

1. Write a function that calculates the power of a number. Pass the base and the power as parameters. Use RAM locations to pass the values to the function. When testing use small numbers to avoid possible overflow.

This example uses  $4^3$ , and the result is stored at [04]:

The screenshot shows an 8086 assembly editor with the following code:

```
AL 01000000 40 +064 IP 00010011 13 +019
BL 00000001 01 +001 SP 10111111 BF -065
CL 00000000 00 +000 SR 00000010 02 +002
DL 00000100 04 +004 IS02
```

Source Code:

```
JMP Start
DB 00
DB 00
DB 00

Start:
    MOV AL,4           ;Move the base (2) into AL
    MOV [02],AL        ;Move Al into [02]
    MOV BL,3           ;Move the power (3) into BL
    MOV [03],BL        ;Move BL into [3]
    CALL 30            ;Call the function
    HALT

    ORG 30             ;Create the function
    MOV AL,[02]        ;Move the base into AL
    MOV BL,[03]        ;Move the power into BL
    MOV DL,[02]        ;Move the base into DL for multiplication with AL

loop:
    MUL AL,DL          ;Perform multiplication
    SUB BL,1           ;Decrement counter (CL)
    CMP BL,1           ;Compare the counter (CL) with 1
    JNZ loop           ;If the counter is not 1, repeat the loop
    MOV [04],AL        ;Store the answer in [04]
    RET
END
```

RAM Source Code View:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
00 JMP STAR04 03 40 MOV AL 4 MOV [02]AL MOV BL 3 MOV [03]
10 BL CALL30 03 40 END END END END END END END END END
20 END END END END END END END END END END END END END
30 MOV AL [02]MOV BL [03]MOV DL [02]MUL AL DL SUB BL 1 CMP
40 BL 1 JNZ LOOPMOV [04]AL RET END END END END END END
50 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
70 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
90 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
B0 END END END END END END END END END END END END END
C0
D0
E0
F0
```

Buttons: X Hexadecimal Y ASCII Z Source

2. Take the integer converter from the previous lab, put the code into a function and pass the user's input to the function using the stack to pass the parameters. Enter four numbers and save them into an array.

The screenshot shows an x86 assembly IDE with the following components:

- Assembly Window:** Displays assembly code for a program that reads four numbers into an array. The code includes a main loop and a conversion function.
- RAM Source Code View:** A window showing the memory layout of the program, with addresses and corresponding assembly instructions.

**Assembly Code:**

```

AL 00110001 31 +049 IP 00100000 20 +032
BL 01010100 54 +084 SP 10111111 BF -065
CL 11111111 FF -001 SR 00000010 02 +002
DL 00010001 11 +017 IS0Z

END Program has halted.
Write Run Log Log Assembler Activity

Source Code List File Configuration Tokens Run Log

MOV BL,50
loop:
    MOV CL,[BL]
    CMP CL,FF ;Compare with end of array
    JZ end
    IN 00 ;User input in AL
    PUSH AL ;Push user input onto the stack
    CALL 40 ;Call function
    POP AL ;Pop converted number into AL
    MOV[BX],AL ;Converted values stored in array
    ADD AL,30 ;Add 48 (30 in hex) back to AL for comparison
    INC BL ;Increment BL to go to next location in the array
    CMP AL,0D ;Compare AL with enter
    JNZ loop
end:
    HALT

ORG 40 ;Create function to conver from ASCII
POP DL ;Pop the return address into DL
POP AL ;Get user input from the stack
SUB AL,30 ;Convert from ASCII by subtracting 48 (30 in hex)
PUSH AL ;Push Converted value onto the stack
PUSH DL ;Push the return address back onto the stack
RET

ORG 50 ;Declare Array
DB 00
DB 00
DB 00
DB 00
DB FF ;Set end of the array
END
  
```

**RAM Source Code View:**

Address	Instruction
00	MOV BL 50
10	MOV CL [BL]CMP CL FF JZ END IN 00 PUSHAL CALL
20	POP AL MOV [BL]AL ADD AL 30 INC BL CMP AL 0D JNZ LOOP
30	END END END END END END END END END END END
40	POP DL POP AL SUB AL 30 PUSHAL PUSHDL RET END END END
50	01 02 03 01 FF END END END END END END END END END
60	END END END END END END END END END END END
70	END END END END END END END END END END END
80	END END END END END END END END END END END
90	END END END END END END END END END END END
A0	END END END END END END END END END END END
B0	END END END END END END END END END END END
C0	END END END END END END END END END END END
D0	END END END END END END END END END END END
E0	END END END END END END END END END END END
F0	END END END END END END END END END END END

3. Create the following array [ 2, 5, 10, 0, 6 ]. Create a function that finds and returns the average of the array. Remember in high level programming we can pass an array to a function by passing the pointer of the array and we like to pass the size of the array as well.

The result is stored in DL in the function and pushed onto the stack before returning to the main function.

The screenshot shows a Windows environment with two windows open. The main window is an x86-64 assembler titled "T:\Architecture Labs\Lab6\Lab6Exercise3.ASM". It displays assembly code for a program that calculates the average of an array [2, 5, 10, 0, 6]. The code is as follows:

```

AL 00011010 1A +026 IP 00011010 1A +026
BL 00110101 35 +053 SP 10111110 BE -066
CL 00000101 05 +005 SR 00000000 00 +000
DL 00000100 04 +004 IS0Z

; Write Run Log
; Log Assembler Activity

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

MOV BL,30 ;Move the starting address of the array to BL
PUSH BL ;Push starting address of the array onto the stack

loop:
MOV AL,[BL] ;Move the array element into AL
CMP AL,FF ;Check if its the end of the array
JZ end ;Break the loop if its the end of the array so that CL isn't incremented
INC BL ;Go to next array element
INC CL ;Increment CL to hold length of the array
CMP AL,FF ;Check if its the end of the array
JNZ loop ;Repeat the loop if its not the end of the array

end:
PUSH CL ;Push the size of the array onto the stack
CALL 40
HALT

ORG 30 ;Declare the array
DB 2
DB 5
DB A
DB 0
DB 6
DB FF ;Set the end of the array as FF

ORG 40 ;Create the function
POP DL ;Pop the return address into DL
POP CL ;Pop the array size into CL to use as a counter
POP BL ;Pop the starting address of the array into BL
PUSH DL ;Push return address onto the stack
PUSH CL ;Push size of the array onto the stack
XOR DL,DL ;Reset DL

loop2:
MOV AL,[BL] ;Move the array element into AL
ADD DL,AL ;Add the elements of the array
SUB CL,1 ;Decrement CL
INC BL ;Increment BL to go to next element
CMP CL,0 ;Check if its the end of the array
JNZ loop2 ;Repeat the loop if its not the end of the array
POP CL ;Get size of the array from the stack
DIV DL,CL ;Divide the total of the elements by the length of the array
POP AL ;Pop the return address into AL
PUSH DL ;Push the result onto the stack
PUSH AL ;Push the return address back onto the stack
RET
END

```

The RAM Source Code View window shows the memory layout of the program. The array is stored at address 00000100, and the result is stored at address 00000104.