# ARM ASSEMBLY LAB-3

By

Swapneel Pimparkar (CS18M516)
CS6620a
Prof. Madhu Mutyam

## STATUS UPDATE

The ARM assembly code was written for 32 bit processor and verified using ARMSim simulator successfully.

## EXERCISE – FIND MAX WEIGHT AND MAX WEIGHT STRING FROM RANGE OF DATA LOCATIONS IN MEMORY

The main part of objective/problem statement of the exercise is as follows:

Weight of a 32-bit number is defined as the total number of bits that are set in the 32-bit number. Given a series of 32-bit numbers (in hexadecimal form), write an assembly program to determine which element in the series has the largest weight and store the number in NUM and its weight in WEIGHT.

There are multiple ways to produce the mnemonics.

- Option 1 - Use Brian Kernighan Algorithm (n = n & (n-1))
- Option 2 - Keep shifting left and check the msb. If 1 then increase the count.
- Option 3 - Use Neon SIMD (vector) instructions.
- Option 4 - Using lookup tables.
- Option 5 - Using 64 bit instructions.
- Option 6 - Counting bits set in parallel.

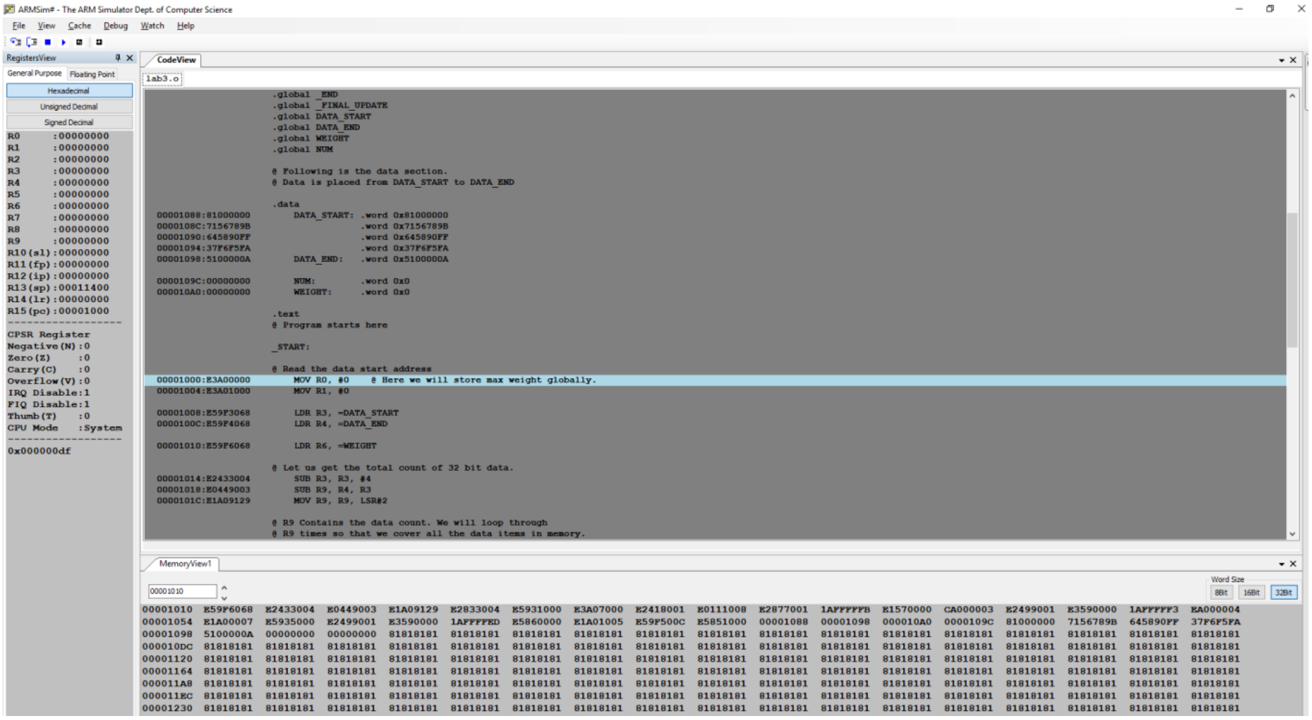
And few more.

For this exercise, algorithm from Brian Kernighan is used. Most of the tricks are present in “Hacker’s Delight” by Henry S. Warren, Jr.

**The algorithm is hand-coded in 32 bit ARM Assembly and verified on ARMSim Simulator.**

## INITIAL STATE SCREENSHOT

The sample output screenshot for the logic is – (Registers listed on the left are to be noted. All are zero to begin with). Similarly, the memory view in bottom pane of the Simulator Window.



```
R0        :00000000
R1        :00000000
R2        :00000000
R3        :00000000
R4        :00000000
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl) :00000000
R11(fp) :00000000
R12(ip) :00000000
R13(sp) :00011400
R14(lr) :00000000
R15(pc) :00001000
-----------------
CPSR Register
Negative(N):0
Zero(Z)      :0
Carry(C)     :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)     :0
CPU Mode     :System
-----------------
0x000000df
```

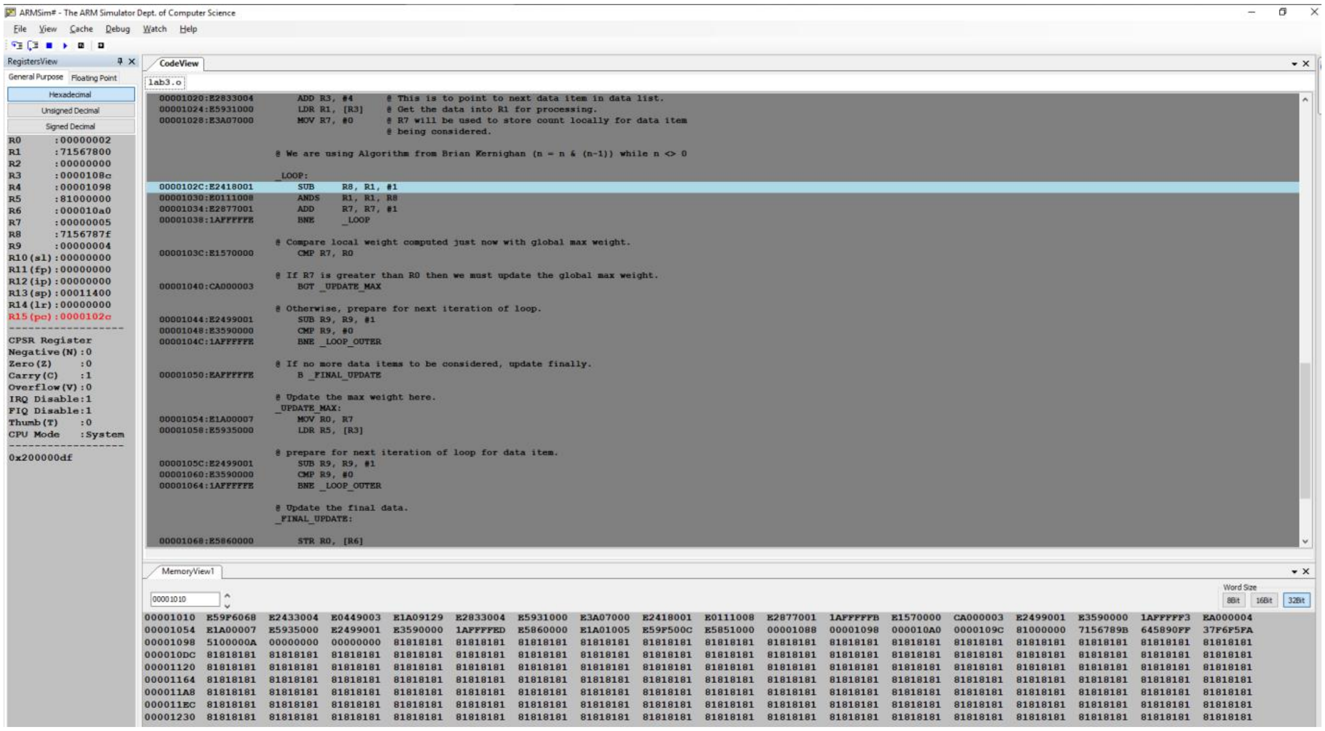
```
                        .global _END
                        .global _FINAL_UPDATE
                        .global DATA_START
                        .global DATA_END
                        .global WEIGHT
                        .global NUM

                        @ Following is the data section.
                        @ Data is placed from DATA_START to DATA_END

                        .data
00001088:81000000           DATA_START: .word 0x81000000
0000108C:7156789B                       .word 0x7156789B
00001090:645890FF                       .word 0x645890FF
00001094:37F6F5FA                       .word 0x37F6F5FA
00001098:5100000A           DATA_END:   .word 0x5100000A

0000109C:00000000           NUM:        .word 0x0
000010A0:00000000           WEIGHT:     .word 0x0

                        .text
                        @ Program starts here

                        _START:

                        @ Read the data start address
00001000:E3A00000           MOV R0, #0    @ Here we will store max weight globally.
00001004:E3A01000           MOV R1, #0

00001008:E59F3068           LDR R3, =DATA_START
0000100C:E59F4068           LDR R4, =DATA_END

00001010:E59F6068           LDR R6, =WEIGHT

                        @ Let us get the total count of 32 bit data.
00001014:E2433004           SUB R3, R3, #4
00001018:E0449003           SUB R9, R4, R3
0000101C:E1A09129           MOV R9, R9, LSR#2

                        @ R9 Contains the data count. We will loop through
                        @ R9 times so that we cover all the data items in memory.
```

```
00001010
```

```
00001010 E59F6068 E2433004 E0449003 E1A09129 E2833004 E5931000 E3A07000 E2418001 E0111008 E2877001 1AFFFFFB E1570000 CA000003 E2499001 E3590000 1AFFFFF3 EA000004
00001054 E1A00007 E5935000 E2499001 E3590000 1AFFFFED E5860000 E1A01005 E59F500C E5851000 00001088 00001098 000010A0 0000109C 81000000 7156789B 645890FF 37F6F5FA
00001098 5100000A 00000000 00000000 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
000010DC 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
00001120 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
00001164 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
000011A8 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
000011EC 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
00001230 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
```

## INTERMEDIATE STATE SCREENSHOT

Intermediate execution screenshot is as follows:



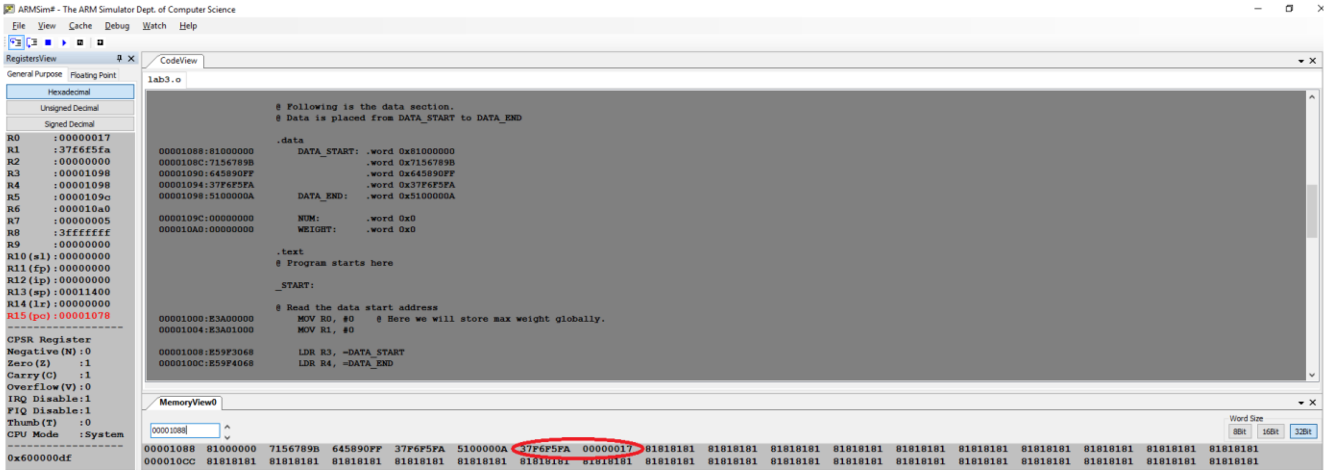
```
R0        :00000002
R1        :71567800
R2        :00000000
R3        :0000108c
R4        :00001098
R5        :81000000
R6        :000010a0
R7        :00000005
R8        :7156787f
R9        :00000004
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):0000102c
------------------
CPSR Register
Negative(N):0
Zero(Z)     :0
Carry(C)    :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)    :0
CPU Mode    :System
------------------
0x200000df
```

```
00001020:E2833004        ADD R3, #4       @ This is to point to next data item in data list.
00001024:E5931000        LDR R1, [R3]     @ Get the data into R1 for processing.
00001028:E3A07000        MOV R7, #0       @ R7 will be used to store count locally for data item
                                          @ being considered.

                    @ We are using Algorithm from Brian Kernighan (n = n & (n-1)) while n <> 0

                    _LOOP:
0000102C:E2418001        SUB     R8, R1, #1
00001030:E0111008        ANDS    R1, R1, R8
00001034:E2877001        ADD     R7, R7, #1
00001038:1AFFFFFE        BNE     _LOOP

                    @ Compare local weight computed just now with global max weight.
0000103C:E1570000        CMP R7, R0

                    @ If R7 is greater than R0 then we must update the global max weight.
00001040:CA000003        BGT _UPDATE_MAX

                    @ Otherwise, prepare for next iteration of loop.
00001044:E2499001        SUB R9, R9, #1
00001048:E3590000        CMP R9, #0
0000104C:1AFFFFFE        BNE _LOOP_OUTER

                    @ If no more data items to be considered, update finally.
00001050:EAFFFFFE        B _FINAL_UPDATE

                    @ Update the max weight here.
                    _UPDATE_MAX:
00001054:E1A00007        MOV R0, R7
00001058:E5935000        LDR R5, [R3]

                    @ prepare for next iteration of loop for data item.
0000105C:E2499001        SUB R9, R9, #1
00001060:E3590000        CMP R9, #0
00001064:1AFFFFFE        BNE _LOOP_OUTER

                    @ Update the final data.
                    _FINAL_UPDATE:

00001068:E5860000        STR R0, [R6]
```

```
00001010  E59F6068  E2433004  E0449003  E1A09129  E2833004  E5931000  E3A07000  E2418001  E0111008  E2877001  1AFFFFFB  E1570000  CA000003  E2499001  E3590000  1AFFFFF3  EA000004
00001054  E1A00007  E5935000  E2499001  E3590000  1AFFFFED  E5860000  E1A01005  E59F500C  E5851000  00001088  00001098  000010A0  0000109C  81000000  7156789B  645890FF  37F6F5FA
00001098  5100000A  00000000  00000000  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
000010DC  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
00001120  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
00001164  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
000011A8  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
000011EC  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
00001230  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
```

## FINAL STATE SCREENSHOT

At the end, the memory location updated with values & is as follows (and it is as per expectations). The final data is circled for the memory addresses.



```
R0       :00000017
R1       :37f6f5fa
R2       :00000000
R3       :00001098
R4       :00001098
R5       :0000109c
R6       :000010a0
R7       :00000005
R8       :3fffffff
R9       :00000000
R10(sl)  :00000000
R11(fp)  :00000000
R12(ip)  :00000000
R13(sp)  :00011400
R14(lr)  :00000000
R15(pc)  :00001078
------------------
CPSR Register
Negative(N):0
Zero(Z)    :1
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x600000df
```

```
                        @ Following is the data section.
                        @ Data is placed from DATA_START to DATA_END

                        .data
00001088:81000000           DATA_START: .word 0x81000000
0000108C:7156789B                       .word 0x7156789B
00001090:645890FF                       .word 0x645890FF
00001094:37F6F5FA                       .word 0x37F6F5FA
00001098:5100000A           DATA_END:   .word 0x5100000A

0000109C:00000000           NUM:        .word 0x0
000010A0:00000000           WEIGHT:     .word 0x0

                        .text
                        @ Program starts here

                        _START:

                        @ Read the data start address
00001000:E3A00000           MOV R0, #0    @ Here we will store max weight globally.
00001004:E3A01000           MOV R1, #0

00001008:E59F3068           LDR R3, =DATA_START
0000100C:E59F4068           LDR R4, =DATA_END
```

```
00001088
00001088  81000000  7156789B  645890FF  37F6F5FA  5100000A  37F6F5FA  00000017  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
000010CC  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181  81818181
```