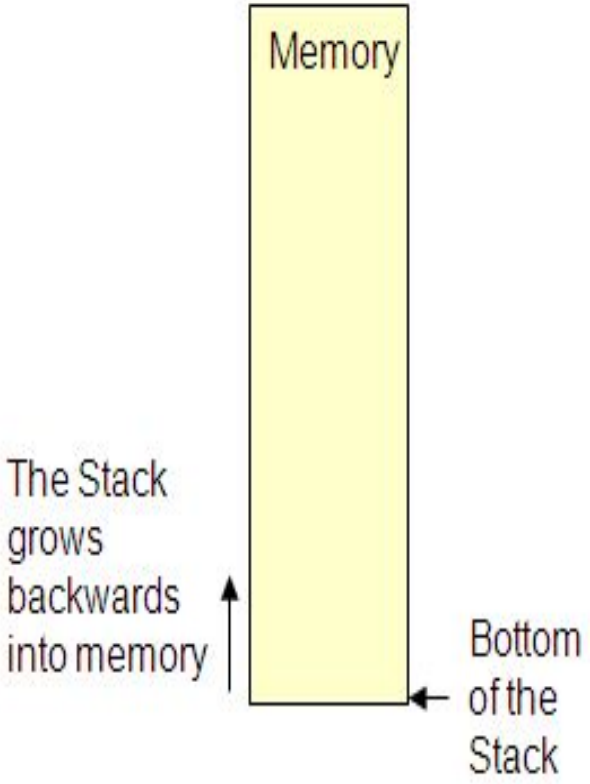
Stacks & Subroutines Slides-7

Dr. Ritika

Department of Computer Application

The Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO (Last In First Out.) structure.
- The stack normally grows backwards into memory.
- In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.



The Stack

- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.

- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.

LXI SP, FFFFH

- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

Saving Information on the Stack

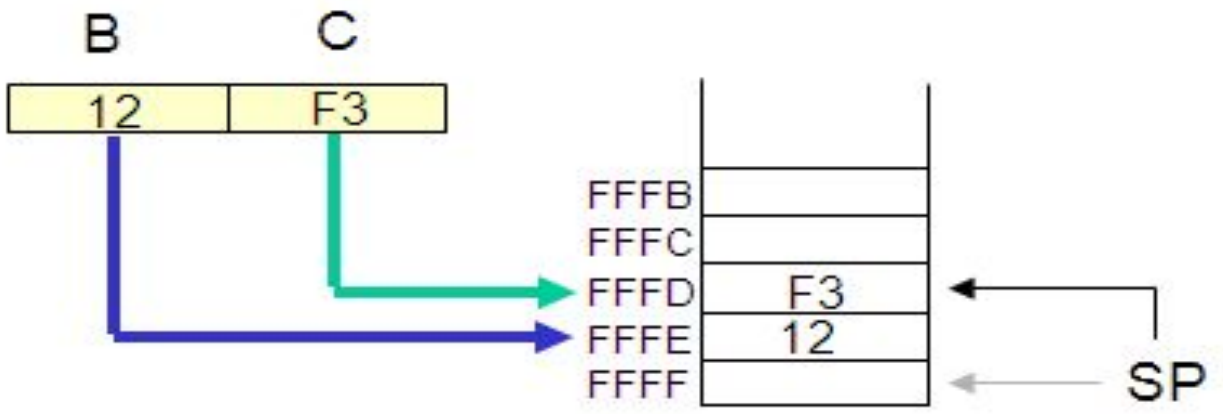
- Information is saved on the stack by PUSHing it on.
- It is retrieved from the stack by POPing it off.

- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.

- Both PUSH and POP work with register pairs ONLY.

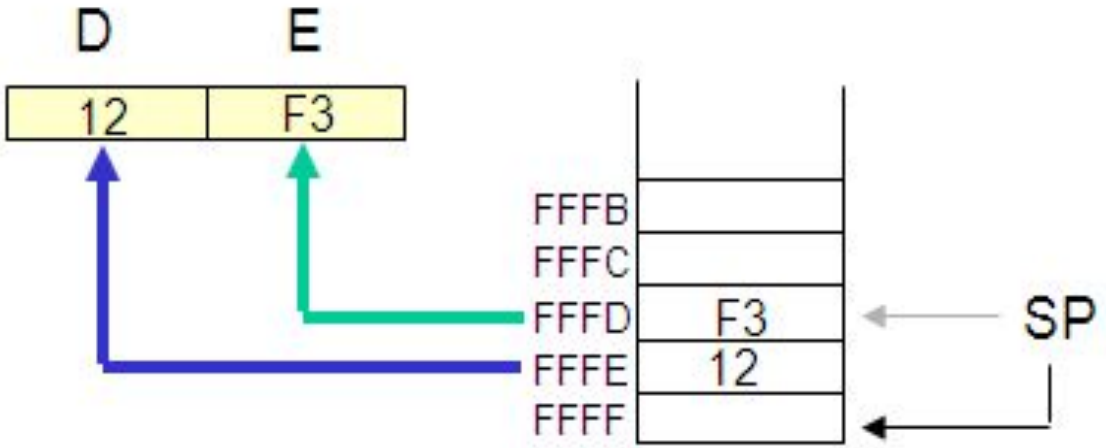
The PUSH Instruction

- PUSH B/D/H/PSW
  - Decrement SP
  - Copy the contents of register B to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of register C to the memory location pointed to by SP



The POP Instruction

- POP B/D/H/PSW
  - Copy the contents of the memory location pointed to by the SP to register E
  - Increment SP
  - Copy the contents of the memory location pointed to by the SP to register D
  - Increment SP



Operation of the Stack

- During pushing, the stack operates in a “decrement then store” style.
  - The stack pointer is decremented first, then the information is placed on the stack.

- During poping, the stack operates in a “use then increment” style.
  - The information is retrieved from the top of the the stack and then the pointer is incremented.

- The SP pointer always points to “the top of the stack”.

LIFO

- The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

```
PUSH B
PUSH D
...
POP D
POP B
```

- **Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE.**

The PSW Register Pair

- The 8085 recognizes one additional register pair called the PSW (Program Status Word).

- This register pair is made up of the Accumulator and the Flags registers.

- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.

- The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

Cautions with PUSH and POP

- PUSH and POP should be used in opposite order.

- There has to be as many POP's as there are PUSH's.
  - If not, the RET statement will pick up the wrong information from the top of the stack and the program will fail.

- It is not advisable to place PUSH or POP inside a loop.

Program to Reset and display Flags

- Clear all Flags.
- Load 00H in the accumulator, and demonstrate that the zero flag is not affected by data transfer instruction.
- Logically OR the accumulator with itself to set the Zero flag, and display the flag at PORT1 or store all flags on the stack.

Program to Reset and display Flags

- XX00 LXI SP, XX99H Initialize the stack
  - 03 MVI L, 00H Clear L
  - 05 PUSH H Place (L) on stack
  - 06 POP PSW Clear Flags
  - 07 MVI A, 00H Load 00H
  - 09 PUSH PSW Save Flags on stack
  - 0A POP H Retrieve flags in L
  - 0B MOV A, L
  - 0C OUT PORT0 Display Flags (00H)
  - 0E MVI A, 00H Load 00H Again

Program to Reset and display Flags

- XX10 ORA A Set Flags and reset CY, AC
  - 11 PUSH PSW Save Flags on Stack
  - 12 POP H Retrieve Flags in L
  - 13 MOV A, L
  - 14 ANI 40H Mask all Flags except Z
  - 16 OUT PORT1 Displays 40H
  - 18 HLT End of Program

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Subroutines

- A subroutine is a group of instructions that will be used repeatedly in different locations of the program.

- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

- In Assembly language, a subroutine can exist anywhere in the code.
- However, it is customary to place subroutines separately from the main program.

Subroutines

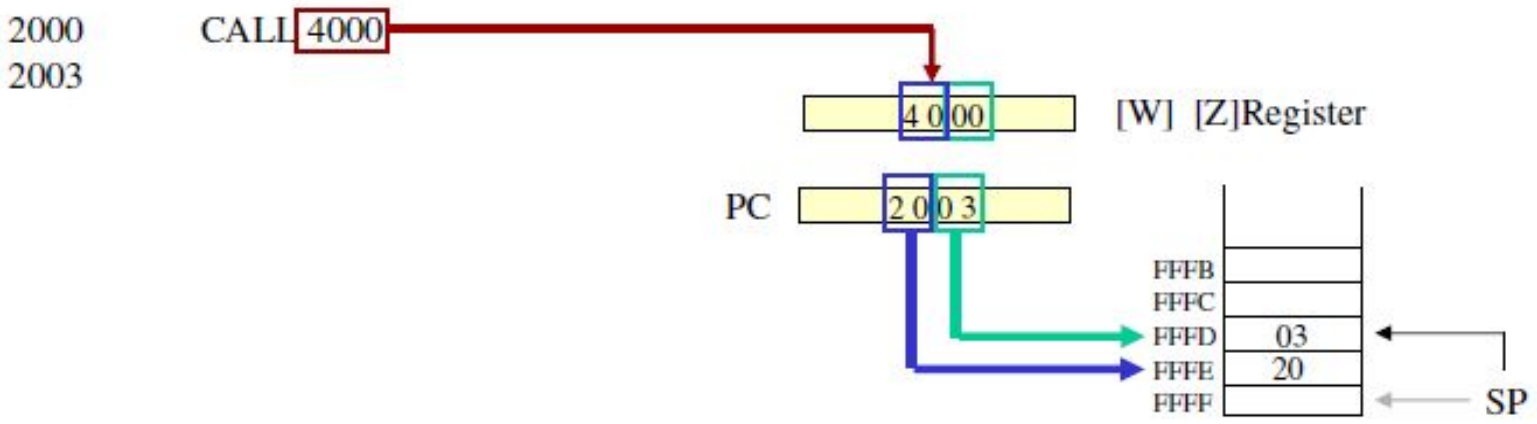
- The 8085 has two instructions for dealing with subroutines.

- The **CALL** instruction is used to redirect program execution to the subroutine.

- The **RTE** instruction is used to return the execution to the calling routine.

The CALL Instruction

- **CALL 4000H**
  - ○3-byte instruction.
  - ○Push the address of the instruction immediately following the CALL onto the stack and decrement the stack pointer register by two.
  - ○Load the program counter with the 16-bit address supplied with the CALL instruction.
  - ○Jump Unconditionally to memory location.

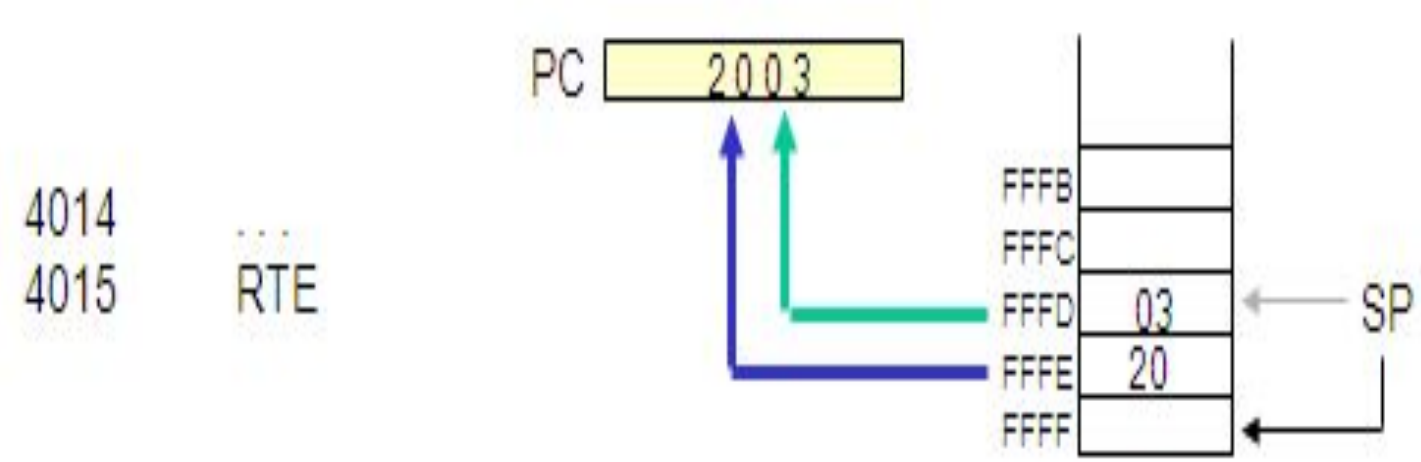


The CALL Instruction

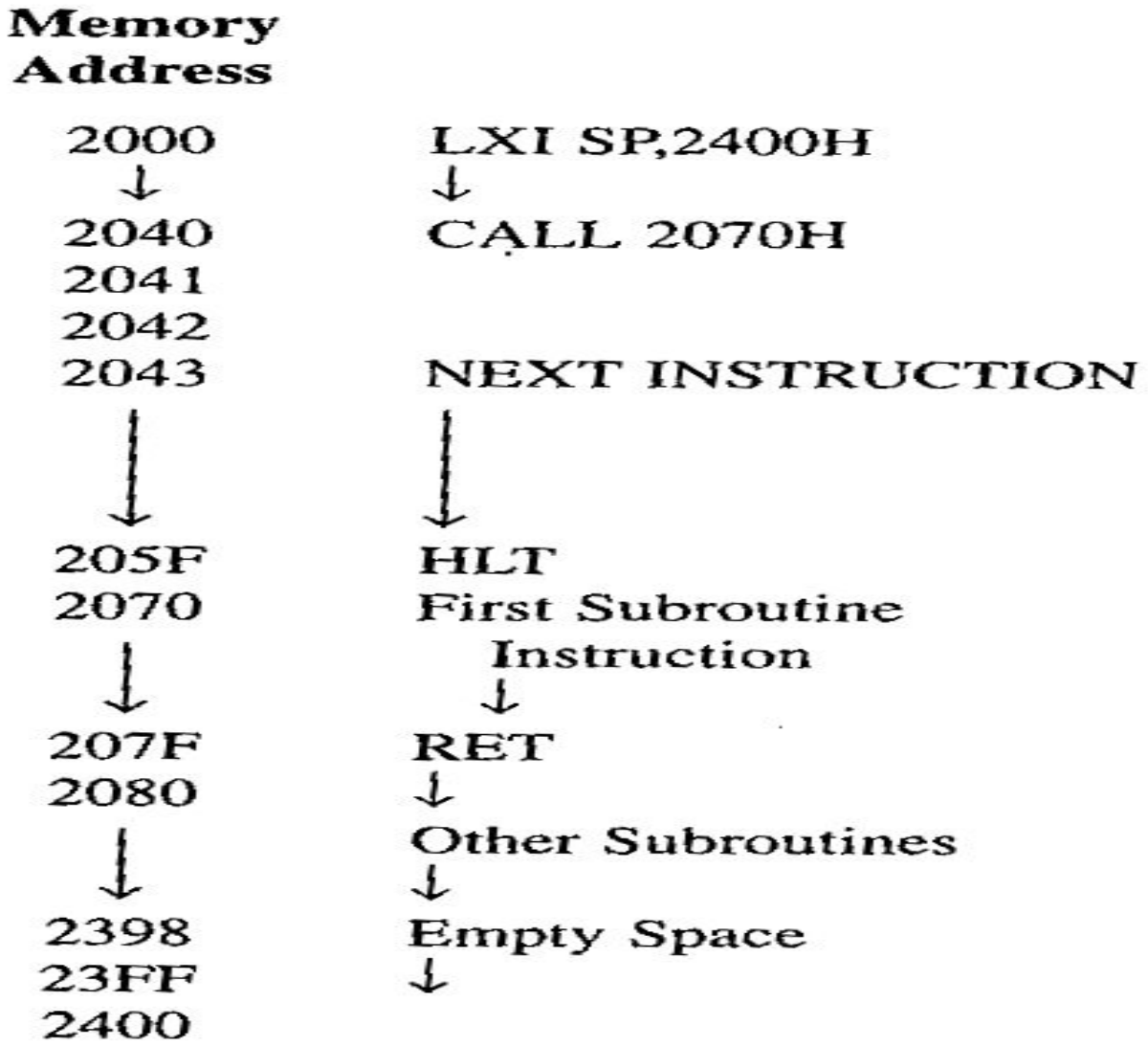
- MP Reads the subroutine address from the next two memory location and stores the higher order 8bit of the address in the W register and stores the lower order 8bit of the address in the Z register
- Push the address of the instruction immediately following the CALL onto the stack [Return address]
- Loads the program counter with the 16-bit address supplied with the CALL instruction from WZ register.

The RTE Instruction

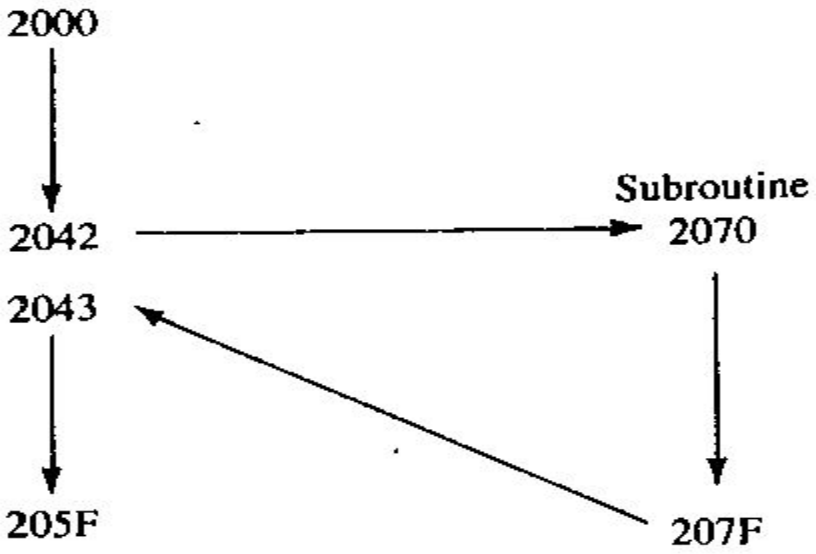
- **RTE**
  - ○1-byte instruction
  - ○Retrieve the return address from the top of the stack and increments stack pointer register by two.
  - ○Load the program counter with the return address.
  - ○Unconditionally returns from a subroutine.



Illustrates the exchange of information between stack and Program Counter



Program Execution



Memory Address	Machine Code	Mnemonics	Comments
2040	CD	CALL 2070H	;Call subroutine located at the memory
2041	70		; location 2070H
2042	20		
2043	NEXT	INSTRUCTION	

CALL Execution

- **Instruction requires five machine cycles and eighteen T-states:** Call instruction is fetched, 16-bit address is read during M2 and M3 and stored temporarily in W/Z registers. In next two cycles content of program counter are stored on the stack (address from where microprocessor continue it execution of pr

Instruction: CALL 2070H

Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand	→ 70
M_3 Memory Read	23FF	2042	20 43	20 Operand	→ 20
M_4 Memory Write	23FE (SP–2)	23FF	20 43	20 (PCH)	
M_5 Memory Write	23FE	23FE	20 43	43 (PCL)	(20) (70)
M_1 Opcode Fetch of Next Instruction		20 70 → (W)(Z)	2071		(2070) (W)(Z)

Memory Address	Code (H)
2040	CD
2041	70
2042	20

Data Transfer During the Execution of the CALL Instruction

RET Execution

- **Program execution sequence is transferred to the memory location 2043H location.M1 is normal fetch cycle during M2 contents of stack pointer are placed on address bus so 43H data is fetched and stored on Z register and SP is upgraded. Similarly for M3. Program sequence is transfered to2043H by placing contents of**

Memory Address	Code (H)
207F	C9

Contents of Stack Memory	
23FE	43
23FF	20

Machine Cycles	Stack Pointer (23FE)	Address Bus (AB)	Program Counter	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FE	207F	2080	C9 Opcode	
M_2 Memory Read	23FF	23FE		43 (Stack)	43
M_3 Memory Read	2400	23FF		20 (Stack−1)	20
M_1 Opcode Fetch of Next Instruction		2043 (W) (Z)	2044		2043 (W) (Z)

Data Transfer During the Execution of the RET Instruction

Passing Data to a Subroutine

- In Assembly Language data is passed to a subroutine through registers.
  - The data is stored in one of the registers by the calling program and the subroutine uses the value from the register.

- The other possibility is to use agreed upon memory locations.
  - The calling program stores the data in the memory location and the subroutine retrieves the data from the location and uses it.

RESTART, CONDITIONAL CALL & RETURN INSTRUCTIONS

In addition to the unconditional CALL and RET instructions, the 8085 instruction set includes eight Restart instructions and eight conditional Call and Return instructions.

RST Instruction

- RST instructions are 1-byte Call instructions that transfer the program execution to a specific location on page 00H.
- Executed the same way as Call instructions, the 8085 stores the contents of the program counter (the address of the next instruction) on the top of the stack and transfers the program to the Restart location.
- These instructions are generally used in conjunction with the interrupt.

RST 0 Call 0000H	RST 4 Call 0020H
RST 1 Call 0008H	RST 5 Call 0028H
RST 2 Call 0010H	RST 6 Call 0030H
RST 3 Call 0018H	RST 7 Call 0038H

RESTART, CONDITIONAL CALL & RETURN INSTRUCTIONS

The conditional Call and Return instructions are based on four data conditions(flags): Carry, Zero, Sign, and Parity.

In case of a conditional Call the program is transferred to the subroutine if condition is met

In case of a conditional Return instruction, the sequence returns to the main program if the condition is met

Conditional C

CC	Call subroutine if Carry flag is set (CY = 1)
CNC	Call subroutine if Carry flag is reset (CY = 0)
CZ	Call subroutine if Zero flag is set (Z = 1)
CNZ	Call subroutine if Zero flag is reset (Z = 0)
CM	Call subroutine if Sign flag is set (S = 1, negative number)
CP	Call subroutine if Sign flag is reset (S = 0, positive number)

RESTART, CONDITIONAL CALL & RETURN INSTRUCTIONS

CPE	Call subroutine if Parity flag is set (P = 1, even parity)
CPO	Call subroutine if Parity flag is reset (P = 0, odd parity)

Conditional RETURN

RC	Return if Carry flag is set (CY = 1)
RNC	Return if Carry flag is reset (CY = 0)
RZ	Return if Zero flag is set (Z = 1)
RNZ	Return if Zero flag is reset (Z = 0)
RM	Return if Sign flag is set (S = 1, negative number)
RP	Return if Sign flag is reset (S = 0, positive number)
RPE	Return if Parity flag is set (P = 1, even parity)
RPO	Return if Parity flag is reset (P = 0, odd parity)

A Proper Subroutine

- **According to Software Engineering practices, a proper subroutine:**
  - **Is only entered with a CALL and exited with an RTE**
  - **Has a single entry point**
    - **Do not use a CALL statement to jump into different points of the same subroutine.**

Writing Subroutines

Write a Program that will display FF and 11 repeatedly on the seven segment display. Write a 'delay' subroutine and Call it as necessary.

C000: LXI SP, FFFF

C003: MVI A, FF

C005: OUT 00

C007: CALL C014

C00A: MVI A, 11

C00C: OUT 00

C00E: CALL 1420

C011: JMP C003

Writing Subroutines

DELAY: C014: MVIB, FF

C016: MVIC, FF

C018: DCR C

C019: JNZ C018

C01C: DCR B

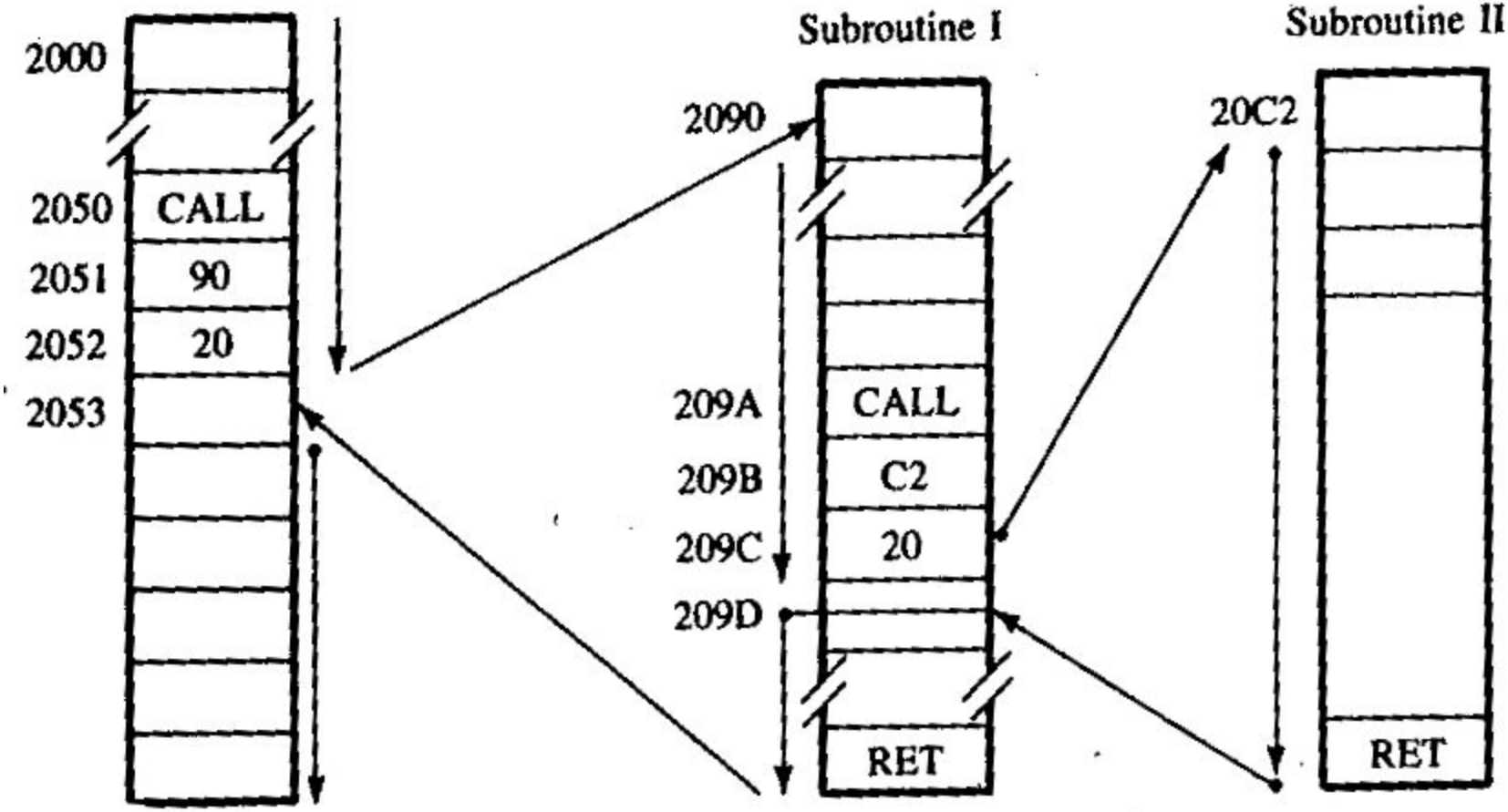
C01D: JNZ C016

C020: RET

Nesting Subroutines

The programming technique of a subroutine calling another subroutine

This process is limited only by the number of available stack locations.

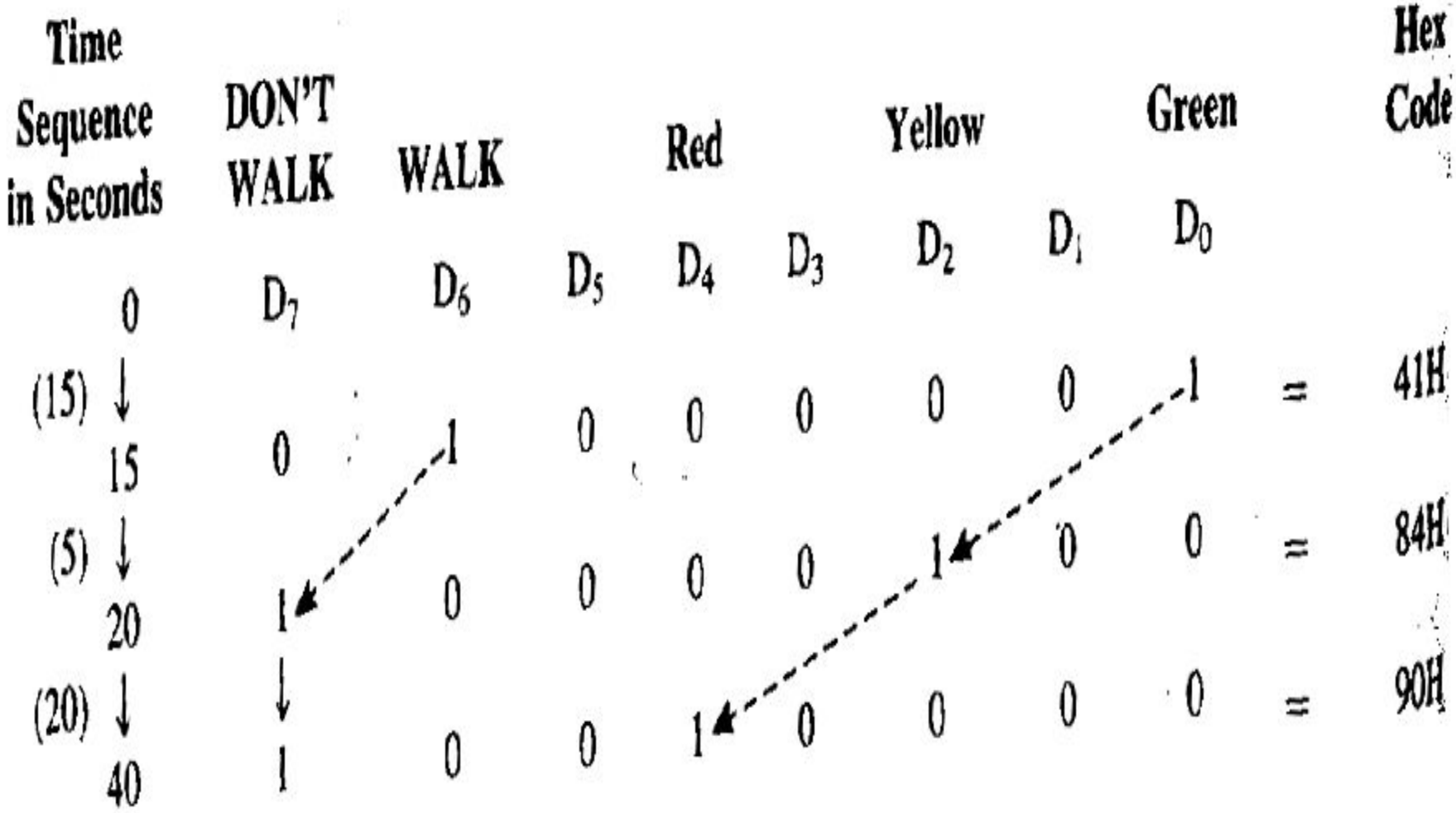


Problem Statement

Write a program to provide the given on/off time to three traffic lights (Green, Yellow, and Red) and two pedestrian signs (WALK and DON'T WALK). The signal lights and signs are turned on/off by the data bits of an output port as shown below:

Lights	Data Bits	On Time
1. Green	D_0	15 seconds
2. Yellow	D_2	5 seconds
3. Red	D_4	20 seconds
4. WALK	D_6	15 seconds
5. DON'T WALK	D_7	25 seconds

Problem Statement



Problem Statement

	Mnemonics	Comments
	LXI SP,XX99	;Initialize stack pointer at location XX99H
		;High-order address (page) of user memory
START:	MVI A,41H	;Load accumulator with the bit pattern for
		; Green light and WALK sign
	OUT PORT#	;Turn on Green light and WALK sign

Problem Statement

```
MVI B,0FH       ;Use B as a counter to count 15seconds.
                ;   B is decremented in the subroutine
CALL DELAY      ;Call delay subroutine located at XX50H

                ;High-order address (page) of user memory
MVI A,84H       ;Load accumulator with the bit pattern for
                ;   Yellow light and DON'T WALK
OUT PORT#       ;Turn on Yellow light and DON'T WALK
                ;   and turn off Green light and WALK
MVI B,05        ;Set up 5-second delay counter
CALL DELAY

                ;High-order address of user memory
MVI A,90H       ;Load accumulator with the bit pattern for
                ;   Red light and DON'T WALK
OUT PORT#       ;Turn on Red light, keep DON'T WALK on,
                ;   and turn off Yellow light
MVI B,14H       ;Set up the counter for 20-second delay
CALL DELAY

JMP START       ;Go back to location START to repeat the
                ;   sequence
```

Problem Statement

```
DELAY:   PUSH D          ;Save contents of DE and accumulator
         PUSH PSW
SECOND:  LXI D,COUNT     ;Load register pair DE with a count for
                         ;    1-second delay

Loop:    DCX D           ;Decrement register pair DE
         MOV A,D
         ORA E           ;OR (D) and (E) to set Zero flag
         JNZ LOOP        ;Jump to Loop if delay count is not equal to 0

         DCR B           ;End of 1 second delay; decrement the counter
         JNZ SECOND      ;Is this the end of time needed? If not, go
                         ;    back to repeat 1-second delay
                         ;High-order memory address of user memory
         POP PSW         ;Retrieve contents of saved registers
         POP D
         RET             ;Return to main program
```