CS271 - Computer Architecture and Assembly Language

Midterm Exam Winter 2019

Name: ____ Student ID: ____________________________

Part 1, Multiple choice questions [10 points]

1) **(2 points)** Which register is used for holding the address of the top of the stack:
   a) EIP
   b) EBP
   c) ESP
   d) ECX

2) **(2 points)** Sort the following types of memory using numbers 1-4 with 1 been the slowest and 4 the fastest memory:
   a) Optical Disk _1__
   b) Registers _4__
   c) Main Memory _2__
   d) Cache _3__

3) **(2 points)** The four parts of a CPU are:
   a) address bus, registers, control unit, arithmetic logic unit
   b) data bus, memory unit, control unit, arithmetic logic unit
   c) clock, registers, control unit, arithmetic logic unit
   d) clock, memory unit, control unit, instruction fetch unit

4) **(2 points)** Which one from the number below is the binary representation of the hexadecimal number 1111?
   a) 0100011001110000
   b) 0001000100010001
   c) 0000100111100000
   d) 0101011001111000

5) **(2 points)** Hamming codes are used for:
   a) Representing floating point numbers in binary
   b) Hold the address of special purpose registers
   c) Represent phone area codes
   d) Detect single bit data errors

Part 2, Answer all the questions [40 points]

1) a) **(5 points)** Convert the decimal number -21.6562 to the IEEE754 **single precision** floating point format. Show the process.

Step 1 -> Find the sign: Positive = 1

Step 2 -> Convert the number to binary:

21 - 16 = 5	=> 1	0.6562 – 0.5 = 0.1562	=> 1
5 – 8 = x	=> 0	0.1562 – 0.25 = x	=> 0
5 – 4 = 1	=> 1	0.1562 – 0.125 = 0.0312	=> 1
1 – 2 = x	=> 0	0.0312 – 0.0625 = x	=> 0
1 – 1 = 0	=> 1	0.0312-0.0312 = 0	=> 1

21 = 10101 0.6562 = 0.10101

Together => 10101.101010000….000000

Step 3 -> Normalize & Find the real exponent: 1.0101101010000….000000 x 2^4

Step 4 -> Find exponent: 127 + 4 = 131 = 10000011

Step 5 -> ANSWER: 1 10000011 01011010100000000000000

b) **(5 points)** Show the hamming code of the 8-bit number 23 in **odd and even parity**. Show the progress

Step 1 -> Convert the number to binary: 23 = 00010111

Step 2 -> Calculate the parity bits: log8 + 1 => 3 + 1 = 4

Step 3 -> Construct the number without filling the parity bits: _ _ 0 _ 001 _ 0111

Step 4 -> ANSWER: **even parity:** 010000110111, **odd parity:** 100100100111

2) **(10 points)** Here is a partial “listing file” for a MASM program. Show the contents of the specified registers and the system stack before and after the execution of each statement. When a value gets replaced, lightly cross out the previous value (instead of erasing it). The first row is completed as a n example.

```
        main    PROC
0000    call    intro
0004    push    0047h
0008    call    getData
000C    pop     eax
0010
:
0017    exit
        main    ENDP
..............................................................................

        intro   PROC
:
0024    call    DoNothing
0028
:
002C    ret
        intro   ENDP
..............................................................................

        DoNothing       PROC

0030    ret

        DoNothing       ENDP
..............................................................................

        getData         PROC
:
0048    call            DoNothing
004C
0050    ret
        getData         ENDP
```

Address/ Instruction	EIP Before	EIP After	ESP Before	ESP After
0000 call intro	0000	0024	0500	04FC
0024 call DoNothing	0024	0030	04FC	04F8
0030 ret	0030	0028	04F8	04FC
002C ret	002C	0004	04FC	0500
0004 push 0047h	0004	0008	0500	04FC
0008 call getData	0008	0048	04FC	04F8
0048 call DoNothing	0048	0030	04F8	04F4
0030 ret	0030	004C	04F4	04F8
0050 ret	0050	000C	04F8	04FC
00B0 pop eax	00B0	0010	04FC	0500

System stack	
Memory address	Memory contents
04F4	~~004C~~
04F8	~~0028~~ ~~000C~~
04FC	~~0004~~ ~~0047~~
0500	xxxx

3) **(10 points)** Convert the signed numbers below.

Decimal	Binary	Hexadecimal
+2460	0000100110011100	099C
-1019	1111110000000101	FC05
-963	1111110000111101	FC3D

4) **(10 points)** Write MASM code to implement the given high-level pseudo-code. The *for* loops must be implemented with the *loop* instruction. The .data segment is given:

```
if (eax < 5) AND (ebx = 0)
{
    while(eax < 5)
    {
        print yes
        eax = eax +1
    }
}
else if (edx = 0)
{
    for(ecx = 0; ecx < 2; ecx++)
    {
        edx = edx + 1
        for(ecx = 0; ecx < 3; ecx++)
        {
            print no
        }
    }
}
else
{
    print yes
}
```

```
.data
            yes     BYTE            ”Yes”,0
            no      BYTE            ”No”,0
.code

cmp eax, 5
JGE Else_if
cmp ebx, 0
JNE Else_if
While1:
     cmp eax, 5
     JGE outofwhile
    mov OFFSET yes
    call WriteString
    inc eax
JMP While1
Outofwhile:
JMP Outofif
Else_if:
     cmp edx, 0
     JNE Else
    mov ecx, 2
Outloop:
    puch ecx
    inc edx
    mov ecx, 3
    inloop:
          mov OFFSET no
          call WriteString
          loop
     pop ecx
     loopl
     JMP Outofif
Else:
     mov OFFSET yes
     call WriteString
     JMP Outofif
Outofif:
```

Part 3, Exrta credits [10 points]

5) Find the coding and logical errors on the code below and explain how to fix them. Ten or above errors take full grade.

```
.data
    prompt1     BYTE        "Hi, can I have your name please?" ,0
    prompt2     BYTE        "Can I have your age please?" ,0
    prompt3     BYTE        "Can I have your score please?" ,12         SHOULD BE 0
    answer1     BYTE        "Our names are: " ,0
    answer2     BYTE        "Our ages are: " ,0
    answer3     BYTE        "Our score's average is: " ,0
    myName      BYTE        ”Elmer Fudd”,0
    yourName    BYTE 30     DUP(0)
    myAge       BYTE        45          SHOULD BE DWORD
    yourAge     DWORD       ?
    myScore     DWORD       7
    yourScore   QWORD       ?           SHOULD BE DWORD
    avrg        WORD        ?           SHOULD BE DWORD

.code
main        PROC
    mov     edx, OFFSET prompt1
    call    WriteString
    ret     CrLf                        SHOULD BE CALL
    mov     edx, yourName               NO OFFSET
    call    ReadString

    mov     edx, OFFSET prompt2
    call    WriteString
    call    ret                         WRONG INSTRUCTION
    call    ReadInt                     NO STORE OF EAX

    mov     edx, prompt3                NO OFFSET
    call    WriteString
    call    ReadInt
    mov     yourScore, eax

    mov     ecx, OFFSET answer1         WRONG REGISTER
    call    WriteString
    mov     ecx, OFFSET  myName         WRONG REGISTER
    call    WriteString
    mov     ecx, OFFSET yourName        WRONG REGISTER
    call    WriteString
    call    CrLf

    mov     edx, OFFSET answer2
    call    WriteString
    mov     eax, myAge
    call    WriteString                 WRONG PROCEDURE
    call    WriteString                 UNNECESSARY PROCEDURE
    call    CrLf

    mov     eax, myScore
```

```
      mov     ebx, yourScore                    SIZE MISMATCH
      add     eax, ebx
      mov     ebx, 2
      cdq
      div     ebx                               NO STORE OF EAX

      mov     edx, answer3                      NO OFFSET
      call    WriteString
      mov     eax, avrg                         SIZE MISMATCH
      call    WriteInt
      call    CrLf
main ENDP
END main
```