DS3578-2.4

MA2910

RADIATION HARD MICROPROGRAM CONTROLLER

The industry standard MA2910 Microprogram Controller forms part of the MA2900 family of devices.

Offering a building block approach to microcomputer and controller design, each device in the range is expandable permitting efficient emulation of any microcode-controlled machine. The family has been designed for operation in severe environments such as space, and is qualified to the highest levels of reliability.

The MA2910 Micro-program Controller is an address sequencer intended for sequence control of microinstructions stored in microprogram memory in high speed microprocessor applications.

All internal elements are full 12 bits wide and address up to 4096 words with one chip. The device has an integral settable 12 bit internal loop counter for repeating instructions and counting loop iterations.

The MA2910 has four address sources which allow Microprogram Address to be selected from the microgram counter, branch address bus, 9 level push/pop stack, or internal holding register.

The MA2910 supports 100ns cycle times and has an integral decoder function to enable external devices onto branch address bus which eliminates the requirement for an external decoder.

FEATURES

- Fully Compatible with Industry Standard 2910A
- CMOS SOS Technology
- Radiation Hard and High SEU Immunity
- High Speed / Low Power
- Fully TTL Compatible

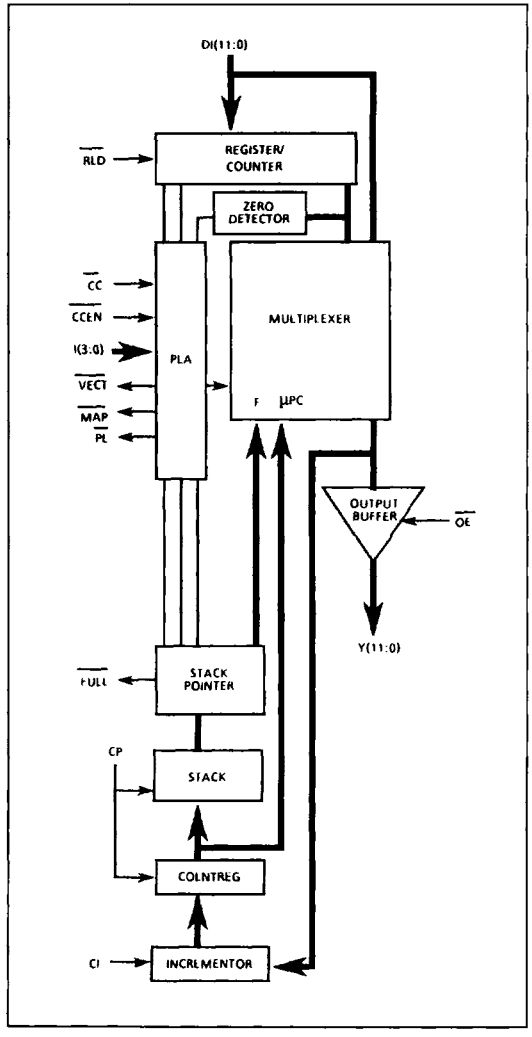


Figure 1: Block Diagram

OPERATION

The MA2910 is a SOS microprogram controller intended for use in high speed microprocessor applications. Besides the capability of sequential access, it provides conditional branching to any microinstruction within its 4096-microword range.

A last-in, first-out stack provides microsubroutine return linkage and looping capability; there are nine nesting levels of microsubroutines. Microinstruction loop count control is provided with a count capacity of 4096.

The device is controlled by 16, 4-bit microinstructions. The PLA decodes the microinstructions on I(3:P) and produces select control codes for the multiplexer, register/counter, microprogram counter register, and stack. The 4-bit microinstructions also generate three active low enable signals ($\overline{PL}$, $\overline{VECT}$, and $\overline{MAP}$) for external use. The operation of each device block is detailed below:

MULTIPLEXER

The MA2910 contains a four-input multiplexer that is used to select either the register/counter, direct input, microprogram counter, or stack as the source of the next microinstruction address.

REGISTER/COUNTER

The register/counter consists of 12 D-type, edgetriggered flip-flops, with a common clock enable. It is operated during microinstructions (8,9,15) as a 12-bit down counter, with result = zero available as a microinstruction branch test criterion. This provides efficient iteration of microinstructions.

The register/ counter is arranged such that if it is preloaded with a number N and is then used as a loop termination counter, the sequence will be executed exactly N+1 times. During instruction 15, a three way branch under combined control of the loop counter and the condition code is available. When its load control, RLD, is LOW, new data is loaded on the next positive control transition.

The output of the register/counter is available to the multiplexer as a source for the next microinstruction address. The direct input furnishes a source of data for loading the register /counter.

MICROPROGRAM COUNTER-REGISTER

The Microprogram Counter Register (μPC) is composed of a 12-bit incrementer followed by a 12-bit register. The (μPC) can be used in one of two ways: When the carry-in to the incrementer is HIGH, the microprogram register is loaded onto the next clock cycle with the current Y output word plus one (Y + 1 → μPC). Sequential microinstructions are thus executed. When the carry-in is LOW, the incrementer passes the Y output unmodified so that the μPC is reloaded with the same Y word on the next clock cycle (Y → μPC). The same microinstruction is thus executed any number of times.

STACK AND STACK POINTER

The third source available at the multiplexer input is a 9-word by 12-bit stack. The stack is used to provide return address linkage when executing microsubroutines or loops. The stack 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 POP.

Explicit control of the stack pointer occurs during instruction 0 (RESET), which makes the stack empty by resetting the SP to zero. After a RESET, and whenever the stack is empty, the contents of the top of the stack are undefined until a push occurs. Any POPs performed while the stack is empty put undefined data on the outputs and leave the stack at zero.

The stack pointer operates as an up/down counter. During microinstructions 1,4, and 5, the PUSH operation may occur. This causes the stack pointer to increment and the file to be written with the required return linkage. On the cycle following the PUSH, the return data is at the new location pointed to by the stack pointer.

During five microinstructions, a POP operation may occur. The stack pointer decrements at the next rising clock edge following a POP, effectively removing old information from the top of the stack.

The stack pointer linkage is such that any sequence of pushes, pops, or stack references can be achieved. At RESET (instruction 0), the depth of nesting becomes zero. For each PUSH, the nesting depth increases by one; for each POP, the depth decreases by one.

PIN DESCRIPTIONS

VDD and GND (Power and Ground)

The MA2910 operates from a single supply voltage of 5V + 10%

D (0 to 11) (Direct Input)

These connections provide direct input to the register/ counter, and the multiplexer. D0 is the least significant bit and D1 the most significant

I (0 to 3) (Instruction bus)

The data on these inputs is read on the rising edge of CP. It determines the instruction to be executed in accordance with table 1.

$\overline{CC}$ (Condition Code)

This active low input is used to determine the result of conditional instruction. LOW indicates a TRUE condition.

$\overline{CCEN}$ (Condition code enable)

This active low input enables the CC input. When CCEN is HIGH, CC is ignored and a conditional operation executed as though CC were LOW (TRUE).

$\overline{CI}$ (Carry input)

When HIGH this input causes the microprogramme counter register to increment on the rising edge of CP. When LOW the counter remains unchanged.

$\overline{RLD}$ (Register load)

This active low input loads the register/counter from the D bus on the rising edge of CP. It will override any HOLD or DEC instruction specified by data on the I bus.

Y (0 to 11) (Microcode address)

This is a 12 bit wide tristate output bus. It carries the microcode address generated according to the instruction read in from the I bus. OE can be used to put the bus in a high impedance state. This allows another to take control of the microcode address bus.

$\overline{OE}$ (Output enable)

This active low input is used to enable the 12 lines of the Y bus.

CP (Clock Pulse)

A LOW-to-HIGH transition on this input is used to trigger all state changes within the device.

$\overline{FULL}$ (stack full)

The active low output FULL indicates that 9 items have been loaded onto the stack .

$\overline{PL}$, $\overline{MAP}$ & $\overline{VECT}$ (pipeline, map and vector)

These active low outputs are set according to the instruction being executed. At any time only one is active.

They may be used to select from one of three possible external sources for microprogramme jumps, being used directly as three-state enables for these sources.

Typically: $\overline{PL}$ enables the primary source of microprogramme jumps, usually part of a pipeline register; $\overline{MAP}$ enables a PROM which maps an instruction to a microcode starting location; $\overline{VECT}$ enables an optional third source, after a vector from DMA or interrupt source.

I_3 - I_0	MNEMONIC	NAME	REGISTER /CONTROL	FAIL CCEN = LOW & CC = HIGH		PASS CCEN = HIGH & CC = LOW		REGISTER/ CONTROL	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	O	CLEAR	HOLD	PL
1	CJS	COND JS P PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL VECTOR	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
10	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
11	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
12	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
13	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
14	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
15	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	PO P	PC	POP	HOLD	P L

Note 1: If $\overline{CCEN}$ = LOW & $\overline{CC}$ = HIGH, hold, else load.

Figure 2: Table of Instructions

INSTRUCTION SET

The MA2910 provides 16 instructions which select the address of the next microinstruction to be executed. 4 of the instructions are unconditional and their effect depends only on the instruction. 10 of the instructions have an effect which is partially controlled by external conditions. 3 of the instructions have an effect which is partially controlled by the contents of the internal register/counter. In this discussion it is assumed the CI is tied HIGH.

In the 10 conditional instructions, the result of the data-dependent test is applied to $\overline{CC}$. If the $\overline{CC}$ input is LOW, the test is considered passed, and the action specified in the name occurs; otherwise, the test has failed and an alternate (often simply the execution of the next sequential microinstruction) occurs. Testing of $\overline{CC}$ may be disabled for a specific microinstruction by setting $\overline{CCEN}$ HIGH, which unconditionally forces the action specified in the name; that is it forces a pass.Other ways of using $\overline{CCEN}$ include; (1) tying it HIGH, which is useful if no microinstruction is data-dependent; (2) tying it LOW if data-dependent instructions are never forced unconditionally; or (3) tying it to the source of MA2910 instruction bit I_0, which leaves instructions 4,6 and 10 as data-dependent but leaves others unconditional. All of these tricks save one bit of microcode width

The effect of three instructions depend upon the contents of the register/counter. Unless the counter holds a value of zero, it is decremented; if it does hold zero, it is held and a different microprogram next address is selected.These instructions are useful for executing a microinstruction loop a finite number of times. Instruction 15 is affected both by the external condition code and the internal register/counter.

The most effective technique for understanding the MA2910 is to simply take each instruction and review its operation. In order to provide some feel for the actual execution of these instructions, examples of all 16 instructions are included.

The examples given should be interpreted in the following manner: The intent is to show microprogram flow as various microprogram memory words are executed.

For example, the CONTINUE instruction (number 14) simply means that the contents of the microprogram memory word 50 are executed, then the contents of word 51 are executed. This is followed by the contents of 52 and 53 The microprogram addresses used in the examples were arbitrarily chosen and have no meaning other than to show instruction flow. The 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. While no special symbology is used for the conditional instructions, the following text will explain what the conditional choices are in each example.

Instruction 0: JZ (Jump to Zero, or Reset).

This instruction unconditionally specifies that the address of the next microinstruction is zero. Many designs use this feature for power-up sequences and provide the power-up firmware beginning at microprogram memory word location 0.

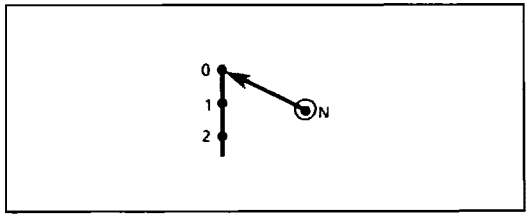


Figure 3: 0 JUMP ZERO (JZ)

Instruction 1: Conditional Jump-to-Subroutine.

This instruction is a conditional Jump-to-Subroutine via the address provided in the pipeline register. As shown in figure 4, the machine might have executed words at address 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 test is passed, the next instruction executed will be the contents of microprogram memory location 90. If the test has failed, 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 the instruction either in location 90 or in location 53 to be executed next. If the test input is such that the location 90 is selected, value 53 will be pushed onto the 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 found at location 93.

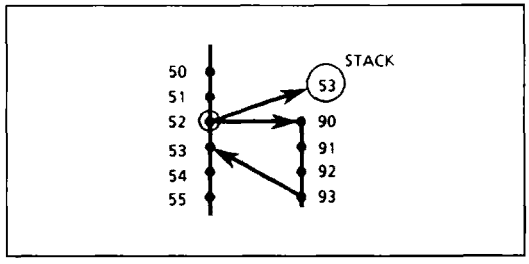


Figure 4: COND JSB PL (CJS)

Instruction 2: Jump-Map.

This is an unconditional instruction which causes the $\overline{MAP}$ output to be enabled so that the next microinstruction location is determined by the address supplied via the mapping PROMs. Normally, the JUMP MAP instruction is used at the end of the instruction fetch sequence for the machine.

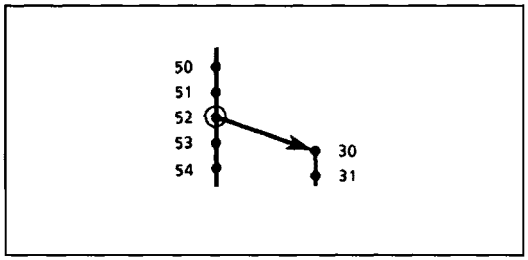


Figure 5: 2 JUMP MAP (JMAP)

In the example of Figure 5, 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 3: Conditional Jump Pipeline.

This instruction derives its branch address from the pipeline register branch address value (BR_0-BR_{11}). 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 functions. This usually has the effect of resetting the input under test until some point in the future.

The example 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 be either location 53 or 30, in this example. If the test is passed, the value currently in the pipeline register (30) will be selected. If the test fails, the next address selected will be contained in the microprogram counter which, in this example, is location 53.

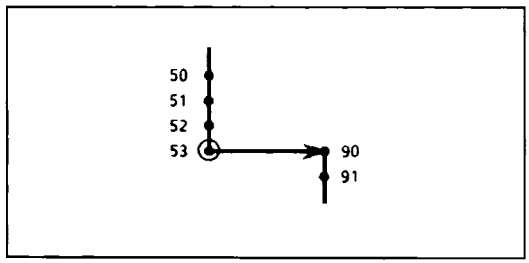


Figure 6: 3 COND JUMP PL (CLP)

Instruction 4: Push/Conditional, Load Counter.

This instruction is used primarily for setting up loops in microprogram firmware. In this example, when instruction 52 is in the pipeline register, a PUSH will be made onto the stack and the counter will be loaded based on the condition. When a PUSH occurs, the value pushed is always the next sequential instruction address. In this case, the address is 53. If the test fails, the counter is not loaded; if it is passed, the counter is 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 8 will describe how to use the pushed value and the register/ counter for looping.

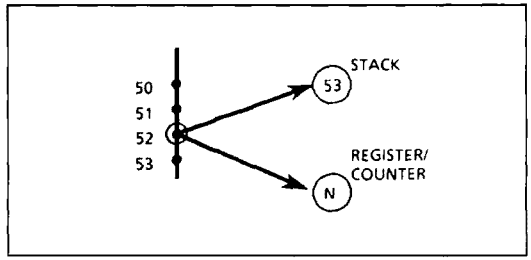


Figure 7: 4 PUSH/COND LD CNTR (PUSH)

Instruction 5: Conditional Jump-to-Subroutine.

This instruction is a Conditional Jump-to-Subroutine via the register/counter of the contents of the PIPELINE register. 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. Lets assume that the branch address

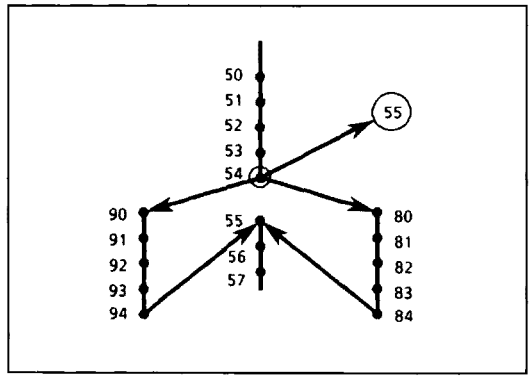


Figure 8: 5 COND JSB R/PL (JSRP)

fields of instruction 53 contain the value 90 so that it will be in the MA2910 register/counter when the contents of the address 54 are in the pipeline register.

This requires that the instruction at address 53 loads the register/counter. Now,during the execution of instruction 5 (at address 54), if the test failed, the contents of the register (value=90) will select the address of the next microinstruction. If the test input passes, 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 6: Conditional Jump Vector.

This instruction provides the capability to take the branch address from a third source heretofore not discussed. In order for this instruction to be useful, the MA2910 output $\overline{VECT}$ is used to control a three-state control input of a register, buffer, or PROM containing the next microprogram address. This instruction provides one technique for performing interrupt type branching at the microprogram level. Since this instruction is conditional, a pass causes the next address to be taken from the vector source, while failure causes the next address to be taken from the microprogram counter.

In the example, if the Conditional Jump Vector instruction is contained at location 52, execution will continue at vector address 20 if the $\overline{CC}$ input is LOW and the microinstruction at address 53 will be executed if the $\overline{CC}$ input is HIGH.

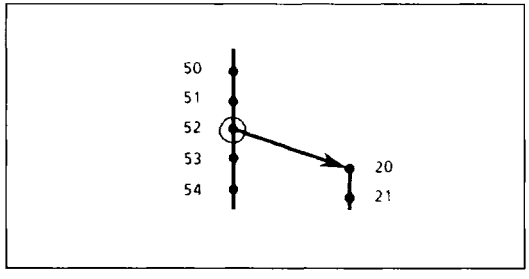


Figure 9: 6 COND JUMP VECTOR (CJV)

Instruction 7: Conditional Jump.

Conditional Jump via the contents of the MA2910 Register/ Counter or the contents of the Pipeline register. This instruction is very similar to instruction 5; the Conditional Jump-to-subroutine via R or PL. The major difference between instruction 5 and instruction 7 is that no push onto the stack is performed with 7.

The example depicts this instruction as a branch to one of the two locations depending on the test condition. The example assumes the pipeline register contains the value 70 when the contents of address 52 are being executed. As the contents of address 53 are clocked into the pipeline register, the value 70 is loaded into the register/counter in the MA2910. The value 80 is available when the contents of the address 53 are in the pipeline register. Thus, control is transferred to either address 70 or address 80 depending on the test condition.

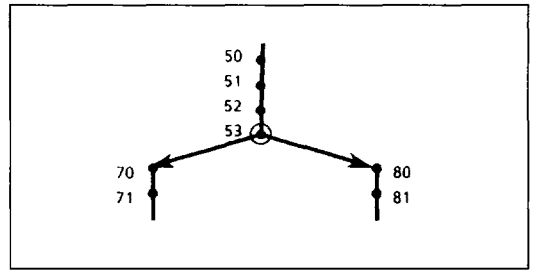


Figure 10: 7 COND JUMP R/PL (JRP)

Instruction 8: Repeat Loop, Counter ≠ Zero.

This microinstruction makes use of the decrementing capability of the register/counter. To be useful, some previous instruction, such as 4, must have loaded a count value into the register/counter. This instruction checks to see whether the register/counter contains a non-zero value. If so, the register/ counter is decremented, and the address of the next microinstruction is taken from the top of the stack.

If the register/counter contains zero, the loop exit condition is occurring; control falls through to the next sequential microinstruction by selecting µPC; the stack is POP'd by decrementing the stack pointer, but the contents of the top of the stack are thrown away.

In this example, location 50 is most likely to have contained 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 this example, since the loop test is made at the end of the instructions to be repeated (microaddress 54), the proper value to be loaded by the instructions at address 50 is one less than the desired number of passes through the loop .

This method allows a loop to be executed 1 to 4096 times. If it desired to execute the loop from 0 to 4095 times, the firmware should be written to make the loop exit test immediately after loop entry.

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.

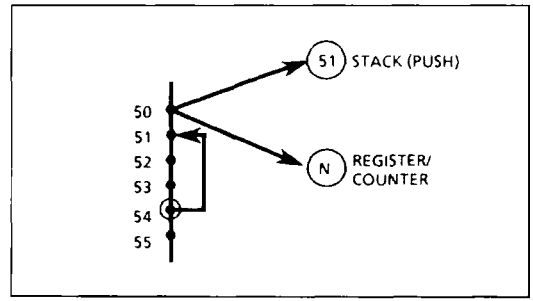


Figure 11: 8 ERPEAT LOOP, CNTR ≠ 0 (RFCT)

Instruction 9: Repeat Pipeline Register, Counter ≠ Zero

This instruction is similar to instruction 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 extension; that is, by using this instruction, a loop with the counter can still be performed when subroutines are nested nine deep. This instruction's operation is very similar to that of instruction 8. The differences are that on this instruction, a failed test condition causes the source of the next microinstruction address to be the D inputs; and, when the test condition is passed, this instruction does not perform a POP because the stack is not being used.

In this example, the REPEAT PIPELINE, COUNTER J ZERO instruction is instruction 52 and is shown as a single microinstruction loop. The address in the pipeline register would be 52. Instruction 51 in this example could be the LOAD COUNTER AND CONTINUE instruction (number 12). While the example shows a single microinstruction loop, by simply changing the address in a pipeline register, multi-instruction loops can be performed in this manner for a fixed number of times as determined by the counter.

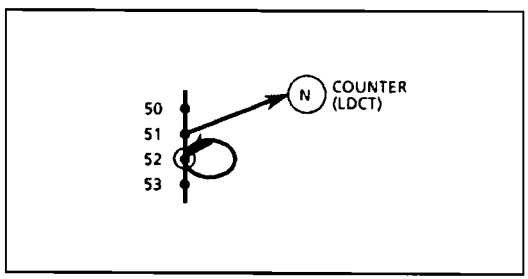


Figure 12: 9 REPEAT PL, CNTR ≠ 0 (RPCT)

Instruction 10: Conditional return form Subroutine.

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 performed only if the test is passed.

If the test is failed, the next sequential microinstruction is performed. This example depicts the use of the conditional RETURN-FROM-SUBROUTINE instruction in both the conditional and the unconditional modes.

This example first shows a JUMP-TO-ROUTINE at instruction location 52 where control is transferred to location 90. At location 93, a conditional RETURN-FROM-SUBROUTINE instruction is performed. If the test is passed, the stack is accessed and the program will transfer to the next instruction at address 53. If the test is failed, the next microinstruction at address 94 will be executed, the program will continue to address 97 where the subroutine is complete. To perform an unconditional RETURN-FROM-SUBROUTINE, the conditional RETURN-FROM-SUBROUTINE instruction is executed unconditionally; the microinstruction at address 97 is programmed to force $\overline{CCEN}$ HIGH, disabling the test and the forced PASS causes an unconditional return.

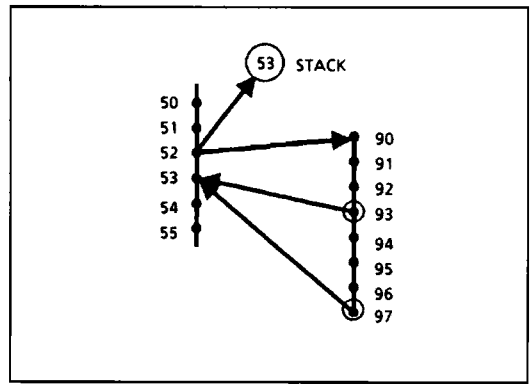


Figure 13: 10 COND RETURN (CRTN)

Instruction 11: Conditional Jump Pipeline register address and POP stack.

This instruction provides another technique for loop termination and stack maintenance. The example shows a loop being performed from address 55 back to address 51. The instructions at locations 52,53, and 54 are all conditional JUMP and POP instructions. At address 52, if the $\overline{CC}$ input is LOW, a branch will be made to address 70 and the stack will be properly maintained via a POP. Should the test fail, the instruction at location 53 (the next sequential instruction) will be executed. Likewise, at address 53, either the instruction at 90 or 54 will be subsequently executed, respective to the test being passed or failed. The instruction at 54 follows the same rules, going to either 80 or 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 occur before proceeding to another sequence of instructions. This provides the powerful jump-table programming technique at the firmware level .

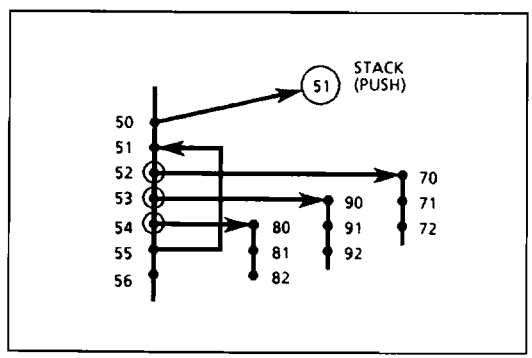


Figure 12: 9 REPEAT PL, CNTR ≠ 0 (RPCT)

Instruction 12: Load Counter and Continue.

This instruction 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.

Altogether there are three ways of loading the counter: the explicit load by this instruction 12; the conditional load included as part of instruction 4; and use of $\overline{RLD}$ input along with any instructions.

The use of $\overline{RLD}$ with any instruction overrides any counting or decrementation specified in the instruction, calling for a load instead. Its use provides additional microinstruction power, at the expense of one bit of microinstruction width

Instruction 12 is exactly equivalent to the combination of instruction 14 and $\overline{RLD}$ LOW. Its purpose is to provide a simple capability to load the register/counter in those implementations which do not provide microprogrammed control for $\overline{RLD}$.

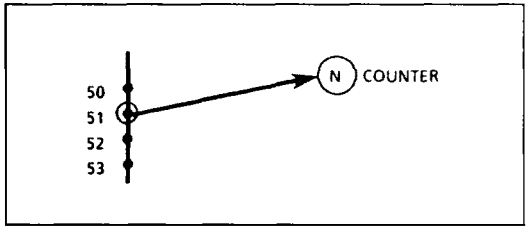


Figure 15: 12 LD CNTR & CONTINUE (LDCT)

Instruction 13: Test End-of-Loop.

This instruction provides the capability of conditionally 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 test is failed, else to continue to the next sequential instruction.

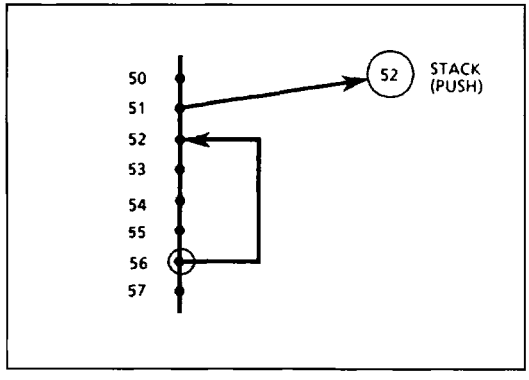


Figure 16: 13 TEST END LOOP (LOOP)

The example shows the TEST END-OF-LOOP microinstruction at address 56. If the test fails, the microprogram will branch to address 52. Address 52 is on the stack because a PUSH instruction had been executed at address 51. If the test is passed at instruction 56, the loop is terminated and the next sequential microinstruction at address 57 is executed which also causes the stack to be POP'd; thus accomplishing the required stack maintenance.

Instruction 14: CONTINUE.

This simply causes the microprogram counter to increment so that the next sequential microinstruction is executed. This is the simplest microinstruction of all and should be the default instruction which the firmware requests whenever there is nothing better to do.

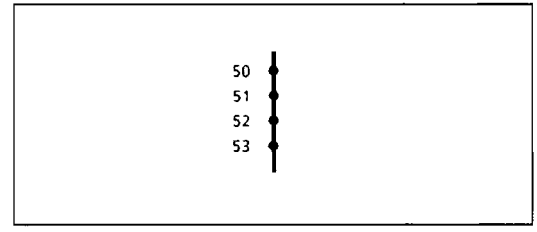


Figure 17: 14 CONTINUE (CONT)

Instruction 15: Three-Way-Branch.

This instruction is the most complex and provides for testing of both a data-dependent condition and the counter during one microinstruction and provides for selecting among one of three microinstruction addresses as the next microinstruction to be performed. Like instruction 8, a previous instruction will have loaded a count into the register/counter while pushing a microbranch address onto the stack.

Instruction 15 performs a decrement-and-branch-until-zero function similar to instruction 8. The next address is taken from the top of the stack until the count reaches zero. When the counter reaches zero the next address comes from the pipeline register. The above action continues as long as the test condition fails. If at any execution of instruction 15 the test condition is passed, no branch is taken and the microprogram counter register furnishes the next address. When the loop is ended, either by a count becoming zero, or by passing the conditional test, the stack is POP'd by decrementing the stack pointer, since interest in the value contained at the top of the stack is then complete.

The application of instruction 15 can enhance performance of a variety of machine-level instructions. For instance: (1) a memory search instruction to be terminated either by finding a desired memory content or by reaching the search limit; (2) variable-field-length arithmetic terminated early upon finding that the content of the portion of the field still unprocessed is all zeroes; (3) key search in a disc controller processing variable length records; (4) normalization of a floating point number.

As one example, consider the case of a memory search instruction. As shown, the instruction at microprogram address 63 can be instruction 4 (PUSH), which will push the value 64 onto the microprogram stack and load the number N, which is one less than the number of memory locations to be searched before ending the search. Location 64 contains a microinstruction which fetches the next operand from the memory area being searched and compares it with the search key. Location 65 contains a microinstruction which tests the result of the comparison and is a THREE-WAY BRANCH for microprogram control. If no match is found, the test fails and the microprogram goes back to location 64 for the next operand address.

When the count becomes zero, the microprogram branches to location 72, and carries out the instruction at location 72, if no match is found. If a match occurs on any execution of the THREE-WAY BRANCH at location 65, control falls through to location 66 which handles the case. Whether the instruction ends by finding a match or not, the stack will have been POP'd once thus removing the value 64 from the top of the stack.

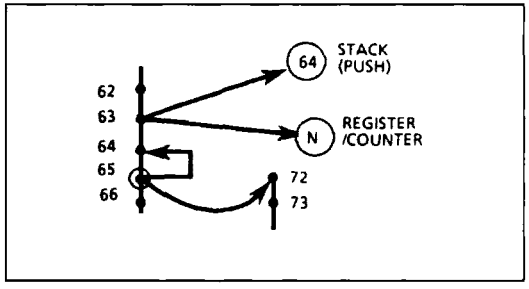


Figure 18: 15 THREE-WAY BRANCH (TWB)

ARCHITECTURE

ONE LEVEL PIPELINE BASED (RECOMMENDED)

One level pipeline provides better speed than most other architectures as the Microprogram Memory and the MA2901 array are in parallel paths.

This is the recommended architecture for all MA2900 designs.

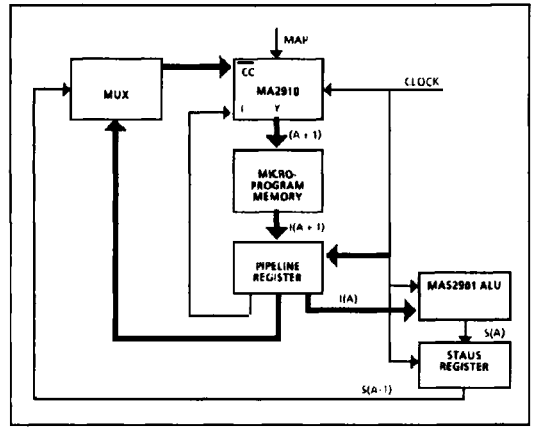


Figure 19a: One level Pipeline Based

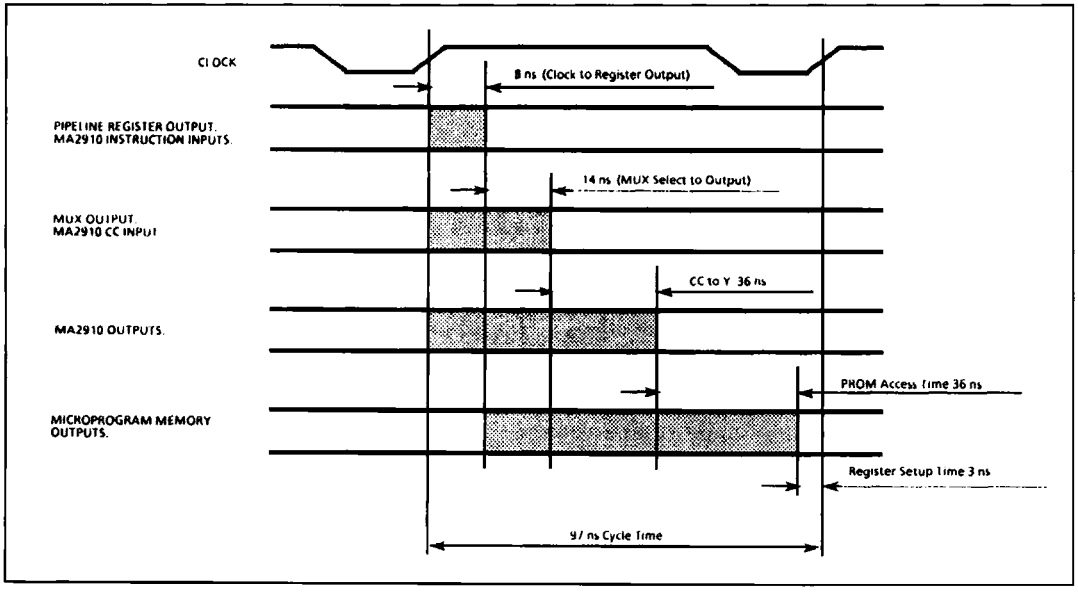


Figure 19b: Timing relationship in the CCU

Instruction Based

A Register at the Microprogram Memory output contains the microinstruction being executed. The Microprogram Memory and MA2901 delay are in series. Conditional branches are executed on the same cycle as the ALU operation generating the condition.

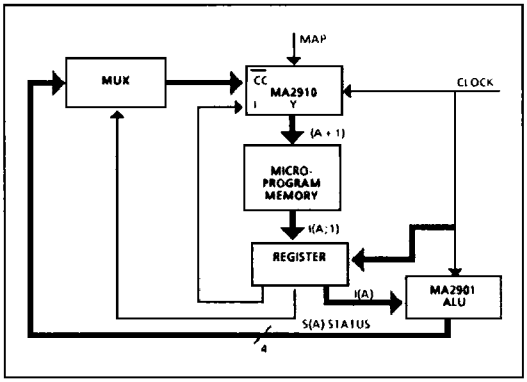


Figure 20: Instruction Based

Data Based

The Status Register provides conditional branch control based on results of the previous ALU cycle. The Microprogram memory and the MA2901 are in series within the critical path.

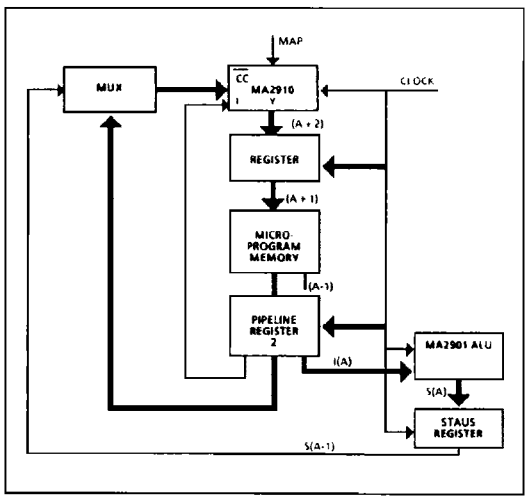


Figure 21: Data Based

Address Based

The Register at the MA2910 output contains the microinstruction being executed. The Microprogram Memory and MA2901 are in series within the critical path. This architecture is of comparable speed to the Instruction Based architecture, but requires fewer register bits, since only the address (typically 10 to 12 bits) is stored instead of the instruction.

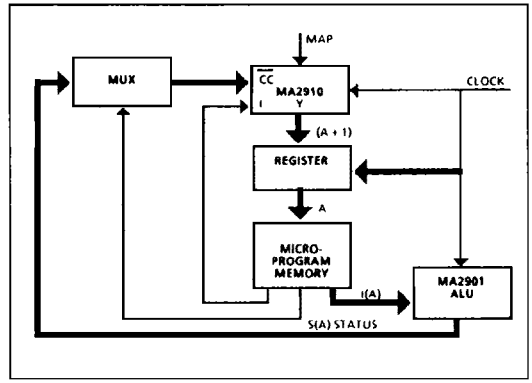


Figure 22: Address Based

Two Level pipeline Based

This architecture provides the highest possible speed. It is, however, more difficult to program as the selection of a microinstruction occurs two instructions ahead of its execution.

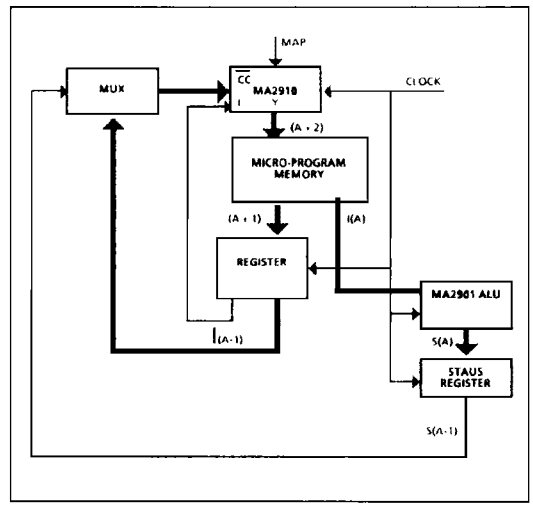


Figure 23: Two Level Pipeline Based

DC CHARACTERISTICS AND RATINGS

Parameter	Min	Max	Units
Supply Voltage	-0.5	7	V
Input Voltage	-0.3	V_{DD}+0.3	V
Current Through Any Pin	-20	+20	mA
Operating Temperature	-55	125	°C
Storage Temperature	-65	150	°C

Note: Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these conditions, or at any other condition above those indicated in the operations section of this specification, is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Figure 24: Absolute Maximum Ratings

Subgroup	Definition
1	Static characteristics specified in Figure 26 at +25°C
2	Static characteristics specified in Figure 26 at +125°C
3	Static characteristics specified in Figure 26 at -55°C
9	Switching characteristics specified in Figures 27 to 29 at +25°C
10	Switching characteristics specified in Figures 27 to 29 at +125°C
11	Switching characteristics specified in Figures 27 to 29 at -55°C

Figure 25: Definition of Subgroups

Symbol	Parameter	Conditions	Total dose radiation not exceeding $3x10^5$ Rad (Si)			Units
			Min.	Typ.	Max.	
V_{DD}	Supply voltage	-	4 5	5.0	5 5	V
V_{IH}	Input high voltage	-	2.0	-	-	V
V_{IL}	Input low voltage	-	-	-	0 8	V
V_{OH}	Output high voltage	I_{OH} = -2mA	2.4	-	-	V
V_{OL}	Output low voltage	I_{OL} = 5mA	-	-	0.4	V
I_{IN}	Input leakage current (Note 1)	V_{DD} = 5.5V, V_{IN} = V_{SS} or V_{DD}	-	-	±10	µA
I_{OZ}	Tristate leakage current (Note 1)	V_{DD} = 5.5V, V_{IN} = V_{SS} Or V_{DD}	-	-	±50	µA
I_{DD}	Power supply current	Static, V_{DD} = 5.5V	-	0.1	10	mA

Mil-Std-883, method 5005, subgroups 1, 2, 3
V_{DD} = 5V ±10%, over full operating temperature range.
Note 1: Worst case at T_A = +125°C, guaranteed at T_A = -55°C. 300K Rad(Si) values at higher radiation levels are available on request.

Figure 26: Operating Electrical Characteristics

AC ELECTRICAL PARAMETERS

1. V_{DD} = 5V ±10%. C_{CL} = 50pF
2. Operating temperature is specified when ordering (see ordering information section on last page).
3. Enable/Disable times measured to 0.5V change on output voltage level with C_L = 50pF.
4. Time measurement Reference Level = 1.5 Volts.
5. Input Pulse = V_{SS} to 3.0 Volts.
6. Set-up and hold times measured relative to CP.

Input	t_s	t_h
Di → R	16	5
Di → PC	20	5
I_0-I_3	30	5
CC	35	0
CCEN	35	0
CI	15	5
RLD	15	5

Figure 27: Set-up and Hold Times

Minimum Clock LOW Time	20ns
Minimum Clock HIGH Time	35ns
Minimum Clock Period	55ns

Figure 28: Clock Requirements

Input	Y	PL, VECT, MAP	FULL
D_0-D_{11}	30	-	-
I_0-I_3	45	30	-
CC	45	-	-
CCEN	45	-	-
CP	60	-	32
OE Enable (Note 1)	25	-	-
OE Disable (Note 1)	25	-	-

Figure 29: Combinational Delays

Mil-Std-883, method 5005, subgroups 9, 10, 11

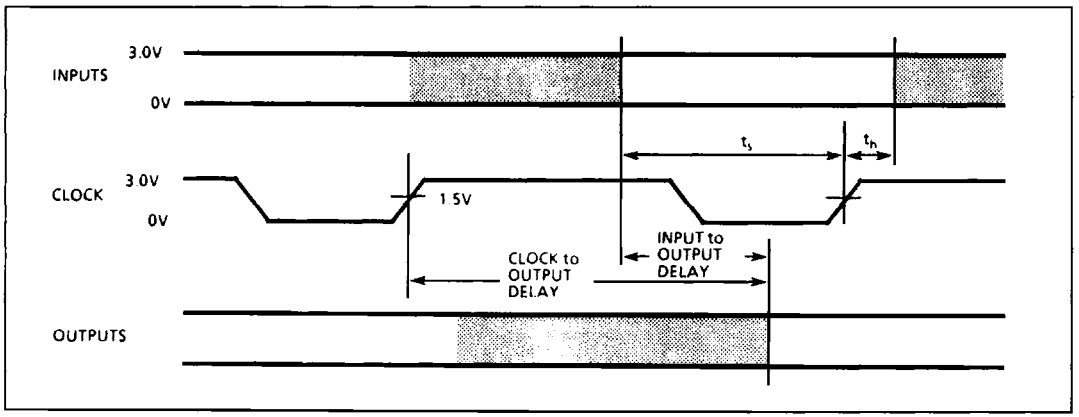
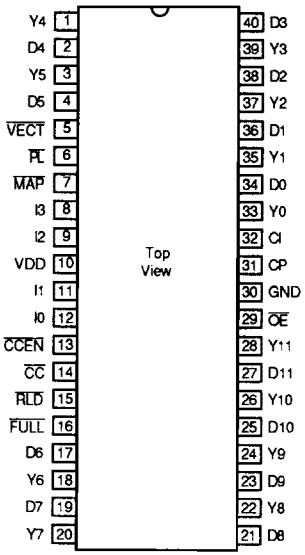


Figure 30: AC Timings

OUTLINES & PIN ASSIGNMENTS

Ref	Millimetres			Inches		
	Min.	Nom.	Max.	Min.	Nom.	Max.
A	-	-	5.715	-	-	0.225
A1	0.38	-	1.53	0.015	-	0.060
b	0.35	-	0.59	0.014	-	0.023
c	0.20	-	0.36	0.008	-	0.014
D	-	-	51.31	-	-	2.020
e	-	2.54 Typ.	-	-	0.100 Typ.	-
e1	-	15.24 Typ.	-	-	0.600 Typ.	-
H	4.71	-	5.38	0.185	-	0.212
Me	-	-	15.90	-	-	0.626
Z	-	-	1.27	-	-	0.050
W	-	-	1.53	-	-	0.060

XG405



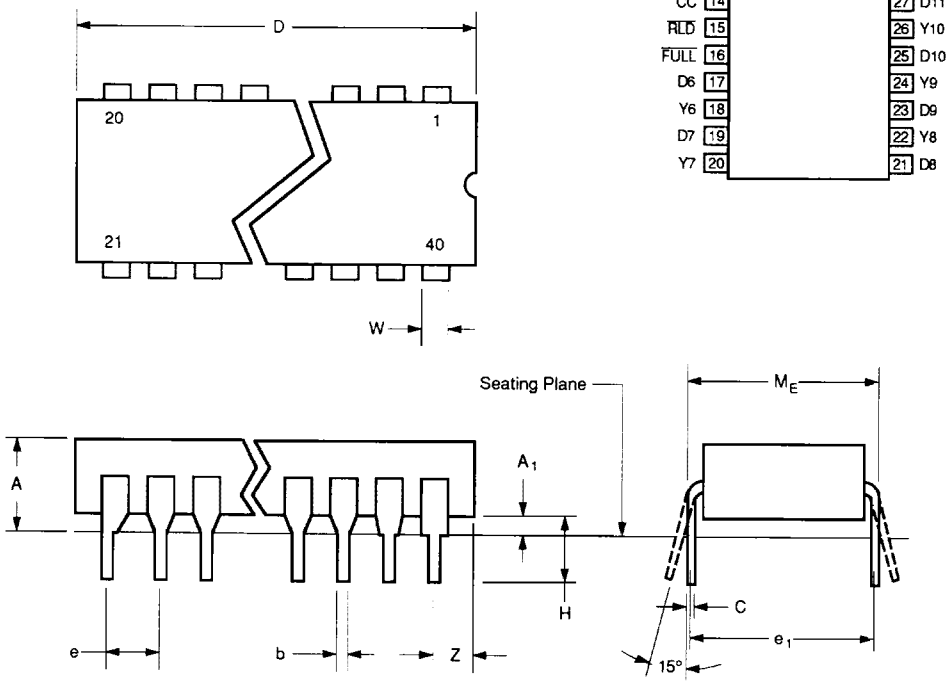


Figure 31: 40-Lead Ceramic DIL (Solder Seal) - Package Style C

RADIATION TOLERANCE

Total Dose Radiation Testing

For product procured to guaranteed total dose radiation levels, each wafer lot will be approved when all sample devices from each lot pass the total dose radiation test.

The sample devices will be subjected to the total dose radiation level (Cobalt-60 Source), defined by the ordering code, and must continue to meet the electrical parameters specified in the data sheet. Electrical tests, pre and post irradiation, will be read and recorded.

GEC Plessey Semiconductors can provide radiation testing compliant with Mil-Std-883 method 1019 Ionizing Radiation (total dose) test.

Total Dose (Function)	$1x10^6$ Rad(Si)
Transient Upset	$>1x10^{10}$ Rad(Si)/sec
Neutron Hardness (Function to specification)	$1x10^{15}$ neutrons/cm^2
Single Event Upset (GSO 10% worst case)	$<10^{-10}$ errors/bitday
Latch-up	Not possible

Figure 32: Radiation Hardness Parameters

ORDERING INFORMATION