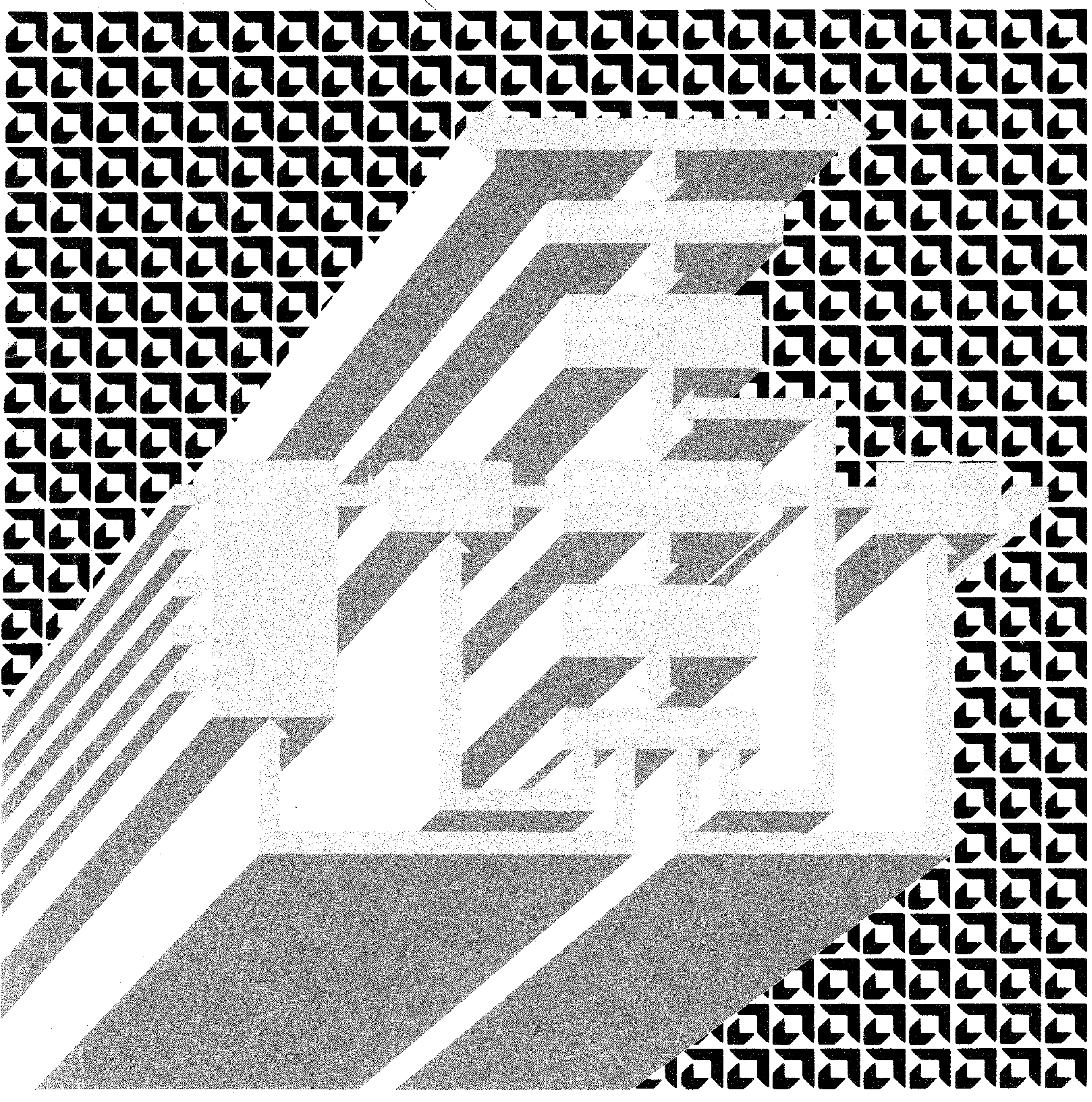
Advanced Micro Devices, Inc.

Microprogramming Handbook

$5.00



Advanced Micro Devices

Advanced Micro Devices

Microprogramming Handbook And Am2910 Emulation

By John R. Mick and Jim Brick

Second Edition

Advanced Micro Devices cannot assume responsibility for use of any circuitry described other than circuitry entirely embodied in an Advanced Micro Devices' product.

MICROPROGRAMMING A BIPOLAR MICROPROCESSOR

By John R. Mick and Jim Brick

INTRODUCTION

With the advent of the Am2901 4-bit microprocessor slice and the Am2909/Am2911 bipolar microprogram sequencers, the design engineer can upgrade the performance of existing systems or implement new systems taking advantage of the latest state-of-the-art technology in Low-Power Schottky integrated circuits. These devices, however, utilize a new concept in machine design not familiar to many design engineers. This technique is called microprogramming.

Basically, a microprogrammed machine is one in which a coherent sequence of microinstructions is used to execute various commands required by the machine. If the machine is a computer, each sequence of microinstructions can be made to execute a machine instruction. All of the little elemental tasks performed by the machine in executing the machine instruction are called microinstructions. The storage area for these microinstructions is usually called the microprogram memory.

A microinstruction usually has two primary parts. These are: (1) the definition and control of all elemental micro-operations to be carried out and (2) the definition and control of the address of the next microinstruction to be executed.

The definition of the various micro-operations to be carried out usually includes such things as ALU source operand selection, ALU function, ALU destination, carry control, shift control, interrupt control, data-in and data-out control, and so forth. The definition of the next microinstruction function usually includes identifying the source selection of the next microinstruction address and, in some cases, supplying the actual value of that microinstruction address.

Microprogrammed machines are usually distinguished from non-microprogrammed machines in the following manner. Older, non-microprogrammed machines implemented the control function by using combinations of gates and flip-flops connected in a somewhat random fashion in order to generate the required timing and control signals for the machine. Microprogrammed machines, on the other hand, are normally considered highly ordered and more organized with regard to the control function field. In its simplest definition, a microprogram control unit consists of the microprogram memory and the structure required to determine the address of the next microinstruction.

UNDERSTANDING THE MICROPROGRAM MEMORY

The microprogram memory is simply an N word by M bit memory used to hold the various microinstructions. Figure 1 depicts the word number definition of an N word microprogrammed memory. For an N word memory, the address locations are usually defined as location 0 through location N-1. For example, a 256-word microprogram memory will have address locations 0 through 255.

Each word of the microprogram memory consists of M bits. These M bits are usually broken into various field definitions. A typical example of field definition is shown in Figure 1 where a 32-bit word made up of nine fields is depicted. It should be noted that the fields can consist of various numbers of bits. For example, field number 1 is 5 bits wide, field number 2 is 8 bits wide, field number 6 is 1 bit wide, and so forth. It is the definition of the various fields of a microprogram word that is usually referred to as FORMATTING. Thus, Figure 1 shows the FORMAT of a 32-bit microinstruction.

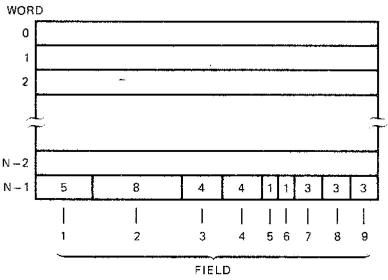


Figure 1. Organization of an N-Word by M-Bit Microprogram Memory.

An example of how microinstruction fields are defined in a typical machine microprogram memory word, as shown in Figure 1, is as follows:

Field 1 – General purpose
Field 2 – Branch address
Field 3 – Next address control
Field 4 – Interrupt control
Field 5 – Fast clock/slow clock select
Field 6 – Carry control
Field 7 – ALU source operand control
Field 8 – ALU function control
Field 9 – ALU destination control

SEQUENCING THROUGH MICROINSTRUCTIONS

Once the microprogram format has been defined, it is necessary to execute sequences of these microinstructions if the machine is to perform any real function. In its simplest form, all that is required to sequence through a series of microinstructions is a microprogram address counter. Such a simplified microprogram memory address control is shown in Figure 2. The microprogram address counter simply increments by one on each clock cycle to select the address of the next microinstruction. For example, if the microprogram address counter contains address 23, the next clock cycle will increment the counter and it will select address 24. The counter will continue to increment on each clock cycle there by selecting address 25, address 26, address 27, and so forth. If this were the only control available, the machine would not be very flexible but it would be able to execute a fixed pattern of microinstructions.

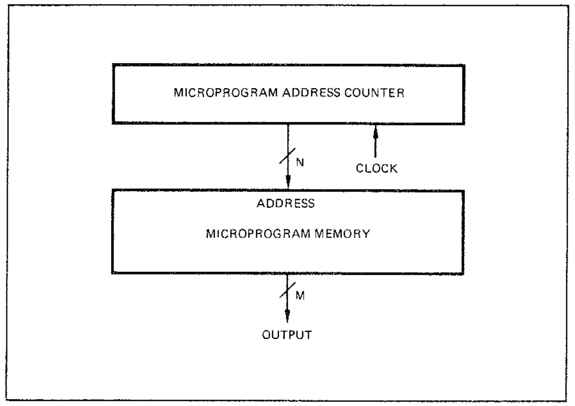


Figure 2. Microprogram Memory Address Control Using a Counter.

The technique of continuing from one microinstruction to the next sequential microinstruction is usually referred to as CONTINUE or EXECUTE. Thus, in microprogram control definition, we will use the CONTINUE (CONT) or EXECUTE (EXEC) statement to mean simply incrementing to the next microinstruction.

MICROPROGRAM BRANCHING OR JUMPING

If the microprogram control unit is to have the ability to select other than the next microinstruction, the control unit must be able to load a BRANCH or JUMP address. Figure 3 shows a control unit architecture whereby a BRANCH address can be parallel loaded into the microprogram address counter. The load control is a single bit field within the microprogram word format. Let us call this one-bit field the microprogram address counter load enable bit. When this bit is at logic 0, a load will be inhibited and when this bit is logic 1, a load will be enabled. If the load is enabled, the BRANCH address contained within the microprogram memory will be parallel loaded into the microprogram address counter. This results in the ability to perform an N-way branch. For example, if the branch address field is eight bits wide, a BRANCH to any address in the memory space from word 0 through word 255 can be performed.

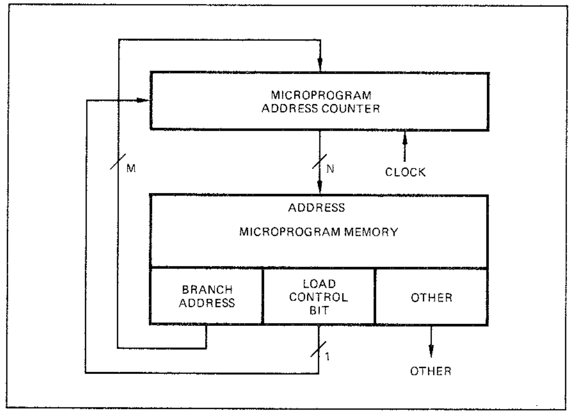


Figure 3. Branching or Jumping in a Microprogram Memory Space Requires a Branch Address and a Load Control Signal.

The simple branching control feature described in Figure 3 allows a microprogram memory controller to execute sequential microinstructions or perform a BRANCH (BR) or JUMP (JMP) to any address either before or after the address currently contained in the microprogram address counter.

CONDITIONAL BRANCHING

While the BRANCH or JUMP instruction has added some flexibility to the sequencing of microprogram instructions, the controller still lacks any decision-making capability. This decision-making capability is provided by the CONDITIONAL BRANCH (COND BR) instruction. Figure 4 shows a functional block diagram of a microprogram memory/address controller providing the capability to branch on either of two different conditions. In this example, the load select control is a two-bit field used to control a four-input multiplexer. When the two-bit field is equivalent to binary zero, the multiplexer selects the zero input which forces the load control inactive. Thus, the CONTINUE microprogram control instruction is executed. When the two-bit load select field contains binary one, the D_1 input of the multiplexer is selected. Now, the load control is a function of the Condition 1 input. If Condition 1 is logic 0, the microprogram address counter increments and if Condition 1 is logic 1, the branch address will be parallel loaded in the next clock cycle. This operation is defined as a CONDITIONAL BRANCH. If the load select input contains binary 2, the D_2 input is selected and the same conditional function is performed with respect to the Condition 2 input. If the load select field contains binary 3, the D_3 input of the multiplexer is selected. Since the D_3 input is tied to logic HIGH, this forces the microprogram address counter to the load mode independent of anything else. Thus, the branch address is loaded into the microprogram address counter on the next clock cycle and an UNCONDITIONAL JUMP is executed. This load select control function definition is shown in Table I.

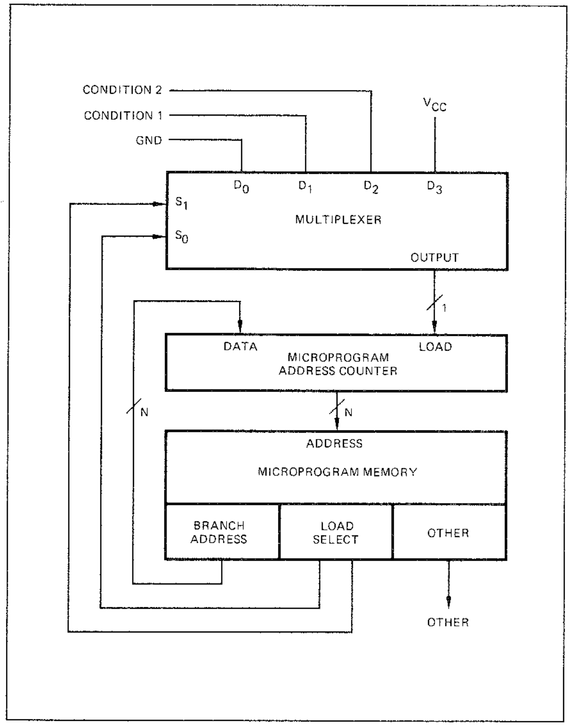


Figure 4. A Two-Bit Control Field Can be Used to Select CONTINUE, BRANCH, or CONDITIONAL BRANCH.

TABLE I
LOAD SELECT CONTROL FUNCTION

S_1 S_0	Function
0 0	Continue
0 1	Jump Condition 1 True
1 0	Jump Condition 2 True
1 1	Jump Unconditional

An example of the power of CONDITIONAL BRANCHING is shown in Figures 5A and 5B. Here, ignoring the start-up problem, let us assume the microprogram address counter contains the word 0. The instruction being executed is a conditional test on Condition 1 input. If Condition 1 is true, branch to word 4 and if Condition 1 is false, continue. Word 1 performs a test on Condition 2. This test results in a branch to word 8 if true, or a continue if false. Word 2 performs a condition test on Condition 1. If true, a branch to word 12 occurs, if false a continue to word 3 occurs. Word 3 is an unconditional branch to the starting address at word 0. In this example, if a jump occurs to either word 4, 8 or 12. The net result is that the next four instructions are executed and then an unconditional branch to word 0 occurs. The flow table for this 16-word microprogram is shown in Figure 5B.

MICROPROGRAM MEMORY

Word			
0	4	1	
1	8	2	
2	12	1	
3	0	3	
4	X	0	
5	X	0	
6	X	0	
7	0	3	
8	X	0	
9	X	0	
10	X	0	
11	0	3	
12	X	0	
13	X	0	
14	X	0	
15	0	3	
	BRANCH ADDRESS	LOAD SELECT	

X = Don't Care

Figure 5A. Conditional Branching Example.

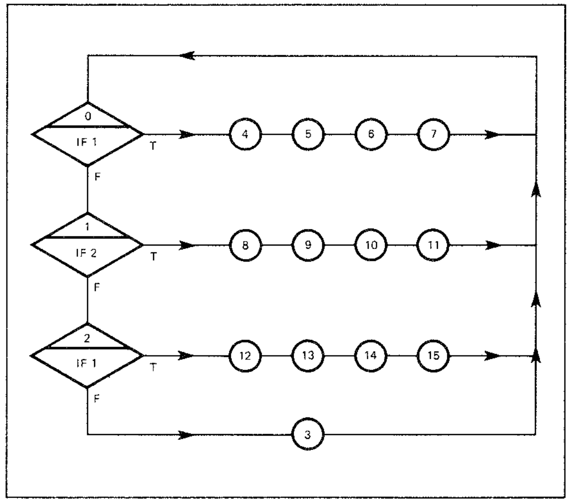


Figure 5B. Flow Table for CONDITIONAL BRANCH Example.

The microprogram control flow described in Figures 5A and 5B represent a simple state machine design controller that looks for either Condition 1 to occur, Condition 2 to occur, or not Condition 2 followed by Condition 1 occurring before a reset to word 0. In the event any of the conditions do occur, a small series of tasks (four microinstructions) are to be executed and then the machine returned to word 0. If none of the test conditions occur, the machine is reset at word 3 and returned to the test at word 0. At power-up, an asynchronous reset could be applied to the microprogram address counter to initialize the machine. Although Figure 5A and Figure 5B describe an extremely simple microprogram control unit, it is representative of the microprogram control power contained in only three basic microinstructions. These microinstructions are the CONTINUE instruction, the UNCONDITIONAL JUMP instruction, and the CONDITIONAL JUMP instruction.

OVERLAPPING THE MICROPROGRAM INSTRUCTION FETCH

Now that a few basic microprogram address control instructions have been defined, let us examine the control instructions used in a microprogram control unit featuring the overlap fetching of the next microinstruction. This technique is also known as "pipelining." The block diagram for such a microprogram control unit is shown in Figure 6. The key difference when compared with previous microprogrammed architectures is the existence of the "pipeline register" at the output of the microprogram memory. By definition, the pipeline register (or microword register) contains the microinstruction currently being executed by the machine. Simultaneously, while this microinstruction is being executed, the address of the next microinstruction is applied to the microprogram memory and the contents of that memory word are being fetched and set-up at the inputs to the pipeline register. This technique of pipelining can be used to improve the performance of the microprogram control unit. This results because the contents of the microprogram memory word required for the next cycle are being fetched on an overlapping basis with the actual execution of the current microprogram word. It should be realized that when the pipeline approach is used, the microprogram fetch, the current microinstruction being executed and the results of the previous microinstruction are available with respect to each other simultaneously. Said another way, the design engineer must be aware of the fact that some registers contain the results of the previous microinstruction executed, some registers contain the current microinstruction being executed, and some registers contain data for the next microinstruction to be executed.

Let us now compare the block diagram of Figure 6 with that shown in Figure 4. The major difference, of course, is the addition of the pipeline register at the output of the microprogram control memory. Also, notice the addition of the address multiplexer at the source of the microprogram memory address. This address multiplexer is used to select the microprogram counter register or the pipeline register as the source of the next address for the microprogram memory. The condition code multiplexer is used to control the address multiplexer in this address selection. By placing an incrementer at the output of the address multiplexer, it is possible to always generate the current microprogramming address "plus one" at the input of the microprogram counter register.

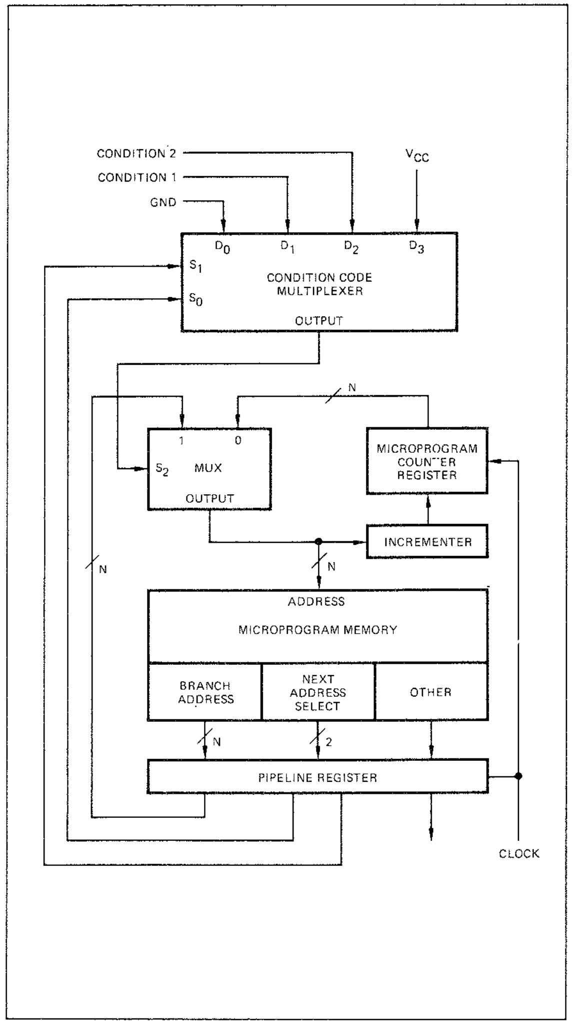


Figure 6. Overlapping (or Pipelining) the Fetch of the Next Microinstruction.

In Figure 4, the microprogram address counter was described as a device such as the Am25LS161 counter. In the implementation as shown in Figure 6, the Am25LS161 counter is not appropriate. Instead, an incrementer and register are used to give the equivalent effect of a counter.

The key difference between using a true binary counter and the incrementer register described here is as follows. When the branch address from the pipeline register is selected by the multiplexer, the incrementer will combinatorially prepare that address plus one for entry into the microprogram counter register. This entry will occur on the LOW-to-HIGH transition of the clock. Thus, the microprogram counter register can always be made to contain address plus one, independent of the selection of the next microinstruction address. When the address multiplexer is switched so that the microprogram counter register is selected as the source of the microprogram memory address, the incrementer will again set-up address plus one for entry into the microprogram counter register. Thus, when the address multiplexer selects the microprogram counter register, the address multiplexer, incrementer and microprogram counter register appear to operate as a normal binary counter.

The condition code multiplexer S_0S_1 operates in exactly the same fashion as described for the condition code multiplexer of Figure 4. That is, binary zero in the pipeline register (the current microinstruction being executed) forces an unconditional selection of the microprogram register via D_0. Binary one or binary two in the next address select control bits of the pipeline register cause a conditional selection at the address multiplexer via D_1 or D_2. Thus, a CONDITIONAL BRANCH can be executed. Binary three in the next address select portion of the pipeline register causes an UNCONDITIONAL BRANCH instruction to be executed via D_3.

When the overall machine timing is studied, it will be observed that the key difference between overlap fetching and non-overlap fetching involves the propagation delay of the microprogram memory. In the non-pipelined architecture, the microprogram memory propagation delay must be added to the propagation delay of all the other elements of the machine. In the overlap fetch architecture, the propagation delay associated with the next microprogram memory address fetch is a separate loop independent of the other portion of the machine.

SUBROUTINING IN MICROPROGRAMMING CONTROL

Thus far, we have examined the CONTINUE instruction as well as the CONDITIONAL AND UNCONDITIONAL JUMP instructions for overlap fetch. Just as in the programming of minicomputers and microcomputers, the advantages of SUBROUTINING can be realized in microprogramming. The idea here, of course, is that the same block of microcode (or even a single microinstruction) can be shared by several microinstruction sequences of a microcode. This results in the overall reduction in the total number of microprogram memory words required by the design. If we are to jump to a subroutine, what is required is the ability to store an address to which the subroutine should return when it has completed its execution. Examining the block diagram of Figure 7, we see the addition of a subroutine and loop (push/pop) stack (also called the file), and its associated stack pointer. The control signals required by the stack are an enable stack signal ($\overline{FE}$) which will be used to tell the file whenever we wish to perform a push or a pop, and a push/pop control (PUP) used to control the direction of the stack pointer (push or pop).

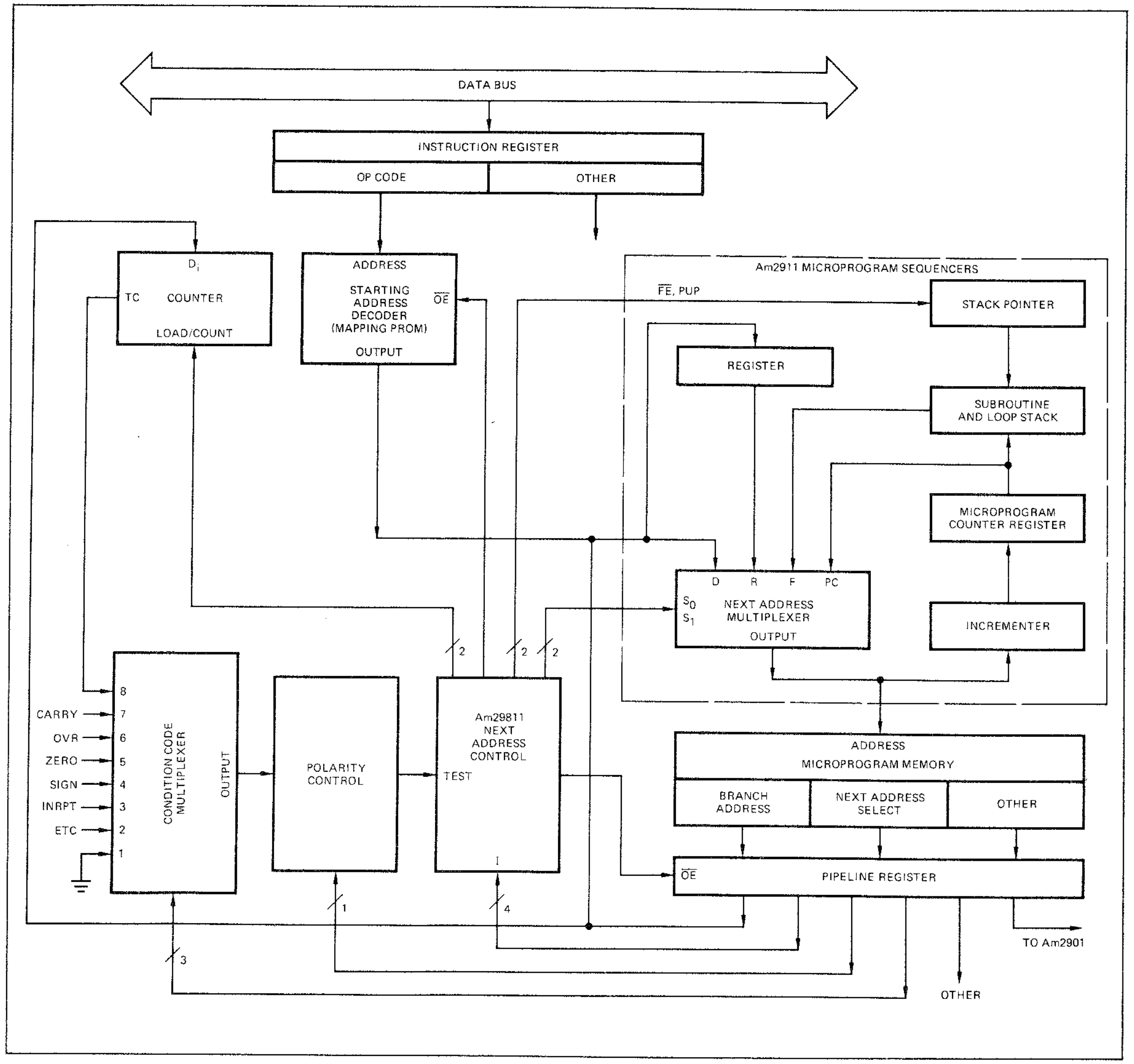


Figure 7. A Typical Computer Control Unit Using the Am2911 and Am29811.

In this architecture, the stack pointer always points to the address of the last microinstruction written to the file. This allows the "next address multiplexer" to read the file at any time via port F. When this selection is performed, the last word written on the stack will be the word applied to the microprogram memory. The condition code multiplexer of the previous example has also been replaced by a next address control unit. This next address control unit is the Am29811, which can execute 16 different next address control functions where most of these functions are conditional. Thus, the device has four instruction inputs as well as one condition code test input which is connected to the condition code multiplexer. Note also that the next address control field of the microprogram word has been expanded to a four-bit field. Outputs from the Am29811 are used to control the stack pointer and the next address multiplexer of the Am2911. In addition, the device has outputs to control the three-state enable of the pipeline register and the three-state enable of the starting address decode PROM. Also, the Am29811 has outputs to control a counter that can be used as a loop counter or event counter.

The 16 instructions associated with next address control for the Am29811 are listed in Table II-A and functionally described in Table II-B. As is easily seen by referring to Table II-A, three of the instructions in this set are associated with subroutining in microprogram memory. The first instruction of this set, is a simple conditional JUMP-TO-SUBROUTINE where the source of the subroutine address is in the pipeline register. The RETURN-FROM-SUBROUTINE instruction is also conditional and is used to return to the next microinstruction following the JUMP-TO-SUBROUTINE instruction. There is also a conditional JUMP-TO-ONE-OF-TWO-SUBROUTINES, where the subroutine address is either in the PIPELINE register or in the internal REGISTER in the Am2911. This instruction will be explained in more detail later.

TABLE II-A
Am29811 INSTRUCTION SET

MNEMONIC	I_3 I_2 I_1 I_0	INSTRUCTION
JZ	L L L L	Jump to Address Zero
CJS	L L L H	Conditional Jump-to-Subroutine with Jump Address in Pipeline Register.
JMAP	L L H L	Jump to Address at Mapping PROM Output.
CJP	L L H H	Conditional Jump to Address in Pipeline Register
PUSH	L H L L	Push Stack and Conditionally Load Counter
JSRP	L H L H	Jump-to-Subroutine with Starting Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
CJV	L H H L	Conditional Jump to Vector Address.
JRP	L H H H	Jump to Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
RFCT	H L L L	Repeat Loop if Counter is not Equal to Zero.
RPCT	H L L H	Repeat Pipeline Address if Counter is not Equal to Zero.
CRTN	H L H L	Conditional Return-from-Subroutine.
CJPP	H L H H	Conditional Jump to Pipeline Address and Pop Stack.
LDCT	H H L L	Load Counter and Continue.
LOOP	H H L H	Test End of Loop.
CONT	H H H L	Continue to Next Address.
JP	H H H H	Jump to Pipeline Register Address.

TABLE II-B
FUNCTIONAL DESCRIPTION OF Am29811 INSTRUCTION SET

MNEMONIC	INPUTS			OUTPUTS				
	INSTRUCTION I_3 I_2 I_1 I_0	FUNCTION	TEST INPUT	NEXT ADDR SOURCE	FILE	COUNTER	MAP-E	PL-E
JZ	L L L L	JUMP ZERO	X	D	HOLD	L L*	H	L
CJS	L L L H	COND JSB PL	L	PC	HOLD	HOLD	H	L
			H	D	PUSH	HOLD	H	L
JMAP	L L H L	JUMP MAP	X	D	HOLD	HOLD	L	H
CJP	L L H H	COND JUMP PL	L	PC	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
PUSH	L H L L	PUSH/COND LD CNTR	L	PC	PUSH	HOLD	H	L
			H	PC	PUSH	LOAD	H	L
JSRP	L H L H	COND JSB R/PL	L	R	PUSH	HOLD	H	L
			H	D	PUSH	HOLD	H	L
CJV	L H H L	COND JUMP VECTOR	L	PC	HOLD	HOLD	H	H
			H	D	HOLD	HOLD	H	H
JRP	L H H H	COND JUMP R/PL	L	R	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
RFCT	H L L L	REPEAT LOOP, CNTR ≠ 0	L	F	HOLD	DEC	H	L
			H	PC	POP	HOLD	H	L
RPCT	H L L H	REPEAT PL, CNTR ≠ 0	L	D	HOLD	DEC	H	L
			H	PC	HOLD	HOLD	H	L
CRTN	H L H L	COND RTN	L	PC	HOLD	HOLD	H	L
			H	F	POP	HOLD	H	L
CJPP	H L H H	COND JUMP PL & POP	L	PC	HOLD	HOLD	H	L
			H	D	POP	HOLD	H	L
LDCT	H H L L	LOAD CNTR & CONTINUE	X	PC	HOLD	LOAD	H	L
LOOP	H H L H	TEST END LOOP	L	F	HOLD	HOLD	H	L
			H	PC	POP	HOLD	H	L
CONT	H H H L	CONTINUE	X	PC	HOLD	HOLD	H	L
JP	H H H H	JUMP PL	X	D	HOLD	HOLD	H	L

L = LOW H = HIGH X = Don't Care DEC = Decrement *LL = Special Case

TYPICAL COMPUTER CONTROL UNIT ARCHITECTURE USING THE Am2911 AND Am29811

The microprogram memory control unit block diagram of Fig. 7 is easily implemented using the Am2911 and Am29811. This architecture provides a structured state machine design capable of executing many highly sophisticated next address control instructions. The Am2911 contains a next address multiplexer that provides four different inputs from which the address of the next microinstruction can be selected. These are the direct input (D), the register input (R), the program counter (PC), and the file (F). The starting address decoder (mapping PROM) output and the pipeline register output are connected together at the D input to the Am2911 and are operated in the three-state mode.

The architecture of Figure 7 shows an instruction register capable of being loaded with a machine instruction word from the data bus. The op code portion of the instruction is decoded using a mapping PROM to arrive at a starting address for the microinstruction sequence required to execute the machine instruction. When the microprogram memory address is to be the first microinstruction of the machine instruction sequence, the Am29811 next address control unit selects the multiplexer D input and enables the three-state output from the mapping PROM. When the current microinstruction being executed is selecting the next microinstruction address as a JUMP function, the JUMP address will be available at the multiplexer D input. This is accomplished by having the Am29811 select the next address multiplexer D input and also enabling the three-state output of the pipeline register branch address field. The register enable input to the Am2911 is connected to ground so that this register will always load the value at the Am2911 D input. The value at D is clocked into the Am2911's register (R) at the end of the current microcycle, which makes the D value of *this* microcycle available as the R value of the *next* microcycle. Thus, by using the branch address field of two sequential microinstructions, a conditional JUMP-TO-ONE-OF-TWO-SUBROUTINES or a conditional JUMP-TO-ONE-OF-TWO-BRANCH-ADDRESSES can be executed by either selecting the D input or the R input of the next address multiplexer.

When sequencing through continuous microinstructions in microprogram memory, the program counter in the Am2911 is used. Here, the Am29811 simply selects the PC input of the next address multiplexer. In addition, most of these instructions enable the three-state outputs of the pipeline register associated with the branch address field, which allows the register within the Am2911 to be loaded.

The 4 x 4 stack in the Am2911 is used for looping and subroutining in microprogram operations. Up to four levels of subroutines or loops can be nested. Also, loops and subroutines can be intermixed as long as the four-word depth of the stack is not exceeded.

The 16 instructions for next address control were defined in Table II. Table III shows the truth table for the Am29811. There are four instruction inputs, one test input, and eight outputs. The outputs are connected at various points throughout the computer control unit architecture as shown in Figure 7.

DETAILED DESCRIPTION OF THE Am2911 AND Am29811 IN A COMPUTER CONTROL UNIT OR STRUCTURED STATE MACHINE

The detailed connection diagram of a straight-forward computer control unit is shown in Figure 8 (A pull-out at the back of this book). This design features all of the next address control functions described previously in this paper. In addition, a few features have also been added.

Referring to Figure 8, the instruction register consists of two Am25LS377 Eight-Bit Registers with Clock Enable. These registers are designated as U1 and U2 and provides ability to selectively load a 16-bit instruction. This particular design assumes that the instruction word consists of an eight-bit op code as well as eight bits of other data. Therefore, the op code is decoded using three 256-word by 4-bit PROM's. The Am29761 has been selected for this function and is shown in Figure 8 as U3, U4 and U5.

The basic control function for the microprogram memory is provided by the Am2911's. In this design, three Am2911's (U6, U7, and U8) are used so that up to 4K words of microprogram memory can be addressed. The microprogram memory can consist of PROM's, ROM's, or RAM's, depending on the particular design and the point of its development. This particular design shows the capability of a 64-bit microword; however, the actual number of bits used will vary from design to design.

The pipeline register associated with the computer control unit consists of five integrated circuits designated U16, U17, U18, U19 and U20.

One of the features of the architecture depicted in Figure 8 is the event counter shown as U9, U10, and U11. This event counter consists of three Am25LS163's connected as a 12-bit counter. The counter can be parallel loaded with a 12-bit word from pipeline registers U18, U19, and U20. The multiplexer and D-type flip-flop (U21 and U22) at the counter overflow output (U9) is present to improve system cycle time and will be described in detail later.

This design also features a 16-input condition code multiplexer using two Am74S251's, which are designated U12 and U14. Condition code polarity control capability has been added to the design by using an Am74S158 Two-Input Multiplexer designated as U13. The W outputs and Y outputs from U12 and U14 have been connected together but only one set of outputs will be enabled at a time via the three-state control signal designated as R_{20} and $\overline{R_{20}}$. Since the Y output is inverting and the W output is non-inverting, the two-input multiplexer, U13, can be used to select the test condition as either inverting or non-inverting. This allows the test input on the Am29811 Next Address Control Unit, U15, to execute conditional instructions on either the inverted or non-inverted polarity of the test signal. For example, a CONDITIONAL BRANCH may be performed on either carry set or carry reset. Likewise, the same CONDITIONAL BRANCH might be performed on either the *sign* bit as a logic one or the *sign* bit as a logic zero. Note that the Am29811 Next Address Control Unit has eight outputs. Four outputs to control the Am2911's S_0, S_1, PUP, and $\overline{\text{FE}}$ inputs. Two outputs to control the three-state enables of the devices connected to the D inputs, i.e., a map enable (MAP E) to select the mapping PROMs and a pipeline enable (PL E) to enable the three-state Am2918 outputs which make up a 12-bit wide branch address field.The remaining two Am29811 outputs are for loading and enabling the Am25LS163 counters. $\overline{\text{CNT ENABLE}}$ from the Am29811 is active-LOW while the Am25LS163 counter

TABLE III
Am29811 TRUTH TABLE

MNEMONIC	FUNCTION	INPUTS					OUTPUTS							
							NEXT ADDR SOURCE		FILE		COUNTER			
		I_3	I_2	I_1	I_0	TEST	S_1	S_0	$\overline{\text{FE}}$	PUP	$\overline{\text{LOAD}}$	$\overline{\text{EN}}$	$\overline{\text{MAP E}}$	$\overline{\text{PL E}}$
	PIN NO.	14	13	12	11	10	4	5	3	2	6	7	1	9
JZ	JUMP ZERO	L	L	L	L	L	H	H	H	H	L	L	H	L
		L	L	L	L	H	H	H	H	H	L	L	H	L
CJS	COND JSB PL	L	L	L	H	L	L	L	H	H	H	H	H	L
		L	L	L	H	H	H	H	L	H	H	H	H	L
JMAP	JUMP MAP	L	L	H	L	L	H	H	H	H	H	H	L	H
		L	L	H	L	H	H	H	H	H	H	H	L	H
CJP	COND JUMP PL	L	L	H	H	L	L	L	H	H	H	H	H	L
		L	L	H	H	H	H	H	H	H	H	H	H	L
PUSH	PUSH/COND LD CNTR	L	H	L	L	L	L	L	L	H	H	H	H	L
		L	H	L	L	H	L	L	L	H	L	H	H	L
JSRP	COND JSB R/PL	L	H	L	H	L	L	H	L	H	H	H	H	L
		L	H	L	H	H	H	H	L	H	H	H	H	L
CJV	COND JUMP VECTOR	L	H	H	L	L	L	L	H	H	H	H	H	H
		L	H	H	L	H	H	H	H	H	H	H	H	H
JRP	COND JUMP R/PL	L	H	H	H	L	L	H	H	H	H	H	H	L
		L	H	H	H	H	H	H	H	H	H	H	H	L
RFCT	REPEAT LOOP, CTR ≠ 0	H	L	L	L	L	H	L	H	L	H	L	H	L
		H	L	L	L	H	L	L	L	L	H	H	H	L
RPCT	REPEAT PL, CTR ≠ 0	H	L	L	H	L	H	H	H	H	H	L	H	L
		H	L	L	H	H	L	L	H	H	H	H	H	L
CRTN	COND RTN	H	L	H	L	L	L	L	H	L	H	H	H	L
		H	L	H	L	H	H	L	L	L	H	H	H	L
CJPP	COND JUMP PL & POP	H	L	H	H	L	L	L	H	L	H	H	H	L
		H	L	H	H	H	H	H	L	L	H	H	H	L
LDCT	LD CNTR & CONTINUE	H	H	L	L	L	L	L	H	H	L	H	H	L
		H	H	L	L	H	L	L	H	H	L	H	H	L
LOOP	TEST END LOOP	H	H	L	H	L	H	L	H	L	H	H	H	L
		H	H	L	H	H	L	L	L	L	H	H	H	L
CONT	CONTINUE	H	H	H	L	L	L	L	H	H	H	H	H	L
		H	H	H	L	H	L	L	H	H	H	H	H	L
JP	JUMP PL	H	H	H	H	L	H	H	H	H	H	H	H	L
		H	H	H	H	H	H	H	H	H	H	H	H	L

L = Low
H = High

requires an active-HIGH enable, therefore $\overline{\text{CNT ENABLE}}$ from the Am29811 is passed through one section of the Two-Input Multiplexer (U13) for inversion. An alternative counter, the Am25LS169, has enable as active-LOW; therefore, this inversion through U13 is not required.

At this point, a discussion of the typical operation of this computer control unit is in order. First, bits 0-11 of the microprogram memory output word, are connected to the pipeline register designated U18, U19, and U20. The Am2918 has been selected for this portion of the pipeline register because of its continuous outputs and three-state outputs. The three-state outputs are connected to the D inputs of the Am2911 to provide a branch address whenever needed. These 12 bits are designated BR_0-BR_{11}. The Q outputs of these same Am2918's are designated R_0-R_{11} and are connected to the parallel load input of the Am25LS163 Counters. Thus, the counter can be loaded with any value between 0 and 4,095. Many designs will take advantage of R_0-R_{11} and use it as a general purpose field whenever the counter is not being loaded or a jump pipeline is not being performed. Using a microprogram memory field for more than one function (branch address and counter load value in this example) is called FORMATTING and will be covered in greater detail later. The other two devices in the pipeline register shown on the architecture of Figure 8 are U16 and U17. First, U17 receives four bits (12, 13, 14, and 15) from the microprogram memory to provide four-bit instruction field to the Am29811. This four-bit field, designated R_{12}-R_{15}, provides the actual next address control instruction for the computer control unit. R_{16} is the polarity control bit for the test input and is connected to the select input of the Am74S158 Two-Input Multiplexer. When R_{16} is LOW, the signal at the Am29811 test input will be inverted, but when R_{16} is HIGH, the test input will be non-inverted.

The Am74S175 has been used as part of the pipeline register (U16) because it has both inverting and non-inverting outputs. Signals R_{17}, R_{18}, and R_{19} are used to control the One-of-Eight Multiplexer (U12 and U14) A, B, and C inputs. Pipeline register output R_{20} and $\overline{R_{20}}$ are used to enable either the U12 outputs or the U14 outputs such that a one-of-sixteen multiplexer function is implemented. In this design, the TEST 0 input of U14 is connected to ground. This provides a convenient path for converting any of the conditional instructions to non-conditional instructions. That is, any of the conditional instructions can be executed unconditionally by selecting the TEST 0 input which is connected to ground and forcing the polarity control to either the inverting or non-inverting condition. This allows the execution of unconditional JUMP, unconditional JUMP-TO-SUBROUTINE, and unconditional RETURN-FROM-SUBROUTINE instructions.

Bit 21 from the microprogram memory utilizes a flip-flop in U17 as part of the pipeline register. This output, R_{21}, is used as the enable input to the instruction register. Needless to say, other techniques for encoding this enable function in a formatted field could be provided.

UNDERSTANDING THE Am29811

The Am29811 is a next-address control unit for use in conjunction with the Am2911 Bipolar Microprogram Sequencer. This device provides 16 different instructions which control the selection of the next microinstruction to be executed. In addition, a test input is available on the device making many of these 16 instructions conditional. The Am29811 provides control of the S_0, S_1, PUSH/POP (PUP), and File Enable ($\overline{\text{FE}}$) inputs to the Am2911. The instruction set for the Am29811 was shown in Table II and the actual TRUTH TABLE was given in TABLE III.

Perhaps the best technique for understanding the Am29811 is to simply take each instruction and review its operation. In order to provide some feel for the actual execution of these instructions, Figure 9 is included and depicts examples of all 16 instructions.

The examples given in Figure 9 should be interpretted in the following manner. The intent is to show microprogram flow as various microprogram memory words are executed. For example, the continue instruction, instruction number 14 as shown in Figure 9, simply means that the contents of microprogram memory word 50 is executed, then the contents of word 51 is executed. This is followed by the contents of microprogram memory word 52 and the contents of microprogram memory word 53. The microprogram location counter addresses used in the examples were *arbitrarily* chosen and have no meaning other than to show instruction flow. One exception to this is the first example, JUMP ZERO, which forces the microprogram location counter to address ZERO. Each dot refers to the time that the contents of the microprogram memory word is in the pipeline register. The instruction field being dealt with in these examples is that of the Am29811 Next Address Control Unit, bits I_0, I_1, I_2, and I_3, and are supplied via pipeline register U17 (Figure 8). While no special symbology is used for the conditional instructions, the text to follow will explain what the conditional choices are in each example.

It might be appropriate at this time to mention that AMD has a microprogram assembler called AMDASM, which has the capability of using the Am29811 instructions in symbolic representation. AMDASM's Am29811 instruction symbolics (or mnemonics) are given in Figure 9, in parenthesis – for each instruction, and were also shown in Table II.

The first instruction, number 0, is the JUMP ZERO function. As depicted in Figure 9, no matter what instruction is currently being executed, the address of the next instruction is 0 and the contents of word 0 will be executed next. It should be noted at this time that if the counter is also used, one half of an Am74S139 Decoder will be required to provide the proper connection for the ZERO input on the Am2911. This particular connection is shown in Figure 11 and will be discussed in greater detail in the section describing extended enable control. Normally, the JUMP ZERO instruction provides one technique for returning to a known address when the pipeline register is cleared. Many designs use this feature for power-up sequences and provide the power-up firmware beginning at microprogram memory word location 0. In these cases, the "zero" input of the Am2911's may be connected elsewhere and this instruction not used.

Instruction number 1 is a CONDITIONAL JUMP-TO-SUBROUTINE via the address provided in the pipeline register. As shown in Figure 9, the machine might have executed words at addresses 50, 51, and 52. When the contents of address 52 is in the pipeline register, the next-address control function is the CONDITIONAL JUMP-TO-SUBROUTINE. Here, if the condition is passed (TEST input HIGH), the next instruction executed will be the contents of microprogram memory location 90. If the test has failed (LOW at the TEST input), the JUMP-TO-SUBROUTINE will not be executed; the contents of microprogram memory location 53 will be executed instead. Thus, the CONDITIONAL JUMP-TO-SUBROUTINE instruction at location 52 will cause either location 90 or location 53 to be executed next. If the TEST input is such that location 90 is executed, value 53 will be pushed onto the Am2911 internal stack. This provides the return linkage for the machine when the subroutine beginning at location 90 is completed. In this example, the subroutine was completed at location 93 and a return-from-subroutine would be executed.

Instruction number 2 is the JUMP MAP instruction. This is an unconditional instruction which causes the mapping PROM outputs to be enabled so that the next microinstruction location is determined by the address supplied via the mapping PROM's. Normally, the JUMP MAP instruction is used at the end of the fetch sequence for the machine. In the example of Figure 9, microinstructions at locations 50, 51, 52, and 53 might have been the fetch sequence and at its completion at location 53, the jump map function would be contained in the pipeline register. This example shows the mapping PROM outputs to be 90; therefore, an unconditional jump to microprogram memory address 90 is performed.

Instruction number 3, CONDITIONAL JUMP PIPELINE, derives its branch address from the pipeline register branch address value (BR_0–BR_{11} in Figure 9). This instruction provides a technique for branching to various microprogram sequences depending upon the test condition inputs. Quite often, state machines are designed which simply execute tests on various inputs waiting for the condition to come true. When the true condition is reached, the machine then branches and executes a set of microinstructions to perform some function. This usually has the effect of resetting the input being tested until some point in the future. Figure 9 shows the conditional jump via the pipeline register address at location 52. When the contents of microprogram memory word 52 are in the pipeline register, the next address will either be location 53 or location 30 in this example. If the test is passed (TEST input is HIGH), the value currently in the pipeline register (30) will be selected. If the input being tested fails (TEST input is LOW), the next address selected will be contained in the microprogram counter which, in this example, is 53. This is exactly the same technique as described earlier in Figure 5 of this paper.

Instruction number 4 is the PUSH/CONDITIONAL LOAD COUNTER instruction, and is used primarily for setting up loops in microprogram firmware. In Figure 9, when instruction 52 is in the pipeline register, a PUSH will be made on the stack and the counter will be loaded based on the condition. When the stack is pushed, the value pushed is always the next sequential instruction address. In this case, the address is 53. If the TEST input is LOW, the counter is not loaded, but if the TEST input is HIGH, the counter will be loaded with the value contained in the pipeline register branch address field. Thus, a single microinstruction can be used to set up a loop to be executed a specific number of times. Instruction number 8 will describe how to use the pushed value and counter for looping.

Instruction number 5 is a CONDITIONAL JUMP-TO-SUBROUTINE via the Am2911 internal REGISTER or the con-

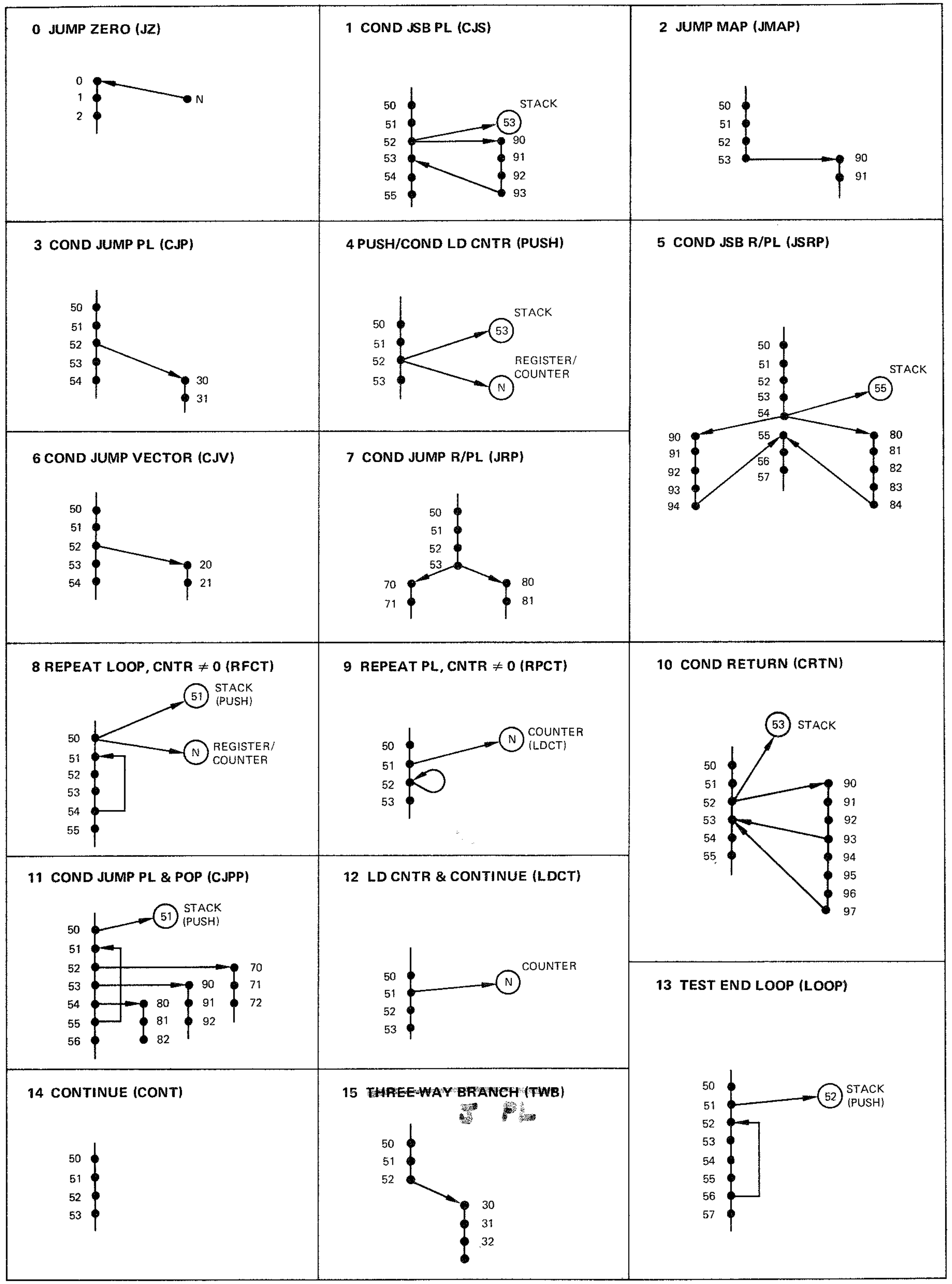


Figure 9. Am29811 Next Address Execution Examples.

tents of the PIPELINE register. As shown in Figure 9, a push is always performed and one of two subroutines executed. In this example, either the subroutine beginning at address 80 or the subroutine beginning at address 90 will be performed. A return-from-subroutine (instruction number 10) returns the microprogram flow to address 55. In order for this microinstruction control sequence to operate correctly, both the next address fields of instruction 53 and the next address fields of instruction 54 would have to contain the proper value. Let's assume that the branch address fields of instruction 53 contain the value 90 so that it will be in the Am2911 register when the contents of address 54 are in the pipeline register (remembering that during any current microcycle, R contains the value of D from the previous microcycle). Now if the TEST input is LOW, the contents of the register (value = 90) will select the address of the next microinstruction. If the TEST input is HIGH, the pipeline register contents (value = 80) will determine the address of the next microinstruction. Therefore, this instruction provides the ability to select one of two subroutines to be executed based on a test condition.

Instruction number 6 is a CONDITIONAL JUMP VECTOR instruction which provides the capability to take the branch address from a third source heretofore not discussed. In order for this instruction to be usable, the Am29811 three-state control fields must be decoded using one half of an Am74S139 Two-Line to Four-Line Decoder. When this instruction is executed, both $\overline{\text{MAP E}}$ and $\overline{\text{PL E}}$ outputs of the Am29811 are HIGH. Using $\overline{\text{MAP E}}$ and $\overline{\text{PL E}}$ as respective inputs to the Am74S139 2A and 2B inputs allows the Two-Line to Four-Line Decoder to select a third source via the $2Y_3$ output. U15, U16, U24, and U25 in Figure 11 clearly show this configuration. This instruction provides one technique for performing interrupt type branching at the microprogram level and is discussed in greater detail in the section describing extended enable control. Since this instruction is conditional, a HIGH at the TEST input of the Am29811 causes the next address to be taken from the vector source, while a LOW at the TEST input causes the next address to be taken from the program counter. In the example of Figure 9, if the CONDITIONAL JUMP VECTOR instruction is contained at location 52, execution will continue at vector address 20 if the TEST input is HIGH and the microinstruction at address 53 will be executed if the TEST input is LOW.

Instruction number 7 is a CONDITIONAL JUMP via the contents of the Am2911 internal REGISTER or the contents of the PIPELINE register. This instruction is very similar to instruction number 5; the conditional jump-to-subroutine via R or PL. The major difference between instruction number 5 and instruction number 7 is that no push on the stack is performed with the latter. Figure 9 depicts this instruction as a branch to one of two locations depending on the test condition. The example in Figure 9 assumes the pipeline register contains the value 70 when the contents of address 52 is being executed. As the contents of address 53 is clocked into the pipeline register, the value 70 is loaded into the register in the Am2911. The value 80 is available when the contents of address 53 are in the pipeline register. Thus, control is transferred to either address 70 or address 80 depending on the TEST input to the Am29811.

Instruction number 8 is the REPEAT LOOP, COUNTER ≠ ZERO instruction. This instruction is useful only if the Am25LS163 Counters are in the system. This instruction can be used to perform a timed wait or to execute some microinstruction a specific number of times. For example, a byte swap can be performed in a 16-bit machine by executing a single-bit rotate instruction eight times. The execution of this instruction assumes that the condition code multiplexer is selecting the counter TC output and applying it to the TEST input of the Am29811 with the proper polarity. If the counter is not equal to zero (all 1's is the zero condition for the Am25LS163), the test failed and the loop will be performed. This is accomplished by selecting the file output as the address of the next microinstruction. When this occurs, a count signal is applied to the Am25LS163 Counters. If the counter is equal to "zero" (TEST input HIGH), the loop is not repeated but rather the next microinstruction from the program counter is selected as the next address. In addition, the counter is not decremented but rather its current value is held. Since an exit has been made from the loop, the address on the stack is no longer needed and the file is maintained by performing a POP. This clears the file of the address that was PUSHed to set up the loop. In the example of Figure 8, the Am25LS163 Counter is used. It must be loaded with the one's complement of the desired number. The Am25LS163 is then incremented to the all 1's condition to execute the "counter not equal to zero" function. This will be discussed in more detail later.

An example of the repeat loop counter not equal to zero instruction is shown in Figure 9. In this example, location 50 most likely would contain a PUSH/Conditional Load counter instruction which would have caused address 51 to be PUSHed on the stack and the counter to be loaded with the proper value for looping the desired number of times. In the design depicted in Figure 8, the loop shown in Figure 9 will be executed N+2 times where N is the value loaded in the counter (one's complement actually loaded).

The REPEAT LOOP, COUNTER ≠ ZERO instruction is actually contained in the microinstruction at address 54. Here, depending on the test input, the sequence will either branch to address 51 or continue to address 55.

Single microinstruction loops provide a highly efficient capability for executing a specific microinstruction a fixed number of times. Examples include fixed rotates, byte swap, fixed point multiply, and fixed point divide.

Instruction number 9 is the REPEAT PIPELINE REGISTER, COUNTER ≠ ZERO instruction. This instruction is similar to instruction number 8 except that the branch address now comes from the pipeline register rather than the file. In some cases, this instruction may be thought of as a one-word file extention; that is, by using this instruction, a loop with the counter can still be performed when subroutines are nested four deep. This instruction's operation is very similar to that of instruction number 8 with the only difference being that on this instruction, a failed test condition causes the source of the next microinstruction address to be the D inputs. Also, when the test condition is passed, this instruction does not perform a POP because the stack is not being used.

In the example of Figure 9, the REPEAT PIPELINE, COUNTER ≠ ZERO instruction is instruction 52 and is shown as a single microinstruction loop. The address in the pipeline register would be 52. Most likely, instruction 51 in this example would be the Load Counter and Continue instruction (number 12) so that the counter is loaded properly. While the example shows a single microinstruction loop, by simply changing the address in a pipeline register, multi-instruction loops could be performed in this manner for a fixed number of times as determined by the counter.

Instruction number 10 is the conditional RETURN-FROM-SUBROUTINE instruction. As the name implies, this instruction is used to branch from the subroutine back to the next microinstruction address following the subroutine call. Since this instruction is conditional, the return is only performed if the test is passed (TEST input HIGH). If the test is failed (TEST input LOW), the next microinstruction in this subroutine sequence is performed. The example in Figure 9 depicts the use of the conditional RETURN-FROM-SUBROUTINE instruction in both the conditional and unconditional mode. This example shows a jump-to-subroutine at instruction 52 where control is transferred to instruction 90. At instruction 93, a conditional RETURN-FROM-SUBROUTINE instruction is performed. If the TEST input is true, the stack is accessed and the program will transfer to the next instruction at address 53. If the TEST input is false, the next microinstruction at address 94 will be executed. The program will continue to instruction 97 where the subroutine is complete. To perform an unconditional RETURN-FROM-SUBROUTINE, the conditional RETURN-FROM-SUBROUTINE instruction is executed and the condition code is forced to the pass state by selecting the known signal at the condition code multiplexer (Test 0 of U14 in Figure 8), thus forcing the TEST input to be HIGH.

Instruction number 11 is the CONDITIONAL JUMP PIPELINE register address and POP stack instruction. This instruction provides another technique for loop termination and stack maintenance. The example in Figure 9 shows a loop being performed from instruction 55 back to instruction 51. Instruction 52, instruction 53, and instruction 54 are all conditional JUMP and POP instructions. At instruction 52, if the TEST input is passed (true), a branch will be made to instruction 70 and the stack will be properly maintained via a POP. Should the test input be false, instruction 53 (the next sequential instruction) will be executed. Likewise, at instruction 53, either instruction 90 or instruction 54 will be subsequently executed, respective to the test being true or false. Instruction 54 follows the same rules, going to either instruction 80 or instruction 55. An instruction sequence as described here, using the CONDITIONAL JUMP PIPELINE and POP instruction, is very useful when several inputs are being tested and the microprogram is looping waiting for any of the inputs being tested to come true before proceeding to another sequence of instruction.

Instruction 12 is the LOAD COUNTER and CONTINUE instruction, which simply enables the counter to be loaded with the value at its parallel inputs. These inputs are normally connected to the pipeline branch address field which (in the architecture being described here) serves to supply either a branch address or a counter value depending upon the microinstruction being executed. In this case, the instruction is a load counter and the value contained in the pipeline branch address register is parallel loaded into the Am25LS163 Counters. As mentioned earlier in this paper, Am25LS169 Counters, connected in the down count mode, may be used in place of the Am25LS163's and this connection is shown in Figure 10. Note that the counter enable signal, $\overline{\text{CNT ENABLE}}$ of U15, is not inverted through U13 as when using the Am25LS163 Counters.

Figure 10 shows the carry/borrow output of the last counter stage connected to $1C_2$ of an Am25LS153 Four-Input multiplexer. When the counters are enabled for counting ($\overline{\text{CNT ENABLE}}$), the A input to the Am25LS153 is pulled LOW and the carry/borrow from the last counter (U9) is gated to the D_0 input of a Am74S174-type D flip-flop (U22). This has the effect of pipelining the terminal count signal so that carry/borrow propagation delay through the counters does not become a factor in machine timing. The result of this pipelining is that one additional count is added to the count cycle. If the value N is loaded into the Am25LS169 counter, the loop or sequence will be executed N+2 times. That is, a loop count of 98 loaded into the counter and a Repeat Loop Counter Not Equal to Zero instruction performed will cause the instruction in the loop to be executed 100 times. Still referring to Figure 10, when counter load (CNT LOAD) and counter enable (CNT ENABLE) are either both LOW or both HIGH, the Q_0 output of U22 is routed back through U21 (via $1C_0$ or $1C_3$), thus holding the current pipelined counter state. The capability of branching out of a counter controlled microprogram loop before the counter reaches zero exists, which means that the counter pipeline, U22, could contain a HIGH or a LOW at the time of the exit. Therefore, the signal $\overline{\text{CNT LOAD}}$ gets routed to the B input of U21 to force a non-zero (HIGH) condition into U22 (via the fixed HIGH at $1C_1$ on U21) when the counter is loaded. This prevents a "zero-true" condition test on the first test following a LOAD counter. Pre-loading U22 to a HIGH condition on $\overline{\text{CNT LOAD}}$ also insures a proper "two" count when the counter is loaded with zero, i.e.; the instruction will be executed N+2 times where N is the load value in the counter. It should be especially noted that when the Am25LS163 Counters are being used as shown in Figure 8, the one's complement of the value N is loaded into the counters. The Am25LS163 only counts up, therefore, each "DEC" (decrement) instruction causes the counter to increment and the "zero" condition at the carry out (CO) is reached when the counter contains all one's (HIGH's). In any event, the result is the execution of N+2 "decrement" (DEC) cycles.

Instruction number 13 is the TEST END-OF-LOOP instruction, which provides the capability of exiting a loop at the bottom; that is, this is a conditional instruction that will cause the microprogram to loop, via the file, if the condition is false and to continue to the next sequential instruction if true. The example in Figure 9 shows the TEST END-OF-LOOP microinstruction at address 56. If the test input is false (logic LOW), the microprogram will branch to address 52. Address 52 is on the stack because a PUSH instruction was executed at address 51. If the test input is true at instruction 56, the loop is terminated and the next sequential microinstruction at address 57 is executed, which also causes the stack to be POPed; thus accomplishing the required file maintenance.

Instruction number 14 is the CONTINUE instruction, which simply causes the program counter in the Am2911 to increment so that the next sequential microinstruction is executed.

Instruction number 15 is an unconditional JUMP PIPELINE REGISTER instruction. This provides the ability to unconditionally branch to any address contained in the branch address field of the microprogram. Thus, an unconditional N-way branch can be performed. Use of this instruction as opposed to a forced conditional jump pipeline instruction simply allows the condition code multiplexer select field to be shared (formatted) with other functions.

SKIP INSTRUCTION AND LOOP INSTRUCTION

In recent months, there has been some discussion in various microprogramming groups concerning the desirability of the "SKIP" instruction in microprogram control. It is generally believed that there are more powerful techniques for accomplishing this function with faster speed and fewer microinstructions. For example, if the intent of the "SKIP" instruction is to skip around a branch instruction, then the conditional jump is a more powerful technique. If the "SKIP" instruction is intended to determine only whether the intervening instruction is executed or not, then the conditional jump-to-subroutine instruction becomes the more powerful approach and saves overall memory space. In fact, for every example of potential uses of the "SKIP" instruction, the Am29811/Am2911 instruction set provides a more powerful technique for accomplishing the stated task.

One of the most important features of the Am29811/Am2911 instruction set lies in the ability of these devices to execute *single microinstruction loops.* Not only is this extremely powerful, but is critical if the minimum microprogram memory space is to be used in such algorithms as multiply and divide. Also, single microinstruction subroutines can be performed using these devices and this provides a sophisticated technique for saving microprogram memory while performing the minimum number of execution cycles.

A HIGH PERFORMANCE COMPUTER CONTROL UNIT

The high performance CCU (Figure 11) is of a similar basic design as the previously described CCU. The major differences are, referring to Figure 11, the addition of an extended enable control (U16), a vector input (U24 and U25), and an Am29803 16-way Branch Control Unit (U23). These performance enhancements are more related to function than to actual circuit speed. The use of these enhancements by the microprogram provides greater flexability in controlling a machines environment, and can reduce the microinstruction count required to perform a particular task, which has the effect of increasing overall system throughput.

In describing this high performance CCU design, those sections which remain unchanged from the previous description (Figure 8), will not be covered again. This includes the mapping PROMS, sequencer, Am29811, counter, condition test inputs and associated polarity control, and the pipeline register. The areas that will be covered are: extended enable control (U16), Vector inputs (U24 and U25), and the Am29803 16-way Branch Control Unit (U23).

Extended Enable Control

Extended enable control is accomplished via an Am74S139 dual two-to-four line decoder in conjunction with the Am29811 next address control unit. In Figure 8, $\overline{\text{PL E}}$ and $\overline{\text{MAP E}}$ of the Am29811 were connected directly to the components that they are to control (pipeline registers and mapping PROMS, respectively). Likewise, $\overline{\text{CNT LOAD}}$ and $\overline{\text{CNT ENABLE}}$ are connected directly to the counters that they control (with the exception that $\overline{\text{CNT ENABLE}}$ requires inversion when using Am25LS163 counters). In Figure 11, $\overline{\text{PL E}}$, $\overline{\text{MAP E}}$, $\overline{\text{CNT LOAD}}$, and $\overline{\text{CNT ENABLE}}$ go to the inputs of the Am74S139 two-to-four line decoder (U16). When either $\overline{\text{PL E}}$ or $\overline{\text{MAP E}}$ is LOW, then either $2Y_1$ or $2Y_2$ of U16 is LOW and either the pipeline branch address registers or mapping PROMS are enabled. If both $\overline{\text{PL E}}$ and $\overline{\text{MAP E}}$ are HIGH, then output $2Y_3$ of U16 is LOW enabling the three state outputs of U24 and U25 which are alternate microprogram starting address decoders (alternate mapping PROMS), and called VECTOR INPUT in this design. Likewise, $\overline{\text{CNT LOAD}}$ and $\overline{\text{CNT ENABLE}}$ follow the same rules, enabling the counter to load or count via $1Y_1$ and $1Y_2$ of U16.

Vector Input

The "Vector Input" provides the system designer with a powerful next starting address control. For example, one possible use might be as an interrupt vector. For instance, use the "Interrupt Request" output of an Am2914 Vectored Priority Interrupt Controller (or group of Am2914's) as an input to one of the conditional test inputs of multiplexers (U12 or U14). Then connect the Am2914 Vector Out lines to the vector mapping PROMS (Vector input U24 and U25). The microprogram then could, at the appropriate time, test for a pending interrupt and if present, jump in microprogram memory directly to the routine which handles the specific interrupt as requested via the Am2914 Vector Output lines. This routine will take the proper steps to preserve the status of the interrupt system, and then will service the interrupt. This is one of many possible uses for the Vector Input. Other possible uses include both hardware and software "TRAP" routines and so forth. As can be seen, the design presented here uses the Vector Enable line (output $2Y_3$ of U16) to enable an alternate starting address input at the Am2911. This, however, does not preclude the use of other devices in place of mapping PROMS as the D-input vector source.

Am29803 16-Way Branch Control Unit

The Am29803 provides 16-way branch control when used in conjunction with the Am2909 bipolar microprocessor sequencer, and is shown as U23 in Figure 11 with its pipeline register U22. The Am29803 has four TEST-inputs, four INSTRUCTION-inputs, four OR-outputs, and an enable control. The four OR-outputs connect directly to the Am2909 OR-inputs (U8 in Figure 11). The four INSTRUCTION-inputs to the Am29803 provide control over the TEST-inputs and OR-outputs, and are provided by the microprogram via the pipeline register U22 (Figure 11).

Basically, the INSTRUCTION-inputs (I_0—I_3) provide sixteen instructions (0—F_{16}) which can select sixteen possible combinations of the TEST-inputs and provide a specific output on the OR-outputs depending upon the state of the inputs being tested. (The subscript 16 refers to base 16). All possible combinations of INSTRUCTION-inputs, TEST-inputs, and OR-outputs are shown in Table IV.

Note that instruction zero does not test any inputs (a disable instruction). Instructions 1, 2, 4, and 8 test one input and can cause a branch to one of two words. Instructions 3, 5, 6, 9, 10 and 12 test two inputs and can jump to one of four words (a 4-word page). Instructions 7, 11, 13 and 14 test three inputs and can jump on an eight word page. Instruction number 15 tests all four inputs and the result can jump to any word on a sixteen word page.

USING THE Am29803

In the architecture of Figure 11, the Am29803 allows 2-way, 4-way, 8-way, or 16-way branching as determined by selectable combinations of the TEST-inputs. Referring to Table IV, the ZERO instruction (all instruction bits LOW) inhibits the testing of any TEST-inputs, thus providing LOW OR-outputs. Any single TEST-input selected (T_0, T_1, T_2 or T_3) will result in OR_0 being HIGH or LOW in correspondence with the

TABLE IV
Am29803 FUNCTION TABLE

Function	I_3	I_2	I_1	I_0	T_3	T_2	T_1	T_0	OR_3	OR_2	OR_1	OR_0
No Test	L	L	L	L	X	X	X	X	L	L	L	L
Test T_0	L	L	L	H	X X	X X	X X	L H	L L	L L	L L	L H
Test T_1	L	L	H	L	X X	X X	L H	X X	L L	L L	L L	L H
Test T_0 & T_1	L	L	H	H	X X X X	X X X X	L L H H	L H L H	L L L L	L L L L	L L H H	L H L H
Test T_2	L	H	L	L	X X	L H	X X	X X	L L	L L	L L	L H
Test T_0 & T_2	L	H	L	H	X X X X	L L H H	X X X X	L H L H	L L L L	L L L L	L L H H	L H L H
Test T_1 & T_2	L	H	H	L	X X X X	L L H H	L H L H	X X X X	L L L L	L L L L	L L H H	L H L H
Test T_0, T_1 & T_2	L	H	H	H	X X X X X X X X	L L L L H H H H	L L H H L L H H	L H L H L H L H	L L L L L L L L	L L L L H H H H	L L H H L L H H	L H L H L H L H
Test T_3	H	L	L	L	L H	X X	X X	X X	L L	L L	L L	L H
Test T_0 & T_3	H	L	L	H	L L H H	X X X X	X X X X	L H L H	L L L L	L L L L	L L H H	L H L H
Test T_1 & T_3	H	L	H	L	L L H H	X X X X	L H L H	X X X X	L L L L	L L L L	L L H H	L H L H
Test T_0, T_1 & T_3	H	L	H	H	L L L L H H H H	X X X X X X X X	L L H H L L H H	L H L H L H L H	L L L L L L L L	L L L L H H H H	L L H H L L H H	L H L H L H L H
Test T_2 & T_3	H	H	L	L	L L H H	L H L H	X X X X	X X X X	L L L L	L L L L	L L H H	L H L H
Test T_0, T_2 & T_3	H	H	L	H	L L L L H H H H	L L H H L L H H	X X X X X X X X	L H L H L H L H	L L L L L L L L	L L L L H H H H	L L H H L L H H	L H L H L H L H
Test T_1, T_2 & T_3	H	H	H	L	L L L L H H H H	L L H H L L H H	L H L H L H L H	X X X X X X X X	L L L L L L L L	L L L L H H H H	L L H H L L H H	L H L H L H L H
Test T_0, T_1, T_2 & T_3	H	H	H	H	L L L L L L L L H H H H H H H H	L L L L H H H H L L L L H H H H	L L H H L L H H L L H H L L H H	L H L H L H L H L H L H L H L H	L L L L L L L L H H H H H H H H	L L L L H H H H L L L L H H H H	L L H H L L H H L L H H L L H H	L H L H L H L H L H L H L H L H

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

polarity of the selected TEST-input. Selecting any combination of two TEST-inputs results in the outputs OR_0 and/or OR_1 being HIGH or LOW, following a mapped one-to-one relationship, i.e., OR_0 and OR_1 will follow the TEST-inputs, but no matter which pair of TEST-inputs are selected, their HIGH/LOW condition is mapped to the OR_0 and OR_1 outputs. Likewise, selecting any three TEST-inputs, will map their HIGH/LOW condition to the OR_0, OR_1, and OR_2 outputs. Selecting all four TEST-inputs, of course, causes a one-to-one relationship to exist between the HIGH/LOW conditions of the TEST-inputs and the corresponding OR-outputs. Refer to Table IV to verify the relationships between INSTRUCTION-inputs, TEST-input, and OR-output. It is very important that the mapping relationship between these signals be completely understood. When using the Am29803 TEST-OR capability as shown in Figure 11, the microprogrammer must position the applicable microcode within microprogram memory so that the low-order address bits are available for ORing. Sequencer instructions using the Am2909/2911 D-inputs (JRP, JSRP, JP, and CJS in particular) are ideally suited for the Am29803 TEST-OR capability. The jump-to-location, available via pipeline BR_0–BR_{11} or the Am2909/2911 register, can contain the address of a branch table. A branch table is merely a sequential series of unconditional jump instructions. The particular jump instruction executed is determined by the low-order address bits; that is, the first jump instruction in a branch table must start at a location in microprogram memory whose low-order address bit (or bits) is zero. If a single Am29803 TEST-input is selected (2-way branching) then only the least significant bit in the beginning branch table address needs to be zero. Two Am29803 TEST-inputs selected (4-way branching) requires that the branch table start on an address with the low-order two bits equal to zero; 8-way branching requires three low-order zero bits, and 16-way branching requires four low-order zero address bits. Understanding this branch control concept is really quite simple. The branch table is located in microprogram memory beginning at a location whose address has sufficient low-order zero bits to accommodate the number of selected Am29803 TEST-inputs. If, for instance, three TEST-inputs were selected, the first jump instruction in the branch table must be at an address whose low-order three bits are zero, such as address $0F8_{16}$. The second jump instruction in the branch table would begin in microprogram memory address $0F9_{16}$. The third jump at location $0FA_{16}$, the fourth at $0FB_{16}$, etc. Through all eight locations ($0F8_{16}$–$0FF_{16}$). Assume the following pipeline instruction (referring to Figure 11): (1) U22 selects three Am29803 TEST-inputs, (2) U18 instructs the Am29811 Next Address Controller to select the Am2909/2911 D-inputs, (3) U16 enables the pipeline branch address as the D source, and (4) U19, U20, and U21 supplies the address $0F8_{16}$ as the branch address. The Am29803 TEST-inputs will be ORed into the low-order three bit positions, thus providing a jump entry into the branch table *indexed* by the value of the OR bits. Each instruction in the branch table is usually a jump instruction, which allows the selection of a particular microcode routine determined by the value presented at the Am29803 TEST-inputs. These jump instructions are the first instruction of the particular sequence.

There are, of course, many other ways to use the Am29803 16-way Branch Control Unit. A variant of the use just described here is given in the next section showing some AMDASM microprogram assembler examples.

The microprogram memory address supplied via an Am2909 sequencer can be modified by the Am29803 16-way Branch Control Unit. Remember, however, that the microcode associated with this address modification relies on certain address bits being zero, therefore this microcode is not arbitrarily relocatable. The above discussion describes using the D-input and branching to provide low-order zero's to use the OR inputs. Through proper design, the Register, PC Counter, or File can be used equally well.

PROGRAMMING THE HIGH PERFORMANCE CCU WITH AMDASM

The high performance CCU shown in Figure 11, uses 26 bits of the microword. To provide ease in understanding the concepts, no FORMATTING is performed in the following examples, a 64 bit wide microinstruction will be assumed, although it is understood that the actual width of the microprogram depends upon the specific system architecture. Also, because this paper deals with only the CCU, the bits in the microword which control the ALU's and other functions will not be referred to.

Table V summarizes the CCU fields defined and the required parameters to be used in each field. Using the definitions in Table V, any microprogram can be easily assembled. Because the High Performance CCU is extremely versatile, it is very difficult to show all of the possibilities available for the user. However, some of its capabilities will be demonstrated in the following examples:

TABLE V
HIGH PERFORMANCE COMPUTER CONTROL UNIT ASSEMBLY DEFINITIONS

22–25	21	16–20	12–15	0–11	BIT NO.
Am29803 Instructions	Inst. Register	Test Select** and Polarity	Am29811 Instructions	Numerical Field*	Field Description
T0 T1 T01 T2 T02 T12 T3 T03 T13 T013 T23 T023 T123 T012 T0123 NOT	IN	The Test Number (1–14) in Decimal, and: CNTR for Test Select. (Unconditional by default) INV for Test Polarity (noninverted by default)	JZ CJS JMAP CJP PUSH JSRP CJV JRP RFCT RPCT CRTN CJPP LDCT LOOP CONT JP	Any 4 Digit (12 Bit) Octal Number	Parameters To Be Used

*One variable field

**Two variable field

```
; THIS IS AN AMDASM MICROPROGRAM ASSEMBLY EXAMPLE.
; AMDASM REQUIRES TWO PHASES; DEFINITION AND ASSEMBLY.
;
; FOLLOWING IS THE DEFINITION PHASE AND THE DEFINITIONS
; REFER TO FIGURE 11.
;
WORD     64                    ; DEFINE A 64 BIT MICROINSTRUCTION
;
;
; THE FIVE MAIN CCU FIELDS ARE AS FOLLOWS:
;
;           M0 –M11:  A 12 BIT NUMERICAL FIELD USED TO
;                     SUPPLY THE PIPELINE BRANCH ADDRESS
;                     OR COUNTER LOAD VALUE.
;           M12–M15:  THE AM29811 INSTRUCTION
;           M16–M20:  CONDITION CODE TEST SELECT & POLARITY CONTROL
;           M21     :  INSTRUCTION REGISTER READ-IN
;           M22–M25:  THE AM29803 INSTRUCTION
;
;
; DEFINE THE DEFAULT PIPELINE BRANCH FIELD.
; IT WILL FORCE THE MICROPROGRAM TO THE HIGHEST
; MICROPROGRAM MEMORY LOCATION IF LEFT IN DEFAULT FORM.
;
NUMB:     DEF      52X,  12V%Q#7777
;
;
; DEFINE THE CONDITIONAL TEST SELECT FIELD AND POLARITY CONTROL
; DEFAULTS ARE:  NONINVERTED AND UNCONDITIONAL.
; TESTS ARE ACTIVE LOW!
;
TEST:     DEF      43X,  4V%:D#0, 1VB#0,  16X
;
CNTR:     EQU      15          ; COUNTER ZERO TEST SELECT
INV:      EQU      B#1         ; POLARITY CONTROL
;
;
; DEFINE THE AM29811 NEXT ADDRESS CONTROL UNIT
; INSTRUCTION MNEMONICS.
;
JZ:       DEF      48X,  H#0,  12X     ; JUMP ZERO
CJS:      DEF      48X,  H#1,  12X     ; CONDITIONAL JUMP SUBROUTINE
JMAP:     DEF      48X,  H#2,  12X     ; JUMP MAP
CJP:      DEF      48X,  H#3,  12X     ; CONDITIONAL JUMP PIPELINE
PUSH:     DEF      48X,  H#4,  12X     ; PUSH/CONDITIONAL LOAD COUNTER
JSRP:     DEF      48X,  H#5,  12X     ; COND JUMP SUBROUTINE REGISTER/PIPELINE
CJV:      DEF      48X,  H#6,  12X     ; CONDITIONAL JUMP VECTOR
JRP:      DEF      48X,  H#7,  12X     ; CONDITIONAL JUMP REGISTER/PIPELINE
RFCT:     DEF      48X,  H#8,  12X     ; REPEAT FILE LOOP ON COUNTER .NE. ZERO
RPCT      DEF      48X,  H#9,  12X     ; REPEAT PIPELINE ON COUNTER .NE. ZERO
CRTN:     DEF      48X,  H#A,  12X     ; CONDITIONAL RETURN
CJPP:     DEF      48X,  H#B,  12X     ; CONDITIONAL JUMP PIPELINE & POP
LDCT:     DEF      48X,  H#C,  12X     ; LOAD COUNTER & CONTINUE
LOOP:     DEF      48X,  H#D,  12X     ; TEST END LOOP (CONDITIONAL LOOP ON FILE)
CONT:     DEF      48X,  H#E,  12X     ; CONTINUE
JP:       DEF      48X,  H#F,  12X     ; JUMP PIPELINE
```

```
;
;
; THE DEFAULT FOR DATA BUS READ-IN OF INSTRUCTION REGISTER IS DISABLE
;
DB:       DEF      42X,  1VB#1, 21X
IN:       EQU      B#0
;
; DEFINE THE AM29803 16–WAY BRANCH CONTROL UNIT
; INSTRUCTION MNEMONICS.
;
NOT:      DEF      38X,  H#0,  22X
T0:       DEF      38X,  H#1,  22X
T1:       DEF      38X,  H#2,  22X
T01:      DEF      38X,  H#3,  22X
T2:       DEF      38X,  H#4,  22X
T02:      DEF      38X,  H#5,  22X
T12:      DEF      38X,  H#6,  22X
T012:     DEF      38X,  H#7,  22X
T3:       DEF      38X,  H#8,  22X
T03:      DEF      38X,  H#9,  22X
T13:      DEF      38X,  H#A,  22X
T013:     DEF      38X,  H#B,  22X
T23:      DEF      38X,  H#C,  22X
T023:     DEF      38X,  H#D,  22X
T123:     DEF      38X,  H#E,  22X
T0123:    DEF      38X,  H#F,  22X
;
END                        ; END  OF  DEFINITION  PHASE
;
; BEGIN  ASSEMBLY  PHASE
;
;
```

```
; EXAMPLE 1.
;
; VISUALIZE A 16-BIT PROCESSOR IN A REAL-TIME ENVIRONMENT
; GATHERING AND MANIPULATING DATA. PART OF THIS DATA ARRIVES
; IN 8-BIT BYTES SO SWAPPING IS NECESSARY. ALSO, THERE ARE
; TWO CONTROL SIGNALS WHICH REQUIRE IMMEDIATE ATTENTION
; WHEN ACTIVE. ASSUME THAT THESE CONTROL SIGNALS ARE CONNECTED
; TO T2 AND T3 OF THE AM29803 16-WAY BRANCH CONTROL UNIT, FOLLOWING
; IS THE AMDASM OUTPUT FOR THIS EXAMPLE'S ASSEMBLY PHASE,
; WHICH INCLUDES THE SOURCE LISTING AND OUTPUT BIT PATTERN.
; IN THIS EXAMPLE, THE MICROPROGRAM STARTS AT LOCATION
; 0360 OCTAL. AS MENTIONED EARLIER, THE ALU PORTION OF
; THESE EXAMPLES IS NOT DEALT WITH.
;
;
;
0001   ORG  H#0F0
0002   SWAP:   NUMB 0006* & TEST , & PCLC & T0123
0003           RFCT & TEST CNTR , & T0123
0004           CJV & TEST , & T0123

0005   ORG  H#0F4
0006   ORTEST2:   TEST , & JPL & NUMB H#1F0 ;#2 HANDLER AT LOCATION 1F0

0007   ORG  H#0F8
0008   ORTEST3:   TEST , & JPL & NUMB H#2F0 ;#3 HANDLER AT LOCATION 2F0

0009   ORG  H#0FC
0010   ORTEST23:  TEST , & JPL & NUMB H#3F0 ;#2 AND#3 HANDLER AT LOC 3F0

0011   END

00F0 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX1111X00000 0100111111111001
00F1 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX1111X11110 1000XXXXXXXXXXXX
00F2 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX1111X00000 0110XXXXXXXXXXXX
00F4 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 1111000111110000
00F8 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 1111001011110000
00FC XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 1111001111110000
```

```
; EXAMPLE 2.
;
; ALIGNMENT CAN BE REALIZED IN ONE MICROINSTRUCTION. ASSUME
; THAT F3 OF THE MOST SIGNIFICANT ALU SLICE IS CONNECTED TO
; TEST 13 OF THE CONDITION MULTIPLEXERS. NOTE THAT NEGATIVE
; NUMBERS CAN BE ALIGNED IN THE SAME MANNER BY SIMPLY
; OMITTING THE VARIABLE "INV". ALSO, IF THE COUNTER IS CLEARED
; BEFORE STARTING ALIGNMENT, IT WILL CONTAIN THE NUMBER OF
; SHIFTS REQUIRED TO DO THE ALIGNMENT (OR THE COMPLIMENT
; IF USING AM25LS169 COUNTERS).
;
;
;
0001  ORG  Q#0770
0002  ALIGN:    NUMB 0770 & TEST 13, INV & RPCT   ;(ALU TO SHIFT UP)
0003  END

01F8 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11011 1001000111111000
```

```
; EXAMPLE 3.
;
; A DIVISION ROUTINE.  ASSUME F = 0 OF THE ALU IS CONNECTED TO
; TEST–12 (AND F3 TO TEST–13 AS BEFORE), AND SIXTEEN
; DIVISION STEPS ARE REQUIRED. IF THE FINAL REMAINDER IS NEGATIVE, IT MUST BE
; RESTORED BY ADDING IT TO THE DIVISOR. THE VECTOR INPUT IS SET UP
; FOR THE ERROR ROUTINE. NOTE USAGE OF THE AMDASM CONVENTION
; "$" TO DENOTE THE CURRENT PROGRAM COUNTER.
;
;
;
0001  ORG  Q#1000
0002  DIVIDE:   LDCT & TEST, INV & NUMB D#14%*      ;(ALU OUTPUTS DIVISOR)
0003            TEST 12, INV & CJV                  ;IF = 0:  ERROR
0004            RPCT & TEST CNTR, & NUMB $          ;LOOP
0005            TEST 13, INV & NUMB $+2 & CJP       ;IF  R <0, CORRECT
0006            TEST, & JMAP                        ;EXIT TO MAP
0007            TEST, & JMAP                        ;ALU ADDS REMAINDER TO DIVISOR, EXIT MAP
0008  END

0200 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00001 1100111111110001
0201 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11001 0110XXXXXXXXXXXX
0202 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11110 1001001000000010
0203 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11011 0011001000000101
0204 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 0010XXXXXXXXXXXX
0205 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 0010XXXXXXXXXXXX
```

There are, of course, different ways to define the assembly mnemonics and the user is encouraged to try and find the most suitable for him. The more sophisticated the definitions are, the easier it is to write the assembly statements. The above definitions and examples demonstrate how convenient it is to use AMDASM to program the High Performance Computer Control Unit. AMDASM is a microprogram assembler available on CSC Timesharing.

CCU TIMING

The minimum clock cycle that can be used in a CCU design is usually determined by the component delays along the longest "pipeline register clock to logic to pipeline register clock" path. At the beginning of any given clock cycle, data available at the output of microprogram memory, counter status, and any other data and/or status fields, are latched into their associated pipeline registers. At this point, all delay paths begin. Visual inspection will not always point out the longest signal delay path. The obviously long paths are a good place to start, but each definable path should be calculated on a component-by-component basis until the truely longest logic signal path is found. Referring to Figure 11, an obviously long path is from the counter pipeline register (U26); through the Am2911 sequencers; to the pipeline register at the output of microprogram memory. Example 1 in Figure 12 gives representative propagation delay calculations for this path in the CCU defined in Figure 11.

The propagation delay path shown as Example 2 in Figure 12 has a longer delay than that shown in Example 1. This fact generally would not have been known without actually adding together the propagation delays of the individual components in that path.

The delay path calculated in Example 2 assumes that a signal polarity change at the Am29811 TEST-input will not cause a signal fluctuation at the $\overline{\text{MAP E}}$ and/or the $\overline{\text{PL E}}$ outputs. Since the Am29811 uses a decoding matrix internally, we must assume that a "glitch" at the $\overline{\text{MAP E}}$ or $\overline{\text{PL E}}$ outputs is possible. For purposes of discussion, assume that a change at the Am29811's TEST-input also caused the $\overline{\text{MAP E}}$ and/or $\overline{\text{PL E}}$ outputs to "glitch". This means that the D-source select path must now be added to Example 2, and is shown as Example 3 in Figure 12.

Needless to say, the design engineer should identify the longest signal path within the CCU design by actual calculation. The longest, worst case propagation delay path must be known in order to set the minimum system clock cycle time. Once this path is identified, then changes to increase speed can be identified. For example, if the microprogram memory becomes 45ns, a 20ns improvement is realized.

FORMATTING IN MICROPROGRAM MEMORY

Many engineers designing a microprogrammed machine for the first time find difficulty in determining an approach to the microprogramming problem. At Advanced Micro Devices, we have found the following approach to be the most useful for engineers attempting their first microprogramming design job. This approach involves initially assuming only one format for the microprogram word will be used. That is, the machine architecture should be determined as required for the performance desired and the total word width of the microprogram memory temporarily ignored. This can result in a fairly wide microprogram word at the onset of the design. However, by taking this approach, the design engineer does not become immediately bogged down in making several trade-offs as to the various formats useful for the microprogram words. If the machine architecture is designed to achieve the performance required and the microprogram word is expanded such that bits are available to control the various portions of the machine as required, then the initial design task becomes straightforward.

After the initial architecture has been laid out, the design should proceed so that small groups of microinstructions are written for the key machine instructions. Particular emphasis should be placed on the machine instruction set which requires the highest use in the firmware operation. For example, particular emphasis should be placed on the fetch routine since it is used on every machine instruction cycle. Likewise, if the machine will typically execute many register-to-register arithmetic operations, the microprogram sequence used in these instructions should be reviewed carefully. As small sequences of firmware are written, it will become apparent that some fields of the microprogram might be shared in the memory. For example, perhaps the interrupt control microprogram bits and the A source operand microprogram bits never occur in the same microinstruction. Thus, if four bits are used for interrupt control and four bits are used for the "A" source operand select, then these four bits might use the same four-bit field in the memory. This would save a four-bit field throughout the entire microprogram memory addressing space. However, it will be necessary to determine which field is present in the pipeline register word. Thus, one bit may be dedicated in the entire microprogram word field to distinguish between format number 1 and format number 2. When this bit is a zero, the machine recognizes format 1 and applies these four bits to the source operand control. When this bit is a logic 1, the machine recognizes that format number 2 is in the pipeline register and applies this field to the interrupt control.

EXAMPLE 1

Signal U26, U12, U13, U15, U6, Microprogram Memory, and Pipeline is calculated as follows:

Component	Path	Propagation Delay
U26	CP-Q	12ns
U12	D-Y	12ns
U13	A/B-Y	6ns
U15	TEST-OUTPUT	55ns
U6, U7, U8	S_0, S_1-Y	40ns
Microprogram Memory	A-D	65ns
Pipeline	T_S	5ns

Total Propagation Delay: 195ns

EXAMPLE 2

Signal path U17, U14, U13, U15, U6, Microprogram Memory, and U17.

Component	Path	Propagation Delay
U17	CP-Q	17ns
U14	S-W/Y	21ns
U13	A/B-Y	6ns
U15	TEST-OUTPUT	55ns
U6, U7, U8	S_0, S_1-Y	40ns
Microprogram Memory	A-D	65ns
U17	T_S	5ns

Total Propagation Delay: 209ns

EXAMPLE 3

The same path as defined in Example 2 above but assuming that $\overline{\text{MAP E}}$ and/or $\overline{\text{PL E}}$ fluctuate when the polarity of U15's TEST-input changes.

Component	Path	Propagation Delay
U17	CP-Q	17ns
U14	S-W/Y	21ns
U13	A/B-Y	6ns
U15	TEST-OUTPUT	55ns
U16	A/B-Y	12ns
U19, U20, U21	OE	19ns
U6, U7, U8	D-Y	20ns
Microprogram Memory	A-D	65ns
U17	T_S	5ns

Total Propagation Delay: 220ns

Figure 12. Speed Calculation Examples

From this discussion, it should be apparent that four different formats might be utilized by the machine and a two-bit field used to identify which of the formats is in the pipeline register. There are designs where part of the pipeline contains multiple registers for use by the same microinstruction field. In this case, the two bits determining the microinstruction format can be decoded by a Two-Line to Four-Line decoder (such as the Am74LS139). The output from the decoder can enable a register with a clock enable function as on the Am25LS07 (four-bit), Am25LS08 (six-bit), or Am25LS377 (eight-bit) so that the microprogram memory word is loaded into the proper register for use by the machine.

The primary advantage of formatting the microprogram word is to save bits in width for the microprogram memory. Occasionally, as the formatting becomes more complex, it will be found that two fields are required simultaneously. This can result in an extra microinstruction being required to be able to perform all of the functions required by the microprogram control. Normally, anywhere from two to eight different formats can be found useful in the microprogramming of different machines.

For the design engineer doing his first microprogramming job, the emphasis should be on layout of the architecture of the machine to meet the required specification. Should the design accidentally use too many bits in microword width, it would be unfortunate but not catastrophic. The machine would still perform properly and meet specification and, in many cases, only one or two extra integrated circuits would have been used. Based on the cost of Field Programmable Read Only Memories, this cost is not unacceptable. Also, it is possible that such an approach may considerably reduce the overall design time of the machine. As the designer gains experience, he will become more proficient in formatting microprogram words and will soon find he can generate microprogram control architecture with reasonable speed.

TYPICAL Am2900 MICROPROCESSOR

In order to provide an overall view of a typical Am2900 Bipolar Microprocessor, the block diagram of Figure 13 is presented. Here, the computer control unit (CCU) as described in this Application Note is depicted. In addition, the typical connection scheme for the Am2901 Bipolar Microprocessor slices is shown. The four Am2901 devices in the block diagram form a typical 16-bit architecture. Also shown in Figure 13 is the general connection for the Am2914 Priority Interrupt Controller and the Am2930 Program Control Unit. The Am2914 is currently available while the Am2930 is scheduled for introduction in the first quarter of 1977.

The block diagram also shows the Am2917 as the bus interface unit. Note that the Am2917 can interface directly with a data bus and the drive levels and receive levels are such that the instruction register can be built using standard Low-Power Schottky devices. This is possible because the receiver threshold on the Am2917 is designed identically to the thresholds on standard power Schottky and the Low-Power Schottky devices.

SUMMARY

The Am2909 and Am2911 provide a powerful solution to the microprogram memory sequence control problem. These devices are particularly well suited for high performance computer control units are structured state machine designs using overlap fetch of the next microinstruction. The Am29811 Next Address Control Unit provides the most powerful instruction set currently available for next microinstruction control. The Am29811 is available in the 16-pin package while the Am2911 is available in the space-saving, 20-pin package. Three Am2911's and one Am29811 provide for addressability of 4K words of microprogram memory utilizing minimum board area.

The Am2909 and Am29803 combination allow for 16-way branching on a single microinstruction. This can improve throughput by a factor of four as well as saving many words of microprogram memory. In addition, the Advanced Micro Devices' Microprogram Assembler (AMDASM) is a highly powerful tool for use in conjunction with the instruction set of the Am29811.

All of the devices described in the application note are competively priced, currently available and multiple sourced. The devices are also available with specifications guaranteed over the full military temperature range.

SECTION 2

MICROPROGRAM CONTROL CIRCUITS

FUTURE PRODUCTS

Am2909 • Am2911

Microprogram Sequencers

DISTINCTIVE CHARACTERISTICS

- 4-bit slice cascadable to any number of microwords
- Internal address register
- Branch input for N-way branches
- Cascadable 4-bit microprogram counter
- 4 x 4 file with stack pointer and push pop control for nesting microsubroutines.
- Zero input for returning to the zero microcode word
- Individual OR input for each bit for branching to higher microinstructions (Am2909 only).
- Three-state outputs
- All internal registers change state on the LOW-to-HIGH transition of the clock
- Am2909 in 28-pin package
- Am2911 in 20-pin package

GENERAL DESCRIPTION

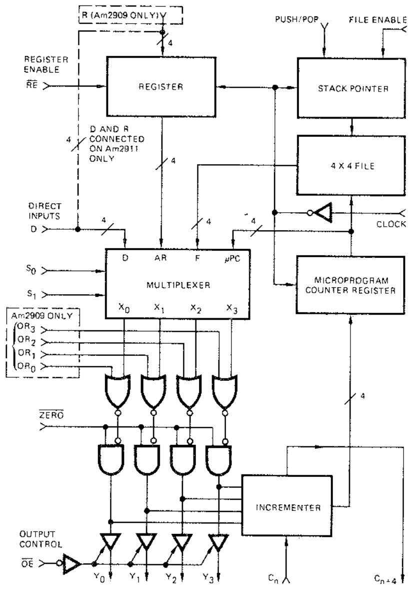
The Am2909 is a four-bit wide address controller intended for sequencing through a series of microinstructions contained in a ROM or PROM. Two Am2909's may be interconnected to generate an eight-bit address (256 words), and three may be used to generate a twelve-bit address (4K words).

The Am2909 can select an address from any of four sources. They are: 1) a set of external direct inputs (D); 2) external data from the R inputs, stored in an internal register; 3) a four-word deep push/pop stack; or 4) a program counter register (which usually contains the last address plus one). The push/pop stack includes certain control lines so that it can efficiently execute nested subroutine linkages. Each of the four outputs can be OR'ed with an external input for conditional skip or branch instructions, and a separate line forces the outputs to all zeroes. The outputs are three-state.

The Am2911 is an identical circuit to the Am2909, except the four OR inputs are removed and the D and R inputs are tied together. The Am2911 is in a 20-pin, 0.3" centers package.

TABLE OF CONTENTS

MICROPROGRAM SEQUENCER BLOCK DIAGRAM



ARCHITECTURE OF THE Am2909/Am2911

The Am2909/Am2911 are bipolar microprogram sequencers intended for use in high-speed microprocessor applications. The device is a cascadable 4-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 2.

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. On the Am2911, the direct inputs are also used as inputs to the register. This allows an N-way branch where N is any word in the microcode.

The Am2909/Am2911 contains a microprogram counter (μPC) that is composed of a 4-bit incrementer followed by a 4-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→μPC.) 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→μPC). 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 operations (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 subroutines can be performed. Since the stack is 4 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 $\overline{OE}$). 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/Am2911 feature 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.

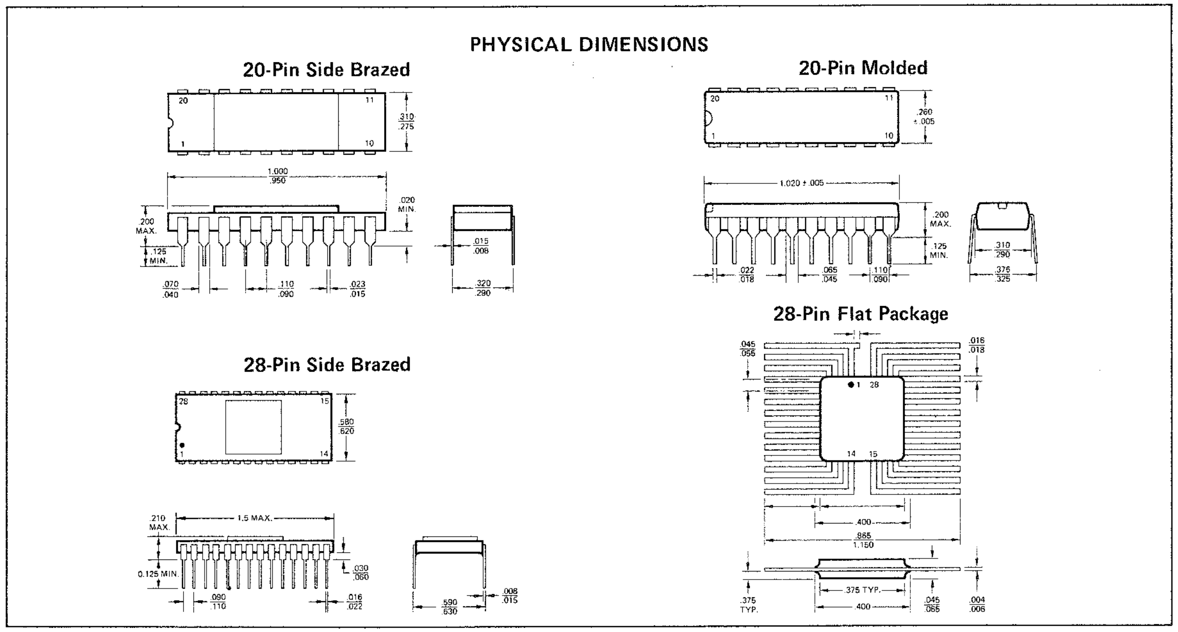
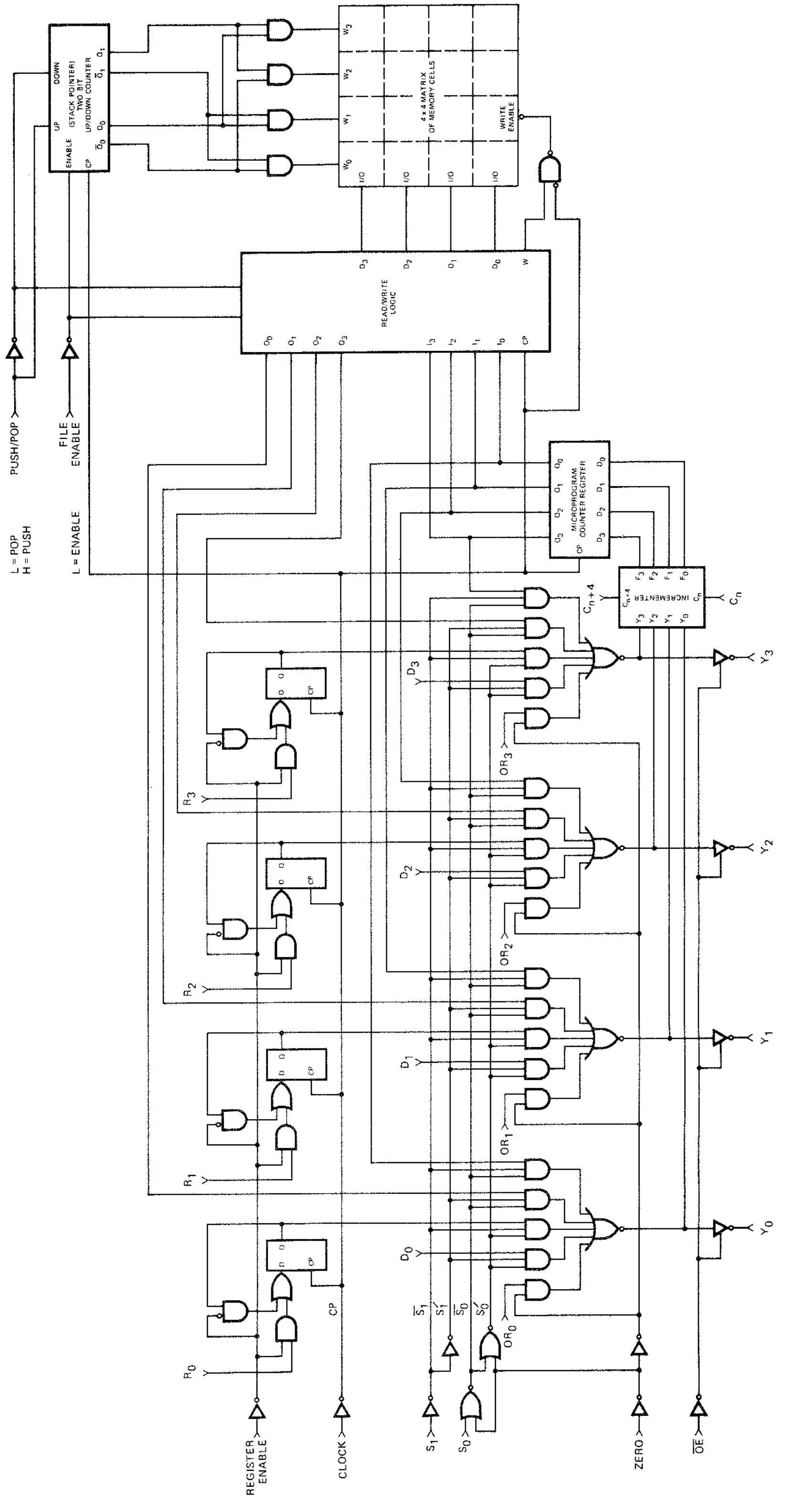


Figure 1.



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

Figure 2. Microprogram Sequencer Block Diagram.

DEFINITION OF TERMS

A set of symbols is used in this data sheet to represent various internal and external registers and signals used with the Am2909. Since its principle application is as a controller for a microprogram store, it is necessary to define some signals associated with the microcode itself. Figure 3 illustrates the basic interconnection of Am2909, memory, and microinstruction register. The definitions here apply to this architecture.

Inputs to Am2909/Am2911

S_1, S_0	Control lines for address source selection
$\overline{FE}$, PUP	Control lines for push/pop stack
$\overline{RE}$	Enable line for internal address register
OR_i	Logic OR inputs on each address output line
$\overline{ZERO}$	Logic AND input on the output lines
$\overline{OE}$	Output Enable. When $\overline{OE}$ is HIGH, the Y outputs are OFF (high impedance)
C_n	Carry-in to the incrementer
R_i	Inputs to the internal address register
D_i	Direct inputs to the multiplexer
CP	Clock input to the AR and μPC register and Push-Pop stack

Outputs from the Am2909/Am2911

Y_i	Address outputs from Am2909. (Address inputs to control memory.)
C_{n+4}	Carry out from the incrementer

Internal Signals

μPC	Contents of the microprogram counter
REG	Contents of the register
STK0-STK3	Contents of the push/pop stack. By definition, the word in the four-by-four file, addressed by the stack pointer is STK0. Conceptually data is pushed into the stack at STK0; a subsequent push moves STK0 to STK1; a pop implies STK3 → STK2 → STK1 → STK0. Physically, only the stack pointer changes when a push or pop is performed. The data does not move. I/O occurs at STK0.
SP	Contents of the stack pointer

External to the Am2909/Am2911

A	Address to the control memory
I(A)	Instruction in control memory at address A
μWR	Contents of the microword register (at output of control memory). The microword register contains the instruction currently being executed.
T_n	Time period (cycle) n

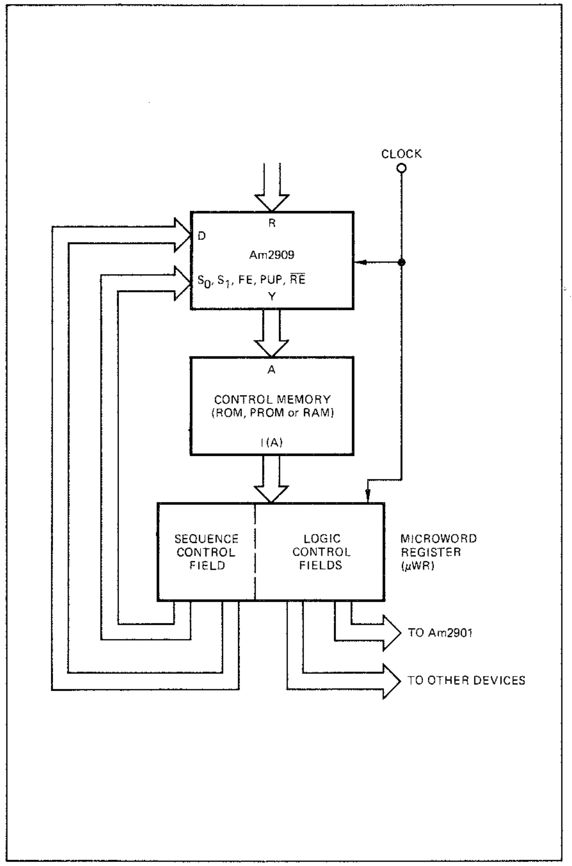
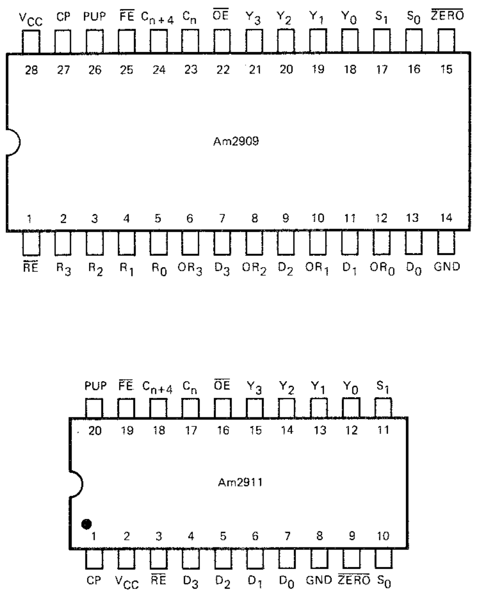


Figure 3. Microprogram Sequencer Control.

CONNECTION DIAGRAMS
Top Views


V_{CC} CP PUP $\overline{FE}$ C_{n+4} C_n $\overline{OE}$ Y_3 Y_2 Y_1 Y_0 S_1 S_0 $\overline{ZERO}$
28 27 26 25 24 23 22 21 20 19 18 17 16 15
Am2909
1 2 3 4 5 6 7 8 9 10 11 12 13 14
$\overline{RE}$ R_3 R_2 R_1 R_0 OR_3 D_3 OR_2 D_2 OR_1 D_1 OR_0 D_0 GND
PUP $\overline{FE}$ C_{n+4} C_n $\overline{OE}$ Y_3 Y_2 Y_1 Y_0 S_1
20 19 18 17 16 15 14 13 12 11
Am2911
1 2 3 4 5 6 7 8 9 10
CP V_{CC} $\overline{RE}$ D_3 D_2 D_1 D_0 GND $\overline{ZERO}$ S_0

Note: Pin 1 is marked for orientation.

Figure 4.

OPERATION OF THE Am2909/Am2911

Figure 5 lists the select codes for the multiplexer. The two bits applied from the microword register (and additional combinational logic for branching) determine which data source contains the address for the next microinstruction. The contents of the selected source will appear on the Y outputs. Figure 5 also shows the truth table for the output control and for the control of the push/pop stack. Figure 6 shows in detail the effect of S_0, S_1, $\overline{FE}$ and PUP on the Am2909. These four signals define what address appears on the Y outputs and what the state of all the internal registers will be following the clock LOW-to-HIGH edge. In this illustration, the microprogram counter is assumed to contain initially some word J, the address register some word K, and the four words in the push/pop stack contain R_a through R_d.

Address Selection

OCTAL	S_1	S_0	SOURCE FOR Y OUTPUTS	SYMBOL
0	L	L	Microprogram Counter	μPC
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)

H = High
L = Low
X = Don't Care

Figure 5.

CYCLE	S_1, S_0, $\overline{FE}$, PUP	μPC	REG	STK0	STK1	STK2	STK3	Y_{OUT}	COMMENT	PRINCIPLE USE
N N+1	0 0 0 0 —	J J+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	J —	Pop Stack	End Loop
N N+1	0 0 0 1 —	J J+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	J —	Push μPC	Set-up Loop
N N+1	0 0 1 X —	J J+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	J —	Continue	Continue
N N+1	0 1 0 0 —	J K+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	K —	Pop Stack; Use AR for Address	End Loop
N N+1	0 1 0 1 —	J K+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	K —	Push μPC; Jump to Address in AR	JSR AR
N N+1	0 1 1 X —	J K+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	K —	Jump to Address in AR	JMP AR
N N+1	1 0 0 0 —	J Ra+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	Ra —	Jump to Address in STK0; Pop Stack	RTS
N N+1	1 0 0 1 —	J Ra+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	Ra —	Jump to Address in STK0; Push μPC	
N N+1	1 0 1 X —	J Ra+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	Ra —	Jump to Address in STK0	Stack Ref (Loop)
N N+1	1 1 0 0 —	J D+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	D —	Pop Stack; Jump to Address on D	End Loop
N N+1	1 1 0 1 —	J D+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	D —	Jump to Address on D; Push μPC	JSR D
N N+1	1 1 1 X —	J D+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	D —	Jump to Address on D	JMP D

X = Don't care, 0 = LOW, 1 = HIGH, Assume C_n = HIGH
Note: STKO is the location addressed by the stack pointer.

Figure 6. Output and Internal Next-Cycle Register States for Am2909/Am2911.

Figure 7 illustrates the execution of a subroutine using the Am2909. The configuration of Figure 3 is assumed. The instruction being executed at any given time is the one contained in the microword register (μWR). The contents of the μWR also controls (indirectly, perhaps) the four signals S_0, S_1, $\overline{FE}$, and PUP. The starting address of the subroutine is applied to the D inputs of the Am2909 at the appropriate time.

In the columns on the left is the sequence of microinstructions to be executed. At address J+2, the sequence control portion of the microinstruction contains the comand "Jump to subroutine at A". At the time T_2, this instruction is in the μWR, and the Am2909 inputs are set-up to execute the jump and save the return address. The subroutine address A is applied to the D inputs from the μWR and appears on the Y outputs. The first instruction of the subroutine, I(A), is accessed and is at the inputs of the μWR. On the next clock transition, I(A) is loaded into the μWR for execution, and the return address J+3 is pushed onto the stack. The return instruction is executed at T_5. Figure 8 is a similar timing chart showing one subroutine linking to a second, the latter consisting of only one microinstruction.

CONTROL MEMORY

Execute Cycle	Microprogram	
	Address	Sequencer Instruction
	J−1	–
T_0	J	–
T_1	J+1	–
T_2	J+2	JSR A
T_6	J+3	–
T_7	J+4	–
	–	–
	–	–
	–	–
	–	–
	–	–
T_3	A	I(A)
T_4	A+1	–
T_5	A+2	RTS
	–	–
	–	–
	–	–
	–	–
	–	–
	–	–
	–	–

Execute Cycle		T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Clock Signals											
Am2909 Inputs (from μWR)	S_1, S_0 $\overline{FE}$ PUP D	0 H X X	0 H X X	3 L H A	0 H X X	0 H X X	2 L L X	0 H X X	0 H X X		
Internal Registers	μPC STK0 STK1 STK2 STK3	J+1 – – – –	J+2 – – – –	J+3 – – – –	A+1 J+3 – – –	A+2 J+3 – – –	A+3 J+3 – – –	J+4 – – – –	J+5 – – – –		
Am2909 Output	Y	J+1	J+2	A	A+1	A+2	J+3	J+4	J+5		
ROM Output	(Y)	I(J+1)	**JSR A**	I(A)	I(A+1)	**RTS**	I(J+3)	I(J+4)	I(J+5)		
Contents of μWR (Instruction being executed)	μWR	I(J)	I(J+1)	**JSR A**	I(A)	I(A+1)	**RTS**	I(J+3)	I(J+4)		

Figure 7. Subroutine Execution.

C_n = HIGH

CONTROL MEMORY

Execute Cycle	Microprogram	
	Address	Sequencer Instruction
	J−1	–
T_0	J	–
T_1	J+1	–
T_2	J+2	JSR A
T_9	J+3	–
	–	–
	–	–
	–	–
	–	–
T_3	A	–
T_4	A+1	–
T_5	A+2	JSR B
T_7	A+3	–
T_8	A+4	RTS
	–	–
	–	–
	–	–
	–	–
T_6	B	RTS
	–	–
	–	–

Execute Cycle		T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Clock Signals											
Am2909 Inputs (from μWR)	S_1, S_0 $\overline{FE}$ PUP D	0 H X X	0 H X X	3 L H A	0 H X X	0 H X X	3 L H B	2 L L X	0 H X X	2 L L X	0 H X X
Internal Registers	μPC STK0 STK1 STK2 STK3	J+1 – – – –	J+2 – – – –	J+3 – – – –	A+1 J+3 – – –	A+2 J+3 – – –	A+3 J+3 – – –	B+1 A+3 J+3 – –	A+4 J+3 – – –	A+5 J+3 – – –	J+4 – – – –
Am2909 Output	Y	J+1	J+2	A	A+1	A+2	B	A+3	A+4	J+3	J+4
ROM Output	(Y)	I(J+1)	**JSR A**	I(A)	I(A+1)	**JSR B**	**RTS**	I(A+3)	**RTS**	I(J+3)	I(J+4)
Contents of μWR (Instruction being executed)	μWR	I(J)	I(J+1)	**JSR A**	I(A)	I(A+1)	**JSR B**	**RTS**	I(A+3)	**RTS**	I(J+3)

Figure 8. Two Nested Subroutines. Routine B is Only One Instruction.

C_n = HIGH

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	–65°C to +150°C
Temperature (Ambient) Under Bias	–55°C to +125°C
Supply Voltage to Ground Potential	–0.5 V to +7.0 V
DC Voltage Applied to Outputs for HIGH Output State	–0.5 V to +V_{CC} max.
DC Input Voltage	–0.5 V to +7.0 V
DC Output Current, Into Outputs	30 mA
DC Input Current	–30 mA to +5.0 mA

OPERATING RANGE

P/N	Ambient Temperature	V_{CC}
Am2909/2911DC, PC	0°C to +70°C	4.75V to 5.25V
Am2909/2911DM, FM	–55°C to +125°C	4.50V to 5.50V

STANDARD SCREENING

(Conforms to MIL-STD-883 for Class C Parts)

Step	MIL-STD-883 Method	Conditions	Level	
			Am2909/Am2911PC, DC	Am2909/Am2911DM, FM
Pre-Seal Visual Inspection	2010	B	100%	100%
Stabilization Bake	1008	C 24-hour 150°C	100%	100%
Temperature Cycle	1010	C –65°C to +150°C 10 cycles	100%	100%
Centrifuge	2001	B 10,000 G	100% *	100%
Fine Leak	1014	A 5 x 10^{-8} atm-cc/cm^3	100% *	100%
Gross Leak	1014	C2 Fluorocarbon	100% *	100%
Electrical Test Subgroups 1 and 7	5004	See below for definitions of subgroups	100%	100%
Insert Additional Screening here for Class B Parts				

Group A Sample Tests
Subgroup 1
Subgroup 2
Subgroup 3
Subgroup 7
Subgroup 8
Subgroup 9
	5005	See below for definitions of subgroups	
			LTPD = 5
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
				
				LTPD = 5
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7

*Not applicable for Am2909PC or Am2911PC.

ADDITIONAL SCREENING FOR CLASS B PARTS

Step	MIL-STD-883 Method	Conditions	Level
			Am2909/Am2911DMB, FMB
Burn-In	1015	D 125°C 160 hours min.	100%
Electrical Test Subgroup 1 Subgroup 2 Subgroup 3 Subgroup 7 Subgroup 9	5004		 100% 100% 100% 100% 100%
Return to Group A Tests in Standard Screening			

ORDERING INFORMATION

Package Type	Temperature Range	Am2909 Order Number	Am2911 Order Number
Molded DIP	0°C to +70°C	AM2909PC	AM2911PC
Hermetic DIP	0°C to +70°C	AM2909DC	AM2911DC
Hermetic DIP	–55°C to +125°C	AM2909DM	AM2911DM
Hermetic Flat Pak	–55°C to +125°C	Am2909FM	–
Dice	0°C to +70°C	Am2909XC	–

GROUP A SUBGROUPS

(as defined in MIL-STD-883, method 5005)

Subgroup	Parameter	Temperature
1	DC	25°C
2	DC	Maximum rated temperature
3	DC	Minimum rated temperature
7	Function	25°C
8	Function	Maximum and minimum rated temperature
9	Switching	25°C
10	Switching	Maximum Rated Temeperature
11	Switching	Minimum Rated Temperature

ELECTRICAL CHARACTERISTICS OVER OPERATING RANGE (Unless Otherwise Noted)

Parameters	Description	Test Conditions (Note 1)			Min.	Typ. (Note 2)	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = MIN., V_{IN} = V_{IH} or V_{IL}	MIL	I_{OH} = −1.0mA	2.4			Volts
			COM'L	I_{OH} = −2.6mA	2.4			
V_{OL}	Output LOW Voltage	V_{CC} = MIN., V_{IN} = V_{IH} or V_{IL}	I_{OL} = 4.0mA				0.4	Volts
			I_{OL} = 8.0mA				0.45	
			I_{OL} = 12mA (Note 5)				0.5	
V_{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs			2.0			Volts
V_{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs	MIL				0.7	Volts
			COM'L				0.8	
V_I	Input Clamp Voltage	V_{CC} = MIN., I_{IN} = −18mA					−1.5	Volts
I_{IL}	Input LOW Current	V_{CC} = MAX., V_{IN} = 0.4V	C_n				−1.08	mA
			Push/Pop, $\overline{OE}$				−0.72	
			Others (Note 6)				−0.36	
I_{IH}	Input HIGH Current	V_{CC} = MAX., V_{IN} = 2.7V	C_n				40	μA
			Push/Pop				40	
			Others (Note 6)				20	
I_I	Input HIGH Current	V_{CC} = MAX., V_{IN} = 7.0V	C_n, Push/Pop				0.2	mA
			Others (Note 6)				0.1	
I_{OS}	Output Short Circuit Current (Note 3)	V_{CC} = MAX.			−40		−100	mA
I_{CC}	Power Supply Current	V_{CC} = MAX. (Note 4)				80	130	mA
I_{OZL}	Output OFF Current	V_{CC} = MAX., $\overline{OE}$ = 2.7V	V_{OUT} = 0.4V				−20	μA
I_{OZH}			V_{OUT} = 2.7V				20	

Notes:
1. For conditions shown as MIN. or MAX., use the appropriate value specified under Electrical Characteristics for the applicable device type.
2. Typical limits are at V_{CC} = 5.0V, 25°C ambient and maximum loading.
3. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.
4. Apply GND to C_n, R_0, R_1, R_2, R_3, OR_0, OR_1, OR_2, OR_3, D_0, D_1, D_2, and D_3. Other inputs open. All outputs open. Measured after a LOW-to-HIGH clock transition.
5. The 12mA guarantee applies only to Y_0, Y_1, Y_2 and Y_3.
6. For the Am2911, D_i and R_i are internally connected. Loading is doubled (to same values as Push/Pop).

SWITCHING CHARACTERISTICS OVER OPERATING RANGE

All parameters are guaranteed worst case over the operating voltage and temperature range for the device type.
(Grade C = 0°C to +70°C, 4.75V to 5.25V; Grade M = −55°C to +125°C, 4.5V to 5.5V)

TABLE I
MINIMUM CLOCK REQUIREMENTS

Minimum Clock LOW Time	50
Minimum Clock HIGH Time	30

TABLE II
MAXIMUM COMBINATORIAL PROPAGATION DELAYS

OUTPUTS / INPUTS	Y_i	C_{n+4}
$\overline{OE}$	25	–
$\overline{ZERO}$	35	45
OR_i	20	32
S_0, S_1	40	50
D_i	20	32
C_n	–	18

TABLE III
MAXIMUM DELAYS FROM CLOCK TO OUTPUTS

FUNCTIONAL PATH	GRADE	CLOCK TO Y_i	CLOCK TO C_{n+4}
Register (S_1 S_0 = LH)	C	48	58
	M	55	65
μ Program Counter (S_1 S_0 = LL)	C	48	58
	M	55	65
File (S_1 S_0 = HL)	C	70	80
	M	80	90

R_L = 2.0kΩ C_L = 15pF

TABLE IV
SET-UP AND HOLD TIME REQUIREMENTS

EXTERNAL INPUTS	t_s	t_h
$\overline{RE}$	20	5.0
R_i	15	2.0
PUSH/POP	20	5.0
$\overline{FE}$	20	0
C_n	15	0
D_i	35	0
OR_i	20	0
S_0, S_1	50	0
$\overline{ZERO}$	45	0

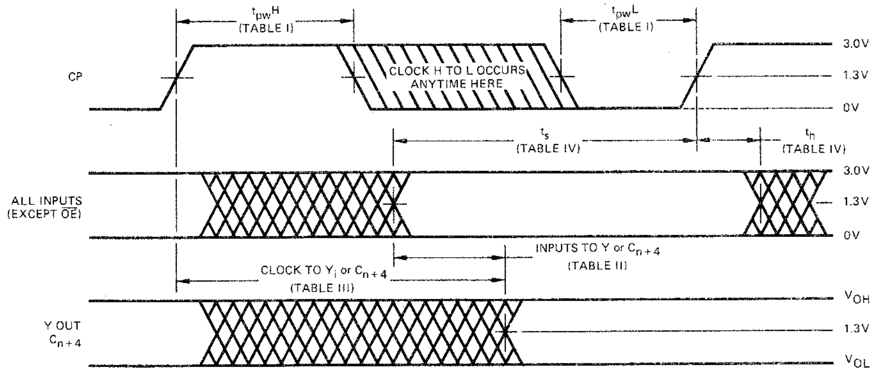


Figure 12. Switching Waveforms. See Tables for Specific Values.

Am29803

16-Way Branch Control Unit

DISTINCTIVE CHARACTERISTICS

- 16 separate instructions – 2, 4, 8, or 16-way branch in one microprogram execution cycle
- Four individual test inputs
- Four individual outputs for driving the four OR inputs on the Am2909 Microprogram Sequencer
- Provides maximum branch capability in a microprogram control unit using the Am2909
- Advanced Low-Power Schottky processing
- 100% reliability assurance testing in compliance with MIL-STD-883

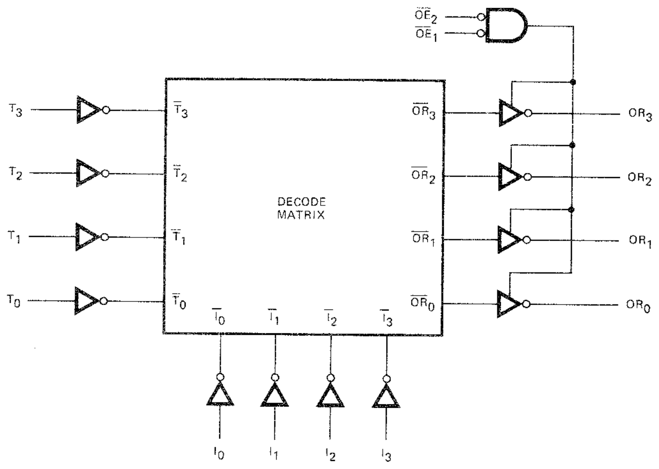
FUNCTIONAL DESCRIPTION

The Am29803 is a Low-Power Schottky processed device that provides 16-way branch control when used in conjunction with the Am2909 Microprogram Sequencer.

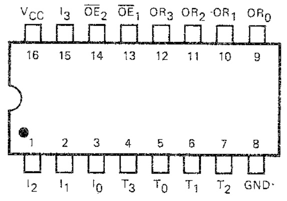
The device features 16 instructions that provide all combinations of simultaneous testing of four different inputs. The device has four outputs that are used to drive the four OR inputs of the Am2909 Microprogram Sequencer.

The "zero" instruction inhibits the testing of any of the four test (T) inputs. The remaining 15 instructions are used to test combinations of 1, 2, 3, or 4 of the T inputs simultaneously. If one T input is being tested, the Am29803 will select one of two possible addresses. If two T inputs are being tested, the device will select one of four possible addresses. If three T inputs are being tested, the device will select one of eight possible addresses. If all four T inputs are being tested, the device will select one of sixteen addresses as the field used to drive the OR inputs of the Am2909.

LOGIC DIAGRAM

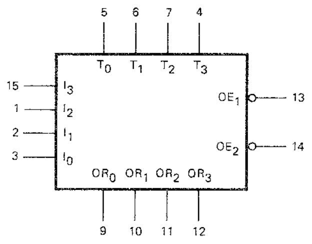


CONNECTION DIAGRAM
Top View



Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

ELECTRICAL CHARACTERISTICS

The Following Conditions Apply Unless Otherwise Specified:

COM'L	T_A = 0°C to +70°C	V_{CC} = 5.0V ±5%	MIN. = 4.75V	MAX. = 5.25V
MIL	T_A = −55°C to +125°C	V_{CC} = 5.0V ±10%	MIN. = 4.50V	MAX. = 5.50V

DC CHARACTERISTICS OVER OPERATING RANGE

Parameters	Description	Test Conditions (Note 1)		Min.	Typ. (Note 2)	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = MIN., I_{OH} = −2.0mA V_{IN} = V_{IH} or V_{IL}		2.4			Volts
V_{OL}	Output LOW Voltage	V_{CC} = MIN. V_{IN} = V_{IH} or V_{IL}	I_{OL} = 8.0mA			0.4	Volts
			I_{OL} = 16mA			0.45	
V_{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs		2.0			Volts
V_{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs	MIL			0.7	Volts
			COM'L			0.8	
V_I	Input Clamp Voltage	V_{CC} = MIN., I_{IN} = −18mA				−1.5	Volts
I_{IL}	Input LOW Current	V_{CC} = MAX., V_{IN} = 0.4V				−0.1	mA
I_{IH}	Input HIGH Current	V_{CC} = MAX., V_{IN} = 2.7V				10	μA
I_I	Input HIGH Current	V_{CC} = MAX., V_{IN} = 7.0V				1.0	mA
I_O	Off-State (High-Impedance) Output Current	V_{CC} = MAX.	V_O = 0.4V			−40	μA
			V_O = 2.4V			40	
I_{SC}	Output Short Circuit Current (Note 3)	V_{CC} = MAX.		−12		−85	mA
I_{CC}	Power Supply Current (Note 4)	V_{CC} = MAX.			55	80	mA

Notes: 1. For conditions shown as MIN. or MAX., use the appropriate value specified under Electrical Characteristics for the applicable device type.
2. Typical limits are at V_{CC} = 5.0V, 25° ambient and maximum loading.
3. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.
4. Inputs grounded; outputs open.

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	−65°C to +150°C
Temperature (Ambient) Under Bias	−55°C to +125°C
Supply Voltage to Ground Potential Continuous	−0.5V to +7.0V
DC Voltage Applied to Outputs for High Output State	−0.5V to +V_{CC} max.
DC Input Voltage	−0.5V to 5.5V
DC Output Current, Into Outputs	30mA
DC Input Current	−30mA to +5.0mA

SWITCHING CHARACTERISTICS

(T_A = +25°C, V_{CC} = 5.0V)

Parameters	Description	Test Conditions	Min.	Typ.	Max.	Units
t_{PLH} t_{PHL}	I_i to OR_i	C_L = 15pF R_L = 2.0kΩ		45	60	ns
t_{PLH} t_{PHL}	T_i to OR_i			45	60	ns
t_{ZH} t_{ZL}	$\overline{OE}_i$ to OR_i			25	30	ns
t_{HZ} t_{LZ}	$\overline{OE}_i$ to OR_i	C_L = 5.0pF R_L = 2.0kΩ		25	30	ns

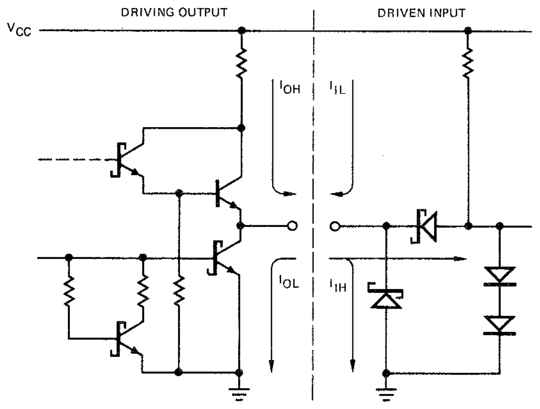
SWITCHING CHARACTERISTICS OVER OPERATING RANGE

			COM'L		MIL		
			T_A = 0°C to +70°C V_{CC} = 5.0V ±5%		T_A = −55°C to +125°C V_{CC} = 5.0V ±10%		
Parameters	Description	Test Conditions	Min.	Max.	Min.	Max.	Units
t_{PLH} t_{PHL}	I_i to OR_i	C_L = 15pF R_L = 2.0kΩ		70		80	ns
t_{PLH} t_{PHL}	T_i to OR_i			70		80	ns
t_{ZH} t_{ZL}	$\overline{OE}_i$ to OR_i			35		40	ns
t_{HZ} t_{LZ}	$\overline{OE}_i$ to OR_i			35		40	ns

DEFINITION OF FUNCTIONAL TERMS

I_0, I_1, I_2, I_3	The four instruction inputs to the device
T_0, T_1, T_2, T_3	The four test inputs for the device
OR_0, OR_1, OR_2, OR_3	The four outputs of the device that are connected to the four OR inputs of the Am2909
$\overline{OE}_1, \overline{OE}_2$	Output Enable. When either $\overline{OE}$ input is HIGH, the OR_i outputs are in the high impedance state. When both the $\overline{OE}_1$ and $\overline{OE}_2$ inputs are LOW, the OR outputs are enabled and the selected data will be present.

LOW-POWER SCHOTTKY INPUT/OUTPUT CURRENT INTERFACE CONDITIONS



Note: Actual current flow direction shown.

GUARANTEED LOADING RULES OVER OPERATING RANGE (In Unit Loads)

A Low-Power Schottky TTL Unit Load is defined as 20μA measured at 2.7V HIGH and −0.36mA measured at 0.4V LOW.

				Output LOW	
Pin No.'s	Input/Output	Input Load	Output HIGH	MIL	COM'L
1	I_2	0.5	–	–	–
2	I_1	0.5	–	–	–
3	I_0	0.5	–	–	–
4	T_3	0.5	–	–	–
5	T_0	0.5	–	–	–
6	T_1	0.5	–	–	–
7	T_2	0.5	–	–	–
8	GND	–	–	–	–
9	OR_0	–	100	44	44
10	OR_1	–	100	44	44
11	OR_2	–	100	44	44
12	OR_3	–	100	44	44
13	$\overline{OE}_1$	0.5	–	–	–
14	$\overline{OE}_2$	0.5	–	–	–
15	I_3	0.5	–	–	–
16	V_{CC}	–	–	–	–

FUNCTION TABLE

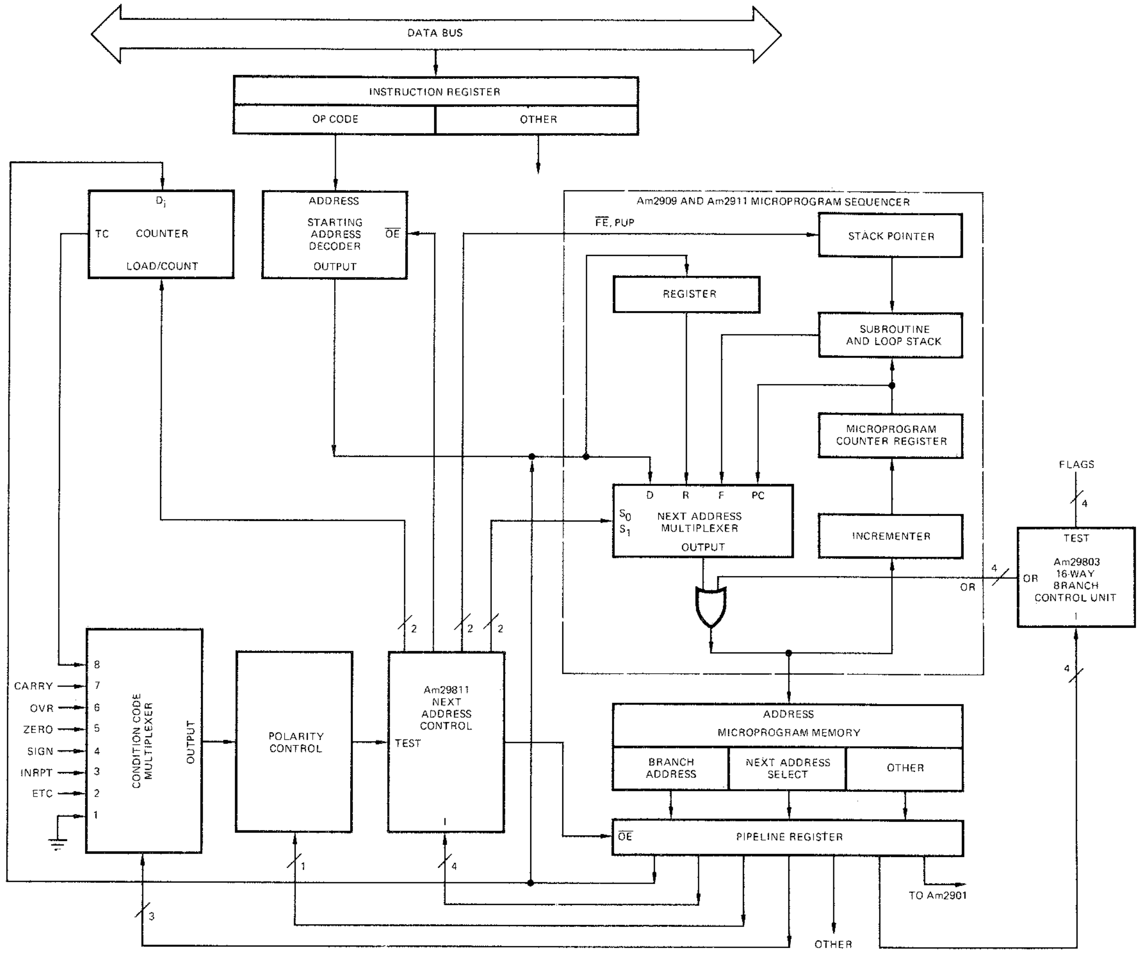
Function	I_3	I_2	I_1	I_0	T_3	T_2	T_1	T_0	OR_3	OR_2	OR_1	OR_0
No Test	L	L	L	L	X	X	X	X	L	L	L	L
Test T_0	L	L	L	H	X	X	X	L	L	L	L	L
					X	X	X	H	L	L	L	H
Test T_1	L	L	H	L	X	X	L	X	L	L	L	L
					X	X	H	X	L	L	L	H
Test T_0 & T_1	L	L	H	H	X	X	L	L	L	L	L	L
					X	X	L	H	L	L	L	H
					X	X	H	L	L	L	H	L
					X	X	H	H	L	L	H	H
Test T_2	L	H	L	L	X	L	X	X	L	L	L	L
					X	H	X	X	L	L	L	H
Test T_0 & T_2	L	H	L	H	X	L	X	L	L	L	L	L
					X	L	X	H	L	L	L	H
					X	H	X	L	L	L	H	L
					X	H	X	H	L	L	H	H
Test T_1 & T_2	L	H	H	L	X	L	L	X	L	L	L	L
					X	L	H	X	L	L	L	H
					X	H	L	X	L	L	H	L
					X	H	H	X	L	L	H	H
Test T_0, T_1 & T_2	L	H	H	H	X	L	L	L	L	L	L	L
					X	L	L	H	L	L	L	H
					X	L	H	L	L	L	H	L
					X	L	H	H	L	L	H	H
					X	H	L	L	L	H	L	L
					X	H	L	H	L	H	L	H
					X	H	H	L	L	H	H	L
					X	H	H	H	L	H	H	H
Test T_3	H	L	L	L	L	X	X	X	L	L	L	L
					H	X	X	X	L	L	L	H
Test T_0 & T_3	H	L	L	H	L	X	X	L	L	L	L	L
					L	X	X	H	L	L	L	H
					H	X	X	L	L	L	H	L
					H	X	X	H	L	L	H	H
Test T_1 & T_3	H	L	H	L	L	X	L	X	L	L	L	L
					L	X	H	X	L	L	L	H
					H	X	L	X	L	L	H	L
					H	X	H	X	L	L	H	H
Test T_0, T_1 & T_3	H	L	H	H	L	X	L	L	L	L	L	L
					L	X	L	H	L	L	L	H
					L	X	H	L	L	L	H	L
					L	X	H	H	L	L	H	H
					H	X	L	L	L	H	L	L
					H	X	L	H	L	H	L	H
					H	X	H	L	L	H	H	L
					H	X	H	H	L	H	H	H
Test T_2 & T_3	H	H	L	L	L	L	X	X	L	L	L	L
					L	H	X	X	L	L	L	H
					H	L	X	X	L	L	H	L
					H	H	X	X	L	L	H	H
Test T_0, T_2 & T_3	H	H	L	H	L	L	X	L	L	L	L	L
					L	L	X	H	L	L	L	H
					L	H	X	L	L	L	H	L
					L	H	X	H	L	L	H	H
					H	L	X	L	L	H	L	L
					H	L	X	H	L	H	L	H
					H	H	X	L	L	H	H	L
					H	H	X	H	L	H	H	H
Test T_1, T_2 & T_3	H	H	H	L	L	L	L	X	L	L	L	L
					L	L	H	X	L	L	L	H
					L	H	L	X	L	L	H	L
					L	H	H	X	L	L	H	H
					H	L	L	X	L	H	L	L
					H	L	H	X	L	H	L	H
					H	H	L	X	L	H	H	L
					H	H	H	X	L	H	H	H
Test T_0, T_1, T_2 & T_3	H	H	H	H	L	L	L	L	L	L	L	L
					L	L	L	H	L	L	L	H
					L	L	H	L	L	L	H	L
					L	L	H	H	L	L	H	H
					L	H	L	L	L	H	L	L
					L	H	L	H	L	H	L	H
					L	H	H	L	L	H	H	L
					L	H	H	H	L	H	H	H
					H	L	L	L	H	L	L	L
					H	L	L	H	H	L	L	H
					H	L	H	L	H	L	H	L
					H	L	H	H	H	L	H	H
					H	H	L	L	H	H	L	L
					H	H	L	H	H	H	L	H
					H	H	H	L	H	H	H	L
					H	H	H	H	H	H	H	H

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

ORDERING INFORMATION

Package Type	Temperature Range	Order Number
Molded DIP	0°C to +70°C	AM29803PC
Hermetic DIP	0°C to +70°C	AM29803DC
Dice	0°C to +70°C	AM29803XC
Hermetic DIP	–55°C to +125°C	AM29803DM
Hermetic Flat Pak	–55°C to +125°C	AM29803FM
Dice	–55°C to +125°C	AM29803XM

APPLICATION



A typical computer control unit using the Am2909, Am2911, Am29803 and Am29811. Note that the least significant microprogram sequencer is an Am2909 and the more significant sequencers are Am2911's.

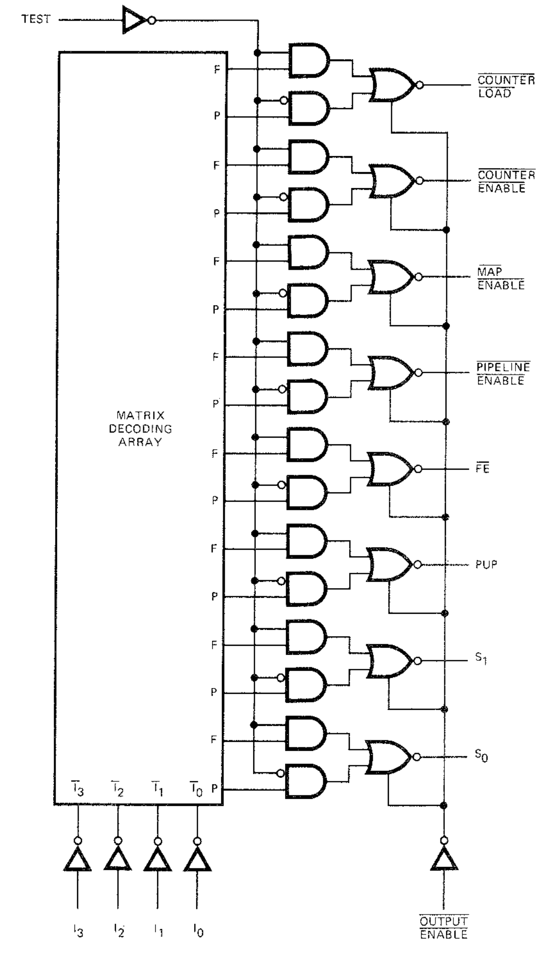
Am29811

Next Address Control Unit

DISTINCTIVE CHARACTERISTICS

- Next address control unit for the Am2911 Microprogram Sequencer
- 16 next address instructions
- Test input for conditional instructions
- Separate outputs to control the Am2911, an independent event counter, and a mapping PROM/branch address interface
- Advanced Low-Power Schottky technology
- 100% reliability assurance testing in compliance with MIL-STD-883

LOGIC DIAGRAM



TEST
MATRIX DECODING ARRAY
F
P
$\overline{I_3}$ $\overline{I_2}$ $\overline{I_1}$ $\overline{I_0}$ P
$\overline{\text{COUNTER LOAD}}$
$\overline{\text{COUNTER ENABLE}}$
$\overline{\text{MAP ENABLE}}$
$\overline{\text{PIPELINE ENABLE}}$
$\overline{\text{FE}}$
PUP
S_1
S_0
I_3 I_2 I_1 I_0
$\overline{\text{OUTPUT ENABLE}}$

P = Pass
F = Fail

FUNCTIONAL CHARACTERISTICS

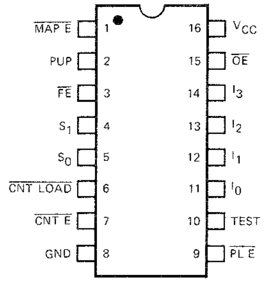
The Am29811 is a Low-Power Schottky device designed specifically for next address control of the Am2911 Microprogram Sequencer. The device contains all outputs required to control a high-performance computer control unit or a structured state machine design using microprogramming techniques.

Sixteen instructions are available by using a four-bit instruction field I_{0-3}. In addition, a test input is available such that conditional instructions can be performed based on a condition code test input.

The full instruction set consists of such functions as conditional jumps, conditional jump-to-subroutine, conditional return-from-subroutine, conditional repeat loops, conditional branch to starting address, and so forth.

One Am29811 can be used to control any number of Am2911 Microprogram Sequencers. The Am2911 Sequencer is a four-bit slice itself. Thus, one Am29811 Next Address Control Unit and three Am2911 Microprogram Sequencers can be used to build the most powerful, state-of-the-art, microprogram sequencer capable of controlling 4k words of microprogram memory.

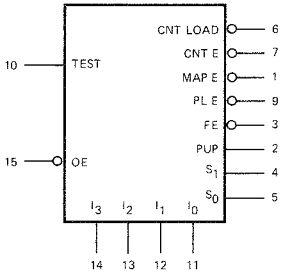
CONNECTION DIAGRAM
Top View



$\overline{\text{MAP E}}$ 1
PUP 2
$\overline{\text{FE}}$ 3
S_1 4
S_0 5
$\overline{\text{CNT LOAD}}$ 6
$\overline{\text{CNT E}}$ 7
GND 8
9 $\overline{\text{PL E}}$
10 TEST
11 I_0
12 I_1
13 I_2
14 I_3
15 $\overline{\text{OE}}$
16 V_{CC}

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

ELECTRICAL CHARACTERISTICS

The Following Conditions Apply Unless Otherwise Specified:

COM'L	T_A = 0°C to +70°C	V_{CC} = 5.0 V ±5%	MIN. = 4.75 V	MAX. = 5.25 V
MIL	T_A = –55°C to +125°C	V_{CC} = 5.0 V ±10%	MIN. = 4.50 V	MAX. = 5.50 V

DC CHARACTERISTICS OVER OPERATING RANGE

Parameters	Description	Test Conditions (Note 1)		Min.	Typ. (Note 2)	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = MIN., I_{OH} = –2.0 mA V_{IN} = V_{IH} or V_{IL}.		2.4			Volts
V_{OL}	Output LOW Voltage	V_{CC} = MIN. V_{IN} = V_{IH} or V_{IL}	I_{OL} = 8.0 mA			0.4	Volts
			I_{OL} = 16 mA			0.45	
V_{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs		2.0			Volts
V_{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs	MIL			0.7	Volts
			COM'L			0.8	
V_I	Input Clamp Voltage	V_{CC} = MIN., I_{IN} = –18 mA				–1.5	Volts
I_{IL}	Input LOW Current	V_{CC} = MAX., V_{IN} = 0.4 V				–0.1	mA
I_{IH}	Input HIGH Current	V_{CC} = MAX., V_{IN} = 2.7 V				10	μA
I_I	Input HIGH Current	V_{CC} = MAX., V_{IN} = 7.0 V				1.0	mA
I_O	Off-State (High-Impedance) Output Current	V_{CC} = MAX.	V_O = 0.4 V			–40	μA
			V_O = 2.4 V			40	
I_{SC}	Output Short Circuit Current (Note 3)	V_{CC} = MAX.		–12		–85	mA
I_{CC}	Power Supply Current (Note 4)	V_{CC} = MAX.			55	80	mA

Notes: 1. For conditions shown as MIN. or MAX., use the appropriate value specified under Electrical Characteristics for the applicable device type.

2. Typical limits are at V_{CC} = 5.0 V, 25°C ambient and maximum loading.

3. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.

4. Inputs grounded; outputs open.

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	–65°C to +150°C
Temperature (Ambient) Under Bias	–55°C to +125°C
Supply Voltage to Ground Potential Continuous	–0.5 V to +7.0 V
DC Voltage Applied to Outputs for High Output State	–0.5 V to +V_{CC} max.
DC Input Voltage	–0.5 V to +7.0 V
DC Output Current, Into Outputs	30 mA
DC Input Current	–30 mA to +5.0 mA

SWITCHING CHARACTERISTICS

(T_A = +25°C, V_{CC} = 5.0 V)

Parameters	Description	Test Conditions	Min.	Typ.	Max.	Units
t_{PLH} t_{PHL}	I_i to Any Output	C_L = 15 pF R_L = 2.0 kΩ		30	50	ns
t_{PLH} t_{PHL}	Test to Any Output			30	50	ns
t_{ZH} t_{ZL}	$\overline{\text{OE}}$ to Any Output			30	40	ns
t_{HZ} t_{LZ}	$\overline{\text{OE}}$ to Any Output	C_L = 5.0 pF R_L = 2.0 kΩ		18	35	ns

SWITCHING CHARACTERISTICS OVER OPERATING RANGE

			COM'L		MIL		
			T_A = 0°C to +70°C V_{CC} = 5.0 V ±5%		T_A = –55°C to +125°C V_{CC} = 5.0 V ±10%		
Parameters	Description	Test Conditions	Min.	Max.	Min.	Max.	Units
t_{PLH} t_{PHL}	I_i to Any Output	C_L = 15 pF R_L = 2.0 kΩ		55		75	ns
t_{PLH} t_{PHL}	Test to Any Output			55		75	ns
t_{ZH} t_{ZL}	$\overline{\text{OE}}$ to Any Output			40		50	ns
t_{HZ} t_{LZ}	$\overline{\text{OE}}$ to Any Output			35		40	ns

DEFINITION OF FUNCTIONAL TERMS

I_0, I_1, I_2, I_3 The four instruction inputs to the Am29811.

TEST The condition code input to the device. When the test input is LOW, the device assumes the test has failed. When the test input is HIGH, the device assumes the condition code required has been met; the test has passed.

$\overline{\text{Counter Load}}$ This output is used to drive the parallel load input of an Am25LS169 up/down counter.

$\overline{\text{Counter Enable}}$ This output is used to drive the counter enable input of an Am25LS169 up/down counter.

$\overline{\text{Map Enable}}$ This output is used to control the three-state outputs of the mapping PROM or PLA used to provide the initial starting address for each machine instruction.

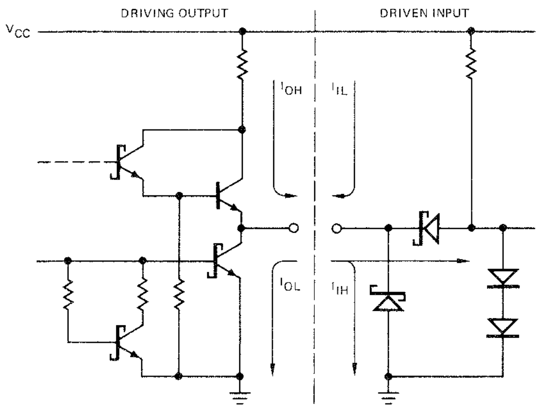
$\overline{\text{Pipeline Enable}}$ This output is used to control the three-state output of the pipeline register (Am2918) containing the branch address for the computer control unit.

$\overline{\text{FE}}$ File Enable This output is used to drive the file enable input of the Am2911. When the file enable output is LOW, a stack operation will take place.

PUP Push/Pop. The PUP output is used to drive the push/pop input of the Am2911 Microprogram Sequencer. When the PUP output is HIGH, a push will take place when the file is enabled. When the PUP output is LOW, a pop will take place when the file is enabled.

S_0, S_1 These two outputs are used to drive the S_0 and S_1 inputs to the Am2911 Microprogram Sequencer. These outputs control whether the direct input, the register, the microprogram counter, or the stack is selected as the source of the next address for the microprogram memory.

LOW-POWER SCHOTTKY INPUT/OUTPUT CURRENT INTERFACE CONDITIONS



Note: Actual current flow direction shown.

GUARANTEED LOADING RULES OVER OPERATING RANGE (In Unit Loads)

A Low-Power Schottky TTL Unit Load is defined as 20µA measured at 2.7V HIGH and −0.36mA measured at 0.4V LOW.

Pin No.'s	Input/Output	Input Load	Output HIGH	Output LOW MIL	Output LOW COM'L
1	$\overline{\text{MAP E}}$	–	100	44	44
2	PUP	–	100	44	44
3	$\overline{\text{FE}}$	–	100	44	44
4	S_1	–	100	44	44
5	S_0	–	100	44	44
6	$\overline{\text{CNT LOAD}}$	–	100	44	44
7	$\overline{\text{CNT E}}$	–	100	44	44
8	GND	–	–	–	–
9	$\overline{\text{PL E}}$	–	100	44	44
10	TEST	0.5	–	–	–
11	I_0	0.5	–	–	–
12	I_1	0.5	–	–	–
13	I_2	0.5	–	–	–
14	I_3	0.5	–	–	–
15	$\overline{\text{OE}}$	–	100	44	44
16	V_{CC}	–	–	–	–

INSTRUCTION TABLE

MNEMONIC	I_3 I_2 I_1 I_0	INSTRUCTION
JZ	L L L L	Jump to Address Zero
CJS	L L L H	Conditional Jump-to-Subroutine with Jump Address in Pipeline Register.
JMAP	L L H L	Jump to Address at Mapping PROM Output.
CJP	L L H H	Conditional Jump to Address in Pipeline Register
PUSH	L H L L	Push Stack and Conditionally Load Counter
JSRP	L H L H	Jump-to-Subroutine with Starting Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
CJV	L H H L	Conditional Jump to Vector Address.
JRP	L H H H	Jump to Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
RFCT	H L L L	Repeat Loop if Counter is not Equal to Zero.
RPCT	H L L H	Repeat Pipeline Address if Counter is not Equal to Zero.
CRTN	H L H L	Conditional Return-from-Subroutine.
CJPP	H L H H	Conditional Jump to Pipeline Address and Pop Stack.
LDCT	H H L L	Load Counter and Continue.
LOOP	H H L H	Test End of Loop.
CONT	H H H L	Continue to Next Address.
JP	H H H H	Jump to Pipeline Register Address.

FUNCTION TABLE

MNEMONIC	INPUTS			OUTPUTS				
	INSTRUCTION I_3 I_2 I_1 I_0	FUNCTION	TEST INPUT	NEXT ADDR SOURCE	FILE	COUNTER	MAP-E	PL-E
JZ	L L L L	JUMP ZERO	X	D	HOLD	L L*	H	L
CJS	L L L H	COND JSB PL	L	PC	HOLD	HOLD	H	L
			H	D	PUSH	HOLD	H	L
JMAP	L L H L	JUMP MAP	X	D	HOLD	HOLD	L	H
CJP	L L H H	COND JUMP PL	L	PC	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
PUSH	L H L L	PUSH/COND LD CNTR	L	PC	PUSH	HOLD	H	L
			H	PC	PUSH	LOAD	H	L
JSRP	L H L H	COND JSB R/PL	L	R	PUSH	HOLD	H	L
			H	D	PUSH	HOLD	H	L
CJV	L H H L	COND JUMP VECTOR	L	PC	HOLD	HOLD	H	H
			H	D	HOLD	HOLD	H	H
JRP	L H H H	COND JUMP R/PL	L	R	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
RFCT	H L L L	REPEAT LOOP, CNTR ≠ 0	L	F	HOLD	DEC	H	L
			H	PC	POP	HOLD	H	L
RPCT	H L L H	REPEAT PL, CNTR ≠ 0	L	D	HOLD	DEC	H	L
			H	PC	HOLD	HOLD	H	L
CRTN	H L H L	COND RTN	L	PC	HOLD	HOLD	H	L
			H	F	POP	HOLD	H	L
CJPP	H L H H	COND JUMP PL & POP	L	PC	HOLD	HOLD	H	L
			H	D	POP	HOLD	H	L
LDCT	H H L L	LOAD CNTR & CONTINUE	X	PC	HOLD	LOAD	H	L
LOOP	H H L H	TEST END LOOP	L	F	HOLD	HOLD	H	L
			H	PC	POP	HOLD	H	L
CONT	H H H L	CONTINUE	X	PC	HOLD	HOLD	H	L
JP	H H H H	JUMP PL	X	D	HOLD	HOLD	H	L

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

DEC = Decrement
*LL = Special Case

TRUTH TABLE

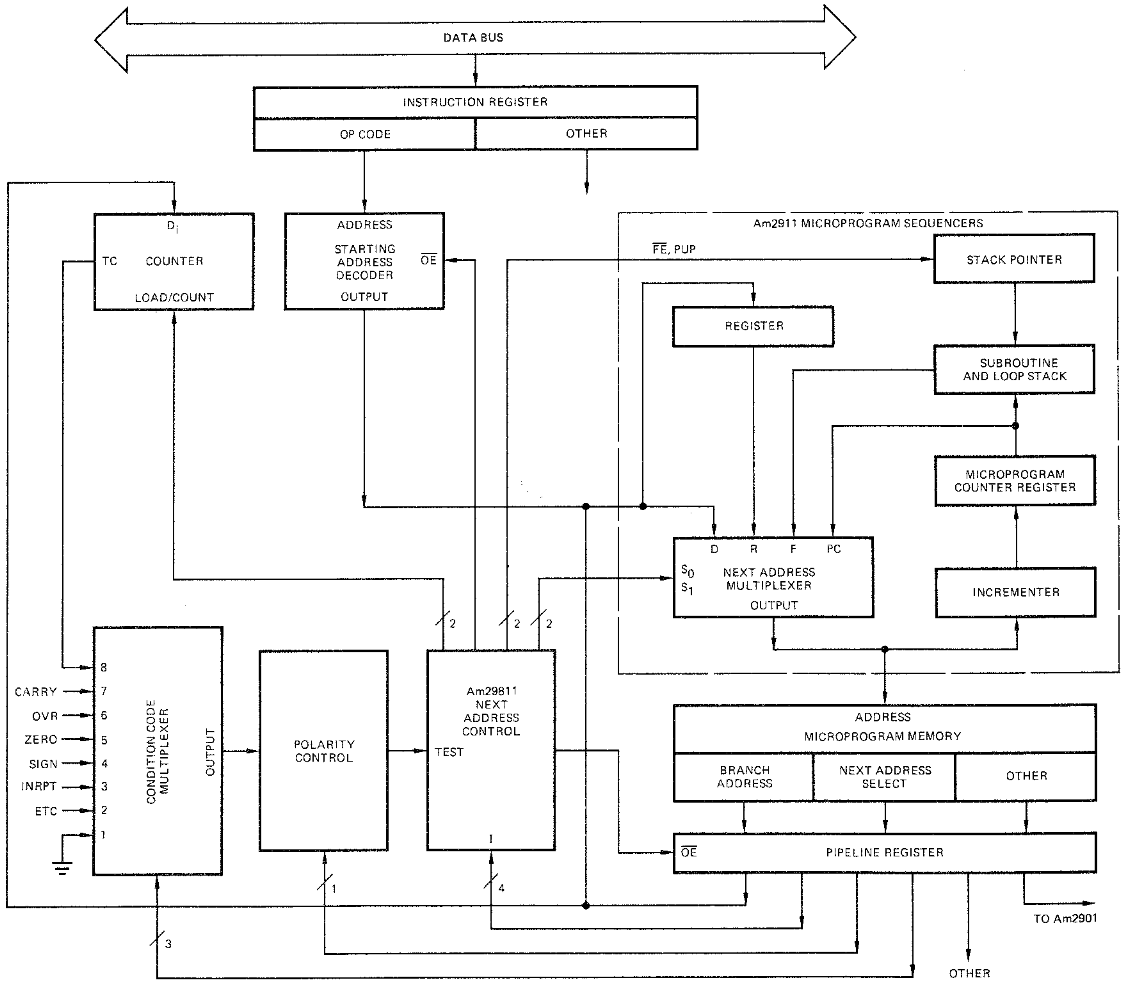
MNEMONIC	FUNCTION	INPUTS					OUTPUTS							
							NEXT ADDR SOURCE		FILE		COUNTER			
		I_3	I_2	I_1	I_0	TEST	S_1	S_0	$\overline{\text{FE}}$	PUP	$\overline{\text{LOAD}}$	$\overline{\text{EN}}$	$\overline{\text{MAP E}}$	$\overline{\text{PL E}}$
	PIN NO.	14	13	12	11	10	4	5	3	2	6	7	1	9
JZ	JUMP ZERO	L	L	L	L	L	H	H	H	H	L	L	H	L
		L	L	L	L	H	H	H	H	H	L	L	H	L
CJS	COND JSB PL	L	L	L	H	L	L	L	H	H	H	H	H	L
		L	L	L	H	H	H	H	L	H	H	H	H	L
JMAP	JUMP MAP	L	L	H	L	L	H	H	H	H	H	H	L	H
		L	L	H	L	H	H	H	H	H	H	H	L	H
CJP	COND JUMP PL	L	L	H	H	L	L	L	H	H	H	H	H	L
		L	L	H	H	H	H	H	H	H	H	H	H	L
PUSH	PUSH/COND LD CNTR	L	H	L	L	L	L	L	L	H	H	H	H	L
		L	H	L	L	H	L	L	L	H	L	H	H	L
JSRP	COND JSB R/PL	L	H	L	H	L	L	H	L	H	H	H	H	L
		L	H	L	H	H	H	H	L	H	H	H	H	L
CJV	COND JUMP VECTOR	L	H	H	L	L	L	L	H	H	H	H	H	H
		L	H	H	L	H	H	H	H	H	H	H	H	H
JRP	COND JUMP R/PL	L	H	H	H	L	L	H	H	H	H	H	H	L
		L	H	H	H	H	H	H	H	H	H	H	H	L
RFCT	REPEAT LOOP, CTR ≠ 0	H	L	L	L	L	H	L	H	L	H	L	H	L
		H	L	L	L	H	L	L	L	L	H	H	H	L
RPCT	REPEAT PL, CTR ≠ 0	H	L	L	H	L	H	H	H	H	H	L	H	L
		H	L	L	H	H	L	L	H	H	H	H	H	L
CRTN	COND RTN	H	L	H	L	L	L	L	H	L	H	H	H	L
		H	L	H	L	H	H	L	L	L	H	H	H	L
CJPP	COND JUMP PL & POP	H	L	H	H	L	L	L	H	L	H	H	H	L
		H	L	H	H	H	H	H	L	L	H	H	H	L
LDCT	LD CNTR & CONTINUE	H	H	L	L	L	L	L	H	H	L	H	H	L
		H	H	L	L	H	L	L	H	H	L	H	H	L
LOOP	TEST END LOOP	H	H	L	H	L	H	L	H	L	H	H	H	L
		H	H	L	H	H	L	L	L	L	H	H	H	L
CONT	CONTINUE	H	H	H	L	L	L	L	H	H	H	H	H	L
		H	H	H	L	H	L	L	H	H	H	H	H	L
JP	JUMP PL	H	H	H	H	L	H	H	H	H	H	H	H	L
		H	H	H	H	H	H	H	H	H	H	H	H	L

L = LOW
H = HIGH

ORDERING INFORMATION

Package Type	Temperature Range	Order Number
Molded DIP	0°C to +70°C	AM29811PC
Hermetic DIP	0°C to +70°C	AM29811DC
Dice	0°C to +70°C	AM29811XC
Hermetic DIP	-55°C to +125°C	AM29811DM
Hermetic Flat Pak	-55°C to +125°C	AM29811FM
Dice	-55°C to +125°C	AM29811XM

APPLICATION



A Typical Computer Control Unit Using the Am2911 and Am29811

Am2922

Eight Input Multiplexer With Control Register

DISTINCTIVE CHARACTERISTICS

- High-speed eight-input multiplexer
- On-chip select field and polarity control register
- Buffered common register enable
- Buffered common asynchronous clear input
- Polarity control to select multiplexer input data as inverting or non-inverting
- Three-state outputs for expansion
- Features improved noise margin, higher drive, and faster operation
- 100% reliability assurance testing in compliance with MIL-STD-883

FUNCTIONAL DESCRIPTION

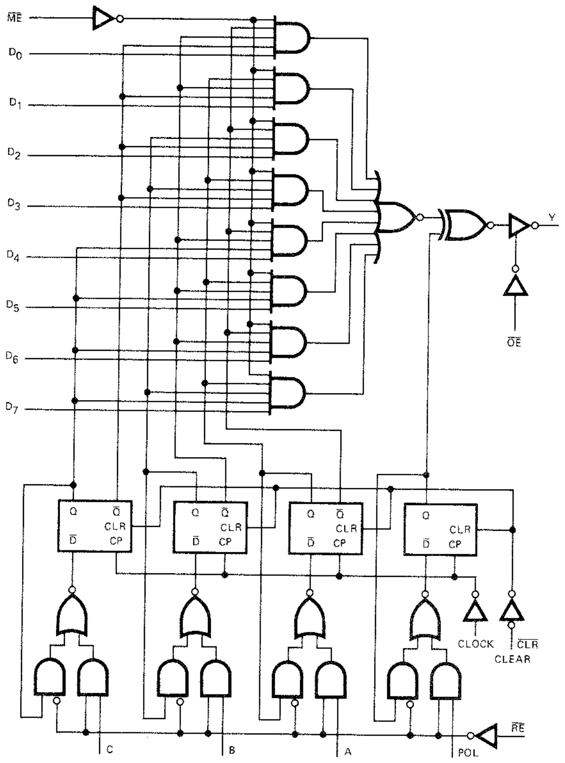
The Am2922 multiplexer with control register is an advanced Low-Power Schottky circuit. The device features extremely high speed from clock to output and is intended for use in high-speed computer control units or structured state machine designs.

The device contains an internal register to hold the A, B, and C multiplexer select field as well as the POL (polarity) control bit. These input bits (A, B, C, POL) are stored in the register, when the register is enabled. The register enable input ($\overline{RE}$) is used to selectively load data into the register. When $\overline{RE}$ is LOW, new data is entered into the register on the LOW-to-HIGH transition of the clock. When $\overline{RE}$ is HIGH, the register will retain its current data. An asynchronous master clear ($\overline{CLR}$) input is used to reset the internal Q output of the register to a logic LOW level.

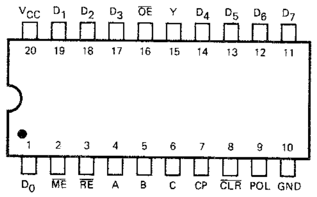
The A, B, and C outputs of the register control the eight-input multiplexer data select. These control signals switch one of eight data inputs into the exclusive NOR gate used to control output polarity. The logic signal contained in the polarity control flip-flop is used to control the polarity at the output. A HIGH on the polarity control flip-flop output represents true (non-inverting) output data and a LOW at the polarity control flip-flop output causes the input data to be inverted at the output. In a computer control unit, this allows testing of either true or complemented flag data at the microprogram sequencer test input.

An active LOW multiplexer enable input ($\overline{ME}$) allows the selected data to be passed through the multiplexer to the output buffer. When the $\overline{ME}$ is LOW, the incoming data may be passed to the output. The device also features a three-state output enable control input ($\overline{OE}$) to control the three-state output for expansion. When $\overline{OE}$ is LOW, the output is enabled. When $\overline{OE}$ is HIGH, the output is in the high impedance state.

LOGIC DIAGRAM

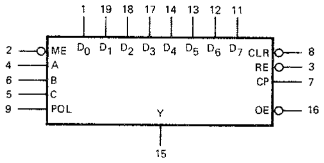


CONNECTION DIAGRAM
Top View



Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



V_{CC} = Pin 20
GND = Pin 10

EMULATING THE Am2910 WITH THE Am2911/Am29811

It is quite straightforward to emulate almost all of the functions of the Am2910 by using three Am2911's and one Am29811 by combining the features of Figure 8, Figure 10 and Figure 11 from the Advanced Micro Devices Microprogramming Handbook. In addition, the first 15 instructions of the Am29811 match directly with the instructions of the Am2910.

The Am2910 is most directly represented by the architecture shown in Figure 11 with two major exceptions. The first of these involves the least significant sequencer slice U8. If the Am2909 (U8), as shown in Figure 11, is replaced with the Am2911 (U8), as shown in Figure 8, the first major change has been completed. Needless to say, the Am29803 (U23) and Am2918 (U22) as shown in Figure 11 can no longer be used. The second change required in Figure 11 involves the three Am25LS163 counters (U9, U10, and U11). This counter section should be replaced with the down counter configuration as shown in Figure 10 utilizing the three Am25LS169 counters (U9, U10, and U11). Finally, the $\overline{\text{RE}}$ inputs on the Am2911's should be connected to the counter load line instead of ground. If these three changes are implemented, the logic diagram of Figure 11 will very nearly represent the Am2910.

This connection will now come very close to emulating the Am2910 in the highest speed possible configuration. The only differences between this configuration and the Am2910 are as follows. First, the Am2910 three-way branch instruction cannot be executed. Second, the actual value loaded in the counter (U9, U10, and U11) will be executed N+2 times where N is the value loaded in the counter. In the Am2910, the microinstruction will be executed N+1 times, based on the value loaded on the counter. Incidentally, if zero is loaded into the counter in the Am2910, the microinstruction will be executed one time when in the pipeline register. If speed is not a consideration, the four-input multiplexer (U26) and the D flip-flop (U27) can be removed and the carry output from the counter (U9) can be connected directly to the test 15 input of the eight-input multiplexer (U12). This will result in an exact emulation of the counter on the Am2910 chip. However, the total ripple time of the three-stage counter now must be added to the propagation time from the test input to the pipeline register in order to compute the minimum microcycle time. (This has been optimized differently on the Am2910.)

Thus, based on the configuration shown in Figure 11, the Am2910 replaces eight integrated circuits. These include U6, U7, U8, U9, U10, U11, U15 and U16. It also eliminates the ability to use U22 and U23 in the N-way branch mode. The counter test connection formed via U26, U27, and U12 is performed internally on the Am2910 and no condition code multiplexer (U12) connection is required. We might also mention that the function performed by U13, U14, and U17 have been combined into a single high-speed device known as the Am2922 Condition Code Multiplexer.

At the current time, it is anticipated that by utilizing the Am2910, Am2922, high-speed PROM, and a high-speed pipeline register, it will be possible to run 100ns microcycles based on worst-case propagation delays. Please refer to the Am2910 data sheet for further details on the operation of the Am2910.

MULTIPLEXER SELECT

R_{20}	R_{19}	R_{18}	R_{17}	SELECT
0	0	0	0	TEST 0
0	0	0	1	TEST 1
0	0	1	0	TEST 2
		•		•
		•		•
		•		•
1	1	1	1	TEST 15

POLARITY CONTROL

R_{16}	OUTPUT
0	COMPLEMENT OF TEST
1	TRUE TEST

NEXT ADDRESS CONTROL

R_{15}	R_{14}	R_{13}	R_{12}	FUNCTION
X	X	X	X	NEXT INSTRUCTION

MACHINE INSTRUCTION REGISTER

R_{21}	FUNCTION
0	LOAD
1	HOLD

CONTROL VALUE

R_{11}-R_0	FUNCTION
XXX---XXX	VALUE

JUMP ADDRESS

BR_{11}-BR_0	FUNCTION
XXX---XXX	JUMP ADDRESS

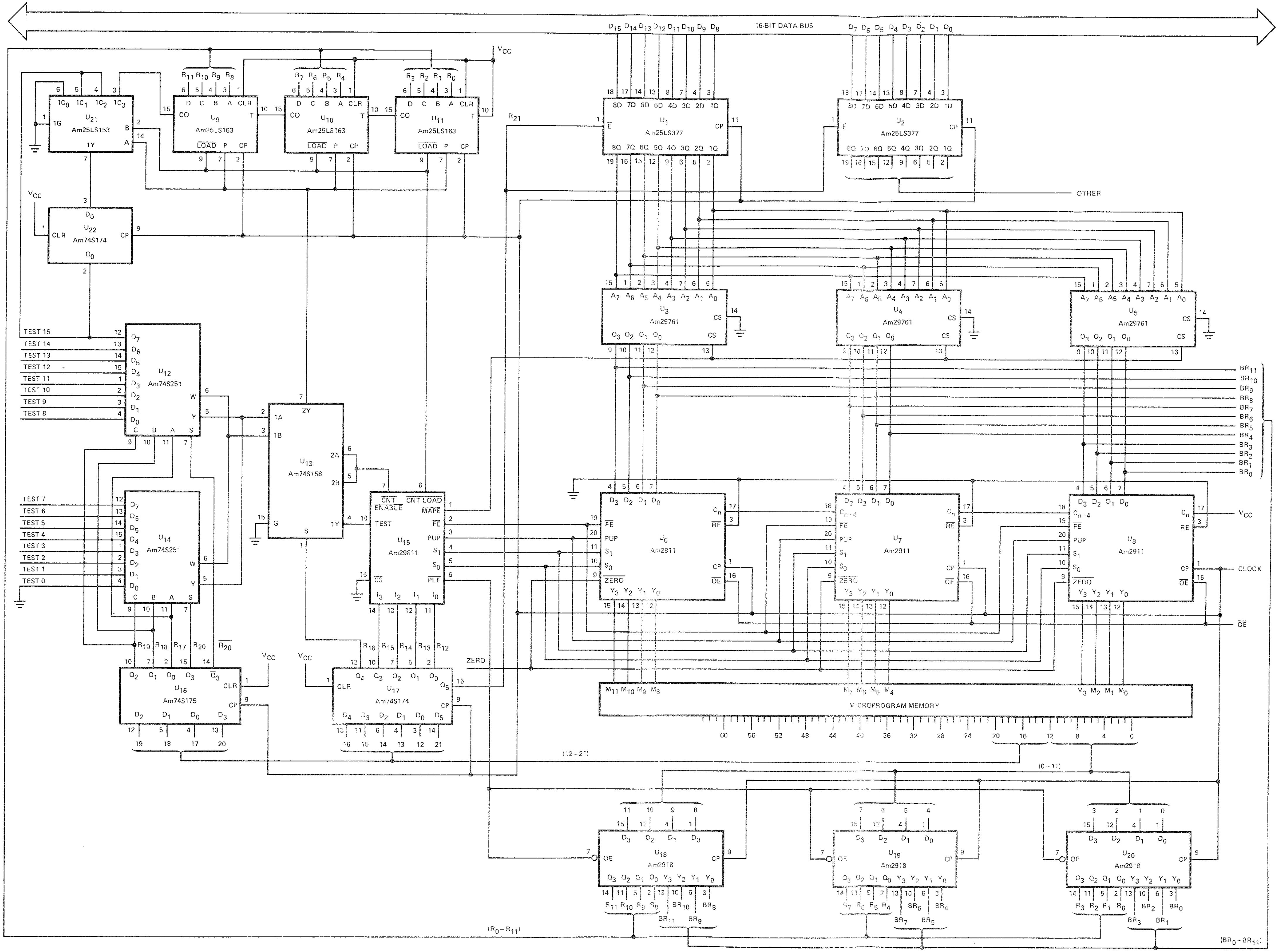


Figure 8. Computer Control Unit with Am2911.

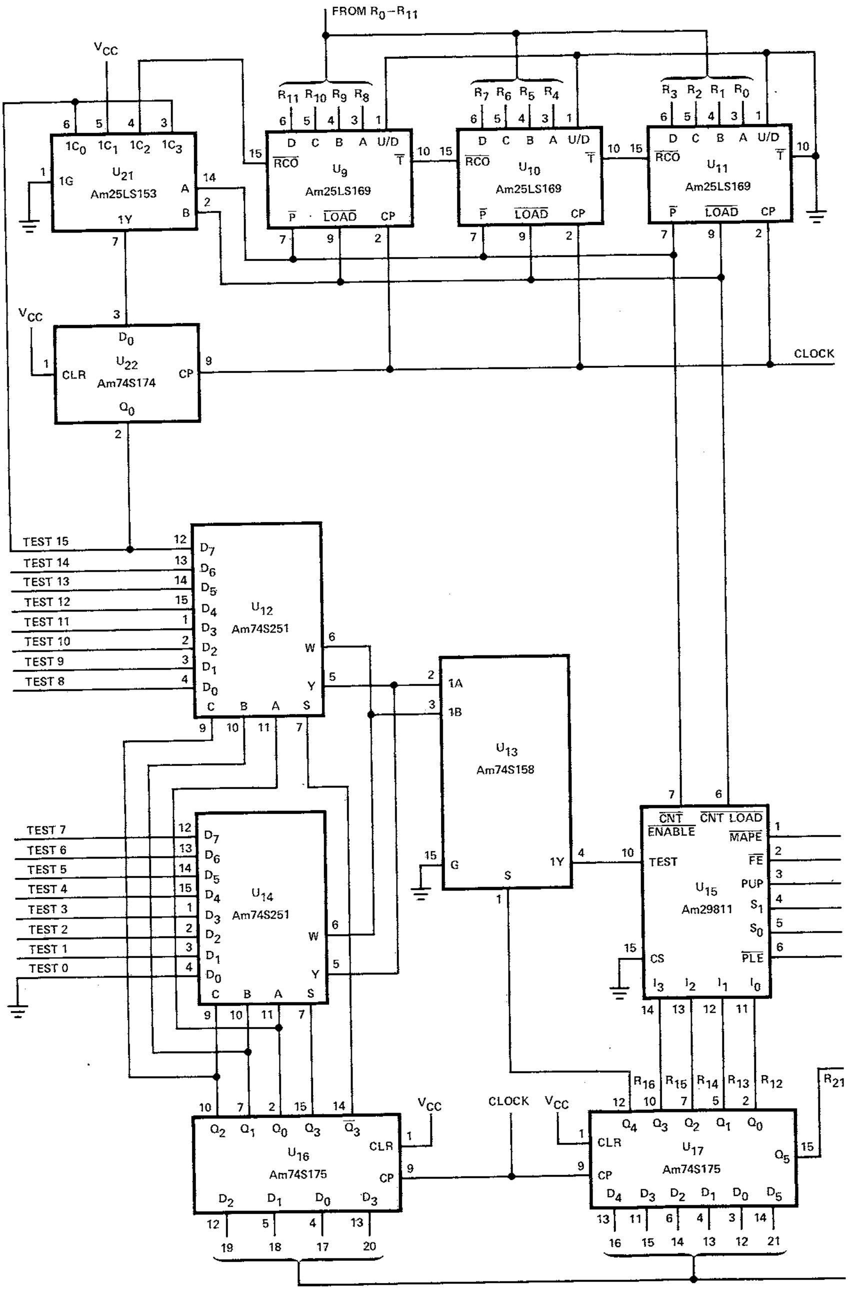


Figure 10. Am25LS169 Counter Connection.

MULTIPLEXER SELECT

R_{20}	R_{19}	R_{18}	R_{17}	SELECT
0	0	0	0	TEST 0
0	0	0	1	TEST 1
0	0	1	0	TEST 2
	⋮			⋮
1	1	1	1	TEST 15

POLARITY CONTROL

R_{16}	OUTPUT
0	COMPLEMENT OF TEST
1	TRUE TEST

NEXT ADDRESS CONTROL

R_{15}	R_{14}	R_{13}	R_{12}	FUNCTION
X	X	X	X	NEXT INSTRUCTION

MACHINE INSTRUCTION REGISTER

R_{21}	FUNCTION
0	LOAD
1	HOLD

COUNTER VALUE

R_{11}–R_0	FUNCTION
XXX---XXX	VALUE

JUMP ADDRESS

BR_{11}–BR_0	FUNCTION
XXX---XXX	JUMP ADDRESS

OR BRANCH CONTROL

R_{25}	R_{24}	R_{23}	R_{22}	FUNCTION
X	X	X	X	TEST INSTRUCTION

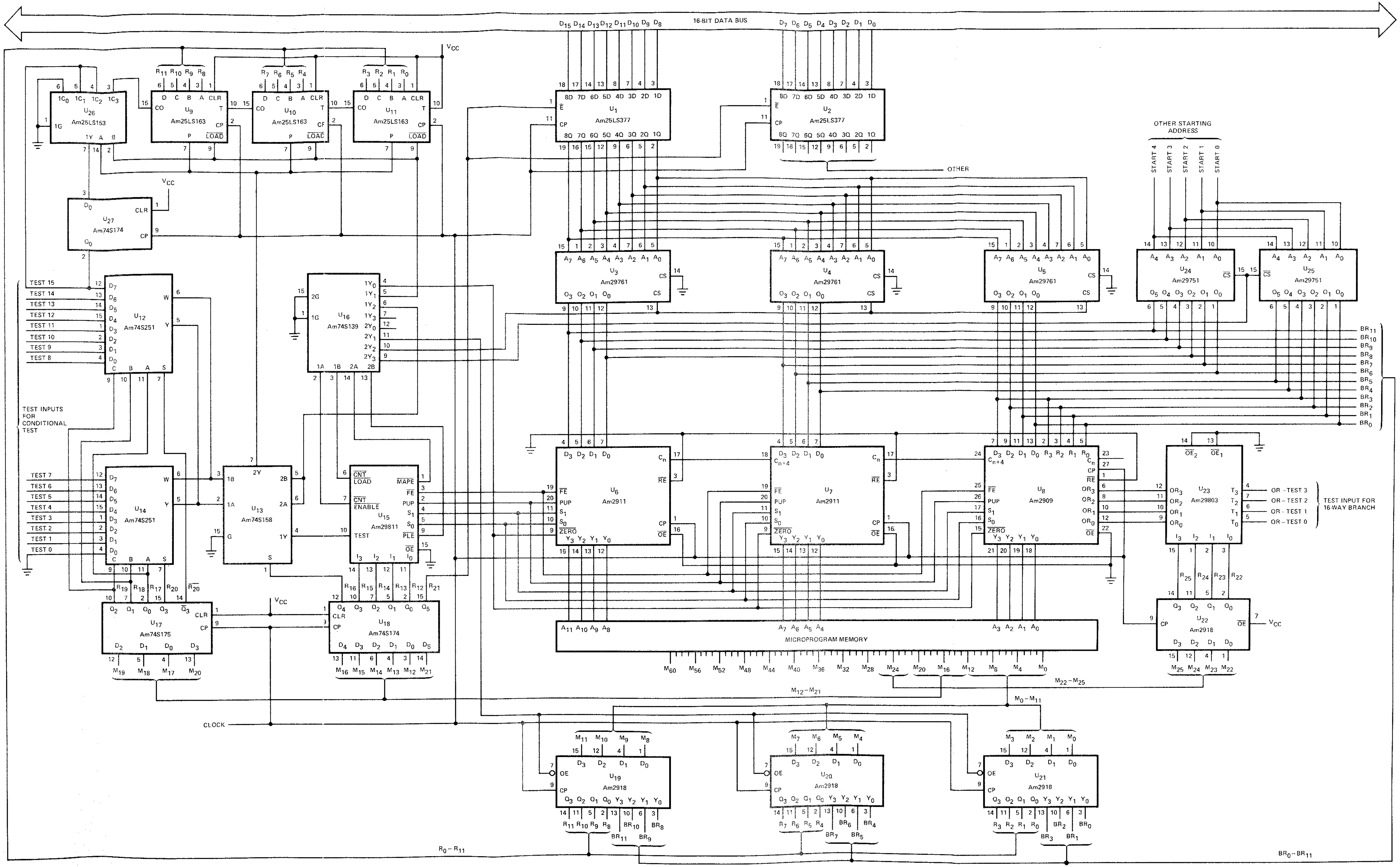


Figure 11. High Performance Computer Control Unit with Am2909/2911.

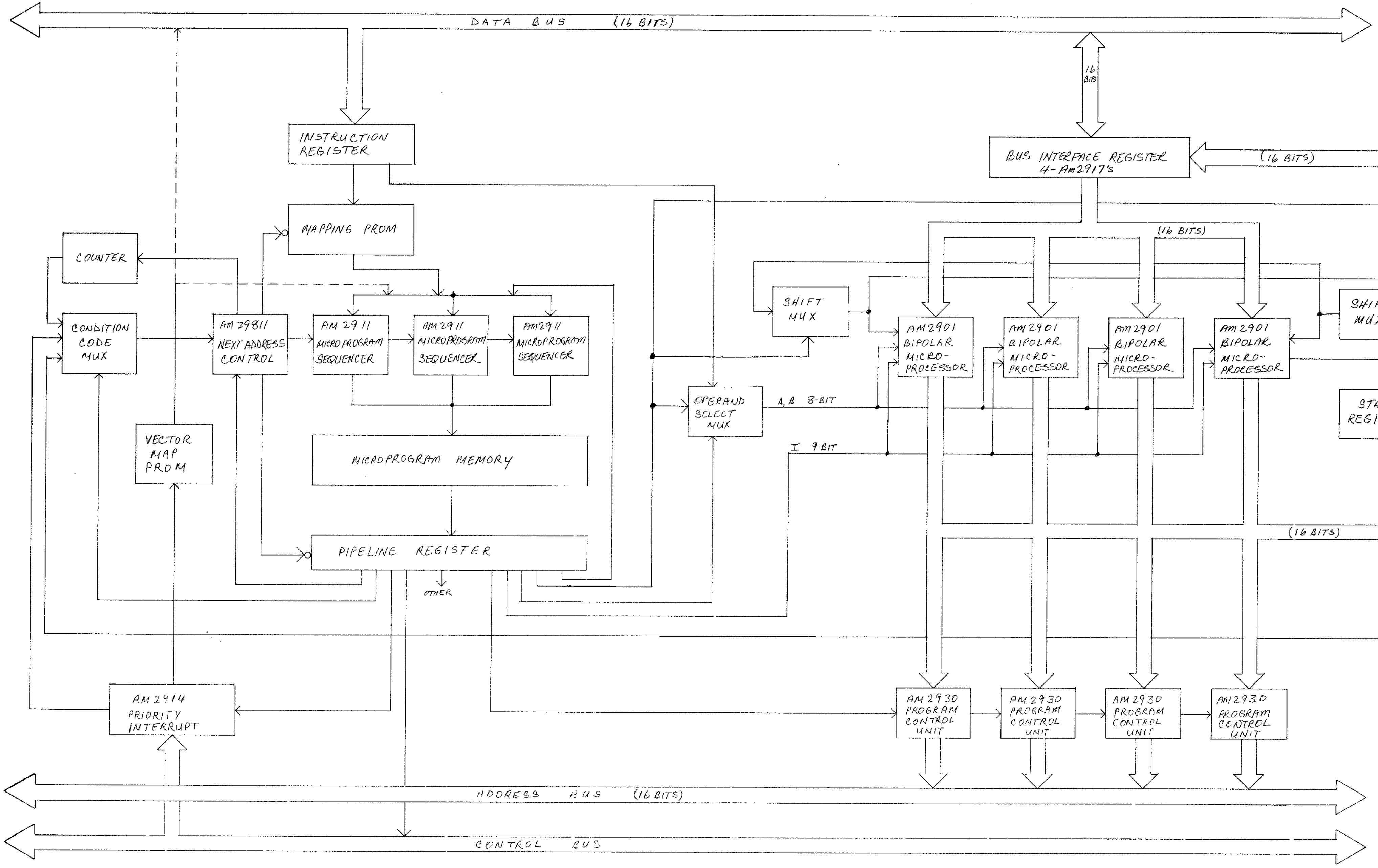

DATA BUS (16 BITS)
16 BITS
INSTRUCTION REGISTER
BUS INTERFACE REGISTER 4-Am2917's
(16 BITS)
MAPPING PROM
(16 BITS)
COUNTER
SHIFT MUX
SHIFT MUX
CONDITION CODE MUX
Am 29811 NEXT ADDRESS CONTROL
AM 2911 MICROPROGRAM SEQUENCER
AM 2911 MICROPROGRAM SEQUENCER
Am2911 MICROPROGRAM SEQUENCER
AM2901 BIPOLAR MICRO-PROCESSOR
Am2901 BIPOLAR MICRO-PROCESSOR
Am2901 BIPOLAR MICRO-PROCESSOR
AM2901 BIPOLAR MICRO-PROCESSOR
OPERAND SELECT MUX
A, B 8-BIT
I 9 BIT
STATUS REGISTER
VECTOR MAP PROM
MICROPROGRAM MEMORY
PIPELINE REGISTER
(16 BITS)
OTHER
AM 2914 PRIORITY INTERRUPT
AM 2930 PROGRAM CONTROL UNIT
Am 2930 PROGRAM CONTROL UNIT
Am 2930 PROGRAM CONTROL UNIT
Am 2930 PROGRAM CONTROL UNIT
ADDRESS BUS (16 BITS)
CONTROL BUS

FIGURE 13. TYPICAL Am2900 MICROPROCESSOR

CONTRACT NO.		ADVANCED MICRO DEVICES 901 THOMPSON PLACE, SUNNYVALE, CALIF.
ORIGINATOR J. MICK + J. BRICK	DATE 10/13/76	TYPICAL Am2900 MICROPROCESSOR
CHECKED	DATE	
APPROVED	DATE	
APPROVED	DATE	SIZE D · CLASS · DRAWING NO.
APPROVED	DATE	SCALE: · SHEET

ADVANCED
MICRO
DEVICES INC.
901 Thompson Place
Sunnyvale
California 94086
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306