360 INTRODUCTORY PROGRAMMING

360PGM

STUDENT WORKBOOK

CAI #48

REVISION 1
8/25/70

INDEX

PAGE	TITLE
1-1	FIXED-POINT INST.
1-8	LOGICAL INST.
1-16	BRANCHING INST.
1-26	STATUS SWITCHING
3-1	LOAD
3-9	STORE
3-13	ADD
3-18	SUBTRACT
3-23	BRANCH ON CONDITION
3-25	COMPARE
3-28	MULTIPLY
3-31	DIVIDE
3-34	SHIFT
3-41	FIXED-POINT DATA FORMATS (PACK AND UNPACK)
3-45	CONVERT TO BINARY
3-48	CONVERT TO DECIMAL
4-1	MOVE
4-6	AND
4-11	OR
4-21	TEST UNDER MASK
4-23	COMPARE LOGICAL
4-27	SHIFT LOGICAL
4-32	LOAD ADDRESS
4-34	TRANSLATE

PAGE	TITLE
4-37	TRANSLATE AND TEST
4-41	EXECUTE
4-44	INSERT CHARACTER
4-46	STORE CHARACTER
5-1	BRANCH ON COUNT
5-4	BRANCH ON INDEX
5-8	BRANCH AND LINK
5-11	SUPERVISOR CALL
5-13	SET PROGRAM MASK
5-15	SET SYSTEM MASK
5-17	LOAD PSW
5-19	STORAGE KEYS (SSK AND ISK)
5-23	TEST AND SET
9-1	DECIMAL INST.
9-8	ADD DECIMAL
9-11	ZERO AND ADD
9-14	SUBTRACT DECIMAL
9-17	MULTIPLY DECIMAL
9-19	DIVIDE DECIMAL
9-22	COMPARE DECIMAL
9-24	EDIT
9-41	EDIT AND MARK

PAGE	TITLE
11-1	I/O INST.
11-16	START I/O
11-19	TEST I/O
11-21	HALT I/O
12-1	I/O PROGRAMMING PROJECT

FIXED-POINT INSTRUCTIONS

FIXED-POINT

The fixed-point instruction set performs binary arithmetic on operands serving as addresses, index quantities, and counts, as well as fixed-point data. In general, both operands are signed and 32 bits long. Negative quantities are held in two's-complement form. One operand is always in one of the 16 general registers; the other operand may be in main storage or in a general register.

The instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as for the sign control, radix (base) conversion, and shifting of fixed-point operands. The entire instruction set is included in the standard instruction set.

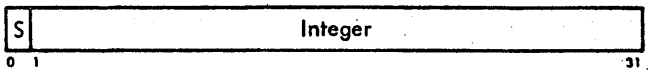
The condition code is set as a result of all sign control, add, subtract, compare, and shift operations.

DATA FORMAT

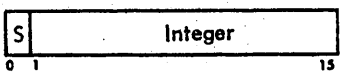
Fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the integer field. When held in one of the general registers, a fixed-point quantity has a 31-bit integer field and occupies all 32 bits of the register. Some multiply, divide, and shift operations use an operand consisting of 64 bits with a 63-bit integer field. These operands are located in a pair of adjacent general registers and are addressed by an even address referring to the leftmost register of the pair. The sign-bit position of the rightmost register contains

of the integer. In register-to-register operations the same register may be specified for both operand locations.

Full Word Fixed-Point Number



Halfword Fixed-Point Number



Fixed-point data in main storage occupy a 32-bit word or a 16-bit halfword, with a binary integer field of 31 or 15 bits, respectively. The conversion instructions use a 64-bit decimal field. These data must be located on integral storage boundaries for these units of information, that is, double word, fullword, or halfword operands must be addressed with three, two, or one low-order address bit(s) set to zero.

A halfword operand in main storage is extended to a fullword as the operand is fetched from storage. Subsequently, the operand participates as a fullword operand.

In all discussions of fixed-point numbers in this publication, the expression "32-bit signed integer" denotes a 31-bit integer with a sign bit, and the expression "64-bit signed integer" denotes a 63-bit integer with a sign bit.

NUMBER REPRESENTATION

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary notation

with the sign bit set to zero. Negative numbers are represented in two's-complement notation with a one in the sign bit.

Two's-complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for sign.

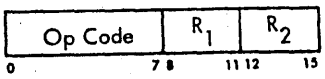
The CPU cannot represent the complement of the maximum negative number. When an operation, such as subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed-point overflow exception is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

The sign bit is leftmost in a number. In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out of the integer field.

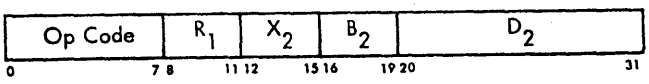
INSTRUCTION FORMAT

Fixed-point instructions use the following three formats:

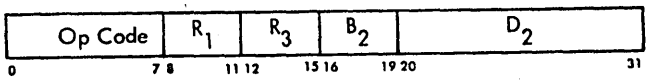
RR Format



RX Format



RS Format



In these formats, R1 specifies the general register containing the first operand. The second operand location, if any, is defined differently for each format.

In the RR format, the R2 field specifies the general register containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general registers specified by the X2 and B2 fields are added to the content of the D2 field to form an address designating the storage location of the second operand.

In the RS format, the content of the general register specified by the B2 field is added to the content of the D2 field. This sum designates the storage location of the second operand in LOAD MULTIPLE and STORE MULTIPLE. In the shift operations, the sum specifies the number of bits of the shift. The R3 field specifies the address of a general register in LOAD MULTIPLE and STORE MULTIPLE and is ignored in the shift operations.

A zero in an X2 or B2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed before operation execution.

Results replace the first operand, except for STORE and CONVERT TO DECIMAL, where the result replaces the second operand.

The contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembley language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and R1, R2 the operand designation.

FIXED-POINT PROGRAM INTERRUPTIONS

Exceptional operand designations, data, or results cause a program interruption. When a program interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in fixed-point arithmetic.

Protection: The key of an operand in storage does not match the protection key in the PSW. The operation is suppressed for a store violation. Therefore, the condition code and data in registers and storage remain unchanged. The only exception is STORE MULTIPLE, which is terminated; the amount of data stored is unpredictable and should not be used for further computation. The operation is terminated on any fetch violation.

Addressing: An address designates an operand location outside the available storage for a particular installation. In most cases, the operation is terminated. Therefore, the result data are unpredictable and should not be used for further computation. The exceptions are STORE, STORE HALFWORD, and CONVERT TO DECIMAL, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the components from which an address is generated - the content of the D2 field and the contents of the registers specified by X2 and B2.

Specification: A double-word operand is not located on a 64-bit boundary, a fullword operand is not located on a 32-bit boundary, a halfword operand is not located on a 16-bit boundary, or an instruction specifies an odd register address for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Data: A sign or a digit code of the decimal operand in CONVERT TO BINARY is incorrect. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Fixed-Point Overflow: The result of a sign-control add, subtract, or shift operation overflows. The interruption occurs only when the fixed-point overflow mask bit is one. The operation is completed by placing the truncated low-order result in the register and setting the condition code to 3. The overflow bits are lost. In add-type operations the sign stored in the register is the opposite of the sign of the sum or difference. In shift operations the sign of the shifted number remains unchanged. The state of the mask bit does not affect the result.

Fixed-Point Divide: The quotient of a division exceeds the register size, including division by zero, or the result in CONVERT TO BINARY exceeds 31 bits. Division is suppressed. Therefore, data in the registers remain unchanged. The conversion is completed by recording the truncated low-order result in the register.

LOGICAL INSTRUCTIONS

A set of instructions is provided for the logical manipulation of data. Generally, the operands are treated as eight-bit bytes. In a few cases the left or right four bits of a byte are treated separately or operands are shifted a bit at a time. The operands are either in storage or in general registers. Some operands are introduced from the instruction stream.

Processing of data in storage proceeds left to right through fields which may start at any byte position. In the general registers, the processing, as a rule, involves the entire register contents.

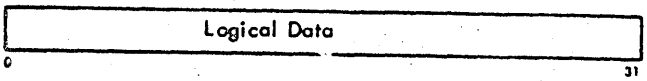
The set of logical operations includes moving, comparing, bit testing, translating, and shift operations. All logical operations are part of the standard instruction set.

The condition code is set as a result of all logical comparing, connecting, testing, and editing operations.

DATA FORMAT

Data reside in general registers or in storage or are introduced from the instruction stream. The data size may be a single or doubleword, a single character, or variable length. When two operands participate they have equal length.

Fixed-Length Logical Information

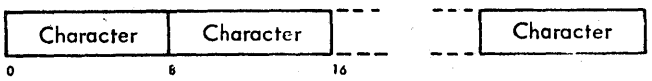


Data in general registers normally occupy all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of a register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair of registers participate.

The LOAD ADDRESS introduces a 24-bit address into a general register. The high-order eight bits of the register are made zero.

In storage-to-register operations, the storage data occupy either a word of 32 bits or a byte of eight bits. The word must be located on word boundaries, that is, its address must have the two low-order bits zero.

Variable-Length Logical Information



In storage-to-storage operations, data have a variable field-length format, starting at any byte address and continuing for up to a total of 256 bytes. Processing is left to right.

Operations introducing data from the instruction stream into into storage, as immediate data, are restricted to an eight-bit byte. Only one byte is introduced from the instruction stream, and only one byte in storage participates.

Use of general register 1 is implied in TRANSLATE AND TEST. A 24-bit address may be placed in this register during operation. The TRANSLATE AND TEST also implies general register 2. The low-order eight bits of register 2 may be replaced by a function byte during a translate-and-test operation.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the content of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function.

A list is specified by an initial address - the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The actual address used to address the list is obtained by adding the argument to the low-order positions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. In cases where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

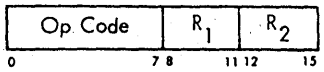
In a storage-to-storage operation, the operand fields may be defined in such a way that they overlap. The effect of this overlap depends upon the operation. When the operands remain unchanged, as in the COMPARE or TRANSLATE AND TEST, overlapping does not affect the execution of the operation.

In the case of MOVE, and TRANSLATE, one operand is replaced by new data, and the execution of the operation may be affected by the amount of overlap and the manner in which data are fetched or stored. For purposes of evaluating the effect of overlapped operands, consider that data are handled one eight-bit byte at a time. All overlapping fields are considered valid.

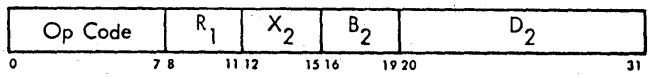
INSTRUCTION FORMAT

Logical instructions use the following five formats:

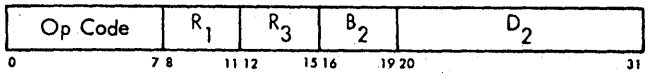
RR Format



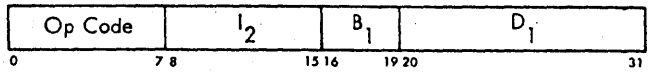
RX Format



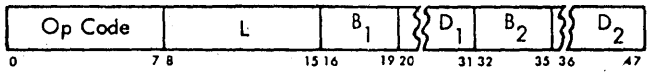
RS Format



SI Format



SS Format



In the RR, RX, and RS formats, the content of the register specified by R1 is called the first operand.

In the SI and SS formats, the contents of the general register specified by B1 is added to the content of the D1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the

right of this first byte is specified by the L field in the SS format. In the SI format the operand size is one byte.

In the RR format, the R2 field specifies the register containing the second operand. The same register may be specified for the first and second operand.

In the RX format, the contents of the general registers specified by the X2 and B2 fields are added to the content of the D2 field to form the address of the second operand.

In the RS format, used for shift operations, the contents of the general register specified by the B2 field is added to the content of the D2 field. This sum is not used as an address but specifies the number of bits of the shift. The R3 field is ignored in the shift operations.

In the SI format, the second operand is the eight-bit immediate data field, I2, of the instruction.

In the SS format, the content of the general register specified by B2 is added to the content of the D2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A zero in any of the X2, B1, or B2 fields indicates the absence of the corresponding address or shift-amount component. An instruction can specify the same general register both for

address modification and for operand location. Address modification is always completed prior to operation execution.

Results replace the first operand, except in STORE CHARACTER, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all general registers and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the result locations, general register 1 in EDIT AND MARK, and general registers 1 and 2 in TRANSLATE AND TEST.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the new IBM System/360 assembly language are shown with each instruction: For MOVE NUMERICS, for example, MVN is the mnemonic and D1 (L,B1), D2 (B2) the operand designation.

LOGICAL OPERATION EXCEPTIONS

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored in an old PSW and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in logical operations.

Protection: The key of an operand in storage does not match the protection key in the PSW. The operation is suppressed on a store violation. Therefore, the condition code and data in registers and storage remain unchanged. The only exceptions are the variable length, storage-to-storage operations (those containing a length specification), which are terminated. The operation is terminated on any fetch violation. For terminated operations, the result data and condition code, if affected, are unpredicatable and should not be used for further computation.

Addressing: An address designates an operand location outside the available storage for the installation: In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation. The exceptions are the immediate operations; AND (NI), EXCLUSIVE OR (XI), OR (OI) and MOVE (MVI) plus the STORE CHARACTER (STC), which are suppressed.

Specification: A fullword operand in a storage-to-register operation is not located on a 32-bit boundary or an odd register address is specified for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Data: A digit code of the second operand in EDIT or EDIT AND MARK is invalid. The operation is terminated. The result data and the condition code are unpredictable and should not be used

for further computation.

Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of LOAD ADDRESS is not tested. The address restrictions do not apply to the components from which an address is generated - the contents of the D1 and D2 fields, and the contents of the registers specified by X2, B1, and B2.

BRANCHING INSTRUCTIONS

Instructions are performed by the central processing unit primarily in the sequential order of their locations. A departure from this normal sequential operation may occur when branching is performed. The branching instructions provide a means for making a two-way choice, to reference a subroutine, or to repeat a segment of coding, such as a loop.

Branching is performed by introducing a branch address as a new instruction address.

The branch address may be obtained from one of the general registers or it may be the address specified by the instruction. The branch address is independent of the updated instruction address.

The detailed operation of branching is determined by the condition code which is part of the program status word (PSW) or by the results in the general registers which are specified in the loop-closing operations.

During a branching operation, the rightmost half of the PSW, including the updated instruction address, may be stored before the instruction address is replaced by the branch address. The stored information may be used to link the new instruction sequence with the preceding sequence.

The instruction EXECUTE is grouped with the branching instructions. The branch address of EXECUTE designates a single in-instruction to be inserted in the instruction sequence. The updated instruction address normally is not changed in this operation, and only the instruction located at the branch address is executed.

All branching operations are provided in the standard instruction set.

NORMAL SEQUENTIAL OPERATION

Normally, operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction-address field of the PSW. The instruction address is increased by the number of bytes of the instruction to address the next instruction in sequence. This new instruction, replaces the previous contents of the instruction-address field in the PSW. The current instruction is executed, and the same steps are repeated, using the updated instruction address to fetch the next instruction.

Conceptually, an instruction is fetched from storage after the preceding operation is completed and before execution of the current operation.

A change in the sequential operation may be caused by branching, status switching, interruption, or manual intervention. Sequential operation is initiated and terminated from the system control panel.

Programming Note

It is possible to modify an instruction in storage by means of the immediately preceding instruction.

SEQUENTIAL OPERATION EXCEPTIONS

Exceptional instruction addresses or operation codes cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. (In this manual, part of the description of each class of instructions is a list of the program interruptions that may occur for these instructions.) The new PSW is not checked for exceptions when it becomes current. These checks occur when the next instruction is executed. The following program interruptions may occur in normal instruction sequencing, independently of the instruction performed.

Operation: An operation exception occurs when the CPU attempts to decode an operation code that is not assigned. The operation exception can be accompanied by an addressing or specification exception if the instruction class associated with the undefined operation has uniform requirements or operand designation. An instruction class is a group of instructions whose four leftmost bits are identical.

Protection: A protection exception occurs when an attempt is made to fetch an instruction halfword from a fetch-protected location. This error can occur when normal instruction sequencing goes from an unprotected region into a protected

region, or following a branching or load-PSW operation or an interruption.

Addressing: An addressing exception occurs when an instruction halfword is located outside the available storage for the particular installation.

Specification: A specification exception occurs when the instruction address in the PSW is odd. This odd address error can occur only after a branching or load PSW operation or after an interruption.

A specification exception will occur when the protection key is nonzero and the protection feature is not installed. This error can occur after a PSW is loaded or after an interruption.

In each case, the instruction is suppressed; therefore, the condition code and data in storage and register remain unchanged. The instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction length code in the old PSW.

Programming Notes

When a program interruption occurs, the current PSW is stored in the old PSW location. The instruction address stored as part of this old PSW is thus the updated instruction address, having been updated by the number of halfwords indicated in the instruction-length code of the same PSW. The interruption

code in this old PSW identifies the cause of the interruption and aids in the programmed interpretation of the old PSW.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

DECISION-MAKING

Branching may be conditional or unconditional. Unconditional branches replace the updated instruction address with the branch address. Conditional branches may use the branch address or may leave the updated instruction address unchanged. When branching takes place, the instruction is called successful; otherwise, it is called unsuccessful.

Whether a conditional branch is successful depends on the result of operations concurrent with the branch or preceding the branch. The former case is represented by BRANCH ON COUNT and the branch-on-index instructions. The latter case is represented by BRANCH ON CONDITION, which inspects the condition code that reflects the result of a previous arithmetic, logical, or I/O operation.

The condition code provides a means for data-depended decision-making. The code is inspected to qualify the execution of the condition-branch instructions. The code is set by some operations to reflect the result of the operation, independently of the previous setting of the code. The code remains unchanged for all other operations.

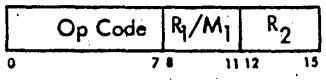
The condition code occupies bit positions 34 and 35 of the PSW. When the PSW is stored during status switching, the condition code is preserved as part of the PSW. Similarly, the condition code is stored as part of the rightmost half of the PSW in a branch-and-link operation. A new condition code is obtained by a LOAD PSW OR SET PROGRAM MASK or by the new PSW loaded as a result of an interruption.

The condition code indicates the outcome of some of the arithmetic, logical, or I/O operations. It is not changed for any branching operation, except for EXECUTE. In the case of EXECUTE, the condition code is set or left unchanged by the subject instruction, as would have been the case had the subject instruction been in the normal instruction stream.

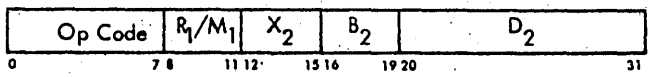
INSTRUCTION FORMATS

Branching instructions use the following three formats:

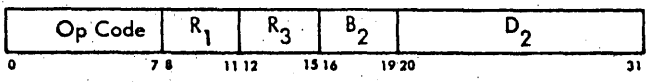
RR Format



RX Format



RS Format



In these formats R1 specifies the address of a general register. In BRANCH ON CONDITION a mask field (M1) identifies the bit values of the condition code. The branch address is defined differently for the three formats.

In the RR format, the R2 field specifies the address of a general register containing the branch address, except when R2 is zero, which indicates no branching. The same register may be specified by R1 and R2.

In the RX format, the contents of the general registers specified by the X2 and B2 fields are added to the content of the D2 field to form the branch address.

In the RS format, the content of the general register specified by the B2 field is added to the content of the D2 field to form the branch address. The R3 field in this format specifies the location of the second operand and

implies the location of the third operand. The first operand is specified by the R1 field. The third operation location is always odd. If the R3 field specifies and even register, the third operand is obtained from the next higher addressed register. If the R3 field specifies an odd register, the third operand location coincides with the second operand location.

A zero in a B2 or X2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register for both address modification and operand location. The order in which the contents of the general register are used for the different parts of an operation is:

1. Address computation.
2. Arithmetic or link information storage.
3. Replacement of the instruction address by the branch address obtained under step 1.

Results are placed in the general register specified by R1. Except for the storing of the final results, the contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for

the IBM System/360 assembly language are shown with each instruction. For BRANCH ON INDEX HIGH, for example, BXH is the mnemonic and R1, R3, D2(B2) the operand designation.

Programming Note

In several instructions the branch address may be specified in two ways: in the RX format, the branch address is the address specified by X2, B2, and D2; in the RR format, the branch address is in the low-order 24 bits of the register specified by R2. Note that the relation of the two formats in branch-address specification is not the same as in operand-address specification. For operands, the address specified by X2, B2, and D2 is the operand address, but the register specified by R2 contains the operand itself.

EXECUTE EXCEPTIONS

Exceptional operand designations and a subject-instruction operation code specifying EXECUTE cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause. Exceptions that cause a program interruption in the use of EXECUTE are:

Execute: An EXECUTE instruction has as its subject instruction another execute.

Protection: An EXECUTE specifies a subject instruction halfword in a fetch-protected area.

Addressing: The branch address of EXECUTE designates an instruction-halfword location outside the available storage for the particular installation.
Specification: The branch address of EXECUTE is odd.

These four execptions occur only for EXECUTE. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Exceptions arising for the subject instruction of EXECUTE are the same as would have arisen had the subject instruction been in the normal instruction stream. However, the instruction address stored in the old PSW is the address of the instruction following EXECUTE. Similarly, the instruction-length code in the old PSW is the instruction length code (2) of EXECUTE.

The address restrictions do not apply to the components from which an address is generated -- the content of the D1 field and the content of the register specified by B1.

Programming Note

An unavailable or odd branch address of a successful branch is detected during the execution of the next instruction and not as part of the branch.

STATUS SWITCHING INSTRUCTIONS

A set of operations is provided to switch the status of the CPU, of storage, and of communications between systems.

The overall CPU status is determined by several program-state alternatives, each of which can be changed independently to its opposite and most of which are indicated by a bit in the program status word (PSW). The CPU status is further defined by the instruction address, the condition code, the instruction-length code, the storage-protection key, and the interruption code. These all occupy fields in the PSW.

Protection of main storage is achieved by matching a key in storage with a protection key in the PSW or in a channel. The protection status of storage may be changed by introducing new storage keys, using SET STORAGE KEY. The storage keys may be inspected by using INSERT STORAGE KEY.

PROGRAM STATES

The four types of program-state alternatives, which determine the overall CPU status, are named Problem/Supervisor, Wait/Running, Masked/Interruptible, and Stopped/Operating. These states differ in the way they affect the CPU functions and in the way their status is indicated and switched. The masked states have several alternatives; all other states have only one alternative.

All program states are independent of each other in their function, indication, and status switching. Status switching

does not affect the contents of the arithmetic registers or the execution of I/O operations but may affect the timer operation.

PROBLEM STATE

The choice between supervisor and problem state determines whether the full set of instructions is valid. The names of these states reflect their normal use.

In the problem state all I/O, protection, and direct-control instructions are invalid, as well as LOAD PSW, SET SYSTEM MASK, and DIAGNOSE. These are called priviledged instructions. A priviledged instruction encountered in the problem state constitutes a priviledge-operation exception and causes a program interruption. In the supervisor state all instructions are valid.

When bit 15 of the PSW is zero, the CPU is in the supervisor state. When bit 15 is one, the CPU is in the problem state. The supervisor state is not indicated on the operator sections of the system control panel.

The CPU is switched between problem and supervisor state by changing bit 15 of the PSW. This bit can be changed only by introducing a new PSW. Thus status switching may be performed by LOAD PSW, using a new PSW with the desired value of bit 15. Since LOAD PSW is a priviledged instruction, the CPU must be in the supervisor state prior to the switch. A new PSW is also introduced when the CPU is interrupted. The SUPERVISOR CALL

causes an interruption and thus may change the CPU state. Similarly, initial program loading introduces a new PSW and with it a new CPU state. The new PSW may introduce the problem or supervisor state regardless of the preceding state. No explicit operator control is provided for changing the supervisor state.

WAIT STATE

In the wait state no instructions are processed, and storage is not addressed repeatedly, whereas in the running state, instruction fetching and execution proceed in the normal manner.

When bit 14 of the PSW is one, the CPU is waiting. When bit 14 is zero, the CPU is in the running state. The wait state is indicated on the operator control section of the system control panel by the wait light.

The CPU is switched between wait and running state by changing bit 14 of the PSW. This bit can be changed only be introducing an entire new PSW, as is the case with the problem-state bit. Thus, switching from the running state may be achieved by the priviledged instruction LOAD PSW, by an interruption such as for SUPERVISOR CALL, or by initial program loading. Switching from the wait state may be achieved by an I/O or external interruption, or again, by initial program loading. The new PSW may introduce the wait or running state regardless of the preceding state. No explicit operator control is provided for changing the wait state.

To leave the wait state without manual intervention, the CPU should be interruptible for some active I/O or external interruption source.

MASKED STATES

The CPU may be masked or interruptible for all I/O, external, and machine-check interruptions and for some program interruptions. When the CPU is interruptible for a class of interruptions, these interruptions are accepted. When the CPU is masked, the system interruptions remain pending, while the program and machine-check interruptions are ignored.

The system mask bits (PSW bits 0-7), the program mask bits (PSW bits 36-39), and the machine-check mask bit (PSW bit 13) indicate as a group the masked state of the CPU. When a mask bit is one, the CPU is interruptible for the corresponding interruptions. When the mask bit is zero, these interruptions are masked off. The system mask bits indicate the masked state of the CPU for multiplexor and selector channels and the external signals. The program mask bits indicate the masked state for four of the 15 types of program exceptions. The machine-check mask bit pertains to all machine checks. Program interruptions not maskable, as well as the supervisor-call interruption, are always taken. The masked states are not indicated on the operator sections of the system control panel.

Most mask bits do not affect the execution of CPU operations. The only exception is the significance mask bit, which determines the manner in which a floating-point operation is completed when a significance exception occurs.

The interruptible state of the CPU is switched by changing the mask bits in the PSW. The program mask may be changed separately by SET PROGRAM MASK, and the system mask may be changed separately by the privileged instruction SET SYSTEM MASK. The machine-check bit can be changed only by introducing an entire new PSW, as in the case with the problem-state and wait-state bits. Thus, change in the entire masked status may be achieved by the privileged instruction LOAD PSW, by an interruption such as for SUPERVISOR CALL, or by initial program loading. The new PSW may introduce a new masked state regardless of the preceding state. No explicit operator control is provided for changing the masked state.

To prevent an interruption-handling routine from being interrupted before necessary housekeeping steps are performed, the new PSW for that interruption should mask the CPU for further interruptions of the kind that caused the interruption.

PROTECTION

Protection is provided to protect the contents of certain areas of main storage from destruction (or misuse) caused by erroneous storing (or storing and fetching) of information during the execution of a program. Locations may be protected against store violations or against store and fetch violations but never against fetch violations alone. This protection is achieved by identifying blocks of storage with a key and comparing this key with a protection key

supplied with the data to be stored. The detection of a mismatch causes the access to be suppressed, and a protection exception is recognized.

The key in storage is not part of addressable storage. The key is changed by SET STORAGE KEY and is inspected by INSERT STORAGE KEY. The protection key of the CPU occupies bits 8-11 of the PSW.

The protection system is always active. It is independent of the problem, supervisor, or masked state of the CPU and of the type of instruction or I/O command being executed.

When an instruction causes a protection mismatch, the protected main-storage location remains unchanged.

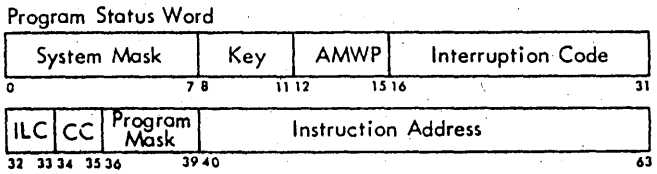
PROGRAM STATUS WORD

The PSW contains all information not contained in storage or registers but required for proper program execution. By storing the PSW, the program can preserve the detailed status of the CPU for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU may be changed.

In certain circumstances all of the PSW is stored or loaded; in others, only part of it. The entire PSW is stored, and a new PSW is introduced when the CPU is interrupted. The rightmost 32 bits are stored in BRANCH AND LINK. The LOAD PSW introduces a new PSW; SET PROGRAM-

MASK introduces a new condition code and program-mask field in the PSW; SET SYSTEM MASK introduces a new system-mask field.

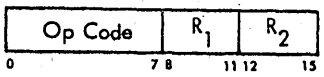
The PSW has the following format:



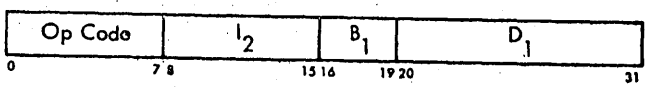
INSTRUCTION FORMAT

Status-switching instructions use the following two formats:

RR Format



SI Format



In the RR format, the R1 field specifies a general register, except for SUPERVISOR CALL. The R2 field specifies a general register in SET STORAGE KEY and INSERT STORAGE KEY. The R1 and R2 fields in SUPERVISOR CALL contain an identification code. In SET PROGRAM MASK the R2 field is ignored.

In the SI format the eight-bit immediate field (I2) of the instruction contains an identification code. The I2 field is ignored in LOAD PSW, SET SYSTEM MASK, and TEST AND SET. The content of the general register specified by B1 is added to the content of the D1 field to form an address designating the location of an operand in storage. Only one operand location is required in status-switching operations.

A zero in the B1 field indicates the absence of the corresponding address component.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and D1(B1) the operand designation.

STATUS-SWITCHING EXCEPTIONS

Exceptional instructions, operand designations, or data cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW identifies the cause of the interruption. The following exception conditions cause a program interruption in status-switching operations.

Operation: The protection feature is not installed and the instruction is SET STORAGE KEY OR INSERT STORAGE KEY.

Privileged Operation: A LOAD PSW, SET SYSTEM MASK, SET STORAGE KEY, INSERT STORAGE KEY, or DIAGNOSE is encountered while the CPU is in the problem state.

Protection: The key of an operand in storage does not match the protection key in the PSW. The instruction is suppressed on a store violation, except for TEST AND SET, which is terminated. The operation is terminated on a fetch violation.

Addressing: An address designates a location outside the available storage for the installation. The operation is terminated, except for DIAGNOSE, which is suppressed.

Specification: The operand address of a LOAD PSW does not have all three low-order bits zero; the operand address of DIAGNOSE does not have as many low-order zero bits as required for the particular CPU; the block address specified by SET STORAGE KEY OR INSERT STORAGE KEY does not have the four low-order bits all zero; or the protection feature is not installed and a PSW with a nonzero protection key is introduced.

When an instruction is suppressed, storage and external signals remain unchanged, and the PSW is not changed by information from storage.

When an interruption is taken, the instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction-length code in the old PSW.

Operand addresses are tested only when used to address storage. The address restrictions do not apply to the components from which an address is generated: the content of the D1 field and the content of the register specified by B1.

FIXED POINT FAMILY

LOAD

The primary purpose of the LOAD instruction is to transfer the contents of any location to that of another location. The location of the data to be transferred is specified by the second address field. The first address field specifies the location to which the data will be transferred. The second operand remains unchanged.

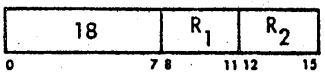
The term "Operand" refers to the data that is operated on by the instruction. The location of the Operand is specified by the "address field." The "first address field" has a subscript 1 attached as; R1 or D1 (B1). The "second address field" is also described by its subscripts as; R2 or D2(X2,B2).

The following LOAD instructions exhibit additional characteristics:

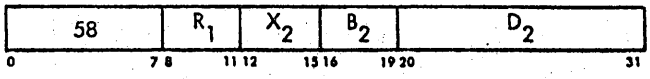
LH	Expands a halfword operand to a fullword operand by propagating the sign bit to the left through the 16 high-order bit positions.
LTR	Sets the Condition Code as a result of the data contained in the second operand.
LCR	The second operand is changed to the two's complement form when transferred.
LPR	The second operand is changed to a positive number (IF NEGATIVE)
LNR	The second operand is changed to a negative number (IF POSITIVE)
LM	Allows more than one register to be loaded at a time.

Load

LR R_1, R_2 [RR]



L $R_1, D_2(X_2, B_2)$ [RX]



1. The fullword specified by the second address field [R2 or D2(X2,B2)] is placed in the register specified by the first address field (R1).
2. The second operand remains unchanged.

EXAMPLES

1. LR (Load Register)
   Load the contents of register 5 into register 3.

   SYMBOLIC LR 3,5 MACHINE 18 35

		Before	After
GPR	3	F1 96 0A CD	7F 00 19 86
GPR	5	7F 00 19 86	7F 00 19 86

2. L (Load)
   Load the contents of storage address 1000 (FIELD1) into register 7. (GPR F = 00 00 10 00)

   SYMBOLIC L 7,FIELD1 MACHINE 58 70 F0 00

		Before	After
GPR	7	00 00 00 00	00 00 FC DE
Storage	1000	00 00 FC DE	00 00 FC DE

CONDITION CODE

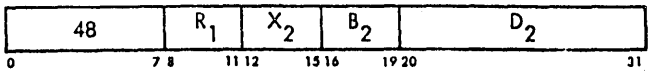
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (fetch L only)
2. Addressing (L only)
3. Specification (L only)

Load Halfword

LH $R_1, D_2(X_2, B_2)$ [RX]



1. The halfword second operand [D2(X2,B2)] is placed in the register specified by the first address field (R1).
2. The halfword operand is expanded to a fullword by propagating the sign bit through the 16 high-order bit positions.
3. The second operand remains unchanged.

EXAMPLES

1. LH
   Load the halfword contents of storage location 1002 (FIELD1+2) into register C. (GPR F = 00 00 10 00)

   SYMBOLIC LH 12,FIELD1+2 MACHINE 48 C0 F0 02

		Before	After
GPR	C	00 00 00 00	FF FF 9F 10
Storage	1000	F0 80 9F 10	F0 80 9F 10

CONDITION CODE

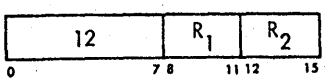
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification

Load and Test

LTR R_1, R_2 [RR]



1. The contents of the register specified by the second operand (R2) are placed in the location specified by the first operand (R1).
2. The sign and value of the second operand determines the setting of the Condition Code.
3. The second operand remains unchanged.

EXAMPLES

1. LTR (Load and Test Register)
   Load the contents of register 1 into register A and set the Condition Code.

   SYMBOLIC LTR 10,1 MACHINE 12 A1

	Before	After
GPR 1	0C 10 00 00	0C 10 00 00
GPR A	AB 19 24 35	0C 10 00 00
Condition Code		

2. LTR
   Test the contents of register 2 without loading into another location.
   SYMBOLIC LTR 2,2 MACHINE 12 22

	Before	After
GPR 2	91 23 45 6A	91 23 45 6A
Condition Code	1	

CONDITION CODE

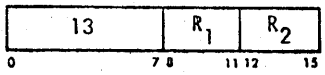
0	Operand is zero
1	Operand is negative
2	Operand is positive
3	--

PROGRAM INTERRUPTIONS

1. None

Load Complement

LCR R_1, R_2 [RR]



1. The contents of the register specified by the second address field (R2) are placed in the register designated by the first address field (R1).
2. The second operand is complemented prior to placing in the first operand location.
3. A second operand containing a value of zero would remain unchanged.
4. A second operand containing the maximum negative number will not be complemented and will cause a fixed point overflow. The Condition Code is set to 3.
5. Both positive and negative values are complemented.

EXAMPLES

1. LCR (Load Complement Register)
   Complement and load the contents of register 6 into register 8.

   SYMBOLIC LCR 8,6 MACHINE 13 86

		Before	After
GPR	6	FF FF FF FF	FF FF FF FF
GPR	8	00 00 00 00	00 00 00 01

   Condition Code 2

2. LCR
   Complement the contents of register D.

   SYMBOLIC LCR 13,13 MACHINE 13 DD

		Before	After
GPR	D	00 00 00 AC	FF FF FF 54

   Condition Code 1

CONDITION CODE

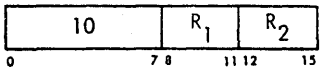
0 Result is zero
1 Result is negative
2 Result is positive
3 Overflow

PROGRAM INTERRUPTIONS

1. Fixed Point Overflow

Load Positive

LPR R_1, R_2 [*RR*]



1. The absolute value contained in the register specified by the second address field (R2) is placed in the register specified by the first address field (R1).
2. Negative numbers are complemented while positive numbers remain unchanged.
3. A second operand containing the value of zero remains unchanged.
4. A second operand containing the maximum negative number will not be complemented and will cause a fixed point overflow. The Condition Code is set to 3.

EXAMPLES

1. LPR (Load Positive Registers)
   Load the absolute value of register E into register 0.

SYMBOLIC LPR 0,14 MACHINE 10 0E

		Before	After
GPR	0	79 0A 05 63	3A FC 19 A8
GPR	E	3A FC 19 A8	3A FC 19 A8

Condition Code 2

2. LPR
Change the contents of register 9 to its absolute value.

SYMBOLIC LPR 9,9 MACHINE 10 99

		Before	After
GPR	9	80 00 00 01	7F FF FF FF

Condition Code 2

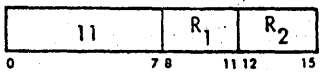
CONDITION CODE

0 Result is zero
1 --
2 Result is positive
3 Overflow

PROGRAM INTERRUPTIONS
1. Fixed Point overflow

Load Negative

LNR R_1, R_2 [RR]



1. The two's complement of the absolute value contained in the register specified by the second address field (R2) is placed in the register specified by the first address field (R1).
2. Positive numbers are complemented while negative numbers remain unchanged.
3. A second operand containing the value of zero remains unchanged.

EXAMPLES

1. LNR
Complement the absolute value contained in register 3 and place in register 7.

SYMBOLIC LNR 7,3 MACHINE 11 73

		Before	After
GPR	3	70 00 00 10	70 00 00 10
GPR	7	00 00 00 00	8F FF FF F0

Condition Code 1

2. LNR
   Complement the absolute value contained in register 8.

```
SYMBOLIC  LNR 8,8                    MACHINE 11 88

                              Before        After
  GPR      8          90 00 00 00    90 00 00 00
  Condition Code  1
```

CONDITION CODE

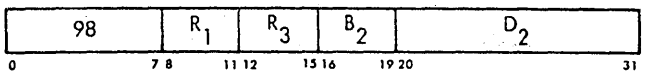
0	Result is zero
1	Result is negative
2	--
3	--

PROGRAM INTERRUPTIONS

1. None

Load Multiple

LM $R_1, R_3, D_2(B_2)$ [RS]



1. The general registers starting with the register specified by the first address field (R1) and ending with the register specified by the third address field (R3) are loaded from the locations designated by the second address field (B2,D2).
2. The registers are loaded from storage beginning at the address specified by the second address field and continuing in increments of four bytes until all specified registers are loaded.
3. The registers are loaded in ascending order beginning with the register specified by R1 and continuing up to and including the register specified by R3.
4. All combinations of register addresses specified by R1 and R3 are valid.
5. When the address specified by R3 is less than R1, the register addresses wrap around from F to 0.
6. The second operand remains unchanged.

EXAMPLES

1. LM
   Load registers 9 through B from storage locations 1000 (DATA) through 100B. (GPR F = 00 00 10 00)

```
SYMBOLIC  LM 9,11,DATA              MACHINE 98 9B F0 00
```

		Before	After
GPR	9	00 00 00 00	00 00 02 00
GPR	A	00 00 00 00	00 00 00 04
GPR	B	00 00 00 00	00 00 FC DE
Storage	1000	00 00 02 00	00 00 02 00
	1004	00 00 00 04	00 00 00 04
	1008	00 00 FC DE	00 00 FC DE

2. LM

Load register E through 1 from storage locations 1000 (DATA) through 100F. (GRP F = 00 00 10 00)

SYMBOLIC LM 14,1,DATA MACHINE 98 E1 F0 00

		Before	After
GPR	E	00 00 00 05	12 34 56 AC
GPR	F	00 00 10 00	00 00 10 00
GPR	0	23 1C 1A 23	AB CD EF 01
GPR	1	5A 7C 00 00	13 72 A2 19
Storage	1000	12 34 56 AC	12 34 56 AC
	1004	00 00 10 00	00 00 10 00
	1008	AB CD EF 01	AB CD EF 01
	100C	13 72 A2 19	13 72 A2 19

CONDITION CODE

1. Remains unchanged

PROGRAMMING INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification

FIXED POINT FAMILY

STORE

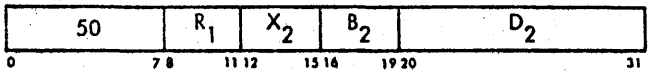
The STORE instruction is used to transfer the contents of the General Purpose Registers to main storage. The location of the data to be transferred is specified by the first and third (when applicable) address fields. The second address field designates the beginning address in main storage where the data will be placed.

The following STORE instructions exhibit additional characteristics:

STH	Stores the low-order 16 bits of a register.
STM	Stores more than one register at a time.

Store

ST $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the register specified by the first address field (R1) are stored at the location designated by the address field [D2(X2,B2)].

EXAMPLES

1. ST (Store)
   Store the contents of register D into storage location 1004 (DATA+4). (GPR F = 00 00 10 00)

   SYMBOLIC ST 13,DATA+4 MACHINE 50 D0 F0 04

		Before	After
GPR	D	0A C1 85 69	0A C1 85 69
Storage	1004	0F F0 16 72	0A C1 85 69

CONDITION CODE

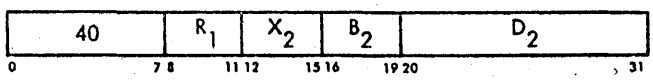
1. Remains unchanged

PROGRAMMING INTERRUPTIONS

1. Protection (store only)
2. Addressing
3. Specification

Store Halfword

STH $R_1, D_2(X_2, B_2)$ [RX]



1. The low-order 16 bits of the register specified by the first address field (R1) are stored at location designated by the address field [D2(X2,B2)].

EXAMPLE

1. STH (Store Halfword)
   Store the low order two bytes of register 0 into storage location 1000 (DATA). (GPR F = 00 00 10 00)

   SYMBOLIC STH 0,DATA MACHINE 40 00 F0 00

		Before	After
GPR	0	00 FC 04 50	00 FC 04 50
Storage	1000	FE D9 16 25	04 50 16 25

CONDITION CODE

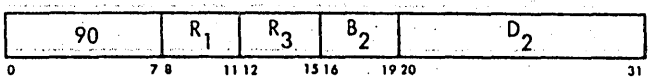
1. Remains unchanged.

PROGRAM INTERRUTPIONS

1. Protection (store only)
2. Addressing
3. Specification

Store Multiple

STM $R_1, R_3, D_2(B_2)$ [RS]



1. The general register starting with the register specified by the first address field (R1) and ending with the register specified by the third address field (R3) are stored in the location designated by the second address field (D2,B2).
2. The registers are stored in ascending order beginning with the register specified by the first address field and continuing until all specified registers are stored.
3. The beginning storage address is incremented by 4 bytes after each register is stored. This continues until all specified registers have been stored.
4. All combinations of register address specified by R1 and R3 are valid.
5. When the address specified by R3 is less than R1, the register addresses wrap around from F to 0.

EXAMPLES

1. STM (Store Multiple)
   Store the contents of register 4 and 5 into storage location 1000 (DATA) through 1007. (GPR F = 00 00 10 00)

   SYMBOLIC STM 4,5,DATA MACHINE 90 45 F0 00

		Before	After
GPR	4	46 00 00 A1	46 00 00 A1
GPR	5	00 01 1A 23	00 01 1A 23
Storage	1000	27 AE FC D4	46 00 00 A1
	1004	00 00 00 02	00 01 1A 23

2. STM

Store the content of registers F and 0 into storage location 1008 (DATA+8) through 100F. (GPR F = 00 00 10 00)

SYMBOLIC STM 15,0,DATA+8 MACHINE 90 F0 F0 08

		Before	After
GPR	0	00 00 00 04	00 00 00 04
GPR	F	00 00 10 00	00 00 10 00
Storage	1008	00 00 00 08	00 00 10 00
	100C	00 00 20 00	00 00 00 04

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Protection (store only)
2. Addressing
3. Specification

FIXED POINT FAMILY

ADD

The ADD instruction is used to perform the addition of two operands. The second operand is added to the first operand and the result replaces the contents of the first operand.

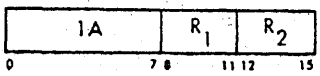
The following ADD instructions exhibit additional characteristics:

AH — Prior to the addition, it expands a halfword operand to a fullword by propagating the sign bit through the the 16 high-order bits positions.

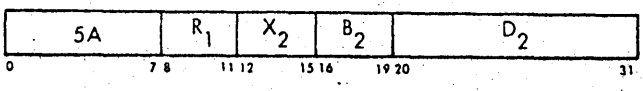
AL — Following the add, it records the occurrence of a carry out of the sign position in the Condition Code.

Add

AR R_1, R_2 [RR]



A $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the location specified by the second address field (R2 or D2(X2,B2)) are added to the contents of the register specified by the first address field (R1).
2. The sum replaces the first operand (R1).
3. Addition is performed by adding all 32 bits of both operands.
4. If the carry out of the sign-bit position and the high-order numeric bit position agree, the sum is satisfactory. If they disagree, an overflow occurs.
5. A positive overflow results in a negative sum.
6. A negative overflow results in a positive sum.
7. A register may be added to itself.

EXAMPLES

1. AR (Add Registers)
   Add the contents of register 4 to the contents of register 7 and have the result placed in register 7.

   SYMBOLIC AR 7,4 MACHINE 1A 74

		Before	After
GPR	4	70 00 12 3C	70 00 12 3C
GPR	7	00 00 0A 81	70 00 1C BD
Condition Code		2	

2. A (Add)
   Add the fullword contents of storage location 1000 (OPER1) to register 3. (GPR F = 00 00 10 00)

   SYMBOLIC A 3,OPER1 MACHINE 5A 30 F0 00

		Before	After
GPR	3	00 00 00 04	00 00 00 03
Storage	1000	FF FF FF FF	FF FF FF FF
Condition Code		2	

CONDITION CODE

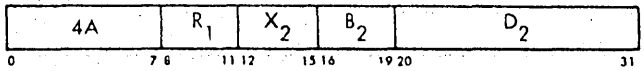
0	Sum is zero
1	Sum is negative
2	Sum is positive
3	Overflow

PROGRAM INTERRUPTIONS

1. Protection (fetch A only)
2. Addressing (A only)
3. Specification (A only)
4. Fixed-point overflow

Add Halfword

AH $R_1, D_2(X_2, B_2)$ [RX]



1. The halfword designated by the second address field (D2 (X2,B2)) is added to the register specified by the first address field (R1).
2. The halfword second operand is expanded to a fullword, prior to addition, by propagating the sign-bit value through the 16 high-order bit positions.
3. The sum replaces the contents of the register specified by the first address field (R1).
4. Addition is performed by adding all 32 bits of both operands.
5. If the carry out of the sign-bit position and the high-order numeric bit position agree, the sum is satisfactory. If they disagree, an overflow occurs.
6. A positive overflow results in a positive sum.

EXAMPLES

1. AH (Add Halfword)
   Add the halfword contents of storage location 1002 (OPER1+2) to register 5.

   SYMBOLIC AH 5,OPER1+2 MACHINE 4A 50 F0 02

		Before	After
GPR	5	80 00 00 01	80 00 00 00
Storage	1000	00 00 FF FF	00 00 FF FF
Condition Code		1	

CONDITION CODE

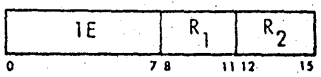
0 Sum is zero
1 Sum is negative
2 Sum is positive
3 Overflow

PROGRAM INTERRUPTIONS

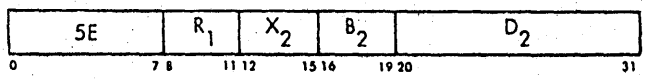
1. Protection (fetch only)
2. Addressing
3. Specification
4. Fixed-point overflow

Add Logical

ALR R_1, R_2 [RR]



AL $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the second address field [R2 or D2 (X2,B2)] are added to the register specified by the first address field (R1).
2. The sum replaces the first operand (R1).
3. Logical addition is performed by adding all 32 bits of both operands without further change to the resulting sign bit.
4. An overflow condition is not indicated.
5. A carry out of the sign position is recorded in the the Condition Code.

EXAMPLES

1. ALR (Add Logical Register)
   Logically add the contents of register 9 to register 7 and place the sum in register 7.

   SYMBOLIC ALR 7,9 MACHINE 1E 79

		Before	After
GPR	7	00 00 00 30	00 00 01 30
GPR	9	00 00 01 00	00 00 01 00

   Condition Code 1

2. AL (Add Logical)
Logically add the contents of register 9 to storage location 1004 (OPER1+4). (GPR F = 00 00 10 00)

SYMBOLIC AL 9,OPER1+4 MACHINE 5E 90 F0 04

	Before	After
GPR 9	80 00 00 00	00 00 00 01
Storage 1004	80 00 00 01	80 00 00 01

Condition Code 3

CONDITION CODE

0 Sum is zero (no carry)
1 Sum is not zero (no carry)
2 Sum is zero (carry)
3 Sum is not zero (carry)

PROGRAM INTERRUPTIONS

1. Protection (fetch AL only)
2. Addressing (AL only)
3. Specification (AL only)

FIXED POINT FAMILY

SUBTRACT

The primary purpose of the SUBTRACT instruction is to find the difference between two operands. The location specified by the second address field contains the subtrahend. The minuend is contained in the register specified by the first address field. The first operand is replaced by the difference.

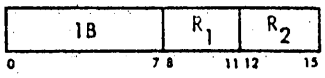
The following SUBTRACT instructions exhibit additional characteristics:

SH — Prior to subtraction the halfword operand is expanded to a full word by propagating the sign bit through the high-order 16 bit positions.

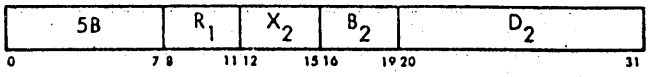
SL — Following the subtraction it records the occurrence of a carry out of the sign position in the Condition Code.

Subtract

SR R_1, R_2 [RR]



S $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the location specified by the second address field [R2 or D2(X2,B2)] are subtracted from the contents of the register specified by the first address field (R1).
2. The difference replaces the first (R1).
3. Subtraction is performed by complement addition.
4. If the carry out of the sign-bit position and the high-order numeric bit position agree, the difference is satisfactory. If they disagree, an overflow occurs.
5. A register may be cleared by subtracting it from itself.

EXAMPLES

1. SR (Subtract Register)
Subtract the contents of register 7 from the contents of register 5.

SYMBOLIC SR 5,7 MACHINE 1B 57

		Before	After
GPR	5	00 00 2F ED	00 00 2E 1F
GPR	7	00 00 01 CE	00 00 01 CE

Condition Code 2

2. SR
Clear register 7 by subtraction.

SYMBOLIC SR 7,7 MACHINE 1B 77

		Before	After
GPR	7	00 00 01 CE	00 00 00 00

Condition Code 0

3. S (Subtract)
Subtract the fullword contents at storage location 200C (HOURS+12) from register C. (GPR F = 00 00 20 00)

SYMBOLIC	S 12,HOURS+12		MACHINE 5B C0 F0 0C
GPR	C	80 00 00 00	00 00 00 00
Storage	200C	80 00 00 00	80 00 00 00
Condition Code		0	

CONDITION CODE

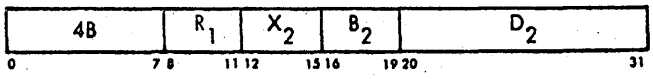
0 Difference is zero
1 Difference is negative
2 Difference is positive
3 Overflow

PROGRAM INTERRUPTIONS

1. Protection (fetch S only)
2. Addressing (S only)
3. Specification (S only)
4. Fixed-point overflow

Subtract Halfword

SH $R_1, D_2(X_2, B_2)$ [RX]



1. The halfword designated by the second address field (D2(X2,B2)) is subtracted from the contents of the register specified by the first address field (R1).
2. The halfword operand is expanded to a fullword, prior to subtraction, by propagating the sign-bit value through the 16 high-order bit positions.
3. The difference replaces the first operand.
4. Subtraction is performed by complement addition of all 32 bits of both operands.
5. If the carry out of the sign-bit position and the high-order numeric bit position agree, the difference is satisfactory. If they disagree, an overflow occurs.

EXAMPLES

1. SH (Subtract Halfword)
   Subtract the halfword operand at storage location 2000 (HOURS) from the contents of register E. (GPR F = 00 00 20 00)

   SYMBOLIC SH 14,HOURS MACHINE 4B E0 F0 00

		Before	After
GPR	E	06 17 FC 10	06 18 1C 00
Storage	2000	E0 10 FC D8	E0 10 FC D8
Condition Code		2	

CONDITION CODE

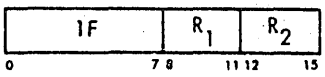
0 Difference is zero
1 Difference is negative
2 Difference is positive
3 Overflow

PROGRAM INTERRUPTIONS

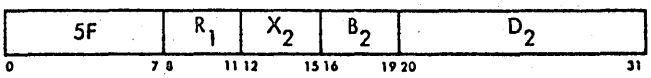
1. Protection (fetch only)
2. Addressing
3. Specification
4. Fixed-Point Overflow

Subtract Logical

SLR R_1, R_2 [RR]



SL $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the second address field (R2 or D2(X2,B2)) are subtracted from the register specified by the first address field (R1).
2. The difference replaces the first operand (R1).
3. Logical subtraction is performed by complement addition. All 32 bits of both operands are complement added without further change to the resulting sign bit.
4. An overflow condition is not indicated.
5. A carry out of the sign position is recorded in the Condition Code.

EXAMPLES

1. SLR (Subtract Logical Register)
Logically subtract the contents of register E from the contents of register 3 and place the difference in register 3.

SYMBOLIC SLR 3,E MACHINE 1F 3E

		Before	After
GPR	3	00 00 00 1E	00 00 00 29
GPR	E	FF FF FF F5	FF FF FF F5
Condition Code		1	

2. SL (Subtract Logical)
Logically subtract the fullword contents at storage location 2000 (HOURS) from register 6.
(GPR F = 00 00 20 00)

SYMBOLIC SL 6,HOURS MACHINE 5F 60 F0 00

		Before	After
GPR	6	00 00 00 FE	00 00 00 00
Storage	2000	00 00 00 FE	00 00 00 FE
Condition Code		2	

CONDITION CODE

0 --
1 Difference is not zero (no carry)
2 Difference is zero (carry)
3 Difference is not zero (carry)

PROGRAM INTERRUPTIONS

1. Protection (fetch SL only)
2. Addressing (SL only)
3. Specification (SL only)

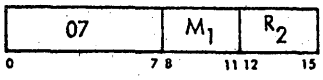
BRANCHING FAMILY

BRANCH ON CONDITION

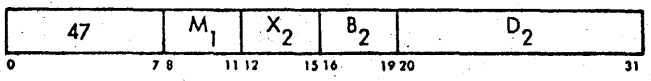
The BRANCH on CONDITION instruction provides a means of leaving the normal instruction sequence. The effective address of the second address field is the branch address. This branch address will be used to replace the next instruction address if the Mask Field matches the Condition Code. The first operand is a 4-bit Mask Field contained in bits 8-11 of the instruction.

Branch On Condition

BCR M_1, R_2 [RR]



BC $M_1, D_2(X_2, B_2)$ [RX]



1. The Mask Field (M1) is compared to the Condition Code to determine if the branch will occur.
2. This is performed by comparing the Condition Code to its corresponding Mask Field bit as follows:

INSTRUCTION BIT (M1),	CONDITION CODE
8	0
9	1
10	2
11	3

   The existing Condition Code determines which Mask Field bit is checked. If that Mask bit is a one, the branch will occur.
3. The updated instruction address is replaced with the contents of the location designated by second address field [R2 or D2(X2,B2)] when the Condition Code and the Mask Field match.
4. Normal instruction sequencing begins at the new address [contents of R2 or D2(X2,B2)] when the branch is taken.
5. When a match condition does not exist, this instruction performs no action; and the updated instruction address will determine the next sequential instruction.
6. The first and second operands remain unchanged.
7. If R2 is zero, no branch is taken (no-op).

EXAMPLES

1. BCR (Branch on Condition Register)
   Branch to the location specified by register 6 if the Condition Code is 0 or 1.

   SYMBOLIC BCR 12,6 MACHINE 07 C6

2. BC (Branch on Condition)
   Unconditionally branch to location 41CE (TEXT+2). (GPR F = 00 00 40 00)

   SYMBOLIC BC 15,TEST+2 MACHINE 47 F0 F1 CE

CONDITION CODE

1. Remains unchanged

PROGRAMMING INTERRUPTIONS

1. None

FIXED POINT FAMILY

COMPARE

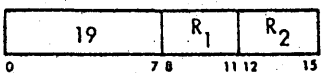
The COMPARE instruction is used to determine the similarity or difference between two operands. The contents of the register specified by the first address field are compared to the contents of the location designated by the second address field. The Condition Code is set to reflect the similarity or difference between the two operands.

The following COMPARE instruction exhibits an additional characteristic:

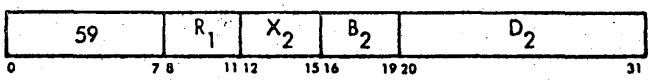
CH Expands a halfword second operand to a fullword by propagating the sign bit through the 16 high-order bit positions prior to comparing.

Compare

CR R_1, R_2 [RR]



C $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the register specified by the first address field (R1) are compared to the contents of the location designated by the second address field (R2 or D2 (X2,B2)) .
2. Comparison is algebraic, treating both operands as 32-bit signed integers.
3. The result of the comparison determines the Condition Code.
4. Both operands remain unchanged.

EXAMPLES

1. CR (Compare Registers)
Compare the contents of register A to the contents of register B.

SYMBOLIC CR 10,11 MACHINE 19 AB

		Before	After
GPR	A	07 00 10 0C	07 00 10 0C
GPR	B	87 00 00 0C	87 00 00 0C
Condition Code		2	

2. C (Compare)
Compare the contents of register 1 to a fullword operand storage location 4000 (TEXT). (GPR F = 00 00 40 00)

SYMBOLIC C 1,TEXT MACHINE 59 10 F0 00

		Before	After
GPR	1	1A 23 1A 12	1A 23 1A 12
Storage	4000	1A 23 1B 33	1A 23 1B 33
Condition Code		1	

CONDITION CODE

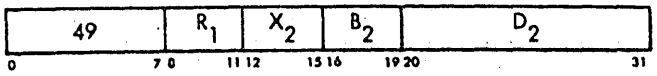
0	Operands are equal
1	First operand is low
2	First operand is high
3	--

PROGRAMMING INTERRUPTIONS

1. Protection (fetch C only)
2. Addressing (C only)
3. Specification (C only)

Compare Halfword

CH $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the register specified by the first address field (R1) are algebraically compared to the halfword designated by the second address field [D2 (X2,B2)].
2. The halfword operand is expanded to a fullword by propagating the sign bit value through the high-order 16-bit positions prior to the compare.
3. The result of the compare is indicated by the Condition Code.

EXAMPLES

1. CH (Compare Halfword)
   Compare the contents of register 2 to a halfword operand at storage location 400C (TEXT+12).
   (GPR F = 00 00 40 00)

   SYMBOLIC CH 2,TEXT+12 MACHINE 49 20 F0 0C

	Before	After
GPR 2	FF FF 80 00	FF FF 80 00
Storage 400C	80 00 CD EF	80 00 CD EF
Condition Code	0	

CONDITION CODE

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

PROGRAMMING INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification

FIXED POINT FAMILY

MULTIPLY

The MULTIPLY instruction is used to find the product of two integers (numbers). The first address field must specify an even address register. The multiplicand is located at a register address of one greater than the even register specified by the first operand. Multiplicands would always be contained in an odd address register (1, 3, 5, 7, 9, B, D or F).

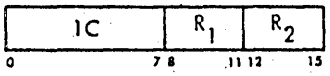
The product of these two integers is 64 bits long. It is placed in the even register designated by the first address field and the odd register which contained the multiplicand. This register pair will be referred to as the EVEN-ODD register pair.

The following MULTIPLY instruction exhibits an additional characteristic:

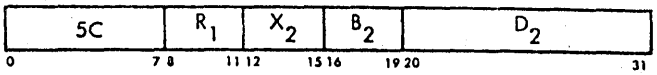
MH Expands a halfword second operand to a fullword by propagating the sign bit value through the 16 high-order bit positions prior to multiplication.

Multiply

MR R_1, R_2 [RR]



M $R_1, D_2(X_2, B_2)$ [RX]



1. The first address field (R1) must contain an EVEN register address.
2. This address specifies an EVEN-ODD register pair.
3. The contents of the ODD register (multiplicand) are multiplied by the second operand (R2 or D2(X2,B2)).
4. The product of this multiplication is 64 bits and is placed in the EVEN-ODD register pair.
5. The sign of the product is determined by the rules of algebra.
6. A specification exception will occur if the register specified by the first operand is at an odd address.

EXAMPLES

1. MR (Multiply Register)
Multiply the contents of register 5 by the contents of register 9.

SYMBOLIC MR 4,9 MACHINE 1C 49

		Before	After
GPR	4	9C 01 04 66	00 00 00 E0
GPR	5	70 00 00 00	00 00 00 00
GPR	9	00 00 02 00	00 00 02 00

2. M (Multiply)
Multiply the contents of register B by the fullword contents at storage location 3004 (MULT+4). (GPR F=00 00 30 00)

SYMBOLIC M 10,MULT+4 MACHINE 5C A0 F0 04

		Before	After
GPR	A	12 AC 1F 60	00 00 00 00
GPR	B	FF FF FF FE	00 00 00 02
Storage	3004	FF FF FF FF	FF FF FF FF

CONDITION CODE

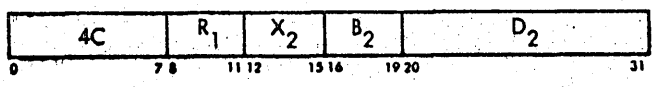
1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Protection (fetch M only)
2. Addressing (M only)
3. Specification

Multiply Halfword

MH $R_1, D_2(X_2, B_2)$ [RX]



1. The register specified by the first address field (R1) is multiplied by the halfword designated by the second address field [D2(X2,B2)].
2. The halfword operand is expanded to a fullword by propagating the sign bit value through the 16 high-order bit positions prior to multiplication.
3. The product is 32 bits and replaces the multiplicand in the register specified by R1.
4. The sign of the product is determined by the rules of algebra.
5. All register addresses are valid.
6. If the product exceeds 32 bits the high-order bits are lost.

EXAMPLES

1. MH (Multiply Halfword)
   Multiply the contents of register 3 by the halfword at storage location 3000(MULT). (GPR F = 00 00 30 00)

   SYMBOLIC MH 3,MULT MACHINE 4C 30 F0 00

		Before	After
GPR	3	00 00 00 21	00 00 00 A5
Storage	3000	00 05 CF 01	00 05 CF 01

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification

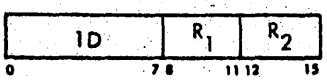
FIXED POINT FAMILY

DIVIDE

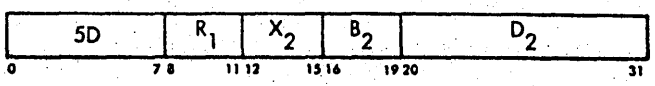
The DIVIDE instruction will produce the quotient of a 64-bit integer (number). The first operand must always be an even numbered register (0, 2, 4, etc.). This even register and the register whose address is one greater form an EVEN-ODD register pair. The dividend is a 64 bit signed integer that occupies this EVEN-ODD register pair.

The contents of the EVEN-ODD register is divided by the contents of the location specified by the second address field. The quotient is placed in the ODD register of the EVEN-ODD pair. The remainder is placed in the EVEN register.

DR R_1, R_2 [RR]



D $R_1, D_2(X_2, B_2)$ [RX]



1. The first address field (R1) must contain an even-address register.
2. This address specifies an EVEN-ODD register pair which contains the 64-bit dividend.
3. The contents of the EVEN-ODD register pair are divided by the second operand (R2 or D2(X2,B2)).
4. The quotient will be placed in the ODD register, and the EVEN register will be used to contain the remainder.
5. The sign of the quotient is determined by the rules of algebra while the remainder will have the sign of the dividend.
6. When the size of the dividend and divisor is such that the quotient cannot be contained in a 32-bit register, a fixed point divide exception occurs and the instruction is aborted.
7. A specification exception will occur if the register specified by the first operand is at an odd address.
8. The D instruction must specify an operand located on a fullword boundary.

EXAMPLES

1. DR (Divide Register)
Divide the contents of registers 6 and 7 by the contents of register 5.

SYMBOLIC DR 6,5 MACHINE 1D 65

		Before	After
GPR	5	00 00 02 BC	00 00 02 BC
GPR	6	00 00 00 A3	00 00 01 8B
GPR	7	EC D5 3E 83	38 F3 23 3A

2. D (Divide)
Divide the contents of registers 8 and 9 by the fullword contents at storage location 3500 (DTA). (GPR F = 00 00 30 00)

SYMBOLIC D 8,DTA MACHINE 5D 80 F5 00

		Before	After
GPR	8	00 00 00 00	00 00 00 00
GPR	9	00 A0 00 00	00 50 00 00
Storage	3500	00 00 00 02	00 00 00 02

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (fetch D only)
2. Addressing (D only)
3. Specification
4. Fixed-point divide

SHIFT:

There are two main purposes of the shift instructions, editing of register data and arithmatic operations. Editing is normally done in the four "LOGICAL SHIFT" instructions which will be covered later.

The arithmatic operations of a SHIFT instruction is to multiply or divide an integer by some multiple of 2. The first address field specifies a register or an EVEN-ODD register pair that contains this integer. The second address field does not reference a storage location but is used to generate an effective address. The low-order 6 bit positions of this effective address determine by what multiple of 2 the interger will be divided or multiplied.

Multiplication is accomplished by shifting the specified register(s) contents to the left. The number of bit positions that the integer is shifted is determined by the low-order 6-bit positions of the effective address. Each bit position that the integer is shifted is equivalent to a multiplication by 2.

Division is accomplished by shifting the specified register(s) contents to the right. The number of bit positions that the integer is shifted is determined by the low-order 6-bit positions of the effective address. Each bit position that the integer is shifted is equivalent to a division by 2.

Shift Left Single

SLA $R_1, D_2(B_2)$ [RS]

1. The contents of the register specified by the first address field (R1) are shifted to the left.
2. The low-order 6 bits of the effective address [D2(B2)] specify the number of positions that the first operand will be shifted.
3. All 31 integer bits participate in the shift and 0's are placed in the vacated low order bit positions of the first operand.
4. Each bit position that the integer is shifted equates to a multiplication by 2.
5. When the low order 6 bits of the effective address exceed a decimal value of 30, the entire integer will be shifted out of the specified register.
6. The entire integer being shifted out of the specified register results in the value of 0 for a positive integer and minus 2,147,486,648 for a negative integer.
7. When a bit unlike the sign is shifted out of bit position 1, an overflow will occur.

EXAMPLES

1. SLA (Shift Left Algebraic)
   Multiply the contents of register 9 by 4 using the shift instruction and without specifying a base register.

```
SYMBOLIC  SLA 9,2(0) MACHINE    8B 90 00 02

                          Before        After

  GPR     9           00 00 0A 00   00 00 28 00

  Condition Code  2
```

CONDITION CODE

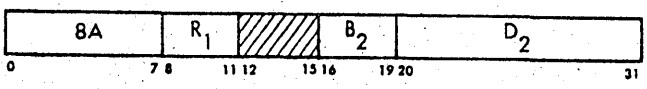
0	Result is 0
1	Result is less than 0
2	Result is greater than 0
3	Overflow

PROGRAM INTERRUPTIONS

1. Fixed point overflow

Shift Right Single

SRA $R_1, D_2(B_2)$ [RS]



1. The contents of the register specified by the first address field (R1) are shifted to the right.
2. The low-order 6 bits of the effective address [D2(B2)] specify the number of positions that the first operand will be shifted.
3. All 31 integer bits participate in the shift and bits like the sign are placed in the vacated high order bit positions of the first operand.
4. Each bit position that the integer is shifted equates to a division by 2.
5. When the low order 6 bits of the effective address exceed a decimal value of 30, the entire integer will be shifted out of the specified register.
6. The entire integer being shifted out of the specified register results in the value of 0 for a positive integer and -1 for a negative integer.
7. Low-order bits are shifted out without inspection and are lost.

EXAMPLES

1. SRA (Shift Right Algebraic)
Divide the contents of register 4 by 2 using the shift instruction and without specifying a base register.

SYMBOLIC SRA 4,1(0) MACHINE 8A 40 00 01

		Before	After
GPR	4	00 00 00 FB	00 00 00 7D
Condition Code		2	

2. SRA (Shift Right Algebraic)
Divide the contents of register A by 2 using the shift instruction and specifying a base register of 4. (GPR 4 = 00 00 00 01)

SYMBOLIC SRA 10,0(4) MACHINE 8A A0 40 00

		Before	After
GPR	A	FF FF FF FB	FF FF FF FD
Condition Code		1	

CONDITION CODE

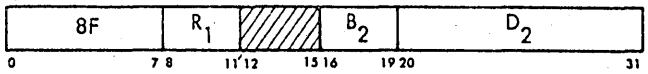
0	Result is 0
1	Result is less than 0
2	Result is greater than 0
3	--

PROGRAM INTERRUPTIONS

1. None

Shift Left Double

SLDA $R_1, D_2(B_2)$ [RS]



1. The contents of the EVEN-ODD register pair specified by the first address field (R1) are shifted to the left.
2. The low-order 6 bits of the effective address D2(B2) specify the number of the positions of the first operand will be shifted.
3. The operand is treated as a number with 63 integer bits and a sign in a sign position of the even register.
4. The high-order position of the ODD REGISTER contains an integer bit in the sign bit position.
5. All 63 integer bits participate in a shift and 0's are placed in the vacated low-order bit positions of the EVEN-ODD register pair.
6. Each bit position that the integer is shifted equates to a multiplication by 2.
7. When the low-order 6 bits of the effective address exceed a decimal value of 62, the entire integer will be shifted out of the specified registers.
8. When a bit unlike the sign is shifted out of bit position 1 of the even register, an overflow will occur.
9. A specification exception will occur if the register address specified by the first operand is odd.

EXAMPLES

1. SLDA (Shift Left Double Algebraic)
   Using the shift instruction, multiply the contents of register pair A and B by 8. Do not specify a base register.

   SYMBOLIC SLDA 10,3(0) MACHINE 8F A0 00 03

		Before	After
GPR	A	00 00 00 00	00 00 00 04
GPR	B	80 00 01 20	00 00 09 00
Condition Code		2	

2. SLDA (Shift Left Double Algebraic)
Using the shift instruction multiply the contents of register pair C and D by 4 using base register 6. (GPR 6 = 00 00 00 02)

SYMBOLIC SLDA 12,0(6) MACHINE 8F C0 60 00

		Before	After
GPR	C	80 00 00 00	80 00 00 01
GPR	D	7C 00 00 00	F0 00 00 00
Condition Code		3	

ONDITION CODE

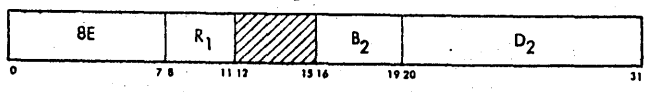
0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

ROGRAM INTERRUPTIONS

1. Specification
2. Fixed point overflow

Shift Right Double

SRDA R_1, $D_2(B_2)$ [RS]



1. The contents of the EVEN-ODD register pair specified by the first address field (R1) are shifted to the right.
2. The low-order 6 bits of the effective address D2(B2) specify the number of positions that the first operand will be shifted.
3. The operand is treated as a number with 63 integer bits and a sign in the sign position of even register.

4. The high-order position of the ODD register contains an integer bit in the sign bit position.
5. All 63 integer bits participate in the shift and bits like the sign are placed in the vacated high-order bit positions of the EVEN-ODD register pair .
6. Every bit position that the integer is shifted equates to a division by 2.
7. When the low-order 6 bits of the effective address exceeds a decimal value of the specified registers.
8. A specification exception will occur if the reigster address specified by the first operand is ODD.

EXAMPLES

1. SRDA (Shift Right Double Algebraic)
Using the shift instruction, divide the contents of register pair 2 and 3 by 16. Do not specify a base register.

SYMBOLIC SRDA 2,4(0) MACHINE 8E 20 00 04

		Before	After
GPR	2	00 0C 1F 39	00 00 C1 F3
GPR	3	FC 2A 67 49	9F C2 A6 74

Condition Code 2

2. SRDA (Shift Right Double Algebraic)
Using the shift instruction, divide the contents of register pair 4 and 5 by 8 using a base register. (GPR 1 = 00 00 00 02)

SYMBOLIC SRDA 4,1(1) MACHINE 8E 40 10 01

		Before	After
GPR	4	FF FF FF FF	FF FF FF FF
GPR	5	FF FF FF 00	FF FF FF E0

Condition Code 1

CONDITION CODE

0	Result is 0
1	Result is less than 0
2	Result is greater than 0
3	--

PROGRAM INTERRUPTIONS

1. Specification

FIXED POINT FAMILY

DATA FORMATS

Arithmetic data entering a system from an I/O (Input/Output) device normally occupies the zoned data format. When zoned data is to be used by either fixed-point or decimal instructions it must be converted into the data format used by these instructions. The PACK instruction will convert the incoming zoned data to packed decimal data.

The packed decimal data is used by decimal instructions, but must again be converted for use by fixed-point instructions. The CONVERT TO BINARY instruction performs the task of converting packed decimal data into fixed-point data.

When arithmetic data is to be taken from the system and sent to a formatted I/O device (card punch, printer, typewriter, etc.), it must be in the zoned data format. If the data is in the packed decimal data format, the UNPACK instruction will change it into the zoned format. Data in the fixed-point format must frist be changed to the packed decimal format by using the CONVERT TO DECIMAL instruction prior to conversion by the UNPACK instruction.

PACK

The Pack instruction converts data in the zoned format to packed decimal data. The starting address and length of the zoned data field to be converted are specified by the second operand.

The packing of zoned data is accomplished in a right-to-left sequence. The first step is placing the zone of the low-order zoned digit in the low-order 4 bits of the packed data field designated by the first operand. The digits contained in the zoned data field are now placed right-to-left, adjacent to each other and these 4 bits that become the sign of the packed-decimal number.

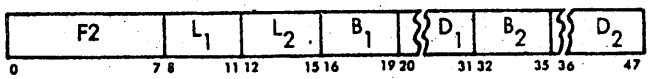
UNPACK

The Unpack instruction converts data from the packed decimal format to the zoned format. The starting address and length of the packed decimal field are specified by the second operand.

The unpacking of data is accomplished by placing, right-to-left, the digits of the packed-decimal data into the low-order 4 bits of each byte contained in the first operand's field. The zone (high-order 4 bits) supplied to these bytes is 1111 (0101 for USASCII-8) except the zone of the low-order byte which is set to the sign of the packed decimal data.

Pack

PACK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The second address field [D2(L2,B2)] defines a field in storage that contains zoned-arithmetic data.
2. The first address field [D1(L1,B1)] defines a field in storage where the contents of the second operands will be placed after being converted to the packed decimal data format.
3. The sign of the zoned data field (zone of the low-order byte) becomes the sign of the packed decimal data and is placed in the low-order 4 bits of the first operand field.
4. The digits of the zoned field are fetched one at a time and placed adjacent to the sign of the packed field and each other.
5. Fetching is performed in a right-to-left sequence and neither the sign or digit are checked for validity.
6. A first operand field larger than the resultant packed decimal number is supplied with zeros in the high-order unused positions.
7. If the first operand field is exhausted prior to completing the transfer of all the zoned digits the remaining zoned digits will be ignored.
8. First and second operand fields may overlap in any desired manner.

EXAMPLES

1. PACK (Pack)
Convert the zoned eight byte field located at storage location 4200(ZONE) to a packed decimal operand and place in a five-byte field beginning at storage location 4350(DEC). (GPR F = 00 00 40 00)

SYMBOLIC PACK DEC(5),ZONE(8)

MACHINE F2 47 F3 50 F2 00

		Before	After
Storage	4200	F2 F5 F6 F3	F2 F5 F6 F3
	4204	F7 F0 F1 D4	F7 F0 F1 D4
	4350	FF FF FF FF	02 56 37 01
	4354	FF FF FF FF	4D FF FF FF

2. PACK (Pack)
Switch the two packed-decimal digits at storage location 4351(DEC+1). (GPR F = 00 00 40 00)

SYMBOLIC PACK DEC+1(1),DEC+1(1)

MACHINE F2 00 F3 51 F3 51

	Before	After
Storage 4350	02 56 37 01	02 65 37 01

CONDITION CODE

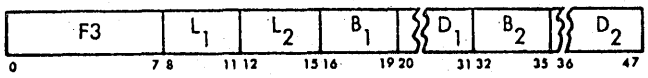
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

Unpack

UNPK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The second address field [D2(L2,B2)] defines a field in storage that contains packed-decimal data that will be converted to zoned data.
2. The first address field [D1(L1,B1)] specifies a field in storage where the result of the conversion (zoned data) will be placed.
3. The sign of the packed-decimal field (low-order 4 bits of the low-order packed byte) becomes the zone of the low-order zoned data byte.
4. The packed data is transferred a byte at a time in a right-to-left sequence and are not checked for valid digit or sign codes.
5. A packed digit is placed in the low-order 4 bits of each byte in the first operands field and is supplied with a zone of 1111 (0101 for USASCII-8 mode).
6. Zeros are supplied as high-order digits to be unpacked when the first-operand field is larger than the unpacked result.
7. If the first operand field is shorter than the unpacked result the high-order packed digits are ignored.
8. Overlapping of fields, greater than two bytes, requires the first operand field to begin at an address greater than the low-order byte of the second operand field. The number of bytes greater is equal to the second operand's length minus two.

EXAMPLE

1. UNPK (Unpack)
   Unpack the 6-byte field at storage location 4600 (DEC1) and place in a 12-byte field located at 4800(ZONE1). (GPR F = 00 00 45 00)

   SYMBOLIC UNPK ZONE1(12),DEC1(6)

   MACHINE F3 B5 F3 00 F1 00

		Before	After
Storage	4600	12 34 56 78	12 34 56 78
	4604	90 0C 23 45	90 0C 23 45
	4800	00 00 00 00	F0 F1 F2 F3
	4804	00 00 00 00	F4 F5 F6 F7
	4808	00 00 00 FF	F8 F9 F0 C0

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

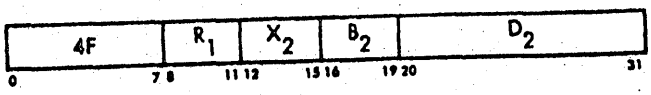
FIXED POINT FAMILY

CONVERT TO BINARY

The CONVERT TO BINARY instruction will convert a decimal number (Base 10) to its equivalent binary number (Base 2). The second address field designates a doubleword in storage, whose data is in the packed decimal data format. The contents of this storage location will be converted and placed in the register specified by the first address field.

Convert to Binary

CVB $R_1, D_2(X_2, B_2)$ [RX]



1. The decimal contents of the storage location designated by the second address field [D2(X2,B2)] are changed to binary and are placed in the register specified by the first address field (R1).
2. The second operand must be a doubleword in storage whose contents are in the packed decimal data format.
3. This doubleword is checked for valid sign and digit codes. A data exception will occur if either is invalid.
4. The second operand must be located on a double word boundary.
5. The maximum positive decimal number that can be converted is 2, 147, 483, 647. The maximum negative number is 2, 147, 483, 648.
6. If the maximum positive or negative number is exceeded, the low-order 32 binary bits are placed in the register specified by R1 and a fixed point divide exception will occur.
7. The number is located as a right aligned signed integer before and after conversion.
8. The contents of the second operand remain unchanged.

EXAMPLES

1. CVB (Convert Binary)
Convert the decimal number at storage address 1020 (DATA+32) to binary and place in register C. (GPR F = 00 00 10 00)

```
SYMBOLIC  CVB 12,DATA+32          MACHINE 4F C0 F0 20
```

		Before	After
GPR	C	00 00 FF FF	0E 00 00 00
Storage	1020	00 00 00 23	00 00 00 23
	1024	48 81 02 4C	48 81 02 4C

2. CVB (Convert Binary)
Convert the decimal number at storage address 1028 (DATA+40) to binary and place in register 3. (GPR F = 00 00 10 00)

```
SYMBOLIC  CVB 3,DATA+40          MACHINE 4F 30 F0 28

                          Before          After

  GPR      3              C9 16 00 12     FF 2A 96 B7

  Storage  1028           00 00 00 01     00 00 00 01
           102C           39 86 12 1D     39 86 12 1D
```

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification
4. Data
5. Fixed-point divide

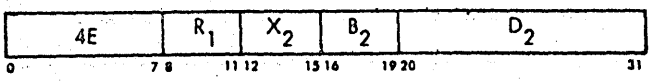
FIXED POINT FAMILY

CONVERT TO DECIMAL

The CONVERT TO DECIMAL instruction will convert a binary number (Base 2) to its equivalent decimal number (Base 10). The first address field specifies the register which contains the binary number. The second address field designates a doubleword in storage where the decimal result will be placed after the conversion.

Convert to Decimal

CVD $R_1, D_2(X_2, B_2)$ [RX]



1. The binary contents of the register specified by the first address field (R1) are changed to packed decimal and are placed in the doubleword storage location designated by the second address field [D2(X2,B2)].
2. The second operand must be located on a doubleword boundary.
3. The result placed in the second operand location will be a right aligned packed decimal integer.
4. The sign placed in the low order hex digit will be C or A for plus and D or B for minus.
5. The choice between the two sign representations is determined by the state of PSW bit 12.
6. Any binary value contained in a register can be converted and will not exceed the doubleword length that the second operand designates.
7. The contents of the first operand remain unchanged.

EXAMPLES

1. CVD (Convert Decimal)
   Convert the binary contents of register C to decimal and place at storage location 1020(DATA+32). (GPR F = 00 00 10 00)

```
SYMBOLIC  CVD 12,DATA+32          MACHINE 4E C0 F0 20

                          Before          After

GPR       C         0E 00 00 00    0E 00 00 00

Storage   1020      00 00 00 00    00 00 00 23
          1024      00 00 00 00    48 81 02 4C
```

2. CVD (Convert Decimal)
Convert the binary contents of register 3 to decimal and place at storage location 1028(DATA+40). (GPR F = 00 00 10 00)

SYMBOLIC CVD 3,DATA+40 MACHINE 4E 30 F0 28

		Before	After
GPR	3	FF 2A 96 B7	FF 2A 96 B7
Storage	1028	00 00 00 20	00 00 00 01
	102C	C9 99 99 99	39 86 12 1D

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection (store only)
2. Addressing
3. Specification

LOGICAL FAMILY

MOVE

The MOVE instruction is used to transfer data from one location to another. The location of the data to be moved is specified by the second address field. The first address field designates the location to which the data will be moved. The data is contained either in storage or within the instruction. The general-purpose registers are not used to receive or supply this data.

The following MOVE instructions exhibit additional characteristics:

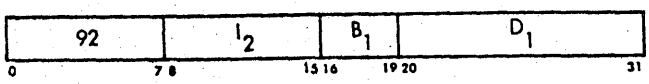
MVN — This permits the moving of the low-order 4 bits contained in any one byte or group of bytes.

MVZ — Allows the moving of the high order 4 bits contained in any one byte or group of bytes.

MVO — Provides a means of shifting data in the packed decimal data format and changing the sign of a packed decimal field.

Move

MVI $D_1(B_1), I_2$ [SI]



1. The second operand (I2) is contained within the instruction.
2. This one byte of data is placed at the location designated by the first address field [D1(B1)].
3. The second operand, contained within the instruction, is unchanged.

EXAMPLES

1. MVI (Move Immediate)
   Place the data FA into storage location 1001 (DATA+1) using the MOVE IMMEDIATE instruction. (GPR F = 00 00 10 00)

   SYMBOLIC MVI DATA+1,X'FA' MACHINE 92 FA F0 01

	Before	After
Storage 1000	00 00 00 00	00 FA 00 00

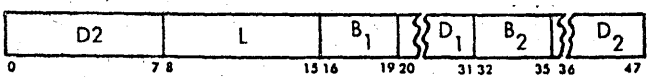
CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

MVC $D_1(L, B_1), D_2(B_2)$ [SS]



1. The length field (L) specifies the length in bytes of the first and second operands.
2. The maximum number of bytes that can be specified by the length field is 256.
3. The hexidecimal value of this field is always one less than the number of bytes actually transferred.
4. The second address field [D2(B2)] specifies a starting address in storage where the data is located.

5. The first address field [D1(B1)] designates a starting address in storage where the data will be placed.
6. The data is moved left to right through each field one byte at a time.
7. The data being transferred in a left to right sequence, a byte at a time, allows the propogation of a byte through storage.
8. This continues until all specified bytes are transferred.

EXAMPLES

1. MVC (Move Characters)
   Move the contents of storage location 1000(DATA through 1007 to storage locations 1100 (DATA+256) through 1107. (GPR F = 00 00 10 00)

   SYMBOLIC MVC DATA+256(8),DATA MACHINE D2 07 F1 00 F0 00

		Before	After
Storage	1000	00 33 33 33	00 33 33 33
	1004	22 22 22 22	22 22 22 22
	1008	11 11 11 11	11 11 11 11
	1100	00 00 00 00	00 33 33 33
	1104	CE 1F 00 00	22 22 22 22
	1108	39 12 1A BC	39 12 1A BC

2. MVC (Move Characters)
   Propogate the byte located at storage location 1000(DATA) through storage to location 100B. (GPR F = 00 00 10 00)

   SYMBOLIC MVC DATA+1(11),DATA MACHINE D2 0A F0 01 F0 00

		Before	After
Storage	1000	00 33 33 33	00 00 00 00
	1004	22 22 22 22	00 00 00 00
	1008	11 11 11 11	00 00 00 00

CONDITION CODE

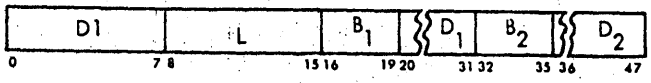
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

Move Numerics

MVN $D_1(L, B_1), D_2(B_2)$ [SS]



1. The length field (L) specifies the length in bytes of the first and second operands.
2. The maximum number of numerics (low order 4 bits of a byte) that can be specified by the length field is 256.
3. The hexadecimal value of the length field is always one less than the numeric actually moved.
4. The first address field D1(B1) specifies a starting address in storage where the numeric is placed.
5. The second address field D2(B2) designates a starting address in storage where the numeric is located.
6. The low-order 4 bits of the byte specified by the second address field are moved left to right one numeric at a time.
7. They are placed inthe low-order 4 bits of the byte specified by the first address field.
8. The fields may overlap in any desired manner.

EXAMPLES

1. MVN (Move Numerics)
   Move the numeric portion of storage address 100C (DATA+12) through 100F to location 1020 (DATA+32) through 1023 (GPR F = 00 00 10 00).

   SYMBOLIC MVN DATA+32(4),DATA+12 MACHINE D1 03 F0 20 F0 0C

		Before	After
Storage	100C	F1 F2 F3 F4	F1 F2 F3 F4
	1020	F9 F8 F3 C1	F1 F2 F3 C4

CONDITION CODE

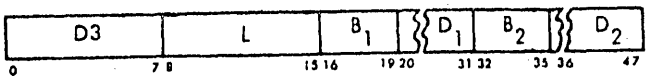
1. Remains unchanged

PROGRAM INTERRUPTION

1. Protection
2. Addressing

Move Zones

MVZ $D_1(L, B_1), D_2(B_2)$ [SS]



1. The length field (L) specifies the length in bytes of the first and second operand.
2. The maximum number of zones (high order 4-bits of a byte) that can be specified by the length field is 256.
3. The hexidecimal value of the length field is always one less that the number of zones actually moved.
4. The first address field [D1(B1)] specifies a starting address in storage where the zones will be placed.
5. The second address field [D2(B2)] designates a starting address in storage where the zones are located.
6. The high-order 4 bits of the byte specified by the second address field are moved left to right one zone at a time.
7. They are placed in the high-order 4 bits of the byte specified by the first address field.
8. The fields may overlap in any desired manner.

EXAMPLES

1. MVZ (Move Zones)
   Move the zone portion of storage address 100F (DATA+15) to location 1023 (DATA+35) (GPR F = 00 00 10 00)

```
SYMBOLIC  MVZ DATA+35(1),DATA+15
MACHINE   D3 00 F0 23 F0 0F
```

		Before	After
Storage	100C	F1 F2 F3 F4	F1 F2 F3 F4
	1020	F1 F2 F3 C4	F1 F2 F3 F4

CONDITION CODE

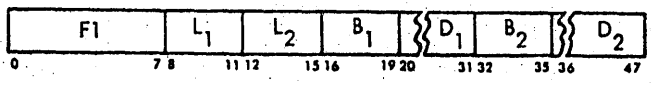
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

Move with Offset

MVO $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The second operand [D2 (L2,B2] is placed in the first operand [D1(L1,B1] location.
2. The second operand is placed to the left of and adjacent to the low-order four bits of the first operand's field.
3. The resultant field is a combination of the second operand and the low-order four bits of the first operand.
4. The fields are processed right-to-left and may overlap in any desired manner.
5. A first operand field length greater than, can be occupied by the second operand, is supplied with high-order zeros.
6. A second operand field length greater than the first operand field causes the remaining bytes to be ignored.

EXAMPLE

1. Move the 6-byte data field beginning at storage location 4200 (NUMB) to the 9-byte field at storage location 4600 (PLUS) and allot a position value to the resultant field. (GPR F = 00 00 40 00)

   SYMBOLIC MVO PLUS (9),NUMB(6)

   MACHINE F1 85 F6 00 F2 00

		Before	After
Storage	4200	12 34 56 78	12 34 56 78
	4204	90 09 8D 0C	90 09 8D 0C
	4600	55 44 33 22	00 00 01 23
	4604	11 00 11 22	45 67 89 00
	4608	3C 13 79 26	9C 13 79 26

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

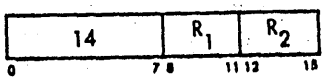
1. Protection
2. Addressing

LOGICAL FAMILY

AND

The AND instruction finds the logical product of the bits contained in the locations specified by the first and second address fields. The operands are treated as unstructured logical quantities, and the AND is applied bit by bit. A bit position in the result will be made equal to one if the corresponding bit positions in both operands are equal to one. If these conditions are not met, that bit position will be made a zero. The bit-by-bit result is placed in location specified by the first address field.

NR R_1, R_2 [RR]



1. The contents of the register specified by the second address field (R2) are ANDED with the contents of the register specified by the first address field (R1).
2. The result replaces the first operand.
3. ANDING is performed one bit at a time until all bits have been ANDED.
4. ANDING is commonly used to set any one bit or a group of bits to zero.

EXAMPLE

1. NR (AND Registers)
   AND the contents of register 6 to the contents of register A.

```
SYMBOLIC  NR 10,6     MACHINE     14 A6

                          Before          After

  GPR     6        AF AF AF AF     AF AF AF AF
  GPR     A        5F 5F 5F 0C     0F 0F 0F 0C

  Condition Code   1
```

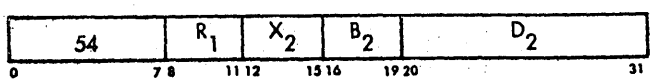
CONDITION CODE

0	Result is 0
1	Result not 0
2	--
3	--

PROGRAM INTERRUPTIONS

1. None

N $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the location designated by the second address field [D2(X2,B2)] are ANDED with the contents of the register specified by the first address field (R1).

2. The result replaces the first operand.
3. ANDING is performed one bit at a time until all bits are ANDED.
4. ANDING is commonly used to set any one bit or group of bits to 0.

EXAMPLES

1. N (AND)
   AND the contents of location 5000 (Bits) to register 8 (GPR F = 00 00 50 00)

   SYMBOLIC N 8,Bits MACHINE 54 80 F0 00

		Before	After
GPR	8	3C 3C 3C C3	00 00 00 C3
Storage	5000	C3 C3 C3 C3	C3 C3 C3 C3
Condition Code		1	

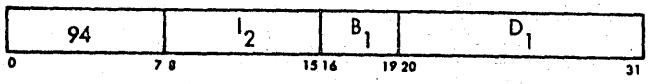
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection (Fetch only)
2. Addressing
3. Specification

NI $D_1(B_1), I_2$ [SI]



1. The second operand (I2), contained within the instruction, is ANDED with the single byte designated by the first address field. [D1(B1)].
2. The result replaces the byte specified by the first address field.
3. ANDING is performed one bit at a time until all bits have been ANDED.
4. ANDING is commonly used to set any one bit or group of bits to 0.

EXAMPLES

1. NI (AND Immediate)
   Set bit 5 at location 5006 (Bits+6) to 0.
   (GPR F = 00 00 50 00)

   SYMBOLIC NI Bits+6,X'FB' MACHINE 94 FB F0 06

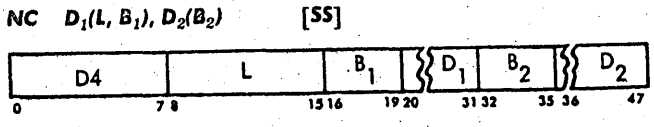
	Before	After
Storage 5004	CB 91 FE 01	CB 91 FA 01
Condition Code	1	

CONDITION CODE

0	Result is 0
1	Result not 0
2	--
3	--

PROGRAM INTERRUPTIONS

1. Protection (store only)
2. Addressing



1. The contents of the location designated by the second address field [D2(B2)] are ANDED with the contents of the location designated by the first address field [D1(B1)]
2. The number of bytes ANDED is specified by the length field (L).
3. The result replaces the first operand.
4. ANDING is performed one bit at a time until all bits have been ANDED.
5. ANDING is commonly used to set any one bit or group of bits to 0.

EXAMPLES

1. NC (AND Characters)
   AND the contents of storage location 5008 (Bits+8) through 500B to 500C (Bits+12) through 500F (GPR F = 00 00 50 00)

SYMBOLIC NC Bits+12(4),Bits+8

MACHINE D4 03 F0 0C F0 08

		Before	After
Storage	5008	00 00 00 FC	00 00 00 FC
	500C	1A 23 96 4D	00 00 00 4C
Condition Code		1	

CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

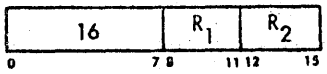
1. Protection
2. Addressing

LOGICAL FAMILY

OR

The OR instruction computes the logical sum of the bits contained in the locations specified by the first and second address fields. Operands are treated as unstructured logical quantities, and the inclusive OR is applied bit by bit. A bit position in the result will be made a one (1) if the corresponding bit position in either operand is equal to one (1). Both bit positions being equal to zero set that result bit to a zero. The bit-by-bit result is placed in location specified by the first address field.

OR R_1, R_2 [RR]



1. The contents of the register specified by the second address field (R2) are OR'ED with the contents of the register specified by the first address field (R1).
2. The result replaces the first operand.
3. The corresponding bit position in the result will be made a one if either of the bit positions in the operands is equal to a one.
4. If neither of the bit positions in the operands is equal to one, the corresponding bit position in the result will be made a zero.
5. OR'ING is performed a bit at a time until all bits have been OR'ED.
6. OR'ING is commonly used to set any one bit or group of bits to a one.

EXAMPLES

1. OR (OR Registers)
   OR the contents of register 4 to the contents of register 1.

   SYMBOLIC OR 1,4 MACHINE 1614

		Before	After
GPR	1	A5 A5 A5 A5	FF FF FF FF
GPR	4	5A 5A 5A 5A	5A 5A 5A 5A

   Condition Code 1

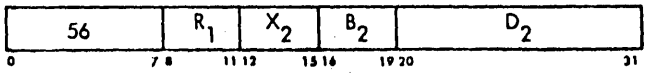
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUTPION

1. None

O $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the storage locations specified by the second address field [D2(X2,B2)] is OR'ED with the contents of the register specified by the first address field (R1).
2. The result replaces the first operand (R1).
3. The corresponding bit position in the result will be made a one if either of the bit positions in the operand is equal to a one.
4. If neither of the bit positions in the operands is equal to one, the corresponding bit position in the result will be made a zero.
5. OR'ING is performed a bit at a time until all bits have been OR'ED.
6. OR'ING is commonly used to set any one bit or group of bits to a one.

EXAMPLES

1. O (OR)
   OR the contents of storage location 5010 (Bits+16) to the contents of register 0. (GPR F = 00 00 50 00)

   SYMBOLIC O 0,Bits+16 MACHINE 56 00 F0 10

		Before	After
GPR	0	01 02 03 04	11 43 6F 0E
Storage	5010	10 43 6C 0E	10 43 6C 0E
Condition Code		1	

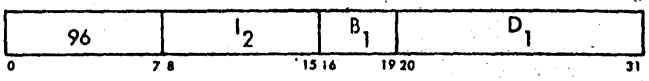
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing
3. Specification

OI $D_1(B_1), I_2$ [SI]



1. The second operand (I2), contained within the instruction, is OR'ED with the contents of the storage location specified by the first address field [D1(B1)].
2. The result replaces the first operand.
3. The corresponding bit position in the result will be made a one if either of the bit positions in the operands is equal to a one.
4. If neither of the bit positions in the operands is equal to one, the corresponding bit position in the result will be made a zero.
5. OR'ING is performed a bit at a time until all of the bits have been OR'ED.
6. OR'ING is commonly used to set any one bit or group of bits to a one.

EXAMPLES

1. OI (OR Immediate)
   Set bit 5 of storage location 5017 (Bits+23) to a one. (GPR F = 00 00 50 00)

   SYMBOLIC OI Bits+23,4 MACHINE 96 04 F0 17

	Before	After
Storage 5014	00 00 00 C3	00 00 00 C7
Condition Code	1	

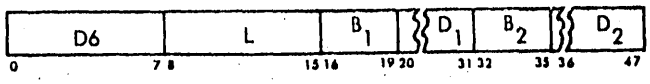
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTION

1. Protection
2. Addressing

OC $D_1(L, B_1), D_2(B_2)$ [SS]



1. The contents of the storage location specified by the second address field [D2(B2)] are OR'ED with the contents of the storage location designated by the first address field [D1(B1)].
2. The number of bytes to be OR'ED is specified by the length field (L).
3. The result replaces the first operand.
4. The corresponding bit position in the result will be made a one if either of the bit positions in the operand is equal to a one.
5. If neither of the bit positions in the operands is equal to a one, the corresponding bit position in the result will be made a zero.
6. OR'ING is performed a bit at a time until all bits have been OR'ED.
7. OR'ING is commonly used to set any one bit or group of bits to a one.

EXAMPLES

1. OC (OR Characters)
   OR 6 bytes starting at storage location 5020 (Bits+32) to 6 bytes beginning at location 5028 (Bits+40). (GPR F = 00 00 50 00)

   SYMBOLIC OC Bits+40(6),Bits+32

   MACHINE D6 05 F0 28 F0 20

		Before	After
Storage	5020	80 00 00 02	80 00 00 02
	5024	00 00 FC 3E	00 00 FC 3E
	5028	40 00 00 01	C0 00 00 03
	502C	00 00 13 29	00 00 13 29

   Condition Code 1

CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

LOGICAL FAMILY

EXCLUSIVE OR

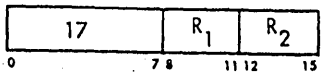
The EXCLUSIVE OR instruction is used to find the modulo two sum of the bits of two operands. The second address field specifies the location of data that will be EXCLUSIVE OR'ED with data at the location specified by the first address field. Operands are treated as unstructured logical quantities, and the EXCLUSIVE OR is applied bit by bit. A bit position in the result will be made a 1 if either (not both) of the corresponding bit positions in the operands is equal to a one. Both bit positions being equal to zero or one set the result bit position to a 0. The bit-by-bit result replaces the first operand.

The expression, "modulo two sum," means (to us) "What is left when any two is cast out." The following table shows the modulo two sum (exclusive OR).

```
0+0=0          1001     1111
0+1=1        +1010    +0111
1+0=1          0011     1000
```

The strange part is that one plus one is zero.

XR R_1, R_2 [RR]



1. The contents of the register specified by the second address field (R2) are EXCLUSIVE OR'ED with the contents of the register specified by the first address field (R1).
2. The result replaces the first operand.
3. The corresponding bit position in the result will be made a one if either (not both) of the bit positions in the operands is equal to a one.
4. If both bit positions are equal to a one or zero, the corresponding bit positions in the result will be made a zero.
5. The EXCLUSIVE OR'ING is performed one bit position at a time until all bits have been Exclusive OR'ED.
6. Any field EXCLUSIVE OR'ED with itself becomes all zeros.
7. EXCLUSIVE OR'ING is commonly used to invert any one bit or group of bits.

EXAMPLES

1. XR (EXCLUSIVE OR Registers)
   EXCLUSIVE OR the contents of register 9 to the contents of register C.

   SYMBOLIC XR 12,9 MACHINE 17 C9

		Before	After
GPR	9	FF 11 CC 55	FF 11 CC 55
GPR	C	11 22 88 AA	EE 33 44 FF

   Condition Code 1

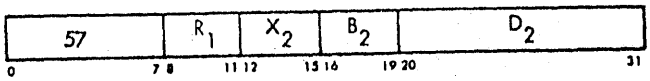
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUTPIONS

1. None

X $R_1, D_2(X_2, B_2)$ [RX]



1. The contents of the storage location specified by the second address field [D2(X2,B2)] are EXCLUSIVE OR'ED with the contents of register specified by the first address field (R1).
2. The result replaces the first operand.
3. The corresponding bit positions in the result will be made a one if either (not both) of bit positions in the operands is equal to a one.
4. If both bit positions are equal to a one or zero, the corresponding bit positions in the result will be made a 0.
5. The EXCLUSIVE OR'ING is performed one bit position at a time until all bits have been EXCLUSIVE OR'ED.
6. Any field EXCLUSIVE OR'ED with itself becomes all zeros.
7. EXCLUSIVE OR'ING is commonly used to invert any one bit or group of bits.

EXAMPLE

1. X (EXCLUSIVE OR)
   EXCLUSIVE OR the contents of storage location 5050 (Bits+80) to register 1. (GPR F = 00 00 50 00)

   SYMBOLIC X 1,Bits+80 MACHINE 57 10 F0 50

		Before	After
GPR	1	FF 00 FF 00	FF 00 00 FF
Storage	5050	00 00 FF FF	00 00 FF FF
Condition Code		1	

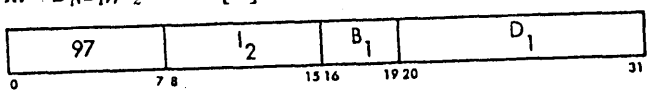
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
3. Addressing
3. Specification

XI $D_1(B_1), I_2$ [SI]



1. The second operand (I2,) contained within the instruction, is EXCLUSIVE OR'ED with the contents of the storage location designated by the first address field [D1(B1)].
2. The result replace the first operand.
3. The corresponding bit position in the result will be made a one if either (not both) of the bit positions in the operands is equal to a one.
4. If both bit positions are equal to a one or zero the corresponding bit position in the result will be made a zero.
5. The EXCLUSIVE OR'ING is performed one bit position at a time until all bits have been EXCLUSIVE OR'ED.
6. Any field EXCLUSIVE OR'ED with itself becomes all zeros.
7. EXCLUSIVE OR'ING is commonly used to invert any one bit or group of bits.

1. XI (EXCLUSIVE OR Immediate)
   Invert bit 0 and 3 of storage location 5056 (Bits+86). (GPR F = 00 00 50 00)

   SYMBOLIC XI Bits+86,X'90' MACHINE 97 90 F0 56

	Before	After
Storage 5054	C3 01 0F 23	C3 01 9F 23
Condition Code	1	

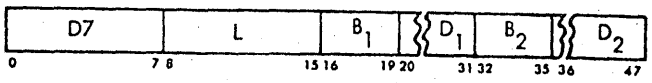
CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection (store only)
2. Addressing

XC $D_1(L, B_1), D_2(B_2)$ [SS]



1. The contents of the storage locations specified by the second address field [D2(B2)] are EXCLUSIVE OR'ED with the contents of storage location designated by the first address field [D1(B1)].
2. The number of bytes that will be EXCLUSIVE OR'ED is designated by the length field (L).
3. The result replaces the first operand.
4. The corresponding bit position in the result will be made a one if either (not both) of the bit positions in the operands is equal to a one.
5. If both bit positions are equal to a one or zero, the corresponding bit positions in the result will be made a zero.
6. The EXCLUSIVE OR'ING is performed one bit position at a time until all bits have been EXCLUSIVE OR'ED.
7. Any field EXCLUSIVE OR'ED with itself becomes all zeros.
8. EXCLUSIVE OR'ING is commonly used to invert any one bit or group of bits.

EXAMPLE

1. XC (EXCLUSIVE OR Character)
   EXCLUSIVE OR storage locations 5050 (Bits+80 through 5057 with itself. (GPR F _ 00 00 50 00)

   SYMBOLIC XC Bits+80(8),Bits+80

   MACHINE D7 07 F0 50 F0 50

		Before	After
Storage	5050	00 00 FF FF	00 00 00 00
	5054	C3 01 9F 23	00 00 00 00
Condition Code		0	

CONDITION CODE

0 Result is 0
1 Result not 0
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

LOGICAL FAMILY

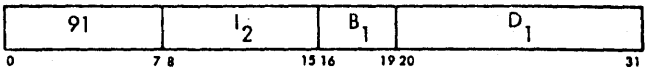
TEST UNDER MASK

The TEST UNDER MASK instruction is normally used to test the condition of a bit or group of bits within a byte. The byte to be tested is specified by the first address field. The bits to be tested are designated by the mask field. The result of this test is used to set the Condition Code.

Storage remains unchanged as a result of the TM instruction.

Test Under Mask

TM $D_1(B_1), I_2$ [SI]



1. The location of the byte to be tested is specified by the first address field (D1,B1).
2. The second operand (I2) is a mask field contained within the instruction.
3. This mask field determines the bits of the byte to be tested.
4. The result of this test determines the setting of the Condition Code.

EXAMPLES

1. TM (Test Under Mask)
   Using this instruction, test bits 4 through 7 of storage location 4500 (SWTH). (GPR F = 00 00 45 00)

   SYMBOLIC TM 0(15),X'0F' 91 0F F0 00

	Before	After
Storage 4500	08 04 8C 0D	08 04 8C 0D
Condition Code	1	

2. TM
   Test Bits 28 and 29 of storage location 4500 (FWTH). (GPR F = 00 00 45 00)

   SYMBOLIC TM 3(15),X'0C' MACHINE 91 0C F0 03

	Before	After
Storage 4500	07 04 AC 0D	07 04 AC 0D
Condition Code	3	

CONDITION CODE

0 Selected bits all 0; mask result 0
1 Selected bits mixed 0 and 1
2 --
3 Selected bits all 1

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing

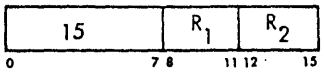
LOGICAL FAMILY

COMPARE LOGICAL

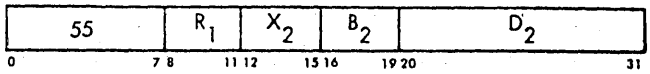
The COMPARE-LOGICAL instruction is used to determine the similarity or difference between two operands. The contents of a location specified by the first address field are compared to the contents of the location designated by the second address field. The Condition Code is set to reflect the similarity or difference between the two operands.

Compare Logical

CLR R_1, R_2 [*RR*]



CL $R_1, D_2(X_2, B_2)$ [*RX*]



1. The contents of the register specified by the first address field (R1) is compared to the contents of the location designated by the second address field (R2 or D2 (X2,B2)).
2. The logical comparison is left to right a bit at a time until an inequality is found or the field are completed.
3. The result of the comparison determines the setting of the Condition Code.
4. Both operands remain unchanged.

EXAMPLES

1. CLR (Compare Logical Registers)
   Logically compare the contents of register 6 to the contents of register 1.

SYMBOLIC	CLR 6,1	MACHINE	1561
		Before	After
GPR	1	07 3C 1A 28	07 3C 1A 28
GPR	6	00 0F 1C 29	00 0F 1C 29
Condition Code		1	

2. CL (Compare Logical)
   Logically compare the contents of register 1 to the contents of storage location 4000 (TEST). (GPR F = 00 00 40 00)

SYMBOLIC CL 1,TEST MACHINE 55 10 F0 00

		Before	After
GPR	1	07 3D 1D 2F	07 3D 1D 2F
Storage	4000	F7 3E 50 00	F7 3E 50 00
Condition Code 2			

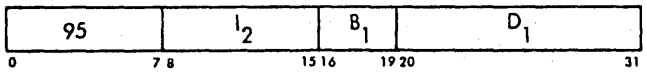
CONDITION CODE

0	Operands are equal
1	First operand is low
2	First operand is high
3	--

PROGRAM INTERRUPTIONS

1. Protection (fetch CL only)
2. Addressing (CL only)
3. Specification (CL only)

CLI $D_1(B_1), I_2$ [SI]



1. The contents of the storage location designated by the first address field (D1 (B1)) is compared to the second operand (I2).
2. The logical comparison is left to right a bit at a time until an inequality is found or the fields are completed.
3. The result of the comparison determines the setting of the Condition Code.
4. Both operands remain unchanged.

EXAMPLES

1. CLI (Compare Logical Immediate)
   Logically compare bits 0 - 9 of storage location 4007 (TEXT+7) to 0E (GPR F = 00 00 40 00)

   SYMBOLIC CLI TEXT+7,X'0E' MACHINE 95 0E F0 07

	Before	After
Storage 4004	FC 23 0E 16	FC 23 0E 16
Condition Code	2	

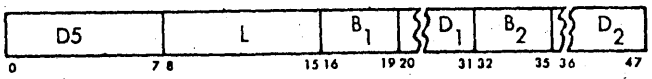
CONDITION CODE

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing

CLC $D_1(L, B_1), D_2(B_2)$ [SS]



1. The contents of storage locations beginning with the location designated by the first address field (D1(B1)) are compared to the contents of the storage locations beginning with the location designated by the second address field (D2(B2)).
2. The number of bytes compared is determined by the length field (L) in the instruction.
3. The logical comparison is left to right a bit at a time until an inequality is found or the fields are completed.
4. The result of the comparison determines the setting of the Condition Code.
5. Both operands remain unchanged.

EXAMPLE

1. CLC (Compare Logical Character)
   Logically compare the contents of storage location 4000 (Text) through 4007 with storage locations 4022 (Text+34) through 4029. (GPR F = 00 00 40 00)

   SYMBOLIC CLC TEXT(8),TEXT+34 MACHINE D5 07 F0 00 F0 22

		Before	After
Storage	4000	07 3C 5F FF	07 3C 5F FF
	4004	FC 23 0E 16	FC 23 0E 16
	4020	36 47 07 3C	36 47 07 3C
	4024	5F FF FC 23	5F FF FC 23
	4028	0E 16 37 2A	0E 16 37 2A
Condition Code		0	

CONDITION CODE

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing

LOGICAL FAMILY

SHIFT LOGICAL

The LOGICAL SHIFT is used primarily to move logical data within a register or an EVEN-ODD register pair. The differences between the logical and the algebraic shift is the absence of the Condition Code setting and the participation of the sign-bit position in all logical shifts.

The first address field specifies a register or an EVEN-ODD register pair that contains the data to be shifted. The second address field does not reference a storage location, but is used to generate an effective address. The decimal value of the low-order 6 bits of this address determines the number of bit positions the data will be shifted.

Register data can be edited by using the shift instructions. Extraneous data can be deleted and/or relevant data can be positioned in single or double registers.

SLL $R_1, D_2(B_2)$ [RS]

1. The contents of the register specified by the first address field (R1) are shifted to the left.
2. The low order 6 bits of the effective address [D2(B2)] specifies the number of positions that the first operand will be shifted.
3. All 32 bits participate in the shift and 0's are placed in the vacated low-order bit positions of the first operand.
4. When the low order 6 bits of the effective address exceed a decimal value of 31, the entire integer will be shifted out of the specified register.
5. The entire integer being shifted out of the specified register results in all 0's.

EXAMPLES

1. SLL (Shift Left Logical)
   Shift the entire contents of register 3 6 positions to the left.

   SYMBOLIC SLL 3,6(0) MACHINE 89 30 00 06

		Before	After
GPR	3	80 03 16 28	00 C5 8A 00

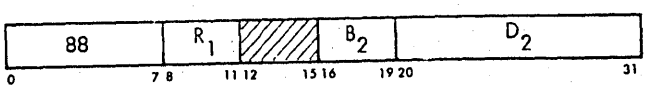
CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. None

SRL $R_1, D_2(B_2)$ [RS]



1. The contents of the register specified by the first address field (R1) is shifted to the right.
2. The low-order 6 bits of the effective address [D2(B2)] specifies the number of positions that the first operand will be shifted.
3. All 32 bits participate in the shift and zeroes are placed in the vacated high-order bit positions of the first operand.
4. When the low-order 6 bits of the effective address exceed a decimal value of 31, the entire integer will be shifted out of the specified register.
5. The entire integer being shifted out of the specified register results in that register containing all 0's.
6. Low-order bits are shifted out without inspection and are lost.

EXAMPLES

1. SRL (Shift Right Logical)
   Using register 4 as a indirect shift specification, perform a SRL on register C.

   SYMBOLIC SRL 12,0(4) MACHINE 88 C0 40 00

		Before	After
GPR	4	00 00 00 20	00 00 00 20
GPR	C	96 42 AA AE	00 00 00 00

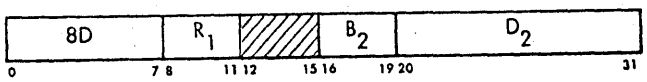
CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. None

SLDL $R_1, D_2(B_2)$ [RS]



1. The contents of the EVEN-ODD register pair specified by the first address field (R1) are shifted to the left.
2. The low-order 6 bit of the effective address [D2(B2)] specify the number of positions that the first operand will be shifted.
3. The operand is treated as 64 logical bits.
4. All 64 bits participate in a shift and 0's are placed in the vacated low-order bit positions of the EVEN-ODD register pair.

5. When the low order 6 bits of the effective address exceed a decimal value of 63 the entire integer will be shifted out of the specified registers.
6. A specification exception will occur if the register address specified by the first address field is ODD.

EXAMPLE

1. SLDL (Shift Left Double Logical)
Shift the contents of register pair 8 and 9 two positions to the left.

SYMBOLIC SLDL 8,2(0) MACHINE 8D 80 00 02

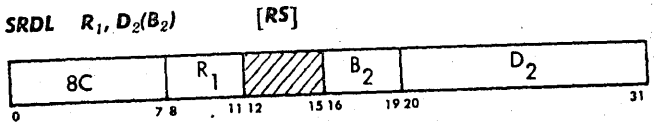
		Before	After
GPR	8	01 22 33 44	04 88 CD 11
GPR	9	55 66 77 88	55 99 DE 20

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Specification



1. The contents of the EVEN-ODD register pair specified by the first address field (R1) are shifted to the right.
2. The low-order 6 bits of the effective address [D2(B2)] specify the number of positions that the first operand will be shifted.
3. The operand is treated as 64 bits of logical data.
4. All 64 bits participate in a shift and 0's are placed in the vacated high-order bit positions of the EVEN-ODD register pair.
5. When the low order 6 bits of the effective address exceed a decimal value of 63, the entire integer will be shifted out of the specified registers.
6. A specification exception will occur if the register address specified by the first operand is ODD.

EXAMPLE

1. SRDL (Shift Right Double Logical)
   Shift the contents of register pair E and F the number of positions specified by the contents of register 1.

   SYMBOLIC SRDL 14,0(1) MACHINE 8C E0 10 00

		Before	After
GPR	1	00 00 00 04	00 00 00 04
GPR	E	86 00 00 07	08 60 00 00
GPR	F	2F C1 39 04	72 FC 13 90

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Specification

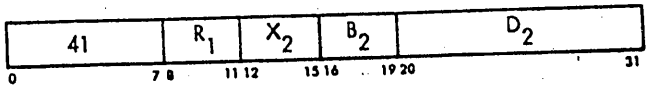
LOGICAL FAMILY

LOAD ADDRESS

The LOAD ADDRESS instruction provides an efficient method for placing a constant into one of the general registers.

The first address field specifies the destination of the constant. The second address field does not provide an address, but is instead used to supply the desired constant by loading the effective address (low-order 24 bits) into the registers specified by the first operand.

LA $R_1, D_2(X_2, B_2)$ [RX]



1. The low-order 24 bits of the generated-effective address [D2(X2,B2)] are loaded into the register specified by the first address field (R1).
2. The high-order 8 bits of the register are made equal to zero.

EXAMPLES

1. LA (Load Address)
   Load a constant of 4 into register 1 without specifying a base or index register.

   SYMBOLIC LA 1,4(0,0) MACHINE 41 10 00 04

		Before	After
GPR	1	31 8C 1A 06	00 00 00 04

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. None

TRANSLATE

The TRANSLATE instruction provides a means of easily converting one code or set of characters into another code or set of characters. This is accomplished by use of a table in storage and a knowledge of the collating sequence. The table must be constructed prior to the use of the Translate Instruction by the programmer.

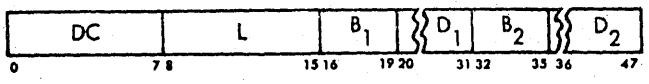
The collating sequence is the decimal value of all characters arranged in sequence by their values. This value, which is directly taken from its binary bit configuration, is used to place that character in its alpha-numerical order. This alpha-numerical order is the collating sequence.

The table is constructed using the collating sequence of the code or characters you are translating (argument bytes). The table contains the code or characters into which you are translating (function bytes). The table is formed by placing the correct function byte at the collating value of each argument byte.

The table is completed when each argument byte or character combination has the desired function byte at its collating value in the table. This table's starting address is specified by the second address field.

The argument bytes are added to the starting address of the table a byte at a time. The function byte located at the resultant address replaces the argument byte. This continues until the value specified by the length field is reached.

TR $D_1(L, B_1), D_2(B_2)$ [SS]



1. Construct a table in storage of the function bytes using the argument bytes collating sequence.
2. The second address field (B2,D2) specifies the starting address of the table.
3. The first address field (B1,D1) designates the starting address of the argument bytes.
4. The number of bytes to be translated is specified by the length field (L).
5. An argument byte is fetched and added to the starting address of the table.
6. The function byte at the resultant address replaces the argument byte.
7. This continues until the number of bytes processed equals the value specified by the length field (L).

EXAMPLE

1. TR (Translate)
   Translate 8 EBCDIC bytes into USASCII-8 equivalent. The bytes are located at location 1500 (BCD) and the table is located at location 1000(TABLE). (GPR F = 00 00 10 00)

   SYMBOLIC TR BCD(8),TABLE

   MACHINE DC 07 F5 00 F0 00

	Before	After
Storage 1500	40 D7 E8 D7	40 B0 B9 B0
	61 F3 F6 F0	4F 53 56 50

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

TRANSLATE AND TEST

The TRANSLATE AND TEST instruction is used to scan a field for delimiters or any character that has been assigned a special meaning by the programmer. It is not used to translate data as the name implies.

A table is constructed by the programmer prior to issuing this instruction. This table contains non-zero function bytes which are placed in the collating sequence of the special characters. All other positions of the table are made zero.

The starting address of the argument bytes is designated by the first address field. The second address field specifies the starting address of the table.

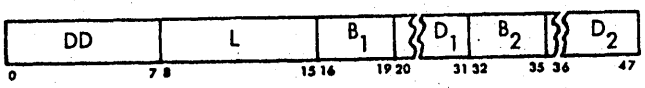
The numerical value of the argument byte is added to the starting address of the table. If the function byte at that location is zero, the operation continues by fetching the next argument byte and adding the value of that byte to the starting address of the table. This continues until a function byte containing non-zeros is encountered. When this occurs, the address of the argument byte that encountered a non-zero in the table is inserted in the low-order 24 bits of register 1. The high-order 8 bits of the register remain unchanged. The non-zero function byte is placed in the low-order 8 bits of register 2. The high-order 24 bits of this address remain unchanged. The Condition Code would be set to 1 if the scan did not complete, or 2 if it did complete.

If the scan completes (length field exhausted) and no non-zero function bytes are encountered, the Condition Code will be set to 0.

The address in register 1 is used to determine the number of argument bytes that have been scanned. The low-order 8 bits in register 2 can be tested to see what special characters had been encountered.

This means that we can, with a single instruction, inspect a complete field of argument bytes, looking for whatever interests us: error characters, end-of-message codes, blocks, commas, delimiters or whatever.

TRT $D_1(L, B_1), D_2(B_2)$ [SS]



1. Construct a table in storage of the special characters, using the argument bytes collating sequence.
2. The second address field (B2,D2) specifies the starting address of the table.
3. The first address field (B1,D1) designates the starting address of the argument bytes.
4. The number of bytes to be scanned is specified by the length field.
5. An argument byte is fetched and added to the starting address of table.
6. If the function byte at the resultant location is zeros, the next argument byte is fetched and the operation continues.
7. If a function byte of non-zeros is encountered, that byte is placed in the low-order eight positions of register 2. The argument byte's address is loaded into the low-order 24-bit positions of register 1.

EXAMPLE

1. TRT (Translate and Test)
   Scan 8 argument bytes beginning at 2500 (BYTES).
   The starting address of the table is 2000 (TABLE1).
   (GPR F = 00 00 20 00)

   SYMBOLIC TRT BYTES(8),TABLE1

   MACHINE DD 07 F5 00 F0 00

		Before	After
GPR	1	FF FF FF FF	FF FF FF FF
GPR	2	FF FF FF FF	FF FF FF FF
Storage	2500	B2 C3 B4 63	B2 C3 B4 63
		E4 C8 F2 07	E4 C8 F2 07

Translate and Test Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2040	00	00	00	00	00	00	00	00	00	00	00	00	10	20	25	00
2050	90	00	00	00	00	00	00	00	00	00	00	30	35	40	45	00
2060	80	85	00	00	00	00	00	00	00	00	00	50	55	00	00	00
2070	00	00	00	00	00	00	00	00	00	00	00	60	65	70	75	00
2080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Note: If the character codes in the statement being translated occupy a range smaller than 00_{16} through FF_{16}, a table less than 256 bytes can be used.

Condition Code 0

2. TRT
   Using the same table scan 8 argument bytes beginning at location 2508. (GPR F = 00 00 20 00)

   SYMBOLIC TRT Bytes+8(8),TABLE 1

   MACHINE DD 07 F5 08 F0 00

		Before	After
GPR	1	FF FF FF FF	FF 00 25 0D
GPR	2	FF FF FF FF	FF FF FF 55
Storage	2508	23 F2 C3 74	23 F2 C3 74
		51 6C D7 DA	51 6C D7 DA

Condition Code 1

CONDITION CODE

0 All function bytes are zero
1 Non-zero function byte before first operand field was exhausted
2 Last function byte is non-zero
3 --

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing

BRANCHING FAMILY

EXECUTE

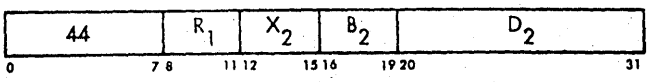
The EXECUTE instruction provides a means of performing an instruction outside of the normal instruction stream. The address of the instruction to be performed is specified by the second address field.

The first address field specifies a register whose low-order 8 bits are ORED with bits 8-15 of the instruction to be performed.If the first address field is zero, no ORING takes place. This feature gives the EXECUTE instruction great power and versatility and is commonly used with TRANSLATE and MOVE instructions to change or set their length fields. It can also be used to change index, mask, immediate data and arithmetic register values of the instruction specified by the second operand.

The specified instruction is then performed and upon completion, the program returns to the normal instruction sequence.

Execute

EX $R_1, D_2(X_2, B_2)$ [RX]



1. The instruction is used as a pointer to the instruction you wish to perform.
2. The instruction to be performed is specified by the second address field [D2(X2,B2)].
3. Bits 8-15 of that instruction are OR'ED with the low-order 8 bits of the register specified by the first address field (R1).
4. The instruction designated by the second address field is then performed and upon completion, the normal instruction sequencing continues.
5. The execution and exception handling of the designated instructions are exactly as if the instruction were obtained in normal sequential operation, except for instruction address and instruction length.
6. The instruction designated by the second address field remains unchanged in storage.

EXAMPLE

1. EX (Execute)
Execute the instruction at location 9000(TRANS) and modify bits 8-15 with the low-order 8 bits of register C. (GPR F = 00 00 90 00)

SYMBOLIC EX 12, TRANS MACHINE 44 C0 F0 00

GPR C = 26 47 3C 1E

Instruction Addressed Instruction Performed

DC 00 F3 18 F500 DC 1E F3 18 F500

Condition Code - Set by the translate instruction after execution.

2. EX
Execute the instruction at location 9006 (TRANS+6) and modify bits 8-15 with the low order 8 bits of register 9. (GPR F = 00 00 90 00)

SYMBOLIC EX 9, TRANS+6 MACHINE 44 90 F0 06

GPR 9 = 01 46 AA 16

Instruction Addressed Instruction Performed

1A24 1A36

Condition Code - Set by the ADD instruction after completion.

CONDITION CODE

1. May be set by the designated instruction.

PROGRAM INTERRUPTIONS

1. Execute
2. Protection (fetch only)
3. Addressing
4. Specification

LOGICAL FAMILY

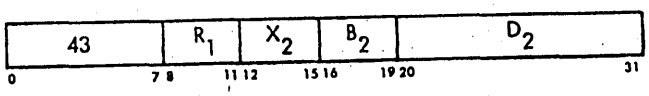
INSERT CHARACTER

We have discussed instructions which use the low-order eight bits of a register. The INSERT CHARACTER instruction provides the means for placing these eight bit characters in the low-order eight bits of a register.

The second address field designates the characters to be inserted. This character is inserted in the low-order eight bit positions of the register specified by the first operand.

This instruction is commonly used prior to the EXECUTE instruction to set the desired OR'ING field.

IC $R_1, D_2(X_2, B_2)$ [RX]



1. The character designated by the second address field [D2(X2,B2)] is placed in the low-order 8 bits of the register specified by the first address field (R1).
2. The remaining bits of the specified register are not changed.

EXAMPLE

1. IC (Insert Character)
   Place the character located at 6500(CHAR) into bits 24-31 of register 4. (GPR F = 00 00 00 00)

   SYMBOLIC IC 4,CHAR MACHINE 43 40 F5 00

		Before	After
GPR	4	EF 2C 04 7B	EF 2C 04 1F
Storage	6500	1F 37 2A ED	1F 37 2A ED

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Protection (fetch only)
2. Addressing

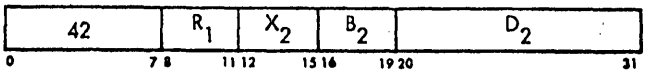
LOGICAL FAMILY

STORE CHARACTER

The STORE CHARACTER instruction enables a programmer to place the low-order 8 bits of any register into storage. This instruction can be particularly useful for the further examination of the function byte that is stored in low-order 8 bits of register 2 when performing a TRANSLATE AND TEST instruction.

The character to be stored is located in the low-order 8 bits of the register specified by the first address field. The second address field designates the location where the character will be stored.

STC $R_1, D_2(X_2, B_2)$ [RX]



1. The low-order 8 bits of the register specified by the first address field (R1) are stored at the location designated by the second address field [D2(X2,B2)].
2. The first operand remains unchanged.

EXAMPLE

1. STC (Store Character)
   Store the low-order 8 bits of register 2 into location 1400(MATCH). (GPR F = 00 00 12 00)

   SYMBOLIC STC 2,MATCH MACHINE 42 20 F200

		Before	After
GPR	2	29 00 14 27	29 00 14 27
Storage	1400	34 FE 12 9A	27 FE 12 9A

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Protection (store only)
2. Addressing

BRANCHING FAMILY

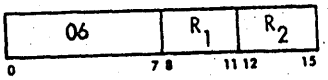
BRANCH ON COUNT

The BRANCH ON COUNT instruction permits the construction of program loops that avoid repetitive instruction sequences.

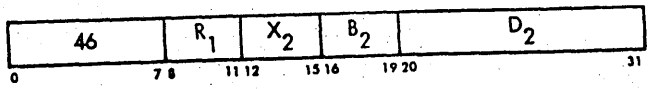
The 32-bit contents of the register specified by the first address field are algebraically reduced by 1. If the result is not 0, the program will branch to the address specified by the second address field. A result of zero permits normal instruction sequencing and no branching takes place.

The RR version of this instruction permits counting without branching if the second address field is given a value of zero.

BCTR R_1, R_2 [RR]



BCT $R_1, D_2(X_2, B_2)$ [RX]



1. The fullword contents of the register specified by the first address field (R1) are algebracially reduced by 1.
2. If the result of this subtraction is zero, the program continues with normal instruction sequencing.
3. A non-zero result (+ or-) will cause the program to branch to the location designated by the second [R2 or D2(X2,B2)].
4. If the second address field of the RR format specifies register 0, counting is performed, but no branching will occur.
5. An overflow occurring on the transition from the max negative number to the max positive number would be ignored.

EXAMPLE

1. BCTR (Branch on Count Register)
Branch* to the location in register 4 when the count in register 5 is not equal to zero.

SYMBOLIC BCTR 5,4 MACHINE 06 54

		Before	After
GPR	4	00 00 F1 04	00 00 F1 04
GPR	5	00 00 00 04	00 00 00 03

* In the above example, branching would occur.

2. BCTR (Branch on Count Register)
Perform a counting function without a branch. Register 6 will contain the count.

SYMBOLIC BCTR 6,0 MACHINE 06 60

		Before	After
GPR	6	00 00 00 00	FF FF FF FF

3. BCT (Branch on Count)
Branch* to location 2000(LOOP) when the contents of register 8 do not equal zero. (GPR F = 00 00 20 00)

SYMBOLIC BCT 8,LOOP MACHINE 46 80 F0 00

		Before	After
GPR	8	00 00 00 01	00 00 00 00

* In the above example the branch would not occur.

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. None

BRANCHING FAMILY

BRANCH ON INDEX

The incrementing and testing of the index value is the major purpose of the BRANCH ON INDEX instructions.

An increment contained in the register specified by the third address field is added to the contents of the register specified by the first address field. The result of this addition replaces the first operand.

The first operand is now compared to the third operand if the third operand is an odd register. If the third operand is an even register, the contents of the register whose address is one greater, will be used.

BRANCH ON INDEX HIGH

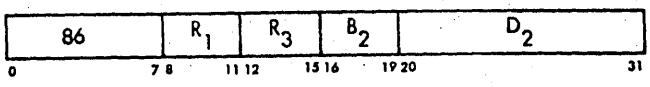
If the result of the comparison shows the sum to be greater, the next instruction address is replaced with the address specified by the second address field. The sum being low or equal results in normal instruction sequencing.

BRANCH ON INDEX LOW OR EQUAL

This instruction will continue with normal instruction sequencing if the result of the comparison shows the sum to be greater. A comparison of low or equal causes the updated instruction address to be replaced by the address specified by the second address field.

Branch On Index High

BXH $R_1, R_3, D_2(B_2)$ [RS]



1. The contents of register specified by the third address field (R3) are added to the contents of the register specified by the first address field (R1).
2. The sum replaces the first operand.
3. The first operand is now compared to the third operand, if the address of the third operand is an odd register. A third operand address specifying an even register causes the comparison to be made with the register whose address is one greater.
4. The sum being lower or equal to the comparand results in normal instruction sequencing.
5. If the sum is greater than the comparand, the updated instruction address is replaced with the effective address specified by the second address field (B2,D2).
6. The addition and comparison are performed using normal fixed point arithmetic.
7. When both the first and the comparand specifies the same location, the original contents are used for the comparand.
8. The second operand remains unchanged.

EXAMPLE

1. BXH (Branch on Index High)
Branch to location 6500(LOC) if the sum of register 4 and the index register 1 is greater than the contents of the comparand register. The following instruction will result in a branch. (GPR F = 00 00 65 00)

SYMBOLIC BXH 4,1,LOC MACHINE 86 41 F0 00

		Before	After
GPR	1	00 00 00 04	00 00 00 04
GPR	2	01 AC 0F 23	01 AC 0F 23
GPR	4	00 00 65 30	00 00 65 34

2. BXH (Branch on Index High)
Branch to location 6500(LOC) if the sum of register 4 and the index register 6 is greater than the comparand. The following instruction will not result in a branch. (GPR F = 00 00 65 00)

SYMBOLIC 4,6,LOC MACHINE 86 46 F0 00

		Before	After
GPR	4	00 00 65 80	00 00 65 84
GPR	6	00 00 00 04	00 00 00 04
GPR	7	00 00 65 A0	00 00 65 A0

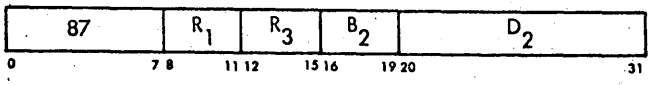
CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. None

BXLE $R_1, R_3, D_2(B_2)$ [RS]



1. The contents of register specified by the third address field (R3) are added to the contents of the register specified by the first address field (R1).
2. The sum replaces the first operand.
3. The first operand is now compared to the third operand, if the third operand is in an odd register. The third operand being in an even register causes the comparison to be made with the register whose address is one greater.
4. If the sum is lower or equal to the comparand, the updated instruction address is replaced with the effective address specified by the second address field (B2,D2).
5. The sum being greater than the comparand results is normal instruction sequencing.
6. The addition and comparison are performed using normal fixed point arithmetic.
7. When both the first and the comparand specifies the same location, the original contents are used for the comparand.
8. The second operand remains unchanged.

EXAMPLE

1. BXLE (Branch on Index Low or Equal)
   Branch to location 3200(THERE) if the sum of register 9 and the index register 2 is lower or equal to the comparand. The following instruction will result in a branch. (GPR F = 00 00 30 00)

   SYMBOLIC BXLE 9,2,THERE MACHINE 87 92 F2 00

		Before	After
GPR	2	00 00 00 04	00 00 00 04
GPR	3	00 00 30 58	00 00 30 58
GPR	9	00 00 30 54	00 00 30 58

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. None

BRANCHING FAMILY

BRANCH AND LINK

Routines are commonly used to perform repetitive tasks in programming. Maximum flexibility in utilizing these routines can only be achieved by allowing the routines to be isolated from the program. This isolated routine must be readily available to the program and assure the program that the same condition will exist within the system upon completion of the routine.

To satisfy these conditions we must have an instruction perform the following:

1. Branch to a routine specified by an address.
2. Save the updated instruction address so that we may return to the program.
3. Save the Program Mask and the Condition Code so that it may be reloaded if changed at the end of the routine.

The BRANCH AND LINK instruction satisfies all the above conditions. The second address field specifies the branch address. The updated instruction address, Condition Code, Instruction Length Code, and Program Mask are placed in the register specified by the first address field. This is accomplished by loading the designated register with the rightmost 32 bits of the PSW.

Leaving a routine and returning to the program can easily be accomplished by utilizing two instructions. The Set Program Mask* is used to reload the program mask and the Branch on Condition Register

*The SET PROGRAM MASK is to be explained.

to branch to the correct point in the program. These two instructions are normally the final two instructions of a routine.

The following is in the sequence of performance.

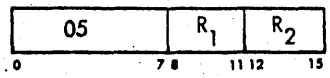
```
          PROBLEM PROGRAM                          ROUTINE

STORGE    1000 AR   3,4
          1002 S    3,TEN                          2500 L    1,PMRTNX
          1006 BAL  2,ROUTNE                       2504 SPM  1
                                          ROUTNE     2506 LR   3,4
                                                     2508 L    5,DATA
                                                     250C SR   4,5
                                                     250E SPM  2
                                                     2510 BCR  15,2
          100A C,9,CNST
          100E etc.
```

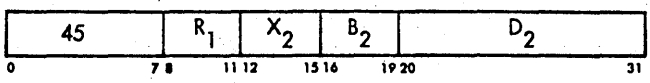
The above should clarify the use of a BRANCH AND LINK, the entry into a routine and the departure from a routine.

Branch and Link

BALR R_1, R_2 [RR]



BAL $R_1, D_2(X_2, B_2)$ [RX]



1. The rightmost 32 bits of the PSW are placed in the register specified by the first address field (R1).
2. The program branches to the address specified by the second address field [R2 or D2 (X2,B2)].
3. A branch will not occur if the register specified by the second operand is GPR 0 when using the BALR instruction.
4. The Instruction Length Code will be set to 2 if the BAL is the subject instruction of an EXECUTE.

EXAMPLE

1. BALR (Branch and Link Registers)
Branch to storage location contained in Register 4. Save the updated instruction address and the Program Mask in register 7. The PSW before executing the BALR is FE060000B9001052.

SYMBOLIC BALR 7,4 MACHINE 05 74

		Before	After
GPR	4	00 00 16 00	00 00 16 00
GPR	7	00 14 00 00	B9 00 10 54

2. BAL (Branch and Link)
Branch to storage location 2500 (RTN1). Save the Program Mask and the updated instruction address in Register 1. (GPR F = 00 00 20 10) The PSW before executing the BAL is 81060000C40014FC.

SYMBOLIC BAL 1, RTN1 MACHINE 45 10 F4 F0

		Before	After
GPR	1	01 CA 46 29	C4 00 15 00

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. None

STATUS SWITCHING FAMILY

SUPERVISOR CALL

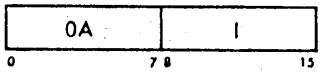
A SUPERVISOR CALL is a special instruction used by the problem programmer to force an interrupt. A SUPERVISOR CALL interrupt differs from the other classes of interrupts in that the problem program initiates the interrupt to return control of the system to the control program.

This instruction is normally issued if the problem program: has need of a common supervisor routine; has need of the control program to issue a privileged operation; has ended. A privileged operation is any instruction that, if made available to the problem programmer, would disrupt the normal control and sequencing of the system. Privileged instructions will be examined later in this course, they include Input/Output operations, protection control, and PSW controls.

The SUPERVISOR CALL instruction sends an interrupt code to the control program. The control program analyzes this interrupt code and is able to determine the particular action the problem program requires. The number and variety of these request codes (interrupt codes) that are available to the problem programmer depends entirely upon the particular control program under which the problem program will be executed.

The SUPERVISOR CALL is of the RR format, but the register fields are combined and labeled I. This field contains the interrupt code which is sent to the control program for analysis and action.

SVC I [RR]



1. This instruction forces a supervisor call interrupt.
2. The contents of the I Field are placed in bits 24-31 of the SVC old PSW.
3. The control program examines the interrupt code to determine the action requested by this code.
4. The control program then performs the specified action and if the problem program has not completed, it will return control of the system to the problem program.

EXAMPLE

1. SVC (Supervisor Call)
   Issue a supervisor call using an interrupt code of HEX 47.

   SYMBOLIC SVC X'47' MACHINE 0A 47

	Before	After
PSW Bits 0-31	E1 71 00 00	E1 71 00 47

CONDITION CODE

1. Remains unchanged in the old PSW.

PROGRAM INTERRUPTIONS

1. None

STATUS SWITCHING FAMILY

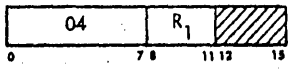
SET PROGRAM MASK

The SET PROGRAM MASK instruction is used to set bits 34-39 of the PSW. This includes both the Program Mask and the Condition Code. This instruction is the only means by which a problem programmer can change the Program Mask and Condition Code.

The first operand specifies the register which contains the new Program Mask and Condition Code. This instruction ignores the second operand.

Set Program Mask

SPM R_1 [RR]



1. Bits 2-7 of the register specified by the first address field (R1) replace bits 34-39 of the PSW.
2. Bits 0, 1 and 8-31 of this register are ignored.
3. The second operand field is not used.

EXAMPLE

1. SPM (Set Program Mask)
   Set bits 34-39 of the PSW to the contents of bits 2-7 of register 2.

   SYMBOLIC SPM 2 MACHINE 04 20

		Before	After
GPR	2	D2 00 54 04	D2 00 54 04
PSW	32-63	B4 00 59 46	92 00 59 48

CONDITION CODE

1. The code is set according to bits 2-3 of the register specified by the first address field.

PROGRAM INTERRUPTIONS

1. None

STATUS SWITCHING FAMILY

SET SYSTEM MASK

The SET SYSTEM MASK is a privileged instruction used by the control program to change the system mask of the PSW. This is done to allow or disallow interrupts from Input/Output devices or external sources.

The byte at the location designated by the first address field replaces the system mask of the current PSW. The second address field of this instruction is ignored.

SSM $D_1(B_1)$ [SI]

1. The first address field (D1, B1) specifies a byte in storage that replaces the system mask of the current PSW.
2. The second address field is ignored.
3. This is a privileged instruction and only be executed while in the supervisor state.

EXAMPLE

1. SSM (Set System Mask)
   Set the system mask from storage location 2504 (MASK). This will allow interrupts from channels 0, 1, and 4. (GPR F = 00 00 25 00)

   SYMBOLIC SSM MASK MACHINE 80 00 F0 04

	Before	After
Storage 2504	C8 D1 FF 00	C8 D1 FF 00
PSW Bits 0-31	F1 41 00 00	C8 40 00 00

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Privileged operation
2. Protection (fetch only)
3. Addressing

STATUS SWITCHING FAMILY

LOAD PSW

When an interrupt occurs, the CURRENT PSW is stored at the permanent assigned storage location for the OLD PSW of that particular interrupt. A NEW PSW is loaded by circuitry and the system will enter the SUPERVISOR state. This PSW will address an interrupt handling routine to clear the interrupt and take action as required.

Upon completion of this interrupt routine, the processing capabilities of the system are usually returned to the problem program. This is done by the privileged instruction LOAD PSW which is the final instruction of the interrupt routine. The first operand of this instruction contains the doubleword address of the permanent assigned storage location for the OLD PSW of the particular interrupt that initiated the above sequence. Bits 16-33 of this doubleword location are not loaded as the CURRENT PSW (The interrupt code and ILC are made 0). The remainder of the doubleword becomes the CURRENT PSW and instruction sequencing proceeds with the instruction address. This instruction does not use the second operand field.

LPSW $D_1(B_1)$ [SI]

1. The doubleword located at the address specified by the first address field (D1,B1) replaces the current PSW.
2. The first operand must be on a doubleword boundary.
3. Bits 16-33 of this doubleword are not transferred to the current PSW.
4. This instruction is privileged and cannot be issued by a problem program.

EXAMPLE

1. LPSW (Load Program Status Word)
   Load a Program Status Word from the Input/Output OLD PSW location as the CURRENT PSW.

   SYMBOLIC LPSW 56 MACHINE 82 00 00 38

		Before	After
CURRENT PSW	0-31	00 00 00 00	F1 41 00 00
	32-63	40 00 34 60	60 00 94 26
Storage	0038	F1 41 00 10	F1 41 00 10
	003C	A0 00 94 26	A0 00 94 26

CONDITION CODE

1. Set by bits 34 & 35 of the LOADED PSW.

PROGRAM INTERRUPTIONS

1. Privileged operation
2. Protection (fetch only)
3. Addressing
4. Specification

STATUS SWITCHING FAMILY

STORAGE KEYS

The main storage sizes available with a system are always divisable by 2,048. The quotient of this number divided into the main storage size determines the number of storage blocks within any given system. Storage blocks are a convienent means by which storage can be sectioned for use with the storage and fetch protection feature.

When a problem program is loaded, it informs the control program of its storage requirements. The control program then assigns the problem program the number of storage blocks needed for execution. The storage blocks are assigned a storage key and the problem program PSW is given the protection key to match the storage key.

SET STORAGE KEY

The SET STORAGE KEY instruction allows the control program to assign the storage blocks a storage key that will correspond with the protection key of the problem program's PSW.

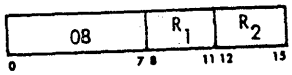
The storage block to be assigned a key is specified by bits 8-20 of the register designated by the second address field. The contents of the first operand, bits 24-27 (24-28 for fetch protection), contain the key assigned to the storage block designated by the second operand.

INSERT STORAGE KEY

The INSERT STORAGE KEY instruction is the only means by which a storage key can be inspected.

Bits 8-20 of the second operand specify the storage key to be inspected. The first address field designates the register where the key will be placed for inspection. The key will occupy bits 24-27 (24-28 for fetch protect) of the register specified by the first operand.

SSK R_1, R_2 [RR]



1. The storage block addressed by bits 8-20 of the register specified by the second address field (R2) will be given a key.
2. Bits 0-7 and 21-27 are ignored but bits 28-31 must contain zeros or a specification error will occur.
3. The key is taken from bits 24-27 (24-28 when fetch protect is installed) of the register specified by the first address field (R1).
4. The remainder of the bits in this register are ignored.
5. This is a privileged instruction that can only be issued by the control program.

EXAMPLE

1. SSK (Set Storage Key)
   Assign the storage key in register 3 to the storage address of 800-FFF. (GPR E = 00 00 08 00)

   SYMBOLIC SSK 3,14 MACHINE 08 3E

		Before	After
GPR	3	1C 23 46 7B	1C 23 46 7B
GPR	14	00 00 08 00	00 00 08 00
Storage Key 800-FFF		4	7

CONDITION CODE

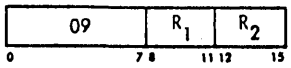
1. Remains unchanged

PROGRAM INTERRUPTIONS

1. Operation (protection feature not installed).
2. Privileged operation
3. Addressing
4. Specification

Insert Storage Key

ISK R_1, R_2 [RR]



1. The storage block addressed by bits 8-20 of the second operand (R2) supplies its key for inspection.
2. Bits 0-7 and 21-27 are ignored, but bits 28-31 must contain zeros or a specification error will occur.
3. The key is placed in bits 24-27 (24-28 when fetch protection is installed) of the first operand (R1).
4. The low-order 4 (3 if fetch protection is installed) bits of this register are set to zero and the remainder ignored.
5. This is a privileged instruction that can only be executed by the Control Program.

EXAMPLE

1. ISK (Insert Storage Key)
   Take the storage key from the storage block 3000 -37FF and place in register 1. (GPR 9 = 00 00 35 00 and fetch protect not installed)

   SYMBOLIC ISK 1,9 MACHINE 09 19

		Before	After
GPR	1	FF FF FF FF	FF FF FF FF
GPR	9	00 00 35 00	00 00 35 00
Storage Key 3000 -37FF		5	5

CONDITION CODE

1. Remains unchanged.

PROGRAM INTERRUPTIONS

1. Operation (protection feature not installed)
2. Privileged operation
3. Addressing
4. Specification

TEST AND SET

When a system is loaded with, and operating on, two or more programs; the assignment of available storage space to any one program becomes difficult. Allowing more than one program the use of the same storage area would alleviate much of this problem.

Assigning the same area to different programs requires a method of determining the "in use" status of a particular storage area. This function is the prime purpose of the TEST AND SET instruction.

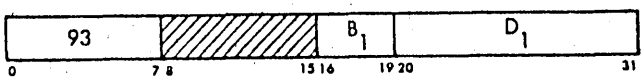
The TEST AND SET instruction can perform this task by testing a control byte. This control byte is determined by the programmer and is usually the first byte of the common storage area. The Condition Code will be set to the value of the leftmost bit in a control byte as a result of this test. If this bit contains a value of one, the common storage area is being used by another program. A bit value of zero indicates that this storage area is free for use.

Prior to setting the Condition Code the control byte is set to all ones to prevent the possibility of two programs testing the common storage area at the same time and finding it to be free for their use.

When a program finishes with a common storage area, it must reset the first bit of the control byte so that the area is again available to other programs.

The address of the control byte is specified by the first operand. The second operand field of the instruction is ignored.

TS $D_1(B_1)$ [SI]



1. The first address field (D1,B1) specifies the byte in storage whose leftmost bit will be set into the Condition Code.
2. Prior to setting this bit into the Condition Code the addressed byte is set to all ones.
3. If the instruction is a protection violation, the Condition Code results are unpredictable.

EXAMPLE

1. TS (Test and Set)
   Test storage address 2301(SAREA) to see if the common storage area is in use. (GPR F = 00 00 23 00)

   SYMBOLIC TS SAREA MACHINE 93 00 F0 01

	Before	After
Storage 2300	1C 00 23 16	1C FF 23 16
Condition Code	0	

CONDITION CODE

0 Leftmost bit of byte specified is zero.
1 Leftmost bit of byte specified is one.
2 --
3 --

PROGRAM INTERRUPTIONS

1. Protection
2. Addressing

DECIMAL INSTRUCTIONS

Decimal arithmetic operates on data in the packed format. In this format, two decimal digits are placed in one eight-bit byte.

Data are interpreted as integers, right-aligned in their fields. They are kept in true notation with a sign in the right four bits of the low-order eight-bit byte.

Processing takes place right to left between main-storage locations. All decimal arithmetic instructions use a two-address format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, indicating the number of additional bytes that the operand extends beyond the first byte.

The decimal arithmetic instruction set provides for adding, subtracting, comparing, multiplying, and dividing, as well as the format conversion of variable length operands.

The condition code is set as a result of all arithmetic and comparison operations.

DATA FORMAT

Decimal operands reside in main storage only. They occupy fields that may start at any byte address and are composed of one to 16 eight-bit bytes.

Lengths of the two operands specified in an instruction need not be the same. If necessary they are considered to be extended with zeros to the left of the high-order digits. Results never exceed the limits set by address and length specification. Lost carries or lost digits from arithmetic operations are signaled as a decimal overflow exception.

In the packed format, two decimal digits normally are placed adjacent in a byte, except for the rightmost byte of the field. In the rightmost byte a sign is placed to the right of the decimal digit. Both digits and a sign are encoded and occupy four bits each.

NUMBER REPRESENTATION

Numbers are represented as right-aligned true integers with a plus or minus sign.

The digits 0-9 have the binary encoding 0000-1001. The codes 1010-1111 are invalid as digits. This set of codes is interpreted as sign codes, with 1010, 1100, 1110, and 1111 recognized as plus and with 1011 and 1101 recognized as minus. The codes 0000-1001 are invalid as sign codes.

The sign and zone codes generated for all decimal arithmetic results differ for the Extended Binary-Coded-Decimal Interchange

Code (EBCDIC) and the USA Standard Code for Information Interchange (USASCII-8). The choice between the two codes is determined by bit 12 of the PSW. When bit 12 is zero, the preferred EBCDIC codes are generated; these are plus, 1100; minus, 1101; and zone, 1111. When bit 12 is one, the preferred USASCII-8 codes are generated; these are plus, 1010; minus, 1011; and zone, 0101.

CONDITION CODE

The results of all add-type and comparison operations are used to set the condition code. All other decimal arithmetic operations leave the code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect two types of results for decimal arithmetic. For most operations the states 0, 1, and 2 indicate a zero, less than zero, and greater than zero content of the result field; the state 3 is used when the result of the operation overflows.

For the comparison operation, the states 0, 1 and 2 indicate that the first operand compared equal, low, or high.

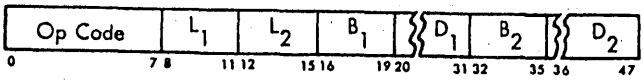
CONDITION CODE SETTING FOR DECIMAL ARITHMETIC

	0	1	2	3
Add Decimal	zero	<zero	>zero	overflow
Compare Decimal	equal	low	high	--
Subtract Decimal	zero	<zero	>zero	overflow
Zero and Add	zero	<zero	>zero	overflow

INSTRUCTION FORMAT

Decimal instructions use the following format:

SS Format



For this format, the content of the general register specified by B1 is added to the content of D1 field to form an address. This address specifies the leftmost byte of the first operand field. The number of operand bytes to the right of this byte is specified by the L1 field of the instruction. Therefore, the length in bytes of the first operand field is 1-16, corresponding to a length code in L1 of 0000-1111. The second operand field is specified similarly by the L2, B2, and D2 instruction fields.

A zero in the B1 or B2 field indicates the absence of the corresponding address component.

Results of operations are always placed in the first operand field. The result is never stored outside the field specified by the address and length. In the event the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension never modifies storage. The second operand field and the contents of general registers remain unchanged.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic

and D1(L1,B1),D2(L2,B2) the operand designation.

INSTRUCTIONS

The decimal arithmetic instructions and their mnemonics and operand codes follow. All instructions use the SS format and assume packed operands and results. The table indicates when the condition code is set and the exceptions in operand designation, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Add Decimal	AP	SS T,C	P,A, D,DF	FA
Subtract Decimal	SP	SS T,C	P,A, D,DF	FB
Zero and Add	ZAP	SS T,C	P,A, D,DF	F8
Compare Decimal	CP	SS T,C	P,A, D	F9
Multiply Decimal	MP	SS T	P,A,S,D	FC
Divide Decimal	DP	SS T	P,A,S,D,DK	FD

Notes:

A Addressing exception

C Condition Code is set

D Data exception

DF Decimal-overflow exception

DK Decimal-divide exception

P Protection exception

S Specification exception

T Decimal feature

PROGRAMMING NOTE

The moving, editing, and logical comparing instructions may also be used in decimal calculations.

DECIMAL PROGRAM INTERRUPTIONS

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in decimal arithmetic:

Operation: The decimal feature is not installed, and the instruction is ADD DECIMAL, SUBTRACT DECIMAL, ZERO AND ADD, COMPARE DECIMAL, MULTIPLY DECIMAL, DIVIDE DECIMAL, EDIT, OR EDIT AND MARK. The instruction is suppressed. Therefore, the condition code and data in storage and registers remain unchanged.

Protection: The key of an operand in storage does not match the protection key in the PSW.

The operation is terminated for either a store or a fetch violation by a decimal instruction; the result data and condition code are unpredictable.

Addressing: An address designates an operand location outside the available storage for the installation.

The operation is terminated. The result data and the condition code are unpredictable and should not be used for further computation.

These address exceptions do not apply to the components from which an address is generated: i.e. The base or index register may have a value larger than the storage size, but the effective address (EA) may not. For example, if a base register or index register contained 6A FF FF FF and the displacement was 079, the EA would be 00 0078.

Specifications: A multiplier or a divisor size exceeds 15 digits and sign or exceeds the multiplicand or dividend size. The instruction is suppressed; therefore, the condition code and data in storage and registers remain unchanged.

Data: A sign or digit code of an operand in ADD DECIMAL, SUBTRACT DECIMAL, ZERO AND ADD, COMPARE DECIMAL, MULTIPLY DECIMAL, DIVIDE DECIMAL, EDIT, OR EDIT AND MARK is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these operations overlap incorrectly. The operation is terminated. The result data and the condition code are unpredictable and should not be used for further computation.

Decimal Overflow: The result of ADD DECIMAL, SUBTRACT DECIMAL, or ZERO AND ADD overflows. The program interruption occurs only when the decimal-over-flow mask bit is one. The operation is completed by placing the truncated low-order result in the result field and setting the condition code to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.

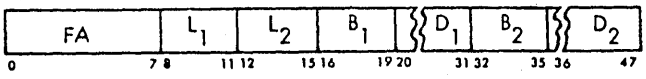
Decimal Divide: The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

DECIMAL FAMILY

ADD DECIMAL

The Add Decimal instruction provides a means by which two packed decimal operands can be algebraically added. The packed decimal data field specified by the second operand is added to the packed decimal data field designated by the first operand. The sum replaces the first operand.

AP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The packed decimal field specified by the second address field [D2(L2,B2)] is algebraically added to the packed decimal field designated by the first address field [D1(L1,B1)].
2. All digits and signs are checked for validity prior to the addition.
3. The sum replaces the first operand.
4. Decimal overflow may be caused by either a carry out of the high-order digit position or a first operand field shorter than the resultant sum.
5. PSW bit 37 will allow a decimal overflow to be masked.
6. First and second operand fields may overlap in any desired manner.

EXAMPLES

1. AP (Add Packed)
   Add the contents of a 5 byte field at storage location 2010 (FIELD1) to a 7 byte field at location 2131 (FIELD2). (GPR F = 00 00 20 00)

   SYMBOLIC AP FIELD2(7),FIELD1(5)

   MACHINE FA 64 F1 31 F0 10

		Before	After
Storage	2010	22 44 66 88	22 44 66 88
	2014	9C 12 34 56	9C 12 34 56
	2130	1D 00 55 44	1D 00 55 66
	2134	33 22 11 3C	77 89 00 2C

   Condition Code 2

2. AP (Add Packed)
   Add the contents of a 4 byte field at storage location 2200(FIELD3) to a 4 byte field at location 2204. (GPR F = 00 00 22 00)

   SYMBOLIC AP FIELD3+4(4),FIELD3(4)

   MACHINE FA 33 F0 04 F0 00

		Before	After
Storage	2200	76 54 32 1D	76 54 32 1D
	2204	12 34 56 7C	64 19 75 4D
Condition Code		1	

CONDITION CODE

0 Sum is zero
1 Sum is negative
2 Sum is positive
3 Overflow

PROGRAM INTERRUPTIONS

1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Data
5. Decimal overflow

DECIMAL FAMILY

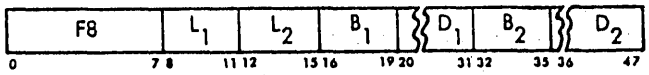
ZERO AND ADD

The Zero and Add instruction performs the equivalent of adding a packed decimal number to zero. This instruction is very flexible and can be used to accomplish any of the following:

1. Expand or reduce the field size of an operand.
2. Move a field from one location to another.
3. Assure that a field is zero prior to adding a value to it.

The packed decimal data field of the second operand is placed in the field of the first operand.

ZAP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The field specified by the first address field [D1(L1,B1)] is loaded with packed decimal data designated by the second address field [D2(L2,B2)].
2. This operation is the equivalent of adding the second operand to a zero first operand.
3. The second operand is checked for valid sign and digit codes.
4. Decimal overflow will occur if the contents of the second operand cannot be contained within the field of the first operand.
5. Decimal overflow may be masked by bit 37 of the PSW.
6. First and second operand fields may overlap in any desired manner.

EXAMPLE

1. ZAP (Zero and Add Packed)
   Zero and add to storage locations 2500 (FIELD6) through 2507, the contents of a two-byte field beginning at location 2508. (GPR F = 00 00 23 00)

   SYMBOLIC ZAP FIELD6(8),FIELD6+8(2)

   MACHINE F8 71 F2 00 F2 08

		Before	After
Storage	2500	16 23 48 97	00 00 00 00
	2504	26 11 55 3D	00 00 86 4C
	2508	86 4C 27 1C	86 4C 27 1C

   Condition Code 2

CONDITION CODE

0 Result is zero
1 Result is negative
2 Result is positive
3 Overflow

PROGRAM INTERRUPTIONS

1. Operation (decimal feature not installed)
2. Protection

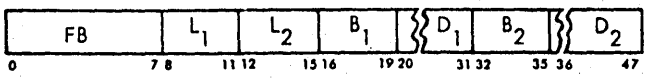
3. Addressing
4. Data
5. Decimal overflow

DECIMAL FAMILY

SUBTRACT DECIMAL

The Subtract Decimal instruction will find the algebraic difference between two packed decimal data fields. The first operand is the minuend and the subtrahend is the second operand. The difference will replace the first operand. This instruction can also be used to zero a field by specifying identical first and second operand starting addresses and lengths.

SP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The packed decimal data field specified by the second address field [D2(L2,B2)] is subtracted from the packed decimal data field designated by the first address field [D1(L1,B1)].
2. All digits and signs are checked for validity prior to the algebraic subtraction.
3. The difference replaces the first operand's field.
4. Decimal overflow will occur if the difference cannot be contained in the first operand's field.
5. PSW bit 37 allows the masking of decimal overflow.
6. First and second operand fields may overlap in any desired manner.

EXAMPLES

1. SP (Subtract Packed)
Zero storage locations 2500 (FIELD6) through 250B. (GPR F = 00 00 23 00)

SYMBOLIC SP FIELD6(12),FIELD6(12)

MACHINE FB BB F2 00 F2 00

		Before	After
Storage	2500	00 00 00 00	00 00 00 00
	2504	00 00 86 4C	00 00 00 00
	2508	86 4C 27 1C	00 00 00 0C
Condition Code		0	

2. SP (Subtract Packed)
Subtract storage locations 3114(SUB) through 311A from locations 3100(MIN) through 3107. (GPR F = 00 00 31 00)

SYMBOLIC SP MIN(8),SUB(7)

MACHINE FB 76 F0 00 F0 14

		Before	After
Storage	3100	99 99 99 99	99 22 33 44
	3104	99 99 99 9D	55 66 77 8D
	3114	77 66 55 44	77 66 55 44
	3118	33 22 1D 0C	33 22 1D 0C
Condition Code		1	

CONDITION CODE

0 Difference is zero
1 Difference is negative
2 Difference is positive
3 Overflow

PROGRAM INTERRUPTIONS

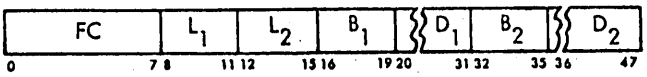
1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Data
5. Decimal Overflow

DECIMAL FAMILY

MULTIPLY DECIMAL

To find the product of two packed decimal data fields the programmer would issue a Multiply Decimal instruction. The first operand is the multiplicand and the multiplier is the second operand. The product will replace the multiplicand upon completion of the instruction.

MP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The multiplicand [D1(L1,B1)] is algebraically multiplied by the multiplier [D2(L2,B2)] and the product replaces the multiplicand.
2. The length of the multiplier must be less than the length of the multiplicand and cannot exceed 15 digits plus sign or a specification exception will occur.
3. To prevent product overflow a data exception will occur if the multiplicand field does not contain high-order zeros equal to or greater than the length of the multiplier.
4. The maximum product size is 31 digits and at least one high-order digit of the product is zero.
5. The multiplier and product fields may overlap when their low-order bytes coincide.

EXAMPLE

1. MP (Multiply Packed)
   Multiply the 6-byte field at storage location 5010(CAND) by the 3-byte field at location 5018(PLIER). (GPR F = 00 00 50 00)

   SYMBOLIC MP CAND(6),PLIER(3)

   MACHINE FC 52 F0 10 F0 18

		Before	After
Storage	5010	00 00 00 27	00 00 69 20
	5014	35 4D 12 8D	56 2C 12 8D
	5018	00 25 3D 2C	00 25 3D 2C

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

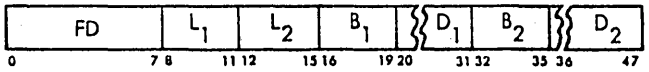
1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Specification
5. Data

DECIMAL FAMILY

DIVIDE DECIMAL

The Divide Decimal instruction will find the quotient of two packed decimal data fields. The dividend field is the first operand and the divisor is the second operand. The quotient and remainder replaces the dividend upon execution of the instruction.

DP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The packed decimal data field (dividend) specified by the first address field [D1(L1,B1)] is divided by the packed decimal data field (divisor) designated by the second address field [D2(L2,B2)].
2. The remainder replaces the rightmost portion of the dividend and occuppies the same number of digits as the divisor.
3. The quotient replaces the leftmost remaining positions of the dividend.
4. A divisor which exceeds 15 digits and sign or is greater than or equal to the length of the dividend will cause a specification exception.
5. The dividend, divisor, quotient, and remainder are all signed numbers, right aligned in their assigned field.
6. Division and signs are controlled by the rules of algebra.
7. Overflow cannot occur, but a quotient that cannot be contained in its assigned field will cause a decimal divide exception.

EXAMPLE

1. DP (Divide Packed)
   Divide the twelve-byte field at storage location 4130(DEND) by the one-byte field at location 4A2C(ISOR). (GPR F = 00 00 41 00)

   SYMBOLIC DP DEND (12),ISOR(1)

   MACHINE FD B0 F0 30 F9 2C

		Before	After
Storage	4130	00 88 44 66	44 22 33 11
	4134	22 44 66 88	22 33 44 00
	4138	00 88 44 0C	44 22 0D 0D
	4A2C	2D 13 26 79	2D 13 26 79

CONDITION CODE

1. Remains unchanged

PROGRAM INTERRUPTIONS

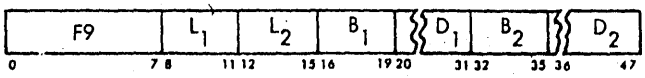
1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Specification
5. Data
6. Decimal divide

DECIMAL FAMILY

COMPARE DECIMAL

This instruction performs a numeric comparison between two packed decimal data fields. The first operand is compared to the second operand and the result of the comparison determines the setting of the Condition Code.

CP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



1. The packed decimal field designated by the first address field [D1(L1,B1)] is compared to the field specified by the second address field [D2(L2, B2)].
2. The result of the Comparison is indicated by the setting of the Condition Code.
3. Comparison is right to left and all signs and digits are checked for validity.
4. If the fields compared are unequal in length, the shorter field is expanded with high-order zeros prior to the comparisons.
5. Operands may overlap in any desired manner.

EXAMPLE

1. CP (Compare Packed)
   Compare the one-byte field at location 2508 (FIRST) to the two-byte field at location 250B(SND). (GPR F = 00 00 25 00)

   SYMBOLIC CP FIRST(1),SND(2)

   MACHINE F9 01 F0 08 F0 0B

		Before	After
Storage	2508	7C 4C 2D 00	7C 4C 2D 00
	250C	7C 21 6D 14	7C 21 6D 14
Condition Code		0	

CONDITION CODE

0 Operands are equal
1 First operand is lower
2 First operand is higher
3 --

PROGRAM INTERRUPTIONS

1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Data

EDIT

EDIT

The Edit instruction is used in the preparation of printed reports to give them a high degree of legibility and therefore greater usefulness. With proper planning, it is possible to suppress nonsignificant zeros, insert commas and decimal points, insert minus signs or credit symbols, and specify where suppression of leading zeros should stop for small numbers. All of these actions are done by the machine in one left-to-right pass.

We begin with a simple requirement to suppress leading zeros; no punctuation is to be inserted. We have a field to be edited, called DATA. It is four bytes long, and the decimal data is in packed format; the packed format for data to be edited is a requirement of the EDIT instruction.

The data to be edited is designated by the second operand and the first operand must specify a field containing a "pattern" of characters that controls the editing. After execution of the instruction, the location specified by the first operand contains the edited result. (The original pattern is destroyed by the editing process.) The pattern is in zoned format, as is the result; the Edit instruction causes the conversion of the data to be edited from packed to zoned format.

We said that in our example the data field to be edited was four bytes long, that is, seven decimal digits, and sign, which we shall assume to be plus. The pattern must accordingly be at least eight bytes long: seven for the digits and one at the left to designate the "fill character." The fill character may be any character, but is usually a blank. This is the character that is substituted for nonsignificant zeros.

The leftmost character of the pattern in our case will be the character blank (hexadecimal 40). The other seven characters will contain a special coding, hexadecimal 20, called a digit selector, which is used to indicate to the Edit instruction that a digit from the source data may go into the corresponding position.

Let us see how all this works out in our example. Suppose we set up an eight-byte working storage field named WORK into which we move the pattern (located in an area called PATTRN). Then we will perform our edit using WORK and DATA as the two operands. The two instructions necessary to do the job are:

```
MVC  WORK,PATTRN
ED   WORK,DATA
```

After execution of the two instructions, WORK contains our edited result. PATTRN still contains the original pattern and can transmit that original pattern to WORK for the editing of any new value in DATA. At PATTRN there should be the

following characters, written here in hexadecimal:

40 20 20 20 20 20 20 20

The 40 is the hexadecimal code for a blank. The 20 is the hexadecimal code for the digit selector. Suppose now that at DATA there is

00 01 00 0+

The edited result would be

b b b 1 0 0 0

where the b's stand for blanks. All zeros to the left of the first nonzero digit have been replaced by blanks; but zeros to the right of the first nonzero digit have been moved to WORK without change. This is the desired action. Figure 1 shows a series of values for DATA and the resultant edited results in WORK, using the pattern stated. Note that the high-order position of WORK contains the fill character, a blank. The values of DATA are packed decimal; the edited results are changed during execution of the Edit instruction to zoned decimal format.

```
 BDDDDDDD
 40 20 20 20 20 20 20 20

1234567   1234567
0120406    120406
0012345     12345
0001000      1000
0000123       123
0000012        12
0000001         1
0000000
```

Figure 1

Examples of the application of the Edit instruction. The first line gives the editing pattern used, first in a symbolic form and then in hexadecimal coding. In the symbolic form, B stands for blank and D for digit selector.

The fill character that we supply as the leftmost character of the pattern may be any character that we wish. It is fairly common practice to print dollar amounts of asterisks to the left of the first significant digit in order to protect against fraudulent alteration. This is usally called asterisk protection.

To do this, we need only change the leftmost character of the pattern of the previous example. The hexadecimal code for an asterisk is 5C; hence the new pattern is

```
 5C 20 20 20 20 20 20 20
```

Figure 2 shows the edited results for the same DATA values.

```
 *DDDDDDD
 5C 20 20 20 20 20 20 20

1234567  *1234567
0120406  **120406
0012345  ***12345
0001000  ****1000
0000123  *****123
0000012  ******12
0000001  *******1
0000000  ********
```

Figure 2

Examples of the application of the Edit instruction with an asterisk as the fill character.

Any characters in the pattern other than the digit selector and two other control characters that we shall study later are called message characters. They are not replaced by digits from the data. Instead, they are either replaced by the fill character (if a significant digit has not been encountered yet), or left as they are (if a significant digit has been found). Suppose, for instance, that we set up a PATTRN as follows:

```
  40 20 6B 20 20 20 6B 20 20 20
```

The 6B is hexadecimal coding for a comma, and it is a message character. The edited result will contain commas in the two positions shown, unless they are to the left of the first nonzero digit, in which case they are suppressed. Figure 3 shows the result of the same data values.

```
 BD,DDD,DDD
 40 20 6B 20 20 20 6B 20 20 20

1234567   1,234,567
0120406     120,406
0012345      12,345
0001000       1,000
0000123         123
0000012          12
0000001           1
0000000
```

Figure 3

Examples of the application of the Edit instruction with blank fill and the insertion of commas.

The message characters inserted are, naturally, not limited to commas. A frequent application is to insert a decimal point as well as commas. Let us assume that the data values we have been using are to be interpreted as dollars-and-cents amounts. We need to arrange for a comma to set off the thousands of dollars, and a decimal point to designate cents. The characters in PATTRN, where 6B is a comma and 4B is a decimal point, should be as follows:

```
  40 20 20 6B 20 20 20 4B 20 20
```

The edited results this time are in Figure 4.

```
BDD,DDD.DD
40 20 20 6B 20 20 20 4B 20 20

1234567   12,345.67
0120406    1,204.06
0012345      123.45
0001000       10.00
0000123        1.23
0000012          12
0000001           1
0000000
```

Figure 4

Examples of the application of the Edit instruction with blank fill and the insertion of comma and decimal point.

We see here something that would normally not be desired: amounts under one dollar have been edited with the decimal point suppressed. We would oridinarily prefer to have the decimal point. This can be done by placing a significance starter in the pattern.

The control character, which has the hexadecimal code 21, is either replaced by a digit from the data or replaced by the fill character, just as a digit selector is. The difference is that the operation proceeds as though a significant digit had been found in the position occupied by the significance starter. In other words, succeeding characters to the right will not be suppressed. (An exception to this generalization may occur when we want to print sign indicators, a subject that will be explored later.)

The pattern for this action, assuming we still want the comma and decimal point as before, should be

```
40 20 20 6B 20 20 21 4B 20 20
```

The effect is this: if nothing but zeros has been found by the time we reach the significance starter (code 21) in a left-to-right scan, the signficance starter will turn on the significance indicator. This indicator will cause succeeding characters to be treated as though a nonzero digit had been found. The result is that the decimal point will always be left in the result, as will zeros to the right of the decimal point. The edited results this time are shown in Figure 5.

One useful point to remember is that the total number of digit selectors plus significance starters in the pattern must equal the number of digits in the field to be edited. Note that this is the case in all our examples.

```
BDD,DDS.DD
40 20 20 6B 20 20 21 4B 20 20

1234567   12,345.67
0120406    1,204.06
0012345      123.45
0001000       10.00
0000123        1.23
0000012         .12
0000001         .01
0000000         .00
```

Figure 5

Examples of the application of the Edit instruction with blank fill, comma and decimal point insertion, and significance starter. In the symbolic pattern, S stands for significance starter.

We can begin to get a little idea of how the machine does its work on this instruction by noting that the significance indicator is initially in the off state before the scan begins. Scanning proceeds source digit by source digit. The significance indicator stays off until a nonzero data digit is found, or until the significance starter is encountered; either even causes the indicator to be turned on.

Source digits 1-9 always replace a digit selector or significance starter, but whether a zero source digit will do so depends upon the state of the significance indicator. If the significance digit was found at some previous character position, or a significance indicator is off, you know that no significant digit has been found so far during the scan; therefore, the fill character appears in the result, rather than a zero from the data.

It may be useful to refer to the Table, which includes a summary of how the state of the significance indicator affects the editing operation under all conditions of consequence that you may encounter. The table also shows how the significance indicator itself is affected.

In the table, the four columns at the left list all the significant combinations of the four conditions that can be encountered in the execution of the editing operation. The two columns at the right under Results show the action taken for each case - that is, the type of character placed in the result field and the new type of character placed in the result field and the new setting of the significance indicator. Use of the field separator will be discussed in a later paragraph.

CONDITIONS				RESULTS	
PATTERN CHARACTER	PREVIOUS STATE OF SIGNIFICANCE INDICATOR	SOURCE DIGIT	LOW-ORDER SOURCE DIGIT IS A PLUS SIGN	RESULT CHARACTER	STATE OF SIGNIFICANCE INDICATOR AT END OF DIGIT EXAMINATION
Digit selector	off	0	*	fill character	off
		1-9	no	source digit	on
		1-9	yes	source digit	off
	on	0-9	no	source digit	on
		0-9	yes	source digit	off
Significance starter	off	0	no	fill character	on
		0	yes	fill character	off
		1-9	no	source digit	on
		1-9	yes	source digit	off
	on	0-9	no	source digit	on
		0-9	yes	source digit	off
Field separator	*	**	**	fill character	off
Message character	off	**	**	fill character	off
	on	**	**	message character	on

*No effect on result character and new state of significance indicator.
**Not applicable because source digit not examined.

TABLE

We have so far ignored the sign portion of the source data, which (in the packed decimal format is required for the Edit instruction) is in the four low-order bits of the rightmost byte. These bits are examined each time the Edit instruction is executed. If the sign is plus, the significance indicator will then be turned off, as shown in the table; if the sign is minus, the significance indicator will be left on. The information will not appear in the result, however, if there are no further pattern characters to be scanned. As a matter of fact, if any of the source fields in the examples above had been negative, the results shown would have been exactly the same.

Suppose, however, that pattern characters remain after the sign position has been examined. The action of the significance indicator in controlling the instruction continues just as before, although the setting of the significance indicator was accomplished by a difference condition. There are, of course, no more digits to move. Hence we will not want to place digit selectors in the pattern in this position, but, rather, sign indicators, such as a minus sign or CR for credit. The action taken with the characters in the pattern is the same now as it was before: they remain unchanged if the significance indicator is on, but are replaced by the fill character if the significance indicator is off.

What we do, then, is to place the pattern the characters we want to print if the quantity is negative. If the data is indeed negative, our sign will be left, but if the data is positive, the sign will be replaced by the fill character.

Let us set up a suitable pattern for the example data. Let us print the letters CR for negative numbers, with one blank between the rightmost digit and the C. In hexadecimal, CR is C3 D9, so the pattern becomes:

```
40 20 20 6B 20 20 21 4B 20 20 40 C3 D9
```

Figure 6 shows the results for sample data values as before, together with two negative values.

```
BDD,DDS.DDBCR
40 20 20 6B 20 20 21 4B 20 20 40 C3 D9

1234567   12,345.67
0120406    1,204.06
0012345      123.45
0001000       10.00
0000123        1.23
0000012         .12
0000001         .01
0000000         .00
-0098765     987.65 CR
-0000000        .00 CR
```

Figure 6

Examples of the application of the Edit instruction with blank fill, comma and decimal point insertion, significance starter and CR symbol for negative numbers. In the symbolic pattern, C and R are themselves.

If we use an asterisk now as the fill character, positive quantities will have three asterisks following the cents, as shown in Figure 7. This may or may not be desired. There are other ways to handle the signs, as we shall see next.

We have seen above that an amount of zero prints in the general form .00 when a significance starter is used. It may in some cases be desirable to make such an amount print

as all blanks or all asterisks. This is very easily done by making use of the way the condition code is set by execution of the Edit instruction:

Code	Instruction
0	Result field is zero
1	Result field is less than zero
2	Result field is greater than zero

```
*DD,DDS.DDBCR
5C 20 20 6B 20 20 21 4B 20 20 40 C3 D9

1234567  *12,345.67***
0120406  **1,204.06***
0012345  ****123.45***
0001000  *****10.00***
0000123  ******1.23***
0000012  *******.12***
0000001  *******.01***
0000000  *******.00***
-0098765 ****987.65 CR
-0000000 *******.00 CR
```

Figure 7

Examples of the application of the Edit instruction using asterisk fill.

This means that after completion of the Edit we can make a simple Branch on Condition test of the condition code and move blanks or asterisks to the result field if it is zero. The movement is particularly simple because the fill character is still there in the field and an overlapped Move Characters instruction can be used as follows:

```
        BC  6,SKIP
        MVC WORK+1(12),WORK
SKIP
```

The explicit length of 12 is based on the most recent pattern, which has a total of 13 characters. The MVC, written, picks

up the leftmost character and moves it to the leftmost-plus-one position. It then picks up the leftmost-plus-one character and moves it to the leftmost-plus-two position, etc., in effect propagating the leftmost character through the field. This is precisely what we want if the fill character is the one to be substituted. (If some other character is desired, a suitable Move Characters instruction can, of course, be written.)

Figure 8 shows our familiar data values with zero fields blanked, and Figure 9 shows them with zero fields filled with asterisks. Only the fill character differs in the two programs that would produce the results shown in Figure 8 and 9; the Edit, the Branch on Condition, and the Move Characters are the same in both case.

The condition code can also be used to distinguish between positive and negative numbers when it is necessary to present the sign in some manner that is not possible by using the automatic features of the Edit. We might, for instance, wish to test the condition code and use the results of the test to place a plus sign or minus sign to the left of the edited result.

The Edit instruction can be used to edit several fields with one instruction. Doing so uses a final control character, the field separator (hexadecimal 22). This character is replaced in the pattern by the fill character, and causes the significance indicator to be set to the off state. The characters following, both in the pattern and in the source

data, are handled as described for a single field. In other words, it is possible to set up a pattern to edit a whole series of quantities, even an entire line, with one instruction. The packed source fields must, of course, be contiguous in storage, but this is often no inconvenience. One limitation is that the condition code, upon completion of such an instruction, gives information only about the last field encountered after a field separator.

```
 BDD,DDS.DDBCR
 40 20 20 6B 20 20 21 4B 20 20 40 C3 D9

1234567   12,345.67
0120406    1,204.06
0012345      123.45
0001000       10.00
0000123        1.23
0000012         .12
0000001         .01
0000000
-0098765      987.65  CR
-0000000
```

Figure 8

Examples of the application of the Edit instruction, showing the blanking of zero fields by the use of two additional instructions.

```
 *DD,DDS.DDBCR
 5C 20 20 6B 20 20 21 4B 20 20 40 C3 D9

1234567  *12,345.67***
0120406  **1,204.06***
0012345  ****123.45***
0001000  *****10.00***
0000123  ******1.23***
0000012  *******.12***
0000001  *******.01***
0000000  **********
-0098765 ****987.65 CR
-0000000 **********
```

Figure 9

Examples of the application of the Edit instruction with asterisk fill and zero filled with asterisks instead of being blanked.

Let us consider the example shown in Figure 10. Suppose that at DATA we have a sequence of three fields. The leftmost of the fields has four bytes, the next has three, and the rightmost has five bytes. The first is to be printed with commas separating groups of three digits. The values are always positive and, therefore, no sign control is desired. Zero values will be blank since we shall not use a significance starter.

1234567C12345C123456789C	1,234,567	12.345	1,234,567.89
0123456C01234C012345678C	123,456	1.234	123,456.78
0010009C00123C001000000C	10,009	0.123	10,000.00
0004502C98007D000001210C	4,502	98.007-	12.10
0000800C00012C000000006C	800	0.012	.06
0000001C00001D000000001C	1	0.001-	.01
0000000C00000C000000000C		0.000	

Figure 10

Examples of multiple edits. On each line the first field is a combination of three items; all three were edited with one Edit, giving the three results shown to the right. The editing pattern and additional instructions are shown in the text.

The second field is to be printed with three digits to the right of the decimal point, with a significance starter to force amounts less than 1 to be printed with a zero before the decimal point. Positive quantities are to be printed without a sign, and negative quantities are to be printed with a minus sign immediately to the right of the number.

The third number is a dollar amount that could be as great as $9,999,999.99. Commas and decimal point are needed just

as shown. Amounts less than $1 are to be printed with the decimal point as the leftmost character. Zero amounts are to be blanked. Signs are not to be printed.

There is to be at least one blank between the first and second edited result, and at least three between the second and third.

Let us write out the necessary pattern in shorthand form, with b standing for a blank, d for digit selector, f for field separator, s for significance starter, and other characters for themselves:

bd,ddd,dddfsd.ddd-fbbd,ddd,dds.dd

The required blank between the first and second edited result will be placed there by the replacement of the field separator with the fill character. The significance starter in the part of the pattern corresponding to the second field will give the required handling of quantities less than 1. The extra two blanks between the second and third results are provided by the blanks in the part of the pattern corresponding to the third data item. (These are not treated as new fill characters; only the leftmost character in the entire pattern is so regarded.) Notice that the total of digit selectors plus significance starters is equal to the number of digits in each field to be edited.

Instructions to do the required actions at Figure 10 are as follows:

```
        MVC    WORK,PATTRN
        ED     WORK,DATA
        BC     6,SKIP
        MVC    WORK+30(3),WORK+18
SKIP
```

The choice of addresses in the final MVC that blanks a zero field is somewhat arbitrary. We reason that if the entire field is zero, the first three positions of it are surely blank by now; hence a three-character MVC from there to the last three positions of the field will be correct.

Figure 10 shows initial source data values and edited results. The packed source fields must be adjacent as shown; we address the leftmost character.

EDIT AND MARK

The Edit and Mark instruction (EDMK) makes possible the insertion of floating currency symbols. By this we mean the placement in the edited result of a dollar sign (or pound sterling symbol) in the character position immediately to the left of the first significant digit. This serves as protection against alteration, since it leaves no blank spaces. It is a somewhat more attractive way to provide protection than the asterisk fill.

The operation of the instruction is precisely the same as the Edit instruction, with one additional action. The execution of the Edit and Mark places in register 1 the address of the first significant digit. The currency symbol is needed one position to the left of the first significant digit. Consequently, we subtract one from the contents of register 1 after the execution of the Edit and Mark and place a dollar sign in that position.

There is one complication; if significance is forced by a significance starter in the pattern, nothing is done with register 1. Before going into the Edit and Mark, therefore, we place in register 1 the address of the significance starter plus one. Then, if nothing happens to register 1, we still get plus one. Then, if nothing happens to register 1, we still get the dollar sign in the desired position by using the procedure described above.

Let us suppose that we are again working with a four-byte

source data field, which we are to edit with a comma, a decimal point, and CR for negative numbers. Accordingly, the pattern (in shorthand form) should be

```
bdd,dds.ddbCR
```

The significance starter here is six positions to the right of the leftmost character of the pattern. The complete program to give the required editing and the floating dollar sign is as follows:

```
        MVC     WORK,PATTRN
        LA      1,WORK+7
        EDMK    WORK,DATA
        BCTR    1,0
        MVC     0(1,0),DOLLAR
DOLLAR  DC      C'S'
```

The Load Address instruction as written, places in register 1 the address of the position one beyond the significance starter. If significance is forced, this address remains in register 1, but otherwise the address of the first significant digit is placed in register 1 as part of the execution of the Edit and Mark. The Branch on Count Register instruction with a second operand of zero reduces the 1st operand register contents by 1 and does not branch. There are, of course, other ways to subtract 1 from the contents of register 1, but this is the easiest and fastest. In the Move Characters instruction we write an explicit displacement of zero, an explicit length of 1, and an explicit base register number of 1. The net effect is to move a one-character field from DOLLAR to the address specifed by the base register 1. This is the desired action.

Figure 11 shows the effect on sample data values. Zero fields could be blanked by methods we have seen above.

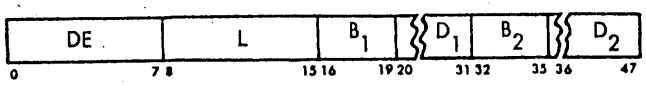
```
BBD,DDS.DDBCR
40 20 20 6B 20 20 21 4B 20 20 40 C3 D9

1234567   $12,345.67
0120406    $1,204.06
0012345      $123.45
0001000       $10.00
0000123        $1.23
0000012          $.12
0000001          $.12
0000000          $.01
-0098765     $987.65  CR
-0000000        $.00  CR
```

Figure 11

Examples of the application of the Edit and Mark instruction to get a floating currency symbol.

ED $D_1(L, B_1), D_2(B_2)$ [SS]



1. The packed decimal data beginning at the address specified by the second address field (D2,B2) is converted to zoned data and placed in storage beginning at the location designated by the first address field [D1(L1,B1)].
2. The placement of a decimal digit in the first operand's field is controlled by a pattern contained within that field.
3. The pattern is a combination of special characters which allows the suppression of non-significant zeros and the insertion of punctuation into resultant field.
4. This pattern must be established prior to issuing the EDIT instruction.
5. The pattern is destroyed when the source digits (packed decimal digits) are transferred to the first operand's field.
6. Operands are processed left to right a byte at a time.
7. The packed data is checked for valid sign and digit codes.
8. Overlapping fields will result in an unpredictable outcome.

EXAMPLE

1. ED (Edit)
   Edit the four-byte field located at storage address 1040(DEK) and place in the 10-byte pattern field located at 1000 (RESULT).
   (GPR F = 00 00 10 00)

   SYMBOLIC ED RESULT(10),DEK

   MACHINE DE 09 F0 00 F0 40

		Before	After
Storage	1000	40 20 20 6B	40 F2 F2 6B
	1004	20 20 21 4B	F4 F4 F6 4B
	1008	20 20 56 20	F6 F7 56 20
	1040	22 44 66 7C	22 44 66 7C

   Condition Code 2

CONDITION CODE

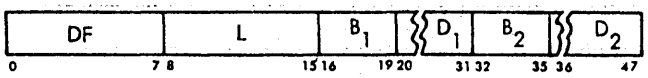
0 Source digits all zeros
1 Source digits are negative
2 Source digits are positive
3 --

PROGRAM INTERRUPTIONS

1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Data

Edit and Mark

EDMK $D_1(L, B_1), D_2(B_2)$ [SS]



1. The packed decimal data beginning at the address specified by the second address field (D2, B2) is converted to zoned data and placed in storage beginning at the location desginated by the first address field [D1 (L1,B1)].
2. The placement of a decimal digit in the first operand's field is controlled by a pattern contained within that field.
3. The pattern is a combination of special characters which allows the suppression of non-significant zeros and the insertion of punctuation into resultant fields.
4. The address of the first significant digit encountered is stored in bits 8-31 of register 1.
5. This pattern must be established prior to issuing the EDIT and MARK instruction.
6. The pattern is destroyed when the source digits (packed decimal digits) are transferred to the first operand's field.
7. Operands are processed left to right, a byte ar a time.
8. The packed data is checked for valid sign and digit codes.
9. Overlapping fields will result in a unpredictable outcome.

EXAMPLE

1. EDMK (Edit and Mark)
   Edit and Mark the five-byte field located at storage address 1020 (FIELD) and place in the thirteen-byte pattern field located at 1100 (OTPT). (GPR F = 00 00 10 00)

```
SYMBOLIC  EDMK   OTPT(13),FIELD
MACHINE   DF 0C F1 00 F0 20
```

		Before	After
Storage	1100	40 20 6B 20	40 40 40 40
	1104	20 20 6B 20	F7 F5 6B F3
	1108	20 21 4B 20	F6 F0 4B F2
	110C	20 40 21 20	F0 40 21 20
	1020	00 75 36 02	00 75 36 02
	1024	0C 22 42 16	0C 22 42 16
GPR	1	00 00 00 00	00 00 1I 04
Condition Code		2	

CONDITION CODE

0 Source digits all zeros
1 Source digits are negative
2 Source digits are positive
3 --

PROGRAM INTERRUPTIONS

1. Operation (decimal feature not installed)
2. Protection
3. Addressing
4. Data

INPUT/OUTPUT INSTRUCTIONS

The transferring of information between a system and its devices is accomplished through an Input/Output Operation.

An input/output I/O operation involves the use of an input/output device. Input/output devices perform I/O operations under control of control units, which are attached to the central processing unit (CPU) by means of channels.

Input/Output devices include such equipment as card read punches, magnetic tape units, direct-access-storage devices (disks and drums), typewriter-keyboard devices. printers, teleprocessing, devices, and process control equipment.

Input/output operations are initiated and controlled by information with three types of formats: instructions, commands, and orders. Instructions are decoded by the CPU and are part of the CPU program. Commands are decoded and executed by the channels and I/O devices. One or more commands arranged for sequential execution form a channel program.

Functions peculiar to a device, such as rewinding tape or positioning the access mechanism on a disk drive, are specified by orders. Orders are decoded and executed by I/O devices. The CPU controls I/O operations by means of four I/O instructions: START I/O, TEST I/O, HALT I/O, and TEST CHANNEL.

The instruction TEST CHANNEL addresses a channel; it does not address an I/O device. The other three I/O instructions address a channel and a device on that channel.

INPUT/OUTPUT DEVICE ADDRESSING

The first operand of and I/O instruction designates an effective storage address. The low-order 16 bits of this effective address become the I/O address.

An I/O device is designated by the 16-bit I/O address. The high-order 8 bits of this I/O address specify a channel to which the desired I/O device is attached. The low-order 8 bits of this I/O address specify the actual device.

The channel-address field provides for identifying up to 256 channels, out of which only channels 0-6 may be installed; channel-address 7 and up are considered invalid. Channel 0 is a multiplexor channel; channels numbered 1-6 may be either multiplexor or selector channels, as shown below. The number and type of channels available, as well as their address assignment, depend on the system model and the particular installation.

ADDRESS		
CHANNEL	DEVICE	ASSIGNMENT
0000 0000	XXXX XXXX	Devices on channel 0
0000 0001	XXXX XXXX	Devices on channel 1
0000 0010	XXXX XXXX	Devices on channel 2
0000 0011	XXXX XXXX	Devices on channel 3
0000 0100	XXXX XXXX	Devices on channel 4
0000 0101	XXXX XXXX	Devices on channel 5
0000 0110	XXXX XXXX	Devices on channel 6
0000 0111	XXXX XXXX	
TO		INVALID
1111 1111	XXXX XXXX	

The device address identifies the particular I/O device and control unit on the designated channel. The address identifies, for example, a particular magnetic tape drive, disk access mechanism, or transmission line. Any number in the range 0-255 can be used as a device address, providing facilities for addressing up to 256 devices per channel.

Devices that do not share a control unit with other devices may be assigned any device address in the range 0-255, provided the address is not recognized by any other control unit. Logically, such devices are not distinguishable from their control unit, and both are identified by the same address.

Devices sharing a control unit (i.e., magnetic tape drives or disk access mechanisms) are assigned addresses within sets of sequential numbers.

Except for the rules described, the assignment of channel and drive device addresses is arbitrary. The assignment is made at the time of installation, and the addresses normally remain fixed thereafter.

STATES OF THE INPUT/OUTPUT SYSTEM

The state of the I/O system identified by an I/O address depends on the collective state of the channel, subchannel, and I/O device. Each of these components of the I/O system can have up to four states, as far as the response to an I/O instruction is concerned. These states are listed in the

following table. The name of the state is followed by its abbreviation and a brief definition.

CHANNEL	ABBREV	DEFINITION
Available	A	None of the following states
Interruption Pending	I	Interruption immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational

SUBCHANNEL	ABBREV	DEFINITION
Available	A	None of the following states
Interruption Pending	I	Information for CSW available in subchannel
Working	W	Subchannel executing an operation
Not operational	N	Subchannel not operational

I/O DEVICE	ABBREV	DEFINITION
Available	A	None of the following states
Interruption Pending	I	Interruption condition pending in device
Working	W	Device executing an operation
Not operational	N	Device not operational

A channel, subchannel, or I/O device that is available, that contains a pending interruption condition, or that is working, is said to be OPERATIONAL. The states of containing an interruption condition, working, or being not operational are collectively referred to as NOT AVAILABLE.

The device referred to in the preceding table includes both the device proper and its control unit. For some types of devices, such as magnetic tape units, the working and the interruption-pending states can be caused by activity in the addressed device or control unit. A shared control unit imposes its state on all devices attached to the control unit. The states of the devices are not related to those of the channel and subchannel.

When the response to an I/O instruction is determined on the basis of the states of the channel and subchannel, the DEVICES are not interrogated. Thus, ten composite states are identified as conditions for the execution of the I/O instruction. Each composite state is identified in the following discussion by three alphabetic characters; the first character position identifies the state of the channel, the second identifies the state of the subchannel, and the third refers to the state of the device. Each character position can contain A, I, W, or N, denoting the state of the component. The symbol X in place of a letter indicates that the state of the corresponding component is not significant for the execution of the instruction.

Available (AAA): The addressed channel, subchannel, control unit, and I/O device are operational, are not engaged in the execution of any previously initiated operations, and do not contain any pending interruption conditions.

Interruption Pending in Device (AAI) or Device Working (AAW): The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or contains a pending interruption condition. These situations are possible:

1. The device is executing an operation after signaling the channel-end condition, such as rewinding tape or seeking on a disk file.

2. The control unit associated with the device is executing an operation after signaling the channel-end condition, such as backspacing file on a magnetic tape unit.

3. The device or control unit is executing an operation on another subchannel or channel.

4. The device or control unit contains the device-end, control-unit-end, or attention condition or, on the selector channel, the channel-end condition associated with an operation terminated by HALT I/O.

Device Not Operational (AAN): The addressed channel and subchannel are available. The addressed I/O device is not operational. A device appears not operational when no control unit recognizes the address. This occurs when the control unit is not provided in the system, when power is off in the unit, or when the control unit has been logically switched off the I/O interface. The not-operational state is indicated also when the control unit is provided and is designed to attach the device, but the device has not been installed and the address has not been assigned to the control unit.

If the addressed device is not installed or has been logically removed from the control unit, but the associated control unit is operational and the address has been assigned to the control unit, the device is said to be not-ready. When an instruction is addressed to a device in the not-ready state, the control unit responds to the selection and indicates unit check whenever the not-ready state precludes a successful execution of the operation.

Interruption Pending In Subchannel (AIX): The addressed channel is available. An interruption condition is pending

in the addressed subchannel because of the termination of the portion of the operation involving the use of channel facilities. The subchannel is in a position to provide information for a complete CSW. The interruption condition can indicate termination of an operation at the addressed I/O device or at another device on the subchannel. The state of the addressed device is not significant, except when TEST I/O is addressed to the device associated with the terminated operation, in which case the CSW contains status information provided by the device.

The state AIX does not occur on the selector channel. On the selector channel, the existence of an interruption condition in the subchannel immediately causes the channel to assign to this condition the highest priority for I/O interruptions and, hence, leads to the state IIX.

Subchannel Working (AWX): The addressed channel is available. The addressed subchannel is executing a previously initiated operation or chain of operations in the multiplex mode and has not yet reached the channel end for the last operation. The state of the addressed device is not significant, except when HALT I/O is issued, in which the case the CSW contains status provided by the device.

The subchannel-working state does not occur on the selector channel since all operations on the selector channel are executed in the burst mode and cause the channel to be in the working state(WWX).

Subchannel Not Operational (ANX): The addressed channel is available. The addressed subchannel on the multiplexor channel is not operational. A subchannel is not operational when it is not provided in the system. This state cannot occur on the selector channel.

Interruption Pending in Channel (IXX): The addressed channel is not working and has established which device will cause the next I/O interruption from this channel. The state where the channel contains a pending interruption TEST CHANNEL. This instruction does not cause the subchannel and I/O device to be interrogated. The other I/O instructions consider the channel available when it contains a pending interruption condition. When the channel assigns priority for interruption among devices, the interruption condition is preserved in the I/O device or subchannel.

Channel Working (WXX): The addressed channel is operating in the burst mode. In the case of the multiplexor channel, a burst of bytes is currently being handled. In the case of the selector channel, an operation or a chain of operations is currently being executed, and the channel end for the last operation has not yet been reached. The states of the addressed device and, in the case of the multiplexor channel, of the subchannel are not significant.

Channel Not Operational (NXX): The addressed channel is not operational, or the channel address in the instruction is invalid. A channel is not operational when it is not provided in

the system, when power is off in the channel, or when it has been switched to the test mode. The states of the addressed I/O device and subchannel are not significant.

CONDITION CODE

The results of certain tests by the channel and device, and the original state of the addressed part of the I/O system are used during the execution of an I/O instruction to set one of four condition codes in bit positions 34 and 35 of the PSW. The condition code is set at the time the execution of the instruction is completed, that is, the time the CPU is released to proceed with the next instruction. The condition code indicates whether or not the channel has performed the function specified by the instruction and, if not, the reason for the rejection. Immediately following branch-on-condition operations can use the code for decision-making.

The following table lists the conditions and the corresponding Condition Codes for each instruction. The digits in the table represent the numeric value of the code. The instruction start I/O can set code 0 or 1 for the AAA state, depending on the type of operation that is initiated.

CONDITIONS				CONDITION CODE FOR START TEST I/O	CONDITION CODE FOR HALT TEST I/O	CONDITION CODE FOR HALT TEST CHAN	
Available	A	A	A	0,1*	0	1*	0
Interruption pend. in device	A	A	I	1*	1*	1*	0
Device working	A	A	W	1*	1*	1*	0
Device not operational	A	A	N	3	3	3	0
Interruption pend. in subchannel	A	I	X	†			
For the addressed device				2	1*	0	0
For another device				2	2	0	0
Subchannel working	A	W	X	†2	2	1*	0
Subchannel not operational	A	N	X	†3	3	3	0
Interruption pend. in channel	I	X	X	† see note below			1
Channel working	W	X	X	†2	2	2	2
Channel not operational	N	X	X	†3	3	3	3
Error							
Channel equipment error				1*	1*	1*	—
Channel programming error				1*	—	—	—
Device error				1*	1*	—	—

*The CSW or its status portion is stored at location 64 during execution of the instruction.

†The symbol X stands for A, I, W, and N, and indicates that the state of the corresponding component is not significant. As an example, AIX denotes the states AIA, AII, AIW, and AIN, while IXX represents a total of 16 states, some of which do not occur.

—The condition cannot be identified during execution of the instruction.

NOTE: For the purpose of executing START I/O, TEST I/O, and HALT I/O, a channel containing a pending interruption condition appears the same as an available channel, and the condition-code setting depends upon the states of the subchannel and device. The condition codes for the IXX states are the same as for the AXX states, where the X's represent the states of the subchannel and the device. As an example, the condition-code for the IAA state is the same as for the AAA state, and the condition code for the IAW state is the same as for the AAW state.

The AVAILABLE condition is indicated only when no errors are detected during the execution of the I/O instruction. When a programming error occurs in the information placed in the CAW or CCW and the addressed channel or subchannel is working, either Condition Code 1 or 2 may be set, depending upon the model. Similarly, either code 1 or 3 may be set when a programming error occurs and a part of the addressed I/O system is not operational.

When a subchannel on the multiplexor channel contains a pending interruption condition (state AIX), the I/O device associated with the terminated operation normally is in the interruption-pending state. When the channel detects during execution of TEST I/O that the device is not operational, condition code 3 is set. Similarly, Condition Code 3 is set when HALT I/O is addressed to a subchannel in the working state and operating in the multiplex mode (state AWX), but the device turns out to be not operational. The not-operational state in both situations can be caused by operator intervention or by machine malfunctioning.

The error conditions listed in the preceding table include all equipment or programming errors detected by the channel or the I/O device during execution of the I/O instruction. Except for channel equipment errors, in which case, depending on the model, machine check may be indicated and no CSW may be stored, the status portion of the CSW identifies the error. Three types of errors can occur:

Channel Equipment Error: The channel can detect the following equipment errors during execution of START I/O, TEST I/O, and HALT I/O:

1. The device address that the channel received on the interface during initial selection either has a parity error or is not the same as the one the channel sent out. Some device other than the one addressed may be malfunctioning.

2. The unit-status byte that the channel received on the interface during initial selection has a parity error.

3. A signal from the I/O device occurred during initial selection at an invalid time or had invalid duration.

4. The channel detected an error in its control equipment.

The channel may perform the malfunction-reset function, depending on the type of error and the model. If a CSW is stored, channel control check or interface control check is indicated, depending on the type of error.

Channel Programming Error: The channel can detect the following programming errors during execution of START I/O.

1. Invalid CCW address in CAW
2. Invalid CCW address specification in CAW
3. Invalid storage protection key in CAW
4. Invalid CAW format
5. Location of first CCW protected for fetching
6. First CCW specifies transfer in channel
7. Invalid command code in first CCW
8. Initial data address exceeds addressing capacity of Model (see "Definition of Storage Area")
9. Invalid count in first CCW
10. Invalid format of first CCW

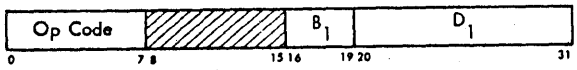
The CSW indicates program check, except for condition 5, in which case protection check is indicated.

Device Error: Programming or equipment errors detected by the device during the execution of START I/O are indicated by unit check or unit exception in the CSW.

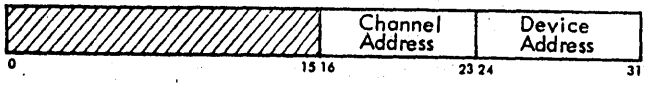
The conditions responsible for unit check and unit exception for each type of I/O device are detailed in the SRL publication for the device.

INSTRUCTION FORMAT

All I/O instructions use the following SI format:



Bit positions 8-15 of the instruction are ignored. The content of the B1 field designates a register. The sum obtained by the addition of the content of register B1 and content of the D1 field identifies the channel and the I/O device. The sum has the format:



Bit positions 0-7 are not part of the address. Bit positions 8-15, which constitute the high-order portion of the address, are ignored. Bit positions 16-23 of the sum contain the channel address, while bit positions 24-31 identify the device on the channel and, additionally in the case of the multiplexor channel, the subchannel.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for

the IBM System/360 assembly language are shown with each instruction. In the case of START I/O, for example, SIO is the mnemonic and D1(B1) the operand designation.

INSTRUCTIONS

The mnemonics, format, and operation codes of the I/O instructions follow. The table also indicates that all I/O instructions cause program interruption when they are encountered in the problem state, and that all I/O instructions set the condition code.

NAME	MNEMONIC	TYPE	EXCEPTION	CODE
Start I/O	SIO	SI,C	M	9C
Test I/O	TIO	SI,C	M	9D
Halt I/O	HIO	SI,C	M	9E
Test Channel	TCH	SI,C	M	9F

NOTES
C Condition code is set
M Privileged-operation exception

INPUT/OUTPUT INSTRUCTIONEXCEPTION HANDLING

Before the channel is signaled to execute an I/O instruction, the instruction is tested for validity by the CPU. Exceptional conditions detected at this time cause a program interruption. When the interruption occurs, the current PSW is stored as the program old PSW and is replaced by the program new PSW. The interruption code in the old PSW identifies the cause of the interruption.

The following exception may cause a program interruption:

Privileged Operation: An I/O instruction is encountered when the CPU is in the problem state. The instruction is suppressed

before the channel has been signaled to execute it. The CSW, the condition code in the PSW, and the state of the addressed subchannel and I/O device are not affected by the attempt to execute an I/O instruction while in the problem state.

INPUT/OUTPUT FAMILY

START I/O

The Start I/O instruction provides a means by which an I/O operation may be initiated. A write, read, readbackword, control or sense operation will begin at the device specified by the low-order 16 bits of the effective address.

The initiation of a Start I/O to an available device results in the Channel Address Work (CAW) being sent to the channel that was addressed. The channel utilizes the protection key of the CAW and fetches the Channel Command Word (CCW) from the storage location specified by the CAW.

This CCW specifies the operation to be performed, the main-storage area to be used, and the action to be taken when the operation is completed. The contents of the CCW and CAW must be established by the programmer prior to issuing a Start I/O.

If the addressed device was not available when interrogated by the Start I/O instruction, the entire operation is aborted and the Condition Code is set to reflect the cause of the abortion.

SIO $D_1(B_1)$ [SI]

1. The low-order 16 bits of the effective address (D1,B1) form the I/O address.
2. The CAW is sent to the channel indicated by the I/O address.
3. The protection key contained in the CAW is assumed by the channel and the effective address is used to fetch the CCW.
4. The CCW is decoded to provide the types of operation, the next sequential action to be taken, and if necessary, the number of bytes, and a storage.
5. The contents of the CAW and CCW must be formulated by the programmer prior to issuing the Start I/O instruction.
6. This instruction is a privileged operation and may only be issued by the control program.
7. A Condition Code other than 0 results in the instruction be aborted.
8. The CSW is stored at the end of an operation for a Condition Code of 0 or 1.

EXAMPLES

1. SIO (Start Input/Output)
The following example consists of three instructions. The Load Address and Store instruction will be used to establish the address of the CCW at location 5400(CCW1) in the CAW. The protection key will be zero and this example will perform a write operation of 10 bytes from location 5000 (STOR). The device is a tape drive on channel 2 with a device address of 80. (GPR F = 00 00 50 00)

```
   SYMBOLIC                          MACHINE

LA  1,CCW1                          41 10 F4 00
ST  1,72(0)                      50 10 00 48
SIO X'280'(0)                    9C 00 02 80

                          Before        After

 STOR     5000    FF FF FF FF    FF FF FF FF
          5004    FF FF FF FF    FF FF FF FF
          5008    FF FF FF FF    FF FF FF FF
```

```
                              Before          After

    CCW1        5400      01 00 50 00     01 00 50 00
                5404      00 00 00 0A     00 00 00 0A

    CSW         00 00 54 08 0C 00 00 00

    Condition Code  0
```

2. SIO (Start Input/Output)
   This example will perform a read operation of 80 bytes into storage location 5500(DATAIN). The input device is card reader located on the multiplexor channel at address 0C. The CCW is located at storage location 5408 (CCW2). (GPR F = 00 00 50 00)

```
  SYMBOLIC                          MACHINE
LA  1,CCW2                      41 10 F4 08
ST  1,72(0)                     50 10 00 48
SIO 12(0)                       9C 00 00 0C

                              Before          After

    CCW2        5408      02 00 55 00     02 00 55 00
                540C      00 00 00 50     00 00 00 50

    DATAIN      5500      00 00 00 00     F1 F2 F3 F6
                5546      00 00 00 00     C6 F3 F5 F7

    CSW         00 00 54 10 0C 00 00 00

    Condition Code  0
```

CONDITION CODE

0	Channel executing operation
1	CSW has been stored
2	Channel or subchannel busy
3	Channel or device not operational

PROGRAM INTERRUPTIONS

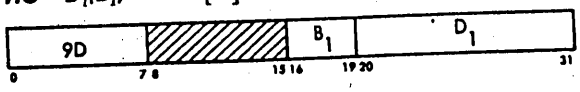
1. Privileged operation

INPUT/OUTPUT FAMILY

TEST I/O

The channel and device specified by the low-order 16 bits of the effective address are interrogated. The result of this interrogation is reflected in the setting of the Condition Code. If this setting is 1, more detailed information concerning the status of this channel and device can be found in the CSW. This instruction also permits a program to clear an interrupt at a selective I/O device.

TIO $D_1(B_1)$ [SI]



1. The low-order 16 bits of the effective address (D1,B1) form the I/O address.
2. The exact status of the addressed channel and device is reflected by the Condition Code.
3. A Condition Code of 1 indicates that further status information has been placed in the CSW.
4. Issuing this instruction will clear pending interrupts on the majority of I/O devices.
5. This instruction is a privileged operation and may only be performed by the control program.

EXAMPLE

1. TIO (Test Input/Output)
   Test the device at address 82 on channel 1 to determine its status. This device being tested will show a busy condition. (GPR C = 00 00 01 82)

   SYMBOLIC TIO 0(12) MACHINE 9D 00 C0 00

   Condition Code 1

   CSW 00 00 00 00 10 00 00 00

CONDITION CODE

0 Available
1 CSW has been stored
2 Channel or subchannel busy
3 Channel or device not operational

PROGRAM INTERRUPTIONS

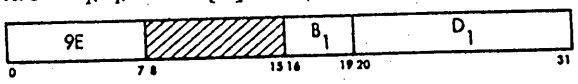
1. Privileged operation

INPUT/OUTPUT FAMILY

HALT I/O

The instruction HALT I/O provides the program with a means of terminating an I/O operation before all data specified by the operation has been transferred or before the operation at the device had reached its normal ending point. This would permit a program to immediately free a selector channel for an operation of higher priority. The device to be halted is specified by the low-order 16 bits of the effective address.

HIO $D_1(B_1)$ [SI]



1. The low-order 16 bits of the effective address (D1,B1) form the I/O address.
2. The device specified by the I/O address is selected and its operation terminated.
3. This instruction has no effect on devices that are not in the working state or are executing an operation of a fixed duration, such as rewinding a tape.
4. The status portion of the CSW is updated if a device terminated an operation, the control unit was busy and would not accept the HALT I/O, or an equipment malfunction occurred during execution of this instruction.
5. The termination of operation on the selector channel causes the channel and subchannel to be placed in the interrupt-pending state.
6. This operation is not terminated on the multiplexor channel until all outstanding requests for data have been serviced.
7. The instruction is a privileged operation and may only be issued by the control program.

EXAMPLE

1. HIO (Halt Input/Output)
   Halt the disk drive (address 31) on channel 2.

   SYMBOLIC HIO X'231'(0)

   MACHINE 9E 00 02 31

	Before	After
CSW	00 00 46 08 00 00 00 50	00 00 46 08 00 00 00 22
Condition Code	2	

CONDITION CODE

0 Interrupt pending in subchannel
1 CSW has been stored
2 Burst operation terminated
3 Not operational

PROGRAM INTERRUPTIONS

1. Privileged operation

INPUT/OUTPUT FAMILY

TEST CHANNEL

The channel specified by bits 16-23 of the effective address are tested and the result of the test is indicated in the Condition Code. This is very similar to the Test I/O but only involves the channel and it is not concerned with the subchannel or devices.

TCH $D_1(B_1)$ [SI]

1. Bits 16-23 of the effective address specify the channel to be tested.
2. The result of this test is used to set the Condition Code.
3. No device is selected or subchannel interrogated.
4. This is a privileged instruction and can only be issued by the Control Program.

EXAMPLE

1. TCH (Test Channel)
   Determine the status of channel 4.

   SYMBOLIC TCH X'400'(0)

   MACHINE 9F 00 04 00

   Condition Code 2

CONDITION CODE

0 Channel available
1 Interrupt pending in channel
2 Channel operating in burst mode
3 Channel not operational

PROGRAM INTERRUPTIONS

1. Privileged operation

SECTION 12

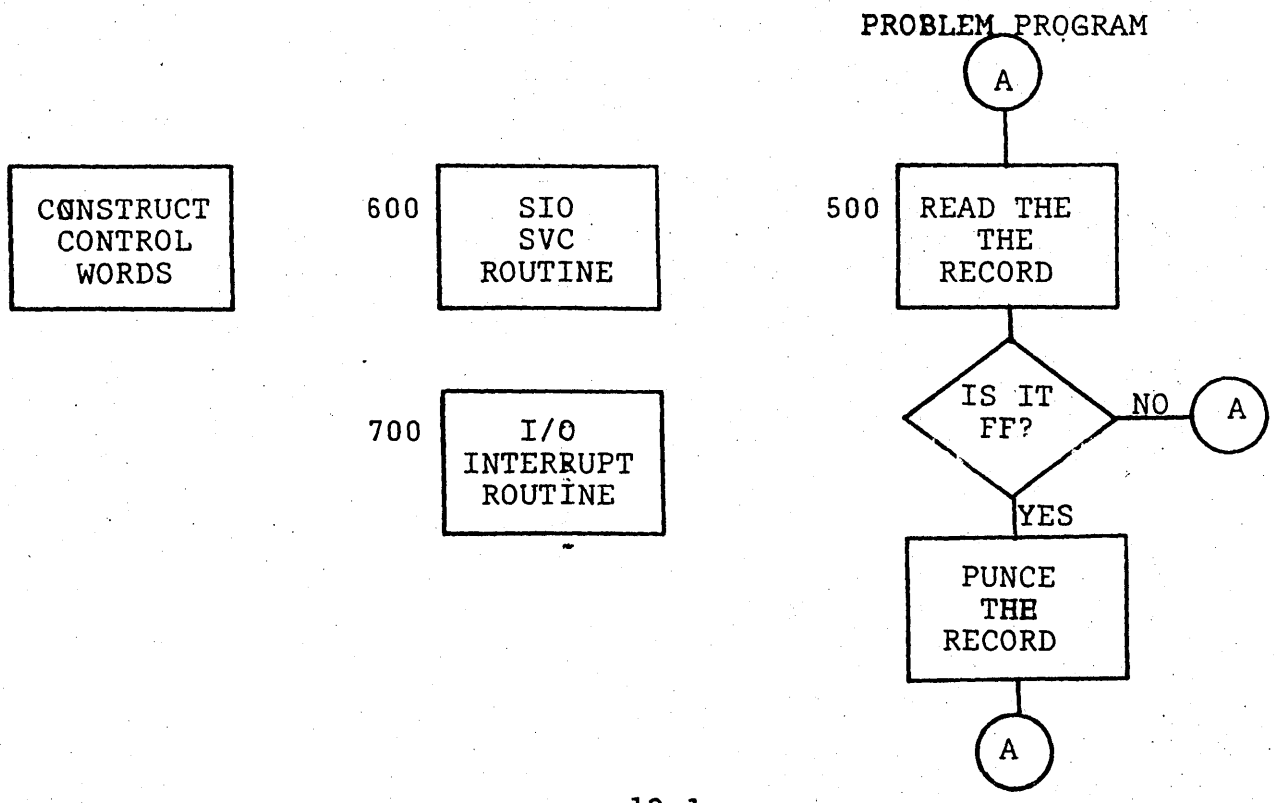
PROJECT

1. Construct PSW's and CCW's for specified conditions.
2. Translate a problem program flowchart into a program.
3. Translate an SVC routine flowchart into instructions.
4. Translate an I/O interrupt routine flowchart into instructions.

THE PROBLEM

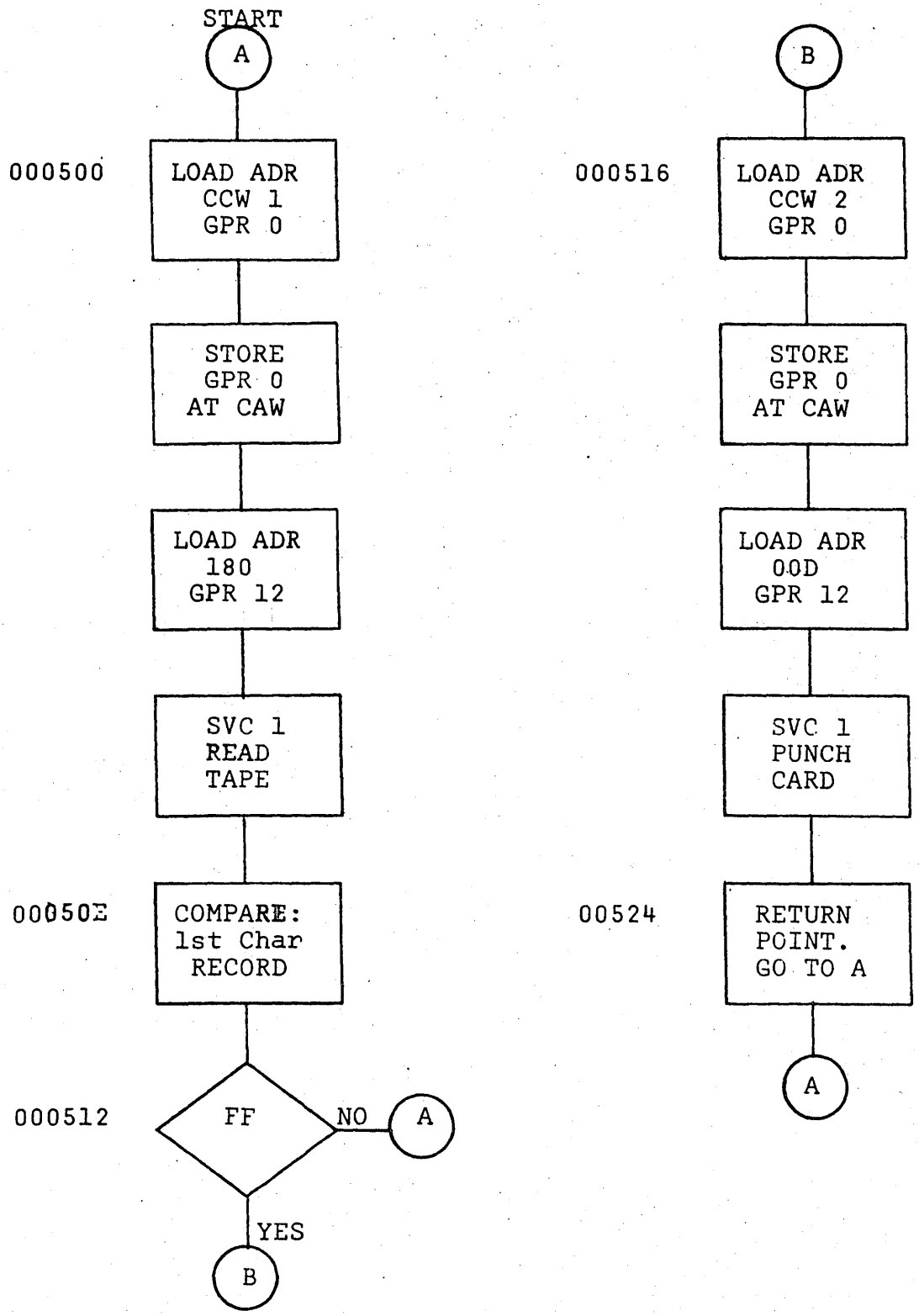
We have a tape on drive #180 with 80-character records. Some key records are identified by an X'FF' as their first byte. We wish to transfer the X'FF' records to cards on punch #00D. Since we are to work in the problem state, we will need to use an SVC to do both Start I/O's. At the end of our tape file we will read a TAPE MARK which will set status bit 39 on in the CSW. At this time we will place the machine in the wait state with all ones in the IC.

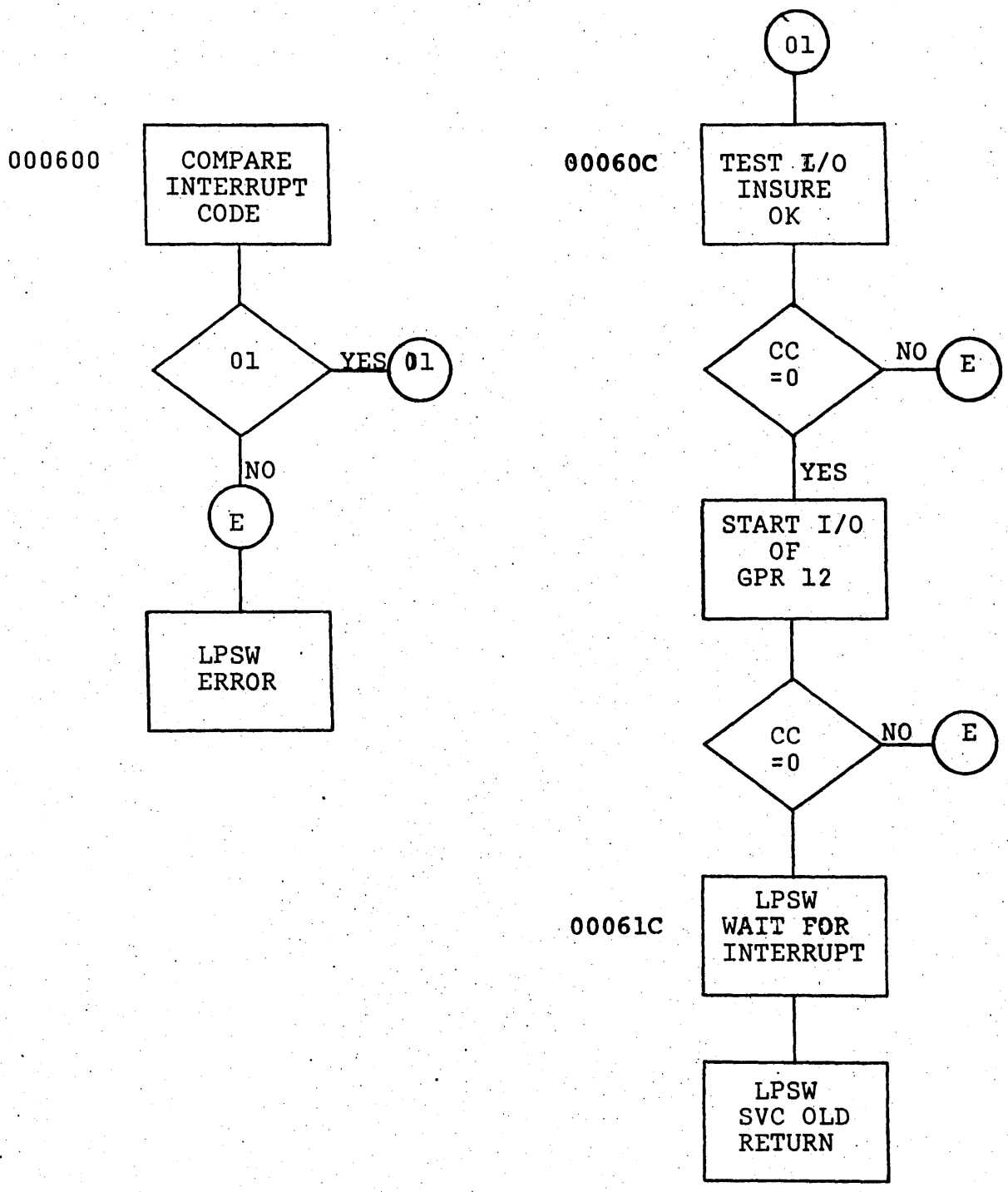
GENERAL FLOWCHART



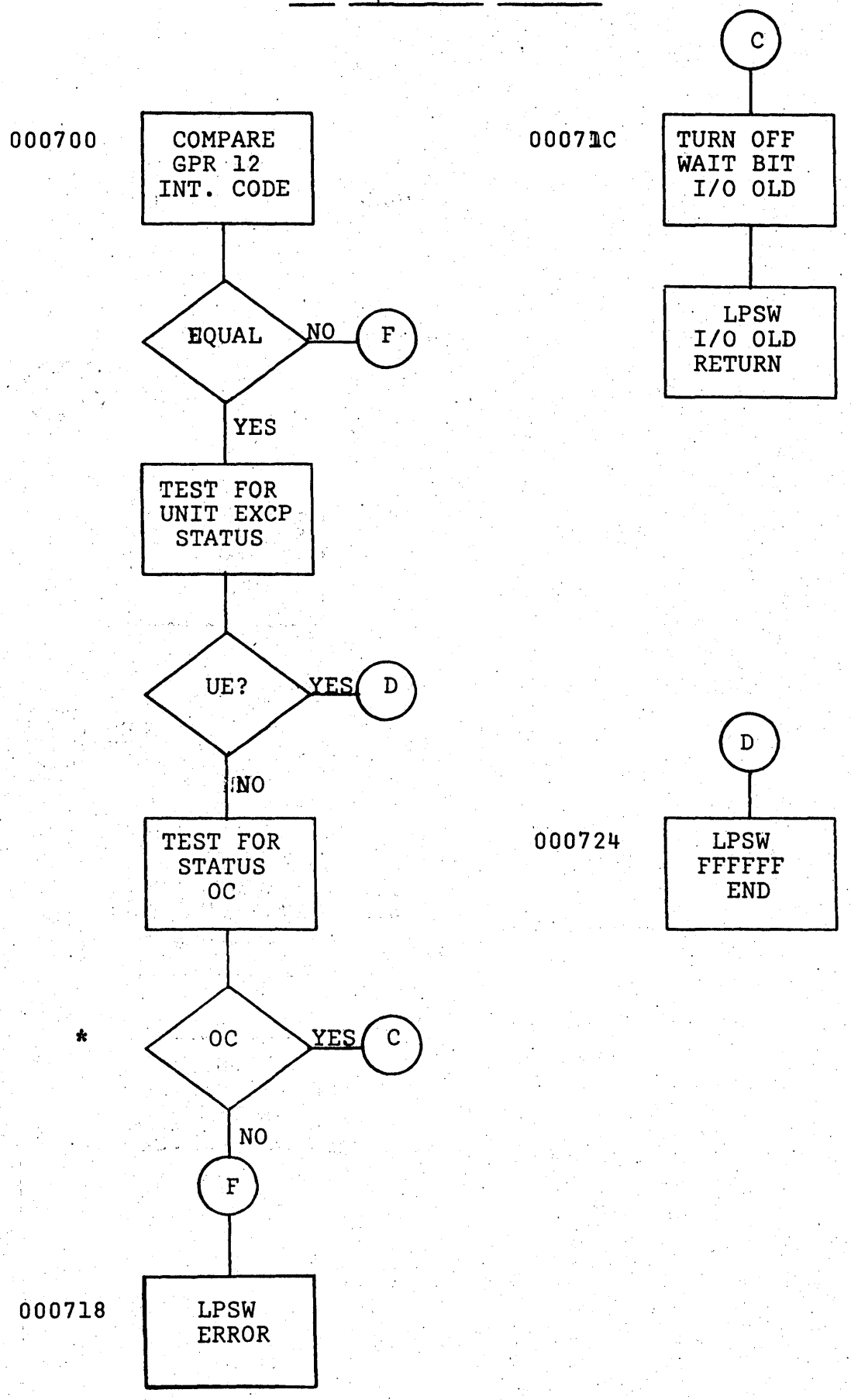
CONTROL WORD CONSTRUCTION (IN HEX)

IPL PSW	000	
(Problem State PSW)	004	
CSW	040	
	044	
CAW	048	
SVC New PSW	060	
	064	
I/O New PSW	078	
	07C	
CCW 1	550	
(Read Tape)	554	
CCW 2	558	
(Punch Card)	55C	
55C		
Wait PSW	560	
(For Interrupt)	564	
End PSW	568	01020000
(When Tape Mask)	56C	00FFFFFF
Error PSW	570	01020000
	574	00F0F0F0
Record Area	5A0	
	5EF	


START
A
000500
LOAD ADR
CCW 1
GPR 0
STORE
GPR 0
AT CAW
LOAD ADR
180
GPR 12
SVC 1
READ
TAPE
00050E
COMPARE:
1st Char
RECORD
000512
FF
NO
A
YES
B
B
000516
LOAD ADR
CCW 2
GPR 0
STORE
GPR 0
AT CAW
LOAD ADR
00D
GPR 12
SVC 1
PUNCH
CARD
00524
RETURN
POINT.
GO TO A
A


000600
COMPARE INTERRUPT CODE
01
YES
01
NO
E
LPSW ERROR
01
00060C
TEST I/O INSURE OK
CC =0
NO
E
YES
START I/O OF GPR 12
CC =0
NO
E
00061C
LPSW WAIT FOR INTERRUPT
LPSW SVC OLD RETURN

I/O INTERRUPT ROUTINE



* "OC" not always valid - See page 12-17.

THE PROBLEM PROGRAM

CODING

```
000500

504

508

50C

50E

512

516

51A

51E

522

000524
```

CODING

```
000600

604

608

60C

610

614

618

61C

000620
```

THE I/O ROUTINE

CODING

```
000700

        704

        708

        70C

        710

        714

        718

        71C

        720

000724
```

27100

CONSTRUCT CONTROL WORDS

Our first step is to build the PSW's and CCW's that we will use in our program. The IPL PSW is used to change the CPU from a program loading state to a running state. It will become our current PSW for our problem program.

The IPL PSW is to have the following characteristics:

1. Machine Check on
2. Problem state on
3. Instruction address of X'000500'
4. All other fields zero.

Enter the first half of this PSW. (8 hex digits)

27101

Record your first half of the IPL PSW on page 12-2 of your book of figures. As we continue, record your answers on page 12-2 and you will have a ready reference.

Enter the second half of the IPL PSW. (8 hex digits)

27102

The SVC new PSW is to have these characteristics:

1. Machine Check on
2. Supervisory state
3. Instruction address of X'000600'
4. All other fields zero.

Enter the first half of SVC new PSW (8 hex digits).

27103

Enter the second half of the SVC new PSW

27104

The I/O new PSW is to have these characteristics:

1. Machine check on
2. Supervisory state
3. Instruction address of X'000700'
4. All other fields zero.

Enter the first half of I/O new PSW.

27105

Enter the second half of I/O new PSW.

27106

The "Wait" PSW will have the following:

1. Permit channel 0 and 1 interrupts.
2. Machine check on
3. Wait bit on
4. Supervisory state
5. Instruction address X'000620'
6. All other fields zero

Enter the first half of the "Wait" PSW.

27107

Enter the second half of the "Wait" PSW.

27110
CCW1 will be used to give a Read command to our tape drive. The CCW format is at the top of Reference Card 11 and the 2400 Tape Drive commands are at the bottom of Reference Card 12. CCW1 must have:

1. Command Code for a Read
2. Data Address at X'0005A0'
3. Only the SILI flag on. (Bit 34)
4. Count to be 80 decimal bytes.

Enter the first half of CCW1.

27111
Enter the second half of CCW1.

27112
CCW2 is to have the following.

1. 2540 Punch Command (Reference Card 12)
   Type BB, SS=00; D=0
2. Data Address at X'0005A0'
3. No Flags
4. Count to be 80 decimal bytes.

Enter the first half of CCW2.

27113
Enter the second half of CCW2.

27114
The CAW is set up by the Problem programmer. The CSW will be set up by hardware after the I/O interrupt. See 12-17 for more information on the CSW.

At the "END" (after the tape mark), the IAR or IC (instruction counter) will contain all ones due to the END PSW. If an error occurs, the IC will contain F0F0F0 due to the Error PSW. This is a common way to signal the operation of machine conditions.

No Question. Just EOB.

27120
Now we will deal with the Problem program (12-3). The first instruction is a Load Address of CCW1 into GPR0. CCW1 location is specified on 12-2. Enter the instruction.

27121
Now store the CCW1 address at X'000048.'

27122
CCW1 says Read, 80 bytes into location X'0005A0. The CAW points to CCW1. This information will tell the channel what we want to do, where the data will go and how much. The only thing left is "Who" supplies the data. We will use GPR 12 to tell the SVC routine the device address. Enter the command to Load Address, X'180', into GPR 12.

27123
You can record your answers to the problem program on page 12-6. Enter the instruction for an SVC with a code of X'01'.

27124
This will be all that is required of the problem programmer as to getting the tape record. The next problem program instruction will be the compare for X'FF' as the first byte of the record. But there will be a considerable delay while the SVC interrupt and a subsequent I/O interrupt takes place. Nevertheless, the next instruction of the problem program will be a compare (CLI) of the first byte of the tape record. Enter that instruction.

27125
The CC will now reflect the result of the above CLI. If the first byte is X'FF';we will have to deal further with this record. Since this was a logical compare, the result must be "equal" or "A Low ." If it is not equal, we will get the next tape record and test that, etc. Enter a BC instruction to go to connector A if the byte is not X'FF.'

27126
It may appear that our branch could go to location X'00050C' (in this case it could have) rather than X'000500.' But since we do not insure the integrity of our GPR's and CAW could (it doesn't) change we do it this way. No Question. EOB.

27130
It may appear that the next instruction should be a branch to connector "B" but it is not necessary. Connector "B" will be our next sequential address. We get there only if our records first byte was X'FF.' The instruction at X'000516' is to LA of CCW2 into GPR 0. Enter it.

27131
Now store it at "CAW".

27132
Place the address of the punch in GPR 12.

27133
Do an SVC with code X'01' again.

27134
This demonstrates that the SVC routine does not care about the kind of device (tape or card punch) or which command is used (read or write). The SVC routine issues the SI/O. The CAW points to the CCW who has the order. GPR 12 has the device address. After the SVC and I/O interrupts, control will be given to the instruction following the SVC. That instruction will be a BC, unconditionally, to connnector A. Enter it.

27135
Now you will return to location X'000500' and get the next card. That completes the coding for the problem program. The next step will be the SVC program. No Question. EOB.

27140

THE SVC ROUTINE

The SVC routine is to start at location X'000600.' Its major function is to Start I/O and to return control to the instruction following the instruction that called for it. That is the instruction whose address is in SVC old PSW. First we will check that the call is for routine #01. We will do this with a CLI instruction whose immediate field is X'01' against the fourth byte of SVC old PSW. Enter it.

27141

Record the SVC coding on page 12-7.

Your next instruction is to be a BC on equal to location X'00060C.' Enter it.

27142

Usually there are many SVC routines and we would select the correct one by examining the interrupt code. In our case we are going to recognize any code other than X'01' as an error. We will LPSW of our error PSW if the code is not X'01'. The error PSW location is noted on page 12-2. Enter the instruction.

27143

We expect to bypass the LPSW instruction at X'000608' and do the instruction at connector 01 which is to be a Test I/O. Enter the coding for the TI/O instruction. (don't forget where we placed the device address).

27144

We expect that the CC after the TI/O will be 0 (available). On any condition other than 0 you are to branch to the instruction at X'000608' (Load error PSW). Enter that BC instruction.

27145

The next instruction is a Start I/O. Enter it.

27146

The instruction at X'000618' is to be identical to the instruction at X'000610.' It may be redundant, but I like it. EOB.

27147

We will now go into the wait state while the device is operating. Enter the LPSW instruction to activate the "Wait" PSW. (See 12-2)

27150
We are going to "wait" (the computer will hang) while the device is reading tape or punching the card. When the device is completed an interrupt will be generated and the device status will be examined. If the status is OK control will be returned to the instruction whose address is in the "wait" PSW. The instruction that loaded the "wait" PSW is at X'00061C' and the PSW points to X'000620'. The last instruction of the interrupt routine must do an LPSW of the I/O old PSW ("wait" PSW). But just before the "wait" PSW is loaded something must be done or we will again go into the WAIT STATE. Which bit of the "wait" PSW must be changed?

27151
The I/O routine has returned us to the SVC routine by loading the "wait" (I/O old) PSW. We must now return to the Problem program. The Problem program PSW is at location X'_____'.

27152
Enter the instruction that will load the Problem PSW.

27153
The Problem PSW will return us to address X'00050E or X'000524' depending on which SVC instruction was issued. In either case the SVC routine is ended. EOB.

27160

THE I/O INTERRUPT ROUTINE

We reach this routine due to an I/O interrupt. The sequence of events is: The Problem Program issues an SVC, the SVC routine issued a Start I/O and then went into the Wait State. The I/O interrupt routine is flowcharted on 12-5 and you can record the machine code instructions on 12-8.

The device that caused the interrupt must be (in our simple program) the same device that was started. The address of the started device is in GPR 12 and the interrupting device's address is in I/O old PSW's interrupt code. To compare these we will use a _____. (C, CR, CH, CLR, CL, CLC, CLI)

27161

Enter the compare instruction for the above.

27162

Enter a BC instruction (for any condition except equal) to address X'000718'.

27163

The device and channel status are contained in the CSW. The CSW is at address _____.

27164

The 3rd HW of the CSW contains the status (see Reference Card 11). We will ignore the channel status byte (bits 40-47) and examine the device status byte. What is the address of the device status byte?

27165

Enter a TM instruction to check for Unit Exception only. Unit Exception means that we have read the Tape Mark.

27166

Enter a BC instruction for condition "one" to branch to address X'000724.'

27167

If the device status was not "UE" we will check for Channel End and Device End. This test should be more complicated than shown but we will test as though both conditions were always presented simultaneously (See 12-17 for explanation). Enter a CLI instruction for the above.

27170
Enter a BC instruction to branch to address X'00071C' if the CC is zero. (status equal X'0C')

27171
If the status was not X'0C' we have an error condition. In this case we will load the Error PSW. Enter the LPSW instruction (the error PSW is at X'000570).

27172
We expect to skip the error PSW by branching from address X'000714' to X'00071C'.
Our interrupt status was OK so we may return to our SVC routine. This could be done in our program by a simple branch to address X'000620'. But in order to show a more common method we will load the I/O old PSW. If there were more ways to enter the Wait State we would have to load the I/O old PSW to insure we returned to the correct SVC routine. Another point, the wait PSW had, naturally, bit 14 set on. Before we load the I/O old PSW we must turn bit 14 off. In turning off bit 14 we must insure that we do not change bits 8-13 and 15 as they may, in general, contain other bit on. The simplest way of doing this is with an AND or an Exclusive Or instruction in the SI type (NI or XI). Enter the instruction to turn off bit 14 and no other of the I/O old PSW. (Use either a NI or an XI instruction)

27173
Now enter the instruction to load the I/O old PSW as the current PSW.

27174
The above instruction will take us to address X'000620' (return us to the SVC routine).
Location X'000620' of the SVC routine is a LPSW of SVC old PSW which returns us to either location X'00050E' or to X'000524' depending on which SVC instruction was issued in the Problem program. EOB.

27175
The last instruction of the I/O interrupt routine is at connector "D". This instruction is to load the "END" PSW when we read the Tape Mark (unit exception). Enter that instruction.

CHANNEL END - DEVICE END

The flowchart on 12-5, Compare Status 0C, is not valid for a 2540 card reader-punch. The Control Unit of the 2540 contains a buffer that will hold 80 characters and as soon as the channel has sent over the 80 characters, the Control Unit will signal Channel End without a Device End. The 2540's Control Unit will then transfer the buffer data to "Punch Magnets" that will permit the punch dies to punch the correct holes in the card. The punching is done a row at a time until all twelve rows are punched and the card is moved out of the punch station. Now the device is ready to accept a new order and the control unit will signal the channel with another status condition, Device End. There would actually be two interrupts and the interrupt routine would have to have a scheme to keep track of the status.

We have ignored this on our little program for purposes of simplicity.

On the other hand the test of "0C' is valid on a read tape because the tape is an unbuffered device. The channel is connected to the control unit until all data is transferred and until the tape is stopped in the IRG. At this time the control unit sends a Channel End and Device End in one status byte which becomes bits 32 to 39 (B'00001100' = X'0C') of the CSW.