COE 147 Spring 2014
Lab 6 Solution: 1-bit Adders and Number Representation

Part 1: Programming

```
.data
A:     .space      64
B:     .space      64
C:     .space      64

A_str:       .asciiz     "Please enter the first 16-bit binary number: "
B_str:       .asciiz     "Please enter the second 16-bit binary number: "
C_str:       .asciiz     "Sum is: "
OV_str:      .asciiz     "\nOverflow bit: "

.text
      j main            # DO NOT EDIT THIS LINE

########################
# PLACE YOUR CODE BELOW #
########################

# BitAdder
#     adds two bits with the carry in and outputs the 1-bit sum and carry out for
the next step
# INPUT:
#     BitAdder expects arguments in $a0, $a1, $a2
#     $a0 = specific bit (of values either 0 or 1 in decimal) from A, do not pass
character '0' or '1'
#     $a1 = specific bit (of values either 0 or 1 in decimal) from B, do not pass
character '0' or '1'
#     $a2 = carry in (of values either 0 or 1 in decimal) from previous step
# OUTPUT:
#     $v0 = 1-bit sum in $v0
#     $v1 = carry out for the next stage
BitAdder:
      # prologue
      subi  $sp, $sp, 20
      sw    $s0, 0($sp)
      sw    $s1, 4($sp)
      sw    $s2, 8($sp)
      sw    $s3, 12($sp)
      sw    $ra, 16($sp)

      # body
      add   $s0, $a0, $a1
      add   $s0, $s0, $a2

      li    $s1, 1
      li    $s2, 2
      li    $s3, 3

      li    $v0, 0      # sum
      li    $v1, 0      # carry out
```

```
      beq   $s0, $zero, BIT_ADDR_DONE
      beq   $s0, $s1, SUM_ONE
      beq   $s0, $s2, SUM_TWO
      beq   $s0, $s3, SUM_THREE

SUM_THREE:
      li    $v0, 1      # set sum bit

SUM_TWO:
      li    $v1, 1      # set carry out
      j     BIT_ADDR_DONE

SUM_ONE:
      li    $v0, 1      # set sum

BIT_ADDR_DONE:

      # epilogue
      lw    $s0, 0($sp)
      lw    $s1, 4($sp)
      lw    $s2, 8($sp)
      lw    $s3, 12($sp)
      lw    $ra, 16($sp)
      addi  $sp, $sp, 20

      # return
        jr $ra


# AddNumbers
#     it adds two strings, each of which represents a 16-bit number
# INPUT:
#     $a0 = address of A
#     $a1 = address of B
#     $a2 = address of C
# OUTPUT:
#     $v0 = overflow bit (either 0 or 1 in decimal)
AddNumbers:
      # prologue
      subi        $sp, $sp, 32
      sw          $s0, 0($sp)
      sw          $s1, 4($sp)
      sw          $s2, 8($sp)
      sw          $s3, 12($sp)
      sw          $s4, 16($sp)
      sw          $s5, 20($sp)
      sw          $s6, 24($sp)
      sw          $ra, 28($sp)

      # body
      # loop 16 times for 16 bits
      li    $s6, 16           # counter

      move  $s0, $a0          # s1 points to A
      move  $s1, $a1          # s2 points to B
```

```
      move  $s2, $a2          # s3 points to C

      # start from bit-0
      addi        $s0, $s0, 15
      addi        $s1, $s1, 15
      addi        $s2, $s2, 15

      li    $s3, 0                  # initial carry in

      li    $s4, 0x30               # character 0
      li    $s5, 0x31               # character 1

LOOP:
      lb    $t1, 0($s0)       # load current bit-character from A
      lb    $t2, 0($s1)       # load current bit-character from B

      subi  $a0, $t1, 0x30          # $a0 = bit from A (converted from
bit-character by subtracting 0x30)
      subi  $a1, $t2, 0x30          # $a1 = bit from B (converted from
bit-character by subtracting 0x30)
      move  $a2, $s3          # $a2 = carry in from previous stage

      # add bit-by-bit
      jal BitAdder

      # put '1' or '0' in C based on the sum bit
      beq   $v0, $zero, PUT_ZERO

PUT_ONE:
      sb    $s5, 0($s2) # stores '1' at the current bit position at C (because
$s5='1' and $s2 points to current bit position of C)
      j     MOVE_ONTO_NEXT_BIT

PUT_ZERO:
      sb    $s4, 0($s2) # stores '0' at the current bit position at C (because
$s4='0' and $s2 points to current bit position of C)

MOVE_ONTO_NEXT_BIT:

      # set carry in for the next stage
      move  $s3, $v1

      #  move to the next bit position
      subi        $s0, $s0, 1
      subi        $s1, $s1, 1
      subi        $s2, $s2, 1

      # check if the summation is finished or not
      subi       $s6, $s6, 1
      beq        $s6, $zero, DONE

      # not finished yet
      j     LOOP

      # loop ends
DONE:
```

```
      # put a NULL character at the end of C
      la          $s0, C
      sb          $zero, 16($s0)
      # set $v0 with the overflow
      move  $v0, $v1

      # epilogue
      lw          $s0, 0($sp)
      lw          $s1, 4($sp)
      lw          $s2, 8($sp)
      lw          $s3, 12($sp)
      lw          $s4, 16($sp)
      lw          $s5, 20($sp)
      lw          $s6, 24($sp)
      lw          $ra, 28($sp)
      addi        $sp, $sp, 32

      # return
      jr $ra


#===============================================
#Do NOT edit the rest of the code in this file.
#===============================================

main: #
        jal setRegisterStates

      # print A_str
      la    $a0, A_str
      li    $v0, 4
      syscall

      # read A
      la    $a0, A
      li    $a1, 64
      li    $v0, 8
      syscall

      # print B_str
      la    $a0, B_str
      li    $v0, 4
      syscall

      # read B
      la    $a0, B
      li    $a1, 64
      li    $v0, 8
      syscall

      # clip A and B to 16-character long
      li    $t0, 0x00
      la    $t1, A
      sb    $t0, 16($t1)
      la    $t1, B
```

```
      sb    $t0, 16($t1)

      # call AddNumbers
      la    $a0, A
      la    $a1, B
      la    $a2, C
        jal AddNumbers

      # save overflow bit
      move  $t3, $v0

      # clip C to 16-characters
      li    $t0, 0x00
      la    $t1, C
      sb    $t0, 16($t1)

      # print C_str
      la    $a0, C_str
      li    $v0, 4
      syscall

      # print C
      la    $a0, C
      li    $v0, 4
      syscall

      # print OV_str
      la    $a0, OV_str
      li    $v0, 4
      syscall

      # print overflow
      move  $a0, $t3
      li          $v0, 1
      syscall

      # done
        jal checkRegisterStates

        li $v0, 10          #Exit
        syscall

setRegisterStates:
    li $s0, -1
    li $s1, -1
    li $s2, -1
    li $s3, -1
    li $s4, -1
    li $s5, -1
    li $s6, -1
    li $s7, -1
    sw $sp, old_sp_value
    sw $s0, ($sp)       #Write something at the top of the stack
    jr $ra

checkRegisterStates:
```

```
    bne $s0, -1, checkRegisterStates_failedCheck
    bne $s1, -1, checkRegisterStates_failedCheck
    bne $s2, -1, checkRegisterStates_failedCheck
    bne $s3, -1, checkRegisterStates_failedCheck
    bne $s4, -1, checkRegisterStates_failedCheck
    bne $s5, -1, checkRegisterStates_failedCheck
    bne $s6, -1, checkRegisterStates_failedCheck
    bne $s7, -1, checkRegisterStates_failedCheck

    lw $t0, old_sp_value
    bne $sp, $t0, checkRegisterStates_failedCheck

    lw $t0, ($sp)
    bne $t0, -1, checkRegisterStates_failedCheck

    jr $ra                      #Return: all registers passed the check.

    checkRegisterStates_failedCheck:
        la $a0, failed_check    #Print out the failed register state message.
        li $v0, 4
        syscall

        li $v0, 10              #Exit prematurely.
        syscall

.data
      old_sp_value:   .word 0
      failed_check:   .asciiz "One or more registers was corrupted by your code.\n"
```

Part 2: Written Part

1. Add the following unsigned binary numbers (show the carry and overflow bits)

```
1 1111 1      1
  0010 0110 1001
+ 1111 1100 0101
 --------------
1 0010 0010 1110
^- Overflow
```

2. Subtract the following unsigned binary numbers (show the borrow and underflow bits). Do not convert to two's-complement.

2
12 0 2 0 1~~0~~2
~~00~~1~~1~~ ~~0~~11~~1~~ ~~010~~1

```
- 1110 1000 1110
 --------------
? 0100 1110 0111
^- Underflow
```

3. Convert the following decimal numbers to binary numbers (represent each as a 16-bit number):

```
  1639: 0000011001100111
 48265: 1011110010001001
  1010: 0000001111110010
```

4. Convert the following unsigned binary numbers to decimal numbers:

```
Number 1: 10000001 01011110 = 33118
Number 2: 00000110 01010011 =  1619
```

5. Convert the following decimal numbers into 9-bit binary numbers (with sign-magnitude):

```
  48: 000110000
-126: 101111110
 -34: 100100010
```

6. Convert the following 9-bit binary numbers (with sign-magnitude) to decimal numbers:

```
010011110:  158
100110111:  -55
110101010: -170
```

7. Convert the following decimal numbers into 9-bit binary numbers in 1's complement form:

```
  56: 000111000
-145: 101101110
 -52: 111001011
```

8. Convert the following 8-bit binary numbers in 1's complement to decimal numbers:

```
01010011:  83
11010010: -45
11110111:  -8
```

9. Convert the following decimal numbers into 9-bit binary numbers in 2's complement form:

```
196: 011000100
```

```
-17: 111101111
-95: 110100001
```

10. Convert the following 8-bit binary numbers in 2's complement to decimal numbers:

```
01010101:  85
10111101: -67
11010000: -48
```