CPE 233 Software Assignment 4

Arrays in Assembly

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Mateo Vang (mkvang@calpoly.edu)

February 5, 2024

Table of Contents

1 Flow Charts

1.1 Fibonacci Addition Flowchart

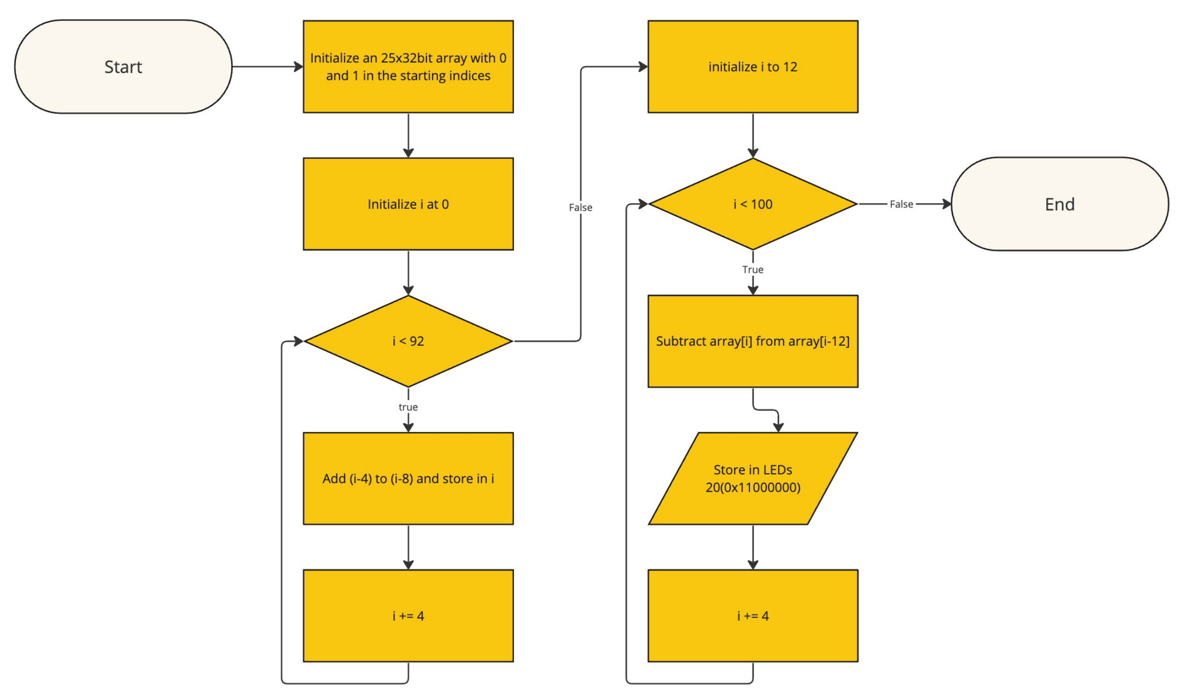

Figure 1: Creating and Manipulating a Fibonacci Sequence Flowchart

1.2 Array Sorting Flowchart

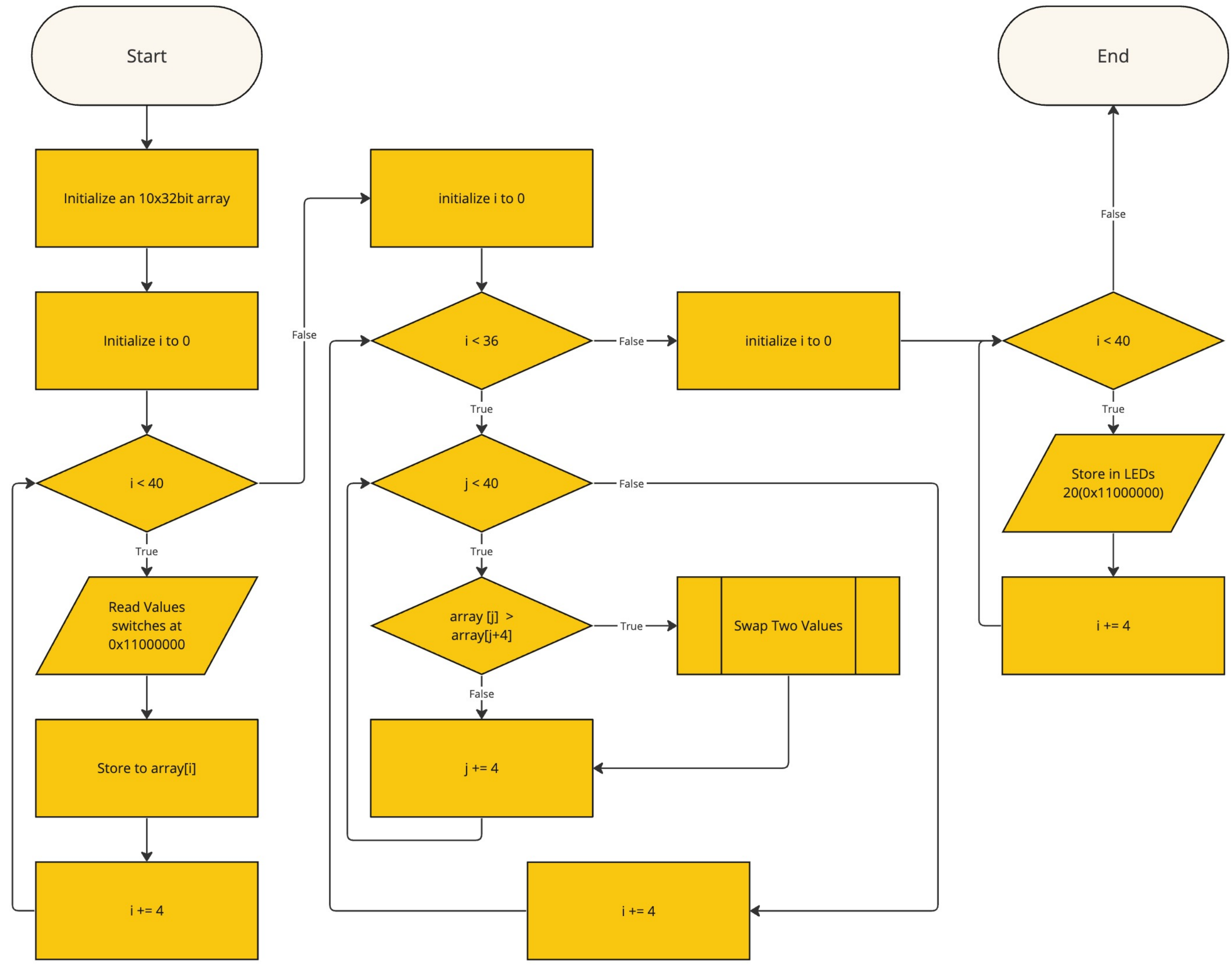

Figure 2: Array Sorting Flow Chart

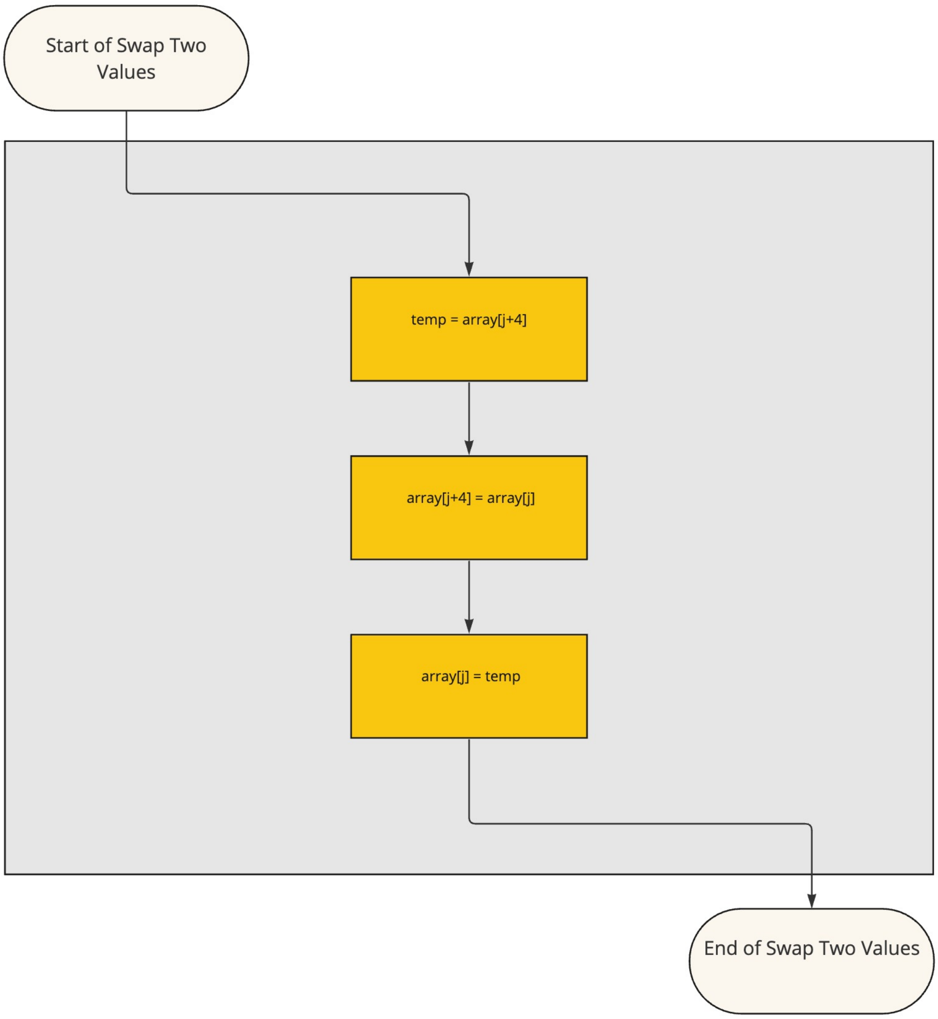

Figure 3: Swapping Two Values Sub-process Flow Chart

2 Assembly Instructions

2.1 Fibonacci Addition Listing

```
# file: SW4-FibonacciAddition.asm
# brief: Assembly code for generateing the Fibonacci sequence and then
# adding values.
#
# This file contains the assembly code for generating the first 25 Fibonacci
# numbers and then adding every other number to each other and then writing
# the output to the LEDs.
#
# author: Mateo Vang
# date: 02-03-2024

.data
    ARRAY: .word 0 # 0x6000
        .word 1 # 0x6004
        .space 92 # 92 bytes = 23 words


.text
    li t0, 0
    li t6, 92 # conditional change to 92 when submitting
    la s0, ARRAY # Fn-2 Address
    la s1, ARRAY # Fn-1 Address
    li s2, 0x11000020 # LEDs address

    CreateFibo:     bge t0, t6, EndCreate
            lw t3, 0(s0) # Fn-2 Pointer
            lw t4, 4(s1) # Fn-1 Pointer
            add t5, t3, t4 # creates the next number in the Fibonacci sequence
            addi s0, s0, 4 # increments to the next Fn-2
            addi s1, s1, 4 # increments to the next Fn-1
            sw t5, 4(s0) # stores Fn into the next address of the array
            addi t0, t0, 4 # increments to the next word address
    j CreateFibo

    EndCreate: # Now we get ready to subtract the Fibo numbers and store them in LEDs
         li t0, 12
         li t6, 100 # conditional change to 100 when submitting

         # resetting our address pointers
         sub s0, s0, t6
         addi s0, s0, 8 # s0 points to Fn-3 initialized to 0
         sub s1, s1, t6
         addi s1, s1, 20 # s1 points to Fn initialized t0 12


    SubFibo: bge t0, t6, end
         lw t3, 0(s0) # Fn-3 Pointer
         lw t4, 0(s1) # Fn Pointer
```

```
        sub t5, t4, t3 # creates the next value to store in LEDs
        addi s0, s0, 4 # increments to the next Fn-3
        addi s1, s1, 4 # increments to the next Fn
        sw t5, 0(s2) # stores our value into the LEDs
        addi t0, t0, 4
    j SubFibo

    end: nop
```

Listing 1: Assembly Code for the Fibonacci Sequence in Figure 1

2.2 Array Sorting Listing

```
# file: SW4-ArraySorting.asm
# brief: Assembly code for sorting an array.
#
# This file contains the assembly code for sorting an array of 10 32-bit
# unsigned numbers. The sorting algorithm that was implemented in this
# project is bubble sort.
#
# author: Ethan Vosburg
# date: 02-03-2024

.data
sortArray:
    # Create space in an array for 10 32-bit unsigned numbers
    .space 40

.text
    # Initialize registers
    li      t0, 0               # Counter for loops
    li      t1, 0               # Counter for bubble sort
    li      t2, 40              # Condition for finishing switch read
    li      t3, 36              # Condition for Bubble sort pass
    lui     s0, 0x11000         # Load io address
    la      s1, sortArray       # Load array address

readSwitches:
    # Read in switches from 0x11000000
    bge     t0, t2, endLoad     # Check if loading from switches is done
    lw      t6, 0(s0)           # Read the switches in to a t6 temporary
    sw      t6, 0(s1)           # Store the switch value in sortArray
    addi    s1, s1, 4           # Iterate to the next address
    addi    t0, t0, 4           # Iterate the loop variable
    j       readSwitches

endLoad:
    li      t0, 4               # Reset counter for loop

bubbleBegin:
    bgeu    t0, t2, bubbleEnd   # Check if bubble sort is done
    la      s1, sortArray       # Reset array address for next pass
    li      t1, 0               # Reset the pass counter
```

```
passBegin:
    bgeu    t1, t3, passDone    # Check is the current pass is done
    lw      t4, 0(s1)           # Load j
    lw      t5, 4(s1)           # Load j + 1
    bleu    t4, t5, noSwap      # If the left number is greater, swap
    # Swap the values
    sw      t5, 0(s1)
    sw      t4, 4(s1)

noSwap:
    addi    s1, s1, 4           # Iterate the index counter
    addi    t1, t1, 4           # Iterate index count
    j       passBegin

passDone:
    addi    t0, t0, 4           # Iterate pass count
    j       bubbleBegin
bubbleEnd:

    li      t0, 0               # Counter for loops
    la      s1, sortArray       # Reset array address for write-out

writeSwitches:
    # Wtite to the switches at 0x11000020
    bge     t0, t2, endWrite    # Check is writing out switches is done
    lw      t6, 0(s1)           # Read Read the switch value in sortArray
    sw      t6, 0x20(s0)        # Write t6 to the switches
    addi    s1, s1, 4           # Iterate to the next address
    addi    t0, t0, 4           # Iterate the loop variable
    j       writeSwitches

endWrite:
# Program Done
```

Listing 2: Assembly Code for Array Sorting in Figure 2

3 RARS Verification

3.1 Fibonacci Verification

The test cases below demonstrate the code correctly computes the first 25 numbers of the Fibonacci Sequence.

Figure 4: Flow Chart 1 Fibonacci Sequence Verification

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000001	0x00000001	0x00000002	0x00000003	0x00000005	0x00000008	0x0000000d
0x00006020	0x00000015	0x00000022	0x00000037	0x00000059	0x00000090	0x000000e9	0x00000179	0x00000262
0x00006040	0x000003db	0x0000063d	0x00000a18	0x00001055	0x00001a6d	0x00002ac2	0x0000452f	0x00006ff1
0x00006060	0x0000b520	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The test cases below demonstrate the code performs the desired outputs.

Table 1: Flow Chart 1 Test Cases

Fn-Fn-3	Decimal Equation	My Calculations	RARS Outputs
F3-F1	2-0	0x2	0x2
F5-F2	5-1	0x4	0x4
F24-F21	46368-10946	0x8A5E	0x8A5E

1. Test case 1 shows the first result to be outputted to the LEDs
2. Test case 2 shows that the program doesn't repeat the same result.
3. Test case 3 shows that the program stops once no item exists 3 spots away.

3.2 Array Sorting Verification

Figure 5: Test Case 1: Opposite Sequential

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003	0x00000002
0x00006020	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x00006020	0x00000008	0x00000009	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 6: Test Case 2: Random Numbers

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000009	0x00000007	0x00000008	0x00000005	0x00000006	0x00000003	0x00000004	0x00000001
0x00006020	0x00000002	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x00006020	0x00000008	0x00000009	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 7: Test Case 3: Already Ordered

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x00006020	0x00000008	0x00000009	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x00006020	0x00000008	0x00000009	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 8: Test Case 4: Minimum Memory Number and Maximum Memory Number

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0x00000000	0x00000000	0x00000000
0x00006020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0xffffffff	0xffffffff	0xffffffff
0x00006020	0xffffffff	0xffffffff	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 9: Test Case 5: All Bits Shifted

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0x55555555	0x55555555	0x55555555
0x00006020	0x55555555	0x55555555	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00006000	0x55555555	0x55555555	0x55555555	0x55555555	0x55555555	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa
0x00006020	0xaaaaaaaa	0xaaaaaaaa	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The test cases above demonstrate the code produces the desired outputs.

1. Test case 1 shows values decreasing to show a worst-case scenario where no numbers are already in order.
2. Test case 2 shows numbers that are randomly in order and not in order.
3. Test case 3 shows numbers that are already to verify that they will not be placed out of order.
4. Test case 4 shows the minimum possible values of the memory array and the maximum possible values of the memory array.
5. Test case 5 shows alternating bits to ensure that all bits are cycled at least once without any errors.