Program Assembly Code:

AREA Reset, CODE, Readonly

ENTRY

ADD0 EQU 0x40000000

ADDR RN R0

COUNT RN R1

NUM1P1 RN R2

NUM1P2 RN R3

NUM1P3 RN R4

NUM2P1 RN R5

NUM2P2 RN R6

NUM2P3 RN R7

HOLD1 RN R8

HOLD2 RN R9

HOLD3 RN R10

;load address1 into r1

LDR ADDR,=ADD0

MOV COUNT,#99

;initializing values

MOV NUM1P1, #0

MOV NUM1P2, #0

MOV NUM1P3, #0

MOV NUM2P1, #0

MOV NUM2P2, #0

MOV NUM2P3, #1

AGAIN

;add numbers

ADDS HOLD3, NUM1P3, NUM2P3 ADCS HOLD2, NUM1P2, NUM2P2 ADC HOLD1, NUM1P1, NUM2P1

;store numbers

STR HOLD3, [ADDR]

ADD ADDR, ADDR, #4

STR HOLD2, [ADDR]

ADD ADDR, ADDR, #4

STR HOLD1, [ADDR]

ADD ADDR, ADDR, #4

;num2 = new num1

MOV NUM1P1, NUM2P1

MOV NUM1P2, NUM2P2

MOV NUM1P3, NUM2P3

;hold = new num2

MOV NUM2P1, HOLD1

MOV NUM2P2, HOLD2

MOV NUM2P3, HOLD3

;loop to count times

SUBS COUNT, COUNT, #1

;branch to again if count is not yet done

BNE AGAIN

stop B stop

END

Snapshots of Code:

```
AREA Reset, CODE, Readonly
    ENTRY
2
                                                                                      ADDS HOLD3, NUM1P3, NUM2P3
    ADD0 EOU 0x40000000
                                                                                     ADCS HOLD2,
                                                                                                  NUM1P2, NUM2P2
     ADDR
             RN RO
                                                                                     ADC HOLD1,
                                                                                                  NUM1P1, NUM2P1
5
     COUNT
             RN R1
    NUM1P1 RN R2
                                                                         33
     NUM1P2 RN R3
                                                                         34
35
                                                                                      STR HOLD3, [ADDR]
                                                                                     ADD ADDR, ADDR, #4
     NUM1P3 RN R4
                                                                         36
37
38
                                                                                     STR HOLD2, [ADDR]
ADD ADDR, ADDR, #4
    NUM2P1 RN R5
10
    NUM2P2 RN R6
                                                                                      STR HOLDI, [ADDR]
11
    NUM2P3 RN R7
                                                                         39
40
41
42
43
44
45
46
47
48
                                                                                     ADD ADDR, ADDR, #4
13
    HOT.D2
             RN R9
                                                                                      MOV NUM1P1, NUM2P1
14
    HOLD3 RN R10
                                                                                     MOV NUM1P2, NUM2P2
15
                                                                                     MOV NUM1P3, NUM2P3
              :load addressl into rl
              LDR ADDR, =ADD0
                                                                                     ;hold = new num2
                                                                                     MOV NUM2P1, HOLD1
MOV NUM2P2, HOLD2
18
             MOV COUNT, #99
              ;initializing values
                                                                                     MOV NUM2P3, HOLD3
              MOV NUM1P1, #0
              MOV NUM1P2, #0
              MOV NUM1P3, #0
                                                                            SUBS COUNT, COUNT, #1
              MOV NUM2P1, #0
              MOV NUM2P2, #0
                                                                                      ;branch to again if count is not yet done
```

Figure 2.1 - Snapshot of Code in Keil

Figure 2.2 - Snapshot of Code in Keil

In Figure 2.1 which shows lines 1-25 I first declared all of the registers and constants I would use throughout the code. Second I initialized the first two values of the Fibonacci sequence by moving 0 into num1 and 1 into num2. Finally I split up the numbers into three registers so that I wouldn't run into any problems when I reached the higher numbers that would be too big to be represented by 32 bits. In Figure 2.2 which shows lines 27-55 I made a loop that would first add all three parts of the number and store them in a separate variable called hold. I then stored the numbers into their respective addresses and added 4 to the address so the next part of the number saved would not overwrite the last saved one. Next, I stored the hold into num2, num2 into num1 so the sequence could continue the next loop around. Finally I had a counter that would subtract once after every loop and a check to make sure the program did not run infinitely. Once the count reached 0 after 100 loops the program was done.

SRAM Memory Locations:

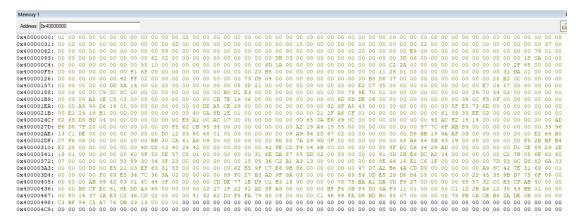


Figure 2.3 - SRAM Memory Location starting at 0x40000000 after program execution

CPSR Register:

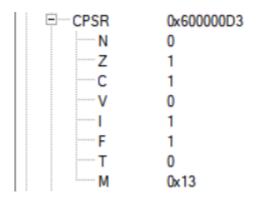


Figure 2.4 - CPSR Register after program execution

Used Registers after Code Execution:

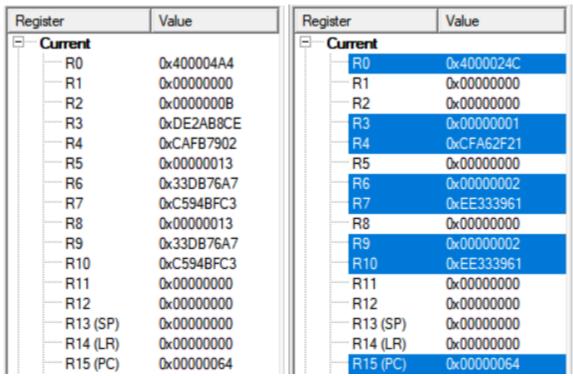


Figure 2.5 - After 100 loops (full program) Figure 2.6 - After 50 loops (half program)

To confirm that my program worked I was able to check through the registers tab. As I explained I stored the numbers added into a temporary variable named hold broken into 3 parts in registers R8-R10. In Figure 2.5 you can see R8 holds 0x13, R9 holds 0x33DB76A7, and R10 holds 0xC594BFC3. When you store those values next to each other in memory as I did as shown in Figure 2.3 you get 0x1333DB76A7C594BFC3 which equals the 100th Fibonacci number when converted to decimal. Just to be safe I also checked the 50th number in Figure 2.6 where R9 and R10 combined to hold 0x2EE333961 which is the 50th number in the Fibonacci sequence in hexadecimal.