Your name: _____

ICS312 – Spring 2014

Midterm #1

Exercise #1:	Data segments	23	pts	
Exercise #2:	Arithmetic	9	pts	
Exercise #3:	If-then-else	10	pts	
Exercise #4:	Jump instructions	10	pts	
Exercise #5:	Mystery program	5	pts	
Exercise #6:	Dividing	5	pts	[Extra Credit]
		57+5	pts	

Exercise #1: Data segments

Consider the following .data segment declaration in NASM assembly:

```
A     times 2 dw     0AA12h
B     db         "a","b",0
C     dd         -13, 10
D     dw         1111000010101100b
E     db         "abc"
```

Question #1 [2pts]: How many bytes doe this segment declare?

20 bytes

Question #2 [10pts]: What are the corresponding byte values(**in hex)** store in RAM on a Little Endian Machine

12AA12AA 616200 F3FFFFFF0A000000 ACF0 616263

Question #3 [5pts]: Which of the five label declarations (A, B, C, D, E) would lead to different byte values in memory on a Big Endian machine?

A, C, D

Question #4 [5pts]: Consider the following code fragment (with numbered instruction), assuming the above .data segment

1) mov eax, [B]
2) mov ebx, C
3) add ebx, 6
4) mov [ebx], eax
5) mov cx, [D]

What is the value stored in cx at the end of the program's execution? Show your work by briefly saying what happens for each instruction

1) eax = F3006261

2) ebx points to C

3) ebx points 6 bytes past C

4) write eax to that address:

12AA12AA 616200 F3FFFFFF0A006162 00F3 616263

5) cx = F300

Exercise #2: Arithmetic [9pts]

Consider the following fragments of assembly code. For each, indicate whether the Carry bit (CF) is 0 or 1, and whether the Overflow bit (OF) is 0 or 1, at the point where the arrow points. Also indicate the decimal number printed to the screen by the last two instructions in each fragment.

Code fragment #1:

```
        mov    al, 072h
        mov    bl, 081h
        add    al, bl
                    ⇐          OF=_0___    CF=_0__
        movsx eax, al
        call print_int         Output =__-13__
```

Code fragment #2:

```
        mov    al, 0B3h
        mov    bl, 072h
        add    al, bl
                    ⇐          OF=_0__     CF=__1_
        movsx eax, al
        call print_int         Output =__37__
```

Code fragment #3:

```
        mov    al, 08Eh
        mov    bl, 092h
        add    al, bl
                    ⇐          OF=__1_     CF=__1_
        movsx eax, al
        call print_int         Output =_32_
```

Exercise #3: If-then-else [10pts]

```
if ((ecx > 4) && (ebx == 12)) {
     eax = 42;
} else {
     eax = eax + 1;
}
```

Write a fragment of assembly code (not a whole program with segment declarations, start-up/clean-up code, etc.) that is a translation of the high-level code above. Conveniently the variables have been given the names of the x86 registers that you should use to store the corresponding values. All values are **signed**. *(The table of jump instructions is on the last page of this exam in case you need it.)*

```
        cmp   ecx, 4
        jng   else
        cmp   ebx, 12
        jnz   else
        mov   eax, 42
        jmp   end
else:
        inc   eax
end:
```

Exercise #4: Jump Instructions [10pts]

The following fragment of code go through bytes in memory, starting with the byte stored at label L1 and ending with the byte stored at label L2 (included). The program interprets each byte as an 8-bit signed number. It counts the how many of these numbers are even, and stores this count at label count_even, and it counts how many of these numbers are larger than or equal to -10 but strictly lower than 20, and stores this count at label count_range. The program then prints out the largest of the two counts. Both counts are unsigned numbers.

In the listing below, all branch/jump instructions have been written as jxxx_1, jxxx_2, jxxx_3, etc. For each indicate the actual x86 jump/branch instruction so that the program operates as described above. There are multiple options for each, just pick one. Remember that addresses are always considered unsigned numbers. *(The list of all jmp instructions is provided on the last page of this exam if you need it.)*

```
            ...
                mov         dword [count_even], 0
                mov         dword [count_range], 0
                mov         ebx, L1
mainloop:       mov         dl, [ebx]

                cmp         dl, -10
                jxx_1       skip1
                cmp         dl, 20
                jxxx_2      skip1
                inc         dword [count_range]
        skip1:
                movzx       ax, dl
                mov         cl, 2
                idiv        cl
                cmp         ah, 0
                jxxx_3      skip2
                inc         dword [count_even]
        skip2:
                inc         ebx
                cmp         ebx, L2
                jxxx_4      mainloop

                mov         eax, [count_even]
                cmp         eax, [count_range]
                jxxx_5      skip3
                mov         eax, [count_range]
        skip3:
                call    print_int
                call    print_nl
```

Here are all valid answers:

jxx_1:
- must be one of: jl, jnge
  - must be a signed operation because it deals with array elements
  - if dl < -10, then we're out of the range, and must skip incrementing count_range.
  - So it's a "strictly lower than" (jl) or "not greater or equal than" (jnge) branch.

jxx_2:
- must be one of: jge, jnl
  - must be a signed operation because it deals with array elements
  - if dl >= 20, then we're out of the range, and must skip incrementing count_range.
  - so it's a "greater or equal than" (jge) or "not strictly lower than" (jnl) branch.

jxx_3:
- must be one of jnz, jne, jg, ja
  - If the remainder is non-zero (jnz, jne) then we skip incrementing count_even.
  - If the remainder is >0 (ja, jg) then we skip incrementing count_even
    - both signed and unsigned will work, because we know a remainder is positive

jxx_4:
- must be one of jbe, jna
  - must be an unsigned operation because it deals with addresses
  - if ebx <= L2 then we must branch fo loop again (byte [L2] must be included)
  - so it's a "below or equal" (jbe) or a "not strictly above" (jna) branch.

jxx_5:
- must be one of jae, jnb, ja, jnbe, jg, jnle, jge, jnl
  - Can be signed or unsigned because it's a count, and we know it's positive
  - we must jump if count_even is the maximum of the two counters
  - count_even is the maximum if either:
    - eax >= count_range (jae, jge)
    - eax > count_range (ja, jg)
    - !(eax <= count_range) (jnbe, jnle)
    - !(eax < count_range) (jnb, jnl)

Exercise #5: Mystery Program [5pts]

Explain in **one English sentence** what the following program does (e.g., “this program computes the sum of two numbers”). DO NOT describe the operation of the program (e.g., "first the program sets eax to 12, then the program adds the content of memory at address L1 to eax, etc."). *Write your answer inside the box to the right of the code.*

```
%include "asm_io.inc"

segment .data
        msg1      db    "> ", 0
        msg2      db    "Results:", 0

segment .text
        global  asm_main

asm_main:
        enter   0,0     ; setup
        pusha
        ;;;;;;;;;;;;;;;;;;;;;;

        mov     eax, msg1
        call    print_string
        call    read_int
        mov     ebx, eax
        mov     ecx, eax
        mov     dl, 9
for:
        mov     eax, msg1
        call    print_string
        call    read_int
        cmp     eax, ebx
        jnl     nope
        mov     ebx, eax
nope:
        cmp     eax, ecx
        jl      nope2
        mov     ecx, eax
nope2:
        dec     dl
        jnz     for
        mov     eax, ecx
        sub     eax, ebx
        call    print_int
        call    print_nl

        ;;;;;;;;;;;;;;;;;;;;;;
        popa                    ; cleanup
        mov     eax, 0
        leave
        ret
```

This program prompts the user for 10 numbers and prints the difference between the largest and the smallest entered integers

Exercise #6: Dividing [Extra Credit: 5pts]

Write a fragment of assembly code (not a whole program with segment declarations, start-up/clean-up code, etc.) that reads in a signed integer from the keyboard and sets dl to 1 if the entered integer is divisible by 30, and does nothing otherwise

```
        call    read_int
        mov     edx, 0
        idiv    30
        cmp     edx,0
        jnz     end
        mv      dl, 1
end:
```

cmp x, y			
signed		unsigned	
Instruction	branches if	Instruction	branches if
JE	x = y	JE	x = y
JNE	x != y	JNE	x != y
JL, JNGE	x < y	JB, JNAE	x < y
JLE, JNG	x <= y	JBE, JNA	x <= y
JG, JNLE	x > y	JA, JNBE	x > y
JGE, JNL	x >= y	JAE, JNB	x >= y