# CS278 EXAM 2 CONCEPTUAL (40% of total Exam 2 Score)

# NAME: Aaron Borjas

**INSTRUCTIONS**:

* **Fill out your name and answers for this exam, and email a copy to Matt on the due date.**

You are allowed to look at any of your previous programming solutions, your textbook, or any of the PowerPoint slides on blackboard while working on this problem. **You must write answers in your own words and for the short answer questions you must** **cite where you found your answer (lecture/slide number) or (Book/page)**.

* You are allowed to write programs (and debug them) to confirm your answers.
* Since this is a take-home exam you will need to work on the honor system. If we determine that you cheated by talking to other students and/or copying answers you will receive a 0 on BOTH portions of this exam.
* **Honor Statement: I affirm that all my answers were determined solely by me and I did not copy or talk to other students about this exam:**

**Name: Aaron Borjas Date: 4/18/20**

## Fill in the Blanks and Short Answer

1. (1 pts) The **movzx** instruction **zero-extends** a smaller operand into a larger operand.

1. (1 pts) Assume the **AL** register contains 127 (decimal). Will the instruction **ADD AL, 3** set the **Zero** flag?

Explain your answer here: No because AL is an 8-bit register, and an 8-bit register can hold a value up to 255 (decimal)

1. (1 pts) The **PUSH** instruction will **decrement** the **stack pointer** by 4 if the argument to push is a 32 bit register or memory.

1. (1 pts) The popad instruction pushes all the 32-bit general-purpose registers off of the stack and back into the actual registers.
2. (1 pts) The **Zero flag** is set when the result of an arithmetic operation equals zero.

1. (1 pts) The **Carry flag** is set when an instruction generates a result that is too large (or too small) for the destination operand.

1. (1 pts) The **Sign flag** is set if the destination operand is negative, **and** it is not set if the destination operand is positive.

1. (1 pts) The **Overflow flag** indicates if the result of a signed arithmetic operation is too large or too small to fit into the destination.
2. (1 pts) The call instruction transfers control to a subroutine (i.e. procedure) and   
     
   the ret returns control back.

1. (1 pts) What three Irvine32 library functions do you need to use if you want to use a program to write data to a file?

WriteToFile, OpenInputFile, CloseFile

## Fill in the Blanks and Short Answer

1. (5 pts) For each instruction given below,

a. Describe if the instruction is **legal** or **illegal** (**if illegal, describe why**).   
 b. If the instruction is legal, place the **hexadecimal value** of the destination register in the space provided.

**.data**

**byte1 BYTE 0FFh,1,2**

**byte2 BYTE 14h**

**word1 WORD 0FFFFh,1,2**

**word2 WORD 3**

**word3 SWORD 7FFFh,8000h**

**word4 SWORD 9000h**

**dword1 DWORD 10h,20h,30h,40h**

**dArray DWORD 10 DUP(?)**

**.code**

**main PROC**

**mov ax,word2**

**dec ax** ; a. 2h

**neg ax** ; b. FFFE

**mov ax,word1** ; c. FFFF

**neg ax**  ; d. 1h

**add ax,000Fh** ; e. 10h

1. (1 pts) Write a one line instruction that moves the **most significant 8 bits** of **dword1** into  
    the **BL** register

mov bl, byte ptr [dword1]

1. (3 pts) Indicate the **decimal value** of the destination operand AX:

**.data**

**dArray DWORD 10 DUP(?)**

**.code**

**main PROC**

**mov ax, TYPE dArray** ; a. 4

**mov ax, LENGTHOF dArray** ; b. 10

**mov ax, SIZEOF dArray** ; c. 40

1. (1 pts) Will the following code jump to the label **L1**? Why or Why Not? No it will not. xor bl, bl always evaluates to 0, and jnz only jumps when the condition is *NOT* zero

xor bl,bl

jnz L1

## Short Answer Problems

1. (2 pts) Convert the following two lines of C++ into the equivalent assembly language statements.

**int dArray[]={0,1,2};**

**dArray[0] = dArray[1] + dArray[2];**

dArray DWORD 0, 1, 2

mov al, [dArray+4]

add al, [dArray+8]

mov [dArray], al

1. (2 pts) Translate the following 3 lines of C++ code into equivalent assembly language statements.  
    Assume **EAX** and **EBX** are set randomly and are both integers. **Do NOT use any assembly macros in your code**:

ECX = 0;

if ( EAX < EBX )

{ ECX = 1;

}

cmp eax, ebx

jnc label1

label1:

mov ecx, 1

## Fill In the Blank

1. (2 pts) F What is the final **hexadecimal** value of **AL** after both mov’s complete? (HINT: Remember that memory for variables are laid out next to each other. So the memory for word4 occurs directly after the memory for word3.)

.data

**byte1 BYTE 0FFh,1,2**

**byte2 BYTE 14h**

**word1 WORD 0FFFFh,1,2**

**word2 WORD 3**

**word3 SWORD 7FFFh,8000h**

**word4 SWORD 9000h**

**dword1 DWORD 10h,20h,30h,40h**

**dArray DWORD 10 DUP(?)**

**.code**

**main PROC**

**mov ebx,OFFSET byte1**

**mov al,[ebx+5]**

1. (2 pts) \_\_\_\_\_\_20h\_\_\_\_\_\_\_What is the final **hexadecimal** value of **EAX** after both mov’s complete?

**.data**

**byte1 BYTE 0FFh,1,2**

**byte2 BYTE 14h**

**word1 WORD 0FFFFh,1,2**

**word2 WORD 3**

**word3 SWORD 7FFFh,8000h**

**word4 SWORD 9000h**

**dword1 DWORD 10h,20h,30h,40h**

**dArray DWORD 10 DUP(?)**

**.code**

**main PROC**

**mov edx,2**

**mov eax,dword1[edx]**

1. (2 pts) 2020h\_\_\_\_\_\_\_\_\_\_What is the final **hexadecimal** value of **AX** after the loop completely finishes?

**.data**

**word1 WORD 1000h,2000h,3000h,4000h,5000h**

**dword1 DWORD 10000h,20000h,30000h,40000h**

**.code**

**main PROC**

**mov edx,OFFSET word1+6**

**mov ecx,2**

**mov ax,0**

**L1:**

**mov ax,[edx]**

**add ax,20h**

**sub edx,4**

**loop L1**

1. (2 pts) 10000h What is the final **hexadecimal** value of **EAX** after the **xor** instruction executes:

**.data**

**word1 WORD 1000h,2000h,3000h,4000h,5000h**

**dword1 DWORD 10000h,20000h,30000h,40000h**

**.code**

**main PROC**

**mov eax, dword1[8]**

**mov edx, dword1[4]**

**xor eax,edx**

## Short Answer

1. (2) Describe the purpose of the **stack area** of a program’s memory.

The stack of a programs memory keeps track of stuff relating to subroutines and procedures. Usually the stack is accessed with the ESP register. However, the user mainly interacts with ESP through the call, ret, push, and pop instructions.

1. (2) Describe **what happens to the stack** when the **CALL statement** executes.

When the call statement is used, it pushes its return address on the stack and then proceeds to take the called procedure’s address and put it in the instruction pointer.

1. (2) Describe **what happens to the stack** when the **RET statement** executes.

When the ret statement is used, it pops the return address from the stack (that was put there by the call statement) into the instruction pointer.

1. (2) Describe the **purpose of table-driven selection**. What control structure would this correspond to in a language like C++ or java?

The point of table-driven selection is to replace multiway selection structure, or if/else statement type code. This would most likely correspond to a switch statement in c++ or java.