**Foundation of Digital Logic and Processors**

**Project: One-cycle MIPS Processor**

Student ID: 2017011090

Name: 游子權

**一、處理器結構**

1. a) 31代表寫入的是第31號寄存器，即$ra，在執行jal指令時需要將Program Counter中所載地址存入$ra中，故需使Register File的Write register一埠連接31。

b) 其對應的是偏移量（shamt），在移位相關指令（sll, srl, sra）時需要用到。

c) 其對應的是（Program Counter所存的地址+4），在jal, jalr指令中，在跳轉之前記錄目前指令的下一條指令所在地址時會用到。

d) 其對應的是read register 1 中所存的data，在跳轉到指定寄存器所存地址的指令（如jr, jalr）中會用到

e) 因為要決定用邏輯擴展（高位補0）還是用算術擴展（高位補上符號位），根據CPU.v第46行，若為ExtOp==0則採邏輯擴展，若為ExtOp == 1則採算術擴展。

f) 不需要修改任何處理器結構，因為全零的指令對應到MIPS中“sll $0, $0, 0”這條指令，即將0號寄存器的內容左移零位存到零號寄存器，這條指令相當於啥也不做(nop)，故不需要修改任何結構。

1. 真值表如下：

|  | **PCSrc[1:0]** | **Branch**  **[2:0]** | **RegWrite** | **RegDst[1:0]** | **MemRead** | **MemWrite** | **MemtoReg[1:0]** | **ALUSrc1** | **ALUSrc2** | **ExtOp** | **LuOp** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lw | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| sw | 0 | 0 | 0 | X | 0 | 1 | X | 0 | 1 | 1 | 0 |
| lui | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | X | 1 |
| add | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| addu | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| sub | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| subu | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| addi | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| addiu | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| and | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| or | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| xor | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| nor | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| andi | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| sll | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | X | X |
| srl | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | X | X |
| sra | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | X | X |
| slt | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| sltu | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | X | X |
| slti | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| sltiu | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| beq | 0 | 1 | 0 | X | 0 | 0 | X | 0 | 0 | 1 | 0 |
| **bne** | 0 | **2** | **0** | **X** | 0 | 0 | **X** | 0 | 0 | 1 | 0 |
| **bgtz** | 0 | **3** | **0** | **X** | 0 | 0 | **X** | 0 | 0 | 1 | 0 |
| **blez** | 0 | **4** | **0** | **X** | 0 | 0 | **X** | 0 | 0 | 1 | 0 |
| **bltz** | 0 | **5** | **0** | **X** | 0 | 0 | **X** | 0 | 0 | 1 | 0 |
| **bgez** | 0 | **6** | **0** | **X** | 0 | 0 | **X** | 0 | 0 | 1 | 0 |
| **bltzal** | 0 | **5** | 1 | **2** | 0 | 0 | **2** | 0 | 0 | 1 | 0 |
| **bgezal** | 0 | **6** | 1 | **2** | 0 | 0 | **2** | 0 | 0 | 1 | 0 |
| j | 1 | X | 0 | X | 0 | 0 | X | X | X | X | X |
| jal | 1 | X | 1 | 2 | 0 | 0 | 2 | X | X | X | X |
| jr | 2 | X | 0 | X | 0 | 0 | X | X | X | X | X |
| jalr | 2 | X | 1 | 1 | 0 | 0 | 2 | X | X | X | X |

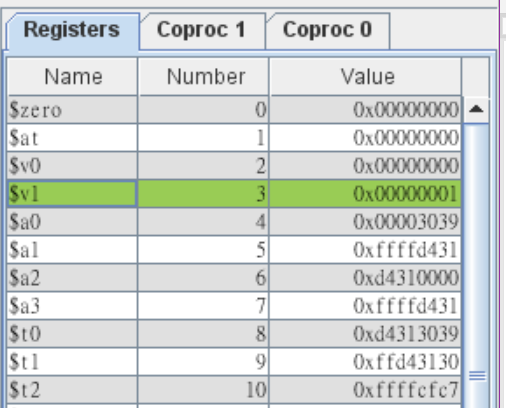
【註】1. 根據CPU.v代碼，ExtOp若為0則採邏輯擴展（高位補0），若為1則採算術擴展（高位補符號位）

2. 指令“addu”、“subu”、“addiu”、“sltiu”中u的意思不是指無符號運算，而是不帶Overflow檢測。但是sltu中的u是指無符號比較，而“andi”則是將0擴展之後的立即數與寄存器中數值做與運算。

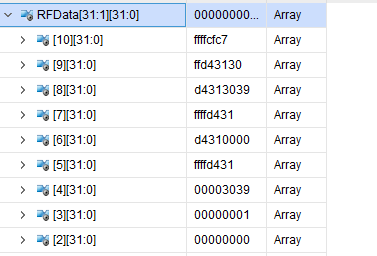
**二、完成控制器**

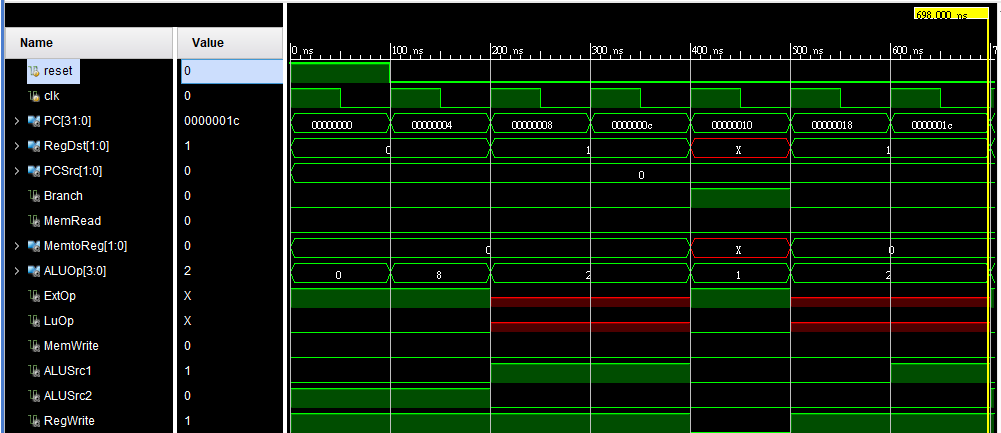
1. 代碼理解：

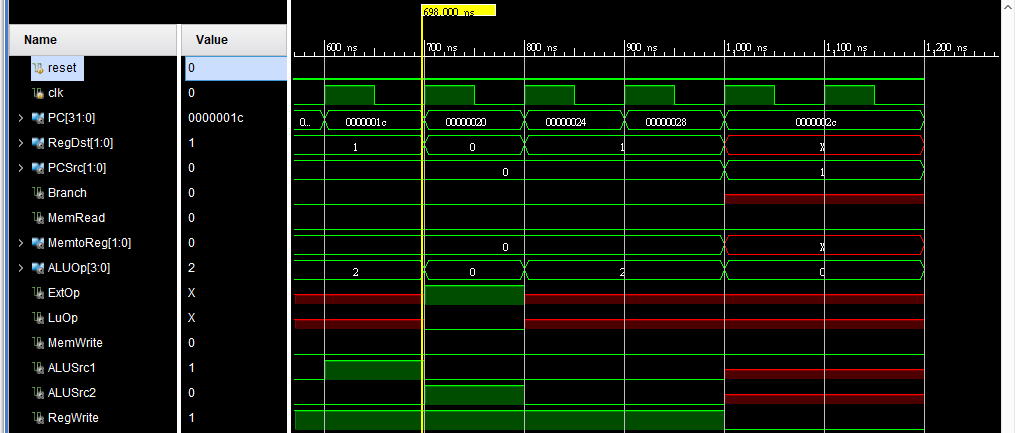
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Addr. |  | **$a0** | **$a1** | **$a2** | **$a3** |
| 0x00 | addi $a0, $zero, 12345 | 0x00003039 | - | - | - |
| 0x04 | addiu $a1, $zero, -11215 | 0x00003039 | 0xffffd431 | - | - |
| 0x08 | sll $a2, $a1, 16 | 0x00003039 | 0xffffd431 | 0xd4310000 | - |
| 0x0c | sra $a3, $a2, 16 | **0x00003039** | **0xffffd431** | **0xd4310000** | **0xffffd431** |
| 0x10 | beq $a3, $a1, L1 | ∵$a3 == $a1 🡪branch to L1! | | | |
| 0x14 | lui $a0, -11111 | This instruction is omitted in 0x10. | | | |
|  |  | **$t0** | **$t1** | **$t2** |  |
| 0x18 | L1: add $t0, $a2, $a0 | 0xd4313039 | - | - |  |
| 0x1c | sra $t1, $t0, 8 | 0xd4313039 | 0xffd43130 | - |  |
| 0x20 | addi $t2, $zero, -12345 | **0xd4313039** | **0xffd43130** | **0xffffcfc7** |  |
|  |  | **$v0** | **$v1** |  |  |
| 0x24 | slt $v0, $a0, $t2 | 0x00000000 | - | ∵Since $a0>$t2∴$v0=0 | |
| 0x28 | sltu $v1, $a0, $t2 | **0x00000000** | **0x00000001** | ∵Since $a0<(unsinged)$t2∴$v0=1 | |
| 0x2c | Loop:j Loop | Infinite Loop! The program will stuck here! | | | |

* 這段程式執行足夠長時間後會進入位於0x2c的無限迴圈。故Program Counter將停在0x2c。
* 上表粗體、淺色底的格子分別為寄存器$a0~$a3,$t0~$t2,$v0~$v1的終值。計算過程如表格所示。
* 如果已知某一時刻在某寄存器中存放著數 0xffffcfc7，仍無法判斷出它是有符號數還是無符號數。因為它只是一串32位二進制數，具體意義為何應綜觀程式的結構及其設計的邏輯來判定與運用。
* 右圖為用MARS執行之結果：與預期之結果相符。

1. 使用 Vivado模擬結果如下圖：



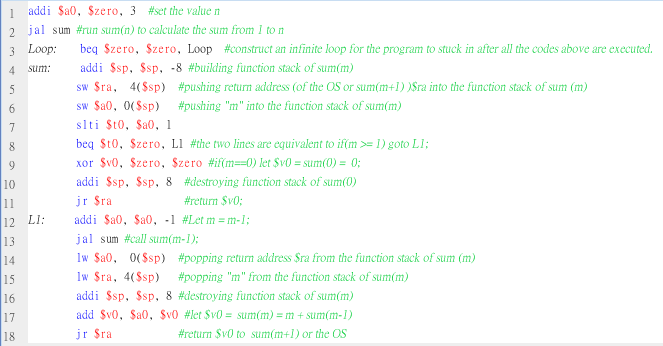




* 1. 一般而言，PC 信號自0開始，逐週期加4，但因其於0x10處遇分支指令beq且依條件判斷進行跳轉，故其略過了0x14的指令，自0x10跳至0x18。
  2. Branch 信號僅在PC==0x10時為 1。它使得PC依0x10位址的指令條件進行了跳轉。
  3. 設時鐘周期為100ns。100~200ns 期間對應第2個週期，PC 值為0x04。對應的指令是“addiu $a1, $zero, -11215”。此時$a1的值為0（因為其被初始化為0）。200~300ns 期間，對應第3個週期，PC值為0x08， $a1 的值變為上條指令行完畢後的結果0xffffd431，即11215的二補碼。這是因為在下個週期開始，計算出來的數值方寫入$a1。下一條指令立即使用到了$a1 的值，不會出現錯誤。因為在本週期的一開始，$a1的新值已經存在寄存器堆中了。
  4. 執行時間足夠長之後（如 1100ns 時）寄存器$a0~$a3,$t0~$t2,$v0~$v1 中的值如上圖所示，與預期一致。

**三、執行匯編程式**

1. 代碼理解與注釋（附於mips2.asm中）：

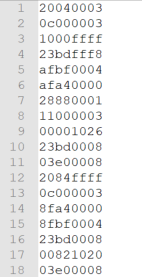


* 如果第一行的 3 是任意正整數n，這段程式可以輸出1+2+…+n的結果。

三段程式碼的功用分述如下：

* Loop（第3行）：構造無限迴圈，使得程序結束後進入此迴圈而使Program Counter不致越界。
* sum（第4~11行）：構造sum(m)的function stack，並處理m==0時返回0之情形。
* L1 （第12~18行）：是sum(m)中的分支，處理sum(m-1)的調用，並在sum(m-1)返回值後處理sum(m) = sum(m-1)+m 之情形。

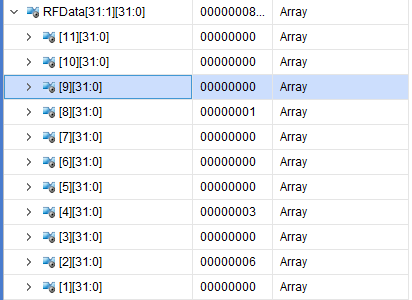
1. 將這段組合語言程式翻譯成機器碼如下（附於mips2.machine）：設所有地址、指令皆可化為32位二進制數，由高至低為第31位至第0位。其中，指令的存儲地址自0x00000000（第1行），而後每行+4，至0x00000044（第18行）止。



* 對於beq語句，其[31:26]為000004，[25:21]為其一欲比較數值之寄存器位址；[20:16]為其另一欲比較數值之寄存器位址。我會確認其欲跳轉位置自本條指令後一條數起之指令數（本條指令後一條記為第0條），如果跳轉到自身則記-1。將該指令數轉成2進制保留其[15:0]位作用指令的[15:0]位。
* 對於jal 語句，其[31:26]為000003，我將 Loop，sum，L1標籤所對應之語句地址，取其[28:3]位，放入指令的[25:0]位。
* 立即數-1被譯為1的二補碼，即0xffffffff。
* 立即數-8 被譯為8的二補碼，即立即數1的二補碼減去7，即：

0xffffffff – 0x00000007 = 0x00000008

1. 修改 InstructionMemory.v（附於code/中）並模擬後，結果如下：
   1. 執行時間足夠長之後（如 5000ns 時），寄存器$a0（RFData[4]）的值即為第1行所給定的輸入n=3，$v0 （RFData[2]）的值即為1+2+…+n的結果，此例中為6。與預期結果相符。



* 1. 觀察、描述並解釋 PC,$a0,$v0,$sp,$ra 如何變化。（波形檔案附於test\_cpu\_behav\_sum.wcfg中）
* PC是Program Counter，其永遠儲存目前執行的指令的地址。每一個時鐘上升沿變化，一般情形下，其值+4，遇jal, beq等跳轉指令時，依指令改寫PC至相應指令的地址。
* $a0傳入sum()函數的參數，在function stack的建構過程（每次jal前）中，其值自3遞減至0，在function stack解構（每次jr $ra前）過程中，其值自0遞增至3。
* $v0儲存函數返回值，每次函數返回（jr $ra）之前$v0將會更新。從模擬中可以看出，$v0按照0，1，1+2=3，1+2+3=6的規律變化。
* $sp是stack pointer。儲存stack頂端的地址。從模擬中可以看出，每一層函式呼叫，$sp遞減8（有新元素$a0和$ra入棧）。函數返回之前，$sp遞增8（$a0和$ra出棧）。
* $ra：在構建function stack過程中，主調函數必須保護現場，故使用jal儲存返回後的地址。在解構function stack過程中，會從stack讀出主調函數指定的返回地址至$ra並返回，故其在每次跳出時都會被重寫一次，惟本程式中jal儲存的位置僅有第4行或第13行，故$ra寄存器的值循0x00🡪0x08🡪0x34🡪0x08變化，最後停有0x08的無限迴圈中。

