



CAL POLY

CPE 233 Software Assignment 4

Arrays in Assembly

Report by:

Ethan Vosburg (evosburg@calpoly.edu)

Mateo Vang (mkvang@calpoly.edu)

February 6, 2024

Table of Contents

1 Flow Charts	3
1.1 Fibonacci Addition Flowchart	3
1.2 Array Sorting Flowchart	4
2 Assembly Instructions	6
2.1 Fibonacci Addition Listing	6
2.2 Array Sorting Listing	7
3 RARS Verification	9
3.1 Fibonacci Verification	9
3.2 Array Sorting Verification	10

1 Flow Charts

1.1 Fibonacci Addition Flowchart

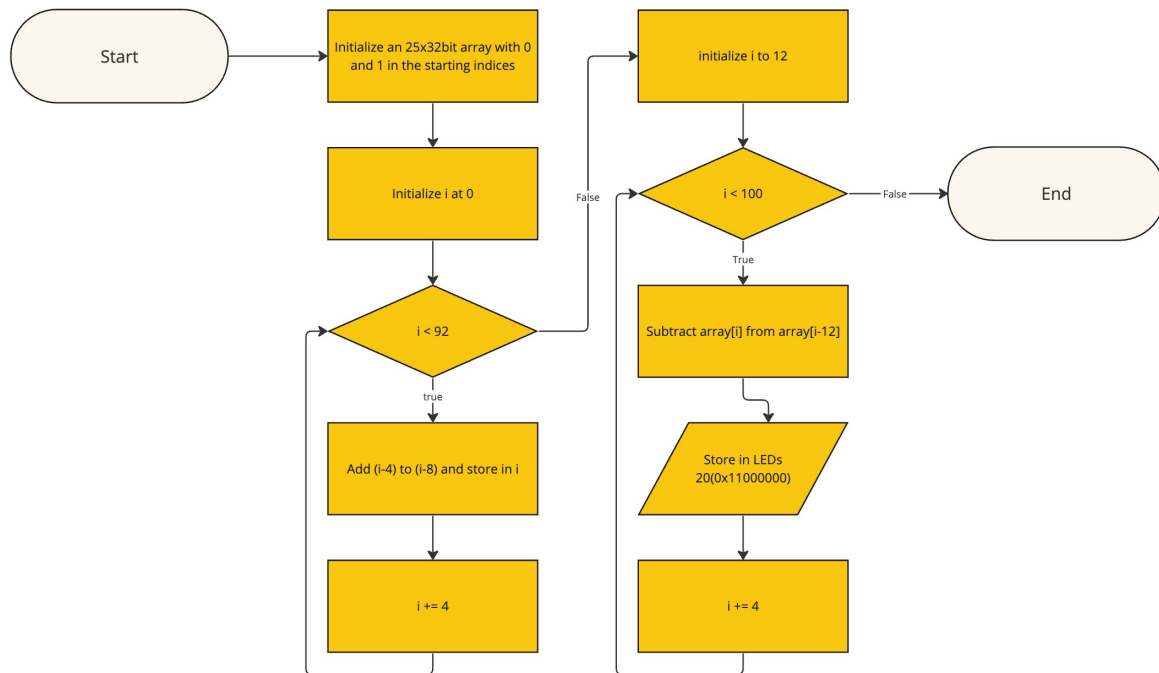


Figure 1: Creating and Manipulating a Fibonacci Sequence Flowchart

1.2 Array Sorting Flowchart

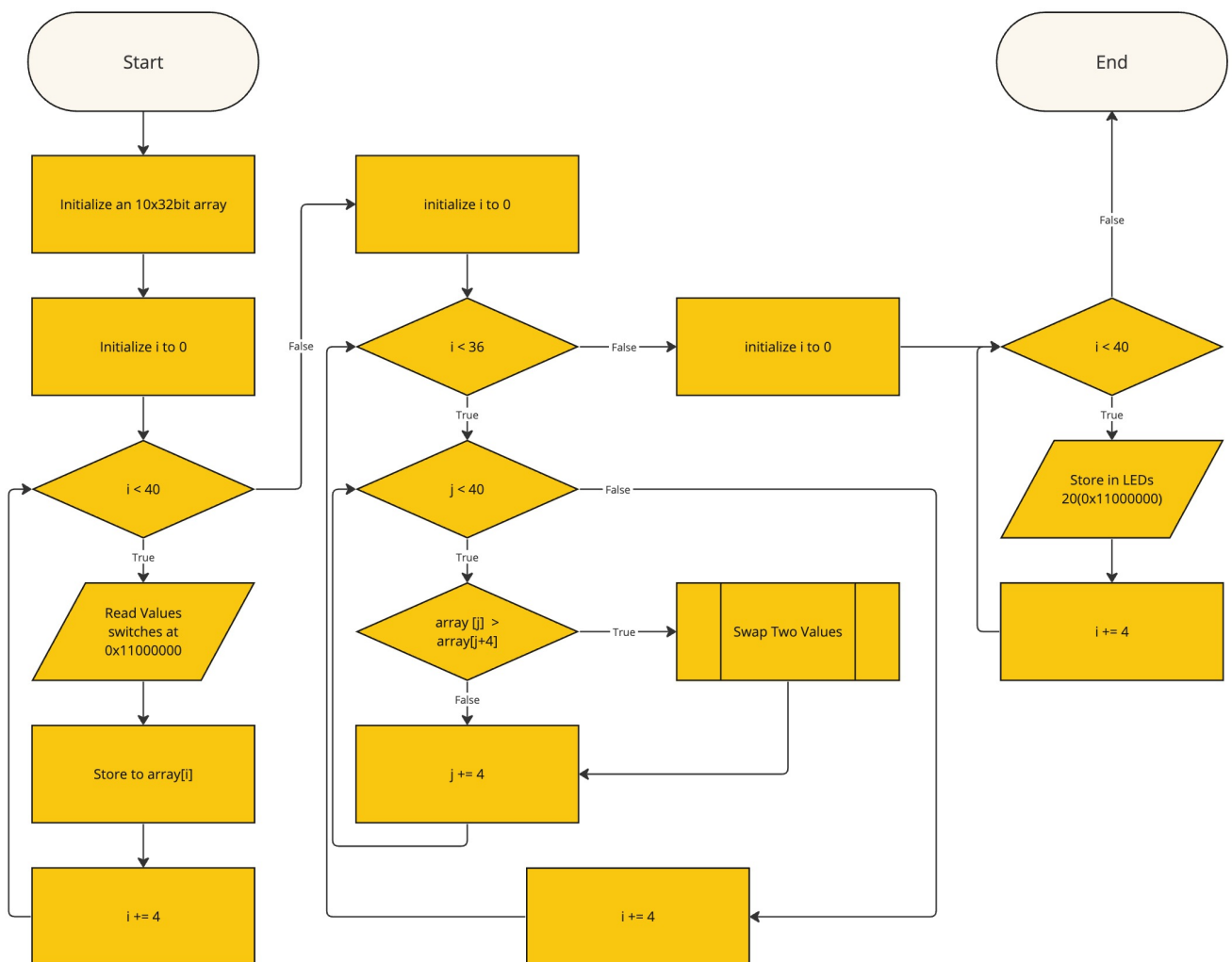


Figure 2: Array Sorting Flow Chart

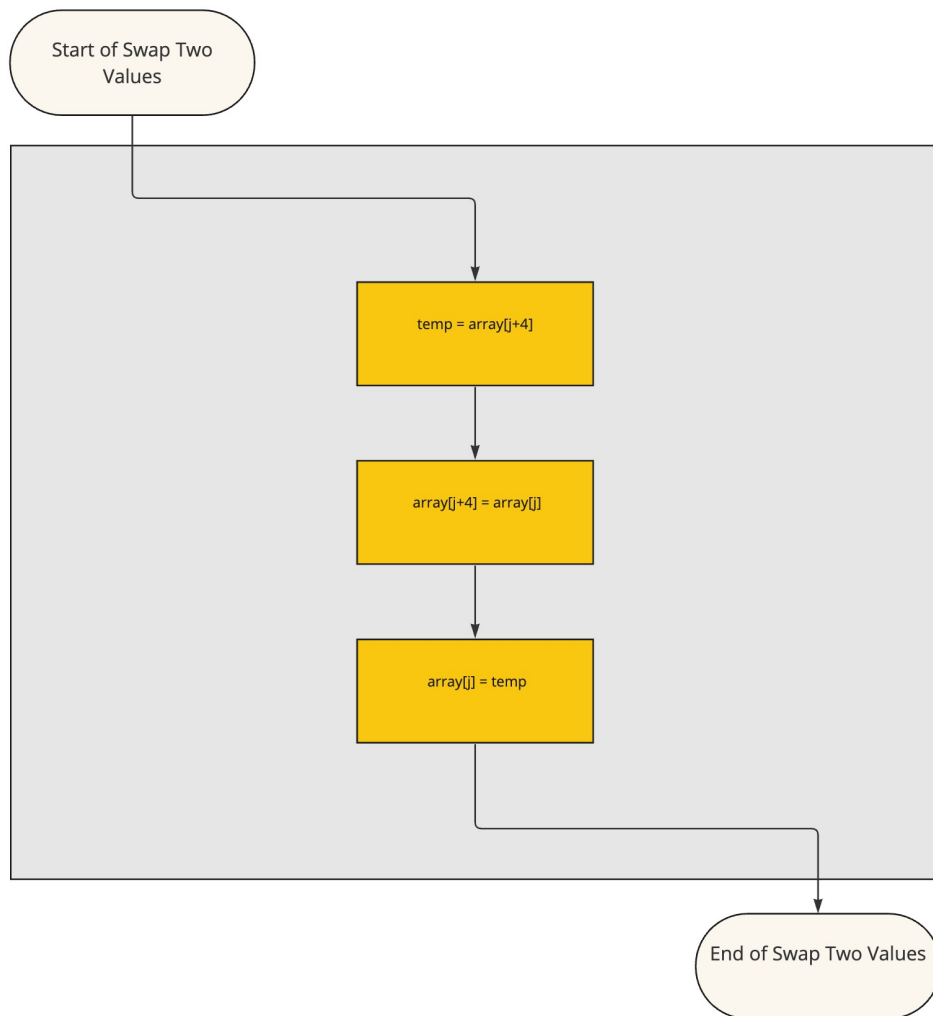


Figure 3: Swapping Two Values Sub-process Flow Chart

2 Assembly Instructions

2.1 Fibonacci Addition Listing

```
1  # file: SW4-FibonacciAddition.asm
2  # brief: Assembly code for generateing the Fibonacci sequence and then
3  # adding values.
4  #
5  # This file contains the assembly code for generating the first 25 Fibonacci
6  # numbers and then adding every other number to each other and then writing
7  # the output to the LEDs.
8  #
9  # author: Mateo Vang
10 # date: 02-03-2024
11
12 .data
13     ARRAY:
14         .word 0          # 0x6000
15         .word 1          # 0x6004
16         .space 92        # 92 bytes = 23 words
17
18
19 .text
20     li t0, 0
21     li t6, 92            # conditional change to 92 when submitting
22     la s0, ARRAY         # Fn-2 Address
23     la s1, ARRAY         # Fn-1 Address
24     li s2, 0x11000020    # LEDs address
25
26 CreateFibo:
27     bge t0, t6, EndCreate
28     lw t3, 0(s0)         # Fn-2 Pointer
29     lw t4, 4(s1)         # Fn-1 Pointer
30     add t5, t3, t4       # creates the next number in the Fibonacci sequence
31     addi s0, s0, 4       # increments to the next Fn-2
32     addi s1, s1, 4       # increments to the next Fn-1
33     sw t5, 4(s0)         # stores Fn into the next address of the array
34     addi t0, t0, 4       # increments to the next word address
35     j CreateFibo
36
37 EndCreate:              # Setup subtract the Fibo numbers and store them in LEDs
38     li t0, 12
39     li t6, 100          # conditional change to 100 when submitting
40
41     # resetting our address pointers
42     sub s0, s0, t6
43     addi s0, s0, 8       # s0 points to Fn-3 initialized to 0
44     sub s1, s1, t6
45     addi s1, s1, 20      # s1 points to Fn initialized t0 12
46
47
48 SubFibo:
49     bge t0, t6, end
```

```

50    lw t3, 0(s0)           # Fn-3 Pointer
51    lw t4, 0(s1)           # Fn Pointer
52    sub t5, t4, t3         # creates the next value to store in LEDs
53    addi s0, s0, 4         # increments to the next Fn-3
54    addi s1, s1, 4         # increments to the next Fn
55    sw t5, 0(s2)           # stores our value into the LEDs
56    addi t0, t0, 4
57    j SubFibo
58
59 end:

```

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

2.2 Array Sorting Listing

```

1  # file: SW4-ArraySorting.asm
2  # brief: Assembly code for sorting an array.
3  #
4  # This file contains the assembly code for sorting an array of 10 32-bit
5  # unsigned numbers. The sorting algorithm that was implemented in this
6  # project is bubble sort.
7  #
8  # author: Ethan Vosburg
9  # date: 02-03-2024
10
11 .data
12 sortArray:
13     # Create space in an array for 10 32-bit unsigned numbers
14     .space 40
15
16 .text
17     # Initialize registers
18     li t0, 0               # Counter for loops
19     li t1, 0               # Counter for bubble sort
20     li t2, 40              # Condition for finishing switch read
21     li t3, 36              # Condition for Bubble sort pass
22     lui s0, 0x11000        # Load io address
23     la s1, sortArray       # Load array address
24
25 readSwitches:
26     # Read in switches from 0x11000000
27     bge t0, t2, endLoad    # Check if loading from switches is done
28     lw t6, 0(s0)           # Read the switches in to a t6 temporary
29     sw t6, 0(s1)           # Store the switch value in sortArray
30     addi s1, s1, 4         # Iterate to the next address
31     addi t0, t0, 4         # Iterate the loop variable
32     j readSwitches
33
34 endLoad:
35     li t0, 4               # Reset counter for loop
36
37 bubbleBegin:
38     bgeu t0, t2, bubbleEnd # Check if bubble sort is done

```

```
39      la      s1, sortArray      # Reset array address for next pass
40      li      t1, 0              # Reset the pass counter
41
42 passBegin:
43      bgeu    t1, t3, passDone    # Check is the current pass is done
44      lw      t4, 0(s1)           # Load j
45      lw      t5, 4(s1)           # Load j + 1
46      bleu    t4, t5, noSwap      # If the left number is greater, swap
47      # Swap the values
48      sw      t5, 0(s1)
49      sw      t4, 4(s1)
50
51 noSwap:
52      addi    s1, s1, 4           # Iterate the index counter
53      addi    t1, t1, 4           # Iterate index count
54      j       passBegin
55
56 passDone:
57      addi    t0, t0, 4           # Iterate pass count
58      j       bubbleBegin
59 bubbleEnd:
60
61      li      t0, 0              # Counter for loops
62      la      s1, sortArray      # Reset array address for write-out
63
64 writeSwitches:
65      # Write to the switches at 0x11000020
66      bge     t0, t2, endWrite    # Check is writing out switches is done
67      lw      t6, 0(s1)           # Read Read the switch value in sortArray
68      sw      t6, 0x20(s0)        # Write t6 to the switches
69      addi    s1, s1, 4           # Iterate to the next address
70      addi    t0, t0, 4           # Iterate the loop variable
71      j       writeSwitches
72
73 endWrite:
74 # 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

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
F4-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.