Assignment 5

Write 8086 assembly programs for the following.

1. **(Array initialization) Allocate 100 bytes of data, and assign value from 0-99 to those bytes.**

```
    .MODEL small
    .STACK 64
    .DATA
vec DB 100 dup(0)
    .CODE
main PROC
START:
      MOV ax, SEG vec
      MOV ds, ax
      MOV si, OFFSET vec ; LEA si, vec
      MOV ax, 0
      MOV bx, 100

LOOP:
    MOV [si], ax
    INC si
    INC ax
    CMP ax, bx
    JNZ LOOP

main ENDP
end main
```

2. **(If/else translation) Allocate a word in the memory with any value and another word for storing its absolute value. Write the program to perform the conversion.**

```
    .MODEL small
    .STACK 64

    .DATA
num1 dw -2
num2 dw 0

    .CODE
main PROC
START:
      MOV ax, SEG num1
      MOV ds, ax
      LEA si, num1
      MOV ax, [si]
      CMP ax, 0
      JL BELOW ; jump if less, signed value comparison
      MOV bx, ax
      JMP END
BELOW:
      MOV bx, 0
      SUB bx, ax
END:  MOV [si+2], bx
main ENDP

end main
```

3. **(Function argument and return value) Convert the following C code to the 8086 assembly code. You need to handle the function call argument passing and return value properly. [Hint: you can**

use stack for passing the argument and register AX for receiving the return value of the function.]

```
void main() {
        int a, b, c, d;
        a = -1;
        b = 1;
        c = abs(a);
        d = add(a, b);
}

int abs(a) {
        if (a > 0)
                return a;
        else
                return -a;
}

int add(a, b) {
        return a+b;
}
```

There are different ways to handle the argument passing and return value. We will use the following way called **CDECL calling convention**.
(https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions)

- Arguments are passed on the stack in Right-to-Left order, and return values are passed in ax.
- The **calling** function cleans the stack

The main function:

```
push ax ; pass a
call abs;
add sp, 2 ; clean the stack
mov c, ax; write the return value to c

push bx ; pass b
push ax ; pass a
call add;
add sp, 4 ; clean the stack
mov d, ax; write the return value to d
```

abs function (assume a far subroutine):

```
push bp ; save bp register before using it, we use bp to access the stack
mov bp, sp
mov ax, [bp + 4] ; because the [bp] and [bp+2] store CS and IP so we cannot
directly use pop instructions to get the argument.
 ; now ax gets the argument a
```

```
… ; codes to conversion
pop bp ; restore the register so other can use it
ret
```

add function (assume a far subroutine):

```
push bp ; save bp register before using it, we use bp to access the stack
mov bp, sp
mov ax, [bp + 4] ; because the [bp] and [bp+2] store CS and IP so we cannot
directly use pop instructions to get the argument.
 ; now ax gets the argument a
push bx, save the bx register before overwriting it
mov bx, [bp + 6] ; now ax gets the argument b
add ax, bx
pop bx ; restore the register so other can use it
pop bp ; restore the register so other can use it
ret
```

4. **(Function local variable) Convert the foo() function to assembly (no need to convert main). You need to handle the local variables properly. [Hint: 1. you can use the stack for local variables and remember to clean the stack before return.]**

   void main() {
   int a;

   a = foo();
   }

   void foo() {
   int a, b, c;
   }

We will use the stack for storing the local variables

The foo function:

```
push bp      ; save the value of bp
mov bp, sp ; bp now points to the top of the stack
sub bp, 6 ; space allocated on the stack for the three local variables
;a = [bp], b = [bp + 2], c = [bp + 4]
pop bp
ret

或者下边的做法：
push bp
mov bp, sp
sub sp, 6
;a=[sp],b=[sp+2],c=[sp+4]

mov sp, bp ;clean the stack
pop bp
ret
```