John Gangemi
U68714612
CDA4205

HOMEWORK #2 & #3

2.6.1

```
int temp = 0;
int Array[5] = {2,4,3,6,1};

// sets array contents to {1,4,3,6,2} using swap
temp = Array[0];
Array[0] = Array[4];
Array[4] = temp;

// sets array contents to {1,2,3,6,4} using swap
temp = Array[1];
Array[1] = Array[4];
Array[4] = temp;

//sets array contents to {1,2,3,4,6} using swap
temp = Array[3];
Array[3] = Array[4];
Array[4] = temp;
```

2.6.2

```
# base address of Array in register s6

lw St0, 0($s6)
lw $t1, 16($s6)
sw $t1, 0($s6)
sw $t0, 16($s6)

lw St0, 4($s6)
lw $t1, 16($s6)
sw $t1, 4($s6)
sw $t0, 16($s6)

lw St0, 12($s6)
lw $t1, 16($s6)
sw $t1, 12($s6)
sw $t0, 16($s6)
```

2.27

```
addi $t0, $zero, 0        # initialize i  = 0

LOOP1:
slt $t5, $t0, $s0        # set t5 = 1 if i < a
beq $t5, $zero, EXIT        # exit if t5 = 0
addi $t1, $zero, 0        # initialize j = 0
j LOOP2

LOOP2:
slt $t6, St1, $s1        # set t6 = 1 if j < b
beq $t6, $zero, BREAK        # break if t6 = 0
add $t2, $t0, $t1        # t2 = i + j
sll $t3, $t1, 4        # t3 = (4 * j) * 4 bytes for offset
add $t3, $t3, $s2        # add base address to offset
sw $t2, 0($t3)        # store (i+j)  in D[j]
addi $t1, $t1, 1        # increment j
j LOOP2

BREAK:
addi $t0, $t0, 1        # increment i
j LOOP1

EXIT:
```

2.28

To implement the code from exercise 2.27 using MIPS it requires 15 instructions where 4 instructions are needed in LOOP1 segment, 8 instructions are needed in LOOP2 segment, and 2 instructions are needed in BREAK segment and a single instruction in the beginning.

If a = 10, b = 1, and elements of D[] = 0 then the total number of MIPS instructions executed is 163. Each iteration of 'i' less than 10 must process 16 instructions. When 'i' is equal to 10 there are 2 instructions to process that signal the end of the code. Also there is a single instruction to initialize 'i' in the beginning of the program.

(16 X 10 iterations) + 2 + 1 = 163 instructions

2.29

```
int i = 0;
while ( i < 100 )
{
        result = result + MemArray[i];
        i++;
}
```

or

```
for (int i = 0; i < 100; i++)
{
        result = result + MemArray[i];
}
```

2.30

```
addi $t1, $s0, 400        # set i = 400 + base address of MemArray

LOOP:
lw $s1, 0($s0)        # load data at mem location s0 into s1
add $s2, $s2, $s1        # result += MemArray[i]
addi $s0, $s0, 4        # reference next word (element) in memory (MemArray)
bne $t1, $s0, LOOP        # if i != last address of MemArray, re-process loop
```

Reduced instruction count to 5 from previous 7 in exercise 2.29

2.31

```
# function argument stored in register a0
# return value stored in register v0
# register t0 holds value 1
# register t1 holds value returned from first recursive call

FIB:
addi $sp, $sp, -8       # adjust stack
sw $ra, 4($sp)       # save return address
sw $a0, 0($sp)       # save function argument

beq $a0, $zero, IF       # if n == 0, go to IF
addi $t0, $zero, 1       # t0 = 1
beq $a0, $t0, ELSEIF       # if n == 1, go to ELSEIF

addi $a0, $a0, -1       # n = n -1
jal FIB       # recursive call to fib(n-1)
lw $a0, 0($sp)       # restore previous function argument
lw $ra, 4($sp)       # restore previous return address
addi $sp, $sp, 8       # empty stack

add $t1, $v0, $zero       # store current value returned from fib(n-1)

addi $a0, $a0, -2       # n = n -2
jal FIB       # recursive call to fib(n-2)
lw $a0, 0($sp)       # restore previous function argument
lw $ra, 4($sp)       # restore previous return address
addi $sp, $sp, 8       # empty stack

add $v0, $t1, $v0       # return fib(n-1) + fib(n-2)
jr $ra       # return to caller

IF:
addi $v0, $zero, 0       # return 0
addi $sp, $sp, 8       # empty stack
jr $ra       # return to caller

ELSEIF:
addi $v0, $zero, 1       # return 1
addi $sp, $sp, 8       # empty stack
jr $ra       # return to caller
```