CHAPTER 9

RESET AND INTERRUPT CONSIDERATIONS

9.0 VECTORS

Before developing the concepts of how the MCS650X Microprocessors handle interrupts and start-up, a brief definition of the concept of vector pointers needs to be developed.

In the sections on Jumps and Branches, it was always assumed that the program counter is changed by the microprocessor under control of the programmer while accessing addresses which were in program sequence. In order to get the microprocessor started and in order to properly handle external control or interrupt, there has been developed a different way of setting the program counter to point at a specific location. This concept is called vectored pointers. A vector pointer consists of a program counter high and program counter low value which, under control of the microprocessor, is loaded in the program counter when certain external events occur. The word vector is developed from the fact that the microprocessor directly controls the memory location from which a particular operation will fetch the program counter value and hence the concept of vector.

By allowing the programmer to specify the vector address and then by allowing the programmer to write coding that the address points to, the microprocessor makes available to the programmer all of the control necessary to develop a general purpose control program. The microprocessor has fixed address in memory from which it picks up the vectors. By this

implementation, minimum hardware in the microprocessor is obtained. Locations FFFA through FFFF are reserved for vector pointers for the microprocessor. Into these locations are stored respectively the interrupt vectors or pointers for: non-maskable interrupt, reset and interrupt request.

9.1 RESET OR RESTART

In the microprocessor, there is a state counter which controls when the microprocessor is going to use the program counter to access memory to pick up an instruction, then after the instruction is loaded, the microprocessor goes through a fixed sequence of interpreting instructions and then develops a series of operations which are based on the OP CODE decoding.

Up to this point, it has been assumed that the program counter was set at some location and that all program counter changes are then directed by the program once the program counter had been initialized.

Instructions exist for the initialization and loading of all other registers in the microprocessor except for the initial setting of the program counter. It is for this initial setting of the program counter to a fixed location in the restart vector location specified by the microprocessor programmer that the reset line in the microprocessor is primarily used.

The reset line is controlled during power on initialization and is a common line which is connected to all devices in the microcomputer system which have to be initialized to a known state. The initialization of most I/O devices is such that they are brought up in a benign state such that with minimum coding in the microcomputer, the programmer can configure and control the I/O in an orderly fashion.

The concept has important systems implications in systems where damage can be done if peripheral devices came up in unknown states. Therefore, in the MCS650X, power on or reset control operates at two levels.

First, by holding of an external line to ground, and having this external line connected to all the devices during power up transient conditions, the entire microcomputer system is initialized to a known disabled state. Second, the releases of the reset line from the ground or TTL zero condition to a TTL one condition causes the microprocessor to be automatically initialized, first by the internal hardware vector which causes it to be pointed to a known program location, and secondly through a software program which is written by the user to control the orderly start-up of the microcomputer system.

All of the MCS650X family parts also obey a discipline that while the reset line is low, the system is in a stop or reset state. The microprocessor is guaranteed to be in a Read state and upon release of the reset line from ground to positive, the microprocessor will continue to hold the line in a Read state until it has addressed the specified vectored count location, at which time control of the microprocessor is available to the programmer.

NOTE: The MC6800 family also follows this convention.

9.2 START FUNCTION

While the reset line is in the low state, it can be assumed that internal registers may be initialized to any random condition; therefore, no conditions about the internal state of the microprocessor are assumed other than that the microprocessor will, one cycle after the reset line goes high, implement the following sequence:

Example 9.1: Illustration of Start Cycle

Cycles	Address Bus	Data Bus	External Operation	Internal Operation
1	?	?	Don't Care	Hold During Reset
2	? + 1	?	Don't Care	First Start State
3	0100 + SP	?	Don't Care	Second Start State
4	0100 + SP-1	?	Don't Care	Third Start State
5	0100 + SP-2	?	Don't Care	Fourth Start State
6	FFFC	Start PCL	Fetch First Vector	
7	FFFD	Start PCH	Fetch Second Vector	Hold PCL
8	PCH PCL	First OP CODE	Load First OP CODE	

The start cycle actually takes seven cycles from the time the reset line is let go to TTL plus. On the eighth cycle, the vector fetched from the memory location FFFC and FFFD is used to access the next instruction. The microprocessor is now in a normal program load sequence, the location where the vector points should be the first OP CODE which the programmer desires to perform.

The second point that should be noted is that the microprocessor actually accesses the stack three times during the start sequence in cycles 3, 4 and 5. This is because the start sequence is in effect a specialized form of interrupt with the exception that the read/write line is disabled so that no writes to stack are accomplished during any of the cycles.

9.3 PROGRAMMER CONSIDERATIONS FOR INITIALIZATION SEQUENCES

There are two major facts to remember about initialization. One, the only automatic operations of the microprocessor during reset are to turn on the interrupt disable bit and to force the program counter to the vector location specified in locations FFFC and FFFD and to load the first instruction from that location. Therefore, the first operation in any normal program will be to initialize the stack. This should be done by having previously decided what the stack value should be for initial operations and then doing a LDX immediate of this value followed by a TXS. By this simple operation, the microprocessor is ready for any interrupt or non-maskable interrupt operation which might occur during the rest of the start-up sequence.

Once this is accomplished, the two non variable operations of the machine are under control. The program counter is initialized and under programmer control and the stack is initialized and under program control. The next operations during the initialization sequences will consist of configuring and setting up the various control functions necessary to perform the I/O desired for the microprocessor.

Specific discussion for considerations regarding the start-up are covered in Section 11.

The major things which have to be considered include the current state of the I/O device and the non destructive operations that will allow the state to be changed to the active state.

The initialization programs mostly consist of loading accumulator A immediately with a bit pattern and storing it in the data control register of an I/O device.

Note: The interrupt disable is automatically set by the microprocessor during the start sequence. This is to minimize the possibility of a series of interrupts occurring during the start-up sequence because of uncontrolled external values although it is usually possible to control interrupts as part of the configuration.

The programmer should consider two effects. First, that the non maskable interrupt is not blockable by this technique since it would be possible to configure a device that was connected to a non maskable interrupt and have to service the interrupt immediately. Secondly, the mask must be cleared at the end of the start sequence unless the user has specific reason to inhibit interrupts after he has done the start-up sequence. Therefore, the next to last instruction of the start-up sequence should be CLI.

It should be noted that the start-up routine is a series of sequential operations which should occur only during power on initialization and is the first step in the programmed logic machine.

Because the execution of the routine during power on occurs very seldom in the normal operation of the machine, the coding for power on sequence should tend to minimize the use of memory space rather than speed.

The last instruction in the start-up sequence should initialize the decimal mode flag to the normal setting for the program.

The next instruction should be the beginning of the user's normal programming for his device, everything preceding that being known as "housekeeping."

9.4 RESTART

It should be noted that the basic microprocessor control philosophy allows for a single common reset line which initializes all devices. This line can be used to clear the microprocessor to a known state and to reset all peripherals to a known state; therefore, it can be used as a result of power interruption, during the power on sequence, or as an external clear by the user to re-initialize the system.

As discussed in the hardware manual, restart is often used as an aid to making sure the microprocessor has been properly interconnected and that programs have been loaded in the correct locations.

9.5 INTERRUPT CONSIDERATIONS

Up until this point, the microprocessor has to proceed under programmer control through a variety of sequences. The only way for the programmer to change the sequence of operations of the microprocessor was to change the program counter location to point at new operations. The microprocessor is in control of fetching the next instruction at the conclusion of the current instruction. The only way that external events could control the microprocessor, if it were not for interrupts, would be for the programmer to periodically interrupt or stop processing data and check to see whether or not an external event which might cause him to change his direction has occurred. The problem with this technique is that