ESE 545 | Project

MICRO- ARCHITECTURE OF SYNERGISTIC PROCESSING UNIT OF SONY CELL

SUBMITTED BY LAHARI YALAMANDALA (115133017) & GEORGE MADATHIL SAVIOUR (114834447)

Table of Content:

1) SPU-Cell Model:

The SPU processor core consists of two processing pipes, register file, and data forwarding circuits which is represented the Architecture block diagram. Each of the core units are explained below: -

Fetch: This units fetches instruction from the Instruction memory and passes it to the decode stage. When any hazards are detected by the decode unit, this unit stalls the PC (Program Counter). Thereby no new instructions will be fetched. If a branch is taken, this unit starts reading Instructions from the new Program Counter.

Decode: Decode unit performs varying op code length instruction decoding and also detects data and structural hazards and performs stall operations to overcome these hazards.

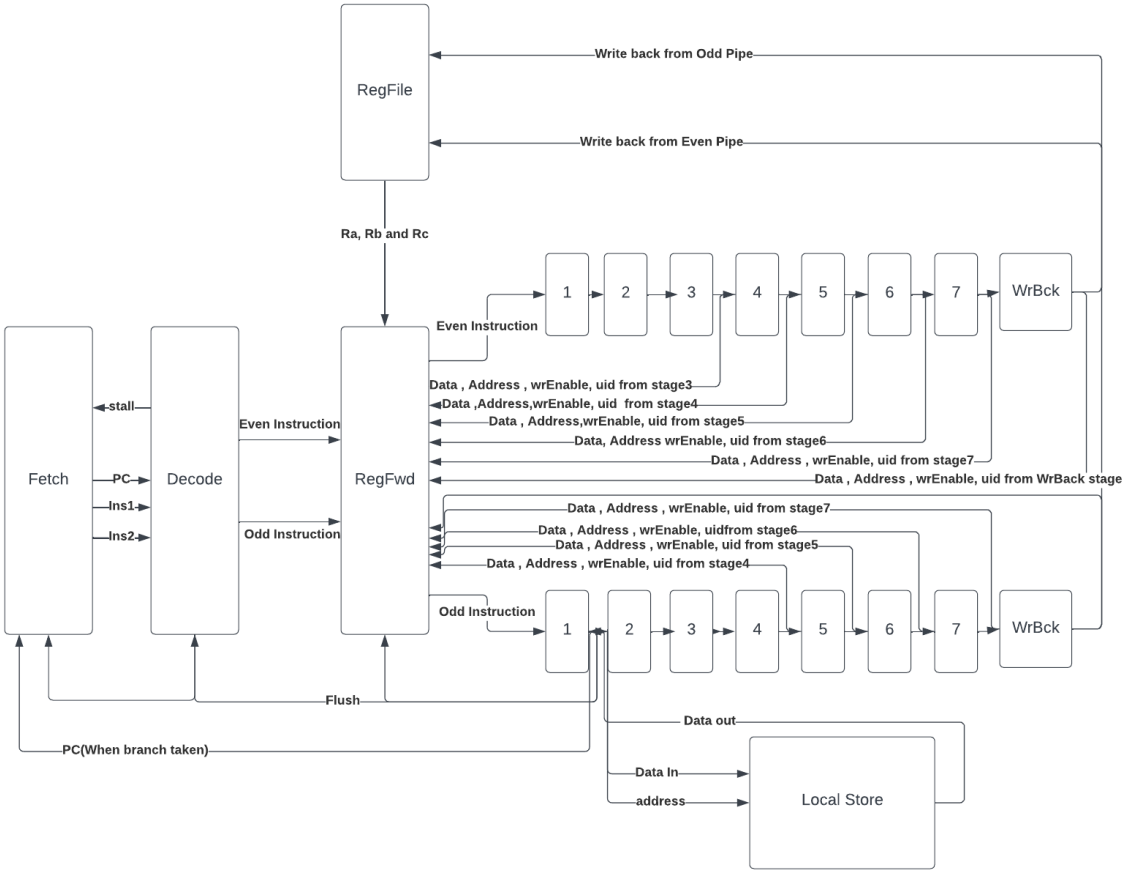
Register File: The register file consists of 6 Read Ports and 2 Write Ports and has 128 entries with 128 bits each. The Register file has a write latency of 1 clock cycle. The register file data is sent to the even/odd pipes through the register forward unit for execution. Once the execution is completed, the results from the pipes are written back to the destination address.

Register Forward Unit: Register forward unit directs data from Decode Unit to the execution pipe lines. It forwards the operand data for each instruction either from the Register file or from the any stages of the execution pipes.

Even Pipe: This unit performs all the even type instructions as mentioned in the ISA table . In total, this pipe has 8 stages which includes 7 stages and a write back stage. Even pipe has 5 functional units Simple Fixed -1, Simple Fixed -2, Single Precision 1, Single Precision 2 and Byte. The latency and Unit id can be found in ISA table.
Odd Pipe: This unit performs all the odd type instructions as mentioned in the ISA table . In total, this SDasdasdPermute and Local Store.The latency and Unit id can be found in ISA table.

Local Store: Local Store of size 32KB is used to store data memory.

Model Diagram:



On top of the above signals, Decode unit receives address, uid and write enable from all the stages in the even and odd pipe lines.

2) Summary Table:

SPU-lite Model Code	Parser code	Loading of the instruction memory with instruction code from the file created by the parser	Dual-instruction fetch_decode_RF read_ execute (7 stages)_WB (no hazards)	Structural hazard resolution	Data hazard resolution by forwarding (no stall)	Data hazard resolution by stalling & forwarding	Control hazard resolution for branches	4x4 SP FP matrix multiply using multiply-add instructions
Code: Link	Code: Link	Code:Link	Code: Link Test: Link	Code: Link Test: Link	Code: Link Test :Link	Code: Link Test:Link	Code: Link Test:Link	Test:Link

3) Parser:

Parser changes the assembly code into binary representation of each instructions.

Parser implementation can be found here at Parser.cc

4) ISA table:

Each instruction having varying op code length as seen in SPU cell ISA is given a fixed 11 bits op code length in the decode unit. The opcode assigned by decode united can be seen in the definitions.sv.

5) EXECUTION PIPE	UNIT	INSTRUCTIONS	UNIT PIPELINE DEPTH	INSTRUCTION LATENCY
Even	SIMPLE FIXED1(FX1) (1)	Add Halfword, Add Halfword Immediate, Add Word, Add Word Immediate, Subtract from Halfword, Subtract from Word,Count Leading Zeros, Form Select Mask for Bytes, Form Select Mask for Halfwords,Form Select Mask for Words, And, And Word Immediate, And Halfword Immediate, Or, Or Word Immediate, Or Halfword Immediate, Nor, Exclusive Or, Exclusive Or Halfword Immediate, Exclusive Or Word Immediate, Equivalent, form select mask bytes immediate, or_byte_immediate, xor bytes immediate, Compare equal halfword, Compare equal halfword	2	3

		immediate, Compare equal word, Compare equal word immediate, Compare greater than halfword, Compare greater than halfword immediate, Compare greater than word, Compare greater than word immediate		
Even	SIMPLE FIXED2(FX2) (2)	shift left halfword, shift left halfword immediate, shift left word, shift left word immediate, Rotate Halfword, Rotate Halfword Immediate, rotate word, rotate word immediate,rotate and mask halfword, rotate and mask halfword immediate,	3	4
Even	SINGLE PRECISION1 (3)	Floating Multiply, Floating Multiply and Add, Floating Multiply and Subtract, Floating Add, Floating subtract	6	7
Even	SINGLE PRECISION2 (4)	Multiply and Add, , Multiply, Multiply Immediate, Multiply Unsigned	7	8
Even	BYTE (5)	Count ones in bytes, average bytes, Absolute differences in bytes, Sum bytes into halfwords	3	4
Odd	PERMUTE (6)	Gather Bits from Halfwords, Gather Bits from Words, Shuffle Byte, Shift Left Quadword by bits, Shift left quadword by bits immediate ,Shift left quadword by bytes, Shift left quadword by bytes immediate,Rotate quadword by bytes, Rotate quadword by bytes immediate	3	4
Odd	LOAD AND STORE (7)	Load quadword(d-form), Load quadword(a-form), Store Quadword (d-form), Store Quadword(a-form), Immediate Load Halfword, Immediate Load word, Immediate Load Address,	6	7
Odd	BRANCHES (8)	Branch relative, Branch absolute, branch if not zero word,branch if zero word,Branch relative, Branch absolute,branch if not zero word,branch if zero word,	1	1

		Branch if not zero halfword,Branch if zero halfword		

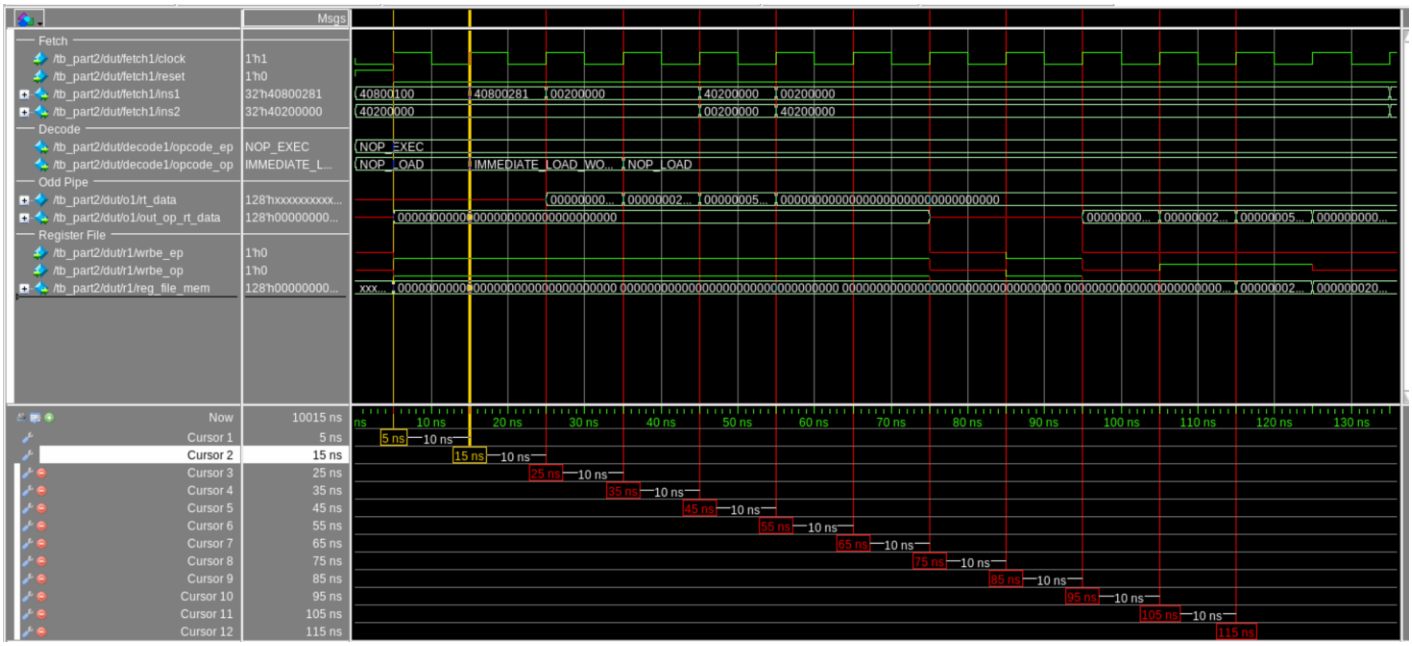
6) Verification Results:

1) Dual instruction fetch_decode_RF read_ execute (7 stages) _WB (no hazards)

Test Code:

```
il 0,2
nop
il 1,5
nop
lnop
nop
lnop
nop
nop
lnop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
```

Waveform:



Explanation:

15ns: Decode stage receives Instruction 1 and Instruction 2

25ns: RF stage receives Instruction 1 and Instruction for even and odd pipe
35ns: Odd pipe receives instruction.
105ns: Odd pipe output instruction result.
115ns: Result written to Register file.

2) Structural Hazard:

a. Case1: Instruction two has raw and structural hazard and instruction 1 has no hazards.

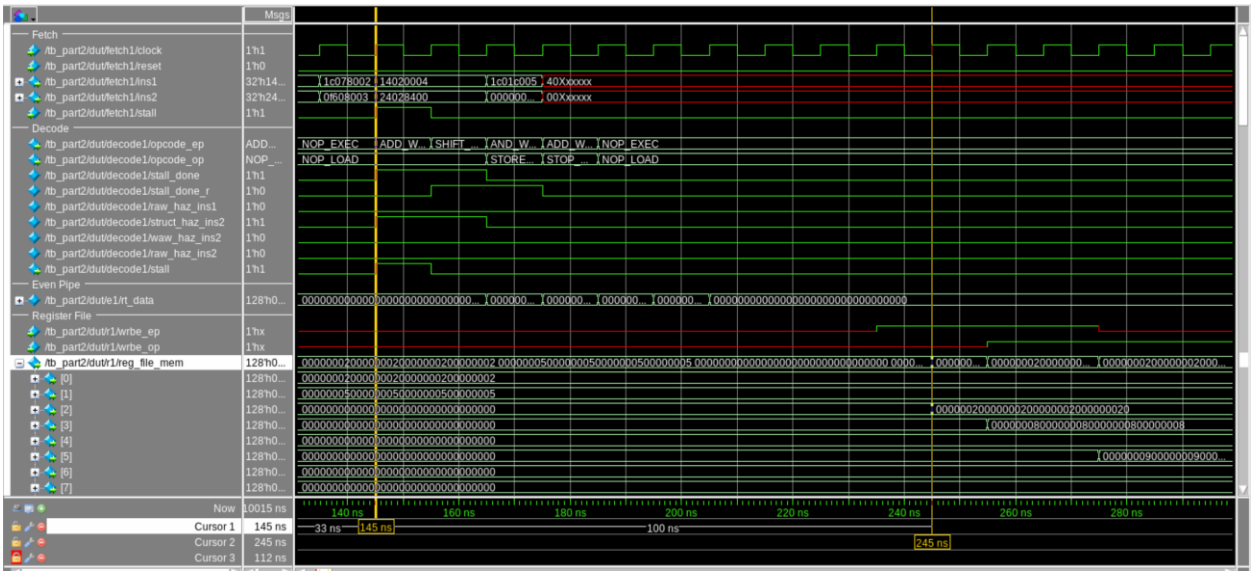
Test code:

```
1  il 0,2
2  nop
3  il 1,5
4  nop
5  lnop
6  nop
7  lnop
8  nop
9  nop
10 lnop
11 lnop
12 nop
13 lnop
14 nop
15 lnop
16 nop
17 lnop
18 nop
19 lnop
20 nop
21 lnop
22 nop
23 lnop
24 nop
25 lnop
26 nop
27 ai 2,0,30
28 shli 3,0,2
29 andi 4,0,8
30 stqd 0,10(0)
31 ai 5,0,7
32 stop
```

Line 27 and Line 28 Same pipe.

Explanation:

Add word and shift left half word immediate are given as instruction 1 and instruction2. At 145ns, structural hazard is detected for instruction 2 in the decode stage. **Stall_done_r** is used to indicate that instruct 2 was stall and now needs to be passed for execution since instruction 1 was passed in the previous cycle. At 245ns, result for instruction 1 is written to the register file at address 2. At 255ns instruction 2 result is written to the register file at address 3.



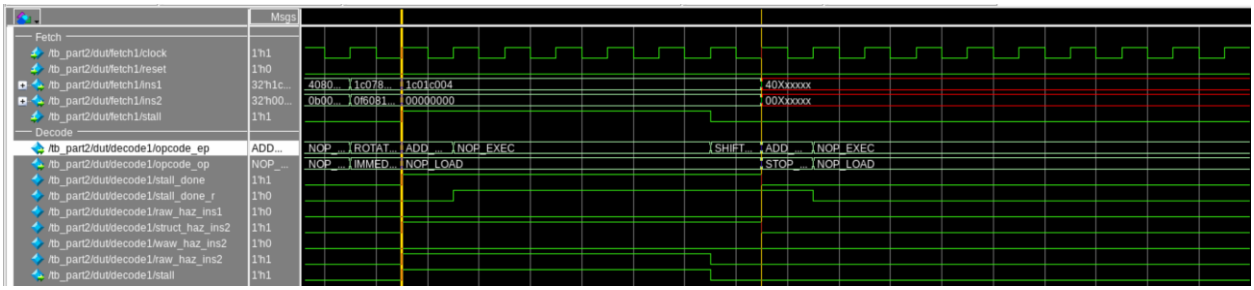
b. Case2: Instruction2 has raw and structural hazard and Instruction 1 no hazard.

Test code:

```
il 0,2
nop
il 1,5
nop
lnop
nop
lnop
nop
nop
lnop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
il 2,2
rot 5,1,0
ai 3,0,30
shli 4,2,2
ai 4,0,7
stop
```

Line 30, shli having raw and struct hazard.

Waveform:

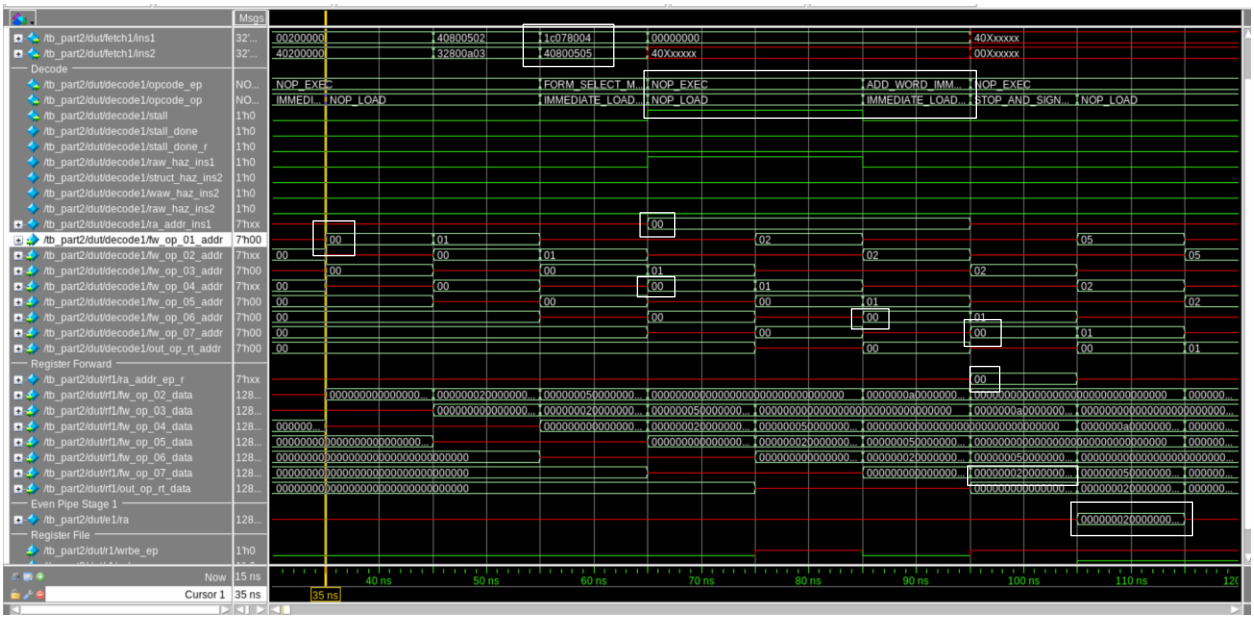


3) Data hazard resolution by stalling & forwarding:

Raw hazard for instruction 1 and no hazard for instruction 2.

Test code:

```
il 0,2
nop
il 1,5
nop
lnop
nop
lnop
nop
il 2,10
fsmbi 3,20
ai 4,0,30
il 5,10
stop
```



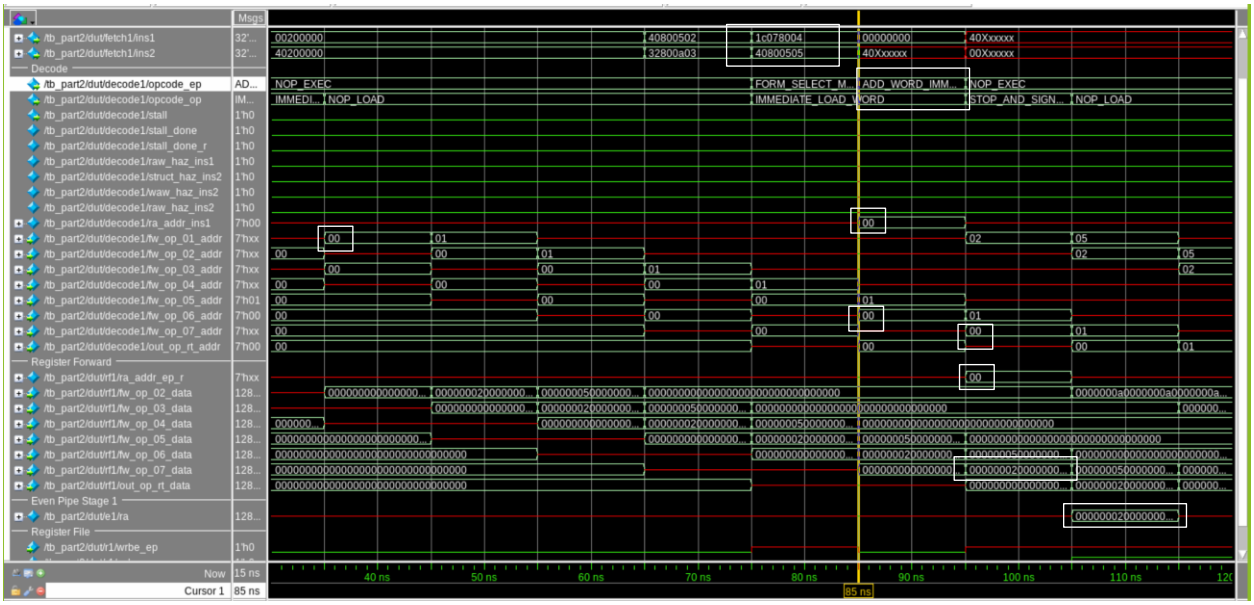
4) Data hazard resolution by forwarding (no stall):

Raw hazard for instruction 1 and no hazard for instruction 2.

Testcode:

```
il 0,2
nop
il 1,5
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
il 2,10
fsmbi 3,20
ai 4,0,30
il 5,10
stop
```

Waveform:

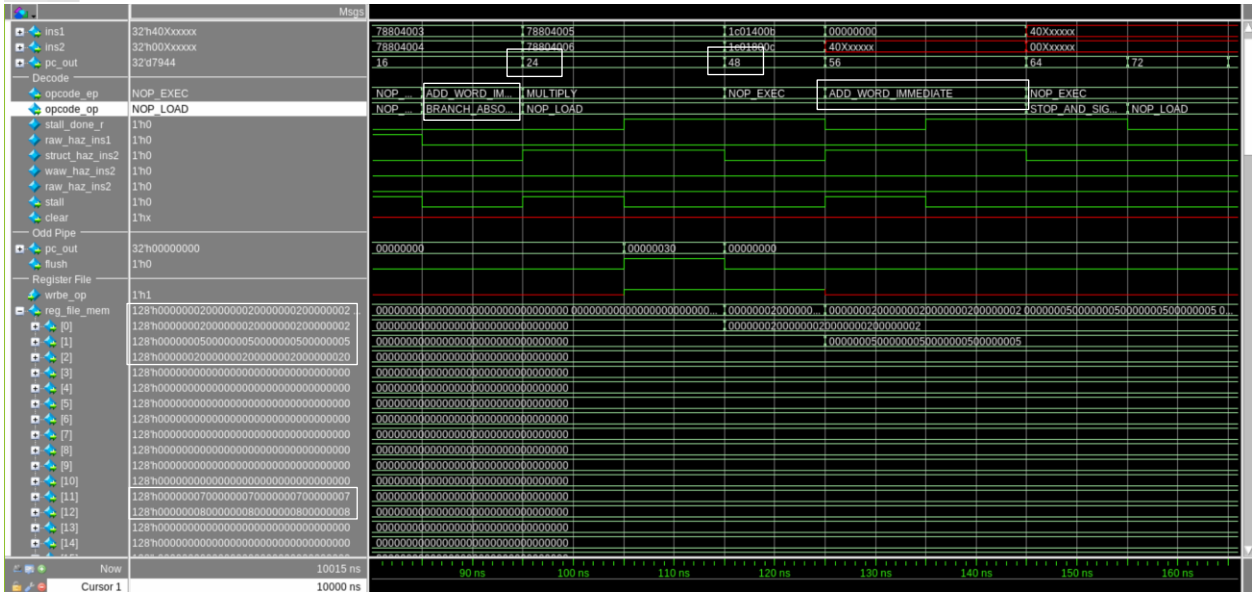


5) Control hazard resolution for branches:

a. Case1: If branch instruction is the second instruction and pc address points to even instruction

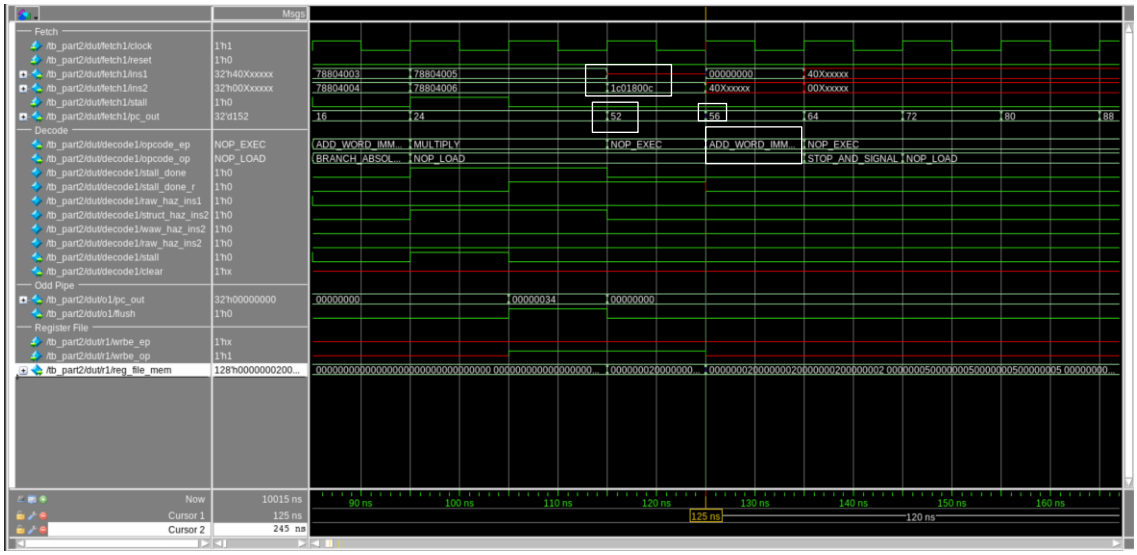
Test code:

```
1   il 0,2
2   il 1,5
3   ai 2,0,30
4   bra 12
5   mpy 3,0,1
6   mpy 4,0,1
7   mpy 5,0,1
8   mpy 6,0,1
9   mpy 7,0,1
10  mpy 8,0,1
11  mpy 9,0,1
12  mpy 10,0,1
13  ai  11,0,5
14  ai  12,0,6
15  stop
```



b. Case 2: If branch is second instruction and pc address points to odd instruction

```
1   il 0,2
2   il 1,5
3   ai 2,0,30
4   bra 13
5   mpy 3,0,1
6   mpy 4,0,1
7   mpy 5,0,1
8   mpy 6,0,1
9   mpy 7,0,1
10  mpy 8,0,1
11  mpy 9,0,1
12  mpy 10,0,1
13  ai 11,0,5
14  ai 12,0,6
15  stop
```

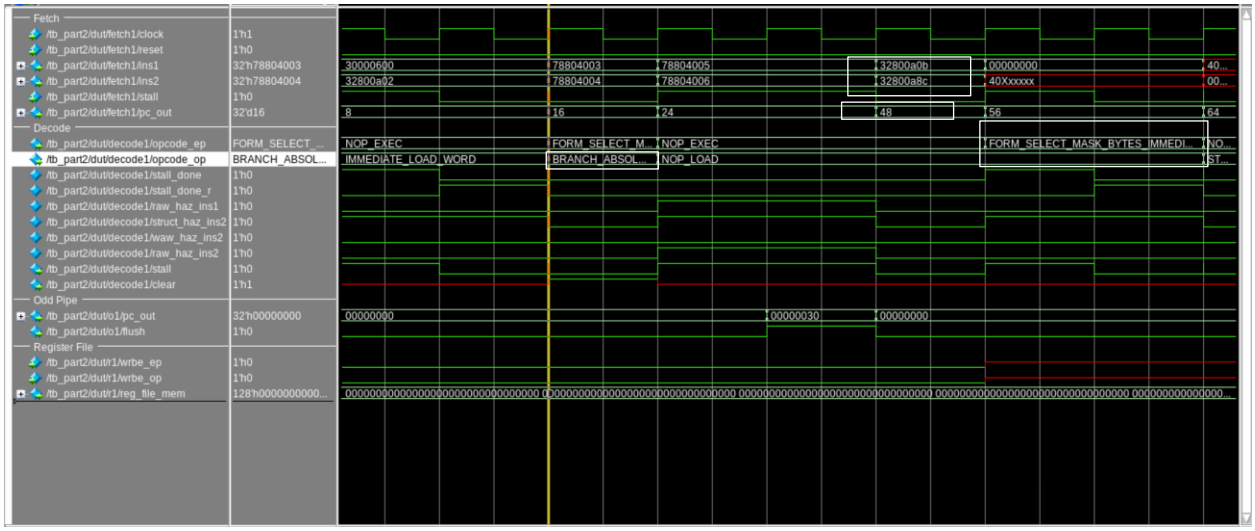


c. Case 3: Branch instruction is first instruction and pc address points to even instruction.

Test Code:

```
1   il 0,2
2   il 1,5
3   bra 12
4   fsmbi 2,20
5   mpy 3,0,1
6   mpy 4,0,1
7   mpy 5,0,1
8   mpy 6,0,1
9   mpy 7,0,1
10  mpy 8,0,1
11  mpy 9,0,1
12  mpy 10,0,1
13  fsmbi 11,20
14  fsmbi 12,21
15  stop
```

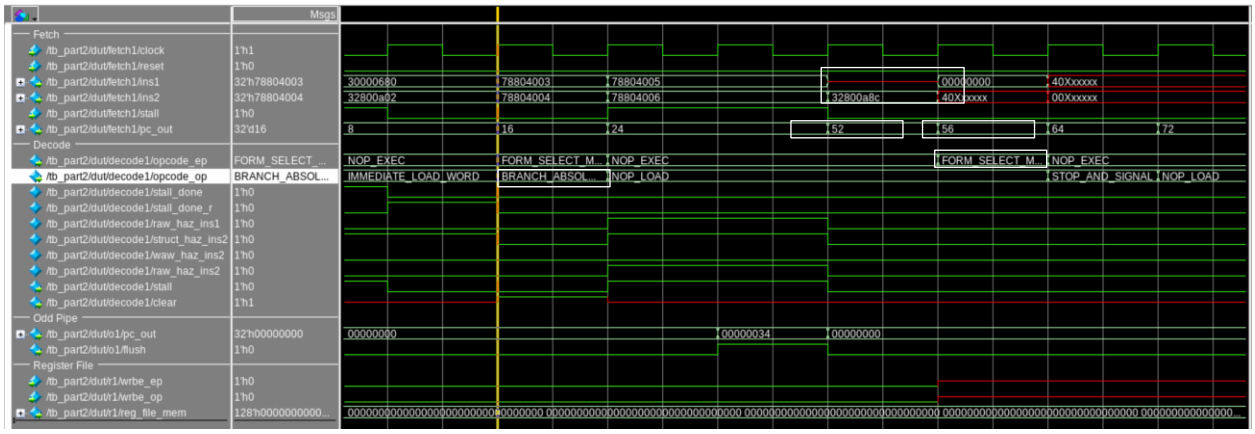
Waveform:



d. Case4: If branch is first instruction and pc address points to odd instruction

Code:

```
1   il 0,2
2   il 1,5
3   bra 13
4   fsmbi 2,20
5   mpy 3,0,1
6   mpy 4,0,1
7   mpy 5,0,1
8   mpy 6,0,1
9   mpy 7,0,1
10  mpy 8,0,1
11  mpy 9,0,1
12  mpy 10,0,1
13  fsmbi 11,20
14  fsmbi 12,21
15  stop
```

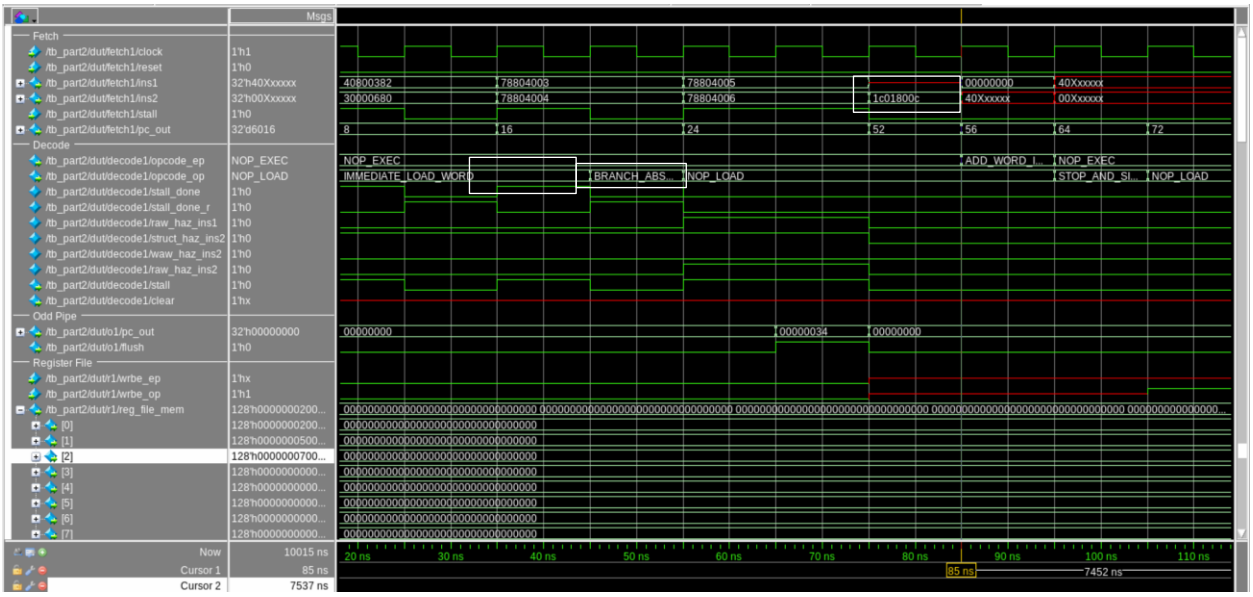


e. Case 5 Control hazard, branch absolute (inst1 is branch, ins2 is odd pipe)

Test code:

```
1   il 0,2
2   il 1,5
3   il 2,7
4   bra 13
5   mpy 3,0,1
6   mpy 4,0,1
7   mpy 5,0,1
8   mpy 6,0,1
9   mpy 7,0,1
10  mpy 8,0,1
11  mpy 9,0,1
12  mpy 10,0,1
13  ai 11,0,5
14  ai 12,0,6
15  stop
```

Waveform:

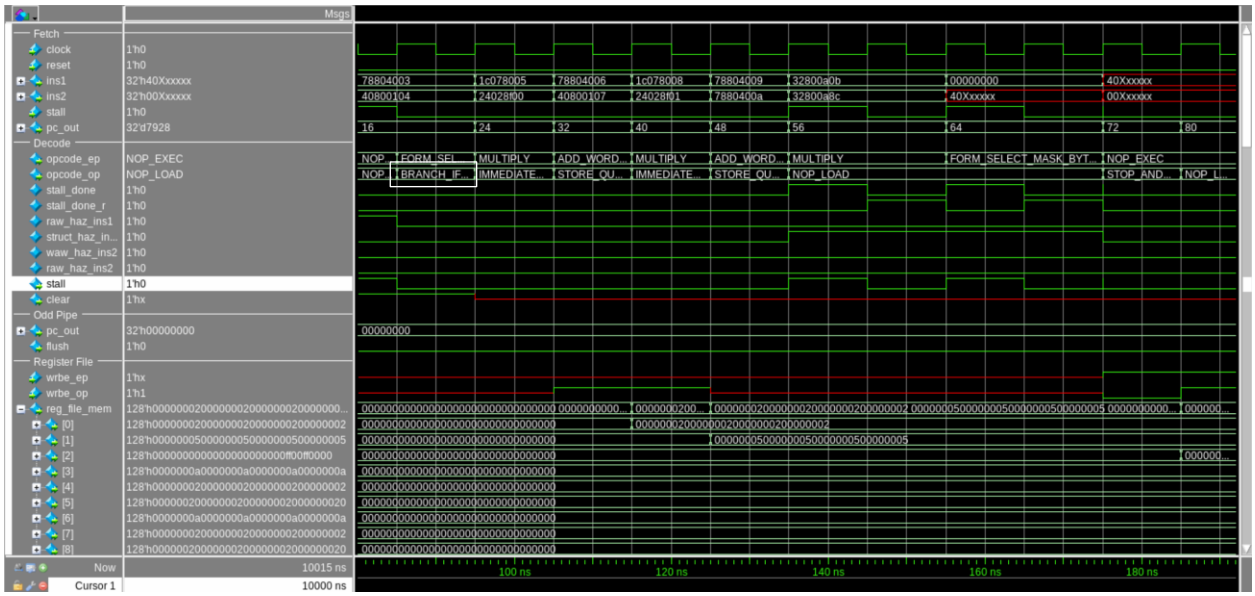


f. Case 6:Control hazard, branch if not zero(inst2 is branch) and PC is even - not taken

Test Code:

```
1   il 0,2
2   il 1,5
3   brz 0,12
4   fsmbi 2,20
5   mpy 3,0,1
6   il 4,2
7   ai 5,0,30
8   stqd 0,10(0)
9   mpy 6,0,1
10  il 7,2
11  ai 8,0,30
12  stqd 1,10(0)
13  mpy 9,0,1
14  mpy 10,0,1
15  fsmbi 11,20
16  fsmbi 12,21
17  stop
```

Waveform:



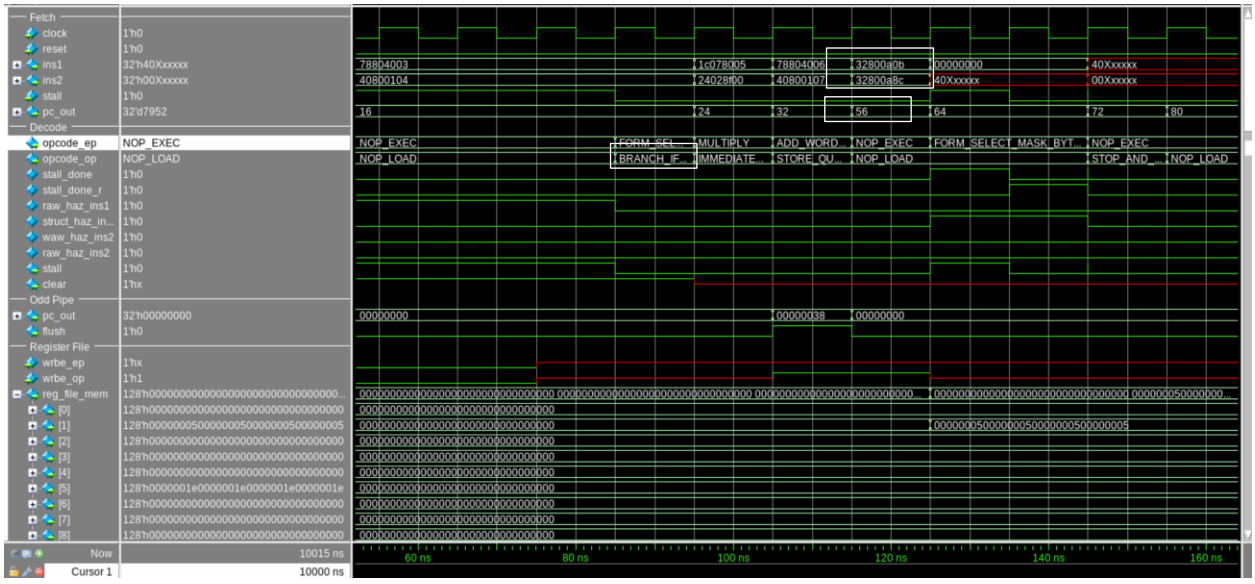
g. Case 7: Control hazard, branch if not zero(inst2 is branch) and PC is even - taken

Test Code:

```
1   il 0,0
2   il 1,5
3   brz 0,12
4   fsmbi 2,20
5   mpy 3,0,1
6   il 4,2
7   ai 5,0,30
8   stqd 0,10(0)
9   mpy 6,0,1
10  il 7,2
11  ai 8,0,30
12  stqd 1,10(0)
13  mpy 9,0,1
14  mpy 10,0,1
15  fsmbi 11,20
16  fsmbi 12,21
17  stop
```

In the above code, control of the code jumps from PC =8 to PC = 56.

Waveform:



6) SP FP 4X4 matrix multiplication:

Test Code:

```
il 9,256
il 18, 512
il 10,1
il 13,4
il 15,16
fsm 11,10
il 0,4
sf 0,10,0
il 1,4
sf 1,10,1
il 2,0
il 3,4
sf 3,10,3
mpya 12,0,13,3
mpy 14,12,15
mpya 16,3,13,1
mpy 17,16,15
a 21,17,9
lqd 6,0(14)
lqd 7,0(21)
fma 2,6,7,2
brnz 3,-8
mpya 19,0,13,1
mpy 20,19,15
a 22,20,18
stqd 2,0(22)
brnz 1,-17
brnz 0,-19
nop
lnop
nop
lnop
nop
lnop
nop
lnop
nop
stop
```

Waveforms:

Matrix A is preloaded from 0-255 in Local store. Matrix B from 256-511. The result of A*B is stored from 512-767 locations in local store.

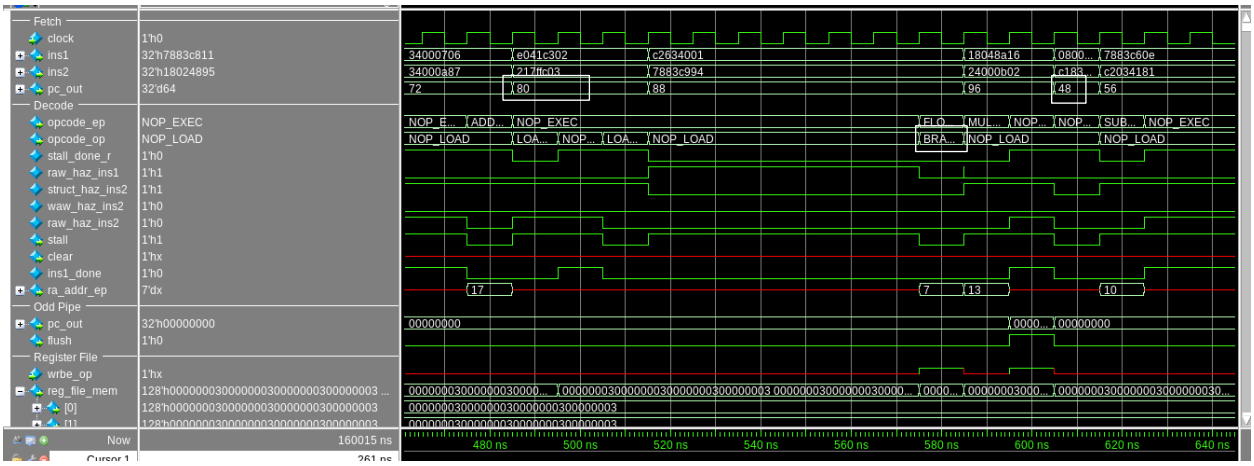
$$A = \begin{matrix} 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \end{matrix} \quad B = \begin{matrix} 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 \end{matrix} \quad C = A*B = \begin{matrix} 10.0 & 20.0 & 30.0 & 40.0 \\ 10.0 & 20.0 & 30.0 & 40.0 \\ 10.0 & 20.0 & 30.0 & 40.0 \\ 10.0 & 20.0 & 30.0 & 40.0 \end{matrix}$$

The hexadecimal representation for the floating-point value that is stored in the local store for C is

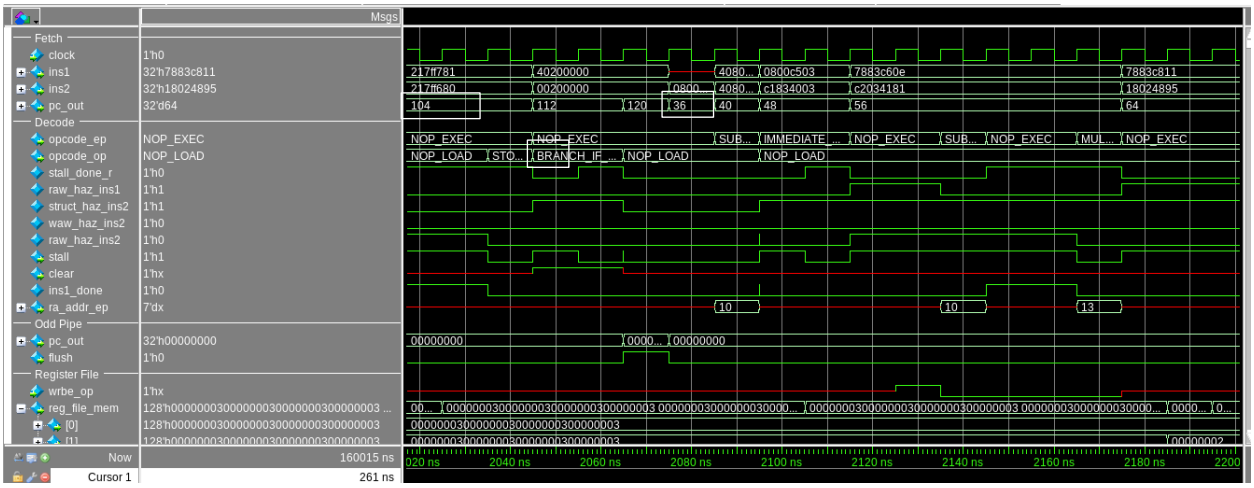
$$C = A*B = \begin{matrix} 0x41200000 & 0x41a00000 & 0x41f00000 & 0x42200000 \\ 0x41200000 & 0x41a00000 & 0x41f00000 & 0x42200000 \\ 0x41200000 & 0x41a00000 & 0x41f00000 & 0x42200000 \\ 0x41200000 & 0x41a00000 & 0x41f00000 & 0x42200000 \end{matrix}$$

3 nested for-loops are implemented using 3 branch instructions.

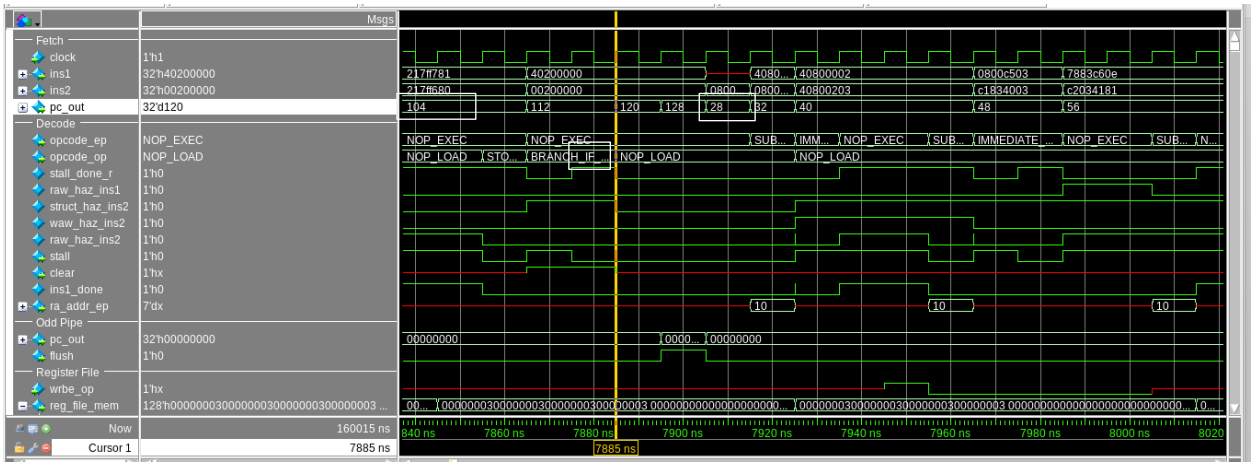
For the innermost loop:



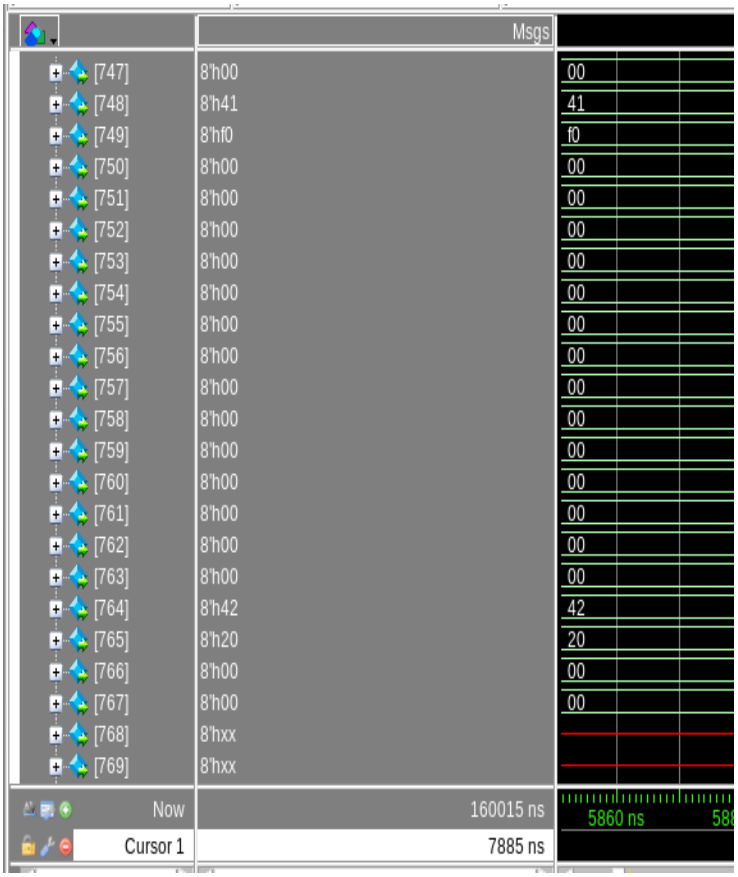
For the 2nd loop:



For the outer loop:



A*B result in Local store:



7) Appendix

a. Parser.cc

```
#include <iostream>

#include <cstring>

#include <fstream>

#include <string>

#include <sstream>


using namespace std;


char *DecToBin(string nums, unsigned bit);
```

```
int main()
{
   ifstream inFile("Assembly.txt");
   ofstream outFile;
   outFile.open(("Binary.txt"));
   string line, str_null= "000";
   string str[10];
        string temp;
   while(getline(inFile, line))
   {
      char *token = std::strtok(&line[0], " ,()");
         int i=0;
                                        //printf("1 %s",&line[0]);

         while (token != NULL)
                                        {
                                                str[i] = token;
                                                //if(token != NULL)
                                                //{
                                                temp = token;
                                                //}
                                                token = std::strtok(NULL, " ,()");
                                                i++;
                                        }
                                        if(i!=0){
                                        if(str[0] =="ah") {outFile << "00011001000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ah - Add Halfword
                                        else if(str[0] =="ahi") {outFile << "00011101000"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ahi - Add Halfword Immediate
                                        else if(str[0] =="a") {outFile << "00011000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  a - Add Word
                                        else if(str[0] =="ai") {outFile << "00011100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ai - Add Word Immediate
                                        else if(str[0] =="sfh") {outFile << "00001001000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // sfh - Subtract from halfword
                                        else if(str[0] =="sfhi") {outFile << "00001101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // sfhi - Subtract from halfword immediate
                                        else if(str[0] =="sf") {outFile << "00001000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //   sf - Subtract from Word
                                        else if(str[0] =="sfi") {outFile << "00001100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  sfi - Subtract from Word Immediate
                                        else if(str[0] =="mpy") {outFile << "01111000100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  mpy - Multiply
```

```
                                    else if(str[0] =="mpyu") {outFile << "01111001100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  mpyu - Multiply Unsigned

                                    else if(str[0] =="mpyi") {outFile << "01110100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  mpyi - Multiply Immediate

                                    else if(str[0] =="mpya") {outFile << "1100"<<DecToBin(str[1], 7)<<DecToBin(str[3], 7)<<DecToBin(str[2],
7)<<DecToBin(str[4], 7)<<endl;}  //  mpya - Multiply and Add

                                    else if(str[0] =="clz") {outFile << "01010100101"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // clz - count leading zeros

                                    else if(str[0] =="fsmbi") {outFile << "001100101"<<DecToBin(str[2],16)<<DecToBin(str[1], 7)<<endl;}  // fsmbi
FORM_SELECT_MASK_BYTES_IMMEDIATE

                                    else if(str[0] =="fsmb") {outFile << "00110110110"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;} // fsmb FORM_SELECT_MASK_BYTES

                                    else if(str[0] =="fsmh") {outFile << "00110110101"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;} // fsmh FORM_SELECT_MASK_HALFWORD

                                    else if(str[0] =="fsm") {outFile << "00110110100"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;} //  fsm FORM_SELECT_MASK_WORD

                                    else if(str[0] =="and") {outFile << "00011000001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // and - And

                                    else if(str[0] =="andhi") {outFile << "00010101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // andhi - And half WORD Immediate

                                    else if(str[0] =="andi") {outFile << "00010100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // andi - And WORD Immediate

                                    else if(str[0] =="or") {outFile << "00001000001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // or - Or

                                    else if(str[0] =="orbi") {outFile << "00000110"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // orbi - Or byte Immediate

                                    else if(str[0] =="orhi") {outFile << "00000101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // orhi - Or half Word Immediate

                                    else if(str[0] =="ori") {outFile << "00000100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ori - Or Word Immediate

                                    else if(str[0] =="nor") {outFile << "00001001001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // nor - Nor

                                    else if(str[0] =="xor") {outFile << "01001000001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // xor - Exclusive Or

                                    else if(str[0] =="xorbi") {outFile << "01000110"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // xorbi - Exclusive Or Byte Immediate

                                    else if(str[0] =="xorhi") {outFile << "01000101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // xorhi - Exclusive Or half Word Immediate

                                    else if(str[0] =="xori") {outFile << "01000100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // xori - Exclusive Or Word Immediate

                                    else if(str[0] =="fa") {outFile << "01011000100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // fa - Floating Add

                                    else if(str[0] =="fs") {outFile << "01011000101"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // fs - Floating Subtract

                                    else if(str[0] =="fm") {outFile << "01011000110"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // fm - Floating Multiply

                                    else if(str[0] =="fma") {outFile << "1110"<<DecToBin(str[1], 7)<<DecToBin(str[3], 7)<<DecToBin(str[2],
7)<<DecToBin(str[4], 7)<<endl;}  //fma - Floating Multiply and Add

                                    else if(str[0] =="fms") {outFile << "1111"<<DecToBin(str[1], 7)<<DecToBin(str[3], 7)<<DecToBin(str[2],
7)<<DecToBin(str[4], 7)<<endl;}  // fms - Floating Multiply and Subtract

                                    else if(str[0] =="eqv") {outFile << "01001001001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // eqv - EQUIVALENT
```

```
                                    else if(str[0] =="shlh") {outFile << "00001011111"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shlh - SHIFT_LEFT_HALFWORD

                                    else if(str[0] =="shlhi") {outFile << "00001111111"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shlhi - SHIFT_LEFT_HALFWORD_IMMEDIATE

                                    else if(str[0] =="shl") {outFile << "00001011011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shl - SHIFT_LEFT_WORD

                                    else if(str[0] =="shli") {outFile << "00001111011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shli - SHIFT_LEFT_WORD_IMMEDIATE

                                    else if(str[0] =="roth") {outFile << "00001011100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // roth - ROTATE_HALFWORD

                                    else if(str[0] =="rothi") {outFile << "00001111100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rothi - ROTATE_HALFWORD_IMMEDIATE

                                    else if(str[0] =="rot") {outFile << "00001011000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rot - Rotate Word

                                    else if(str[0] =="roti") {outFile << "00001111000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // roti - ROTATE_WORD_IMMEDIATE

                                    else if(str[0] =="rothm") {outFile << "00001011101"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rothm - Rotate and Mask halfWord

                                    else if(str[0] =="rothmi") {outFile << "00001111101"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rothmi - Rotate and Mask halfWord immediate

                                    else if(str[0] =="cntb") {outFile << "01010110100"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cntb - Count Ones in Bytes

                                    else if(str[0] =="avgb") {outFile << "00011010011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // avgb - Average Bytes

                                    else if(str[0] =="absdb") {outFile << "00001010011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // absdb - Absolute Differences of Bytes

                                    else if(str[0] =="sumb") {outFile << "01001010011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  sumb - Sum Bytes into Halfwords

                                    else if(str[0] =="ceqh") {outFile << "01111001000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ceqh - Compare Equal halfWord

                                    else if(str[0] =="ceqhi") {outFile << "01111101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  ceqhi - Compare Equal halfWord Immediate

                                    else if(str[0] =="ceq") {outFile << "01111000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ceq - Compare Equal Word

                                    else if(str[0] =="ceqi") {outFile << "01111100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // ceqi - Compare Equal Word Immediate

                                    else if(str[0] =="cgth") {outFile << "01001001000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cgth - Compare Greater Than halfword

                                    else if(str[0] =="cgthi") {outFile << "01001101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cgthi - Compare Greater Than halfWord Immediate

                                    else if(str[0] =="cgt") {outFile << "01001000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cgt - Compare Greater Than word

                                    else if(str[0] =="cgti") {outFile << "01001100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cgti - Compare Greater Than Word Immediate

                                    else if(str[0] =="clgt") {outFile << "01011000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // clgt - Compare Logical Greater Than word

                                    else if(str[0] =="clgti") {outFile << "01011100"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // clgti - Compare Logical Greater Than word Immediate

                                    else if(str[0] =="clgth") {outFile << "01011001000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // clght - Compare Logical Greater Than half word

                                    else if(str[0] =="clgthi") {outFile << "01011101"<<DecToBin(str[3], 10)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  clghti - Compare Logical Greater Than halfword Immediate
```

```
                                    else if(str[0] =="cg") {outFile << "00011000010"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // cg -   CARRY_GENERATE

                                    else if(str[0] =="addx") {outFile << "11010000000"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // addx - Add Extended

                                    else if(str[0] =="bg") {outFile << "00001000010"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // bg -   BORROW_GENERATE

                                    else if(str[0] =="sfx") {outFile << "01101000001"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // sfx - Subtract from Extended

                                    else if(str[0] =="nop") {outFile << "01000000001"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;}  // nop - No Operation even(Execute)

                                    else if(str[0] =="lnop") {outFile << "00000000001"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;}  //lnop - No Operation odd(Execute)

                                    else if(str[0] =="stop") {outFile << "00000000000"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;} //stop

                                    else if(str[0] =="lqa") {outFile << "001100001"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  // lqa - Load
Quadword (a-form)

                                    else if(str[0] =="lqd") {outFile << "00110100"<<DecToBin(str[2], 10)<<DecToBin(str[3], 7)<<DecToBin(str[1],
7)<<endl;}  // lqd - Load Quadword (D-form)

                                    else if(str[0] =="stqa") {outFile << "001000001"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  // stqa - Store
Quadword (a-form)

                                    else if(str[0] =="stqd") {outFile << "00100100"<<DecToBin(str[2], 10)<<DecToBin(str[3], 7)<<DecToBin(str[1],
7)<<endl;}  //stqd - Store Quadword (D-form)

                                    else if(str[0] =="ilh") {outFile << "010000011"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  // ilh -
Immediate Load Halfword

                                    else if(str[0] =="il") {outFile << "010000001"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  //  il - Immediate
Load Word

                                    else if(str[0] =="ila") {outFile << "0100001"<<DecToBin(str[2], 18)<<DecToBin(str[1], 7)<<endl;}  // ila - Immediate
Load Address

                                    else if(str[0] =="gb") {outFile << "00110110000"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // gb - GATHER_BITS_FROM_WORDS

                                    else if(str[0] =="gbh") {outFile << "00110110001"<<DecToBin(str_null, 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  gbh - GATHER_BITS_FROM_halfWORDS

                                    else if(str[0] =="shufb") {outFile << "1011"<<DecToBin(str[1], 7)<<DecToBin(str[3], 7)<<DecToBin(str[2],
7)<<DecToBin(str[4], 7)<<endl;}  //  shufb -SHUFFLE_BYTES

                                    else if(str[0] =="shlqbii") {outFile << "00111111011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  Shift Left Quadword by bits Immediate

                                    else if(str[0] =="shlqbi") {outFile << "00111011011"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shlqbi - Shift Left Quadword by bits

                                    else if(str[0] =="shlqby") {outFile << "00111011111"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  //  shlqby - Shift Left Quadword by Bytes

                                    else if(str[0] =="shlqbyi") {outFile << "00111111111"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // shlqbyi - Shift Left Quadword by Bytes Immediate

                                    else if(str[0] =="rotqby") {outFile << "00111011100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rotqby - Rotate Quadword by Bytes

                                    else if(str[0] =="rotqbyi") {outFile << "00111111100"<<DecToBin(str[3], 7)<<DecToBin(str[2], 7)<<DecToBin(str[1],
7)<<endl;}  // rotqbyi - Rotate Quadword by Bytes Immediate

                                    else if(str[0] =="br") {outFile << "001100100"<<DecToBin(str[1], 16)<<DecToBin(str_null, 7)<<endl;}  //br - Branch
Relative

                                    else if(str[0] =="bra") {outFile << "001100000"<<DecToBin(str[1], 16)<<DecToBin(str_null, 7)<<endl;}  // bra -
Branch Absolute

                                    else if(str[0] =="brnz") {outFile << "001000010"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  //  brnz -
Branch if Not Zero Word
```

```
                                else if(str[0] =="brz") {outFile << "001000000"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  //  brz - Branch
if Zero Word

                                else if(str[0] =="brhnz") {outFile << "001000110"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  //  brhnz -
Branch if Not Zero halfWord

                                else if(str[0] =="brhz") {outFile << "001000100"<<DecToBin(str[2], 16)<<DecToBin(str[1], 7)<<endl;}  // brhz -
Branch if halfZero Word

                                else {cout<<str[0]<<" Instruction not Found"<<endl;}}

                                //printf("Data is %s\n",token);}


                                /*else if(i==1)

                                {

                                        //printf("data is %s\n",temp);

                                        if(temp =="nop") {outFile << "01000000001"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;}  // nop - No Operation even(Execute)

                                        else if(temp =="lnop") {outFile << "00000000001"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;}  //lnop - No Operation odd(Execute)

                                        else if(temp =="stop") {outFile << "00000000000"<<DecToBin(str_null, 7)<<DecToBin(str_null,
7)<<DecToBin(str_null, 7)<<endl;printf("stop\n");} // stop

                                        else {cout<<temp<<" Instruction not Found Single"<<endl;}

                                }

                                else if(i==0)

                                {

                                        printf("Empty line\n");

                                }*/

        }

   outFile.close();

}



//Decimal to Binary

char* DecToBin(string nums, unsigned bit)

{

   int num;

   istringstream (nums) >> num;

   char *binStr = new char (bit + 1);

   int len = bit;


   binStr[bit] = '\0';

   while (bit--)   binStr[bit] = '0';


   if (num == 0)

      return binStr;
```

```
        int r;
        while (num && len)
        {
          r = num % 2;
          binStr[--len] = r + '0';
          num /= 2;
        }

        return binStr;
      }
```

b. Fetch.sv:

```
module fetch(clock,reset,stall,flush,pc_in,ins1,ins2,pc_out,flush_fetch);
        input clock,reset;
        input stall;  //structural or data hazards make stall one;last indicates that the last instruction is executed
        input flush; //bt_addr is branch taken address
        input [0:31] pc_in;
        output logic [0:31] ins1,ins2;

        output logic [0:31] pc_out;
        logic [0:7] imem [2048];
        logic [0:31] imem_temp[512];
        logic last;
        output logic flush_fetch;
        logic pc_inc_4,stall_r;
        int pc;
        initial begin
    $readmemb("Binary.txt", imem_temp);
                for(int i = 0;i<512/*imem_temp[i]!=0*/;i++)begin
                        imem[4*i]     = imem_temp[i][0:7];
                        imem[(4*i)+1] = imem_temp[i][8:15];
                        imem[(4*i)+2] = imem_temp[i][16:23];
                        imem[(4*i)+3] = imem_temp[i][24:31];
                end
        end
```

```
always_ff @(posedge clock) begin
        if(reset) begin
                pc_out <= 0;
                flush_fetch <= 0;
        end
        else if (flush) begin //flush is 1 when the branch is taken
                pc_out <= pc_in;
                flush_fetch <= flush;
        end
        else if (stall) begin
                pc_out <= pc_out;
                flush_fetch <= flush;
        end
        //if previous PC from branch was an odd address the next cycle
        //PC should be incremented by 4 not 8, because instruction onw was passed a X.
        else if(pc_inc_4 ==1) begin
                pc_out <= pc_out + 4;
                flush_fetch <= flush;
        end
        else begin
                pc_out <= pc_out+8;
                flush_fetch <= flush;
        end
end

always_ff @(posedge clock) begin
        if(reset == 1) begin
                stall_r <= 0;
        end
        else begin
                stall_r <= stall;
        end
end

always_comb begin
        pc_inc_4 = 0;
        pc = pc_out;
        // Stop and Signal seen Yet?
```

```
                if (last == 1  && stall_r == 0) begin
                        ins1[0:31] = {11'b01000000001, 21'bx};
                        ins2[0:31] = {11'b00000000001, 21'bx};
                end
                else begin
                        //If new PC from branch instruct is an odd address
                        if(flush_fetch == 1 && pc_out[29] == 1) begin
                                if ({imem[pc],imem[pc+1][0:2]} == 11'b00000000000) begin
                                        last = 1;
                                end
                                        ins1[0:31] = 'dx;
                                        ins2[0:31] = {imem[pc],imem[pc+1],imem[pc+2],imem[pc+3]};
                                        //Set this for the next cycle PC to be incremented by 4 not 8.
                                        pc_inc_4  = 1;
                        end
                        else begin
                                //Normal case
                                if({imem[pc],imem[pc+1][0:2]} != 11'b00000000000 && {imem[pc+4],imem[pc+5][0:2]} != 11'b00000000000)
begin
                                        ins1[0:31] = {imem[pc],imem[pc+1],imem[pc+2],imem[pc+3]};
                                        ins2[0:31] = {imem[pc+4],imem[pc+5],imem[pc+6],imem[pc+7]};
                                end

                                else if ({imem[pc],imem[pc+1][0:2]} == 11'b00000000000) begin
                                        last = 1;
                                        ins1[0:31] = {imem[pc],imem[pc+1],imem[pc+2],imem[pc+3]};
                                        ins2[0:31] = {11'b01000000001, 21'bx};
                                end
                                else begin    //(imem[pc+4] = 11'b00000000000)
                                        //So that next cycle
                                        last = 1;
                                        ins1[0:31] = {imem[pc],imem[pc+1],imem[pc+2],imem[pc+3]};
                                        ins2[0:31] = {imem[pc+4],imem[pc+5],imem[pc+6],imem[pc+7]};
                                end
                        end
                end
        end
endmodule
```

c. Decode.sv:

```
import definitions::*;

module decode(clock,reset,pc_in,ins1,ins2,flush_fetch,flush,pc_out,opcode_ep,I7_ep,I10_ep,I16_ep,I18_ep,

wr_en_rt_ep,opcode_op,I7_op,I10_op,I16_op,I18_op,wr_en_rt_op,ra_addr_ep,rb_addr_ep,

rc_addr_ep,rt_addr_ep,ra_addr_op,rb_addr_op,rc_addr_op,rt_addr_op,clear,

fw_ep_01_addr,fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr,

fw_op_01_addr,fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr,

rf_addr_ep_r,rf_addr_op_r,wr_en_rt_ep_r, wr_en_rt_op_r,wr_en_ep_01,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep_05,wr_en_ep_06,wr_en_ep_07,

wr_en_op_01,wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,

fw_uid_ep_02, fw_uid_ep_03, fw_uid_ep_04, fw_uid_ep_05, fw_uid_ep_06, fw_uid_ep_07,

fw_uid_op_02,fw_uid_op_03,fw_uid_op_04,fw_uid_op_05,fw_uid_op_06,fw_uid_op_07,stall

);

        parameter UNIT_1 = 4'd1;

        parameter UNIT_2 = 4'd2;

        parameter UNIT_3 = 4'd3;

        parameter UNIT_4 = 4'd4;

        parameter UNIT_5 = 4'd5;

        parameter UNIT_6 = 4'd6;

        parameter UNIT_7 = 4'd7;

        parameter UNIT_8 = 4'd8;

        input clock,reset;

        input [0:31] pc_in;

        input [0:31] ins1,ins2;

        input flush;

        input flush_fetch;


        output logic [0:31] pc_out;

        output opcode opcode_ep;

        output logic [0:6] I7_ep;

        output logic [0:9] I10_ep;

        output logic [0:15]I16_ep;

        output logic [0:17]I18_ep;

        output logic wr_en_rt_ep;

        output opcode opcode_op;

        output logic [0:6] I7_op;

        output logic [0:9] I10_op;
```

```
        output logic [0:15]I16_op;
        output logic [0:17]I18_op;
        output logic wr_en_rt_op;
        output logic [0:6] ra_addr_ep;
        output logic [0:6] rb_addr_ep;
        output logic [0:6] rc_addr_ep;
        output logic [0:6] rt_addr_ep;
        output logic [0:6] ra_addr_op;
        output logic [0:6] rb_addr_op,rc_addr_op;
        output logic [0:6] rt_addr_op;
        output logic clear;

        output logic stall;

        logic [0:3]  ins1_4;
logic [0:3]  ins2_4;
logic [0:6]  ins1_7;
logic [0:6]  ins2_7;
logic [0:7]  ins1_8;
logic [0:7]  ins2_8;
logic [0:8]  ins1_9;
logic [0:8]  ins2_9;
logic [0:10] ins1_11;
logic [0:10] ins2_11;

        opcode opcode_ins1;
        logic [0:31] ins1_r, ins2_r;
        logic ins1_type, ins2_type;
        logic [0:6] I7_ins1;
        logic [0:7] I8_ins1;
        logic [0:9] I10_ins1;
        logic [0:15]I16_ins1;
        logic [0:17]I18_ins1;
        logic wr_en_rt_ins1;
        opcode opcode_ins2;
        logic [0:6] I7_ins2;
        logic [0:7] I8_ins2;
```

```
        logic [0:9] I10_ins2;
        logic [0:15]I16_ins2;
        logic [0:17]I18_ins2;
        logic wr_en_rt_ins2;
        logic [0:6] ra_addr_ins1;
        logic [0:6] rb_addr_ins1;
        logic [0:6] rc_addr_ins1;
        logic [0:6] rt_addr_ins1;
        logic [0:6] ra_addr_ins2;
        logic [0:6] rb_addr_ins2,rc_addr_ins2;
        logic [0:6] rt_addr_ins2;
        logic stall_done, stall_done_r,ins1_done,ins1_done_r,wi1_done,wi1_done_r;
        logic raw_haz_ins1, struct_haz_ins2, waw_haz_ins2, raw_haz_ins2;
        logic flush_fetch_r;



        input [0:6] fw_ep_01_addr,fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr;//Address probes from even pipe stages
        input [0:6] fw_op_01_addr,fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr;//Address probes from odd pipe stages
        input wr_en_ep_01,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep_05,wr_en_ep_06,wr_en_ep_07;// write enable probles from even pipe stage
        input wr_en_op_01,wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06;// write enable probles from odd pipe stage
        input logic [0:3]  fw_uid_ep_02, fw_uid_ep_03, fw_uid_ep_04, fw_uid_ep_05, fw_uid_ep_06, fw_uid_ep_07;
        input logic [0:3] fw_uid_op_02,fw_uid_op_03,fw_uid_op_04,fw_uid_op_05,fw_uid_op_06,fw_uid_op_07;


        input [0:6] rf_addr_ep_r,rf_addr_op_r;// Address probes from RF stage
        input wr_en_rt_ep_r, wr_en_rt_op_r;// write enable probles from RF stage



        assign ins1_4  = ins1_r[0:3];
    assign ins2_4  = ins2_r[0:3];
    assign ins1_7  = ins1_r[0:6];
    assign ins2_7  = ins2_r[0:6];
    assign ins1_8  = ins1_r[0:7];
    assign ins2_8  = ins2_r[0:7];
    assign ins1_9  = ins1_r[0:8];
    assign ins2_9  = ins2_r[0:8];
    assign ins1_11 = ins1_r[0:10];
    assign ins2_11 = ins2_r[0:10];
```

```
always_ff @(posedge clock) begin
		if(reset == 1 || flush == 1)begin
			pc_out <= 0;
			ins1_r <= {11'b01000000001, 21'bx};
			ins2_r <= {11'b00000000001, 21'bx};
			flush_fetch_r <= 0;
		end
		else if(stall == 1) begin
			pc_out <= pc_out;
			ins1_r <= ins1_r;
			ins2_r <= ins2_r;
			flush_fetch_r <= flush_fetch_r;
		end
		else begin
			ins1_r <= ins1;
			ins2_r <= ins2;
			pc_out <= pc_in;
			flush_fetch_r <= flush_fetch;// Change
		end
end

always_ff @(posedge clock) begin
		stall_done_r <= stall_done;
		ins1_done_r  <= ins1_done; //Change
		wi1_done_r  <= wi1_done; //new change lah
end

always_comb begin

		stall = 0;
		raw_haz_ins1 = 0;
		struct_haz_ins2 = 0;
		waw_haz_ins2 = 0;
		raw_haz_ins2 = 0;
		wi1_done = 0;
//Stall
```

```
if(stall_done_r == 0) begin //George added stall == 0)

        //Instruction 1 raw

        if(     (rf_addr_ep_r == ra_addr_ins1 &&  wr_en_rt_ep_r) ||

                (rf_addr_ep_r == rb_addr_ins1 &&  wr_en_rt_ep_r) ||

                (rf_addr_ep_r == rc_addr_ins1 &&  wr_en_rt_ep_r) ||

                (rf_addr_op_r == ra_addr_ins1 &&  wr_en_rt_op_r) ||

                (rf_addr_op_r == rb_addr_ins1 &&  wr_en_rt_op_r) ||

                (rf_addr_op_r == rc_addr_ins1 &&  wr_en_rt_op_r) ||


                ((fw_ep_01_addr == ra_addr_ins1) && wr_en_ep_01 == 1) ||

                ((fw_ep_02_addr == ra_addr_ins1) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||

                ((fw_ep_03_addr == ra_addr_ins1) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||

                ((fw_ep_04_addr == ra_addr_ins1) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||

                ((fw_ep_05_addr == ra_addr_ins1) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||

                ((fw_ep_06_addr == ra_addr_ins1) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||


                ((fw_ep_01_addr == rb_addr_ins1) && wr_en_ep_01 == 1) ||

                ((fw_ep_02_addr == rb_addr_ins1) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||

                ((fw_ep_03_addr == rb_addr_ins1) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||

                ((fw_ep_04_addr == rb_addr_ins1) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||

                ((fw_ep_05_addr == rb_addr_ins1) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||

                ((fw_ep_06_addr == rb_addr_ins1) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||



                ((fw_ep_01_addr == rc_addr_ins1) && wr_en_ep_01 == 1) ||

                ((fw_ep_02_addr == rc_addr_ins1) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||

                ((fw_ep_03_addr == rc_addr_ins1) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||

                ((fw_ep_04_addr == rc_addr_ins1) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||

                ((fw_ep_05_addr == rc_addr_ins1) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||

                ((fw_ep_06_addr == rc_addr_ins1) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||


                ((fw_op_01_addr == ra_addr_ins1) && wr_en_op_01 == 1) ||

                ((fw_op_02_addr == ra_addr_ins1) && wr_en_op_02 == 1) ||

                ((fw_op_03_addr == ra_addr_ins1) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||

                ((fw_op_04_addr == ra_addr_ins1) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||
```

```
                                ((fw_op_05_addr == ra_addr_ins1) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1)) ||



                                ((fw_op_01_addr == rb_addr_ins1) && wr_en_op_01 == 1) ||
                                ((fw_op_02_addr == rb_addr_ins1) && wr_en_op_02 == 1) ||
                                ((fw_op_03_addr == rb_addr_ins1) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||
                                ((fw_op_04_addr == rb_addr_ins1) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||
                                ((fw_op_05_addr == rb_addr_ins1) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1)) ||



                                ((fw_op_01_addr == rc_addr_ins1) && wr_en_op_01 == 1) ||
                                ((fw_op_02_addr == rc_addr_ins1) && wr_en_op_02 == 1) ||
                                ((fw_op_03_addr == rc_addr_ins1) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||
                                ((fw_op_04_addr == rc_addr_ins1) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||
                                ((fw_op_05_addr == rc_addr_ins1) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1)) ) begin

                                raw_haz_ins1 = 1;
                        end
                end

                if(raw_haz_ins1 == 0) begin  //new change lah
                        wi1_done = 1;
                end

                //structural hazard
                if((ins1_type == EVEN && ins2_type == EVEN) ||
(ins1_type == ODD  && ins2_type == ODD)) begin
                        struct_haz_ins2 = 1;
                end
                //3.WAW



                if((rt_addr_ins1 == rt_addr_ins2) && (rt_addr_ins1 !== 7'dx) && (rt_addr_ins2 !== 7'dx)) begin
               /* (rt_addr_i1 == ra_addr_i2 || rt_addr_i2 == ra_addr_i1) ||
                (rt_addr_i1 == rb_addr_i2 || rt_addr_i2 == rb_addr_i1) ||
                (rt_addr_i1 == rc_addr_i2 || rt_addr_i2 == rc_addr_i1)) */
                                waw_haz_ins2 = 1;
```

```
                //              if (stall_done_r == 1) waw_haz_ins2 = 0;
end


        //RAW for ins2
        //if(stall_done_r == 0) begin // added by george

                if( (rt_addr_ins1 == ra_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||  //next 6 lines new change added lah
                        (rt_addr_ins1 == rb_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||
                        (rt_addr_ins1 == rc_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||
                        (rt_addr_ins1 == ra_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||
                        (rt_addr_ins1 == rb_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||
                        (rt_addr_ins1 == rc_addr_ins2 && wr_en_rt_ins1 && wi1_done_r == 0) ||

                        (rf_addr_ep_r == ra_addr_ins2 &&  wr_en_rt_ep_r) ||
                        (rf_addr_ep_r == rb_addr_ins2 &&  wr_en_rt_ep_r) ||
                        (rf_addr_ep_r == rc_addr_ins2 &&  wr_en_rt_ep_r) ||
                        (rf_addr_op_r == ra_addr_ins2 &&  wr_en_rt_op_r) ||
                        (rf_addr_op_r == rb_addr_ins2 &&  wr_en_rt_op_r) ||
                        (rf_addr_op_r == rc_addr_ins2 &&  wr_en_rt_op_r) ||

                        ((fw_ep_01_addr == ra_addr_ins2) && wr_en_ep_01 == 1) ||
                        ((fw_ep_02_addr == ra_addr_ins2) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||
                        ((fw_ep_03_addr == ra_addr_ins2) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||
                        ((fw_ep_04_addr == ra_addr_ins2) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||
                        ((fw_ep_05_addr == ra_addr_ins2) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||
                        ((fw_ep_06_addr == ra_addr_ins2) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||

                        ((fw_ep_01_addr == rb_addr_ins2) && wr_en_ep_01 == 1) ||
                        ((fw_ep_02_addr == rb_addr_ins2) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||
                        ((fw_ep_03_addr == rb_addr_ins2) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||
                        ((fw_ep_04_addr == rb_addr_ins2) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||
                        ((fw_ep_05_addr == rb_addr_ins2) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||
                        ((fw_ep_06_addr == rb_addr_ins2) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||


                        ((fw_ep_01_addr == rc_addr_ins2) && wr_en_ep_01 == 1) ||
```

```
                                ((fw_ep_02_addr == rc_addr_ins2) && (fw_uid_ep_02 == UNIT_2 || fw_uid_ep_02 == UNIT_3 || fw_uid_ep_02 == UNIT_4 ||
fw_uid_ep_02 == UNIT_5) && (wr_en_ep_02 == 1)) ||

                                ((fw_ep_03_addr == rc_addr_ins2) && (fw_uid_ep_03 == UNIT_3 || fw_uid_ep_03 == UNIT_4) && (wr_en_ep_03 == 1)) ||

                                ((fw_ep_04_addr == rc_addr_ins2) && (fw_uid_ep_04 == UNIT_3 || fw_uid_ep_04 == UNIT_4) && (wr_en_ep_04 == 1)) ||

                                ((fw_ep_05_addr == rc_addr_ins2) && (fw_uid_ep_05 == UNIT_3 || fw_uid_ep_05 == UNIT_4) && (wr_en_ep_05 == 1))  ||

                                ((fw_ep_06_addr == rc_addr_ins2) && (fw_uid_ep_06 == UNIT_4) && (wr_en_ep_06 == 1)) ||


                                ((fw_op_01_addr == ra_addr_ins2) && wr_en_op_01 == 1) ||

                                ((fw_op_02_addr == ra_addr_ins2) && wr_en_op_02 == 1) ||

                                ((fw_op_03_addr == ra_addr_ins2) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||

                                ((fw_op_04_addr == ra_addr_ins2) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||

                                ((fw_op_05_addr == ra_addr_ins2) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1)) ||



                                ((fw_op_01_addr == rb_addr_ins2) && wr_en_op_01 == 1) ||

                                ((fw_op_02_addr == rb_addr_ins2) && wr_en_op_02 == 1) ||

                                ((fw_op_03_addr == rb_addr_ins2) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||

                                ((fw_op_04_addr == rb_addr_ins2) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||

                                ((fw_op_05_addr == rb_addr_ins2) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1)) ||




                                ((fw_op_01_addr == rc_addr_ins2) && wr_en_op_01 == 1) ||

                                ((fw_op_02_addr == rc_addr_ins2) && wr_en_op_02 == 1) ||

                                ((fw_op_03_addr == rc_addr_ins2) && (fw_uid_op_03 == UNIT_7) && (wr_en_op_03 == 1)) ||

                                ((fw_op_04_addr == rc_addr_ins2) && (fw_uid_op_04 == UNIT_7) && (wr_en_op_04 == 1)) ||

                                ((fw_op_05_addr == rc_addr_ins2) && (fw_uid_op_05 == UNIT_7) && (wr_en_op_05 == 1))              ) begin


                                raw_haz_ins2 = 1;

                        end

                //end


                if (stall_done_r == 1 && raw_haz_ins2 == 0 && raw_haz_ins1 == 0 ) begin

                        stall = 0;

                end

                else if (raw_haz_ins2 == 1 || waw_haz_ins2 == 1 || struct_haz_ins2 == 1 || raw_haz_ins1 ==1) begin

                        stall = 1;

                end
```

```
end



//Output
always_comb begin
opcode_ep  = NOP_EXEC;
I7_ep     = 'dx;
I10_ep    = 'dx;
I16_ep    = 'dx;
I18_ep    = 'dx;
wr_en_rt_ep = 'dx;
opcode_op = NOP_LOAD;
I7_op = 'dx;
I10_op = 'dx;
I16_op = 'dx;
I18_op = 'dx;
wr_en_rt_op = 'dx;
ra_addr_ep = 'dx;
rb_addr_ep = 'dx;
rc_addr_ep = 'dx;
rt_addr_ep = 'dx;
ra_addr_op = 'dx;
rb_addr_op = 'dx;
rc_addr_op = 'dx;
rt_addr_op = 'dx;
stall_done = 0;
ins1_done  = 0;

if(pc_out[29] == 1 && flush_fetch_r) begin
        if(ins2_type == EVEN) begin
                opcode_ep   = opcode_ins2;
                I7_ep       = I7_ins2;
                I10_ep      = I10_ins2;
                I16_ep      = I16_ins2;
                I18_ep      = I18_ins2;
                wr_en_rt_ep = wr_en_rt_ins2;
```

```
                ra_addr_ep  = ra_addr_ins2;
                rb_addr_ep  = rb_addr_ins2;
                rc_addr_ep  = rc_addr_ins2;
                rt_addr_ep  = rt_addr_ins2;
                opcode_op   = NOP_LOAD;
                I7_op                   = 'dx;
                I10_op                  = 'dx;
                I16_op                  = 'dx;
                I18_op                  = 'dx;
                wr_en_rt_op = 'dx;
                ra_addr_op  = 'dx;
                rb_addr_op  = 'dx;
                rc_addr_op  = 'dx;
                rt_addr_op  = 'dx;
        end
        else begin
                opcode_ep   = NOP_EXEC;
                I7_ep       = 'dx;
                I10_ep      = 'dx;
                I16_ep      = 'dx;
                I18_ep      = 'dx;
                wr_en_rt_ep = 'dx;
                ra_addr_ep  = 'dx;
                rb_addr_ep  = 'dx;
                rc_addr_ep  = 'dx;
                rt_addr_ep  = 'dx;
                opcode_op   = opcode_ins2;
                I7_op       = I7_ins2;
                I10_op      = I10_ins2;
                I16_op      = I16_ins2;
                I18_op      = I18_ins2;
                wr_en_rt_op = wr_en_rt_ins2;
                ra_addr_op  = ra_addr_ins2;
                rb_addr_op  = rb_addr_ins2;
                rc_addr_op  = rc_addr_ins2;
                rt_addr_op  = rt_addr_ins2;
        end
end
```

```
//Case 1: (if ins1 type == EVEN && ins2 type == ODD && no hazard)
else if (stall == 0 && struct_haz_ins2 == 0 && waw_haz_ins2 == 0 && ins1_done_r == 0) begin
        if (ins1_type == EVEN && ins2_type == ODD) begin
                opcode_ep  = opcode_ins1;
                I7_ep      = I7_ins1;
                I10_ep     = I10_ins1;
                I16_ep     = I16_ins1;
                I18_ep     = I18_ins1;
                wr_en_rt_ep = wr_en_rt_ins1;
                ra_addr_ep  = ra_addr_ins1;
                rb_addr_ep  = rb_addr_ins1;
                rc_addr_ep  = rc_addr_ins1;
                rt_addr_ep  = rt_addr_ins1;
                opcode_op  = opcode_ins2;
                I7_op      = I7_ins2;
                I10_op     = I10_ins2;
                I16_op     = I16_ins2;
                I18_op     = I18_ins2;
                wr_en_rt_op = wr_en_rt_ins2;
                ra_addr_op  = ra_addr_ins2;
                rb_addr_op  = rb_addr_ins2;
                rc_addr_op  = rc_addr_ins2;
                rt_addr_op = rt_addr_ins2;
        end
        //Case 2: (if ins1 type == ODD && ins2 type == EVEN && no hazard)
        else begin
                opcode_ep  = opcode_ins2;
                I7_ep      = I7_ins2;
                I10_ep     = I10_ins2;
                I16_ep     = I16_ins2;
                I18_ep     = I18_ins2;
                wr_en_rt_ep = wr_en_rt_ins2;
                ra_addr_ep  = ra_addr_ins2;
                rb_addr_ep  = rb_addr_ins2;
                rc_addr_ep  = rc_addr_ins2;
                rt_addr_ep  = rt_addr_ins2;
                opcode_op  = opcode_ins1;
```

```
                        I7_op          = I7_ins1;
                        I10_op         = I10_ins1;
                        I16_op         = I16_ins1;
                        I18_op         = I18_ins1;
                        wr_en_rt_op = wr_en_rt_ins1;
                        ra_addr_op  = ra_addr_ins1;
                        rb_addr_op  = rb_addr_ins1;
                        rc_addr_op  = rc_addr_ins1;
                        rt_addr_op  = rt_addr_ins1;
                end
end
//raw hazard for instr 1
else if (raw_haz_ins1 == 1) begin
                opcode_ep  = NOP_EXEC;
                I7_ep          = 'dx;
                I10_ep         = 'dx;
                I16_ep         = 'dx;
                I18_ep         = 'dx;
                wr_en_rt_ep = 'dx;
                ra_addr_ep  = 'dx;
                rb_addr_ep  = 'dx;
                rc_addr_ep  = 'dx;
                rt_addr_ep  = 'dx;
                opcode_op  = NOP_LOAD;
                I7_op                  = 'dx;
                I10_op                 = 'dx;
                I16_op                 = 'dx;
                I18_op                 = 'dx;
                wr_en_rt_op = 'dx;
                ra_addr_op  = 'dx;
                rb_addr_op  = 'dx;
                rc_addr_op  = 'dx;
                rt_addr_op  = 'dx;
end
else if ((waw_haz_ins2 == 1 || struct_haz_ins2 == 1 ) && stall_done_r == 0) begin
                if(ins1_type == EVEN) begin
                        opcode_ep  = opcode_ins1;
                        I7_ep          = I7_ins1;
```

```
                I10_ep        = I10_ins1;
                I16_ep        = I16_ins1;
                I18_ep        = I18_ins1;
                wr_en_rt_ep = wr_en_rt_ins1;
                ra_addr_ep  = ra_addr_ins1;
                rb_addr_ep  = rb_addr_ins1;
                rc_addr_ep  = rc_addr_ins1;
                rt_addr_ep  = rt_addr_ins1;
                opcode_op   = NOP_LOAD;
                I7_op                   = 'dx;
                I10_op                  = 'dx;
                I16_op                  = 'dx;
                I18_op                  = 'dx;
                wr_en_rt_op = 'dx;
                ra_addr_op  = 'dx;
                rb_addr_op  = 'dx;
                rc_addr_op  = 'dx;
                rt_addr_op  = 'dx;
                stall_done    = 1;
                $monitor($time,"Hello");
end
else begin
                opcode_ep   = NOP_EXEC;
                I7_ep         = 'dx;
                I10_ep        = 'dx;
                I16_ep        = 'dx;
                I18_ep        = 'dx;
                wr_en_rt_ep = 'dx;
                ra_addr_ep  = 'dx;
                rb_addr_ep  = 'dx;
                rc_addr_ep  = 'dx;
                rt_addr_ep  = 'dx;
                opcode_op   = opcode_ins1;
                I7_op         = I7_ins1;
                I10_op        = I10_ins1;
                I16_op        = I16_ins1;
                I18_op        = I18_ins1;
                wr_en_rt_op = wr_en_rt_ins1;
```

```
                ra_addr_op  = ra_addr_ins1;
                rb_addr_op  = rb_addr_ins1;
                rc_addr_op  = rc_addr_ins1;
                rt_addr_op  = rt_addr_ins1;
                stall_done= 1;
        end
end
//raw hazard ins2
else if (raw_haz_ins2 == 1) begin
        if(waw_haz_ins2 == 0 && struct_haz_ins2 == 0 && ins1_done_r == 0) begin
                if(ins1_type == EVEN) begin
                        opcode_ep   = opcode_ins1;
                        I7_ep       = I7_ins1;
                        I10_ep      = I10_ins1;
                        I16_ep      = I16_ins1;
                        I18_ep      = I18_ins1;
                        wr_en_rt_ep = wr_en_rt_ins1;
                        ra_addr_ep  = ra_addr_ins1;
                        rb_addr_ep  = rb_addr_ins1;
                        rc_addr_ep  = rc_addr_ins1;
                        rt_addr_ep  = rt_addr_ins1;
                        opcode_op   = NOP_LOAD;
                        I7_op                   = 'dx;
                        I10_op                  = 'dx;
                        I16_op                  = 'dx;
                        I18_op                  = 'dx;
                        wr_en_rt_op = 'dx;
                        ra_addr_op  = 'dx;
                        rb_addr_op  = 'dx;
                        rc_addr_op  = 'dx;
                        rt_addr_op  = 'dx;
                        stall_done    = 1;
                        ins1_done   = 1;
                end
                else begin
                        opcode_ep   = NOP_EXEC;
                        I7_ep       = 'dx;
                        I10_ep      = 'dx;
```

```
                    I16_ep        = 'dx;
                    I18_ep        = 'dx;
                    wr_en_rt_ep = 'dx;
                    ra_addr_ep  = 'dx;
                    rb_addr_ep  = 'dx;
                    rc_addr_ep  = 'dx;
                    rt_addr_ep  = 'dx;
                    opcode_op   = opcode_ins1;
                    I7_op         = I7_ins1;
                    I10_op        = I10_ins1;
                    I16_op        = I16_ins1;
                    I18_op        = I18_ins1;
                    wr_en_rt_op = wr_en_rt_ins1;
                    ra_addr_op  = ra_addr_ins1;
                    rb_addr_op  = rb_addr_ins1;
                    rc_addr_op  = rc_addr_ins1;
                    rt_addr_op  = rt_addr_ins1;
                    stall_done= 1;
                    ins1_done   = 1;
            end
end
else begin
            opcode_ep   = NOP_EXEC;
            I7_ep         = 'dx;
            I10_ep        = 'dx;
            I16_ep        = 'dx;
            I18_ep        = 'dx;
            wr_en_rt_ep = 'dx;
            ra_addr_ep  = 'dx;
            rb_addr_ep  = 'dx;
            rc_addr_ep  = 'dx;
            rt_addr_ep  = 'dx;
            opcode_op   = NOP_LOAD;
            I7_op                 = 'dx;
            I10_op                = 'dx;
            I16_op                = 'dx;
            I18_op                = 'dx;
            wr_en_rt_op = 'dx;
```

```
                ra_addr_op  = 'dx;
                rb_addr_op  = 'dx;
                rc_addr_op  = 'dx;
                rt_addr_op  = 'dx;
                stall_done  = 1;
                ins1_done   = 1;
        end
end
else begin
        if(ins2_type == EVEN) begin
                opcode_ep   = opcode_ins2;
                I7_ep       = I7_ins2;
                I10_ep      = I10_ins2;
                I16_ep      = I16_ins2;
                I18_ep      = I18_ins2;
                wr_en_rt_ep = wr_en_rt_ins2;
                ra_addr_ep  = ra_addr_ins2;
                rb_addr_ep  = rb_addr_ins2;
                rc_addr_ep  = rc_addr_ins2;
                rt_addr_ep  = rt_addr_ins2;
                opcode_op   = NOP_LOAD;
                I7_op                   = 'dx;
                I10_op                  = 'dx;
                I16_op                  = 'dx;
                I18_op                  = 'dx;
                wr_en_rt_op = 'dx;
                ra_addr_op  = 'dx;
                rb_addr_op  = 'dx;
                rc_addr_op  = 'dx;
                rt_addr_op  = 'dx;
                //stall_done = 1;
        end
        else begin
                opcode_ep   = NOP_EXEC;
                I7_ep       = 'dx;
                I10_ep      = 'dx;
                I16_ep      = 'dx;
                I18_ep      = 'dx;
```

```
                                wr_en_rt_ep = 'dx;
                                ra_addr_ep  = 'dx;
                                rb_addr_ep  = 'dx;
                                rc_addr_ep  = 'dx;
                                rt_addr_ep  = 'dx;
                                opcode_op   = opcode_ins2;
                                I7_op       = I7_ins2;
                                I10_op      = I10_ins2;
                                I16_op      = I16_ins2;
                                I18_op      = I18_ins2;
                                wr_en_rt_op = wr_en_rt_ins2;
                                ra_addr_op  = ra_addr_ins2;
                                rb_addr_op  = rb_addr_ins2;
                                rc_addr_op  = rc_addr_ins2;
                                rt_addr_op  = rt_addr_ins2;
                                //stall_done= 1;
                        end
        end
        end




        always_comb begin
                        opcode_ins1  = NOP_EXEC;
opcode_ins2  = NOP_LOAD;
ins1_type  = EVEN;
ins2_type  = ODD;
ra_addr_ins1 = 'dx;
rb_addr_ins1 = 'dx;
rc_addr_ins1 = 'dx;
rt_addr_ins1 = 'dx;
ra_addr_ins2 = 'dx;
rb_addr_ins2 = 'dx;
rc_addr_ins2 = 'dx;
rt_addr_ins2 = 'dx;
I7_ins1   = 'dx;
I8_ins1   = 'dx;
```

```
I10_ins1  = 'dx;
I16_ins1  = 'dx;
I18_ins1  = 'dx;
I7_ins2   = 'dx;
I8_ins2   = 'dx;
I10_ins2  = 'dx;
I16_ins2  = 'dx;
I18_ins2  = 'dx;
                wr_en_rt_ins1 = 'dx;
                clear = 'dx;

                if(ins1_4 == 4'b1100 || ins1_4 == 4'b1110 || ins1_4 == 4'b1111 || ins1_4 == 4'b1011) begin  //4
                        case(ins1_4)
                                4'b1100:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = MULTIPLY_AND_ADD;
                                                wr_en_rt_ins1 = 1;
                                        end
                                4'b1110:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = FLOATING_MULTIPLY_AND_ADD;
                                                wr_en_rt_ins1 = 1;
                                        end
                                4'b1111:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = FLOATING_MULTIPLY_AND_SUBTRACT;
                                                wr_en_rt_ins1 = 1;
                                        end
                                4'b1011:
                                        begin
                                                ins1_type = ODD;
                                                opcode_ins1 = SHUFFLE_BYTES;
                                                wr_en_rt_ins1 = 1;
                                        end
                        endcase
```

```
                ins1_type  = EVEN;
rt_addr_ins1 = ins1_r[4:10];
ra_addr_ins1 = ins1_r[11:17];
rb_addr_ins1 = ins1_r[18:24];
rc_addr_ins1 = ins1_r[25:31];
            end


        else if(ins1_9 == 9'b001100101 || ins1_9 == 9'b001100100 || ins1_9 == 9'b001100000 || ins1_9 == 9'b001100110 ||
                    ins1_9 == 9'b001100010 || ins1_9 == 9'b001000010 || ins1_9 == 9'b001000000 || ins1_9 == 9'b001000000 ||
                    ins1_9 == 9'b001000110 || ins1_9 == 9'b001000100 || ins1_9 == 9'b001100001 || ins1_9 == 9'b001000001 ||
                    ins1_9 == 9'b010000011 || ins1_9 == 9'b010000001 ) begin  //9


                I16_ins1  = ins1_r[9:24];
rt_addr_ins1 = ins1_r[25:31];
                case(ins1_9)
                9'b001100101:
                        begin
                                ins1_type = EVEN;
                                opcode_ins1 = FORM_SELECT_MASK_BYTES_IMMEDIATE;
                                wr_en_rt_ins1 = 1;
                        end
                9'b001100100:
                        begin
                                ins1_type = ODD;
                                opcode_ins1 = BRANCH_RELATIVE;
                                wr_en_rt_ins1 = 0;
                                clear = 1;
                                rt_addr_ins1 = 7'dx;
                        end
                9'b001100000:
                        begin
                                ins1_type = ODD;
                                opcode_ins1 = BRANCH_ABSOLUTE;
                                wr_en_rt_ins1 = 0;
                                clear = 1;
                                rt_addr_ins1 = 7'dx;
                        end
```

```
9'b001100110:
        begin
                ins1_type = ODD;
                opcode_ins1 = BRANCH_RELATIVE_AND_SET_LINK;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
        end
9'b001100010:
        begin
                ins1_type = ODD;
                opcode_ins1 = BRANCH_ABSOLUTE_AND_SET_LINK;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
        end
9'b001000010:
        begin
                ins1_type = ODD;
                opcode_ins1 = BRANCH_IF_NOT_ZERO_WORD;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
        end
9'b001000000:
        begin
                ins1_type = ODD;
                opcode_ins1 = BRANCH_IF_ZERO_WORD;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
        end
9'b001000110:
        begin
```

```
                ins1_type = ODD;
                opcode_ins1 = BRANCH_IF_NOT_ZERO_HALFWORD;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
            end
        9'b001000100:
            begin
                ins1_type = ODD;
                opcode_ins1 = BRANCH_IF_ZERO_HALFWORD;
                wr_en_rt_ins1 = 0;
                clear = 1;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
            end
        9'b001100001:
            begin
                ins1_type = ODD;
                opcode_ins1 = LOAD_QUADWORD_AFORM;
                wr_en_rt_ins1 = 1;
            end

        9'b001000001:
            begin
                ins1_type = ODD;
                opcode_ins1 = STORE_QUADWORD_AFORM;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
            end

        9'b010000011:
            begin
                ins1_type = ODD;
                opcode_ins1 = IMMEDIATE_LOAD_HALFWORD;
                wr_en_rt_ins1 = 1;
            end
```

```
                        9'b010000001:
                                begin
                                        ins1_type = ODD;
                                        opcode_ins1 = IMMEDIATE_LOAD_WORD;
                                        wr_en_rt_ins1 = 1;
                                end
                        endcase
                end
                else if(ins1_8 == 8'b00011101 || ins1_8 == 8'b00011100 || ins1_8 == 8'b00001101 || ins1_8 == 8'b00001100 ||
                                ins1_8 == 8'b01110100 || ins1_8 == 8'b00010101 || ins1_8 == 8'b00010100 || ins1_8 == 8'b00000110 ||
                                ins1_8 == 8'b00000101 || ins1_8 == 8'b00000100 || ins1_8 == 8'b01000110 || ins1_8 == 8'b01000101 ||
                                ins1_8 == 8'b01000100 || ins1_8 == 8'b01111101 || ins1_8 == 8'b01111100 || ins1_8 == 8'b01001101 ||
                                ins1_8 == 8'b01001100 || ins1_8 == 8'b01011100 || ins1_8 == 8'b01011101 || ins1_8 == 8'b00110100 ||
                                ins1_8 == 8'b00100100) begin  //8


                        rt_addr_ins1 = ins1_r[25:31];
ra_addr_ins1 = ins1_r[18:24];
I10_ins1  = ins1_r[8:17];


                        case(ins1_8)
                                8'b00011101:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = ADD_HALFWORD_IMMEDIATE;
                                                wr_en_rt_ins1 = 1;
                                        end
                                8'b00011100:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = ADD_WORD_IMMEDIATE;
                                                wr_en_rt_ins1 = 1;
                                        end
                                8'b00001101:
                                        begin
                                                ins1_type = EVEN;
                                                opcode_ins1 = SUB_FROM_HALFWORD_IMMEDIATE;
                                                wr_en_rt_ins1 = 1;
                                        end
```

```
8'b00001100:
        begin
                ins1_type = EVEN;
                opcode_ins1 = SUB_FROM_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b01110100:
        begin
                ins1_type = EVEN;
                opcode_ins1 = MULTIPLY_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b00010101:
        begin
                ins1_type = EVEN;
                opcode_ins1 = AND_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b00010100:
        begin
                ins1_type = EVEN;
                opcode_ins1 = AND_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b00000110:
        begin
                ins1_type = EVEN;
                opcode_ins1 = OR_BYTE_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b00000101:
        begin
                ins1_type = EVEN;
                opcode_ins1 = OR_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b00000100:
        begin
```

```
                ins1_type = EVEN;
                opcode_ins1 = OR_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b01000110:
        begin
                ins1_type = EVEN;
                opcode_ins1 = XOR_BYTE_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b01000101:
        begin
                ins1_type = EVEN;
                opcode_ins1 = XOR_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b01000100:
        begin
                ins1_type = EVEN;
                opcode_ins1 = XOR_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
        end
8'b01111101:
begin
        ins1_type = EVEN;
        opcode_ins1 = COMPARE_EQUAL_HALFWORD_IMMEDIATE;
        wr_en_rt_ins1 = 1;
end

8'b01111100:
begin
        ins1_type = EVEN;
        opcode_ins1 = COMPARE_EQUAL_WORD_IMMEDIATE;
        wr_en_rt_ins1 = 1;
end

8'b01001101:
begin
```

```
                ins1_type = EVEN;
                opcode_ins1 = COMPARE_GREATER_THAN_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
end
8'b01001100:
begin
                ins1_type = EVEN;
                opcode_ins1 = COMPARE_GREATER_THAN_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
end
8'b01011100:
begin
                ins1_type = EVEN;
                opcode_ins1 = COMPARE_LOGICAL_GREATER_THAN_WORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
end

8'b01011101:
begin
                ins1_type = EVEN;
                opcode_ins1 = COMPARE_LOGICAL_GREATER_THAN_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
end
8'b00110100:
begin
                ins1_type = ODD;
                opcode_ins1 = LOAD_QUADWORD_DFORM;
                wr_en_rt_ins1 = 1;
end
8'b00100100:
begin
                ins1_type = ODD;
                opcode_ins1 = STORE_QUADWORD_DFORM;
                rc_addr_ins1 = ins1_r[25:31];
                rt_addr_ins1 = 7'dx;
end
```

```
		endcase
end
else begin
		rt_addr_ins1 = ins1_r[25:31];
		ra_addr_ins1 = ins1_r[18:24];
		rb_addr_ins1 = ins1_r[11:17];
		case(ins1_11)
			11'b00011001000:
				begin
					ins1_type = EVEN;
					opcode_ins1 = ADD_HALFWORD;
					wr_en_rt_ins1 = 1;
				end
			11'b00011000000:
				begin
					ins1_type = EVEN;
					opcode_ins1 = ADD_WORD;
					wr_en_rt_ins1 = 1;
				end
			11'b00001001000:
				begin
					ins1_type = EVEN;
					opcode_ins1 = SUB_FROM_HALFWORD;
					wr_en_rt_ins1 = 1;
				end
			11'b00001000000:
				begin
					ins1_type = EVEN;
					opcode_ins1 = SUB_FROM_WORD;
					wr_en_rt_ins1 = 1;
				end
			11'b01111000100:
				begin
					ins1_type = EVEN;
					opcode_ins1 = MULTIPLY;
					wr_en_rt_ins1 = 1;
				end
			11'b01111001100:
```

```
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = MULITPLY_UNSIGNED;
                        wr_en_rt_ins1 = 1;
                end
        11'b01010100101:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COUNTING_LEADING_ZEROS;
                        wr_en_rt_ins1 = 1;
                        rb_addr_ins1 = 7'dx;
                end
        11'b00110110110:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FORM_SELECT_MASK_BYTES;
                        wr_en_rt_ins1 = 1;
                        rb_addr_ins1 = 7'dx;
                end
        11'b00110110101:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FORM_SELECT_MASK_HALFWORD;
                        wr_en_rt_ins1 = 1;
                        rb_addr_ins1 = 7'dx;
                end
        11'b00110110100:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FORM_SELECT_MASK_WORD;
                        wr_en_rt_ins1 = 1;
                        rb_addr_ins1 = 7'dx;
                end
        11'b00011000001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = AND;
                        wr_en_rt_ins1 = 1;
```

```
                end
11'b00001000001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = OR;
                        wr_en_rt_ins1 = 1;
                end
11'b00001001001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = NOR;
                        wr_en_rt_ins1 = 1;
                end
11'b01001000001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = XOR;
                        wr_en_rt_ins1 = 1;
                end
11'b01011000100:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FLOATING_ADD;
                        wr_en_rt_ins1 = 1;
                end
11'b01011000101:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FLOATING_SUBTRACT;
                        wr_en_rt_ins1 = 1;
                end
11'b01011000110:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = FLOATING_MULTIPLY;
                        wr_en_rt_ins1 = 1;
                end
11'b00111011100:
```

```
                begin
                        ins1_type = ODD;
                        opcode_ins1 = ROTATE_QUADWORD_BY_BYTES;
                        wr_en_rt_ins1 = 1;
                end
        11'b00111111100:
                begin
                        ins1_type = ODD;
                        opcode_ins1 = ROTATE_QUADWORD_BY_BYTES_IMMEDIATE;
                        wr_en_rt_ins1 = 1;
                        I7_ins1 = ins1_r[11:17];
                        rb_addr_ins1 = 7'dx;
                end
        11'b01001001001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = EQUIVALENT;
                        wr_en_rt_ins1 = 1;
                end
        11'b00001011111:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SHIFT_LEFT_HALFWORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b00001111111:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SHIFT_LEFT_HALFWORD_IMMEDIATE;
                        wr_en_rt_ins1 = 1;
                        I7_ins1 = ins1_r[11:17];
                        rb_addr_ins1 = 7'dx;
                end
        11'b00001011011:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SHIFT_LEFT_WORD;
                        wr_en_rt_ins1 = 1;
```

```
                end


        11'b00001111011:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SHIFT_LEFT_WORD_IMMEDIATE;
                        wr_en_rt_ins1 = 1;
                        I7_ins1 = ins1_r[11:17];
                        rb_addr_ins1 = 7'dx;
                end
        11'b00001011100:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = ROTATE_HALFWORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b00001111100:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = ROTATE_HALFWORD_IMMEDIATE;
                        wr_en_rt_ins1 = 1;
                        I7_ins1 = ins1_r[11:17];
                        rb_addr_ins1 = 7'dx;
                end
        11'b00001011000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = ROTATE_WORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b00001111000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = ROTATE_WORD_IMMEDIATE;
                        wr_en_rt_ins1 = 1;
                        I7_ins1 = ins1_r[11:17];
                        rb_addr_ins1 = 7'dx;
                end
```

```
11'b00001011101:
        begin
                ins1_type = EVEN;
                opcode_ins1 = ROTATE_AND_MASK_HALFWORD;
                wr_en_rt_ins1 = 1;
        end
11'b00001111101:
        begin
                ins1_type = EVEN;
                opcode_ins1 = ROTATE_AND_MASK_HALFWORD_IMMEDIATE;
                wr_en_rt_ins1 = 1;
                I7_ins1 = ins1_r[11:17];
                rb_addr_ins1 = 7'dx;
        end
11'b01010110100:
        begin
                ins1_type = EVEN;
                opcode_ins1 = COUNT_ONES_IN_BYTES;
                wr_en_rt_ins1 = 1;
                rb_addr_ins1 = 7'dx;
        end
11'b00011010011:
        begin
                ins1_type = EVEN;
                opcode_ins1 = AVERAGE_BYTES;
                wr_en_rt_ins1 = 1;
        end

11'b00001010011:
        begin
                ins1_type = EVEN;
                opcode_ins1 = ABSOLUTE_DIFFERENCE_OF_BYTES;
                wr_en_rt_ins1 = 1;
        end

11'b01001010011:
```

```
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SUM_BYTES_INTO_HALFWORDS;
                        wr_en_rt_ins1 = 1;
                end

        11'b01111001000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_EQUAL_HALFWORD;
                        wr_en_rt_ins1 = 1;
                end

        11'b01111000000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_EQUAL_WORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b01001001000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_GREATER_THAN_HALFWORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b01001000000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_GREATER_THAN_WORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b01011000000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_LOGICAL_GREATER_THAN_WORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b01011001000:
```

```
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = COMPARE_LOGICAL_GREATER_THAN_HALFWORD;
                        wr_en_rt_ins1 = 1;
                end
        11'b00011000010:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = CARRY_GENERATE;
                        wr_en_rt_ins1 = 1;
                end
        11'b01101000000:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = ADD_EXTENDED;
                        wr_en_rt_ins1 = 1;
                end
        11'b00001000010:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = BORROW_GENERATE;
                        wr_en_rt_ins1 = 1;
                end

        11'b01101000001:
                begin
                        ins1_type = EVEN;
                        opcode_ins1 = SUBTRACT_FROM_EXTENDED;
                        wr_en_rt_ins1 = 1;
                end

        11'b00000000001:
                begin
                        ins1_type = ODD;
                        opcode_ins1 = NOP_LOAD;
                        rt_addr_ins1 = 7'dx;
                        ra_addr_ins1 = 7'dx;
                        rb_addr_ins1 = 7'dx;
```

```
            end

        11'b01000000001:
            begin
                ins1_type = EVEN;
                opcode_ins1 = NOP_EXEC;
                rt_addr_ins1 = 7'dx;
                ra_addr_ins1 = 7'dx;
                rb_addr_ins1 = 7'dx;
            end

        11'b00000000000:
            begin
                ins1_type = ODD;
                opcode_ins1 = STOP_AND_SIGNAL;
                rt_addr_ins1 = 7'dx;
                ra_addr_ins1 = 7'dx;
                rb_addr_ins1 = 7'dx;
            end


        11'b00110110000:
            begin
                ins1_type = ODD;
                opcode_ins1 = GATHER_BITS_FROM_WORDS;
                wr_en_rt_ins1 = 1;
                rb_addr_ins1 = 7'dx;
            end

        11'b00110110001:
            begin
                ins1_type = ODD;
                opcode_ins1 = GATHER_BITS_FROM_HALFWORDS;
                wr_en_rt_ins1 = 1;
                rb_addr_ins1 = 7'dx;
            end
```

```
                                        11'b00111011011:
                                                begin
                                                        ins1_type = ODD;
                                                        opcode_ins1 = SHIFT_LEFT_QUADWORD_BY_BITS;
                                                        wr_en_rt_ins1 = 1;
                                                end

                                        11'b00111111011:
                                                begin
                                                        ins1_type = ODD;
                                                        opcode_ins1 = SHIFT_LEFT_QUADWORD_BY_BITS_IMMEDIATE;
                                                        wr_en_rt_ins1 = 1;
                                                        I7_ins1 = ins1_r[11:17];
                                                        rb_addr_ins1 = 7'dx;
                                                end

                                        11'b00111011111:
                                                begin
                                                        ins1_type = ODD;
                                                        opcode_ins1 = SHIFT_LEFT_QUADWORD_BY_BYTES;
                                                        wr_en_rt_ins1 = 1;
                                                end

                                        11'b00111111111:
                                                begin
                                                        ins1_type = ODD;
                                                        opcode_ins1 = SHIFT_LEFT_QUADWORD_BY_BYTES_IMMEDIATE;
                                                        wr_en_rt_ins1 = 1;
                                                        I7_ins1 = ins1_r[11:17];
                                                        rb_addr_ins1 = 7'dx;
                                                end
                                endcase
end

if(ins2_4 == 4'b1100 || ins2_4 == 4'b1110 || ins2_4 == 4'b1111 || ins2_4 == 4'b1011) begin  //4
                                case(ins2_4)
                                        4'b1100:
                                                begin
```

```
                                        ins2_type = EVEN;
                                        opcode_ins2 = MULTIPLY_AND_ADD;
                                        wr_en_rt_ins2 = 1;
                                    end
                                4'b1110:
                                    begin
                                        ins2_type = EVEN;
                                        opcode_ins2 = FLOATING_MULTIPLY_AND_ADD;
                                        wr_en_rt_ins2 = 1;
                                    end
                                4'b1111:
                                    begin
                                        ins2_type = EVEN;
                                        opcode_ins2 = FLOATING_MULTIPLY_AND_SUBTRACT;
                                        wr_en_rt_ins2 = 1;
                                    end
                                4'b1011:
                                    begin
                                        ins2_type = ODD;
                                        opcode_ins2 = SHUFFLE_BYTES;
                                        wr_en_rt_ins2 = 1;
                                    end
                            endcase
                            ins2_type  = EVEN;
rt_addr_ins2 = ins2_r[4:10];
ra_addr_ins2 = ins2_r[11:17];
rb_addr_ins2 = ins2_r[18:24];
rc_addr_ins2 = ins2_r[25:31];
                        end


                    else if(ins2_9 == 9'b001100101 || ins2_9 == 9'b001100100 || ins2_9 == 9'b001100000 || ins2_9 == 9'b001100110 ||
                                        ins2_9 == 9'b001100010 || ins2_9 == 9'b001000010 || ins2_9 == 9'b001000000 || ins2_9 == 9'b001000000 ||
                                        ins2_9 == 9'b001000110 || ins2_9 == 9'b001000100 || ins2_9 == 9'b001100001 || ins2_9 == 9'b001000001 ||
                                        ins2_9 == 9'b010000011 || ins2_9 == 9'b010000001 ) begin  //9


                            I16_ins2  = ins2_r[9:24];
rt_addr_ins2 = ins2_r[25:31];
```

```
case(ins2_9)
9'b001100101:
        begin
                ins2_type = EVEN;
                opcode_ins2 = FORM_SELECT_MASK_BYTES_IMMEDIATE;
                wr_en_rt_ins2 = 1;
        end
9'b001100100:
        begin
                ins2_type = ODD;
                opcode_ins2 = BRANCH_RELATIVE;
                wr_en_rt_ins2 = 0;
                rt_addr_ins2 = 7'dx;
        end
9'b001100000:
        begin
                ins2_type = ODD;
                opcode_ins2 = BRANCH_ABSOLUTE;
                wr_en_rt_ins2 = 0;
                rt_addr_ins2 = 7'dx;
        end
9'b001100110:
        begin
                ins2_type = ODD;
                opcode_ins2 = BRANCH_RELATIVE_AND_SET_LINK;
                wr_en_rt_ins2 = 0;
                rc_addr_ins2 = ins2_r[25:31];
                rt_addr_ins2 = 7'dx;
        end
9'b001100010:
        begin
                ins2_type = ODD;
                opcode_ins2 = BRANCH_ABSOLUTE_AND_SET_LINK;
                wr_en_rt_ins2 = 0;
                rc_addr_ins2 = ins2_r[25:31];
                rt_addr_ins2 = 7'dx;
        end
9'b001000010:
```

```
                begin
                        ins2_type = ODD;
                        opcode_ins2 = BRANCH_IF_NOT_ZERO_WORD;
                        wr_en_rt_ins2 = 0;
                        rc_addr_ins2 = ins2_r[25:31];
                        rt_addr_ins2 = 7'dx;
                end
        9'b001000000:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = BRANCH_IF_ZERO_WORD;
                        wr_en_rt_ins2 = 0;
                        rc_addr_ins2 = ins2_r[25:31];
                        rt_addr_ins2 = 7'dx;
                end
        9'b001000110:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = BRANCH_IF_NOT_ZERO_HALFWORD;
                        wr_en_rt_ins2 = 0;
                        rc_addr_ins2 = ins2_r[25:31];
                        rt_addr_ins2 = 7'dx;
                end
        9'b001000100:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = BRANCH_IF_ZERO_HALFWORD;
                        wr_en_rt_ins2 = 0;
                        rc_addr_ins2 = ins2_r[25:31];
                        rt_addr_ins2 = 7'dx;
                end
        9'b001100001:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = LOAD_QUADWORD_AFORM;
                        wr_en_rt_ins2 = 1;
                end
```

```
                9'b001000001:
                        begin
                                ins2_type = ODD;
                                opcode_ins2 = STORE_QUADWORD_AFORM;
                                rc_addr_ins2 = ins2_r[25:31];
                                rt_addr_ins2 = 7'dx;
                        end

                9'b010000011:
                        begin
                                ins2_type = ODD;
                                opcode_ins2 = IMMEDIATE_LOAD_HALFWORD;
                                wr_en_rt_ins2 = 1;
                        end

                9'b010000001:
                        begin
                                ins2_type = ODD;
                                opcode_ins2 = IMMEDIATE_LOAD_WORD;
                                wr_en_rt_ins2 = 1;
                        end
                endcase
        end
        else if(ins2_8 == 8'b00011101 || ins2_8 == 8'b00011100 || ins2_8 == 8'b00001101 || ins2_8 == 8'b00001100 ||
                        ins2_8 == 8'b01110100 || ins2_8 == 8'b00010101 || ins2_8 == 8'b00010100 || ins2_8 == 8'b00000110 ||
                        ins2_8 == 8'b00000101 || ins2_8 == 8'b00000100 || ins2_8 == 8'b01000110 || ins2_8 == 8'b01000101 ||
                        ins2_8 == 8'b01000100 || ins2_8 == 8'b01111101 || ins2_8 == 8'b01111100 || ins2_8 == 8'b01001101 ||
                        ins2_8 == 8'b01001100 || ins2_8 == 8'b01011100 || ins2_8 == 8'b01011101 || ins2_8 == 8'b00110100 ||
                        ins2_8 == 8'b00100100) begin  //8

                rt_addr_ins2 = ins2_r[25:31];
ra_addr_ins2 = ins2_r[18:24];
I10_ins2  = ins2_r[8:17];

                case(ins2_8)
                        8'b00011101:
                                begin
                                        ins2_type = EVEN;
```

```
                opcode_ins2 = ADD_HALFWORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b00011100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = ADD_WORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b00001101:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SUB_FROM_HALFWORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b00001100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SUB_FROM_WORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b01110100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = MULTIPLY_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b00010101:
            begin
                ins2_type = EVEN;
                opcode_ins2 = AND_HALFWORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
            end
        8'b00010100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = AND_WORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
```

```
            end
8'b00000110:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = OR_BYTE_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b00000101:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = OR_HALFWORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b00000100:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = OR_WORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b01000110:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = XOR_BYTE_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b01000101:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = XOR_HALFWORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b01000100:
            begin
                        ins2_type = EVEN;
                        opcode_ins2 = XOR_WORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
            end
8'b01111101:
```

```
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_EQUAL_HALFWORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end

8'b01111100:
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_EQUAL_WORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end

8'b01001101:
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_GREATER_THAN_HALFWORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end
8'b01001100:
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_GREATER_THAN_WORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end
8'b01011100:
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_LOGICAL_GREATER_THAN_WORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end

8'b01011101:
begin
        ins2_type = EVEN;
        opcode_ins2 = COMPARE_LOGICAL_GREATER_THAN_HALFWORD_IMMEDIATE;
        wr_en_rt_ins2 = 1;
end
```

```
                8'b00110100:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = LOAD_QUADWORD_DFORM;
                        wr_en_rt_ins2 = 1;
                end
                8'b00100100:
                begin
                        ins2_type = ODD;
                        opcode_ins2 = STORE_QUADWORD_DFORM;
                        rc_addr_ins2 = ins2_r[25:31];
                        rt_addr_ins2 = 7'dx;
                end


        endcase
end
else begin
        rt_addr_ins2 = ins2_r[25:31];
        ra_addr_ins2 = ins2_r[18:24];
        rb_addr_ins2 = ins2_r[11:17];
        case(ins2_11)
                11'b00011001000:
                        begin
                                ins2_type = EVEN;
                                opcode_ins2 = ADD_HALFWORD;
                                wr_en_rt_ins2 = 1;
                        end
                11'b00011000000:
                        begin
                                ins2_type = EVEN;
                                opcode_ins2 = ADD_WORD;
                                wr_en_rt_ins2 = 1;
                        end
                11'b00001001000:
                        begin
                                ins2_type = EVEN;
                                opcode_ins2 = SUB_FROM_HALFWORD;
```

```
                wr_en_rt_ins2 = 1;
            end
        11'b00001000000:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SUB_FROM_WORD;
                wr_en_rt_ins2 = 1;
            end
        11'b01111000100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = MULTIPLY;
                wr_en_rt_ins2 = 1;
            end
        11'b01111001100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = MULITPLY_UNSIGNED;
                wr_en_rt_ins2 = 1;
            end
        11'b01010100101:
            begin
                ins2_type = EVEN;
                opcode_ins2 = COUNTING_LEADING_ZEROS;
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end
        11'b00110110110:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FORM_SELECT_MASK_BYTES;
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end
        11'b00110110101:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FORM_SELECT_MASK_HALFWORD;
```

```
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end
        11'b00110110100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FORM_SELECT_MASK_WORD;
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end
        11'b00011000001:
            begin
                ins2_type = EVEN;
                opcode_ins2 = AND;
                wr_en_rt_ins2 = 1;
            end
        11'b00001000001:
            begin
                ins2_type = EVEN;
                opcode_ins2 = OR;
                wr_en_rt_ins2 = 1;
            end
        11'b00001001001:
            begin
                ins2_type = EVEN;
                opcode_ins2 = NOR;
                wr_en_rt_ins2 = 1;
            end
        11'b01001000001:
            begin
                ins2_type = EVEN;
                opcode_ins2 = XOR;
                wr_en_rt_ins2 = 1;
            end
        11'b01011000100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FLOATING_ADD;
```

```
                wr_en_rt_ins2 = 1;
            end
        11'b01011000101:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FLOATING_SUBTRACT;
                wr_en_rt_ins2 = 1;
            end
        11'b01011000110:
            begin
                ins2_type = EVEN;
                opcode_ins2 = FLOATING_MULTIPLY;
                wr_en_rt_ins2 = 1;
            end
        11'b00111011100:
            begin
                ins2_type = ODD;
                opcode_ins2 = ROTATE_QUADWORD_BY_BYTES;
                wr_en_rt_ins2 = 1;
            end
        11'b00111111100:
            begin
                ins2_type = ODD;
                opcode_ins2 = ROTATE_QUADWORD_BY_BYTES_IMMEDIATE;
                wr_en_rt_ins2 = 1;
                I7_ins2 = ins2_r[11:17];
                rb_addr_ins2 = 7'dx;
            end
        11'b01001001001:
            begin
                ins2_type = EVEN;
                opcode_ins2 = EQUIVALENT;
                wr_en_rt_ins2 = 1;
            end
        11'b00001011111:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SHIFT_LEFT_HALFWORD;
```

```
                wr_en_rt_ins2 = 1;
            end
        11'b00001111111:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SHIFT_LEFT_HALFWORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
                I7_ins2 = ins2_r[11:17];
                rb_addr_ins2 = 7'dx;
            end
        11'b00001011011:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SHIFT_LEFT_WORD;
                wr_en_rt_ins2 = 1;
            end

        11'b00001111011:
            begin
                ins2_type = EVEN;
                opcode_ins2 = SHIFT_LEFT_WORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
                I7_ins2 = ins2_r[11:17];
                rb_addr_ins2 = 7'dx;
            end
        11'b00001011100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = ROTATE_HALFWORD;
                wr_en_rt_ins2 = 1;
            end
        11'b00001111100:
            begin
                ins2_type = EVEN;
                opcode_ins2 = ROTATE_HALFWORD_IMMEDIATE;
                wr_en_rt_ins2 = 1;
                I7_ins2 = ins2_r[11:17];
                rb_addr_ins2 = 7'dx;
```

```
                end
11'b00001011000:
                begin
                        ins2_type = EVEN;
                        opcode_ins2 = ROTATE_WORD;
                        wr_en_rt_ins2 = 1;
                end
11'b00001111000:
                begin
                        ins2_type = EVEN;
                        opcode_ins2 = ROTATE_WORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
                        I7_ins2 = ins2_r[11:17];
                        rb_addr_ins2 = 7'dx;
                end


11'b00001011101:
                begin
                        ins2_type = EVEN;
                        opcode_ins2 = ROTATE_AND_MASK_HALFWORD;
                        wr_en_rt_ins2 = 1;
                end
11'b00001111101:
                begin
                        ins2_type = EVEN;
                        opcode_ins2 = ROTATE_AND_MASK_HALFWORD_IMMEDIATE;
                        wr_en_rt_ins2 = 1;
                        I7_ins2 = ins2_r[11:17];
                        rb_addr_ins2 = 7'dx;
                end
11'b01010110100:
                begin
                        ins2_type = EVEN;
                        opcode_ins2 = COUNT_ONES_IN_BYTES;
                        wr_en_rt_ins2 = 1;
                        rb_addr_ins2 = 7'dx;
                end
```

```
11'b00011010011:
        begin
                ins2_type = EVEN;
                opcode_ins2 = AVERAGE_BYTES;
                wr_en_rt_ins2 = 1;
        end

11'b00001010011:
        begin
                ins2_type = EVEN;
                opcode_ins2 = ABSOLUTE_DIFFERENCE_OF_BYTES;
                wr_en_rt_ins2 = 1;
        end

11'b01001010011:
        begin
                ins2_type = EVEN;
                opcode_ins2 = SUM_BYTES_INTO_HALFWORDS;
                wr_en_rt_ins2 = 1;
        end

11'b01111001000:
        begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_EQUAL_HALFWORD;
                wr_en_rt_ins2 = 1;
        end

11'b01111000000:
        begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_EQUAL_WORD;
                wr_en_rt_ins2 = 1;
        end
11'b01001001000:
        begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_GREATER_THAN_HALFWORD;
```

```
                wr_en_rt_ins2 = 1;
            end
        11'b01001000000:
            begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_GREATER_THAN_WORD;
                wr_en_rt_ins2 = 1;
            end
        11'b01011000000:
            begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_LOGICAL_GREATER_THAN_WORD;
                wr_en_rt_ins2 = 1;
            end
        11'b01011001000:
            begin
                ins2_type = EVEN;
                opcode_ins2 = COMPARE_LOGICAL_GREATER_THAN_HALFWORD;
                wr_en_rt_ins2 = 1;
            end
        11'b00011000010:
            begin
                ins2_type = EVEN;
                opcode_ins2 = CARRY_GENERATE;
                wr_en_rt_ins2 = 1;
            end
        11'b01101000000:
            begin
                ins2_type = EVEN;
                opcode_ins2 = ADD_EXTENDED;
                wr_en_rt_ins2 = 1;
            end
        11'b00001000010:
            begin
                ins2_type = EVEN;
                opcode_ins2 = BORROW_GENERATE;
                wr_en_rt_ins2 = 1;
            end
```

```
11'b01101000001:
        begin
                ins2_type = EVEN;
                opcode_ins2 = SUBTRACT_FROM_EXTENDED;
                wr_en_rt_ins2 = 1;
        end

11'b00000000001:
        begin
                ins2_type = ODD;
                opcode_ins2 = NOP_LOAD;
                rt_addr_ins2 = 7'dx;
                ra_addr_ins2 = 7'dx;
                rb_addr_ins2 = 7'dx;
        end

11'b01000000001:
        begin
                ins2_type = EVEN;
                opcode_ins2 = NOP_EXEC;
                rt_addr_ins2 = 7'dx;
                ra_addr_ins2 = 7'dx;
                rb_addr_ins2 = 7'dx;
        end

11'b00000000000:
        begin
                ins2_type = ODD;
                opcode_ins2 = STOP_AND_SIGNAL;
                rt_addr_ins2 = 7'dx;
                ra_addr_ins2 = 7'dx;
                rb_addr_ins2 = 7'dx;
        end


11'b00110110000:
        begin
```

```
                ins2_type = ODD;
                opcode_ins2 = GATHER_BITS_FROM_WORDS;
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end

        11'b00110110001:
            begin
                ins2_type = ODD;
                opcode_ins2 = GATHER_BITS_FROM_HALFWORDS;
                wr_en_rt_ins2 = 1;
                rb_addr_ins2 = 7'dx;
            end


        11'b00111011011:
            begin
                ins2_type = ODD;
                opcode_ins2 = SHIFT_LEFT_QUADWORD_BY_BITS;
                wr_en_rt_ins2 = 1;
            end

        11'b00111111011:
            begin
                ins2_type = ODD;
                opcode_ins2 = SHIFT_LEFT_QUADWORD_BY_BITS_IMMEDIATE;
                wr_en_rt_ins2 = 1;
                I7_ins2 = ins2_r[11:17];
                rb_addr_ins2 = 7'dx;
            end

        11'b00111011111:
            begin
                ins2_type = ODD;
                opcode_ins2 = SHIFT_LEFT_QUADWORD_BY_BYTES;
                wr_en_rt_ins2 = 1;
            end
```

```
                                        11'b00111111111:
                                                begin
                                                        ins2_type = ODD;
                                                        opcode_ins2 = SHIFT_LEFT_QUADWORD_BY_BYTES_IMMEDIATE;
                                                        wr_en_rt_ins2 = 1;
                                                        I7_ins2 = ins2_r[11:17];
                                                        rb_addr_ins2 = 7'dx;
                                                end
                                endcase
        end
        if(pc_out[29] == 1 && flush_fetch_r) begin
                if(ins2_type == EVEN) begin
                        ins1_type = ODD;
                end
                else begin
                        ins1_type = EVEN;
                end
        end
        end
endmodule
```

d. Register Forward.sv:

```
import definitions::*;

module rf_fw(clock,reset,clear,wr_en_rt_ep, wr_en_rt_op,
flush,pc,opcode_ep,ra_addr_ep,rb_addr_ep,rc_addr_ep,rt_addr_ep,I7_ep,I10_ep,I16_ep,I18_ep,opcode_op,ra_addr_op,rb_addr_op,rc_addr_op,rt_addr_op,I7_op,
I10_op,I16_op,I18_op,

ra_rf_data_ep, rb_rf_data_ep, rc_rf_data_ep, ra_rf_data_op,rb_rf_data_op,rc_rf_data_op,opcode_ep_r,
rt_addr_ep_r,I7_ep_r,I10_ep_r,I16_ep_r,I18_ep_r,opcode_op_r,rt_addr_op_r,I7_op_r,I10_op_r,I16_op_r,I18_op_r,

fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr,fw_ep_02_data,fw_ep_03_data,fw_ep_04_data,f
w_ep_05_data,fw_ep_06_data,fw_ep_07_data,out_ep_rt_data,

fw_uid_ep_02, fw_uid_ep_03, fw_uid_ep_04, fw_uid_ep_05, fw_uid_ep_06,
fw_uid_ep_07,fw_uid_ep_08,fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr,

fw_op_02_data,fw_op_03_data,fw_op_04_data,fw_op_05_data,fw_op_06_data,fw_op_07_data,out_op_rt_data,fw_uid_op_02,fw_uid_op_03,fw_uid_op_04,fw_u
id_op_05,fw_uid_op_06,fw_uid_op_07,fw_uid_op_08,out_ep_wr_en_rt,

ra_fw_data_ep,rb_fw_data_ep,rc_fw_data_ep, ra_fw_data_op,rb_fw_data_op,rc_fw_data_op, pc_out, wr_en_rt_ep_r,
wr_en_rt_op_r,ls_wr_en_op,ls_wr_en_op_r,
```

```
wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,wr_en_op_07,out_op_wr_en_rt,wr_en_ep_01,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,
wr_en_ep_05,wr_en_ep_06,wr_en_ep_07,clear_r,

ra_addr_op_r,rb_addr_op_r,rc_addr_op_r,ra_addr_ep_r,rb_addr_ep_r,rc_addr_ep_r


);
                parameter UNIT_1 = 4'd1;
                parameter UNIT_2 = 4'd2;
                parameter UNIT_3 = 4'd3;
                parameter UNIT_4 = 4'd4;
                parameter UNIT_5 = 4'd5;
                parameter UNIT_6 = 4'd6;
                parameter UNIT_7 = 4'd7;
                parameter UNIT_8 = 4'd8;
                input flush;
                input clear;
                input clock,reset;
                input [0:31] pc; //?
                input opcode opcode_ep;
                input [0:6] ra_addr_ep;
                input [0:6] rb_addr_ep;
                input [0:6] rc_addr_ep;
                input [0:6] rt_addr_ep;
                input [0:6] I7_ep;
                input [0:9] I10_ep;
                input [0:15]I16_ep;
                input [0:17]I18_ep;
                input wr_en_rt_ep, wr_en_rt_op;
                input ls_wr_en_op;
                output logic ls_wr_en_op_r;



                input opcode opcode_op;
                input [0:6] ra_addr_op;
                input [0:6] rb_addr_op,rc_addr_op;
                //input [0:6] rc_addr_op;
                input [0:6] rt_addr_op;
                input [0:6] I7_op;
                input [0:9] I10_op;
```

```
input [0:15]I16_op;
input [0:17]I18_op;
input [0:127]ra_rf_data_ep, rb_rf_data_ep, rc_rf_data_ep, ra_rf_data_op,rb_rf_data_op,rc_rf_data_op;  //inputs from reg file

output opcode opcode_ep_r;
output logic [0:6] ra_addr_ep_r;
output logic [0:6] rb_addr_ep_r;
output logic [0:6] rc_addr_ep_r;
output logic wr_en_rt_ep_r, wr_en_rt_op_r;
output logic [0:6] rt_addr_ep_r;
output logic [0:6] I7_ep_r;
output logic [0:9] I10_ep_r;
output logic [0:15]I16_ep_r;
output logic [0:17]I18_ep_r;

output opcode opcode_op_r;
output logic [0:6] ra_addr_op_r;
output logic [0:6] rb_addr_op_r;
output logic [6:0] rc_addr_op_r;
output logic [0:6] rt_addr_op_r;
output logic [0:6] I7_op_r;
output logic [0:9] I10_op_r;
output logic [0:15]I16_op_r;
output logic [0:17]I18_op_r;
output logic clear_r;

input logic [0:6] fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr;
input logic [0:127] fw_ep_02_data,fw_ep_03_data,fw_ep_04_data,fw_ep_05_data,fw_ep_06_data,fw_ep_07_data,out_ep_rt_data;
input logic [0:3]  fw_uid_ep_02, fw_uid_ep_03, fw_uid_ep_04, fw_uid_ep_05, fw_uid_ep_06, fw_uid_ep_07,fw_uid_ep_08;

input logic [0:6] fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr;
input logic [0:127] fw_op_02_data,fw_op_03_data,fw_op_04_data,fw_op_05_data,fw_op_06_data,fw_op_07_data,out_op_rt_data;
input logic [0:3] fw_uid_op_02,fw_uid_op_03,fw_uid_op_04,fw_uid_op_05,fw_uid_op_06,fw_uid_op_07,fw_uid_op_08;

input  logic wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,wr_en_op_07,out_op_wr_en_rt;
input  logic wr_en_ep_01,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep_05,wr_en_ep_06,wr_en_ep_07,out_ep_wr_en_rt;

output logic [0:127] ra_fw_data_ep,rb_fw_data_ep,rc_fw_data_ep, ra_fw_data_op,rb_fw_data_op,rc_fw_data_op;
```

```
output logic [0:31] pc_out;
logic flush_r;

always_ff @(posedge clock) begin
        //flush_r <= flush;
        if(reset == 1 || flush == 1)begin
                opcode_ep_r  <= NOP_EXEC;
                ra_addr_ep_r <= 0;
    rb_addr_ep_r <= 0;
                rc_addr_ep_r <= 0;
                rt_addr_ep_r <= 0;
                I7_ep_r                 <= 0;
                I10_ep_r    <= 0;
                I16_ep_r    <= 0;
                I18_ep_r    <= 0;

                //opcode_op_r  <= NOP_EXEC; //Change
                opcode_op_r  <= NOP_LOAD;
                ra_addr_op_r <= 0;
    rb_addr_op_r <= 0;
                rc_addr_op_r <= 0;
                rt_addr_op_r <= 0;
                I7_op_r                 <= 0;
                I10_op_r    <= 0;
                I16_op_r    <= 0;
                I18_op_r    <= 0;
                pc_out                  <= 0;
                wr_en_rt_ep_r<= 0;
                wr_en_rt_op_r<= 0;
                ls_wr_en_op_r<= 0;
                clear_r     <= 0;

        end
        else begin
                opcode_ep_r  <= opcode_ep;
                ra_addr_ep_r <= ra_addr_ep;
    rb_addr_ep_r <= rb_addr_ep;
                rc_addr_ep_r <= rc_addr_ep;
```

```
                rt_addr_ep_r <= rt_addr_ep;
                I7_ep_r                 <= I7_ep;
                I10_ep_r    <= I10_ep;
                I16_ep_r    <= I16_ep;
                I18_ep_r    <= I18_ep;

                opcode_op_r  <= opcode_op;
                ra_addr_op_r <= ra_addr_op;
        rb_addr_op_r <= rb_addr_op;
        rc_addr_op_r <= rc_addr_op;
                rt_addr_op_r <= rt_addr_op;
                I7_op_r                 <= I7_op;
                I10_op_r    <= I10_op;
                I16_op_r    <= I16_op;
                I18_op_r    <= I18_op;
                pc_out                  <= pc;
                wr_en_rt_ep_r<= wr_en_rt_ep;
                wr_en_rt_op_r<= wr_en_rt_op;
                ls_wr_en_op_r<= ls_wr_en_op;
                clear_r     <= clear;

            end



        end

        always_comb begin

            if                  ((ra_addr_ep_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) ra_fw_data_ep =
fw_ep_03_data;
            else if ((ra_addr_ep_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) &&
wr_en_ep_04 == 1) ra_fw_data_ep = fw_ep_04_data;
            else if ((ra_addr_ep_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) &&
wr_en_ep_05 == 1) ra_fw_data_ep = fw_ep_05_data;
            else if ((ra_addr_ep_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) &&
wr_en_ep_06 == 1) ra_fw_data_ep = fw_ep_06_data;
            else if ((ra_addr_ep_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 ||
fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  ra_fw_data_ep = fw_ep_07_data;
```

```
                else if ((ra_addr_ep_r == out_ep_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 ||
fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  ra_fw_data_ep = out_ep_rt_data;

                else if ((ra_addr_ep_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) ra_fw_data_ep = fw_op_04_data;

                else if ((ra_addr_ep_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) ra_fw_data_ep = fw_op_05_data;

                else if ((ra_addr_ep_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) ra_fw_data_ep = fw_op_06_data;

                else if ((ra_addr_ep_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1)
ra_fw_data_ep = fw_op_07_data;

                else if ((ra_addr_ep_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1)
ra_fw_data_ep = out_op_rt_data;

                else ra_fw_data_ep = ra_rf_data_ep;




                if                      ((rb_addr_ep_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) rb_fw_data_ep =
fw_ep_03_data;

                else if ((rb_addr_ep_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) &&
wr_en_ep_04 == 1) rb_fw_data_ep = fw_ep_04_data;

                else if ((rb_addr_ep_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) &&
wr_en_ep_05 == 1) rb_fw_data_ep = fw_ep_05_data;

                else if ((rb_addr_ep_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) &&
wr_en_ep_06 == 1) rb_fw_data_ep = fw_ep_06_data;

                else if ((rb_addr_ep_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 ||
fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  rb_fw_data_ep = fw_ep_07_data;

                else if ((rb_addr_ep_r == out_ep_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 ||
fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  rb_fw_data_ep = out_ep_rt_data;

                else if ((rb_addr_ep_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) rb_fw_data_ep = fw_op_04_data;

                else if ((rb_addr_ep_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) rb_fw_data_ep = fw_op_05_data;

                else if ((rb_addr_ep_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) rb_fw_data_ep = fw_op_06_data;

                else if ((rb_addr_ep_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1)
rb_fw_data_ep = fw_op_07_data;

                else if ((rb_addr_ep_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1)
rb_fw_data_ep = out_op_rt_data;

                else rb_fw_data_ep = rb_rf_data_ep;



                if                      ((rc_addr_ep_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) rc_fw_data_ep =
fw_ep_03_data;

                else if ((rc_addr_ep_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) &&
wr_en_ep_04 == 1) rc_fw_data_ep = fw_ep_04_data;

                else if ((rc_addr_ep_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) &&
wr_en_ep_05 == 1) rc_fw_data_ep = fw_ep_05_data;

                else if ((rc_addr_ep_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) &&
wr_en_ep_06 == 1) rc_fw_data_ep = fw_ep_06_data;

                else if ((rc_addr_ep_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 ||
fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  rc_fw_data_ep = fw_ep_07_data;

                else if ((rc_addr_ep_r == out_ep_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 ||
fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  rc_fw_data_ep = out_ep_rt_data;

                else if ((rc_addr_ep_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) rc_fw_data_ep = fw_op_04_data;
```

```
                else if ((rc_addr_ep_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) rc_fw_data_ep = fw_op_05_data;

                else if ((rc_addr_ep_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) rc_fw_data_ep = fw_op_06_data;

                else if ((rc_addr_ep_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1)
rc_fw_data_ep = fw_op_07_data;

                else if ((rc_addr_ep_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1)
rc_fw_data_ep = out_op_rt_data;

                else rc_fw_data_ep = rc_rf_data_ep;



                if                      ((ra_addr_op_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) ra_fw_data_op =
fw_ep_03_data;

                else if ((ra_addr_op_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) &&
wr_en_ep_04 == 1) ra_fw_data_op = fw_ep_04_data;

                else if ((ra_addr_op_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) &&
wr_en_ep_05 == 1) ra_fw_data_op = fw_ep_05_data;

                else if ((ra_addr_op_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) &&
wr_en_ep_06 == 1) ra_fw_data_op = fw_ep_06_data;

                else if ((ra_addr_op_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 ||
fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  ra_fw_data_op = fw_ep_07_data;

                else if ((ra_addr_op_r == out_op_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 ||
fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  ra_fw_data_op = out_op_rt_data;

                else if ((ra_addr_op_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) ra_fw_data_op = fw_op_04_data;

                else if ((ra_addr_op_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) ra_fw_data_op = fw_op_05_data;

                else if ((ra_addr_op_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) ra_fw_data_op = fw_op_06_data;

                else if ((ra_addr_op_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1)
ra_fw_data_op = fw_op_07_data;

                else if ((ra_addr_op_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1)
ra_fw_data_op = out_op_rt_data;

                else ra_fw_data_op = ra_rf_data_op;



                if                      ((rb_addr_op_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) rb_fw_data_op =
fw_ep_03_data;

                else if ((rb_addr_op_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) &&
wr_en_ep_04 == 1) rb_fw_data_op = fw_ep_04_data;

                else if ((rb_addr_op_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) &&
wr_en_ep_05 == 1) rb_fw_data_op = fw_ep_05_data;

                else if ((rb_addr_op_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) &&
wr_en_ep_06 == 1) rb_fw_data_op = fw_ep_06_data;

                else if ((rb_addr_op_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 ||
fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  rb_fw_data_op = fw_ep_07_data;

                else if ((rb_addr_op_r == out_op_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 ||
fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  rb_fw_data_op = out_op_rt_data;

                else if ((rb_addr_op_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) rb_fw_data_op = fw_op_04_data;

                else if ((rb_addr_op_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) rb_fw_data_op = fw_op_05_data;

                else if ((rb_addr_op_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) rb_fw_data_op = fw_op_06_data;
```

```
                else if ((rb_addr_op_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1) rb_fw_data_op = fw_op_07_data;

                else if ((rb_addr_op_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1) rb_fw_data_op = out_op_rt_data;

                else rb_fw_data_op = rb_rf_data_op;



                if                ((rc_addr_op_r == fw_ep_03_addr) && (fw_uid_ep_03 == UNIT_1) && wr_en_ep_03 == 1) rc_fw_data_op = fw_ep_03_data;

                else if ((rc_addr_op_r == fw_ep_04_addr) && ((fw_uid_ep_04 == UNIT_1) || fw_uid_ep_04 == UNIT_2 || fw_uid_ep_04 == UNIT_5) && wr_en_ep_04 == 1) rc_fw_data_op = fw_ep_04_data;

                else if ((rc_addr_op_r == fw_ep_05_addr) && ((fw_uid_ep_05 == UNIT_1) || fw_uid_ep_05 == UNIT_2 || fw_uid_ep_05 == UNIT_5) && wr_en_ep_05 == 1) rc_fw_data_op = fw_ep_05_data;

                else if ((rc_addr_op_r == fw_ep_06_addr) && ((fw_uid_ep_06 == UNIT_1) || fw_uid_ep_06 == UNIT_2 || fw_uid_ep_06 == UNIT_5) && wr_en_ep_06 == 1) rc_fw_data_op = fw_ep_06_data;

                else if ((rc_addr_op_r == fw_ep_07_addr) && ((fw_uid_ep_07 == UNIT_1) || fw_uid_ep_07 == UNIT_2 || fw_uid_ep_07 == UNIT_3 || fw_uid_ep_07 == UNIT_5) && wr_en_ep_07 == 1)  rc_fw_data_op = fw_ep_07_data;

                else if ((rc_addr_op_r == out_op_rt_addr) && ((fw_uid_ep_08 == UNIT_1) || fw_uid_ep_08 == UNIT_2 || fw_uid_ep_08 == UNIT_3 || fw_uid_ep_08 == UNIT_5 || fw_uid_ep_08 == UNIT_4) && out_ep_wr_en_rt == 1)  rc_fw_data_op = out_op_rt_data;

                else if ((rc_addr_op_r == fw_op_04_addr)  && (fw_uid_op_04 == UNIT_6) && wr_en_op_04 == 1) rc_fw_data_op = fw_op_04_data;

                else if ((rc_addr_op_r == fw_op_05_addr)  && (fw_uid_op_05 == UNIT_6) && wr_en_op_05 == 1) rc_fw_data_op = fw_op_05_data;

                else if ((rc_addr_op_r == fw_op_06_addr)  && (fw_uid_op_06 == UNIT_6) && wr_en_op_06 == 1) rc_fw_data_op = fw_op_06_data;

                else if ((rc_addr_op_r == fw_op_07_addr)  && (fw_uid_op_07 == UNIT_6 || fw_uid_op_07 == UNIT_7) && wr_en_op_07 == 1) rc_fw_data_op = fw_op_07_data;

                else if ((rc_addr_op_r == out_op_rt_addr)  && (fw_uid_op_08 == UNIT_6 || fw_uid_op_08 == UNIT_7) && out_op_wr_en_rt == 1) rc_fw_data_op = out_op_rt_data;

                else rc_fw_data_op = rc_rf_data_op;



        end


endmodule
```

e. Register_File.sv:

```
module reg_file(clock,reset,ra_addr_ep,rb_addr_ep,rc_addr_ep,ra_addr_op,rb_addr_op,rc_addr_op,rt_addr_ep,rt_addr_op,rt_data_ep,rt_data_op,wrbe_ep, wrbe_op,o_ra_data_ep,o_rb_data_ep,o_rc_data_ep,o_ra_data_op,o_rb_data_op,o_rc_data_op,reg_file_mem);

        input clock,reset;

        input [6:0] ra_addr_ep;

        input [6:0] rb_addr_ep;

        input [6:0] rc_addr_ep;

        input [6:0] rt_addr_ep;
```

```
        input [6:0] ra_addr_op;
        input [6:0] rb_addr_op;
        input [6:0] rc_addr_op;
        input [6:0] rt_addr_op;
        input [127:0] rt_data_ep;
        input [127:0] rt_data_op;
        input wrbe_ep, wrbe_op;
        output logic [127:0] o_ra_data_ep;
        output logic [127:0] o_rb_data_ep;
        output logic [127:0] o_rc_data_ep;
        output logic [127:0] o_ra_data_op;
        output logic [127:0] o_rb_data_op;
        output logic [127:0] o_rc_data_op;

        output logic [0:127] reg_file_mem [128];



        assign o_ra_data_ep   = reg_file_mem[ra_addr_ep];
        assign        o_rb_data_ep  = reg_file_mem[rb_addr_ep];
        assign        o_rc_data_ep  = reg_file_mem[rc_addr_ep];
        assign        o_ra_data_op  = reg_file_mem[ra_addr_op];
        assign        o_rb_data_op  = reg_file_mem[rb_addr_op];
        assign        o_rc_data_op  = reg_file_mem[rc_addr_op];
        always_ff @(posedge clock) begin
                        if(reset) begin
  for(int i = 0; i < 128; i++) begin
    reg_file_mem[i] <= 128'd0;
  end
end
                        //$display("ra read =%d address =%d ",o_ra_data_op,ra_addr_op);
                        /*o_ra_data_ep <= reg_file_mem[ra_addr_ep];
                        o_rb_data_ep <= reg_file_mem[rb_addr_ep];
                        o_rc_data_ep <= reg_file_mem[rc_addr_ep];
                        o_ra_data_op <= reg_file_mem[ra_addr_op];
                        o_rb_data_op <= reg_file_mem[rb_addr_op];
                        o_rc_data_op <= reg_file_mem[rc_addr_op];*/
                        if(wrbe_ep == 1) begin
```

```
                        reg_file_mem[rt_addr_ep] <= rt_data_ep;
                end
                if(wrbe_op == 1) begin
                        reg_file_mem[rt_addr_op] <= rt_data_op;  // if both wrbe signals are 1 & rt_addr is same then rt_data_op is written to
rt_addr address
                end
        end
endmodule
```

f. Even_pipe.sv:

```
import definitions::*;

module even_pipe(clock,reset,clear_in,rt_addr_in, flush_in,opcode_ep_in,ra_in,rb_in,rc_in,I7_in,I10_in,I16_in,I18_in,wr_en_rt_in,
rt_addr,fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr,
fw_ep_02_data,fw_ep_03_data,fw_ep_04_data,fw_ep_05_data,fw_ep_06_data,fw_ep_07_data, out_ep_rt_data,out_ep_wr_en_rt,
uid_ep_02, uid_ep_03, uid_ep_04, uid_ep_05, uid_ep_06, uid_ep_07,uid_ep_08,
fw_ep_01_lat,fw_ep_02_lat,fw_ep_03_lat,fw_ep_04_lat,fw_ep_05_lat,fw_ep_06_lat,fw_ep_07_lat,wr_en_rt,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep
_05,wr_en_ep_06,wr_en_ep_07
);
        input flush_in;
        input clock,reset;
        input clear_in;
        input [0:6] rt_addr_in;
        input opcode opcode_ep_in;
        input [0:127] ra_in,rb_in,rc_in;
        input [0:6] I7_in;
        input [0:9] I10_in;
        input [0:15]I16_in;
        input [0:17]I18_in;
        input wr_en_rt_in;

        logic clear;
        output logic [0:6] rt_addr;
        opcode opcode_ep;
        logic [0:127] ra,rb,rc;
        logic [0:6] I7;
        logic [0:9] I10;
        logic [0:15]I16;
        logic [0:17]I18;
        output logic wr_en_rt;
```

```
logic [0:WORD-1] rep_left_Bit_32_I7;
logic [0:HALF_WORD-1] rep_left_Bit_16_I7;
logic [0:WORD-1] rep_left_Bit_32_I16;
logic [0:WORD-1] rep_left_Bit_32_I10;
logic [0:HALF_WORD-1] rep_left_Bit_16_I10;
logic [0:127] rt_data;
logic [0:3] uid;
//input wrbe[127:0];*/
logic [0:6] fw_ep_01_addr;
output logic [0:6] fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr;
logic [0:127] fw_ep_01_data;
output logic [0:127] fw_ep_02_data,fw_ep_03_data,fw_ep_04_data,fw_ep_05_data,fw_ep_06_data,fw_ep_07_data, out_ep_rt_data;
output logic wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep_05,wr_en_ep_06,wr_en_ep_07;
output logic out_ep_wr_en_rt;
output logic [0:3] fw_ep_01_lat,fw_ep_02_lat,fw_ep_03_lat,fw_ep_04_lat,fw_ep_05_lat,fw_ep_06_lat,fw_ep_07_lat;
//output logic [0:3] out_ep_unit_lat;
logic [0:3] uid_ep_01;
output logic [0:3] uid_ep_02, uid_ep_03, uid_ep_04, uid_ep_05, uid_ep_06, uid_ep_07,uid_ep_08;
//output logic [0:3] out_ep_uid;
logic [0:3] unit_lat;



logic temp_1;
logic [0:31] temp_32, temp_32_1;
logic [0:15] temp_16, temp_16_1;
logic [0:3] temp_4;
logic [0:7] temp_8,temp_8_1;
assign rep_left_Bit_32_I16 = {{16{I16[0]}}, I16};
assign rep_left_Bit_32_I10 = {{22{I10[0]}}, I10};
assign rep_left_Bit_16_I10 = {{6{I10[0]}}, I10};
assign rep_left_Bit_16_I7  = {{9{I7[0]}}, I7};
assign rep_left_Bit_32_I7  = {{25{I7[0]}}, I7};

real temp1_32_real,temp2_32_real,temp3_32_real,temp4_32_real;

int s;
```

```
always_ff @(posedge clock) begin

        rt_addr   <= rt_addr_in;
        opcode_ep <= opcode_ep_in;
        ra                      <= ra_in;
        rb              <= rb_in;
        rc                      <= rc_in;
        I7      <= I7_in;
        I10     <= I10_in;
        I16     <= I16_in;
        I18                     <= I18_in;
        wr_en_rt  <= wr_en_rt_in;
        clear    <= clear_in;

        // Clear stage 1 when flush occurs
        if(flush_in == 1) begin
        rt_addr   <= 0;
        opcode_ep <= NOP_EXEC;
        ra                      <= 0;
        rb              <= 0;
        rc                      <= 0;
        I7      <= 0;
        I10     <= 0;
        I16     <= 0;
        I18                     <= 0;
        wr_en_rt  <= 0;
        clear    <= 0;
        end

end


always_ff @(posedge clock) begin
        if(reset == 1)begin

                uid_ep_02    <= 0;
                uid_ep_03    <= 0;
```

```
uid_ep_04     <= 0;
uid_ep_05     <= 0;
uid_ep_06     <= 0;
uid_ep_07     <= 0;
uid_ep_08     <= 0;



fw_ep_02_data <= 0;
fw_ep_03_data <= 0;
fw_ep_04_data <= 0;
fw_ep_05_data <= 0;
fw_ep_06_data <= 0;
fw_ep_07_data <= 0;
out_ep_rt_data   <= 0;



wr_en_ep_02                <= 0;
wr_en_ep_03                <= 0;
wr_en_ep_04                <= 0;
wr_en_ep_05                <= 0;
wr_en_ep_06                <= 0;
wr_en_ep_07                <= 0;
out_ep_wr_en_rt  <= 0;



fw_ep_02_lat  <= 0;
fw_ep_03_lat  <= 0;
fw_ep_04_lat  <= 0;
fw_ep_05_lat  <= 0;
fw_ep_06_lat  <= 0;
fw_ep_07_lat  <= 0;



fw_ep_02_addr <= 0;
fw_ep_03_addr <= 0;
fw_ep_04_addr <= 0;
fw_ep_05_addr <= 0;
```

```
                fw_ep_06_addr <= 0;
                fw_ep_07_addr <= 0;
                out_ep_rt_addr   <= 0;

end
else if(flush_in == 1 && clear == 1) begin

                uid_ep_02     <= 0;
                uid_ep_03     <= uid_ep_02;
                uid_ep_04     <= uid_ep_03;
                uid_ep_05     <= uid_ep_04;
                uid_ep_06     <= uid_ep_05;
                uid_ep_07     <= uid_ep_06;
                uid_ep_08     <= uid_ep_07;



                fw_ep_02_data <= 0;
                fw_ep_03_data <= fw_ep_02_data;
                fw_ep_04_data <= fw_ep_03_data;
                fw_ep_05_data <= fw_ep_04_data;
                fw_ep_06_data <= fw_ep_05_data;
                fw_ep_07_data <= fw_ep_06_data;
                out_ep_rt_data   <= fw_ep_07_data;



                wr_en_ep_02                <= 0;
                wr_en_ep_03                <= wr_en_ep_02;
                wr_en_ep_04                <= wr_en_ep_03;
                wr_en_ep_05                <= wr_en_ep_04;
                wr_en_ep_06                <= wr_en_ep_05;
                wr_en_ep_07                <= wr_en_ep_06;
                out_ep_wr_en_rt  <= wr_en_ep_07;



                fw_ep_02_lat  <= 0;
                fw_ep_03_lat  <= fw_ep_02_lat;
                fw_ep_04_lat  <= fw_ep_03_lat;
```

```
                fw_ep_05_lat  <= fw_ep_04_lat;
                fw_ep_06_lat  <= fw_ep_05_lat;
                fw_ep_07_lat  <= fw_ep_06_lat;



                fw_ep_02_addr <= 0;
                fw_ep_03_addr <= fw_ep_02_addr;
                fw_ep_04_addr <= fw_ep_03_addr;
                fw_ep_05_addr <= fw_ep_04_addr;
                fw_ep_06_addr <= fw_ep_05_addr;
                fw_ep_07_addr <= fw_ep_06_addr;
                out_ep_rt_addr   <= fw_ep_07_addr;
end
else begin

                uid_ep_02     <= uid;
                uid_ep_03     <= uid_ep_02;
                uid_ep_04     <= uid_ep_03;
                uid_ep_05     <= uid_ep_04;
                uid_ep_06     <= uid_ep_05;
                uid_ep_07     <= uid_ep_06;
                uid_ep_08     <= uid_ep_07;



                fw_ep_02_data <= rt_data;
                fw_ep_03_data <= fw_ep_02_data;
                fw_ep_04_data <= fw_ep_03_data;
                fw_ep_05_data <= fw_ep_04_data;
                fw_ep_06_data <= fw_ep_05_data;
                fw_ep_07_data <= fw_ep_06_data;
                out_ep_rt_data   <= fw_ep_07_data;



                wr_en_ep_02                <= wr_en_rt;
                wr_en_ep_03                <= wr_en_ep_02;
                wr_en_ep_04                <= wr_en_ep_03;
```

```
                wr_en_ep_05             <= wr_en_ep_04;
                wr_en_ep_06             <= wr_en_ep_05;
                wr_en_ep_07             <= wr_en_ep_06;
                out_ep_wr_en_rt  <= wr_en_ep_07;



                fw_ep_02_lat  <= unit_lat;
                fw_ep_03_lat  <= fw_ep_02_lat;
                fw_ep_04_lat  <= fw_ep_03_lat;
                fw_ep_05_lat  <= fw_ep_04_lat;
                fw_ep_06_lat  <= fw_ep_05_lat;
                fw_ep_07_lat  <= fw_ep_06_lat;



                fw_ep_02_addr <= rt_addr;
                fw_ep_03_addr <= fw_ep_02_addr;
                fw_ep_04_addr <= fw_ep_03_addr;
                fw_ep_05_addr <= fw_ep_04_addr;
                fw_ep_06_addr <= fw_ep_05_addr;
                fw_ep_07_addr <= fw_ep_06_addr;
                out_ep_rt_addr   <= fw_ep_07_addr;
        end
    end


    always_comb begin
        case(opcode_ep)
            ADD_HALFWORD:
                begin
                    for(int i=0; i < 8; i++) begin
                        rt_data[i*HALF_WORD +: HALF_WORD] = ra[i*HALF_WORD +: HALF_WORD] +
rb[i*HALF_WORD +: HALF_WORD];
                    end
                    unit_lat = 4'd3;
                    uid = 4'd1;
                    $display($time,"Inside ADD_HALFWORD");
                end
```

```
ADD_HALFWORD_IMMEDIATE:
        begin
            for(int i=0; i < 8; i++) begin
                rt_data[i*HALF_WORD +: HALF_WORD] = ra[i*HALF_WORD +: HALF_WORD] +
rep_left_Bit_16_I10 ;
            end
            unit_lat = 4'd3;
            uid = 4'd1;
            $display($time,"Inside ADD_HALFWORD_IMMEDIATE");
        end
ADD_WORD:
        begin
        $display($time,"Inside ADD_WORD");
            for(int i=0; i < 4; i++) begin
                rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] + rb[i*WORD +: WORD];
            end
            unit_lat = 4'd3;
            uid = 4'd1;
        end
ADD_WORD_IMMEDIATE:
        begin
            for(int i=0; i < 4; i++) begin
                rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] + rep_left_Bit_32_I10 ;
            end
            unit_lat = 4'd3;
            uid = 4'd1;
        end
SUB_FROM_HALFWORD:
        begin
        $display($time,"Inside SUB_FROM_HALFWORD");
            for(int i=0; i < 8; i++) begin
                rt_data[i*HALF_WORD +: HALF_WORD] = rb[i*HALF_WORD +: HALF_WORD] +
~(ra[i*HALF_WORD +: HALF_WORD]) + 1;
                $display($time,"Result = %d",rt_data[i*HALF_WORD +: HALF_WORD]);

            end
            unit_lat = 4'd3;
            uid = 4'd1;
```

```
                                        end
                                SUB_FROM_HALFWORD_IMMEDIATE://new
        begin
                                        $display($time,"Inside SUB_FROM_HALFWORD_IMMEDIATE");
          for(int i=0; i < 8; i++) begin
            rt_data[i*HALF_WORD +: HALF_WORD] = rep_left_Bit_16_I10 + ~(ra[i*HALF_WORD +: HALF_WORD]) + 1;
                                                $display($time,"Result = %d",rt_data[i*HALF_WORD +: HALF_WORD]);
          end
          unit_lat = 4'd3;
          uid = 4'd1;
        end
                                SUB_FROM_WORD: //new change lah
                                        begin
                                                for(int i=0; i < 4; i++) begin
                                                        //rt_data[i*WORD +: WORD] = rb[i*WORD +: WORD] + ~(ra[i*WORD +: WORD]) + 1;
                                                        //$display($time,"rb[i*WORD +: WORD] = %d (ra[i*WORD +: WORD]) =%d",rb[i*WORD
+: WORD] , ra[i*WORD +: WORD]);
                                                        rt_data[i*WORD +: WORD] = rb[i*WORD +: WORD] - ra[i*WORD +: WORD];
                                                        //$display($time,"Result = %d",rt_data[i*WORD +: WORD]);

                                                end
                                                unit_lat = 4'd3;
                                                uid = 4'd1;
                                        end
                                SUB_FROM_WORD_IMMEDIATE://new
         begin
                                        $display($time,"Inside SUB_FROM_WORD_IMMEDIATE");
          for(int i=0; i < 4; i++) begin
                                                $display($time,"rep_left_Bit_32_I10 = %d (ra[i*WORD +: WORD]) =%d",
rep_left_Bit_32_I10, ra[i*WORD +: WORD]);
            //rt_data[i*WORD +: WORD] = rep_left_Bit_32_I10 + ~(ra[i*WORD +: WORD]) + 1;
                                                rt_data[i*WORD +: WORD] = rep_left_Bit_32_I10 - ra[i*WORD +: WORD];
                                                $display($time,"Result = %d",rt_data[i*WORD +: WORD]);
          end
          unit_lat = 4'd3;
                                                uid = 4'd1;
        end

                                MULTIPLY: //check
```

```
        begin
                                                $display($time,"MULTIPLY");
          for(int i=0; i < 4; i++) begin
            rt_data[i*WORD +: WORD] = $signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]) * $signed(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]);
                                                                $display($time,"Result = %d  op1 =%d op2 =%d ",rt_data[i*WORD +:
WORD],$signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]),$signed(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]));
          end
          uid = 4'd4;
                                                        unit_lat = 4'd8;
        end
                                        MULITPLY_UNSIGNED:
                                                begin
                                                $display($time,"Inside MULITPLY_UNSIGNED");
          for(int i=0; i < 4; i++) begin
            rt_data[i*WORD +: WORD] = $unsigned(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]) * $unsigned(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]);
                                                                $display($time,"Result = %d  op1 =%d op2 =%d ",rt_data[i*WORD +:
WORD],$unsigned(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]),$unsigned(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]));
                                                        end


          uid = 4'd4;
                                                        unit_lat = 4'd8;
        end
      MULTIPLY_IMMEDIATE:
        begin
                                                $display($time,"Inside MULTIPLY_IMMEDIATE");
          for(int i=0; i < 4; i++) begin
            rt_data[i*WORD +: WORD] = $signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]) * $signed(rep_left_Bit_16_I10);
                                                                $display($time,"Result = %d  op1 =%d op2 =%d ",rt_data[i*WORD +:
WORD],$signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]),$signed(rep_left_Bit_16_I10));
          end
          uid = 4'd4;
                                                        unit_lat = 4'd8;
        end


      MULTIPLY_AND_ADD:
        begin
                                                $display($time,"MULTIPLY_AND_ADD");
          for(int i=0; i < 4; i++) begin
```

```
            rt_data[i*WORD +: WORD] = $signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]) * $signed(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]) + $signed(rc[i*WORD +: WORD]);

                                                                $display($time,"Result = %d  op1 =%d op2 =%d op3 =%d",rt_data[i*WORD +: WORD],$signed(ra[((i*WORD)+HALF_WORD) +: HALF_WORD]),$signed(rb[((i*WORD)+HALF_WORD) +: HALF_WORD]),rc[i*WORD +: WORD] );

          end

          uid = 4'd4;

                                                        unit_lat = 4'd8;

        end

                                    COUNTING_LEADING_ZEROS:

                                                begin

          for(int i=0; i < 4; i++) begin

            temp_1 = 'd0;

            temp_32 = ra[i*WORD +: WORD];

            for(int j=0; j < WORD; j++) begin

              if(temp_32[j] == 1 && temp_1 == 'd0) begin

                temp_1 = 'd1;

                rt_data[i*WORD +: WORD] = j;

              end

            end

            if(temp_1 == 'd0) rt_data[i*WORD +: WORD] = 32;

          end

          uid = 4'd1;

                                                        unit_lat = 4'd3;

                                                end

                                    FORM_SELECT_MASK_BYTES://GMS extra

                                                begin

                                                        temp_16 = ra[2*BYTE +:HALF_WORD];

                                                        for(int i=0; i < HALF_WORD; i++) begin

                                                                if(temp_16[i] == 0)begin

                                                                        rt_data[i*BYTE+:BYTE] = 8'd0;

                                                                end

                                                                else begin

                                                                        rt_data[i*BYTE+:BYTE] = 8'hff;

                                                                end

                                                        end

                                                        uid = 4'd1;

                                                        unit_lat = 4'd3;

                                                end

                                    FORM_SELECT_MASK_BYTES_IMMEDIATE://new
```

```
        begin
                for(int i=0; i < HALF_WORD; i++) begin
                        if(I16[i] == 0)begin
                                rt_data[i*BYTE+:BYTE] = 8'd0;
                        end
                        else begin
                                rt_data[i*BYTE+:BYTE] = 8'hff;
                        end
                end
                uid = 4'd1;
                unit_lat = 4'd3;
        end
FORM_SELECT_MASK_HALFWORD://GMS
        begin
                temp_8 = ra[3*BYTE +:BYTE];
                for(int i=0; i < BYTE; i++) begin
                        if(temp_8[i] == 0)begin
                                rt_data[i*HALF_WORD+:HALF_WORD] = 16'd0;
                        end
                        else begin
                                rt_data[i*HALF_WORD+:HALF_WORD] =  16'hffff;
                        end
                end
                uid = 4'd1;
                unit_lat = 4'd3;
        end
FORM_SELECT_MASK_WORD://GMS
        begin
                temp_4 = ra[28+:NIBBLE];
                for(int i=0; i < NIBBLE; i++) begin
                        if(temp_4[i] == 0)begin
                                rt_data[i*WORD+:WORD] = 32'd0;
                        end
                        else begin
                                rt_data[i*WORD+:WORD] =  32'hffffffff;
                        end
                end
                uid = 4'd1;
```

```
                unit_lat = 4'd3;
            end
        AND:
            begin
                for(int i=0; i < 4; i++) begin
                    rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] & rb[i*WORD +: WORD];
                end
                $display($time,"rt %h ra %h rb %h", rt_data,ra,rb);
                uid = 4'd1;
                unit_lat = 4'd3;
            end

        AND_HALFWORD_IMMEDIATE:
            begin
                for(int i=0; i < 8; i++) begin
                    rt_data[i*HALF_WORD +: HALF_WORD] = ra[i*HALF_WORD +: HALF_WORD] &
rep_left_Bit_16_I10 ;
                end
                uid = 4'd1;
                unit_lat = 4'd3;
            end
        AND_WORD_IMMEDIATE:
            begin
                for(int i=0; i < 4; i++) begin
                    rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] & rep_left_Bit_32_I10 ;
                end
                uid = 4'd1;
                unit_lat = 4'd3;
            end
        OR:
            begin
                for(int i=0; i < 4; i++) begin
                    rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] | rb[i*WORD +: WORD];
                end
                uid = 4'd1;
                unit_lat = 4'd3;
            end
        OR_BYTE_IMMEDIATE://New
```

```
        begin
          for(int i=0; i < 4; i++) begin
            temp_8 = I10 & 16'h00ff;
            temp_32 = {4{temp_8}};
            rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] | temp_32;
          end
          uid = 4'd1;
          unit_lat = 4'd3;
        end
                                OR_HALFWORD_IMMEDIATE:
                                        begin
                                                for(int i=0; i < 8; i++) begin
                                                        rt_data[i*HALF_WORD +: HALF_WORD] = ra[i*HALF_WORD +: HALF_WORD] |
rep_left_Bit_16_I10 ;
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                OR_WORD_IMMEDIATE:
                                        begin
                                                for(int i=0; i < 4; i++) begin
                                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] | rep_left_Bit_32_I10 ;
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                NOR:
                                        begin
                                                for(int i=0; i < 4; i++) begin
                                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] | ~(rb[i*WORD +: WORD]);
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                XOR:
                                        begin
                                                for(int i=0; i < 4; i++) begin
                                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] ^ rb[i*WORD +: WORD];
```

```
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                XOR_BYTE_IMMEDIATE://new
        begin
          for(int i=0; i < 4; i++) begin
            temp_8 = I10 & 16'h00ff;
            temp_32 = {4{temp_8}};
            rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] ^ temp_32;
          end
          uid = 4'd1;
          unit_lat = 4'd3;
        end
                                XOR_HALFWORD_IMMEDIATE:
                                        begin
                                                for(int i=0; i < 8; i++) begin
                                                        rt_data[i*HALF_WORD +: HALF_WORD] = ra[i*HALF_WORD +: HALF_WORD] ^
rep_left_Bit_16_I10 ;
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                XOR_WORD_IMMEDIATE:
                                        begin
                                                for(int i=0; i < 4; i++) begin
                                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] ^ rep_left_Bit_32_I10 ;
                                                end
                                                uid = 4'd1;
                                                unit_lat = 4'd3;
                                        end
                                 FLOATING_ADD://new
        begin
          for(int i=0; i < 4; i++) begin
            temp1_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
            temp2_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
            temp3_32_real = temp1_32_real + temp2_32_real;
            if (temp3_32_real < -SMAX)                  rt_data[i*WORD +: WORD] = -$shortrealtobits(SMAX);
```

```
            else if (temp3_32_real > SMAX)                rt_data[i*WORD +: WORD] =  $shortrealtobits(SMAX);
            else if (temp3_32_real > -SMIN && temp3_32_real < SMIN) rt_data[i*WORD +: WORD] =  0;
            else                                rt_data[i*WORD +: WORD] =  $shortrealtobits(temp3_32_real);
          end
          uid = 4'd3;
                                                        unit_lat = 4'd7;
        end

      FLOATING_SUBTRACT://new
        begin
          for(int i=0; i < 4; i++) begin
            temp1_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
            temp2_32_real = $bitstoshortreal(rb[i*WORD +: WORD]);
            temp3_32_real = temp1_32_real - temp2_32_real;
            if (temp3_32_real < -SMAX)                  rt_data[i*WORD +: WORD] = -$shortrealtobits(SMAX);
            else if (temp3_32_real > SMAX)                rt_data[i*WORD +: WORD] =  $shortrealtobits(SMAX);
            else if (temp3_32_real > -SMIN && temp3_32_real < SMIN)  rt_data[i*WORD +: WORD] =  0;
            else                                rt_data[i*WORD +: WORD] =  $shortrealtobits(temp3_32_real);
          end
          uid = 4'd3;
                                                        unit_lat = 4'd7;
        end

                                    FLOATING_MULTIPLY:
        begin
                                                $display($time,"FLOATING_MULTIPLY -SMAX = %d SMAX = %d \n -SMIN = %d SMIN %d",-SMAX,SMAX,-
SMIN,SMIN);
          for(int i=0; i < 4; i++) begin
                                                        temp1_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
                                                        temp2_32_real = $bitstoshortreal(rb[i*WORD +: WORD]);
                                                        temp3_32_real = temp1_32_real * temp2_32_real;
                                                        if (temp3_32_real < -SMAX)                                                  rt_data[i*WORD +:
WORD] = -$shortrealtobits(SMAX);
            else if (temp3_32_real > SMAX)                          rt_data[i*WORD +: WORD] =  $shortrealtobits(SMAX);
            else if (temp3_32_real > -SMIN && temp3_32_real < SMIN) rt_data[i*WORD +: WORD] =  0;
            else                                                    rt_data[i*WORD +: WORD] =  $shortrealtobits(temp3_32_real);
          uid = 4'd3;
                                                        unit_lat = 4'd7;
                                                        end
```

```
        end
                                FLOATING_MULTIPLY_AND_ADD:
                                        begin
          for(int i=0; i < 4; i++) begin
                                                temp1_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
                                                temp2_32_real = $bitstoshortreal(rb[i*WORD +: WORD]);
                                                temp3_32_real = $bitstoshortreal(rc[i*WORD +: WORD]);
                                                temp4_32_real = temp1_32_real * temp2_32_real + temp3_32_real;
                                                $display(" temp1_32_real %d temp2_32_real %d temp3_32_real
%d",temp1_32_real,temp2_32_real,temp3_32_real);

                                                if (temp4_32_real < -SMAX)                                      rt_data[i*WORD +:
WORD] = -$shortrealtobits(SMAX);
            else if (temp4_32_real > SMAX)                      rt_data[i*WORD +: WORD] =  $shortrealtobits(SMAX);
            else if (temp4_32_real > -SMIN && temp4_32_real < SMIN) rt_data[i*WORD +: WORD] =  0;
            else                                                rt_data[i*WORD +: WORD] =  $shortrealtobits(temp4_32_real);
          uid = 4'd3;
                                                unit_lat = 4'd7;
                                                end
        end
                                FLOATING_MULTIPLY_AND_SUBTRACT:
                                        begin
          for(int i=0; i < 4; i++) begin
                                                temp1_32_real = $bitstoshortreal(ra[i*WORD +: WORD]);
                                                temp2_32_real = $bitstoshortreal(rb[i*WORD +: WORD]);
                                                temp3_32_real = $bitstoshortreal(rc[i*WORD +: WORD]);
                                                temp4_32_real = temp1_32_real * temp2_32_real - temp3_32_real;
                                                if (temp4_32_real < -SMAX)                                      rt_data[i*WORD +:
WORD] = -$shortrealtobits(SMAX);
            else if (temp4_32_real > SMAX)                      rt_data[i*WORD +: WORD] =  $shortrealtobits(SMAX);
            else if (temp4_32_real > -SMIN && temp4_32_real < SMIN) rt_data[i*WORD +: WORD] =  0;
            else                                                rt_data[i*WORD +: WORD] =  $shortrealtobits(temp4_32_real);
          uid = 4'd3;
                                                unit_lat = 4'd7;
                                                end
        end
                                EQUIVALENT:
                                        begin
          for(int i=0; i < 4; i++) begin
```

```
            rt_data[i*WORD +: WORD] = ra[((i*WORD)+HALF_WORD) +: HALF_WORD] ^ (~rb[((i*WORD)+HALF_WORD) +: HALF_WORD]);
        end
        uid = 4'd1;
                                                unit_lat = 4'd3;
    end
                                SHIFT_LEFT_HALFWORD:
                                        begin
                                                for(int i=0; i < 8; i++) begin
                                                        s = rb[i*HALF_WORD +: HALF_WORD] & 16'h001F;
                                                        //$display("s value is for even instr1 : %d, rb value is %h rb3 = %b,rb4 = %b,rb5 = %b,",s,
rb[20:31],rb[3],rb[4],rb[5]);
                                                        temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                                                        for(int b=0; b < 16; b++) begin
                                                                if (b+s < 16)begin
                                                                        temp_16_1[b] = temp_16[b + s];
                                                                end
                                                                else begin
                                                                        temp_16_1[b] = 0;
                                                                end
                                                        end
                                                        rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                                                end
                                                unit_lat = 4'd4;
                                                uid = 4'd2;
                                        end
                                SHIFT_LEFT_HALFWORD_IMMEDIATE:
                                        begin
                                                s = rep_left_Bit_16_I7 & 16'h001F;
                                                for(int i=0; i < 8; i++) begin
                                                        temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                                                        //$display("temp_16 value is for even instr1 : %d",temp_16);
                                                        for(int b=0; b < 16; b++) begin
                                                                if (b+s < 16)begin
                                                                        temp_16_1[b] = temp_16[b + s];
                                                                end
                                                                else begin
                                                                        temp_16_1[b] = 0;
                                                                end
```

```
                        end
                        rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                    end
                    unit_lat = 4'd4;
                    uid = 4'd2;
                end
SHIFT_LEFT_WORD:
                begin
                    for(int i=0;i<4;i++) begin
                        s = rb[i*WORD +: WORD] & 32'h0000003F;
                        //$display("s value is for even instr1 : %d",s);
                        temp_32 = ra[i*WORD +: WORD];
                        //$display("ra value is for even instr1 : %d",ra);
                        //$display("temp_32 value is for even instr1 : %d",temp_32);
                        for(int b=0;b<32;b++) begin
                            if (b+s < 32) begin
                                temp_32_1[b] = temp_32[b + s];
                            end
                            else begin
                                temp_32_1[b] = 0;
                            end
                        end
                        rt_data[i*WORD +: WORD] = temp_32_1;
                        //$display("rt_data value is for even instr1 : %d",rt_data);
                    end
                    unit_lat = 4'd4;
                    uid = 4'd2;
                end
SHIFT_LEFT_WORD_IMMEDIATE:
                begin
                    s = rep_left_Bit_32_I7 & 32'h0000003F;
                    //$display("s value is for even instr2 : %d",s);
                    for(int i=0;i<4;i++) begin
                        temp_32 = ra[i*WORD +: WORD];
                        for(int b=0;b<32;b++) begin
                            if (b+s < 32) begin
                                temp_32_1[b] = temp_32[b + s];
                            end
```

```
                                        else begin
                                                temp_32_1[b] = 0;
                                        end
                                end
                                rt_data[i*WORD +: WORD] = temp_32_1;
                        end
                        unit_lat = 4'd4;
                        uid = 4'd2;
                end
ROTATE_HALFWORD:
                begin
                        for(int i=0;i<8;i++) begin
                                s = rb[i*HALF_WORD +: HALF_WORD] & 16'h000F;
                                temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                                for(int b=0;b<16;b++) begin
                                        if (b+s < 16) begin
                                                temp_16_1[b] = temp_16[b + s];
                                        end
                                        else begin
                                                temp_16_1[b] = temp_16[b + s - 16];
                                        end
                                end
                                rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                        end
                        unit_lat = 4'd4;
                        uid = 4'd2;
                end
ROTATE_HALFWORD_IMMEDIATE:
                begin
                        for(int i=0;i<8;i++) begin
                                s = rep_left_Bit_16_I7 & 16'h000F;
                                temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                                for(int b=0;b<16;b++) begin
                                        if (b+s < 16) begin
                                                temp_16_1[b] = temp_16[b + s];
                                        end
                                        else begin
                                                temp_16_1[b] = temp_16[b + s - 16];
```

```
                                        end
                                end
                                rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                        end
                        unit_lat = 4'd4;
                        uid = 4'd2;
                end
        ROTATE_WORD:
                begin
                        for(int i=0;i<4;i++) begin
                                s = rb[i*WORD +: WORD] & 32'h0000001F;
                                temp_32[0+:WORD] = ra[i*WORD +: WORD];
                                for(int b=0;b<32;b++) begin
                                        if (b+s < 32) begin
                                                temp_32_1[b] = temp_32[b + s];
                                        end
                                        else begin
                                                temp_32_1[b] = temp_32[b + s - 32];
                                        end
                                end
                                rt_data[i*WORD +: WORD] = temp_32_1;
                        end
                        unit_lat = 4'd4;
                        uid = 4'd2;
                end
        ROTATE_WORD_IMMEDIATE:
                begin
                        s = rep_left_Bit_32_I7 & 32'h0000001F;
                        for(int i=0;i<4;i++) begin
                                temp_32[0+:WORD] = ra[i*WORD +: WORD];
                                for(int b=0;b<32;b++) begin
                                        if (b+s < 32) begin
                                                temp_32_1[b] = temp_32[b + s];
                                        end
                                        else begin
                                                temp_32_1[b] = temp_32[b + s - 32];
                                        end
                                end
```

```
                        rt_data[i*WORD +: WORD] = temp_32_1;
                end
                unit_lat = 4'd4;
                uid = 4'd2;
        end
ROTATE_AND_MASK_HALFWORD:       //did prof ask us to remove rotate mask instr?
        begin
                for(int i=0;i<8;i++) begin
                        s = (0 - rb[i*HALF_WORD +: HALF_WORD]) & 16'h001F;
                        temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                        //$display("s value is for even rotate  instr2 : %d",s);
                        //$display("ra value is for even rotate  instr2 : %h",ra);
                        for(int b=0;b<16;b++) begin
                                if (b >= s) begin
                                        temp_16_1[b] = temp_16[b - s];
                                end
                                else begin
                                        temp_16_1[b] = 0;
                                end
                        end
                        rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                        //$display("rt value is for even rotate  instr2 : %h",rt_data);
                end
                unit_lat = 4'd4;
                uid = 4'd2;
        end
ROTATE_AND_MASK_HALFWORD_IMMEDIATE:
        begin
                for(int i=0;i<8;i++) begin
                        s = (0 - rep_left_Bit_32_I7) & 16'h001F;
                        temp_16 = ra[i*HALF_WORD +: HALF_WORD];
                        //$display("s value is for even rotate Imm instr2 : %d",s);
                        //$display("ra value is for even rotate Imm  instr2 : %h",ra);
                        for(int b=0;b<16;b++) begin
                                if (b >= s) begin
                                        temp_16_1[b] = temp_16[b - s];
                                end
                                else begin
```

```
                                                temp_16_1[b] = 0;
                                        end
                                end
                                rt_data[i*HALF_WORD +: HALF_WORD] = temp_16_1;
                                //$display("rt value is for even rotate Imm instr2 : %h",rt_data);
                        end
                        unit_lat = 4'd4;
                        uid = 4'd2;
                end
        COUNT_ONES_IN_BYTES:
                begin
                        for(int i=0;i<16;i++) begin
                                temp_8_1 = 0;
                                temp_8 = ra[i*BYTE +: BYTE];
                                for(int b=0;b<8;b++) begin
                                        if(temp_8[b] == 1) begin
                                                temp_8_1 = temp_8_1+1;
                                        end
                                end
                                rt_data[i*BYTE +: BYTE] = temp_8_1;
                        end
                        unit_lat = 4;
                        uid = 5;
                end
        AVERAGE_BYTES:
                begin
                        for(int i=0;i<16;i++) begin
                                rt_data[i*BYTE +: BYTE] = ({8'd0,ra[i*BYTE +: BYTE]} + {8'd0,rb[i*BYTE +: BYTE]} + 1);
//check the code with spu instr pdf
                        end
                        unit_lat = 4;
                        uid = 5;
                end
        ABSOLUTE_DIFFERENCE_OF_BYTES:
                begin
                        for(int i=0;i<16;i++) begin
                                if($unsigned(ra[i*BYTE +: BYTE]) > $unsigned(rb[i*BYTE +: BYTE])) begin
                                        rt_data[i*BYTE +: BYTE] = ra[i*BYTE +: BYTE] - rb[i*BYTE +: BYTE];
```

```
                                        end
                                        else begin
                                                rt_data[i*BYTE +: BYTE] = rb[i*BYTE +: BYTE] - ra[i*BYTE +: BYTE];
                                        end
                                end
                                unit_lat = 4;
                                uid = 5;
                        end
                SUM_BYTES_INTO_HALFWORDS:
                        begin
                                for(int i=0;i<4;i++) begin
                                        rt_data[2*i*HALF_WORD +: HALF_WORD] = rb[4*i*BYTE +: BYTE] + rb[(4*i + 1)*BYTE +:
BYTE] + rb[(4*i + 2)*BYTE +: BYTE] + rb[(4*i + 3)*BYTE +: BYTE];
                                        rt_data[(2*i+1)*HALF_WORD +: HALF_WORD] = ra[4*i*BYTE +: BYTE] + ra[(4*i + 1)*BYTE
+: BYTE] + ra[(4*i + 2)*BYTE +: BYTE] + ra[(4*i + 3)*BYTE +: BYTE];
                                end
                                unit_lat = 4;
                                uid = 5;
                        end
                COMPARE_EQUAL_HALFWORD:
                        begin
                                for(int i=0;i<8;i++) begin
                                        if(ra[i*HALF_WORD +: HALF_WORD] == rb[i*HALF_WORD +: HALF_WORD]) begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
                                        end
                                        else begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
                                        end
                                end
                                unit_lat = 3;
                                uid = 1;
                        end
                COMPARE_EQUAL_HALFWORD_IMMEDIATE:
                        begin
                                for(int i=0;i<8;i++) begin
                                        if(ra[i*HALF_WORD +: HALF_WORD] == rep_left_Bit_16_I10) begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
                                        end
                                        else begin
```

```
                                        rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
                                    end
                                end
                                unit_lat = 3;
                                uid = 1;
                            end
                        COMPARE_EQUAL_WORD:
                            begin
                                for(int i=0;i<4;i++) begin
                                    if(ra[i*WORD +: WORD] == rb[i*WORD +: WORD]) begin
                                        rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                    end
                                    else begin
                                        rt_data[i*WORD +: WORD] = 32'h00000000;
                                    end
                                end
                                unit_lat = 3;
                                uid = 1;
                            end
                        COMPARE_EQUAL_WORD_IMMEDIATE:
                            begin
                                for(int i=0;i<4;i++) begin
                                    if(ra[i*WORD +: WORD] == rep_left_Bit_32_I10) begin
                                        rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                    end
                                    else begin
                                        rt_data[i*WORD +: WORD] = 32'h00000000;
                                    end
                                end
                                unit_lat = 3;
                                uid = 1;
                            end
                        COMPARE_GREATER_THAN_HALFWORD:
                            begin
                                for(int i=0;i<8;i++) begin
                                    if($signed(ra[i*HALF_WORD +: HALF_WORD]) > $signed(rb[i*HALF_WORD +:
HALF_WORD])) begin
                                        rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
```

```
                                        end
                                        else begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
                                        end
                                end
                                unit_lat = 3;
                                uid = 1;
                        end
                COMPARE_GREATER_THAN_HALFWORD_IMMEDIATE:
                        begin
                                for(int i=0;i<8;i++) begin
                                        if($signed(ra[i*HALF_WORD +: HALF_WORD]) > rep_left_Bit_16_I10) begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
                                        end
                                        else begin
                                                rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
                                        end
                                end
                                unit_lat = 3;
                                uid = 1;
                        end
                COMPARE_GREATER_THAN_WORD:
                        begin
                                for(int i=0;i<4;i++) begin
                                        if($signed(ra[i*WORD +: WORD]) > $signed(rb[i*WORD +: WORD])) begin
                                                rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                        end
                                        else begin
                                                rt_data[i*WORD +: WORD] = 32'h00000000;
                                        end
                                end
                                unit_lat = 3;
                                uid = 1;
                        end
                COMPARE_GREATER_THAN_WORD_IMMEDIATE:
                        begin
                                for(int i=0;i<4;i++) begin
                                        if($signed(ra[i*WORD +: WORD]) > rep_left_Bit_32_I10) begin
```

```
                                        rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                end
                                else begin
                                        rt_data[i*WORD +: WORD] = 32'h00000000;
                                end
                        end
                        unit_lat = 3;
                        uid = 1;
                end

// additional instructions added from other than the instruction set
COMPARE_LOGICAL_GREATER_THAN_WORD:
                begin
                        for(int i=0;i<4;i++) begin
                                if(ra[i*WORD +: WORD] > rb[i*WORD +: WORD]) begin
                                        rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                end
                                else begin
                                        rt_data[i*WORD +: WORD] = 32'h00000000;
                                end
                        end
                        unit_lat = 3;
                        uid = 1;
                end
COMPARE_LOGICAL_GREATER_THAN_WORD_IMMEDIATE:
                begin
                        for(int i=0;i<4;i++) begin
                                if(ra[i*WORD +: WORD] > rep_left_Bit_32_I10) begin
                                        rt_data[i*WORD +: WORD] = 32'hFFFFFFFF;
                                end
                                else begin
                                        rt_data[i*WORD +: WORD] = 32'h00000000;
                                end
                        end
                        unit_lat = 3;
                        uid = 1;
                end
COMPARE_LOGICAL_GREATER_THAN_HALFWORD:
```

```
		begin
			for(int i=0;i<8;i++) begin
				if(ra[i*HALF_WORD +: HALF_WORD] > rb[i*HALF_WORD +: HALF_WORD]) begin
					rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
				end
				else begin
					rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
				end
			end
			unit_lat = 3;
			uid = 1;
		end
COMPARE_LOGICAL_GREATER_THAN_HALFWORD_IMMEDIATE:
		begin
			for(int i=0;i<8;i++) begin
				if(ra[i*HALF_WORD +: HALF_WORD] > rep_left_Bit_16_I10) begin
					rt_data[i*HALF_WORD +: HALF_WORD] = 16'hFFFF;
				end
				else begin
					rt_data[i*HALF_WORD +: HALF_WORD] = 16'h0000;
				end
			//$display("rt value is for even comp instr2 : %h",rt_data[i*HALF_WORD +: HALF_WORD]);
			end
			unit_lat = 3;
			uid = 1;
		end
CARRY_GENERATE:
		begin
			for(int i=0;i<4;i++) begin
				temp_32 = ra[i*WORD +: WORD] + rb[i*WORD +: WORD];
				rt_data[i*WORD +: WORD] = {31'b0,temp_32[0]};
			end
			unit_lat = 2;
			uid = 1;
		end
ADD_EXTENDED:
		begin
			for(int i=0;i<4;i++) begin
```

```
                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] + rb[i*WORD +: WORD] + rc[31 +
i*WORD];
                                end
                                unit_lat = 2;
                                uid = 1;
                        end

                BORROW_GENERATE:
                        begin
                                for(int i=0;i<4;i++) begin
                                        if($unsigned(rb[i*WORD +: WORD]) >= $unsigned(ra[i*WORD +: WORD])) begin
                                                rt_data[i*WORD +: WORD] = 1;
                                        end
                                        else begin
                                                rt_data[i*WORD +: WORD] = 0;
                                        end
                                end
                                unit_lat = 2;
                                uid = 1;
                        end
                SUBTRACT_FROM_EXTENDED:
                        begin
                                for(int i=0;i<4;i++) begin
                                        rt_data[i*WORD +: WORD] = ra[i*WORD +: WORD] - rb[i*WORD +: WORD] + rc[31 +
i*WORD];
                                end
                                unit_lat = 2;
                                uid = 1;
                        end
                NOP_EXEC:  //code change
                        begin
                                rt_data = 0;
                                unit_lat = 0;
                                uid = 0;
                        end
                /*STOP_AND_SIGNAL:   //code change
                        begin
                                $finish;
                        end*/
```

```
                endcase

        end


endmodule
```

g. Odd_pipe.sv:

```
import definitions::*;


module
odd_pipe(clock,reset,rt_addr_in,pc_in,opcode_op_in,ra_in,rb_in,rc_in,I7_in,I10_in,I16_in,I18_in,wr_en_rt_in,rt_addr,fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr,

fw_op_02_data,fw_op_03_data,fw_op_04_data,fw_op_05_data,fw_op_06_data,fw_op_07_data,out_op_rt_data,out_op_wr_en_rt,

fw_op_01_lat,fw_op_02_lat,fw_op_03_lat,fw_op_04_lat,fw_op_05_lat,fw_op_06_lat,fw_op_07_lat,uid_op_02,uid_op_03,uid_op_04,uid_op_05,uid_op_06,uid_op_07, uid_op_08,flush, pc_out,

ls_data_rd_in, ls_data_out,
ls_addr_rd_out,ls_wr_en_op_in,ls_wr_en_op,wr_en_rt,wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,wr_en_op_07,




//Debug

rep_left_Bit_32_I10

);

        input clock,reset;

        input [0:6] rt_addr_in;

        input opcode opcode_op_in;

        input [0:127] ra_in,rb_in,rc_in;

        input [0:31]pc_in;

        input [0:6] I7_in;

        input [0:9] I10_in;

        input [0:15]I16_in;

        input [0:17]I18_in;

        input wr_en_rt_in;

        input ls_wr_en_op_in;

        output logic ls_wr_en_op;


        output logic [0:6] rt_addr;

        opcode opcode_op;

        logic [0:127] ra,rb,rc;

        logic [0:31]pc;

        logic [0:6] I7;
```

```
logic [0:9] I10;
logic [0:15]I16;
logic [0:17]I18;
output logic wr_en_rt;


logic [0:WORD-1] rep_left_Bit_32_I16,rep_left_Bit_32_I16_1,rep_left_Bit_32_I16_2zero;
output logic [0:WORD-1] rep_left_Bit_32_I10;
logic [0:HALF_WORD-1] rep_left_Bit_16_I10;
logic [0:127] rt_data;
//input wrbe[127:0];
//logic [0:6] fw_op_01_addr;
output logic [0:6] fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr;
//logic [0:127] fw_op_01_data;
output logic [0:127] fw_op_02_data,fw_op_03_data,fw_op_04_data,fw_op_05_data,fw_op_06_data,fw_op_07_data,out_op_rt_data;
//logic wr_en_op_01;
output logic wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,wr_en_op_07;
output logic out_op_wr_en_rt;
output logic [0:3] fw_op_01_lat,fw_op_02_lat,fw_op_03_lat,fw_op_04_lat,fw_op_05_lat,fw_op_06_lat,fw_op_07_lat;
//output logic [0:3] out_op_unit_lat;
logic [0:3] uid_op_01;
output logic [0:3] uid_op_02,uid_op_03,uid_op_04,uid_op_05,uid_op_06,uid_op_07,uid_op_08;
output logic [0:31] pc_out;
input [0:127] ls_data_rd_in;
output logic [0:127] ls_data_out;
output logic [0:14] ls_addr_rd_out;
output logic flush;
logic [0:3] unit_lat;
logic [0:3] uid;
int s;
logic temp_1;
logic [0:31] temp_32;
logic [0:15] temp_16;
logic [0:3] temp_4;
logic [0:7] temp_8;
logic [0:255] temp_256;
logic [0:127] temp_128;
assign rep_left_Bit_32_I16 = {{12{I16[0]}}, I16,4'b0000};
```

```
assign rep_left_Bit_32_I16_1 = {{16{I16[0]}}, I16};

assign rep_left_Bit_32_I10 = {{18{I10[0]}}, I10,4'b0000};

assign rep_left_Bit_16_I10 = {{6{I10[0]}}, I10};

assign rep_left_Bit_32_I16_2zero = {{14{I16[0]}}, I16,2'b00};


parameter MASK1 = 32'hfffffff0;


always_ff @(posedge clock) begin
        if(flush == 1) begin
                rt_addr   <= 0;
                opcode_op <= NOP_EXEC;
                ra                      <= 0;
                rb              <= 0;
                rc                      <= 0;
                I7      <= 0;
                I10     <= 0;
                I16     <= 0;
                I18                     <= 0;
                wr_en_rt  <= 0;
                ls_wr_en_op <=0;
                pc                      <= 0;
        end
        else begin
                rt_addr   <= rt_addr_in;
                opcode_op <= opcode_op_in;
                ra                      <= ra_in;
                rb              <= rb_in;
                rc                      <= rc_in;
                I7      <= I7_in;
                I10     <= I10_in;
                I16     <= I16_in;
                I18                     <= I18_in;
                wr_en_rt  <= wr_en_rt_in;
                ls_wr_en_op <=ls_wr_en_op_in;
                pc                      <= pc_in;
        end
end
```

```
always_ff @(posedge clock) begin
        if(reset == 1)begin
                uid_op_02     <= 0;
                uid_op_03     <= 0;
                uid_op_04     <= 0;
                uid_op_05     <= 0;
                uid_op_06     <= 0;
                uid_op_07     <= 0;
                uid_op_08     <= 0;

                fw_op_02_data <= 0;
                fw_op_03_data <= 0;
                fw_op_04_data <= 0;
                fw_op_05_data <= 0;
                fw_op_06_data <= 0;
                fw_op_07_data <= 0;
                out_op_rt_data<= 0;

                wr_en_op_02           <= 0;
                wr_en_op_03           <= 0;
                wr_en_op_04           <= 0;
                wr_en_op_05           <= 0;
                wr_en_op_06           <= 0;
                wr_en_op_07           <= 0;
                out_op_wr_en_rt  <= 0;

                fw_op_02_lat  <= 0;
                fw_op_03_lat  <= 0;
                fw_op_04_lat  <= 0;
                fw_op_05_lat  <= 0;
                fw_op_06_lat  <= 0;
                fw_op_07_lat  <= 0;
                //out_op_unit_lat  <= 0;

                fw_op_02_addr <= 0;
                fw_op_03_addr <= 0;
                fw_op_04_addr <= 0;
                fw_op_05_addr <= 0;
```

```
                fw_op_06_addr <= 0;
                fw_op_07_addr <= 0;
                out_op_rt_addr   <= 0;
end
else if(flush == 1) begin

                uid_op_02     <= 0;
                uid_op_03     <= uid_op_02;
                uid_op_04     <= uid_op_03;
                uid_op_05     <= uid_op_04;
                uid_op_06     <= uid_op_05;
                uid_op_07     <= uid_op_06;
                uid_op_08     <= uid_op_07;

                fw_op_02_data <= 0;
                fw_op_03_data <= fw_op_02_data;
                fw_op_04_data <= fw_op_03_data;
                fw_op_05_data <= fw_op_04_data;
                fw_op_06_data <= fw_op_05_data;
                fw_op_07_data <= fw_op_06_data;
                out_op_rt_data          <= fw_op_07_data;

                wr_en_op_02             <= 0;
                wr_en_op_03             <= wr_en_op_02;
                wr_en_op_04             <= wr_en_op_03;
                wr_en_op_05             <= wr_en_op_04;
                wr_en_op_06             <= wr_en_op_05;
                wr_en_op_07             <= wr_en_op_06;
                out_op_wr_en_rt  <= wr_en_op_07;

                fw_op_02_lat  <= 0;
                fw_op_03_lat  <= fw_op_02_lat;
                fw_op_04_lat  <= fw_op_03_lat;
                fw_op_05_lat  <= fw_op_04_lat;
                fw_op_06_lat  <= fw_op_05_lat;
                fw_op_07_lat  <= fw_op_06_lat;
                //out_op_unit_lat  <= fw_op_07_lat;
```

```
                fw_op_02_addr <= 0;
                fw_op_03_addr <= fw_op_02_addr;
                fw_op_04_addr <= fw_op_03_addr;
                fw_op_05_addr <= fw_op_04_addr;
                fw_op_06_addr <= fw_op_05_addr;
                fw_op_07_addr <= fw_op_06_addr;
                out_op_rt_addr   <= fw_op_07_addr;

end
else begin

                uid_op_02     <= uid;
                uid_op_03     <= uid_op_02;
                uid_op_04     <= uid_op_03;
                uid_op_05     <= uid_op_04;
                uid_op_06     <= uid_op_05;
                uid_op_07     <= uid_op_06;
                uid_op_08     <= uid_op_07;


                fw_op_02_data <= rt_data;
                fw_op_03_data <= fw_op_02_data;
                fw_op_04_data <= fw_op_03_data;
                fw_op_05_data <= fw_op_04_data;
                fw_op_06_data <= fw_op_05_data;
                fw_op_07_data <= fw_op_06_data;
                out_op_rt_data              <= fw_op_07_data;

                wr_en_op_02                 <= wr_en_rt;
                wr_en_op_03                 <= wr_en_op_02;
                wr_en_op_04                 <= wr_en_op_03;
                wr_en_op_05                 <= wr_en_op_04;
                wr_en_op_06                 <= wr_en_op_05;
                wr_en_op_07                 <= wr_en_op_06;
                out_op_wr_en_rt  <= wr_en_op_07;

                fw_op_01_lat  <= unit_lat;//decode
```

```
                                fw_op_02_lat  <= fw_op_01_lat;
                                fw_op_03_lat  <= fw_op_02_lat;
                                fw_op_04_lat  <= fw_op_03_lat;
                                fw_op_05_lat  <= fw_op_04_lat;
                                fw_op_06_lat  <= fw_op_05_lat;
                                fw_op_07_lat  <= fw_op_06_lat;
                                //out_op_unit_lat  <= fw_op_07_lat;


                                fw_op_02_addr <= rt_addr;
                                fw_op_03_addr <= fw_op_02_addr;
                                fw_op_04_addr <= fw_op_03_addr;
                                fw_op_05_addr <= fw_op_04_addr;
                                fw_op_06_addr <= fw_op_05_addr;
                                fw_op_07_addr <= fw_op_06_addr;
                                out_op_rt_addr   <= fw_op_07_addr;


                        end
                end
                always_comb begin
                        flush = 0;
                        pc_out = 0;
                        case(opcode_op)
                                LOAD_QUADWORD_DFORM:
                                        begin
//                                              $display($time," Inside LOAD_QUADWORD_DFORM Address = %h RA = %h \nMAsk = %h
rep_left_Bit_32_I10 = %h ls_addr_rd_out = %h\n THing:
%h",ls_addr_rd_out,$signed(ra[0:31]),MASK1,$signed(rep_left_Bit_32_I10),ls_addr_rd_out,$signed($signed(rep_left_Bit_32_I10) + $signed(ra[0:31])) );
                                                ls_addr_rd_out = $signed($signed(rep_left_Bit_32_I10) + $signed(ra[0:31])) & MASK1;
                                                rt_data         = ls_data_rd_in;
                                                unit_lat   = 4'd7;
                                                uid             = 4'd7;
                                        end
                                LOAD_QUADWORD_AFORM:
                                        begin
                                                //$display($time, "Inside LOAD_QUADWORD_AFORM
                                                ls_addr_rd_out = rep_left_Bit_32_I16_2zero & MASK1;
                                                rt_data         = ls_data_rd_in;
                                                unit_lat   = 4'd7;
```

```
                                        uid             = 4'd7;
                                end
                        STORE_QUADWORD_DFORM:
                                begin
                                        ls_addr_rd_out = $signed($signed(rep_left_Bit_32_I10) + $signed(ra[0:31])) & MASK1;
                                        ls_data_out = rc;
                                        unit_lat   = 4'd7;
                                        uid             = 4'd7;
                                end
                        STORE_QUADWORD_AFORM:
                                begin
                                $display($time, "Inside STORE_QUADWORD_AFORM  rep_left_Bit_32_I16_2zero %h %h
%h",rep_left_Bit_32_I16_2zero, I16,{{14{I16[0]}}, I16,2'b00});
                                        ls_addr_rd_out = rep_left_Bit_32_I16_2zero & MASK1;
                                        ls_data_out = rc;
                                        unit_lat   = 4'd7;
                                        uid             = 4'd7;
                                end
                        IMMEDIATE_LOAD_HALFWORD:
                                begin
                                        for(int i=0; i< 8;i++)          rt_data[i*HALF_WORD+:HALF_WORD] = I16;
                                        unit_lat   = 4'd7;
                                        uid             = 4'd7;
                                end
                        IMMEDIATE_LOAD_WORD:
                                begin
                                        for(int i=0; i< 4;i++)          rt_data[i*WORD+:WORD] = rep_left_Bit_32_I16_1;
                                        unit_lat   = 4'd7;
                                        uid             = 4'd7;
                                end
                        IMMEDIATE_LOAD_ADDRESS:
                                begin
                                        for(int i=0; i< 4;i++)          rt_data[i*WORD+:WORD] = {14'b00000000000000,{I18}};
                                        unit_lat   = 4'd7;
                                        uid             = 4'd7;
                                end
                        GATHER_BITS_FROM_WORDS:
                                begin
```

```
                temp_4 = 0;
                        for(int i=0;i < 4; i++) begin
                                temp_4[i] = ra[i*WORD + WORD - 1];
                        end
                        rt_data[0*WORD +: WORD] = {28'b0, temp_4};
                        rt_data[1*WORD +: WORD] = 0;
                        rt_data[2*WORD +: WORD] = 0;
                        rt_data[3*WORD +: WORD] = 0;
                        unit_lat = 4'd4;
                        uid = 4'd6;
                end
        GATHER_BITS_FROM_HALFWORDS:
                begin
                temp_8 = 0;
                        for(int i=0;i < 8; i++) begin
                                temp_8[i] = ra[i*HALF_WORD + HALF_WORD - 1];
                        end
                        rt_data[0*WORD +: WORD] = {24'b0, temp_8};
                        rt_data[1*WORD +: WORD] = 0;
                        rt_data[2*WORD +: WORD] = 0;
                        rt_data[3*WORD +: WORD] = 0;
                        unit_lat = 4'd4;
                        uid = 4'd6;
                        $display($time,"Inside GATHER_BITS_FROM_HALFWORDS");
                end
        SHUFFLE_BYTES:
                begin
                temp_256 = {ra, rb};
                        for(int i = 0;i< WORD; i++) begin
                                temp_8 = rc_in[i*BYTE+:BYTE];
                                if(temp_8[0:1] == 2'b10) begin
                                        rt_data[i*BYTE +:BYTE] = 8'h0;
                                end
                                else if(temp_8[0:2] == 3'b110) begin
                                        rt_data[i*BYTE +:BYTE] = 8'hff;
                                end
                                else if(temp_8[0:2] == 3'b111) begin
                                        rt_data[i*BYTE +:BYTE] = 8'h80;
```

```
                                        end
                                        else begin
                                                temp_8 = temp_8 & 8'h1F;
                                                rt_data[i*BYTE +:BYTE] = temp_256[i*temp_8+:BYTE];
                                        end
                                end
                                unit_lat = 4'd4;
                                uid = 4'd6;
                        end
                SHIFT_LEFT_QUADWORD_BY_BITS:
                        begin
                                s = rb[29:31];
                                for(int b=0; b<128; b++) begin
                                        if(b+s < 128) begin
                                                temp_128[b] = ra[b+s];
                                        end
                                        else begin
                                                temp_128[b] = 0;
                                        end
                                end
                                rt_data = temp_128;
                                unit_lat = 4'd4;
                                uid = 4'd6;
                        end
                SHIFT_LEFT_QUADWORD_BY_BITS_IMMEDIATE:
                        begin
                                s = I7 & 8'h07;
                                for(int b=0; b<128; b++) begin
                                        if(b+s < 128) begin
                                                temp_128[b] = ra[b+s];
                                        end
                                        else begin
                                                temp_128[b] = 0;
                                        end
                                end
                                rt_data = temp_128;
                                unit_lat = 4'd4;
                                uid = 4'd6;
```

```
                end
SHIFT_LEFT_QUADWORD_BY_BYTES:
                begin
                        s = rb[27:31];
                        for(int b=0; b<16; b++) begin
                                if(b+s < 16) begin
                                        temp_128[b*BYTE +:BYTE] = ra[(b+s)*BYTE +: BYTE];
                                end
                                else begin
                                        temp_128[b*BYTE +:BYTE] = 0;
                                end
                        end
                        rt_data = temp_128;
                        unit_lat = 4'd4;
                        uid = 4'd6;
                end
SHIFT_LEFT_QUADWORD_BY_BYTES_IMMEDIATE:
                begin
                        s = I7 & 8'h1F;
                        for(int b=0; b<16; b++) begin
                                if(b+s < 16) begin
                                        temp_128[b*BYTE +:BYTE] = ra[(b+s)*BYTE +: BYTE];
                                end
                                else begin
                                        temp_128[b*BYTE +:BYTE] = 0;
                                end
                        end
                        rt_data = temp_128;
                        unit_lat = 4'd4;
                        uid = 4'd6;
                end
ROTATE_QUADWORD_BY_BYTES:
                begin
                        s = rb[28:31];
                        for(int b=0; b<16; b++) begin
                                if(b+s < 16) begin
                                        temp_128[b*BYTE +:BYTE] = ra[(b+s)*BYTE +: BYTE];
                                end
```

```
                                        else begin
                                                temp_128[b*BYTE +:BYTE] = ra[(b+s-16)*BYTE +: BYTE];
                                        end
                                end
                                rt_data = temp_128;
                                unit_lat = 4'd4;
                                uid = 4'd6;
                        end

ROTATE_QUADWORD_BY_BYTES_IMMEDIATE:
                        begin
                                s = I7[3:6];
                                for(int b=0; b<16; b++) begin
                                        if(b+s < 16) begin
                                                temp_128[b*BYTE +:BYTE] = ra[(b+s)*BYTE +: BYTE];
                                        end
                                        else begin
                                                temp_128[b*BYTE +:BYTE] = ra[(b+s-16)*BYTE +: BYTE];
                                        end
                                end
                                rt_data = temp_128;
                                unit_lat = 4'd4;
                                uid = 4'd6;
                        end

BRANCH_RELATIVE:
                        begin
                                pc_out = $signed(pc) + $signed(rep_left_Bit_32_I16_2zero);
                                $display("pc value relative is %d", pc);
                                $display("I16 value relative is %d",rep_left_Bit_32_I16_2zero);
                                $display("signed pc value relative is %d", $signed(pc));
                                $display("signed I16 value relative is %d",$signed(rep_left_Bit_32_I16_2zero));
                                $display("pc_out relative is %d", pc_out);
                                unit_lat = 4'd1;
                                uid = 4'd8;
                                flush = 1;
                        end
BRANCH_ABSOLUTE:
```

```
		begin
			pc_out = rep_left_Bit_32_I16_2zero;
			$display($time,"pc_out absolute is %d", pc_out);
			unit_lat = 4'd1;
			uid = 4'd8;
			flush = 1;
			$display($time,"flush for br absolute is %d", flush);
		end
BRANCH_IF_NOT_ZERO_WORD:
		begin
			if(rc[0:31] != 32'd0) begin
				pc_out = pc + $signed(rep_left_Bit_32_I16_2zero) & 32'hFFFFFFFC;
				flush = 1;
			end
			unit_lat = 4'd1;
			uid = 4'd8;
			//$display($time,"flush for br if n zero is %d", flush);

		end
BRANCH_IF_ZERO_WORD:
		begin
			if(rc[0:31] == 32'd0) begin
				pc_out = pc + $signed(rep_left_Bit_32_I16_2zero) & 32'hFFFFFFFC;
				flush = 1;
			end
			unit_lat = 4'd1;
			uid = 4'd8;
			$display($time,"flush for br if  zero word is %d", flush);
		end
BRANCH_IF_NOT_ZERO_HALFWORD:
		begin
			if(rc[16:31] != 0) begin
				pc_out = pc + $signed(rep_left_Bit_32_I16_2zero) & 32'hFFFFFFFC;
				flush = 1;
			end
			unit_lat = 4'd1;
			uid = 4'd8;
		end
```

```
                BRANCH_IF_ZERO_HALFWORD:
                        begin
                                if(rc[2:3] == 0) begin
                                        pc_out = pc + $signed(rep_left_Bit_32_I16_2zero) & 32'hFFFFFFFC;
                                        flush = 1;
                                end
                                unit_lat = 4'd1;
                                uid = 4'd8;
                        end
                NOP_LOAD:
                        begin
                                rt_data = 0;
                                unit_lat = 0;
                                uid = 0;
                        end
                STOP_AND_SIGNAL:
                        begin
                                rt_data = 0;
                                unit_lat = 0;
                                uid = 0;
                        end
        endcase
    end
endmodule
```

h. Definitions.sv:

```
package definitions;
    parameter WORD = 32;
    parameter HALF_WORD = 16;
    parameter BYTE = 8;
    parameter QUAD_WORD = 128;
    parameter NIBBLE = 4;

    parameter SMAX = $bitstoshortreal(32'h7fffffff);
    parameter SMIN = $bitstoshortreal(32'h80000000);

    parameter EVEN = 0;
```

```
parameter ODD = 1;


typedef enum logic[10:0] {
                                                    ADD_HALFWORD
                                = 11'b00011001000,
                                                    ADD_HALFWORD_IMMEDIATE
            = 11'b00011101000,
                                                    ADD_WORD
                                = 11'b00011000000,
                                                    ADD_WORD_IMMEDIATE
            = 11'b00011100000,
                                                    SUB_FROM_HALFWORD
                        = 11'b00001001000,
                                                    SUB_FROM_HALFWORD_IMMEDIATE
            = 11'b00001101000,
                                                    SUB_FROM_WORD
                                = 11'b00001000000,
                                                    SUB_FROM_WORD_IMMEDIATE
                        = 11'b00001100000,
                                                    MULTIPLY
                                = 11'b01111000100,
                                                    MULITPLY_UNSIGNED
                        = 11'b01111001100,
                                                    MULTIPLY_IMMEDIATE
                        = 11'b01110100000,
                                                    MULTIPLY_AND_ADD
                        = 11'b11000000000,
                                                    COUNTING_LEADING_ZEROS
                        = 11'b01010100101,
                                                    FORM_SELECT_MASK_BYTES_IMMEDIATE
= 11'b00110010100,
                                                    FORM_SELECT_MASK_BYTES
            = 11'b00110110110,
                                                    FORM_SELECT_MASK_HALFWORD
            = 11'b00110110101,
                                                    FORM_SELECT_MASK_WORD
                        = 11'b00110110100,
                                                    AND
                                                = 11'b00011000001,
                                                    AND_HALFWORD_IMMEDIATE
                        = 11'b00010101000,
                                                    AND_WORD_IMMEDIATE
                                = 11'b00010100000,
                                                    OR
                                                = 11'b00001000001,
                                                    OR_BYTE_IMMEDIATE
                        = 11'b00000110000,
```

```
                                                    OR_HALFWORD_IMMEDIATE
                = 11'b00000101000,
                                                    OR_WORD_IMMEDIATE
            = 11'b00000100000,
                                                    NOR
                                                = 11'b00001001001,
                                                    XOR
                                                = 11'b01001000001,
                                                    XOR_BYTE_IMMEDIATE
                = 11'b01000110000,
                                                    XOR_HALFWORD_IMMEDIATE
                = 11'b01000101000,
                                                    XOR_WORD_IMMEDIATE
                = 11'b01000100000,
                                                    FLOATING_ADD
                        = 11'b01011000100,
                                                    FLOATING_SUBTRACT
                = 11'b01011000101,
                                                    FLOATING_MULTIPLY
            = 11'b01011000110,
                                                    FLOATING_MULTIPLY_AND_ADD
            = 11'b11100000000,
                                                    FLOATING_MULTIPLY_AND_SUBTRACT
= 11'b11110000000,
                                                    EQUIVALENT
                                    = 11'b01001001001,
                                                    SHIFT_LEFT_HALFWORD
                = 11'b00001011111,
                                                    SHIFT_LEFT_HALFWORD_IMMEDIATE
= 11'b00001111111,
                                                    SHIFT_LEFT_WORD
                        = 11'b00001011011,
                                                    SHIFT_LEFT_WORD_IMMEDIATE
            = 11'b00001111011,
                                                    ROTATE_HALFWORD
                        = 11'b00001011100,
                                                    ROTATE_HALFWORD_IMMEDIATE
            = 11'b00001111100,
                                                    ROTATE_WORD
                                    = 11'b00001011000,
                                                    ROTATE_WORD_IMMEDIATE
                = 11'b00001111000,
                                                    ROTATE_AND_MASK_HALFWORD
            = 11'b00001011101,
                                                    ROTATE_AND_MASK_HALFWORD_IMMEDIATE
= 11'b00001111101,
                                                    COUNT_ONES_IN_BYTES
                = 11'b01010110100,
```

= 11'b00011010011,

= 11'b00001010011,

= 11'b01001010011,

= 11'b01111001000,

= 11'b01111101000,

= 11'b01111000000,

= 11'b01111100000,

= 11'b01001001000,

= 11'b01001101000,

= 11'b01001000000,

= 11'b01001100000,

= 11'b01011000000,

= 11'b01011100000,

11'b01011001000,

11'b01011101000,

= 11'b00011000010,

= 11'b01101000000,

= 11'b00001000010,

= 11'b01101000001,

= 11'b00000000001,

= 11'b01000000001,

= 11'b00000000000,

= 11'b00110100000,

= 11'b00110000100,

AVERAGE_BYTES

ABSOLUTE_DIFFERENCE_OF_BYTES

SUM_BYTES_INTO_HALFWORDS

COMPARE_EQUAL_HALFWORD

COMPARE_EQUAL_HALFWORD_IMMEDIATE

COMPARE_EQUAL_WORD

COMPARE_EQUAL_WORD_IMMEDIATE

COMPARE_GREATER_THAN_HALFWORD

COMPARE_GREATER_THAN_HALFWORD_IMMEDIATE

COMPARE_GREATER_THAN_WORD

COMPARE_GREATER_THAN_WORD_IMMEDIATE

COMPARE_LOGICAL_GREATER_THAN_WORD

COMPARE_LOGICAL_GREATER_THAN_WORD_IMMEDIATE

COMPARE_LOGICAL_GREATER_THAN_HALFWORD =

COMPARE_LOGICAL_GREATER_THAN_HALFWORD_IMMEDIATE =

CARRY_GENERATE

ADD_EXTENDED

BORROW_GENERATE

SUBTRACT_FROM_EXTENDED

NOP_LOAD

NOP_EXEC

STOP_AND_SIGNAL

LOAD_QUADWORD_DFORM

LOAD_QUADWORD_AFORM

```
                        = 11'b00100100000,
                        = 11'b00100000100,
                        = 11'b01000001100,
                                = 11'b01000000100,
                        = 11'b01000010000,
                        = 11'b00110110000,
                = 11'b00110110001,
                                = 11'b10110000000,
                = 11'b00111011011,
= 11'b00111111011,
= 11'b00111011111,
= 11'b00111111111,
                = 11'b00111011100,
= 11'b00111111100,
                                = 11'b00110010000,
                                = 11'b00110000000,
                        = 11'b00100001000,
                                = 11'b00100000000,
                = 11'b00100011000,
                        = 11'b00100010000,
= 11'b00110011000,
= 11'b00110001000

                parameter LS_SIZE = 320000;
```

```
STORE_QUADWORD_DFORM

STORE_QUADWORD_AFORM

IMMEDIATE_LOAD_HALFWORD

IMMEDIATE_LOAD_WORD

IMMEDIATE_LOAD_ADDRESS

GATHER_BITS_FROM_WORDS

GATHER_BITS_FROM_HALFWORDS

SHUFFLE_BYTES

SHIFT_LEFT_QUADWORD_BY_BITS

SHIFT_LEFT_QUADWORD_BY_BITS_IMMEDIATE

SHIFT_LEFT_QUADWORD_BY_BYTES

SHIFT_LEFT_QUADWORD_BY_BYTES_IMMEDIATE

ROTATE_QUADWORD_BY_BYTES

ROTATE_QUADWORD_BY_BYTES_IMMEDIATE

BRANCH_RELATIVE

BRANCH_ABSOLUTE

BRANCH_IF_NOT_ZERO_WORD

BRANCH_IF_ZERO_WORD

BRANCH_IF_NOT_ZERO_HALFWORD

BRANCH_IF_ZERO_HALFWORD

BRANCH_RELATIVE_AND_SET_LINK

BRANCH_ABSOLUTE_AND_SET_LINK

} opcode;
```

```
endpackage
```

i. Local store.sv:

```
import definitions::*;

module local_store(clock,lsa_addr,lsa_data_in,lsa_data_out,ls_wr_en,ls_mem);
        input clock;
        input ls_wr_en;
        input [0:14] lsa_addr;
        input [0:127] lsa_data_in;
        output logic [0:127] lsa_data_out;
        output logic [0:7] ls_mem [LS_SIZE];

        initial begin
    $readmemb("preload_data", ls_mem);
  end

        always_comb begin
                for (int i=0; i<16; i++) begin
                        lsa_data_out[i*BYTE +: BYTE] = ls_mem[i + lsa_addr];
                end
        end

        always_ff @(posedge clock) begin
                if (ls_wr_en == 1) begin
                        for (int i=0; i<16; i++) begin
                                ls_mem[i + lsa_addr] <= lsa_data_in[i*BYTE +: BYTE];
                        end
                end
        end
endmodule
```

j. Top Module part2.sv:

```
import definitions::*;

module topmodule_part2(clock,reset);
```

```
input clock,reset;
opcode opcode_ep;

logic ls_wr_en_op_in;



logic [0:31] pc_out;

logic flush,flush_fetch;
logic [0:127] ra_rf_data_ep, rb_rf_data_ep, rc_rf_data_ep, ra_rf_data_op, rb_rf_data_op,rc_rf_data_op;//DRO
opcode opcode_ep_r, opcode_op_r;
logic [0:6] I7_ep_r, I7_op_r;
logic [0:9] I10_ep_r, I10_op_r;
logic [0:15] I16_ep_r, I16_op_r;
logic [0:17] I18_ep_r, I18_op_r;
logic [0:3] fw_uid_ep_02, fw_uid_ep_03, fw_uid_ep_04, fw_uid_ep_05, fw_uid_ep_06, fw_uid_ep_07,fw_uid_ep_08; //DRO
logic [0:127] fw_ep_02_data, fw_ep_03_data, fw_ep_04_data, fw_ep_05_data, fw_ep_06_data, fw_ep_07_data; //DRO

logic [0:3] fw_uid_op_02, fw_uid_op_03, fw_uid_op_04, fw_uid_op_05, fw_uid_op_06, fw_uid_op_07,fw_uid_op_08; //DRO
logic [0:127] fw_op_02_data, fw_op_03_data, fw_op_04_data, fw_op_05_data, fw_op_06_data, fw_op_07_data; //DRO

logic [0:31] pc_r;

logic [0:127] ra_fw_data_ep, rb_fw_data_ep, rc_fw_data_ep, ra_fw_data_op, rb_fw_data_op,rc_fw_data_op; //DRO
logic wr_en_rt_ep_r, wr_en_rt_op_r;
logic [0:127] out_ep_rt_data,out_op_rt_data; //DRO
logic out_ep_wr_en_rt, out_op_wr_en_rt; //DRO
logic [0:3] fw_ep_01_lat, fw_ep_02_lat, fw_ep_03_lat, fw_ep_04_lat, fw_ep_05_lat, fw_ep_06_lat, fw_ep_07_lat;
logic [0:3] fw_op_01_lat, fw_op_02_lat, fw_op_03_lat, fw_op_04_lat, fw_op_05_lat, fw_op_06_lat, fw_op_07_lat;
logic [0:127] ls_data_rd_in,ls_data_out; // DRO
logic [0:14]  ls_addr_rd_out; //DRO
logic ls_wr_en_op_r,ls_wr_en_op;
logic [0:6] rt_addr_ep_r,rt_addr_op_r;
logic [0:6] rt_addr_in;



logic wr_en_op_01,wr_en_op_02,wr_en_op_03,wr_en_op_04,wr_en_op_05,wr_en_op_06,wr_en_op_07;
logic wr_en_ep_01,wr_en_ep_02,wr_en_ep_03,wr_en_ep_04,wr_en_ep_05,wr_en_ep_06,wr_en_ep_07;
```

```
//Debug
logic [0:127]reg_file_mem [128];
logic [0:7] ls_mem [LS_SIZE];
logic [0:31] rep_left_Bit_32_I10;


logic stall;
logic [0:31] pc_out_fetch;
logic [0:31] ins1, ins2;
logic [0:31] pc_in;
logic clear;
logic [0:6] I7_ep;
logic [0:9] I10_ep;
logic [0:15]I16_ep;
logic [0:17]I18_ep;
logic wr_en_rt_ep;
opcode opcode_op;
//input [0:31]pc_op;
logic [0:6] I7_op;
logic [0:9] I10_op;
logic [0:15]I16_op;
logic [0:17]I18_op;
logic wr_en_rt_op;
logic [0:6] ra_addr_ep;
logic [0:6] rb_addr_ep;
logic [0:6] rc_addr_ep;
logic [0:6] rt_addr_ep;
logic [0:6] ra_addr_op;
logic [0:6] rb_addr_op,rc_addr_op;
logic [0:6] rt_addr_op;
logic [0:6] fw_ep_01_addr,fw_ep_02_addr,fw_ep_03_addr,fw_ep_04_addr,fw_ep_05_addr,fw_ep_06_addr,fw_ep_07_addr,out_ep_rt_addr;
logic [0:6] fw_op_01_addr,fw_op_02_addr,fw_op_03_addr,fw_op_04_addr,fw_op_05_addr,fw_op_06_addr,fw_op_07_addr,out_op_rt_addr;
logic [0:6] ra_addr_ep_r;
logic [0:6] rb_addr_ep_r;
logic [0:6] rc_addr_ep_r;
logic [0:6] ra_addr_op_r;
```

```
logic [0:6] rb_addr_op_r;
logic [0:6] rc_addr_op_r;

fetch       fetch1(.clock(clock),
                   .reset(reset),
                   .stall(stall),
                   .flush(flush),
                   .pc_in(pc_out),
                   .pc_out(pc_out_fetch),
                   .ins1(ins1),
                   .ins2(ins2),
                   .flush_fetch(flush_fetch)
                   );
decode      decode1(.clock(clock),
                   .reset(reset),
                   .pc_in(pc_out_fetch),
                   .ins1(ins1),
                   .ins2(ins2),
                   .flush_fetch(flush_fetch),
                   .flush(flush),
                   .pc_out(pc_in),
                   .opcode_ep(opcode_ep),
                   .I7_ep(I7_ep),
                   .I10_ep(I10_ep),
                   .I16_ep(I16_ep),
                   .I18_ep(I18_ep),
                   .wr_en_rt_ep(wr_en_rt_ep),
                   .opcode_op(opcode_op),
                   .I7_op(I7_op),
                   .I10_op(I10_op),
                   .I16_op(I16_op),
                   .I18_op(I18_op),
                   .wr_en_rt_op(wr_en_rt_op),
                   .ra_addr_ep(ra_addr_ep),
                   .rb_addr_ep(rb_addr_ep),
                   .rc_addr_ep(rc_addr_ep),
                   .rt_addr_ep(rt_addr_ep),
                   .ra_addr_op(ra_addr_op),
```

```
.rb_addr_op(rb_addr_op),
.rc_addr_op(rc_addr_op),
.rt_addr_op(rt_addr_op),
.clear(clear),
.fw_ep_01_addr(fw_ep_01_addr),
.fw_ep_02_addr(fw_ep_02_addr),
.fw_ep_03_addr(fw_ep_03_addr),
.fw_ep_04_addr(fw_ep_04_addr),
.fw_ep_05_addr(fw_ep_05_addr),
.fw_ep_06_addr(fw_ep_06_addr),
.fw_ep_07_addr(fw_ep_07_addr),
.out_ep_rt_addr(out_ep_rt_addr),
.fw_op_01_addr(fw_op_01_addr),
.fw_op_02_addr(fw_op_02_addr),
.fw_op_03_addr(fw_op_03_addr),
.fw_op_04_addr(fw_op_04_addr),
.fw_op_05_addr(fw_op_05_addr),
.fw_op_06_addr(fw_op_06_addr),
.fw_op_07_addr(fw_op_07_addr),
.out_op_rt_addr(out_op_rt_addr),
.rf_addr_ep_r(rt_addr_ep_r),
.rf_addr_op_r(rt_addr_op_r),
.wr_en_rt_ep_r(wr_en_rt_ep_r),
.wr_en_rt_op_r(wr_en_rt_op_r),
.wr_en_ep_01(wr_en_ep_01),
.wr_en_ep_02(wr_en_ep_02),
.wr_en_ep_03(wr_en_ep_03),
.wr_en_ep_04(wr_en_ep_04),
.wr_en_ep_05(wr_en_ep_05),
.wr_en_ep_06(wr_en_ep_06),
.wr_en_ep_07(wr_en_ep_07),
.wr_en_op_01(wr_en_op_01),
.wr_en_op_02(wr_en_op_02),
.wr_en_op_03(wr_en_op_03),
.wr_en_op_04(wr_en_op_04),
.wr_en_op_05(wr_en_op_05),
.wr_en_op_06(wr_en_op_06),
.fw_uid_ep_02(fw_uid_ep_02),
```

```
                                        .fw_uid_ep_03(fw_uid_ep_03),
                                        .fw_uid_ep_04(fw_uid_ep_04),
                                        .fw_uid_ep_05(fw_uid_ep_05),
                                        .fw_uid_ep_06(fw_uid_ep_06),
                                        .fw_uid_ep_07(fw_uid_ep_07),
                                        .fw_uid_op_02(fw_uid_op_02),
                                        .fw_uid_op_03(fw_uid_op_03),
                                        .fw_uid_op_04(fw_uid_op_04),
                                        .fw_uid_op_05(fw_uid_op_05),
                                        .fw_uid_op_06(fw_uid_op_06),
                                        .fw_uid_op_07(fw_uid_op_07),
                                        .stall(stall)

);

        rf_fw rf1(      .clock(clock),
                                                .reset(reset),
                                                .clear(clear),
                                                .wr_en_rt_ep(wr_en_rt_ep),
                                                .wr_en_rt_op(wr_en_rt_op),
                                                .ls_wr_en_op(ls_wr_en_op_in),
                                                .flush(flush),
                                                .pc(pc_in),
                                                .opcode_ep(opcode_ep),
                                                .ra_addr_ep_r(ra_addr_ep_r),
                                                .rb_addr_ep_r(rb_addr_ep_r),
                                                .rc_addr_ep_r(rc_addr_ep_r),
                                                .ra_addr_op_r(ra_addr_op_r),
                                                .rb_addr_op_r(rb_addr_op_r),
                                                .rc_addr_op_r(rc_addr_op_r),
                                                .ra_addr_ep(ra_addr_ep),
                                                .rb_addr_ep(rb_addr_ep),
                                                .rc_addr_ep(rc_addr_ep),
                                                .rt_addr_ep(rt_addr_ep),
                                                .I7_ep(I7_ep),
                                                .I10_ep(I10_ep),
                                                .I16_ep(I16_ep),
                                                .I18_ep(I18_ep),
```

```
.opcode_op(opcode_op),
.ra_addr_op(ra_addr_op),
.rb_addr_op(rb_addr_op),
.rc_addr_op(rc_addr_op),
.rt_addr_op(rt_addr_op),
.I7_op(I7_op),
.I10_op(I10_op),
.I16_op(I16_op),
.I18_op(I18_op),
.ra_rf_data_ep(ra_rf_data_ep),
.rb_rf_data_ep(rb_rf_data_ep),
.rc_rf_data_ep(rc_rf_data_ep),
.ra_rf_data_op(ra_rf_data_op),
.rb_rf_data_op(rb_rf_data_op),
.rc_rf_data_op(rc_rf_data_op),
.opcode_ep_r(opcode_ep_r),
.rt_addr_ep_r(rt_addr_ep_r),
.I7_ep_r(I7_ep_r),
.I10_ep_r(I10_ep_r),
.I16_ep_r(I16_ep_r),
.I18_ep_r(I18_ep_r),
.opcode_op_r(opcode_op_r),
.rt_addr_op_r(rt_addr_op_r),
.I7_op_r(I7_op_r),
.I10_op_r(I10_op_r),
.I16_op_r(I16_op_r),
.I18_op_r(I18_op_r),
.fw_ep_02_addr(fw_ep_02_addr),
.fw_ep_03_addr(fw_ep_03_addr),
.fw_ep_04_addr(fw_ep_04_addr),
.fw_ep_05_addr(fw_ep_05_addr),
.fw_ep_06_addr(fw_ep_06_addr),
.fw_ep_07_addr(fw_ep_07_addr),
.out_ep_rt_addr(out_ep_rt_addr),
.fw_ep_02_data(fw_ep_02_data),
.fw_ep_03_data(fw_ep_03_data),
.fw_ep_04_data(fw_ep_04_data),
.fw_ep_05_data(fw_ep_05_data),
```

```
.fw_ep_06_data(fw_ep_06_data),
.fw_ep_07_data(fw_ep_07_data),
.out_ep_rt_data(out_ep_rt_data),
.fw_uid_ep_02(fw_uid_ep_02),
.fw_uid_ep_03(fw_uid_ep_03),
.fw_uid_ep_04(fw_uid_ep_04),
.fw_uid_ep_05(fw_uid_ep_05),
.fw_uid_ep_06(fw_uid_ep_06),
.fw_uid_ep_07(fw_uid_ep_07),
.fw_uid_ep_08(fw_uid_ep_08),
.fw_op_02_addr(fw_op_02_addr),
.fw_op_03_addr(fw_op_03_addr),
.fw_op_04_addr(fw_op_04_addr),
.fw_op_05_addr(fw_op_05_addr),
.fw_op_06_addr(fw_op_06_addr),
.fw_op_07_addr(fw_op_07_addr),
.out_op_rt_addr(out_op_rt_addr),
.fw_op_02_data(fw_op_02_data),
.fw_op_03_data(fw_op_03_data),
.fw_op_04_data(fw_op_04_data),
.fw_op_05_data(fw_op_05_data),
.fw_op_06_data(fw_op_06_data),
.fw_op_07_data(fw_op_07_data),
.out_op_rt_data(out_op_rt_data),
.fw_uid_op_02(fw_uid_op_02),
.fw_uid_op_03(fw_uid_op_03),
.fw_uid_op_04(fw_uid_op_04),
.fw_uid_op_05(fw_uid_op_05),
.fw_uid_op_06(fw_uid_op_06),
.fw_uid_op_07(fw_uid_op_07),
.fw_uid_op_08(fw_uid_op_08),
.ra_fw_data_ep(ra_fw_data_ep),
.rb_fw_data_ep(rb_fw_data_ep),
.rc_fw_data_ep(rc_fw_data_ep),
.ra_fw_data_op(ra_fw_data_op),
.rb_fw_data_op(rb_fw_data_op),
.rc_fw_data_op(rc_fw_data_op),
.pc_out(pc_r),
```

```
                                        .wr_en_rt_ep_r(wr_en_rt_ep_r),
                                        .wr_en_rt_op_r(wr_en_rt_op_r),
                                        .ls_wr_en_op_r(ls_wr_en_op_r),
                                        .wr_en_op_02(wr_en_op_02),
                                        .wr_en_op_03(wr_en_op_03),
                                        .wr_en_op_04(wr_en_op_04),
                                        .wr_en_op_05(wr_en_op_05),
                                        .wr_en_op_06(wr_en_op_06),
                                        .wr_en_op_07(wr_en_op_07),
                                        .wr_en_ep_01(wr_en_ep_01),
                                        .wr_en_ep_02(wr_en_ep_02),
                                        .wr_en_ep_03(wr_en_ep_03),
                                        .wr_en_ep_04(wr_en_ep_04),
                                        .wr_en_ep_05(wr_en_ep_05),
                                        .wr_en_ep_06(wr_en_ep_06),
                                        .wr_en_ep_07(wr_en_ep_07),
                                        .out_ep_wr_en_rt(out_ep_wr_en_rt),
                                        .out_op_wr_en_rt(out_op_wr_en_rt),
                                        .clear_r(clear_r));

even_pipe e1(.clock(clock),
                                        .reset(reset),
                                        .clear_in(clear_r),
                                        .rt_addr_in(rt_addr_ep_r),
                                        .flush_in(flush),
                                        .opcode_ep_in(opcode_ep_r),
                                        .ra_in(ra_fw_data_ep),
                                        .rb_in(rb_fw_data_ep),
                                        .rc_in(rc_fw_data_ep),
                                        .I7_in(I7_ep_r),
                                        .I10_in(I10_ep_r),
                                        .I16_in(I16_ep_r),
                                        .I18_in(I18_ep_r),
                                        .wr_en_rt_in(wr_en_rt_ep_r),
                                        .rt_addr(fw_ep_01_addr),
                                        .fw_ep_02_addr(fw_ep_02_addr),
                                        .fw_ep_03_addr(fw_ep_03_addr),
                                        .fw_ep_04_addr(fw_ep_04_addr),
```

```
                                        .fw_ep_05_addr(fw_ep_05_addr),
                                        .fw_ep_06_addr(fw_ep_06_addr),
                                        .fw_ep_07_addr(fw_ep_07_addr),
                                        .out_ep_rt_addr(out_ep_rt_addr),
                                        .fw_ep_02_data(fw_ep_02_data),
                                        .fw_ep_03_data(fw_ep_03_data),
                                        .fw_ep_04_data(fw_ep_04_data),
                                        .fw_ep_05_data(fw_ep_05_data),
                                        .fw_ep_06_data(fw_ep_06_data),
                                        .fw_ep_07_data(fw_ep_07_data),
                                        .out_ep_rt_data(out_ep_rt_data),
                                        .out_ep_wr_en_rt(out_ep_wr_en_rt),
                                        .uid_ep_02(fw_uid_ep_02),
                                        .uid_ep_03(fw_uid_ep_03),
                                        .uid_ep_04(fw_uid_ep_04),
                                        .uid_ep_05(fw_uid_ep_05),
                                        .uid_ep_06(fw_uid_ep_06),
                                        .uid_ep_07(fw_uid_ep_07),
                                        .uid_ep_08(fw_uid_ep_08),
                                        .fw_ep_01_lat(fw_ep_01_lat),
                                        .fw_ep_02_lat(fw_ep_02_lat),
                                        .fw_ep_03_lat(fw_ep_03_lat),
                                        .fw_ep_04_lat(fw_ep_04_lat),
                                        .fw_ep_05_lat(fw_ep_05_lat),
                                        .fw_ep_06_lat(fw_ep_06_lat),
                                        .fw_ep_07_lat(fw_ep_07_lat),
                                        .wr_en_rt(wr_en_ep_01),
                                        .wr_en_ep_02(wr_en_ep_02),
                                        .wr_en_ep_03(wr_en_ep_03),
                                        .wr_en_ep_04(wr_en_ep_04),
                                        .wr_en_ep_05(wr_en_ep_05),
                                        .wr_en_ep_06(wr_en_ep_06),
                                        .wr_en_ep_07(wr_en_ep_07));

odd_pipe o1(.clock(clock),
                                        .reset(reset),
                                        .rt_addr_in(rt_addr_op_r),
                                        .pc_in(pc_r),
```

```
.opcode_op_in(opcode_op_r),
.ra_in(ra_fw_data_op),
.rb_in(rb_fw_data_op),
.rc_in(rc_fw_data_op),
.I7_in(I7_op_r),
.I10_in(I10_op_r),
.I16_in(I16_op_r),
.I18_in(I18_op_r),
.wr_en_rt_in(wr_en_rt_op_r),
.ls_wr_en_op_in(ls_wr_en_op_r),
.rt_addr(fw_op_01_addr),
.fw_op_02_addr(fw_op_02_addr),
.fw_op_03_addr(fw_op_03_addr),
.fw_op_04_addr(fw_op_04_addr),
.fw_op_05_addr(fw_op_05_addr),
.fw_op_06_addr(fw_op_06_addr),
.fw_op_07_addr(fw_op_07_addr),
.out_op_rt_addr(out_op_rt_addr),
.fw_op_02_data(fw_op_02_data),
.fw_op_03_data(fw_op_03_data),
.fw_op_04_data(fw_op_04_data),
.fw_op_05_data(fw_op_05_data),
.fw_op_06_data(fw_op_06_data),
.fw_op_07_data(fw_op_07_data),
.out_op_rt_data(out_op_rt_data),
.out_op_wr_en_rt(out_op_wr_en_rt),
.fw_op_01_lat(fw_op_01_lat),
.fw_op_02_lat(fw_op_02_lat),
.fw_op_03_lat(fw_op_03_lat),
.fw_op_04_lat(fw_op_04_lat),
.fw_op_05_lat(fw_op_05_lat),
.fw_op_06_lat(fw_op_06_lat),
.fw_op_07_lat(fw_op_07_lat),
.uid_op_02(fw_uid_op_02),
.uid_op_03(fw_uid_op_03),
.uid_op_04(fw_uid_op_04),
.uid_op_05(fw_uid_op_05),
.uid_op_06(fw_uid_op_06),
```

```
                                        .uid_op_07(fw_uid_op_07),
                                        .uid_op_08(fw_uid_op_08),
                                        .flush(flush),
                                        .pc_out(pc_out),
                                        .ls_data_rd_in(ls_data_rd_in),
                                        .ls_data_out(ls_data_out),
                                        .ls_addr_rd_out(ls_addr_rd_out),
                                        .ls_wr_en_op(ls_wr_en_op),
                                        .wr_en_op_07(wr_en_op_07),
                                        //Debug
                                        .rep_left_Bit_32_I10(rep_left_Bit_32_I10),
                                        .wr_en_rt(wr_en_op_01),
                                        .wr_en_op_02(wr_en_op_02),
                                        .wr_en_op_03(wr_en_op_03),
                                        .wr_en_op_04(wr_en_op_04),
                                        .wr_en_op_05(wr_en_op_05),
                                        .wr_en_op_06(wr_en_op_06)
                                        );

reg_file r1(.clock(clock),
                                        .reset(reset),
                                        .ra_addr_ep(ra_addr_ep_r),
                                        .rb_addr_ep(rb_addr_ep_r),
                                        .rc_addr_ep(rc_addr_ep_r),
                                        .ra_addr_op(ra_addr_op_r),
                                        .rb_addr_op(rb_addr_op_r),
                                        .rc_addr_op(rc_addr_op_r),
                                        .rt_addr_ep(out_ep_rt_addr), // RF change
                                        .rt_addr_op(out_op_rt_addr), // RF change
                                        .rt_data_ep(out_ep_rt_data), // RF change
                                        .rt_data_op(out_op_rt_data), // RF change
                                        .wrbe_ep(out_ep_wr_en_rt),
                                        .wrbe_op(out_op_wr_en_rt),
                                        .o_ra_data_ep(ra_rf_data_ep),
                                        .o_rb_data_ep(rb_rf_data_ep),
                                        .o_rc_data_ep(rc_rf_data_ep),
                                        .o_ra_data_op(ra_rf_data_op),
                                        .o_rb_data_op(rb_rf_data_op),
```

```
                                        .o_rc_data_op(rc_rf_data_op),
                                        .reg_file_mem(reg_file_mem));// Debug

        local_store l1(.clock(clock),
                                                .lsa_addr(ls_addr_rd_out),
                                                .lsa_data_in(ls_data_out),
                                                .lsa_data_out(ls_data_rd_in),
                                                .ls_wr_en(ls_wr_en_op),
                                                .ls_mem(ls_mem));

endmodule
```

k. Testbench.sv:

```
import definitions::*;

module tb_part2();

        logic clock, reset;
        topmodule_part2 dut(.clock(clock),
                                                .reset(reset)
                                                );


        initial clock = 0;
        always #5 clock = ~clock;


        initial begin

                // Before first clock edge, initialize
                reset = 1;
                @(posedge clock);
                reset = 0;
                @(posedge clock);

                repeat(1000) @(posedge clock);
                $finish;
        end
```

```
endmodule
```