FYI

Firmware Level Interrupts for the 2900

Dr. Donnamaie E. White
Advanced Micro Devices, Inc.
Sunnyvale, CA

Interrupt handling at the firmware or software level is desirable when the interrupts are recognized under programmer control. These "software-interrupts" are used in emulations and basic CPU structures, while real-time or "hard interrupts" are necessary in controller environments. Firmware level interrupts for a 2900 Family-based Computer Control Unit (CCU) or controller can be implemented with a 2914 Priority Interrupt Circuit (one can handle up to 8 levels of prioritized interrupts), a 2910 Microprogram Sequencer, registers, PROMs for implementing a memory map, a vector map and microprogram control memory.

The instruction register contains the current machine instruction, or macroinstruction. The memory-mapping PROM accepts the instruction register (IR) opcode portion on its address lines and produces an output microprogram address, which is the start address for the microroutine that implements the hardware steps to support the opcode.

The vector-mapping PROM accepts the 2914 priority-encoded vector output on its address input and generates a microprogram address on its output lines, which is the start address for the interrupt routine that will service the interrupt.

The 2914 accepts up to 8 interrupt requests encoded by an internal priority interrupt encoder. The interrupts can be individually masked, in which case they do not reach the encoder, and interrupts may also be nested under programmer control. A status fence prevents interrupts (of a priority less than that level currently in service) from causing an interrupt request. Both the status fence register and the mask register may be stored in the main memory of the system or may be loaded from the main memory of the system.

Fig 1 shows output enables for the three microprogram address sources: (1) the branch address/counter field in the microword of the microinstruction register (pipeline register for the control memory); (2) the vector map output and (3) the mapping PROM output are all driven from fields in the microword. This averages 20 ns faster than sourcing these signals from the 2910.

The microword for the structure shown in Fig 1 is given in Fig 2. The microword format is shown in a structured layout, conceptually clear to a human reading the microprogram.

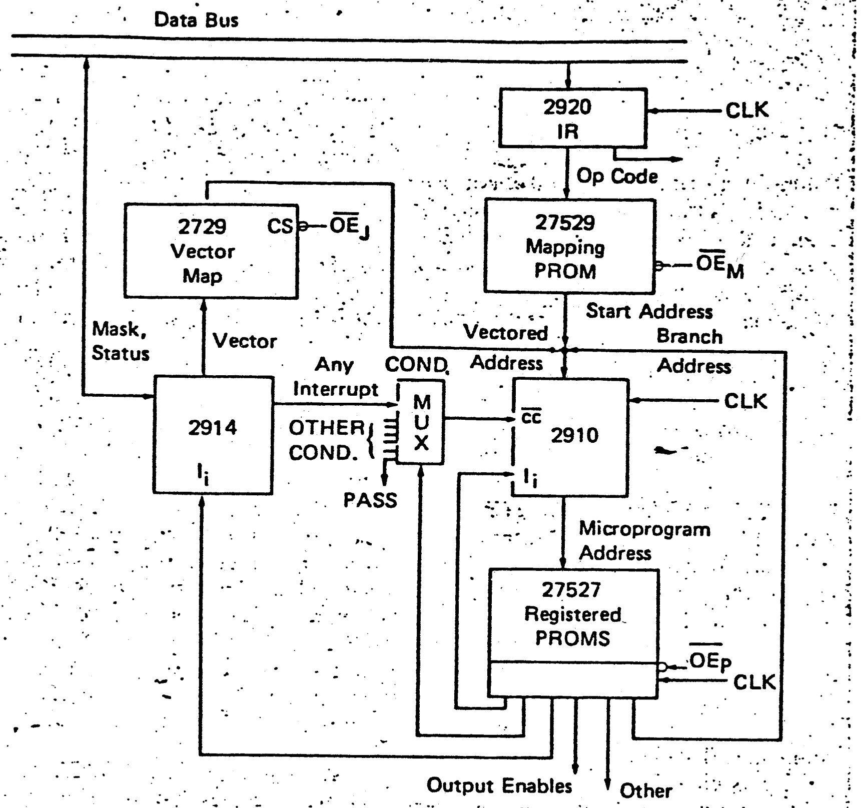

Figure 1: Firmware level interrupts for the 2900 series.

← Sequence		Control →	← Interrupt →		← Output	Enables →		
2910 Inst.	Cond Mux.	BR Addr/ Counter	2914 INST-/EN	$\overline{\text{2914}}$ INT DIS	$\overline{OE}$ PL	$\overline{OE}$ Map	$\overline{OE}$ Vect	Other
4	3	12	5	1	1	1	1	• • •

Figure 2: The microword conceptual layout

From left to right the microword fields shown are: (1) 2910 Instruction Field — the 4 bits required for the basic sixteen instructions (there are more instructions). (2) Conditional MUX — the multiplexer used to select which test input will be examined: a 1-of-8 MUX is chosen here. Tests possible besides the 'ANY INTERRUPT TEST are PASS, ALU status (from a status register), etc. (3) Branch Address or 2910 Counter — the field used to specify an address or a value. (4) 2914 Instruction Field — the basic four instruction lines plus the I_{EN}line. (5) 2914 $\overline{INT}$ DIS — the interrupt disable; used to protect against interrupts during start-up in sensitive code areas, etc. (6) $\overline{OE}_{PL}$ — output enable for the branch address field. (7) $\overline{OE}_{MAP}$ — output enable for the memory mapping PROM. (8) $\overline{OE}_{VECT}$ — output enable for the vector mapping PROM.

For handling interrupts, several options are available in the hardware shown in **Fig 1** and the microword in Fig 2. Using a typical CPU microprogram structure as shown in **Fig 3,** the flow is characterized by the common microinstruction steps necessary for each execution; i.e., load PC to MAR; fetch next macroinstruction to Instruction Register and decode the opcode.

The PC is incremented as part of the PC→MAR step in its own step, or even as part of the fetch instruction step. Within this common code sequence, usually at the completion of the current microinstruction microroutine, a test can be made for the existence of an interrupt request.

Because the test is at a given location, such as the end of a microroutine or the beginnng of a common code section, there is no need to use a subroutine. All routines return to a fixed address.

The microinstruction sequence is shown in A.

At the interrupt routine address generated by the vector map, the interrupt is first cleared from the 2914 and acknowledged (if the interrupt is a level signal rather than a pulse). The end of the interrupt routine would have an unconditional jump to the common code section, labeled START in the code shown in B.

Unconditional jumps can be generated via a PASS on the multiplexer (CC/2910) is active LOW) or via the CCENpin of the 2910. This method uses the multiplexer solution. During the start of the routine it may be desirable to load or to save either the Mask Register, the Status Register or both.

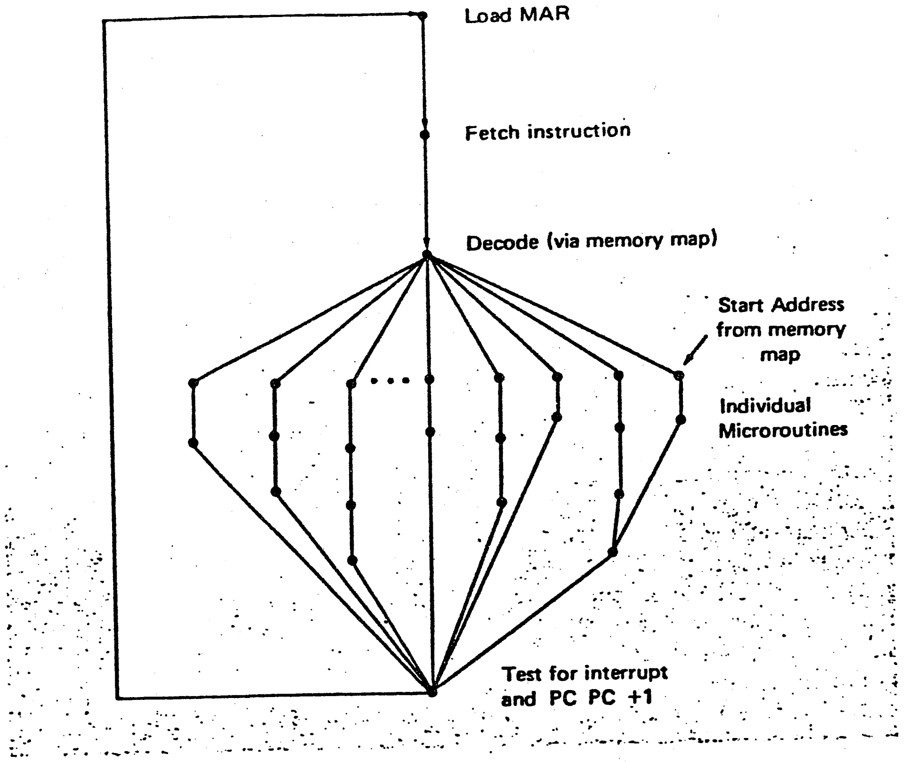

Figure 3: Sample CPU flow form.

At the end of the microroutine, it may be desirable to restore these registers. The status register must be lowered as it is automatically set when the 2914 READ VECTOR instruction is executed.

Second, if an individual microroutine is long, it may be desirable to test for an interrupt during some quiescent point within the microroutine. This is possible using the hardware and microword previously given. The test in this instance would be a conditional jump ~~to~~ subroutine where the subroutine start address is provided via the vector map. See C.

The subroutine, which would be similar in its initial and final steps to any other interrupt routine would return to the calling location via an unconditional return. See D.

Nested interrupts are permissable with either approach, provided that the interrupt routine itself tests for interrupts. The stack limit on the 2910 is five.

A

2910 INST	COND MUX	BR# ADDR COUNTER	2914 INST	$\overline{I}_{EN}$	2914 $\overline{\text{INT DIS}}$	$\overline{OE}_{PL}$	$\overline{OE}_{MAP}$	$\overline{OE}_{VECT}$
CJV OR CJP	ANY	#	READ VECTOR	EN	EN	DIS	DIS	EN

B

2910 INST	COND MUX	BR# ADDR COUNTER	2914 INST	$\overline{I}_{EN}$	2914 $\overline{\text{INT DIS}}$	$\overline{OE}_{PL}$	$\overline{OE}_{MAP}$	$\overline{OE}_{VECT}$
CJP	PASS	START	#	DIS	EN	EN	DIS	DIS

C

2910 INST	COND MUX	BR# ADDR COUNTER	2914 INST	$\overline{I}_{EN}$	2914 $\overline{\text{INT DIS}}$	$\overline{OE}_{PL}$	$\overline{OE}_{MAP}$	$\overline{OE}_{VECT}$
CJS	ANY	#	READ VECTOR	EN	EN	DIS	DIS	EN

D

2910 INST	COND MUX	BR# ADDR COUNTER	2914 INST	$\overline{I}_{EN}$	2914 $\overline{\text{INT DIS}}$	$\overline{OE}_{PL}$	$\overline{OE}_{MAP}$	$\overline{OE}_{VECT}$
CRTN	PASS	#	#	DIS	EN	EN	DIS	DIS

Figure 4: Microintruction routines for handling interrupts