Assembly Language LAB

Islamic University – Gaza
Engineering Faculty
Department of Computer Engineering
2013
ECOM 2125: Assembly Language LAB
Created by: Eng. Ahmed M. Ayash
Modified and Presented By: Eihab S. El-Radie

Lab # 8

Stacks & Subroutines

Objective:

To learn how to use the stack and how to write assembly procedures.

Part1: Stack

The stack is part of memory used for temporary storage of addresses and data. It's **LIFO** structure.
The **SS** and **SP** registers point to the top of the stack like this: **SS:[SP]**.So the SS register is the segment and the SP register contains the offset of the most recently pushed value.

The **PUSH** instruction is used to place values on the stack and the **POP** instruction is used to remove values from the stack. We rarely manipulate ESP directly; instead, it is indirectly modified by instructions such as CALL, RET, PUSH, and POP.

- When we dealing with stack in the program we write:

  .stack

- To determine the size of the stack, which we want to use it, we put it as follows:

  .stack 100h

Where 100h is the size of the stack

1. PUSH Instruction

The PUSH instruction first decrements ESP and then copies either a 16- or 32-bit source operand into the stack.

- A 16-bit operand causes ESP to be decremented by 2.
- A 32-bit operand causes ESP to be decremented by 4.

❖ There are three instruction formats:
  - PUSH r /m16
  - PUSH r /m32
  - PUSH imm32

2. POP Instruction

The POP instruction first copies the contents of the stack element pointed to by ESP into a 16- or 32-bit destination operand and then increments ESP.

- A 16-bit operand causes ESP to be incremented by 2.
- A 32-bit operand causes ESP to be incremented by 4.

- There are two instruction formats:
  - POP r /m16
  - POP r /m32

3. PUSHFD and POPFD Instructions

The PUSHFD instruction pushes the 32-bit EFLAGS register on the stack, and POPFD pops the stack into EFLAGS:

- pushfd
- popfd

16-bit programs use the **PUSHF** instruction to push the 16-bit FLAGS register on the stack and **POPF** to pop the stack into FLAGS.

4. PUSHAD, PUSHA, POPAD, and POPA

- The **PUSHAD** instruction pushes all of the 32- bit general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, and EDI.

- ESP (value before executing PUSHAD)

- The **POPAD** instruction pops the same registers off the stack in reverse order.

- The **PUSHA** instruction pushes the l6-bit general-purpose registers (AX, CX, DX, BX, SP, BP, SI, DI) on the stack in the order listed.

- The **POPA** instruction pops the same registers in reverse order.

Notes: we will use the following interrupt services:

- **Service 01h:** DOS get character function

```
mov ah,01h    ; returns ASCII code of character to AL
int 21h       ; and echo it to the monitor
```

- **Service 02h:** DOS print character function

```
mov ah,02h
mov dl,ASCII#    ; ASCII code of character for print in DL
int 21h
```

- **Service 08h:** Get character without echo

```
mov ah,08h    ; returns ASCII code of character to AL
int 21h       ; but don't echo it to the monitor
```

Lab work:

Exercise 1:

Write an assembly language program that asks the user to enter a password. The program prints the password as stars on the screen. If the password is right, the program should print **'Correct Password**'. Else, it will print '**Incorrect Password**'

edit ex81.asm

File Edit Search View Options Help

E:\[university]\[[TA]]\{{Assembly}}\Lab work\ex81.asm

```
 .model small
 .386
 .stack 100h
 .data
password db 'assembly'
msg db "Please, Enter your Password: ", 0ah, 0dh,'$'
correct_msg db 0ah, 0dh, 'Correct Password', 0ah, 0dh, '$'
wrong_msg db 0ah, 0dh, 'Incorrect Password', 0ah, 0dh, '$'

 .code
main:
 mov ax,@data
 mov ds, ax

 mov ah, 9
 mov dx, offset msg
 int 21h

 mov bx, offset password
 mov cx, sizeof password; if you work on TASM, use immediate value
x:
 mov ah,08h    ;get char without echo
 int 21h
 push ax      ; entered
 push [bx]
 inc bx
 mov ah, 2
 mov dl, '*'
 int 21h
loop x

 mov cx, sizeof password
y:
 pop bx
 pop ax
 cmp al, bl
 jne not_equal
loop y
 mov ah, 9
 mov dx, offset correct_msg
 int 21h
 jmp exit
not_equal:
 mov ah, 9
 mov dx, offset wrong_msg
 int 21h
exit:
 mov ah, 4ch
 int 21h
End main
```

F1=Help | Line:38 Col:34

Output:

C:\Windows\system32\cmd.exe

```
E:\_UNIVE~1\__TA__~1\{{ASSE~1\LABWOR~1>ex81
Please, Enter your Password:
********
Correct Password

E:\_UNIVE~1\__TA__~1\{{ASSE~1\LABWOR~1>
```

Part2: Subroutines (Procedures)

A subroutine is a special part of the program that can be called for execution from any point in the program. The subroutine is written to provide a function that must be performed frequently at various points in the main program. Whenever the function must be performed, a single instruction is inserted into the main body of the program to **CALL** the subroutine. **RET** instruction must be included at the end of the subroutine to return to the main program.

- **The CALL instruction:** calls a procedure
  - pushes offset of next instruction on the stack
  - copies the address of the called procedure into EIP
- **The RET instruction**: returns from a procedure
  - pops top of stack into EIP

- Following is an assembly language procedure named sample:

```
sample PROC
.
.
ret
sample ENDP
```

hint: To convert a hexadecimal digit (x) to its ASCII code (y):

$$y = \begin{cases} x+30h & \text{if } 0<x<9 \\ x+37h & \text{if } A<x<F \\ ERROR & \text{else} \end{cases}$$

Lab work:

Exercise 2:

This program calls a subroutine that finds the largest unsigned number in a vector with N elements:

Administrator: cmd - edit ex2.asm

File Edit Search View Options Help

C:\Masm615\Assembly\Lab8\ex2.asm

```
  .model small
  .386
  .stack 100h
  .data
array db 0FEh,12h,13h,20h,90h,0AFh,33h,76h,0FFh,11h
msg db 'The Largest Number is $'
 .code
main:
 mov ax, @data
 mov ds, ax

 mov si, offset array
 mov cx,lengthof array
 call largest
 push ax
 mov ah, 9
 mov dx,offset msg
 int 21h
 pop ax
 mov dl,ah  ; largest
 call print
 exit:
 mov ah,4ch
 int 21h
largest Proc
 mov ah,[si]
 dec cx
 inc si
start:
 mov al,[si]
 cmp ah,al
 ja x
 mov ah,al
x:
 inc si
 loop start
 ret
 largest EndP
print:
 push dx
 shr dl, 4
 call hexa2ascii
 mov ah, 2
 int 21h
 pop dx
 and dl, 0fh
 call hexa2ascii
 mov ah, 2
 int 21h
 ret
hexa2ascii:
 cmp dl, 9
 ja abcdef
 add dl, 30h
 ret
abcdef:
 add dl,37h
 ret
END main
```

F1=Help Li

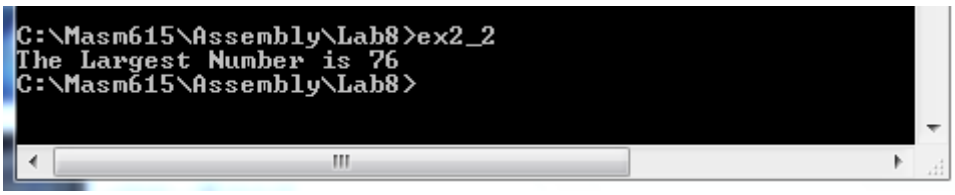
Output:

```
C:\Masm615\Assembly\Lab8>ex2
The Largest Number is FF
C:\Masm615\Assembly\Lab8>
```

Homework:

1. Write an assembly program that gets a character from Keyboard and displays its ASCII.

2. Modify the code of exercise 2 to find the largest signed number in a vector with N elements.



3. Write a program to determine if a bcd number (0-9) is prime. (A prime number is evenly divisible by only itself and 1). Your program should prompt the user for a number and then display a message indicating whether or not the number is prime.