**VLSI System Design (Graduate Level)**

**Fall 2020**

**HOMEWORK I**

**REPORT**

**Must do self-checking before submission:**

**Compress all files described in the problem into one tar**

**All SystemVerilog files can be compiled under SoC Lab environment**

**All port declarations comply with I/O port specifications**

**Organize files according to File Hierarchy Requirement**

**No any waveform files in deliverables**

**Student name: 杜冠勳**

**Student ID: N26094883**

|  |  |  |
| --- | --- | --- |
| **SUMMARY** | | |
| 1. **Requirements** | **implement a simplified pipeline CPU with the following features**   1. **Implement the 33 instructions** 2. **The number of pipeline stage is 5.** 3. **Register File size: 32x32-bit ➔ x0 is read only 0.** 4. **Instruction memory size: 16Kx32-bit  Data memory size: 16Kx32-bit** 5. **Timescale: 1ns/10ps** 6. **Maximum Clock period: 20ns (50MHz)** | |
| **Verification** | **PROG 0** | **Verification for 33 instruction** |
| **PROG 1** | **Implentment Bubblesort algorithm for sorting** |
| **PROG 2** | **Implentment Multiplication** |
| **PROG 3** | **Implentment Greatest common divisor** |

|  |  |
| --- | --- |
| **INTRODUCTION FOR DESIGN OF THE CPU** | |
| **SPEC** | |
| **PIPELINE STAGE** | **5** |
| **INSTRUCTION** | **33** |
| **FULL GRAPH** | |
|  | |

|  |  |  |
| --- | --- | --- |
| **STAGE1 INSTRUCTION FETCH** | | |
|  | | |
| **MODULE** | **FUNCTION** | **SPECIAL\_DESIGN** |
| **DIDER4** | **Get the Instruction Address by dividing**  **the pc\_counter result by4** | **Using shift right 2 bits instead using the divider which can reduce hardware area.** |
| **PAUSE INSTRUCTION** | **If there is any load use data hazard happen,this module will enable to pause the instruction.**  **The instruction an d pc value will be rewrited tio the register in next clock.** | **NONE** |
| **rst\_contoller** | **If there is any branch happen this module will enable flush the register. The module will be more detailed discuss later.** | **NONE** |

|  |  |  |
| --- | --- | --- |
| **STAGE2 INSTRUCTION DECODE & STAGE5 WRITE BACK** | | |
|  | | |
| **MODULE** | **FUNCTION** | **SPECIAL\_DESIGN** |
| **DECODER** | **Decoding 33 bits instruction into**  **different elements which is used for the following module** | **NONE** |
| **SIGNED UNSIGNED EXTENDED\_**  **SELECT\_IMM** | **Make the Immediate data referring to the type of the instruction** | **NONE** |
| **REG** | **32×32-bit. registers** | **Posedge clk write back**  **Negedge clk read data** |
| **CONTROL** | **Make the Control signal** | **NONE** |
| **rst\_contoller** | **If there is any branch or load use data hazard happen**  **this module will enable flush the register.**  **The module will be more detailed discuss later.** | **NONE** |

|  |  |  |
| --- | --- | --- |
| **STAGE3 ALU EXECUTION** | | |
|  | | |
| **MODULE** | **FUNCTION** | **SPECIAL\_DESIGN** |
| **ALU\_IN** | **Data hazard signal will be the input module will decide the input of the alu.** | **NONE** |
| **ALU\_RD** | **calculate for the data store in $rd** | **NONE** |
| **ALU\_ADDR** | **calculate for the Data memory’s Address which will be stored data next clock.** | **NONE** |
| **PC\_ADDR** | **calculate the jumping address and make the signal to enable the system jump or not.** | **NONE** |
| **rst\_contoller** | **If there is any branch happen this module will enable flush the register. The module will be more detailed discuss later.** | **NONE** |

|  |  |  |
| --- | --- | --- |
| **STAGE4 WRITE OR READ MEMORY** | | |
|  | | |
| **MODULE** | **FUNCTION** | **SPECIAL\_DESIGN** |
| **DIVIDER4** | **Divide the address value by 4**  **Quotient will be the real address value of data memory**  **Reminder will be used for LB SB instruction to load and store data to the corresponding byte.** | **Using shift right 2 bits instead using the divider which can reduce hardware area.** |
| **LOW\_BYTE\_CONTROL** | **Control for LB SB instruction to load and store data to the corresponding byte.** | **NONE** |
| **WB\_CONTROL** | **Wb\_control signal will decide data from memrory and alu writing back to rd rigister** | **NONE** |

|  |  |
| --- | --- |
| **OTHER CONTROL MODULE** | |
| **MODULE** | **LOAD\_DATA\_HAZARD** |
|  | **FUNCTION** | **SPECIAL\_DESIGN** |
| **CONTROL FLOW** | |
| **Detect if there is any load use data hazard**  **pc\_stall\_stage1=**  (id\_exe\_read\_mem==1'b1 && (if\_id\_rs1\_addr==id\_exe\_rd\_addr || if\_id\_rs2\_addr==id\_exe\_rd\_addr))?1'b1:1'b0;  **Detect if there is any jump signal happened at the same time and prevent pc\_stall signal flush the jump action.**  **Jump signal has the priority**  pc\_st all=pc\_jump\_confirm?1'b0:pc\_stall\_stage1;  instruction\_stall=pc\_stall; | |
| **MODULE** | **FORWARDING UNIT** |
|  |
| **CONTROL FLOW** | |
| **Detect if there is any data hazard between memory and register $rs1**  rs1\_mem\_hazard=(mem\_wb\_write\_reg==1'b1 && mem\_wb\_rd\_addr!=5'd0 && mem\_wb\_rd\_addr==rs1\_addr)?1'b1:1'b0;  **Detect if there is any data hazard between ALU and register $rs1**  rs1\_exe\_hazard=(exe\_mem\_write\_reg==1'b1 && exe\_mem\_rd\_addr!=5'd0 && exe\_mem\_rd\_addr==rs1\_addr)?1'b1:1'b0;  **Detect if there is any data hazard between memory and register $rs2**  rs2\_mem\_hazard=(mem\_wb\_write\_reg==1'b1 && mem\_wb\_rd\_addr!=5'd0 && mem\_wb\_rd\_addr==rs2\_addr)?1'b1:1'b0;  **Detect if there is any data hazard between ALU and register $rs2**  rs2\_exe\_hazard=(exe\_mem\_write\_reg==1'b1 && exe\_mem\_rd\_addr!=5'd0 && exe\_mem\_rd\_addr==rs2\_addr)?1'b1:1'b0; | |

|  |  |
| --- | --- |
| **MODULE** | **rst\_controller**  **left side (for exe\_mem stage and if \_id stage)**  **right side (id\_exe stage)** |
|  |
| **CONTROL FLOW** | |
| **(for exe\_mem stage and if \_id stage)**  **Global rst is cpu’s rst ; jump and branch instruction will make [enable jump signal] active high;**  **if brach conditiction is established then pc\_jump control will active high enable to jump address.**  rst\_data=global\_rst?1'b1:( enable\_jump ? ( pc\_jump\_control ? local\_rst:1'b0) :1'b0);  **(for id\_exe stage)**  **1.Global rst is cpu’s rst ;**  **2.Modele will check the pc\_stall signal to decide whether pause the pipeline register or not.**  **Next, jump and branch instruction will make [enable jump signal] active high; if branch condition is**  **established then pc\_jump control will active high enable to jump address.**  rst\_data=global\_rst?1'b1: ( pc\_stall ? 1'b1: ( enable\_jump ? ( pc\_jump\_control ? local\_rst:1'b0) :1'b0) ); | |

|  |  |
| --- | --- |
| **MODULE** | **PC\_CONTROL** |
|  |
| **CONTROL FLOW** | |
| **pc\_stall signal will enable rewrite the instruction address again**  **enable\_jump signal will enable to jump address.**  **if(pc\_stall==1'b1)**  **pc\_data=pc;**  **else**  **pc\_data=enable\_jump?(pc\_jump\_control?pc\_jump\_address:next\_pc):next\_pc;** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **R-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **ADD** | | **Machine code** | **Assembly code** |
| **0x006282b3** | **add t0 t0 t1** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **control module(cl ) produce alu\_rd\_signal(function of the alu);active write\_reg signal; and register(rigt)read the data by the address decode by decoder**  **Decoder decode opcode:33** | | |
| **EXE stage** | **Alu\_in\_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu**  **Alu\_rd (ard)calculate the value that will store to the $rd address** | | |
| **MEM**  **stage** | **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **R-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **AND** | | **Machine code** | **Assembly code** |
| **0x0062f2b3** | **and t0 t0 t1** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **control module(cl ) produce alu\_rd\_signal(function of the alu);active write\_reg signal; and register(rigt)read the data by the address decode by decoder**  **Decoder decode opcode:33,funt3:7,fun7:0** | | |
| **EXE stage** | **Alu\_in\_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu**  **Alu\_rd\_control signal=9 execute and function**  **Alu\_rd (ard)calculate the value that will store to the $rd address** | | |
| **MEM**  **stage** | **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **I-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **LW** | | **Machine code** | **Assembly code** |
| **0x0002a283** | **Lw t0 0(t0)** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:3,funt3:2**  **Control module’output extended\_control control the imm data**  **Control module produce read mem signal** | | |
| **EXE stage** | **Alu\_addr calculate the address(alu\_addr\_out) that is going to load data.**  **Because thers is a load data hazard ,the pc\_stall signal active high to pause instruction.** | | |
| **MEM**  **stage** | **The address is divided by 4 to be the real data memory address**  **The OE signal is active high because it connects with read mem signal**  **Wb\_control control the value loaded from memory to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register**  **The valus is stored in register in 5 stage** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **I-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **ADDI** | | **Machine code** | **Assembly code** |
| **0xfff00293** | **addi t0 t0 -1** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:19funt3:0,alu\_control:10**  **Control module’output extended\_control control the imm data** | | |
| **EXE stage** | **Alu\_addr calculate the address(alu\_addr\_out) that is going to load data.**  **The calculation way is selected by alu\_control** | | |
| **MEM**  **stage** | **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register**  **The valus is stored in register in 5 stage** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **S-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **SW** | | **Machine code** | **Assembly code** |
| **0xfe542e23** | **Sw t0,-4(s0)** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:35funt3:2**  **Control module’output extended\_control control the imm data**  **Control module activate the write mem signal** | | |
| **EXE stage** | **Alu\_rd (ard)calculate the value that will store to the $rd address**  **The calculation way is selected by alu\_control** | | |
| **MEM**  **stage** | **The address is divided by 4 to be the real data memory address**  **The 4 bits signal is active low because it connects with read mem signal**  **The data from register is stored in negedge cycle** | | |
| **WB**  **stage** | **NO ACTION** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **S-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **SB** | | **Machine code** | **Assembly code** |
| **0xfe542e23** | **Sw t0,-4(s0)** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:35funt3:0**  **Control module’output extended\_control control the imm data**  **Control module activate the write mem signal**  **Control module activate the low byte write signal** | | |
| **EXE stage** | **Alu\_rd (ard)calculate the value that will store to the $rd address**  **The calculation way is selected by alu\_control** | | |
| **MEM**  **stage** | **The address is divided by 4 to be the real data memory address and the written bytes will selected by reminder.**  **The 4 bits signal is 1110 because the low byte control signal only allow write in the low 8bits data.**  **The data from register is stored in negedge cycle** | | |
| **WB**  **stage** | **NO ACTION** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **B-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **BEQ** | | **Machine code** | **Assembly code** |
| **0x02628863** | **Beq t0,t1 858** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:99funt3:0**  **Control module’output extended\_control control the imm data**  **Control module activate the enable jump signal**  **Control module control the alu\_pc function with alu pc control signal=1** | | |
| **EXE stage** | **Alu\_PC determine whether to jump or not. src1!=src2 in this section which will not activate jump signal**  **The next pc address isn’t jump** | | |
| **MEM**  **stage** | **NO ACTION** | | |
| **WB**  **stage** | **NO ACTION** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **B-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **BNE** | | **Machine code** | **Assembly code** |
| **0x 00531c63** | **Bne t1,t0,8b8** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:99funt3:1**  **Control module’output extended\_control control the imm data**  **Control module activate the enable jump signal**  **Control module control the alu\_pc function with alu pc control signal=2** | | |
| **EXE stage** | **Alu\_PC determine whether to jump or not. src1==src2 in this section which will not activate jump signal**  **The next pc address isn’t jump** | | |
| **MEM**  **stage** | **NO ACTION** | | |
| **WB**  **stage** | **NO ACTION** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **U-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **AUIPC** | | **Machine code** | **Assembly code** |
| **0x 00000297** | **Auipc t0 0x0** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:23**  **Control module’output extended\_control control the imm data**  **Control module enable the write reg signal**  **Control module control the alu\_ed function with alu rd control signal=20** | | |
| **EXE stage** | **Alu\_in\_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu**  **Alu\_rd\_control signal=9 execute and function**  **Alu\_rd (ard)calculate the value that will store to the $rd address** | | |
| **MEM**  **stage** | **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register**  **The valus is stored in register in 5 stage** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **U-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **LUI** | | **Machine code** | **Assembly code** |
| **0x 135793b7** | **Lui t2 0x13579** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID stage** | **Decoder decode opcode:55**  **Control module’output extended\_control control the imm data**  **Control module activate the write reg signal**  **Control module control the alu\_rd function with alu rd control signal=21** | | |
| **EXE stage** | **Alu\_in\_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu**  **Alu\_rd\_control signal 21 execute and function**  **Alu\_rd (ard)calculate the value that will store to the $rd address** | | |
| **MEM**  **stage** | **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register**  **The valus is stored in register in 5 stage** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **J-TYPE INSTRUCTION VERIFICATION** | | | |
| **INSTRUCTION**  **JAL** | | **Machine code** | **Assembly code** |
| **0x0080036f** | **Jal t1,b10** |
| **wave** | | | |
|  | | | |
| **Explanation** | | | |
| **ID**  **stage** | **Decoder decode opcode:111funt3**  **Control module’output extended\_control control the imm data**  **Control module activate the write reg signal Control module control the alu\_rd function with alu rd control signal=14**  **Control module control the alu\_rd function with alu pc control signal=7**  **Control module activate the enable jump signal** | | |
| **EXE**  **stage** | **Alu\_in\_selector detect the hazard signal produced by forwarding unit and choose the data in memory or alu**  **Alu\_rd\_control signal=14 execute and function**  **Alu\_rd (ard)calculate the value that will store to the $rd address** | | |
| **MEM**  **stage** | **Alu\_PC determine to jump to the address pc+immin the next cylce**  **By pass the value calculate by Alu\_rd**  **Wb\_control control the value calculate by Alu\_rd to be the write back data.** | | |
| **WB**  **stage** | **Write\_reg signal enable to write back data in register**  **The valus is stored in register in 5 stage** | | |

|  |  |  |
| --- | --- | --- |
| **SUPER LINT ANALYZE** | | |
| **TOTAL LINES:1943** | | |
|  | | |
| **WARNING:62** | | **1-62/1943\*100%=97%** |
| **MOST FREQUENT WARNING/ERRORS IN YOUR CODE** | | |
| **Module name not comparing** | **C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\01.png** | |
| **Modify** | **For example change cpu\_rtl.sv to cpu\_rtl** | |
| **Default statement isn’t require** | **C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\03.png** | |
| **Modify** | **If the case statement is full case delete the default case** | |
| **After fixing the problem** | | |
| **WARNING58** | | **1-58/1943\*100%=98%** |

|  |
| --- |
| **LESSON I LEARNED** |
| **經由這次的作業 除了複習以前學過的計算機組織架構,也體認到實作時 考慮控制的因素會非常多,例如當我 compile 已經過program0 卻無法過program1 的原因在於當我執行jump指令時,若前一刻有load data hazard 的出現則可能會覆蓋掉 jump訊號 經由修改過後 就可以compile過了,這時我之前學習計組時 沒有想到的問題,而恰巧program0 沒有測試到 LW 接續 JUMP 指令的pattern 導致會有 program 0 過 program 1 沒過的情形發生。此次實作 讓我對CPU架構更為熟悉 獲益良多** |