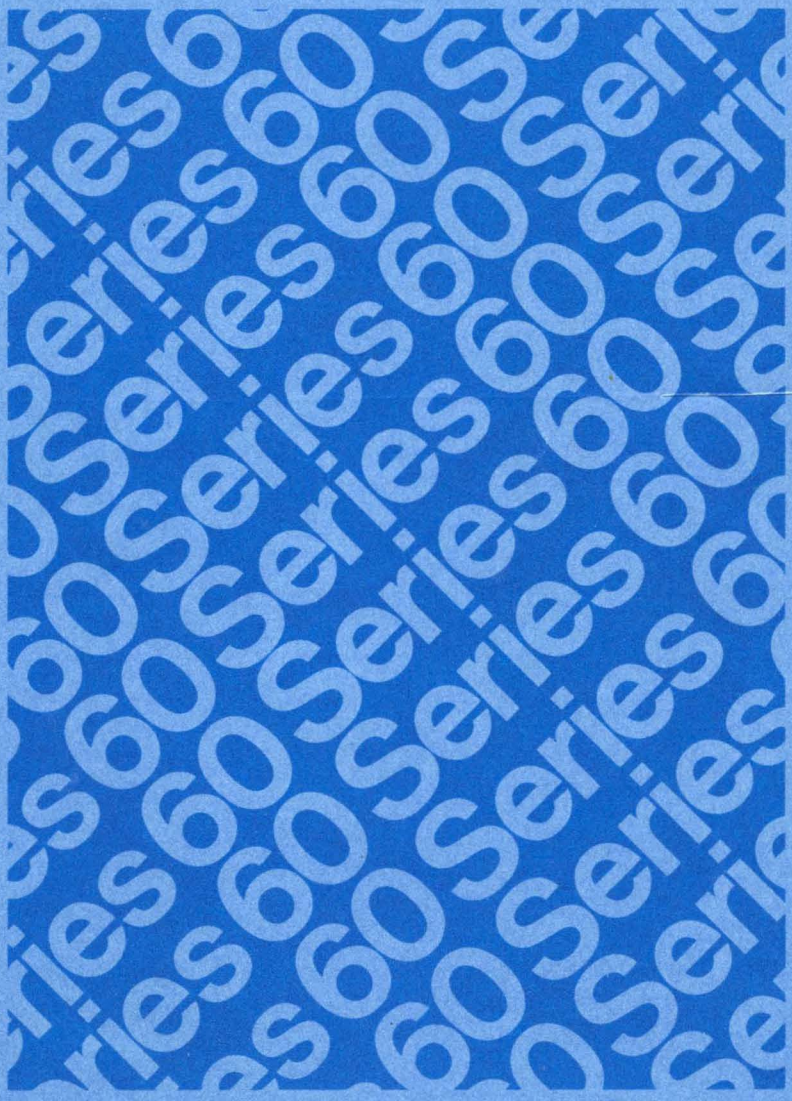
Honeywell



LEVEL 6

HARDWARE

TYPE CPF9509 WRITABLE CONTROL STORE USER'S GUIDE

TYPE CPF9509
WRITABLE CONTROL STORE
USER'S GUIDE

Document No.: 71010620-100 Order No.: FQ41, Rev. 0

System Integrity Disclaimer

Honeywell normally assumes responsibility for assuring the compatible coexistence of the total computer system, including hardware and software modules, as specified in appropriate Honeywell literature. The assumption of this responsibility is based on extensive planning, specification, stability, and qualification testing of each component and of the integrated system.

The Writable Control Store allows a user to control the underlying hardware base of a Model 43 or higher system. Because user microprogramming can bypass both the normal hardware and software integrity controls, Honeywell cannot ensure system integrity, compatibility, or performance once the WCS is utilized to execute user generated firmware.

PREFACE

This manual describes how to use the Writable Control Store (WCS) feature, enabling a user to successfully generate and execute firmware routines in the Central Processing Unit (CPU), and includes a description of the Model 43/53 CPU. It is written to provide a microprogrammer with an understanding of the microinstruction codes, the assembler, and the loading procedures to perform the above tasks. It assumes that the reader has a working knowledge of the Level 6 architecture, the CPU, the associated system software, and the applicable operation procedures. For those who are unfamiliar with this information, it is recommended that they familiarize themselves with the material contained in the following support documentation.

- Honeywell Level 6 Minicomputer Handbook (Order No. AS22)
- GCOS 6 Program Preparation (Order No. CB01)
- GCOS 6 Commands (Order No. CB02)
- GCOS 6 Assembly Language Reference (Order No. CB07)
- GCOS 6 MOD 400 Program Execution and Checkout (Order No. CB21)
- GCOS 6 MOD 400 System Building (Order No. CB23)
- GCOS 6 MOD 400 Operator's Guide (Order No. CB24)

CONTENTS

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1 - INTRODUCTION

Today's continued advancement of computer technology has produced a user-accessible control store extension within a Central Processor Unit (CPU) called Writable Control Store (WCS). This WCS feature provides a user with extremely powerful hardware instructions that extend as well as enhance the standard CPU software instruction set to achieve optimum CPU performance for software routines critical to a particular application, and allows them to execute at speeds comparable to the native CPU instructions. They accomplish this by enabling a user to write customized microprograms (firmware) into a CPU to replace repetitive software routines. Much of the speed enhancement derives from the ability of the firmware to perform several simultaneous operations in a single firmware step.

The application for the WCS feature varies considerably from one user to another. It enables a microprogrammer to enhance and augment the native CPU firmware to produce the equivalent of a customer built, specialized central processor. Software routines (or programs) designed to run without the WCS feature are unaffected by its presence.

The closeness of the firmware to hardware requires that the microprogrammer have a solid understanding of all the CPU functions, characteristics, and limitations. The key to effective custom-microprogramming lies in identifying the most frequently used algorithms for a particular application, and coding only those functions into microprogram routines. With effective microprogramming, a user can increase the overall performance of the CPU with a minimum conversion of software programs to microprograms.

1.1 GENERAL CHARACTERISTICS

The WCS feature augments the control store facility in the CPU with additional storage capacity that is alterable by the user. Its basic functional characterictics are:

- Capacity of 1K or 2K 64-bit words.
- WCS assembler to assemble firmware routines.
- Loader to load the assembled routines into the WCS.
- Facility for a Microcode Analyzer to display and monitor firmware sequencing and key data with the execution of each firmware step.

This feature does NOT replace the standard firmware, but provides a micorprogrammer with the opportunity to add to it.

1.2 USING THE WCS

All microprograms (firmware routines) are initially processed through the WCS assembler, which accepts mnemonic source input and creates object files suitable for loading into control store. The WCS loader accepts the assembler generated object code and loads it into the WCS. When User Generic instructions are processed in the CPU, control is passed from the standard firmware to the user firmware in the WCS, where control of the CPU is maintained until completion of the user microroutine. Upon completion, the user firmware returns CPU control to the standard firmware. The entry point into the standard firmware is normally at the starting location of a microroutine used to fetch the next procedure word (i.e., next instruction). However, the user can specify any return address. It should be noted that while user firmware is in control of the CPU, the user is responsible for maintaining software service routines, such as polling for interrupts, testing for traps, etc.

Writing microprograms should be performed in a systematic and orderly manner. The user should:

1. Define the function of each new instruction.

2. Acquire a good working knowledge of CPU functions, characteristics, and limitations.

3. Define the task(s) the hardware must perform, such as drawing a cycle on a screen, and design a hardware algorithm to perform the task(s).

4. Determine the firmware steps required to implement the subject algorithm.

NOTE

A firmware step can perform several parallel operations. Hence, a microprogrammer who may tend to think in terms of sequential programming operations, can save considerable steps and timing in the microprogram by combining the applicable operations into one step, thus taking maximum advantage of the parallel operation, as well as writing a more efficient microprogram. For example, an Add and a Branch operation can be combined into a single firmware step instead of two steps.

5. Enter each firmware step as a source statement on a suitable coding form.

6. Transcribe each source statement onto punched cards, or enter each statement directly onto a disk file via an on-line terminal.

7. Process the source file through the WCS assembler to obtain a machine-language object file, and output the object file to disk or magnetic tape. This operation also produces a combined listing of the source file, object file, and assembler diagnostic messages for use during debugging.

8. Load the object file into the CPU for analysis and debugging.

A Microcode Anaylzer is available that selectively displays pertinent CPU and WCS information for use in debugging microprograms. Included are the control facilities to single step the CPU through a microprogram, to set up a halt address that halts the CPU at any specified address, and to display the last 16 steps executed by the CPU.

SECTION 2 - HARDWARE

The microprogrammer must know the hardware configuration in use to properly program the firmware. An understanding of the function of each unit and its relation to the entire system will make the task easier.

The hardware required to implement user generated microprograms consists of two basic units (see Figure 2-1); the Central Processor Unit (CPU) and the Writable Control Store (WCS). The Microcode Analyzer is a tool to aid microprogram debugging.

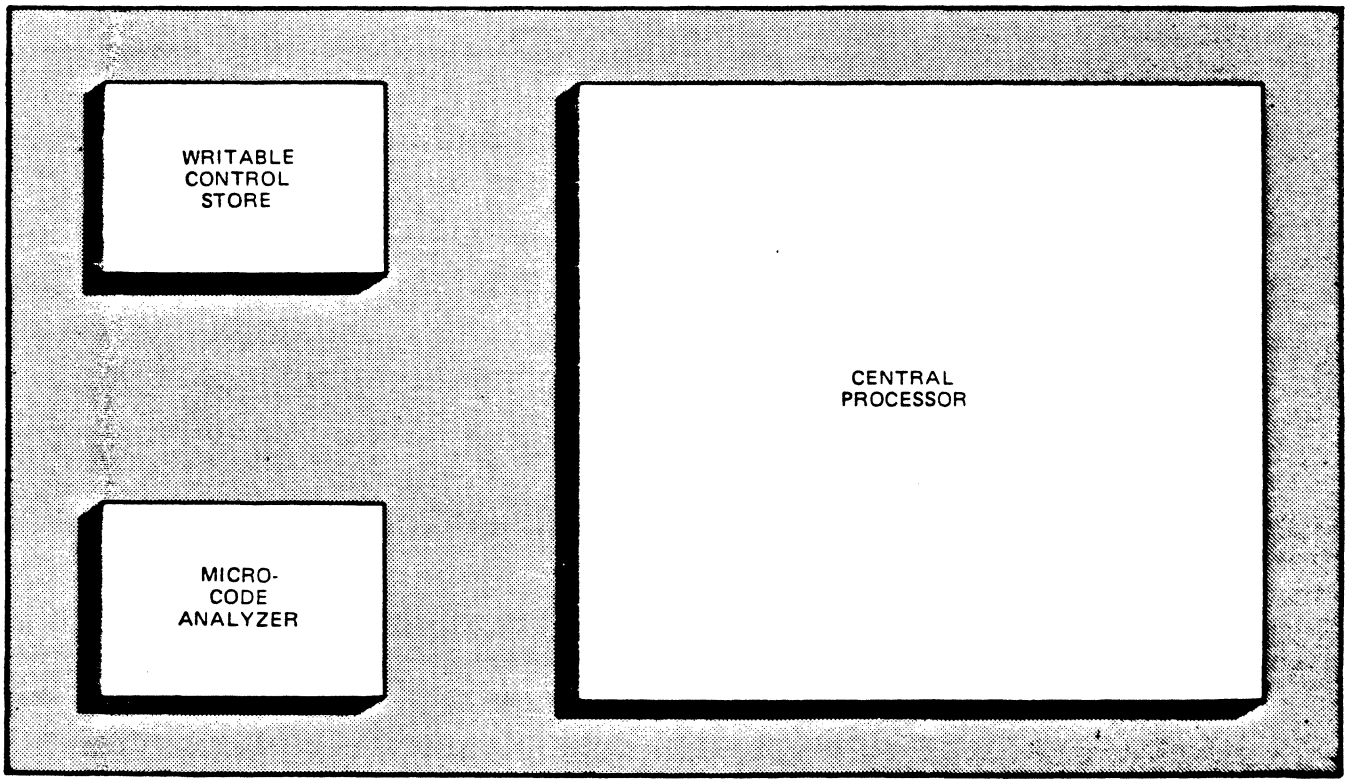


Figure 2-1 Hardware Configuration

The CPU is the computing and control portion of the Level 6 system, which processes the data and address information required by the microprogram. It controls not only the flow of information within the CPU but also the flow of data between the CPU and other units of the Level 6 system. Control over system operations is accomplished by selecting, interpreting, and controlling execution of all software instructions.

The WCS provides a variable extension of the native control store, which manipulates the central processor hardware. The WCS firmware functions as an extension of the native control store firmware, manipulating the CPU hardware in exactly the same manner as the native firmware. The WCS firmware is not a replacement for the standard firmware used to implement the base instruction set, but provides the user with the opportunity to add to it.

Key features of the CPU in the microprogramming environment, in addition to the 26 software-visible registers, are a number of data paths, firmware registers, and control flops; some of which are dedicated to specific functions, either by hardware structure or by native firmware usage. The CPU registers differ in length, functionality (shifting, counting, etc.) and ability to communicate with other CPU elements and the Level 6 system. Regardless of length, the bits of each register or data path are numbered from left (most significant bit) to right (least significant bit), starting with zero. Thus, a 4-bit register would have bits numbered 0 through 3, while a 20-bit register or data path would have bits numbered 0 through 19.

All addresses and many other values in this manual are expressed in hexadecimal (base 16) notation, using the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. When there is any danger of ambiguity between hexadecimal and decimal numbers, the hexadecimal number is written in this manual using the pound sign (#) as a suffix.

The registers, flops, data paths, and processing elements that comprise the CPU are divided into six functional areas: microprocessor, internal bus, Megabus* interface, miscellaneous hardware, firmware sequencing, and master clock (see Figure 2-2). Each of these areas is controlled by a distinct set of microinstructions, which are described in detail in Section Three.

The remainder of this section describes:

- The six areas of the CPU
- Entering and leaving the WCS
- Use of the CPU registers, counters, and control flops.

*Trademark of Honeywell Informations Systems, Inc.

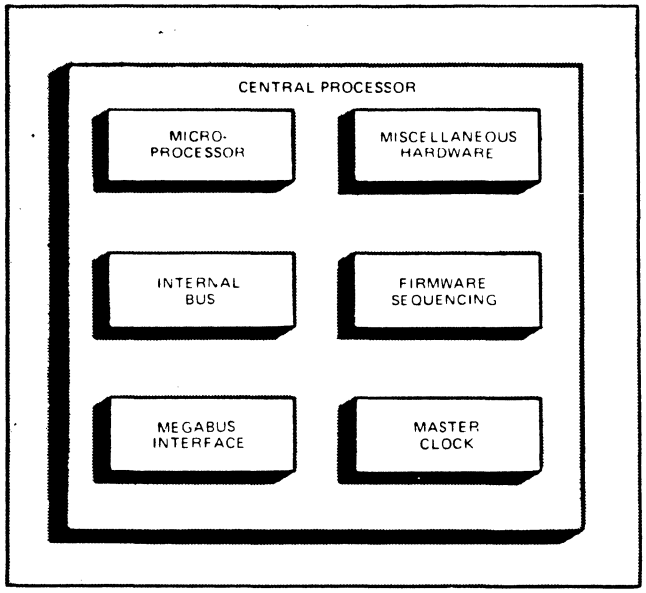


Figure 2-2 Central Processor Area

2.1 MICROPROCESSOR AREA

The microprocessor area (see Figure 2-3) performs the arithmetic, logical, and shift operations required by the Level 6 system, including storage of operands for subsequent use by the microprogram and over half of the software visible registers.

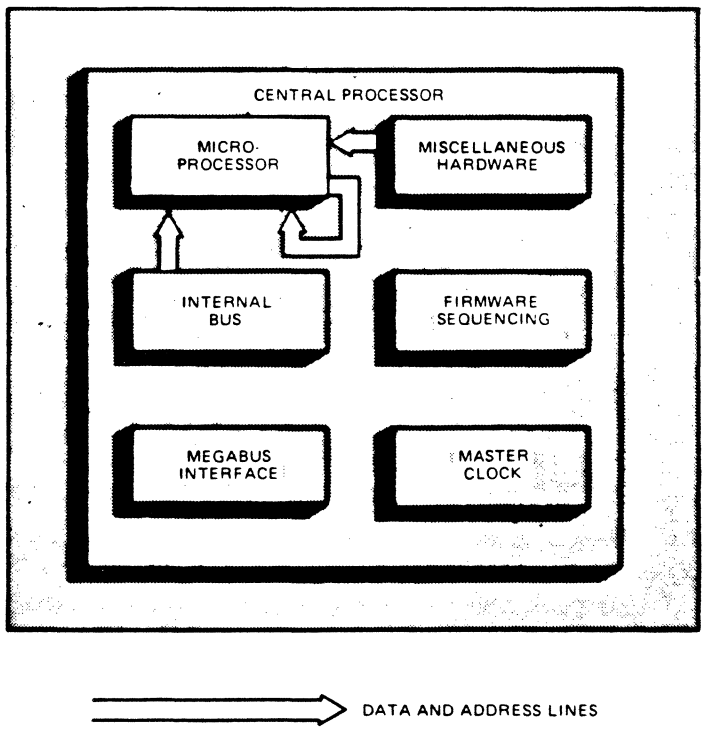


Figure 2-3 Microprocessor Area

The interface to the microprocessor area from the internal bus area enables processing of operands from CPU areas that are external to the microprocessor area. The microprocessor area includes a register file that provides storage for sixteen 20-bit registers. Among these are software visible registers R1 through R7 and B1 through B7. Results of the microprocessor operations can (optionally) be stored within the microprocessor area, but regardless of whether or not these storage facilities are used, the results are available for testing and/or distribution (via the internal bus) to destinations outside the microprocessor area.

2.2 INTERNAL BUS AREA

The internal bus area (see Figure 2-4) selects data from any one of several sources, and makes the data available to destinations both inside and outside the internal bus area.

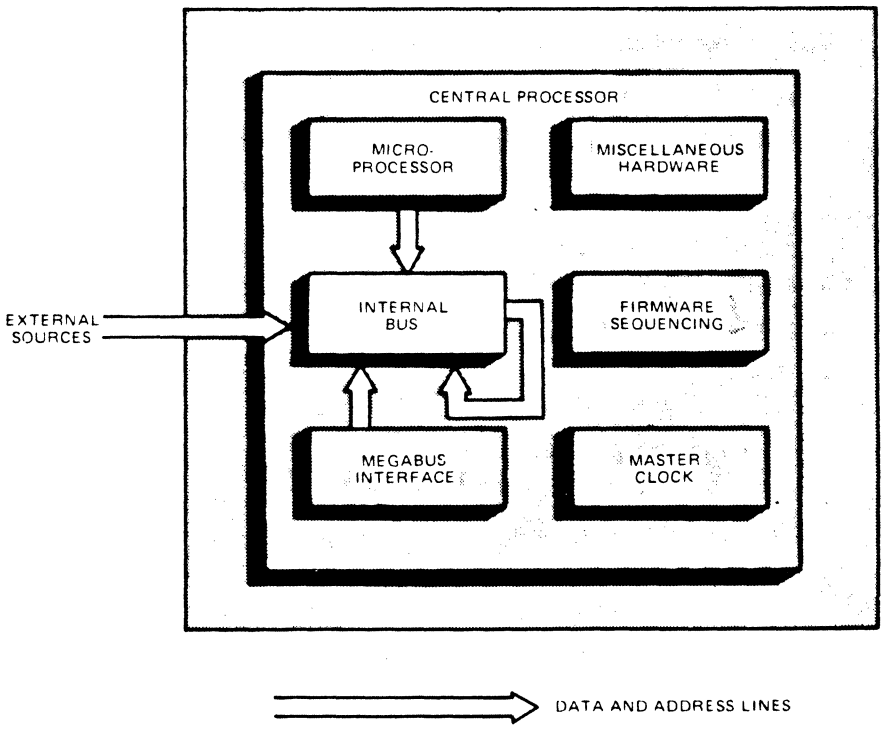


Figure 2-4 Internal Bus Area

Elements that functions as internal bus sources include:

- Microprocessor outputs
- Sixteen additional 20-bit registers (RAM)
- Megabus buffer registers
- Constant-generation facilities
- Other sources.

Elements (within the internal bus area) that may serve as destinations for the internal bus data include:

- 16 RAM registers
- Memory address register and program counter
- Indicator register
- Other registers.

In general, the internal bus microinstruction permits selecting a single source and optionally delivering copies to one destination in each of the four categories previously listed. Internal bus data are also available for use by the other CPU areas.

2.3 MEGABUS INTERFACE AREA

Megabus cycles originating from the CPU are processed by the Megabus interface area (see Figure 2-5). For example, if the CPU wants to store a word in main memory, it sends the word together with its memory address (via the Megabus interface area) down the Megabus to main memory.

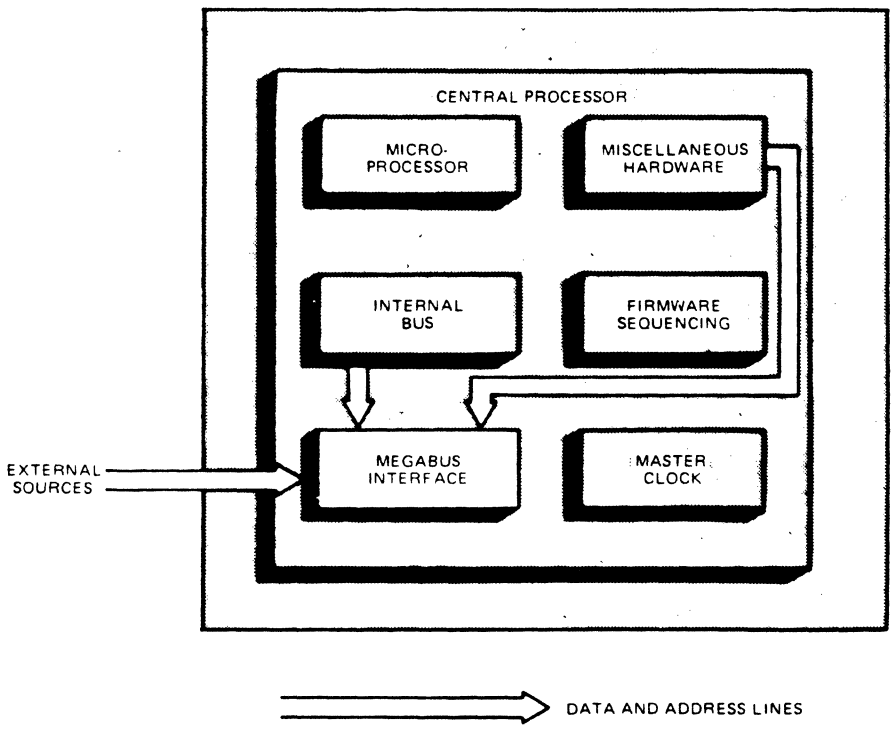


Figure 2-5 Megabus Interface Area

The six types of communication permitted over the Megabus are: (1) memory read request, (2) I/O read request, (3) read response, (4) memory write, (5) I/O write, and (6) interrupt. The microprogrammer can use combinations of these Megabus operations depending on the type of communication desired with other units of the Level 6 system.

NOTE

Communication types (1), (2), (4), and (5) are subject to control by the Megabus interface area, while types (3) and (6) are controlled by the internal bus area.

The interfaces among the internal bus, miscellaneous hardware, and Megabus interface areas enables the sending of data, address, and control information to the Level 6 Megabus from CPU areas that are outside the Megabus interface area.

2.4 MISCELLANEOUS HARDWARE AREA

The miscellaneous hardware area (see Figure 2-6) includes the remaining control flops and registers required by the CPU.

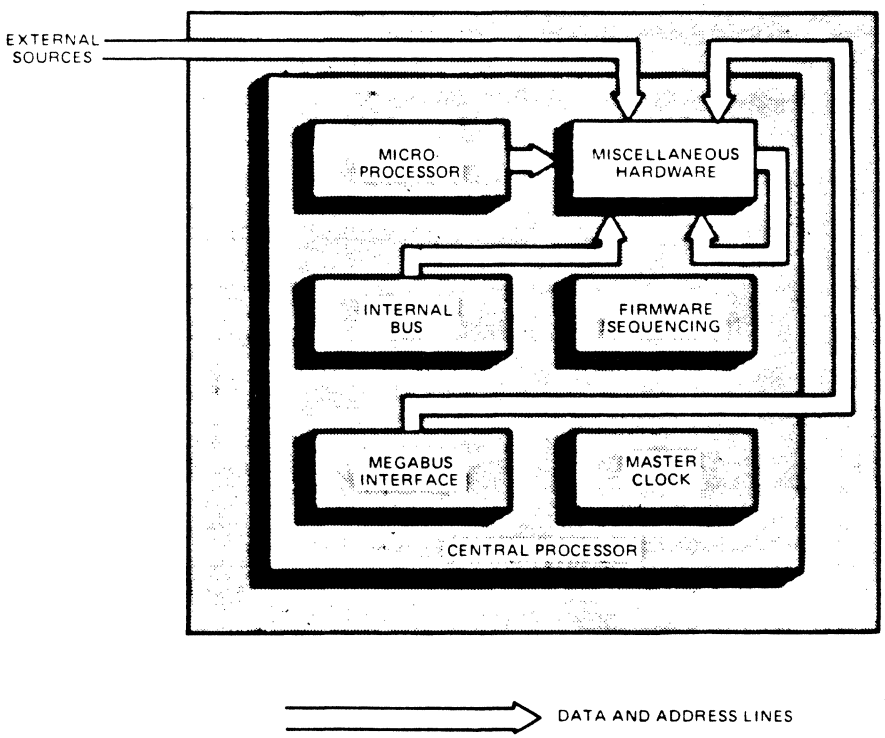


Figure 2-6 Miscellaneous Hardware Area

The interface to the miscellaneous hardware area from the microprocessor, internal bus, and Megabus interface areas permit the microprogrammer to alter the state of the various control elements using sources both internal and external to the miscellaneous hardware area.

These control elements modify the actions of other CPU areas, and can also be used to save signals generated in the current firmware step for use in subsequent firmware steps. For example, if the microprogrammer wants to postpone a firmware branch based on whether or not an arithmetic operation produced overflow, the overflow signal from the microprocessor area can be used to alter the state of a control flop within the miscellaneous hardware area. The state of the control flop can later be used as the test conditon for a firmware branch operation.

2.5 FIRMWARE SEQUENCING AREA

The firmware sequencing area (see Figure 2-7) provides the next firmware address for the control store. Every firmware step specifies a test condition that interrogates various flops and signals from other CPU areas to determine which of two alternate addresses is the next firmware address. Unconditional branching is supported by a test condition that always evaluates "false".

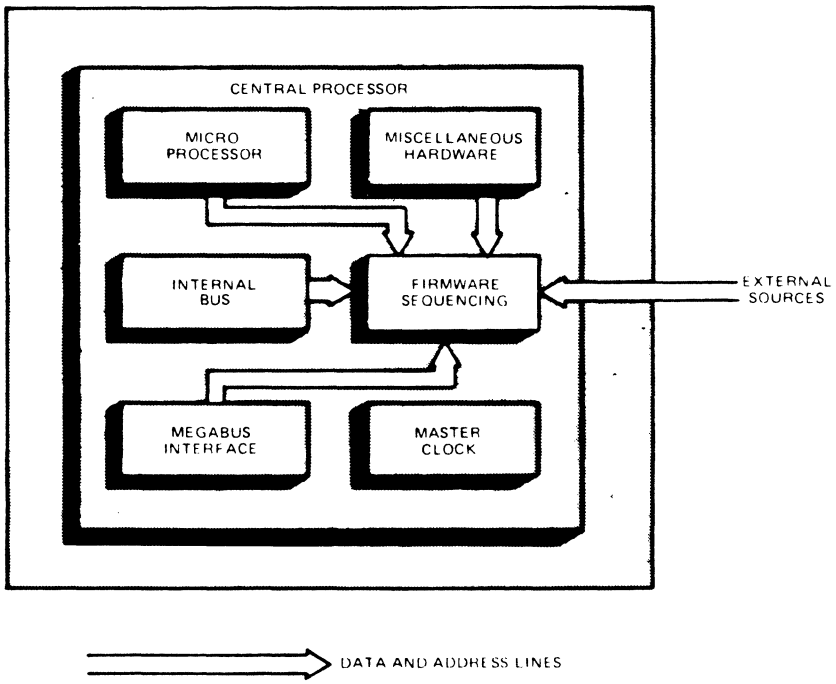


Figure 2-7 Firmware Sequencing Area

There are two addressing modes; Transparent and Sequential. In the Transparent mode, the mode used by the native firmware, every firmware step explicitly specifies the address of its successor. The Sequential mode appears more like typical software in that control generally proceeds to the next sequential location. The Transparent mode makes it possible to produce more compact code, whereas the Sequential mode is easier to learn.

2.6 MASTER CLOCK AREA AND TIMING CONSIDERATIONS

The master clock area (see Figure 2-8) generates the timing signals necessary for proper operation of the CPU.

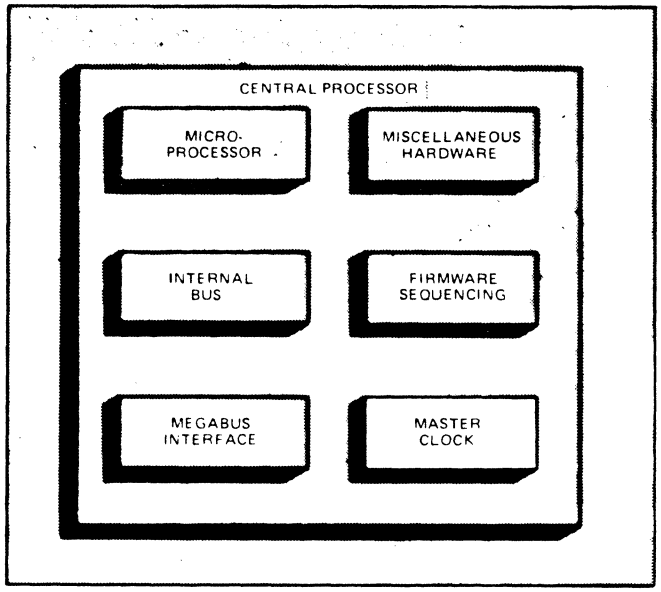


Figure 2-8 Master Clock Area

The timing signals distributed throughout the CPU provide four clock cycles that differ only in the duration of the cycle. The duration of the clock cycle for each firmware step is selected by the firmware assembler to provide the fastest performance consistent with reliable operation of the hardware. In rare circumstances, it will be necessary for the microprogrammer to override the assembler clock controls. This action will be required when the duration of a clock cycle must be increased to accommodate conditions arising from the actions of a prior firmware step.

All firmware controlled registers and flops in all areas of the CPU, with the sole exception of the 12-bit instruction register (F), are loaded, cleared, incremented, and/or shifted synchronously at the end of the firmware step calling for such action(s). Any testing, copying, etc., dependent on the content of a register that is being altered in the same firmware step, may be assumed to operate on the current contents of the register (i.e., the register contents before being altered by the current firmware step), except as explicitly noted.

Special timing considerations apply to firmware steps involving Megabus read request or Megabus write operations. Read requests are not generally completed until well into the next firmware step, imposing some restrictions on the microprogrammer. If the read request is not immediately followed by a firmware step that uses the response, the address selection must be maintained during the first step following the read request step. The acknowledge signal from the Megabus may be copied and/or tested during the firmware step immediately following the read request step.

During Megabus write operations, the acknowledge signal from the Megabus is received in time to be copied within the same firmware step, but not early enough to affect firmware sequencing reliably (refer to subsection 3.3).

A different situation exists regarding read responses (via the Megabus) to a CPU request. If the response arrives before the firmware is ready to use it, the data are buffered until requested. If the firmware attempts to use the data before the response has been received, the interface hardware automatically stalls the CPU master clock until the data arrives. The microprogrammer must avoid requesting data from a Megabus buffer if the read request was rejected; there is no limit to the patience of the master clock awaiting data that will not be received.

2.7 ENTERING AND LEAVING THE WCS

To minimize timing problems when transferring microprogram control from native control to the WCS, or vice versa, advantage is taken of the synchronization capability already designed into the Megabus interface area. Control is transferred by causing the CPU to issue a Megabus cycle (I/O write) addressed to the WCS. By properly timing its acceptance of this command, the WCS hardware automaticlly assures a clean transfer of control.

Native firmware performs the above operation whenever the first word of an instruction lies in the range 0080# through 00BF#. The location to which control is transferred is one of the first 16 locations in the WCS; the specific location is identified by the least significant hexadecimal digit of the instruction word. The content of the various CPU registers and flops at the time of entry into the WCS is described in subsection 2.8.

When it is desired to return control from the WCS to the native control store, the user must create the appropriate I/O control word (with the WCS channel number and a function code of 25#) and transmit this command to the Megabus, simultaneously specifying the native control store address to which control should return (refer to examples 6 and 7 in subsection 3.7). Normally, return is to location 020# for the next instruction fetch operation.

Trap conditions of two kinds can occur:

- Conditions detected by hardware during Megabus cycles.
- Conditions detected by firmware test and branch operations.

The first category includes:

1. Bus parity errors, memory parity errors, and uncorrectable errors detected by the memory Error Detection and Correction (EDAC) logic. These are sensed by the CPU when the bus data are sourced to the internal bus.

2. References to unavailable resources. These are sensed by the CPU during a write cycle, or during the firmware step following a read request cycle.

3. Illegal addresses. These are detected by WRAP testing (refer to subsection 2.8.2 - WRAP Control Flop), or by the Memory Management Unit (MMU); sensing of illegal addresses by the CPU is timed similarly to that of unavailable resource references.

When any of the above trap conditions are encountered, hardware forces the firmware to location 000 in the native control store; this is the starting address for the native firmware sequence that analyzes the trap condition and generates an appropriate trap.

All other trap conditions are considered under the second category, i.e., they are detected by conditional firmware branches as required by the functional specification applicable to the instruction being executed. When such a trap is detected, the firmware should exit to the location TRAP (33B#) in the native firmware, after first ensuring that:

1. RAM location 0 contains the instruction word to be reported*.

2. The XB and CTR register contents are appropriate for the Z-word*.

3. The Q register contains the address (if any) to be reported in the A-word*.

4. Register B0 contains a trap vector code equal to 40# minus the desired trap number*.

*Refer to Honeywell Level 6 Minicomputer Handbook (Order Number AS22) for a description of Trap Save Area contents. The following subsection defines all CPU registers, counters, and control flops that are visible to the microprogrammer, including those registers and counters mentioned above.

2.8 USE OF CPU ELEMENTS

The CPU registers, counters, and control flops that are visible to the microprogrammer can be classified into four categories:

- Software Visible (SWV)
- Firmware Dedicated (DEDIC)
- Working Storage (WORK)
- Autonomous (AUTO).

The possible uses for elements in the above categories are described in the following subsections and summarized in Table 2-1.

2.8.1 Software Visible

The registers in this category should not be altered except as explicitly required by the functional definition of the current instruction.

Registers D1 through D7

Registers D1 through D7 reside in bits 4 through 19 of microprocessor register file locations 1 through 7 (bits 0 through 3 of each register are not software visible, but are not usually useful as working storage).

Registers B1 through B7

Registers B1 through B7 reside in microprocessor register file locations 9 through F.

Register M1 through M7

Registers M1 through M7 reside in bits 12 through 19 of internal bus RAM locations 1 through 7 (refer to subsection 2.8.2 - RAM locations 4 and 6 and subsection 2.8.3 - RAM locations 1, 3, 5, and 7 for the use of other bits in these locations).

T and RDB Registers

The T and RDB registers reside in RAM locations A and B (respectively).

S and I Registers

The S register contains the system status and security codes for use within the CPU. The I register contains the CPU indicators.

P Register

The P register normally functions as a program counter, but can be freed for use as a working register (refer to subsection 2.8.3 - P register).

2.8.2 Firmware Dedicated

The registers and flops in this category contain control information and/or trap context. They should be loaded only with the information described herein so that the native functionality is preserved. They can also be used as sources for this information.

RAM Location 0

This RAM location contains the instruction word to be reported when a trap occurs.

RAM Locations 4 and 6 (bits 4 through 11)

Bits 4 through 11 of these RAM locations contain the mode information for:

1. Enabling the Real Time Clock (RTC).
2. Enabling the Watch Dog Timer (WDT).

RAM locations C and D

RAM locations C and D contain pointers to the most recently accepted Commercial Instruction Processor (CIP) and Scientific Instruction Processor (SIP) instructions, respectively. Each location must remain NULL if the corresponding external processor is not configured.

RAM location F

This RAM location must be NULL except when it points to the next word of procedure, freeing the P register for use as a working register (refer to subsection 2.8.3 - P Register).

CTR Register

The CTR register is a 4-bit counter that indicates the number of procedure words consumed in the processing of the current instruction. It is incremented (or cleared) every time the P register is incremented (refer to subsection 2.8.3 - P register).

XB Register

The XB register is a 4-bit shift register that supplies trap context information regarding indexing of bit or byte operations (refer to subsection 2.8.3 - XB register).

WRAP Control Flop

The WRAP control flop facilitates the checking of address-arithmetic firmware to detect attempts to exceed the 20-bit capacity of the address registers. If the WRAP control flop is on, any access to the Megabus (read request or write operation) will result in the transmission of an illegal address or I/O channel number. This action results in no response via the Megabus, which is interpreted as an "unavailable resource".

Whenever an index value or other displacement is to be added to a base address, the algebraic sign of the displacement should first be copied into the SIGN control flop. Then, the address modification can be performed, using the sign-extension capabilities of the microprocessor area (refer to subsection 3.1). Simultaneously, the carry signal from the most significant end of the microprocessor should be compared with the state of the SIGN control flop, and the comparison result copied to the WRAP control flop.

Read-Modify-Write Control Flop (RMWF)

RMWF is set to One when memory has been locked for the duration of a read-modify-write operation. RMWF is set and cleared by the Megabus interface area CHGLOCK operand (refer to subsection 3.3) if the F register contains one of the following instruction codes (refer to subsection 2.8.3 - F/SEL Instruction Register):

F = 002, 003, 006, 007#, or
880# $\leq$ F $\leq$ 897#, or
8A0# $\leq$ F $\leq$ 8B7#.

LOAD, TRAFFIC, and PANOK Control Flops

These flops communicate control information between firmware and the operator. Although they are described here for completeness, it is not anticipated that normal user firmware will involve any of them.

The LOAD flop can be set and cleared both by the operator and by firmware in the miscellaneous hardware area. During the system startup operations, LOAD is normally set by the operator and, when bootload action is completed, cleared by firmware. Thereafter, this flop usually remains off, but is sometimes set briefly by firmware as a means of preventing a trap to location 000 when a Megabus cycle is addressed to a possibly unavailable resource. LOAD must never be left on at the end of an instruction.

The TRAFFIC flop is loaded by firmware in the miscellaneous hardware area to control the corresponding indicator on the control panel. However, the flop is held off by hardware unless the control panel is in the Run mode. As the native firmware extracts the first word of each instruction, the word is tested

to determine if the instruction op-code is HLT. The result of this test is transferred (via the ZERO flop) to the TRAFFIC flop (subject to hardware override if the Run mode is not in effect).

The PANOK flop synchronizes the servicing of operator requests. It is set to Zero whenever the CLEAR or EXECUTE push-button is depressed and when in Register-Change mode, a hexa-decimal key is depressed. This flop is set to One by the firm-ware that services the request, and is used to prevent multiple servicing of a single key-stroke.

EFFRING, NONPROC, NOCHEK, SEGERR, and PROV Controls

These signals and flops support normal MMU operations, permit temporary alteration of access rules, and reports errors detected by the MMU.

EFFRING is a 2-bit register containing the "effective ring number", which the MMU uses to determine the degree of privilege appropriate to the current instruction, and against which memory access requests are tested. Native firmware loads EFFRING from the S register RING field at the start of each instruction. Correct procedure requires that EFFRING be modified to decrease its privilege level whenever, in the course of formulating an ad-dress, it uses data that might have been generated by a less privileged program.

NONPROC establishes a temporary change in the rules of ac-cess. Memory references which use the P register as the address source normally require "Execute" permission; when NONPROC has been set, they require only "Read" permission (refer to sub-section 2.8.3 - P register).

NOCHEK establishes a temporary suspension of the rules of ac-cess (it does not affect the mapping of segmented virtual addres-ses to physical, nor the detection of illegal, non-existent ad-dresses). The intent of this functionality is to remove restric-tions on memory access by system firmware (interrupt and trap handler, RTC/WDT service, panel routine, etc.).

SEGERR signals that the MMU has detected an error in a vir-tual address; the referenced segment is not valid, its size has been exceeded, or a protection violation has been detected. If SEGERR occurs during a memory reference, it causes the transmis-sion of an illegal physical address. This action results in no response via the Megabus, which is interpreted as an "unavailable resource". If no memory reference or access-rights test (refer to subsection 3.3.3. - MMURDACC, MMUWRACC) is requested, the SEGERR signal is ignored.

PROV signals that the MMU has detected a protection violation (failure of access rights check) on an otherwise legal address (i.e., an address in a valid segment and within the segment size). If a protection violation occurs during a memory

reference, the PROV flop is locked in the set state until cleared by the firmware function NOCHEK (this function is normally issued by the trap generation firmware). If a protection violation occurs during a firmware step that explicitly requests an access-rights test, the next firmware step may copy PROV to the MISC control flop.

NEWXR Control Flop

The NEWXR flop is used in the Transparent mode to distinguish between reentrant invocations of the XR "splatter" branch. This flop is set when the SEL register is loaded from the internal bus (e.g., during instruction fetch); it is cleared when a branch is performed to XR, XE, XW, or XF (refer to subsection 3.5), and when the WRAP flop is set or cleared (refer to subsection 3.4). The effect of NEWXR is shown in Appendix E.

2.8.3 Working Storage

These are the registers and flops available to the microprogrammer for temporary storage of operands, imtermediate results, control information, etc., during execution of each instruction. Their contents cannot generally be depended on to retain information between instructions, especially in the presence of asynchronous actions such as interrupts, unexpected traps, and RTC/WDT service requirements.

RAM Location 8

This RAM location initially* contains a pointer to the next word of procedure (copy of P register), but may be altered as desired.

RAM Locations 9 and E

These RAM locations initially contain no predictable information.

RAM Locations 1, 3, 5, and 7 (bits 0 through 11)

Bits 0 through 11 of these RAM locations normally contain zeros, and are unused by the native firmware. These fields are not easily altered because bits 12 through 19 of these locations contain software visible data (M registers) that must be preserved. They do, however, have the advantage of being able to retain formation between instructions and for prolonged periods.

*In this context, "initially" refers to the instant when the native firmware transfers control of the User Generic execution to WCS firmware.

Register File Locations 0 and 8

Register file locations 0 and 8 are called registers D0 and B0, respectively, and initially contain no predictable information.

Q Register

The Q register initially contains a pointer to the first word of the current instruction, but may be altered as desired.

F/SEL Instruction Register(s)

The F/SEL register initially contains the first word of the current instruction, but may be altered as desired. Specifically, it may be useful to copy all or a portion of subsequent instruction words into the instruction register to facilitate decoding and/or branch control.

NOTE

Numerous mappings of the contents of the instruction register are available as listed in Tables 3-8, 3-9, and Appendix E. These instructions are designed for the native repertoire; their utility to the WCS microprogrammer will depend to a large extent on the functional specification of the new instructions, and on their similarity to the native instructions.

Y Register

The Y register initially contains no predictable information. This register is primarily designed to hold nonprocedural addresses (or I/O control words) for transmission to the Megabus; however, during periods of inactivity in the Megabus interface area, the Y register may be used for other tasks.

H Register

The H register initially contains no predictable information. The principal functions of this register are to:

- Facilitate the handling of halfwords (bytes) by accepting 16-bit quantities from the internal bus, and later delivering them back to the internal bus with the two halves inter-changed.
- Assist the expansion of a 8-bit algebraic quantities to 16 bits.
- Retrieve 20-bit addresses from two adjacent words in memory.

LINK Register

The LINK register initially contains no predictable information. Loaded from the internal bus, this register is usable only by the firmware sequencing area: Transparent mode XL, XL0, and XL1 or Sequential mode LBRANCH.

MISC, SHIN1, SHIN2, SIGN, and ZERO Control Flops

These flops are initially cleared to zero.

DDLEQ0 Control Flop

This flop initially contains no predictable information.

XB Register

The XB register is initially cleared to zero. Depending on the functional definition of the current instruction, the contribution of this register to the Z-Word* may include no useful information. In this event, the XB register may be used for working storage.

P Register

As previously indicated, the P register can be freed of its normal procedure-pointing duties, and used for any purpose desired. To accomplish this the user must first copy the content of P into RAM location F (RAMF) and must notify the MMU to treat subsequent procedure references as data read operations for the purpose of checking access rights. The P register may then be loaded and used as desired. If an unexpected trap should occur (e.g. a parity fault), the native firmware will report the content of RAMF in the trap context in lieu of P. Similarly, the Z-word* will report an instruction size of one, rather than the content of CTR. Before returning control to native firmware (e.g., at the end of the instruction), the user must copy the procedure pointer from RAMF back to the P register, and then clear RAMF to NULL.

The user may find it worthwhile to invest the overhead required by the previous paragraph if the User Generic instructions to be implemented involves reading a long consecutive nonprocedural data string from memory. The relative economy of time and code space possible when using P as the address register for such an operation can quickly repay the investment.

*Refer to the Honeywell Level 6 Minicomputer Handbook (Order Number AS22) for a description of Trap Save Area contents.

2.8.4 Autonomous

These buffer registers and flops are set by external agencies (i.e., agencies which are not firmware controlled), and supply information for firmware use.

Acknowledge (ACK) Control Flop

The ACK control flop remembers whether the most recent Megabus action was accepted or rejcted: if ACK is ON, the action was accepted, if ACK is off, the action was rejected.

Megabus Data Buffer (BD)

The Megabus data buffer retains the data received from memory or an I/O channel in response to the most recent CPU read request. The content of BD remains valid, and may be reused, until the firmware issues the next read request or references the Megabus procedure buffer.

Megabus Procedure Buffer (BP)

The Megabus procedure buffer supplies the next procedure word from memory (i.e., the word pointed to by the P register). BP can be read only once per word because this read operation increments the P register.

Megabus Interrupt Buffer (RUP)

The Megabus interrupt buffer retains the latest external interrupt received (i.e., one not involved with RTC, WDT, or LEV actions). The content of RUP is valid while the Interrupt Busy (INTBSY) flop is on.

YELLOW and PARER Megabus Data Error Flops

The YELLOW and PARER flops signal detection of an error in memory or on the Megabus. YELLOW remembers if at least one data error was corrected by memory EDAC hardware since the last interrogation of this flop; YELLOW is cleared each time it is interrogated by the firmware. PARER remembers if the most recent Megabus buffer reference (BD or BP) reported either a Megabus parity error or a data error not correctable by memory EDAC hardware. Unless the control panel is in Load, Read, or Write mode, the setting of PARER forces the firmware to native location 000 for suitable trap generation (refer to subsection 2.7).

EXTRAP, INTBSY, and TICK Service Request Flops

The EXTRAP, INTBSY, and TICK service request flops are set by hardware to signal a requirement for a break in firmware flow. If execution of the current instruction is lengthy, as defined below, the user is responsible for polling these flops frequently enough to avoid degradation of system performance. The native

firmware performs this polling while extracting the first word of each instruction. It is recommended that, when polling detects such a request, the user should execute a branch to suitable firmware which will prepare the current instruction to be resumed or restarted, as appropriate, then exit to native firmware (location 020#).

EXTRAP is true when one or more external processors (CIP or SIP) has detected a trap condition. User firmware that does not communicate with CIP or SIP need not poll EXTRAP. If a Megabus request to such a processor is rejected, the firmware should test EXTRAP to determine whether the rejection might have been caused by a trap. If not, the Megabus request can be repeated; otherwise, a return to native firmware is recommended. EXTRAP becomes false when all external processors with trap conditions have delivered their trap words.

INTBSY is set when an external interrupt of high enough priority is received and accepted by the CPU hardware. No further interrupts, regardless of priority, can be accepted until firmware services buffer RUP, reloads the level field in the S register, and clears INTBSY. This flop must be polled often enough to ensure the interrupt response perfomance appropriate to the application.

TICK is set every 8-1/3 milliseconds by a crystal-controlled oscillator in the CPU, signalling the need for service of the YELLOW logic, the RTC and/or WDT, and the control panel. TICK must be polled at least 120 times per second.

Table 2-1 Use of CPU Elements (Sheet 1 of 4)

ELEMENT	LENGTH	TYPE	AREA	PRIMARY NATIVE FIRMWARE USAGE
ACK	1	AUTO	Megabus	Megabus Cycle Acknowlgement Storage
B0	20	WORK	Microprocessor	
B1-B7	20	SW	Microprocessor	Base Registers
BD	16	AUTO	Internal Bus	Megabus Data Read Buffer
BP	16	AUTO	Internal Bus	Megabus Procedure Read Buffer
CTR	4	DEDIC	Miscellaneous Hardware	Count Instruction Length
D0	20	WORK	Microprocessor	
D1-D7	20	SWV	Microprocessor	Data Registers (R1-R7)

Table 2-1 Use of CPU Elements (Sheet 2 of 4)

ELEMENT	LENGTH	TYPE	AREA	PRIMARY NATIVE FIRMWARE USAGE
DDLEQ0	1	WORK	Miscellaneous	CIP Descriptor Length Hardware
EFFRING	2	DEDIC	Megabus	MMU Effective Ring Number
EXTRAP	1	AUTO	*	External-Processor Trap
F	12	WORK	Internal Bus	Instruction MSB
H	16	WORK	Internal Bus	Byte Swapping
I	8	SWV	Internal Bus/ Miscellaneous Hardware	Indicator Register
INTBSY	1	AUTO	*	Interrupt Busy Signal
LINK	8	WORK	Internal Bus	Firmware Sequencing
LOAD	1	DEDIC	Miscellaneous Hardware	Inhibit Missing - Resource Recognition
MISC	1	WORK	Miscellaneous Hardware	
NEWXR	1	DEDIC	Firmware Sequencing	Distinguish Reentrant XR Branch
NOCHEK	1	DEDIC	Miscellaneous Hardware	Inhibit MMU Access Rights Checking
NONPROC	1	DEDIC	Miscellaneous Hardware	Inhibit MMU Execute Checking
P	20	SWV	Internal Bus	Program Counter
PANOK	1	DEDIC	Miscellaneous Hardware	Acknowledge Panel Request
PARER	1	AUTO	Megabus	Bus Parity Error or EDAC Uncorrectable Error.
PROV	1	DEDIC	Megabus	MMU Protection Violation

Table 2-1 Use of CPU Elements (Sheet 3 of 4)

ELEMENT	LENGTH	TYPE	AREA	PRIMARY NATIVE FIRMWARE USAGE
Q	20	WORK	Microprocessor	32-bit Shifts
RAM0	20	DEDIC	Internal Bus	Holds Copy of Instruction Word
RAM1-RAM7	20	SWV +DEDIC	Internal Bus	Mode Registers (M1-M7) Plus Control Information
RAM8-RAM9	20	WORK	Internal Bus	
RAMA	20	SWV	Internal Bus	Stack Pointer (T)
RAMB	20	SWV	Internal Bus	Remote Descriptor Base Register (RDB)
RAMC	20	DEDIC	Internal Bus	CIP Instruction Pointer
RAMD	20	DEDIC	Internal Bus	SIP Instruction Pointer
RAME	20	WORK	Internal Bus	
RAMF	20	DEDIC	Internal Bus	Usually Must Be Null
RMWF	1	DEDIC	Megabus	Remember That CPU Locked Memory
RUP	16	AUTO	Internal Bus	Interrupt Word
S	16	SWV	Internal Bus	System Status Register
SEGERR	1	DEDIC	Megabus	MMU Segment Error
SEL	4	WORK	Internal Bus	Instruction LSB
SHIN1, SHIN2	2	WORK	Miscellaneous Hardware	Control Shift End Effects
SIGN	1	WORK	Miscellaneous Hardware	
TICK	1	AUTO	*	8.3 MS Interval Mark
TRAFFIC	1	DEDIC	Miscellaneous Hardware	Control Panel Indicator

Table 2-1 Use of CPU Elements (Sheet 4 of 4)

ELEMENT	LENGTH	TYPE	AREA	PRIMARY NATIVE FIRMWARE USAGE
WRAP	1	DEDIC	Miscellaneous Hardware	Address-Wrap-Around Storage
XB	4	DEDIC	Miscellaneous Hardware	Subword Indexing, Hex Decoder
Y	20	WORK	Internal Bus	Megabus Address Register
YELLOW	1	AUTO	Megabus	EDAC-Corrected Errors
ZERO	1	WORK	Miscellaneous Hardware	ALU Zero Result Storage

*Not controlled by firmware.

SECTION 3 - MICROINSTRUCTIONS

The CPU hardware is controlled by firmware words (steps), each step consisting of several microinstructions that manipulate the hardware to perform desired operations. The CPU can execute up to six microincstrucions simultaneously during any given firmware step, permitting simultaneous control over the six functional CPU areas: microprocessor, internal bus, Megabus interface, miscellaneous hardware, firmware sequencing, and master clock.

3.1 MICROPROCESSOR AREA

The principal elements of the microprocessor area (see Figure 3-1 include:

- Register File (RF)
- Q Register
- Arithmetic/Logic Unit (ALU)
- Shift Logic.

The register file is a memory consisting of sixteen 20-bit locations, numbered 0 through F. The first eight locations are designated as registers D0 through D7, while the last eight locations are designated as registers B0 through B7. Up to two of these registers may be addressed simultaneously to serve as sources and/or destination for the ALU. Simultaneously, the contents of one register file location may be made available to the internal bus area.

The Q register is an additional 20-bit register that is available as an ALU source and/or destination.

The ALU performs all CPU arithmetic and logical functions. It has two 20-bit inputs and one 20-bit output. The ALU output may be directed to a destination within the microprocessor and/or made available to the internal bus area. Three auxiliary signals are generated based on the ALU results: Carry (CRY), Overflow (OVFL), and Zero detect (AUZ). For the purpose of generating these signals, the ALU function may be treated as a 16-bit or a 20-bit operation as shown in the following listing; however, the actual arithmetic or logical operation is always 20 bits wide.

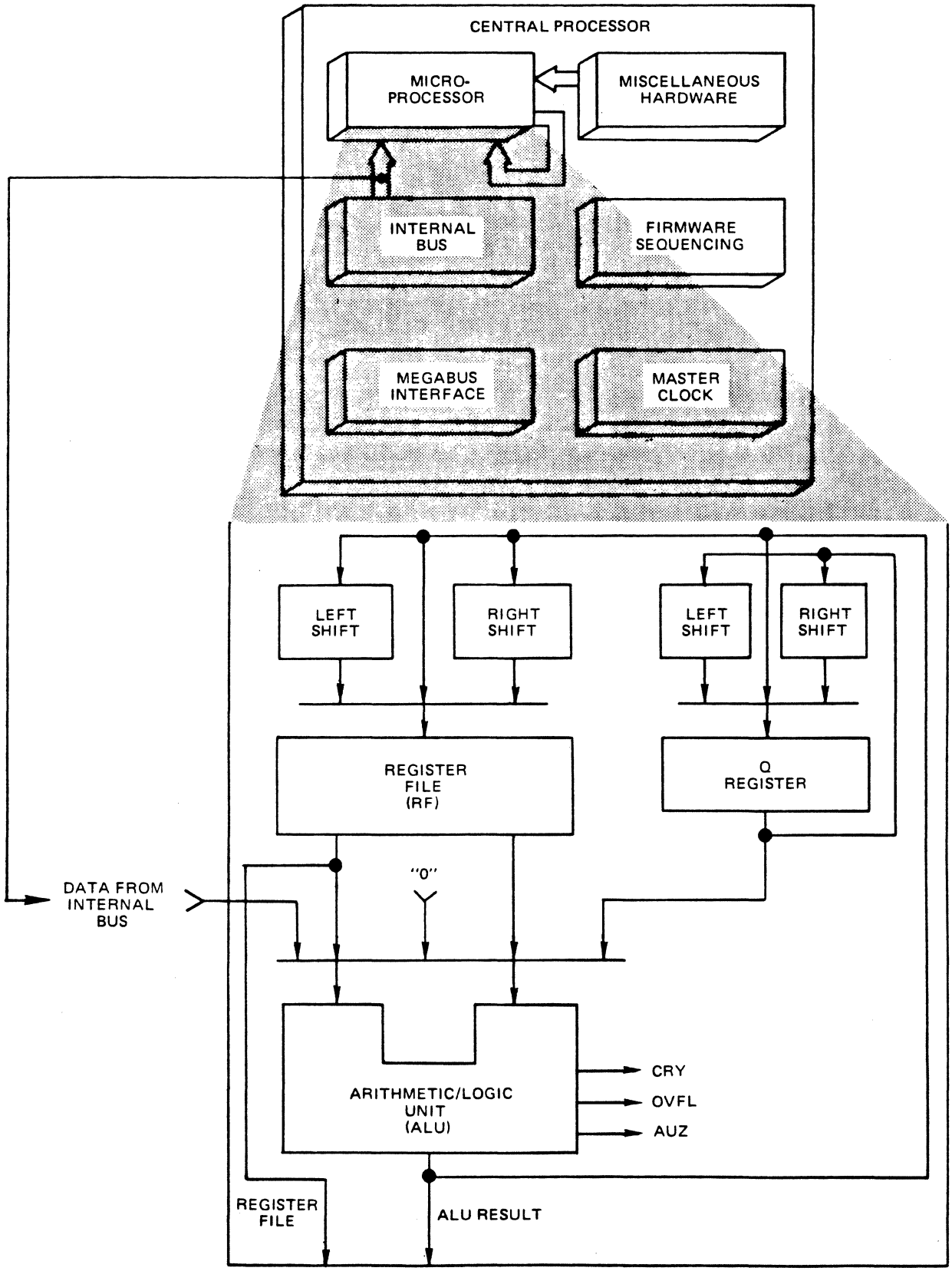


Figure 3-1 Microprocessor Area

SIGNAL	16-BIT ARITHMETIC	20-BIT ARITHMETIC	16-BIT LOGICAL	20-BIT LOGICAL
CRY	Carry from bit 4	Carry from bit 0	Undefined*	Undefined*
OVFL	Overflow from bit 4	ALU result bit 0 in liew of overflow	Undefined*	ALU result bit 0 in lieu of overflow
AUZ	=1 if ALU result bits 4-19 = 0	=1 if ALU result bits 0-19 = 0	=1 if ALU result bits 4-19 = 0	=1 if ALU result bits 0-19 = 0

*Except for the logical AND function as described in subsection 3.1.3.

The shift logic is designed to perform single-bit left or right shifts on the 16 least significant bits of the ALU result before they are written into the register file. The Q register can be shifted simultaneously to facilitate operations on 32-bit quantities (refer to subsection 3.1.4).

3.1.1 Syntax

The microinstructions that affect this functional area control the Register File Arithmetic and Logic Unit (RALU). These microinstructions contain a function and up to four operands as follows:

function △ SRC1, SRC2, DEST, SHIFT

function △ SRC1, DEST, SHIFT

where:

SRC1 is an input source.

SRC2 is an additional input source.

DEST is the destination for results (optional operand).

SHIFT is the shift operand for results (optional operand).

3.1.2 Microprocessor Sources and Destinations

Sources to the ALU may be any one (or two) of the following:

- Internal Bus (refer to subsection 3.2).
- Q Register
- Register File Location
- Another (or the same) Register File Location
- ZERO (20 bits, all zeros).

Destinations, if any, in the microprocessor may be to the following:

- Q Register
- Register File location.

Whether or not a microprocessor destination is specified, the ALU result, CRY, OVFL, and AUZ are available for simultaneous use by microinstructions in other areas. If SRC1, SRC2, and DEST all specify register file locations, DEST must be the same as either SRC1 or SRC2.

The possible sources and destinations for the microprocessor area are summarized in Table 3-1. Register file locations may be specified explicitly or as a function of fileds in the F/SEL instruction registers. The mnemonics are derived from the B-field (SEL register bits 1 through 3), the M-field (F register bits 9 through 11), and the N-field (F register bits 1 through 3). If the firmware step just previously executed altered the contents of SEL, those operands that depend on fields in SEL will use its previous contents (i.e., the contents before SEL was altered at the end of the firmware step just previsously executed).

If two register file locations are selected by operands SRC1, SRC2, and/or DEST, they must both be members of the same group, where the available groups are:

- D0, D3, D6, D7, B0, B3, B6, and B7.
- D0, D1, D4, D5, B0, B1, B4, and B5.
- D0, D2, D6, B0, B2, and B6.
- D0, DN, DN3, DNE, B0, BN, BN3, and BNE.
- D0, DB, DB3, DBE, B0, BB, BB3, and BBE.
- D0 DB, DB3, DBE, and REGSEL.
- D0, DM, DM3, DME, B0, BM, BM3, and BME.

Other restrictions on the selection of one of the microprocessor source operands are imposed occasionally by the simultaneous action of microinstructions in other areas. To facilitate cross-referencing, it is necessary to introduce the concept of "restricted selection." A microprocessor source operand is designated a restricted selection if it specified a register file location, and:

- it is the second operand of microinstruction ANDC, ADDSE, or ADD1SE, or
- it is also the internal bus source (refer to subsection 3.2.2 - Register File Locations), or
- the destination is a different register file operand, or
- the other source is the Q register, or
- the other source is the internal bus, or

- the other source is a register file operand that does not satisfy the restricted-selection rule(s) in question (i.e., if both sources are register file locations, at least one must satisfy the restricted-selection rule).

NOTE

The restricted selection concept applies only when a simultaneous microinstruction defines a restricted selection rule.

When 20-bit versions of CRY, OVFL, and AUZ are used, and the function is neither ADDSE nor ADD1SE, the restricted-selection source must be a D register (except as selected by REGSEL).

Table 3-1 ALU Source and Destination Operands
(Sheet 1 of 2)

OPERAND	EXPLANATION
B0-B7	RALU Base Registers B0 through B7 (register file locations 8-F)
BB	RALU Base Register (B0 through B7) specified by SEL(1-3).
BB3	RALU Base Register (B0 through B3) specified by SEL(2-3).
BBE	RALU Even Base Register (B0, B2, B4, or B6) specified by SEL(1-2).
BI	Internal Bus (source only).
BM	RALU Base Register (B0 through B7) specified by F(9-11).
BM3	RALU Base Register (B0 through B3) specifiefd by F(10-11).
BME	RALU Even Base Register (B0, B2, B4, or B6) specified by F(9-10).
BN	RALU Base Register (B0 through B7) specified by F(1-3).
BN3	RALU Base Register (B0 through B3) specified by F(2-3).
BNE	RALU Even Base Register (B0, B2, B4, or B6) specified by F(1-2).
D0-D7	RALU Data Registers D0 through D7 (register file locations 0-7)

Table 3-1 ALU Source and Destination Operands
(Sheet 2 of 2)

OPERAND	EXPLANATION
DB	RALU Data Register (D0 through D7) specified by SEL(1-3).
DB3	RALU Data Register (D0 through D3) specified by SEL(2-3).
DBE	RALU Even Data Register (D0, D2, D4, or D6) specified by SEL(1-2).
DM	RALU Data Register (D0 through D7) specified by F(9-11).
DM3	RALU Data Register (D0 through D3) specified by F(10-11).
DME	RALU Even Data Register (D0, D2, D4, or D6) specified by F(9-10).
DN	RALU Data Register (D0 through D7) specified by F(1-3).
DN3	RALU Data Register (D0 through D3) specified by F(2-3).
DNE	RALU Even Data Register (D0, D2, D4, or D6) specified by F(1-2).
Q	RALU Q Register
REGSEL	RALU Register D0 through D8 (register file locations 0-F) specified by SEL(0-3).
ZERO	Zero (source only).

3.1.3 Microprocessor Functions

The 14 microprocessor arithmetic and logical functions are described in the following paragraphs and summarized in Table 3-2.

ADD

SRC1 is added to SRC2.

ADD1

SRC1 is added to SRC2 plus 1.

ADDSE (add sign extended)

SRC2 must be a D register. The SIGN flop is assumed to contain a copy of bit 4 of that D register. The SIGN flop is copied to the four most significant bits of SRC2 which sign-extends SRC2 from 16 to 20 bits. The extended value is added to SRC1. SCR1 must specify the Q register or a register file location. SRC1 and DEST cannot be different register file operands. This operation is useful, for example, when employing a D register as an index register.

ADD1SE (add One sign extended)

The description of ADD1SE is identical to that of ADDSE, except:

- SRC1 must specify the Q register.
- The sign extended SRC2 is added to SRC1 plus One.

AND

SRC1 is ANDed with SRC2. CRY and OVFL are forced to One.

ANDC (AND with complement)

SRC1 is ANDed with the One's complement of SRC2. SRC1 may not be BI, SRC2 may not specify the Q register, and DEST and SRC1 may not specify different register file operands.

COPY

SRC1 appears as the ALU output.

DECR

One is subtracted from SRC1. SRC1 may not be ZERO.

INCR

One is added to SRC1. SRC1 may not be ZERO.

OR

SRC1 is inclusive-ORed with SRC2.

SUB

SRC2 is subtracted from SRC1. The ALU carry and overflow signals act as if the One's complement of SRC2 was added to SRC1 plus One.

SUB1

SRC2 is subtracted from SRC1 minus One. The ALU carry and overflow signals act as if the One's complement of SRC2 was added to SRC1.

XOR

SRC1 is exclusive-ORed with SRC2.

XORC (Exclusive OR with Complement)

SRC1 is exclusive-ORed with SRC2 and the result inverted.

Table 3-2 ALU Functions

FUNCTION	ALU OUTPUT
ADD	SRC1 + SRC2
ADD1	SRC1 + SRC2 + 1
ADDSE	SRC1 + SRC2 sign extended
ADD1SE	SRC1 + SRC2 sign extended + 1
AND	SRC1 $\wedge$ SRC2
ANDC	SRC1 $\wedge$ $\overline{\text{SRC2}}$
COPY	SRC1
DECR	SRC1 - 1
INCR	SRC1 + 1
OR	SRC1 $\vee$ SRC2
SUB	SRC1 - SRC2
SUB1	SRC1 - SRC2 - 1
XOR	SRC1 $\veebar$ SRC2
XORC	$\overline{\text{SRC1} \veebar \text{SRC2}}$

3.1.4 Microprocessor Shift Operands

Shift operands are used to shift the ALU result left or right by one bit position. A shift operand may be specified only if DEST is a register file location. The shift operations are optimized for 16- or 32-bit quantities. Although the shifts operate on 20-bit registers, the operation on the most significant four bits is generally not useful, especially for right shifts.

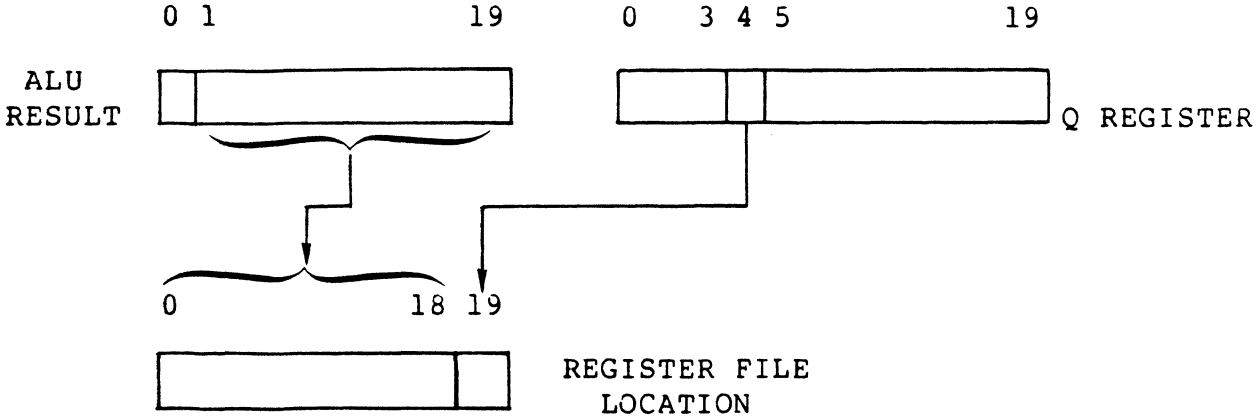
In most shift operations, the bit shifted into the vacated bit position is designated as SHIN (shift input), and is controlled by three flops; SHIN1, SHIN2, and MISC (refer to subsection 3.4). The SHIN function is selected as follows:

MISC	SHIN1	SHIN2	SHIN
0	0	0	Internal bus bit 4
0	0	1	Internal bus bit 4 inverted
0	1	0	0
0	1	1	Q register bit 19*
1	0	0	XB register bit 1 (refer to subsection 3.4).
1	0	1	Y register bit 4 (refer to subsection 3.2).
1	1	0	0
1	1	1	Q register bit 19*

*During shift right operations; otherwise undefined.

SL (Single Left Shift)

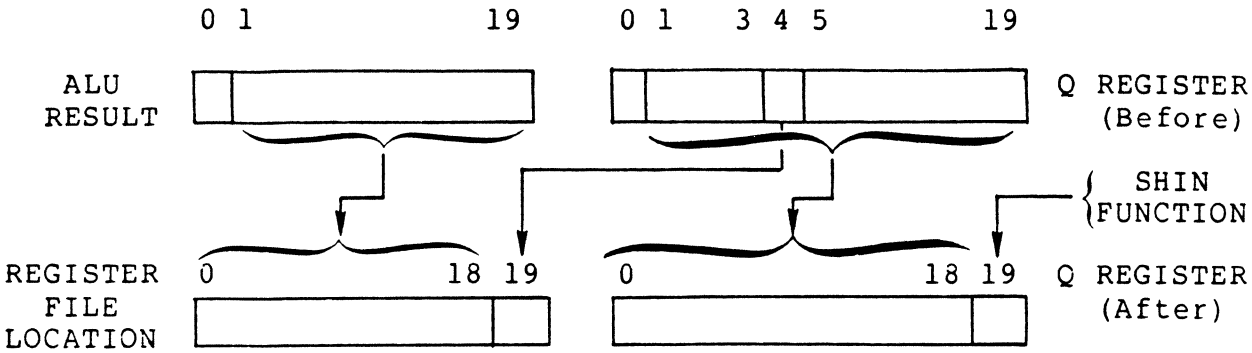
Bits 1 through 19 of the ALU result are placed in bits 0 through 18 of the selected register file location; bit 19 of the selected register file location receives a copy of Q register bit 4.



DL (Double Left Shift)

Bits 1 through 19 of the ALU result are placed in bits 0 through 18 of the selected register file location; bit 19 of the selected register file location receives a copy of Q register

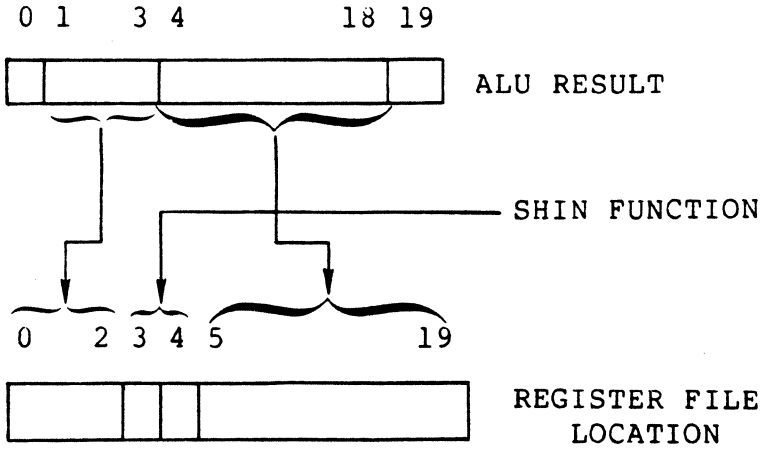
bit 4. Q register bits 1 through 19 are placed in Q register bits 0 through 18; Q register bit 19 receives a copy of the SHIN function.



Conceptually, the rigthmost 16 bits of the ALU result are concatenated with the rightmost 16 bits of the Q register and shifted left one bit position with the SHIN function shifted in on the right. The result is placed in the rightmost bit positions of the register file location and the Q register, respectively.

SR (Single Right Shift)

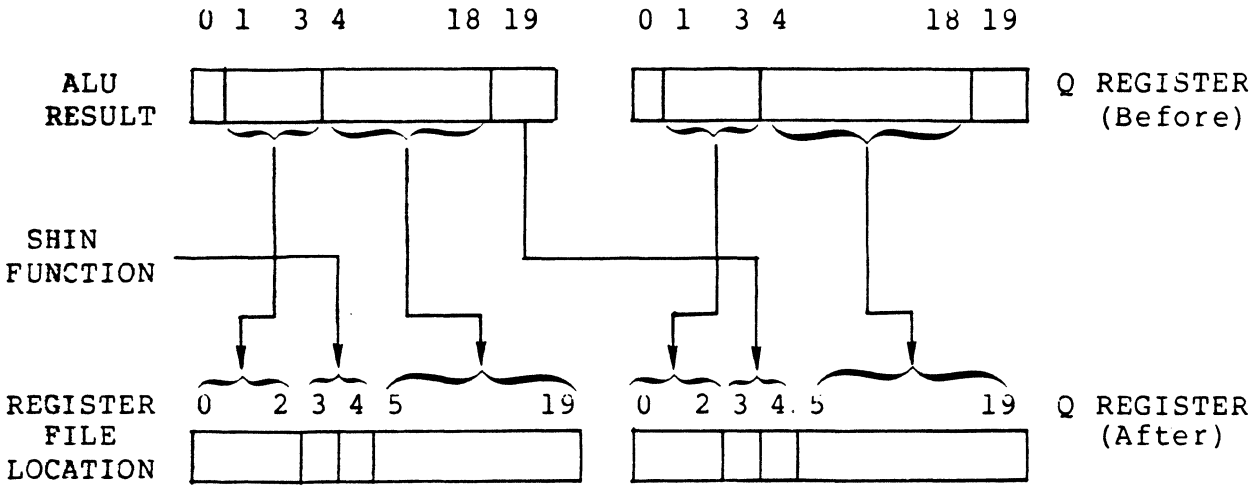
Bits 4 through 18 of the ALU result are placed in bits 5 through 19 of the selected register file location; bit 4 of the selected register file location receives a copy of the SHIN function. Bits 1 through 3 of the ALU result are placed in bits 0 through 2 of the selected register file location; bit 3 of the selected register file location receives a copy of the SHIN function.



Conceptually, the rightmost 16 bits of the ALU result are shifted right one bit position with the SHIN function shifted in on the left. The result is placed in the 16 rightmost bit positions of the register file location.

DR (Double Right Shift)

Bits 4 through 18 of the ALU result are placed in bits 5 through 19 of the selected file location; bit 4 of the selected register file location receives a copy of the SHIN function. Q register bits 4 through 18 are placed in Q register bits 5 through 19; bit 19 of the ALU result is placed in Q register bit 4. Bits 1 through 3 of the ALU result are placed in bits 0 through 2 of the selected register file location; bit 3 of the selected register file location receives a copy of the SHIN function. Q register bits 1 through 3 are placed in Q register bits 0 through 2; bit 19 of the ALU result is placed in Q register bit 3.



Conceptually, the least significant 16 bits of the ALU result and the least significant 16 bits of the Q register are concatenated, shifted right one bit position with the SHIN function filling the most significant bit, and the result placed in the least significant 16 bits of the register file location and the Q register, respectively.

3.1.5 Microprocessor Examples

The following are examples of source statements for the microprocessor area.

STATEMENT		MEANING
ADD	D0, B0, Q	Q ← ALU result ← B0+D0
SUB1	B0, D0	ALU result ← B0-D0-1
COPY	B0, B0, SR	ALU result ← B0
		B0 ← ALU result shifted right
XORC	D0, D0, D0	D0 ← ALU result ← FFFFF#
INCR	Q, B3	B3 ← ALU result ← Q+1
ADD	BI, B2, B3	Invalid (DEST must be same as SRC1 or SRC2)
ADD	D0, B0, Q, DR	Invalid (SHIFT operand may only be specified if DEST is register file location)
ADDSE	B0. B0, B0	Invalid (SRC2 must be D register for ADDSE)
ADD1SE	B0, D0, B0	Invalid (SRC1 must be Q for ADDSE)
INCR	ZERO, D0	Invalid (SRC1 cannot be ZERO for INCR)
OR	D1, D2, D1	Invalid (D1 and D2 are incompatible)
ANDC	D1, Q, D1	Invalid (Q may not be SRC for ANDC)

3.2 INTERNAL BUS AREA

The internal bus area (See Figure 3-2) provides a 20-bit wide data path that transfers data among elements of the CPU as directed by the firmware. This data path is defined as the CPU Internal Bus (BI).

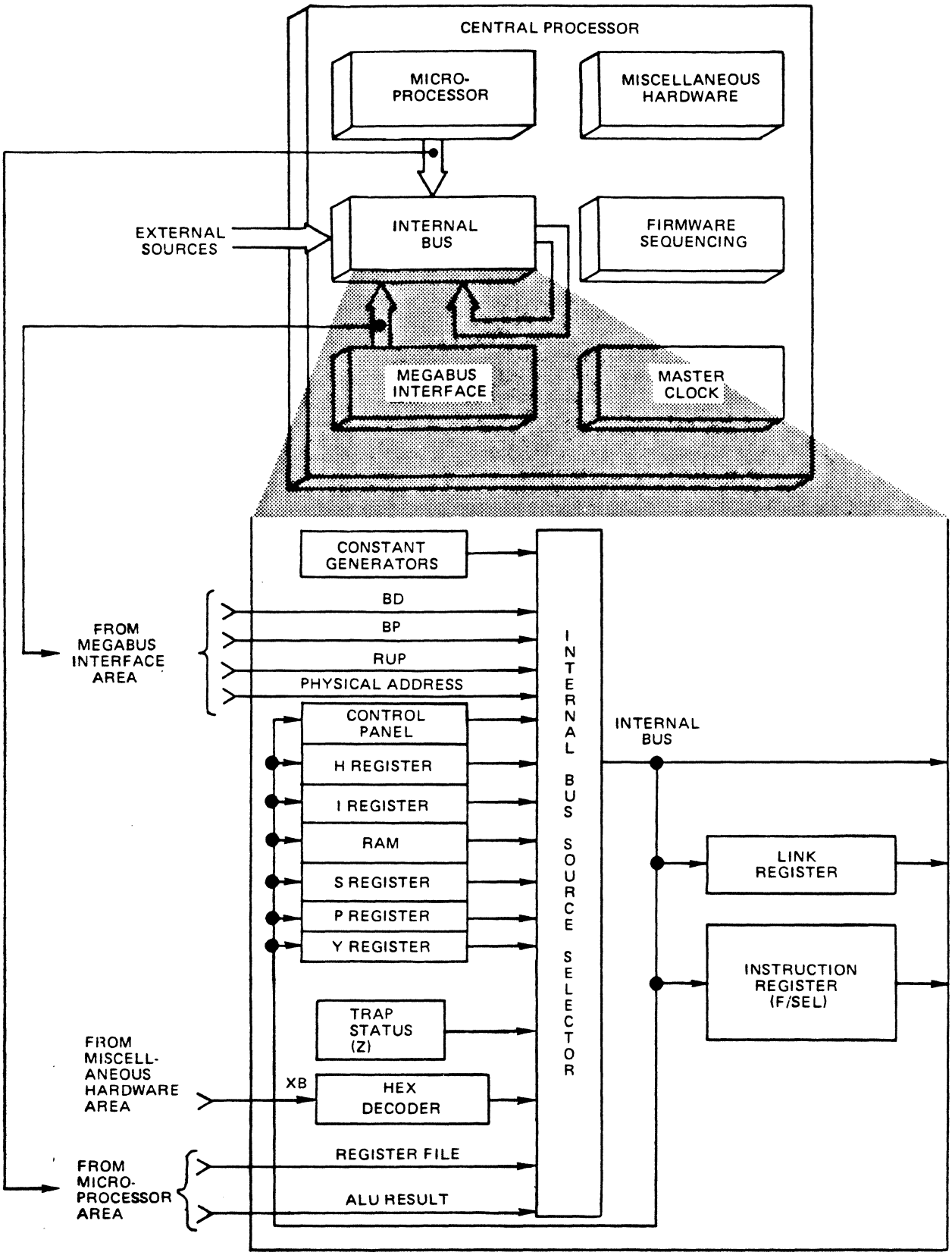


Figure 3-2 Internal Bus Area

The internal bus selects a source from one of the following five categories.

- Sources from microprocessor (RALU)
- RAM locations
- Megabus buffers
- Constants
- Other sources.

The data on the internal bus are simultaneously available to destinations in each of the following four categories:

- Megabus address registers
- RAM locations
- Indicator register (I)
- Other destinations.

3.2.1 Syntax

BI is the microinstruction which controls the sources and destinations of the internal bus. The BI microinstruction contains up to six operands as follows:

```
BI      SRC, DEST,.....,DEST

BI      SRC, SRCMOD, DEST,.....,DEST
```

where:

SRC specifies the internal bus source.

SRCMOD specifies a source modifier - may be specified only if SRC is a microprocessor source (ALU or a register file location).

DEST specifies an internal bus destination. Destinations, if any, may be specified in any order, but at most one destination may be specified from each of the four destination categories.

3.2.2 Internal Bus Sources

The following are available as internal bus sources:

- Sources from the microprocessor area
- RAM locations
- Megabus buffers
- Constants
- Other internal bus sources.

These sources are described below and summarized in Table 3-3.

3.2.2.1 Sources from the Microprocessor Area

The internal bus sources from the microprocessor area are: (1) the ALU result and (2) register file locations.

ALU

The entire ALU result (refer to subsection 3.1) may be placed on the internal bus. If SRCMOD is specified, only part of the ALU result will be used.

Register File Locations

An entire register file location (refer to subsection 3.1) may be placed on the internal bus. If SRCMOD is specified, only part of the register file location will be used. Refer to subsection 3.1.2 for possible selection restrictions. When a register file location is the internal bus source, the microprocessor destination may not be null, but must also be a register file location (although not, in general, the same location) and no shift modifier may be specified. When no microprocessor area microinstruction is specified, the assembler automatically satisfies this requirement by copying an arbitrarily selected location back into itself.

SRCMOD Operand

The optional SRCMOD operand specifies that only part of the 20-bit source from the microprocessor area will be placed on the internal bus. The two SRCMOD operands are L4 and R8.

If L4 is specified, internal bus bits 0 through 3 and also bits 16 through 19 receive copies of bits 0 through 3 of the selected source from the microprocessor area; internal bus bits 4 through 15 are forced to Zeros.

If R8 is specified, internal bus bits 12 through 19 receive a copy of bits 12 through 19 of the selected source from the microprocessor area; internal bus bits 4 through 11 receive eight copies of bit 8 of the H register; internal bus bits 0 through 3 are forced to Zeros.

3.2.2.2 RAM locations

The RAM is a random access memory consisting of sixteen 20-bit locations, numbered 0 through F. Locations 1 through 7 are also called M1 through M7. Any of these locations may be placed on the internal bus, although RAM locations cannot serve simultaneously as an internal bus source and destination.

A RAM location may be specified explicitly or as a function of the fields in the F/SEL instruction registers. If the firmware step just previously executed altered the contents of SEL, those operands that depend on fields in SEL will use its previous

contents (i.e., the contents before SEL was altered at the end of the firmware step just previously executed).

If, simultaneously, a register file location is used as a restricted selection source (refer to subsection 3.1.2) for the microprocessor, any RAM location referenced must correspond, in the sense indicated below, to the restricted-selection register-file source:

- D0 :: RAM0
- D1-D7 :: RAM1-RAM7 = M1-M7
- B0-B7 :: RAM8-RAMF
- REGSEL :: RAMSEL
- DB :: MB
- DB3 :: MB3
- DBE :: MBE
- DM :: MM
- DM3 :: MM3
- DME :: MME
- DN :: MN
- DN3 :: MN3
- DNE :: MNE

3.2.2.3 Megabus Buffers

The Megabus buffers that serve as internal bus sources are:

- Data buffer
- Procedure buffer
- Interrupt buffer.

Data Buffer (BD)

The Megabus data buffer is a 16-bit register containing the response to the most recent non-procedure read request (RDREQ, refer to subsection 3.3) issued either to memory or to a peripheral device. When this buffer is used as the internal bus source, bits 0 through 3 receive either four Zero's or a copy of H register bits 12 through 15, depending on the SRC operand specified.

Procedure Buffer (BP)

The Mebagus procedure buffer is a 16-bit register that contains the next word in the procedure stream, pointed to by the P register. The P register is incremented each time the buffer is used as a soruce. When this buffer is used as an internal bus source, bits 0 through 3 receive either four Zero's or a copy of H register bits 12 through 15, depending on the SRC operand specified.

Interrupt Buffer (RUP)

The Megabus interrupt buffer is a 16-bit register that contains the most recently accepted interrupt word (interrupting channel number in bits 0 through 9 and interrupt level in bits 10 through 15). When RUP is used, bits 0 through 3 of the internal bus are forced to Zero's.

NOTE

Servicing of interrupts requires a complex algorithm, and the user is advised to return control to the native firmware if/when this is required.

3.2.2.4 Constants

Constants that are available as internal bus sources are:

- Numeric constants
- Operands IDCy and IDSy.

Numeric Constants

A numeric constant may be specified as an internal bus source. The least significant nine bits of the constant are considered to be a signed integer, which is sign extended to 16 bits and placed on the least significant 16 bits of the internal bus; the most significant four bits of the bus are forced to Zero's. Numeric constants are thus of the form Oxxyz#, where x equals 0 or F. The second least significant digit of the next firmware address (refer to subsection 3.5) must equal y.

Operands IDCy and IDSy

These operands are intended to facilitate the creation of Megabus control words for communicating with external processors (CIP, WCS, etc.), and function as follows:

- Internal bus bits 0 through 3, 5 through 10, and 15 are forced to Zeros.

- Internal bus bit 4 receives a copy of Y register bit 4.

- Internal bus bits 11 through 14 receive the channel number of the associated external processor (bits 11 and 12 receive 01 if IDSy is sepcified and 10 if IDCy is specified; bits 13 and 14 receive the CPU ID).

- Internal bus bits 16 through 19 receive y, where y is any hexadecimal digit, 0 through F. The second least significant digit of the next firmware address (refer to subsection 3.5) must equal y.

These operands are optimized for constructing Megabus I/O control words for the CIP and SIP. They may be modified as required if it is necesssary to address other processors. Example 6 in subsection 3.7 shows the modification necessary to address the WCS.

3.2.2.5 Other Internal Bus Sources

The other internal bus sources are:

- Control panel
- H register
- Indicator register (I)
- Trap Status Z-word
- Status register (S)
- P register
- Y register
- MMU physical address
- HEX decoder.

Control Panel

The internal bus can receive control and status information from the Level 6 control panel. If the control panel is specified as a soruce, the internal bus receives the following "request word".

0 3	4 7	8	9	10	11	12	13	14	15	16	17 19
H(12-15)	HEX DIGIT	RUN	9BDF	PLUS	PANOK	CDEF	WRITE	0	R/W	ABEF	LOW ORDER REGISTER SELECT DIGIT

NOTES

1. Bits 0 through 3 receive H register bits 12 through 15, bit 14 is always Zero.

2. Bits 4 through 7 receive the value of the hexadecimal key currently being depressed, Zero if no key is depressed. Bit 11 receives a Zero when any hexadecimal key has been actuated. Refer to subsection 3.4 for firmware acknowledgement and setting of this bit (FLOPS operand PANOK).

3. Bits 12, 16, and 9 receive the least significant 3 bits of the leftmost register selection hexadecimal digit. Bits 17 through 19 receive the least significant 3 bits of the rightmost register selection digit.

4. The remaining 4 bits receive mode information. Bit 8 is Zero only if the panel RUN indicator is illuminated. Bit 10 is Zero only if the panel PLUS indicator is illuminated. Bit 13 is One only if the panel WRITE indicator is illuminated. Bit 15 is One only if either the panel WRITE or READ indicator is illuminated.

For example, assume the internal bus receives the following bit pattern from the control panel:

0000 1110 1110 1001 0110

then:

H register bits 12 through 15 are Zero.

The "E" key is currently being depressed and has not yet been acknowledged.

Register D6 has been selected.

The panel is in Read mode, and is not in Plus or Run mode.

H Register

The H register is a 16-bit register primarily used for swapping halfwords (bytes). There are two ways the internal bus can receive the H register. Either the two halves of the H register can be interchanged, i.e., internal bus bits 0 through 3 are forced to Zero, bits 4 through 11 receive H(8-15), and bits 12 through 19 receive H(0-7), or the left half of the H can be sign-extended, i.e., internal bus bits 0 through 3 are forced to Zero, bits 4 through 11 receive eight copies of H(0), and bits 12 through 19 receive H(0-7).

Indicator Register (I)

The I register is 8 bits wide and contains the seven software-visible indicator bits. Bits 4 through 11 of the internal bus receive the I register. The other 12 bits are forced to Zero.

Trap Status Z-Word

The trap status Z-word consists of pertinent hardware information that is delivered to the Memory Trap Save Area (TSA) when an exception condition is detected in the CPU. The information for the Z-word is brought together from several areas to serve as a source to the internal bus.

When the Z-word is specified as the internal bus source, internal bus bits 8 through 11 receive the XB register, bits 12 and 13 receive S register bits 1 and 2 (ring number), bits 16 through

19 receive the CTR counter, and bit 4 receives a Zero unless F register bit 0 is a Zero, or unless F register bits 9 through 11 = 101 and the SEL register does not contain Zero. All other bits are forced to Zero.

Status Register (S)

The S register contains the software-visible system status and security codes. When this register is an internal bus source, either internal bus bits 4 through 19 receive the S register content with bits 0 through 3 forced to Zeros, or bit 0 through 11 and bit 13 receive Zero, bit 12 receives a One and bits 14 through 19 receive S(10-15). S(10-15) is also called Level (LVL).

P Register

The P register is a 20-bit software-visible counter that is primarily used to address memory during procedural reads (refer to subsection 3.3).

Y Register

The Y register is a 20-bit counter primarily used to supply addresses to the Megabus during non-procedural reads and writes (refer to subsection 3.3).

MMU Physical Address

When this source is chosen, the internal bus receives the 20-bit physical address calculated by the MMU during the previous firmware step. MMU implies Y as an internal bus destination. Refer to subsection 3.3.3 (MMUSELECT).

Hexadecimal Decoder

The hexadecimal decoder logic produces a 16-bit mask, consisting of 15 One's surrounding a Zero in the bit location corresponding to the 4-bit value in the XB register. Internal bus bits 4 through 19 receive the 16-bit mask; bits 0 through 3 are forced to Zeros.

Table 3-3 Internal Bus Sources (Sheet 1 of 4)

MNEMONIC	CATEGORY	DATA RECEIVED BY INTERNAL BUS
ALU	RALU	ALU Result
B0 through B7	RALU	Base Register B0 through B7 (register file locations 8-F)
BB	RALU	Base Register B0 through B7 specified by SEL(1-3)
BB3	RALU	Base Register B0 through B3 specified by SEL(2-3)
BBE	RALU	Even Base Register B0, B2, B4, B6 specified by SEL(1-2)
BM	RALU	Base Register B0 through B7 specified by F(9-11)
BM3	RALU	Base Register B0 through B3 specified by F(10-11)
BME	RALU	Even Base Register B0, B2, B4, B6 specified by F(9-10)
BN	RALU	Base Register B0 through B7 specified by F(1-3)
BN3	RALU	Base Register B0 through B3 specified by F(2-3)
BNE	RALU	Even Base Register B0, B2, B4, B6 specified by F(1-2)
D0 through D7	RALU	Data Register D0 through D7 (register file locations 0-7)
DB	RALU	Data Register D0 through D7 specified by SEL(1-3)
DB3	RALU	Data Register D0 through D3 specified by SEL(2-3)
DBE	RALU	Even Data Register D0, D2, D4, D6 specified by SEL(1-2)
DM	RALU	Data Register D0 through D7 specified by F(9-11)
DM3	RALU	Data Register D0 through D3 specified by F(10-11)

Table 3-3 Internal Bus Sources (Sheet 2 of 4)

MNEMONIC	CATEGORY	DATA RECEIVED BY INTERNAL BUS
DME	RALU	Even Data Register D0, D2, D4, D6 specified by F(9-10)
DN	RALU	Data Register D0 through D7 specified by F(1-3)
DN3	RALU	Data Register D0 through D3 specified by F(2-3)
DNE	RALU	Even Data Register D0, D2, D4, D6 specified by F(1-2)
REGSEL	RALU	Register File location specified by SEL(0-3)
L4	SRCMOD	BI(0-3) ← SRC(0-3) BI(4-15) ← 0 BI(16-19) ← SRC(0-3) where SRC must be a source from the RALU category
R8	SRCMOD	BI(0-3) ← 0 BI(4-11) ← H(8) BI(12-19) ← SRC(12-19) where SRC must be a source from the RALU category
M1 through M7	RAM	RAMx where x = 1-7
MB	RAM	RAM0 through RAM7 specified by SEL(1-3)
MB3	RAM	RAM0 through RAM3 specified by SEL(2-3)
MBE	RAM	Even RAM location (RAM0, RAM2, RAM4, or RAM6), specified by SEL(1-2)
MM	RAM	RAM0 through RAM7, specified by F(9-11)
MM3	RAM	RAM0 through RAM3, specified by F(10-11)
MME	RAM	Even RAM location (RAM0, RAM2, RAM4, or RAM6), specified by F(9-10)
MN	RAM	RAM0 through RAM7, specified by F(1-3)
MN3	RAM	RAM0 through RAM3, specified by F(2-3)
MNE	RAM	Even RAM location (RAM0, RAM2, RAM4, or RAM6), specified by F(1-2)

Table 3-3 Internal Bus Sources (Sheet 3 of 4)

MNEMONIC	CATEGORY	DATA RECEIVED BY INTERNAL BUS
RAM0 through RAMF	RAM	RAMx where x = 0-F
RAMSEL	RAM	RAM0 through RAMF, specified by SEL(0-3)
BD	BUS	BI(0-3) ← 0 BI(4-19) ← BD
BDH	BUS	BI(0-3) ← H(12-15) BI(4-19) ← BD
BP	BUS	BI(0-3) ← 0 BI(4-19) ← BP
BPH	BUS	BI(0-3) ← H(12-15) BI(4-19) ← BP
RUP	BUS	BI(0-3) ← 0 BI(4-19) ← RUP
IDC0 through IDCF	CONSTANT	BI(0-3) ← 0 BI(4) ← Y(4) BI(5-12) ← 00000010 BI(13-14) ← S(8-9) BI(15) ← 0 BI(16-19) ← Y where y = 0-F
IDS0 through ISDF	CONSTANT	BI(0-3) ← 0 BI(4) ← Y(4) BI(5-12) ← 00000001 BI(13-14) ← S(8-9) BI(15) ← 0 BI(16-19) ← Y where Y = 0-F
K0--	CONSTANT	BI(0-11) ← 0 BI(12-19) ← unrestricted
KF--	CONSTANT	BI(0-3) ← 0 BI(4-11) ← FF BI(12-19) ← unrestricted
K--0 through K--F	CONSTANT	BI(0-3) ← 0 BI(4-15) ← unrestricted BI(16-19) ← z where z = 0-F

Table 3-3 Internal Bus Sources (Sheet 4 of 4)

MNEMONIC	CATEGORY	DATA RECEIVED BY INTERNAL BUS
Numeric Value =xyz#	CONSTANT	0xxyz where x = 0 or F y = 0 through F z = 0 through F
H	OTHER	BI(0-3) ← 0 BI(4-11) ← H(8-15) BI(12-19) ← H(0-7)
HL8	OTHER	BI(0-3) ← 0 BI(4-11) ← H(0) BI(12-19) ← H(0-7)
I	OTHER	BI(0-3) ← 0 BI(4-11) ← I BI(12-19) ← 0
LVL	OTHER	00080# + S(10-15)
MMU	OTHER	Physical address (implies Y as internal bus destination).
P	OTHER	P
PANEL	OTHER	BI(0-3) ← H(12-15) BI(4-19) ← Panel Request Word
S	OTHER	BI(0-3) ← 0 BI(4-19) ← S
XBHEX	OTHER	BI(0-3) ← 0 BI(XB+4) ← 0 other 15 bits receive One's
Y	OTHER	Y
Z	OTHER	BI(0-3) ← 0 BI(4) ← f(F,SEL) BI(5-7) ← 0 BI(8-11) ← XB BI(12-13) ← S(1-2) BI(14-15) ← 0 BI(16-19) ← CTR where: f(F,SEL) = $\overline{F(0)}$ F(9) $\overline{F(10)}$ F(11) (SEL(0) V SEL(1) V SEL(2) V SEL(3)

3.2.3 Internal Bus Destinations

Internal bus destinations are divided into four categories: (1) Megabus address registers, (2) RAM locations, (3) indicator register, and (4) other destinations. These destinations are described below and are summarized in Table 3-4. At most, one register from each category may be selected simultaneously, except as noted.

3.2.3.1 Megabus Address Registers

The Megabus address registers that are available as internal bus destinations include:

- P register
- Y register

P Register

The P register may be loaded with the 20 bits from the internal bus.

CAUTION

This register is software visible.

Y Register

The Y register may be loaded from the internal bus in three ways. First, the entire 20 bits of the internal bus can be copied to Y. Second, bits 4 through 19 can be copied to the corresponding bits of the Y register, leaving Y(0-3) unchanged. Third, the two-bit CPU ID, S(12-13), can be substituted for internal bus bits 10 and 11, and the 20-bit result copied to the Y register.

3.2.3.2 RAM Locations

When a RAM location is specified as a destination, the entire 20 bits of the internal bus are copied into the specified location. A RAM location may be specified either explicitly or as a function of fields in the F/SEL instruction registers. If the firmware step just previously executed altered the contents of SEL, those operands that depend on fields in SEL will use its previous contents (i.e., the contents before SEL was altered at the end of the firmware step just previously executed). If a RAM location is specified as an internal bus source, none can be simultaneously specified as a destination.

If, simultaneously, a register file location is used as a restricted selection source (refer to subsection 3.1.2) for the microprocessor and/or the internal bus, any RAM location referenced must correspond, in the sense indicated below, to the restricted-selection register-file source:

- D0 :: RAM0
- D1-D7 ::RAM1-RAM7 = M1-M7
- B0-B7 :: RAM8-RAMF
- REGSEL :: RAMSEL
- DB :: MB
- DB3 :: MB3
- DBE :: MBE
- DM :: MM
- DM3 :: MM3
- DME :: MME
- DN :: MN
- DN3 :: MN3
- DNE :: MNE

3.2.3.3 Indicator Register (I)

When the I register is specified as a destination, I(0) receives internal bus bit 12; I(2-7) receives bits 14 through 19. I(1) is always Zero. Individual bits of this register can also be modified by microinstructions in the miscellaneous hardware area (refer to subsection 3.4). When the I register is a destination, the internal bus source must be from the microprocessor area, the RAM, or the Megabus buffers.

3.2.3.4 Other Destinations

Other internal bus destinations include:

- Control Panel
- Status Register
- LINK Register
- H Register
- Instruction Registers.

Control Panel

The internal bus contents can be stored in the control panel displays. One operand causes internal bus bits 4 through 19 to be stored in the rightmost four hexadecimal digits. Another operand causes internal bus bits 16 through 19 to be stored in the leftmost hexadecimal digit of the display.

Status Register (S)

Internal bus bits 5 and 6 can be copied into the ring number field, S(1-2), or internal bus bits 14 through 19 can be loaded into the level field, S(10-15). If F(5) is Zero when the level field is loaded, the interrupt busy flop is cleared.

LINK Register

Internal bus bits 11 through 18 can be copied into the 8-bit LINK register. The LINK register is referenced by the firmware sequencing area.

H Register

Internal bus bits 4 through 19 can be copied into the 16-bit H register. SEL may be specified as a simultaneous destination.

Instruction Registers (F and SEL)

The F and SEL registers can be loaded from the internal bus in three ways. First, internal bus bits 4 through 15 can be copied into the 12-bit F register and bits 16 through 19 into the 4 bit SEL register. Second, internal bus bits 12 through 15 can be copied into F(8-11) and bits 16 through 19 into SEL. Third, internal bus bits 16 through 19 can be copied into the SEL register. In the last case, the H register may be specified as a simultaneous destination.

Table 3-4 Internal Bus Destinations (Sheet 1 of 2)

MNEMONIC	CATEGORY	ACTION
P	MEGABUS	P ← BI
Y	MEGABUS	Y ← BI
YR16	MEGABUS	Y(4-19) ← BI(4-19)
YRELOC	MEGABUS	Y(0-9) ← BI(0-9) Y(10-11) ← S(12-13) Y(12-19) ← BI(12-19)
M1-M7	RAM	Mx ← BI(0-19) where x = 1 to 7
MB	RAM	RAM0 through RAM7, specified by SEL(1-3), ← BI
MB3	RAM	RAM0 through RAM3, specified by SEL(2-3), ← BI
MBE	RAM	Even RAM register (RAM0, RAM2, RAM4, or RAM6), specified by SEL(1-2), ← BI
MM	RAM	RAM0 through RAM7, specified by F(9-11), ← BI
MM3	RAM	RAM0 through RAM3, specified by F(10-11), ← BI
MME	RAM	Even RAM register (RAM0, RAM2, RAM4, or RAM6), specified by F(9-10), ← BI
MN	RAM	RAM0 through RAM7, specified by F(1-3), ← BI
MN3	RAM	RAM0 through RAM3, specified by F(2-3), ← BI

Table 3-4 Internal Bus Destinations (Sheet 2 of 2)

MNEMONIC	CATEGORY	ACTION
MNE	RAM	Even RAM register (RAM0, RAM2, RAM4, or RAM6), specified by F(1-2), ← BI
RAM0-RAMF	RAM	RAMx ← BI where x = 0 to F
RAMSEL	RAM	RAM0 through RAMF, specified by SEL, ← BI
I	I	I(0) ← BI(12) I(1) ← 0 I(2-7) ← BI(14-19)
F	OTHER	F ← BI(4-15) SEL ← BI(16-19)
FR8	OTHER	F(8-11) ← BI(12-15) SEL ← BI(16-19)
H	OTHER	H ← BI(4-19)
LINK	OTHER	LINK ← BI(11-18)
LVL	OTHER	S(10-15) ← BI(14-19) if F(5) = 0, INTBSY ← 0
PANEL	OTHER	4 Least Significant Display Digits ← BI(4-19)
PANEL4	OTHER	Most Significant Display Digit ← BI(16-19)
RING	OTHER	S(1-2) ← BI(5-6)
SEL	OTHER	SEL ← BI(16-19)

3.2.4 Internal Bus Examples

The following are examples of source statements for the internal bus area.

STATEMENT	MEANING
BI ALU,R8	BI(0-3) ← 0 BI(4-11) ← H(8) BI(12-19) ← ALU result (12-19)
BI RAMC,P	P ← BI ← BAMC
BI BP,F	BI(0-3) ← 0 BI(4-19) ← BP F ← BI(4-15) SEL ← BI(16-19)
BI FC0#,M6	M6 ← BI ← OFFC0#

3.3 MEGABUS INTERFACE AREA

The following activities may occur on the Megabus:

- Memory Write
- I/O (non-memory) Write
- Memory Read Request
- I/O (non-memory) Read Request
- Read Response
- Interrupt

To avoid Megabus bottlenecks, read requests are considered complete when they have been accepted (or rejected) by the addressed unit, and a separate Megabus cycle is used for the read response. Various kinds of read requests originating in the CPU are "tagged" so that the corresponding responses will be delivered to the appropriate buffer (BD, BP).

The microinstructions in the Megabus interface area (see Figure 3-3) are designed to request reads, perform writes, and maintain associated address registers and flops. In general, the operation of the cache and Memory Management Unit (MMU) is transparent; however, occasional firmware control is necessary.

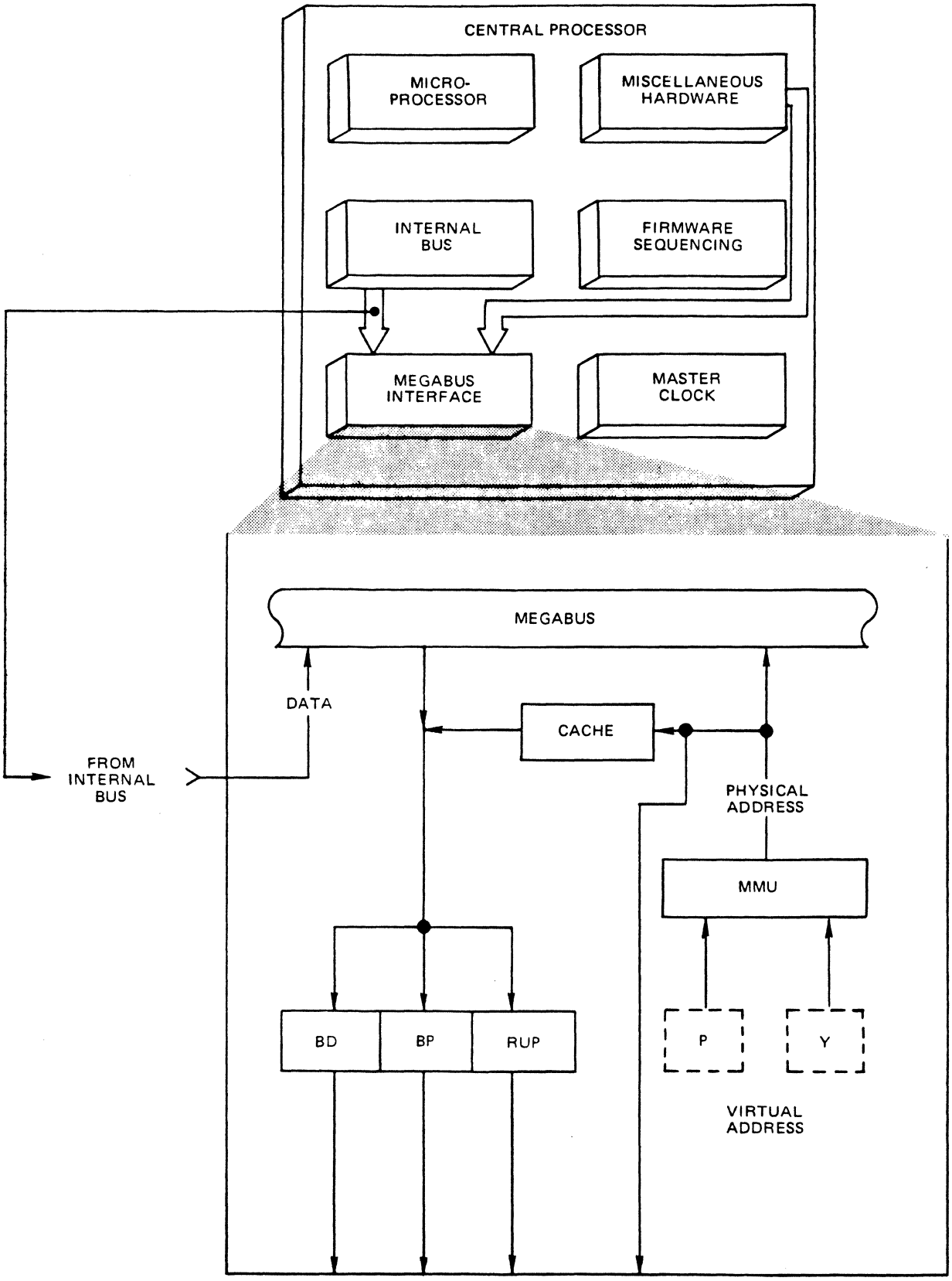


Figure 3-3 Megabus Interface Area

Perform Memory Write

When data is to be written to memory, the Y register is first made to contain the address of the desired word. If a byte rather than a word is to be written, XB register bit 0 (refer to subsection 3.4) is made to contain the byte offset (0 for left, 1 for right). The firmware may then perform the write, supplying the data to be written on the internal bus (bits 4 through 19 for word writes, bits 4 through 11 for left-byte writes, and bits 12 through 19 for right-byte writes). The MMU translates the virtual address in the Y register to the corresponding physical address and checks for "write" permission.

The Acknowledge (ACK) flop is set to One if the memory write is accepted and Zero if it is rejected. It may be copied into MISC and/or I(I), either simultaneously or anytime thereafter prior to the next Megabus read request or write operation. It may be tested, starting in the next firmware step, anytime prior to the next Megabus read request or write operation. A simultaneous test has undefined results. Almost all memory writes are accepted, so it is usually unnecessary to copy or test ACK.

Perform I/O (Non-Memory) Write

When data is to be written to I/O (non-memory) devices, the Megabus address lines must be made to contain the appropriate control word (10-bit channel number, 6-bit function code). To achieve this, Y register bits 5 through 19 must be made to contain the most significant 15 bits of the control word, and XB register bit 0 must be made to contain the least significant bit. The firmware may then perform the write. If the data to be sent is 16 bits wide, it is supplied on internal bus bits 4 through 19. If it is 21 bits wide (as is the case when supplying a device controller with a memory address), data bits 0 through 4 are supplied by Y bits 0 through 4 and data bits 5 through 20 are supplied by internal bus bits 4 through 19. The firmware is responsible for converting any virtual addresses to physical for the controller by using the MMU as an internal bus source. The MMU does neither translation nor checking during I/O writes.

The ACK flop is set to One if the write is accepted and Zero if it is rejected. It may be copied into MISC and/or I(I), either simultaneously or anytime prior to the next Megabus read or write operation. It may also be tested in the next firmware step or anytime thereafter prior to the next Megabus read request or write operation. A simultaneous test has undefined results. It is good practice on I/O writes to check ACK or copy it to I(I) for testing by the software.

Request Memory Read

Memory reads are called "procedural" when P supplies the address and "data" when Y supplies the address.

When a data read request is to be performed, the Y register is first made to contain the address of the desired word. The firmware may then send a read request to the memory subsystem. The MMU translates the virtual address in Y to the corresponding physical address and checks for "read" permission. The next (or any subsequent) firmware step may use BD as an internal bus source to retrieve the requested data. The ACK flop is set to One if the memory read request is accepted and Zero if it is rejected. ACK may be copied and tested in the next firmware step or anytime thereafter prior to the next Megabus read request or write operation. Almost all memory data read requests are accepted, so it is usually unnecessary to copy or test ACK.

Reading procedural words from memory is, from the firmware's viewpoint, simplified by the inclusion of extra hardware to expedite this common operation. Unlike data reads, it is never necessary to include an explicit procedure read request; the procedure buffer (BP) can be treated as always containing the memory word addressed by P. This is true because the hardware automatically performs read requests whenever necessary. The MMU translates the virtual address in P to the corresponding physical address and checks for "execute" permission. When BP is specified as an internal bus source, P is automatically incremented so as to point to the next procedural word. These automatic actions actions facilitate creation of compact firmware code for reading procedure. For higher performance, an explicit procedural read request may be specified in advance of the step which uses BP as an internal bus source. To facilitate the firmware coding of branch instructions, if an explicit procedural read request is specified simultaneously with Y as an internal bus source and P as an internal bus destination, the memory address is supplied by Y.

Request I/O (Non-Memory) Read

When an I/O (non-memory) read request is to be performed, the Megabus address lines must be made to contain the appropriate control word (10-bit channel number, 6-bit function code). To achieve this, Y register bits 5 through 9 must be made to contain the most significant 15 bits of the control word, and XB register bit 0 must be made to contain the least significant bit. The MMU does neither translation nor checking during I/O read requests. The next (or any subsequent) firmware step may use BD as an internal bus source to retrieve the requested data. The ACK flop is set to One if the read request is accepted and Zero if it is rejected. ACK may be copied and tested in the next firmware step or anytime thereafter prior to the next Megabus read request or write operation. It is good practice on I/O read requests to check ACK or copy it to I(I) for testing by the software.

3.3.1 Syntax

The microinstructions in this area specify the firmware control on the Megabus interface. These microinstructions contain a function with from zero to two operands as follows:

Function operand, operand

No Megabus interface microinstruction should be specified if the internal bus calls for BD, BDH, BP, BPH, P, Y, or MMU as a source, or for P, Y, YR16, or YRELOC as a destination, except as noted.

3.3.2 Megabus Interface Functions

The Megabus interface functions are described below and summarized in Table 3-5.

WRTWORD

WRTWORD performs a word write to memory or I/O. The only valid operand combinations are:

- CHGLOCK
- INCY
- I-O
- I-O, INCY.

The short form WRT is synonymous with WRTWORD. WRTWORD or WRT may be specified even when Y is an internal bus source.

WRTBYTE

WRTBYTE performs a byte write to memory or I/O. The only valid operand combinations are:

- CHGLOCK
- INCY
- I-O
- I-O, INCY.

WRTBYTE may be specified even when Y is an internal bus source.

RDREQP

RDREQP initiates a procedural read request. No operands may be specified. RDREQP may be specified even if Y is an internal bus source and P is an internal bus destination, in which case the memory address is specified in Y rather than P (refer to subsection 3.3.3 - PURGE). RDREQP may also be specified even when P is an internal bus source.

RDREQ

RDREQ initiates an I/O or memory data read request. The only valid operand combinations are:

- NORMAL
- I-O
- CHGLOCK
- NOCACHE
- CHGLOCK, NOCACHE.

RDREQ (with any of the above operand combinations) may be specified even when Y is an internal bus source.

BUS

BUS performs auxiliary operations when no read request or write operation is desired. The only valid operands for BUS are:

- YSELECT
- PSELECT
- MMUSELECT
- INCY
- INCP
- PURGE
- MMURDACC
- MMUWRACC.

3.3.3 Megabus Interface Operands

The Megabus interface operands (described below0 are used to modify the basic functions.

NORMAL

A memory data read request is performed.

I-O

A non-memory read request or write operation is performed. If this operand is omitted, any read request or write operation is directed to memory.

INCP

The P register is incremented at the end of the current firmware step. Note that when BP or BPH is specified as an internal bus source, P is automatically incremented. INCP may be specified even when P is an internal bus source.

INCY

The Y register is incremented at the end of the current firmware step. Even when BD, BDH, or Y is an internal bus source, INCY may be specified as a BUS operand.

PSELECT

If the previous firmware step initiated a procedural read request (RDREQP), and the current step does not use the requested data (by specifying BP or BPH as an internal bus source), then PSELECT (or INCP or PURGE) must be specified to ensure correct operation of the hardware. PSELECT may be specified even when P is an internal bus source.

YSELECT

If the previous firmware step initiated a non-procedural read request (RDREQ), and the current step does not use the requested data (by specifying BD or BDH as an internal bus source), then YSELECT (or INCY) must be specified to ensure correct operation of the hardware. Furthermore, no RDREQP should be performed in any step between an RDREQ and the step that uses the data. YSELECT may be specified even when Y is an internal bus source and/or when P is an internal bus destination.

PURGE

Notifies hardware that BP is obsolete. This notification is necessary only in the case where Y is to be copied to P (e.g., software jump) during a procedural read request (refer to subsection 3.3.1 - RDREQP). This action must be preceded by a BUS PURGE.

NOCACHE

The cache, if installed, is bypassed during the memory read. The only time it is necessary to use this operand is when the requested data is to be directly copied from BD to P (i.e., BI BDH,P).

CHGLOCK

Allows the implementation of selected software instructions that facilitate intercommunication and synchronizing in multiprocessor systems. The intent is to protect against overlap and consequent interference between "test-and-set" operations initiated independently by two or more CPU's on a common memory location.

Since the "lock" mechanism is implemented in the main memory subsystem, use of this argument unconditionally implies NOCACHE, (the latter argument, though redundant, may be included for clarity).

All other effects of CHGLOCK are conditional on the content of the F register. If F = 002, 003, 006, or 007, or if 880#≤F ≤897# or if 8A0#≤F≤ 8B7# the current instruction is classifed as a Read-Modify-Write (RMW) operation, and the interlock operates; otherwise, CHGLOCK has no further effect.

The operation of the interlock is dependent on CPU control flop RMWF, which is assumed to be initally cleared. The operand CHGLOCK causes the read request or write cycle on the Megabus to be accompanied by a Set-Lock code. The memory, if it is not already locked, accepts the transmission and the memory module becomes locked. If the memory had already been locked, it would have rejected the transmission (ACK = 0); CPU firmware, sensing the rejection, could repeat the request or take other appropriate action.

When the Set-Lock request is acknowledged, CPU hardware sets the RMWF control flop. As a result, the next CHGLOCK occurrence causes a Megabus cycle accompanied by an UNLOCK code. The memory unconditionally accepts the code, and the module involved reverts to (or remains in) the unlocked state.

It is imperative that any firmware algorithm which locks any memory module not be permitted to terminate without unlocking the same module. Failure to observe this requirement can produce unrecoverable system deadlocks, which can be very elusive to diagnose.

MMURDACC, MMUWRACC

Modifies the access rights checking performed in preparation for MMUSELECT (refer to subsection 3.3.3 - MMUSELECT). MMURDACC or MMUWRACC may be specified even when Y is an internal bus source.

MMUSELECT

It it is necessary to check the validity of a virtual memory addrress or the program's access rights to that location, the following actions are taken. The Y register is first made to contain the address. The virtual address is then transmitted to the MMU by specifying MMURDACC (for a read access check) or MMUWRACC (for a write access check) as a BUS operand. In the next firmware step, the physical address may be selected as an internal bus source (BI MMU). If the physical address is not needed and the internal bus can profitably be used for another purpose, then the microinstruction BI MMU can be replaced by BUS MMUSELECT. Simultaneously, the address validity can be tested using the condition IFADRER (refer to subsection 3.5) and any protection violation can be copied into control flop MISC (FLOPS MSPROV). If either BI MMU or BUS MMUSELECT is specified, the Y register will be loaded from the internal bus whether or not Y is specified as an internal bus destination.

Table 3-5 Megabus Interface Microinstructions
(Sheet 1 of 2)

MICROINSTRUCTION		MEANING
BUS	INCP	P ← P+1 CRT ← CTR+1
BUS	INCY	Y ← Y+1
BUS	MMURDACC	Map Y address and request read permission.
BUS	MMUSELECT	Make available SEGERR and PROV.
BUS	MMUWRACC	Map Y address and request write permission.
BUS	PSELECT	Finish preceding RDREQP.
BUS	PURGE	Refresh BP.
BUS	YSELECT	Finish preceding RDREQ.
RDREQ	NORMAL	Initial memory data read request.
RDREQ	CHGLOCK	Initate memory data read request. Set/clear lock in memory. Bypass cache.
RDREQ	I-O	Initiate non-memory data read request.
RDREQ	NOCACHE	Initiate memory data read request. Bypass cache.
RDREQP		Initiate procedural read request.
WRT	CHGLOCK	Perform memory word write. Clear/set lock in memory.
WRT	I-O	Perform non-memory word write.
WRT	I-O, INCY	Perform non-memory word write. Y ← Y+1
WRT	INCY	Perform memory word write. Y ← Y+1
WRTBYTE	CHGLOCK	Perform memory byte write. Clear/set lock in memory.
WRTBYTE	I-O	Perform non-memory byte write.

Table 3-5 Megabus Interface Microinstructions
(Sheet 2 of 2)

MICROINSTRUCTION	MEANING
WRTBYTE I-O, INCY	Perform non-memory byte write. Y ← Y+1
WRTBYTE INCY	Perform memory byte write. Y ← Y+1
WRTWORD CHGLOCK	Perform memory word write. Clear/set lock in memory.
WRTWORD I-O	Perform non-memory word write.
WRTWORD I-O, INCY	Perform non-memory word write. Y ← Y+1
WRTWORD INCY	Perform memory word write. Y ← Y+1

3.4 MISCELLANEOUS HARDWARE AREA

The miscellaneous hardware area controls a number of auxiliary flops and registers. There are four categories of miscellaneous hardware:

- I - Indicator Register
- CTR - Counter Register
- MMU - MMU Controls
- GP - Other Hardware.

3.4.1 Syntax

The syntax of microinstructions in the miscellaneous hardware area is:

FLOPS operand,....,operand

where:

Each operand specifies a flop or register and the value used to load it.

Multiple operands may be used to specify simultaneous actions on the flops and registers.

3.4.2 Indicator Register (I) Bits

The 8 bits of the I register may be loaded from the internal bus bits of I using the FLOPS microinstruction. If one or more

operands in this category are used, the internal bus source, if any, must be from the microprocessor area, the RAM, or the Megabus buffers.

OV	-	C	B	I	G	L	U

The Overflow indicator (OV) may be set from:

- OVFL (ALU overflow signal, refer to subsection 2.1).
- Internal bus bits 4 and 5 (refer to Table 3-6 for exact function).

The Carry indicator (C) may be set from:

- CRY (ALU carry signal, refer to subsection 3.1).
- Internal bus bit 4.
- Internal bus bit 19.
- Q register bit 19 just prior to rightshift in this step.

The Bit indicator (B) may be set from:

- AUZ (ALU zero detect, refer to subsectgion 3.1).
- Internal bus bit 4.

The Input/Output indicator (I) may be set from the Megabus acknowledge signal (refer to subsection 3.3.).

The Greater Than indicator (G) may be set from:

- Internal bus bit 2 and AUZ (ALU zero detect) - refer to Table 3-6 for exact function.
- Complement of SIGN flop (refer to subsection 3.4.5).

The Less Than indicator (L) may be set from:

- Internal bus bit 4.
- ALU result bit 0.
- SIGN flop (refer to subsection 3.4.5).

The Unlike Signs indicator (U) may be set from internal bus bit 4.

3.4.3 Counter Register (CTR)

The 4-bit CTR counter is used by the native firmware to count the instruction length. It is made available to the internal bus when the trap status Z-word is an internal bus source. It is incremented each time BP is specified an an internal bus source (operands BP or BPH, refer to subsection 3.2), or INCP is specified in the BUS microinstruction (refer to subsection 3.3), unless a FLOPS operand specifies it is to be initialized to a Zero or One.

3.4.4 MMU Controls

The MMU controls modifying and restoring the MMU functionality as summarized below (refer to subsection 3.3).

- Calculate effective ring number: EFFRING is loaded with the less priviledged of the previous value of the effective ring number and the write permission ring number associated with the address Y (generally used during indirect references).

- Check procedure as data: NOCHEK is set to One, meaning that until further notice (see below), procedural read requests need only "read" permission, not "execute" permission.

- Suppression access rights checking: until further notice (see below), no access rights will be checked. Address mappiing and boundary checks are unaffected.

- Check data descriptor length: loads control flop DDLEQ0 to remember whether internal bus bits 8 through 12 are Zero.

- Initialize effective ring number: EFFRING is restored from S register (bits 1-2). NOPROC and NOCHEK are cleared to Zeros, restoring normal access rights checking.

- Validate range: test that the range specified on the internal bus, added to the address in the Y register, does not exceed the size of the segment defined by the latter address. The results of the test are reported in two testable flops, Segment Error (SEGERR) and Protection Violation (PROV); refer to subsection 3.3.3 - MMUSELECT.

3.4.5 Other Hardware

The other flops and registers, and the functions from which they can be set are as follows:

LOAD Flop

The LOAD flop is normally set and cleared by the control panel. It can also be set and cleared by firmware. If LOAD is set, the CPU will not automatically cause a trap if it detects an unavailable resource.

MISC Flop

The MISC flop may be set from:

- Complement of internal bus bit 19.

- Internal bus bits 4 through 9 equal to Zero.

- CRY - ALU Carry signal (refer to subsection 3.1).
- ACK - Megabus Acknowledge signal (refer to subsection 3.3).
- PROV - MMU Protection Violation signal (refer to subsection 3.3).
- Zero
- One

PANOK Flop

The PANOK flop is cleared by the control panel hardware when a hexadecimal keypad pushbutton is being depressed; it must be set by firmware when the control panel request has been serviced. When PANEL is an internal bus source (refer to subsection 3.2), a copy of this flop appears as internal bus bit 11.

SHIN1 Flop

The SHIN1 flop may be set from:

- I(B) - I register Bit Test indicator
- Zero
- One.

SHIN2 Flop

The SHIN2 flop may be set from:

- Complement of SIGN flop
- Zero
- One.

SIGN Flop

The SIGN flop may be set from:

- Internal bus bit 0
- Internal bus bit 4
- Internal bus bit 19
- One
- Zero (by using the firmware sequencing condition IFRPTR).

TRAFFIC Flop

The TRAFFIC flop may be set from the complement of the ZERO flop.

WRAP Flop

The WRAP flop may be set from the inequality of the ALU Carry signal (CRY) and the SIGN flop.

XB Register

The XB register may be cleared to Zero or may be shifted right by one bit, receiving either 0, 1, or bit 19 of the ALU result.

ZERO Flop

The ZERO flop may be set from:

- AUZ - ALU zero detect signal (refer to subsection 3.1).
- QLT active flop from control panel - equals One only if the last CPU Quality Logic Test (QLT) execution failed.
- Zero.
- One.

3.4.6 FLOPS Operands and Restrictions

The FLOPS operands are specified in Table 3-6. In general, only one operand from each of the four categories may be specified except as noted. When a specific operand implies other operands, it is recommended that the implied operands be coded explicitly, to improve listing clarity.

Table 3-7 lists the only legal combinations of internal bus destinations (refer to subsection 3.2) and GP category FLOPS operands. For example, if H is an internal bus destination, then SGBI4 must be specified, and no other GP category operands are permitted. If none of the listed internal bus destinations are specified, then SGBI4 may be used by itself or in combination with several other GP category operands, as shown. If an incomplete combination is specified in the source code, the WCS Assembler might choose a combination with undesired side effects.

All operands which are functions of the ALU signals AUZ, OVFL, and CRY may be used simultaneously only with other operands which force the same length (16 or 20 bits). Thus, ICRY20 may be used with ZRAUZ20 but not ZRAUZ. This restriction also applies to test conditions (refer to subsection 3.5.4) which are functions of AUZ, OVFL, or CRY.

Table 3-6 Miscellaneous Hardware Operands (Sheet 1 of 4)

OPERAND	CATEGORY	ACTION	NOTES AND RESTRICTIONS
IACK	I	I(I)← Megabus ACK flop	
IBBI4	I	I(B)← BI(4) I(C)← CRY I(OV)← OVFL	
IBNAZ	I	I(B)← $\overline{AUZ}$	Refer to Note 2a.
IBNAZ20	I	I(B)← $\overline{AUZ}$	Refer to Note 2b.
ICBI4	I	I(C)← BI(4)	
ICBI19	I	I(C)← BI(19)	
ICQSR	I	I(C)← Q(19)	Microprocessor must specify right shift.
ICRY	I	I(C)← CRY	May also specify IOVFL w/wo IBBI4. Refer to Note 2A.
ICRY20	I	I(C)← CRY	May also specify IOVFL w/wo IBBI4. Refer to Note 2b.
IGL	I	I(G)← $\overline{BI(4) \vee AUZ}$ I(L)← BI(4)	
IGL20	I	I(G)← $\overline{ALU\ result\ (0) \vee AUZ\ (20\ bits)}$ I(L)← ALU result (0)	Refer to Note 2b.
IGLU	I	I(G)← $\overline{SIGN}$ I(L)← SIGN I(U)← BI(4)	
IO4NE5	I	I(OV)← BI(4) ⊻ BI(5)	
IOVFL	I	I(OV)← OVFL	

Table 3-6 Miscellaneous Hardware Operands (Sheet 2 of 4)

OPERAND	CATEGORY	ACTION	NOTES AND RESTRICTIONS
CTR0	CTR	CTR ← 0	1. Implies INCP as a BUS operand unless BP or BPH is internal bus source. 2. Implies XBSR unless XBSR0, XBSR1, or XB0 is specified. 3. Bit 1 of the 11-bit branch address must equal Zero.
CTR1	CTR	CTR ← 1	1. Implies INCP as a BUS operand unless BP or BPH is internal bus source. 2. Implies XBSR unless XBSR0, XBSR1, or XB0 is specified. 3. Bit 1 of the 11-bit branch address must equal One.
DDLEQJ	MMU	Check data descriptor length. Calculate effective ring number.	Implies microprocessor function XORC.
NOCHEK	MMU	Suppress access rights checking.	1. Implies RINGCALC unless microprocessor function INCR, ADD1, AND, or SUB specified. 2. No FLOPS operand in CTR or GP groups may be specified.
NONPROC	MMU	Check procedure as data.	Implies microprocessor function ANDC.
RINGCALC	MMU	Calculate effective ring number.	Implies microprocessor function XOR.
RINGINIT	MMU	Initialize effective ring number. Check data descriptor length.	Implies microprocessor function OR unless COPY (with non ZERO SRC1) is specified.

Table 3-6 Miscellaneous Hardware Operands (Sheet 3 of 4)

OPERAND	CATEGORY	ACTION	NOTES AND RESTRICTIONS
VALID8	MMU	Validate range.	1. Implies microprocessor function SUB unless DDLEQ0, NONPROC, or RINGINIT is also specified. 2. No FLOPS operand in CTR or GP groups may be specified.
LOAD0	GP	LOAD ← 0	
LOAD1	GP	LOAD ← 1	
MS0	GP	MISC ← 0	
MS1	GP	MISC ← 1	
MS4-9EQ0		MISC ← $\overline{BI(4) \vee BI(5) \vee BI(6) \vee BI(7) \vee BI(8) \vee BI(9)}$	
MSACK	GP	MISC ← Megabus ACK flop	
MSCRY	GP	MISC ← CRY	Refer to Note 2a.
MSCRY20	GP	MISC ← CRY	Refer to Note 2b.
MSNBI19	GP	MISC ← BI(19)	
MSPROV	GP	MISC ← MMU protection violation signal	
PANOK	GP	PANOK ← 1	
SG1	GP	SIGN ← 1	
SGBI0	GP	SIGN ← BI(0)	
SGBI4	GP	SIGN ← BI(4)	
SGBI19	GP	SIGN ← BI(19)	
SH00	GP	SHIN1 ← 0 SHIN2 ← 0	
SH01	GP	SHIN1 ← 0 SHIN2 ← 1	

Table 3-6 Miscellaneous Hardware Operands (Sheet 4 of 4)

OPERAND	CATEGORY	ACTION	NOTES AND RESTRICTIONS
SH10	GP	SHIN1 ← 1 SHIN2 ← 0	
SH11	GP	SHIN1 ← 1 SHIN2 ← 1	
SH1IB	GP	SHIN1 ← I(B)	
SH2NSG	GP	SHIN2 ← $\overline{\text{SIGN}}$	
TRAFNZR	GP	TRAFFIC ← $\overline{\text{ZERO}}$	
WRAP	GP	WRAP ← SIGN ∀ CRY	Refer to Note 2b.
XBO	GP	XB ← 0	
XBSR	GP	XB(0) ← ALU result (19) XB(1-3) ← XB(0-2)	If right shift not specified in micro-processor area, results are undefined. Refer to Note 2a.
XBSR0	GP	XB(0) ← 0 XB(1-3) ← XB(0-2)	Refer to Note 2b.
XBSR1	GP	XB(0) ← 1 XB(1-3) ← XB(0-2)	Refer to Note 2b.
ZR0	GP	ZERO ← 0	
ZR1	GP	ZERO ← 1	
ZRAUZ	GP	ZERO ← AUZ	Refer to Note 2a.
ZRAUZ20	GP	ZERO ← AUZ	
ZRQLT	GP	ZERO ← QLT active flop	

NOTES

1. For restrictions and permissible combinations in the GP category, refer to Table 3-7.

2a. Forces AUZ, CRY, and OVFL to 16-bit versions.

2b. Forces AUZ, CRY, and OVFL to 20-bit versions (refer to subsection 3.1.2).

Table 3-7 Permissible GP Combinations
(Sheet 1 of 2)

DESTINATION(S) OF INTERNAL BUS	GP CATEGORY OPERANDS
PANEL	None
PANEL4	None
LVL	None
RING	None
LINK	None
H,SEL	None
H	SGBI4
SEL	SH11
FR8	SG1
FR8	SGBI19, MS4-9EQ0
F	XB0
F	XB0, MS0, SH00, ZRAUZ
F	SB0, MS0, SH00, ZRAUZ20
None of the above	SGBI4, MSNBI19, ZRAUZ
None of the above	SGBI4, MSNBI19, ZRAUZ20
None of the above	SGBI4, SH2NSG
None of the above	SGBI4, MS1
None of the above	SBGI19, MS1
None of the above	SGBI19, ZRQLT
None of the above	SGBI4, XBSR
None of the above	SGBI4, XBSR0
None of the above	SGBI4, XBSR1
None of the above	ZR0, XBSR
None of the above	ZR0, XBSR0

Table 3-7 Permissible GP Combinations
(Sheet 2 of 2)

DESTINATION(S)	GP CATEGORY OPERANDS
None of the above	ZR0, XBSR1
None of the above	ZR1, XBSR
None of the above	ZR1, XBSR0
None of the above	ZR1, XBSR1
None of the above	SH00, XBSR
None of the above	SH00, XBSR0
None of the above	SH00, XBSR1
None of the above	SH10, XBSR
None of the above	SH10, XBSR0
None of the above	SH10, XBSR1
None of the above	Any single GP-category operand, except ZRQLT, XBO, MSNBI19, or MS4-9EQ0

3.4.7 Miscellaneous Hardware Examples

The following are examples of source statements for the miscellaneous hardware area.

STATEMENT	MEANING
FLOPS IGL20, CTR0, NONPROC, XBSR	
FLOPS IOVFL, ICRY	
FLOPS IACK, IBBI4	Invalid - cannot choose to operands from same category except as noted.
FLOPS NOCHEK, CTR1	Invalid - NOCHEK incompatible with operands from CTR and GP categories.
FLOPS NOCHEK, NONPROC	Invalid - NOCHEK and NONPROC require different microprocessor functions.

STATEMENT	MEANING
FLOPS SH00	
FLOPS SH00, MS0	Incomplete specification - may result in undesired side effects.
FLOPS SH00, MS1	Invalid - illegal combination of GP category operands.

3.5 FRIMWARE SEQUENCING AREA

The firmware sequencing area generates the address of the next firmware step to be executed. Both conditional and unconditional branching are supported.

3.5.1 Transparent and Sequential Mode Differences

The two firmware sequencing modes supported by WCS, Transparent and Sequential, are mutually exclusive. The WCS option hardware must be set by a manually operated switch, into one mode or the other. The following discussion assumes that the setting of this switch is established when the system is installed, and that all user firmware intended for one installation will execute in the same mode. The assembler must be told which mode applies to the firmware being processed (refer to Section four), to produce object code consistent with this mode, as well as appropriate diagnostics when the restrictions applicable to this mode are violated.

The obvious difference between the two modes appears in the assignment of control store addresses to successive steps of a firmware routine. In this respect, Sequential mode looks more like typical software, and hence will seem more familiar to most readers. The microporgrammer assigns an initial address (origin), after which the assembler automatically increments the address for each new step. Conditional branches represent a choice between continuing in sequence and taking some other action (branch to a specified location, or "call" a subroutine, or "return").

In contrast, the microprogrammer working with transparent code will rarely allow the assembler to assign the address of the next step, even when the values are adjacent. Every step in Transparent mode explicitly specifies the address of its successor, which may equally reside anywhere in the 2048-location firmware bank. Conditional branches represent a choice between the address thus specified and an alternate address (produced either by a fixed modification of the specified address, or by reference to the LINK register, or by hardware dedicated to analyis of the F/SEL instruction register).

In either mode, when no branching is required, the assembler generates the necessary code to progress from step to step. Nevertheless, it should be noted that, in transparent object code, the bits responsible for specifying the next address are occupied at almost every step, whereas in sequential object code, these bits are unused except where branching is called for. This destinction is important because some of these same bits are involved in the generation of constants (refer to subsection 3.2.2.4 and 3.7) by the firmware.

Therefore, when a particular constant needs to be generated, a restriction is imposed on the value of the next-address bits. In Transparent mode, this restriction requires some extra bookkeeping to keep track of address assignments. In Sequential mode, the restriction vanishes when no simultaneous branching is involved; otherwise, it usually requires insertion of an extra firmware step.

Thus, the choice between the two modes should be based on the expected frequencies of branching and of constant usage, and the consequent likelihood of their interfering with each other.

Secondary considerations include: (1) the relatively greater ease of subroutine calling in Sequential mode, (2) the greater freedom with which three or more decision steps can select between the same two destinations in Transparent mode, and (3) the fact that the native CPU firmware is written and executed in Transparent mode. In summary, Transparent mode makes it possible to produce more compact code, whereas Sequential mode is easier to learn.

3.5.2 Transparent Mode Syntax

The Transparent mode firmware sequencing microinstruction takes one of the following forms:

```
GOTO        addr

condition   true, false
```

where:

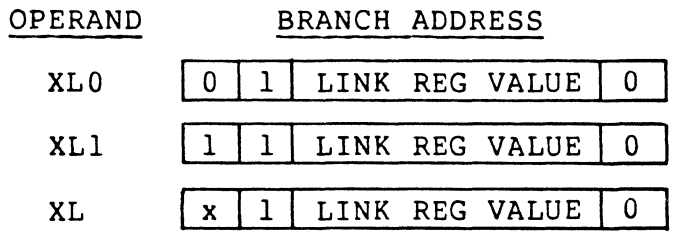
addr is the unconditional-branch address.

true is the branch address if condition is true.

false is the branch address if condition is false.

In Transparent mode, the branch address operands (true, false, and addr) may be an address value (refer to subsection 3.5.5) or a reserved word (LINK register operand or "splatter" operand). An address value is used if the branch address is a fixed location. A LINK register operand is used to specify the 11-bit branch address as a function of the 8-bit LINK register

(refer to subsection 3.2.4) as shown in Figure 3-4. A splatter operand is used to generate a branch address based on the value of the F/SEL instruction register (refer to Appendix E). The legal Transparent mode address operands are summarized in Table 3-8).



x = high order bit of alternate branch address (One if none)

Figure 3-4 Link Register Operands

The only valid unconditional branch is GOTO, where control is transferred to the specified branch address. In conditional branching, either the "true" or "false" operand must specify a value or reserved word, and the other must specify a value or be null. If this latter operand is null it represents the address of the next firmware statement in the source. If neither operand is specified as a reserved word, one of the addresses must be equal to the other address ORed with 3.

Table 3-8 Transparent Mode Branch Address Operands

OPERAND	GENERATED BRANCH ADDRESS
value	Specified address **
XL	Function of LINK register and of alternate branch address.
XL0	Function of LINK register (most significant bit = 0).
XL1	Function of LINK register (most significant bit = 1).
XA*	Fixed location based on F/SEL (Address Syllable).
XB*	Fixed location based on F/SEL (Commercial Address Syllable).
XE*	Fixed location based on F/SEL (Execute).
XR*	Fixed location based on F/SEL (Read Operand Data).
XW*	Fixed location based on F/SEL (Write Result).
XF	Fixed location = 020# (Fetch Instruction).

*F may not be loaded (refer to subsection 3.2) simultaneously with specifying these branch address operands (refer to Appendix E for maps of locations addressed as functions of F/SEL).

**If internal bus source is a constant (refer to subsection 3.2.2.4), the second least significant hexadecimal digit of "value" must equal digit "y" of the constant.

3.5.3 Sequential Mode Syntax

The Sequential mode firmware sequencing microinstruction takes one of the following forms:

```
GOTO        addr

CALL        addr

LBRANCH     addr

RETURN

Condition   true

Condition   true,RETURN

Condition   true,,CALL
```

```
Condition    ,false

Condition    RETURN,false

Condition    ,false,CALL
```

where:

addr is the unconditional branch address.

true is the branch address if the test condition is true.

false is the branch address if the test condition is false.

Unconditional branches in Sequential mode include:

- GOTO - Control is transferred to the specified branch address.
- CALL - Control is transferred to the specified branch address after saving the address of the current firmware step plus one in the "return-address" register. Nested calls are not supported.
- RETURN - Control is transferred to the firmware step pointed to by the return address register.
- LBRANCH - Control is transferred to one of 256 locations based on the three most significant bits of the 11-bit operand "addr", concatenated with the LINK register value.

The operand "addr" must be an address value (refer to subsection 3.5.6) greater than 001. If the internal bus source is a constant (refer to subsection 3.2.2.4), the second least significant digit of the "value" must equal digit y of the constant.

In Sequential mode conditional branching, either the "true" or "false" operand must specify an address value, and the other must be null or the reserved word RETURN. If null is used, a conditional CALL may be specified by using the three operand format.

3.5.4 Conditions

Table 3-9 summarizes the list of permissible test conditions, divided into six categories:

- Microprocessor Tests
- Internal Bus Tests
- Instruction Register (F/SEL) Tests
- Megabus Interface Tests
- Miscellaneous Hardware Tests
- Other Tests

Conditions that test a single bit or flop (or a logical function of several) result in "true" if the value is One and "false" if the value is Zero. Conditions that test a relational expression (e.g., F(1-3) = 7) result in "true" if the relation is satisfied and "false" if it is not.

If a register or flop is being loaded and tested simultaneously, the test applies to the value before the load. The F register may not be loaded and tested simultaneously.

Microprocessor tests which are functions of AUZ, OVFL, or CRY may not force these signals to detect on different lengths (16 or 20 bits) than any simultaneous FLOPS operands (refer to subsection 3.4) which are functions of these signals.

Table 3-9 Firmware Sequencing Conditions (Sheet 1 of 5)

CONDITION	SIGNAL OR FUNCTION TESTED	NOTES & RESTRICTIONS
	MICROPORCESSOR TESTS	
IFALU0	ALU(0)	Refer to Note 1.
IFAUZ	AUZ	Refer to Note 1.
IFAUZ20	AUZ	Refer to Note 2.
IFCRY	CRY	Refer to Note 1.
IFCRY20	CRY	Refer to Note 2.
IFOVFL	OVFL	Refer to Note 1.
IFQSR	Q(19)	Microprocessor area must specify DR or SR.
IFSHIN	SHIN	
IFSHZ	SHIN V AUZ	Refer to Note 1.
IFSHZ20	SHIN V AUZ	Refer to Note 2.
	INTERNAL BUS TEST	
IF4EQ5	BI(4) = BI(5)	
IFBI4	BI(4)	
IFBI12	BI(12)	
IFBI19	BI(19)	

Table 3-9 Firmware Sequencing Conditions (Sheet 2 of 5)

CONDITION	SIGNAL OR FUNCTION TESTED	NOTES & RESTRICTIONS
IFBINUM	Internal bus bit determined by the register number field, F(1-3), as follows: F Register Bits 1-3 → Bit Tested 0 0 0 → I(I) 0 0 1 → BI(13) 0 1 0 → BI(14) 0 1 1 → BI(15) 1 0 0 → BI(16) 1 0 1 → BI(17) 1 1 0 → BI(18) 1 1 1 → BI(19)	
IFPMUX	Previous firmware step had P, BP or BPH as internal bus source, or previous step had BUS operand PURGE, INCP, RDREQP, or PSELECT and did not have P as internal bus destination.	
IFPRIV	Privilege bit of S register, S(1)	
IFTRACE	M1(0)	
	INSTRUCTION REGISTER (F/SEL) TESTS	
IFBCND	Branch condition (appropriate to branch instruction in F register - refer to Table 3-10).	
IFBINUM	Bit determined by register number field, F(1-3), as follows: F (1-3) → Bit Tested 0 0 0 → I(I) 0 0 1 → BI(13) 0 1 0 → BI(14) 0 1 1 → BI(15) 1 0 0 → BI(16) 1 0 1 → BI(17) 1 1 0 → BI(18) 1 1 1 → BI(19)	

Table 3-9 Firmware Sequencing Conditions (Sheet 3 of 5)

CONDITION	SIGNAL OR FUNCTION TESTED	NOTES & RESTRICTIONS
IFDSELEQ0	SEL = 0	SEL←SEL-1
IFF11	F(11)	
IFF4	F(4)	
IFF5	F(5)	
IFF6	F(6)	
IFF7	F(7)	
IFF8	F(8)	
IFF9	F(9)	
IFGTWD	Instruction data field size > 16 bits (function of F - refer to Table 3-11).	
IFHALF	Instruction data field size = 8 bits (function of F - refer to Table 3-11).	
IFNUM7	F(1-3) = 7	
IFQUAD	Instruction data field size = 64 bits (function of F - refer to Table 3-11).	
IFREGAD	Address syllable calls for register addressing, i.e., [F(0) = 0] V [F(9-11) = 5 ∧ SEL ≠ 0]	
IFSCISTR	Instruction has SIP Store op-code	
IFSEL0	SEL(0)	
IFSEL1	SEL(1)	
IFSEL3	SEL(3)	
IFSELEQ0	SEL = 0	
IFSL1-3EQ7	SEL(1-3) = 7	
IFWORD	Instruction data field size = 16 bits (function of F - refer to Table 3-11).	

Table 3-9 Firmware Sequencing Conditons (Sheet 4 of 5)

CONDITION	SIGNAL OR FUNCTION TESTED	NOTES & RESTRICTIONS
	MEGABUS INTERFACE AREA TESTS	
IFACK	Megabus acknowledge flop.	
IFPARER	Parity error indicator.	
IFPMUX	Previous firmware step had P, BP, or BPH as internal bus source, or previous step had BUS operand PURGE, INCP, RDREQP, or PSELECT and did not have P as internal bus destination.	
IFRMWF	A previous firmware step successfully performed a lock, and no unlock has yet been performed.	
IFRPTRP	INTBSY V EXTRAP	SIGN ← 0
IFRUP	INTBSY	
IFYELLOW	YELLOW flag	TICK ← 0 YELLOW ← 0 Set when any EDAC-corrected memory reads were encountered since last test.
	MISCELLANEOUS HARDWARE TESTS	
IFADRER	Address error (= WRAP V SEGERR)	
IFIC	I(C)	
IFII	I(I)	
IFDDLEQ0	DDLEQ0 flop	Microprocessor destination must be Q or null.
IFLOAD	LOAD flop	
IFMISC	MISC flop	
IFMIZR	MISC V ZERO	
IFSHIN	SHIN	

Table 3-9 Firmware Sequencing Conditions (Sheet 5 of 5)

CONDITION	SIGNAL OR FUNCTION TESTED	NOTES & RESTRICTIONS
IFSHIN1	SHIN1 flop	
IFSHIN2	SHIN2 flop	
IFSHZ	SHIN V AUZ	Refer to Note 1.
IFSHZ20	SHIN V AUZ	Refer to Note 2.
IFSIGN	SIGN flop	
IFXB0	XB(0)	
IFZERO	ZERO flop	
	OTHER TESTS	
IFCACHE	Cache present	
IFCIP	CIP present	
IFEXEC	Control panel Execute pushbutton	
IFLAF	LAF - Long address form	
IFLOCK	Control panel lock function	
IFSIP	SIP present	
IFTICK	TICK flop	Set every 8-1/3 ms
IFWCS	WCS present	

NOTES

1. Forces CRY, OVFL, and AUZ to 16-bit versions

2. Forces CRY, OVFL, and AUZ to 20 bit versions (refer to subsection 3.1.2)

Table 3-10 IFBCND Test Function

F(4-7) \ F(0-3)	0,8	1-7, 9-F
0	*	SIGN
1	X	ZERO
2	I(L)	$\overline{\text{SIGN V ZERO}}$
3	I(G)	MISC
4	I(OV)	SIGN
5	I(B)	ZERO
6	I(C)	$\overline{\text{SIGN V ZERO}}$
7	I(I)	MISC
8	I(L $\forall$ U)	SIGN
9	I($\overline{\text{L V G}}$)	ZERO
A	I(G $\forall$ U)	$\overline{\text{SIGN V ZERO}}$
B	I(U)	MISC
C	1	X
D	1	X
E	1	X
F	1	X

X = Undefined

* True unless power is failing

Table 3-11 Data Field Size Tests (Sheet 1 of 2)

F(0-3)	F(4-7)	F(8-11)	IFHALF	IFWORD	IFGTWD	IFQUAD
0	0	0-1	0	1	1-M	0
0	0	2-3	0	1	1-M	0
0	0	4-5	0	1	1-M	0
0	0	6-7	0	1	1-M	0
0	1-E	0-F	0	1	1-M	0
0	F	0-7	0	1	0	0
0	F	8-F	0	1	1-M	0
1-7	0	0-F	0	0	0	0
1-7	1-2	0-F	0	1	1-M	0
1-7	3-6	0-F	0	0	0	0
1-7	7-B	0-F	0	1	1-M	0
1-7	C-F	0-F	0	0	0	0
8	0	0-F	0	1	0	0
8	1	0-F	1-M	M	0	0
8	2	0-7	0	1	0	0
8	2	8-F	0	0	0	0
8	3	0-F	0	1	0	0
8	4	0-F	0	0	1	0
8	5	0-F	0	1	0	0
8	6	0-F	0	1	0	0
8	7	0-7	0	1	0	0
8	7	8-F	1	0	0	0
8	8	0-7	0	0	0	0
8	8	8-F	0	1	0	0
8	9	0-7	0	0	0	0
8	9	8-F	0	1	0	0
8	A	0-7	0	0	0	0
8	A	8-F	0	1	0	0
8	B	0-7	0	0	0	0
8	B	8-F	0	1	0	0
8	C	0-7	0	1	0	0
8	C	8-F	0	0	1	0
8	D	0-7	0	0	1	0
8	D	8-F	0	1-L	L	0
8	E-F	0-F	0	1	0	0
9-F	0-2	0-7	0	1	0	0
9-F	0-2	8-F	1	0	0	0
9-F	3	0-F	0	1	0	0
9-F	4-7	0-7	0	1	0	0
9-F	4-7	8-F	1	0	0	0
9-F	8-9	0-7	0	1	0	0
9-F	8-9	8-F	0	0	1	S
9-F	A-B	0-F	0	1	0	0
9-F	C-D	0-7	0	0	1	S
9-F	C-D	8-F	0	1-L	L	0
9-F	E-F	0-7	0	1	0	0
9-F	E-F	8-F	0	1-L	L	0

Table 3-11 Data Field Size Tests (Sheet 2 of 2)

where:
L = 1 if LAF; otherwise, L = 0
M = 1 if MISC V ZERO; otherwise, M = 0
S = selected bit of scientific mode register (M4) or F:

F(0-3) =	9	A	B	C	D	E	F
S =	M4(2)	M4(4)	M4(6)	F(5)	M4(2)	M4(4)	M4(6)

3.5.5 Address Values

The WCS consists of a maximum of 2048 locations, making it possible to specify addresses in 11 bits. In fact, the next address field of the firmware word (refer to Appendix B) is indeed 11 bits. However, to differentiate WCS firmware locations from native firmware locations, a high order One is often appended to WCS location values (e.g., Microcode Analyzer and WCS loader).

An address value may be specified to the assembler as a literal, symbol, or statement reference (refer to Section 4). When encoding the value of the next address field, the assembler will use the low order 11 bits. When printing the location of a particular firmware step, the assembler will use 12 bits. At the user's discretion the high order bit can be specified as 0 or 1.

In Sequential mode, an address value may not be equal to 000#, 001#, 800#, or 801#. If any of these locations is specified, a hardware trap will result, causing the firmware to branch to native firmware location 000 (refer to subsection 2.7.1). The assembler will issue a diagnostic message if any of these address values is specified.

3.5.6 Firmware Sequencing Examples

Tables 3-12 and 3-13 are examples of source statements for both Transparent and Sequential modes.

Table 3-12 Source Statements for Transparent Mode

STATEMENT		MEANING
GOTO	TAG	→ TAG
GOTO	XA	→ XA splatter based on F/SEL
IFF5	802#,803#	If F(50 = 1, → 802#; else 803#
IFF5	TAG,XL	If F(5) = 1, → TAG; else → location which is function of LINK register.
IFF5	XA,XB	Invalid - at least one operand must specify value.
IFF5	805#,803#	Invalid - 805# V 003# = 803# and also 803# V 003# = 805#

Table 3-13 Source Statements for Sequential Mode

STATEMENT		MEANING
GOTO	TAG	→TAG
CALL	TAG	CALL TAG Return address register ← current address + 1
RETURN		Return to address in return address register.
LBRANCH	900#	→location in LINK register + 900#
IFF5	TAG	If F(5) = 1, →TAG
IFF5	TAG	If F(5) = 0, →TAG
IFF5	TAG,RETURN	If F(5) = 0, →address in return address register; else →TAG
IFF5	TAG,,CALL	If F(5) = 1, CALL TAG
IFF5	,RETURN	Invalid - one operand must be a branch address.
IFF5	802#,803#	Invalid - both operands may not specify a branch address in Sequential mode.

3.6 MASTER CLOCK AREA

There are four clock speeds that control the duration of each firmware step. The assembler selects, for each step, the fastest clock speed permissible based on the actions specified in the step. In rare circumstances, the selected clock speed must be overridden based on the actions of a previous step.

3.6.1 Syntax

The microinstructions in the master clock area take one of two forms:

HL
VL

where:

HL specifies a "half long" clock speed.

VL specifies a "very long" clock speed.

3.6.2 Usage of Master Clock Microinstructions

VL must be specified if:

- The current step specifies IFHALF, IFWORD, IFGTWD, or IFQUAD and the previous step altered the MISC or ZERO flops.
- The current step specifies IFREGAD and the previous step altered SEL.

HL must be specified if:

- A microprocessor shift modifier (SL, SR, DL, or DR) is specified, BI is a microprocessor source, and the shift input (SHIN) comes from the internal bus (SHIN depends on the previous setting of MISC, SHIN1, and SHIN2).

NOTE

Neither HL nor VL should be specified during a Megabus "write" step.

3.7 EXAMPLES OF FIRMWARE ROUTINES

This subsection provides examples of several firmware routines, illustrating the effective utilization of various microinstructions in accomplishing common tasks. The coding of the first five examples is extracted from actual native firmware. They therefore contain some irrelevant material, which has been retained here to illustrate the parallelism permitted in the firmware.

Example 1 - Procedure and Read from Memory

```
                                2               TITLE     EXAMPLE 1
                                3     *                   PROCEDURE READ FROM MEMORY
                                4
                                5     * THIS EXAMPLE FETCHES THE FIRST WORD OF THE NEXT INSTRUCTION AND DEPOSITS
                                6     * COPIES IN RAM LOCATION 0, IN REGISTER D0, AND IN REGISTER F/SEL.  IT ALSO
                                7     * SIMULTANEOUSLY:
                                8     *         A) SETS CONTROL FLOP ZERO TO INDICATE IF THE RECEIVED INSTRUCTION IS AN HLT
                                9     *         B) CLEARS REGISTER XB AND FLOPS SHN1, SHN2, MISC, AND SIGN
                               10     *         C) INITIALIZES THE RING EFFECTIVE LOGIC OF THE MMU
                               11     *         D) TESTS FOR THE PRESENCE OF AN EXTERNAL INTERRUPT OR TRAP PENDING
                               12
                               13     INTO      EQU       20A#
                               14     INTE      EQU       20B#
                               15
020  E82A FE31 D239 020A       16     FETCH     020#      BI        BP,RAM0,F          ; COPY PROCEDURE WORD TO RAM0,F/SEL
                               17                         OR        BI,ZERO,D0         ;   AND TO D0
                               18     *                                               ("OR" MAY BE REPLACED BY "COPY"
                               19     *                                                 IN MCSA REV 03.00)
                               20                         FLOPS     ZRAUZ,SH00,XB0,    ; PERFORM A), B),
                               21                                   MS0,RINGINIT       ;   AND C)
                               22                         IFRPTRP   INTE,INTO          / POLL FOR INTERRUPT OR TRAP
```

Example 2 - Non-Procedure Read from Memory

```
                               23               TITLE     EXAMPLE 2
                               24     *                   NON-PROCEDURE READ FROM MEMORY
                               25
                               26     * THIS EXAMPLE FETCHES AN ADDRESS FROM THE MEMORY LOCATION(S) POINTED TO
                               27     * BY THE Y-REGISTER CONTENT, AND DEPOSITS A COPY IN THE P-REGISTER.  IT ALSO
                               28     * SIMULTANEOUSLY SETS REGISTER D0 EQUAL TO THE COMPLEMENT OF REGISTER Q.
                               29
1D4  009C AF01 180A B095       30               1D4#      COPY      ZERO               ; GENERATE CONSTANT = 0
                               31                         BI        ALU,H              ; H <- 0
                               32                         RDREQ     NOCACHE            ; REQUEST MAIN MEMORY LOCATION Y
                               33                         IFLAF     *+1,*+3            / TEST FOR LONG ADDRESS FORM
                               34
095  B013 CE30 D900 03C6       35               095#      BI        BD,H               ; H <- FIRST HALF OF LAF ADDRESS
                               36                         BUS       INCY               / Y <- Y+1
                               37
3C6  0093 CF01 1000 0097       38               3C6#      RDREQ     NOCACHE            / REQUEST SECOND HALF OF ADDRESS
                               39
097  8027 AE50 E000 024C       40               097#      XORC      Q,ZERO,D0          ; DEPOSIT COMPLEMENT OF Q IN D0
                               41                         BI        BDH,P              / PUT HALVES TOGETHER IN P
```

Example 3 - Write Into Memory

```
                                42              TITLE     EXAMPLE 3
                                43    *                   WRITE INTO MEMORY
                                44
                                45    * THIS EXAMPLE WRITES THE ADDRESS CONTAINED IN REGISTER D0 INTO THE
                                46    * MEMORY LOCATION(S) POINTED TO BY RAM LOCATION 8.
                                47
24C  C893 CF00 800A 8256        48              24C#      BI        RAM8,Y              ; Y <- RAM8
                                49                        IFLAF     *+1,*+2             / TEST FOR LONG ADDRESS FORM
                                50
256  0023 C661 4000 0257        51              256#      BI        D0,L4               ; LEFT 4 BITS OF D0 VIA BI TO
                                52                        WRT       INCY                /   FIRST WORD OF LAF STORE
                                53    *                                                 Y <- Y+1
                                54
257  00A3 C701 4000 037F        55              257#      BI        D0                  ; (REST OF) D0 VIA BI
                                56                        WRT       INCY                /   TO MEMORY
                                57    *                                                 Y <- Y+1
```

Example 4 - I/O Read

```
                                58              TITLE     EXAMPLE 4
                                59    *                   I/O READ
                                60
                                61    * THIS EXAMPLE REQUESTS INPUT FROM THE I/O SUBSYSTEM, USING THE CHANNEL
                                62    * NUMBER AND FUNCTION CODE CONTAINED IN REGISTER D0, AND (IF THE REQUEST
                                63    * IS ACKNOWLEDGED) COPIES THE REPLY INTO REGISTER B0. IT THEN EXITS TO STORE
                                64    * THE REPLY AS REQUIRED.  THE I/O INDICATOR IS SET/CLEARED IF THE REQUEST
                                65    * IS/ISN'T ACKNOWLEDGED.  NOTE THAT THE CHANNEL NUMBER/FUNCTION CODE MUST
                                66    * BE PLACED IN Y(5-19)/XB(0) FOR TRANSMISSION TO THE MEGABUS.
                                67
37F  04D3 CF00 2134 0369        68              37F#      COPY      D0,B0,SR            ; SHIFT CHANNEL/FUNCTION TO B0
                                69                        FLOPS     XBSR,SH00           ;   AND XB(0); CLEAR SHN1 AND SHN2
                                70                        IFSIGN    I030,I03E           / I030 FOR I/O WRITE (SEE EXAMPLE 5)
                                71
369  4943 C300 8014 00A9        72    I03E      369#      BI        B0,YR16             ; SHIFTED INFORMATION INTO Y(4-19)
                                73                        IFSHIN    *+2,*+1             / TEST XB(1) VIA SHIN
                                74
0A9  04AC AF01 28C0 036A        75              0A9#      COPY      ZERO,R0             ; XB(1)=0 SO B0 <- ZERO'S
                                76                        RDREQ     I-O                 ; REQUEST I-O READ
                                77                        FLOPS     MS0                 ; MISC <- 0
                                78                        GOTO      *+2                 / SKIP NEXT STEP
                                79
0AB  44A7 9F01 28C0 036A        80              0AB#      XORC      B0,B0,R0            ; XB(1)=1 SO B0 <- ONE'S
                                81                        RDREQ     I-O                 ; REQUEST I-O READ
                                82                        FLOPS     MS0                 / MISC <- 0
                                83
36A  CCD3 C750 851D F37D        84              36A#      BI        RAM8,Y              ; RESTORE Y FROM RAM8
                                85                        COPY      R0,B0,SR            ; SHIFT B0 TO
                                86                        FLOPS     XBSR,IACK           ;   RESTORE XB(0); I(I) <- ACK
                                87                        IFACK     *+1,XF              / IF ACK=0, EXIT TO NEXT FETCH
                                88
37D  8423 FA30 F000 D7FF        89              37D#      BI        RD                  ; COPY READ RESPONSE
                                90                        COPY      BI,B0               ;   TO B0
                                91                        GOTO      XW                  / GO TO STORE RESPONSE
```

Example 5 - I/O Write

```
                                92              TITLE     EXAMPLE 5
                                93    *                   I/O WRITE
                                94
                                95    * THIS EXAMPLE TRANSMITS DATA TO THE I/O SUBSYSTEM, USING THE CHANNEL
                                96    * NUMBER AND FUNCTION CODE CONTAINED IN REGISTER D0.  THE I/O INDICATOR IS
                                97    * SET/CLEARED IF THE REQUEST IS/ISN'T ACKNOWLEDGED.  THE EXAMPLE BEGINS WITH
                                98    * THE FIRST STEP (37F#) OF EXAMPLE 4, BUT THE FIRST BRANCH LEADS TO A NETWORK
                                99    * OF STEPS RESULTING IN THE COPYING OF THE APPROPRIATE DATA TO REGISTER 4.
                               100
                               101
37F  04D3 CF00 2134 036B       102              37F#      COPY      D0,B0,SR            ; SHIFT CHANNEL/FUNCTION TO B0
                               103                        FLOPS     XBSR,SH00           ;   AND XB(0); CLEAR SH#1 AND SH#2
                               104                        IFSIGN    I030,I03E           / I03E FOR I/O READ (SEE EXAMPLE 4)
                               105
365  0093 CF00 2000 03A0       106    I030      36B#                                  / FIRMWARE SEQUENCE TO MOVE DATA TO REGISTER 4
                               107
3A0  40A3 4F00 B000 034C       108    XLIOUT    3A0#      COPY      Q,D0                ; D0 <- Q
                               109                        BI        B0,YR16             / SHIFTED INFORMATION INTO Y(4-19)
                               110
34C  00A3 C751 792A F34D       111              34C#      BI        D0                  ; DO VIA BI
                               112                        WRT       I-O                 ;   TO I-O CHANNEL
                               113                        FLOPS     MS1,IACK            ; MISC <- 1; I(I) <- ACK
                               114                        IFF8      *+1,XF              / IF NOT IOLD INSTRUCTION, EXIT TO FETCH
```

Example 6 - Exit From WCS Transparent Mode

```
                               115              TITLE     EXAMPLE 6
                               116    *                   EXIT FROM WCS TRANSPARENT MODE
                               117
                               118    * THIS EXAMPLE ILLUSTRATES THE SEQUENCE REQUIRED TO RETURN FROM THE WCS IN
                               119    * TRANSPARENT MODE TO NATIVE FIRMWARE.  IN GENERAL, WHEN A USER HAS COMPLETED
                               120    * AN INSTRUCTION OR HAS SENSED A REQUEST FOR SERVICE BY AN INTERRUPT, REAL-TIME
                               121    * CLOCK, ETC., THE RETURN IS TO THE NATIVE FIRMWARE LOCATION LABELLED "FETCH"
                               122    * (SEE EXAMPLE 1).  UNDER EXCEPTIONAL CIRCUMSTANCES (E.G., TRAP CONDITION
                               123    * DETECTION), RETURN MAY BE TO OTHER NATIVE FIRMWARE LOCATIONS.
                               124
                               125    * TO INVOKE THE "NORMAL" RETURN, AN ALGORITHM MIGHT END WITH THE FUNCTIONAL
                               126    * EQUIVALENT OF:
340  0093 CF00 28C0 07FC       127                        FLOPS     MS0                 ; MISC <- 0 (NO TRAP)
                               128                        GOTO      EXITTRN             / GO TO EXIT ROUTINE
                               129
                               130    * WHEN A TRAP CONDITION IS DETECTED, ONE MIGHT EMPLOY THE FUNCTIONAL
                               131    * EQUIVALENT OF:
                               132
                               133    TRAP      EQU       33B#                          NATIVE TRAP FIRMWARE
                               134    TV=X      EQU       0031#                         FOR TV15 (OTHER VALUES FOR OTHER TRAPS)
                               135
34E  8423 FC10 2900 0234       136                        BI        TV=X                ; CREATE TRAP VECTOR #X
                               137                        COPY      BI,B0               ;   AND SAVE IN B0
                               138                        FLOPS     MS1                 / MISC <- 1 (TRAP)
                               139
234  0093 CF00 2000 07FC       140              234#      GOTO      EXITTRN             / GO TO EXIT ROUTINE
                               141
                               142    * THE ACTUAL EXIT ROUTINE LOOKS AS FOLLOWS:
                               143
FFC  C423 7E90 2500 07FD       144    EXITTRN   FFC#      BI        IDCF                ; START WITH CIP CHANNEL,
                               145                        COPY      BI,D0               ;   FUNCTION IF IN D0
                               146                        FLOPS     XBSR1               / XB(0) <- 1
                               147
FFD  8026 DC00 B000 07FE       148              FFD#      BI        0FD#,Y              ; Y(0-4) <- 0; USE 000FD# TO
                               149                        XOR       BI,D0,D0            /   MODIFY CHANNEL TO WCS,
                               150    *                                                 FUNCTION TO 25
                               151
FFE  00A3 CF00 B000 07FF       152              FFE#      BI        D0,YR16             / Y(5-19) <- CONTROL WORD/2
                               153
FFF  8013 C641 7035 F336       154              FFF#      BI        Y                   ; MUST SPECIFY NON-ALU BI SRC FOR WRT
                               155                        WRT       I-O                 ; TRANSMIT EXIT CODE TO WCS
                               156                        IFMISC    TRAP,XF             / RETURN DEPENDING ON MISC
```

Example 7 - Exit From WCS Sequential Mode

```
                                157             TITLE     EXAMPLE 7
                                158     *                 EXIT FROM WCS SEQUENTIAL MODE
                                159
                                160     * THIS EXAMPLE ILLUSTRATES THE SEQUENCE REQUIRED TO RETURN FROM THE WCS IN
                                161     * SEQUENTIAL MODE TO NATIVE FIRMWARE.  IN GENERAL, WHEN A USER HAS COMPLETED
                                162     * AN INSTRUCTION OR HAS SENSED A REQUEST FOR SERVICE BY AN INTERRUPT, REAL-TIME
                                163     * CLOCK, ETC., THE RETURN IS TO THE NATIVE FIRMWARE LOCATION LABELLED "FETCH"
                                164     * (SEE EXAMPLE 1).  UNDER EXCEPTIONAL CIRCUMSTANCES (E.G., TRAP CONDITION
                                165     * DETECTION), RETURN MAY BE TO OTHER NATIVE FIRMWARE LOCATIONS.
                                166
                                167             SEQUENTIAL
                                168
                                169     * TO INVOKE THE "NORMAL" RETURN, AN ALGORITHM MIGHT END WITH THE FUNCTIONAL
                                170     * EQUIVALENT OF:
123  0093 CF00 28C0 87FC        171             123#      FLOPS     MS0                 ; MISC <- 0 (NO TRAP)
                                172                       GOTO      EXITSEQ             / GO TO EXIT ROUTINE
                                173
                                174     * WHEN A TRAP CONDITION IS DETECTED, ONE MIGHT EMPLOY THE FUNCTIONAL
                                175     * EQUIVALENT OF:
                                176
                                177     TRAP      EQU       338#                        NATIVE TRAP FIRMWARE
                                178     TV-X      EQU       0031#                       FOR TV15 (OTHER VALUES FOR OTHER TRAPS)
                                179
124  0093 C800 2900 C7FC        180                       FLOPS     MS1                 ; MISC <- 1 (TRAP)
                                181                       CALL      EXITSEQ             / INVOKE EXIT SUBROUTINE
                                182
125  8423 F411 7000 833B        183                       BI        TV-X                ; CREATE TRAP VECTOR #X
                                184                       COPY      BI,B0               ;   AND SAVE IN B0
                                185                       WRT       I-O                 ; TRANSMIT EXIT CODE TO WCS
                                186                       GOTO      TRAP                / GO TO NATIVE TRAP FIRMWARE
                                187
                                188     * THE ACTUAL EXIT ROUTINE LOOKS AS FOLLOWS:
                                189
FFC  C023 7E90 2500 07FF        190     EXITSEQ   FFC#      BI        IDCF                ; START WITH CIP CHANNEL,
                                191                       COPY      BI,D0               ;   FUNCTION IF IN D0
                                192                       FLOPS     XBSR1               / XB(0) <- 1
                                193
FFD  8026 DCD0 8000 07FF        194                       BI        0FD#,Y              ; Y(0-4) <- 0; USE 000FD TO
                                195                       XOR       BI,D0,D0            /   MODIFY CHANNEL TO WCS,
                                196     *                                               FUNCTION TO 25
                                197
FFE  0043 C800 8035 A7FF        198                       BI        D0,YR16             ; Y(5-19) <- CONTROL WORD/2
                                199                       IFMISC    RETURN,*+1          / RETURN IF TRAP
                                200
FFF  8013 C641 7000 8020        201                       BI        Y                   ; MUST SPECIFY NON-ALU BI SRC FOR WRT
                                202                       WRT       I-O                 ; TRANSMIT EXIT CODE TO WCS
                                203                       GOTO      FETCH               / GO TO NATIVE FETCH FIRMWARE
```

SECTION 4 - WCS ASSEMBLY LANGUAGE

Preparation of microprograms is simplified using the WCS assembly language. It relieves the microprogrammer of many time-consuming duties associated with writing a microprogram in actual machine-language. For example, it allows the microprogrammer to employ meaningful symbolic tags rather than absolute control store addresses to specify firmware locations.

The assembly system can be divided into two parts: the WCS assembly language and the WCS microinstruction assembler. The WCS assembly language is used to write a microprogram (source file), and the assembler translates the source file into the actual machine-language microprogram (object file).

The microprogrammer can use any type of coding form to write a microprogram because of the free form used for coding source statements. The desired microinstructions and associated operands are entered onto the coding form, with each statement representing one firmware word. These source statements may then be transcribed onto punched cards or entered into a disk file via a teletype or other terminal using the MDT editor. In either case, the data constitutes the source file that will be processed by the assembler.

The assembler reads the source file and produces a machine-language object file. It converts mnemonic codes (microinstructions) into machine-language codes, assigns absolute control store addresses to symbolic names, and completely assembles the final microprogram, storing it onto a disk file or magnetic tape. Another output from the assembler is a listing of the source file and corresponding object codes, plus diagnostic messages. Figure 4-1 illustrates the relationship among the input source file, the assembler, and the output object file.

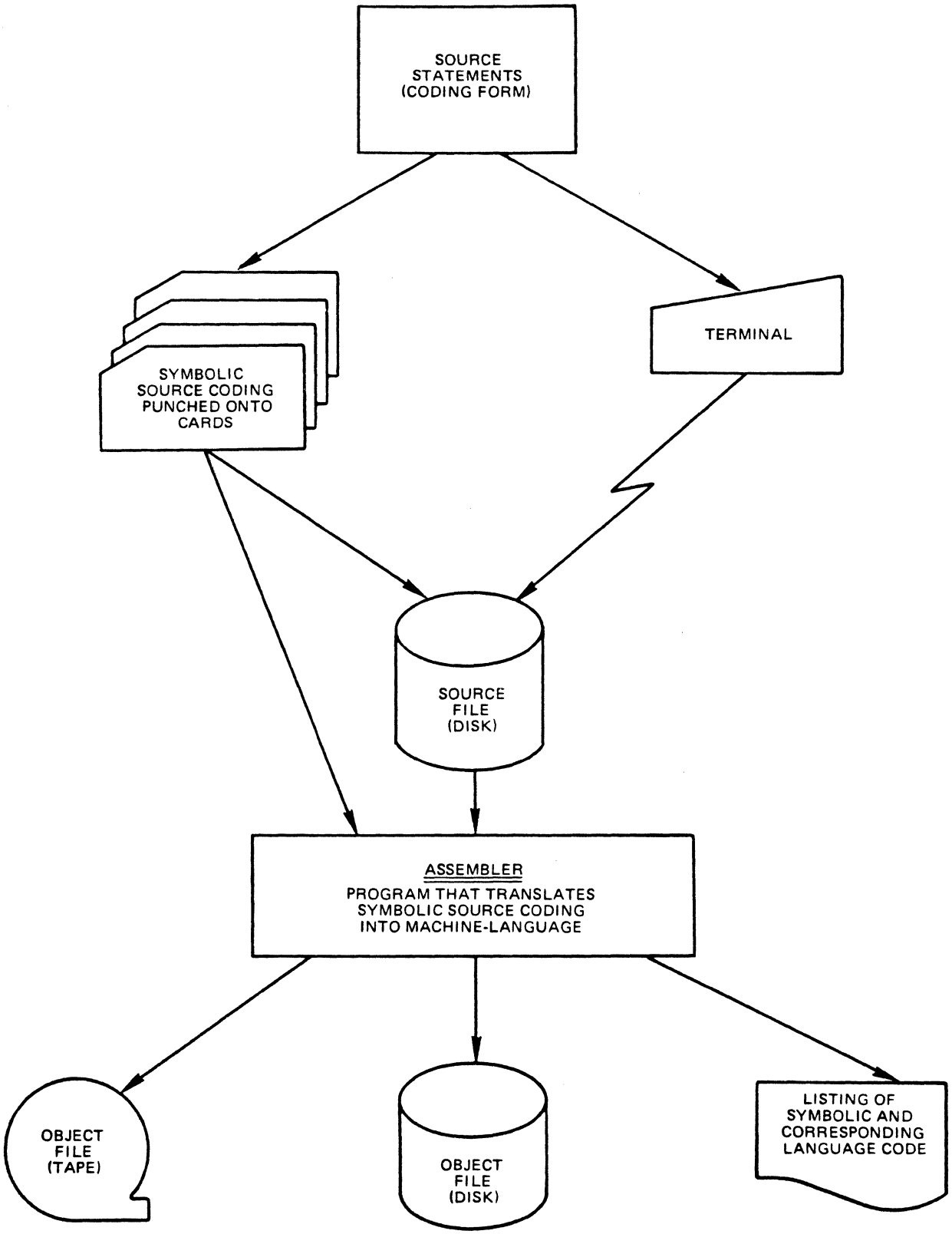


Figure 4-1 Relationship of Source File, Assembler, and Object File

Descriptions and examples within this section use the following conventions:

{ } - Indicates that one of the options enclosed in the braces must be selected.

[ ] - Indicates that the enclosed option may or may not be selected.

... - Indicates that the immediately preceding option may be repeated.

h - Indicates a hexadecimal digit.

d - Indicates a decimal digit.

o - Indicates an octal digit.

b - Indicates a binary digit.

△ - Indicates that one or more spaces or horizontal tab characters are required.

The following special characters must be coded exactly as shown:

- comma ——————→ ,
- left parenthesis ——→ (
- right parenthesis ——→)
- asterisk ——————→ *
- slash ——————→ /
- plus sign ——————→ +
- hyphen or minus sign →- -
- quote ——————→ '

4.1 ELEMENTS OF WCS ASSEMBLY LANGUAGE

The principal elements of the WCS assembly language are:

- Mnemonic codes
- Symbolic names
- Constants
- Statement references
- Punctuation

These elements are combined to form a source file that consists of: (1) microinstructions, (2) assembler control statements that direct assembler operations, and (3) statements that define constants used by the microprogram.

4.1.1 Mnemonic Codes

The mnemonic codes combine to specify the microinstructions to be created. These codes are also called reserved words because they are only recognized for their meanings as described in Section Three. Mnemonic codes can be any length, although only the first six characters are recognized by the assembler. For example, the test condition mnemonic IFRPTRP may also be coded as IFRPTR for a six-character identification, or IFRPTRAP to further clarify its meaning. Also, any spelling errors beyond the sixth character are ignored by the assembler.

4.1.2 Symbolic Names

Symbolic names are mnemonics that are not recognized as reserved words (refer to subsection 4.1.1). These names are specified by the microprogrammer and may be used to label firmware statements, operand values, and constants. Regardless of their use, the symbolic names must conform to the following rules.

1. Names may consist of any number of characters, but the first six characters uniquely identify the name (the assembler will ignore all characters after the sixth). For example, the symbolic names ADDRSS and ADDRSS00 are recognized by the assembler as the same name.

2. Names may be composed of alphabetic characters, decimal digits, and/or special characters as defined below:

   alphabetic characters - A through Z

   decimal digits - 0 through 9

   special characters - dollar sign ($), underscore (_), and hyphen (-).

3. The first character of each name may be any one of the above characters except a hyphen or decimal digit.

4. Lowercase alphabetic characters are considered equivalent to the corresponding uppercase characters. However, the assembler does preserve the case of each character in the output listing.

5. A name may not be equivalent to a reserved mnemonic code.

Symbolic names are divided into two categories: firmware statement labels and EQU symbols. Firmware statement labels may each be defined only once in a microprogram and have addresses assigned as their values. EQU symbols, on the other hand, may be defined any number of times by EQU statements which equate the symbol to different values.

4.1.3 Constants

Numeric constants permit the microprogrammer to express values within the range 0 through $2^{64}-1$. The assembler recognizes numeric constants in hexadecimal, decimal, octal, or binary form as indicated below.

Hexadecimal

Hexadecimal constants must be in one of the following two formats:

X'[[h][Δ]]...' or h[h]...#

where:

X specifies a hexadecimal constant.
h equals any digit within the range 0 through F.
specifies the end of a hexadecimal constant string.

Decimal

Decimal constants must be in the following format:

d[d]...

where:

d equals any digit within the range 0 through 9.

Octal

Octal constants must be in one of the following two formats:

C'[[o][Δ]]...' or o[o]...C

where:

C specifies an octal constant or the end of an octal constant string.

o equals any digit within the range 0 through 7.

Binary

Binary constants must be in one of the following two formats:

B'[[b][Δ]]...' or b[b]...B

where:

B specifies a binary constant or the end of a binary constant string.

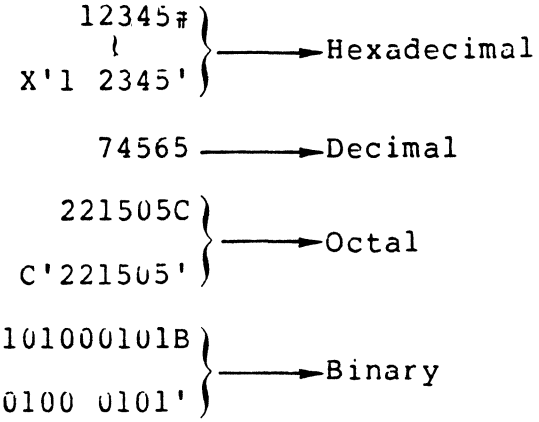
b equals 0 or 1.

All constants coded in the above formats are converted by the assembler into a 64-bit value that is right justified and either zero filled or truncated from the left to fit into the receiving field. For example, if a hexadecimal address is coded as X'1234', the address is actually interpreted as X'234', retaining the low order eleven bits.

NOTE

No warning message is issued when One bits are truncated.

The following are examples of numeric constants with all values being equal.



Numeric constants coded in the format X'', C'', or B'' are considered zero values by the assembler.

4.1.4 Statement References

Statement references permit the microprogrammer to reference firmware statements that are a specific number of steps away from the current firmware statement, without using a label. The "offset" portion of the reference is the count of firmware statements to the referenced firmware statement. The proper format for statement references is:

$$*\left[\{\pm\}[\text{offset}]\right]$$

where:

offset is either an EQU symbol or a numeric constant (refer to subsections 4.3.3 and 4.1.3, respectively).

+ indicates a forward reference.

- indicates a backward reference.

*, *+, *- indicates the address of the current firmware statement.

Thus, the "offset" is used to count steps forward or backward from the current firmware statement to locate the referenced firmware statement. This is generally not equivalent to adding or subtracting the offset and the current address. For example, assume that the following two statements appear consecutively in a source file.

```
LABEL1    5    GOTO    *+1
LABEL2   10    GOTO    *-1
```

The statement tagged LABEL1 is loaded at location 805# and the statement tagged LABEL2 at location 80A#. The argument *+1 references location 80A#, while the argument *-1 references location 805#. This is in contrast to many other assembly languages where *+1 would reference location 806#, and *-1 would reference location 809#.

If a statement reference exceeds the boundaries of a microprogram (i.e., refers to a statement prior to the first or after the last statement), the assembler considers the reference equal to zero and generates an appropriate error message. In all other cases, the reference equals the address of the referenced statement.

4.1.5 Punctuation

Recognized punctuation for the WCS assembly language includes:

Comma [,] - Separates multiple arguments for a single op-code.

Left parenthesis [(] - Optional marker of an argument list.

Right parenthesis [)] - Optional terminator of an argument list.

Semicolon [;] - Links the current line of coding and the following line into one statement. A line terminated with a ; indicates that the statement continues on the following line. It also separates comments from a microinstruction.

Blank - Separates the various language units when no other punctuation is used.

Slash [/] - Separates comments from the microinstruction on the last line of a step. It is also used to cause page ejection in printed output.

4.2 SOURCE STATEMENT FORMATS

Source line formats are classified into four main groups: firmware statements, pseudo-op (or control) statements, blank lines, and comment lines. Firmware statements generate object code to control the CPU hardware, whereas pseudo-op statements direct assembler operations. When coding WCS assembly language source statements, the user must conform to the formatting conventions described herein.

Source code for the assembler is loaded into a sequential file which, if contained on a disk, may be edited using the text editor. Each line of coding may be up to 255 characters in length and can contain a line number; however, only the first 92 characters appear in the assembly listing. If a line number is used, it must start in the first character position of the line, and must be comprised of all decimal characters. Ignoring the line number, the assembler begins processing the line with the leftmost nondecimal character. The types of source lines available to the microprogrammer include:

- Firmware statements
- Pseudo-op statements
- Blank lines
- Comment lines

The descriptions of these source statements refer to the current address counter. This counter is internal to the assembler and contains the address of the current firmware statement. The counter is incremented by one at the end of each firmware statement and may also be loaded from the address field of any statement.

4.2.1 Firmware Statement

The proper format for the firmware statement is:

[label] [△address] [△microinstructions]

The "address", if present, is either an EQU symbol or a constant. The current address counter is made to equal the contents of this field.

The "label" or symbolic name, if present, is made to equal the contents of the current address counter, after the counter has been modified by the address field of the statement. This label is referred to as the firmware statement label and must start in the first character position after the line number, if any.

The "microinstructions" have the form:

micro opcode [operand list] [△micro opcode [operand list]] ...

where micro opcode is any of the reserved words that represent microoperations.

The "operand list" has the form:

$$\left\{\begin{matrix}(\\ \Delta\end{matrix}\right\}\text{operand...,operand}\left\{\begin{matrix})\\ \Delta\end{matrix}\right\}$$

where operand is either a symbolic name, mnemonic, constant, or statement reference. The use of any particular type of operand is restricted by the particular type of micro op-code being used. Micro op-codes and their operands are described in detail in Section Three.

Examples of legal forms for an operand list are:

FORM	DESCRIPTION
operand	single operand
operand,)	two operands, the second being null
(,)	two null operands
operand,,operand	three operands, the second being null.

An operand list is terminated by a right parenthesis, an end of statement, or an operand not followed by a comma (ignoring spaces and semicolons). Thus, in the following incorrect firmware statement:

```
ADD      BI,D0 D1
```

D0 is the last operand and D1 is interpreted as a micro op-code.

A firmware statement must have at least a label, an address, or a microinstruction. When no microinstructions are present, the statement is assembled as the default value (refer to subsection 4.3.1).

4.2.2 Pseudo-Op Statement

The pseudo-op statement provides only control information for the assembler; no object code is produced. Although the pseudo-op statement must conform to the following format, it may occupy any number of lines by using the semicolon. The proper format for the pseudo-op statement is:

[label][Δaddress] Δpseudo opcode[Δpseudo op information][comment]

The "address" is loaded into the current address counter.

The "label" is a user supplied symbolic name and, except for EQU statements, the label is treated as a firmware statement label for the next firmware step. The symbolic name is made to equal the current address counter after modification by the address field. The label must start in the first character position after the line number, if any.

The following is an example of a pseudo-op statement:

```
BETA     1F#     NO LIST     TURN OFF;
                             ASSEMBLER LISTING
```

where BETA is the label which, along with the current address counter, is made to equal the address (1F#). NO LIST is the pseudo-op. The comment starts with TURN and ends on the second line with LISTING. The comment is continued on the second line because of the semicolon (;) after OFF. The following example is equivalent to the example given above, except it illustrates how the semicolon (;) can be used to extend a pseudo-op statement across several lines.

```
BETA     ;
1F#      ;
NO       ;
LIST     ;
TURN OFF ASSEMBLER LISTING
```

The pseudo-ops that are available to the microprogrammer are described in detail in subsection 4.3.

4.2.3 Blank Lines

The blank line permits spacing of the listing and is otherwise ignored by the assembler. Blank lines contain no information other than an optional line number. A blank line imbedded in a firmware statement does not terminate the statement.

4.2.4 Comment Lines

A line containing a slash (/) or an asterisk (*) as the first character of the line is treated as a comment line and has no affect on the continuation of the current statement. The comment line beginning with a / causes the output listing to slew to the top of form before printing the comment, whereas the comment line beginning with an * is merely printed on the next line.

4.3 CONTROL STATEMENTS

Control statements (or pseudo-ops) are not assembled into the object file, but rather provide the assembler with listing control, assembly control, and background information. Only the first six characters are used by the assembler to distinguish among the pseudo-ops; however, the full mnemonic names are given to improve readability in the source listing. The pseudo-ops available to the microprogrammer include:

- DEFAULT
- END
- EQU
- LABEL
- LIST
- NATIVE
- NLST
- NO
- SEQUENTIAL
- TITLE

4.3.1 DEFAULT Statement

The presence of DEFAULT as the first micro op-code in a firmware statement results in the firmware image not being placed at the current address, but is used to fill in firmware image bits that have not been set by the microinstructions. The bits not set by the microinstructions in the DEFAULT statement are set by the previous default value. The original default value used by the assembler is:

X'0093 B700 2000 07FF'

The DEFAULT statement is used most frequently to modify the automatic clock speed setting feature of the assembler (refer to subsection 3.6). The assembler will not select a clock speed any higher than the speed specified in the current default value. If, during firmware debug operations, a timing problem is suspected, the following DEFAULT statement can be specified:

```
DEFAULT    VL
```

The above statement always causes the assembler to select the very long clock speed.

4.3.2 END Statement

The END statement may be used to mark the termination of various firmware routines, and may appear anywhere in the source as often as desired. This statement has no effect on the assembly process.

4.3.3 EQU Statement

The proper format for an EQU statement is:

label [Δaddress] Δ EQU Δ { constant / EQU symbol } [Δcomment]

The EQU statement equates the statement label and EQU symbol to the value of the constant or EQU symbol name that follows. The EQU symbol name must have been previously defined in an EQU statement. The statement address (if present) is used only to load the current address counter.

EQU symbols may be defined more than once, but remain undefined until the first EQU statement is encountered. When the EQU symbol is defined, it retains this definition until it is redefined in a subsequent EQU statement (i.e., the value assigned to an EQU symbol is retained by the symbol for all subsequent firmware steps until the end of the source coding or until another EQU statement changes the value assigned to the symbol).

An EQU symbol must be defined in an EQU statement before it is referenced. Any reference to an EQU symbol before it is initially defined will be interpreted as a reference to a firmware statement label, resulting in a diagnostic message.

EQU statements may be used throughout the source coding as illustrated in the following example.

```
(1)     ALPHA           EQU         1
                        .
                        .
(2)                     BI          ALPHA
                        .
                        .
(3)     ALPHA           EQU         2
                        .
                        .
(4)                     BI          ALPHA
                        .
                        .
```

At line 1, in the above example, the EQU symbol (ALPHA) is defined as being equal to a constant of 1. This permits the BI microinstruction located on line 2 to source this constant (ALPHA) onto the internal bus. At line 3, ALPHA is redefined to equal a constant of 2, permitting the BI microinstruction located on line 4 to source this constant onto the internal bus using the same EQU symbol (ALPHA).

4.3.4 LABEL Statement

The LABEL statement defines its "label" and "address" fields without performing any other special functions. Thus, this statement may be interpreted as the statement label for the next firmware statement as illustrated in the following example.

```
(1)     LABELA          83E#        LABEL
(2)                                 microcode
(3)     LABELB          84E#        LABEL
(4)                     85E#        microcode
```

LABELA (located on line 1 in the above example) and the contents of the current address counter are made to equal address 83E#. Thus, LABELA is referred to as the statement label for the microcode on line 2. This microcode is assembled at address 83E#.

LABELB (located on line 3 in the above example) and the contents of the current address counter are made to equal address 84E#. However, in this case, the microcode on line 4 is assembled at address 85E# rather than 84E# because it has its own address. This prevents LABELB from being referred to as the statement label for the microcode on line 4.

4.3.5 LIST Statement

The LIST statement instructs the assembler to restart the output listing with the LIST statement; the default mode is LIST. Therefore, LIST is unnecessary until it is desired to resume the output listing after the NLST or NO LIST statements.

4.3.6 NATIVE Statement

The NATIVE statement invokes the native (transparent) firmware branching mode for subsequent firmware statements. The default mode is NATIVE; therefore, NATIVE is unnecessary except for documentation purposes.

4.3.7 NLST Statement

The NLST statement suspends the source listing starting with the NLST statement; the default mode is LIST. Subsequent lines are not listed unless they are in error or until the next LIST statement.

4.3.8 NO LIST Statement

This statement is equivalent to NLST. Source statements starting with the NO LIST statement up to but not including the following LIST statement are not listed unless they are in error.

4.3.9 SEQUENTIAL Statement

The SEQUENTIAL statement invokes the sequential firmware branching scheme for subsequent firmware statements. The default mode is NATIVE; therefore, this statement must be used for sequential mode assemblies.

4.3.10 TITLE Statement

The TITLE statement names the code, provides a revision number, and specifies the text for the heading field on the first line of each page in the output listing. The TITLE statement takes two forms.

TITLE name,rev,title

This form specifies the name of the source, the revision number, and the text for the heading line, and may appear anywhere in the source file as the first TITLE statement. All subsequent TITLE statements will use the second form described below.

TITLE title

This form specifies only the text for the heading line (i.e., it replaces the text supplied by the initial TITLE statement).

The above forms for the TITLE statement cause a slew to the top of form, and may be used as many times as desired. Also, the TITLE statement is not required. If there is no TITLE statement, the assembler assumes the source name is WCSRTN and the revision and title fields are all spaces. All fields for the TITLE statement are optional, null sets the respective fields to blanks.

4.4 INTERPRETING WCS ASSEMBLY LISTINGS

The assembler output listing provides the user with a printed copy of the source file, the corresponding machine-language code, and diagnostics. Figure 4-2 is a sample output listing that defines the various areas of the printed output.

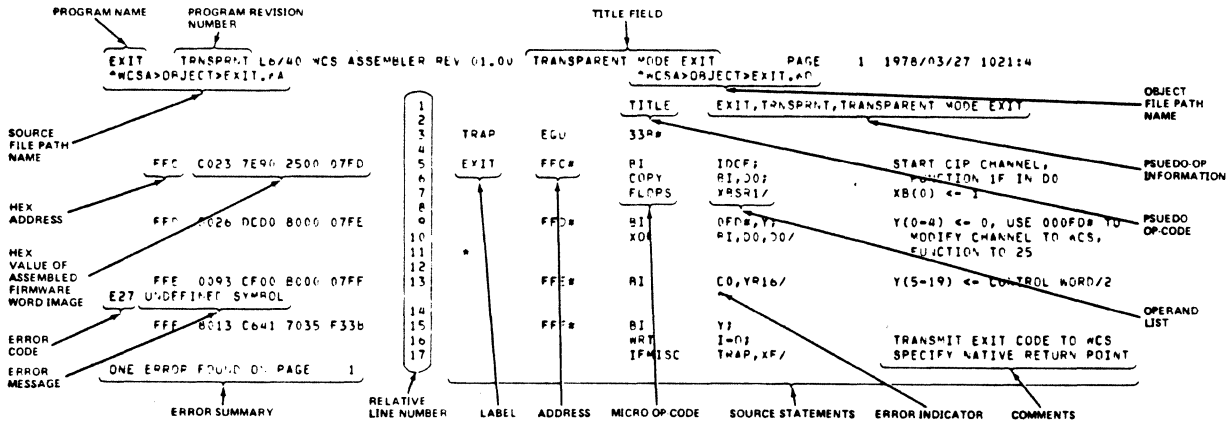


Figure 4-2 Sample Output Listing

Included with most error messages in an output listing is the caret (^) symbol. This symbol appears below the source character or firmware word in error. In cases of null operands, the caret may point to a seemingly irrelevant position.

4.5 WCS ASSEMBLER OBJECT DECK FORMAT

The assembler object deck file is used as input by the WCS loader to load the WCS. Figure 4-3 is a sample file dump of an object deck produced by the assembler. There are six types of object deck records as indicated below.

```
00001 0017 0103 5100 0000 0045 5849 5420 2020 2054 524E 5350 524E 54          ..Q....EXIT    TRNSPRNT
00002 0015 0331 3937 382F 3033 2F32 3720 3130 3231 3A34 332E 37              .1978/03/27 1021:43.7
00003 001F 0454 5241 4E53 5041 5245 4E54 204D 4F44 4520 4558 4954 2020 2020 2020 20   .TRANSPARENT MODE EXIT
00004 0005 0A00 000F FC                                                      .....
00005 0009 0CC0 237E 9025 0007 FD                                            ..#~.%...
00006 0009 0C80 26DC D080 0007 FE                                            ..&.....
00007 0009 0C00 93CF 00R0 0007 FF                                            .........
00008 0009 0CR0 13C6 4170 35F3 3B                                            ....Ap5.;
00009 0005 FF00 0000 00                                                      .....

            EOF
```

Figure 4-3 Sample File Dump

RECORD TYPE	BYTE POSITIONS	DEFINITION
Program ID	0-6	0103510000000#
	7-12	Program Name
	13-14	Spaces
	15-22	Revision
Date	0	03#
	1-18	Date and time
Title	0	04#
	1-28	Title field from first TITLE statement
Origin	0	0A#
	1-4	Address
Data	0	0C#
	1-8	Assembler firmware image
End of file	0	FF#
	1-4	Reserved for future use

4.6 ASSEMBLER OUTPUT LISTING ERROR MESSAGES

All of the error messages that are produced on the output listing by the assembler are defined herein. These messages describe the nature of the problem, its possible causes, and suggested solutions. The following error messages are presented exactly as they appear in the output listing (see Figure 4-2).

E01 MISSING SYMBOL FOR EQU DEFINITION

The EQU statement is missing a statement label.

E02 MISSING VALUE FOR EQU DEFINITION

The EQU statement requires a constant or previously defined EQU symbol after the EQU mnemonic.

E03 SYMBOL PREV USED - REMAINS UNDEFINED

An EQU symbol may not be used before it is defined. The label of this EQU statement has been previously referenced and is assumed to be a firmware statement label in that reference.

E04 SYMBOL PREVIOUSLY USED AS LABEL

The EQU statement's label is already a label for a previous non-EQU statement. The label definition for this EQU statement is ignored.

E05 MULTIPLY DEFINED LABEL

This statement's label definition is ignored because it has been defined in a previous firmware statement.

E06 ILLEGAL MICRO OP

The indicated reserved word is only used as an operand. It cannot (by itself) specify a microoperation.

E07 TOO MANY OPERANDS

The indicated micro op-code cannot accommodate the number of arguments used.

E08 PSEUDO OP CANNOT BE OPERAND

The indicated reserved word is a pseudo-op and cannot be used as an operand for a micro op-code.

E09 MISPLACED VALUE

The statement already has an address or the previous operand list has been completed. A comma might be missing.

E10 MISPLACED SYMBOL

The assembler is expecting a microinstruction but has detected a nonreserved word. Either the symbol was intended for the previous operand list, in which case a comma might be missing, or a reserved word might have been misspelled.

E11 MISPLACED STATEMENT REFERENCE

The assembler expects a microinstruction at the indicated position but has encountered a statement reference. There might be a comma missing.

E12 PSEUDO OP IN FIRMWARE STATEMENT

The indicated pseudo-op cannot be used in a firmware statement.

E13 MISPLACED PUNCTUATION

The assembler is expecting a microinstruction but has encountered a stray punctuation mark. Operands might have been intended for the previous microinstruction but it should not have any.

E14 LABEL PREVIOUSLY DEFINED IN EQU STATEMENT

The indicated firmware statement label has been previously defined in an EQU statement. The label is ignored.

E15 REQUIRED OPERAND MISSING

The operand list does not have the minimum number of required operands for the associated microinstruction, or an illegal null argument has been encountered.

E16 ILLEGALLY FORMED STRING

Some character is probably missing.

E17 SYMBOL TABLE FULL

The assembler cannot save any more symbols. The indicated definition or reference will go unresolved. An increase in the size using the -SIZE argument in the command line will alleviate the problem (refer to Section Five).

E18 ILLEGAL CHARACTER

The indicated character is not in the assembler's set of legal characters.

E19 ILLEGAL DIGIT

The indicated character is not legal for the radix type specified at the beginning or end of the constant.

E20 QUOTE MISSING

The terminating quotation mark (') for the indicated constant was not found; however, the constant has been accepted.

E21 ILLEGAL LEX STATE

The state number for the lexical analysis routine has been inadvertently changed. The current string, up to and including the indicated character, is ignored; the routine will try to recognize the next string starting with the next character.

Reattempt the assembly and, if the error persists, report the problem and retain the source that caused it.

E22 ILLEGAL LEX NEXT STATE

The lexical analysis routine's calculation of its next state is out of range. The current string, up to and including the indicated character, is ignored; the routine will try to recognize the next string starting with the next character. Reattempt the assembly and, if the error persists, report the problem and retain the source that caused it.

E23 EOF ENDED STATEMENT

The previous line of the indicated statement ends with a semicolon, but additional lines have not been found.

E24 SYMBOL IS ILLEGAL OPERAND

The symbol at the indicated operand position is illegal for the current microinstruction.

E25 MISPLACED + OR -

The indicated + or - is not part of a statement reference. There might be two of these characters in a row, the * may be missing, or the - might be used as the first character of a symbolic name.

E26 NA FIELD = 0 or 1

In Sequential Branching mode, the branch address cannot be equal to 0 or 1 because the hardware will inadvertently detect a disaster condition and branch to native firmware location 000.

E27 UNDEFINED SYMBOL

The indicated user symbol has not been defined as either an EQU symbol or a firmware statement label.

E28 IMAGES LIST OVERFLOW

The assembler cannot accommodate any more temporary firmware images in the remainder of the symbol table. Increase work space using the -SIZE option in the command line.

E29 VALUE ASSIGNMENT CONFLICT

The assembler cannot assemble the value of the indicated element without changing the value of at least one other bit in the firmware image. In this manner, the assembler detects illegal coding combinations. This error occurs when one or more of the restrictions summarized in Appendix D have been violated. If it is not obvious which restrictions have been violated, the user

should reference Appendix C to determine the firmware image bits that are in conflict.

E30 REFERENCED STATEMENT DOES NOT EXIST

The referenced statement (i.e., the current statement plus the specified offset) is before the first or after the last firmware statement number. This message may also appear for the last statement in the Transparent mode when the assembler attempts to set the next address field to *+1 in the absence of an explicit firmware sequencing microinstruction.

E31 F REGISTER SET AND TEST

CPU timing cycles do not permit both the copying of data into the F/SEL register and testing the data in the same firmware step.

E32 F REGISTER SET AND SPLATTER BRANCH

Splatter branching based upon the contents of the F register cannot be reliably performed if the contents of the F register are changed in the same firmware step.

E33 ALU OUTPUT TO MEGABUS

There is insufficient time in the current CPU cycle to transfer the output of the ALU (via the internal bus) to the Megabus and initiate a write cycle.

E34 AXXX DOUBLING OP. LS NOT = RS (See Appendix C)

For the basic microinstruction AXXX, the left select and right select fields must have the same values.

E36 ILLEGAL NO OPTION

The only option that may appear with pseudo-op NO is LIST.

E37 START BIT POSITION NOT BETWEEN 0 AND 63

The starting bit position for the indicated SET microinstruction is not between 0 and 63 (refer to Appendix C).

E38 INVALID BIT RANGE

The field size of the indicated SET microinstruction extends the field range beyond bit position 63 (refer to Appendix C).

E39 ONE SEQ MODE OPERAND MUST BE VALUE

One of the first two operands of a condition in the sequential mode must specify a branch address (literal, statement reference, or symbol).

E40 ONE OPERAND MUST BE NULL OR RETURN

One of the first two operands of a condition in the Sequential mode must be either null or the mnemonic RETURN.

E41 ONE NATIVE OPERAND MUST BE VALUE

Both operands of a condition in the Transparent mode may not specify XL or splatter branching.

E42 ILLEGAL BRANCH IN SEQUENTIAL MODE

The branch type specified may be used only in the Transparent branching mode.

E43 ILLEGAL BRANCH IN NATIVE MODE

The branch type specified may be used only in the Sequential branching mode.

E44 INCOMPATIBLE BRANCH ADDRESS

In the Transparent branching mode, the low order two bits for either of the two address values do not equal three, or the upper nine bits of the two address values do not equal each other.

E45 MNEMONIC IS ILLEGAL OPERAND

The indicated reserved word is not a legal operand for the current microinstruction.

E46 VALUE IS ILLEGAL OPERAND

The use of an EQU symbol, constant, or statement reference in the indicated position is illegal for the current micro op-code.

E47 STATEMENT REF IS ILLEGAL OPERAND

A statement reference cannot be used as an operand in the indicated position for the current micro op-code.

E48 MISSPELLED OPCODE

The indicated op-code was not recognized as a reserved mnemonic by the assembler.

E49 LABEL IS A RESERVED MNEMONIC

The indicated reserved mnemonic is in the label field (i.e., the first character after the line number).

E50 SYNTAX STATE ERROR

The state number for the syntax analysis routine has been inadvertently changed. The current statement is ignored; the routine will attempt to process the next statement. Reattempt the assembly and, if the error persists, report the problem and retain the source that caused it.

E51 CANNOT GENERATE GOTO *+1

In the absence of a firmware sequencing microinstruction in the Transparent mode, the assembler attempted to generate code for a GOTO *+1 statement. The attempt failed because bits in the next address field had already been set to values different than the address of *+1. Microinstructions that affect the next address field include: (1) a constant as an internal bus source, and (2) the FLOPS operands CTR0 and CTR1.

SECTION 5 - OPERATING AND SYSTEM DEBUGGING PROCEDURES

To complete development of user-generated firmware, the microprogrammer must perform three tasks: (1) assemble the source file to produce a machine-language object file, (2) load the object file or microprogram into the WCS, and (3) execute and debug the microprogram.

The information contained herein describes the procedures that are necessary to perform these tasks and includes:

- Using the WCS assembler.
- Loading the WCS.
- Debugging WCS microprograms.
- WCS patch procedure.
- Microcode Analyzer.

These procedures are written with the assumption that the reader has read and is familiar with the material contained in Sections One through Four. If not, it is recommended that the reader review this material before proceeding with Section Five.

5.1 USING THE WCS ASSEMBLER

Before a user-generated microprogram can be executed, the source file must first be assembled to produce machine-language object code that can be loaded into the WCS.

The assembly process is initiated using the Writable Control Store Assembler (WCSA) command. This command invokes the GCOS 6 writable control store assembler component, which assembles the WCS source program unit, applying the specified options.

The proper format for the WCSA command is:

```
WCSA path [ctl_arg]
```

where:

path specifies the name of the file containing the source unit to be assembled.

[ctl_arg] represents one or more control arguments chosen from the following:

-NO_OBJ or -NO indicates that the generation of the object text unit is to be suppressed. If this argument is omitted, the object text unit is written to the file path.WO.

-NO_LIST or -NL indicates that the source listing is to be suppressed. If this argument is omitted, the source listing is written to file path.L.

-LIST_ERRS or -LE specifies that the list file shall contain only those statements which have assembly errors and their associated error messages.

-SIZEnn or -SZnn specifies the number (01 through 63 decimal) of 1024-word memory blocks that are to be used for the assembler's work tables. If this argument is omitted, the assembler will request 1024 words from the task's groups memory pool.

-COUT out_path indicates that the listing which would otherwise be written to the file path.L is to be written to file out_path.

-OBJECT obj_path or -OBJ obj_path indicates that the object text unit which would otherwise be written to the file path.WO is written to file obj_path.

The path parameter can assume any of the acceptable forms of a path name; a simple name indicates that a source program unit residing in the working directory is to be assembled. The assembler appends a .WA suffix to path if it is not provided in the command line. The assembler then gets the source file path.WA. Should the search fail, the assembler drops the .WA and reattempts finding the file. In this manner, the assembler may accept input from a peripheral device such as a card reader or tape drive.

If the -COUT control argument is not specified, the source listing (if requested) is written to a file created by the assembler in the working directory, having a file name of path.L. The file can be subsequently listed by using the PRINT utility command. If a different file is specified by using the -COUT argument, out_path is the name of the file containing the listing. The assembler appends nothing to out_path.

If the -OBJECT or -OBJ arguments are not specified, the object text unit, when not suppressed, is written to a file created by the assembler in the working directory. The file name is path.WO where path is the last or only element in the path parameter. If either -OBJECT or -OBJ is given, the object text unit's file name is obj_path. No suffix is appended to obj_path.

If the listing and object files already exist, they are overwritten by the output produced by the assembler.

The following are examples illustrating the use of the WCSA command.

Example 1

```
WCSA EXTOP -SIZE 5 -COUT>SPD>LPT00
```

The writable control store assembly language source program EXTOP.WA residing in the current working directory is to be assembled. The source listing and errors are to be printed on line printer 00. The object text unit is to be written to the file EXTOP.WO in the working directory. If EXTOP.WO already exists, its contents will be overwritten by the new object text unit data. A maximum of five 1024-word blocks of memory are to be used for working tables during the assembly.

Example 2

```
WCSA > SPD>CDR00  -OBJ WCSDCK.WO
```

The card deck in card reader 00 is to be assembled. The source listing with errors is to be written to CDR00.L in the current working directory. The object text unit is to be written to file WCSDCK.WO, also in the current working directory. The card deck must be terminated by an EOF card (11-9-8-5 multipunch) and reloaded for the second pass.

5.2 LOADING THE WCS

The object code created by the WCS assembler and stored on a disk can be loaded into the WCS using the WCS loader. The loader is capable of loading multiple firmware files into the WCS, filling unused locations with an operator-supplied firmware word, dumping the contents of the WCS, and disabling the WCS (effectively disengaging the WCS from the system). It is also possible to select any WCS connected to the Megabus by specifying the channel number of the associated CPU. Each of these functions may be requested either independently of or concurrently with any other function. However, it must be understood that if "filling" the WCS is requested, it is performed after all firmware text files have been loaded and, if "dumping" is requested, it is performed after loading and filling. Disabling the WCS, if requested, will be the last option executed.

Multiple firmware files are loaded in the order in which they are named in the parameter list. No attempt is made to prevent multiple loading of the same location, but any location written to more than once during a single pass through the loader will be identified in a warning message.

Automatic loading of the WCS with a certain set of firmware text files, or automatic disabling of the WCS can be performed at startup. The operator need only modify the START_UP.EC for the system to include the canned request.

5.2.1 Writable Control Store Load (WCSLD) Command

The WCS loader is invoked using the WCSLD command, which is of the form:

```
WCSLD [path1] [path2] ... [-DUMP [([XXX],[YYY])]]
      [-FILL (xxxx,xxxx,xxxx,xxxx)] [-OFF] [-CPU (X)]
```

where:

path1 is the full or relative path name of an object text file.

-FILL (xxxx,xxxx,xxxx,xxxx) or -FL (xxxx,xxxx,xxxx,xxxx) indicates that all locations not written to in the course of loading firmware files are to be filled with the firmware word (16 hexadecimal characters separated into groups of four by commas) within the parentheses.

-DUMP [([xxx],[yyy])] or -D [([xxx],[yyy])] indicates that the contents of the WCS within the range xxx to yyy, inclusive, are to be dumped to the USER-OUT system file after loading is complete; the default range is all of the addresses in the WCS.

-CPU (X) or -CP (X) indicates that all other parameters to the loader apply to the WCS associated with the CPU on Channel X.

-OFF indicates that after all other options are performed, the WCS is to be set off-line (disabled).

NOTE

All numeric arguments are enclosed within parentheses and expressed in hexadecimal. Optional arguments are enclosed in braces.

The argument -DUMP may be followed by an optional range qualification enclosed in parentheses and of the form:

```
([xxx],[yyy])
```

where:

xxx specifies the "Start" of the range in hexadecimal.

yyy specifies the "End" of the range in hexadecimal.

Start defaults to the low address for RAMs in the indicated control store, while End defaults to the high address. For the largest possible WCS the low address is 800 and the high address is FFF. Range is specified in terms of 64-bit firmware words.

Each argument is optional, but at least one argument must be passed to the loader. The order of the arguments is immaterial, and each argument is considered independent of every other argument. For example, -DUMP might be passed as an argument more than once, each time with a different range or no range at all. If a range is used, it will be the last explicit range encountered.

5.2.2 Error Handling

The error codes that can be generated by the loader are:

ERROR CODE	MEANING
1E07	Illegal parameter (accompanied by the parameter)
1E12	No parameter.
1E13	Invalid WCS status (accompanied by the status)
1E14	No RAMs in WCS
1E15	Attempt to output out-of-range address (accompanied by the address)
1E16	Attempt to write to nonexistent address (accompanied by the address)
1E17	Fill option not honored, because format of firmware word illegal or word omitted (accompanied by the firmware word)
1E18	Text file parameter invalid, does not end in ".WO" (accompanied by the parameter)

Firmware files are assumed to have been generated by the WCS assembler, and therefore must have names ending in the suffix .WO. Loaded files are reported to the operator by the name in the title statement revision, assembly date, and 20 characters of the secondary identification; files are not identified by external name (i.e., name in directory entry). It is possible that the WCS may contain PROMs instead of RAMs, in which case the -DUMP option can be used to dump the contents of the PROM and disable the WCS. However, regardless of whether the WCS contains PROMs, or RAMs, or nothing at all, the -OFF option can be used to disable the WCS.

Since addresses may range from 800 to FFF, the loader turns on the most significant bit of any WCS address passed as part of a dump range or as an address within a firmware file. To detect an address that is out of range the loader polls the two slots on the WCS to determine the occupant of each. The loader then establishes the low and high addresses for the WCS depending on whether it finds a RAM, PROM, or neither in the slot.

Error 1E15 results from an attempt to output an out-of-range address to the WCS. Instead of outputting a bad address the loader reports the error and leaves the WCS's internal address register unchanged.

Error 1E16 results from an attempt to write to an out-of-range address, (i.e., an address beyond the highest address in the WCS). Upon encountering this error while loading firmware, the loader terminates the firmware file currently being loaded and goes on to the next firmware file.

5.3 DEBUGGING WCS MICROPROGRAMS

This subsection suggests methods for finding and correcting errors in user-generated firmware. The principal test tool for this activity is the Microcode Analyzer (refer to subsection 5.5). The techniques described herein are intended to help the user get started in the debugging activity. The detailed actions taken after this initial effort depend heavily on the precise nature of the user firmware and the fault being pursued, and are necessarily left to the ingenuity of the user.

It is asssumed that the user has assembled and loaded his firmware, has exercised it by embedding appropriate User Generic instructions in the software, and suspects that one of the following conditions exists:

- The User Generic firmware has not been entered.
- The User Generic firmware has been entered, and does not exit.
- The User Generic firmware has exited to report an unexpected trap condition.
- The User Generic firmware produces unexpected results.

5.3.1 User Generic Not Invoked

If the user suspects this is the case, he can confirm or eliminate the possibility by setting into the Microcode Analyzer a firmware-address halt at the entry ("splash") point for the instruction in question. This address is 800# plus the least significant digit of the instruction word. If, when the program is run, the CPU does not stop at the splash point, attention should be directed to the software to determine the reason.

NOTE

When a firmware-address halt is invoked, the central CPU clock stops after execution of the specified step. The "current" display on the Microcode Analyzer will contain the internal bus value generated in the next step and the address of the step after that. When selecting the firmware address at which to set the halt, it is important to choose one that is not followed by a Megabus read request or write operation that might address an unavailable resource because the Megabus time-out could override the address halt.

A firmware-address halt set at native firmware splatter location 1CO# stops the CPU upon recognition of every user generic instruction, if this is desired. Stepping the CPU clock forward from this point permits examination of the splash action. If the native firmware branches to the trap algorithm (location 33B#) instead of entering the WCS code, the user should consider the possibility that the WCS is not on-line (i.e., has not been written into since its last initialization).

5.3.2 Instruction Does Not Exit

If the control panel becomes unresponsive (register selection ineffective, etc.), the firmware has probably encountered one of three situations:

- The clock is "stalled", waiting for a response to a Megabus read request.
- Execution of the native trap firmware has triggered the unavailable resource response, resulting in an infinitely recursive trap.
- The user firmware has entered a loop in which exit conditions cannot be satisfied.

To distinguish among these situations, the user should place the Microcode Analyzer in the Step mode. If the RUN indicator on the Microcode Analyzer remains illuminated, the clock is stalled. In this case, the hexadecimal displays on the Microcode Analyzer are not meaningful. To determine the firmware location of the stall (and the firmware flow which led to it), first depress the Stall Examine (E) key to extinguish the RUN indicator. The firmware history can then be explored for possible occurrences of:

- Use of data from Megabus without a prior request.
- Use of data from Megabus when the preceding request was rejected.

If the RUN indicator extinguishes when the Step mode is entered, a loop condition exists. Repeated actuation of the Execute key on the Microcode Analyzer will permit exploration of the nature and extent of the loop. The user must consider that the failure of the firmware to exit from the loop is probably due to some action or condition established prior to entering the loop. Further investigation might benefit from restarting the program after establishing a firmware address halt at or near the entry to the loop. When the CPU stops at this point, firmware history should help explain the cause of the problem. In selecting the firmware address at which to set this halt, the following considerations may be helpful. If the loop is contained in WCS firmware, the user should employ his knowledge of that firmware to choose a point near the loop entry, avoiding (if possible) those shared by any prior, nonfaulty, executions of the code.

If the loop includes native location 33B#, a recursive trap is involved. Location 33B# can then be used as the firmware-address halt location, but this should be avoided, if possible, because of the frequency with which various traps are normally invoked by the operating system. Usually, native location 003# is also included in the firmware loop and can be used as a more selective point at which to set the firmware-address halt.

5.3.3 Instruction Exits Via Unexpected Trap

If the instruction causes an unexpected trap, the techniques described in the preceding subsection can be used to facilitate retracing the steps leading to the trap exit. Note that, in general, when the firmware history of 16 steps is inadequate to identify the beginning of the problem, it can be used to select another firmware-address halt setting 16 steps earlier. Following a restart of the program, this will provide 16 steps of earlier history, and so on back to the original fetch of the instruction.

Unexpected (i.e., hardware-detected) traps generally result from any one of the following five causes:

1. Reference to a resource which is not available on the Megabus.
2. Reference to a non-existent or invalid memory location.
3. Violation of memory protection.
4. Detection of a parity error in data received via the Megabus.
5. Receipt of data from memory with an error not correctable by EDAC.

It should be obvious that the first three causes may arise from either software or firmware errors, as well as from deliberate actions. It is less obvious that uncorrectable memory errors can also be induced by improper firmware coding, which violates timing requirements in the CPU - Megabus interface when the memory is being written into. Such mistakes are flagged by the WCS Micro Assembler, and should always be corrected before attempting to execute the assembled code.

5.3.4 Instruction Executes and Produces Unexpected Results

If the instruction and the program containing it appear to execute and exit normally, but the results are not those that were anticipated, the user must devise techniques for analyzing the problem. These will depend on the functional definition of the instruction, the algorithm used to implement it, and the nature of the fault syndrome. In this case, the internal bus display on the Microcode Analyzer, as well as the firmware address pattern, will be useful. To obtain maximum information from this display, it is good microprogramming practice to employ the internal bus in all firmware steps. When use of the internal bus is not required by the functional activity in a given firmware step, a source should be used which is likely to assist the debugging effort. For this reason, the default code generated by the Assembler in such firmware steps is defined to display the activity within the microprocessor.

Another technique that may prove useful in debugging firmware functionality is the modification of the firmware and/or insertion of firmware patches to test or display register contents and other conditions of interest. For this purpose, the user is referred to descriptions of the On-Line Editor, the WCS Assembler, and the WCS Loader for the tools necessary to modify, reassemble, and load firmware patches.

5.4 WCS PATCH PROCEDURE

The WCS user, while debugging his microprogram, may want to alter (patch) the contents of selected firmware words. This patching is most easily accomplished by exiting WCS execution and invoking the text editor. The source file is then corrected and reassembled. The new object code is loaded into the WCS, replacing only those addresses specified in the source. An Execute Command (EC) file might be created to run through the Editor, WCS Assembler, and Loader while permitting interactive update in the editor. Given relatively short source files, this turnaround is fairly quick.

Care should be taken in the above procedure during loading. In a situation where multiple object files are loaded, the design may rely upon the loader's loading all references to a firmware word. Therefore, the last reference for each firmware word is used. If the updated selected source is overlapped by other routines in the total load, a full reload may be required.

For those applications where the source size adversely affects turnaround time, a second procedure may be adopted. Here, as before, the corrections are noted in the source listing. The editor is invoked and the corrected statements are entered from the keyboard or removed selectively from the source. The addresses for these statements must be specified to guarantee proper firmware modification. Labels used in branching may now be undefined and should either be replaced by constants or defined by LABEL statements. A special file may be added to the edit file to provide the correct DEFAULT statement and Branching mode. The edited file is then written to a new file.

After exiting the editor, the source file must be assembled, with all errors being either noted or corrected. Acceptable object code is then loaded into the WCS by the WCS loader, affecting only those locations specified in the source file. The WCS is now loaded and testing may continue.

NOTE

> The WCS loader accepts multiple object files. Thus, if the WCS memory has been altered or its contents are at least questionable, the update may still take place, with the new object file being appended to the list of object files previously loaded. Since the new object file will be loaded last, its changes will be used.

5.5 MICROCODE ANALYZER

The Microcode Analyzer is a tool designed to assist the user during microprogram debug activities as previously described in subsection 5.3. It allows the user to display addresses and other useful data, while providing addditional facilities to assist the debugging effort. The material contained herein describes these facilities.

5.5.1 Front Panel

The front panel, shown in Figure 5-1, consists of a hexadecimal keyboard, eight HEX displays, and several miscellaneous switches and indicators. The keyboard input (shown at the right) is a regular hexadecimal keyboard.

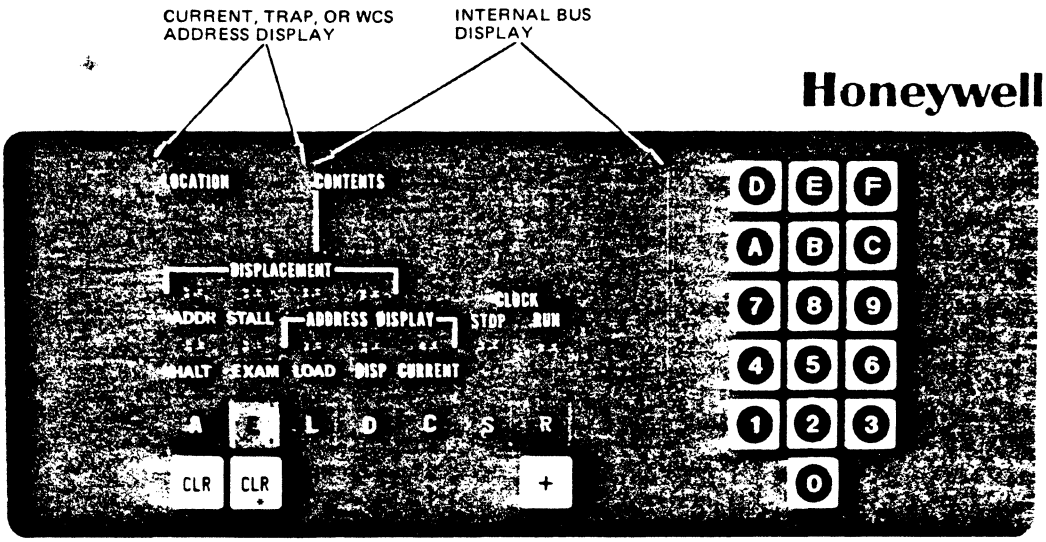


Figure 5-1 Front Panel

5.5.1.1 Front Panel Keys

The front panel provides the user with a total of 16 keys (excluding the keypad) that control various analyzer functions. The function of each key is as follows:

1. S key - places the CPU master clock in Single Step mode and illuminates the STOP indicator.

2. R key - readies the CPU master clock and illuminates the RUN indicator.

3. + key - if the STOP indicator is illuminated, it produces a single clock pulse; otherwise, it starts the master clock for continuous operation (or until the next address halt).

4. C key - displays the current internal bus value in the five rightmost hexadecimal displays, the next address in the three leftmost hexadecimal displays, and illuminates the CURRENT indicator.

5. D key - displays one of the previous 16 current addresses and illuminates the DISP indicator; selection of the previous address is performed using the keypad.

6. L key - displays the halt address in the three leftmost hexadecimal displays.

7. E key - if the STOP indicator is illuminated, it releases the clock stall to allow the display to function.

8. A key - causes the analyzer to halt when the current address equals the halt address and illuminates the ADDR HALT indicator.

9. CLR (halt) key - terminates the operation invoked by the A key described previously.

5.5.1.2 Front Panel Indicators

The front panel provides the user with a total of 11 indicators that specify the current mode of operation. Each of these indicators is defined below.

1. ADDR HALT indicator - indicates that the A key has been depressed.

2. LOAD indicator - indicates that the L key has been depressed.

3. DISPLACED (DISP) indicator - indicates that the D key has been depressed.

4. CURRENT indicator - indicates that the C key has been depressed.

5. STOP indicator - indicates that the S key has been depressed.

6. RUN indicator - indicates that the R key has been depressed.

7. DISPLACEMENT indicators - indicates the displacement of a previous next address from the current next address.

5.5.1.3 Internal Bus Display

The rightmost five hexadecimal displays show the 20 bits from the internal bus.

In addition to the current BI field display, it is possible to display previous history. To permit this, the last 16 firmware steps are stored and may be displayed by front panel manipulation.

5.5.1.4 Address Display

The leftmost three hexadecimal displays show the address of the firmware step to be executed next. This could be from the native PROM or from the WCS. These three displays are also used to display halt addresses.

Native PROM Display

When this is displayed, the leftmost digit of the address hexadecimal displays is seven or less. The maximum number in this case is 7FF.

WCS Display

When this is displayed, the number shown is a minimum of 800 and a maximum of FFF.

Halt Address

To display a Halt address, depress the L key, and the LOAD indicator should illuminate. The halt address may now be changed by entering the desired value, digit by digit, using the keypad.

5.5.2 Normal Operation

During normal operation the displays are continuously being updated at approximately six megacycles.

5.5.2.1 Operate in Single Step Mode

To operate in Single Step mode, perform the following steps:

1. Depress the S key (the CLOCK STOP indicator should illuminate).

2. Depress the + key for each clock cycle.

5.5.2.2 Return to Continuous Operation

To return to continuous operation perform the following steps:

1. Depress the R key (the CLOCK RUN indicator should illuminate).

2. Depress the + key to start the clock.

5.5.2.3 Set Up a Halt Address

To set up a halt address perform the following steps:

1. Stop the clock.

2. Depress the L key (the LOAD indicator should illuminate).

3. Load the address using the hexadecimal keypad. As each digit is loaded the register content is shifted left.

4. Restart the clock by first depressing the R key, then the + key.

NOTE

Steps 1 and 4 may be omitted at the expense of not visibly displaying the address entered.

5.5.2.4 Halt at a Particular Address

To halt at a particular address, perform the following steps:

1. Set up the address (refer to subsection 5.5.2.3).

2. Depress the A key (the ADDR HALT indicator should illuminate).

3. The system halts after execution of the firmware step at the halt address.

NOTE

When a firmware-address halt is invoked, the central CPU clock stops after execution of the specified step. The "current" display on the Microcode Analyzer will contain the internal bus value generated in the next step and the address of the step after that. When selecting the firmware address at which to set the halt, it is important to choose one that is not followed by a Megabus read request or write operation that might address an unavailable resource because the Megabus time-out could override the address halt.

5.5.2.5 Disable Address Halt

To disable address halt, depress the CLR key (located under the A key).

5.5.2.6 Display the Current Data

To display the current data, depress the C key.

5.5.2.7 Display History

To display history perform the following steps:

1. Depress the D key (the DISP indicator should illuminate).

2. Set up the required displacement using the hexadecimal keypad (the displacement indicators will confirm the entry in binary code).

   The hexadecimal display will show the next address and the BI bits corresponding to the displacement. A displacement of zero causes the sixteenth previous history step to be displayed.

5.5.2.8 Synchronize Oscilloscope

To synchronize an oscilloscope perform the following steps:

1. Set up address (refer to subsection 5.5.2.3).
2. Depress CLR key (located under the A key).
3. Coaxial connector (located at rear of analyzer) supplies a sync pulse every time the selected step is accessed.

SECTION 6 - PROGRAMMING CONSIDERATIONS

Those users that do not wish to use the Honeywell operating system for loading the WCS must develop their own software loader. The material presented in this section is intended to provide the user with the background information necessary to perform this task, and includes:

- A description of the logical and physical layout of a WCS assembler object deck.
- A description of the program steps required to load the firmware image into the WCS.
- A definition of User Generics and their relationship to WCS entry points.

6.1 LOGICAL AND PHYSICAL LAYOUT

A description of the logical layout for the WCS assembler object deck is contained in subsection 4.5. The description of the physical layout for files is contained in the GCOS 6 Data File Organizations and Formats Manual (Order Number CB05).

6.2 LOADING FIRMWARE IMAGE INTO WCS

Loading the firmware into the WCS requires communication over the Megabus to the WCS from its associated CPU. During these Megabus transfers, the Megabus address lines will carry the channel number of the WCS.

The channel number assignment is directly related to the CPU to which the WCS is attached because of their committed association. The channel number of the WCS for each CPU is:

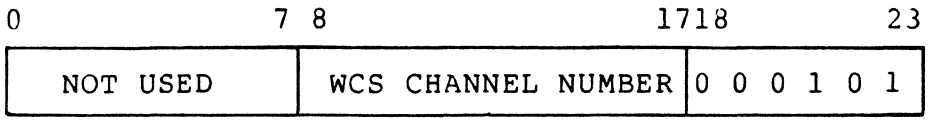
CHANNEL NUMBER	
CPU	WCS
0000	03C0
0040	0380
0080	0340
00C0	0300

Since the WCS channel numbers are less than 0400#, the I/O instructions used to communicate with the WCS must employ a non-procedural form for the channel-control word (refer to the Honeywell Level 6 Minicomputer Handbook - Order Number AS22).

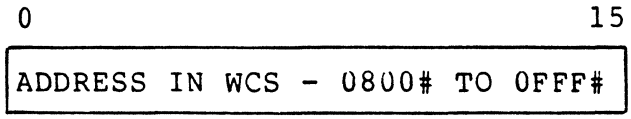
The two instructions necessary to load the firmware image into the WCS are the Output WCS Address and Output WCS Data instructions.

Output WCS Address

This instruction selects the WCS RAM starting location for data to be loaded into the WCS. The format of the Megabus address and data lines during the Output WCS Address instruction are:



ADDRESS BUS



DATA BUS

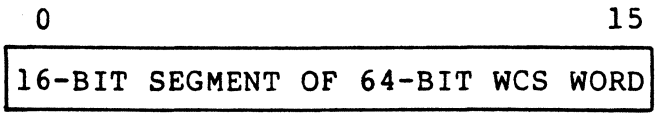
The information for the address bus is pointed to by the Control Address Syllable, while the information for the data bus is pointed to by the Data Address Syllable.

Each 64-bit word in the WCS is divided into four 16-bit segments. When an address is transferred to the WCS, the first segment (i.e., the most significant 16 bits) is selected to receive the data from the next succeeding Output WCS Data instruction. Subsequent Output Data instructions increment the segment pointer and the control store address so that only the Output WCS Address instruction is necessary to load any number of contiguous locations in the WCS. Four Output Data instructions are necessary to load each 64-bit firmware word.

Output WCS Data

This instruction loads a 16-bit segment of the WCS location selected by the Output WCS Address instruction. The format of the Megabus address and data lines during the Output WCS Data instruction are:

ADDRESS BUS



DATA BUS

The information for the address bus is pointed to by the Control Address Syllable, while the information for the data bus is pointed to by the Data Address Syllable.

The segment pointer is incremented after each data transfer so that a subsequent data transfer loads the next 16-bit segment of the same WCS location. When four data transfers have occurred, the control store address is incremented to select the next WCS location. Noncontiguous WCS locations can be loaded by issuing intermediate Output WCS Address instructions at the beginning of the data transfer for each firmware word.

Typical Loading Sequence

Figure 6-1 illustrates a typical loading sequence. The four Output Data I/O instructions are identical and are shown only to illustrate the necessity of testing for completion after every fourth transfer.

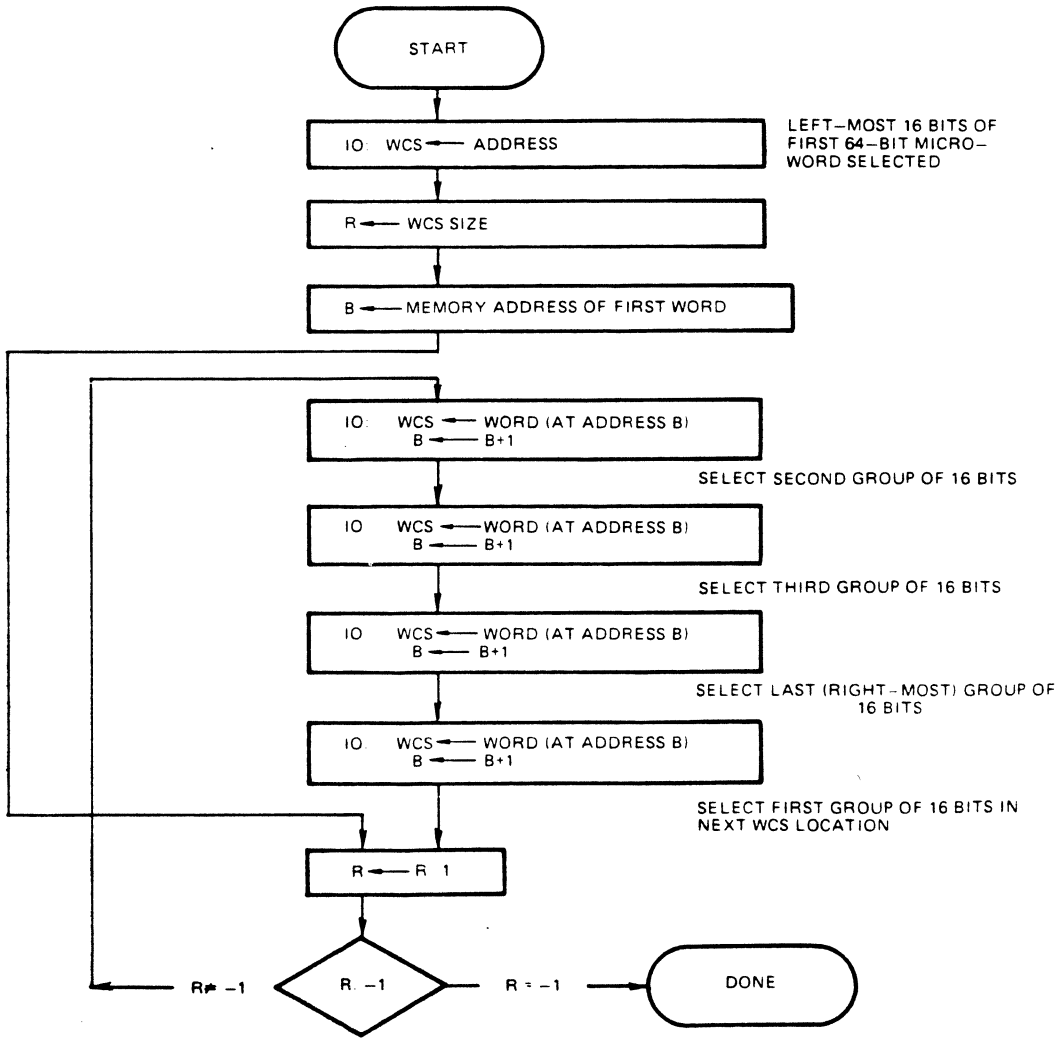
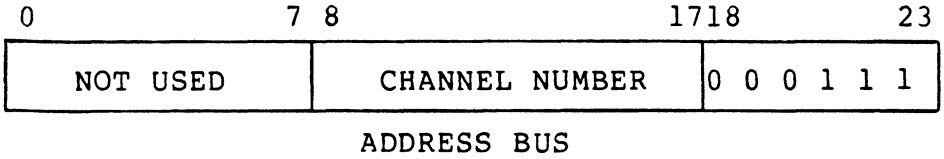


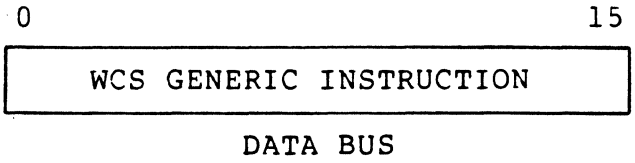
Figure 6-1 Typical Loading Procedure

The memory addresses for the microcode transfers are unrestricted with regard to sequence or contiguity. Address forms need only comply with the restrictions imposed by the Data Address Syllable (DAS) forms permitted in the I/O instructions of the CPU.

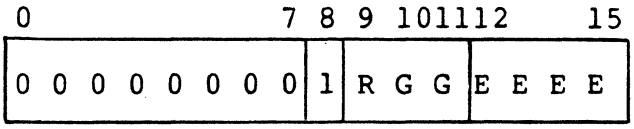
6.3 RELATIONSHIP OF USER GENERICS TO WCS ENTRY POINTS

The transfer of control from a user's program in memory to a user-generated firmware routine in the WCS is accomplished using the WCS Generic instruction. When the CPU encounters a WCS Generic instruction while processing a program, it sends the following Output Task Word command to the WCS via the Megabus.





The first word of the Generic instruction from the program in memory is transferred, over the Megabus data lines, to the WCS, and is of the form:



where:

Bits 0 through 7 are not used (all Zeros).

Bit 8 equals One.

Bit 9 (R) equals Zero and is reserved for future use.

Bits 10 and 11 (GG) are the auxiliary command code bits. These bits are not decoded but can be tested by the microprogram.

Bits 12 through 15 (EEEE) select one of the first 16 locations in the WCS.

This instruction word selects a specific entry point within the first 16 WCS locations. The selected entry point is the starting address of the user-generated firmware routine.

The Generic instruction codes (op-codes) range from 0080 through 00BF. Table 6-1 lists each of the available op-codes and the corresponding WCS entry point. The op-code is also stored in RAM location 0 and the F/SEL register (refer to subsections 2.8.2 and 2.8.3, respectively). RAM0 and the F/SEL register can be used for further firmware branching to distinguish among up to four op-codes per WCS entry point.

Table 6-1 WCS Entry Points

GENERIC INSTRUCTION (OP-CODE)	WCS ENTRY POINT
0080, 0090, 00A0, 00B0	800
0081, 0091, 00A1, 00B1	801
0082, 0092, 00A2, 00B2	802
0083, 0093, 00A3, 00B3	803
0084, 0094, 00A4, 00B4	804
0085, 0095, 00A5, 00B5	805
0086, 0096, 00A6, 00B6	806
0087, 0097, 00A7, 00B7	807
0088, 0098, 00A8, 00B8	808
0089, 0099, 00A9, 00B9	809
008A, 009A, 00AA, 00BA	80A
008B, 009B, 00AB, 00BB	80B
008C, 009C, 00AC, 00BC	80C
008D, 009D, 00AD, 00BD	80D
008E, 009E, 00AE, 00BE	80E
008F, 009F, 00AF, 00BF	80F

Appendix A

Writable Control Store Assembler Abort Codes (xx1D)

When the assembler detects an error that prevents completion of the assembly process, an appropriate "abort" code is generated, informing the operator of the error condition. The abort codes currently in use are as follows:

xx1D07 arg INVALID CONTROL ARGUMENT

The specified control argument is unrecognized. Reenter the command using a valid control argument.

xx1D0B INVALID -SIZE ARGUMENT

The value specified in -SIZE (-SZ) is zero, greater than 63, or nonnumeric.

xx1D0C FILE NAME NOT DESIGNATED

The source file name is missing. Reenter the command using the file name.

xx1D13 OBJ_PATH ARGUMENT IS MISSING

The OBJ_PATH ARGUMENT (object unit name) following -OBJECT (-OBJ) is missing or the argument list is too short. Reenter the command using the correct argument.

xx1D14 OUT_PATH ARGUMENT IS MISSING

The OUT_PATH ARGUMENT (output listing file name) following -COUT is missing or the argument is too short. Reenter the command using the correct argument.

xx1D17 SOFTWARE ERROR - PLEASE RERUN

An illegal pseudo-op-code number has been received. Rerun the Assembler. If the error persists, please contact Honeywell.

xx1D18 SOFTWARE ERROR - PLEASE RERUN

An illegal nonencoding token type has been detected. Rerun the Assembler. If the error persists, please contact Honeywell.

Appendix B

Firmware Word Format

The format of the 64-bit firmware word is illustrated in Figure B-1.

As shown in Figure B-1, the firmware word is divided into 15 fields: AD, AF, AS, BI6, BR, BS, C, CK, DI, GP, LS, NA, RS, SM, and TC. Combinations of these fields control different portions of the CPU hardware.

AD Field (Bits 9 through 11)

The AD field controls:

- The destination of the ALU result within the microprocessor area
- The microprocessor area shift logic
- The availability of microprocessor area elements to the internal bus.

When the AD field specifies a destination within the register file, the RS/SM fields interact to select the specific register file location that receives the ALU result - RF(R). When the AD field makes a register file location available to the Internal Bus (BI), the LS/SM fields interact to select the specific register file location that is made available - RF(L). Table B-1 lists the operations performed for each decode of the AD field.

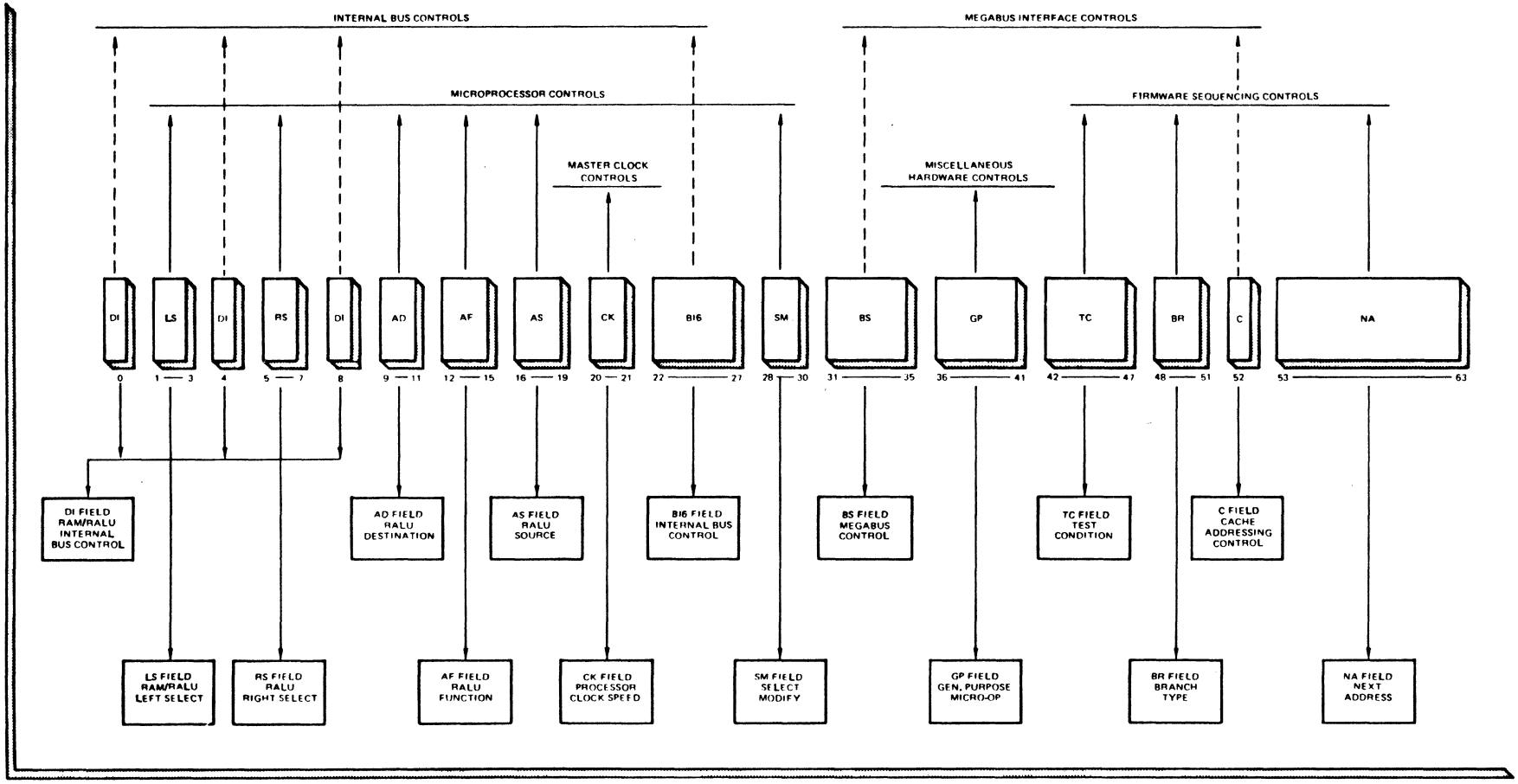


Figure B-1 Firmware Word Format

Table B-1 RALU Destination (AD) Field Decodes

AD DECODE	OPERATION PERFORMED
0	Q ← ALU result ALU result made available to BI
1	ALU result made available to BI
2	RF(R) ← ALU result RF(L) made available to BI
3	RF(R) ← ALU result ALU result made available to BI
4	DR (double word right shift) ALU result made available to BI
5	SR (single word right shift) ALU result made available to BI
6	DL (double word left shift) ALU result made available to BI
7	SL (single word left shift) ALU result made available to BI

AF Field (Bits 12 through 15)

The AF field controls:

- The function performed by the ALU
- Changes in MMU operation.

The ALU has two input ports called J and K. The data available to these ports is controlled by the AS field. Table B-2 lists the operations performed for each decode of the AF field.

Table B-2 RALU Function (AF) Field Decodes

AF DECODE	OPERATION PERFORMED
0	J+K
1	K-J-1
2	J-K-1
3	J V K
4	Undefined
5	K ∧ $\overline{J}$
6	J ⊻ K
7	$\overline{J \veebar K}$
8	J+K+1
9	K-J
A	J-K
B	J V K RINGINIT
C	J ∧ K
D	K ∧ $\overline{J}$ NONPROC
E	J V K RINGCALC
F	$\overline{J \veebar K}$ DDLEQ0 RINGCALC

AS Field (Bits 16 through 19)

The AS field controls:

- The inputs to the ALU
- Whether CRY, OVFL, and AUZ are generated on 16- or 20-bit results
- XB(0) when shifting the XB register.

AS bits 1 through 3 control the data made available to the ALU input ports (J and K). If data is sourced from the register file, the specific register file location is selected by the interaction of the LS/SM fields and/or the RS/SM fields. Table B-3 lists the data made available to these ports for each decode of AS bits 1 through 3.

Table B-3 RALU Source (AS) Field Decodes

AS DECODE (BITS 17-19)	ALU INPUT SOURCE	
	J PORT	K PORT
0	RF(L)	Q
1	RF(L)	RF(R)
2	ZERO	Q
3	ZERO	RF(R)
4	ZERO	RF(L)
5	BI	RF(L)
6	BI	Q
7	BI	ZERO

Definitions

RF(R) - Denotes RS/SM Field Interaction
RF(L) - Denotes LS/SM Field Interaction

If AS(0) equals One, functions CRY, OVFL, and AUZ are generated on a 16-bit result. If AS(0) equals Zero, functions CRY, OVFL, and AUZ are generated on a 20-bit result.

If AS(0) and LS(0) both equal Zero, RF(L) is treated as a 16-bit quantity sign-extended to 20 bits, and functions CRY, OVFL, and AUZ are generated on a 20-bit result. Table B-4 lists the only legal combinations of the AS and AF fields for this case.

Table B-4 Legal Combinations of AS and AF Fields

AS DECODE	AF DECODE	OPERATION PERFORMED
1	6	ALU result ← $RF(L)_{SE}$ *2
2	0	ALU result ← $RF(L)_{SE}$ +Q
2	8	ALU result ← $RF(L)_{SE}$ +Q+1
3	0	ALU result ← $RF(L)_{SE}$ +RF(R)

NOTE

$RF(L)_{SE}$ = RF(L) sign-extended to 20 bits by replacing the four most significant bits with copies of the SIGN flop.

When shifting the XB register (GP Field = 04, 05, 09, 0A, 14 or 15), AS(0) controls what is received in XB(0). If AS(0) equals One, XB(0) receives bit 19 of the ALU result. If AS(0) equals Zero, XB(0) receives the complement of RS(0).

BI6 Field (Bits 22 through 27)

The BI6 field controls:

- Most internal bus sources
- Data received by I register (or bits thereof).

Table B-5 lists the data received by the internal bus (BI) or the I register for each decode of the BI6 field.

Table B-5 Internal Bus Selector (BI6) Decodes
(Sheet 1 of 2)

BI6 DECODE	OPERATION PERFORMED
0z	BI(0-11)← 0 BI(13-15)← NA(3-6) BI(16-19)← 0z
1z	BI(0-3)← 0 BI(4-11)← FF BI(12-15)← NA(3-6) BI(16-19)← z
20	BI srcmod = R8 (DI field must = 5)
21	BI← IDSy where y = NA(3-6)
22	BI← HL8
23	BI← BD or BP (if made available by BS field) BI← RUP (if BD, BP not made available)
24	BI← P or Y or MMU (as made available by BS field)
25	BI← BDH or BPH (if made available by BS field) BI← Panel (if BD and BP not made available)
26	BI srcmod = L4 (DI field must = 0)
27	BI← LVL
28	Undefined
29	BI← IDCy where y = NA(3-6)
2A	BI← H (bytes swapped)
2B	BI← Z

Table B-5 Internal Bus Selector (BI6) Decodes
(Sheet 2 of 2)

BI6 DECODE	OPERATION PERFORMED
2C	Undefined
2D	BI ← XBHEX
2E	BI ← I
2F	BI ← S
30	No action
31	I ← BI(12-19)
32	I(OV) ← 1 if BI(4) = BI(5); else, unchanged
33	I(OV) ← OVFL
34	Undefined
35	I(I) ← ACK
36	I(B) ← $\overline{\text{AUZ}}$
37	I(B) ← BI(4) I(C) ← CRY I(O) ← OVFL
38	I(G) ← $\overline{\text{BI(4) V AUZ}}$ I(L) ← BI(4)
39	I(G) ← $\overline{\text{ALU Result (0) V AUZ}}$ I(L) ← ALU Result (0)
3A	I(G) ← $\overline{\text{SIGN}}$ I(L) ← SIGN I(U) ← BI(4)
3B	I(C) ← CRY I(O) ← OVFL
3C	I(C) ← Q(19) - AD field must be 4 or 5
3D	I(C) ← BI(19)
3E	I(C) ← BI(4)
3F	I(C) ← CRY

BR Field (Bits 48 through 51)

The BR field controls the type of branching to be performed, depending on the current addressing mode (Transparent or Sequential). However, in either case, the effect of this field depends on the contents of the Test Condition (TC) and Next Address (NA) fields, Tables B-6 and B-7 list the types of branching performed in Transparent mode and Sequential mode (respectively) for each decode of the BR field.

Table B-6 Branch Type (BR) Field Decodes (Transparent Mode)

	BRANCH TYPE	
BR DECODE	TEST CONDITION TRUE	TEST CONDITION FALSE
0	→NA V 3	→NA
1	→XL	→NA
2	→XA Splatter	→NA
3	→XB Splatter	→NA
4	→XR Splatter NEWXR←0	→NA
5	→XW Splatter NEWXR←0	→NA
6	→XE Splatter NEWXR←0	→NA
7	→XF (location 020) NEWXR←0	→NA
8	→NA	→NA V 3
9	→NA	→XL
A	→NA	→XA Splatter
B	→NA	→XB Splatter
C	→NA	→XR Splatter NEWXR←0
D	→NA	→XW Splatter NEWXR←0
E	→NA	→XE Splatter NEWXR←0
F	→NA	→XF (location 020) NEWXR←0

Table B-7 Branch Type (BR) Field Decodes (Sequential Mode)

BR DECODE	BRANCH TYPE	
	TEST CONDITION TRUE	TEST CONDITION FALSE
0	→NA	→CSAC
2	→NA	→CSRAR
4	→NA CSRAR←CSAC	→CSAC
8	→CSAC	→NA
A	→CSRAR	→NA
C	→CSAC	→NA CSRAR←CSAC
F	(TC must = 0)	LBRANCH

NOTE

The Control Store Address Counter (CSAC) contains the address of the current firmware step + 1. The Control Store Return Address Register (CSRAR) contains the address for subroutine returns.

BS Field (Bits 31 through 35)

The BS field controls:

- The Megabus interface area
- The availability of BD, BP, P, Y, and MMU as internal bus sources
- P and Y as internal bus destinations
- CTR.

Table B-8 lists the operations performed for each decode of the BS field.

Table B-8 Bus Control (BS) Field Decodes
(Sheet 1 of 3)

BS DECODE	OPERATION PERFORMED
00	BUS PSELECT P available as BI source
01	BUS PURGE P available as BI source
02	BUS YSELECT or BUS MMURDACC Y available as BI source
03	BUS INCY Y available as BI source
04	BUS PSELECT P available as BI source Y ← BI
05	BUS INCP P available as BI source IF GP field = 4, 5, 6, 8, 9, A, 14, or 15 CTR ← NA (1); else, CTR ← CTR+1
06	BUS MMUSELECT MMU available as BI source Y ← BI
07	BUS MMUWRACC Y available as BI source
08	BUS YSELECT Y available as BI source Y ← BI
09	BUS YSELECT Y available as BI source P ← BI
0A	BUS YSELECT Y available as BI source YRELOC ← BI
0B	BUS YSELECT Y available as BI source YR16 ← BI
0C	BD is BI source Y ← BI
0D	BUS INCY BD is BI source

Table B-8 Bus Control (BS) Field Decodes
(Sheet 2 of 3)

BS DECODE	OPERATION PERFORMED
0E	BD is BI source P ← BI
0F	BD available as BI source
10	RDREQ CHGLOCK Y available as BI source
11	RDREQ NORMAL (if C = 1) RDREQ NOCACHE (if C = 0) Y available as BI source
12	RDREQ I-O Y available as BI source
13	Undefined
14	WRTWORD INCY (if CK(0) = 0) WRTBYTE INCY (if CK(0) = 1) Y available as BI source
15	WRTWORD CHGLOCK (if CK(0) = 0) WRTBYTE CHGLOCK (if CK(0) = 1) Y available as BI source
16	WRTWORD I-O, INCY (if CK(0) = 0) WRTBYTE I-O, INCY (if CK(0) = 1) Y available as BI source
17	WRTWORD I-O (if CK(0) = 0) WRTBYTE I-O (if CK(0) = 1) Y available as BI source
18	Undefined
19	RDREQP P available as BI source
1A	RDREQP Y available as BI source P ← BI
1B	Undefined
1C	BP is BI source P ← P + 1 Y ← BI If GP field = 4, 5, 6, 8, 9, A 14, or 15 CTR ← NA(1); else, CTR ← CTR+1

Table B-8 Bus Control (BS) Field Decodes
(Sheet 3 of 3)

BS DECODE	OPERATION PERFORMED
1D	BP is BI source P ← P+1 If GP field = 4, 5, 6, 8, 9, A, 14 or 15 CTR ← NA(1); else, CTR ← - CTR+1
1E	Undefined
1F	Undefined

C Field (Bit 52)

The C field controls whether or not the Cache, if installed, is to be used during read operations. In general, this bit must be Zero. For procedural read requests (BS field equals 19, 1A, 1C, or 1D), bit 52 must be a One. For data read requests (BS field equals 11), bit 52 will be a One if the Cache is being used, and Zero if it is being bypassed.

CK Field (Bits 20 and 21)

The CK field controls:

- The firmware clock speed during nonwrite operations
- Whether a byte or word is written during write operations.

Table B-9 lists the clock speeds for each decode of the CK field.

Table B-9 CPU Clock Speed (CK) Field Decodes

CK DECODE	CLOCK SPEED
0	Very Long
1	Half Long
2	Half Fast
3	Very Fast

NOTES

1. If bit 20 equals Zero during a Megabus write operation, a word is written.

2. If bit 20 equals One during a Megabus write operation, a byte is written.

DI Field (Bits 0, 4, and 8)

The DI field controls:

- The RAM as an internal bus source and destination
- Whether the microprocessor area output made available to the internal bus by the AD field is used as an internal bus source.

When the RAM is an internal bus source or destination, the specific RAM location is selected by the interaction of the LS/SM fields. Table B-10 lists the operations performed by each decode of the DI field.

Table B-10 Internal Bus Control (DI) Field Decodes

DI DECODE	OPERATION PERFORMED
0	BI SRCMOD is L4 (BI6 field must = 26)
1	BI ← microprocessor output
2	BI ← microprocessor output RAM(L) ← BI
3	Undefined
4	No action
5	BI SRCMOD is R8 (BI6 field must = 20)
6	RAM(L) ← BI
7	BI ← RAM(L)

GP Field (Bits 36 through 42)

The GP field controls:

- The GP category of miscellaneous hardware
- Certain internal bus destinations.

Table B-11 lists the operations performed for each decode of the GP field.

Table B-11 General Purpose (GP) Field Decodes
(Sheet 1 of 4)

GP DECODE	OPERATION PERFORMED
00	No action
01	H←BI SIGN←BI(4)
02	SIGN←BI(4) ZERO←$\overline{\text{AUZ}}$ MISC←$\overline{\text{BI(19)}}$
03	F(8-11)←BI(12-15) SEL←BI(16-19) SIGN←1 NEWXR←1
04	XB(1-3)←XB(02) XB(0) controlled by AS(0) SHIN2←0 SHIN1←0
05	XB(1-3)←XB(0-2) XB(0) controlled by AS(0) ZERO←1
06	F←BI(4-15) SEL←BI(16-19) XB←0 NEWXR←1
07	SEL←BI(16-19) SHIN2←1 SHIN1←1 NEWXR←1
08	F←BI(4-15) SEL←BI(16-19) ZERO←AUX SHIN2←0 SHIN1←0 XB←0 MISC←0 NEWXR←1
09	XB(1-3)←XB(0-2) XB(0) controlled by AS(0) SHIN2←0 SHIN1←1 MISC←0

Table B-11 General Purpose (GP) Field Decodes
(Sheet 2 of 4)

GP DECODE	OPERATION PERFORMED
0A	XB(1-3)←XB(0-2) XB(0) controlled by AS(0) ZERO←0
0B	F(8-11)←BI(12-15) SEL←BI(16-19) MISC←[BI(4-9) = 0] SIGN←BI(19) NEWXR←1
0C	SIGN←BI(4) MISC←1
0D	SIGN←BI(19) MISC←1
0E	SIGN←BI(19) ZERO←QLT active flop
0F	Undefined
10	SHIN2←$\overline{\text{SIGN}}$
11	SHIN2←$\overline{\text{SIGN}}$ SIGN ←BI(4)
12	SHIN1←I(B)
13	SIGN←1
14	XB(1-3)←XB(0-2) XB(0) controlled by AS(0)
15	XB(1-3)←XB(0-2) XB(0) controlled by AS(0) SIGN←BI(4)
16	ZERO←0
17	ZERO←1
18	ZERO←AUZ
19	SIGN←BI(4)
1A	SIGN←BI(0)
1B	SIGN←BI(19)

Table B-11 General Purpose (GP) Field Decodes
(Sheet 3 of 4)

GP DECODE	OPERATION PERFORMED
1C	SHIN2 ← 0 SHIN1 ← 0
1D	SHIN2 ← 0 SHIN1 ← 1
1E	SHIN2 ← 1 SHIN1 ← 0
1F	SHIN2 ← 1 SHIN1 ← 1
20	F ← BI(4-15) SEL ← BI(16-19) NEWXR ← 1
21	Undefined
22	F(8-11) ← BI(12-15) SEL ← BI(16-19) NEWXR ← 1
23	MISC ← 0
24	MISC ← 1
25	MISC ← CRY
26	MISC ← ACK
27	MISC ← MMU protection violation signal
28	Undefined
29	Undefined
2A	SEL ← BI(16-19) NEWXR ← 1
2B	H ← BI SEL ← BI(16-19) NEWXR ← 1
2C	H ← BI
2D	Undefined
2E	Undefined

Table B-11 General Purpose (GP) Field Decodes
(Sheet 4 of 4)

GP DECODE	OPERATION PERFORMED
2F	Undefined
30	Undefined*
31	Undefined*
32	WRAP ← $\overline{\text{CRY} \neq \text{SIGN}}$ NEWXR ← 0
33	S(1-2) ← BI(5-6)
34	S(10-15) ← BI(14-19) if F(5) = 0, INTBSY ← 0
35	LINK ← BI(11-18)
36	Undefined*
37	If AF = 9, B, D, or F, MMU validates range If AF = 8, A, C, or E, suppress MMU access rights check (NOCHEK)
38	Panel (most significant digit of display) ← BI(16-19)
39	Panel (least significant four digits of display) ← BI(4-19)
3A	Undefined*
3B	LOAD ← 1
3C	Undefined
3D	TRAFFIC ← $\overline{\text{ZERO}}$
3E	LOAD ← 0
3F	PANOK ← 1

*Used by native firmware

LS Field (Bits 1 through 3)

The LS field controls:

- The register file address when the AS field specifies RF(L) as an ALU input
- The register file address when the AD field makes a register file location available to the internal bus
- The RAM address when the DI field specifies the RAM as an internal bus source or destination
- The interpretation of AS(0) equals Zero (refer to description of AS Field).

The LS field interacts with the SM field to select specific register file and/or RAM locations. The three-bit LS field specifies a four-bit constant that is logically ANDed with the four-bit value specified by the SM field to calculate the register file and/or RAM address. Table B-12 lists the constants generated from each decode of the LS field.

Table B-12 RALU Left Select (LS) Field Decodes

LS DECODE	RESULTING CONSTANT
0	0
1	3
2	6
3	7
4	8
5	B
6	E
7	F

NA Field (Bits 53 through 63)

The NA field controls:

- The next address for firmware sequencing
- Constant generation on the internal bus
- CTR initialization.

The NA field, in conjunction with the TC and BR fields, controls the next firmware address. In general, the entire NA field specifies one of the alternative branch addresses. In

Transparent mode, if an XL or a splatter branch is being used (BR field ≠ 0 or 8), NA(0) determines the high order bit of the 11-bit address. In Sequential mode, if an LBRANCH command is being used (TC Field = 0 and BR Field = F), NA(0-2) determines the high order three bits of the address.

A constant is generated on the internal bus when BI6(0) equals Zero, or when BI6 equals 21 or 29. In all of these cases, four bits of the constant are derived from NA(3-6).

When the BS field increments the P register (BS Field = 05, 1C, or 1D) and the GP field is controlling the XB register (GP Field = 04, 05, 06, 08, 09, 0A, 14, or 15), then CTR is set to NA(1).

RS Field (Bits 5 through 7)

The RS field controls:

- The register file address when the AS field specifies RF(R) as an ALU input
- The register file address when the AD field specifies a register file location as the destination for the ALU result
- XB(0) when shifting the XB register.

The RS field interacts with the SM field to select specific register file locations. The three-bit RS field specifies a four-bit constant that is logically ANDed with the four-bit value specified by the SM field to calculate the register file address. Table B-13 lists the constants generated from each decode of the RS field.

Table B-13 RALU Right Select (RS) Field Decodes

RS DECODE	RESULTING CONSTANT
0	0
1	3
2	6
3	7
4	8
5	B
6	E
7	F

When shifting the XB register (GP Field = 04, 05, 09, 0A, 14, or 15) and AS(0) equals Zero, XB(0) receives the complement of RS(0).

SM Field (Bits 28 through 30)

The SM field interacts with the LS and RS fields to select specific register file and/or RAM addresses. The three-bit SM field specifies a four-bit value that is logically ANDed with the four-bit constant specified by the LS field to calculate addresses controlled by LS. The four-bit value specified by the SM field is also logically ANDed with the four-bit constant specified by the RS field to calculate addresses controlled by RS. The value specified by the SM field is either a constant or a function of the F/SEL register. Table B-14 lists the four-bit value resulting from each decode of the SM field.

Table B-14 Select Modify (SM) Field Decodes

SM DECODE	RESULTING 4-BIT VALUE
0	B'1111' (constant of all 1's)
1	1,F(1-3)
2	1,F(9-11)
3	B'1110'
4	Undefined
5	B'1101'
6	1,SEL(1-3)
7	SEL(0-3)

TC Field (Bits 42 through 47)

The TC field specifies the test condition to be used during firmware sequencing. Table B-15 lists the test conditons for each decode of the TC field.

Table B-15 Test Condition (TC) Field Decodes
(Sheet 1 of 3)

TC DECODE	CONDITION
00	0 (always false)
01	IFSCISTR
02	IFSIP
03	IFCIP
04	IFHALF
05	IFWORD
06	IFGTWD
07	IFQUAD
08	IFBCND
09	$\overline{\text{IFEXEC}}$
0A	IFLAF
0B	$\overline{\text{IFLOAD}}$
0C	IFMIZR
0D	IFSHIN1
0E	IFSHIN2
0F	IFPMUX
10	$\overline{\text{IFBINUM}}$; $\overline{\text{IFII}}$ when F(1-3) = 0
11	IFCRY - AS(0) = 1 IFCRY20 - AS(0) = 0
12	$\overline{\text{IFWCS}}$
13	IFBI4
14	IFSHIN
15	$\overline{\text{IF4EQ5}}$
16	IFSHZ - AS(0) = 1 IFSHZ20 - AS(0) = 0

Table B-15 Test Condition (TC) Field Decodes
(Sheet 2 of 3)

TC DECODE	CONDITION
17	IFAUZ - AS(0) = 1 IFAUZ20 - AS(0) = 0
18	$\overline{\text{IFCACHE}}$
19	IFRMWF
1A	IFLOCK
1B	IFTRACE
1C	IFIC
1D	IFACK
1E	IFPRIV
1F	IFPARER
20	$\overline{\text{IFSELEQ0}}$
21	Undefined
22	IFNUM7
23	IFXB0
24	IFF4
25	IFF5
26	IFF6
27	IFF7
28	IFF8
29	$\overline{\text{IFDSELEQ0}}$ (SEL←SEL-1)
2A	IFF9
2B	IFF11
2C	IFSEL0
2D	IFSEL1
2E	$\overline{\text{IFSL1-3EQ7}}$

Table B-15 Test Condition (TC) Field Decodes
(Sheet 3 of 3)

TC DECODE	CONDITION
2F	IFSEL3
30	IFTICK
31	IFYELLOW (YELLOW, TICK←0)
32	IFREGAD
33	IFZERO
34	IFSIGN
35	IFMISC
36	IFSEL2
37	IFADRER
38	$\overline{\text{IFRUP}}$
39	IFRPTR (SIGN←0)
3A	IFBI12
3B	IFOVFL - AS(0) = 1 IFALU0 - AS(0) = 0
3C	Undefined
3D	Undefined
3E	IFQSR (AD = 4, 5) $\overline{\text{IFDDLEQ0}}$ (AD = 0, 1)
3F	IFBI19

Appendix C

Reserved Word List and Encodings

This appendix contains an alphabetical list of reserved words, and the values they cause to be encoded into the bits of the firmware word. The AREA column indicates the area with which the reserved word is associated and may be interpreted as follows:

- BI - Internal Bus Area
- BUS - Megabus Interface Area
- CK - Master Clock Area
- FLOPS - Miscellaneous Hardware Area
- PRV - Included to maintain compatibility with a previous assembler
- PSEUDO - Pseudo op-code
- RALU - Microprocessor Area
- SEQ - Firmware Sequencing Area.

The ENCODING column indicates the fields affected (refer to Appendix B) and the corresponding hexadecimal values. When multiple values are specified, they are listed in their order of preference by the assembler.

WORD	AREA	ENCODING
AD	PRV	-
ADD	RALU	AF = 0
ADD1	RALU	AF = 8
ADD1SE	RALU	AF = 8; LS(0) = 0
ADDL	PRV	-

WORD	AREA	ENCODING
ADDR	PRV	-
ADDSE	RALU	AF = 0, 6; LS(0) = 0
ADFN	PRV	-
ADFQ	PRV	-
ADFR	PRV	-
ADLR	PRV	-
ADSL	PRV	-
ADSR	PRV	-
AF	PRV	-
AFADD	PRV	-
AFAND	PRV	-
AFINC	PRV	-
AFIOR	PRV	-
AFJ-1	PRV	-
AFJ-K	PRV	-
AFK-1	PRV	-
AFK-J	PRV	-
AFKNJ	PRV	-
AFXNR	PRV	-
AFXOR	PRV	-
ALU	BI	AD = 1, 0, 3, 4, 5, 6, 7; BS(1-2) = 0, 1, 2; [BI6(0-1) = 3; DI = 1,2, or BI6 = 26, 20; DI(1) = 0]
AMIOR	PRV	-
AMKNJ	PRV	-
AMXNR	PRV	-
AMXOR	PRV	-

WORD	AREA	ENCODING
AND	RALU	AF = C
ANDC	RALU	AF = 5 or D
AS	PRV	-
ASIL	PRV	-
ASIQ	PRV	-
ASIZ	PRV	-
ASLQ	PRV	-
ASLR	PRV	-
ASZL	PRV	-
ASZQ	PRV	-
ASZR	PRV	-
AWIL	PRV	-
AWIQ	PRV	-
AWIZ	PRV	-
AWLQ	PRV	-
AWLR	PRV	-
AWZL	PRV	-
AWZQ	PRV	-
AWZR	PRV	-
AXLP	PRV	-
AXLQ	PRV	-
AXLR	PRV	-
AXXX	PRV	-
B0 through B7	RALU, BI	Refer to Table C-1
BB	RALU, BI	Refer to Table C-1

WORD	AREA	ENCODING
BB3	RALU, BI	Refer to Table C-1
BBE	RALU, BI	Refer to Table C-1
BD	BI	DI = 4, 6; BS = 0F, 0D, 0C, 0E; [BI6 = 23 or BI6(0-1) = 3]
BDH	BI	DI = 4, 6; BS = 0F, 0D, 0C, 0E; BI6 = 25
BI	RALU	AS(1-3) = 5, 6, 7; may also affect setting of AF, depending on microprocessor function
BI	BI	-
BI6	PRV	-
BIA	PRV	-
BIB	PRV	-
BIH	PRV	-
BII	PRV	-
BIL	PRV	-
BIN	PRV	-
BIP	PRV	-
BIR	PRV	-
BIS	PRV	-
BITC	PRV	-
BITS	PRV	-
BIV	PRV	-
BIX	PRV	-
BIZ	PRV	-
BM	RALU	Refer to Table C-1
BM3	RALU	Refer to Table C-1
BME	RALU	Refer to Table C-1

WORD	AREA	ENCODING
BN	RALU	Refer to Table C-1
BN3	RALU	Refer to Table C-1
BNE	RALU	Refer to Table C-1
BP	BI	DI = 4, 6; BS = 1D, 1C; [BI6 = 23 or BI6(0-1) = 3]
BPH	BI	DI = 4, 6; BS = 1D, 1C; BI6 = 25
BR	PRV	-
BS	PRV	-
BUS	BUS	-
CALL	SEQ-OP-CODE SEQ-OPERAND	BR = C; TC = 0 BR(1-3) = 4
CHGLOCK	BUS	BS = 10, 15; C = 0
CK	PRV	-
CKHF	PRV	-
CKHL	PRV	-
CKVF	PRV	-
CKVL	PRV	-
COPY	RALU	AF = 3, B, C
CTR0	FLOPS	GP = 14, 6, 8, 15, 0A, 5, 4, 9; BS = 5, 1D, 1C; NA(1) = 0
CTR1	FLOPS	GP = 14, 6, 8, 15, 0A, 5, 4, 9; BS = 5, 1D, 1C; NA(1) = 1
D0 through D7	RALU, BI	Refer to Table C-1
DB	RALU, BI	Refer to Table C-1
DB3	RALU, BI	Refer to Table C-1
DBE	RALU, BI	Refer to Table C-1
DDLEQ0	FLOPS	AF = F, B

WORD	AREA	ENCODING
DECR	RALU	[AS(1-3) = 2, 3, 4; AF = 1], or [AS(1-3) = 7; AF = 2]
DEFAULT	PSEUDO	-
DI	PRV	-
DIA	PRV	-
DIC	PRV	-
DIN	PRV	-
DIPE	PRV	-
DIPF	PRV	-
DIR	PRV	-
DIW	PRV	-
DL	RALU	AD = 6
DM	RALU, BI	Refer to Table C-1
DM3	RALU, BI	Refer to Table C-1
DME	RALU, BI	Refer to Table C-1
DN	RALU, BI	Refer to Table C-1
DN3	RALU, BI	Refer to Table C-1
DNE	RALU, BI	Refer to Table C-1
DR	RALU	AD = 4
DRCB	PRV	-
DRCI	PRV	-
DRCL	PRV	-
DSHL	PRV	-
DSHP	PRV	-
DSHU	PRV	-
DSHY	PRV	-
DSTL	PRV	-

WORD	AREA	ENCODING
DSTU	PRV	-
DSTY	PRV	-
DWHL	PRV	-
DWHU	PRV	-
DWWL	PRV	-
DWWU	PRV	-
END	PSEUDO	-
EQU	PSEUDO	-
F	BI	GP = 20, 06, 08
FLOPS	FLOPS	-
FR8	BI	GP = 22, 03, 0B
GOTO	SEQ	TC = 0; BR = 8
GP	PRV	-
GP4	PRV	-
H	BI-SRC	DI = 4, 6; BI6 = 2A; BS(1-2) = 0, 1, 2
	BI-DEST	GP = 2C, 01, 2B
HL	CK	CK(0) = 0
HL8	BI	DI = 4, 6; BI6 = 22; BS(1-2) = 0, 1, 2
I	BI-SRC	DI = 4, 6; BI6 = 2E; BS(1-2) = 0, 1, 2
	BI-DEST	BI6 = 31
I-O	BUS	BS = 12, 17, 16; C = 0
IACK	FLOPS	BI6 = 35
IBBI4	FLOPS	BI6 = 37
IBNAZ	FLOPS	BI6 = 36; AS(0) = 1

WORD	AREA	ENCODING
IBNAZ20	FLOPS	BI6 = 36; AS(0) = 0; LS(0) = 0 - refer to Note
ICBI4	FLOPS	BI6 = 3E
ICBI19	FLOPS	BI6 = 3D
ICQSR	FLOPS	BI6 = 3C; AD = 4, 5
ICRY	FLOPS	BI6 = 3F, 3B, 37; AS(0) = 1
ICRY20	FLOPS	BI6 = 3F, 3B, 37; AS(0) = 0; LS(0) = 1 - refer to Note
IDC0 through IDCF	BI	DI = 4, 6; BI6 = 29; BS(1-2) = 0, 1, 2; NA(3-6) = y where y = 0 through F
IDS0 through IDSF	BI	DI = 4, 6; BI6 = 21; BS(1-2) = 0, 1, 2; NA(3-6) = y where y = 0 through F
IF4EQ5	SEQ	TC = 15
IFACK	SEQ	TC = 1D
IFADRER	SEQ	TC = 37
IFALU0	SEQ	TC = 3B; AS(0) = 0; LS(0) = 1 - refer to Note
IFAUZ	SEQ	TC = 17; AS(0) = 1
IFAUZ20	SEQ	TC = 17; AS(0) = 0; LS(0) = 1 - refer to Note
IFBCND	SEQ	TC = 08
IFBI4	SEQ	TC = 13
IFBI12	SEQ	TC = 3A
IFBI19	SEQ	TC = 3F
IFBINUM	SEQ	TC = 10
IFCACHE	SEQ	TC = 18
IFCIP	SEQ	TC = 03
IFCRY	SEQ	TC = 11; AS(0) = 1

WORD	AREA	ENCODING
IFCRY20	SEQ	TC = 11; AS(0) = 0; LS(0) = 1 - refer to Note
IFDDLEQ0	SEQ	TC = 3E; AD = 0, 1
IFDSELEQ0	SEQ	TC = 29
IFEXEC	SEQ	TC = 09
IFF4	SEQ	TC = 24
IFF5	SEQ	TC = 25
IFF6	SEQ	TC = 26
IFF7	SEQ	TC = 27
IFF8	SEQ	TC = 28
IFF9	SEQ	TC = 2A
IFF11	SEQ	TC = 2B
IFGTWD	SEQ	TC = 06
IFHALF	SEQ	TC = 04
IFIC	SEQ	TC = 1C
IFII	SEQ	TC = 10
IFLAF	SEQ	TC = 0A
IFLOAD	SEQ	TC = 0B
IFLOCK	SEQ	TC = 1A
IFMISC	SEQ	TC = 35
IFMIZR	SEQ	TC = 0C
IFNUM7	SEQ	TC = 22
IFOVFL	SEQ	TC = 3B; AS(0) = 1
IFPARER	SEQ	TC = 1F
IFPMUX	SEQ	TC = 0F
IFPRIV	SEQ	TC = 1E
IFQSR	SEQ	TC = 3E; AD = 4, 5

WORD	AREA	ENCODING
IFQUAD	SEQ	TC = 07
IFREGAD	SEQ	TC = 32
IFRMWF	SEQ	TC = 19
IFRPTRP	SEQ	TC = 39
IFRUP	SEQ	TC = 38
IFSCISTR	SEQ	TC = 01
IFSEL0	SEQ	TC = 2C
IFSEL1	SEQ	TC = 2D
IFSEL2	SEQ	TC = 36
IFSEL3	SEQ	TC = 2F
IFSELEQ0	SEQ	TC = 20
IFSHIN	SEQ	TC = 14
IFSHIN1	SEQ	TC = 0D
IFSHIN2	SEQ	TC = 0E
IFSHZ	SEQ	TC = 16; AS(0) = 1
IFSHZ20	SEQ	TC = 16; AS(0) = 0; LS(0) = 1 - refer to Note
IFSIGN	SEQ	TC = 34
IFSIP	SEQ	TC = 02
IFSL1-3EQ7	SEQ	TC = 2E
IFTICK	SEQ	TC = 30
IFTRACE	SEQ	TC = 1B
IFWCS	SEQ	TC = 12
IFWORD	SEQ	TC = 05
IFXB0	SEQ	TC = 23
IFYELLOW	SEQ	TC = 31
IFZERO	SEQ	TC = 33

WORD	AREA	ENCODING
IGL	FLOPS	BI6 = 38; AS(0) = 1
IGL20	FLOPS	BI6 = 39; AS(0) = 0; LS(0) = 1 - refer to Note
IGLU	FLOPS	BI6 = 3A
INCP	BUS	BS = 05
INCR	RALU	AF = 8
INCY	BUS	BS = 03, 0D, 14, 16; C = 0
IO4NE5	FLOPS	BI6 = 32
IORC	PRV	-
IOVFL	FLOPS	BI6 = 33, 3B, 37
IOWH	PRV	-
IOWU	PRV	-
IOWW	PRV	-
K	PRV	-
K--0 through K--F	BI	DI = 4, 6; BI6(0-1) = 0, 1; BI6(2-5) = y where y = 0 through F BS(1-2) = 0, 1, 2
	PRV	-
K0--	BI	DI = 4, 6; BI6(0-1) = 0; BS(1-2) = 0, 1, 2
	PRV	-
KF--	BI	DI = 4, 6; BI6(0-1) = 1; BS(1-2) = 0, 1, 2
	PRV	-
L4	BI	DI = 0; BI6 = 26
LABEL	PSEUDO	-
LBRANCH	SEQ	BR = F; TC = 0
LINK	BI	GP = 35

WORD	AREA	ENCODING
LIST	PSEUDO	-
LOAD0	FLOPS	GP = 3E
LOAD1	FLOPS	GP = 3B
LS	PRV	-
LSA0	PRV	-
LSA1	PRV	-
LSA2	PRV	-
LSA3	PRV	-
LSA4	PRV	-
LSA5	PRV	-
LSA6	PRV	-
LSA7	PRV	-
LSB0	PRV	-
LSB1	PRV	-
LSB2	PRV	-
LSB3	PRV	-
LSB4	PRV	-
LSB5	PRV	-
LSB6	PRV	-
LSB7	PRV	-
LSBB	PRV	-
LSBB3	PRV	-
LSBN	PRV	-
LSBX7	PRV	-
LSD0	PRV	-
LSD1	PRV	-

WORD	AREA	ENCODING
LSD2	PRV	-
LSD3	PRV	-
LSD4	PRV	-
LSD5	PRV	-
LSD6	PRV	-
LSD7	PRV	-
LSDB	PRV	-
LSDB6	PRV	-
LSDN	PRV	-
LSDN6	PRV	-
LSDX	PRV	-
LSDX7	PRV	-
LSM0	PRV	-
LSM1	PRV	-
LSM2	PRV	-
LSM3	PRV	-
LSM4	PRV	-
LSM5	PRV	-
LSM6	PRV	-
LSM7	PRV	-
LSMB	PRV	-
LSMB6	PRV	-
LSMN	PRV	-
LSMN6	PRV	-
LSSEL	PRV	-
LVL	BI	-

WORD	AREA	ENCODING
M1 through M7	BI	Refer to Table C-2
MB	BI	Refer to Table C-2
MB3	BI	Refer to Table C-2
MBE	BI	Refer to Table C-2
MM	BI	Refer to Table C-2
MM3	BI	Refer to Table C-2
MME	BI	Refer to Table C-2
MMU	BI	DI = 4, 6; BI6 = 24; BS = 06
MMURDACC	BUS	BS = 02; C = 0
MMUSELECT	BUS	BS = 06; C = 0
MMUWRACC	BUS	BS = 07; C = 0
MN	BI	Refer to Table C-2
MN3	BI	Refer to Table C-2
MNE	BI	Refer to Table C-2
MS0	FLOPS	GP = 23, 09, 08
MS1	FLOPS	GP = 24, 0C, 0D
MS4-9EQ0	FLOPS	GP = 0B
MSACK	FLOPS	GP = 26
MSCRY	FLOPS	GP = 25; AS(0) = 1
MSCRY20	FLOPS	GP = 25; AS(0) = 0; LS(0) = 1 - refer to Note
MSNBI19	FLOPS	GP = 02
MSPROV	FLOPS	GP = 27
NA	PRV	-
NATIVE	PSEUDO	-
NLST	PSEUDO	-

WORD	AREA	ENCODING
NO	PSEUDO	-
NOCACHE	BUS	BS = 11, 10; C = 0
NOCHEK	FLOPS	GP = 37; AF = E, 8, A, C
NONPROC	FLOPS	AF = D
NORMAL	BUS	BS = 11; C = 1
OR	RALU	AF = 3, B
P	BI-SRC	DI = 4, 6; BI6 = 24; BS = 00, 01, 04, 05, 19
	BI-DEST	BS = 09, 0E, 1A
PAGJ	PRV	-
PANEL	BI-SRC	DI = 4, 6; BI6 = 25; BS(1-2) = 0, 1
	BI-DEST	GP = 39
PANEL4	BI	GP = 38
PANOK	FLOPS	GP = 3F
PINC	PRV	-
PLOD	PRV	-
PMUX	PRV	-
PRCI	PRV	-
PRCP	PRV	-
PSELECT	BUS	BS = 00, 04; C = 0
PSHL	PRV	-
PSHY	PRV	-
PSTI	PRV	-
PSTL	PRV	-
PSTY	PRV	-
PURG	PRV	-

WORD	AREA	ENCODING
PURGE	BUS	BS = 01; C = 0
Q	RALU-SRC1,SRC2	AS(1-3) = 0, 2, 6; may also affect setting of AF, depending on microprocessor function
	RALU-DEST	AD = 0
R8	BI	DI = 5; BI6 = 20
RAM0 through RAMF	BI	Refer to Table C-2
RAMSEL	BI	Refer to Table C-2
RDREQ	BUS	-
RDREQP	BUS	BS = 19, 1A; C = 1
REGSEL	RALU	Refer to Table C-1
RETURN	SEQ-OP-CODE	BR = 2; TC = 0
	SEQ-OPERAND	BR = 2, A
RI	PRV	-
RING	BI	GP = 33
RINGCALC	FLOPS	AF = E
RINGINIT	FLOPS	AF = B
RS	PRV	-
RSB0	PRV	-
RSB1	PRV	-
RSB2	PRV	-
RSB3	PRV	-
RSB4	PRV	-
RSB5	PRV	-
RSB6	PRV	-
RSB7	PRV	-

WORD	AREA	ENCODING
RSBB	PRV	-
RSBB3	PRV	-
RSBN	PRV	-
RSD0	PRV	-
RSD1	PRV	-
RSD2	PRV	-
RSD3	PRV	-
RSD4	PRV	-
RSD5	PRV	-
RSD6	PRV	-
RSD7	PRV	-
RSDB	PRV	-
RSDB6	PRV	-
RSDN	PRV	-
RSDN6	PRV	-
RSDX	PRV	-
RSDX7	PRV	-
RSSEL	PRV	-
RUP	BI	DI = 4, 6; BS(1-2) = 0, 1, 2; [BI6 = 23 or BI6(0-1) = 3]
S	BI	DI = 4, 6; BI6 = 2F; BS(1-2) = 0, 1, 2
SEL	BI	GP = 2A, 2B, 07
SEQUENTIAL	PSEUDO	-
SET	PRV	-
SG1	FLOPS	GP = 13, 03
SGBI0	FLOPS	GP = 1A

WORD	AREA	ENCODING
SGBI4	FLOPS	GP = 19, 15, 0C, 11, 01, 02
SGBI19	FLOPS	GP = 1B, 0E, 0D, 0B
SH00	FLOPS	GP = 1C, 04, 08
SH01	FLOPS	GP = 1D, 09
SH10	FLOPS	GP = 1E
SH11	FLOPS	GP = 1F, 07
SH1IB	FLOPS	GP = 12
SH2NSG	FLOPS	GP = 10, 11
SL	RALU	AD = 7
SM	PRV	-
SM0	PRV	-
SMD	PRV	-
SME	PRV	-
SMN	PRV	-
SMR	PRV	-
SMS	PRV	-
SMX	PRV	-
SR	RALU	AD = 5
SUB	RALU	AF = 9, A
SUB1	RALU	AF = 1, 2
TC	PRV	-
TITLE	PSEUDO	-
TRAFNZ	FLOPS	GP = 3D
UNUSED	PSEUDO	-
VALID8	FLOPS	GP = 37; AF = 9, B, D, F
VL	CK	CK = 0

WORD	AREA	ENCODING
WRAP	FLOPS	GP = 32; AS(0) = 0; LS(0) = 1 - refer to Note
WRT	BUS	CK(0) = 0
WRTBYTE	BUS	CK(0) = 1
WRTWORD	BUS	CK(0) = 0
X-F	PRV	-
X0F	PRV	-
X0T	PRV	-
XA	SEQ	BR = 2, A
XAF	PRV	-
XAT	PRV	-
XB	SEQ	BR = 3, B
XB0	FLOPS	GP = 06, 08
XBF	PRV	-
XBHEX	BI	DI = 4, 6; BI6 = 2D; BS(1-2) = 0, 1, 2
XBSR	FLOPS	GP = 14, 15, 0A, 05, 04, 09; AS(0) = 1; AD = 4, 5
XBSR0	FLOPS	GP = 14, 15, 0A, 05, 04, 09; AS(0) = 0; RS(0) = 1; LS(0) = 1 - refer to Note
XBSR1	FLOPS	GP = 14, 15, 0A, 05, 04, 09 AS(0) = 0; RS(0) = 0; LS(0) = 1 - refer to Note
XBT	PRV	-
XE	SEQ	BR = 6, E
XEF	PRV	-
XET	PRV	-
XF	SEQ	BR = 7, F
XFF	PRV	-

WORD	AREA	ENCODING
XFT	PRV	-
XL	SEQ	BR = 1, 9
XL0	SEQ	BR = 1, 9; NA(0) = 0
XL1	SEQ	BR = 1, 9; NA(0) = 1
XLF	PRV	-
XLT	PRV	-
XOR	RALU	AF = 6, E
XORC	RALU	AF = 7, F
XR	SEQ	BR = 4, C
XRF	PRV	-
XRT	PRV	-
XW	SEQ	BR = 5, D
XWF	PRV	-
XWT	PRV	-
Y	BI-SRC	DI = 4, 6; BI6 = 24; [BS = 02, 03, 07, 08, 09, 0A, 0B, 1A or BS(0-1) = 2]
	BI-DEST	BS = 08, 04, 06, 0C, 1C
YBAY	PRV	-
YGJW	PRV	-
YINC	PRV	-
YLOD	PRV	-
YMUX	PRV	-
YOLD	PRV	-
YR16	BI	BS = 0B
YREL	PRV	-
YRELOC	BI	BS = 0A

WORD	AREA	ENCODING
YSELECT	BUS	BS = 02, 08, 09, 0A, 0B C = 0
Z	BI	DI = 4, 6; BI6 = 2B; BS(1-2) = 0, 1, 2
ZERO	RALU	AS(1-3) = 2, 3, 4, 7; may also affect setting of AF depending on microprocessor function
ZR0	FLOPS	GP = 16, 0A
ZR1	FLOPS	GP = 17, 05
ZRAUZ	FLOPS	GP = 18, 02, 08; AS(0) = 1
ZRAUZ20	FLOPS	GP = 18, 02, 08; AS(0) = 0; LS(0) = 1 - refer to Note
ZRQLT	FLOPS	GP = 0E

NOTE

If ADDSE or ADD1SE is the microprocessor function, then LS(0) = 0

Table C-1 Register File Operand Encodings

- When a register file location is specified as SRC1 or SRC2 in the microprocessor area:

  1. AS(1-3) equals 0, 1, 4, or 5; LS and SM are set according to Table C-3, or

  2. AS(1-3) equals 1 or 3; RS and SM are set according to Table C-3.

- When a register file location is specified as DEST in the microprocessor area, AD equals 3, 2, 4, 5, 6, or 7; RS and SM are set according to Table C-3.

- When a register file location is specified as the internal bus source, AD equals 2; LS and SM are set according to Table C-3, and

  1. DI equals 1 or 2, or

  2. BI6 equals 26 and DI equals 0 (L4), or

  3. BI6 equals 20 and DI equals 5 (R8).

Table C-2 RAM Location Operand Encodings

- LS and SM are set according to Table C-3.
- If a RAM location is specified as the internal bus source, DI equals 7, BI6(0-1) equals 3, and BS(1-2) equals 0, 1, or 2.
- If a RAM location is specified as the internal bus destination, DI equals 6 or 2.

Table C-3 LS/RS and SM Encoding Values (Sheet 1 of 2)

REGISTER FILE	RAM LOCATION	LS/RS	SM
B0	RAM8	4	-
B1	RAM9	5	5
B2	RAMA	5	3
B3	RAMB	5	0
B4	RAMC	6	5
B5	RAMD	7	5
B6	RAME	6	0 or 3
B7	RAMF	7	0
BB	-	7	6
BB3	-	5	6
BBE	-	6	6
BM	-	7	2
BM3	-	5	2
BME	-	6	2
BN	-	7	1
BN3	-	5	1
BNE	-	6	1
D0	RAM0	0	-
D1	RAM1, M1	1	5
D2	RAM2, M2	1	3
D3	RAM3, M3	1	0
D4	RAM4, M4	2	5
D5	RAM5, M5	3	5
D6	RAM6, M6	2	0 or 3
D7	RAM7, M7	3	0
DB	MB	3	6
DB3	MB3	1	6
DBE	MBE	2	6
DM	MM	3	2
DM3	MM3	1	2
DME	MME	2	2
DN	MN	3	1
DN3	MN3	1	1
DNE	MNE	2	1
REGSEL	RAMSEL	7	7

Appendix D

Summary of Restrictions

This appendix provides a list of firmware coding restrictions for each CPU area. For a number of these restrictions, violations will result in an "E29 VALUE ASSIGNMENT CONFLICT" diagnostic message. An asterisk (*) specifies that the assembler might not diagnose violations of the indicated restriction (i.e., restrictions that depend on the step just previously executed, or that include operands which imply other operands).

Microprocessor Area

1. If SRC1, SRC2, and DEST all specify register file locations, then DEST must be the same as either SRC1 or SRC2.

2. If two register file mnemonics are used among SRC1, SRC2, and DEST, then both mnemonics must be from the same group, as shown in Table D-1.

3. If SRC1 or SRC2 is a "restricted selection" register file operand (refer to Table D-2) and a FLOPS operand or firmware sequencing condition forces AUZ, CRY, and OVFL to detect on 20 bits (refer to Table D-5), the restricted selection operand must be a B register or REGSEL. This restriction does not apply if ADDSE or ADD1SE is specified.

4. If a shift modifier (SL, DL, SR, or DR) is specified, DEST must specify a register file location.

*5. If a register file operand is specified as a function of fields in the F/SEL instruction registers, and the firmware step just previously executed altered the contents of SEL, those operands that depend on SEL will use its previous contents (i.e., the contents before SEL was altered at the end of the firmware step just previously executed).

6. The following microprocessor functions have special restrictions on their operands.

FUNCTION	SRC1	SRC2	DEST
ADDSE	RF or Q only	D register only	If DEST and SRC1 both RF, DEST must = SRC1.
ADD1SE	Q only	D register only	
ANDC	Not BI	Not Q If SRC1 = ZERO, SRC2 must be BI.	If SRC1, SRC2, and DEST all RF, DEST must = SRC1.
DECR	Not ZERO	-	-
INCR	Not ZERO	-	-

Definition

RF = Register file location.

7. SRC1 and SRC2 must not both be ZERO, nor both Q, nor both BI.

Internal Bus Area

1. When a register file location is specified as the internal bus source, there are restrictions on the microprocessor area microinstruction, if specified. DEST must be a register file location from the same group, as shown in Table D-1. No shift modifier may be specified. If a "restricted selection" register file operand (refer to Table D-2) is present, the internal bus source must be the same operand.

2. A SRCMOD operand (R8 or L4) may be specified only when the source is from the microprocessor area.

3. If a RAM location is specified as an internal bus source or destination, the following restrictions apply:

   - A RAM location may not be an internal bus source and destination simultaneously.

   - If any register file operand is specified in the microprocessor microinstruction, the RAM location must come from the same group, as shown in Table D-1.

   - If a "restricted selection" register file operand (refer to Table D-2) is present in the microprocessor microinstruction or a register file location is specified as the internal bus source, the RAM location must "correspond" to the register file operand as shown in Table D-3.

4. "I" may not be an internal bus destination if any of the following is the internal bus source:

   - BDH
   - BPH
   - H
   - HL8
   - I
   - LVL
   - MMU
   - P
   - PANEL
   - S
   - XBHEX
   - Y
   - Z
   - Constant category
   - Microprocessor source with SRCMOD operand.

5. When a constant of the form xxyz#, IDCy, or IDSy is specified as the internal bus source, there are restrictions on the firmware sequencing microinstruction. The second least significant hexadecimal digit of all address value operands (bits 3 through 6 of an 11 bit value) must equal y. In the Transparent mode, if no firmware sequencing microinstruction is specified, the second least significant hexadecimal digit of *+1 must equal y.

*6. FLOPS operands in the GP or CTR categories, NOCHEK, and VALID8 may not be specified if any operand from the other internal bus destinations category is used, except as noted in Table D-4 and under Miscellaneous Hardware Area, Item 5.

*7. If BD, BP, or RUP is the internal bus source, and I is an internal bus destination or a FLOPS operand from the indicator register (I) category is specified, bits 0 through 3 of the internal bus are undefined.

*8. If a register file or RAM location is specified as a function of fields in the F/SEL instruction registers and the firmware step just previously executed altered the contents of SEL, those operands that depend on SEL will use its previous contents (i.e., the contents before SEL was altered at the end of the firmware step just previously executed).

*9. Use of BP or BPH as an internal bus source automatically causes P to be incremented.

Megabus Interface Area

1. A Megabus interface area microinstruction may not be specified if BD, BDH, BP, BPH, MMU, P, RUP, or Y is the internal bus source or if P, Y, YR16, or YRELOC is an internal bus destination, except as follows:

MICROINSTRUCTION	PERMISSIBLE BI SOURCES	PERMISSIBLE BI DESTINATIONS
BUS INCP	P	
BUS INCY	BD, BDH, Y	
BUS MMURDACC	Y	
BUS MMUSELECT	MMU	*Implies Y as BI destination
BUS MMUWRACC	Y	
BUS PSELECT	P	Y
BUS PURGE	P	P, Y, YR16, YRELOC
BUS YSELECT	Y	
RDREQ NORMAL	Y	
RDREQ CHGLOCK	Y	
RDREQ I-O	Y	
RDREQ NOCACHE	Y	
RDREQP	P, Y	P (requires Y as source)
WRT CHGLOCK	Y	
WRT I-O	Y	
WRT I-O, INCY	Y	
WRT INCY	Y	
WRTBYTE CHGLOCK	Y	
WRTBYTE I-O	Y	
WRTBYTE I-O, INCY	Y	
WRTBYTE INCY	Y	
WRTWORD CHGLOCK	Y	
WRTWORD I-O	Y	
WRTWORD I-O, INCY	Y	
WRTWORD INCY	Y	

2. An internal bus source must be specified in any step in which a WRT, WRTBYTE, or WRTWORD is specified.

3. The ALU result may not be the internal bus source in the same step in which a WRT, WRTBYTE, or WRTWORD is specified.

*4. BUS PURGE must be specified in some step prior to one which includes both RDREQP and BI, Y, P.

Miscellaneous Hardware Area

1. If a FLOPS operand from the indicator register (I) category is used, the following internal bus sources may not be specified.

   - BDH
   - BPH

- H
- HL8
- I
- LVL
- MMU
- P
- PANEL
- S
- XBHEX
- Y
- Z
- Constant Category
- Microprocessor source with SRCMOD operand.

*2. At most one FLOPS operand from the following group may be specified in a single step: CTR0, CTR1, NOCHEK, VALID8, or GP category (except as noted in Table D-4 and in Item 5 below).

*3. FLOPS operands from the GP category and internal bus destinations from the "other destinations" category may only be combined as shown in Table D-4.

4. FLOPS operands or firmware sequencing conditions which force microprocessor signals, AUZ, CRY, and OVFL to detect on 16 bits may not be specified simultaneously with FLOPS operands or firmware sequencing conditions which force AUZ, CRY, and OVFL to detect on 20 bits (refer to Table D-5).

*5. CTR0 and CTR1 imply XBSR unless XB0, XBSR0, or XBSR1 is specified. P is incremented when these operands are used. CTR0 restricts bit 1 (assuming 11-bit values) of firmware sequencing address value operands to 0 and CTR1 restricts bit 1 to 1.

*6. ICQSR or XBSR may be specified only when a right shift (SR or DR) is specified in the microprocessor micro-instruction.

7. Use of XBSR0 requires any register file microprocessor operand that is not a "restricted selection" (refer to Table D-1) to be a B register. Use of XBSR1 requires any register file operand that is not a restricted selection to be a D register.

*8. Operands in the MMU category imply other firmware functions as follows:

OPERAND	IMPLIED FUNCTION	OTHER FUNCTIONS WHICH OVERRIDE IMPLIED FUNCTION
DDLEQ0	XORC	
NOCHEK	RINGCALC	INCR, ADD1, AND, SUB
NONPROC	ANDC	
RINGCALC	XOR	
RINGINIT	OR	COPY
VALID8	SUB	DDLEQ0, NONPROC, RINGINIT

Firmware Sequencing Area

1. When a constant of the form xxyz#, IDCy, or IDSy is specified as the internal bus source, there are restrictions on the firmware sequencing microinstruction. The second least significant hexadecimal digit of all address value operands (bits 3 through 6 of an 11-bit value) must equal y. In the Transparent mode, if no firmware sequencing microinstruction is specified, the second least significant hexadecimal digit of *+1 must equal y.

2. FLOPS operand CTR0 restricts bit 1 (assuming 11-bit values) of address value operands to 0, and CTR1 restricts bit 1 to 1.

3. If both operands of a condition specify address values (legal in Transparent mode only). one operand must equal the other operand ORed with 3.

4. Firmware sequencing conditions or FLOPS operands which force microprocessor signals AUZ, CRY, and OVFL to detect on 16 bits may not be specified simultaneously with conditions or FLOPS operands which force AUZ, CRY, and OVFL to detect on 20 bits (refer to Table D-5).

5. When XL0 is used as an operand of a condition, bit 0 (assuming 11-bit values) of the other operand must be 0. When XL1 is used, bit 0 must be 1.

*6. IFQSR may be used only when a right shift (SR or DR) is specified in the microprocessor microinstruction.

7. IFDDLEQ0 may be used only when DEST in the microprocessor area microinstruction is Q or null.

*8. XA, XB, XE, XF, XR, or XW may not be specified simultaneously with F or FR8 as an internal bus destination.

9. At most, one of the following group may be an operand of a Transparent mode condition: XA, XB, XE, XF, XL, XL0, XL1, XR, or XW.

Table D-1 Register File/RAM Locations Legal Groups

- D0, D3, D6, D7, B0, B3, B6, B7, M3, M6, M7, RAM0, RAM3, RAM6, RAM7, RAM8, RAMB, RAME, RAMF.
- D0, D1, D4, D5, B0, B1, B4, B5, M1, M4, M5, RAM0, RAM1, RAM4, RAM5, RAM8, RAM9, RAMC, RAMD.
- D0, D2, D6, B0, B2, B6, M2, M6, RAM0, RAM2, RAM6, RAM8, RAMA, RAME.
- D0, DN, DN3, DNE, B0, BN, BN3, BNE, MN, MN3, MNE, RAM0, RAM8.
- D0, DB, DB3, DBE, B0, BB, BB3, BBE, MB, MB3, MBE, RAM0, RAM8.
- D0, DB, DB3, DBE, REGSEL, MB, MB3, MBE, RAM0, RAMSEL.
- D0, DM, DM3, DME, B0, BM, BM3, BME, MM, MM3, MME, RAM0, RAM8.

Table D-2 Register File "Restricted Selection" Criteria

A Register file operand which is microprocessor SRC1 or SRC2 or internal bus SRC is a "restricted selection" operand if:

- It is the internal bus source
- It is SRC2 for the functions ANDC, ADDSE, or ADD1SE
- The microprocessor DEST is a different register file operand
- The other microprocessor source is Q
- The other microprocessor source is BI
- The other microprocessor source is another register file operand which does not satisfy the restriction(s) (i.e., if SRC1 and SRC2 are both register file operands, one must be a "restricted selection" operand).

Table D-3 Register File/RAM Location Operand Correspondence

REGISTER FILE OPERAND	CORRESPONDING RAM LOCATION OPERAND
D0	RAM0
D1-D7	RAM1-RAM7 or M1-M7
B0-B7	RAM8-RAMF
REGSEL	RAMSEL
DB	MB
DB3	MB3
DBE	MBE
DM	MM
DM3	MM3
DME	MME
DN	MN
DN3	MN3
DNE	MNE

Table D-4 Permissible GP Combinations (Sheet 1 of 2)

DESTINATION(S) OF INTERNAL BUS	GP CATEGORY OPERANDS
PANEL	None
PANEL4	None
LVL	None
RING	None
LINK	None
H,SEL	None
H	SGBI4
SEL	SH11
FR8	SG1
FR8	SGBI19, MS4-9EQ0
F	XB0
F	XB0, MS0, SH00, ZRAUZ
F	XB0, MS0, SH00, ZRAUZ20
None of the above	SGB14, MSNBI19, ZRAUZ
None of the above	SGBI4, MSNBI19, ZRAUZ20

Table D-4 Permissible GP Combinations (Sheet 2 of 2)

DESTINATION(S) OF INTERNAL BUS	GP CATEGORY OPERANDS
None of the above	SGBI4, SH2NSG
None of the above	SGBI4, MS1
None of the above	SGBI19, MS1
None of the above	SGBI19, ZRQLT
None of the above	SGBI4, XBSR
None of the above	SGBI4, XBSR0
None of the above	SGBI4, XBSR1
None of the above	ZR0, XBSR
None of the above	ZR0, XBSR0
None of the above	ZR0, XBSR1
None of the above	ZR1, XBSR
None of the above	ZR1, XBSR0
None of the above	ZR1, XBSR1
None of the above	SH00, XBSR
None of the above	SH00, XBSR0
None of the above	SH00, XBSR1
None of the above	SH10, XBSR
None of the above	SH10, XBSR0
None of the above	SH10, XBSR1
None of the above	Any single GP-category operand, except ZRQLT, XB0, MSNBI19, or MS4-9EQ0

Table D-5 Operands Affecting AUZ, CRY, and OVFL

OPERAND TYPE	FORCE 16-BIT DETECT	FORCE 20-BIT DETECT
FLOPS Operands	IBNAZ	IBNAZ20
	ICRY	ICRY20
		IGL20
	MSCRY	MSCRY20
		WRAP
	XBSR	XBSR0
		XBSR1
	ZRAUZ	ZRAUZ20
Conditions	IFAUZ	IFAUZ20
	IFCRY	IFCRY20
	IFOVFL	IFALU0
	IFSHZ	IFSHZ20

Appendix E

Instruction Register Maps

The maps supplied in this appendix represent the decoding patterns available to the microprogrammer via the firmware sequencing conditions and reserved operands. The patterns were selected to facilitate interpretation of the native instruction set; their utility for user firmware will depend on the similarity of structure between the user-defined instructions and those of the native set.

The first group of maps (see Tables E-1 through E-5) define the "splatter" branches available in the Transparent mode. Each splatter, when it is used, generates a specific 10-bit address as a function of the instruction register content (and a few other bits of context). The most significant bit and the least significant bit of each such address are always zero. The 10-bit address is relative to the 1K bank of the firmwware which contains the alternate next address. Thus, for example, if register F contained 888#, the code:

```
IFF5  ABC#,XW
```

would transfer control to location 964#.

In the native firmware, the five splatter branches function as follows:

XA - decode the Address syllable of those instructions that use one, and the op-code of others. Column XA of Table E-1 represents either the generated 10-bit address, or a note directing the reader to a subsequent table for further information. XA normally is used with SIGN = 0.

XB - decode the address syllable of data descriptors in commercial (Business) instructions. Column XB of Table E-1 represents either the generated 10-bit address or a note directing the reader to a subsequent table for further information. XB normally is used with SIGN = 1.

XE - decode the op-code of most single-operand and double-operand instructions for Execution. Column XE of Table E-1 represents the generated 10-bit address (modifiers L and M are defined in the notes.

XR - classify the op-code/address syllable as to operand type, and Read the operand, after completing any indirect addressing and indexing actions necessary. Table E-2 defines, as a function of address syllable type, which column of Table E-1 represents the generated 10-bit address (the control flop NEWXR helps distinguish between reentrant XR references).

XW - classify the op-code/address-syllable as to operand type, and Write the result accordingly. Table E-3 defines, as a function of address syllable type, which column of Table E-1 represents the generated 10 bit address.

Table E-6 defines four testable "conditions" which (in the native firmware) categorize instructions as to data field size: bit, halfword, word, double-word, or quadruple-word (the only native quadruple-word operands are scientific data, which depend on the values of bits 2, 4, and 6 of register M4). Also shown in the last column of Table E-6 is the set of instructions for which the "memory lock" (CHGLOCK) functionality is invoked.

Table E-1 Main Splatter Map (Sheet 1 of 2)

Table E-1 Main Splatter Map

F REGISTER BITS											
0-3	4-7	8-11	XA	XB	XE	XR1	XR2	XR3	XR4	XW1	XW2
0	0	0-F	note AG	note BG	018	1A8	1AE	1A0	1A4	160	160
0	1	0-F	1E0	1C0	018	1A8	1AE	1A0	1A4	160	160
0	2-B	0-F	note AI0	note BI0	018	1A8	1AE	1A0	1A4	160	160
0	C-E	0-F	1E0	1C0	018	1A8	1AE	1A0	1A4	160	160
0	F	0-F	note AI0	note BI0	018	1A8	1AE	1A0	1A4	160	160
1-7	0	0-F	note AH	note BH	018	1A8	1AE	1A0	1A4	160	160
1-7	1-2	0-F	1E0	1C0	018	1A8	1AE	1A0	1A4	160	160
1-7	3-6	0-F	note AI3	note BI1	018	1A8	1AE	1A0	1A4	160	160
1-7	7	0-F	note AI1	note BI1	018	1A8	1AE	1A0	1A4	160	160
1-7	8-B	0-F	note AI2	note BI1	018	1A8	1AE	1A0	1A4	160	160
1-7	C	0-7	180	1C0	03C	1A8	1A2	1A0	1A4	160	160
1-7	C	8-F	1A0	1C0	03C	1A8	1A2	1A0	1A4	160	160
1-7	D	0-7	180	1C0	11C	1A8	1A2	1A0	1A4	160	160
1-7	D	8-F	1A0	1C0	11C	1A8	1A2	1A0	1A4	160	160
1-7	E	0-7	180	1C0	0BA	1A8	1A2	1A0	1A4	164	16C
1-7	E	8-F	1A0	1C0	0BA	1A8	1A2	1A0	1A4	164	16C
1-7	F	0-7	180	1C0	0FA	1A8	1A2	1A0	1A4	164	16C
1-7	F	8-F	1A0	1C0	0FA	1A8	1A2	1A0	1A4	164	16C
8	0	0-7	note AS1	note BS1	194	1A8	1B4	1A0	1A4	164	16C
8	0	8-F	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
8	1	0-7	note AS1	note BS1	194	1A8+2M	1B4	1A0	1A4	164+2M	16C+2M
8	1	8-F	note AS0	note BS0	194	1A8+2M	1B4	1A0	1A4	164+2M	16C+2M
8	2	0-7	note AS1	note BS1	19A	1A8	1A2	1A0	1A4	164	16C
8	2	8-F	note AS0	note BS0	13C	1AA	1A2	1A0	1A4	164	16C
8	3	0-7	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
8	3	8-F	note AS3	note BS3	058	1A8	1B0	1A0	1A4	160	160
8	4	0-7	note AS0	note BS0	156	1A6	1A2	1A0	1A4	168	16A
8	4	8-F	note AS0	note BS0	196	1A6	1A2	1A0	1A4	168	16A
8	5	0-7	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
8	5	8-F	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
8	6	0-7	note AS1	note BS1	05A	1A8	1A2	1A0	1A4	164	16C
8	6	8-F	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
8	7	0-7	note AS1	note BS1	0BE	1A8	1B6	1A0	1A4	164	16C
8	7	8-F	note AS1	note BS1	0BE	1AA	1B6	1A0	1A4	166	16E
8	8	0-7	note AS1	note BS1	19C	1AA	1B8	1A0	1A4	164	16C
8	8	8-F	note AS1	note BS1	17A	1A8	1B8	1A0	1A4	164	16C
8	9	0-F	note AS1	note BS1	17C	1AA	1B8	1A0	1A4	164	16C
8	9	8-F	note AS0	note BS0	0D8	1A8	1A2	1A0	1A4	164	16C
8	A	0-7	note AS1	note BS1	15C	1AA	1B8	1A0	1A4	164	16C
8	A	8-F	note AS1	note BS1	15A	1A8	1B8	1A0	1A4	164	16C
8	B	0-7	note AS1	note BS1	1BC	1AA	1B8	1A0	1A4	164	16C
8	B	8-F	note AS3	note BS3	058	1A8	1B0	1A0	1A4	160	160
8	C	0-7	note AS1	note BS1	0FE	1A8	1B6	1A0	1A4	164	16C
8	C	8-F	note AS0	note BS0	176	1A6	1A2	1A0	1A4	168	16A
8	D	0-7	note AS1	note BS1	17E	1A6	1B6	1A0	1A4	164	16C
8	D	8-F	note AS2	note BS2	0D6	1A8-2L	1A2	1A0	1A4	164+4L	16A
8	E	0-7	note AS0	note BS0	138	1A8	1A2	1A0	1A4	164	16C
8	E	8-F	note AS1	note BS1	03A	1A8	1A2	1A0	1A4	164	16C
8	F	0-7	note AS0	note BS0	05E	1A8	1B6	1A0	1A4	164	16C
8	F	8-F	note AS0	note BS0	05E	1A8	1B6	1A0	1A4	164	16C
9-F	0	0-7	note AS0	note BS0	118	1A8	1A2	1A0	1A4	164	16C
9-F	0	8-F	note AS0	note BS0	03C	1AA	1A2	1A0	1A4	164	16C
9-F	1	0-7	note AS0	note BS0	038	1A8	1A2	1A0	1A4	164	16C
9-F	1	8-F	note AS0	note BS0	11C	1AA	1A2	1A0	1A4	164	16C
9-F	2	0-7	note AS0	note BS0	0DA	1A8	1A2	1A0	1A4	164	16C
9-F	2	8-F	note AS0	note BS0	05C	1AA	1A2	1A0	1A4	164	16C
9-F	3	0-7	note AS0	note BS0	11A	1A8	1A2	1A0	1A4	164	16C
9-F	3	8-F	note AS3	note BS3	058	1A8	1AE	1A0	1A4	160	160
9-F	4	0-7	note AS0	note BS0	0BC	1A8	1A2	1A0	1A4	160	160
9-F	4	8-F	note AS0	note BS0	0BC	1AA	1A2	1A0	1A4	164	16C
9-F	5	0-7	note AS0	note BS0	0DC	1A8	1A2	1A0	1A4	160	160
9-F	5	8-F	note AS0	note BS0	0DC	1AA	1A2	1A0	1A4	164	16C
9-F	6	0-7	note AS0	note BS0	0EC	1A8	1A2	1A0	1A4	160	160
9-F	6	8-F	note AS0	note BS0	0EC	1AA	1A2	1A0	1A4	164	16C
9-F	7	0-7	note AS1	note BS1	11E	1A8	1B6	1A0	1A4	164	16C
9-F	7	8-F	note AS1	note BS1	0DE	1AA	1B6	1A0	1A4	166	16E
9-F	8	0-7	note AS0	note BS0	03C	1A8	1A2	1A0	1A4	160	160
9-F	8	8-F	note AS0	note BS0	15E	1A6	1B6	1A0	1A4	164	16C
9-F	9	0-7	note AS0	note BS0	11C	1A8	1A2	1A0	1A4	160	160
9-F	9	8-F	note AS0	note BS0	15E	1A6	1B6	1A0	1A4	164	16C
9-F	A	0-7	note AS0	note BS0	0BA	1A8	1A2	1A0	1A4	164	16C
9-F	A	8-F	note AS1	note BS1	0F8	1A8	1A2	1A0	1A4	164	16C
9-F	B	0-7	note AS0	note BS0	0FA	1A8	1A2	1A0	1A4	164	16C
9-F	B	8-F	note AS3	note BS3	058	1A8	1B2	1A0	1A4	160	160
9-F	C	0-7	note AS0	note BS0	15E	1A6	1B6	1A0	1A4	164	16C
9-F	C	8-F	note AS2	note BS2	036	1A8-2L	1A2	1A0	1A4	164+4L	16A
9-F	D	0-7	note AS0	note BS0	1BE	1A6	1B6	1A0	1A4	164	16C
9-F	D	8-F	note AS2	note BS2	116	1A8-2L	1A2	1A0	1A4	164+4L	16A
9-F	E	0-7	note AS1	note BS1	018	1A8	1A2	1A0	1A4	164	16C
9-F	E	8-F	note AS3	note BS3	016	1A8-2L	1A2	1A0	1A4	164+4L	16A
9-F	F	0-7	note AS1	note BS1	0DE	1AE	1B6	1A0	1A4	164	16C
9-F	F	8-F	note AS3	note BS3	136	1A8-2L	1B6	1A0	1A4	164+4L	16A

Table E-1 Main Splatter Map (Sheet 2 of 2)

where:

L = 1 if LAF; otherwise, L = 0

M = 1 if MISC V ZERO; otherwise, M = 0

NOTES

AG - refer to Table E-4, variation XAG
AH - refer to Table E-4, variation XAH
AI0 - refer to Table E-4, variation XAI, column 0
AI1 - refer to Table E-4, variation XAI, column 1
AI2 - refer to Table E-4, variation XAI. column 2
AI3 - refer to Table E-4, variation XAI, column 3
AS0 - refer to Table E-4, variation XAS, R=0, Z=0
AS1 - refer to Table E-4, variation XAS, R=0, Z=1
AS2 - refer to Table E-4, variation XAS, R=1, Z=0
AS3 - refer to Table E-4, variation XAS, R=1, Z=1
BG - refer to Table E-5, variation XBG
BH - refer to Table E-5, variation XBH
BI0 - refer to Table E-5, variation XBI, column 0
BI1 - refer to Table E-5, variation XBI, column 1
BS0 - refer to Table E-5, variation XBS, R=0, Z=0
BS1 - refer to Table E-5, variation XBS, R=0, Z=1
BS2 - refer to Table E-5, variation XBS, R=1, Z=0
BS3 - refer to Table E-5, variation XBS, R=1, Z=1

Table E-2 Key to XR

SEL \ F(9-11)	0	1-3	4	5	6	7
0	XR2	XR1	XR2	XR2	XR2	XR2
1-7	XR2	XR1	XR2	XR3	XR2	XR2
8	XR4	XR4	XR4	XR2	XR2	XR2
9-B	XR4	XR4	XR4	XR1	XR1	XR1
C	XR4	XR4	XR4	XR4	XR4	XR4
D-F	XR4	XR4	XR4	XR1	XR1	XR1

NOTE

If XRNEW = 0, use XR2 instead of XR1.

Table E-3 Key to XW

SEL \ F(9-11)	0-4	5	6-7
0	XW1	XW1	XW1
1-7	XW1	XW2	XW1
8	XW1	XW1	XW1
9-F	XW1	XW2	XW1

Table E-4 XA Variations (Sheet 1 of 2)

SEL \ F(8-11)	0	1	2	3	4-5	6	7	8-F
0	1E2	1FC	1F2	1F0	1E0	1FE	1E0	1C0
1	1E4	1FE	1F0	1F0	1E0	1FE	1E0	1C0
2	1E6	1EA	1EC	1F0	1E0	1FE	1E0	1C0
3	1E8	1EA	1F2	1F0	1E0	1FE	1E0	1C0
4	1EA	1E0	1F2	1F0	1E0	1E0	1E0	1C0
5	1EA	1E0	1F0	1F0	1E0	1E0	1E0	1C0
6	1EA	1E0	1F2	1EC	1E0	1E0	1E0	1C0
7	1EA	1E0	1F0	1EC	1E0	1E0	1E0	1C0
8	1EE	1E0	1F2	1EC	1E0	1E0	1E0	1C0
9	1E0	1E0	1F0	1F4	1E0	1E0	1E0	1C0
A	1F8	1E0	1F0	1F2	1E0	1E0	1E0	1C0
B	1FA	1E0	1F4	1F2	1E0	1E0	1E0	1C0
C	1FC	1E0	1F0	1F2	1E0	1E0	1E0	1C0
D	1FE	1E0	1F0	1F2	1E0	1E0	1E0	1C0
E	1E0	1E0	1F6	1F6	1E0	1E0	1E0	1C0
F	1E0	1E0	1EC	1F6	1E0	1E0	1E0	1C0

VARIATION XAG (SIGN = 0)

SEL \ F(8-11)	0	1-3	4	5-6	7	8	9-B	C	D-E	F
0	1E8	1E8	1E8	1E8	1E8	1C8	1C8	1C8	1C8	1C8
1-7	1F8	1FA	1FA	1FA	1FA	1D8	1DA	1DA	1DA	1DA
8	1E2	1E4	1E2	1E8	1EC	1C2	1C4	1C2	1C8	1CC
9-F	1F8	1FA	1FA	1FA	1FA	1D8	1DA	1DA	1DA	1DA

VARIATION XAG (SIGN = 1)

F(8-11)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
XAH	0A0	0A2	0A4	0A6	0A8	0AA	0AC	0AE	0C0	0C2	0C4	0C6	0C8	0CA	0CC	0CE

VARIATION XAH

F(9-11),SEL \ Column	0	1	2	3
00	0E4	120	140	160
01	0E2	120	140	160
02-3F	0E0	122	142	162
40-7F	0E6	122	142	162

VARIATION XAI

Table E-4 XA Variations (Sheet 2 of 2)

SEL \ F(9-11)	0	1-3	4	5	6	7
0	090	092	084	09E	09E	094+2Z
1-7	080	082	086	098+2R	088	08A
8	090	092	084	09E	09E	084
9-B	080	082	086	08C	08C	08C
C	080	082	086	09E	09E	09E
D-F	080	082	086	08E	08E	08E

VARIATION XAS

Table E-5 XB Variations (Sheet 1 of 2)

SEL \ F(8-11)	0	1	2	3	4-5	6	7	8-F
0	1C2	1DC	1D2	1D0	1C0	1DE	1C0	1C0
1	1C4	1DE	1D0	1D0	1C0	1DE	1C0	1C0
2	1C6	1CA	1CC	1D0	1C0	1DE	1C0	1C0
3	1C8	1CA	1D2	1D0	1C0	1DE	1C0	1C0
4	1CA	1C0	1D2	1D0	1C0	1C0	1C0	1C0
5	1CA	1C0	1D0	1D0	1C0	1C0	1C0	1C0
6	1CA	1C0	1D2	1CC	1C0	1C0	1C0	1C0
7	1CA	1C0	1D0	1CC	1C0	1C0	1C0	1C0
8	1CE	1C0	1D2	1CC	1C0	1C0	1C0	1C0
9	1C0	1C0	1D0	1D4	1C0	1C0	1C0	1C0
A	1D8	1C0	1D0	1D2	1C0	1C0	1C0	1C0
B	1DA	1C0	1D4	1D2	1C0	1C0	1C0	1C0
C	1DC	1C0	1D0	1D2	1C0	1C0	1C0	1C0
D	1DE	1C0	1D0	1D2	1C0	1C0	1C0	1C0
E	1C0	1C0	1D6	1D6	1C0	1C0	1C0	1C0
F	1C0	1C0	1CC	1D6	1C0	1C0	1C0	1C0

VARIATION XBG (SIGN = 0)

SEL \ F(8-11)	0	1-3	4	5-6	7	8	9-B	C	D-E	F
0	1C8	1C8	1C8	1C8	1C8	1C8	1C8	1C8	1C8	1C8
1-7	1D8	1DA	1DA	1DA	1DA	1D8	1DA	1DA	1DA	1DA
8	1C2	1C4	1C2	1C8	1CC	1C2	1C4	1C2	1C8	1CC
9-F	1D8	1DA	1DA	1DA	1DA	1D8	1DA	1DA	1DA	1DA

VARIATION XBG (SIGN = 1)

Table E-5 XB Variations (Sheet 2 of 2)

F(8-11)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
XBH	1C0	1C2	1C4	1C6	1C8	1CA	1CC	1CE	1C0	1C2	1C4	1C6	1C8	1CA	1CC	1CE

VARIATION XBH

F(9-11),SEL \ Column	0	1
00	1C4	1C0
01	1C2	1C0
02-3F	1C0	1C2
40-7F	1C6	1C2

VARIATION XBI

SEL \ F(9-11)	0	1-3	4	5	6	7
0	1D0	1D2	1C4	1DE	1DE	1D4+2Z
1-7	1C0	1C2	1C6	1D8+2R	1C8	1CA
8	1D0	1D2	1C4	1DE	1DE	1C4
9-B	1C0	1C2	1C6	1CC	1CC	1CC
C	1C0	1C2	1C6	1DE	1DE	1DE
D-F	1C0	1C2	1C6	1CE	1CE	1CE

VARIATION XBS

Table E-6 Test Conditions

F(0-3)	F(4-7)	F(8-11)	IFHALF	IFWORD	IFGTWD	IFQUAD	ENABLE CHGLOCK
0	0	0-1	0	1	1-M	0	0
0	0	2-3	0	1	1-M	0	1
0	0	4-5	0	1	1-M	0	0
0	0	6-7	0	1	1-M	0	1
0	1-E	0-F	0	1	1-M	0	0
0	F	0-7	0	1	0	0	0
0	F	8-F	0	1	1-M	0	0
1-7	0	0-F	0	0	0	0	0
1-7	1-2	0-F	0	1	1-M	0	0
1-7	3-6	0-F	0	0	0	0	0
1-7	7-B	0-F	0	1	1-M	0	0
1-7	C-F	0-F	0	0	0	0	0
8	0	0-F	0	1	0	0	0
8	1	0-F	1-M	M	0	0	0
8	2	0-7	0	1	0	0	0
8	2	8-F	0	0	0	0	0
8	3	0-F	0	1	0	0	0
8	4	0-F	0	0	1	0	0
8	5	0-F	0	1	0	0	0
8	6	0-F	0	1	0	0	0
8	7	0-7	0	1	0	0	0
8	7	8-F	1	0	0	0	0
8	8	0-7	0	0	0	0	1
8	8	8-F	0	1	0	0	1
8	9	0-7	0	0	0	0	1
8	9	8-F	0	1	0	0	0
8	A	0-7	0	0	0	0	1
8	A	8-F	0	1	0	0	1
8	B	0-7	0	0	0	0	1
8	B	8-F	0	1	0	0	0
8	C	0-7	0	1	0	0	0
8	C	8-F	0	0	1	0	0
8	D	0-7	0	0	1	0	0
8	D	8-F	0	1-L	L	0	0
8	E-F	0-F	0	1	0	0	0
9-F	0-2	0-7	0	1	0	0	0
9-F	0-2	8-F	1	0	0	0	0
9-F	3	0-F	0	1	0	0	0
9-F	4-7	0-7	0	1	0	0	0
9-F	4-7	8-F	1	0	0	0	0
9-F	8-9	0-7	0	1	0	0	0
9-F	8-9	8-F	0	0	1	S	0
9-F	A-B	0-F	0	1	0	0	0
9-F	C-D	0-7	0	0	1	S	0
9-F	C-D	8-F	0	1-L	L	0	0
9-F	E-F	0-7	0	1	0	0	0
9-F	E-F	8-F	0	1-L	L	0	0

where:

L = 1 if LAF; otherwise, L = 0

M = 1 if MISC V ZERO; otherwise, M = 0

S = selected bit of scientific mode register (M4) or F:

F(0-3) =	9	A	B	C	D	E	F
S =	M4(2)	M4(4)	M4(6)	F(5)	M4(2)	M4(4)	M4(6)

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

M&TO HARDWARE PUBLICATIONS
USER COMMENTS FORM

DOCUMENT TITLE: _____

PART NO.: _____

ORDER NO.: _____

ERRORS:

HOW DO YOU USE THIS DOCUMENT?

THEORY _____ ☐

MAINTENANCE _____ ☐

TROUBLESHOOTING _____ ☐

OTHER: _____

_____

DOES THIS MANUAL SATISFY YOUR REQUIREMENTS?

YES ☐ NO ☐

IF NOT, PLEASE EXPLAIN _____

_____

_____

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

FIRST CLASS
Permit No. 39531
Waltham, Ma.

Business Reply Mail NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

HONEYWELL INFORMATION SYSTEMS INC.
200 SMITH STREET
WALTHAM, MA. 02154

MAIL STATION 872A
HARDWARE PUBLICATIONS, BILLERICA

Honeywell

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

M&TO HARDWARE PUBLICATIONS
USER COMMENTS FORM

DOCUMENT TITLE: _____

PART NO.: _____

ORDER NO.: _____

ERRORS:

HOW DO YOU USE THIS DOCUMENT?

THEORY ☐

MAINTENANCE ☐

TROUBLESHOOTING ☐

OTHER: _____

_____

DOES THIS MANUAL SATISFY YOUR REQUIREMENTS?

YES ☐ NO ☐

IF NOT, PLEASE EXPLAIN _____

_____

_____

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

FIRST CLASS
Permit No. 39531
Waltham, Ma.

Business Reply Mail NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

HONEYWELL INFORMATION SYSTEMS INC.
200 SMITH STREET
WALTHAM, MA. 02154

MAIL STATION 872A
HARDWARE PUBLICATIONS, BILLERICA

Honeywell

Honeywell

Honeywell Information Systems
In the U S A 200 Smith Street, MS 486 Waltham, Massachusetts 02154
In Canada 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico Avenida Nuevo Leon 250 Mexico 11, D F