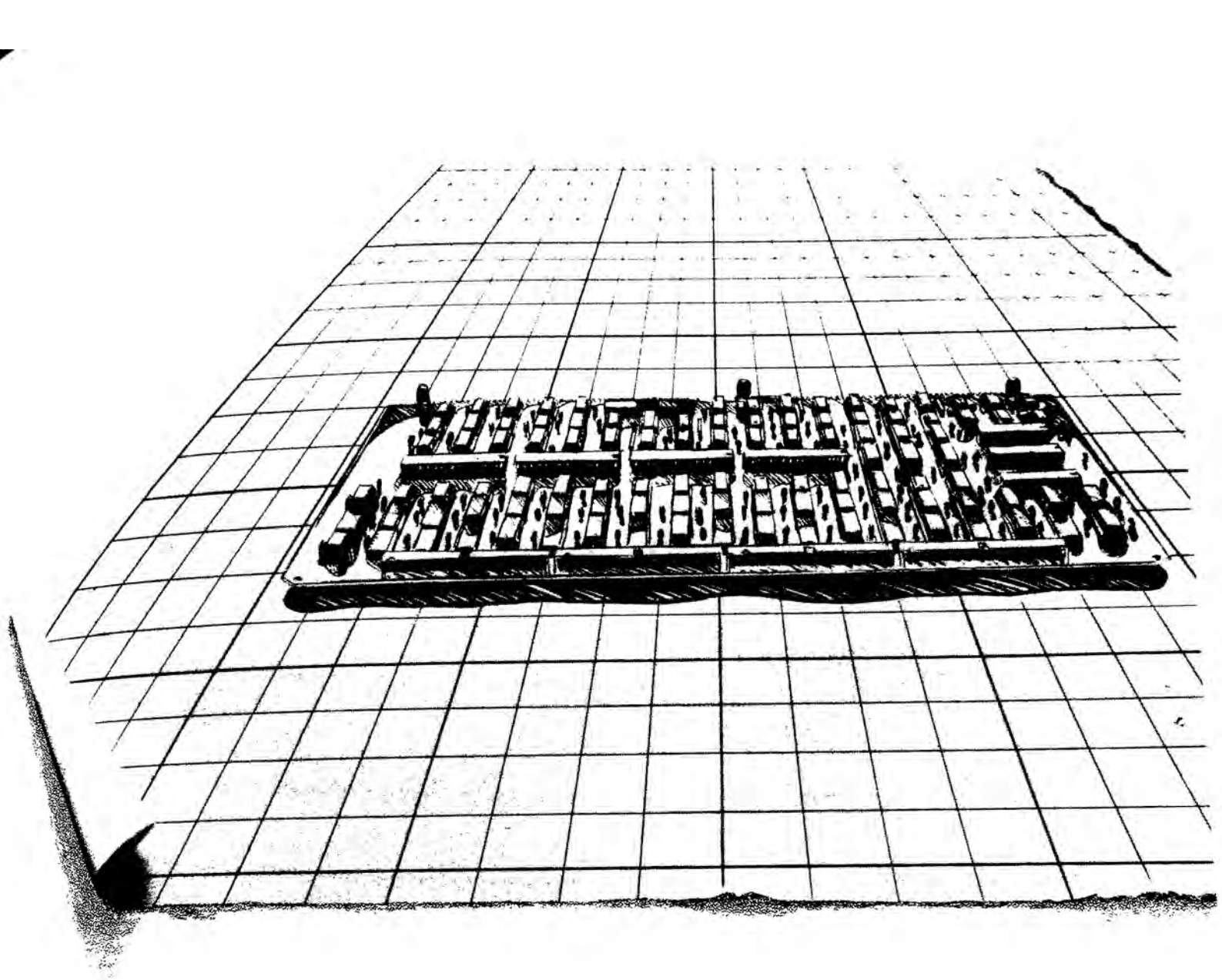


Chapter VIII
HEX-29

INTRODUCTION

Modern digital systems are becoming faster and increasingly complex. As a result, more is being demanded of digital design engineers. Fortunately, there is a design technique that can greatly simplify the design process. It can also lead to cleaner, more efficient, more reliable finished devices. This technique is called MICROPROGRAMMING. Do not be confused by this word; it has nothing whatever to do with machine level language or programming a microprocessor. Microprogramming is inherently more powerful than programming in a processor's instruction set for many reasons, not the least of which is the access to the entire functional resources of the hardware on a machine cycle by machine cycle basis. An excellent treatment of microprogramming and microprogrammed machines is available from AMD in previous application notes. Perhaps the easiest to comprehend introduction to this subject is in AMD's *Microprogramming Handbook*. This is highly recommended reading for any newcomer to this area of digital design.

Though microprogramming has always been an inherently more powerful design technique since its invention in 1955, it has been little used until recently (1976), and with some justification. The reason is quite simple. The very large majority of IC's available until the 1976-1978 time frame were specifically designed to be used with 'random logic' design techniques. Since these random logic IC's were poorly suited to the highly structured nature of well designed microprogrammed systems, the potential advantages of microprogrammed systems could not be realized easily.

Fortunately for all of us, in the mid 1970's AMD made a significant decision to develop a very extensive family of Schottky technology IC's specifically optimized for use in microprogrammed systems. These circuits belong to the Am2900 family as well as the Am25S, Am26S, Am27S, and Am25LS families. The acceptance has been so great that many of the other large IC manufacturers are now second sourcing many of these parts and introducing others. So, in just three to four years, microprogrammed machine design has come of age. Now, for most any job of medium to very high complexity, a microprogrammed system is the only way to go if a microprocessor isn't fast or versatile enough.

The purpose of this application note is to illustrate the use of microprogramming and 'bit-slice' technology in a high performance 16-bit time-sharing CPU.This application note is unique in that the CPU being described is the heart of a new commercially available minicomputer system. Thus, it is possible to examine the nature of the CPU as it relates to a complete basic minicomputer system. For this reason, a very short section follows that describes the basic system elements and the system goals toward which the CPU was designed.

The product described herein is called the "HEX-29" CPU. Information on the AMD devices embodied in this application note should be directed to AMD via your local AMD representatives. Inquiries about the HEX-29 CPU and minicomputer system for OEM and/or end user applications should be directed to:

HEX-29
Digital Microsystems, Inc.
4448 Piedmont
Oakland, CA 94661
(415) 658-8532

SYSTEM DESIGN GOALS

In any significant project it is mandatory that reasonable, coherent system design goals be spelled out before serious work is begun. This can be a surprisingly short list of general specifications, but a well thought out system philosophy can make all the difference. Most important, everyone involved should have a copy so everyone will be pulling in the same direction

The following list represents the system design goals for the HEX-29 CPU and system.

1. Compact, reliable, easy to use.
2. Multi-user, multi-task, timesharing.
3. Fast, code-efficient high level language processing.
4. Low cost for complete system.
5. Intelligent microprogrammed channel controllers for high speed I/O.

Indeed, this seems like a short list, but it is the list from which the more detailed specifications were developed. For example, in order to be compact, switching power supply technology is employed. Reliability evolves from many factors including burn in and testing cycles. Probably the single largest cause of 'flakiness' in digital systems is insufficient cooling. An oversize fan moves about five times the volume of air past the IC's as is normally recommended. This large, slower speed fan has the additional advantage that the lower frequency 'white noise' it generates is far less annoying than the 'whine' from smaller high speed fans.

So, it is easy to see that many of the more specific details of system design will fall readily out of these overall design goals. The features of the final HEX-29 system are shown below. It should be instructive to trace each of these features to one (or more) of the design goals listed above. Reviewing this list will also prepare the context for the more detailed sections to follow in later sections.

HEX-29 FEATURES

VERY FAST
- 160ns basic machine cycle
- Only two machine cycles for many instructions
- Microprogrammed clock for increased through-put

COMPLETE SET OF DATA TYPES
- Bit operations
- Nibble operations
- Byte operations
- Word operations
- Double word operations
- Quad word operations
- Variable field operations

EXTENSIVE REGISTER SET
- 16 general purpose/defined purpose registers
- 16 memory management registers
- Extended function condition code register
- 4 interrupt control/status registers

MICROPROGRAMMED
- Expandable instruction set (on board)
- Writable/fixed control store capability
- Integral fixed/floating point processor
- Highly structured, comprehendible, modular design

SOPHISTICATED MEMORY MANAGEMENT
- Multi-user and multi-task timesharing structure
- Complete intertask protection and security
- Megabyte addressing space (expandable)
- Software protectable pages for shared re-entrant coding
- Dual mode operating capability

MULTIPLE STACK PROCESSOR
- Sophisticated program linkage through defined control stack pointer
- Multiple, general register, data stack processing

SOPHISTICATED INTERRUPT STRUCTURE
- 8 level maskable vectored prioritized hardware interrupts
- Second level prioritized expansion on each hardware level
- 256 levels of program controlled software interrupts
- Invalid memory access trap is a vectored interrupt
- Non-existent instruction trap is a vector interrupt
- Breakpoint instruction is a special vectored interrupt
- Automatic mode switching on all interrupts

HIGH THRU-PUT DMA/REFRESH CONTROL
- 8 level prioritized DMA requests and acknowledges
- Up to four Mega-byte/second DMA transfer rate without slowing program execution
- Up to 12 Mega-byte/second DMA transfer rate
- Integral transparent dynamic memory refresh control

EXTENSIVE HIGH LEVEL INSTRUCTION SET
- Multitude of data types handled
- Enormous variety of addressing modes
- General register and defined register classes of instructions
- Many very fast numeric and string macroinstructions
- Integral 16 and 32-bit integer and 64-bit floating point ADD, SUB, MUL, DIV, CMP, NEG, etc.
- Advanced character, byte and word string processing
- Microcoded high level language primitives

VERY HIGH QUALITY PHYSICAL DESIGN
- Four layer P.C. cards throughout system (internal GND and V_{CC})
- All bus signals interleaved with direct return ground path
- All bus signals active low; three-state to inactive level

INTELLIGENT CHANNEL CONTROLLERS
- Microprogrammed floppy disk and hard disk controllers
- Services multiple users I/O simultaneously and transparent to CPU program execution
- Reduces executive program complexity and speeds execution

SOFTWARE SUPPORT
- Multi-user/multi-task time sharing operating system includes sophisticated file management features
- Sophisticated resident macro-assembler
- Customized micro-assembler
- Superfast, super extended BASIC interpreter
- True PASCAL compiler (not interpreter)
- Advanced editor and word processor package
- More software coming

It should be clear from this list that the HEX-29 minicomputer is a powerful/sophisticated design. This is DIRECTLY attributable to the availability of the excellent Schottky technology I.C.'s available from AMD for use in microprogrammed digital systems.

In a well designed microprogrammed system there should be VERY few random logic gate packages required. In the HEX-29 CPU, there are only a few gates used as such. If anywhere near 20% of a microprogrammed system is composed of gate packages, it is probable that the design can be further simplified to replace the random logic with microcode and/or structured logic techniques. It is important to note that the more functions that are implemented with structured logic and controlled by microcode bits, the more versatile and general is the whole design.

MICROPROGRAMMED MACHINES

It is highly recommended that AMD's MICROPROGRAMMING HANDBOOK be studied before this application note if a detailed understanding of the HEX-29 CPU is desired. The idea is, of course, that the basic principles of microprogrammed machines be familiar before this specific example is examined. The Am2900 Learning and Evaluation Kit is also recommended as a practical introduction tool. For those only interested in the capabilities of a well designed microprogrammed CPU, that reading is not entirely necessary, and Section V of this Application note will be superfluous. Section IV is a more general discussion for these readers, but is also necessary for those going on to Section V.

A short discussion of microprogrammed systems appears here only as a short refresher for those who have studied the MICROPROGRAMMING HANDBOOK by John Mick and Jim Brick of AMD.

Any microprogrammed machine can be divided into the following two discrete parts:

1. Control store and microprogram control
2. Data routing and function logic

These two sections of a microprogrammed machine are really quite nearly independent. In effect, the control store and microprogram control section is the 'boss and brains' of the operation. It issues all of the orders and makes all the decisions. The data routing and function logic devices are merely puppets that carry out the commands selected by the microprogram control logic from the control store. Note that 'microword memory' and 'microcode' are used interchangeably with 'control store' and are synonomous.

Control Store and Microprogram Control

The control store is simply a number of PROM's. The number of locations in this memory is chosen to be large enough to hold the desired number of microprogram routines. The width of the word is chosen to have sufficient bits to control all of the possible functions in the data routing and function logic. Admittedly, RAM or EPROM could be used as the memory devices, but it is best to

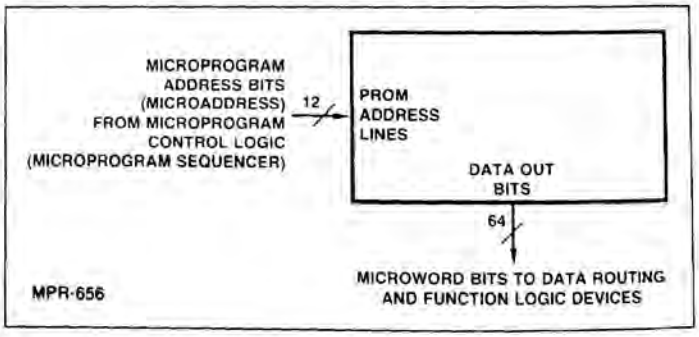


Figure 1.

think of it as an array of read only memory devices. So, schematically an example of a control store array looks like Figure 1.

In practice, there is a register between the microword data bits and the actual data routing and function control devices. This register assures that all bits change simultaneously at the beginning of each new microinstruction cycle and allows the execution of one microinstruction with the fetching of the next. The addition of this 'pipeline register' is shown in the Figure 2 expansion of our block schematic.

The remaining part of this section is the microprogram control unit, more commonly called the microprogram sequencer. The microprogram sequencer is nothing more than a presettable bi-

nary counter with a few extra functions. Figure 3 shows this device in place.

We show the sequencer as a 12-bit binary counter with a few other inputs. The outputs (Y) drive the address lines of the control store PROMs. So, each time the system clock rises, the counters increment and sequential addresses are accessed from the PROM. Note that the current output of the control store is captured in the pipeline register on this same LOW-to-HIGH transition. Thus, the sequencer is always fetching the NEXT control store word which will control the fetching of the next, and so on and so on.

Note that there are several bits from the pipeline register that are routed back to the sequencer. In our example, 12 bits are used as a microword branch address and another bit is used as a preset enable (load) line. Normally, each cycle of the system clock increments the sequencer outputs and the next microword is fetched from the control store. However, somewhere down the line we are going to want to branch to a microcode sequence that is not 'in line' with the code that is currently executing. It is very easy to see how this is done.

The microaddress of the routine to which we want to branch is imbedded in the current microword, 12 bits in our example. The microword bit that is connected to the load input of the sequencer is coded to be low on this cycle. So, the sequencer, which is really just a 12-bit counter with a unique load control in our example, will cause the branch address we selected to pass through to the output of the counters and fetch the microword from the microaddress to which we branched. The routine will now continue to execute sequentially addressed microwords until we execute another branch code.

The only other really necessary function we need from our sequencer is the ability to do conditional branches. In other words, we want to be able to branch to some microcode routine, but only if a certain condition exists. As usual, this capability is easily added; only one multiplexer is needed. Figure 4 shows the new configuration.

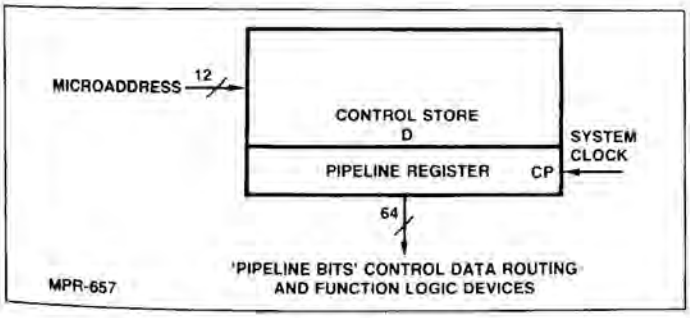


Figure 2.

Now two additional microword bits control the conditions under which a microbranch will take place. If input 0 is selected, a branch will always take place since the logic LOW level on input 0 will appear at the load input of the sequencer. Conversely, if input 3 is selected, a HIGH logic level is always routed through the multiplexer to the load input and a load is not performed. Thus the next sequential microinstruction is fetched. So far we can do branch and continue functions with the multiplexer.

If we select inputs 1 or 2 on the condition select multiplexer, we may get one of two conditions. If the selected input is HIGH, it will be routed to the load input of the sequencer and no load will take place. But if the selected condition is at a LOW logic level, the load input of the sequencer is pulled LOW, a load is performed, and a branch has been accomplished. Since a branch only occurs when the condition bit is LOW, this function is called a 'branch on condition = 0'. Clearly a 'branch on condition = 1' can be implemented simply by inverting the condition bit before it enters the multiplexer.

So as far as controlling the flow of microprograms goes, it is clear that we can make it look very much like assembly language programming of a microcomputer. We can execute sequential microinstructions (in line code), branch conditionally, or branch unconditionally. If we use real live sequencers like the Am2909, Am2910 or Am2911 instead of binary counters we get several other very important functions including micro-subroutining and looping.

When we substitute Am2909's, Am2910's or Am2911's as our sequencers, the final element of our complete microprogram memory and control section is in place. Figure 5 shows this configuration.

The next address PROM of Figure 5 converts the microcode branch function bits into one of two sets of bits that control the function performed by the Am2911's. Which of the two is chosen depends upon the logic level of the particular condition bit that is selected.

This is the basic structure of any microprogram control unit regardless of what the rest of the system looks like. The width of the microword data word, the microaddress field, the condition select field, etc., will change as needed, but the structure remains the same. Note that some of the microword data bits are used to control the microprogram sequencing logic. The bits left over are used to control the data routing and function logic in the device; i.e., everything else!

Data Routing and Function Logic

The data routing and function logic section of a microprogrammed machine closely reflects the job the device is to perform. In this respect there is some similarity with random logic

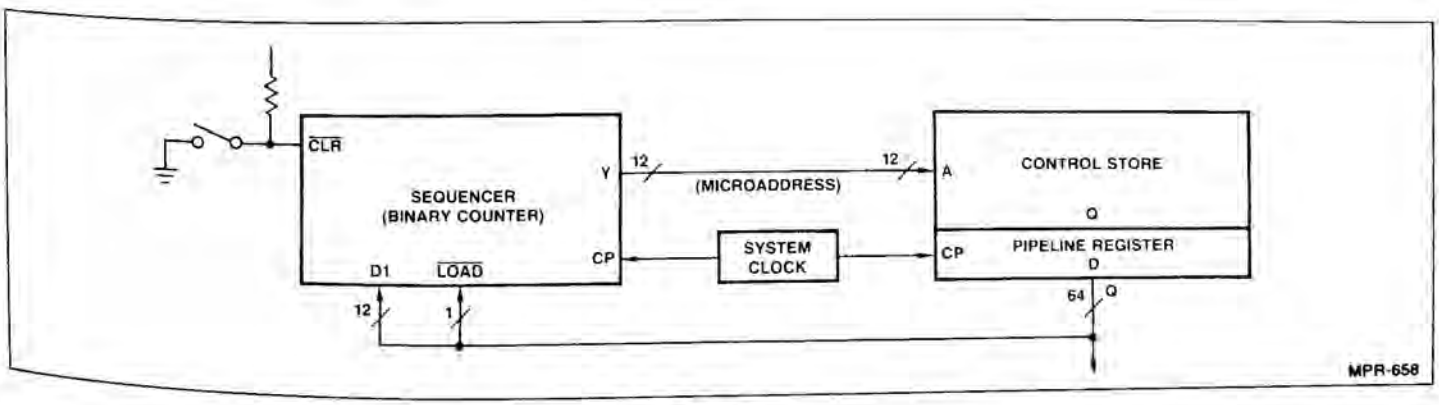


Figure 3.

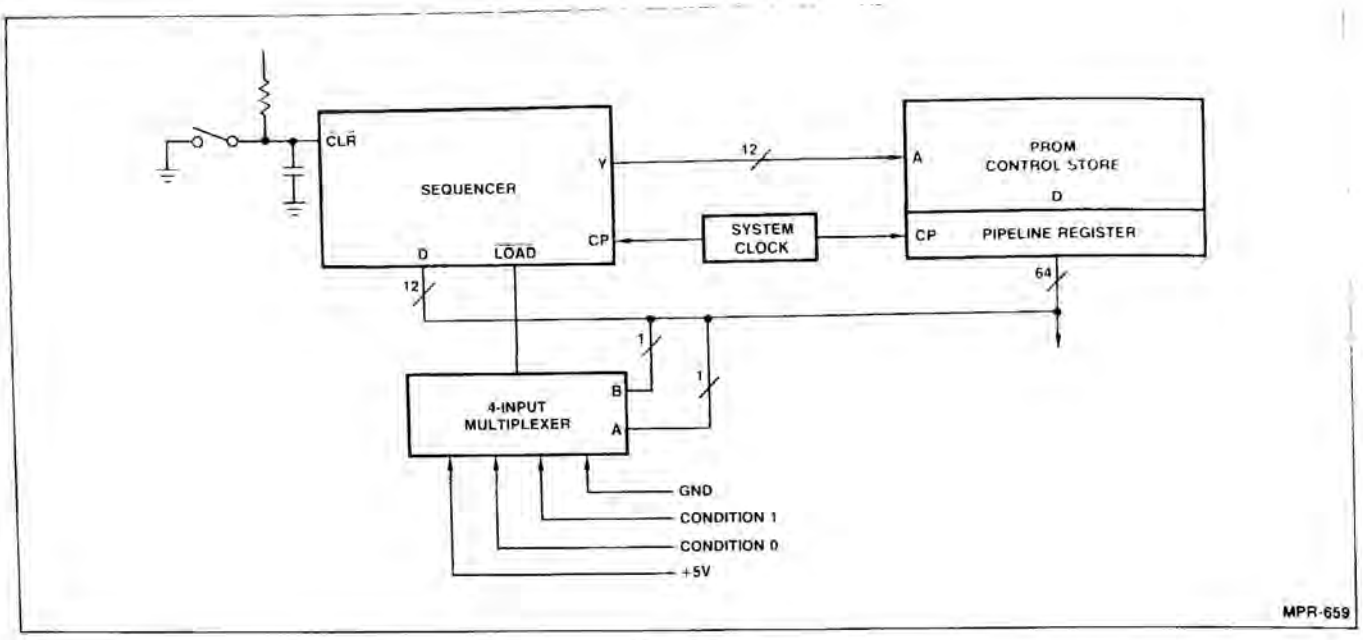


Figure 4.

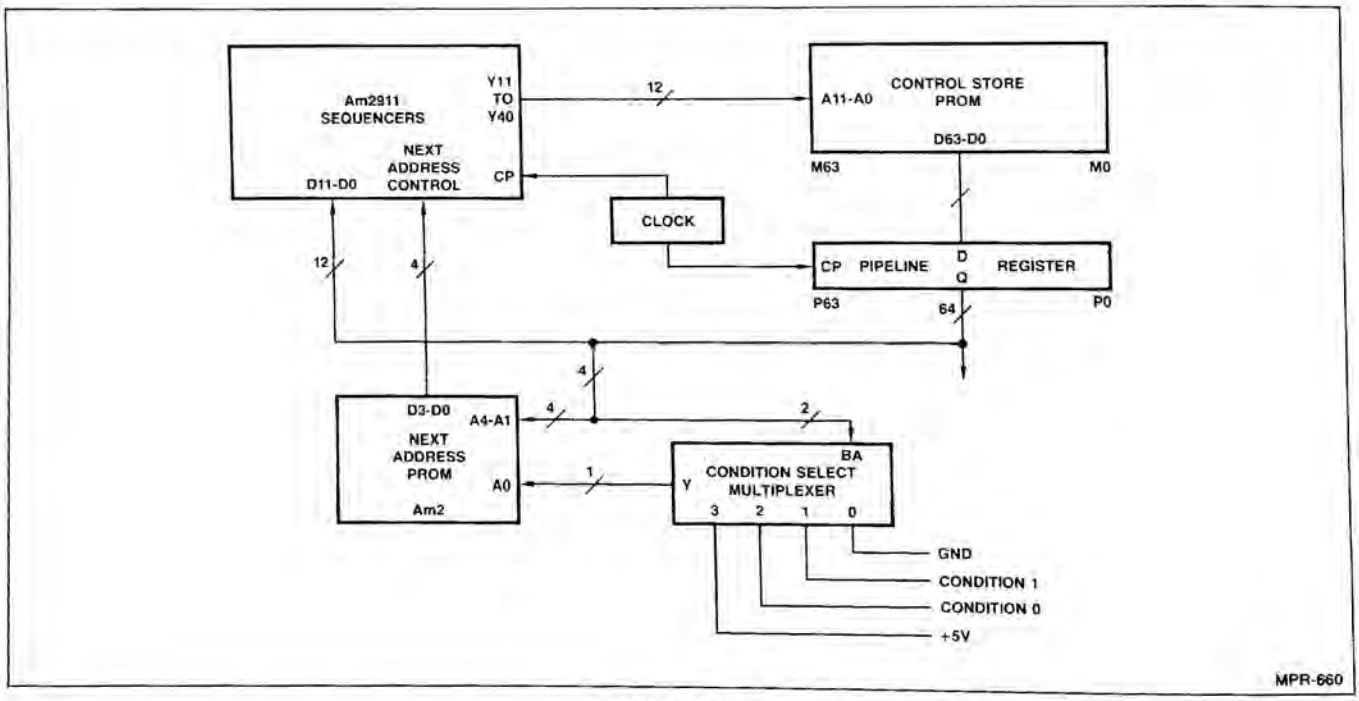


Figure 5.

designs. The key difference is the glue that binds all of the small functional units that make a device work. In a random logic design it is a more or less random array of gates and flip-flops that interconnect and control these functional units.

The chief advantage of a microprogrammed machine is that this random logic is largely replaced by the coherent sequences of control bits that is the microprogram. Problems such as race conditions, undesirable interactions between functional elements and marginal timing nearly disappear in a microprogrammed design. Often there are one or two internal data buses on which all transfers of internal data between functional units take place.

Think of several possible sources of information that may be needed in a particular design. If they are all three-statable devices, microword bits could be tied to the output enable of each and the desired device enabled onto the internal bus on a microcycle by microcycle basis. Likewise one or more devices may capture this data. Microword bits attached to the clock pulse (CP) inputs of registers and the like can achieve this function.

Further, microword bits select other functions to be performed, for example an ALU or shift function. Much of Section V of this application note will demonstrate the use of these data routing and function logic control bits.

GENERAL SPECIFICATIONS

The following section of this application note explores the design of the HEX-29 CPU on an intermediate level. It will be similar in detail to the detailed hardware and software specifications given for most microprocessors by the manufacturers. In other words, all the information needed to use the HEX-29 CPU, including bus timing and instruction set, are examined. This will serve to demonstrate what can be achieved in a medium level microprogrammed machine. It will also serve as a necessary transition for those planning to study the more detailed internal structure of the CPU in the next section of this application note.

It is very important, when designing a microprogrammed machine, that the target device be specified in detail approaching that given in this section. Only then can an intelligent attempt at hardware design begin. It is especially important to define a clean, simple, reliable interface between the microprogrammed device and other system elements. Considerable attention should also be paid to defining data types, instruction formats, interrupt requirements, etc.

Internal CPU Registers

The HEX-29 CPU has 36 internal registers. Of these, 16 are memory management (map) registers, 16 are general purpose registers, three are associated with the interrupt structure, and one is the condition code register.

Table 1 shows the functions associated with the 16 general purpose registers of the HEX-29. It is most significant that all 16 general purpose registers have alternate functions. This should not imply that they are not true general purpose registers however. Any register can be used as an accumulator, stack pointer, index register, memory pointer, data counter, etc., in most instructions. To increase coding efficiency and execution speed, however, some instructions use the defined register assignments in Table 1.

TABLE 1.

Name	Alternate Name	Alternate Function
RF	PC	Program Counter
RE	SP	Stack Pointer
RD	RD	Data Passing
RC	Y	Y Index Register
RB	X	X Index Register
RA	A	Accumulator
R9	CT	Counter
R8	SC	Scratchpad
R7	R7	FP1 (LSW)
R6	R6	FP1
R5	R5	FP1 (MSW)
R4	R4	FP1 (EXP)
R3	R3	FP0 (LSW) DW1 (LSW)
R2	R2	FP0 DW1 (MSW)
R1	R1	FP0 (MSW) DW0 (LSW)
R0	R0	FP0 (EXP) DW0 (MSW)

Notes: FP1 = Floating Point Register 1.
FP0 = Floating Point Register 0.
DW1 = Double Word Register 1.
DW0 = Double Word Register 0.

For example, the instruction set of the HEX-29 CPU can load immediate, push, pop, and move indexed and direct any of the multiple register combinations (FP1, FP0, DW1, DW0) in one instruction. One mode of indexed addressing and many byte processing instructions benefit greatly from the alternate use of some registers.

Condition Code Register

The condition code register contains all zeros in its upper byte. The bit assignments in the low byte are shown in Table 2.

TABLE 2. CONDITION CODE REGISTER BITS.

Position	Name	Function
Bit 7	U2	User Flag #2
Bit 6	U1	User Flag #1
Bit 5	U0	User Flag #0
Bit 4	H	Half Sign Flag (Bit 7; MSb of low byte)
Bit 3	Z	Zero Flag
Bit 2	N	Negative Flag (MSb of result)
Bit 1	V	2's Complement overflow flag
Bit 0	C	Carry Flag (arithmetic and shift carry)

The user flags (U2, U1, U0) are an extra feature of the HEX-29 CPU. They are not altered by any but five special flag modification instructions (SETF, CLRF, COMF, POPF, LDF). These op codes set, clear, complement, pop, or load the flags respectively. Since the user flags are immune to change except by these special purpose flag altering instuctions, they are excellent for passing status information between routines.

The half sign flag (H) is set if the result of an operation contains a 1 in the most significant bit of the low byte; otherwise it is cleared. This flag is useful in many byte processing and loop counting routines.

If the result of an operation is zero, the zero flag (Z) is set, or else it is cleared. This is the most useful of all the flags and is used on comparisons, arithmetic and logical operations, loop counting, etc. . . .

When the most significant bit of the result of an operation is a logic 1, the negative flag (N) is set. Otherwise it is cleared. Note that in two's complement notation, the most significant bit of a number determines the sign of the number. If it is a logic 1, the number is negative; if it is a logic 0, the number is positive.

If the two's complement result of an arithmetic operation results in a two's complement overflow, the V flag is set. This flag is also used as a general error flag by the HEX-29 CPU. For example, the V flag is set if a divide by zero instruction is attempted. In floating point notation, if the exponent becomes too large or small, (arithmetic overflow/underflow), the V flag is set to so indicate.

The carry flag (C) is used for two purposes. It is a source and/or destination bit in shift and rotate instructions, and as a carry-out bit when an arithmetic function result is too large to fit in the appropriate destination register. The convention with regard to the carry flag on addition and subtraction follows:

C flag = 1 if 1. Binary add results in a carry out.
2. Binary subtract results in no borrow.

C flag = 0 if 1. Binary add results in no carry out.
2. Binary subtract results in a borrow.

All of the condition code flags, except the user flags, have some special meanings in some of the complex 'macro' instructions. These are described in the detailed section on the HEX-29 instruction set.

Interrupt Registers

There are three special purpose interrupt registers in the HEX-29 CPU. They are:

1. Mask Register
2. Status Register
3. Vector Register

These registers are command driven, that is, the register selected is a function of the interrupt command being executed. More detailed information on the nature of these registers appears later in this application note.

Memory Management Registers

A sophisticated memory management structure is embodied in the HEX-29 CPU. Integral to this structure is the set of 16 memory map registers. These 8-bit registers contain transformation values that allow multiple users and tasks to share the processing time of the CPU without interacting with each other. Each task logged onto the HEX-29 is unique from all others through its memory map image. When it is chosen to run on the CPU, its memory map image becomes synonomous with the CPU memory map registers. More detailed information on this aspect of the HEX-29 CPU appears later in this application note.

Instruction Formats

The instruction formats of the HEX-29 CPU are simple and few in number. For this reason, the HEX-29 instruction set is not difficult to learn and use, even though it is very extensive and quite sophisticated.

Emphasis on the use of 4-bit (hexadecimal), and 8-bit (byte) fields in the instruction formats simplify the organization of the instruction set. All of the instruction formats used in the HEX-29 are shown in Figure 6.

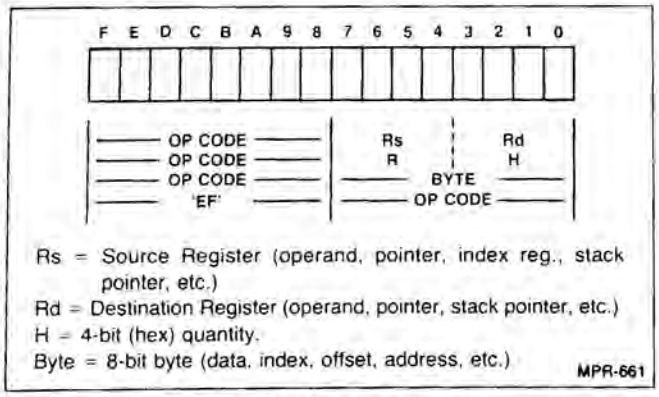


Figure 6. Instruction Formats.

Most instructions involve operations on 16-bit words. However, the HEX-29 instruction set also includes op-codes that operate on the following data types:

1. 1 Bit (Bit)
2. 4 Bits (Hex or Nibble)
3. 8 Bits (Byte)
4. 16 Bits (Word)
5. 32 Bits (Double Word)
6. 64 Bits (Quad Word: Floating Point)
7. N Bits (Variable Format)

In addition to working on the fixed length data types, there are many 'macro' instructions that operate on variable length character, byte, and word strings in memory. These strings can be either contiguous in memory or in the form of linked lists. Several of these 'macro' instructions are highly optimized microcoding of the most critical routines used in high level language processing.

The multiplicity of data types processed efficiently by the HEX-29 increases its ability to meet the diverse demands of modern computing.

Addressing Modes and Assembly Language

Much of the power and simplicity of the HEX-29 instruction set is derived from the large number of useful addressing modes available for the most used basic functions such as MOV, ADD, SUB, INC, DEC, CMP, etc. Addressing modes specify where operands of an instruction are to be found and where the result is to be stored.

The 16 general purpose 16-bit registers are designated R0, R1, R2 . . . RC, RD, RE, RF. These are the primary names of the 16 registers and refer directly to the corresponding registers. In other words, when 'RD' is written in a HEX-29 Assembly Language (HAL) program, the contents of this register are used as an operand or destination in the instruction.

The use of a register as a pointer to memory is called memory pointer addressing. The names M0, M1, M2, . . . MC, MD, ME, MF apply to the 16 general purpose registers when they are used as memory pointers.

When a register points to a memory location which contains the address of the memory location holding the value of interest, the register is said to be an "indirect pointer". The names I0, I1, I2, . . . IC, ID, IE, IF are used to specify the 16 general purpose registers when they are being used with this type of addressing.

Indexed addressing is possible using the names Z0, Z1, Z2, . . . , ZC, ZD, ZE. The use of one of these names means that the data is at the address formed by adding the contents of the register referenced to the contents of word following the instruction in main memory.

Most often, when a register is used as a memory pointer (MD for example), or as an indirect pointer (I9 for example), it is extremely desirable that the register auto-increment or perhaps auto-decrement since programs, lists, and stacks are ordered in a positive direction through memory.

In HEX-29 Assembly Language (HAL) it is quite simple to specify that a memory, indirect pointer register, etc. is auto-incremented or auto-decremented by appending a '+' or '−' character to the respective register specification.

For example:

MOV M7+, R6 The contents of memory pointed to by R7 is moved into R6. R7 is then incremented.

MOV RA, ME− Decrement RE. Then move the contents of RA into the memory location pointed to by RE.

It is significant that auto-incrementing takes place after the operation while auto-decrementing takes place before the operation. (auto-post-increment and auto-pre-decrement.)

Several very fundamental addressing modes arise from auto-incrementing memory and indirect pointers. Consider the following examples:

A. Program Counter (RF) as an auto-incrementing pointer yields 'immediate addressing'.
   MOV MF+, RA = Move immediate into RA.
   ADD MF+, R6 = Add immediate to R6.
   MOV MF+, RF = Jump to address in immediate word.

B. Stack Pointer (RE) as an auto-incrementing pointer yields 'stack addressing'.
   MOV ME+, R2 = Pop top of stack into R2.
   XOR ME+, R1 = Pop top of stack and XOR into R1.
   MOV ME+, RF = Return from subroutine!

C. General registers used as data stack pointers.
   ADD MD+, MD = Add top two members of data stack + leave result on top of the stack.
   CMP MD+, M8+ = Compare top members of two stacks + remove these values from the stacks.
   AND MF+, M6 = AND immediate word with the top member of stack pointed to by R6.

D. Program Counter (RF) as an auto-incrementing indirect pointer yields 'direct addressing'.
   MOV IF+, R7 = Move direct into R7.
   ADD IF+, RC = Add direct into RC.
   MOV IF+, IF+ = Move direct to direct.

It should be clear that these examples represent only a few of the most useful of many possible uses of auto-incrementing and auto-decrementing with memory and indirect pointers. Careful study of the HEX-29 instruction set will reveal many more uses not examined in these examples.

Classes of Instructions

The instruction set of the HEX-29 includes many different functions and a multitude of addressing modes. Nonetheless, all instructions fall into one of two classes of instructions. The general register class of instructions are extremely flexible because of the enormous number of variations inherent in each op-code. The defined register class of instructions permits extremely fast and memory efficient code for often used functions and register sets. The power of the HEX-29 instruction set is derived from an extensive combination of the most powerful and efficient instructions from each class.

General Register Instructions

In a general register instruction, the function and addressing mode are specified in the op-code field (upper byte). The lower byte then holds two 4-bit (hex) values that specify the registers used in the instruction. It should be clear, therefore, that for every general register instruction there are 256 possible specific actions that can be performed.

The full power of these instructions may not be evident without an example. A discussion of just 5 of the 256 possible variations on the MOV M+, R instruction will demonstrate the extreme flexibility of each and every general register instruction. Execution of the MOV M+, R instruction proceeds as follows:

1. Contents of Rs are moved to the address bus.
2. Rs is auto-incremented by one.
3. The data addressed by Rs is loaded into Rd.

In this instruction, Rs is used as an auto-incrementing memory pointer, hence the M+ notation. Rs and Rd are the source on destination registers. Below is the set of 5 examples of how the one op-code can be used to implement a number of important functions.

1. MOV MF+, R3 (W) = Load immediate word (W) into R3.
2. MOV MF+, RF (W) = Jump direct to address (W).
3. MOV ME+, R6 = Pop top of control stack into R6.
4. MOV MD+, R4 = Load next member of list into R4.
5. MOV ME+, RF = Return from subroutine.

Taking a few minutes to review this section and understand how all of these functions are achieved with the single MOV M+, R op-code should reveal the nature of the power and flexibility of the general register class of instructions.

Defined Register Instructions

A defined register instruction is an instruction whose function, addressing mode, and register assignments are all defined in the op-code field (upper byte). The low byte is then available for use as an offset for short relative branching instructions, an immediate byte or character, an 8-bit index value, an 8-bit logical mask, etc. With this class of instructions, often only one word is required for the entire instruction. This speeds execution and improves coding efficiency markedly in most applications. It is for this class of instruction that the alternate register function assignments appear in the model of the HEX-29 register set. An example of a one word defined register instruction with the two word instruction it can replace follows:

```
ADC X, A     26     Defined Register Instruction
ADC ZB, RA   0026   General Register Instruction
```

Both of the instructions accomplish the same thing. In both cases RB (X index register) is used as an index register and RA (Accumulator) is the destination operand. The value in memory at the address pointed to by the sum of the X index register (RB) plus the hex constant 26 is added to the contents of the Accumulator (RA) and the sum left in Accumulator (RA).

The significant difference between the two instructions is that the defined register instruction takes only half as much code (one word vs. two), and executes faster since there are fewer memory accesses and fewer machine cycles. Very often the defined register instruction will be adequate for the job. But when the choices of registers RB and RA are not acceptable or if an 8-bit index offset is not large enough, the general register instruction would be the proper choice. It allows any register pair to be specified as the index register and destination/accumulator and has a 16-bit index offset in the word following the instruction.

As mentioned earlier, it is largely the ability of mixing defined and general purpose instructions freely that makes programs written in HEX-29 Assembly Language very code efficient and fast.

HEX-29 Instruction Set

The HEX-29 instruction set is quite extensive. It not only offers all of the basic functions in a wide variety of addressing modes, it also includes a multitude of special purpose instructions. These special purpose instructions cover many important aspects of programming including program control, numeric processing, string manipulation and searching, list processing, etc.

Fortunately, all of these types of instructions fall into one of only four different instruction formats. These were shown in Figure 6. Table 3 shows all of the instructions for the HEX-29 machine.

TABLE 3. SUMMARY OF MNEMONICS ARITHMETIC OPERATIONS.

ADC	Add words plus carry
ADD	Add words w/o carry
ADDB	Add byte to word
ADDH	Add hex value (nibble) to word
DADD	Add double word values (32 bits + 32 bits → 32 bits)
FADD	Add floating point values (64-bit FP + 64-bit FP → 64-bit FP)
SBB	Subtract with borrow
SUB	Subtract w/o borrow
SUBB	Subtract byte from word
SUBH	Subtract hex value from word
RSUB	Subtract words in reverse order
DSUB	Subtract double word values (32 bits − 32 bits → 32 bits)
FSUB	Subtract floating point values (64-bit FP − 64-bit FP → 64-bit FP)
UMUL	Unsigned word multiply (16 bits * 16 bits → 32 bits)
SMUL	Signed word multiply (16 bits * 16 bits → 32 bits)
DMUL	Double word signed multiply (32 bits * 32 bits → 64 bits)
FMUL	Floating point multiply (64-bit FP * 64-bit FP → 64-bit FP)
UDIV	Unsigned word divide (16 bits ÷ 16 bits → 16 bits + 16-bit remainder)
SDIV	Signed word divide (16 bits ÷ 16 bits → 16 bits + 16-bit remainder)
DDIV	Double word signed divide (32 bits ÷ 32 bits → 32-bit + 32-bit remainder)
FDIV	Floating point divide (64-bit FP ÷ 64-bit FP → 64-bit FP)
CMP	Compare words
CMPB	Compare byte with word
CMPBA	Compare byte with byte
CMPH	Compare positive hex value (nibble) with a word
CMPHN	Compare negative hex value (nibble) with a word
CMPHA	Compare hex value (nibble) with another nibble
DCMP	Compare signed double word values
FCMP	Compare floating point values
NEG	Negate word (2's complement)
DNEG	Negate signed double word value
FNRM	Normalize floating point number
DTST	Test signed double word value for zero + sign
FTST	Test floating point value for zero + sign
INC	Increment word by one
DEC	Decrement word by one

Shifts & Rotates

ASR	Arithmetic shift right
ASL	Arithmetic shift left
CSL	Count and shift left (until MSb=1)
DSL	Double word shift left
DSR	Double word shift right
LSR	Logical shift right
RCL	Rotate closed left
ROL	Rotate left (through carry flag)
ROR	Rotate right (through carry flag)
VSL	Variable shift left (0 to 15 places)
VSR	Variable shift right (0 to 15 places)

TABLE 3. SUMMARY OF MNEMONICS ARITHMETIC OPERATIONS. (Cont.)

Logical Operations

AND	Boolean AND words
ANDB	Boolean AND byte with word
IOR	Boolean inclusive OR words
IORB	Boolean inclusive OR byte with word
XOR	Boolean exclusive OR words
XORB	Boolean exclusive OR byte with word
COM	Complement word
CLR2	Clear the specified 2 registers
BTS	Bit set
BTC	Bit clear
BTI	Bit invert
BTT	Bit test
CLRF	Clear specified flags
SETF	Set specified flags
COMF	Complement specified flags

Data Movement

MVN	Move, no flags altered
MOV	Move, update flags
MVM	Move multiple words
MVB	Move a byte
LDB	Load a byte
STB	Store a byte
MVH	Move a positive nibble
MVHN	Move a negative nibble
LDI2	Load immediate 2 registers
XCH	Exchange contents of two registers
DXCH	Exchange contents of DW1 and DW0
FXCH	Exchange contents of FP1 and FP0
XCHM	Exchange top two members of any stack
DUP	Duplicate top member of any stack
SWT	'Switch'. Store register indexed and reload indexed
JAM	Move any bit field from one word to another
SWP	Swap high and low bytes in a word
PSH2	Push any two registers onto control stack
POP2	Pop top two words on control stack into two registers
PSHF	Push flags (condition code register) onto control stack
POPF	Pop top of control stack into condition code register
PSH8	Push 8 registers onto control stack
POP8	Pop 8 registers from control stack
PSHD	Push R8, R9, RA, RB, RC, RD onto control stack
POPD	Pop R8, R9, RA, RBRC, RD from control stack
LDINT	Load interrupt register
RDINT	Read interrupt register
RMM	Read a memory map location
LMM	Load a memory map location
FMM	Fill memory map
BMBF	Block move bytes forward in memory
BMBR	Block move bytes reverse in memory
BMWF	Block move words forward in memory
BMWR	Block move words reverse in memory

PROGRAM CONTROL

EXR	Execute contents of register as an instruction
RTI	Return from interrupt
BPT	Breakpoint trap
JFS	Jump if specified flags are set
JFC	Jump if specified flags are clear
CFS	Call subroutine if specified flags are set
CFC	Call subroutine if specified flags are clear
JIFS	Jump indirect if specified flags are set
JIFC	Jump indirect if specified flags are clear
CIFS	Call subroutine if specified flags are set
CIFC	Call subroutine if specified flags are clear
RTFS	Return from subroutine if specified flags are set
RTFC	Return from subtoutine if specified flags are clear
JMP	Jump to the address specified
CALL	Call subroutine
CEX	Call executive (software interrupt)
BGT	Branch if greater than
BGE	Branch if greater than or equal
BLT	Branch if less than
BLE	Branch if less than or equal
*TTWB	Transition table word branch
*TTBB	Transition table byte branch
DBNZ	Decrement and Branch Non-Zero
BZD	Branch on zero or decrement
*CBB	Compare and branch if in bounds
BR	Branch
BSR	Branch to subroutine
BC	Branch if carry flag set
BNC	Branch if carry flag not set
BV	Branch if overflow flag set
BNV	Branch if overflow flag not set
BN	Branch if negative flag set
BNN	Branch if negative flag not set
BZ	Branch if zero flag set
BNZ	Branch if zero flag not set
BH	Branch if half sign flag set
BNH	Branch if half sign flag not set

Miscellaneous Instructions

NOP	No operation for 2 to 256 cycles
*SCNB	Scan for match with specified byte
*SCNW	Scan for match with specified word
*SEAF	Basic fixed entry length list search
*SEAL	Basic variable entry length linked list search

*These 'macro' instructions are examined in more detail on the following pages.

SUMMARY OF SELECTED 'MACRO' INSTRUCTIONS

UMUL

Unsigned 16-bit multiply
16 bits 16 bits → 32-bit answer
R3 R2 → R3 (MSW of answer)
→ R2 (LSW of answer)
→ R1 (LSW of answer)
→ R0 (MSW of answer)

If V=1 then R0 is not zero (Answer is longer than 16 bits)
If N=1 then MSb of R0 = 1. (No particular significance)
If Z=1 then answer is zero (R1 and R0 are cleared)

SMUL

Signed 16-bit multiply (Two's complement notation)
16 bits 16 bits → 32-bit answer
R3 R2 → R3 (MSW of answer)
→ R2 (LSW of answer)
→ R1 (LSW of answer)
→ R0 (MSW of answer)
If V=1 then answer is longer than 16 bits (overflowed LSW)
If N=1 then answer is negative
If Z=1 then answer is zero (R1 and R0 are cleared)

UDIV

Unsigned 16-bit divide
16 bits / 16 bits → 16-bit answer and 16-bit remainder
R3 / R2 → R2 R3 holds remainder
If V=1 then an attempt to divide by zero was refused
If N=1 then MSB of answer = 1
If Z=1 then answer is zero (R2 = 0 R3 need not be zero)

SDIV

Signed 16-bit divide (Two's complement notation)
16 bits / 16 bits → 16-bit answer and 16-bit remainder
R3 / R2 → R2 R3 holds remainder
If V=1 then an attempt to divide by zero was refused, or overflow
If N=1 then the answer is negative
If Z=1 then the answer is zero (R2 = 0 R3 need not be zero
R3 has sign of numerator

DADD

Double word signed add
32 bits + 32 bits → 32 bits
DW1 + DW0 → DW0 ie. R3,R2 + R1,R0 → R1,R0
The C flag is treated the same as in Rngle word addition
If V=1 then a two's complement overflow occurred
If N=1 then the answer is negative
If Z=1 then the answer is zero

DSUB

Double word signed subtract (Two's complement notation)
32 bits − 32 bits → 32 bits
DW1 − DW0 → DW0 ie. R3,R2 − R1,R0 → R1,R0
The C flag is treated the same as in single word subtract
IF V=1 then a two's complement overflow occurred
If N=1 then the answer is negative
If Z=1 then the answer is zero
If one divides "8000" by "FFFF" (−32768 ÷ −1) the answer is "8000" (+32768). However, 8000 is a negative number in two's complement, so an overflow has occurred

DMUL

Double word signed multiply
32 bits = 32 bits → 64 bits
DW1 DW0 → DW0,DW1 ie. R3,R2 R1,R0 → R1,R0,R3,R
NOTE: The order of the answer words is as follows:
MSW → R2
MSW − 1 → R3
MSW − 2 → R0
MSW − 3 → R1 (LSW)

SUMMARY OF SELECTED 'MACRO' INSTRUCTIONS (Cont.)

The reason for this seemingly unnecessary odd order concerns the results that are desired in DW0 (R0,R1) at the end of the operation. The desired result of 32-bit math operations are nearly always 32-bit answers. However, a 32-bit * 32-bit multiply can generate up to 64 bits. Therefore, the least significant 32 bits of the answer are stored in DW0 where the answer is expected on all double word (DW) instructions. The most significant 32 bits must be stored in DW1, therefore the seemingly reversed order of storage. If the V flag = 0 at the completion of an operation, then only the 32 bits in DW0 are significant and the user program can store this 32-bit double word without fear of losing significant bits. So, in the normal situation where only the least significant 32 bits of the answer is desired and the more significant 32 bits of the answer does not contain significant bits, the answer is where the normal convention specifies; in DW0. If the V flag is found set and it is desirable to save the 64-bit result rather than go to an error routine, a simple DXCH will exchange the contents of DW1 and DW0 and leave the 64-bit answer in a logical order with the MSW in R0 and the LSW in R3. It can then be buffered with any of the floating point register 0 buffer instructions. If V=1 then the answer has greater than 32 bits of significance.
If N=1 then the answer is negative
If Z=1 then the answer is zero

DDIV Double word signed divide (Two's complement notation)
32 bits / 32 bits → 32-bit answer and 32-bit remainder
DW1 / DW0 → DW0 Remainder → DW1
If V=1 then attempted divide by zero was refused, or overflow
If N=1 then answer is negative
If Z=1 then answer is zero (DW0 = 0. DW1 not tested)

DCMP Double word compare (Two's complement notation)
32 bits − 32 bits → Nowhere (Update V,N,Z flags)
DW1 − DW0 → Nowhere
The C flag is treated the same as in a single word compare
If V=1 then a two's complement overflow occurred
If N=1 then the difference is a negative value
If Z=1 then the difference is zero

DXCH Double word exchange
Operates on any contents of DW1 and DW0
DW1 → TEMP DW0 → DW1 TEMP → DW0
DW1 = R3 and R2
DW0 = R1 and R0
No flags are altered

DNEG Double word negate (Two's complement notation)
0000 0000 − 32 bits → 32 bits
0000 0000 − DW0 → DW0
If V=1 then a two's complement overflow occurred DW0 = 8 0000 0000
If N=1 then the final value in DW0 is negative
If Z=1 then the final value in DW0 is zero

TST DW0 Double word test value (Two's complement notation)
Set flags based upon the contents of DW0
0000 0000 + DW0 → Nowhere (Update V,N,Z)
If V=1 then a valid 2's complement value overflows the LSW
If N=1 then the value in DW0 is negative
If Z=1 then the value in DW0 is zero

FPADD Floating point add Double Precision (64 bits)
Standard HEX-29 floating point format
FP1 + FP0 → FP0
If V=1 then an overflow in the 2's complement exponent occurred
If N=1 then the answer is negative
If Z=1 then the answer is zero

FPSUB Floating point subtract Double Precision (64 bits)
Standard HEX-29 floating point format
FP1 − FP0 → FP0
If V=1 then an overflow in the 2's complement exponent occurred
If N=1 then the answer is negative
If Z=1 then the answer is zero

SUMMARY OF SELECTED 'MACRO' INSTRUCTIONS (Cont.)

FPMUL Floating point multiply Double Precision (64 bits)
Standard HEX-29 floating point format
FP1 FP0 → FP0
If V=1 then an overflow in the 2's complement exponent occurred
If N=1 then the answer is negative
If Z=1 then the answer is zero

FPDIV Floating point divide Double Precision (64 bits)
Standard HEX-29 floating point format
FP1 / FP0 → FP0
If V=1 then an overflow in the 2's complement exponent occurred or negative zero refused.
If N=1 then the answer is negative
If Z=1 then the answer is zero

FPCMP Floating point compare Double Precision (64 bits)
Standard HEX-29 floating point format
Compare the magnitudes of FP1 and FP0
If N XOR V = 1, then FP1 < FP0
If Z=1 then FP1 = FP0

NOTE: WE HAVE TO FURTHER DEFINE THE WAY THIS WORKS, BUT THIS INSTRUCTION WILL SET THE FLAGS SUCH THAT THE 2's COMPLEMENT BRANCH ON THE EF PAGE WILL WORK!!!

FPNRM Floating point normalize Double Precision (64 bits)
Standard HEX-29 floating point format
The sign of the mantissa must be in the MSb of the exponent word before this instruction is executed
Shift mantissa left and increment exponent until MSb of the MSW of the mantissa is one. (Operates on FP0 only)
If V=1 there was a 2's complement overflow of the exponent
The C flag is trashed
N=1 result is negative
Z=1 result is zero

FPXCH Floating point exchange Double Precision (64 bits)
Operates on any contents of FP1 & FP0 (R7 thru R0)
FP1 → TEMP FP0 → FP1 TEMP → FP0
FP1 = R7, R6, R5, R4
FP0 = R3, R2, R1, R0
No flags are altered

TST FP0 Floating point test Double Precision (64 bit)
Standard HEX-29 floating point format
Set the flags based upon the contents of FP0
If N=1 then the value in FP0 is negative
If Z=1 then the value in FP0 is zero

SEAL BASIC string variable / numeric or string matrix element search
The SEAL instruction provides a very flexible way to rapidly and efficiently search linked lists for a particular entry. In each entry in the list, the first two 16-bit words are ordered as follows: The first word of each entry is the link offset to the next entry in the linked list. The second word is the entry name word. Any 16-bit value can be used in this field.

The name of the entry to be searched for must be put in the accumulator (RA) before this instruction is executed. The format of the instruction is as follows:

SEAL F,Md where F is the literal binary value 1111.

The destination field of the instruction (Md) specifies the register that must point to the beginning of the linked list. Starting at this point, this instruction will link its way thru the list looking for a match between the word after the link offset word (the entry name) and the contents of the accumulator (RA).

At the completion of the instruction, the Z flag indicates the results of the instruction in the following manner:

Z=1 No match was found in list (End of list reached)
Z=0 A match was found and Md is pointing to the word after the entry name that matched the accumulator

Since the link offset word is a two's complement value, it can link to any other location in memory. The link offset is equal to the difference between the address of the next link offset word and the address of the current link offset word, minus one.

Note that this instruction can be used to search linked lists with entry names that are much longer than 16 bits with ease. For example, if the entry names to be matched are 2 words long, all that need be done is to compare the word at which the pointer is aimed with the second word of the desired variable name. If it matches, then the pointer now points to the first element in the list after the double word entry name. If it does not match, the search can be continued by backing up the pointer to the link offset of the current entry and re-executing the SEAL instruction.

At the completion of the instruction, the contents of the register specified by the Md field in the instruction will contain the address of the word AFTER the variable name in the list entry that matched the one in the accumulator (RA). At the completion of the instruction the Z flag will indicate the results of the instruction execution. If the Z flag is at a zero level, the search was successful and the pointer to the table (Md) contains the appropriate value. On the other hand, if the Z flag is set to a one level, no match to the variable name in the accumulator was found anywhere in the linked list.

LO VN da da ... da da LO VN da ... da da LO VN ...

LO = Link Offset word
VN = Variable Name word
da = data entries irrelevant to instruction

SEAF Basic fixed link offset variable search

The SEAF instruction provides a very flexible way to rapidly and efficiently search lists for a particular entry. It is slightly different from the SEAF instruction in that the link offset word is not imbedded in the list entries. Instead, this instruction assumes that all list entries are of the same length (even though the internal formats may vary). The value of the link offset is the immediate word following the BASF op code word.

Perhaps the most obvious use of this instruction is for searching a numeric variable list for a specific variable name followed by the value. The lists entries can be any length, so single and double word integers and floating point lists can all be handled with equal ease, but not all with the same instruction since the list entries will not be the same length for all of these.

The link offset word following the instruction is a two's complement number. Therefore, any fixed length can be searched forwards or backwards in memory. The link offset constant equals the number of words in each list entry, or its 2's complement for a backwards search.

Again the variable name word to be searched for must be put into the accumulator (RA) before the BASF instruction is executed. And the contents of the destination field register (specified by Md) points to the first element of the list. The form of this instruction is shown below:

SEAF 0,Md where 0 = binary 0000

SCNW Scan for word

The SCNW instruction is of the following form:

SCNW Ms,Md

This instruction scans a table of words (pointed to by Rs) for a match with the contents of the accumulator. Each time a word is fetched from the table, Rd is incremented. If Rd contains zero at the beginning of the instruction, then it will contain the number of the words searched in the source table before a match with the accumulator occurred.

Alternatively Rd may contain a pointer to another table. When a match between the accumulator and the source table occurs, Rd will point to a corresponding entry in the 'destination' table.

If the source list pointer and the destination list pointer are the same, then the two tables are interleaved; ie. the combined list would start:

Source list word #1
Destination list word #1
Source list word #2
Destination list word #2
Source list word #3
etc.
etc.

This instruction can be very useful in command processing routines and for searching lists that are not linked within the list itself (see BASS and BASF).

The last entry in the source list must be a zero. If no matches were found previous to this zero word, then the Z flag is set. If the Z flag was not set, then a match was found and the pointers are valid. This instruction is interruptable on a word by word basis.

SCNB Scan for byte

The scan for byte instruction (SCNB) works identically to the scan for word instruction except that the source list contains bytes packed into words. Thus the source list is only half as long as the destination list (if there is one).

Note that both lists must start on word boundaries. Only the low byte of the accumulator is used in the compare with the source bytes. The contents of the accumulator are not affected by the instruction. This instruction is interruptable on every other byte that is compared. The Z flag has the same meaning as for the SCNW instruction.

Instruction Matrix

A convenient way to present all of the basic op-codes of the HEX-29 CPU is by way of an 'instruction matrix'. The eight-bit op-code in the upper byte is broken into two nibbles. The most significant nibble of the op-code appears on the left side of the matrix shown in Figure 7. The lower nibble appears along the top row. The second matrix shown in Figure 8, is called the 'extended function' matrix. In the HEX-29 CPU, the low byte of the instruction word is interpreted as an 'extended function' op-code if the upper byte is an 'EF' hex.

Memory Management

The HEX-29 incorporates a sophisticated memory management structure. Though very clean and elegant in implementation, the capabilities of the processor are greatly extended by this circuitry. Transparent to the user not requiring its many features, this structure is vital to many very important applications; most significantly the support of multi-user, multi-task, time-sharing operations.

To all programs executing on the HEX-29, all memory addresses are 16 bits long. But before these 16 lines reach the system bus, they pass through the memory management section of the HEX-29 CPU. In this circuitry, the most significant four bits (A15-A12) are 'mapped' into eight bits on the bus (a 'write-protect' bit (WP) and seven address lines (A18-A12)).The net increase of three address bits expands the total addressable memory space to 512k words or 1 Megabyte. The WP bit is used to write protect the memory in blocks as desired by the executive program.

Since each of the 16 locations in the memory map represents a 4k word block (or page), up to 64k words can be addressed by a program at any time. Any location in the memory may contain any 8-bit value, so memory that is contiguous to a program need not be contiguous in physical memory. For clarity, Figure 9 shows schematically how this 'memory mapping' works.

The low 4k words of physical address space is reserved for the nucleus of an operating system, also called an executive or supervisor program. This is called physical page zero. The contents of the memory map can only be altered if the low location of the memory map contains all zeros. Since this is synonymous with the physical page zero address block, only the executive program is able to change the contents of the memory map. And since all I/O devices and channel control blocks are located in physical page zero, all I/O must also be done through the executive program. Likewise, all hardware and software interrupts invoke the supervisor automatically.

Because of this simple but fool-proof security scheme, complete protection of all users memory space and I/O devices can easily be maintained by the executive program.

Also note that the supervisor program can safely make programs that are re-entrant available to several users simultaneously as long as it write protects the code. Since user programs are often no larger than the host program under which it is running, this technique can result in a savings of 30% to 50% in system memory usage.

Occasionally, for special purposes, a single user may wish sole access to the entire resources of the system. Examples would include programs too large to run in a single user's 128k bytes of memory. Or perhaps a new I/O access method. In any case, it is possible for a single user on the system to gain complete control

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	MVN R, R	MOV R, R	ADD R, R	ADC R, R	SUB R, R	SBC R, R	AND R, R	IOR R, R	XOR R, R	CMP R, R	RSUB R, R	INC R, R	DEC R, R	COM R, R	NEG R, R	SWP R, R
1	MVN M+R	MOV M+R	ADD M+R	ADC M+R	SUB M+R	SBC M+R	AND M+R	IOR M+R	XOR M+R	CMP M+R	RSUB M+R	INC M, R	DEC M, R	COM M, R	NEG M, R	SWP M, R
2	MVN I+R	MOV I+R	ADD I+R	ADC I+R	SUB I+R	SBC I+R	AND I+R	IOR I+R	XOR I+R	CMP I+R	RSUB I+R	INC I+R	DEC I+R	COM I+R	NEG I+R	SWP I+R
3	MVN Z, R	MOV Z, R	ADD Z, R	ADC Z, R	SUB Z, R	SBC Z, R	AND Z, R	IOR Z, R	XOR Z, R	CMP Z, R	RSUB Z, R	INC Z, R	DEC Z, R	COM Z, R	NEG Z, R	SWP Z, R
4	MVN X, A	MOV X, A	ADD X, A	ADC X, A	SUB X, A	SBC X, A	AND X, A	IOR X, A	XOR X, A	CMP X, A	RSUB X, A	INC X, SC	DEC X, SC	COM X, SC	NEG X, SC	SWP X, SC
5	MVN M+M	MOV M+M	ADD M+M	ADC M+M	SUB M+M	SBC M+M	AND M+M	IOR M+M	XOR M+M	CMP M+M+	RSUB M+M					
6	LDI2 R, R	CLR2 R, R	PSH2 R, R	POP2 R, R	XCH R, R	ASR R, R	ASL R, R	ROR R, R	ROL R, R	LSR R, R	RCL R, R	CSL R, R	VSR R, R	VSL R, R	DSR R, R	DSL R, R
7	BTS R, H	BTC R, H	BTI R, H	BTT R, H	MVH R, H	MVHN R, H	ADDH R, H	SUBH R, H	CMPHA R, H	CMPH R, H	CMPHN R, H	FMM R, H	VSR R, H	VSL R, H	EXR R	JAM R, R, W
8	BTS Z, H	BTC Z, H	BTI Z, H	BTT Z, H	ANDB B, A	IORB B, A	XORB B, A	SWT R, Z	MOV A, Y	MOV Y, A	LDBI R, R	STBI R, R	XCH M DUP M	COMF B	MVN CC, R	MOV R, CC
9	MVB B, A	MVB Z, Z	MVB Z, R	MVB R, Z	LDB M, M	STB M, M	ADDB B, A	SUBB B, A	CMPBA B, A	CMPB B, A	CMPB Z, Z	CMPB R, Z	CMPB R, R	CMPB M, M	SETF B	CLRF B
A	MOV M, R	MOV I, R	MOV M+M+	MOV M+I+	MOV M+Z	MOV M+M-	MOV A, X	MOV Z, I+	MOV Z, Z	MOV Z, M-	MOV RD, Y	MOV RB, Y	MOV R9, Y	MVM RR, Y	LDINT M+H	LDINT R, H
B	MOV R, M	MOV R, I	MOV R, M+	MOV R, I+	MOV R, Z	MVN R, M-	MOV R, M-	MOV I+I+	MOV I+, Z	MOV I+M-	MOV Y, RD	MOV Y, RB	MOV Y, R9	MVM Y, R, R	RMM R, R	RDINT R, H
C	MVM FP0 M+, DW0	MVM FP1 M+DW1	MVM FP0 DW0, M-	MVM FP1 DW1, M-	MVM FP0 Z, DW0	MVM FP1 Z, DW1	MVM FP0 DW0, Z	MVM FP1 DW1, Z	MVM X, FP0	MVM X, FP1	MVM FP0, X	MVM FP1, X	MVM X, DW0	MVM X, DW1	MVM DW0, X	MVM DW1, X
D	JFS B	JFC B	CFS B	CFC B	JIFS B	JIFC B	CIFS B	CIFC B	RTFS B	RTFC B	JMP R	CALL R	CALL X	CALL Y	CALL Z	CEX B
E	BR +L	B-R +L	BC ±B	BNC ±B	BV ±B	BNV ±B	BN ±B	BNN ±B	BZ ±B	BNZ ±B	BH ±B	BNH ±B	DBNZ ±B	BZD ±B	CBB ±B	EF
F	BR -L	BSR -L	CALLO B	JMPO B					BMWF M, M	BMWR M, M	SEAL M SEAF M	SCNW M, M	SCNB M, M	TTWB M, M	TTBB M, M	NOP B

Figure 7. HEX-29 Instructions.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FADD	FSUB	FMUL	FDIV	FCMP	FXCH	FNRM FP0	FTST FP0								
1	DADD	DSUB	DMUL	DDIV	DCMP	DXCH	DNEG DW0	DTST DW0								
2	SMUL	SDIV	UMUL	UDIV												
3	PSHF	PSH8	PSHD	LMM A	RTI											
4	POPF	POP8	POPD		BPT											
8	MVM FP0, FP1	MVM FP1, FP0	MVM ABS, FP0	MVM ABS, FP1	MVM FP0, ABS	MVM FP1, ABS										
C	MVM DW0, DW1	MVM DW1, DW0	MVM ABS, DW0	MVM ABS, DW1	MVM DW0, ABS	MVM DW1, ABS										
D	CALL REL	CALL ABS														
E	BGT	BGE	BLT	BLE												
F	BMBF	BMBR	SRCH													

Figure 8. HEX-29 EF Instructions.

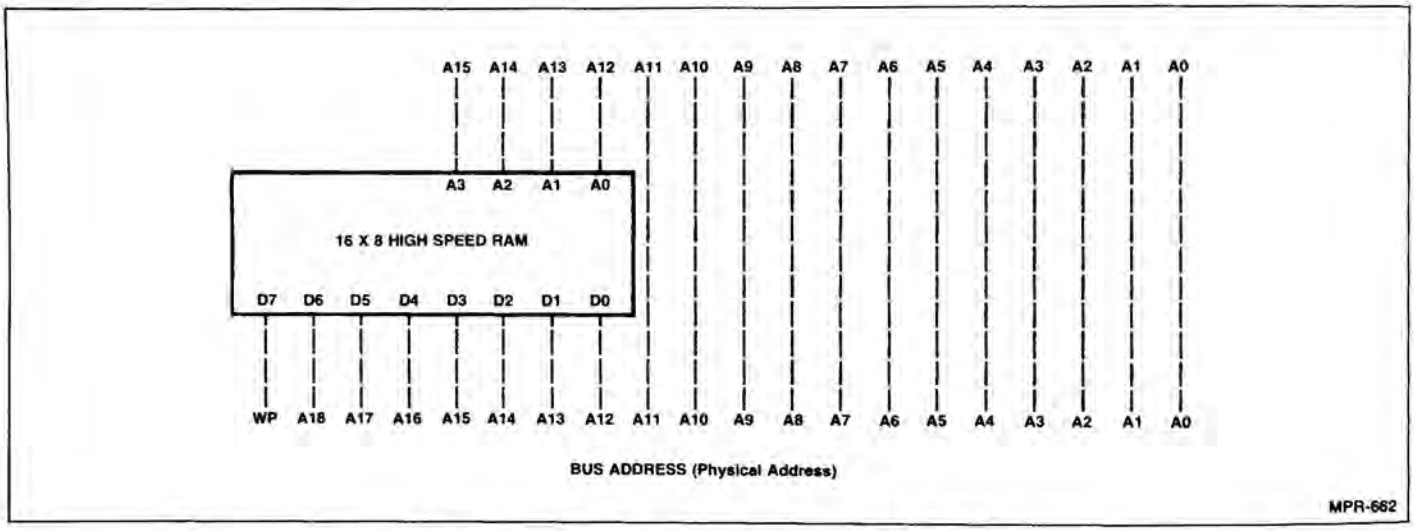


Figure 9. Memory Mapping Program Address (Logical Address).

and access to the system by assigning himself as the executive program. This can only be accomplished after a system reset. Hence only those with physical access to the computer (and who have a reset key) can accomplish this operation. This user is then empowered with all of the features and capabilities of the machine with no limitations. Direct access to all of the system I/O devices, the entire interrupt structure, the memory map, etc., is then at the command of the single user in the executive or supervisor mode.

Most often, each user needs only one or two 4k pages of memory in addition to the host program which is probably shared. Thus it would be very wasteful if each user were to have access to a full 65k words of physical memory space. For this reason, a page of physical memory has a special designation in the system.

The highest possible physical address block when write protected is called the 'invalid access' block. Whenever a user accesses memory that the supervisor has mapped into the invalid access block, the processor 'traps' to a special location in the supervisor program called the 'invalid access trap'. This occurs before the current machine cycle is completed. This is treated identically to an interrupt by the processor except that the current instruction is not completed.

Any number of actions can be taken by the supervisor at this time. This will usually depend upon the resources of the machine and the circumstances under which the problem arose. For example, the executive program could inform the program that its memory space had been exceeded, or perhaps just allocate another block of memory to that user's memory map and continue the execution of the offending program. A more detailed discussion of the sequence of events that takes place upon an invalid access appears in the section on the interrupt structure of the HEX-29 CPU.

The highest physical address page, when not write protected, is called the 'dead page'. No action of any kind takes place in this block and there is no memory there for the program to reference. Any number of pages from any number of users may be assigned to this physical page without fear of interaction. This is the block that will normally be assigned by the executive program to all user areas that are not needed or are not to be used.

Interrupt Structure

The HEX-29 CPU contains a powerful interrupt structure. As with memory management, this aspect of the CPU operation is largely transparent to users of the system. In most applications the HEX OPERATING SYSTEM FOR TIMESHARING (HOST) program services all interrupts. Nonetheless, it is useful to know the basic structure of the interrupt system. The three types of interrupts serviced by the HEX-29 CPU are examined in the following paragraphs.

The hardware interrupts are caused by signals from physical devices outside of the processor. These signals, generated by peripherals, their controllers, or the real time clock, serve to notify the CPU of some condition or requirement of the interrupting device.

The HEX-29 CPU has eight hardware interrupts. They are individually maskable and are prioritized into eight levels. Each priority level has its own vector associated with it. In other words, each interrupt level has a corresponding memory location through which program control is passed upon that level interrupt. These memory locations are within the defined executive page (physical page 0) and thus all interrupts cause the HEX-29 to switch into executive mode automatically. The eight hardware interrupt levels and the associated memory locations are shown below.

Hardware Interrupt Level		Memory Location of Vector
Highest Priority	7	0407_H
	6	0406_H
	5	0405_H
	4	0404_H
	3	0403_H
	2	0402_H
	1	0401_H
Lowest Priority	0	0400_H

So, for example, when an interrupt occurs on level 3, the HEX-29 CPU will enter supervisor mode, save the users PC and SP, and call the appropriate service routine at the address stored in memory location 0403_H.

Normally, each hardware interrupt level is reserved for a class of devices such as hard disc controllers, floppy disc controllers, serial channels, etc. If, for example, there are eight serial devices that are interrupting on level 0, the service routine is required to locate the one (or more) devices that are requesting service on that interrupt level and processes them accordingly. This could be done by polling all the serial devices whenever the interrupt was received. A more efficient technique, used in the HEX-29 system, is to further prioritize the like devices on a given interrupt level. Then when an interrupt occurs, a vector is read by the executive program that instantly informs it of the highest priority device requesting service on that level. When that device is serviced, the vector is read again to locate any other devices in need of service (if any), and finally resumes normal program execution when all devices are serviced.

A software interrupt is an instruction that, when executed, causes an interrupt to occur. The mnemonic used for this op-code in the HEX-29 CPU is 'CEX', which stands for 'call executive'. This instruction passes an 8-bit vector to the 'HOST' operating system which is used to determine the action requested by the program executing the CEX. Except that this interrupt is caused by a program rather than a physical device, the CEX operates in the same manner as a hardware interrupt. It vectors through memory location 040C. A pseudo software interrupt is the breakpoint 'BPT' instruction which vectors through memory location 040B. The BPT instruction does not pass an 8-bit vector to the executive and is thus useful in program debugging.

The third type of interrupt is called a 'trap'. A trap takes place when certain conditions occur that require the processor's immediate attention. For example, if the program currently running on the CPU tries to execute an op-code for which there is no defined instruction, an 'invalid instruction trap' occurs. This is essentially a service to notify a user that his program was defective and that an attempt was made to execute an op-code which has no meaning. These locations are left blank in the instruction matrix since they can subsequently be defined as new instructions. This 'trap' vectors through memory address 040D and acts identically to all other interrupts. The only other trap in the HEX-29 CPU is the 'invalid memory access' condition. This is discussed in more detail in the previous section on memory management. The 'invalid memory access' trap vectors through memory address 0408.

Table 4 shows the memory locations that are defined in the HEX-29 for interrupt handling.

TABLE 4. INTERRUPT MEMORY LOCATIONS.

Memory Location	System Defined Uses
040F	Reserved
040E	Reserved
040D	Vector for invalid instruction trap
040C	Vector for call executive (CEX) instruction
040B	Vector for breakpoint (BPT) instruction
040A	Temperature storage for user stack pointer
0409	Temperature storage for executive stack pointer
0408	Vector for invalid memory access trap
0407	Vector for hardware interrupt level 7
0406	Vector for hardware interrupt level 6
0405	Vector for hardware interrupt level 5
0404	Vector for hardware interrupt level 4
0403	Vector for hardware interrput level 3
0402	Vector for hardware interrupt level 2
0401	Vector for hardware interrupt level 1
0400	Vector for hardware interrupt level 0

Again, note that all interrupts are processed identically so that the one return from interrupt (RTI) instruction properly terminates all interrupt service routines.

DMA/REFRESH CONTROL

In order that an efficient multi-user or multi-task system be implemented, it is necessary that the processor not be burdened with the relatively slow transfer of programs and data between system memory and mass storage devices such as floppy and hard disks. For this reason, the controllers for these devices are designed with a high degree of intelligence and self-reliance. These controllers take virtually all of the burden of mass storage transfers upon themselves. This frees the HEX-29 CPU to execute programs for all users not waiting for these mass storage transfers to take place. Because these controllers are essentially separate special purpose microprogrammed CPUs, they are often called 'peripheral processors', 'channel processors', or just 'channels'.

For this scheme to be effective, both the CPU and the channel processors must be accessing system memory concurrently. Fortunately, the inherent structure and operation of the HEX-29 CPU is eminently suited to this requirement.

In every instruction there is at least one machine cycle during which the HEX-29 CPU is decoding or internally executing an instruction. During these machine cycles the CPU does not use the system bus; the system bus and memory are available for access by devices other than the HEX-29 CPU. This is called a 'Free DMA cycle' or 'bus available' cycle. During these machine cycles a channel processor may read or write memory without interfering with, or assistance from the HEX-29 CPU. The act of accessing system memory by any device other than the CPU is called 'direct memory access' or DMA since the channel processor is directly accessing system memory without CPU assistance or intervention.

Resident in the HEX-29 CPU is a very clean, very powerful multi-level prioritized DMA structure. Within this structure up to ten groups of devices can share the system bus on a priority basis. Normally the priority levels are assigned on the basis of transfer speeds . . . the faster the device is able to support memory transfers, the higher the priority it is assigned. In this manner several channel processors can access system memory concurrently at the intervals they require. The DMA structure of the HEX-29 CPU can support very high combined transfer rates with multiple DMA devices using this technique. With high speed memory, the HEX-29 CPU need not even slow down its program execution to support a concurrent combined DMA transfer rate of 4 Megabytes per second. With slower memory, this figure drops to about 2 to 3 Megabytes per second. Even this slower rate corresponds to concurrent DMA by one high speed hard disk plus several floppy disks plus room to spare. Still, the CPU can be halted, if necessary, to achieve combined DMA rates of up to 12 Megabytes per second maximum.

The support of dynamic memory in the HEX-29 system is simplified by signals associated with this DMA structure. Whenever there are no devices requesting the bus for DMA, a signal on the bus indicates this condition. Dynamic memory refresh controllers can take advantage of these unused free DMA cycles to refresh internal dynamic RAM chips if desired. Even when very heavy use of the bus by DMA devices occurs, it is unlikely that too few of these unused free DMA cycles will be available for the dynamic memory refresh controllers. In this event, however, another signal can be used to disable all other DMA priorities and allow the refresh controllers as much time as is required.

SYSTEM BUS AND TIMING

When specifying the bus signals and their timing relationships during the early design stage of the HEX-29 CPU, utmost attention was paid to simplicity and reliability. The result is that there are very few signals required to interface to the bus properly, and the timing requirements are quite straight forward and easy to meet.

The following section is a description of the mnemonic names and functions of the HEX-29 system bus signals:

System Bus

A18-A0 (Address Bus) — Three-state outputs. A18-A0 are the 19 physical address lines of the HEX-29 system address bus. A18 is the most significant bit, A0 is the least significant bit. These outputs are three-stated whenever the bus is available ($\overline{BA}$ is low).

D15-D0 (Data Bus) — Three-state and bi-directional input/outputs. D15-D0 are the 16 lines that make up the HEX-29 system data bus. D15 is the most significant bit, D0 is the least significant bit.

WP (also $\overline{WE}$) (Write Protect) — Three-state output. WP is used to protect areas of memory from being written. Practically speaking this signal is active-LOW and would have been called $\overline{WE}$ (Write Enable) if not for possible confusion with the read/$\overline{write}$ signal which also must be LOW to write memory.

R/$\overline{W}$ (Read/$\overline{Write}$) — Three-state output. The R/$\overline{W}$ signal determines whether a read or write operation is performed. A LOW level of the R/$\overline{W}$ line indicates a write memory is to be performed if VMA (valid memory access) is also LOW when the system clock (CLK) goes LOW.

$\overline{VMA}$ (Valid Memory Access) — Three-state output. $\overline{VMA}$ is LOW during all cycles that a memory access (read or write) will be performed by the processor.

CLK — Output, not three-state. CLK is the system clock. All timing in the HEX-29 system is defined relative to this signal. For convenience, the period of each machine cycle that the clock is high is called ϕ_1 (phase 1) and the period that it is low is called ϕ_2 (phase 2). All system 'chip selects' are derived from this signal.

$\overline{SDMA}$ — Output, not three-state. $\overline{SDMA}$ is mnemonic for 'synchronize direct memory access'. This bus signal is LOW the cycle before DMA is permissible. The sole purpose of this signal is to notify DMA devices early of an upcoming 'free DMA' cycle. This will make it easier to 'grab the bus' very early in a 'free DMA' cycle to improve the address generation timing.

$\overline{BA}$ (Bus Available) — Output, not three-state. $\overline{BA}$ is LOW on all cycles during which DMA is permitted by the CPU. When $\overline{BA}$ is LOW, all three-stateable outputs from the HEX-29 CPU card are turned off and control is relinquished to DMA devices for the current cycle. $\overline{BA}$ is mnemonic for 'bus available'.

$\overline{STR}$ (Stretch Clock) — Input to HEX-29 CPU. When an addressed device is not fast enough to be reliably accessed (read or written) within the minimum access time of the HEX-29 CPU, it should pull the $\overline{STR}$ signal LOW. For each 40ns that $\overline{STR}$ is held LOW, the system clock is lengthened by 40ns and thus the access time required of the addressed device. This signal can be held LOW for as many as 40ns increments as required to meet the access time of the addressed memory or I/O device.

$\overline{CLR}$ (Clear) — Output, not three-state. $\overline{CLR}$ is a LOW level pulse which is just a 'cleaned up' $\overline{RESET}$ signal. Any device that requires an initialization pulse should use this line.

$\overline{I7}$-$\overline{I0}$ (Interrupts) — Inputs to HEX-29 CPU. $\overline{I7}$-$\overline{I0}$ are the eight hardware interrupt inputs. $\overline{I7}$ is the highest priority and $\overline{I0}$ is the lowest. These inputs are negative edge catching; that is, an interrupt signal is recognized by the interrupt circuitry in the HEX-29 CPU when the line goes LOW. These

lines should be driven by open collector outputs so that multiple devices can interrupt on the same priority level.

$\overline{R7}$-$\overline{R0}$ (DMA Requests) — Inputs to HEX-29 CPU. $\overline{R7}$-$\overline{R0}$ are the eight DMA request inputs. $\overline{R7}$ is the highest priority, $\overline{R0}$ is the lowest. These lines are active-LOW; i.e., a LOW level requests DMA time.

$\overline{Q7}$-$\overline{Q0}$ (DMA Acknowledge) — Outputs, not three-state. $\overline{Q7}$-$\overline{Q0}$ are the eight DMA acknowledge lines that reply to the corresponding DMA request lines ($\overline{R7}$-$\overline{R0}$). A reply to the highest requesting priority is acknowledged by a LOW level on the corresponding acknowledge line. Only one of these lines will be LOW at any given time; i.e., the highest priority request gets the acknowledge.

$\overline{NRQ}$ (No DMA Request) — Output, not three-state. $\overline{NRQ}$ is LOW when no DMA requests ($\overline{R7}$-$\overline{R0}$) are being received. This is used primarily as a signal to dynamic memory refresh controllers that a refresh may be performed on any 'free DMA' cycle.

$\overline{DDMA}$ (Disable DMA) — Input to HEX-29 CPU. When $\overline{DDMA}$ is pulled LOW, no DMA requests are acknowledged. Essentially this line is just the highest priority DMA request line – except there is no corresponding acknowledge signal. This signal is normally reserved for dynamic memory refresh controllers. If the refresh interval is about to expire and some locations have not yet been refreshed, this line can be pulled LOW to disable all other DMA devices and assure adequate time to refresh the remaining locations. Note that $\overline{NRQ}$ is not LOW when $\overline{DDMA}$ is active (LOW). The $\overline{DDMA}$ line should be driven by open collector outputs.

$\overline{HALT}$ — Input to HEX-29 CPU. When pulled LOW, the $\overline{HALT}$ input will cause the processor to terminate program execution at the conclusion of the current instruction. At this time the bus will become continuously available for DMA as all three-state outputs of the HEX-29 CPU will turn off and $\overline{BA}$ will go active (LOW). This line can be held LOW indefinitely. When released, the processor will continue program execution. This line should be driven by open collector outputs.

$\overline{FETCH}$ (Fetch Instruction) — Output, not three-state. This signal is LOW only on memory read cycles when an instruction is being fetched from system memory. Otherwise this signal is normally not used except during system development and debugging for single instruction execution.

$\overline{RESET}$ — Input to HEX-29 CPU. This is the signal from which system reset ($\overline{CLR}$) is derived. Normally this input is simply grounded with a pushbutton or keyswitch to reset the HEX-29 system.

OSC (Oscillator) — Output, not three-state. This is the crystal controlled master oscillator from which the system clock is derived. The period of this oscillator is normally 40ns. (25MHz).

System Timing

In any microprogrammed system which must interface to a number of external devices (as a CPU must), it is critical that considerable forethought be given to the methods of inter-device communication. It is quite common to design and build devices that operate with very high degrees of reliability - only to find that overall system reliability is inadequate when the various devices are interfaced.

One of the utmost goals in designing the HEX-29 CPU was to develop an extremely reliable, easy to use, system bus definition. Simplicity and reliability go hand in hand and this is reflected in the HEX-29 system bus. Perhaps the single most important decision in this regard was to define that all memory and I/O device accesses by the processor or DMA devices would share one set of timing rules. In other words, one set of timing specifications applies to any kind of access of any device by any other device. Some systems have different timing requirements for all sorts of reasons; a few examples are listed here.

1. Memory read timing is more critical (shorter) if the memory being fetched is an instruction.
2. Variations exist in the set-up and hold times required on read memory vs. write memory cycles.
3. Memory devices and I/O devices use some different signals and timing specifications.
4. DMA devices are required to meet a different set of timing requirements than the processor.
5. Interrupt processing routines violate the normal memory access techniques.

Special cases carry special problems and should be avoided like the plague. It is always best and easiest to have all devices and situations share one set of control signals and one set of timing relationships. Another good practice put into effect on the HEX-29 CPU is the exclusive use of active-LOW bus signals. This is important in many respects. First, bipolar logic IC's can sink (pull LOW) far more current than they can source. Thus any noise spikes need to carry far more energy to force the signal into an invalid level. Secondly, all signals that three-state (turn-off) will be pulled-up (float) to the inactive level. Furthermore, this scheme tends to reduce the power required by bus signal drivers and therefore reduce heat dissipation.

Physical design is also important to system reliability. It is wise to use four layer PC cards with GND and V_{CC} planes as the internal layers, as do all of the HEX-29 system cards. An additional feature of the HEX-29 system bus is that all signals are interlaced with GND traces that return directly to the internal GND plane next to each bus signal. System termination should also be provided whenever signals must travel more than 18". Bypass capacitors should abound on all system cards, one per three IC's as a minimum. The HEX-29 averages one per IC.

The timing of each machine cycle in a HEX-29 system is a combination of synchronous and asynchronous characteristics. Actually, all signals are synchronous with – or are synchronized by – the master oscillator from which the system clock is derived. Thus, despite the fact that some signals seem to be asynchronous, they are actually synchronized automatically with the system clock. The simplicity of this approach will become clear once the relationship of all signals to the system clock is explained.

The conventions regarding the HEX-29 system clock are very simple. All machine cycles begin when the system clock goes HIGH and end simultaneously with the begining of the next machine cycle. The period of time that the system clock (CLK) is HIGH is called ϕ_1 (phase 1) and the period of time that it is LOW is called ϕ_2 (phase 2). See Figure 10 for clarification.

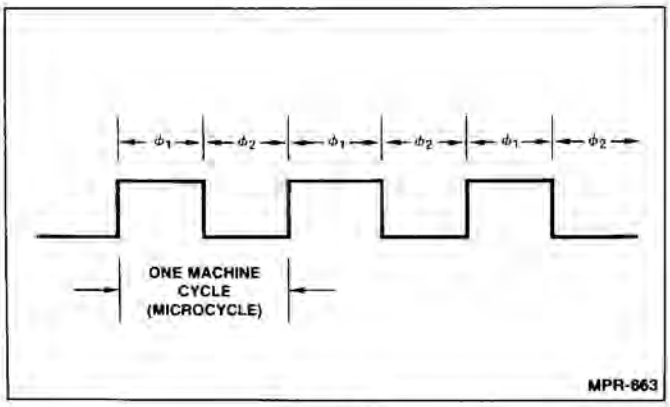


Figure 10.

During all memory and I/O accesses, the processor (or DMA controller) must guarantee that all address lines and control signals are valid for at least 20ns before the end of ϕ_1 (falling edge of clock). Depending upon the addressing mode, the processor will require a variable period of time to generate a valid address. Thus it is the responsibity of the processor to control the period of ϕ_1 to meet its requirements. If no external accesses are made by the CPU, ϕ_1 and ϕ_2 will last only 80ns each unless a DMA device takes control of the bus on that cycle and requires longer times.

Similarly, ϕ_2 is controlled by the memory and I/O devices on the bus. If none are being accessed on a particular machine cycle, no control need be exercised on the system clock and ϕ_2 will last for 80ns. However, when accessed, many memory and I/O devices more than 80ns to perform a successful read or write operation. They must be able to lengthen ϕ_2 of the system clock to increase the access time appropriately. This is accomplished with the $\overline{STR}$ bus signal. When a device is accessed that requires that ϕ_2 be longer than 80ns, it must bring $\overline{STR}$ LOW within 50ns of the falling edge of system clock (i.e., 50ns into ϕ_2). For every 40ns that $\overline{STR}$ is held LOW, the system clock is held in its present state for an additional 40ns. ϕ_2 can thus be extended indefinitely as required by the access time of the addressed device. ϕ_1 can also be extended in 40ns increments with the $\overline{STR}$ signal if so required by DMA devices with slow address generation times, or the like.

A DMA device must gain access to the bus before it can access the memory location that it desires. This is very simple. It simply pulls its DMA request line LOW and waits for the corresponding DMA acknowledge signal to go LOW in reply. Then, at the beginning of the first machine cycle which finds these signals plus $\overline{SDMA}$ LOW, the DMA device has been granted access to the bus and may immediately generate the appropriate signals on the address, data, and control buses to accomplish the transfer. The memory device being accessed does not care whether it is the processor or a DMA device on the bus since the bus signals and timing used by the memory card is identical for both. Thus it controls ϕ_2 with the $\overline{STR}$ signal as necessary and the access is completed in exactly the same manner as if it had been the processor controlling the bus. The Boolean equation for a DMA device gaining access to the bus follows – and Figure 11 is a schematic showing how easy the implementation can be.

$\overline{Q_X} \cdot \overline{R_X} \cdot \overline{SDMA} \cdot CLK$ = DMA device has access for the current cycle

X = any DMA priority level

The timing relationships for the HEX-29 bus are shown in Figure 12.

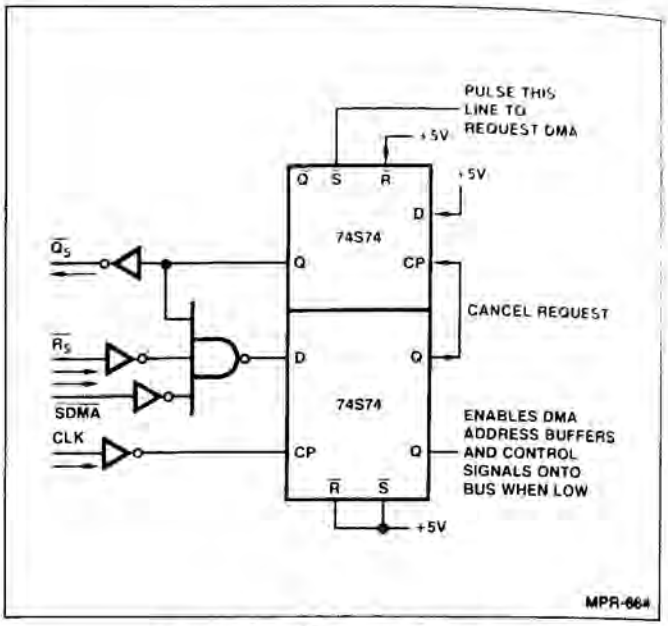


Figure 11. DMA Bus Signals.

INTERNAL OPERATION

Block Diagram

The block diagram of the HEX-29 CPU (Figure 13) shows the following functional modules:

1. System Clock
2. Microprogram Control
3. μWord Memory (Control Store)
4. Am2901A Bit Slice ALU/Register Sets
5. ALU Arithmetic Carry In Control
6. Shift and Rotate Linkages
7. Condition Code Control
8. Am2901A Output Bus
   a. Data Output Latches
   b. Address Latches
   c. Memory Management RAM
   d. Condition Code Register
9. Am2901A Input Bus
   a. Data Bus Input Registers
   b. Byte Swap Input Registers
   c. Microword Data Registers
   d. Clear Byte/Bit Set Logic
   e. Instruction Decode PROMs
   f. Condition Code Register
10. Interrupt Control
11. DMA/Refresh Control

Sections 8 and 9 are more difficult to isolate on the block diagram since they are the buses that connect many function modules together. A full detailed schematic of the HEX-29 is shown in Figure 14; a fold out drawing at the back of the chapter. A discussion of the function of each of the above modules follows.

System Clock (Figure 15)

All timing in the HEX-29 CPU is controlled by the system clock. The positive going edge of the system clock (LOW-to-HIGH transition) marks the end of one machine cycle and the beginning of the next. All input signals to the HEX-29 CPU from the system bus are captured on this edge. The next microinstruction is clocked into the pipeline register on this edge.

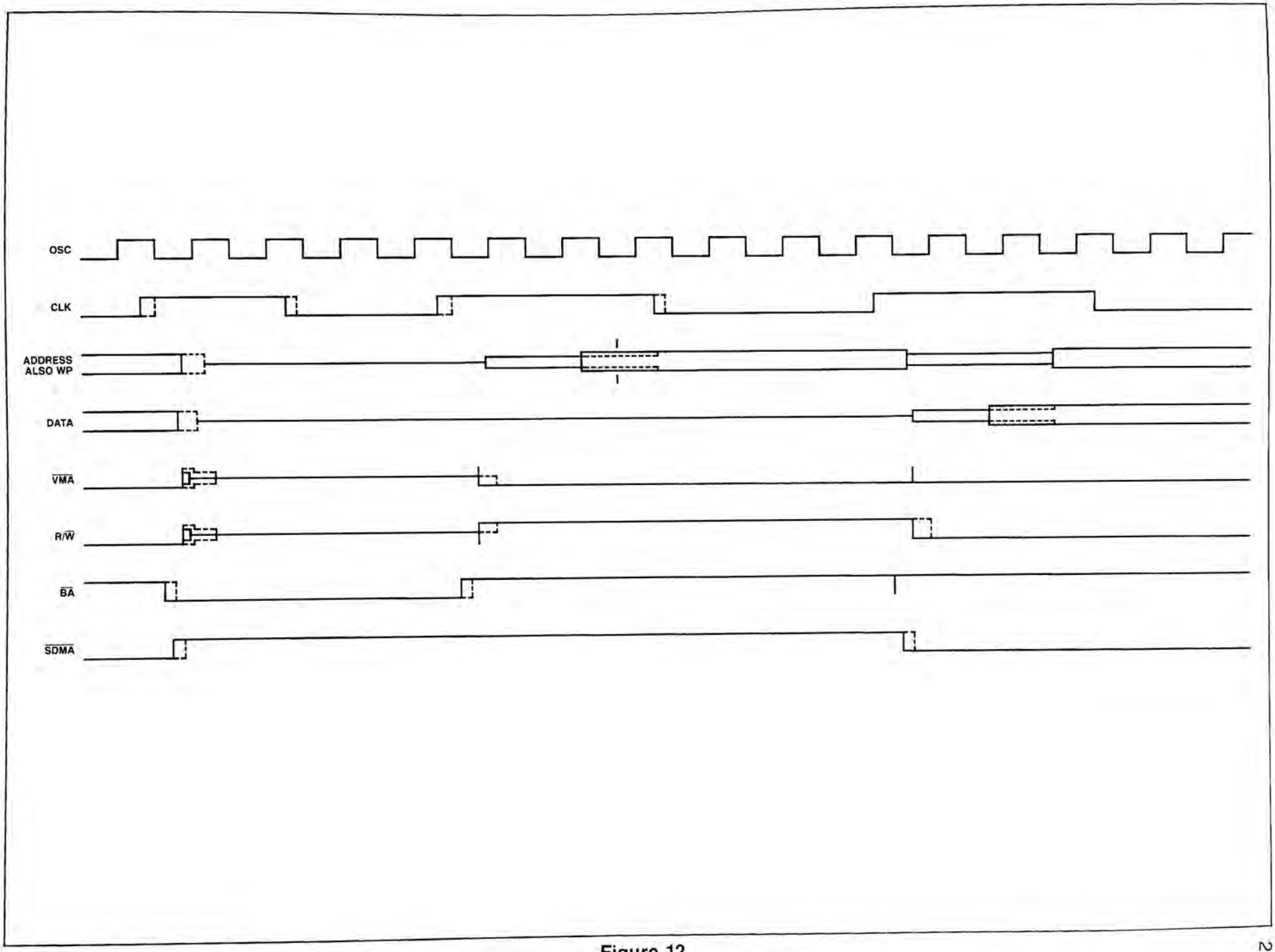


Figure 12.

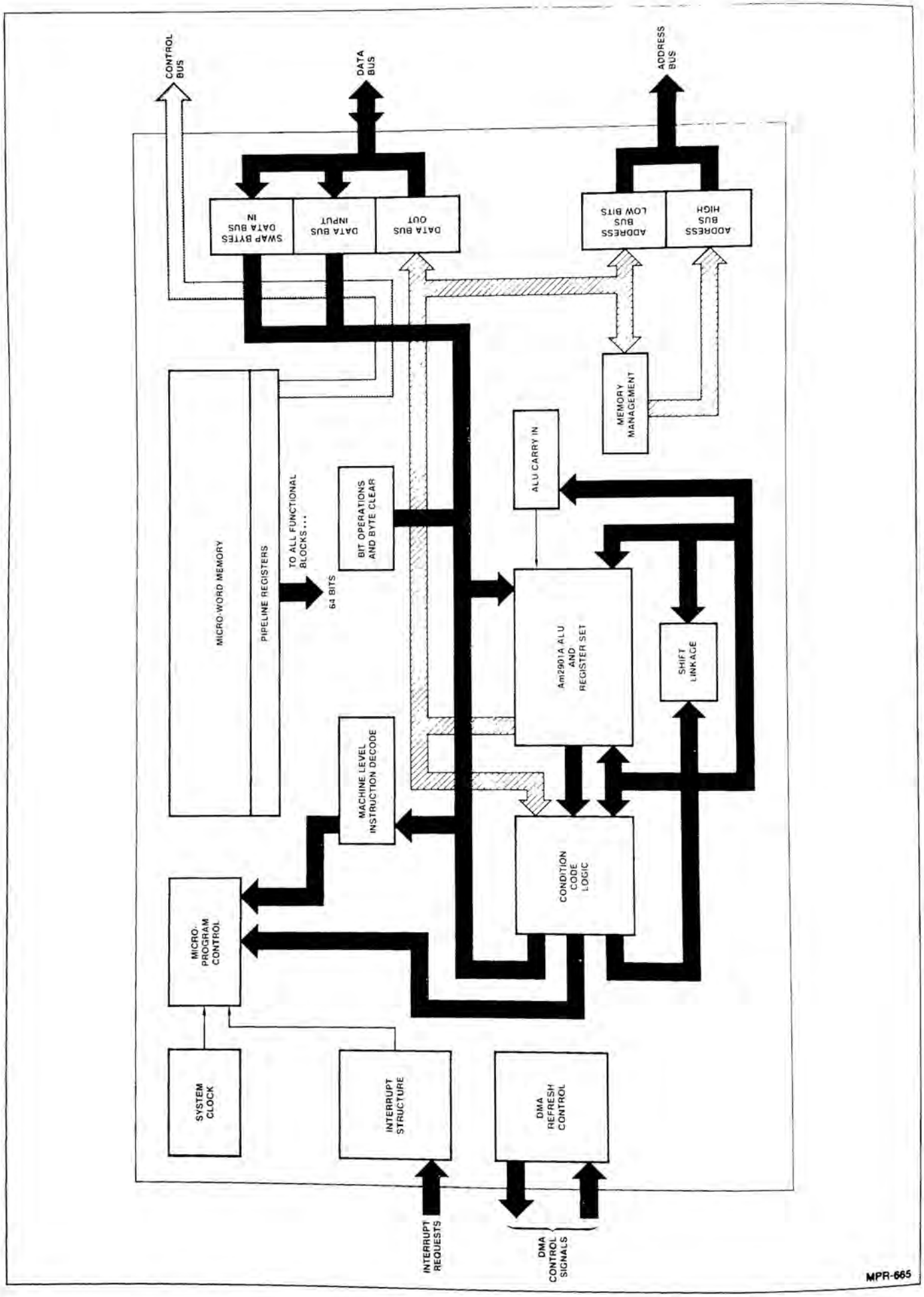


Figure 13. System Block Diagram.

Normally a system clock is a simple square wave or more complex waveform with a fixed period and duty cycle. But the system clock of the HEX-29 CPU is microprogrammed. In other words, the period and duty cycle are selected by microword bits in each microcycle. The advantage of this approach is one of through-put (speed).

In any CPU, some internal operations require longer to execute reliably than others. And one or more of these operations requires the maximum length of time to complete reliably. This is called the worst case delay path or "critical path". Normally the period of time required to perform this "critical path" operation is chosen as the clock period for all instructions.

Since the "critical path" operation may take a factor of 30% to 100% longer to execute than typical operations, it is clear that much processor time is being wasted in any typical program. Two microword bits are used to control the HEX-29 microprogrammed system clock so that each microcycle lasts only as long as necessary for the operation being performed. An overall speed gain of about 30% to 40% is realized with this technique. This was discussed in detail in Chapter II and Chapter III.

The master oscillator from which the system clock is derived is a 25MHz crystal controlled oscillator. Phase 1 (ϕ_1) of the system clock cycle (Figures 10 and 12) is programmed to be 2, 3, 4 or 5 times the 40ns fundamental period of the oscillator. The duration of ϕ_2 of the system clock is 80ns. Since main memory will rarely be as fast as 80ns access time, a method to allow system memory cards to lengthen ϕ_2 is also provided with the $\overline{STR}$ bus signal. When the $\overline{STR}$ signal is low, the Am74S161 is disabled from counting and the state of the clock will not change until it is released and it counts out normally.

The conventions regarding the system clock are very simple and were chosen as the easiest to interface with a variety of memory and I/O devices.

All machine cycles begin when the system clock goes HIGH. The period of time that the clock remains at a HIGH logic level is called ϕ_1. ϕ_2 is the period that it is LOW. During all memory access (and I/O since I/O is memory mapped), the processor guarantees that all address lines and control bus signals (R/$\overline{W}$, $\overline{VMA}$,WP, etc.) are valid and stable at least 20ns before the end of ϕ_1. In other words, the CPU must make all bus signals valid at least 20ns before ϕ_2 begins.

Depending upon the addressing mode being used, the processor will require more or less time to make all necessary signals to the system bus and memory cards valid.

For example, indexed addressing requires an arithmetic operation from the Am2901B's rather than logical operations or a direct pass, therefore indexed addressing is bound to take slightly longer than immediate, direct, or pointer addressing.

It is for these indexed operations and some others that ϕ_1 can be lengthened in 40ns increments by microword bits ST_1 and ST_0. So the processor controls the system clock during ϕ_1 to meet its requirements. When there is no memory access, the minimum 80ns for ϕ_1 is generally more than adequate. Simple addressing modes require 80ns-120ns. The most complex addressing modes can take 160ns to 190ns using the worst case specs for all IC's in the address generation path.

At the end of ϕ_1 (the beginning of ϕ_2), the processor relinquishes control of the system clock to the memory or I/O device that is being accessed. Since I/O is mapped into normal memory space, there is only one set of timing rules for both memory and I/O accesses. If no more than 80ns is required to properly complete the read or write operation, then ϕ_2 will last only 80ns. But the access time of most main memory cards will be greater than 80ns so a way of increasing the duration of ϕ_2 is provided with $\overline{STR}$ bus signals.

If this signal ($\overline{STR}$) is pulled LOW within the first 50ns of ϕ_2, ϕ_2 will be lengthened by 40ns for every 40ns that $\overline{STR}$ is held LOW. Thus ϕ_2 can be extended indefinitely to match the access time of the device being addressed. Naturally this input should be driven by open collector outputs so that all cards can share the one $\overline{STR}$ line.

Though the $\overline{STR}$ signal is intended to be used during ϕ_2 on memory reference cycles, it works in an identical fashion during ϕ_1. This can be used to advantage by DMA controllers that require more than 60ns to generate valid address, data, or control signals on transparent DMA cycles.

A jumper option on the microprogrammable system clock allows the default period of ϕ_2 to be increased from 80ns to 120ns on memory reference cycles only. This is useful in systems where no memory or I/O devices have access times of 80ns or less, and/or when more than 50ns is required to pull $\overline{STR}$ LOW to lengthen ϕ_2. Figure 16 is a table of the ϕ_1 and default ϕ_2 periods available with the microprogrammed clock on the HEX-29 CPU.

ST1	ST0	$\overline{VMA}$	ϕ_1 Period	Default ϕ_2 Period	Default ϕ_2 Period with VMA Option Jumpered
1	1	1	80ns	80ns	80ns
1	1	0	80ns	80ns	120ns
1	0	1	120ns	80ns	80ns
1	0	0	120ns	80ns	120ns
0	1	1	160ns	80ns	80ns
0	1	0	160ns	80ns	120ns
0	0	1	200ns	80ns	80ns
0	0	0	200ns	80ns	120ns

Figure 16. Microprogrammed System Clock Timing.

Microprogram Control

The microprogram control section (Figure 17) of the HEX-29 CPU performs several functions; they are:

1. System reset and initialization
2. Interrupt and halt control
3. Machine level instruction to microinstruction mapping
4. Microinstruction sequencing and microsubroutining
5. Invalid Access Memory Management Trap

When the system reset button or keyswitch is closed, the input to a one-shot is pulled LOW. When it is released, the rising edge triggers a 500 μs pulse. This is synchronized with the system by gating it through a flip-flop driver by system clock. The resulting signal is used to zero the outputs of the Am2909 microprocessor sequencer. Thus, when the one-shot times out, the microprogram will begin execution at microaddress 000. The microcode needed to initialize the system is stored at this and the following several microaddresses and assures the proper system start-up.

Each time a machine level instruction is fetched, the microprogram control logic checks for a hardware interrupt or halt signal from the system bus. If either signal is active, the microprogram branches to the appropriate microinstruction address to execute the appropriate microcode to service the request. The interrupt routine will buffer user registers, switch to supervisor mode, and call a machine level routine through a vector table element as defined by the priority level of the interrupt. If the halt

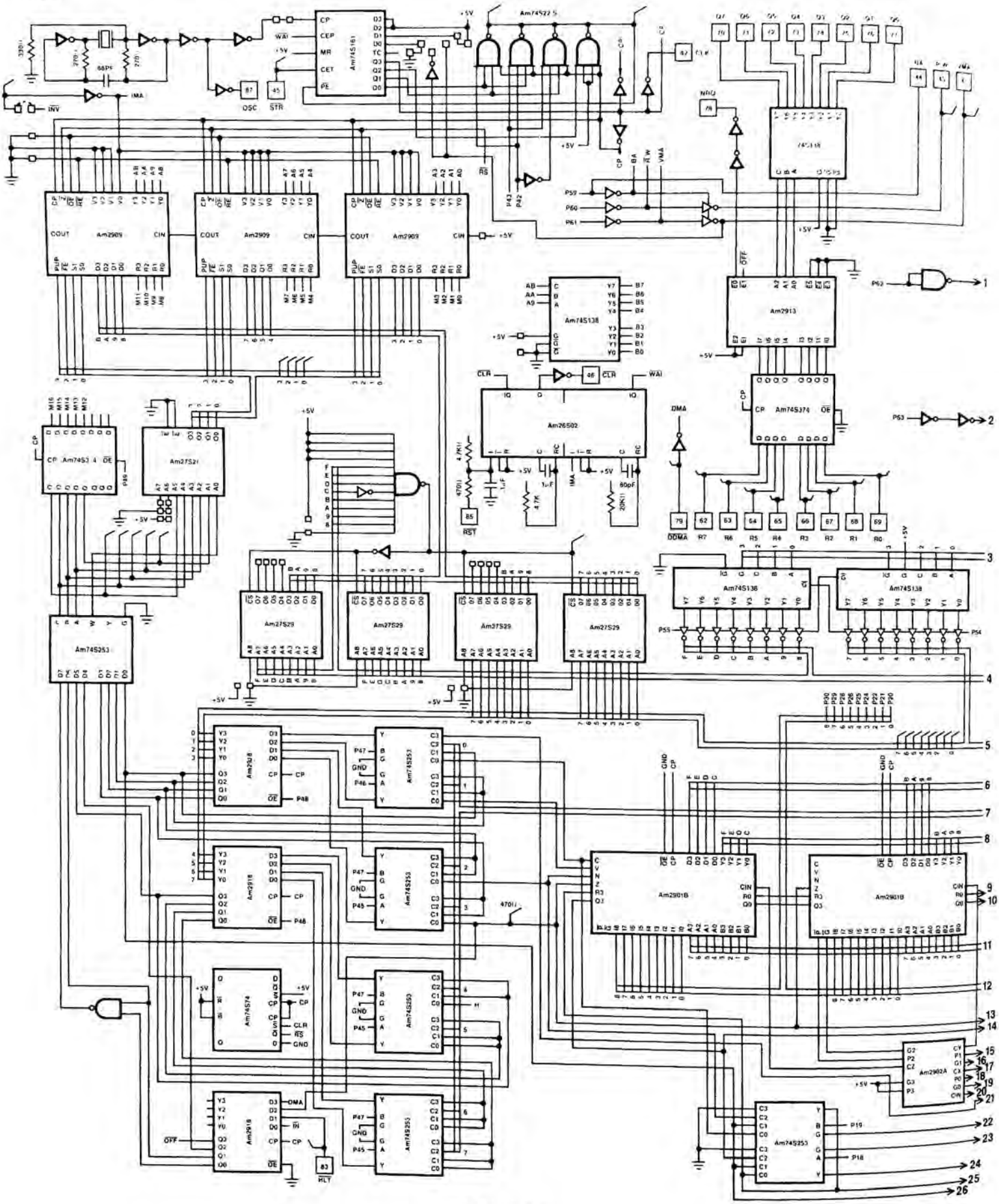
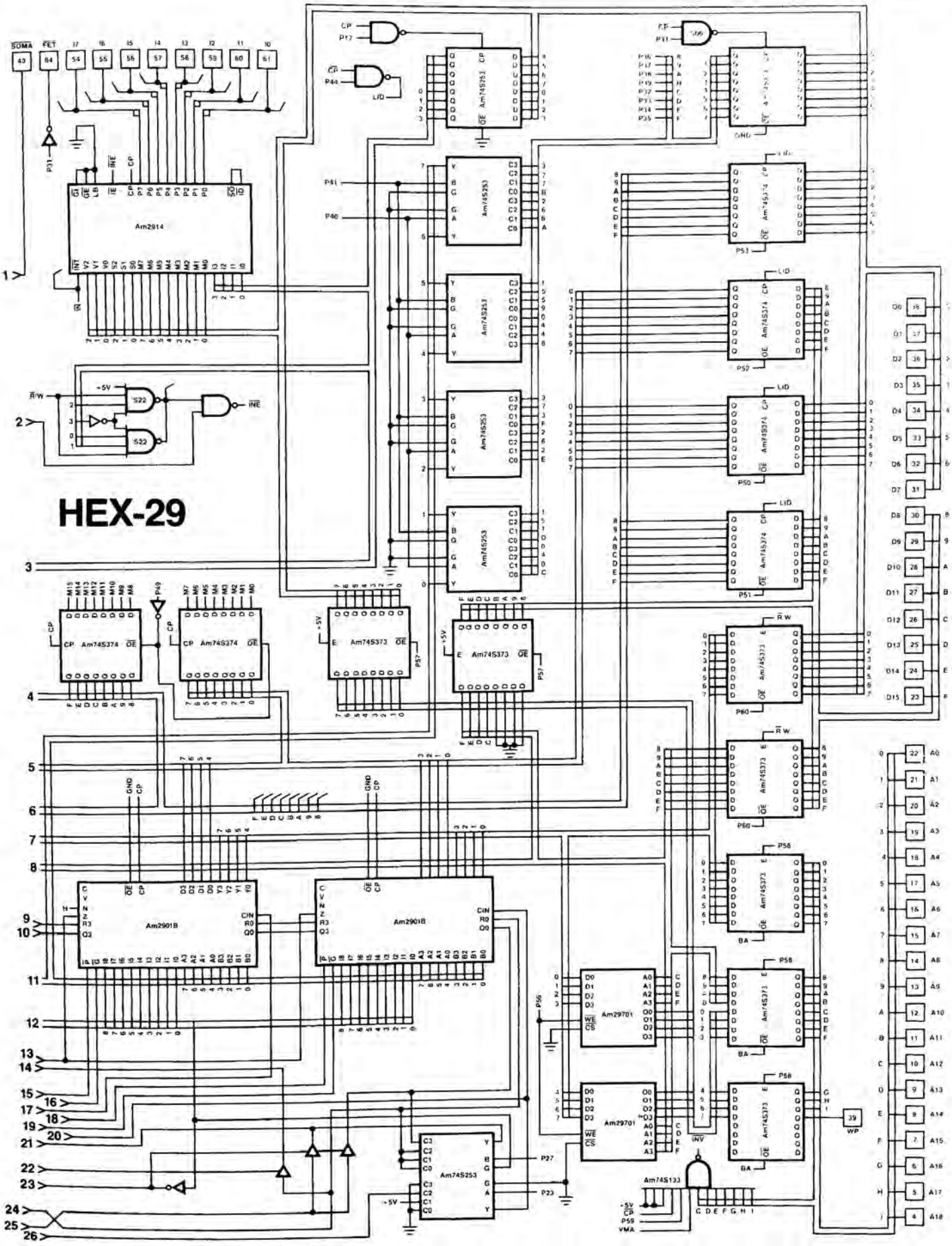

Am74S161
Am74S22 5
Am2909
Am2913
Am74S138
Am26S02
Am74S374
Am27S21
Am27S29
Am74S253
Am2918
Am2901B
Am74S74
Am2902A
OSC
STR
INV
IMA
WAI
RST
DMA
HLT
OFF
+5V
GND

Figure 14a.


HEX-29
Am2914
Am745253
Am74S374
Am74S373
Am2901B
Am29701
Am74S133

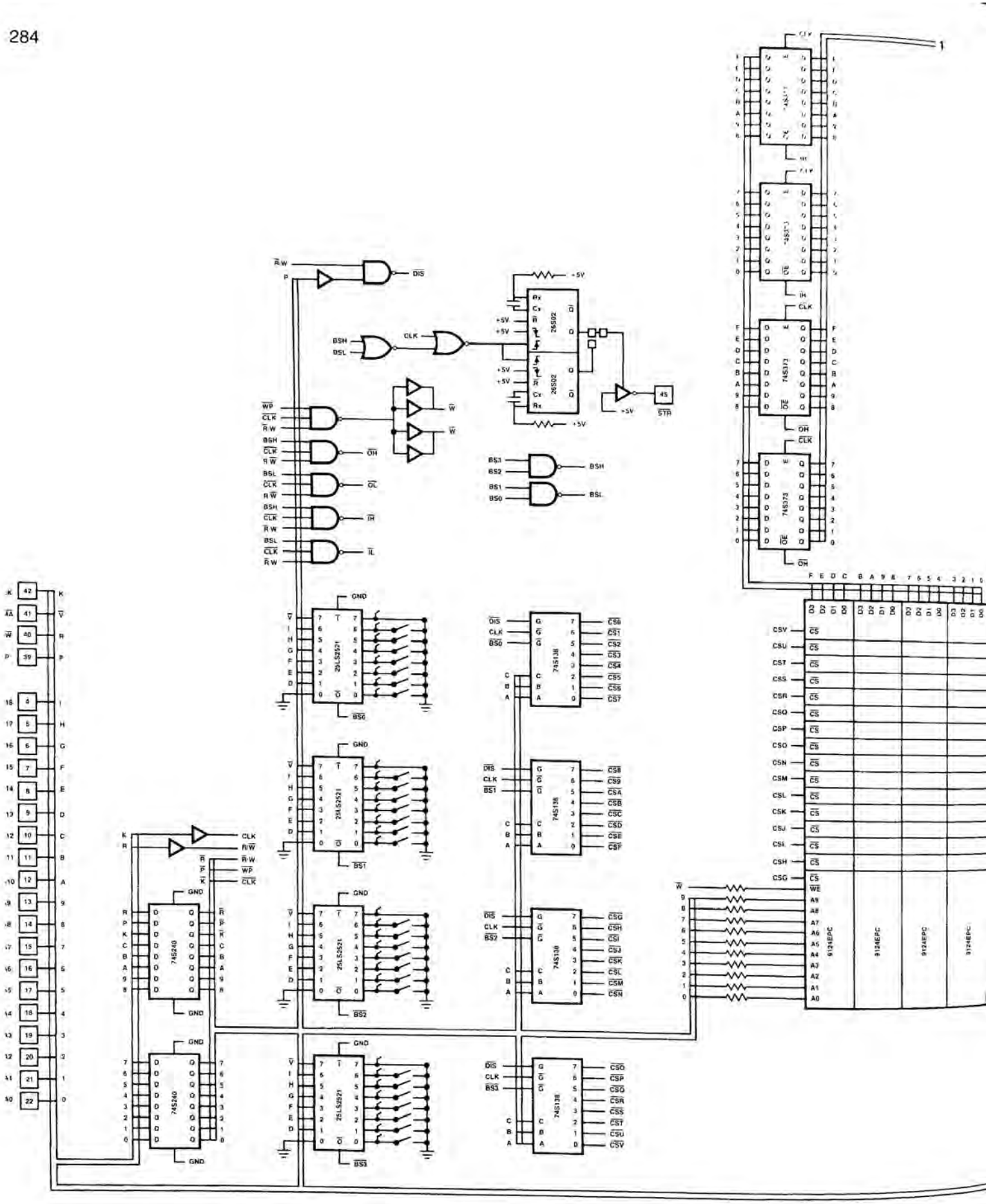
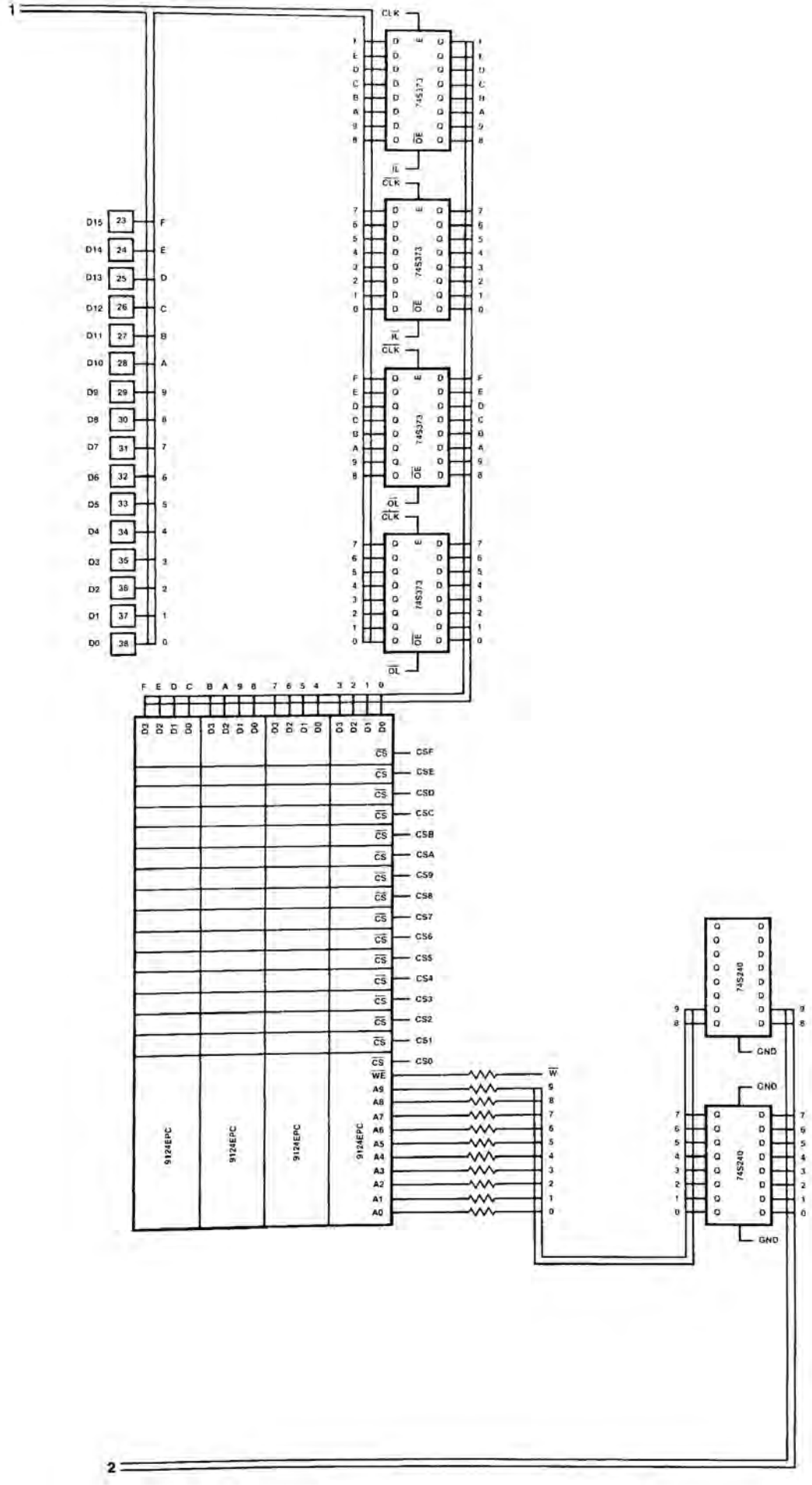
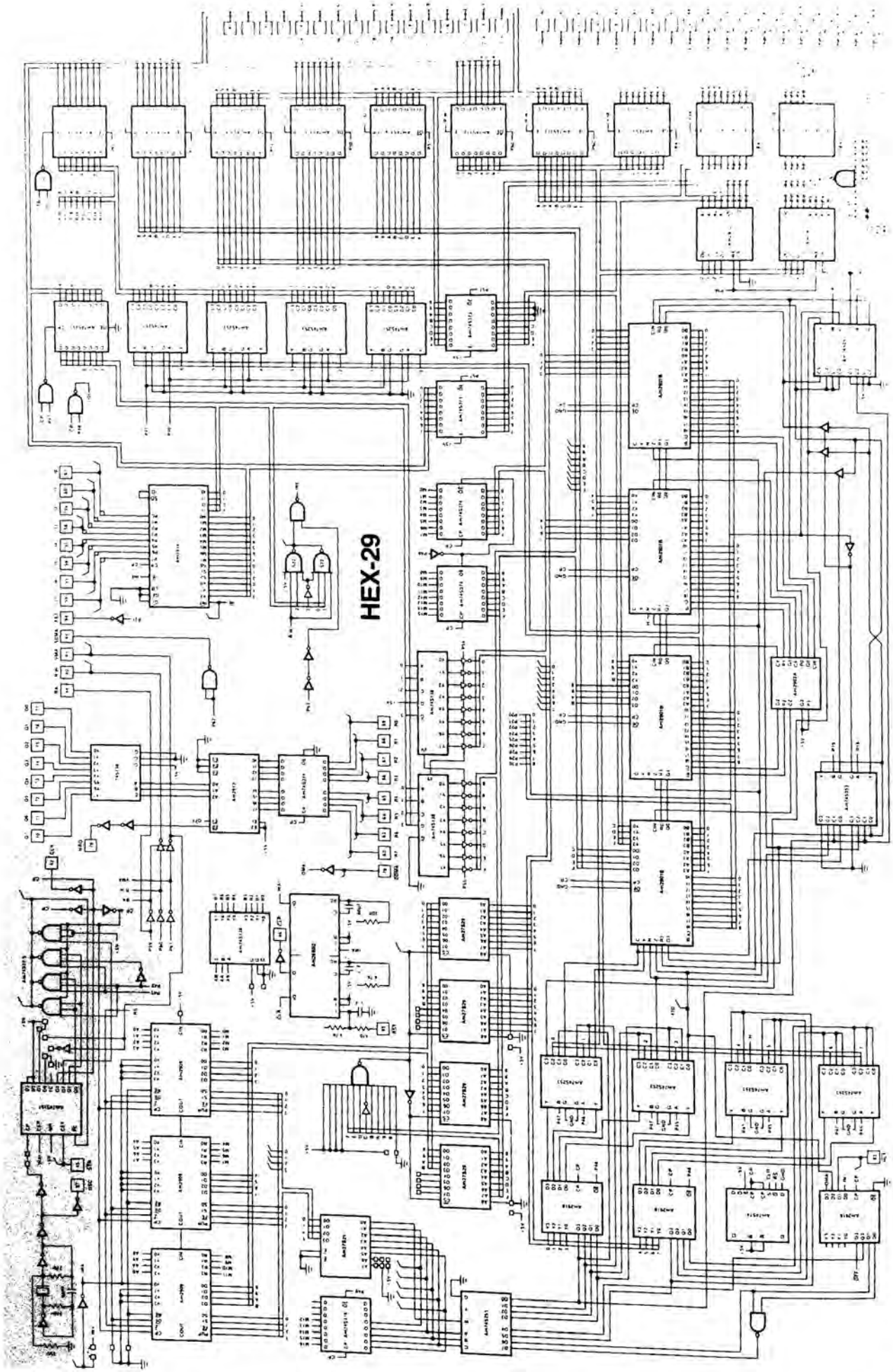


Figure 14b.


1
CLK
D15 23 F
D14 24 E
D13 25 D
D12 26 C
D11 27 B
D10 28 A
D9 29 9
D8 30 8
D7 31 7
D6 32 6
D5 33 5
D4 34 4
D3 35 3
D2 36 2
D1 37 1
D0 38 0
74S373
F E D C B A 9 8 7 6 5 4 3 2 1 0
D3 D2 D1 D0
CS0
CS1
CS2
CS3
CS4
CS5
CS6
CS7
CS8
CS9
CSA
CSB
CSC
CSD
CSE
CSF
WE
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
9124EPC
74S240
GND
2



MPR-666

Figure 15.

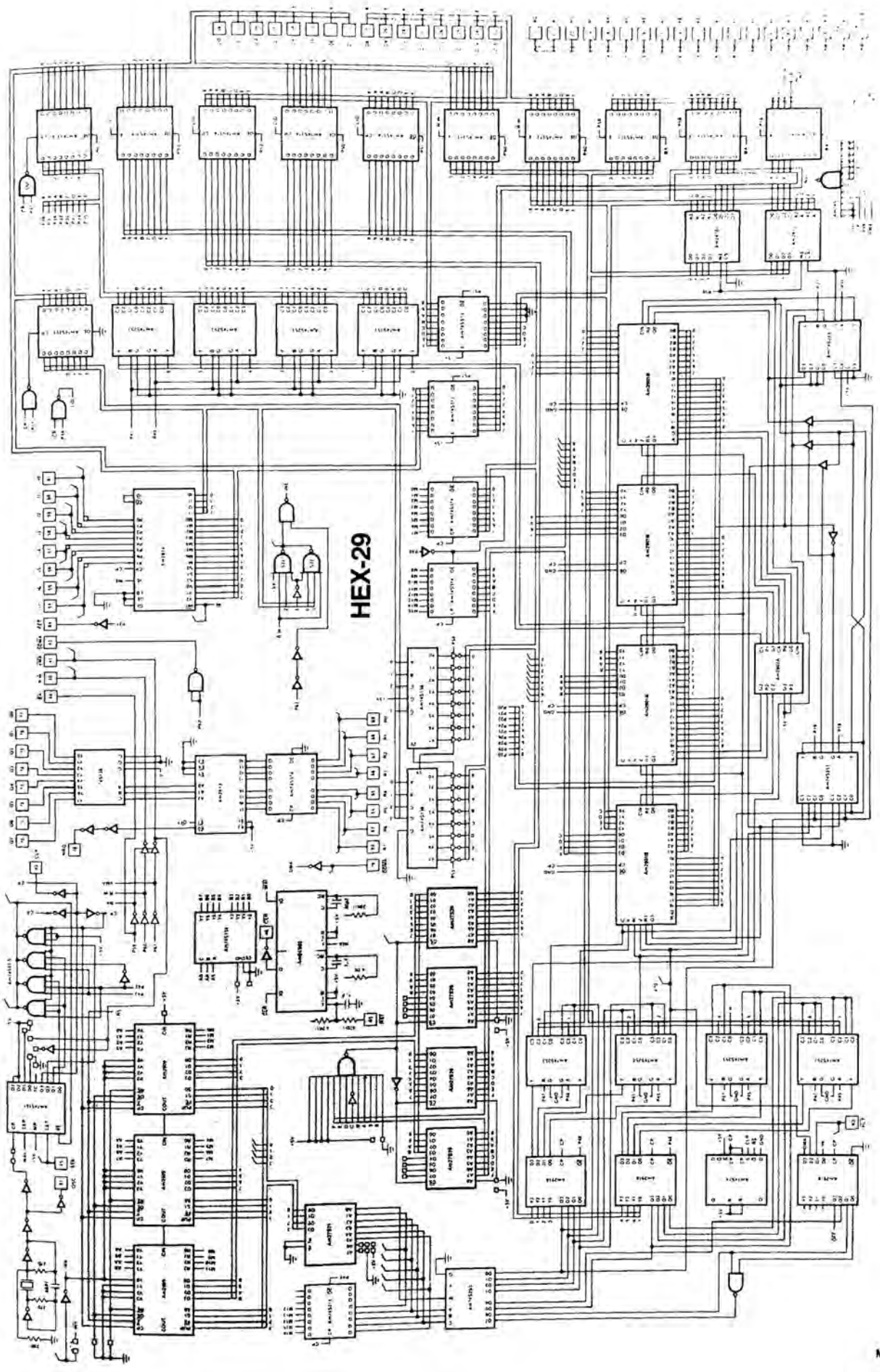


Figure 17.

signal is pulled LOW, the external system bus is released to DMA devices or refresh controllers until the halt bus line is released and the program continues execution.

When an instruction has been fetched and there are no interrupts or halt signals pending, the microprogram must begin executing microinstructions at a new microaddress. This microaddress is a function of the machine instruction to be executed. The "mapping" of the machine level instruction into a microaddress is done courtesy of the Am27S29 instruction decode PROM's. The opcode is placed on the PROM address lines and the microaddress appears at the outputs which are connected to the direct inputs to the Am2909's. The Am2909's simply pass this microaddress to the microword memory by executing a Branch to Address on direct inputs function.

This, and all other microprogram sequencer operations are selected by the outputs of the microprogram branch PROM which is driven by microword bits. This PROM, an AM27S21 contains the output combinations required to execute a variety of microprogram control functions including microbranching, microsubroutining, and two-way microbranching either unconditionally or upon condition code bits selected by microword bits. The function code for this PROM is shown in Figure 18.

As part of the multi-user, multi-task time sharing capabilities, the HEX-29 CPU provides an invalid memory access trap. In this structure, the executive program can assign any unused page of user memory space as either non-existent (transparent) or as an invalid access area. If any user instruction attempts to access memory in a page that has been assigned as an invalid access page, the microprogram control logic takes action.

Before the current machine cycle completes, the next instruction address is forced to the highest value in the current 512-word microword block using the Am2909 OR inputs. At this point a microbranch to the invalid access trap microroutine is performed. The invalid access is processed just like another (highest) level of hardware vectored interrupt except that the current machine level instruction does not complete before the microprogram recognizes and acts upon the condition.

Address	Function
0	BR C = 0 or continue
1	BR C = 1 or continue
2	BR V = 0 or continue
3	BR V = 1 or continue
4	BR N = 0 or continue
5	BR N = 1 or continue
6	BR Z = 0 or continue
7	BR Z = 1 or continue
8	BR H = 0 or continue
9	BR H = 1 or continue
A	BR LZ = 0 or continue
B	BR LZ = 1 or continue
C	BR HLT = 0 or continue
D	BR HLT = 1 or continue
E	BR IH = 0 or continue
F	BR IH = 1 or continue
10	BR
11	Not used
12	CALL
13	Not used
14	CALL N = 0
15	Not used
16	RTS Z = 1
17	Not used
18	RTS
19	Not used
1A	Not used
1B	Not used
1C	Not used
1D	Not used
1E	BRMAP IH = 0 or BR
1F	CONTINUE

Figure 18. Microprogram Sequencer Branch Code.

MICROWORD MEMORY

Any number of memory device types could have been chosen for the microword memory in the HEX-29 CPU. RAM has the advantage that it is dynamically alterable, but if this feature is utilized much more hardware support would have been necessary and the overall cost increased significantly. Besides, the effect of writable control store can be simulated with fixed memory devices by microcode bank switching at much lower cost and complexity if the feature is desirable. For development of new microcode routines, RAM writable control store in the address space of another computer system offers many advantages. This is particularly true if the other computer happens to support a microassembler and file management system as does the System 29.*

Though EROM's and EAROM's are also viable microword memory devices for microcode development, they are much too slow to make efficient use of the rest of the high speed microprogrammed processor in the production device.

Fuse-link bipolar PROM's are the only viable microword memory devices for production systems for a variety of reasons. They are very fast, (45ns maximum access on the HEX-29 CPU), small (512 x 8 in 20 pins), less expensive than fast RAM, and more flexible than a mask ROM would be. It is a simple matter to alter or extend the microprogram of commercial systems with fuse-link PROM microword memory.

As mentioned, the microword memory of the HEX-29 is composed of AM27S29 512 x 8 fuse-link PROM's and is shown in Figure 19. These space efficient 20 pin parts have worst case access times of 45ns over the commercial temperature and voltage range. Up to 4k of microword memory can be addressed by the set of three Am2909 microprogram sequencers on the CPU card. Space for up to 2k of microword memory PROM's is available on the HEX-29 CPU card. Though a perfectly adequate instruction set can be coded in less than 512-words of microword memory, the HEX-29 has a very extensive high level instruction set including 16 and 32-bit integer and 64-bit floating point ADD, SUB, MUL, DIV, CMP, and extensive buffering instructions. In addition to the extremely complete numeric processing package, numerous nibble, character, byte, and word macroinstructions are implemented with scans, linked and unlinked searches, block moves, and etc. A stack processor is a subset of this more than complete instruction set. For all of the capabilities of the HEX-29 CPU, less than 1.5k of microword memory was required. Thus, more than 0.5k of space remains for future expansion by the user before a larger PC card is needed (extremely unlikely).

Connections for the microword data, address and select lines are available at connectors at the top of the HEX-29 CPU card. Thus, it is quite straightforward to support off-board microword memory.

* System 29 is a development system for microprogrammed systems available from Advanced Micro Computers.

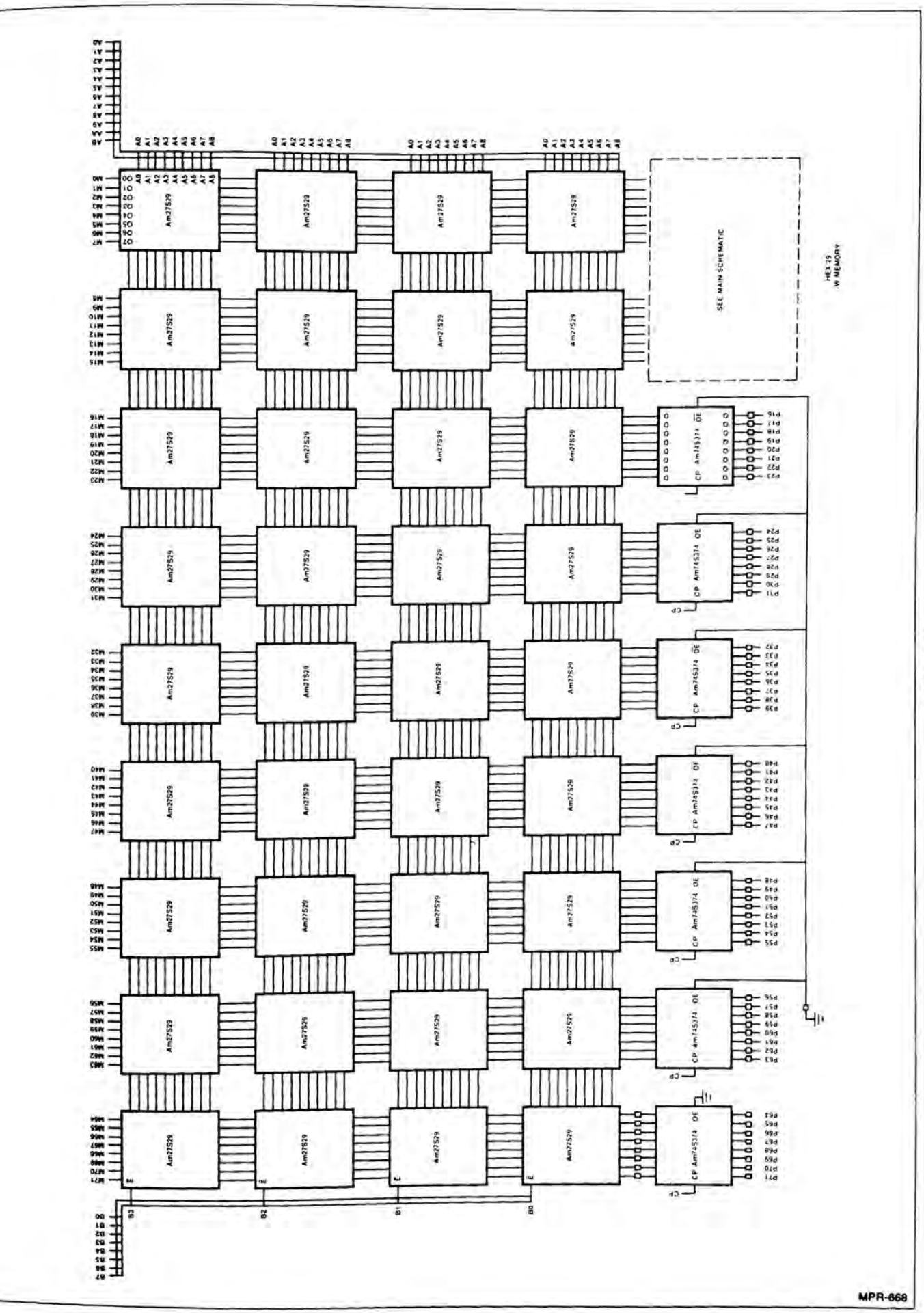


Figure 19.

It is even perfectly reasonable to use an off-board writable control store with up to 2k of microword RAM concurrently with up to 2k of PROM resident on the PC card.

If the on board PROM contains an instruction set, it is then a simple matter to use the off board writable control store to develop new microcode for the machine interactively on the one HEX-29 system!

The outputs of the microword memory devices are attached to the inputs of Am74S374 registers. These registers are called the pipeline registers since they allow the fetching of the next micro-instruction concurrently with execution of the current one. Clocking of the pipeline registers occurs on the LOW-to-HIGH transition of the system clock. The outputs of the pipeline registers are the 64 microword (or pipeline) bits that control every aspect of the processor.

These 64 bits can be logically grouped into several functional fields a follows:

1. Microword Data/Microbranch Address and Control
2. ALU Source Select
3. ALU Destination Select
4. ALU Function Select
5. ALU Carry In Select
6. Shift Linkage Select
7. ALU A and B Specifications
8. A and B Fields Select
9. Enable onto ALU inputs Select
10. Latch External Data Inputs
11. Latch CPU Outputs
12. Control Bus Signals
13. Microprogrammed Clock
14. Condition Code Controls
15. Enable Interrupt Circuitry
16. Memory Map Control

Notice that with the exception of the microword data and microbranch address and control fields, no other fields are overlapped. This is a 'horizontally' structured design. Overlapping several fields leads to 'vertically' structured systems. This latter class of machines can save some microword memory, but only at the expense of through-put and increased hardware complexity. Now that the cost of the PROM's has come down significantly, the savings accrued from using a vertically structured design approach is generally insignificant when compared with the overall system cost.

A summary of the functions of the microword bits is shown in Figure 20.

Am2901B ALU/REGISTER SETS

The heart of the HEX-29 CPU is the set of four Am2901B bit slice ALU/Register Sets depicted in Figure 21. All arithmetic and logical operations are performed in these bipolar LSI IC's, including address generation. The user accessible set of 16 registers and routing functions are also internal to these remarkable and extremely versatile chips.

The operation of these units, though very elegant and comprehendible, is too lengthy to include here and the user is referred to the *Am2900 Family Data Book* by AMD.

Carry lookahead is accomplished by the Am2901B's and an external IC, the Am2902A. Shift control is partially within the Am2901B's and is supported by other external circuitry to be discussed later.

A summary sheet of the Am2901B ALU functions appears on page 29 but should be supplemented by studying the AMD literature already mentioned. A good supplement is the AMD *Schottky and Low Power Schottky Handbook*

The A and B input fields to the Am2901B's are multiplexed by 4 Am74S253's in the following four ways.

Am2901B B Inputs	Am2901B A Inputs
μword Memory	μword Memory
Upper Nibble ABL	Upper Nibble ABL
Lower Nibble ABL	Lower Nibble ABL
Upper Nibble ABL	Lower Nibble ABL

ABL = A,B Latch (On data bus bits 27-20.)

CARRY IN CONTROL

The arithmetic carry-in (C_N) signal (Figure 22) to the Am2901B bit slices can be selected from four sources as follows:

1. Logic 0 (No carry-in add instruction, borrow in subtract instruction.)
2. Logic 1 (Carry-in in add instruction, no borrow in subtract.)
3. Carry Flag (C bit in condition code register.)
4. Q Shift Bit (Double length shifts.)

Note that the natural state of the Carry Flag output from the Am2901B is 1 for carry on add, 0 for no carry on add, 1 for no borrow on subtract, and 0 for borrow on subtract. This convention has been maintained in the condition code and carry in logic. Some other machines operate differently with respect to this convention, but others do not and the HEX-29 maintains the faster convention for lack of a good reason to alter it. Some programmers will be required to remember this convention while others will be used to it.

SHIFT AND ROTATE LINKAGE

The shift and rotate linkage (Figure 23a) of the HEX-29 is composed of an Am74S253 and an Am74LS125 plus the internal shift control structure of the Am2901B's. The functions that can be performed by this circuitry are shown in Figure 23b.

The solid lines in Figure 23b delineate the basic shift linkages. The dotted lines are optional linkages which can also be enabled. With these linkages, all of the normal shifts and rotates can be performed plus a number of double word shifts including special shifts for high speed multiplies and divides.

CONDITION CODE CONTROL

The condition code register shown in Figure 24 of the HEX-29 has eight flags. The definitions and placement of these flags are defined in Figure 25.

In addition to the very useful and fairly common C, V, N, Z flags, a half sign is provided for easier byte processing. The three user flags are not changed by any of the normal arithmetic or logical operations. However, they can be read by the processor and written by the processor with special instructions such as load flags, read flags, set bits in flags, clear bits in flags, invert bits in flags. The fact that none of the user flags is changed by any but this type of special routine is very significant. It means that various routines and program segments can pass flags back and forth freely without fear of modification or restriction on the instructions that can be executed. Reading the condition code flags into the processor, or branching or subroutining upon combinations of bits set or clear does not alter the flags.

Name		Function	
0. μD0	BRA0	Microprogram Branch Address	Microword data to Internal data bus to Am2901B's
1. μD1	BRA1		
2. μD2	BRA2		
3. μD3	BRA3		
4. μD4	BRA4		
5. μD5	BRA5		
6. μD6	BRA6		
7. μD7	BRA7		
8. μD8	BRA8		
9. μD9	BRA9		
10. μDA	BRAA		
11. μDB	BRAB		
12. μDC	BRC0	Microprogram Branch Control	
13. μDD	BRC1		
14. μDE	BRC2		
15. μDF	BRC3		
16.	BRC4		
17. LIN		Latch in low nibble of data bus	
18. ROT0		Control Bits Shift and Rotate MUX	A
19. ROT1			B
20. SRC0		Am2901 Source Select Code	
21. SRC1			
22. SRC2			
23. CIA		Carry-In MUX Select Bit A	
24. ALU0		Am2901 ALU Function Code	
25. ALU1			
26. ALU2			
27. CIB		Carry-In MUX Select Bit B	
28. DST0		Am2901 Destination Code	
29. DST1			
30. DST2			
31. FET		Fetch Instruction this cycle	
32. B0		Am2901 'B' field register specification	
33. B1			
34. B2			
35. B3			
36. A0		Am2901 'A' field register specification	
37. A1			
38. A2			
39. A3			
40. ABM0		A, B fields MUX Select Bit 0 (A, B fields on Am2901)	
41. ABM1		A, B fields MUX Select Bit 1 (A, B fields on Am2901)	
42. ST0		Microprogrammed system clock stretch bit 0	
43. ST1		Microprogrammed system clock stretch bit 1	
44. LDI		Latch Data In – Both Swapped and Unswapped	
45. LNZ		Latch N, Z, H flags – MUX Select	
46. LCV		Latch C, V flags – MUX Select	
47. LCC		Latch Condition Codes – U2, U1, U0, H, Z, N, V, C MUX Select	
48. $\overline{\text{RCC}}$		Read Condition Codes onto internal bus, low byte	
49. SDA		Select Microword bits 15-0 to internal bus, else branch code	
50. $\overline{\text{DIL}}$		Data In Low Byte Enable onto internal bus	
51. $\overline{\text{DIH}}$		Data In High Byte Enable onto internal bus	
52. $\overline{\text{SWPL}}$		Swapped Data In Low Byte Enabled onto internal bus	
53. $\overline{\text{SWPH}}$		Swapped Data In High Byte Enabled onto internal bus	
54. $\overline{\text{CLL}}$		Clear Low Byte on internal bus – Bit Set Enable HIGH	
55. $\overline{\text{CLH}}$		Clear High Byte on internal bus – Bit Set Enable LOW	
56. $\overline{\text{LMM}}$		Load Memory Map – Write into Memory Map RAM	
57. $\overline{\text{RMM}}$		Read Memory Map – Enable Memory Map to data bus	
58. LAD		Latch Address – Enable Transparent Address Latch	
59. $\overline{\text{BA}}$		Bus Available – Busses available for DMA this cycle	
60. R/$\overline{\text{W}}$		REad/Write Memory (Write if low)	
61. $\overline{\text{VMA}}$		Valid Memory Address (Read or Write) this cycle	
62. SDMA		Sync. DMA – Active cycle before bus is available	
63. INE		Interrupt Logic Enable	

Figure 20A. Microword Bits.

0	BR C = 0
1	BR C = 1
2	BR V = 0
3	BR V = 1
4	BR N = 0
5	BR N = 1
6	BR Z = 0
7	BR Z = 1
8	BR HS = 0
9	BR HS = 1
A	BR LZ = 0
B	BR LZ = 1
C	BR HLT = 0
D	BR HLT = 1
E	BR IH = 0
F	BR IH = 1

10	BR
11	
12	BSR
13	
14	BSR N = 0
15	
16	RTS Z = 1
17	
18	RTS
19	
1A	
1B	
1C	
1D	
1E	BR IH = 1 or MAP
1F	CONTINUE

Figure 20B. Am2909 Microprogram Branch Control, Bits 12-16.

	LCC	LCU	LNZ	
LCN	0	0	0	New CVNZH
LC	0	0	1	New CV Old NZH
LN	0	1	0	Old CV New NZH
(Nom.)	0	1	1	Old CVNZH
BCC	1	0	0	Bus → CVNZHV
	1	0	1	Bus → CV Old NZH
	1	1	0	Shift Old V Bus → NZHU
LSC	1	1	1	Shift C Old V Old NZH

Figure 20C. Condition Code Manipulation, Bits 45-47.

ALU	CIB	CIA	0	1	2	3	4	5	6	7
0	0	0	A + Q	A + B	Q	B	A	D + A	D + Q	D
	0	1	A + Q + 1	A + B + 1	Q + 1	B + 1	A + 1	D + A + 1	D + Q + 1	D + 1
	1	0	A + Q + C	A + B + C	Q + C	B + C	A + C	D + A + C	D + Q + C	D + C
1	0	0	Q − A − 1	B − A − 1	Q − 1	B − 1	A − 1	A − D − 1	Q − D − 1	− D − 1
	0	1	Q − A	B − A	Q	B	A	A − D	Q − D	− D
	1	0	Q − A − C	B − A − C	Q − C	B − C	A − C	A − D − C	Q − D − C	− D − C
2	0	0	A − Q − 1	A − B − 1	− Q − 1	− B − 1	− A − 1	D − A − 1	D − Q − 1	D − 1
	0	1	A − Q	A − B	− Q	− B	− A	D − A	D − Q	D
	1	0	A − Q − C	A − B − C	− Q − C	− B − C	− A − C	D − A − C	D − Q − C	D − C
3	–	–	A∨Q	A∨B	Q	B	A	D∨A	D∨Q	D
4	–	–	A∧Q	A∧B	0	0	0	D∧A	D∧Q	0
5	–	–	A∧Q	A∧B	Q	B	A	D∧A	D∧Q	0
6	–	–	A∀Q	A∀B	Q	B	A	D∀A	D∀Q	D
7	–	–	A∀Q	A∀B	Q	B	A	D∀A	D∀Q	D

Figure 20D. Am2901 Source, Carry-in & Function Select, Bits 20-27.

DST	Rotates
0	F → Q
1	NONE
2	F → B A → Y
3	F → B
4	RIGHT F/2 → B Q/2 → Q
5	RIGHT F/2 → B
6	LEFT 2F → B 2Q → Q
7	LEFT 2F → B

Figure 20E. Am2901 Destination Codes, Bits 28-30.

ABMUX	A reg	B reg
0	μW_A	μW_B
1	R_S	R_D
2	R_S	R_S
3	R_D	R_D

Figure 20F. Am2901 A, B Field Selects, Bits 40-41.

	Right	Left
0	MUL	RCL
1	ROR	ROL
2	ASR	DRL
3	LSR	LSL

Figure 20G. Shift & Rotate Control, Bits 18-19.

STR	CLOCK
0	280ns
1	240ns
2	200ns
3	160ns

Figure 20H. Microprogrammed System Clock Stretch, Bits 42-43.

Figure 20 (Cont.).

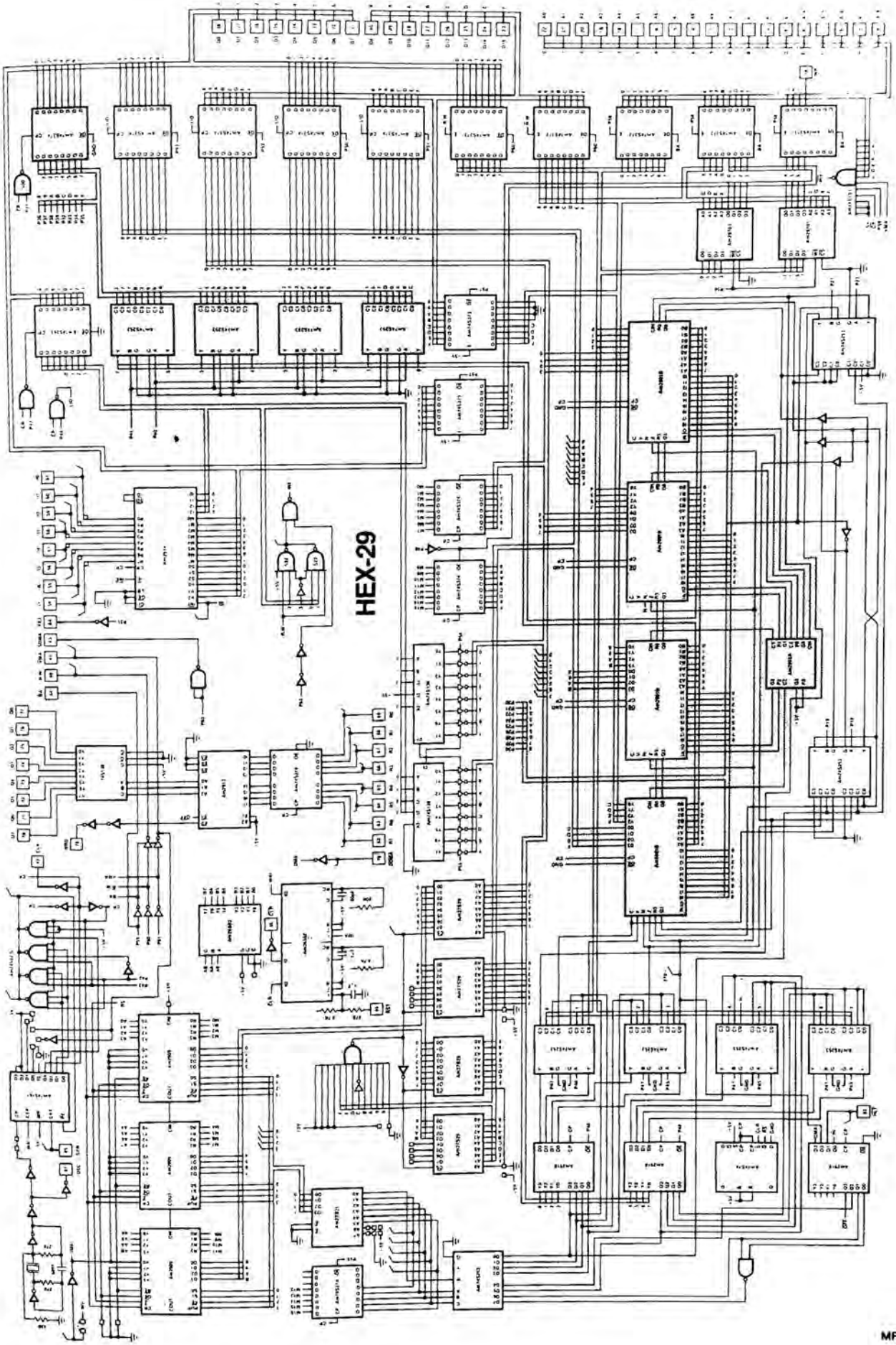


Figure 21.

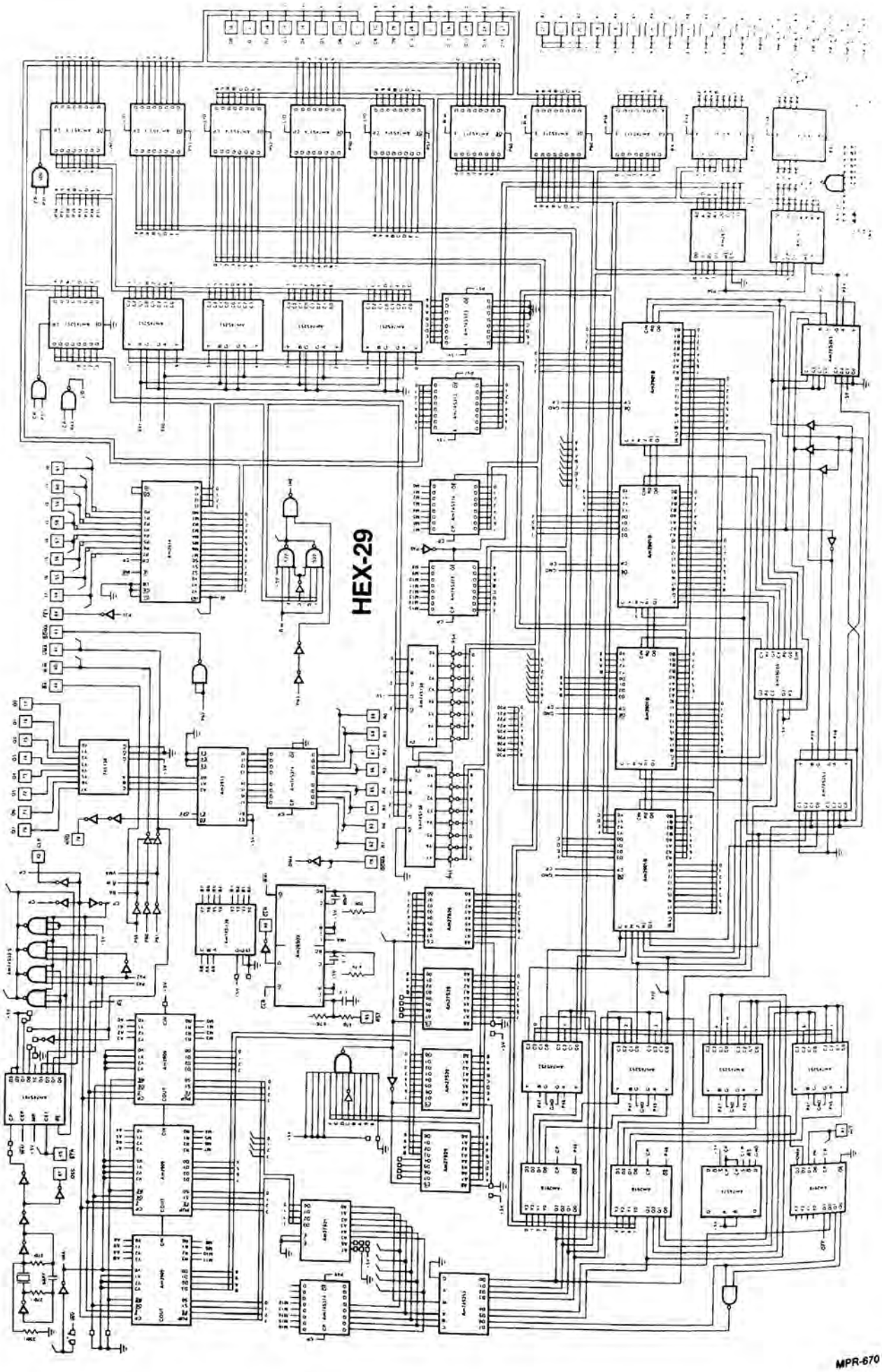


Figure 22.

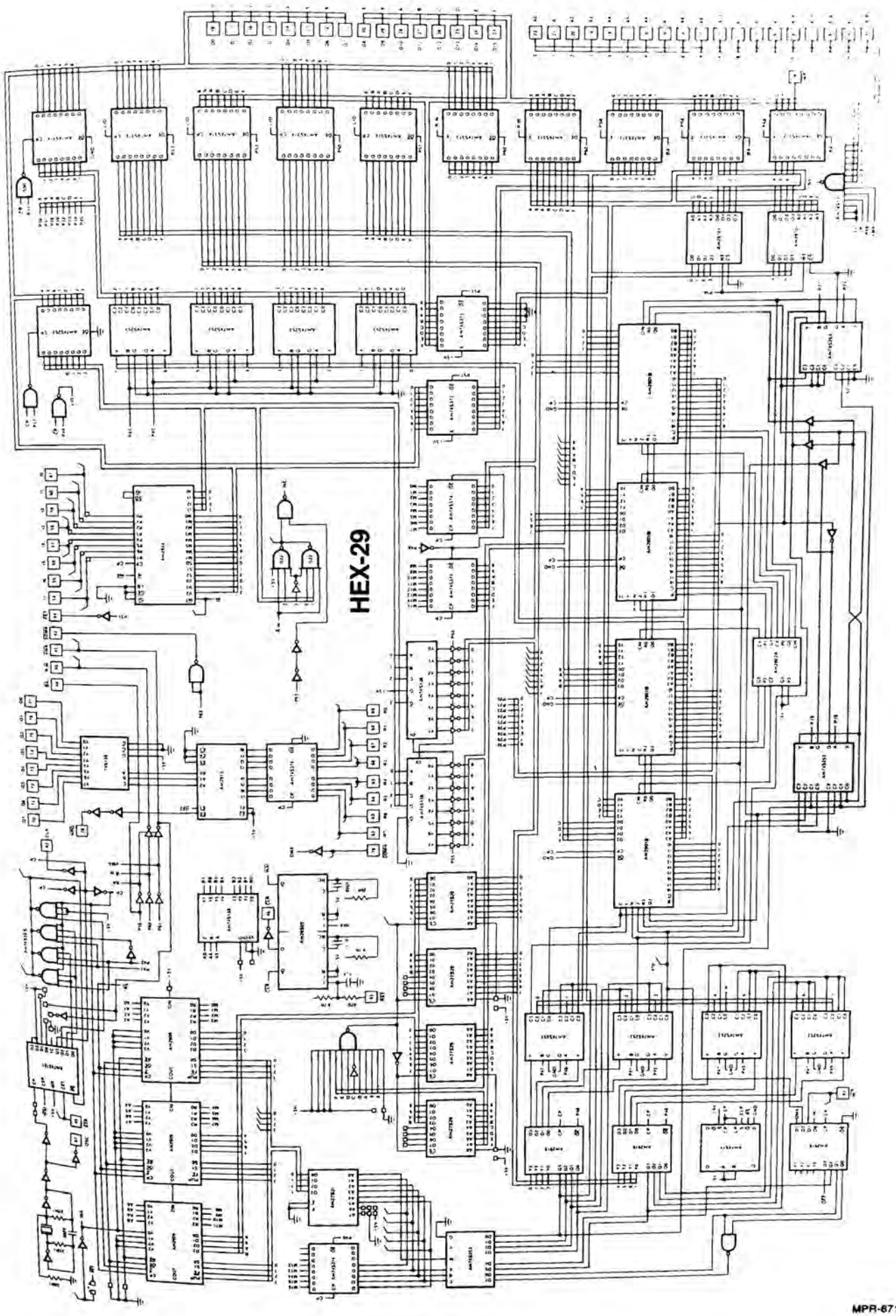


Figure 23A.

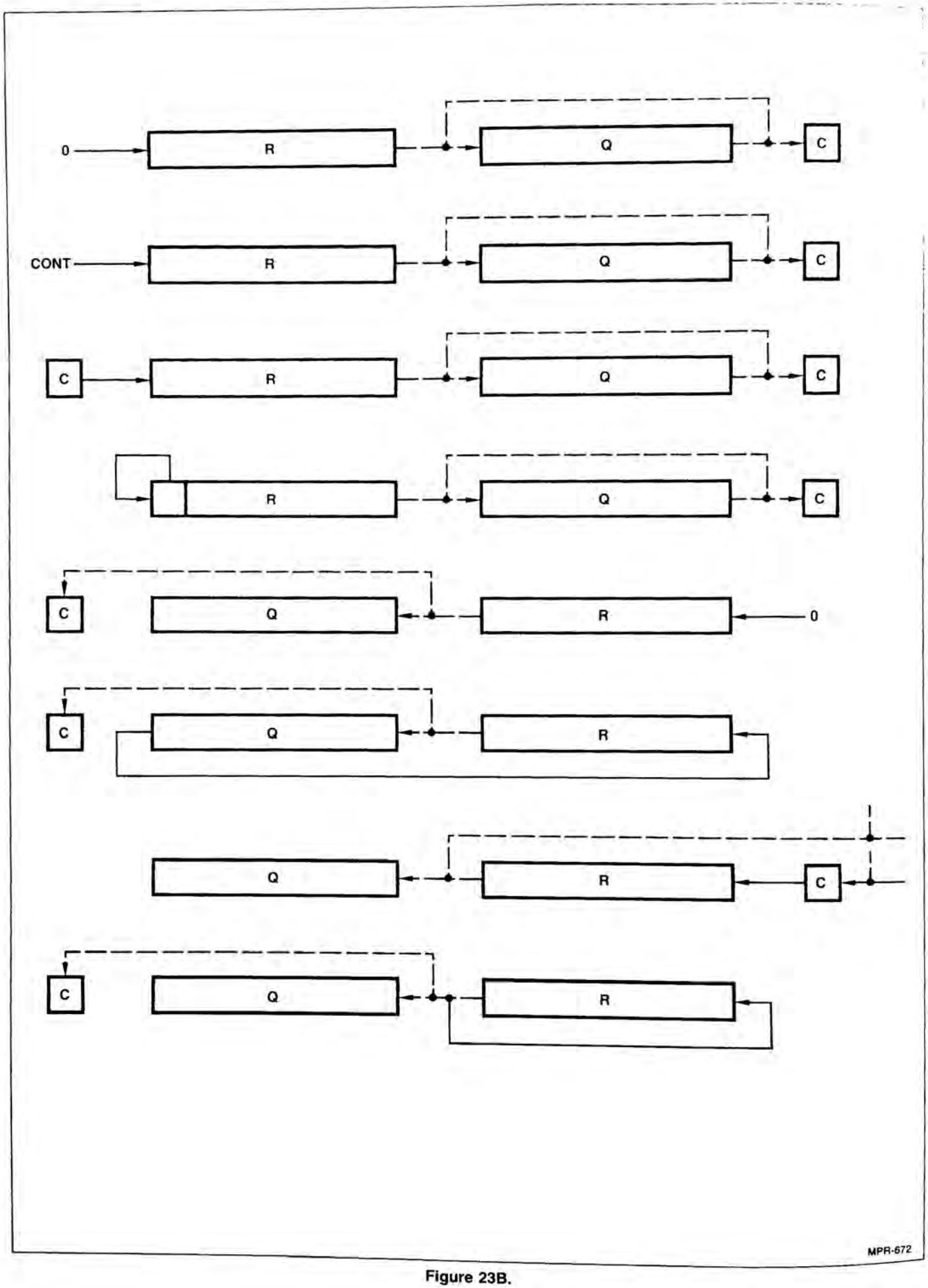


Figure 23B.

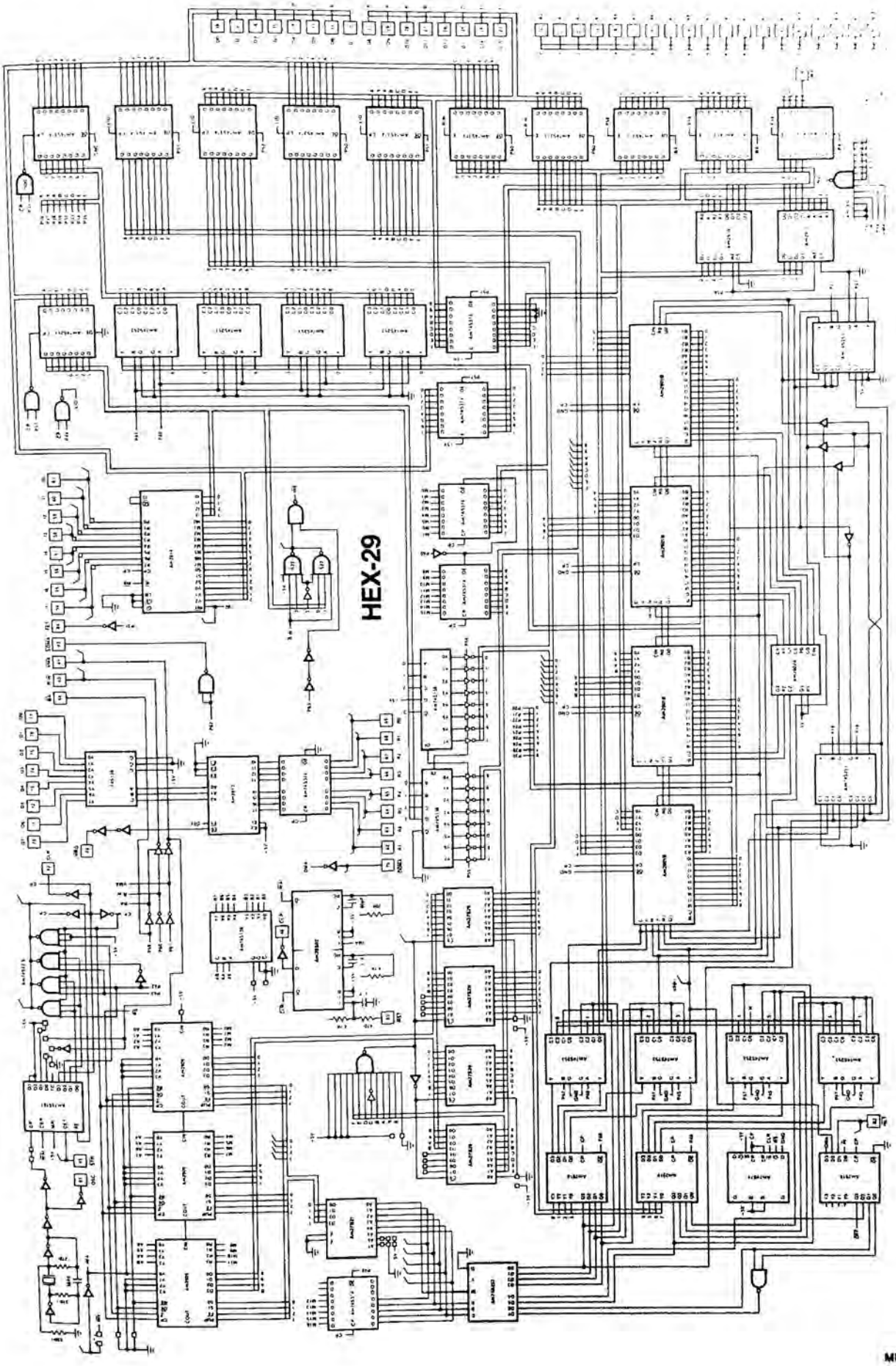


Figure 24.

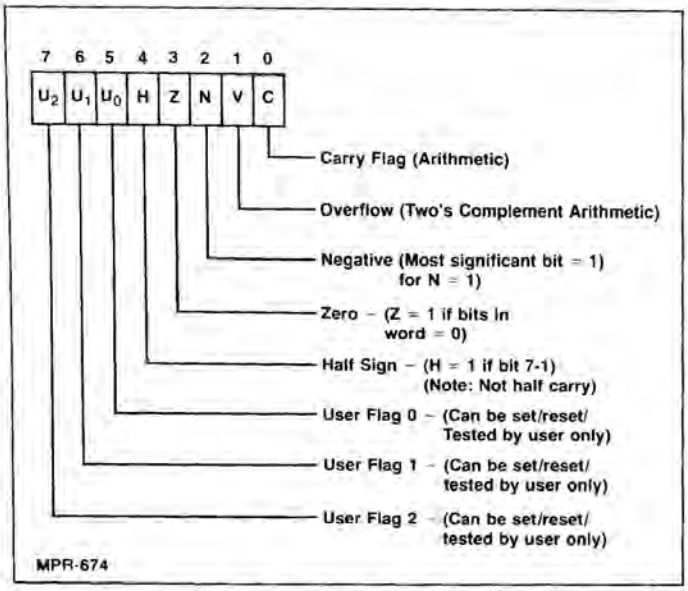


Figure 25.

Eight condition code operations provide all the useful operations needed for complete flexibility. They are shown in Figure 26a and 26b in two different formats. Note that the codes are grouped into three categories; arithmetic (C and V), logical/arithmetic (N, Z, H) and user (U_2, U_1, U_0).

These eight conditions include all the necessary and desirable features such as updating only the shift carry bit and the ability to do operations that read, operate on, and reload the condition code register all in one machine cycle (160ns). Also, a feature of immense importance where microcoded floating point or fixed point math is concerned is the ability to update flags on a cycle by cycle basis! An unusual feature.

	Carry/ Overflow C, V	Negative/Zero/ Half N, Z, H	User Flags U2, U1, U0
7	Shift Bit C,V V	No Change	No Change
*6	Shift Bit C,V V	Load From Bus	Load From Bus
*5	Load From Bus	No Change	No Change
4	Load From Bus	Load From Bus	Load From Bus
3	No Change	No Change	No Change
2	No Change	Update	No Change
1	Update	No Change	No Change
0	Update	Update	No Change

*Less useful than other codes but perfectly legal.

Figure 26A.

Name	U2	U1	U0	H	Z	N	V	C
Shift MSb or LSb into C	NC	NC	NC	NC	NC	NC	NC	S
*Shift into C - Bus Load Rest	B	B	B	B	B	B	NC	S
*Bus Load C & V Flags	NC	NC	NC	NC	NC	NC	B	B
Bus Load All Flags	B	B	B	B	B	B	B	B
No Changes	NC	NC	NC	NC	NC	NC	NC	NC
Update N, Z, H Flags	NC	NC	NC	μ	μ	μ	NC	NC
Update C and V Flags	NC	NC	NC	NC	NC	NC	μ	μ
Update C, V, N, Z, H	NC	NC	NC	μ	μ	μ	μ	μ

μ = updated, NC = unchanged, B = loaded from internal bus, S = Shift Bit

Figure 26B.

Am2901B OUTPUT BUS

Being a highly structured, modular device, the HEX-29 CPU is very bus oriented. The output bus of the Am2901B's generate the addresses and data to the rest of the system devices as well as some internal function. The four logical units on this bus (shown in Figure 27) are:

1. Address Out Latches - (System Address bus)
2. Data Out Latches - (System data bus)
3. Memory Map/Latches - (Memory Management Features)
4. Condition Code MUX - (For updating flags from processor)

Any memory reference requires that an address be valid on the system address bus. The source of this address is generally one of the Am2901B internal registers or modifications thereof from previous fetch cycles (such as indexed addressing).

On a write cycle, data must be placed on the system data bus. This is accomplished in the same manner as address generation except that a different microword bit is used to activate the data latches.

In a multi-user/multi-task/timesharing environment, it is desirable to have a powerful memory management scheme. The HEX-29 CPU implements this via a flexible memory mapping system where the upper four bits of the 16-bit address generated by the Am2901B's are 'mapped' into seven address bits and a write protect bit. Invalid access traps and one Megabyte address space are integral features of this system. The loading of this MAP RAM (2 Am29701's) is also accomplished via the Am2901B output bus.

Another important characteristic of the HEX-29 CPU is its ability to read, write, test and operate upon the eight condition code flags in the byte form. All eight flags can be written to by the Am2901B's at one time, in one microcycle. This is very useful for many flag operations and is absolutely necessary for efficient updating of the user flags for interroutine parameter and condition passing.

The logic of these bussed systems is quite simple. A separate microword bit or bit field is used to cause each of these logical units on the bus to accept the data bus. Therefore, simple micro-programming techniques are applicable to this busing approach.

Am2901B INPUT BUS

Much of the power and modularity of the HEX-29 design is due to the highly structured bus approach on the Am2901B Data Inputs. The logical units that can drive this bus (Figure 28) are listed below:

1. Data Input Registers
2. Swap Input Registers
3. Microword Data Registers
4. Clear Upper Byte/Clear Lower Byte/Bit Op Logic
5. Condition Code Register

Data input from the system bus is captured in the data input registers and the swap input registers. The data input registers bring the upper and lower bytes of the data bus to the corresponding bytes in the Am2901B cascade while the swap input registers switch the upper byte of the data bus to the lower byte on the Am2901B cascade and the lower byte of the data bus to the upper byte of the Am2901B cascade.

Additionally, logic to set all bits in the upper or lower byte to zeros, (clear upper byte and clear lower byte), allow selecting arithmetic or logical zeros in either byte field. If the bit set option is enabled, all bits are pulled low except the one selected by the hexadecimal value in the low nibble of the nibble latch from an instruction or other data source.

All eight condition code bits can be enabled onto the low byte if desired. All flags can thus be sampled by the Am2901B's at once.

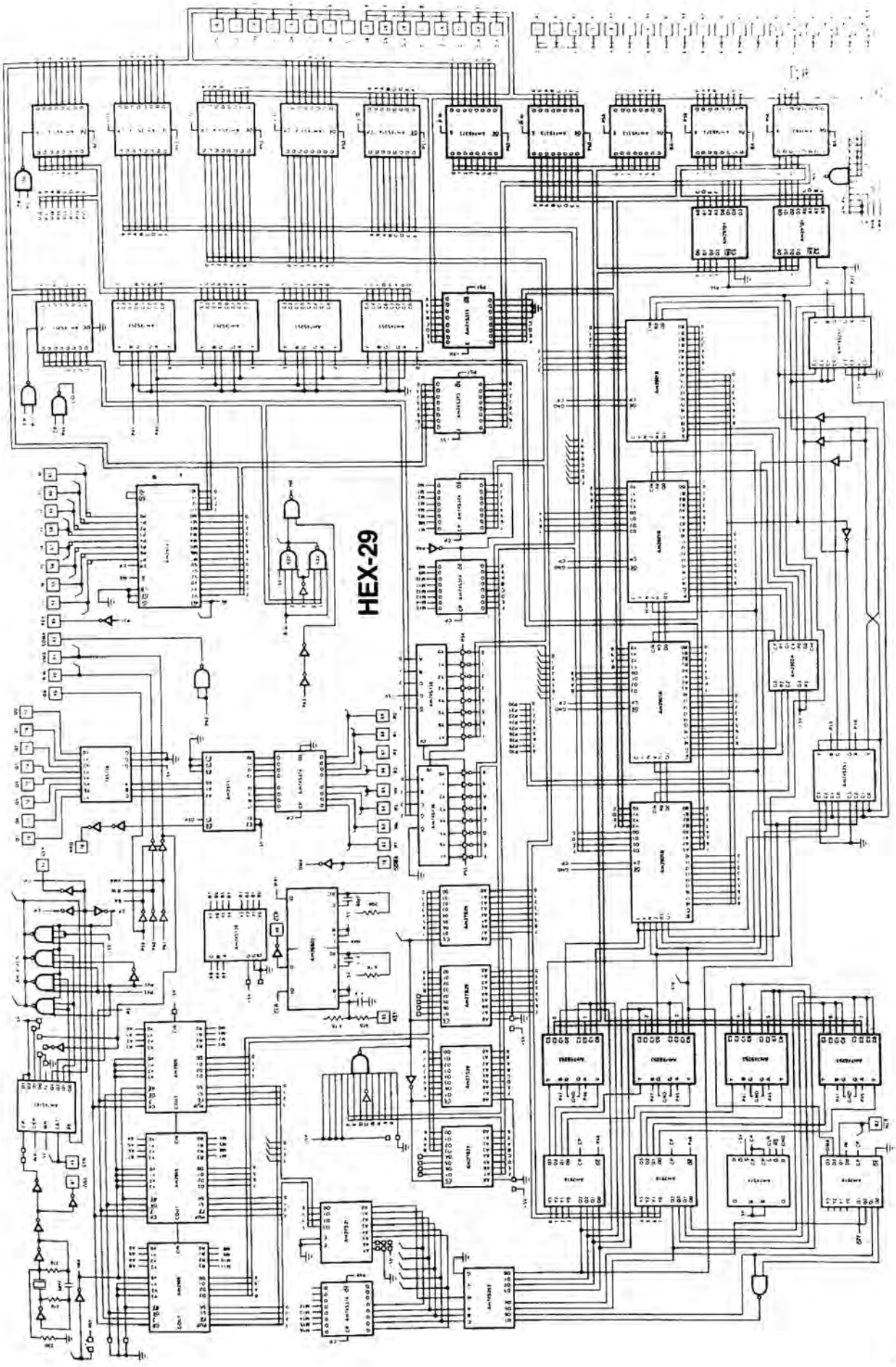


Figure 27.

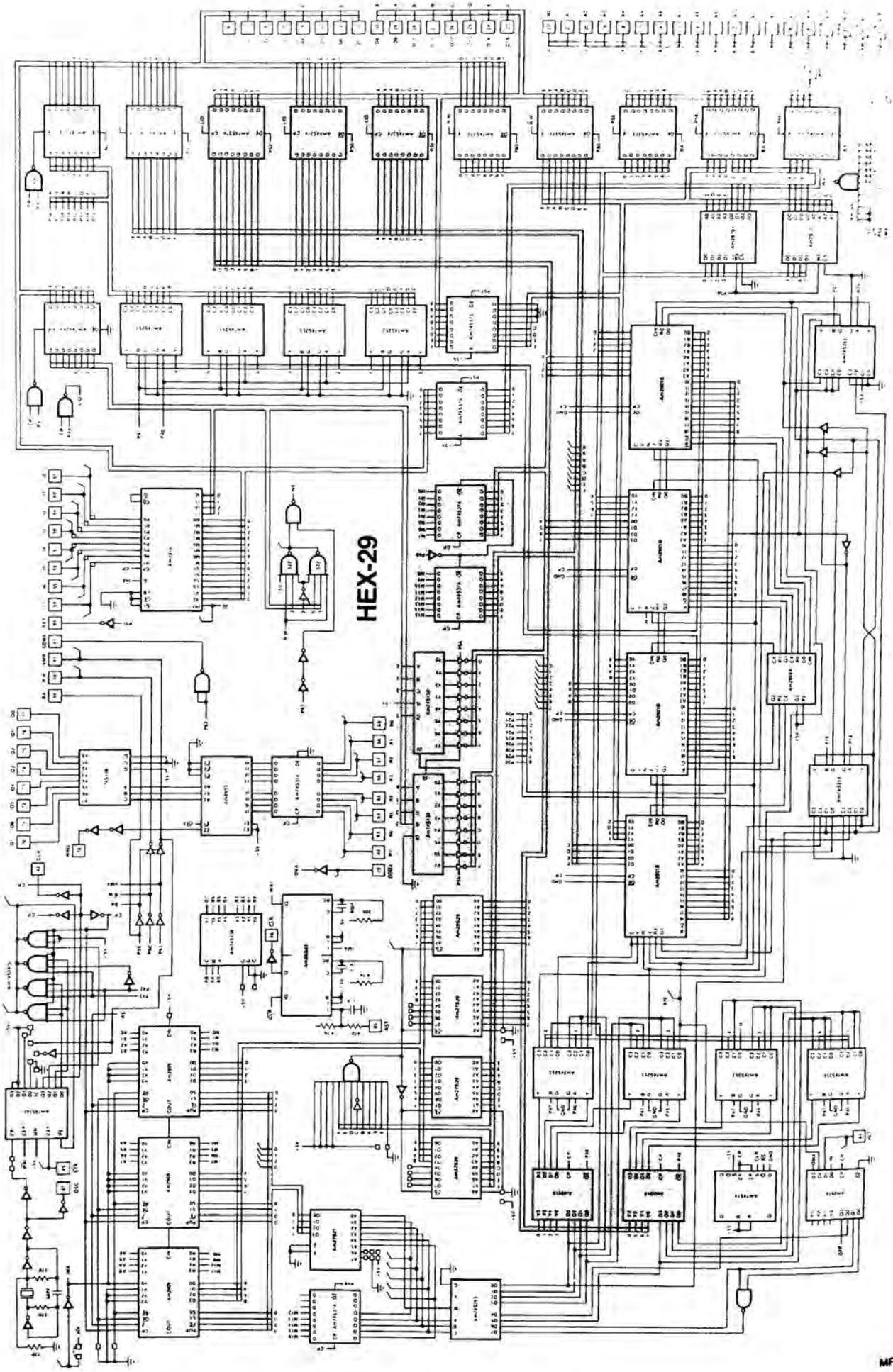


Figure 28.

Data from microword memory from three-state registers in parallel with the pipeline register can be enabled onto the upper and lower bytes for direct loading of the Am2901B's from microprograms.

In the absence of any device being enabled onto a particular byte on this bus, it will be pulled up into a logic 1 state. This can be useful for masking in logical operations and filling or biasing in arithmetic operations.

An important factor in the flexibility of this approach on the HEX-29 is that the upper and lower bytes of the data in registers, swap in registers, and the clear upper/lower byte logic are separately enabled. Also, the condition code register only drives the lower byte and the pull-up feature will operate on either byte individually. Thus the upper and lower bytes can be individually driven on a 'mix and match' basis from several sources.

The versatility so generated allows numerous fast processing modes. See Table 5 for a list of all of the possible combinations of high and low byte inputs to the 2901B's.

TABLE 5.

Into Upper Byte of Am2901's	Into Lower Byte of Am2901's
0. Microword memory bits P15-P8	Microword memory bits P7-P0
1. Bit set value (upper byte)	Bit set value (lower byte)
2. Upper byte – data bus	Lower byte – data bus
3. Upper byte – data bus	Upper byte – data bus
4. Upper byte – data bus	Clear lower byte
5. Upper byte – data bus	All high generator
6. Upper byte – data bus	Condition code register
7. Lower byte – data bus	Lower byte – data bus
8. Lower byte – data bus	Upper byte – data bus
9. Lower byte – data bus	Clear lower byte
10. Lower byte – data bus	All high generator
11. Lower byte – data bus	Condition code register
12. All high generator	Lower byte – data bus
13. All high generator	Upper byte – data bus
14. All high generator	Clear lower byte
15. All high generator	All high generator
16. All high generator	Condition code register
17. Clear upper byte	Lower byte – data bus
18. Clear upper byte	Upper byte – data bus
19. Clear upper byte	All high generator
20. Clear upper byte	Condition code register
*21. Clear upper byte	Clear lower byte

*Note: Interestingly enough this is the only case in the entire table that the hardware CANNOT generate on the bus, but IS the ONLY one of these codes that CAN be generated by the AMD 2901B slices! (How convenient!)

Examples of uses for some of these modes include:

1. Clearing upper byte for 8-bit index offset
2. Fast bit set/clear/test/invert operations
3. Set upper byte high to AND lower byte with upper byte change
4. Clear upper byte to AND off upper byte and operate lower
5. Upper byte of data bus to lower byte for all byte ops on upper byte
6. Load defined values from microcode for tamper-proof constants, vectors, etc.
7. Normal data input or address input without swap or modification.
8. Clear upper byte and data in low-bite immediate ops, etc.

INTERRUPT CONTROL

The powerful maskable priority vectored interrupt system (Figure 29) of the HEX-29 is a direct derivative of the incredible Am2914 bipolar LSI interrupt control IC. This circuit is so well integrated that it uses only one microword bit and requires very little support circuitry. The general set of operations that can be executed by the Am2914 is shown below. For more detailed information on this chip see the *Am2900 Family Data Book*.

F. Enable Request
E. Load Mask Register
D. Disable Request
C. Clear Mask Register
B. Bit Set Mask Register
A. Bit Clear Mask Register
9. Load Status Register
8. Set Mask Register
7. REad Mask Register
6. Read Status Register
5. Read Vector
4. Clear Interrupts Last Vector Read.
3. Clear Interrupts via M Register
2. Clear Interrupts via M Bus
1. Clear all Interrupts
0. Master Clear

Flow charts of the actions taken in microcode by the HEX-29 CPU are shown in Figure 30 and Figure 31.

DMA CONTROL

The DMA structure is quite straightforward. There are eight active-LOW DMA request lines and eight corresponding DMA acknowledge lines. The highest priority requesting a DMA cycle at the beginning of the microcycle before DMA will be allowed gets an acknowledge signal that lasts up until the DMA cycle – at least.

If no devices are requesting DMA, the $\overline{\text{NRQ}}$ (no request) bus signal goes LOW. This is an excellent opportunity for dynamic RAM circuitry to refresh sequential rows on each DMA cycle that $\overline{\text{NRQ}}$ is LOW.

Another input signal $\overline{\text{DDMA}}$, will override all priorities and not acknowledge any level of DMA request. This could be used by dynamic RAM refresh circuitry when it must be permitted to refresh itself soon or chance losing data.

Many schemes of DMA handling can be accomplished with this simple and uncomplicated priority controlled system. An Am74S374 captures the DMA requests (Figure 32) on a cycle by cycle basis. An Am2913 prioritizes these requests and acknowledges the highest level request with a three-bit binary code. An Am74S138 expands this to the eight bits of DMA acknowledge that correspond to the eight input bits. The Am2913 supplies the $\overline{\text{NRQ}}$ bus signal and provides for the $\overline{\text{DDMA}}$ bus signal.

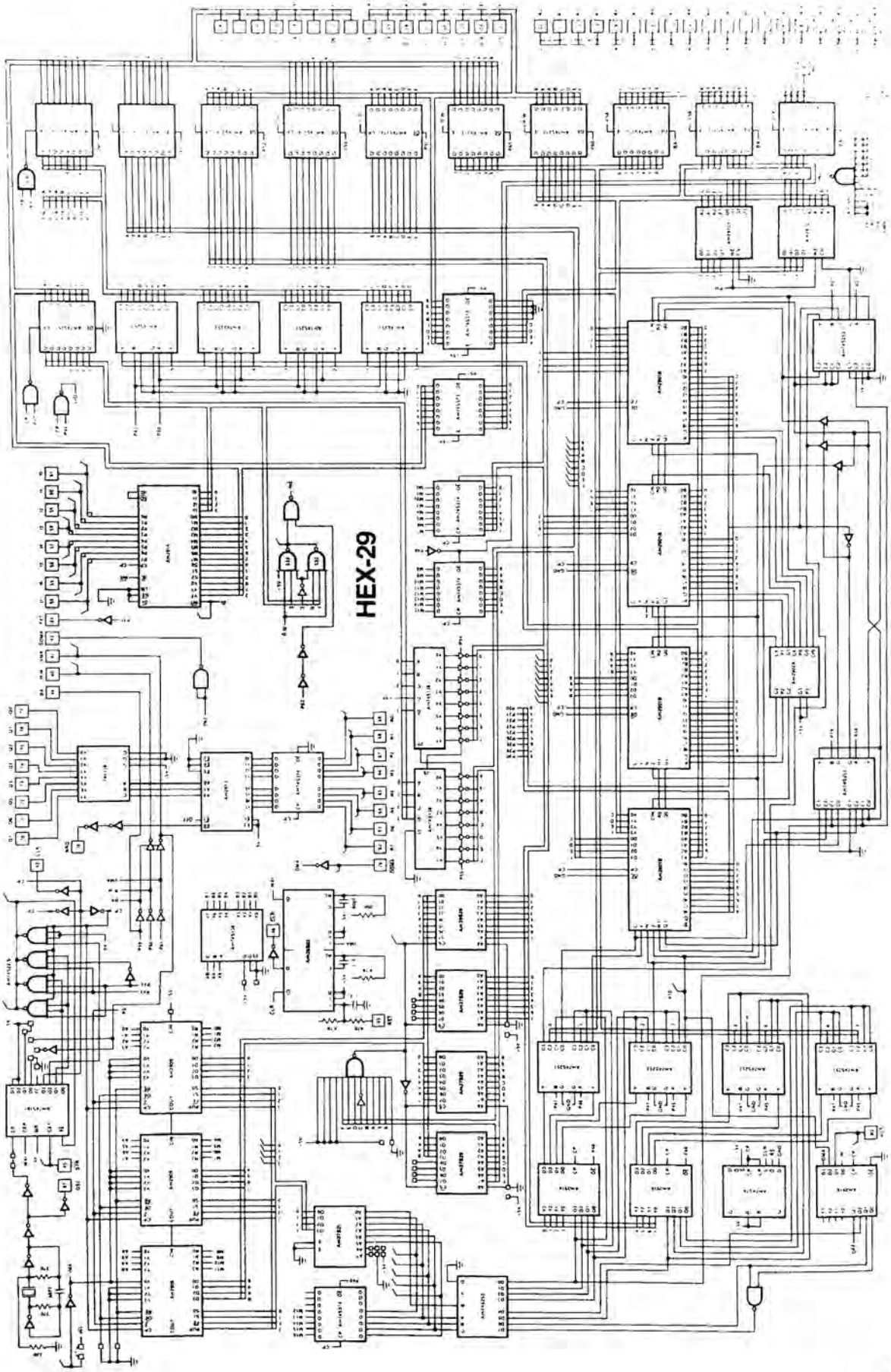


Figure 29.

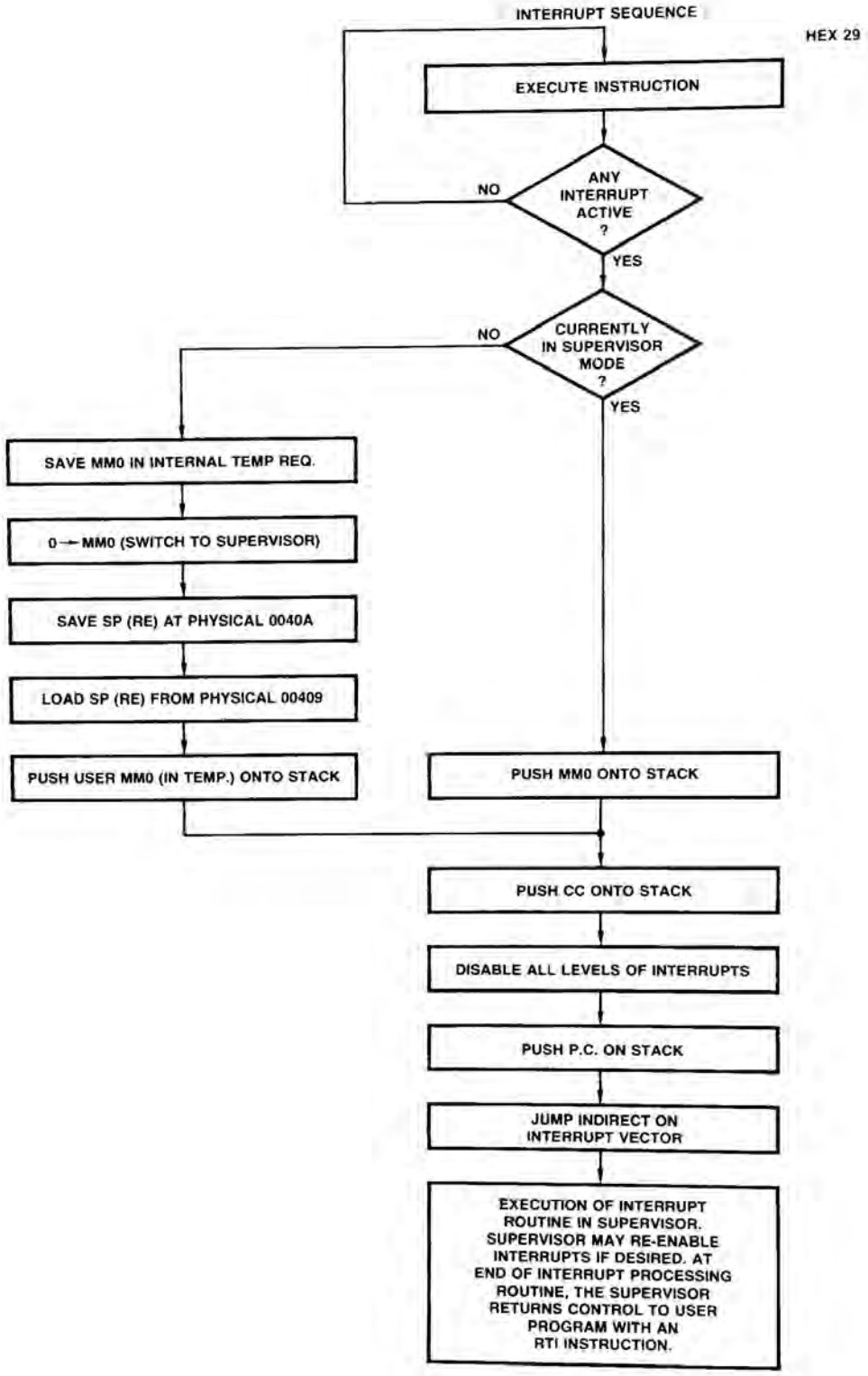

INTERRUPT SEQUENCE
HEX 29
EXECUTE INSTRUCTION
ANY INTERRUPT ACTIVE ?
NO
YES
CURRENTLY IN SUPERVISOR MODE ?
NO
YES
SAVE MM0 IN INTERNAL TEMP REQ.
0 → MM0 (SWITCH TO SUPERVISOR)
SAVE SP (RE) AT PHYSICAL 0040A
LOAD SP (RE) FROM PHYSICAL 00409
PUSH USER MM0 (IN TEMP.) ONTO STACK
PUSH MM0 ONTO STACK
PUSH CC ONTO STACK
DISABLE ALL LEVELS OF INTERRUPTS
PUSH P.C. ON STACK
JUMP INDIRECT ON INTERRUPT VECTOR
EXECUTION OF INTERRUPT ROUTINE IN SUPERVISOR. SUPERVISOR MAY RE-ENABLE INTERRUPTS IF DESIRED. AT END OF INTERRUPT PROCESSING ROUTINE, THE SUPERVISOR RETURNS CONTROL TO USER PROGRAM WITH AN RTI INSTRUCTION.
MPR-678

Figure 30.

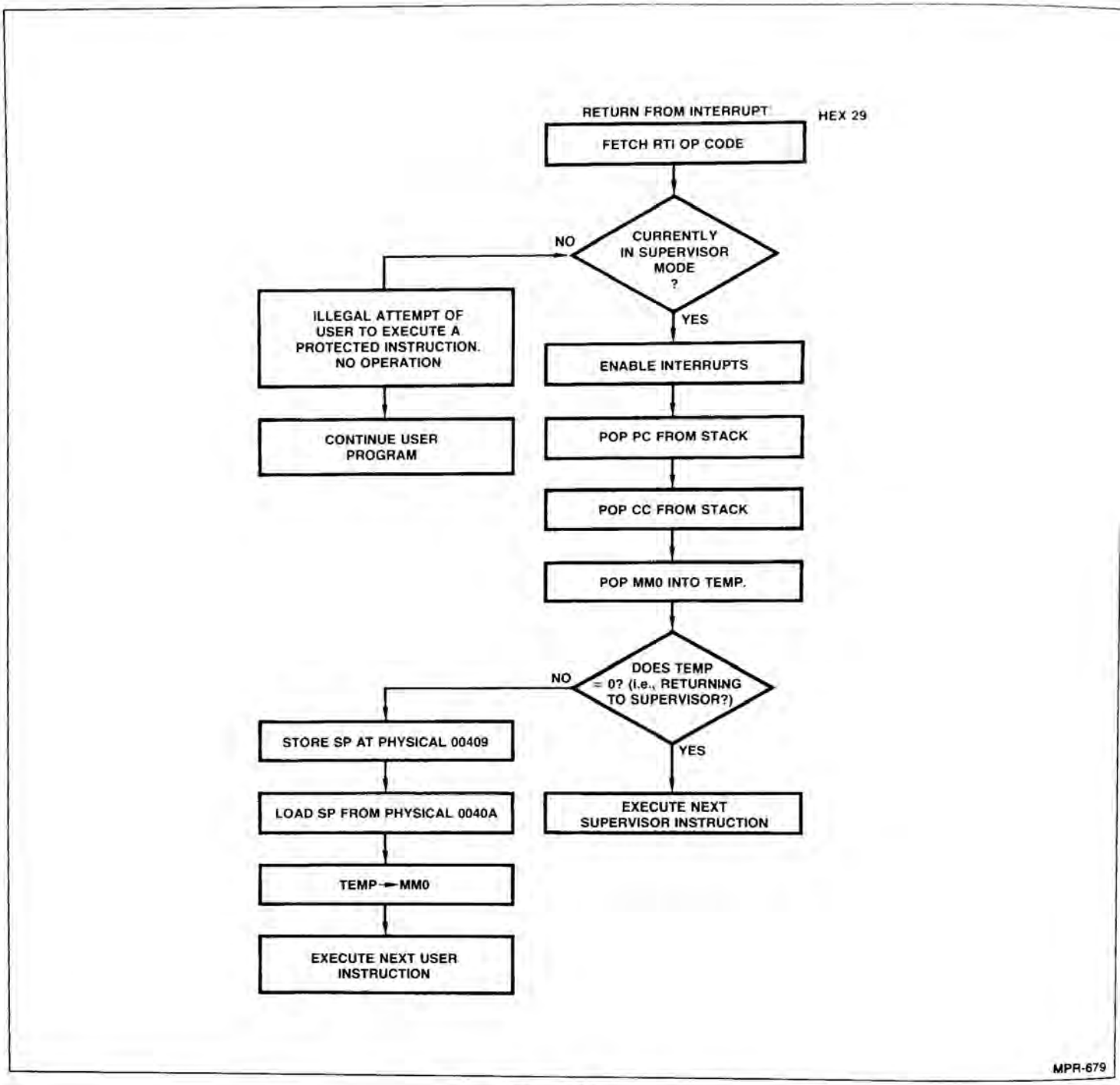

RETURN FROM INTERRUPT
HEX 29
FETCH RTI OP CODE
CURRENTLY IN SUPERVISOR MODE ?
NO
YES
ILLEGAL ATTEMPT OF USER TO EXECUTE A PROTECTED INSTRUCTION. NO OPERATION
CONTINUE USER PROGRAM
ENABLE INTERRUPTS
POP PC FROM STACK
POP CC FROM STACK
POP MM0 INTO TEMP.
DOES TEMP = 0? (i.e., RETURNING TO SUPERVISOR?)
NO
YES
STORE SP AT PHYSICAL 00409
LOAD SP FROM PHYSICAL 0040A
TEMP→MM0
EXECUTE NEXT USER INSTRUCTION
EXECUTE NEXT SUPERVISOR INSTRUCTION
MPR-679

Figure 31.

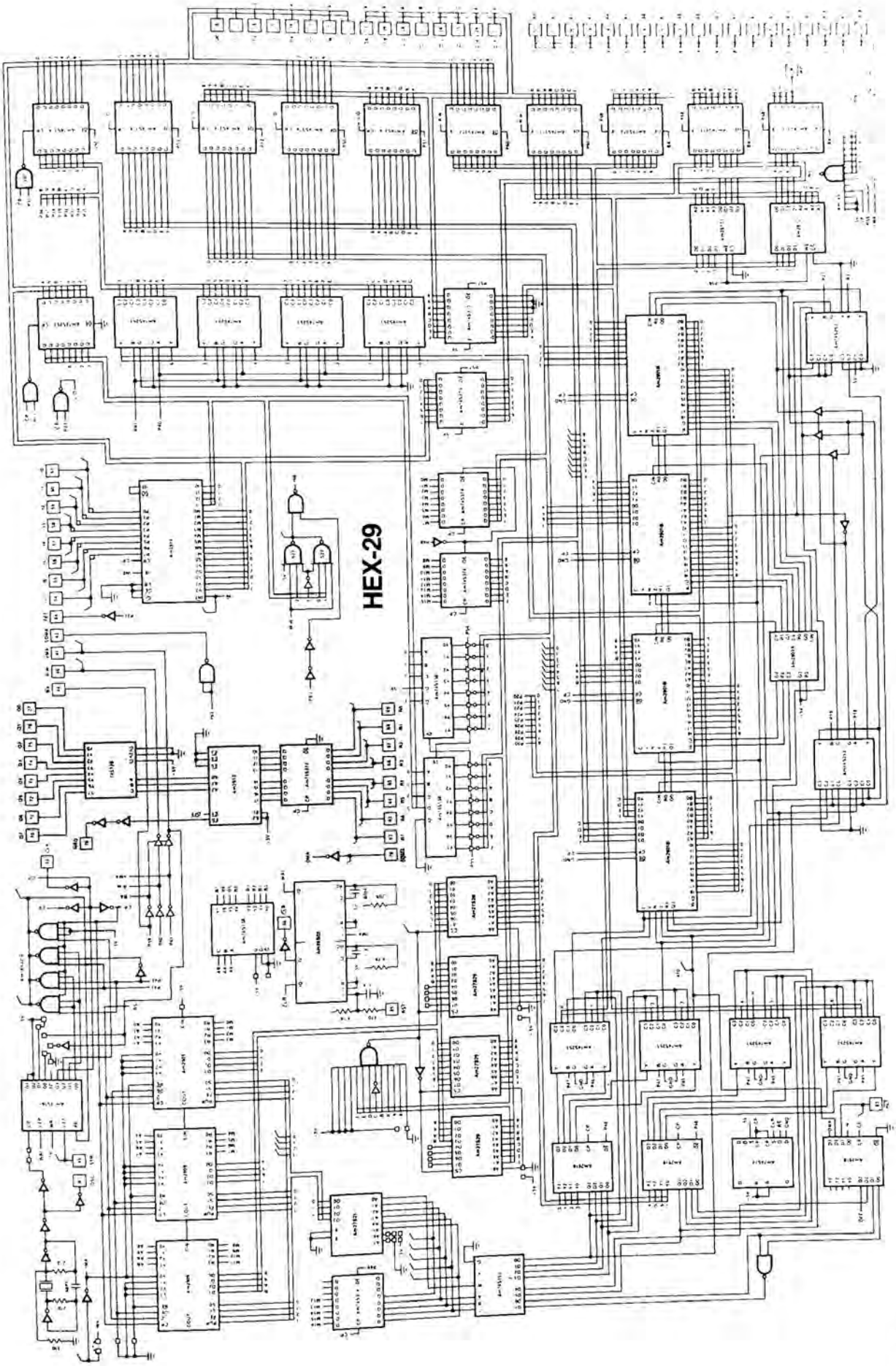


Figure 32.

SYSTEM BUS INTERFACE EXAMPLE HEX-64KBS STATIC MEMORY CARD

It was possible to design the system bus to be very simple to work with because the HEX-29 is a microprogrammed device. The following section discusses an implementation of a 64k byte static memory card for the HEX-29 system bus using Am9124 memory ICs. The purpose is to show that designing cards that interface with the HEX-29 system bus is relatively easy. Note that a design for I/O devices would be similar to this implementation since I/O devices are memory mapped and share exactly the same set of bus signals and timing requirements.

Starting from the left hand of the schematic shown in Figure 33, we find that the low 13 bits of the address bus and the four control bus signals (CLK, $\overline{VMA}$, R/$\overline{W}$, and WP) are buffered from the system bus by two Am74S240 ICs and three sections of an Am74S244. These are inverting and non-inverting buffers respectively, and offer extremely high current drive (64mA sink current) and very high speed (~4 to 6ns) with only very light bus loading (400μA low level).

Ten of the address lines buffered by these ICs then drive the address lines of half of the memory array through series type termination resistors. These resistors (~33 ohms) serve to prevent undershooting zero volts by more than the permissible 0.5V on negative edge transitions of the address lines. This type of termination has the advantage that it does not draw current from the driver ICs; it is highly recommended over split termination for memory arrays where current loading is negligible, but capacitive loading is significant. Note that to further reduce these capacitive loading effects, the address lines of only half of the memory array are driven by one set of buffers. (Find the second set of Am74S240 address buffers at the far right of the schematic.)

The remaining 3 address lines that were buffered by the Am74S240s drive the A, B, and C inputs of (4) Am74S138 one-of-eight decoders. These ICs develop the 32 1k word chip selects that enable the appropriate Am9124 memory ICs for read and write operations when they are addressed.

Of course only one of the Am74S138 ICs should be enabled when the board is addressed. This is a function of the higher address lines, A18-A13. Since each Am74S138 is able to select 1k word blocks of memory, each Am74S138 should be addressable on 8k word boundaries. Decoding the upper address lines (A18-A13) to match selectable 8k boundary addresses is accomplished with four Am25LS2521 8-bit equal to comparators, one for each Am74S138.

The DIP switches on the right hand side of each Am25LS2521 define the conditions under which the corresponding Am74S138 will be selected. When the eight inputs on the left hand side of these chips correspond to the values set on the DIP switch on the right, the Am74S138 is enabled. Note that the $\overline{VMA}$ bus signal (Valid Memory Access) must be LOW to enable the Am25LS2521. Also note that each 8k word bank can be unconditionally removed from the system memory space by leaving the lowest DIP switch open. Thus the board may be filled in 8k word increments if desired.

The Am74S138 ICs are also enabled by the system clock via the CLK signal. Therefore, memory chip selects can only occur during the time that the system clock is LOW (called ϕ_2). The importance of this will be discussed shortly. Another signal that must be valid for these ICs to be enabled is the DIS Signal. Whenever the R/W signal is LOW (indicating a write) and WP (write protect) is HIGH (protect the memory), then the DIS signal is brought LOW. This disables the Am74S138's and blocks the selecting of any memory ICs, thereby write protecting all on-board memory.

Above the memory array on the schematic are the data bus buffers, one set for each half. Again, this is done to reduce capacitive loading, this time on the data lines. Am74S373 octal tri-state latches are used for all eight of these data buffers. The enable inputs are driven by the inversion of the system clock bus signal so that they are transparent during all of ϕ_2, which is when the data is transferred. The appropriate Am74S373 latches are turned on ($\overline{OE}$ LOW) during read and write signals so that the data is buffered in the proper direction.

The Am26S02 one-shot is used to stretch ϕ_2 of the system clock to meet the access time of the memory. Without this signal, ϕ_2 would last only 80ns and the access time specifications of the Am9124 memory ICs would not be met. The Am26S02 is activated whenever memory ICs on the board are addressed when the system clock enters ϕ_2 (negative edge). Once fired, the duration of ϕ_2 is stretched by 40ns for every 40ns that the STR bus signal is held LOW. Since the Am9124 EPC memory devices have an access time of 200ns worst case, ϕ_2 must be stretched by 120ns.

Summary

As can be seen, the HEX-29 16-bit design represents a simple, straightforward design approach to building a high-performance 16-bit processor. This design takes advantage of many of the features of the Am2901 and Am2909. The instruction set shown in this application note is intended to be representative of the more common types of instructions to be executed on a machine of this class. In addition, microcode could be developed to execute a great many additional instructions as well as other classes of instruction such as entire floating point package. This design utilizes microprogram control throughout, and is a good demonstration of parallel microprogramming in a most straightforward application.

AMD wishes to thank Mr. Mike Simmons and Mr. Lee McDonald of HEX for their work on this invited paper as a part of this application note series.

APPENDIX

HEX-29 Microcode

This appendix contains 256 words of HEX-29 microcode. The first part is a definition file which defines the HEX-29 hardware structure for the AMDASM™ assembler. The various inputs to the Am2901 are defined via equates while all other microword fields are literally defined. The second part is the assembly file which symbolically, via terms defined in the definition phase, constructs each microword. Each microword begins with an optional label (such as RESET:). Next is the Am2909 branch control field, followed by all of the remaining control fields. This structure gives the appearance of a conventional assembler, i.e., LABEL, OPERATION, OPERANDS. A microinstruction which has no Am2909 branch control specified, such as microwords 3 and 4, uses microword bits 0-15 (which includes the branch control field) to place immediate data directly on the internal Am2901 bus. The Am2909 is then forced to "CONTINUE" by the "LIN" field. LIN, besides Latching IN the data on the Am2901 bus, disables the microprogram branch control register output, causing the "CONTINUE" function to be selected in the branch control PROM (see Figure 20B).

These 256 microwords represent a reasonable subset of the HEX-29 standard instructions, i.e., branch, conditional branch, data moves (MDV), and, or, add, sub, etc.

```
;
; HEX-29 DEFINITION PHASE (PHASE 1)
;
WORD 64
;
;AM2901 REGISTER EQUATES
;
R0:     EQU     H#0
R1:     EQU     H#1
R2:     EQU     H#2
R3:     EQU     H#3
R4:     EQU     H#4
R5:     EQU     H#5
R6:     EQU     H#6
R7:     EQU     H#7
R8:     EQU     H#8
R9:     EQU     H#9
R10:    EQU     H#A
R11:    EQU     H#B
R12:    EQU     H#C
R13:    EQU     H#D
R14:    EQU     H#E
R15:    EQU     H#F
;
;AM2901 SOURCE (R S) OPERAND EQUATES
;
AQ:     EQU     Q#0
AB:     EQU     Q#1
ZQ:     EQU     Q#2
ZB:     EQU     Q#3
ZA:     EQU     Q#4
DA:     EQU     Q#5
DQ:     EQU     Q#6
DZ:     EQU     Q#7
;
;AM2901 ALU FUNCTION (R FUNCTION S) EQUATES
;
ADD:    EQU     Q#0         ;R+S
SUBR:   EQU     Q#1         ;S-R
SUBS:   EQU     Q#2         ;R-S
OR:     EQU     Q#3         ;R S
AND:    EQU     Q#4         ;R S
NOT:    EQU     Q#5         ;R S
EXOR:   EQU     Q#6         ;R S
EXNOR:  EQU     Q#7         ;R S
;
;AM2901 DESTINATION CONTROL EQUATES
;
QREG:   EQU     Q#0         ;F Q , Y=F
NOOP:   EQU     Q#1         ;NOTHING , Y=F
RAMA:   EQU     Q#2         ;F B , Y=A
RAMF:   EQU     Q#3         ;F B , Y=F
RAMQD:  EQU     Q#4         ;F/2 B , Q/2 Q , Y=F
RAMD:   EQU     Q#5         ;F/2 B , Y=F
RAMQU:  EQU     Q#6         ;2F B , 2Q Q , Y=F
RAMU:   EQU     Q#7         ;2F B , Y=F
;
;
AM2901: DEF     24X,4VX,4VX,1X,3VX,1X,3VX,1X,3VX,20X
;
NOINE:  DEF     B#0,63X             ;INE = INTERRUPT LOGIC ENABLE
INE:    DEF     B#1,63X
;
NOSDMA: DEF     1X,B#0,62X          ;SDMA = SYNC DMA
SDMA:   DEF     1X,B#1,62X
;
VMA:    DEF     2X,B#0,61X          ;VALID MEMORY ACCESS THIS CYCLE
NOVMA:  DEF     2X,B#1,61X
;
WRITE:  DEF     3X,B#0,60X          ;READ/NOT-WRITE MEMORY
READ:   DEF     3X,B#1,60X
;
BA:     DEF     4X,B#0,59X          ;BUS AVAILABLE THIS CYCLE
NOBA:   DEF     4X,B#1,59X
;
NOLAD:  DEF     5X,B#0,58X          ;LAD = ENAB TRANSPARENT ADDR LATCH
LAD:    DEF     5X,B#1,58X
;
RMM:    DEF     6X,B#0,57X          ;READ MEMORY MAP
NORMM:  DEF     6X,B#1,57X
;
LMM:    DEF     7X,B#0,56X          ;LOAD MEMORY MAP
NOLMM:  DEF     7X,B#1,56X
;
CLRHL:  DEF     8X,B#00,54X         ;CLEAR HI/LO BYTE INTERNAL BUS
CLRLO:  DEF     8X,B#01,54X
CLRHI:  DEF     8X,B#10,54X
NOCLR:  DEF     8X,B#11,54X
;
SWPHL:  DEF     10X,B#00,52X        ;SWAP DATA HI/LO INTERNAL BUS
SWPLO:  DEF     10X,B#01,52X
SWPHI:  DEF     10X,B#10,52X
NOSWP:  DEF     10X,B#11,52X
;
DINHL:  DEF     12X,B#00,50X        ;DATA IN HI/LO INTERNAL BUS
DINLO:  DEF     12X,B#01,50X
DINHI:  DEF     12X,B#10,50X
NODIN:  DEF     12X,B#11,50X
;
NOSDA:  DEF     14X,B#0,49X         ;SDA = SELECT MICROWORD BITS 15-0
SDA:    DEF     14X,B#1,49X         ;      TO INTERNAL BUS
;
RCC:    DEF     15X,B#0,48X         ;READ CC TO INTERNAL BUS
NORCC:  DEF     15X,B#1,48X
;
NOLCC:  DEF     16X,Q#0,45X         ;CC MUX SELECT
LCC:    DEF     16X,Q#1,45X
LCV:    DEF     16X,Q#2,45X
LCCLCV: DEF     16X,Q#3,45X
LNZ:    DEF     16X,Q#4,45X
LNZCC:  DEF     16X,Q#5,45X
LNZCV:  DEF     16X,Q#6,45X
LNZCCV: DEF     16X,Q#7,45X
;
NOLDI:  DEF     19X,B#0,44X         ;LDI = LATCH DATA IN, SWAPPED & UNSWAPPED
LDI:    DEF     19X,B#1,44X
;
NOSTR:  DEF     20X,B#00,42X        ;STR = CLOCK STRETCH CONTROL
STR1:   DEF     20X,B#01,42X
STR2:   DEF     20X,B#10,42X
STR3:   DEF     20X,B#11,42X
;
MWAMWB: DEF     22X,B#00,40X        ;2901 A,B MUX SELECT
RSRD:   DEF     22X,B#01,40X
RSRS:   DEF     22X,B#10,40X
RDRD:   DEF     22X,B#11,40X
;
NOFTCH: DEF     32X,B#0,31X         ;FETCH = FETCH INSTRUCTION
FETCH:  DEF     32X,B#1,31X
;
NCCIB:  DEF     36X,B#0,27X         ;CIB = CARRY-IN MUX SELECT BIT B
CIB:    DEF     36X,B#1,27X
;
NOCIA:  DEF     40X,B#0,23X         ;CIA = CARRY-IN MUX SELECT BIT A
CIA:    DEF     40X,B#1,22X
;
RCLMUL: DEF     44X,B#00,18X        ;SHIFT & ROTATE MUX CTL BIT
RCLRCR: DEF     44X,B#01,18X
DRLASR: DEF     44X,B#10,18X
LSLLSR: DEF     44X,B#11,18X
;
NOLIN:  DEF     46X,B#0,17X         ;LATCH-IN LOW NIBBLE OF DATA BUS
LIN:    DEF     46X,B#1,17X
;
BRC0:   DEF     47X,B#00000,12X     ;2909 NEXT INSTRUCTION CONTROL
BRC1:   DEF     47X,B#00001,12X
BRV0:   DEF     47X,B#00010,12X
BRV1:   DEF     47X,B#00011,12X
BRN0:   DEF     47X,B#00100,12X
BRN1:   DEF     47X,B#00101,12X
BRZ0:   DEF     47X,B#00110,12X
BRZ1:   DEF     47X,B#00111,12X
BRHS0:  DEF     47X,B#01000,12X
BRHS1:  DEF     47X,B#01001,12X
BRLZ0:  DEF     47X,B#01010,12X
BRLZ1:  DEF     47X,B#01011,12X
BRHLT0: DEF     47X,B#01100,12X
BRHLT1: DEF     47X,B#01101,12X
BRIH0:  DEF     47X,B#01110,12X
BRIH1:  DEF     47X,B#01111,12X
BRANCH: DEF     47X,B#10000,12X
CALL:   DEF     47X,B#10010,12X
CN0:    DEF     47X,B#10100,12X
RTSZ1:  DEF     47X,B#10110,12X
RET:    DEF     47X,B#11000,12X
BRIHMAP: DEF    47X,B#11110,12X
CONTNUE: DEF    47X,B#11111,12X
;
DATA:   DEF     46X,16VH#0000       ;MICROWORD DATA
;
;
END
;
; HEX-29 MICROPROGRAM ASSEMBLY PHASE
;
;  (FIRST 256 MICROPROGRAM WORDS)
;
;
RESET:  CONTNUE     & AM2901  R0    , R0    , QREG  , ADD   , AQ
/                   & NOINE   & SDMA  & NOVMA & READ
/                   & BA      & NOLAD & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & NOSDA & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & NOLIN
;
        CONTNUE     & AM2901  R0    , R0    , QREG  , ADD   , AQ
/                   & NOINE   & NOSDMA & NOVMA & READ
/                   & BA      & NOLAD & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & NOSDA & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & NOLIN
;
        CONTNUE     & AM2901  R0    , R0    , NOOP  , AND   , DZ
/                   & NOINE   & NOSDMA & NOVMA & WRITE
/                   & NOBA    & NOLAD & NORMM & NOLMM & NOCLR
/                   & NOSWP   & NODIN & NOSDA & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & NOLIN
;
                    & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & INE     & SDMA  & NOVMA & WRITE
/                   & NOBA    & NOLAD & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & SDA   & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & LIN   & DATA  H#000D
;
                    & AM2901  R15   , R15   , RAMF  , OR    , DZ
/                   & INE     & SDMA  & NOVMA & READ
/                   & BA      & NOLAD & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & SDA   & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & LIN   & DATA  H#0200
;
        BRANCH IFETCH & AM2901  R0    , R0    , QREG  , ADD   , AQ
/                   & NOINE   & NOSDMA & NOVMA & READ
/                   & BA      & NOLAD & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & NOSDA & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA   & RCLMUL & NOLIN
;
;
        ORG     H#0008
;
IFETCH:             & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE   & SDMA  & VMA   & READ
/                   & NOBA    & LAD   & NORMM & LMM   & NOCLR
/                   & NOSWP   & NODIN & NOSDA & NORCC & LCCLCV
/                   & LDI     & NOSTR & MWAMWB & FETCH & NOCIB
/                   & NOCIA   & RCLMUL & LIN   & DATA  H#F000
INSTR:  BRIHMAP INR & AM2901  R15   , R15   , RAMF  , ADD   , ZB
/                   & NOINE   & NOSDMA & NOVMA & READ
/                   & BA      & LAD   & NORMM & LMM   & NOCLR
/                   & NOSWP   & DINHL & NOSDA & NORCC & LCCLCV
/                   & NOLDI   & NOSTR & MWAMWB & NOFTCH & NOCIB
/                   & CIA     & RCLMUL & NOLIN
;
        ORG     H#000C
;
BRA+:   BRANCH INSTR & AM2901  R15   , R15   , RAMF  , ADD   , DA
/                   & NOINE   & SDMA  & VMA   & READ
/                   & NOBA    & LAD   & NORMM & LMM   & CLRLO
/                   & NOSWP   & DINHI & NOSDA & NORCC & LCCLCV
/                   & LDI     & NOSTR & MWAMWB & FETCH & NOCIB
/                   & NOCIA   & RCLMUL & NOLIN
;
BRA-:   BRANCH INSTR & AM2901  R15   , R15   , RAMF  , ADD   , DA
/                   & NOINE   & SDMA  & VMA   & READ
/                   & NOBA    & LAD   & NORMM & LMM   & NOCLR
```

```
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BSR+:   CONTNUE         & AM2901  R14    , R14    , RAMF   , SUBR   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & LAD    & NORMM  & LMM
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & NOCLB
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH IFETCH   & AM2901  R15    , R15    , RAMA   , ADD    , DA
/                       & NOINE  & NOSDMA & VMA    & WRITE
/                       & NOBA   & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BSR-:   CONTNUE         & AM2901  R14    , R14    , RAMF   , SUBR   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & LAD    & NORMM  & LMM
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & NOCLB
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH IFETCH   & AM2901  R15    , R15    , RAMA   , ADD    , DA
/                       & NOINE  & NOSDMA & VMA    & WRITE
/                       & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BC+:    BRC0    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCIB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BC-:    BRC0    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNC+:   BRC1    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNC-:   BRC1    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BV+:    BRV0    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BV-:    BRV0    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & NOCIB
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNV+:   BRV1    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNV-:   BRV1    IFETCH  & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNZ+:   BRZ1    0008    & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NCLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/       BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                       & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
BNZ-:   BRZ1    0008    & AM2901  R15    , R15    , QREG   , ADD    , DA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                       & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN
/
        BRANCH INSTR    & AM2901  R15    , R15    , RAMF   , ADD    , ZQ
/                       & NOINE  & SDMA   & VMA    & READ
/                       & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/
```

```
/                   & LDI    & NOSTR  & MWAMWP & FETCH  & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

BB*:     BRHS0  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , ADD   , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

BB~:     BRHS0  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , ADD   , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

BNB*:    BRHS1  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , ADD   , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

BNB~:    BRHS1  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , ADD   , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCIB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

DBNZ+:   CONTNUE       & AM2901  R9    , R9    , RAMF  , SUBR  , ZA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRLZ1  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , OR    , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

DBNZ-:   CONTNUE       & AM2901  R9    , R9    , RAMF  , SUBR  , ZA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN
         BRLZ1  IFETCH & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , OR    , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

SQUEEZ:  CONTNUE       & AM2901  R0    , R0    , QREG  , AND   , AQ
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & RSRS   & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R0    , R0    , NOOP  , SUBR  , DQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & NOLCC
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & CIA    & RCLMUL & NOLIN

CBB*:    BRANCH BIC    & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

CBB~:    CONTNUE       & AM2901  R15   , R15   , QREG  , ADD   , DA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
```

```
/                   & NOCIA  & RCLMUL & NOLIN

BIC:                & AM2901  R10   , R8    , RAMF  , AND   , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWP & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & LIN    & DATA   & H#FFFF

         CONTNUE       & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWP & NOFTCH & NOCIB
/                      & CIA    & RCLMUL & NOLIN

         CONTNUE       & AM2901  R8    , R8    , NOOP  , SUBR  , DA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & SWPH1  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & CIA    & RCLMUL & NOLIN

BRC0     IFETCH & AM2901  R8    , R8    , NOOP  , SUBR  , DA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & DINBI  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & CIA    & RCLMUL & NOLIN

BRC0     IFETCH & AM2901  R8    , R8    , NOOP  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         BRANCH INSTR  & AM2901  R15   , R15   , RAMF  , ADD   , ZQ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & LDI    & NOSTR  & MWAMWB & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

ALSRR:   CONTNUE       & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & LSLLSR & NOLIN

RFLAG:   BRANCH INSTR  & AM2901  R0    , R0    , NOOP  , ADD   , ZB
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCV
/                      & LDI    & NOSTR  & RSRD   & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

ASRRR:   BRANCH RFLAG  & AM2901  R0    , R0    , RAMD  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & DRLASR & NOLIN

ROLRR:   BRANCH RFLAG  & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & ROLROR & NOLIN

RORRR:   BRANCH RFLAG  & AM2901  R0    , R0    , RAMD  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & ROLROR & NOLIN

RLCRR:   BRANCH RFLAG  & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

MVNCCR:  BRANCH INSTR  & AM2901  R0    , R0    , RAMF  , OR    , DZ
/                      & NOINE  & SDMA   & VMA    & READ
/                      & NOBA   & NOLAD  & NORMM  & LMM    & CLRLO
/                      & NOSWP  & NODIN  & NOSDA  & NCC    & LCCLCV
/                      & LDI    & NOSTR  & RDRD   & FETCH  & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

LSRRR:   BRANCH RFLAG  & AM2901  R0    , R0    , RAMD  , ADD   , ZA
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
/                      & NOCIA  & LSLLSR & NOLIN

VLSRR:                 & AM2901  R0    , R0    , QREG  , AND   , DA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & SDA    & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & RDRE   & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & LIN    & DATA   & H#000F

VLSOOP:  BRLZ1  IFETCH & AM2901  R0    , R0    , NOOP  , ADD   , ZB
/                      & NOINE  & NOSDMA & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCV
/                      & NOLDI  & NOSTR  & RSRS   & NOFTCH & NOCIB
/                      & NOCIA  & RCLMUL & NOLIN

         CONTNUE       & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LNZCCV
/                      & NOLDI  & NOSTR  & RSRS   & NOFTCH & NOCIB
/                      & NOCIA  & LSLLSR & NOLIN

         BRANCH VSLOOP & AM2901  R0    , R0    , QREG  , SUBR  , ZQ
/                      & NOINE  & SDMA   & NOVMA  & READ
/                      & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                      & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                      & NOLDI  & NOSTR  & MWAMWB & NOFTCH & NOCIB
```

```
/                       & NOCIA  & RCLMUL & NOLIN

VSLRR:                  & AM2901  R0    , R0    , QREG  , OR    , DZ
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & SDA    & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & LIN    & DATA    H#000F

        BRANCH VSLOOP   & AM2901  R0    , R0    , QREG  , AND   , DQ
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & CLRLO
/                       & NOSWP  & DINRI  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

DSLRR:  CONTNUE         & AM2901  R0    , R0    , RAMF  , AND   , DZ
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

                        & AM2901  R9    , R9    , QREG  , AND   , DA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & SDA    & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & LIN    & DATA    H#001F

CSLOOP: BRLZ1  IFETCH   & AM2901  R0    , R0    , NOOP  , ADD   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & NOLDI  & NOSTR  & RDRD   & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

        BRN1   IFETCH   & AM2901  R0    , R0    , NOOP  , ADD   , AQ
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LNZCCV
/                       & NOLDI  & NOSTR  & RSRS   & NOFTCH  & NOCIB
/                       & NOCIA  & LSLLSR & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LNZCCV
/                       & NOLDI  & NOSTR  & RDRD   & NOFTCH  & NOCIB
/                       & NOCIA  & ROLROR & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMF  , ADD   , ZB
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & CIA    & RCLMUL & NOLIN

        BRANCH CSLOOP   & AM2901  R0    , R0    , QREG  , SUBR  , ZQ
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

DSLRR:                  & AM2901  R9    , R9    , QREG  , AND   , DA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & SDA    & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN   & DATA    H#001F

DSLOOP: BRLZ1  IFETCH   & AM2901  R0    , R0    , NOOP  , ADD   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & NOLDI  & NOSTR  & RDRD   & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LNZCCV
/                       & NOLDI  & NOSTR  & RSRS   & NOFTCH  & NOCIB
/                       & NOCIA  & LSLLSR & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMU  , ADD   , ZA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LNZCCV
/                       & NOLDI  & NOSTR  & RDRD   & NOFTCH  & NOCIB
/                       & NOCIA  & ROLROR & NOLIN

        BRANCH DSLOOP   & AM2901  R0    , R0    , QREG  , SUBR  , ZQ
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & MWAMWB & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

MOVRCC: BRANCH INSTR    & AM2901  R0    , R0    , NOOP  , ADD   , ZB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LNZ
/                       & LDI    & NOSTR  & RDRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

XCHRR:  CONTNUE         & AM2901  R0    , R0    , QREG  , ADD   , ZA
/                       & NOINE  & SDMA  & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & RDRD   & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

        CONTNUE         & AM2901  R0    , R0    , RAMF  , ADD   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & READ
/                       & BA     & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & NOLDI  & NOSTR  & RSRD   & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

/                       & NOCIA  & RCLMUL & NOLIN

        BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , ADD   , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

MVNRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , AND   , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

MOVRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , ADD   , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

ADDRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , ADD   , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

ADCRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , ADD   , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & CIB
/                       & NOCIA  & RCLMUL & NOLIN

SUBRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , SUBR  , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & CIA    & RCLMUL & NOLIN

SBCRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , SUBR  , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & CIB
/                       & NOCIA  & RCLMUL & NOLIN

ANDRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , AND   , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

IORRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , OR    , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

XORRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , EXOR  , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

CMPRR:  BRANCH INSTR    & AM2901  R0    , R0    , NOOP  , SUBR  , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & CIA    & RCLMUL & NOLIN

INCRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , ADD   , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & CIA    & RCLMUL & NOLIN

DECRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , SUBR  , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

COMRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , EXNOR , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

NEGRR:  BRANCH INSTR    & AM2901  R0    , R0    , RAMF  , SUBS  , ZA
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & CIA    & RCLMUL & NOLIN

SWPRR:  CONTNUE         & AM2901  R0    , R0    , NOOP  , ADD   , ZA
/                       & NOINE  & NOSDMA & NOVMA  & WRITE
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & LCCLCV
/                       & LDI    & NOSTR  & RSRD   & NOFTCH  & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

        BRANCH INSTR    & AM2921  R0    , R0    , RAMF  , OR    , DZ
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & SWPHL  & NODIN  & NOSDA  & NORCC   & LCV
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & NOCIA  & RCLMUL & NOLIN

RSUBRR:  BRANCH INSTR   & AM2901  R0    , R0    , RAMF  , SUBS  , AB
/                       & NOINE  & SDMA  & VMA    & READ
/                       & NOBA   & NOLAD  & NORMM  & LMM     & NOCLB
/                       & NOSWP  & NODIN  & NOSDA  & NORCC   & NOLCC
/                       & LDI    & NOSTR  & RSRD   & FETCH   & NOCIB
/                       & CIA    & RCLMUL & NOLIN
```

```
MOVAX:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMA  , OR    , ZA
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & NODIN  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

MVNXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH INSTR & AM2901  R15   , R10   , RAMA  , OR    , DZ
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & FETCH  & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

MOVXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

AIN:    BRANCH INSTR & AM2901  R15   , R10   , RAMA  , OR    , DZ
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & LDI     & NOSTR  & MWAMWB & FETCH  & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

ADDXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CIRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , ADD   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

ADCXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CIRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , ADD   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & CIB
/                    & NOCIA   & RCLMUL & NOLIN

SUBXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CIRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , SUBR  , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA     & RCLMUL & NOLIN

SBCXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NCBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , SUBR  , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & CIB
/                    & NOCIA   & RCLMUL & NOLIN

ANDXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , AND   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

ICRXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , OR    , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCIB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

XORXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , ADD   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

CMPXA:  CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA     & RCLMUL & NOLIN

INCX:   CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R8    , R8    , RAMF  , ADD   , DZ
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA     & RCLMUL & NOLIN

DECX:   CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R8    , R8    , RAMF  , SUBS  , DZ
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCIB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

COMX:   CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R8    , R8    , RAMF  , EXNOR , DZ
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCIB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

NEGX:   CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R8    , R8    , RAMF  , SUBR  , DZ
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA     & RCLMUL & NOLIN

SWPX:   CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & NOSDMA & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CLRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R8    , R8    , RAMF  , OR    , DZ
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NOBA    & NOLAD  & NORMM  & LMM    & NOCLB
/                    & SWPRL   & NODIN  & NOSDA  & NORCC  & LCV
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

RSUBXA: CONTNUE      & AM2901  R11   , R11   , NOOP  , ADD   , DA
/                    & NOINE   & SDMA   & VMA    & READ
/                    & NOBA    & LAD    & NORMM  & LMM    & CIRLO
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & LCCLCV
/                    & LDI     & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

        BRANCH IFETCH & AM2901  R10   , R10   , RAMF  , SUBS  , DA
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & DINBI  & NOSDA  & NORCC  & NOLCC
/                    & NOLDI   & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA     & RCLMUL & NOLIN

MVNRM-: CONTNUE      & AM2901  R0    , R0    , RAMF  , SUBR  , ZB
/                    & NOINE   & NOSDMA & NOVMA  & READ
/                    & BA      & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP   & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & NOLDI   & NOSTR  & RDAD   & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

MOUT:   BRANCH IFETCH & AM2901  R0    , R0    , RAMA  , ADD   , ZB
/                    & NOINE   & NOSDMA & VMA    & WRITE
/                    & NCBA    & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP   & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & NOLDI   & NOSTR  & RSAS   & NOFTCH & NOCIB
/                    & NOCIA   & RCLMUL & NOLIN

MVNM+R: CONTNUE      & AM2901  R0    , R0    , RAMA  , ADD   , ZA
```

```
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

MVNIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , OR    , DZ
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

MVNI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH MVNIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

MVK2R:  CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH MVNIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

MOVM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

MOVIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , OR    , DZ
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCV
                & NOLDI  & NOSTR  & RSRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

MOVI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

MIN:    BRANCH MOVIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

MOVZR:  CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH MOVIN & AM2901  R0    , R0    , NOOP  , ALD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADDM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

ADDIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , ADD   , DA
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & NOLCC
                & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADDI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH ADDIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADDZR:  CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH ADDIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADCM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
```

```
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

ADCIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , ADD   , DA
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & NOLCC
                & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADCI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH ADCIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ADCZR:  CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH ADCIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

SUBM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

SUBIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , SUBR  , DA
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & NOLCC
                & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

SUBI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH SUBIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

SUBZR:  CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH SUBIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

SUBCM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

SUBCIN: BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , SUBR  , DA
                & NOINE  & NOSDMA & NOVMA  & READ
                & BA     & NOLAD  & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & NOLCC
                & NOLDI  & NOSTR  & RDRD   & NOFTCH & CIB
                & NOCIA  & RCLMUL & NOLIN

SUBCI+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH SUBCIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

SUBCZR: CONTNUE & AM2901  R15   , R15   , RAMA  , ADD   , ZA
                & NOINE  & NOSDMA & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
                & CIA    & RCLMUL & NOLIN

        BRANCH SUBCIN & AM2901  R0    , R0    , NOOP  , ALD   , DA
                & NOINE  & SDMA   & VMA   & READ
                & NOBA   & LAD    & NORMM  & LMM    & NOCLB
                & NOSWP  & DINBL  & NOSDA  & NORCC  & LCCLCV
                & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
                & NOCIA  & RCLMUL & NOLIN

ANDM+R: CONTNUE & AM2901  R0    , R0    , RAMA  , ADD   , ZA
                & NOINE  & SDMA   & VMA   & READ
```

```
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

ANDIN:   BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , AND   , DA
/                   & NOINE  & NOSDMA & NOVMA  & READ
/                   & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCV
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

ANDI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH ANDIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

ANDZR:   CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH ANDIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

IORM+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

IORIN:   BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , OR    , DA
/                   & NOINE  & NOSDMA & NOVMA  & READ
/                   & PA     & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & ICV
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

IORI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH IORIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

IORZR:   CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH IORIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

XORM+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

XORIN:   BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , EXOR  , DA
/                   & NOINE  & NOSDMA & NOVMA  & READ
/                   & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCV
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

XORI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH XORIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

XORZR:   CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH XORIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

CMPM+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & SDMA   & VMA    & READ
```

```
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

CMPIN:   BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , ADD   , DA
/                   & NOINE  & NOSDMA & NOVMA  & READ
/                   & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCV
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

CMPI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH CMPIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

CMPZR:   CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH CMPIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

RSBM+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

RSUBIN:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , SUBS  , DA
/                   & NOINE  & NOSDMA & NOVMA  & READ
/                   & BA     & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & NOLCC
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

RSBI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH RSUBIN & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

RSUBZR:  CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH RSUBIN & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & SDMA   & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

INCMR:   CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

INCOUT:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , ADD   , DZ
/                   & NOINE  & NOSDMA & VMA    & WRITE
/                   & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & ICV
/                   & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

INCI+R:  CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH INCOUT & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

INCZR:   CONTNUE    & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                   & CIA    & RCLMUL & NOLIN

         BRANCH INCOUT & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                   & NOINE  & NOSDMA & VMA    & READ
/                   & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                   & NOSWP  & DINHL  & NOSDA  & NORCC  & LCCLCV
/                   & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                   & NOCIA  & RCLMUL & NOLIN

DECMR:   CONTNUE    & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                   & NOINE  & NOSDMA & VMA    & READ
```

```
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

DECOUT:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , SUBS  , DZ
/                    & NOINE  & NOSDMA & VMA   & WRITE
/                    & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCV
/                    & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

DECI+R:  CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH DECOUT & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN
DECZR:   CONTNUE     & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH DZCOUT & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

COMMR:   CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

COMCUT:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , EXNOR , DZ
/                    & NOINE  & NOSDMA & VMA   & WRITE
/                    & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCV
/                    & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

COMI+R:  CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH COMOUT & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

COMZR:   CONTNUE     & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH COMCUT & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

NEGMR:   CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

NEGCUT:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , SUBR  , DZ
/                    & NOINE  & NOSDMA & VMA   & WRITE
/                    & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & NCLCC
/                    & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

NEGI+R:  CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH NEGOUT & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

NEGZR:   CONTNUE     & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH NEGOUT & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

SWPMR:   CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCIB
```

```
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

SWPOUT:  BRANCH IFETCH & AM2901  R0    , R0    , RAMF  , OR    , DZ
/                    & NOINE  & NOSDMA & VMA   & WRITE
/                    & NOBA   & NOLAD  & NORMM  & LMM    & NOCLB
/                    & SWPRL  & NODIN  & NOSDA  & NORCC  & LCV
/                    & NOLDI  & NOSTR  & RDRD   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

SWPI+R:  CONTNUE     & AM2901  R0    , R0    , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH SWPOUT & AM2901  R0    , R0    , NOOP  , OR    , DZ
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

SWPZR:   CONTNUE     & AM2901  R15   , R15   , RAMA  , ADD   , ZA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & NODIN  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & MWAMWB & NOFTCH & NOCIB
/                    & CIA    & RCLMUL & NOLIN

         BRANCH SWPOUT & AM2901  R0    , R0    , NOOP  , ADD   , DA
/                    & NOINE  & NOSDMA & VMA   & READ
/                    & NOBA   & LAD    & NORMM  & LMM    & NOCLB
/                    & NOSWP  & DINRL  & NOSDA  & NORCC  & LCCLCV
/                    & LDI    & NOSTR  & RSRS   & NOFTCH & NOCIB
/                    & NOCIA  & RCLMUL & NOLIN

END
```