KOMPUTER KORNER

TIM BARRY

WE PREVIOUSLY DISCUSSED THE DATA TRANSfer and arithmetic/logic instructions that can be performed by the 8080. This month we conclude this presentation with a discussion of the transfer of control and processor control instruction groups. (You may wish to refer to the two previous columns for an explanation of some of the features mentioned and the notations used to represent various register groups, data types, and data transfers.)

Transfer of control instructions

Any instruction that is used to transfer the execution of the program from where it is currently executing to another place in memory is considered to be a member of this group. Transfers of control can be considered to be either *returning* or *non-returning:* A non-returning transfer, once executed, has no way of knowing the memory address that it was transferred from because it does not save the address of the next instruction to be executed prior to executing the transfer. A returning transfer, on the other hand, saves the address of the next instruction where the program is executing before the transfer is executed. This saved address can be used later in the program to return control to the place in the program where the transfer occurred. A non-returning transfer is usually called a *program jump* or a *program branch.* A returning transfer is usually called a *subroutine call.* The transfer of control back to a returning transfer is usually called a *subroutine return.*

Transfers of control—whether jumps, calls or returns—can be considered to be either *conditional* or *unconditional.* Unconditional transfers are executed whenever they are encountered, regardless of the state of the processor flags. Conditional transfers are executed based upon the state of internal processor flags. If the specified condition is met, the transfer takes place. If the condition is not met, the instruction is ignored and execution continues on (i.e, Jump if zero, call if no carry, etc.). Different processors have different flags, and hence different conditional instructions. However, all computers will have some conditional instructions, because this is the feature that allows the processor to respond to the results of tests and operations performed by the arithmetic/logic unit.

The 8080 offers both returned and unreturned transfers of control. The unreturned transfers are called *jump* instructions, the returned transfers are called *call* instructions, and the returns to the addresses saved by returned transfers are called *return* instructions. These transfers are all available in both conditional and unconditional forms. In addition, there is one indirect unconditional transfer instruction (PCHL) and a group of special truncated unconditional subroutine calls (RST).

The returning transfer-of-control instructions executed by the 8080 save the return address in the area of memory addressed by the stack pointer. Before the transfer is executed, the program-counter address where execution is to return is pushed into the top two locations of the stack. When a return is executed, the contents of the top two locations of the stack are popped into the program counter, thereby transferring control to that location.

This is a very convenient way to handle subroutine return addresses, but it requires us to pay careful attention to the stack operations performed by the rest of the program. We have other program operations that can *push* and *pop* data using the stack. If these operations don't match properly, the stack pointer may not be pointing to the correct return address when a subroutine return is executed. When this happens, something other than the intended return address is placed into the program counter. This results in the execution of a "return to random address" instruction, usually followed immediately by a dramatic change in program execution. To avoid this problem, you must always make sure that data is taken out of the stack in the reverse order that it is put into it. It also means that the last subroutine called

continued on page 24

KOMPUTER KORNER
continued from page 22

must be the first one returned. The number of unreturned subroutine calls that have been executed prior to the first return being executed is called *subroutine nesting*. When you hear a programmer say that his program is "nested four deep" at some point, he means that four subroutines have been called and none have executed returns. Keeping track of subroutine nesting and data transfers with the stack is called *balancing the stack,* and it is essential to correct program operation.

In addition to balancing the stack, you must be sure that no program operations inadvertently modify the contents of the stack. This can happen when the stack nesting causes it to overlap with other assigned program storage. It can also happen when some routine that transfers data into memory gets out of control and overruns into the stack area. To avoid these problems, it is best to locate the stack in an area of memory that is not used by any other portion of the program. If this is not possible, you must compute stack usage based on nesting and stack use and allocate an area among the rest of your program storage for stack use. The stack is a very powerful feature, but it must be used with care to avoid problems.

Unconditional transfers of control

This group of 8080 instructions is executed whenever they are encountered during program execution.

JMP Addr (3)
Operation Performed: $Addr_{0\text{-}15} \rightarrow PC_{0\text{-}15}$
The 16-bit address included with the instruction is loaded into the program counter.

CALL Addr (3)
Operation Performed: $(PC_{8\text{-}15}) \rightarrow [SP\text{-}1]$
$(PC_{0\text{-}7}) \rightarrow [SP\text{-}2]$
$Addr_{0\text{-}15} \rightarrow PC_{0\text{-}15}$
$(SP) + 2 \rightarrow SP$
The address of the next instruction to be executed after the subroutine returns is pushed onto the stack. Then the 16-bit address included with the instruction is loaded into the program counter.

RET (1)
Operation Performed: $[SP] \rightarrow PC_{0\text{-}7}$
$[SP+1] \rightarrow PC_{8\text{-}15}$
$(SP) + 2 \rightarrow SP$
The contents of memory addressed by the stack pointer are popped into the program counter, transferring control to that location.

PCHL (1)
Operation Performed: $(HL) \rightarrow PC$
The contents of the HL register are transferred into the program counter. (This instruction was also included as a data-transfer instruction.)

RST L (1)
where L is an integer in the range 0-7.
Operation Performed: $(PC_{8\text{-}15}) \rightarrow [SP\text{-}1]$
$(PC_{0\text{-}7}) \rightarrow [SP\text{-}2]$
$8{*}L \rightarrow PC_{0\text{-}15}$
$(SP) + 2 \rightarrow SP$
The RST instruction is actually a call instruction which is hardware defined to call a fixed block of memory locations. Upon execution, a RST instruction behaves exactly as a subroutine call to the location 8*L. Thus the instruction RST 1 would be equivalent to CALL 8. A transfer initiated by a RST instruction is returned using a return instruction, exactly like any other subroutine call. The RST instructions are provided so that an external device can interrupt the 8080 and provide a single-byte instruction to transfer to a device-service routine.

Conditional transfers of control

The 8080 provides a wide selection of conditional transfer instructions. The state of the four ALU flags (carry, zero, sign, and parity) can be used to determine the execution of jumps, calls or returns. The condition code symbols used to represent these various flag conditions are as follows:

Symbol	Condition Tested
Z	Transfer if zero
NZ	Transfer if not zero
C	Transfer if carry
NC	Transfer if no carry
P	Transfer if sign flag is plus
M	Transfer if sign flag is minus
PE	Transfer if parity flag is even
PO	Transfer if parity flag is odd

A conditional transfer is formed by adding the appropriate character prefix (J for jump, C for call, or R for return) to the condition symbol. Jump if parity even would thus be
continued on page 30

KOMPUTER KORNER

continued from page 24

JPE, call if sign is minus would be CM, and so on.

J(COND) Addr (3)

Operation Performed: If (COND) is true, then
$Addr_{0-15} \rightarrow PC_{0-15}$,
otherwise
$(PC) + 3 \rightarrow PC$

If the condition code specified is true, transfer to the location specified by the 16-bit address included with the instruction. Otherwise, continue execution with the next instruction.

C(COND) Addr (3)

Operation Performed: If (COND) is true, then
$(PC_{8-15}) \rightarrow [SP-1]$
$(PC_{0-7}) \rightarrow [SP-2]$
$Addr_{0-15} \rightarrow PC_{0-15}$
$(SP) - 2 \rightarrow SP$,
otherwise
$(PC) + 3 \rightarrow PC$

If the condition code specified is true, the address of the next instruction to be executed upon return is pushed onto the stack and control is transferred to the location specified by the 16-bit address included with the instruction. Otherwise, continue execution with the next instruction.

R(COND) (1)

Operation Performed: If (COND) is true, then $[SP] \rightarrow PC_{0-7}$
$[SP+1] \rightarrow PC_{8-15}$
$(SP) + 2 \rightarrow SP$,
otherwise
$(PC) + 1 \rightarrow PC$

If the condition code specified is true, the top elements of memory as addressed by the stack pointer are popped into the program counter, transferring program execution to that location. Otherwise, continue execution with the next instruction.

Processor control instructions

Computers provide a small group of instructions that can be used to control the operation of the actual CPU hardware. These instructions are concerned with enabling and disabling the computer's interrupt facility, setting up I/O device priority, halting and resetting the processor and so on. These instructions are not used often, but it is important to understand their operation for those times when you will need them and want to use them.

The 8080 has four processor-control instructions. These instructions include an instruction that does nothing (often needed, believe it or not), a computer-halt instruction, and instructions to enable and disable the processor's response to interrupts.

NOP (1)

The NOP operation does nothing. It is present in the instruction set to allow you to delete operations when debugging, leave space for program additions and provide a fixed execution time interval for use in program timing loops.

HLT (1)