Exercises:

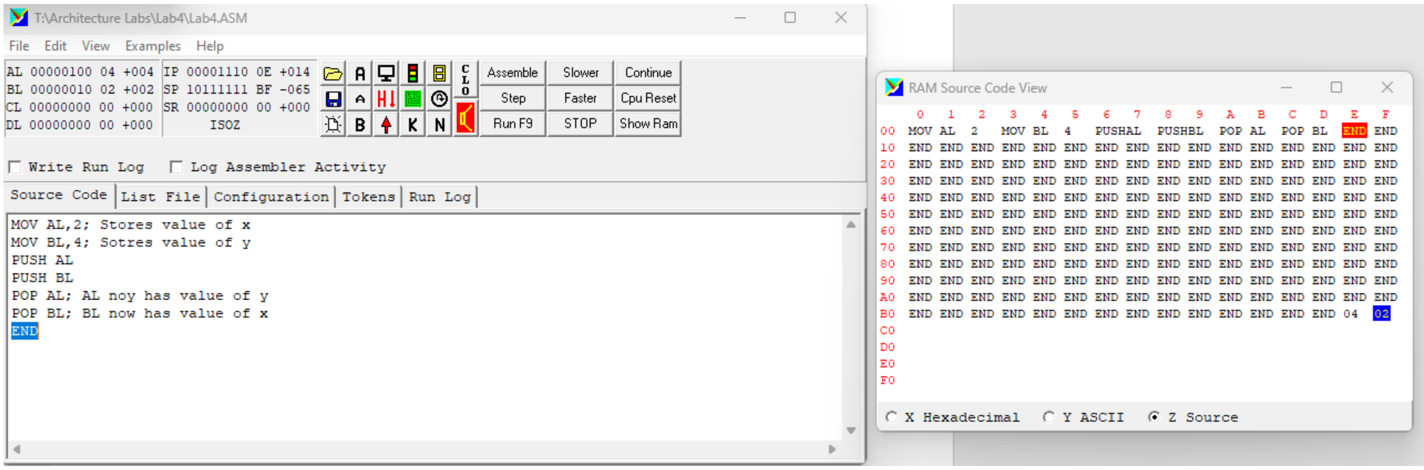
1. Write an assembly program that implements the following algorithm:

x <- 2

y <- 4

swap x and y

Use the stack (with the push and pop instructions) to swap the variables. Clearly mark which registers you are using to store x and y.



```
AL 00000100 04 +004  IP 00001110 0E +014
BL 00000010 02 +002  SP 10111111 BF -065
CL 00000000 00 +000  SR 00000000 00 +000
DL 00000000 00 +000          ISOZ
```

□ Write Run Log □ Log Assembler Activity

```
MOV AL,2; Stores value of x
MOV BL,4; Sotres value of y
PUSH AL
PUSH BL
POP AL; AL noy has value of y
POP BL; BL now has value of x
END
```

```
    0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
00  MOV AL  2   MOV BL  4   PUSHAL  PUSHBL  POP AL  POP BL  END END
10  END END END END END END END END END END END END END END END END
20  END END END END END END END END END END END END END END END END
30  END END END END END END END END END END END END END END END END
40  END END END END END END END END END END END END END END END END
50  END END END END END END END END END END END END END END END END
60  END END END END END END END END END END END END END END END END
70  END END END END END END END END END END END END END END END END
80  END END END END END END END END END END END END END END END END
90  END END END END END END END END END END END END END END END END
A0  END END END END END END END END END END END END END END END END
B0  END END END END END END END END END END END END END END 04  02
C0
D0
E0
F0
```

○ X Hexadecimal ○ Y ASCII ● Z Source

2. Write an assembly program that implements the following algorithm:

x <- 12 // 1100 in binary

y <- 5 // 0101 in binary

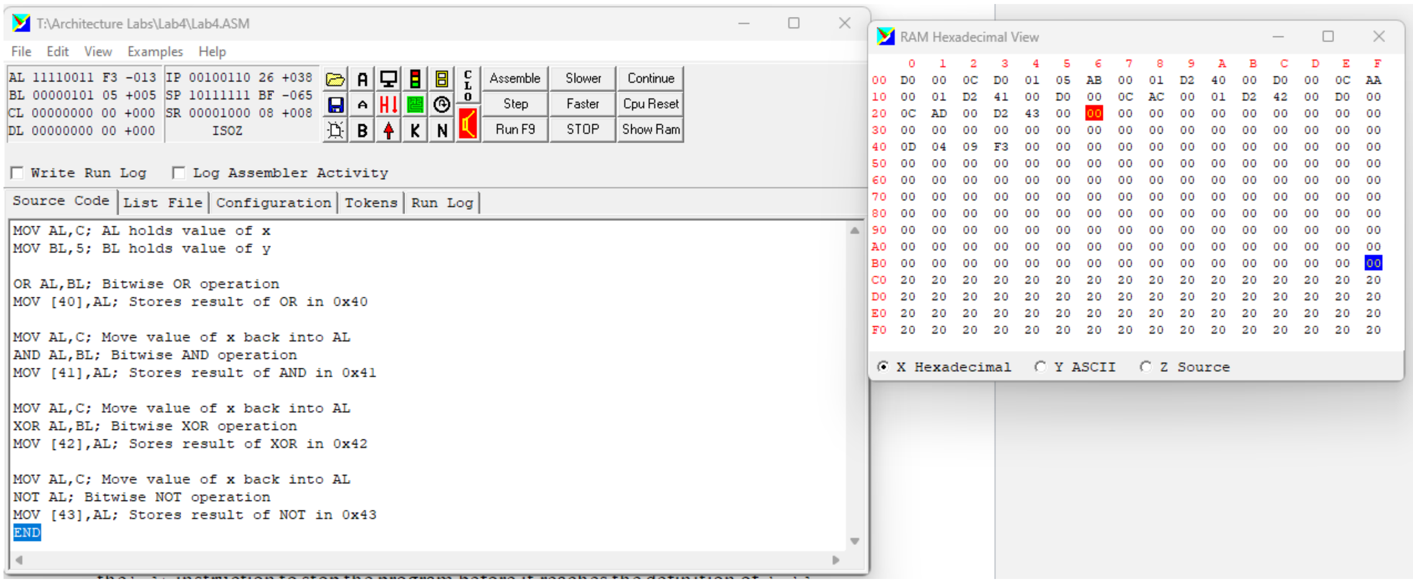
[0x40] <- x | y // | is the bit-wise OR operation

[0x41] <- x & y // & is the bit-wise AND operation

[0x42] <- x ^ y // ^ is the bit-wise XOR operation

[0x43] <- ~x // ~ is the bit-wise NOT operation

Clearly mark which registers you are using to store x and y.



```
AL 11110011 F3 -013   IP 00100110 26 +038
BL 00000101 05 +005   SP 10111111 BF -065
CL 00000000 00 +000   SR 00001000 08 +008
DL 00000000 00 +000          ISOZ
```

☐ Write Run Log ☐ Log Assembler Activity

Source Code | List File | Configuration | Tokens | Run Log

```
MOV AL,C; AL holds value of x
MOV BL,5; BL holds value of y

OR AL,BL; Bitwise OR operation
MOV [40],AL; Stores result of OR in 0x40

MOV AL,C; Move value of x back into AL
AND AL,BL; Bitwise AND operation
MOV [41],AL; Stores result of AND in 0x41

MOV AL,C; Move value of x back into AL
XOR AL,BL; Bitwise XOR operation
MOV [42],AL; Sores result of XOR in 0x42

MOV AL,C; Move value of x back into AL
NOT AL; Bitwise NOT operation
MOV [43],AL; Stores result of NOT in 0x43
END
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00  D0 00 0C D0 01 05 AB 00 01 D2 40 00 D0 00 0C AA
10  00 01 D2 41 00 D0 00 0C AC 00 01 D2 42 00 D0 00
20  0C AD 00 D2 43 00 00 00 00 00 00 00 00 00 00 00
30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40  0D 04 09 F3 00 00 00 00 00 00 00 00 00 00 00 00
50  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
D0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
F0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

● X Hexadecimal ○ Y ASCII ○ Z Source

3. Write an assembly program that implements the following algorithm:

x <- 5

double()

x <- x + 1

double()

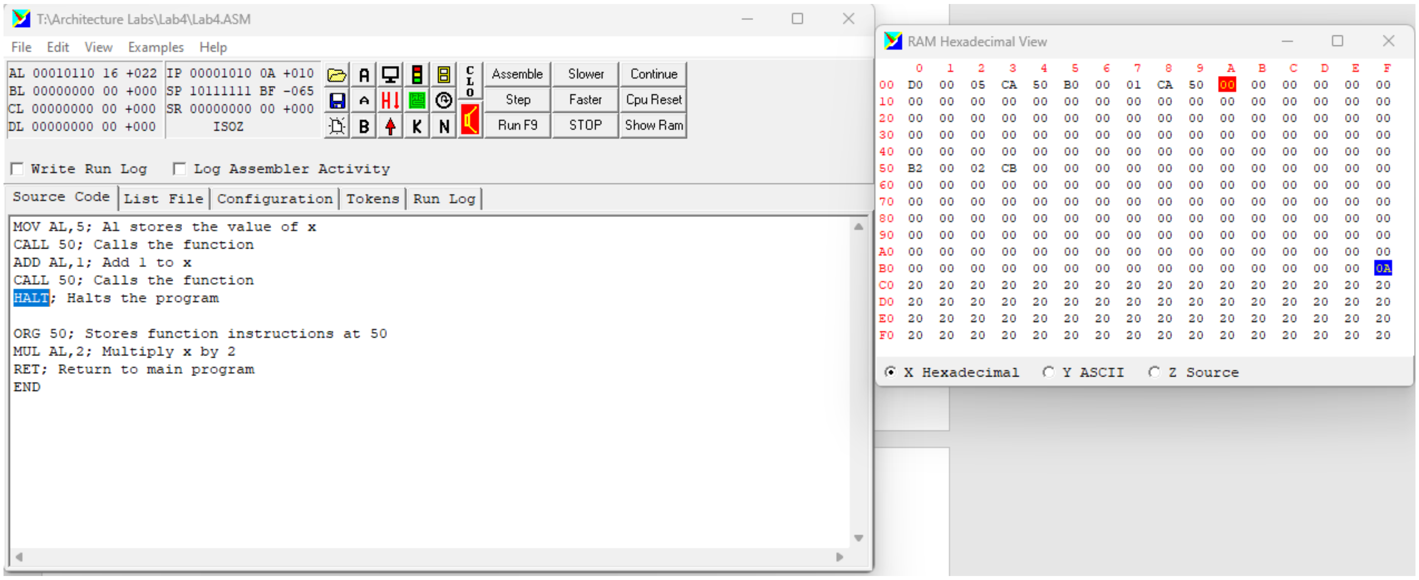
double()

{

    x <- x * 2

}

Clearly mark which register you are using to store the variable x. Do not take shortcuts, you must implement the function double. Hint: consider using the halt instruction to stop the program before it reaches the definition of double.



```
AL 00010110 16 +022  IP 00001010 0A +010
BL 00000000 00 +000  SP 10111111 BF -065
CL 00000000 00 +000  SR 00000000 00 +000
DL 00000000 00 +000         ISOZ
```

☐ Write Run Log   ☐ Log Assembler Activity

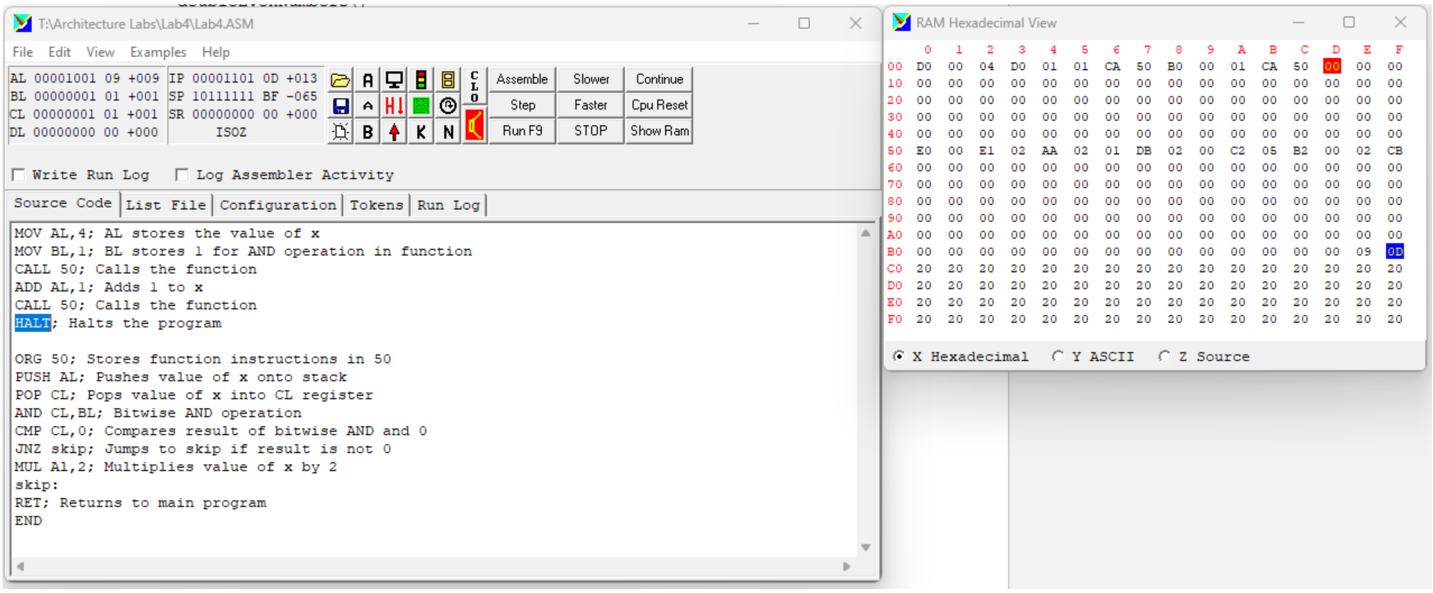
```
MOV AL,5; Al stores the value of x
CALL 50; Calls the function
ADD AL,1; Add 1 to x
CALL 50; Calls the function
HALT; Halts the program

ORG 50; Stores function instructions at 50
MUL AL,2; Multiply x by 2
RET; Return to main program
END
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00  D0 00 05 CA 50 B0 00 01 CA 50 00 00 00 00 00 00
10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50  B2 00 02 CB 00 00 00 00 00 00 00 00 00 00 00 00
60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A
C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
D0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
F0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

● X Hexadecimal   ○ Y ASCII   ○ Z Source

4. Write an assembly program that implements the following algorithm:

```
x <- 4

doubleEvenNumbers()

x <- x + 1

doubleEvenNumbers()

// This function doubles x if x is even, otherwise, it does nothing.


doubleEvenNumbers()

{

      if (x & 1 == 0)

      {

            x <- x * 2

      }

}
```



```
AL 00001001 09 +009   IP 00001101 0D +013
BL 00000001 01 +001   SP 10111111 BF -065
CL 00000001 01 +001   SR 00000000 00 +000
DL 00000000 00 +000          ISOZ
```

□ Write Run Log   □ Log Assembler Activity

```
MOV AL,4; AL stores the value of x
MOV BL,1; BL stores 1 for AND operation in function
CALL 50; Calls the function
ADD AL,1; Adds 1 to x
CALL 50; Calls the function
HALT; Halts the program

ORG 50; Stores function instructions in 50
PUSH AL; Pushes value of x onto stack
POP CL; Pops value of x into CL register
AND CL,BL; Bitwise AND operation
CMP CL,0; Compares result of bitwise AND and 0
JNZ skip; Jumps to skip if result is not 0
MUL Al,2; Multiplies value of x by 2
skip:
RET; Returns to main program
END
```

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 D0 00 04 D0 01 01 CA 50 B0 00 01 CA 50 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 E0 00 E1 02 AA 02 01 DB 02 00 C2 05 B2 00 02 CB
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 0D
C0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
D0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
F0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

● X Hexadecimal   ○ Y ASCII   ○ Z Source