AUBURN
UNIVERSITY

COMP 3350-002 / Fall 2014 / J. Overbey

COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE PROGRAMMING

– EXAM 2 –

FORM A

Form B similar (different order)

- **Calculators, phones, tablets, and other electronic devices are prohibited.**
- This is a closed-book, closed-notes exam.
- Do not begin working until you are instructed to do so.
- You will have 50 minutes to complete this exam.
- Fill in your answers in the space provided.

Name: SOLUTIONS

Problem	Score	Possible
1		8
2		3
3		10
4		18
5		14
6		25
7		10
8		6
9		6
Total:		**100**

1. (*8 points total; 2 points each*) For each sequence of instructions, indicate whether or not the conditional jump will be taken.

0 or 2 points, no partial credit

a.
```
mov al, -2
cmp al, 3
jle label
```
signed
● Will jump to *label* ○ Will not jump

b.
```
mov al, -2
cmp al, 3
jb label
```
← -2 is FEh, which also represents 254
unsigned – treated as 254 < 3
○ Will jump to *label* ● Will not jump

c.
```
mov al, 255
add al, 1
jo label
```
> This will set CF but not OF
○ Will jump to *label* ● Will not jump

d.
```
mov al, -3
test al, 80h
jnz label
```
-3 is FDh = 11111101 b
& 10000000
10000000
↑ Test if sign bit is set
● Will jump to *label* ○ Will not jump

2. (*3 points*) Consider the following.

```
mov al, 1
cmp al, 0       ; After this, flags are: CF = 0, SF = 0, ZF = 1, OF = 0
jg label
```

The `jg` instruction will jump to *label*, since 1 > 0. In a sentence, briefly but precisely explain *how* the `jg` instruction determines whether or not to jump.

It jumps when SF = OF and ZF = 0
↑ 2 points if you got this

3. (*10 points*) Suppose a procedure:

1 point for most other partially correct answers

- Receives one stack argument.
- Has a prologue that issues `enter 4,0` and then pushes ESI.

2 points per entry
1 point if an entry was present but in the wrong slot

The stack frame for this procedure consists of five 4-byte values (suppose they are stored in the memory addresses shown below). In a word or two, describe what is stored in each 4-byte entry of the stack frame.

0013FF6Ch	Argument	
0013FF68h	Return Address	
0013FF64h	Saved Value of EBP	←EBP
0013FF60h	Local Variable	
0013FF5Ch	Saved Value of ESI	←ESP

4. (*18 points total; 3 points each*) Consider the following data section.

```
.data
START = $
array DWORD 11223344h, 55667788h, 9900AABBh
```

3 points each
No partial credit

Suppose START is equal to **00405000h**. What is the value of EAX after each of the following instruction sequences executes? Write your answers in *hexadecimal*.

a. `mov eax, OFFSET array` ; EAX = 00405000h

b. `mov eax, LENGTHOF array` ; EAX = 3

c. `mov eax, array` ; EAX = 11223344h

d. `mov eax, [array + 2]` ; EAX = 77881122

e. `movzx eax, BYTE PTR [array]` ; EAX = 44h

f. `lea eax, [array + 2]` ; EAX = 00405002h

5. (*14 points*) Suppose a program contains the following .data section.

```
.data
array DWORD 11223344h, 55667788h, 9900AABBh
```

7 points for the right memory operand + increment

You want to display the values in the array, one per line:

```
11223344
55667788
9900AABB
```

7 points for comparison + loop control

Fill in the missing instructions to do this.

```
    mov ecx, 0                          ; ECX will count up from 0
top: mov eax, [ array + 4*ecx ]         ; Load the next array element
    call WriteHex                       ; Display that value...
    call Crlf                           ; ...followed by a newline
    add ecx, 1                          ; Increase ECX
    cmp ecx, 3                          ; Are we done?
    jb top                              ; Jump back to show next element
```

← SIZEOF DWORD

OR

array + ecx

4 ← SIZEOF DWORD

12 ↖ SIZEOF array

↖ LENGTHOF array

or jl or jne

6. *(25 points)* Fill in the code for a procedure `remainder` that:

- Uses the STDCALL calling convention.
- Receives two 32-bit, signed integers as arguments on the stack.
- Returns the remainder when the first argument is divided by the second.
- Does not appear (to the caller) to modify any registers except EAX.
- Creates and destroys the stack frame **without** using `enter` or `leave`.

5 points per section

0, 3, or 5 given (wrong, partially correct, or fully correct)

This is roughly similar to the following C/C++ function:

```
int __stdcall remainder(int a, int b) {
    return a % b;
}
```

```
remainder PROC
    ; Prologue (do not use enter)
        push ebp
        mov ebp, esp        ← No local variables
        push ebx
        push edx            ← CDQ will set EDX, so push that too

    ; Load the arguments from the stack into registers
        mov eax, [ebp+8]    ← Put first arg (dividend) into EAX
        mov ebx, [ebp+12]

    ; Perform signed, integer division
        cdq
        idiv ebx            ← Remember: only one operand

    ; Move the remainder into the correct register to return that value
        mov eax, edx

    ; Epilogue (do not use leave)
        pop edx
        pop ebx
        pop ebp             ← No local variables to remove
        ret 8               ← Two args, STDCALL
remainder ENDP
```

Bits numbered 

7. (*10 points total; 2 points each*) Give a single, **bitwise** instruction that:

   a. Sets bit 2 of the value in AL. or al, 100b

   b. Clears bits 4–5 of the value in AL. and al, 11001111b

   c. Flips the low 4 bits of the value in AL. xor al, 00001111b

   d. Sets AL to 0. xor al, al (OR) and al, 0

   e. Multiplies the signed integer value in AL by 4. sal al, 2

   " >>2

8. (*6 points; 1 point per blank)* Suppose x represents an **8-bit** signed integer value. In this problem, you will analyze the value computed by the formula

$$((x \,\&\, 1) \ll 7) \gg^s 7.$$

Fill in the blanks. You may use decimal, hexadecimal, or binary.

First, consider the term x & 1:

$$(x \,\&\, 1) = \begin{cases} \text{1} & \text{if } x \text{ is odd} \\ \text{0} & \text{if } x \text{ is even.} \end{cases}$$

A 7-bit left shift shifts the LSB into the sign bit, so

$$(x \,\&\, 1) \ll 7 = \begin{cases} \text{80h} & \text{if } x \text{ is odd} \\ \text{0} & \text{if } x \text{ is even.} \end{cases}$$

A 7-bit arithmetic right shift copies the sign bit into all 8 bits, so

$$((x \,\&\, 1) \ll 7) \gg^s 7 = \begin{cases} \text{-1} & \text{if } x \text{ is odd} \\ \text{0} & \text{if } x \text{ is even.} \end{cases}$$ ■

9. (*6 points total; 2 points each)*

   a. If 31h is the underlying integer representing a Q5.2 fixed-point value, what is the value it represents, *in decimal?*

   31h = 00110001b $1100.01_2 = 12\frac{1}{4}$

   b. Round the value 101.110_2 to the nearest $1/2$, writing the result in binary.

   $+\ 1$
   110.000 110.0_2

   c. To multiply two Q*m*.*f* fixed-point values a and b, we gave the formula $((a \times b) + (1 \ll (f-1))) \gg^s f$. Briefly explain (in a sentence) why the f-bit arithmetic right shift is necessary.

The product a × b has twice as many fractional bits (2f) as a and b. The right shift moves the binary point into the correct position, truncating/removing the extra fractional bits.