DSL_B09_EXP3

B00902015 蒙捷文
B00902064 宋昊恩
B00902074 阮昱諾
B00902108 陳宣廷

分工情形

- B00902015 蒙捷文: Assembler + Calculator Input
- B00902064 陳宣廷: Simulator + Calculator Debug
- B00902064 宋昊恩: CPU + Report 撰寫
- B00902064 阮昱諾: Calculator 整體設計

ASSEMBLER 目前狀況

- 在實作的過程中有多次擴充Instruction以及修改Machine Code，因此Assembler也必須一直更新
- 額外支援的部分包括「註解」、「變數賦值」、「單register右移」、「單register左移」、「連續三register右移」、「連續三register左移」、「比較Rx、Ry」等
- 目前測試結果一切良好

SIMULATOR 目前狀況

- 跟Assembler一樣，一旦Instruction有任何更新，Simulator也必須與之同時更新
- 原本想法是利用Simulator來協助檢查Assemble是否正確，但是，由於一開始接口沒有切得非常清楚，導致Simulator在設計上也有遇到相當大的困難
- 目前測試結果一切良好

CPU目前狀況

- 主體架構稱作Controller，底下控制四個子架構，分別是CPU、LCD、7-SEG、RAM，Controller負責這四個子架構之間的溝通

- CPU架構底下有一個子架構，稱作ALU，CPU負責重複性的從記憶體中讀取指令，並且按照指令做出相對應的行為

CPU目前狀況 (續)

- ALU底下有四個子架構，其中兩個是用來做加減法的加速運算，可以將線性的加法時間壓縮成以2為底的log時間，另外兩個則是我手動實現乘法器與除法器
- 經過測試這樣可以有效壓低資源量，最後的資源量為2278/68416(3%)，總程式長度為1200行
- 目前測試結果一切良好

VERILOG PROJECT – PIN 腳 (1/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[63]	Output	PIN_AF12	8	B8_N1		
Seg[62]	Output	PIN_AD12	8	B8_N1		
Seg[61]	Output	PIN_AD11	8	B8_N2		
Seg[60]	Output	PIN_AF10	8	B8_N2		
Seg[59]	Output	PIN_AD10	8	B8_N2		
Seg[58]	Output	PIN_AH9	8	B8_N1		
Seg[57]	Output	PIN_AF9	8	B8_N2		
Seg[56]	Output	PIN_AE8	8	B8_N3		
Seg[55]	Output	PIN_AC17	7	B7_N2		
Seg[54]	Output	PIN_AD17	7	B7_N2		
Seg[53]	Output	PIN_AF17	7	B7_N2		
Seg[52]	Output	PIN_AE17	7	B7_N3		

VERILOG PROJECT – PIN 腳 (2/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[51]	Output	PIN_AG16	7	B7_N3		
Seg[50]	Output	PIN_AF16	7	B7_N3		
Seg[49]	Output	PIN_AE16	7	B7_N3		
Seg[48]	Output	PIN_AG13	8	B8_N0		
Seg[47]	Output	PIN_AC19	7	B7_N1		
Seg[46]	Output	PIN_AE19	7	B7_N1		
Seg[45]	Output	PIN_AB19	7	B7_N1		
Seg[44]	Output	PIN_AB18	7	B7_N1		
Seg[43]	Output	PIN_AG4	8	B8_N3		
Seg[42]	Output	PIN_AH5	8	B8_N3		
Seg[41]	Output	PIN_AF7	8	B8_N3		
Seg[40]	Output	PIN_AE7	8	B8_N3		

VERILOG PROJECT – PIN 腳 (3/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[39]	Output	PIN_M4	2	B2_N3		
Seg[38]	Output	PIN_M6	2	B2_N2		
Seg[37]	Output	PIN_M7	2	B2_N2		
Seg[36]	Output	PIN_M8	2	B2_N1		
Seg[35]	Output	PIN_N7	2	B2_N2		
Seg[34]	Output	PIN_N10	2	B2_N2		
Seg[33]	Output	PIN_P4	2	B2_N3		
Seg[32]	Output	PIN_P6	2	B2_N3		
Seg[31]	Output	PIN_L6	2	B2_N1		
Seg[30]	Output	PIN_M2	2	B2_N3		
Seg[29]	Output	PIN_M1	2	B2_N3		
Seg[28]	Output	PIN_N3	2	B2_N3		

VERILOG PROJECT – PIN 腳 (4/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[27]	Output	PIN_N2	2	B2_N3		
Seg[26]	Output	PIN_P3	2	B2_N3		
Seg[25]	Output	PIN_P2	2	B2_N3		
Seg[24]	Output	PIN_P1	2	B2_N3		
Seg[23]	Output	PIN_K6	2	B2_N1		
Seg[22]	Output	PIN_K5	2	B2_N1		
Seg[21]	Output	PIN_K4	2	B2_N2		
Seg[20]	Output	PIN_K1	2	B2_N2		
Seg[19]	Output	PIN_L3	2	B2_N2		
Seg[18]	Output	PIN_L2	2	B2_N3		
Seg[17]	Output	PIN_L1	2	B2_N3		
Seg[16]	Output	PIN_M3	2	B2_N3		

VERILOG PROJECT – PIN 腳 (5/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[15]	Output	PIN_K2	2	B2_N2		
Seg[14]	Output	PIN_E4	2	B2_N0		
Seg[13]	Output	PIN_F4	2	B2_N0		
Seg[12]	Output	PIN_G4	2	B2_N0		
Seg[11]	Output	PIN_H8	2	B2_N0		
Seg[10]	Output	PIN_H7	2	B2_N0		
Seg[9]	Output	PIN_H4	2	B2_N1		
Seg[8]	Output	PIN_H6	2	B2_N0		
Seg[7]	Output	PIN_G2	2	B2_N2		
Seg[6]	Output	PIN_G1	2	B2_N2		
Seg[5]	Output	PIN_H3	2	B2_N1		
Seg[4]	Output	PIN_H2	2	B2_N2		

VERILOG PROJECT – PIN 腳 (6/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
Seg[3]	Output	PIN_H1	2	B2_N2		
Seg[2]	Output	PIN_J2	2	B2_N2		
Seg[1]	Output	PIN_J1	2	B2_N2		
Seg[0]	Output	PIN_K3	2	B2_N2		
clk	Input	PIN_AD15	7	B7_N3		
lcdC[2]	Output	PIN_F3	2	B2_N0		
lcdC[1]	Output	PIN_E2	2	B2_N1		
lcdC[0]	Output	PIN_F2	2	B2_N2		
lcdD[7]	Output	PIN_B2	2	B2_N0		
lcdD[6]	Output	PIN_C3	2	B2_N0		
lcdD[5]	Output	PIN_C2	2	B2_N0		
lcdD[4]	Output	PIN_C1	2	B2_N0		

VERILOG PROJECT – PIN 腳 (7/8)

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
lcdD[3]	Output	PIN_D3	2	B2_N1		
lcdD[2]	Output	PIN_D2	2	B2_N1		
lcdD[1]	Output	PIN_E3	2	B2_N0		
lcdD[0]	Output	PIN_E1	2	B2_N1		
lcdOn	Output	PIN_F1	2	B2_N2		
reset	Input	PIN_T29	6	B6_N0		
swt[17]	Input	PIN_L8	2	B2_N1		
swt[16]	Input	PIN_L7	2	B2_N1		
swt[15]	Input	PIN_L4	2	B2_N2		
swt[14]	Input	PIN_L5	2	B2_N1		
swt[13]	Input	PIN_T9	1	B1_N0		
swt[12]	Input	PIN_U9	1	B1_N0		

VERILOG PROJECT – PIN 腳 (8/8)

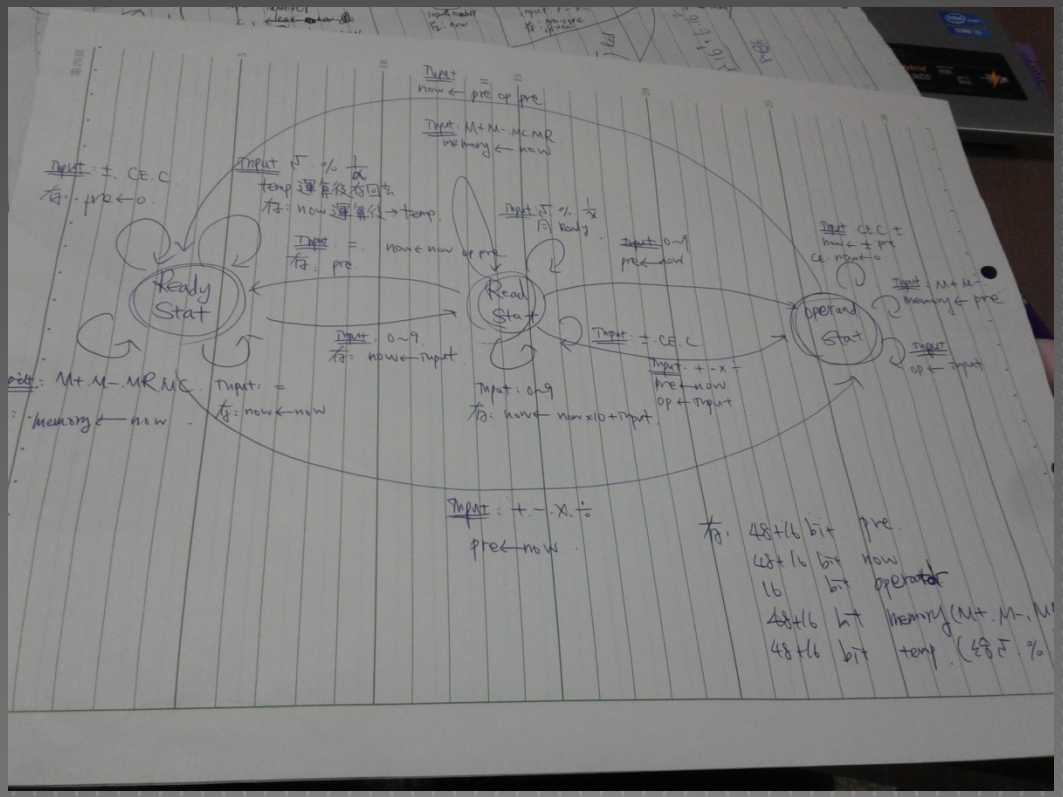
To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
swt[11]	Input	PIN_V10	1	B1_N1		
swt[10]	Input	PIN_W5	1	B1_N1		
swt[9]	Input	PIN_AE27	6	B6_N2		
swt[8]	Input	PIN_AD24	6	B6_N3		
swt[7]	Input	PIN_AD25	6	B6_N3		
swt[6]	Input	PIN_AC23	6	B6_N3		
swt[5]	Input	PIN_AC24	6	B6_N3		
swt[4]	Input	PIN_AC26	6	B6_N3		
swt[3]	Input	PIN_AC27	6	B6_N2		
swt[2]	Input	PIN_AB25	6	B6_N2		
swt[1]	Input	PIN_AB26	6	B6_N2		
swt[0]	Input	PIN_AA23	6	B6_N2		

CALCULATOR 目前情況

- 我們採用類似IEEE 754 Double儲存的架構，對於任何一個大數，我們使用三個Register來儲存數值、一個Register來儲存科學記號，一共是

- 輸入部分，我們決定使用開關的開與關來達成
  - 有四個開關負責給出數字單元0~9 (或16種功能運算)
  - 有一個開關負責輸出訊息(當為On的時候表示送出訊息)
  - 有一個開關決定現在是數字輸入還是功能鍵輸入
  - 另外有四個開關分別監聽8個Register、MAR、MDR、X、Y、W、Carry Flag、PC等，一共使用到10個開關

CALCULATOR – 精美STATE圖



CALCULATOR 目前情況

- 接下來的幾頁，我們會貼上目前Calculator完成部分的Pseudo Code，簡潔起見，我們會用for, if,來做表示，且格式上不會那麼嚴格限制

- 以下是一些前提假設:
  - 運算函式被CALL的時候，兩個數字會分別存在R1~R3, R4~R6 (R7&R8表示2的次方數)，而且在輸入的時候就已經將數字左移到底；因此除了特別的情況(例如:除法)外只需要右移操作
  - 加減法會將結果存回前面那個Register
  - 乘法會將結果存到R7(H)、R8(L)
  - 除法會將商存到R7，餘數存到R8

CALCULATOR 目前情況 – 加法

- Bigadd // Bigadd Flag
- If (Compare R7, R8) RSF3 R8-R7, R1 // R7<R8 R1~R3右移
- Else if (Compare R8, R7) RSF3 R7-R8, R4 // R8<R7 R4~R6右移
- Store [28672], max{R7, R8}
- Add R3, R6
- Add R2, R5(+carry)
- Add R1, R4(+carry)
- Set $add_1, R1
- Set $add_2, R2
- Set $add_3, R3
- Set $add_d, R4 // 將大數存進預設的變數中，讓其他人可以呼叫此函式
- Pop R8
- Jump R8 // 離開此Function 等同於return

CALCULATOR 目前情況 – 減法

- Bigsub // Bigsub Flag
- Load R7, #65535
- Xor R6, R7
- Xor R5, R7
- Xor R4, R7 // 這四行是在對第二個大數做NOT
- Load R7, #1
- Add R6, R7 // 因為2's complement 倒數後還要+1
- Load R7, #0
- Add R5, R7
- Add R4, R7 // 用+0的方式把CARRY加上去

CALCULATOR 目前情況 – 乘法

- Bigmul // Bigmul Flag
- Add R7, R8
- Store memory[0], R7 // 先將R7乘以2的次方數寫出去
- XOR (Test bit 15, R1), (Test bit 15, R4) => R8 // 表示將結果存到R8
- For (i = 0; i < 9; i++) Store [2*i + 28673], R8 // 先把符號弄好存過去

- If(Test bit 15, R1) // if R1<0 R2,R3 都<0 要變號
- For (i = 1; i < 4; i++)
- ~Ri => Ri; // register變號 準備做乘法
- If(Test bit 15, R4){ // if R4<0 R5, R6 都<0 要變號
- For (i = 4; i < 7; i++)
- ~Ri => Ri; // register變號 準備做乘法

CALCULATOR 目前情況 – 乘法 (續)

- Load R7, memory[28673]
- If (R7 == 1){ // 若R7是負的，要變號
- Mul R3,R6 => R7,R8
- ~R7 => R7;
- ~R8 => R8;
- }//end if
- Else Mul R3,R6 => R7,R8
- Store memory[1], R7
- Store memory[2], R8

/* 相同的步驟要重複於所有乘法運算 --- Mul R3,R6 , Mul R3,R5 , Mul R3,R4, Mul R2,R6……Mul R1,R4 */

CALCULATOR 目前情況 – 乘法 (續)

- twenty-five
- Load R1, [28673]
- Load R2, [28674]
- Load R3, [28676]
- Load R4, [28678]
- Load R5, [28684]
- Load R6, [28690]

/* 乘法還有高低位問題，所以兩個大數乘出來的結果共需要6個Register去儲存；這裡讀出來的是儲存在該位置上，最早會被讀出來的數 */

CALCULATOR 目前情況 – 乘法 (續)

- Load R7, [28675]
- Load R8, #0
- Add R2, R7 // 將此數存到相應的位置
- Add R3, R8
- Add R4, R8
- Add R5, R8
- Add R6, R8 // 這邊是為了加上CARRY 將上位的Register +0

- /* 如此對所有當時被存到Memory的數字重複做這件事, 乘法的結果就會被分別存在6個Register裡了(當然必須搞清楚每個數對應到哪個Register) */

CALCULATOR 目前情況 – 乘法 (續)

- Set $mul_6, R1
- Set $mul_5, R2
- Set $mul_4, R3
- Set $mul_3, R4
- Set $mul_2, R5
- Set $mul_1, R6
- Set $mul_d, R7
- Pop R8
- Jump R8

- /* 跟之前的一樣，將大數存進預設的變數中，讓其他人可以呼叫此函式 */

CALCULATOR 目前情況 – 除法

- Bigdiv // Bigdiv Flag
- Sub R7, R8
- XOR (Test bit 15, R1), (Test bit 15, R4) => R8
- Store [28675], R7
- Store [28676], R8

- If(Test bit 15, R1) // if R1<0 R2,R3 都<0 要變號
- For (i = 1; i < 4; i++)
- ~Ri => Ri; // register變號 準備做除法
- If(Test bit 15, R4) // if R4<0 R5, R6 都<0 要變號
- For (i = 4; i < 7; i++)
- ~Ri => Ri; // register變號 準備做除法

CALCULATOR 目前情況 – 除法 (續)

```
For ( j = 0; j < 3; j ++ ){
        For ( i = 15; i >= 0; i -- ){
                Push PC
                bigsub
                Load R8, #1
                If(Testbit 15, R1){             // R1第一位是1 表示R1<R2
                        Push PC
                        bigadd //加回來
                        LSF3 R1, R8;            // 左移1, 做下一個除法
                LSF R7, R8              /* 對Register左移1, 將最後一位填0 */
                }
```

CALCULATOR 目前情況 – 除法 (續)

```
                Else {                  // R1第一位是0 表示R1>R2
                    LSF3 R1, R8;        // 左移1, 做下一個除法
                    LSF R7, R8          // 對Register左移1
                    Or R7, R8           // 最後一位填上1, 除法成功
                }
        }
        Store [28672+j] R7;
}
Load [28672], R1
Load [28673], R2
Load [28674], R3
Load [28675], R4
```

CALCULATOR 目前情況 – 除法 (續)

- If (R4 == 1) // 結果是負的
- For (i = 1; i < 4; i++)
- ~Ri => Ri; // register變號

- Load [28676], R4
- Set $div_1, R1
- Set $div_2, R2
- Set $div_3, R3
- Set $div_d, R4
- Pop R8
- Jump R8

CALCULATOR 目前情況 – 開根號

- Sqrt // Sqrt Flag
- Store [28672], R1
- Store [28673], R2
- Store [28674], R3
- Store [28675], R7 // 把原數存起來 因為運算時一定會動到
- Push PC
- bigdiv
- Set R1, $div_1
- Set R2, $div_2
- Set R3, $div_3
- Set R7, $div_d // 把除法結果Load到Register中

CALCULATOR 目前情況 – 開根號 (續)

- Push PC
- bigadd
- Set R4, $add_1
- Set R5, $add_2
- Set R6, $add_3
- Set R8, $add_d　　// 把加法結果Load到Register中, 再繼續做除法
- Load R7, #1
- Sub R8, R7
- Load R1, [28672]
- Load R2, [28673]

CALCULATOR 目前情況 – 開根號 (續)

- Load R3, [28674]
- Load R7, [28675] // 把原數Load回來準備繼續做除法

/* 重複10次才能逼近開根號的值, 第一次的除數用1下去做(R4~R6) */

- Set $sqr_1, R4
- Set $sqr_2, R5
- Set $sqr_3, R6
- Set $sqr_d, R8 // 最後一樣把結果存在一個開好的變數裡
- Pop R8
- Jump R8 // 跳出此Function

問題與反省檢討

- 分工問題:
  - 這次的Project分工雖然很快速，但是結果卻奇差無比，負責Assembler的同學很晚生出東西來，Simulator、CPU老早寫好卻毫無用武之地；等到Assembler出來已經是期末考周，大家又都各忙各的，使得最後Calculator只有不到兩天的時間可以完成；這也是最後為什麼我們只能夠呈現驗屍報告，而不是完美的計算機。
- 規模問題:
  - 我們在第二次討論的時候決定使用IEEE 754 Double來儲存數字，因為我們認為這是在老師的規定下最適合的儲存格式，雖然我們已經有Double將會十分難以實作的心理準備，但是事實證明我們還是太小看它了。光是將Input轉換成Double儲存就是一番大功夫，科學記號用二進位表示更是難做到不可思議。
  - 夢想太大卻成空，這是我們沒有規劃清楚時間進度的問題。

問題與反省檢討 (續)

- 監督問題:
  - 我們應該要有非常明確的工作回報以及監督制度，一味的默默等待同學主動回應是完全沒有效率的行為，對於每項工作都應該訂定明確的死線。
  - 監督這份工作不應該只是由某一個人來完成，全組組員都有責任、義務催促無法如期完工的同學。
- DSL學習心得:
  - 對於最後只能交出驗屍報告，其實是蠻令人難過的一件事情，但是至少我們都有從中獲得許多；這次CPU成功地壓下資源量，甚至減少到只剩我們Project2資源量的一半，感覺好像真的已經抓到Verilog這個語言的設計精隨。
  - 謝謝老師這學期的教學，希望老師退出教學界後也一樣順心。