16.317: Microprocessor Systems Design I

Fall 2015

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.

Please note that all of the multiple choice questions deal with PIC 16F1829 instructions.

a. If a file register, `x`, is set to `0x31`, and the working register, `W`, is set to `0x70`, what values do those registers hold after executing the instruction `swapf x, W`?

   i. `x = 0x13, W = 0x07`

   ii. `x = 0x13, W = 0x70`

   iii. `x = 0x31, W = 0x07`

   iv. ***`x = 0x31, W = 0x13`***

   v. `x = 0x70, W = 0x31`

1 (continued)

b. Which of the following code snippets will <u>not</u> jump to the label L if `x = 0xFF`?

A.
```
btfss    x, 0
goto     L
```

B.
```
btfsc    x, 7
goto     L
```

C.
```
decfsz   x, F
goto     L
```

D.
```
incfsz   x, F
goto     L
```

i. Only A

ii. Only B

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

iv. B and C

v. A and C

1 (continued)

c. Which of the following instructions will set the carry bit (C) to 0 if the file register `x` is equal to 0x0F, the working register is equal to 0x10, and the carry bit is initially 1?

A. `subwf   x, F`

B. `rlf     x, F`

C. `asrf    x, F`

D. `bcf     x, 0`

i. Only A

ii. Only B

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

iv. A, B, and C

v. A, B, C, and D

d. Which of the following instructions can always be used to decrement the working register, `W`, by 1?

i. `decf    x, W`

ii. `sublw   1`

iii. `subwf   x, W`

iv. `addlw   -1`

v. All of the above (i, ii, iii, and iv)

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.

```
a. cblock 0x70
       x
   endc

   movlw   0x0F           W = 0x0F

   clrf    x              x = 0

   subwf   x, F           x = x - W = 0 - 0x0F = 0xF1

   xorlw   0x36           W = W XOR 0x36 = 0x0F XOR 0x36 = 0x39

   andwf   x, W           W = x AND W = 0xF1 AND 0x39 = 0x31

   lslf    x, F           x = x << 1
                            = 0xF1 << 1 = 1111 0001 << 1
                            = 1110 0010 = 0xE2
                          C = bit shifted out = 1

   btfsc   STATUS, C      Skip next inst. if C == 0 → don't skip

   comf    x, F           x = ~x (flip all bits of x)
                            = ~0xE2 = ~(1110 0010)
                            = 0001 1101 = 0x1D
```

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 some registers and memory locations immediately after the stack frame has been set up for the current function.
- The entire stack frame for the current function is shown, but there may be some additional data stored in the given address range—do not assume that the values shown in memory represent only the contents of the current stack frame.
- The last four instructions executed before entering the body of the current function (which are not the last four instructions executed to set up the stack frame) are:

```
push edx
push ecx
push ebx
call f
```

EAX:	0x0000ABBA
EBX:	0x00001400
ECX:	0x09090909
EDX:	0xFF000000
ESI:	0x11340550
EDI:	0x11340590
ESP:	0x40120154

Address	
0x40120150	0x00000005
0x40120154	0x0000000A
0x40120158	0xFFFF0000
0x4012015C	0x40120200
0x40120160	0x3170F000
0x40120164	0x00001400
0x40120168	0x09090909
0x4012016C	0xFF000000
0x40120170	0x192610AA

a. (5 points) What is the return address for this function? Explain your answer.

Solution: *Knowing the instructions executed before the function call can help you find the return address. We see that the values of the function arguments (`edx`, `ecx`, and `ebx`) are on the stack at addresses 0x4012016C, 0x40120168, and 0x40120164, respectively. The next value in the stack therefore must be the return address, which is pushed when the call instruction is executed. That address is the value stored at address 0x40120160: 0x3170F000.*

b. (4 points) What value does the base pointer (EBP) hold in this function? Explain your answer.

Solution: *The base pointer points to the location just above the saved return address—the location where the previous function's base pointer is stored. Since the return address is stored at 0x40120160, the base pointer must hold the next address: 0x4012015C.*

3 (continued)

c. (4 points) If we assume that each local variable uses four bytes, and also assume that the function saves no registers, how many local variables are declared in this function? Explain your answer.

Solution: *We know that the top of the stack is at address 0x40120154, since we're given the value of ESP. The local variables are stored between the top of the stack and the old base pointer (which is at 0x4012015C, as discussed in (b)), so there are* ***2 local variables*** *stored in those 8 bytes.*

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 one argument, `a1`, and contains one local integer variable, `v1`.

```
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 v1

   mov     eax, DWORD PTR 8[ebp]       ; eax = a1


   add     eax, 10                     ; eax = eax + 10 = a1 + 10

   mov     DWORD PTR -4[ebp], eax      ; v1 = eax = a1 + 10 (copy eax
                                       ;    to memory location for v1)

   sub     DWORD PTR -4[ebp], 20       ; v1 = v1 – 20 = a1 - 10


   idiv    DWORD PTR -4[ebp]           ; eax  = eax / v1
                                       ;      = (a1 + 10) / (a1 – 10)
                                       ; (use signed division; ignore
                                       ;  remainder)

   mov     esp, ebp                    ; Clear space allocated for
                                       ;     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. ***(See original exam for full problem description.)***

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). Your solution should not modify AX or BX.

```
if (AX >= 40) {
   Z = X - Y;
}
else {
   Z = X + Y;
   if (Z > 0)
      X = BX * 8;
   else
      X = BX / 4;
}
```

Solution: *Other solutions may be valid.*

```
   MOV  DX, X                        ; Set Z = X using two MOV
   MOV  Z, DX                        ;   instructions
                                     ; Will either add or subtract
                                     ;   Y later
   CMP  AX, 40                       ; Jump to else case if
   JL   else                         ;   !(AX >= 40) (if AX < 40)
   MOV  DX, Y                        ; Subtract Y from X (since
   SUB  Z, DX                        ;   Z = X before the SUB)
   JMP  done                         ; Skip else case
else:
   MOV  DX, Y                        ; Add Y to X (since Z = X
   ADD  Z, DX                        ;   before the ADD)
   MOV  X, BX                        ; Set X = BX (since X will be
                                     ;   either BX * 8 or BX / 4)
   CMP  Z, 0                         ; If Z <= 0, jump to inner
   JLE  else2                        ;   else case
   SLL  X, 3                         ; X = BX << 3 = BX * 2^3
   JMP  done                         ; Skip inner else case
else2:
   SRA  X, 2                         ; X = BX >> 2 = BX / 2^2
done:                                ; End of code
```

4 (continued)

b. Implement the following loop. Assume that `ARR` is an array of forty 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 = 39; i > 1; i = i - 2) {
   AX = ARR[i-1] + ARR[i-2];
   ARR[i] = AX - ARR[i];
}
```

<u>Solution:</u> *Other solutions may be valid.*

```
   MOV  CX, 39            ; Initialize loop counter (CX is i)
L: LEA  BX, [SI+2*CX]     ; BX = address of ARR[i]
   MOV  AX, [BX-2]        ; AX = ARR[i-1]
   ADD  AX, [BX-4]        ; AX = ARR[i-1] + ARR[i-2]
   SUB  AX, [BX]          ; AX = AX – ARR[i] (OK to overwrite
                          ;   AX since you'll calculate a new
                          ;   value for it in the next iteration
   MOV  [BX], AX          ; ARR[i] = AX – ARR[i]
   SUB  CX, 2             ; CX = i – 2
   CMP  CX, 1
   JG   L                 ; Return to start of loop if i > 1
```

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. Your solution should not modify AX.

```
while ((Y > 0) && (X < 0)) {
   X = X + Z;
   Y = Y - X;
   Z = Z + AX;
}
```

Solution: *Other solutions may be valid.*

```
L: CMP  Y, 0            ; Exit loop if !(Y > 0) → if (Y <= 0)
   JLE  done
   CMP  X, 0            ; Exit loop if !(X < 0) → if (X >= 0)
   JGE  done
   MOV  DX, Z           ; DX = Z
   ADD  X, DX           ; X = X + DX = X + Z
   MOV  CX, X           ; CX = X
   SUB  Y, CX           ; Y = Y - CX = Y - X
   ADD  Z, AX           ; Z = Z + AX
   JMP  L               ; Return to start of loop
done:                   ; End of code
```