CSE 141L Demo

Runyang Tian, Sarah Reine Bulatao

University of California, San Diego, La Jolla, CA, USA

Contents

1. CPU architecture
2. Synthesized schematic
3. Software program
4. Simulation

CPU architecture

1.1 ISA

Type	Format	Corresponding Instructions
SR	6-bit op, 3-bit rs 6-bit op, 3-bit don't care	ADD, SUB, AND, OR, XOR, SRLR BAN, BOR
LS	6-bit op, 3-bit rs	LWR, STR
SI	6-bit op, 3-bit imm	ADDI, SUBI, LWI, BRC, SRL, SLL
DR	3-bit op, 3-bit rs, 3-bit rt	EQ
GR	3-bit op, 3-bit don't care, 3-bit rs	MOV
J	3-bit op, 6-bit target	JR JMP

CPU architecture

1.1 ISA

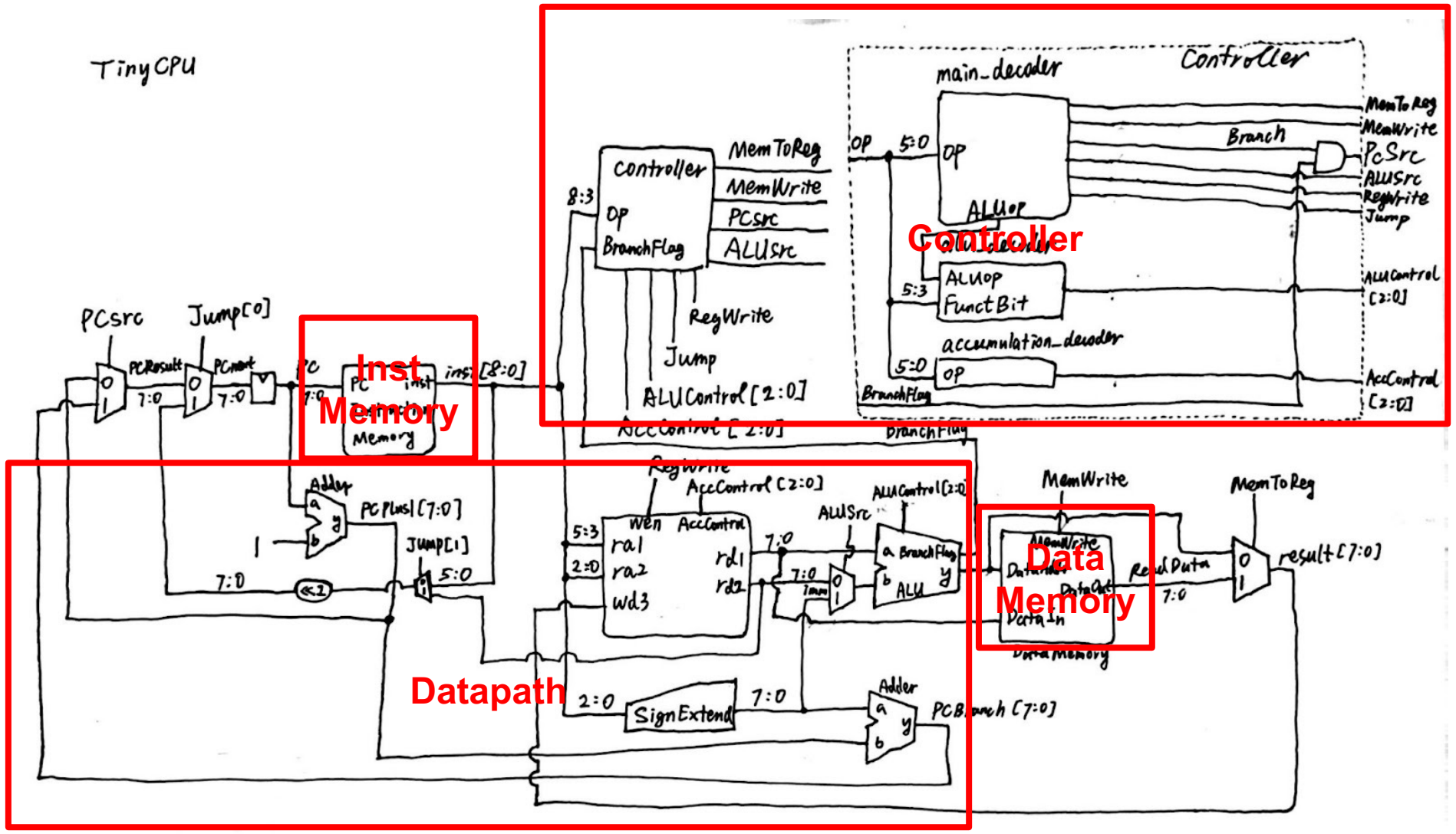
General Purpose Registers: 8-bit r0 to r7
Special Purpose Registers:
8-bit acc (Accumulator)
8-bit BranchFlag (Used for conditional branching)
8-bit pc (Program Counter)

NAME	TYPE	BIT BREAKDOWN	EXAMPLE	NOTES
ADD	SR	000000_xxx 6-bit op=000000 3-bit rs =xxx	000000_001 # Assume Acc = 0000_0010 # Assume rf[001] = 0000_0001 # After ADD, Acc = 0000_0011	Acc = Acc + rs
SUB	SR	000001_xxx 6-bit op=000001 3-bit rs =xxx	000001_010 # Assume Acc = 0000_0011 # Assume rf[010] = 0000_0001 # After SUB, Acc = 0000_0010	Acc = Acc - rs
AND	SR	000010_xxx 6-bit op=000010 3-bit rs =xxx	000010_011 # Assume Acc = 1100_1100 # Assume rf[011] = 1010_1010 # After AND, Acc = 1000_1000	Acc = Acc & rs
EQ	DR	100_xxx_yyy 3-bit op = 100 3-bit rs = xxx 3-bit rt = yyy	100_101_100 # rs = 101 # rt = 100 # rf[101] == rf[100] # BranchFlag = 1	if(rs == rt) BranchFlag = 1
MOV	GR	101_ddd_xxx 3-bit op=101 3-bit ddd 3-bit rs xxx	101_000_001 # Assume rs = 001 # Assume acc = 0000_0011 # rf[001] = 0000_0011	rs = acc

CPU architecture

1.2 Datapath

top.v
- instmem.v
- datamem.v
- mips.v
  - controller.v
    - maindec.v
    - aludec.v
    - accdec.v
  - datapath.v
    - dff.v
    - adder.v
    - mux2.v
    - regfile.v
    - signext.v
    - alu.v



https://github.com/sbulatao/CSE141L-project.git

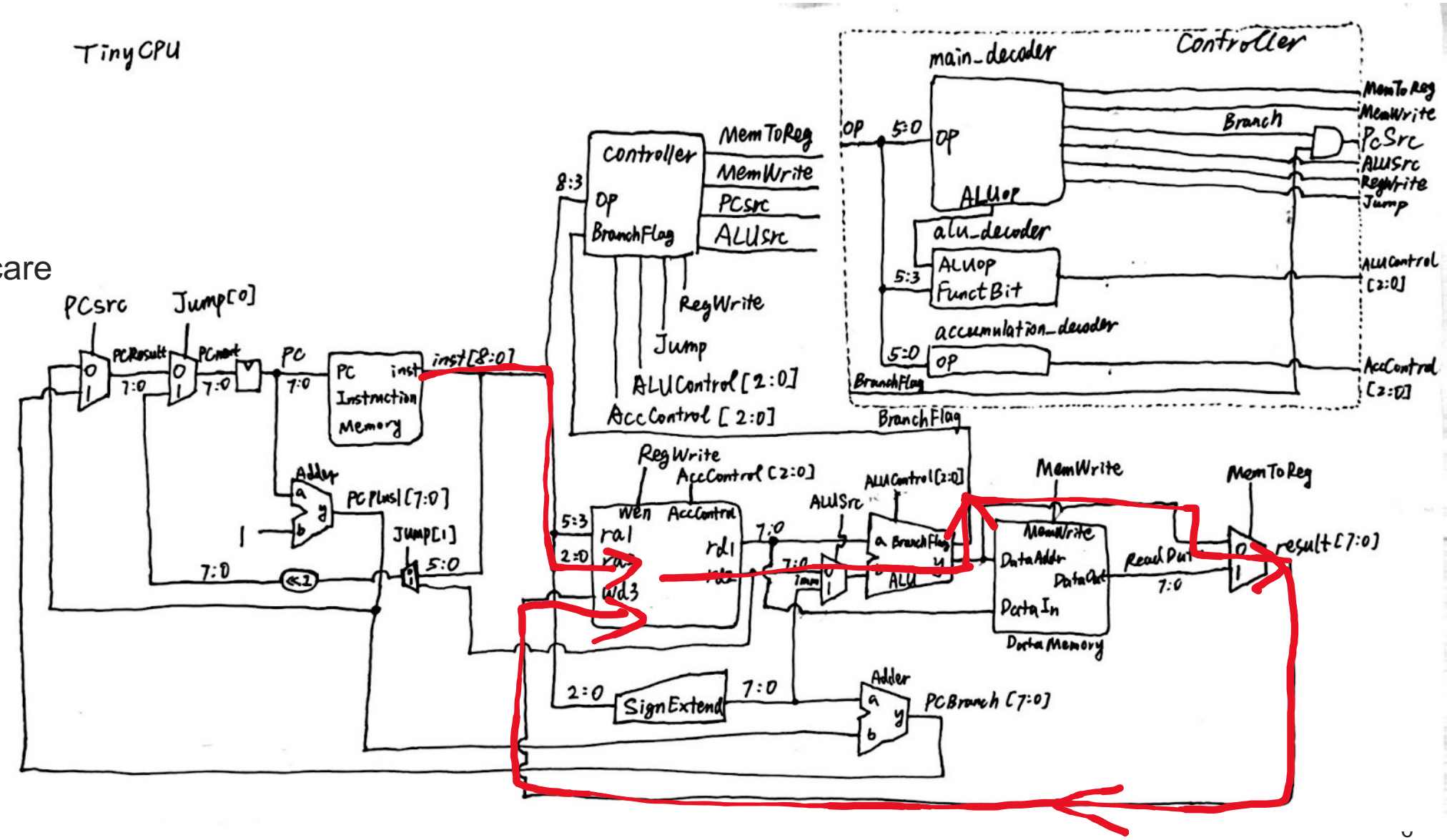
CPU architecture

1.2 Datapath

SR

6-bit op, 3-bit rs
6-bit op, 3-bit don't care

ADD, SUB,
AND, OR,
XOR, SRLR
BAN, BOR



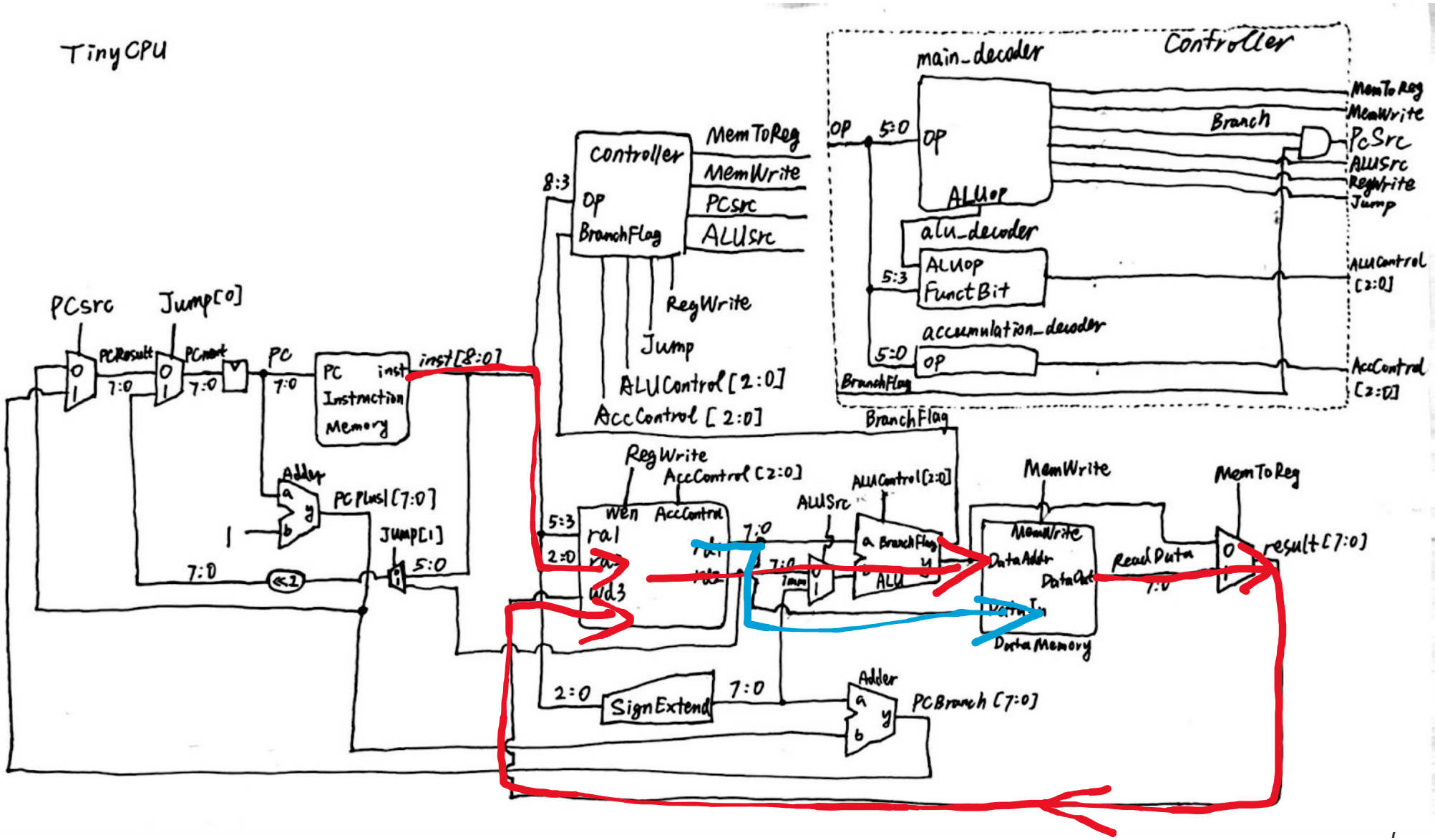
CPU architecture

1.2 Datapath

LS

6-bit op, 3-bit rs

LWR, STR



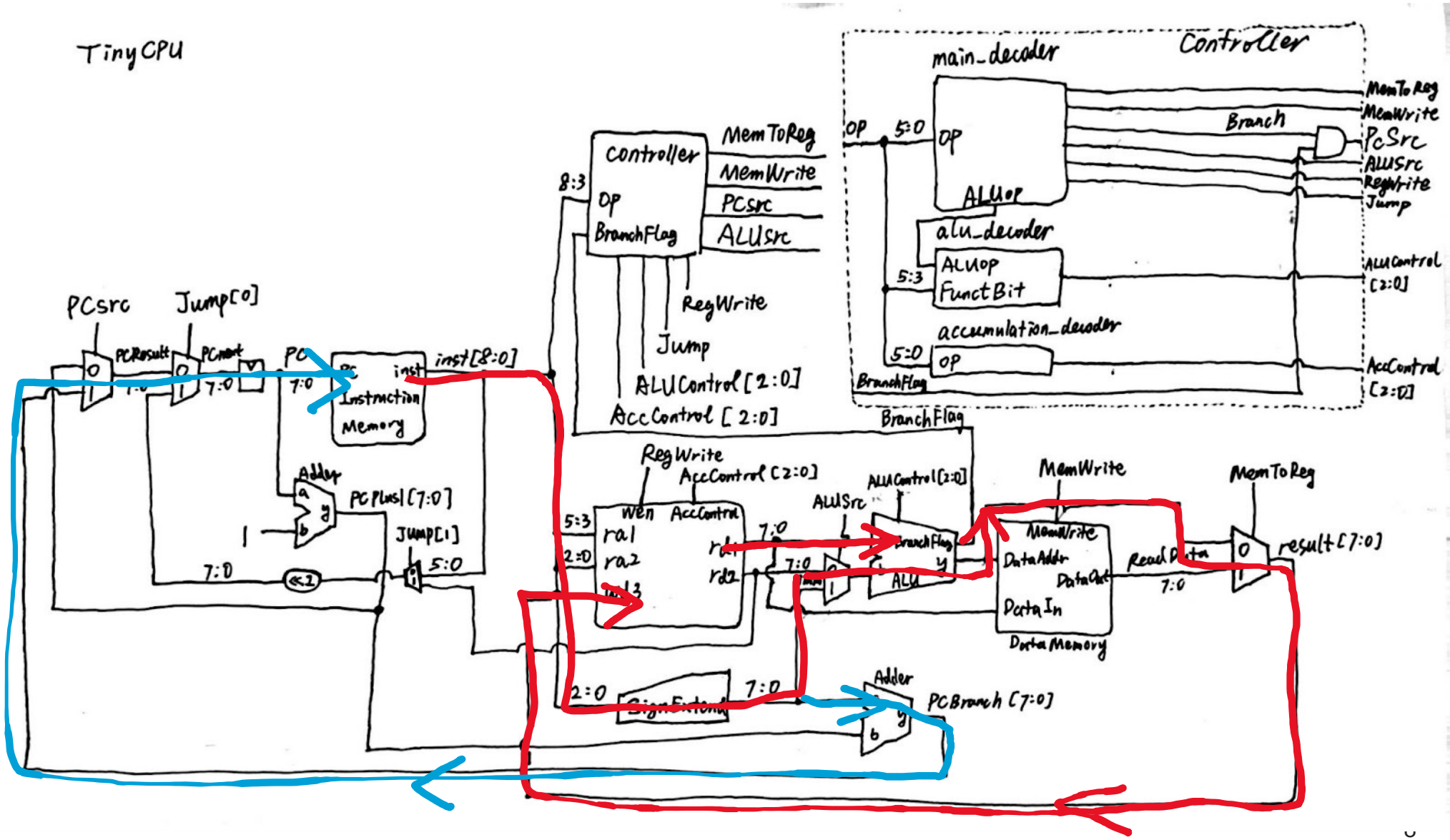
CPU architecture

1.2 Datapath

SI

6-bit op, 3-bit imm

ADDI, SUBI,
LWI, BRC,
SRL, SLL



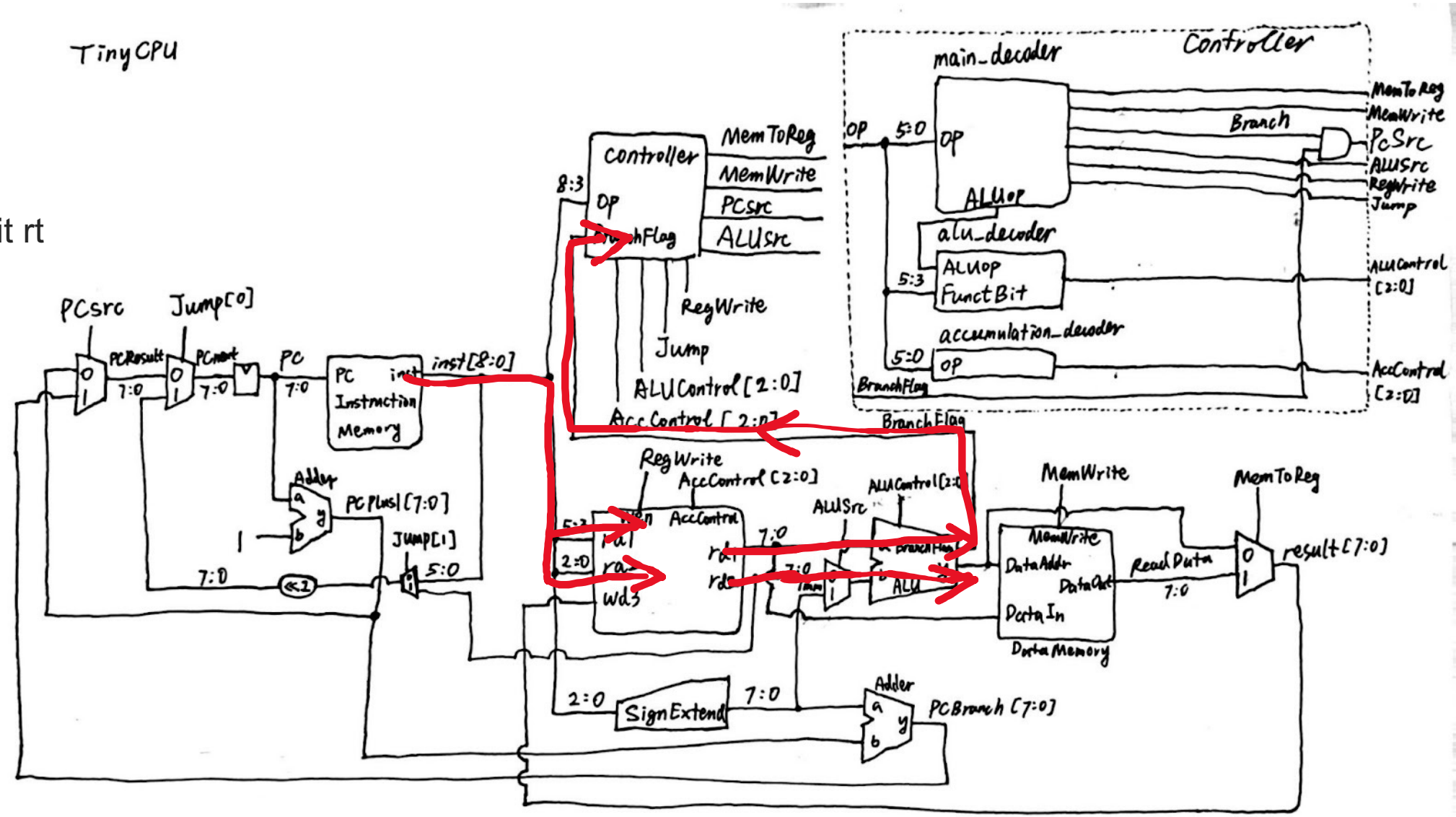
CPU architecture

1.2 Datapath

DR

3-bit op, 3-bit rs, 3-bit rt

EQ



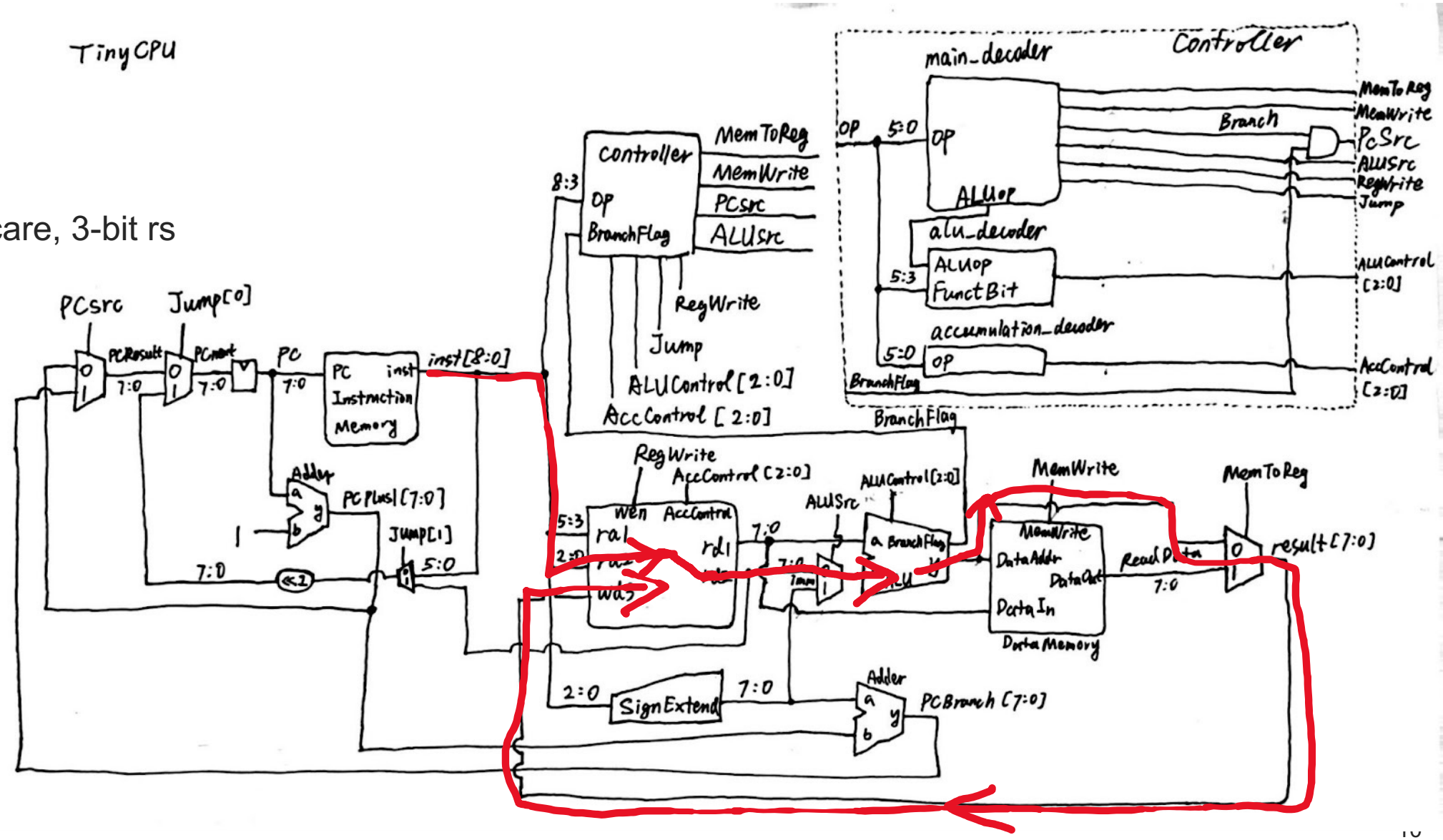
CPU architecture

1.2 Datapath

GR

3-bit op, 3-bit don't care, 3-bit rs

MOV



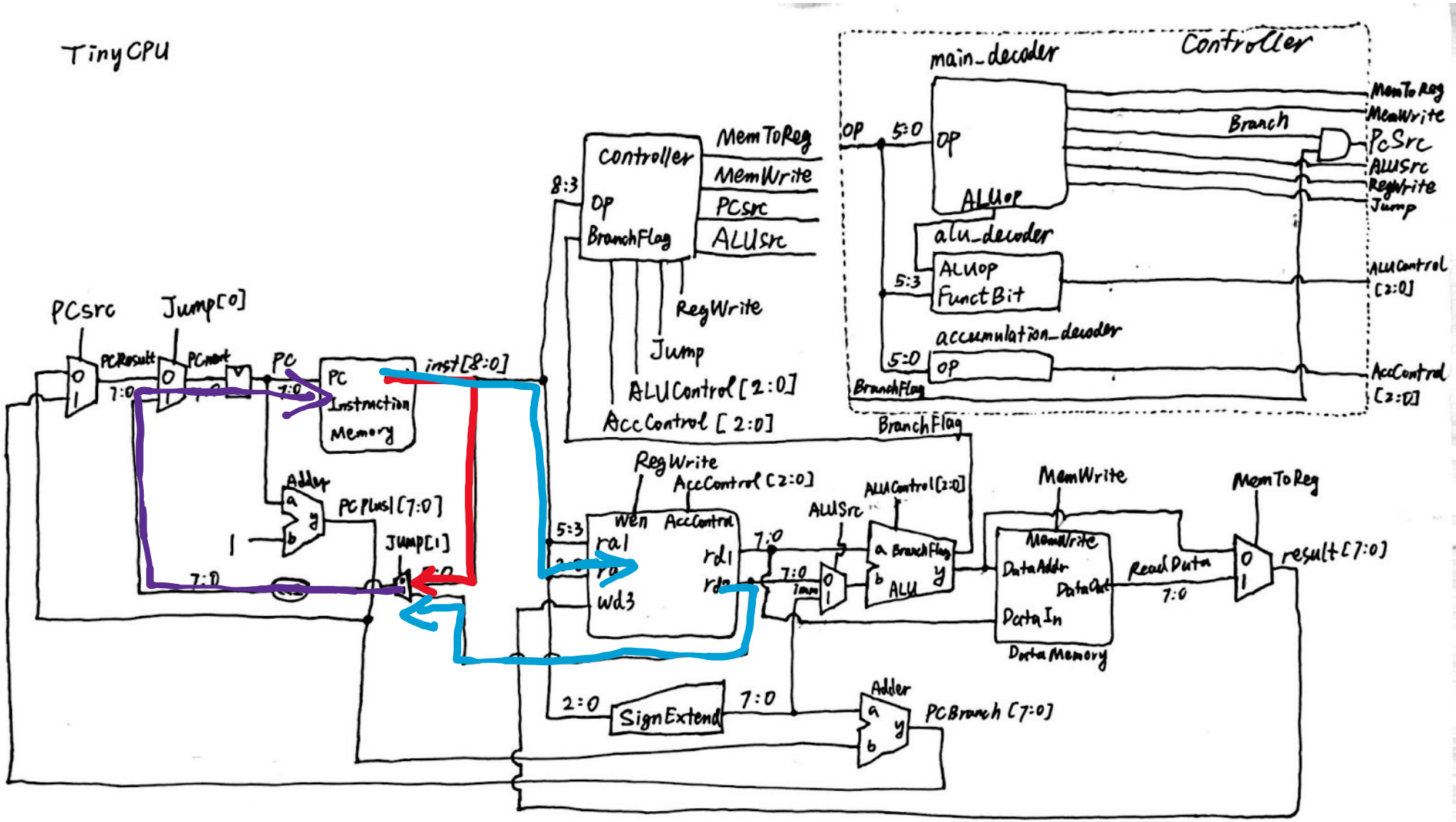
CPU architecture

1.2 Datapath

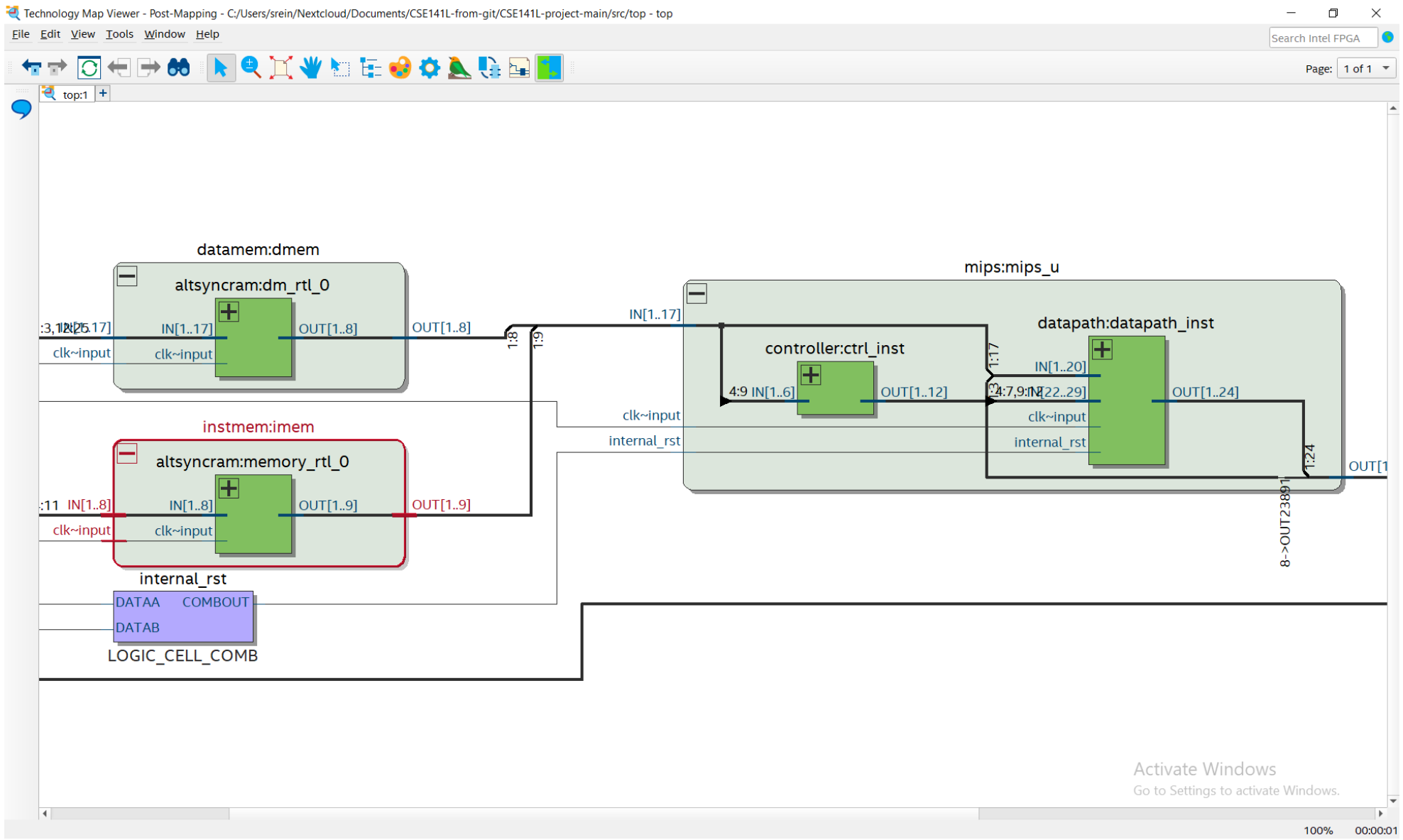
J

3-bit op, 6-bit target

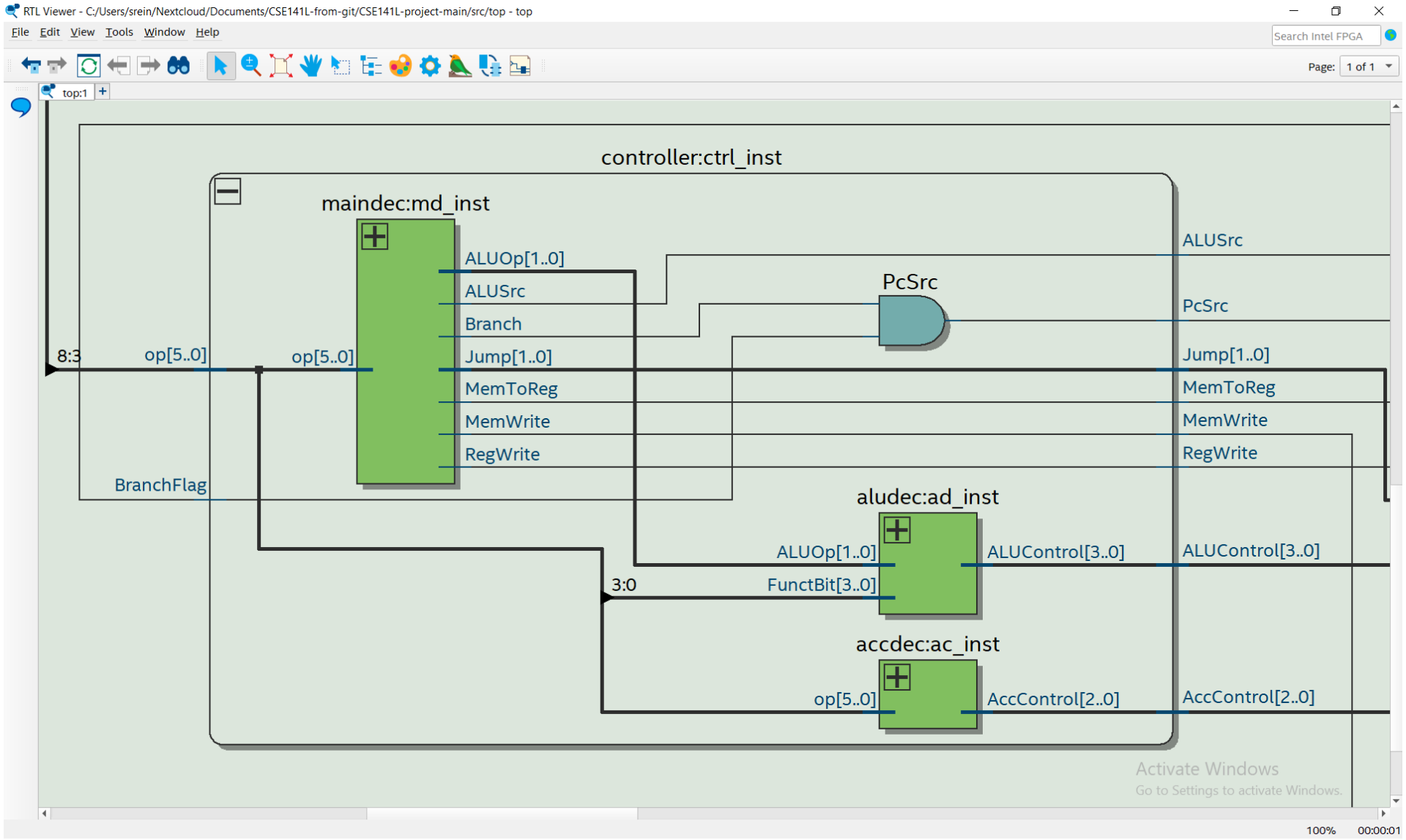
JR, JMP



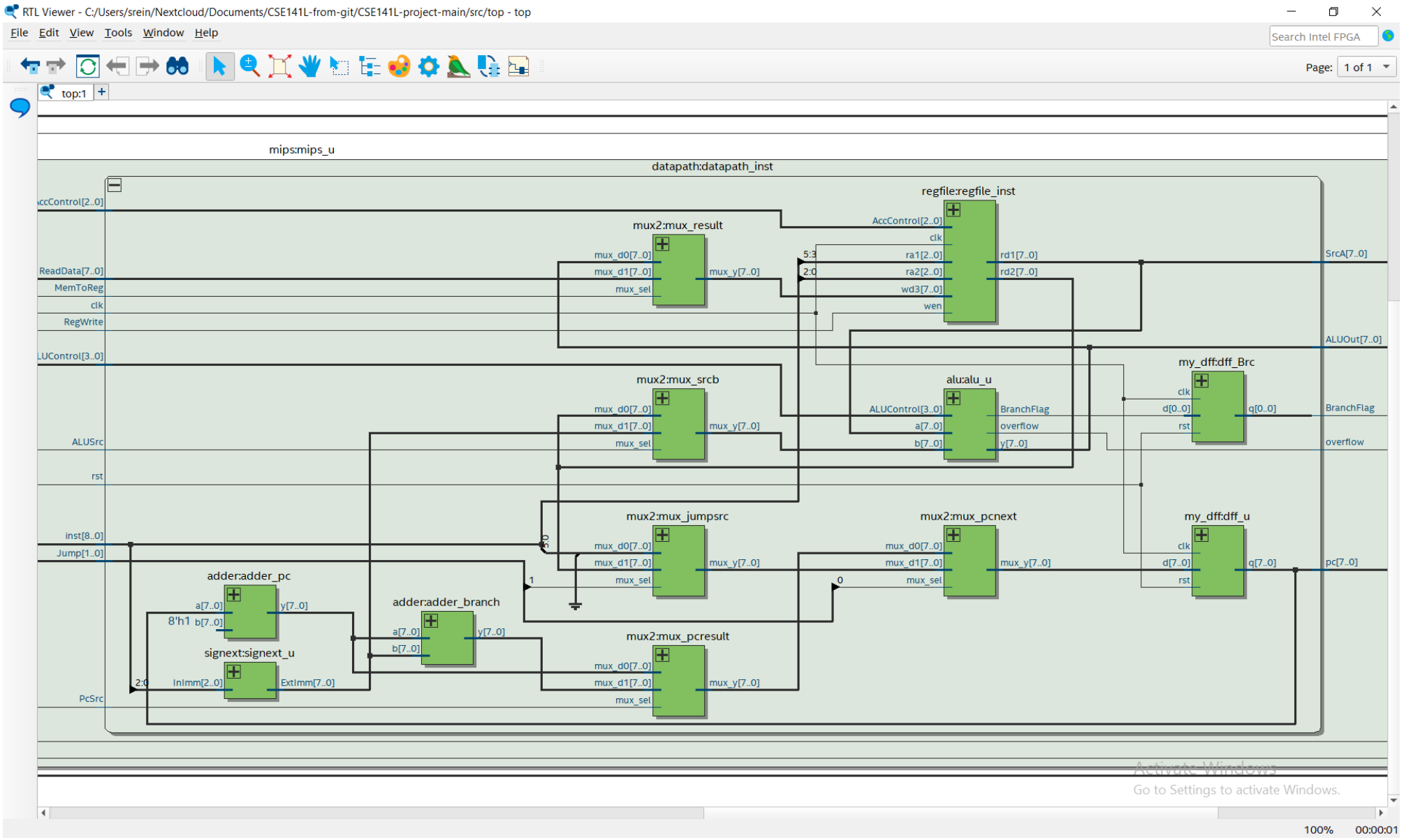
Synthesized schematic



Synthesized schematic



Synthesized schematic



Software program

3.1 Fix2Flt Algorithm Definition

1. Format Definition

Input FI: A 16-bit signed fixed-point number.

FI = [S | I | M]

S: 1-bit Sign

I: 7-bit Int

M: 8-bit Mantissa

Output FL: A 16-bit floating-point.

FL = [S | E | M]

S: 1-bit Sign

E: 5-bit Exponent

M: 10-bit Mantissa

2. Conversion Steps

Sign S:

S = MSB of FI (i.e., FI >> 15).

Absolute Value V:

V = abs(FI).

Special Case (Zero):

If FI[14:0] == 0, return a predefined value for zero (positive or negative) and stop.

Normal Conversion:

Find offset, the position of the most significant '1' in V (range 0-14).

Calculate Exponent E: E = 21-offset.

Calculate Mantissa M: M = The 10 bits from V that immediately follow the most significant ‘1’ or the lowest 8 bits.

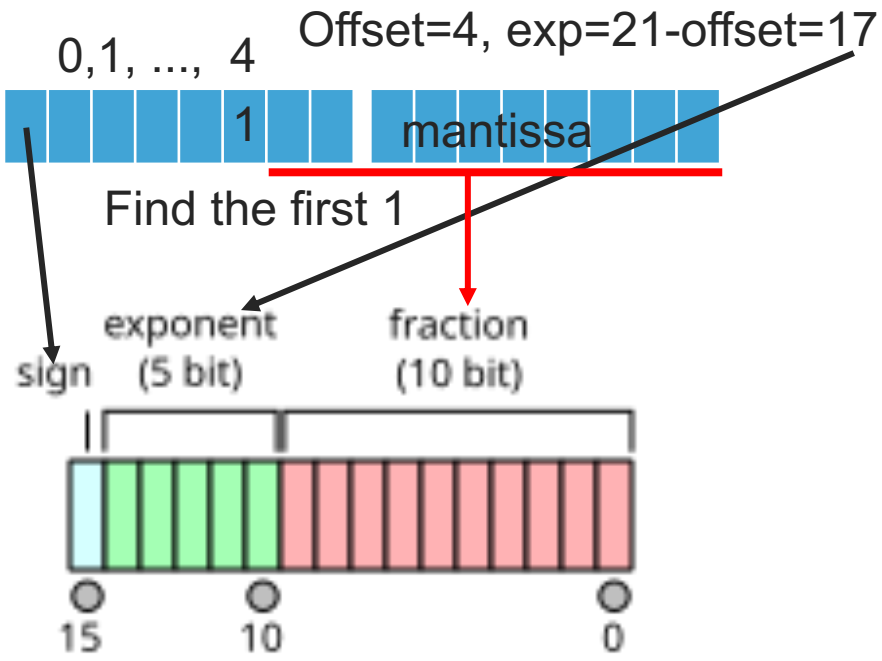
Combine:

F = (S << 15) | (E << 10) | M.

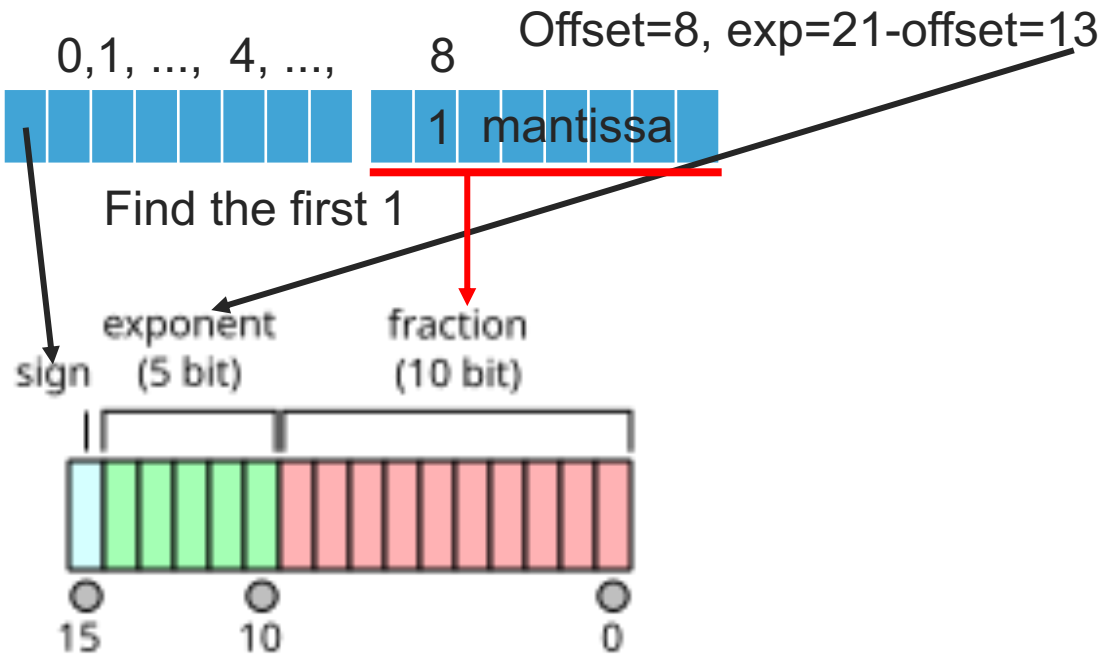
Software program

3.1 Fix2Flt Algorithm Definition

Example1: (number > 1)



Example2: (number <=1)



Software program

3.2 Testcase selection

Case1: 01110100_00111100

Case2: 00110100_11011100

Case3: 00010100_11001010

Case4: 11110001_00101110

Case5: 1000000_00000001

Case6: 11111111_11111010

Case7: 00000000_00000000

Case8: 11111111_11111111

Case9: 01111111_11111111

Case10: 10000000_00000000

Have:

number>1:	Case1,2,3,9
-1<Number<1	Case6,7,8
Number<-1	Case4,5
Number=+0	Case7
Number=-0	Case10
Very large number	Case9
Very small number	Case5
All 0	Case7
All 1	Case8

Software program

3.3 Assembler

The grammar of our assembly code:
1. Case insensitive for all instructions, registers and labels
2. Use ; in front of the comments
3. Allow blank lines
4. Use single or multiple blank spaces between instruction, register and immediate number
5. Do not use punctuations except comments and labels
6. Put label on a separate line and end it with a colon.

In the **first pass**, it reads the source file to find all labels (e.g., LABEL_1:) and records their memory addresses in a label table.

In the **second pass**, it goes through the instructions again, translating each one into its binary representation and using the completed symbol table to resolve the addresses for any jumps or branches.

Finally, it writes the generated binary code to an output file.

Software program

```
; Assembly code example
LOOP_LOW:                    ; PC=121
LWI 1           ; ACC=1
MOV R0          ; R0=1
SLL 7           ; ACC=8'B1000_0000
MOV R7          ; R7=8'B1000_0000
LWI 0           ; CLEAR ACC
OR R2           ; ACC=DATA_LOW
AND R7          ; ACC=DATA_LOW&8'B1000_0000
BOR
MOV R7          ; R7=(INT1[7]==1)
LWI 0           ; ACC=0
MOV R0          ; R0=0
EQ R7 R0
BRC EXP_mins_1_low      ; if(INT1[7]==0)
LWI 5           ; ACC=5       ; else, jump to LABEL_FINISH
SLL 5           ; ACC=5<<5=160
SUBI 1          ; ACC=159
JR
```

Machine code

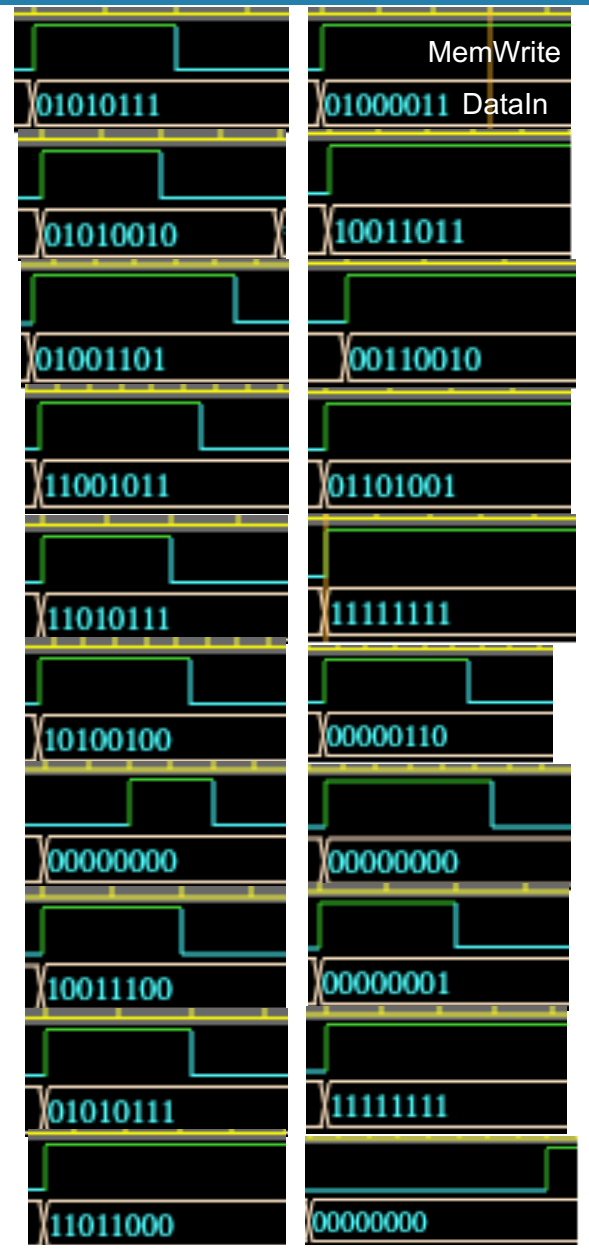
```
010010001
101000000
010111111
101000111
010010000
000011010
000010111
000110000
101000111
010010000
101000000
100111000
010101100
010010101
010111101
010001001
110000000
```

Simulation

Input fixed-point number	Python math result	CPU floating-point result
Case1: 01110100_00111100	01010111_01000011	
Case2: 00110100_11011100	01010010_10011011	
Case3: 00010100_11001010	01001101_00110010	
Case4: 11110001_00101110	11001011_01101001	
Case5: 1000000_00000001	11010111_11111111	
Case6: 11111111_11111010	10100100_00000110	
Case7: 00000000_00000000	00000000_00000000	
Case8: 11111111_11111111	10011100_00000001	
Case9: 01111111_11111111	01010111_11111111	
Case10: 10000000_00000000	11011000_00000000	

```
[ryantian@Tians-Air ~ % c
[ryantian@Tians-Air 141L_
0101011101000011
0101001010011011
0100110100110010
1100101101101001
1101011111111111
1010010000000110
0000000000000000
1001110000000001
0101011111111111
1101100000000000
ryantian@Tians-Air 141L_
```

10/10



Thank you