# VLSI 系統設計-HW5 (共 75 分)

姓名: 李煒權

學號: E24045165

# 授課教師:

# 1.系統簡介(共 25 分)

## 1.指令集格式(5分)

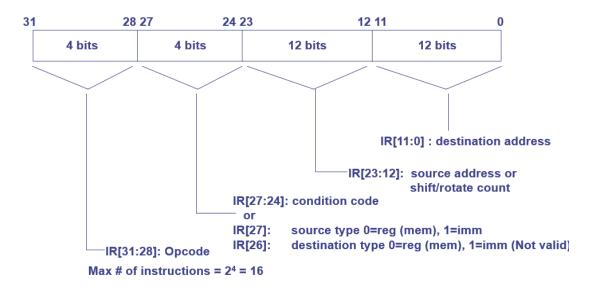
- 13 instructions (added 2 instruction)
- 32 bits instruction register
- 12 bits address register
- 5 bits PSR (processer status register)
- 33 bits result

## 2.指令集格式欄位的名稱、長度、說明(5分)

#### 指令有 32bits 組成

- 1. OPCODE
  - a. 第31~28位元
  - b. 主要分辨指令主要功能 (例: BRA OPCODE 為 0001)
- 2. SRCTYPE
  - a. 第 27 位元
  - b. SRCTYPE 若為 IMD 則為 1, 非 IMD 值則為 0
- 3. DSTTYPE
  - a. 第 26 位元
  - b. DSTTYPE 若為 IMD 則為 1, 非 IMD 值則為 0
- 4. CCODE
  - a. 第 27 ~ 24 位元
  - b. CONDITION CODE, 負責分辨當 BRA 指令時,程式是否要繼續或 跳到指定的行列
- 5. SRC
  - a. 第23~12位元
  - b. SOURCE 地址, 若 SHF 或 ROT 指令則是位移位數
- 6. DST

- a. 第11~0位元
- b. DESTINATION 地址



## 3.BRA 指令的定址(5分)

PRE-instruction: 上個指令在執行就會完成 setccode 這個 task, 這個 task 主要是設定 PSR 值。

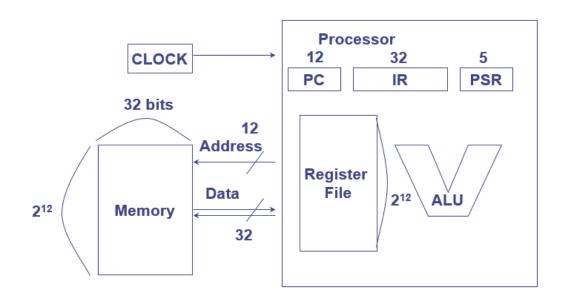
- 1. 在 fetch 階段, IR 接收到了指令後, PC 依然正常加一
- 2. 在 execute 階段,利用 IR 上的 CCODE,比對上條 instuction 已經設定好的 PSR,若條件相符合則跳到指定的指令地址(PC = DST), program counter 改變,若條件不符合則 PC = PC + 1
- 3. 在 write 階段則無須工作

### 4.架構(共 10 分)

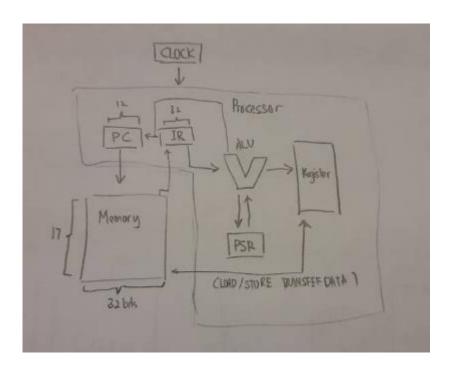
## 1.基本零件(5分)

- 1. CLOCK 產生 cycle 以讓系統達到 state machine 的效果
- 2. Memory 記憶體, 供系統儲存指令後數值, 該位置的值為指令或數值需看指令是怎麼運用的
- 3. Register- Processor 裡的記憶體,複製暫存 processor 裡的一些值
- 4. ALU 運算單元, 負責運算的部分, 例: ADD, MUL等
- 5. PC program counter, 儲存下一條指令在 MEM 的地址
- 6. IR 儲存指令

7. PSR – 存取判斷 result 是否有 Negative, Even, Parity, Zero, Carryout 的特性, 以供 BRA 指令做判斷



# 2.資料路徑(系統架構圖)(需自行製作)(5分)



# 2.系統目前可執行之指令(5分)(需說明新增之指令)

- 1. NOP
- 2. LD
- 3. STR

- 4. BRA
- 5. ADD
- 6. CMP
- 7. MUL
- 8. DIV (NEW) 除法 ALU
- 9. RMD (NEW) 餘數 ALU
- 10. SHF
- 11. ROT
- 12. JMP(NEW) JUMP,任何情況下都跳到指定的下個指;。令,改變PC值
  - 13. HLT

# 3.系統驗證方法與結果分析(共 20 分)

## 1.驗證方法(需說明 Test Program) (10 分)

利用 display 方式,顯示 OPCODE, result, result 存取的位置等,比對自己寫好的 BinaryCode, 檢查程式有沒有與理想中有一樣的結果。

#### 一下是我列印出來的結果:

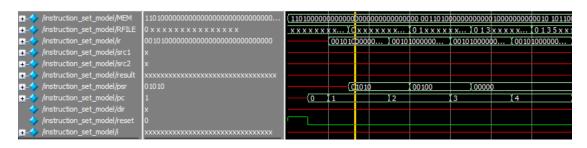
```
OPCODE = LD
                      OPCODE = CMP
LOAD 0 to REFISTER 0
                      REGISTER = 1
OPCODE = LD
LOAD 1 to REFISTER 1
                      OPCODE = ADD
                      OPCODE = LD
                      REGISTER = 1
LOAD 3 to REFISTER 2
                      OPCODE = BRA
OPCODE = LD
LOAD 5 to REFISTER 3
                      NO JUMP
OPCODE = CMP
                      OPCODE = DIV
REGISTER = 1
                      REGISTER =
```

## 2.結果分析(需附上 nWave 截圖) (10 分)

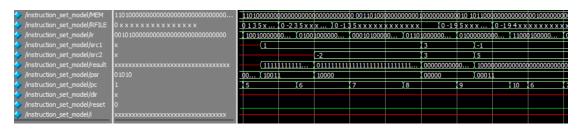
這是我設計的指令,總共有17行指令

```
LD R0, #0
                  BRA #6, ZERO
LD R1, #2
                   DIV R1, R2
                   RMD R1, R2
LD R2, #3
                   SHF R2, #1
LD R3, #5
                   SHF R2, #-1
CMP R3
                   ROT R2, #1
ADD R3, #1
                   ROT R2, #-1
MUL R1, R2
                   STR R2, #2
ADD R3,#1
                  HLT
```

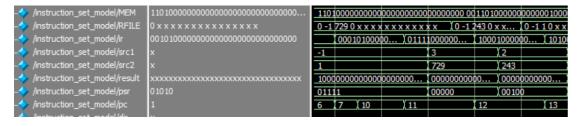
前 4 個 instructions 都為 LD 檔案, RFILE (register) 被置入了幾個值



第 4 個 instruction 就開始運算,這是一個 loop,第九行指令是 JMP 指令,在波形裡 10 直接換到 6 就完成了 JMP instruction, JMP 回到第 6 行 instuction 進行判斷時候要繼續在 loop 裡面運算。



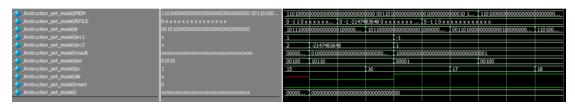
經過了 5 次的 loop,BRA 的條件判斷成功直接跳到第十行指令繼續執行程式



這兩個指令為 SHF 指令, 主要在向左或向右位移位元

// /instruction_set_model/MEM	110100000000000000000000000000000000000	1101000000	00000000000	00000000000	0 001101000	0000000000	0000000000
/instruction_set_model/RFILE	0 x x x x x x x x x x x x x	0-110	0-120xx	xxxxxx	xxx	0-110xx	xxxxxx
/instruction_set_model/ir	001010000000000000000000000000000000000	1010000000	0000000	1010100000	00000000001	0000000	1011100000
/instruction_set_model/src1	x	-1			1		
/instruction_set_model/src2	x	1			2		X
/instruction_set_model/result	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0000000000	00000000000	0000000	0000000000	00000000000	000000
/instruction_set_model/psr	01010	00110			00100		X
/instruction_set_model/pc	1	13		14			15
/instruction_set_model/dir	х	<b> </b>					
/instruction_set_model/reset	0						
/instruction_set_model/i	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	111111111	1111111111	.11111111	0000000000	00000000000	000000

這兩個指令則是 ROT 指令. 在 PC 等於 18 時程式結束



## 4.問題與討論(5分)

我的問題是我們課堂學的 RISC 指令是簡易版的嗎? 因為我在網絡上查的 RISC 附上的指令,在 IR 上面有三個值的輸入(SRC1,SCR2,DST),兩個值的輸入不太方便,例如說一直要利用兩個值相加,在只能輸入兩個值做加法時必定有一組號碼會改變,這樣會導致不能一直重用這兩個值,要一直 load 才有辦法達成。

### 5.心得(加分, 5分)

這次的作業讓我更理解整個 code 的架構,細節等,透過觀察波形 debug,雖然看得很累,但卻是最直接的 debug 方式。在程式裡原本只加入 DIV 及 RMD 新的功能,在在設計 loop 運算時遇到了一點問題,在 BRA 判斷必須比較麻煩地用一些數值作為判斷,所以我增加了 JUMP 指令,能忽視所有情況地 JUMP,最後才想起原來在要在 BRA 指令的 CCODE 改為 ALWAYS 就達到 JUMP 的功能,自己白寫了一些程式。