16.317: Microprocessor Systems Design I

Spring 2015

Homework 4 Solution

1. *(40 points) Write the following subroutine in x86 assembly:*

    `int fib(int n)`

Given a single integer argument, `n`*, return the* `n`*th value of the Fibonacci sequence—a sequence in which each value is the sum of the previous two values. The first 15 values are shown below—note that the first value is returned if n is 0, not 1.*

`n`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`fib(n)`	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

Solution: How you implement the low-level code for this version of the Fibonacci function depends on the algorithm you use. What follows is both C code and assembly for the algorithm implemented either with or without recursion.

```
int fib(int n) {    // FIBONACCI WITHOUT RECURSION
     int i;               // Loop index
     int first, sec;      // Two previous Fibonacci values
     int cur;             // Value from current iteration

     // For n == 0 or n == 1, fib(n) == n
     if (n <= 1)
          return n;

     // Use loop to calculate fib(n)--at each step,
     //   current value is sum of previous two values
     else {
          first = 0;
          sec = 1;
          for (i = 2; i <= n; i++) {
               cur = first + sec;
               first = sec;
               sec = cur;
          }
          return cur;
     }
}
```

```
fib       PROC                        ; Start of subroutine
  push    ebp                         ; Save ebp
  mov     ebp, esp                    ; Copy ebp to esp
  sub     esp, 8                      ; Create space for first,
                                      ;   sec (cur, if needed,
                                      ;   will be in eax)
  push    ebx                         ; Save ebx and ecx (both
  push    ecx                         ;  (overwritten in fn)

; CODE FOR: if (n <= 1) return n
  cmp     DWORD PTR 8[ebp], 1         ; Compare n to 1
  jg      L1                          ; If n isn't <= 1, jump
                                      ;   to else case
  mov     eax, DWORD PTR 8[ebp]       ; eax = n (eax holds
                                      ;   return value)
  jmp     L3                          ; Jump to end of function

; CODE FOR: first = 0; sec = 1
L1:
  mov     DWORD PTR -4[ebp], 0        ; first = 0
  mov     DWORD PTR -8[ebp], 1        ; sec = 1

; CODE FOR: loop initialization
; Note that the loop will execute n – 1 iterations, so we
;   can initialize ECX to n - 1 and use loop instructions
  mov     ecx, DWORD PTR 8[ebp]       ; cx = n
  dec     ecx                         ; cx = cx – 1 = n – 1

; CODE FOR: cur = first + sec; first = sec; sec = cur
L2:
  mov     eax, DWORD PTR -4[ebp]      ; cur = eax = first
  add     eax, DWORD PTR -8[ebp]      ; cur = first + sec
  mov     ebx, DWORD PTR -8[ebp]      ; ebx = sec
  mov     DWORD PTR -4[ebp], ebx      ; first = ebx = sec
  mov     DWORD PTR -8[ebp], eax      ; sec = eax = cur

; CODE FOR: decrement loop counter & go to start of loop
  loop    L2

; CLEANUP (NOTE: No additional code needed for return cur
;   in else case, since cur is already stored in eax)
L3:
  pop     ecx                         ; Restore ecx
  pop     ebx                         ; Restore ebx
  mov     esp, ebp                    ; Clear first, sec
  pop     ebp                         ; Restore ebp
  ret                                 ; Return from subroutine
fib       ENDP
```

```
int fib(int n) {        // FIBONACCI WITH RECURSION

      // For n == 0 or n == 1, fib(n) == n
      if (n <= 1) return n;

      // Otherwise, value is sum of two previous steps
      else return fib(n-1) + fib(n-2);
}

fib         PROC                        ; Start of subroutine
   push     ebp                         ; Save ebp
   mov      ebp, esp                    ; Copy ebp to esp
   push     ebx                         ; Save ebx (overwritten
                                        ;  in function)

; CODE FOR: if (n <= 1) return n
   cmp      DWORD PTR 8[ebp], 1         ; Compare n to 1
   jg       L1                          ; If n isn't <= 1, jump
                                        ;   to else case
   mov      eax, DWORD PTR 8[ebp]       ; eax = n (eax holds
                                        ;   return value)
   jmp      L2                          ; Jump to end of function

; CODE FOR: calling fib(n-1)
L1:
   mov      ebx, DWORD PTR 8[ebp]       ; Copy n to ebx
   dec      ebx                         ; ebx = n - 1
   push     ebx                         ; Push n - 1 to pass it
                                        ;   as argument
   call     fib                         ; Call fib(n-1)
                                        ; Return value in eax

; CODE FOR: calling fib(n-2)
; NOTE: We can take advantage of the fact that n-1 is still
;  on the stack--decrement that value, and we'll have the
;  value n-2 to pass to our next function call
   mov      ebx, eax                    ; ebx = eax = fib(n-1)
   dec      DWORD PTR [esp]             ; Value at top of stack =
                                        ;   (n-1) - 1 = n-2
   call     fib                         ; Call fib(n-2)
                                        ; Return value in eax

; CODE FOR: return fib(n-1) + fib(n-2)
   add      eax, ebx                    ; eax = fib(n-1)+fib(n-2)

; CLEANUP
L2:
   add      esp, 4                      ; Clear argument passed to
                                        ;   fib(n-2)
   pop      ebx                         ; Restore ebx
   pop      ebp                         ; Restore ebp
   ret                                  ; Return from subroutine
fib         ENDP
```

2. (60 points) Show the result of each PIC 16F684 instruction in the sequences below. Be sure to show not only the state of updated registers, but also the carry (C) and zero (Z) bits.

a.
```
cblock 0x20
    x
endc
```

movlw	0x05	**W = <u>0x05</u>**
sublw	0x15	**W = 0x15 – W = 0x15 – 0x05 = <u>0x10</u>**
clrf	x	**x = <u>0x00</u>**
comf	x, F	**x = ~x = ~0x00 = <u>0xFF</u>**
xorwf	x, F	**x = x XOR W = 0xFF XOR 0x10 = <u>0xEF</u>**
swapf	x, W	**W = value in x with nibbles swapped = <u>0xFE</u>**
btfsc	x, 7	**Test bit 7 of x and skip next instruction if bit is 0 → x = 0xEF = $\underline{1}110\ 1111_2$ → bit 7 = 1 → <u>do not skip</u>**
bsf	x, 0	**Set bit 0 of x → x = $1110\ 111\underline{1}_2$ before set → No change, since bit 0 already is 1**

b.

```
cblock 0x20
    A
    B
endc
```

clrf	A	**A = 0x00**
movlw	0x11	**W = 0x11**
movwf	B	**B = W = 0x11**
addlw	0x34	**W = W + 0x34 = 0x11 + 0x34 = 0x45**
subwf	A, F	**A = A – W = 0x00 – 0x45 = 0xBB**
comf	A, W	**W = ~A = ~ 0xBB = 0x44**
swapf	A, F	**Swap nibbles of A → A = 0xBB**

c.

```
cblock 0x40
    var1
endc
```

movlw	0x1E	**W = 0x1E**
movwf	var1	**var1 = W = 0x1E**
rrf	var1, F	**Rotate var1 1 bit right through carry → (var1, C) = 0001 1110 0 rotated right → (var1, C) = 0000 1111 0 → var1 = 0x0F, C = 0**
xorwf	var1, W	**W = var1 XOR W = 0x0F XOR 0x1E = 0000 1111 XOR 0001 1110 = 0001 0001 = 0x11**
btfss	var1, 4	**Test bit 4 of var1; skip next instruction if bit = 1 → var1 = 0x0F = 0000 1111 → Bit is 0 → do not skip**
iorlw	0x06	**W = W OR 0x06 = 0x11 OR 0x06 = 0001 0001 OR 0000 0110 = 0001 0111 = 0x17**
andwf	var1, F	**var1 = var1 AND W = 0x0F AND 0x17 = 0000 1111 AND 0001 0111 = 0000 0111 = 0x07**
bcf	var1, 0	**Clear bit 0 of var1 → var1 = 0x07 = 0000 0111 → After clear, var1 = 0000 0110 = 0x06**

d.
```
cblock 0x70
    num1, num2
endc
```

movlw	0xAA	**W = 0xAA**
andlw	0x0F	**W = W AND 0x0F = 0xAA AND 0x0F = 0x0A**
movwf	num1	**num1 = W = 0x0A**
xorlw	0xFF	**W = W XOR 0xFF = 0x0A XOR 0xFF = 0xF5**
movwf	num2	**num2 = W = 0xF5**
asrf	num2, F	**num2 = num2 >> 1 (keep sign intact)** **= 0xF5 >> 1 = 0x1111 0101 >> 1** **= 0x1111 1010 = 0xFA** **C = bit shifted out = 1**
lslf	num1, W	**W = num1 << 1 = 0x0A << 1** **= 0000 1010 << 1 = 0001 0100 = 0x14** **C = bit shifted out = 0**
xorwf	num2, F	**num2 = num2 XOR W = 0xFA XOR 0x14 = 0xEE**
comf	num2, W	**W = ~num2 (flip all bits of num2)** **= ~0xEE = 0x11**