EECE.3170: Microprocessor Systems Design I

Summer 2017

Homework 4 Solution

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

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

Recall that:

- *Subroutine arguments are passed on the stack, and can be accessed within the body of the subroutine starting at address EBP+8.*
- *At the start of each subroutine:*
  - i. *Save EBP on the stack*
  - ii. *Copy the current value of the stack pointer (ESP) to EBP*
  - iii. *Create space within the stack for each local variable by subtracting the appropriate value from ESP. For example, if your function uses four integer local variables, each of which contains four bytes, subtract 16 from ESP. Local variables can then be accessed starting at the address EBP-4.*
  - iv. *Save any registers the function uses other than EAX, ECX, and EDX.*
- *A subroutine's return value is typically stored in EAX.*

See Lectures 14 and 16-18 for more details on subroutines, the x86 architecture, and the conversion from high-level concepts to low-level assembly.

Solution: Solution is shown on the next page; note that many different solutions are possible. The key points are:

- Setting up the stack frame appropriately (save base pointer; point base pointer to appropriate location; create space for local variable(s); save any overwritten registers except eax).
- Adding v1 + v2 while appropriately accessing different memory locations (only one memory operand per instruction; accessing arguments at right addresses relative to ebp)
- Computing return value while appropriately accessing different memory locations
- "Cleaning up" stack frame (restoring saved registers; clearing space for local variable(s); restoring base pointer)

```
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 x

   push    ebx                         ; Save ebx on the stack
   push    edx                         ; Save edx on the stack

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

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

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

   mov     eax, ebx                    ; eax = ebx = x

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

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

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

   pop     edx                         ; Restore edx
   pop     ebx                         ; Restore ebx

   mov     esp, ebp                    ; Clear x
   pop     ebp                         ; Restore ebp

   ret                                 ; Return from subroutine
f  ENDP
```

2. *(60 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
```