CS202 project

Members

Name	ID	work	contribution
贾禹帆	12111224	uart, memory(data and instruction) and top module	50%
汤嘉阳	12110715	instruction fetech, instruction decoder, ALU module	50%
李子阳	12110228	assembly code	0%

CPU architecture

- I type

1. lui
2. andi
3. ori
4. addi
5. lw
6. sw
7. addiu
8. xori
9. slti
10. beq
11. bne
12. slti
13. sltiu

- R type

1. and
2. or
3. add
4. addu
5. xor
6. nor
7. sub
8. subu
9. slt
10. sltu
11. sll
12. srl
13. sllv
14. srlv
15. sra
16. srav

- J type

1. jal
2. jr
3. j

Registers

- 32 * 32bits registers

Address

- harvard structure
- byte unit addressing
- Total size of data memory&instruction memory = 16384bits

IO

- MMIO with polling:
  - 0xFFFF_FC60: led IO address, each of low 23 bits control one led's on or off
  - 0xFFFF_FC70: switch IO address, each of low 23 bits represent one switch's status

CPI

- single cycle, clock cycle is 20MHz

Interface

- clock: use the 100MHz crystal oscillator binding on minisys Y18 pin and do clock divide:
  - cpu use 20MHz
  - uart use 10MHz
- reset: bind minisys rst button(on pin P20) as global positive enabled rst signal
- uart: use given uart ip core

Structure

- module connection: show in file `./connection.pdf`
- module design:

```
// memory
module data_mem(input clk,    // cpu clock
                rst_n,     // reset signal
                mem_write,              // 0 read 1 write
                input[22:0]switch_in, // switch input signal
                input[31:0]address,  // memory address input
                input[31:0]data_in,  // input data in(for mem write)
                output[31:0]data_out, // data output
                output [22:0]led_out, // led signal output
                output [63:0]seg_out, // output for 7-segments tube
                input upg_rst_i,  // uart rst signal
                input upg_clk_i,  // uart clock
                input upg_wen_i,  // uart write enable
                input [13:0] upg_adr_i, // uart write address
                input [31:0] upg_dat_i, // uart data input
                input upg_done_i);  // uarte write done

// instruction fetech
module Ifetc32(input clock,     //cpu clock
    reset,      //reset signal
    input [31:0] Addr_result, // the calculated address from ALU
    input Zero, // while Zero is 1, it means the ALUresult is zero
    input [31:0] Read_data_1,
    input Branch, // while Branch is 1,it means current instruction is beq
    input nBranch, // while nBranch is 1,it means current instruction is bnq
```

```
    input Jmp, // while Jmp 1, it means current instruction is jump
    input Jal,  // while Jal is 1, it means current instruction is jal
    input Jr, // while Jr is 1, it means current instruction is jr
    output[31:0] Instruction, // the instruction fetched from this module
    output[31:0] branch_base_addr, // (pc+4) to ALU which is used by branch type inst
    output reg [31:0] link_addr, // (pc+4) to Decoder which is used by jal instructio
    input upg_rst_i,  // uart rst signal
    input upg_clk_i,  // uart clock
    input upg_wen_i,  // uart write enable
    input [13:0] upg_adr_i, // uart write address
    input [31:0] upg_dat_i, // uart data input
    input upg_done_i);  // uarte write done

module executs32(
input[31:0] Read_data_1; // from Read_data_1 of the decoding unit
input[31:0] Read_data_2; // from Read_data_2 of the decoder unit
input[31:0] Sign_extend; // the immediate number of extensions from the decoder unit
input[5:0] Function_opcode; // The function code of the r-form instruction from the j
input[5:0] Exe_opcode; // the opcode from the finger unit
input[1:0] ALUOp; // control code of the operation instruction from the control unit
input[4:0] Shamt; // instruction[10:6] from the fetch unit, specifying the number of
input Sftmd; // from the control unit, indicating that it is a shift instruction
input ALUSrc; // from the control unit, indicating that the second operand is an imme
input I_format; // from the control unit, indicating that it is an I-type instructior
input Jr; // from the control unit, indicating a JR instruction
output Zero; // A value of 1 indicates that the calculated value is 0
output[31:0] ALU_Result; // Calculated data result
output[31:0] Addr_Result; // Calculated address result
input[31:0] PC_plus_4); // PC+4 from the fetch unit

module decode32(
output[31:0] read_data_1, //first operand number
output[31:0] read_data_2, //second operand number
input[31:0] Instruction, // The instruction fetched
input[31:0] mem_data, // DATA taken from the DATA RAM or I/O port for writing data to
input[31:0] ALU_result, // The result of an operation from the Arithmetic logic unit
input Jal, // From the control unit Controller, when the value is 1, it indicates tha
input RegWrite, // From the control unit Controller, when the value is 1, do register
input MemtoReg, // From the control unit Controller, indicating that DATA is written
input RegDst,//Control write to which specified register in instructions. when RegDst
output[31:0] Sign_extend, //immediate number after extend
input   clock, //cpu clock
reset,// reset signal
input[31:0] opcplus4); // The JAL instruction is used to write the return address to
//the $ra register, what we have got here is PC + 4

module control32(
input[5:0]   Opcode,            // instruction[31..26] - the high 6 bits of the instr
input[5:0]   Function_opcode,   // instructions[5..0] - the low 6 bits of the instruc
```

```
output Jr,// 1 if the current instruction is jr, 0 otherwise
output RegDST,// 1 if the destination register is rd, 0 if the destination register i
output ALUSrc,// 1 if the second operand (Binput in ALU) is an immediate value (excep
output MemtoReg,// 1 if data written to register is read from memory or I/O, 0 if dat
output RegWrite,// 1 if the instruction needs to write to register, 0 otherwise
output MemWrite,//  1 if the instruction needs to write to memory, 0 otherwise
output Branch,//  1 if the current instruction is beq, 0 otherwise
output nBranch,//  1 if the current instruction is bne, 0 otherwise
output Jmp,//  1 if the current instruction is j, 0 otherwise
output Jal,//  1 if the current instruction is jal, 0 otherwise
output I_format,//  1 if the instruction is I-type except for beq, bne, lw and sw; 0
output Sftmd,//  1 if the instruction is a shift instruction, 0 otherwise
output[1:0]  ALUOp,// if the instruction is R-type or I_format, ALUOp's higher bit is
Problem Information);
```

Test

test name	methord	category	result
ifetc test	oj test	unit test	passed
control test	oj test	unit test	passed
decode test	oj test	unit test	passed
execut test	oj test	unit test	passed
memory&io test	vivado testbeanch simlution	unit test	passed
top test	vivado testbeanch simlution	integrate test	passed
top onboard test	on board test	integrate test	passed
uart test	on board test	integrate test	passed
code test	run assembly code on cpu and show the result	integrate test	passed

Conclusion

- If the code run well on MARS, then there must be something wrong in hardware
- Vivado simulation save test time