16.317: Microprocessor Systems Design I

Fall 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.

For all parts of the question, assume you have an x86 processor running in protected mode.

a. Which of the following selector values indicate that the given segment is in global memory?

A. `SS = 0138h`
B. `DS = FF13h`
C. `GS = 118Bh`
D. `ES = 5371h`

i. A and C

ii. B and D

iii. A, B, and C

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

v. None of the above

1 (continued)

b. Which of the following statements about local memory accesses in protected mode are **<u>true</u>**?

A. The LDTR is a selector that points to a descriptor in the global descriptor table; that descriptor provides information about the current local descriptor table.

B. The LDTR stores the base address and limit (maximum offset) for the current local descriptor table.

C. The LDTR is changed every time a task makes an access to local memory.

D. The LDTR cache stores the base address and limit for the current local descriptor table.

i. Only A

ii. Only B

iii. <u>A and D</u>

iv. B and C

v. B, C, and D

1 (continued)

Again, assume you have an x86 processor running in protected mode. Parts (c) and (d) of the question refer to the following table, which shows the current local descriptor table. The descriptor shown at address 18800000h is the first descriptor in this table.

Memory	Address
Base = 11308100h Limit = 0007h	18800000h
Base = 1578AA00h Limit = 0FFFh	18800008h
Base = 33731700h Limit = 0027h	18800010h
Base = 54440000h Limit = FFFFh	18800018h
Base = 09AC2200h Limit = 00FFh	18800020h
Base = 35700100h Limit = 007Fh	18800028h

c. If DS = 000F, what is the starting address of the current data segment?

i. 000F0h

ii. 11308100h

iii. ***1578AA00h***

iv. 18800008h

v. DEADBEEFh

d. Say the selector SS breaks down as follows: requested priority level (RPL) = 3, table indicator (TI) = 1, index = 3. If ESP = 0000FFC0h, what linear address corresponds to the logical address SS:ESP?

i. 0000FFC0h

ii. 1880FFD8h

iii. 54440000h

iv. ***5444FFC0h***

v. ABAD1DEAh

2. (40 points) ***Subroutines; HLL → assembly***

The following questions deal with the simple C function shown below, which takes two integer arguments (`v1` and `v2`), contains one local variable (`x`) and returns the value shown:

```
int f(int v1, int v2) {
  int x = v1 – v2;
  return (x << 4) - x;
}
```

a. (16 points) Draw the stack frame for this function if it is called with 10 and 2 as its arguments (in other words, a program contains the function call `f(10, 2)`). Be as specific as possible—in particular:

- Show all known values—if, for example, the argument `v1` is equal to 20, write the value 20 in your diagram, not the argument name `v1`.
- For all arguments or variables with unknown values, write the argument or variable name.
- Clearly indicate where the stack pointer (`esp`) and base pointer (`ebp`) point in the current stack frame. You do not need to know the values of these registers.

Assume the function saves the register `ebx` on the stack, since it overwrites that register.

Solution: *As the diagram below shows, the stack frame is set up as follows:*

- *Arguments are passed first, in reverse order.*
- *The return address ("saved `EIP`") of the function is pushed when the function is called.*
- *The old value of the base pointer ("old `EBP`") is pushed next.*
- *Space for local variable x is then created.*
- *The register `ebx` is saved last.*
- *After the stack frame is set up, `EBP` should point to the saved copy of its previous value; while `ESP` points to the top value on the stack (the saved version of `EBX`)*

High addresses	
	`v2 = 2`
	`v1 = 10`
	`Saved EIP`
EBP →	`Old EBP`
	`x`
ESP →	`ebx`
Low addresses	

2 (continued)

b. (24 points) A partially completed x86 assembly version of this function is written below. Complete the function by writing the appropriate instructions in the blank spaces provided. The comments next to each blank or instruction describe the purpose of that instruction.

The C version of the function is provided below for your reference. Note that a variable of type `int` is a 32-bit signed integer.

```
int f(int v1, int v2) {
   int x = v1 – v2;
   return (x << 4) – x;
}
```

```
f  PROC                                ; Start of function f
   push    ebp                         ; Save ebp
   mov     ebp, esp                    ; Copy ebp to esp

   sub     esp, 4                      ; Create space on stack for x

   push    ebx                         ; Save ebx on the stack

   mov     ebx, DWORD PTR 8[ebp]       ; ebx = v1

   sub     ebx, DWORD PTR 12[ebp]      ; ebx = v1 - v2

   mov     DWORD PTR -4[ebp], ebx      ; x = ebx = v1 - v2 (copy ebx
                                       ;    to memory location for x)
   sll     ebx, 4                      ; ebx = ebx << 4 = x << 4

   sub     ebx, DWORD PTR -4[ebp]      ; ebx = ebx – x = (x << 4) - x

   pop     ebx                         ; Restore ebx

   mov     esp, ebp                    ; Clear space allocated for
    (or add esp, 4)                    ;     local variable
   pop     ebp                         ; Restore ebp

   ret                                 ; Return from subroutine
f  ENDP
```

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

For each part of this problem, write a short x86 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 (X > 10) {
   if (Y > 0) {
      DX = X;
   }
   else {
      DX = Y;
   }
}
else
   DX = 0;
```

Solution:

```
     CMP  X, 10           ; If Y is not less than 10,
     JLE  L2              ;   go to L2 (outer "else" case)
     CMP  Y, 0            ; Outer if case: test Y
     JLE  L1              ; If Y is not greater than 0, go to L1
                          ;   (inner "else" case)
     MOV  DX, X           ; Inner "if" case: DX = X
     JMP  L3              ; Skip to end of statement
L1:  MOV  DX, Y           ; Inner "else" case: DX = Y
     JMP  L3              ; Skip to end of statement
L2:  MOV  DX, 0           ; Outer "else" case: DX = 0
L3:  ...                  ; End of statement
```

3 (continued)

b. Implement the following loop. Assume that ARR is an array of 11 16-bit values. The starting address of this array is in the register SI when the loop starts—you can use that register to help you access values within the array.

```
for (i = 0; i <= 10; i++) {
    ARR[i] = ARR[i] + AX;
    AX = BX – ARR[i];
}
```

Solution: Note that this solution assumes that CX holds the value of "i".

```
     MOV  CX, 0          ; Initialize CX = i = 0
L:   CMP  CX, 10         ; If CX is not less than or equal to 10
     JG   EXITL          ;    exit loop
     ADD  [SI+2*CX], AX  ; ARR[i] = ARR[i] + AX
     MOV  AX, BX         ; AX = BX
     SUB  AX, [SI+2*CX]  ; AX = BX – ARR[i]
     INC  CX             ; i++
     JMP  L              ; Return to start of loop
EXITL: ...               ; Label at end of loop
```

3 (continued)

c. Implement the following loop. As in part (a), assume “X” and “Y” are 16-bit variables in memory that can be accessed by name.

```
while (X < Y) {
   X = X + 1;
   Y = Y – 1;
}
```

Solution:

```
L:   MOV  AX, X          ; AX = X
     CMP  AX, Y          ; Compare X to Y; if X is not less than
     JGE  EXITL          ;    Y, exit loop
     INC  X              ; X = X + 1
     DEC  Y              ; Y = Y + 1
     JMP  L              ; Return to start of loop
EXITL: ...               ; End of loop
```