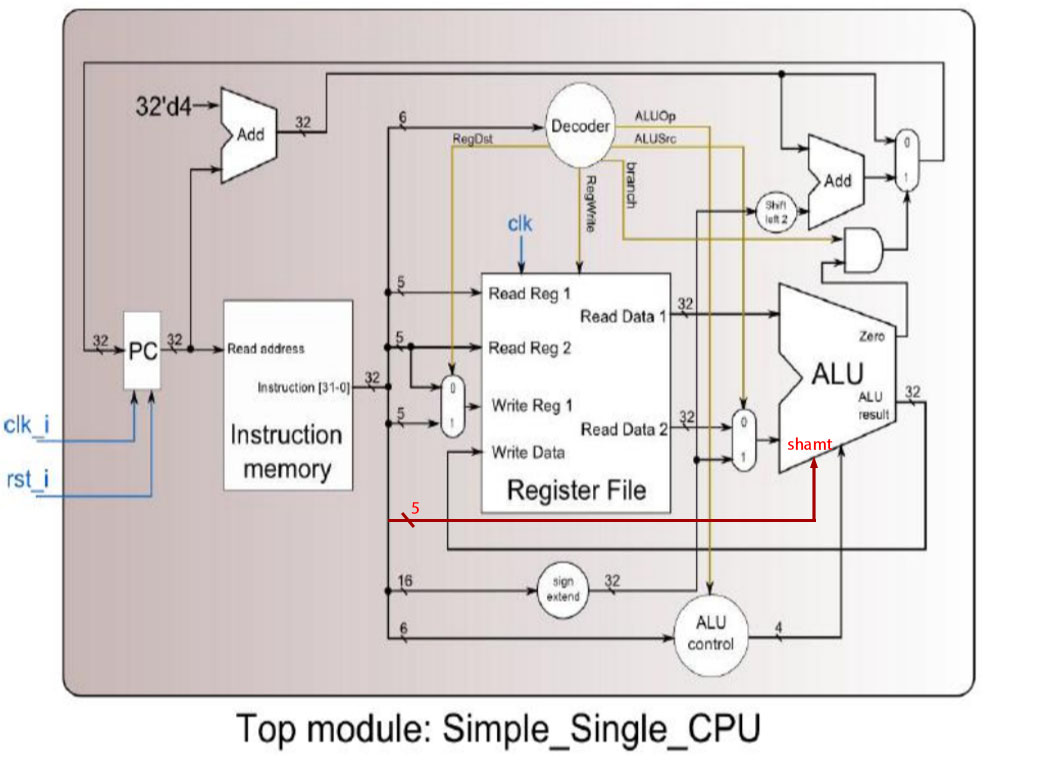
**Computer Organization**

**Architecture diagram:**



**Detailed description of the implementation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Opcode | ALU  Op | ALU  Src | Reg  Dst | Branch | ALUCtrl | ALU操作對應Verilog Code |
| addu | 000\_000 | 0 | 0 | 1 | 0 | 2 | a + b |
| subu | 6 | a - b |
| and | 0 | a & b |
| or | 1 | a | b |
| slt | 7 | a < b ? 1 : 0 |
| sra | 3 | $signed(b) >>> shamt |
| srav | 4 | $signed(b) >>> a |
| addi | 001\_000 | 1 | 1 | 0 | 0 | 2 | a + b |
| sltiu | 001\_011 | 2 | 1 | 0 | 0 | 7 | a < b ? 1 : 0 |
| beq | 000\_100 | 3 | 0 | 0 | 1 | 6 | a - b |
| lui | 001\_111 | 4 | 1 | 0 | 0 | 5 | b << 16 |
| ori | 001\_101 | 5 | 1 | 0 | 1 | 1 | a | b |
| bne | 000\_101 | 6 | 0 | 0 | 1 | 8 | a == b ? 1 : 0 |

上表為各指令對應之Control Signal

1. addu、subu、and、or、slt、srav：

讀取Rs、Rt位置的暫存器資料，經過ALU進行對應的運算後，寫回Rd位置的暫存器。

1. addi、ori：

讀取Rs位置的暫存器資料，並將最後16 bit extend sign之後，兩者經過ALU進行對應的運算，寫回Rt位置的暫存器。

1. lui：

將最後16 extend sign之後，經過ALU左移16位元之後（最低16為為0），寫回Rt位置的暫存器。

1. sltiu

讀取Rt位置的暫存器資料，並將最後16 bit extend sign之後，根據兩者大小關係將Rt位置的暫存器，寫入1或0。

1. srav：

將shamt數值接到ALU，讀取Rt位置的暫存器資料，將其資料右移shamt之後寫回Rt位置的暫存器。

1. beq：

讀取Rs、Rt位置的暫存器資料，經過ALU相減，如果兩個相等，則ALU的Zero輸出為1，在電路右上選擇經過adder加完後的PC存回PC暫存器。

1. bne：

讀取Rs、Rt位置的暫存器資料，經過ALU判斷相等與佛否，如果兩個不相等，則ALU的Zero輸出為1，在電路右上選擇經過adder加完後的PC存回PC暫存器。

**Problems encountered and solutions:**

實作srav時問題比較大，因為srav感覺上是類似I-type的操作，但卻是使用R-type的指令，最後選擇直接將shamt接到ALU上進行計算，不知道是不是好的做法。

**Lesson learnt (if any):**

做完這個作業後對簡單CPU有更清楚的了解，雖然因為沒什麼寫過verilog的經驗，一開始花了比較久的時間，不過還好最後能成功完成。