Question 4 (25 points):

The first function is called `fun`. Given the value of an integer i, `fun` computes the value of an integer f_i, that is defined by the following equations:

$$\begin{aligned} f_0 &= 1 \\ f_1 &= 2 \\ f_i &= f_{i-2} + (-1)^i \times f_{i-1} \end{aligned} \tag{1}$$

Hint: another way to write the expression for f_i is as follows:

$$f_i = \begin{cases} 1 & \text{if } i = 0 \\ 2 & \text{if } i = 1 \\ f_{i-2} - f_{i-1} & \text{if } i \neq 1 \text{ and } i \text{ is odd} \\ f_{i-2} + f_{i-1} & \text{if } i \neq 0 \text{ and } i \text{ is even} \end{cases} \tag{2}$$

The specification for `fun` is as follows:

- **parameters:**

  `$a0`: i

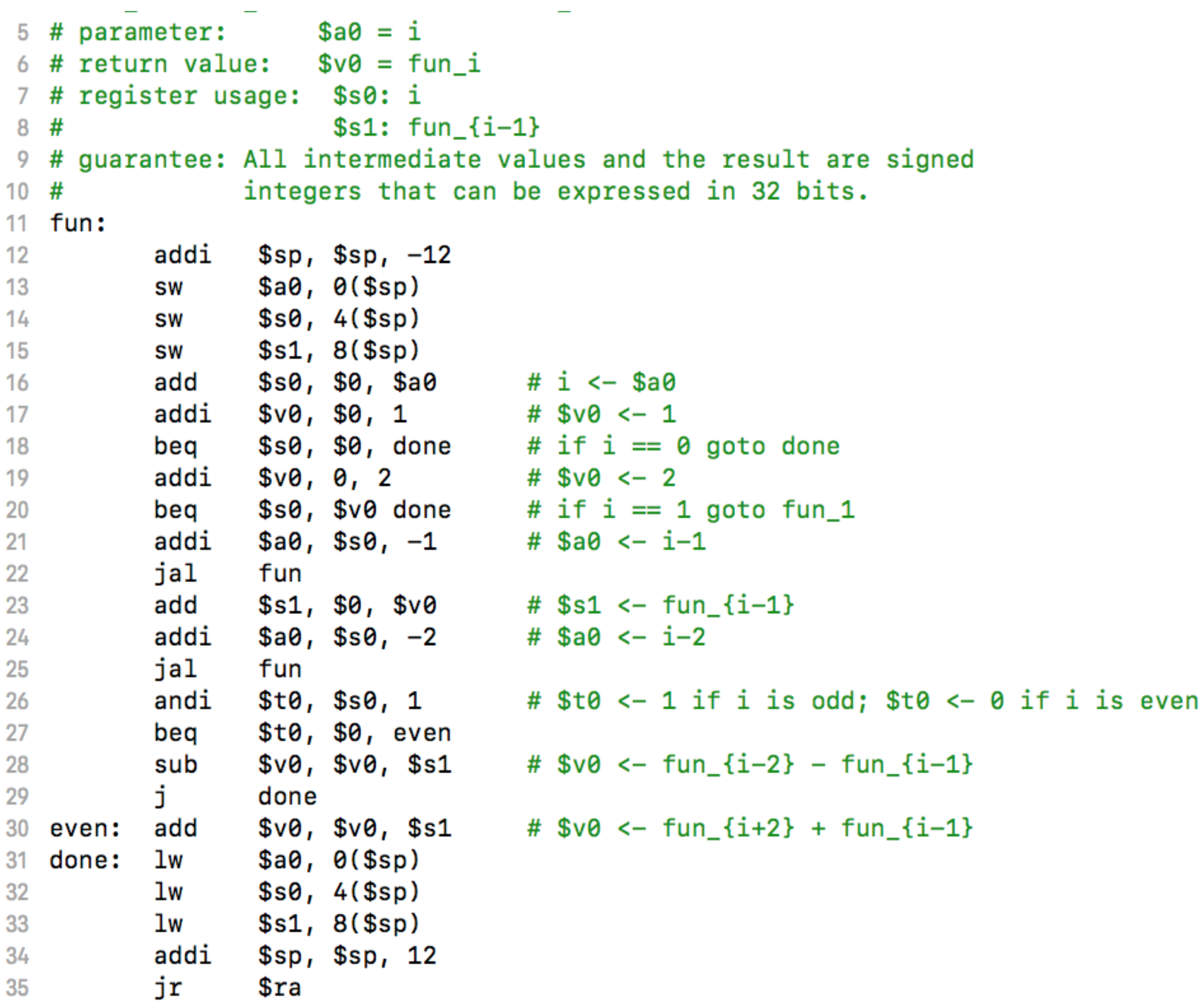
- **return value:**

  - `$v0`: f_i

- **guarantee:**

  - The value of i, all the intermediate values, and of f_i can be expressed as 32-bit integers.

Your implementation of `fun` must follow all the MIPS calling conventions for saving/restoring registers.



```
# parameter:      $a0 = i
# return value:   $v0 = fun_i
# register usage:  $s0: i
#                  $s1: fun_{i-1}
# guarantee: All intermediate values and the result are signed
#            integers that can be expressed in 32 bits.
fun:
       addi   $sp, $sp, -12
       sw     $a0, 0($sp)
       sw     $s0, 4($sp)
       sw     $s1, 8($sp)
       add    $s0, $0, $a0      # i <- $a0
       addi   $v0, $0, 1        # $v0 <- 1
       beq    $s0, $0, done     # if i == 0 goto done
       addi   $v0, 0, 2         # $v0 <- 2
       beq    $s0, $v0 done     # if i == 1 goto fun_1
       addi   $a0, $s0, -1      # $a0 <- i-1
       jal    fun
       add    $s1, $0, $v0      # $s1 <- fun_{i-1}
       addi   $a0, $s0, -2      # $a0 <- i-2
       jal    fun
       andi   $t0, $s0, 1       # $t0 <- 1 if i is odd; $t0 <- 0 if i is even
       beq    $t0, $0, even
       sub    $v0, $v0, $s1     # $v0 <- fun_{i-2} - fun_{i-1}
       j      done
even:  add    $v0, $v0, $s1     # $v0 <- fun_{i+2} + fun_{i-1}
done:  lw     $a0, 0($sp)
       lw     $s0, 4($sp)
       lw     $s1, 8($sp)
       addi   $sp, $sp, 12
       jr     $ra
```

Figure 1: A solution for `fun`.

Question 5 (25 points): The second function that you will write is `maxfun`. Given an integer k, `maxfun` returns the maximum value of f_i in the interval $[0, k]$. The [ ] indicates that the limits of the interval are included in the computation of the maximum. To compute f_i `maxfun` must call the function `fun`. The specification for `maxfun` is as follows:

- **parameters:**

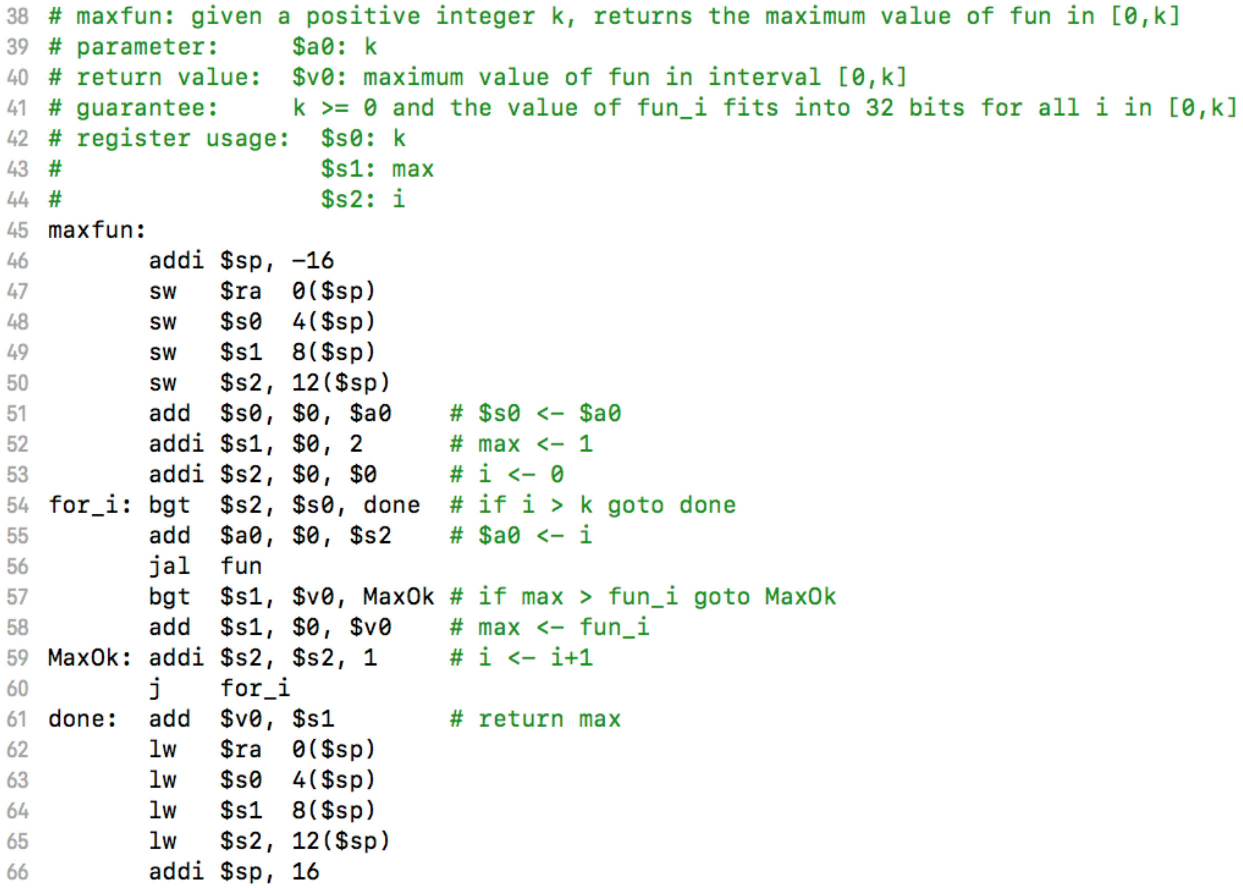
  `$a0`: k

- **return value:**

  `$v0`: maximum value of f_i in the interval $[0, k]$.

- **guarantee**

  – the value of f_i in all points in the interval $[0, k]$ can be expressed as a 32-bit integer.



```
# maxfun: given a positive integer k, returns the maximum value of fun in [0,k]
# parameter:     $a0: k
# return value:  $v0: maximum value of fun in interval [0,k]
# guarantee:     k >= 0 and the value of fun_i fits into 32 bits for all i in [0,k]
# register usage:  $s0: k
#                  $s1: max
#                  $s2: i
maxfun:
       addi $sp, -16
       sw   $ra  0($sp)
       sw   $s0  4($sp)
       sw   $s1  8($sp)
       sw   $s2, 12($sp)
       add  $s0, $0, $a0    # $s0 <- $a0
       addi $s1, $0, 2      # max <- 1
       addi $s2, $0, $0     # i <- 0
for_i: bgt  $s2, $s0, done  # if i > k goto done
       add  $a0, $0, $s2    # $a0 <- i
       jal  fun
       bgt  $s1, $v0, MaxOk # if max > fun_i goto MaxOk
       add  $s1, $0, $v0    # max <- fun_i
MaxOk: addi $s2, $s2, 1     # i <- i+1
       j    for_i
done:  add  $v0, $s1        # return max
       lw   $ra  0($sp)
       lw   $s0  4($sp)
       lw   $s1  8($sp)
       lw   $s2, 12($sp)
       addi $sp, 16
```

Figure 2: A solution to the `maxfun` function.