National University of Computer and Emerging Sciences



**Laboratory Manual**

*for*

*Computer Organization and Assembly Language*

Course Instructors

Lab Instructor(s)

Section

Semester

**Department of Computer Science**

# COAL Lab 10 Manual

|  |
| --- |
| **Objectives:**     * A flash back to Irvine32 Library * Runtime stack * Modular approach, Defining & Using Procedures * Problems & Assignments |

## 10.1 Introduction

1. How to divide programs into manageable units by calling subroutines;
2. How programming languages use the runtime stack to track subroutine calls.

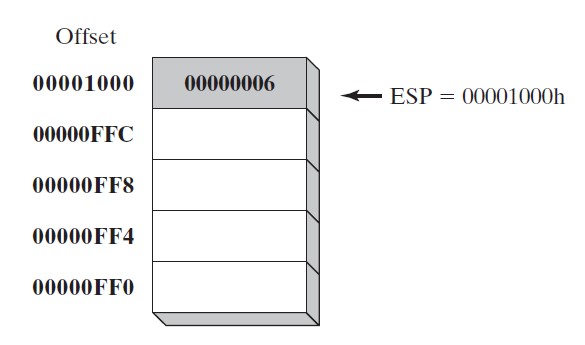
## 10.2 Irvine32 library

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Command** | **Input** | **Output** | **Function** |
| 1. | **ReadChar** | User | AL | Waits for a single character to be typed at the keyboard and returns the character. |
| 2. | **ReadDec** | EAX | Reads an unsigned 32-bit decimal integer from the keyboard, terminated by the Enter key. |
| 3. | **ReadHex** | Reads a 32-bit hexadecimal integer from the keyboard, terminated by the Enter key. |
| 4. | **ReadInt** | Reads a 32-bit signed decimal integer from the keyboard, terminated by the Enter key. |
| 5. | **ReadString** | EDX = offset buffer  ECX = Sizeof  buffer | EAX=No. of character entered buffer = string | Reads a string from the keyboard, terminated by the Enter key. |
| 6. | **WaitMsg** | Nill | Nill | Displays a message and waits for a key to be pressed. |
| 7. | **WriteBin** | EAX | Console | Writes an unsigned 32-bit integer to the console window in ASCII binary format. |
| 8. | **WriteBinB** | EAX= value EBX= type | Writes a binary integer to the console window in byte, word, or doubleword format. |
| 9. | **WriteChar** | AL | Writes a single character to the console window. |
| 10. | **WriteDec** | EAX | Writes an unsigned 32-bit integer to the console window in decimal format. |
| 11. | **WriteHex** | Writes a 32-bit integer to the console window in hexadecimal format. |
| 12. | **WriteHexB** | EAX= value EBX= type | Writes a byte, word, or doubleword integer to the console window in hexadecimal format. |
| 13. | **WriteInt** | EAX | Writes a signed 32-bit integer to the console window in decimal format. |
| 14. | **WriteString** | EDX= offset string |  | Writes a null-terminated string to the console window. |

## 10.3 Runtime Stack

We concentrate specifically on the runtime stack. It is supported directly by hardware in the CPU, and it is an essential part of the mechanism for calling and returning from procedures. Most of the time, we just call it the stack.

The runtime stack is a memory array managed directly by the CPU, using the ESP register, known as the stack pointer register. The ESP register holds a 32-bit offset into some location on the stack. We rarely manipulate ESP directly; instead, it is indirectly modified by instructions such as CALL, RET, PUSH, and POP. ESP always points to the last value to be added to, or pushed on, the top of stack.



### Applications

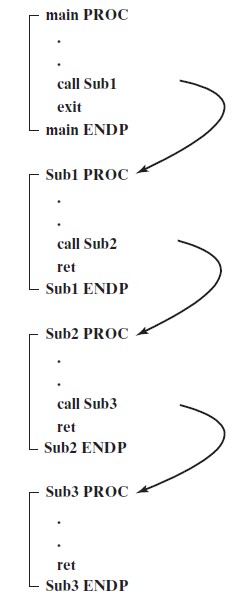
There are several important uses of runtime stacks in programs:

1. A stack makes a convenient temporary save area for registers when they are used for more than one purpose. After they are modified, they can be restored to their original values.
2. When the CALL instruction executes, the CPU saves the current subroutine’s return address on the stack.
3. When calling a subroutine, you pass input values called arguments by pushing them on the stack.
4. The stack provides temporary storage for local variables inside subroutines.

### Instructions

|  |  |  |
| --- | --- | --- |
| Command | Syntax | Function |
| **PUSH** | PUSH *reg/mem32* PUSH *imm32* | A 32-bit operand causes ESP to be decremented by 4. |
| **POP** | POP *reg/mem16* POP *reg/mem32* | Copies the contents of the stack element pointed to by ESP into a 16- or 32-bit destination operand and then increments ESP by 2 (word size) by 4 (dword size). |
| **PUSHFD** | pushfd | Pushes the 32-bit EFLAGS register on the stack. |
| **POPFD** | popfd | Pops the stack into EFLAGS |
| **PUSHAD** | pushad | Pushes all of the 32-bit general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, |
|  |  | ESP (value before executing PUSHAD), EBP, ESI, and EDI. |
| **POPAD** | popad | Pops the same registers off the stack in reverse order |

## 10.4 Defining and Using Procedures



### USES Operator

The USES operator, coupled with the PROC directive, lets you list the names of all registers modified within a procedure. USES tells the assembler to do two things: First, generate PUSH instructions that save the registers on the stack at the beginning of the procedure. Second, generate POP instructions that restore the register values at the end of the procedure.

ArraySum PROC **USES esi ecx**

|  |  |  |
| --- | --- | --- |
| mov eax,0  L1: |  | ; set the sum to zero |
| add eax,[esi] |  | ; add each integer to sum |
| add esi,TYPE DWORD |  | ; point to next integer |
| loop L1 |  | ; repeat for array size |
| ret  ArraySum ENDP |  | ; sum is in EAX |

# Problem(s) / Assignment(s)

|  |  |
| --- | --- |
| **Discussion & Practice** | **Estimated completion time: 1 hr, 30 mins** |

|  |
| --- |
| **Problem 10.1:** *Random Number Generator* **Estimated completion time:20 mins**    Write a program that should,   1. Randomly generate 10 unsigned integers in the range 0 to 4,294,967,294. 2. Next, it generates 10 signed integers in the range -50 to +49. Use builtin procedures *Random32*, *Randomize*, *RandomRange* as shown below.   Follow modular approach i.e., make a procedure RAND\_U for unsigned integers and RAND\_S for signed integers. Your main should only be used for calling the procedures.    **Sample output:**  3221236194 2210931702 974700167 367494257 2227888607  926772240 506254858 1769123448 2288603673 736071794  -34 +27 +38 -34 +31 -13 -29 +44 -48 -43  Solution:  include Irvine32.inc  .data  space byte ' ',0  number dword 0  lower\_range equ +50  upper\_range equ +49  .code  main proc  call RAND\_U  call RAND\_S  exit1:  call crlf  exit  main endp  RAND\_U proc  mov ecx,10  l1:  mov eax,4294967294  call randomrange  mov number,eax  mov edx,offset number  call writedec  mov edx,offset space  call writestring  loop l1  ret  RAND\_U endp  RAND\_S proc  call crlf  call crlf  mov ebx,0  mov ecx,10  loop1:  mov eax,upper\_range + lower\_range  call randomrange  sub eax,lower\_range  mov number,eax  mov edx,offset number  call writeint  mov edx,offset space  call writestring  loop loop1  ret  RAND\_S endp  end main |

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Input | Output | Function |
| **Random32** | Seed | EAX | Returns a 32-bit random integer. |
| **Randomize** |  |  | Initializes the starting seed value of the Random32 and RandomRange procedures. The seed equals the time of day, accurate to 1/100 of a second. |
| **RandomRange** | EAX= n | EAX | Produces a random integer within the range of 0 to n-1. |

|  |  |
| --- | --- |
| **Problem 10.2:** *Reversing a String*  Write a program using LOOP instruction with index addressing that copies a string from source to target, reversing the character order in the process. Use the following variables:  **source BYTE "This is the source string",0 target BYTE SIZEOF source DUP('#')**  Use DumpMem to display the string. If your program works correctly, it will display the following sequence of hexadecimal bytes:  67 6E 69 72 74 73 20 65 63 72 75 6F 73 20 65 68  74 20 73 69 20 73 69 68 54 | **Estimated completion time:15 mins** |

Use Stack to implement this program.

Solution :

include irvine32.inc

.data

source BYTE "This is the source string",0

len equ $-source

target BYTE SIZEOF source DUP(80)

.code

mov eax ,0

mov ebx,0

;;mov eax,len

main proc

mov esi,offset source

mov ecx,lengthof source

;;add esi,eax

mov edi , offset target

l1:

push [esi]

inc esi

loop l1

mov ecx,lengthof source

l2:

pop eax

mov [edi],eax

inc edi

loop l2

MOV ESI,OFFSET target

MOV ECX,LENGTHOF target

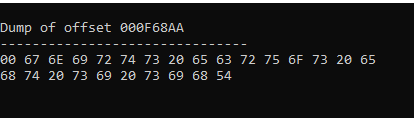
MOV EBX, TYPE target

CALL DumpMEM

exit

main endp

end main



## You are done with your exercise(s), make your submission ☺