16.317: Microprocessor Systems Design I

Fall 2014

Exam 1 Solution

1. (20 points, 5 points per part) ***Multiple choice***

For each of the multiple choice questions below, clearly indicate your response by circling or underlining the single choice you think best answers the question.

a. How many iterations does the following loop execute?

```
          MOV  CX, 0008H
          MOV  AX, 0000H
START:    ADD  AX, 0003H
          CMP  AX, CX
          LOOPNE START
```

i. 2

ii. 3

iii. 4

iv. 6

v. <u>8</u>

b. Assuming A, B, C, and D are all signed integers, what compound condition does the following instruction sequence test?

```
MOV     AX, D
CMP     A, AX
SETL    BL
SUB     AX, B
CMP     AX, C
SETGE   BH
AND     BL, BH
```

i. `(A < D) && (B >= C)`

ii. `(A < D) && (D >= C)`

iii. `(A <= D) && (D – B >= C)`

iv. ***`(A < D) && (D – B >= C)`***

v. `(A <= D) && (B – D >= C)`

c. Given the conditional statement below:

```
if (AX == BX)
   CX++;
else
   CX--;
```

Which of the following assembly sequences correctly implements this conditional statement?

```
[✓ i.] CMP  AX, BX
       JE   L1
       DEC  CX
       JMP  L2
L1:    INC  CX
L2:                      ; End of statement
```

```
  ii.  CMP  AX, BX
       JNE  L1
       DEC  CX
       JMP  L2
L1:    INC  CX
L2:                      ; End of statement
```

```
 iii.  CMP  AX, BX
       JE   L1
       DEC  CX
L1:    INC  CX
L2:                      ; End of statement
```

```
  iv.  CMP  AX, BX
       JNE  L1
       DEC  CX
L1:    INC  CX
L2:                      ; End of statement
```

d. If `AX = 1001H`, which of the following choices correctly shows the results of performing the two bit scan instructions (`BSF` and `BSR`) on this register?

i. ***`BSF DX, AX      → ZF = 1, DX = 0000h`***
`BSR DX, AX      → ZF = 1, DX = 000Ch`

ii. `BSF DX, AX      → ZF = 1, DX = 0000h`
`BSR DX, AX      → ZF = 1, DX = 0003h`

iii. `BSF DX, AX      → ZF = 0, DX = 0000h`
`BSR DX, AX      → ZF = 0, DX = 000Ch`

iv. `BSF DX, AX      → ZF = 1, DX = 000Ch`
`BSR DX, AX      → ZF = 1, DX = 0000h`

v. `BSF DX, AX      → ZF = 0, DX` unchanged
`BSR DX, AX      → ZF = 0, DX` unchanged

2. (30 points) ***Data transfers and memory addressing***

For each data transfer instruction shown below, list all changed registers and/or memory locations and their final values. If memory is changed, be sure to explicitly list **all changed bytes**. Also, indicate if each instruction performs an aligned memory access, an unaligned memory access, or no memory access at all.

Initial state:

EAX: 00000000h
EBX: 00000005h
ECX: 00000003h
EDX: 0000FE98h
ESI: 00092900h
EDI: 00092910h

Address	Lo			Hi
92900h	05	99	33	82
92904h	A0	11	FE	20
92908h	CC	08	19	27
9290Ch	02	17	20	14
92910h	16	31	70	AA
92914h	BE	CD	FA	00
92918h	49	64	7A	0F

Instructions:

`MOVSX EAX, WORD PTR [ESI+EBX]` Aligned? Yes [✓ No] Not a memory access

Address = ESI + EBX = 00092900h + 00000005h = 00092905h
EAX = sign-extended word at 00092905h = FFFFFE11h

`MOV   [EDI+2*ECX], DX` Aligned? [✓ Yes] No Not a memory access

Address = EDI + 2 * ECX = 00092910h + 2*00000003h = 00092916h
mem(92916h) = DX = FE98h (byte at 92916h = 98h, 92917h = FEh)

`LEA   CX, [SI+07FEh]` Aligned? Yes No [✓ Not a memory access]

CX = SI + 07FEh = 2900h + 07FEh = 30FEh

`XCHG  [ESI+EBX+02h], DL` Aligned? [✓ Yes] No Not a memory access

Address = ESI + EBX + 02h = 00092900h + 5 + 2 = 00092907h
mem(92907h) = DL = 98h
DL = mem(92907h) = 20h

`MOVZX EBX, BYTE PTR [ESI+001Ah]` Aligned? [✓ Yes] No Not a memory access

Address = ESI + 001Ah = 00092900h + 001Ah = 9291Ah
EBX = zero-extended byte at 9291Ah = 0000007Ah

3. (25 points) ***<u>Arithmetic instructions</u>***

For each instruction in the sequence shown below, list <u>all</u> changed registers and/or memory locations and their new values. If memory is changed, be sure to explicitly list **<u>all changed bytes</u>**. Where appropriate, you should also list the state of the carry flag (CF).

<u>Initial state:</u>

EAX: 00000114h
EBX: 0000FFE8h
ECX: 00001013h
EDX: 00000004h
CF: 1
ESI: 000316F0h

Address	Lo			Hi
31700H	04	07	08	00
31704H	83	00	01	01
31708H	05	01	71	31
3170CH	20	40	60	80
31710H	05	00	AB	0F
31714H	00	16	11	55

<u>Instructions:</u>

`SUB CX, [ESI+20h]` *(original exam had typo: SI, not ESI)*

Address = ESI + 20h = 316F0h + 20h = 31710h

CX = CX – (31710h) = 1013h – 0005h = <u>100Eh</u>, <u>CF = 0</u>

`ADC BX, AX`

BX = BX + AX + CF = FFE8h + 0114h + 0 = <u>00FCh</u>, <u>CF = 1</u>

`INC BL`

BL = BL + 1 = FCh + 1 = <u>FDh</u>

`MUL BYTE PTR [ESI+10h]` *(original exam had typo: SI, not ESI)*

Address = ESI + 10h = 316F0h + 10h = 31700h
AX = AL * (31700h) = 14h * 04h = 20 * 4 = 80 = <u>0050h</u>

`NEG CX`

CX = -CX = -(100Eh) = -(0001 0000 0000 1110$_2$***)***

= 1110 1111 1111 0001$_2$ ***+ 1***

= 1110 1111 1111 0010$_2$ ***= <u>EFF2h</u>***

4. (25 points) ***<u>Logical instructions</u>***

For each instruction in the sequence shown below, list <u>all</u> changed registers and/or memory locations and their new values. If memory is changed, be sure to explicitly list **<u>all changed bytes</u>**. Where appropriate, you should also list the state of the carry flag (CF).

<u>Initial state:</u>

EAX: 000000FFh
EBX: 00001172h
ECX: 00000005h
EDX: 0000F63Ch
CF: 0

Address	Lo			Hi
72300h	C0	00	02	10
72304h	10	10	15	5A
72308h	89	01	05	B1
7230Ch	20	40	AC	DC
72310h	04	08	05	83

<u>Instructions:</u>

```
RCL   DL, 3
```

(CF, DL) = (CF,DL) rotated left by 3 = (0,3Ch) rotated left by 3

- ➔ ***<u>0 00</u>11 1100 rotated left by 3 = 1 1110 0000***
- ➔ ***<u>CF = 1; DL = E0h</u>***

```
XOR   DX, AX
```

DX = DX XOR AX = F6E0h XOR 00FFh = <u>F61Fh</u>

```
SAR   DH, 4
```

DH = DH >> 4 (keep sign intact) = F6 >> 4 = <u>FFh</u>, <u>CF = 0</u>

```
SHL   DL, 4
```

DL = DL << 4 = 1Fh << 4 = <u>F0h</u>, <u>CF = 1</u>

```
NOT   DX
```

DX = ~DX = ~FFF0h = <u>000Fh</u>

5. (10 points) ***Extra credit***

Complete the code snippet below by writing the appropriate x86 instruction into each of the blank spaces. The purpose of each instruction is described in a comment to the right of the blank.

```
LEA  EDX, [ECX+ESI+00002014h]           ; Set EDX to the sum
                                        ;  ECX + ESI + 00002014h
MOV  BX, [EDX]                          ; Load a word into BX
                                        ;  from the address
                                        ;  stored in EDX
IDIV BX                                 ; Divide the signed
                                        ;  double word in
                                        ;  registers (DX,AX) by
                                        ;  the value in BX
MOV  [93014h], AX                       ; Use two instructions
MOV  [93016h], DX                       ;  to store the quotient
                                        ;  of the division at
                                        ;  address 93014h, and
                                        ;  store the remainder
                                        ;  of the division at
                                        ;  the next aligned
                                        ;  address after 93014h

SHL  BX, 2                              ; Multiply BX by 4 and
                                        ;  store the result in
                                        ;  BX using a single
                                        ;  instruction
XOR  BX, F00Fh                          ; Complement the upper 4
                                        ;  and lower 4 bits of
                                        ;  BX, but don't change
                                        ;  any other bits
ROR  BX, 4                              ; Modify BX so that the
or                                      ;  8 bits you just
ROL  BX, 12                             ;  changed are in BH,
                                        ;  and the remaining
                                        ;  bits are in BL
BT   BL, 0                              ; Check the value of the
                                        ;  lowest bit of BL
CMOVNC [ESI], BH                        ; If that lowest bit is
                                        ;  0, store the value of
                                        ;  BH at the address
                                        ;  stored in ESI
```