16.317: Microprocessor Systems Design I

Fall 2013

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. If `AX = 0FF0h`, which of the following instructions will set `CF = 1` and change `AX` to `0EF0h`?

A. `BTR AX, 8`
B. `BT AX, 8`
C. `BTC AX, 8`
D. `BTS AX, 8`

i. A and C

ii. A and D

iii. B and C

iv. B and D

v. None of the above

1 (continued)

b. 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

c. If AX = 000Fh and CF = 0, initially, what is the result of the instruction ROR AX, 4?

i. AX = 00F0h, CF = 0

ii. ***AX = F000h, CF = 1***

iii. AX = E000h, CF = 1

iv. AX = 0000h, CF = 1

v. AX = 00FFh, CF = 1

1 (continued)

d. If `AX = 0001h` and `CF = 1`, initially, what is the result of the instruction `RCL AX, 2`?

i. `AX = 0000h, CF = 0`

ii. `AX = 0003h, CF = 0`

iii. `AX = 0004h, CF = 0`

iv. ***`AX = 0006h, CF = 0`***

v. `AX = 1000h, CF = 1`

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: 00000006h
ECX: 00000001h
EDX: 0000FF00h
ESI: 0000F000h
EDI: 00001000h
DS: 9300h
ES: 9200h

Address	Lo			Hi
93000h	B0	21	AA	36
93004h	15	99	FE	0C
93008h	CE	12	60	EB
9300Ch	89	0A	0B	FF
93010h	00	11	03	20
93014h	08	17	A1	B8
93018h	99	30	CB	ED

Instructions:

`MOV ES:[DI+10h], BL` Aligned? ***Yes*** No Not a memory access

EA = DI + 0010h = 1000h + 0010h = 1010h
SBA = 92000h (access ES)
LA = SBA + EA = 92000h + 1010h = 93010h
Byte @ 93010h = BL = 06h

`LEA DI, [SI+4*CX]` Aligned? Yes No ***Not a memory access***

EA = SI + (4 * CX) = F000h + (4 * 0001h) = F004h
DI = EA = F004h

`MOV AX, [SI+1003h]` Aligned? Yes ***No*** Not a memory access

EA = SI + 1003h = F000h + 1003h = ~~1~~0003h (EA is only 16 bits)
SBA = 93000h (access DS)
LA = SBA + EA = 93000h + 0003h = 93003h
AX = word @ 93003h = 1536h

`MOVZX EDX, BYTE PTR ES:[BX+1000h]` Aligned? ***Yes*** No Not a memory access

EA = BX + 1000h = 0006h + 1000h = 1006h
SBA = 92000h (access ES)
LA = SBA + EA = 92000h + 1006h = 93006h
EDX = zero-extended byte @ 93006h = 000000FEh

`MOVSX EBX, WORD PTR [000Eh]` Aligned? ***Yes*** No Not a memory access

EA = 000Eh
SBA = 93000h (access DS)
LA = SBA + EA = 93000h + 000Eh = 9300Eh
EBX = sign-extended word at 9300Eh = FFFFFF0Bh

3. (25 points) ***Arithmetic instructions***

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

Initial state:
EAX: 0000FFF7h
EBX: 000000A4h
ECX: 00000003h
EDX: 0000FFFEh
CF: 1
ESI: 00000004H
DS: 3170H

Address	Lo			Hi
31700H	04	00	08	00
31704H	83	00	01	01
31708H	05	01	71	31
3170CH	20	40	60	80
31710H	02	00	AB	0F
31714H	00	16	11	55

Instructions:

```
SBB   BX, [SI]
```

EA = SI = 0004H; SBA = 31700 (access DS) → LA = 31704h
BX = BX - word at 31704h - CF
= 00A4h - 0083h - 1 = 0020h

```
ADD   AX, BX
```

AX = AX + BX = FFF7h + 0020h = 0017h
CF = 1

```
DEC   AX
```

AX = AX - 1 = 0016h

```
IDIV  CL
```

AL = AX / CL = 0016h / 03h = 22 / 3 = 7 = 07h
AH = AX % CL (remainder) = 22 % 3 = 1 = 01h

```
NEG   DL
```

DL = -DL = -FEh = -(1111 1110$_2$)
= 0000 0001$_2$ + 1 = 0000 0010$_2$ = 02h

4. (25 points) ***Logical instructions***

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

Initial state:
EAX: 000000E7h
EBX: 00003300h
ECX: 00000002h
EDX: 0000F63Ch
CF: 0
DS: 7230h

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

Instructions:

```
XOR   AL, [0DH]
```

LA = SBA + EA = 72300h + 0Dh = 7230Dh
AL = AL XOR (byte at 7230Dh) = E7h XOR 40h = A7h

```
AND   AL, BH
```

AL = AL AND BH = A7h AND 3Ch = 23h

```
ROR   AL, CL
```

AL = AL rotated right 2 bits (since CL = 2)
= 23h rotated right 2 bits = 0010 0011$_2$ rot. right 2 bits
= 1100 1000$_2$ = C8h
CF = copy of last bit rotated out = 1

```
SAR   AL, 4
```

AL = AL >> 4 (arithmetic shift)
= C8h >> 4 = 1100 1000$_2$ >> 4 = 1111 1100$_2$ = FCh
CF = last bit shifted out = 1

```
RCL   AL, 3
```

AL = AL rotated left through carry 3 bits
(CF,AL) = 1 1111 1100$_2$ rotated left 3 bits
= 1 1110 0111$_2$
AL = 1110 0111$_2$ = E7h, CF = 1

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

Complete the program 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.

```
MOV  AX, 6317h      ; Use two instructions
                    ;   to establish 63170h
                    ;   as the starting address
MOV  DS, AX         ;   of the data segment

LES  SI, [0000h]    ; Load the first two
                    ;   bytes stored in
                    ;   the current data
                    ;   segment into SI,
                    ;   and the next two
                    ;   bytes into ES

LEA  DI, [SI+1000h] ; Set DI = SI + 1000h
                    ;   using a single
                    ;   instruction

MOV  AX, ES:[SI]    ; Load two bytes of data
                    ;   into AX from the
                    ;   extra segment (ES),
                    ;   starting at offset
                    ;   specified by SI

MOV  BX, ES:[SI+2]  ; Load the next two
                    ;   bytes of data from
                    ;   the extra segment
                    ;   into BX

ADD  AX, BX         ; Find the sum of the
                    ;   previous two values

SAR  AX, 1          ; Divide the result of
                    ;   the previous
                    ;   instruction by 2
                    ;   without using a
                    ;   divide instruction
                    ; Keep the sign intact

MOV  ES:[DI], AX    ; Store the previous
                    ;   instruction's result
                    ;   into the extra
                    ;   segment at offset
                    ;   specified by DI
```