16.317: Microprocessor Systems Design I

Spring 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. Which of the following statements about x86 real mode memory accesses are true?

A. By default, if an instruction that accesses memory does not explicitly specify a segment, that instruction will access the data segment.

B. In an x86 processor, all memory accesses involving multiple bytes must be aligned.

C. To calculate a linear address in the data segment, add the 16-bit value in DS to the 16-bit effective address. (For example, if DS = 1000h and the effective address is 2000h, the linear address will be 3000h.)

i. None of the above

ii. Only A *(2/5 points given)*

iii. Only B *(2/5 points given)*

iv. A and B

v. B and C

b. If `EAX = 10203040h` and `EBX = AABBCCDDh`, which of the following instructions will change `EAX` to `102030DDh` and `EBX` to `AABBCC40h`?

i. `XCHG EAX, EBX`

ii. `XCHG AX, BX`

iii. `XCHG AL, BL`

iv. `XCHG AH, BH`

v. None of the above

1 (continued)

c. If `EAX = 00000003h` and `EBX = 00000005h`, what will the result of the instruction `IMUL BL` be?

i. `EAX = 00000005h`

ii. EAX = 0000000Fh

iii. `EAX = 00000015h`

iv. `EAX = 0000FFF8h`

v. `EAX = 00000005h, EDX = 00000003h`

d. Which of the following instructions will set `CF = 1` if `EAX = 0000F001h` and `EBX = 00001000h`?

A. `ADD AX, BX`

B. `SUB BX, AX`

C. `SHR AX, 1`

D. `SHL BX, 1`

i. Only A

ii. Only D

iii. A and C

iv. B and D

v. A, B, and C

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: 00000002h
ECX: 00000001h
EDX: 00001FFEh
ESI: 0000F00Fh
EDI: 0000A000h
DS: 1245h
ES: 1046h

Address	Lo			Hi
12450h	02	17	20	14
12454h	16	31	70	AA
12458h	BE	CD	FA	00
1245Ch	49	64	7A	0F
12460h	FF	11	02	60
12464h	01	04	65	7F
12468h	99	30	88	78

Instructions:

`MOV EAX, [BX+4*CX]` Aligned? Yes ***No*** Not a memory access

EA = BX + (4 * CX) = 0002h + (4 * 0001h) = 0006h
SBA = 12450h (access DS)
LA = SBA + EA = 12450h + 0006h = 12456h
EAX = Double word @ 12456h = CDBEAA70h

`MOV ES:[DI+8005h], DL` Aligned? ***Yes*** No Not a memory access

EA = DI + 8005h = A000h + 8005h = ~~1~~2005h (EA is only 16 bits)
SBA = 10460h (access ES)
LA = SBA + EA = 10460h + 2005h = 12465h
Byte @ 12465h = DL = FEh

`LEA BX, [SI+0FF3h]` Aligned? Yes No ***Not a memory access***

EA = SI + 0FF3h = F00Fh + 0FF3h = ~~1~~0002h (EA is only 16 bits)
BX = EA = 0002h

`MOVSX EDX, BYTE PTR ES:[CX+2009h]` Aligned? ***Yes*** No Not a memory access

EA = CX + 2009h = 0001h + 2009h = 200Ah
SBA = 10460h (access ES)
LA = SBA + EA = 10460h + 200Ah = 1246Ah
EDX = sign-extended byte @ 1246Ah = DL = FFFFFF88h

`MOVZX EBX, WORD PTR [0009h]` Aligned? Yes ***No*** Not a memory access

EA = 0009h
SBA = 12450h (access DS)
LA = SBA + EA = 12450h + 0009h = 12459h
EBX = zero-extended word @ 12459h = 0000FACDh

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: 00000014h
EBX: 0000FF08h
ECX: 00000003h
EDX: 00000004h
CF: 1
ESI: 00000008H
DS: 3170H

Address	Lo			Hi
31700H	04	07	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:

```
ADD   CX, [SI]
```

EA = SI = 0008h; SBA = 31700h (access DS) → LA = 31708h
CX = CX + word at 31708h
= 0003h + 0105h = 0108h
CF = 0

```
SBB   BX, AX
```

BX = BX - AX - CF
= FF08h - 0014h - 0 = FEF4h
CF = 0

```
DEC   BH
```

BH = BH – 1 = FEh – 1 = FDh

```
IDIV  BYTE PTR [0001h]
```

EA = 0001h; SBA = 31700h (access DS) → LA = 31701h
AL = AX / byte @ 31701h = 0014h / 07h = 20 / 7 = 2 = 02h
AH = AX % byte @ 317071h (remainder) = 20 % 7 = 6 = 06h

```
NEG   DX
```

DX = -DX = -0004h $= -(0000\ 0000\ 0000\ 0100_2)$
$= 1111\ 1111\ 1111\ 1011_2 + 1$
$= 1111\ 1111\ 1111\ 1100_2 = FFFCh$

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: 000000FFh
EBX: 00001172h
ECX: 00000005h
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:

```
AND   BL, [07H]
```

EA = SI = 0007h; SBA = 72300h (access DS) → LA = 72307h
BL = BL AND byte @ 72307h = 72h AND 5Ah = 52h

```
XOR   AL, BL
```

AL = AL XOR BL = FFh XOR 52h = ADh

```
SAR   AL, CL
```

AL = AL >> CL = AL >> 5 (arithmetic shift—keep sign intact)
= ADh >> 5 = 1010 1101$_2$ >> 5 = 1111 1101$_2$ = FDh
CF = last bit shifted out = 0

```
SHL   AL, 4
```

AL = AL << 4
= FDh << 4 = 1111 1101$_2$ << 4 = 1101 0000$_2$ = D0h
CF = last bit shifted out = 1

```
NOT   AL
```

AL = ~AL (flip bits) = ~D0h = ~1101 0000$_2$ = 0010 1111$_2$ = 2Fh

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. You should assume the starting address of the data segment is appropriately set up for you.

```
MOV  BX, [0000h]     ; Load the first two
                     ;  bytes in the data
                     ;  segment into BX

LGS  CX, [0002h]     ; Use one instruction to
                     ;  load the next two
                     ;  bytes in the data
                     ;  segment into CX,
                     ;  and the two bytes
                     ;  after that into GS

SUB  CX, BX          ; Find the difference
                     ;  CX – BX, storing
                     ;  the result in CX

MOV  AX, CX –or-     ; Use two instructions
MOV  AX, BX          ;  to take the new
                     ;  value in CX and
IMUL BX –or-         ;  multiply it by BX
IMUL CX              ; Hint: the result may
(could use MUL       ;  not be in either
Instead of IMUL)     ;  of those registers

MOV  GS:[0000h], AX  ; Store all 32 bits of
                     ;  that result in the
                     ;  first four bytes of
MOV  GS:[0002h], DX  ;  segment GS, using two
                     ;  instructions (least
                     ;  significant bits 1st)

AND  BX, 0FFFh       ; Clear the upper 4
                     ;  bits of BX, but don't
                     ;  change any other bits
                     ; (Clear = set to 0)

XOR  BX, 000Fh       ; Flip the lowest 4 bits
                     ;  of BX, but don't
                     ;  change any other bits

MOV  [CX+DI], BX     ; Store BX into the data
                     ;  segment at the offset
                     ;  specified by the sum
                     ;  of DI and CX
```