Additional Programming techniques

Dr. M Krishnam Raju
Assistant Professor
Electronics & Communication Engineering

Content

1. Counter
2. Time delay
3. Stack and Subroutines
4. Restart, Call and Return,
5. Instruction
6. Code conversion

Counter

- A counter refers to a register or a set of registers used for counting events or operations in a specific sequence.

Types of Counters in 8085

- **Program-Controlled Counter:**
  - Use the general-purpose registers (B, C, D, E, H, and L) to implement a counter.
  - A simple counter program typically involves initializing the register with a starting value, and then the value is decremented or incremented in each cycle (often within a loop) until a specified condition is met.
- **External Counter:**
  - The 8085 microprocessor can interface with external counters via input/output ports. In such cases, the counter would be an external device or an additional IC that is connected to the 8085.
- **Timer/Counter in 8085:**
  - The 8085 does not have an internal timer or dedicated counter. However, a counter or timer can be simulated using the delay loops or by interfacing with external devices like the 8253 Programmable Interval Timer (PIT).

Counter

How to Implement a Simple Counter in 8085?

A simple program to implement a counter using the 8085 microprocessor could involve the following steps:

- Initialize the counter register. For example, initialize register B with a value, say 0x05 (5 in decimal).
- Create a loop. In each loop, decrement or increment the counter until it reaches zero.
- Exit the loop when the counter reaches zero.

```
MVI B, 05H   ; Load the value 5 into register B (counter)
START:
   NOP       ; No Operation (could be replaced by an operation you want to count)
   DCX B     ; Decrement register B (counter)
   MOV A, B  ; Move the counter value to register A
   CPI 00H   ; Compare the value of A with 0
   JNZ START ; Jump to START if A is not zero (counter has not reached 0)
   HLT       ; Halt the program when the counter reaches 0
```

Counter

Explanation:

- The program starts by loading the value 5 into register B.
- In the loop (START label), it decrements the value of register B using DCX B.
- After each decrement, the value of register B is moved to register A, and a comparison (CPI 00H) is done to check if the counter has reached zero.
- If the counter hasn't reached zero, the program jumps back to the START label, and the process repeats.
- When the counter reaches zero, the program halts.

External Counter Interface (e.g., 8253 Timer/Counter):

To implement more sophisticated timing or counting operations (like generating precise time delays or counting external events), you would typically use an external IC like the 8253 Programmable Interval Timer. The 8085 can communicate with such devices to control or read the state of an external counter.

Time Delay

Time delay refers to the intentional pause or delay inserted into the program execution, typically for timing control or to create a specific interval for external devices.

There are several ways to create a time delay in the 8085:

Using a Loop (Software Delay)

A simple method for generating a delay is by creating a loop that runs for a certain number of iterations. The execution time of this loop is determined by the number of iterations and the clock cycle time of the 8085. Each iteration of the loop takes a fixed number of clock cycles.

```
MVI B, 0xFF      ; Load 255 into register B (this will be the counter)
DELAY:
DCR B            ; Decrement register B
JNZ DELAY        ; Jump back to DELAY if B is not zero
```

Time Delay

In this example:

- The loop decrements register B and jumps back if the register is not zero. This will repeat 255 times, creating a delay.
- The actual time delay depends on the number of clock cycles required for each instruction. Since the 8085 operates with a 3-clock cycle for most instructions, the delay can be calculated based on the number of iterations.

Using an External Timer

Another method is to use an external timing circuit, such as a 555 timer, or a counter connected to the 8085. The microprocessor can monitor a status register or use interrupts to detect when the timer reaches a specific value.

Using a Timer (Hardware Delay)

The 8085 has a built-in 16-bit timer (T2) that can be programmed to generate specific time delays. This timer is usually used for more accurate and longer delays.

Time Delay

Delay using a fixed clock

- The 8085 runs at a specific clock speed (typically 3 or 5 MHz). By knowing the number of clock cycles per instruction and controlling the number of instructions, you can compute a delay.
- Each instruction takes a certain number of clock cycles:
  - MVI takes 7 cycles.
  - DCR takes 5 cycles.
  - JNZ takes 10 cycles.
- To calculate a delay, you multiply the number of cycles by the number of instructions to get the total number of clock cycles, which can then be converted into time.
- Calculation for Time Delay: For an 8085 microprocessor running at 3 MHz:
  - 1 clock cycle = 1/3,000,000 seconds = 0.333 microseconds.
  - If you want a delay of 1 millisecond (1000 microseconds), the number of clock cycles required = 1000 / 0.333 ≈ 3000 cycles.
  - If your loop takes 7 cycles for MVI, 5 cycles for DCR, and 10 cycles for JNZ, you can estimate how many iterations of the loop are needed for the delay.

Stack and Subroutines

Stack in 8085 Microprocessor:

The stack is a portion of memory used for temporary storage. It operates on a "Last In, First Out" (LIFO) principle, meaning the last data pushed onto the stack is the first data to be popped off.

- **Stack Pointer (SP):** The stack is managed using a special 16-bit register called the Stack Pointer. The Stack Pointer holds the address of the top of the stack.
- **Push Operation:** When data is pushed onto the stack, it is stored in memory at the address pointed to by the Stack Pointer, and then the Stack Pointer is decremented.
- **Pop Operation:** When data is popped from the stack, the Stack Pointer is incremented, and the data at the new Stack Pointer location is accessed.

Stack and Subroutines

Subroutines:

A subroutine is a block of code designed to perform a specific task, which can be called from various points in the program. Subroutines help in reducing code repetition and making programs more modular and maintainable.

- **CALL Instruction:** This instruction is used to call a subroutine. It saves the current Program Counter (PC) onto the stack so that the program can return to the correct location after the subroutine finishes execution. The address is the starting location of the subroutine. When the CALL instruction is executed, the address of the next instruction (the instruction after the CALL) is pushed onto the stack, and the control is transferred to the subroutine.
- **RET Instruction:** After a subroutine has completed its task, the RET (Return) instruction is used to return control to the main program. It pops the return address from the stack and loads it into the Program Counter, causing the program to continue execution from that point.

JMP, CALL, and RET

Instruction	Mnemonic	Description
Jump Instructions		
JUMP (Always)	JMP	Unconditional jump to a specified address.
JUMP (Conditional)	JC	Jump if carry flag is set.
JUMP (Conditional)	JNC	Jump if carry flag is not set.
JUMP (Conditional)	JZ	Jump if zero flag is set.
JUMP (Conditional)	JNZ	Jump if zero flag is not set.
JUMP (Conditional)	JP	Jump if sign flag is reset (positive).
JUMP (Conditional)	JM	Jump if sign flag is set (negative).
Call Instructions		
CALL (Unconditional)	CALL	Call a subroutine (unconditional jump to subroutine address).
CALL (Conditional)	CC	Call a subroutine if carry flag is set.
CALL (Conditional)	CNC	Call a subroutine if carry flag is not set.
CALL (Conditional)	CZ	Call a subroutine if zero flag is set.
CALL (Conditional)	CNZ	Call a subroutine if zero flag is not set.
CALL (Conditional)	CP	Call a subroutine if sign flag is reset (positive).
CALL (Conditional)	CM	Call a subroutine if sign flag is set (negative).
Return Instructions		
RETURN (Unconditional)	RET	Return from subroutine (unconditional).
RETURN (Conditional)	RC	Return from subroutine if carry flag is set.
RETURN (Conditional)	RNC	Return from subroutine if carry flag is not set.
RETURN (Conditional)	RZ	Return from subroutine if zero flag is set.
RETURN (Conditional)	RNZ	Return from subroutine if zero flag is not set.
RETURN (Conditional)	RP	Return from subroutine if sign flag is reset (positive).
RETURN (Conditional)	RM	Return from subroutine if sign flag is set (negative).

Jump (JMP) Instruction: Used to transfer the control of the program unconditionally to a specified memory location.

Unconditional Jump: Directly jumps to the specified address.

Conditional Jump: The program jumps to the specified address only if a certain condition is met (e.g., Z flag set, sign flag set, etc.).

JMP, CALL, and RET

Call Instruction:

- The CALL instruction is used to call a subroutine (a set of instructions that are executed separately from the main program).
- The program control is transferred to the specified memory location (subroutine address).
- When the subroutine is completed, the control returns to the next instruction following the CALL instruction using a RET (Return) instruction.

Return (RET) Instruction:

- The RET instruction is used to return from a subroutine to the main program (i.e., the address after the CALL instruction).
- The return address is popped from the stack, and program execution continues from there.

Code conversion

Binary to BCD Conversion

```
Convert 8-bit binary number in register A to BCD
MVI H, 00H       ; Clear register H (BCD higher byte)
MVI L, 00H       ; Clear register L (BCD lower byte)
MVI B, 08        ; Set B to 8 (loop counter for 8 bits)
START:
   MOV C, A      ; Copy A to C
   RLC           ; Rotate left through carry
   JC ADD_10     ; If carry, add 10 to BCD
   DCR L         ; Decrement lower BCD byte
   JMP NEXT
ADD_10:
   ADD M         ; Add 10 (decimal) to L register
NEXT:
   DCR B         ; Decrement counter (loop)
   JNZ START     ; Repeat loop till 8 bits are processed
Now registers H and L contain the BCD equivalent of the binary number in A
```

Code conversion

Decimal to Binary Conversion

```
Convert a decimal number in register A to binary and store in register B
MVI B, 00H      ; Clear register B (for storing the binary result)
MVI C, 08       ; Set the loop counter to 8
LOOP:
   RLC          ; Rotate A left through carry (shift left, MSB to carry)
   JC SET_BIT   ; If carry, set the corresponding bit in B
   MOV A, B     ; No carry, move current B value back to A
   JMP SKIP
SET_BIT:
   ORA B        ; Set the corresponding bit in B register
SKIP:
   DCR C        ; Decrement counter
   JNZ LOOP     ; Loop until 8 bits are processed
Now register B contains the binary equivalent of the decimal number in A
```

Code conversion

Hexadecimal to Binary Conversion

Convert a hexadecimal digit in register A to its binary equivalent in register B

```
MVI B, 00H      ; Clear register B (binary result)
MVI C, 04       ; 4-bit conversion (1 hex digit has 4 bits)
LOOP:
   RLC          ; Rotate A left through carry
   JC SET_BIT   ; If carry, set the corresponding bit in B
   MOV A, B     ; No carry, move current B value back to A
   JMP SKIP
SET_BIT:
   ORA B        ; Set the corresponding bit in B register
SKIP:
   DCR C        ; Decrement counter
   JNZ LOOP     ; Loop until 4 bits are processed
```

Now register B contains the binary equivalent of the hex digit in A

Code conversion

BCD to Binary Conversion

Convert a 2-digit BCD in HL register pair to binary in A register

Assume BCD number is stored in H and L (higher byte = tens, lower byte = ones)

```
MVI A, 00H       ; Clear A to store the binary result
MOV B, H         ; Copy the higher BCD digit (tens place) to B
MVI C, 10        ; Set multiplier (10 for tens place)

MUL_TENS:
   CALL MULTIPLY ; Multiply BCD tens by 10
   MOV A, L      ; Move result to A (now A has the binary equivalent)
```

Now repeat for the lower digit in L (ones place)

Parul® University

NAAC GRADE A++

https://paruluniversity.ac.in/