

Final Report (Group 20)

B08901042 莊明翰

B08901044 蘇子雋

B08901185 曾啟睿

注意:SIZE_TEXT, SIZE_DATA, SIZE_STACK都設成2倍，也就是72。

CPU架構

1. Control Unit (Instruction Decoder)

主要負責解析Instruction、生成immediate以及各式各樣的control訊號：

- branch: 用來配合ALU_zero判斷PC_nxt
- jal/jalr: 用來判斷PC_nxt以及ALU_data1是PC還是rs1_data
- is_AUIPC: 用來判斷ALU_data1是PC還是rs1_data以及rd_data
- mem_wen_D: 決定是要讀/寫記憶體
- mem_to_reg: 決定要給rd的資料是mem_rdata_D或是ALU_result
- ALU_control: 告訴ALU/mulDiv應該要做什麼運算
- ALU_src: 決定ALU第二個輸入是rs2還是imm
- regWrite: 決定有沒有要改寫register, 例如auipc、jalr、load...
- imm: 有各個type immediate的生成規則, 最後依據instruction解析結果決定要輸出哪一個

2. ALU

為了要能夠執行規定的至少12種指令，ALU總共只需要5種operations：add, sub, xor, beq, slti, 再加上為了要執行HW1而加入的srli, bge, 總共7種operations，不過一開始看投影片以為會用到的srai和slli也一起做出來了。

3. ALU(Multi-cycle)

將HW2補上作為multdiv的module。並且在進行mul的instruction時，不去更新PC的值。接著會詳細說明如何設置控制信號。

4. Wire connection

依據從ALU及Control unit給出的各種訊號決定輸入、輸出、及PC端的所有更新值。並使用PC_IMM_ADDER來計算部分運算所需更新的PC值結果。

Extra Instruction

1. JAL

jal指令會將pc+4存到rd並將pc_nxt設為pc+imm, 由於jalr是修改自U-type的J-type指令所以需要額外新增一個J-type的immediate生成規則，以及一個jal的control訊號，讓ALU的src1可以拿到pc、rd可以得到PC+4。

2. JALR

jalr指令會將pc+4存到rd並將pc_nxt設為rs1+imm, 因為jalr是標準I-type所以不需要額外新增immediate生成規則, 但還是要新增一個jalr訊號作為其他元件的參考依據。

此外新增了一個ALU_control 4'b0010, 在ALU中稱為ALU_JUMP, 此時ALU會計算pc+4, 而ALU_data1會被設為pc、ALU_data2則是dont care, ALU_result固定輸出ALU_data1+4也就是PC+4並存到rd。

3. AUIPC

auipc指令會把imm放到高位元的20bits並與PC相加寫入rd, 因此需要額外新增一個U-type immediate生成規則, 其他則依賴另一個新增的is_AUIPC訊號讓ALU_data1是PC。

Multi-cycle Instruction

設置了PC_pause 做為控制信號, 當Control的部分給出使用multicycle的訊號時, PC_pause的值設定為1其餘情況設定為0, 而一旦出現ready訊號(表示完成了multi-cycle的ALU), 再將其值設定為0, 使PC_next能讀入下一個PC + 4的值。另外設定mode參數等於1時做除法、等於0時做乘法。

Simulation time

依序為leaf、perm、HW1

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

=====
Simulation complete via $finish(1) at time 275 NS + 0

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

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

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

=====
Simulation complete via $finish(1) at time 2285 NS + 0
```

Observation

jal, jalr, auipc三個指令的操作十分相像, PC_nxt和rd到底要是甚麼值需要搞清楚, 並且把運算分別交給ALU(算出rd)和PC_imm_adder(算出PC_nxt)計算。

在HW1的部分, 因為加上HW1的扣長度會超過預定的36, 於是就直接把所有size, 包含SIZE_TEXT, SIZE_DATA, SIZE_STACK都設成2倍, 也就是72, 便很順利地過了。

debug的部分因為nWave裡只能看到machine code的指令, 不好debug, 一開始還自己手動把machine code翻成assembly, 不過後來發現其實可以拿去jupyter跑一下就知道了。

這次的作業有種集大成的感覺, 也有用到過去的HW2跟HW1, 透過verilog做出了第四章學到的整個CPU架構, 經過這次的作業不光熟悉了verilog的撰寫過程, 在寫code的過程中也一步一步加深了對於CPU架構中各個MUX及接線的印象。另外一個很大的困難點在於把三個人各自負責的部分組裝起來, 因為一開始只有大概分配要做的部件, 而沒有仔細規定每一個部件應該要有什麼API, 以及API的格式, 因此後來是在互相檢查、trial and error中磨出最終結果, 雖然因為專案小, 所以沒有太嚴重無法處理的問題, 但是也確實多走了許多彎路, 所以說良好的溝通還是至關重要。

Register table

| Register Name | Type | Width | Bus | MB | AR | AS | SR | SS | ST |
|--|-----------|---------|--------------|----|----|----|----|----|----|
| PC_reg | Flip-flop | 31 | Y | N | Y | N | N | N | N |
| PC_reg | Flip-flop | 1 | N | N | N | Y | N | N | N |
| <hr/> | | | | | | | | | |
| Warning: /home/raid7_2/userb08/b08042/CA/CA_final/Verilog/CHIP.v:217: signed to Warning: /home/raid7_2/userb08/b08042/CA/CA_final/Verilog/CHIP.v:227: signed to | | | | | | | | | |
| Inferred memory devices in process in routine reg_file line 220 in file '/home/raid7_2/userb08/b08042/CA/CA_final/Verilog/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 |
| <hr/> | | | | | | | | | |
| Statistics for MUX_OPs | | | | | | | | | |
| block name/line | Inputs | Outputs | # sel inputs | | | | | | |
| reg_file/211 | 32 | 32 | 5 | | | | | | |

```

=====
|      Line      | full/ parallel |
=====
|      522       | auto/auto     |
=====

Statistics for case statements in always block at line 552 in file
  '/home/raid7_2/userb08/b08042/CA/CA_final/Verilog/CHIP.v'
=====
|      Line      | full/ parallel |
=====
|      554       | auto/auto     |
=====

Inferred memory devices in process
  in routine mulDiv line 602 in file
  '/home/raid7_2/userb08/b08042/CA/CA_final/Verilog/CHIP.v'.
=====
| 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   | Y   | 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   |
=====

Presto compilation completed successfully.
Current design is now '/home/raid7_2/userb08/b08042/CA/CA_final/Verilog/CHIP.db:CHIP'
Loaded 6 designs.
Current design is 'CHIP'.
CHIP reg_file Control ALU PC_IMM_ADDER mulDiv
design_vision> █

```

Work distribution table

莊明翰：

- ALU
- Wire connection
- 測試、debug
- 各自負責部分的report

蘇子雋：

- Control unit
- Control訊號與其他元件互動的維護
- Wire connection
- 各自負責部分的report

曾啟睿：

- ALU(Multi-cycle)
- Wire connection
- 各自負責部分的report