Scot Matson
CS47-01
Patra
Homework 2

1. FIBONACCI

```
.data
prompt:         .asciiz "Enter a positive integer to indicate how many Fibonacci
numbers to store in the stack..\n"
cursor:         .asciiz ">> "
endMsg:         .asciiz     "Sequence Completed.\n"
error:          .asciiz "A Fibonacci sequence must have at least 1 value. No
numbers were pushed to the stack."
result:         .asciiz "\n**Printing Fibonacci sequence from the stack.**\n"
delimit:    .asciiz " "

.text
main:
      # User prompt.
      li    $v0, 4
      la    $a0, prompt
      syscall

      li    $v0, 4
      la    $a0, cursor
      syscall

      # Get user input.
      li    $v0, 5
      syscall

      # Start fibonacci procedure
      move  $s0, $v0
      jal fibonacci

      # Notify user of completed sequence.
      li    $v0, 4
      la    $a0, result
      syscall

      # Display the sequence.
      #sw   $a1, index
      jal   stackReader

      # Exit the application
      li    $v0, 10
      syscall

#-----------------------------------------------------------------------------------#
fibonacci:
      ble   $s0, 0, fibZero

      # Initialize first position.
      addi  $a1, $zero, 1

      beq   $s0, 1, fibOne

      # Initialize second position.
```

```
	addi  $a2, $zero, 1

	# Push the stack.
	sw    $a1, ($sp)
	addi  $sp, $sp, -4
	sw    $a2, ($sp)

	# Initialize a loop counter (2 positions already stored)
	addi  $a1, $zero, 2
	fibLoop:
		beq   $s0, $a1, fibExit

		# Get the first two fibonacci values
		addi  $sp, $sp, 4
		lw    $t0, ($sp)
		addi  $sp, $sp, -4
		lw    $t1, ($sp)
		addi  $sp, $sp, -4

		# Add them together
		add   $t2, $t0, $t1

		# Store the new value
		sw    $t2, ($sp)

		# Increment the loop counter and re-enter the loop.
		addi  $a1, $a1, 1

		j fibLoop

	fibExit:
		jr    $ra

fibZero:
	# Check value
	li    $v0, 4
	la    $a0, error
	syscall

	jr    $ra

fibOne:
	# Push the stack.
	addi  $sp, $sp, -4
	sw    $a1, 0($sp)

	jr    $ra

#---------------------------------------------------------------------------------#
stackReader:
	# Set the counter.
	addi  $a1, $zero, 0
	stackLoop:
		# Exit upon completion.
		beq   $s0, $a1, stackExit

		# Print the current index.
		lw    $t0, ($sp)
		addi  $sp, $sp, 4
```

```
        li    $v0, 1
        move  $a0, $t0
        syscall

        li    $v0, 4
        la    $a0, delimit
        syscall

        addi  $a1, $a1, 1

        j stackLoop

stackExit:
        li    $v0, 4
        la    $a0, endMsg
        syscall

        jr    $ra
```

2) MERGE

```
.data
var_a:          .word 2 3 5 5 8 10 11 17 18 20
var_b:          .word 5 6 7 8 14 15 17
var_m:          .word 10
var_n:          .word 7

exitMsg:    .asciiz "Merge completed.\n"
spacer:         .asciiz " "

# Expected sequence in this location - 2 3 5 5 5 6 7 8 8 10 11 14 15 17 17 18 20
var_c: .word 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

.text
main:
      # Initialize the indices.
      # var_a head.
      addi  $s0, $zero, 0
      # var_b head.
      addi  $s1, $zero, 0
      # var_c head
      addi  $s2, $zero, 0

      # Store lengths
      lw    $s3, var_m
      lw    $s4, var_n
      mul   $s3, $s3, 4
      mul   $s4, $s4, 4

      jal merge

      # Notify user of completion.
      li    $v0, 4
      la    $a0, exitMsg
      syscall

      jal printMerge

      # Send OS exit signal.
      li    $v0, 10
      syscall

#-----------------------------------------------------------------------#

merge:
      # When either case is exhausted of values, append all that remain.
      beq   $s0, $s3, appendB
      beq   $s1, $s4, appendA

      # Get values
      lw    $t0, var_a($s0)
      lw    $t1, var_b($s1)

      # Find the smallest value
      sle   $a0, $t0, $t1
      beq   $a0, 1, atoc

      # Else...
```

```
      j btoc

#----------------------------------------------------------#

atoc:
      # Store the value
      sw    $t0, var_c($s2)

      #Advance the pointers.
      addi  $s0, $s0, 4
      addi  $s2, $s2, 4

      j merge
btoc:
      # Store the value
      sw    $t1, var_c($s2)

      # Advance the pointers
      addi  $s1, $s1, 4
      addi  $s2, $s2, 4

      j merge

#---------------------------------------------------------------#

appendA:
      # Append all remaining values of A to C.
      appendLoopA:
            beq   $s0, $s3, exitB
            lw    $t0, var_a($s0)

            # Store the value
            sw    $t0, var_c($s2)

            #Advance the pointers.
            addi  $s0, $s0, 4
            addi  $s2, $s2, 4

            j appendLoopA
      exitA:
            jr    $ra

appendB:
      # Append all remaining values of B to C.
      appendLoopB:
            beq   $s1, $s4, exitB
            lw    $t1, var_b($s1)

            # Store the value
            sw    $t1, var_c($s2)

            # Advance the pointers
            addi  $s1, $s1, 4
            addi  $s2, $s2, 4
      exitB:
            jr    $ra

#---------------------------------------------------------------#
```

```
printMerge:
	# Set index
	addi  $t0, $zero, 0

	# Set condition
	add   $t1, $s3, $s4

	while:
		beq   $t0, $t1, exit
		lw    $a0, var_c($t0)

		# Print the value
		li    $v0, 1
		syscall

		li    $v0, 4
		la    $a0, spacer
		syscall

		# Increment the index
		addi  $t0, $t0, 4

		j while

	exit:
		jr    $ra
```

3(a)

A = (6)_base10 6 / 2 = 3 R 0 3 / 2 = 1 R 1 1 / 2 = 0 R 1 A = (6)_base10 = (0110)_base2	A = (-3)_base10 3 / 2 = 1 R 1 1 / 2 = 0 R 1 A = (3)_base10 = (0011)_base2 Two's Complement Flip the bits = 1100, +1 = 1101 A= (-3)_base10 = (1101)_base2

3(b)

				0	1	1	0
(x)				1	1	0	1
				0	1	1	0
		Carry(1)	0	0	0	0	
	Carry(1)	0	1	1	0		
(+)	0	1	1	0			
	1	0	0	1	1	1	0

3(c)

Zero Extend: 0110	0000 0110
Sign Extend: 0110	0000 0110
Zero Extend: 1101	0000 1101
Sign Extend: 1101	1111 1101

4(a) F(x, y, z) = (xy)' + z

X	Y	Z	X'	Y'	XY	(XY)'	(XY)'+Z
0	0	0	1	1	0	1	1
0	0	1	1	1	0	1	1
0	1	0	1	0	0	1	1
0	1	1	1	0	0	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	0	1	1
1	1	0	0	0	1	0	0
1	1	1	0	0	1	0	1

4(b) F(x, y, z) = (X'YZ') + (XY'Z)

X	Y	Z	X'	Y'	Z'	X'YZ'	XY'Z	(X'YZ') + (XY'Z)
0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0	0
0	1	0	1	0	1	1	0	1
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0

5(a)

$$
\begin{aligned}
Z' &= X'Y'Z'+XY'Z'+X'YZ'+XYZ' \\
&= Z'(X'Y'+XY'+X'Y+XY) \\
&= Z'(X'(Y'Y)+X(Y'Y)) \\
&= Z'(X'(1)+X(1)) \\
&= Z'(X'+X) \\
&= Z'(1) \\
&= Z'
\end{aligned}
$$

5(b)

$$
\begin{aligned}
A'B+CD' &= (A'+C)(A'+D')(B+C)(B+D') \\
&= (A'+CD')(B+C)(B+D') \\
&= (A'+CD')(B+CD') \\
&= A'B+CD'
\end{aligned}
$$

6(a) f(A,B,C,D) = Em(1, 2, 3, 4, 6, 7, 9, 11, 12, 13, 14, 15)

AB/CD	00	01	11	10
00	0 0	1 1	1 3	1 2
01	1 4	0 5	1 7	1 6
11	1 12	1 13	1 15	1 14
10	0 8	1 9	1 11	0 10

Prime Implicants: A'C + AD + BD' + B'D + CD + AB
Essential Prime Implicants: A'C + AD + BD' + B'D

6(b) f(w, x, y, z) = Em(0, 5, 10, 15) + d(2, 7, 8, 13)

wx/yz	00	01	11	10
00	1	0	0	X
01	0	1	X	0
11	0	X	1	0
10	X	0	0	1

0000	0101
0010	0111
1000	1101
1010	1111

X'Z' + XZ

Prime Implicants: X'Z' + XZ
Essential Prime Implicants: X'Z' + XZ

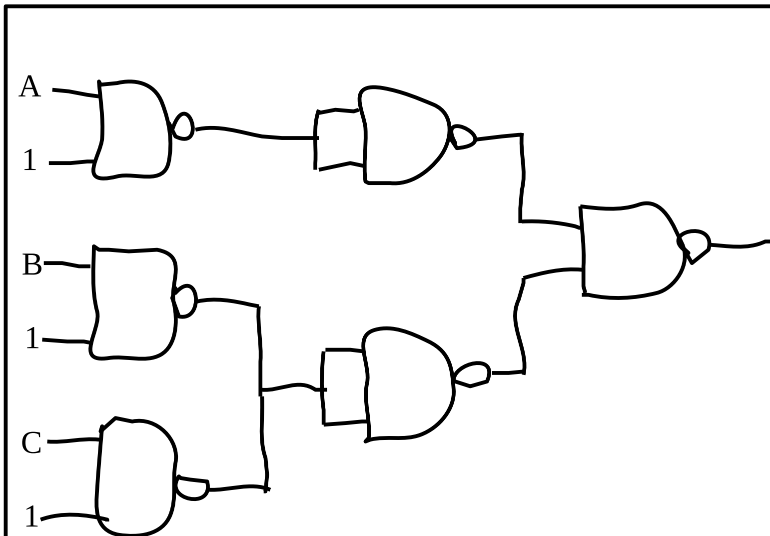
7. 4-Bit Single Digit Numbers (A, B, C, D)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

AB/CD	00	01	11	10
00	0 1	1 1	3 1	2 1
01	4 1	5 1	7 1	6 1
11	12	13	15	14
10	8 1	9 1	11	10

0000 0000
0001 0001
0011 1000
0010 1001
0100
0101
0111
0110

A' + (B'C')



A' + (B'C')