Nov. 3, 2007

COMPUTER ENGINEERING DEPARTMENT

COE 205

COMPUTER ORGANIZATION & ASSEMBLY PROGRAMMING

Major Exam I

Second Semester (071)

Time: 7:30-9:30 PM

Student Name : _ KEY ____

Student ID.   : ____

Question	Max Points	Score
Q1	50	
Q2	10	
Q3	15	
Q4	25	
Total	100	

Dr. Aiman El-Maleh

[50 Points]

(Q1) Indicate whether the following is **true** or **false**, and if it is false **correct** it (correct the answer and not the question):

(1) (True, **False**) The smallest (negative) number that can be represented using 16-bit 2`s complement in hexadecimal is FFFF and the largest positive number in hexadecimal is 7FFF.

The smallest (negative) number that can be represented using 16-bit 2`s complement in hexadecimal is **8000** and the largest positive number in hexadecimal is **7FFF**.

(2) (True, **False**) Assume that the CPU has just read a 64-bit instruction from the linear address 00404000H. Then, the linear address of the next instruction that this CPU is going to read is 00404002H.

The linear address of the next instruction that this CPU is going to read is **00404008H** =00404000H +8 since the size of the instruction is 8 Bytes=64bit/8.

(3) (True, **False**) The 8086 processor is a 16-bit machine with an address and data bus of 16 bits while the Pentium IV processor is a 32-bit machine with an address and data bus of 32 bits.

The 8086 processor is a 16-bit machine with an address bus of **20** bits and data bus of 16 bits while the Pentium IV processor is a 32-bit machine with an address bus of **36** bits and data bus of **64** bits.

(4) (True, **False**) For addition operations, an end carry out of the most significant bit indicates incorrect result for both signed and unsigned numbers.

An end carry out of the most significant bit indicates incorrect result for **unsigned** numbers.

(5) (**True**, False) With a 32-bit address bus and 64-bit data bus, the maximum memory size than can be accessed by a processor is 4GByte and the maximum number of bytes that can be read or written in a single cycle is 8 Bytes.

2^{32}=4GByte, 64/8=8 Bytes.

(6) (True, **False**) The addressing mode of the source operand in the instruction MOV AX, [ESI] is register addressing mode.

The addressing mode of the source operand is **register-indirect** addressing mode.

(7) (True, **False**) The addressing mode of the source operand in the instruction MOV EAX, offset MSG is direct addressing mode.

The addressing mode of the source operand is **immediate** addressing mode.

(8) (**True**, False) Assuming 8-bit representation of numbers, the binary number 10100100 is equal to -36 in sign-magnitude representation, -91 in 1`s complement representation, and -92 in 2`s complement representation.

(9) (True, **False**) Assuming variable Array is defined as shown below:

Array WORD 10h, 20h

The content of register AX after executing the instruction *MOV AX, Array+1* will be *20h.*

The content of register AX will be **2000h**.

(10) (True, **False**) The assembler allocates 17 bytes for the variable *Array* defined below:

Array WORD 5, 4 dup(2, 3 dup(0))

The assembler allocates (1+4*(1+3))*2=**17*2=34** bytes for the variable *Array.*

(11) (**True**, False) Assume that DS=12FF, CS=E6F0, ES=F135, SS=ABCD IP=0016, and SI=526F. Based on 16-bit addressing in real mode, the linear address of the next instruction to be fetched from memory is E6F16.

The linear address of the next instruction to be fetched from memory is CS*16+IP=E6F00+0016=E6F16.

(12) (True, **False**) Assuming that AL=FEh and BL=00h, the two instructions ADD AL, 1 and SUB BL, 1 produce the same result in registers AL and BL and the same effect on flags.

They produce the same result in AL and BL registers=FF, but they produce opposite results in the Carry and Auxiliary flags. Other flags will have the same values.

(13) (True, **False**) The instruction set architecture of a processor consists of its control unit, data path, memory, and the instruction set.

The instruction set architecture of a processor consists of its instruction set, **programmer-accessible registers** and memory.

(14) (**True**, False) Assume that AX=8111h and BX=F265h. Executing the instruction *ADD AX, BX* sets the overflow flag and the carry flag to 1, while it sets the sign flag, the zero flag, the parity flag, and the auxiliary flag to 0.

8111h + F265h=7376h
OF=1, CF=1, SF=0, ZF=0, PF=0, AF=0.

(15) (**True**, False) Assume that AX=8111h and BX=F265h. Executing the instruction *SUB AX, BX* sets the parity flag, the auxiliary flag, the carry flag and the sign flag to 1, while it sets the zero flag and the overflow flag to 0.

8111h-F265h=8111h + 0D9Bh=8EACh
OF=0, CF=1, SF=1, ZF=0, PF=1, AF=1.

(16) (True, **False**) Assume that AX=F0F0h. Executing the instruction *NEG AX* produces the result AX=0F0Fh.

It produces the result AX=**0F10**h.

(17) (True, **False**) Assume that AX=00FFh. Executing the instruction INC *AL* produces the result AX=0100h.

It produces the result AX=**0000**h.

(18) (**True**, False) Assume that AX=009Fh. Executing the instruction MOVSX *BX, AL* produces the result BX=FF9Fh.

(19) (**True**, False) Assuming that AX=4 and given the following definition of ARRAY:

ARRAY WORD 1, 2, 3

Execution the code below does not change the content of AX and changes the content of ARRAY to:

ARRAY WORD 3, 2, 1

```
XCHG AX, ARRAY[0]
XCHG AX, ARRAY[4]
XCHG AX, ARRAY[0]
```

(20) (**True**, False) After executing the code shown below, the content of register AX will be 32 and the content of register CX will be 0.

```
        MOV CX, 4
        MOV AX, 2
NEXT:
        ADD AX, AX
        LOOP NEXT
```

(21) (True, **<u>False</u>**) Given a magnetic disk with the following properties:

- Rotation speed = 7200 RPM (rotations per minute)
- Average seek = 8 ms, Sector = 512 bytes, Track = 200 sectors

The average time to access a block of 64 consecutive sectors is 13.5 ms.

Average access time= Seek Time + Rotation Latency + Transfer Time
Rotations per second=7200/60 =120 RPS
Rotation time in milliseconds=1000/120=8.33 ms
Time to transfer 64 sectors=(64/200)* 8.33=2.67 ms
Average access time=8 + 4.17 + 2.67 =**14.84 ms**.

(22) (True, **<u>False</u>**) As part of the instruction set architecture of the Pentium-IV processor, it has Six 32-bit general-purpose registers, Four 16-bit segment registers in addition to Processor Status Flags (EFLAGS), Instruction Pointer (EIP) and Instruction Register (IR).

As part of the instruction set architecture of the Pentium-IV processor, it has **Eight** 32-bit general-purpose registers, **Six** 16-bit segment registers in addition to Processor Status Flags (EFLAGS), and Instruction Pointer (EIP).

(23) (True, **<u>False</u>**) Assuming that the instruction execution cycle is composed of a five-stage pipeline as follows: Instruction Fetch, Instruction Decode, Operand Fetch, Instruction Execute, Result Writeback. Assume that each stage requires one clock cycle to complete. Then, executing 10 instructions using this pipeline will require 50 clock cycles.

Executing 10 instructions using this five-stage pipeline will require 5+10-1=14 clock cycles.

(24) (True, **<u>False</u>**) Assuming the following data segment, and assuming that the first variable X is given the linear address **00404000h**, then the linear address for variable Y will be **00404003h.**

```
.DATA
X       BYTE  10,11,12
ALIGN  2
Y       WORD  13
```

The linear address for variable Y will be **00404004h.**

(25) (True, **<u>False</u>**) Assuming the following data segment, the content of register ECX=0000000C after executing the instruction **MOV ECX, SIZEOF MSG.**

```
.DATA
MSG   WORD   10 DUP(0), 5, 10
```

The content of register ECX=**00000018.**

[
10 Points]

(Q2) Consider a program that has the following data segment assuming a flat memory model:

I	*EQU*	*255*
J	*BYTE*	*-1*
K	*WORD*	*I*

Indicate whether the following are valid **<u>Pentium</u>** instructions or not. **<u>If invalid, give the reason</u>**:

1. MOV AX, J+1

Invalid. Size mismatch as AX is word while J+1 is a byte.

2. MOV BH, offset J

Invalid. Size mismatch as BH is a byte while offset J is a 32-bit address.

3. MOV DS, I+1

Invalid. Not allowed to move a constant directly into a segment register.

4. MOV ES, K

Valid.

5. MOV [EBX], 1

Invalid. There is ambiguity as constants do not have size.

6. ADD [ECX], AL

Valid.

7. NEG [EAX]

Invalid. There is ambiguity as addresses do not have size.

8. MOV AH, AL+1

Invalid. There is no such addressing mode for incrementing the content of a register except when it is used in addressing.

9. MOVSX EAX, j

Valid.

10. DEC DS

Invalid. Segment registers cannot be used with arithmetic instructions.

[15 Points]

(Q3) Suppose that the following directives are declared in the data segment with a starting linear address of 00404000. Show the linear addresses of allocated memory and their corresponding content in hexadecimal. Note that the ASCII code for character 'a' is 61h and that of character 'A' is 41h. The ASCII code of character '0' is 30h.

```
I     BYTE     -5, 251, '5a'
      WORD     -5, 0EFFH
J     DWORD    -120, 120
      WORD     17
K     EQU      32H
L     BYTE     K+5, K*5
      BYTE     3, 2 dup(1,-1)
```

Variable	Linear Address	Content	Variable	Linear Address	Content
I	00404000	FB		0040400F	00
	00404001	FB		00404010	11
	00404002	35		00404011	00
	00404003	61	L	00404012	37
	00404004	FB		00404013	FA
	00404005	FF		00404014	03
	00404006	FF		00404015	01
	00404007	0E		00404016	FF
J	00404008	88		00404017	01
	00404009	FF		00404018	FF
	0040400A	FF			
	0040400B	FF			
	0040400C	78			
	0040400D	00			
	0040400E	00			

[25 Points]

(Q4) Write **separate** assembly programs to do the following using the smallest possible instructions. You do not need to show the full structure of the program, **just show the needed assembly instructions in your solution**.

(i) Assume that you have an Array of integers, declared as IntArray, with each integer defined as a **Word**. Write an assembly program to **reverse** the content of IntArray. **Your program should work for any array size**.

For example, assume the following array definition:
IntArray Word 1, 2, 3, 4, 5, 6

After executing the program, the content of IntArray will be:
IntArray Word 6, 5, 4, 3, 2, 1

```
        MOV ECX, Lengthof Array/2
        MOV ESI, Offset Array
        MOV EDI, Offset Array+Sizeof Array -2
Next:
        MOV AX, [ESI]
        XCHG AX, [EDI]
        MOV [ESI], AX
        ADD ESI, 2
        SUB EDI, 2
        LOOP Next
```

(ii) Assume that you have a two-dimensional array of integers, declared as TwoDArray, with each integer defined as a **Byte**. Write an assembly program to **increment each integer in row i by the value i+1.** Assume that the number of rows and number of columns in the array are defined in the constants NRow and NCol, respectively. **Your program should work for any array size**.

For example, assume the following array definition:

```
NRow EQU 4
NCol EQU 5
TwoDArray    Byte 1, 2, 3, 4, 5
             Byte 6, 7, 8, 9, 10
             Byte 11, 12, 13, 14, 15
             Byte 16, 17, 18, 19, 20
```

After executing the program, the content of TwoDArray will be:

```
TwoDArray    Byte 2, 3, 4, 5, 6
             Byte 8, 9, 10, 11, 12
             Byte 14, 15, 16, 17, 18
             Byte 20, 21, 22, 23, 24
```

```
        MOV ECX, NRow
        MOV AL, 1              ; set row increment value
        MOV EBX, 0             ; array index
RLoop:
        MOV EDX, ECX           ; save row loop counter
        MOV ECX, NCol
CLoop:
        ADD TwoDArray[EBX], AL
        INC EBX
        Loop   CLoop
        MOV ECX, EDX           ; restore row loop counter
        INC AL                 ; update row increment value
        Loop RLoop
```