ESE 545 Computer Architecture

Cell SPU-Lite Project Report

Due on Wed. May 2nd, 2018

Professor: Mikhail Dorojevets

Ning Kang #111577912
Weilun Cheng #111633412

1. Introduction: SPU-Lite Model

The SPU core is a SIMD RISC style processor with 128 general purpose registers. Every general purpose register is 128 bit wide, which allows for multiple data operation. All the instructions are encoded into 32 bit fixed length instruction formats which simplify pipeline instructions. Instructions are sent, two at a time, from the Cache to the issue control unit. The SPU-Lite issues and completes all instruction in program order and does not reorder or rename its instructions. The SPU-Lite provides hardware to complete instruction issue and distribution, thus software does not require NOP padding when dual issue is not possible. Instructions are fetched from Cache if Cache hit. Otherwise, Cache replacement will be executed to load instructions from Local Storage if Cache miss, which performs as main memory. Cache only stores instructions. Operands are fetched from register or forward network and sent to the execution components. Load and Store operations transfer 16 bytes of data between the register and the Local Store. In this project, we implements the SPU-Lite model by using Verilog on ModelSim. We follows incremental development principle to guarantee every step correct. Before coding start, instructions set are chose for function requirement and reviewed by professor. In the step 2, the basic pipeline architecture are implemented. And then more components to implement complicate functions, such as two instruction distribution, stall control, data forwarding, memory access, floating operations, etc. Finally, the 22 matrix floating multiply are verified on the DUT.

2. Basic Pipeline Design

The SPU-Lite follows 5 stages pipeline architecture. However, the purpose of simulation of the different latency for different instructions need to append multiple forward modules between execution module and memory access module. Thus the actual SPU-Lite is deeper pipeline architecture. The basic modules in the pipepline stages and its source code file name is shown in Figure 1 as below. The connection between the modules is too complicated to show here.

More detail about the architecture is described as below.

(1). Fetch

PC module will provide the address of instruction, in which the value of PC register is the address of instruction. The implementation is in pc_reg.v.

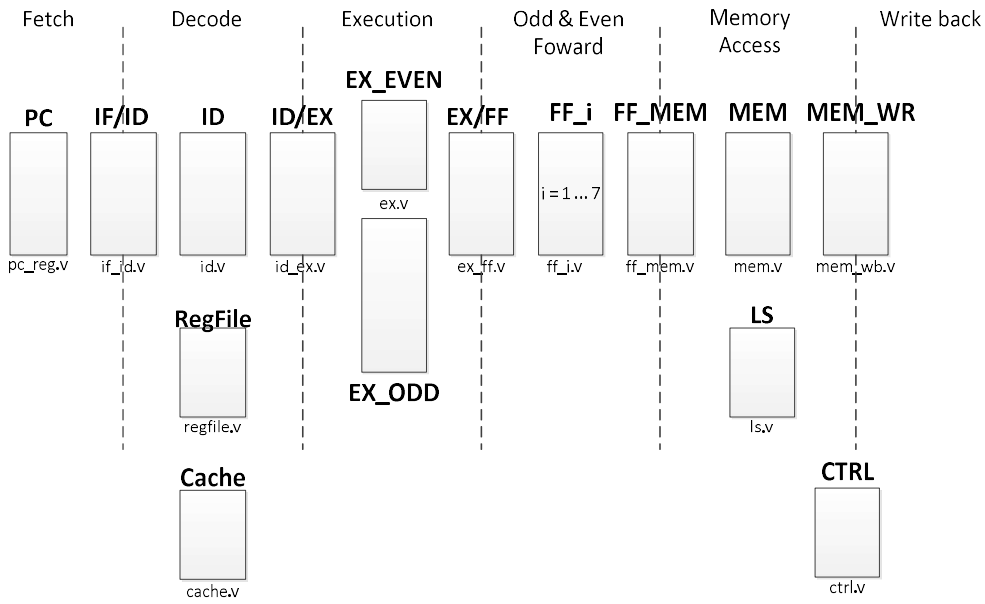


Figure 1. Modules and its file of Stages in SPU-Lite Pipeline

IF/ID module transfer the results, like fetched instructions, address, etc., to decode stage in next clock cycle. It is implemented in if_id.v.

(2). Decode

ID module decode the instructions and generate operation code, operands, and RT register address, etc. At the same time, ID module need to distribute these data to even pipe and odd pipe and send stall request to CTRL module if the two instructions both are even pipe or odd pipe operation.

Regfile module implements 128 registers, each has 128 bits wide. The implementation is in regfile.v.

ID/EX module transfer the ID results to EX module in the next clock cycle. The implementation is in

id_ex.v.

(3). Execution

EX module performs the operation according to the results of ID module. Here we split EX into EX_EVEN and EX_ODD to execute even pipe and one odd pipe at the same time if the fetched instruction happened to be decoded into one even instruction and one odd instruction.

EX/MEM transfers the executed results to MEM module in the next clock cycle. The implementation is in file ex_mem.v.

(4). FF1 to FF7

To simulate the latency of the instruction execution, FF1 to FF7 modules appended after EXE module even if the executed results have been obtained in the execution clock cycle. These modules simply forwards results to the next stage in every clock cycle. The implementation is in ff1.v to ff7.v

(5). Memory Access

MEM module will access to local store if the operation is instruction load or store. The implementation is in file mem.v.

(6). Write Back

MEM/WB transfers results to register file during the next clock cycle. It is implemented in mem_wb.v.

Besides the modules above, we could see CTRL module in the diagram, which is used for pipeline stall and clearance. It is in ctrl.v.

We also have 4K byte Cache which used for store instructions in cache module. It is implemented in cache.v. The hit and miss function are implemented. A cache stall request will send to CTRL module to stall the pipeline if instruction misses and the stall request will be cleared if hits. Finally, the LS module performs as Local storage to store instructions with 32 bits wide and data with 128 bits wide. The LS totally is 256K bytes and splits into instruction sector and data sector to avoid structural hazard. It is implemented in file ls.v.

We will take ORI instruction as an example to test pipeline function. Besides the implementation modules we discussed as above, we only highlights the related codes about ORI here. In ID stage, the 64 bits wide 2 instructions were split to inst_l and inst_h with 32 bit wide. To simplify the testbench, we put the ORI in inst_l and pad a LNOP in ints_h. Although the opcode of ORI only has 8 bits, the longest opcode in SPU ISA is 11 bits. Thus we unified the opcode wide to 11 bits and sensitively compare the 11 bits. Thus the decode code is shown like below.

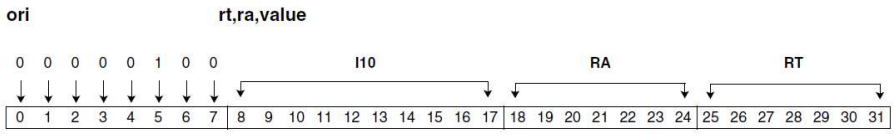
```
always @ (*) begin
    casez (op_l)
        11'b00000100???: begin      //EXE_ORI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_ORI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            imm_e <= {{22{inst_l[8]}},inst_l[8:17]}; //word: 32b
            rt_addr_e <= inst_l[25:31];
        end
        default: begin
        end
    endcase
end

always @ (*) begin
    casez (op_h)
        11'b00000000001: begin      //EXE_LNOP
            pipe_h = `PIPE_ODD;
            uid_o <= `UID_0;
            wreg_o = `WR_DISABLE;
            op_code_o <= `EXE_LNOP;
            ra_rd_o <= `RD_DISABLE;
            rb_rd_o <= `RD_DISABLE;
            rt_addr_o <= 0;
        end
        default: begin
        end
    endcase
end
```

In execution stage, operation will be executed based on operand RA and operand RB (an immediate operand in this case) according to the results of decode stage. Here we only show the ORI operation code as below.

```
always @ (*) begin
    if(rst == `RST_ENABLE) begin
        o_rt_e <= `ZERO_QWORD128;
    end else begin
        case (i_opcode_e)
            end
            `EXE_ORI: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= i_ra_e[`WORD0] | i_rb_e[`WORD0];
                o_rt_e[`WORD1] <= i_ra_e[`WORD1] | i_rb_e[`WORD1];
                o_rt_e[`WORD2] <= i_ra_e[`WORD2] | i_rb_e[`WORD2];
                o_rt_e[`WORD3] <= i_ra_e[`WORD3] | i_rb_e[`WORD3];
            end
            default:begin
                o_rt_e <= `ZERO_QWORD128;
            end
        endcase
    end
end
```

The top-level module SPU is implemented in spu.v, which is used to comlete instantiation and connection for those modules in pipeline. Then we could connect SPU, Cache, and LS to build up a whole system as SPE module. The complete source code could be referred in appendix B.

According to SPU ISA datasheet, the ORI instruction code should be constructed as below.



According to the instruction format as above, we could manually generate binary code of the test program. The test program is shown in the left side as below, the transformation to Hexadecimal code is shown in the right side. The lower 64 bits 0x0020000 is the LNOP instruction to avoid two

instructions fetch and distribution stall issue. We have to create a file cache.data to store program we coded as below.

```
ori r1, r0, 0x11    # r1 = 0x0 | 0x0011 = 0x11 -> rt = 0x0011001100110011      04044001002000000
ori r2, r0, 0x22    # r2 = 0x0 | 0x0022 = 0x22 -> rt = 0x0022002200220022      0408800200200000
ori r3, r0, 0x33    # r3 = 0x0 | 0x0033 = 0x33 -> rt = 0x0033003300330033      040cc00300200000
ori r4, r0, 0x44    # r4 = 0x0 | 0x0044 = 0x44 -> rt = 0x0044004400440044      0411000400200000
ori r5, r0, 0x55    # r5 = 0x0 | 0x0055 = 0x55 -> rt = 0x0055005500550055      0415400500200000
ori r6, r0, 0x66    # r6 = 0x0 | 0x0066 = 0x66 -> rt = 0x0066006600660066      0419800600200000
ori r7, r0, 0x77    # r7 = 0x0 | 0x0077 = 0x77 -> rt = 0x0077007700770077      041dc00700200000
ori r8, r0, 0x88    # r8 = 0x0 | 0x0088 = 0x88 -> rt = 0x0088008800880088      0422000800200000
ori r9, r0, 0x99    # r9 = 0x0 | 0x0099 = 0x99 -> rt = 0x0099009900990099      0426400900200000
ori r10, r0, 0x3ff  # r10 = 0x0 | 0x3ff = 0x3ff -> rt = 0xffffffffffffffff     04ffc00a00200000
```

Finally, we need to create the test bench, in which provides the clock signal (1 ns), reset signal. The code is shown as below. Now we could start simulation to verify the functionality of the SPU-Lite. Once the simulation is done, we could observe the wave diagram to check whether or not the SPU-Lite runs correctly.

```
//*************************************************************************************/
// Module:      SPE
// File:        spe.v
// Description: the top level file of SPE
// History:     Created by Ning Kang, Mar 31, 2018
//*************************************************************************************/

//test bench indicate the signal of clock and the reset.
`timescale 1ns/1ps

module TB_SPE;

    reg CLOCK_50;
    reg rst;

    initial begin
            CLOCK_50 = 1'b0;
            forever #10 CLOCK_50 = ~CLOCK_50;
    end

    initial begin
            rst = 1'b1;
            #195 rst= 1'b0;
            #1000 $stop;
    end

    SPE dut(
        .clk(CLOCK_50),
        .rst(rst)
);

endmodule
```

The wave diagram of the test program is shown as below.

Figure 2 (a). SPU-Lite Pipeline Stages – Instruction Fetch

(1) At the first rising edge of reset signal, SPU-Lite starts to fetch instruction. The first fetched

instruction is 404400100200000, which will send to the port of if_inst. Decode will start in the next clock cycle.

(2) In decode stage, the instructions code splits into two individual instructions: 04044001(ORI) and 00200000(LNOP). Now, the id_pc is 0x0000000, and if_pc is 0x00000008, fetch stage starts to fetch the next instructions code 0408800200200000. The operands in RA and RB are generated. In this case, RA stored four 32 bits zero and RB stored four 32bits 0x00000011. The value of wreg_e is 1, which means the executed result should be written back to register file. The value of rt_addr_e is 7h'01, which means the executed result will be written back to r1. Because the instruction in odd pipe is LNOP, we ignore its value as all the operands and result would be zeros.

(3) In the execution stage, the result is obtained as 0x00000011000000011000000011000000011 as we expected. Similarly, the third instruction has been fetched, and the second instruction decode completes.

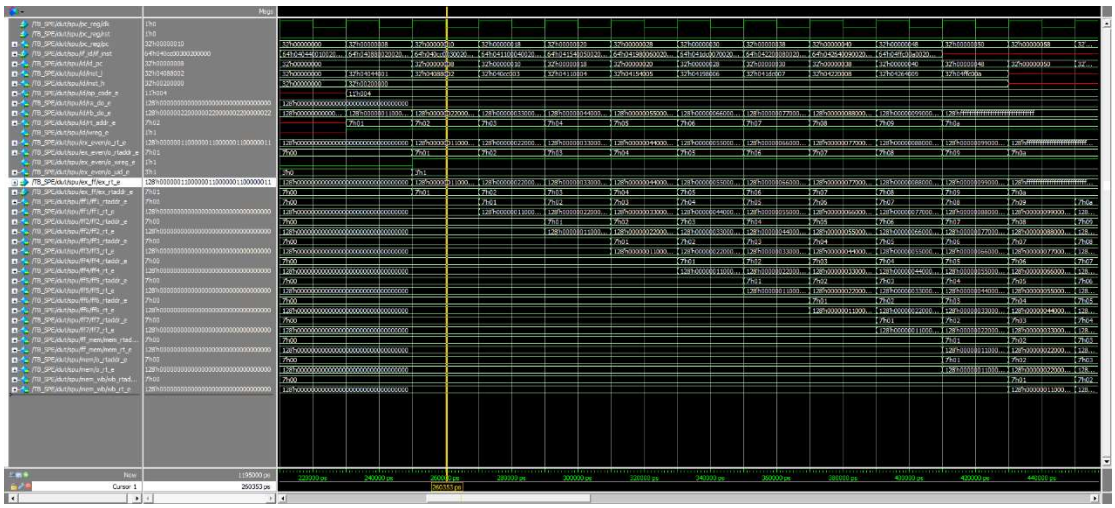


Figure 2(b). SPU-Lite Pipeline Stages - Execution

(4) The forward stages only simply forwards the executed results to next stage. By observing the waveform, we know that the pipelines works well.

Figure 2(c). SPU-Lite Pipeline Stages – Memory Access

(5) In memory access stage, ORI instruction doesn't need to access local storage, so it is only simply forwards executed results to Write Back stage. By observing diagram Figure 2(c), it turns out that all the instruction operation results have been completed till the memory access stage of the first instruction.

(6) By observing the later clock cycles, we could know that the last instruction result generated and its register address is 0xa, which completely match our expectation.

Till now, we completed the basic model build-up for SPU-Lite, in which the pipeline mechanism has been implemented and the simulation proves that it works very well.

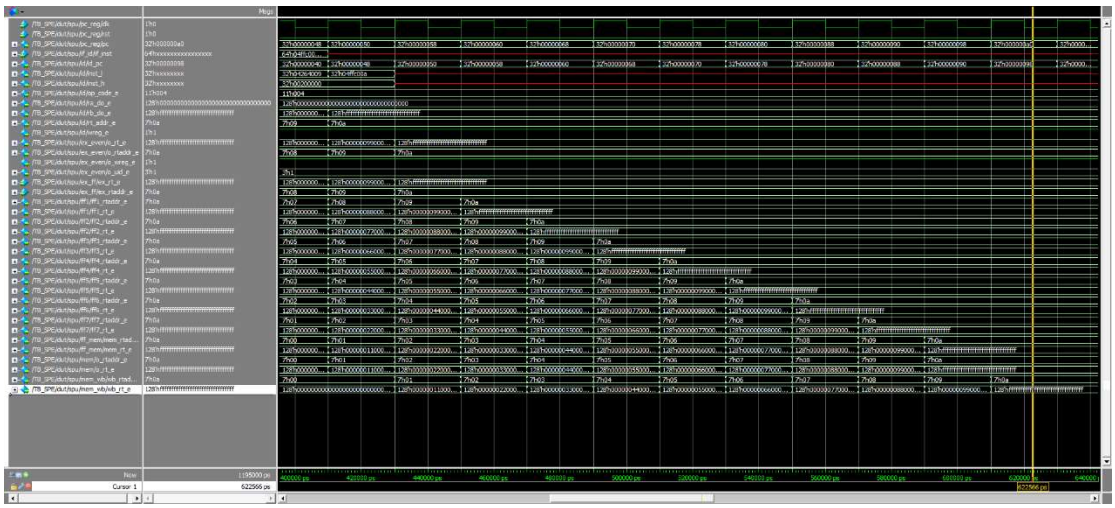


Figure 2(d) SPU-Lite Pipeline Stages – Write Back

3. Instruction Sets: Design and Implementation

In SPU-Lite ISA, we have several type of the instruction sets. According to their types, we defined their Unit id to distinguish their latencies. The classification of ISA includes:

(1). Load/Store: lqd, stqd;

For load and store instruction, we defined UID as 5, and their latency is 6.

(2).Arithmetic: ah, ahi, a, ai, sfh, sfhi, sf, sfhi, sf, sfi;

For arithmetic operations, we defined UID as 1, and their latency is 2.

(3).Multiply-accumulate: mpy, mpyi;

For multiply accumulate operations, we defined UID as 4, and their latency is 7.

(4).Logic: and, andbi, andhi, andi, or, orbi, orhi, ori, xor, xorbi, xori, nand, nor;

For logic operations, we defined unit id as 1, and their latency is 2.

(5).Shift and Rotate: shlqbi, shiqbii, rotqby, rotqbyi, cbd;

For shift and rotate operations, we defined unit id as 4, and their latency is 4.

(6).Compare: ceqb, ceqbi, ceqh, ceqhi, ceq, ceqi, cgtb, cgtbi, cgth, cgthi, cgt, cgti;

For compare operations, same as logic operations, we defined unit id as 1, and latency is 2.

(7).Branch: bra, brsl, brasl, bi, bisl, brz, brhnz, brhz;

For branch instructions, the unit id is defined as 7, and the latency is 4.

(8). Floating point: fa, fs, fm, fceq, fcgt;

For floating point instructions, the unit id is defined as 2 and the latency is 6.

(9). Byte: cntb, avgb;

For byte instructions, the unit id is defined as 3 and their latency is 4.

(10). NOP/LNOP

For nop and lnop operations, the unit id is defined as 0 and their latency is 0.

The complete instruction sets and their RTL implementation could be referred in Appendix A.

We will implemented all the instructions besides load, store, branch and floating points in this section. Load and store instruction will be discussed and implemented in cache and memory section. Branch instructions will be discussed and implemented in Branch section. Floating points will be in the section of floating point instruction design.

First, we define all the operation code as macro in defines.v. Only part of macro definition is shown

as below. The complete source code could by find in Appendix B.

```
//Instructions
`define EXE_LQD      8'b00110100
`define EXE_STQD     8'b00100100
`define EXE_AH       11'b00011001000
`define EXE_AHI      8'b00011101
`define EXE_A        11'b00011000000
`define EXE_AI       8'b00011100
`define EXE_SFH      11'b00001001000
`define EXE_SFHI     8'b00001101
`define EXE_SF       11'b00001000000
`define EXE_SFI      8'b00001100
`define EXE_MPY      11'b01111000100
`define EXE_MPYU     11'b01111001100
`define EXE_MPYI     8'b01110100
`define EXE_AND      11'b00011000001
`define EXE_ANDBI    8'b00010110
`define EXE_ANDHI    8'b00010101
`define EXE_ANDI     8'b00010100
`define EXE_OR       11'b00001000001
`define EXE_ORBI     8'b00000110
`define EXE_ORHI     8'b00000101
`define EXE_ORI      8'b00000100
`define EXE_XOR      11'b01001000001
`define EXE_XORBI    8'b01000110
`define EXE_XORHI    8'b01000101
`define EXE_XORI     8'b01000100
`define EXE_NAND     11'b00011001001
`define EXE_NOR      11'b00001001001
`define EXE_SHLQBI   11'b00111011011
`define EXE_SHLQBII  11'b00111111011
`define EXE_ROTQBY   11'b00111011111
`define EXE_ROTQBYI  11'b00111111111
`define EXE_CBD      11'b00111110100
`define EXE_CEQB     11'b01111010000
`define EXE_CEQBI    8'b01111110
`define EXE_CEQH     11'b01111001000
`define EXE_CEQHI    8'b01111101
`define EXE_CEQ      11'b01111000000
`define EXE_CEQI     8'b01111100
`define EXE_CGTB     11'b01001010000
`define EXE_CGTBI    8'b01001110
`define EXE_CGTH     11'b01001001000
`define EXE_CGTHI    8'b01001101
`define EXE_CGT      11'b01001000000
`define EXE_CGTI     8'b01001100
```

Similarly as we discussed in section 2, in the instructions decode stage, we need to sensitively compare the higher 11 bits to decide what operation it is. Because PC register fetched two instructions and send to decode module, thus the decode module need to split the two instruction into two individual instructions and decode every instruction to decide the operation code. The decode source code is too long to show here, we only provide part of them as an example, the complete source code could be find in Appendix B.

```
always @ (*) begin
    casez (op_l)
        11'b00110100???: begin      //EXE_LQD
            pipe_l = `PIPE_ODD;
            uid_o <= `UID_5;   //latency=6
            wreg_o = `WR_ENABLE;
            op_code_o <= `EXE_LQD;
            ra_rd_o <= `RD_ENABLE;
            rb_rd_o <= `RD_DISABLE;
            ra_addr_o <= inst_l[18:24];
            imm_o <= {{22{inst_l[8]}},inst_l[8:17]};
            rt_addr_o <= inst_l[25:31];
        end
        11'b00100100???: begin      //EXE_STQD
            pipe_l = `PIPE_ODD;
            uid_o <= `UID_5;   //latency=6
            wreg_o = `WR_ENABLE;
            op_code_o <= `EXE_STQD;
            ra_rd_o <= `RD_ENABLE;
            rb_rd_o <= `RD_DISABLE;
            ra_addr_o <= inst_l[18:24];
            imm_o <= {{22{inst_l[8]}},inst_l[8:17]};
            rt_addr_o <= inst_l[25:31];
        end
        11'b00011001000: begin      //EXE_AH
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_AH;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b00011101???: begin      //EXE_AHI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_AHI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            imm_e <= {{6{inst_l[8]}},inst_l[8:17]}; //HalfWord:16b
            rt_addr_e <= inst_l[25:31];
        end
        11'b00011100???: begin      //EXE_AI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_AI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            imm_e <= {{22{inst_l[8]}},inst_l[8:17]}; //Word:32b
            rt_addr_e <= inst_l[25:31];
        end
        11'b00001001000: begin      //EXE_SFH
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SFH;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b00001101???: begin      //EXE_SFHI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SFHI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            imm_e <= {{6{inst_l[8]}},inst_l[8:17]}; //HalfWOrd: 16b
            rt_addr_e <= inst_l[25:31];
        end
        11'b00001000000: begin      //EXE_SF
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SF;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b00001100???: begin      //EXE_SFI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SFI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            imm_e <= {{22{inst_l[8]}},inst_l[8:17]};
            rt_addr_e <= inst_l[25:31];
        end
```

For any fetched instruction, the main purpose of decode includes: 1. confirm the registers of two operands, 2. what execution it is, 3. target register where the executed results should be written back. We take several typical instruction to explain this process.

1. AH: Add Halfword

   (1). Operands registers: AH requires to read the value of register RA and RB, so we set ra_rd_e as 1 and rb_rd_e as 1. In SPU-Lite, all the register address is 7 bits wide, which allows system access to totally 128 registers. By default the address of RA is bit 18 to bit 24 of instruction, and the address of RB is bit 11 to bit 17 of instruction.

   (2). Execution type: AH instruction is half word add, which allows 8 half words add in the total 128 bits operands. We set the case as 00011001000 to compare with instruction bit 0 to bit 10.

   (3). Address of target register: AH instruction need to write the executed result to target address, so we need to set wreg_e to 1, and the rt_addr_e is bit 25 to 31 of the instruction where will store the result.

2. AHI: Add Halfword Immediate

   (1). Operands registers: AHI only need to read the value of register RA, so we set ra_rd_e as 1 and rb_rd_e as 0. By default the address of RA is bit 18 to bit 24 of instruction. The value 0 of rb_rd_e implicates that immediate would be used as the operand. The imm_e is the value of immediate after extended to 128 bits.

   (2). Execution type: AHI instruction is half immediate add, so we set wreg_e to 1, and the rt_addr_e is bit 25 to 31 of the instruction where will store the result.

The operand RB or immediate number is obtained depending on the read enable signal. The code

```
// Obtain even pipe operand ra & rb
always @ (*) begin
    if(rst == `RST_ENABLE) begin
        ra_do_e <= `ZERO_QWORD128;
    end else if ((ra_rd_e == `RD_ENABLE) && (ex_wreg_i_e == `WR_ENABLE) && (ex_waddr_i_e == ra_addr_e)) begin //fowarding data from ex module
        ra_do_e <= ex_wdata_i_e;
    end else if ((ra_rd_e == `RD_ENABLE) && (mem_wreg_i_e == `WR_ENABLE) && (mem_waddr_i_e == ra_addr_e)) begin //fowarding data from mem
        ra_do_e <= mem_wdata_i_e;
    end else if(ra_rd_e == 1'b1) begin
        ra_do_e <= ra_i_e;
    end else if(ra_rd_e == 1'b0) begin
        ra_do_e <= imm_e;
    end else begin
            ra_do_e <= `ZERO_QWORD128;
    end
end

always @ (*) begin
    if(rst == `RST_ENABLE) begin
        rb_do_e <= `ZERO_QWORD128;
    end else if ((rb_rd_e == `RD_ENABLE) && (ex_wreg_i_e == `WR_ENABLE) && (ex_waddr_i_e == rb_addr_e)) begin //fowarding data from ex module
        rb_do_e <= ex_wdata_i_e;
    end else if ((rb_rd_e == `RD_ENABLE) && (mem_wreg_i_e == `WR_ENABLE) && (mem_waddr_i_e == rb_addr_e)) begin //fowarding data from mem
        rb_do_e <= mem_wdata_i_e;
    end else if(rb_rd_e == 1'b1) begin
        rb_do_e <= rb_i_e;
    end else if(rb_rd_e == 1'b0) begin
        rb_do_e <= imm_e;
    end else begin
            rb_do_e <= `ZERO_QWORD128;
    end
end
```

is shown as below.

The execution code need to be added into EX code. We only show AH and AHI as below, complete source code could be referred in Appendix B.

```
`EXE_AH: begin
    o_wreg_e <= i_wreg_e;
    o_rtaddr_e <= i_rtaddr_e;
    o_uid_e <= i_uid_e;
    //o_even <= `PIPE_EVEN;
    o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] + i_rb_e[`HALFWORD0];
    o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] + i_rb_e[`HALFWORD1];
    o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] + i_rb_e[`HALFWORD2];
    o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] + i_rb_e[`HALFWORD3];
    o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] + i_rb_e[`HALFWORD4];
    o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] + i_rb_e[`HALFWORD5];
    o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] + i_rb_e[`HALFWORD6];
    o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] + i_rb_e[`HALFWORD7];
end
`EXE_AHI: begin
    o_wreg_e <= i_wreg_e;
    o_rtaddr_e <= i_rtaddr_e;
    o_uid_e <= i_uid_e;
    //o_even <= `PIPE_EVEN;
    o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] + i_rb_e[`HALFWORD0];
    o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] + i_rb_e[`HALFWORD1];
    o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] + i_rb_e[`HALFWORD2];
    o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] + i_rb_e[`HALFWORD3];
    o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] + i_rb_e[`HALFWORD4];
    o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] + i_rb_e[`HALFWORD5];
    o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] + i_rb_e[`HALFWORD6];
    o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] + i_rb_e[`HALFWORD7];
```

Before the verification, some register need to initialize for specific operations. Now we could prepare the test program to verify our instructions. To avoid dual instructions distribution issue, we padded LNOP to these instruction to combine them to code of 64 bits two instructions.

```
#instruction test program
#initia r0=1,r1=0x00110011001100110011001100110011,r2=0x22,
#            r3=0x33333333333333333333333333333333,
#            r4=0x44444444444444444444444444444444,
#            r5=0x55005500550055005500550055005500,
#            r6=0x00000066000000660000006600000066,
#            r7=0x55555555555555555555555555555555,
#            r8=0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa,
#            r9=0x00005555000055550000555500005555,
#           r10=0x01550155015501550155015501550155,
#           r11=0x00560056005600560056005600560056,
#           r12=0x00001111000011110000111100001111,
#           r13=0x55550000555500005555000055550000,
#           r14=0x00000000111111110000000011111111,
#           r15=0x55555555000000005555555500000000,
ah r64,r0,r1        #r0+r1=0x00110011001100110011001100110012
ahi r65,r1,0x10     #r1+10=0x00210021002100210021002100210021
a r66,r1,r2         #r1+r2=0x00110011001100110011001100110033
ai r67,r1,0x11      #r1+10=0x00110011001100110011001100110022
sfh r68,r0,r1       #r1+r0=0x00110011001100110011001100110010
sfhi r69,r1,0x12    #12-r1=0x00010001000100010001000100010001
sf r70,r3,r4        #r4-r3=0x11111111111111111111111111111111
sfi r71,r0,0x10     #10-r0=0x00000010000000100000001000000000f
mpy r72,r0,r5       #r0*r1=0x00550055005500550055005500550055
mpyi r73,r6,0x1     #r0*1 =0x00660066006600660066006600660066
and r74,r0,r1       #r0&r1=0x00000000000000000000000000000001
andbi r75,r7,0x1    #r7&1 =0x00000001000000010000000100000001
andhi r76,r7,0x1    #r7&1 =0x00010001000100010001000100010001
andi r77,r7,0x1     #r7&1 =0x00000001000000010000000100000001
or r78,r7,r1        #r7|r1=0x55555555555555555555555555555555
orbi r79,r8,1       #r7|1 =0xaaaaaaabaaaaaaabaaaaaaabaaaaaaab
orhi r80,r8,1       #r7|1 =0xaaabaaabaaabaaabaaabaaabaaabaaab
ori r81,r8,1        #r7|1 =0xaaaaaaabaaaaaaabaaaaaaabaaaaaaab
xor r82,r7,r8       #r7^r8=0xffffffffffffffffffffffffffffffff
xorbi r83,r8,0x1    #r8^1 =0xaaaaaaabaaaaaaabaaaaaaabaaaaaaab
xorhi r84,r8,0x1    #r5^1 =0xaaabaaabaaabaaabaaabaaabaaabaaab
xori r85,r8,0x1     #r5^1 =0xaaaaaaabaaaaaaabaaaaaaabaaaaaaab
nand r86,r7,r8      #~(r5&r8)=0xffffffffffffffffffffffffffffffff
nor r87,r7,r8       #~(r7|r8)=0x00000000000000000000000000000000
ceqb r88,r7,r9      # 0x0000ffff00ffff00ffff00ffff00ffff
ceqbi r89,r10,0x02  # 0xff00ff00ff00ff00ff00ff00ff00ff00
ceqh r90,r7,r9      # 0x0000ffff0000ffff0000ffff0000ffff
ceqhi r91,r10,0x55  # 0x00000000000000000000000000000000
ceq r92,r9,r10      # 0x00000000000000000000000000000000
ceqi r93,r11,0x55   # 0x00000000000000000000000000000000
cgtb r94,r10,r11    # 0xff00ff00ff00ff00ff00ff00ff00ff00
cgtbi r95,r10,0x02  # 0x00ff00ff00ff00ff00ff00ff00ff00ff
cgth r96,r12,r13    # 0x0000ffff0000ffff0000ffff0000ffff
cgthi r97,r13,0x1   # 0xffff0000ffff0000ffff0000ffff0000
cgt r98,r14,r15     # 0x00000000ffffffff00000000ffffffff
cgti r99,r15,0x1    # 0xffffffff00000000ffffffff00000000
```

After simulation, we could obtain the wave output as below. The executed results could be read from the register o_rt_e. According to the operations execution order, we check the results cycle by cycle. It turns out that all the operation results are exactly the result we expected. It proves that our instruction implementation is correct.

Figure 3. Instruction Functionality Verification

4. Dual-Instruction Fetch and Decode Data Hazard

In the previous test program, we padded an instruction of LNOP to the prior instruction if it is EVEN instruction, which would fill the odd pipe. Otherwise an instruction of NOP would be padded to fill the even pipe. However, in practical program, SPU-Lite doesn't know the prior instruction is EVEN or ODD unless compiler recognizes the instruction and reorder or pad NOP to meet the requirement of dual instruction fetch and issue. Therefore, hardware need to be design to solve this problem. The thinking about dual-instruction fetch is to resend the fetched instruction again if the two instructions both is odd or even. The methodology is described as below.

(1). Check the two instruction operation code. As the code we discussed above, PC register fetches two instructions in one 64 bits wide code. In the decode stage, we need to split this code into two single instructions, which is inst_l from bit 0 to 31 and inst_h from the bit 32 to 63. What need to pay attention here is that the inst_l is always the prior instruction, and the inst_h is the following instruction.

(2). Check the execution is even pipe or odd pipe. In the decode stage, besides operation code, operands, write enable bit, target address, etc. we also need to mark the pipe tag as EVEN or ODD, which will send back to fetch stage and be used for following instruction decode.

(3). Send stall request to CTRL module. In decode stage, if pipe of inst_l and inst_h is same, say, they both are EVEN pipe or both ODD pipe, a stall request need to be set and send to CTRL module. Once CTRL module received the stall request, the stall signal will send to PC_REG to stall the instruction fetch stage.

(4). Pad the fetched instruction higher 32 bits with NOP and resend to decode stage. Now, instruction fetch has been stalled. The event happened in decode stage also has been obtained from pipe_l and pipe_h. Thus the next instructions code need to be combined by the higher bits 32 to 63 and a NOP as even pipe or a LNOP as ODD pipe. We defined this instructions code as inst_echo.

(5). Clear stall request. According to our design, the inst_echo must be one even instruction and one odd instruction. So if this condition meets, we clear the stall request to let the pipeline going on.

The signal connection between multiple modules is shown in the diagram as below. More detail about stall mechanism would be discussed in section 5.

The diagram of dual-instruction issue is shown as below, in which includes CTRL modules for stall

signal distribution and stall request collection.

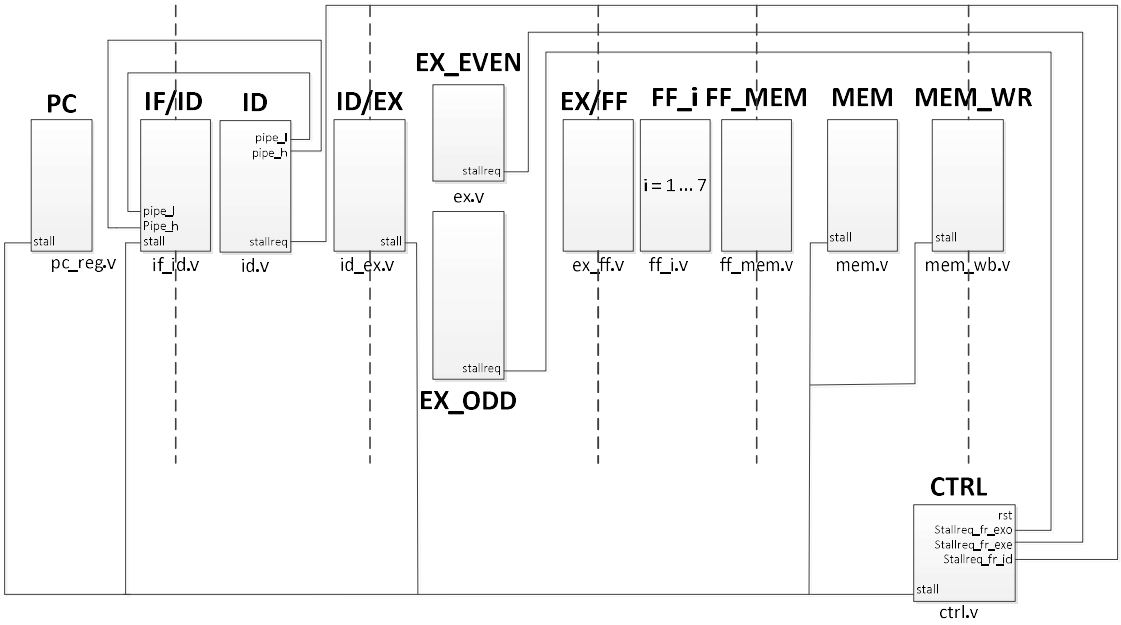


Figure 3. Dual-Instruction Decode & Issue

According to the design we discussed above, multiple modules code need to modify to implement dual-instruction issue and PC stall.

In PC_REG module, if the stall equals 7, where we defined 7 in defines.v as the stall request from ID module, the value of PC won't add 8 as usual to get the next new instructions code. PC register keeps its previous value to guarantee the correctness of the program order.

```
always @ (posedge clk) begin
    if (ce == `CHIP_DISABLE) begin
        pc <= `ZERO_WORD32;
    end else if (stall[0] == `NOSTOP) begin
        if(branch_flag_i == `BRANCH) begin
            pc <= branch_target_addr_i;
        end else if (stall == 13'h7) begin
            pc <= pc;
        end else begin
            pc <= pc + 4'h8;
        end
    end
end
```

In IF_ID modules, we creates a 64 bits register to store the present instructions code when there is no stall. When stall request from ID module sets, ID module will send pipe information and stall bits to IF_ID, then what instructions code should be forwarded. As discussed above, if the two operations decoded by ID module are ODD pipe, we padded a NOP with the higher bits instruction; if the two operations are EVEN pipe, we padded a LNOP with the higher bits instruction.

```
always @ (posedge clk) begin
    if ( rst == `RST_ENABLE ) begin
        id_pc <= `ZERO_WORD32;
        id_inst <= `ZERO_DWORD64;
    end else if (stall[1] == `STOP && stall[2] == `NOSTOP) begin //
        id_pc <= `ZERO_WORD32;
        id_inst <= `ZERO_DWORD64;
    //end else if ((stall[2] == `STOP) && (stall[3] == `NOSTOP))begin
    end else if ((stall == 13'h7) && (pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_ODD) ) begin
        id_pc <= if_pc;
        id_inst <= {inst_echo[32:63], 32'h40200000}; // resend previous 2 instructions to ID
    end else if ((stall == 13'h7) && (pipe_l==`PIPE_EVEN)&&(pipe_h==`PIPE_EVEN) ) begin
        id_pc <= if_pc;
        id_inst <= {inst_echo[32:63], 32'h00200000}; // resend previous 2 instructions to ID
    end else if (stall[1] == `NOSTOP) begin
        id_pc <= if_pc;
        id_inst <= if_inst;
        inst_echo <= if_inst; // reserve current 2 instructions
    end
end
```

In the decode module, we already know that it is even pipe or odd pipe of lower bits instruction and higher bits instruction. Therefore, we compare these two variables with pipe odd and pipe even to decide whether or not stall request need to send to CTRL module.

```
always @ (*) begin
    // Do not need to consider these 2 conditions because EVEN and
    //odd pipe variables already split in Decode module
    //if ((pipe_h==`PIPE_EVEN)&&(pipe_h==`PIPE_ODD))begin end
    //if ((pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_EVEN)) begin end

    if ((pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_ODD)) begin
        stallreq <= 1; //stall == 7
    end else if ((pipe_l==`PIPE_EVEN)&&(pipe_h==`PIPE_EVEN)) begin
        stallreq <= 1; //stall == 7
    end else begin
        stallreq <= 0;
    end
end
```

Besides that, we also have to add CTRL module to send and collect stall signals to support dual instructions decode and issue. Further discussion would be done in section 5 when we design data hazard resolution.

```
//*************************************************
// Module:      Control
// File:        CTRL.v
// Description: CTRL module for pipeline stall
// History:     Created by Ning Kang, Apr 20, 2018
//*************************************************

`include "defines.v"

/*
stall <= 13'b0000000001  //PC holds
stall <= 13'b0000000011  //IF stall
stall <= 13'b0000000111  //ID stall
stall <= 13'b0000001111  //EX stall
stall <= 13'b0000011111  //MEM stall
stall <= 13'b0000111111  //WR stall
*/
module CTRL(
    input rst,
    input wire stallreq_fr_id,
    input wire stallreq_fr_ex,
    output reg [0:12] stall
);

    always @ (*) begin
        if (rst == `RST_ENABLE) begin
            stall <= 13'b0000000000;
        end else if (stallreq_fr_ex == `STOP) begin
            stall <= 13'b0000001111;
        end else if (stallreq_fr_id == `STOP) begin
            stall <= 13'b0000000111;
        end else begin
            stall <= 13'b0000000000;
        end
    end
endmodule
```

Now code preparation is done, simulation could start. We still use the program in section 3, but

don't pad LNOP with the EVEN pipe instructions. So every 64 bits instructions send to ID module is two even pipe instructions, which require stall and send instruction echo from IF_ID to ID again with an ODD pipe no operation.

```
#instruction test program                                                    #hex
#initia r0=1,r1=0x0011001100110011001100110011,r2=0x22,
#           r3=0x33333333333333333333333333333333,
#           r4=0x44444444444444444444444444444444,                          190000e4
#           r5=0x55005500550055005500550055005500,                          1d0400e5
#           r6=0x00000066000000660000006600000066,                          180080e6
#           r7=0x55555555555555555555555555555555,                          1c0440e7
#           r8=0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa,                          09004068
#           r9=0x00005555000055550000555500005555,                          0d0480e9
#          r10=0x01550155015501550155015501550155,                          080101f0
#          r11=0x00560056005600560056005600560056,                          0c040047
#           r12=0x00001111000011110000111100001111,                         78814048
#           r13=0x55550000555500005555000055550000,                         74004349
#           r14=0x00000000111111110000000011111111,                         1820404a
#           r15=0x55555555000000005555555500000000,                         160043cb
ah r64,r0,r1          #r0+r1=0x0011001100110011001100110012                 150043cc
ahi r65,r1,0x10       #r1+10=0x00210021002100210021002100210021             140043cd
a r66,r1,r2           #r1+r2=0x0011001100110011001100110033                 082043ce
ai r67,r1,0x11        #r1+10=0x0011001100110011001100110022                 0600444f
sfh r68,r0,r1         #r1+r0=0x0011001100110011001100110010                 05004450
sfhi r69,r1,0x12      #12-r1=0x0001000100010001000100010001                 04004451
sf r70,r3,r4          #r4-r3=0x11111111111111111111111111111111             482203d2
sfi r71,r0,0x10       #10-r0=0x0000001000000010000000100000000f             46004453
mpy r72,r0,r5         #r0*r1=0x0055005500550055005500550055                 45004454
mpyi r73,r6,0x1       #r0*1 =0x0066006600660066006600660066                 44004455
and r74,r0,r1         #r0&r1=0x0000000000000000000000000000                 192203d6
andbi r75,r7,0x1      #r7&1 =0x0000000100000001000000010000                 092203d7
andhi r76,r7,0x1      #r7&1 =0x0001000100010001000100010001                 7a0243d8
andi r77,r7,0x1       #r7&1 =0x0000000100000001000000010000                 7e008559
or r78,r7,r1          #r7|r1=0x55555555555555555555555555555555             790243da
orbi r79,r8,1         #r7|1 =0xaaaaaaaabaaaaaaabaaaaaaabaaaaaab              7d15455b
orhi r80,r8,1         #r7|1 =0xaaabaaabaaabaaabaaabaaabaaabaaab              780284dc
ori r81,r8,1          #r7|1 =0xaaaaaaaabaaaaaaabaaaaaaabaaaaaab              7c1545dd
xor r82,r7,r8         #r7^r8=0xffffffffffffffffffffffffffffffff             4a02c55e
xorbi r83,r8,0x1      #r8^1 =0xaaaaaaaabaaaaaaabaaaaaaabaaaaaab              4e00855f
xorhi r84,r8,0x1      #r5^1 =0xaaabaaabaaabaaabaaabaaabaaabaaab              49034660
xori r85,r8,0x1       #r5^1 =0xaaaaaaaabaaaaaaabaaaaaaabaaaaaab              4d0046e1
nand r86,r7,r8        #~(r5&r8)=0xffffffffffffffffffffffffffffffff           4803c762
nor r87,r7,r8         #~(r7|r8)=0x00000000000000000000000000000000           4c0047e3
ceqb r88,r7,r9        # 0x0000ffff00ffff00ffff00ffff00ffff
ceqbi r89,r10,0x02    # 0xff00ff00ff00ff00ff00ff00ff00ff00
ceqh r90,r7,r9        # 0x0000ffff0000ffff0000ffff0000ffff
ceqhi r91,r10,0x55    # 0x00000000000000000000000000000000
ceq r92,r9,r10        # 0x00000000000000000000000000000000
ceqi r93,r11,0x55     # 0x00000000000000000000000000000000
cgtb r94,r10,r11      # 0xff00ff00ff00ff00ff00ff00ff00ff00
cgtbi r95,r10,0x02    # 0x00ff00ff00ff00ff00ff00ff00ff00ff
cgth r96,r12,r13      # 0x0000ffff0000ffff0000ffff0000ffff
cgthi r97,r13,0x1     # 0xffff0000ffff0000ffff0000ffff0000
cgt r98,r14,r15       # 0x00000000ffffffff00000000ffffffff
cgti r99,r15,0x1      # 0xffffffff00000000ffffffff00000000
```

In the right side above, the transformation of program to hexadecimal code is shown for instruction code stored in cache and it would be easy to observe its functionality.

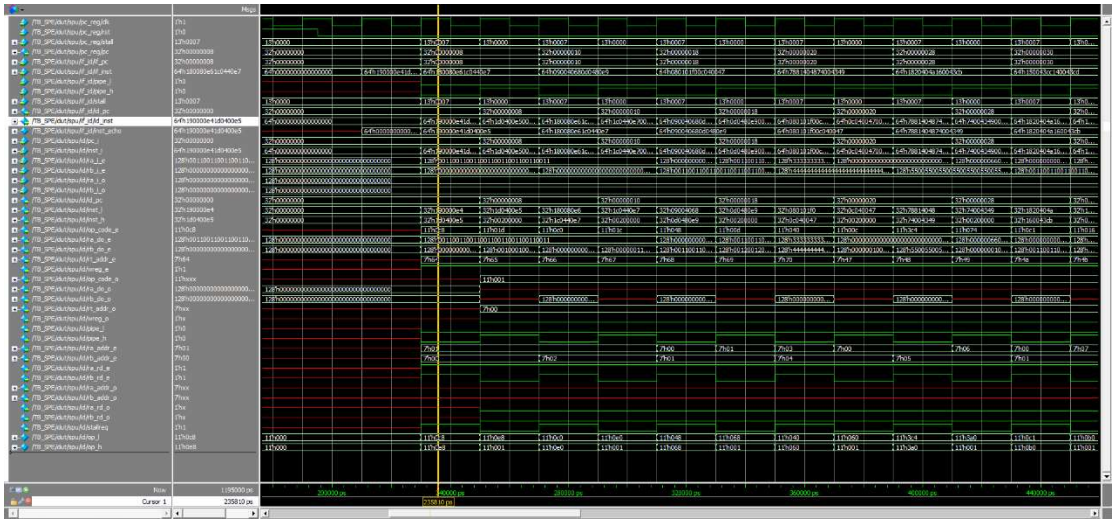


Figure 3. (a) Dual-Instruction Decode & Issue

From the figure shown above, we could see that in the first clock cycle, PC register fetched the first

64 bits instructions 0x190000e41d0400e5 in fetch stage, and split them into inst_l and inst_h in the second clock cycle where is in decode stage. In this clock cycle, the inst_echo stored the previous instructions 0x190000e41d0400e5 although the following instructions 0x180080e61c0440e7 has been fetched to register of if_inst.

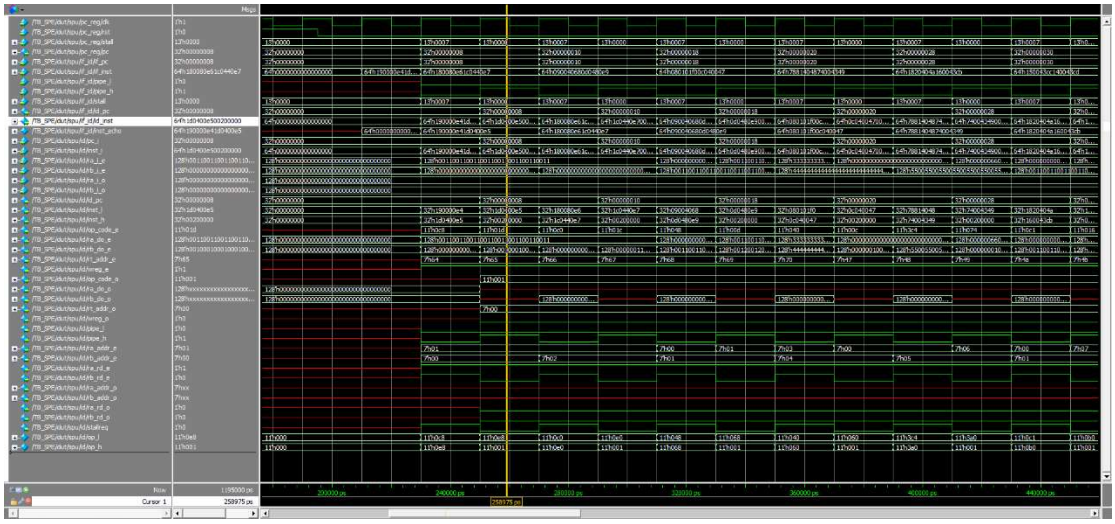


Figure 3. (b) Dual-Instruction Decode & Issue

In third clock cycle, we noticed that no new instruction fetch because stall register had been set to 0x0007 in the second clock cycle. The ID module in this clock cycle fetches instructions from inst_echo, which is 0x1d0400e500200000 that is combined by the inst_h and LNOP. At the same time, because two instructions is in different pipe, so the stall request set to 0 and the stall bits reset to zeros in this clock cycle.

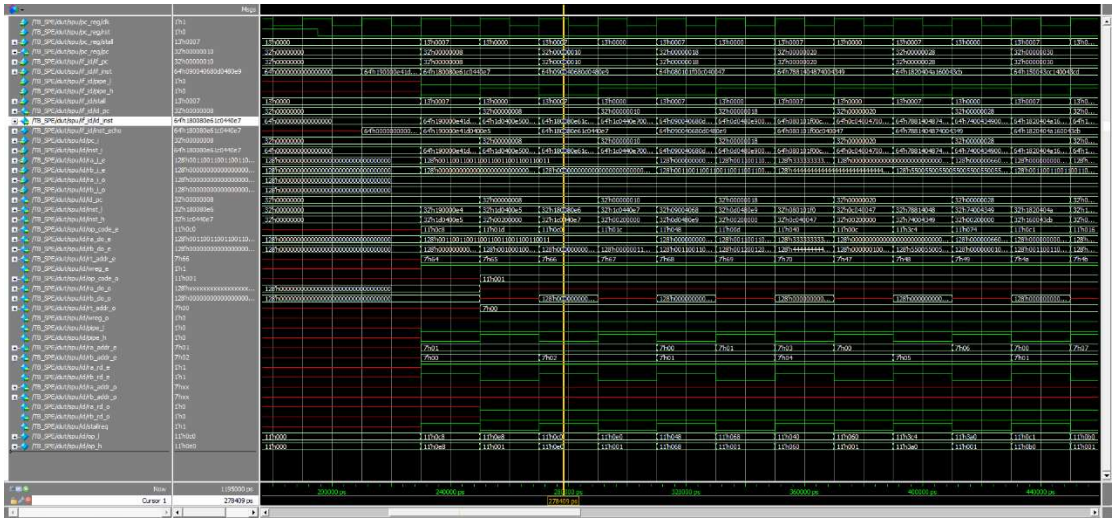


Figure 3. (c) Dual-Instruction Decode & Issue

In fourth clock cycle, the blocked instructions 0x180080e61c044e7 in third clock cycle transfer to

decode stage and because these two instructions are both even pipe operation, the stall bits set to 0x0007 again to block PC increments.

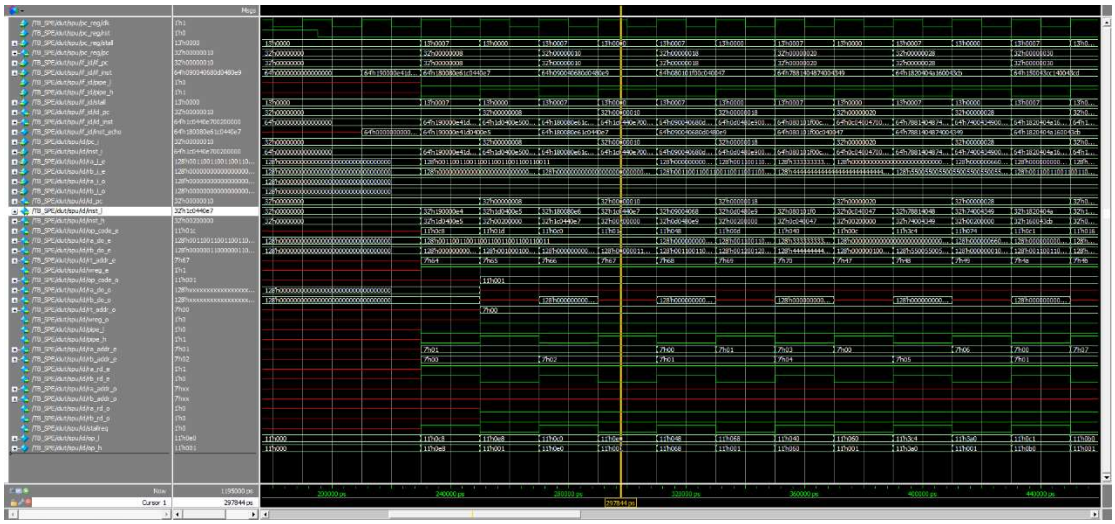


Figure 3. (d) Dual-Instruction Decode & Issue

Again, instructions split to two instructions and pad LNOP to lower 32 bits instruction. The higher 32 bits instruction padding with LNOP resend to decode stage again to keep the instructions sequence in ordering. Till now, it turns out that the dual-instructions decode and issue works well. There is no instruction being missed and all the intructions are fetched in order. By moving to following clock cycle one by one to check the intruction fetch, decode and execution, we find that it completely meets the test program oder as we expected.

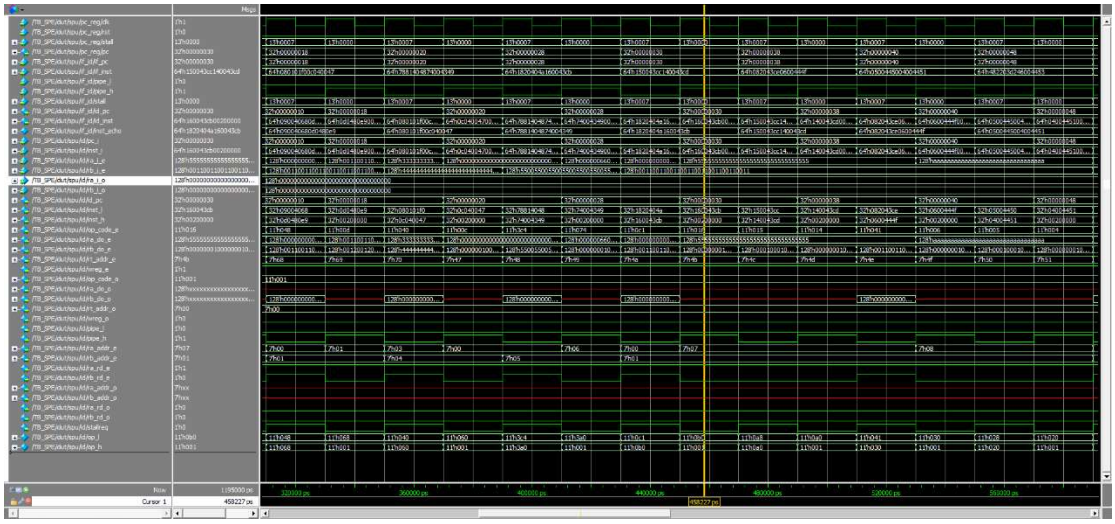


Figure 3. (e) Dual-Instruction Decode & Issue

5. Data Hazard Resolution: Forwarding and Stall

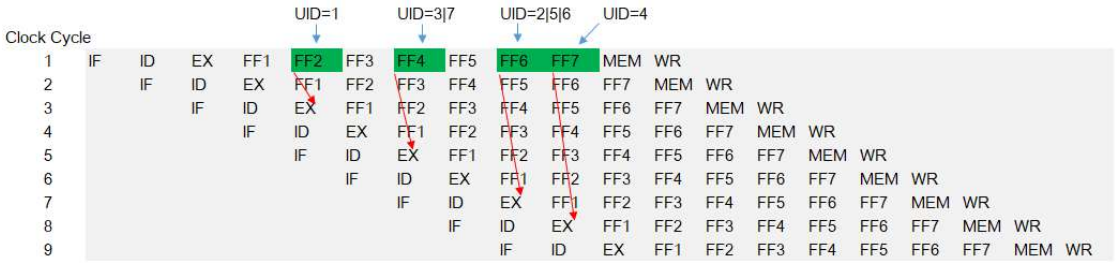
In the pipeline, data hazard will happen if the operands of one instruction depends on previous instruction executed results. The data hazard of pipeline includes: RAW, WAR, WAR.

1. RAW: Read After Write. Assuming the instruction j executed after instruction i, RAW means instruction j only could read data from the register after the instruction i write back data to it. If instruction try to read the data before instruction i completes write back, an error data would be read out.
2. WAR: Write After Read. Assuming the instruction j executed after instruction i, WAR means instruction j only could write to the register after instruction i read it. If instruction j write the register before instruction i read it, an error data would be read out.
3. WAW: Write After Write. Assuming the instruction j executed after instruction i, WAW means instruction j only could write data to the register after instruction i write data back to it. If instruction j write it before instruction i write, the value of register would be error.

For SPU-Lite, the register write operation only happens in write back stage, thus there is no WAW hazard. Furthermore, as the register read only happens in decode stage and register write only happens in write back stage, thus there is no WAR hazards. Therefore, SPU-Lite only exists RAW hazards. According to the UID definition and its latency in ISA, we classify the latency of instructions as below. Because different instruction has different latency, thus the execution stage only could obtain previous executed resulted after multiple clock cycles according to the latency clock cycles.

UID	Latency
1	2
2	6
3	4
4	7
5	6
6	6
7	4

The figure in the below shows that when the executed result could be forwarded in the forwarding network. By observing this pipeline in below, it turns out that for operation with UID equals 2, the executed result could be sent to execution after 2 clock cycles. If UID equals 3 or 7, the executed results only could be forwarded after 4 clock cycles. Similarly, if UID equals 2, 5 or 6, the executed results could be forwarded after 6 clock cycles. And if UID equals 7, the executed result could be forwarded after 7 clock cycles.



However, if the instruction j in clock cycle 2 requires the instruction executed result in clock cycle 1 with UID as 1. What could we do to avoid data hazard? The simple methodology is stall. Take the instruction with UID 2 as an example. If the instruction in clock cycle 2 need to read the register data which is exactly the target register in clock cycle 1, the EX module need to send stall request to CTRL module and set stall bits to binary 13'b0000000001111. And then CTRL module send stall bits to PC_REG module to stall the PC increment until 2 clock cycles passed. After 2 clock cycles, PC register increment resumes and EX module need to clear stall request to let pipeline going on. Similarly, we could set stall and clear it after multiple clock cycles for other UID operations. Generally, for an instruction with UID 1 executes in clock cycle 3, operands only could be forwarded from FF2 if the one of the operands address equals the target address in previous EX stage or FF1 stage. For an instruction with UID 2, 5 or 6 executes in clock cycle 7, operands only could be forwarded from FF6 if the one of the operands address equals the target address in previous stage of EX, FF1, FF2, FF3, FF4 or FF5. For an instruction with UID 4 executes in clock cycle 8, operands only could be forwarded from FF7 if the one of the operands address equals the target address in previous stage of EX, FF1, FF2, FF3, FF4, FF5, FF6. Based on this methodology, the diagram of SPU-Lite need to update as below.

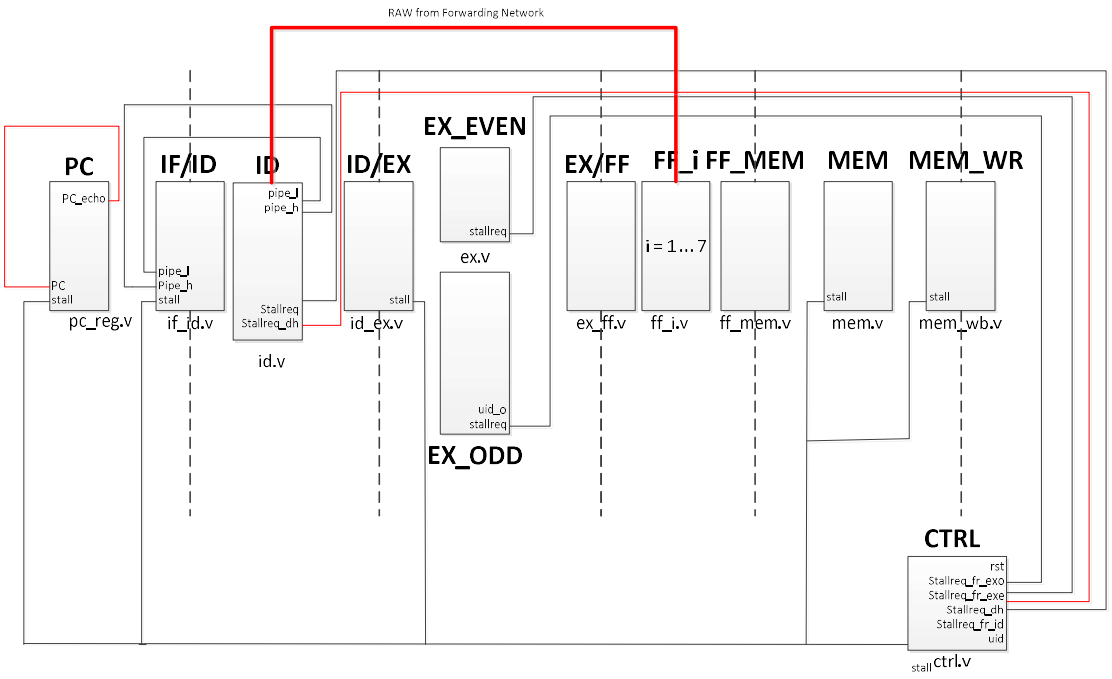


Figure 4. Data Forwarding

In this diagram, the bold red line indicates lines of target address, executed result, register enable bit of the forward network. The full connection between forward network and decode module is too complicated to completely show in this diagram. Furthermore, a specific stall request signal, which works as the signal to report data hazard, need to send to CTRL module. Correspondingly, CTRL module need to set stall bits and broadcast it to PC module to stall PC increment. By the way, because the value of PC has been incremented two time after 2 clock cycle. Here we added a register PC_echo to store the previous PC value and send back PC if the stall request is data hazard stall.

The code modification to avoid data hazard is mainly in ID module.

At first, the forward network output about target address, register writer enable and executed result need to be added in the input wire of ID module. We could leverage these inputs to decide that which stage of forward network could be used as the data forwarding source. We only take the RA register in even pipe as an example to explain the forwarding logic, As the source code we show in below. Other operands is very similar like this process, please refer to the complete source code in Appendix B. As the source code show in below, if the UID of current operation equals 1, and address of register A equals the target register address in FF2, then we let the value of RA registers equals the value of executed result in FF2. In this case, if the address of register RA equals the target register address in FF1 or EX module, which means the previous operation is still on-going, then the value

```
// Obtain even pipe operand ra & rb
always @ (*) begin
    if(rst == `RST_ENABLE) begin
        ra_do_e <= `ZERO_QWORD128;
    end else if ((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_1)&&(ff2_wreg_i_e==`WR_ENABLE)&&(ff2_waddr_i_e==ra_addr_e)) begin //fowarding data from FF network
        ra_do_e <= ff2_rt_e;
        stallreq_dh <= 0;
    end else if (((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_waddr_i_e==ra_addr_e))) begin
        ra_do_e <= `ZERO_QWORD128;
        stallreq_dh <= 1;
    end else if ((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_3)&&(ff4_wreg_i_e==`WR_ENABLE)&&(ff4_waddr_i_e==ra_addr_e)) begin
        ra_do_e <= ff4_rt_e;
        stallreq_dh <= 0;
    end else if (((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&
            (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))) begin
        ra_do_e <= `ZERO_QWORD128;
        stallreq_dh <= 1;
    end else if ((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_2)&&(ff6_wreg_i_e==`WR_ENABLE)&&(ff6_waddr_i_e==ra_addr_e)) begin
        ra_do_e <= ff6_rt_e;
        stallreq_dh <= 0;
    end else if (((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&
            (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||
            ((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))) begin
        ra_do_e <= `ZERO_QWORD128;
        stallreq_dh <= 1;
    end else if ((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_4)&&(ff7_wreg_i_e==`WR_ENABLE)&&(ff7_waddr_i_e==ra_addr_e)) begin
        ra_do_e <= ff7_rt_e;
        stallreq_dh <= 0;
    end else if (((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&
            (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||
            ((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff6_waddr_i_e==ra_addr_e))) begin
        ra_do_e <= `ZERO_QWORD128;
        stallreq_dh <= 1;
    end else if ((ra_rd_e==`RD_ENABLE)&&(mem_wreg_i_e==`WR_ENABLE)&&(mem_waddr_i_e==ra_addr_e)) begin //fowarding data from mem
        ra_do_e <= mem_wdata_i_e;
        stallreq_dh <= 0;
    end else if (((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&
            (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||
            ((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff6_waddr_i_e==ra_addr_e))||
            ((ra_rd_e==`RD_ENABLE)&&(ff7_waddr_i_e==ra_addr_e))) begin
        ra_do_e <= `ZERO_QWORD128;
        stallreq_dh <= 1;
    end else if(ra_rd_e==1'b1) begin
        ra_do_e <= ra_i_e;
    end else if(ra_rd_e==1'b0) begin
        ra_do_e <= imm_e;

    end
end
```

can't be forwarded from these two module. Thus we need to set stall request to 1 and send to CTRL module. Similarly, for other UID operation with different latency, we need to compare the target register address in the forward network with the address of operand register. If the equality is established, the value of target register could directly forward to the operand. Otherwise, stall need to be added until the condition meets.

Now the test program could be prepare to verify the functionality of data forward and stall. The test program and initialization of r1, r2 is shown as below.

```
#data hazard test program
r1=0x00010001000100010001000100010001
r2=0x00020002000200020002000200020002
ah r10,r2,r1         #r10=r0+r1 =0x00030003000300030003000300030003
ah r11,r10,r1        #r11=r10+r1=0x00040004000400040004000400040004
ah r12,r11,r1        #r12=r11+r1=0x00050005000500050005000500050005
```

In this program, the operands r10 of the second AH instruction depends on the executed result of the first instruction. The third AH operation depends on the result of second instruction. It is a typical RAW in practical. After simulation, we obtained the wave form as below. It turns out that in the second clock cycle, ID module decoded the first instruction and the execution completed in the third clock cycle. However, as we discussed before, we only could forward the result from the forward network according to the UID of operation to simulate its latency. Therefore, ID module send a stall request to CTRL, and CTRL module distribute it to PC_REG module to stall the pipeline. After 4 clock cycle as we expected, result of the first instruction forwarded to register RA and the

Figure 5. (a) Data Hazard: Forwarding and Stall

stall request is cleared. Now the second instruction obtained correct value of register RA and completes execution. The result is 0x0004000400040004000400040004 as we expected.

Figure 5. (b) Data Hazard: Forwarding and Stall

Similarly, the third instruction obtained result of the second instruction after 4 clock cycles. The compute result is 0x0005000500050005000500050005, which is the value we expected.

This simple test program proved that our data forwarding and stall mechanism works well and the modularized implementation is very helpful to design and debug.

6. Branch Instruction Design

To reduce the consumption during branch instructions, SPU-Lite check the branch condition in decode stage. If the condition meets branch requirement, then we calculate target address in execution stage and send it back to PC register in FF4 module because of the latency of branch operation is 4. The update of value of PC register includes 3 situation.

(1). PC equals PC+8. This is the most general condition. PC plus 8 to point to next two instructions during every clock cycle.

(2). PC keeps current value. This condition will happen when pipeline stall. We have discussed the stall mechanism in previous section.

(3). PC equals the target address, which generates from result of the operation of branch instruction. If it is a branch instruction and branch condition meets the requirement, then the target address should be assigned to PC register.

To implement branch instructions, we need to modify the system diagram with some extended ports. What we have to pay attention here is that the executed result only could be sent back to PC in FF4 stage because of branch instruction latency, even if the results already have be obtained in the unit of odd execution. You can refer to the diagram as below.

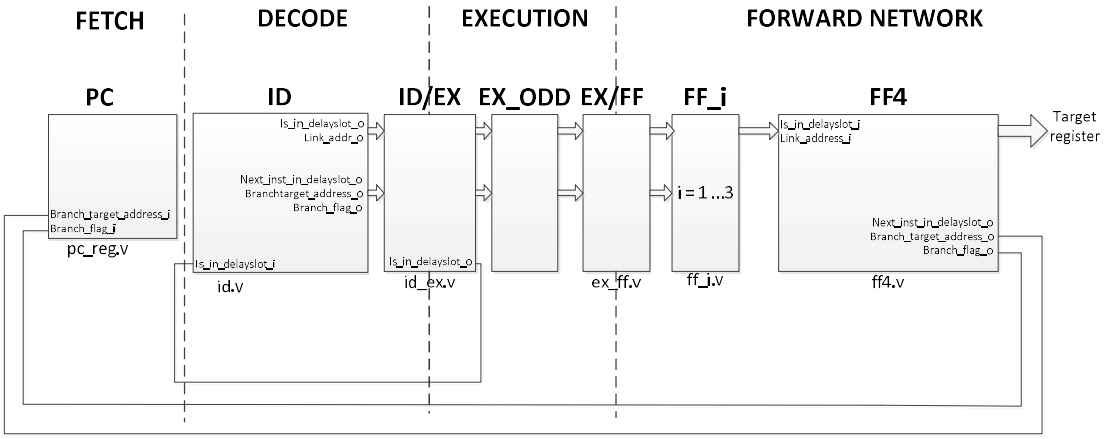


Figure 6. SPU-Lite diagram for Branch Implementation

In the PC module, two input ports, branch target address and branch flag, need to be added. Here the main purpose is to obtain the target branch address if branch flag had been set when the condition of branch triggered.

```
always @ (posedge clk) begin
    if (ce == `CHIP_DISABLE) begin
        pc = `ZERO_WORD32;
    end else if (stall[0] == `NOSTOP) begin
        if((branch_flag_i == `BRANCH) &&(i==0)) begin
            pc = branch_target_addr_i;
            i = 1;
        end else if (stall==13'h7) begin
            pc = pc;
        end else if (stall==13'h8) begin
            pc = pc_echo;
        end else begin
            pc = pc + 4'h8;
            pc_echo = pc-4'h8;
        end
    end
end
```

The code modification in ID stage mainly is to add the ports of branch information, which should be send to forward network. We only shows several cases of branch instruction decode as below because the complete code is too long to show here. Similarly, the ID/EX module need to transfer these information to execution unit. The code modification is very easy and only simply transfer the data to next stage, so we don't show this part of code in this section. You could refer to the complete code in Appendix B.

```
11'b00110010??: begin      //EXE_BR
    pipe_1 = `PIPE_ODD;
    uid_o <= `UID_7;
    wreg_o = `WR_DISABLE;
    op_code_o <= `EXE_BR;
    ra_rd_o <= `RD_DISABLE;
    rb_rd_o <= `RD_DISABLE;
    next_inst_in_delayslot_o <= `IN_DELAYSLOT;
    branch_flag_o <= `BRANCH;
    branch_target_addr_o <= {{14{inst_1[9]}},inst_1[9:24],2'b00}; // Extended on the right with 2 0bits
    link_addr_o <= `ZERO_WORD32;
end
11'b00110000??: begin      //EXE_BRA
    pipe_1 = `PIPE_ODD;
    uid_o <= `UID_7;
    wreg_o = `WR_DISABLE;
    op_code_o <= `EXE_BRA;
    ra_rd_o <= `RD_DISABLE;
    rb_rd_o <= `RD_DISABLE;
    next_inst_in_delayslot_o <= `IN_DELAYSLOT;
    branch_flag_o <= `BRANCH;
    branch_target_addr_o <= {{14{inst_1[9]}},inst_1[9:24],2'b00};
    link_addr_o <= `ZERO_WORD32;
end
11'b00110011??: begin      //EXE_BRSL
    pipe_1 = `PIPE_ODD;
    uid_o <= `UID_7;
    wreg_o = `WR_ENABLE;
    op_code_o <= `EXE_BRSL;
    ra_rd_o <= `RD_DISABLE;
    rb_rd_o <= `RD_DISABLE;
    rt_addr_o <= inst_1[25:31];
    next_inst_in_delayslot_o <= `IN_DELAYSLOT;
    branch_flag_o <= `BRANCH;
    branch_target_addr_o <= {{14{inst_1[9]}},inst_1[9:24],2'b00};
    link_addr_o <= `ZERO_WORD32;
end
11'b00110001??: begin      //EXE_BRASL
    pipe_1 = `PIPE_ODD;
    uid_o <= `UID_7;
    wreg_o = `WR_ENABLE;
    op_code_o <= `EXE_BRASL;
    ra_rd_o <= `RD_DISABLE;
    rb_rd_o <= `RD_DISABLE;
    rt_addr_o <= inst_1[25:31];
    next_inst_in_delayslot_o <= `IN_DELAYSLOT;
    branch_flag_o <= `BRANCH;
    branch_target_addr_o <= {{14{inst_1[9]}},inst_1[9:24],2'b00};
    link_addr_o <= `ZERO_WORD32;
```

Because all of branch instructions are type of odd pipe, therefore, the code modification in EX module only happens in the module of EX_ODD.

```
`EXE_BR:begin
    o_uid_o <= i_uid_o;
    //o_odd <= `PIPE_ODD;
    o_ex_branch_flag <= i_ex_branch_flag;
    o_ex_branch_target_addr <= i_ex_branch_target_addr;
    o_ex_link_addr <= i_ex_link_addr;
    o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
end
`EXE_BRA:begin
    o_uid_o <= i_uid_o;
    //o_odd <= `PIPE_ODD;
    o_ex_branch_flag <= i_ex_branch_flag;
    o_ex_branch_target_addr <= i_ex_branch_target_addr;
    o_ex_link_addr <= i_ex_link_addr;
    o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
end
`EXE_BRSL:begin
    o_uid_o <= i_uid_o;
    //o_odd <= `PIPE_ODD;
    o_rt_o <= ((ex_pc + 4) & 128'h00000000000000000000000ffffffff); //RT0:3 = pc+4; RT4:15=0
    o_ex_branch_flag <= i_ex_branch_flag;
    o_ex_branch_target_addr <= i_ex_branch_target_addr;
    o_ex_link_addr <= i_ex_link_addr;
    o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
end
`EXE_BRASL:begin
    o_uid_o <= i_uid_o;
    //o_odd <= `PIPE_ODD;
    o_rt_o <= ((ex_pc + 4) & 128'h00000000000000000000000ffffffff); //RT0:3 = pc+4; RT4:15=0
    o_ex_branch_flag <= i_ex_branch_flag;
    o_ex_branch_target_addr <= ex_pc + i_ex_branch_target_addr;
    o_ex_link_addr <= i_ex_link_addr;
    o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
end
`EXE_BI:begin
    o_uid_o <= i_uid_o;
    //o_odd <= `PIPE_ODD;
    o_ex_branch_flag <= i_ex_branch_flag;
    o_ex_branch_target_addr <= (i_ra_o[0:31] & 32'hfffffffc);
    o_ex_link_addr <= i_ex_link_addr;
    o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
end
```

In the following stages, from EX_FF to FF4, we just simply send information of target branch address, branch flag, etc. to next stages. We don't show code of every stage here. In top level file spu.v, the connection between module FF4 and PC_REG has to be established to transfer the branch flag and target branch address.

Now we could prepare test program to verify the functionality of branch instructions. Here we take the instruction of BRA as an example. The test code is shown as below.

```
#branch
nop
lnop
ah r10,r2,r1        #r10=r0+r1 =0x00030003000300030003000300030003
lnop
nop
lnop
nop
lnop
nop
bra  0x1            #r10=r0+r1 =0x00030003000300030003000300030003
```

After simulation we obtained the test result and wave diagram as below.

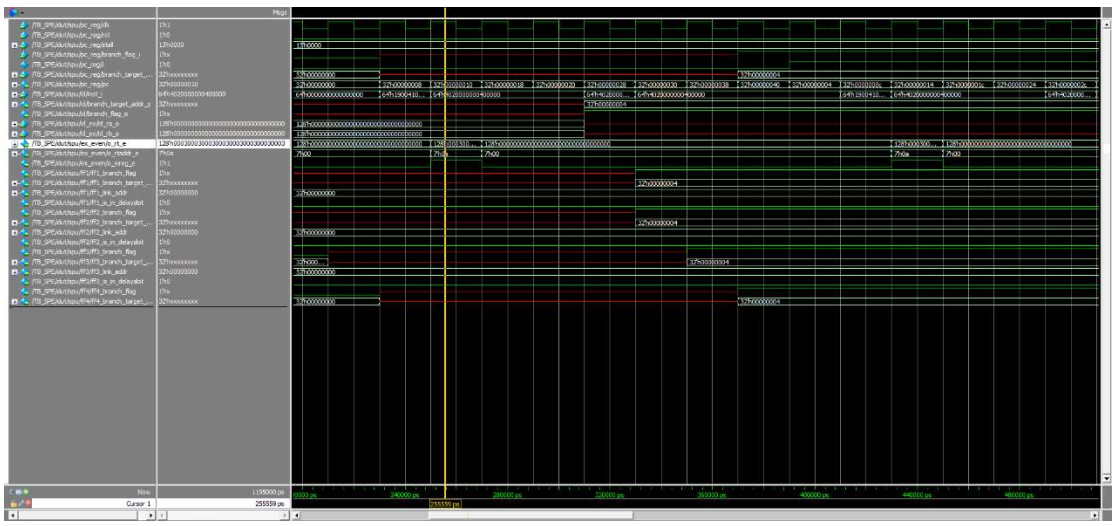


Figure 7(a). Branch

We could observe the PC register in the diagram. In the third clock cycle, the first AH instruction completes calculation and obtains the correct result 0x0003000300030003000300030003. During the following clock cycles, the value of PC register continues to increment. After 5 clock cycles, the PC value incremented to 0x40 and here the PC fetches the fifth instruction, the BRA.

By observing the waveform diagram of Figure 7(b), it shows that the PC value jumps to 0x4 as we expected, and fetches the second AH instruction. Thus the AH calculation executes again and obtains the calculation result 0x0003000300030003000300030003 after 3 clock cycles in the execution stage.

The test program proved that the branch instructions functionality works well and they could

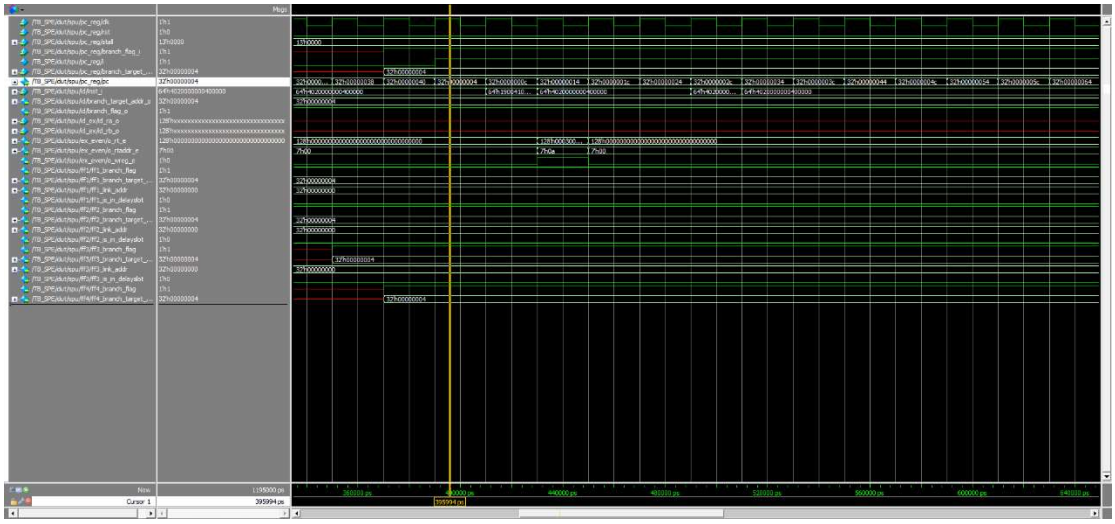


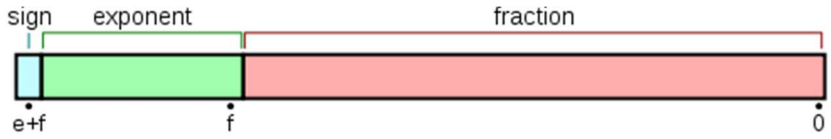
Figure 7(b). Branch

execute correctly to generate the results as the test program expected.

7. Floating Points Instruction Design

1. Floating Points Format

We use the floating points format as the IEEE 754. For 32-bit single floating points, the most 1st bit is sign bit, the 2nd -9th bits as the exponent field, the 10th – 32th as the significand field.



2. Floating Points Add

1) First, we judge the sign and exponents of two operands, to decide the operation for the significand field and the sign of result.

2) Then compare the exponents to calculate the diff between two operands. Move the significand field of the min operand right for diff bits, making two operands have the same exponents. Use the round ceil method to calculate the significand part.

3) Join the sign, exponents fields and significand field together, get the calculation result.

```
for (i=0;i<4;i=i+1) begin
        if (ra_temp_word[i][0]==rb_temp_word[i][0])
                begin
                //judge tha max and min of two operands
                if (ra_temp_word[i][1:9]>rb_temp_word[i][1:9]) begin
                        max=ra_temp_word[i];
                        min=rb_temp_word[i];
                end
                else begin
                        min=ra_temp_word[i];
                        max=rb_temp_word[i];
                end
                //the exponoents diff
                float_diff=max[1:9]-min[1:9];
                //resize the min operand
                min[9:31]=min[9:31]>>float_diff;
                rt_temp_word[i][9:31]=max[9:31]+min[9:31];
                rt_temp_word[i][0:8]=max[0:8];
                end
        else begin
                if (ra_temp_word[i][1:9]>rb_temp_word[i][1:9]) begin
                        max=ra_temp_word[i];
                        min=rb_temp_word[i];
                end
                else begin
                        min=ra_temp_word[i];
                        max=rb_temp_word[i];
                end
                float_diff=max[1:9]-min[1:9];
                min[9:31]=min[9:31]>>float_diff;
                rt_temp_word[i][9:31]=max[9:31]-min[9:31];
                rt_temp_word[i][0:8]=max[0:8];
                end
        end
```

3. Floating Points Multiply

For Floating points multiply, it's much easier the adding.

1) First, calculate the sign of result =A.sign ~^ B.sign

2) Second, add the exponents of two operands together

3) Finally, multiply two significand fields, get the result.

```
`EXE_FM: begin
        o_wreg_e<=i_wreg_e;
        o_rtaddr_e<=i_rtaddr_e;
        o_uid_e<=i_uid_e;
        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
        ra_temp_word[3] <= i_ra_e[`WORD3];
        rb_temp_word[0] <= i_rb_e[`WORD0];
        rb_temp_word[1] <= i_rb_e[`WORD1];
        rb_temp_word[2] <= i_rb_e[`WORD2];
        rb_temp_word[3] <= i_rb_e[`WORD3];
        for (i=0;i<4;i=i+1) begin
                rt_temp_word[i][0]<=ra_temp_word[i][0]~^rb_temp_word[0];
                rt_temp_word[i][1:8]<=ra_temp_word[i][1:8]+rb_temp_word[i][1:8];
                rt_temp_word[i][9:31]<=ra_temp_word[i][9:31]*rb_temp_word[i][9:31];
        end
        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];

end
```

8. Cache and Memory Hierarchy Design

(1). Cache

For the SPU architecture, the cache only for storing instructions, so we design a 32bits*1024 cache, total data store size is 4K Bytes. Every cache line includes valid bit (1 bits), tag (high 20 bits of the memory address) and instruction data (32 bits). Each cache line is 53bits wide, and there is 1024 line, so the actual size of cache is 6.7KB. The cache was associative in the direct mapping way. The instruction seeking is based on the PC, the 1st-20th bits are used as the tag in cache, the 21st-29th bits are used as the index of the cache line, the last 2 bits are byte offset. PC is also the physical address in the memory. The structure shows as the picture blow.

Valid Bit	Tag (20 bits)	Instruction (32 bits)

Cache have a 64-bits path to transfer instruction to instruction fetch unit, because of the dual issue architecture, cache transfers two instruction each time.

```
//check the valid bit and tag
if (ins_cc[index1][0:0] && (ins_cc[index1][1:20])==tag1)
        begin
        // fetch instrutions from Cache!!!
        inst1= ins_cc[index1];
        miss1=1'b0;
        end
else
        //miss condition
        begin
        miss1=1'b1;
        addr_to_memory=pc;
        end

//check the instrucation2
if (ins_cc[index2][0:0] && (ins_cc[index2][1:20])==tag2)
        begin
        // fetch instrutions from Cache!!!
        inst2= ins_cc[index2];
        miss2=1'b0;
        end
else
        //miss condition
        begin
        miss2=1'b1;
        addr_to_memory=addr2;
        end

//send the miss flag to contrl unit
miss=(miss1 || miss2);

//join inst1 and inst2 together
if (miss==0)
        inst={inst1,inst2};
end
```

(2). Memory

We design a 256KB memory, and the first 128K for instruction as i-memory, the second 128K for data ds d-memory. The memory structure is as 128bits*16384. And it has 32bits wide output

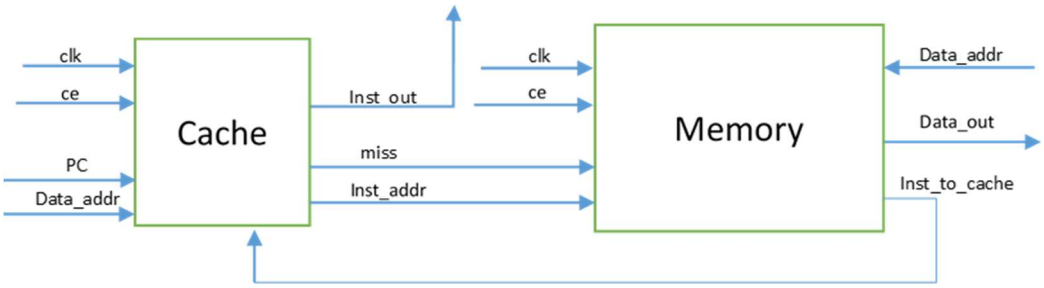
bandwidth for transferring instruction to cache and a 128bits wide output bandwidth for data transferring.

```
always @(*) begin
        if (~enable)    begin
                //data_out<=`ZERO_QWORD128;
                //inst_out<=`ZERO_WORD32;
        end
        else begin
                // store data to memory
                if (write_enable) begin
                                //decode the data adddress to sovle structure hazard
                                if (data_addr[13]==1'b1)
                                        //if the 13th bit of address is 1
                                        //means the address is in the data memory parition
                                        memory_data[data_addr[14:27]]<=data_in;
                                end

                else begin
                        //fetch data to register
                        data_out<=memory_data[data_addr[14:27]];

                        //send instruction data to cache
                        temp<=memory_data[inst_addr[14:27]];
                        offset<=inst_addr[28:29];
                        if (offset==2'b00)
                        inst_out<=temp[0:31];
                        else if (offset==2'b01)
                        inst_out<=temp[32:63];
                        else if (offset==2'b10)
                        inst_out<=temp[64:95];
                        else if (offset==2'b11)
                        inst_out<=temp[96:127];
                inst_addr_to_cache<=inst_addr;
                end
        end
end
```

(3). Cache & Memory

For instruction seeking, based on the PC, first look up in cache, find the correspond cache line, then check the valid bit and the tag, to confirm is it the right and valid instruction. If yes, send the instruction. If not, send the miss signal to control unit stall fetching and seeking the instruction in memory, the PC is the memory address. We take the temporal locality strategy, so after finding the instruction, fetch it to cache and send it to decode unit.

The block diagram shows the signals between cache and memory with some signals to control units.



As the picture shows, in the time of the yellow line, the cache run into miss condition, send miss signal to control unit and pass address to memory to fetch the data. When cache hit, it takes 1 cycle to get the data, when it miss, it would take three cycle to get the data from memory.

9. Parser Design

For easy to write program to test the instruction set, hazard and dependence, we write a parser to translate SPU instruction to binary code using C++. We use a hash table to store the operation code. Read a instruction, split the operand by special character as “,” , “ ” , ” ”(space) , “(” , “)”. After getting the operands like the index of registers or the immediate value, we convert it to a 7,10 or 16 bits binary code based on different operation code. Finally, join all the binary code together output to a file which can be read by Verilog program. The c++ source code is attached as appendixes in the last of this report.

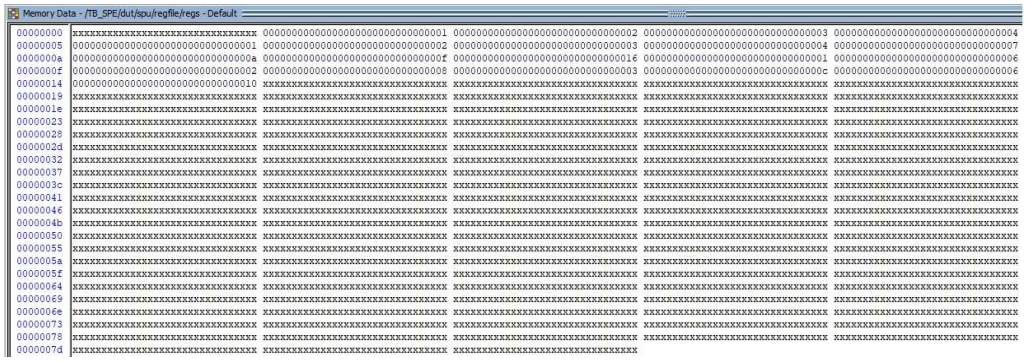
10. Matrix Multiply

We design two 2×2 matrix, $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, for calculating the result of $A \times B$, we store the data of two matrixs in the register, the matrix A stored in the r1, r2, r3, r4, matrix B stored in the r5, r6 r7, r8. The result matrix would be stored in r9, r10, r11, r12.

$$\text{result} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 5 & 22 \end{bmatrix}$$

Blow is the code to realize the calculation.

```
1  mpy r13,r1,r5
2  mpy r14,r2,r7
3  mpy r15,r1,r6
4  mpy r16,r2,r8
5  mpy r17,r3,r5
6  mpy r18,r4,r7
7  mpy r19,r3,r6
8  mpy r20,r4,r8
9  a r9,r13,r14
10 a r10,r15,r16
11 a r11,r17,r18
12 a r12,r19,r20
```



11. Conclusion

In this project, we implemented the SPU-Lite model and most of required functionalities. The test programs we run to do the verification as we discussed above proved that our model works well. However, a better CPU model requires more test cases and complicated pattern to verify, especially on timing issue. Therefore, if we want to build up a more stable CPU model, more testbench need to be run and dig out the bugs exists in the system. Besides that, we learned lots of lessons from this project. First, design documents, like system diagram, is very necessary for such a huge project. Otherwise, it would be very hard to start coding and debugging work. Second, fully understanding about the computer architecture is prerequisite condition to design a CPU module. For example, you couldn't implement data hazard if you don't understand the causes of data hazard and mechanism of data forwarding and stall. Finally, the skill of hardware description language is very important for such a project, which includes Verilog coding knowledge and simulation tools skill. In another side, the procedure of SPU-Lite model establishment helps us better understanding the knowledge of computer architecture. The SPU-Lite model we established is not a perfect system and may still have some bugs need further works to address and fix. The summary of functionality implementation is shown in the table as below.

	SPU-lite Model Code	Parser code	Loading of the memory with instructions & instruction cache imiss processing	Dual-instruction fetch_decode_issue_ execute0 (no hazards)	Structural hazard resolution	Data hazard resolution by forwarding (no stall)	Data hazard resolution by stalling & forwarding	Control hazard resolution for branches	2x2 matrix multiply function
Page # for Code& Testing Results	2,3	10	8	4	8	5	5	6	11
Comments by Design Team	Pass	Pass	Pass	Pass	Pass	Pass	Pass		Pass
Comments by Instructor									

Appendix A. Instruction Set

The selected instructions of SPU-Lite is showed in the table as below.

Name	Symbol	RTL	Unit	Unit id	Pipe	Latency
Load Quadword (d-form)	Lqd rt, ra(symbol)	RT ← LocStor(LSA, 16)	Load Store	5	odd	6
Store Quadword (d-form)	stqd rt,symbol(ra)	LocStor(LSA,16) ← RT	Load Store	5	odd	6
Add Halfword	ah rt,ra,rb	RT0:1 ← RA0:1 + RB0:1 RT2:3 ← RA2:3 + RB2:3 RT4:5 ← RA4:5 + RB4:5 RT6:7 ← RA6:7 + RB6:7 RT8:9 ← RA8:9 + RB8:9 RT10:11 ← RA10:11 + RB10:11 RT12:13 ← RA12:13 + RB12:13 RT14:15 ← RA14:15 + RB14:15	Simple Fixed	1	Even	2
Add Halfword Immediate	ahi rt,ra,value	s ← RepLeftBit(I10,16) RT0:1 ← RA0:1 + s RT2:3 ← RA2:3 + s RT4:5 ← RA4:5 + s RT6:7 ← RA6:7 + s RT8:9 ← RA8:9 + s RT10:11 ← RA10:11 + s RT12:13 ← RA12:13+ s RT14:15 ← RA14:15 + s	Simple Fixed	1	Even	2
Add Word	a rt,ra,rb	RT0:3 ← RA0:3 + RB0:3 RT4:7 ← RA4:7 + RB4:7 RT8:11 ← RA8:11 + RB8:11 RT12:15 ← RA12:15 + RB12:15	Simple Fixed	1	Even	2
Add Word Immediate	ai rt,ra,value	t ← RepLeftBit(I10,32) RT0:3 ← RA0:3 + t RT4:7 ← RA4:7 + t RT8:11 ← RA8:11 + t RT12:15 ← RA12:15 + t	Simple Fixed	1	Even	2
Subtract from Halfword	sfh rt,ra,rb	RT0:1 ← RB0:1 + (¬RA0:1) + 1 RT2:3 ← RB2:3 + (¬RA2:3) + 1 RT4:5 ← RB4:5 + (¬RA4:5) + 1 RT6:7 ← RB6:7 + (¬RA6:7) + 1 RT8:9 ← RB8:9 + (¬RA8:9) + 1 RT10:11 ← RB10:11 + (¬RA10:11) + 1 RT12:13 ← RB12:13 + (¬RA12:13) + 1 RT14:15 ← RB14:15 + (¬RA14:15) + 1	Simple Fixed	1	Even	2

		t ← RepLeftBit(I10,16) RT0:1 ← t + (¬RA0:1) + 1 RT2:3 ← t + (¬RA2:3) + 1 RT4:5 ← t + (¬RA4:5) + 1 RT6:7 ← t + (¬RA6:7) + 1 RT8:9 ← t + (¬RA8:9) + 1 RT10:11 ← t + (¬RA10:11) + 1 RT12:13 ← t + (¬RA12:13) + 1 RT14:15 ← t + (¬RA14:15) + 1	Simple Fixed	1	Even	2
Subtract from Halfword Immediate	sfhi rt,ra,value					
		RT0:3 ← RB0:3 + (¬RA0:3) + 1 RT4:7 ← RB4:7 + (¬RA4:7) + 1 RT8:11 ← RB8:11 + (¬RA8:11) + 1 RT12:15 ← RB12:15 + (¬RA12:15) + 1	Simple Fixed	1	Even	2
Subtract from Word	sf rt,ra,rb					
		t ← RepLeftBit(I10,32) RT0:3 ← t + (¬RA0:3) + 1 RT4:7 ← t + (¬RA4:7) + 1 RT8:11 ← t + (¬RA8:11) + 1 RT12:15 ← t + (¬RA12:15) + 1	Simple Fixed	1	Even	2
Subtract from Word Immediate	sfi rt,ra,value					
		RT0:3 ← RA2:3 * RB2:3 RT4:7 ← RA6:7 * RB6:7 RT8:11 ← RA10:11 * RB10:11 RT12:15 ← RA14:15 * RB14:15	Single Precision	4	Even	7
Multiply	mpy rt,ra,rb					
		t ← RepLeftBit(I10,16) RT0:3 ← RA2:3 * t RT4:7 ← RA6:7 * t RT8:11 ← RA10:11 * t RT12:15 ← RA14:15 * t	Single Precision	4	Even	7
Multiply Immediate	mpyi rt,ra,value					
		RT0:3 ← RA0:3 & RB0:3 RT4:7 ← RA4:7 & RB4:7 RT8:11 ← RA8:11 & RB8:11 RT12:15 ← RA12:15 & RB12:15	Simple Fixed	1	Even	2
And	and rt,ra,rb					
		b ← I10 & 0x00FF bbbb ← b || b || b || b RT0:3 ← RA0:3 & bbbb RT4:7 ← RA4:7 & bbbb RT8:11 ← RA8:11 & bbbb RT12:15 ← RA12:15 & bbbb	Simple Fixed	1	Even	2
And Byte Immediate	andbi rt,ra,value					
		t ← RepLeftBit(I10,16) RT0:1 ← RA0:1 & t RT2:3 ← RA2:3 & t RT4:5 ← RA4:5 & t RT6:7 ← RA6:7 & t RT8:9 ← RA8:9 & t RT10:11 ← RA10:11 & t	Simple Fixed	1	Even	2
And Halfword Immediate	andhi rt,ra,value					

		RT12:13 ← RA12:13 & t RT14:15 ← RA14:15 & t				
		t ← RepLeftBit(I10,32) RT0:3 ← RA0:3 & t RT4:7 ← RA4:7 & t RT8:11 ← RA8:11 & t	Simple			
And Word Immediate	andi rt,ra,value	RT12:15 ← RA12:15 & t	Fixed	1	Even	2
		RT0:3 ← RA0:3 | RB0:3 RT4:7 ← RA4:7 | RB4:7 RT8:11 ← RA8:11 | RB8:11	Simple			
Or	or rt,ra,rb	RT12:15 ← RA12:15 | RB12:15	Fixed	1	Even	2
		b ← I10 & 0x00FF bbbb ← b || b || b || b RT0:3 ← RA0:3 | bbbb RT4:7 ← RA4:7 | bbbb RT8:11 ← RA8:11 | bbbb	Simple			
Or Byte Immediate	orbi rt,ra,value	RT12:15 ← RA12:15 | bbbb	Fixed	1	Even	2
		t ← RepLeftBit(I10,16) RT0:1 ← RA0:1 | t RT2:3 ← RA2:3 | t RT4:5 ← RA4:5 | t RT6:7 ← RA6:7 | t RT8:9 ← RA8:9 | t RT10:11 ← RA10:11 | t RT12:13 ← RA12:13 | t	Simple			
Or Halfword Immediate	orhi rt,ra,value	RT14:15 ← RA14:15 | t	Fixed	1	Even	2
		t ← RepLeftBit(I10,32) RT0:3 ← RA0:3 | t RT4:7 ← RA4:7 | t RT8:11 ← RA8:11 | t	Simple			
Or Word Immediate	ori rt,ra,value	RT12:15 ← RA12:15 | t	Fixed	1	Even	2
		RT0:3 ← RA0:3 ⊕ RB0:3 RT4:7 ← RA4:7 ⊕ RB4:7 RT8:11 ← RA8:11 ⊕ RB8:11	Simple			
Exclusive Or	xor rt,ra,rb	RT12:15 ← RA12:15 ⊕ RB12:15	Fixed	1	Even	2
		b ← I10 & 0x00FF bbbb ← b || b || b || b RT0:3 ← RA0:3 ⊕ bbbb RT4:7 ← RA4:7 ⊕ bbbb RT8:11 ← RA8:11 ⊕ bbbb	Simple			
Exclusive Or Byte Immediate	xorbi rt,ra,value	RT12:15 ← RA12:15 ⊕ bbbb	Fixed	1	Even	2

Exclusive Or Halfword Immediate	xorhi rt,ra,value	t ← RepLeftBit(I10,16) RT0:1 ← RA0:1 ⊕ t RT2:3 ← RA2:3 ⊕ t RT4:5 ← RA4:5 ⊕ t RT6:7 ← RA6:7 ⊕ t RT8:9 ← RA8:9 ⊕ t RT10:11 ← RA10:11 ⊕ t RT12:13 ← RA12:13 ⊕ t RT14:15 ← RA14:15 ⊕ t	Simple Fixed	1	Even	2
Exclusive Or Word Immediate	xori rt,ra,value	t ← RepLeftBit(I10,32) RT0:3 ← RA0:3 ⊕ t RT4:7 ← RA4:7 ⊕ t RT8:11 ← RA8:11 ⊕ t RT12:15 ← RA12:15 ⊕ t	Simple Fixed	1	Even	2
Nand	nand rt,ra,rb	RT0:3 ← ¬(RA0:3 & RB0:3) RT4:7 ← ¬(RA4:7 & RB4:7) RT8:11 ← ¬(RA8:11 & RB8:11) RT12:15 ← ¬(RA12:15 & RB12:15)	Simple Fixed	1	Even	2
Nor	nor rt,ra,rb	RT0:3 ← ¬(RA0:3 | RB0:3) RT4:7 ← ¬(RA4:7 | RB4:7) RT8:11 ← ¬(RA8:11 | RB8:11) RT12:15 ← ¬(RA12:15 | RB12:15)	Simple Fixed	1	Even	2
Shift Left Quadword by Bits	shlqbi rt,ra,rb	s ← RB29:31 for b = 0 to 127 if b + s < 128 then rb ← RAb + s else rb ← 0 end RT ← r	Permute	4	Odd	4
Shift Left Quadword by Bits Immediate	shlqbii rt,ra,value	s ← I7 & 0x07 for b = 0 to 127 if b + s < 128 then rb ← RAb + s else rb ← 0 end RT ← r	Permute	4	Odd	4
Rotate Quadword by Bytes	rotqby rt,ra,rb	s ← RB28:31 for b = 0 to 15 ifb+s<16thenrb ←RAb+s else rb ←RAb+s-16 end RT ← r	Permute	4	Odd	4
Rotate Quadword by Bytes Immediate	rotqbyi rt,ra,value	s ← I714:17 for b = 0 to 15 ifb+s<16thenrb ←RAb+s else rb ←RAb+s-16 end RT ← r	Permute	4	Odd	4

Generate Controls for Byte Insertion (d-form)	cbd rt,symbol(ra)	t ← (RA0:3 + RepLeftBit(I7,32)) & 0x0000000F RT ← 0x101112131415161718191A1B1C1D1E1F RTt ← 0x03	Permute	4	Odd	4
Compare Equal Byte	ceqb rt,ra,rb	for i = 0 to 15 If RAi = RBi then RTi ← 0xFF else RTi ← 0x00 end	Simple Fixed	1	Even	2
Compare Equal Byte Immediate	ceqbi rt,ra,value	for i = 0 to 15 If RAi = I102:9 then RTi ← 0xFF else RTi ← 0x00 end	Simple Fixed	1	Even	2
Compare Equal Halfword	ceqh rt,ra,rb	for i = 0 to 15 by 2 If RAi::2 = RBi::2 then RTi::2 ← 0xFFFF else RTi::2 ← 0x0000 end	Simple Fixed	1	Even	2
Compare Equal Halfword Immediate	ceqhi rt,ra,value	for i = 0 to 15 by 2 If RAi::2 = RepLeftBit(I10,16) then RTi::2 ← 0xFFFF else RTi::2 ← 0x0000 end	Simple Fixed	1	Even	2
Compare Equal Word	ceq rt,ra,rb	for i = 0 to 15 by 4 If RAi::4 = RBi::4 then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Simple Fixed	1	Even	2
Compare Equal Word Immediate	ceqi rt,ra,value	for i = 0 to 15 by 4 If RAi::4 = RepLeftBit(I10,32) then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Simple Fixed	1	Even	2
Compare Greater Than Byte	cgtb rt,ra,	for i = 0 to 15 If RAi > RBi then RTi ← 0xFF else RTi ← 0x00 end	Simple Fixed	1	Even	2
Compare Greater Than Byte Immediate	cgtbi rt,ra,value	for i = 0 to 15 If RAi > I102:9 then RTi ← 0xFF else RTi ← 0x00 end	Simple Fixed	1	Even	2
Compare Greater Than Halfword	cgth rt,ra,rb	for i = 0 to 15 by 2 If RAi::2 > RBi::2 then RTi::2 ← 0xFFFF else RTi::2 ← 0x0000 end	Simple Fixed	1	Even	2

Compare Greater Than Halfword Immediate	cgthi rt,ra,value	for i = 0 to 15 by 2 If RAi::2 > RepLeftBit(I10,16) then RTi::2 ← 0xFFFF else RTi::2 ← 0x0000 end	Simple Fixed	1	Even	2
Compare Greater Than Word	cgt rt,ra,r	for i = 0 to 15 by 4 If RAi::4 > RBi::4 then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Simple Fixed	1	Even	2
Compare Greater Than Word Immediate	cgti rt,ra,value	for i = 0 to 15 by 4 If RAi::4 > RepLeftBit(I10,32) then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Simple Fixed	1	Even	2
Branch Relative	br symbol	PC ← (PC + RepLeftBit(I16 || 0b00,32))	Branch	7	Odd	4
Branch Absolute	bra symbol	PC ← RepLeftBit(I16 || 0b00,32)	Branch	7	Odd	4
Branch Relative and Set Link	brsl rt,symbol	RT0:3 ← (PC + 4) RT4:15 ← 0 PC ← (PC + RepLeftBit(I16 || 0b00,32))	Branch	7	Odd	4
Branch Absolute and Set Link	brasl rt,symbol	RT0:3 ← (PC + 4) RT4:15 ← 0 PC ← RepLeftBit(I16 || 0b00,32)	Branch	7	Odd	4
Branch Indirect	bi ra	PC ← RA0:3 & 0xFFFFFFFC if (E = 0 and D = 0) then interrupt enable status is not modified else if (E = 1 and D = 0) then enable interrupts at target else if (E = 0 and D = 1) then disable interrupts at target else if (E = 1 and D = 1) then reserved	Branch	7	Odd	4
Branch Indirect and Set Link	bisl rt,ra	t ← RA0:3 & LSLR & 0xFFFFFFFC u ← LSLR & (PC + 4) RT0:3 ← u RT4:15 ← 0x00 PC ← t if (E = 0 and D = 0) then interrupt enable status is not modified else if (E = 1 and D = 0) then enable interrupts at target else if (E = 0 and D = 1) then disable interrupts at target	Branch	7	odd	44

		else if (E = 1 and D = 1) then reserved				

Branch If Not Zero Word	brnz rt,symbol	If RT0:3 ≠ 0 then
		PC ← (PC + RepLeftBit(I16 || 0b00)) & 0xFFFFFFFC
		else
		PC ← (PC+4) & 0x0003FFFF
			Branch	7	Odd	4

Branch If Zero Word	brz rt,symbol	If RT0:3 = 0 then
		PC ← (PC + RepLeftBit(I16 || 0b00)) & 0xFFFFFFFC
		else
		PC ← (PC + 4) & 0x0003FFFF
			Branch	7	Odd	4

Branch If Not Zero Halfword	brhnz rt,symbol	If RT2:3 ≠ 0 then
		PC ← (PC + RepLeftBit(I16 || 0b00)) & 0xFFFFFFFC
		else
		PC ← (PC + 4) & 0x0003FFFF
			Branch	7	Odd	4

Branch If Zero Halfword	brhz rt,symbol	If RT2:3 = 0 then
		PC ← (PC + RepLeftBit(I16 || 0b00)) & 0xFFFFFFFC
		else
		PC ← (PC + 4) & 0x0003FFFF
			Branch	7	Odd	4

Floating Add	fa rt,ra,rb	RT0:3 ← RA0:3 + RB0:3 RT4:7 ← RA4:7 + RB4:7 RT8:11 ← RA8:11 + RB8:11 RT12:15 ← RA12:15 + RB12:15	Single Precision	2	Even	6
Floating Subtract	fs rt,ra,rb	RT0:3 ← RB0:3 + (¬RA0:3) + 1 RT4:7 ← RB4:7 + (¬RA4:7) + 1 RT8:11 ← RB8:11 + (¬RA8:11) + 1 RT12:15 ← RB12:15 + (¬RA12:15) + 1	Single Precision	2	even	6
Floating Multiply	fm rt,ra,rb	RT0:3 ← RA2:3 * RB2:3 RT4:7 ← RA6:7 * RB6:7 RT8:11 ← RA10:11 * RB10:11 RT12:15 ← RA14:15 * RB14:15	Single Precision	2	even	6
Floating Multiply and Add	fma rt,ra,rb,rc	t0 ← RA2:3 * RB2:3 t1 ← RA6:7 * RB6:7 t2 ← RA10:11 * RB10:11 t3 ← RA14:15 * RB14:15 RT0:3 ← t0 + RC0:3 RT4:7 ← t1 + RC4:7 RT8:11 ← t2 + RC8:11 RT12:15 ← t3 + RC12:15	Single Precision	2	even	6

Floating Compare Equal	fceq rt,ra,rb	for i = 0 to 15 by 4 If RAi::4 = RBi::4 then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Single Precision	2	even	6
Floating Compare Greater Than	fcgt rt,ra,rb	for i = 0 to 15 by 4 If RAi::8 > RBi::4 then RTi::4 ← 0xFFFFFFFF else RTi::4 ← 0x00000000 end	Single Precision	2	even	6
Count Ones in Bytes	cntb rt,ra	for j = 0 to 15 c=0 b ← RAj For m = 0 to 7 end end If bm = 1 then c ← c + 1 RTj ← c	Byte	3	even	4
Average Bytes	avgb rt,ra,rb	for j = 0 to 15 RTj ← ((0x00 || RAj) + (0x00 || RBj) + 1)7:14 end	Byte	3	even	4
No Operation (Load)	lnop			0	odd	
No Operation (Execute)	nop			0	even	

Appendix B. Source Code: Cell SPUE-Lite (Verilog)

```
//*****************************************************************************
*********/
// File:            defines.v
// Description: macro defines
// History:       Created by Ning Kang, Mar 18,2018
//*****************************************************************************
**********/

`define RST_ENABLE 1'b1
`define RST_DISABLE 1'b0

`define ZERO_WORD32 32'h00000000
`define ZERO_DWORD64 64'h0000000000000000
`define ZERO_QWORD128 128'h00000000000000000000000000000000
`define WR_ENABLE 1'b1
`define WR_DISABLE 1'b0
`define RD_ENABLE 1'b1
`define RD_DISABLE 1'b0
`define CHIP_ENABLE 1'b1
`define CHIP_DISABLE 1'b0
`define OPCODE_BUS_11 0:10

`define STOP 1'b1
`define NOSTOP 1'b0
`define PIPE_EVEN 1'b0
`define PIPE_ODD 1'b1

`define UID_0 3'b000
`define UID_1 3'b001
`define UID_2 3'b010
`define UID_3 3'b011
`define UID_4 3'b100
`define UID_5 3'b101
`define UID_6 3'b110
`define UID_7 3'b111

`define BRANCH            4'h1

`define NOTBRANCH 1'b0
`define IN_DELAYSLOT 1'b1
`define NOT_IN_DELAYSLOT 1'b0
```

```
/*
`define InstValid 1'b0
`define InstInvalid 1'b1
`define InterruptAssert 1'b1
`define InterruptNotAssert 1'b0
`define TrapAssert 1'b1
`define TrapNotAssert 1'b0
`define True_v 1'b1
`define False_v 1'b0
*/

`define lqd                 2'b01
`define stqd                2'b10
//Instructions
`define EXE_LQD         8'b00110100
`define EXE_STQD   8'b00100100
`define EXE_AH          11'b00011001000
`define EXE_AHI         8'b00011101
`define EXE_A       11'b00011000000
`define EXE_AI      8'b00011100
`define EXE_SFH         11'b00001001000
`define EXE_SFHI    8'b00001101
`define EXE_SF      11'b00001000000
`define EXE_SFI         8'b00001100
`define EXE_MPY         11'b01111000100
`define EXE_MPYU  11'b01111001100
`define EXE_MPYI    8'b01110100
`define EXE_AND         11'b00011000001
`define EXE_ANDBI 8'b00010110
`define EXE_ANDHI 8'b00010101
`define EXE_ANDI    8'b00010100
`define EXE_OR          11'b00001000001
`define EXE_ORBI    8'b00000110
`define EXE_ORHI    8'b00000101
`define EXE_ORI     8'b00000100
`define EXE_XOR         11'b01001000001
`define EXE_XORBI  8'b01000110
`define EXE_XORHI  8'b01000101
`define EXE_XORI    8'b01000100
`define EXE_NAND  11'b00011001001
`define EXE_NOR         11'b00001001001
`define EXE_SHLQBI11'b00111011011
`define EXE_SHLQBII      11'b00111111011
```

```
`define EXE_ROTQBY      11'b00111011111
`define EXE_ROTQBYI     11'b00111111111
`define EXE_CBD         11'b00111110100
`define EXE_CEQB   11'b01111010000
`define EXE_CEQBI  8'b01111110
`define EXE_CEQH   11'b01111001000
`define EXE_CEQHI  8'b01111101
`define EXE_CEQ         11'b01111000000
`define EXE_CEQI   8'b01111100
`define EXE_CGTB   11'b01001010000
`define EXE_CGTBI  8'b01001110
`define EXE_CGTH   11'b01001001000
`define EXE_CGTHI  8'b01001101
`define EXE_CGT         11'b01001000000
`define EXE_CGTI   8'b01001100
`define EXE_BR          9'b001100100
`define EXE_BRA         9'b001100000
`define EXE_BRSL   9'b001100110
`define EXE_BRASL  9'b001100010
`define EXE_BI      11'b00110101000
`define EXE_BISL    11'b00110101001
`define EXE_BRNZ   9'b001000010
`define EXE_BRZ         9'b001000000
`define EXE_BRHNZ 9'b001000110
`define EXE_BRHZ   9'b001000100
`define EXE_FA      11'b01011000100
`define EXE_FS      11'b01011000101
`define EXE_FM          11'b01011000110
`define EXE_FCEQ   11'b01111000010
`define EXE_FCGT   11'b01011000010
`define EXE_CNTB   11'b01010110100
`define EXE_AVGB   11'b00011010011
`define EXE_STOP   11'b00000000000
`define EXE_LNOP   11'b00000000001
`define EXE_NOP         11'b01000000001

//Local storage
`define LS_ADDR_BUS14 0:13
`define LS_ADDR_BUS18 0:17
`define LS_DATA_BUS128 0:127
`define LS_NUM_256K 16384 //256K=(16384*128)/8bit

//Cache
//`define CACHE_BUS4 0:3
```

```
`define INST_ADDR_BUS32 0:31
`define INST2_BUS64 0:63
`define INST_BUS32 0:31
//`define INST_ILB_SIZE32 32
`define CACHE_NUM_4K 1024
`define MISS1 0:0
`define CACHE_INDEX10 0:9
`define CACHE_WIDTH53 0:52
`define CACHE_TAG20 0:19

//MEMEORY
`define MEMORY_WIDTH32 0:31
`define MEMORY_WIDTH128 0:127
`define MEMORY_256K 16384

//Register File
/*
//`define DoubleRegWidth 64
//`define DoubleRegBus 63:0
//`define RegNumLog2 7
*/

`define REG_ADDR_BUS7 0:6
`define REG_BUS128 0:127
`define REG_WIDTH128 128
`define REG_NUM128 128
`define NOP_REG_ADDR 7'b0000000

//OPERATION BYTES
`define HALFWORD0 0:15
`define HALFWORD1 16:31
`define HALFWORD2 32:47
`define HALFWORD3 48:63
`define HALFWORD4 64:79
`define HALFWORD5 80:95
`define HALFWORD6 96:111
`define HALFWORD7 112:127

`define WORD0 0:31
`define WORD1 32:63
`define WORD2 64:95
`define WORD3 96:127

`define HWORD_R16B0 16:31
```

```
`define HWORD_R16B1 48:63
`define HWORD_R16B2 80:95
`define HWORD_R16B3 112:127

`define BYTE0 0:7
`define BYTE1 8:15
`define BYTE2 16:23
`define BYTE3 24:31
`define BYTE4 32:39
`define BYTE5 40:47
`define BYTE6 48:55
`define BYTE7 56:63
`define BYTE8 64:71
`define BYTE9 72:79
`define BYTE10 80:87
`define BYTE11 88:95
`define BYTE12 95:111
`define BYTE13 80:95
`define BYTE14 96:111
`define BYTE15 112:127


//*************************************************************************
*********/
// Module:      PC_REG - PC register
// File:        pc_reg.v
// Description: PC register stores the instruction address and +8 per clock cycle
//              PC is 32 bits fixed length
// History:     Created by Weilun Cheng, Mar 15,2018
//              Modify by Ning Kang, Mar 25, 2018 - modify the PC length
//*************************************************************************
**********/

`include "defines.v"

module PC_REG(

    input wire clk,         //SPC clock
    input wire rst,         //SPU reset
    input wire [0:12] stall, //from CTRL module

    input wire id_pc,
    input wire branch_flag_i,
    //input wire branch_clr,
```

```
    input wire[0:31] branch_target_addr_i,

    output reg[0:31] pc,       //PC register: instruction address, to if_pc
    output reg ce          //Cache eanble, posedge active

);

    reg[0:31] pc_echo;
    //reg branch_flag;
    reg i=0;


    always @(*) begin
        //branch_flag = (branch_flag_i&&branch_clr);
        i = 0;
    end

    always @ (posedge clk) begin
        if (rst == `RST_ENABLE) begin
            ce <= `CHIP_DISABLE;
        end else begin
            ce <= `CHIP_ENABLE;
        end
    end

    always @ (posedge clk) begin
        if (ce == `CHIP_DISABLE) begin
            pc = `ZERO_WORD32;
        end else if (stall[0] == `NOSTOP) begin
            if((branch_flag_i == `BRANCH) &&(i==0)) begin
                pc = branch_target_addr_i;
                i = 1;
            end else if (stall==13'h7) begin
                pc = pc;
            end else if (stall==13'h8) begin
                pc = pc_echo;
            end else begin
                pc = pc + 4'h8;
                pc_echo = pc-4'h8;
            end
        end
    end
```

```
endmodule


//*********************************************************************************
*********/
// Module:      IF_ID
// File:        if_id.v
// Description: Buffer between instruction fetch an instruction decode
// History:     Created by Weilun Cheng, Mar 15,2018
//          Modify by Ning Kang, Mar 25, 2018
//              - change if_pc to 32 bits fixed length
//              - modify inst from 32 bits to 64 bits for 2 ins fetch at a time
//          Modify by Ning Kang, Apr 24, 2018
//              - add Stall mechanism
//*********************************************************************************
**********/

`include "defines.v"

module IF_ID(

    input wire clk, //SPU clock
    input wire rst, //SPU reset

    input wire[`INST_ADDR_BUS32] if_pc, //PC in instruction fetch stage, from pc to id_pc
    input wire[`INST2_BUS64] if_inst, //2 instructions fetch in IF stage, to id_inst
    //input wire[`INST2_BUS64] inst_echo,
    input wire pipe_l, pipe_h,

    input wire[0:12] stall, // ctrl bits from CTRL module

    output reg[`INST_ADDR_BUS32] id_pc, //PC in instruction decode stage
    output reg[`INST2_BUS64] id_inst //2 instructions in ID stage, further split to 2 ins in ID
);

    reg[0:63] inst_echo;

    always @ (posedge clk) begin
        if ( rst == `RST_ENABLE ) begin
            id_pc <= `ZERO_WORD32;
            id_inst <= `ZERO_DWORD64;
        end else if (stall[1] == `STOP && stall[2] == `NOSTOP) begin //
            id_pc <= `ZERO_WORD32;
            id_inst <= `ZERO_DWORD64;
```

```
        //end else if ((stall[2] == `STOP) && (stall[3] == `NOSTOP))begin
        end else if ((stall == 13'h7) && (pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_ODD) ) begin
            id_pc <= if_pc;
            id_inst <= {inst_echo[32:63], 32'h40200000}; // resend previous 2 instructions to ID
        end else if ((stall == 13'h7) && (pipe_l==`PIPE_EVEN)&&(pipe_h==`PIPE_EVEN) ) begin
            id_pc <= if_pc;
            id_inst <= {inst_echo[32:63], 32'h00200000}; // resend previous 2 instructions to ID
        end else if (stall == 13'h7) begin
            id_pc <= if_pc;
            id_inst <= if_inst;
        end else if (stall[1] == `NOSTOP) begin
            id_pc <= if_pc;
            id_inst <= if_inst;
            inst_echo <= if_inst; // reserve current 2 instructions
        end
    end

endmodule




//*********************************************************************************
*********/
// Module:      ID
// File:        id.v
// Description: instruction decode
//          Issue Control allows to issue up to 2 instrunctions per cycle to nine
//          execution units organized into 2 execution pipelines
//          1. Even: Simple fixed / Shift / Single Precision / Floating Integer / Byte
//          2. Odd: Permute / Local Store / Channel / Branch
// History:     Created by Ning Kang, Mar 26, 2018
//              - implemented dual issue feature
//              - implemented Cell SPU ISA features
//          Modify by Ning Kang, Apr 24, 2018
//              - Implemented all operation
//              - add output of stall request
//          MOdify by Ning Kang, Apr25, 2018
//              - Implemented data forwarding
//*********************************************************************************
**********/

`include "defines.v"
```

```
module ID(
    input wire rst,             // reset
    input wire clk,             // SPU clock

    input wire[0:31] pc_i,          //pc address in decode stage
    input wire[0:63] inst_i,    //instruction (actually 2) in decode stage

    //Code start...added by Ning, Kang for data forwarding
    input wire ex_wreg_i_e,
    input wire [`REG_BUS128] ex_wdata_i_e,
    input wire [`REG_ADDR_BUS7] ex_waddr_i_e,

    input wire ff1_wreg_i_e,
    input wire [`REG_BUS128] ff1_rt_e,
    input wire [`REG_ADDR_BUS7] ff1_waddr_i_e,

    input wire ff2_wreg_i_e,
    input wire [`REG_BUS128] ff2_rt_e,
    input wire [`REG_ADDR_BUS7] ff2_waddr_i_e,

    input wire ff3_wreg_i_e,
    input wire [`REG_BUS128] ff3_rt_e,
    input wire [`REG_ADDR_BUS7] ff3_waddr_i_e,

    input wire ff4_wreg_i_e,
    input wire [`REG_BUS128] ff4_rt_e,
    input wire [`REG_ADDR_BUS7] ff4_waddr_i_e,

    input wire ff5_wreg_i_e,
    input wire [`REG_BUS128] ff5_rt_e,
    input wire [`REG_ADDR_BUS7] ff5_waddr_i_e,

    input wire ff6_wreg_i_e,
    input wire [`REG_BUS128] ff6_rt_e,
    input wire [`REG_ADDR_BUS7] ff6_waddr_i_e,

    input wire ff7_wreg_i_e,
    input wire [`REG_BUS128] ff7_rt_e,
    input wire [`REG_ADDR_BUS7] ff7_waddr_i_e,

    input wire ex_wreg_i_o,
    input wire [`REG_BUS128] ex_wdata_i_o,
    input wire [`REG_ADDR_BUS7] ex_waddr_i_o,
```

```
input wire ff1_wreg_i_o,
input wire [`REG_BUS128] ff1_rt_o,
input wire [`REG_ADDR_BUS7] ff1_waddr_i_o,

input wire ff2_wreg_i_o,
input wire [`REG_BUS128] ff2_rt_o,
input wire [`REG_ADDR_BUS7] ff2_waddr_i_o,

input wire ff3_wreg_i_o,
input wire [`REG_BUS128] ff3_rt_o,
input wire [`REG_ADDR_BUS7] ff3_waddr_i_o,

input wire ff4_wreg_i_o,
input wire [`REG_BUS128] ff4_rt_o,
input wire [`REG_ADDR_BUS7] ff4_waddr_i_o,

input wire ff5_wreg_i_o,
input wire [`REG_BUS128] ff5_rt_o,
input wire [`REG_ADDR_BUS7] ff5_waddr_i_o,

input wire ff6_wreg_i_o,
input wire [`REG_BUS128] ff6_rt_o,
input wire [`REG_ADDR_BUS7] ff6_waddr_i_o,

input wire ff7_wreg_i_o,
input wire [`REG_BUS128] ff7_rt_o,
input wire [`REG_ADDR_BUS7] ff7_waddr_i_o,

input wire mem_wreg_i_e,
input wire [`REG_BUS128] mem_wdata_i_e,
input wire [`REG_ADDR_BUS7] mem_waddr_i_e,
input wire mem_wreg_i_o,
input wire [`REG_BUS128] mem_wdata_i_o,
input wire [`REG_ADDR_BUS7] mem_waddr_i_o,
//Code end

//from register file: ra / rb
input wire[0:127] ra_i_e,
input wire[0:127] rb_i_e,

input wire[0:127] ra_i_o,
input wire[0:127] rb_i_o,

//If last instruction is branch, then the present instruction decode will set it to 1, else false
```

```
input wire is_in_delayslot_i,


output reg[0:31] link_addr_o,
output reg is_in_delayslot_o,
output reg next_inst_in_delayslot_o,
output reg[0:31] branch_target_addr_o,
output reg branch_flag_o,

//pc
output wire[0:31] id_pc,
output wire[0:31] inst_l,
output wire[0:31] inst_h,

//info send to exe: ra, rb, rt, wr_en, opcode
output reg[0:10] op_code_e,         // even op code
output reg[0:127] ra_do_e,    // even ra
output reg[0:127] rb_do_e,    // even rb
output reg[0:6] rt_addr_e,     // even rt address
output reg wreg_e,                 // even wr enable
output reg[0:2] uid_e,

output reg[0:10] op_code_o,  // odd op code
output reg[0:127] ra_do_o,    // odd ra
output reg[0:127] rb_do_o,    // odd rb
output reg[0:6] rt_addr_o,     // odd rt
output reg wreg_o,         // odd wr enable
output reg[0:2] uid_o,
output reg[0:9] im_o,

//output pipe_l, pipe_h: 1-odd, 0-even
output reg pipe_l,
output reg pipe_h,

//info send to register file
output reg[0:6] ra_addr_e,
output reg[0:6] rb_addr_e,
output reg ra_rd_e,
output reg rb_rd_e,

output reg[0:6] ra_addr_o,
output reg[0:6] rb_addr_o,
output reg ra_rd_o, rb_rd_o,
```

```
    //output reg[0:31] pc_e, pc_o,
    output reg stallreq,
    output reg stallreq_dh
);

    //split inst_i into even ins and odd ins
    //wire [0:31] inst_l, inst_h;
    assign inst_l = inst_i[0:31];
    assign inst_h = inst_i[32:63];

    assign id_pc = pc_i;

    //initia stallreq as no stop
    //assign stallreq = `NOSTOP;

    //obtian ins opcode
    wire [0:10] op_l = inst_l[0:10];
    wire [0:10] op_h = inst_h[0:10];

    reg[`REG_BUS128] imm_e, imm_o;
    //reg imm_e, imm_o;

    wire[0:63] pc_plus_8;
    wire[0:63] pc_plus_4;

    //assign pc_plus_8 = pc_i + 8;
    //assign pc_plus_4 = pc_i + 4;
    //reg stall_req_for_pipe_split;
    //assign stallreq = stall_req_for_pipe_split;
    //reg [0:9] ttt; // only for debug
    reg [0:7] b;
    reg [0:7] byte_t;
    reg [0:15] hword_t;
    reg [0:31] word_t;

    always @ (*) begin
        casez (op_l)
            11'b00110100???: begin        //EXE_LQD
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_5;      //latency=6
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_LQD;
                ra_rd_o <= `RD_ENABLE;
                rb_rd_o <= `RD_DISABLE;
```

```
                ra_addr_o <= inst_l[18:24];
                imm_o <= {{22{inst_l[8]}},inst_l[8:17]};
                rt_addr_o <= inst_l[25:31];
            end
            11'b00100100???: begin        //EXE_STQD
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_6;      //latency=6
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_STQD;
                ra_rd_o <= `RD_ENABLE;
                rb_rd_o <= `RD_ENABLE;
                ra_addr_o <= inst_l[18:24];
                im_o <= inst_l[8:17];
                rb_addr_o <= inst_l[25:31];
            end
            11'b00011001000: begin              //EXE_AH
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b00011101???: begin        //EXE_AHI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                hword_t = {{6{inst_l[8]}},inst_l[8:17]}; //HalfWord:16b
                imm_e                                                          <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00011000000: begin              //EXE_A
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_A;
```

```
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b00011100???: begin        //EXE_AI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                word_t = {{22{inst_l[8]}},inst_l[8:17]};//Word:32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00001001000: begin              //EXE_SFH
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SFH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b00001101???: begin        //EXE_SFHI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SFHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                hword_t = {{6{inst_l[8]}},inst_l[8:17]};//HalfWOrd: 16b
                imm_e                                                                   <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00001000000: begin              //EXE_SF
                pipe_l = `PIPE_EVEN;
```

```
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SF;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b00001100???: begin        //EXE_SFI
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_1;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_SFI;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_DISABLE;
            ra_addr_e <= inst_l[18:24];
            word_t = {{22{inst_l[8]}},inst_l[8:17]};
            imm_e <= {word_t,word_t,word_t,word_t};
            rt_addr_e <= inst_l[25:31];
        end
        11'b01111000100: begin              //EXE_MPY
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_4;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_MPY;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b01111001100: begin              //EXE_MPYU
            pipe_l = `PIPE_EVEN;
            uid_e <= `UID_4;
            wreg_e = `WR_ENABLE;
            op_code_e <= `EXE_MPYU;
            ra_rd_e <= `RD_ENABLE;
            rb_rd_e <= `RD_ENABLE;
            ra_addr_e <= inst_l[18:24];
            rb_addr_e <= inst_l[11:17];
            rt_addr_e <= inst_l[25:31];
        end
        11'b01110100???: begin        //EXE_MPYI
```

```
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_4;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_MPYI;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_l[18:24];
        hword_t = {{6{inst_l[8]}},inst_l[8:17]}; //Hword: 16b
        word_t = {16'h0000,hword_t};
        imm_e <= {word_t,word_t,word_t,word_t};
        rt_addr_e <= inst_l[25:31];
    end
    11'b00011000001: begin              //EXE_AND
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_AND;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b00010110???: begin      //EXE_ANDBI
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_ANDBI;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_l[18:24];
        word_t = (inst_l[8:17]&16'h00ff);
        imm_e <= {word_t,word_t,word_t,word_t}; // 8*4=32b
        rt_addr_e <= inst_l[25:31];
    end
    11'b00010101???: begin      //EXE_ANDHI
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_ANDHI;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_l[18:24];
        hword_t = {{6{inst_l[8]}},inst_l[8:17]}; //Hwaord: 16b
```

```
                imm_e                                                    <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00010100???: begin      //EXE_ANDI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ANDI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                word_t = {{22{inst_l[8]}},inst_l[8:17]}; //Word: 32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00001000001: begin          //EXE_OR
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_OR;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b00000110???: begin      //EXE_ORBI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORBI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                word_t = (inst_l[8:17]&16'h00ff); // 8*4=32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00000101???: begin      //EXE_ORHI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORHI;
```

```
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                hword_t = {{6{inst_l[8]}},inst_l[8:17]}; //Hword: 16b
                imm_e                                                    <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b00000100???: begin      //EXE_ORI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                word_t = {{22{inst_l[8]}},inst_l[8:17]}; //word: 32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001000001: begin          //EXE_XOR
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_XOR;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01000110???: begin      //EXE_XORBI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_XORBI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                //imm_e <= {{22{inst_l[8]}},inst_l[8:17]};
                word_t = (inst_l[8:17]&32'h00ff);
                //b = (inst_l[8:17]&16'h00ff);
                imm_e <= {word_t,word_t,word_t,word_t}; // 8*4=32b
                rt_addr_e <= inst_l[25:31];
```

```
                end
                11'b01000101???: begin        //EXE_XORHI
                    pipe_l = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_XORHI;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_DISABLE;
                    ra_addr_e <= inst_l[18:24];
                    hword_t = {{6{inst_l[8]}},inst_l[8:17]}; //Hword:16b
                    imm_e                                                                    =
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                    rt_addr_e <= inst_l[25:31];
                end
                11'b01000100???: begin        //EXE_XORI
                    pipe_l = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_XORI;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_DISABLE;
                    ra_addr_e <= inst_l[18:24];
                    word_t ={{22{inst_l[8]}},inst_l[8:17]}; //Word: 32b
                    imm_e <= {word_t,word_t,word_t,word_t};
                    rt_addr_e <= inst_l[25:31];
                end
                11'b00011001001: begin              //EXE_NAND
                    pipe_l = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_NAND;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_ENABLE;
                    ra_addr_e <= inst_l[18:24];
                    rb_addr_e <= inst_l[11:17];
                    rt_addr_e <= inst_l[25:31];
                end
                11'b00001001001: begin              //EXE_NOR
                    pipe_l = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_NOR;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_ENABLE;
```

```
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b00111011011: begin              //EXE_SHLQBI
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_SHLQBI;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_ENABLE;
        ra_addr_o <= inst_l[18:24];
        rb_addr_o <= inst_l[11:17];
        rt_addr_o <= inst_l[25:31];
    end
    11'b00111111011: begin              //EXE_SHLQBII
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_SHLQBII;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_l[18:24];
        //imm_o <= {{25{inst_l[11]}},inst_l[11:17]};
        imm_o <= inst_l[11:17];
        rt_addr_o <= inst_l[25:31];
    end
    11'b00111011111: begin              //EXE_ROTQBY
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_ROTQBY;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_ENABLE;
        ra_addr_o <= inst_l[18:24];
        rb_addr_o <= inst_l[11:17];
        rt_addr_o <= inst_l[25:31];
    end
    11'b00111111111: begin              //EXE_ROTQBYI
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_ROTQBYI;
        ra_rd_o <= `RD_ENABLE;
```

```
                rb_rd_o <= `RD_DISABLE;
                ra_addr_o <= inst_l[18:24];
                imm_o <= inst_l[11:17];
                rt_addr_o <= inst_l[25:31];
            end
            11'b00111110100: begin          //EXE_CBD
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_4;
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_CBD;
                ra_rd_o <= `RD_ENABLE;
                rb_rd_o <= `RD_DISABLE;
                ra_addr_o <= inst_l[18:24];
                imm_o <= inst_l[11:17];
                rt_addr_o <= inst_l[25:31];
            end
            11'b01111010000: begin          //EXE_CEQB
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQB;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01111110???: begin      //EXE_CEQBI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQBI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                //imm_e <= inst_l[8:17];
                byte_t = (inst_l[10:17] & 8'hff); // I10 only need bit 2:9
                imm_e                                                                   <=
{byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_
t,byte_t,byte_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01111001000: begin          //EXE_CEQH
                pipe_l = `PIPE_EVEN;
```

```
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01111101???: begin      //EXE_CEQHI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                hword_t = {{6{inst_l[8]}},inst_l[8:17]}; // Extended to 16bits by replicating leftmost bit
                imm_e <= {hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01111000000: begin          //EXE_CEQ
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQ;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01111100???: begin      //EXE_CEQI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                word_t = {{22{inst_l[8]}},inst_l[8:17]}; //Extended to 32bits by replicating leftmost bit
```

```
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001010000: begin          //EXE_CGTB
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTB;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001110???: begin      //EXE_CGTBI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTBI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                //imm_e <= inst_l[8:17];
                byte_t = inst_l[10:17]; // I10 only need bit 2:9
                imm_e                                                                   <=
{byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_
t,byte_t,byte_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001001000: begin          //EXE_CGTH
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001101???: begin      //EXE_CGTHI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
```

```
                op_code_e <= `EXE_CGTHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                //imm_e <= inst_l[8:17];
                hword_t = {{6{inst_l[8]}},inst_l[8:17]}; // Extended to 16bits by replicating leftmost bit
                imm_e <= {hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001000000: begin          //EXE_CGT
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGT;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_l[18:24];
                rb_addr_e <= inst_l[11:17];
                rt_addr_e <= inst_l[25:31];
            end
            11'b01001100???: begin      //EXE_CGTI
                pipe_l = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTI;
                ra_rd_e <= `RD_ENABLE; //read imm from ra
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_l[18:24];
                //imm_e <= inst_l[8:17];
                word_t = {{22{inst_l[8]}},inst_l[8:17]}; //Extended to 32bits by replicating leftmost bit
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_l[25:31];
            end
            11'b001100100??: begin      //EXE_BR
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_DISABLE;
                op_code_o <= `EXE_BR;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
```

```
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00}; // Extended on the right with 2 0bits
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100000??: begin        //EXE_BRA
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_DISABLE;
                op_code_o <= `EXE_BRA;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100110??: begin        //EXE_BRSL
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_BRSL;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                rt_addr_o <= inst_l[25:31];
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100010??: begin        //EXE_BRASL
                pipe_l = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_BRASL;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                rt_addr_o <= inst_l[25:31];
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
                link_addr_o <= `ZERO_WORD32;
            end
            11'b00110101000: begin              //EXE_BI
```

```
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_BI;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_l[18:24];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        //branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b00110101001: begin              //EXE_BISL
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_BISL;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_l[18:24];
        rt_addr_o <= inst_l[25:31];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        //branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000010??: begin      //EXE_BRNZ
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE; //RT only used to READ for BRNZ
        op_code_o <= `EXE_BRNZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        rt_addr_o <= inst_l[25:31];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000000??: begin      //EXE_BRZ
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE; //RT only used to READ for BRZ
        op_code_o <= `EXE_BRZ;
```

```
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        rt_addr_o <= inst_l[25:31];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000110??: begin        //EXE_BRHNZ
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_BRHNZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000100??: begin        //EXE_BRHZ
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_BRHZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_l[9]}},inst_l[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b01011000100: begin              //EXE_FA
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_2;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_FA;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b01011000101: begin              //EXE_FS
```

```
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_2;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_FA;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b01011000110: begin          //EXE_FM
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_2;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_FM;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b01111000010: begin          //EXE_FCEQ
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_2;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_FCEQ;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b01011000010: begin          //EXE_FCGT
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_2;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_FCGT;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b01010110100: begin          //EXE_CNTB
```

```
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_3;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_CNTB;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_l[18:24];
        //rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b00011010011: begin          //EXE_AVGB
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_3;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_AVGB;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_l[18:24];
        rb_addr_e <= inst_l[11:17];
        rt_addr_e <= inst_l[25:31];
    end
    11'b00000000000: begin          //EXE_STOP
        //???
    end
    11'b00000000001: begin          //EXE_LNOP
        pipe_l = `PIPE_ODD;
        uid_o <= `UID_0;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_LNOP;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        rt_addr_o <= 0;
    end
    11'b01000000001: begin          //EXE_NOP
        pipe_l = `PIPE_EVEN;
        uid_e <= `UID_0;
        wreg_e = `WR_DISABLE;
        op_code_e <= `EXE_NOP;
        ra_rd_e <= `RD_DISABLE;
        rb_rd_e <= `RD_DISABLE;
        rt_addr_e <= 0;
    end
    default: begin
    end
```

```
        endcase
    end

    always @ (*) begin
        casez (op_h)
            11'b00110100???: begin        //EXE_LQD
                pipe_h = `PIPE_ODD;
                uid_o <= `UID_5;      //latency=6
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_LQD;
                ra_rd_o <= `RD_ENABLE;
                rb_rd_o <= `RD_DISABLE;
                ra_addr_o <= inst_h[18:24];
                imm_o <= {{22{inst_h[8]}},inst_l[8:17]};
                rt_addr_o <= inst_h[25:31];
            end
            11'b00100100???: begin        //EXE_STQD
                pipe_h = `PIPE_ODD;
                uid_o <= `UID_5;      //latency=6
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_STQD;
                ra_rd_o <= `RD_ENABLE;
                rb_rd_o <= `RD_DISABLE;
                ra_addr_o <= inst_h[18:24];
                imm_o <= {{22{inst_h[8]}},inst_l[8:17]};
                rt_addr_o <= inst_h[25:31];
            end
            11'b00011001000: begin              //EXE_AH
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b00011101???: begin        //EXE_AHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AHI;
                ra_rd_e <= `RD_ENABLE;
```

```
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]}; //HalfWord:16b
                imm_e <= {hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00011000000: begin          //EXE_A
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_A;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b00011100???: begin      //EXE_AI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_AI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = {{22{inst_h[8]}},inst_h[8:17]};//Word:32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00001001000: begin          //EXE_SFH
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SFH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b00001101???: begin      //EXE_SFHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
```

```
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SFHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]};//HalfWOrd: 16b
                imm_e                                                   <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00001000000: begin          //EXE_SF
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SF;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b00001100???: begin      //EXE_SFI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_SFI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = {{22{inst_h[8]}},inst_h[8:17]};
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01111000100: begin          //EXE_MPY
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_4;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_MPY;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
```

```
11'b01111001100: begin              //EXE_MPYU
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_4;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_MPYU;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_h[18:24];
        rb_addr_e <= inst_h[11:17];
        rt_addr_e <= inst_h[25:31];
end
11'b01110100???: begin        //EXE_MPYI
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_4;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_MPYI;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_h[18:24];
        hword_t = {{6{inst_h[8]}},inst_h[8:17]}; //Hword: 16b
        word_t = {16'h0000,hword_t};
        imm_e <= {word_t,word_t,word_t,word_t};
        rt_addr_e <= inst_h[25:31];
end
11'b00011000001: begin              //EXE_AND
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_AND;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_h[18:24];
        rb_addr_e <= inst_h[11:17];
        rt_addr_e <= inst_h[25:31];
end
11'b00010110???: begin        //EXE_ANDBI
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_ANDBI;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_DISABLE;
        ra_addr_e <= inst_h[18:24];
        word_t = (inst_h[8:17]&16'h00ff);
```

```
                imm_e <= {word_t,word_t,word_t,word_t}; // 8*4=32b
                rt_addr_e <= inst_h[25:31];
            end
            11'b00010101???: begin        //EXE_ANDHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ANDHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]}; //Hwaord: 16b
                imm_e                                                    <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00010100???: begin        //EXE_ANDI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ANDI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = {{22{inst_h[8]}},inst_h[8:17]}; //Word: 32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00001000001: begin              //EXE_OR
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_OR;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b00000110???: begin        //EXE_ORBI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORBI;
```

```
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = (inst_h[8:17]&16'h00ff); // 8*4=32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00000101???: begin      //EXE_ORHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]}; //Hword: 16b
                imm_e                                                   <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b00000100???: begin      //EXE_ORI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_ORI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = {{22{inst_h[8]}},inst_h[8:17]}; //word: 32b
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001000001: begin          //EXE_XOR
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_XOR;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b01000110???: begin      //EXE_XORBI
```

```
                    pipe_h = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_XORBI;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_DISABLE;
                    ra_addr_e <= inst_h[18:24];
                    //imm_e <= {{22{inst_h[8]}},inst_h[8:17]};
                    word_t = (inst_h[8:17]&32'h00ff);
                    //b = (inst_h[8:17]&16'h00ff);
                    imm_e <= {word_t,word_t,word_t,word_t}; // 8*4=32b
                    rt_addr_e <= inst_h[25:31];
                end
                11'b01000101???: begin        //EXE_XORHI
                    pipe_h = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_XORHI;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_DISABLE;
                    ra_addr_e <= inst_h[18:24];
                    hword_t = {{6{inst_h[8]}},inst_h[8:17]}; //Hword:16b
                    imm_e                                                                    =
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                    rt_addr_e <= inst_h[25:31];
                end
                11'b01000100???: begin        //EXE_XORI
                    pipe_h = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_XORI;
                    ra_rd_e <= `RD_ENABLE;
                    rb_rd_e <= `RD_DISABLE;
                    ra_addr_e <= inst_h[18:24];
                    word_t ={{22{inst_h[8]}},inst_h[8:17]}; //Word: 32b
                    imm_e <= {word_t,word_t,word_t,word_t};
                    rt_addr_e <= inst_h[25:31];
                end
                11'b00011001001: begin              //EXE_NAND
                    pipe_h = `PIPE_EVEN;
                    uid_e <= `UID_1;
                    wreg_e = `WR_ENABLE;
                    op_code_e <= `EXE_NAND;
                    ra_rd_e <= `RD_ENABLE;
```

```
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_h[18:24];
        rb_addr_e <= inst_h[11:17];
        rt_addr_e <= inst_h[25:31];
    end
    11'b00001001001: begin              //EXE_NOR
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_NOR;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_h[18:24];
        rb_addr_e <= inst_h[11:17];
        rt_addr_e <= inst_h[25:31];
    end
    11'b00111011011: begin              //EXE_SHLQBI
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_SHLQBI;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_ENABLE;
        ra_addr_o <= inst_h[18:24];
        rb_addr_o <= inst_h[11:17];
        rt_addr_o <= inst_h[25:31];
    end
    11'b00111111011: begin              //EXE_SHLQBII
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_SHLQBII;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_h[18:24];
        //imm_o <= {{25{inst_h[11]}},inst_h[11:17]};
        imm_o <= inst_h[11:17];
        rt_addr_o <= inst_h[25:31];
    end
    11'b00111011111: begin              //EXE_ROTQBY
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_ROTQBY;
```

```
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_ENABLE;
        ra_addr_o <= inst_h[18:24];
        rb_addr_o <= inst_h[11:17];
        rt_addr_o <= inst_h[25:31];
    end
    11'b00111111111: begin              //EXE_ROTQBYI
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_ROTQBYI;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_h[18:24];
        imm_o <= inst_h[11:17];
        rt_addr_o <= inst_h[25:31];
    end
    11'b00111110100: begin              //EXE_CBD
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_4;
        wreg_o = `WR_ENABLE;
        op_code_o <= `EXE_CBD;
        ra_rd_o <= `RD_ENABLE;
        rb_rd_o <= `RD_DISABLE;
        ra_addr_o <= inst_h[18:24];
        imm_o <= inst_h[11:17];
        rt_addr_o <= inst_h[25:31];
    end
    11'b01111010000: begin              //EXE_CEQB
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_CEQB;
        ra_rd_e <= `RD_ENABLE;
        rb_rd_e <= `RD_ENABLE;
        ra_addr_e <= inst_h[18:24];
        rb_addr_e <= inst_h[11:17];
        rt_addr_e <= inst_h[25:31];
    end
    11'b01111110???: begin      //EXE_CEQBI
        pipe_h = `PIPE_EVEN;
        uid_e <= `UID_1;
        wreg_e = `WR_ENABLE;
        op_code_e <= `EXE_CEQBI;
```

```
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                //imm_e <= inst_h[8:17];
                byte_t = (inst_h[10:17] & 8'hff); // I10 only need bit 2:9
                imm_e <= {byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01111001000: begin          //EXE_CEQH
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b01111101???: begin      //EXE_CEQHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]}; // Extended to 16bits by replicating leftmost bit
                imm_e <= {hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01111000000: begin          //EXE_CEQ
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQ;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
```

```
                rt_addr_e <= inst_h[25:31];
            end
            11'b01111100???: begin      //EXE_CEQI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CEQI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                word_t = {{22{inst_h[8]}},inst_h[8:17]}; //Extended to 32bits by replicating leftmost bit
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001010000: begin          //EXE_CGTB
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTB;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001110???: begin      //EXE_CGTBI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTBI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                //imm_e <= inst_h[8:17];
                byte_t = inst_h[10:17]; // I10 only need bit 2:9
                imm_e <= {byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t,byte_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001001000: begin          //EXE_CGTH
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
```

```
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTH;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001101???: begin      //EXE_CGTHI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTHI;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                //imm_e <= inst_h[8:17];
                hword_t = {{6{inst_h[8]}},inst_h[8:17]}; // Extended to 16bits by replicating
leftmost bit
                imm_e                                                                   <=
{hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t,hword_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001000000: begin          //EXE_CGT
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGT;
                ra_rd_e <= `RD_ENABLE;
                rb_rd_e <= `RD_ENABLE;
                ra_addr_e <= inst_h[18:24];
                rb_addr_e <= inst_h[11:17];
                rt_addr_e <= inst_h[25:31];
            end
            11'b01001100???: begin      //EXE_CGTI
                pipe_h = `PIPE_EVEN;
                uid_e <= `UID_1;
                wreg_e = `WR_ENABLE;
                op_code_e <= `EXE_CGTI;
                ra_rd_e <= `RD_ENABLE; //read imm from ra
                rb_rd_e <= `RD_DISABLE;
                ra_addr_e <= inst_h[18:24];
                //imm_e <= inst_h[8:17];
                word_t = {{22{inst_h[8]}},inst_h[8:17]}; //Extended to 32bits by replicating
```

```
leftmost bit
                imm_e <= {word_t,word_t,word_t,word_t};
                rt_addr_e <= inst_h[25:31];
            end
            11'b001100100??: begin      //EXE_BR
                pipe_h = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_DISABLE;
                op_code_o <= `EXE_BR;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00}; // Extended on
the right with 2 0bits
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100000??: begin      //EXE_BRA
                pipe_h = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_DISABLE;
                op_code_o <= `EXE_BRA;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100110??: begin      //EXE_BRSL
                pipe_h = `PIPE_ODD;
                uid_o <= `UID_7;
                wreg_o = `WR_ENABLE;
                op_code_o <= `EXE_BRSL;
                ra_rd_o <= `RD_DISABLE;
                rb_rd_o <= `RD_DISABLE;
                rt_addr_o <= inst_h[25:31];
                next_inst_in_delayslot_o <= `IN_DELAYSLOT;
                branch_flag_o <= `BRANCH;
                branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
                link_addr_o <= `ZERO_WORD32;
            end
            11'b001100010??: begin      //EXE_BRASL
                pipe_h = `PIPE_ODD;
```

```
            uid_o <= `UID_7;
            wreg_o = `WR_ENABLE;
            op_code_o <= `EXE_BRASL;
            ra_rd_o <= `RD_DISABLE;
            rb_rd_o <= `RD_DISABLE;
            rt_addr_o <= inst_h[25:31];
            next_inst_in_delayslot_o <= `IN_DELAYSLOT;
            branch_flag_o <= `BRANCH;
            branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
            link_addr_o <= `ZERO_WORD32;
        end
        11'b00110101000: begin              //EXE_BI
            pipe_h = `PIPE_ODD;
            uid_o <= `UID_7;
            wreg_o = `WR_DISABLE;
            op_code_o <= `EXE_BI;
            ra_rd_o <= `RD_ENABLE;
            rb_rd_o <= `RD_DISABLE;
            ra_addr_o <= inst_h[18:24];
            next_inst_in_delayslot_o <= `IN_DELAYSLOT;
            branch_flag_o <= `BRANCH;
            //branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
            link_addr_o <= `ZERO_WORD32;
        end
        11'b00110101001: begin              //EXE_BISL
            pipe_h = `PIPE_ODD;
            uid_o <= `UID_7;
            wreg_o = `WR_ENABLE;
            op_code_o <= `EXE_BISL;
            ra_rd_o <= `RD_ENABLE;
            rb_rd_o <= `RD_DISABLE;
            ra_addr_o <= inst_h[18:24];
            rt_addr_o <= inst_h[25:31];
            next_inst_in_delayslot_o <= `IN_DELAYSLOT;
            branch_flag_o <= `BRANCH;
            //branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
            link_addr_o <= `ZERO_WORD32;
        end
        11'b001000010??: begin      //EXE_BRNZ
            pipe_h = `PIPE_ODD;
            uid_o <= `UID_7;
            wreg_o = `WR_DISABLE; //RT only used to READ for BRNZ
            op_code_o <= `EXE_BRNZ;
            ra_rd_o <= `RD_DISABLE;
```

```
        rb_rd_o <= `RD_DISABLE;
        rt_addr_o <= inst_h[25:31];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000000??: begin        //EXE_BRZ
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE; //RT only used to READ for BRZ
        op_code_o <= `EXE_BRZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        rt_addr_o <= inst_h[25:31];
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000110??: begin        //EXE_BRHNZ
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_BRHNZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
    11'b001000100??: begin        //EXE_BRHZ
        pipe_h = `PIPE_ODD;
        uid_o <= `UID_7;
        wreg_o = `WR_DISABLE;
        op_code_o <= `EXE_BRHZ;
        ra_rd_o <= `RD_DISABLE;
        rb_rd_o <= `RD_DISABLE;
        next_inst_in_delayslot_o <= `IN_DELAYSLOT;
        branch_flag_o <= `BRANCH;
        branch_target_addr_o <= {{14{inst_h[9]}},inst_h[9:24],2'b00};
        link_addr_o <= `ZERO_WORD32;
    end
```

```
11'b01011000100: begin          //EXE_FA
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_2;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_FA;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_ENABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b01011000101: begin          //EXE_FS
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_2;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_FA;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_ENABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b01011000110: begin          //EXE_FM
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_2;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_FM;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_ENABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b01111000010: begin          //EXE_FCEQ
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_2;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_FCEQ;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_ENABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
```

```
11'b01011000010: begin          //EXE_FCGT
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_2;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_FCGT;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_ENABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b01010110100: begin          //EXE_CNTB
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_3;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_CNTB;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_DISABLE;
    ra_addr_e <= inst_h[18:24];
    //rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b00011010011: begin          //EXE_AVGB
    pipe_h = `PIPE_EVEN;
    uid_e <= `UID_3;
    wreg_e = `WR_ENABLE;
    op_code_e <= `EXE_AVGB;
    ra_rd_e <= `RD_ENABLE;
    rb_rd_e <= `RD_DISABLE;
    ra_addr_e <= inst_h[18:24];
    rb_addr_e <= inst_h[11:17];
    rt_addr_e <= inst_h[25:31];
end
11'b00000000000: begin          //EXE_STOP
    //???
end
11'b00000000001: begin          //EXE_LNOP
    pipe_h = `PIPE_ODD;
    uid_o <= `UID_0;
    wreg_o = `WR_DISABLE;
    op_code_o <= `EXE_LNOP;
    ra_rd_o <= `RD_DISABLE;
    rb_rd_o <= `RD_DISABLE;
    rt_addr_o <= 0;
```

```
                end
                11'b01000000001: begin              //EXE_NOP
                    pipe_h = `PIPE_EVEN;
                    uid_e <= `UID_0;
                    wreg_e = `WR_DISABLE;
                    op_code_e <= `EXE_NOP;
                    ra_rd_e <= `RD_DISABLE;
                    rb_rd_e <= `RD_DISABLE;
                    rt_addr_e <= 0;
                end
                default: begin
                end
            endcase
        end

        always @ (*) begin
            // Do not need to consider these 2 conditions because EVEN and odd pipe variables already split in Decode module
            //if ((pipe_h==`PIPE_EVEN)&&(pipe_h==`PIPE_ODD))begin end
            //if ((pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_EVEN)) begin end

            if ((pipe_l==`PIPE_ODD)&&(pipe_h==`PIPE_ODD)) begin
                stallreq <= 1; //stall == 7
            end else if ((pipe_l==`PIPE_EVEN)&&(pipe_h==`PIPE_EVEN)) begin
                stallreq <= 1; //stall == 7
            end else begin
                stallreq <= 0;
            end
        end

        // is_in_delayslot_o: is the present instruction in ID a delayslot instruction
        always @ (*) begin
            if (rst == `RST_ENABLE) begin
                is_in_delayslot_o <= `NOT_IN_DELAYSLOT;
            end else begin
                is_in_delayslot_o <= `IN_DELAYSLOT;
            end
        end

        // Obtain even pipe operand ra & rb
        always @ (*) begin
            if(rst == `RST_ENABLE) begin
                ra_do_e <= `ZERO_QWORD128;
            end                                     else                                    if
```

```
((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_1)&&(ff2_wreg_i_e==`WR_ENABLE)&&(ff2_waddr_i_e
==ra_addr_e)) begin //fowarding data from FF network
                ra_do_e <= ff2_rt_e;
                stallreq_dh <= 0;
        end                                             else                                    if
(((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==ra_addr_e))) begin
                ra_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
        end                                             else                                    if
((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_3)&&(ff4_wreg_i_e==`WR_ENABLE)&&(ff4_waddr_i_e
==ra_addr_e)) begin
                ra_do_e <= ff4_rt_e;
                stallreq_dh <= 0;
        end                                             else                                    if
(((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&

    (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e)))
begin
                ra_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
        end                                             else                                    if
((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_2)&&(ff6_wreg_i_e==`WR_ENABLE)&&(ff6_waddr_i_e
==ra_addr_e)) begin
                ra_do_e <= ff6_rt_e;
                stallreq_dh <= 0;
        end                                             else                                    if
(((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&

    (ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||(
(ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||
                ((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))) begin
                ra_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
        end                                             else                                    if
((ra_rd_e==`RD_ENABLE)&&(uid_e==`UID_4)&&(ff7_wreg_i_e==`WR_ENABLE)&&(ff7_waddr_i_e
==ra_addr_e)) begin
                ra_do_e <= ff7_rt_e;
                stallreq_dh <= 0;
        end                                             else                                    if
(((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&
```

```
(ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||(
(ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||

((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff6_
waddr_i_e==ra_addr_e))) begin
            ra_do_e <= `ZERO_QWORD128;
            stallreq_dh <= 1;
        end                                             else                                            if
((ra_rd_e==`RD_ENABLE)&&(mem_wreg_i_e==`WR_ENABLE)&&(mem_waddr_i_e==ra_addr_e))
begin //fowarding data from mem
            ra_do_e <= mem_wdata_i_e;
            stallreq_dh <= 0;
        end                                             else                                            if
(((ra_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&

(ff2_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==ra_addr_e))||(
(ra_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==ra_addr_e))||

((ra_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==ra_addr_e))||((ra_rd_e==`RD_ENABLE)&&(ff6_
waddr_i_e==ra_addr_e))||
                ((ra_rd_e==`RD_ENABLE)&&(ff7_waddr_i_e==ra_addr_e))) begin
            ra_do_e <= `ZERO_QWORD128;
            stallreq_dh <= 1;
        end else if(ra_rd_e==1'b1) begin
            ra_do_e <= ra_i_e;
        end else if(ra_rd_e==1'b0) begin
            ra_do_e <= imm_e;

        end
    end

    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            rb_do_e <= `ZERO_QWORD128;
        end                                             else                                            if
((rb_rd_e==`RD_ENABLE)&&(uid_e==`UID_1)&&(ff2_wreg_i_e==`WR_ENABLE)&&(ff2_waddr_i_e
==rb_addr_e)) begin //fowarding data from FF network
            rb_do_e <= ff2_rt_e;
            stallreq_dh <= 0;
        end                                             else                                            if
(((rb_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==rb_addr_e))) begin
```

```
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                         else                                        if
((rb_rd_e==`RD_ENABLE)&&(uid_e==`UID_3)&&(ff4_wreg_i_e==`WR_ENABLE)&&(ff4_waddr_i_e
==rb_addr_e)) begin
                rb_do_e <= ff4_rt_e;
                stallreq_dh <= 0;
            end                                         else                                        if
(((rb_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&

    (ff2_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==rb_addr_e)))
begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                         else                                        if
((rb_rd_e==`RD_ENABLE)&&(uid_e==`UID_2)&&(ff6_wreg_i_e==`WR_ENABLE)&&(ff6_waddr_i_e
==rb_addr_e)) begin
                rb_do_e <= ff6_rt_e;
                stallreq_dh <= 0;
            end                                         else                                        if
(((rb_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&

    (ff2_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==rb_addr_e))||(
(rb_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==rb_addr_e))||
                    ((rb_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==rb_addr_e))) begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                         else                                        if
((rb_rd_e==`RD_ENABLE)&&(uid_e==`UID_4)&&(ff7_wreg_i_e==`WR_ENABLE)&&(ff7_waddr_i_e
==rb_addr_e)) begin
                rb_do_e <= ff7_rt_e;
                stallreq_dh <= 0;
            end                                         else                                        if
(((rb_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&

    (ff2_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==rb_addr_e))||(
(rb_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==rb_addr_e))||

    ((rb_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff6
_waddr_i_e==rb_addr_e))) begin
                rb_do_e <= `ZERO_QWORD128;
```

```
                stallreq_dh <= 1;
            end                                                 else                                                if
((rb_rd_e==`RD_ENABLE)&&(mem_wreg_i_e==`WR_ENABLE)&&(mem_waddr_i_e==rb_addr_e))
begin //fowarding data from mem
                rb_do_e <= mem_wdata_i_e;
                stallreq_dh <= 0;
            end                                                 else                                                if
(((rb_rd_e==`RD_ENABLE)&&(ex_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff1_wad
dr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&

        (ff2_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff3_waddr_i_e==rb_addr_e))||(
(rb_rd_e==`RD_ENABLE)&&(ff4_waddr_i_e==rb_addr_e))||

        ((rb_rd_e==`RD_ENABLE)&&(ff5_waddr_i_e==rb_addr_e))||((rb_rd_e==`RD_ENABLE)&&(ff6
_waddr_i_e==rb_addr_e))||
                        ((rb_rd_e==`RD_ENABLE)&&(ff7_waddr_i_e==rb_addr_e))) begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end else if(rb_rd_e==1'b1) begin
                rb_do_e <= rb_i_e;
            end else if(rb_rd_e==1'b0) begin
                rb_do_e <= imm_e;
            end
        end

        always @ (*) begin
            if (rst ==`RST_ENABLE) begin
                is_in_delayslot_o <= `NOT_IN_DELAYSLOT;
            end else begin
                is_in_delayslot_o <= is_in_delayslot_i;
            end
        end

        // Obtain odd pipe operand ra & rb
        always @ (*) begin
            if(rst == `RST_ENABLE) begin
                ra_do_o <= `ZERO_QWORD128;
            end                                                 else                                                if
(((ra_rd_o==`RD_ENABLE)&&(uid_o==`UID_4)&&(ff4_wreg_i_o==`WR_ENABLE)&&(ff4_waddr_i_o
==ra_addr_o)) ||

        ((ra_rd_o==`RD_ENABLE)&&(uid_o==`UID_7)&&(ff4_wreg_i_o==`WR_ENABLE)&&(ff4_waddr
_i_o==ra_addr_o))) begin //fowarding data from FF network
                ra_do_o <= ff4_rt_o;
```

```
                stallreq_dh <= 0;
            end                                                 else                                                if
(((ra_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&(ff1_wad
dr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&

    (ff2_waddr_i_o==ra_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==ra_addr_o)))
begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                                 else                                                if
(((ra_rd_o==`RD_ENABLE)&&(uid_o==`UID_5)&&(ff6_wreg_i_o==`WR_ENABLE)&&(ff6_waddr_i_o
==ra_addr_o)) ||

    ((ra_rd_o==`RD_ENABLE)&&(uid_o==`UID_6)&&(ff6_wreg_i_o==`WR_ENABLE)&&(ff6_waddr
_i_o==ra_addr_o)))begin
                ra_do_o <= ff6_rt_o;
                stallreq_dh <= 0;
            end                                                 else                                                if
(((ra_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&(ff1_wad
dr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&

    (ff2_waddr_i_o==ra_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==ra_addr_o))||(
(ra_rd_o==`RD_ENABLE)&&(ff4_waddr_i_o==ra_addr_o))||
                    ((ra_rd_o==`RD_ENABLE)&&(ff5_waddr_i_o==ra_addr_o))) begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                                 else                                                if
((ra_rd_o==`RD_ENABLE)&&(mem_wreg_i_o==`WR_ENABLE)&&(mem_waddr_i_o==ra_addr_o))
begin //fowarding data from mem
                ra_do_o <= mem_wdata_i_o;
                stallreq_dh <= 0;
            end                                                 else                                                if
(((ra_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&(ff1_wad
dr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&

    (ff2_waddr_i_o==ra_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==ra_addr_o))||(
(ra_rd_o==`RD_ENABLE)&&(ff4_waddr_i_o==ra_addr_o))||

    ((ra_rd_o==`RD_ENABLE)&&(ff5_waddr_i_o==ra_addr_o))||((ra_rd_o==`RD_ENABLE)&&(ff6
_waddr_i_o==ra_addr_o))||
                    ((ra_rd_o==`RD_ENABLE)&&(ff7_waddr_i_o==ra_addr_o))) begin
                rb_do_e <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end else if(ra_rd_o==1'b1) begin
```

```
                ra_do_o <= ra_i_o;
            end else if(ra_rd_o==1'b0) begin
                ra_do_o <= imm_o;
            end
        end

        always @ (*) begin
            if(rst == `RST_ENABLE) begin
                rb_do_o <= `ZERO_QWORD128;
            end                                                 else                                                if
(((rb_rd_o==`RD_ENABLE)&&(uid_o==`UID_4)&&(ff4_wreg_i_o==`WR_ENABLE)&&(ff4_waddr_i_
o==rb_addr_o)) ||

        ((rb_rd_o==`RD_ENABLE)&&(uid_o==`UID_7)&&(ff4_wreg_i_o==`WR_ENABLE)&&(ff4_waddr
_i_o==rb_addr_o)))begin //fowarding data from FF network
                rb_do_o <= ff4_rt_o;
                stallreq_dh <= 0;
            end                                                 else                                                if
(((rb_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff1_wa
ddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&

        (ff2_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==rb_addr_o)))
begin
                rb_do_o <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                                 else                                                if
(((rb_rd_o==`RD_ENABLE)&&(uid_o==`UID_5)&&(ff6_wreg_i_o==`WR_ENABLE)&&(ff6_waddr_i_
o==rb_addr_o))||

        ((rb_rd_o==`RD_ENABLE)&&(uid_o==`UID_6)&&(ff6_wreg_i_o==`WR_ENABLE)&&(ff6_waddr
_i_o==rb_addr_o))) begin
                rb_do_o <= ff6_rt_o;
                stallreq_dh <= 0;
            end                                                 else                                                if
(((rb_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff1_wa
ddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&

        (ff2_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==rb_addr_o))||
((rb_rd_o==`RD_ENABLE)&&(ff4_waddr_i_o==rb_addr_o))||
                    ((rb_rd_o==`RD_ENABLE)&&(ff5_waddr_i_o==rb_addr_o))) begin
                rb_do_o <= `ZERO_QWORD128;
                stallreq_dh <= 1;
            end                                                 else                                                if
((rb_rd_o==`RD_ENABLE)&&(mem_wreg_i_o==`WR_ENABLE)&&(mem_waddr_i_o==rb_addr_o))
```

```
begin //fowarding data from mem
                rb_do_o <= mem_wdata_i_o;
                stallreq_dh <= 0;
        end                                             else                                            if
(((rb_rd_o==`RD_ENABLE)&&(ex_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff1_wa
ddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&

    (ff2_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff3_waddr_i_o==rb_addr_o))||
((rb_rd_o==`RD_ENABLE)&&(ff4_waddr_i_o==rb_addr_o))||

    ((rb_rd_o==`RD_ENABLE)&&(ff5_waddr_i_o==rb_addr_o))||((rb_rd_o==`RD_ENABLE)&&(ff6
_waddr_i_o==rb_addr_o))||
                ((rb_rd_o==`RD_ENABLE)&&(ff7_waddr_i_o==rb_addr_o))) begin
                rb_do_o <= `ZERO_QWORD128;
                stallreq_dh <= 1;
        end else if(rb_rd_o==1'b1) begin
                rb_do_o <= rb_i_o;
        end else if(rb_rd_o==1'b0) begin
                rb_do_o <= imm_o;
        end else begin
                rb_do_o <= `ZERO_QWORD128;
        end
    end



endmodule


//************************************************************************
********/
// Module:      REGFILE
// File:        regfile.v
// Description: Register File
//          SPU architecture defines a set of 128 general-purpose registers (GPRs),
//          each of which contains 128 data bits.
// History:     Created by Ning Kang, Mar 25, 2018
//************************************************************************
**********/

`include "defines.v"

module REGFILE(
```

```
    input wire clk,
    input wire rst,

    //Write register: even pipe w_e(rt_e) and odd pipe w_o(rt_o)
    input wire rt_we_e, rt_we_o, // rt: even write enable & odd write enable
    input wire[`REG_ADDR_BUS7] rt_addr_e, rt_addr_o,// rt address
    input wire[`REG_BUS128] rt_data_e, rt_data_o, // rt data

    //Read register: even pipe ra_e, rb_e rt_e and odd pipe ra_o, rb_o, rt_o
    input wire ra_e_e, rb_e_e, ra_e_o, rb_e_o, // even ra/rb enable & odd ra/rb enable
    input wire[`REG_ADDR_BUS7] ra_addr_e, rb_addr_e, ra_addr_o, rb_addr_o,

    output reg[`REG_BUS128] ra_data_e, rb_data_e, ra_data_o, rb_data_o

);

    reg[`REG_BUS128] regs[0:`REG_NUM128-1];

    //********************************************** Start... only for test
    initial begin
    regs[7'b0000001]<=128'h00010001000100010001000100010001;
    regs[7'b0000010]<=128'h00020002000200020002000200020002;
    /*
    regs[7'b0000000]<=128'h00000000000000000000000000000001;
    regs[7'b0000001]<=128'h00110011001100110011001100110011;
    regs[7'b0000010]<=128'h00000000000000000000000000000022;
    regs[7'b0000011]<=128'h33333333333333333333333333333333;
    regs[7'b0000100]<=128'h44444444444444444444444444444444;
    regs[7'b0000101]<=128'h55005500550055005500550055005500;
    regs[7'b0000110]<=128'h00000066000000660000006600000066;
    regs[7'b0000111]<=128'h55555555555555555555555555555555;
    regs[7'b0001000]<=128'haaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa;
    regs[7'b0001001]<=128'h00005555000055550000555500005555;
    regs[7'b0001010]<=128'h01550155015501550155015501550155;
    regs[7'b0001011]<=128'h00560056005600560056005600560056;
    regs[7'b0001100]<=128'h00001111000011110000111100001111;
    regs[7'b0001101]<=128'h55550000555500005555000055550000;
    regs[7'b0001110]<=128'h00000000111111110000000011111111;
    regs[7'b0001111]<=128'h55555555000000005555555500000000;
    */
    end
    //**********************************************  Test end

    //RT write
```

```
    always @ (posedge clk) begin
        if (rst == `RST_DISABLE) begin
            if(rt_we_e == `WR_ENABLE) begin
                regs[rt_addr_e] <= rt_data_e;
            end
        end
        if (rst == `RST_DISABLE) begin
            if(rt_we_o == `WR_ENABLE) begin
                regs[rt_addr_o] <= rt_data_o;
            end
        end
    end

    //RA read
    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            ra_data_e <= `ZERO_WORD32;
        end else if((ra_addr_e == rt_addr_e) && (rt_we_e == `WR_ENABLE) && (ra_e_e ==
`RD_ENABLE)) begin //avoid RAW between 1st inst and 11th inst
            ra_data_e <= rt_data_e;
        end else if(ra_e_e == `RD_ENABLE) begin
                ra_data_e <= regs[ra_addr_e];
        end else begin
                ra_data_e <= `ZERO_WORD32;
        end

        if(rst == `RST_ENABLE) begin
            ra_data_o <= `ZERO_WORD32;
        end else if((ra_addr_o == rt_addr_o) && (rt_we_o == `WR_ENABLE) && (ra_e_o ==
`RD_ENABLE)) begin
            ra_data_o <= rt_data_o;
        end else if(ra_e_o == `RD_ENABLE) begin
                ra_data_o <= regs[ra_addr_o];
        end else begin
                ra_data_o <= `ZERO_WORD32;
        end
    end

    //RB read
    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            rb_data_e <= `ZERO_WORD32;
        end else if((rb_addr_e == rt_addr_e) && (rt_we_e == `WR_ENABLE) && (rb_e_e ==
`RD_ENABLE)) begin
```

```
                rb_data_e <= rt_data_e;
            end else if(ra_e_e == `RD_ENABLE) begin
                    rb_data_e <= regs[rb_addr_e];
            end else begin
                    rb_data_e <= `ZERO_WORD32;
            end

            if(rst == `RST_ENABLE) begin
                rb_data_o <= `ZERO_WORD32;
            end else if((rb_addr_o == rt_addr_o) && (rt_we_o == `WR_ENABLE) && (rb_e_o ==
`RD_ENABLE)) begin
                    rb_data_o <= rt_data_o;
            end else if(rb_e_o == `RD_ENABLE) begin
                    rb_data_o <= regs[rb_addr_o];
            end else begin
                    rb_data_o <= `ZERO_WORD32;
            end
        end

endmodule


//************************************************************************
*********/
// Module:      ID_EX
// File:        id_ex.v
// Description: registers foward to ex between ID and EX
// History:     Created by Ning Kang, Mar 27, 2018
//              Modify by Ning Kang, Apr 20, 2018 - added branch variables
//************************************************************************
**********/

`include "defines.v"

module ID_EX(
        input wire clk,
        input wire rst,

        //info from ID
        input wire[0:31] id_pc,

        input wire[0:31] id_inst_l,
        input wire[0:31] id_inst_h,
```

```
input wire[0:10] id_opcode_e,
input wire[0:127] id_ra_e,
input wire[0:127] id_rb_e,
input wire[0:6] id_rtaddr_e,
input wire[0:2] id_uid_e,
//input wire id_even,
input wire id_wreg_e,

input wire[0:10] id_opcode_o,
input wire[0:127] id_ra_o,
input wire[0:127] id_rb_o,
input wire[0:6] id_rtaddr_o,
input wire[0:2] id_uid_o,
input wire[0:9] id_imm_o,
//input wire id_odd,
input wire id_wreg_o,

input wire[0:12] stall,

input wire[0:31] id_link_addr,
input wire id_is_in_delayslot,
input wire id_next_inst_in_delayslot,
input wire[0:31] id_branch_target_addr,
input wire id_branch_flag,

//info to EX
output reg[0:31] ex_pc,
output reg[0:31] ex_inst_l,
output reg[0:31] ex_inst_h,

output reg[0:10] ex_opcode_e,
output reg[0:127] ex_ra_e,
output reg[0:127] ex_rb_e,
output reg[0:6] ex_rtaddr_e,
output reg ex_wreg_e,
output reg[0:2] ex_uid_e,
//output reg ex_even,


output reg[0:10] ex_opcode_o,
output reg[0:127] ex_ra_o,
output reg[0:127] ex_rb_o,
output reg[0:6] ex_rtaddr_o,
output reg ex_wreg_o,
```

```
    output reg[0:2] ex_uid_o,
    output reg[0:9] ex_imm_o,
    //output reg ex_odd,

    output reg[0:31] ex_link_addr,
    output reg ex_is_in_delayslot,
    output reg[0:31] ex_branch_target_addr,
    output reg ex_branch_flag,

    output reg is_in_delayslot_o
);

    always @ (posedge clk) begin
        if (rst == `RST_ENABLE) begin
            ex_opcode_e <= `EXE_NOP;
            ex_ra_e <= `ZERO_WORD32;
            ex_rb_e <= `ZERO_WORD32;
            ex_rtaddr_e <= `NOP_REG_ADDR;
            ex_wreg_e <= `WR_DISABLE;
            ex_opcode_o <= `EXE_NOP;
            ex_ra_o <= `ZERO_WORD32;
            ex_rb_o <= `ZERO_WORD32;
            ex_rtaddr_o <= `NOP_REG_ADDR;
            ex_wreg_o <= `WR_DISABLE;
            ex_link_addr <= `ZERO_WORD32;
            ex_is_in_delayslot <= `NOT_IN_DELAYSLOT;
            is_in_delayslot_o <= `NOT_IN_DELAYSLOT;
            ex_branch_target_addr <= `ZERO_WORD32;
            ex_branch_flag <= `NOTBRANCH;
            ex_pc <= 0;
        end else if(stall[2] == `STOP && stall[3] == `NOSTOP) begin
            ex_opcode_e <= `EXE_NOP;
            ex_ra_e <= `ZERO_WORD32;
            ex_rb_e <= `ZERO_WORD32;
            ex_rtaddr_e <= `NOP_REG_ADDR;
            ex_wreg_e <= `WR_DISABLE;
            ex_opcode_o <= `EXE_NOP;
            ex_ra_o <= `ZERO_WORD32;
            ex_rb_o <= `ZERO_WORD32;
            ex_rtaddr_o <= `NOP_REG_ADDR;
            ex_wreg_o <= `WR_DISABLE;
            ex_wreg_o <= `WR_DISABLE;
            ex_link_addr <= `ZERO_WORD32;
            ex_is_in_delayslot <= `NOT_IN_DELAYSLOT;
```

```
                is_in_delayslot_o <= `NOT_IN_DELAYSLOT;
                ex_branch_target_addr <= `ZERO_WORD32;
                ex_branch_flag <= `NOTBRANCH;
                ex_pc <= 0;
            end else if(stall[2] == `NOSTOP)begin
                ex_opcode_e <= id_opcode_e;
                ex_ra_e <= id_ra_e;
                ex_rb_e <= id_rb_e;
                ex_rtaddr_e <= id_rtaddr_e;
                ex_uid_e <= id_uid_e;
                //ex_even <= id_even;
                ex_wreg_e <= id_wreg_e;
                ex_opcode_o <= id_opcode_o;
                ex_ra_o <= id_ra_o;
                ex_rb_o <= id_rb_o;
                ex_rtaddr_o <= id_rtaddr_o;
                ex_uid_o <= id_uid_o;
                ex_imm_o <= id_imm_o;
                //ex_odd <= id_odd;
                ex_wreg_o <= id_wreg_o;
                ex_link_addr <= id_link_addr;
                ex_is_in_delayslot <= id_is_in_delayslot;
                is_in_delayslot_o <= id_next_inst_in_delayslot;
                ex_branch_target_addr <= id_branch_target_addr;
                ex_branch_flag <= id_branch_flag;
                ex_inst_l <= id_inst_l;
                ex_inst_h <= id_inst_h;
                ex_pc <= id_pc;
            end
        end

endmodule


//************************************************************************************
*********/
// Module:      EX
// File:        ex.v
// Description: instruction execution stage
// History:     Created by Weilun Cheng, Mar 15,2018
//          Modify by Ning Kang, Mar 27, 2018
//              - implement even/odd pipe
//          Modify by Ning Kang, Apr 20, 2018
//              - split even/odd pipe
```

```
//                  - add clk cycle signal to simulate operation latency
//*************************************************************************
**********/

`include "defines.v"

//Even pipe
module EX_EVEN(

      input wire clk,
      input wire rst,

      input wire[0:10] i_opcode_e,
      input wire[0:127] i_ra_e,
      input wire[0:127] i_rb_e,
      input wire[0:6] i_rtaddr_e,
      input wire i_wreg_e,
      input wire[0:2] i_uid_e,
      //input wire i_even,

      output reg[0:127] o_rt_e,
      output reg[0:6] o_rtaddr_e,
      output reg o_wreg_e,
      output reg[0:2] o_uid_e
      //output reg o_even

);


      reg [0:7] ra_temp_byte [0:15];
      reg [0:7] rb_temp_byte [0:15];
      reg [0:7] rt_temp_byte [0:15];
      reg [0:15] ra_temp_hword [0:7];
      reg [0:15] rb_temp_hword [0:7];
      reg [0:15] rt_temp_hword [0:7];
      reg [0:31] ra_temp_word [0:3];
      reg [0:31] rb_temp_word [0:3];
      reg [0:31] rt_temp_word [0:3];
      reg [0:63] rt_temp_qword [0:3];
      reg [0:7] float_diff;
      reg[0:31] max,min;

      integer i;
      integer c;
```

```
integer b;
integer m;

reg [0:31] s;

always @ (*) begin
    if(rst == `RST_ENABLE) begin
        o_rt_e <= `ZERO_QWORD128;
    end else begin
        case (i_opcode_e)
            `EXE_AH: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] + i_rb_e[`HALFWORD0];
                o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] + i_rb_e[`HALFWORD1];
                o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] + i_rb_e[`HALFWORD2];
                o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] + i_rb_e[`HALFWORD3];
                o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] + i_rb_e[`HALFWORD4];
                o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] + i_rb_e[`HALFWORD5];
                o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] + i_rb_e[`HALFWORD6];
                o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] + i_rb_e[`HALFWORD7];
            end
            `EXE_AHI: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] + i_rb_e[`HALFWORD0];
                o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] + i_rb_e[`HALFWORD1];
                o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] + i_rb_e[`HALFWORD2];
                o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] + i_rb_e[`HALFWORD3];
                o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] + i_rb_e[`HALFWORD4];
                o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] + i_rb_e[`HALFWORD5];
                o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] + i_rb_e[`HALFWORD6];
                o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] + i_rb_e[`HALFWORD7];
            end
            `EXE_A: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= i_ra_e[`WORD0] + i_rb_e[`WORD0];
```

```
                o_rt_e[`WORD1] <= i_ra_e[`WORD1] + i_rb_e[`WORD1];
                o_rt_e[`WORD2] <= i_ra_e[`WORD2] + i_rb_e[`WORD2];
                o_rt_e[`WORD3] <= i_ra_e[`WORD3] + i_rb_e[`WORD3];
            end
            `EXE_AI: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= i_ra_e[`WORD0] + i_rb_e[`WORD0];
                o_rt_e[`WORD1] <= i_ra_e[`WORD1] + i_rb_e[`WORD1];
                o_rt_e[`WORD2] <= i_ra_e[`WORD2] + i_rb_e[`WORD2];
                o_rt_e[`WORD3] <= i_ra_e[`WORD3] + i_rb_e[`WORD3];
            end
            `EXE_SFH: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`HALFWORD0] <= (~i_ra_e[`HALFWORD0]) + i_rb_e[`HALFWORD0]
+ 1;
                o_rt_e[`HALFWORD1] <= (~i_ra_e[`HALFWORD1]) + i_rb_e[`HALFWORD1]
+ 1;
                o_rt_e[`HALFWORD2] <= (~i_ra_e[`HALFWORD2]) + i_rb_e[`HALFWORD2]
+ 1;
                o_rt_e[`HALFWORD3] <= (~i_ra_e[`HALFWORD3]) + i_rb_e[`HALFWORD3]
+ 1;
                o_rt_e[`HALFWORD4] <= (~i_ra_e[`HALFWORD4]) + i_rb_e[`HALFWORD4]
+ 1;
                o_rt_e[`HALFWORD5] <= (~i_ra_e[`HALFWORD5]) + i_rb_e[`HALFWORD5]
+ 1;
                o_rt_e[`HALFWORD6] <= (~i_ra_e[`HALFWORD6]) + i_rb_e[`HALFWORD6]
+ 1;
                o_rt_e[`HALFWORD7] <= (~i_ra_e[`HALFWORD7]) + i_rb_e[`HALFWORD7]
+ 1;
            end
            `EXE_SFHI: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`HALFWORD0] <= (~i_ra_e[`HALFWORD0]) + i_rb_e[`HALFWORD0]
+ 1;
                o_rt_e[`HALFWORD1] <= (~i_ra_e[`HALFWORD1]) + i_rb_e[`HALFWORD1]
```

```
+ 1;
                o_rt_e[`HALFWORD2] <= (~i_ra_e[`HALFWORD2]) + i_rb_e[`HALFWORD2]
+ 1;
                o_rt_e[`HALFWORD3] <= (~i_ra_e[`HALFWORD3]) + i_rb_e[`HALFWORD3]
+ 1;
                o_rt_e[`HALFWORD4] <= (~i_ra_e[`HALFWORD4]) + i_rb_e[`HALFWORD4]
+ 1;
                o_rt_e[`HALFWORD5] <= (~i_ra_e[`HALFWORD5]) + i_rb_e[`HALFWORD5]
+ 1;
                o_rt_e[`HALFWORD6] <= (~i_ra_e[`HALFWORD6]) + i_rb_e[`HALFWORD6]
+ 1;
                o_rt_e[`HALFWORD7] <= (~i_ra_e[`HALFWORD7]) + i_rb_e[`HALFWORD7]
+ 1;
            end
            `EXE_SF: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= (~i_ra_e[`WORD0]) + i_rb_e[`WORD0] + 1;
                o_rt_e[`WORD1] <= (~i_ra_e[`WORD1]) + i_rb_e[`WORD1] + 1;
                o_rt_e[`WORD2] <= (~i_ra_e[`WORD2]) + i_rb_e[`WORD2] + 1;
                o_rt_e[`WORD3] <= (~i_ra_e[`WORD3]) + i_rb_e[`WORD3] + 1;
            end
            `EXE_SFI: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= (~i_ra_e[`WORD0]) + i_rb_e[`WORD0] + 1;
                o_rt_e[`WORD1] <= (~i_ra_e[`WORD1]) + i_rb_e[`WORD1] + 1;
                o_rt_e[`WORD2] <= (~i_ra_e[`WORD2]) + i_rb_e[`WORD2] + 1;
                o_rt_e[`WORD3] <= (~i_ra_e[`WORD3]) + i_rb_e[`WORD3] + 1;
            end
            `EXE_MPY: begin
                o_wreg_e <= i_wreg_e;
                o_rtaddr_e <= i_rtaddr_e;
                o_uid_e <= i_uid_e;
                //o_even <= `PIPE_EVEN;
                o_rt_e[`WORD0] <= i_ra_e[`HWORD_R16B0] * i_rb_e[`HWORD_R16B0];
                o_rt_e[`WORD1] <= i_ra_e[`HWORD_R16B1] * i_rb_e[`HWORD_R16B1];
                o_rt_e[`WORD2] <= i_ra_e[`HWORD_R16B2] * i_rb_e[`HWORD_R16B2];
                o_rt_e[`WORD3] <= i_ra_e[`HWORD_R16B3] * i_rb_e[`HWORD_R16B3];
            end
```

```
`EXE_MPYU: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`HWORD_R16B0] * i_rb_e[`HWORD_R16B0];
        o_rt_e[`WORD1] <= i_ra_e[`HWORD_R16B1] * i_rb_e[`HWORD_R16B1];
        o_rt_e[`WORD2] <= i_ra_e[`HWORD_R16B2] * i_rb_e[`HWORD_R16B2];
        o_rt_e[`WORD3] <= i_ra_e[`HWORD_R16B3] * i_rb_e[`HWORD_R16B3];
    end
    `EXE_MPYI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`HWORD_R16B0] * i_rb_e[`HWORD_R16B0];
        o_rt_e[`WORD1] <= i_ra_e[`HWORD_R16B1] * i_rb_e[`HWORD_R16B1];
        o_rt_e[`WORD2] <= i_ra_e[`HWORD_R16B2] * i_rb_e[`HWORD_R16B2];
        o_rt_e[`WORD3] <= i_ra_e[`HWORD_R16B3] * i_rb_e[`HWORD_R16B3];
    end
    `EXE_AND: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] & i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] & i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] & i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] & i_rb_e[`WORD3];
    end
    `EXE_ANDBI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] & i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] & i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] & i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] & i_rb_e[`WORD3];
    end
    `EXE_ANDHI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
```

```
        //o_even <= `PIPE_EVEN;
        o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] & i_rb_e[`HALFWORD0];
        o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] & i_rb_e[`HALFWORD1];
        o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] & i_rb_e[`HALFWORD2];
        o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] & i_rb_e[`HALFWORD3];
        o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] & i_rb_e[`HALFWORD4];
        o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] & i_rb_e[`HALFWORD5];
        o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] & i_rb_e[`HALFWORD6];
        o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] & i_rb_e[`HALFWORD7];
    end
    `EXE_ANDI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] & i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] & i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] & i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] & i_rb_e[`WORD3];
    end
    `EXE_OR: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] | i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] | i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] | i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] | i_rb_e[`WORD3];
    end
    `EXE_ORBI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] | i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] | i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] | i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] | i_rb_e[`WORD3];
    end
    `EXE_ORHI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
```

```
        //o_even <= `PIPE_EVEN;
        o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] | i_rb_e[`HALFWORD0];
        o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] | i_rb_e[`HALFWORD1];
        o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] | i_rb_e[`HALFWORD2];
        o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] | i_rb_e[`HALFWORD3];
        o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] | i_rb_e[`HALFWORD4];
        o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] | i_rb_e[`HALFWORD5];
        o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] | i_rb_e[`HALFWORD6];
        o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] | i_rb_e[`HALFWORD7];
    end
    `EXE_ORI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] | i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] | i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] | i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] | i_rb_e[`WORD3];
    end
    `EXE_XOR: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] ^ i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] ^ i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] ^ i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] ^ i_rb_e[`WORD3];
    end
    `EXE_XORBI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] ^ i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] ^ i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] ^ i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] ^ i_rb_e[`WORD3];
    end
    `EXE_XORHI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
```

```
        //o_even <= `PIPE_EVEN;
        o_rt_e[`HALFWORD0] <= i_ra_e[`HALFWORD0] ^ i_rb_e[`HALFWORD0];
        o_rt_e[`HALFWORD1] <= i_ra_e[`HALFWORD1] ^ i_rb_e[`HALFWORD1];
        o_rt_e[`HALFWORD2] <= i_ra_e[`HALFWORD2] ^ i_rb_e[`HALFWORD2];
        o_rt_e[`HALFWORD3] <= i_ra_e[`HALFWORD3] ^ i_rb_e[`HALFWORD3];
        o_rt_e[`HALFWORD4] <= i_ra_e[`HALFWORD4] ^ i_rb_e[`HALFWORD4];
        o_rt_e[`HALFWORD5] <= i_ra_e[`HALFWORD5] ^ i_rb_e[`HALFWORD5];
        o_rt_e[`HALFWORD6] <= i_ra_e[`HALFWORD6] ^ i_rb_e[`HALFWORD6];
        o_rt_e[`HALFWORD7] <= i_ra_e[`HALFWORD7] ^ i_rb_e[`HALFWORD7];
    end
    `EXE_XORI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= i_ra_e[`WORD0] ^ i_rb_e[`WORD0];
        o_rt_e[`WORD1] <= i_ra_e[`WORD1] ^ i_rb_e[`WORD1];
        o_rt_e[`WORD2] <= i_ra_e[`WORD2] ^ i_rb_e[`WORD2];
        o_rt_e[`WORD3] <= i_ra_e[`WORD3] ^ i_rb_e[`WORD3];
    end
    `EXE_NAND: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= ~(i_ra_e[`WORD0] & i_rb_e[`WORD0]);
        o_rt_e[`WORD1] <= ~(i_ra_e[`WORD1] & i_rb_e[`WORD1]);
        o_rt_e[`WORD2] <= ~(i_ra_e[`WORD2] & i_rb_e[`WORD2]);
        o_rt_e[`WORD3] <= ~(i_ra_e[`WORD3] & i_rb_e[`WORD3]);
    end
    `EXE_NOR: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
        o_rt_e[`WORD0] <= ~(i_ra_e[`WORD0] | i_rb_e[`WORD0]);
        o_rt_e[`WORD1] <= ~(i_ra_e[`WORD1] | i_rb_e[`WORD1]);
        o_rt_e[`WORD2] <= ~(i_ra_e[`WORD2] | i_rb_e[`WORD2]);
        o_rt_e[`WORD3] <= ~(i_ra_e[`WORD3] | i_rb_e[`WORD3]);
    end
    `EXE_CEQB: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
```

```
//o_even <= `PIPE_EVEN;

ra_temp_byte[0] <= i_ra_e[`BYTE0];
ra_temp_byte[1] <= i_ra_e[`BYTE1];
ra_temp_byte[2] <= i_ra_e[`BYTE2];
ra_temp_byte[3] <= i_ra_e[`BYTE3];
ra_temp_byte[4] <= i_ra_e[`BYTE4];
ra_temp_byte[5] <= i_ra_e[`BYTE5];
ra_temp_byte[6] <= i_ra_e[`BYTE6];
ra_temp_byte[7] <= i_ra_e[`BYTE7];
ra_temp_byte[8] <= i_ra_e[`BYTE8];
ra_temp_byte[9] <= i_ra_e[`BYTE9];
ra_temp_byte[10] <= i_ra_e[`BYTE10];
ra_temp_byte[11] <= i_ra_e[`BYTE11];
ra_temp_byte[12] <= i_ra_e[`BYTE12];
ra_temp_byte[13] <= i_ra_e[`BYTE13];
ra_temp_byte[14] <= i_ra_e[`BYTE14];
ra_temp_byte[15] <= i_ra_e[`BYTE15];

rb_temp_byte[0] <= i_rb_e[`BYTE0];
rb_temp_byte[1] <= i_rb_e[`BYTE1];
rb_temp_byte[2] <= i_rb_e[`BYTE2];
rb_temp_byte[3] <= i_rb_e[`BYTE3];
rb_temp_byte[4] <= i_rb_e[`BYTE4];
rb_temp_byte[5] <= i_rb_e[`BYTE5];
rb_temp_byte[6] <= i_rb_e[`BYTE6];
rb_temp_byte[7] <= i_rb_e[`BYTE7];
rb_temp_byte[8] <= i_rb_e[`BYTE8];
rb_temp_byte[9] <= i_rb_e[`BYTE9];
rb_temp_byte[10] <= i_rb_e[`BYTE10];
rb_temp_byte[11] <= i_rb_e[`BYTE11];
rb_temp_byte[12] <= i_rb_e[`BYTE12];
rb_temp_byte[13] <= i_rb_e[`BYTE13];
rb_temp_byte[14] <= i_rb_e[`BYTE14];
rb_temp_byte[15] <= i_rb_e[`BYTE15];

for (i=0; i<16; i=i+1) begin
      if (ra_temp_byte[i] == rb_temp_byte[i]) begin
            rt_temp_byte[i] = 8'hff;
      end else begin
            rt_temp_byte[i] = 8'h00;
      end
end
```

```
        o_rt_e[`BYTE0] <= rt_temp_byte[0];
        o_rt_e[`BYTE1] <= rt_temp_byte[1];
        o_rt_e[`BYTE2] <= rt_temp_byte[2];
        o_rt_e[`BYTE3] <= rt_temp_byte[3];
        o_rt_e[`BYTE4] <= rt_temp_byte[4];
        o_rt_e[`BYTE5] <= rt_temp_byte[5];
        o_rt_e[`BYTE6] <= rt_temp_byte[6];
        o_rt_e[`BYTE7] <= rt_temp_byte[7];
        o_rt_e[`BYTE8] <= rt_temp_byte[8];
        o_rt_e[`BYTE9] <= rt_temp_byte[9];
        o_rt_e[`BYTE10] <= rt_temp_byte[10];
        o_rt_e[`BYTE11] <= rt_temp_byte[11];
        o_rt_e[`BYTE12] <= rt_temp_byte[12];
        o_rt_e[`BYTE13] <= rt_temp_byte[13];
        o_rt_e[`BYTE14] <= rt_temp_byte[14];
        o_rt_e[`BYTE15] <= rt_temp_byte[15];
    end
    `EXE_CEQBI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_byte[0] <= i_ra_e[`BYTE0];
        ra_temp_byte[1] <= i_ra_e[`BYTE1];
        ra_temp_byte[2] <= i_ra_e[`BYTE2];
        ra_temp_byte[3] <= i_ra_e[`BYTE3];
        ra_temp_byte[4] <= i_ra_e[`BYTE4];
        ra_temp_byte[5] <= i_ra_e[`BYTE5];
        ra_temp_byte[6] <= i_ra_e[`BYTE6];
        ra_temp_byte[7] <= i_ra_e[`BYTE7];
        ra_temp_byte[8] <= i_ra_e[`BYTE8];
        ra_temp_byte[9] <= i_ra_e[`BYTE9];
        ra_temp_byte[10] <= i_ra_e[`BYTE10];
        ra_temp_byte[11] <= i_ra_e[`BYTE11];
        ra_temp_byte[12] <= i_ra_e[`BYTE12];
        ra_temp_byte[13] <= i_ra_e[`BYTE13];
        ra_temp_byte[14] <= i_ra_e[`BYTE14];
        ra_temp_byte[15] <= i_ra_e[`BYTE15];

        rb_temp_byte[0] <= i_ra_e[`BYTE0]; //rb=I10[2:9]

        for (i=0; i<16; i=i+1) begin
            if (ra_temp_byte[i] == rb_temp_byte[0]) begin
```

```
                rt_temp_byte[i] = 8'hff;
            end else begin
                rt_temp_byte[i] = 8'h00;
            end
        end

        o_rt_e[`BYTE0] <= rt_temp_byte[0];
        o_rt_e[`BYTE1] <= rt_temp_byte[1];
        o_rt_e[`BYTE2] <= rt_temp_byte[2];
        o_rt_e[`BYTE3] <= rt_temp_byte[3];
        o_rt_e[`BYTE4] <= rt_temp_byte[4];
        o_rt_e[`BYTE5] <= rt_temp_byte[5];
        o_rt_e[`BYTE6] <= rt_temp_byte[6];
        o_rt_e[`BYTE7] <= rt_temp_byte[7];
        o_rt_e[`BYTE8] <= rt_temp_byte[8];
        o_rt_e[`BYTE9] <= rt_temp_byte[9];
        o_rt_e[`BYTE10] <= rt_temp_byte[10];
        o_rt_e[`BYTE11] <= rt_temp_byte[11];
        o_rt_e[`BYTE12] <= rt_temp_byte[12];
        o_rt_e[`BYTE13] <= rt_temp_byte[13];
        o_rt_e[`BYTE14] <= rt_temp_byte[14];
        o_rt_e[`BYTE15] <= rt_temp_byte[15];
    end
    `EXE_CEQH: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_hword[0] <= i_ra_e[`HALFWORD0];
        ra_temp_hword[1] <= i_ra_e[`HALFWORD1];
        ra_temp_hword[2] <= i_ra_e[`HALFWORD2];
        ra_temp_hword[3] <= i_ra_e[`HALFWORD3];
        ra_temp_hword[4] <= i_ra_e[`HALFWORD4];
        ra_temp_hword[5] <= i_ra_e[`HALFWORD5];
        ra_temp_hword[6] <= i_ra_e[`HALFWORD6];
        ra_temp_hword[7] <= i_ra_e[`HALFWORD7];

        rb_temp_hword[0] <= i_rb_e[`HALFWORD0];
        rb_temp_hword[1] <= i_rb_e[`HALFWORD1];
        rb_temp_hword[2] <= i_rb_e[`HALFWORD2];
        rb_temp_hword[3] <= i_rb_e[`HALFWORD3];
        rb_temp_hword[4] <= i_rb_e[`HALFWORD4];
        rb_temp_hword[5] <= i_rb_e[`HALFWORD5];
```

```
        rb_temp_hword[6] <= i_rb_e[`HALFWORD6];
        rb_temp_hword[7] <= i_rb_e[`HALFWORD7];

        for (i=0; i<8; i=i+1) begin
            if (ra_temp_hword[i] == rb_temp_hword[i]) begin
                rt_temp_hword[i] = 16'hffff;
            end else begin
                rt_temp_hword[i] = 16'h0000;
            end
        end

        o_rt_e[`HALFWORD0] <= rt_temp_hword[0];
        o_rt_e[`HALFWORD1] <= rt_temp_hword[1];
        o_rt_e[`HALFWORD2] <= rt_temp_hword[2];
        o_rt_e[`HALFWORD3] <= rt_temp_hword[3];
        o_rt_e[`HALFWORD4] <= rt_temp_hword[4];
        o_rt_e[`HALFWORD5] <= rt_temp_hword[5];
        o_rt_e[`HALFWORD6] <= rt_temp_hword[6];
        o_rt_e[`HALFWORD7] <= rt_temp_hword[7];
    end
    `EXE_CEQHI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_hword[0] <= i_ra_e[`HALFWORD0];
        ra_temp_hword[1] <= i_ra_e[`HALFWORD1];
        ra_temp_hword[2] <= i_ra_e[`HALFWORD2];
        ra_temp_hword[3] <= i_ra_e[`HALFWORD3];
        ra_temp_hword[4] <= i_ra_e[`HALFWORD4];
        ra_temp_hword[5] <= i_ra_e[`HALFWORD5];
        ra_temp_hword[6] <= i_ra_e[`HALFWORD6];
        ra_temp_hword[7] <= i_ra_e[`HALFWORD7];

        rb_temp_hword[0] <= i_rb_e[`HALFWORD0]; //the 16bit Imm #

        for (i=0; i<8; i=i+1) begin
            if (ra_temp_hword[i] == rb_temp_hword[0]) begin
                rt_temp_hword[i] = 16'hffff;
            end else begin
                rt_temp_hword[i] = 16'h0000;
            end
        end
```

```
        o_rt_e[`HALFWORD0] <= rt_temp_hword[0];
        o_rt_e[`HALFWORD1] <= rt_temp_hword[1];
        o_rt_e[`HALFWORD2] <= rt_temp_hword[2];
        o_rt_e[`HALFWORD3] <= rt_temp_hword[3];
        o_rt_e[`HALFWORD4] <= rt_temp_hword[4];
        o_rt_e[`HALFWORD5] <= rt_temp_hword[5];
        o_rt_e[`HALFWORD6] <= rt_temp_hword[6];
        o_rt_e[`HALFWORD7] <= rt_temp_hword[7];
    end
    `EXE_CEQ: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
        ra_temp_word[3] <= i_ra_e[`WORD3];

        rb_temp_word[0] <= i_rb_e[`WORD0];
        rb_temp_word[1] <= i_rb_e[`WORD1];
        rb_temp_word[2] <= i_rb_e[`WORD2];
        rb_temp_word[3] <= i_rb_e[`WORD3];

        for (i=0; i<4; i=i+1) begin
            if (ra_temp_word[i] == rb_temp_word[i]) begin
                rt_temp_word[i] = 32'hffffffff;
            end else begin
                rt_temp_word[i] = 32'h00000000;
            end
        end

        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];
    end
    `EXE_CEQI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;
```

```
            ra_temp_word[0] <= i_ra_e[`WORD0];
            ra_temp_word[1] <= i_ra_e[`WORD1];
            ra_temp_word[2] <= i_ra_e[`WORD2];
            ra_temp_word[3] <= i_ra_e[`WORD3];

            rb_temp_word[0] <= i_rb_e[`WORD0];

            for (i=0; i<4; i=i+1) begin
                  if (ra_temp_word[i] == rb_temp_word[0]) begin
                        rt_temp_word[i] = 32'hffffffff;
                  end else begin
                        rt_temp_word[i] = 32'h00000000;
                  end
            end

            o_rt_e[`WORD0] <= rt_temp_word[0];
            o_rt_e[`WORD1] <= rt_temp_word[1];
            o_rt_e[`WORD2] <= rt_temp_word[2];
            o_rt_e[`WORD3] <= rt_temp_word[3];
      end
      `EXE_CGTB: begin
            o_wreg_e <= i_wreg_e;
            o_rtaddr_e <= i_rtaddr_e;
            o_uid_e <= i_uid_e;
            //o_even <= `PIPE_EVEN;

            ra_temp_byte[0] <= i_ra_e[`BYTE0];
            ra_temp_byte[1] <= i_ra_e[`BYTE1];
            ra_temp_byte[2] <= i_ra_e[`BYTE2];
            ra_temp_byte[3] <= i_ra_e[`BYTE3];
            ra_temp_byte[4] <= i_ra_e[`BYTE4];
            ra_temp_byte[5] <= i_ra_e[`BYTE5];
            ra_temp_byte[6] <= i_ra_e[`BYTE6];
            ra_temp_byte[7] <= i_ra_e[`BYTE7];
            ra_temp_byte[8] <= i_ra_e[`BYTE8];
            ra_temp_byte[9] <= i_ra_e[`BYTE9];
            ra_temp_byte[10] <= i_ra_e[`BYTE10];
            ra_temp_byte[11] <= i_ra_e[`BYTE11];
            ra_temp_byte[12] <= i_ra_e[`BYTE12];
            ra_temp_byte[13] <= i_ra_e[`BYTE13];
            ra_temp_byte[14] <= i_ra_e[`BYTE14];
            ra_temp_byte[15] <= i_ra_e[`BYTE15];
```

```
        rb_temp_byte[0] <= i_rb_e[`BYTE0];
        rb_temp_byte[1] <= i_rb_e[`BYTE1];
        rb_temp_byte[2] <= i_rb_e[`BYTE2];
        rb_temp_byte[3] <= i_rb_e[`BYTE3];
        rb_temp_byte[4] <= i_rb_e[`BYTE4];
        rb_temp_byte[5] <= i_rb_e[`BYTE5];
        rb_temp_byte[6] <= i_rb_e[`BYTE6];
        rb_temp_byte[7] <= i_rb_e[`BYTE7];
        rb_temp_byte[8] <= i_rb_e[`BYTE8];
        rb_temp_byte[9] <= i_rb_e[`BYTE9];
        rb_temp_byte[10] <= i_rb_e[`BYTE10];
        rb_temp_byte[11] <= i_rb_e[`BYTE11];
        rb_temp_byte[12] <= i_rb_e[`BYTE12];
        rb_temp_byte[13] <= i_rb_e[`BYTE13];
        rb_temp_byte[14] <= i_rb_e[`BYTE14];
        rb_temp_byte[15] <= i_rb_e[`BYTE15];

        for (i=0; i<16; i=i+1) begin
            if (ra_temp_byte[i] > rb_temp_byte[i]) begin
                rt_temp_byte[i] = 8'hff;
            end else begin
                rt_temp_byte[i] = 8'h00;
            end
        end

        o_rt_e[`BYTE0] <= rt_temp_byte[0];
        o_rt_e[`BYTE1] <= rt_temp_byte[1];
        o_rt_e[`BYTE2] <= rt_temp_byte[2];
        o_rt_e[`BYTE3] <= rt_temp_byte[3];
        o_rt_e[`BYTE4] <= rt_temp_byte[4];
        o_rt_e[`BYTE5] <= rt_temp_byte[5];
        o_rt_e[`BYTE6] <= rt_temp_byte[6];
        o_rt_e[`BYTE7] <= rt_temp_byte[7];
        o_rt_e[`BYTE8] <= rt_temp_byte[8];
        o_rt_e[`BYTE9] <= rt_temp_byte[9];
        o_rt_e[`BYTE10] <= rt_temp_byte[10];
        o_rt_e[`BYTE11] <= rt_temp_byte[11];
        o_rt_e[`BYTE12] <= rt_temp_byte[12];
        o_rt_e[`BYTE13] <= rt_temp_byte[13];
        o_rt_e[`BYTE14] <= rt_temp_byte[14];
        o_rt_e[`BYTE15] <= rt_temp_byte[15];
    end
    `EXE_CGTBI: begin
        o_wreg_e <= i_wreg_e;
```

```
o_rtaddr_e <= i_rtaddr_e;
o_uid_e <= i_uid_e;
//o_even <= `PIPE_EVEN;

ra_temp_byte[0] <= i_ra_e[`BYTE0];
ra_temp_byte[1] <= i_ra_e[`BYTE1];
ra_temp_byte[2] <= i_ra_e[`BYTE2];
ra_temp_byte[3] <= i_ra_e[`BYTE3];
ra_temp_byte[4] <= i_ra_e[`BYTE4];
ra_temp_byte[5] <= i_ra_e[`BYTE5];
ra_temp_byte[6] <= i_ra_e[`BYTE6];
ra_temp_byte[7] <= i_ra_e[`BYTE7];
ra_temp_byte[8] <= i_ra_e[`BYTE8];
ra_temp_byte[9] <= i_ra_e[`BYTE9];
ra_temp_byte[10] <= i_ra_e[`BYTE10];
ra_temp_byte[11] <= i_ra_e[`BYTE11];
ra_temp_byte[12] <= i_ra_e[`BYTE12];
ra_temp_byte[13] <= i_ra_e[`BYTE13];
ra_temp_byte[14] <= i_ra_e[`BYTE14];
ra_temp_byte[15] <= i_ra_e[`BYTE15];

rb_temp_byte[0] <= i_ra_e[`BYTE0]; //rb=I10[2:9]

for (i=0; i<16; i=i+1) begin
	if (ra_temp_byte[i] > rb_temp_byte[0]) begin
		rt_temp_byte[i] = 8'hff;
	end else begin
		rt_temp_byte[i] = 8'h00;
	end
end

o_rt_e[`BYTE0] <= rt_temp_byte[0];
o_rt_e[`BYTE1] <= rt_temp_byte[1];
o_rt_e[`BYTE2] <= rt_temp_byte[2];
o_rt_e[`BYTE3] <= rt_temp_byte[3];
o_rt_e[`BYTE4] <= rt_temp_byte[4];
o_rt_e[`BYTE5] <= rt_temp_byte[5];
o_rt_e[`BYTE6] <= rt_temp_byte[6];
o_rt_e[`BYTE7] <= rt_temp_byte[7];
o_rt_e[`BYTE8] <= rt_temp_byte[8];
o_rt_e[`BYTE9] <= rt_temp_byte[9];
o_rt_e[`BYTE10] <= rt_temp_byte[10];
o_rt_e[`BYTE11] <= rt_temp_byte[11];
o_rt_e[`BYTE12] <= rt_temp_byte[12];
```

```
        o_rt_e[`BYTE13] <= rt_temp_byte[13];
        o_rt_e[`BYTE14] <= rt_temp_byte[14];
        o_rt_e[`BYTE15] <= rt_temp_byte[15];
    end
    `EXE_CGTH: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_hword[0] <= i_ra_e[`HALFWORD0];
        ra_temp_hword[1] <= i_ra_e[`HALFWORD1];
        ra_temp_hword[2] <= i_ra_e[`HALFWORD2];
        ra_temp_hword[3] <= i_ra_e[`HALFWORD3];
        ra_temp_hword[4] <= i_ra_e[`HALFWORD4];
        ra_temp_hword[5] <= i_ra_e[`HALFWORD5];
        ra_temp_hword[6] <= i_ra_e[`HALFWORD6];
        ra_temp_hword[7] <= i_ra_e[`HALFWORD7];

        rb_temp_hword[0] <= i_rb_e[`HALFWORD0];
        rb_temp_hword[1] <= i_rb_e[`HALFWORD1];
        rb_temp_hword[2] <= i_rb_e[`HALFWORD2];
        rb_temp_hword[3] <= i_rb_e[`HALFWORD3];
        rb_temp_hword[4] <= i_rb_e[`HALFWORD4];
        rb_temp_hword[5] <= i_rb_e[`HALFWORD5];
        rb_temp_hword[6] <= i_rb_e[`HALFWORD6];
        rb_temp_hword[7] <= i_rb_e[`HALFWORD7];

        for (i=0; i<8; i=i+1) begin
            if (ra_temp_hword[i] > rb_temp_hword[i]) begin
                rt_temp_hword[i] = 16'hffff;
            end else begin
                rt_temp_hword[i] = 16'h0000;
            end
        end

        o_rt_e[`HALFWORD0] <= rt_temp_hword[0];
        o_rt_e[`HALFWORD1] <= rt_temp_hword[1];
        o_rt_e[`HALFWORD2] <= rt_temp_hword[2];
        o_rt_e[`HALFWORD3] <= rt_temp_hword[3];
        o_rt_e[`HALFWORD4] <= rt_temp_hword[4];
        o_rt_e[`HALFWORD5] <= rt_temp_hword[5];
        o_rt_e[`HALFWORD6] <= rt_temp_hword[6];
        o_rt_e[`HALFWORD7] <= rt_temp_hword[7];
```

```
end
`EXE_CGTHI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_hword[0] <= i_ra_e[`HALFWORD0];
        ra_temp_hword[1] <= i_ra_e[`HALFWORD1];
        ra_temp_hword[2] <= i_ra_e[`HALFWORD2];
        ra_temp_hword[3] <= i_ra_e[`HALFWORD3];
        ra_temp_hword[4] <= i_ra_e[`HALFWORD4];
        ra_temp_hword[5] <= i_ra_e[`HALFWORD5];
        ra_temp_hword[6] <= i_ra_e[`HALFWORD6];
        ra_temp_hword[7] <= i_ra_e[`HALFWORD7];

        rb_temp_hword[0] <= i_rb_e[`HALFWORD0]; //the 16bit Imm #

        for (i=0; i<8; i=i+1) begin
            if (ra_temp_hword[i] > rb_temp_hword[0]) begin
                rt_temp_hword[i] = 16'hffff;
            end else begin
                rt_temp_hword[i] = 16'h0000;
            end
        end

        o_rt_e[`HALFWORD0] <= rt_temp_hword[0];
        o_rt_e[`HALFWORD1] <= rt_temp_hword[1];
        o_rt_e[`HALFWORD2] <= rt_temp_hword[2];
        o_rt_e[`HALFWORD3] <= rt_temp_hword[3];
        o_rt_e[`HALFWORD4] <= rt_temp_hword[4];
        o_rt_e[`HALFWORD5] <= rt_temp_hword[5];
        o_rt_e[`HALFWORD6] <= rt_temp_hword[6];
        o_rt_e[`HALFWORD7] <= rt_temp_hword[7];
end
`EXE_CGT: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
```

```
        ra_temp_word[3] <= i_ra_e[`WORD3];

        rb_temp_word[0] <= i_rb_e[`WORD0];
        rb_temp_word[1] <= i_rb_e[`WORD1];
        rb_temp_word[2] <= i_rb_e[`WORD2];
        rb_temp_word[3] <= i_rb_e[`WORD3];

        for (i=0; i<4; i=i+1) begin
            if (ra_temp_word[i] > rb_temp_word[i]) begin
                rt_temp_word[i] = 32'hffffffff;
            end else begin
                rt_temp_word[i] = 32'h00000000;
            end
        end

        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];
    end
    `EXE_CGTI: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
        ra_temp_word[3] <= i_ra_e[`WORD3];

        rb_temp_word[0] <= i_rb_e[`WORD0];

        for (i=0; i<4; i=i+1) begin
            if (ra_temp_word[i] > rb_temp_word[0]) begin
                rt_temp_word[i] = 32'hffffffff;
            end else begin
                rt_temp_word[i] = 32'h00000000;
            end
        end

        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
```

```
                    o_rt_e[`WORD3] <= rt_temp_word[3];
                end
                `EXE_FA: begin
                    o_wreg_e <= i_wreg_e;
                    o_rtaddr_e <= i_rtaddr_e;
                    o_uid_e <= i_uid_e;
                    //o_even <= `PIPE_EVEN;
//                  o_rt_e[`WORD0] <= i_ra_e[`WORD0] + i_rb_e[`WORD0];
//                  o_rt_e[`WORD1] <= i_ra_e[`WORD1] + i_rb_e[`WORD1];
//                  o_rt_e[`WORD2] <= i_ra_e[`WORD2] + i_rb_e[`WORD2];
//                  o_rt_e[`WORD3] <= i_ra_e[`WORD3] + i_rb_e[`WORD3];
                    ra_temp_word[0] <= i_ra_e[`WORD0];
                    ra_temp_word[1] <= i_ra_e[`WORD1];
                    ra_temp_word[2] <= i_ra_e[`WORD2];
                    ra_temp_word[3] <= i_ra_e[`WORD3];
                    rb_temp_word[0] <= i_rb_e[`WORD0];
                    rb_temp_word[1] <= i_rb_e[`WORD1];
                    rb_temp_word[2] <= i_rb_e[`WORD2];
                    rb_temp_word[3] <= i_rb_e[`WORD3];
                    for (i=0;i<4;i=i+1) begin
                        if (ra_temp_word[i][0]==rb_temp_word[i][0])
                            begin
                            //judge tha max and min of two operands
                            if (ra_temp_word[i][1:9]>rb_temp_word[i][1:9]) begin
                                max=ra_temp_word[i];
                                min=rb_temp_word[i];
                            end
                            else begin
                                min=ra_temp_word[i];
                                max=rb_temp_word[i];
                            end
                            //the exponoents diff
                            float_diff=max[1:9]-min[1:9];
                            //resize the min operand
                            min[9:31]=min[9:31]>>float_diff;
                            rt_temp_word[i][9:31]=max[9:31]+min[9:31];
                            rt_temp_word[i][0:8]=max[0:8];
                            end
                        else begin
                            if (ra_temp_word[i][1:9]>rb_temp_word[i][1:9]) begin
                                max=ra_temp_word[i];
                                min=rb_temp_word[i];
                            end
                            else begin
```

```
                                min=ra_temp_word[i];
                                max=rb_temp_word[i];
                            end
                            float_diff=max[1:9]-min[1:9];
                            min[9:31]=min[9:31]>>float_diff;
                            rt_temp_word[i][9:31]=max[9:31]-min[9:31];
                            rt_temp_word[i][0:8]=max[0:8];
                            end
                        end
                    o_rt_e[`WORD0] <= rt_temp_word[0];
                    o_rt_e[`WORD1] <= rt_temp_word[1];
                    o_rt_e[`WORD2] <= rt_temp_word[2];
                    o_rt_e[`WORD3] <= rt_temp_word[3];
                end
                `EXE_FS: begin
                    o_wreg_e <= i_wreg_e;
                    o_rtaddr_e <= i_rtaddr_e;
                    o_uid_e <= i_uid_e;
                    //o_even <= `PIPE_EVEN;
                    o_rt_e[`WORD0] <= (~i_ra_e[`WORD0]) + i_rb_e[`WORD0] + 1;
                    o_rt_e[`WORD1] <= (~i_ra_e[`WORD1]) + i_rb_e[`WORD1] + 1;
                    o_rt_e[`WORD2] <= (~i_ra_e[`WORD2]) + i_rb_e[`WORD2] + 1;
                    o_rt_e[`WORD3] <= (~i_ra_e[`WORD3]) + i_rb_e[`WORD3] + 1;
                end
                `EXE_FM: begin
                    o_wreg_e<=i_wreg_e;
                    o_rtaddr_e<=i_rtaddr_e;
                    o_uid_e<=i_uid_e;
                    ra_temp_word[0] <= i_ra_e[`WORD0];
                    ra_temp_word[1] <= i_ra_e[`WORD1];
                    ra_temp_word[2] <= i_ra_e[`WORD2];
                    ra_temp_word[3] <= i_ra_e[`WORD3];
                    rb_temp_word[0] <= i_rb_e[`WORD0];
                    rb_temp_word[1] <= i_rb_e[`WORD1];
                    rb_temp_word[2] <= i_rb_e[`WORD2];
                    rb_temp_word[3] <= i_rb_e[`WORD3];
                    for (i=0;i<4;i=i+1) begin
                        rt_temp_word[i][0]<=ra_temp_word[i][0]~^rb_temp_word[0];
                        rt_temp_word[i][1:8]<=ra_temp_word[i][1:8]+rb_temp_word[i][1:8];

rt_temp_word[i][9:31]<=ra_temp_word[i][9:31]*rb_temp_word[i][9:31];
                    end
                    o_rt_e[`WORD0] <= rt_temp_word[0];
                    o_rt_e[`WORD1] <= rt_temp_word[1];
```

```
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];

    end
    `EXE_FCEQ: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
        ra_temp_word[3] <= i_ra_e[`WORD3];

        rb_temp_word[0] <= i_rb_e[`WORD0];
        rb_temp_word[1] <= i_rb_e[`WORD1];
        rb_temp_word[2] <= i_rb_e[`WORD2];
        rb_temp_word[3] <= i_rb_e[`WORD3];

        for (i=0; i<4; i=i+1) begin
            if (ra_temp_word[i] == rb_temp_word[i]) begin
                rt_temp_word[i] = 32'hffffffff;
            end else begin
                rt_temp_word[i] = 32'h00000000;
            end
        end

        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];
    end
    `EXE_FCGT: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_word[0] <= i_ra_e[`WORD0];
        ra_temp_word[1] <= i_ra_e[`WORD1];
        ra_temp_word[2] <= i_ra_e[`WORD2];
        ra_temp_word[3] <= i_ra_e[`WORD3];
```

```
        rb_temp_word[0] <= i_rb_e[`WORD0];
        rb_temp_word[1] <= i_rb_e[`WORD1];
        rb_temp_word[2] <= i_rb_e[`WORD2];
        rb_temp_word[3] <= i_rb_e[`WORD3];

        for (i=0; i<4; i=i+1) begin
            if (ra_temp_word[i] > rb_temp_word[i]) begin
                rt_temp_word[i] = 32'hffffffff;
            end else begin
                rt_temp_word[i] = 32'h00000000;
            end
        end

        o_rt_e[`WORD0] <= rt_temp_word[0];
        o_rt_e[`WORD1] <= rt_temp_word[1];
        o_rt_e[`WORD2] <= rt_temp_word[2];
        o_rt_e[`WORD3] <= rt_temp_word[3];
    end
    `EXE_CNTB: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_byte[0] <= i_ra_e[`BYTE0];
        ra_temp_byte[1] <= i_ra_e[`BYTE1];
        ra_temp_byte[2] <= i_ra_e[`BYTE2];
        ra_temp_byte[3] <= i_ra_e[`BYTE3];
        ra_temp_byte[4] <= i_ra_e[`BYTE4];
        ra_temp_byte[5] <= i_ra_e[`BYTE5];
        ra_temp_byte[6] <= i_ra_e[`BYTE6];
        ra_temp_byte[7] <= i_ra_e[`BYTE7];
        ra_temp_byte[8] <= i_ra_e[`BYTE8];
        ra_temp_byte[9] <= i_ra_e[`BYTE9];
        ra_temp_byte[10] <= i_ra_e[`BYTE10];
        ra_temp_byte[11] <= i_ra_e[`BYTE11];
        ra_temp_byte[12] <= i_ra_e[`BYTE12];
        ra_temp_byte[13] <= i_ra_e[`BYTE13];
        ra_temp_byte[14] <= i_ra_e[`BYTE14];
        ra_temp_byte[15] <= i_ra_e[`BYTE15];

            for (i=0; i<16; i=i+1) begin
                c = 0;
                b = ra_temp_byte[i];
```

```
                    for (m=0; m<8; m=m+1) begin
                        if (b[m] == 1) begin
                                c = c + 1;
                    end
                    end
                    rt_temp_byte[i] = c;
                end
        o_rt_e[`BYTE0] <= rt_temp_byte[0];
        o_rt_e[`BYTE1] <= rt_temp_byte[1];
        o_rt_e[`BYTE2] <= rt_temp_byte[2];
        o_rt_e[`BYTE3] <= rt_temp_byte[3];
        o_rt_e[`BYTE4] <= rt_temp_byte[4];
        o_rt_e[`BYTE5] <= rt_temp_byte[5];
        o_rt_e[`BYTE6] <= rt_temp_byte[6];
        o_rt_e[`BYTE7] <= rt_temp_byte[7];
        o_rt_e[`BYTE8] <= rt_temp_byte[8];
        o_rt_e[`BYTE9] <= rt_temp_byte[9];
        o_rt_e[`BYTE10] <= rt_temp_byte[10];
        o_rt_e[`BYTE11] <= rt_temp_byte[11];
        o_rt_e[`BYTE12] <= rt_temp_byte[12];
        o_rt_e[`BYTE13] <= rt_temp_byte[13];
        o_rt_e[`BYTE14] <= rt_temp_byte[14];
        o_rt_e[`BYTE15] <= rt_temp_byte[15];
end
`EXE_AVGB: begin
        o_wreg_e <= i_wreg_e;
        o_rtaddr_e <= i_rtaddr_e;
        o_uid_e <= i_uid_e;
        //o_even <= `PIPE_EVEN;

        ra_temp_byte[0] <= i_ra_e[`BYTE0];
        ra_temp_byte[1] <= i_ra_e[`BYTE1];
        ra_temp_byte[2] <= i_ra_e[`BYTE2];
        ra_temp_byte[3] <= i_ra_e[`BYTE3];
        ra_temp_byte[4] <= i_ra_e[`BYTE4];
        ra_temp_byte[5] <= i_ra_e[`BYTE5];
        ra_temp_byte[6] <= i_ra_e[`BYTE6];
        ra_temp_byte[7] <= i_ra_e[`BYTE7];
        ra_temp_byte[8] <= i_ra_e[`BYTE8];
        ra_temp_byte[9] <= i_ra_e[`BYTE9];
        ra_temp_byte[10] <= i_ra_e[`BYTE10];
        ra_temp_byte[11] <= i_ra_e[`BYTE11];
        ra_temp_byte[12] <= i_ra_e[`BYTE12];
        ra_temp_byte[13] <= i_ra_e[`BYTE13];
```

```
            ra_temp_byte[14] <= i_ra_e[`BYTE14];
            ra_temp_byte[15] <= i_ra_e[`BYTE15];

            rb_temp_byte[0] <= i_rb_e[`BYTE0];
            rb_temp_byte[1] <= i_rb_e[`BYTE1];
            rb_temp_byte[2] <= i_rb_e[`BYTE2];
            rb_temp_byte[3] <= i_rb_e[`BYTE3];
            rb_temp_byte[4] <= i_rb_e[`BYTE4];
            rb_temp_byte[5] <= i_rb_e[`BYTE5];
            rb_temp_byte[6] <= i_rb_e[`BYTE6];
            rb_temp_byte[7] <= i_rb_e[`BYTE7];
            rb_temp_byte[8] <= i_rb_e[`BYTE8];
            rb_temp_byte[9] <= i_rb_e[`BYTE9];
            rb_temp_byte[10] <= i_rb_e[`BYTE10];
            rb_temp_byte[11] <= i_rb_e[`BYTE11];
            rb_temp_byte[12] <= i_rb_e[`BYTE12];
            rb_temp_byte[13] <= i_rb_e[`BYTE13];
            rb_temp_byte[14] <= i_rb_e[`BYTE14];
            rb_temp_byte[15] <= i_rb_e[`BYTE15];

                for (i=0; i<16; i=i+1) begin
                    s = ({8'h00,ra_temp_byte[i]} + {8'h00,rb_temp_byte[i]}) + 1;
                    rt_temp_byte[i] = s[7:14];
                end
            o_rt_e[`BYTE0] <= rt_temp_byte[0];
            o_rt_e[`BYTE1] <= rt_temp_byte[1];
            o_rt_e[`BYTE2] <= rt_temp_byte[2];
            o_rt_e[`BYTE3] <= rt_temp_byte[3];
            o_rt_e[`BYTE4] <= rt_temp_byte[4];
            o_rt_e[`BYTE5] <= rt_temp_byte[5];
            o_rt_e[`BYTE6] <= rt_temp_byte[6];
            o_rt_e[`BYTE7] <= rt_temp_byte[7];
            o_rt_e[`BYTE8] <= rt_temp_byte[8];
            o_rt_e[`BYTE9] <= rt_temp_byte[9];
            o_rt_e[`BYTE10] <= rt_temp_byte[10];
            o_rt_e[`BYTE11] <= rt_temp_byte[11];
            o_rt_e[`BYTE12] <= rt_temp_byte[12];
            o_rt_e[`BYTE13] <= rt_temp_byte[13];
            o_rt_e[`BYTE14] <= rt_temp_byte[14];
            o_rt_e[`BYTE15] <= rt_temp_byte[15];
        end
        `EXE_NOP: begin
            o_rt_e <= 0;
            o_rtaddr_e <= 0;
```

```
                        o_wreg_e <= 0;
                        o_uid_e <= 0;
                    end
                    default:begin
                        o_rt_e <= `ZERO_QWORD128;
                    end
                endcase
            end
        end
endmodule

//Odd pipe
module EX_ODD(

    input wire clk,
    input wire rst,

    input wire[0:10] i_opcode_o,
    input wire[0:127] i_ra_o,
    input wire[0:127] i_rb_o,
    input wire[0:6] i_rtaddr_o,
    input wire i_wreg_o,
    input wire[0:2] i_uid_o,
    input wire[0:9] i_imm_o,
    //input wire i_odd,

    input reg[0:31] i_pc,
    input reg[0:31] i_ex_link_addr,
    input reg i_ex_id_is_in_delayslot,
    input reg[0:31] i_ex_branch_target_addr,
    input reg i_ex_branch_flag,

    output reg[0:127] o_rt_o,
    output reg[0:6] o_rtaddr_o,
    output reg o_wreg_o,
    output reg[0:2] o_uid_o,
    //memory address
    output reg[0:31] o_memory_addr_o,
    //output reg o_odd,

    output reg[0:31] ex_pc,
    output reg[0:31] o_ex_link_addr,
    output reg o_ex_is_in_delayslot,
    output reg[0:31] o_ex_branch_target_addr,
```

```
    output reg o_ex_branch_flag

);

    reg [0:127] rt_temp_b;
    reg [0:1] sh;
    reg [0:7] imm_8b;
    reg [0:31] imm_32b;
    reg [0:31] t;
    reg [0:3] s;
    integer b;

    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            o_rt_o <= `ZERO_QWORD128;
        end else begin
            case (i_opcode_o)
                `EXE_LQD: begin
                    o_rtaddr_o<=i_rtaddr_o;
                    o_wreg_o<=i_wreg_o;
                    o_uid_o<=i_uid_o;
                    o_memory_addr_o<=i_ra_o+i_rb_o;
                end
                `EXE_STQD: begin
                    o_wreg_o<=i_wreg_o;
                    o_uid_o<=i_uid_o;
                    o_rt_o<=i_rb_o;
                    o_memory_addr_o<=i_imm_o+i_ra_o;
                end
                `EXE_SHLQBI: begin
                    o_wreg_o <= i_wreg_o;
                    o_rtaddr_o <= i_rtaddr_o;
                    o_uid_o <= i_uid_o;
                    //o_odd <= `PIPE_ODD;
                    sh = i_rb_o[29:31];
                    for (b=0; b<128; b=b+1) begin
                        if ((b+sh) < 128) begin
                            o_rt_o[b] = i_ra_o[b+sh];
                        end else begin
                            o_rt_o[b] = 0;
                        end
                    end
                end
                `EXE_SHLQBII: begin
```

```
        o_wreg_o <= i_wreg_o;
        o_rtaddr_o <= i_rtaddr_o;
        o_uid_o <= i_uid_o;
        //o_odd <= `PIPE_ODD;
        imm_8b = i_rb_o[0:7] & 8'h07;

        for (b=0; b<128; b=b+1) begin
            if ((b+imm_8b) < 128) begin
                o_rt_o[b] = i_ra_o[b+imm_8b];
            end else begin
                o_rt_o[b] = 0;
            end
        end
    end
    `EXE_ROTQBY: begin
        o_wreg_o <= i_wreg_o;
        o_rtaddr_o <= i_rtaddr_o;
        o_uid_o <= i_uid_o;
        //o_odd <= `PIPE_ODD;
        s = i_rb_o[28:31];
        for (b=0; b<128; b=b+1) begin
            if ((b+s) < 16) begin
                o_rt_o[b] = i_ra_o[b+s];
            end else begin
                o_rt_o[b] = i_ra_o[b+s-4'hf];
            end
        end
    end
    `EXE_ROTQBYI: begin
        o_wreg_o <= i_wreg_o;
        o_rtaddr_o <= i_rtaddr_o;
        o_uid_o <= i_uid_o;
        //o_odd <= `PIPE_ODD;
        //s = i_rb_o[28:31];
        s = i_rb_o[14:17]; //only mask the I7[14:17]
        for (b=0; b<128; b=b+1) begin
            if ((b+s) < 16) begin
                o_rt_o[b] = i_ra_o[b+s];
            end else begin
                o_rt_o[b] = i_ra_o[b+s-4'hf];
            end
        end
    end
    `EXE_CBD: begin
```

```
                o_wreg_o <= i_wreg_o;
                o_rtaddr_o <= i_rtaddr_o;
                o_uid_o <= i_uid_o;
                //o_odd <= `PIPE_ODD;
                imm_32b = {{22{i_rb_o[0]}},i_rb_o[0:7]};
                t = (i_ra_o[0:31] + imm_32b) & 32'h0000000F;
                case (t)
                    32'h00000000:                   o_rt_o                  =
128'h10111213141516171819a1b1c1d1e03;
                    32'h00000001:                   o_rt_o                  =
128'h10111213141516171819a1b1c1d031f;
                    32'h00000002:                   o_rt_o                  =
128'h10111213141516171819a1b1c031e1f;
                    32'h00000003:                   o_rt_o                  =
128'h10111213141516171819a1b031d1e1f;
                    32'h00000004:                   o_rt_o                  =
128'h10111213141516171819a031c1d1e1f;
                    32'h00000005:                   o_rt_o                  =
128'h10111213141516171819031b1c1d1e1f;
                    32'h00000006:                   o_rt_o                  =
128'h10111213141516171803a1b1c1d1e1f;
                    32'h00000007:                   o_rt_o                  =
128'h10111213141516170319a1b1c1d1e1f;
                    32'h00000008:                   o_rt_o                  =
128'h10111213141516031819a1b1c1d1e1f;
                    32'h00000009:                   o_rt_o                  =
128'h10111213141503171819a1b1c1d1e1f;
                    32'h0000000a:                   o_rt_o                  =
128'h10111213140316171819a1b1c1d1e1f;
                    32'h0000000b:                   o_rt_o                  =
128'h10111213031516171819a1b1c1d1e1f;
                    32'h0000000c:                   o_rt_o                  =
128'h10111203141516171819a1b1c1d1e1f;
                    32'h0000000d:                   o_rt_o                  =
128'h10110313141516171819a1b1c1d1e1f;
                    32'h0000000e:                   o_rt_o                  =
128'h10031213141516171819a1b1c1d1e1f;
                    32'h0000000f:                   o_rt_o                  =
128'h03111213141516171819a1b1c1d1e1f;
                    default: o_rt_o = 128'h10111213141516171819a1b1c1d1e1f;
                endcase
            end
            `EXE_BR:begin
                o_uid_o <= i_uid_o;
```

```
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= i_ex_branch_target_addr;
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRA:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= i_ex_branch_target_addr;
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRSL:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_rt_o  <= ((ex_pc + 4) & 128'h00000000000000000000000ffffffff);
//RT0:3 = pc+4; RT4:15=0
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= i_ex_branch_target_addr;
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRASL:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_rt_o  <= ((ex_pc + 4) & 128'h00000000000000000000000ffffffff);
//RT0:3 = pc+4; RT4:15=0
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= ex_pc + i_ex_branch_target_addr;
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BI:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= (i_ra_o[0:31] & 32'hfffffffc);
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BISL:begin
                        o_uid_o <= i_uid_o;
```

```
                        //o_odd <= `PIPE_ODD;
                        o_rt_o <= ((ex_pc + 4) & 128'h00000000000000000000000ffffffff);
//RT0:3 = pc+4; RT4:15=0
                        o_ex_branch_flag <= i_ex_branch_flag;
                        o_ex_branch_target_addr <= (i_ra_o[0:31] & 32'hfffffffc);
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRNZ:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        if (o_rt_o[0:31] != 0) begin
                            o_ex_branch_target_addr <= ((ex_pc + i_ex_branch_target_addr) &
32'hfffffffc);
                        end else begin
                            o_ex_branch_target_addr <= ex_pc + 4;
                        end
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRZ:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        if (o_rt_o[0:31] == 0) begin
                            o_ex_branch_target_addr <= ((ex_pc + i_ex_branch_target_addr) &
32'hfffffffc);
                        end else begin
                            o_ex_branch_target_addr <= ex_pc +4;
                        end
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRHNZ:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        if (o_rt_o[16:31] != 0) begin
                            o_ex_branch_target_addr <= ((ex_pc + i_ex_branch_target_addr) &
32'hfffffffc);
                        end else begin
                            o_ex_branch_target_addr <= ex_pc +4;
                        end
```

```
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_BRHZ:begin
                        o_uid_o <= i_uid_o;
                        //o_odd <= `PIPE_ODD;
                        o_ex_branch_flag <= i_ex_branch_flag;
                        if (o_rt_o[16:31] == 0) begin
                            o_ex_branch_target_addr <= ((ex_pc + i_ex_branch_target_addr) &
32'hfffffffc);
                        end else begin
                            o_ex_branch_target_addr <= ex_pc +4;
                        end
                        o_ex_link_addr <= i_ex_link_addr;
                        o_ex_is_in_delayslot <= i_ex_id_is_in_delayslot;
                    end
                    `EXE_LNOP: begin
                        o_rt_o <= 0;
                        o_rtaddr_o <= 0;
                        o_wreg_o <= 0;
                        o_uid_o <= 0;
                        //ex_pc <= 0;
                        //o_ex_link_addr <= 0;
                        //o_ex_is_in_delayslot <= 0;
                        //o_ex_branch_target_addr <= 0;
                        //o_ex_branch_flag <= 0;
                    end
                    default:begin
                        o_rt_o <= `ZERO_WORD32;
                    end
                endcase
            end
        end
endmodule


//*************************************************************************************/
/
//*************************************************************************************/
/
//******************** Only back-up: ORI test code ****************************//
//*************************************************************************************/
/
//*************************************************************************************/
```

```
/
/*
module EX(

	input wire clk,
	input wire rst,

	input wire[0:10] i_opcode_e,
	input wire[0:127] i_ra_e,
	input wire[0:127] i_rb_e,
	input wire[0:6] i_rtaddr_e,
	input wire i_wreg_e,
	input wire[0:2] i_uid_e,
	input wire i_even,

	input wire[0:10] i_opcode_o,
	input wire[0:127] i_ra_o,
	input wire[0:127] i_rb_o,
	input wire[0:6] i_rtaddr_o,
	input wire i_wreg_o,
	input wire[0:2] i_uid_o,
	input wire i_odd,

	output reg[0:127] o_rt_e,
	output reg[0:6] o_rtaddr_e,
	output reg o_wreg_e,
	output reg[0:2] o_uid_e,
	output reg o_even,

	output reg[0:127] o_rt_o,
	output reg[0:6] o_rtaddr_o,
	output reg o_wreg_o,
	output reg[0:2] o_uid_o,
	output reg o_odd
);

	always @ (*) begin
		if(rst == `RST_ENABLE) begin
			o_rt_e <= `ZERO_WORD32;
		end else begin
			case (i_opcode_e)
				`EXE_ORI:begin
					o_wreg_e <= i_wreg_e;
					o_rtaddr_e <= i_rtaddr_e;
```

```
                        o_rt_e <= i_ra_e | i_rb_e;
                        o_uid_e <= i_uid_e;
                        o_even <= i_even;
                    end
                    default:begin
                        o_rt_e <= `ZERO_WORD32;
                    end
                endcase
            end
        end

        always @ (*) begin
            if(rst == `RST_ENABLE) begin
                o_rt_o <= `ZERO_WORD32;
            end else begin
                case (i_opcode_o)
                    `EXE_ORI:begin
                        o_wreg_o <= i_wreg_o;
                        o_rtaddr_o <= i_rtaddr_o;
                        o_rt_o <= i_ra_o | i_rb_o;
                        o_uid_o <= i_uid_o;
                        o_odd <= i_odd;
                    end
                    default:begin
                        o_rt_o <= `ZERO_WORD32;
                    end
                endcase
            end
        end

endmodule
*/


//*************************************************************************************
*********/
// Module:      EX_FF
// File:        ex_FF.v
// Description: The stage between Execution and FF
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************************
**********/

`include "defines.v"
```

```
module EX_FF(

	input wire clk,
	input wire rst,

	//info from execution stage
	input wire[`REG_BUS128] ex_rt_e,
	input wire[`REG_ADDR_BUS7] ex_rtaddr_e,
	input wire ex_wreg_e,
	input wire[0:2] ex_uid_e,

	input wire[`REG_BUS128] ex_rt_o,
	input wire[`REG_ADDR_BUS7] ex_rtaddr_o,
	input wire ex_wreg_o,
	input wire[0:2] ex_uid_o,
	//memory address
	input wire[0:31] ex_memory_addr_o,

	input reg[0:31] ex_link_addr,
	input reg ex_is_in_delayslot,
	input reg[0:31] ex_branch_target_addr,
	input reg ex_branch_flag,

	//info to ff stage
	output reg[`REG_ADDR_BUS7] ff_rtaddr_e,
	output reg ff_wreg_e,
	output reg[`REG_BUS128] ff_rt_e,
	output reg[0:2] ff_uid_e,

	output reg[`REG_ADDR_BUS7] ff_rtaddr_o,
	output reg ff_wreg_o,
	output reg[`REG_BUS128] ff_rt_o,
	output reg[0:2] ff_uid_o,
	//memory address
	output reg[0:31] ff_memory_addr_o,

	output reg ff_branch_flag,
	output reg[0:31] ff_branch_target_addr,
	output reg[0:31] ff_link_addr,
	output reg ff_is_in_delayslot
);
```

```
    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            ff_rtaddr_e <= 0;
            ff_rt_e <= 0;
            ff_wreg_e <= 0;
            ff_uid_e <= 0;
            ff_rtaddr_o <= 0;
            ff_rt_o <= 0;
            ff_wreg_o <= 0;
            ff_uid_o <= 0;
            ff_branch_flag <= 0;
            ff_branch_target_addr <= 0;
            ff_link_addr <= 0;
            ff_is_in_delayslot <= 0;
        end else begin
            ff_rtaddr_e <= ex_rtaddr_e;
            ff_rt_e <= ex_rt_e;
            ff_wreg_e <= ex_wreg_e;
            ff_uid_e <= ex_uid_e;
            ff_rtaddr_o <= ex_rtaddr_o;
            ff_rt_o <= ex_rt_o;
            ff_wreg_o <= ex_wreg_o;
            ff_uid_o <= ex_uid_o;
            ff_branch_flag <= ex_branch_flag;
            ff_branch_target_addr <= ex_branch_target_addr;
            ff_link_addr <= ff_link_addr;
            ff_is_in_delayslot <= ff_is_in_delayslot;
            ff_memory_addr_o<=ex_memory_addr_o;
        end
    end

endmodule


//*************************************************************************************
*********/
// Module:      FF1
// File:        FF1.v
// Description: The stage FF1
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************************
**********/

`include "defines.v"
```

```
module FF1(

    input wire clk,
    input wire rst,

    //info from ex/ff
    input wire[`REG_ADDR_BUS7] iff_rtaddr_e,
    input wire iff_wreg_e,
    input wire[`REG_BUS128] iff_rt_e,
    input wire[0:2] iff_uid_e,

    input wire[`REG_ADDR_BUS7]      iff_rtaddr_o,
    input wire iff_wreg_o,
    input wire[`REG_BUS128] iff_rt_o,
    input wire[0:2] iff_uid_o,
    //memory address
    input wire[0:31] iff_memory_addr_o,

    input reg iff_branch_flag,
    input reg[0:31] iff_branch_target_addr,
    input reg[0:31] iff_link_addr,
    input reg iff_is_in_delayslot,

    //info to ff2 stage
    output reg[`REG_ADDR_BUS7] ff1_rtaddr_e,
    output reg ff1_wreg_e,
    output reg[`REG_BUS128] ff1_rt_e,
    output reg[0:2] ff1_uid_e,

    output reg[`REG_ADDR_BUS7] ff1_rtaddr_o,
    output reg ff1_wreg_o,
    output reg[`REG_BUS128] ff1_rt_o,
    output reg[0:2] ff1_uid_o,
    //memory address
    output reg[0:31] ff1_memory_addr_o,

    output reg ff1_branch_flag,
    output reg[0:31] ff1_branch_target_addr,
    output reg[0:31] ff1_link_addr,
    output reg ff1_is_in_delayslot

);
```

```
    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            ff1_rtaddr_e <= 0;
            ff1_rt_e <= 0;
            ff1_wreg_e <= 0;
            ff1_uid_e <= 0;
            ff1_rtaddr_o <= 0;
            ff1_rt_o <= 0;
            ff1_wreg_o <= 0;
            ff1_uid_o <= 0;
            ff1_branch_flag <= 0;
            ff1_branch_target_addr <= 0;
            ff1_link_addr <= 0;
            ff1_is_in_delayslot <= 0;
        end else begin
            ff1_rtaddr_e <= iff_rtaddr_e;
            ff1_rt_e <= iff_rt_e;
            ff1_wreg_e <= iff_wreg_e;
            ff1_uid_e <= iff_uid_e;
            ff1_rtaddr_o <= iff_rtaddr_o;
            ff1_rt_o <= iff_rt_o;
            ff1_wreg_o <= iff_wreg_o;
            ff1_uid_o <= iff_uid_o;
            ff1_memory_addr_o<=iff_memory_addr_o;
            ff1_branch_flag <= iff_branch_flag;
            ff1_branch_target_addr <= iff_branch_target_addr;
            ff1_link_addr <= iff_link_addr;
            ff1_is_in_delayslot <= iff_is_in_delayslot;
        end
    end

endmodule


//*************************************************************************************
*********/
// Module:      FF2
// File:        FF2.v
// Description: The stage FF2
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************************
**********/

`include "defines.v"
```

```
module FF2(

	input wire clk,
	input wire rst,

	//from FF1
	input wire[`REG_ADDR_BUS7] iff2_rtaddr_e,
	input wire iff2_wreg_e,
	input wire[`REG_BUS128] iff2_rt_e,
	input wire[0:2] iff2_uid_e,

	input wire[`REG_ADDR_BUS7] iff2_rtaddr_o,
	input wire iff2_wreg_o,
	input wire[`REG_BUS128] iff2_rt_o,
	input wire[0:2] iff2_uid_o,
	//memory address
	input wire[0:31] iff2_memory_addr_o,

	input reg iff2_branch_flag,
	input reg[0:31] iff2_branch_target_addr,
	input reg[0:31] iff2_link_addr,
	input reg iff2_is_in_delayslot,

	//to FF3
	output reg[`REG_ADDR_BUS7] ff2_rtaddr_e,
	output reg ff2_wreg_e,
	output reg[`REG_BUS128] ff2_rt_e,
	output reg[0:2] ff2_uid_e,

	output reg[`REG_ADDR_BUS7] ff2_rtaddr_o,
	output reg ff2_wreg_o,
	output reg[`REG_BUS128] ff2_rt_o,
	output reg[0:2] ff2_uid_o,
	//memory address
	output reg[0:31] ff2_memory_addr_o,

	output reg ff2_branch_flag,
	output reg[0:31] ff2_branch_target_addr,
	output reg[0:31] ff2_link_addr,
	output reg ff2_is_in_delayslot

);
```

```
    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            ff2_rtaddr_e <= 0;
            ff2_rt_e <= 0;
            ff2_wreg_e <= 0;
            ff2_uid_e <= 0;
            ff2_rtaddr_o <= 0;
            ff2_rt_o <= 0;
            ff2_wreg_o <= 0;
            ff2_uid_o <= 0;
            ff2_branch_flag <= 0;
            ff2_branch_target_addr <= 0;
            ff2_link_addr <= 0;
            ff2_is_in_delayslot <= 0;
        end else begin
            ff2_rtaddr_e <= iff2_rtaddr_e;
            ff2_rt_e <= iff2_rt_e;
            ff2_wreg_e <= iff2_wreg_e;
            ff2_uid_e <= iff2_uid_e;
            ff2_rtaddr_o <= iff2_rtaddr_o;
            ff2_rt_o <= iff2_rt_o;
            ff2_wreg_o <= iff2_wreg_o;
            ff2_uid_o <= iff2_uid_o;
            ff2_memory_addr_o<=iff2_memory_addr_o;
            ff2_branch_flag <= iff2_branch_flag;
            ff2_branch_target_addr <= iff2_branch_target_addr;
            ff2_link_addr <= iff2_link_addr;
            ff2_is_in_delayslot <= iff2_is_in_delayslot;
        end
    end

endmodule


//************************************************************************************
*********/
// Module:      FF3
// File:        FF3.v
// Description: The stage FF3
// History:     Created by Ning Kang, Mar 28, 2018
//************************************************************************************
**********/
```

```
`include "defines.v"

module FF3(

    input wire clk,
    input wire rst,

    //from FF2
    input wire[`REG_ADDR_BUS7] iff3_rtaddr_e,
    input wire iff3_wreg_e,
    input wire[`REG_BUS128] iff3_rt_e,
    input wire[0:2] iff3_uid_e,

    input wire[`REG_ADDR_BUS7] iff3_rtaddr_o,
    input wire iff3_wreg_o,
    input wire[`REG_BUS128] iff3_rt_o,
    input wire[0:2] iff3_uid_o,
    //memory address
    input wire[0:31] iff3_memory_addr_o,

    input reg iff3_branch_flag,
    input reg[0:31] iff3_branch_target_addr,
    input reg[0:31] iff3_link_addr,
    input reg iff3_is_in_delayslot,

    //to FF4
    output reg[`REG_ADDR_BUS7] ff3_rtaddr_e,
    output reg ff3_wreg_e,
    output reg[`REG_BUS128] ff3_rt_e,
    output reg[0:2] ff3_uid_e,

    output reg[`REG_ADDR_BUS7] ff3_rtaddr_o,
    output reg ff3_wreg_o,
    output reg[`REG_BUS128] ff3_rt_o,
    output reg[0:2] ff3_uid_o,
    //memory address
    output reg[0:31] ff3_memory_addr_o,

    output reg ff3_branch_flag,
    output reg[0:31] ff3_branch_target_addr,
    output reg[0:31] ff3_link_addr,
    output reg ff3_is_in_delayslot
);
```

```
    always @ (posedge clk) begin
        if(rst == `RST_ENABLE) begin
            ff3_rtaddr_e <= 0;
            ff3_rt_e <= 0;
            ff3_wreg_e <= 0;
            ff3_uid_e <= 0;
            ff3_rtaddr_o <= 0;
            ff3_rt_o <= 0;
            ff3_wreg_o <= 0;
            ff3_uid_o <= 0;
            ff3_branch_flag <= 0;
            ff3_branch_target_addr <= 0;
            ff3_link_addr <= 0;
            ff3_is_in_delayslot <= 0;
        end else begin
            ff3_rtaddr_e <= iff3_rtaddr_e;
            ff3_rt_e <= iff3_rt_e;
            ff3_wreg_e <= iff3_wreg_e;
            ff3_uid_e <= iff3_uid_e;
            ff3_rtaddr_o <= iff3_rtaddr_o;
            ff3_rt_o <= iff3_rt_o;
            ff3_wreg_o <= iff3_wreg_o;
            ff3_uid_o <= iff3_uid_o;
            ff3_memory_addr_o<=iff3_memory_addr_o;
            ff3_branch_flag <= iff3_branch_flag;
            ff3_branch_target_addr <= iff3_branch_target_addr;
            ff3_link_addr <= iff3_link_addr;
            ff3_is_in_delayslot <= iff3_is_in_delayslot;
        end
    end

endmodule


//************************************************************************************
*********/
// Module:      FF4
// File:        FF4.v
// Description: The stage FF4
// History:     Created by Ning Kang, Mar 28, 2018
//************************************************************************************
**********/
```

```
`include "defines.v"

module FF4(

    input wire clk,
    input wire rst,

    //from FF3
    input wire[`REG_ADDR_BUS7] iff4_rtaddr_e,
    input wire iff4_wreg_e,
    input wire[`REG_BUS128] iff4_rt_e,
    input wire[0:2] iff4_uid_e,

    input wire[`REG_ADDR_BUS7] iff4_rtaddr_o,
    input wire iff4_wreg_o,
    input wire[`REG_BUS128] iff4_rt_o,
    input wire[0:2] iff4_uid_o,
    //memory address
    input wire[0:31] iff4_memory_addr_o,

    input reg iff4_branch_flag,
    input reg[0:31] iff4_branch_target_addr,
    input reg[0:31] iff4_link_addr,
    input reg iff4_is_in_delayslot,

    //to FF5
    output reg[`REG_ADDR_BUS7] ff4_rtaddr_e,
    output reg ff4_wreg_e,
    output reg[`REG_BUS128] ff4_rt_e,
    output reg[0:2] ff4_uid_e,

    output reg[`REG_ADDR_BUS7] ff4_rtaddr_o,
    output reg ff4_wreg_o,
    output reg[`REG_BUS128] ff4_rt_o,
    output reg[0:2] ff4_uid_o,
    //memory address
    output reg[0:31] ff4_memory_addr_o,
    //PC_REG
    output reg ff4_branch_flag,
    output reg[0:31] ff4_branch_target_addr,
    output reg ff4_is_in_delayslot
);

    always @ (posedge clk) begin
```

```
            if(rst == `RST_ENABLE) begin
                ff4_rtaddr_e <= 0;
                ff4_rt_e <= 0;
                ff4_wreg_e <= 0;
                ff4_uid_e <= 0;
                ff4_rtaddr_o <= 0;
                ff4_rt_o <= 0;
                ff4_wreg_o <= 0;
                ff4_uid_o <= 0;
                ff4_branch_flag <= 0;
                ff4_branch_target_addr <= 0;
                ff4_is_in_delayslot <= 0;
            end else begin
                ff4_rtaddr_e <= iff4_rtaddr_e;
                ff4_rt_e <= iff4_rt_e;
                ff4_wreg_e <= iff4_wreg_e;
                ff4_uid_e <= iff4_uid_e;
                ff4_rtaddr_o <= iff4_rtaddr_o;
                ff4_wreg_o <= iff4_wreg_o;
                ff4_uid_o <= iff4_uid_o;
                ff4_branch_flag = iff4_branch_flag;
                ff4_branch_target_addr <= iff4_branch_target_addr;
//ff4_branch_target_addr <= 32'h100;
                ff4_is_in_delayslot <= iff4_is_in_delayslot;
                if (ff4_branch_flag == `BRANCH) begin
                    ff4_rt_o <= iff4_link_addr;
                end else begin
                    ff4_rt_o <= iff4_rt_o;
                end
                ff4_memory_addr_o<=iff4_memory_addr_o;
            end
        end

endmodule


//*************************************************************************
*********/
// Module:      FF5
// File:        FF5.v
// Description: The stage FF5
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************
**********/
```

```
`include "defines.v"

module FF5(

	input wire clk,
	input wire rst,

	//from FF4
	input wire[`REG_ADDR_BUS7] iff5_rtaddr_e,
	input wire iff5_wreg_e,
	input wire[`REG_BUS128] iff5_rt_e,
	input wire[0:2] iff5_uid_e,

	input wire[`REG_ADDR_BUS7] iff5_rtaddr_o,
	input wire iff5_wreg_o,
	input wire[`REG_BUS128] iff5_rt_o,
	input wire[0:2] iff5_uid_o,
	//memory address
	input wire[0:31] iff5_memory_addr_o,

	//to FF6
	output reg[`REG_ADDR_BUS7] ff5_rtaddr_e,
	output reg ff5_wreg_e,
	output reg[`REG_BUS128] ff5_rt_e,
	output reg[0:2] ff5_uid_e,

	output reg[`REG_ADDR_BUS7] ff5_rtaddr_o,
	output reg ff5_wreg_o,
	output reg[`REG_BUS128] ff5_rt_o,
	output reg[0:2] ff5_uid_o,
	//memory address
	output reg[0:31] ff5_memory_addr_o
);


	always @ (posedge clk) begin
		if(rst == `RST_ENABLE) begin
			ff5_rtaddr_e <= 0;
			ff5_rt_e <= 0;
			ff5_wreg_e <= 0;
			ff5_uid_e <= 0;
			ff5_rtaddr_o <= 0;
			ff5_rt_o <= 0;
```

```
                ff5_wreg_o <= 0;
                ff5_uid_o <= 0;
            end else begin
                ff5_rtaddr_e <= iff5_rtaddr_e;
                ff5_rt_e <= iff5_rt_e;
                ff5_wreg_e <= iff5_wreg_e;
                ff5_uid_e <= iff5_uid_e;
                ff5_rtaddr_o <= iff5_rtaddr_o;
                ff5_rt_o <= iff5_rt_o;
                ff5_wreg_o <= iff5_wreg_o;
                ff5_uid_o <= iff5_uid_o;
                ff5_memory_addr_o<=iff5_memory_addr_o;
            end
        end

endmodule


//*************************************************************************
*********/
// Module:      FF6
// File:        FF6.v
// Description: The stage FF6
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************
**********/

`include "defines.v"

module FF6(

        input wire clk,
        input wire rst,

        //from FF5
        input wire[`REG_ADDR_BUS7] iff6_rtaddr_e,
        input wire iff6_wreg_e,
        input wire[`REG_BUS128] iff6_rt_e,
        input wire[0:2] iff6_uid_e,

        input wire[`REG_ADDR_BUS7] iff6_rtaddr_o,
        input wire iff6_wreg_o,
        input wire[`REG_BUS128] iff6_rt_o,
        input wire[0:2] iff6_uid_o,
```

```
    //memory address
    input wire[0:31] iff6_memory_addr_o,

    //to FF7
    output reg[`REG_ADDR_BUS7] ff6_rtaddr_e,
    output reg ff6_wreg_e,
    output reg[`REG_BUS128] ff6_rt_e,
    output reg[0:2] ff6_uid_e,

    output reg[`REG_ADDR_BUS7] ff6_rtaddr_o,
    output reg ff6_wreg_o,
    output reg[`REG_BUS128] ff6_rt_o,
    output reg[0:2] ff6_uid_o,
    //memory address
    output reg[0:31] ff6_memory_addr_o
);


    always @ (posedge clk) begin
        if(rst == `RST_ENABLE) begin
            ff6_rtaddr_e <= 0;
            ff6_rt_e <= 0;
            ff6_wreg_e <= 0;
            ff6_uid_e <= 0;
            ff6_rtaddr_o <= 0;
            ff6_rt_o <= 0;
            ff6_wreg_o <= 0;
            ff6_uid_o <= 0;
        end else begin
            ff6_rtaddr_e <= iff6_rtaddr_e;
            ff6_rt_e <= iff6_rt_e;
            ff6_wreg_e <= iff6_wreg_e;
            ff6_uid_e <= iff6_uid_e;
            ff6_rtaddr_o <= iff6_rtaddr_o;
            ff6_rt_o <= iff6_rt_o;
            ff6_wreg_o <= iff6_wreg_o;
            ff6_uid_o <= iff6_uid_o;
            ff6_memory_addr_o<=iff6_memory_addr_o;
        end
    end
endmodule


//************************************************************************
```

```
*********/
// Module:      FF7
// File:        FF7.v
// Description: The stage FF7
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************
**********/

`include "defines.v"

module FF7(

    input wire clk,
    input wire rst,

    //from FF6
    input wire[`REG_ADDR_BUS7] iff7_rtaddr_e,
    input wire iff7_wreg_e,
    input wire[`REG_BUS128] iff7_rt_e,
    input wire[0:2] iff7_uid_e,

    input wire[`REG_ADDR_BUS7] iff7_rtaddr_o,
    input wire iff7_wreg_o,
    input wire[`REG_BUS128] iff7_rt_o,
    input wire[0:2] iff7_uid_o,
    //memory address
    input wire[0:31] iff7_memory_addr_o,

    //to FF/MEM
    output reg[`REG_ADDR_BUS7] ff7_rtaddr_e,
    output reg ff7_wreg_e,
    output reg[`REG_BUS128] ff7_rt_e,
    output reg[0:2] ff7_uid_e,

    output reg[`REG_ADDR_BUS7] ff7_rtaddr_o,
    output reg ff7_wreg_o,
    output reg[`REG_BUS128] ff7_rt_o,
    output reg[0:2] ff7_uid_o,
    //memory address
    output reg[0:31] ff7_memory_addr_o
);

    always @ (posedge clk) begin
        if(rst == `RST_ENABLE) begin
```

```
                ff7_rtaddr_e <= 0;
                ff7_rt_e <= 0;
                ff7_wreg_e <= 0;
                ff7_uid_e <= 0;
                ff7_rtaddr_o <= 0;
                ff7_rt_o <= 0;
                ff7_wreg_o <= 0;
                ff7_uid_o <= 0;
            end else begin
                ff7_rtaddr_e <= iff7_rtaddr_e;
                ff7_rt_e <= iff7_rt_e;
                ff7_wreg_e <= iff7_wreg_e;
                ff7_uid_e <= iff7_uid_e;
                ff7_rtaddr_o <= iff7_rtaddr_o;
                ff7_rt_o <= iff7_rt_o;
                ff7_wreg_o <= iff7_wreg_o;
                ff7_uid_o <= iff7_uid_o;
                ff7_memory_addr_o<=iff7_memory_addr_o;
            end
        end
endmodule


//**********************************************************************************
*********/
// Module:      FF_MEM
// File:        FF_MEM.v
// Description: The stage between FF and MEM
// History:     Created by Ning Kang, Mar 28, 2018
//**********************************************************************************
**********/

`include "defines.v"

module FF_MEM(

    input wire clk,
    input wire rst,

    //info from ff stage
    input wire[`REG_ADDR_BUS7] ff7_rtaddr_e,
    input wire ff7_wreg_e,
    input wire[`REG_BUS128] ff7_rt_e,
    input wire[0:2] ff7_uid_e,
```

```
    input wire[`REG_ADDR_BUS7] ff7_rtaddr_o,
    input wire ff7_wreg_o,
    input wire[`REG_BUS128] ff7_rt_o,
    input wire[0:2] ff7_uid_o,
    //memory address
    input wire[0:31] ff7_memory_addr_o,

    input wire[0:12] stall,

    //info to MEM stage
    output reg[`REG_ADDR_BUS7] mem_rtaddr_e,
    output reg mem_wreg_e,
    output reg[`REG_BUS128] mem_rt_e,

    output reg[`REG_ADDR_BUS7] mem_rtaddr_o,
    output reg mem_wreg_o,
    output reg[`REG_BUS128] mem_rt_o,
    //memory address
    output reg[0:31] mem_memory_addr_o,
    output reg[0:2] mem_uid_o
);

    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            mem_rtaddr_e = `NOP_REG_ADDR;
            mem_rt_e = `ZERO_QWORD128;
            mem_wreg_e = `WR_DISABLE;
            mem_rtaddr_o = `NOP_REG_ADDR;
            mem_rt_o = `ZERO_QWORD128;
            mem_wreg_o = `WR_DISABLE;
        end else if (stall[3] == `STOP && stall[11]) begin
            mem_rtaddr_e = `NOP_REG_ADDR;
            mem_rt_e = `ZERO_QWORD128;
            mem_wreg_e = `WR_DISABLE;
            mem_rtaddr_o = `NOP_REG_ADDR;
            mem_rt_o = `ZERO_QWORD128;
            mem_wreg_o = `WR_DISABLE;
        end else begin
            mem_rtaddr_e = ff7_rtaddr_e;
            mem_rt_e = ff7_rt_e;
            mem_wreg_e = ff7_wreg_e;
            mem_rtaddr_o = ff7_rtaddr_o;
            mem_rt_o = ff7_rt_o;
```

```
                mem_wreg_o = ff7_wreg_o;
                mem_uid_o = ff7_uid_o;
                mem_memory_addr_o=ff7_memory_addr_o;
            end
        end

endmodule


//*************************************************************************
*********/
// Module:      MEM
// File:        mem.v
// Description: The stage of memory access
// History:     Created by Ning Kang, Mar 28, 2018
//*************************************************************************
**********/

`include "defines.v"

module MEM(

    input wire rst,

    //the info from ff
    input wire[`REG_ADDR_BUS7] i_rtaddr_e,
    input wire i_wreg_e,
    input wire[`REG_BUS128] i_rt_e,

    input wire[`REG_ADDR_BUS7] i_rtaddr_o,
    input wire i_wreg_o,
    input wire[`REG_BUS128] i_rt_o,
    //memory address
    input wire[`MEMORY_WIDTH32] i_memory_addr_o,
    input wire[`REG_BUS128] rt_from_memory,
    input wire[0:2] i_uid_o,

    //the info to mem
    output reg[`REG_ADDR_BUS7] o_rtaddr_e,
    output reg o_wreg_e,
    output reg[`REG_BUS128] o_rt_e,

    output reg[`REG_ADDR_BUS7] o_rtaddr_o,
    output reg o_wreg_o,
```

```
    output reg[`REG_BUS128] o_rt_o,
    //imformation for memory
    output reg[`MEMORY_WIDTH32] mem_data_addr,
    output reg write_memory_enable,
    output reg[`REG_BUS128] rt_to_memory

);

    always @ (*) begin
        if(rst == `RST_ENABLE) begin
            o_rtaddr_e = 0;
            o_rt_e = 0;
            o_wreg_e = 0;
            o_rtaddr_o = 0;
            o_rt_o = 0;
            o_wreg_o = 0;
        end else begin
            if (i_uid_o==`UID_5)
                begin
                mem_data_addr=i_memory_addr_o;
                o_rtaddr_o = i_rtaddr_o;
                o_wreg_o = i_wreg_o;
                o_rt_o=rt_from_memory;
                write_memory_enable=1'b0;
                end
            else if (i_uid_o==`UID_6)
                begin
                write_memory_enable=1'b1;
                mem_data_addr=i_memory_addr_o;
                rt_to_memory=i_rt_o;
                end
            else
                begin
                o_rtaddr_e = i_rtaddr_e;
                o_rt_e = i_rt_e;
                o_wreg_e = i_wreg_e;
                o_rtaddr_o = i_rtaddr_o;
                o_rt_o = i_rt_o;
                o_wreg_o = i_wreg_o;
                end
        end
    end

endmodule
```

```
`include "defines.v"
module MEMORY(
	input wire clk,
	input wire enable,
	input wire[`INST_ADDR_BUS32]inst_addr,
	input wire[`INST_ADDR_BUS32]data_addr,
	input wire write_enable,
	input wire[`MEMORY_WIDTH128] data_in,
	output reg[`MEMORY_WIDTH128] data_out,
	output reg[`MEMORY_WIDTH32] inst_out,
	output reg[`MEMORY_WIDTH32] inst_addr_to_cache
);
	//memory is    128bits*1628
	//total memory size is 256k
	reg[`MEMORY_WIDTH128] memory_data[0:`MEMORY_256K-1];
	reg[`MEMORY_WIDTH128] temp;
	reg[0:1] offset;

	initial $readmemh ( "memory.data", memory_data );

	always @(*) begin
		if (~enable)	begin
			//data_out<=`ZERO_QWORD128;
			//inst_out<=`ZERO_WORD32;
		end
		else begin
			// store data to memory
			if (write_enable) begin
					//decode the data adddress to sovle structure hazard
					if (data_addr[13]==1'b1)
						//if the 13th bit of address is 1
						//means the address is in the data memory parition
						memory_data[data_addr[14:27]]<=data_in;
					end

			else begin
				//fetch data to register
				data_out<=memory_data[data_addr[14:27]];

				//send instruction data to cache
				temp<=memory_data[inst_addr[14:27]];
				offset<=inst_addr[28:29];
```

```
                    if (offset==2'b00)
                    inst_out<=temp[0:31];
                    else if (offset==2'b01)
                    inst_out<=temp[32:63];
                    else if (offset==2'b10)
                    inst_out<=temp[64:95];
                    else if (offset==2'b11)
                    inst_out<=temp[96:127];
                inst_addr_to_cache<=inst_addr;
                end
            end
        end
endmodule


//**********************************************************************************
*********/
// Module:      MEM_WB
// File:        mem_wb.v
// Description: The stage of memory access
// History:     Created by Ning Kang, Mar 28, 2018
//**********************************************************************************
**********/

`include "defines.v"

module MEM_WB(

    input wire clk,
    input wire rst,

    //info from memory
    input wire[`REG_ADDR_BUS7] mem_rtaddr_e,
    input wire mem_wreg_e,
    input wire[`REG_BUS128] mem_rt_e,

    input wire[`REG_ADDR_BUS7] mem_rtaddr_o,
    input wire mem_wreg_o,
    input wire[`REG_BUS128] mem_rt_o,

    input wire[0:12] stall,

    //info for write back
    output reg[`REG_ADDR_BUS7] wb_rtaddr_e,
```

```
    output reg wb_wreg_e,
    output reg[`REG_BUS128] wb_rt_e,

    output reg[`REG_ADDR_BUS7] wb_rtaddr_o,
    output reg wb_wreg_o,
    output reg[`REG_BUS128] wb_rt_o

);

    always @ (posedge clk) begin
        if(rst == `RST_ENABLE) begin
            wb_rtaddr_e = `NOP_REG_ADDR;
            wb_rt_e = `ZERO_QWORD128;
            wb_wreg_e = `WR_DISABLE;
            wb_rtaddr_o = `NOP_REG_ADDR;
            wb_rt_o = `ZERO_QWORD128;
            wb_wreg_o = `WR_DISABLE;
        end else if(stall[11] == `STOP && stall[12] == `NOSTOP) begin
            wb_rtaddr_e = `NOP_REG_ADDR;
            wb_rt_e = `ZERO_QWORD128;
            wb_wreg_e = `WR_DISABLE;
            wb_rtaddr_o = `NOP_REG_ADDR;
            wb_rt_o = `ZERO_QWORD128;
            wb_wreg_o = `WR_DISABLE;
        end else if(stall[12] == `NOSTOP) begin
            wb_rtaddr_e = mem_rtaddr_e;
            wb_rt_e = mem_rt_e;
            wb_wreg_e = mem_wreg_e;
            wb_rtaddr_o = mem_rtaddr_o;
            wb_rt_o = mem_rt_o;
            wb_wreg_o = mem_wreg_o;
        end
    end


endmodule


//*************************************************************************
*********/
// Module:      CACHE
// File:        Cache.v
// Description: Cache instance - 4KB
// History:     Created by Ning Kang, Apr 20, 2018
```

```
//*********************************************************************************
**********/

`include "defines.v"

module CACHE(

	input wire clk,
	input wire ce,
	input wire[`INST_ADDR_BUS32] pc, //32bits
	input wire[`INST_BUS32] inst_from_memory,
	input wire[`INST_BUS32] addr_from_memory,
	output reg[`INST2_BUS64] inst, //64bits for 2 inst
	output reg[`MISS1] miss,
	output reg[`INST_ADDR_BUS32] addr_to_memory

);

	reg[`CACHE_WIDTH53] ins_cc[0:`CACHE_NUM_4K-1]; //fetch instruction from 4K Cache
	reg[`INST_ADDR_BUS32] addr2;//addr2 is the addr for inst2 addr2=addr+4
	reg[`INST_BUS32] inst1;
	reg[`INST_BUS32] inst2;
	reg[`CACHE_INDEX10] index1;
	reg[`CACHE_TAG20] tag1;
	reg[`CACHE_INDEX10] index2; //index for inst2
	reg[`CACHE_TAG20] tag2;//tag for inst2
	reg[`MISS1] miss1;
	reg[`MISS1] miss2;

	initial $readmemb ( "cache.data", ins_cc );

	always @ (posedge clk)
	begin
		if (ce == `CHIP_DISABLE)
			begin
				inst <= `ZERO_DWORD64;
			end
		else
			begin
			//decode the address get the cache_index and tag
			index1=pc[20:29];
			tag1=pc[0:19];
			addr2=pc+32'h4;
			index2=addr2[20:29];
```

```
            tag2=addr2[0:19];

            //check the valid bit and tag
            if (ins_cc[index1][0:0] && (ins_cc[index1][1:20])==tag1)
                begin
                // fetch instrutions from Cache!!!
                inst1= ins_cc[index1];
                miss1=1'b0;
                end
            else
                //miss condition
                begin
                miss1=1'b1;
                addr_to_memory=pc;
                end

            //check the instrucation2
            if (ins_cc[index2][0:0] && (ins_cc[index2][1:20])==tag2)
                begin
                // fetch instrutions from Cache!!!
                inst2= ins_cc[index2];
                miss2=1'b0;
                end
            else
                //miss condition
                begin
                miss2=1'b1;
                addr_to_memory=addr2;
                end

            //send the miss flag to contrl unit
            miss=(miss1 || miss2);

            //join inst1 and inst2 together
            if (miss==0)
                inst={inst1,inst2};
            end
end


//miss condition, fetch data from memory to cache
always@(*)
begin
```

```
        ins_cc[addr_from_memory[20:29]]<={1'b1,addr_from_memory[0:19],inst_from_memory};
        end
endmodule


//*************************************************************************************
*********/
// Module:      Control
// File:        CTRL.v
// Description: CTRL module for pipeline stall
// History:     Created by Ning Kang, Apr 20, 2018
//*************************************************************************************
**********/

`include "defines.v"

/*
stall <= 13'b0000000001    //PC holds
stall <= 13'b0000000011    //IF stall
stall <= 13'b0000000111   //ID stall
stall <= 13'b0000001111    //EX stall
stall <= 13'b0000011111   //MEM stall
stall <= 13'b0000111111    //WR stall
*/
module CTRL(
        input rst,
        input wire stallreq_fr_id,
        input wire stallreq_dh,
        input wire stallreq_fr_ex,
        input wire stallreq_fr_cache,
        output reg [0:12] stall
);

        always @ (*) begin
                if (rst == `RST_ENABLE) begin
                        stall <= 13'b0000000000;
                end else if (stallreq_fr_ex == `STOP) begin
                        stall <= 13'b0000001111;
                end else if (stallreq_fr_id == `STOP) begin
                        stall <= 13'b0000000111;
                end else if (stallreq_fr_cache == `STOP) begin
                        stall <= 13'b0000000011;
                end else if (stallreq_dh == `STOP) begin
                        stall <= 13'b0000001000;
```

```
            end else begin
                stall <= 13'b0000000000;
            end
        end
endmodule


//*************************************************************************
*********/
// Module:      SPU
// File:        spu.v
// Description: the top file of SPU
// History:     Created by Ning Kang, Mar 30, 2018
//              Modify by Ning Kang, Apr 25, 2018 - correct the connection between ID/EX
//*************************************************************************
**********/

`include "defines.v"

module SPU(

    input wire clk,
    input wire rst,

    //stall request from cache
    input wire    stallreq_fr_cache,
    //input wire[`REG_BUS128] rom_data_i,
    input wire[`INST2_BUS64] rom_data_i,
    //output wire[`REG_BUS128] rom_addr_o,
    output wire[`INST_ADDR_BUS32] rom_addr_o,
    input wire[`MEMORY_WIDTH32] i_memory_addr_o,
    input wire[`REG_BUS128] rt_from_memory,
    output wire[0:31] mem_data_addr,
    output wire write_memory_enable,
    output wire[`REG_BUS128] rt_to_memory,
    output wire rom_ce_o

);

    wire[`INST_ADDR_BUS32] pc;
    wire[`INST_ADDR_BUS32] id_pc_i;
    wire[`INST2_BUS64] id_inst_i;

    //connection between the output of ID and the input of ID/EX
```

```
wire[10:0] id_opcode_e;
wire[`REG_BUS128] id_ra_e;
wire[`REG_BUS128] id_rb_e;
wire[`REG_ADDR_BUS7] id_rtaddr_e;
wire id_wreg_e;
//wire id_even;
wire[0:2] id_uid_e;

wire[0:10] id_opcode_o;
wire[`REG_BUS128] id_ra_o;
wire[`REG_BUS128] id_rb_o;
wire[`REG_ADDR_BUS7] id_rtaddr_o;
wire id_wreg_o;
wire[0:9] id_imm_o;
//wire id_odd;
wire[0:2] id_uid_o;

wire is_in_delayslot;

wire[0:31] id_link_addr_o;
wire id_is_in_delayslot_o;
wire id_next_inst_in_delayslot_o;
wire[0:31] id_branch_target_addr_o;
wire id_branch_flag_o;

wire[0:31] id_pc;
wire[0:31] inst_l;
wire[0:31] inst_h;

wire pipe_l;
wire pipe_h;

//connection between the output of ID/EX and the input of EX(ODD&EVEN)
wire[0:10] iex_opcode_e;
wire[`REG_BUS128] iex_ra_e;
wire[`REG_BUS128] iex_rb_e;
wire[`REG_ADDR_BUS7] iex_rtaddr_e;
wire[0:2] iex_uid_e;
wire iex_wreg_e;

wire[0:10] iex_opcode_o;
wire[`REG_BUS128] iex_ra_o;
wire[`REG_BUS128] iex_rb_o;
wire[`REG_ADDR_BUS7] iex_rtaddr_o;
```

```
wire[0:2] iex_uid_o;
wire[0:9] iex_imm_o;
wire iex_wreg_o;

wire[0:31] ex_pc;
wire[0:31] ex_inst_l;
wire[0:31] ex_inst_h;

wire[0:31] ex_link_addr;
wire ex_is_in_delayslot;
wire[0:31] ex_branch_target_addr;
wire ex_branch_flag;

//connection between ID/EX and ID
wire is_in_delayslot_o;

//connection between the output of EX and the input of EX/FF
wire[`REG_BUS128] oex_rt_e;
wire[`REG_ADDR_BUS7] oex_rtaddr_e;
wire oex_wreg_e;
wire[0:2] oex_uid_e;

wire[`REG_BUS128] oex_rt_o;
wire[`REG_ADDR_BUS7] oex_rtaddr_o;
wire oex_wreg_o;
wire[0:2] oex_uid_o;
wire[0:31]oex_memory_addr_o;

wire[0:31] o_ex_pc;
wire[0:31] o_ex_link_addr;
wire o_ex_is_in_delayslot;
wire[0:31] o_ex_branch_target_addr;
wire o_ex_branch_flag;

//connection between the output of EX/FF and the input of FF1
wire iff1_wreg_e;
wire[`REG_ADDR_BUS7] iff1_rtaddr_e;
wire[`REG_BUS128] iff1_rt_e;
wire[0:2] iff1_uid_e;

wire iff1_wreg_o;
wire[`REG_ADDR_BUS7] iff1_rtaddr_o;
wire[`REG_BUS128] iff1_rt_o;
wire[0:2] iff1_uid_o;
```

```
wire[0:31] iff1_memory_addr_o;

wire iff1_branch_flag;
wire[0:31] iff1_branch_target_addr;
wire[0:31] iff1_link_addr;
wire iff1_is_in_delayslot;

//connection between the output of FF1 and the input of FF2
wire iff2_wreg_e;
wire[`REG_ADDR_BUS7] iff2_rtaddr_e;
wire[`REG_BUS128] iff2_rt_e;
wire[0:2] iff2_uid_e;
wire[0:31] iff2_memory_addr_o;

wire iff2_wreg_o;
wire[`REG_ADDR_BUS7] iff2_rtaddr_o;
wire[`REG_BUS128] iff2_rt_o;
wire[0:2] iff2_uid_o;

wire iff2_branch_flag;
wire[0:31] iff2_branch_target_addr;
wire[0:31] iff2_link_addr;
wire iff2_is_in_delayslot;

//connection between the output of FF2 and the input of FF3
wire iff3_wreg_e;
wire[`REG_ADDR_BUS7] iff3_rtaddr_e;
wire[`REG_BUS128] iff3_rt_e;
wire[0:2] iff3_uid_e;

wire iff3_wreg_o;
wire[`REG_ADDR_BUS7] iff3_rtaddr_o;
wire[`REG_BUS128] iff3_rt_o;
wire[0:2] iff3_uid_o;
wire[0:31] iff3_memory_addr_o;

wire iff3_branch_flag;
wire[0:31] iff3_branch_target_addr;
wire[0:31] iff3_link_addr;
wire iff3_is_in_delayslot;

//connection between the output of FF3 and the input of FF4
wire iff4_wreg_e;
wire[`REG_ADDR_BUS7] iff4_rtaddr_e;
```

```
wire[`REG_BUS128] iff4_rt_e;
wire[0:2] iff4_uid_e;

wire iff4_wreg_o;
wire[`REG_ADDR_BUS7] iff4_rtaddr_o;
wire[`REG_BUS128] iff4_rt_o;
wire[0:2] iff4_uid_o;
wire[0:31]iff4_memory_addr_o;

wire iff4_branch_flag;
wire[0:31] iff4_branch_target_addr;
wire[0:31] iff4_link_addr;
wire iff4_is_in_delayslot;

//connection between the output of FF4 and the input of FF5
wire iff5_wreg_e;
wire[`REG_ADDR_BUS7] iff5_rtaddr_e;
wire[`REG_BUS128] iff5_rt_e;

wire iff5_wreg_o;
wire[`REG_ADDR_BUS7] iff5_rtaddr_o;
wire[`REG_BUS128] iff5_rt_o;
wire[0:2] iff5_uid_o;
wire[0:31] iff5_memory_addr_o;

//connection between the output of FF5 and the input of FF6
wire iff6_wreg_e;
wire[`REG_ADDR_BUS7] iff6_rtaddr_e;
wire[`REG_BUS128] iff6_rt_e;

wire iff6_wreg_o;
wire[`REG_ADDR_BUS7] iff6_rtaddr_o;
wire[`REG_BUS128] iff6_rt_o;
wire[0:2] iff6_uid_o;
wire[0:31]iff6_memory_addr_o;

//connection between the output of FF6 and the input of FF7
wire iff7_wreg_e;
wire[`REG_ADDR_BUS7] iff7_rtaddr_e;
wire[`REG_BUS128] iff7_rt_e;

wire iff7_wreg_o;
wire[`REG_ADDR_BUS7] iff7_rtaddr_o;
wire[`REG_BUS128] iff7_rt_o;
```

```
wire[0:2] iff7_uid_o;
wire[0:31]iff7_memory_addr_o;

//connection between the output of FF7 and the input of FF/MEM
wire off7_wreg_e;
wire[`REG_ADDR_BUS7] off7_rtaddr_e;
wire[`REG_BUS128] off7_rt_e;

wire off7_wreg_o;
wire[`REG_ADDR_BUS7] off7_rtaddr_o;
wire[`REG_BUS128] off7_rt_o;
wire[0:31]off7_memory_addr_o;
wire[0:2] off7_uid_o;
//connection between the output of FF/MEM and the input of MEM
wire imem_wreg_e;
wire[`REG_ADDR_BUS7] imem_rtaddr_e;
wire[`REG_BUS128] imem_rt_e;

wire imem_wreg_o;
wire[`REG_ADDR_BUS7] imem_rtaddr_o;
wire[`REG_BUS128] imem_rt_o;
wire[0:31]imem_memory_addr_o;
wire[0:2] imem_uid_o;
//connection between the output of MEM and the input of MEM/WB
wire omem_wreg_e;
wire[`REG_ADDR_BUS7] omem_rtaddr_e;
wire[`REG_BUS128] omem_rt_e;

wire omem_wreg_o;
wire[`REG_ADDR_BUS7] omem_rtaddr_o;
wire[`REG_BUS128] omem_rt_o;

//connection between the output of MEM/WB and the input of WB
wire iwb_wreg_e;
wire[`REG_ADDR_BUS7] iwb_rtaddr_e;
wire[`REG_BUS128] iwb_rt_e;

wire iwb_wreg_o;
wire[`REG_ADDR_BUS7] iwb_rtaddr_o;
wire[`REG_BUS128] iwb_rt_o;

//connection between I/O of ID and I/O of Regfile
wire ira_rd_e;
wire irb_rd_e;
```

```
wire[`REG_BUS128] ora_data_e;
wire[`REG_BUS128] orb_data_e;
wire[`REG_ADDR_BUS7] ira_addr_e;
wire[`REG_ADDR_BUS7] irb_addr_e;

wire ira_rd_o;
wire irb_rd_o;
wire[`REG_BUS128] ora_data_o;
wire[`REG_BUS128] orb_data_o;
wire[`REG_ADDR_BUS7] ira_addr_o;
wire[`REG_ADDR_BUS7] irb_addr_o;

//connection between CTRL and multiple modules
wire [0:12] stall;
wire stallreq_fr_id;
wire stallreq_fr_ex;
wire stallreq_dh;

//connection between FF4 and PC_REG
wire branch_flag;
wire[0:31] branch_target_addr;

//pc_reg
PC_REG pc_reg(
	.clk(clk),
	.rst(rst),
	.stall(stall),
	.id_pc(id_pc),
	.branch_flag_i(branch_flag),
	//.branch_clr(branch_clr),
	.branch_target_addr_i(branch_target_addr),
	//.branch_target_addr_i(iff4_branch_target_addr),
	.pc(pc),
	.ce(rom_ce_o)

);

assign rom_addr_o = pc;

//IF/ID
IF_ID if_id(
	.clk(clk),
	.rst(rst),
	.if_pc(pc),
```

```
        .if_inst(rom_data_i),
        .stall(stall),
        .pipe_l(pipe_l),
        .pipe_h(pipe_h),
        .id_pc(id_pc_i),
        .id_inst(id_inst_i)
    );

    //ID
    ID id(
        .rst(rst),
        .clk(clk),
        .pc_i(id_pc_i),
        .inst_i(id_inst_i),

        //from FF network
        .ex_wreg_i_e(oex_wreg_e),
        .ex_wdata_i_e(oex_rt_e),
        .ex_waddr_i_e(oex_rtaddr_e),

        .ff1_wreg_i_e(iff2_wreg_i_e),
        .ff1_rt_e(iff2_rt_e),
        .ff1_waddr_i_e(iff2_waddr_i_e),

        .ff2_wreg_i_e(iff3_wreg_i_e),
        .ff2_rt_e(iff3_rt_e),
        .ff2_waddr_i_e(iff3_waddr_i_e),

        .ff3_wreg_i_e(iff4_wreg_i_e),
        .ff3_rt_e(iff4_rt_e),
        .ff3_waddr_i_e(iff4_waddr_i_e),

        .ff4_wreg_i_e(iff5_wreg_i_e),
        .ff4_rt_e(iff5_rt_e),
        .ff4_waddr_i_e(iff5_waddr_i_e),

        .ff6_wreg_i_e(iff7_wreg_i_e),
        .ff6_rt_e(iff7_rt_e),
        .ff6_waddr_i_e(iff7_waddr_i_e),

        .ff7_wreg_i_e(off7_wreg_i_e),
        .ff7_rt_e(off7_rt_e),
        .ff7_waddr_i_e(off7_waddr_i_e),
```

```
.ex_wreg_i_o(oex_wreg_o),
.ex_wdata_i_o(oex_rt_o),
.ex_waddr_i_o(oex_rtaddr_o),

.ff1_wreg_i_o(iff2_wreg_i_o),
.ff1_rt_o(iff2_rt_o),
.ff1_waddr_i_o(iff2_waddr_i_o),

.ff2_wreg_i_o(iff3_wreg_i_o),
.ff2_rt_o(iff3_rt_o),
.ff2_waddr_i_o(iff3_waddr_i_o),

.ff3_wreg_i_o(iff4_wreg_i_o),
.ff3_rt_o(iff4_rt_o),
.ff3_waddr_i_o(iff4_waddr_i_o),

.ff4_wreg_i_o(iff5_wreg_i_o),
.ff4_rt_o(iff5_rt_o),
.ff4_waddr_i_o(iff5_waddr_i_o),

.ff6_wreg_i_o(iff7_wreg_i_o),
.ff6_rt_o(iff7_rt_o),
.ff6_waddr_i_o(iff7_waddr_i_o),

.ff7_wreg_i_o(off7_wreg_i_o),
.ff7_rt_o(off7_rt_o),
.ff7_waddr_i_o(off7_waddr_i_o),

//from mem
.mem_wreg_i_e(omem_wreg_e),
.mem_wdata_i_e(omem_rt_e),
.mem_waddr_i_e(omem_rtaddr_e),
.mem_wreg_i_o(omem_wreg_o),
.mem_wdata_i_o(omem_rt_o),
.mem_waddr_i_o(omem_rtaddr_o),


//from register file
.ra_i_e(ora_data_e),
.rb_i_e(orb_data_e),

.ra_i_o(ora_data_o),
.rb_i_o(orb_data_o),
```

```
//from ID_EX
.is_in_delayslot_i(is_in_delayslot),

//to ID_EX
.link_addr_o(id_link_addr_o),
.is_in_delayslot_o(id_is_in_delayslot_o),
.next_inst_in_delayslot_o(id_next_inst_in_delayslot_o),
.branch_target_addr_o(id_branch_target_addr_o),
.branch_flag_o(id_branch_flag_o),

//to ID_EX
.id_pc(id_pc),
.inst_l(inst_l),
.inst_h(inst_h),


//to register file
.ra_rd_e(ira_rd_e),
.rb_rd_e(irb_rd_e),
.ra_addr_e(ira_addr_e),
.rb_addr_e(irb_addr_e),

.ra_rd_o(ira_rd_o),
.rb_rd_o(irb_rd_o),
.ra_addr_o(ira_addr_o),
.rb_addr_o(irb_addr_o),

//info send to id/exe
.op_code_e(id_opcode_e),
.ra_do_e(id_ra_e),
.rb_do_e(id_rb_e),
.rt_addr_e(id_rtaddr_e),
.wreg_e(id_wreg_e),
//.even(id_even),
.uid_e(id_uid_e),

.op_code_o(id_opcode_o),
.ra_do_o(id_ra_o),
.rb_do_o(id_rb_o),
.rt_addr_o(id_rtaddr_o),
.wreg_o(id_wreg_o),
//.odd(id_odd),
.uid_o(id_uid_o),
.im_o(id_imm_o),
```

```
        .pipe_l(pipe_l),
        .pipe_h(pipe_h),
        .stallreq(stallreq_fr_id),
        .stallreq_dh(stallreq_dh)
);

//Regfile
REGFILE regfile(
        .clk (clk),
        .rst (rst),
        .rt_we_e(iwb_wreg_e),
        .rt_addr_e (iwb_rtaddr_e),
        .rt_data_e (iwb_rt_e),
        .ra_e_e (ira_rd_e),
        .ra_addr_e (ira_addr_e),
        .ra_data_e (ora_data_e),
        .rb_e_e (irb_rd_e),
        .rb_addr_e (irb_addr_e),
        .rb_data_e (orb_data_e),

        .rt_we_o(iwb_wreg_o),
        .rt_addr_o (iwb_rtaddr_o),
        .rt_data_o (iwb_rt_o),
        .ra_e_o (ira_rd_o),
        .ra_addr_o (ira_addr_o),
        .ra_data_o (ora_data_o),
        .rb_e_o (irb_rd_o),
        .rb_addr_o (irb_addr_o),
        .rb_data_o (orb_data_o)
);

//ID/EX
ID_EX id_ex(
        .clk(clk),
        .rst(rst),

        //from ID
        .id_pc(id_pc),
        .id_inst_l(inst_l),
        .id_inst_h(inst_h),

        .id_opcode_e(id_opcode_e),
        .id_ra_e(id_ra_e),
        .id_rb_e(id_rb_e),
```

```
.id_rtaddr_e(id_rtaddr_e),
.id_uid_e(id_uid_e),
.id_wreg_e(id_wreg_e),

.id_opcode_o(id_opcode_o),
.id_ra_o(id_ra_o),
.id_rb_o(id_rb_o),
.id_rtaddr_o(id_rtaddr_o),
.id_uid_o(id_uid_o),
.id_wreg_o(id_wreg_o),
.id_imm_o(id_imm_o),

.id_link_addr(id_link_addr_o),
.id_is_in_delayslot(id_is_in_delayslot_o),
.id_next_inst_in_delayslot(id_next_inst_in_delayslot_o),
.id_branch_target_addr(id_branch_target_addr_o),
.id_branch_flag(id_branch_flag_o),

//from CTRL
.stall(stall),

//to EX
.ex_pc(ex_pc),
.ex_inst_l(ex_inst_l),
.ex_inst_h(ex_inst_h),

.ex_opcode_e(iex_opcode_e),
.ex_ra_e(iex_ra_e),
.ex_rb_e(iex_rb_e),
.ex_rtaddr_e(iex_rtaddr_e),
.ex_wreg_e(iex_wreg_e),
.ex_uid_e(iex_uid_e),

.ex_opcode_o(iex_opcode_o),
.ex_ra_o(iex_ra_o),
.ex_rb_o(iex_rb_o),
.ex_rtaddr_o(iex_rtaddr_o),
.ex_wreg_o(iex_wreg_o),
.ex_uid_o(iex_uid_o),
.ex_imm_o(iex_imm_o),

.ex_link_addr(ex_link_addr),
.ex_is_in_delayslot(ex_is_in_delayslot),
.ex_branch_target_addr(ex_branch_target_addr),
```

```
        .ex_branch_flag(ex_branch_flag),

        // to ID
        .is_in_delayslot_o(is_in_delayslot)
    );

    //EX
    EX_EVEN ex_even(
        .rst(rst),
        .clk(clk),

        //from ID_EX
        .i_opcode_e(iex_opcode_e),
        .i_ra_e(iex_ra_e),
        .i_rb_e(iex_rb_e),
        .i_rtaddr_e(iex_rtaddr_e),
        .i_wreg_e(iex_wreg_e),
        .i_uid_e(iex_uid_e),

        //to EX_FF
        .o_rtaddr_e(oex_rtaddr_e),
        .o_wreg_e(oex_wreg_e),
        .o_rt_e(oex_rt_e),
        .o_uid_e(oex_uid_e)
    );

    EX_ODD ex_odd(
        .rst(rst),
        .clk(clk),

        //from ID_EX
        .i_opcode_o(iex_opcode_o),
        .i_ra_o(iex_ra_o),
        .i_rb_o(iex_rb_o),
        .i_rtaddr_o(iex_rtaddr_o),
        .i_wreg_o(iex_wreg_o),
        .i_uid_o(iex_uid_o),
        .i_imm_o(iex_imm_o),

        .i_pc(ex_pc),
        .i_ex_link_addr(ex_link_addr),
        .i_ex_id_is_in_delayslot(ex_is_in_delayslot),
        .i_ex_branch_target_addr(ex_branch_target_addr),
        .i_ex_branch_flag(ex_branch_flag),
```

```
        //EX to EX/FF
        .o_rtaddr_o(oex_rtaddr_o),
        .o_wreg_o(oex_wreg_o),
        .o_rt_o(oex_rt_o),
        .o_uid_o(oex_uid_o),
        .o_memory_addr_o(oex_memory_addr_o),


        .ex_pc(o_ex_pc),
        .o_ex_link_addr(o_ex_link_addr),
        .o_ex_is_in_delayslot(o_ex_is_in_delayslot),
        .o_ex_branch_target_addr(o_ex_branch_target_addr),
        .o_ex_branch_flag(o_ex_branch_flag)
);

//EX_FF
EX_FF ex_ff(
        .clk(clk),
        .rst(rst),

        //from EX
        .ex_rtaddr_e(oex_rtaddr_e),
        .ex_wreg_e(oex_wreg_e),
        .ex_rt_e(oex_rt_e),
        .ex_uid_e(oex_uid_e),

        .ex_rtaddr_o(oex_rtaddr_o),
        .ex_wreg_o(oex_wreg_o),
        .ex_rt_o(oex_rt_o),
        .ex_uid_o(oex_uid_o),
        .ex_memory_addr_o(oex_memory_addr_o),

        .ex_link_addr(o_ex_link_addr),
        .ex_is_in_delayslot(o_ex_is_in_delayslot),
        .ex_branch_target_addr(o_ex_branch_target_addr),
        .ex_branch_flag(o_ex_branch_flag),

        //to FF1
        .ff_rtaddr_e(iff1_rtaddr_e),
        .ff_wreg_e(iff1_wreg_e),
        .ff_rt_e(iff1_rt_e),
        .ff_uid_e(iff1_uid_e),
```

```
        .ff_rtaddr_o(iff1_rtaddr_o),
        .ff_wreg_o(iff1_wreg_o),
        .ff_rt_o(iff1_rt_o),
        .ff_uid_o(iff1_uid_o),
        .ff_memory_addr_o(iff1_memory_addr_o),

        .ff_branch_flag(iff1_branch_flag),
        .ff_branch_target_addr(iff1_branch_target_addr),
        .ff_link_addr(iff1_link_addr),
        .ff_is_in_delayslot(iff1_is_in_delayslot)
    );

    //FF1
    FF1 ff1(
        .clk(clk),
        .rst(rst),

        //from EX_FF
        .iff_rtaddr_e(iff1_rtaddr_e),
        .iff_wreg_e(iff1_wreg_e),
        .iff_rt_e(iff1_rt_e),
        .iff_uid_e(iff1_uid_e),

        .iff_rtaddr_o(iff1_rtaddr_o),
        .iff_wreg_o(iff1_wreg_o),
        .iff_rt_o(iff1_rt_o),
        .iff_uid_o(iff1_uid_o),
        .iff_memory_addr_o(iff1_memory_addr_o),

        .iff_branch_flag(iff1_branch_flag),
        .iff_branch_target_addr(iff1_branch_target_addr),
        .iff_link_addr(iff1_link_addr),
        .iff_is_in_delayslot(iff1_is_in_delayslot),

        //to FF2
        .ff1_rtaddr_e(iff2_rtaddr_e),
        .ff1_wreg_e(iff2_wreg_e),
        .ff1_rt_e(iff2_rt_e),
        .ff1_uid_e(iff2_uid_e),

        .ff1_rtaddr_o(iff2_rtaddr_o),
        .ff1_wreg_o(iff2_wreg_o),
        .ff1_rt_o(iff2_rt_o),
        .ff1_uid_o(iff2_uid_o),
```

```
        .ff1_memory_addr_o(iff2_memory_addr_o),

        .ff1_branch_flag(iff2_branch_flag),
        .ff1_branch_target_addr(iff2_branch_target_addr),
        .ff1_link_addr(iff2_link_addr),
        .ff1_is_in_delayslot(iff2_is_in_delayslot)
    );

    //FF2
    FF2 ff2(
        .clk(clk),
        .rst(rst),

        //from FF1
        .iff2_rtaddr_e(iff2_rtaddr_e),
        .iff2_wreg_e(iff2_wreg_e),
        .iff2_rt_e(iff2_rt_e),
        .iff2_uid_e(iff2_uid_e),

        .iff2_rtaddr_o(iff2_rtaddr_o),
        .iff2_wreg_o(iff2_wreg_o),
        .iff2_rt_o(iff2_rt_o),
        .iff2_uid_o(iff2_uid_o),
        .iff2_memory_addr_o(iff2_memory_addr_o),

        .iff2_branch_flag(iff2_branch_flag),
        .iff2_branch_target_addr(iff2_branch_target_addr),
        .iff2_link_addr(iff2_link_addr),
        .iff2_is_in_delayslot(iff2_is_in_delayslot),

        //to FF3
        .ff2_rtaddr_e(iff3_rtaddr_e),
        .ff2_wreg_e(iff3_wreg_e),
        .ff2_rt_e(iff3_rt_e),
        .ff2_uid_e(iff3_uid_e),

        .ff2_rtaddr_o(iff3_rtaddr_o),
        .ff2_wreg_o(iff3_wreg_o),
        .ff2_rt_o(iff3_rt_o),
        .ff2_uid_o(iff3_uid_o),
        .ff2_memory_addr_o(iff3_memory_addr_o),

        .ff2_branch_flag(iff3_branch_flag),
        .ff2_branch_target_addr(iff3_branch_target_addr),
```

```
        .ff2_link_addr(iff3_link_addr),
        .ff2_is_in_delayslot(iff3_is_in_delayslot)
    );

    //FF3
    FF3 ff3(
        .clk(clk),
        .rst(rst),

        //from FF2
        .iff3_rtaddr_e(iff3_rtaddr_e),
        .iff3_wreg_e(iff3_wreg_e),
        .iff3_rt_e(iff3_rt_e),
        .iff3_uid_e(iff3_uid_e),

        .iff3_rtaddr_o(iff3_rtaddr_o),
        .iff3_wreg_o(iff3_wreg_o),
        .iff3_rt_o(iff3_rt_o),
        .iff3_uid_o(iff3_uid_o),
        .iff3_memory_addr_o(iff3_memory_addr_o),

        .iff3_branch_flag(iff3_branch_flag),
        .iff3_branch_target_addr(iff3_branch_target_addr),
        .iff3_link_addr(iff3_link_addr),
        .iff3_is_in_delayslot(iff3_is_in_delayslot),

        //to FF4
        .ff3_rtaddr_e(iff4_rtaddr_e),
        .ff3_wreg_e(iff4_wreg_e),
        .ff3_rt_e(iff4_rt_e),
        .ff3_uid_e(iff4_uid_e),

        .ff3_rtaddr_o(iff4_rtaddr_o),
        .ff3_wreg_o(iff4_wreg_o),
        .ff3_rt_o(iff4_rt_o),
        .ff3_uid_o(iff4_uid_o),
        .ff3_memory_addr_o(iff4_memory_addr_o),

        .ff3_branch_flag(iff4_branch_flag),
        .ff3_branch_target_addr(iff4_branch_target_addr),
        .ff3_link_addr(iff4_link_addr),
        .ff3_is_in_delayslot(iff4_is_in_delayslot)
    );
```

```
//FF4
FF4 ff4(
        .clk(clk),
        .rst(rst),

        //from FF3
        .iff4_rtaddr_e(iff4_rtaddr_e),
        .iff4_wreg_e(iff4_wreg_e),
        .iff4_rt_e(iff4_rt_e),
        .iff4_uid_e(iff4_uid_e),

        .iff4_rtaddr_o(iff4_rtaddr_o),
        .iff4_wreg_o(iff4_wreg_o),
        .iff4_rt_o(iff4_rt_o),
        .iff4_uid_o(iff4_uid_o),
        .iff4_memory_addr_o(iff4_memory_addr_o),

        .iff4_branch_flag(iff4_branch_flag),
        .iff4_branch_target_addr(iff4_branch_target_addr),
        .iff4_link_addr(iff4_link_addr),
        .iff4_is_in_delayslot(iff4_is_in_delayslot),

        //to FF5
        .ff4_rtaddr_e(iff5_rtaddr_e),
        .ff4_wreg_e(iff5_wreg_e),
        .ff4_rt_e(iff5_rt_e),
        .ff4_uid_e(iff5_uid_e),

        .ff4_rtaddr_o(iff5_rtaddr_o),
        .ff4_wreg_o(iff5_wreg_o),
        .ff4_rt_o(iff5_rt_o),
        .ff4_uid_o(iff5_uid_o),
        .ff4_memory_addr_o(iff5_memory_addr_o),

        .ff4_branch_flag(branch_flag),
        .ff4_branch_target_addr(branch_target_addr),
        .ff4_is_in_delayslot(iff4_is_in_delayslot)
);

//FF5
FF5 ff5(
        .clk(clk),
        .rst(rst),
```

```
        //from FF4
        .iff5_rtaddr_e(iff5_rtaddr_e),
        .iff5_wreg_e(iff5_wreg_e),
        .iff5_rt_e(iff5_rt_e),
        .iff5_uid_e(iff5_uid_e),

        .iff5_rtaddr_o(iff5_rtaddr_o),
        .iff5_wreg_o(iff5_wreg_o),
        .iff5_rt_o(iff5_rt_o),
        .iff5_uid_o(iff5_uid_o),
        .iff5_memory_addr_o(iff5_memory_addr_o),

        //to FF6
        .ff5_rtaddr_e(iff6_rtaddr_e),
        .ff5_wreg_e(iff6_wreg_e),
        .ff5_rt_e(iff6_rt_e),
        .ff5_uid_e(iff6_uid_e),

        .ff5_rtaddr_o(iff6_rtaddr_o),
        .ff5_wreg_o(iff6_wreg_o),
        .ff5_rt_o(iff6_rt_o),
        .ff5_uid_o(iff6_uid_o),
        .ff5_memory_addr_o(iff6_memory_addr_o)
    );

    //FF6
    FF6 ff6(
        .clk(clk),
        .rst(rst),

        //from FF4
        .iff6_rtaddr_e(iff6_rtaddr_e),
        .iff6_wreg_e(iff6_wreg_e),
        .iff6_rt_e(iff6_rt_e),
        .iff6_uid_e(iff6_uid_e),

        .iff6_rtaddr_o(iff6_rtaddr_o),
        .iff6_wreg_o(iff6_wreg_o),
        .iff6_rt_o(iff6_rt_o),
        .iff6_uid_o(iff6_uid_o),
        .iff6_memory_addr_o(iff6_memory_addr_o),

        //to FF6
        .ff6_rtaddr_e(iff7_rtaddr_e),
```

```
        .ff6_wreg_e(iff7_wreg_e),
        .ff6_rt_e(iff7_rt_e),
        .ff6_uid_e(iff7_uid_e),

        .ff6_rtaddr_o(iff7_rtaddr_o),
        .ff6_wreg_o(iff7_wreg_o),
        .ff6_rt_o(iff7_rt_o),
        .ff6_uid_o(iff7_uid_o),
        .ff6_memory_addr_o(iff7_memory_addr_o)
    );

    //FF7
    FF7 ff7(
        .clk(clk),
        .rst(rst),

        //from FF6
        .iff7_rtaddr_e(iff7_rtaddr_e),
        .iff7_wreg_e(iff7_wreg_e),
        .iff7_rt_e(iff7_rt_e),
        .iff7_uid_e(iff7_uid_e),

        .iff7_rtaddr_o(iff7_rtaddr_o),
        .iff7_wreg_o(iff7_wreg_o),
        .iff7_rt_o(iff7_rt_o),
        .iff7_uid_o(iff7_uid_o),
        .iff7_memory_addr_o(iff7_memory_addr_o),

        //to FF/MEM
        .ff7_rtaddr_e(off7_rtaddr_e),
        .ff7_wreg_e(off7_wreg_e),
        .ff7_rt_e(off7_rt_e),
        .ff7_uid_e(off7_uid_e),

        .ff7_rtaddr_o(off7_rtaddr_o),
        .ff7_wreg_o(off7_wreg_o),
        .ff7_rt_o(off7_rt_o),
        .ff7_uid_o(off7_uid_o),
        .ff7_memory_addr_o(off7_memory_addr_o)
    );

    //FF_MEM
    FF_MEM ff_mem(
        .clk(clk),
```

```
        .rst(rst),

        //from FF
        .ff7_rtaddr_e(off7_rtaddr_e),
        .ff7_wreg_e(off7_wreg_e),
        .ff7_rt_e(off7_rt_e),
        .ff7_uid_e(off7_uid_e),

        .ff7_rtaddr_o(off7_rtaddr_o),
        .ff7_wreg_o(off7_wreg_o),
        .ff7_rt_o(off7_rt_o),
        .ff7_uid_o(off7_uid_o),
        .ff7_memory_addr_o(off7_memory_addr_o),

        //from CTRL
        .stall(stall),

        //to MEM
        .mem_rtaddr_e(imem_rtaddr_e),
        .mem_wreg_e(imem_wreg_e),
        .mem_rt_e(imem_rt_e),

        .mem_rtaddr_o(imem_rtaddr_o),
        .mem_wreg_o(imem_wreg_o),
        .mem_rt_o(imem_rt_o),
        .mem_uid_o(imem_uid_o),
        .mem_memory_addr_o(imem_memory_addr_o)
    );

    //MEM
    MEM mem(
        .rst(rst),

        //from FF/MEM
        .i_rtaddr_e(imem_rtaddr_e),
        .i_wreg_e(imem_wreg_e),
        .i_rt_e(imem_rt_e),

        .i_rtaddr_o(imem_rtaddr_o),
        .i_wreg_o(imem_wreg_o),
        .i_rt_o(imem_rt_o),
        .i_memory_addr_o(imem_memory_addr_o),
        .rt_from_memory(rt_from_memory),
        .i_uid_o(imem_uid_o),
```

```
        //to MEM/WB
        .o_rtaddr_e(omem_rtaddr_e),
        .o_wreg_e(omem_wreg_e),
        .o_rt_e(omem_rt_e),

        .o_rtaddr_o(omem_rtaddr_o),
        .o_wreg_o(omem_wreg_o),
        .o_rt_o(omem_rt_o),
        .write_memory_enable(write_memory_enable),
        .rt_to_memory(rt_to_memory),
        .mem_data_addr(mem_data_addr)
    );

    //MEM/WB
    MEM_WB mem_wb(
        .clk(clk),
        .rst(rst),

        //from MEM
        .mem_rtaddr_e(omem_rtaddr_e),
        .mem_wreg_e(omem_wreg_e),
        .mem_rt_e(omem_rt_e),

        .mem_rtaddr_o(omem_rtaddr_o),
        .mem_wreg_o(omem_wreg_o),
        .mem_rt_o(omem_rt_o),

        //from CTRL
        .stall(stall),

        //to Write back
        .wb_rtaddr_e(iwb_rtaddr_e),
        .wb_wreg_e(iwb_wreg_e),
        .wb_rt_e(iwb_rt_e),

        .wb_rtaddr_o(iwb_rtaddr_o),
        .wb_wreg_o(iwb_wreg_o),
        .wb_rt_o(iwb_rt_o)
    );

    CTRL ctrl(
        .rst(rst),
        .stallreq_fr_id(stallreq_fr_id),
        .stallreq_fr_ex(stallreq_fr_ex),
```

```
        .stallreq_fr_cache(stallreq_fr_cache),
        .stallreq_dh(stallreq_dh),
        .stall(stall)
    );

endmodule


//*************************************************************************
*********/
// Module:      SPE
// File:        spe.v
// Description: the top level file of SPE
// History:     Created by Ning Kang, Mar 31, 2018
//              Modified by Ning Kang, Apr 20, 2018 - change ILB to Cache
//*************************************************************************
**********/

`include "defines.v"

module SPE(

    input wire clk,
    input wire rst

);

    //connect to Cache
    wire[`INST_ADDR_BUS32] inst_addr;
    wire[`INST2_BUS64] inst;
    wire rom_ce;
    wire[`INST_ADDR_BUS32] pass_addr;
    wire[`INST_ADDR_BUS32] inst_from_memory;
    wire[`INST_ADDR_BUS32] addr_from_memory;
    wire    stallreq_fr_cache;
    wire[0:127] rt_data;

    wire[0:31] data_addr;
    wire write_memory_enable;
    wire[`REG_BUS128] rt_to_memory;

    SPU spu(
        .clk(clk),
        .rst(rst),
```

```
        .stallreq_fr_cache(stallreq_fr_cache),
        .rom_addr_o(inst_addr),
        .rom_data_i(inst),
        .rom_ce_o(rom_ce),
        .mem_data_addr(data_addr),
        .rt_from_memory(rt_data),
        .write_memory_enable(write_memory_enable),
        .rt_to_memory(rt_to_memory)
    );

    CACHE cache0(
        .clk(clk),
        .ce(rom_ce),
        .pc(inst_addr),
        .inst_from_memory(inst_from_memory),
        .addr_from_memory(addr_from_memory),
        .inst(inst),
        .miss(stallreq_fr_cache),
        .addr_to_memory(pass_addr)
    );
    MEMORY memory0(
        .clk(clk),
        .enable(rom_ce),
        .write_enable(write_memory_enable),
        .inst_addr(pass_addr),
        .inst_out(inst_from_memory),
        .inst_addr_to_cache(addr_from_memory),
        .data_addr(data_addr),
        .data_out(rt_data),
        .data_in(rt_to_memory)
        );


endmodule


//*************************************************************************
*********/
// Module:      SPE
// File:        spe.v
// Description: the top level file of SPE
// History:     Created by Ning Kang, Mar 31, 2018
//*************************************************************************
```

```
**********/

//test bench indicate the signal of clock and the reset.
`timescale 1ns/1ps

module TB_SPE;

	reg CLOCK_50;
	reg rst;

	initial begin
		CLOCK_50 = 1'b0;
		forever #10 CLOCK_50 = ~CLOCK_50;
	end

	initial begin
		rst = 1'b1;
		#195 rst= 1'b0;
		#1000 $stop;
	end

	SPE dut(
		.clk(CLOCK_50),
		.rst(rst)
);

endmodule
```

Appendix C. Source Code: Parser (C++)

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <string>
#include <vector>
using namespace std;

string extend(string str)
{
	string result;
	string temp;
	int num;
	if (str.find("r")==0)
	{
		str=str.substr(1);
		num=stoi(str);
		std::bitset<7> a(num);
		ostringstream stream;
		stream<<a;
		result=stream.str();
	}
	else
	{
		num=stoi(str);
		std::bitset<10> a(num);
		ostringstream stream;
		stream<<a;
		result=stream.str();
	}
	return result;
}

int main()
{
	//read the code from file
	ifstream infile("code.txt", ios::in);
	string line;
	//output hexadecimal
	ofstream outfile("encode.txt",ios::out);

	//map store the word and the opcode
```

```
map<string, string> code_map;
code_map["lqd"]="00110100";
code_map["stqd"]="00100100";
code_map["ah"]="00011001000";
code_map["ahi"]="00011101";
code_map["a"]="00011000000";
code_map["ai"]="00011100";
code_map["sfh"]="00001001000";
code_map["sfhi"]="00001101";
code_map["sf"]="00001000000";
code_map["sfi"]="00001100";
code_map["mpy"]="01111000100";
code_map["mpyu"]="01111001100";
code_map["mpyi"]="01110100";
code_map["and"]="00011000001";
code_map["andbi"]="00010110";
code_map["andhi"]="00010101";
code_map["andi"]="00010100";
code_map["or"]="00001000001";
code_map["orbi"]="00000110";
code_map["orhi"]="00000101";
code_map["ori"]="00000100";
code_map["xor"]="01001000001";
code_map["xorbi"]="01000110";
code_map["xorhi"]="01000101";
code_map["xori"]="01000100";
code_map["nand"]="00011001001";
code_map["nor"]="00001001001";
code_map["shlqbi"]="00111011011";
code_map["shlqbii"]="00111111011";
code_map["rotqby"]="00111011100";
code_map["rotqbyi"]="00111111100";
code_map["cbd"]="00111110100";
code_map["ceqb"]="01111010000";
code_map["ceqbi"]="01111110";
code_map["ceqh"]="01111001000";
code_map["ceqhi"]="01111101";
code_map["ceq"]="01111000000";
code_map["ceqi"]="01111100";
code_map["cgtb"]="01001010000";
code_map["cgtbi"]="01001110";
code_map["cgth"]="01001001000";
code_map["cgthi"]="01001101";
code_map["cgt"]="01001000000";
```

```
code_map["cgti"]="01001100";
code_map["br"]="001100100";
code_map["bra"]="001100000";
code_map["brsl"]="001100110";
code_map["brasl"]="001100010";
code_map["bi"]="00110101000";
code_map["bisl"]="00110101001";
code_map["brnz"]="001000010";
code_map["brz"]="001000000";
code_map["brhnz"]="001000110";
code_map["brhz"]="001000100";
code_map["fa"]="01011000100";
code_map["fs"]="01011000101";
code_map["fm"]="01011000110";
code_map["fma"]="1110";
code_map["dfceq"]="01111000011";
code_map["dfcgt"]="01011000011";
code_map["cntb"]="01010110100";
code_map["avgb"]="00011010011";
code_map["stop"]="00000000000";
code_map["inop"]="00000000001";
code_map["nop"]="01000000001";

int error_line=0;

//read each line of the input file
while(getline(infile,line)!=0)
{
	vector<int> split_index;
	int size=line.size();
	string binary;
	split_index.push_back(-1);
	// get the split index
	for (int i=0;i<size;i++)
	{
		if (line[i]==' ' || line[i]==',' || line[i]=='(' || line[i]==')')
		{
			split_index.push_back(i);
		}
	}
	split_index.push_back(size);
	cout<<line<<endl;
	vector<string> operand;
	for (int i=0;i<split_index.size()-1;i++)
```

```
        {
            string temp(line,split_index[i]+1,split_index[i+1]-split_index[i]-1);
            operand.push_back(temp);            }
        map<string,string>::iterator iter;
        iter=code_map.find(operand[0]);
        if (iter==code_map.end())
        {
            binary="00000000000000000000000000000000";
            cout<<binary<<endl;
            outfile<<binary<<endl;
            error_line++;
            continue;
        }
        if (operand[0]=="lqd" || operand[0]=="stqd")
        {
            binary+=code_map[operand[0]];
            binary+=extend(operand[2])+extend(operand[3])+extend(operand[1]);

        }
        else if(operand[0]=="br" || operand[0]=="bra")
        {
            int num=stoi(operand[1]);
            std::bitset<16> a(num);
            ostringstream stream;
            stream<<a;
            string s=stream.str();
            binary+=code_map[operand[0]]+s+"0000000";
        }
        else if(operand[0]=="brsl" || operand[0]=="brasl" || operand[0]=="brnz" ||
operand[0]=="brz" || operand[0]=="brhnz" || operand[0]=="brhz")
        {
            int num=stoi(operand[2]);
            std::bitset<16> a(num);
            ostringstream stream;
            stream<<a;
            string s=stream.str();
            binary+=code_map[operand[0]]+s+extend(operand[1]);
        }
        else if(operand[0]=="bi")
        {
            binary+=code_map[operand[0]]+"0000000"+extend(operand[1])+"0000000";
        }
        else if(operand[0]=="bisl")
        {
```

```
        binary+=code_map[operand[0]]+"0000000"+extend(operand[2])+extend(operand[1]);
            }
            else if(operand[0]=="fma")
            {

        binary+=code_map[operand[0]]+extend(operand[1])+extend(operand[3])+extend(operand[2]
)+extend(operand[4]);
            }
            else if(operand[0]=="cntb")
            {

        binary+=code_map[operand[0]]+"0000000"+extend(operand[2])+extend(operand[1]);
            }
            else if(operand[0]=="stop" || operand[0]=="nop" || operand[0]=="lnop")
            {
                binary+=code_map[operand[0]]+"00000000000000000000";
            }
            else if(operand[0]=="rotqbyi" || operand[0]=="shlqbii" || operand[0]=="cbd")
            {
                int num=stoi(operand[3]);
                std::bitset<7> a(num);
                ostringstream stream;
                stream<<a;
                string s=stream.str();
                binary+=code_map[operand[0]]+s+extend(operand[2])+extend(operand[1]);
            }
            else
            {
                binary+=code_map[operand[0]];
                for (int i=operand.size()-1;i>0;i--)
                {
                    binary+=extend(operand[i]);
                }
            }
            cout<<binary<<endl;
            outfile<<binary<<endl;
        }
        cout<<"error_line: "<<error_line<<endl;
        return 0;
}
```