

## Fibonacci

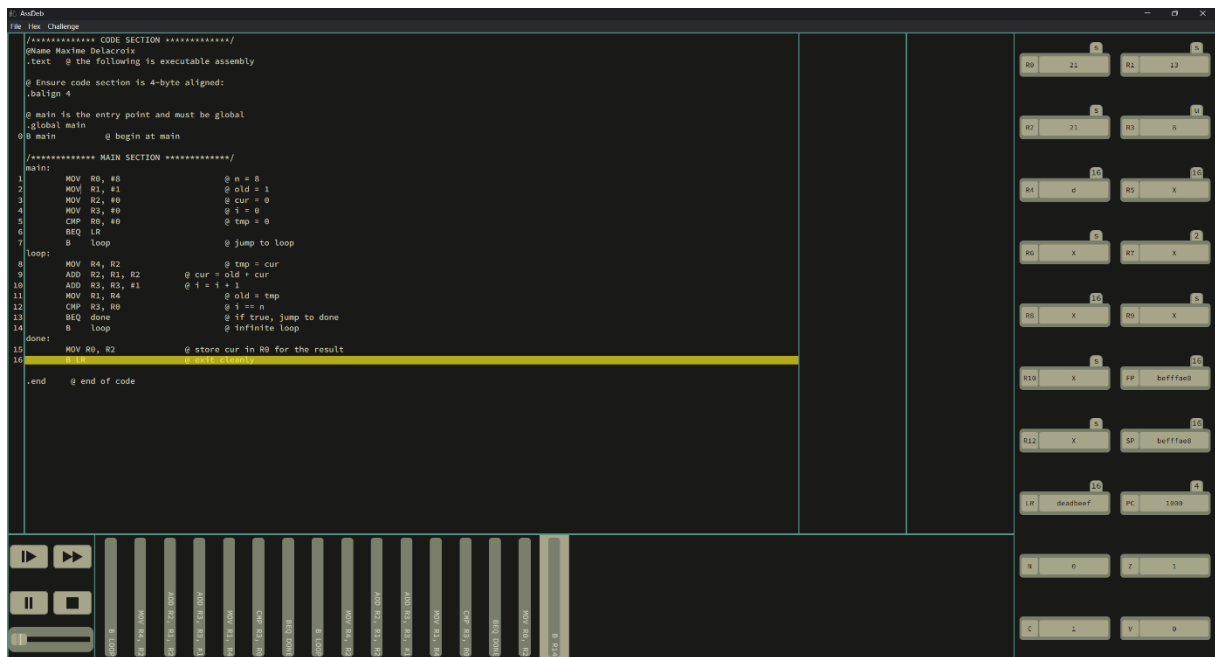
### 1. Homework\_5\_fib.s

```
1  /***** CODE SECTION *****/
2  @Name Maxime Delacroix
3  .text @ the following is executable assembly
4
5  @ Ensure code section is 4-byte aligned:
6  .balign 4
7
8  @ main is the entry point and must be global
9  .global main
10 B main @ begin at main
11
12 /***** MAIN SECTION *****/
13 main:
14     MOV R0, #8 @ n = 8
15     MOV R1, #1 @ old = 1
16     MOV R2, #0 @ cur = 0
17     MOV R3, #0 @ i = 0
18     CMP R0, #0 @ tmp = 0
19     BEQ LR
20     B loop @ jump to loop
21 loop:
22     MOV R4, R2 @ tmp = cur
23     ADD R2, R1, R2 @ cur = old + cur
24     ADD R3, R3, #1 @ i = i + 1
25     MOV R1, R4 @ old = tmp
26     CMP R3, R0 @ i == n
27     BEQ done @ if true, jump to done
28     B loop @ infinite loop
29 done:
30     MOV R0, R2 @ store cur in R0 for the result
31     B LR @ exit cleanly
32
33 .end @ end of code
```

### 2. Result of fib(8)

The result of fib(8) = 21 = 0x15

### 3. AssDeb



## Floating point

### 1. Hand analysis

a)	2	=	0 1000 0000 0000 0000 0000 0000	=	0x4000 0000
b)	3,5	=	0 1000 0000 1100 0000 0000 0000	=	0x4060 0000
c)	0,50390625	=	0 0111 1110 0000 0010 0000 0000	=	0x3F01 0000
d)	65535.6875	=	0 1000 1110 1111 1111 1111 0110 0000	=	0x477F FF80
e)	65536.19	=	0 1000 1111 0000 0000 0000 0011 0000	=	0x4780 0019

## 2. Homework\_5\_fp.s

```
1  /***** CODE SECTION *****/
2  @Name Maxime Delacroix
3  @Floating Point Addition
4  .text @ the following is executable assembly
5
6  @ Ensure code section is 4-byte aligned:
7  .balign 4
8
9  @ main is the entry point and must be global
10 .global main
11 b main @ begin at main
12
13 /***** FPNUMS *****/
14 @ These addresses contain the two fp numbers to be added
15
16 fpNum0: .word 0x40600000
17 fpNum1: .word 0x40400000
18
19
20 /***** MAIN SECTION *****/
21 main:
22     ldr r0, fpNum0 @ r0 = fpNum0
23     ldr r1, fpNum1 @ r1 = fpNum1
24
25     @ Shift the numbers to the right by 23 bits to
26     @ get the exponents
27     lsr r2, r0, #23 @ r2 = r0 >> 23
28     lsr r3, r1, #23 @ r3 = r1 >> 23
29
30     @ Mask the exponent to get the fraction
31     lsl r4, r0, #8 @ r4 = r0 << 8
32     lsl r5, r1, #8 @ r5 = r1 << 8
33     orr r4, r4, #0x80000000 @ Adding the leading 1
34     orr r5, r5, #0x80000000 @ Adding the leading 1
35
36     @ Subtract and raise the flags
37     subs r6, r2, r3 @ r6 = r2 - r3
38
39     @ Compare the exponent and store the bigger one in r7
40     movge r7, r2 @ r7 = r2 if r2 >= r3
41     movlt r7, r3 @ r7 = r3 if r2 < r3
42
43     @ Shift the fraction
44     lsrge r5, r5, r6 @ r5 = r5 >> r6 if r2 > r3
45     sublt r6, r3, r2 @ r6 = r3 - r2 if r2 < r3
46     lsrlt r4, r4, r6 @ r4 = r4 >> r6 if r2 < r3
47
48     @ Add the fraction
49     adds r4, r4, r5 @ r4 = r4 + r5
50     @ r4 hold now the sum of the fraction
51
52     @ Normalize the fraction
53     lsrce r4, r4, #1 @ r4 = r4 >> 1 if r4 overflowed
54     addcs r7, r7, #1 @ r7 = r7 + 1 if r4 overflowed
55
56     @ Rounding the result
57
58     @ Strip the leading 1 off the resulting mantissa
59     @ and merge the signe, exponent and fraction bits
60     lsl r7, r7, #23 @ r7 = r7 << 23
61     lsl r4, r4, #1 @ r4 = r4 << 1
62     lsr r4, r4, #9 @ r4 = r4 >> 9
63     orr r0, r7, r4 @ r0 = r7 | r4
64
65     b lr
66 .end @ end of code
```

### 3. Result of additions

1.0 + 1.0 = 2.0 (0x3F80 0000 + 0x3F80 0000 = 0x4000 0000)

2.0 + 1.0 = 3.0 (0x4000 0000 + 0x3F80 0000 = 0x4040 0000)

3.0 + 3.5 = 6.5 (0x4040 0000 + 0x4060 0000 = 0x40D0 0000)

0.50390625 + 65535.6875 = 65536.19 (0x3F01 0000 + 0x477F FFB0 = 0x4780 0018)

### 4. AssDeb

The screenshot displays the AssDeb debugger interface. The main window shows assembly code for a floating-point addition routine. The code is written in ARM assembly and includes comments explaining the steps: shifting numbers to the right by 23 bits to get exponents, masking the exponent to get the fraction, subtracting and raising the flags, comparing the exponent and storing the bigger one in r7, shifting the fraction, adding the fraction, normalizing the fraction, and rounding the result. The code ends with a comment indicating the end of code.

On the right side, the register window shows the values of registers R0 through R15. R0 is 47800018, R1 is 1057930144, R2 is 142, R3 is 126, R4 is 18, R5 is 6180, R6 is 16, R7 is 0x7110000000000000, R8 is 0, R9 is 0, R10 is 0, R11 is 0, R12 is 0, R13 is 0, R14 is 0, and R15 is 0. The PC register is 1200.

At the bottom, the instruction window shows the current instruction being executed, which is a branch instruction: `branch r7, #0`. The instruction is highlighted in yellow.