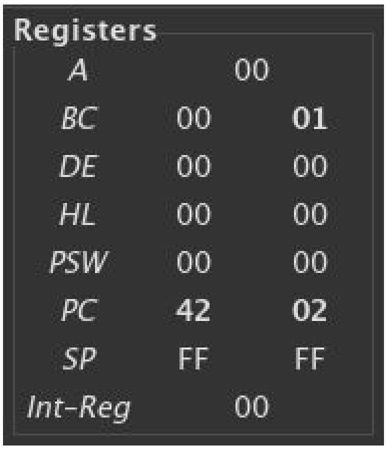
# MIT LAB 2

## SAHIL BONDRE: U18CO021

### (1) Write a program to shift 8-bit no by three bits left. Assume data is in register C.

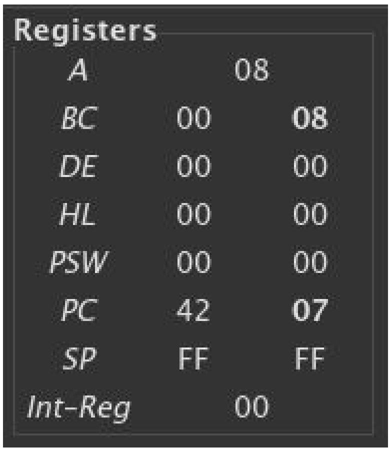
```
; initialize data in C
MVI C, 01H
; move to A
MOV A, C
; shift
RLC
RLC
RLC
; move back to C
MOV C, A
HLT
```



Registers

| | | |
|---|---|---|
| A | 00 | |
| BC | 00 | 01 |
| DE | 00 | 00 |
| HL | 00 | 00 |
| PSW | 00 | 00 |
| PC | 42 | 02 |
| SP | FF | FF |
| Int-Reg | 00 | |

C had 0x01 initially



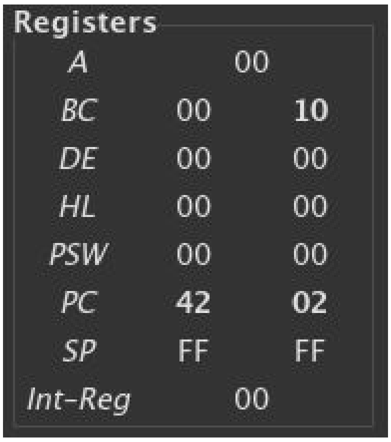
Registers

| | | |
|---|---|---|
| A | 08 | |
| BC | 00 | 08 |
| DE | 00 | 00 |
| HL | 00 | 00 |
| PSW | 00 | 00 |
| PC | 42 | 07 |
| SP | FF | FF |
| Int-Reg | 00 | |

After execution C had 0x08 due to left shift

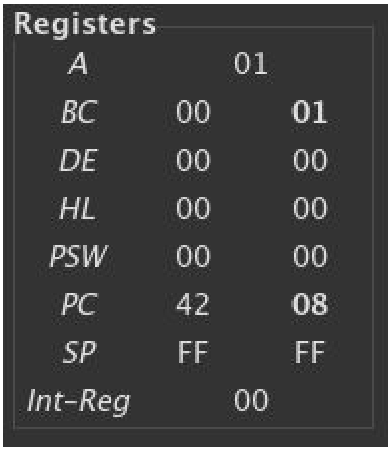
### (2) Write a program to shift 8-bit data four bits right. Assume data is in register C.

```
; initialize data in C
MVI C, 10H
; move to A
MOV A, C
; shift
RRC
RRC
RRC
RRC
; move back to C
MOV C, A
HLT
```



```
Registers
   A          00
   BC    00       10
   DE    00       00
   HL    00       00
  PSW    00       00
   PC    42       02
   SP    FF       FF
 Int-Reg      00
```

C had 0x10 initially

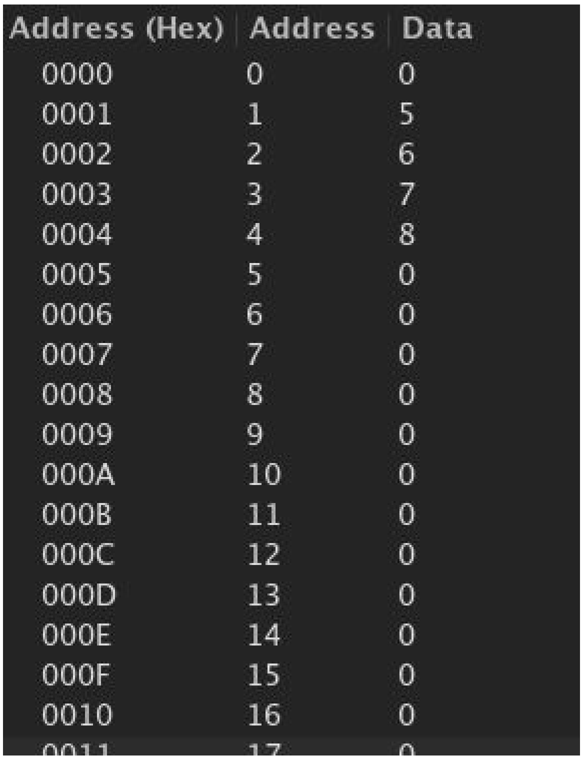


```
Registers
   A          01
   BC    00       01
   DE    00       00
   HL    00       00
  PSW    00       00
   PC    42       08
   SP    FF       FF
 Int-Reg      00
```

After execution C had 0x01 due to right shift

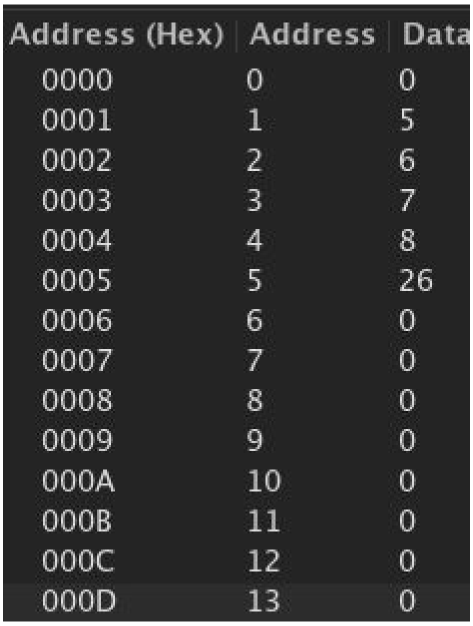
### (3) Program to Find Sum of Series of 8-bit Numbers

```
; clear accumulator
MVI A, 00H
; set memory location
LXI H, 0001H
; FIRST NUMBER
; move data to c
MOV C, M
ADD C
; increase memory by one
INX H

; SECOND NUMBER
MOV C, M
ADD C
; increase memory by one
INX H

; THIRD NUMBER
MOV C, M
ADD C
; increase memory by one
INX H

; FOURTH NUMBER
MOV C, M
ADD C

; Store to 0x0005
STA 0005H

HLT
```



| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 5 |
| 0002 | 2 | 6 |
| 0003 | 3 | 7 |
| 0004 | 4 | 8 |
| 0005 | 5 | 0 |
| 0006 | 6 | 0 |
| 0007 | 7 | 0 |
| 0008 | 8 | 0 |
| 0009 | 9 | 0 |
| 000A | 10 | 0 |
| 000B | 11 | 0 |
| 000C | 12 | 0 |
| 000D | 13 | 0 |
| 000E | 14 | 0 |
| 000F | 15 | 0 |
| 0010 | 16 | 0 |
| 0011 | 17 | 0 |

numbers to be added stored in 0001, 0002, 0003, 0004

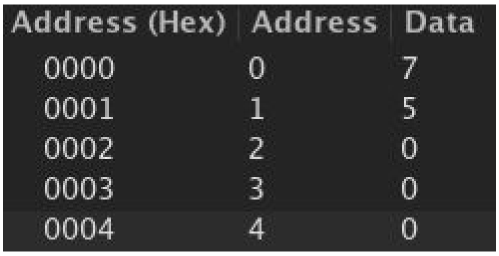


| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 5 |
| 0002 | 2 | 6 |
| 0003 | 3 | 7 |
| 0004 | 4 | 8 |
| 0005 | 5 | 26 |
| 0006 | 6 | 0 |
| 0007 | 7 | 0 |
| 0008 | 8 | 0 |
| 0009 | 9 | 0 |
| 000A | 10 | 0 |
| 000B | 11 | 0 |
| 000C | 12 | 0 |
| 000D | 13 | 0 |

sum of numbers stored in 0005 after execution

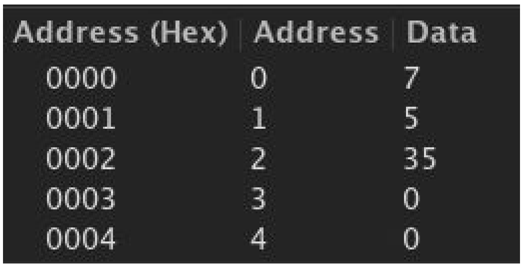
### (4) Program to Multiply Two 8-bit Numbers

```
; set B
MVI B, 00H
; set A
LDA 1H
MOV C, A
LDA 0H
MOV D, A
MVI A, 00H
; recurse
REPEAT: ADD D
JNC FWD
INR B
FWD: DCR C
JNZ REPEAT
STA 2H
; move to A
MOV A, B
; store to memory
STA 3H
HLT
```



| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 7 |
| 0001 | 1 | 5 |
| 0002 | 2 | 0 |
| 0003 | 3 | 0 |
| 0004 | 4 | 0 |

7 and 5 to be multiplied

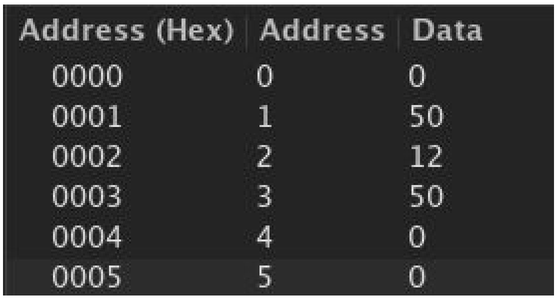


| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 7 |
| 0001 | 1 | 5 |
| 0002 | 2 | 35 |
| 0003 | 3 | 0 |
| 0004 | 4 | 0 |

35 stored as multiplication in 0x0002

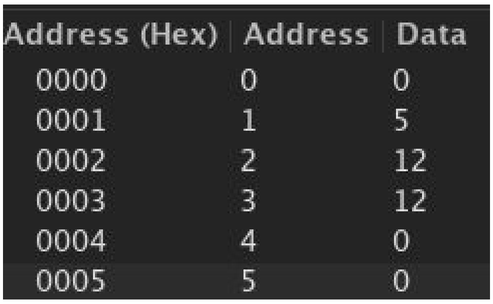
### (5) Largest of two 8-bit numbers

```
; set memory location
LXI H, 0001H
; first number
MOV A, M
INX H
; second number
MOV B, M
CMP B
; if carry is set A is less else B is less
JNC 420BH
MOV A, B
INX H
; Move answer to 0x0003
MOV M, A
HLT
```



| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 50 |
| 0002 | 2 | 12 |
| 0003 | 3 | 50 |
| 0004 | 4 | 0 |
| 0005 | 5 | 0 |

max of 50 and 12 placed in 0003



| Address (Hex) | Address | Data |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 5 |
| 0002 | 2 | 12 |
| 0003 | 3 | 12 |
| 0004 | 4 | 0 |
| 0005 | 5 | 0 |

max of 5 and 12 placed in 00003