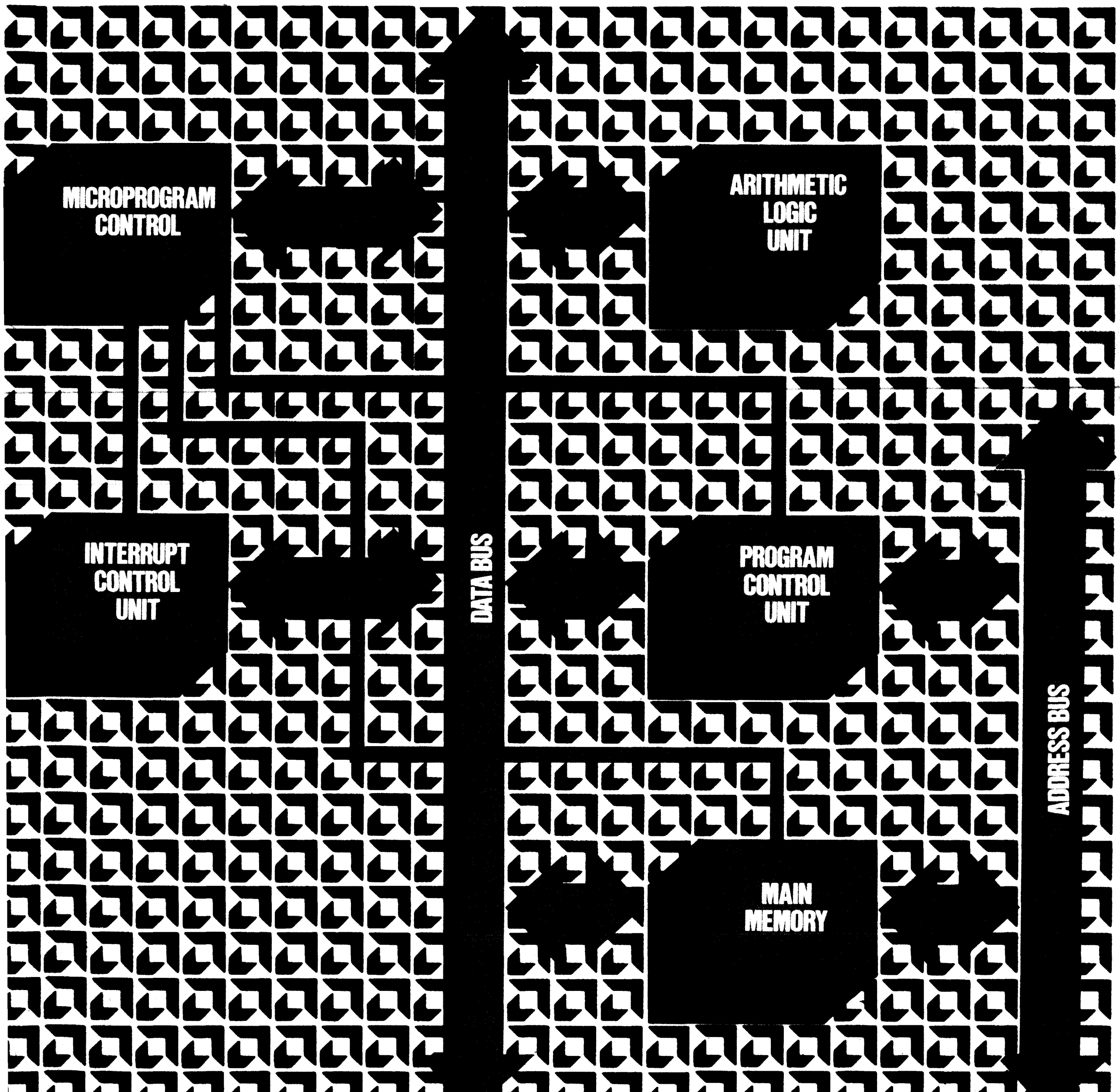
Advanced Micro Devices, Inc.

# A Microprogrammed 16-Bit Computer

$5.00



# TABLE OF CONTENTS

# INTRODUCTION

The purpose of this Application Note is to acquaint the reader with some of the many facets of computer design. Beginning with a statement of purpose, we will consider the specification of a computer from a functional standpoint. Presentation of a modular architecture will then help the designer to understand the flow of data and instructions through the system and to help resolve the inevitable problems of resource contention (the desire to use a bus or functional subsystem for more than one purpose at a time). From a management perspective, early definition of the architecture modules allows a number of people to work on the design of different controlled subsystems concurrently.

The use, or intended use of a microcomputer will be illuminated by its instruction set, the form or forms of its instructions, and the form and format of its data words. The computer hardware, then, must satisfy all of the functional requirements of the machine instructions. (In a stored-program computer, the "machine instructions" are those system control statements that reside in the main memory and are fetched and executed by the computer control unit.)

Having defined the instruction set, the various controlled subsystem components may be designed. The state transition diagrams may then be specified as can the control signal requirements of the system controller.

# MICROPROGRAMMED MACHINES

The Original Equipment Manufacturer, OEM, has a number of goals that he would like to satisfy in the design of any new product. He must first satisfy his current requirements; however, he would like to build into his equipment the ability to expand the machine, increase its speed or thru-put, and allow for additional functions and modernization. Secondly, very few machines are ever designed without some flaw that escapes the manufacturer's attention until it gets into the hands of the users in the field; the ability to change the machine without extensive engineering change orders is paramount. A machine that is dedicated to a specific task or a class of tasks, rather than a general purpose processor is becoming the rule rather than the exception as hardware costs become less important than software costs. It is very desirable, then, to be able to build a processor that is capable of executing very complex tasks without a large stored program; perhaps, a single macro-instruction will suffice. The OEM may have a previous generation of hardware that was implemented with random logic in the form of MSI and SSI elements. Because of a large investment in software, it would be desirable to implement the same instruction set using newer technology with an emulation in hardware of the previous machine.

A microprogrammed processor will allow the manufacturer the flexibility to satisfy any or all of those requirements. Rather than trying to define a microprogrammed machine in an elegant, abstract way, the concepts of the microprogram will be demonstrated by example.

# CHAPTER I ENGINEERING MANAGEMENT

From an engineering management standpoint it is most important that the following events occur in the order presented: production of a written specification, an architectural design, derivation of a management schedule based on available resources, the implementation of the various architectural blocks, system integration, an engineering review, redesign, and documentation. A management review should take place at each of these milestones before the next phase of the development is started.

## Specification

It is extremely important that a written specification for the system is completed and agreed upon by all interested parties before any other work is initiated. Otherwise, the program is bound to be extremely inefficient in manpower, will probably require much more redesign activity, and may be very demoralizing to the technical staff. Further, there will be no way to measure progress.

In the definition of a computer, the specification may be delineated by at least the following sections: Design Goals, Formats, Instruction Requirements, Addressing Methods, Input-Output, Memory, and Control Panel. The form of the specification may change from company-to-company, or even from design-to-design within one company, but the intent of the specification should remain the same. The purpose of the specification is to provide the philosophy behind the product, a complete functional description, and any important limitations that are to be imposed. This is a reference document for the design, marketing, production, and management staffs.

## Architecture

The architectural design will provide a description of each of the controlled functions in the machine, each of the buses in the system and the order of functional module contention resolution, if any, and the system controller.

Each of the controlled functions, e.g., the Arithmetic Logic Unit, should be described functionally, and each storage device within the functional block should be described. The Computer Control Unit should be presented in a similar manner.

The buses, and the approach to bus control that will be implemented should be described from a functional viewpoint.

## Management Schedule

Having defined the architectural blocks of the system, it is then possible to start the design of the system. While the controlled elements of the system are being conceptualized the Computer Control Unit and the memory system timing philosophies should be worked on in detail. Ultimately, this portion of the design will provide the speed limitation on the system. Considering the selected architecture, each class of instruction should be flow-charted as a general case and a state-transition graph for the processor should be derived. Each machine instruction to be implemented should then be examined carefully to illuminate the demands placed on each subsystem; these demands should be tabulated by subsystem so that the designer of each subsystem understands what his portion of the system must be capable of doing. This point in the development cycle is an excellent time for a program review to determine whether the design goals can be met with the approach that has been initiated.

The implementation of the various architectural blocks should then be scheduled. These tasks are very straight-forward and should provide no surprises if the initial conceptual work was done carefully. Each subsystem should be completely tested by itself before any attempt is made to connect the subsystems together.

After completing the subsystem tests, the subsystems should be integrated (connected together using common buses, clocks, and control signals). From a management standpoint, system integration is very inefficient. Except for a very large system only one or two engineers can work on the complete system tests. Generally, the engineer who designed the Computer Control Unit and one other senior engineer should be assigned to this task. This leaves the remainder of the project staff idling without any direct contribution to the system design and test effort. This is an excellent time to start the documentation effort and liaison with the various production groups.

The engineering review really consists of two separate reviews separated in time. The first review is internal to the engineering department and consists of design trade-offs. The second review is a company-wide, management level review which should consider the effects of the design on incoming inspection and Q.A., purchasing, automatic test, mechanical and electrical assembly, final test, marketing, sales, engineering support, field service, etc. The technically oriented business decisions should also be addressed during the second meeting. The level and cost of inventory, the manufacturing cost of the product, the marketing related support costs, reliability and warranty exposure, spare parts inventory, time to repair and field service costs, cost of operating and maintenance supplies, installation time and cost, environmental constraints, size, power consumption (can it be installed without having to rewire the installation facility), all of the human factors and esthetic considerations, packaging, effect on installed air conditioning and/or requirements for special cooling, "brownout" protection, UL, CSA, IEC, etc., liability. From the corporate viewpoint, is the current staff, facility, and equipment capable of supporting manufacturing and selling of the product? Quite often the business considerations significantly outweigh the technical considerations. It is an exceptional, engineering department, or a poor management team, that is capable of passing this second engineering review. For this reason it is important that a redesign period be scheduled into the program plan.

How much time should be scheduled for the redesign? This is very hard to judge and will change from company to company, but the time for redesign is usually controlled directly by the longest lead-time requirement for any non-design related task or product. For example, packaging components typically have longer lead-times associated with them than do standard IC's. In any case the engineering redesign time estimate, and the longest lead-time estimate, and the engineering reimplementation time, and a less stringent engineering review time might be summed together to provide a conservative redesign task.

Good quality, complete documentation may mean the difference between financial success and failure. The documentation effort should continue with the direct support of the original engineering team. The product build-up, "explosion," starting with the highest level assembly and terminating with the lowest level assembly is important because it will provide purchasing order quantities that they may purchase against a forecast demand. This documentation should also include an approved vendor list of each component.

Schematics, mechanical layouts, cabling diagrams, and any other information necessary to produce, test, trouble-shoot, purchase components, etc., must also be supplied during this phase of the development. A mechanism for instituting engineering change orders must also be provided.

# CHAPTER II SPECIFICATION OF THE MACHINE

*Because this application note is being written for a broad spectrum of users and users' requirements, this specification and the resulting hardware design may not be the "best" implementation of a modern processor. There are a number of excursions from what might be considered an optimum design to illustrate technique or design trade-offs.*

## DESIGN GOALS

The intention of this design is to provide a third generation computer with a general-purpose instruction set that is optimized for high-speed input-output operations.

### Function

A typical functional specification is presented here for the purpose of presenting an application requirement.

This processor shall be capable of interfacing to 12-bit, 2's complement analog-to-digital and digital-to-analog converters with 500ns conversion times. Further, the ability to interface with dual synchronous or asynchronous communications lines operating up to 50 kilo-baud per channel and two parallel, full word-length channels directly connected to a host computer. (These interface designs are not part of this application note.)

The initial design must provide for 2's complement add and subtract, the basic logic functions, indexed data movement instructions and a set of executive level I/O instructions that will provide for minimum service or interrupt request overhead. The computer must allow for expansion of its basic instruction set and for the addition, in microcode, of statistical and communications protocol algorithms.

At least one channel of direct memory access, DMA, and 16-levels of vectored priority interrupt are required. The processor must be capable of acknowledging and processing an interrupt without polling, calculating, or otherwise determining which channel caused the interrupt. The DMA function must be slaved to the interrupt controller. That is, when an interrupt request occurs either a vectored interrupt or a direct memory access is initiated.

The input-output channel interface must be very simple. All service contention (requests for input-output service generated by multiple data channels at the same time) will be resolved in the interrupt controller.

### Speed

The main memory full cycle time must be less than or equal to the 500ns data conversion times including address decoding, data source and destination control and processor overhead functions.

The processor cycle time should be the minimum necessary to service the various controlled functions without variable or asynchronous clock functions. All of the central processor subsystems will be synchronous, but the memory and input-output controllers will be required to furnish a ready signal after each attempt to read or write has been satisfied. This will permit memory and peripheral devices of different speeds to be attached to the processor.

### Electrical

All of the circuit elements must run off a single five volt power supply only; current requirements will be determined during the design effort. Other voltages will be supplied for control of peripheral devices only.

### Mechanical

A 9x12 inch printed circuit board format will be used. Two 2x36 edge connectors will be used on the bottom edge of the card (along the major dimension).

Approximately 78 sixteen-pin integrated circuits, or an equivalent number of larger packages, may be mounted on a single, two-layer printed circuit board if cables are connected to the top of the card. Otherwise, about 96 sixteen-pin packages, or equivalent, may be installed on one board. This circuit density is around one 16-pin package per square inch.

If a four-layer printed circuit board is used, with the two buried layers being power and ground distribution, on board decoupling and filter capacitors may be removed and the circuit density may be increased to one 16-pin package per 0.75 square inches. This yields a package count of 101 per board or 128 per board depending on whether or not cables are connected to the top of the board.

Cooling will be convective or forced-air depending upon the power density on the boards. Printed circuit board spacing will also depend on the power density and the ambient, intake air supply temperature.

### Technology

Due to the speed requirement and the single 5V power supply availability, a bipolar integrated circuit technology must be used for the digital logic. To reduce the current requirement, low-power Schottky devices will be used where possible. Similarly, low-power, high-speed memory devices that are designed in static, n-channel MOS technology should be utilized for main memory.

## MAIN MEMORY FORMATS

Main memory will contain instruction and data information. To maximize the data transfer role and the instruction efficiency a main memory word size of 16-bits is specified. No parity bit will be used. The bits will be numbered from 0 to 15.

### Data Format

The main memory arithmetic data format will be integer, 2's complement. The least significant bit will be 0 and the most significant bit 14. Bit 15 will be the sign bit. With the exception of the sign bit, the binary weight of any bit position will be 2, raised to the power of the bit position. The sign bit has the weight $-2^{15}$.

Logical data formats will include 16-bit words and 8-bit bytes. Two 8-bit bytes may be stored in a single main memory word. The lower byte is contained in memory bit positions 0 to 7, and the upper byte is contained in bit positions 8 to 15.

### Instruction Format

Machine instructions will consist of one or two main memory words. (The general format will be similar to an IBM 360 instruction.) There will be two single word instruction formats and two double word instruction formats: Register-to-Register (RR), Special Format (SF), Register Indexed (RX), and Register Immediate (RI).

The operation to be performed by the current instruction will reside in the upper byte of the memory word, bits 8 through 15, providing for 256 different instructions. The remainder of the first word, and the second word if it exists, will be used for operand(s). (Operands provide data, addresses, or data source and destination information relating to the current instruction.)

## INSTRUCTION REQUIREMENTS

Machine instructions may be broken down into a number of general functional classes: arithmetic, logic, shift, data movement, program control (branch), input-output, and executive. It is best to think of the instructions in these classes because it will be easier to define, implement, and test the instructions necessary to complete the tasks imposed on the product by its work environment.

### Arithmetic

The following arithmetic functions are required:

- Add
- Add with Carry
- Subtract
- Subtract with Carry
- Arithmetic Compare
- Two's Complement (Negate)
- Increment
- Decrement

### Logic

The following logical functions are required:

- And
- Inclusive-Or
- Exclusive-Or
- Logical Compare (NOR)
- Complement

### Shift

The following shift functions are required:

- Shift Register Left
- Shift Register Right
- Shift Register Left with Carry
- Shift Register Right with Carry
- Rotate Register Left
- Rotate Register Right
- Rotate Register Left with Carry
- Rotate Register Right with Carry

### Data Movement

The following data movement instructions are required:

- Load Register from Memory
- Store Register in Memory
- Transfer from Register-to-Register

### Program Control

The following program control instructions are required:

- Jump
- Jump-to-Subroutine
- Return

### Input-Output

The following input-output instructions are required:

- Read Data into Register
- Read Status into Register
- Write Data from Register
- Write Status from Register

### Executive

*The executive instructions are difficult to define in a preliminary specification because they relate to the processor design, not to the tasks the processor must accomplish. All of the machine instructions necessary to load interrupt vectors, direct memory addresses and word counts, and interrupt masks and pointers must be included in this set of instructions, as well as enable and disable interrupt and DMA commands, etc.*

## ADDRESSING METHODS

The program control and data addressing techniques available in the machine level programs will depend upon the instruction class that is used.

In the register-to-register instruction, RR, Figure 1a, the data source and destination are both register addresses; $R_1$ and $R_2$. Very similar to the RR instruction is the special-function, SF, Figure 1b. The data to be tested, manipulated, compared, or otherwise operated on is selected by the operand field $R_1$. The second field, F, is used to specify a particular function.

In the case of the register-indexed instruction, RX, Figure 1c, the first operand is specified by field $R_1$ and the second operand is specified by the sum of the address field, word 2, and the contents of the general purpose register selected for use as an index register by the field X. The zero-th register may not be used as an index register. If zero is specified in the X field the effective address of the second operand is the address field contents of the second instruction word.

The register-immediate instruction, RI, Figure 1d, is very similar to the RX instruction except the second operand data is already contained in the A-field (the second instruction word). As in the RX instruction, the A-field of the RI instruction will be added to the contents of the register specified by the X-field, but, instead of forming an effective address for the second operand, the result of the summation is literal data that will be used *immediately* as the second operand. Again, if X is zero, the second word is not modified prior to use.

Next instruction addressing (program control) will be accomplished with a program counter, PC, a push-pop stack, STACK, and a set of program modification instructions. The program modification instructions jump, jump-to-subroutine, and return-from-subroutine will all affect the next instruction address by controlling either the PC or the STACK.

## INPUT-OUTPUT

Peripheral controllers will be self-contained, high-speed, parallel transfer devices that will be capable of both programmed and automatic operation. All hardware necessary to control the peripheral device, request CPU service and handle the transfer of data will be contained in the controller.

### Programmed I/O

The peripheral controller will be capable of handling two output instructions: Output Data and Output Command. The device address will be impressed on the eight lower-order address bus lines and the data or command word will be imposed on the system data bus.

The processor will expect a ready signal from the peripheral controller before it will execute a programmed I/O instruction. If the I/O port is not ready when the instruction execution is attempted, it will go into a wait state until the device ready signal can be synchronized with the Computer Control Unit.

### Automatic I/O

Automatic I/O will consist of three different types of input-output service requests: vectored interrupt, cycle stealing direct memory access, and CPU freeze direct memory access. There will be 16-levels of priority encoded service request, the necessary contention logic, and 16-levels of request acknowedged circuitry. All service requests will be processed through the Interrupt Control Unit.

An interrupt request will be serviced after the current instruction being executed is finished, and before the next instruction is fetched. The PC will be pushed onto the stack and a vector address in main memory, previously assigned to a particular level of interrupt, will be loaded into the PC. Execution of a return from interrupt instruction will cause the stack to be popped onto the PC and normal processing will resume.

Cycle stealing Direct Memory Access will not have to wait for an instruction to be concluded to be given permission to steal a memory cycle. If the memory is not being used during a given cycle the DMA controller is free to attempt an access during which time interrupts may not be serviced. This mode of operation only permits the reading or writing of a single word, and then the DMA controller must relinquish system control.

Block transfer Direct Memory Access will wait for service in the same manner that an Interrupt Request waits. When the current instruction being executed is complete, the DMA controller will usurp control of the data bus and peripheral processor until the entire DMA sequence is satisfied. A block transfer DMA operation may not be interrupted.

## MAIN MEMORY

Main memory will consist of static RAM's organized in 4096 words by 16-bits. Up to 16 of these memory blocks may be directly addressed by any instruction or control unit, without paging, for a total addressing range of 64K words.

The memory will be built from NMOS static RAM's using a single power supply of +5Vdc and providing an access time of 200ns or less and a total cycle time of 400ns or less. The memory system must be capable of providing a READY flag in response to a read or write request.

The maximum load presented to any data bus or address bus by any single memory board will be 0.4mA or less. A low-capacitance, low-current bus will be maintained.

## CONTROL PANEL

A system control panel will be provided to aid both the logic designer and the programmer in implementing and debugging their tasks. As well as examining or writing main memory and monitoring CPU status, the control panel shall be capable of loading programs, interrupting or restarting programs, pausing and continuing tasks, check-point/restart, etc.

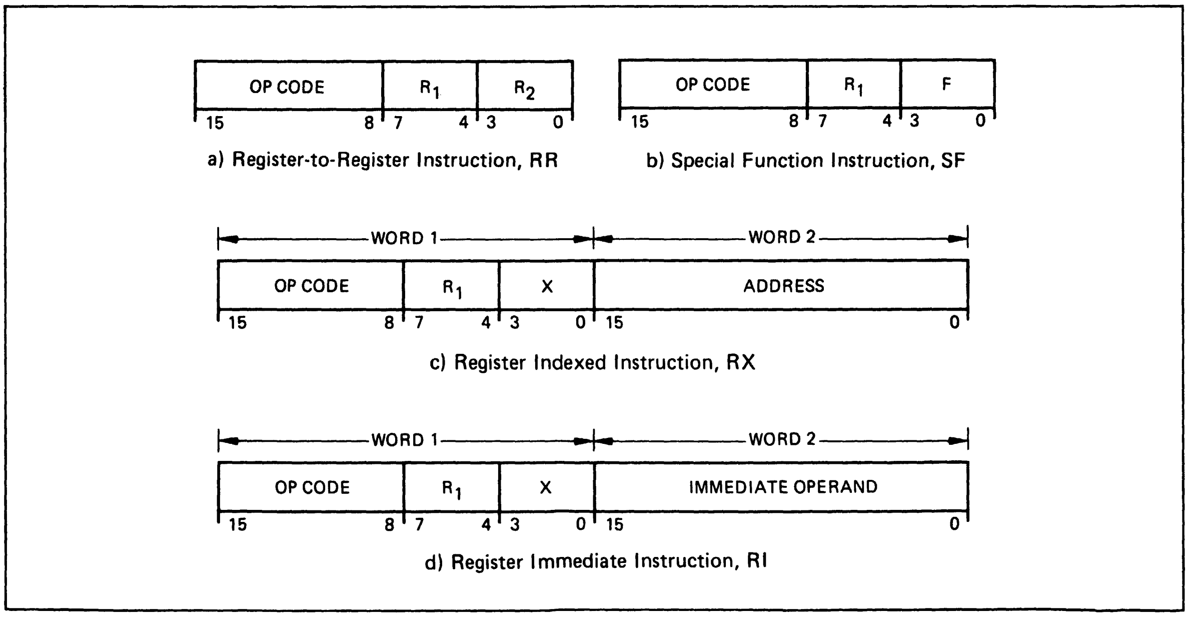


**Figure 1. Instruction Types.**

# CHAPTER III ARCHITECTURE

The computer architecture is depicted in Figure 2. There are two major buses in the processor: A 16-bit wide address bus, and a 16-bit wide data bus. There is a third bus in the system, the control bus, but it will be discussed later with the computer control unit. The data and address buses are three-state buses; each system element that connects to a bus will present one Am25LS, low-power Schottky unit load to the bus.

## Arithmetic Logic Unit

The Arithmetic Logic Unit, ALU, will be used to perform all of the combinatorial arithmetic and logic functions as well as the shift and rotate instructions. Arithmetic and logical comparisons, 1's and 2's complement, and incrementing and decrementing will be accomplished in the ALU. The ALU will also contain 16 general purpose registers and a Q-register for use as a scratch pad or for long shift sequences. The ALU may act as either a source or a destination for data on the data bus.

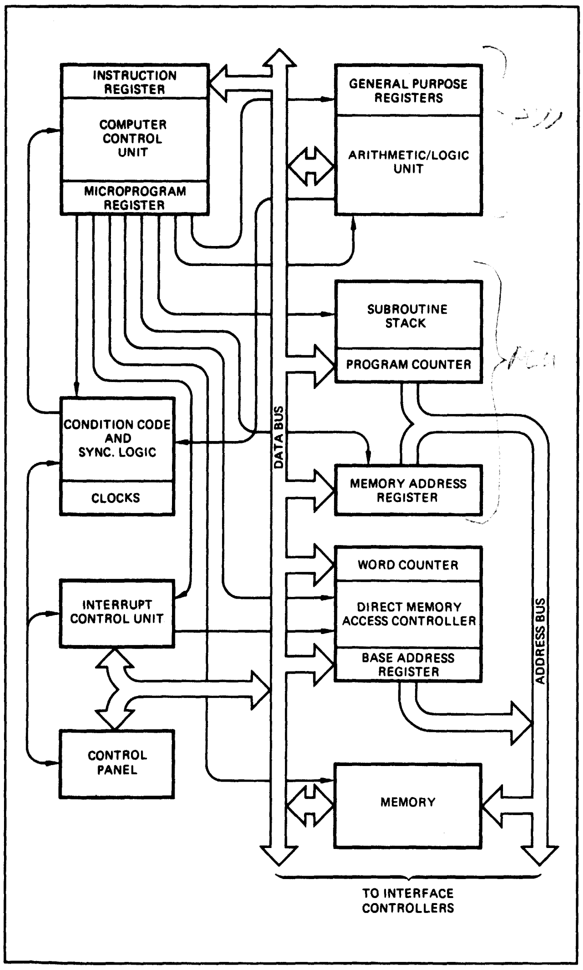


**Figure 2. Computer Architecture.**

## Program Control Unit

The Program Control Unit, PCU, consists of three important elements: the Program Counter, PC, the Memory Address Register, MAR, and the Subroutine Stack. When the system's primary power is turned on, the PC is cleared to 0. From that time on the PC will point to (contain the address of) the next instruction to be fetched and executed. The MAR is used to hold the second operand of an instruction or the effective address of the second operand which was calculated in the ALU. In either case the MAR is loaded from the data bus. Either the PC or the MAR is enabled onto the address bus at the start of any memory cycle.

There are three types of instruction that the PCU will execute when it is not merely incrementally stepping its way through a program of instructions. They are called program modification instructions and consist of: Jump, Jump-to-Subroutine, and Return.

The Jump instruction is a two word instruction. The first word, with the operation code "Jump," is loaded into the instruction register and decoded. If the instruction is a register indexed, RX, type and the condition code is true, an effective jump address will be calculated and the contents of that address will be jammed into the PC. If the instruction type is register immediate, RI, the second word of the instruction will be added to the index value and directly loaded into the PC. Once the jump takes place the PC will continue incrementing its way through program storage until another program modification instruction is encountered or its largest value has been reached. The Jump-to-Subroutine instruction is very similar except that the Stack is used. If the condition code in the first word of the instruction is true, the contents of the PC are pushed onto the Last-In-First-Out, (LIFO) Subroutine Stack. The address of the subroutine, having been derived in the same manner as the Jump address, is loaded into the PC and processing continues. The Return instruction is executed at the end of a subroutine sequence and causes the last entry into the Stack to be popped onto the PC, thereby returning the program control to the address where the subroutine was called plus one.

## Memory

With a total memory size of 64K x 16 we will look at the mechanical design goal of 12 x 9 inch printed circuit boards. It immediately becomes obvious that a single board will hold a 16K x 16 memory module along with its associated decoders, buffers, and transceivers. Modules of different speed memories may be mixed in a single system due to an asynchronous memory ready signal which is synchronized by the CCU.

CCu = computer control unit.

## Interrupt Control Unit

ICU

The ICU is capable of receiving 16 levels of interrupt request from peripheral controllers. Each of those interrupt request lines is capable of being masked (enabled or disabled), and cleared under program control. Further, a minimum interrupt level, or fence, may be implemented with an instruction. The highest level interrupt will win the contention race and will be presented in the form of 4-bit interrupt address vector.

The 4-bit vector is presented to a 16 word by 16-bit two port RAM which is loaded under program control with interrupt vector addresses. The data input port and the "B" output port are connected to the data bus. The "A" output port is used to detect a DMA attempt.

When an interrupt request is received, the CCU grants interrupt permission at a convenient point in the microinstruction sequence. The interrupt acknowledge is transmitted back to the requesting peripheral controller and the interrupt vector is enabled onto the system data bus. With the interrupt vector as the address source the PCU is commanded to do a Jump-to-Subroutine. When the interrupt program is finished, a Return-from-Interrupt will cause the interrupt to be cleared and the PCU to do a pop. Normal data processing will continue.

The only invalid interrupt vector address is $FFFF_{16}$, which is the highest address the machine is capable of handling. This address is reserved for DMA linkages. If the DMA linkage condition is detected on the "A" output port of the interrupt vector RAM, control is passed to the DMA controller.

## Direct Memory Aceess

A single DMA controller channel is planned for this machine although any number may be added. For example, if 16 levels of DMA were desired, 16 interrupt vector addresses could be invalidated, i.e., $FFF0_{16}$ through $FFFF_{16}$. (The DMA controller may reside either within the CPU, or on the interface circuit board.)

The DMA consists of two 16-bit binary counters with a terminal count detector. One of the counters is called the address counter, and its outputs are connected to the address bus through a three-state buffer. The other is the word tranfer counter. Both counters' parallel inputs are connected to the data bus, so that they may be initialized.

The address counter is initially loaded with the address to, or from, which the first data transfer will take place, depending upon the direction of the data flow. The requesting peripheral controller will be the source of the destination. The word counter is initially loaded with the 2's complement of the number of words to be transferred. Each time a data transfer takes place, both the address and word counters are incremented simultaneously. When the word counter reaches its terminal count the last DMA transfer will be completed.

## Computer Control Unit

The CCU consists of an instruction register whose input port is connected to the data bus and whose Op code output port is connected to a starting address PROM. The output of the starting address PROM is connected to the direct input of a microprogram sequencer, the output of which drives a microprogram PROM. The microprogram memory is 256 words deep by 40-bits wide which is sufficient for the present machine design. A pipeline register is used to hold the current microinstruction to increase the system thru-put.

The outputs from the Micro Program Register, MPR, control each of the architectural blocks. In fact, some of the MPR information is fed back to control the CCU. The system clocks, condition code and synchronization logic and control panel logic are all components of the CCU.

# CHAPTER IV
# MACHINE INSTRUCTIONS

This section presents the machine instructions that are to be implemented in the processor. They exist in four instruction types (Figure 1) and seven instruction classes. The four types include register-register, special function, register indexed, and register immediate. The seven classes include arithmetic, logic, shift, data movement, program control, input-output and executive. Not all instructions will be implemented in all four types because they either do not make much sense or are redundant, or they have little or no value. In the symbolic descriptions, the expression on the left of the arrow is the location or device to be changed. Parentheses and square brackets denote "the contents of"; braces indicate terms which are collected prior to use.

## ARITHMETIC

All of the instructions in this class are executed in the ALU portion of the computer. The condition codes available in the ALU include carry, "C," the most significant bit, "S," overflow, "O," word contents zero, "Z," upper byte zero, "U," lower byte zero, "L," A greater than B, "A," and A less than B, "B." The condition codes "A" and "B" are only valid after a subtract function of the form A minus B. Condition codes only change after the execution of ALU instructions.

### ADD

Add Registers, RR

| | | |
|---|---|---|
| ADR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) + (R_2)$

Add Register Indexed, RX

| | | | |
|---|---|---|---|
| ARX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) + [A + (X)]$

Add Register Immediate, RI

| | | | |
|---|---|---|---|
| ARI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) + A + (X)$

Add Register to Memory, RX

| | | | |
|---|---|---|---|
| ARM | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$\{A + (X)\} \leftarrow (R_1) + [A + (X)]$

Note: For all RI and RX formats, if X = 0 then $(X) \equiv 0$.

Add Register Indexed with Carry, RX

| | | |
|---|---|---|
| ARC | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) + (R_2) + C$

Add Registers Indexed with Carry, RX

| | | | |
|---|---|---|---|
| AXC | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) + [A + (X)] + C$

Add Register Immediate with Carry, RI

| | | | |
|---|---|---|---|
| AIC | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) + A + (X) + C$

### SUBTRACT

Subtract Registers, RR

| | | |
|---|---|---|
| SUR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) - (R_2)$

Subtract Register Indexed, RX

| | | | |
|---|---|---|---|
| SRX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) - [A + (X)]$

Subtract Register Immediate, RI

| | | | |
|---|---|---|---|
| SRI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) - \{A + (X)\}$

Subtract Registers with Borrow, RR

| | | |
|---|---|---|
| SRC | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) - (R_2) - C$

Subtract Registers Indexed with Borrow, RX

| | | | |
|---|---|---|---|
| SXC | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1) - [A + (X)] - C$

**COMPARE**

Arithmetic Compare Registers, RR

| | | |
|---|---|---|
| CAR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$CC \leftarrow (R_1) : (R_2)$

Arithmetic Compare Register Indexed, RX

| | | | |
|---|---|---|---|
| CAX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$CC \leftarrow (R_1) : [A + (X)]$

Arithmetic Compare Register Immediate, RI

| | | | |
|---|---|---|---|
| CAI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$CC \leftarrow (R_1) : \{A + (X)\}$

**COMPLEMENT**

Two's Complement Register, RR

| | | |
|---|---|---|
| TWOS | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (\overline{R_1}) + 1$

**INCREMENT AND DECREMENT**

Increment Register, RR

| | | |
|---|---|---|
| INCR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) + 1$

Note: For all RI and RX formats, if X = then $(X) \equiv 0$.

Increment Memory Indexed, RX

| | | | |
|---|---|---|---|
| INCM | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$\{A + (X)\} \leftarrow [A + (X)] + 1$

Decrement Register, RR

| | | |
|---|---|---|
| DECR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1) - 1$

Decrement Memory Indexed, RX

| | | | |
|---|---|---|---|
| DECM | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$\{A + (X)\} \leftarrow [A + (X)] - 1$

**LOGIC**

All of the instructions in the logic class are executed in the ALU portion of the computer. The same condition codes that were presented with the arithmetic instructions are incumbent on the logic instructions. The condition codes only change after the execution of an ALU instruction.

**AND**

And Registers, RR

| | | |
|---|---|---|
| NRR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1)$ . AND . $(R_2)$

And Registers Indexed, RX

| | | | |
|---|---|---|---|
| NRX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . AND . $[A + (X)]$

And Registers Immediate, RI

| | | | |
|---|---|---|---|
| NRI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . AND . $\{A + (X)\}$

**INCLUSIVE-OR**

Or Registers, RR

| | | |
|---|---|---|
| ORR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1)$ . OR . $(R_2)$

Or Register Indexed, RX

| ORX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . OR . $[A + (X)]$

Or Register Immediate, RI

| ORI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . OR . $\{A + (X)\}$

## EXCLUSIVE-OR

Exclusive-Or Registers, RR

| XRR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_1)$ . EXOR . $(R_2)$

Exclusive-Or Regsiter Indexed, RX

| XRX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . EXOR . $[A + (X)]$

Exclusive-Or Register Immediate, RI

| XRI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow (R_1)$ . EXOR . $\{A + (X)\}$

## LOGICAL COMPARE

Logical Compare Registers, RR

| CLR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

$CC \leftarrow (R_1) : (R_2)$

Logical Compare Register Indexed, RX

| CLX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$CC \leftarrow (R_1) : [A + (X)]$

Logical Compare Register Immediate, RI

| CLI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$CC \leftarrow (R_1) : \{A + (X)\}$

Note: For all RI and RX formats, if $X = 0$ then $(X) \equiv 0$.

## COMPLEMENT

Complement Register, RR

| CMPL | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (\overline{R_1})$

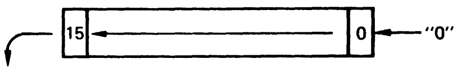
## SHIFT INSTRUCTIONS

All of the shift instructions are executed in the ALU portion of the computer. Only the carry and zero condition codes may be affected by these instructions.

### LOGICAL SHIFT

Shift Register Left, SF

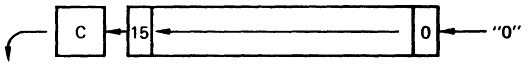
| SRL | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



Each bit in the register selected by $R_1$ is shifted left N places. The register is filled with zeroes.

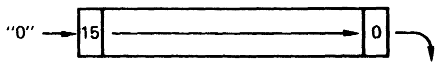
Shift Register Left with Carry, SF

| SRLC | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



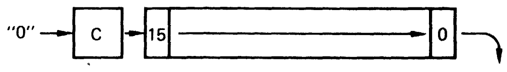
Shift Register Right, SF

| SRR | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



Shift Register Right with Carry, SF

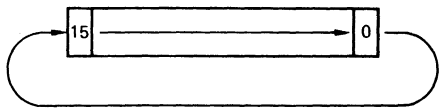
| SRRC | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



## ROTATE

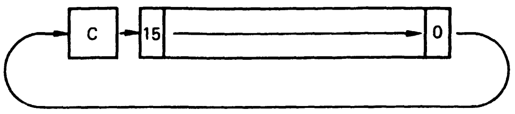
Rotate Register Right, SF

| RRR | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



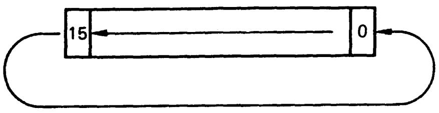
Rotate Register Right with Carry, SF

| RRRC | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



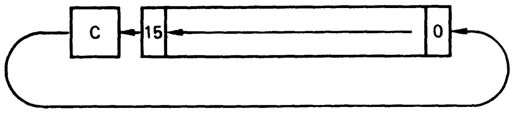
Rotate Register Left, SF

| RRL | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



Rotate Register Left with Carry, SF

| RRLC | $R_1$ | N |
|---|---|---|
| 15 8 | 7 4 | 3 0 |



## DATA MOVEMENT

The data movement instructions are used to transfer data from memory to a general purpose register, from a general purpose register to memory, and from register to register.

Load Register, RR

| LDR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

$R_1 \leftarrow (R_2)$

Load Register Indexed, RX

| LRX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow [A + (X)]$

Load Register Immediate, RI

| LRI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$R_1 \leftarrow A + (X)$

Note: For all RI and RX formats, if X = 0 then (X) ≡ 0.

Store Register Indexed, RX

| STX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

$\{A + (X)\} \leftarrow R_1$

## PROGRAM CONTROL

The program control instructions contain a 4-bit condition code selection field. The condition code selected must be true for the program control instruction to be executed. Eight of the condition codes are generated in the ALU and the remaining eight reside in the CCU. One of the condition codes, "1111," selects a constant logic level "1" condition providing an unconditional instruction. The condition codes will be discussed in some detail in the ALU and CCU sections.

### JUMP

Jump through Register, SF

| JTR | CC | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

CC = True; PC ← $(R_2)$
CC = False; PC ← (PC) + 1

Jump Indexed, RX

| JMPX | CC | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

CC = True; PC ← [A + (X)]
CC = False; PC ← (PC) + 1

Jump Immediate, RI

| JMPI | CC | X | A |
|---|---|---|---|
| 15 8 | 7. 4 | 3 0 | 15 0 |

CC = True; PC ← A + (X)
CC = False; PC ← (PC) + 1

### JUMP-TO-SUBROUTINE

Jump-to-Subroutine through Register, SF

| JSR | CC | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

CC = True; Stack ← (PC), PC ← $(R_2)$
CC = False; PC ← (PC) + 1

Jump-to-Subroutine Indexed, RX

| JSX | CC | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

CC = True; Stack ← (PC), PC ← [A + (X)]
CC = False; PC ← (PC) + 1

Jump-to-Subroutine Immediate, RI

| JSI | CC | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

CC = True; Stack ← (PC), PC ← A + (X)
CC = False; PC ← (PC) + 1

**RETURN**

Return-from-Subroutine, SF

| RTN | CC | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

CC = True; PC ← (Stack)
CC = False; PC ← (PC) + 1

## INPUT-OUTPUT

There are two types of input-output instructions: Data and Control. When either type of instruction is executed, an input-output device address will be loaded into the Memory Address Register and presented on the eight least significant address lines. Depending on the instruction type, the device address will be transferred from a register, come indirectly from a memory location, or be calculated from an immediate operand. In any case, the upper 8 bits of the address bus are not defined within the central processor design, so they may be defined and used by the I/O designer.

### DATA

Read Data into Register, RR

| READ | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

ADDR† ← ($R_2$)
$R_1$ ← (DB)††

†ADDR is the mnemonic for the address bus.
††DB is the mnemonic for the data bus.

Read Data into Memory Indexed, RX

| RDMX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← ($R_1$)
[A + (X)] ← (DB)

Read Data into Memory Immediate, RI

| RDMI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← ($R_1$)
A + (X) ← (DB)

Read Data into Register Immediate, RI

| RDRI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← A + (X)
$R_1$ ← (DB)

Note: For all RI and RX formats, if X = 0 then (X) ≡ 0.

Write Data from Register, RR

| WRITE | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

ADDR ← ($R_2$)
DB ← ($R_1$)

Write Data from Memory Indexed, RX

| WRMX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← ($R_1$)
DB ← [A + (X)]

Write Data Immediate, RI

| WRMI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← ($R_1$)
DB ← A + (X)

Write Data from Register Immediate, RI

| WRRI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← A + (X)
DB ← ($R_1$)

### CONTROL

Read Status, RR

| RSTS | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

ADDR ← ($R_2$)
$R_1$ ← (DB)

Read Status Immediate, RI

| RDSI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← A + (X)
$R_1$ ← (DB)

Write Command, RR

| WCMD | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

ADDR ← ($R_2$)
DB ← ($R_1$)

Write Command Immediate, RI

| WCMI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ADDR ← A + (X)
DB ← ($R_1$)

## EXECUTIVE

The executive instructions control the interrupt control unit and the direct memory access controller. They will cause addresses to be loaded and functions to be turned on or off. This set of instructions also provides control information to other functional systems in the processor.

### ALU

Set Carry, RR

| STC | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

CARRY ← "1"

Reset Carry, RR

| RTC | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

CARRY ← "0"

### ICU

Mask Interrupt Register, RR

| MIR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

MASKR ← ($R_2$)

Mask Interrupt Register Indexed, RX

| MIX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

MASKR ← [A + (X)]

Mask Interrupt Register Immediate, RI

| MII | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

MASKR ← A + (X)

Load Interrupt Service Vector, RR

| ISVR | $SR_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

SR ← ($R_2$) (The address in $R_2$ becomes the vector for the interrupt level specified in $SR_1$.)

Load Interrupt Service Vector Indexed, RX

| ISVX | $SR_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

SR ← [A + (X)]

Note: For all RI and RX formats, if X = 0 then (X) ≡ 0.

Load Interrupt Service Vector Immediate, RI

| ISVI | $SR_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

SR ← A + (X)

Clear Interrupt Request, RR

| CIR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

ICC ← ($R_2$) (Clears interrupt request for each level where $R_2$ contains a "1".)

Clear Interrupt Request Indexed, RX

| CIRX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ICC ← [A + (X)]

Clear Interrupt Request Immediate, RI

| CIRI | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

ICC ← A + (X)

Clear Interrupt Vector, RR

| CIV | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

V ← "0" (Clears last vector read.)

Read Status Register, RR

| RSR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

$R_1$ ← (S)

Load Status Register, RR

| LSR | $R_1$ | $R_2$ |
|---|---|---|
| 15 8 | 7 4 | 3 0 |

S ← ($R_2$)

Load Status Register Indexed, RX

| LSRX | $R_1$ | X | A |
|---|---|---|---|
| 15 8 | 7 4 | 3 0 | 15 0 |

S ← [A + (X)]

Load Status Register Immediate, RI

| | | | |
|---|---|---|---|
| LSRI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$S \leftarrow A + (X)$

Enable Interrupt System, RR

| | | |
|---|---|---|
| EIS | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

EIFF ← "1"

Disable Interrupt System, RR

| | | |
|---|---|---|
| DIS | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

EIFF ← "0"

Clear Interrupt Mask, RR

| | | |
|---|---|---|
| CIM | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

M ← "0"

Return-from-Interrupt, RR

| | | |
|---|---|---|
| RFI | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

Pop Stack; Enable Interrupts

Load Base Address Register, RR

| | | |
|---|---|---|
| LBAR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$BAR \leftarrow (R_2)$

Load Base Address Register Indexed, RX

| | | | |
|---|---|---|---|
| LBAX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$BAR \leftarrow [A + (X)]$

Load Base Address Register Immediate, RI

| | | | |
|---|---|---|---|
| LBAI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$BAR \leftarrow A + (X)$

Load Word Count Register, RR

| | | |
|---|---|---|
| LWCR | $R_1$ | $R_2$ |
| 15 8 | 7 4 | 3 0 |

$WC \leftarrow (R_2)$

Load Word Count Register Indexed, RX

| | | | |
|---|---|---|---|
| LWCX | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$WC \leftarrow [A + (X)]$

Load Word Count Register Immediate, RI

| | | | |
|---|---|---|---|
| LWCI | $R_1$ | X | A |
| 15 8 | 7 4 | 3 0 | 15 0 |

$WC \leftarrow A + (X)$

Note: For all RI and RX formats, if $X = 0$ then $(X) \equiv 0$.

# CHAPTER V
# ARITHMETIC LOGIC UNIT

A block diagram of the arithmetic logic unit is presented in Figure 3. All of the arithmetic and logic instructions are executed in the ALU, as well as effective address generation for the RX and RI classes of instructions. The ALU is a 16-bit parallel subsystem with a high-speed lookahead carry generator.

At the heart of the ALU subsystem is four 4-bit slice Am2901's, providing 16 general purpose registers, each of which is 16 bits wide. Also contained in the Am2901 is a parallel combinatorial logic array (to be discussed below) with the appropriate generate and propagate signals that are provided to the Am2902 lookahead carry unit. The Am2902 accepts the propagate and generate signals from each 4-bit slice and provides the carry-in signal to the next stage and an anticipatory signal to successive stages. The generate and propagate signals that are output from the Am2902 provide information that can be decoded for compare functions to show the relative magnitude between the two values. (Compare does not work for 2's complement numbers.)

To allow for various shift methods, an external set of multiplexers is provided to introduce long and short shifts and rotates and to provide a way to link carry and/or logical "0" or "1" into the shift path.

The signals generated in the Am2901 showing the status of the ALU plus the status and condition signals that are derived in the ALU are stored into a register each time the ALU is used. A multiplexer is used to select one-of-eight test conditions for presentation to the computer control unit for the purpose of providing information on which a decision is made to execute a conditional jump.

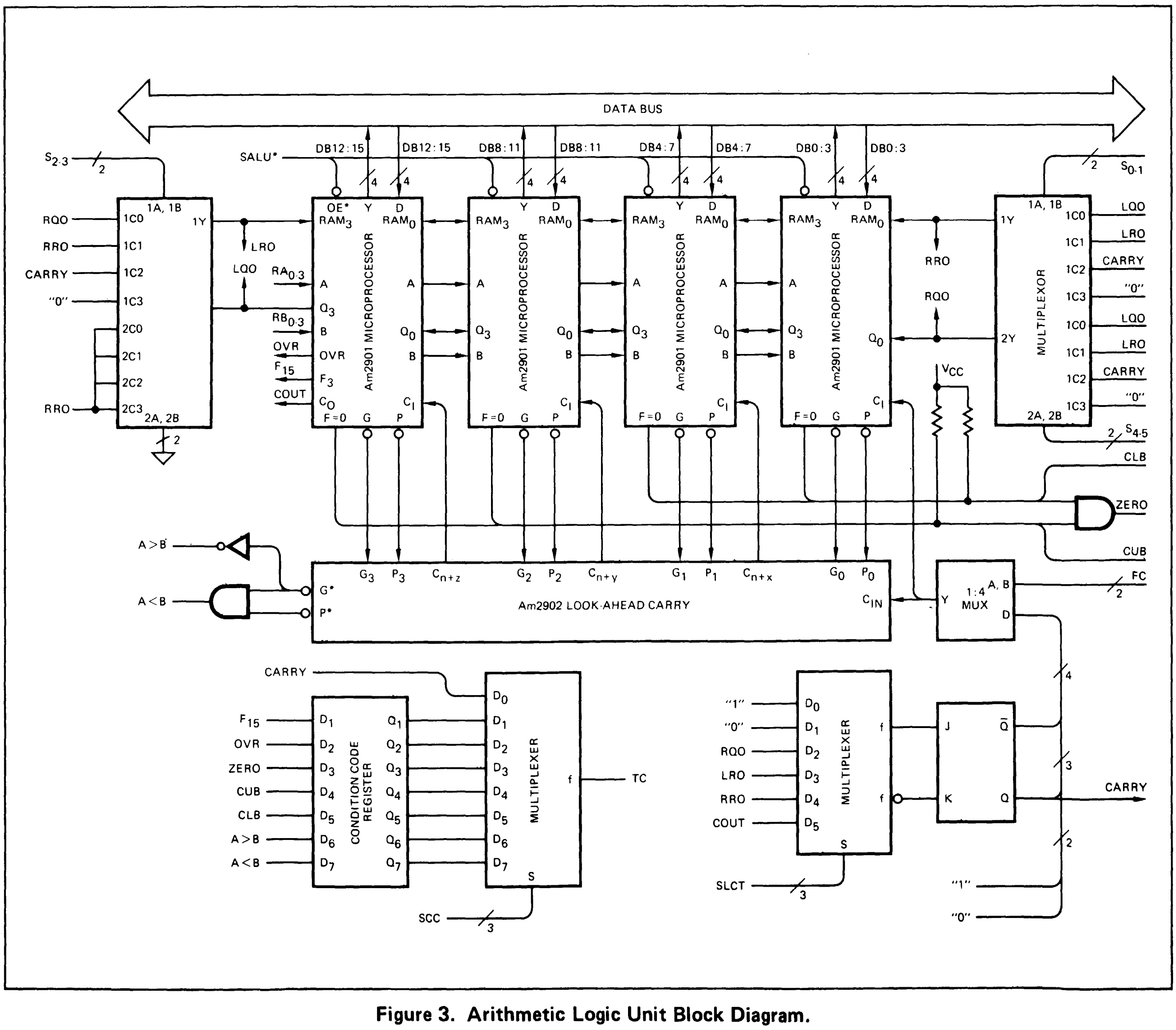


Figure 3. Arithmetic Logic Unit Block Diagram.

Additionally, logic is provided to present the proper signal to the carry flip-flop for storage during the next clock cycle. The output of the carry flip-flop is gated with a signal Force Carry-In which is controlled from microcode.

## THE Am2901

A detailed block diagram of the bipolar microprogrammable microprocessor structure is shown in Figure 4. The circuit is a four-bit slice cascadable to any number of bits. Therefore, all data paths within the circuit are four bits wide. The two key elements in the Figure 4 block diagram are the 16-word by 4-bit, 2-port RAM and the high-speed ALU.

Data in any of the 16 words of the Random Access Memory (RAM) can be read from the A-port of the RAM as controlled by the four-bit A address field input. Likewise, data in any of the 16 words of the RAM as defined by the B address field input can be simultaneously read from the B-port of the RAM. The same code can be applied to the A select field and B select field; in which case, the identical file data will appear at both the RAM A-port and B-port outputs simultaneously.

When enabled by the RAM write enable (RAM EN), new data is always written into the file (word) defined by the B address field of the RAM. The RAM data input field is driven by a three-input multiplexer. The configuration is used to shift the ALU output data (F), if desired.

The RAM A-port data outputs and RAM B-port data outputs drive separate four-bit latches. These latches hold the RAM data while the clock input is LOW. This eliminates any possible race conditions that could occur while new data is being written into the RAM.

The high-speed Arithmetic Logic Unit (ALU) can perform three binary arithmetic and five logic operations on the two 4-bit input words R and S. The R input field is driven from a 2-input multiplexer, while the S input field is driven from a 3-input multiplexer. Both multiplexers also have an inhibit capability; that is, no data is passed. This is equivalent to a "zero" source operand.

Referring to Figure 4, the ALU R input multiplexer has the RAM A-port and the direct data inputs (D) connected as inputs. Likewise, the ALU S input multiplexer has the RAM A-port, the RAM B-port and the Q register connected as inputs.

This multiplexer scheme gives the capability of selecting various parts of the A, B, D, Q and "0" inputs as source operands to the ALU. These five inputs, when taken two at a time, result in ten possible combinations of source operand pairs. These combinations include AB, AD, AQ, A0, BD, BQ, B0, DQ, D0 and Q0. It is apparent that AD, AQ and A0 are somewhat redundant with BD, BQ AND B0 in that if the A address and B address are the same, the identical function results. Thus, there are only seven completely non-redundant source operand pairs for the ALU. The Am2901 microprocessor implements eight of these pairs. The microinstruction inputs used to select the ALU source operands are the $I_0$, $I_1$ and $I_2$ inputs. The definitions of $I_0$, $I_1$ and $I_2$ for the eight source-operand combinations are as shown in Figure 5. Also shown is the octal code for each selection.

The two source operands not fully described as yet are the D input and Q input. The D input is the four-bit wide direct data field input. This port is used to insert all data into the working registers inside the device. Likewise, this input can be used in the ALU to modify any of the internal data files. The Q register is a separate four-bit file intended primarily for multiplication and division routines, but it can also be used as an accumulator or holding register for some applications.

The ALU itself is a high-speed arithmetic/logic operator capable of performing three binary arithmetic and five logic functions. The $I_3$, $I_4$, and $I_5$ microinstruction inputs are used to select the ALU function. The definition of these inputs is shown in Figure 6. The octal code is also shown for reference. The normal technique for cascading the ALU of several devices is in a lookahead carry mode. Carry generate, G, and carry propagate, P, are outputs of the device for use with a carry lookahead generator such as the Am2902. A carry-out, $C_{n+4}$, is also generated and is available as an output for use as the carry flag in a status register. Both carry-in ($C_n$) and carry-out ($C_{n+4}$) are active HIGH.

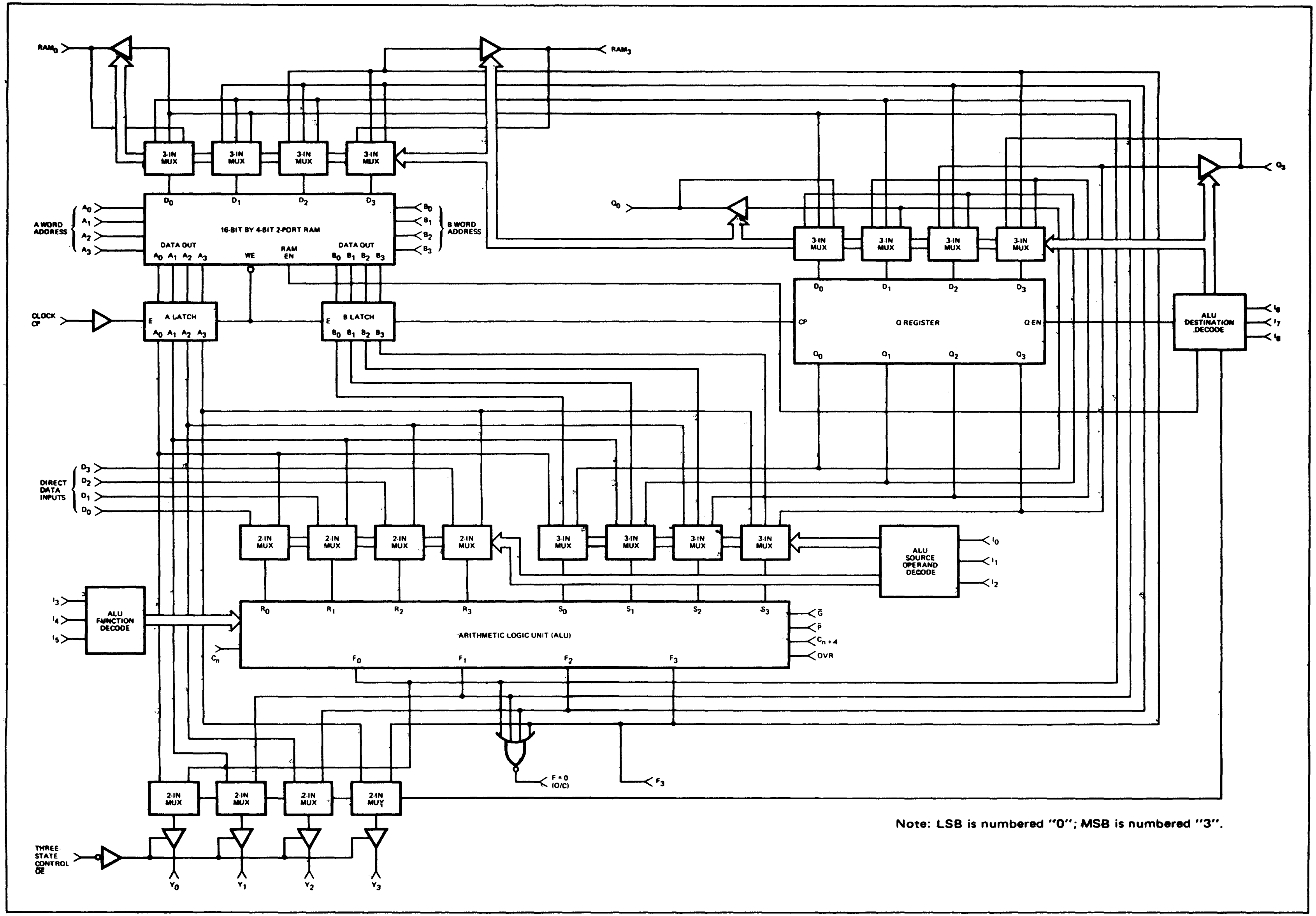
The ALU has three other status-oriented outputs. These are $F_3$, F = 0, and overflow (OVR). The $F_3$ output is the most significant (sign) bit of the ALU and can be used to determine positive or negative results without enabling the three-state data outputs. $F_3$ is non-inverted with respect to the sign bit output, $Y_3$. The F = 0 output is used for zero detect. It is an open-collector output and can be wire OR'd between microprocessor slices. F = 0 is HIGH when all F outputs are LOW. The overflow output (OVR) is used to flag arithmetic operations that exceed the available two's complement number range. The overflow output (OVR) is HIGH when overflow exists, that is, when $C_{n+3}$ and $C_{n+4}$ are not the same polarity.

The ALU data output is routed to several destinations. It can be a data output of the device and it can also be stored in the RAM or the Q register. Eight possible combinations of ALU destination functions are available as defined by the $I_6$, $I_7$, and $I_8$ microinstruction inputs. These combinations are shown in Figure 7.

The four-bit data output field (Y) features three-state outputs and can be directly bus organized. An output control (OE) is used to enable the three-state outputs. When OE is HIGH, the Y outputs are in the high-impedance state.

A two-input multiplexer is also used at the data output such that either the A-port of the RAM or the ALU outputs (F) are selected at the device Y outputs. This selection is controlled by the $I_6$, $I_7$, and $I_8$ microinstruction inputs. Refer to Figure 4 for the selected output for each microinstruction code combination.

As was discussed previously, the RAM inputs are driven from a three-input multiplexer. This allows the ALU outputs to be entered non-shifted, shifted up one position (X2) or shifted down one position (÷ 2). The shifter has two ports; one is labeled $RAM_0$ and the other is labeled $RAM_3$. Both of these ports consist of a buffer-driver with a three-state output and an input to the multiplexer. Thus, in the shift-up mode, the RO buffer is enabled and the RI multiplexer input is enabled. Likewise, in the shift down mode, the LO buffer and LI input are enabled. In the no-shift mode, both the LO and RO buffers are in the high-impedance state, and the multiplexer inputs are not selected. This shifter is controlled from the $I_6$, $I_7$, and $I_8$ microinstruction inputs as defined in Figure 7. Similarly, the



Note: LSB is numbered "0"; MSB is numbered "3".

Figure 4. Detailed Am2901 Microprocessor Block Diagram.

| MICRO CODE | | | | ALU SOURCE OPERANDS | |
|---|---|---|---|---|---|
| $I_2$ | $I_1$ | $I_0$ | Octal Code | R | S |
| L | L | L | 0 | A | Q |
| L | L | H | 1 | A | B |
| L | H | L | 2 | 0 | Q |
| L | H | H | 3 | 0 | B |
| H | L | L | 4 | 0 | A |
| H | L | H | 5 | D | A |
| H | H | L | 6 | D | Q |
| H | H | H | 7 | D | 0 |

**Figure 5. ALU Source Operand Control.**

| MICRO CODE | | | | ALU Function | Symbol |
|---|---|---|---|---|---|
| $I_5$ | $I_4$ | $I_3$ | Octal Code | | |
| L | L | L | 0 | R Plus S | $R + S$ |
| L | L | H | 1 | S Minus R | $S - R$ |
| L | H | L | 2 | R Minus S | $R - S$ |
| L | H | H | 3 | R OR S | $R \vee S$ |
| H | L | L | 4 | R AND S | $R \wedge S$ |
| H | L | H | 5 | $\bar{R}$ AND S | $\bar{R} \wedge S$ |
| H | H | L | 6 | R EX-OR S | $R \forall S$ |
| H | H | H | 7 | R EX-NOR S | $\overline{R \forall S}$ |

**Figure 6. ALU Function Control.**

Q register is driven from a three-input multiplexer. In the no-shift mode, the multiplexer enters the ALU data into the Q register. In either the shift-up or shift-down mode, the multiplexer selects the Q register data appropriately shifted up or down. The Q shifter also has two ports; one is labeled $Q_0$ and the other is $Q_3$. The operation of these two ports is similar to the RAM shifter and is also controlled from $I_6$, $I_7$, and $I_8$ as shown in Figure 7.

The clock input to the Am2901 controls the RAM, the Q register, and the A and B data latches. When enabled, data is clocked into the Q register on the LOW-to-HIGH transition of the clock. When the clock input is HIGH, the A and B latches are open and will pass whatever data is present at the RAM outputs. When the clock input is LOW, the latches are closed and will retain the last data entered. If the RAM-EN is enabled, new data will be written into the RAM file (word) defined by the B address field when the clock input is LOW.

There are eight source operand pairs available to the ALU as selected by the $I_0$, $I_1$, and $I_2$ instruction inputs. The ALU can perform eight functions, five logic and three arithmetic. The $I_3$, $I_4$, and $I_5$ instruction inputs control the function selection. The carry input, $C_n$, also affects the ALU results when in the arithmetic mode. The $C_n$ input has no affect in the logic mode. When $I_0$ through $I_5$ and $C_n$ are viewed together, the matrix of Figure 8 results. This matrix fully defines the ALU/source operand function for each state.

The ALU function can also be examined on a "task" basis; i.e., add, subtract, AND, OR, etc. In the arithmetic mode, the carry will affect the function performed while in the logic mode, the carry will have no bearing on the ALU output. Figure 9 defines the various logic operations that the Am2901 can perform, and Figure 10 shows the arithmetic functions of the device. Both carry-in LOW ($C_n = 0$) and carry-in HIGH ($C_n = 1$) are defined in these operations.

| MICRO CODE | | | | RAM FUNCTION | | Q-REG. FUNCTION | | Y OUTPUT | RAM SHIFTER | | Q SHIFTER | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I_8$ | $I_7$ | $I_6$ | Octal Code | Shift | Load | Shift | Load | | $RAM_0$ | $RAM_3$ | $Q_0$ | $Q_3$ |
| L | L | L | 0 | X | NONE | NONE | $F \rightarrow Q$ | F | X | X | X | X |
| L | L | H | 1 | X | NONE | X | NONE | F | X | X | X | X |
| L | H | L | 2 | NONE | $F \rightarrow B$ | X | NONE | A | X | X | X | X |
| L | H | H | 3 | NONE | $F \rightarrow B$ | X | NONE | F | X | X | X | X |
| H | L | L | 4 | DOWN | $F/2 \rightarrow B$ | DOWN | $Q/2 \rightarrow Q$ | F | $F_0$ | $IN_3$ | $Q_0$ | $IN_3$ |
| H | L | H | 5 | DOWN | $F/2 \rightarrow B$ | X | NONE | F | $F_0$ | $IN_3$ | $Q_0$ | X |
| H | H | L | 6 | UP | $2F \rightarrow B$ | UP | $2Q \rightarrow Q$ | F | $IN_0$ | $F_3$ | $IN_0$ | $Q_3$ |
| H | H | H | 7 | UP | $2F \rightarrow B$ | X | NONE | F | $IN_0$ | $F_3$ | X | $Q_3$ |

X= Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.
B= Register Addressed by B inputs.
Up is toward MSB, Down is toward LSB.

**Figure 7. ALU Destination Control.**

| OCTAL $I_{543}$ | $I_{210}$ OCTAL | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
|  | ALU Source / ALU Function | A, Q | A, B | O, Q | O, B | O, A | D, A | D, Q | D, O |
| 0 | $C_n$ = L R Plus S | $A+Q$ | $A+B$ | $Q$ | $B$ | $A$ | $D+A$ | $D+Q$ | $D$ |
|  | $C_n$ = H | $A+Q+1$ | $A+B+1$ | $Q+1$ | $B+1$ | $A+1$ | $D+A+1$ | $D+Q+1$ | $D+1$ |
| 1 | $C_n$ = L S Minus R | $Q-A-1$ | $B-A-1$ | $Q-1$ | $B-1$ | $A-1$ | $A-D-1$ | $Q-D-1$ | $-D-1$ |
|  | $C_n$ = H | $Q-A$ | $B-A$ | $Q$ | $B$ | $A$ | $A-D$ | $Q-D$ | $-D$ |
| 2 | $C_n$ = L R Minus S | $A-Q-1$ | $A-B-1$ | $-Q-1$ | $-B-1$ | $-A-1$ | $D-A-1$ | $D-Q-1$ | $D-1$ |
|  | $C_n$ = H | $A-Q$ | $A-B$ | $-Q$ | $-B$ | $-A$ | $D-A$ | $D-Q$ | $D$ |
| 3 | R OR S | $A \vee Q$ | $A \vee B$ | $Q$ | $B$ | $A$ | $D \vee A$ | $D \vee Q$ | $D$ |
| 4 | R AND S | $A \wedge Q$ | $A \wedge B$ | 0 | 0 | 0 | $D \wedge A$ | $D \wedge Q$ | 0 |
| 5 | $\bar{R}$ AND S | $\bar{A} \wedge Q$ | $\bar{A} \wedge B$ | $Q$ | $B$ | $A$ | $\bar{D} \wedge A$ | $\bar{D} \wedge Q$ | 0 |
| 6 | R EX-OR S | $A \veebar Q$ | $A \veebar B$ | $Q$ | $B$ | $A$ | $D \veebar A$ | $D \veebar Q$ | $D$ |
| 7 | R EX-NOR S | $\overline{A \veebar Q}$ | $\overline{A \veebar B}$ | $\bar{Q}$ | $\bar{B}$ | $\bar{A}$ | $\overline{D \veebar A}$ | $\overline{D \veebar Q}$ | $\bar{D}$ |

+ = Plus; – = Minus; $\vee$ = OR; $\wedge$ = AND; $\veebar$ = EX-OR

**Figure 8. Source Operand and ALU Function Matrix.**

## DETAILED LOGIC

Data is input to the ALU from the data bus via the direct inputs, D. The instruction field $I_{6-8}$ will determine whether the destination of the data is one of the 16 general purpose registers selected by the register field $R_B$, the temporary register Q, or whether no storage action is to be taken. The remainder of the instruction field, $I_{0-5}$, will determine which two data sources will be selected for operation by the combinatorial arithmetic and logic array and the function to be performed by the array.

Data is output from the ALU through the Y outputs. The Y outputs have a high-impedance, three-state condition that is controlled by the signal OE*; the signal SALU* which enables OE* comes from the CCU and will be discussed with the bus control concepts. The source of the output data and any operations that are performed on it are controlled by $R_A$, $R_B$, and $I_{0-8}$.

During any operation in the Am2901, the data or operational status signals are monitored. The signals OVR, overflow, and $F_{15}$, the sign bit, are important status bits for arithmetic operations and tests. There is a function equal to zero detector in the Am2901 with an open-collector output. The two zero detectors from the least significant byte (Figure 11) FO1 and FO2, are tied together with a pull-up resistor and form the signal CLB, compare lower byte. Similarly, the two zero detectors from the upper byte, FO3 and FO4, are tied together to form CUB, compare upper byte. In turn, CLB and CUB are AND'd together to form the signal ZERO; the result of the previous operation was a logical or arithmetic "0" for the entire word.

The generate and propagate signals, G* and P*, originating from the most significant end of the Am2902 lookahead carry generator are used to provide arithmetic comparisons between two words. If the arithmetic function A–B is performed in the Am2901, the signal G* may be inverted to derive the condition A>B; if the result G equals "1," then A is greater than B. By taking the logic AND of P* and G*, the signal A<B may be derived; this signal is also positive true. If A is arithmetically equivalent to B, the signal ZERO will be true.

(These tests are for *magnitude only* comparisons. To compare signed, two's complement numbers, the logic would be modified to use $F_{15}$ and ZERO.)

The seven basic or derived signals, $F_{15}$, OVR, ZERO, CUB, CLB, A>B, and A<B are presented to an eight-bit register, Am25LS273 or equivalent (Figure 11). This register is loaded everytime the ALU is selected for an operation by the CCU; the ALU clock, CP1, is generated in the CCU. At the time the computer is turned on or a master reset operation is manually performed, the signal TOCLR*, turn-on clear, is generated to initialize the selected storage devices in the system to an appropriate known state. This initialization places the control unit in a known state, as well as the various signals upon which CCU decisions are made.

The output of the condition code register is presented to an eight-input multiplexer (Figure 11), an Am25LS251, with three-state outputs. In addition to the seven signals from the condition code register, the condition of the carry flip-flop, CARRY, is also provided to the multiplexer. The microcontrol signals $C_{3-6}$ are used to select an ALU condition code for conditional branching in microcode or machine code.

The carry generator and flip-flop logic are shown in Figure 12. The input to the carry flip-flop is selected from one-of-six sources by an Am25LS151 multiplexer. The flip-flop may be

| Octal $I_{543}, I_{210}$ | Group | Function |
|---|---|---|
| 4 0 | AND | $A \wedge Q$ |
| 4 1 | | $A \wedge B$ |
| 4 5 | | $D \wedge A$ |
| 4 6 | | $D \wedge Q$ |
| 3 0 | OR | $A \vee Q$ |
| 3 1 | | $A \vee B$ |
| 3 5 | | $D \vee A$ |
| 3 6 | | $D \vee Q$ |
| 6 0 | EX-OR | $A \veebar Q$ |
| 6 1 | | $A \veebar B$ |
| 6 5 | | $D \veebar A$ |
| 6 6 | | $D \veebar Q$ |
| 7 0 | EX-NOR | $\overline{A \veebar Q}$ |
| 7 1 | | $\overline{A \veebar B}$ |
| 7 5 | | $\overline{D \veebar A}$ |
| 7 6 | | $\overline{D \veebar Q}$ |
| 7 2 | INVERT | $\overline{Q}$ |
| 7 3 | | $\overline{B}$ |
| 7 4 | | $\overline{A}$ |
| 7 7 | | $\overline{D}$ |
| 6 2 | PASS | Q |
| 6 3 | | B |
| 6 4 | | A |
| 6 7 | | D |
| 3 2 | PASS | Q |
| 3 3 | | B |
| 3 4 | | A |
| 3 7 | | D |
| 4 2 | "ZERO" | 0 |
| 4 3 | | 0 |
| 4 4 | | 0 |
| 4 7 | | 0 |
| 5 0 | MASK | $\overline{A} \wedge Q$ |
| 5 1 | | $\overline{A} \wedge B$ |
| 5 5 | | $\overline{D} \wedge A$ |
| 5 6 | | $\overline{D} \wedge Q$ |

**Figure 9. ALU Logic Mode Functions ($C_n$ Irrelevant).**

| Octal | $C_n$ = 0 (Low) | | $C_n$ = 1 (High) | |
|---|---|---|---|---|
| $I_{543}, I_{210}$ | Group | Function | Group | Function |
| 0 0 | ADD | A+Q | ADD plus one | A+Q+1 |
| 0 1 | | A+B | | A+B+1 |
| 0 5 | | D+A | | D+A+1 |
| 0 6 | | D+Q | | D+Q+1 |
| 0 2 | PASS | Q | Increment | Q+1 |
| 0 3 | | B | | B+1 |
| 0 4 | | A | | A+1 |
| 0 7 | | D | | D+1 |
| 1 2 | Decrement | Q–1 | PASS | Q |
| 1 3 | | B–1 | | B |
| 1 4 | | A–1 | | A |
| 2 7 | | D–1 | | D |
| 2 2 | 1's Comp. | –Q–1 | 2's Comp. (Negate) | –Q |
| 2 3 | | –B–1 | | –B |
| 2 4 | | –A–1 | | –A |
| 1 7 | | –D–1 | | –D |
| 1 0 | Subtract (1's Comp) | Q–A–1 | Subtract (2's Comp) | Q–A |
| 1 1 | | B–A–1 | | B–A |
| 1 5 | | A–D–1 | | A–D |
| 1 6 | | Q–D–1 | | Q–D |
| 2 0 | | A–Q–1 | | A–Q |
| 2 1 | | A–B–1 | | A–B |
| 2 5 | | D–A–1 | | D–A |
| 2 6 | | D–Q–1 | | D–Q |

**Figure 10. ALU Arithmetic Mode Functions.**

set to "1" or cleared to "0" by selecting either $V_{CC}$ or ground. For arithmetic operations, the source of data for the carry flip-flop will be the carry-out signal, COUT, from the most significant Am2901. For a shifting operation, the most significant bit from the RAM-Register, LRO, may be selected. If no change in the carry status is desired in the carry signal itself, CARRY may be selected. Carry signal selection is accomplished through the microcode signals $C_{0-2}$. The ALU clock, CP1, is used to clock-in the new carry data. For some ALU functions, it is desirable to force the carry-in signal to "1" to "0." The microcode signals FC0 and FC1 accomplish this function. The signal CIN is the carry-in signal that is presented to the Am2902 lookahead carry generator and the least significant Am2901.

The detailed logic for the shift linkages is shown in Figure 13. The most and least significant RAM shifter bits are LRO and RRO. And the most and least significant Q-Register bits are LQO and RQO.

The potential logic shift linkages are shown diagrammatically in Figure 14, while the rotate shift linkages are shown in Figure 15. Because there are a large number of instruction/signal combinations to keep track of, the information has been tabulated in Figure 16. By diagramming the data flow through the registers for each instruction, a table may be compiled by instruction type showing the input port and the signal name. It can be seen from the Am2901 destination control signals $I_{678}$ that whenever there is a shift right that $I_7$ is at logic zero. The RAM and Q-LSB multiplexers may therefore be enabled by $I_7^*$ and the RAM and Q-MSB multiplexers may be enabled by $I_7$. Because the only instances when the Q-MSB multiplexer is needed is for long right shifts or rotates, the select signals may be constant as shown. The remaining select bits, $S_0$ to $S_5$, control the remainder of the shift linkage. Further, since the MSB and LSB multiplexers are not used simultaneously, $S_0$, and $S_1$ may be common to $S_2$ and $S_3$, or $S_2$ and $S_3$ may be common to $S_4$ and $S_5$.

## MICROPROGRAM CONTROL OF THE ALU

A summary of the ALU microinstruction control signals is presented in worksheet form in Figure 17. A work sheet of this type will prove very helpful during the design and test of the ALU as a subsystem. Also, the worksheet provides a simple mechanism for algorithm development on a subsystem level. Space is provided for line number and the instruction description, and a column is provided for each control signal that is not derived in the ALU but rather is sourced from the CCU or another control source.

The signal TOCLR* is generated automatically when power is turned on in the computer, or manually when a master reset is generated from the control panel. This signal is used to initialize the entire system from a known starting point; all SSI and MSI storage elements are directly cleared or preset by this signal.

CP1 is the Am2901 clock. This signal is generated in the computer control unit whenever the ALU instructions are being executed. All storage elements operate on the LOW-to-HIGH transition of CP1. Whenever data is to originate in the ALU and be transferred elsewhere, the signal SALU* is true and causes the data selected in the ALU to be impressed on the data bus (Figure 3).

The RAM selection fields A and B may originate from the Instruction Register or from the microinstruction field. Therefore, for most instructions the hardware designer does not have to concern himself with the actual contents of the signals $RA_{0-3}$ and $RB_{0-3}$, but rather the fact that they are used. The

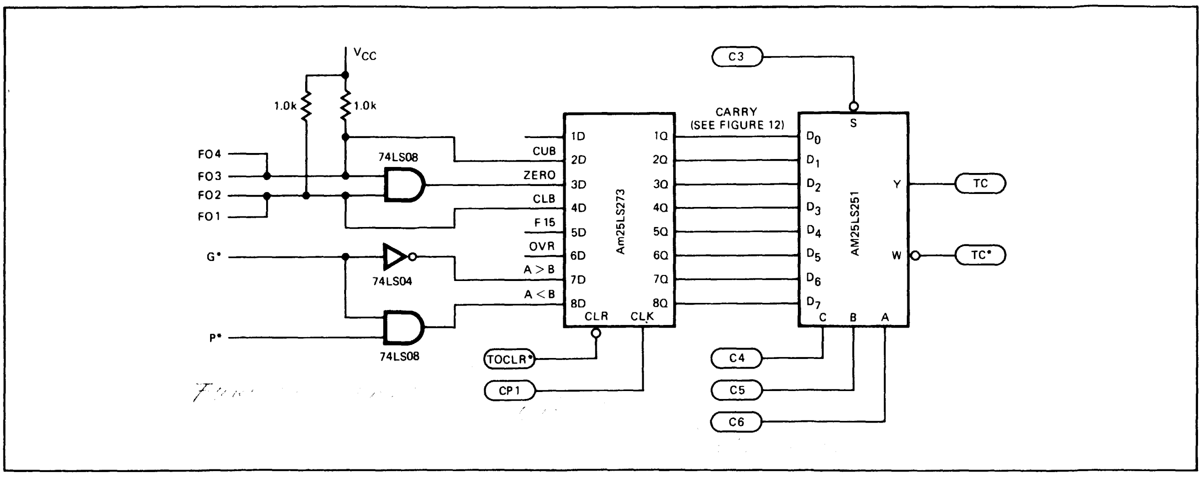


Figure 11. Detailed Logic Diagram Condition Code Generator, Register and Multiplexer.

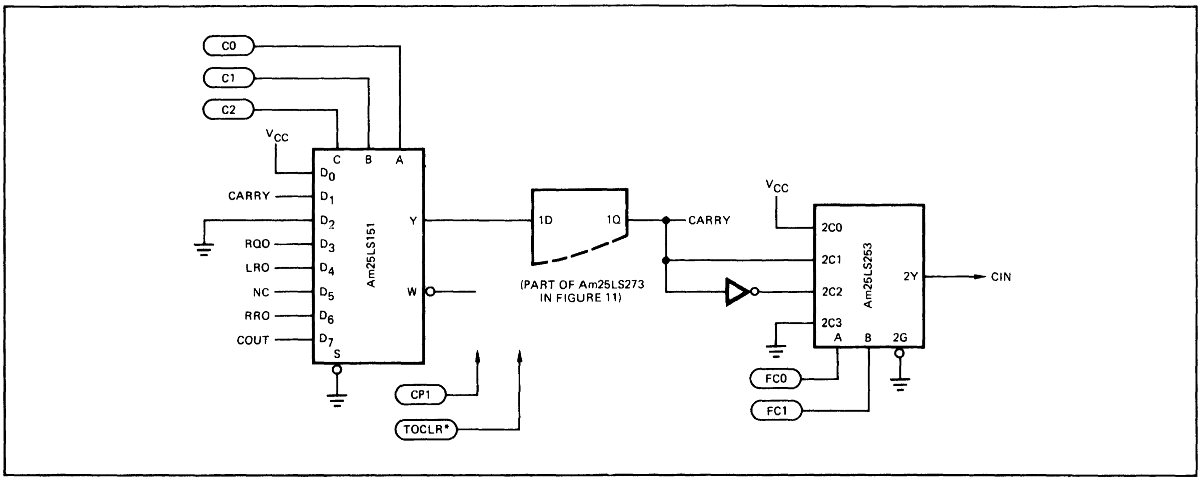


Figure 12. Detailed Logic Diagram Carry Generator and Flip-Flop.

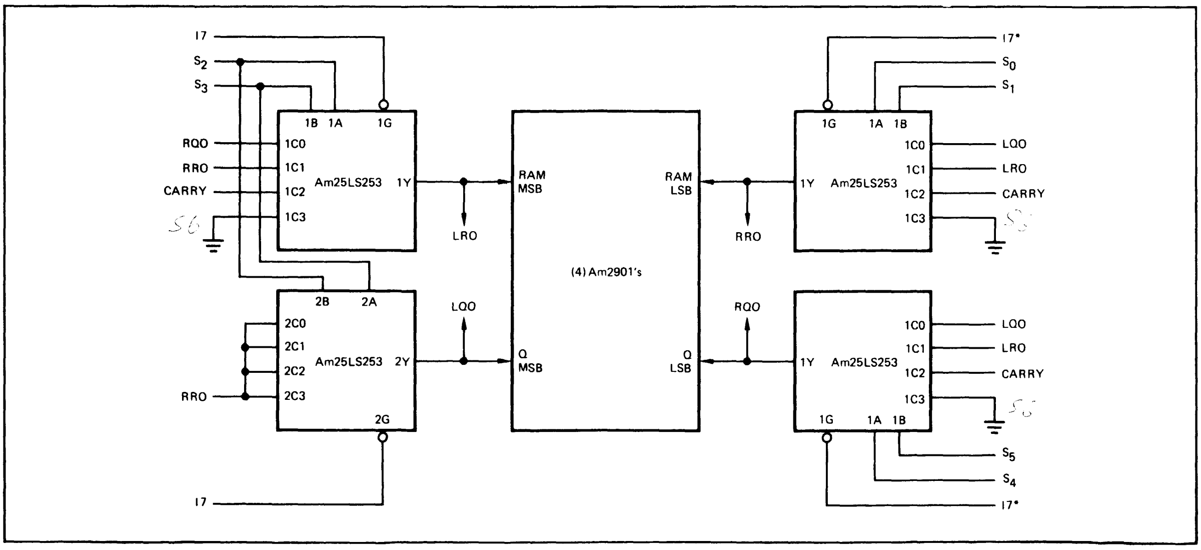


Figure 13. Detailed Logic Diagram Shift Linkages.

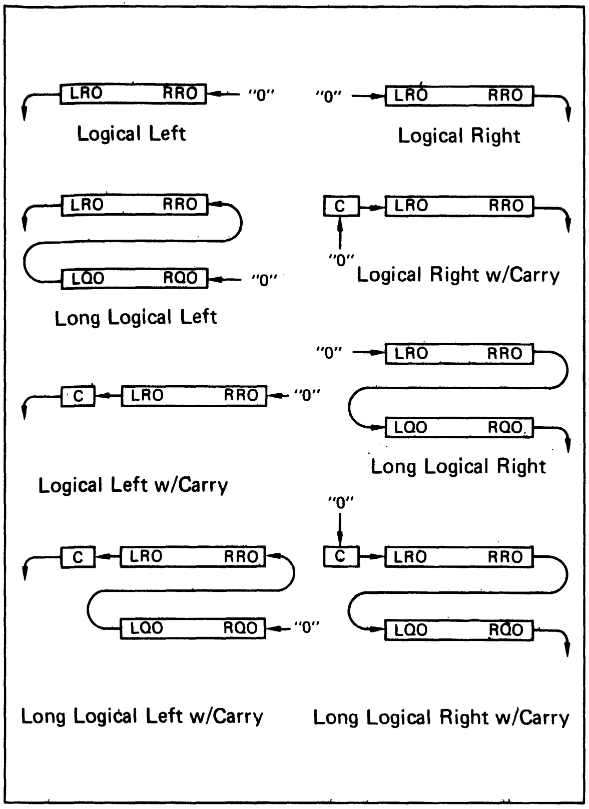


**Figure 14. Logical Shift Linkages.**

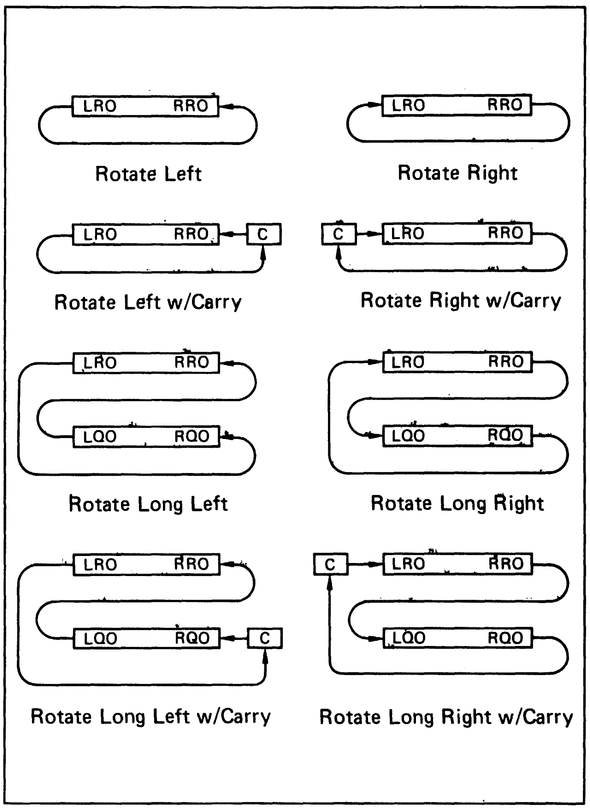


**Figure 15. Rotate Shift Linkages.**

| INSTRUCTION | $I_{678}$ | SHIFT MUX. OUTPUT | | | | SHIFT LINKAGE CONTROLS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | RAM-LSB | | RAM-MSB | | Q-LSB | |
| | | LRO | RRO | LQO | RQO | S0 | S1 | S2 | S3 | S4 | S5 |
| Shift Right Short W/O Carry | 101 | "0" | Z | RRO | Z | X | X | 1 | 1 | X | X |
| Shift Right Short W/ Carry | 101 | C | Z | RRO | Z | X | X | 0 | 1 | X | X |
| Shift Right Long W/O Carry | 001 | "0" | Z | RRO | Z | X | X | 1 | 1 | X | X |
| Shift Right Long W/ Carry | 001 | C | Z | RRO | Z | X | X | 0 | 1 | X | X |
| Shift Left Short W/O Carry | 111 | Z | "0" | Z | X | 1 | 1 | X | X | X | X |
| Shift Left Short W/ Carry | 111 | Z | "0" | Z | X | 1 | 1 | X | X | X | X |
| Shift Left Long W/O Carry | 011 | Z | LQO | Z | "0" | 0 | 0 | X | X | 1 | 1 |
| Shift Left Long W/ Carry | 011 | Z | LQO | Z | "0" | 0 | 0 | X | X | 1 | 1 |
| Rotate Right Short W/O Carry | 101 | RRO | Z | RRO | Z | X | X | 1 | 0 | X | X |
| Rotate Right Short W/ Carry | 101 | C | Z | RRO | Z | X | X | 0 | 1 | X | X |
| Rotate Right Long W/O Carry | 001 | RQO | Z | RRO | Z | X | X | 0 | 0 | X | X |
| Rotate Right Long W/ Carry | 001 | C | Z | RRO | Z | X | X | 0 | 1 | X | X |
| Rotate Left Short W/O Carry | 111 | Z | LRO | Z | X | 1 | 0 | X | X | X | X |
| Rotate Left Short W/ Carry | 111 | Z | C | Z | X | 0 | 1 | X | X | X | X |
| Rotate Left Long W/O Carry | 011 | Z | X | Z | LRO | 0 | 0 | X | X | 1 | 0 |
| Rotate Left Long W/ Carry | 011 | Z | X | Z | C | 0 | 0 | X | X | 0 | 1 |

Z = HIGH Impedance State
X = Don't Care

[illegible]

**Figure 16. Shift Linkage Instruction, Multiplexer and Control Functions.**

| MICROCODE LINE NO. | ALU INSTRUCTION | TOCLR* | CP1 | SALU* | RAM SELECTION FIELDS A & B | | | | | | | | Am2901 INSTRUCTION CONTROL | | | | | | | | | CARRY CONTROL | | | | | COND. SELECT | | | | SHIFT LINKAGE CONTROL | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | RA0 | RA1 | RA2 | RA3 | RB0 | RB1 | RB2 | RB3 | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | C0 | C1 | C2 | FC0 | FC1 | C3 | C4 | C5 | C6 | S0 | S1 | S2 | S3 | S4 | S5 |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 17. ALU Microinstruction Control Fields.

| MICROCODE LINE NO. | ALU INSTRUCTION | TOCLR* | CP1 | SALU* | RAM SELECTION FIELDS A & B | | | | | | | | Am2901 INSTRUCTION CONTROL | | | | | | | | | CARRY CONTROL | | | | | COND. SELECT | | | | SHIFT LINKAGE CONTROL | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | RA0 | RA1 | RA2 | RA3 | RB0 | RB1 | RB2 | RB3 | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | C0 | C1 | C2 | FC0 | FC1 | C3 | C4 | C5 | C6 | S0 | S1 | S2 | S3 | S4 | S5 |
| 1 | Add Registers | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | Add Register to Memory | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | Add Registers with Carry | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | Arithmetic Compare Registers | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | Two's Complement Register | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | Increment Register | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | Exclusive-OR Registers | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | Logical Compare Registers | 1 | ↑ | 1 | | | | | | | | | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | Shift Register Left (Up) | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | X | X | X | X | X | X | 1 | 1 | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | Shift Register Left (Up) with Carry | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | 1 | 1 | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | Rotate Register Left (Up) | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | X | X | X | X | X | X | 1 | 0 | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | Load Register, RR | 1 | ↑ | 1 | | | | | | | | | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | Load Register, RX | 1 | ↑ | 1 | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | Store Register (R) | 1 | 0 | 0 | | | | | | | | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | Store Register (Q) | 1 | 0 | 0 | | | | | | | | | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | Set Carry | 1 | ↑ | 1 | | | | | | | | | X | X | X | X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X |

Figure 18. ALU Micro Instruction Control Examples.

carry control, condition select, and shift linkage control fields were all discussed in detail above.

Sixteen of the instructions described in Section 5 that affect the ALU are presented here (Figure 18) in microcode form. It is important to remember that the microcode presented in this table is not complete in itself; it is merely that portion of the code that controls the execution of the ALU.

All of these instructions are relatively straight-forward in that they only require a single machine cycle except for Add Register to Memory, Line 2. Because one of the operands is coming from memory and the other operand is coming from a general purpose register, the sum of the two operands must be temporarily stored in the Q Register. On the next cycle, then, a Store Q-Register, Line 15, must be executed to return the sum to the addressed memory location.

Note also the difference between the RR and RX Load Register instructions of Lines 12 and 13, and the fact that the ALU does not receive a clock pulse, CP1, during the execution of the Store Register instructions of Lines 14 and 15.

If the reader does not have experience in the design of micro-controlled processors, it will be useful to rewrite the micro-instructions for some of the ALU instructions that were presented in Figure 18. Figure 17 will provide a good work-sheet.

# CHAPTER VI PROGRAM CONTROL UNIT

The program control unit is built into the computer to maintain the address of the next instruction to be executed, the next instruction operand to be fetched, or to control program modification. Program modification consists of controlling jumps, jumps-to-subroutines, and returns-from-subroutine; a process also called program linkage.

The address of the next instruction to be executed is maintained in the Program Counter (or Program Counter Register). The CCU fetches the first word of the instruction and loads it into the Instruction Register which resides in the CCU. At the same time that the instruction load occurs, the PC register is incremented, thereby pointing to the next instruction or the second word of the current instruction. If the address(es) of the instruction operand(s) have to be calculated, the calculation will be performed in the ALU. The calculated address is called an Effective Address, EA, and will be loaded into the MAR for the operand fetch.

A block diagram of this basic PCU is shown in Figure 19 in the form it might take with an MSI implementation. Notice that all address information is introduced into the PCU from the data bus. The PC register is resident in the subroutine stack which is 16 levels deep. The RAM used for the subroutine stack is controlled by a four-bit up-down counter used as the stack pointer. When the primary power is turned on, the stack pointer is cleared to zero and the zero-th word in the stack becomes the PC register. The output of the RAM is connected to the input of a latch which in turn is connected to a full adder used as an incrementer. The output of the incrementer and the data bus is presented to a 2-input multiplexer, the output of which is connected to the data input of the RAM. (Although not shown specifically in the diagram, the latch is necessary to avoid a race condition.)

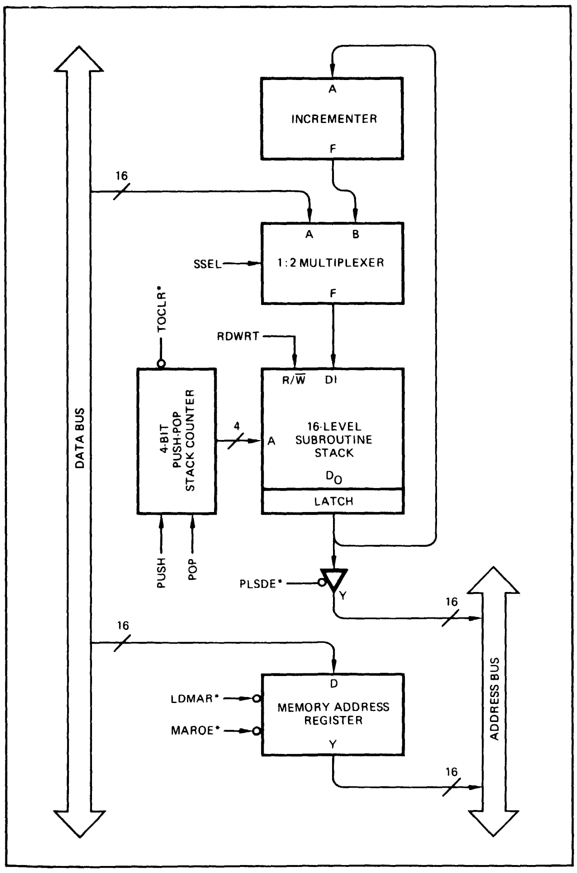


**Figure 19. MSI Program Control Unit Block Diagram.**

If a Jump instruction is executed, the stack input multiplexer selects the data bus as the address source and the address is loaded into the PC register. If a Jump-to-Subroutine instruction is executed, the PC register is incremented at the same clock pulse (this is the return address) and stack pointer is incremented (or PUSHed) and the address of the first instruction of the subroutine is loaded into the new RAM word which is now the PC register for this subroutine level. When a Return-from-Subroutine instruction is encountered, the stack pointer is decremented (or POPed) and the program execution will continue with the instruction after the Jump-to-Subroutine.

Up to 15 levels of subroutine may be implemented with this hardware implementation. The MSI PCU described above takes approximately thirty 16-pin packages.

Figure 20 provides an LSI implementation of the PCU described above in four 20-pin and one 16-pin package. (The one difference between the two systems is that the LSI version is only capable of four levels of subroutine.) The LSI PCU is implemented with four Am2911's and a microcontrol PROM. The Am2911 is a four-bit slice microsequencer which is derived from the 28-pin Am2909.

## THE Am2909 AND Am2911

The Am2909 is a bipolar microprogram sequencer intended for use in high-speed microprocessor applications. The device is a cascadable four-bit slice such that two devices allow addressing of up to 256 words of microprogram and three devices allow addressing of up to 4K words of microprogram. A detailed logic diagram is shown in Figure 21.

The device contains a four-input multiplexer that is used to select either the address register, direct inputs, microprogram counter, or file as the source of the next microinstruction address. This multiplexer is controlled by the $S_0$ and $S_1$ inputs.

The address register consists of four-D-type, edge-triggered flip-flops with a common clock enable. When the address register enable is LOW, new data is entered into the register on the clock LOW-to-HIGH transition. The address register is available at the multiplexer as a source for the next microinstruction address. The direct input is a four-bit field of inputs to the multiplexer and can be selected as the next microinstruction address. This allows an N-way branch where N is any word in the microcode.

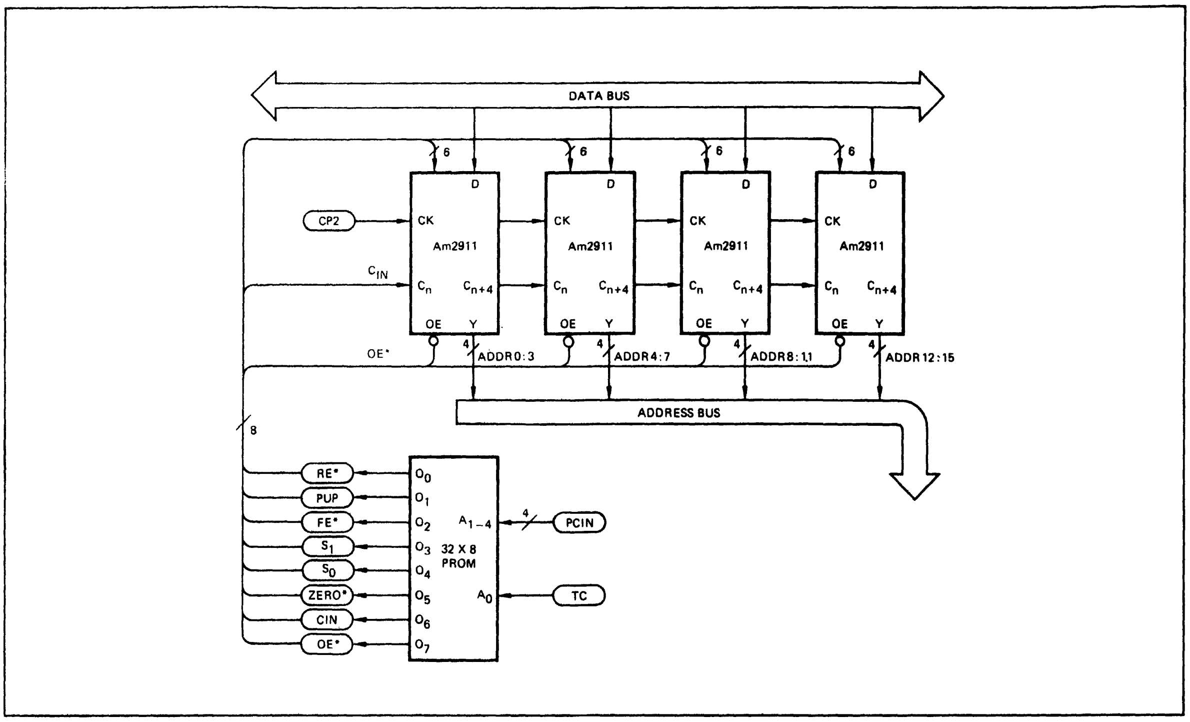


**Figure 20. LSI Program Control Unit Block Diagram.**

The Am2909 contains a microprogram counter (μPC) that is composed of a four-bit incrementer followed by a four-bit register. The incrementer has carry-in ($C_n$) and carry-out ($C_{n+4}$) such that cascading to larger word lengths is straightforward. The μPC can be used in either of two ways. When the least significant carry-in to the incrementer is HIGH, the microprogram register is loaded on the next clock cycle with the current Y output word plus one ($Y+1 \rightarrow \mu PC$). Thus, sequential microinstructions can be executed. If this least significant $C_n$ is LOW, the incrementer passes the Y output word unmodified and the microprogram register is loaded with the same Y word on the next clock cycle ($Y \rightarrow \mu PC$). Thus, the same microinstruction can be executed any number of times by using the least significant $C_n$ as the control.

The last source available at the multiplexer input is the 4 x 4 file (stack). The file is used to provide return address linkage when executing microsubroutines. The file contains a built-in stack pointer (SP) which always points to the last file word written. This allows stack reference operation (looping) to be performed without a push or pop.

The stack pointer operates as an up/down counter with separate push/pop and file enable inputs. When the file enable input is LOW and the push/pop input is HIGH, the PUSH operation is enabled. This causes the stack pointer to increment and the file to be written with the required return linkage – the next microinstruction address following the subroutine jump which initiated the PUSH.

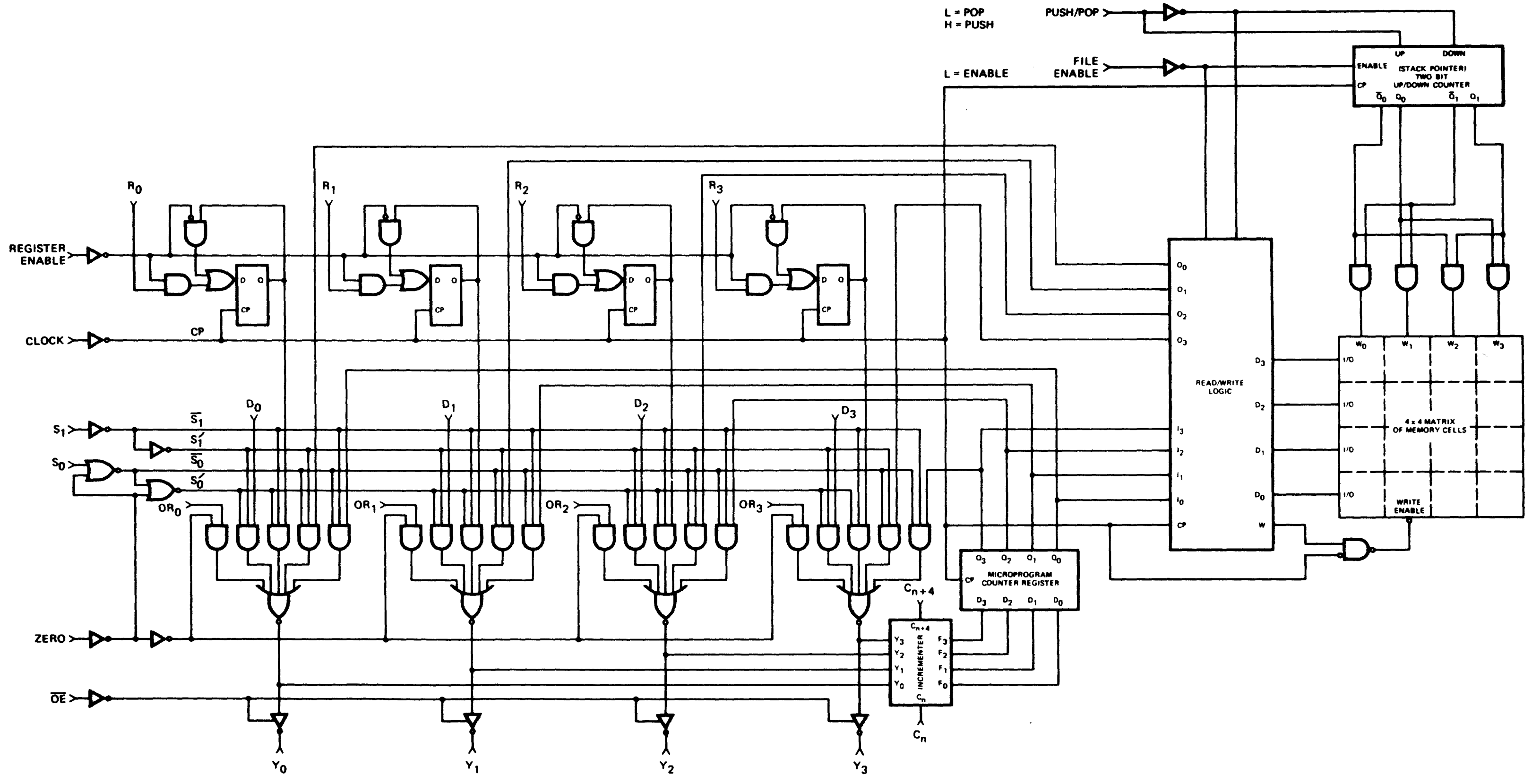
If the file enable input is LOW and the push/pop control is LOW, a POP operation occurs. This implies the usage of the return linkage during this cycle and thus a return-from subroutine. The next LOW-to-HIGH clock transition causes the stack pointer to decrement. If the file enable is HIGH, no action is taken by the stack pointer regardless of any other input.

The stack pointer linkage is such that any combination of pushes, pops or stack references can be achieved. One microinstruction subroutine can be performed. Since the stack is four words deep, up to four microsubroutines can be nested.

The ZERO input is used to force the four outputs to the binary zero state. When the ZERO input is LOW, all Y outputs are LOW regardless of any other inputs (except OE). Each Y output bit also has a separate OR input such that a conditional logic one can be forced at each Y output. This allows jumping to different microinstructions on programmed conditions.

The Am2909 features three-state Y outputs. These can be particularly useful in military designs requiring external Ground Support Equipment (GSE) to provide automatic checkout of the microprocessor. The internal control can be placed in the high impedance state, and preprogrammed sequences of microinstructions can be executed via external access to the control ROM/PROM.

The Am2911 microprogram sequencer, Figure 22, is a 20-pin version of the Am2909. There are two differences between the Am2911 and the Am2909: (1) the direct and register inputs are separate on the Am2909 and common on the Am2911, and (2) there are no OR inputs on the Am2911. Otherwise, the two parts are identical. Although intended especially for control of microprogram memory, these devices contain much of the logic required to control the main memory as well. The Am2911 is used in the PCU described here.



Note: $R_i$ and $D_i$ connected together on Am2911 and $OR_i$ removed.

**Figure 21. Microprogram Sequencer Block Diagram; Am2909.**

## DETAILED LOGIC AND MICRO CONTROL

Almost all of the logic detail is in the Am2911 itself. The remainder of the logic in the PCU lies in the control PROM, an Am29751. There are five address inputs on the PROM: four instruction inputs and one test condition input. Although the test condition signal, TC, is separated in the discussion of the circuit, it is important to understand that TC is in fact a PCU instruction bit. TC does not originate in the microcode, but rather from the ALU and CCU test condition multiplexers. The eight PROM outputs provide all of the control necessary for the Am2911's.

As with the ALU, the PCU clock CP2 is gated by the computer control unit. CP2 is withheld from the PCU for those instructions that are not to cause address storage or modification. There are no other external control signals distributed to the PCU.

The memory address register will be the address or holding register in the Am2911. The PC register will be the micro-PC register, and the existing stack will be used as it would be in a microsequencer. The data inputs will be connected to the data bus and the three-state outputs, Y, will be connected to the address bus. Control of the various components within the Am2909 and Am2911 is summarized in Figure 24.

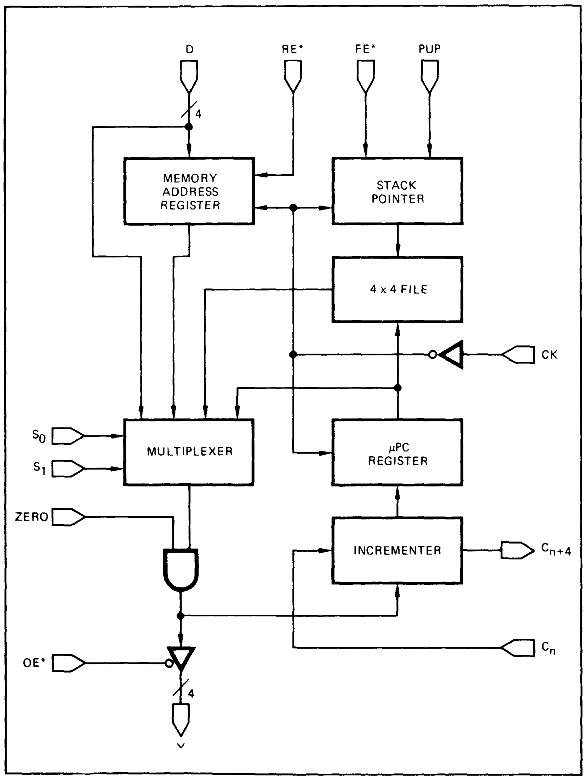


**Figure 22. Am2911 Microsequencer Block Diagram.**

With the four PCU instruction input lines and the test condition line in mind, 16 microinstructions were characterized (Figure 23). The clock, microcode and test condition inputs are presented on the left and the eight PROM outputs are presented on the right. Instruction $0_F$ is a clear instruction that is used in the power-up sequence to start the machine at address $0000_F$, while instruction $1_F$ is the normal execution instruction. That is, $1_F$ causes the PC register to be incremented during each clock cycle.

There are three load instructions: Load MAR, $2_F$, Load PC, $3_F$ and Load PC Incremented, $4_F$. Source MAR, $5_F$, and Source DB, $6_F$, do not change the storage condition of the Am2911's, as the clock pulse is withheld. The internal address multiplexer is merely used to select a source for the address and to present it to the address bus. These first seven instructions are unconditional; the microsequencer control bits are the same regardless of the test condition input.

The Suspend instruction, $7_F$, is conditional, however, and will cause the PC register to stop incrementing and the Y address outputs to go to the high impedance condition. The suspend instruction is used for DMA access and diagnostic and other test conditions. The Hold instruction, $8_F$, is similar to the suspend instruction. The two differences are that it is unconditional, and the Y address outputs are active.

The program modification instructions Jump, Jump-to-Subroutine, and Return, 9 and $10_F$, $11_F$, and $12_F$, respectively, are all conditional. If the test fails – the test condition is not true – the program continues. There are two Jump instructions – $9_F$ selects the MAR as the source of the jump address, and $10_F$ selects the data bus as the jump address source.

The last three instructions, $13_F$, $14_F$, and $15_F$ are Loop instructions. Begin Loop, $13_F$, causes the starting address for the loop to be pushed onto the stack. The Loop command instruction, $15_F$, will reference the stack for the Loop starting address if the test condition is false and cause a jump to that location. If the test condition is true, meaning that the loop is complete, the command to loop is ignored and an Execute is performed as is a Pop stack, thereby clearing the loop reference. Another way to exit the loop is the Branch and Pop instruction, $14_F$. If TC is true, a branch is executed; otherwise, the instruction is ignored and a normal Execute instruction is performed. The loop instructions are presented here for reference only. There are no machine instructions that use them.

| CLOCK | MICROCODE | | | | TEST COND. | PCU INSTRUCTION | | MICROSEQUENCER CONTROL | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CP2 | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | | | OE* | CIN | ZERO* | S0 | S1 | FE* | PUP | RE* |
| ↑ | 0 | 0 | 0 | 0 | 0 | CLEAR | | 0 | 1 | 0 | X | X | 1 | X | 1 |
| ↑ | 0 | 0 | 0 | 0 | 1 | CLEAR | | 0 | 1 | 0 | X | X | 1 | X | 1 |
| ↑ | 0 | 0 | 0 | 1 | 0 | EXECUTE (ADDR ← (PCR)) | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 0 | 0 | 0 | 1 | 1 | EXECUTE (ADDR ← (PCR)) | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 0 | 0 | 1 | 0 | 0 | LOAD MAR (MAR ← (DB)) | | 0 | 0 | 1 | 0 | 0 | 1 | X | 0 |
| ↑ | 0 | 0 | 1 | 0 | 1 | LOAD MAR | | 0 | 0 | 1 | 0 | 0 | 1 | X | 0 |
| ↑ | 0 | 0 | 1 | 1 | 0 | LOAD PCR (PCR ← (DB)) | | 1 | 0 | 1 | 1 | 1 | 1 | X | 1 |
| ↑ | 0 | 0 | 1 | 1 | 1 | LOAD PCR | | 1 | 0 | 1 | 1 | 1 | 1 | X | 1 |
| ↑ | 0 | 1 | 0 | 0 | 0 | LOAD PCR INCR. (PCR ← (DB) +1) | | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| ↑ | 0 | 1 | 0 | 0 | 1 | LOAD PCR INCR. | | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | SOURCE MAR (ADDR ← (MAR)) | NO CK | 0 | 0 | 1 | 1 | 0 | 1 | X | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | SOURCE MAR | NO CK | 0 | 0 | 1 | 1 | 0 | 1 | X | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | SOURCE DB (ADDR ← (DB)) | NO CK | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | SOURCE DB | NO CK | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| ↑ | 0 | 1 | 1 | 1 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 0 | 1 | 1 | 1 | 1 | SUSPEND (ADDR ← 3-STATE) | | 1 | 0 | 1 | X | X | 1 | X | 1 |
| ↑ | 1 | 0 | 0 | 0 | 0 | HOLD (PCR ← (PCR): ADDR ← (PCR)) | | 0 | 0 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 0 | 0 | 1 | HOLD | | 0 | 0 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 0 | 1 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 0 | 1 | 1 | JUMP REGISTER (PCR ← (REG)) | | 0 | 1 | 1 | 1 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 1 | 1 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 1 | 0 | 1 | JUMP DIRECT (PCR ← (DB)) | | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| ↑ | 1 | 0 | 1 | 1 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 0 | 1 | 1 | 1 | JUMP-TO-SUBROUTINE | | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| ↑ | 1 | 1 | 0 | 0 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 1 | 0 | 0 | 1 | RETURN-FROM-SUBROUTINE | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| ↑ | 1 | 1 | 0 | 1 | 0 | PUSH LOOP STARTING ADDRESS | | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| ↑ | 1 | 1 | 0 | 1 | 1 | PUSH LOOP STARTING ADDRESS | | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| ↑ | 1 | 1 | 1 | 0 | 0 | EXECUTE | | 0 | 1 | 1 | 0 | 0 | 1 | X | 1 |
| ↑ | 1 | 1 | 1 | 0 | 1 | BRANCH AND POP | | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| ↑ | 1 | 1 | 1 | 1 | 0 | LOOP | | 0 | 1 | 1 | 0 | 1 | 1 | X | 1 |
| ↑ | 1 | 1 | 1 | 1 | 1 | EXECUTE AND POP | | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

**Figure 23. PCU Microinstruction Control PROM.**

**Address Selection**

| OCTAL | $S_1$ | $S_0$ | SOURCE FOR Y OUTPUTS | SYMBOL |
|---|---|---|---|---|
| 0 | L | L | Microprogram Counter | $\mu$PC |
| 1 | L | H | Register | REG |
| 2 | H | L | Push-Pop stack | STK0 |
| 3 | H | H | Direct inputs | $D_i$ |

**Output Control**

| $OR_i$ | $\overline{ZERO}$ | $\overline{OE}$ | $Y_i$ |
|---|---|---|---|
| X | X | H | Z |
| X | L | L | L |
| H | H | L | H |
| L | H | L | Source selected by $S_0$ $S_1$ |

Z = High Impedance

**Synchronous Stack Control**

| $\overline{FE}$ | PUP | PUSH-POP STACK CHANGE |
|---|---|---|
| H | X | No change |
| L | H | Increment stack pointer, then push current PC onto STK0 |
| L | L | Pop stack (decrement stack pointer) |

**Figure 24. Am2909 and Am2911 Control Signal Summary.**

# CHAPTER VII MEMORY SYSTEM

The memory system is static and has a 64K-word by 16-bit architecture. As the design goal stated, an intention to build the processor on 9 x 12 inch printed circuit boards, a memory partitioning scheme must be employed because a 64K x 16 memory will not fit on one card. A 16K-word by 16-bit subsystem (Figure 25) will fit nicely, however. The Am9140 4K x 1 static RAM was chosen for this application.

## Am9130 AND Am9140

The Am9130 products are high performance, low-power, NMOS, 4096-bit, static, read/write random access memories. They are implemented as 1024 words by 4 bits per word. The data input and output signals are bussed together and share common I/O pins.

The Am9140 products are high performance, low-power, 4K-bit, static, read/write random access memories. They are implemented as 4096 words by 1 bit per word. The data input and output signals use separate pins for maximum flexibility.

All interface signal levels are identical to TTL specifications, providing good noise immunity and simplified system design. The three-state output will drive two full TTL loads or eight low-power Schottky loads for increased fan-out, better capacitive drive and improved bus interface capability.

Operational cycles are initiated when the Chip Enable signal goes HIGH. When the read or write is complete, Chip Enable goes LOW to prepare the memory for the next cycle. Address and Chip Select signals are latched on-chip to help simplify system timing. Output data is also latched and is available until into the next operating cycle.

Figures 26 and 27 show the block diagrams for the Am9130 and Am9140. The internal functions are similar, differing only in the treatment of I/O lines and the corresponding changes in the column circuitry to accommodate the alternate organizational structure.

Chip Enable is the master control input that coordinates all internal functions. When CE goes HIGH, a memory operation is initiated. When the active operation is complete, CE goes LOW to preset the memory for the next cycle. There are no maximum time restrictions on either CE polarity.

Notice that there are ten address lines and four I/O lines for the Am9130. The Am9140 requires two more addresses but two fewer I/O lines, so the pin configurations are nearly identical. The address inputs are latched into the on-chip address register by the rising edge of CE. The row addresses propagate through the row decoder and into the storage matrix to select one of 64 rows. The column addresses propagate through the column decoder to the output of the matrix to select the appropriate column(s) for input to the sense amplifier(s) during a read operation or output from the write amplifier(s) during a write operation.

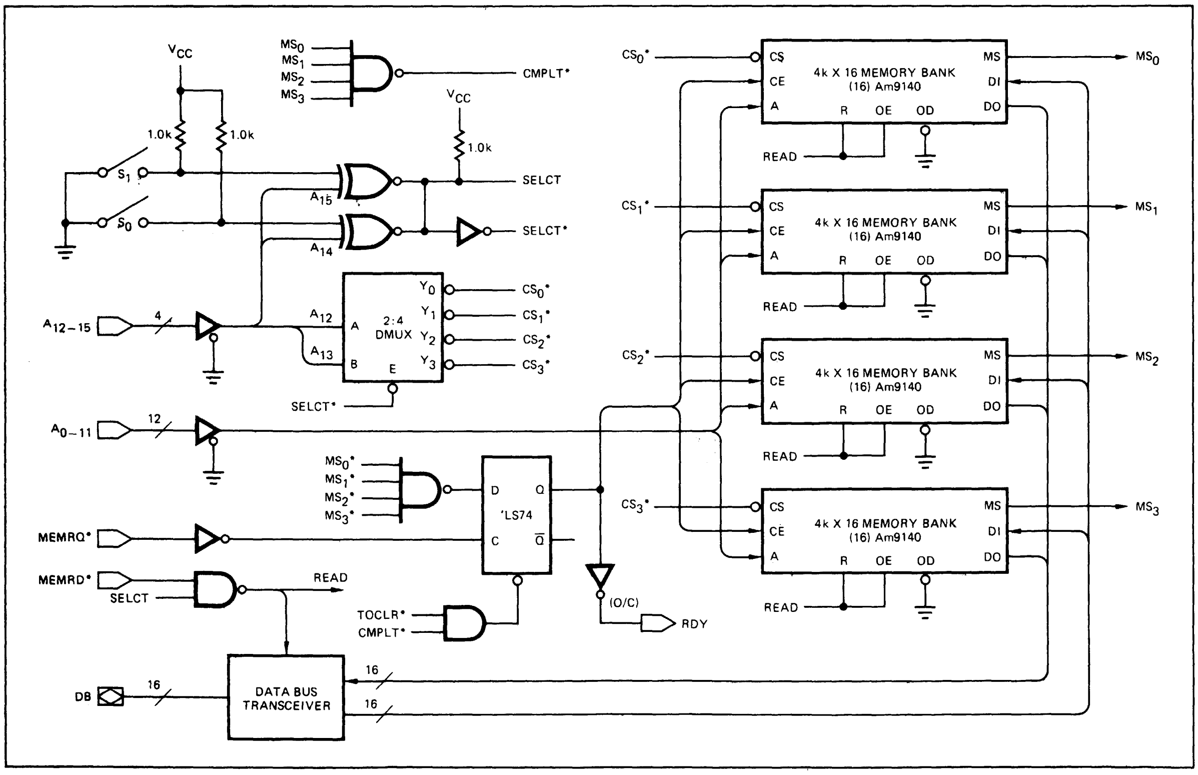


Figure 25. Memory Subsystem Block Diagram.

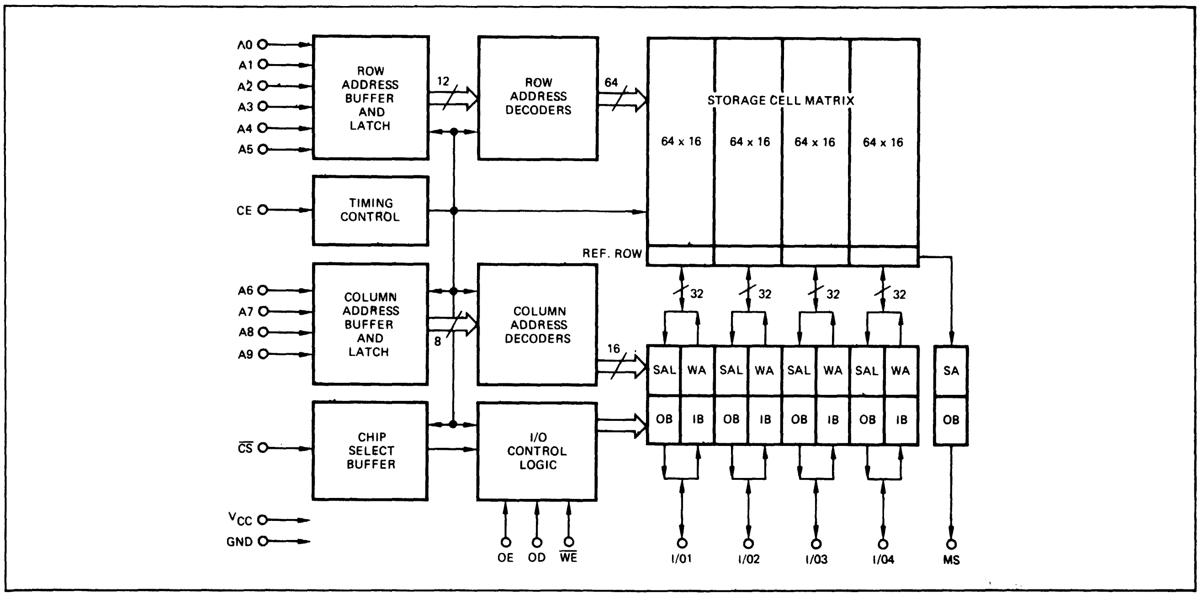


Figure 26. Am9130 Block Diagram.

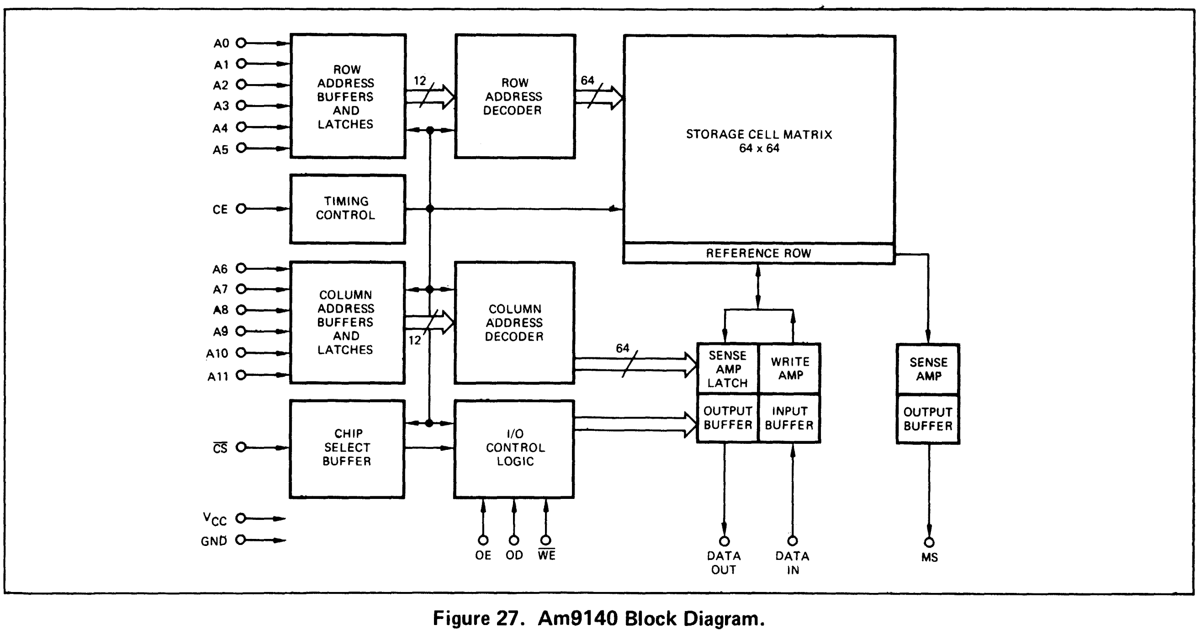


Figure 27. Am9140 Block Diagram.

The Chip Select input is also latched on-chip and acts as the high order address input. It controls the input and output buffers of the memory and allows several chips to be wired together for increased capacity.

The Am9140 has separate single Data-In and Data-Out lines. They may be bussed together externally if desired. The Am9130 has four bi-directional Data I/O lines that provide output data during read cycles and accept input data during write cycles.

The Read input signal controls the nature of the operation being performed by the memory. When it is HIGH, the write amplifiers are disabled and only a read can be executed. When it is LOW during CE HIGH time, the write amplifiers are enabled and incoming data will be written in the addressed cell(s).

Output Enable and Output Disable inputs are used to control the mode of the three-state output buffer(s). OE must be HIGH and OD must be LOW in order for output data to appear. If either OE is LOW or OD is HIGH, the data output(s) will be off and present a high impedance interface.

The Memory Status output is a control signal that describes the internal state of the memory. It indicates when output data is available and valid; and it shows when the CE transitions may take place.

## DETAILED LOGIC

Referring again to Figure 25, the memory is organized in four memory banks of 4K x 16. This requires 16 Am9140's per memory bank. Each bank will be addressed with address bus bits, $A_{0-11}$, which are common, and by a chip select bit, $CS_{0-3}$, which is unique to each bank yet common within the bank. The DC address current parameter is minimal with a maximum of 10µA per line per device; however, with an input capacitance of 9.0pF per device the AC load is more significant. For this reason, a buffer has been used between the address bus and the Am9140's; a good choice for this buffer is the Am25LS241.

The chip select lines are driven by a 2:4 demultiplexer (Am25LS139) whose input address is determined by address bus lines $A_{12}$ and $A_{13}$. The demultiplexer is enabled by SELCT*, which is connected to the gate, being true. SELCT* is derived from a comparison of the two most significant address bits, $A_{14}$ and $A_{15}$, and two switches that reside on each memory subsystem printed circuit board, $S_0$ and $S_1$. When the computer system is assembled, each memory board in the system must be given a unique "page" address. As there are only four pages allowed in this particular computer, the two switches are capable of specifying the page and the two address bits are capable of selecting the proper page. The two-bit comparator is a pair of open collector Exclusive-NOR gates.

The memory status signals, $MS_{0-3}$, are generated by taking the logical AND of all 16 individual MS's in each row, while $MS_{0-3}$* are derived by taking the logical NOR of the same signals.

The CCU controls the memory system with two signals: MEMRQ*, and MEMRD*. MEMRQ*, when true, is a request for a memory cycle. MEMRD*, when true, indicates that the requested cycle is a read cycle; when false, a write cycle has been requested. Although MEMRQ* initiates a cycle request, the Am9140 requires the chip enable signal, CE, to be true to start a cycle. CE will be true if a memory cycle request has been generated, and the page was selected, SELCT, and none of the RAM's is currently completing a memory cycle, $MS_{0-3}$* are all false. This combinatorial signal STRT*, is loaded into a D flip-flop whose Q output is CE. CE is then distributed to the chip enable controls and in turn is inverted with an open-collector device that is connected to the ready line, RDY, making it false. When memory access has been completed, all of the memory status lines, $MS_{0-3}$, are true, generating a complete signal, CMPLT*. CMPLT* resets the status flip-flop making RDY true.

MEMRD is also gated with SELCT, precluding a faulty attempt to write which might result in a loss of data. The derived signal, READ, is used to control the direction of the data bus transceiver, Am25LS241's, and the Am9140's output enable and read/write signals, OE and R.

# CHAPTER VIII
# INTERRUPT CONTROL UNIT

The interrupt control unit block diagram is shown in Figure 28. The ICU accepts interrupts from input/output devices, processes them, selects the most significant interrupt request, acknowledges the interrupt, and provides the address vector for the interrupt processing subroutine in main memory. The interrupt request is processed by an Am2914 which is supported by an Am2913. The interrupt vector is stored in an Am29705.

## Am2914

The Am2914, Figure 29, is a high-speed, eight-bit priority interrupt unit that is cascadable to handle any number of priority interrupt request levels. The device provides a full vectored priority output as determined by the highest priority interrupt request.

The key elements of the Am2914 include an eight-bit dual rank interrupt register, an eight-bit mask register, and a three-bit status register. The device is fabricated using advanced low-power Schottky technology and is packaged in a 40-pin ceramic DIP.

The Am2914 can perform 16 instructions (Figure 30), associated with interrupt control, vector read, mask register control, and status register control. The device can be cascaded in either a ripple mode or a parallel mode to provide high-speed priority interrupt detection.

Circuits in the Am2914 include a group enable flip-flop that is used as a status pointer. In larger systems, only one group enable flip-flop will be LOW at a time. This results in inhibiting interrupt requests in all lower priority blocks. The block in which this enable flip-flop is LOW can be thought of as the decoded binary levels of the more significant status register bits.

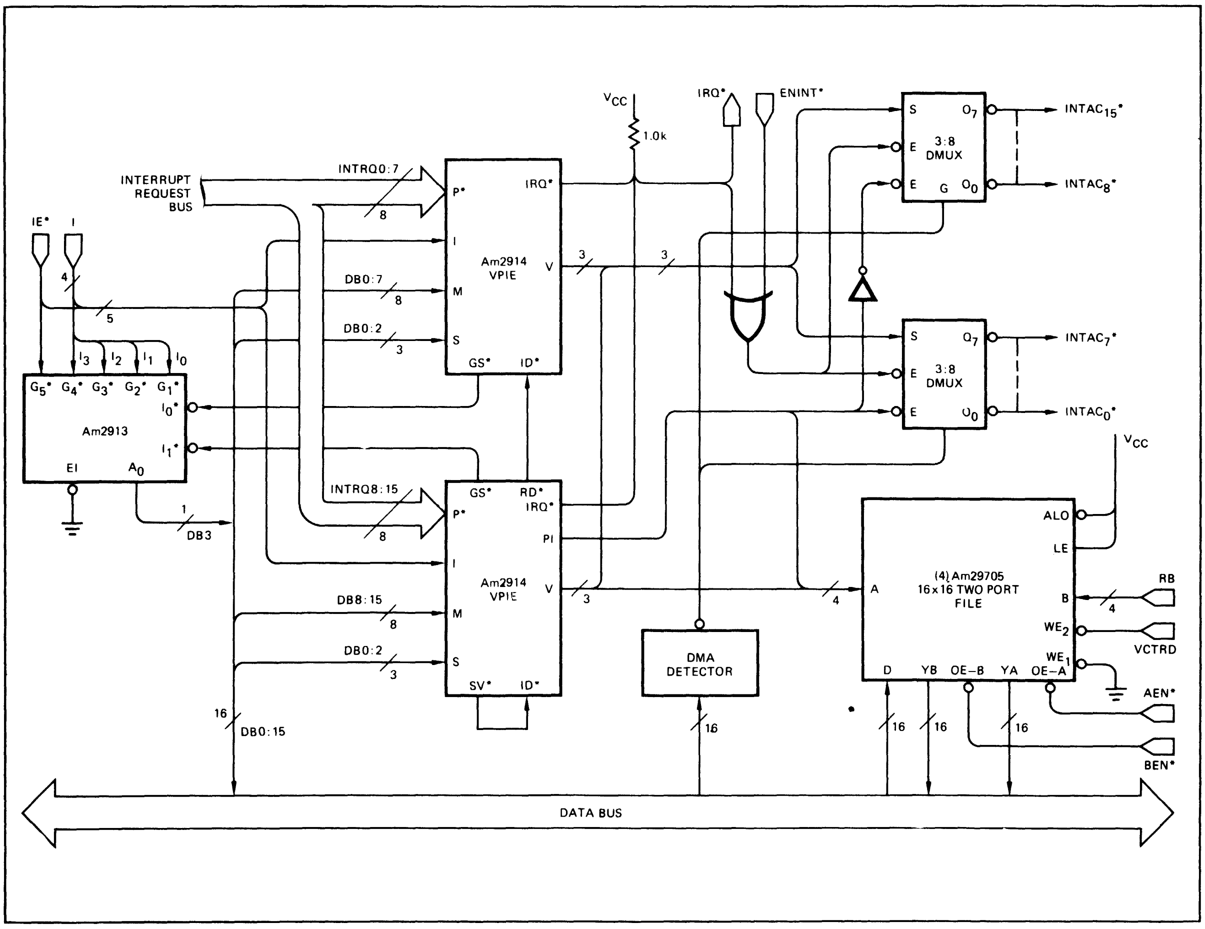


Figure 28. Interrupt Control Unit Block Diagram.

The microinstruction set for the Am2914 priority interrupt circuit is in Figure 30. In this instruction set, the master clear instruction is selected as binary zero such that during a power up sequence, the pipeline register in the microprogram control unit of the central processor can be cleared (0000). Thus, on the next clock cycle, the Am2914 will execute the master clear function. This includes clearing all interrupt registers as well as the mask register, status registers, and group enable flip-flop except at the least significant block. The least significant group enable flip-flop is set LOW by tieing the Group Advance Receive input LOW. All other Group Advance Receive inputs are tied to Group Advance Send outputs, and these are forced HIGH during this instruction. This clear instruction also enables the interrupt request flip-flop such that a fully interrupt driven system can be easily initiated from any interrupt.

The read vector output instruction is used to read the vector value of the highest priority causing the interrupt. The vector outputs are three-state drivers that are enabled onto the $V_0V_1V_2$ bus during this instruction. When multiple Am2914's are used, the higher order vector bits can be encoded using an Am2913 priority encoder. This instruction also causes the value "vector plus one" to be automatically loaded into the status register. Thus, the status register always points to the lowest level at which the user is willing to accept an interrupt. All lower priority levels than that pointed to by the status register are, therefore, inhibited. By loading the status register with the value "vector plus one," multiple interrupts at the same priority level are inhibited. In addition, this instruction loads the current vector value into a holding register such that this value can be used by the clear interrupt from the vector instruction. This allows the user to read the vector associated with the interrupt, update the status register to any value (possibly much higher than "vector plus one"), and then clear the interrupt flip-flop in the interrupt register associated with the vector read.

## Am2913

The Am2913 was designed especially as a support component for the Am2914 Vector Priority Interrupt Encoder. Similar to the Am9318, Am25LS148, and Am74LS148, the Am2913 is an eight-input priority encoder which is used to derive the vector for the most significant Am2914 location (Figure 33). The vector output, $A_{0\text{-}2}$, is three-state and controlled by the gating function $G_{1\text{-}5}$.

The Am2913 is cascadable by connecting the enable output, EO*, to the enable input, EI*, of the next least significant Am2913.

## Am29705

The Am29704 and Am29705 are 16-word by 4-bit, 2-port RAM's built using advanced low-power Schottky processing. These RAM's feature two separate output ports such that any two 4-bit words can be read from the outputs simultaneously. Each output port has a four-bit latch but a common Latch Enable (LE) input is used to control all eight latches. The device has two Write Enable (WE) inputs and is designed such that the Write Enable 1 ($WE_1$) and Latch Enable (LE) inputs can be wired together to make the operation of the RAM appear edge triggered.

The device has a fully decoded four-bit A address field to address any of the 16 memory words for the A output port. Likewise, a four-bit B address input is used to simultaneously select any of the 16 words for presentation at the B output port. New incoming data is written into the four-bit RAM word selected by the B address. The D inputs are used to load new data into the device.

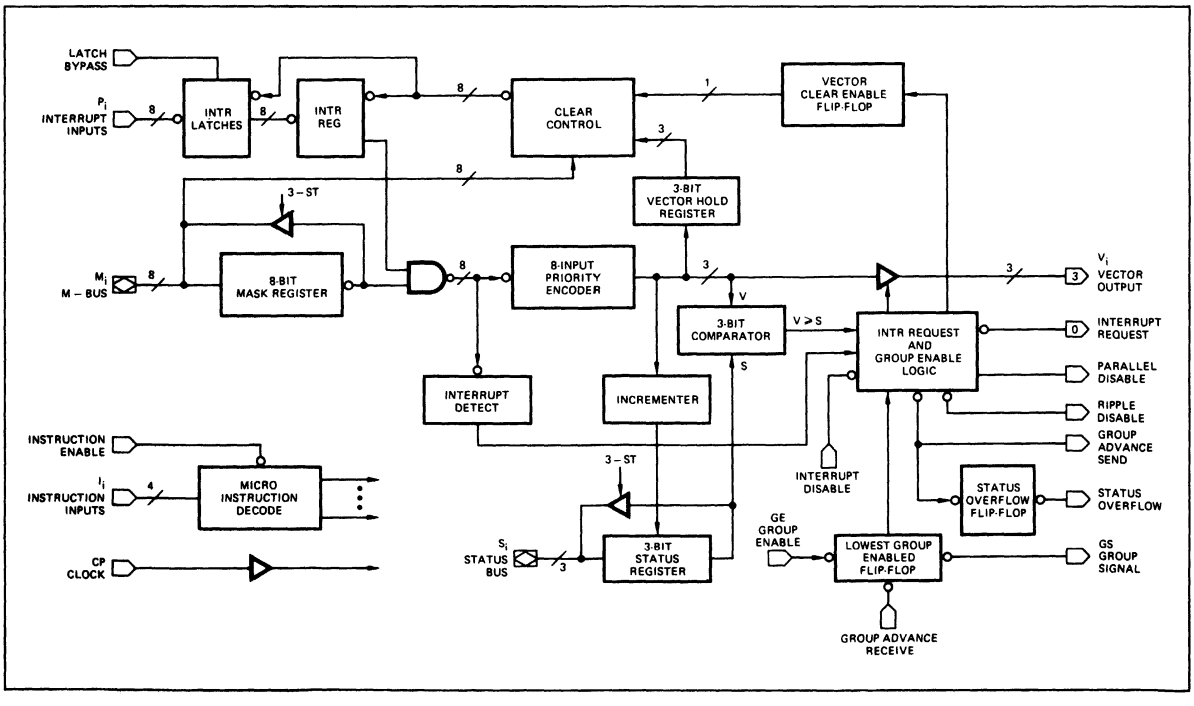


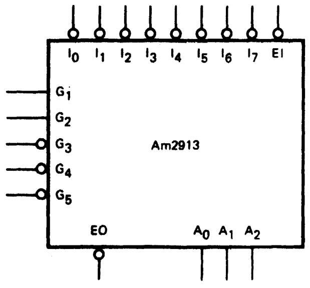
**Figure 29. Am2914 Vectored Priority Interrupt Encoder Block Diagram.**

| Decimal $I_3I_2I_1I_0$ | | Decimal $I_3I_2I_1I_0$ | |
|---|---|---|---|
| | **Mask Register Functions** | | **Vectored Output** |
| 14 | Load mask register from M bus | 5 | Read vector output to V outputs, load V+1 into status register, load V into vector hold register and set vector clear enable flip-flop. |
| 7 | Read mask register to M bus | | |
| 12 | Clear mask register (enables all priorities) | | |
| 8 | Set mask register (inhibits all interrupts) | | |
| 10 | Bit clear mask register from M bus | | **Priority Interrupt Register Clear** |
| 11 | Bit set mask register from M bus | 1 | Clear all interrupts |
| | | 3 | Clear interrupts from mask register data (uses the M bus) |
| | **Status Register Functions** | | |
| 9 | Load status register from S bus and LGE flip-flop from GE input | 2 | Clear interrupts from M bus data |
| 6 | Read status register to S bus | 4 | Clear the individual interrupt associated with the last vector read |
| | **Interrupt Request Control** | | **Master Clear** |
| 15 | Enable interrupt request | 0 | Clear all interrupts, clear mask register, clear status register, clear LGE flip-flop, enable interrupt request |
| 13 | Disable interrupt request | | |

**Figure 30. Microinstruction Set for Am2914 Priority Interrupt Circuit.**

The Am29704 has open-collector outputs, and the Am29705 features three-state outputs so that several devices can be cascaded to increase the total number of memory words in the system. The A output port is in the high impedance state when the $\overline{OE\text{-}A}$ input is HIGH. Likewise, the B output port is in the high impedance state when the $\overline{OE\text{-}B}$ input is HIGH. Four devices can be paralleled using only one Am25LS139 decoder for output control.

The Write Enable inputs control the writing of new data into the RAM. When both Write Enable inputs are LOW, new data



| $\overline{EI}$ | $\overline{I_0}$ | $\overline{I_1}$ | $\overline{I_2}$ | $\overline{I_3}$ | $\overline{I_4}$ | $\overline{I_5}$ | $\overline{I_6}$ | $\overline{I_7}$ | $\overline{EO}$ | $A_0$ | $A_1$ | $A_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | X | X | X | X | X | X | X | X | H | L | L | L |
| L | H | H | H | H | H | H | H | H | L | L | L | L |
| L | X | X | X | X | X | X | X | L | H | H | H | H |
| L | X | X | X | X | X | X | L | H | H | L | H | H |
| L | X | X | X | X | X | L | H | H | H | H | L | H |
| L | X | X | X | X | L | H | H | H | H | L | L | H |
| L | X | X | X | L | H | H | H | H | H | H | H | L |
| L | X | X | L | H | H | H | H | H | H | L | H | L |
| L | X | L | H | H | H | H | H | H | H | H | L | L |
| L | L | H | H | H | H | H | H | H | H | L | L | L |

| $G_1$ | $G_2$ | $\overline{G_3}$ | $\overline{G_4}$ | $\overline{G_5}$ | $A_0$ | $A_1$ | $A_2$ |
|---|---|---|---|---|---|---|---|
| H | H | L | L | L | ENABLED | | |
| L | X | X | X | X | Z | Z | Z |
| X | L | X | X | X | Z | Z | Z |
| X | X | H | X | X | Z | Z | Z |
| X | X | X | H | X | Z | Z | Z |
| X | X | X | X | H | Z | Z | Z |

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care
Z = HIGH Impedance

**Figure 33. Am2913 Function Diagram and Truth Tables.**

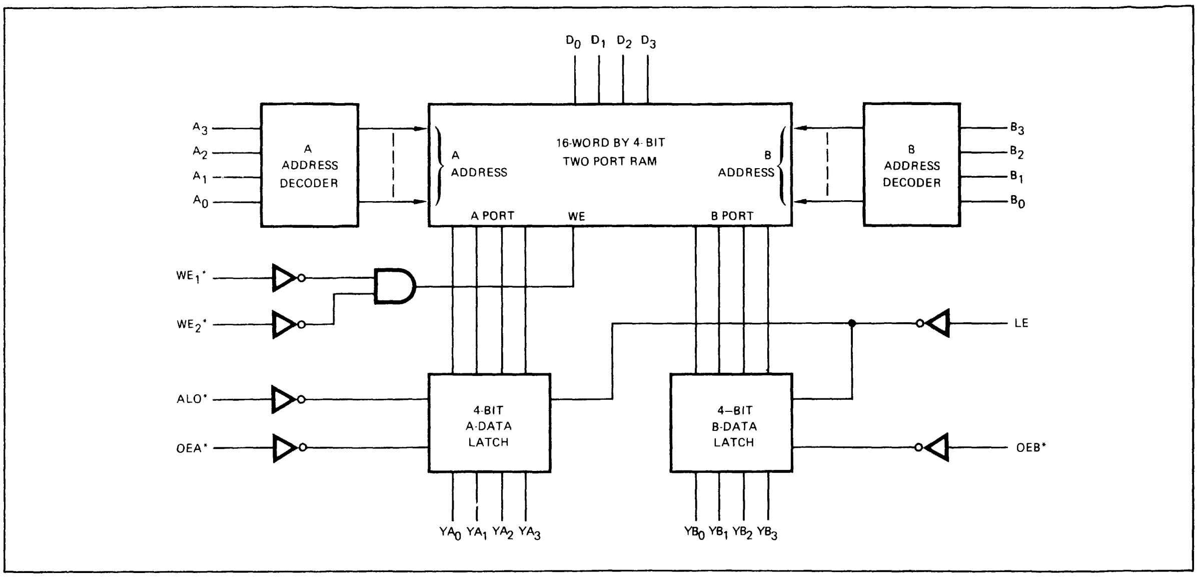


Figure 34. Am29705 Logic Diagram.

WRITE CONTROL

| $\overline{WE}_1$ | $\overline{WE}_2$ | Function | RAM Outputs at Latch Inputs | |
|---|---|---|---|---|
| | | | A-Port | B-Port |
| L | L | Write D Into B | A data (A ≠ B) | D input data |
| X | H | No write | A data | B data |
| H | X | No write | A data | B data |

H = HIGH
L = LOW
X = Don't care

YA READ

| Inputs | | | YA Output | Function |
|---|---|---|---|---|
| $\overline{OE-A}$ | $\overline{A-LO}$ | LE | | |
| H | X | X | Z | High impedance |
| L | L | X | L | Force YA LOW |
| L | H | H | A – Port RAM data | Latches transparent |
| L | H | L | NC | Latches retain data |

H = HIGH Z = High impedance
L = LOW NC = No change
X = Don't care

YB READ

| Inputs | | YB Output | Function |
|---|---|---|---|
| $\overline{OE-B}$ | LE | | |
| H | X | Z | High impedance |
| L | H | B – Port RAM data | Latches transparent |
| L | L | NC | Latches retain data |

H = HIGH Z = High impedance
L = LOW NC = No change
X = Don't care

Figure 35. Am29705 Function Tables.

is written into the word selected by the B address field. When either Write Enable input is HIGH, no data is written into the RAM. The memory outputs follow the data inputs during writing if the latches are open.

The logic diagram and function tables for the Am29705 are shown in Figures 34 and 35, respectively.

## DETAILED LOGIC

The ICU is one of the most complex subsystems in the computer because of all the interaction required. The 16 interrupt requests, $INTRQ_{0-15}$*, originate in different input/output ports and are presented to the Am2914's for service. In the event of an interrupt request, the signal IRQ* is generated and transmitted to the CCU for service. When the CCU finishes the current instruction and branches to fetch the next instruction, the interrupt request causes a branch to the microprogram interrupt service routine where the enable interrupt signal ENINT* is generated. The vector of the highest level interrupt ($INTRQ_{15}$* is the highest and $INTRQ_0$* is the lowest) is read out and presented to the interrupt acknowledge demultiplexers and the interrupt service vector registers.

The interrupt service vector register file, Am29705, has its data input and outputs connected to the data bus. The B port, the port which selects the write address, is address controlled by the instruction register $R_1$ field (IR7:4), RB. The read/write control, VCTRD, when true, causes a read and when false causes a write. BEN* enables the B data port on to the data bus. The A port address is the vector address generated by the Am2914's. During an interrupt service procedure, the A data port output enable, AEN*, is caused to be true, placing the main memory interrupt service vector address on the data bus. The data bus is monitored by the DMA detector which is, in effect, a 16-input NAND gate. If the vector address placed on the data bus is a $FFFF_{16}$, the service request is for a DMA cycle. If the current interrupt request is not for a DMA cycle, an interrupt acknowledge signal, $INTAC_n$*, is generated by a 4:16 demultiplexer made up of two Am25LS138 3:8 demultiplexers.

| IE* | $I_3$ | $I_2$ | $I_1$ | $I_0$ | MICROINSTRUCTION FUNCTION |
|---|---|---|---|---|---|
| H | X | X | X | X | Inhibit Instruction |
| L | 0 | 0 | 0 | 0 | RESET |
| L | 0 | 0 | 0 | 1 | CLEAR INTERRUPTS |
| L | 0 | 0 | 1 | 0 | CLEAR INTERRUPTS VIA M-BUS |
| L | 0 | 0 | 1 | 1 | CLEAR INTERRUPTS VIA M-REGISTER |
| L | 0 | 1 | 0 | 0 | CLEAR VECTOR |
| L | 0 | 1 | 0 | 1 | READ VECTOR, STATUS ← V+1 |
| L | 0 | 1 | 1 | 0 | READ STATUS |
| L | 0 | 1 | 1 | 1 | READ MASK |
| L | 1 | 0 | 0 | 0 | SET MASK |
| L | 1 | 0 | 0 | 1 | LOAD STATUS |
| L | 1 | 0 | 1 | 0 | BIT CLEAR MASK |
| L | 1 | 0 | 1 | 1 | BIT SET MASK |
| L | 1 | 1 | 0 | 0 | CLEAR MASK |
| L | 1 | 1 | 0 | 1 | INHIBIT INTERRUPT REQUESTS |
| L | 1 | 1 | 1 | 0 | LOAD MASK |
| L | 1 | 1 | 1 | 1 | ENABLE INTERRUPT REQUESTS |

**Figure 36. Microinstruction Format.**

As there are two 8-input Am2914's generating a 3-bit vector, a fourth bit must be generated to resolve the ambiguity between the two devices. The Am2913 was employed for this purpose. Its gate inputs, $G_{1-5}$, are connected to the Am2914 instruction lines, $I_{0-3}$ and IE*, so that the output of the Am2913 is enabled for the read status command, I = 0110B, and IE* is true. Only one bit of the priority encoder's vector is needed, and it is generated by connecting the most significant Am2914 group status signal, BS* to $I_1$* and the least significant block status to $I_0$*.

The Am2914's are also connected to the data bus, thereby allowing the 16 bits of mask register to be read or loaded, the interrupts to be cleared, and the status register to be read or loaded. The instruction codes for the Am2914 are presented in Figure 36.

# CHAPTER IX
# DIRECT MEMORY ACCESS

The DMA controller (Figure 37) in this machine is relatively simple in structure and completely controlled by the CCU. There are two 16-bit registers in the machine, a base address register and a word counter.

The DMA initialization counter requires that the base address register is loaded with the address of the first word to be transferred. In the same process, the word counter is loaded with the two's complement of the number of data words to be transferred. The DMA interrupt service vector, FFFF$_{16}$, is loaded into the interrupt service vector register, and the transfer direction flip-flop is set to the appropriate condition. IO* in a true condition means a write into memory from the selected input/output port, while a false condition is a memory to input/output device transfer.

Both registers were built out of Am25LS161 four-bit binary, synchronous counters. And the Am25LS2541 non-inverting octal buffers were selected for address bus isolation.

When the first DMA request is generated, the proper read or write in main memory occurs. At the end of the memory cycle, both the word counter and the base address register are incremented. The base address register is then pointing to the address of the next main memory transfer, and the number of words to be transferred has been decremented by one. This process continues until the word counter reaches its maximum or terminal count; TERM is true. If terminal count on the base address register, DMOVR, becomes true, the memory has overflowed.

Microcontrol of the DMAC consists of two load commands, LDWC* and LDBAR*, two increment commands, INCWC and INCBA, an output enable control, ENADR*, and a transfer direction, DIR. As with the other controlled subsystems, the clock, CLK, is gated.

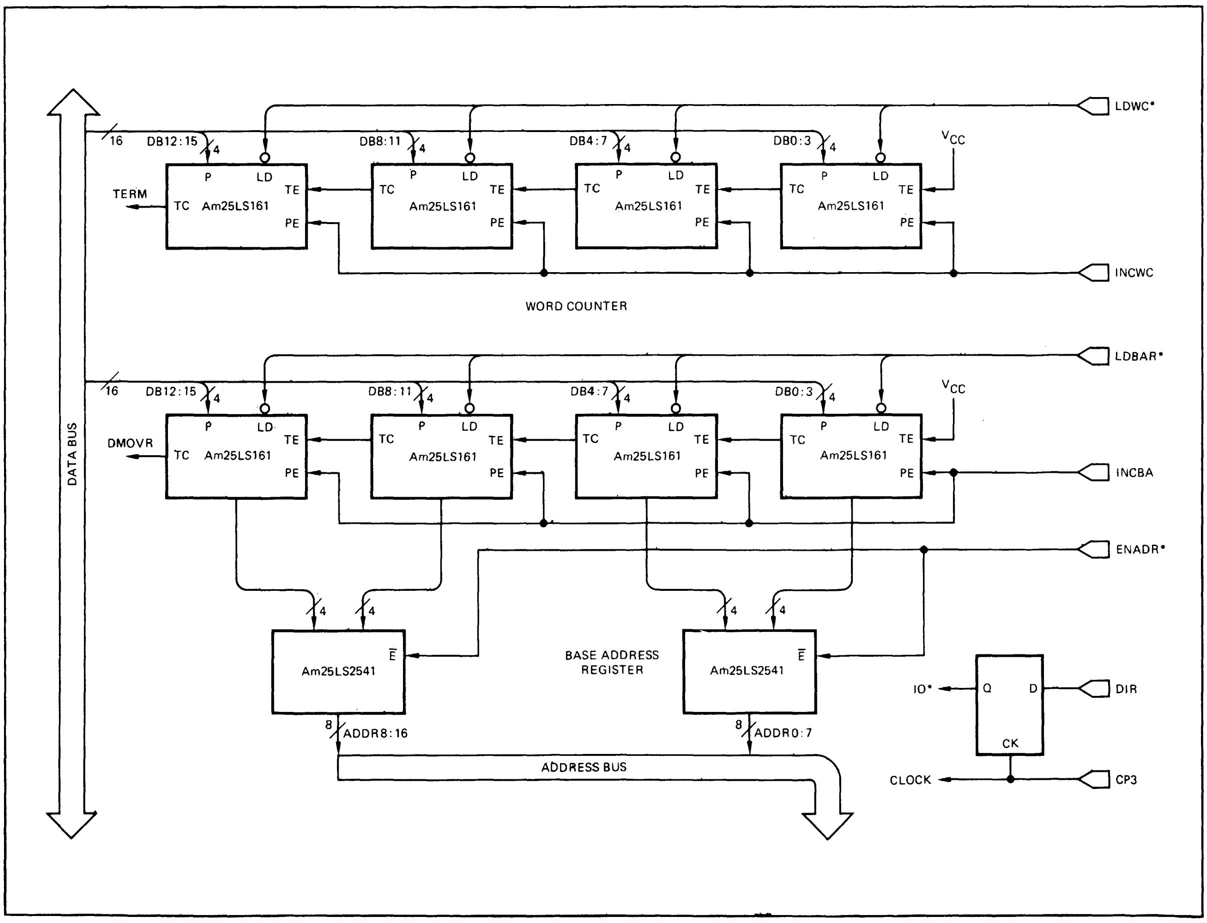


Figure 37. Direct Memory Access Controller.

# CHAPTER X
# THE STATE MACHINE

Every computer has a predictable sequence of functions that occurs depending upon the instruction being executed or the automatic hardware functions being performed, i.e., interrupt requests, control panel functions, etc. The graph depicting the functional flow through the computer sequence is called a state transition diagram and is implemented by execution of the microprogram.

The state transition diagram for this computer is shown in Figure 38. The sequence starts with a power-up initiation function. All of the registers are cleared, the control flip-flops are set, and, if the computer has a control panel, the machine is probably set to a Halt condition at the end of State 1.

State 2 tests the Halt request. In its absence, the CCU performs a memory read cycle to fetch an instruction. At the end of the cycle, the instruction word is loaded into the instruction register.

During State 3, the instruction is decoded, and a determination is made as to whether this instruction is one word or two words long. If it is a single word register-to-register, RR, or a single cycle special function, SF, instruction, the instruction is immediately executed and a transfer is made to State 8. If a multiple cycle SF instruction has been encountered, the same microinstruction is executed until the SF instruction is satisfied. If the instruction is a two-word instruction, the CCU fetches the operand and stores it in the Q register, then continuing to State 4.

State 4 is used to modify the operand and directly or indirectly use it. For RX instructions, a test is made of the X field, IR0:3, for a zero value. If X does equal zero, the address of the second operand is in Q and must be transferred to the MAR. If X is not equal to zero, the contents of the X-th general purpose register is added to the contents of the Q register and the result is loaded into the MAR. A transition is then made to State 6.

If an immediate instruction is detected in State 4, a test for zero is also made on X. When X does not equal zero, the contents of the X-th register is added to the contents of the Q register, and the result forms the second operand which is stored in the Q register. Then transition is made to State 5, or, if X equals zero, the transition is immediately made to State 5.

State 5 is the execution state for RI instructions. The only transition from this state is to State 8.

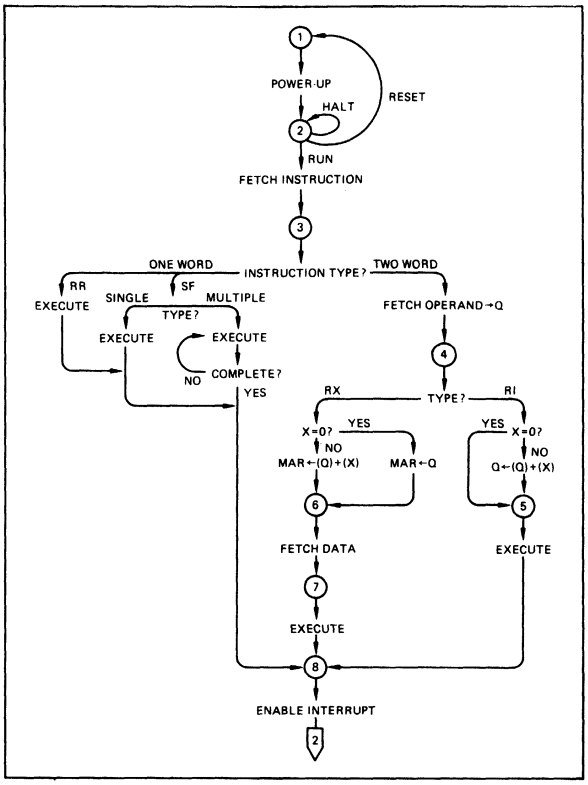


**Figure 38. Central Processor State Transition Diagram.**

State 6 fetches the second operand for RX instructions from the computed address that was previously loaded in the MAR during State 4. The next state from State 6 is State 7 where the RX instructions are executed.

State 8 allows the interrupt system to be enabled, and then the CCU attempts to branch back to State 2 to fetch the next instruction. If there has not been an interrupt, the attempt to branch is successful; otherwise, the interrupt request must be processed. When the interrupt has been serviced in firmware, a branch to State 2 occurs and the interrupt service routine in main memory is processed. Execution of a Return from Interrupt instruction will allow the controller to be returned to its pre-interrupt condition and for normal processing to resume.

# CHAPTER XI
# COMPUTER CONTROL UNIT

Having defined each of the controlled subsystems in the computer, and the machine instruction set, the machine instruction formats, the "state machine", and the microcontrol bits, the design of the computer control unit to drive this processor is a relatively straightforward task.

The architecture of the CCU is presented in block diagram form in Figure 39. The instruction register contains the current machine level instruction that is being interpreted and executed incrementally by the microprogram. The Op Code portion of the instruction is presented to a mapping PROM as an address. In turn, the output of the mapping PROM provides the microsequencer with the starting address of the microprogram that satisfies the requirements of the current instruction.

Initially, the starting address is passed directly through the microsequencer to the address field of the microprogram PROM. The first microinstruction appears at the output of the PROM some time later and is loaded into the microinstruction register (the pipeline register) on the rising edge of the next clock pulse. Concomitant with being passed directly to the microprogram PROM, an incrementer inside the sequencer adds 1 to the starting address and sets up the input to the microprogram counter register which is loaded with "address-plus-one" by the same clock pulse that loaded the microinstruction register. Subsequently, the source of the microprogram PROM address is switched from the mapping PROM to the microprogram counter and the process continues.

The timing, synchronization, and control logic includes the power-up initialization hardware, the system clock, the instruction decoder for the microsequencer, conditional instruction hardware, and the logic that allows processor asynchronous events to be synchronized to the CPU.

Having been given permission to operate from the synchronization logic, the bus control and clock distribution hardware gates data sources onto the data bus, and clocks data from the data bus into controlled destinations.

## INSTRUCTION DECODING

A detailed logic diagram of the microsubroutine starting address hardware is shown in Figure 40. The instruction register consists of two sections. First is the eight-bit Op Code register that is made from two Am2918 two-input, quad registers, and second is the dual operand register comprised of two 4-bit registers, Am2918.

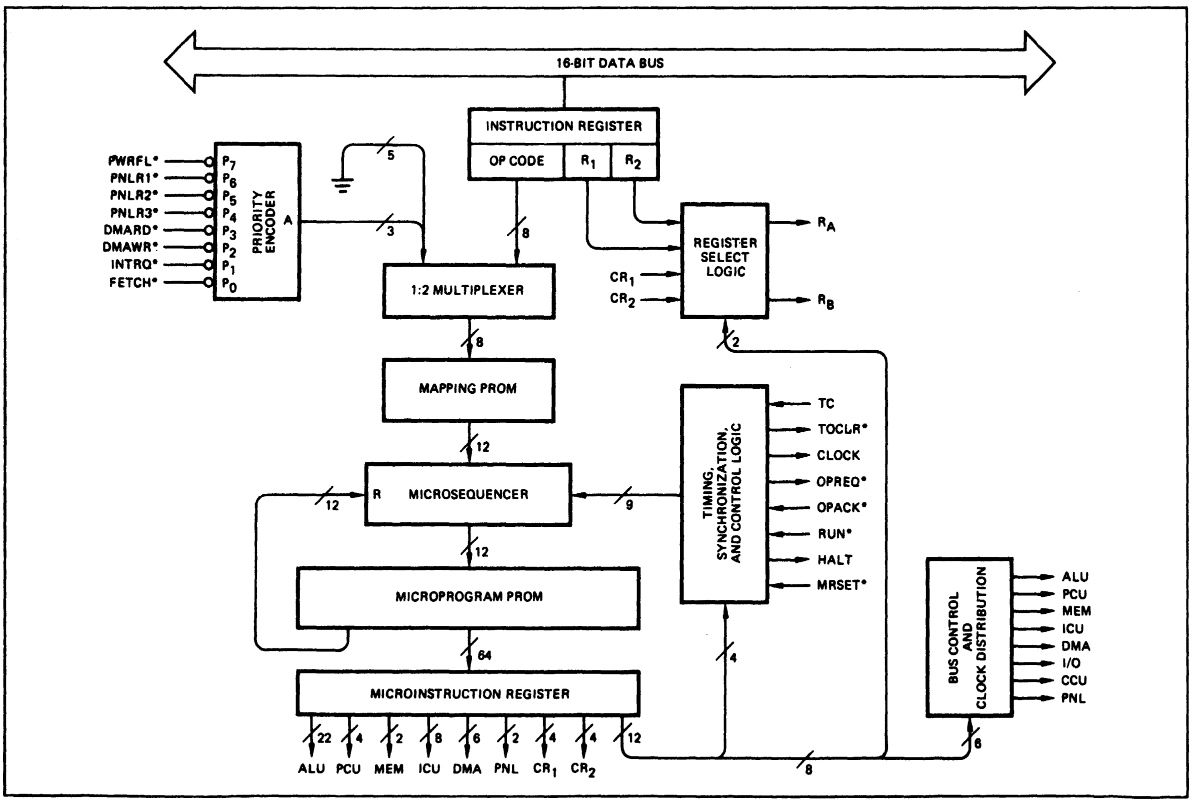


Figure 39. Computer Control Unit Block Diagram.

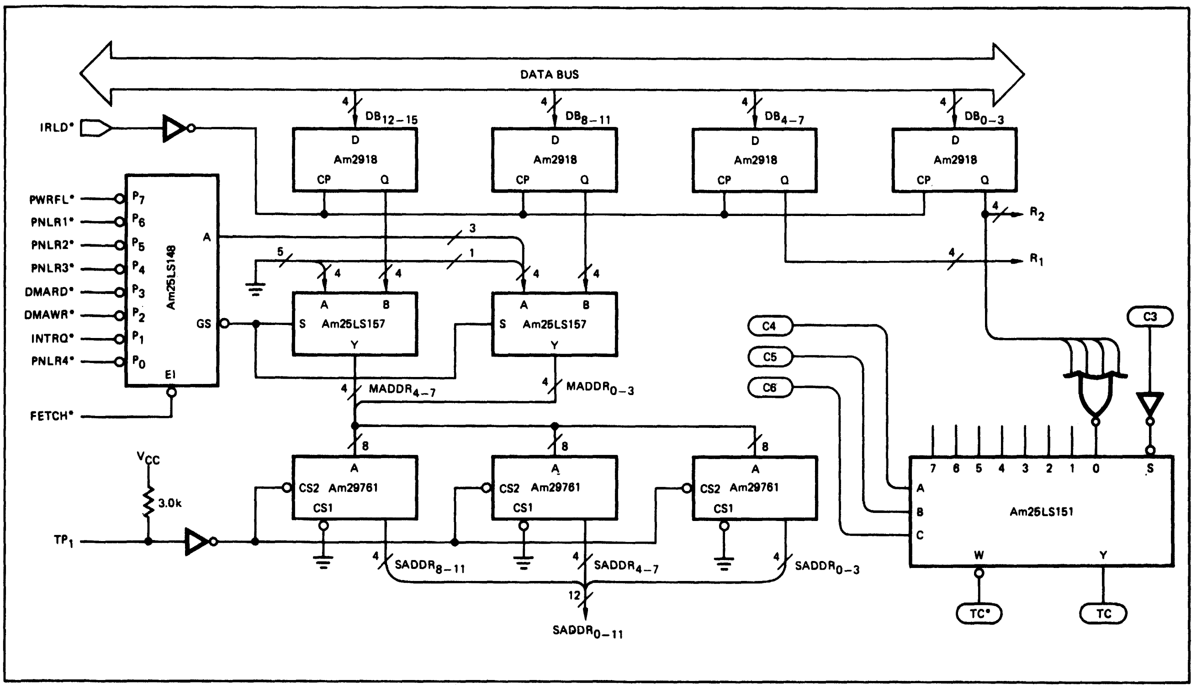


**Figure 40. Microsubroutine Starting Address Block Diagram and Test Condition Multiplexer.**

At the beginning of an instruction sequence, the machine instruction is parallel loaded into the instruction register, and the Op Code is decoded to provide a starting address as described above. And, during the entire execution of the instruction, interrupt requests are disabled. But when the instruction is completed, the microcontroller executes a microprogram that fetches the next instruction and inserts it into the instruction register. Between the end of the last instruction and the beginning of the instruction fetch, routine machine level service requests are enabled.

Machine level service requests include power-fail interrupts, PWRFL*, control panel service requests, PNLR1*, PNLR2*, and PNLR3*, an attempt to DMA read or write, DMARD* or DMAWR*, stored program interrupt requests, INTRQ*, and the instruction fetch itself, FETCH*. Each of these functions has a separate microprogram with a unique starting address. Presented in order of priority, these service requests are encoded in an Am25LS148, and the three-bit vector is selected as the input to the Op Code multiplexer. Op Code/micro-interrupt selection is accomplished when FETCH* is true causing the priority encoder gate select, GS*, to go low, thereby switching the Am2918 output multiplexer. Three bits of vector and five bits of "0" are inserted into the Op Code multiplexer leaving Op Codes $00_{16}$ through $07_{16}$ unavailable for machine instruction assignment.

Generating machine level service request addresses in this manner has two advantages: speed and a smaller microprogram word width. While the last microinstruction for the current machine instruction is being executed, the fetch signal may be enabled, and the first instruction of the most significant service routine may be readv to be loaded into the microinstruction register by the very next clock pulse. If the proposed technique, or one similar to it, is not implemented, a series of test and branch instructions must be executed. This technique saves microinstruction word width because the PROM field would have to be 12 bits wider to provide a branch address at the same time that other machine functions were being exercised.

During the instruction fetch and class determination functions, it is necessary to detect the value of the X register field for RX and RI instructions for zero. If this 4-bit field is non-zero the contents of the specified register is added to the second operand to calculate an effective address for the data word to be used. Figure 40 allows for the testing of the zero condition as well as other signals used for conditional microprogram control.

The register select logic shown in Figure 41 is provided to allow for the instruction operand flexibility shown in the defined instruction set of Chapter IV. These multiplexers allow the source and destination registers for the Am2901 ALU and Am29705 ICU/DMA circuits to be selected from either instruction register operand field or from one of two fields in microprogram memory. Providing the two register select fields in microprogram memory allows algorithms to be executed on the system.

## MICROPROGRAM SEQUENCING

The Am2909 microprogram sequencer was described in some detail in Chapter VI, and a block diagram is provided here in Figure 42 as a reference to be used with Figure 39. Three Am2909's are being used to provide an address range of 12 bits or 4096 words of microprogram storage. All three have their equivalent control lines connected together and all 12 OR inputs are tied to ground. The least significant carry-in signal is tied to $V_{CC}$ and all interstage carry-outs and carry-ins are appropriately connected.

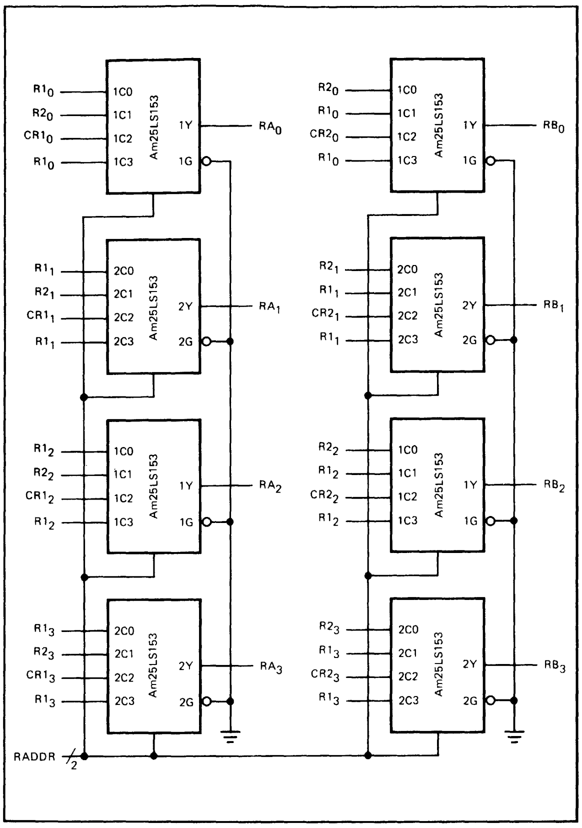


Figure 41. Instruction Register/Microprogram Register Select Logic.

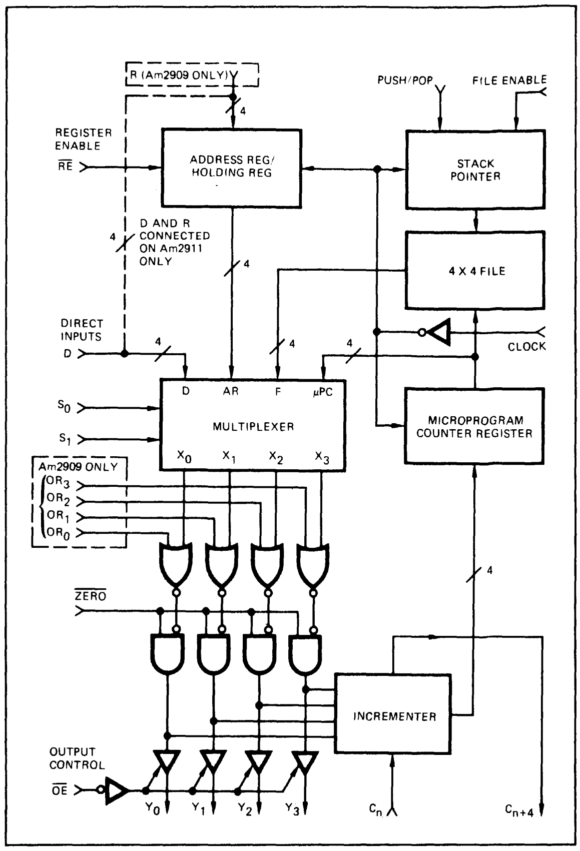


Figure 42. Am2909 Microprogram Sequencer Block Diagram.

The direct inputs are tied to the outputs of the starting address PROM's; and the register inputs are tied directly to the outputs of the microprogram PROM's, in the branch address field, thereby forming part of the microprogram register. All three RE*'s are connected to ground causing the address register inside of the Am2909 to be loaded on every clock pulse.

A 32 X 8 PROM, Am29751, Figure 43, is used to control the microsequencers. Of the five address inputs, four are used as instruction bits and the fifth is used as a qualified (by the output, $O_7$) test condition. The other seven outputs are used for control of FE*, PUP, S0, S1, and ZERO*, and for generation of OPREQ* and HLTRQ*. OPREQ* is used for functions thay may have mixed speeds or be asynchronous such as memory or I/O reading and writing. HLTRQ* is used to halt the processor. The control signals for the microsequencer PROM come from an Am25LS161 which is the least significant four bits of the microinstruction register. This device was chosen for two reasons: during the power-up sequence it may be cleared to "0," and with its terminal count gate it will detect the presence of all "1's" in the register. The terminal count detector generates the FETCH* condition.

The system clock is one-half of a 74LS124 VCO. Because of the speed and need for accuracy in this application, the VCO is being crystal-excited by a 5MHz series resonant device. The turn-on clear signal, TOCLR*, is generated by an Am555 timer using an RC time constant (t = 1.1RC) and should have a time constant of 100msec or greater for most systems.

A rather powerful microinstruction set is summarized for the Am2909 in Figure 44. The inputs to the control PROM are shown on the left, and the output control signals are shown on the right. Unconditionally occupying the zero-th position is the instruction RESET. This instruction causes the address out of the Am2909 to go to "0." The second instruction is EXECUTE which is also unconditional. EXECUTE selects the microprogram counter as the source of address information. EXECUTE is the default condition for most of the conditional instructions.

There are two jump and two jump-to-subroutine instructions that reference either the direct inputs or the register inputs. There are two return instructions, a classical RETURN for in-line programs, and a return with a LOOP default condition causing the subroutine to loop until the return condition is true.

There is an unconditional PUSH instruction for initializing a loop, a BRANCH and POP instruction that allows an escape from a loop without exiting from the bottom. Both conditional and unconditional loops are provided. An unconditional POP is also provided to clear a loop condition or an unwanted return address.

HALT is conditional and causes the CPU to freeze until there is a manual intervention or the primary power is cycled off and on. EXECUTE ASYNCHRONOUS and EXECUTE and FETCH are both unconditional and were discussed above.

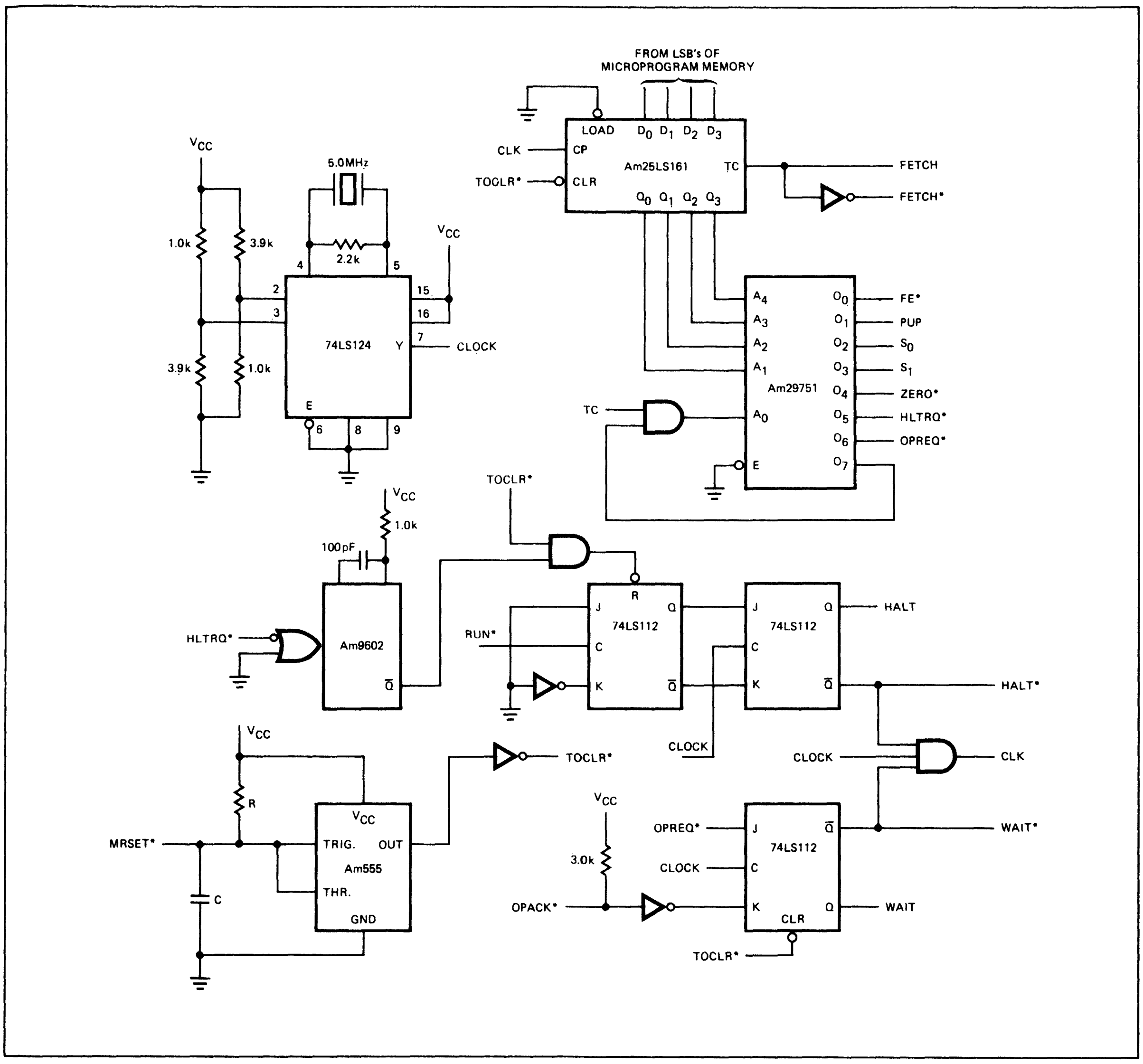


**Figure 43. Timing, Synchronization and Control Logic Schematic.**

## DATA BUS CONTROL

The general form of the bus functional control logic is shown in Figure 45. There are three bits of data bus source and three bits of data bus destination information is provided. The source decoder gates information onto the data bus while the destination decoder distributes clock pulses to transfer the data from the data bus to a storage device.

Figure 46 shows an actual set of clock distribution and output gate control logic that will satisfy the requirements of the CPU. Signals $BSRC_{0-2}$ control the data bus source and $BDST_{0-2}$ control the data bus destination. The derived signals are distributed to the various subsystems.

## MICROPROGRAM FORMATTING

The microinstruction register of Figure 39 shows all of the control bits necessary to implement all of the control functions for all of the controlled elements. If summed, the total is 76 bits. A microinstruction word that wide would require a lot of PROM's, and a lot of registers, and a lot of printed circuit board room. This might be acceptable for very high-speed processors or even desirable for very flexible machines. For almost all applications, however, there is a cost/performance trade-off.

The process of narrowing the microinstruction word is called formatting. The first step in formatting is to determine which functions are desired or required to go together. Three examples of formatting are shown in Figure 47. These formats require 48-bit wide microinstruction words. The word width may be further shortened to 40 bits if it is deemed undesirable to do algorithm control purely from microcode.

The decision to use formatted microwords or unformatted microwords is up to the designer. In an unformatted system, the microinstruction will not only be wider than in the formatted words, but a significant number of the bits will not be

| MICROCODE | | | | TC | INSTRUCTION | TCEN | OPREQ* | HLTRQ* | ZERO* | $S_1$ | $S_0$ | PUP | FE* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | RESET | 0 | 1 | 1 | 0 | X | X | X | 1 |
| 0 | 0 | 0 | 0 | 1 | RESET | 0 | 1 | 1 | 0 | X | X | X | 1 |
| 0 | 0 | 0 | 1 | 0 | EXECUTE | 0 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 0 | 0 | 1 | 1 | EXECUTE | 0 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 0 | 1 | 0 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 0 | 1 | 0 | 1 | JUMP-THROUGH-REGISTER | 1 | 1 | 1 | 1 | 0 | 1 | X | 1 |
| 0 | 0 | 1 | 1 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 0 | 1 | 1 | 1 | JUMP DIRECT | 1 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| 0 | 1 | 0 | 0 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 1 | 0 | 0 | 1 | JSB-THROUGH-REGISTER | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 1 | 0 | 1 | 1 | JSB DIRECT | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 0 | 1 | 1 | 0 | 1 | RETURN | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | LOOP | 1 | 1 | 1 | 1 | 1 | 0 | X | 1 |
| 0 | 1 | 1 | 1 | 1 | RETURN | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | PUSH | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | PUSH | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 1 | 0 | 0 | 1 | 1 | BRANCH AND POP | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | LOOP | 0 | 1 | 1 | 1 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | 0 | 1 | LOOP | 0 | 1 | 1 | 1 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | 1 | 0 | EXECUTE AND POP | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | LOOP | 1 | 1 | 1 | 1 | 1 | 0 | X | 1 |
| 1 | 1 | 0 | 0 | 0 | POP | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | POP | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | EXECUTE | 1 | 1 | 1 | 1 | 0 | 0 | X | 1 |
| 1 | 1 | 0 | 1 | 1 | HALT | 1 | 1 | 0 | 1 | 0 | 0 | X | 1 |
| 1 | 1 | 1 | 0 | 0 | EXECUTE ASYNC. | 0 | 0 | 1 | 1 | 0 | 0 | X | 1 |
| 1 | 1 | 1 | 0 | 1 | EXECUTE ASYNC. | 0 | 0 | 1 | 1 | 0 | 0 | X | 1 |
| 1 | 1 | 1 | 1 | 0 | EXECUTE AND FETCH | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |
| 1 | 1 | 1 | 1 | 1 | EXECUTE AND FETCH | 0 | 1 | 1 | 1 | 1 | 1 | X | 1 |

**Figure 44. Am2909 Microprogram Instructions and PROM Pattern.**

used during any given microinstruction. The effect to be considered is the fact that there is a lower (usually significantly) ROM efficiency and greater flexibility for unformatted microinstructions in comparison to formatted microinstructions. Further, formatted microinstruction storage will generally be deeper (require a greater address range) than the unformatted equivalent whereas the unformatted microinstruction will be wider than the formatted equivalent. The use of either technique necessitates the consideration of the mechanical layout problems in the system, package count, and power requirements. The reason that these trade-offs are important can be shown by example: sizing estimates for the entire instruction set shown in this document are 160 ROM words unformatted versus 210 ROM words formatted, with microinstruction widths of 76 bits and 48 bits, respectively.

Let us consider the industry standard 256 x 4 PROM. Both formatted and unformatted versions of the microcode will fit into the depth, 256 words, but due to the additional width, the unformatted version requires 19 PROM packages versus the formatted 12 PROM packages. The formatted version not only saves seven 16-pin PROM packages, but five to seven square inches of printed circuit board for the seven 16-pin pipeline microinstruction registers for a total savings of 10 to 14 square inches and a total power savings of about 1.1 typical to 1.7 maximum amperes. Clearly, performance needs to be a primary factor in the decision process for the final design. A reproducible microprogramming form is included here, Figure 48, to allow various instructions and techniques to be evaluated using as a base the microprogramming examples of Figure 18.

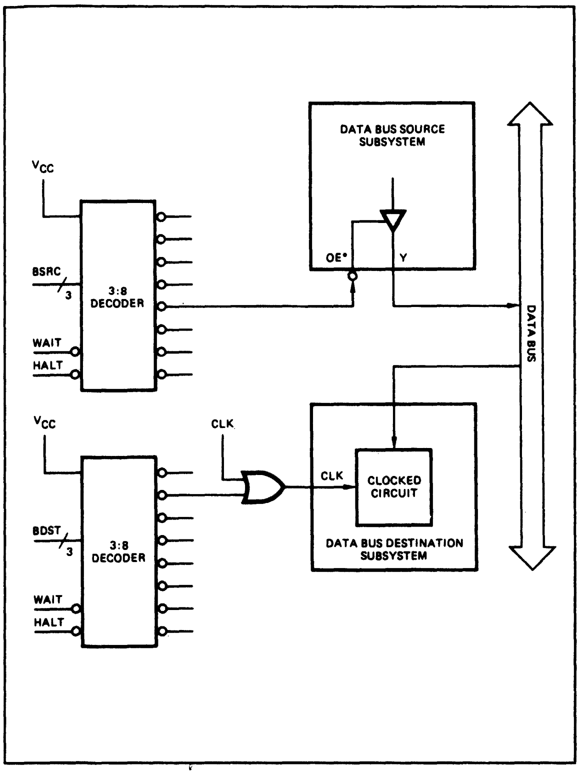


**Figure 45. Data Bus Control Logic Technique.**

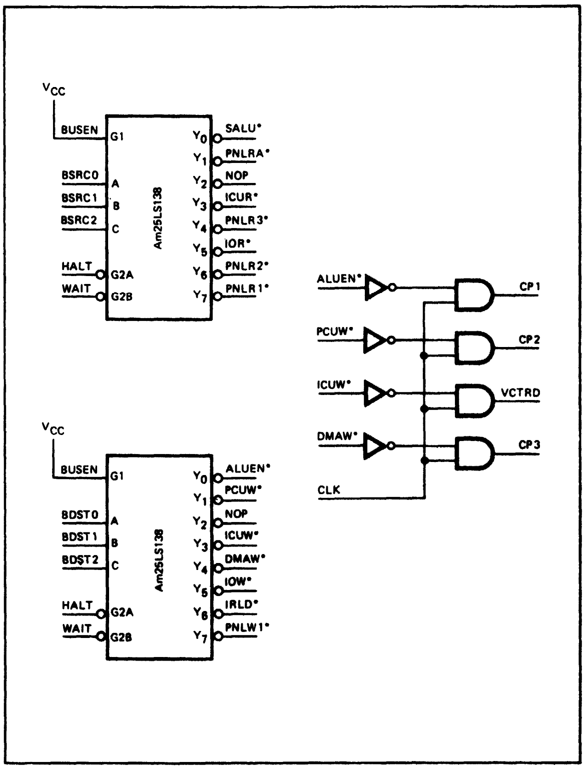


**Figure 46. Bus Control and Clock Distribution Logic.**

| ARITHMETIC LOGIC UNIT | | | | | CR1 | CR2 | PCU | MEM | COMPUTER CONTROL UNIT | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| COND | CARRY CNTRL | SHIFT CNTRL | $I_{0-5}$ | $I_{6,7,8}$ | | | | | BUS CONTROL | RADDR | $\mu$I |
| 47 45 | 44 40 | 39 35 | 34 29 | 28 26 | 25 22 | 21 18 | 17 14 | 13 12 | 11 6 | 5 4 | 3 0 |

a) Arithmetic and Program Control Format

| | ICU | DMA | ALU | CR1 | PANEL | PCU | MEM | COMPUTER CONTROL UNIT | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $I_{6,7,8}$ | | | | | BUS CONTROL | RADDR | $\mu$I |
| 47 43 | 42 35 | 34 29 | 28 26 | 25 22 | 21 18 | 17 14 | 13 12 | 11 6 | 5 4 | 3 0 |

b) Interrupt, DMA and Control Panel Format

| BRANCH ADDRESS | | ALU | CR1 | TEST COND | PCU | MEM | COMPUTER CONTROL UNIT | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $I_{6,7,8}$ | | | | | BUS CONTROL | RADDR | $\mu$I |
| 47 36 | 35 29 | 28 26 | 25 22 | 21 18 | 17 14 | 13 12 | 11 6 | 5 4 | 3 0 |

c) Microsequence Program Modification Format

**Figure 47. Microprogram Formats.**