16.317: Microprocessor Systems Design I

Spring 2014

Exam 2 Solution

1. (16 points, 4 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. Which of the following statements about interrupts and exceptions are true?

A. An interrupt vector is a function used to handle an interrupt.

B. When an interrupt occurs, the processor saves all processor state (registers and flags), stores the return address (the address of the next instruction) on the stack, and then actually handles the interrupt.

C. Exceptions and interrupts both cause the currently running program to pause until the interrupt handling is complete.

D. If multiple devices share the same interrupt input line, the only possible way to determine which one caused an interrupt is a software solution in which a function checks each device.

i. Only A

ii. Only B

iii. A and D

iv. <u>B and C</u>

v. B, C, and D

b. Which of the following PIC instructions allows you to subtract the constant value 3 from the current contents of the working register?

i. `sublw 3`

ii. `subwf 3, W`

iii. `subwfc 3, W`

iv. ***<u>`addlw -3`</u>***

c. Given a file register `x`, which of the following PIC instructions will set the least significant bit of `x` to `0` if, initially, `x = 0x0F` and `C = 0`?

A. `bcf  x, 0`

B. `lslf x, F`

C. `rlf  x, F`

D. `bsf  x, 0`

i. Only A

ii. A and B

iii. ***<u>A, B, and C</u>***

iv. A, B, C, and D

v. B, C, and D

d. Which of the following instructions can <u>always</u> be used to clear the lowest four bits of the working register, W, while leaving the upper four bits of the register unchanged?

i. `clrw`

ii. `sublw   0x0F`

iii. `iorlw   0xF0`

iv. `xorlw   0x0F`

v. ***<u>`andlw   0xF0`</u>***

2. (16 points) ***Rotate, bit test, and bit scan instructions***
For each instruction in the sequence shown below, list all changed registers and/or memory locations and their new values. If memory is changed, be sure to explicitly list **all changed bytes**. Where appropriate, you should also list the state of the carry flag (CF) and zero flag (ZF).

Initial state:
EAX: 000000E7h
EBX: 00000033h
ECX: 00000002h
EDX: 00000000h
CF: 0
DS: 7230h

Address	Lo			Hi
72300h	C0	00	02	10
72304h	10	10	15	5A
72308h	89	01	05	B1
7230Ch	20	40	AC	DC
72310h	04	08	05	83

Instructions:

```
ROR    AL, CL
```

AL = AL rotated right by CL = E7h rotated right by 2
= $1110\ 0111_2$ rotated right 2 = $\underline{1111\ 1001_2 = F9h}$
CF = last bit rotated = $\underline{1}$

```
BTC    BL, 1
```

CF = bit 1 of BL → BL = 33h = $0011\ 00\underline{1}1_2$ → $\underline{CF = 1}$
Complement bit 1 of BL → BL = $0011\ 00\underline{0}1_2$ → $\underline{31h}$

```
RCR    AL, 4
```

AL = AL rotated right through carry by 4
(AL,CF) = $1111\ 1001\ 1_2$
After rotate, (AL,CF) = $0011\ 1111\ 1_2$ → $\underline{AL = 3Fh,\ CF = 1}$

```
BSR    DX, BX
```

Since BX is not 0, $\underline{ZF = 1}$
DX = position of first non-zero bit in BX, scanning from MSB to LSB
→ BX = 31h = $00\underline{1}10001_2$ → $\underline{DX = 0005h}$

3. (32 points) ***Subroutines; HLL → assembly***

The following questions deal with the simple C function shown below, which takes three integer arguments (`v1`, `v2`, and `v3`), contains one local variable (`x`) and returns the value shown:

```
int f(int v1, int v2, int v3) {
   int x = v1 + v2;
   return (x + v3) * (x – v3);
}
```

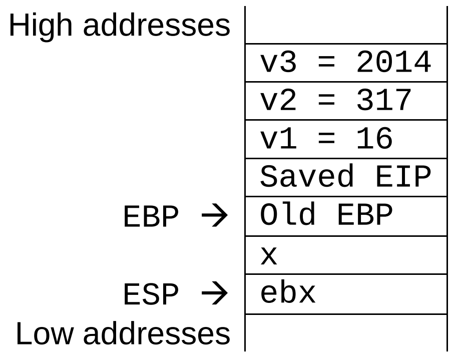
a. (14 points) Draw the stack frame for this function if it is called with 16, 317, and 2014 as its arguments (in other words, a program contains the function call `f(16, 317, 2014)`). Be as specific as possible—in particular:

- 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`)*



3 (continued)

b. (18 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 v3) {
   int x = v1 + v2;
   return (x + v3) * (x – v3);
}
```

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

   sub     esp, 4                    ; Create space on the stack for
                                     ;   local variable x

   push    ebx                       ; Save ebx on the stack

   mov     ebx, 8[ebp]               ; ebx = v1

   add     ebx, 12[ebp]              ; ebx = v1 + v2

   mov     -4[ebp], ebx              ; x = ebx = v1 + v2

   mov     eax, ebx                  ; eax = ebx = x

   add     eax, 16[ebp]              ; eax = eax + v3 = x + v3

   sub     ebx, 16[ebp]              ; ebx = ebx – v3 = x – v3

   imul    ebx                       ; (edx,eax) = eax * ebx
                                     ;   = (x + v3) * (x – v3)

   pop     ebx                       ; Restore ebx

   mov     esp, ebp                  ; Clear space for x
   pop     ebp                       ; Restore ebp
   ret                               ; Return from subroutine
f  ENDP
```

4. (36 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" 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 the register AX).

```
if (X < AX) {
  Y = Y + 10;
}
else if (X > AX) {
  Y = X + 10;
}
else {
  Y = X;
}
```

Solution:

```
     CMP  X, AX     ; Compare X to AX—can use comparison results
                    ;   multiple times
     JL   L1        ; If (X < AX), go to L1 (if case)
     JG   L2        ; If (X > AX), go to L2 (else if case)
     MOV  BX, X     ; Else case: BX = X
     MOV  Y, BX     ; Y = BX = X
     JMP  FIN       ; Skip if and else if cases

L1:  ADD  Y, 10     ; If case: Y = Y + 10
     JMP  FIN       ; Skip else if case

L2:  MOV  BX, X     ; Else if case: BX = X
     MOV  Y, BX     ; Y = BX = X
     ADD  Y, 10     ; Y = X + 10

FIN: ...            ; End of statement
```

3 (continued)

b. Implement the following loop. Assume that ARR is an array of twenty-one 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 < 21; i = i+3) {
   AX = ARR[i] + ARR[i+1];
   ARR[i+2] = AX + BX;
}
```

Solution:

```
     MOV  CX, 0          ; CX = i = 0
L:   CMP  CX, 21         ; If i is not less than 21,
     JGE  FIN            ;   loop is done—exit
     MOV  AX, [SI+2*CX]  ; AX = ARR[i]
     INC  CX             ; i = i + 1
     ADD  AX, [SI+2*CX]  ; AX = ARR[i] + ARR[i+1]
     INC  CX             ; i = i + 1 = original i + 2
     MOV  [SI+2*CX], AX  ; ARR[i+2] = AX
     ADD  [SI+2*CX], BX  ; ARR[i+2] = AX + BX
     JMP  L              ; Return to start of loop
FIN: ...                 ; End of 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! Also, note that the || symbol indicates a logical OR, and the && symbol indicates a logical AND.)

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

Solution:

```
     MOV  DX, X      ; DX = X
     CMP  DX, Y      ; If DX < Y,
     SETL CL         ;   CL = 1
     CMP  BX, DX     ; If BX == DX (BX == X)
     SETE CH         ;   CH = 1
     OR   CL, CH     ; CL = 1 if (X < Y || BX == X)
     CMP  AX, Y      ; If AX < Y,
     SETL DL         ;   DL = 1
     CMP  BX, X      ; If BX > X,
     SETG DH         ;   DH = 1
     AND  DL, DH     ; DL = 1 if (AX < Y && BX > X)
     OR   CL, DL     ; CL = 1 if (X < Y || BX == X ||
                     ;          (AX < Y && BX > X))
     JZ   FIN        ; Skip addition if CL == 0 (condition false)
     ADD  AX, BX     ; AX = AX + BX
FIN: ...             ; End of statement
```