VLSI System Design (Graduate Level)
Fall 2020

HOMEWORK I
REPORT

Must do self-checking before submission:
☑ Compress all files described in the problem into one tar
☑ All SystemVerilog files can be compiled under SoC Lab environment
☑ All port declarations comply with I/O port specifications
☑ Organize files according to File Hierarchy Requirement
☑ No any waveform files in deliverables

Student name: 杜冠勳
Student ID: N26094883

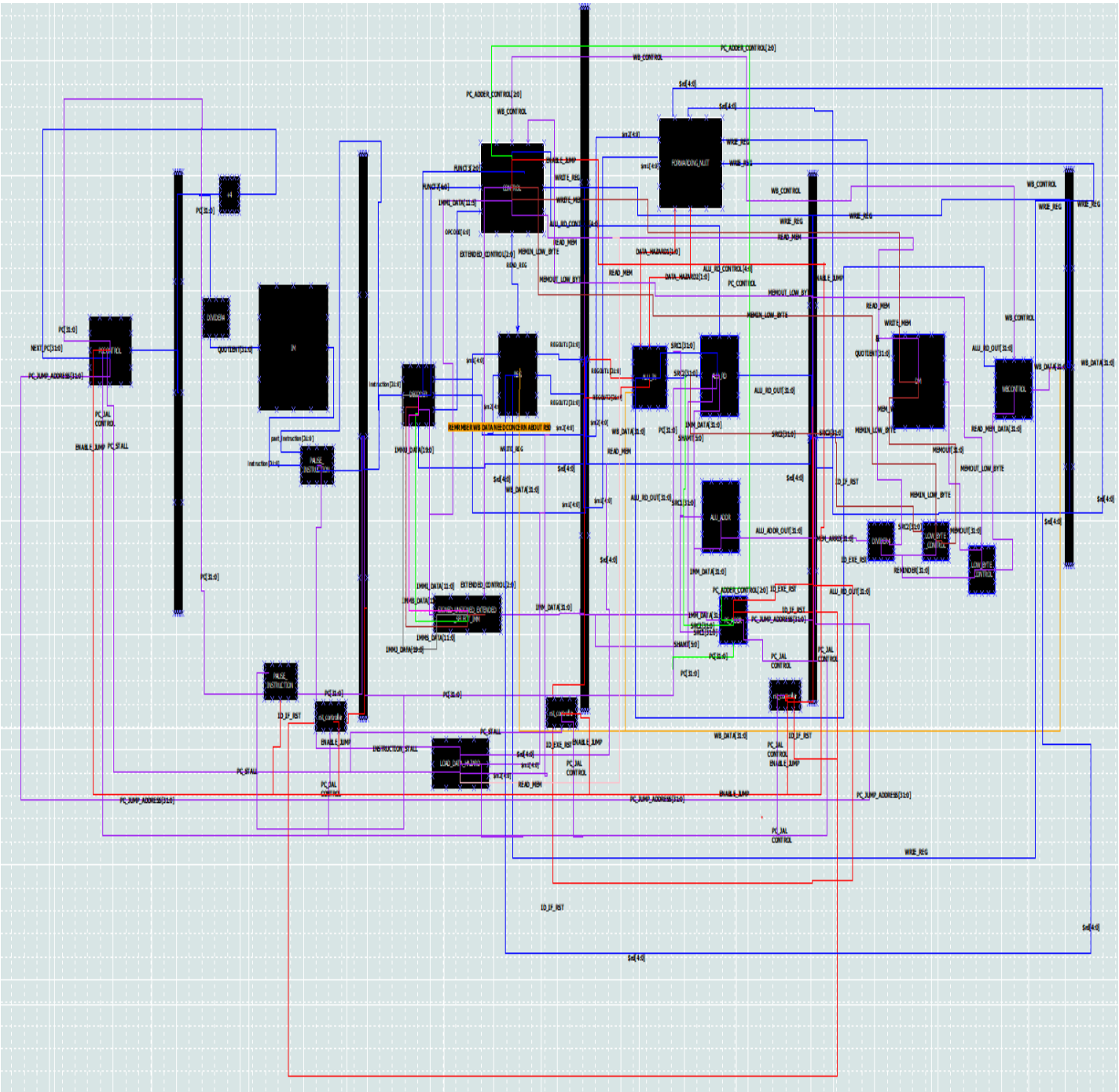
SUMMARY		
Requirements	**implement a simplified pipeline CPU with the following features** **1. Implement the 33 instructions** **2. The number of pipeline stage is 5.** **3. Register File size: 32x32-bit ➔ x0 is read only 0.** **4. Instruction memory size: 16Kx32-bit ☐ Data memory size: 16Kx32-bit** **5. Timescale: 1ns/10ps** **6. Maximum Clock period: 20ns (50MHz)**	
Verification	**PROG 0**	**Verification for 33 instruction**
	PROG 1	**Implentment Bubblesort algorithm for sorting**
	PROG 2	**Implentment Multiplication**
	PROG 3	**Implentment Greatest common divisor**

INTRODUCTION FOR DESIGN OF THE CPU

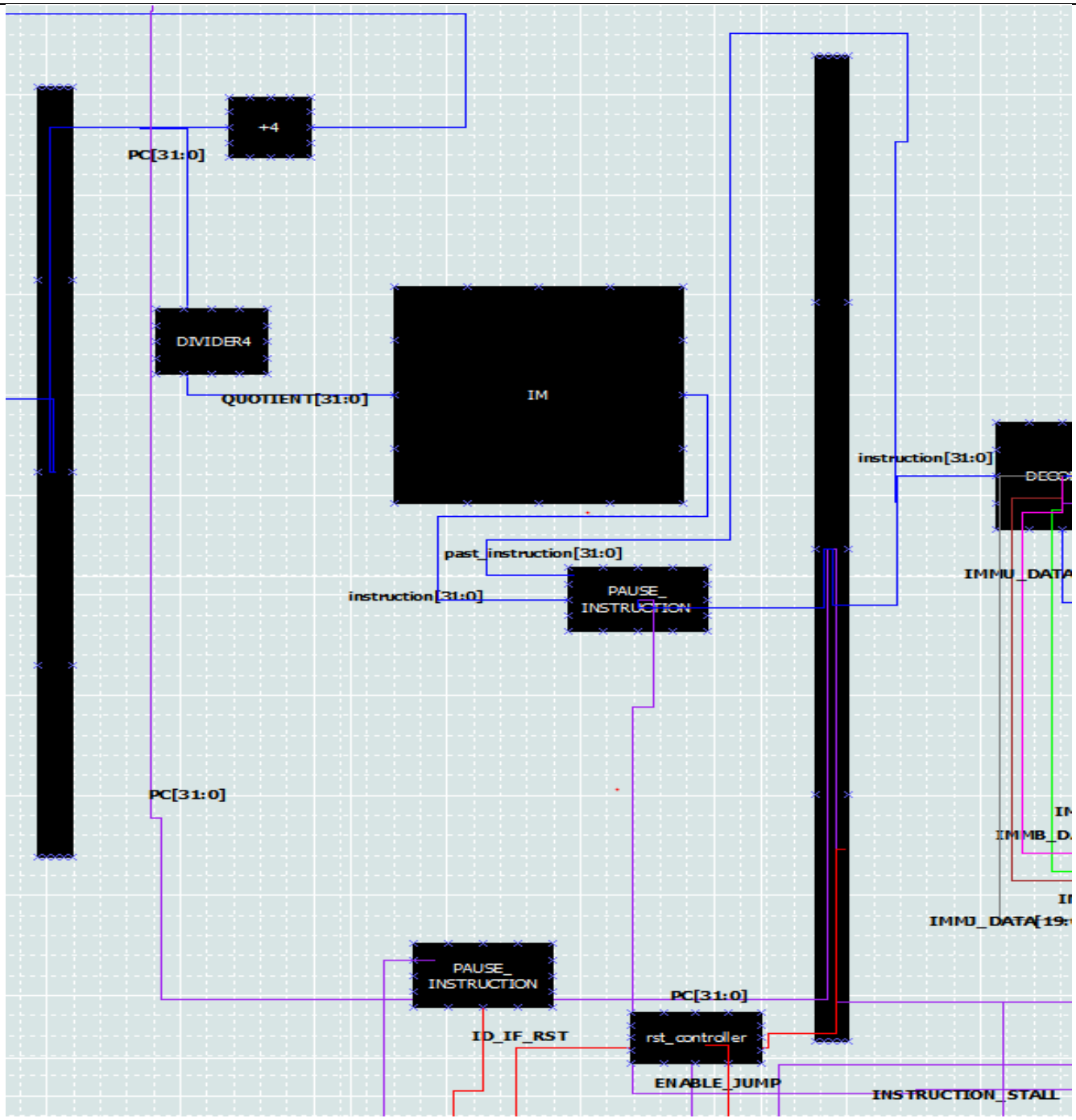
SPEC

PIPELINE STAGE	**5**
INSTRUCTION	**33**

FULL GRAPH

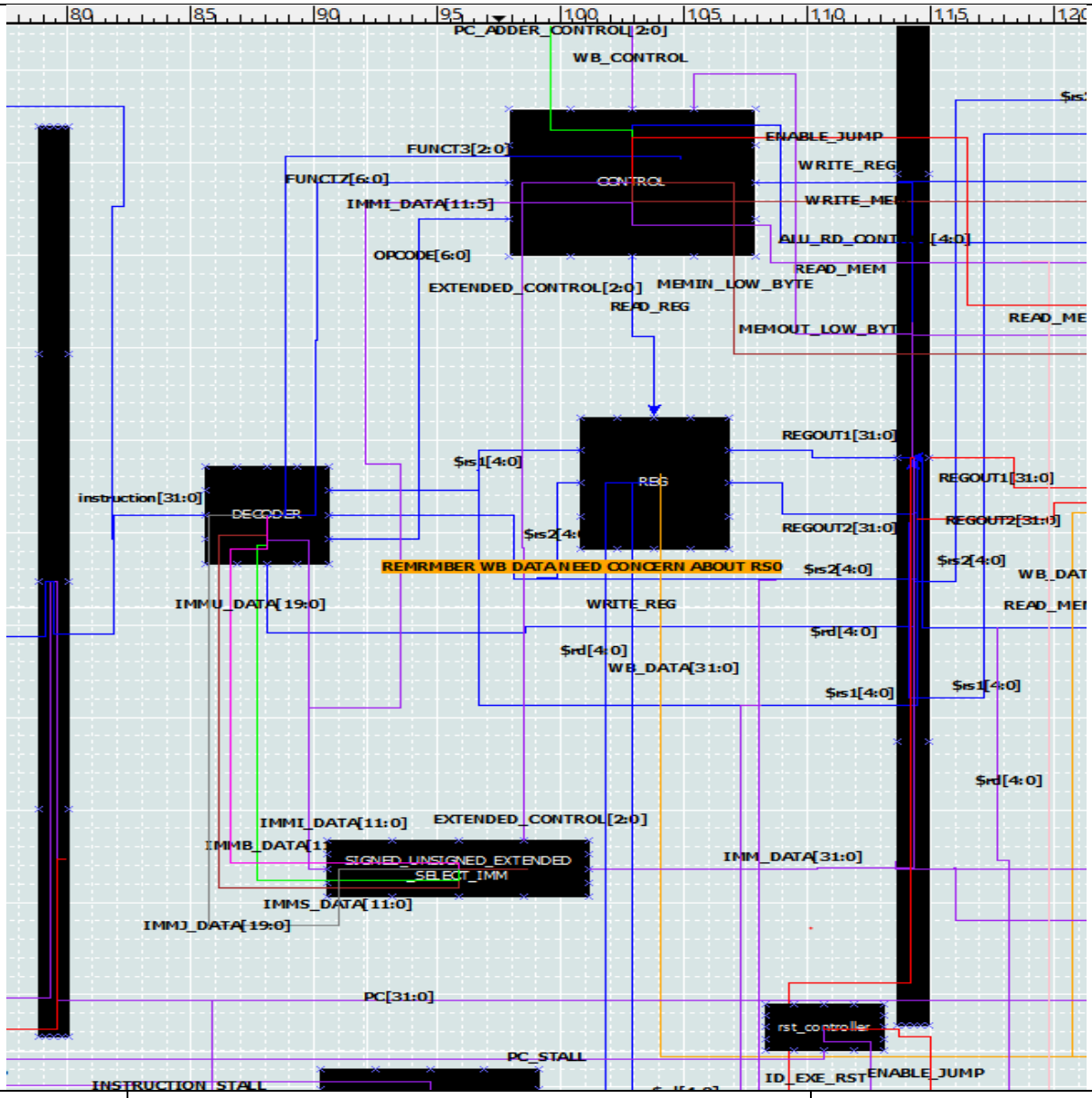


STAGE1 INSTRUCTION FETCH



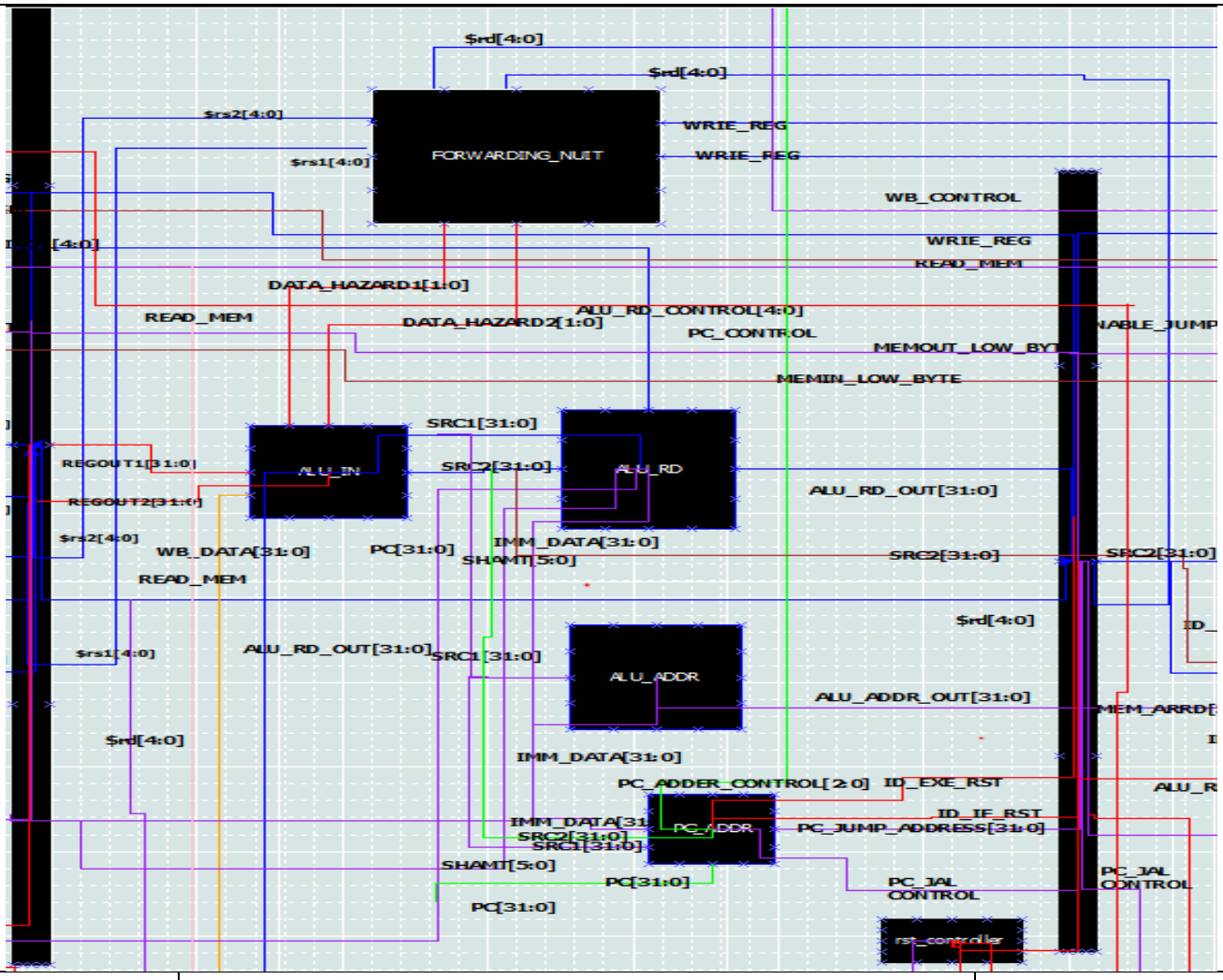
MODULE	FUNCTION	SPECIAL_DESIGN
DIDER4	Get the Instruction Address by dividing the pc_counter result by4	Using shift right 2 bits instead using the divider which can reduce hardware area.
PAUSE INSTRUCTION	If there is any load use data hazard happen,this module will enable to pause the instruction. The instruction an d pc value will be rewrited tio the register in next clock.	NONE
rst_contoller	If there is any branch happen this module will enable flush the register. The module will be more detailed discuss later.	NONE

STAGE2 INSTRUCTION DECODE & STAGE5 WRITE BACK



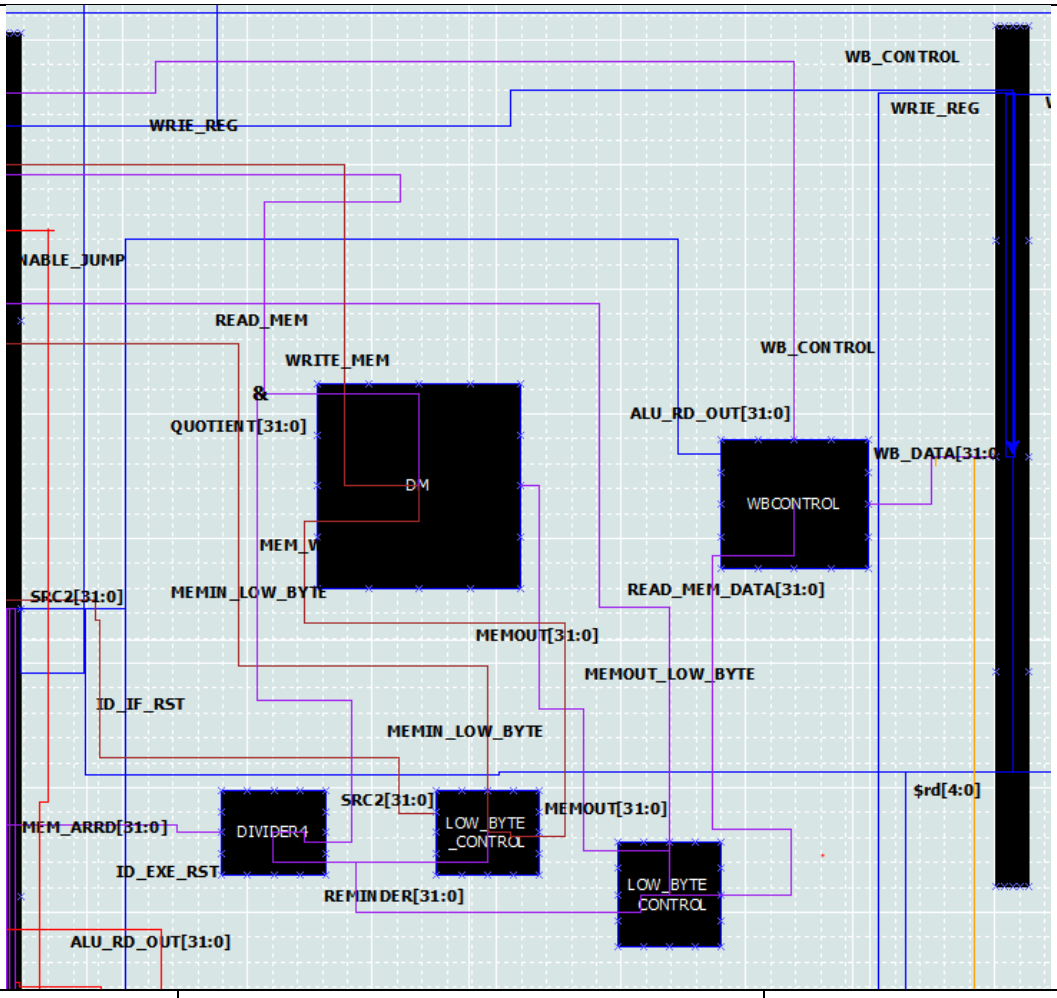
MODULE	FUNCTION	SPECIAL_DESIGN
DECODER	Decoding 33 bits instruction into different elements which is used for the following module	NONE
SIGNED UNSIGNED EXTENDED_ SELECT_IMM	Make the Immediate data referring to the type of the instruction	NONE
REG	32×32-bit. registers	Posedge clk write back Negedge clk read data
CONTROL	Make the Control signal	NONE
rst_contoller	If there is any branch or load use data hazard happen this module will enable flush the register. The module will be more detailed discuss later.	NONE

STAGE3 ALU EXECUTION



MODULE	FUNCTION	SPECIAL_DESIGN
ALU_IN	Data hazard signal will be the input module will decide the input of the alu.	NONE
ALU_RD	calculate for the data store in $rd	NONE
ALU_ADDR	calculate for the Data memory's Address which will be stored data next clock.	NONE
PC_ADDR	calculate the jumping address and make the signal to enable the system jump or not.	NONE
rst_contoller	If there is any branch happen this module will enable flush the register. The module will be more detailed discuss later.	NONE

STAGE4 WRITE OR READ MEMORY

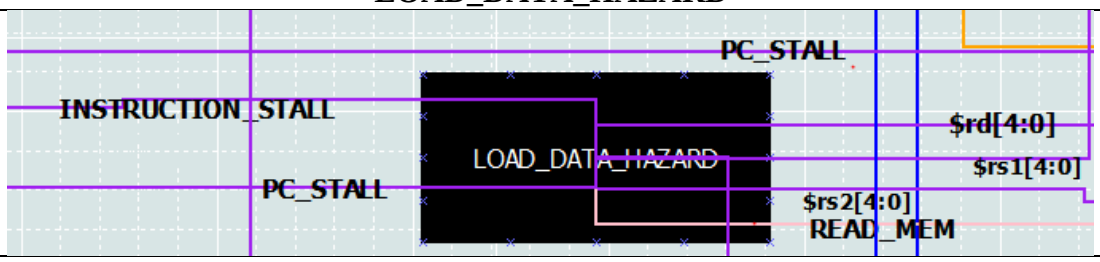


MODULE	FUNCTION	SPECIAL_DESIGN
DIVIDER4	Divide the address value by 4 Quotient will be the real address value of data memory Reminder will be used for LB SB instruction to load and store data to the corresponding byte.	Using shift right 2 bits instead using the divider which can reduce hardware area.
LOW_BYTE_CONTROL	Control for LB SB instruction to load and store data to the corresponding byte.	NONE
WB_CONTROL	Wb_control signal will decide data from memrory and alu writing back to rd rigister	NONE

OTHER CONTROL MODULE

MODULE

LOAD_DATA_HAZARD

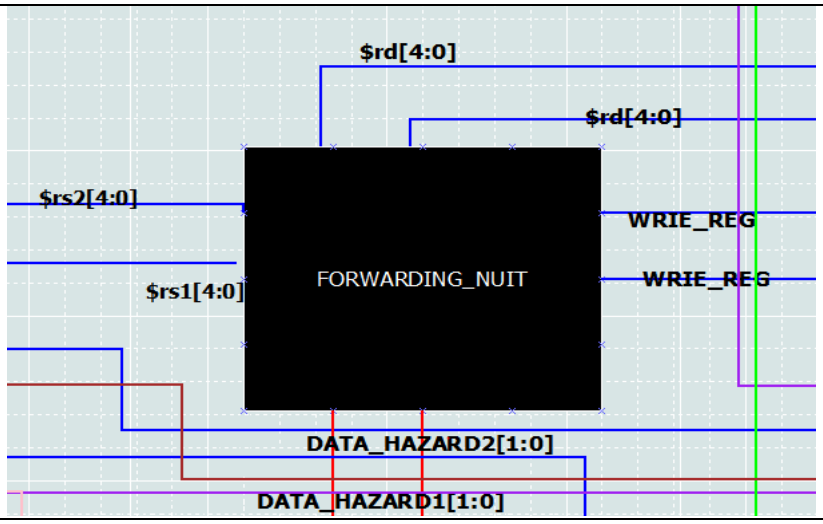


CONTROL FLOW

Detect if there is any load use data hazard
pc_stall_stage1=
(id_exe_read_mem==1'b1 && (if_id_rs1_addr==id_exe_rd_addr || if_id_rs2_addr==id_exe_rd_addr))?1'b1:1'b0;
Detect if there is any jump signal happened at the same time and prevent pc_stall signal flush the jump action.
Jump signal has the priority
pc_st all=pc_jump_confirm?1'b0:pc_stall_stage1;
instruction_stall=pc_stall;

MODULE

FORWARDING UNIT



CONTROL FLOW

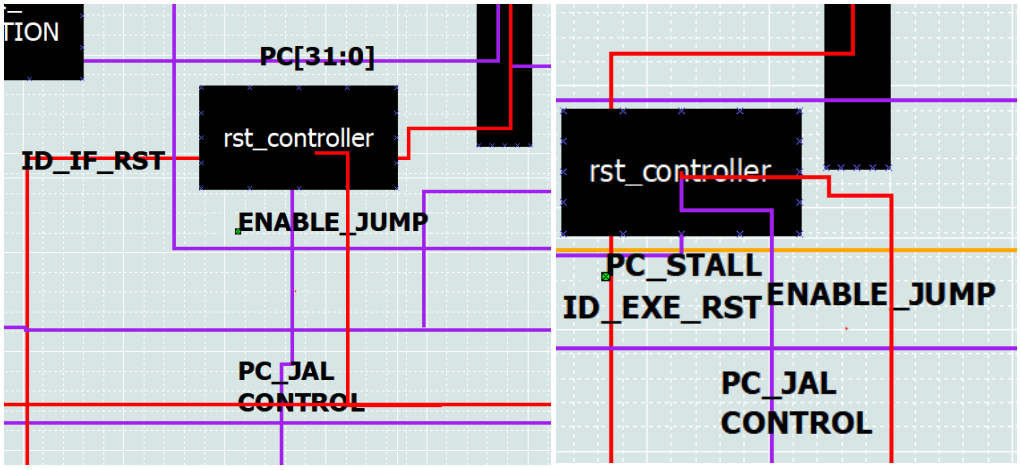
Detect if there is any data hazard between memory and register $rs1

rs1_mem_hazard=(mem_wb_write_reg==1'b1 && mem_wb_rd_addr!=5'd0 && mem_wb_rd_addr==rs1_addr)?1'b1:1'b0;
Detect if there is any data hazard between ALU and register $rs1

rs1_exe_hazard=(exe_mem_write_reg==1'b1 && exe_mem_rd_addr!=5'd0 && exe_mem_rd_addr==rs1_addr)?1'b1:1'b0;
Detect if there is any data hazard between memory and register $rs2

rs2_mem_hazard=(mem_wb_write_reg==1'b1 && mem_wb_rd_addr!=5'd0 && mem_wb_rd_addr==rs2_addr)?1'b1:1'b0;
Detect if there is any data hazard between ALU and register $rs2

rs2_exe_hazard=(exe_mem_write_reg==1'b1 && exe_mem_rd_addr!=5'd0 && exe_mem_rd_addr==rs2_addr)?1'b1:1'b0;

MODULE	**rst_controller** **left side (for exe_mem stage and if _id stage)** **right side (id_exe stage)**
	

CONTROL FLOW

(for exe_mem stage and if _id stage)

Global rst is cpu's rst ; jump and branch instruction will make [enable jump signal] active high;

if brach conditiction is established then pc_jump control will active high enable to jump address.

rst_data=global_rst?1'b1:(enable_jump ? (pc_jump_control ? local_rst:1'b0) :1'b0);

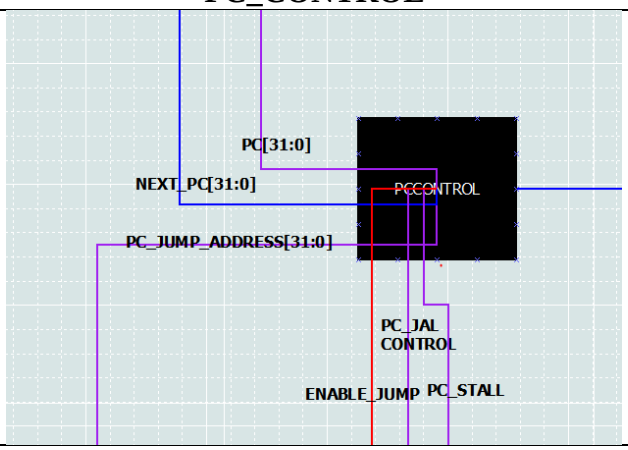
(for id_exe stage)

1.Global rst is cpu's rst ;

2.Modele will check the pc_stall signal to decide whether pause the pipeline register or not.

Next, jump and branch instruction will make [enable jump signal] active high; if branch condition is established then pc_jump control will active high enable to jump address.

rst_data=global_rst?1'b1: (pc_stall ? 1'b1: (enable_jump ? (pc_jump_control ? local_rst:1'b0) :1'b0));

MODULE	PC_CONTROL
	

CONTROL FLOW

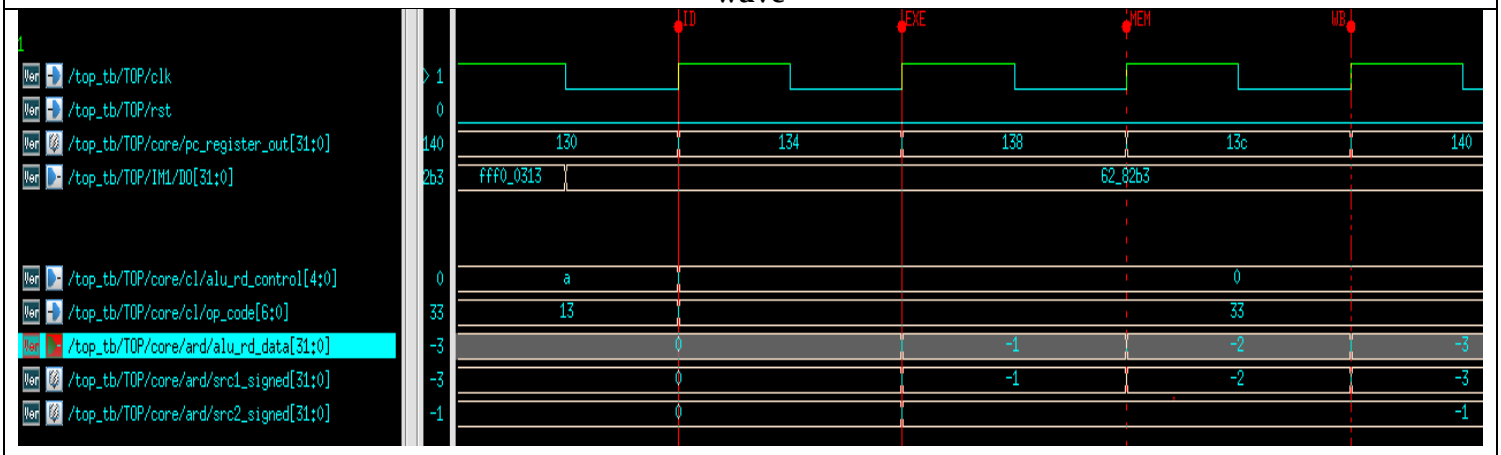
pc_stall signal will enable rewrite the instruction address again

enable_jump signal will enable to jump address.

```
if(pc_stall==1'b1)

      pc_data=pc;

else

   pc_data=enable_jump?(pc_jump_control?pc_jump_address:next_pc):next_pc;
```

R-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
ADD	**0x006282b3**	**add t0 t0 t1**

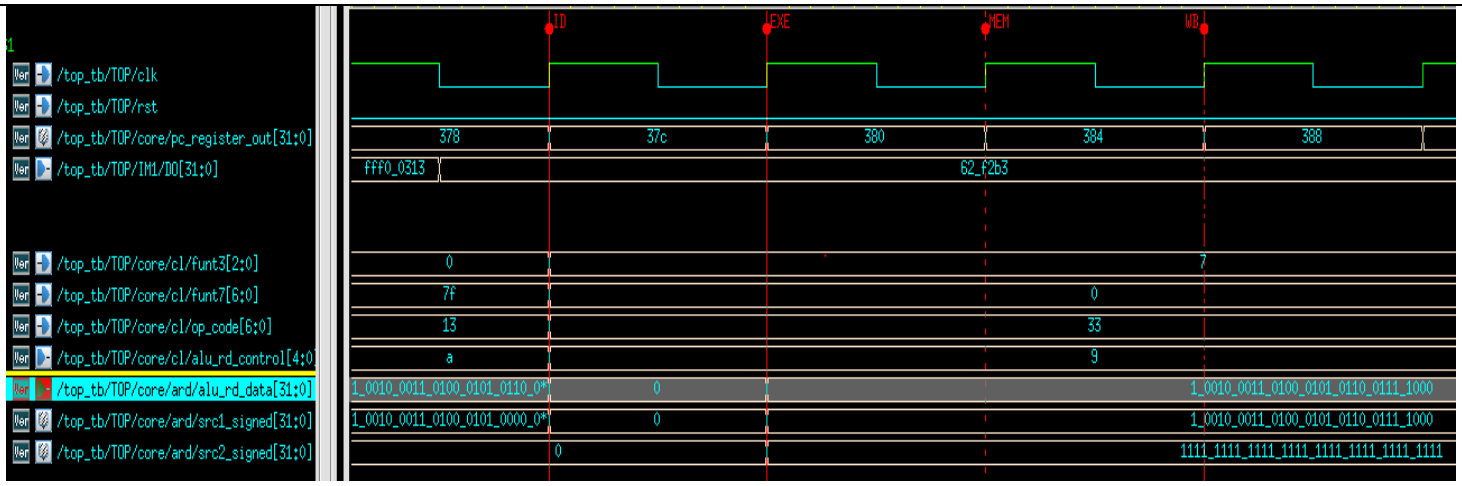
wave



Explanation	
ID stage	**control module(cl) produce alu_rd_signal(function of the alu);active write_reg signal; and register(rigt)read the data by the address decode by decoder** **Decoder decode opcode:33**
EXE stage	**Alu_in_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu** **Alu_rd (ard)calculate the value that will store to the $rd address**
MEM stage	**By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register**

R-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION AND	**Machine code**	**Assembly code**
	0x0062f2b3	**and t0 t0 t1**

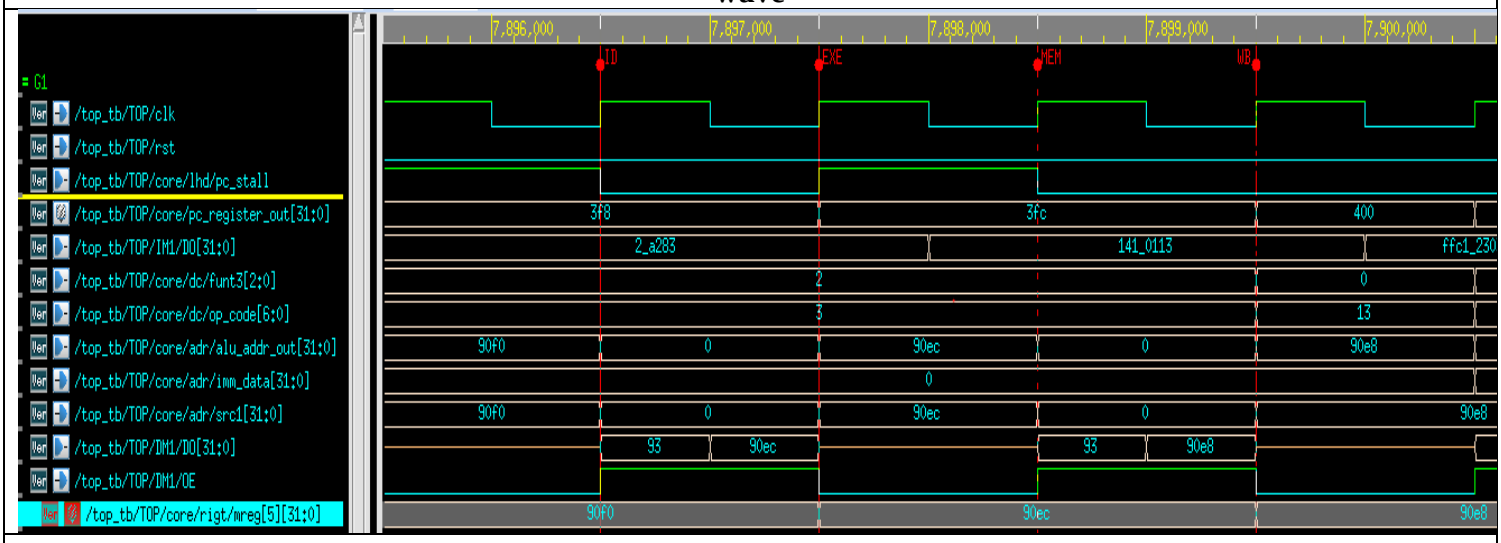
wave



Explanation	
ID stage	**control module(cl) produce alu_rd_signal(function of the alu);active write_reg signal; and register(rigt)read the data by the address decode by decoder** **Decoder decode opcode:33,funt3:7,fun7:0**
EXE stage	**Alu_in_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu** **Alu_rd_control signal=9 execute and function** **Alu_rd (ard)calculate the value that will store to the $rd address**
MEM stage	**By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register**

I-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
LW	**0x0002a283**	**Lw t0 0(t0)**

wave

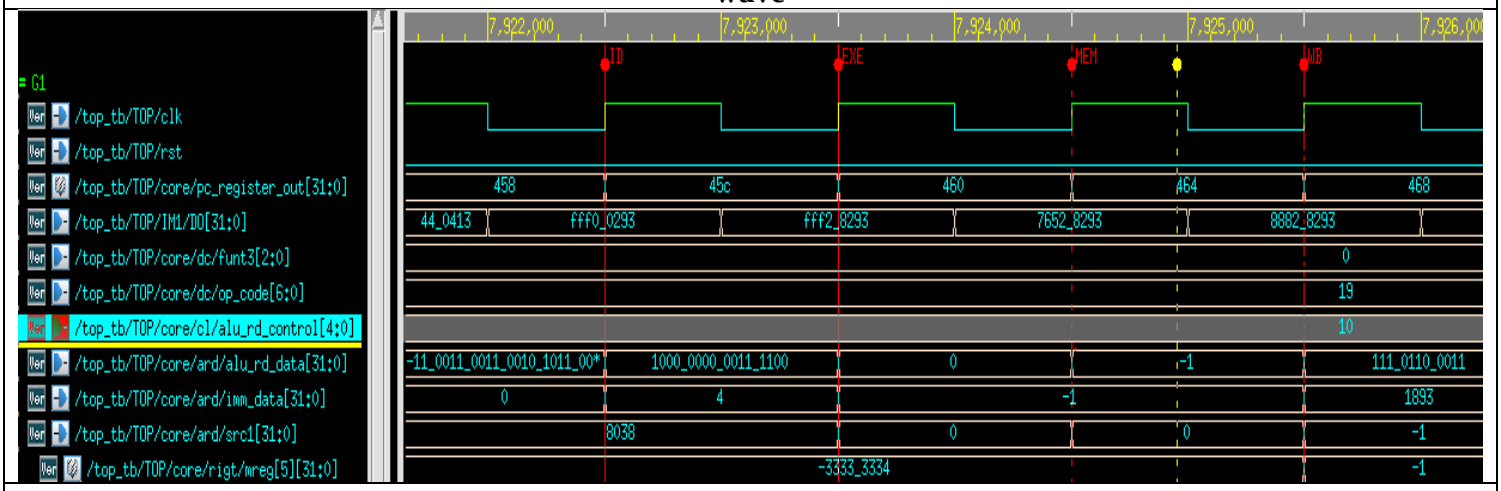


Explanation

ID stage	**Decoder decode opcode:3,funt3:2** **Control module'output extended_control control the imm data** **Control module produce read mem signal**
EXE stage	**Alu_addr calculate the address(alu_addr_out) that is going to load data.** **Because thers is a load data hazard ,the pc_stall signal active high to pause instruction.**
MEM stage	**The address is divided by 4 to be the real data memory address** **The OE signal is active high because it connects with read mem signal** **Wb_control control the value loaded from memory to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register** **The valus is stored in register in 5 stage**

I-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
ADDI	**0xfff00293**	**addi t0 t0 -1**

wave

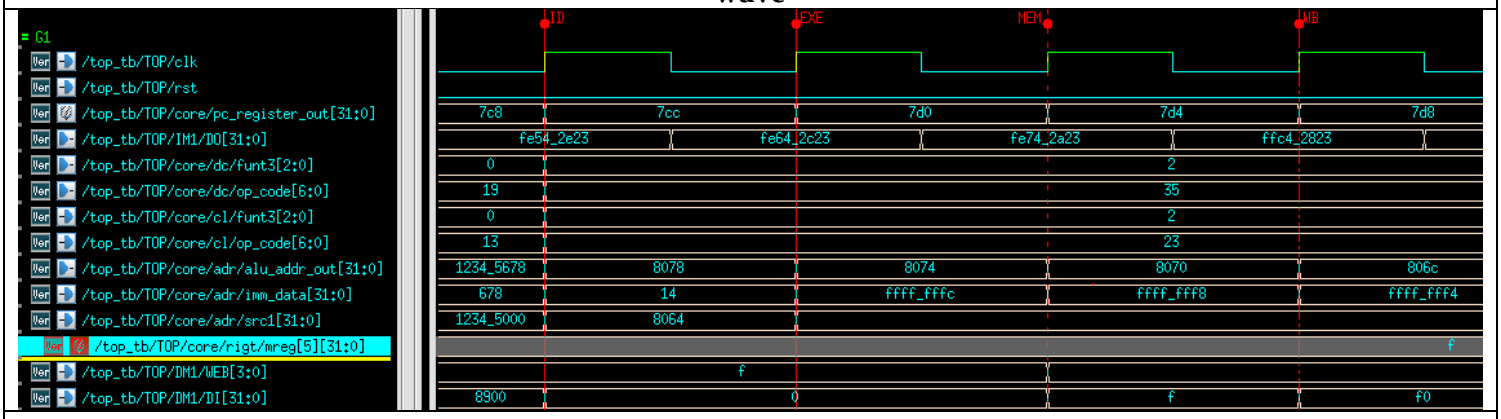


Explanation

ID stage	**Decoder decode opcode:19funt3:0,alu_control:10** **Control module'output extended_control control the imm data**
EXE stage	**Alu_addr calculate the address(alu_addr_out) that is going to load data.** **The calculation way is selected by alu_control**
MEM stage	**By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register** **The valus is stored in register in 5 stage**

S-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	Machine code	Assembly code
SW	0xfe542e23	Sw t0,-4(s0)

wave

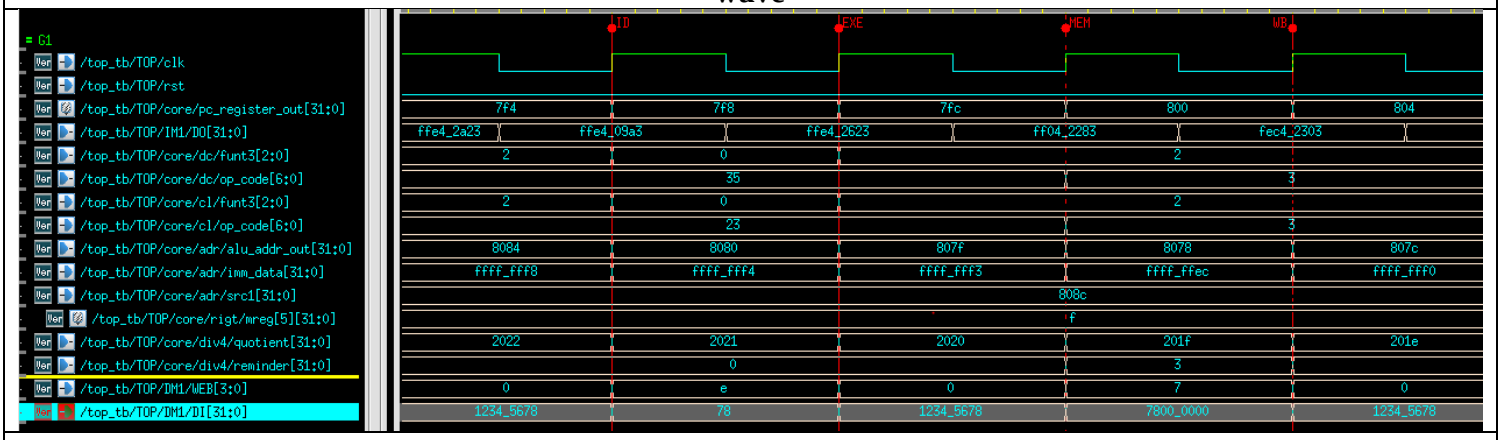


Explanation

ID stage	Decoder decode opcode:35funt3:2 Control module'output extended_control control the imm data Control module activate the write mem signal
EXE stage	Alu_rd (ard)calculate the value that will store to the $rd address The calculation way is selected by alu_control
MEM stage	The address is divided by 4 to be the real data memory address The 4 bits signal is active low because it connects with read mem signal The data from register is stored in negedge cycle
WB stage	NO ACTION

S-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
SB	**0xfe542e23**	**Sw t0,-4(s0)**

wave

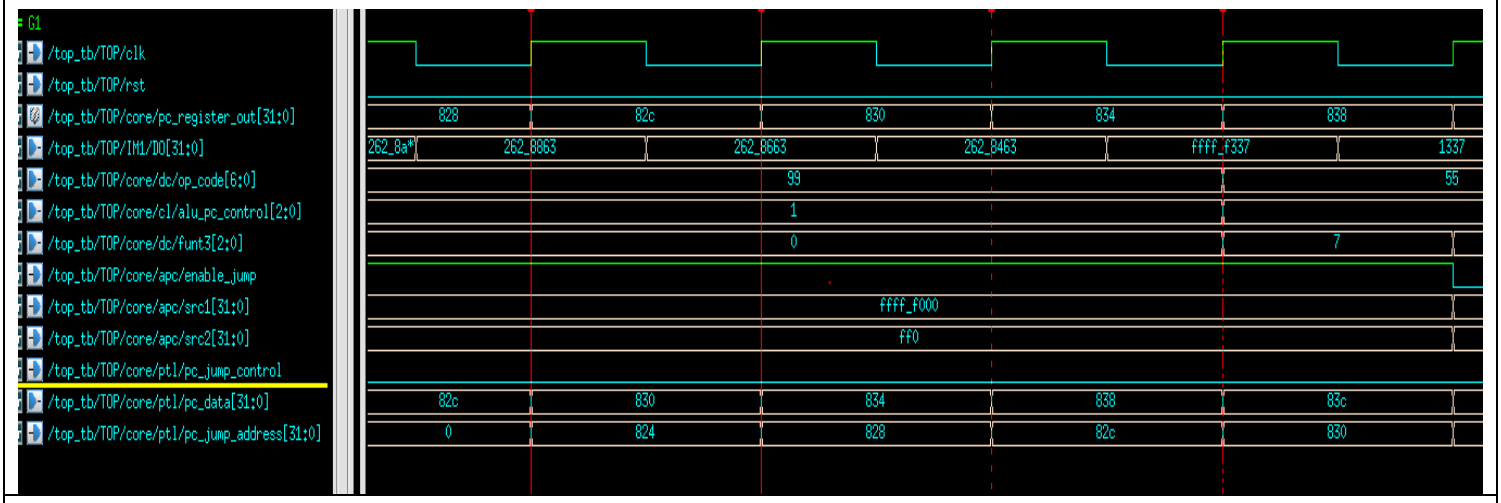


Explanation

ID stage	**Decoder decode opcode:35funt3:0** **Control module'output extended_control control the imm data** **Control module activate the write mem signal** **Control module activate the low byte write signal**
EXE stage	**Alu_rd (ard)calculate the value that will store to the $rd address** **The calculation way is selected by alu_control**
MEM stage	**The address is divided by 4 to be the real data memory address and the written bytes will selected by reminder.** **The 4 bits signal is 1110 because the low byte control signal only allow write in the low 8bits data.** **The data from register is stored in negedge cycle**
WB stage	**NO ACTION**

B-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	Machine code	Assembly code
BEQ	0x02628863	Beq t0,t1 858

wave

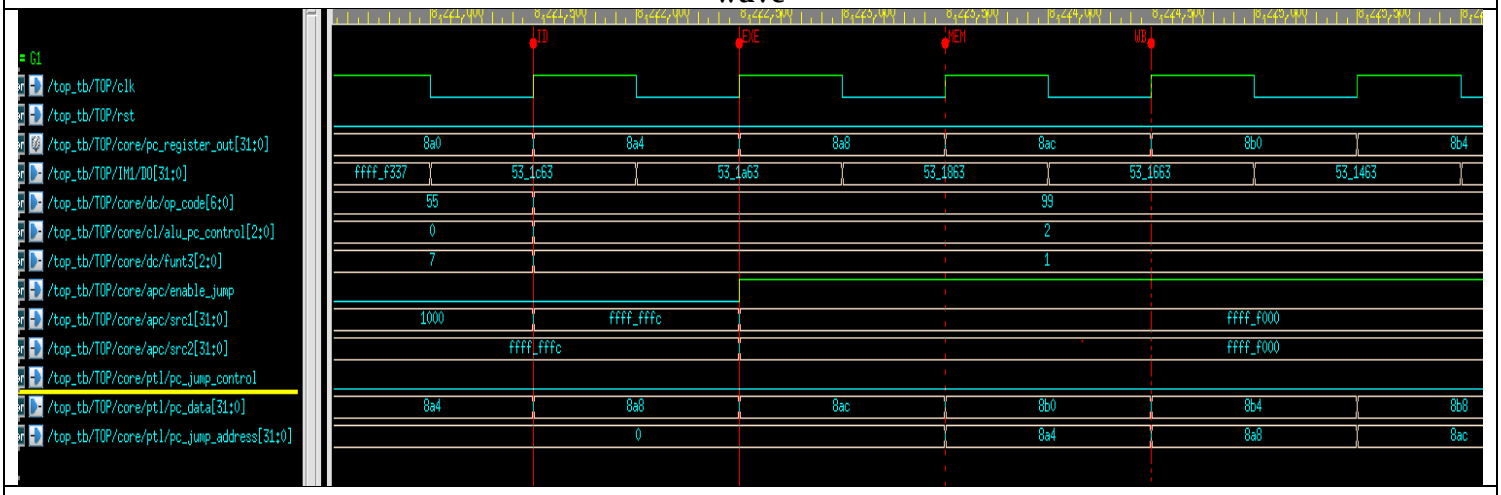


Explanation

ID stage	**Decoder decode opcode:99funt3:0** **Control module'output extended_control control the imm data** **Control module activate the enable jump signal** **Control module control the alu_pc function with alu pc control signal=1**
EXE stage	**Alu_PC determine whether to jump or not. src1!=src2 in this section which will not activate jump signal** **The next pc address isn't jump**
MEM stage	**NO ACTION**
WB stage	**NO ACTION**

B-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
BNE	**0x 00531c63**	**Bne t1,t0,8b8**

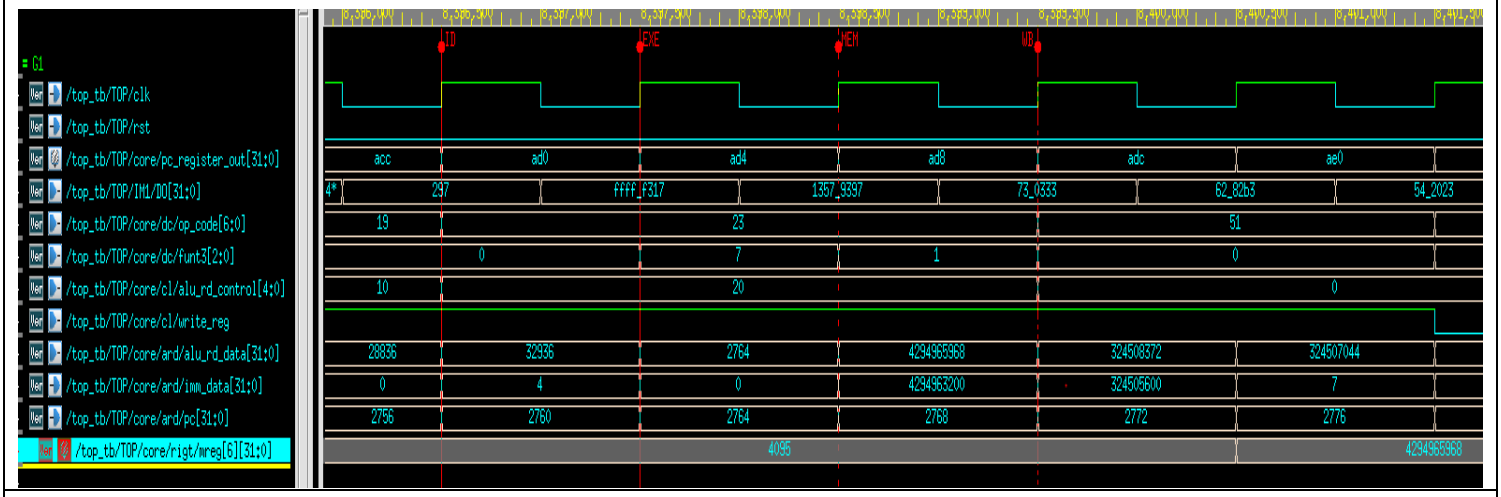
wave



Explanation	
ID stage	**Decoder decode opcode:99funt3:1** **Control module'output extended_control control the imm data** **Control module activate the enable jump signal** **Control module control the alu_pc function with alu pc control signal=2**
EXE stage	**Alu_PC determine whether to jump or not. src1==src2 in this section which will not activate jump signal** **The next pc address isn't jump**
MEM stage	**NO ACTION**
WB stage	**NO ACTION**

U-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
AUIPC	**0x 00000297**	**Auipc t0 0x0**

wave

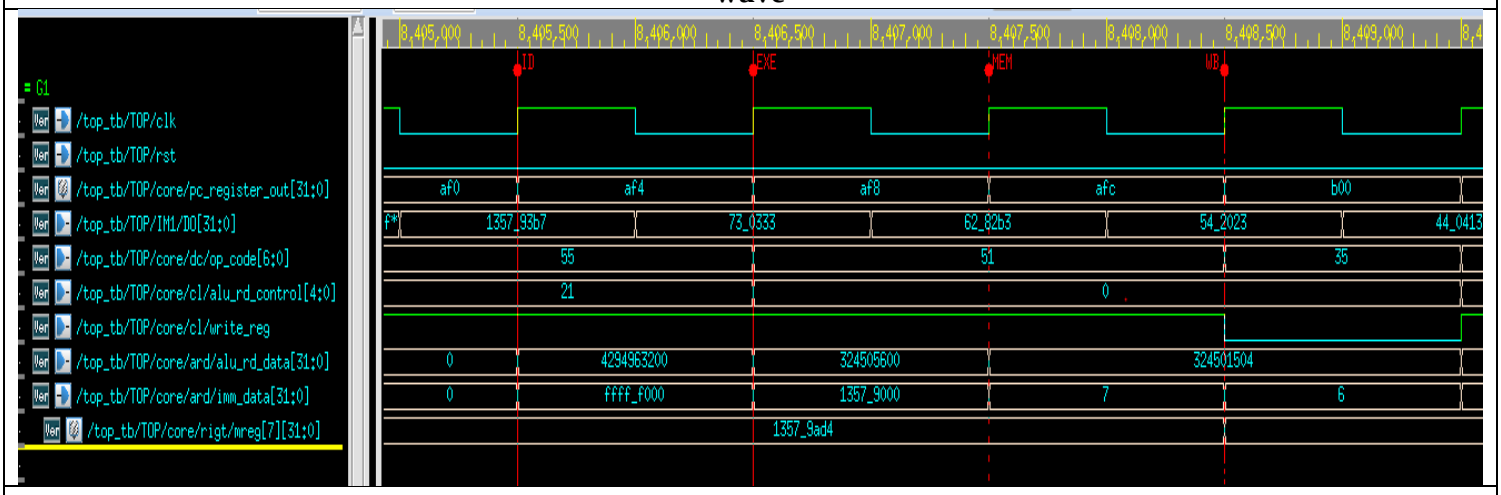


Explanation

ID stage	**Decoder decode opcode:23** **Control module'output extended_control control the imm data** **Control module enable the write reg signal** **Control module control the alu_ed function with alu rd control signal=20**
EXE stage	**Alu_in_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu** **Alu_rd_control signal=9 execute and function** **Alu_rd (ard)calculate the value that will store to the $rd address**
MEM stage	**By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register** **The valus is stored in register in 5 stage**

U-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
LUI	**0x 135793b7**	**Lui t2 0x13579**

wave

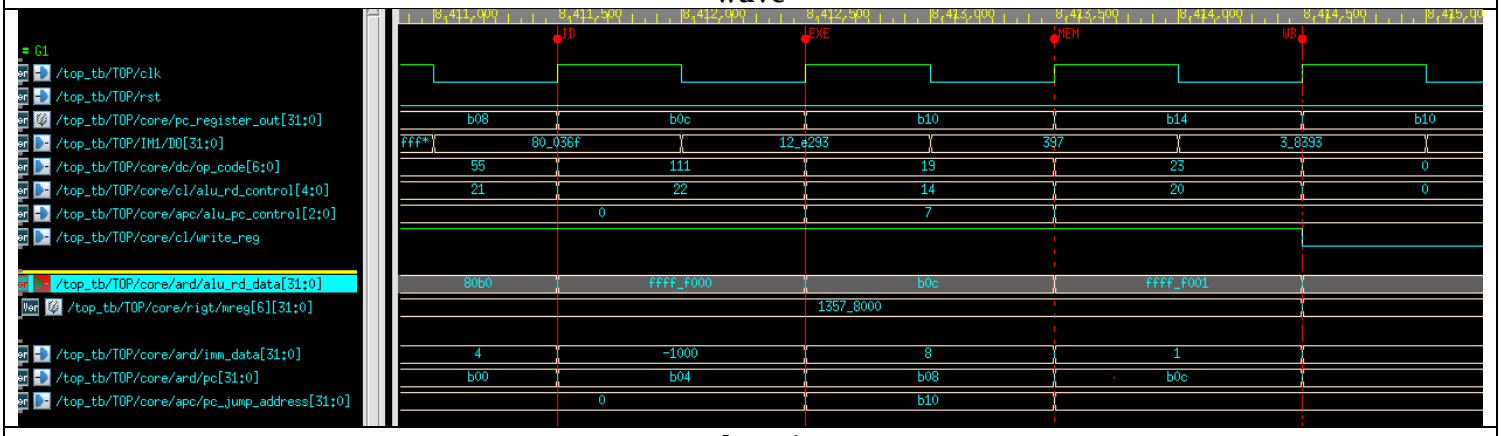


Explanation

ID stage	**Decoder decode opcode:55** **Control module'output extended_control control the imm data** **Control module activate the write reg signal** **Control module control the alu_rd function with alu rd control signal=21**
EXE stage	**Alu_in_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu** **Alu_rd_control signal 21 execute and function** **Alu_rd (ard)calculate the value that will store to the $rd address**
MEM stage	**By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register** **The valus is stored in register in 5 stage**

J-TYPE INSTRUCTION VERIFICATION		
INSTRUCTION	**Machine code**	**Assembly code**
JAL	**0x0080036f**	**Jal t1,b10**

wave



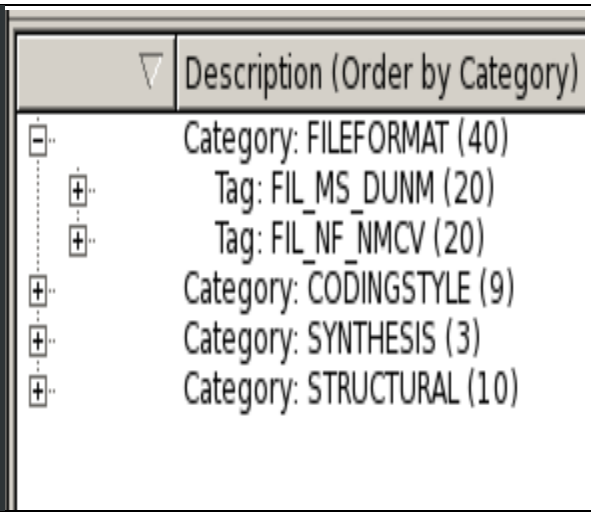
Explanation

ID stage	**Decoder decode opcode:111funt3** **Control module'output extended_control control the imm data** **Control module activate the write reg signal Control module control the alu_rd function with alu rd control signal=14** **Control module control the alu_rd function with alu pc control signal=7** **Control module activate the enable jump signal**
EXE stage	**Alu_in_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu** **Alu_rd_control signal=14 execute and function** **Alu_rd (ard)calculate the value that will store to the $rd address**
MEM stage	**Alu_PC determine to jump to the address pc+immin the next cylce** **By pass the value calculate by Alu_rd** **Wb_control control the value calculate by Alu_rd to be the write back data.**
WB stage	**Write_reg signal enable to write back data in register** **The valus is stored in register in 5 stage**

SUPER LINT ANALYZE

TOTAL LINES:1943

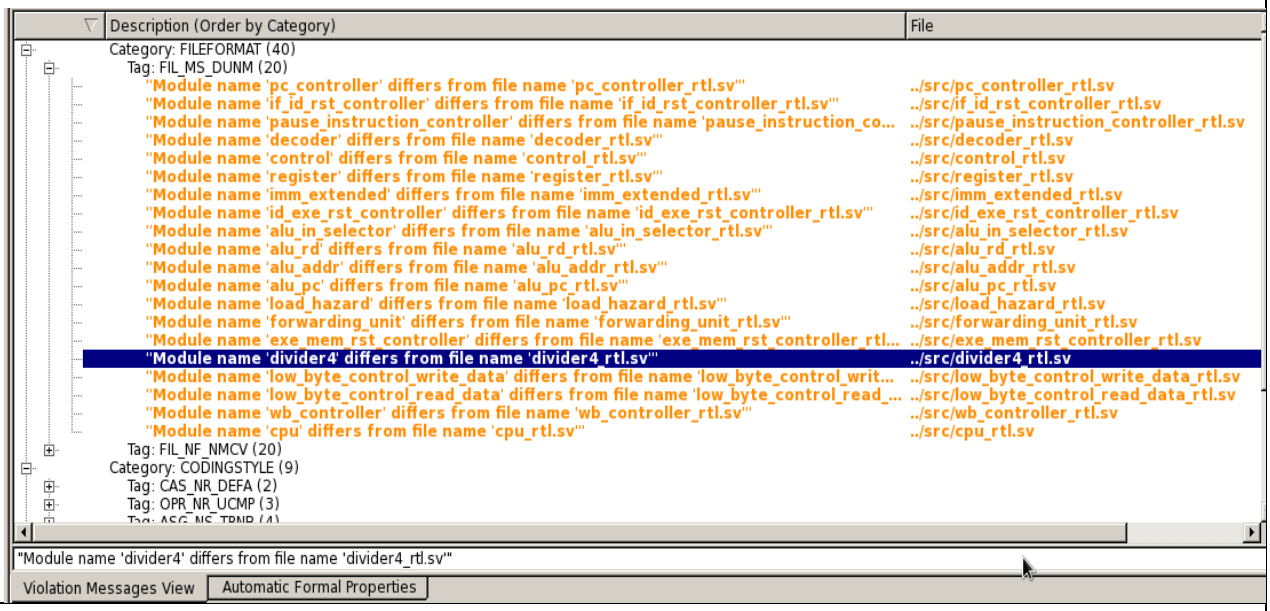
```
[khduh@ncku src]$ wc -l *.sv
   20 alu_addr_rtl.sv
   77 alu_in_selector_rtl.sv
  123 alu_pc_rtl.sv
  132 alu_rd_rtl.sv
  312 control_rtl.sv
  472 cpu_rtl.sv
   50 decoder_rtl.sv
   19 divider4_rtl.sv
   21 exe_mem_rst_controller_rtl.sv
   34 forwarding_unit_rtl.sv
   25 id_exe_rst_controller_rtl.sv
   22 if_id_rst_controller_rtl.sv
   65 imm_extended_rtl.sv
   57 imm_extended_tb.sv
   29 load_hazard_rtl.sv
   50 low_byte_control_read_data_rtl.sv
   63 low_byte_control_write_data_rtl.sv
   21 pause_instruction_controller_rtl.sv
   20 pause_pc_controller_rtl.sv
   38 pc_controller_rtl.sv
  111 register_rtl.sv
   99 SRAM_wrapper.sv
   61 top.sv
   22 wb_controller_rtl.sv
 1943 total
```



WARNING:62	**1-62/1943*100%=97%**

MOST FREQUENT WARNING/ERRORS IN YOUR CODE

Module name not comparing



Modify — **For example change cpu_rtl.sv to cpu_rtl**

Default statement isn't require

Modify — **If the case statement is full case delete the default case**

After fixing the problem

WARNING58	**1-58/1943*100%=98%**

LESSON I LEARNED
經由這次的作業 除了複習以前學過的計算機組織架構,也體認到實作時 考慮控制的因素會非常多,例如當我 compile 已經過 program0 卻無法過 program1 的原因在於當我執行 jump 指令時,若前一刻有 load data hazard 的出現則可能會覆蓋掉 jump 訊號 經由修改過後就可以 compile 過了,這時我之前學習計組時 沒有想到的問題,而恰巧 program0 沒有測試到 LW 接續 JUMP 指令的 pattern 導致會有 program 0 過 program 1 沒過的情形發生。此次實作 讓我對 CPU 架構更為熟悉 獲益良多