ADD A TRAP VECTOR
FOR UNIMPLEMENTED 6502 OPCODES

BY CARL W. MOSER
3239 Linda Dr.
Winston-Salem, NC 27106

NOTE: Ideas in this article are at the present purely conceptual and have not been implemented. They are believed to be valid and are presented in this preliminary stage for feedback to aid in the ultimate implementation.

One feature the 6502, 6800, 8080, and most other microprocessors lack is hardware to "trap" unimplemented opcodes. This can aid in debugging a program in that the processor would never "hang up" as can now happen especially when executing software not fully debugged. In fact there is at least one unimplemented opcode in the 6800 that requires a power down procedure to reset the processor. Fortunately the 6502 does not have to be powered down for any of its unimplemented opcodes.

What if you could detect those unimplemented opcodes, what have you gained? Well, you've gained in that during debugging you could find out where the execution of the opcode occurred. This is very beneficial but you could also implement those opcodes in your own custom extension of the 6502 instruction set. How would you like a multiply or divide instruction, several one byte calls, auto incrementing and auto decrementing locations, block transfer, a whole group of 16 bit operations, etc. Before I go on, let's present the hardware which could detect an illegal opcode and if detected vector to high order memory for processing.

Trap Vector Circuitry

Figure 1 outlines the circuitry which will detect if the 6502 is trying to execute an illegal opcode. One characteristic of the 6502 is that it will output a SYNC pulse (pin 7) every time it reads the opcode portion of an instruction. The ROM of figure 1 has its address inputs connected to the 6502 data bus. This ROM is programmed to output a '1' every time a byte on the data bus corresponds to an illegal opcode.

Whenever the 6502 inputs an opcode, the SYNC will go to a '1', and the output from the ROM will be a '1' if the data on the bus is an unimplemented opcode. If both of these signals are logic 1, the bus transmitter is disabled, a bus driver is enabled, which forces 0's on the data bus. Well this is a break instruction (hex 00). The microprocessor will execute it and vector to FFFE for processing. This processing could obtain the address of the BRK instruction from the stack, input it, and test if it was actually a BRK instruction or an unimplemented opcode. You may be asking, why don't we get forced zeros again? The answer is that the SYNC pulse only occurs during the inputting of the opcode—not when inputting the second or third byte of the instruction or the loaded or stored data that occurs when executing the instruction.

A Possible 6502 Extended Instruction Set

What is the difference between a register and memory? Nothing except that with today's hardware technology, a few locations can economically be made faster than others. Thus execution is presently faster through a register than in ordinary memory. The result is to bottleneck data flow by channeling information through a small set of memory locations (registers). Also, why do we have a register designated as an accumulator? The answer is another hardware limitation dictated by economics.

The extended instruction set I will describe has no accumulators. In fact any memory location is equivalent to an accumulator. This instruction set has 30 autoincrementing index locations, and 32 autodecrementing index locations. The reason for specifying a designated set of index locations is to make the instruction set easier to implement.

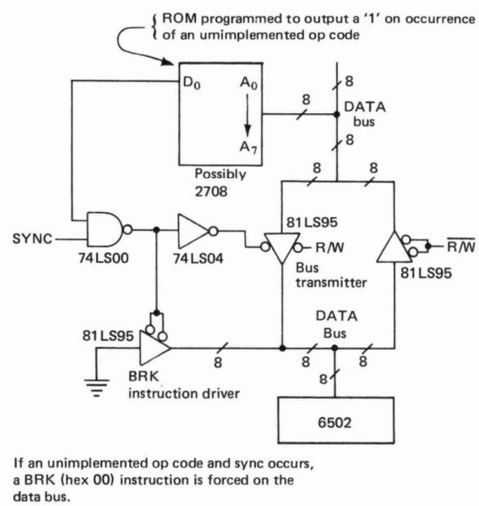

Figure 1. Trap vector detection hardware.

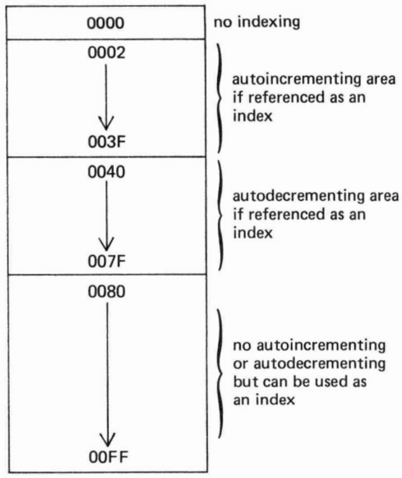

Figure 2. Memory map of index locations.

In this instruction set extension, any time a memory location from 0002 through 003F is specified as an index, autoincrementing will occur on the index contents. Also, locations 0040 through 007F will provide autodecrementing. If any other location is specified as the index, no autoincrementing or autodecrementing will occur. Functionally, this is equivalent to 30 autoincrementing and 32 autodecrementing index registers. This is illustrated in figure 2.

A possible instruction set extension is listed in table A. These instructions provide 16 bit operations and feature instructions to move a block of memory to another area (BLT), multiply (MUL and MUI), divide (DIV and DVI), and compare (CM and CMI) which sets the N, Z, and C flags plus sets the overflow flag (0) if FFFF (-1) occurs.

A typical instruction is: MOV Dlable (Id), Slabel (Is) This instruction should assemble as:

opcode
address lo } source
address hi }
Is — zero page address of 16 bit source index
address lo } destination
address hi }
Id — zero page addrs of 16 bit destination index

NO. BYTES	MNEMONIC		DESCRIPTION
6	BLT	Dlabel (#),Slabel	Block Transfer
7	MOV	Dlabel (Id), Slabel (Is)	Move - memory to memory
6	MVI	Dlabel (Id), immediate data	Move - immediate to memory
7	ADD		Add
7	SUB		Subtract
7	MUL		Multiply
7	DIV	Dlabel (Id), Slabel (Is)	Divide
7	XOR		Exclusive or
7	OR		Logical or
7	AN		Logical and
6	ADI		Add
6	SUI		Subtract
6	MUI		Multiply
6	DVI	Dlabel (Id), immediate data	Divide
6	XRI		Exclusive or
6	ORI		Logical or
6	ANI		Logical and
7	CM	Dlabel (Id), Slabel (Is)	Compare
6	CMI	Dlabel (Id), immediate data	Compare immediate
5	SHR	Dlabel (Id), #	Shift right a number of places
5	SHL	Dlabel (Id), #	Shift left a number of places

NOTE:
Dlabel = Destination address
Slabel = Source address
Id = Destination index
Is = Source index

Table A — Possible extension of the 6502 instruction set.

Continued to p. 41