPCKPNT,UNIT=2314,SPACE=(CYL,(1)),DISP=OLD,
               VOLUME=SER=111111
//ASPQ1    DD  DSNAME=ASPIO,UNIT=3330,SPACE=(CYL,(403)),DISP=OLD,
               VOLUME=SER=ASPQ1
//ASPQ2    DD  DSNAME=ASPIO,UNIT=3330,SPACE=(CYL,(403)),DISP=OLD,
               VOLUME=SER=ASPQ2
//ASPQ3    DD  DSNAME=ASPIO,UNIT=3330,SPACE=(CYL,(403)),DISP=OLD,
               VOLUME=SER=ASPQ3
//ASPOUT   DD  SYSOUT=A
```

```
//ASPABEND DD  SYSOUT=A,DCB=BLKSIZE=764
//ASPSNAP  DD  SYSOUT=A,DCB=BLKSIZE=764
//SYSABEND DD  SYSOUT=A
//ASPSADMP DD  DSN=ASP.DUMP,DISP=OLD
//NJPL1    DD  UNIT=053
//IEFRDER  DD  DUMMY
//IEFPDSI  DD  DSN=SYS1.PROCLIB,DISP=SHR
//IEFDATA  DD  DUMMY
//ASPDRDS  DD  DSN=DISKRD,DISP=SHR,VOL=SER=222222,UNIT=SYSDA
//ASPIN    DD  DATA
```

In the JCL example the JOBLIB DD card has two concatenated data sets called ASPMNT and ASP. The DSN=ASP contains the ASP programs and modules as received from the IBM Program Information Department (PID), and DSN=ASPMNT could contain maintenance. For further information on ASP maintenance refer to the table of contents.

ASPNUC is the program name to be executed and TIME=1440 negates OS job step timing.

When the initialization process has been completed, a checkpoint is written on the specified ASPCKPNT device. This checkpoint contains the information that is necessary to restart the system should a failure occur. This record is repetitively updated during execution to reflect the current status of the system. DDNAME CHKPNT defines the device and space allocation. The SPACE parameter must indicate 1 cylinder (CYL).

ASPQ1, ASPQ2, ASPQ3 are the DD names used for examples to define the ASP queue devices. The ASP queue devices may be 2314 or 3330. The DD names selected by the user are to be referenced in the ASP initialization FORMAT and TRACK control cards. The space must be specified as cylinders, SPACE=CYL.

The // ASPOUT DD card designates the data set for the printing of the COLDSTART deck by initialization processing.

The // ASPABEND DD card designates the data set for the snap of the OS/ASP region by ASPABEND when the DUMP parameter is used on the OPTIONS card in the ASP initialization deck.

The // ASPSNAP DD card could be allocated to a device (such as a printer), which will allow the data to be printed during or immediately following ASP initialization. It is used by initialization when the option ANALYZE=YES is used on the RESTART card to SNAP possible illegal control blocks due to user modifications. It is also used for the output from the Dump Core DSP.

The DCB=BLKSIZE=764 parameter is required on both the //ASPSNAP and //ASPABEND DD cards.

The //ASPABEND, //SYSABEND, and //ASPSNAP DD cards may be directed to different SYSOUT classes or may be assigned to one unit through the use of the UNIT=AFF= designation in the JCL.

The //ASPSADMP DD card designates the data set for core-image dumps. If a direct access device, the data set must be preallocated and initialized the same as the OS SYS1.DUMP data set, but must not be SYS1.DUMP. A non-labeled tape data set may be used also. Refer to OS Service Aids Manual under IMDPRDMP for allocation and initialization requirements.

If Network Job Processing (NJP) is to be used in the ASP system, the associated line connecting the terminals has to be defined. In the example //NJPL1 defines a line for NJP usage. The DD name used will be referenced by the NJPTERM card in the initialization deck.

The IEFRDER, IEFPDSI and IEFDATA DD cards are required and are used by the ASP R/I DSP. If the IEFPDSI DD card is left out initialization will be terminated and an error message provided.

// ASPDRDS describes a previously created partitioned data set that contains jobs to be input via the Disk Reader facility.

//ASPIN DD DATA defines the data set which contains the ASP initialization deck. This data set normally is part of the input stream; however, it can reside on a system input device, a magnetic tape volume, or a direct access volume. The data set must be unblocked and fixed format. After reading the END,IPL card in the initialization deck, ASP will leave the tape or the card reader positioned at the record or card following. ASP will not scan to an end-of-file.

INITIALIZATION CONTROL CARDS

The ASP system configuration and processing options are specified by means of initialization control cards, which appear in the input stream when ASP is loaded. Because many of the options will affect the overall performance of the system, initialization control cards should be specified by the installation's system programming staff. The selection and specification of the options should be given considerable thought because of the impact on system performance. The deck that is submitted will be validity checked during the initialization process.

The initialization deck can be used for a system COLDSTART or RESTART . A COLDSTART is signified by placing the COLDSTART initialization control card as the first card in the initialization deck. In conjunction with the COLD START card the FORMAT card defines the ASP queue devices that are to be formatted. Once the ASP queue devices have been formatted future COLDSTARTs should use the TRACK control card in place of the FORMAT card unless the buffer size is changed. If the buffer size is changed a FORMAT card must be used to reformat the ASP queue. The TRACK card identifies the queue devices previously formatted. The jobs existing in the queues will be ignored during a cold start using TRACK cards.

A RESTART can be used on subsequent ASP initializations to reference the last checkpoint entry made and execute the jobs remaining in the queues. The TRACK control card is to be used in conjunction with the RESTART card. The RESTART and TRACK cards are to replace the COLDSTART and FORMAT cards for a restart. If the COLDSTART is left out of the deck, the initialization process will default to a restart.

When restarting the ASP system, the ASP RESTART deck must specify BUFFER card options (except for AMOUNT and IOBS, which may change) which are identical to those used during the previous cold start when the queue was formatted. The same parameters should be used on the TRACK cards as were used on the FORMAT cards for the COLDSTART of the current ASP execution. All other ASP control cards may be varied between restarts to alter the configuration or execution options.

ASP initialization control cards are identified by a keyword starting in column 1 of the card. When required, continuation cards may be used by punching a nonblank character in column 72 of the first card and continuing the data in column 1 of the next card. A keyword parameter may not be split between two cards; each keyword and its parameters must be complete in one card. Except where noted, all parameters must be spelled out completely. The keywords and parameters may not contain embedded blanks. If a keyword is used more than once on an initialization card, the last one encountered will be used unless the error is severe enough to cause a diagnostic message. There are some instances, noted in the initialization statement descriptions, when a

duplicate keyword is treated as an operand continuation of the previous matching keyword. The first card of an ASP initialization control card deck identifies the type of initialization, either COLDSTART or RESTART. The control card deck terminates with the ENDINISH card. All other control cards for initialization are placed between these cards. If a given control card is not applicable (such as the NJPTERM card in an installation without remote terminal equipment), or if the use of default options results in a null parameter field, the control card should be omitted. Comments cards may appear at any point in the ASP initialization deck. Comments cards are identified by placing an asterisk in column 1. Continuations of a comment card are not allowed.

The ASP IPL deck must immediately follow the ASP initialization control card deck. It controls IPL of the control program on each Main Processor in the system and may contain the operator IPL dialogue.

The ASP system operation is defined by use of initialization cards, some cards are required and some have sequence dependencies. The following tables list the initialization cards that are required and those cards that are sequence dependent. The final list is an alphabetical list of all of the cards.

Required Control Cards

BUFFER
CONSOLE
DEVICE
ENDASPIO
ENDINISH
FORMAT|TRACK
STANDARDS
SYSOUT

Sequence Dependent Control Cards

The following required or optional cards must be the first cards in the deck. The first card must be COLDSTART or RESTART if supplied and the last must be ENDASPIO. The sequence of cards between these two cards is not critical. These cards are required to be placed first for initialization to determine required storage needs. The inclusion of any initialization control card that is not in this grouping will be flagged as an error.

[COLDSTART|RESTART]	optional
BADTRACK	optional
BUFFER	
TRACK|FORMAT	
OPTIONS	optional
ASPCORE	optional
ENDASPIO	

Alphabetical Listing

ACCOUNT
ASPCORE
BADTRACK
BUFFER
CLASS
COLDSTART
CONSOLE
DEADLINE
DEVICE
ENDASPIO
ENDINISH
FORMAT

```
GROUP
IPL
MAINPROC
NJPTERM
OPTIONS
PFK
PRINTER
RESCTLBK
RESIDENT
RESTART
RI
RIDATSTN
RIPARM
RJPLINE
RJPTERM
SELECT
SETNAME
SETPARAM
STANDARDS
SYSOUT
TRACK
```

ACCOUNT

The ACCOUNT card gives the specifications to be used by the ASP PURGE DSP in producing job accounting information by PURGE. If the ACCOUNT card is omitted, no accounting cards will be produced. The keyword parameters defined are:

```
ACCOUNT[,CARDS={NO|YES}]
       [,DETAIL={NO|YES}]
       [,PUNCH={(PUN)|device}]
```

CARDS ={YES|NO}

Master cards:

YES — Master accounting cards will be produced for all jobs processed.

NO — No accounting will be done; ensuing parameters are ignored.

DETAIL ={YES|NO}

Detail cards:

YES — A detail card will be produced for each Scheduler Element executed in conjunction with each job processed.

NO — No detail cards will be punched.

PUNCH ={(PUN)|device}

The output device on which accounting output is to be produced. The format of PUNCH is identical to that described in the ASP Operator's Manual for the IN= and OUT= parameters of callable DSPs; that is, it may be given as a specific device by device-name or device-address, or generalized by type as (PUN) or (PUN2540P), or restricted to a group as (PUN, group) or (,group). If a specific device is named, all accounting output will be punched on the stated device; this method will cause a delay in purging a job if another function is in control of that punch as a job is terminating. If the parameter is generally specified, output will occur on the first appropriate punch available at the termination of each processed job. ASP account cards are stacker selected to the center pocket of a 2540 and therefore do not get intermixed with job output. If a generalized specification is given, a merge operation for all punches will probably be required to put the accounting data into the proper sequence for further processing.

An example of an ACCOUNT card that will cause master and detail cards to be produced on punch PU1 is:

```
ACCOUNT,CARDS=YES,DETAIL=YES,PUNCH=PU1
```

The master and detail cards produced may be summarized and printed by using the ASP Accounting Print (ACCPR) utility program.

The format of the master card is:

Columns	Contents
1-8	Account number
9-12	ASP job number

The format of the detail card is:

Columns	Contents
1-8	Account number
9-12	ASP job number

13-20	Job name	13-20	Job name
21-25	Date job completed	21-25	Date job completed
26-35	Programmer's name	26-33	DSP name
36-40	Number of lines printed	36-40	Number of lines printed
41-44	Number of cards punched	41-44	Number of cards punched
45-48	Number of cards read	45-48	Number of cards read
49-50	Job priority		
51-58	Main Processor name		
59-60	Number of devices set up		
61	Deadline switch - on if Deadline Scheduling was used		
62-67	Job time on from Input, hhmmss	62-67	DSP time on
68-73	Job time off from Purge, hhmmss	68-73	DSP time off
74-79	Elapsed time on Main Processor	74-79	Elapsed time of DSP
80	Master/detail and CALLDSP/Input Service origin indicator	80	Master/detail and CALLDSP/Input Service origin indicator

ASPCORE

The ASPCORE card defines free storage operating minimums for scheduling work in the ASP region. The parameter definitions are:

```
ASPCORE[,MINCORE={14K|nnnK}]
       [,MARGCORE={20K|nnnK}]
       [,ASPOOL={0|0-127}
```

MINCORE specifies the remaining free storage to be reserved for current jobs. At this level no new work will be scheduled. JSS will however allow Dump Core (DC) to be scheduled.

MARGCORE specifies a marginal amount of free storage. When this level is reached, scheduling is limited to one function at a time.

ASPOOL defines the subpool ASP is to use when obtaining storage.

When the amount of free storage exceeds the MARGCORE value normal scheduling will resume. A reasonable span from MINCORE to MARGCORE is 6K bytes.

BADTRACK

The BADTRACK card inhibits the use of a disk track in an ASP direct access storage device on the Support Processor. The cylinder and track address can be obtained from the OS Input/Output Supervisor message, IEA000I, which is issued when a permanent input/output error occurs. Only one track may be specified on each BADTRACK card, and the maximum number of BADTRACK cards permitted defaults to 100, see BUFFER card. The keyword parameters are:

```
BADTRACK,DDNAME=device-name
        ,CYL=nnnn
        ,TRK=nnnn
```

DDNAME=device-name

The one- to eight-character name of the DD card that defines the space on the device containing the bad track.

CYL=nnnn

A four-digit field that specifies the cylinder containing the bad track. This field is in hexadecimal notation.

TRK=nnnn

A four-digit field that specifies the track in error. This field is also in hexadecimal notation.

The first example below shows a BADTRACK card to inhibit the use of track 5 in cylinder 102 on an IBM 2314 Disk Storage Drive.

```
BADTRACK,DDNAME=ASPQ1,CYL=0066,TRK=0005

BADTRACK,DDNAME=ASPQ1,CYL=0066,TRK=000A

BADTRACK,DDNAME=ASPQ2,CYL=000F,TRK=0010
```

BUFFER

The BUFFER card establishes the ASP buffer pool and fixes the relationship between the buffer pool and the disk track format. All parameter specifications, except for AMOUNT and IOBs, must be the same in the COLDSTART deck and its associated RESTART deck. Guidelines for selecting values are contained in Chapter 6 under ASPIO. The keyword parameters are:

```
BUFFER,BUFSIZE=nnnnn
      ,RECORDS=nn
      ,AMOUNT=nnn
      ,IOBS=nnn
     [,DASD={3330|2314}]
     [,FD={128|nnn}]
     [,MAXBTS={100|nnn}]
     [,TRACE={YES|NO}]
     [,HIARCHY={0|1}]
     [,TAT={FULL|HALF}]
```

BUFSIZE=nnnnn

A three- to five-digit field specifying the size in bytes of each buffer, as well as the size of each record on the ASP direct access storage device. The buffer size should never be smaller than 792 bytes (preferably 1020), and should be coordinated with the RECORDS parameter. Buffer size must be a multiple of 4 (fullword).

RECORDS=nn

A one- or two-digit field specifying the number of physical records per track on the ASP direct access storage device.

The following table gives the maximum buffer size for the range of possible record modes on the 3330:

Records Per Track	Buffer Size
14	804
13	876
12	960
11	1060
10	1180
9	1324
8	1508
7	1744
6	2056
5	2496
4	3156
3	4252
2	6444
1	13028

IBM 2314 Record Sizes

Records Per Track	Buffer Size
8	792
7	920
6	1092
5	1332
4	1692
3	2296
2	3520
1	7292

AMOUNT=nnn

A one- to three-digit field that specifies the number of buffers to be created for the system.

IOBS=nnn

A one- to three-digit field that specifies the maximum depth of buffering for the Support Processor.

DASD={3330|2314}

The type of direct access storage device used for the ASP queue. (Space is defined on DD cards referenced by the FORMAT or TRACK cards). All such space must be on the device type specified here, that is; 3330 or 2314.

FD ={128|nnn}

Defines the number of entries to be made available in the File Directory. A File Directory contains a four-word entry for every open data set.

MAXBTS={100|nnn}

A one- to three-digit field for limiting the number of BADTRACK cards.

TRACE={YES|NO}

Generates the ASPIO Trace Table in a free core storage area.

HIARCHY={0|1}

Indicates the number of the hierarchy (0 or 1) from which the buffer pool is to be allocated. Storage will always be obtained from hierarchy 0 when the 2361 Core Storage Unit (LCS) is switched offline.

TAT ={FULL|HALF}

Determines the incremental quantity of queue space allocated to a job. FULL specifies a full cylinder. HALF specifies half cylinder increments.

Example:

```
BUFFER,BUFSIZE=792,RECORDS=8,IOBS=3,DASD=2314,AMOUNT=90,          X
HIARCHY=0,TAT=HALF
```

CLASS

The CLASS card defines the characteristics of the ASP job classes. Up to 255 job classes can be defined. A CLASS card must define each Job Class that may appear on an OS job card or on a //*MAIN control card. If an undefined Job Class is used the job will be canceled. If a CLASS keyword is not used on the OS job card or //*MAIN card the default class name on the STANDARDS control card will be used. If a default is not supplied the class name ASPBATCH will be assigned. This class definition consists of CLASS card default values. The following keywords are available:

```
CLASS,NAME=job-class-name
     [,IORATE={MED|HIGH|LOW}]
     [,FAILURE={CANCEL|HOLD|PRINT|RESTART}]
     [,JOBSTEP={CHKPNT|NOCHKPNT}]
     [,PRTY=nn]
     [,JPRTY={ASP|JOB}]
     [,GROUP={default|job-class-group-name}]
     [,TDEPTH=depth]
     [,MDEPTH=(main-processor-name,depth,main.....)]
     [,TLIMIT=(class-name,limit,class-name,limit...)]
     [,MLIMIT=(main-proc-name,class-name,limit,class-name,limit,...)]
     [,SYSTEM={ANY|LOCAL|REAL|main-name|(main-name,main-name,...)|
              /(main-name,main-name,...)}]
```

NAME=job-class-name

One- to eight-alphameric characters specifying the name of the job class. This name corresponds to the CLASS keyword on the JOB card or //*MAIN control card. The NAME parameter must be the first parameter on the CLASS card.

The CLASS keyword may be used on the JOB card only if the CLASS name is A through O. Otherwise the CLASS keyword must be placed on the //*MAIN control card.

IORATE={LOW|HIGH|MED}

This parameter describes the I/O to CPU ratio for the jobs in this class as being Low, High, or Medium I/O. Main Service attempts to balance the mixture of jobs executing on a Main Processor based upon this IORATE . If this parameter is omitted, the relative I/O usage by jobs in this class will be assumed medium I/O. This parameter can be overriden on a job basis with the IORATE parameter on the //*MAIN control card.

FAILURE={CANCEL|HOLD|PRINT|RESTART}

Specifies the job recovery option to be used in case of system failure. CANCEL cancels the job on Main. HOLD holds the job for restart on Main. PRINT prints the job and then puts the job in hold for restart on Main. RESTART restarts the job on Main. This parameter may be overriden on a job basis with the FAILURE keyword on the //*MAIN control card. If this parameter is omitted, the FAILURE option from the STANDARDS initialization control card will be used.

JOBSTEP={CHKPNT|NOCHKPNT}

Specifies the job step checkpoint option. CHKPNT causes a checkpoint to be taken at the end of each job step on Main. This checkpoint contains the current status of all ASP data sets for the job (that is, the ASP JDS record for the job). NOCHKPNT stops the checkpoint facility. This parameter may be overriden on a job basis with the

JOBSTEP keyword on the //*MAIN control card. If this parameter is omitted, the JOBSTEP option from the STANDARDS control card will be used.

JPRTY={ASP|JOB}

If JOB is specified, the job will be run on Main using the PRTY parameter from the JOB card. If PRTY is not supplied on the OS JOB card the PRTY value from the CLASS or STANDARDS card is used. If this parameter is omitted or JPRTY=ASP is used, the execution priority will be assigned by ASP using the DPRTY value from the SELECT card. Job priority is changed after job selection but before job execution. Therefore the original priority is used for job selection and for any post-execution processing.

PRTY =nn

The ASP job priority to be assigned to each job in this job class, expressed as a decimal number from 0 to 14. This parameter may be overriden on a job basis with the PRTY keyword on the job card. If this parameter is omitted, the PRTY parameter from the STANDARDS control card will be used.

GROUP ={default|job-class-group-name}

The name of a Job Class Group to which this Job Class is to be assigned. This parameter should correspond with the NAME parameter on a GROUP initialization control card. If this parameter is omitted, the Job Class will be assigned to the default group name defined on the STANDARDS control card.

TDEPTH =depth

The maximum number of jobs of this class that can be run on the total ASP system at any one time; expressed as a decimal number from 0 to 255. If this parameter is omitted there will be no TDEPTH constraint on this class.

MDEPTH =(main-processor-name,depth,main-processor-name,depth,...)

The maximum number of jobs of this class that can be run on a particular Main Processor at any one time. Each Main Processor name must correspond with the NAME keyword on a MAINPROC control card. Each depth operand is expressed as a decimal number from 0 to 15. If this parameter is omitted, there will be no MDEPTH constraints on this class.

TLIMIT =(class-name,limit,class-name,limit,...)

The maximum number of jobs of other job classes that can run on the total ASP system and still allow jobs in this job class to be scheduled. If any class limit is exceeded, a job of this class will not be scheduled. Each class name must correspond with the NAME keyword on another CLASS control card. Each limit operand is expressed as a decimal number from 0 to 255. If the TLIMIT parameter is omitted, there will be no TLIMIT constraints for this class. For example, suppose Job Class ABC defined Job Class XYZ as being a TLIMIT of one job. The CLASS card would be:

```
CLASS,NAME=ABC,TLIMIT=(XYZ,1)
```

Under this constraint an ABC class job would be scheduled on an ASP Main Processor only when one or less class XYZ jobs were running in the ASP system.

MLIMIT =(main-processor-name,class-name,limit,class-name,limit,...)

The maximum number of jobs of other job classes that can run on a particular Main Processor and still allow jobs in this job class to be scheduled. The MLIMIT keyword should be repeated for each Main Processor's class limitations. If any class limit is exceeded, a job of this class will not be scheduled on that particular Main Processor. Each Main Processor name must correspond with the NAME keyword on a MAINPROC control card. Each class name must correspond with the NAME keyword on another CLASS control card. Each limit operand is expressed as a decimal number from 0 to 15. If the MLIMIT parameter is omitted, there will be no MLIMIT constraints for this class. For example, suppose Job Class ABC defined classes NOP and QRS as being MLIMITs of zero jobs on Main Processor SY1 and MLIMITs of one job on SY2. The CLASS card would be:

```
CLASS,NAME=ABC,MLIMIT=(SY1,NOP,0,QRS,0),MLIMIT=(SY2,NOP,1,QRS,1)
```

An ABC class job would be scheduled on SY1 only when no NOP and QRS jobs were running on SY1. An ABC job would be scheduled on SY2 only when one or less NOP and QRS jobs were running on SY2.

SYSTEM =(listed-above)

The SYSTEM parameter defines the Main Processor name(s) or type of system to be used for this class.

ANY defaults to any system (real or local) that will satisfy the job requirement.

LOCAL job is to be run on a local Main Processor only. The LOCAL Main Processor is the main in which ASP is resident.

REAL job is to run on any real Main Processor that will satisfy the job requirements.

main-name or (main-name,main-name,...) defines the only Main Processor(s) to be considered for this job.

/(main-name,main-name) defines the Main Processor(s) to be excluded from consideration for this job.

COLDSTART

This card signifies that the Support Processor is to be started with all of its queues empty. All space assigned to ASP on the direct access storage device is made available.

```
COLDSTART  [,JOBNO={(1,999)|(start,end)}]
```

JOBNO =(start,end)

Specifies that numbering of jobs by ASP will occur within a specified range. When the end of the range is reached, numbering resumes at the beginning of the range, skipping those numbers that are still assigned to jobs in the system. The range may be from 1 through 9999. Leading zeros are not required. If all job numbers are assigned, the next ASP DSP requesting a number will be put into a wait and a message will be written to the operator stating that no job numbers are available.

Example: COLDSTART,JOBNO=(100,9000)

CONSOLE

The CONSOLE card describes an operator console attached to the Support Processor and the message classes to be assigned to the console. One card is entered for each operator console. Console definition is independent of printer definition; the same unit may be assigned both functions.

Care should be taken when assigning a unit to two functions; that is, console and printer, to avoid intermixed printing.

```
CONSOLE,DDNAME={console-name|rjp-line-name}
       ,TYPE={1052|1053|1403|1443|2250|2260|2740|3060|3066|3210|3211|
              3215|3277|3284|3286|5450|RJP}
       ,UNIT=device-address     (not req. for TYPE=RJP)
       ,DEST={destination|(dest1,dest2,...)}
      [,MAIN=main-processor-name]
      [,ALTCON=console-ddname]
      [,LEVEL={15|nn}]
      [,DEPTH={50|nnn}]
      [,TIME={2|nn}]
      [,LL={device-default|physical-line-length}]
```

DDNAME={console-name|rjp-line-name}

Console name is the name to be used to define the console. This name is to be used in operator commands; for example, SWITCH, and in initialization cards; for example, DEVICE.

The rjp-line-name defines an RJP line which contains a remote workstation with a console and the name is to be identical with that used in the RJPLINE initialization card. The line-name is used only with the parameter TYPE=RJP on the CONSOLE card.

TYPE=

Specifies the type of device to be used as a console.

UNIT=device-address

The device address of the console. This parameter should not be specified when TYPE=RJP.

DEST={destination|(dest1,dest2...)}

DEST defines the message class(es) to be directed to this console. There are 96 message classes available. If a message class is not covered by a console, that message class will print on the master console for the support system or on the first console defined in the ASP initialization deck if no master console exists.

The valid message classes are:

ALL Messages pertinent to all consoles. This class will be assigned automatically to all consoles unless DEST=NONE is specified.

ERR Messages pertinent to equipment failure and ASP Failsoft messages.

LOG Messages defining an active job and function.

MN Messages pertinent to the Main Processor.

MLG If MLOG=YES has been specified in the STANDARDS control card, all input and output messages, except SEC, will be recorded on the MLOG console in addition to their original source or destination console(s).

SEC Security messages. Messages sent to this console will be written here only.

SUP Messages pertinent to the general operation of the Support Processor.

TAP Messages pertinent to ASP background utilities with tape unit requirements such as Tape-to-Print, Card-to-Tape.

TP Messages pertinent to teleprocessing equipment.

UR Messages pertinent to the operation of the unit record equipment.

The terms NONE, OUTPUT, and TOTAL are not specific message classes to be counted in the 96 available classes.

NONE No messages to be sent to this console.

OUTPUT All message classes except MLG to be sent to this console.

TOTAL All 96 message classes including MLG are to be sent to this console.

The remaining 86 message classes are combined into three groups; DGROUP , MGROUP , and SGROUP . Individual classes may be selected or by entering the group name in the DEST= parameter all of the classes in that group will be destined for this console.

DGROUP

D1 to D22 Messages pertinent to a user-defined console configuration.

MGROUP

M1 to M32 May be used to define a destination class unique to a Main Processor.

SGROUP

S1 to S32 Setup console. Refer to the MAIN and SUPPORT parameters on the DEVICE card for an example of usage.

MAIN=main-processor-name

The MAIN parameter defines this console as a main input console for the specified Main Processor. The name must correspond to the name given on a MAINPROC card. Any console may be designated as a Main input console and be associated with a specified Main Processor. This console may then communicate with OS commands without the need for the SEND verb. On such consoles, ASP commands must be preceeded by an * (asterisk).

ALTCON=console-ddname

In the event that a console goes out of ready and buffer depth is reached, or a console has an unrecoverable error, the console will be automatically switched to the specified alternate console. If this parameter is omitted it will default to the OS primary console, or,

if that console is not available, it will default to the first available console in the Console Status Table. If the user does not want this console to automatically switch, the ALTCON ddname must be the same ddname as this console being defined.

LEVEL={15|nn}

The LEVEL parameter defines the authority level (0-15) for issuing commands. Each console should be defined with a specific level. It is the user's responsibility to define the levels and write (if he chooses) a CONSAUTH module to replace the skeleton module supplied with the system. The CONSAUTH routine determines if a given console is authorized to issue a given command.

DEPTH={50|nnn}

The number of messages to be queued to a console. When a console has its full complement of messages queued (reaches depth), any ASP function subsequently issuing messages to it must wait until the required buffers become available. From 1 to 255 may be specified. If this parameter is omitted, the assumed number of messages is 50.

TIME={2|nn}

A one- or two-digit number specifying the delay, in seconds, between output messages on a graphic device. This field must not exceed 20 seconds. This parameter affects graphic devices only, for example; 2250, 2260, 3060, 3066, 3277, and 5450. Care must be taken not to set this value too high, which may cause a large backlog of messages and cause the system to wait because depth has been exceeded. A page mode facility may be implemented on the 3277 by entering a delay time of zero. The Page Mode is particularly effective when the total number of messages queued for the 3277 exceeds 22 message lines. When the delay is set to zero, the 3277 will display messages until the screen is full. The display will then freeze with the message "DISPLAY HALTED-CLEAR TO RESTART" on the bottom line. After examing the screen the operator must depress the CLEAR key to release this "page" of display and procede to the next "page". After a fifteen-second interval the display will automatically be stepped to the next "page".

LL={device-default|physical-line-length}

The LL parameter defines the longest line to be printed on this console. If the actual line to be written is longer than the specified value the line will be continued to a second line. A line that is to be continued will be broken at a comma or a blank. The default values are:

Graphic: 74 for 2250/3060

80 for 2260/5450/3066 and other graphic consoles

Hard Copy: 120 for all hard copy consoles.

Examples of the use of the CONSOLE cards on a system that has an IBM 3277 Display Station to display all output messages; an IBM 1443 Printer as the master log console; an IBM 1052 Printer/Keyboard for Main Processor and Support Processor operation; an IBM 2740 Communications Terminal for device mounting, tape, unit record, and teleprocessing; and a 2740 to monitor all user-defined setup messages.

```
CONSOLE,DDNAME=CN1,UNIT=041,TYPE=3277,DEST=OUTPUT
CONSOLE,DDNAME=CN2,UNIT=00E,TYPE=1443,DEST=MLG,DEPTH=150
CONSOLE,DDNAME=CN3,UNIT=01F,TYPE=1052,DEST=(MN,SUP,ERR)
```

```
CONSOLE,DDNAME=CN4,UNIT=0BA,TYPE=2740,DEST=(TP,TAP,UR)
CONSOLE,DDNAME=CN5,UNIT=0BB,TYPE=2740,DEST=SGROUP
```

Note: When using RJP,
one CONSOLE card
must be provided for each line specified on an RJPLINE card.
These CONSOLE cards should be provided as follows:

```
DDNAME=rjpline-name

    as specified on the RJPLINE Initialization control card.

TYPE=RJP

DEST={NONE|D1,...D22}

[DEPTH=nnn]
```

Examples are:

```
CONSOLE,DDNAME=TULSA,TYPE=RJP,DEST=NONE
CONSOLE,DDNAME=BOSTON02,TYPE=RJP,DEST=NONE
```

Destination classes may be assigned to RJP consoles (for example, DEST=D3) giving the operator the ability to broadcast messages to those consoles.

DEADLINE

Deadline Scheduling is used to increase a job's priority to enable it to complete by a specific time (the deadline time). The DEADLINE card defines the types of algorithms available for affecting a job's scheduling.

```
DEADLINE,type=(prty,lead[,pinc,int])
```

type

Is any single character identifier (A-Z or 0-9) parameter and is referenced in the //*MAIN card, DEADLINE parameter.

prty

Specifies the initial change in priority. Prty is a priority level to be set or an incremental value to be added to a job's priority. If it is an incremental value, it must be preceded by a plus sign (+).

lead

Specifies the amount of lead time, in hours and minutes, prior to the job's deadline that the initial priority change (prty) should be made.

pinc

Specifies the subsequent priority change or subsequent increments to be added to the priority. If it is an increment it must be preceded by a plus sign (+).

int

Specifies the time interval between application of the subsequent priority change (pinc).

There are no default values for the subsequent values, pinc and int. Therefore only the initial change (prty and lead) will be made if the subsequent values are omitted.

The time values for lead and int may be specified in minutes or hours, that is, 10 minutes, 10M or 2 hours, 2H.

Examples:

DEADLINE, A=(10,1H,+1,30M) specifies deadline type A. Type A will cause priority 10 to be set one hour prior to a job's deadline and will increment the priority by 1 every 30 minutes until the job is complete.

DEADLINE, B=(+4,3H,11,2H) specifies deadline type B. Type B will cause the job's priority to be incremented by 4 three hours prior to the deadline and will set the priority to 11 if the job hasn't completed two hours later.

DEVICE

The DEVICE card identifies the devices attached to the Support Processor and/or the Main Processor. One card is entered for each device (card reader, card punch, printer, disk unit, and tape unit) that may be assigned to the Support Processor or that is mountable on the Main Processor and, hence, is assigned to and controlled by the Support Processor. When devices are shared between the Support Processor and Main Processor or between Main Processors, this condition is indicated with a single DEVICE card with multiple entries. For each physical device, there should be one, and only one, DEVICE card. If a device is on the Support Processor, only the STYPE, SUPPORT, GTYPE must be provided. For a Main device, the MTYPE and MAIN parameters must be provided. If a device is shared between Support and one or more Main Processors, all five parameters must be provided. The DEVICE card is also used to define pseudo devices on the Support Processor. The pseudo devices are used to control TSO support (ACM), Internal Job Processing (IJM), the ASP Reader/Interpreter (ARI), each Main Processor (SYS), and Network Job Processor (NJP). A pseudo device is to be defined for each function that is to be active. These devices can be varied on or offline as any real device and have entries created in the internal tables (SUPUNITS and SYSUNITS).

```
DEVICE,STYPE={1403|2400|2501|2540|3211|3400|3505|3525|3525I|
              ACMP|IJMP|LINE|MAIN|OSRDR|user-specified}
       ,SUPPORT=(unit-address,name,[console-dest-class|LOG][,{ON|OFF}]
       ,GTYPE={ACM|ARI|IJM|NJP|PUN|PRT|RDR|SYS|TA7|TA9|
               user-supplied-device-type}
      [,MSTATUS=({TA|DA},{RM|PR|ND},vol-ser)]
       ,MTYPE=main-device-type-from SETNAMES card
       ,MAIN=(unit-address,system-name,[console-class|
              MDSLOG-parm][,{ON|OFF}])
      [,GNAME={LOCAL|group-name}]
```

STYPE

STYPE, in conjunction with the GTYPE parameter, defines the specific types of devices attached to the Support Processor. The STYPE parameter contains those devices supported by ASP and indicates that the user may define a unique type. If the user defines his own STYPE he is restricted to names of five characters or less and must provide the support for that device. The STYPE parameter must precede the SUPPORT parameter. Because of the flexibility of user STYPE's, no test is made for valid types.

SUPPORT

The SUPPORT parameter is used in conjunction with the STYPE and GTYPE to define a Support Processor unit. The unit address, device name, console destination class, and optional OFF parameters are positional. Unit address and device name must be specified where NONE is a valid unit address if needed. If console class is not specified, the class LOG is assumed. The unit address is the system addressable address; device-name is user-defined and is restricted to one to eight characters and cannot be the same as the GNAME or its default. For IJP and ACDS DEVICE cards (STYPE=IJMP or ACMP, GTYPE=IJM or ACM), the unit address must be NONE and the device name must be a Main Processor name prefixed by IJ or AC, respectively. The console class is the console destination supporting this device, typically the console physically nearest this device; it may be any of the 96 possible classes (refer to the CONSOLE card). The optional OFF parameter, if used, specifies that the device is to be offline to the ASP system initially.

GTYPE

Is a general classification name. This parameter is used to combine associated machine types, that is, a GTYPE of PRT might be combined with an STYPE of 1403 or 3211. A user-supplied GTYPE must be three characters. Refer to Chapter 6, Support Device Grouping.

MSTATUS

MSTATUS will be used to indicate device type, volume removability, and an expected volume serial for permanently resident devices. Device type is either TA or DA indicating tape or direct access devices. Removability is indicated by RM; PR indicates the volume on that device is considered permanently resident. A permanently resident volume indicates that the volume will not be dismounted and no MOUNT messages will be issued for that volume by SETUP. ND implies PR and additionally that any request for the volume on the device will be allowed to access the volume without regard to the DISP of the reference. The volume serial can be used to indicate the volume mounted on the device, to inform MDS of its presence, prior to IPLing the Main Processor to which it is attached. If MSTATUS is used it must precede the MAIN parameter.

MTYPE

The type of Main Processor device that the DEVICE card is specifying. This is a one- to eight-character identifier that equates this device to a SETNAME card MTYPE. The MTYPE parameter must precede the MAIN parameter or MSTATUS parameter.

MAIN

Used when the device is attached to one or more Main Processors. The MAIN parameter must be used in conjunction with MTYPE to define Main Processor device. The physical device address and the system name, as specified in a MAINPROC card, to which the device is attached must be specified. If the device is shared between two or more Main Processors, pairs of device addresses and Main Processor names are used to identify this condition. If a tape is shared between two channels of the same Main Processor via a Tape Switching Unit, the tape address that was SYSGENed should be the only one to appear. For an explanation of the console class and OFF parameter see the SUPPORT parameter. All SETUP messages for this unit will be routed to the class specified in this parameter. If a console class is not specified, the console class supplied on the MDSLOG parameter (SETPARM card) will be used.

GNAME

A group classification name to be used to combine devices by physical location. If input is received from a device in this GNAME category, an attempt will be made by ASP to send the associated output to the same GNAME location. Refer to group discussion in Chapter 6 of this manual.

Examples:

1. Defining a 1403 for the Support Processor.

   ```
   DEVICE,STYPE=1403,SUPPORT=(00F,PR1,UR),GTYPE=PRT
   ```

   Defining a 3211 printer would be done the same way except that STYPE=3211.

2. Defining a card reader and a card punch.

```
DEVICE,STYPE=2540,SUPPORT=(00C,RD1,UR),GTYPE=RDR
DEVICE,STYPE=2540,SUPPORT=(00D,PU1,UR),GTYPE=PUN
```

3. Defining a Main Processor. A DEVICE card must be supplied for each Main Processor.

```
DEVICE,STYPE=MAIN,SUPPORT=(NONE,SY1),GTYPE=SYS
```

4. Defining two pseudo readers for the ASP R/I. To support a Main Processor at least one pseudo reader must be defined.

```
DEVICE,STYPE=OSRDR,SUPPORT=(NONE,OSRD1),GTYPE=ARI
DEVICE,STYPE=OSRDR,SUPPORT=(NONE,MYRDR1),GTYPE=ARI
```

5. Defining an NJP line.

```
DEVICE,STYPE=LINE,SUPPORT=(050,HARRY),GTYPE=NJP
```

The corresponding DD card in the ASP JCL would be:

```
//HARRY DD  UNIT=050
```

6. Defining the IJP support. At least one device must be specified for each Main Processor that will communicate with IJP.

```
DEVICE,STYPE=IJMP,SUPPORT=(NONE,IJSY1),GTYPE=IJM
```

7. Defining TSO support. One device must be specified for each Main Processor that will communicate with the ADCS DSP.

```
DEVICE,STYPE=ACMP,SUPPORT=(NONE,ACSY1),GTYPE=ACM
```

8. The next DEVICE card describes a printer (GTYPE=PRT) as a 1403 (STYPE=1403) with a unit address of 00F. The unique name for this printer is PR2 and the associated console class is UR. The group name associated with this printer is MACHROOM.

```
DEVICE,GTYPE=PRT,STYPE=1403,SUPPORT=(00F,PR2,UR),GNAME=MACHROOM
```

9. The next DEVICE card describes a nine-track tape on Support and two Mains. It is initially offline to Support and SY2. The console class for support is S7, for SY1 it is S9 and SY2 it is S6. The double comma (S9,,) indicates the absence of the offline parameter for SY1. By default the group name (GNAME) is LOCAL.

```
DEVICE,GTYPE=TA9,STYPE=2400,SUPPORT=(180,T90,S7,OFF),MTYPE=SYSSQ,
      MAIN=(180,SY1,S9,,280,SY2,S6,OFF)
```

10. To utilize the interpreter feature on the 3525.

```
DEVICE,STYPE=3525I,SUPPORT=(00D,PU2,UR),GTYPE=PUN
```

ENDASPIO

The ENDASPIO card is used to note the end of the required initialization cards. This card has no parameters.

```
ENDASPIO
```

ENDINISH

ENDINISH must be the last card in the initialization deck. The IPL deck should follow if supplied.

```
ENDINISH
```

FORMAT

The FORMAT card specifies the direct access storage devices for the ASP queue to be formatted during initialization. This card should be used only when a new, unformatted volume is being introduced into the system or when the ASP buffer size is changed on the BUFFER card. After formatting has been completed, a TRACK card may be substituted for the FORMAT card. This FORMAT card must be used only during cold start initialization. Its use during a restart will destroy the queue and cause the user to do a cold start. DDNAME provides the one- to eight-character name of the DD card that defines the space on the device to be formatted.

```
FORMAT,DDNAME=ddname
```

Examples of the FORMAT card are:

```
FORMAT,DDNAME=ASPQ1
FORMAT,DDNAME=ASPQ2
```

GROUP

The GROUP card defines the characteristics of an ASP Job Class Group. A GROUP card must define each Job Class Group used on a CLASS control card. Up to 255 Job Class Groups can be defined.

```
GROUP,NAME=job-class-group-name
     ,EXRESC=(main-processor-name
              ,{dedicated-initiator-count|system-task-name}
             [,dedicated-main-storage-size]
             [,allocation-option]
             [,deallocation-option]
             [,storage-protect-option])
     [,PRTY={0|nn}]
```

NAME=job-class-group-name

One to eight alphameric characters specifying the name of the job class group. The NAME parameter must be the first parameter on the GROUP card. A job class is assigned to a job class group by placing the group name on its CLASS control card.

```
EXRESC=(main-processor-name
       ,{dedicated-initiator-count|system-task-name}
       [,dedicated-main-storage-size]
       [,allocation-option]
       [,unallocation-option]
       [,storage-protection-option])
```

This keyword parameter defines the execution resources (that is, OS initiators and Main Storage space) to be assigned to a Job Class Group. The EXRESC keyword should be repeated for each Main Processor's execution resources. The keywords are position dependent.

Each Main Processor name must correspond with the NAME keyword on a MAINPROC control card.

The dedicated initiator count defines the number of initiators to be assigned exclusively to the Job Class Group. When scheduling jobs from this Group on Main only these dedicated initiators will be used. Therefore, this count defines the maximum number of jobs of this group that can run concurrently on a particular Main Processor. Dedicated initiators that become idle will not be used for scheduling any other Job Class Group. A maximum of 16 initiators can be assigned to a Job Class Group. Each initiator will have a six-character identifier. The first three characters are always ASP. The fourth character is the OS job class assigned to the ASP job class group. The fifth and sixth characters are a two-digit sequence number. The OS job classes that ASP will use can be defined with the JOBCLASS parameter on the MAINPROC control card. Using the default JOBCLASS the initiator IDs will be ASPA01, ASPA02, etc. The dedicated initiator count for the default group ASPBATCH is the value MINIT less the number of IPL allocated initiators assigned to other Job Class Groups. (The MINIT parameter is defined on the SELECT card; the default value is 2.)

The name of an OS system task may be used in place of the dedicated initiator count. If the task fencing option is used only the task named by this parameter will be allowed inside the fence. Since a system task must be started by the operator, the dynamic allocation and unallocation fencing options are not available.

The dedicated Main Storage size option defines the size of the Main Storage area to be assigned exclusively to this Job Class Group. This Main Storage area (also referred to as a Main Storage Fence) is a

contiguous block of storage in which one or more jobs of this Group will execute. Therefore this parameter defines the maximum amount of Main Storage to be used by this Group on a particular Main Processor. If region space within this Fence becomes available it will not be used for scheduling any other Job Class Group. If Main Storage Fencing is not used, the jobs in this Group will compete for region space on a Group priority basis with any other unfenced Job Class Groups. The Fence size is specified as nnnnnK, where nnnnn is a two- to five-digit decimal number expressing the Fence size as a multiple of 1024. The fence size cannot be less than the OS minimum initiator region size (MINPART as defined at SYSGEN time or at IPL time).

The allocation option determines when the execution resources are to be allocated to the Job Class Group. There are three options available: DYNAMIC|IPL|MANUAL. If DYNAMIC is used, the execution resources are allocated when the first job of this Job Class Group becomes eligible for scheduling on Main. If IPL is used, the resources are allocated whenever the Main Processor is IPLed. If subsequent reallocation is required, it must be done manually. Or if MANUAL is used, the resources will be allocated whenever the operator enters one of the following ASP Modify commands (see the Operator's Manual for the format of these Modify commands):

1. A Modify command to turn the Group on.

2. A Modify command to alter the number of initiators assigned to the Group.

3. A Modify command to change to a new Select mode and this new Select mode causes the Group to be turned on (see the description of the GROUP keyword on the SELECT control card).

The unallocation option determines when the execution resources are to be released from the Job Class Group. Two options are available: DYNAMIC|MANUAL. If DYNAMIC is used, the resources are released after the last job of the Group terminates on Main. If MANUAL is used, the execution resources will not be released until the operator disables the Job Class Group with an ASP MODIFY command. The option for the default GROUP ASPBATCH is MANUAL.

The storage protection option determines the type of storage protect key to assign to the jobs in the Group. This option can be used only when the Job Class Group is fenced. Two options are available: OSKEY|ONEKEY. If OSKEY is used, each job in the Group that executes on Main will be assigned a unique storage protection key and therefore is protected from storage violations from all other jobs on the Main Processor. If the ONEKEY option is used, all jobs in the Group that execute on Main will be assigned the same storage protection key. This option protects these jobs from storage violations by jobs in other job class groups and protects other job class groups from storage violation by jobs in this job class group. However, jobs in this job class group are not protected from storage violations by jobs in this same job class group. This option permits more than 15 OS initiators to be active at the same time on a particular Main Processor.

PRTY={0|nn}

The GROUP priority, expressed as a decimal number from 0 to 15. If this parameter is omitted, priority level zero will be used. Groups compete for execution resources (that is, initiators and Main storage space), on a group priority basis. Jobs in a high priority job class group will be scheduled on Main before jobs in a low priority job class group. Execution resources are allocated to job class groups on a group priority basis.

For example, to dedicate three initiators and 400K of Main Storage to Job Class Group ABC on a dynamic basis on Main Processor SY1, the following Group card would be used:

```
GROUP,NAME=ABC,EXRESC=(SY1,3,400K)
```

To dedicate dynamically one initiator and 150K of Main Storage on Main Processor SY2 and to release the resources under operator control, the following Group card would be used:

```
GROUP,NAME=HOT,EXRESC=(SY2,1,150K,,MANUAL)
```

IPL DECK

The IPL deck controls IPL of the MVT control program on each Main Processor in the system. It should contain the operator dialogue required to start up the system, in order to minimize operator action. The IPL deck is not required for a local or dummy mode Main Processor.

The IPL deck must immediately follow the ENDINISH card. Each IPL set must begin with:

IPL,NAME=main-processor-name[,TYPE=MVT]

The parameter NAME= specifies the name of the Main Processor, as defined in a MAINPROC card.

The operator IPL dialogue may be manual or automatic. In the manual mode, IPL messages from the Main Processor appear on the console, prefixed by R=. The operator must respond to these messages, using the *SEND verb. The input messages then appear on the console, prefixed by S=. When the IPL dialogue is complete, processing is initiated with the reply by the ASP operator:

*START,main-processor-name,END IPL

In the automatic mode, the operator dialogue is punched into a series of cards that follow each IPL card. Both the output and the input messages must be specified and must be in the order in which they occur. Output messages that do not have corresponding IPL text cards are allowed and will always be printed. The Main Processor output messages may be shortened to include just the alphameric prefix (such as R=IEE101A); the text will be printed on the console if the MAINPROC card DISPLAY option specifies IPL or ALL. The operator responses must be prefixed by S= and must be written out completely.

A PREJOB card may be included in the IPL deck if a PREJOB is desired. The format is:

PREJOB[,SYNC|,ASYNC],START jobname[,valid start command parameters]

The PREJOB START command will be sent to the Main Processor, and if SYNC was specified, Main Scheduling will not occur until the PREJOB has terminated.

Note: The initiation of OS functions, readers, writers, and initiators is performed by ASP. Commands for initiation of these functions should not be put in the auto-IPL text.

The last card of the IPL deck is IPL,END. There must be only one IPL,END card and it must be behind the last set of IPL text and is required for all Main Processors; Real, Local, or Dummy.

The following cards should be included.

the S=SET DATE

As a result, the WTOR message IEE114A will be logged on the master console. The operator can then respond with the applicable message by using the SEND verb. (If the auto feature has been sysgened, AUTO=NONE should be included in the response.) The MSV06 format message will not be issued to the operator in this case.

IPL Deck Example:

```
IPL,NAME=SY1,TYPE=MVT
R=IEAMVTA
S=R **,'U'
R=IEE101A
S=SET DATE
R=IEE103A
S=S MT,REGION=70K
PREJOB,ASYNC,START MPPREJOB,REGION=40K
R=AMTR04 MT INIT COMP
IPL,NAME=MAIN2
R=IEAMVTA
S=REPLY **,'RSVC=01,SQS=4'
S=MN JOBNAMES
S=SET DATE
S=V 180,OFFLINE
IPL,NAME=TSOSYS
IPL,END
```

MAINPROC

A MAINPROC control card is required for each Main Processor in the ASP system.

```
MAINPROC,NAME=main-processor-name
        ,MAINCTC=device-address
        ,ADAPTER=device-address
        ,CTCCUA=(ctc-control-unit-address,...)
       [,MDEST={console-dest|M1}]
       [,SYSTEM={REAL|LOCAL|DUMMY}]
       [,PROC={CATL|device-address}]
       [,Q={CATL|device-address}]
       [,QFORMAT={ASK|YES|NO}]
       [,SELECT={job-selection-mode-name|ASPBATCH}]
       [,ID=main-processor-message-identifier]
       [,MSGCLASS={B|msgclass}]
       [,JOBCLASS={A|(os-jobclass,os-jobclass,...)}]
       [,RID={receive-message-id|R#}]
       [,SID={send-message-id|S#}]
       [,SMFID={smf-id|00}]
```

NAME=main-processor-name

One- to eight-alphameric characters specifying the external name of the Main Processor. This name is used in ASP console commands to communicate with the Main Processor. A DEVICE control card using this Main Processor name must appear in the ASP initialization deck.

MAINCTC=device-address

A three-digit field specifying the device address of the Channel-to-Channel Adapter (CTC) on the Main Processor. If SYSTEM=LOCAL is specified, this parameter must specify the address of the dummy CTC that is to be simulated. If SYSTEM=DUMMY (used with POLYASP) is specified, this parameter is not required.

ADAPTER=device-address

A three-digit field specifying the device address of the CTC on the Support Processor. If SYSTEM=LOCAL is specified, this address must be the same as the MAINCTC specification. If SYSTEM=DUMMY is specified, this parameter is not required.

CTCCUA=(ctc-control-unit-address,ctc-control-unit-address,...)

One to sixteen hexadecimal digits defining the Channel-to-Channel (CTC) control unit addresses on which CTC devices are SYSGENed on the Main Processor (that is, the CTC control units defined by the SYSGEN IOCONTRL macro). This permits identification of I/O messages for these devices so that they will not be routed to tape setup consoles. For example, if four CTC control units, 370, 380, 390, and 3A0 were SYSGENed on Main the following would be used:

```
CTCCUA=(7,8,9,A)
```

MDEST={console-dest|M1}

The ASP operator console class (M1-M32) to which Main Processor messages should be directed for the Main Processor that is being defined. If this parameter is omitted, messages for this Main Processor will be logged on the M1 console destination class (see the CONSOLE initialization card).

SYSTEM={REAL|LOCAL|DUMMY}

If REAL is specified, the Main Processor is separate from the Support Processor. If LOCAL is specified, the Main Processor is to reside in unoccupied regions of the Support Processor. If DUMMY is specified, a real Main Processor is simulated by passing jobs through the Main Service function without actual execution.

PROC={CATL|device-address}

If this parameter is omitted or the CATL operand is used, the SYS1.PROCLIB data set is assumed to be cataloged on the Main Processor. A three-digit field specifying the address of the procedure library may also be used. This parameter is not required if SYSTEM=LOCAL or SYSTEM=DUMMY is specified.

Q={CATL|device-address}

If this parameter is omitted or the CATL operand is used, the SYS1.SYSJOBQE data set is assumed to be cataloged on the Main Processor. A three-digit field specifying the address of the direct access device on which the volume that contains the OS job queue resides may also be used. This parameter is not required if SYSTEM=LOCAL or SYSTEM=DUMMY is specified.

QFORMAT={ASK|YES|NO}

If this parameter is omitted or the ASK operand is used, the operator must reply to the MSV06 ISSUE SYS1.SYSJOBQE FORMAT OPTION message. If YES is specified, a SET command requesting SYS1.SYSJOBQE formatting will be sent to Main. If NO is specified, no SYS1.SYSJOBQE formatting is requested. If either YES or NO is used, no MSV06 message is issued and therefore no operator assistance is required.

SELECT={job-selection-mode-name|ASPBATCH}

The initial job selection mode for the Main Processor. This name must correspond with the NAME parameter on a SELECT control card. If this parameter is omitted a selection mode consisting of SELECT card default values will be used. This default SELECT mode is named ASPBATCH.

ID=main-processor-message-id

One to eight alphameric characters to be prefixed to every Main Processor message logged on the MDEST console. If this parameter is omitted, all messages received from a Main Processor and logged on MDEST are prefixed with R=. If DISPLAY=ALL or IPL is in effect, all messages sent to a Main will be logged on MDEST and prefixed with S=. The specified message ID will prefix the R= or S=. For example, ID=SY2 will log a received message as SY2 R=message-text. A message sent to Main will be logged as SY2 S=message-text.

MSGCLASS={B|msgclass}

One OS message class is reserved for ASP. This output class is assigned by ASP to every job scheduled on Main. As each job terminates on Main the ASPWRITR routine in MAINTASK retrieves the job scheduler messages for the job from this output class. If this parameter is omitted, MSGCLASS=B will be used.

JOBCLASS={A|(OS-jobclass,OS-jobclass,..)}

One to fifteen OS job classes reserved for ASP. A different OS job class is required for each active ASP Job Class Group. If this parameter is omitted, OS job class A is used. An OS job class should be reserved for each GROUP initialization card plus one for the default group, ASPBATCH.

RID={receive-message-id|R#}

One to eight alphameric characters to be prefixed to every message received from the Main Processor and logged on the MDEST console. For example if RID=SY2 is used the messages received will be logged as SY2 R=message-text. However if a pound sign is the last character of the RID operand, the operand will immediately precede the equal (=) sign. For example if RID=2# is used the messages received will be logged as 2=message-text. If this parameter is omitted and the ID keyword is not used the messages received will be logged as R=message-text.

SID={send-message-id|S#}

This parameter is identical to RID except that it applies to the messages sent to the Main Processor that are logged on MDEST. For example if SID=2S# is used the messages sent to Main will be logged as 2S=message-text. If this parameter is omitted and the ID keyword is not used the messages sent to Main will be logged on MDEST as S=message-text. Note that the DISPLAY parameter on the SELECT card controls which messages are logged on MDEST.

SMFID={smf-id|00}

A two-character CPU ID for SMF. This ID is inserted into the OS SMF job queue records before a job is scheduled for the Main Processor.

NJPTERM

The NJPTERM card identifies the remote NJP ASP Support Processor terminals to and from which ASP may send and receive data. It establishes the name for both operator and programmer use. It describes the terminal block size for use in obtaining transmittal buffers and defines the name of the line linking the two ASP terminals.

```
NJPTERM,NAME=terminal-name
       ,BLKSIZE=nnnn
       ,LINES=(ddname1,{A|B})(ddname2,{A|B})....
```

The keyword parameters are:

NAME=terminal-name

One to eight alphameric characters designating the name of a remote ASP terminal to which this system is connected via a communication line. This name is to be used on the //*FORMAT NJPIO card when a job is to be sent to a remote location and by the console operator when sending jobs to the remote terminal. Note the difference in terminal-name length with that of RJP.

BLKSIZE=nnnn

A decimal number specifying the size of the transmission blocks to be used to transmit the data over the line. This parameter must be greater than the ASP buffer size by at least 13 bytes and must be exactly equal to the BLKSIZE used at the remote location defining the same line connection.

LINES=(ddname1,{A|B})(ddname2,{A|B})....

Defines the line connecting the two terminals. This ddname must appear on a DD card associated with the unit address for the line. A or B specifies the adapter interface.

Examples of the NJPTERM card are:

```
NJPTERM,NAME=LAX,BLKSIZE=2000,LINES=(TRMRDR,A)

NJPTERM,NAME=CHICA,BLKSIZE=1000,LINES=(ALINE,A)(BLINE,A)
(CLINE,B)
```

OPTIONS

There are a number of execution options for the ASP system that assist in the testing of the system. They are implemented by keyword parameters and are defined as follows:

```
OPTIONS[,DUMP={ASP|OS|SA}]
       [,ADDSAVE=nn]
       [,TRACE=nnn]
```

DUMP Specifies the type of dump to be taken in the event ASP abnormally terminates or program checks.

ASP Specifies a dump of the OS control blocks and ASP region with the ASP control blocks formatted and is written to the ASPABEND data set.

OS Specifies a dump of the OS nucleus, SQS, OS control blocks, and ASP region with the ASP control blocks formatted. Written to the ASPABEND data set.

SA Specifies a core-image dump to be written to the ASPSADMP data set. The core-image dump can be printed with the IMDPRDMP OS service aid program with the ASP modification to format the ASP control blocks.

ADDSAVE=nn

Specifies the number of additional save areas for use by ASP. This number will be permanently allocated. A suggested value is one per FCT.

TRACE=nnn

The TRACE parameter is used to generate a log of the Dispatching and Calls of ASP functions, where nnn is the number of entries. Refer to Chapter 9, Debugging Aids In ASP, for a sample trace table. This feature cannot be controlled by an operator. Each entry is 32 bytes.

Example: OPTIONS,DUMP=OS,ADDSAVE=30,TRACE=100

PFK

The Program Function Key (PFK) initialization card defines the operator command to be entered via the program function keys of a 3277. A table is created in main storage to contain the defined message. The size of the table is determined by the length of the longest message multiplied by the highest key number (K=nn) used. The key number is used to index into the table to locate the message to be submitted. An undefined program function key may be used by the operator to cancel a message being entered. The text can not be continued onto another card.

```
PFK,K=nn[,E={YES|NO}],M=<text>
```

K=nn

This parameter defines the program function key number. The nn value may be 1 through 12 for a 3277.

E={YES|NO}

Determines when the message is to be entered. YES specifies that the message is to be entered upon function key depression. NO specifies that the message is to be displayed with the cursor at the end of the partial message, enabling the operator to complete the message.

M=<text>

Message text to be associated with this key. This must be the last parameter on the PFK card.

Examples:

```
PFK,K=1,M=<*X RJP>
PFK,K=2,E=NO,M=<*S RJP,L=>
RPK,K=4,M=<*X DC>
PFK,K=5,E=NO,M=<*S DC,>
PFK,K=6,M=<*C DC>
```

PREJOB

The PREJOB card is described under the IPL deck, of which it is an optional part. It is not an initialization card.

PRINTER

The PRINTER card describes a printer attached to the Support Processor. One card is entered for each printer. The parameters describe the cold start or restart status of the printer and its setup characteristics. This information is used by Print Service to perform printer resource scheduling. If no PRINTER card is supplied for a given printer, default values will be taken from the STANDARDS card.

```
PRINTER,NAME=name from DEVICE card
      [,TYPE={1403|3211|RJP}]
      [,UCS={YES|NO}]
      [,FORMS=({YES|NO},{STANDARD|starting-forms-name})]
      [,CARRIAGE={STANDARD|starting-carriage-tape-name}]
      [,TRAIN=({YES|NO},{STANDARD|AN|HN|PN|QN|QNC|RN|SN|TN|XN|YN|
                         PCSAN|PCSHN})]
      [,HEADER={YES|NO}]
      [,BURST={YES|NO}]
      [,BUFDED={NO|YES|(YES,{3|1})}]
```

NAME=name

The name of the printer being described. This name must be the same as that in the DEVICE card for the printer, except for TYPE=RJP. A DEVICE is not required for RJP and this name may be any name.

TYPE={1403|3211|RJP}

Defines the printer type. RJP indicates the printer is on a remote RJP workstation.

UCS={YES|NO}

Specifies whether the printer is equipped with an IBM 2821 Control Unit with a Universal Character Set buffer. For remote RJP printers, this parameter should be specified as UCS=NO.

FORMS=({YES|NO},{STANDARD|starting-forms-name})

Specifies whether the forms on this printer may be removed and different forms mounted during execution; also, gives the name of the forms that are on the printer when the system is initialized. The forms name may be from one to eight characters in length. Standard forms are those defined in the ASP initialization STANDARDS card. If the combination of DEST=printer in the //*FORMAT card and DLOCATE=YES in the ASP initialization STANDARDS card causes the printer to be assigned, FORMS=YES is forced for the data set. If the FORMS parameter is specified the YES|NO must be specified.

CARRIAGE={STANDARD|starting-carriage-tape-name}

Specifies the name of the carriage tape that is on the printer when the system is initialized. The name may be from one to eight characters in length. The standard carriage tape is the one defined in the STANDARDS card. If FORMS=NO is specified, it is assumed that the carriage tape also may not be changed during execution. For 3211 type printers a module must be included in the ASP library for each carriage tape name. The FCB name is FCB2 plus the first four characters of the carriage tape name.

TRAIN=({YES|NO},{STANDARD|AN|HN|PN|QN|QNC|RN|SN|TN|XN|YN|PCSAN|PCSHN})

Specifies whether the train on this printer may be removed and a different train mounted during execution. Also gives the name of the train that is on the printer when the system is initialized. The

standard train is the one defined in the STANDARDS card. TRAIN=STANDARD may not be used in conjunction with the STANDARDS card specification TRAIN=ANY. If the combination of DEST=printer in the //*FORMAT card and DLOCATE=YES in the ASP initialization STANDARDS card causes this printer to be assigned, TRAIN=YES is forced for the data set. TRAIN modules for 3211 printers must be named the same as their 1403 equivalents with the suffix 11; for example, PN for a 1403 and PN11 for a 3211. These should be specified in the TRAIN parameter as PN. If the TRAIN parameter is specified the YES|NO must be specified.

HEADER={YES|NO}

Specifies whether block header pages should be printer for each data set on this printer. If HEADER=NO is specified, no header pages will be printed for the job or its data sets.

BURST={YES|NO}

Specifies whether burst pages should be printed after each job. If BURST=NO is specified, no burst pages of any kind will be printed on this printer.

BUFDED={NO|YES|(YES,{3|1})}

BUFDED specifies that dedicated buffers are to be used by Print Service. By dedicating buffers, printer output can be optimized. YES indicates that three buffers are to be dedicated, (YES,1) indicates one dedicated buffer.

The PRINTER card fills in the Printer Resource Table (PRT). For each DEVICE card with STYPE=1403 or 3211 an entry is made in the PRT and is completed with the information from the STANDARDS card. The PRINTER card will overlay the information from the STANDARDS card.

RESCTLBK

The RESCTLBK card is an optional ASP initialization control card which allows the user to preallocate main storage for the high usage ASP Function Control Table (FCT) and Resident Job Queue (RESQUEUE) tables. This can reduce GETMAIN overhead and its associated core fragmentation problems. This control card is not optional if the installation intends to utilize the ASP device-to-device or group-to-group reassignment capabilities as described in the ASP Operator's Manual (MODIFY - ASSIGN). The format of the RESCTLBK card is:

```
RESCTLBK[,ASG=nnn]
        [,FCT=nnn]
        [,RQ=nnn]
        [,VUT=nnn]
```

ASG=nnn

This indicates the number of assignment table entries. nnn is a decimal number from 1 to 999. nnn represents the maximum number of device-to-device or group-to-group assignments that will be allowed. If it is not specified, none will be allowed. Each entry is the 16 bytes and ten is a recommended number of entries.

FCT=nnn

This indicates the number of Function Control Table (FCT) entries to preallocate. nnn is a decimal number from 1 to 999. Each FCT is constructed so as to include: a GETUNIT list large enough to include two device entries, a save area as used by the ASP ASAVE routines. The ASP Transfer Vector Table (TVT) has pointers to the beginning (PAFCTTOP) and end (PAFCTBTM) of the preallocated FCTs which are not in use.

RQ=nnn

This indicates the number of Resident Queue (RESQUEUE) entries to preallocate. nnn is a decimal number from 1 to 999. As with the FCT, the ASP TVT has pointers to the preallocated RESQUEUE entries not in use (PARQTOP, PARQBTM).

VUT=nnn

Indicates the number of resident volume unavailable table entries to be reserved. nnn represents a decimal number from 1 to 999.

RESIDENT

The RESIDENT card instructs the ASP initialization program to make an ASP system module permanently resident. Each parameter specifies the one- to eight-character name of Each parameter specifies the one- to eight-character name of the designated modules. Multiple RESIDENT cards may be used to load additional data control sections for reentrant programs such as Input Service or Print Service. If the number of resident copies is insufficient at any given point in processing, additional copies will be loaded as required via ALOAD, and subsequently deleted via ADELETE. RESIDENT cards must never be submitted for members of ASPNUC (which is permanently resident).

```
RESIDENT,MODULE={name|(name,name,...)}
```

Examples:

```
RESIDENT,MODULE=JSS
RESIDENT,MODULE=(PRTDATA,PRTDATA,PN),MODULE=PCHDATA
```

If JSS is made resident appropriate flags are set so no further ALOAD or ADELETE processing for the JSS module will occur.

Note: If OS R/I modules are to be included in the ASP partition via ASP "RESIDENT" cards, place the "RESIDENT" cards for alias modules before the Main module resident cards. This should prevent multiple copies of the same module from being loaded.

Recommended modules to be made resident:

```
MSVDATA (Multiple copies)
RDDATA  (Multiple copies)
PRTDATA (Multiple copies)
MSVINIT
JSS
MSVTERM
MDSALLOC
MDSBRKDN
PN (or Train used most heavily)
```

Further information may be obtained in the section: "Determining What Modules to be made Resident" in Chapter 6.

RESTART

The format of the RESTART card is:

```
RESTART[,ANALYZE=YES][,JOBNO=(1,999)|(start,end)]
```

This card signifies that the Support Processor is to resume execution of the jobs, if any, in its job queues. The internal numbering of jobs resumes where it ceased when the system was stopped, and all space on the ASP direct access storage device that was allocated to existing jobs remains allocated throughout the remainder of the job processing.

If ASP system processing was terminated abnormally due to equipment or system failure, it is possible that a job that was being executed on the Main Processor may not have terminated correctly and will not be processed correctly when restarted. In this case, the job must be returned to the programmer for corrective action. Whenever ASP is terminated abnormally, the event should be noted and ANALYZE=YES specified on the RESTART card for the next system restart.

The ANALYZE parameter specifies that an analysis of the ASP job queue should be made during the restart processing, deleting any job or jobs that would result in an inability to restart the system. The operator is informed of the action taken, and snaps are taken to assist the system programmer. Using restart without analysis, if normal restart fails, the machine can then be restarted with analysis. The snaps are taken to the data set or unit designated by the //ASPABEND DD card. The JOBNO parameter should be coordinated with the COLDSTART JOBNO parameter.

RI

The RI card is used to initialize the OS Reader/Interpreter in ASP. If it is not used, the BLDL function will be null and the PARM options will default.

```
RI,BLDL=(proc1,proc2,...),
 [,TAFETCH={dest|S1}]
 [,DAFETCH={dest|S1}]
```

BLDL=(proc1,proc2,...)

The BLDL keyword identifies a list of procedure names that are frequently used at an installation. A directory entry will be maintained in Main storage for each name in the list that is found in SYS1.PROCLIB on the Support Processor. Directory I/O search time is eliminated for a procedure name that appears both in an EXEC statement and the resident BLDL table.

TAFETCH={dest|S1}

The TAFETCH keyword is used to direct tape volume fetch messages to a console class (S1-S32). The message will be displayed on the console(s) with a destination class that matches the dest operand specified in this keyword [see DEST in the CONSOLE statement for a description of destination classes].

DAFETCH={dest|S1}

The DAFETCH keyword is used for the same purpose as TAFETCH, except it is oriented to direct access volume fetch messages.

Example:

```
RI,BLDL=(ASMFC,ASMFCL,ASMFCLG),TAFETCH=S2,DAFETCH=S3
```

RIDATSTN

The RIDATSTN statement is used to specify a list of data set names that are to be maintained by the ASP Reader/Interpreter in main storage. Setup will be bypassed for all dsnames appearing in the list when they appear as catalog references on a DD statement. If the statement is not used, the function is null.

When the name of a data set appears in a job as a system catalog reference (that is, no UNIT and VOLUME parameters are present on a DD statement), the list of data set names built from the RIDATSTN statement is searched for a matching name. If a match is found, setup for that data set name is bypassed. If a dsname is not found in the list, setup requirements are determined by issuing a LOCATE command to extract unit and volume information from the system catalog of the Main Processor specified in a //*MAIN statement or from the first Main Processor defined by a MAINPROC statement at Initialization time. The data set names cannot be continued to a second card, an additional RIDATSTN card should be supplied.

```
RIDATSTN,data-set1,data-set2,...
```

Example:

```
RIDATSTN,SYS1.PROCLIB,SYS1.LINKLIB,SYS1.MACLIB,SYS1.SVCLIB

RIDATSTN,SYS1.ASPLIB
```

RIPARM

The RIPARM statement is used to pass a set of options to the Operating System Reader/Interpreter when it is being initialized. If it is not used, the option list will default.

```
RIPARM,PARM=(option-list)

PARM=(0xx99905001024905231SYSDA   E00011A)
      bpptttooommmiiicccrlsssssssaaaaefh
```

The PARM keyword specifies a set of options used by the OS Reader/Interpreter. The list must consist of 35 characters and is explained in OS/360 MVT Guide GC28-6720 under "READER". The list shows the defaults used if you do not include an RIPARM statement during initialization. Positions in the list without an underline are always sent to the Reader/Interpreter as shown regardless of what appears on the RIPARM statement. Positions marked "xx" signify it is of no consequence in the ASP environment.

Example:

```
RIPARM,PARM=(0009990500 1C24905230SYSDA    E00011A)
```

RJPLINE

The RJPLINE card identifies to the ASP Support Processor the characteristics of a line to be used for RJP transmissions. Optionally, there are provisions for pre-establishing an RJP terminal to this line.

```
RJPLINE,N=ddname
       ,A=line-adapter-address
      [,I={A|B}]
      [,P=password]
      [,S={2000|line speed}]
      [,F=DIAL]
      [,F=NTRS]
      [,G=Line-group-name]
      [,T=terminal-name]
      [,O=AUTO]
```

N= Line ddname, up to eight alphameric characters. The name may not be the same as the terminal name, RJPTERM card.

A= Line adapter address, three alphanumeric characters.

I=A|B Line interface address, one alpha character with default of A.

P= Line password, up to eight alpha characters. The default is eight blanks. (No password protection).

S= Line baud rating, up to six numeric characters. The default is 2000 baud.

F=DIAL Line feature, when specified indicates a switched line. Otherwise, a dedicated or leased line environment will be assumed.

F=NTRS No transparency feature installed on terminals on this line. If omitted, transparency feature is assumed.

G= The line-group-name, up to eight alphabetic characters. The default is eight blanks (no group name).

T= Preassignment of the given terminal name (5 characters) which is associated with a line. This parameter is restricted to non-programmable terminals, for example, 2780, 2770 or 3780. When using T= no /*SIGNON card should be submitted.

O=AUTO Line option, when specified, indicates the RJP line will be started automatically (no operator START command is required).

Note: The use of multiple F parameters is permissible, for example F=DIAL, F=NTRS.

RJPLINE examples:

Illustration of the RJPLINE card:

1. RJPLINE,N=LINE001,A=050

2. RJPLINE,N=001,A=050,I=B,P=SECRET,S=4800,F=DIAL,T=TERM5 ,T=TERM5,

RJPTERM

The RJPTERM card defines to the ASP Support Processor a remote RJP terminal. Without this definition a terminal signing on under the ASP RJP support will be canceled immediately. This card provides a description of each terminal device along with the operating characteristics of the terminal. A groupname facility is available on a terminal basis.

```
RJPTERM,N=terminal-ddname
       ,T={1130|2770|2780|2922|3780|M202|M205|S360|SYS3}
      [,RD={0|n}]
      [,PU={0|n}]
      [,PR={0|n}]
      [,B={400|nnnn}]
      [,C={0|nn}]
      [,PRW={132|nnnn}]
      [,PUW={80|nnnn}]
      [,F=HTAB]
      [,F=NTRS]
      [,O=BFIX]
      [,G={terminal-ddname|group name}]
      [,F=XBUF]
      [,O=AUTR]
```

RJPTERM,keyword-parameters

N= Terminal ddname, must be five alphameric characters. The name may not be the same as the line name, RJPLINE card.

T= Terminal description type as follows:

2770	-	2770 hardware terminal
2780	-	2780 hardware terminal
2922	-	2922 programmable terminal
3780	-	3780 hardware terminal
1130	-	1130 CPU terminal
SYS3	-	System 3, Model 10 CPU terminal
S360	-	All System/360 CPU terminals, Model 22 and above.
M202	-	Model 20, Submodel 2 CPU terminal
M205	-	Model 20, Submodel 5 CPU terminal

RD= Number of remote reader devices, one numeric character with an upper limit of seven.

PU= Number of remote punch devices, one numeric character with an upper limit of seven.

PR= Number of remote printer devices, one numeric character with an upper limit of seven.

B= Terminal buffer size, up to four numeric characters, and must correspond with the workstation package.

C={0|nn}

Remote terminal console support authority level. The value for nn may be 0 to 15. RJP will convert the value supplied to one of the following four levels:

C = 15 converts to level 15

C = 14-10 converts to level 10

C = 9-5 converts to level 5

C = 4-0 converts to level 0

Levels:

0 - no remote terminal console support

5 - INQUIRY (*I) and MESSAGE (*Z), CANCEL (*C), START (*S), or RESTART (*R) facility.

10 - INQUIRY (*I), MESSAGE (*Z), CALL (*X), START (*S), CANCEL (*C), VARY (*V), MODIFY (*F), or RESTART (*R) facility for jobs submitted by this terminal or devices attached to this terminal.

15 - All

PRW= Maximum print record size, up to terminal buffer size. The default is 132-byte print record.

PUW= Maximum punch record size, up to terminal buffer size. The default is 80-byte punch record.

F=HTAB Terminal printer feature, when specified indicates a 2780/2770 with the horizontal format control feature.

F=NTRS Terminal feature, when specified indicates the transparency feature is not included. Otherwise, the transparency feature is assumed.

O=BFIX Terminal option, when specified indicates the output buffers used by RJP will be dedicated and will not be swapped.

G= Groupname, eight characters or less. The default will be the terminal ddname.

F=XBUF When specified indicates the extended buffer feature (2770 and 2780) is available on the RJP hardware terminal.

O=AUTR Terminal option, when specified will provide the automatic call ASP reader function in the local system.

Note: The use of multiple F or O parameters on the same card is permissible, for example F=XBUF,F=NTRS.

Examples:

1. RJPTERM,N=LOSAN,T=SYS3,RD=1,PU=1,PR=1,B=800

2. RJPTERM,N=NEWYK,T=S360,RD=2,PU=2,PR=4,B=600,C=2, X
PRW=144,O=BFIX,G=SECRET

SELECT

A SELECT card defines the scheduling controls for a particular job selection mode. A job selection mode is assigned to an ASP Main Processor by the SELECT parameter on the MAINPROC card. If this parameter is omitted, a selection mode consisting of SELECT card default values will be used. The selection modes may also be changed dynamically with the ASP MODIFY verb. A SELECT card must define each SELECT mode used on a MAINPROC control card or referenced in an ASP MODIFY command.

```
SELECT,NAME=job-selection-mode-name
      [,MINIT={2|nn}]
      [,MBAR={15|nn|PRTY}]
      [,SBAR={15|nn|PRTY}]
      [,MJSPAN={ALL|nnn}]
      [,SJSPAN={ALL|nnn}]
      [,MPSPAN={ALL|nn}]
      [,SPSPAN={ALL|nn}]
      [,MAGE={NO|YES}]
      [,SAGE={NO|YES}]
      [,MAGER={1|nnn}]
      [,SAGER={1|nnn}]
      [,MAGEL={14|nn}]
      [,SAGEL={14|nn}]
      [,DISPLAY={NONE|ALL|IPL|SIZE|MLOG}]
      [,DPRTY={2|nn}]
      [,SDEPTH={5|nnn}]
      [,INCR={1|nn}]
      [,INCL={14|nn}]
      [,JOBMIX=(n1,n2,n3....,n45)]
      [,CHOICE=({BJOB|,BMIX|,BFIT|,FMIX|,FJOB|,FFIT})]
      [,CLASS={(job-class-name,job-class-name,...)|
               /(job-class-name,...)}]
      [,GROUP={(job-class-group-name,initiator-count,...)|
               /(job-class-group-name,job-class-group-name,...)}]
```

NAME=job-selection-mode-name.

One to eight alphameric characters specifying the name of the job selection mode. The NAME parameter must be the first parameter on the SELECT card.

This name can be referenced using the SELECT keyword on either the MAINPROC control card or the ASP MODIFY command.

MINIT={2|nn}

The maximum number of OS initiators to be started on the Main Processor(s) using this job selection mode; expressed as a one- or two-digit decimal number. The total number of active initiators as defined by the EXRESC parameter in the group cards will not be allowed to exceed the MINIT value. If this value is exceeded the excessive initiators will not be started and an unable to allocate message will be issued.

MBAR={15|nn|PRTY}

A job priority level below which job selection will not be attempted, expressed as a decimal number between 0 and 15. Instead of a priority level, PRTY may be specified to cause every priority level to be treated as a barrier. In the PRTY mode, a scan for jobs in the next lower priority level will not occur until all jobs of the current priority level have been scheduled. If the parameter is omitted, priority level 15 is assumed. For example, if barrier was

set to 10, priority levels 10-15 would always be eligible for scheduling and priority levels 0-9 would not be eligible for scheduling until all jobs in priority levels 10-15 had been scheduled.

SBAR={15|nn|PRTY}

A job priority used to reserve devices for jobs by MDS. Jobs above this priority will be selected first for setup and have their devices reserved. If all of the job's devices are available, it is scheduled for the Main selection process. In the event all of the required devices are not available for a job, it will retain the reserved devices until the next pass by MDS. After all jobs above the barrier have been examined, MDS will setup jobs below the barrier that do not need any of the reserved devices. PRTY may be specified to cause every priority to be treated as a barrier or a decimal number from 0 to 15 may be specified The default is priority 15.

MJSPAN={ALL|nnn} and SJSPAN={ALL|nnn}

The number of jobs in the queue to be examined in selecting a job to be scheduled for MAIN or SETUP, expressed as a one- to three-digit decimal number. If this parameter is omitted or the operand ALL is used, all jobs in the queue, if necessary, will be examined. The default value for ALL is 32,767.

MPSPAN={ALL|nn} and SPSPAN={ALL|nn}

The number of priority levels to be examined in the selecting of a job to be scheduled for MAIN or SETUP, expressed as a decimal number between 1 and 16. If this parameter is omitted or the ALL operand is used, all jobs in the queue, if necessary, will be examined.

MAGE={NO|YES} and SAGE={NO|YES}

If YES is specified, a job bypassed during MAIN or SETUP job selection that is at the top of its priority queue will be put at the bottom of the next higher priority queue. If this parameter is omitted or the NO operand is used, no job aging will occur.

MAGER={1|nnn} and SAGER={1|nnn}

An aging rate, expressed as a decimal number between 1 and 255, specifies the number of times a job must be eligible for aging before its job priority is actually incremented. For example, MAGER=10 means that a job must be passed over during Main job selection and be at the top of its priority queue ten times before that job is put at the bottom of the next higher priority queue. If this parameter is omitted or the value one is used, a job will be aged each time it becomes eligible for aging.

MAGEL={14|nn} and SAGEL={14|nn}

An aging priority limit past which a job cannot be aged; expressed as a decimal number between 0 and 15. For example, MAGEL=10 means that a job will not be aged if its priority is 10 or greater. If this parameter is omitted or the value 14 is used, jobs will not be aged past priority level 14.

DISPLAY={ALL|IPL|MLOG|NONE|SIZE}

Specifies the type of Main Processor messages to be displayed on the MDEST and MLOG operator console. If ALL is specified, every message received from the Main Processor or sent to the Main Processor will

be logged on the MDEST console. If IPL is specified, every message sent or received from Main will be logged when the Main Processor is being IPLed. If MLOG is specified, every message received from the Main Processor or sent to the Main Processor will be logged on the MLOG console. The other DISPLAY options control the message logged on MDEST. If SIZE is specified, the MTSZ001 response from the SIZE command will be logged on MDEST. If the DISPLAY parameter is omitted or NONE is specified, none of the messages sent to Main will be logged on MDEST. In addition, the following messages received from Main will not be logged.

AMTKXXX	MAINTASK messages	IEF281I	Device offline
ASPXXXX	'ASP' prefixed messages	IEF284I	Dataset not deleted
MTFNXXX	Fence CMD messages	IEF285I	Dataset status
ATTRXXX	QASIGN responses	IEF301I	WTR closed
IEEASP0	Size response	IEF384I	Dataset not deleted
IEE301I	Job canceled	IEF429I	Initiator waiting
IEE600I	MCS accepted reply	IEF868I	WTR waiting
IEF161I	RDR closed	IHC002I	Fortran STOP
IEF237I	Device allocated	MTSZ002	Job step change

The following will not be logged if they are for a CTC device.

IEF234X	REMOVE/DISMOUNT MSG
IEF233X	MOUNT MSG
IEF280X	VOLUME KEEP MSGS
IECXXXX	DATA MNGMT MSGS

DPRTY={2|nn}

The priority level assigned to jobs for execution on Main when JPRTY=ASP is used on the CLASS or //*MAIN control card for the job, expressed as a decimal number from 0 to 13. If this parameter is omitted or DPRTY=2 is used, priority level 2 will be assigned to all jobs scheduled on a Main Processor. The dispatching sequence of jobs executing on a Main Processor at this priority level will be dynamically adjusted based upon their ratio of CPU to I/O usage.

SDEPTH={5|nnn}

The maximum number of jobs that may be set up at one time on this Main Processor, expressed as a decimal number from 0 to 255. This parameter limits pre-execution setup so that it will not get too far ahead of the Main Processor. If this parameter is omitted or the value 5 is used, no more than 5 jobs will be setup at one time.

INCR={1|nn}

A decimal number from 0 to 15 that is automatically added to the priority of the job which has been set up. If a job has a priority of 5 when it is set up, and INCR=4 is specified, the job's priority will be elevated to 9 after the devices have been allocated and set up. This parameter expedites the processing of jobs once devices have been assigned to them. If this parameter is omitted or the value 1 is used, a job's priority will be incremented by 1 after set up.

INCL={14|nn}

A setup priority increment limit past which a job's priority cannot be incremented by setup, expressed as a decimal number from 0 to 15. If this parameter is omitted or the value 14 is used, jobs will not be incremented past priority level 14 by setup.

JOBMIX=(iorate-mix-parameters)

One to 45 decimal numbers between 1 and 15 specifying the optimal IORATE job mix for 1 to 15 active initiators. These parameters will update the distributed job mix table which consists of a 3 by 15 matrix. Each matrix column represents the job mix for a particular number of active initiators. Each row expresses the three job counts for low IORATE, high IORATE, and medium IORATE. The JOBMIX parameters enter the table columnwise. That is, the first three numbers are the optimal low, high, and medium IORATE job counts when one initiator is active, the next three numbers are the job mixes when two initiators are active, etc. If this parameter is omitted, the following jobmix values are used:

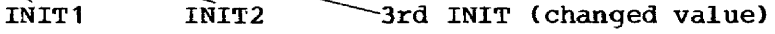
ACTIVE INITIATORS	1	2	3	4	5	6	7	8
Low I/O Jobs	1	1	1	1	1	1	1	2
High I/O Jobs	1	1	1	2	3	3	4	4
Medium I/O Jobs	1	1	1	1	1	2	2	2

ACTIVE INITIATORS	9	10	11	12	13	14	15
Low I/O Jobs	2	2	2	2	2	3	3
High I/O Jobs	5	5	6	7	7	7	8
Medium I/O Jobs	2	3	3	3	4	4	4

When the number of JOBMIX operands required to change an Initiator's jobmix values cannot be contained on a single card, the values may be resumed on a continuation card in another JOBMIX keyword.

Example: To change the JOBMIX for the high I/O jobs for three initiators:

```
JOBMIX=(1,1,1,1,1,1,1,2) or JOBMIX=(,,,,,,,2)
```



Example: To change the JOBMIX for initiators 3 and 4 when no more space exists on the first card:

```
JOBMIX=(2,4,2) ...,JOBMIX=(,,,,,,2,2,2),C
```

CHOICE={(job-selection-choices)|(BJOB,BFIT)}

One to three job selection choices to control the order in which jobs are selected for execution on Main. During a scheduling pass each of the specified scheduling choices are tried until a job is selected or the scheduling choices are exhausted. Six scheduling choices are available; however, only the last three listed below may be followed by alternate choices. A job is defined as meeting the IORATE requirements when the number of active jobs of this job's IORATE is less than the job count in the JOBMIX table for this IORATE and current initiator depth. To determine the Best Mix IORATE, the IORATEs are examined in the following order: Low, High, and Medium IORATE. The first IORATE that can be scheduled based on the JOBMIX table is defined as being the Best Mix IORATE.

- FJOB - The first job in the queue will be scheduled if it fits on Main. Otherwise, no job will be scheduled.
- FFIT - The first job in the queue that fits on Main will be scheduled. If none of the jobs fit on Main, no job will be scheduled.

- BFIT - The largest job in the queue that fits on Main will be scheduled. If none of the jobs fit on Main, no job will be scheduled.

- FMIX - The first job in the queue that fits on Main and also meets any of the current IORATE requirements will be scheduled. If none of the jobs in the queue meet this criteria the next scheduling choice, if supplied, will be tried.

- BMIX - The first job in the queue that fits on Main and has a Best Mix IORATE will be scheduled. If none of the jobs in the queue meet this criteria, the next scheduling choice, if supplied, will be tried.

- BJOB - The largest job in the queue that fits on Main and has a Best Mix IORATE will be scheduled. If none of the jobs in the queue meet this criteria, the next scheduling choice, if supplied, will be tried.

  When determining the largest job in the queue, only the hierarchy 0 region size is examined. Therefore, jobs that execute entirely in hierarchy 1 (LCS) will be treated as though they have a zero region requirement.

If the CHOICE parameter is omitted, CHOICE=(BJOB,BFIT) will be used.

```
CLASS={(job-class-name,job-class-name,...)|/(job-class-name,
       job-class-name,...)}
```

The Job Classes that can be scheduled under this SELECT mode. Or, if all the Class names are preceded by a /, the Job Classes that cannot be scheduled under this SELECT mode.

Example: If only Job Classes A and B can be scheduled under SELECT mode SHIFT2, use:

```
SELECT,NAME=SHIFT2,CLASS=(A,B)
```

Example: If all Job Classes except Class A and B can be scheduled under SELECT mode SHIFT2, use:

```
SELECT,NAME=SHIFT2,CLASS=/(A,B)
```

```
GROUP={(job-class-group-name,initiator-count,...)|
     /(job-class-group-name,job-class-group-name,...)}
```

The Job Class Groups that can be scheduled under this SELECT mode and the number of initiators to be assigned to each of these groups. Or if all the Group names are preceded by a / the Job Class Groups that cannot be scheduled under this SELECT mode.

Example: If only Job Class Group G1 using 3 initiators can be scheduled under SELECT mode SHIFT2, use:

```
SELECT,NAME=SHIFT2,MINIT=3,GROUP=(G1,3)
```

Example: If all Job Class Groups except Group G1 can be scheduled and all Job Classes except Class A and B can be scheduled under SELECT mode SHIFT2, use:

```
SELECT,NAME=SHIFT2,GROUP=/G1,CLASS=/(A,B)
```

The CLASS and GROUP keywords may be repeated on a SELECT card if the Class or Group names cannot fit on one card. However, when repeating these keywords, including and excluding Class or Group names is not

allowed. For example CLASS=A, CLASS=/B is not allowed. Instead use CLASS=A, which includes Class A and excludes all other classes from scheduling, or use CLASS=/B, which excludes Class B and includes all other classes for scheduling. If the CLASS (or GROUP) keyword is omitted from the SELECT card, all Classes (or Groups) will be made eligible for scheduling. All Classes and Groups are eligible for scheduling at ASP Cold Start time unless this status is changed by the initial SELECT mode (see the SELECT keyword on the MAINPROC card). Jobs in a Group that is eligible for scheduling will not be scheduled until the Group's execution resources are allocated. Allocation is controlled by allocation option on the Group control card. The current status of all Classes and Groups and all Select mode parameters is checkpointed so that if ASP is restarted the current Class, Group and Select status will be restored.

SETNAME

The SETNAME card identifies the keywords that may be used in the UNIT subparameter of a DD statement to identify a device or a device class. Through proper organization of device class names, a system programmer can provide the ability for programmers to identify their unit allocation requirements including specific seven- or nine-track tape allocation and channel balancing. Class names for tapes, seven-track tapes, nine-track tapes, shared-channel tapes, tapes on a specific channel, and so forth, can be established, thus providing ample means for the programmer to identify his needs. Refer to Chapter 6 for a further discussion on SETNAME card usage. The keyword parameters are:

```
SETNAME,MTYPE=type
       ,NAMES=(name1,name2,name3,...)
```

MTYPE=type

The type of device the card is describing. This field equates to the MTYPE field in the DEVICE control card. The MTYPE parameter must precede the NAMES parameter.

NAMES=(name1,name2,...,namen)

Identifies all of the possible device and class names that are satisfied by the selection of the device type being described. The names are one to eight characters long, separated by commas, and are specified successively on the card. The names specified must be the same as those specified on the OS SYSGEN macros UNITNAME.

Although a name may appear in more than one SETNAME card, all MTYPE's thus referred to must be of the same type.

The example below illustrates a two-channel, four-tape-drive Main Processor with seven-track and a nine-track drive on each channel. The device type names chosen are those that comply with the SYSGEN generics.

24009C1	nine-track, channel 1
24007C1	seven-track, channel 1
24009C2	nine-track, channel 2
24007C2	seven-track, channel 2

The device classes defined by those cards are:

TAPE	any tape
TAPE9	any nine-track tape
TAPE7	any seven-track tape
1TAPE	any tape on channel 1
2TAPE	any tape on channel 2
24009C1,24007C1	
24009C2,24007C2	specific device identifiers

The SETNAME cards for the above definitions are:

```
SETNAME,MTYPE=24009C1,NAMES=(24009C1,TAPE,TAPE9,1TAPE)

SETNAME,MTYPE=24007C1,NAMES=(24007C1,TAPE,TAPE7,1TAPE)

SETNAME,MTYPE=24009C2,NAMES=(24009C2,TAPE,TAPE9,2TAPE)

SETNAME,MTYPE=24007C2,NAMES=(24007C2,TAPE,TAPE7,2TAPE)
```

Note: The following OS unit names must be specified if allocation of cataloged data sets specifying these units is to be achieved in ASP:

2314	2400-1	3400-2
3330	2400-2	3400-3
2311	2400-3	3400-4
2400	2400-4	

SETPARAM

This card sets a number of setup execution parameters that control the way in which jobs are scheduled by MDS for device allocation and subsequent execution. Each of these parameters applies a separate constraint to the setup algorithm to provide installation control of device scheduling. If the SETPARAM card is omitted, no MDS processing will be provided. Volume fetch operation is independent of MDS and is controlled by the FETCH option on the STANDARDS card. Refer to Chapter 6 for a further discussion of SETPARAM. The keyword parameters are:

```
SETPARAM,DEPTH=nn
        [,ADDRSORT={YES|NO}]
        [,ALLOCATE={AUTO|MANUAL}]
        [,MDSLOG={S1|dest}]
        [,REMOUNT={0|1|nn}]
```

DEPTH=nnn

A decimal number from 0 to 255 that defines the maximum number of jobs that may be set up at one time. This parameter limits pre-execution setup so that it will not get too far ahead of the Main Processor. This DEPTH value applies to the total system (total for all Main Processors). The jobs in execution plus those jobs in SETUP are counted.

ADDRSORT={YES|NO}

This parameter specifies the sequence of the SETUNIT table entries. The entries are always sorted by device type (one type is assigned to each unique MTYPE on DEVICE cards) within Main Processor; however, the ADDRSORT determines whether devices are to be sorted within type. YES indicates units are to be put in order of ascending device address by doing a hexadecimal sort. NO will leave then in the order they appear in the DEVICE initialization control cards.

ALLOCATE={AUTO|MANUAL}

The manner in which device setup is to be performed:

AUTO	As many jobs as possible will be set up concurrently immediately after the GET messages are issued with no operator intervention.
MANUAL	The operator must reply *S SETUP,nnn to the GET messages for a job to enable it to be set up.

MDSLOG={S1|console-dest}

Defines the console class to which error and log messages will be directed.

REMOUNT={0|1|nn}

A decimal number defining the number of times a job will attempt reverification. When errors are detected in volume mounting for a job, mount messages for the volumes in error will be issued until the remount value is reached. The job's devices are then deallocated and the job reenters SETUP. REMOUNT=1 will allow the operator two attempts to correctly mount the job's volumes, the original MOUNT messages and one retry.

An example of a SETPARAM card that limits the depth of setup to ten jobs and logs error messages on the console with a DEST=S4 is:

```
SETPARAM,DEPTH=10,ALLOCATE=MANUAL,MDSLOG=S4
```

STANDARDS

The STANDARDS card establishes basic installation standards. Among these are default parameters, any one of which is applied to a job only if the corresponding information is not otherwise available. Printer resource standards parameters are applied to define STANDARD (the assumed option) in the //*FORMAT and PRINTER control cards. Estimate parameters, system failure and job step options are used if the corresponding fields are omitted from the //*MAIN control card. The priority and class parameters provide default options for the corresponding JOB card specifications, if the class name is a single character (A-O), it may be specified on the JOB card. The remaining parameters establish installation-determined standards that apply to all jobs entered into the system.

```
STANDARDS[,FORMS=forms-name]
         [,CARRIAGE=carriage-tape-name]
         [,TRAIN={AN|GN|HN|PN|QN|QNC|RN|SN|TN|XN|YN|PCSAN|PCSHN|ANY}]
         [,LINES=({1|nnn}[,{WARNING|CANCEL|DUMP}])]
         [,CARDS=({2|nnn}[,{WARNING|CANCEL|DUMP}])]
         [,PRTY={0|nn}]
         [,CLASS={class-name|ASPBATCH}]
         [,GROUP={ASPBATCH|group-name}]
         [,FLOCATE={YES|NO}]
         [,DLOCATE={YES|NO}]
         [,CONSBUF={nnnn|10}]
         [,SQS={nnnnK|3K}]
         [,SEQCHK={YES|NO}]
         [,NOPR={A|C|H}]
         [,NOPU={A|C|H}]
         [,HIARCHY={0|1|D}]
         [,FAILURE={CANCEL|HOLD|PRINT|RESTART}]
         [,JOBSTEP={CHKPNT|NOCHKPNT}]
         [,MLOG={NO|YES}]
         [,DLOG={NO|YES}]
         [,NJPNAME=support-processor-terminal-name]
         [,FETCH={ALL|SETUP|NONE}]
```

FORMS=forms-name

The name of the installation-standard printer forms. This field may be from one- to eight-characters in length.

CARRIAGE=carriage-tape-name

The name of the installation-standard carriage tape. This field may be from one to eight characters in length.

TRAIN={AN|GN|HN|PN|QN|QNC|RN|SN|TN|XN|YN|PCSAN|PCSHN|ANY}

The name of the installation-standard printer train. If ANY is specified, Printer Resource Scheduling will accept any print train mounted on a printer that otherwise fulfills processing requirements unless a specific print train is designated in the //*FORMAT control card for a particular data set.

LINES=({1|nnn}[,{WARNING|CANCEL|DUMP}])

The estimated number of physical lines of data, in thousands, to be printed for a given job. The nnn value may be one to ten digits. The second subparameter specifies the action to be taken when the line estimates are exceeded:

WARNING (or W)	Issue operator warning and continue processing
CANCEL (or C)	Cancel the job
DUMP (or D)	Cancel the job with OS ABEND dump

CARDS=({2|nnn}n[,{WARNING|CANCEL|DUMP}])

The estimated number of cards, in hundreds, to be punched for a given job. The nnn value may be one to ten digits. The second subparameter is identical to that of LINES, above.

PRTY={0|nn}

Standard job priority, expressed as a one- or two-digit value in the range 0 through 14.

CLASS={ASPBATCH|class-name}

The name of the installation-standard job class. This job class will be assigned to all jobs that do not include the CLASS keyword on either the JOB card or the //*MAIN control card. If a class-name other than ASPBATCH is specified, a CLASS initialization control card defining that class is required.

GROUP={ASPBATCH|group-name}]

The name of the installation-standard job class group. This group will be assigned to all job classes that do not include the GROUP keyword on the CLASS control card. If a group-name other than ASPBATCH is specified, a GROUP control card defining that group is required.

FLOCATE={YES|NO}

If YES is specified, Print Service will scan the setup status of each defined Support Processor printer to find a unit on which the required forms, carriage tape, and print train are mounted. This method minimizes the amount of operator intervention, but may increase the number of printers used by a job. If NO is specified, the first available printer that allows forms and/or train changes will be used by each data set. In this event, the amount of operator intervention may be increased, but output will tend to be on the minimum number of printers.

DLOCATE={YES|NO}

Specifies whether the DEST= parameter in the //*FORMAT card is to be honored. DLOCATE=NO causes the DEST= parameter to be ignored, that is, treated as ANY. This option may be used if a disproportionate number of data sets are directed to a given printer, in order to allocate processing more evenly.

CONSBUF={10|nnnn}

Specifies the number, one to four digits, of console buffers to be allocated and formatted. If required buffers are not available from the Console Buffer Pool, they will be obtained via the AGETMAIN routines.

SQS={3K|nnnnK}

Minimum system queue space for OS/MVT, where nnnn is a one- to four-digit decimal number stating the SQS size as a multiple of 1024. ASP will not schedule an MVT job when the minimum SQS is not available.

SEQCHK={YES|NO}

Specifies whether Input Service is to sequence check input object decks. Sequence checking is performed in columns 73 through 80 and requires an increment of 1. If a sequence error is found, a message is written on SYSMSG; after the entire deck is read, the job is sent to Print Service to print SYSMSG and is then purged.

NOPR={A|C|H}

Specifies the action to be taken by Print Service if a //*FORMAT card DEST= parameter assigns a data set to an undefined printer.

A Reassign the data set to any local printer.

C Do not print the data set.

H Put the data set into hold status.

If a data set was put into hold status, the job will be put into HOLD status after all other data sets are processed, and it may be released by the operator, at which time Print Service will ask the operator for the correct printer name. If the data set has been printed or canceled, no operator response is required.

NOPU={A|C|H}

Specifies the action to be taken by Punch Service if a //*FORMAT DEST= parameter assigns a data set to an undefined punch:

A Reassign the data set to any local punch.

C Do not punch the data set.

H Put the data set into hold status.

If the data set was put into hold status, the job will be put into HOLD status after all other data sets are processed and it may be released by the operator, at which time Punch Service will ask the operator for the correct punch name. If the data set has been punched or canceled, no operator response is required.

HIARCHY={0|1|D}

Indicates the number of the hierarchy (0 or 1) from which storage is to be allocated by the AGETMAIN macro. If the D operand is used, storage will be allocated from the hierarchy in which the routine issuing the AGETMAIN resides. Storage will always be obtained from hierarchy 0 when the 2361 Core Storage Unit (LCS) is switched offline.

FAILURE={CANCEL|HOLD|PRINT|RESTART}

Specifies the job recovery option to be used in case of Main system failure. CANCEL cancels the job on Main. HOLD holds the job for restart on Main. PRINT prints the job and then puts the job in hold for restart on Main. RESTART restarts the job on Main.

JOBSTEP={CHKPNT|NOCHKPNT}

Specifies the job step checkpoint option CHKPNT causes a checkpoint to be taken at the end of each job step on Main. This checkpoint contains the current status of all ASP data sets for the job (that is, the JDS record for the job).

MLOG={NO|YES}

If MLOG=yes is specified, all input and output messages will, in addition to their original source or destination console(s), be written on the console designated as MLG in the CONSOLES control card to provide a master log of system activity.

DLOG={NO|YES}

If DLOG=YES is specified, all input and output messages will, in addition to their original source or destination console(s), be written to disk in the OS SYSLOG data set via the OS WTL macro.

Note: This facility should not be specified before first exploring any performance impact caused by the additional WTL overlay.
NJPNAME=support-processor-terminal-name

NJPNAME=support-processor-terminal-name

Specifies the terminal name for this ASP Support Processor for Network Job Processing (NJP). For a line connecting this ASP Support Processor with a remote ASP Support Processor for NJP use, that remote terminal uses this name as the destination terminal name on an NJPTERM card defining the connection.

FETCH={ALL|NONE|SETUP}

The ASP R/I will issue FETCH messages to the console designated for tape and disk setup.

ALL

messages will be issued for all ddnames allocated for ASP managed devices to be setup.

NONE

no fetch messages.

SETUP

Only messages for the ddname specified in the SETUP parameter on the //*MAIN card. If no SETUP parameter is supplied the FETCH parameter will default to ALL.

Example:

```
STANDARDS,FORMS=SINGLE,CARRIAGE=SINGLE6,TRAIN=PN,LINES=(20),X
CARDS=(10,CANCEL),PRTY=0,CLASS=A,NOPR=A,NOPU=A,MLOG=YES,     X
NJPNAME=LAX
```

SYSOUT

The SYSOUT card is used to define SYSOUT classes that will be processed by ASP. The values specified in this card will override the STANDARDS and PRINTER cards value for a class. Refer to Figure 16 for the default values. Use of the SYSOUT card can eliminate use of the //*FORMAT card in many cases. One SYSOUT card is required for each class defined. The MSGCLASS defined on the MAINPROC card must be defined by SYSOUT cards.

```
SYSOUT,CLASS={A-Z|0-9}
      [,DEST={printer-name|punch-name}]
      [,FORMS=forms-name]
      [,CARR=carriage-tape-name]
      [,TRAIN=same as FORMAT card]
      [,CONTROL={PROGRAM|SINGLE|DOUBLE}]
      [,COPIES={1|nn}]
      [,OVFL={ON|OFF}]
      [,INT={YES|NO}]
      [,TYPE={PRINT|PUNCH|USER1|USER2|TSO|DSISO}]
```

CLASS={A-Z|0-9}

Defines the specific SYSOUT class.

DEST={printer-name|punch-name}

Defines a specific printer to print this class.

FORMS=forms-name

Defines the form name to be used for this class.

CARR=carriage-tape-name

Defines the carriage tape.

TRAIN=same as format card

Defines the train type.

CONTROL={PROGRAM|SINGLE|DOUBLE}

Defines the carriage spacing control.

COPIES={1|nn}

Specifies from 0-99 original copies to be printed.

OVFL={ON|OFF}

Specifies the overflow option desired.

INT={YES|NO}

Specifies the interpret option for punch output (3525I).

TYPE={PRINT|PUNCH|USER1|USER2|TSO|DSISO}

Determines what postprocessing action ASP should take. USER1 or USER2 specify that the data generated on the Main Processor should be returned to ASP for postprocessing but no postprocessing action will be taken. This facility may be used for user plotter or paper tape output. TSO specifies that if this job originated at a TSO terminal, the data sets of this class should be returned to the TSO terminal

user. When TSO is specified all parameters other than class are ignored. DSISO indicates that each data set created by ASP Main Service which specified this case is to have its own track allocation table (TAT). That is, tracks allocated by ASPIO routines for a data set of this class will not be allocated from the job's TAT but rather a new TAT will be constructed for this data set's track allocation. Thus a data set of this class can be processed and purged independently of the rest of a job's data sets.

Examples:

```
SYSOUT,CLASS=P,DEST=PR1,FORMS=PAYCK,TRAIN=QNC,CARR=PAY
SYSOUT,CLASS=9,DEST=PUN,TYPE=PUNCH,COPIES=2
```

TRACK

The format of the TRACK card is:

```
TRACK,DDNAME=ddname
```

The TRACK card identifies the number of tracks that have been allocated (and previously formatted) on each ASP direct access storage device. The ddname is used to refer to the space allocated in a DD card. After the direct access storage device has been formatted, this card must be substituted on a one-for-one basis with a corresponding FORMAT card in the initialization COLDSTART deck, and must always be the one that appears in the RESTART deck. When entering new TRACK cards during a RESTART, care should be taken in selecting the new DD name. ASP, during a COLD START will internally sort the DD names from the FORMAT or TRACK cards and will reference these DD names in the sorted sequence. The new DD entries should have DD names that will follow those in the COLDSTART, FORMAT or TRACK cards.

MAINTASK EXECUTION

One of the procedures that will be started on an ASP Main Processor is the MAINTASK (MT) procedure, which will cause MAINTASK to be executed. The starting of the MT procedure will occur during the IPL deck processing phase of ASP initialization. This is accomplished by including the following card in the ASP IPL text:

```
S=S MT,PARM='values',REGION=xxK
```

The values which can be specified in the PARM field and the REGION= field are discussed below.

For a local main system, the IPL deck is not required. If the user elects not to use it, the S MT will be issued automatically by the MSVIPL module. In this case the PARM= and REGION= fields which appear on the //MT EXEC card in the procedure will be used. If there is no PARM= on the S MT command or in the procedure, the default is PARM='ABC'.

The following is an example of the JCL statements which will be needed if all MAINTASK functions are to be used:

```
 1. //MT         EXEC PGM=MAINTASK,PARM='ABCDE',REGION=52K
 2. //STEPLIB    DD  DSN=ASP.STEPLIB,DISP=SHR
 3. //ASPQRDR    DD  UNIT=(CTC,,DEFER),LABEL=(,NL),VOL=SER=ASPQRDR,
    //              DSN=ASP.QRDR
 4. //ASPQWTR    DD  UNIT=(CTC,,DEFER),LABEL=(,NL),
 5. //              DCB=(BLKSIZE=133,LRECL=133,RECFM=FB)
 6. //SYSABEND   DD  SYSOUT=F,SPACE=(CYL,(1,1))
 7. //ASPDATA    DD  DUMMY
 8. //CTCDD      DD  UNIT=(CTC,,DEFER),LABEL=(,NL)
 9. //ADSG01     DD  UNIT=2314,DISP=OLD,VOL=SER,ASPSR1,SPACE=(CYL,(1,1))
10. //ADSG02     DD  UNIT=2314,DISP=OLD,VOL=SER=ASPSR2,SPACE=(CYL,(1,1))
11. //ADCTC      DD  UNIT=(CTC,,DEFER)
12. //OUTDATA    DD  DUMMY
13. //CTCDD2     DD  UNIT=(CTC,,DEFER),LABEL=(,NL),DCB=BLKSIZE=760
14. //SNAPDD     DD  SYSOUT=F,SPACE=(CYL,(1,1))
```

The DD statements in the above example represent all the DD statements required, if all functions of MAINTASK are to be used. Since all functions are not required some DD statements may be omitted as described below.

Required JCL for MAINTASK execution:

```
//MT        EXEC    PGM=MAINTASK,PARM='values',REGION=xxK
//STEPLIB   DD      DSN=ASP.MAINTASK,DISP=SHR
//ASPQRDR   DD      UNIT=(CTC,,DEFER),LABEL=(,NL),VOL=SER=ASPQRD,
//                  DSN=ASP.QRDR
//ASPQWTR   DD      UNIT=(CTC,,DEFER),LABEL=(,NL),
//                  DCB=(BLKSIZE=133,LRECL=133,RECFM=FB)
//SYSABEND  DD      SYSOUT=F,SPACE=(CYL,(1,1))
//SNAPDD    DD      SYSOUT=F,SPACE=(CYL,(1,1))
```

The EXEC card names the program (MAINTASK) to be executed, passes to MAINTASK at execution time through the parameter field the information concerning which subtasks are to be attached and allocates the region size. The PARM and REGION fields can be overriden by the PARM and REGION operands associated with the S MT in the IPL deck used during ASP INITIALIZATION.

The PARM values are as follows:

PARM='A'

Specifies the attaching of the command modules (ASPVER , ASPLOC , ASPFENCE) required to process the VERIFY, SIZE, FENCE, ISOLATE, and LOCATE commands and the modules for the Reader/Interpreter (ASPQALL , ASPQRDR) and the SMB writer (ASPWRITER). 'ABC' is the default and will be assumed if the PARM field does not appear on the EXEC card or as an operand of the S MT command. If a PARM field is present, then the A must be included. If it is not, the command modules will not be attached.

PARM='B' or 'B(tt,ll,hh)'

Specifies the attaching of the dynamic dispatcher (DYNDISP) to monitor the execution of jobs.

tt - Designates, in tenths of seconds, the time interval between the dispatching of the monitor itself. The default is one second (10).

ll - Designates the lowest priority job which will be monitored. The default is zero (00).

hh - Designates the highest priority job which will be monitored. The default is two (02).

PARM='C' or 'C(tt)'

Specifies the attaching of the Channel-to-Channel Monitor (ASPCTCM). This function is not used on a Local main.

tt - Designates, in seconds, the time interval between the dispatching of the monitor itself. The default is 10 seconds.

PARM='D'

Specifies the attaching of the modules required to submit TSO jobs to an ASP system and return TSO output to the TSO terminal operator. These modules are ADSGEN1 , ADSGEN , ASUBMIT .

PARM='E' or 'E(x,y,z)'

Specifies the attaching of the AOUTPUT writer for sending print and punch data sets from a Main Processor output queue to an ASP system for printing and punching, or execution.

x - Designates the output class to be transmitted to Print Service. The default is Class 1.

y - Designates the output class to be transmitted to Punch Service. The default is output class 2.

z - Designates the output class to be transmitted to Input Service. The default is output class 3.

Example 1: PARM='ABCDE' - Starts all functions with the defaults.

Example 2: PARM='AB(20,4,15)C' - Starts the command processor, R/I modules, the SMB writer; the dynamic dispatcher with a two-second interval, low priority of 4 and high priority of 15; and the Channel-to-Channel Monitor with its default.

Example 3: PARM='B(,,5)' - Starts only the dynamic dispatcher with the default time interval and low priority, but changes the high priority to 5. The command modules are not started unless explicitly requested.

Example 4: PARM='C(5)DE(C,D)' - Starts the Channel-to-Channel Monitor with a five-second interval; the TSO modules; and the AOUTPUT writer with a print class of C and a punch class of D. Again, the command modules will not be started.

The REGION requirements will change depending upon the functions which are to be used. The requirements are:

A	MAINTASK, with the command processors, R/I modules, and SMB writer	32K
B	Dynamic Dispatcher	1K
C	Channel-to-Channel Monitor	1K
D	TSO Modules	14K + ASP buffersize
E	Output Writer	6.4K + 2(CTCDD2 Blocksize) + 2(Blocksize of output data set to be transmitted)

Example 1: PARM='ABCDE',REGION=56K. (Assumes 1K ASP buffersize and 1K blocksize of output data sets).

Example 2: PARM='ABC',REGION=34K

Example 3: PARM='A',REGION=32K

Example 4: PARM='AD',REGION=44K (Assume 1K ASP buffersize)

Example 5: PARM='AE',REGION=38K (Assume 1K CTCDD2 blocksize and 1K output data set blocksize).

Additional DD statements are required when the TSO and output functions are to be used.

For TSO:

```
//ASPDATA  DD  DUMMY
//CTCDD    DD  UNIT=(CTC,,DEFER),LABEL=(,NL)
//ADSG01   DD  UNIT=2314,DISP=OLD,VOL=SER=serial-1,SPACE=(CYL,(1,1))
//ADSGnn   DD  UNIT=2314,DISP=OLD,VOL=SER=serial-2,SPACE=(CYL,(1,1))
//ADCTC    DD  UNIT=(CTC,,DEFER)
```

The first two DD statements (ASPDATA, CTCDD) are required for submitting TSO jobs to ASP. The last three are used to send output back to the TSO user. The number of ADSGnn DD statements is not fixed. The user may include as many as he wishes as long as the first four characters of the ddname are ADSG. The ADSGEN module will use them one at a time from first to last and then cycle back to the first again. This allows the user to use as many DD statements as he wishes and have the output distributed to different disk packs.

For AOUTPUT writer:

```
//OUTDATA  DD  DUMMY
//CTCDD2   DD  UNIT=(CTC,,DEFER),LABEL=(,NL),DCB=BLKSIZE=xxx
```

These statements are required for transmitting print and/or punch data sets from a Main Processor to a Support Processor for printing, punching, or execution by ASP. The DCB=BLKSIZE=xxx parameter on the CTCDD2 DD statement is required and the values xxx must be equal to or less than the ASP buffersize on the support system which will accept the data. (This is the ddname that was referred to in the region calculation section.)

In addition to the initialized starting functions at MAINTASK initiator time, the operator can also start additional functions, or stop functions through an operator command. If this facility is permitted, then the region size specified at MAINTASK initiation must be large enough to hold all the modules which will be active at any one time. In addition, the JCL statements which are required for the function to be exercised must have been provided in the MT procedure.

CHAPTER 6. ASP SYSTEM CONFIGURATION DESIGN CONSIDERATIONS

The selection of devices that comprise an ASP system is dictated by installation requirements. Consequently, it is very difficult to state general recommendations concerning equipment configurations. The local system programmer must establish the local configuration requirements and must evaluate them relative to the operating characteristics of the ASP system, augmenting the original equipment configuration with components that will enable the installation to realize the full potential of its configuration. This chapter defines some of the options, both for equipment and for ASP program initialization, that will affect the performance, throughput, and tuning of an ASP configuration.

The physical configuration of the Main Processor is dictated strictly by the programs and applications used. The placement of the Channel-to-Channel Adapter should be made according to the same criteria used in placing two tape units for system input and output. Since the adapter is not subject to channel overrun conditions, it generally should be attached to the lowest priority channel of the Main Processor. This rule applies also to the Support Processor. Refer to Chapter 3, OS SYSTEM GENERATION, for a discussion on CTC lockout conditions under Special Considerations of ASP SYSGEN.

The configuration of the Support Processor is dictated by the number of support functions to be supported simultaneously, by the volume of work to be handled, and by the number and type of telecommunication terminals to be attached. There is no fixed rule concerning the selection of the Support Processor model. The selection is determined by the teleprocessing requirements and throughput objectives of the installation. The equipment decisions for a Support Processor should involve the size of storage, the size of the ASP direct access storage queue device, the selection of console operator terminals, and telecommunication units and are strictly a function of local requirements. However, a shortage of printing and punching output devices tends to backlog the job queue and, as a result, requires a larger direct access storage device queue.

MINIMUM SYSTEM REQUIREMENTS

The configuration requirements for the Main Processor are identical with those of a stand-alone processor operating under OS control program with F-level components except that the channel-to-channel adapter replaces the normal system input and output devices. The modifications to the OS-MVT Control Program nucleus that are required for operation as an ASP Main or Support Processor increase the nucleus by approximately 3000 bytes. Most ASP functions required on the OS Main Processor are incorporated in the module MAINTASK, which operates as a system task on the Main Processor. MAINTASK size is approximately 35K without TSO support and 48K with TSO support.

MINIMUM SYSTEM REQUIREMENTS

The minimum Support Processor for execution is a System/360 Model 50HG or System/370 Model 145HG with two selector channels, an Operator Console, one 2540 or 3525 and 3505 Card Read Punch, one 1403 or 3211 Printer, and two 2314 or two 3330 Disk Storage Drives.

ASP REGION SIZE ESTIMATION

To calculate the ASP region required for a system, add to the ASP nucleus size your typical operating ASP functions. To this total add the control blocks built by initialization. To this sum add any additional storage for user modifications, ASP background utilities, resident modules and a factor for core fragmentation. Make sure you include the proper number of buffers which is a considerable part of the area built by initialization.

MINIMUM REGION CALCULATION

	Size	Buffers
Basic ASP	72 K	7
Includes:		
ASPNUC		
JSS		
PURGE		
PURGE		
CONSOLES		
MDS		
INQUIRY		
First Main Processor (Local)	17 K	
Each additional Main Processor	1 K	
Each Job on a Main Processor	.06K	5
First Card RDR or RJP Remote Reader	11 K	8
Each additional Reader	.7 K	2
First Printer or RJP Remote Printer	10 K	3
Each additional Printer	1 K	3
First Punch or RJP Remote Punch	5 K	3
Each additional Punch	.8 K	3
First OS R/I subtask (R/I DSP)	14 K	7
Each additional OS R/I Subtask	3 K	7
First RJP Terminal (Signed On)	19 K	
Each additional terminal	1.2 K	

REGION REQUIREMENTS FOR TABLES BUILT BY INITIALIZATION

Initialization Card	Parameter on Control Card	Size (Bytes)
BUFFER	BUFSIZE,AMOUNT	BUFSIZE x AMOUNT
	TRACE=YES	2048
	IOBS,RECORDS	IOBS x (32+(Records x 24))
CONSOLE		1 x 188
DEVICE	(EACH)	x 20
	SUPPORT	1 x 44
	MAIN	EACH MAIN x 24
OPTION	ADDSAVE	ADDSAVE x 64
	TRACE	TRACE x 32
PRINTER		1 x 48
RESCTLBK	RQ	RQ x 76
	ASG	ASG x 20

	VUT	VUT x 12
	FCT	FCT x 304
RESIDENT		SIZE OF MODULES
RI	BLDL	EACH NAME x 14
RIDATSTN		EACH NAME SIZE +1
RIPARM		1 x 48
RJPLINE		1 x 28
RJPTERM		1 x 28
SETNAME	MTYPE	1 x 10
	NAMES	EACH NAME x 10
STANDARDS	CONSBUF	CONSBUF x 96
SYSOUT		1 x 30

ASPIO

ASPIO permits a system programmer to design the track format and buffer pool to meet local requirements. A minimum of 40 buffers is usually recommended. Although ASPIO will obtain buffers from available storage when the buffer pool is exhausted, a buffer pool should be established which is large enough to accommodate the minimum requirements for the maximum number of DSP's expected to be active at one time. ASPIO is designed to optimize transmissions of records to disk and, therefore, to permit smaller-than-track size records to be handled efficiently. In single-record mode, the smaller the record, the more efficient the processing of that record and, therefore, the more efficient the scheduling of jobs. Offsetting the requirement for small records for the scheduler is the limitation of Channel-to-Channel transmissions to buffer size. Since blocked transmissions are desirable for performance reasons, the minimum buffer size should be chosen to accommodate the largest record size that the Channel-to-Channel Adapter (CTC) is expected to accept. In addition, the physical device characteristics of the direct access storage devices being used must be considered.

ASPIO permits the system programmer to specify the depth of queuing of disk Input/Output requests at the IOS level. The precise effect of this parameter cannot be stated generally since it depends on buffer pool size and on the balance of direct access storage device input/output activity within a system. Too few input/output blocks (IOB's) will result in some loss of efficiency in transmitting data to the direct access storage devices, while too many will reduce the effectiveness of the priority scheduling of input/ output by ASPIO. Unless local evaluation proves otherwise, it is recommended that three IOB's be specified for every direct access storage device assigned to the ASP direct access storage device pool. For example, if five IBM 2314 disk drives are assigned to ASPIO, then the IOB parameter is set to 15.

GUIDELINES FOR DETERMINING A MINIMUM REQUIREMENT FOR THE ASP BUFFER POOL

Seven to eleven buffers for each active job on a Main Processor

Seven buffers for each OS R/I subtask

Three buffers for each reader

Three buffers for each printer

Ten buffers for other active ASP functions (optional)

In selecting the size of the buffer pool, consideration should be given to the fact that all functions might be active at the same time. It should also be noted that when all functions are active, there will be fewer GETMAIN areas for buffers available due to the number of load modules and CSECT's in the dynamic area.

To dynamically evaluate the buffer usage and storage availability the INQUIRY command may be used to determine the buffer usage and the number of times buffers were obtained via a GETMAIN request. Refer to the ASP Operator's Manual, Chapter 5, INQUIRY for ASP Buffer Pool.

Effective buffer sizes are in the range from 900 to 1400 bytes. A consideration should be made in MVT ASP Support Processors when the dynamic area is being used for buffers. Storage allocation is assigned in 2048-byte blocks when a request for space is made. A buffer size of 1024 bytes or greater would allow only one ASP buffer per 2K block. This would cause storage fragmentation problems in regard to obtaining ASP buffers in the dynamic area. Whereas there would be ample space available in the dynamic area once all the 2K blocks are in use, there would be no areas of contiguous storage large enough to obtain an ASP buffer.

The number of records per track is dependent upon buffer size and is on the BUFFER Initialization control card in the ASP System Programmer's Manual. A minimum number of buffers should be determined using the estimates shown. An effective buffer size can then be chosen based on the number of buffers and the amount of the ASP region to be devoted to the buffer pool. The ASP region will essentially consist of three distinct areas:

- A fixed area which includes the ASP nucleus and all of the generated ASP tables
- The buffer pool
- A free or dynamic area

The dynamic area will be available to all GETMAIN requests and for the loading of all nonresident modules, and is available for use to the buffer pool when no fixed buffers are available for use. The size of this area should depend on the installation's use of the various facilities of the ASP system. With the use of the RESIDENT card in the Initialization deck, heavily used modules can be made resident and will occupy part of this dynamic area. Careful consideration should be given in selecting the size of the ASP region to ensure that active ASP functions are not continually awaiting for storage in the dynamic area. Such a condition will degrade overall performance, and can (and should) be avoided when possible by selecting a region size sufficient to meet all of the ASP facilities that will be used.

OS READER/INTERPRETER UNDER ASP

1. Testing JCL. A facility exists in the ASP R/I controlling DSP (module RICONTL) to allow a job to be processed by the OS Reader/Interpreter (R/I) in order to obtain interpreted JCL, including diagnostic messages when applicable, but to suppress entry into Main Service thereby avoiding Setup and/or Main Processor time. The job's interpreted JCL is printed by ASP Print Service, enabling the programmer to visually verify JCL results before allowing further progression of the job through

the system. This facility is activiated whenever any step of a job contains an EXEC statement with PGM=JCLTEST on it. This is similar to the OS facility of PGM=IEFBR14, except no Main Processor time is used with the ASP facility.

2. User-supplied Exit Routine. There are five user exits provided within the ASP R/I DSP facility. All of these exits must be written in reentrant code and must be included with the linkedit of the ASP module which gives it control. An example of the linkedit control cards would be as follows:

    RIEXITS is the ASP module that gives the exit control. IEFACT is the user exit module.

```
INCLUDE     DD1(IEFACT)
INCLUDE     DD2(RIEXITS)
ENTRY       RIEXITS
NAME        RIEXITS(R)
```

The five user exits described below are referenced by their ASP assembly label. It is the users responsibility to maintain the integrity of the ASP control blocks and verify that OS control block formatting has not changed.

ASPX001: ASP R/I Accounting Routine Exit (Internal Text Exit).

This exit is provided within the ASP module RIEXITS. Control is given to a user module named IEFACT if that name was resolved by the OS Linkage Editor. This exit will provide the user a pointer to the OS internal list that was created from each JCL statement processed by the OS R/I. The internal list will not have been modified by the ASP module RIEXITS (entry point = RIXACCT) before the user gains control. Once the ASP exit and user exit returns, the OS R/I will pass the internal text to its JCL processors (JOB, EXEC or DD). As an extension of the above, it should be noted that if this user exit abends, it will abend the OS R/I subtask under ASP (see ASP R/I subtask failure considerations).

This exit might be taken by a user if he wishes to extract or modify selected JCL parameters before the OS R/I creates control blocks from the internal list. Modifying or extracting JCL parameters can be accomplished via the exit in a convenient keyword scan technique (for more detailed information refer to the OS MVT Job Management PLM, GY28-6660, "Internal List Entry Format" and the assembled listing of RIEXITS, cross-reference label ASPX001).

The user routine, IEFACT, must save registers 0 through 15 upon entry and restore registers 0 through 15 at exit. Return to the caller must be by using register 14. The routine must be reentrant. The following input is available at entry to the user routine:

Reg 1 - address of a four-word parameter list.

Word 1 - Address of the internal text buffer for the JCL statement

Word 2 - Address of the QMPA for the input queue entry

Word 3 - Address of the Queue Manager Routine entry point

Word 4 - Address of the JCL statement image being processed

Reg 12- Address of the Interpreter Work Area (IWA)

ASPX002: ASP R/I Find Routine Exit (Procedure Library Find Exit).

This exit is provided within the ASP module RIEXITS. Control is given to a user module named IEFACTF if that name is resolved by the OS linkage editor.

This exit will provide the user the addresses of the procedure name requested and of the DCB for the procedure library. If the user is taking advantage of the ASP R/I BLDL processing, the user exit will not receive control if the procedure name matched one of BLDL names. If no match was found, it is the responsibility of this user exit to issue the OS FIND macro before returning to the OS R/I. As an extension of the above, it should be noted that a user abend will create an abend in the OS R/I subtask under ASP (see ASP R/I Failure Considerations).

This exit might be taken by a user if he wishes to modify which library a procedure will be found. (For more detailed information refer to the assembled listing of RIEXITS, cross-reference label ASPX002 and the OS Supervisor and Data Management Macro Instructions - FIND).

ASPX003: ASP R/I JCT Exit (JCT Access Facility Exit).

This exit is provided within the ASP module RICBSCAN. Control is given to a user module named ASPEX003 if that name is resolved by the OS linkage editor. The exit will provide the user with the address of the OS JCT (refer to OS MVT Job Management PLM, GY28-6660 - "Job Control Table"). This exit is taken before any ASP R/I DSP JCT analysis occurs; therefore, an abend or an incorrect modification of the JCT in the user module could cause the ASP R/I DSP to abend.

This exit might be taken by a user if he wishes to access or modify the JCT or any OS control block pointed to by the JCT (via TTR). An example of the above might be a user requirement to check the accounting information contained on the JOB card. Since the JCT contains the TTR of the ACT, the user could check the ACT contents for installation standards. (For more details check assembled listing of RICBSCAN, cross-reference label ASPX003.)

ASPX004: ASP R/I SCT Exit (SCT Access Facility Exit).

This exit is provided within the ASP module RICBSCAN. Control is given to a user module named ASPEX004 if that name is resolved by the OS linkage editor. This exit will provide to the user the address of the OS SCT (reference OS MVT Job Management PLM, GY28-6660 - "Step Control Table"). This exit is taken before any ASP R/I DSP SCT analysis occurs, therefore, an abend

or an incorrect modification of the SCT in the user exit could result in the abend of the ASP R/I DSP.

This exit might be taken by a user if he wishes to access or modify the SCT or any of the OS control blocks pointed to by the SCT (via TTR). An example of the above might be a user requirement to monitor region requests within the system. The exit could check all SCT region sizes to ensure that no job would run if it exceeded installation standards. (For more details, check the assembled listing of RICBSCAN, cross-reference label ASPX004.)

ASPX005: ASP R/I SIOT/JFCB Exit (SIOT/JFCB Access Facility Exit).

This exit is provided within the ASP module RICBSCAN. Control is given to the user module ASPEX005 if that name is resolved by the OS linkage editor. This exit will provide to the user, the address of the SIOT and its associated JFCB (reference OS MVT Job Management PLM, GY28-6660 "Step Input/Output Table" and "Job File Control Block"). This exit is taken before any ASP R/I DSP SIOT/JFCB processing occurs; therefore, an abend or an incorrect modification of the SIOT/JFCB in the users exit could cause the ASP RI DSP to abend.

This exit might be taken by a user if he wishes to access or modify the SIOT or JFCB. An example of the above might be a user requirement to monitor the use of demand allocated devices (UNIT=1A0). By looking at the unit field of the SIOT, this exit could cancel all jobs requesting demand allocation. (For more details about this exit check the assembled listing of RICBSCAN, cross-referenced label ASPX005.)

The internal text buffer codes are mapped by macro IEFVKEYS and the IWA is mapped by IEFVMIWA. More detailed information can be found in MVT Job Management PLM, GY28-6660.

3. In-Stream Procedures. It is recommended that an in-stream procedure be used discriminately in a job, since an in-stream procedure can take an inordinate amount of interpreter time to process compared to processing from the procedure library.

4. Multiple Reader/Interpreter. Since the RI DSP processes jobs in a serial fashion, jobs may begin to backlog waiting for the R/I DSP to process them. When this happens consistently at your installation, you may wish to consider increasing the number of R/I functions active simultaneously under ASP. This is accomplished by specifying a DEVICE initialization card with GTYPE=ARI and STYPE=OSRDR for each R/I desired. The R/I DSP may be varied offline and online by using the device name specified on the DEVICE statement for each R/I. For example, if an R/I was specified as DEVICE, STYPE=OSRDR, GTYPE=ARI, SUPPORT=(NONE,OSRI1), you would issue a *V OSRI1,OFF.

5. Updating the Procedure Library data set SYS1.PROCLIB. When you have specified a resident BLDL list for PROCLIB on the R/I initialization statement, you should be aware that if you update a procedure on SYS1.PROCLIB with the same name as one in the list, the updated procedure will not be accessed until the next ASP initialization has been completed.

6. SYS1.PROCLIB alterations for ASP prior to Version 3. It is important that the contents of the procedure library used on the ASP Support Processor be specified as though it were intended to run on a standalone OS system. That is, SYSOUT references must not be converted to UNIT=(CTC,,DEFER) as was done for Main Processors on versions of ASP prior to Version 3. All procedures are referenced on the Support Processor only, regardless of the final Main Processor destination for a job.

7. The ASPNOTE ASPN001 gives the user an idea of how he might obtain the OS Data Set Name on his fetch messages. If this facility is desired, check the assembled listing of RIFETCH at the cross-reference label ASPN001.

8. Main storage considerations. The Resident OS Reader/Interpreter modules occupy 58K of Main storage. The installation has an option to either make the modules resident in the Link Pack Area, or in the ASP region. See Appendix C for a list of which modules must be resident, including alias names. The modules must be resident to prevent severe main storage fragmentation in the ASP region due to LINK,LOAD, and XCTL issued by the OS R/I. If the modules are resident in LPA, the main storage required to start an OS RDR is reduced by 26K; this should be reflected in the REGION parameter of the RDR procedure or in the RDR START command if it is a practice at the installation to occasionally start an OS RDR (for example, S RDR,REGION=22K).

   To make the modules resident in Link Pack Area, you must add the module names to the standard list of reenterable module names in SYS1.PARMLIB (member IEAIGG00), or to an alternate list(s) (IEAIGGxx where xx can be any two alphameric characters). See MVT Guide, GC28-6720 "Using Link Pack Area" for further discussion of how this is done.

   To make the modules resident in the ASP region, you must add the names to RESIDENT statements at ASP initialization time. This technique also requires the REGION parameter on the ASP EXEC statement to be increased by 58K to accommodate the additional storage required.

   If the modules are not resident in the Link Pack Area, it will be necessary to increase MINPART by 6K. This happens because the REENTRANT OS R/I modules coming from SYS1.LINKLIB during START command processing fragment subpool storage in such a way to require 6K of additional region space. This additional MINPART requirement is eliminated after START command processing is complete (that is, when the program that was started begins execution). You can supply the new MINPART at SYSGEN in the SCHEDULR macro, or at IPL in response to the IEA101A SPECIFY SYSTEM PARAMETERS message (MIN keyword).

9. The R/I SMF exit (IEFUJV) will be activated within the OS R/I subtask if SMF is included and activated within the support processor. When the exit is activated during OS R/I subtask initialization, either the user's copy or the IBM copy of IEFUJV will be loaded into the ASP region.

   If the user can gain Access to the OS IWA (Interpreter Work Area) on entry to his SMF exit, a means of referencing ASP control blocks does exist. The field 'IWANELJC' in the IWA contains a pointer to the "RIDATA" of the ASP R/I DSP. Examples of information contained within "RIDATA" are as follows: jobname, ASP assigned job number, ASP TVTABLE pointer, SYSMSG FDB pointer, and the user reserved save area.

10. The ASP R/I DSP has the responsibility of supplying ASP with the region requirements of all OS jobs. The basic logic in this area is to obtain the region size from the first OS Step Control Table (SCT). The region parameter at this point is the same one the normal OS system runs from; that is, the normal OS rules of region determination of a step apply. The user should be aware that if there is no region specified on the OS JOB card and if increasing region parameters are specified on successive OS EXEC cards, his job can wait for region space in his second thru nth steps while running on the OS main. This condition is not unlike the normal OS response to increasing region sizes.

SUPPORT DEVICE GROUPING

General description of the use of device group name (GNAME parameter on DEVICE card) in ASP. The GROUP name is used on the ASP device initialization control card as a means of grouping together input and output devices. The normal reason for grouping a pool of I/O devices is physical or geographical location of these devices. The purpose of device grouping is to insure that job output will be directed to appropriate output devices within the group from which the job originated. Some examples of where grouping might be desirable are:

1. A large installation has readers, printers, and punches on more than one floor of a building or it has a remote reader, printer, and punch on a channel repeater some distance away from the local installation. Through assignment of a GROUP name on the appropriate device cards in the ASP initialization deck, job output for jobs submitted from a device on a floor within the building or from a device attached to the channel repeater would be directed to an output device in the same group (that is, physical location).

2. By default all devices attached to an ASP RJP terminal have the terminal-name as their groupname. Thus, output of a job submitted from an RJP terminal is returned to the same terminal. However, more than one RJP terminal may have the same groupname; this is specified via the G= parameter on the RJPTERM ASP initialization control card. Thus two or more RJP terminals in close physical proximity may share the same groupname with the result that a job submitted from any RJP terminal in the group may have its output directed to any appropriate device on any terminal in the group. This provides an automatic balancing of output activity between the terminals in the group. Another use of this type of grouping would be to group one high-speed terminal on a high-speed line with one or more low-cost, low-speed, dial line terminals. The dialup terminals would be used essentially as input devices and the majority of job output of jobs from these terminals would be processed on the high-speed terminal. This approach, if applicable to the user installation, could result in both line cost and terminal cost savings.

3. The use of device groups in ASP is dynamically alterable via the ASP MODIFY operator command (see ASP Console Operator Guide, MODIFY verb) or via the ORG= parameter which may be specified by the application programmer on the ASP //*MAIN control card. (See ASP Application Programmer's Manual, //*MAIN card.) The default for groupname on DEVICE cards in the ASP initialization deck, when it is not specified is "LOCAL". It should be noted that an ASP input device (RDR, TA7, TA9) should not be defined with a groupname unless there are appropriate output devices (printers, punches, etc.) that have the same groupname.

4. A unique group name which may be used in "ANYLOCAL". Specifications of ANYLOCAL on a device card in the GNAME= parameter (Input Devices only) will allow output for jobs submitted from these devices to be processed on any local (non-RJP) device of the appropriate type. This parameter may also be specified via the ORG= parameter on the //*MAIN card or the DEST= parameter on the //*FORMAT card. This group name (ANYLOCAL) should not be referenced in any assignment by group (see ASP Operator's Manual, under MODIFY verb).

Explanatory diagrams of the foregoing discussion follow.

1. Installation Device Configuration/Location.

```
Physical
Location:

FLOOR1          READER-NAME=RD1,GROUPNAME=FLOOR1
                PUNCH-NAME=PU1,GROUPNAME=FLOOR1
                PRINTER-NAME=PR1,GROUPNAME=FLOOR1
                PRINTER-NAME=PR2,GROUPNAME=FLOOR1


FLOOR2          READER-NAME=RD2,GROUPNAME=FLOOR2
                PUNCH-NAME=PU2,GROUPNAME=FLOOR2
                PRINTER-NAME=PR3,GROUPNAME=FLOOR2


Devices         READER-NAME=RD4,GROUPNAME=CHANREP
attached to a   PRINTER-NAME=PR4,GROUPNAME=CHANREP
channel         PUNCH-NAME=PU3,GROUPNAME=CHANREP
repeater


RJP terminal    READER-NAME=RM001RD1,GROUPNAME=REMOTEA
Name=RM001      PRINTER-NAME=RM001PR1,GROUPNAME=REMOTEA
                PUNCH-NAME=RM001PU1,GROUPNAME=REMOTEA
physically      TERM TYPE=S360,LEASED LINE,HI-SPEED
near each
other

RJP terminal    READER-NAME=RM002RD1,GROUPNAME=REMOTEA


RJP terminal    READER-NAME=RM003RD1,GROUPNAME=RM003
name=RM003      PRINTER-NAME=RM003PR1,GROUPNAME=RM003
                PUNCH-NAME=RM003PU1,GROUPNAME=RM003
                TERM TYPE=S360,LEASED LINE,HI-SPEED
```

In the diagram for example:

Jobs inputted from RD1 (FLOOR1) would have their output directed to output devices on FLOOR1 only (PU1,PR1, or PR2).

Jobs inputted from RD2 (FLOOR2) would have their output directed to output devices on FLOOR2 only (PU2, PR3)

Jobs inputted from RD4 (connected to channel repeater) would have their output directed only to output devices also on the channel repeater (PU3, PR4)

Jobs inputted from remote readers RM001RD1 or RM002RD1 would have their output directed to the RM001 output devices. (RM001PU1, RM001PR1) i.e., RM002 has no output devices but shares groupname REMOTEA with RM001.

Jobs inputted from remote reader RM003RD1 would have their output directed to RM003 output devices. (RM003PU1, RM003PR1).

A job submitted from any of the input devices previously described which contained in its JCL a //*MAIN control card with ORG=ANYLOCAL may have its output directed to any of the appropriate local (non-RJP) devices, (PU1, PR1, PR2, PU2, PR3, PU3, PR4).

MAIN DEVICE SCHEDULING

Main Device Scheduling (MDS) is requested by the presence of the SETPARAM initialization card. To omit MDS, do not include the SETPARAM card. All tables (i.e., SETNAMES, SETUNITS) will be generated if needed but no MDS processing will be possible.

The operation of the MDS is affected by the parameters chosen for the ASP initialization cards: DEVICE, SETPARAM, SETNAME, and SELECT.

DEVICE

The MAIN subparameter of the DEVICE card contains a console class for the device being defined. By using the same destination for devices in a specific location, messages pertaining to these devices will be routed to the specified class. For example, given two 2314 facilities and one tape:

```
DEVICE,MTYPE=2314,MSTATUS=DA,MAIN=(130,SY1,S5,ON)
        •
        •
        •
DEVICE,MTYPE=2314,MSTATUS=DA,MAIN=(137,SY1,S5,ON)
DEVICE,MTYPE=2314,MSTATUS=DA,MAIN=(230,SY1,S6,ON)
        •
        •
        •
DEVICE,MTYPE=2314,MSTATUS=DA,MAIN=(237,SY1,S6,ON)
DEVICE,MTYPE=24009,MAIN=(180,SY1,S7,ON)
```

messages could be split specifying:

```
CONSOLE,DDNAME=CNDISKA,....DEST=S5
CONSOLE,DDNAME=CNDISKB,....DEST=S6

CONSOLE,DDNAME=CNTAPE,....DEST=S7
CONSOLE,DDNAME=MDSLOG,....DEST=SGROUP
```

The MSTATUS subparameter allows the specification of device removability and expected volume serial. These parameters, normally defaulted to removable (RM) and no volume serial, can be used to indicate a managed device that contains a permanently resident volume. This will inform MDS of the volumes present prior to the Main Processor being IPLed. For example:

```
DEVICE,MTYPE=2314,MSTATUS=(DA,PR,JOBLIB),MAIN=(230,SY1,S5,ON)
```

would establish the JOBLIB volume as permanently resident on Main Processor SY1 unit 230 and prevent MDS from requesting it to be mounted on another system.

Note: When the Main is IPL'ed the status of all managed devices is interrogated and overrides specifications established by the DEVICE card at initialization.

Volume removability may be specified as ND indicating that no disposition checking is to be done for the volume. All references to this direct access volume will be treated as if they were DISP=SHR even though they are not, and the volume is considered as ASP mounted. Care should be exercised in the use of this option since data set contention and/or invalid volume updating (in a shared DASD environment) since normal MDS checking is bypassed.

In a multiple main configuration problems can occur if duplicate volume serials exist for direct access devices. If a pack must be on more than one main and cannot be physically placed on a shared device, the devices should be defined as "logically" shared and made non-removable (either ASP mounted or OS permanently resident or OS reserved). For example:

```
DEVICE,MTYPE=2314,MSTATUS=DA,MAIN=(230,SY1,S1,132,SY2,S2)
```

If this is not done jobs referencing the duplicated volume will not be setup and will remain in the MDS mount queue.

SETPARAM

The ADDRSORT subparameter can be used to dictate the order in which MDS looks at devices in attempting allocation. This is accomplished by coding ADDRSORT=NO and results in the SETUNITS table(s) being ordered in the same sequence as the DEVICE cards in the initialization deck. This may be useful in cases when device locations are not physically ordered by ascending device address.

SETNAME

It is required that all names specified in the NAMES subparameter must be those specifiable as UNIT parameter in the OS DD card. That is, all generic names for managed direct access and tape devices must be included in SETNAMEs.

By specifying the same names in a different order for the same MTYPE it is possible to create a "preference" order of device selection. This preference might be to achieve channel separation if possible while still allowing this job to run if separation can not be achieved. For example with devices on three channels:

```
DEVICE,MTYPE=2314CH1,MSTATUS=DA,MAIN=(130,SY2,S1,ON)
DEVICE,MTYPE=2314CH2,MSTATUS=DA,MAIN=(230,SY2,S1,ON)
DEVICE,MTYPE=2314CH3,MSTATUS=DA,MAIN=(330,SY2,S1,ON)

SETNAME,MTYPE=2314CH1,NAMES=(DACH1)
SETNAME,MTYPE=2314CH2,NAMES=(DACH1,DACH2)
SETNAME,MTYPE=2314CH3,NAMES=(DACH1,DACH2,DACH3)
SETNAME,MTYPE=2314CH1,NAMES=(DACH2,DACH3)
SETNAME,MTYPE=2314CH2,NAMES=(DACH3)
```

Requests for DACH1 would attempt allocation on Channel 1 then Channel 2 and finally on Channel 3. Similarly requests for DACH2 would attempt allocation first on Channel 2 then Channel 3 then Channel 1.

Although a name may appear in more than one SETNAME card, all MTYPEs applying to the name must be the same type (either DA or TA).

Note: To use devices outside of MDS control define OS names to include the desired devices and then do NOT include these names as SETNAMES.

SELECT

Several subparameters on the SELECT card affect the operation of MDS allocation on a Main Processor basis (SDEPTH, SBAR, SPSPAN, SJSPAN, INCR, INCL, SAGER, SAGEL, SAGE). Through these MDS allocation may be biased toward one Main Processor (a larger SDEPTH, SPSPAN and SJSPAN), devices may be pooled but still made more available to one Main Processor (a higher SBAR), and jobs on a specific Main Processor may be favored for selection (a higher INCR, INCL and AGEing parameters).

Note: The DEPTH parameter on the SETPARAM card applies to all jobs in setup while SDEPTH is for one Main Processor only.

DJC DESIGN CONSIDERATIONS

For integrity of complex networks, it is suggested that a prototype of the net be created using the specified DJC control cards with IEFBR14 jobs and executed in a production environment.

HOW TO CHANGE INTERNAL TABLES

INPUT SERVICE READERS

Some of the default values used by the readers phase of Input Service are set by EQUATE statements placed at the beginning of the RDINISH module assembly. These are:

RDIBTCHS=10 — Batch size - number of OS jobs per ASP Input Service interpreter job

Range: 1 to n $n = \frac{\text{ASP buffer size - 12 - L'JDSFSIZE}}{\text{L'JDSVSIZE}}$

A buffer size of 792 bytes, for example, gives a maximum n=19. For a 2K buffer, n=50. For a 4K buffer, n=101.

RDIMXJDS=50 — Absolute maximum batch size permitted. The actual maximum used is the lesser of RDIMXJDS or a maximum that is internally calculated (as for RIDBTCHS). An operator over-specification results in the use of the "actual maximum" herein defined.
range: 1 to n

RDIBLKCR=5 — Number of CCW's to be chained for CR. A value greater than 5 requires restructuring the data CSECT. "One" is equivalent to *X CR,C.
range: 1 to 5

RDIBLKTR=1 — Default tape blocking factor. If most tapes read by TR have the same blocking factor, respecify this value to save operator entry of the TBLK= parameter.
range: 1 to 200

```
RDIPRTY=15          Priority of scheduling the ASP interpreter job.
   range:  0 to 15  Reducing this value has two possible benefits.  It
                    reduces the size of the ASP job queue, since each
                    interpreter job represents (RDIBTCHS) OS jobs.
                    And it reduces the ASP DASD space for these jobs,
                    since an interpreted job requires several files.
                    The chief disadvantage is that these uninterpreted
                    jobs are inaccessible to all of the system
                    outside of the interpreter, and thus cannot be
                    modified or inquired upon.

RDITAPMD=X'C3'      Default tape mode = 9/1600.
   other:           'C8' = 9/800, '93' - 7/800, '53' = 7/556,
                    '13' = 7/200.

RDITAPND=X'07'      Tape unit action at EOF = Rewind.
   other:           '03' No Rewind, '0F' Rewind-Unload.

&DRGPORG SETC 'ANYLOCAL'   The 'device group' for DR.  In order to get
                           output from jobs entered via DR at least one
                           output device (punch, printer) must be in the
                           same group.

&POLYGRP SETC "ANYLOCAL"   The attributed group for a POLYASP reader.
```

DETERMINING WHAT MODULES TO BE MADE RESIDENT

ALOADS - ALOAD Statistics DSP

The ALOADS DSP is designed to assist the system programmer in choosing which modules are candidates to be made resident in the ASP region of main storage. When the DSP is activated it accumulates statistics for each ALOAD macro issued within the ASP region. The data includes module name and how many times each module was referenced by an ALOAD. The statistics are summarized and printed on the SUP console when the ALOADS DSP is canceled.

```
To invoke ALOADS     -   *X,ALOADS
To start recording   -   *S,ALOADS
To stop recording    -   *C,ALOADS (causes data printout)
                                        Signon
```

The following module residency suggestions are based on account experience and use of the ALOADs DSP. They are suggestions only and each user should evaluate their own requirements in this area.

Reason	Suggested Resident
High Main Processor Activity	MSVINIT MSVTERM MSVQMAP MSVDATA (1 per ASP scheduled initiator)
High INQUIRY/MODIFY ATIVITY	INQDRVR MODDRVR WTDJCT
High Job Setup Activity	MDSALLOC MDSBRKDN MDSVERFY
High Reader/Interpreter Activity	RICONTL RICBSCAN RIFETCH
High Print Service Activity	PRTSETUP PRINT PRTOUT PRTDATA (1 per normally active printer (local or RJP))
High Input Service Activity (see also High Reader/Interpreter Activity)	RDLOGIC CR RDDATA (1 per normally active input device (local or RJP))

Generally, unless some unique situation dictates otherwise, the JSS module should definitely be made resident.

CHAPTER 7. WRITING DYNAMIC SUPPORT PROGRAMS

INTRODUCTION

Support functions of the ASP system are implemented by Dynamic Support Programs (DSPs). DSP's are multiprogrammed ASP system components that are scheduled via the ASP Job Segment Scheduler (JSS). DSP's can be directly related to a job's execution, such as, Reader/Interpreter, Main, or can be a background utility, such as, Card-to-Tape. All of the resources of ASP are available to the DSP's.

The user has the option of defining and writing his own DSP's. When defining a function or program to be written that is not directly related to JOB execution the user should consider all of the options available for executing the function. Some of the options are:

- Writing the program to execute as a normal OS job
- Writing the program and entering it as a procedure in SYS1.PROCLIB
- Writing the program to execute as a DSP in the ASP region

An advantage of using either of the first two options is in the event of an abnormal termination. ASP, having the responsibility of scheduling many Main Processors concurrently as well as the supporting functions, could be impacted by a failing DSP. The DSP should be completely checked out under control of POLYASP before being placed in a production system.

ASP has provided a feature, Disk Reader, that can read members of a partitioned data set and then have them submitted to ASP for scheduling as jobs on a Local or Real Main Processor. This is comparable to OS starting jobs via the START command, that is, START INIT. Jobs started via the Disk Reader can take advantage of the 'HOTJOB' feature in ASP.

GENERAL DISCUSSION

Time sharing in the ASP program is accomplished via repetitive relinquishing of control to the ASP Multifunction Monitor. The multifunction monitor's allocation of control is based upon the priority assigned to the DSP when it was introduced into the ASP system. Consequently, the DSP's with the highest priorities have more opportunities for CPU time than do DSP's with lower priorities.

A DSP is scheduled for execution when a job requiring its function is ready for processing and the required resources (printer, card reader, tapes, and so forth) if necessary for scheduling, are available. The Job Segment Scheduler scans the Job Control Table (JCT) for jobs that are ready for processing. This scan occurs when a job enters the system (for example, from a card reader), or when a support function terminates and returns to the Job Segment Scheduler. When all devices that are necessary for scheduling for a DSP are available, the DSP is loaded from the system residence device (using the OS LOAD facility via the ASP ALOAD macro).

When the DSP is activated, it becomes a support function, and an entry is made in the Function Control Table (FCT) for that function. This entry contains the general register save areas and the return point for the function when it relinquishes control. (Control is relinquished by a support function when the function issues an AWAIT macro.) In

addition, the FCT entry contains a pointer to the Active Job Description/Accounting Block Table (Active JDAB Table) for the scheduled job. This FCT entry also contains the Scheduler Element sequence number for the scheduled DSP.

A DSP communicates with the Job Segment Scheduler, through a series of tables, to locate the job and the data to be processed. The primary table that identifies a job is the Job Description/Accounting Block (JDAB). A JDAB entry is created for each job by Input Service or the call DSP function when the job enters the system. The entry contains job control information, such as job name, job number, and estimates of execution parameters; it also contains the Scheduler Element for each segment of processing for the job. In addition, the JDAB contains a pointer to the Job Data Sets block (JDS). The JDS contains the File Description Block (FDB) information and the data set name for each data set created for the job.

When a DSP becomes active, it locates it's scheduler element and the associated parameter cards (located in conjunction with the Scheduler Element). At entry to the DSP, the Support Processor devices specified by type in its Device Requirements Table entry may have already been allocated to it. If these devices satisfy the processing requirements of the DSP, no GETUNIT is required. If, however, the parameters associated with this job specify by name units other than the ones assigned, the DSP should return the devices via PUTUNIT macro and obtain the new set of devices by means of the GETUNIT macro. The discussions of GETUNIT and PUTUNIT in Appendix A describe the various means of obtaining and releasing devices.

Data sets to be processed are located by a pointer in the Job Description Accounting Block to the Job Data Sets Block. The JDS contains an FDB and a data set name for each data set associated with the job.

If a driver module is being executed, it must ALOAD (and later ADELETE) any other modules it requires during execution. However, a reentrant program that requires a discrete data area for each execution does not load its own CSECT. The Job Segment Scheduler performs this task, since the name of the CSECT is included in the DSP Dictionary. The Job Segment Scheduler loads the appropriate CSECT address into register 13 prior to execution of the reentrant DSP. The only required statement to address this CSECT is:

```
        USING   CSECT-name,R13
```

Additional storage, if required, is obtained by use of the AGETMAIN macro-instruction, and unit record and tape devices are opened using the ASPOPEN macro-instruction.

Once the DSP has verified its parameters and has successfully obtained its devices, it issues a signon message to notify the operator that the job is on the system, and then issues the LOGIN macro-instruction. LOGIN establishes the means by which the operator may communicate with the DSP and obtains the DSP's time-on the system via the OS TIME macro-instruction.

The DSP has now satisfied all its preprocessing requirements, and processing of the data sets may commence. Processing should consist of short segments (one or two milliseconds). For most DSPs these breaks occur naturally. If lengthly computation is required, however, AWAIT macros should be interspersed at regular intervals. This permits DSP's of higher priority to execute. A DSP designed for lengthy computation must be assigned a low priority to ensure that the higher priority DSP's have adequate time for execution.

When processing is complete, the DSP issues a LOGOUT macro-instruction to terminate operator communication and to post the JDAB with the DSP's time-on and time-off. It then uses the APUTMAIN and ADELETE macro-instructions to free storage and to delete any processing modules that it ALOADed, and closes its devices using ASPCLOSE. Finally, the DSP updates the JDS, and JDAB, returns them to the direct access storage device, and returns control to the Job Segment Scheduler.

Note that specialized ASP macro-instructions ALOAD, ADELETE, AGETMAIN, APUTMAIN, ASPOPEN, ASPEXCP, ASPCLOSE, and ASPDCB are to be used in place of the corresponding OS functions.

Return to the Job Segment Scheduler may be made under circumstances other than normal end-of-job. Erroneous DSP parameters that preclude processing should be indicated by a console message, followed by an immediate return to the Job Segment Scheduler. If the required devices are not available when the job is scheduled, the DSP may cancel the job so that it may be reentered into the system at a later time, or it may return to the Job Segment Scheduler for specialized rescheduling, in which case, the DSP is rescheduled when the devices become available for its use. A DSP may allow the operator to issue a *RESTART or *CANCEL command during processing, or it may be forced to terminate prematurely because of unrecoverable error conditions. For consistency of accounting operations, a DSP should always issue a LOGOUT instruction if it has commenced processing, regardless of the reason for termination. However, if the *CANCEL command is received as a reply to the signon message, the LOGOUT should be omitted to avoid time posting, and the Job Segment Scheduler will assume the function of terminating operator-DSP communication.

When the DSP returns control to the Job Segment Scheduler, any devices allocated to the DSP are released, the DSP is deleted, its CSECT (if any) is deleted, and the Job Segment Scheduler performs end-of-function processing. The DSP may return to the Job Segment Scheduler with one of the following return codes in Register 15:

Code	Definition
0	The function is completed. The Scheduler Element in the JCT for this job and function is set complete, and this job is returned to the queue.
4	The function has returned for later scheduling. The Scheduler Element in the JCT for this job and function is set not-active and not-complete, and the job is returned to the queue in operator hold status. It must be released for further processing to occur.
8	The function has returned for specialized rescheduling. The rescheduling requirements may be in a buffer or the DSP's FCT GETUNIT list if two or less devices are required. Register 1 must contain the address of the buffer or list. If a buffer is used the first four bytes of the buffer must be zero followed by list(s) containing the reschedule requirements in the format of the GETUNIT list. The job will be rescheduled when the device requirements in the list can be satisfied.
12	The FCTJPURG return to JSS by a DSP indicates that all job scheduler elements except PURGE are to be set complete and the job should be purged from the system without any further activity.

Code	Definition
16	The function has returned for canceling with print. The Scheduler Elements in the JCT for this job for all functions except Print and Purge are set complete, and this job is returned to the queue.

PROGRAMMING CONSIDERATIONS

The writing of a DSP is governed by a number of programming conventions unique to ASP. These conventions were established to implement the multiprogrammed environment in the most efficient manner possible. They include general register assignments, loading conventions, table referencing techniques, DSP communication conventions, and restrictions on the use of OS Control Program Services.

DSP's are coded in the OS Macro Assembler Language. All macros and resident ASP programs have been programmed with the following general register assignments:

Register	Description
0-1	Parameter registers -- not saved by called program
2-9	DSP scratch registers -- saved by called program
10	Standard base register for all DSP's
11	Pointer to Function Control Table entry for DSP
12	Pointer to ASP Transfer Vector Table (TVTABLE)
13	Used by DSP to establish CSECT base, or as a pointer to a save area for OS
14	Return address of calling program
15	Address of called program

Registers 0 through 9 and registers 13, 14, and 15 are available to a DSP except when the DSP is calling another program, either an ASP resident program, an OS supervisor program, or a subprogram of the DSP. Because registers 0, 1, 14, and 15 are used in program calls, it is advisable to restrict the use of these registers.

Register 10 is the standard base register for all DSP's. DSP's may be coded with other base registers, but adherence to the standard base register convention will facilitate program checkout.

Register 11 points to the FCT entry for the DSP. The Multifunction Monitor depends upon the correct setting of register 11 at all times to maintain control of the system. Register 11 must never be altered by a DSP.

Register 12 locates the TVTABLE. Since all DSP's and their subprograms are loaded dynamically, they must not contain unresolved symbols. Consequently the TVTABLE is designed to provide a communication link between the DSP's and the ASP resident functions. All ASP macros depend upon the TVTABLE to provide this communication. Note that the ASP system must be totally reassembled whenever the order of the entries in the TVTABLE is changed.

To provide flexibility and insulation over subroutine calls in the ASP system, a linkage routine was written to save and restore registers over subroutine calls. The calling sequence was implemented by three macros:

1. ACALL: generates linkage through ASAVERTN to any other ASP subroutine.

2. ARETURN: generates return linkage with return codes back through ASAVERTN to the original calling routine.

3. SAVENTRY: generates a map of the register save area used by ASAVERTN.

The save areas are obtained from one of three sources: (1) a chain of save areas reserved for an FCT, (2) a pool of save areas available to all FCT's, and (3) a save area obtained by AGETMAIN. Reserved save areas are obtained by AGETMAIN and are chained off the FCT prior to their use. The pool of save areas is built at ASP initialization time as specified by the ADDSAVE parameter on the OPTIONS card. AGETMAIN save areas are obtained as needed.

All tables in the ASP system are referenced via dummy sections (DSECT's). This convention facilitates changes to these tables and permits the reference to these tables to be standardized. Reference to table fields assuming contiguity or a given order in the table should not be made. All DSECT's are controlled by ASP macros. These macros are always placed at the head of a program for ease of reference and should be followed by a control section (CSECT) for the program being assembled. A TVTABLE macro, which establishes the DSECT for the TVTABLE, should be included for every program in the ASP system.

DSP's must be written as serially reusable modules or reentrant. They are loaded via the ALOAD macro and may not contain any unresolved symbols. It is desirable to organize a DSP into a serially reusable or reentrant driver and one or more serially reusable or reentrant subprograms that may be dynamically loaded. When dynamic subprogram loading is used by a DSP, the loading must be via the ALOAD macro. The DSP is responsible for deleting (via the ADELETE macro) all subprograms that it loads.

Task switching in the ASP system is accomplished via the AWAIT or AWAITOFF macros. These macros signify that the DSP is waiting for an event to be completed before the DSP can proceed. A DSP with a long compute cycle (several milliseconds) should issue an AWAIT macro, with the event it is waiting for already posted as completed. This permits a DSP with higher priority to execute. The normal time interval between AWAITs is one-to-two milliseconds. (Note that the ASP Input/Output routines will usually cause an AWAIT macro to be used as each buffer is emptied.)

All OS Control Program Services are available to ASP; however, a DSP should not use a WTO, WTOR, WAIT, or LINK macro, nor use any component of OS that uses one of these macros. Note that the queued access methods (QSAM or QISAM) use these macros. The use of these macros disrupts the flow of processing on the Support Processor and may cause a degradation in system performance, by possibly causing ASP itself to wait. Care should be taken also to ensure that any OS function that is used does not cause a system ABEND following a noncatastrophic error exit.

CONSOLE SERVICE CONSIDERATIONS

All DSP operator communication is effected through the ASP Console Service routines. Messages are routed to operator consoles by Console Service via the MESSAGE macro on the basis of the console number and/or the destination code (DEST). Each console maintains bit indicators for the message destination codes it accepts.

For two-way communication with the operator, Console Service permits a DSP to define a program entry point for accepting asynchronous entries. This entry point is defined via the LOGIN macro. The asynchronous program entry point must be in the resident portion of the DSP or the DSP's CSECT and must be programmed to permit entry at any time during processing. The asynchronous program provides the interface between the Console Service routine and the DSP. After LOGIN, the operator may refer to a DSP by using the ddname or unit address of any of its associated (via GETUNIT) devices. The DSP name itself may also be used if there is only one copy of the DSP currently logged in.

Messages sent by an operator from a console may be answered via the MESSAGE macro. This macro permits a reply to be routed back to the operator console of origin. The console number of the originating console is in the input console message format.

ASP INPUT/OUTPUT CONSIDERATIONS

The ASP Input/Output programs (ASPIO) provide complete management of the direct access storage devices assigned to the ASP system. The program provides dynamic direct access storage device storage allocation, buffering, blocking, and deblocking for DSP's on sequential data sets. In addition, there is a single record transmission facility for accessing the disk resident tables of ASP, such as the JDAB and JDS.

A 20-byte control block, the File Description Block (FDB), is established for each multiple record data set to be read or written. The FDB is referenced on all calls to ASPIO programs in order to define the data to be processed. The ASPIO programs work in a locate mode under control of the programmer. The programmer is responsible for performing all the necessary data moves into or out of the buffer pool after the ASPIO programs have located the space. In addition, the programmer is responsible for issuing an AOPEN and the corresponding ACLOSE macro for all data sets referenced, as well as for ensuring that there is a corresponding [illegible] macro for each ALOCATE macro that is issued.

Occasionally the ASPIO programs are unable to respond immediately to an input/output request and are forced to issue an AWAIT for the calling DSP. This AWAIT is issued internally, outside the control of the DSP programmer. Consequently, the programmer must allow for this AWAIT to take place whenever an ASPIO macro is used and must remove any time-dependency from that area of his program.

ASP tables are read from and written to disk by a separate set of input/output macros. Since these tables are single record data sets, their treatment differs from that of normal ASP data sets. The FDB associated with these tables is only six bytes in length. Since the tables are not read in a buffered mode, there is no requirement to open or close them. Rather, the AREAD macro calls for the read of a track, and the AWRITE macro rewrites the track to disk. If a table is read and is not updated (and consequently does not have to be rewritten), it may be released via the ARELEASE macro. An AREAD must be followed by either an AWRITE or an ARELEASE. To create a new single-track record, an AWRITE macro is issued with zeros in the last two bytes of the FDB, and the data buffer address stored in the first four bytes.

EXAMPLES OF ASPIO USAGE

The following examples illustrate the use of ASPIO in the four basic modes: single-record read, single-record write, multiple record read, and multiple record write. There are, of course, many optional uses of these macros; the examples below illustrate only the most common, direct usages of ASPIO:

- Single-record read. This example illustrates the reading of a single-record data set. It is assumed that a six-byte single-record File Description Block (FDB) has already been created, such as a parameter buffer FDB in the Job Description Accounting Block (JDAB). Assuming that location AFDB contains the FDB for the record to be read, the following sequence should be used:

```
LA          R1,AFDB
AREAD       FDB=(R1)
    •
    •
    •
LA          R1,AFDB
ARELEASE    FDB=(R1)
```

  The AREAD causes ASPIO to get a buffer and to read the required record. Upon return from AREAD, the address of the buffer that contains the record is in the first four bytes of the FDB. After processing is complete, the user should either write the buffer back with an AWRITE or release it, as illustrated, with ARELEASE.

- Single-record write. The example below illustrates the writing of a single-record data set. To create a new single-record data set, a programmer must use a six-byte FDB with the last two bytes set to zero to signify to ASPIO that this is a new single-record data set. In addition, the programmer must obtain a buffer from ASPIO via the GETBUF macro and must place the address of the buffer into the first four bytes of the FDB. On the other hand, if the record to be written had been read previously, the FDB that was used to read the record should be used so that the record will be replaced in its original location. This example illustrates the writing of a new record and assumes that an FDB, AFDB, exists. The write sequence is:

```
AGETBUF
ST          R0,AFDB
LA          R1,AFDB
AWRITE      FDB=(R1)
```

  When the instruction sequence above has been executed, the record will have been transcribed to disk and the buffer will have been returned to the buffer pool.

- Multiple record read. All multiple record disk data sets are controlled by ASPIO, which is responsible for track allocation, blocking of logical records into physical records, buffering, and physical disk input/output. The ASPIO program works in locate mode, and it is the responsibility of the user to perform the necessary processing tasks on the data as it resides in the ASPIO buffer. Each successive locate for data releases the previous logical record; consequently, either the ASPIO user must complete processing of the current record before requesting the next record, or he must move the current record out of the ASPIO buffer in order to retain it.

  A 20-byte FDB, which is usually located in the Job Data Sets Block (JDS) for the job, is used for controlling multiple record

transmissions. Processing is initialized by the use of the AOPEN macro, after which the programmer must issue an ADEBLOCK for each successive record he reads. When processing is complete, the ASPIO user must issue an ACLOSE macro to terminate reading.

In the following example, it is assumed that the FDB is located at AFDB, that the DSP requires low priority, and that transmission is to be single record. The instructions necessary to read the data are:

```
   LA          R1,AFDB
   AOPEN       FDB=(R1),PRTY=3,TYPE=IN
 X LA          R1,AFDB
   ADEBLOCK    FDB=(R1),EOF=Y,EOD=Z (not illustrated)

   (Register 1=location of first byte of record)

   (Register 0=byte count of record)
       •
       •
       •
   B           X
 Y LA          R1,AFDB
   ACLOSE      FDB=(R1)
```

- Multiple record write. To write a multiple record disk data set, the ASPIO user follows a procedure that is very similar to that for reading. The FDB, AOPEN, and ACLOSE processes are identical. However, in using locate mode to write, the ASPIO user issues two macros per logical record instead of one. The first macro, ALOCATE, locates the required space, after which the user must move the data to be written into the located area. The second macro, ABLOCK, signals to ASPIO that the data has been moved. The byte count for ALOCATE must always be equal to or greater than the ABLOCK count. As in previous examples, the following sample program assumes an FDB at location AFDB and, in this case, a count in a field COUNT.

```
   LA          R1,AFDB
   AOPEN       FDB=(R1),PRTY=2,TYPE=OUT
   LA          R1,AFDB
   L           R0,COUNT
   ALOCATE     FDB=(R1),COUNT=(R0)
```

(Register 1=pointer to the first byte of the located area, and the user must now move data into that area.)

```
   LA          R1,AFDB
   L           R0,COUNT
   ABLOCK      FDB=(R1),COUNT=(R0)
```

(Data is now blocked into the buffer, and ASPIO is ready to accept another request.)

```
   LA          R1,AFDB
   ACLOSE      FDB=(R1)
```

ASSEMBLING A DSP

All components of the ASP system are maintained in data sets called ASP.SOURCE and ASP.MACROS. These data sets contain the program modules and the macros unique to ASP. When existing modules are to be modified, the OS IEBUPDTE program is used to move and to update the module to be assembled into a scratch data set prior to assembly. If the module to be assembled does not exist in ASP.SOURCE, it is placed directly into

the input stream. In either case, during the assembly phase, the OS Macro Library (SYS1.MACLIB) is concatenated with ASP.MACROS to provide access to the ASP macros.

Note: All ASP modules are to be linkedited with the "RENT" parameter in the linkedit step. When writing a DSP an excellent reference source would be an ASP DSP of similar function.

DSP CHECKLIST

The following list represents the requirements for writing DSP's:

- The REGISTER macro-instruction or equivalent equate statements must be used.
- All ALOCATEs must have associated ABLOCKs even if the count is zero.
- DSP's are to be serially reusable or reentrant.
- An ALOAD macro is to be used when subprogram modules are to be loaded.
- Dynamically loaded subprogram modules are to be written as reentrant programs.
- All modules that are ALOADed must be deleted via the ADELETE macro prior to termination.
- DSP's may not have unresolved external references. Access to resident components is achieved via the Transfer Vector Table.
- DSP's must issue a LOGIN for operator communication and must support all verbs with a positive response (that is, a DSP may not ignore a console message).
- Upon termination, a DSP must issue a LOGOUT unless it has done no processing.
- A DSP must never cause the ASP system to terminate abnormally. Unrecovered error conditions should be signaled to the operator for a decision concerning the appropriate action. The user should take advantage of the Failsoft facility.
- In general, all unit record input/output should be performed at the execute channel program (EXCP) level, using ASPOPEN, ASPEXCP, ASPCLOSE, and ASPDCB.
- DSPs that execute without interrupt for more than a few milliseconds should issue dummy AWAITs to allow higher priority DSPs to execute.
- The OS macros WTO, WTOR, LINK, and WAIT must not be used, directly or indirectly.
- OS functions that use ABEND as an error exit must not be used, directly or indirectly.

DSP INITIALIZATION

The following list represents the recommended sequence of required tasks for the initialization phase of a DSP: (This outline is only a guide since, at times, steps are omitted or the sequence is varied to reflect particular processing requirements.)

1. Establish base register for DSP.
2. Locate JDAB for the job via the Active JDAB Table (AJDAB).
3. Read the JDAB.
4. Locate Scheduler Element for the support function and read the parameter buffer, if any.
5. Extract necessary information from the parameter buffer and release the buffer.
6. Inspect devices that were assigned (if any) and request reassignment of devices via GETUNIT as required. Cancel or request specialized rescheduling if required devices are unavailable.
7. Locate and read the Job Data Sets Control block (JDS), if applicable.
8. Extract pertinent FDB(s) for data set(s) to be processed and release the JDS, if applicable.
9. Release the JDAB.
10. Issue a signon message.
11. Issue the LOGIN macro-instruction.
12. Begin execution.

DSP TERMINATION

The following list represents the recommended sequence of required tasks for terminating processing by a DSP:

1. Close all open data sets.
2. Read the JDAB.
3. Read the JDS.
4. Update the FDB(s) for all data sets written.
5. Write the JDS.
6. Post required accounting information (lines printed, cards punched, etc.) in the JDAB.
7. Issue LOGOUT macro-instruction.
8. Write JDAB to disk.
9. Load JSS return from TVT table.
10. Return to the Job Segment Scheduler with the appropriate completion code in R15.

REQUIREMENTS FOR WRITING DYNAMIC SUPPORT PROGRAMS FOR RJP

The code required for initiation of I/O to a local device is the same as to a remote device. That is, the user must issue an ASPOPEN, ASPEXCP, and ASPCLOSE sequence. Use of the ASPDCB control block macro is

required for interface standardization between the user and RTAM. The RJPCAN parameter in the ASPEXCP macro must specify a close exit path, meaning the user must issue an ASPCLOSE. For issuing console messages to and from the remote terminal, the ASSOCNTL=D console number will be in the allocated SUPUNITS entry. Data chaining is only supported for print type devices, and command chaining is supported for all output devices. The Writing Dynamic Support Programs Chapter within this manual provides a detailed discussion of DSP standardized conventions.

For support of callable DSP's using both remote input and output devices, the user must follow the following logic. Prior to opening of any devices, all START processing commands must be satisfied (that is, *S CC in response to the CC logon message). The output must be ASPOPEN prior to any input device.

DSP FAILSOFT

ASP DSP Failsoft provides an interface to allow recovery from program checks which would otherwise result in loss of the ASP system. The interface is provided through use of the SPIE facility of OS and a set of user-supplied loadable recovery modules. The SPIE exit routine will save the program check status and cause an entry to the retry scheduler code upon return to OS. The retry scheduler calls ASPABEND and then ALOADs and calls the retry driver module. Upon return, a check is made of the return code to determine if the failing DSP should be retried or terminated and the appropriate action is taken.

Modules:

AFSDRVR: ASP Failsoft Driver - Controls the flow between the recovery modules. ALOADS, CALLS, and ADELETEs the general and/or special recovery routines and returns to the retry scheduler with a return code specifying retry or terminate. The code is passed to the driver module by the recovery routines.

AFSINIT: ASP Failsoft Initialization - Initiate the recovery CSECT and notify the operator of the failure. Causes control to pass to AFSDC, AFSRCVY, or AFSxxxxn special recovery module depending on availability of special recovery modules and operator input.

AFSDC: ASP Failsoft Dump Storage Interface - Provides the operator with the information necessary to call the ASP DC DSP in order to attempt recovery by displaying and/or altering storage. The operator may cause control to go to the driver and a retry or to the standard recovery routine.

AFSRCVY: ASP Failsoft Termination - Frees units belonging to the DSP. ADELETEs loaded modules and data CSECTs where possible.

AFSTERM: ASP Failsoft Termination - Restores the DSP for a retry attempt.

AFSxxxn: DSP Failsoft Speicalized Recovery Modules - Special modules written to handle unique recovery considerations for specific DSP's. xxxx is replaced by a unique name from the DSP dictionary for the DSP and n is the suffix supplied by the DSP in FCTFSSUF field 01 for FCT to indicate which load is desired. n may be any alphameric character other than blank and must be placed in the FCT dynamically.

ABENDMON: Contains SPIE and STAE exits and retry scheduler.

AFSCOMN: Issues messages to the recovery console when no action is required. The message must be moved to location AFSMSG and has a maximum length of 65. Linkage is BAL R9, AFSCOMN.

AFSCOMA: Same as AFSCOMN except action type messages are issued. Linkage is BAL R9,AFSCOMA. The address of the action buffer is saved at location AFSBUFSV and is returned in R0. It is the callers responsibility to issue a DEQMSG macro at the appropriate time.

AFSWAIT: AWAITs for one of two events: 1. An operator input message which is moved to AFSMBUF and bit AFSMPEND in byte AFSECF Set. 2. An ATIME interval expires in which case bit AFSTPEND in byte AFSECF is set. Linkage is BAL R9,AFSWAIT.

Programming Interfaces:

AFSDRVR ALOADs, ACALLs, and ADELETs each module required to handle a specific error. Addressability to called modules is provided in register 10 and addressability to the AFSDRVR CSECT is provided in register 13. Return to AFSDRVR is on Register 14 with an offset. The meaning of each offset is as follows:

Offset Return		Function
0	-	Recovery was unsuccessful
4	-	Recovery was successful, retry the DSP
8	-	Schedule the next specialized recovery module
12	-	Terminate the recovery

For a module to use offset 8, the suffix of the next module to receive control must be put in the FCTFSSUF field of the FCT. The FCT is addressable on register 11. Registers 0 and 1 are preserved and are passed to the next module called. Thus parameters can be passed between modules.

The offset 12 return is for use when a catastrophic error occurs. The system does not attempt any further recovery. The DSP is permanent AWAITed immediately.

There are three subroutines in the AFSDRVR CSECT for use of the recovery modules. They are:

1. A message routine when no action is required by the operator.
2. A message routine when action is required of the operator.
3. A WAIT routine for an ATIME completion or a console message.

Macros:

AFSDSECT - Maps the recovery work area and generates AFSDRVR CSECT.

FAILDSP - Generates linkage to the retry scheduler to cause a DSP to fail. Used by ASP routines to fail a DSP when invalid data or usage is detected.

Work Area:

The Failsoft work area is mapped and generated by the AFSDSECT macro. When TYPE=CSECT is coded, AFSDRVR is generated which is also the work area. The format is as follows:

1. AFSDRVR control routine.

2. Message routine where no action is required.

3. Message routine where action is required.

4. AWAIT routine for use of recovery modules.

5. ATIME exit routine.

6. Message acceptance routine.

7. Constants and work cells for recovery modules including message buffers.

8. Area for saved FCT fields.

9. Area for STATUS at time of failure.

   a) Completion code

   b) PSW

   c) Registers 0 through 15

CHAPTER 8. POLYASP

POLYASP permits concurrent execution of the ASP system in two or more regions of the same processor. Each region contains the complete ASP system and therefore each region will run independently of the others.

POLYASP is particularly useful in system testing. For example, a production version of ASP might be running in one region while system programmers are checking out a test version of ASP in another region. A further use of POLYASP is in NJP (Network Job Processing) testing. It is now possible for one POLYASP region to send ASP jobs over TP lines to a second POLYASP region.

Only one POLYASP region can run in the local Main Processor mode.

Running POLYASP requires an ASP Initialization deck for each POLYASP region. In addition, the following changes should be made to each deck:

1. Provide a different ASP queue and ASP checkpoint data set for each POLYASP region.

2. Allocate the ASP output data sets (ASPOUT, ASPABEND, and ASPSNAP) to a different device for each POLYASP region.

3. Determine the ASP support devices to be used by each POLYASP region. These devices may be defined in more than one initialization deck, but only one region can use a particular device at a time.

To facilitate ASP system checkout in a POLYASP environment, production-type jobs are frequently required for execution under the test version of ASP. To accomplish this checkout, a POLYASP reader capability may be used to pass ASP jobs from one POLYASP region to another. The ASP jobs are transferred in card-image form only; no ASP control blocks are passed. This method allows complete ASP release independence. If a control block or buffer size is changed in one of the POLYASP regions, ASP jobs may still be interchanged between the ASP regions with the POLYASP reader. The technique is as follows:

The POLYASP region from which jobs are to be taken is called the sending region. The Dump Job (DJ) DSP is called in this region specifying the receiving POLYASP region as the output device (see below for details).

The POLYASP region in which the jobs are to be executed is called the receiving region. The Input Service (CR) DSP is called in this region specifying the sending POLYASP region as the input device.

The following ASP Initialization control cards are necessary to use the POLYASP reader facility.

1. A device card defining the sending POLYASP region as an input device. On this card the first four characters of the device name must be POLY. Also, GTYPE must be PLY; STYPE must be POLYI; and GNAME must be the OS job name of the sending POLYASP region.

2. A device card defining the receiving POLYASP region as an output device. The first four characters of the device name must be POLY. In addition; GTYPE must be PLY, STYPE must be POLYO, and GNAME must be the OS job name of the receiving POLYASP region.

As an example, one ASP Initialization deck has an OS jobname of EX50PLY1 and the POLYASP region is called POLY1; this is the production version of ASP. The second ASP Initialization deck has a jobname of EX50PLY2 and the POLYASP region is referred to as POLY2; this is the test version of ASP.

The additional control cards required for deck EX50PLY1 are:

```
DEVICE,STYPE=POLYI,SUPPORT=(NONE,POLY2),GTYPE=PLY,GNAME=EX50PLY2
DEVICE,STYPE=POLYO,SUPPORT=(NONE,POLY2OUT),GTYPE=PLY,GNAME=EX50PLY2
```

The additional control cards required for deck EX50PLY2 are:

```
DEVICE,STYPE=POLYI,SUPPORT=(NONE,POLY1),GTYPE=PLY,GNAME=EX50PLY1
DEVICE,STYPE=POLYO,SUPPORT=(NONE,POLY1OUT),GTYPE=PLY,GNAME=EX50PLY1
```

To send production jobs from POLY1 to POLY2, the following would be entered on a POLY2 console:

```
*X CR,IN=POLY1
```

And the following would be entered on a POLY1 console:

```
*X DJ,OUT=POLY2OUT
```

Communication is now established between the two POLYASP regions. All output options of DJ are available when operating in the POLYASP reader mode. For example, to send all jobs in the POLY1 (production) queue to the POLY2 (test) region, this command should be entered:

```
*S DJ,Q
```

This command will cause DJ to transfer all production jobs in POLY1 to the CR DSP in POLY2. CR will then enter each job into the POLY2 queue.

As a testing aid, two new DJ output parameters, HOLD and PURGE, have been added. The default is PURGE, in this event each job is purged from the ASP queue after it has been transferred. The HOLD option will put the job in hold status (in the sending region only) after the job is transfered. These options may be used in the START command after any DJ output parameter. For example, to send ASP job 0053 to POLY2 and also keep the job in hold status in the POLY1 queue for later execution, this command should be used:

```
*S DJ,J=53,HOLD
```

POLYASP READER RESTRICTIONS

The current DJ rules for dumping jobs apply when using the POLYASP reader. That is, a job cannot be dumped if it has been set up or if it is active in a DSP function. In addition, the POLYASP reader will transfer only input data sets; therefore, a job that was sent to another POLYASP region for execution cannot be transferred back to the originating region for printing or punching.

It should be noted that DJ will transfer card images only when operating in the POLYASP reader mode. The normal mode of DJ is to dump ASP control blocks and data records to tape.

When the POLYASP reader facility is used, the TCBUSER field is used in the sending POLYASP region's TCB.

Dummy Main allows ASP jobs to be processed by the Main Device Scheduler and Main Service without the presence of either the setup devices or a

Main Processor. This feature is intended for ASP system checkout and is particularly useful in a POLYASP environment.

A dummy Main Processor is defined by specifying SYSTEM=DUMMY on the MAINPROC control card. All other MAINPROC parameters, as well as definition of any Main Processor SETUP devices, are specified as though the dummy Main were a real Main Processor. These SETUP devices, however, are treated strictly as dummy devices.

When a dummy Main Processor is used, module MSVDUMMY is loaded in place of module MAINIO. All other modules remain unchanged. MSVDUMMY replies to messages sent by ASP to the dummy Main Processor; these replies are assembled as part of the MSVDUMMY module. When a job is scheduled on the dummy Main Processor, the JCLIN data set is duplicated into the SYSMSG data set. The job is then terminated on the dummy Main Processor.

CHAPTER 9. DEBUGGING AIDS IN ASP

The following discussion describes the debugging aids available in an ASP environment.

ASP ABEND DUMPS

This discussion assumes that the first step taken is to obtain a dump of Support Processor storage and that the reader is familiar with the dump format. Dumps may be obtained in one of three ways; standard OS ABEND dump, standalone dumps, and the ASP *DUMP command. When a catastrophic error is detected by OS, an automatic dump is initiated if possible. If such a dump is not possible, a standalone dump (using, for example, the IBM standalone dump programs IMDSADMP) must be taken. On the other hand, if the system is operating abnormally and the operator wishes to initiate a dump, the *DUMP command may be issued. If this command is ignored, the standalone dump may be used.

OS system dumps contain status codes indicating the reason for the dump. ASP-initiated dumps (via ABEND) provide user codes as documented in the ASP Messages and Codes, Manual. All other codes are OS codes. These codes are identified in IBM System/360 Operating System, Messages and Codes (GC28-6631). If the user specified DUMP=ASP/OS in his ASP Initialization deck OPTIONS card, all ASP control blocks and tables are formatted and written to the //ASPABEND DD data set, if DUMP=SA is coded, a core-image dump is taken. When formatting core image dumps (standalone dumps), with IMDPRDMP be sure there is a JOBLIB or STEPLIB DD card pointing to the ASP load module library. This is required in order to get the ASP control blocks formatted. One load module for IMDPRDMP (IMDPRFSR) is modified by ASP and must be used instead of the OS version. System dumps will identify, through the load list or request blocks (RB's), the currently active and in-core programs and their beginning locations. In the case of a program check, the next doubleword following the interrupt location is the program check old PSW, followed by 16 words containing the contents of the general registers at the time of the interrupt. With this information and with the program and linkage editor listings for ASP, the programmer can begin to isolate the problem.

In most cases when a dump is obtained, Register 10 is the base register of the DSP in control at the time, and Register 11 contains the FCT entry for the DSP. Each FCT entry contains the contents of registers that are saved for the DSP, the base register for the DSP, the AWAIT mask, and the event completion flag address. Therefore, by means of the FCT entries, it is possible to determine the status of all active DSP's. The FCT entry also contains the address of the DSP Dictionary entry for the DSP. By tracing the DSP Dictionary entry, the programmer may identify the DSP associated with this FCT entry. The FCT entry also contains the address of the CSECT (if any) associated with the DSP, and a pointer to the GETUNIT List for the DSP, which, in turn, contains pointers to the Support Units Table entries. Each FCT entry also contains pointers to the following FCT entry, so that the progress of the job may be traced.

Location 54 (hexadecimal) contains the address of the OS trace table. A scan of this table will obtain the last SVC issued, the start input/output address, and the device-end interrupt conditions for devices that have or had input/output activity. Each entry contains either the device address or the SVC number; it also contains the

contents of registers 15, 0, and 1 when interrupt input/output activity was issued.

Finally, in any active FCT entry, Register 12 contains the address of the ASP TVTABLE. This table contains the addresses of all resident programs, such as all internal ASPIO routines, GETUNIT, PUTUNIT, LOGIN, and LOGOUT; it also contains pointers to some tables, such as the Support Units and System Units Tables, the DSP Dictionary, the Main Processor Control Table, and the first and last entries in the FCT queue.

Certain conditions may cause the ASP system to produce a deliberate ABEND dump. In each of these cases, the reason for the dump may be identified by the user completion code. The ASP Messages and Codes Manual documents the meanings of these completion codes.

In the event of a non-critical DSP failure, the ASP system will not terminate. Instead, DSP Failsoft will interrupt the DSP and allow a system programmer to use the DC DSP for further analysis of the failure.

In the event ASP abnormally terminates, a dump will be provided as defined in the OPTIONS initialization card.

The following are excerpts from a sample dump created by using the DC DSP:

The DC (Dump Core) DSP produces a dump by calling ASPABEND with the PSW printing to *CALL. In other types of failures the invalid instruction or an ASP DMxxx completion code will be printed on line four. The bottom line of the first page gives the active FCT at the time of the dump and its address.

CONSOLE STATUS TABLE

The Console Status Table lists all of the consoles defined via the CONSOLE initialization card. The last entry (location 178FA0) is created internally to be used by the ASP routines to communicate with each other without having the message printed on an operator console. Console CN1 (DDNAME) has seven untyped messages, which are listed with location specifying the console buffer address. By examining the format of the untyped messages for console CNPR2, you will see that it is the MLOG console.

MAINPROC TABLES

A Main Processor Control Table is created for every Main Processor defined by a MAINPROC initialization card. The table for SY1 (location 142040) lists the last two messages sent or received by that Main Processor. By examining the DSTL number it can be seen that the last reference to the CTC was for the console (70). This Main Processor has a job, job number four. The RESQUEUE entry for this job is at location 17EC68.

PRINTER RESOURCE TABLE

The Printer Resource Table is created as a result of the PRINTER initialization card or DEVICE cards referring to 1403 or 3211 printers.

SUPPORT UNITS TABLE

The SUPUNITS Table defines all of the devices to be used by the Support system. The SUPPORT, GTYPE, STYPE, and GROUP parameters are supplied on the DEVICE initialization card. Each SUPUNITS entry points to the entry in the SYSUNITS entry. The SYSUNITS table contains all devices defined to ASP by the DEVICE cards.

SETNAMES TABLE

The SETNAMES Table is created by the SETNAMES initialization card. It's here that the names supplied are correlated to a type to be used in the SETUNITS Table. The SETUNITS Table describes all of the devices on a particular Main Processor that are managed by ASP. There is a SETUNITS Table for each Main Processor defined. The MTYPE parameter on the DEVICE card is used to generate this table.

SYSUNITS TABLE

The SYSUNITS Table printout shows that unit 180 is shared by the Support and Main Processor. The volumes mounted on these devices are also listed along with their Main Processor and status.

ASP I/O TRACE TABLE

To read the ASP I/O Trace Table, read across the line. Look at the routine and caller that is indicated at the end of the table. The caller is ISDRVR, to be determined by locating the return address (CALLER column) in a storage map (location 135156). It can be determined that this was a read request by following the entries. The sequence is the READ, OPEN, Disk, GETBUF which gets the buffer located at 16F14C, GETIOB and the address of the IOB is 164A78. At this time the I/O operation is started. After it is completed the IOB is returned via the PUTIOB, IOB 164A78 and then back to DISK to determine if any more work is to be done. By comparing the buffer address on the GETBUF and the buffer pool it can be seen the buffer is out of the pool. By comparing the address of a buffer gotten with an address in the buffer pool it can be determined if it was obtained by a GETMAIN.

ASP MAIN STORAGE MAP

ASP will sort the subpool entries and free core elements by address-starting with the low storage address. The module names within a subpool are indicated with their size. The change may be used to determine how Main Storage is utilized. The enclosed example is only partial.

ASP TRACE TABLE

TRACE specified on the OPTIONS initialization card will create the ASP TRACE Table. This table displays the entries into the ASAVE routine and the passing of control by the Multifunction Monitor. This table is read across. The first entry is a call by MAIN, FCT address of MAIN is 164008. The information passed to GETPUTMN (enter 136C30) is register 0 and 1, the address of the save area to be used by the ASAVE routine and the return address (13B998) to MAIN.

The next entry is a passing of control to MAIN, FCT number two at 17D048. COND= 5 indicates an AWAIT condition, COND= 8 indicates an

AWAITOFF condition. The address (13B5E8) after AWAIT is the entry point at the AWAIT routine.

FUNCTION CONTROL TABLE

Now look at the formatted FCT entry for ISDRVR, location 17D410. This FCT is posted but not in control because the prior FCT, DSP name DC, is active. ISDRVR has made a call to routine in IONUC, as shown by the entry point in the Active Save Area Chain. Register 1 points to an FDB and the saved base is register 10, the module base of the caller. The remaining registers, 2 through 9 are the saved registers at the time of the call. When the FCT is placed in an AWAIT, the registers are saved in the FCT. Registers 10, in the FCT, is the module base at the time of the AWAIT and register 14 is the return address to ASAVE.

RJP TABLES

On the last page of the dump is a Resident RJP line and Terminal Table. This table is built from the RJPLINE and RJPTERM initialization card. The SUPUNITS entry for TYPE=LINE will be filled in at the time the *S RJP,L= is issued. The FDB address is for the formatted control blocks.

The TYPE=TERM SUPUNITS is filled in when the terminal enters the SIGNON card. The preformatted control blocks are pointed to by its FDB.

```
* ASP ABEND DUMP * , HHMMSS=062417, DATE=72268


SYSTEM ABEND CODE IS 0C1

PROGRAM ABENDED IN MODULE DC      , LOCATION 14DE94 (REL LOC 000F3C), MODULE BASE IS 14CF58

INTERRUPTING INSTRUCTION IS *CALL

REGISTERS AT TIME OF INTERRUPT
REGS 0-7    0014E0E8   00000048   0014DE94   0014E1C3   00000000   00000000   0117F6A8   0016832C
REGS 8-15   001683FC   9C14D076   0014CF58   0017D170   0013A128   00140F58   4014D5F0   0014DD4C

MAP OF ASP NUCLEUS
1357C0   CONSINPT
136380   CONSQMGR
136C30   GETPUTMN
136EA8   JSSDR
137490   FUNCTLTB
137BEE   DSPDCTNY
138340   DEVREQ
138490   IODATA
1391A8   IGNUC
13A12E   IVTABLE
13A690   GETPUTUN
13AFC0   ALDADEL
138270   ASAVERTN
1385E8   ASPCONTL
13C230   TRACKS
13C808   CONSOLES
13CF6E   NETCONTL
13D53E   MTDDRVR
13D690   JOBCONTL
13E6B8   CONSCONS
13E8A8   ASPABEND
13F508   ASPCKPT
13F638   CKPTDATA
13F978   ASPOPEN
13FCF0   CALLDRVR
13FF1E   INITIATE
140470   INTCOM
1406BC   JOBNUM
140918   LOGINOUT
140A00   ABENDMON
141170   AMIG
141238   IORINS
141598   CONSAUTH
1415B0   ADEQUEUE

ACTIVE FCT ENTRY IS DC       AT 17D170



                              CONSOLE STATUS

 LOC     CON-NAME  CUA  CNUM  TYPE  FLAGS  FLAG2  DVFLG  SWTO  DEPTH  DEPQD
178E30   CN1       01F  0001  1052   22     00     18           99      7

         UNTYPED MESSAGES THIS CONSOLE
          LOC      PRTY TIME     TEXT
          176958    05  062405   PUR01 JOB 0002,DR        PURGED
          178988    05  062405   ISV01 JOB 0004 IS JA1     , PRTY=00      NET-ID=N1
          178838    05  062407   DSP01 JOB 0005 IS DJCUPDAT, CALLED BY INTERCOM
          1784D8    05  062409   ISV01 JOB 0006 IS JA2     , PRTY=00      NET-ID=N1
          178688    05  062412   ISV01 JOB 0007 IS JA3     , PRTY=00      NET-ID=N1
          178838    05  062415   AMSV01 JOB 0004,JA1      IS ON SY1      NET-ID=N1
          1785F8    05  062415   ISV01 JOB 0008 IS JA4     , PRTY=00      NET-ID=N1

 LOC     CON-NAME  CUA  CNUM  TYPE  FLAGS  FLAG2  DVFLG  SWTO  DEPTH  DEPQD
178E8C   CN2260    041  0002  2260   22     00     10           50      3

         UNTYPED MESSAGES THIS CONSOLE
          LOC      PRTY TIME     TEXT
          1787D8    05  062412   ISV01 JOB 0007 IS JA3     , PRTY=00      NET-ID=N1
          178598    05  062415   AMSV01 JOB 0004,JA1      IS ON SY1      NET-ID=N1
          178AD8    05  062415   ISV01 JOB 0008 IS JA4     , PRTY=00      NET-ID=N1

 LOC     CON-NAME  CUA  CNUM  TYPE  FLAGS  FLAG2  DVFLG  SWTO  DEPTH  DEPQD
178EE8   CNPR1     00E  0003  1403   00     00     00           20      0

 LOC     CON-NAME  CUA  CNUM  TYPE  FLAGS  FLAG2  DVFLG  SWTO  DEPTH  DEPQD
178F44   CNPR2     00F  0004  1403   22     00     10           20      1

         UNTYPED MESSAGES THIS CONSOLE
          LOC      PRTY TIME     TEXT
          1788F8    05  062417   MLG            SY1 S=F MT,ATTR001' JA1
          178658    09  062417       CN2260    *S DC,FORMAT

 LOC     CON-NAME  CUA  CNUM  TYPE  FLAGS  FLAG2  DVFLG  SWTO  DEPTH  DEPQD
178FA0   DUMMY          7FFF  DUMY   00     00     00           32      0
```

MAIN PROCESSOR CONTROL TABLES

LOC	NAME	CSECT	TYPE	ACT	SW	DEV	BAR	DEPTH	DEEP	DSTL	SNS	JOB	RDRFDB	RCSECT	WTRFDB	WCSECT	REL
142040	SY1	1709A8	LOCAL	MVT	00	270	1			70	00	17EC68	000000	000000	000000	000000	207

SY1 R= ATTR
SY1 S=F MT,ATTR001' JA1 'ASPBATCH

LOC	NAME	NUMBER	JDS	RESQ	MSG	ACT	INIT
1709A8	JA1	0004	CLOSED	17EC68	B		

LOC	NAME	CSECT	TYPE	ACT	SW	DEV	BAR	DEPTH	DEEP	DSTL	SNS	JOB	RDRFDB	RCSECT	WTRFDB	WCSECT	REL
146370	SY3	IDLE	REAL	MVT	00	370	1			00		000000	000000	000000	000000	000000	

SY3 R=
SY3 S=

RESQUEUE TABLE

LOC	JOB-NO	JOB-NAME	SETUP	REGION	CLASS	PRIORITY	FUNCTION	MAIN	FLG2	ACTIVE	HOLD
17EC68	4	JA1	NO	00052	A	0	ON MAIN	FF	04	YES	NO

PRINTER RESOURCES TABLE

LOC	NAME	FORMS	CARRIAGE	TRAIN	UCS	C-FORMS	C-TRAIN	RJP
179A20	PR1	1PART	6/INCH	PN	YES	YES	NO	NO
179A50	PR2	1PART	6/INCH	PN	YES	YES	NO	NO
179A80	PR3	1PART	6/INCH	PN	YES	YES	NO	NO
179AB0	PR4	1PART	6/INCH	PN	YES	YES	NO	NO
179AE0	RM002PR1	1PART	6/INCH	PN	NO	YES	NO	YES

SUPUNITS TABLE

LOC	TYPE	DDNAME	GROUP	UNIT	FLAG1	FLAG2	SYSUNIT	DCTADD	UCB
17B850	ACMACMP	ACSY1	LOCAL		00	00	17EAD0		0000
17B88C	ACMACMP	ACSY2	LOCAL		00	00	17EAE8		0000
17B8C8	ARIOSRDR	MYRDR1	LOCAL		00	00	17E8C0		6000
17BC04	IJMIJMP	IJSY1	LOCAL		00	00	17EAA0		0000
17BC40	IJMIJMP	IJSY2	LOCAL		00	00	17EAB8		0000
17BC7C	PRT1403	PR1	LOCAL	00E	00	00	17EB90		1848
17BCB8	PRT1403	PR2	LOCAL	00F	80	00	17EBA8		1868
17BCF4	PRT1403	PR3	LOCAL	038	80	00	17EBC0		19C0
17BD30	PRT1403	PR4	LOCAL	039	80	00	17EBD8		1A10
17BD6C	PUN2540	PU1	LOCAL	00D	00	00	17EC20		1830
17BDA8	PUN2540	PU2	MACH2	03A	80	00	17EC38		19F8
17BDE4	RDR2540	RD1	LOCAL	00C	00	00	17EBF0		1818
17BE20	RDR2540	RD2	MACH2	039	80	00	17EC08		19E0
17BE5C	SYSMAIN	SY1	LOCAL	270	00	00	17E8D8		21A0
17BE98	SYSMAIN	SY3	LOCAL		80	00	17E8F0		0000
17BED4	TA924009	T91	LOCAL	180	A0	00	17E908		1E10
17BF10	TA924009	T92	LOCAL	181	A0	00	17E920		1E44
17BF4C	TA924009	T94	MACH2	1A1	A0	00	17E938		1F2C
17BF88	TA924009	T93	MACH2	1A0	A0	00	17E950		1EF8

SETNAMES TABLE

LOC	TYPE	NAME	ALT-TYPE	CLASS
17B71A	01	2314	NO	DA
17B724	01	SYSDA	NO	DA
17B72E	01	SYSDX	NO	DA
17B742	02	TAPE9	NO	TA
17B74C	02	2400	NO	TA
17B756	02	2400-3	NO	TA
17B760	02	SYSSQ	NO	TA
17B774	03	DCH4	YES	DA
17B788	04	DCH4	YES	DA
17B792	04	DCH5	YES	DA
17B7A6	05	DCH4	NO	DA
17B7B0	05	DCH5	YES	DA
17B7BA	05	DCH6	YES	DA
17B7CE	03	DCH5	NO	DA
17B7D8	03	DCH6	YES	DA
17B7EC	04	DCH6	NO	DA

SETUNITS TABLE FOR SY1

LOC	TYPE	ADDRESS	OFFLINE	MOUNT-ID	RESQUEUE	SYSUNIT
17B538	C1	130	NO			17E980
17B54C	01	131	NO			17E968
17B560	01	132	NO			17E998
17B574	C1	133	NO			17E9B0
17B588	01	134	NO			17E9C8
17B59C	01	230	NO			17E9E0
17B5B0	C1	231	NO			17E9F8
17B5C4	01	232	NO			17EA10
17B5D8	C1	233	NO			17EA28
17B5EC	01	234	NO			17EA40
17B600	01	235	NO			17EA58
17B614	C1	236	NO			17EA70
17B628	01	237	NO			17EA88
17B63C	C2	180	NO			17E908
17B65C	C2	181	NO			17E920
17B664	02	1A0	YES			17E950
17B678	C2	1A1	YES			17E938

SETUNITS TABLE FOR SY3

LOC	TYPE	ADDRESS	OFFLINE	MOUNT-ID	RESQUEUE	SYSUNIT
17B690	01	335	YES			17EB48
17B6A4	02	380	YES			17EB78
17B6B8	02	382	YES			17EB60
17B6CC	C3	430	YES			17EB30
17B6E0	04	530	YES			17EB18
17B6F4	C5	630	YES			17EB00

SYSUNITS TABLE

LOC	MAINADD	SUPADD	VOL-ID	LABEL	MAIN	ALLOCATED	ASSIGNED	BARRIER	DEMAND	NO-ALT
17E8C0					00	NO	NO	NO	NO	NO
17E8D8		270			00	NO	NO	NO	NO	NO
17E8F0					00	NO	NO	NO	NO	NO
17E908	180	180			01	NO	NO	NO	NO	NO
17E920	181	181	LGL001	N	01	NO	NO	NO	NO	NO
17E938	1A1	1A1			01	NO	NO	NO	NO	NO
17E950	1A0	1A0			01	NO	NO	NO	NO	NO
17E968	131		JOBLIB		01	NO	NO	NO	NO	NO
17E980	130		ASPUT1		01	NO	NO	NO	NO	NO
17E998	132		ASP305		01	NO	NO	NO	NO	NO
17E9B0	133		ASP206		01	NO	NO	NO	NO	NO
17E9C8	134		ASPCUE		01	NO	NO	NO	NO	NO
17E9E0	230		ASPUT2		01	NO	NO	NO	NO	NO
17E9F8	231		ASPUT3		01	NO	NO	NO	NO	NO
17EA10	232		ASPQUE		01	NO	NO	NO	NO	NO
17EA28	233		APT360		01	NO	NO	NO	NO	NO
17EA40	234		APTWRK		01	NO	NO	NO	NO	NO
17EA58	235		TSO207		01	NO	NO	NO	NO	NO
17EA70	236		DNCSYS		01	NO	NO	NO	NO	NO
17EA88	237		STORG2		01	NO	NO	NO	NO	NO
17EAA0					00	NO	NO	NO	NO	NO
17EAB8					00	NO	NO	NO	NO	NO
17EAD0					00	NO	NO	NO	NO	NO
17EAE8					00	NO	NO	NO	NO	NO
17EB00	630				02	NO	NO	NO	NO	NO
17EB18	530				02	NO	NO	NO	NO	NO
17EB30	430				02	NO	NO	NO	NO	NO
17EB48	335				02	NO	NO	NO	NO	NO
17EB60	382		TAP382		02	NO	NO	NO	NO	NO
17EB78	380		TAP380		02	NO	NO	NO	NO	NO
17EB90		00E			00	NO	NO	NO	NO	NO
17EBA8		00F			00	NO	NO	NO	NO	NO
17EBC0		038			00	NO	NO	NO	NO	NO
17EBD8		03B			00	NO	NO	NO	NO	NO
17EBF0		00C			00	NO	NO	NO	NO	NO
17EC08		039			00	NO	NO	NO	NO	NO
17EC20		00D			00	NO	NO	NO	NO	NO
17EC38		03A			00	NO	NO	NO	NO	NO

ASP I/O TRACE TABLE

ROUTINE	CALLER		ROUTINE	CALLER		ROUTINE	CALLER		ROUTINE	CALLER	
PUTBUF	13955C	173620	PUTIOB	1399AE	164B58	PUTBUF	1399EE	16D5C4	DISK	139A12	
RD/WRITE	150EF8		OPEN	139398		DISK	1392DE		GETBUF	139884	167568
GETIOB	139D24	164B58	RD/WRITE	145074		OPEN	139398		DISK	1392DE	
GETIOB	139D24	164AE8	PUTIOB	1399AE	164B58	DISK	139A12		RELEASE	141E2E	
PUTBUF	13997C	16832C	OPEN	14F158		LOCATE	14F178		GETBUF	138526	1743E4
BLOCK	14F192		CLOSE	14F1A2		LOCATE	1395D2		OUTPUT	1395F0	
DISK	139400		GETIOB	139D24	164B58	PUTIOB	1399AE	164AE8	PUTBUF	1399EE	16AC78
DISK	139A12		RD/WRITE	13D2CE		OPEN	139398		DISK	1392DE	
GETBUF	139884	16E388	GETIOB	139D24	164AE8	PUTIOB	1399AE	164B58	PUTBUF	1399EE	1743E4
DISK	139A12		PUTBUF	14F382	169EB4	RD/WRITE	14F284		OPEN	139398	
DISK	1392DE		GETIOB	139D24	164B58	PUTIOB	1399AE	164AE8	DISK	139A12	
RD/WRITE	13D464		OPEN	139398		DISK	1392DE		GETIOB	139D24	164AE8
PUTIOB	1399AE	164B58	PUTBUF	1399EE	167568	DISK	139A12		LOCATE	14F416	
BLOCK	14F42C		LOCATE	14F416		BLOCK	14F42C		CLOSE	14F2E6	
LOCATE	1395D2		OUTPUT	1395F0		DISK	139400		GETIOB	139D24	164B58
PUTIOB	1399AE	164AE8	PUTBUF	1399EE	16E388	DISK	139A12		RD/WRITE	145274	
OPEN	139398		RD/WRITE	13931A		OPEN	139398		DISK	1392DE	
GETBUF	139884	164C1C	GETIOB	139D24	164AE8	PUTIOB	1399AE	164B58	PUTBUF	1399EE	1690F0
DISK	139A12		RD/WRITE	14F2F6		OPEN	139398		DISK	1392DE	
GETIOB	139D24	164B58	PUTIOB	1399AE	164AE8	DISK	139A12		DISK	1392DE	
GETIOB	139D24	164AE8	PUTIOB	1399AE	164B58	PUTBUF	1399EE	17285C	DISK	139A12	
RD/WRITE	14094C		OPEN	139398		DISK	1392DE		GETBUF	139B84	16FF10
GETIOB	139D24	164B58	PUTIOB	1399AE	164AE8	PUTBUF	1399EE	168A3C	DISK	139A12	
RD/WRITE	145284		OPEN	139398		DISK	1392DE		GETIOB	139D24	164AE8
DISK	1392DE		GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	PUTBUF	1399EE	16AC78
DISK	139A12		RD/WRITE	13D2CE		OPEN	139398		DISK	1392DE	
GETBUF	139884	1659E0	GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	DISK	139A12	
RD/WRITE	13D464		OPEN	139398		DISK	1392DE		GETIOB	139D24	164A78
PUTIOB	1399AE	164A78	PUTBUF	1399EE	1659E0	DISK	139A12		RD/WRITE	145274	
OPEN	139398		RD/WRITE	13931A		OPEN	139398		DISK	1392DE	
GETBUF	139884	171A98	GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	DISK	139A12	
DISK	1392DE		GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	PUTBUF	1399EE	16D5C4
DISK	139A12		RD/WRITE	145284		OPEN	139398		DISK	1392DE	
GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	PUTBUF	1399EE	171A98	DISK	139A12	
RD/WRITE	130730		OPEN	139398		DISK	1392DE		GETBUF	139884	1667A4
GETIOB	139D24	164A78	PUTIOB	1399AE	164A78	DISK	139A12		RD/WRITE	13D798	
OPEN	139398		DISK	1392DE		GETIOB	139D24	164A78	PUTIOB	1399AE	164A78
PUTBUF	1399EE	1667A4	DISK	139A12		RD/WRITE	135156		OPEN	139398	
DISK	1392DE		GETBUF	139884	16F14C	GETIOB	139D24	164A78	PUTIOB	1399AE	164A78
DISK	139A12										

ASP BUFFER POOL

LOC	FDB	STATUS	LOC	FDB	STATUS	LOC	FDB	STATUS	LOC	FDB	STATUS
164C1C	16BAB4	IN	1659E0	17E80C	IN	1667A4	13F648	IN	167568	000000	IN
16832C	000000	IN	1690F0	17EC74	IN	169EB4	14A318	IN	16AC78	000000	IN
16BA3C	17D9E0	OUT	16C800	13E340	IN	16D5C4	000000	IN	16E388	17E80C	IN
16F14C	14A340	OUT	16FF10	000000	OUT	170CD4	14A340	IN	171A98	14A2F0	IN
17285C	17E830	IN	173620	14A3EC	OUT	1743E4	14A32C	IN	1751A8	000000	OUT

ASP REGION USAGE

SP	START	END	LENGTH	CONTENTS
251	135000	1417FF	00C800	51200 SPACE ASSIGNED TO DQE
	135000	135047	000048	72 FREE SPACE
	135048	1356AF	000668	1640 MODULE ISDRVR
	1356B0	1356FF	000050	80 MODULE TRCERTN
	1357C0	1417FF	00C100	49408 MODULE ASPNUC
251	141800	141FFF	000800	2048 SPACE ASSIGNED TO DQE
	141800	14182F	000030	48 FREE SPACE
	141830	141CAF	000480	1152 MODULE PRTDATA
	141CB0	141CEF	000040	64 FREE SPACE
	141CF0	141FFF	000310	784 MODULE RICBAM
251	142000	1427FF	000800	2048 SPACE ASSIGNED TO DQE
	142000	14203F	000040	64 FREE SPACE
	142040	142407	0003C8	968 MODULE MPCDATA*
	1424C8	14257F	000178	376 FREE SPACE
	142580	1427FF	000280	640 MODULE CONS2260
	142800	142FFF	000800	2048 FREE BLOCK QUEUE ELEMENT
251	143000	1437FF	001800	6144 SPACE ASSIGNED TO DQE
	143000	143007	000008	8 FREE SPACE
	143008	14322F	000228	552 MODULE CONS1403
	143230	1437FF	001500	5584 MODULE JSS
	144800	1457FF	001000	4096 FREE BLOCK QUEUE ELEMENT
251	161C00	1617FF	000800	2048 SPACE ASSIGNED TO DQE
	161000	16112F	000130	304 FREE SPACE
	161130	1612A7	000178	376 MODULE CONS1052
	1612A8	161727	000480	1152 MODULE PRTDATA
	161728	1617FF	0000D8	216 FREE SPACE
	161800	163FFF	002800	10240 FREE BLOCK QUEUE ELEMENT
0	164000	1777FF	013800	79872 SPACE ASSIGNED TO DQE
	164000	164007	000008	8 FREE SPACE
	177800	177FFF	000800	2048 FREE BLOCK QUEUE ELEMENT
0	178000	178FFF	001000	4096 SPACE ASSIGNED TO DQE
	178000	178077	000078	120 FREE SPACE
	1784E0	178497	000018	24 FREE SPACE
0	179800	17A7FF	001000	4096 SPACE ASSIGNED TO DQE
C	178C00	1787FF	000800	2048 SPACE ASSIGNED TO DQE
	178C00	178017	000018	24 FREE SPACE
0	178800	178FFF	000800	2048 SPACE ASSIGNED TO DQE
	17BEC0	17B81F	000020	32 FREE SPACE
251	17C000	17C7FF	000800	2048 SPACE ASSIGNED TO DQE
	17C000	17C16F	000170	368 FREE SPACE
	17C170	17C33F	0001D0	464 MODULE MDSDATA
	17C340	17C7FF	0004C0	1216 FREE SPACE
	17C800	17CFFF	000800	2048 FREE BLOCK QUEUE ELEMENT
0	17D000	17D7FF	000800	2048 SPACE ASSIGNED TO DQE
	17D000	17D007	000008	8 FREE SPACE

ASP TRACE TABLE

TYPE	FCTPTR	R0	R1	R2	R3	R4	DSPNAME
CALL	164008	PARM0 00000016	PARM1 FFFFFFFF	RETURN 5013B998	ENTER 00136C30	SAVEAREA 17F2A0	MAIN
DISPATCH	17D048	MASK=C3 COND=5	ECFAD 1423E9	RETURN 147A9E	AWAIT 5013B5E8	FCT NUM 2	MAIN
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=40 COND=5	ECFAD 14A333	RETURN 139600	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
DISPATCH	17D410	MASK=10 COND=5	ECFAD 14A331	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 1014A331	PARM1 0014A32C	RETURN 401457AC	ENTER 001355C0	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 0014A2F0	PARM1 0014A2D8	RETURN 40135620	ENTER 0013936E	SAVEAREA 17F2A0	ISDRVR
DISPATCH	000000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	C00000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 14A2DD	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 C0C00000	PARM1 0014A2F0	RETURN 40145074	ENTER 0013936E	SAVEAREA 17F260	ISDRVR
DISPATCH	000000	MASK=00 COND=0	ECFAD C00000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 14A2F5	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 00000000	PARM1 801451D8	RETURN 401451E8	ENTER 00136380	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 05C00000	PARM1 0014A520	RETURN 701451FE	ENTER 0013D060	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 00000014	PARM1 00000012	RETURN 5013D076	ENTER 0013E08E	SAVEAREA 17F2A0	ISDRVR
CALL	17D410	PARM0 1017E800	PARM1 0014A4A8	RETURN 60145216	ENTER 0013D29C	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 1017E800	PARM1 0017E80C	RETURN 6013D2CE	ENTER 00139338	SAVEAREA 17F2A0	ISDRVR
DISPATCH	137490	MASK=00 COND=5	ECFAD 13E6C7	RETURN 13CDA4	AWAIT 5113B5E8	FCT NUM 1	CONSOLES
DISPATCH	C00000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	000000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	000000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	137490	MASK=00 COND=5	ECFAD 13E6C7	RETURN 13CDA4	AWAIT 5113B5E8	FCT NUM 1	CONSOLES
CALL	137490	PARM0 0013E6B8	PARM1 00000000	RETURN 4013CDD2	ENTER 00141580	SAVEAREA 137558	CONSOLES
DISPATCH	17D048	MASK=C3 COND=5	ECFAD 1423E9	RETURN 147A9E	AWAIT 5013B5E8	FCT NUM 2	MAIN
CALL	17D048	PARM0 C0C00000	PARM1 0014210C	RETURN 40148314	ENTER 00136380	SAVEAREA 17D130	MAIN
DISPATCH	137490	MASK=00 COND=5	ECFAD 13E6C7	RETURN 13CDA4	AWAIT 5113B5E8	FCT NUM 1	CONSOLES
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 17E811	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 1017E811	PARM1 1017E800	RETURN 40145242	ENTER 0013D434	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 0013A4CC	PARM1 0017E80C	RETURN 6013D464	ENTER 0013936E	SAVEAREA 17F2A0	ISDRVR
DISPATCH	137490	MASK=00 COND=5	ECFAD 13E6C7	RETURN 13CDA4	AWAIT 5113B5E8	FCT NUM 1	CONSOLES
CALL	137490	PARM0 0013E688	PARM1 00000000	RETURN 4013CDD2	ENTER 00141580	SAVEAREA 137558	CONSOLES
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 17E811	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 1017E811	PARM1 0014A520	RETURN 50145250	ENTER 0013D104	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 0014A2FC	PARM1 0014A2E8	RETURN 40145274	ENTER 0013936E	SAVEAREA 17F260	ISDRVR
CALL	17D410	PARM0 0014A2F6	PARM1 0014A2F0	RETURN 5013931A	ENTER 00139338	SAVEAREA 17F2A0	ISDRVR
DISPATCH	C00000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	000000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	C00000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	137596	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 14A2F5	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
DISPATCH	C00000	MASK=00 COND=0	ECFAD 000000	RETURN 000000	AWAIT 00000000	FCT NUM 13	
DISPATCH	137598	MASK=80 COND=5	ECFAD 139F35	RETURN 139986	AWAIT 5013B5E8	FCT NUM 6	ASPIO
DISPATCH	17D410	MASK=10 COND=5	ECFAD 14A2E0	RETURN 13B3BE	AWAIT 5013B5E8	FCT NUM 9	ISDRVR
CALL	17D410	PARM0 C0C00000	PARM1 0014A2F0	RETURN 40145284	ENTER 0013936E	SAVEAREA 17F260	ISDRVR

```
137490  PROGRAM NAME IS CONSOLES

        ECF OF X'C1' AT 13E6C7 IS POSTED

        ECF OF X'00' AT 000000 IS NOT POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00137558       00000000        00000000        FF1378E8       00000000       00000000       00000000

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST        RJECF-RJPTR
            00000000                       00000000        00000000       00000000       00137550       00000000

            FCTFLAGS        FAILSOFT                      ****USER*****
            1 2 3 U         F S STAE        TRFDB         HALF   FULL                WORK                    FSCOD
            00004000        00000000    000000000000      0000 00000000      47F0E000F4F1F750

              RSCNT          FSLOC          FSRTN           RSVD           RSVD           RSVD           RSVD
              0000           00000000       00000000       47F0A488       47F0A314       9110601C       4780AE24

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 13C0A4
        REG 0-7    0013E688   00178DC0   14178BF8   0013E6C7   00178734   001784C0   23178E20   00000000
        REG 8-15   00C00000   00178FFC   0013C8D8   00137490   0013A128   0013E688   4013CDA4   5113B5E8


17D048  DSP NAME IS    MAIN        JOB NUMBER IS     0        JOB PRIORITY IS  0        DSP PRIORITY IS  21
        LOAD MODULE IS MAINIO       MODULE BASE IS 147A80      SE SEQUENCE IS   0        JOB FDB IS FF0600808000

        ECF OF X'C3' AT 1423E8 IS POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00170130       00000000        00000000        15137D78       00000000       EEEEEEEE       0014211A

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST        RJECF-RJPTR
            00000000                       00000000        00000000       00000000       0017D108       00000000

            FCTFLAGS        FAILSOFT                      ****USER*****
            1 2 3 U         F S STAE        TRFDB         HALF   FULL                WORK                    FSCOD
            00000000        00000000    000000000000      0000 00000000      47F0E004F4F1F720

              RSCNT          FSLOC          FSRTN           RSVD           RSVD           RSVD           RSVD
              0000           00000000       00000000       00000000       00000000       00000000       00000000

        GETUNIT LIST - DDNAME      TYPE     SUPUNIT
                       SY1                 178E5C

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 147A9E
        REG 0-7    C31423E9   0013E6C7   A0146826   50148226   8013B6E0   00148FA8   0000001F   90148224
        REG 8-15   00000000   00142040   00147A80   00170048   0013A128   00000000   40147A9E   5013B5E8


178920  DSP NAME IS    MAIN        JOB NUMBER IS     0        JOB PRIORITY IS  0        DSP PRIORITY IS  21
        LOAD MODULE IS MSVDUMMY     MODULE BASE IS 149308      SE SEQUENCE IS   0        JOB FDB IS FF0600808000

        ECF OF X'FB' AT 146719 IS NOT POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00178A08       00000000        00000000        15137D78       00000000       EEEEEEEE       0014644A

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST        RJECF-RJPTR
            00000000                       00000000        00000000       00000000       00178E60       00000000

            FCTFLAGS        FAILSOFT                      ****USER*****
            1 2 3 U         F S STAE        TRFDB         HALF   FULL                WORK                    FSCOD
            00000000        00000000    000000000000      0000 00000000      0000000000000000

              RSCNT          FSLOC          FSRTN           RSVD           RSVD           RSVD           RSVD
              0000           00000000       00000000       00000000       00000000       00000000       00000000

        GETUNIT LIST - DDNAME      TYPE     SUPUNIT
                       SY3                 178E98

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 1493D8
        REG 0-7    FB146719   00000000   00000000   00000000   00000000   00000000   00000000   00000000
        REG 8-15   00000000   00146370   00149308   00178920   0013A128   00000000   001493D8   50000000


164008  DSP NAME IS    MAIN        JOB NUMBER IS     0        JOB PRIORITY IS  0        DSP PRIORITY IS  20
        LOAD MODULE IS MSVINIT      MODULE BASE IS 14F918      SE SEQUENCE IS   0        JOB FDB IS FF0600808000

        ECF OF X'FF' AT 142238 IS NOT POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00C00000       00164DD0        00000000        14137D78       00000000       EEEEEEEE       00000000

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST        RJECF-RJPTR
            00C00000                       00000000        00000000       00000000       00164DC8       00000000

            FCTFLAGS        FAILSOFT                      ****USER*****
            1 2 3 U         F S STAE        TRFDB         HALF   FULL                WORK                    FSCOD
            00C00000        00000000    000000000000      0000 00000000      47F0E0000000F0F1

              RSCNT          FSLOC          FSRTN           RSVD           RSVD           RSVD           RSVD
              0000           00000000       00000000       00000000       00000000       00000000       00000000

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 14F1CC
        REG 0-7    FF142238   0017F658   00142040   00178498   00000015   0000000E   00142250   00178378
        REG 8-15   0017F658   00142040   0014EF98   00164008   0013A128   001783A0   5014F1CC   5013B5E8

        ACTIVE SAVE AREA CHAIN

         LOC       FLG/CHAIN    WORK      RETURN     ENTRY PNT   REG 0      REG 1      SAVED BASE
        164DD0     F0000000   00000000   4014FCA6   0014EF98   0014EF98   00000016   0014F918
        REG 2-9    00000001   4014FA5C   00000002   00168A40   00179926   0017EC68   001690F0   00142040
```

```
137598  PROGRAM NAME IS ASPIO

        ECF OF X'80' AT 139F35 IS NOT POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00137660       00000000        00000000        0D137C10       00000000       00000000       00000000

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST         RJECF-RJPTR
            0000000C                       00000000        D0000000       00000000       00137658       00000000

             FCTFLAGS       FAILSOFT                       ****USER*****
             1 2 3 U        F S STAE          TRFDB        HALF   FULL              WORK                  FSCOD
            00004000       00000000      000000000000     0000 00000000      0000000000000000

              RSCNT          FSLOC          FSRTN           RSVD            RSVD           RSVD           RSVD
               0000          00000000       00000000        D207301C       70184530       A3B69101       90044780

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 139986
        REG 0-7    80139F35   FFFFFFF4   0013A118   80139AFA   051667A4   00000000   00000002   00164A78
        REG 8-15   00164A50   00164110   001391A8   00137598   0013A128   00000000   00139986   501385E8


17D2C0  DSP NAME IS    DJCUPDAT    JOB NUMBER IS     5        JOB PRIORITY IS 15        DSP PRIORITY IS   7
        LOAD MODULE IS DJCUPDAT    MODULE BASE IS 14FE10      SE SEQUENCE IS   1        JDB FDB IS 01026307E4D0

        ECF OF X'40' AT 13A378 IS NOT POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            0017D3CC       00000000        01178440        07138228       0001DE88       06240779       001507C8

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST         RJECF-RJPTR
            00000000                       00000000        D0000000       00000000       0F17D380       00000000

             FCTFLAGS       FAILSOFT                       ****USER*****
             1 2 3 U        F S STAE          TRFDB        HALF   FULL              WORK                  FSCOD
            20000400       00000000      000000000000     0000 00000000      47F0E00000000000

              RSCNT          FSLOC          FSRTN           RSVD            RSVD           RSVD           RSVD
               0000          00000000       00000000        00000000       00000000       00000000       00000000

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 14FE36
        REG 0-7    4013A378   0072268F   00000000   00000000   00000000   00000000   00000000   07138228
        REG 8-15   00000000   00000000   0014FE10   0017D2C0   0013A128   0017DD80   5014FE36   501385E8


7D170   DSP NAME IS    DC          JOB NUMBER IS     1        JOB PRIORITY IS 15        DSP PRIORITY IS   3
        LOAD MODULE IS DC          MODULE BASE IS 14CF58      SE SEQUENCE IS   1        JDB FDB IS 01016305E4F0

        ECF OF X'FF' AT 14E0FC IS POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00000000       0017D27C        0117F6A8        03138160       00000000       06224028       0014E0AE

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST         RJECF-RJPTR
            00000000                       00000000        D0000000       00000000       0F17D230       00000000

             FCTFLAGS       FAILSOFT                       ****USER*****
             1 2 3 U        F S STAE          TRFDB        HALF   FULL              WORK                  FSCOD
            20000400       00000000      000000000000     0000 00000000      47F0E000F3F3F841

              RSCNT          FSLOC          FSRTN           RSVD            RSVD           RSVD           RSVD
               0000          00000000       00000000        00000000       00000000       00000000       00000000

        GETUNIT LIST - DDNAME     TYPE    SUPUNIT
                                          000000

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 14D5F0
        REG 0-7    FF14E0FC   0013E6C7   0014DE94   0014E1C3   00000000   00000000   0117F6A8   0016832C
        REG 8-15   001683FC   9014D076   0014CF58   0017D170   0013A128   0014DF58   4014D5F0   501385E8

        ACTIVE SAVE AREA CHAIN

         LOC        FLG/CHAIN     WORK       RETURN     ENTRY PNT    REG 0      REG 1     SAVED BASE
        17D27C      F0000000    050C0000   4014D5F0   0014DD4C   0014E0E8   00000048   0014CF58
        REG 2-9     0014DE94    0014E1C3   00000000   00000000   0117F6A8   0016832C   001683FC   9014D076


17D410  DSP NAME IS    ISDRVR      JOB NUMBER IS     3        JOB PRIORITY IS 15        DSP PRIORITY IS   3
        LOAD MODULE IS ISDRVR      MODULE BASE IS 135048      SE SEQUENCE IS   1        JDB FDB IS 01026404E4B0

        ECF OF X'10' AT 14A345 IS POSTED

             RESAV          SAVCH          SESEQ-AJDB      PRTY-DSPDC      CSECT          TIMON          LOGIN
            00000000       0017D51C        0117D008        03138188       0001E0A0       06240290       0014A298

             TNEXT           TUID           TIMEI          TFLAG-TIMEX     CBPTR          GLIST         RJECF-RJPTR
            00000000                       00000000        D0000000       00000000       0F17D4D0       00000000

             FCTFLAGS       FAILSOFT                       ****USER*****
             1 2 3 U        F S STAE          TRFDB        HALF   FULL              WORK                  FSCOD
            20000400       00000000      000000000000     0000 00000000      47F0E000F4F1F519

              RSCNT          FSLOC          FSRTN           RSVD            RSVD           RSVD           RSVD
               0000          00000000       00000000        00000000       00000000       00000000       00000000

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 13B3BE
        REG 0-7    1014A345   0014A340   00164AA8   00164A18   001641E8   FFFFFFF8   5013B3BE   90139398
        REG 8-15   00000000   00164110   001391A8   0017D410   0013A128   0014A298   5013B3BE   501385E8

        ACTIVE SAVE AREA CHAIN

         LOC        FLG/CHAIN     WORK       RETURN     ENTRY PNT    REG 0      REG 1     SAVED BASE
        17D51C      F0000000    00CC0000   60135156   00139338   00062416   0014A340   00135048
        REG 2-9     00000000    0000002C   00000000   00000000   0014A340   0014A32C   00000000   00000000
```

```
1379D8  PROGRAM NAME IS JSS

        ECF OF X'FF' AT 13A4DA IS POSTED

          RESAV          SAVCH          SESEQ-AJDB     PRTY-DSPDC     CSECT          TIMON          LOGIN
          00137AA0       00000000       00000000       00137CB0       00000000       00000000       001373FA

          TNEXT          TUID           TIMEI          TFLAG-TIMEX    CBPTR          .GLIST         RJECF-RJPTR
          00000000                      00000000       00000000       00000000       00137A98       00000000

          FCTFLAGS       FAILSOFT                      ****USER*****
          1 2 3 U        F S STAE       TRFDB          HALF   FULL           WORK                   FSCOD
          00004000       00000000       000000000000   0000 00000000  47F0E000F1F3F350

          RSCNT          FSLOC          FSRTN          RSVD           RSVD           RSVD           RSVD
          0000           00000000       00000000       D3A38725       4130D3A3       D2033008       D248D207

        FCT REGISTER SAVE AREA      AWAIT RETURN IS 1373F6
        REG 0-7    FF13A4DA   00000000   0017EC C4   0017EC 68   0014361A   0013E338   00000000   00137D78
        REG 8-15   0016C8D0   0016CA14   00136EA8   001379D8   0013A128   00144230   501373F6   5013B5E8
```

RESIDENT RJP LINE AND TERMINAL TABLE

LOC	DDNAME	TYPE	GROUP	PASSWORD	SUPUNITS	FLG1	FLG2	FLG3	FDB
178400	LINE01	LINE			000000	10	00	00	01026301E4D0
178428	LINE02	LINE			000000	10	00	00	01016302E4D0
178444	LINE03	LINE			000000	10	00	00	01026302E4D0
178460	RM002	TERM	RM002		000000	00	00	00	01016303E4D0
17847C	RM004	TERM	RM004		000000	00	00	00	01026303E4D0

ANALYSIS AT ASP INITIALIZATION

In addition to the above ABEND dumps, the system provides snapshots of any control blocks that contain errors causing one or more jobs to be deleted from the ASP job queue during ASP restart analysis. Each snapshot contains the registers at the time of the snap and the contents of the control block containing the error. The snap ID code identifies the control block printed:

SNAP ID	CONTROL BLOCK
01	JCT list of FDBs for priority 00-15
02	JCT record
03	JST record
04	JDAB record
05	R/I LTTR table
06	JDS record
07	Parameter buffer
09	Job TAT
0A	DJC NCB buffer

This information is to assist the system programmer in diagnosing the nature and cause of erroneous control block data.

STORAGE DUMP ON INITIALIZATION ERROR

A storage dump can be obtained upon occurrence of a specific error message issued to ASPOUT during ASP Initialization. This is activated by:

```
*INTDEBUG,n,message-test $$
```

*INTDEBUG (starting in column 1) activates the debugging monitor in INITRTNS. N (1 to 9) means to dump on the nth occurrence of the message being monitored. Message text (columns 13-71) is text to compare against the message up to the $$.

A new debugging interval starts (the previous one is reset) each time an *INTDEBUG card is read and interpreted. The INISH deck is read and interpreted, one card at a time, until ENDINISH is read. At that point, final processing is done, which could result in error messages issued by INITGEN, INITMDS, INITRJP2, INITRI, RIATTACH, INITCNS, and INITJOB.

Example 1

The third appearance of message ERR453 contains unmeaningful data. A dump is needed to analyze the cause. Since ERR453 is issued by INITMDS after ENDINISH is read, the *INTDEBUG,3,*ERR453W MAIN $$ can be placed in front of the ENDINISH CARD.

Example 2

It is not apparent why message ERR061 is being issued for a BUFFER card. A dump is needed to analyze the cause. Since ERR061 is issued by INITRTNS before ENDINISH is read, the message "*INTDEBUG,1,*ERR061C BAD KEYWORD, IOBS$$" is placed in front of the BUFFER card.

DUMP CORE DSP (DC)

A further debugging aid in ASP is the DSP Dump Core (DC). The Dump Core program displays selected areas of core, named FCTs, and load-list entry points on the operator console. It also allows the ASPABEND module to be called to provide a complete formatted printout of the ASP tables and to snap both SDATA and PDATA parameters on the //ASPSNAP DD unit.

Note: This module is provided primarily as a debugging tool for the exclusive use of the system programmer. It should be used only under controlled circumstances and with full knowledge of its impact on system performance. It nonetheless provides the system programmer with a powerful tool to aid in problem diagnosis, and in system maintenance and checkout.

The format of the operator input message to call the program is:

```
*CALL,DC[,KEY=nnn]
```

where nnn is an installation password protect key (see discussion below). When the program is scheduled in response to the above call, it responds with a message requesting that a start or cancel be issued.

Possible responses to this message are:

- *CANCEL DC
- *RESTART DC
- *START DC[,keyword-parameters]

The CANCEL response indicates that the services of the program are no longer required. DC closes the assigned units, resets all active traps, restores appropriate pointers, and returns to JSS to be deleted from the active functions of the system.

The RESTART response is significant only if the facilities available with password protection are being used (see discussion below).

The keyword parameters associated with the START response are:

```
*START DC[,FCT=name[,UNIT=device-name]]
    or   [,FIND={module-name|module-name,nn}]
    or   [,C=XXXXXX,B=nn|BASE=SSSSSS,C=YYYYYY,B=nn]
    or   [,FORMAT]
    or   [,SNAP,SD=sparms,PD=pparms,FORMAT]
```

[FCT=name][,UNIT=device-name]

The DSP Dictionary name of an active function can be entered in the name field. The contents of the FCT will be displayed on the console. If more than one function is active with this name, the first function encountered in the FCT chain will be displayed, unless the parameter UNIT= is also given, stating the ASP device name (PR1, RD2, etc.).

The display format for this option is:

THE FOLLOWING IS THE FCT=core-location, DSP=dictionary-name, followed by:

```
    Registers 0-4
    Registers 5-9
    Registers 10, 12-15
```

and the named fields of the FCT; for example, FCTLOGIN, FCTNEXT, FCTSAVCH, etc. When the FCT has been displayed, the start or cancel request is reissued, and a new request may be entered.

[FIND=module-name|module-name,nn]

This parameter requests the entry point address and base of the named module. If more than one such module exists on the load list, the nn parameter should be used, specifying which module in the sequence of load list occurrences should be displayed. CSECT=YES modules can be requested with or without the ASP-appended asterisk; for example:

FIND=RDDATA or FIND=RDDATA*

[C=xxxxxx,B=nn|BASE=ssssss,C=yyyyyy,B=nn]

This parameter requests nn (hexadecimal) bytes of storage, starting at location xxxxxx or at base ssssss, plus displacement yyyyyy. The contents of the specified storage location are displayed on the operator console. Note nn is rounded up to a multiple of 10 (hexadecimal) bytes.

[FORMAT]

This parameter requests that the module ASPABEND be given control in order to provide a formatted ASPABEND printout of the ASP tables, after which control is returned to DC for the next request. Note that this does not result in an ABEND of the ASP program; it does, however, dominate the entire system, placing all other ASP functions in abeyance during the execution of ASPABEND. Note also that ASPABEND uses the //ASPSNAP DD for ASPABEND output. This unit should be assigned to a printer or tape, or to an OS data set. If assigned to SYSOUT, the output will not be available to a WTR until the ASP execution ends (for example, *RETURN or ABEND) when the output is queued by OS.

[SNAP,SD=sparms,PD=pparms,FORMAT]

The positional parameter SNAP calls for the facilities of the OS SNAP macro. SD=sparms is the specification of SDATA as defined for the OS SNAP macro. PD=pparms refers to the PDATA specification of the OS SNAP macro. Note that, ASPABEND is called to take the SNAP and, as with ASPABEND, SNAP issues no AWAITs; thus, the system is tied up while the snap is being taken. Note also that the SNAP output will go to the //ASPSNAP DD unit. The parameter FORMAT is defined above.

In addition to the above facilities, certain other facilities are provided under password protection call. The password is entered when the DSP is called, as:

*CALL DC[,KEY=password]

Allowable passwords are assembled into an internal table, which the installation should tailor to its own needs as to both size and content. The table is referenced by the location symbol KEYTBL. Two other tables of interest are the active trap storage areas, named TABLE, and a dynamic patch area, named PATCH, which can also be tailored to the needs of the installation. These tables are located in DC DSP modules.

Additional keyword parameters associated with the START response when password protection is in effect are:

- [[,BASE=xxxxxx],C=yyyyyy,S=aaaa]

  where aaaa is from two to sixteen characters, which are the EBCDIC representation of from one to eight hexadecimal bytes to be stored in core location yyyyyy or in location xxxxxx, displaced yyyyyy.

- [,ACTIVE]

  displays the active trap storage area.

- [,PATCH]

  displays the dynamic patch area.

- [,SNAPON]

  causes the TVTABLE entry DYNDUMP (coded DC A(ARETURN)) to be changed to point to a location in DC (coded DC A(DYNDUMPX)), which implements the ASPSNAP macro and the dynamic traps.

- [,SNAPOFF]

  resets the TVTABLE' entry DYNDUMP to its original contents, thus turning off the SNAP facility of the ASPSNAP macro and dynamic traps. Note that a *S DC,SNAP,... is effective whether the SNAPON command has been issued or not.

- [,TRAP=xxxxxx,START=aaaaaa,END=bbbbbb]

  or

  [,BASE=yyyyyy,TRAP=xxxxxx,START=aaaaaa,END=bbbbbb]

  A trap will be established at location xxxxxx, or at location xxxxxx plus yyyyyy, which must be an instruction, must not be a branching type operation, and must be at least six bytes away from any other trap. The core area to be snapped may be defined by START and END as absolute addresses, or by BASE, START, and END, in which case START and END are displacements from BASE. The instruction at location xxxxxx will be saved, a zero OP code will be stored in its place, and a SPIE filter will be implemented so that the start-to-end area will be snapped when the location is executed. The original instruction will then be executed and return made to the next instruction location.

  Note: A routine which executes in protect key zero cannot be trapped. Any attempt to trap such routine will cause ASP to terminate.

To reset the active traps, issue this command:

```
*R DC
```

CBPRNT DSP

The CBPRNT DSP can be used to snap ASP and OS control blocks. The control blocks will be written to the SYSMSG data set, and printed at the job's termination. The user may select the control blocks to be printed or he may take the default list.

The default list is:

TAT	Track Allocation Table
JDAB	Job Description Accounting Block
JDS	Job Data Sets
JST	Job Setup Table
FRP	Format Parameter Buffer
OSCB	OS Control Blocks

An example is:

```
//SAMPLE   JOB 836,ASP,MSGLEVEL=1
//*PROCESS RICONTL
//*PROCESS CBPRNT
BLOCK=JST,JDAB
//*PROCESS MAIN
//*PROCESS CBPRNT
//*PROCESS PRINT
//SS1  EXEC  PGM=IEFBR14
```

The following is an example of CBPRNT usage. In the example a //*PROCESS card was placed before RICONTL and after MAIN to show the affect of the Reader/Interpreter (R/I) on the job's JCL. The default control blocks were selected, if the user has specified BLOCK=, only the selected control blocks will be printed.

The first JDAB control block formatted shows fixed portion and the generated Scheduler Elements (SE). The first SE is created by ISDRVR to provide the time-on and the time-off for when Input Service was creating this job. This information is provided to the user for accounting information. The SE's one through seven are for the supplied //*PROCESS cards plus PURGE.

The FRP control block is the formatted //*FORMAT cards supplied for Print and Punch Service.

JDS control block entries are the data set entries for this job. Some of the entries are created by Input Service for the R/I. JCLIN is a data set of the actual JCL as used by the physical card reader. SYSMSG is the data set to contain the OS SMB's.

The JCBTAB data set entry is created by Input Service for ASP R/I to place the logical TTR table to be used by ASP R/I control block access method. JCBIN is the data set reserved for the OS control blocks which are read by the ASPQRDR routine on the Main Processor. This data set will contain the actual SYS1.SYSJOBQE TTRs of the OS control blocks. ASPI0001 contains the DD* or the DD DATA information. ASP0001 is a dummy entry created because a //*FORMAT control card was supplied.

The Job Setup Table (JST) is created by Input Service but will not be completed until the ASP R/I has completed all of the required work. A completed JST is shown later in this example. The JOB TAT control block (JBTAT) indicates the one-track group has been allocated to this job. Following the JBTAT are the LOCATE and responses generated by the ASP R/I DSP. The OS SMB's that are retrieved via the ASPQWTR interface in MAINIO. By comparing the SMB output with the supplied JCL it should be noted that the ASP R/I DSP does not process the ASP control cards. The ASP control cards are not placed in the JCLIN data set.

The next JDAB, FRPs formatted indicates how it appears after Main Service. By comparing the JDS control block entry now with the previous, it can be seen that the JCBTAB and JCBIN are completed and have been utilized.

Compare the present JST with the previous JST and the supplied JCL.

```
 ISV40 JCB ORIGIN FROM GROUP=MACH2    , DSP=CR , DEVICE=RD2      , 039
//CBPRNT    JOB  836,ASP
//*PROCESS CBPRNT
//*PROCESS RICONTL
//*PROCESS MAIN
//*PROCESS CBPRNT
//*PROCESS PRINT
//*FORMAT   PR,DDNAME=ASP0001,DEST=MACH2,TRAIN=PN,COPIES=2
//*PROCESS PUNCH
//*FORMAT   PU,DDNAME=JCLIN
//S1        EXEC  PGM=IEFBR14
//ASP0001   DD   SYSOUT=A
//T1        DD   SYSOUT=Z
//S2        EXEC  PGM=IEFBR14
//DD2       DD   DSN=ASP.WOOF305,DISP=SHR
//SYSIN     DD   *
/*


          ***********************************************************************************************
          ***********************************************************************************************
          ***********************************************************************************************
          *                                ASP CONTROL BLOCK PRINT OUTPUT                                *
          ***********************************************************************************************
          ***********************************************************************************************
          ***********************************************************************************************

DEFAULTS USED - BLOCK=OSCB,JST,JDS,FRP,TAT,JDAB


          **********************************************************************************************
          *                                 JDAB CONTROL BLOCK                                         *
          **********************************************************************************************

FIXED   0000 01023001000000000000000001013001 E4900001C3C2D7D9D5E34040D4C1C3C8 *...............U...CBPRNT  MACH*
        0020 F2404040F0F0F0F3FFFFFFFF00000000 000000000101300E690000000005C005 *2   0003................W.......*
        0040 00000000000000000800001000000000 000000000000000006161C3C2D7D9D5E3 *..........................CBPRNT*
        0060 404040D1D6C24040F8F3F66BC1E2D740 40404040404040404040404040404040 *   JOB  836.ASP                *
        0080 40404040404040404040404040404040 40404040404040404040404040404040 *                                *
        00A0 40404040404040002700000000000000 00000000000000000000C00000000000 *          .......................*
        00C0 00000000000000000000000000000000                                   *................               *

SE #    0000 00000000000000252037262037280000 00000000000000000000000000100000 *................................*
0000    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 000000000000012BEEEEEEEEEEEE0000 00000000000000000000000000000000 *................................*
0001    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 000000000000020FEEEEEEEEEEEE0000 00000000000000000000000000000000 *................................*
0002    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 000000000000030BEEEEEEEEEEEE0000 00000000000000000000000000000000 *................................*
0003    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 000000000000042BEEEEEEEEEEEE0000 00000000000000000000000000000000 *................................*
0004    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 01033000E4900509EEEEEEEEEEEE0000 00000000000000000000000000000000 *....U...........................*
0005    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 01043000E490060AEEEEEEEEEEEE0000 00000000000000000000000000000000 *....U...........................*
0006    0020 00000034000000000000000000000000 00000000                          *....................            *

SE #    0000 0000000000000708EEEEEEEEEEEE0000 00000000000000000000000000000000 *................................*
0007    0020 00000034000000000000000000000000 00000000                          *....................            *
```

```
*****************************************************************************************************
*                              FRP CONTROL BLOCKS FOR PRINT + PUNCH DSPS                            *
*****************************************************************************************************

*   PRINT SCHEDULER ELEMENT

FIXED  0000 0103300000000000000002C00000000 00000000000000000000000C0000000C *.............................*
       0020 00000000000000000000000                                          *............                 *

FRP #  0000 D4C1C3C8F24040000000000000000000 E2E3C1D5C4C1D9C4E2E3C1D5C4C1D9C4 *MACH2    ........STANDARDSTANDARD*
0001   0020 D7D5404040404040030200000000000 00440000000000000000000C1E207F0 *PN       ..................ASP0*
       0040 F0F0F140                                                         *001                        *

*   PUNCH SCHEDULER ELEMENT

FIXED  0000 0104300000000000000002C00000000 00000000000000000000000000000000 *.............................*
       0020 00000000000000000000000                                          *............                 *

FRP #  0000 D4C1C3C8F24040000000000000000000 E2E3C1D5C4C1D9C4E2E3C1D5C4C1D9C4 *MACH2    ........STANDARDSTANDARD*
0001   0020 E2E3C1D5C4C1D9C4030100000000000 00440000000000000000000D1C3D3C9 *STANDARD.................JCLI*
       0040 D5404040                                                         *N                          *


*****************************************************************************************************
*                                   JDS CONTROL BLOCK                                               *
*****************************************************************************************************


FIXED  0000 0101300100000000000012800200000 00000000000000000000000000000000 *.............................*

JDS #  0000 0108300010980000000000680006058C 01CB300000000044002C0000000C000C *.............................*
0001   0020 00000000D1C3D3C9D5404040                                          *....JCLIN                  *

JDS #  0000 0005F9A810880000010002690005EAC8 0106300000000080002C0C0000000000 *..9.....................*
0002   0020 00000000E2E8E2D4E2C74040                                          *....SYSMSG                 *

JDS #  0000 0C000000000000000000C0C000000000 0C0000000000010000020000000000 *.............................*
0003   0020 00000000D1C3C2E3C1C24040                                          *....JCBTAB                 *

JDS #  0000 0000000000000000000000000000000C 0C0000000000000002C0000000000 *.............................*
0004   0020 0C00000001C3C2C9D5404040                                          *....JCBIN                  *

JDS #  0000 0107300010980000000003040006062C 0107300000000000002CC000000000 *.............................*
0005   0020 00000000C1E2D7C9F0F0F0F1                                          *....ASPI0001               *

JDS #  000C 0C0C000000000000000000000000000C 00000000000000088002C000000000C *.............................*
0006   0020 00000000C1E2D7F0F0F0F140                                          *....ASP0001                *


*****************************************************************************************************
*                                   JST CONTROL BLOCK                                               *
*****************************************************************************************************


FIXED  0000 0102300000000000000000C28000C00CC CC0000028000000C0000000000000C00C *.............................*
       0020 0000000000000000                                                 *........                   *


*****************************************************************************************************
*                                   JOB TAT CONTROL BLOCK                                           *
*****************************************************************************************************


FIXED  CCC0 0101300001063002C08E0014E3C1E300 0000000100010000                *............TAT.........  *


JBTAT  0000 0C0000000000000000000000000000000 0000000000001000000C000000000C *.............................*
       0020 0C000CC0000000000000000000000000 0000000000000000000000000000000C *.............................*
       0040 000000000000C000000000000000000C 0C0000000C0000C000000C000000000C *.............................*
       0060 C000000000000000C0000000000000000 0000000000000000000000000000000C *.............................*
       0080 C000000000000000000000000C0C0CC0C 0000000000000000000000000000000C *.............................*
       00A0 0000000000000000000000000000000C 0000000000000000000000000000000C *.............................*
       00C0 000C000C000C0C000000000C000C000C 0C0000000000000000000000000000000C *.............................*
       00E0 0000000000000C0000000000000000000 0C0C000000000000000000000000000C *.............................*
       0100 0000000000000000000000C00C0000000 0C0000000000C0000000C00000C0000C *.............................*
       0120 0000000000000000000000000000000000 0000000000000000000000000000000C *.............................*

LOCATE' 0003ASP.WOOF305
ALC00030E001/ASP3050004


 AMDS01 JOB 0003 (CBPRNT  ) IN SETUP ON MAIN=SY2
 AMDS02 CD2      USING D ASP305 ON 232
//CBPRNT   JOB  836,ASP
//S1       EXEC  PGM=IEFBR14
//ASP0001  DD   SYSOUT=A
//T1       DD   SYSOUT=Z
IEF236I ALLOC. FOR CBPRNT   S1
IEF237I 276   ALLOCATED TO ASP0001
IEF237I 131   ALLOCATED TO T1
IEF142I - STEP WAS EXECUTED - COND CODE 0000
IEF285I   SYS72270.T203644.SV001.CBPRNT.R0000001       DELETED
IEF285I   VOL SER NOS= ASPUT3.
//S2       EXEC  PGM=IEFBR14
//CD2      DD   DSN=ASP.WOOF305,DISP=SHR
//SYSIN DD UNIT=(CTC,,DEFER),DSNAME=&&ASPI0001,                          *
//             VOLUME=SER=010003,DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
//
IEF236I ALLOC. FOR CBPRNT   S2
IEF237I 232   ALLOCATED TO CD2
IEF237I 376   ALLOCATED TO SYSIN
IEF142I - STEP WAS EXECUTED - COND CODE 0000
IEF285I   ASP.WOOF305                                  KEPT
IEF285I   VOL SER NOS= ASP305.
IEF285I   SYS72270.T203644.RV001.CBPRNT.ASPI0001       DELETED
IEF285I   VOL SER NOS= 010003.
IEF375I  JOB /CBPRNT  / START 72270.2037
IEF376I  JOB /CBPRNT  / STOP  72270.2037 CPU   0MIN 00.06SEC
 AMDS09 JOB 0003 (CBPRNT  ) IN BREAKDOWN
```

```
        *****************************************************************************************
        *                                  JDAB CCNTRCL BLOCK                                   *
        *****************************************************************************************

FIXED   0000 01023001000000000000000C001013001 E4900001C3C2D7D9D5E34040D4C1C3C8 *...............U...CBPRNT  MACH*
        0020 F2404040F0F0F0F3FFFFFFFFE2E8F240 40404040010130000E690000000050005 *2   0003....SY2      ....W.......*
        0040 00000000000000001800001000000000 0000000001000000616lC3C2D7D9D5E3 *.........................CBPRNT*
        006C 404040D1D6C24040F8F3F66BC1E2D740 40404040404040404040404040404040 *   JOB  836.ASP                *
        0080 4C404040404040404040404040404040 4C404040404040404040404040404040 *                               *
        00A0 40404040404040400270000000000000 00000000000000000000000000000000 *        .......................*
        00C0 0C000000000000000000000000000000                                  *................               *

SE #    0000 000000000000025203726203728000C0 0C000000000000000000000000100000 *...............................*
0000    0020 000000340000000000000000000000 0C000000                         *...................             *

SE #    0000 0000000000012B20372820373000C0 00000000000000000000000000000000 *...............................*
0001    0020 000000340000000000000000000000 00000000                         *...................             *

SE #    0000 0C000000000020F2037302037340CC C0000000000000000000000000000000 *...............................*
0002    0020 00000C3400000000000000000000000 00000000                         *...................             *

SE #    0000 0C00000000030B2037412037560CC 0CC00000000000000000000000000000 *...............................*
C003    0020 000000340000000000000000000000 00000000                         *...................             *

SE #    00C0 000000000000042BEEEEEEEEEEEE0000 00000000000000000000000000000000 *...............................*
0004    0020 00000C340000000000000000000000 00000000                         *...................             *

SE #    0000 01033000E4900509EEEEEEEEEEEE0000 00000000000000000000000000000000 *....U..........................*
0005    C020 00000034000000000000000000000C 00000000                         *...................             *

SE #    0000 01043000E490060AEEEEEEEEEEEE0000 00000000000000000000000000000000 *....U..........................*
0006    0020 000000340000000000000000000000 C0000000                         *...................             *

SE #    0000 00000000000C0708EEEEEEEEEEEE0000 00000000000000000000000000000000 *...............................*
0007    0020 00000C3400000000000000000000000 00000000                         *...................             *


        *****************************************************************************************
        *                          FRP CONTROL BLOCKS FOR PRINT + PUNCH DSPS                    *
        *****************************************************************************************


*   PRINT SCHEDULER ELEMENT

FIXED   0000 0103300000000000000002C00000000 00000000000000000000C0000000000 *...............................*
        C020 00000C00000C000000000000                                          *...........                     *

FRP #   0000 D4C1C3C8F24C40400000000000000000 E2E3C1D5C4C1D9C4E2E3C1D5C4C1D9C4 *MACH2     .......STANDARDSTANDARD*
0001    0020 D7D54040404C40400302000000000000 0044000000000000000000000C1E2D7F0 *PN        ...................ASP0*
        0040 F0F0F140                                                          *001                            *

*   PUNCH SCHEDULER ELEMENT

FIXED   C000 01043000000C00000000002C00000000 0000000000000000000C0000000000 *...............................*
        0020 000000000000000000000000                                          *...........                     *

FRP #   0000 D4C1C3C8F24C40400C0000000000000000 E2E3C1D5C4C1D9C4E2E3C1D5C4C1D9C4 *MACH2     .......STANDARDSTANDARD*
0001    0020 E2E3C1D5C4C1D9C40301000000000000 0C4400000000000000000000D1C3D3C9 *STANDARD...................JCLI*
        0040 D5404040                                                          *N                              *


        *****************************************************************************************
        *                                  JDS CONTRCL BLOCK                                    *
        *****************************************************************************************


FIXED   0000 010130010000C0000000128002C0000 00000000000000000000000000000000 *...............................*

JDS #   0000 010830009038000000000006800060290 01083000000001044002C0C0000C00000 *...............................*
C001    0020 00000000D1C3D3C9D5404040                                          *....JCLIN                       *

JDS #   0000 0005ED381088000002000040005F04C 01053005000010800002C000000000000 *.............0.................*
0002    0020 00000000E2E8E2D4E2C74040                                          *....SYSMSG                      *

JDS #   C000 01033004E0380000000001B40005E548 01033004000010000002C000000000000 *.............V.................*
C003    0020 00000000D1C3C2E3C1C24040                                          *....JCBTAB                      *

JDS #   0000 01043004F0380000000000240005E9F4 01013005000000000002C000000000000 *....0.........Z4................*
0004    0020 0C0000000D1C3C2C9D5404040                                         *....JCBIN                       *

JDS #   0000 0107300010980000000003040006062C 01073000000001000002C000000000000 *...............................*
C005    0020 00000000C1E2D7C9F0F0F0F1                                          *....ASPI0001                    *

JDS #   0000 00000000000000000000000000000000 000000000000088002C000000000000 *...............................*
0006    0020 00000000C1E2D7F0F0F0F140                                          *....ASP0001                     *
```

```
*****************************************************************************************************
*                                    JST CONTROL BLOCK                                              *
*****************************************************************************************************

FIXED  0000 0102300000000000000000AC000000C0 0000002800200C1000100033801020C *.....................................*
       0020 0000000000000000                                                 *........                           *

STEP   0000 F1000002E2F1404040404040404C4040 4040404OF0F0F5F2F0F0F0F00002C00C *1...S1                 00520000....*
0001   0020 0100000000CC0000C0000000                                         *............                       *

STEP   0000 F1000000E2F2404040404040404040 4040404OF0F0F5F2F0F0F0F000000000 *1...S2                 00520000....*
0002   0020 0200000000000000000C0000                                         *............                       *

DD     0000 F2E20000C4C4F2404040404OF2F3F1F4 4040404OC1E2D7F3F0F5F2F3F200400D *2S..DD2      2314     ASP305232. .*
0001   0020 0000000000008000000000000                                        *............                       *


*****************************************************************************************************
*                                ASP R/I CONTROL BLOCKS BEFORE MAIN SERVICE                         *
*****************************************************************************************************

JCT    0000 0000010001000190C3C2D7E9C5E34040 0CC0000000000000000000000000000C *.......A.CBPRNT  ...............*
0001   0020 0000020000000700000003000001DC0 0000000000000000000000000000000C *.................................*
       0040 0000000000000C0C0000C0000C0000000 0000000000000000000001B0000000001 *.................................*
       0060 0000000000000000000000000000000C 0000020000004040404040404040404C *.........................       *
       0080 4040404040400000060CA000C0000000 0000000000000000000000000000C00C *       ...........................*
       00A0 0000000000000000000000000000000C                                 *...............                    *

SMB    0000 0000070500000800000000000A30000 0000000000000000000000000000000C *.................................*
0007   0020 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0040 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0060 0000000000000000000000000000000C 00C0000000000000000000000000000C *.................................*
       0080 0000000000000000000000000000000C 0C0000000000000000000000000000C0 *.................................*
       00A0 0000000000000000000000000000C0C0                                 *...............                    *

SMB    0000 0000080500000090000000000062FE19 0E6161C3C2D7D9D5E303030D1D6C2C2D7 *..................CBPRNT..JOB..*
0008   0020 F8F3F668BC1E2D7FE1A046161E2F10704 C5E7C5C3020BD7C7D47EC9C5C6C2D9F1 *836.ASP.....S1..EXEC..PGM.IEFBR1*
       0040 F4FE1A096161C1E2D7FCFCF0F10202C4 C402D8E2E8E2D6E4E37EC1FE15046161 *4.....ASP0001..DD..SYSOUT.A.....*
       0060 E3F10702C4C402D8E2E8E2E6E4E37EE9 0000000000000000000000000000000D *T1..DD..SYSOUT.Z.................*
       0080 0000000000C0000000000C000000000C 0C0000000000000C0000C000000000C0 *.................................*
       00A0 0000000000000000000000000C0000C0 0000000000000000000000000000000C *.................................*

SMB    0000 0000013050000000000C0000000003FFE3A 0261610D33E5D6D3E4D4C57EE2C5D97E *...................VOLUME.SER.*
0013   0020 F0F1F0F0F0F36BC4C3C27E4DD3D9C5C3 D37EF8F06BC2D3D2E2C9E5C57EF8F06B *010003.DCB..LRECL.80.BLKSIZE.80.*
       0040 D9C5C3C6D47EC65DFE05026161000000 0C0000000000000000000C0000000000 *RECFM.F.........................*
       0060 0000000000000000000000000C000000 0C0000000000000000000000000000C0 *.................................*
       0080 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       00A0 0C0000000000000000000C0C0000C0000 0000000000000000000000000000000C *.................................*

ACT    0000 0000030100000C0C00C1E2D74040404040 4040404040404040404040C4000000001 *........ASP                  ....*
0003   0020 03F8F3F6000000000000000000000000C 0000000000000000000000000000000C *.836.............................*
       0040 0000000000000000000000000C0000000 0C0000000C0000C0C0C0000000C0000C *.................................*
       0060 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0080 0000000000000000000000000000000C 0C0000000C0000000C0000000000000C *.................................*
       00A0 0000000000000000000000000000000C                                 *...............                    *

DSNC   0000 00001B0F0000000000000000B0001800B C1E2D74BE6D6D6C6F3F0F500000000000 *...............ASP.WOOF305.....*
001B   0020 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0040 0000000000000000000000000000000C 0CC0000000C0000000C0000000000000 *.................................*
       0060 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0080 0C0000000000C0000000000C0C0000000 0C0000000000000000000000000000C0 *.................................*
       00A0 0000000000000000000000000000000C                                 *...............                    *

SCD    0000 C1000201000001E907110301071100C0 0000000000000000000000000000000C *A.......Z.......................*
001D   0020 0000000000C0C0000000000000000000 0000000000000000C0000C0000000000 *.................................*
       0040 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0060 0000000000000000000000C0C0000C0C 0C0000000C0000000C0000000C0000C0 *.................................*
       0080 0000000000000000000000000000000C 0000000000000000000C0000000000C0 *.................................*
       00A0 0C0000000000C0000000C0000C0000C0000 0000000000000000000000000000000C *.................................*

JMR    0000 C3C2D7D9D5E34C4000714C250072270F E2F1F5F0404040404040404002A0C000 *CBPRNT  .........S150         ....*
0006   0020 0000000000000000000714CCA00722 70F 0C0000000000A0000000000000000000 *.................................*
       0040 0000000000000000000000000000000C 0000000000000000CC0C0CC0C0C0000000 *.................................*
       0060 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0080 0C0000000000000000000000000000C0 0C0000000000000000000C0000000000 *.................................*
       00A0 0000000000000000000000000000000C                                 *...............                    *

SCT    0000 0000020200058761000000002000000500 0000000000000D0000000C5000000C80C *.................................*
0002   0020 0000110000000000000000040404040 40404040E2F1404040404040000000000 *............        S1       ....*
       0040 0200000000000000000000001000000 0C0000000C0000000034C0000000100B *.................................*
       0060 0000000000000000000000000C9C5C6C2 D9F1F440000000000000C0C0000000000 *............IEFBR14 ............*
       0080 0C0000000C00C0000000C0000000000000C0 0C0000000000000000000000000000C0 *.................................*
       00A0 0000000000000000000000000000000C                                 *...............                    *

ACT    0000 0000110100000000C000C0000000000C 0C0000D0000000000000000C0000000000 *.................................*
0011   0020 0000000000000000000000C0000000000 0000000000C0000000000C0000000C000 *.................................*
       0040 0000000000000000000000000000000C 0000000000000000000000000000000C *.................................*
       0060 000C0000000000000000000000C0000C0 0C0000000000000C0000C0C000000C000 *.................................*
       0080 0C0000000000000000000000000000C0 0000000000000000000000C0000000000 *.................................*
       00A0 CC0C0C000000C0000000000CCCCCCCCCCC                               *...............                    *
```

```
SIOT    0000 00000503C1E2D7F0F0F0F14000000000 00000000000000000000000000000CA00 *....ASP0001 ....................*
0005    0020 0000040000000000000000000000000000 00000000010100040002040 0C3E3C340 *...........................CTC *

        0040 404040404040404040404040404040404040 40000000000000000000000000000000 *                 ..............*
        0060 00000000000000000000000000000000 000000000000000000000C1E2D7D6C1F0 *.........................ASPOA0*
        0080 F0F1404040404040404040404040404040 40404040404040404040404040404040 *01                              *
        00A0 40404040404040404040404040400000                                      *               ..               *

JFCB    0000 E2E8E2F7F2F2F7F04BE3F2F0F3F6F4F4 4BD9E5F0F0F14BC3C2D7D9D5E34BC1E2 *SYS7227C.T203644.RV001.CBPRNT.AS*
0004    0020 D7D6C1F0F0F1404040404040404040 40404040000000000000000000000000 *POA001                ............*
        0040 00000200000000000000000000000000 4EC10E00000C00C100000C00C000C000 *.......................A........*
        0060 00000000000000000000000000000000 00000000000040404040404040404040 *.......................          *
        0080 40404040404040404040404040404040 40404040000000000000000000000000 *                    ............*
        00A0 00000000000000000000000000000100                                      *...............                 *

SIOT    0000 00000A03E3F1404040404040000000000 0000000000000000C000C000C000C800C *....T1          ..................*
000A    0020 00000F00000000000000000000000000 000000000201000400000C00E2E8E2C4 *..............................SYSD*
        0040 C1404040404040404040404040404040 E5C000000000E000001C0000000C000 *A                 Z..............*
        0060 00000000000000000000000000000000 00000000000000000000404040404040 *...........................     *
        0080 40404040404040404040404040404040 40404040404040404040404040404040 *                                *
        00A0 40404040404040404040404040400000                                      *               ..               *

JFCB    0000 E2E8E2F7F2F2F7F04BE3F2F0F3F6F4F4 4BE2E5F0F0F14BC3C2D7D9D5E34BD9F0 *SYS7227C.T203644.SV001.CBPRNT.R0*
000F    0020 F0F0F0F0F0F1404040404040404040 40404040200000000000000000000000 *000001                ............*
        0040 00000200000000000000000000000000 4E010E000000008100000C0000000000 *................................*
        0060 00000000000000000000000000000000 C0000000000040404040404040404040 *.......................          *
        0080 40404040404040404040404040404040 40404040000000000000C32800000CA00 *                    ............*
        00A0 00000000000000000000000000000C1C0                                     *...............                 *

SIOT    0000 00000803C4C4F240404040400000000000 000000000000000C00000C000000001200 *....DD2          ..................*
000B    0020 00000C00000000000000000000000000 00000000010100000000010040404040 *............................    *
        0040 40404040404040404040404040404040C 4C000000000000C000000000C000C000 *                 ..............*
        0060 00000000000000000000000000000000 000000000000000000004040404040 *...........................     *
        0080 40404040404040404040404040404040 40404040404040404040404040404040 *                                *
        00A0 40404040404040404040404040400000                                      *               ..               *

JFCB    0000 C1E2D74BE6D6D6C6F3F0F540404040 40404040404040404040404040404040 *ASP.WOOF305                     *
000C    0020 40404040404040404040404040404040 40404040000000000000000000000000 *                    ............*
        0040 00000200000000000000000000000000 48010E00000000480000000000000000 *................................*
        0060 00000000000000000000000000000000 C0000000000040404040404040404040 *.......................          *
        0080 40404040404040404040404040404040 40404040000000000000000000000000 *                    ............*
        00A0 00000000000000000000000000000C1C0                                     *...............                 *

SIOT    0000 00001203E2E8E2C9D54C4C400000000000 000000000000000000000000000001700 *....SYSIN        ..................*
0012    0020 00001500000000000000000000000000 00C100010201000400020400C3E3C340 *...........................CTC *
        0040 40404040404040404040404040404040 4C000000000000C000000C000C000C000 *                 ..............*
        0060 00000000000000000000000000000000 000000000000000000000C1E2D7C9F0F0 *.........................ASPI00*
        0080 F0F1404040404040404040404040404040 40404040404040404040404040404040 *01                              *
        00A0 40404040404040404040404040400000                                      *               ..               *

****************************************************************************************
*                           ASP R/I CONTROL BLOCKS AFTER MAIN SERVICE                  *
****************************************************************************************

CMPA    0000 C3C2D7D9D5E340400106E10500000228 00020000006F1000000000000006AEFC *CBPRNT  .............1.........0*
        0020 00020050                                                              *....                            *

JCT?    0000 004505000100C290C3C2D7D9D5E34040 00000000000000000000000000000000 *.......B.CBPRNT ...............*

4505    0020 0045060000460800004507000004519C0 00000000000000000000000000000000 *................................*
        0040 00000000000000000000000000000000 00000000000000004518000000001 *................................*
        0060 00000000000000000000000000000000 00450600000040404040404040404040 *.......................          *
        0080 40404040404000450A00A00000000000 00000000000000000000000000000000 *        ........................*
        00A0 00000000000000000000000000000000                                      *...............                 *

SMB?    0000 00460805004600000000000000A30000 00000000000000000000000000000000 *................................*
460B    0020 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0040 00000000000000000000000000000000C 00C00000C00C0000000000C000000000C *................................*
        0060 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0080 00000000000000000000000000000000C 00000000000000000000000000000000 *................................*
        00A0 00000000000000000000000000000000                                      *...............                 *

SMB?    0000 00460C0500460D000000000000062FE19 C86161C3C2D7D9D5E30303D1D6C20207 *.................CBPRNT..JOB..*
460C    0020 F8F3F66BC1E2D7FE1A046161E2F10704 C5E7C5C3020807C7D47EC9C5C6C2D9F1 *836.ASP.....S1..EXEC..PGM.IEFBR1*
        0040 F4FE1A096161C1E2D7F0F0F0F10202C4 C40208E2E8E2D6E4E37EC1FE15046161 *4.....ASP0001..DD..SYSOUT.A.....*
        0060 E3F10702C4C40208E2E8E2D6E4E37EE9 00000000000000000000000000000000 *T1..DD..SYSOUT.Z................*
        0080 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        00A0 00000000000000000000000000000000                                      *...............                 *

SMB?    0000 00460D0500460E00000000000077FE1A 046161E2F20704C5E7C5C3020807C7D4 *..................S2..EXEC..PGM*
460D    0020 7EC9C5C6C2D9F1F4FE26056161C4C4F2 0602C4C40218C4E2D57EC1E2D74BE6D6 *.IEFBR14.....DD2..DD..DSN.ASP.WO*
        0040 D6C6F3F0F56BC4C9E2D77EE2C8D9FE37 076161E2E8E2C9D50102C4C40124E4D5 *OF305.DISP.SHR.....SYSIN..DD..UN*
        0060 C9E37E4DC3E3C36B68C4C5C6C5D95D6B C4E2D5C1D4C57E5050C1E2D7C9F0F0F0 *IT..CTC..DEFER..DSNAME...ASPI000*
        0080 F16B18015C0C00000000000000000000 00000000000000000000000000000000 *1...............................*
        00A0 00000000000000000000000000000000                                      *...............                 *

SMB?    0000 00460E0500000000000000003FFE3A 0261610D33E50603E404C57EE2C5D97E *.....................VOLUME.SER.*
460E    0020 F0F1F0F0F0F36BC4C3C27E4D03D9C5C3 D37EF8F06BC2D3D2E2C9E9C57EF8F06B *010003.DCB..LRECL.80.BLKSIZE.80.*
        0040 D9C5C3C6D47EC65DFE05026161000000 00000000000000000000000000000000 *RECFM.F.........................*
        0060 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0080 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        00A0 00000000000000000000000000000000                                      *...............                 *

ACT?    0000 004507010000000C1E2D74040404040 40404040404040404040404000000001 *........ASP                  ....*
4507    0020 03F8F3F6000000000000000000000000 00000000000000000000000000000000 *.836............................*
        0040 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0060 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0080 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        00A0 00000000000000000000000000000000                                      *...............                 *

CSNQ?   0000 0045180F000000000000000B00018008 C1E2D74BE6D6D6C6F3F0F50000000000 *................ASP.WOOF305.....*
4518    0020 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0040 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0060 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        0080 00000000000000000000000000000000 00000000000000000000000000000000 *................................*
        00A0 00000000000000000000000000000000                                      *...............                 *
```

```
SCT?    0000 0045060200587610000000020045090C 0000000000450E000046CD0000460C00 *.....*
4506    0020 00450F0000000000000000C040404040 40404040E2F1404040404040000000000 *.....          S1      ....*
        0040 020000000000000000000000CC1000000 0CCC000000000000003400000001008E *.....*
        0060 0000000000000000000000000C9C5C6C2 D9F1F440000000000000000000000000 *............IEF8R14 .....*
        0080 0000000000000000000000000000000C C0000000000000000000000000000000 *.....*

        00A0 0CCCCC0000000000000000000000000000                                 *.....                *


ACT?    0000 00450F0100000000000000000000000000 0000000000000000000000000000000 *.....*
450F    0020 0000000000000000000000000000000000 0000000000000000000000000000000 *.....*
        0040 0000000000000000000000000000000000 0CC0000000000000000000000000000 *.....*
        0060 0000000000000000000000000000000000 0000000000000000000000000000000 *.....*
        0080 CC0000000000C0000000000000CC0000000 0000000000000000000000000000000 *.....*
        00A0 0000000000000000000000000000000000                                 *.....                *


SIOT?   0000 00450903C1E2D7FCF0FCF140000C0000 0000000000000000000000000450B00 *....ASP0001 .....*
4509    0020 0045080000000000000000000000000000 0C0000000101000400020400C3E3C340 *.....CTC *
        0040 4040404040404040404040404040404040 4000000000000000000000000000000 *                .....*
        0060 000C000000000C0000000000000000000C 0C0000000000000000C1E2D7D6C1F0 *.....ASPOA0*
        0080 FCF1404040404040404040404040404040 4040404040404040404040404040404 *01                              *
        00A0 4C4040404040404040404040404CC000                                   *             ..               .*


????    0000 E2E8E2F7F2F2F7F04BE3F2F0F3F6F4F4 4BD9E5F0F0F14BC3C2D7D9D5E34BC1E2 *SYS7227C.T203644.RV001.CBPRNT.AS*
4508    0020 D7D6C1F0F0F1404040404040404040404 4040404000000000000000000000000 *POA001                 .....*
        0040 0000020000000000000000000000000000 48010E00000000C1000C0C0000000000 *.....A.....*
        0060 0000000000000000000000000000000000 0C0000000000404040404040404040 *.....                *
        0080 4040404040404040404040404040404040 4C4040400000C0000000000000000000 *                    .....*
        00A0 0C0000000000000000000000000000100                                  *.....                *


SIOT?   0000 00450B03E3F1404040404040000000000 0000000000000000000000C0000450C00 *....T1      .....*
4508    0020 0047020000000000000000000000000000 0000000002010004000000C00E2E8E2C4 *.....SYSD*
        0040 C1404040404040404040404040404040 E50000000047010000470300000000000 *A                Z.....*
        0060 0000000000000000000000000000000000 0000000000000000004040404040404 *.....          *
        0080 4040404040404040404040404040404040 4040404040404040404040404040404 *                    *
        00A0 4040404040404040404040404040000000                                 *             ..               *


????    0000 E2E8E2F7F2F2F7F04BE3F2F0F3F6F4F4 4BE2E5F0F0F14BC3C2D7D9D5E34BD9F0 *SYS7227C.T203644.SV001.CBPRNT.RO*
4702    0020 F0F0F0F0F0F1404040404040404040404 4040404020000000C000C00000000000 *000001                 .....*
        0040 0000020000000000000000000000000000 48010E0000000081000000000000000 *.....*
        0060 000C0C0000000000000000000000000C0C 0CC0000000004040404040404040404 *.....          *
        0080 4040404040404040404040404040404040 4040404000000000000328000000A00 *                    .....*
        00A0 0C0000000000000000000000C0000C0100                                 *.....                *


JCT?    0000 0C0000000000C0C000000C0000000000000 0000000000000000000000C0000000000 *.....*
4703    0020 0000000000000000000000000000000000 0C0000000000000000000000000000000 *.....*
        0040 0C0C0000000C0000000000C0000C0000 0CC0000000000000CC000CC000000C000 *.....*
        0060 0C0000000000000000000000000000000000 0000000000000000000000000000000 *.....*
        0080 0C0000000000C0000000000000C0000000 C0C0000000000000C000000C0000000000 *.....*
        00A0 0000000000000000000000000000000000                                 *.....                *


SIOT?   0000 00450C03C4C4F24040404C4000000000 0CC0000000000000000000C0000451000 *....DD2      .....*
450C    0020 00450D0000000000000000000000000000 0000000001010000000001004040404 *.....*
        0040 4040404040404040404040404040404040 4C0000000000000000000C0000000000 *                .....*
        0060 0000000000000000000000000C0000000000 0000000000000000000004C4040404040 *.....*
        0080 4040C4040404040C4C4C4C4C4C4C4040 4040404040C4040404040404040404040 *                    *
        00A0 4040404040404040404C4040404C00000                                  *             ..               *
```

TERMINATING A DSP VIA THE FAIL COMMAND

There may be conditions where a DSP will not respond to an operator *CANCEL command. The *FAIL command causes the specified function to immediately terminate and enter ASP Failsoft recovery, which will return the resources held by the function. If DUMP is specified, an ASPABEND dump will be taken and the system will proceed normally.

Operation	Operands
FAIL	{dspname|device-name|device-address}[,DUMP]

SNAP DUMPING ASP CONTROL BLOCKS USING DISPLAY/DC DSP

Under some circumstances it may be desirable to examine ASP control blocks (see below) for all jobs in the system (for example to determine flag settings and field values in these control blocks which may be incorrect or adversely affecting the system). This can be accomplished using the display DSP in conjunction with the DC DSP (password mode) in the following manner:

1. *X DC[,KEY=password] (See Dump Core DSP)
2. *S DC,SNAPON

   Note: This will enable SNAP processing for any DSP coded with this capability (for example, RJP).

3. *X DISPLAY,SNAPS[,OUT=printer name]
4. *C DC

The following ASP control blocks will be snapped to the SNAP data set (ASPSNAP DD card in the ASP initialization deck) for all ASP jobs in the system:

1. Job Control Table - ID=001
2. Job Description and Accounting Block - ID=002
3. Job Setup Table - ID=003
4. Specialized Reschedule Block - ID=004

See ASP Console Operator's Manual for a discussion of normal DISPLAY usage.

APPENDIX A: MACRO-INSTRUCTIONS

This appendix contains a discussion of each of the macro-instructions used in the ASP system. Each discussion consists of three elements:

- A functional description defining the purpose and use of the macro
- An illustration of the macro format
- A definition of each of the operands used in the macro

For the convenience of the reader, the macros are arranged in alphabetical order.

The symbols used to express the operands are:

addr	A symbol of from one to eight characters, with the first character alphabetic
(reg)	Absolute register notation in the range 2 through 9 and 13. This type of specification must be enclosed in parentheses. If the user has included the ASP REGISTER macro in his program, either form (n) or (Rn) may be used.
(Rn)	Where n is shown as a specific register number, it represents the register used by the called routine to receive this parameter. If the register specified as (n) or (Rn) is given, no redundant load instruction is generated.
n	A decimal number
X'xx'	A hexadecimal number for X in the range 00 through FF
[ ]	An optional operand
{ }	A mandatory operand containing alternative entries
|	A separator used between alternate operands

Where an operand defining a return location to the program is shown as optional, such as [NORMAL={(reg)|addr}], omission causes resumption of the program at the location immediately following the expansion of the macro-instruction. In this event, a branch instruction is not generated; this is, therefore, the most efficient usage of the macro-instruction. Registers 0, 1, 14, and 15 are loaded destructively by the macro-expansions and cannot be used to pass parameters except as specifically noted in the individual descriptions of the macro-instruction.

SAVE={YES|NO}, which appears as an optional parameter in most ASP macro-instructions generating calling sequences, allows the user to specify whether registers 2 through 9 are to be saved across the call. Registers 10 through 13 are always restored before returning to the caller. The normal use of the option, which is the default, is SAVE=YES. SAVE=NO, which is faster, is recommended only where the user has no further use of registers 2 through 9, for example, on a macro call immediately preceding the exit from the calling function. In the macros herein defined, where the option is not shown, SAVE=YES is forced (that is, registers 2 through 13 are saved) unless specifically noted in the text.

Several macro-instructions are provided to define as DSECTs the fields of blocks and tables used by the system; for example, AJDENTRY, JDABDSCT, and TVTABLE. Each routine that uses such ASP blocks and tables should use the macro-instructions for field definition in order to ensure consistency in the labels used and to simplify modification of the size and/or format of the blocks and tables.

Several of the tables consist of a fixed area plus a variable number of entries. The following addressing conventions are recommended to provide maximum flexibility in table expansion with a minimum of reassembly.

To access the fixed area:

- Load a register with the address of the table.
- Issue a USING statement on this register for the fixed area DSECT name.

To access the first entry in the variable area:

- Load a second register with the address of the fixed area (this step may be omitted if no further reference is to be made to the fixed area).
- Add to this register the halfword length of the fixed area (contained in the fixed area).
- Issue a USING statement on this register for the variable area DSECT name.

To increment to succeeding entries in the variable area:

- Add to the variable-area base register the (halfword) length of the current entry. This length field may occur either in the fixed or the variable area depending upon the individual table.

The "end" and "size" fields which appear in most table DSECTs should be utilized only by the functions creating the tables, which calculate the halfword length fields subsequent functions should use for incrementation. By adherence to these conventions, expansion of a given table requires reassembly only of the function creating it and those functions addressing the added area. All other functions using the table are unaware of the expanded area but dynamically correct incrementation to the next element.

ABACKR

Functional Description

The ABACKR macro-instruction is used to reposition a multiple record data set immediately in front of the previous record or end-of-file indicator.

```
Name        Operation     Operands

[symbol]    ABACKR          FDB={(R1)|(reg)|addr}
                           ,AREA={(R0)|(reg)|addr}
                           ,EOD={(reg)|addr}
                          [,SAVE={YES|NO}]
                          [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set to be repositioned.

AREA — The address of a two-fullword work area to be used by the ABACKR routine.

EOD — The location to which the ABACKR routine returns if end-of-data has been reached.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the ABACKR routine returns after the data set has been repositioned.

ABLOCK

Functional Description

The ABLOCK macro-instruction is used to block a logical record into a data set being written. This macro may be issued only after an ALOCATE macro.

```
Name        Operation     Operands

[symbol]    ABLOCK          FDB={(R1)|(reg)|addr}
                           ,COUNT={(R0)|(reg)|n}
                          [,SAVE={YES|NO}]
                          [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being blocked.

COUNT — The number of bytes to be blocked. If the number is located in a register, it is represented in binary. If the number is

specified absolutely, it is represented as a decimal number. The number is in the range from 0 to buffer size minus 24.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the BLOCK routine returns when the record has been blocked into the data set.

ABLOCKS

Functional Description

The ABLOCKS macro-instruction is used to block a logical record that may span a buffer boundary. This macro may be issued only after an ALOCATES macro-instruction.

```
Name          Operation       Operands

[symbol]      ABLOCKS           FDB={(R1)|(reg)|addr}
                               ,COUNT={(R0)|(reg)|n}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being blocked.

COUNT — The number of bytes to be blocked. If the number is located in a register, it is represented in binary. If the number is specified absolutely, it is represented in decimal. The number is in the range from 0 to buffer size minus 24.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the BLOCKS routine returns when the record has been blocked into the data set.

ABNCODE

Functional Description

The ABNCODE macro-instruction is used by ASPABEND format routines to control the CSECT name of a module. The module entry code uses ABNCODE to set a global symbol to the CSECT name. The format routines then use ABNCODE to pick up the CSECT name from the global symbol.

```
Name          Operation       Operands

[symbol]      ABNCODE         [CODE={ENTRY|EXIT}]
```

Operands

Code — CODE=ENTRY causes generation of a CSECT statement and initialization code for the format module named by "symbol". The symbol is required in this form but is invalid for the other forms of CODE. CODE=ENTRY is valid only once per module. CODE=EXIT causes generation of termination code for a format module. If CODE is omitted, a CSECT statement is generated using the symbol generated with the CODE=ENTRY form; this usage must be the first statement in each format routine within the module.

ABNCVDEC

Functional Description

The ABNCVDEC macro-instruction, the use of which is restricted to ASPABEND, converts a fullword of hexadecimal data to ten decimal digits.

```
Name          Operation       Operands

[symbol]      ABNCVDEC         [DATA={(R1)|(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
```

Operands

DATA — The location of the data to be converted.

NORMAL — The location to which the ABNCVDEC routine returns when the data has been converted. Register 1 contains the address of the ten-digit decimal field.

ABNCVHEX

Functional Description

The ABNCVHEX macro-instruction, the use of which is restricted to ASPABEND, converts four bytes of data to eight bytes of printable hexadecimal notation.

```
Name          Operation       Operands

[symbol]      ABNCVHEX         [DATA={(R1)|(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
```

Operands

DATA — The location of the data to be converted.

NORMAL — The location to which the ABNCVHEX routine returns when the data has been converted. Register 1 contains the address of the eight-byte printable hexadecimal field.

ABNDSECT

Functional Description

The ABNDSECT macro-instruction is used to establish a DSECT that defines entries in a work area used by ASPABEND.

```
Name          Operation       Operands

              ABNDSECT
```

Operands

This macro-instruction contains no operands.

ABNGET

Functional Description

The ABNGET macro-instruction, the use of which is restricted to ASPABEND, obtains the core storage address of a specified area.

```
Name          Operation       Operands

[symbol]      ABNGET           AREA={(R1)|(reg)|addr}
                               [,NORMAL={(reg)|addr}]
```

Operands

AREA — The address of the field to be obtained.

NORMAL — The location to which ABNGET returns when the area has been obtained. Register 1 contains the core storage address of the area. The field has the length of the alignment attribute of the specified area; e.g., double-word alignment obtains 8 bytes of data, half-word alignment obtains 2 bytes.

ABNPUT

Functional Description

The ABNPUT macro-instruction, the use of which is restricted to ASPABEND, causes a line to be printed on the output data set.

```
Name        Operation      Operands

[symbol]    ABNPUT           PCW={(R1)|(reg)|addr}
                            [,NORMAL={(reg)|addr}]
```

Operands

PCW — The print control word for the line to be printed (see PCW macro-instruction).

NORMAL — The location to which the ABNPUT routine returns when the line has been printed.

ABNVRFY

Functional Description

The ABNVRFY macro-instruction, the use of which is restricted to ASPABEND, verifies the validity and fullword alignment of a specified address.

```
Name        Operation      Operands

[symbol]    ABNVRFY          ERROR={(reg)|addr}
                            [,DATA={(R1)|(reg)|addr}]
                            [,NORMAL={(reg)|addr}]
```

Operands

ERROR — The location to which the ABNVRFY routine returns if the address is invalid or not on a fullword boundary.

DATA — The address to be verified.

NORMAL — The location to which the ABNVRFY routine returns if the address is acceptable.

ACALL

Functional Description

The ACALL macro-instruction allows the user to enter any routine through the ASAVE linkage routine, assuring the integrity of registers 10 through 13 or registers 2 through 13.

```
Name        Operation      Operands

[symbol]    ACALL            ENTER={(R15)|(reg)|addr}
                            [,SAVE={YES|NO}]
                            [,EOD={(reg)|addr}]
                            [,EOF={(reg)|addr}]
                            [,NAVAIL={(reg)|addr}]
                            [,ERROR={(reg)|addr}]
                            [,IPL={(reg)|addr}]
                            [,BUSY={(reg)|addr}]
                            [,REJECT={(reg)|addr}]
                            [,RJPCAN={(reg)|addr}]
                            [,NORMAL={(reg)|addr}]
```

Operands

ENTER — The address of the routine to be entered.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

EOD, EOF, NAVAIL, ERROR, IPL, BUSY, REJECT, RJPCAN, NORMAL — Return points. Only those used by the called routine may be specified, as required by the called routine.

ACCARD

Functional Description

The ACCARD macro-instruction defines the fields of the accounting cards created by the accounting routine within the PURGE DSP. These cards can be processed later using the Accounting Print DSP.

```
Name        Operation      Operands

            ACCARD
```

Operands

This macro-instruction contains no operands.

ACDSECT

Functional Description

The ACDSECT macro-instruction is used to describe the data area used by the ACDS (ASP-Created Data Sets) DSP.

```
Name        Operation       Operands

            ACDSECT         [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

ACENTRY

Functional Description

The ACENTRY macro-instruction is used to define a DSECT for the fields of a //*FORMAT AC parameter buffer

```
Name        Operation       Operands

            ACENTRY
```

Operands

This macro-instruction contains no operands.

ACLOSE

Functional Description

The ACLOSE macro-instruction is used to close a multiple-record data set.

```
Name        Operation       Operands

[symbol]    ACLOSE            FDB={(R1)|(reg)|addr}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

FDB — The address of the File Description Block (FDB) for the data set being closed.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the CLOSE routine returns when the data set has been closed.

ADEBLOCK

Functional Description

The ADEBLOCK macro-instruction deblocks a logical record from a data set being read. When the logical record spans a buffer boundary, the ADEBLOCK routine combines the record into a contiguous area.

```
Name          Operation       Operands

[symbol]      ADEBLOCK          FDB={(R1)|(reg)|addr}
                               ,EOF={(reg)|addr}
                               ,EOD={(reg)|addr}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being deblocked.

EOF — The location to which the DEBLOCK routine returns when an end-of-file has been reached.

EOD — The location to which the DEBLOCK routine returns when an end-of-data has been reached.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the DEBLOCK routine returns when the record has been deblocked. Register 1 contains the address of the record, and register 0 contains the number of bytes in the record represented as a binary number.

ADEBS

Functional Description

The ADEBS macro-instruction deblocks a logical record from a data set being read. When that logical record spans a buffer boundary, a pointer to each segment of the record will be provided.

```
Name          Operation       Operands

[symbol]      ADEBS             FDB={(R1)|(reg)|addr}
                               ,EOD={(reg)|addr}
                               ,EOF={(reg)|addr}
                               ,NAVAIL={(REG)|ADDR}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being deblocked.

EOD — The location to which the ADEBS routine returns when end-of-data has been reached.

EOF — The location to which the ADEBS routine returns when end-of-file has been reached.

NAVAIL — Next buffer not available. This implies end of current buffer. No pointers to data are returned.

SAVE — Indicates whether the contents of registers 4 through 9 are to be saved across the macro call. See Note below.

NORMAL — The location to which the ADEBS routine returns when the record has been deblocked.

Note: Upon normal return, if the record is not split between two buffers, register 1 contains the address of the record; register 0 contains the number of bytes in the record, represented as a binary number. If the record just deblocked fits perfectly in the buffer or if there is no more data in the current buffer, register 2 will be -1. In either event, registers 0 and 1 give the complete count and the starting location of the data. However, if the record is split between two buffers, register 2 contains the number of bytes in the second buffer; and register 3 contains the location of those bytes. The NAVAIL return for print service simulates a perfect fit condition without passing a data record.

ADELETE

Functional Description

The ADELETE macro-instruction is used to delete a module from core storage.

```
Name          Operation       Operands

[symbol]      ADELETE           {EPLOC={(R0)|(reg)|addr}|EP=name}
                               [,{CSECT=NO|CSECT=YES,CDE={(R1)|(reg)|addr}]
                                [,SAVE={YES|NO}]
                                [,NORMAL={(reg)|addr}]
```

Operands

EPLOC — A pointer to the name of the module to be deleted. If register notation is used, the specified register contains the address of the name, left-justified, in an eight-character field padded with blanks. If the module has not been ALOADed previously, the ADELETE request is ignored.

EP — The name of the module to be deleted.

CSECT — CSECT=YES indicates that the ADELETE is being issued for a data control section. This option is normally used only by the Job Segment Scheduler and DSP Failsoft.

CDE — The address of the Contents Directory Element for the CSECT to be deleted, as returned by ALOAD.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the ADELETE routine returns when the module has been deleted.

ADEQ

Functional Description

The ADEQ macro-instruction releases a system resource (JCT, RESQUEUE, etc.) for use by other functions.

```
Name          Operation       Operands

[symbol]      ADEQ             {NAME|FCT}={(R1)|(reg)|addr}
                               ,PRTY={(R0)|(reg)|n}
                              [,NORMAL={(reg)|addr}]
```

Operands

NAME — The name of the function to be released. Refer to the RESOURCE macro expansion in the TVTABLE for the valid names.

FCT — Used by JSS and DSP Failsoft to release all resources enqueued on the specified Function Control Table.

PRTY — A number that specifies the priority associated with the resource entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

NORMAL — The location to which the ADEQ routine returns when the specified resource has been released.

AENQ

Functional Description

The AENQ macro-instruction obtains exclusive use of a system resource (JCT, RESQUEUE, etc.).

```
Name        Operation      Operands

[symbol]    AENQ             NAME={(R1)|(reg)|addr}
                            ,PRTY={(R0)|(reg)|n}
                           [,BUSY={WAIT|(reg)|addr}]
                           [,NORMAL={(reg)|addr}]
```

Operands

NAME — The name of the resource to be obtained. Refer to the RESOURCE macro expansion in the TVTABLE for the valid names.

PRTY — A number that specifies the priority associated with the resource entry in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

BUSY — The location to which the AENQ routine returns if the resource is in use by another function. If WAIT is specified, the AENQ routine AWAITs until the resource becomes available, then takes the NORMAL exit. If WAIT is not coded, register 0 upon return to the caller will contain a mask and address that may be used in an AWAITOFF to wait for the resource to become available.

NORMAL — The location to which the AENQ routine returns when the resource has been obtained.

AFSDSECT

Functional Description

The AFSDSECT macro-instruction is used to define a general data area used by ASP Failsoft.

Name	Operation	Operands
	AFSDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

AGETBUF

Functional Description

The AGETBUF macro-instruction is used to get a buffer from the buffer pool.

Name	Operation	Operands
[symbol]	AGETBUF	[SAVE={YES|NO}] [,NORMAL={(reg)|addr}]

Operands

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the GETBUF routine returns when a buffer is available. The buffer address is located in register 0.

AGETMAIN

Functional Description

The AGETMAIN macro-instruction is used to get a contiguous area of core storage.

```
Name        Operation     Operands

[symbol]    AGETMAIN        SIZE={(R0)|(reg)|addr}
                           ,BUSY={(reg)|addr}
                           [,SP={(reg)|n}]
                           [,HIARCHY={(R1)|(reg)|0|1}]
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

SIZE — The number of bytes of core storage desired.

BUSY — The location to which the GETMAINX routine returns if there is insufficient core to satisfy the user's request. Register 0 contains an ECF mask and address which the calling program may use to AWAIT for available core storage.

SP — The number of the subpool from which to obtain storage, where n is 0 to 255. If SP is omitted, the value specified for ASPOOL on the ASPCORE initialization card is used.

HIARCHY — The number of the hierarchy (0 or 1) from which storage is to be allocated. If register notation is used, storage will be allocated either from hierarchy 0 if the register contains zero or from hierarchy 1 if the register is positive. If the register is negative or if the HIARCHY keyword is omitted, storage will be allocated based upon the HIARCHY parameter on the STANDARDS control card.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the GETMAINX routine returns when it is able to satisfy the user's request. Register 0 and register 1 both contain a pointer to the start of the obtained core storage.

AGETPUTM

Functional Description

AGETPUTM is an inner macro which services the common parameters of AGETMAIN and APUTMAIN .

AIOPARMS

Functional Description

The AIOPARMS macro-instruction establishes a DSECT, used by the Disk Input/Output Routines (ASPIO), that defines a table of I/O parameters.

Name	Operation	Operands
	AIOPARMS	

Operands

This macro-instruction contains no operands.

AJDENTRY

Functional Description

The AJDENTRY macro instruction is used to define fields of an entry in the Active JDAB Table.

Name	Operation	Operands
	AJDENTRY	

Operands

This macro-instruction contains no operands.

ALOAD

Functional Description

The ALOAD macro-instruction is used to load a module into core storage. If the module requested has already been ALOADed, an OS/360 LOAD will still be issued to increment the use count for the module. It is the responsibility of the programmer to ensure that modules that may be called simultaneously by different programs are reentrant.

```
Name          Operation       Operands

[symbol]      ALOAD            {EPLOC={(R0)|(reg)|addr}|EP=name}
                               [,CSECT={YES|NO}]
                               [,SAVE={YES|NO}]
                               [,ERROR={(reg)|addr}]
                               [,BUSY={(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
```

Operands

EPLOC — A pointer to the name of the module to be loaded. If register notation is used, the specified register contains the address of an eight-character field. This field contains the left-justified name of the module to be loaded and is filled with blanks.

EP — The name of the module to be loaded.

CSECT — CSECT=YES indicates that the ALOAD is being issued for a data control section. This option is normally used only by the Job Segment Scheduler and DSP Failsoft.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

ERROR — The location to which the ALOAD routine returns if it cannot load the module due to error. Register 15 contains the error code:

```
04  Module does not exist
08  Permanent I/O error in Directory
```

If this parameter is omitted, the above conditions will result in a DM80 for a non-existent module or DM81 for an I/O error.

BUSY — The location to which the ALOAD routine returns if there is insufficient core to satisfy the user's request. Register 0 contains an ECF mask and address which the calling program may use to AWAIT for available core storage. If this parameter is omitted, ALOAD performs an AWAIT until sufficient core becomes available for loading, then takes the NORMAL exit.

NORMAL — The location to which the ALOAD routine returns when the module has been loaded. Register 0 contains the entry point address of the module. If CSECT=YES was specified, register 1 contains the address of the Contents Directory Element for the module.

ALOCATE

Functional Description

The ALOCATE macro-instruction is used to locate space for a logical record in a data set being written. The ALOCATE macro is used in conjunction with the ABLOCK macro. There must be a corresponding ABLOCK macro after every ALOCATE macro for a given data set.

```
Name        Operation     Operands

[symbol]    ALOCATE         FDB={(R1)|(reg)|addr}
                           ,COUNT={(R0)|(reg)|n}
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set in which space is being located.

COUNT — The number of bytes in the record for which space is being located. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number. The number is in the range from 1 to buffer size minus 24.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the LOCATE routine returns when space has been located. Register 1 contains the address of the location for the data.

ALOCATES

Functional Description

The ALOCATES macro-instruction is used to locate space for a logical record that may span a buffer boundary.

```
Name        Operation     Operands

[symbol]    ALOCATES        FDB={(R1)|(reg)|addr}
                           ,COUNT={(R0)|(reg)|n}
                           ,AREA={(R2)|(reg)|addr}
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

FDB — The address of the File Description Block for the data set in which space is being located.

COUNT — The number of bytes in the record for which space is being located. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number. The number is in the range from 1 to buffer size minus 24.

AREA — A four-fullword area to be used by the called routine to return information.

SAVE — Indicates whether the contents of registers 3 through 9 are to be saved across the macro call.

NORMAL — The location to which the LOCATE routine returns when space has been located. The following information is returned in the location designated in the AREA parameter:

1st word	First data area
2nd word	First count
3rd word	Second data area
4th word	Second count

If the ALOCATES results in only one data area being obtained, the 3rd and 4th words will be zero.

ALTHMSG

Functional Description

The ALTHMSG macro-instruction is used to establish the ASP logical track header message format for each ALTH message as either a dummy or defined work area control section.

Name	Operation	Operands
	ALTHMSG	[TYPE={DSECT|NULL}] [,MSG={ALL|OPR|001}]

Operands

TYPE — DSECT establishes a dummy control section data area. NULL establishes a defined control section data area which can be a part of another real control section.

MSG — MSG=OPR creates the operator ALTH message with the label ALTH0 (if TYPE=NULL is specified, ALTH0 is in ASP message format). MSG=001 creates the ALTH001 message with the label ALTH1 (if TYPE=NULL is specified, ALTH1 is in OS WTO message format). MSG=ALL creates both messages.

ANOTE

Functional Description

The ANOTE macro-instruction is used to note the location of a data set for possible repositioning of the data set to this location at another time.

```
Name          Operation       Operands

[symbol]      ANOTE             FDB={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB — The address of the File Description Block for the data set being noted.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the NOTE routine returns when the note operation has been completed. Register 1 contains a pointer to a doubleword field containing the checkpoint information.

AOPEN

Functional Description

The AOPEN macro-instruction is used to open a multiple record data set for subsequent blocking or deblocking of data.

```
Name          Operation       Operands

[symbol]      AOPEN             FDB={(R1)|(reg)|addr}
                               ,PRTY=n
                               ,TYPE={IN|OUT}
                               [,TATPTR={(R0)|(reg)|addr}]
                               [,BUFDED={NO|1|3}]
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being opened.

PRTY — A decimal number in the range from 0 to 15. Specifies the priority in which ASPIO will service the requests for this data set. with 15 as the highest priority.

TYPE — Specifies whether the data set is to be opened for input or output.

TATPTR The address of the Track Allocator Table FDB for this AOPEN. If the TATPTR is not specified, the JBTAT from the callers AJDB will be used.

BUFDED In conjunction with TYPE=IN, BUFDED=1|3 provides one or three dedicated buffers for the caller, which will be dedicated to the file until it is closed. BUFDED=NO specifies no buffers are to be dedicated.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the OPEN routine returns after the data set has been opened.

AOPEND

Functional Description

The AOPEND macro-instruction is used to open a multiple record output data set so that more data may be blocked into the data set.

```
Name          Operation       Operands

[symbol]      AOPEND            FDB={(R1)|(reg)|addr}
                               [,TATPTR={(R0)|(reg)|addr}]
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB The File Description Block address for the data set to be opened.

TATPTR The address of the Track Allocator Table FDB for this AOPEND. If omitted, the JBTAT from the callers AJDB will be used.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the OPEND routine returns after the data set has been opened.

APOINT

Functional Description

The APOINT macro-instruction is used to reposition a data set to the location that the user noted with an ANOTE macro-instruction. This macro must not reference an open file.

```
Name          Operation       Operands

[symbol]      APOINT            FDB={(R1)|(reg)|addr}
                               ,POINTER={(R0)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
                               [,BUFDED={NO|1|3}]
```

Operands

FDB — The File Description Block address for the data set to be repositioned.

POINTER — A pointer to a three-word field that contains the information saved from the ANOTE in the first two words. The third word should be set to zeros if the file being opened via APOINT is to become an input file, or contain a pointer to the JBTAT FDB (TATPTR) if it is to become an output file. This information is used to restart processing at a specific point within the data set.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the POINT routine returns when the data set is repositioned to the requested location.

BUFDED — BUFDED=1|3 provides one or two dedicated buffers for the caller, which will be dedicated to the file until it is closed. BUFDED=NO specifies no buffers are to be dedicated.

APURGE

Functional Description

The APURGE macro-instruction is used to purge a single record file out of STT or job TAT FDB.

```
Name          Operation       Operands

[symbol]      APURGE            FDB={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address of the data set being purged.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the PURGE routine returns when the data set has been purged.

APUTBUF

Functional Description

The APUTBUF macro-instruction is used to return a buffer to the buffer pool.

```
Name          Operation       Operands

[symbol]      APUTBUF           BUFFER={(R0)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

BUFFER — The address of the buffer being returned to the buffer pool.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the PUTBUF routine returns when the buffer has been returned.

APUTMAIN

Functional Description

The APUTMAIN macro-instruction is used to return a contiguous area of core storage, obtained by a previous AGETMAIN request, to the pool of available core storage.

```
Name          Operation       Operands

[symbol]      APUTMAIN          SIZE={(R0)|(reg)|addr}
                               ,AREA={(R1)|(reg)|addr}
                               [,SP={(reg)|0|n}]
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

SIZE The number of bytes of core storage to be returned.

AREA A pointer to the start of the contiguous storage area to be returned.

SP The subpool in which the area resides, where n is 0 to 255.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the PUTMAINX routine returns when it is able to satisfy the user's request.

AREAD

Functional Description

The AREAD macro-instruction is used to read a single-record data set from a direct access storage device.

```
Name        Operation      Operands

[symbol]    AREAD            FDB={(R1)|(reg)|addr}
                             [,SAVE={YES|NO}]
                             [,NORMAL={(reg)|addr}]
```

Operands

FDB The File Description Block address of the data set to be read.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the READ/WRITE routine returns when the read request has been satisfied. When the data set is read, the first four bytes of the FDB contain the buffer address of the data set.

ARELEASE

Functional Description

The ARELEASE macro-instruction is used to restore the track address in the File Description Block for a single-record data set to allow two successive AREADs of the data set.

```
Name          Operation       Operands

[symbol]      ARELEASE          FDB={(R1)|(reg)|addr}
                                [,SAVE={YES|NO}]
                                [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address of the single-record data set.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the RELEASE routine returns when the track address in the FDB has been restored.

ARETURN

Functional Description

The ARETURN macro-instruction generates a return sequence through a SAVE-supported call (i.e., ACALL or any macro-instruction supporting the SAVE keyword).

```
Name          Operation       Operands

[symbol]      ARETURN         [RC={(reg)|0|n}]
```

Operands

RC — The return code, representing the displacement from the return point for the desired macro exit. The contents of the register, or n, must be 0 or a multiple of 4. Any register except 0, 11, 12, or 14 may be specified.

ASGDSECT

Functional Description

The ASGDSECT macro-instruction establishes a DSECT which defines entries in the Assignment Table (see ASP Initialization card "RESCTBLK"). This table is used by the MODIFY verb (see ASP Console Operator's Manual).

Name	Operation	Operands
	ASGDSECT	

Operands

This macro-instruction contains no operands.

ASPCKPNT

Functional Description

The ASPCKPNT macro-instruction causes a checkpoint to be written.

Name	Operation	Operands
[symbol]	ASPCKPNT	[SAVE={YES|NO}] [,NORMAL={(reg)|addr}]

Operands

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the checkpoint routine returns after the checkpoint has been written.

ASPCLOSE

Functional Description

The ASPCLOSE macro-instruction is used to close a unit record or tape device on the Support Processor. Writing of tapemarks and/or tape positioning is not provided.

```
Name          Operation       Operands

[symbol]      ASPCLOSE          DCB={(R1)|(reg)|addr}
                               [,NORMAL={(reg)|addr}]
```

Operands

DCB — The address of the Data Control Block associated with the device to be closed.

NORMAL — The location to which the ASPCLOSE routine returns when the device has been closed.

ASPDCB

Functional Description

The ASPDCB macro-instruction generates a Data Control Block, overlaid with a Data Extent Block, in a special format used by ASPOPEN and ASPCLOSE.

```
Name          Operation       Operands

symbol        ASPDCB            DEB=label
                               ,DEVD={UR|TA}
                               [,DEN={0|1|2|3}]
                               [,TRTCH={C|E|T|ET}]
```

Operands

DEB — The label to be assigned to the starting location of the Data Extent Block.

DEVD — The type of Data Control Block to be generated, unit record or tape.

DEN — Tape density indicator. 0=200 bpi, 1=556 bpi, 2=800 bpi, 3=1600 bpi. This parameter is required if DEVD=TA. If the density is not known until execution of the DSP, it must be OR'ed with the TRTCH in the 33rd byte of the DEB prior to the issuance of ASPOPEN.

TRTCH Tape recording technique. The options are:

	Parity	Data Converter	Translator
C	odd	on	off
E	even	off	off
T	odd	off	on
ET	even	off	on
Operand omitted	odd	off	off

If the TRTCH is not known until execution, it must be OR'ed with the density in the 33rd byte of the DEB prior to issuance of ASPOPEN.

ASPDUMPS

Functional Description

The ASPDUMPS macro-instruction is furnished to provide user completion codes used by ASP for ABEND conditions. It consists entirely of EQU statements. The following statements are typical of those generated:

```
ASPDM001   EQU     1  TERMINAL ERRORS IN INITIALIZATION DECK

ASPDM002   EQU     2  NOT ENOUGH CORE TO INITIALIZE ASP

ASPDM008   EQU     8  OPERATOR INITIATED VIA *DUMP

ASPDM040   EQU    40  ILLEGAL JCT ACCESS

ASPDM080   EQU    80  ALDADEL- CANNOT LOAD MODULE - DOES NOT EXIST
```

```
   Name        Operation       Operands

               ASPDUMPS
```

Operands

This macro-instruction contains no operands.

ASPEOV

Functional Description

The ASPEOV macro-instruction operates in conjunction with Remote Job Processing (RJP). It is issued by various DSP's after end-of-file has been detected by ASPEXCP and triggers transmission of the remaining records blocked in the transmission buffer. If ASPEOV is not executed, this function is performed by ASPCLOSE. If RJP is not active for the DSP, ASPEOV is essentially a NOP.

```
Name          Operation       Operands

[symbol]      ASPEOV            IOB={(R1)|(reg)|addr}
                               ,RJPCAN={(reg)|addr}
                              [,NORMAL={(reg)|addr}]
```

Operands

IOB — The label, or a register containing the address, of the Input/Output Block (IOB) associated with the request.

RJPCAN — The location to which the ASPEOV routine returns if a cancel command is issued from the remote workstation.

NORMAL — The location to which the ASPEOV routine returns when it has normally completed its functions.

ASPEXCP

Functional Description

The ASPEXCP macro-instruction is used to request input/output for a specific device.

```
Name          Operation       Operands

[symbol]      ASPEXCP           IOB={(R1)|(reg)|addr}
                               ,RJPCAN={(reg)|addr}
                              [,NORMAL={(reg)|addr}]
```

Operands

IOB — The label, or a register containing the address, of the Input/Output Block (IOB) associated with the input/output request.

RJPCAN — The location to which the ASPEXCP routine returns if a cancel command is issued from a remote workstation.

NORMAL — The location to which the ASPEXCP routine returns when it has normally completed its functions.

ASPHDR

Functional Description

The ASPHDR macro-instruction establishes a data area, used by Print and Punch Services, that defines the fields and format of the output data set header record.

```
Name        Operation       Operands

            ASPHDR          [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

ASPHIO

Functional Description

The ASPHIO macro-instruction is used by RJP and Console Service to terminate I/O and dequeue the Request Element.

```
Name        Operation       Operands

[symbol]    ASPHIO            UCB={(R1)|(reg)|addr}
                             ,ERROR={(reg)|addr}
                             ,REJECT={(reg)|addr}
                            [,SAVE={YES|NO}]
                            [,NORMAL={(reg)|addr}]
```

Operands

UCB — The address of the Unit Control Block of the device to be halted.

ERROR — The location to which the ASPHIO routine returns if an error was encountered. Register 1 contains one of these codes:

1 Device not operational
2 Invalid control block
3 Catastrophic channel error

REJECT — The location to which the ASPHIO routine returns if the specified device is not active.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the ASPHIO routine returns after completing its functions.

ASPMTCIB

Functional Description

The ASPMTCIB macro-instruction is used to establish a dummy work area control section which defines the OS Communications Input Buffer used by ASP MAINTASK.

```
Name        Operation       Operands

            ASPMTCIB
```

Operands

This macro-instruction contains no operands.

ASPOPEN

Functional Description

The ASPOPEN macro-instruction is used to open a unit record or tape device on the Support Processor.

```
Name        Operation       Operands

[symbol]    ASPOPEN           DCB={(R1)|(reg)|addr}
                             ,IOB={(R0)|(reg)|addr}
                             ,SUPUNIT=(reg)
                            [,NORMAL={(reg)|addr}]
```

Operands

DCB	The address of the Data Control Block associated with the device to be opened. The DCB must be in the format generated by the ASPDCB macro-instruction.
IOB	The address of the Input/Output Block associated with the device to be opened.
SUPUNIT	The address of the Support Units Table entry for the device to be opened. The address is found in the GETUNIT List entry for the device.
NORMAL	The location to which the ASPOPEN routine returns when the device has been opened.

ASPSNAP

Functional Description

The ASPSNAP macro-instruction is used as a debugging tool during development of a user-written module to call for OS snaps and/or ASPABEND execution. It is used in conjunction with the Dump Core (DC) callable DSP, which enables the ASPSNAP facilities.

```
Name        Operation     Operands

[symbol]    ASPSNAP        [DUMP={NO|YES|OS|ASP|SA|STD}]
                           [,FORMAT={YES|NO}]
                           [,IDD={00|n}]
                           [,SD=sdata]
                           [,PD=pdata]
                           [,LST=addr]
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

DUMP — Specifies the type of dump to be taken:

- NO — No core dump
- YES — Dump as identified by IDD, SD, PD, and LST
- OS — SYSABEND dump
- ASP — UDUMP
- SA — Standalone dump
- STD — As stated on Initialization OPTIONS card

FORMAT — Indicates whether tables are to be formatted.

IDD — This operand, applicable if DUMP=YES, provides the OS SNAP ID code.

SD — The values for this operand are identical to the operands for SDATA= as documented in the OS macros for the OS SNAP facility.

PD — The values for this operand are identical to the operands for PDATA= as documented in the OS macros for the OS SNAP facility.

LST — The values for this operand, the location symbol of a list of fullwords, are identical to the LST= parameter of the OS SNAP macro.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the ASPSNAP routine returns when the snap has been taken.

Note: Execution of the ASPSNAP macro will result in only a BR R14 until the ASPSNAP facility is implemented by calling the debugging DSP Dump Core (DC). DC contains the code that interfaces to OS SNAP and ASPABEND. When called, it changes the BR R14 pointer in the ASP TVTABLE to the portion of the DSP that causes the ASPSNAP request to be honored. When DC is canceled, the facility is discontinued. Thus, the output of the ASPSNAP calls can be controlled in a debugging environment.

ATEST

Functional Description

The ATEST macro-instruction determines the availability of a system resource.

```
Name          Operation       Operands

[symbol]      ATEST             NAME={(R1)|(reg)|addr}
                              ,{TYPE=FCT|[TYPE=TEST,]ERROR={(reg)|addr}}
                              [,NORMAL={(reg)|addr}]
```

Operands

NAME — The name of the resource to be tested. Refer to the RESOURCE macro expansion in the TVTABLE for the valid names.

TYPE — If TYPE=FCT is specified, the address of the Function Control Table using the resource is returned to the caller. If TYPE=TEST is specified, control is returned to the normal return point if the issuing function has the resource, to the error return point if not.

ERROR — In conjunction with TYPE=TEST, the location to which ATEST returns if the caller does not have the resource.

NORMAL — In conjunction with TYPE=TEST, the location to which ATEST returns if the caller has the resource. With TYPE=FCT, the only return point; register 1 contains the address of the FCT using the resource or zeros if the resource is not in use.

ATIME

Functional Description

The ATIME macro-instruction is used to request an asynchronous entry to a timer appendage after a specified time interval has elapsed, or to cancel a previous request for such an entry. The ATIME macro may not be used out of an ATIME exit.

```
Name          Operation       Operands

[symbol]      ATIME             TIME={CANCEL|(R0)|(reg)|n}
                               ,ENTER={(R1)|(reg)|addr}
                              [,AREA={(reg)|addr}]
                              [,ID={'name'|(reg)|addr}]
                              [,FCT={(reg)|addr}]
                              [,SAVE={YES|NO}]
                              [,BUSY={(reg)|addr}]
                              [,NORMAL={(reg)|addr}]
```

Operands

TIME — CANCEL indicates that a previously requested time-interval entry is to be canceled. Any other specification represents the time interval to elapse prior to an asynchronous entry to the appendage, specified in hundredths of seconds. An interval of zero is equivalent to "cancel".

ENTER — The address of an appendage which is to be entered after the specified interval has elapsed.

AREA — The address of a four-fullword area to be used by ATIME as the timer queue element. If omitted, this area is generated by ATIME.

ID — The name to be associated with the ATIME queue element. The ID length is limited to five characters.

FCT — The FCT for which this ATIME is to apply. If omitted, the current FCT is assumed.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

BUSY — The location to which the ATIME routine returns control if it is unable to obtain core for a timer queue element. This parameter is required if AREA is not specified.

NORMAL — The location to which the ATIME routine returns when the timer entry has been placed on the queue or canceled.

In addition to the above form, ATIME may be issued in execute and list form.

The execute form consists of the parameters noted above (except that TIME and ENTER are optional) plus the parameter MF=(E,{(reg)|addr}) in which the second parameter provides the address of the list-form expansion of the macro.

The list form consists of the optional parameters TIME, ENTER, AREA, and ID (none of which may use register notation in this form) plus the parameter MF=L. The parameter AREA represents a four-fullword timer queue element to be generated.

The timer appendage routine may use either of two return points. They have the following meanings:

- R14+0 - The time element is to be removed from the ASP timer queue.
- R14+4 - The timer element is to be reused. The ASP timer queue element is left on the queue in an inactive status.

ATRACK

Functional Description

The ATRACK macro-instruction is used to obtain a track address for use in writing data or single-record files to the ASP job queue.

```
Name          Operation       Operands

[symbol]      ATRACK            TATPTR={(R0)|(reg)|addr}
                               ,BUSY={(reg)|addr}
                              [,NORMAL={(reg)|addr}]
```

Operands

TATPTR — The address of the Track Allocation Table FDB (JBTAT) for the file requesting tracks.

BUSY — The location to which the TRACKS routine returns control when no tracks are available in the ASP job queue.

NORMAL — The location to which the TRACKS routine returns control when a track has been assigned.

Register 0 Contains the track address.

ATTRMSG

Functional Description

The ATTRMSG macro-instruction is used to establish the OS job queue allocation message format for each ATTR message as either a dummy or defined work area control section.

```
Name          Operation       Operands

              ATTRMSG          [TYPE={DSECT|NULL}]
                              [,MSG={ALL|001|002|003|004|010|011}]
```

Operands

TYPE — DSECT establishes a dummy control section data area. NULL establishes a defined control section data area which can be a part of another real control section.

MSG — MSG=nn creates the ATTRnnn message with the label ATTRn (the labels for 010 and 011 are ATTRA and ATTRB, respectively). If TYPE=NULL is specified, message ATTR1 and ATTR4 are in ASP message format, the remaining messages in OS WTO format. MSG=ALL creates all of the above messages.

AWAIT

Functional Description

The AWAIT macro-instruction is used by a function to share processing time on the Support Processor. This macro specifies that control is to return to the function issuing the macro only when at least one of the specified events has occurred. An event can be the completion of any other operation. The completion of an event is indicated by setting the appropriate bit in the event completion flag to 1.

More than one AWAIT macro can specify the same bit of an ECF. An ECF bit may be referred to by an AWAIT macro when the bit is 1 or 0. Note that the OS/360 WAIT macro-instruction does not function in this manner.

```
Name        Operation     Operands

[symbol]    AWAIT          {ECFMASK={(R0)|(reg)|X'xx'}
                           ,ECFADD={(R0)|(reg)|addr}|
                            ECFLIST={(R0)|(reg)|addr}}
                          [,SAVE={YES|NO}]
                          [,NORMAL={(reg)|addr}]
```

Operands

ECFMASK — The mask that specifies the bits (events) to be tested in the ECF. The test is satisfied if any or all of the events are complete. If the mask is represented with register notation, the leftmost byte of the register contains the hexadecimal mask. If the mask is represented absolutely in the operand, it is represented as a one-byte hexadecimal character.

ECFADD — The label of the one-byte event completion flag. If register notation is used, the rightmost three bytes of the register contain the address of the ECF.

Note: If the ECF address is represented as the label of the ECF (instead of with register notation), the ECF mask will be represented in the form X'xx'. If the ECF mask is represented by register notation, the ECF address must also be represented by the same register.

ECFLIST — The label of the event completion list. If register notation is used, the register contains the address of the event completion list.

The event completion list must contain from 1 to n fullword entries, positioned on a fullword boundary, of the event completion mask and the event completion address. The first byte of each entry must contain the event completion mask, and the remaining three bytes must contain the event completion address. The list must be terminated with a fullword entry containing binary 1's.

Note: If the ECFLIST operand is used, the ECFMASK and ECFADD operands must be omitted. Any satisfied event completion encountered in the event completion list will enable the DSP to gain control at the address specified by the NORMAL operand.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which control is returned when one or more of the specified events have been completed.

AWAITOFF

Functional Description

The AWAITOFF macro-instruction is used by a function to share processing time on the Support Processor. The macro specifies that control is to be returned to the function issuing the macro only when all of the specified events have occurred. An event can be the completion of any other operation. The completion of an event is indicated by setting the appropriate bit in the Event Completion Flag (ECF) to 0.

More than one AWAITOFF macro-instruction can specify the same bit of an ECF. An ECF bit may be referred to by an AWAITOFF macro when the bit is 0 or 1.

```
Name          Operation      Operands

[symbol]      AWAITOFF        {ECFMASK={(R0)|(reg)|X'xx'}
                              ,ECFADD={(R0)|(reg)|addr}|
                               ECFLIST={(R0)|(reg)|addr}}
                             [,SAVE={YES|NO}]
                             [,NORMAL={(reg)|addr}]
```

Operands

ECFMASK — The mask that specifies the bits (events) to be tested in the ECF. The test is satisfied if all of the events are complete. If the mask is represented with register notation, the leftmost byte of the register contains the hexadecimal mask. If the mask is represented absolutely in the operand, it is represented as a one-byte hexadecimal character.

ECFADD — The label of the one-byte event completion flag. If register notation is used, the rightmost three bytes of the register contain the address of the ECF.

ECFLIST — The label of the event completion list. If register notation is used, the address of the event completion list.

The event completion list must contain from 1 to n fullword entries, positioned on a fullword boundary, of the event completion mask and the event completion address. The first byte of each entry must contain the event completion mask, and the remaining three bytes must contain the event completion address. The list must be terminated with a fullword entry containing binary 1's.

Note: If the ECFLIST operand is used, the ECFMASK and ECFADD operands must be omitted. Any satisfied event completion encountered in the event completion list will enable the DSP to gain control at the address specified by the NORMAL operand.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which control is returned when one or more of the specified events have been completed.

AWEOF

Functional Description

The AWEOF macro-instruction is used to place an end-of-file indicator in a multiple record output data set.

```
Name          Operation       Operands

[symbol]      AWEOF             FDB={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set in use.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the WEOF routine returns after the end-of-file indicator has been placed in the data set.

AWRITE

Functional Description

The AWRITE macro-instruction is used to write a single record data set on a direct access storage device.

```
Name          Operation       Operands

[symbol]      AWRITE            FDB={(R1)|(reg)|addr}
                               [,PUTBUF={YES|NO}]
                               [,TATPTR={(R0)|(reg)|addr}]
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set to be written.

PUTBUF — If PUTBUF=NO is specified, the buffer is not returned. Effectively, use of this operand is equivalent to an AWRITE followed by an AREAD.

TATPTR — The address of the Track Allocator Table FDB for this AWRITE.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the READ/WRITE routine branches when the write request has been satisfied.

CKPTDATA

Functional Description

The CKPTDATA macro-instruction provides a map of the checkpoint area used by ASPCKPNT.

```
Name        Operation       Operands

            CKPTDATA        [TYPE={CSECT|DSECT}]
                            [,CKPT={YES|NO}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

CKPT — CKPT=YES provides a map of the entire data area. CKPT=NO limits expansion to the checkpoint data.

CONBUFCB

Functional Description

The CONBUFCB macro-instruction establishes a DSECT used to define a Console Buffer Control Block.

```
Name        Operation       Operands

            CONBUFCB
```

Operands

This macro-instruction contains no operands.

CONCNVRT

Functional Description

The CONCNVRT macro-instruction converts a console destination class to a displacement and mask suitable for use in the MESSAGE macro.

```
Name        Operation     Operands

[symbol]    CONCNVRT       CLASS={(R0)|(reg)|addr}
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

CLASS — The pointer to the destination class name.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the CONCNVRT routine returns when the class name has been converted. Register 0 contains the displacement and mask in the low-order two bytes. If the result is zero in these two bytes, the class name was invalid.

CONDSECT

Functional Description

The CONDSECT macro-instruction generates various tables used to communicate information between Console Service and functions within the ASP system.

```
Name        Operation     Operands

            CONDSECT       [TYPE={INPUT|OUTPUT|MLOG|FCTQ}]
                           [,CODES={YES|NO}]
```

Operands

TYPE — Defines the tables to be generated. The parameters may also be grouped in any order within parentheses -- e.g., TYPE=(MLOG,OUTPUT) -- to generate multiple tables. The first occurrence of CONDSECT within a control section also causes a set of buffer control equate statements to be generated. The TYPE field is required if CODE=NO is specified. The types of tables are:

INPUT	Format of input message buffer
OUTPUT	Format of console message buffer
MLOG	Format of MLOG message buffer
FCTQ	Format of action message buffer

CODES If TYPE=INPUT is specified or the TYPE parameter is omitted, CODES=YES causes generation of a group of equate statements defining the console input command action codes, e.g., START, CALL, VARY. CODES=NO may be used to suppress generation of these statements with TYPE=INPUT; however, it is invalid if TYPE is omitted.

CONSCONS

Functional Description

The CONSCONS macro-instruction is used to define a general data area used by Console Service.

Name	Operation	Operands
	CONSCONS	[TYPE={CSECT|DSECT}]

Operands

TYPE CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

CONSDATA

Functional Description

The CONSDATA macro-instruction generates a DSECT, used by Console Service, which contains status and control information about each console defined for the system.

Name	Operation	Operands
	CONSDATA	

Operands

This macro-instruction contains no operands.

CONSUNIT

Functional Description

The CONSUNIT macro-instruction generates a DSECT, used by Console Service, that describes the unit control tables for each console. These include the IOB, ECB, DCB, and DEB.

Name	Operation	Operands
	CONSUNIT	

Operands

This macro-instruction contains no operands.

CONTVTBL

Functional Description

The CONTVTBL macro-instruction generates a DSECT, used by Console Service, that defines positional entries in a device-dependent routine.

Name	Operation	Operands
	CONTVTBL	

Operands

This macro-instruction contains no operands.

DELDSECT

Functional Description

The DELDSECT macro-instruction is used to establish either a real or a dummy work area control section used by the DEADLINE DSP.

Name	Operation	Operands
	DELDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

DEQMSG

Functional Description

The DEQMSG macro-instruction is used to dequeue input and/or action messages from the Function Control Table.

```
Name        Operation      Operands

[symbol]    DEQMSG          [TYPE={ACTION|INPUT|CONSOLE|ALL}]
                           [,BUFFER={(reg)|addr}]
                           [,FCT=(reg)]
                           [,CONS=(reg)]
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

TYPE — The type of buffers to be dequeued from the FCT chain. The term CONSOLE is reserved for Console Service to dequeue console buffers from the Console Status Table, and ALL is used by JSS to ensure removal of all queued messages when a DSP has completed processing.

BUFFER — The address of the buffer to be dequeued. If omitted, all buffers of the specified type are dequeued. This parameter may not be used with TYPE=ALL.

FCT — The address of the FCT from which to dequeue the buffers. If omitted, the active FCT is used.

CONS — This parameter is reserved for Console Service and is valid only for TYPE=CONSOLE. It specifies a Console Status Table entry. If TYPE=CONSOLE is specified and CONS is omitted, the entry is assumed to be the currently active entry pointed to by register 9.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the DEQMSG routine returns when its dequeuing functions have been completed.

DEVREQ

Functional Description

The DEVREQ macro-instruction is used in the generation of the Device Requirements Table of RESPARAM. It uses global symbols set by the macro DSPDC which specify required entries for the DSP's defined by DSPDC.

```
Name        Operation      Operands

            DEVREQ
```

Operands

This macro-instruction contains no operands.

DEVSCAN

Functional Description

The DEVSCAN macro-instruction analyzes the operands of the IN and OUT keyword parameters allowed by the callable DSP's and moves the parameter fields into an area designated by the user.

```
Name        Operation      Operands

[symbol]    DEVSCAN          FROM={(R0)|(reg)|addr}
                            ,TO={(R1)|(reg)|addr}
                            ,ERROR={(reg)|addr}
                           [,NORMAL={(reg)|addr}]
```

Operands

FROM — The location at which the scan is to begin; that is, the byte following the equal sign of IN= or OUT=.

TO — The location of a 16-byte field to which the parameters are to be moved.

ERROR — The location to which DEVSCAN returns if the scan cannot be completed. Register 15 contains one of the following error codes:

- 04 Length error
- 08 Delimiter error
- 0C No operand

NORMAL — The location to which the DEVSCAN returns when the fields have been moved. If the parameter was given in parentheses, the first field is in the first 8 bytes of the TO field and the second field is in the second 8 bytes, with binary zeros substituted if either field is omitted. If the parameter was given without parentheses, the first 8 bytes of the TO field are zeros and the second 8 bytes contain the parameter field. Register 1 contains the address of the byte following the delimiter which terminated the scan.

DJCDSECT

Functional Description

The DJCDSECT macro-instruction is used to describe the data area used in Dependent Job Control updating processing.

```
Name        Operation      Operands

            DJCDSECT       [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

DJDSECT

Functional Description

The DJDSECT macro-instruction is used to describe the data area used by Dump Job.

Name	Operation	Operands
	DJDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

DLFQUEUE

Functional Description

The DLFQUEUE macro-instruction establishes a DSECT, used by Deadline Scheduling, that defines entries in the Deadline job queue. The Deadline queue is maintained as a data set on the ASP job queue.

Name	Operation	Operands
	DLFQUEUE	

Operands

This macro-instruction contains no operands.

DLTENTRY

Functional Description

The DLTENTRY macro-instruction establishes a DSECT, used by Deadline Scheduling, that defines entries in the Deadline table. The Deadline table is generated during Initialization from DEADLINE cards.

Name	Operation	Operands
	DLTENTRY	

Operands

This macro-instruction contains no operands.

DSLENTRY

Functional Description

The DSLENTRY macro-instruction defines a save area for Print Service burst page statistics.

```
Name        Operation       Operands

            DSLENTRY
```

Operands

This macro-instruction contains no operands.

DSPDC

Functional Description

The DSPDC macro-instruction is used to generate entries for the Dynamic Support Programs in the Resident Parameter Table. An entry in the table is required for each DSP in order for it to be recognized as being present in the system.

```
Name        Operation       Operands

symbol      DSPDC            [PRTY={1|n}]
                            [,SYNCH={YES|NO}]
                            [,REENT={YES|NO}]
                            [,XABLE={YES|NO}]
                            [,NOREQ={0|n}]
                            [,REQ=(type1,...,typen)]
                            [,CSECT=name]
                            [,MLOAD={YES|NO}]
                            [,GETUNV={C|R}]
                            [,FSNAME=name]
                            [,INIGET={DSP|JSS}]
```

Operands

symbol — The name of the Dynamic Support Program.

PRTY — The priority to be assigned to the DSP, in the range from 1 through 14.

SYNCH — If SYNCH=YES is specified, the DSP is sequence-dependent; that is, it may not be scheduled until all preceding segments of the job are complete, and no succeeding segment of the job may be scheduled until it is complete.

REENT — REENT=YES indicates that the DSP is reentrant.

XABLE — XABLE=YES indicates that the DSP may be called from the console by the operator.

NOREQ — The maximum number of devices required by the DSP. The number of entries in the GETUNIT list associated with the Function Control Table will be equal to this value. Should be specified when known.

REQ — The types of devices known to be required by the DSP. The valid device types are specified on the DEVICE initialization card. Should be specified whenever the DSP unit requirements are known. If more than one device is required, the subparameter list must be enclosed in parentheses. For example, a DSP requiring two seven-track tape devices and a printer could specify NOREQ=3 and REQ=(TA7,TA7,PRT).

CSECT — For reentrant DSP's, the name of the control section to be loaded by the Job Segment Scheduler for each use of the DSP. This name is restricted to seven characters.

MLOAD — If MLOAD=YES is specified, the DSP may be multiply loaded; that is, more than one copy may be active concurrently. This option is limited to single-module reusable programs. If it is used, the DSP name is restricted to seven characters.

GETUNV — Desired disposition of function if devices are not available when GETUNIT is issued. GETUNV=R indicates function is to return for specialized rescheduling, GETUNV=C indicates it is to cancel. Bit DSPRESCH in the DSP Dictionary may be tested during DSP execution to determine which option is desired. All callable DSP's in the distributed ASP system also provide an operator override via keyword parameter to alter the basic disposition for an individual job.

FSNAME — The unique four-character function identifier to be inserted into the ASP Failsoft recovery name (AFSxxxxn).

INIGET — Indicates whether the initial GETUNIT is to be done by the DSP or JSS.

Note: Parameters NOREQ and REQ must be specified if INITGET=JSS is specified, and should always be specified when NOREQ and REQ are known as these values are used in JSS scheduling. In the distributed ASP system the Print, Punch, and RICONTL DSP's specify INIGET=JSS. All other DSP's are scheduled without units and obtain them via GETUNIT macro.

DSPENTRY

Functional Description

The DSPENTRY macro-instruction is used to define the fields of an entry in the Dynamic Support Program (DSP) Dictionary and an entry in the Device Requirements Table.

Name	Operation	Operands
	DSPENTRY	

Operands

This macro-instruction contains no operands.

EFENTRY

Functional Description

The EFENTRY macro-instruction is used to define the fields of the ending function control block used in Job Segment Scheduler (JSS) processing.

Name	Operation	Operands
	EFENTRY	

Operands

This macro-instruction contains no operands.

EQUATE

Functional Description

The EQUATE macro-instruction is furnished to provide standardized symbolic references for condition codes. It consists entirely of EQU statements and is intended to simplify coding and interpretation of the program listings.

The following examples illustrate its usage:

	Without Equate		With Equate	
Compare	BCR	8,7	BCR	EQ,7
Arithmetic	BCR	2,7	BCR	PLUS,7
Test Under Mask	BCR	5,7	BCR	ALLON+MIXED,7

```
   Name          Operation       Operands

                 EQUATE
```

Operands

This macro-instruction contains no operands.

FAILDSP

Functional Description

The FAILDSP macro-instruction allows the user to enter ASP DSP Failsoft upon recognition of an error situation. The ASPDUMPS macro must be issued prior to using the FAILDSP macro.

```
   Name          Operation       Operands

   [symbol]      FAILDSP           CODE={ASPDMnnn|(R1)|(reg)|addr}
                                  [,FCT={(R0)|(reg)|addr}]
                                  [,DUMP={YES|NO|(reg)}]
                                  [,SAVE={YES|NO}]
                                  [,NORMAL={(reg)|addr}]
```

Operands

CODE — The ASPDMnnn failure code identifying the type of failure, or the numeric (binary) equivalent in a register.

FCT — The address of the Function Control Table to be failed with the associated code. If omitted, the current FCT is used.

DUMP — Indicates whether an ASPABEND dump is to be taken before the DSP is failed. (reg) is only valid when FCT=(reg) is used and must specify the same register. If the register is

positive a dump will be taken. If the register is a complement no dump will be taken.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the FAILDSP routine returns. This may be either a location at which the failing situation can be retried or the label on the FAILDSP macro-instruction.

FCTDC

Functional Description

The FCTDC macro-instruction is used to generate the permanent Function Control Table entries in the Resident Parameter Table for the ASP Nucleus portions of Console Service, the ASP Disk Input/Output routines, the Main Device Scheduler, Call DSP, and the Job Segment Scheduler.

```
Name        Operation      Operands

symbol      FCTDC            ECFMASK=xx
                            ,ECFADD=addr
                            ,COND=xx,
                            ,PRTY=xx
                            ,NAME=function-name
                           [,R10=label]
                           [,R13=label]
                           [,R14=label]
                           [,R15=label]
                           [,ANAME=(alternate-function-name1,...,
                             alternate-function-namen)]
                           [,SUCC=addr]
                           [,LOGIN=addr]
                           [,FSNAME-name]
                           [,INISH={YES|NO}]
```

Operands

ECFMASK — The mask that specifies the bits (events) to be tested in the ECF. The test is satisfied if any or all of the events are complete.

ECFADD — The label of the one-byte event completion flag.

COND — The hexadecimal value of the mask field of the Branch on Condition instruction used to test ECFMASK. For example, AWAIT specifies 50 to branch one plus mixed; AWAITOFF specifies 80 to branch zero.

PRTY — Function priority. May be any hexadecimal value, except that 00 and FF are reserved for the Job Segment Scheduler and Console Service, respectively.

NAME — The name by which the operator may communicate with the function.

R10 The external label of an instruction whose address will be in register 10 before the function is entered for the first time.

R13 The external label of an instruction whose address will be in register 13 before the function is entered for the first time.

R14 The external label of an instruction whose address will be in register 14 before the function is entered for the first time. This parameter is required if the FCT is dispatchable.

R15 The external label of an instruction whose address will be in register 15 before the function is entered for the first time.

ANAME The alternate names by which the operator may communicate with the function.

SUCC The label of the FCTDC macro of the succeeding function, which must have an equal or lower priority.

LOGIN The label of the function's console appendage if the function is to be permanently logged in.

FSNAME The unique four-character function identifier to be inserted into the ASP Failsoft recovery name (AFSxxxxn).

INISH NO indicates that the FCT is non-dispatchable until Initialization is complete, YES indicates the FCT is always dispatchable.

FCTDSP

Functional Description

The FCTDSP macro-instruction is used in the generation of the DSP Dictionary of RESPARAM. It uses the global symbols set by the macro FCTDC which specify entries in the DSPDC macro. The DSPDC macro is issued by FCTDSP to generate entries in the DSP Dictionary.

Name	Operation	Operands
	FCTDSP	

Operands

This macro-instruction contains no operands.

FCTENTRY

Functional Description

The FCTENTRY macro-instruction is used to define the fields of an entry in the Function Control Table and an entry in the GETUNIT list.

Name	Operation	Operands
	FCTENTRY	

Operands

This macro-instruction contains no operands.

FDBENTRY

Functional Description

The FDBENTRY macro-instruction is used by the Disk Input/Output Routines (ASPIO) to define the fields in a File Description Block for Single and Multiple Record Data Sets.

Name	Operation	Operands
	FDBENTRY	

Operands

This macro-instruction contains no operands.

FDDSECT

Functional Description

The FDDSECT macro-instruction establishes a DSECT, used by ASPIO, that defines the fields in the File Directory (FD).

Name	Operation	Operands
	FDDSECT	

Operands

This macro-instruction contains no operands.

FINDJNUM

Functional Description

The FINDJNUM macro-instruction determines whether a specific job number is currently in use.

```
Name          Operation       Operands

[symbol]      FINDJNUM          JOBN={(R0)|(reg)|addr}
                               ,ERROR={(reg)|addr}
                              [,NORMAL={(reg)|addr}]
```

Operands

JOBN — The job number to be tested. The number is to be specified as a binary-halfword.

ERROR — The location to which the FINDJNUM routine returns if the job number is not in use.

NORMAL — The location to which the FINDJNUM routine returns if the number is currently assigned to a job in the ASP system.

FRPENTRY

Functional Description

The FRPENTRY macro-instruction is used to define a DSECT for the fields of a //*FORMAT parameter buffer.

```
Name          Operation       Operands

              FRPENTRY
```

Operands

This macro-instruction contains no operands.

Functional Description

The GETUNIT macro-instruction is used to request one or more devices. This macro may be used by a function whenever it requires devices.

```
Name          Operation       Operands

[symbol]      GETUNIT          [LIST={(R1)|(reg)|addr}]
                               ,NAVAIL={(reg)|addr}
                               ,ERROR={(reg)|addr}
                              [,ALLO={YES|NO}]
                              [,SAVE={YES|}]
                              [,NORMAL={(reg)|addr}]
```

Operands

LIST — The address of a list containing sublists in the format of the GETUNIT List. This parameter should be omitted if the GETUNIT List pointed to by the Function Control Table is to be used. Each sublist contains the eight-byte name fields with each sublist terminated by a double-word of FF's. The total list is terminated by a double-word of 7F's.

NAVAIL — The location to which the GETUNIT routine returns if none of the sublists can be allocated. Register 1 points to the first unavailable device in the last sublist of the list.

ERROR — The location to which the GETUNIT routine returns when the last sublist contains a request for a device which does not exist in the Support Units Table and no previous sublist could be satisfied. Register 1 points to the entry in error in the last sublist.

ALLO — Indicates whether or not the specified devices are to be allocated. ALLO=NO is primarily used to check the validity of a unit request. On a normal exit with ALLO=NO, no devices will have been allocated to the calling DSP.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the GETUNIT routine returns when it is able to satisfy a sublist. Register 1 points to the satisfied sublist.

ICARDRD

Functional Description

The ICARDRD macro-instruction is used exclusively by the ASP Initialization modules to read each Initialization control card.

Name	Operation	Operands
[symbol]	ICARDRD	[NORMAL={(reg)|addr}]

Operands

NORMAL — The location to which control is returned when the control card has been read. The card image is contained in field INAREA, and field INPOINT is initialized to the address of INAREA.

ICONVBIN

Functional Description

The ICONVBIN macro-instruction is used exclusively by the ASP Initialization modules to convert a field from decimal to binary format. The data to be converted must be in field SOPER and its length in field LOPER upon issuance of this macro-instruction.

Name	Operation	Operands
[symbol]	ICONVBIN	[NORMAL={(reg)|addr}]

Operands

NORMAL — The location to which control is returned when the data has been converted. Register 1 contains the converted number.

ICONVHEX

Functional Description

The ICONVHEX macro-instruction is used exclusively by the ASP Initialization modules to convert a field from decimal to hexadecimal format. The data to be converted must be in field SOPER and its length in field LOPER upon issuance of this macro-instruction.

Name	Operation	Operands
[symbol]	ICONVHEX	[NORMAL={(reg)|addr}]

Operands

NORMAL — The location to which control is returned when the data has been converted. Register 1 contains the converted number.

IJPDSECT

Functional Description

The IJPDSECT macro-instruction is used to describe the data area used by the IJP DSP.

Name	Operation	Operands
	IJPDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

ILOCUCB

Functional Description

The ILOCUCB macro-instruction is used by Console Service to find the location of a specified Unit Control Block.

```
Name        Operation      Operands

[symbol]    ILOCUCB         {DEVADD=device-address|DEVNAME=device-name}
                            ,ERROR={(reg)|addr}
                           [,NORMAL={(reg)|addr}]
```

Operands

DEVADD — A pointer to a fullword containing the UCB device address (e.g., label DC X'0000001F').

DEVNAME — A pointer to a fullword containing the UCB device name (e.g., label DC X'F0F0F1C6').

ERROR — The location to which control is returned if the UCB cannot be located.

NORMAL — The location to which control is returned when the UCB has been located. Register 1 contains the location of the UCB.

INCNDATA

Functional Description

The INCNDATA macro-instruction establishes a DSECT which defines a Console Status Table used for Initialization.

```
Name        Operation      Operands

            INCNDATA
```

Operands

This macro-instruction contains no operands.

INITMWLE

Functional Description

The INITMWLE macro-instruction is used exclusively by the ASP Initialization modules to scan for the ending right parenthesis of a control card parameter. The starting location of the scan is in a pointer field called INPOINT, which is maintained by the ICARDRD, ISCAN, and ISCAN2 macro routines.

Name	Operation	Operands
[symbol]	INITMWLE	[NORMAL={(reg)|addr}]

Operands

NORMAL — The location to which control is returned when the terminator has been located. INPOINT is updated to point to the byte following the right parenthesis, unless that byte contains a comma, in which event the pointer is incremented past the comma.

INTDSECT

Functional Description

The INTDSECT macro-instruction is used to describe the data area used by ASP Initialization.

Name	Operation	Operands
	INTDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

INTERCOM

Functional Description

The INTERCOM macro-instruction is used by a DSP to simulate operator input of a console message. The text must be a legitimate operator message.

```
Name          Operation       Operands

[symbol]      INTERCOM          CONS={(R0)|(reg)|DUMMY}
                               ,TEXT={(reg)|addr}
                               [,PREFIX={YES|NO}]
                               [,MSG={YES|NO}]
                               [,CHK={YES|NO}]
                               [,SAVE={(YES|NO}]
                               [,BUSY={(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
```

Operands

CONS — A register containing a halfword console ID number; this console will receive the INTERCOM message and any response to the message. If DUMMY is specified, the message and response will appear only on the MLOG console.

TEXT — The address of a string of characters (ranging from 2 to 71 bytes) that contains the count and message text. The first byte of the character string contains the text count, represented in binary, in the range from 1 to 70.

PREFIX — If NO is specified, the name of the DSP issuing INTERCOM will be displayed at the beginning of the message. If YES is specified, a user-supplied eight-byte prefix will be displayed before the message; this prefix must be the eight bytes preceding the count and text.

MSG — Indicates whether the message is to be displayed on the specified console. In either case, the message will appear on the MLOG console.

CHK — Specifies whether console authority checking is required for the message via CONSAUTH for local consoles or CONSANAL for remote consoles.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

BUSY — The location to which the INTERCOM routine returns if no buffers are available to queue the message. Register 0 contains an ECF mask and address which may be used to wait for available buffers. If BUSY is not specified, INTERCOM will AWAIT off the caller's FCT until buffers become available, then take the NORMAL exit.

NORMAL — The location to which the INTERCOM routine returns when the message has been enqueued.

IOBASPIO

Functional Description

The IOBASPIO macro-instruction establishes a DSECT that defines the fields of the Input/Output Block used by the Disk Input/Output Routines (ASPIO).

Name	Operation	Operands
	IOBASPIO	

Operands

This macro-instruction contains no operands.

IOBENTRY

Functional Description

The IOBENTRY macro-instruction is furnished to define fields of the OS Input/Output Block and to provide labels for frequently referenced fields therein.

Name	Operation	Operands
	IOBENTRY	

Operands

This macro-instruction contains no operands.

IONTABLE

Functional Description

The IONTABLE macro-instruction establishes a DSECT, used by ASPIO, that defines a table of parameters specified during Initialization from information on the BUFFER card.

Name	Operation	Operands
	IONTABLE	

Operands

This macro-instruction contains no operands.

ISCAN

Functional Description

The ISCAN macro-instruction is used exclusively by the ASP Initialization modules to isolate the fields of an Initialization control card. The beginning location of the scan is in field INPOINT.

```
Name          Operation       Operands

[symbol]      ISCAN             EOD={(reg)|addr}
                               [,TYPE={KEY|CARD}]
                               [,NORMAL={(reg)|addr}]
```

Operands

EOD — The location to which control is returned if no operand can be found, that is, the end of the card has been reached.

TYPE — TYPE=KEY scans for a keyword and either a single operand or the first operand of a string. TYPE=CARD scans for the positional parameter identifying the control card.

NORMAL — The location to which control is returned when the requested data has been found. INPOINT is incremented to the next scan location. If TYPE=KEY was specified, the keyword is returned in field SPARAM, the first or only operand in field SOPER, and the length of the operand in field LOPER. If TYPE=CARD was specified, the card name appears in field SPARAM. SPARAM and SOPER are 11-byte fields in which the data is left-justified and padded with blanks.

ISCAN2

Functional Description

The ISCAN2 macro-instruction is used exclusively by the ASP Initialization modules to locate the second or later subfield of a keyword parameter within an Initialization control card. The beginning location of the scan is in field INPOINT.

```
Name          Operation       Operands

[symbol]      ISCAN2            EOD={(reg)|addr}
                               [,NORMAL={(reg)|addr}]
```

Operands

EOD — The location to which control is returned if no operand can be found, i.e., the end of the parameter has been reached.

NORMAL — The location to which control is returned when the subfield has been found. INPOINT is incremented to the next scan location. The subfield is in field SOPER and its length in field LOPER. SOPER is an eleven-byte field in which the data is left-justified and padded with blanks.

ISDSECT

Functional Description

The ISDSECT macro-instruction is used to define a general data area containing flags and pointers used by Input Service.

```
Name        Operation       Operands

            ISDSECT         [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

ISORT

Functional Description

The ISORT macro-instruction is used exclusively by the ASP Initialization modules to sort a group of internal table entries.

```
Name        Operation       Operands

[symbol]    ISORT             LIST={(R1)|(reg)|addr}
                              [,NORMAL={(reg)|addr}]
```

Operands

LIST — The location of a fixed-form list describing the entries to be sorted.

NORMAL — The location to which control is returned when the entries have been sorted.

ITREAD

Functional Description

The ITREAD macro-instruction is used exclusively by the ASP Initialization modules to read an internal table entry from disk.

```
Name          Operation       Operands

[symbol]      ITREAD            SPOOL={(R1)|(reg)|addr}
                               ,EOD={(reg)|addr}
                               [,NORMAL={(reg)|addr}]
```

Operands

SPOOL — The location of a list (in the form generated by the SPOOL macro-instruction) defining the set of table entries to be read.

EOD — The location to which control is returned when the end of the table has been reached.

NORMAL — The location to which control is returned when a table entry has been read. Register 1 contains the address of the entry.

ITWRITE

Functional Description

The ITWRITE macro-instruction is used exclusively by the ASP Initialization modules to write an internal table entry to disk.

```
Name          Operation       Operands

[symbol]      ITWRITE           SPOOL={(R1)|(reg)|addr}
                               [,LENGTH={(R0)|(reg)|addr}]
                               [,ADDR={(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
```

Operands

SPOOL — The location of a list (in the form generated by the SPOOL macro-instruction) defining the set of table entries to be written.

LENGTH — The length of each entry in the table. This field must be supplied to the ITWRITE routine, either by use of this parameter, an assembled value in the spool list, or storing it in the length field before this macro-instruction is issued.

ADDR — The address of the entry to be written. This field must be supplied to the ITWRITE routine, either by use of this parameter, an assembled value in the spool list, or storing it in the address field before this macro-instruction is issued.

NORMAL — The location to which control is returned when the entry has been written to disk.

IWASPOUT

Functional Description

The IWASPOUT macro-instruction is used exclusively by the ASP Initialization modules to write out messages describing errors in the Initialization control cards.

Name	Operation	Operands
[symbol]	IWASPOUT	ERRMSG={(R1)|(reg)|addr} [,NORMAL={(reg)|addr}] [,ERRLVL={C|E|W|N}]

Operands

ERRMSG The location of the error message to be issued.

NORMAL The location to which control is returned when the message has been issued.

ERRLVL The error level for the message:

- W - warning - indicates that an ASP function may possibly be impacted. The system continues.
- E - error - indicates that a failure is likely within ASP, but the system will attempt to continue.
- C - catastrophic - this is the default and indicates further execution beyond initialization is impossible. ASP is terminated at the end of initialization with a DM004 completion code.
- N - no error - indicates a message is to be issued without any error level flag being set.

  At the end of initialization, if any ERRLVL has been issued, a message will be directed to the operator (ERR003) describing the most severe level encountered, and whether or not ASP will continue.

JBTDSECT

Functional Description

The JBTDSECT macro-instruction establishes a DSECT, used by ASPIO, that defines fields of a Track Allocator Table buffer.

Name	Operation	Operands
	JBTDSECT	

Operands

This macro-instruction contains no operands.

JCTENTRY

Functional Description

The JCTENTRY macro-instruction is used to define the fields of an entry in the Job Control Table.

Name	Operation	Operands
	JCTENTRY	

Operands

This macro-instruction contains no operands.

JDABDSCT

Functional Description

The JDABDSCT macro-instruction is used to define the fields of a Job Description-Accounting Block.

Name	Operation	Operands
	JDABDSCT	

Operands

This macro-instruction contains no operands.

JDSENTRY

Functional Description

The JDSENTRY macro-instruction is used to define the fields of a Job Data Sets Block.

Name	Operation	Operands
	JDSENTRY	

Operands

This macro-instruction contains no operands.

JNADD

Functional Description

The JNADD macro-instruction is used to add a JNCB entry to the Job-Net-Control-Block chain. Only synchronous access is allowed to the JNCB chain.

```
Name          Operation       Operands

[symbol]      JNADD             ENTRY={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The address of the JNCB to be added to the chain.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the JNADD routine returns when the JNCB has been added.

JNCBHLD

Functional Description

The JNCBHLD macro-instruction is used to hold a specific JNCB within the Job-Net-Control-Block chain. Only synchronous access is allowed to a specific JNCB.

```
Name          Operation       Operands

[symbol]      JNCBHLD           ID={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

ID — The address of the Net-ID of the JNCB to be placed in hold.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the JNCBHLD routine returns after placing the specified JNCB into hold. Register 1 contains the address of the JNCB.

JNCBREL

Functional Description

The JNCBREL macro-instruction is the converse of the JNCBHLD macro and is used to release a previously held JNCB from hold.

```
Name          Operation       Operands

[symbol]      JNCBREL           ID={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

ID — The address of the Net-ID of the JNCB to be released from hold.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the JNCBREL routine returns after releasing the specified JNCB from hold.

JNDEL

Functional Description

The JNDEL macro-instruction is the converse to JNADD and is used to delete a given JNCB from the Job-Net-Control-Block chain. Only synchronous access is allowed to the JNCB chain.

```
Name          Operation       Operands

[symbol]      JNDEL             ID={(R1)|(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

ID — The address of the Net-ID of the JNCB to be deleted.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the JNDEL routine returns after deleting the specified JNCB.

JNGET

Functional Description

The JNGET macro-instruction is used to access the next JNCB in the Job-Net-Control-Block chain. Only synchronous access is allowed to the JNCB chain.

```
Name          Operation       Operands

[symbol]      JNGET            EOF={(reg)|addr}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

EOF — The location to which the JNGET routine returns when no further JNCBs are encountered in the chain.

SAVE — Indicates whether the contents of Registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the JNGET routine returns when the next JNCB has been found. Register 1 contains the address of the JNCB.

JOBNET

Functional Description

The JOBNET macro-instruction is used to define a Job-Net Control Block DSECT.

```
Name          Operation       Operands

              JOBNET
```

Operands

This macro-instruction contains no operands.

JOBNUMBR

Functional Description

The JOBNUMBR macro-instruction is used to define the storage area for parameters used by the Job Number routine (JOBNUM).

Name	Operation	Operands
	JOBNUMBR	

Operands

This macro-instruction contains no operands.

JSTENTRY

Functional Description

The JSTENTRY macro-instruction is used to define the fields of a Job Setup Table.

Name	Operation	Operands
	JSTENTRY	

Operands

This macro-instruction contains no operands.

JSWKDSCT

Functional Description

The JSWKDSCT macro-instruction defines an area, pointed to by the Job Setup Table, containing information supplied by the SETUP parameter on the //*MAIN control card.

Name	Operation	Operands
	JSWKDSCT	

Operands

This macro-instruction contains no operands.

LOCDSECT

Functional Description

The LOCDSECT macro-instruction defines a Locate reply message and fields defining a Locate entry.

```
Name        Operation       Operands

            LOCDSECT
```

Operands

This macro-instruction contains no operands.

LOGIN

Functional Description

The LOGIN macro-instruction permits a two-way communication and transfer of data between Console Service and the function using the macro-instruction. This macro must be executed by each function that allows the receipt of messages and responses from the consoles. A function must log in prior to any communication with Console Service. LOGIN also obtains time-on for the function via the OS/360 TIME macro and stores it in the Function Control Table pending LOGOUT.

```
Name        Operation       Operands

[symbol]    LOGIN             ENTER={(R1)|(reg)|addr}
                             [,SAVE={YES|NO}]
                             [,NORMAL={(reg)|addr}]
```

Operands

ENTER — The label of an instruction within the resident link of the function. LOGIN puts this address into the Function Control Table, where it becomes available to Console Service. Console Service passes control to this location when a message is to be passed to this function. The function, at this time, accepts or rejects the message and returns control to Console Service. The function cannot take any action on the message at this time.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the LOGIN routine returns when the entry defined by this macro has been added to the Function Control Table and time-on has been obtained and stored.

LOGOUT

Functional Description

The LOGOUT macro-instruction moves a function's time-on from the Function Control Table to the Job Description-Accounting Block and obtains and stores the function's time-off in the Job Description-Accounting Block. It also removes an entry from the Function Control Table that was created by the LOGIN macro. If a function does not log out, the removal of the entry is performed by the Job Segment Scheduler, and no time accounting is done. Depending upon user specification, LOGOUT will perform both the AREAD and AWRITE of the JDAB, either one, or neither.

```
Name        Operation     Operands

[symbol]    LOGOUT        [SE={(R0)|(reg)|addr}]
                          [,AWRITE={YES|NO}]
                          [,SAVE={YES|NO}]
                          [,NORMAL={(reg)|addr}]
```

Operands

SE — The address of the Job Description-Accounting Block Scheduler Element for the function to be logged out. If specified, LOGOUT assumes the user has performed the AREAD; if omitted, LOGOUT performs an AREAD and scans for the required entry.

AWRITE — Indicates whether LOGOUT is to perform the AWRITE of the JDAB before returning to the user. If NO is specified, the user may elect to have LOGOUT AREAD and locate the SE and, on return, he may post any additional accounting information in that entry; the user is then required to perform the AWRITE.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the LOGOUT routine returns when it has completed its functions. If AWRITE=NO was specified, register 0 contains the address of the JDAB SE and register 1 contains the AJDJDFDB address to be supplied to AWRITE; if AWRITE=YES was specified, both these registers are zeroed.

MDSDSECT

Functional Description

The MDSDSECT macro-instruction is used to define the Main Device Scheduler Table.

```
Name        Operation     Operands

            MDSDSECT      [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

Functional Description

The MESSAGE macro-instruction is used by the system functions to transmit a message to a specific console or defined group of consoles.

```
Name          Operation       Operands

[symbol]      MESSAGE           TEXT={(reg)|addr}
                               [,CLASS={ALL|D1...D22|ERR|FET|LOG|MLG|
                                 MN|M1...M32|SUP|S1...S32|TAP|TP|UR|(reg)}]
                               [,CONS={(R0)|(reg)|addr}]
                               [,ACTION={YES|NO|MDS}]
                               [,PRTY={5|0...11}]
                               [,SAVE={YES|NO}]
                               [,BUSY={(reg)|addr}]
                               [,NORMAL={(reg)|addr}]
                               [,MF={(E,{(reg)|addr})|L}]
```

Operands

TEXT — The address of a string of characters (ranging from 2 to 71 bytes) that contains the count and message text. The first byte of the character string contains the text count, represented in binary, in the range from 1 to 70.

CLASS — The message class, as defined in the ASP Initialization control cards, to which the message is directed:

ALL	All consoles.
D1...D22	User-defined configuration.
ERR	System error console.
LOG	System log console.
MLG	Master log console.
MN	Main Processor consoles.
M1...M32	Alternate Main Processor configurations.
SUP	Support Processor consoles.
S1...S32	Alternate Support Processor configurations.
TAP	Tape consoles.
TP	Teleprocessing consoles.
UR	Unit record consoles.
SEC	Security messages. (Not logged on MLOG).

If register notation is used, the register must contain a halfword (one-byte mask and one-byte displacement) converted from the class name (see CONCNVRT).

CLASS is required if CONS is omitted (both may be specified).

CONS — If register notation is used, the halfword console number of the console to which the message is directed. Else, the address of a halfword containing this number. CONS is required if CLASS is omitted (both may be specified).

ACTION — Indicates whether the message is to be entered into an action queue, which may be inquired upon. Messages soliciting operator action fall into this category. Action messages must be dequeued via DEQMSG when the requested action has been completed. The buffer address of the message in the action queue is returned in register 0. MDS is a special designation whose use is restricted to the Main Device Scheduler.

PRTY — The priority at which the message is to be entered into the transmission queue. The following priorities have special meanings:

- 0 — The message will not be printed unless directed to MLG. This option may be used to place a message in the action queue only. In this one case, CLASS and CONS may both be omitted.
- 9-11 — Messages may be queued past the depth specified in the ASP Initialization CONSOLE cards.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

BUSY — The location to which the MESSAGE routine returns if no buffers are available to queue the message. Register 0 contains an ECF mask and address which may be used to AWAIT for buffer availability. If this parameter is omitted, MESSAGE will AWAIT off the user's FCT until able to queue the message, then return via the NORMAL exit.

NORMAL — The location to which the MESSAGE routine returns when the message has been successfully queued for transmission. If ACTION=YES was specified, register 0 contains the address of the action queue buffer.

In addition to the above form, MESSAGE may be issued in executable and list form.

The executable form consists of the parameters noted above (except that TEXT is optional), plus the parameter MF=(E,{(reg)|addr}) in which the second parameter provides the address of the list-form expansion of the macro.

The list form consists of the text of the message plus the parameter MF=L. This form requires a label.

MOVEDATA

Functional Description

The MOVEDATA macro-instruction, used by ASPIO, facilitates moves of data longer than 256 bytes.

Name	Operation	Operands
[symbol]	MOVEDATA	[TO={(R2)|(reg)|addr}] [,FROM={(R3)|(reg)|addr}] [,COUNT={(R4)|(reg)|n}]

Operands

TO — The location to which the data is to be moved.

FROM — The location from which the data is to be moved.

COUNT — The number of bytes to be moved.

MPCLSTAB

Functional Description

The MPCLSTAB macro-instruction defines a Main Processor Job Class Table.

Name	Operation	Operands
	MPCLSTAB	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

MPCTLTAB

Functional Description

The MPCTLTAB macro-instruction is used to define the Main Processor Control Table. The FCTENTRY macro-instruction must be issued prior to using this macro.

Name	Operation	Operands
	MPCTLTAB	[TYPE={CSECT|DSECT}] [,SYSTEM={REAL|DUMMY}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

SYSTEM — DUMMY causes the inclusion of the dummy Main Processor extension to the MPCDATA control section.

MPENTRY

Functional Description

The MPENTRY macro-instruction, used by Main Service, defines an area containing information supplied by the Initialization MAINPROC control card.

Name	Operation	Operands
	MPENTRY	[TYPE=CSECT]

Operands

TYPE CSECT establishes a real control section data area. Otherwise, the statements are generated as part of the existing control section.

MPGRPTAB

Functional Description

The MPGRPTAB macro-instruction defines a Main Processor Job Class Group Table.

Name	Operation	Operands
	MPGRPTAB	[TYPE={CSECT|DSECT}]

Operands

TYPE CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

MSENTRY

Functional Description

The MSENTRY macro-instruction, used by Main Service, defines an area containing information supplied by the Initialization SELECT control card.

Name	Operation	Operands
	MSENTRY	[TYPE=CSECT]

Operands

TYPE CSECT establishes a real control section data area. Otherwise, the statements are generated as part of the existing control section.

MSQDATAX

Functional Description

The MSQDATAX macro-instruction is used to establish a dummy work area control section used by the MSVQMAP module of Main Service.

Name	Operation	Operands
	MSQDATAX	

Operands

This macro-instruction contains no operands.

MSVDSECT

Functional Description

The MSVDSECT macro-instruction is used to describe the data area used by Main Service.

Name	Operation	Operands
	MSVDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

MTSVC

Functional Description

The MTSVC macro-instruction gets addressability to the Maintask Vector Table.

Name	Operation	Operands
[symbol]	MTSVC	[VECTOR={MTVT|ATN}]

Operands

VECTOR — If MTVT is given, register 1 upon return from issuance of SVC246 contains a pointer to the beginning of the Maintask Vector Table. If ATN is specified, in addition, the index value of the CTC attention entry in the IOS Attention Table is placed in the Maintask Vector Table.

MTVEREQU

Functional Description

The MTVEREQU macro-instruction generates the ID character equates used in the MAINTASK VERIFY responses. It also defines the control characters used in the VERIFY command.

Name	Operation	Operands
	MTVEREQU	[TYPE={ASP|MT}]

Operands

TYPE — TYPE=ASP generates both external and internal equates. TYPE=MT generates only the internal VERIFY response characters.

MTVT

Functional Description

The MTVT macro-instruction generates a CSECT or DSECT of the MAINTASK Vector Table.

Name	Operation	Operands
	MTVT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

NCB

Functional Description

The NCB macro-instruction is used to define a Net-Control-Block DSECT.

Name	Operation	Operands
	NCB	

Operands

This macro-instruction contains no operands.

NCBTAADD

Functional Description

The NCBTAADD macro-instruction is used to add a given Net-Control-Block to the applicable JNCB. Only synchronous access is allowed.

```
Name        Operation      Operands

[symbol]    NCBTAADD         NCSECT={(R1)|(reg)|addr}
                            [,SAVE={YES|NO}]
                            [,NORMAL={(reg)|addr}]
```

Operands

NCSECT — The address of the Job-Net-Control CSECT which contains the calling parameter list.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which NCBTAADD returns when the given NCB has been added to the applicable JNCB. Register 1 contains the address of the NCB.

NCBTAFND

Functional Description

The NCBTAFND macro-instruction is used to locate a specific Net-Control-Block within a given Job-Net-Control-Block. Only synchronous access is allowed.

```
Name        Operation      Operands

[symbol]    NCBTAFND         ENTRY={(R0)|(reg)|addr}
                            ,NCB={(R1)|(reg)|addr}
                            [,SAVE={YES|NO}]
                            [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — This operand specifies the JNCB of which the given Net-Control-Block is a member.

NCB — The address of the job name associated with the Net-Control-Block to be found.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the NCBTAFND routine returns after locating the requested NCB. Register 1 contains the address of the requested NCB.

NCBTAGET

Functional Description

The NCBTAGET macro-instruction is used to access the next Net-Control-Block within a given JNCB. Only synchronous access is allowed.

```
Name          Operation      Operands

[symbol]      NCBTAGET         ENTRY={(R0)|(reg)|addr}
                              ,EOF={(reg)|addr}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The address of the JNCB to be accessed.

EOF — The location to which NCBTAGET returns when all NCB's have been accessed.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which NCBTAGET returns when the next NCB has been located. Register 1 contains the address of the NCB.

NCBTAPUT

Functional Description

The NCBTAPUT macro-instruction is used to write to disk an updated NCB buffer.

```
Name          Operation      Operands

[symbol]      NCBTAPUT         ENTRY={(R1)|(reg)|addr}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The address of the JNCB of which the applicable NCB is a member.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the NCBTAPUT routine returns when the NCB buffer has been written to disk.

NCBTAREL

Functional Description

The NCBTAREL macro-instruction is used to release a previously read NCB buffer.

```
Name          Operation       Operands

[symbol]      NCBTAREL          ENTRY={(R1)|(reg)|addr}
                                [,SAVE={YES|NO}]
                                [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The address of the JNCB of which the applicable NCB is a member.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the NCBTAREL routine returns when the NCB buffer has been released.

NETDSECT

Functional Description

The NETDSECT macro-instruction is used to describe the data area used in Dependent Job Control net-creation processing.

```
Name          Operation       Operands

              NETDSECT        [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

NJODSECT

Functional Description

The NJODSECT macro-instruction is used to establish either a real or a dummy data area control section to be used by the NJP routines. It contains the DCB, DECB, and LERB for an NJP line.

Name	Operation	Operands
	NJODSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

NJPDSECT

Functional Description

The NJPDSECT macro-instruction is used to establish either a real or a dummy data area control section to be used by the NJP routines.

Name	Operation	Operands
	NJPDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

NJPENTRY

Functional Description

The NJPENTRY macro-instruction defines the fields in the NJP Terminal Table.

Name	Operation	Operands
	NJPENTRY	

Operands

This macro-instruction contains no operands.

NJPPARBF

Functional Description

The NJPPARBF macro-instruction defines the fields in the NJPIO Scheduler Element Parameter Buffer.

Name	Operation	Operands
	NJPPARBF	

Operands

This macro-instruction contains no operands.

PCHDSECT

Functional Description

The PCHDSECT macro-instruction is used to define data storage areas and a console message acceptance routine for Punch Service.

Name	Operation	Operands
	PCHDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

PCW

Functional Description

The PCW macro-instruction, the use of which is restricted to ASPABEND, describes the location and attributes of a print line.

Name	Operation	Operands
[symbol]	PCW	LINE=addr [,INDENT={0|n}] [,LAST={YES|NO}]

Operands

LINE The address of the print line, which consists of a halfword spacing code and text.

INDENT The number of leading blanks to be inserted before the text.

LAST LAST=YES indicates that this is the last print line of a sequence; LAST=NO indicates other lines will follow.

PFKENTRY

Functional Description

The PFKENTRY macro-instruction is used by Console Service to generate a DSECT to map a Program Function Key Table entry.

Name	Operation	Operands
	PFKENTRY	

Operands

This macro-instruction contains no operands.

PRTABLE

Functional Description

The PRTABLE macro-instruction is used to define the fields of a table describing the characteristics of printers attached to the Support Processor (Printer Resources Table). The table is generated during Initialization from PRINTER control cards and is used by Print Service in assigning tasks to the printers.

Name	Operation	Operands
	PRTABLE	

Operands

This macro-instruction contains no operands.

PRTDSECT

Functional Description

The PRTDSECT macro-instruction defines the data control section required by Print Service for each printer attached to the ASP system.

```
Name        Operation       Operands

            PRTDSECT        [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

PURCHAIN

Functional Description

The PURCHAIN macro-instruction is used to purge chained single-record files. The single-record files must have obtained their tracks from the Single Track Table (STT) via TATPTR=MNTRKFDB.

```
Name        Operation       Operands

[symbol]    PURCHAIN          FDB={(R1)|(reg)|addr}
                             ,AREA={(R0)|(reg)|addr}
                            [,SAVE={YES|NO}]
                            [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address for the data set being purged.

AREA — The address of a four-fullword work area to be used by PURCHAIN. The halfword at area+6 must contain the displacement to the chain FDB in each buffer.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the PURCHAIN routine returns when all files in the chain have been purged.

PURDSECT

Functional Description

The PURDSECT macro-instruction is used to define a general data area used by PURGE.

```
Name        Operation       Operands

            PURDSECT        [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

PUTUNIT

Functional Description

The PUTUNIT macro-instruction is used to return one or more devices to availability for allocation to other functions. The devices being returned were received via the GETUNIT macro-instruction or by initial allocation by the Job Segment Scheduler. If, in either case, the devices were obtained by the list associated with the Function Control Table and are required for the duration of the function, PUTUNIT may be omitted, and the Job Segment Scheduler will return the allocated devices when the function returns control to it. PUTUNIT may be used to return units when they are required for only a small fraction of the running time (for example, reading control cards) in order to make the device available to other functions. PUTUNIT must be used if the GETUNIT specified a series of sublists (LIST parameter) in order to supply the location of the list.

```
Name        Operation       Operands

[symbol]    PUTUNIT          [LIST={(R1)|(reg)|addr}]
                             [,SAVE={YES|NO}]
                             [,NORMAL={(reg)|addr}]
```

Operands

LIST — The address of the list which was used by GETUNIT for allocation. This operand should be omitted if the sublist pointed to by the Function Control Table is used.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the PUTUNIT routine returns after the devices have been returned.

RDDSECT

Functional Description

The RDDSECT macro-instruction is used to define a data area control section used by the Input Service DSP's (CR, DR, TR).

```
Name        Operation       Operands

            RDDSECT         [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

RECEIVE

Functional Description

The RECEIVE macro-instruction is used by system functions to access the console output messages from a particular Main Processor.

```
Name        Operation       Operands

[symbol]    RECEIVE          {BUFFER|FDB}={(R1)|(reg)|addr}
                            ,{ID={(R0)|(reg)|addr}|TYPE={ANY|PURGE}}
                             ,MPC={(R2)|(reg)|addr}
                            [,DEQ={AUTO|USER}]
                            [,WAIT={YES|NO}]
                            [,SAVE={YES|NO}]
                            [,IPL={(reg)|addr}]
                            [,BUSY={(reg)|addr}]
                            [,NORMAL={(reg)|addr}]
```

Operands

BUFFER — The location of a buffer containing control information and a data area into which the received message will be moved. The format of the buffer is:

Byte 0 An ECF which is posted as follows:

- X'80' when the requested message is received
- X'40' if an IPL occurs
- X'20' if DEQ=USER is specified and the buffer is still posted (not available). This posting means that a requested message has been missed.

Byte 1 The length in binary of the following data area.

Bytes 2-x The data area, the maximum length of which is 120 bytes.

FDB — The FDB specification is used only to receive Locate responses from a Main Processor. It specifies the location of an ECF followed by two fullwords containing pointers to a buffer and an FDB. The Locate response will be placed into the buffer if only a single response is necessary. A single response will contain from one to three volume serials. The Locate responses will be ABLOCKed into the multirecord FDB if multiple responses are necessary. The ECF will be posted as follows:

- X'80' A single Locate response was placed into the provided buffer. The FDB remains open.
- X'40' An IPL occurred before the last response was received.
- X'08' Multiple Locate responses were received and ABLOCKed into the FDB. The FDB was closed.

ID — The location of a field containing a one-byte binary count of ID characters followed by the one- to eight-byte ID characters of the message to be received.

TYPE — The TYPE parameter is used in conjunction with the BUFFER parameter and is invalid with FDB. TYPE=ANY is used to receive any message that occurs on a particular Main Processor. TYPE=PURGE causes all receive entries with the supplied buffer address to be dequeued.

MPC — The address of the Main Processor Control Table for the Main Processor from which the message is to be received.

DEQ — DEQ=AUTO causes the receive entry to be dequeued when the ECF is posted. DEQ=USER causes the receive entry to remain active after a message has been received; the user must subsequently issue a 'RECEIVE TYPE=PURGE,BUFFER=addr' to dequeue the receive entry. DEQ=USER is invalid with the FDB parameter.

WAIT — WAIT=YES causes the receive routine to AWAIT until the receive occurs and the ECF is posted before returning control to the user. WAIT=NO causes control to be returned before the receive has been satisfied; the user must check the ECF to determine when the receive has been satisfied. If it is later desired to await for a message, either an AWAIT on the ECF, or another RECEIVE may be used. If another RECEIVE is used the BUFFER and ID must be the same as before, however the operands SAVE, IPL, BUSY, or NORMAL may be changed. WAIT=NO is invalid with TYPE=PURGE.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

IPL — The location to which control is returned if an IPL occurs while RECEIVE is AWAITing for a response.

BUSY — The location to which RECEIVE returns if no main storage is available for the entry. Register 0 contains an ECF mask and address which may be used to AWAIT for storage availability. If the parameter is omitted, RECEIVE will AWAIT until able to receive the message, then return via the NORMAL exit. This parameter is invalid with TYPE=PURGE.

NORMAL — The location to which RECEIVE returns when it has successfully completed its functions.

REGISTER

Functional Description

The REGISTER macro-instruction provides standardized symbolic register notation. It consists of EQU statements as shown below:

```
R0      EQU     0
R1      EQU     1
R2      EQU     2
R3      EQU     3
R4      EQU     4
R5      EQU     5
R6      EQU     6
R7      EQU     7
R8      EQU     8
R9      EQU     9
R10     EQU     10
R11     EQU     11
R12     EQU     12
R13     EQU     13
R14     EQU     14
R15     EQU     15
```

```
Name        Operation       Operands

            REGISTER
```

Operands

This macro-instruction contains no operands.

RELSAVE

Functional Description

The RELSAVE macro-instruction releases the register save areas obtained by RSVSAVE.

```
Name        Operation       Operands

[symbol]    RELSAVE           NUMBER={ALL|(R0)|(reg)|n}
                             ,ERROR={(reg)|addr}
                            [,FCT={(R1)|(reg)|addr}]
                            [,NORMAL={(reg)|addr}]
```

Operands

NUMBER — The number of save areas to be released. If n is zero, all reserved save areas are released.

ERROR — The location to which the RELSAVE routine returns if the number of save areas available to be released is less than the number specified. Register 1 contains the residual count.

FCT The address of the Function Control Table containing the save areas to be released. If omitted, the current FCT is used.

NORMAL The location to which the RELSAVE routine returns when the save areas have been returned.

RESOURCE

Functional Description

The RESOURCE macro-instruction specifies the names of critical ASP system resources for use with ADEQ, AENQ, and ATEST.

Name	Operation	Operands
[symbol]	RESOURCE	{NAMES=(name1,name2,...namen)|LIST}

Operands

NAMES Generates a series of equate statements equating each specified resource name with a unique decimal number.

LIST RESOURCE LIST must be preceded by a previous expansion of the NAMES forms. It generates a list of 8-byte constants of the previously-defined names.

RESQUEUE

Functional Description

The RESQUEUE macro-instruction is used to define a Resident Job Queue Table DSECT.

Name	Operation	Operands
	RESQUEUE	

Operands

This macro-instruction contains no operands.

RIAMEP

Functional Description

The RIAMEP macro-instruction establishes either a dummy or defined work area control section used to establish the RICBAM entry points.

Name	Operation	Operands
	RIAMEP	[TYPE={DSECT|NULL}]

Operands

TYPE — DSECT establishes a dummy control section data area. NULL establishes a defined control section data area which can be a part of another real control section.

RICBAMX

Functional Description

The RICBAMX macro-instruction is used to generate instructions which represent the ASP Reader/Interpreter control block access method.

Name	Operation	Operands
	RICBAMX	TYPE={RIAM|RIQM}

Operands

TYPE — RIAM generates code for the RICBAM module. RIQM generates code for the RIQMSERV module.

RICLOSE

Functional Description

The RICLOSE macro-instruction is used to close the ASP Reader/Interpreter control block file. Register 13 is assumed to be pointing to the RIODATA CSECT.

Name	Operation	Operands
[symbol]	RICLOSE	[NORMAL={(reg)|addr}]

Operands

NORMAL — The location to which the RICLOSE routine returns when the close has been satisfied.

RICRE

Functional Description

The RICRE macro-instruction is used to create a new LTTR entry within the ASP R/I LTTR table. Register 13 is assumed to be pointing to the RIODATA CSECT.

Name	Operation	Operands
[symbol]	RICRE	[NORMAL={(reg)|addr}]

Operands

NORMAL The location to which the RICRE routine returns when the entry has been created. Register 0 contains the LTTR entry number.

RIDATAX

Functional Description

The RIDATAX macro-instruction is used to establish either a real or dummy work area control section used by the ASP Reader/Interpreter DSP.

Name	Operation	Operands
	RIDATAX	[TYPE={CSECT|DSECT}]

Operands

TYPE CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

RIEXITX

Functional Description

The RIEXITX macro-instruction establishes either a dummy or defined work area control section used to establish the RIEXIT entry points.

Name	Operation	Operands
	RIEXITX	[TYPE={DSECT|NULL}]

Operands

TYPE DSECT establishes a dummy control section data area. NULL establishes a defined control section data area which can be a part of another real control section.

RIGET

Functional Description

The RIGET macro-instruction is used to read an OS Reader/Interpreter control block from the ASP queue packs. Register 13 is assumed to be pointing to the RIODATA CSECT.

```
Name        Operation      Operands

[symbol]    RIGET            LTTR={(R0)|(reg)|addr}
                            [,NORMAL={(reg)|addr}]
```

Operands

LTTR — The logical TTR which is associated with this control block to be read.

NORMAL — The location to which the RIGET routine returns when the read request has been satisfied. Register 1 contains a pointer to the record. Register 15 contains the return code:

- 0 Valid return
- 4 Zero LTTR given
- 8 Out-of-table extent

RIODATAX

Functional Description

The RIODATAX macro-instruction is used to establish either a real, a dummy or a defined work area control section used by the ASP Reader/Interpreter control block access method.

```
Name        Operation      Operands

            RIODATAX       [TYPE={CSECT|DSECT|NULL}]
```

Operands

TYPE — CSECT establishes a real control section data area named RIODATA. DSECT establishes a dummy control section data area named RIOSTART. NULL establishes a defined control section data area which can be a part of another real control section.

RIOENTRY

Functional Description

The RIOENTRY macro-instruction is used to define the LTTR table header and entries for the ASP Reader/Interpreter control block access method.

Name	Operation	Operands
	RIOENTRY	

Operands

This macro-instruction contains no operands.

RIPUT

Functional Description

The RIPUT macro-instruction is used to write an OS Reader/Interpreter control block to the ASP queue packs. Register 13 is assumed to be pointing to the RIODATA CSECT.

Name	Operation	Operands
[symbol]	RIPUT	LTTR={(R0)|(reg)|addr} ,DATA={(R1)|(reg)|addr} [,NORMAL={(reg)|addr}]

Operands

LTTR — The logical TTR which is associated with the control block to be written.

DATA — The address of the record to be written.

NORMAL — The location to which the RIPUT routines returns when the write request has been satisfied. Register 15 contains the return code:

- 0 Valid return
- 4 Zero LTTR given
- 8 Out-of-table extent

RIREC

Functional Description

The RIREC macro-instruction is used to describe the control block entries of the ASP R/I control block file.

Name	Operation	Operands
	RIREC	

Operands

This macro-instruction contains no operands.

RISERV

Functional Description

The RISERV macro-instruction is used to perform any one or a combination of the following functions for the ASP Reader/Interpreter interface: create a RIODATA work area, read in the LTTR tables, write OS SMB's to SYSMSG (with or without an error message), free the LTTR tables, or free the RIODATA work area.

Name	Operation	Operands
[symbol]	RISERV	JDSFDB={(R9)|(reg)|addr} [,IOAREA={SUPL|CREATE|DELETE}] [,TABLE={SUPL|CREATE|DELETE}] [,SMBDEB={YES|NO}] [,MSG={(R8)|(reg)|addr}] [,NORMAL={(reg)|addr}]

Operands

JDSFDB — The JDS FDB pointer which is associated with the job to be handled. If register 9 is used, no load is generated; however, if register 9 is not used, it will not be destroyed on exit from the routine. The JDS will be in the same status on exit from the routine as it was on entry, that is, in core or not in core.

Note: If either MSG and/or SMBDEB=YES is specified and the FCT does not contain the job's TAT pointer (such as Main Service), the caller should have the JDS in core and SYSMSG opened on entry.

IOAREA — SUPL if register 13 points to the RIODATA on entry. CREATE if user requests a RIODATA to be built or if user requires a RIODATA temporarily for SMBDEB=YES. DELETE if register 13 points to the RIODATA obtained from a previous RISERV macro.

TABLE — SUPL if user has LTTR table in core on entry (table pointer contained in RIODATA). CREATE if user requests LTTR tables

to be read into core or if user requires LTTR tables temporarily for SMBDEB=YES. DELETE if user requests the LTTR tables to be deleted from core.

Note: Tables will be read from the JDS entry of JCBTAB.

SMBDEB — NO if user does not require SMB deblocking into SYSMSG. YES if user requests SMBs to be deblocked into SYSMSG (Note: SYSMSG open status will remain the same as on entry to the macro).

MSG — Address of a one-byte length field followed by the message text. This message will be written to SYSMSG prior to the deblocked SMB's if specified with SMBDEB=YES.

NORMAL — The location to which RISERVX returns when the requested service has been accomplished. If IOAREA=CREATE and SMBDEB=NO, register 13 will point to the RIODATA area requested by the user. Otherwise, the contents of register 13 remain unchanged. In either case, registers 2 through 12 will not be destroyed.

RITABLE

Functional Description

The RITABLE macro-instruction generates a table of pointers to R/I data areas that are accessed by the ASP Reader/Interpreter modules.

Name	Operation	Operands
	RITABLE	

Operands

This macro-instruction contains no operands.

RITPT

Functional Description

The RITPT macro-instruction is used to point at a record entry within the ASP R/I LTTR table. Register 13 is assumed to be pointing to the RIODATA CSECT.

```
Name          Operation       Operands

[symbol]      RITPT             LTTR={(R0)|(reg)|addr}
                                [,NORMAL={(reg)|addr}]
```

Operands

LTTR — The logical TTR which is associated with the control block entry to be accessed.

NORMAL — The location to which the RITPT routine returns when the point has been satisfied. Register 1 contains a pointer to the entry. Register 15 contains the return code:

- 0 Valid return
- 4 Zero LTTR given
- 8 Out-of-table extent

RJPBUF

Functional Description

The RJPBUF macro-instruction is used to define the fields of the RJP TP buffer.

```
Name          Operation       Operands

              RJPBUF          [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

RJPDCT

Functional Description

The RJPDCT macro-instruction is used to establish a map of a line and/or device control block. It is used by the Remote Job Processing routines.

```
   Name          Operation        Operands

                 RJPDCT           [TYPE={CSECT|DSECT}]
                                  [,DEVICE={REMOTE|LINE|ALL}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

DEVICE — REMOTE generates a Remote Device Control Block (RDCT), LINE generates a line control block, and ALL, the default option, generates a composite control block for both line and devices.

RJPDSECT

Functional Description

The RJPDSECT macro-instruction is used to establish either a real or a dummy data area control section used by the Remote Job Processing routines.

```
   Name          Operation        Operands

                 RJPDSECT         [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

RJPTABLE

Functional Description

The RJPTABLE macro-instruction is used to establish either a real or dummy data area section. It defines entries in a resident table used by the Remote Job Processing routines. It contains file pointers and other information related to each terminal and/or line used by RJP.

```
Name        Operation       Operands

            RJPTABLE         [TYPE={CSECT|DSECT}]
                             [,ENTRY={RESIDENT|INISH|ALL}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

ENTRY — RESIDENT, the default option, means the defined CSECT will be resident in core after ASP Initialization. INISH specifies the table is being created for either a line or terminal at Initialization. ALL specifies the DSECT defines all RJP terminal and line entries.

ROUTE

Functional Description

The ROUTE macro-instruction is used by Main Service to route the console message from a Main Processor to any enqueued receives for the message. Register 9 must be preloaded with the address of the appropriate Main Processor Control Table.

```
Name        Operation       Operands

[symbol]    ROUTE            [TYPE=IPL]
                             [,SAVE={YES|NO}]
                             [,NORMAL={(reg)|addr}]
```

Operands

TYPE — If TYPE=IPL is specified, all receive ECF's will be posted with X'40'.

SAVE — Indicate whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which control is returned when ROUTE has completed its functions.

RQTAADD

Functional Description

The RQTAADD macro-instruction is used to add an entry to the Resident Job Queue. The entry will remain at the present core location.

Name	Operation	Operands
[symbol]	RQTAADD	ENTRY={(R1)|(reg)|addr} [,INDEX={(R0)|(reg)|rqindex}] [,NORMAL={(reg)|addr}]

Operands

ENTRY — The Resident Job Queue entry address.

INDEX — The chain into which the entry is to be placed. If the rqindex form is used, the specification must be one of the terms defined for field RQINDEX in the RESQUEUE macro-instruction. If omitted, the index in the entry is used.

NORMAL — The location to which the RQTAADD routine returns control when the entry has been added.

RQTADEL

Functional Description

The RQTADEL macro-instruction is used to delete an entry from the Resident Job Queue. The entry will remain at its present core location and its area will not be freed.

Name	Operation	Operands
[symbol]	RQTADEL	ENTRY={(R1)|(reg)|addr} [,NORMAL={(reg)|addr}]

Operands

ENTRY — The Resident Job Queue entry address.

NORMAL — The location to which the Table Delete routine returns control after the entry has been deleted.

RQTAGEN

RQTAGEN is an inner macro which services the common parameters of RQTAADD , RQTADEL , and RQTAPUT

RQTAPUT

Functional Description

The RQTAPUT macro-instruction is used to move a RESQUEUE entry from one chain to another or change the priority within a chain.

```
Name        Operation     Operands

[symbol]    RQTAPUT        ENTRY={(R1)|(reg)|addr}
                          [,INDEX={(R0)|(reg)|rqindex}]
                          [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The Resident Job Queue entry address.

INDEX — The chain into which the entry is to be placed. If the rqindex form is used, the specification must be one of the terms defined for field RQINDEX in the RESQUEUE macro-instruction. If omitted, the index in the entry is used.

NORMAL — The location to which the Table Put routine returns when a Resident Job Queue entry has been processed.

RSVSAVE

Functional Description

The RSVSAVE macro-instruction obtains register save areas for the exclusive use of the specified Function Control Table.

```
Name        Operation     Operands

[symbol]    RSVSAVE         NUMBER={(R0)|(reg)|n}
                           ,BUSY={(reg)|addr}
                          [,FCT={(R1)|(reg)|addr}]
                          [,SAVE={YES|NO}]
                          [,NORMAL={(reg)|addr}]
```

Operands

NUMBER — The number of save areas to be obtained.

BUSY — The location to which the RSVSAVE routine returns if there is insufficient core available to obtain the required areas. Register 0 contains an ECF mask and address which may be used to AWAIT for sufficient core.

FCT — The address of the Function Control Table to receive the save areas. If omitted, the current FCT is used.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the RSVSAVE routine returns when the save areas have been obtained.

SAVENTRY

Functional Description

The SAVENTRY macro-instruction provides a map of the save area used by the ASAVE routine.

```
Name        Operation       Operands

            SAVENTRY
```

Operands

This macro-instruction contains no operands.

SCTABLE

Functional Description

The SCTABLE macro-instruction establishes a DSECT used to define fields of a table describing the resource characteristics of the SYSOUT output classes.

```
Name        Operation       Operands

            SCTABLE
```

Operands

This macro-instruction contains no operands.

SEND

Functional Description

The SEND macro-instruction is used by system functions to send console messages to a Main Processor.

```
Name        Operation       Operands

[symbol]    SEND              TEXT={(R1)|(reg)|addr}
                             ,MPC={(R0)|(reg)|addr}
                            [,TYPE={MODIFY|F}]
                            [,SAVE={YES|NO}]
                            [,BUSY={(reg)|addr}]
                            [,NORMAL={(reg)|addr}]
```

Operands

TEXT — The address of a character string from 2 to 121 characters long, containing the count in binary in the first byte followed by the data to be sent.

MPC — The address of the Main Processor Control Table for the Main Processor to which the message is being sent.

TYPE — TYPE=MODIFY or TYPE=F indicates that a prefix of 'F MT' is to precede the text. This specification causes a modify of MAINTASK and passes the message to a subtask of MAINTASK.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

BUSY — The location to which the SEND routine returns if no main storage is available for the message. Register 0 contains an ECF mask and address which may be used to AWAIT for storage availability. If this parameter is omitted, SEND will AWAIT until able to transmit the message, then return via the NORMAL exit.

NORMAL — The location to which the SEND routine returns when the message has been successfully transmitted.

SETNAMES

Functional Description

The SETNAMES macro-instruction is used to define the fields of a table of devices that may be set up. The table is generated during Initialization from SETNAME cards and is used by the Main Device Scheduler to correlate unit parameters to device types in the Setup Units Table (SETUNITS).

Name	Operation	Operands
	SETNAMES	

Operands

This macro-instruction contains no operands.

SETUNITS

Functional Description

The SETUNITS macro-instruction is used to define the fields of a Setup Units Table entry for devices on a Main Processor.

Name	Operation	Operands
	SETUNITS	

Operands

This macro-instruction contains no operands.

SMRENTRY

Functional Description

The SMRENTRY macro-instruction is used to define the fields of a chained single-record file which contains the select mode records. These records are generated during initialization from SELECT control cards and are used by Main Service when a SELECT mode change is requested via the ASP MODIFY command. The FDB for the first SMRENTRY record is in the SMRFDB field in TVTABLE.

Name	Operation	Operands
	SMRENTRY	

Operands

This macro-instruction contains no operands.

SORTLIST

Functional Description

The SORTLIST macro-instruction defines an internal sort list used by ASP Initialization.

Name	Operation	Operands
	SORTLIST	

Operands

This macro-instruction contains no operands.

SPOOL

Functional Description

The :SPOOL: macro-instruction is used exclusively by the ASP Initialization modules to generate a three-word list defining a set of tables which are processed by the ITREAD and ITWRITE macro-instructions.

Name	Operation	Operands
symbol	SPOOL	[LENGTH=nn] [,ADDR=addr]

Operands

LENGTH — A decimal value representing the length of each entry in the associated table. If omitted, this halfword value must be stored at symbol+2 prior to issuance of the ITWRITE macro-instruction or supplied to that macro.

ADDR — The address of the associated table. If omitted, this fullword value must be stored at symbol+4 prior to issuance of the ITWRITE macro-instruction or supplied to that macro.

STTABLE

Functional Description

The STTABLE macro-instruction defines the fields in the ASPIO Single Track Table.

Name	Operation	Operands
	STTABLE	

Operands

This macro-instruction contains no operands.

SUPUNITS

Functional Description

The SUPUNITS macro-instruction is used to define the fields of a Support Units Table entry for devices on the Support Processor.

Name	Operation	Operands
	SUPUNITS	

Operands

This macro-instruction contains no operands.

SVCTABLE

Functional Description

The SVCTABLE macro-instruction generates a table that contains pointers used by ASP when intercepting all SVC6 and SVC34 commands issued while ASP is in operation.

Name	Operation	Operands
	SVCTABLE	[TABLE={CSECT|DSECT|NULL}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area. NULL establishes a defined control section data area which can be a part of another real control section.

SYSUNITS

Functional Description

The SYSUNITS macro-instruction is used to define the fields of a System Units Table entry.

Name	Operation	Operands
	SYSUNITS	

Operands

This macro-instruction contains no operands.

TAADD

Functional Description

The TAADD macro-instruction is used to add an entry to the Job Control Table.

```
Name          Operation       Operands

[symbol]      TAADD             ENTRY={(R1)|(reg)|addr}
                               ,PRTY={(R0)|(reg)|n}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

ENTRY — The Job Control Table entry address.

PRTY — A number that specifies the job priority associated with the Job Control Table entry in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the Table Add routine returns when an entry has been added to the Job Control Table.

TACMPR

Functional Description

The TACMPR macro-instruction compresses a specified Job Control Table priority level by removing extraneous space from it.

```
Name          Operation       Operands

[symbol]      TACMPR            PRTY={(R0)|(reg)|n}
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

PRTY — A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the Table Compress routine returns when its functions have been completed.

TADEL

Functional Description

The TADEL macro-instruction is used to delete a completed entry from the Job Control Table.

```
Name          Operation       Operands

[symbol]      TADEL             PRTY={(R0)|(reg)|n}
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

PRTY — A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the Table Delete routine returns when an entry has been deleted from the Job Control Table.

TAFIND

Functional Description

The TAFIND macro-instruction is used to obtain a specific job's entry or the first entry in a Job Control Table priority.

```
Name          Operation       Operands

[symbol]      TAFIND            PRTY={(R0)|(reg)|n}
                               ,EOF={(reg)|addr}
                              [,{JOBNUM={(R1)|(reg)|n}|
                                 JOBNAME={(R1)|(reg)|addr}}]
                               [,SAVE={YES|NO}]
                               [,NORMAL={(reg)|addr}]
```

Operands

PRTY — A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

EOF — The location to which the Table Find routine returns if an entry is not found at the specified priority.

JOBNUM Job number. If specified, the Job Control Table is scanned for the entry associated with this job number. The job number is specified as binary-halfword.

JOBNAME Job name. If specified, the Job Control Table is scanned for the entry associated with this job name. The job name is specified as an eight-byte, left-justified, EBCDIC field and padded with blanks.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the Table Find routine returns when a job has been located. Registers 0 contains the entry length and register 1 contains the address of the table obtained.

TAGET

Functional Description

The TAGET macro-instruction is used to access the next entry in the Job Control Table.

```
Name        Operation      Operands

[symbol]    TAGET            PRTY={(R0)|(reg)|n}
                            ,EOF={(reg)|addr}
                           [,SAVE={YES|NO}]
                           [,NORMAL={(reg)|addr}]
```

Operands

PRTY A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

EOF The location to which the Table Get routine returns when the Job Control Table contains no next entry.

SAVE Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL The location to which the Table Get Routine returns when it has accessed the next entry in the Job Control Table. Register 1 contains the address of the entry, and register 0 contains the size of the entry, represented in binary.

TAJENTRY

Functional Description

The TAJENTRY macro-instruction defines a work area for each priority level within the Job Control Table. It is used by the JCT table access routines and by JSS for scheduling purposes.

```
Name          Operation       Operands

              TAJENTRY
```

Operands

This macro-instruction contains no operands.

TAPUT

Functional Description

The TAPUT macro-instruction is used to write the current Job Control Table record in the chain on the direct access storage device.

```
Name          Operation       Operands

[symbol]      TAPUT             PRTY={(R0)|(reg)|n}
                                [,SAVE={YES|NO}]
                                [,NORMAL={(reg)|addr}]
```

Operands

PRTY — A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the Table Put routine returns when the current Job Control Table record has been written on the direct access storage device.

TARESET

Functional Description

The TARESET macro-instruction resets the Job Control Table scan pointers to the top of the JCT queue for the priority level specified.

Name	Operation	Operands
[symbol]	TARESET	PRTY={(R0)|(reg)|n} [,SAVE={YES|NO}] [,NORMAL={(reg)|addr}]

Operands

PRTY — A number that specifies the job priority associated with the Job Control Table entries in the range 0 to 15. If the number is in a register, it is represented in binary. If the number is specified absolutely, it is represented as a decimal number.

SAVE — Indicates whether the contents of registers 2 through 9 are to be saved across the macro call.

NORMAL — The location to which the Table Reset routine returns when the scan pointers have been reset.

TATPARMS

Functional Description

The TATPARMS macro-instruction establishes a DSECT, used by ASPIO, that defines a table of track allocator parameters.

Name	Operation	Operands
	TATPARMS	

Operands

This macro-instruction contains no operands.

TPCDSECT

Functional Description

The TPCDSECT macro-instruction defines the Tape-to-Printer checkpoint buffer.

Name	Operation	Operands
	TPCDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

TPRDSECT

Functional Description

The TPRDSECT macro-instruction defines the data control section required by the Tape-to-Printer DSP for each printer attached to the ASP system.

Name	Operation	Operands
	TPRDSECT	[TYPE={CSECT|DSECT}]

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

TRACE

Functional Description

The TRACE macro-instruction allows the user to perform a specified form of trace.

```
Name          Operation       Operands

[symbol]      TRACE             TYPE={C|D}
                                [,NORMAL={(reg)|addr}]
```

Operands

TYPE — A character identifying the type of trace required. Registers 0 through 4 must be preloaded with the data to be traced. (See Chapter 9, Debugging Aids in ASP for an explanation of the available types of trace.)

NORMAL — The location to which the TRACE routine returns when it has completed the trace.

TVTABLE

Functional Description

The TVTABLE macro-instruction is used to define a Transfer Vector Table. The Transfer Vector Table allows non-resident routines, which are loaded into core storage by the ALOAD macro-instruction, to branch to resident routines.

The table also contains pointers to several systems tables and various data, or pointers to data, shared by two or more functions within the ASP system.

```
Name          Operation       Operands

              TVTABLE         [TYPE={CSECT|DSECT}]
```

Operands

TYPE — CSECT establishes a real control section data area. DSECT establishes a dummy control section data area.

Register 12 is loaded with the location of the Transfer Vector Table. All routines that define the table as a dummy control section use register 12 as the beginning location of the table.

UCBENTRY

Functional Description

The UCBENTRY macro-instruction is furnished to define the fields of the OS Unit Control Block and to provide labels for frequently referenced fields therein.

Name	Operation	Operands
	UCBENTRY	

Operands

This macro-instruction contains no operands.

VIOLATE

Functional Description

The VIOLATE macro-instruction is used to execute instructions which violate storage protection, such as setting the not-ready bit in the Unit Control Block. It may also be used to alter the setting of the system mask and protection key.

Name	Operation	Operands
[symbol]	VIOLATE	[ENTER={(R1)|(reg)|addr}] [,SSM={ON|OFF}] [,KEY={ASP|ZERO}]

Operands

ENTER — The address of an appendage containing the instructions that violate storage protection. The appendage may not contain any supervisor call or other status-switching instructions and must be terminated by a return on register 14. If omitted, only the protect facility is invoked.

SSM — Indicates whether the system mask is to be enabled (ON) or disabled (OFF). If neither ENTER nor KEY is specified, omission of SSM causes KEY=ASP to be assumed. Specification of SSM as the only parameter leaves the protection key unchanged.

KEY — Indicates whether the protection key is to be set to the key of the ASP region or to zero. If ENTER is specified, KEY=ZERO is assumed and KEY=ASP is invalid. If ENTER is omitted, KEY=ASP is assumed but either specification is valid.

VUTDSECT

Functional Description

The VUTDSECT macro-instruction defines the fields of the volume-unavailable table, which is used by MDSALLOC to determine whether a volume is available for setup. The entries are maintained by the MODIFY command.

```
Name          Operation       Operands

              VUTDSECT
```

Operands

This macro-instruction contains no operands.

WRTCHAIN

Functional Description

The WRTCHAIN macro-instruction is used to write a group of chained single-record files.

```
Name          Operation       Operands

[symbol]      WRTCHAIN          FDB={(R1)|(reg)|addr}
                               ,TATPTR={(R0)|(reg)|addr}
                               ,DISP={(R2)|(reg)|addr}
                              [,PUTBUF={YES|NO}]
                              [,SAVE={YES|NO}]
                              [,NORMAL={(reg)|addr}]
```

Operands

FDB — The File Description Block address of the first single-record file in the chain.

TATPTR — The FDB address of the JBTAT FDB to be used. If the single-record files obtain their tracks from the Single Track Table (STT), FDB=MNTRKFDB should be used. When the single-record file chain consists of records belonging to the entire job's allocation, such as a JST or JDS, the JBTAT FDB in the AJDB should be used.

DISP — The displacement into the single-record file of the chain FDB.

PUTBUF — If PUTBUF=NO is specified, no buffers are returned.

SAVE — Indicates whether the contents of registers 3 through 9 are to be saved across the macro call.

NORMAL — The location to which the WRTCHAIN routine returns control when all records in the chain have been written.

WTDSECT

Functional Description

The WTDSECT macro-instruction establishes a DSECT, used by the Work-to-Do-Driver (WTDDRVR), that defines fields and flags of a work-to-do table.

```
Name          Operation       Operands

              WTDSECT
```

Operands

This macro-instruction contains no operands.

ZCALL

Functional Description

ZCALL is an inner macro which generates a calling sequence and/or return points.

```
Name          Operation       Operands

[symbol]      ZCALL            [tvadd]
                              [,SAVE=&SAVE]
                              [,EOD=&EOD]
                              [,EOF=&EOF]
                              [,NAVAIL=&NAVAIL]
                              [,ERROR=&ERROR]
                              [,IPL=&IPL]
                              [,BUSY=&BUSY]
                              [,REJECT=&REJECT]
                              [,RJPCAN=&RJPCAN]
                              [,NORMAL=&NORMAL]
```

Operands

tvadd	Indicates by its presence that a calling sequence is to be generated. It is the label of the address constant (usually found in TVTABLE) of the routine to be branched to. If the user of the outer macro supplied a label and the outer macro passes that label to ZCALL, it is attached to the first generated instruction.
SAVE	Specifies whether the user's registers 2 through 9 are to be saved across the execution of the called routine. This parameter is normally passed from the user's outer macro specification.
EOD,EOF, NAVAIL, ERROR,IPL, BUSY, REJECT, RJPCAN, NORMAL	Return points from the called routine, generated in the order shown, if present. If no specification is given by the user and the return point is required by the outer macro, the outer macro passes an asterisk as the specification to generate an error message in the assembly. Each return point may be specified in register form or as the label of the return point. Register notation, however, may not be used with SAVE=NO, and is otherwise restricted to registers 2 through 9 or 13.

ZEROCORE

Functional Description

The ZEROCORE macro-instruction, used by ASPIO, clears an area of core to binary zeros.

Name	Operation	Operands
[symbol]	ZEROCORE	AREA={(R3)|(reg)|addr} ,COUNT={(R4)|(reg)|n}

Operands

AREA — The address of the area to be cleared.

COUNT — The number of bytes to be cleared.

ZLOAD

Functional Description

ZLOAD is an inner macro which generates a load instruction, if required, to the specifications passed by the outer macro. If the target register and the input register are identical, no LR instruct is generated. If the user of the outer macro supplied a label and th outer macro passes that label to ZLOAD, it is attached to the load instruction, or a DS 0H is generated for the label if no load instruction is generated. The notations (n) and (Rn) are considered equivalent in all cases.

Name	Operation	Operands
[symbol]	ZLOAD	(reg),key,&key[,load-type]

Operands

(reg) — The register which the outer macro wishes to have loaded.

key — The keyword of the parameter the outer macro is specifying. This term may be used in error messages.

&key — The source location for the load instruction, as specified by the user of the outer macro. If this value is null, an error message is issued.

load-type — Determines the type of load instruction:

Load-type:	Generates:	If:
LA or omitted	LR	&key is (n) or (Rn)
	LA	&key is numeric
	LA	All other

RLA	LA	&key is (n) or (Rn) and n is not 0
	LR+N	&key is (n) or (Rn) and n is 0
	LA	&key is numeric
	LA	All other
L	LR	&key is (n) or (Rn)
	LA	&key is numeric
	L	All other

ZMNOTE

Functional Description

ZMNOTE is an inner macro which issues one of a set of MNOTEs to indicate a user specification error during macro generation. The outer macro issues ZMNOTE to assure consistency in text and severity level specification.

Name	Operation	Operands
	ZMNOTE	number[,param1][,param2][,param3][,param4]

Operands

number — The number of the MNOTE desired in the list.

param1,2,3,4 — The text of the inserts required for the chosen message. For example, if the user has given an incorrect specification, the outer macro issues ZMNOTE 2,KEY,&KEY to generate message 2: MNOTE 7,'ERROR,"KEY=specification" ILLEGAL SPECIFICATION'.

ZTYPE

Functional Description

ZTYPE is an inner macro utilized by outer macros with the TYPE={CSECT|DSECT} parameter to assure consistent treatment thereof.

Name	Operation	Operands
	ZTYPE	&TYPE,name

Operands

&TYPE — The type of control section specified by the user of the outer macro. If CSECT, a CSECT is generated. If DSECT or null (omitted), a DSECT is generated. Any other specification results in a warning MNOTE with a default generation of DSECT.

name — The label to be attached to the CSECT or DSECT statement.

APPENDIX B: ASP NUCLEUS MODULES

This appendix lists the modules, and approximate size of each, that are contained in the nucleus.

Module: name ASPNUC

entry INITIATE

size 49096

CONTROL SECTION

Name	Length
IONUC	3900
JOBCONTL	4130
CONSCONS	1260
ASPABEND	2390
ALDADEL	870
ASPCKPT	300
CKPTDATA	820
ASPCONTL	3142
ASPOPENX	880
CALLDRVR	540
CONSOLES	1640
CONSQMGR	2220
GETPUTMN	620
GETPUTUN	2150
INITIATE	1360
INTCOM	570
IODATA	3330
JOBNUM	610
JSSDR	1400
LOGINOUT	230
RESPARAM	3860
TVTABLE	1380
ABENDMCN	1900
AHIO	190
ASAVERTN	1280
CONSINPT	3190
TRACKS	1690
IORTNS	860
CONSAUTH	20
NETCONTL	1470
ADEQUEUE	590
WTDDRVR	280

APPENDIX C: RESIDENT MODULE REQUIREMENTS

This appendix lists the modules, and approximate size, that are required to be resident.

READER/INTERPRETER MODULES RESIDENT IN MAIN STORAGE

Module Name	Alias	Entry	Size (dec)	Alias For
IEFHRFK2	YES		-	IEFMVTHR
IEFMVTHR			1224	
IEFMVTJA			6192	
IEFVGM1-19			5016	
IEFVGM70-71			480	
IEFVGM78			248	
IEFVHA		YES	34688	
IEFVHCB	YES		-	IEFVHA
IEFVHF	YES		-	IEFVHA
IEFVHREP	YES		-	IEFVHA
IEFVINA			6040	
IEFVJA	YES	YES		IEFMVTJA
IEZDCODE			208	
TOTAL			54096	

APPENDIX D: PROGRAM MODULES OF THE ASP SYSTEM

The following is a representative module list and can be used for storage requirement estimation.

DSP Name or Function	3.0 Size (Approximate)
ABEND	
ABENDMON	7D8
ASPABEND	960
ASPABNDA	668
ASPABNDB	4B8
ASPABNDC	4C8
ASPABNDD	440
ASPABND1	7D0
ASPABND2	608
ASPABND3	758
ASPABND4	760
ASPABND5	4C0
ASPABND6	640
ASPABND7	618
ASPABND8	738
ASPABND9	7A8
ASPDMPRT	3728
CONSOLE SERVICE	
CONSANAL	360
CONSAUTH	18
CONSCONS	4F0
CONSINPT	D48
CONSOLES	690
CONSQMQR	8D0
CONSRMT	388
CONS1052	170
CONS1053	110
CONS1403	228
CONS2260	280
CONS2740	638
CONS3060	3A0
CONS3066	3B8
CONS3277	510
CONS3284	1AD
CONTRAP	178
DEADLINE	
DEADLINE	2B0
DELINIT	268
DELTIME	4C8
DELWORK	F8
DEPENDENT JOB CONTROL	
DJCDATA	280
DJCPROC	98
DJCUPDAT	A28
NETCONTL	5C8
NETDATA	298

DSP Name or Function	3.0 Size
FAILSOFT	
AFSDRVR	2E8
AFSCDRV1	88
AFSCNSL1	30
AFSDC	1F8
AFSINIT	340
AFSRCVY	138
AFSRIC01	2F0
AFSTERM	108
INITIALIZE	
INITIATE	558
INITANAL	9A0
INITCARD	2218
INITCNS	F60
INITDATA	D00
INITGEN	CD0
INITIO	1220
INITIOCD	D38
INITJOB	4D8
INITMDS	AD0
INITMN1	13A8
INITMN2	1690
INITQUE	2C0
INITREST	C88
INITRI	558
INITRJP1	5E0
INITRJP2	8C8
INITRTNS	CC0
INPUT SERVICE	
ISLOGIC	770
ISDATA	368
ISDLN	7B0
ISDRVR	668
ISDTASET	318
ISENDTSK	B10
ISFORMAT	A70
ISJCLIN	5C0
ISJOBCRD	6F8
ISMAIN	DB8
ISNET	900
ISPROCES	6A8
ISSEQ	198
INQUIRY	
INQDRVR	900
INQACSR	9F8
INQBACK	350
INQCONS	3A8
INQDISP	890
INQDJC	460
INQDLN	298
INQMDS	A40
INQQUE	470
INQRJP	748

DSP Name or Function	3.0 Size
INTERNAL JOB PROCESSING	
IJP	518
IJPDATA	500
IJPEND	7F8
IJPINISH	348
IJPSTART	1F8
IJPWTR	E18
MAIN DEVICE SCHEDULING	
MDSALLOC	F40
MDSBRKDN	368
MDSDATA	1C0
MDSDRIVR	760
MDSREST	360
MDSVERFY	5B0
MAIN SERVICE	
MAIN	DA8
MAINIO	DA8
MPCDATA	3A0
MPDDATA	468
MSVCBUP	658
MSVDATA	68
MSVDUMMY	C4D
MSVLOCAL	509
MSVINIT	598
MSVIPL	1240
MSVMVT	10E8
MSVOPER1	5F0
MSVOPER2	B90
MSVQMAP	870
MSVTERM	4D8
MAINTASK	
ADSGEN	1B68
ADSGEN1	F0
AOUTPUT	17D0
ASPCTCM	318
ASPFENCE	1760
ASPLOC	6D0
ASPQALL	3C8
ASPQRDR	490
ASPSVC	2B0
ASPVER	FC0
ASPWRITR	5C8
ASUBMIT	DA0
DYNDISP	410
MAINTASK	C48
MODIFY	
MODACC	1A0
MODASG	6F8
MODCNPR	538
MODCONS	5C0
MODDJC	5E8

DSP Name or Function	3.0 Size
MODDLN	4B8
MODDRVR	858
MODHRQ	428
MODMDS	B20
MODRJP	6F8
NETWORK JOB PROCESSING	
NJP	178
NJPCOMM	1880
NJPDATA	1B8
NJPDJ	8E8
NJPINQ	600
NJPIO	B48
NJPOPEN	4A0
NJODATA	1F0
PRINT SERVICE	
PRINT	128
PRTDATA	478
PRTERROR	348
PRTINISH	4B8
PRTOUT	11C8
PRTSETUP	1688
PRTTERM	8F0
PUNCH SERVICE	
PUNCH	1020
PCHDATA	300
PURGE	
PURGE	8D0
PURDATA	190
READER/ INTERPRETER	
IEFQDELE	24
RIATTCH	428
RICBAM	3A0
RICBSCAN	12D7
RICONTL	CF2
RIDATA	398
RIEXITS	9BC
RIFETCH	B38
RISERVX	500
Readers	
CR	420
DR	280
TR	4C0
RDDATA	2C0
RDINISH	D00
RDLOGIC	EC0
RDOPARMS	780

DSP Name or Function	3.0 Size
REMOTE JOB PROCESSING	
RJP	1F0
RJPMAIN1	D20
RJPMAIN2	FD0
RJPMAIN3	10A0
RJPMAIN4	B50
RJPMAIN5	80
RJPMAIN6	578
OS Modules	
IGG0190$	400
IFG0192Z	400
IKJEFFHR	4C8
IKJEFFR0	A78
IKJEFF53	1F8
Service Routines	
CALLDSP	678
IOERREC	670
JSS	1630
TRCERTN	50
VARY	B30
WTDJCT	1648
UCS Loads	
AN	1F0
AN11	200
GN11	200
HN	1F0
HN11	200
PCSAN	1F0
PCSHN	1F0
PN	1F0
PN11	200
QN	1F0
QNC	1F0
RN	1F0
SN	1F0
TN	1F0
TN11	200
XN	1F0
YN	1F0
Utilities	
ACCPR	1257
ACDS	C88
ACDATA	4B0
ALOADS	C10
ASPNEWS	1D0
CBPRINT	1330
CC	10F0
CNT	670
CP	AB0
CT	10A0
DC	1895

DSP Name or Function	3.0 Size
DISPDJC	BD8
DISPLAY	12C0
DJ	E58
DJDATA	528
DJIN	948
DJNETOUT	2C8
DJOUT	1200
PRUT	C58
TC	FC8
TD	1CC8
TL	EF8
TP	D8
TPRDATA	308
TPRERROR	1D0
TPRINISH	CF8
TPROUT	F90
TPRTERM	48
TT	10D0

APPENDIX E: MULTILEAVING

"Multileaving" is a term that describes a computer-to-computer communication technique developed for use by the HASP system and used by ASP RJP. In a gross sense, multileaving can be defined as the fully synchronized, pseudo-simultaneous, bidirectional transmission of a variable number of data streams between two or more computers using Binary Synchronous Communications facilities.

MULTILEAVING PHILOSOPHY

The basic element for multileaved transmission is the character string. One or more character strings are formed from the smallest external element of transmission, the physical record. These physical records are input to multileaving and may be any of the classic record types (card images, printed lines, tape records, etc.). For efficiency in transmission, each of these data records is reduced to a series of character strings of two basic types:

1. A variable-length nonidentical series of characters

2. A variable number of identical characters

An eight-bit control field, termed a String Control Byte (SCB), precedes each character string to identify the type and length of the string. Thus, a string as in 1 above is represented by an SCB followed by the nonduplicate characters. A string of consecutive, duplicate, nonblank characters (as in 2 above) can be represented by an SCB and a single character (the SCB indicates the duplication count, and the character following indicates the character to be duplicated). In the case of an all-blank character string, only an SCB is required to indicate both the type and the number of blank characters. A data record to be transmitted is segmented into the optimum number of character strings (to take full advantage of the identical character compression) by the transmitting program. A special SCB is used to indicate the grouping of character strings that compose the original physical record. The receiving program can then reconstruct the original record for processing.

In order to allow multiple physical records of various types to be grouped together in a single transmission block, an additional eight-bit control field precedes the group of character strings representing the original physical record. This field, the Record Control Byte (RCB), identifies the general type and function of the physical record (input stream, print stream, data set, etc.). A particular RCB type has been designated to allow the passage of control information between the various systems. Also, to provide for simultaneous transmission of similar functions (that is, multiple input streams, etc.), a stream identification code is included in the RCB. A second eight-bit control field, the Sub-Record Control Byte (SRCB), is also included immediately following the RCB. This field is used to supply additional information concerning the record to the receiving program. For example, in the transmission of data to be printed, the SRCB can be used for carriage control information.

For actual multileaving transmission, a variable number of records may be combined into a variable block size, as indicated previously (that is, RCB,SRCB,SCB1,SCB2,...,SCBn,RCB,SRCB,SCB1,...,etc.). The multileaving design provides for two (or more) computers to exchange transmission blocks, containing multiple data streams as described

above, in an interleaved fashion. To allow optimum use of this capability, however, a system must have the capability to control the flow of a particular data stream while continuing normal transmission of all others. This requirement becomes obvious if one considers the case of the simultaneous transmission of two data streams to a system for immediate transcription to physical I/O devices of different speeds (such as two print streams). To provide for the metering of the flow of individual data streams, a Function Control Sequence (FCS) is added to each transmission block. The FCS is a sequence of bits, each of which represents a particular transmission stream. The receiver of several data streams can temporarily stop the transmission of a particular stream by setting the corresponding FCS bit off in the next transmission to the sender of that stream. The stream can subsequently be resumed by setting the bit on.

Finally, for error detection and correction purposes, a Block Control Byte (BCB) is added as the first character of each block transmitted. The BCB, in addition to control information, contains a modulo 16 block sequence count. This count is maintained and verified by both the sending and receiving systems to exercise a positive control over lost or duplicated transmission blocks.

In addition to the normal binary synchronous text control characters (STX, ETB, etc.), multileaving uses two of the BSC control characters, ACK0 and NAK. ACK0 is used as a "filler" by all systems to maintain communications when data is not available for transmission. NAK is used as the only negative response and indicates that the previous transmission was not successfully received.

A typical multileaving transmission block looks like this:

DLE	BSC Leader (SOH if no transparency feature)
STX	BSC Start-of-Text
BCB	Block Control Byte
FCS	Function Control Sequence
FCS	Function Control Sequence
RCB	Record Control Byte for record 1
SRCB	Sub-Record Control Byte for record 1
SCB	String Control Byte for record 1
DATA	Character String
SCB	String Control Byte for record 1
DATA	Character String
SCB	Terminating SCB for record 1
RCB	RCB for record 2
SRCB	SRCB for record 2
SCB	SCB for record 2
DATA	Character String
SCB	Terminating SCB for record 2
RCB	Transmission Block Terminator
DLE	BSC Leader (SYN if no transparency feature)
ETB	BSC Ending Sequence

MULTILEAVING CONTROL SPECIFICATION

This section describes the bit-by-bit definitions of the various multileaving control fields and includes notes concerning their use.

Record Control Byte (RCB)

O	I	I	I	T	T	T	T

0 7

Usage:	To identify each record type within a transmission block		
Bits:	OIIITTTT	00000000	End of transmission block
	or:		
	O	1	Non-EOT RCB
	III0000		III is control information:
	III	000	Reserved
		001	Request to initiate a function transmission (prototype RCB for function in SRCB)
		010	Permission to initiate a function transmission (RCB for function contained in SRCB)
		011	Reserved
		100	Reserved
		101	Available for location modification
		111	General Control Record (type indicated in SRCB)
	or:		
	O	1	Non-EOT RCB
	LIITTTT		III is used to identify streams of muntiple identical functions (such as multiple print streams to a multiple printer terminal). TTTT is the record type identifier.
	TTTT	0001	Operator message display request
		0010	Operator command
		0011	Normal input record
		0100	Print record
		0101	Punch record
		0110	Data set record
		0111	Terminal message routing request
		1000-1100	Reserved
		1101-1111	Available to user

Sub-Record Control Byte (SRCB)

Usage: To provide supplemental information about a record

Bits: The contents of this control block depend upon the record type. Several types are shown below.

SCRB for General Control Record

---character...
0 7

Usage: To identify the type of generalized control record

Bits:	character	A	Initial terminal sign-on
		B	Final terminal sign-off
		C	Print initialization record
		D	Punch intialization record
		E	Input intialization record
		F	Data set transmission initialization
		G	System configuration status
		H	Diagnostic control record
		I-R	Reserved
		S-Z	Available to user

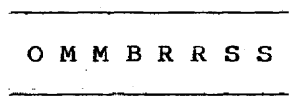
SRCB for Print Records

O M C C C C C C
0 7

Usage: To provide carriage control information for print records

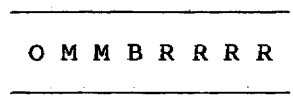
Bits:	O	1	
	M	0	Normal carriage control
		1	Reserved
	CCCCCC	000000	Suppress space
		0000nn	Space nn lines after print
		01nnnn	Skip to channel nnnn after print
		1000nn	Space immediate nn spaces
		11nnnn	Skip immediate to channel nnnn

SRCB for Punch Records



Usage:	To provide additional information for punch records		
Bits:	O	1	
	MM	00	SCB count units = 1
		01	SCB count units = 2
		10	SCB count units = 4
		11	Reserved
	B	0	EBCDIC card image
		1	Column Binary card image
	RR	00	Reserved
	SS	nn	Stacker select information

SRCB for Input Record



Usage:	To provide additional information for input records		
Bits:	O	1	
	MM	00	SCB count units = 1
		01	SCB count units = 2
		10	SCB count units = 4
		11	Reserved
	B	0	EBCDIC card image
		1	Column Binary card image
	RRRR	0000	Reserved

SRCB for Terminal Message Routing Record

O	T	T	T	T	T	T	T

0 7

Usage: To indicate the destination of a terminal message

Bits:			
	O	1	
	TTTTTTT	0000000	Broadcast to all remote systems
		nnnnnnn	Remote system number (1-99) or remote system group as HASPGENed (100-127)

String Control Byte (SCB)

O	K	L	J	J	J	J	J

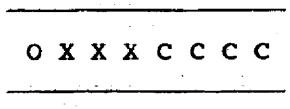
0 7

Usage: Control field for data character strings

Bits:			
	OKLJJJJJ	00000000	End of record
	or:		
	OKLJJJJJ	10000000	Record is continued in next transmission block
	or:		
	O	1	Non-EOR SCB
	K	0	Duplicate character string
	L	0	Duplicate character is blank
		1	Duplicate character is nonblank and follows SCB
	JJJJJ	nnnnn	Duplicate count
	or:		
	O	1	Non-EOR SCB
	K	1	Nonduplicate character string
	LJJJJJ	nnnnn	Character string length

Note: Count units are normally 1 but may be in any other units. The units used may be indicated at function control sign-on or dynamically in the SRCB.

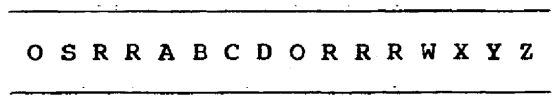
Block Control Byte (BCB)



Usage: Transmission block status and sequence count

Bits:	O	1	
	XXX	000	Reserved
		001	Bypass sequence count validation
		010	Reset expected block sequence count to CCCC
		011	Reserved
		100	Reserved
		101	Available to user
		110	Available to user
		111	Reserved
	CCCC	nnnn	Modulo 16 block sequence count

Function Control Sequence (FCS)



Usage: To control the flow of individual function streams

Bits:	O...O	1...1	
	S	0	Normal processing
		1	Suspend all stream transmission (Wait-a-Bit)
	RR...RRR	00...000	Reserved
	ABCD	nnnn	Print or input stream identification
	WXYZ	nnnn	Punch stream identifiers

Note: These function stream identifiers are oriented only to the recipient. Presence of a bit indicates that function transmission is to be continued; its absence indicates that function transmission is to be suspended.

MULTILEAVING IN BSC/RJP

The previous sections have grossly outlined the specifications of a comprehensive, multileaving communications system. While the HASP/ASP support for programmable BSC workstations is completely consistent with the multileaving design, it does not use certain of the features provided in multileaving:

- The transmission of record types other than print, punch, input, console, and control is not supported.
- The only general control record type used is the terminal sign-on control.
- Only SCB count units of 1 are used.
- No support is included for column binary cards.

INDEX

READER'S COMMENT FORM

GH20-1292-0

System/360 and System/370
ASP Version 3 Asymmetric Multiprocessing System
System Programmer's Manual

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold fold

fold fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.

GH20-1292-0

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 813
1133 Westchester Avenue
White Plains, New York 10604

Fold Fold

S/360 and S/370 Version 3 (ASP) SPM Printed in U.S.A. GH20-1292-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)