Assembly and Machine Language - Spring 1398 (2019) Midterm Exam	Instructor: B. Nasihatkon	دانشگاه صنعتی خواجه نصیرالدین طوسی K. N. TOOSI UNIVERSITY OF TECHNOLOGY
Name:	ID:	Ordibehesht 1397 - May 2018

Functions from the book

`call print_int`	prints EAX as a signed integer
`call print_nl`	prints a newline character

Use 32-bit Netwide assembler code on a Linux machine.

Programming: Write programs in the designated code area as follows:

label	command	arguments
`loop1:`	`call`	`prog2`
	`add`	`eax, ebx`
`prog2:`		

Question 1 (20 points) Update the values of the required registered after running each of the assembly instructions below. Notice that each instruction depends on the previous one. Write down the complete solutions for the signed cases.

command	AX (hex)	**AL** decimal (unsigned)	AL decimal (signed)	AH decimal (signed)
`mov ax,0x12C8`	12C8	200	-56 C8h = 11001000 2C : 00111000=56	18 12h is positive 1*16+2 = 18
`shl ax, 3`	9640	64	64 40h is positive 4*16+0=64	-106 96h = 10010110 2C = 01101010=106
`sar ah, 2`	E540	64	64 unchanged	-27 E5h = 11100101 2C = 00011011 = 27
`ror ax, 1`	72A0	160	-96 A0h = 10100000 2C = 01100000 = 96	114 72h is positive 7*16+2=114
`add al, ah`	7212	18	18 12h is positive 1*16+2 = 18	114 unchanged

Question 2 (20 points)

What does the following code print? How the output relates to the input. What does each of the loops do? Explain each part of the code on the right-hand side. Assume that the input is positive.

```
        call read_int
        mov  ebx, eax

        mov esi, 0
        mov ecx, 1
loop1:
        cmp ecx, ebx
        ja endloop1

        mov eax, ebx
        mov edx, 0
        div ecx

        cmp edx, 0
        jnz notzero

        push ecx
        inc  esi
notzero:

        inc ecx
        jmp loop1
endloop1:


        mov eax, 0
        mov ecx, esi
loop2:
        pop ebx
        add eax, ebx

        loop loop2

        call print_int
        call print_nl
```

Let n be the input. The code reads an input n and prints the sum of its divisors. The first loop pushes the divisors on the stack. The second loop adds up the elements pushed on the stack.

```
ESI = no_of_divisors = 0

for ECX = 1...n {

   EDX = remainder = n % ECX


   if (remainder = 0) {
      push ECX
      no_of_divisors += 1
   }




    ECX += 1
}


EAX = 0
Repeat no_divisors  times {

     EAX += stack.pop();

}

print EAX
```

Question 3 For each piece of assembly code in the left column, write a **single** equivalent assembly instruction. Disregard changes to the FLAGS registers. Explain your answer in the final column. (25 points)

	Single Instruction	Explanation
`rol eax, 7` `and eax, 0xFFFFFF80`	`shl eax, 7`	Rotates the bits of EAX 7 bits to the left, then zeros out the lowest 7 bits.

`jnc nocarry`
`inc eax`
`nocarry:`
`add eax, ebx`
	`ADC eax, ebx`	EAX = EAX + EBX + CarryFlag

`push edx`
 	
`mov edx, 0x80000000`
`and edx, eax`
`shr eax, 1`
`or eax, edx`
 	
`pop edx`
	`SAR eax, 1`	Tests the sign bit of the EAX. Then shifts EAX to the right (fills with zero from left). If the sign bit of EAX was 1 in the first place, sets the last bit of the shifted EAX to 1.

`push edx`
 	
`xor edx,edx`
`mov dl, al`
`shl edx, 24`
`shr eax, 8`
`or eax, edx`
 	
`pop edx`
	`ROR EAX, 8`	Saves the first 8 bits of EAX in DL. Shift EAX 8 bits to the right. Then sets the last 8 bits of EAX to what was saved in the DL.

`push ebx`
`push ecx`
`push edx`
 	
`mov ecx, ebx`
`mov edx, eax`
`not ecx`
`not edx`
 	
`and ebx, edx`
`and eax, ecx`
`or eax, ebx`
 	
`pop edx`
`pop ecx`
`pop ebx`
	`XOR EAX, EBX`	`A XOR B = (not(A) and B)`
		`OR`
		`(A and not(B))`

Question 4 We want to implement a function with a variable number of arguments. `int sum(int n, ...).` The first argument **n** is always equal to the number of the remaining arguments. The function computes and returns the sum of the remaining arguments. For example `sum(3,4,7,5)` returns 16, while `sum(3,4,7,5,8)` is invalid (we never perform such a call). The assembly code below consists of two files: **main.asm** and **sum.asm**. On the left (main.asm) write an assembly code which computes the sum of the registers **eax, ebx, ecx, edx, esi, and edi** by calling the function sum, and then prints it using the **print_int** function. On the right (sum.asm) write the body of the function `sum`. Assume that the first argument **n** is positive. **Observe all C declaration calling conventions.** Define the appropriate derivatives global, extern if needed. **(35 points)**

main.asm

label	command	arguments
segment .text		
main:		
	push	edi
	push	esi
	push	edx
	push	ecx
	push	ebx
	push	eax
	push	6
	call	sum
	add	esp, 28
	call	print_int
	call	print_nl
	mov	ebx, 0
	mov	eax, 1
	int	0x80

sum.asm

label	command	arguments
sum:	push	ebp
	mov	ebp, esp
	mov	eax, 0
	mov	edx, ebp
	add	edx, 12
	mov	ecx, [ebp+8]
loop1:		
	add	eax, [edx]
	add	edx, 4
	loop	loop1
	pop	ebp
	ret	