16.317: Microprocessor-Based Systems I

Spring 2012

Exam 2 Solution

1. (20 points, 5 points per part) ***<u>Multiple choice</u>***

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. Given CS = 1000H, IP = E000, and EBX = 1E001000, which of the following CALL instructions will transfer control to an instruction at physical address 1F000H?

   A. CALL 1000H
   B. CALL F000H
   C. CALL BX
   D. CALL EBX
   E. CALL HOME_BECAUSE_YOUR_MOTHER_MISSES_YOU
      (hey, for all you know, that could be a valid instruction label)

   i. A and C
   ii. B and C
   iii. <u>A and D</u>
   iv. B and D

b. How many iterations does the following loop execute?

```
        MOV  CX, 0008H
        MOV  AX, 0000H
START:  ADD  AX, 0002H
        CMP  AX, CX
        LOOPNE START
```

   i. 2
   ii. <u>3</u>
   iii. 4
   iv. 6
   v. 8

1 (cont.)

c. 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. ***<u>`(A < D) && (D – B >= C)`</u>***

v. `(A <= D) && (B – D >= C)`

d. Which of the following statements about virtual memory are <u>true?</u>

A. When translating a virtual address to a physical address, the virtual page number is replaced by the appropriate physical frame number, while the lower bits of the address—the page offset—remain the same.

B. The number of bits in the page offset depends on the number of pages in the virtual address space.

C. Because all virtual pages cannot fit in physical memory, each page table entry requires a valid bit to indicate if the frame number in that entry is valid.

D. The TLB is a sandwich containing the same ingredients as a BLT, but with those ingredients stacked in the opposite order.

i. Only A

ii. Only C

iii. A and B

iv. ***<u>A and C</u>***

v. A, B, and C

2. (40 points) ***<u>Protected mode memory accesses</u>***

Assume the 80386 is running in protected mode with the state given below. Note that each memory location shown contains a descriptor for a particular segment.

GDTR = 123000080017
LDTR = 0008
LDTR cache: base = 12300028
LDTR cache: limit = 0027

DS = 0006
ESI = 0000CD04
EBX = 00031A0

Memory	Address
Base = 030010F0 Limit = 020F	12300000
Base = 12300020 Limit = 0007	12300008
Base = 12300028 Limit = 0027	12300010
Base = 1200C000 Limit = FFFF	12300018
Base = 12340000 Limit = 00FF	12300020

Memory	Address
Base = AC000000 Limit = 0317	12300028
Base = 01610200 Limit = 03F7	12300030
Base = 03170214 Limit = 030F	12300038
Base = 06B01000 Limit = 0F07	12300040
Base = 05000120 Limit = 000F	12300048

What address does each of the following instructions access? (<u>Hint</u>: solving part (a) should help you solve parts (b) and (c)).

a. `MOV     AX, [00H]`

<u>Solution:</u> *Remember that:*

- *Calculating memory addresses, even in protected mode, is about adding the starting address of a segment to the effective address specified in the instruction.*
- *In protected mode, to find the starting address of a segment, you need to find the descriptor that describes your segment.*
    - *Each descriptor is an entry in either the GDT or the LDT, tables that start at addresses specified by the GDTR or the LDTR cache, respectively.*
    - *The selector for the current segment contains an index field that points to a descriptor in the specified table.*

In this problem, the selector is always the DS register, which has the following binary value:

DS = 0006H = ***$0000\ 0000\ 0000\ 0110_2$ → index = 0, TI = 1, RPL = 10***

For this problem, we ignore the RPL (requested priority level) and focus on the other fields:

- *TI = 1 indicates that this is a local memory access, so the descriptor with information about the data segment is in the LDT.*
- *Index = 0 indicates that the correct descriptor is the first descriptor in that table.*

According to the LDTR cache, the base address of the LDT is 12300028H; the first selector at that address indicates that the data segment has a base address of AC000000H. Therefore, the address being accessed is:

(Segment base) + (effective address) = AC000000H + 00H = ***AC000000H***

2 (cont.)

b. `ADD [SI], CX`

Solution: *Since we found the starting address of DS (AC000000H) in the previous part, we don't need to go through that process again. This instruction uses indexed addressing, so the effective address is equal to the value of SI = CD04H, and the address being accessed is:*

(Segment base) + (effective address) = AC000000H + CD04H = ***AC00CD04H***

Note: *As one of you pointed out after the test (and several people wrote in their solutions), that effective address is greater than the limit of the segment (limit = 0317H) and is therefore invalid.*

Technically, this effective address can be valid—as we briefly discussed in class, the granularity bit in the descriptor can indicate that the segment size that you would normally compute from the limit (limit + 1) is essentially multiplied by 2^{12}, and if that were the case, the effective address would easily fit inside that segment. However, my intent was not to assume that was the case in this problem—I simply made a mistake, for which I apologize. Hopefully, it didn't cause too much confusion.

c. `SHL [BX+10H], 7`

Solution: *As in part (b), we already have the starting address of our data segment. The effective address is BX + 10H = 31A0H + 10H = 31B0H, so the actual address is:*

(Segment base) + (effective address) = AC000000H + 31B0H = ***AC0031B0H***

Note: *The above note about invalid addresses unfortunately applies to this problem, too.*

3. (40 points) ***Assembly language***

For each instruction sequence shown below, list all changed registers, memory locations, and/or flags, as well as their new values.

a. Initial state:

- (EAX) = 0000ABC0H
- (EBX) = 000012ACH
- (ECX) = 00000020H
- (EDX) = 00000000H
- (ESI) = 00000100H
- (EDI) = 00000200H
- (DS:100H) = 00H
- (DS:101H) = F0H
- (DS:110H) = 00H
- (DS:111H) = FFH
- (DS:200H) = 30H
- (DS:201H) = 00H
- (DS:210H) = AAH
- (DS:211H) = AAH
- (DS:220H) = 55H
- (DS:221H) = 55H
- (DS:300H) = AAH
- (DS:301H) = 55H

Also, assume all flags (ZF, CF, SF, PF, OF) are initialized to 0.

Solution:

```
      BSF      DX, AX     → Forward bit scan of AX;
                              index of first non-zero bit
                              stored in DX
                          → AX = ABC0H = 1010 1011 1100 0000₂
                          → First nonzero bit = bit 6
                          → DX = 0006H
                          → ZF = 1 (indicates result is non-
                              zero (in bit scans only))
      JNZ      END        → Jump if ZF == 0
                          → Jump is not taken since ZF == 1
      BT       BX, DX     → Test bit in BX, store value in CF
                          → Index = DX = 6
                          → BX = 12ACH = 0001 0010 1010 1100₂
                          → CF = Bit 6 of BX = 0
      SETNC    [100H]     → Byte at DS:100H = FFH if CF == 0,
                              00H otherwise
                          → Since CF == 0, DS:100H = FFH
END:  AND   CL, [100H]    → CL = CL & DS:100H = CL & FFH
                          → CL unchanged since any byte && FFH
                              remains same → CL = 20H
```

3 (cont.)

b. Initial state:

- (EAX) = 00000016H
- (EBX) = 00000317H
- (ECX) = 00000010H
- (EDX) = 0000ABCDH
- (ESI) = 00000100H
- (EDI) = 00000106H
- (DS:100H) = 0FH
- (DS:101H) = F0H
- (DS:102H) = 00H
- (DS:103H) = FFH
- (DS:104H) = 30H
- (DS:105H) = 00H
- (DS:106H) = AAH
- (DS:107H) = AAH
- (DS:108H) = 55H
- (DS:109H) = 55H
- (DS:10AH) = AAH
- (DS:10BH) = 55H

Also, assume all flags (ZF, CF, SF, PF, OF) are initialized to 0.

Solution:

```
      CMP    AX, BX        → Set flags based on result of AX – BX
                               = 0016H – 0317H = FCFFH
                           → SF = 1 (result negative)
                             ZF = 0 (result non-zero)
                             CF = 1 (borrow out of MSB)
                             OF = 0 (no overflow)
                             PF = 1 (even parity)
      JE     L1            → Jump if comparison shows values equal
                               → Jump not taken
      JG     L2            → Jump if comparison shows AX > BX
                               → Jump not taken
      INC    AX            → AX = AX + 1 = 0017H
                           → Also sets flags, although I did not
                               grade you on these values:
                           → SF = 0 (result positive)
                             ZF = 0 (result non-zero)
                             CF = 0 (no carry out)
                             OF = 0 (no overflow)
                             PF = 1 (even parity)
      JMP    END           → Unconditionally jump to label END—
                               skip next 3 instructions
L1:   DEC    AX
      JMP    END
L2:   MOV    AX, BX
END:  MOV  [DI+02H], AX    → Move word in AX to DS:DI+02H
                               = DS:0106H + 02H = DS:0108H
                           → DS:108H = 17H
                           → DS:109H = 00H
```