16.317: Microprocessor Systems Design I

Fall 2014

Exam 2 Solution

1. (16 points, 4 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 interrupts and exceptions are true?

A. All external interrupts can be disabled so that an interrupt service routine is not called when an external device asserts an interrupt input pin.

B. An interrupt vector is the starting address of an interrupt service routine. Interrupt vectors are typically stored in a table located in memory.

C. When an interrupt occurs, the interrupt service routine is responsible for saving the processor state (all registers, including the flags).

D. Most processors have both dedicated interrupt vectors for specific types of interrupts and general vectors to be used with user-defined interrupts.

i. Only A

ii. Only B

iii. A and C

iv. ***B and D***

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

1 (continued)

b. If the working register is set to `0x10` and a file register, `x`, is set to `0x08`, what is the result of the instruction `subwf x, W`?

- i. `W = 0x08`
- ii. `x = 0x08`
- ***iii.*** ***`W = 0xF8`***
- iv. `x = 0xF8`

c. Which of the following instructions will set the zero bit (Z) to 1 if the file register `x` is equal to 0?

- A. `movwf   x`
- B. `movf    x, F`
- C. `bcf     x, 0`
- D. `addlw   0x00`

- i. Only A
- ***ii.*** ***Only B***
- iii. A and B
- iv. A, B, and C
- v. A, B, C, and D

1 (continued)

d. Which of the following instructions can always be used to set (in other words, change to 1) the upper four bits of the working register, W, while leaving the lower four bits of the register unchanged?

i. `clrw`

ii. `sublw   0x0F`

iii. ***`iorlw   0xF0`***

iv. `xorlw   0x0F`

v. `andlw   0xF0`

2. (16 points) ***Reading PIC assembly***

Show the result of each PIC 16F1829 instruction in the sequences below. Be sure to show the state of the carry (C) bit for any shift or rotate operations.

```
cblock 0x75
   x
endc
```

`movlw 0x79`	**W = 0x79**
`movwf x`	**x = W = 0x79**
`comf x, W`	**W = ~x = ~0x79** **= ~(0111 1001$_2$) = 1000 0110$_2$ = 0x86**
`lslf x, F`	**x = x << 1 = 0x79 << 1** **= 0111 1001$_2$ << 1 = 1111 0010$_2$ = 0xF2** **C = bit shifted out = 0**
`addwf x, W`	**W = x + W = 0xF2 + 0x86 = 0x78**
`btfss x, 6`	**Skip next instruction if bit 6 of x = 1** **→ x = 0xF2 = 1111 0010$_2$ → bit 6 = 1 → skip**
`sublw 0x99`	**Instruction skipped**
`xorwf x, F`	**x = x XOR W = 0xF2 XOR 0x78** **= 1111 0010$_2$ XOR 0111 1000$_2$** **= 1000 1010$_2$ = 0x8A**

3. (28 points) ***Subroutines; HLL → assembly***

The following questions deal with the register and memory contents shown below. Note that:

- These values represent the state of the registers and stack immediately after the stack frame has been set up for the current function.
- The values shown in memory make up the entire stack frame for the current function.

EAX: 0x0000ABBA
EBX: 0x00001234
ECX: 0x00005099
EDX: 0xFFFFFFFF
ESI: 0x11340550
EDI: 0x11340590
ESP: 0x11320140
EBP: 0x1132014C

Address	
0x11320140	0x00000005
0x11320144	0x0000000A
0x11320148	0xFFFF0000
0x1132014C	0x11320164
0x11320150	0x20010550
0x11320154	0x00000002
0x11320158	0x08675309
0x1132015C	0x00000088
0x11320160	0x00197800

a. (3 points) What is the return address for this function?

As shown in the stack frame diagram given with the exam, the return address ("old %EIP") for the function is stored at address EBP+4, which, in this case, is 0x11320150. Therefore, the return address for the function is 0x20010550.

b. (5 points) How many arguments does this function take, and what are their values? Indicate which of the arguments is passed to the function first (for example, when calling a function `f(13, 14, 15)`, the value 13 is passed first). Assume each argument uses four bytes.

The arguments to the function are stored at the highest addresses within the frame, starting at address EBP+8 (0x11320154). Since the problem specifies that the entire stack frame is shown, we can say that there are 4 arguments to this function, with values 0x00000002, 0x08675309, 0x00000088, and 0x00197800. The argument at the highest address is passed first—0x00000002.

3 (continued)

c. (5 points) If we assume that each local variable uses four bytes, how many local variables are declared in this function? Explain your answer.

The lowest addresses within the stack frame contain two types of data: saved registers and local variables. You are told that the values shown represent the state of the stack immediately after the stack frame has been set up. Since none of the topmost values on the stack match the register values, we can infer that this function saves no registers. Therefore, all values with lower addresses than the base pointer are local variables. Since EBP = 0x1132014C and ESP = 0x11320140, a difference of 12 bytes, there are 12 / 4 = 3 local variables in this function.

d. (15 points) A partially completed x86 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. Assume that the function takes two arguments (`v1` and `v2`, in that order) and contains a single local integer variable, `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

  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


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

  ret                              ; Return from subroutine
f ENDP
```

4. (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.**

Note also that your solutions to this question will be short sequences of code, not subroutines. **You do not have to write any code to deal with the stack when solving these problems.**

a. Implement the following conditional statement. You may assume that "`X`", "`Y`", and "`Z`" 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 the register `AX`).

```
if ((X == AX) && (Y == BX) {
  Z = Z / 2;
}
else {
  Z = Z * 4;
  if (Z > X)
     X = Z;
}
```

Solution: *Other solutions may be valid.*

```
     CMP  X, AX
     JNE  L1                ; Jump to else if X != AX
     CMP  Y, BX
     JNE  L1                ; Jump to else if Y != BX
     SHR  Z, 1              ; if case: Z >> 1 is same as Z / 2
                            ; Use SAR if assuming Z signed
     JMP  L2                ; Skip else case
L1:  SHL  Z, 2              ; else case: Z << 2 same as Z * 4
     MOV  DX, Z             ; DX = Z
     CMP  DX, X             ; Compare Z to X (since DX = Z)
     JLE  L2                ; Jump if Z is not greater than X
     MOV  X, DX             ; X = DX = Z
L2:                         ; End of statement
```

4 (continued)

b. Implement the following loop. Assume that `ARR` is an array of forty 32-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 = 40; i > 0; i = i - 2) {
    ARR[i-1] = ARR[i-2] + i;
    ARR[i-2] = ARR[i-1] – i;
}
```

Solution: *Other solutions may be correct. Note that the second line in the body of the loop—and therefore the associated instructions—is completely unnecessary, as it simply overwrites ARR[i-2] with its original value, since ARR[i-1] = ARR[i-2] + i, making ARR[i-1] – i equal to (ARR[i-2] + i) – i = ARR[i-2]*

```
     MOV  ECX, 40              ; Let ECX = i; initialize to 40
L:   LEA  EDX, [SI+4*ECX]      ; EDX = address of ARR[i]
     MOV  EAX, [EDX-8]         ; EAX = ARR[i-2]
     ADD  EAX, ECX             ; EAX = ARR[i-2] + i
     MOV  [EDX-4], EAX         ; ARR[i-1] = ARR[i-2] + 1

     ; Note: next 2 instructions unnecessary as described above
     SUB  EAX, ECX             ; EAX = ARR[i-1] – i
     MOV  [EDX-8], EAX         ; ARR[i-2] = ARR[i-1] – i
     SUB  ECX, 2               ; i = i – 2
     JNZ  L                    ; Return to start of loop if i > 0
```

4 (continued)

c. Implement the following loop. As in part (a), assume “`X`”, “`Y`”, and “`Z`” are 16-bit variables in memory that can be accessed by name. Recall that a while loop is a more general type of loop than the for loop seen in part (b)—a while loop simply repeats the loop body as long as the condition tested at the beginning of the loop is true.

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

Solution: *Other solutions may be correct.*

```
   MOV  DX, Z          ; Z = DX
L: MOV  AX, X          ; AX = X
   CMP  AX, Y          ; Compare X & Y
   JL   FIN            ; Jump to end if X < Y
   ADD  Y, DX          ; Y = Y + DX = Y + Z
   DEC  Y              ; Y = Y – 1 = Y + Z – 1
   SUB  X, DX          ; X = X – DX = X – Z
   INC  X              ; X = X + 1 = X – Z + 1
   JMP  L              ; Return to start of loop
FIN:    ...            ; End of statement
```