CS 241 - Week 2 Tutorial Solution

Assembly Language Programming

Winter 2015

1 Problem 1 - How to write a MIPS loop

```
; $1 = n
; on exit, $3 = nth fibonacci number

; helpful to have 1 in a register
  lis $2
  .word 1

; $3 is always f_n-2, $4 is always f_n-1
; this means that initially $3 is f_0 and $4 is f_1
  add $3, $0, $0
  add $4, $2, $0

; technically, we actually compute
; up to f_n+1, but we require fewer
; special cases in doing so
loop:
  beq $1, $0, end
  sub $1, $1, $2   ; decrement n
  add $5, $4, $0
  add $4, $3, $4
  add $3, $5, $0
  beq $0, $0, loop

end:
  jr $31
```

2 Problem 2 - How to create and use procedures

```
fib:
```

```
; $1 = n
; on exit, $3 = nth fibonacci number

; there are only 3 registers we need to save
; note that we do NOT save $3 because it is
; an output register
  sw $1, -4($30)
  sw $2, -8($30)
  sw $4, -12($30)
  sw $5, -16($30)
  lis $2
  .word 16
  sub $30, $30, $2

; helpful to have 1 in a register
  lis $2
  .word 1

; $3 is always f_n-2, $4 is always f_n-1
; this means that initially $3 is f_0 and $4 is f_1
  add $3, $0, $0
  add $4, $2, $0

; technically, we actually compute
; up to f_n+1, but we require fewer
; special cases in doing so
loop:
  beq $1, $0, end
  sub $1, $1, $2   ; decrement n
  add $5, $4, $0
  add $4, $3, $4
  add $3, $5, $0
  beq $0, $0, loop

end:
; restore saved registers
; first we undo the stack pointer adjustments,
; then it is as simple as reusing the same offsets
; we used to save
  lis $2
  .word 16
  add $30, $30, $2
```

```
  lw $1, -4($30)
  lw $2, -8($30)
  lw $4, -12($30)
  lw $5, -16($30)
  jr $31
```

3 Problem 3 - Printing to stdout and using the stack

```
; $1 = n

; load addresses of procedures
  lis $5
  .word fib
  lis $6
  .word print

; useful constants
  lis $4
  .word 4
  lis $11
  .word 1

; whenever we call a procedure, we must save/restore $31
  sw $31, -4($30)
  lis $31
  .word 4
  sub $30, $30, $31

; put n somewhere else, since $1 is used in printing
  add $7, $1, $0

; use $8 as a counter, start at f_0
  add $8, $0, $0

; compute each fibonacci number, print,
; and store them onto the stack
forward:
  beq $8, $7, reverse
  add $1, $8, $0
  jalr $5  ; compute f_i

  sw $3, -4($30)
```

```
  sub $30, $30, $4

  ; increment counter
  add $8, $8, $11
  beq $0, $0, forward

; read each number from the stack and print it
reverse:
  beq $8, $0, exit

  add $30, $30, $4
  lw $1, -4($30)

  jalr $6 ; print f_i

  ; decrement counter
  sub $8, $8, $11
  beq $0, $0, reverse

; cleanup and exit
exit:
  lis $31
  .word 4
  add $30, $30, $31
  lw $31, -4($30)
  jr $31
```

4 Problem 4 - Various skills

```
; $1 = array starting address
; $2 = number of elements in array
; on exit, $3 = 1 if array is a Fibonacci sequence, 0 otherwise

; load procedure address
  lis $5
  .word fib

; useful constants
  lis $4
  .word 4
  lis $11
  .word 1
```

```
; whenever we call a procedure, we must save/restore $31
  sw $31, -4($30)
  lis $31
  .word 4
  sub $30, $30, $31

; put array address somewhere else, since $1 is used as a parameter to fib
  add $7, $1, $0

; use $1 as a counter, start at f_0
  add $1, $0, $0

top:
  ; got to the end of the array, so it is a fibonacci sequence
  beq $2, $1, isFibSeq

  ; load next element and compute next fibonacci number
  ; if they are not the same then this is not a fibonacci sequence
  lw $8, 0($7)
  jalr $5
  bne $3, $8, notFibSeq

  add $1, $1, $11
  add $7, $7, $4  ; walk the pointer
  beq $0, $0, top

isFibSeq:
  add $3, $11, $0
  beq $0, $0, exit

notFibSeq:
  add $3, $0, $0

; cleanup and exit
exit:
  lis $31
  .word 4
  add $30, $30, $31
  lw $31, -4($30)
  jr $31
```