CPSC 240
Computer Organization & Assembly Language Programming

Spring 2021
Midterm 1

Name: ______________________________

Section	Points
I. Math Problems	/42
II. General Knowledge (SA)	/40
III. General Knowledge (MC)	/14
IV. Short Programs	/54
Total:	**/150**

Instructions:

1. This is an open-book, open-notes, open-internet exam
2. You have two hours for the exam
3. Calculators are allowed
4. Communication with classmates is NOT allowed

Notes:

- If you write the word *"blank"* in the space for the answer, you will automatically receive 25% of the credit for that question
- You may use scratch paper, however, it does not need to be submitted with the test
- Best of luck, this is a simulation so don't feel too pressured :)

I. Math Problems - Short Answer (3 points each)

For each question in this section, there is **one** best possible answer. Calculators are allowed but not suggested. Show all work if applicable.

1. Convert 34⅓ to a 64-bit IEEE-754 number. Answer in Hex.
   1 sign, 11 for the exponent, 52 for the significand

Whole number	34 34/2 = 17 remainder 0 17/2 = 8remainder 1 8/2 = 4 remainder 0 4/2 = 2 remainder 0 2/2 = 1 remainder 0 ½ = - remainder 1 100010
fraction	⅓(can we keep it as a fraction? If kept as fraction ⅓*2=⅔ ⅔*2= 1and ⅓ ⅓*2=⅔ and so on same thing) .33333 * 2 = 0.66666 .66666 * 2 = 1.33333 0.3333 * 2 = 0.66666 .66666 * 2 = 1.33333 .010101
Binary	+100010.01010101010101010101010101010101 //.01 repeats 52 times
Scientific	1.000100101… *2^5
Stored Exponent	011 1111 1111 = 1023 000 0000 0101 = 5 -------------------- 100 0000 0100 =1028
IEEE 754 Binary	0100 0000 0100 0001 0010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010
IEEE 754 Hex	0100=4 0000=0 0100=4 0001=1 0010=2 1010 = A 0x4041 2AAA AAAA AAAA 0x4041 2AAA AAAA AAAA (si)

0x4041 2AAA AAAA AAAA

2. Convert -94.09375 to a 64-bit IEEE-754 number. Answer in Hex.

   0xC057 8C00 0000 0000

Whole number	-94 94 / 2 = 47, 0 47 / 2 = 23, 1 23 / 2 = 11, 1 11 / 2 = 5, 1 5 / 2 = 2, 1 2 / 2 = 1, 0 1 / 2 = 0, 1 -1011110
fraction	.09375 0.09375 * 2 = 0.1875 0.1875 * 2 = 0.375 0.375 * 2 = 0.75 0.75 * 2 = 1.5 0.50 * 2 = 1.00 .00011
Binary	-1011110.00011
Scientific	-1.01111000011 * 2^6 (true exp = 6)
Stored Exponent	011 1111 1111 = 1023 (bias) 000 0000 0110 = 6 -------------------- 100 0000 0101 = 1029
IEEE 754 Binary	1100 0000 0101 0111 1000 0110 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
IEEE 754 Hex	0xC057 8600 0000 0000 (ours) 0xC057 8600 0000 0000 (si)

3. Convert 0.95 to IEEE-754 32-bit hex.
   1 + 8 + 23 = 32
   8 bits = 127 (bias)

Whole number	0
fraction	.95 * 2 = 1.9 .9 * 2 = 1.8 .8 * 2 = 1.6 .6 * 2 = 1.2 .2 * 2 = 0.4 .4 * 2 = 0.8 .8 * 2 = 1.6 .6 * 2 = 1.2 .2 * 2 = 0.4 .4 * 2 = 0.8 .111100110011001100 repeating 1100
Binary	0.111100110011001100 repeating 1100
Scientific	+1.11100110011001100 x 2^-1
Stored Exponent	(2^8-1)= 128 1000 0000 128-1=127 0111 1111 (bias) 127-1=126 126 in binary can do another way(you can ignore this) 126/2 =63 remainder 0 63/2= 31 remainder 1 31/2=15 remainder 1 15/2=7 remainder 1 7/2=3 remainder 1 3/2= 1 remainder 1 ½=0 remainder 1 Need one more bit so 0 0111 1110
IEEE 754 Binary	0011 1111 0111 0011 0011 0011 0011 0011
IEEE 754 Hex	0x3F73 3333 0x3F73 3333 (eric) Im good?

4. Convert 0x3FB9 0000 0000 0000 to a decimal floating-point number. The answer should be in fraction notation.

Hex	0x3FB9 0000 0000 0000
Binary	0011 1111 1011 1001 0000 0000 0000 0000 … 0000
Binary	0011 1111 1010 1001 0000 0000 0000 0000 … 0000
Exponent	011 1111 1010 (stored exponent) = 1019 011 1111 1111 (bias) = 1023 -- 000 0000 0100 = -4 = 1019 - 1023
Significand	1001
Scientific	1.1001 * 2 ^ -4
Fractions Method 1	1 0.1 = 1 * 2^-1 = ½ 0.00 = 0 * 2^-2 = 0 * ¼ 0.00**0** = 0 * ⅛ 0.0001 = 1 * 1/16 = 1/16 (1 + ½ + 0 + 0 + 1/16) * 1/16 1/16 + 1/32 + 1/256 16/256 + 8/256 + 1/256 = **25/256**
Fractions Method 2	1.1001 * 2 ^ -4 11001 * 2 ^ -8 11001. * 2 ^ -8 (16*1) +(8*1)+ (4*0)+ (2*0)+(1*1) (16 + 8 + 1)/(256) = **25/256**

5. What is positive infinity in 64-bit IEEE-754 format? Answer in Hex.

Hex	
Binary	0x7FF0 0000 0000 0000 0111 1111 1111 0000 ...

6. What is negative infinity in 64-bit IEEE-754 format? Answer in Hex.

Hex	0xFFF0 0000 0000 0000
Binary	1111 1111 1111 0000 ...

7. Convert -985 to hex suitable for 32-bit registers.

Convert to Binary	985/2 = 492 r 1 30/2 = 15 r 0 492/2 = 246 r 0 15/2 = 7 r 1 246/2 = 123 r 0 7/2 = 3 r 1 123/2 = 61 r 1 3/2 = 1 r 1 61/2 = 30 r 1 ½ = 0 r 1
Binary	0000 0000 0000 0000 0000 0011 1101 1001
1’s complement done because negative 985	1111 1111 1111 1111 1111 1100 0010 0110
2’s complement done because negative 985	1111 1111 1111 1111 1111 1100 0010 0111
Hex	0xFFFF FC27

8. Convert -3212 to 64-bit integer hex form. Show the final answer in little-endian (for people who naturally read from right to left)

Convert to Binary	3212/2 = 1606 r0 1606/2 = 803 r0 803/2 = 401 r1 401/2 = 200 r1 200/2 = 100 r0 100/2 = 50 r0 50/2 = 25 r0 25/2 = 12 r1 12/2 = 6 r0 6/2 = 3 r0 3/2 = 1 r1 ½ = 0 r1
Binary	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1100 1000 1100
1's complement	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 0011 0111 0011
2's complement	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 0011 0111 0100
Hex (big endian)	0xFFFF FFFF FFFF F374
Hex (little endian	0x74F3 FFFF FFFF FFFF

9. Suppose you are attempting to convert 3932 to binary. What would the third step look like?

3932 / 2 = 1966, remainder = 0
1966 / 2 = 983, remainder = 0
983 / 2 = 491, remainder = 1

10. Calculate 0x00402A43 - 0x0023E07F in hex.

Subtraction	Addition with 2’s Complement
0x00402A43 -0x0023E07F	0x00402A43 -0x0023E07F
3F 93 0x00402A43 -0x0023E07F ----------- 1C49C4	0x00402A43 +0xFFDC1F81 -----------
	11 1 0x00402A43 +0xFFDC1F81 ----------- 1001C49C4 001C49C4

11. Represent 0xFFFF FFFF 075C 43DE in little endian.

Big endian	0xFFFF FFFF 075C 43DE
Little Endian	0xDE43 5C07 FFFF FFFF

12. What is the smallest signed integer in 32-bit two's complement? Answer in Hex.

Binary	1000 0000 0000 0000 0000 0000 0000 0000
Hex	0x8000 0000
Decimal	-2^31?

Spring 2020 Final written section professor answered -2^31 for smallest signed integer

13. What is the smallest twos-complement 32-bit integer? Give your answer in decimal.

SI leader will check on this one

Binary	1000 0000 0000 0000 0000 0000 0000 0000
Hex	0x8000 0000
Decimal	-2^31?

14. What is the hex representation of the largest signed integer in a 48-bit register?

Binary	0111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 47 ones (48 - 1)
Hex	0x7FFF FFFF FFFF
Decimal	2^48 - 1

II. General Knowledge - Short Answer (2 points each)

For each question in this section, clearly indicate your final answer.

1. According to the professor, what is the Opus Magnus of all assembly programming?

   Rollercoaster Tycoon 2

2. According to the professor, who is the King of Assembly?

   Chris Sawyer

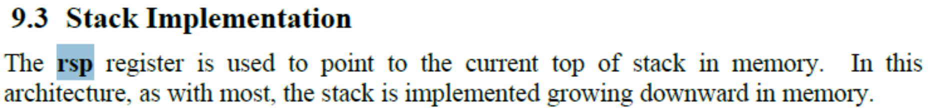
3. What is the full URL leading to the complete legal document written in ASCII text used in the software license promoted in this course for application software?

   `https://www.gnu.org/licenses/`

4. What is the official name of the Linux shell available for Windows 10 machines?
   WSL or Windows subsystem for Linux

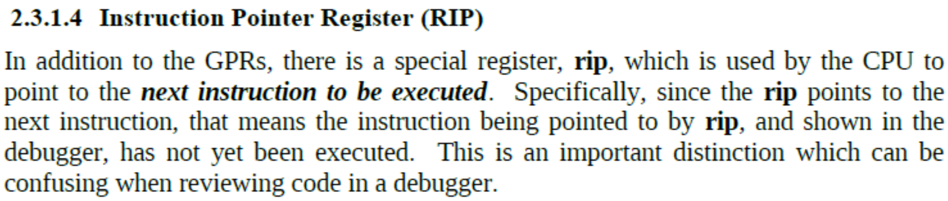
5. What is the backend boundary and what is the frontend boundary of the activation record? Specify each part of the answer with “back” or “front”.

   Back is rbp, front is rip(?) (next instruction)



9.3 Stack Implementation

The **rsp** register is used to point to the current top of stack in memory. In this architecture, as with most, the stack is implemented growing downward in memory.



2.3.1.4 Instruction Pointer Register (RIP)

In addition to the GPRs, there is a special register, **rip**, which is used by the CPU to point to the ***next instruction to be executed***. Specifically, since the **rip** points to the next instruction, that means the instruction being pointed to by **rip**, and shown in the debugger, has not yet been executed. This is an important distinction which can be confusing when reviewing code in a debugger.

6. Explain precisely what cdqe does.

   Converts a doubleword (32-bit value) in the EAX register and turns it into a quadword (64-bit value) in RAX register.

   ```
   mov eax, r7 ; EAX = r7
   cdqe ; RAX = r7
   ```

   //taken from the answer key of a previous exam, the above is also correct.
   Non Technical Answer: The lower half of rax is called eax. Eax has a numeric integer value such as say 7. The job of cdqe is to fill the high half of rax with new bits so that the the integer value of rax is the same, say still equal to 7.

7. A C++ (or C) function has declared float weights[40]; Output the contents of the first 12 dwords of the array in decimal.

   ```
   p/d (long[12])weights
   ```

8. Show the contents of memory starting at 0x0000 7FFF FFFF 8800 and continuing for the next 18 qwords showing the contents of each qword as an unsigned long.

   ```
   x/18ug 0x00007FFFFFFF8800
   ```

9. Change the dword at memory address 0x0000 0FFF 8000 4400 to be 330

   ```
   set  {int} 0x00000FFF80004400 = 330
   ```

10. Change the qword on top of the stack to be -2.

    ```
    Set {long} $rsp = -2
    ```

11. What is the rule that determines if a written number or a stored number is in Little Endian format?

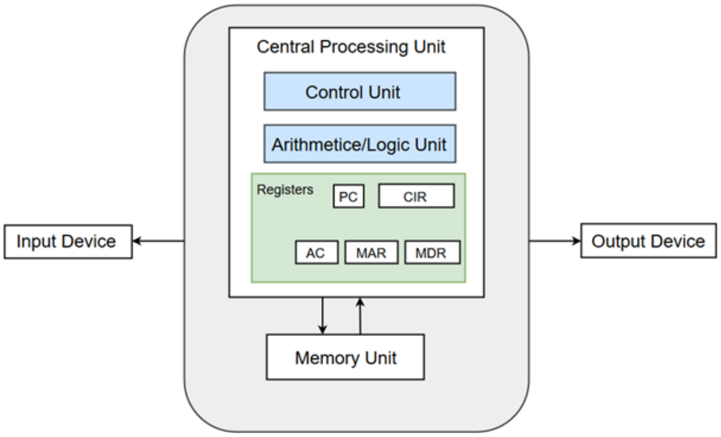
“little end” least significant byte/value (or smallest) is stored first.

12. What are the names (or acronyms) of the “components” of a modern X86 microprocessor?
GPR(r), FPU, SSE (xmm), AVX, ALU, Cache

13. Name all the components of a Von Neumann computer.

CPU, memory, Input/Output devices, arithmetic/logic unit, registers

Von-Neumann Basic Structure:



Processor

Primary storage (memory)

Secondary storage (drives)

Peripheral devices

Bus (communication).

From one of the midterms

14. In the normal order of things, what do we call the first byte of a big-endian number?

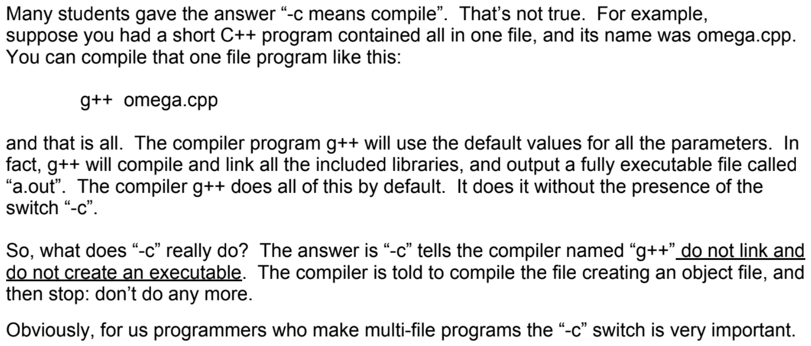
Most significant digit = MSD

15. What is the defining property of a "little-endian" when speaking of numbers?

Defining property of little-endian is that the Least significant digit is read first. (LSD)

16. In the C++ compilation command, there is a switch -c. What is the point of that switch?

-c compiles only; produces object (.o) files, but suppresses linking.
Answer: The "-c" instructs the compiler to "compile only".
If the "-c" is omitted the compiler attempts to "compile and link" in the same operation.
Added note from professor: The answer is "-c" tells the compiler named "g++" do not link and do not create an executable.



Many students gave the answer "-c means compile". That's not true. For example, suppose you had a short C++ program contained all in one file, and its name was omega.cpp. You can compile that one file program like this:

```
g++  omega.cpp
```

and that is all. The compiler program g++ will use the default values for all the parameters. In fact, g++ will compile and link all the included libraries, and output a fully executable file called "a.out". The compiler g++ does all of this by default. It does it without the presence of the switch "-c".

So, what does "-c" really do? The answer is "-c" tells the compiler named "g++" do not link and do not create an executable. The compiler is told to compile the file creating an object file, and then stop: don't do any more.

Obviously, for us programmers who make multi-file programs the "-c" switch is very important.

17. What happens if you include the S parameter in the compilation command of a C++ function.

removes all symbol information from output executable files. This option is passed to ld

18. Most modern computers have a single stack at one end of memory. As we move away from the stack to the other end of memory what is the first important block of data we encounter?

19. What is the name of the integer number system used in most modern computers?

    Binary

20. On the x86 architecture, which way does the stack grow? And the heap? What do the push/pop instructions do to the stack pointer?

21. What is the relationship between assembly language and machine language?

    Assembly is a simple translation to machine code. One human instruction in x86 such as movsd xmm13, xmm14 will generate one line of code. While if you use C++ one statement will multiple lines of machine code. Plus Assembly is essentially the Architecture of the machine, and enables us to learn the tools of the chain.

III. General Knowledge - Multiple Choice (2 points each)

For each question in this section, circle **1** answer. Choose the best answer.

1. Which of these has the highest address?
   a. .bss segment
   b. .data segment
   c. Operating system segment
   d. Cache memory

2. Which of these has a lower address than any of the others?
   a. .bss segment
   b. .data segment
   c. Executing Code ←
   d. Cache Memory

3. Which register would be used to pass the second functional argument in 64-bit Linux?
   a. rsi ←
   b. rdi

c. rax
d. r12

4. A DWORD represents how many bytes?
   a. 2 bytes
   b. ***4 bytes*** ←
   c. 8 bytes
   d. 16 bytes

5. Which jump instruction is equivalent to je?
   a. Jz ←
   b. jle
   c. jne
   d. js

6. Which register points to the current top of the stack?
   a. ***rsp*** ←
   b. rbp
   c. rsi
   d. rdx

7. What is the name of the section where the code is placed?
   a. .data
   b. ***.text*** ←
   c. .bss

IV. Short Programs - Short Answer (6 points each)

For each question in this section, show all of your work if applicable. Clearly indicate your final answer.

1. Your program contains a `printf` block like this:

```
mov rax, 0
mov rdi, computation
mov rsi, r8
call printf
```

While testing you discover a problem, `printf` changes your values stored in `rdx`, `r8`, and `r9`. At the moment there are no unused registers where `rdx`, `r8`, and `r9` can be backed up. Provide an example of how you can fix this issue.

push rdx
push r8

```
push r9
mov rax, 0
mov rdi, computation
mov rsi, r8
call printf
pop r9
pop r8
pop rdx
```

We may need another push/pop to keep stack alignment at 16 bytes

Stack needs to be even to stay in bound. Pop and push adds 8 bytes into and from the stack. And to be on the boundary it must be evenly divisible by 16. Thus having another push and pop should keep alignment.

2. Suppose there are signed integers stored in `r8` and `r9`. Complete the assembly code below such that the code fragment divides the numerator r8 by the denominator `r9` and output the statement: `“The quotient is 23”`

```
segment .data

Message db “The quotient is %ld”, 10, 0

segment .text


;presume r8 r9 had signed integers stored in them
```

div(unsigned)

idiv(signed)

```
mov rax, r8

cqo

idiv r9
```

```
mov r14, rax ;added later apparently gives quotient when there is a remainder.

mov r13, rdx ;gives remainder


push qword 0

mov rax, 1

mov rdi, Message

mov rsi, r13

call printf

pop rax
```

3. Consider the following code fragment. What are the contents of the `rax`, `rbx`, and `r14` registers after the code fragment has been executed? Show your work.

```
mov r15, 21
mov r14, 18

mov rax, r15  ;rax = 21
add r14, 2
add r15, 6
mov rbx, r14    ; rbx = 2
add rbx, 4      ; rbx = 4
mov rax, r15    ; rax = 6
```

rax = 6
rbx = 4
r14 = 2

4. Consider the following code fragment. What are the contents of the `rax` and `rbx` after the code fragment has been executed? Show your work.

```
mov rax, 4
mov rbx, 16
mov rbx, [rax]
mov [rbx], 56
```

-------------Memory-------------

ADDRESS	VALUE
000000	6543210
000004	5189784
000008	1698791
00000C	9816517
000010	9816875
000014	5498156

5. We want a function that copies the data of one array to a second array. Assume `a` and `b` are two arrays of long ints. Both arrays have the same size. Assume `s` is the number of valid data in `a`.
   a. What is the C++ prototype of a function that copies the first s numbers from `a` to `b`?

      void copy(long a[], long b[], long s);

   b. The following facts are given:

      books is an array of 200 quadwords declared in .bss
      documents is an array of 200 quadwords declared in .bss
      `r12` contains an integer, 0 <= `r12` < 200
      `r12` is the number of valid data in books

      Show the block of asm instructions for calling the function that will copy books to documents.

6. If the call to a function is always segfaulting what should the programmer try first to fix this issue?

   Find out where the seg fault may be happening and add push/pops to align stack to 16 bytes

7. Assume that oranges and grapes are arrays of size 100, also assume that they are already declared in the .bss section. You want to create an assembly code that will copy the first 30 values of oranges into the first 30 values of grapes. Translate the C++ code below to assembly.

```
for(int i = 0; i < 30; i++) {
      grapes[k] = oranges[k];
}
```

   mov r14, 30 ; holds the 30 size of oranges

   mov r13, 0 ;loop counter

   mov r12, grapes ;sets start of array of grapes to r12

   mov r15, oranges

```
begin_loop:
cmp r13,r14
jge finish_loop

mov r11, [r15 +8*r13]
mov [r12 + 8*r13], r11
 inc r13
jmp begin_loop
finish_loop:
rest of code till ret...
```