HONEYWELL BULL BILLERICA	SPECIFICATION NUMBER 60165904	DISTRIBUTION CODE	SHEETS 1/xx	REV 3
CUSTOM AND SPECIAL PRODUCTS	TITLE: **ENGINEERING PRODUCT SPECIFICATION** **16-BIT CUSTOM PROCESSOR**			
PREPARED BY R Lemay				
APPROVED BY				

REVISION	AUTHORITY	DATE	SIGNATURE	SHEETS AFFECTED
1	Ultimate	03MAR82	R. Lemay	all
2	Ultimate	17MAY84	R. Lemay	all
3	Honeywell Bull	04JAN88	R. Lemay	see note

This revision is being issued:

1. to reflect a change in the documentation structure. Prior to this revision, the specification data contained in this document and the Technical Description were combined.
2. to announce "THE TRANSFER LANGUAGE COMPILER" (see Section Four).
3. to expose enhancements introduced in the redesign of the product (see Appendix A).

TABLE OF CONTENTS

SECTION 1 INTRODUCTION

1.1 Purpose 1-1

1.2 Scope 1-1

1.3 Disclaimer 1-2

1.4 References 1-2

1.5 Acronyms 1-2

SECTION 2 THE FIRMWARE DICTIONARY

2.1 Naming Conventions 2-2

2.2 Addressing and Sequencing 2-2

2.3 Glossary 2-3

2.4 External Control and Synchronization 2-8

2.5 Declarations 2-11

2.6 Micro-operations 2-12

SECTION 3 FLOWCHARTING CONVENTIONS

3.1 The Symbology 3-1

3.2 Addresses 3-1

3.3 Flow 3-2

3.3.1 Two-way Choice 3-2
3.3.2 Splatters 3-3

SECTION 4 TRANSFER LANGUAGE

4.1 Source File Format 4-1

4.1.1 Line Length 4-1
4.1.2 White Space 4-1
4.1.3 Valid/Invalid Characters 4-1
4.1.4 Comments 4-2

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.2 Lexography 4-2

4.2.1 Case 4-2
4.2.2 Source File Length 4-2
4.2.3 Statement Terminator 4-2
4.2.4 Literal Text Block 4-2
4.2.5 Literal Text Block Restrictions 4-2
4.2.6 Literal Text Block "Compilation" 4-2
4.2.7 Reserved Words 4-3

4.3 Organization of Source File 4-3

4.3.1 Statement Types 4-3
4.3.2 Commentary 4-3
4.3.3 Pre-processor Directives 4-3

4.3.3.1 Include '<pathname>' 4-3
4.3.3.2 Skip <count> 4-3

4.3.4 Local Definitions 4-4

4.3.4.1 <identifier> EQU <predefined identifier> 4-4
4.3.4.2 <identifier> CONST <integer constant> 4-4
4.3.4.3 <label> EXTERN 4-4
4.3.4.4 <label> PUBLIC 4-4

4.3.5 Block Definitions 4-4

4.3.5.1 Block Definition 4-5
4.3.5.2 End 4-5
4.3.5.3 Preserves 4-5
4.3.5.4 Saves 4-5

4.3.6 Procedure 4-5

4.3.6.1 Operation Clause 4-5
4.3.6.2 Assignment Operation Clause 4-6
4.3.6.3 Control Operation Clause 4-6
4.3.6.4 Next Address Specifier Operation Clause 4-6
4.3.6.5 Default Next Address 4-6
4.3.6.6 Procedure Labels 4-6

4.4 Expressions 4-7

4.4.1 Identifier 4-7
4.4.2 Integer Constant 4-7
4.4.3 Evaluation Range 4-7
4.4.4 Operators 4-8

4.4.4.1 Parenthesis 4-8
4.4.4.2 Unary Minus 4-8
4.4.4.3 Unary Tilde 4-8
4.4.4.4 Unary ++ and -- 4-8
4.4.4.5 Unary Bracket Set 4-8
4.4.4.6 Unary ADDR( ) 4-9
4.4.4.7 Unary Select 4-9
4.4.4.8 Binary Operators + and - 4-9
4.4.4.9 Rotate Binary Operators 4-9
4.4.4.10 Underscore Binary Operator 4-10
4.4.4.11 Boolean Binary Operators 4-10
4.4.4.12 Equal Binary Operator 4-10
4.4.4.13 Comma Binary Operator 4-11

4.5 Control Clauses 4-11

4.5.1 Stalls 4-11
4.5.2 Reads 4-11
4.5.3 Writes 4-11
4.5.4 Ldsynd 4-11
4.5.5 Procedure Fetch 4-12
4.5.6 Nofault 4-12

4.6 Next-Address-Specifier Clauses 4-12

4.6.1 Goto 4-12
4.6.2 Splatter 4-12
4.6.3 Call 4-12
4.6.4 Return 4-13
4.6.5 Conditional 4-13

SECTION 5 THE FIRMWARE DEVELOPMENT FACILITY

5.1 FDF Interface 5-2

5.2 The Help Screen 5-3

5.3 The Command Set 5-4

5.3.1 Silo Commands 5-4
5.3.2 Run Controls 5-5
5.3.3 Register Displays 5-6
5.3.4 Epilogue Controls 5-7
5.3.5 Firmware Array Commands 5-8

5.4 Missing-Stall Catcher 5-9

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

INTRODUCTION

The "sixteen-bit" custom processor is a nine megahertz, twenty-four-bit wide, microprogrammable MEGABUS connected firmware engine driven by a ninety-six-bit wide control store word and having a blank identity.

1.1 PURPOSE

This specification imparts information which is necessary for any who wish to microprogram the custom processor. Those who attempt personalization of the custom processor need be capable of writing and testing microcode. For testing microcode, Custom and Special Products offers a Firmware Development Facility which greatly simplifies the task (see section 5).

1.2 SCOPE

This document is intended for the prospective microprogrammer. It describes the operation of the sixteen-bit custom processor at the level of an experienced coder. Others, such as test technicians, might also find the information useful but should refer to the "TECHNICAL DESCRIPTION OF THE 16-BIT CUSTOM PROCESSOR" for a detailed description of hardware and firmware operations.

In addition to this section, this document contains four other sections.

Section 2 contains the firmware dictionary which constitutes the formal specification.

Section 3 contains flowcharting conventions.

Section 4 contains a specification for the Transfer Language Compiler. The Transfer Language allows firmware to be coded at the register-transfer level rather than the "micro-operation" level.

Section 5 contains a description of the Firmware Development Facility which is available to minimize firmware checkout time.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

1.3 DISCLAIMER

The firmware dictionary of Section 2 serves as the specification for the custom processor. The firmware dictionary shall govern in any disagreement between it and other descriptive documents.

1.4 REFERENCES

In order to code firmware to execute on the CUP16, the following additional documents may prove useful:

CUP16 logic block diagrams
for the mother board 60156205

for the daughter board 60156210

Other related documents are:

16-Bit Custom Processor Technical Description 60165905

CUP16 Test procedures..............................71220271

RTL6 assembly language manual LDA-021

ACRONYMS

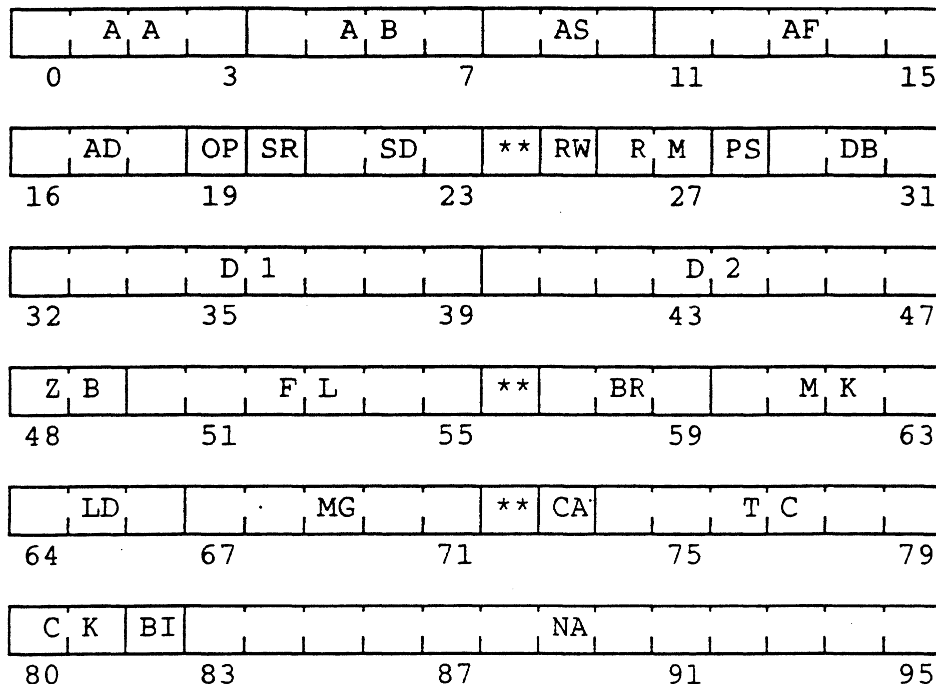
See also Table 2-2

ACRONYM	DEFINITION
C&SP	Custom and Special Products
CUP	CUstom Processor
LBD	Logic Block Diagram (Schematic)
FDF	Firmware Development Facility
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
SCRAM	Stop Code RAM

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

CUSTOM PROCESSOR DICTIONARY

```
MODEL CUP-DICT-1984-05-17;
PARAMETERS;
ROMDEF MAIN,96,16384,006F2000000000000000E000#,$$MCS ;\ -
```



where ** = F/W Parity checks

FIGURE 2-1
FIRMWARE FIELD BIT ALLOCATION

2.1 NAMING CONVENTIONS

Field, Micro, & Step names consist of alphanumeric characters, hyphens, colons, & apostrophes. They must start with an alphabetic character. Generally, the HYPHEN separates words, the APOSTROPHE separates clauses, & the COLON means "equals" or "receives". ("CLAUSE" here refers to descriptions of simultaneous &/or independent actions.)

A double colon is sometimes used to distinguish "partitioned" transfers. A terminal apostrophe signifies logical inversion ("NOT").

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-1
PROM CHIP ALLOCATION

ELSE BANK	Part no.	RLS2.0	RLS2.1	IF BANK	Part no.	RLS2.0	RLS2.1
bits 00-07	-set01rv	L12D		bits 00-07	-set13rv	L10D	
08-15	-set02rv	D11D		08-15	-set14rv	D09D	
16-23	-set03rv	F11D		16-23	-set15rv	F09D	
24-31	-set04rv	J12D		24-31	-set16rv	J07D	
32-39	-set05rv	L07D		32-39	-set17rv	L09D	
40-47	-set06rv	J09D		40-47	-set18rv	J10D	
48-55	-set07rv	L04D		48-55	-set19rv	L06D	
56-63	-set08rv	J04D		56-63	-set20rv	J06D	
64-71	-set09rv	M12M		64-71	-set21rv	M10M	
72-79	-set10rv	K12M		72-79	-set22rv	K10M	
80-87	-set11rv	M07M		80-87	-set23rv	M05M	
88-95	-set12rv	N07M		88-95	-set24rv	N05M	

"-set" is a one letter and three digit PROM set number distinguishing for instance, a disk cache from a 1750A processor, and
"rv" is the firmware revision number of the PROM set.

2.2 ADDRESSING AND SEQUENCING

Firmware steps in the CUP are identified by a "Control Store Address" (CSA) & optionally a mnemonic label. In this discussion, wherever reference is made to a CSA, it should be understood that the associated label may be substituted, but any restrictions on CSA value still apply. CSA's are 14-bit quantities. Values from 2000# through 3FFF# are called the "If bank", 0000# through 1FFF# are called the "Else bank".

Two CSA's are called "TWINS" if they differ by exactly 2000#.

A "SPLATTER BLOCK" is a group of 16 If-bank CSA's which differ only in their LSD (e.g., 2340,2341,2342,...,234F). An Else-bank CSA is said to "correspond" to an If-bank CSA if its twin is a member of the same splatter block. Thus 0342 corresponds to 2349 or 2340 or 2342, etc.

A "SPLATTER VECTOR" is a 4-bit value used, in whole or in part, to select a CSA within a splatter block. This selection is performed by substituting the vector, through a specified mask, into the LSD of the splatter block address.

Firmware sequencing in the CUP is never implicit nor arithmetically defined --- every step specifies its successor or choice of successors.

(1) The succession may be unconditional. For example:

GO-TO(CSA) where CSA = any address in either bank

(2) The succession may involve a choice between an If-bank CSA (CSAI) & a corresponding Else-bank CSA (CSAE), depending on the value of one of 64 "Test Conditions" or their complements --- for example:

IF-ACK(CSAI,CSAE)

branches to CSAI if "ACK" is true, else to CSAE.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

(3) The succession may involve a choice among 2, 4, 8, or 16 members of a splatter block, depending on the number of one-bits in the mask. The choice within the (sub)block is determined by the value of a specified splatter vector.

For example:

BR-P0(F,CSAI)

performs 16-way splatter to a member of the block containing CSAI.

OR

BR-RAMAD(RAC,7,CSAI)

performs 8-way splatter (0,1,2,...,7), controlled by the 3 LSB of RAC.

OR

BR-RAMAD(RAC,E,CSAI)

performs 8-way splatter (0,2,4,...,E), controlled by the 3 MSB of RAC.

(4) Cases 2 & 3 may be combined. For example:

IF-FLAGT5 BR-FLAGS(F,CSAE)

branches to CSAE if FLAGT5 is false, otherwise uses flags T0,T1,T2,T3 to select an entry in the splatter block corresponding to CSAE.

(5) The succession may involve an unconditional subroutine exit. for example:

RETURN

exits to the CSA currently on "top" of the two-level return stack. (At any time, any If-bank CSA may be "PUSH"ed onto the stack.)

(6) The succession may involve the election (by the subroutine) of alternate exits. For example:
RETURN'(D)

exits to the CSA formed by the top stack entry <AND> FFFD#

(7) The succession may involve a choice, based on a test condition, between a subroutine exit (either normal or alternate) and a continuation. For example:

IF-FLAGT6 RETURN(CSAE)

exits to the CSA at the top of the return stack only if FLAGT6 is true, else it continues to CSAE.

2.3 GLOSSARY

Symbols & abbreviations used in descriptive comments are given in Table 2-2.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-2
DICTIONARY SYMBOLS AND ABBREVIATIONS (PART 1)

TERM	BIT #'S	MEANING
&		AND
( )		Bit position(s) or expression grouping e.g. (3-6,9-11) denotes 3,4,5,6,9,10, & 11
, (COMMA)		Data concatenation or separator of items in a sequence of items
,...,		Missing items in implied sequence
<=		Receives
<=SL=		Receives, shifted left one bit position
<=SR=		Receives, shifted right one bit position
<AND>		Logical product (of multi-bit operands)
<GE>		Greater than or Equal to
<IOR>		Inclusive OR (of multi-bit operands)
<LT>		Less Than
<NE>		Not Equal to
<NOT>		Complement (of multi-bit operand)
<XOR>		EXclusive OR (of multi-bit operands)
ADRA	08-31	Address register A, loaded from Z-bus; used to supply address for most non-procedural data references
ADRB	08-31	Address register B, loaded from Z-bus; used to supply address for some non-procedural reads & most writes
ADRP	08-31	Address register P (actually a group of several registers), loaded from the Z-bus and used to supply address for procedural reads. If APLONG is false, ADRP assumes a 512-byte frame size and can be loaded in sections (see APLONG): ADRPH (bits 08-22) is the frame number which is changed infrequently. ADRPL (bits 23-31), the byte position in the current frame, which is loaded by every successful branch. Two versions of ADRP are maintained: One, "PCTR", available to F/W via the D-bus, represents the current offset of code being executed. The other supplies ADRS, and thus the address from which procedure is being prefetched.
ADRS	08-31	Selects Address register ADRA, ADRB, ADRP, or ADRX for delivery to the MEGABUS, the local memory and/or the D-bus. Selection is latched when the request is initiated and held until request is ACK'd or NAK'd.
ADRX	08-31	Register retaining the value of the MEGABUS address bus during the most recently accepted unsolicited MEGABUS cycle.
ALU	08-31	Arithmetic/Logic Unit
ALUF	08-31	ALU Function-generator, capable of: Add, Subtract, Inclusive-OR, AND, NAND, Exclusive-OR, Exclusive-NOR
ALUR	08-31	ALU operand R, chosen from: SP(A), D-BUS, or ZERO
ALUS	08-31	ALU operand S, chosen from: SP(A), SP(B), Q, or ZERO
ALUY	08-31	ALU output Y (available as Z-bus source) equals either ALUF or SP(A)
APLONG		If false, ADRP is load and carry is partitioned 15/9 bits and procedure related page faults are not allowed
APWRAP		Selects 512 or 8192 page size

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-2
DICTIONARY SYMBOLS AND ABBREVIATIONS (PART 2)

TERM	BIT #'S	MEANING
ARAM	08-31	Address-register RAM, containing physical translations of 16 register values
BUFFER-BOUND		see "FRAME-BOUND"
CACHE		A memory which remembers data associated with recent references. The cache is connected to the Custom Processor via a "private" interface. This private interface may instead connect a Local Memory.
CARRY		Carry from MSB of ALU, as captured by IND(1) ALUR ALUS ALUF C A R R Y SIGN SIGN SIGN ADD R-S S-R + + + 0 1 1 + + - 0 0 0 + - + 1 0 1 + - - 0 0 1 - + + 1 1 0 - + - 0 1 0 - - + 1 1 1 - - - 1 0 0
CMDPAR		In addition to address and data parity, a parity bit on the command leads accompanies all MEGABUS transfers. See CNFG.
CNFG		An eight-bit register which controls CUP behavior. All eight bits can be tested: CMDPAR controls Command Parity on MEGABUS CSTEAL controls CUP priority on MEGABUS APLONG controls ADRP partitioning * APWRAP controls assumed page size FCODE1 allows CUP to reinitiate QLTs CNFGA,B,C determine three bits of "Who are you" reply
CS	00-95	Control Store (F/W) output
CSA	00-13	Control Store Address
CSAE	00-13	Control Store Address in Else-bank (0000-1FFF)
CSAI	00-13	Control Store Address in If-bank (2000-3FFF)
CSTEAL		A mechanism which allows the CUP to behave as a low priority MEGABUS requestor even when pulled into a high priority slot. See CNFG.
CYCLE	0-3	Auxiliary counter for selecting block withing Custom Decoder PROM (May not be changed & used in same FW step)
D-BUS	08-31	Data bus input to ALU &/or OUTR. Numerous sources, including concatenation of differently sourced bytes
DOUBLE-ZERO		=1 iff ALUF(08-31)=0 & IND(3)=1, as captured by IND(4)
FCODE1		A mechanism which responds to special MEGABUS cycle (function code 01) which (re)initiates the QLT regardless of the current Custom Processor state. See CNFG.
FIRMWARE	00-95	The CUP executes 96-bit instructions. These instructions can be stored in either a non-writable medium (PROMs) or a writable medium (RAMs). 4K, 8K or 16K of PROMs may be installed. When RAMs are employed, the array is always 16K words.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-2
DICTIONARY SYMBOLS AND ABBREVIATIONS (PART 3)

TERM	BIT #'S	MEANING
FLAGP	0-7	Eight F/W controlled, F/W testable flags, not cleared between instructions. Current assignments include: 0=Passed CUP & MCA QLTS 1=Passed CUP QLT (if FLAGP0 = 0) 2=Force bus errors (if FLAGP0 = 0) 3= 4= 5=DSASTR inhibit 6=On-line mode 7=Trace mode
FLAGT	0-7	Eight F/W controlled, F/W testable flags, cleared (micro "INT") at each $RNI. In addition to general utility, uses are: 0=Local splatter 1= " " & shift end-effects 2= " " 3= " " 4= 5= 6=SCRAM input, invert SCRAM output 7=
FRAME-BOUND		=1 iff Z-bus(08-22) <NE> D-bus(08-22) as captured by IND(6)
FWPROM	00-95	A non-writable firmware array (and also a micro calling for firmware execution to emanate from PROMs)
FWA	00-13	Firmware Write Address register
FWR	00-95	Firmware Write (data) Register
FWRAM	00-95	A writable firmware array (and also a micro calling for firmware execution to emanate from RAMs)
HEX-DECODER	16-31	Bit mask formed by setting to 1 the bit whose position number is 16 + the numeric value of RAMAD, & setting to 0 all other bits
IFF		If and only if
IND	0-7	Indicator register. Samples, on command, values of 8 variables for subsequent testing and/or other use. 0=Overflow indicator 1=Carry indicator 2=Sign indicator 3=Zero indicator 4=Double-zero indicator 5=Odd indicator 6=Frame-bound indicator 7=Stop-code indicator
INRA	16-31	Buffer for receiving non-procedural data requested using ADRA, & for first half of doubleword data
INRB	16-31	Buffer for receiving non-procedural data requested using ADRB, & for second half of doubleword data
INRX	16-31	Buffer for receiving inquiry identification word during unsolicited MEGABUS cycles ("interrupts")
LSB		Least Significant Bit(s)
LSD		Least Significant Digit(s)

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-2
DICTIONARY SYMBOLS AND ABBREVIATIONS (PART 4)

TERM	BIT #'S	MEANING
LOCAL MEMORY		A dual-ported memory array which may be connected to the Custom Processor via a "private" interface. This same "private interface may instead connect a cache.
MCA		Micro-Code Analyzer (or serial number): when "IF-MCA" test says serial-number unit is connected, then N-th byte of serial number is obtained by emitting N to Z-bus(24-31) in one step & executing "Z:OPT" in the next step. When "OPT:Z" strobe is issued, Z-bus bits which equal "one" invoke corresponding actions on the serial-number unit (if present): Z-BUS(08-15) = RFU Z-BUS(16) = Traffic-light on Z-BUS(17) = Traffic-light off Z-BUS(18-31) = RFU
MSB		Most Significant Bit(s)
MSD		Most Significant Digit(s)
NN		2-hex-digit (8 bit) literal constant
NNNN		4-hex-digit (16 bit) literal constant
ODD		Z-bus odd (bit 31=1), as captured by IND(5)
OPREG	0-7	OP-code register loaded by micro LD-OP from P-bus(0-7), to address custom decoder PROM
OUTR	16-31	Output register, loaded from D-bus(16-31), drives data to MEGABUS and to Local interface in all write transactions
OVERFLOW		Arithmetic condition, captured by IND(0); where like-signed operands give opposite-signed sum, or unequal-signed operands give difference with sign opposite to minuend's ALUR SIGN / ALUS SIGN / ALUF SIGN — OVERFLOW ADD R-S S-R + + + — 0 0 0 + + − — 1 0 0 + − + — 0 0 1 + − − — 0 1 0 − + + — 0 1 0 − + − — 0 0 1 − − + — 1 0 0 − − − — 0 0 0
P6SYNC		An indicator signifying that a MEGABUS cycle has occurred since FLAGP6 was last set (intended for testing)
P-BUS	0-7	Bus supplying next byte of procedure from prefetch buffer to RAA, RAB, RAC, splatter logic, Opreg, and/or D-bus.
PCTR	23-31	Counter which tracks byte offset of procedure within current frame (see "ADRP")
PROM		A non-alterable medium in which (for instance) a firmware array is stored.
Q	08-31	"Quotient" register in RALU

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 2-2
DICTIONARY SYMBOLS AND ABBREVIATIONS (PART 5)

TERM	BIT #'S	MEANING
RAA	0-3	Register loaded from P-bus(4-7) for use by RAMAD. Typically retains 2nd nibble of each instruction.
RAB	0-3	Register loaded from P-bus(0-3) for use by RAMAD. Typically retains 3rd nibble of each instruction.
RAC	0-3	Register loaded from P-bus(4-7) for use by RAMAD. Typically retains 4th nibble of each instruction.
RAD	0-3	Counter loaded from shifter(28-31) for use by RAMAD, & also as iteration control. Incremented by "INC-RAD" & by "IF...-RAD:F"
RALU	08-31	Register-file (see "SP") & ALU, constructed of 6 AMD#2901 chips
RAMAD	0-3	ARAM address MUX (= RAA, RAB, RAC, or RAD) also available to hex-decoder and/or to splatter logic
RPSYNC		An indicator signifying that a MEGABUS cycle has occurred since an interrupt from the MEGABUS was received (intended for testing)
SCRAM		Stop-Code RAM, 256 x 1, capable of recognizing stop-codes in byte string operations. Addressed by D-bus(16-23), output available to be captured by IND(7)
SD		Shift Distance (rotate left/right any multiple of four bit positions)
SEMA4		An indicator which detects if a write-unlock has occurred since my most recent read-unlk
SHIFTER	08-31	Z-bus rotated by SD to feed D-bus, &/or SHRG, &/or RAD
SHRG	08-31	SHifter ReGister. On command, captures "shifter" output for later use.
SIGN		ALUF(08), as captured by IND(2)
SP	08-31	Scratchpad (RALU register file) containing 16 work locations, 0-15 (or 0-F).
SP(A)	08-31	First addressed SP entry, available as input to ALUF &/or directly to ALUY
SP(B)	08-31	Second addressed SP entry, available as input to ALUF &/or as receiver from ALUF
STOP-CODE		Any of up to 7 byte values defined to control termination of a byte string instruction
SYNDROME	08-31	Snapshot of status at latest DSASTR (or use of micro "LD-SYND"). Bits represent: 08=Master Clear 09=Powering up 10-13=CUP channel # 14=0 15=Addr. parity bit 16=Timout 17-20=0 21=Data red 22=Data parity,left 23=Data parity,right 24=Proc UAR 25=Proc RED 26=Proc parity 27-28=0 29=FW parity,left 30=FW parity,midl 31=FW parity,rght
TIMER	08-31	A loadable and readable counter which counts firmware steps
UAR		Unavailable Resource, sensed by either MEGABUS timeout (address unrecognized), or procedure crossing frame boundary
UNLOCK		An indicator which detects if an unlock has occurred since my most recent NAK'd cycle
Z-BUS	08-31	Bus supplied by ALUY &/or INRA or INRB, feeds shifter, address registers, ARAM
ZERO		=1 iff ALUF(08-31)=0, as captured by IND(3)

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

2.4 EXTERNAL REFERENCE CONTROL AND SYNCHRONIZATION

The initiation, monitoring, and consummation of external references either via the MEGABUS or via the Local interface (to the cache or Local Memory) requires reasonable care in the use of resources which may still be committed to a previous activity. The CUP has been designed to provide this care automatically whenever practical. However, there remain two kinds of situation in which the microcoder must assume responsibility for defining the required interlock:

1- the degree of interference to be protected against is a function of the sequence of firmware steps executed. (Automated protection for the worst-case sequence would have had to sacrifice performance.)

2- the activity involved is infrequent, and was not deemed to warrant the extra expense of making the interlock automatic.

When an interlock is needed, it is provided by an action referred to as a "STALL": the CUP internal clock pauses near the end of a specified step until a specified condition is satisfied. (If the condition was already satisfied, the clock does not pause, and the stall has no effect.) The CUP hardware provides four kinds of stall conditions to be satisfied:

1- STALL:EMPTY (automated only) -- the clock stalls when the step about to be entered will consume, examine, test, or otherwise depend on the next byte in the procedure stream, and the procedure prefetch buffer is empty. The stall is released as soon as the automatic prefetch mechanism supplies the needed byte of procedure.

2- STALL:ACK -- the clock stalls until/unless the most recently initiated request has been either accepted or rejected. Such a stall is appropriate when the coder wishes to test for possible rejection, to reload an address or data register committed to the previous transaction, or to examine one of the other address registers.

3- STALL:INRA -- the clock stalls until/unless input data register INRA has received whatever data it is due to get as a result of the most recent request. (Note that no stall occurs unless the most recent request was a read-request with at least part of the data destined for INRA.) Such a stall is appropriate when a double-word read has been initiated and the coder wishes to consume the first of the two words and/or test for the possibility that, because of an unavoidable boundary crossing in the memory, the double-word request cannot be satisfied as such, so that the second word must be read separately.

4- STALL:BUSY -- the clock stalls until/unless the CUP's external (Local or MEGABUS) interfaces are quiescent (i.e., the most recent activity has been concluded). Such a stall is appropriate before using the last (or only) word returned in response to a read request, or before initiating a request ("PREFETCH" or "WR-NON-MEM(TEST-P)") which does not automatically provide the required stall action.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

Stalls, when required, must occur subsequent to the last previous request initiation and prior to the step which threatens to use the committed resource. Exception: when the threat involves only register reloading (which occurs at the end of the firmware step), the stall can be concurrent with the threatened action.

Table 2-3 summarizes the circumstances requiring inclusion of "STALL" micro's, as a function of the current (most recently initiated) external activity, and the "threatened" action.

TABLE 2-3
DICTIONARY SYMBOLS AND ABBREVIATIONS

CURRENT REQUEST IN PROGRESS using ADR- / THREATENED ACTION	WR-...		RD-MEM-WORD		RD-MEM-DBLW		proc. fetch ahead	un-known
	A	B	A	B	A	B	P	
WRITE-NON-MEM(TEST-P)	B	B	B	B	B	B	B	B
PREFETCH	B	B						
Z:INRB	-	-	-	B	B	B	-	B
Z:Y-INRB	-	-	-	B	B	B	-	B
Z:INRA	-	-	AB	-	AB	AB	-	AB
Z:Y-INRA	-	-	AB	-	AB	AB	-	AB
D:ADRS ADRS:A	-	KB	-	KB	-	KA	KB	KB
D:ADRS ADRS:B	KB	-	KA	-	KA	-	KB	KB
D:ADRS ADRS:P	KB	KB	KA	KB	KA	KA	-	KB
D:ADRS ADRS:ADRX	KB	KB	KA	KB	KA	KA	KB	KB
IF-ACK	KB	KB	KA	KB	KA	KA	?	?
IF-NOT-ACK	KB	KB	KA	KB	KA	KA	?	?
IF-DBLPL	?	?	?	?	A	A	?	?
IF-NOT-DBLPL	?	?	?	?	A	A	?	?
ADRA:Z	CB	-	CA	-	CA	-	-	CB
ADRB:Z	-	CB	-	CB	-	CA	-	CB
OUTR:D	CB	CB	-	-	-	-	-	CB

KEY:
- \- no explicit stall needed
- A prior STALL-INRA
- B prior STALL:BUSY or ADRPL:Z
- AB prior STALL:INRA, STALL:BUSY, or ADRPL:Z
- KA prior STALL:ACK, STALL:INRA, STALL:BUSY, or ADRPL:Z
- KB prior STALL:ACK, STALL:BUSY, or ADRPL:Z
- CA prior or concurrent STALL:ACK, STALL:INRA, STALL:BUSY, or ADRPL:Z
- CB prior or concurrent STALL:ACK, STALL:BUSY, or ADRPL:Z
- ? situation should not arise

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

2.5 DECLARATIONS

```
PARITY ODD,24,00/24,25/7,127/1      \ Parity on 1st 3rd of F/W word\;
PARITY ODD,56,32/3,28/1,48/8,57/3   \ Parity on 2nd 3rd of F/W word\
         ,100/12,127/1,133/13;
PARITY EVEN,72,64/3,68/1,71/1,73/19 \ Parity on 3rd 3rd of F/W word\
         ,96/8,116/11,127/1,168/4;
ARGDEF AA  (00/4)               \ RALU A-select\
         A/A# B/B# C/C# D/D# E/E# F/F#;
ARGDEF AB  (04/4)               \ RALU B-select\
         A/A# B/B# C/C# D/D# E/E# F/F#;
ARGDEF AD  (16/3,114/1)         \ RALU disposition  \A/A# C/C# E/E#;
ARGDEF AF  (11/5)               \ RALU function            \Z/0;
ARGDEF AS  (08/3,114/1)         \ RALU source(S)   \A/A# C/C# E/E#;
ARGDEF BI  (82/1)               \ Branch condition Invert  \Z/0;
ARGDEF BR  (57/3)               \ BRanch splatter type     \Z/0;
ARGDEF CK  (80/2)               \ Clock speed              \Z/0;
ARGDEF CKM (148/2)              \ Clock speed, manual\
            VF/0       HF/1       HL/2     VL/3;
ARGDEF CY  (172/1)              \ Restrict cycle usage     \Z/0;
ARGDEF D1                       \ D-bus source, bytes 0 & 1\
            (31/1,31/1,31/1,31/1,31/1,31/1,31/1,31/4,133/5)   Z/0;
ARGDEF D2 (138/8)               \ D-bus source, byte 2     \Z/0;
ARGDEF DB  (29/2)               \ D-bus source             \Z/0;
ARGDEF DL                       \ 24-bit literal (17, sign-extended)\
           (31/1,31/1,31/1,31/1,31/1,31/1,31/1,31/4,133/13)   Z/0;
ARGDEF DS  (132/1)              \ Restrict SCRAM-load source  \Z/0;
ARGDEF DX  (29/6,133/13)        \ D-bus source, extended   \Z/0;
PARITY EVEN,35,28/1,133/1       \ D-bus control: invert for "PUSH";
PARITY EVEN,36,28/1,134/1;
PARITY EVEN,37,28/1,135/1;
PARITY EVEN,38,28/1,136/1;
PARITY EVEN,39,28/1,137/1;
PARITY EVEN,40,28/1,138/1;
PARITY EVEN,41,28/1,139/1;
PARITY EVEN,42,28/1,140/1;
PARITY EVEN,43,28/1,141/1;
PARITY EVEN,44,142/1;
PARITY EVEN,45,143/1;
PARITY EVEN,46,144/1;
PARITY EVEN,47,145/1;
ARGDEF FL         (50/6)        \ FLags & indicators, etc.    \Z/0;
ARGDEF FLM (146/2,50/6)         \ Special MEGABUS control\
           LOCK/DA#  UNLOCK/DB# NO-CACHE/C0#;
ARGDEF LDA (65/2,114/1)         \ address-reg load         \Z/0;
ARGDEF LDO (64/1,114/1,114/2)   \ output-data load         \Z/0;
ARGDEF MCA (146/2)  NO-CACHE/3  \ force request to MEGABUS;
ARGDEF MGF (146/1,67/1,170/2,71/1)  \ MEGABUS major function  \Z/0;
ARGDEF MGS (68/2)               \ addr-mux select          \Z/0;
ARGDEF MGS0 (168/1)             \ addr-mux select          \Z/0;
ARGDEF MGS1 (169/1)             \ addr-mux select          \Z/0;
ARGDEF MK  (108/4)              \ Mask for splatters\
           A/A# B/B# C/C# D/D# E/E# F/F#;
ARGDEF MLA (116/2,119/2)        \ Interlock ADRA load      \F/F#;
ARGDEF MLB (120/2,123/2)        \ Interlock ADRB load      \F/F#;
ARGDEF MLO (64/1)               \ Interlock OUTR load      \F/F#;
ARGDEF MLP (71/1,117/2,120/1,122/1)\ Interlock ADRP load   \F/1F#;
ARGDEF MNM (67/1,170/2,50/4,119/4) \ MEGABUS non-memory control\Z/0;
ARGDEF MOP (71/1,54/2)          \ MEGABUS non-memory options\
       CMND/4    REPLY/5   RUPT/4     TEST-A/1  TEST-B/2  TEST-P/0;
ARGDEF MPP (67/2,169/2,71/1,126/1)\ Empty-stall for proc-peeks\X/2F#;
ARGDEF MRP (67/1)               \ Interlock requests       \F/F#;
ARGDEF MRQ (67/1,170/2,71/1,119/4)   \ Interlock requests  \F/F#;
ARGDEF MSA (117/3,126/1)        \ Inteslock ADRA select    \F/F#;
ARGDEF MSB (121/1,124/3)        \ Interlock ADRB select    \F/F#;
ARGDEF MSP (116/2,122/3,126/1)  \ Interlock ADRP select    \F/F#;
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
PARITY EVEN,67,116/11;\
              116 117 118 119 120 121 122 123 124 125 126
-    Load ADRA  X   X       X   X
-    Load ADRB                  X   X       X   X
-    Load ADRP      X   X       X       X
-    Sel  ADRA      X   X   X                           X
-    Sel  ADRB                      X           X   X   X
-    Sel  ADRP  X   X                   X   X   X       X
-    Request                X   X   X   X
;
PARITY  ODD,69,168/2              \ Control PEEK=13# if PTAKE=0, ;
PARITY EVEN,70,170/2              \                 =15# if PTAKE=1  ;
PARITY EVEN,73,146/2              \ Control use of Local interface;
BRCHFLD NAB ,ABS,113/1,83/9, 96/4   \ Next address (splatter-branch);
BRCHFLD NAE ,ABS,113/1,83/9,100/4   \ Next address (Else);
BRCHFLD NAG ,ABS, 82/1,83/9, 96/4   \ Next address (Go-to);
BRCHFLD NAI ,ABS,112/1,83/9,104/4   \ Next address (If);
PARITY EVEN,60,100/1,104/1,108/1\ Build real splatter mask;
PARITY EVEN,61,101/1,105/1,109/1;
PARITY EVEN,62,102/1,106/1,110/1;
PARITY EVEN,63,103/1,107/1,111/1;
PARITY EVEN,92,96/1,100/1         \ Build LSD of NA;
PARITY EVEN,93,97/1,101/1;
PARITY EVEN,94,98/1,102/1;
PARITY EVEN,95,99/1,103/1;
ARGDEF   OP   (19/1)              \ Load OPREG                   \Z/0;
ARGDEF   OPT  (114/1)             \ Non-funct. memo for OPT:Z    \Z/0;
ARGDEF   OPTA (150/4)             \    argument for OPT:Z        \W/0;
ARGDEF   PE   (127/1)             \ Force F/W parity errors      \Z/0;
ARGDEF   PS   (28/1)              \ Push to stack                \Z/0;
BRCHFLD  PSA  ,ABS,28/1,133/13    \ Address pushed to stack;
ARGDEF   PSM  (128/4)               \ Non-funct. memo for map    \Z/0;
ARGDEF   RM   (26/2)              \ ARAM address source select\
              RA/0      RB/1        RC/2      RD/3;
ARGDEF   RW   (25/1)              \ ARAM write control           \Z/0;
ARGDEF   SD   (21/3,114/2)        \ Shift distance\
              R4/4      R8/8        R12/12    R16/16    R20/20
             L4/20     L8/16       L12/12    L16/8     L20/4;
ARGDEF   SR   (20/1)              \ Shift-out hold               \Z/0;
ARGDEF   TC   (112/2,74/6)        \ Test Condition               \Z/0;
ARGDEF   TCX  (74/6)              \ Pseudo-Test Condition        \Z/0;
ARGDEF   ZB   (48/2)              \ Z-bus source                 \Z/0;
     Imaginary bits (beyond 95):
          96-99      NA LSD control
          100-103    NA LSD & mask control
          104-111    Mask control
          112-113    If/Else checking (=1/0)
          114-115    0
          116-126    external request control/checking
          127        Force firmware parity errors
          128-131    Ignore push mask info
          132        Restrict Dbus source for SCRAM load
          133-145    Dbus literal control
```

2.6 MICRO-OPERATIONS

```
MICRO ADRA:Z                          \ Load ADRA <= Z-bus\
              (LDA/4,MLA/F);
MICRO ADRB:Z                          \ Load ADRB <= Z-bus\
              (LDA/6,MLB/F);
MICRO ADRPH:Z                         \ Load ADRP(08-22) <= Z-bus\
              (LDO/8,LDA/2,MLP/F)     \    & OUTR <= D-bus;
MICRO ADRPL:Z                         \ Load ADRP(23-31) <= Z-bus\
              (LDO/0,LDA/2,MLP/F)     \ (can't load OUTR at same time);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO ADRS:A                                  \ ADRS <= ADRA\
              (MGS/2,MGS1/1,MSA/F);
MICRO ADRS:B                                  \ ADRS <= ADRB\
              (MGS/3,MSB/F);
MICRO ADRS:P                                  \ ADRS <= ADRP\
              (MGS/1,MSP/3F#);
MICRO ADRS:ADRX                               \ ADRS <= ADRX\
              (MGS/0,MGS1/1);

MICRO BR-P0(MK,NAB)          \ Splatter on P-bus(0-3)\
              (BR/1,MPP/X);
MICRO BR-RAMAD(RM,MK,NAB)\ Splatter on RAA, RAB, RAC, or RAD\
              (BR/2);
MICRO BR-FLAGS(MK,NAB)       \ Splatter on FLAGT(0-3)\
              (BR/3);
MICRO BR-OPERAND(MK,NAB) \ Splatter on Z-bus(31),RAD(1-3)\
              (BR/4);
MICRO BR-ARITH(MK,NAB)       \ Splatter on \
              (BR/5)         \ I-SGN,I-ZRO,CARRY(08),Q(31 SRO)\;
MICRO BR-DECODE(MK,NAB)      \ Splatter on custom decode of Pbus(0-7),\
              (BR/6,CY/1)\                             & Cycle counter\;

MICRO C(CKM)                 \ Manual clock gear shift override;
MICRO C0(CK/0) ;
MICRO C1(CK/1) ;
MICRO C2(CK/2) ;
MICRO C3(CK/2) ;
MICRO C4(CK/3) ;
MICRO C5(CK/3) ;
MICRO C6(CK/3) ;
MICRO C7(CK/3) ;

MICRO D:ADRS                 \ D-bus <= ADRA, -B, or -P\
              (DX/70004#,DS/1);
MICRO D:RAA-D                \ D-bus <= 00, RAA, RAB, RAC, & RAD\
              (DX/63000#);
MICRO D:HEX(RM)              \ D-bus <= 00, Hex-decoder\
              (DX/62001#);
MICRO D:INRX                 \ D-bus <= 00, Inquiry identification\
              (DX/62200#);
MICRO D:LIT(DL)              \ D-bus <= 17-bit literal   \
             (DB/0)          \ sign-extended to 24       \;
MICRO D:PCTR                 \ D-bus <= 000,9-bit program counter\
              (DX/62010#);
MICRO D:PROC                 \ D-bus <= 0000, Next procedure byte\
              (DX/20020#,MPP/X);
MICRO D:REG(RM)              \ D-bus <= Selected ARAM loc\
              (DX/60008#,DS/1,RW/0);
MICRO D:REG0                 \ D-bus <= ARAM loc zero\
              (DX/70008#,DS/1,RW/0);
MICRO D:SHRG                 \ D-bus <= Saved shifter output\
              (DX/78880#);
MICRO D:SYND                 \ D-bus <= Syndrome from latest DSASTR\
              (DX/70002#)\ or LD-SYND micro usage;
MICRO D:TIMER                \ D-bus <= Timer (08-31) \
              (DX/60000#,FL/04#);
MICRO D:ZSH(SD)              \ D-bus <= Z-bus(rotated by SD)\
              (DX/74440#,DS/1);
MICRO D::00SP                \ D-bus <= 00,SHRG(16-23),P-bus(0-7)\
              (DX/62820#,MPP/X);
MICRO D::00SS                \ D-bus <= 00,SHRG(16-31)\
              (DX/62880#);
MICRO D::00SZ(SD)            \ D-bus <= 00,SHRG(16-23), \
              (DX/62840#,DS/1)   \      Z-bus[rotated by SD](24-31);
MICRO D::00ZS(SD)            \ D-bus <= 00,Z-bus[rotated by SD](16-23), \
              (DX/62480#,DS/1)   \                        SHRG(24-31);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO D::00ZZ(SD)          \ D-bus <= 00,Z-bus[rotated by SD](16-31)\
          (DX/62440#,DS/1);
MICRO D::FFSP              \ D-bus <= FF,SHRG(16-23),P-bus(0-7)\
          (DX/72820#,MPP/X);
MICRO D::FFSS              \ D-bus <= FF,SHRG(16-31)\
          (DX/72880#);
MICRO D::FFSZ(SD)          \ D-bus <= FF,SHRG(16-23),\
          (DX/72840#,DS/1)  \   Z-bus[rotated by SD](24-31);
MICRO D::FFZS(SD)          \ D-bus <= FF,Z-bus[rotated by SD](16-23),\
          (DX/72480#,DS/1)  \                     SHRG(24-31);
MICRO D::FFZZ(SD)          \ D-bus <= FF,Z-bus[rotated by SD](16-31)\
          (DX/72440#,DS/1);
MICRO D::KKP(D1)           \ D-bus <= NNNN,P-bus(0-7)\
          (DB/1,D2/20#,MPP/X);
MICRO D::KKS(D1)           \ D-bus <= NNNN,SHRG(24-31)\
          (DB/1,D2/80#);
MICRO D::KKZ(D1,SD)        \ D-bus <= NNNN,Z-bus[rotated by SD](24-31)\
          (DB/1,D2/40#,DS/1);
MICRO D::SSK(D2)           \ D-bus <= SHRG(08-23),NN\
          (DB/2,D1/88#);
MICRO D::SSP               \ D-bus <= SHRG(08-23),P-bus(0-7)\
          (DX/68820#,MPP/X);
MICRO D::SSZ(SD)           \ D-bus <= SHRG(08-23),\
          (DX/68840#,DS/1)  \  Z-bus[rotated by SD](24-31);
MICRO D::SZZ(SD)           \ D-bus <= SHRG(08-15),\
          (DX/68440#,DS/1)  \  Z-bus[rotated by SD](16-31);
MICRO D::ZSS(SD)           \ D-bus <= Z-bus[rotated by SD](08-15),\
          (DX/64880#,DS/1)  \               SHRG(16-31);
MICRO D::ZZK(SD,D2)        \ D-bus <= Z-bus[rotated by SD](08-23),NN\
          (DB/2,D1/44#,DS/1);
MICRO D::ZZP(SD)           \ D-bus <= Z-bus[rotated by SD](08-23), \
         (DX/64420#,DS/1,MPP/X);             \   P-bus(0-7);
MICRO D::ZZS(SD)           \ D-bus <= Z-bus[rotated by SD](08-23),\
          (DX/64480#,DS/1)                   \   SHRG(24-31);

MICRO ENPROM  (FL/06#)     \ With a delay of one firmware step, \
                           \  execute firmware out of the PROM  \
                           \  array;
MICRO ENBRAM  (FL/07#)     \ With a delay of one firmware step, \
                           \  execute firmware out of the RAM   \
                           \  array;
MICRO FLAGP0:0             \ Perm flag #0 <= 0\
          (FL/30#);
MICRO FLAGP0:1             \ Perm flag #0 <= 1\
          (FL/38#);
MICRO FLAGP1:0             \ Perm flag #1 <= 0\
          (FL/31#);
MICRO FLAGP1:1             \ Perm flag #1 <= 1\
          (FL/39#);
MICRO FLAGP2:0             \ Perm flag #2 <= 0\
          (FL/32#);
MICRO FLAGP2:1             \ Perm flag #2 <= 1\
          (FL/3A#);
MICRO FLAGP3:0             \ Perm flag #3 <= 0\
          (FL/33#);
MICRO FLAGP3:1             \ Perm flag #3 <= 1\
          (FL/3B#);
MICRO FLAGP4:0             \ Perm flag #4 <= 0\
          (FL/34#);
MICRO FLAGP4:1             \ Perm flag #4 <= 1\
          (FL/3C#);
MICRO FLAGP5:0             \ Perm flag #5 <= 0\
          (FL/35#);
MICRO FLAGP5:1             \ Perm flag #5 <= 1\
          (FL/3D#);
MICRO FLAGP6:0             \ Perm flag #6 <= 0\
          (FL/36#);
MICRO FLAGP6:1             \ Perm flag #6 <= 1\
          (FL/3E#);
MICRO FLAGP7:0             \ Perm flag #7 <= 0\
          (FL/37#);
MICRO FLAGP7:1             \ Perm flag #7 <= 1\
          (FL/3F#);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO FLAGT0:0 (FL/20#);     \ Temp flag #0 <= 0\
MICRO FLAGT0:1 (FL/28#);     \ Temp flag #0 <= 1\
MICRO FLAGT1:0 (FL/21#);     \ Temp flag #1 <= 0\
MICRO FLAGT1:1 (FL/29#);     \ Temp flag #1 <= 1\
MICRO FLAGT2:0 (FL/22#);     \ Temp flag #2 <= 0\
MICRO FLAGT2:1 (FL/2A#);     \ Temp flag #2 <= 1\
MICRO FLAGT3:0 (FL/23#);     \ Temp flag #3 <= 0\
MICRO FLAGT3:1 (FL/2B#);     \ Temp flag #3 <= 1\
MICRO FLAGT4:0 (FL/24#);     \ Temp flag #4 <= 0\
MICRO FLAGT4:1 (FL/2C#);     \ Temp flag #4 <= 1\
MICRO FLAGT5:0 (FL/25#);     \ Temp flag #5 <= 0\
MICRO FLAGT5:1 (FL/2D#);     \ Temp flag #5 <= 1\
MICRO FLAGT6:0 (FL/26#);     \ Temp flag #6 <= 0\
MICRO FLAGT6:1 (FL/2E#);     \ Temp flag #6 <= 1\
MICRO FLAGT7:0 (FL/27#);     \ Temp flag #7 <= 0\
MICRO FLAGT7:1 (FL/2F#);     \ Temp flag #7 <= 1\

MICRO F:ADD1   (AF/0);       \ ALUF = ALUR + ALUS + 1\
MICRO F:ADDC   (AF/1);       \ ALUF = ALUR + ALUS + Carry\
MICRO F:ADDC'  (AF/2);       \ ALUF = ALUR + ALUS + (1-Carry) \
MICRO F:ADD    (AF/3);       \ ALUF = ALUR + ALUS\
MICRO F:S-R    (AF/4);       \ ALUF = ALUS - ALUR\
MICRO F:S-R-C' (AF/5);       \ ALUF = ALUS - ALUR - (1-Carry) \
MICRO F:S-R-C  (AF/6);       \ ALUF = ALUS - ALUR - Carry\
MICRO F:S-R-1  (AF/7);       \ ALUF = ALUS - ALUR - 1\
MICRO F:R-S    (AF/8);       \ ALUF = ALUR - ALUS\
MICRO F:R-S-C' (AF/9);       \ ALUF = ALUR - ALUS - (1-Carry)\
MICRO F:R-S-C  (AF/A#);      \ ALUF = ALUR - ALUS - Carry\
MICRO F:R-S-1  (AF/B#);      \ ALUF = ALUR - ALUS - 1\
MICRO F:OR     (AF/F#);      \ ALUF = ALUR <IOR> ALUS\
MICRO F:SR     (AF/10#);     \ ALUF = ALUR <AND> ALUS\
MICRO F:SR'    (AF/17#);     \ ALUF = ALUS <AND> <NOT> ALUR\
MICRO F:XOR    (AF/1B#);     \ ALUF = ALUR <XOR> ALUS\
MICRO F:XNOR   (AF/1F#);     \ ALUF = ALUR <XOR> <NOT> ALUS\
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO F'B:0(AB)              \ SP(B), ALUF, ALUY <= 0\
             (AS/4,AF/10#,AD/6);
MICRO F'B:DEC(AB)            \ SP(B), ALUF, ALUY <= SP(B) - 1\
             (AS/6,AF/7,AD/6);
MICRO F'B:INC(AB)            \ SP(B), ALUF, ALUY <= SP(B) + 1\
             (AS/6,AF/0,AD/6);
MICRO F'Q:0                  \ Q, ALUF, ALUY <= 0\
             (AS/4,AF/10#,AD/0);
MICRO F'Q:DEC                \ Q, ALUF, ALUY <= Q - 1\
             (AS/4,AF/7,AD/0);
MICRO F'Q:INC                \ Q, ALUF, ALUY <= Q + 1\
             (AS/4,AF/0,AD/0);

MICRO FRAMIT                 \ performs actions associated with loading \
      (FL/01)                \ the firmware RAM as a function of \
                             \ CYCLE(4,2,1):
                               CYCLE = 1,1,1 loads FWR(80-95)
                               CYCLE = 1,1,0 loads FWR(64-79
                               CYCLE = 1,0,1 loads FWR(48-63
                               CYCLE = 1,0,0 loads FWR(32-47
                               CYCLE = 0,1,1 loads FWR(16-31
                               CYCLE = 0,1,0 loads FWR(00-15
                               CYCLE = 0,0,1 loads FWA
                               CYCLE = 0,0,0 writes (FWR) at FWA;

MICRO GO-TO(NAG)             \ Unconditional NA\
             (BR/0);

MICRO IF-ACK(NAI,NAE)        \ Branch iff MEGABUS cycle was acknowledged\
             (TC/A4#,BI/0)          \ Requires prior "STALL:ACK";
MICRO IF-NOT-ACK(NAI,NAE)           \ in new designs, use IF-NAK \
            (TC/A4#,BI/1)           \ Requires prior "STALL:ACK";
MICRO IF-APLONG(NAI,NAE) \ Branch iff ADRP is configured \
            (TC/9D#,BI/0,MGS/1)     \ non-partitioned \;
MICRO IF-NOT-APLONG(NAI,NAE)
            (TC/9D#,BI/1,MGS/1);
MICRO IF-AP:512(NAI,NAE) \ Branch iff page size is configured \
            (TC/9D#,BI/0,MGS/2)     \ equal to 512 bytes \;
MICRO IF-NOT-AP:512(NAI,NAE)
            (TC/9D#,BI/1,MGS/2);
MICRO IF-BREAK(NAI,NAE)   \ Branch iff Interrupt, \
             (TC/A7#,BI/0)          \ Tick, or Trace-mode;
MICRO IF-NOT-BREAK(NAI,NAE)
            (TC/A7#,BI/1);
MICRO IF-CACHE(NAI,NAE)   \ Branch iff Cache or Local Memory present \
             (TC/A2#,BI/0)          \ and on-line;
MICRO IF-NOT-CACHE(NAI,NAE)
            (TC/A2#,BI/1);
MICRO IF-CNFG-C(NAI,NAE) \ Branch iff Configuration bit C \
            (TC/9D#,BI/0,MGS/0);
MICRO IF-NOT-CNFG-C(NAI,NAE)
            (TC/9D#,BI/1,MGS/0);
MICRO IF-CNFG-B(NAI,NAE) \ Branch iff Configuration bit B \
            (TC/9E#,BI/0,MGS/0);
MICRO IF-NOT-CNFG-B(NAI,NAE)
            (TC/9E#,BI/1,MGS/0);
MICRO IF-CNFG-A(NAI,NAE) \ Branch iff Configuration bit A \
             (TC/9F#,BI/0,MGS/0);
MICRO IF-NOT-CNFG-A(NAI,NAE)
            (TC/9F#,BI/1,MGS/0);
MICRO IF-CMDPAR(NAI,NAE) \ Branch iff Command Parity enabled \
            (TC/9E#,BI/0,MGS/2);
MICRO IF-NOT-CMDPAR(NAI,NAE)
            (TC/9E#,BI/1,MGS/2);
MICRO IF-CYCLE8(NAI,NAE) \ Branch iff CYCLE(0) = 1 \
             (TC/90#,BI/0,CY/1);
MICRO IF-NOT-CYCLE8(NAI,NAE)
            (TC/90#,BI/1,CY/1);
MICRO IF-CYCLE4(NAI,NAE) \ Branch iff CYCLE(1) = 1 \
             (TC/91#,BI/0,CY/1);
MICRO IF-NOT-CYCLE4(NAI,NAE)
            (TC/91#,BI/1,CY/1);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO IF-CYCLE2(NAI,NAE) \ Branch iff CYCLE(2) = 1 \
           (TC/92#,BI/0,CY/1);
MICRO IF-NOT-CYCLE2(NAI,NAE)
          (TC/92#,BI/1,CY/1);
MICRO IF-CYCLE1(NAI,NAE) \ Branch iff CYCLE(3) = 1 \
           (TC/93#,BI/0,CY/1);
MICRO IF-NOT-CYCLE1(NAI,NAE)
          (TC/93#,BI/1,CY/1);
MICRO IF-CYCLE:F(NAI,NAE) \ Branch iff CYCLE(0-3) = F \
           (TC/94#,BI/0,CY/1);
MICRO IF-NOT-CYCLE:F(NAI,NAE)
          (TC/94#,BI/1,CY/1);
MICRO IF-CSTEAL(NAI,NAE) \ Branch iff cycle steal mode \
          (TC/9E#,BI/0,MGS/1) \ is configured \;
MICRO IF-NOT-CSTEAL(NAI,NAE)
          (TC/9E#,BI/1,MGS/1);
MICRO IF-DBLPL(NAI,NAE)  \ Branch iff Double-pull succeeded \
           (TC/A3#,BI/0);
MICRO IF-NOT-DBLPL(NAI,NAE)
          (TC/A3#,BI/1);
MICRO IF-FALSE(NAI,NAE)  \ Branch never \
           (TC/80#,BI/0);
MICRO IF-NOT-FALSE(NAI,NAE)
          (TC/80#,BI/1);
MICRO IF-FCODE1(NAI,NAE) \ Branch iff Func code 01 is allowed to \
          (TC/9F#,BI/0,MGS/1)  \ start the QLT \;
MICRO IF-NOT-FCODE1(NAI,NAE)
          (TC/9F#,BI/1,MGS/1);
MICRO IF-FLAGT0(NAI,NAE) \ Branch iff Temp flag 0 \
           (TC/B8#,BI/0);
MICRO IF-NOT-FLAGT0(NAI,NAE)
          (TC/B8#,BI/1);
MICRO IF-FLAGT1(NAI,NAE) \ Branch iff Temp flag 1 \
           (TC/B9#,BI/0);
MICRO IF-NOT-FLAGT1(NAI,NAE)
          (TC/B9#,BI/1);
MICRO IF-FLAGT2(NAI,NAE) \ Branch iff Temp flag 2 \
           (TC/BA#,BI/0);
MICRO IF-NOT-FLAGT2(NAI,NAE)
          (TC/BA#,BI/1);
MICRO IF-FLAGT3(NAI,NAE) \ Branch iff Temp flag 3 \
           (TC/BB#,BI/0);
MICRO IF-NOT-FLAGT3(NAI,NAE)
          (TC/BB#,BI/1);
MICRO IF-FLAGT4(NAI,NAE) \ Branch iff Temp flag 4 \
           (TC/BC#,BI/0);
MICRO IF-NOT-FLAGT4(NAI,NAE)
          (TC/BC#,BI/1);
MICRO IF-FLAGT5(NAI,NAE) \ Branch iff Temp flag 5 \
           (TC/BD#,BI/0);
MICRO IF-NOT-FLAGT5(NAI,NAE)
          (TC/BD#,BI/1);
MICRO IF-FLAGT6(NAI,NAE) \ Branch iff Temp flag 6 \
           (TC/BE#,BI/0);
MICRO IF-NOT-FLAGT6(NAI,NAE)
          (TC/BE#,BI/1);
MICRO IF-FLAGT7(NAI,NAE) \ Branch iff Temp flag 7 \
           (TC/BF#,BI/0);
MICRO IF-NOT-FLAGT7(NAI,NAE)
          (TC/BF#,BI/1);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO IF-FLAGP0(NAI,NAE) \ Branch iff Perm flag 0 \
          (TC/A8#,BI/0);
MICRO IF-NOT-FLAGP0(NAI,NAE)
          (TC/A8#,BI/1);
MICRO IF-FLAGP1(NAI,NAE) \ Branch iff Perm flag 1 \
          (TC/A9#,BI/0);
MICRO IF-NOT-FLAGP1(NAI,NAE)
          (TC/A9#,BI/1);
MICRO IF-FLAGP2(NAI,NAE) \ Branch iff Perm flag 2 \
          (TC/AA#,BI/0);
MICRO IF-NOT-FLAGP2(NAI,NAE)
          (TC/AA#,BI/1);
MICRO IF-FLAGP3(NAI,NAE) \ Branch iff Perm flag 3 \
          (TC/AB#,BI/0);
MICRO IF-NOT-FLAGP3(NAI,NAE)
          (TC/AB#,BI/1);
MICRO IF-FLAGP4(NAI,NAE) \ Branch iff Perm flag 4 \
          (TC/AC#,BI/0);
MICRO IF-NOT-FLAGP4(NAI,NAE)
          (TC/AC#,BI/1);
MICRO IF-FLAGP5(NAI,NAE) \ Branch iff Perm flag 5 \
          (TC/AD#,BI/0);
MICRO IF-NOT-FLAGP5(NAI,NAE)
          (TC/AD#,BI/1);
MICRO IF-FLAGP6(NAI,NAE) \ Branch iff Perm flag 6 \
          (TC/AE#,BI/0);
MICRO IF-NOT-FLAGP6(NAI,NAE)
          (TC/AE#,BI/1);
MICRO IF-FLAGP7(NAI,NAE) \ Branch iff Perm flag 7 \
          (TC/AF#,BI/0);
MICRO IF-NOT-FLAGP7(NAI,NAE)
          (TC/AF#,BI/1);
MICRO IF-I-BUF(NAI,NAE)    \ same as "IF-I-FB" \
          (TC/8E#,BI/0);
MICRO IF-NOT-I-BUF(NAI,NAE)
          (TC/8E#,BI/1);
MICRO IF-I-CRY(NAI,NAE)    \ Branch iff Carry indicator \
          (TC/89#,BI/0);
MICRO IF-NOT-I-CRY(NAI,NAE)
          (TC/89#,BI/1);
MICRO IF-I-FB(NAI,NAE)     \ Branch iff Frame-bound indicator \
          (TC/8E#,BI/0);
MICRO IF-NOT-I-FB(NAI,NAE)
          (TC/8E#,BI/1);
MICRO IF-I-LE(NAI,NAE)     \ Branch iff Sign or Double-zero indicator \
          (TC/84#,BI/0);
MICRO IF-NOT-I-LE(NAI,NAE)
          (TC/84#,BI/1);
MICRO IF-I-LE1(NAI,NAE)    \ Branch iff Sign or Zero indicator \
          (TC/86#,BI/0);
MICRO IF-NOT-I-LE1(NAI,NAE)
          (TC/86#,BI/1);
MICRO IF-I-ODD(NAI,NAE)    \ Branch iff Odd indicator \
          (TC/8D#,BI/0);
MICRO IF-NOT-I-ODD(NAI,NAE)
          (TC/8D#,BI/1);
MICRO IF-I-OVF(NAI,NAE)    \ Branch iff Overflow indicator \
          (TC/88#,BI/0);
MICRO IF-NOT-I-OVF(NAI,NAE)
          (TC/88#,BI/1);
MICRO IF-RINT(NAI,NAE)     \ Branch iff Resume Interrupt occurred \
          (TC/9B#,BI/0) \ after CUP's most recent MEGABUS cycle;
MICRO IF-NOT-RINT(NAI,NAE)
          (TC/9B#,BI/1);
MICRO IF-I-SCR(NAI,NAE)    \ Branch iff Stop-code indicator \
          (TC/8F#,BI/0);
MICRO IF-NOT-I-SCR(NAI,NAE)
          (TC/8F#,BI/1);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO IF-I-SGN(NAI,NAE)   \ Branch iff Sign indicator \
          (TC/8A#,BI/0);
MICRO IF-NOT-I-SGN(NAI,NAE)
          (TC/8A#,BI/1);
MICRO IF-I-ZRO(NAI,NAE)   \ Branch iff Zero indicator \
          (TC/8B#,BI/0);
MICRO IF-NOT-I-ZRO(NAI,NAE)
          (TC/8B#,BI/1);
MICRO IF-I-Z'Z(NAI,NAE)   \ Branch iff Double-zero indicator \
          (TC/8C#,BI/0);
MICRO IF-NOT-I-Z'Z(NAI,NAE)
          (TC/8C#,BI/1);
MICRO IF-NAK(NAI,NAE)     \ Branch iff MEGABUS cycle was refused \
          (TC/B6#,BI/0)        \ Requires prior "STALL:ACK";
MICRO IF-NOT-NAK(NAI,NAE)      \ Better to use IF-ACK \
          (TC/B6#,BI/1)        \ Requires prior "STALL:ACK";
MICRO IF-OPT(NAI,NAE)     \ Branch iff event signalled by \
          (TC/A0#,BI/1)     \   Option board;
MICRO IF-NOT-OPT(NAI,NAE)
          (TC/A0#,BI/0);
MICRO IF-P2PGX(NAI,NAE) \ Branch iff either FLAGP2 OR PAGE-X \
          (TC/B4#,BI/0)     \ indicators are set;
MICRO IF-NOT-P2PGX(NAI,NAE)
          (TC/B4#,BI/1);
MICRO IF-P6SYNC(NAI,NAE) \ Branch iff FLAGP6 resynchronizer flop \
          (TC/B2#,BI/0)     \ is set (for test use only);
MICRO IF-NOT-P6SYNC(NAI,NAE)
          (TC/B2#,BI/1);
MICRO IF-PAGE'X(NAI,NAE) \ Branch iff ADRP incremented through \
          (TC/B3#,BI/0)     \  a page boundary);
MICRO IF-NOT-PAGE'X(NAI,NAE)
          (TC/B3#,BI/1);
MICRO IF-PWRVLD(NAI,NAE) \ Branch iff AC input power source OK \
          (TC/B7#,BI/0)     \   For will-writing);
MICRO IF-NOT-PWRVLD(NAI,NAE)
          (TC/B7#,BI/1);
MICRO IF-RAD:F(NAI,NAE)   \ Branch iff RAD = F, then RAD <= RAD + 1 \
          (TC/81#,BI/0);
MICRO IF-NOT-RAD:F(NAI,NAE)
          (TC/81#,BI/1);
MICRO IF-RPSYNC(NAI,NAE) \ Branch iff RUPT resynchronizer flop \
          (TC/98#,BI/0)     \ is set (for test use only);
MICRO IF-NOT-RPSYNC(NAI,NAE)
          (TC/98#,BI/1);
MICRO IF-RUPT'CLR(NAI,NAE)\ Branch iff Interrupt, \
          (TC/A5#,BI/0)     \ then clear Interrupt;
MICRO IF-NOT-RUPT'CLR(NAI,NAE)
          (TC/A5#,BI/1);
MICRO IF-RUPT(NAI,NAE)    \ Branch iff Interrupt\
          (TC/A6#,BI/0);
MICRO IF-NOT-RUPT(NAI,NAE)
          (TC/A6#,BI/1);
MICRO IF-SEMA4(NAI,NAE)   \ Branch iff a write-unlock bus cycle has \
          (TC/99#,BI/0)     \ occurred since my last read-unlock ;
MICRO IF-NOT-SEMA4(NAI,NAE)
          (TC/99#,BI/1);
MICRO IF-TIMER(NAI,NAE)   \ Branch iff TIMER expired the clear \
          (TC/95#,BI/0)     \ indicator;
MICRO IF-NOT-TIMER(NAI,NAE)
          (TC/95#,BI/1);
MICRO IF-UNLOCK(NAI,NAE) \ Branch iff unlock bus cycle has occurred \
          (TC/99#,BI/0) \  since my last cycle that was NAK'd;
MICRO IF-NOT-UNLOCK(NAI,NAE)
          (TC/99#,BI/1);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO IF-YELO(NAI,NAE)     \ Branch iff Memory Yellow, then clear \
        (TC/A1#,BI/0)          \ Yellow. Requires prior "STALL:BUSY";
MICRO IF-NOT-YELO(NAI,NAE)
        (TC/A1#,BI/1);
MICRO IF-Z08(NAI,NAE)      \ Branch iff Z-bus(08)\
         (TC/87#,BI/0);
MICRO IF-NOT-Z08(NAI,NAE)
        (TC/87#,BI/1);
MICRO IF-Z16(NAI,NAE)      \ Branch iff Z-bus(16) \
         (TC/82#,BI/0);
MICRO IF-NOT-Z16(NAI,NAE)
        (TC/82#,BI/1);
MICRO IF-Z24(NAI,NAE)      \ Branch iff Z-bus(24) \
         (TC/83#,BI/0);
MICRO IF-NOT-Z24(NAI,NAE)
        (TC/83#,BI/1);

MICRO INC-CYCLE            \ Increment Cycle counter in mid-step \
         (FL/03#,CY/0);
MICRO INC-RAD              \ RAD <= RAD+1 \
         (FL/13#);

MICRO IND-AR               \ Arithmetic indicators saved: \
         (FL/15#)          \ IND0 <= ALU overflow
                             IND1 <= ALU carry out
                             IND2 <= ALU sign bit (ALUF08)
                             IND3 <= ALU zero-ness (all 24 bits)
                             IND4 <= ALU zero-ness & previous I3 value
                             IND5 <= Z-bus bit 31;
MICRO IND-BB'SC(FL/16#)    \ same as "IND-FB'SC";
MICRO IND-FB'SC            \ Frame-bound & Stop-code indicators saved:\
         (FL/16#)          \ IND6 <= D-bus(08-22) <NE> Z-bus(08-22)
                             IND7 <= SCRAM output <XOR> FLAGT6;
MICRO INT                  \ RAA <= P-bus(4-7) in mid-step, \
         (FL/10#,MPP/X)        \ clear Temp flags & RAD;

MICRO LD-CNFG              \ Load Configuration Register \
      (MGF/13,MGS/0)       \ CMDPAR <= BUSZ(28) \
                           \ FCODE1 <= BUSZ(30) \;
                           \ ACTREN <= BUSZ(31) \;
MICRO LD-CYCLE(AA)         \ CYCLE <= A-address field of SP \
         (FL/02#,CY/0);
MICRO LD-OP                \ OPREG <= P-bus(0-7) in mid-step \
         (OP/1);
MICRO LD-RAB'C             \ RAB,RAC <= P-bus(0-7) in mid-step \
         (FL/11#,MPP/X);
MICRO LD-RAB'C'IND         \ RAB,RAC <= P-bus(0-7) in mid-step & \
         (FL/17#,MPP/X)\        save arithmetic indicators;
MICRO LD-RAD(SD)           \ RAD <= Z-bus(rotated by SD) \
         (FL/12#);
MICRO LD-SCRAM             \ SCRAM(D-bus(16-23)) <= FLAGT6 \
         (FL/14#,CKM/3,DS/0)   \ ILLEGAL when D-bus <= Z-bus;
MICRO LD-SYND              \ SYNDROME <= Status sample \
         (TCX/31#);

MICRO MCA:Z                \ Strobe MCA functions selected \
         (TCX/05#)         \       by Z-bus bits;

MICRO NO-FAULT             \ Suppress errors from \
         (TCX/35#)         \       Z:INRA, Z:INRB, D:PROC;

MICRO OUTR:D               \ Load OUTR <= D-bus \
         (LDO/8);

MICRO PARITY-ERROR         \ Force errors in all thirds of F/W word \
         (PE/1);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO PREFETCH(MCA,MCA)   \ Start new procedure fetch \
               (MGF/18#,MGS/1,MSA/F,MSB/F);

MICRO PTAKE(MCA,MCA)      \ Consume procedure byte \
               (MGF/1F#,MGS/1,MGS0/1,MGS1/1,MSA/F,MSB/F);

MICRO PUSH(PSA,PSM)       \ Push "RETURN CSA" onto stack \
               (PS/1,DB/0);

MICRO RD-MEM-DBLW(FLM,MCA) \ Wait for quiet interface \
        (MRQ/9F#,MGF/19#)  \ Read 2 memory words to INRA&B \
                           \ Needs "ADRS:A" or "ADRS:B";
MICRO RD-MEM-WORD(FLM,MCA) \ Wait for quiet interface \
        (MRQ/8F#,MGF/18#)  \ Read 1 memory word to INRA/B \
                           \ Argument "LOCK" or "UNLOCK" optional \
                           \ needs "ADRS:A" or "ADRS:B";
MICRO RD-NON-MEM           \ Wait for quiet interface \
        (MRQ/8F#,FLM/D8#)  \ Read 1 non-memory word to INRA/B\
                           \ Needs "ADRS:A" or "ADRS:B" to \
                           \ specify channel, function-code;

MICRO REG:Z(RM)           \ ARAM(RM) <= Z-bus\
               (RW/1);

MICRO RETURN(NAB)         \ Pop subroutine stack\
               (BR/7,MK/F);
MICRO RETURN'(MK,NAB)     \ Pop subroutine stack (Alternate return)\
               (BR/7);

MICRO R-I-P    (MRP/0)    \ Request In Progress warning:
                            Don't try to change ADRS selection
                            nor ADR- nor OUTR (if in use) till STALL;

MICRO R:A'S:Q(AA)         \ ALUR <= SP(A),  ALUS <= Q \
               (AS/0);
MICRO R:A'S:B(AA,AB)      \ ALUR <= SP(A),  ALUS <= SP(B) \
               (AS/2);
MICRO R:0'S:Q             \ ALUR <= 000000, ALUS <= Q \
               (AS/4);
MICRO R:0'S:B(AB)         \ ALUR <= 000000, ALUS <= SP(B) \
               (AS/6);
MICRO R:0'S:A(AA)         \ ALUR <= 000000, ALUS <= SP(A) \
               (AS/8);
MICRO R:D'S:A(AA)         \ ALUR <= D-bus,  ALUS <= SP(A) \
               (AS/A);
MICRO R:D'S:Q             \ ALUR <= D-bus,  ALUS <= Q \
               (AS/C);
MICRO R:D'S:0             \ ALUR <= D-bus,  ALUS <= 000000 \
               (AS/E);

MICRO SHRG:Z(SD)          \ SHRG <= Z-bus(rotated by SD) \
               (SR/1);

MICRO STALL:ACK           \ Stall until/unless Ack'd or Nak'd \
               (MGF/1);
MICRO STALL:BUSY          \ Stall until/unless Interface quiet \
               (MGF/3);
MICRO STALL:INRA          \ Stall until/unless INRA full \
               (MGF/2);

MICRO SYSCLR              \ Generate a system-wide clear \
          (MGF/13,MGS/1);
MICRO TIMER:Z (FL/05#)    \ Timer <= Z-bus(08-31);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

```
MICRO WR-LM-CNFG             \ Wait for quiet interface \
        (MRQ/CF#,FLM/9C)     \ Via the local interface, \
                             \ write OUTR into Local Memory Conf Reg \
                             \ Needs "ADRS:A" or "ADRS:B" \
MICRO WR-MEM-BYTE            \ Wait for quiet interface \
               (MRQ/AF#      \ Write half of OUTR to memory \
               ,MLO/0)       \ Needs "ADRS:A" or "ADRS:B";
MICRO WR-MEM-WORD(FLM)       \ Wait for quiet interface \
               (MRQ/BF#      \ Write OUTR to memory \
               ,MLO/0)       \ Argument "LOCK" or "UNLOCK" optional \
                             \ needs "ADRS:A" or "ADRS:B";
MICRO WR-NON-MEM(MOP)        \ Wait for quiet interface \
               (MNM/56F#     \ Send OUTR to channel addressed by ADRS \
               ,MLO/0)       \ Argument = RUPT or
                                          REPLY or
                                          TEST-A or
                                          TEST-B or
                                          TEST-P \
                             \ Needs "ADRS:A" or "ADRS:B" \
                             \ (or "ADRS:P" with TEST- argument);

MICRO Y:F'Q:F                \ ALUY <= ALUF, Q <= ALUF \
               (AD/0);
MICRO Y:F                    \ALUY <= ALUF \
               (AD/2);
MICRO Y:A'B:F(AA,AB)         \ ALUY <= SP(A),SP(B) <= ALUF \
               (AD/4);
MICRO Y:F'B:F(AB)            \ ALUY <= ALUF, SP(B) <= ALUF \
               (AD/6);
MICRO Y:F'BQ:FQSR(AB)        \ ALUY <= ALUF, SP(B),Q <=SR= FLAGT1,ALUF,Q \
               (AD/8);
MICRO Y:F'B:FSR(AB)          \ ALUY <= ALUF, SP(B) <=SR= FLAGT1,ALUF \
               (AD/A);
MICRO Y:F'BQ:FQSL(AB)        \ ALUY <= ALUF, SP(B),Q <=SL= ALUF,Q,FLAGT1 \
               (AD/C);
MICRO Y:F'B:FSL(AB)          \ ALUY <= ALUF, SP(B) <=SL= ALUF,Q(08) \
               (AD/E);

MICRO Z:INRA                 \ Z-bus <= Unspec'd(08-15),INRA(16-31) \
               (ZB/2);
MICRO Z:INRB                 \ Z-bus <= Unspec'd(08-15),INRB(16-31) \
               (ZB/3);
MICRO Z:MCA                  \ Z-bus <= Microcode Analyzer or \
               (ZB/1)        \            serial number;
MICRO Z:OPT                  \ Z-bus <= Option board or FDF or \
               (ZB/1)        \            serial number;
MICRO Z:Y                    \ Z-bus <= ALUY(08-31) \
               (ZB/0);
MICRO Z:Y-INRA               \ Z-bus <= ALUY(08-15),INRA(16-31) \
               (ZB/2);
MICRO Z:Y-INRB               \ Z-bus <= ALUY(08-15),INRB(16-31) \
               (ZB/3);
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

owner Dick Lemay
name memoblnk.prn

at (0 0 cc)
window (0 0 0)
imagesize (0 0)
imagespace (0 0 0 0)
jobheader on
copies 1
pagecollation on
paper letter
jamresistance on
language impress

System Version 3.3 Rev B IP/II, Serial #87:9:79
Page images processed: 1
Pages printed: 1

Paper size (width, height):
2560, 3328
Document length:
584 bytes

Special Processor Engineering

FLOW CHARTING CONVENTIONS

Firmware flowcharts are meant to record a design and to instruct others who may require the knowledge so that they may either advance the art or to maintain the product. The conventions listed below are mainly aimed at consistency so as to provide a means of communicating technical matters with a minimum of ambiguity.

3.1 THE SYMBOLOGY

A rectangle represents a firmware step. A firmware step is sometimes called a firmware "box". The micros (to be) coded into the firmware box are written in the rectangle. Sometimes the actual micros are not recorded but a higher level syntax is used instead. Section 4 is a specification of the higher level language for the 16-bit Custom Processor.

An example of a firmware step in flowchart form is shown in Figure 3-1.

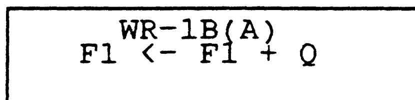


FIGURE 3-1
A FIRST EXAMPLE

3.2 ADDRESSES

The addresses of the firmware steps being documented are annotated in a convenient place but always outside of the rectangle. Because firmware is best debugged by flowchart, it is wise to record the absolute address of each firmware step. It is usually handy to show the symbolic addresses as well, particularly if symbolic addresses are not used indiscriminately (denote the beginning of routines and other major branch destinations only). An example is shown in Figure 3-2.

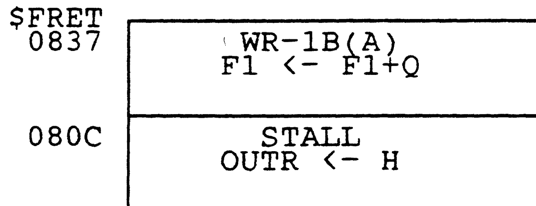


FIGURE 3-2
SIMPLE ADDRESS ANNOTATION

HONEYWELL BULL CONFIDENTIAL AND PROPRIETARY

3.3 FLOW

Firmware steps perform tests in order to control microprogram flow. The simplest test mechanism which the hardware provides allows a two-way choice and the most complex allows a 257-way choice.

3.3.1 TWO-WAY CHOICE

The test condition is written inside the bottom of the rectangle and the two choices are drawn as rectangles below, one at the left and one at the right. The test condition is distinguished from other micro because it is written with a question mark suffix. The right rectangle contains those micros to be executed if the test condition is TRUE and the left rectangle contains those micros to be executed if the test condition is false. In the example shown in Figure 3-3, the zero indicator is tested; if the zero indicator is ON, F4 will receive F1 plus seven whereas if the zero indicator is OFF, a stall is performed and OUTR receives the content of the H register.

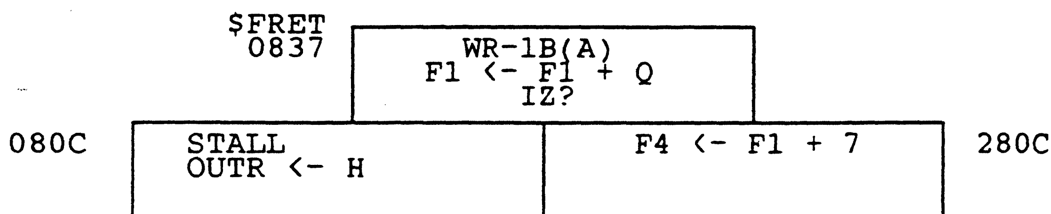


FIGURE 3-3
SIMPLE SEQUENCE CONTROL

Permitted alternatives to the representation shown in Figure 3-3 are shown in Figures 3-4 and 3-5. Although there is no functional difference, such alternatives are useful where complex flow is being documented; e.g., when multiple columns of firmware are drawn on one page.

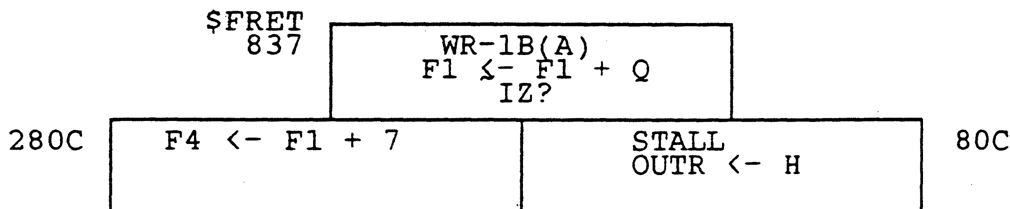


FIGURE 3-4
SIMPLE SEQUENCE CONTROL - ALTERNATIVES

HONEYWELL BULL CONFIDENTIAL AND PROPRIETARY

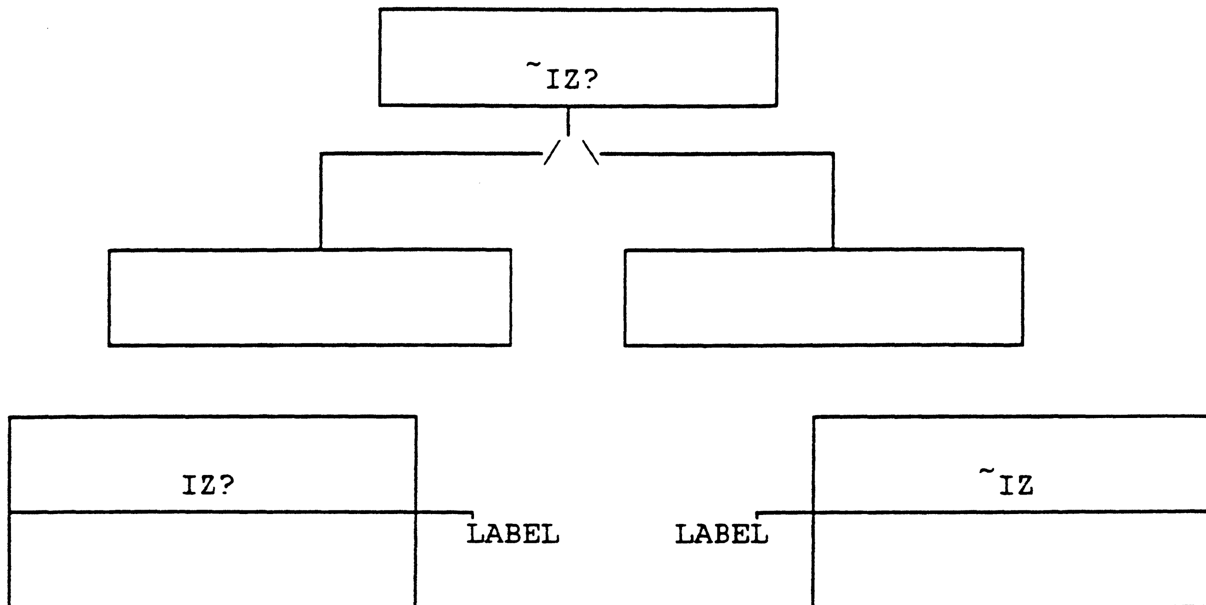


FIGURE 3-5
SIMPLE SEQUENCE CONTROL- OTHER ALTERNATIVES

3.3.2 SPLATTERS

A splatter is a mechanism for determining the value of multiple elements. For instance, a splatter on a four-bit counter has sixteen destinations. The splatter is usually allowed to be conditional and the multiple elements being examined can usually be masked. Figure 3-6 shows a conditional splatter on two FLAGS if the ZERO indicator is false. The test condition (if any) is written inside the bottom of the rectangle. A splatter is visually signified by a hash mark crossing the flowpath line, by the splatter name and parenthesized mask value written adjacent to the hash mark, and by connecting the top line of the rectangles representing the multiple destinations. The leftmost splatter destination is annotated with the base address of the splatter and the other destinations need only be annotated with the varying portion of the destination address. If, as frequently happens, the splatter destinations can not all be drawn nearby, it is helpful to repeat the splatter name and mask value.

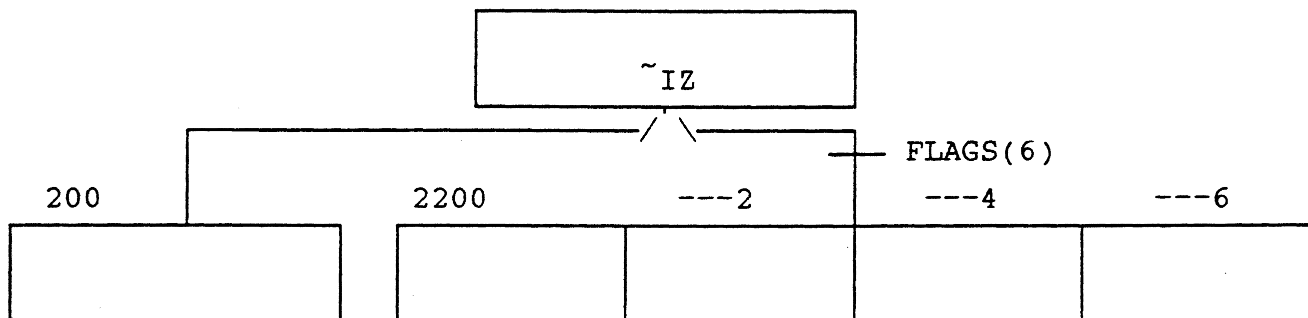


FIGURE 3-6
A CONDITIONAL FOUR-WAY SPLATTER

HONEYWELL BULL CONFIDENTIAL AND PROPRIETARY

THE TRANSFER LANGUAGE
AND
THE TRANSFER LANGUAGE COMPILER

This section describes the Transfer Language which allows microcode to be written at a "higher level" than that permitted by the dictionary of section 2. This section also specifies the compiler which "converts" the high level source statements into the micros of section 2; i.e., the compiler does not directly generate object code.

4.1. SOURCE FILE FORMAT

4.1.1 LINE LENGTH

The source file shall be free-form text consisting of lines of ASCII characters no longer than 82 characters (including line delimiter). The compiler may, but need not, enforce the line-length limit.

4.1.2 WHITE SPACE

The line delimiter, space, horizontal tab character and formfeed character shall be considered "white-space" and are syntactically equivalent. White-space is only necessary where the juxtaposition of two tokens (keywords, identifiers, operators, etc.) would cause the compiler to misinterpret thetwo tokens as a single token of some other type. Example:

GOTOX cannot be interpreted as GOTO X because without the white-space between GOTO and X, the compiler must consider GOTOX as a single token.

4.1.3 VALID/INVALID CHARACTERS

The following characters (expressed in 'C' notation) are never valid ANYWHERE in a source file:

```
'\0' through '\010'
        '\013'
'\016' through '\037'
'\177' through '\377'
```

The following character is valid only inside a literal block:

```
'\\'
```

The following characters are valid only inside of a comment or other type of delimited text (such as the INCLUDE statement's <pathname>):

```
'%' '.'
'?' '`' '{' '}'
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.1.4 COMMENTS

A comment consists of any text not including semicolon or any of the characters mentioned in sections 1.3 and 1.4a beginning with the delimiter /* and ending with the delimiter */. A comment, therefore may span multiple lines.

Comments may not be nested.

A comment is treated syntactically as white-space.

4.2 LEXOGRAPHY

4.2.1 CASE

Upper-case and lower-case letters in reserved words, identifiers and constants are equivalent and will be converted to upper-case in any output file.

4.2.2 SOURCE FILE LENGTH

A source file consists of zero or more comments and/or statements.

4.2.3 STATEMENT TERMINATOR

A statement is terminated by the semicolon character, ';', which may not be used for any other purpose. Especially, the semicolon character may not be used inside a comment, due to problems with the RTL assembler.

4.2.4 LITERAL TEXT BLOCK

Any text beginning with the double-quote character, '"', up to the next double-quote character is considered literal text and must conform exactly to the rules describing the syntax of RTL assembly language. This rule does not apply inside of a comment.

4.2.5 LITERAL TEXT BLOCK RESTRICTIONS

Literal text may be imbedded in a statement, or form a block which could be outside of any other block.

4.2.6 LITERAL TEXT BLOCK "COMPILATION"

Literal text is copied as-is (except that the delimiting quotes are removed) to the compiler's RTL assembly language output file. Therefore, a literal block is opaque to the compiler. This means that any labels defined within the literal block are unknown to the compiler and that the compiler will be unable to detect duplicate usage of any firmware address assigned within the literal block. The compiler will also be unable to detect a conflict between any micro-ops used within the literal block and any micro-ops which the compiler generates in the process mf applying a production rule.

4.2.7 RESERVED WORDS

The list of reserved words is shown in Table 4-1. These reserved words are a part of the language and may not be used for identifiers or block names.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE 4-1
RESERVED WORDS

ACK ADDR	BEGIN BUSY	CALL CASE CLK CMND CONST		END EQU ELSE ENDSW	FALSE
GOTO	HEX HOF	IF INCLUDE			LDSYND
	NOFAULT	PREFETCH PREFETCHN PRESERVES PTAKE PTAKEN PUSH		RD2B RD2BN RD4B RD4BN RDIO RDLK RDULK REPLY RETURN RUPT	SAVES SELECT SKIP STALL SWITCH
TEST TRUE			WT2B WT2BN WTB WTLK WTULK		

4.3. SOURCE FILE ORGANIZATION

4.3.1 STATEMENT TYPES

The source file consists of four types of statements, any of which may be optionally omitted:

- o Pre-processor directives
- o Definitions
- o Block-defining statements
- o Procedure

4.3.2 COMMENTARY

Commentary and literal text may appear anywhere within the source file, subject only to the lexographic restrictions indicated in section two.

4.3.3 PRE-PROCESSOR DIRECTIVES

Pre-processor directives must each be contained on a single line in the source file and must begin in column 1 of the line. The pre-processor understands two directive; INCLUDE and SKIP.

4.3.3.1 INCLUDE '<pathname>';

Open the file specified by <pathname> and insert the text of that file in place of the pre-processor directive as if it were part of the source file. Note: INCLUDEd files may not contain the INCLUDE directive.

4.3.3.2 SKIP <count>;

Where <count> is defined to be an integer or the keyword HOF. This statement is copied directly to the output file, causing <count> lines (or head-of-form) to be skipped when assembled under RTL.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.3.4 LOCAL DEFINITIONS

Four definition statements are provided; EQU, CONST, EXTERN and PUBLIC.

If a definition statement is within a block (see 4.3.5), it is a "local" definition. Local definitions take effect at the place within the block where the definition appears and persist until the end of the block. A definition which is not contained within a block is a "global" definition. Global definitions take effect at the place in the source file where the definition appears and persist until the end of the source file is encountered. (Actually, the situation is more complex -- "end of the source file" means the end of the top-level source file, not the end of some included file which happened to contain a global definition.)

An <identifier> which is defined in a global definition statement may not be re-defined elsewhere. A local definition can, however, override a global definition within the scope of the block in which it appears.

4.3.4.1 <identifier> EQU <predefined identifier>;

This definition causes the compiler to treat <identifier> as a synonym for the specified pre-defined identifier (e.g.: ADRA) or a reserved word (e.g.: BEGIN).

4.3.4.2 <identifier> CONST <integer constant>;

This definition causes the compiler to treat <identifier> as a synonym for some integer constant, allowing symbolic names for "magic numbers".

4.3.4.3 <label> EXTERN;

This definition specifies that the <label> is defined in some other source module. It compiles to the RTL statement "XLOC".

4.3.4.4 <label> PUBLIC;

This definition serves two purposes. First, it compiles to the RTL "XDEF" statement (which allows the label to be referenced by "EXTERN" statements in other source files. Additionally, the <label> is made globally known within a structured file. (Normally, labels within blocks in a structured source file -- see section 3.5 -- are only known within the scope of the block in which they appear. The PUBLIC definition statement overrides the scope limitation normally imposed.) The <label> must be some label defined within the block. There may be any number of PUBLIC statements within a block.

4.3.5 BLOCK DEFINITIONS

Block-defining statements impose scope rules on definitions and labels and indicate whether or not the contents of hardware registers are destroyed by the code contained within or called from the block. There are four block-defining statements: BEGIN, END, PRESERVES and SAVES. The BEGIN statement defines the start of a block; the END statement marks its end. The PRESERVES statement asks the compiler to verify that the specified register(s) are never the target of an assignment or increment operation and to issue a warning message if they are. The SAVES statement asserts (the compiler need not verify the assertion) that the code within this block restores the original contents of specified register(s). This statement allows the compiler to verify register preservation for subroutines which save and restore registers which the subroutine uses as working variables and therefore cannot declare as preserved.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

If any block-defining statement appears within a source file, the file is said to be "structured". If no block-defining statements are specified, the file is said to be "unstructured". No procedure statements may appear outside the scope of a block in a structured file. Blocks are not "nested"; they correspond more closely to 'C' functions than to Pascal procedures. If a block does not contain a PRESERVES or a SAVES statement, it will be assumed that no registers are preserved.

4.3.5.1 BEGIN

The BEGIN statement has the form:

[<block name>] BEGIN;

where <block name> is an identifier which the programmer has chosen to name the block. It is optional. If it is used, it must be unique in the source file.

4.3.5.2 END

The END statement has the form:

[<block name>] END;

where the <block name> must, if specified, match the <block name> of the most recently-defined unmatched BEGIN. The compiler shall verify that all BEGINs have corresponding ENDs.

4.3.5.3 PRESERVES

The PRESERVES statement has the form:

PRESERVES <register_name> [, <register_name>]... ;

where <register__name> may be any of the hardware register names or any user-defined synonym thereof.

4.3.5.4 SAVES

The SAVES statement has the form:

SAVES <register_name> [, <register_name>]... ;

where <register__name> may be any of the hardware register names or any user-defined synonym thereof.

4.3.6 PROCEDURE

All statements other than pre-processor directives, block-defining statements and definitions are considered procedure.

4.3.6.1 OPERATION CLAUSE

Procedure statements consist of one or more operation clauses. Operation clauses are separated by commas, since the comma operator is defined (see section 4.4.6.14) as the simultaneity operator.

Operation clauses may appear within a statement in any order.

An operation clause may be any of the following:

<assignment>
<control>
<next address specifier>

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.3.6.2 ASSIGNMENT OPERATION CLAUSE

An <assignment> clause's syntax is:

{ <identifier> | (<identifier list>) } = <expression>

An <identifier list> is defined as:

<identifier> [, <identifier>]...

4.3.6.3 CONTROL OPERATION CLAUSE

A <control> clause includes such things as loading syndrome, supressing error detection and bus operations. See paragraph 4.5 for details.

4.3.6.4 NEXT ADDRESS SPECIFIER OPERATION CLAUSE

A <next address specifier> clause may be:

<goto>
<splatter>
<conditional>
<call>
<return>

See paragraph 4.6 for details.

The syntax of the next address specifier is:

(<integer constant>) or

(.A)

For those address specifiers which are integer constants, the compiler shall report multiple use of the same firmware address as an error.

A procedure statement must be prefixed by an address specifier. If a procedure statement is labeled, the address specifier must immediately follow the label.

4.3.6.5 DEFAULT NEXT ADDRESS

If a procedure statement does not contain a <next address specifier> clause, the compiler shall assume a <goto> clause which specifies as its target the address of the next statement in the source file. It is an error for the last statement of a block (or the file, if it is unstructured) not to contain a <next address specifier> clause.

4.3.6.6 PROCEDURE LABELS

A procedure statement may be optionally prefixed by a label. A label is any valid <identifier> prefixed by a dollar-sign, '$'. See paragraph 4.4.2 for the definition of an <identifier>. Some labels may be formed automatically by the compiler. The compiler need not detect that the program contains a label that duplicates an automatically-generated one. It will be up to the user to ensure that such duplication does not occur, provided that the compiler's method for generating labels is documented.

A label may be specified as the target of a <next addressspecifier> operation, the operand of an EXTERN or a PUBLIC statement or as the operand of a PUSH or ADDR operator.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

Any identifier which is used in a procedure statement clause must have already been defined or else be a reserved word or pre-defined identifier. Labels, however, need not be defined before they are referenced.

4.4. EXPRESSIONS

An expression consists of identifiers, reserved words, constants and operators.

4.4.1 IDENTIFIER

An identifier may consist of from one to sixteen characters.

An identifier must begin with a letter.

An identifier may contain (after its initial letter) any combination of characters from the set

'A' through 'Z'
'a' through 'z'
'0' through '9'
hyphen

The identifiers shown in Table 4-2 have been pre-defined and may not be used as either block names or user-defined identifiers.

TABLE 4-2
IDENTIFIERS

ABCD	ADRA	ADRB	ADRPH	ADRPL	ADRX	CYCLE
D	DECODE	F	F0	F1	F2	F3
F4	F5	F6	F7	F8	F9	F10
F11	F12	F13	F14	F15	FLAGS	FQ
H	I	IC	IFB	ILE	ILE1	IO
IS	ISCR	IV	IZ	IZZ	INRA	INRB
INRX	MCA	OP	OPT	P0	P1	P2
P3	P4	P5	P6	P7	PB	PCTR
Q	R	R0	RAR	RAMAD	RBR	RCR
RDR	SCRAM	SYND	T0	T1	T2	T3
T4	T5	T6	T7	Y	Z	ZSH

4.4.2 INTEGER CONSTANT

An <integer constant> may be either:

- A decimal constant, consisting of one or more decimal digits.
- A hexadecimal constant, consisting of one or more characters from the set of:

0 1 2 3 4 5 6 7 8 9 A B C D E F
a b c d e f

followed by the sharp-sign, '#', which indicates the radix.

4.4.3 EVALUATION RANGE

Expressions must evaluate within the range from 00000000# through 0000FFFF# or the range 00FF0000# through 00FFFFFF#. The compiler shall detect out-of-range expressions as an error.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.4.4 OPERATORS

The following operators have been defined, in order of precedence:

```
(unary) -   ~          right-to-left evaluation
        ++  --         right-to-left evaluation
        [ ]            right-to-left evaluation
        ADDR() SELECT() right-to-left evaluation
(binop) +   -          left-to-right evaluation
        << >> <@ @>    left-to-right evaluation
        &              left-to-right evaluation
                       left-to-right evaluation
        |              left-to-right evaluation
                       right-to-left evaluation
        =              right-to-left evaluation
        ,              right-to-left evaluation
```

4.4.4.1 PARENTHESIS

Parenthesis may be used to override operator precedence, subject to the restrictions imposed by the hardware target. Parenthesis also surround the argument list of the ADDR, PUSH and SELECT functions. Expressions [if any are allowed] which are arguments of these functions are evaluated in right-to-left order before the function itself.

4.4.4.2 UNARY MINUS

The unary minus operation performs arithmetic negation. Example:

```
-1234
```

4.4.4.3 UNARY TILDE

The unary tilde, '~', operator performs bit-wise negation; i.e.: the operand is exclusive-ORed with all ones. Example:

```
~mask14
```

The unary tilde operator also performs logical negation; i.e.: TRUE becomes FALSE and vice-versa. Example:

```
~I-ZRO
```

4.4.4.4 UNARY ++ and --

The unary operators "++" and "--" represent increment and decrement, respectively. These operators prefix the sub-expression which is being incremented or decremented. Examples:

```
++RDR  /* Increment RDR  */
--F1   /* The same as F1 = F1 - 1  */
```

4.4.4.5 UNARY BRACKET SET

The bracket set ([ ]) is used to represent "as addressed by". Example:

```
F2 = R[RAR];    /* Set F2 to the ARAM value at
                   the location specified by the
                   RAR register. */
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.4.4.6 UNARY ADDR( )

The ADDR function evaluates to the firmware address of its operand (which must be a label. Example:

```
F3 = ADDR($STEP12)    /* Put the firmware address of
                         $STEP12 into register F3. */
```

4.4.4.7 UNARY SELECT

The SELECT function evaluates to the 24-bit value which is formed by the concatenation of bits 0 through 7 (the high-order byte) of its first operand with bits 8 through 15 of its second operand and bits 16 through 23 of its third operand. Valid operands for this function are a byte-sized literal, H, ZSH and PB. Example:

```
SELECT(H, ZSH, ZSH)    /* Combine the high-order byte of the H
                          register with the middle- and low-
                          order bytes of the shifted Z bus */
```

The variable ZSH specifies the Z-bus shifter output. ZSH is set by an assignment of the form:

```
ZSH = Z <@ nn,
```

or

```
ZSH = Z @> nn,
```

where "nn" specifies the rotate distance (always a multiple of four). See section 4.???? for a description of the rotate operator. The use of ZSH is restricted in that (1) there must be a ZSH assignment clause within any statement that includes a reference to ZSH and (2) the ZSH assignment clause must preceed any use of ZSH.

4.4.4.8 BINARY OPERATORS + and -

The binary operators '+' and '-' stand for addition and subtraction, respectively. Note: because hyphen may appear in an identifier, the binary minus operator must always be surrounded by spaces. Examples:

```
F2 + F1           /* Add F2 to F1 */
F3 - F1 - 1       /* Subtract F1 from F3 and
                     subtract 1 from the result */
```

4.4.4.9 ROTATE BINARY OPERATORS

The left- and right-rotate binary operators (<@ and @>, respectively) perform bit-wise rotates by a specified number of bits. Note: the rotate operators must specify a rotation distance which is a multiple of four. Example:

```
F3 = F3 <@ 16;     /* Rotate F3 left 16 bits and
                      put in F3 */
```

The left- and right-shift operators (<< and >>, respectively) perform bit-wise shifts of a distance ofone bit. The second operand of the shift operator specifies the "fill" bit. The shift operations are ALU-based operations whose first operand may be F (the ALU output), an arithmetic / logical expression FQ (F concatenated with the Q register) or an arithmetic / logical expression concatenated with Q. The fill bit actually comes from flag T0, so specifying a fill bit also causes T0 to be set or cleared (except for the second example, below, where the fill bit specification MUST be Q). If altering T0 is not desired, T0 may be specified as the fill bit, in which case T0 is unchanged.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

Examples:

```
F3 = (F3 + F4) >> 0;    /* Add F3 to F4, shift right 1 bit,
                           zero filling the left-most
                           bit and put result into F3 */

F3 = (F2 & F3) << Q;    /* For left shifts of F, the only
                           valid fill bit specifier is Q. */

F3_Q = (F1 ^ F3)_Q << T0; /* For this shift, the fill bit
                             can be specified, but in this
                             case, we did not want to
                             disturb T0. */

F1_Q = (F1 | F2)_Q >> 1;  /* Right-shift long with 1-bit
                             fill */
```

4.4.4.10 UNDERSCORE BINARY OPERATOR

The underscore (__) is the concatenation operator. It is used in the case where a quantity must be constructed by concatenating two other quantites in ALU operations. A more powerful concatenation operator is the SELECT function; see section ????. Example:

```
F5_Q = FQ <<1;   /* Shift F concatenated with Q left one bit
                    (1-filled) and put result in F5 and Q. The
                    F value must have been specified by some
                    other clause within the statement. */
```

The concatenation operator may also be used for such constructs as:

```
RBR_RCR,
RAR_RBR_RCR_RDR (although ABCD is preferred),
Y_INRA
```

and

```
Y_INRB
```

4.4.4.11 BOOLEAN BINARY OPERATORS

Bit-wise (Boolean) AND, Exclusive-OR, and OR operations are defined by means of the operators '&', '^' and '|', respectively. Other operations, such as AND-NOT and Exclusive NOR may be created by using the AND or XOR operators along with the unary negation (~) operator applied to the second operand. Examples:

```
Z = F3 ^ F2;   /* exclusive OR F3 and F2 and put result
                  on Z bus */

F2 = F2 & ~F1; /* AND F2 and NOT F1 and put result in
                  F2 */
```

4.4.4.12 EQUAL BINARY OPERATOR

The assignment operator is the equal-sign. It is included in the list of operators so that its precedence can be represented.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.4.4.13 COMMA BINARY OPERATOR

The comma operator specifies simultaneous operations performed within a single step. Examples:

```
Z = F5, F1 = F5 + F1;    /* Z gets F5 while F1 gets F5 + F1 */
(Z, F1) = F2 + 003F#;    /* both Z and F1 get F2 + 003F# */
(I, Z) = ++F2;           /* The arithmetic indicators, the Z
                            bus and F2 all get F2 + 1 */
```

Note: comma also appears within the argument list of some keywords. When it is encountered in an argument list it is not treated as an operator.

4.5. CONTROL CLAUSES

4.5.1 STALLS

The STALL operation is specified by the syntax:

```
STALL(<event>)
```

where <event> can be any one of ACK, BUSY or INRA.

4.5.2 READS

There are seven READ operations, all of which are formed:

```
<read-op>(<addr reg>)
```

where <addr reg> may be either ADRA or ADRB and <read-op> may be:

```
RD2B -- read two bytes of memory
RD2BN -- read two bytes of memory, no-cache
RD4B -- read four bytes of memory
RD4BN -- read four bytes of memory no-cache
RDIO -- read I/O
RDLK -- read and lock
RDULK -- read and unlock
```

4.5.3 WRITES

There are nine WRITE operations:

```
<write-op>(<addr reg>)
```

where <addr reg> is as defined in the read operations, while <write-op> may be:

```
CMND  -- command
REPLY -- reply (SHBC)
RUPT  -- cause interrupt
TEST  -- wrapped write
WT2B  -- write two bytes to memory
WT2BN -- write two bytes to memory, no-cache
WTB  -- write one byte to memory
WTLK  -- write and lock
WTULK -- write and unlock
```

4.5.4 LDSYND

The LDSYND clause causes the syndrome register to be loaded from the current status information.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.5.5 PROCEDURE FETCH

There are four procedural fetch control operations:

```
PTAKE     -- take one byte of procedure
PTAKEN    -- take one byte; no cache
PREFETCH -- load procedure buffer
PREFETCHN -- load procedure buffer; no cache
```

4.5.6 NOFAULT

The NOFAULT clause causes hardware error condition signals (e.g.: data parity) to be ignored during the current step.

4.6. NEXT ADDRESS SPECIFIER CLAUSES

Eleven next address specifier clauses take as their target operand either some label whose scope includes the statement containing the address specifier clause or an integer constant which is a valid firmware address. The compiler shall report as an error any label which is not resolvable or any integer constant not in the range 0 through [TBD]. The practice of using integer constants as the target of next address specifier clauses is strongly discouraged.

4.6.1 GOTO

The GOTO operation specifies an un-conditional jump to another firmware step. Its syntax is:

```
GOTO <target>
```

where <target> was defined in section 6.1.

4.6.2 SPLATTER

The splatter operation specifies a "computed goto" or multi-way jump to one firmware address of a group of addresses. Its syntax is:

```
SWITCH (<selector>),
    CASE <int>: <target>,
    CASE <int>: <target>,
    [CASE <int>: <target>,] ...
ENDSW
```

where <selector> can be P0, RAMAD, FLAGS, OP, I, or DECODE and <int> may be 0 through 15 (in any convenient integer notation) which specifies the low-order digit of the target address. It is an error to have any <int> more than once in a single <splatter operation>.

4.6.3 CALL

The CALL operation specifies a subroutine call. It is a "macro" statement, a combination of a PUSH operation and a GOTO operation. Its syntax is:

```
CALL <target>
```

The CALL statment expands to a PUSH of the address of the next successive source statement and a GOTO to the <target> that is the CALL's operand.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

4.6.4 RETURN

The RETURN operation is the converse of a CALL. It specifies that the top of the subroutine stack is to be popped and used as the destination firmware address. Its syntax is:

RETURN [(<mask>)]

where <mask> can be 0 through 15.

4.6.5 CONDITIONAL

The conditional operation specifies a two-way (if/else) jump. Its syntax is highly complex, as other next address specifier clauses may be parts of the conditional operation. The following forms are permitted:

1. IF (<test condition>) <target> [ELSE <target>]
2. IF (<test condition>) <return operation> [ELSE <target>]
3. IF (<test condition>) <splatter operation> [ELSE <target>]
4. IF (<test condition>) ELSE <target>
5. IF (<test condition>) ELSE <return operation>
6. IF (<test condition>) ELSE <splatter operation>

Note that in forms 1 and 4, the <target> in the ELSE clause does not necessarilly have to be in the ELSE-bank of firmware addresses; so long as the IF target and the ELSE target are in opposite banks, the compiler can invert the test condition to achieve alignment with hardware requirements.

In forms 1, 2 and 3, if the ELSE portion of the conditional operation is omitted, the address of the next source statement is the destination when the tested condition is FALSE. In forms 4, 5 and 6, the address of the next source statement is the destination when the tested condition is TRUE. Forms 4, 5 and 6 should be avoided unless inversion of the test condition makes the condition expression unreadable.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

THE FIRMWARE DEVELOPMENT FACILITY

This section contains a description of the Firmware Development Facility available for use with the 16-Bit Custom Processor. The section is divided into three parts:

- o A general overview
- o The menu
- o A description of each menu item
- o A description of the optional Missing-Stall Catcher

The Firmware Development Facility (FDF) is an equipment which allows checkout of firmware, coded and assembled under RTL. The FDF consists of:

- o A separate five-card cage with an independent power supply
- o A processor board with a Z80 processor
- o A SILO board with a 4096 location SILO
- o A 16384-by-96 location control-store-PROM substitute
- o A terminal/keyboard unit (e.g., 7300)
- o Appropriate interconnecting cables

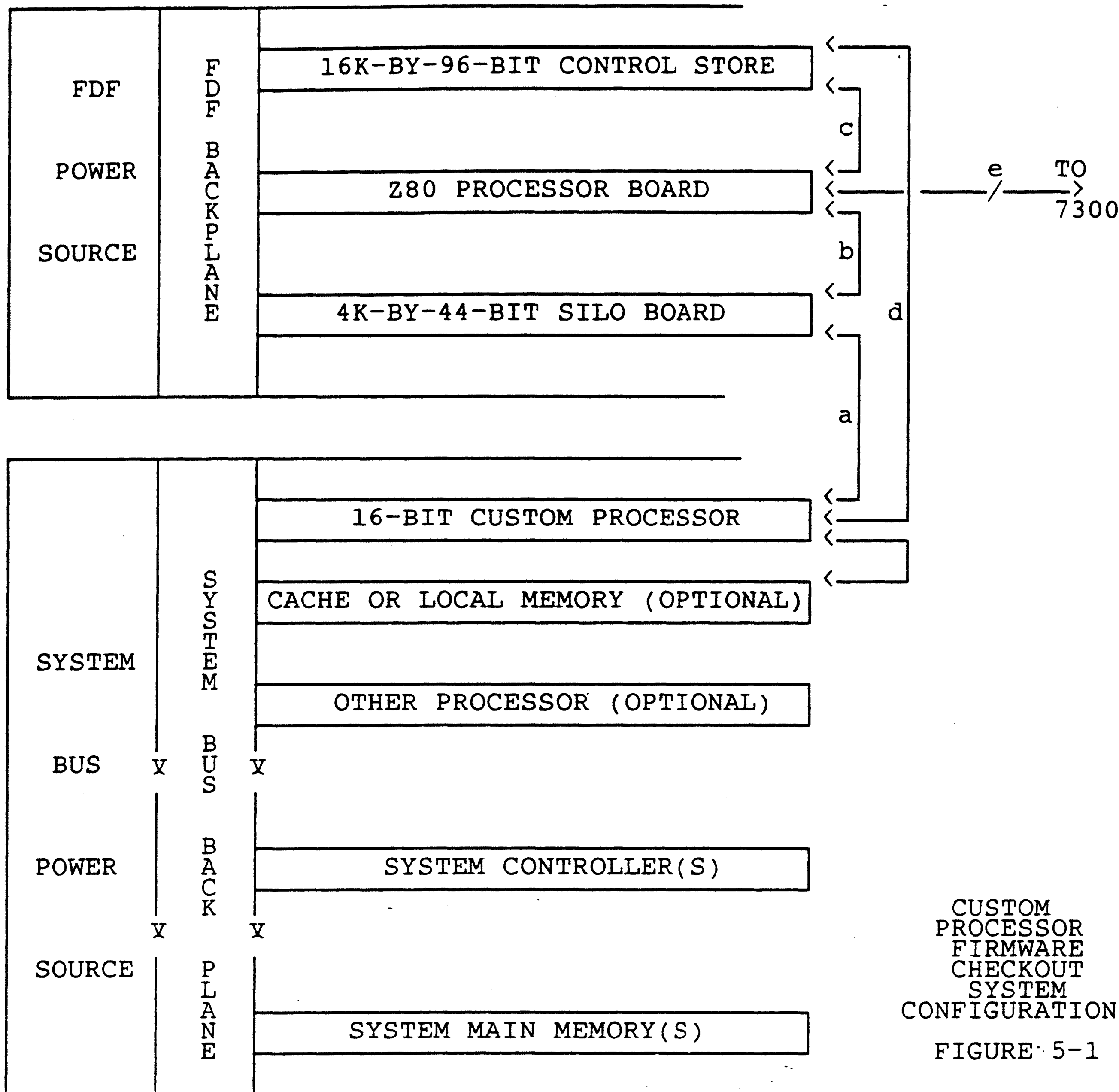
Utilizing the FDF in a development environment, the firmware is tested and finalized. It is then "burnt" into PROMs. A set of PROMs are installed in each Custom Processor. The FDF equipment listed above is not required in an end-user site. The resulting CUP product connects directly to the Megabus system bus and requires no cables of any kind.

5.1 FDF Interfaces

Figure 5-1 is a diagram of the interconnections among the elements which comprise a firmware checkout "test bed".

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

FDF INTERCONNECT DIAGRAM



CUSTOM PROCESSOR FIRMWARE CHECKOUT SYSTEM CONFIGURATION

FIGURE 5-1

```
a: Cables (2) CUP to SILO (04910202-001, 04910203-001)
b: Cable  (1) SILO to Z80 (04910230-001)
c: Cable  (1) Z80 to MEMORY (60128806-001)
d: Cables (3) CUP to MEMORY (04910204-001)
e: Cable  (1) Z80 to TERMINAL (60156745-001) for RS232
                              (60156675-001) for RS422
```


HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.2 THE HELP SCREEN

A question mark and carriage return causes a help screen to be displayed as shown in Figure 5-2. The help screen provides a summary of FDF features.

COMMAND	FUNCTION
=======	========
FWPROM	select PROMs
FWRAM	select RAM
Ca	RAM word @ a
.	pack hexadecimal
\	label fields
Ja	jump to CSA = a
B	display all brkpts
Ba	no brkpts @ CSA = a
Ba:H	halt @ CSA = a
Ba:D	disable silo @ CSA = a
Ba:E	enable silo @ CSA = a
XS+ XS-	external stop enable
:z,z,...	define epilog
*	modify epilog
/ \	scan right, left
	insert, delete
mGOn	rpt n command m times
n*	invoke epilog n
n*S	store epilog n
E	execute current epilog
"	print current display

-	silo (latest offset)
-n	silo offset = n
east,west	next,prev silo entry
north,south	next,prev silo block
nSCAa	scan from -n for addr=a
nSCDd	scan from -n for data=d
A	ADRA, ADRB, ADRP, ADRX
AA	ADRA
AB	ADRB
AP	ADRP
AX	ADRX
F	F0 through FF
Fn	one of above alterable
H	H register alterable
I	INRA, INRB, INRX, SYND
Mn	not supported
OP	RAA, B, C, & D
PB	next procedure byte
PC	program counter (PCTR)
Q	Q register
R	REG0 through REGF
Rn	REGn
=d	latest alterable <= d

FIGURE 5-2
THE FDF HELP SCREEN

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.3 THE COMMAND SET

The command set for the Firmware Development Facility can be divided into five groups as described in the following paragraphs.

5.3.1 SILO COMMANDS

When execution terminates, the silo provides a history of the last 4000 steps executed. Commands which apply to the SILO are shown in Table 5-1.

TABLE 5-1
FDF COMMAND SET
SILO COMMANDS

COMMAND	CAUSES TO BE DISPLAYED
?	Summary of commands (help screen)
-n	Silo location: -n yyyy=zzzzzz where n = silo offset from stop point, in decimal yyyy = firmware address zzzzzz = content of zbus at that address
-	Current offset in silo
nSCAxxxx	Scan SILO for a firmware address = xxxx. Start scanning at an offset of -nnn (default = 4000). Display each match until either: an offset of zero is reached (THE END is displayed) or 23 matches have occurred (hitting the space bar will display the next set of matches; hitting any other key will execute that command.)
nSCDxxxxxx	Scan SILO for a data = xxxxxx. Start scanning at an offset of -nnn (default = 4000). Display each match until either: an offset of zero is reached ("THE END" is displayed) or 23 matches have occurred (hitting the space bar will display the next set of matches; hitting any other key will execute that command.)
nNEXT	Next nnn locations in SILO nnn cannot exceed 4000; default = 1. If offset of 0 is reached, "THE END" is displayed.
NBLK	Next 23 locations in SILO If offset of 0 is reached, "THE END" is displayed.
PBLK	Previous 23 locations in SILO If offset of 4000 is reached, "THE END" is displayed.
nPREV	Previous nnn locations in SILO nnn cannot exceed 4000; default = 1. If offset of 4000 is reached, "THE END" is displayed.
RIGHT ARROW	same as NEXT
DOWN ARROW	same as NBLK
UP ARROW	same as PBLK
LEFT ARROW	same as PREV

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.3.2 RUN CONTROLS

Commands which determine what the history memory should capture and what should cause execution to terminate are shown in Table 5-2.

TABLE 5-2
FDF COMMAND SET
"RUN" CONTROLS

COMMAND	CAUSES ACTIONS
B	Display all active breakpoints
Byyyy	Delete Breakpoint at firmware address yyyy
Byyyy:z	Install Breakpoint at firmware address yyyy where: z = D: Disable capturing of history in SILO z = E: Enable capturing of history in SILO z = H: Address Halt NOTES: 1 - Address is specified by last 14 bits of yyyy. 2 - Breakpoints are armed only if command RUNB or RUNL is used.
FWRAM	CUP uses firmware in external RAM [Default] State appears on line 25
FWPROM	CUP uses firmware in PROMs (mounted on CUP boards) State appears on line 25
INIT	Clears the CUP to the initialized state; i.e., ready to enter location zero at the next clock which may be provided by depressing the RUNN, RUNB, or STEP keys.
Jxxxx	Transfer firmware control to address xxxx
nRUNB	Place CUP in RUN mode, prepared to stop after the nnn-th occurrence of a breakpoint halt. Default nnn=1. (Note: one stop for each of two addresses counts as two stops.) The contents of the EPILOG-preselected action will then be executed.
RUNN	Place the CUP in RUN mode, and continue in that mode until "STOP" or "INIT" is depressed.
nSTEP	Cause the CUP to execute nnn firmware steps (default = 1).
STOP	Put the CUP in STOP mode. NOTE: The only FDF functions allowed when not in STOP mode are "STOP" and "INIT".
XS+	Enable the CUP to stop when the external signal fed into the FDF goes from the low state to the high state.
XS-	Enable the CUP to stop when the external signal fed into the FDF goes from the high state to the low state.
XSD	Disable external stop.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.3.3 REGISTER DISPLAYS

Table 5-3 shows which Custom Processor registers can be displayed and which can be altered.

TABLE 5-3
FDF COMMAND SET
REGISTER DISPLAYS

COMMAND	CAUSES TO BE DISPLAYED
A	ADRA=xxxxxx ADRB=xxxxxx ADRP=xxxxxx ADRX=xxxxxx
AA	ADRA=xxxxxx
AB	ADRB=xxxxxx
AP	ADRP=xxxxxx
AX	ADRX=xxxxxx
F	F0 through FF on two lines: F0=xxxxxx 1=xxxxxx 2=xxxxxx ... F8=xxxxxx 9=xxxxxx 10=xxxxxx ...

Fn	F register #nn where n = 0 through F
	Fn=xxxxxx

G	RAA=x RAB=x RAC=x RAD=x
PB	Procedure byte (right justified): PB=0000xx
PC	PCTR (right justified): PC=000xxx
Q	Q register: Q=xxxxxx
R	ARAM0 through ARAMF on two lines: R0=xxxxxx 1=xxxxxx 2=xxxxxx ... R8=xxxxxx 9=xxxxxx A=xxxxxx ...
Rn	ARAM specified by nn where nn may range from F: Rn=xxxxxx
RA	ARAM location addressed by RAA: RAA=n>xxxxxx (where n is the value of RAA)
RB	ARAM location addressed by RAB: RAB=n>xxxxxx (where n is the value of RAB)
RC	ARAM location addressed by RAC: RAC=n>xxxxxx (where n is the value of RAC)
RD	ARAM location addressed by RAD: RAD=n>xxxxxx (where n is the value of RAD)
S	SHRG register: S=xxxxxx
= xxxxxx	Alter most-recently-displayed "alterable" register to equal xxxxxx.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.3.4 EPILOGUE CONTROLS

The epilogue mechanism is invoked each time the FDF terminates execution of the Custom Processor. Table 5-4 defines the command set which applies to the epilogue mechanism.

TABLE 5-4
FDF COMMAND SET
EPILOGUE CONTROLS

COMMAND	CAUSES ACTIONS
:	Define "EPILOG", a list of preselected commands which will be executed when any STOP is encountered. Format: :COMMAND,COMMAND,COMMAND,etc. No blanks allowed between a comma and the next command. 80 characters maximum. see also \,/, , and keys.
*	Display EPILOG and allow corrections. see also \, /, , and keys.
n*	Retrieve EPILOG #n (n = 2 through 6)
n*S	Save current as EPILOG #n (n = 2 through 6)
/	Skip next character of EPILOG
\	Skip previous character of EPILOG
^	Insert blank into EPILOG
~	Delete character of EPILOG
nGOTOm	Repeat previous m EPILOG commands n times (m,n = 1 through 9)
CLER	Clear display screen (only)
E	Execute (current) EPILOG
"	Transmit present FDF screen display to hard-copy printer, if attached.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.3.5 FIRMWARE ARRAY COMMANDS

The FDF's 16K by 96-bit firmware array contains the firmware to be debugged. Table 5-5 lists the commands which allow the firmware array to be displayed and altered.

TABLE 5-5
FDF COMMAND SET
FIRMWARE ARRAY COMMANDS

COMMAND	CAUSES TO BE DISPLAYED
COMMAND	MEANING (TO DISPLAY AND CHANGE WRITABLE FIRMWARE ARRAY)
Cxxxx\	Display location xxxx, by fields, with headings
Cxxxx	Display location xxxx, by fields
Cxxxx.	Display location xxxx, packed format, ready to modify
\	Revert to field format, with headings
LEFT ARROW	Move cursor to previous field, ready to modify
RIGHT ARROW	Move cursor to next field, ready to modify
.	Revert to packed format, ready to modify
UP ARROW	Move to previous location
DOWN ARROW	Move to next location
LOAD	Prepare RAM for loading of firmware. After loading, actuate "INIT" to clear the CUP. Line 25 will display "LOAD" until "INIT" is depressed.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

5.4 MISSING STALL CATCHER

The Missing Stall Catcher is an additional debugging tool which detects the absence of a required stall micro. The Missing Stall Catcher is installed "in series with" the FDF. It intercepts and monitors the cache/megabus related activity and insures that each transaction contains the prescribe explicit or implicit stall invocation(s). The monitoring activity proceeds as the target firmware load is executing. Thus, all traversed firmware paths are scrutized for missing stalls.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

ENHANCEMENTS

The redesign of the 16-Bit Custom Processor provides the following additional features:

1. A readable 10-bit Channel Number.
2. A configurable cycle stealer so that the Custom Processor can behave as a low priority bus requestor even when "plugged into" a high priority MEGABUS slot.
3. Backward compatible Firmware visibility; i.e., an existing set of PROMs (of any incarnation) will work.
4. A bidirectional "local" interface so that both reads and writes may use the "backdoor"; i.e., not require MEGABUS cycles.
5. A semi-alterable Configuration Register which determines certain operational characteristics of the 16-Bit Custom Processor as shown in Table A-1.
6. A "timeslicing" mechanism which can be programmed to interrupt after 2, 4, 8, or 16 milliseconds have elapsed.
7. Recognition of a MEGABUS cycle "Function Code 01" directed at the CUP channel number, which is interpreted as a local CLEAR and intiates the QLT sequence.
8. A mechanism which interrupts processing when a power failure is detected.
9. An optional 16k bank of writeable firmware space.
10. A multiprocessor feature which only retries NAKed lock requests after detecting that an unlock cycle has occurred.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY

TABLE A-1
CONFIGURATION REGISTER

CONFIGURATION BITS (read only)	Three bits (C,B and A) which determine the value of the three-least significant bits of the reply to the "who-are-you" (function code 26) inquiry.
CSTEAL (read only)	Determines how the Custom Processor will behave when requesting the MEGABUS. When Cycle STEAL is false, the CUP is awarded MEGABUS cycles as a function of its position in the bus. When Cycle STEAL is true, the CUP is awarded MEGABUS cycles only when no other are requesting.
CMDPAR (read/write)	Determines whether or not the CUP will generate and check parity on the MEGABUS command lines.
APLONG (read only)	Determines whether ADRP operates as a full 24-bit register or as a 9-bit/15-bit partitioned register. When APLONG is false, ADRP does not increment beyond bit 23, ADRP(23-31) may be loaded without disturbing ADRP(08-22) and the APWRAP feature is enabled.
AP'512 (read only)	Determines whether the procedure-page-cross detector assumes a page size of 512 or 8192 bytes.
FCODE1 (read/write)	Enables the restart feature. When FCODE1 is true, a MEGABUS command directed at the CUP channel number, having a function code of 01, and having data bit 0 = 1 causes the CUP to cancel all real or imagined stalls or waits and causes it to initiate its QLT sequence.

HONEYWELL BULL CONFIDENTIAL & PROPRIETARY