Input/Output Routines for Assembly Programming

I/O Routines - Output

print_int	prints out to the screen the value of the integer stored in EAX
print_char	prints out to the screen the character whose ASCII value stored in AL
print_string	prints out to the screen the contents of the string at the address stored in EAX. The string must be a C-type string (i.e. null terminated).
print_nl	prints out to the screen a new line character.

I/O Routines - Input

read_int	reads an integer from the keyboard and stores it into the EAX register.
read_char	reads a single character from the keyboard and stores its ASCII code into the EAX register.

Debugging Macros – Dump Registers and Memory Values

dump_regs … dump_regs 1 … dump_regs 2 … dump_regs 3	prints out the values of the registers (in hex) to the screen. It also displays the bits set in the EFLAGS register. For example, if the zero flag is 1, ZF is displayed. If it is 0, it is not displayed. It takes a single integer argument that is printed out as well. This can be used to distinguish the output of different dump regs commands.
dump_mem … dump_mem 1, 100, 5 …	prints out the values of a region of memory (in hex) and also as ASCII characters. It takes three comma delimited arguments. The first is an integer that is used to label the output. The second argument is the address to display (This can be a label.) The last argument is the number of 16-byte paragraphs to display after the address. The memory displayed will start on the first paragraph boundary before the requested address.

Debugging Macros – Dump Stack

dump_stack	prints out the values on the CPU stack. The stack is organized as double words and this routine displays them this way. It takes three comma delimited arguments. The first is an integer label (like dump regs). The second is the number of double words to display below the address that the EBP register holds and the third argument is the number of double words to display above the address in EBP.

```
...
dump_stack  1, 4, 3
...
```

sample_io.asm

```
;       % nasm -f elf sample_io.asm   (1)
;       % nasm -f elf -d ELF_TYPE asm_io.asm     (2)
;       % gcc -o sample_io sample_io.o asm_io.o       (3)
;       % ./sample_io

%include "asm_io.inc"

        segment .data
msg     db      "asm_io library test", 0x0a, 0x00

        segment .text
        global main
main:
        enter   0,0

        dump_regs       1
        ;print out a new line character
        call    print_nl

        ; print out a character whose ASCII value stored in AL
        mov     al, 'A'
        call    print_char
        call    print_nl
        dump_regs       2
        ;print out a new line character
        call    print_nl
```

sample_io.asm

```
 ; print out value of integer stored in EAL
mov     eax,1024*1024*2
call     print_int
 call    print_nl
 dump_regs        3
 ;print out a new line character
 call    print_nl

 ; print out the contents of the string at the address
 ; stored in EAX
 mov     eax, msg
 call    print_string
 dump_regs        4
 ;print out a new line character
 call    print_nl

 call    read_int
 call    print_int
 call    print_nl

 leave
 ret
```

To Run sample_io

1. To assemble your source code
   % nasm –f elf sample_io.asm

2. To assemble asm_io library
   % nasm –f elf –d ELF_TYPE asm_io.asm

3. To link
   % gcc –o sample_io sample_io.o asm_io.o

4. To run
   % ./sample_io

skeleton.asm

```
; text segment must have an GLOBAL entry point 'main'
; text segment must start with 'enter 0, 0' instruction
; text segment must end with leave' and 'ret' instruction
; must include "asm_io.inc"

%include "asm_io.inc"

        segment .data
        ;
        ; initialized data is put in the data segment here
        ;

        segment .text
        global  main
main:
        enter   0,0             ; setup stack frame
        pusha

        ;
        ; code is put in the text segment. Do not modify
        ; the code before or after this comment.
        ;

        popa
        mov     eax, 0          ; return value
        leave                   ; leave stack frame
        ret
```