CA Group 3 Final Project
Members: b07502151孫定洋、b07611008王鐘霆、b08901169 梁正

1.Briefly describe your CPU architecture

- **CHIP:**
  CHIP is composed of several modules: PC, reg_file, Control, Sign_Extend, MUX_2_to_1, MUX_3_to_1, ALU_Control, ALU and mulDiv. Then we define the CHIP FSM to have three states:IDLE, SINGLE(take one cycle) and MULTIPLE(take 33 cycle).
- **reg_file:**
  Use five input ports(rs1_input, rs2_input, rd, writeback, RegWrite) and two output ports (rs1_output, rs2_output) to control the register file.
- **Control:** Use Op_input to decide each output signal. Output includes Branch_output, MemRead_output, MemWrite_output, MemtoReg_output, ALUOp_output, ALUSrc_output_1, ALUSrc_output_2, RegWrite_outupt and jump_select_output. Based on different opcode of instructions, outputs will be changed.
- **Sign_Extend:**
  Use opcode from instruction to determine which type the instruction is, then perform different ways of sign extension according to instruction's type.
- **MUX_2_to_1:**
  Use two inputs ports and a 1-bit select wire to determine the output.
  (used between PC+4 & PC shift left, between register and ALU and for jump decision)
- **MUX_3_to_1:**
  Use three inputs ports and a 2-bits select wire to determine the output.
  (used between memory & register writeback port)
- **ALU_Control:**
  Use 2 bits opcode from Control module and instruction to determine which ALU operation code is output, which is later used by ALU module.
- **ALU:**
  Use operation code derived from output of ALU_Control to determine which function ALU should perform. We implement ADD, SUB, AND, OR, XOR, MUL, DIV, SLTI, SRAI, SLLI. ADD is also used for AUIPC, JAL, JALR, LW, SW, and SUB is also used for BEQ. In ALU module, we include mulDiv module and use both operation code, valid, ready to tell ALU when to perform MUL or DIV.
- **mulDiv:**
  Use codes from hw2 with 2 states (AND, AVG) removed, and output port reduced from 64 to 32 bits.
- **PC:**
  implement by two muxes and one ALU. The first mux output is decided by the branch signal to choose between PC+4 and PC+imm. Then the second mux output, which is PC_nxt, is decided by jump signal to choose between jump address and the output of the first mux.

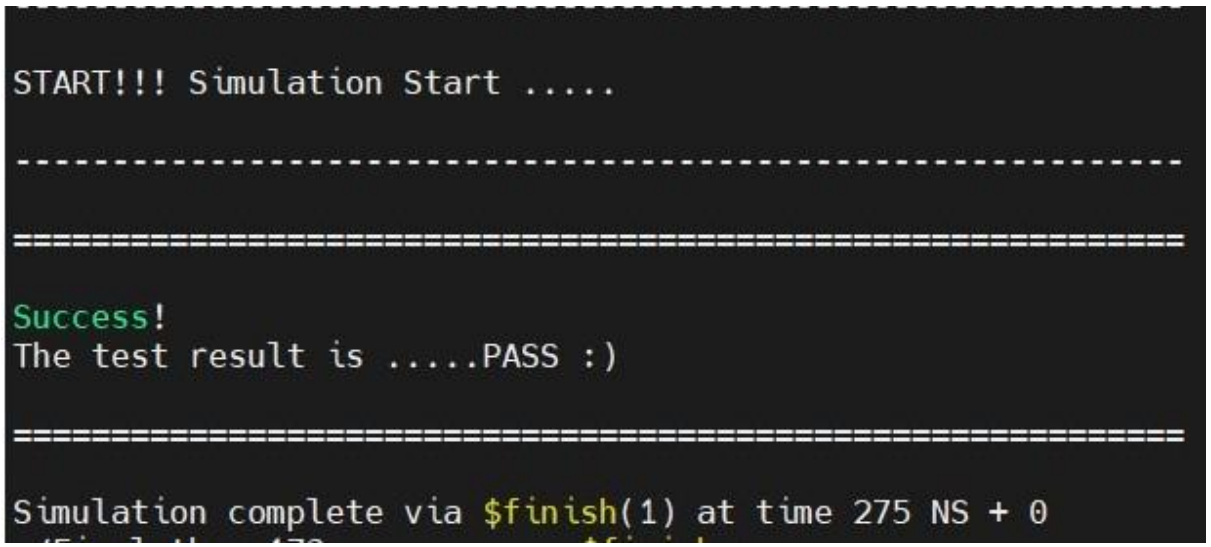
2.Describe how you design the data path of instructions not referred in the lecture slides (jal, jalr, auipc, …)

- **jal**:
  Use 2 MUX between register and ALU to select PC and offset by ALUSrc_control_1 and ALUSrc_control_2. Control module will set jump_select to 1 and ALU to do addition. The result of PC+offset is passed to the MUX_jump to update PC_nxt.
  Also, set MemtoReg to 2'b10 to select a 3-to-1 MUX which enables writing PC+4 back to rd.
- **jalr**:
  Use 2 MUX between register and ALU to select PC and imm by ALUSrc_control_1 and ALUSrc_control_2. Control module will set jump_select to 1 and ALU to do addition. The result of rs1+imm is passed to the MUX_jump to update PC_nxt.
  Also, set MemtoReg to 2'b10 to select a 3-to-1 MUX which enables writing PC+4 back to rd.
- **auipc**:
  Use 2 MUX between register and ALU to select PC and upper imm by ALUSrc_control_1 and ALUSrc_control_2. ALU will add PC and upper imm and the result is written back to rd through a 3-to-1 MUX with MemtoReg equal to 2'b0.

3.Describe how you handle multi-cycle instructions (mul)

Design a FSM for CHIP, when ALU needs to operate mul or div instructions, we will change the state to MULTIPLE state, which switch back to IDLE state after 33 cycles. Also, switch valid to 1 which enables the mulDiv module. This mulDiv FSM counts to 31, then produces its output result. At the same time, assign PC_nxt - 4 to PC_nxt to stop PC address from changing during the calculation.

4.Record total simulation time (CYCLE = 10 ns)
leaf:



perm:

```
START!!! Simulation Start .....

----------------------------------------------------------------

================================================================

Success!
The test result is .....PASS :)

================================================================

Simulation complete via $finish(1) at time 1715 NS + 0
```

bonus:

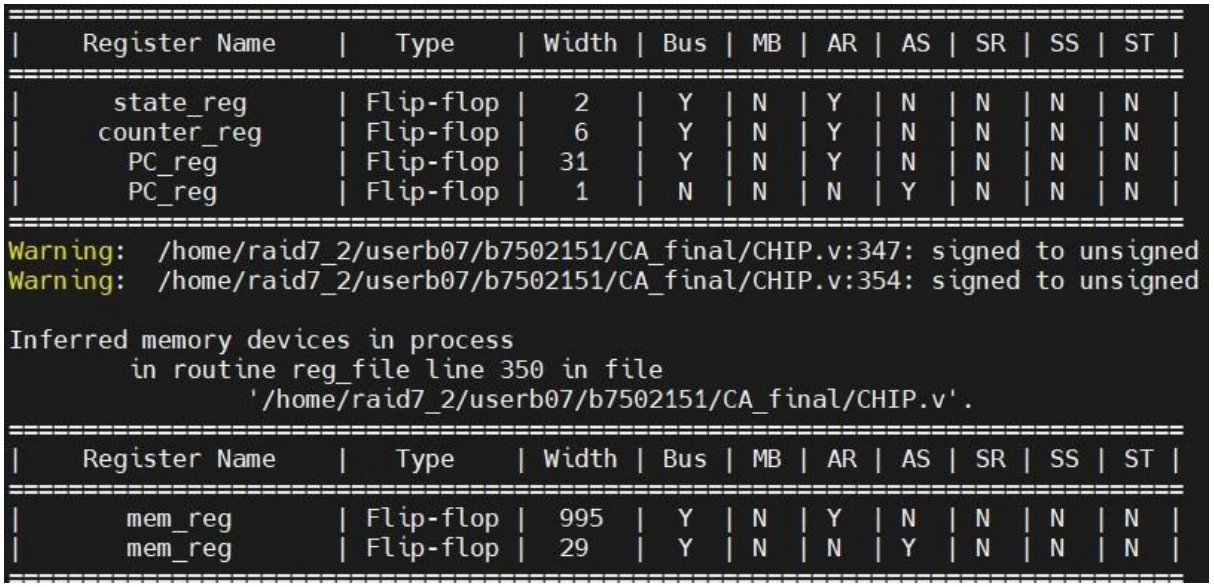
```
START!!! Simulation Start .....

----------------------------------------------------------------

================================================================

Success!
The test result is .....PASS :)

================================================================

Simulation complete via $finish(1) at time 3225 NS + 0
```

5.Describe your observation

- Importance of stalling PC when doing multiple-cycle calculation.
- The valid signal to control the mulDiv calculation's starting cycle and prevent output conflicts.
- ALU operation code can be self-defined instead of following RISC-V manual
- Index of Imm on instruction set listing may be different when it needs to be left shift by 1 bit
- for bonus , we use SIZE_TEXT=100 in Final_tb.v to make sure the process going without error.

6.Snapshot the "Register table" in Design Compiler (p. 22)

```
================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
================================================================================
|      shreg_reg      | Flip-flop |  64   |  Y  | N  | Y  | N  | N  | N  | N  |
|     alu_in_reg      | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|      state_reg      | Flip-flop |   3   |  Y  | N  | Y  | N  | N  | N  | N  |
|     counter_reg     | Flip-flop |   5   |  Y  | N  | Y  | N  | N  | N  | N  |
================================================================================
```



```
=============================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=============================================================================================
|      state_reg      | Flip-flop |   2   |  Y  | N  | Y  | N  | N  | N  | N  |
|     counter_reg     | Flip-flop |   6   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |  31   |  Y  | N  | Y  | N  | N  | N  | N  |
|       PC_reg        | Flip-flop |   1   |  N  | N  | N  | Y  | N  | N  | N  |
=============================================================================================
Warning:  /home/raid7_2/userb07/b7502151/CA_final/CHIP.v:347: signed to unsigned
Warning:  /home/raid7_2/userb07/b7502151/CA_final/CHIP.v:354: signed to unsigned

Inferred memory devices in process
        in routine reg_file line 350 in file
                '/home/raid7_2/userb07/b7502151/CA_final/CHIP.v'.
=============================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=============================================================================================
|       mem_reg       | Flip-flop |  995  |  Y  | N  | Y  | N  | N  | N  | N  |
|       mem_reg       | Flip-flop |  29   |  Y  | N  | N  | Y  | N  | N  | N  |
=============================================================================================
```

7.List a work distribution table

b07502151 孫定洋: PC, CHIP FSM

b07611008 王鐘霆: Control, MUX_2_to_1, Sign_Extend

b08901169 梁正: ALU_Control, ALU, mulDiv, MUX_3_to_1