16.317: Microprocessor Systems Design I

Spring 2013

Exam 2 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 values are stored in a function's stack frame after the function has been called (in other words, after the CALL instruction that calls the function has finished executing)?

   A. Function arguments
   B. Local variables inside the function
   C. The previous value of the base pointer (EBP)
   D. The previous value of the instruction pointer (EIP)

   i. A and C

   ii. B and C

   iii. A and D

   iv. B and D

   v. All of the above (A, B, C, and D)

1 (continued)

b. Say a function contains the following variables:

```
int     x, y;
double  z;
char    list[40];
```

Assume that a variable of type char holds 1 byte, a variable of type int holds 4 bytes, and a variable of type double holds 8 bytes. Which of the following instructions correctly creates enough space on the stack for all of the variables listed above? (Note: All constant values shown below are in decimal, not hexadecimal.)

i. `ADD  ESP, 56`

ii. SUB ESP, 56

iii. `PUSHAD`

iv. `ADD  EBP, 56`

v. `SUB  EBP, 56`

c. Which of the following types of values are not stored on the stack?

i. Function arguments

ii. Local variables

iii. Global variables

iv. Function return addresses

v. Registers saved inside a function, so that the original values can be restored at the end of the function.

1 (continued)
d. How many iterations does the following loop execute?

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

i. 2

ii. 3

iii. 4

iv. 6

v. 8

2. (40 points) ***Protected mode memory accesses***

Assume the 80386 is running in protected mode with the state given below. Note that each memory location shown contains a segment descriptor. Also, please note that you cannot assume the memory range shown contains the entire GDT and LDT. All values shown are in hex.

GDTR = 327201380027
LDTR = 0020
LDTR cache: base = 32720168
LDTR cache: limit = 003F

DS = 0017
ES = 0001
SS = 0019
EDI = 31703509
EBP = 001A05EA

Memory	Address
Base = 32720160 Limit = 0007	32720130
Base = 31700F00 **Limit = 0A17**	**32720138** **ES desc.**
Base = 32720120 Limit = 0017	32720140
Base = 0A1B3200 Limit = FFFF	32720148
Base = 32720200 **Limit = FFFF**	**32720150** **SS desc.**

Memory	Address
Base = 32720168 Limit = 003F	32720158
Base = 32720130 Limit = 0007	32720160
Base = FE0A1340 Limit = FFFF	32720168
Base = 32720100 Limit = 001F	32720170
Base = 3300C000 **Limit = 02FF**	**32720178**

What physical address does each of the following instructions access?

a. `SETL  BYTE PTR [DI+0200H]`

Solution: *To find a segment base address, look first at its selector—in this case, DS:*

DS = 0017H = ***0000 0000 0001 0111***$_2$ → *index = 2, TI = 1 (local), RPL = 3*

Since the LDT starts at address 32720168, the descriptor at 32720178 (which has index 2 within the LDT) describes the data segment. So, the physical address being accessed is:

Seg. base + EA = 3300C000H + (DI+0200H) = 3300C000H + 3709H = ***3300F709H***

b. `SUB  CX, ES:[DI-8]`

Solution: *In this problem, the segment we need is ES, so we break down that selector:*

ES = 0001H = ***0000 0000 0000 0001***$_2$ → *index = 0, TI = 0 (global), RPL = 1*

Since the GDT starts at address 32720138, the descriptor at 32720138 (which has index 0 within the GDT) describes this segment. So, the physical address being accessed is:

Seg. base + EA = 31700F00H + (DI-8) = 31700F00H + 3501H = ***31704401H***

2 (continued)

c. `SHL  WORD PTR SS:[BP-4], 7`

Solution: *In this problem, the segment we need is SS, so we break down that selector:*

SS = 0019H = ***0000 0000 0001 1001***$_2$ → *index = 3, TI = 0 (global), RPL = 1*

Since the GDT starts at address 32720138, the descriptor at 32720150 (which has index 3 within the GDT) describes this segment. So, the physical address being accessed is:

Seg. base + EA = 32720200H + (BP-4) = 32720200H + 05E6H = ***327207E6H***

3. (40 points) ***Conditional instructions***

For each part of this problem, write a short 80386DX code sequence that performs the specified operation. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the space provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

a. Implement the following conditional statement. You may assume that "X" and "Y" refer to 16-bit variables stored in memory, which can be directly accessed using those names (for example, `MOV AX, X` would move the contents of variable "X" to AX).

```
if (AX < 10) {
   CX = X + 10;
}
else if (AX == 20) {
   CX = CX – Y;
}
else {
   CX = X + Y;
}
```

Solution: *Other solutions may be acceptable; the key pieces to this problem are:*

- *Evaluating the two conditions properly*
- *Ensuring that you only execute one of the blocks—the "if" case, "else if" case, or "else" case.*

```
     CMP  AX, 10
     JL   IF        ; Go to "IF" if AX < 10
     CMP  AX, 20
     JE   ELIF      ; Go to "ELIF" if AX == 20
     MOV  CX, X     ; "Else" case--CX = X + Y
     ADD  CX, Y
     JMP  FIN       ; Skip "if", "else if" cases
IF:  MOV  CX, X     ; "If" case--CX = X + 10
     ADD  CX, 10
     JMP  FIN       ; Skip "else if" case
ELIF: SUB CX, Y     ; "Else if" case--CX = CX – Y
FIN:                ; End of statement
```

3 (continued)

b. Implement the following loop. As in part (a), assume "X" is a 16-bit variable in memory that can be accessed by name. (Hint: Any loop that executes the correct number of iterations is acceptable—you do not necessarily have to change your loop counter in exactly the same way as the for loop, since `i` is not used in the body of the loop.)

```
for (i = 0; i < X; i++) {
   AX = AX + X;
   BX = BX - X;
   if (AX == BX)
        break;          // Exit loop early
}
```

Solution: *Other solutions may be valid; the key pieces of this problem are:*

- *Ensuring that the assignment statements are enclosed in a loop with X iterations.*
  - *Note that, as mentioned above, any loop with X iterations will be valid. The solution below takes advantage of the x86 LOOP instructions so that the actual loop counts from X down to 0, rather than counting up.*
- *Comparing AX to BX and exiting the loop early if they are equal.*
  - *Note that this can be accomplished by using a LOOPNE instruction, as shown below, or by adding an explicit jump instruction that leaves the loop when the condition is true.*

```
     MOV  CX, X          ; CX = X = # of loop iterations
L:   ADD  AX, X          ; AX = AX + X
     SUB  BX, X          ; BX = BX – X
     CMP  AX, BX
     LOOPNE L            ; Decrement CX, then check if
                         ;   CX is non-zero and previous compare
                         ;   result is "not equal" (AX != BX)
                         ; If either of those conditions are
                         ;   false, exit loop
```

3 (continued)

c. Implement the following conditional statement. As in part (a), assume "X" and "Y" are 16-bit variables in memory that can be accessed by name. (Note: Make sure you carefully count the parentheses to make sure you combine conditions correctly!)

```
if (((AX < X) && (BX < Y)) || ((AX > Y) && (BX > X))) {
   AX = AX - BX;
}
```

Solution: *Other solutions may be possible; the key piece of this problem is the evaluation of the complex condition shown, which can be done with SETcc instructions:*

```
     CMP  AX, X
     SETL DL         ; (AX < X)
     CMP  BX, Y
     SETL DH         ; (BX < Y)
     AND  DL, DH     ; ((AX < X) && (BX < Y))
     CMP  AX, Y
     SETG CL         ; (AX > Y)
     CMP  BX, X
     SETG CH         ; (BX > X)
     AND  CL, CH     ; ((AX > Y) && (BX > X))
     OR   DL, CL     ; Logical OR of previous complex conditions
                     ; DL is now 1 if the entire condition in the
                     ;   if statement is true
     JZ   SKIP       ; If result of OR is zero, skip subtraction
     SUB  AX, BX     ; AX = AX – BX
SKIP:                ; End of code
```