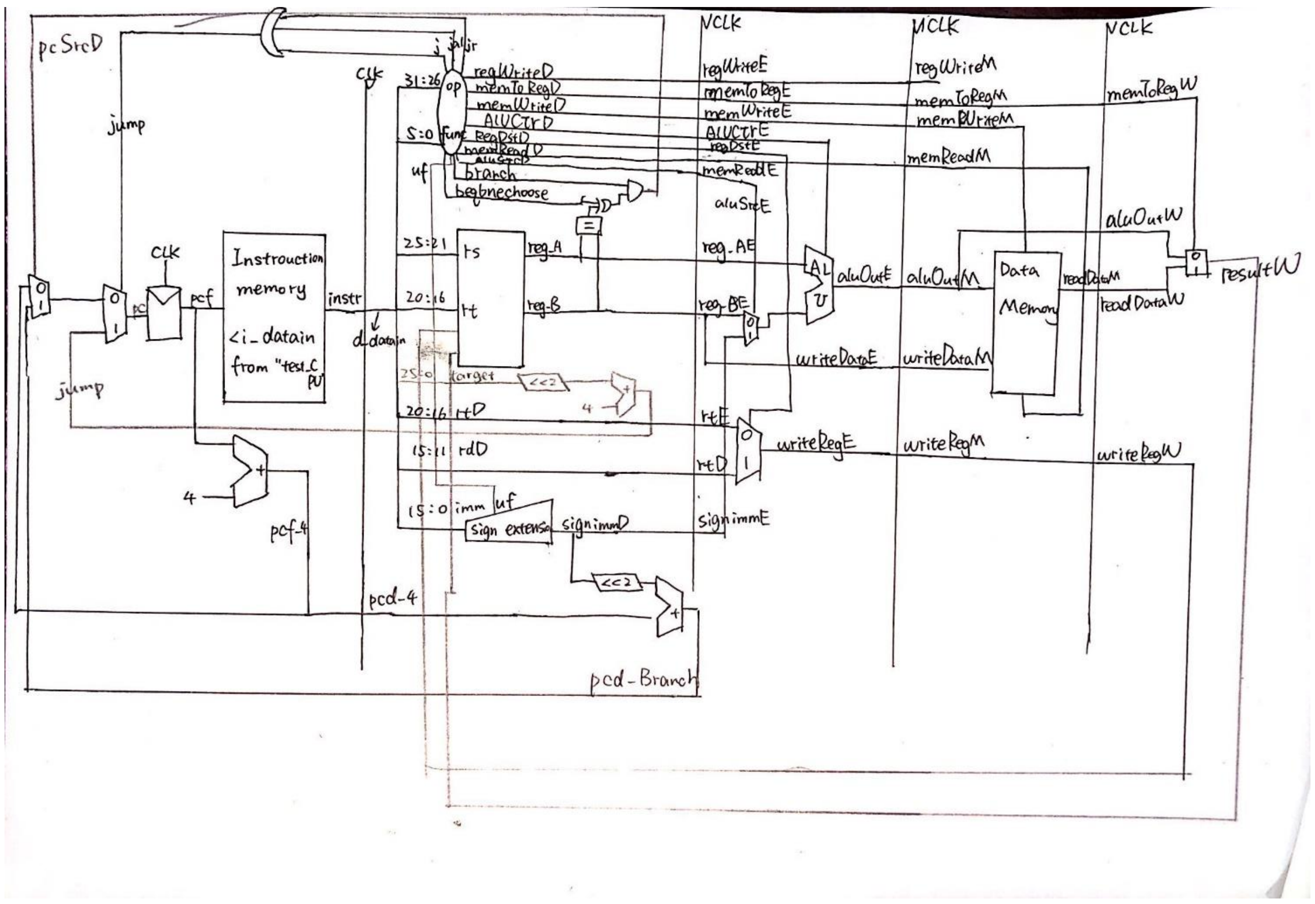
Project 4 Report_118010202_LIU Zhixuan

My block diagram:



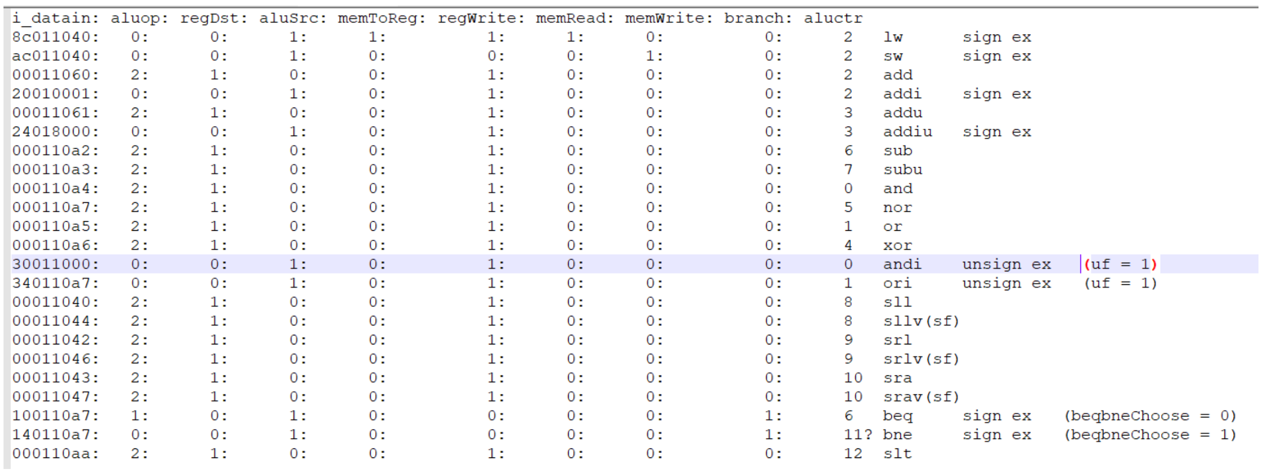
Some details

1) For Instruction Memory part, the test_CPU file will give the i_datain as the instruction

2) For Data Memory part, I allocate a fake memory part in CPU2.v file and I initial some DM as follows. (You can also change the initial things in DM if you want).

```
DM[1] = 32'h0000_00ab;
DM[2] = 32'h0000_3c00;
DM[3] = 32'h0000_0001;
DM[4] = 32'h8000_0000;
DM[5] = 32'h0000_0001;
```

2) all the control signals for each multiplexer are shown in the following paragraph



```
i_datain: aluop: regDst: aluSrc: memToReg: regWrite: memRead: memWrite: branch: aluctr
8c011040:   0:      0:      1:      1:          1:      1:      0:          0:      2   lw      sign ex
ac011040:   0:      0:      1:      0:          0:      0:      1:          0:      2   sw      sign ex
00011060:   2:      1:      0:      0:          1:      0:      0:          0:      2   add
20010001:   0:      0:      1:      0:          1:      0:      0:          0:      2   addi    sign ex
00011061:   2:      1:      0:      0:          1:      0:      0:          0:      3   addu
24018000:   0:      0:      1:      0:          1:      0:      0:          0:      3   addiu   sign ex
000110a2:   2:      1:      0:      0:          1:      0:      0:          0:      6   sub
000110a3:   2:      1:      0:      0:          1:      0:      0:          0:      7   subu
000110a4:   2:      1:      0:      0:          1:      0:      0:          0:      0   and
000110a7:   2:      1:      0:      0:          1:      0:      0:          0:      5   nor
000110a5:   2:      1:      0:      0:          1:      0:      0:          0:      1   or
000110a6:   2:      1:      0:      0:          1:      0:      0:          0:      4   xor
30011000:   0:      0:      1:      0:          1:      0:      0:          0:      0   andi    unsign ex   (uf = 1)
340110a7:   0:      0:      1:      0:          1:      0:      0:          0:      1   ori     unsign ex   (uf = 1)
00011040:   2:      1:      0:      0:          1:      0:      0:          0:      8   sll
00011044:   2:      1:      0:      0:          1:      0:      0:          0:      8   sllv(sf)
00011042:   2:      1:      0:      0:          1:      0:      0:          0:      9   srl
00011046:   2:      1:      0:      0:          1:      0:      0:          0:      9   srlv(sf)
00011043:   2:      1:      0:      0:          1:      0:      0:          0:      10  sra
00011047:   2:      1:      0:      0:          1:      0:      0:          0:      10  srav(sf)
100110a7:   1:      0:      1:      0:          0:      0:      0:          1:      6   beq     sign ex   (beqbneChoose = 0)
140110a7:   0:      0:      1:      0:          0:      0:      0:          1:      11? bne     sign ex   (beqbneChoose = 1)
000110aa:   2:      1:      0:      0:          1:      0:      0:          0:      12  slt
```

3) I designed a pipeline stage. More details can been observed directly in test_CPU.v.

4) For Hazards, I deal with branch hazard, as you can see in my figure, I redesigned the block diagram. More details can be seen in test_CPU5.v

5) For each instruction, it can be seen in the remaining test_CPU files

1. test_CPU.v

in this file, you can see a pipeline structure.

lw: load DM[1]: 32'h0000_00ab to gr[1]

lw: load DM[2]: 32'32'h0000_3c00 to gr[2]

lw: load DM[1]: 32'h0000_00ab to gr[1]

lw: load DM[2]: 32'32'h0000_3c00 to gr[2]

sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

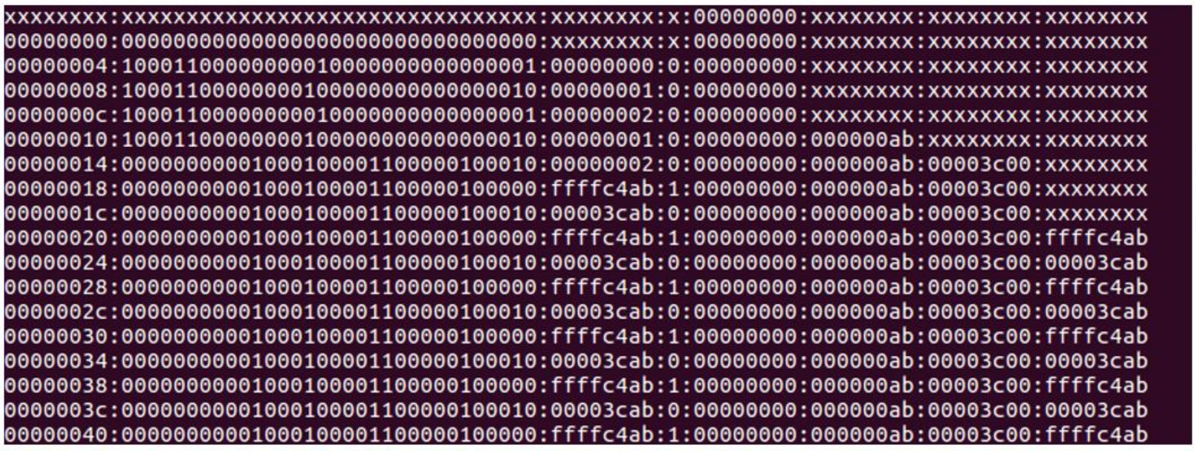
sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

sub: gr[1] - gr[2] = gr[3];

add: gr[1] + gr[2] = gr[3];

output:



```
 uut.pcf,      uut.d_datain,                         aluOutE, neg flag  gr[0]      gr[1]      gr[2]      gr[3]
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000000:00000000000000000000000000000000:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000004:10001100000000010000000000000001:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000008:10001100000000100000000000000010:00000001:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
0000000c:10001100000000010000000000000001:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000010:10001100000000100000000000000010:00000001:0:00000000:000000ab:xxxxxxxx:xxxxxxxx   lw
00000014:00000000001000100001100000100010:00000002:0:00000000:000000ab:00003c00:xxxxxxxx   lw
00000018:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:xxxxxxxx   lw
0000001c:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:xxxxxxxx   lw
00000020:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab   sub
00000024:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab   add
00000028:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab   sub
0000002c:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab   add
00000030:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab   sub
00000034:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab   add
00000038:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab   sub
0000003c:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab   add
00000040:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab   sub
```

2. test_CPU2.v

In this file, I test lw, sub, add, addu, subu, and, or, nor, xor, those instructions

lw: load DM[1]: 32'h0000_00ab to gr[1]

lw: load DM[2]: 32'32'h0000_3c00 to gr[2]

lw: load DM[1]: 32'h0000_00ab to gr[1]

lw: load DM[2]: 32'32'h0000_3c00 to gr[2]

sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

addu gr[1] + gr[2] = gr[3];

subu gr[1] - gr[2] = gr[3];

and gr[1] and gr[2]

or gr[1] or gr[2]

nor gr[1] nor gr[2]

xor gr[1] xor gr[2]

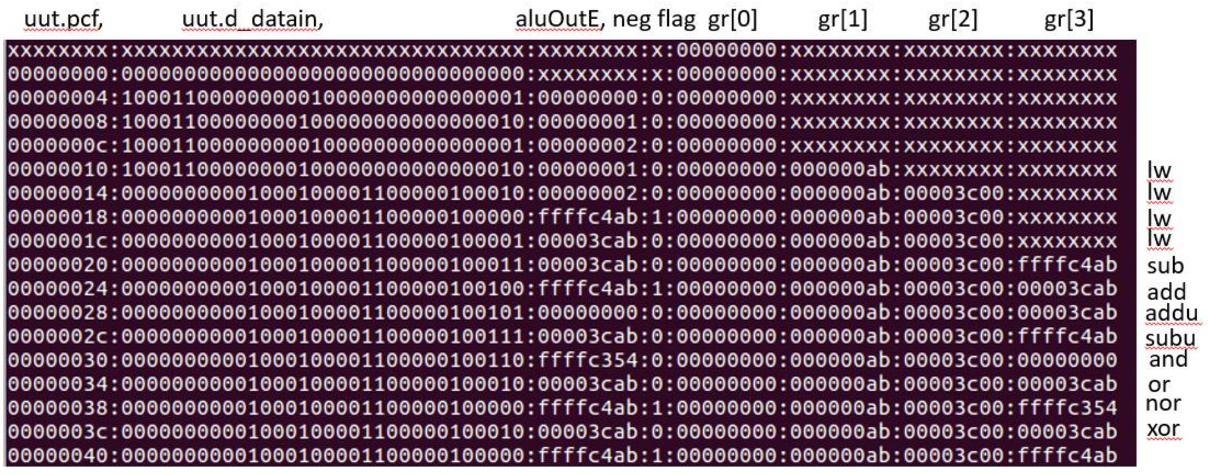
sub gr[1] - gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

sub gr[1] - gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

output sample



```
uut.pcf,   uut.d_datain,                   aluOutE, neg flag  gr[0]     gr[1]     gr[2]     gr[3]
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000000:00000000000000000000000000000000:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000004:10001100000000010000000000000001:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000008:10001100000000010000000000000010:00000001:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
0000000c:10001100000000010000000000000001:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000010:10001100000000010000000000000010:00000001:0:00000000:000000ab:xxxxxxxx:xxxxxxxx
00000014:00000000001000100001100000100010:00000002:0:00000000:000000ab:00003c00:xxxxxxxx
00000018:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:xxxxxxxx
0000001c:00000000001000100001100000100001:00003cab:0:00000000:000000ab:00003c00:xxxxxxxx
00000020:00000000001000100001100000100011:00003cab:0:00000000:000000ab:00003c00:ffffc4ab
00000024:00000000001000100001100000100100:ffffc4ab:1:00000000:000000ab:00003c00:00003cab
00000028:00000000001000100001100000100101:00000000:0:00000000:000000ab:00003c00:00003cab
0000002c:00000000001000100001100000100111:00003cab:0:00000000:000000ab:00003c00:ffffc4ab
00000030:00000000001000100001100000100110:ffffc354:0:00000000:000000ab:00003c00:00000000
00000034:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab
00000038:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc354
0000003c:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00003cab
00000040:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab
```

3. test_CPU3.v

In this test file, I test addi, addiu, andi, ori, sll, sllv, srl, srlv, sra, srav.

(PS: DM[1],[2],[3],[4] have all been initialed in CPU.v)

lw, load DM[1]: 32'h0000_00ab to gr[1]

lw, load DM[2]: 32'32'h0000_3c00 to gr[2]

lw, load DM[3]: 32'h0000_0001 to gr[4]

lw, load DM[4]: 32'h8000_0000 to gr[5]

addi gr[3] = gr[1] + h 2c00;

addiu gr[3] = gr[2] + h 00eb;

andi gr[1] andi imm

ori gr[1] ori imm

sll gr[3] = gr[2] << 3;

sllv gr[3] = gr[2] << gr[4];

srl gr[3] = gr[2] >> 3;

sllv gr[3] = gr[2] >> gr[4];

sra gr[3] = gr[5] >> 3;

srav gr[3] = gr[5] >> gr[4];

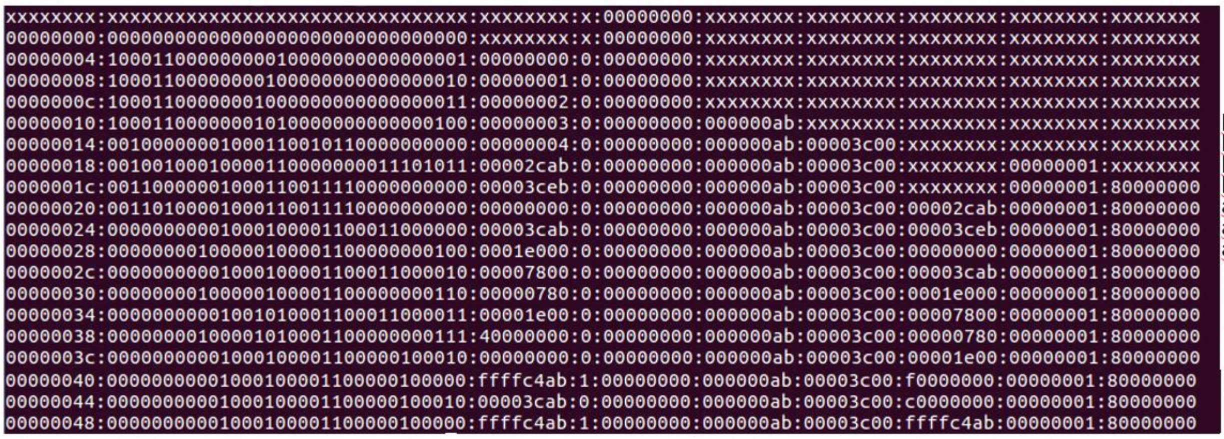
sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

output:



```
   uut.pcf,        uut.d_datain,                     aluOutE, neg flag  gr[0]    gr[1]      gr[2]      gr[3]      gr[4]     gr[5]
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000000:00000000000000000000000000000000:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000004:10001100000000010000000000000001:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000008:10001100000000100000000000000010:00000001:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
0000000c:10001100000000100000000000000011:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000010:10001100000001010000000000000100:00000003:0:00000000:000000ab:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx  lw
00000014:00100000001000110010110000000000:00000004:0:00000000:000000ab:00003c00:xxxxxxxx:xxxxxxxx:xxxxxxxx  lw
00000018:00100100010000110000000011101011:00002cab:0:00000000:000000ab:00003c00:xxxxxxxx:00000001:xxxxxxxx  lw
0000001c:00110000001000110011110000000000:00003ceb:0:00000000:000000ab:00003c00:xxxxxxxx:00000001:80000000  lw
00000020:00110100001000110011110000000000:00000000:0:00000000:000000ab:00003c00:00002cab:00000001:80000000  addi
00000024:00000000001000100001100011000000:00003cab:0:00000000:000000ab:00003c00:00003ceb:00000001:80000000  addiu
00000028:00000000100000100001100000000100:0001e000:0:00000000:000000ab:00003c00:00000000:00000001:80000000  andi
0000002c:00000000001000100001100011000010:00007800:0:00000000:000000ab:00003c00:00003cab:00000001:80000000   ori
00000030:00000000100000100001100000000110:00000780:0:00000000:000000ab:00003c00:0001e000:00000001:80000000   sll
00000034:00000000001001010001100011000011:00001e00:0:00000000:000000ab:00003c00:00007800:00000001:80000000   sllv
00000038:00000000100001010001100000000111:40000000:0:00000000:000000ab:00003c00:00000780:00000001:80000000   srl
0000003c:00000000001000100001100000100010:00000000:0:00000000:000000ab:00003c00:00001e00:00000001:80000000   srlv
00000040:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:f0000000:00000001:80000000   sra
00000044:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:c0000000:00000001:80000000   srav
00000048:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab:00000001:80000000   sub
```

4. test_CPU4.v

in this file I test sw.

lw, load DM[1]: 32'h0000_00ab to gr[1]

lw, load DM[2]: 32'32'h0000_3c00 to gr[2]

lw, load DM[3]: 32'h0000_0001 to gr[4]

lw, load DM[4]: 32'h8000_0000 to gr[5]

sw, store gr[0] to DM[4];

lw, load DM[4]: 32'h8000_0000 to gr[3];

sub gr[1] + gr[2] = gr[3];

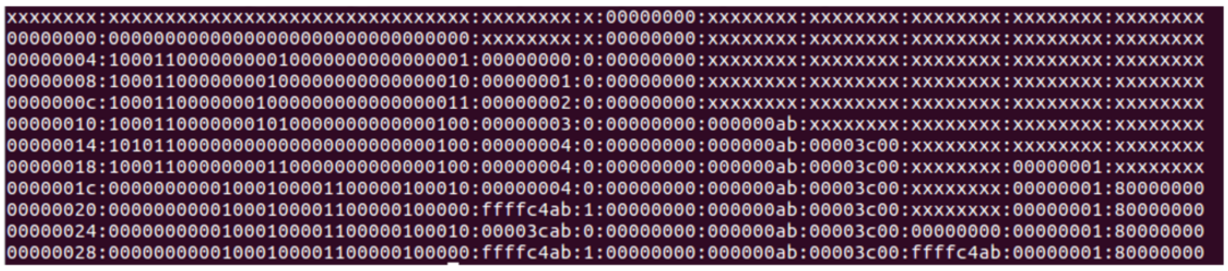
add gr[1] + gr[2] = gr[3];

sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

sample output

uut.pcf, uut.d_datain, aluOutE, neg flag gr[0] gr[1] gr[2] gr[3] gr[4] gr[5]

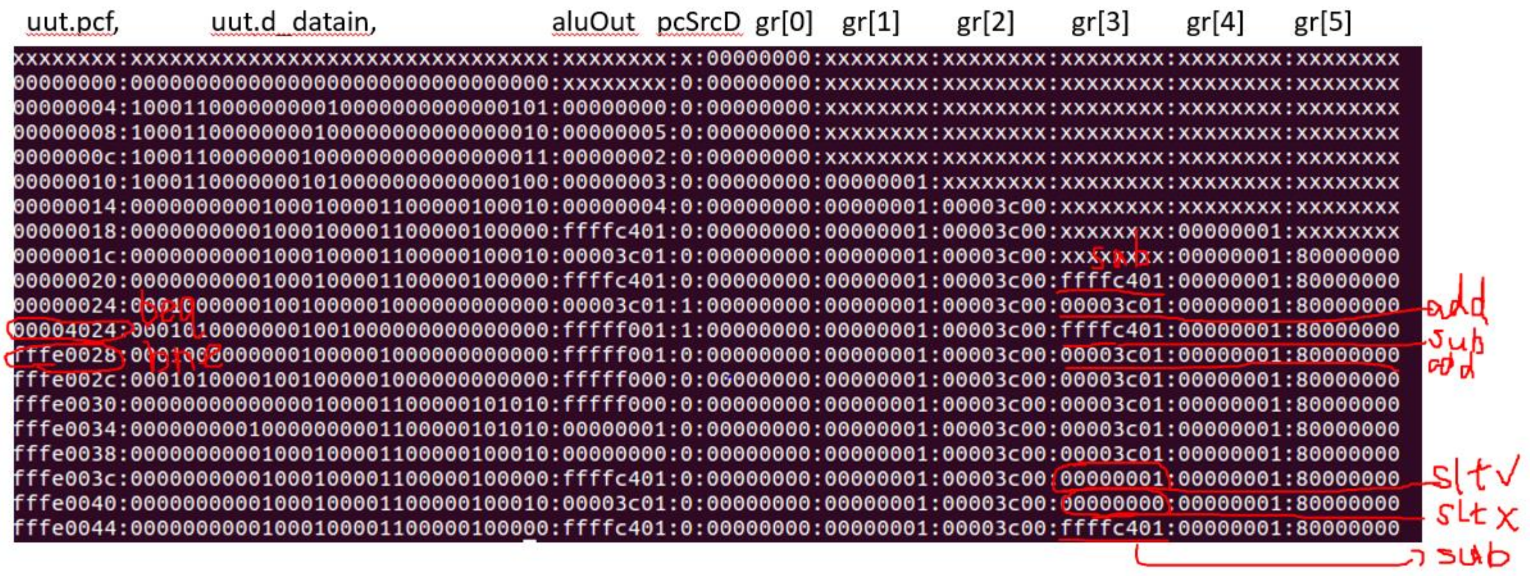


```
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000000:00000000000000000000000000000000:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000004:10001100000000010000000000000001:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000008:10001100000000100000000000000010:00000001:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
0000000c:10001100000001000000000000000011:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000010:10001100000001010000000000000100:00000003:0:00000000:000000ab:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx  lw
00000014:10101100000000000000000000000100:00000004:0:00000000:000000ab:00003c00:xxxxxxxx:xxxxxxxx:xxxxxxxx  lw
00000018:10001100000000110000000000000100:00000004:0:00000000:000000ab:00003c00:xxxxxxxx:00000001:xxxxxxxx  lw
0000001c:00000000001000100001100000100010:00000004:0:00000000:000000ab:00003c00:xxxxxxxx:00000001:80000000  lw
00000020:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:xxxxxxxx:00000001:80000000  sw
00000024:00000000001000100001100000100010:00003cab:0:00000000:000000ab:00003c00:00000000:00000001:80000000  lw
00000028:00000000001000100001100000100000:ffffc4ab:1:00000000:000000ab:00003c00:ffffc4ab:00000001:80000000  sub
```

5. test_CPU5.v: dealing with **branch hazard**

in this file, I test beq, bne ,slt, and I deal with branch hazards.
lw, load DM[5]: 32'h0000_0001 to gr[1]   => gr[1] = 1;
lw, load DM[2]: 32'32'h0000_3c00 to gr[2]
w, load DM[3]: 32'h0000_0001 to gr[4]   => gr[4] = 1;
lw, load DM[4]: 32'h8000_0000 to gr[5]
sub gr[1] + gr[2] = gr[3];
add gr[1] + gr[2] = gr[3];
sub gr[1] + gr[2] = gr[3];
add gr[1] + gr[2] = gr[3];
beq, gr[1] = gr[4], branch;
bne, gr[0] != gr[4], branch;
beq, gr[0] != gr[4], not branch;
bne, gr[1] = gr[4], not branch;
slt gr[0] < gr[2]   => gr[3] = 1;
slt gr[0] > gr[2]   => gr[3] = 0;
sub gr[1] + gr[2] = gr[3];
add gr[1] + gr[2] = gr[3];
sub gr[1] + gr[2] = gr[3];
add gr[1] + gr[2] = gr[3];

sample outpu:


```
 uut.pcf,        uut.d_datain,                   aluOut  pcSrcD gr[0]   gr[1]    gr[2]    gr[3]    gr[4]    gr[5]
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000000:00000000000000000000000000000000:xxxxxxxx:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000004:10001100000000010000000000000101:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000008:10001100000000100000000000000010:00000005:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
0000000c:10001100000001000000000000000011:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000010:10001100000001010000000000000100:00000003:0:00000000:00000001:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000014:00000000001000100001100000100010:00000004:0:00000000:00000001:00003c00:xxxxxxxx:xxxxxxxx:xxxxxxxx
00000018:00000000001000100001100000100000:ffffc401:0:00000000:00000001:00003c00:xxxxxxxx:00000001:xxxxxxxx
0000001c:00000000001000100001100000100010:00003c01:0:00000000:00000001:00003c00:xxxxxxxx:00000001:80000000
00000020:00000000001000100001100000100000:ffffc401:0:00000000:00000001:00003c00:ffffc401:00000001:80000000
00000024:00010000001001000001000000000000:00003c01:1:00000000:00000001:00003c00:00003c01:00000001:80000000
00004024:00010100000001001000000000000000:fffff001:1:00000000:00000001:00003c00:ffffc401:00000001:80000000
fffe0028:00010000000001000001000000000000:fffff001:0:00000000:00000001:00003c00:00003c01:00000001:80000000
fffe002c:00010100001001000001000000000000:fffff000:0:00000000:00000001:00003c00:00003c01:00000001:80000000
fffe0030:00000000000000010000110000101010:fffff000:0:00000000:00000001:00003c00:00003c01:00000001:80000000
fffe0034:00000000010000000001100000101010:00000001:0:00000000:00000001:00003c00:00003c01:00000001:80000000
fffe0038:00000000001000100001100000100010:00000000:0:00000000:00000001:00003c00:00003c01:00000001:80000000
fffe003c:00000000001000100001100000100000:ffffc401:0:00000000:00000001:00003c00:00000001:00000001:80000000
fffe0040:00000000001000100001100000100010:00003c01:0:00000000:00000001:00003c00:00000000:00000001:80000000
fffe0044:00000000001000100001100000100000:ffffc401:0:00000000:00000001:00003c00:ffffc401:00000001:80000000
```

6. test_CPU6.v

=> this test file is designed for j, jal, jr

lw, load DM[5]: 32'h0000_0001 to gr[1] => gr[1] = 1;

lw, load DM[2]: 32'32'h0000_3c00 to gr[2]

lw, load DM[3]: 32'h0000_0001 to gr[4] => gr[4] = 1;

lw, load DM[4]: 32'h8000_0000 to gr[5]

j: jump

jal: jump

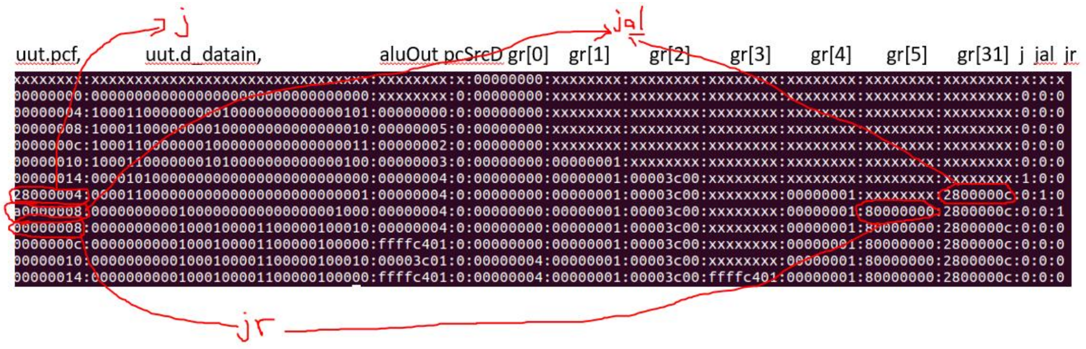
jr: jump

sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];

sub gr[1] + gr[2] = gr[3];

add gr[1] + gr[2] = gr[3];



```
uut.pcf,  uut.d_datain,                    aluOut pcSrcD gr[0]   gr[1]    gr[2]    gr[3]    gr[4]    gr[5]    gr[31]  j jal jr
xxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxx:x:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:x:x:x
00000000:00000000000000000000000000000000:xxxxxxxx:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:0:0:0
00000004:10001100000000010000000000000101:00000000:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:0:0:0
00000008:10001100000000010000000000000010:00000005:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:0:0:0
0000000c:10001100000000010000000000000011:00000002:0:00000000:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:0:0:0
00000010:10001100000000101000000000000100:00000003:0:00000000:00000001:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:0:0:0
00000014:00001010000000000000000000000000:00000004:0:00000000:00000001:00003c00:xxxxxxxx:xxxxxxxx:xxxxxxxx:xxxxxxxx:1:0:0
28000004:00001100000000000000000000000001:00000004:0:00000000:00000001:00003c00:xxxxxxxx:00000001:xxxxxxxx:2800000c:0:1:0
a0000008:00000000001000000000000000001000:00000004:0:00000000:00000001:00003c00:xxxxxxxx:00000001:80000000:2800000c:0:0:1
00000008:00000000001000100001100000100010:00000004:0:00000000:00000001:00003c00:xxxxxxxx:00000001:80000000:2800000c:0:0:0
0000000c:00000000001000100001100000100000:ffffc401:0:00000000:00000001:00003c00:xxxxxxxx:00000001:80000000:2800000c:0:0:0
00000010:00000000001000100001100000100010:00003c01:0:00000004:00000001:00003c00:xxxxxxxx:00000001:80000000:2800000c:0:0:0
00000014:00000000001000100001100000100000:ffffc401:0:00000004:00000001:00003c00:ffffc401:00000001:80000000:2800000c:0:0:0
```