



## CO Project

### 8&16 Bit CPU Simulation

Name:

Abdelrhman Abdelftah Kassem

18010948

## 8-Bit CPU Simulation:

Assembly Code for the used Example:

```
LDA    5
Add     #2
Add     6
STA     7
STOP
```

### Memory Contents:

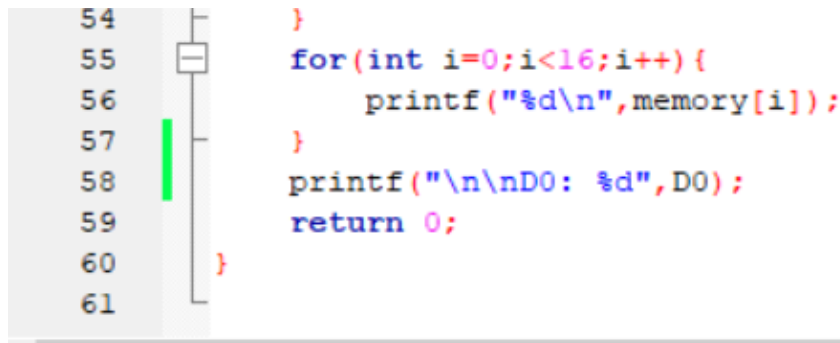
00000101	Load contents of 5 into D0
00110010	Add 2 to D0
00100110	Add the contents of Location 6 to D0
01000111	Store the contents of D0 in Location 7
11100000	Exit
00001111	Location 5 has a value of 15
00000011	Location 6 has a value of 3

Expected Output is that Location 7 now has a value of 20 (15+2+3).

## Edited Code:

Not much was edited here just added the memory contents at the start and made a loop to print the memory contents and the value of D0 at the end.

```
unsigned short int PC = 0;    /*program counter*/
unsigned short int D0 = 0;    /*data register*/
unsigned short int MAR;      /*memory address register*/
unsigned short int MBR;      /*memory buffer register*/
unsigned short int IR;       /*instruction register*/
unsigned short int operand;  /*the 8-bit operand from the IR*/
unsigned short int source;   /*source operand*/
unsigned short int opcode;   /*the 3-bit op-code from the IR*/
unsigned short int amode;    /*the 1-bit addressing mode*/
unsigned short int memory[16] = {0b000000101, 0b00110010, 0b00100110, 0b01000111, 0b11100000, 0b00001111, 0b00000011}; /*the memory*/
unsigned short int run = 1;  /*execute program while run is 1*/
```

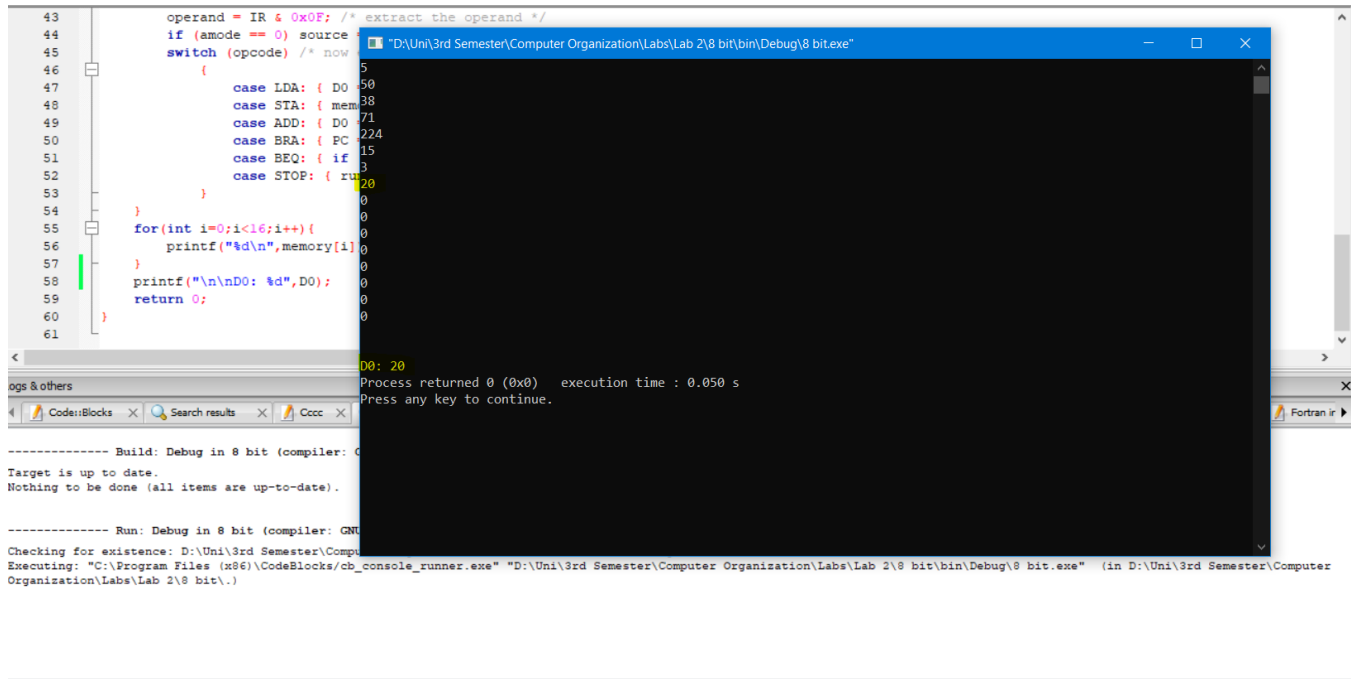


The screenshot shows a code editor with a line number column on the left (54 to 61) and a vertical scrollbar. The code is as follows:

```
54     }
55     for(int i=0;i<16;i++){
56         printf("%d\n",memory[i]);
57     }
58     printf("\n\nD0: %d",D0);
59     return 0;
60 }
61
```

Aside from this the code is identical to the example in the reference book.

## Screenshot of the Output:



The screenshot displays a code editor on the left and a terminal window on the right. The code editor shows assembly-like code with comments and a switch statement. The terminal window shows the execution output, including the address 00: 20 and the message "Process returned 0 (0x0) execution time : 0.050 s".

```
43 operand = IR & 0x0F; /* extract the operand */
44 if (amode == 0) source = 0;
45 switch (opcode) /* now
46 {
47     case LDA: { DO 50
48     case STA: { mem 38
49     case ADD: { DO 71
50     case BRA: { PC 224
51     case BEQ: { if 15
52     case STOP: { ru 3
53 }
54
55 for(int i=0;i<16;i++){
56     printf("%d\n",memory[i])
57 }
58 printf("\n\nD0: %d",D0);
59 return 0;
60 }
61
```

00: 20

Process returned 0 (0x0) execution time : 0.050 s  
Press any key to continue.

----- Build: Debug in 8 bit (compiler: GCC) -----  
Target is up to date.  
Nothing to be done (all items are up-to-date).

----- Run: Debug in 8 bit (compiler: GCC) -----  
Checking for existence: D:\Uni\3rd Semester\Computer Organization\Labs\Lab 2\8 bit\bin\Debug\8 bit.exe"  
Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "D:\Uni\3rd Semester\Computer Organization\Labs\Lab 2\8 bit\bin\Debug\8 bit.exe" (in D:\Uni\3rd Semester\Computer Organization\Labs\Lab 2\8 bit\.)

# 16-Bit CPU Simulation:

## Assembly Code for the used Example:

Move 20,D0

AND 21,D0

MOVE D0,22

MOVE 20,D0

OR 21,D0

MOVE D0,23

MOVE 20,D0

XOR 21,D0

MOVE D0,24

STOP

This Stores contents of memory location 20 into D0 then Applies AND operation on the contents of D0 and memory location 21.

The Code is repeated for Operation OR and XOR.

Each time the result is stored in a distinct subsequent memory location to be view at the end.

## Memory Contents:

00000100 – 00010100 Moves the content of memory Location 20 into D0.

10000100 – 00010101 applies AND operation on D0 and the contents of location 21.

00000000 – 00010110 Stores the content of D0 into Memory location 22.

00000100 – 00010100 Moves the content of memory Location 20 into D0.

10010100 – 00010101 applies OR operation on D0 and the contents of location 21.

00000000 – 00010111 Stores the content of D0 into Memory location 23.

00000100 – 00010100 Moves the content of memory Location 20 into D0.

10100100 – 00010101 applies OR operation on D0 and the contents of location 21.

00000000 – 00011000 Stores the content of D0 into Memory location 24.

11110000 – 00000000 Exit

10110111 Location 20 → The first operand (183).

11001011 Location 21 → The second operand (203).

After Execution.

10000011 Location 22 → The result of the AND operation (131).

11111111 Location 23 → The result of the OR operation (255)

01111100 Location 24 → The result of the XOR operation (124).

## Added Code.

Defining the AND,OR,XOR operations and assigning OP-codes to them:

```
5  #define MOVE 0
6  #define ADD 1
7  #define SUB 2
8  #define BRA 3
9  #define CMP 4
10 #define BEQ 5
11 #define BNE 6
12 #define EXG 7 /*EXG exchanges the contents of two registers */
13 #define AND 8
14 #define OR 9
15 #define XOR 10
16 #define STOP 15
17
```

Implementing the operations:

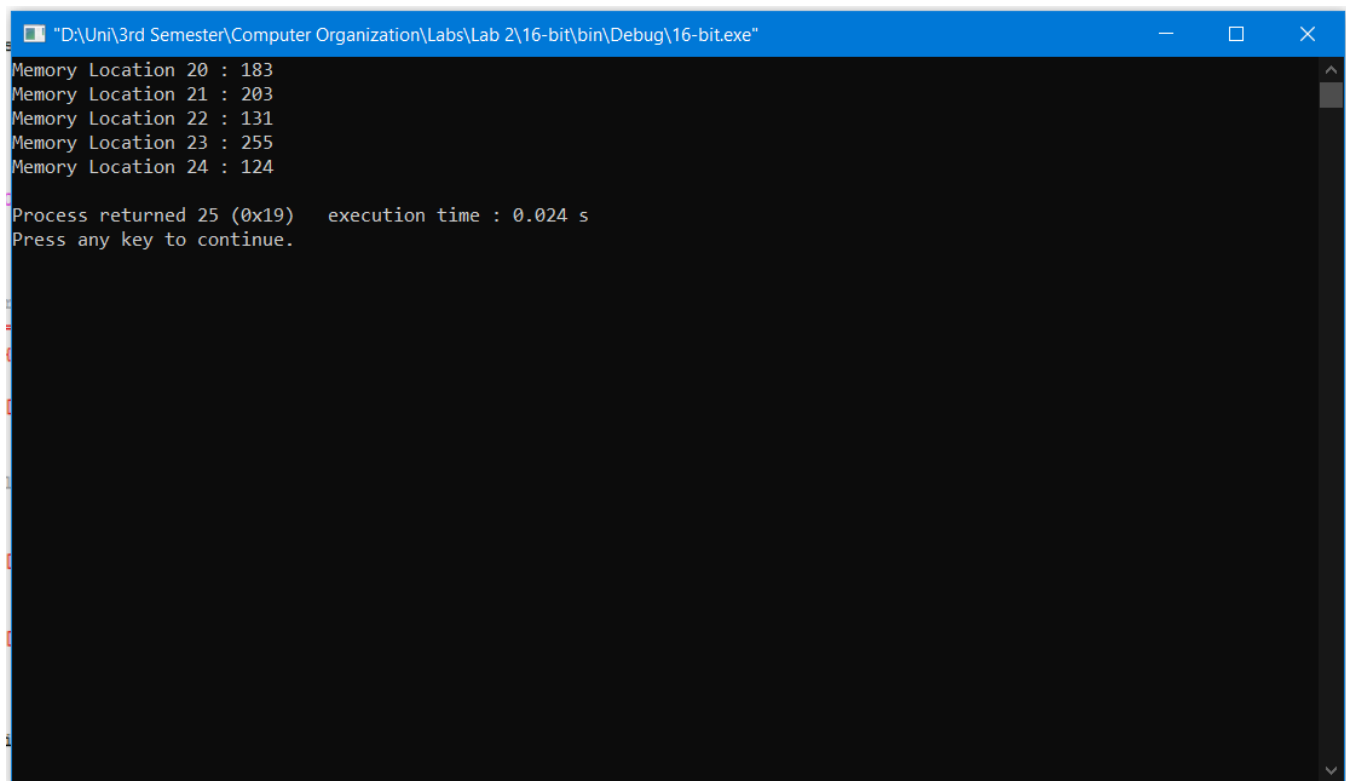
```
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
```

```
}
case AND: {
    if (direction == 0)
        destination = D0 & source;
    else
        D0 = D0 & source;
    break;
}
case OR: {
    if (direction == 0)
        destination = D0 | source;
    else
        D0 = D0 | source;
    break;
}
case XOR: {
    if (direction == 0)
        destination = D0 ^ source;
    else
        D0 = D0 ^ source;
    break;
}
```

Printing the required part from the memory (operands and output):

```
170 }  
171 for(int i=20;i<25;i++){  
172     printf("Memory Location %d : %d\n",i,memory[i]);  
173 }  
174 }  
175
```

Screenshot of the Output:



```
"D:\Uni\3rd Semester\Computer Organization\Labs\Lab 2\16-bit\bin\Debug\16-bit.exe"  
Memory Location 20 : 183  
Memory Location 21 : 203  
Memory Location 22 : 131  
Memory Location 23 : 255  
Memory Location 24 : 124  
Process returned 25 (0x19)   execution time : 0.024 s  
Press any key to continue.
```