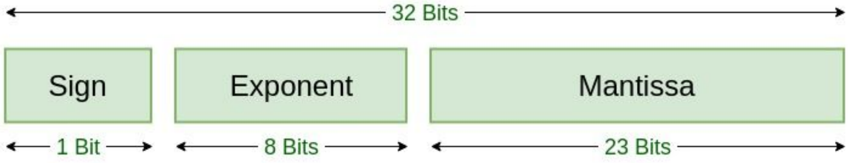
Pipelined Floating Point Adder

By Anu Soneye and Tehya Gaines

Introduction

Single Precision Floating Point Adder

- Single- precision floating point numbers are 32 bits (or two words) where the first bit denotes the sign, next 23 bits are designated as the mantissa, and the 8 bits in between are for the exponent



- The adder first compares the exponents and determines which is larger
- The result is shifted and normalized
- The adder can be executed in two or three clock cycles depending on the control path

Floating Point Addition on Paper

Compare

$$0.5 + -0.4375 = 1.000 \cdot 2^{-1} + -\frac{7}{2^4} = 1.000 \cdot 2^{-1} + -1.110 \cdot 2^{-2}$$

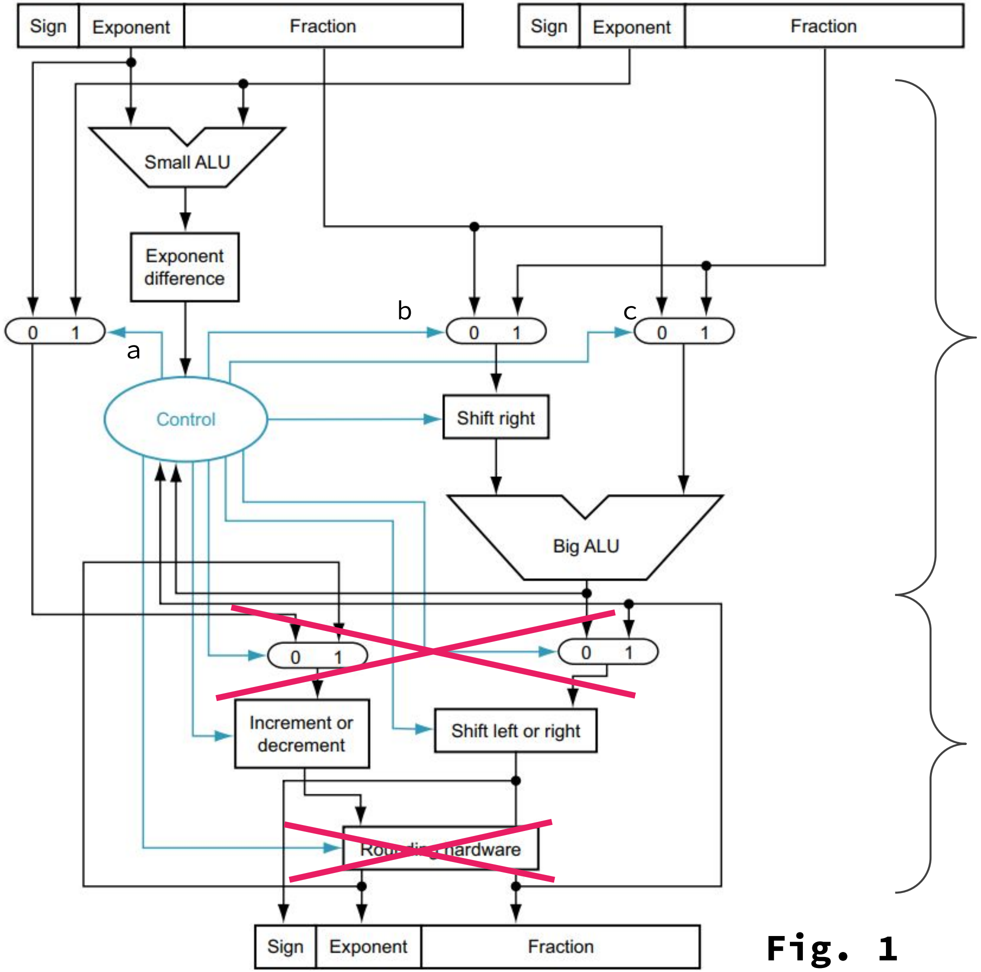
Shifted

Add

$$= 1.000 \cdot 2^{-1} + -0.111 \cdot 2^{-1} = 0.001 \cdot 2^{-1}$$

$$= \overbrace{1.000 \cdot 2}^{\textbf{Normalize}}{}^{-4} = 0.0625$$

Non-Pipelined Version



Legend:
a: Selects Largest Exponent
b: Selects Mantissa of Smaller Number
c: Selects Mantissa of Larger Number

Clock Cycle 1:
Compare, Shift,
Add (CSA)

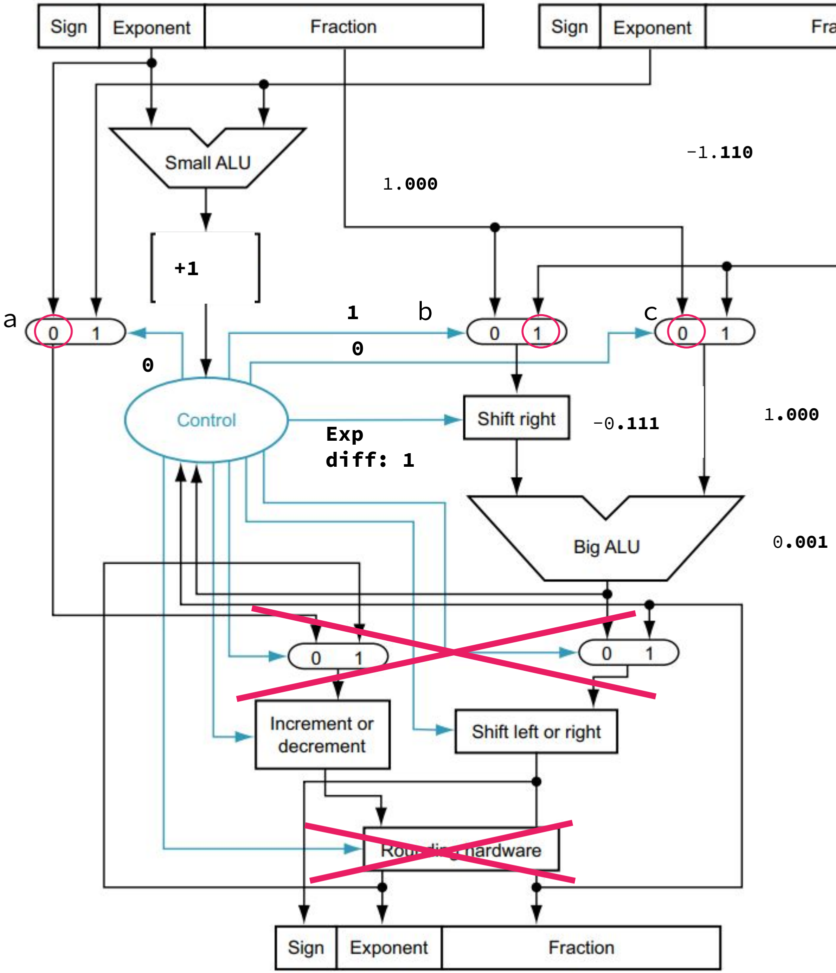
Clock Cycle 2:
Normalize

Fig. 1

Example with Non-Pipelined Version

```
0.5 =>
0 |01111110|00000000000000000000000
```

```
-0.4375 =>
1 |01111101|11000000000000000000000
```



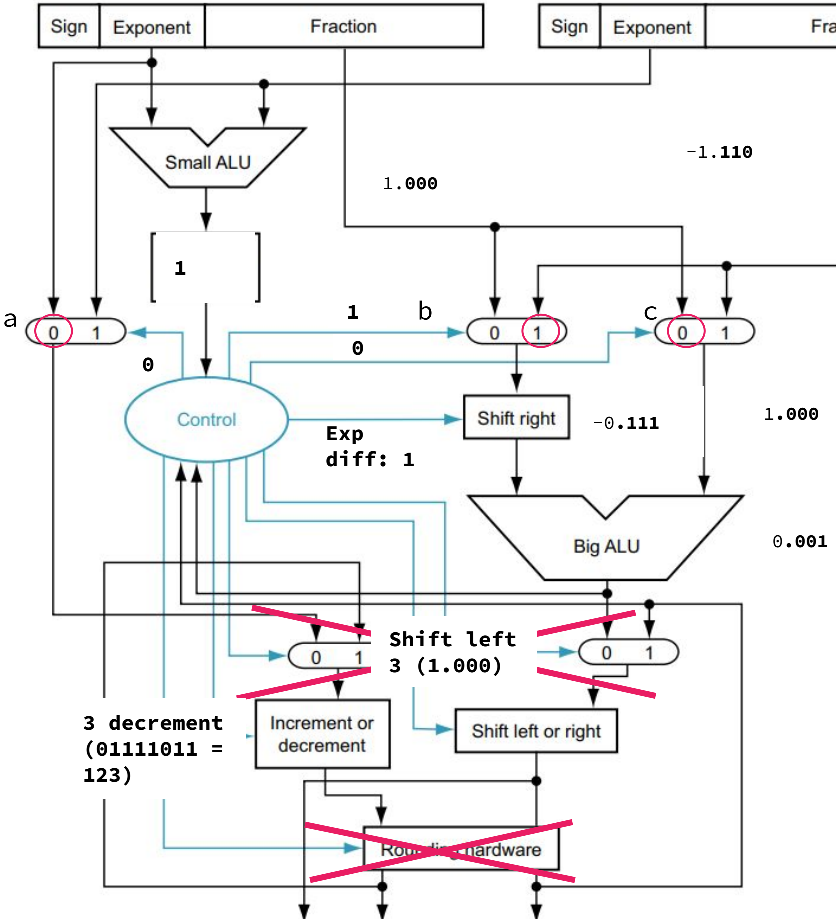
Ex: 0.5 + -0.4375

Clock Cycle 1:
Compare, Shift,
Add (CSA)

```
0.5 =>
0 |01111110|00000000000000000000000
```

```
-0.4375 =>
1 |01111101|11000000000000000000000
```

Ex: 0.5 + -0.4375



Clock Cycle 2:
Normalize

```
0|01111011|00000000000000000000000 = 0.0625
```

Module	Inputs	Output
Small ALU (subtracts)	**A:** 01111110 **B:** 01111101	1
Mux 1 (a)	**A:** 01111110 **B:** 01111101 **Sel:** 0	01111110
Mux 2 (b)	**A:**01.00000000000000000000000 **B:**01.11000000000000000000000 **Sel:** 1	01.11000000000000000000000
Mux 3 (c)	**A:**01.00000000000000000000000 **B:**01.11000000000000000000000 **Sel:** 0	01.00000000000000000000000
Shift Right	**A:**01.11000000000000000000000 **B:** 1	-00.111000000000000000000000
Big ALU (add)	**A:** 111.00100000000000000000000 (-00.111000000000000000000000) **B:** 001.00000000000000000000000 (added another bit to convert to two's complement)	00.001000000000000000000000 *highest bit with a one in abs. of result is at bit 20 (3 = 23 - 20)
Increment/ Decrement	**A:** 01111110 **B:** -1 * 3	01111011
Shift left or right	**A:** 00.001000000000000000000000 **B:** 3 **C:** 00 (left direction)	01.000000000000000000000000

Design Decisions for Pipelined Version

Why we chose creating a 2 stage pipeline over 3 stages?

- To improve the simplicity of our design, we wanted the main Control Unit to generate all of the control signals
  - This meant the main Control Unit needed the output from the Big ALU to determine the control signals for the Incrementer/Decrementer and Shift left/right components which helped normalized the result from the Big ALU
- But, if we had implemented 3 stages then we would need additional hardware separate from the main control unit in the second stage to determine increment and shifting amount for components in the third stage.
- **Benefits of Three Stages:**
  - Potentially increase the speedup (3) when we have many tasks
  - Reduce the clock cycle time period
- **Benefits of Two Stage:**
  - Prioritize simplicity in how control signals are generated
  - Don’t need control signal generating components besides the main Control Unit

Pipelined Version (two stages)

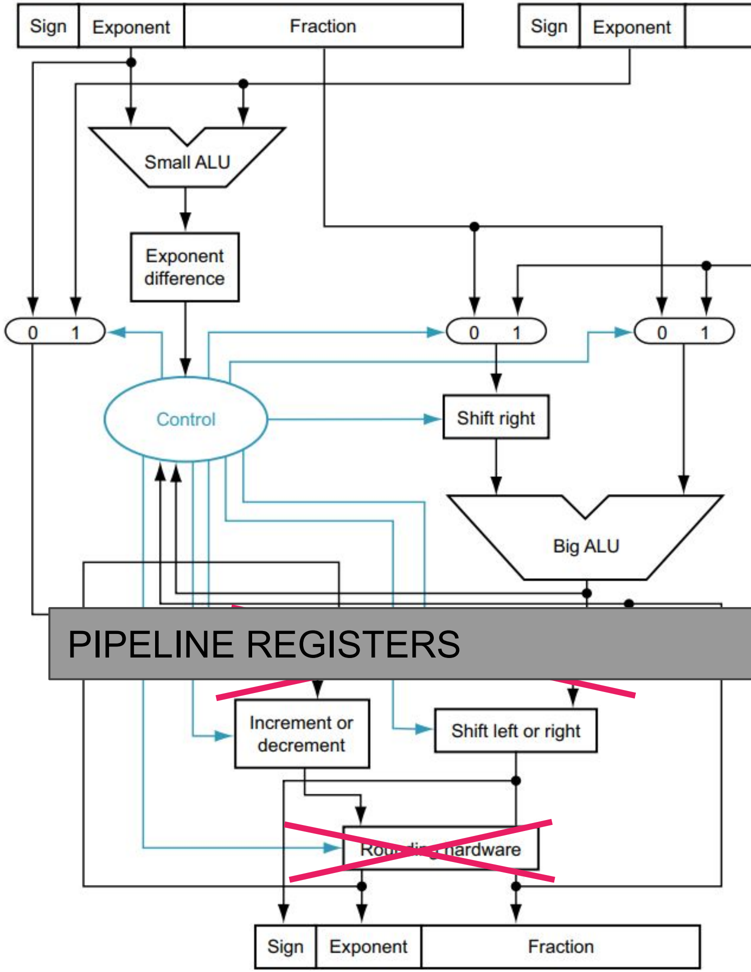
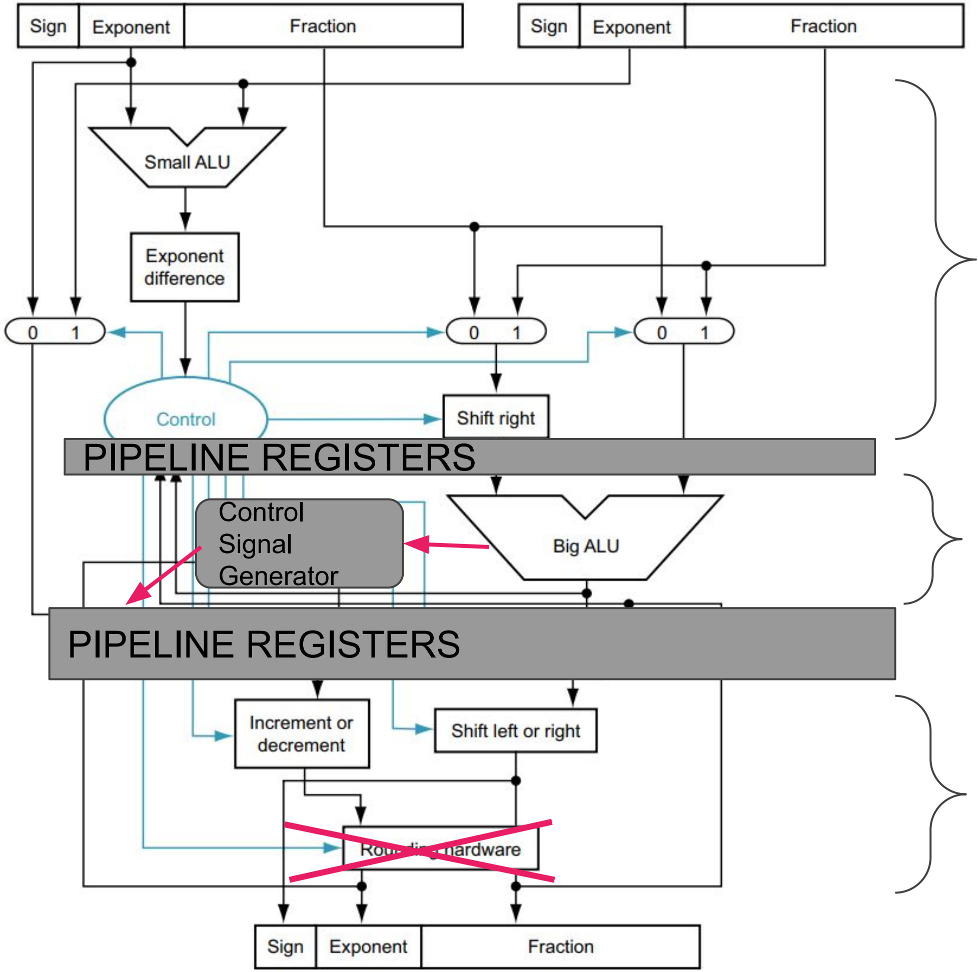


Fig. 5

STAGE 1:
Compare, Shift,
Add (CSA)

STAGE 2:
Normalize

Pipelined Version (three stages)



STAGE 1: Compare, Shift (CS)

STAGE 2: Add

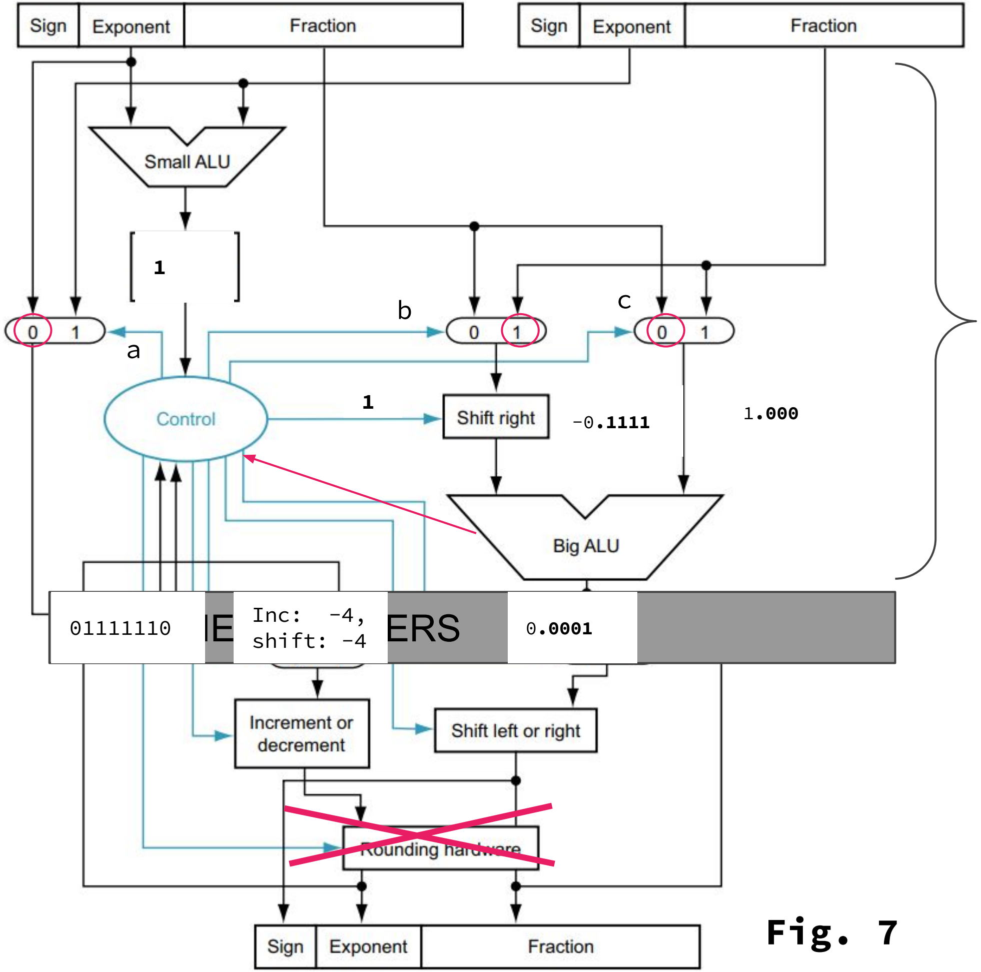
STAGE 3: Normalize

Fig. 6

Example with Pipelined Version (two stages)

```
0.5 =>
0 |01111110|00000000000000000000000
```

```
-0.46875 =>
1 |01111101|11100000000000000000000
```

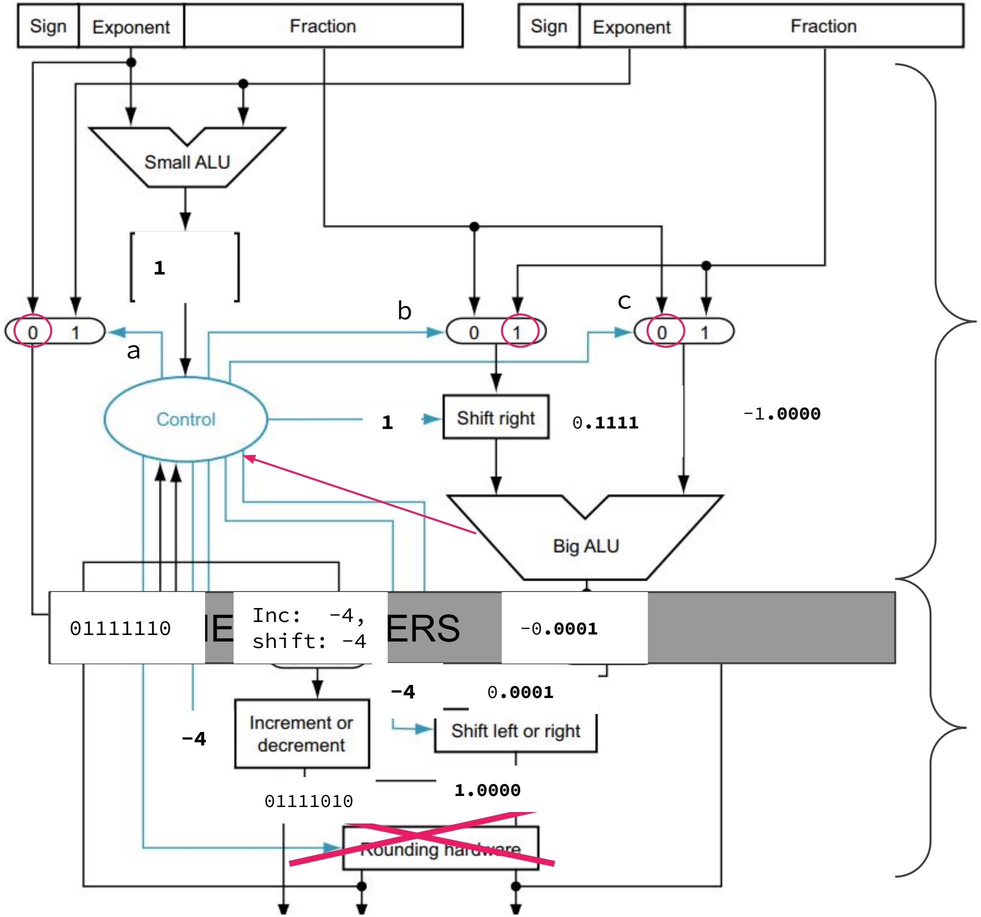


Task 1: 0.5 + -0.46875

Fig. 7

```
-0.5 =>
1 |01111110|00000000000000000000000
```

```
0.46875 =>
0 |01111101|11100000000000000000000
```



Task 2: -0.5 + 0.46875

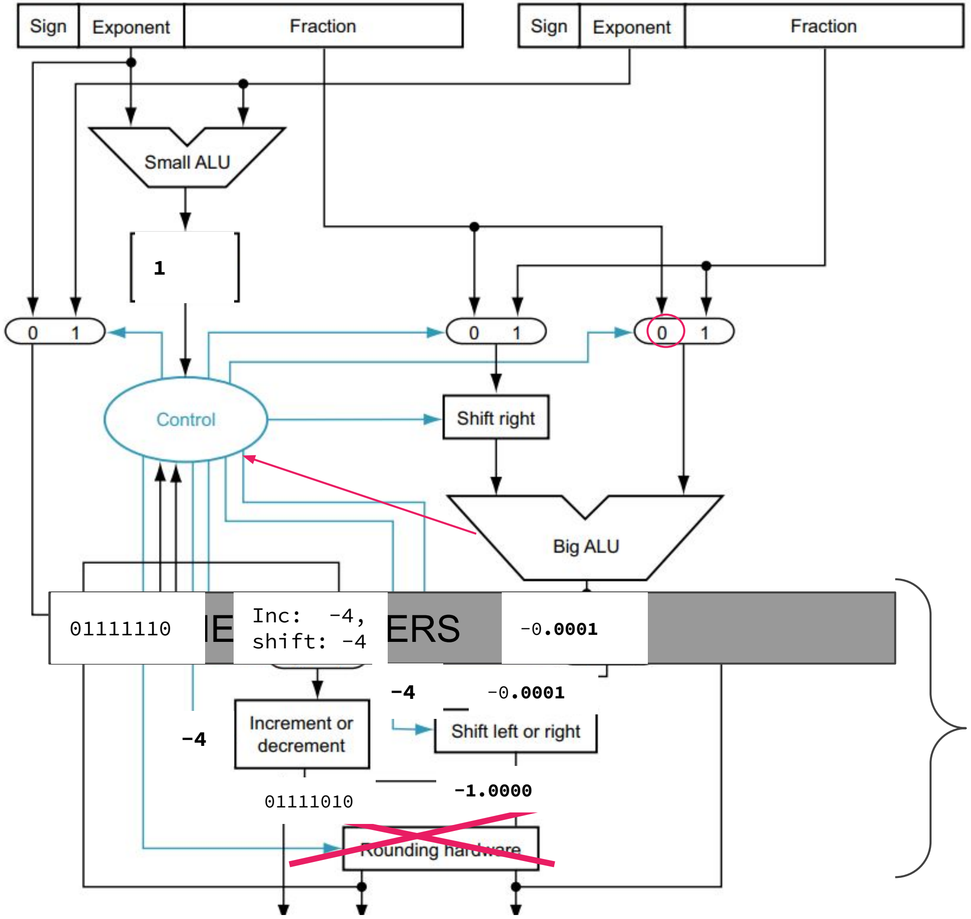
Task 1: 0.5 + -0.46875

0|01111010|00000000000000000000000 = 0.03125

Fig. 8

```
-0.5 =>
1 |01111110|00000000000000000000000
```

```
0.46875 =>
0 |01111101|11100000000000000000000
```

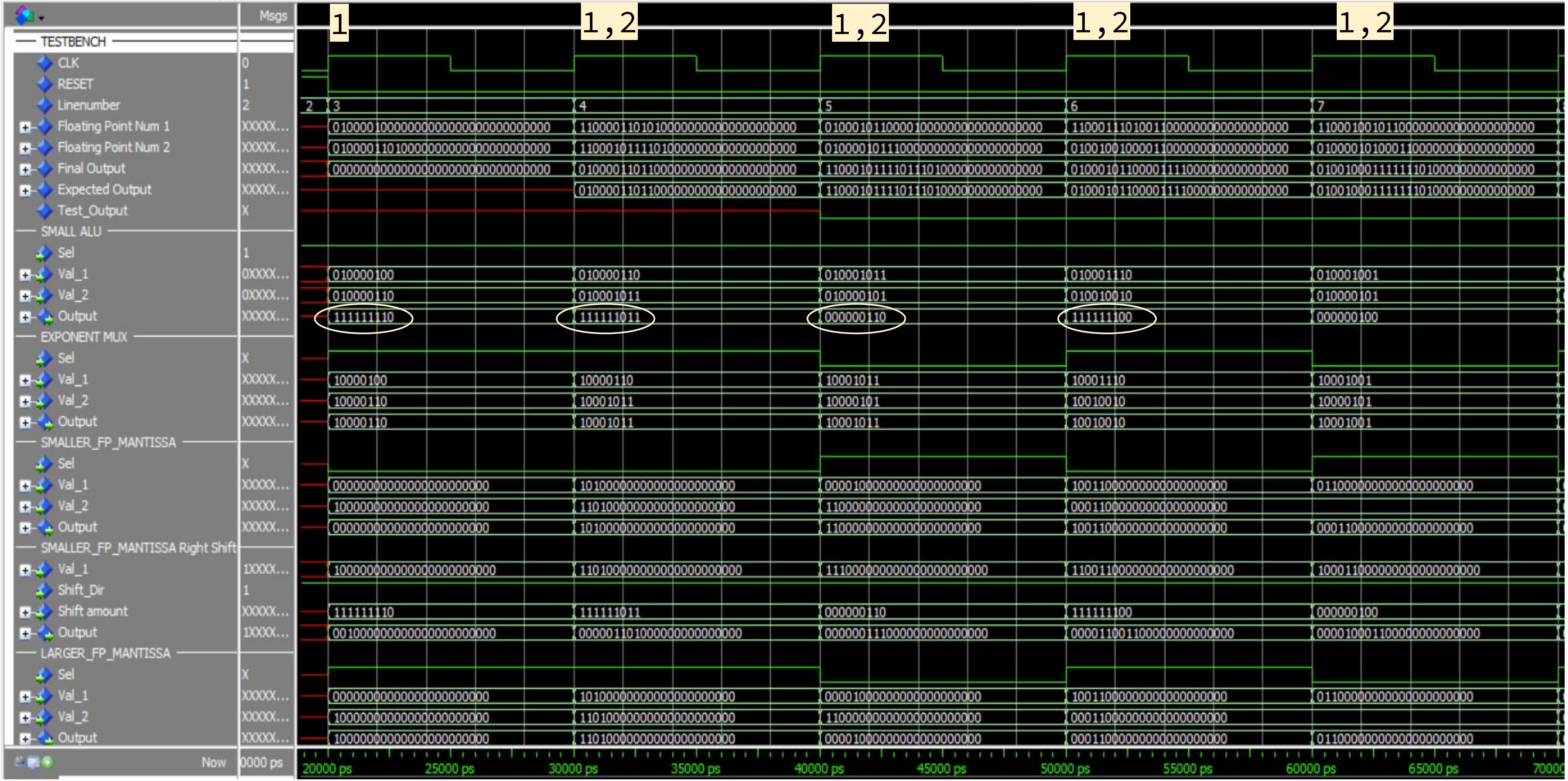


1|01111010|00000000000000000000000 = -0.03125

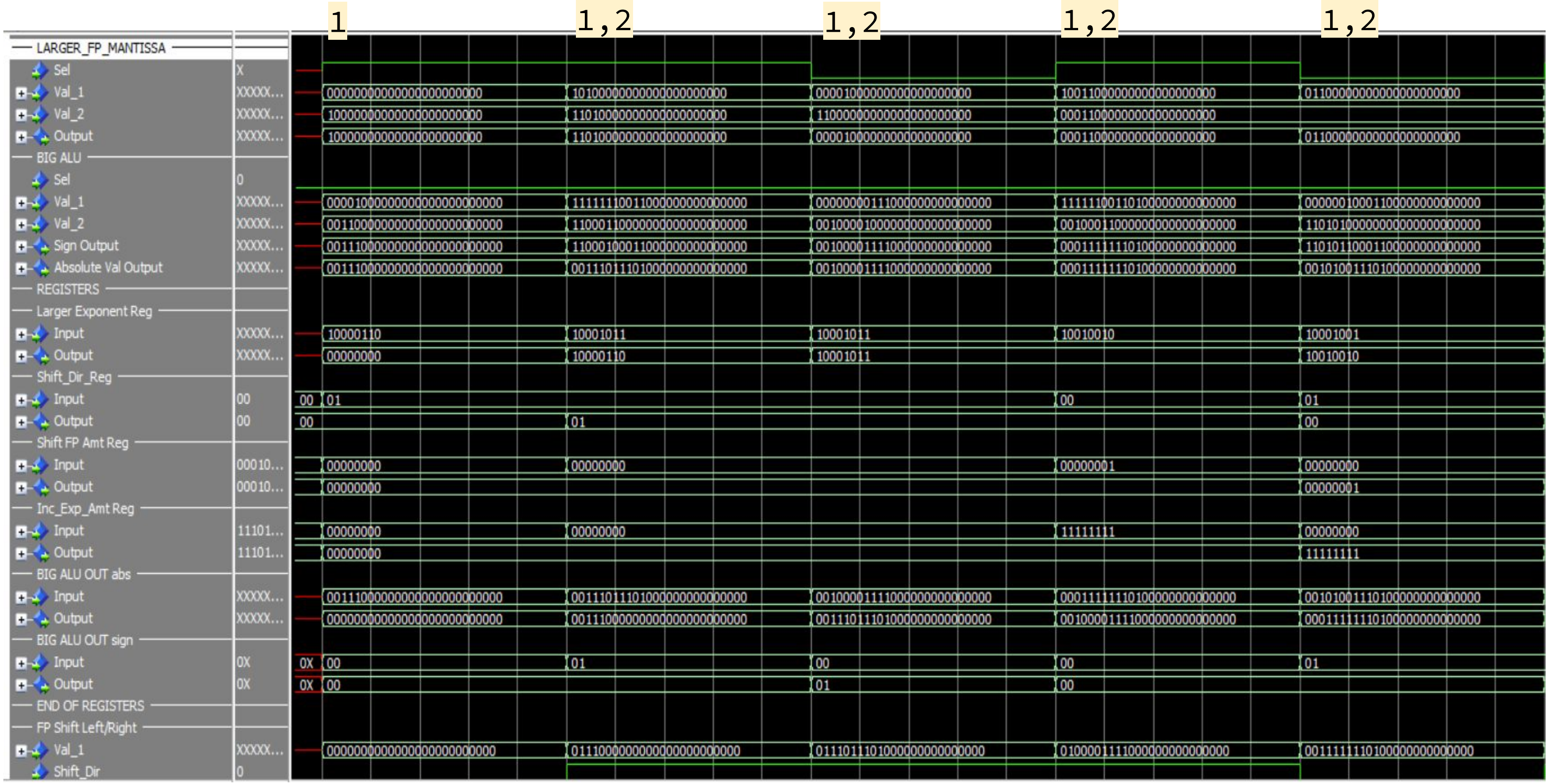
Fig. 9

Task 2: -0.5
+ 0.46875

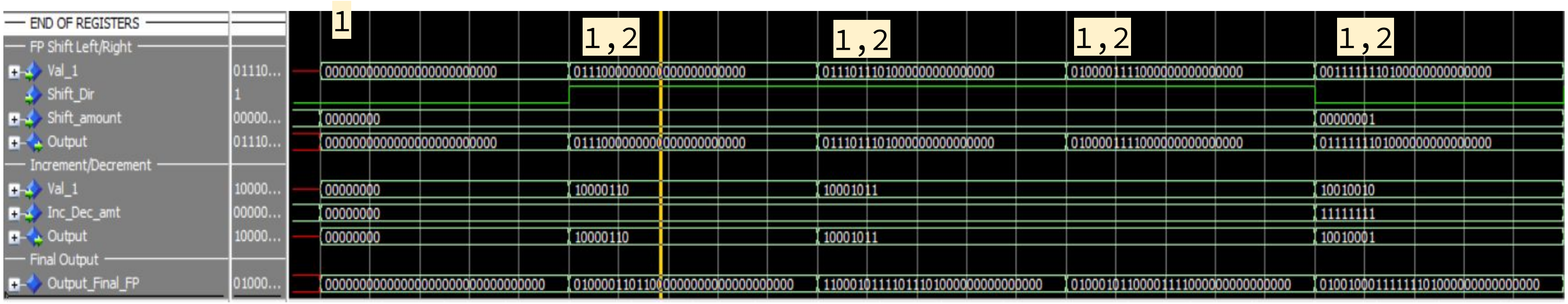
Demonstration and Waveform



	Msgs					
TESTBENCH						
CLK	0					
RESET	1					
Linenumber	2	3	4	5	6	7
Floating Point Num 1	XXXXX...	01000010000000000000000000000000	11000011010100000000000000000000	01000101100001000000000000000000	11000111010011000000000000000000	11000100101100000000000000000000
Floating Point Num 2	XXXXX...	01000011010000000000000000000000	11000101111010000000000000000000	01000010111000000000000000000000	01001001000011000000000000000000	01000010100011000000000000000000
Final Output	XXXXX...	00000000000000000000000000000000	01000011011000000000000000000000	11000101111101110100000000000000	01000101100001111000000000000000	01001000111111101000000000000000
Expected Output	XXXXX...		01000011011000000000000000000000	11000101111101110100000000000000	01000101100001111000000000000000	01001000111111101000000000000000
Test_Output	X					
SMALL ALU						
Sel	1					
Val_1	0XXXX...	010000100	010000110	010001011	010001110	010001001
Val_2	0XXXX...	010000110	010001011	010000101	010010010	010000101
Output	XXXXX...	111111110	111111011	000000110	111111100	000000100
EXPONENT MUX						
Sel	X					
Val_1	XXXXX...	10000100	10000110	10001011	10001110	10001001
Val_2	XXXXX...	10000110	10001011	10000101	10010010	10000101
Output	XXXXX...	10000110	10001011	10001011	10010010	10001001
SMALLER_FP_MANTISSA						
Sel	X					
Val_1	XXXXX...	00000000000000000000000	10100000000000000000000	00001000000000000000000	10011000000000000000000	01100000000000000000000
Val_2	XXXXX...	10000000000000000000000	11010000000000000000000	11000000000000000000000	00011000000000000000000	
Output	XXXXX...	00000000000000000000000	10100000000000000000000	11000000000000000000000	10011000000000000000000	00011000000000000000000
SMALLER_FP_MANTISSA Right Shift						
Val_1	1XXXX...	100000000000000000000000	110100000000000000000000	111000000000000000000000	110011000000000000000000	100011000000000000000000
Shift_Dir	1					
Shift amount	XXXXX...	111111110	111111011	000000110	111111100	000000100
Output	1XXXX...	001000000000000000000000	000001101000000000000000	000000111000000000000000	000011001100000000000000	000010001100000000000000
LARGER_FP_MANTISSA						
Sel	X					
Val_1	XXXXX...	00000000000000000000000	10100000000000000000000	00001000000000000000000	10011000000000000000000	01100000000000000000000
Val_2	XXXXX...	10000000000000000000000	11010000000000000000000	11000000000000000000000	00011000000000000000000	
Output	XXXXX...	10000000000000000000000	11010000000000000000000	00001000000000000000000	00011000000000000000000	01100000000000000000000



			1	1,2	1,2	1,2	1,2
LARGER_FP_MANTISSA							
Sel	X						
Val_1	XXXXX...		00000000000000000000000	10100000000000000000000	00001000000000000000000	10011000000000000000000	01100000000000000000000
Val_2	XXXXX...		10000000000000000000000	11010000000000000000000	11000000000000000000000	00011000000000000000000	
Output	XXXXX...		10000000000000000000000	11010000000000000000000	00001000000000000000000	00011000000000000000000	01100000000000000000000
BIG ALU							
Sel	0						
Val_1	XXXXX...		0000100000000000000000000	1111111001100000000000000	0000000011100000000000000	1111110011010000000000000	0000001000110000000000000
Val_2	XXXXX...		0011000000000000000000000	1100011000000000000000000	0010000100000000000000000	0010001100000000000000000	1101010000000000000000000
Sign Output	XXXXX...		0011100000000000000000000	1100010001100000000000000	0010000111100000000000000	0001111111010000000000000	1101011000110000000000000
Absolute Val Output	XXXXX...		0011100000000000000000000	0011101110100000000000000	0010000111100000000000000	0001111111010000000000000	0010100111010000000000000
REGISTERS							
Larger Exponent Reg							
Input	XXXXX...		10000110	10001011	10001011	10010010	10001001
Output	XXXXX...		00000000	10000110	10001011		10010010
Shift_Dir_Reg							
Input	00	00	01			00	01
Output	00	00		01			00
Shift FP Amt Reg							
Input	00010...		00000000	00000000		00000001	00000000
Output	00010...		00000000				00000001
Inc_Exp_Amt Reg							
Input	11101...		00000000	00000000		11111111	00000000
Output	11101...		00000000				11111111
BIG ALU OUT abs							
Input	XXXXX...		0011100000000000000000000	0011101110100000000000000	0010000111100000000000000	0001111111010000000000000	0010100111010000000000000
Output	XXXXX...		0000000000000000000000000	0011100000000000000000000	0011101110100000000000000	0010000111100000000000000	0001111111010000000000000
BIG ALU OUT sign							
Input	0X	0X	00	01	00	00	01
Output	0X	0X	00		01	00	
END OF REGISTERS							
FP Shift Left/Right							
Val_1	XXXXX...		000000000000000000000000	011100000000000000000000	011101110100000000000000	010000111100000000000000	001111111010000000000000
Shift_Dir	0						



1 1,2 1,2 1,2 1,2

```
END OF REGISTERS
FP Shift Left/Right
Val_1           01110...  000000000000000000000000  011100000000000000000000  011101110100000000000000  010000111100000000000000  001111111010000000000000
Shift_Dir       1
Shift_amount    00000...  00000000                                                                                                00000001
Output          01110...  000000000000000000000000  011100000000000000000000  011101110100000000000000  010000111100000000000000  011111110100000000000000
Increment/Decrement
Val_1           10000...  00000000                  10000110                  10001011                                            10010010
Inc_Dec_amt     00000...  00000000                                                                                                11111111
Output          10000...  00000000                  10000110                  10001011                                            10010001
Final Output
Output_Final_FP 01000...  00000000000000000000000000000000  01000011011000000000000000000000  11000101111011101000000000000000  01000101100001111000000000000000  01001000111111101000000000000000
```

Speedup:

Non-Pipelined Time = 2n clock cycles

Pipelined Time = (n + 1) clock cycles

Speedup = 2n/(n+1), which 2 times as fast for a large amount of tasks

Demonstration