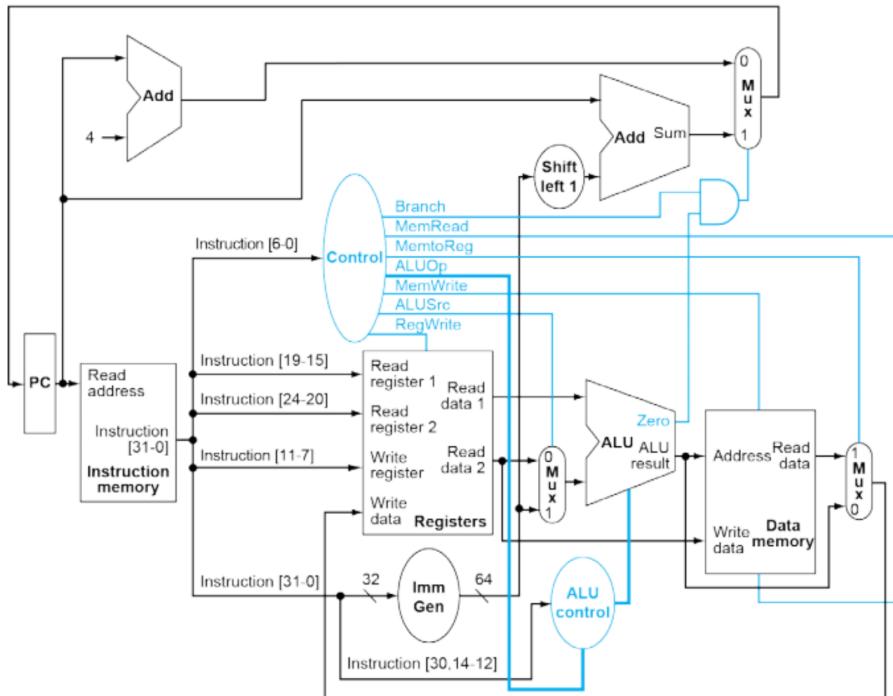


CA Final Project Report

Group 35 B08901158 吳詩昀, B09901081 施伯儒

CPU Architecture



CPU 架構基本和講義上的 datapath 雷同(如上圖)，為了支援不同指令，我們有額外增加了幾個部分：

1. data memory 後多加兩個 mux
 2. alu 的兩個 input 皆有 mux
 3. PC_nxt 多加額外的一個 mux
 4. 多加一個 mux 去選 rs1_data 和 PC
 5. 多加一個 Adder 去加 (4) 中的結果和 immediate
 6. 新增一個負責做 mul, div 運算的 mulDiv unit
 7. ALU Control 多加了 mul, div 指令的處理, 會 issue mulDiv_valid 以及 mulDiv_mode 紿 mulDiv 執行運算
 8. 多加一個 Or gate 在 Branch 的 And gate 之前

Handling Instructions

AUIPC

- 在alu input 和 rs1_data 之間多加一個 mux, 去選擇要 input PC or rs1_data
- control signal 由Control unit 提供

JAL, JALR

- Control unit 多輸出兩個 control signal : Jump, Jump_control
- Jump==1 表示最後會有 jump 的 address 要放到 PC 裡面, 因此在 PC_nxt 才會多加一個 mux 去選出正確的結果 (不論jal jalr Jump都等於1)
- Jump_control 目的在篩選要以 rs1_data 還是 PC 去跟 immediate 做相加, 來更新 PC, 以下分為 JAL 跟 JALR 兩種情況:
 - ❖ jalr : Jump_control==1, 利用 mux 選出 rs1_data, 並放到adder去加上 immediate, 以產生要跳過去的address
 - ❖ jal: Jump_control==0, 同上, 只不過選到的是PC
- 最後, 在 data memory 多加一個 mux, 因為 Jump==1 時, 我們得把 PC+4 存到 register, 所以得用mux去選出rd_data or PC+4 (control signal == Jump)

BGE

-> 令ALU_control產生額外的control輸出到ALU中，而ALU除了zero以外，多輸出一個BGE signal, 新加的Or gate的就是當zero或BGE為1時便輸出1, 接著去和Branch去AND。

-> 產生Zero 和 BGE的規則如下:

```
assign zero = ((in_a==in_b)?1:0) && alu_control==4;
```

```
assign bge = ((in_a>=in_b)?1:0) && alu_control==5;
```

再多和alu_control去And, 目的是確定當下的指令的確是beq or bge

Handling Multi-cycle Instructions

MUL, DIV

- 接收到 MUL 或 DIV 指令後, Control 會以 R-type 的方式處理, 將 ALUOp 紹到 ALU Control Unit
- 在 ALU Control Unit 中會根據他的 func7, fun3 去決定為何種指令

- ❖ 若判定是 MUL，會 issue mulDiv_valid=1 以及 mulDiv_mode=0 紿 mulDiv unit 執行運算
- ❖ 而若判定是 DIV, 則會 issue mulDiv_valid=1 以及 mulDiv_mode=1
 - mulDiv 在接到 valid=1 之後開始進行運算, 並在算完時輸出 ready 跟 result
 - MUL, DIV 跟其他 instructions 不同的是在運算時需要用到 multi-cycle, 因此我們需要停止 PC 的更新、等待 mulDiv 的運算, 我們會利用 mulDiv_valid 跟 mulDiv_ready 兩個直去判斷 mulDiv 是否在運算中
 - ❖ 如果 mulDiv_valid==1 && mulDiv_ready==0, PC 會被賦予原本的值
 - ❖ 其餘的情況才會繼續更新

Simulation & Observation

- ❖ Leaf: $a = 3, b = 9, c = 5, d = 17$

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

由於每個指令都是 single cycle instruction, 所以 Total time 大致為 instruction count \times 1 clock cycle time (10 ns)

- ❖ Perm: $n = 8, r = 5$

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

因為其中有指令是 MUL, 需要 multi-cycle 的執行時間 (32 cycles), 所以 Total time 還要再加 MUL 的額外的 cycle, 以 $n = 8, r = 5$ 為例, 總共有 5 個 MUL, 就至少會需要 5×32 clock cycle time (320 ns) 的時間

- ❖ (Bonus) HW1: $n = 11$

```
Simulation complete via $finish(1) at time 2115 NS + 0
```

hw1_1 跟 perm 一樣含有 MUL, 同樣要處裡 multi-cycle instruction, 再加上 recursive function 讓 instruction count 增加, 所以總共所需要花的時間會大於前兩個 simulation

附註: 因為這裡的 *instruction* 數較多, 在模擬時我們有將我們有將 SIZE_TEXT、SIZE_DATA、SIZE_STACK 調大至 360

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

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

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
alu_in_reg	Flip-flop	32	Y	N	N	N	N	N	N
shreg_reg	Flip-flop	64	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

Work Distribution (組員都好棒 XD)

- ❖ 施伯儒: Control unit, imm_gen, adder, mux 和相關的 wire reg + Report + instructions (auipc,jal,jalr) + 一直把wire前後名字打錯
- ❖ 吳詩昀: 實作 mulDiv unit, ALU unit, ALU control unit 和相關的 wire reg + Report