CS334 计算机组成实验

Lab6

简单的类 MIPS 单周期处理器

周鼎
5140219268

指导老师：王老师

April 7, 2016

1 顶层结构

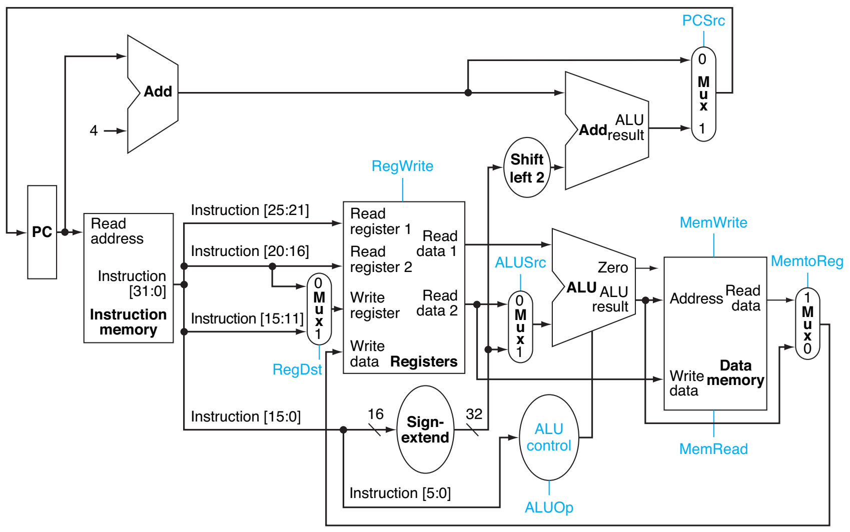

Figure 1: 单周期 MIPS 顶层结构

注：jump 指令相应的 data path 将在后续添加。

2 PC

PC 是一个十分简单然而特别关键的模块：它只需完成在时钟上升沿更新 PC、在随后一个周期中保持的功能；它又是整个设计中程序能否正确执行和跳转的关键, 要注意需要初始置零（也可通过运行时先加 reset 信号实现）。实现的代码和仿真结果见 PC.v 和 Figure2

```
module Pc(
    input clock_in ,
    input [31:0] nextPC,
    output[31:0] currPC,
    input rst
    );
   reg [31:0] PCFile;
   initial PCFile <= 0;

   always @(posedge clock_in)  begin
    if(rst) PCFile <= 0;
    else PCFile <= nextPC;
   end

   assign currPC = PCFile;

endmodule
```

PC.v

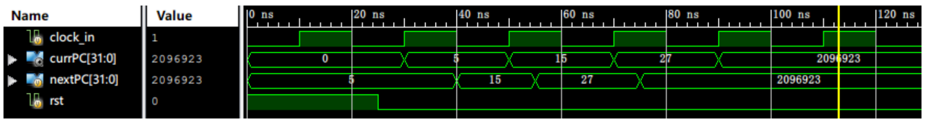

Figure 2: PC 行为仿真

3 InstrMem

处理器执行的指令需要装载在 InstrMen 中，因其十分简单直接在 Top 中实现，不另设计模块。注意指令为 32 位，按 word 读取将 PC 右移 2 位。

```
    //define AND initial Instruction Memory
  reg [31:0] InstrMemory[9:0];
  initial $readmemb("./src/inst.txt",InstrMemory);
    wire [31:0] INSTR;
    wire [31:0] CURR_PC;
  assign INSTR = InstrMemory[CURR_PC>>2];  //fetch instruction by PC
```

instrMem.v

4 signExtender

符号扩展用于 I-type 指令扩展其后 16 位，可简单调用系统任务实现。

```
    wire [31:0] extRes;
    assign extRes = $signed(INSTR[15:0]);
```

signExtender.v

5 MUX

MUX 可通 “MUX=Sel ?Opt1:Opt2” 实现。

```
  assign MEM_REG_MUX = MEM_TO_REG ? MEM_READ_DATA : ALU_RES;
  //R-type or load word
  assign REG_ALU_MUX = ALU_SRC ? EXTENDED_RES : REG_DATA_2;
  //which one to be used by ALU as the 2nd src
  assign REG_WRITE_ADDRESS = REG_DST ? INSTR[15:11] : INSTR[20:16];
  //which reg will be written
```

MUXes.v

6 nextPC

PC 更新有三种情况：

```
    PC <= PC + 4;
    PC <= (INSTR[15:0]<<2) + PC + 4;              //BEQ
    PC <= {INSTR[31:28],((PC + 4)<<2)[27:0]};     //JUMP
```

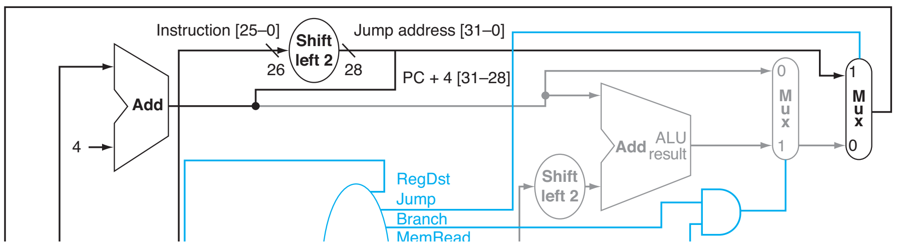

Figure 3: nextPC 生成

```
wire [31:0] JUMP_ADDRESS;
wire [31:0] BEQ_ADDRESS;
//assign JUMP_ADDRESS
assign JUMP_ADDRESS[31:28] = PCp4[31:28];
assign JUMP_ADDRESS[27:2]  = INSTR[25:0];
assign JUMP_ADDRESS[1:0]   = 'b00;
//assign BEQ_ADDRESS
assign BEQ_ADDRESS[31:0] = (EXTENDED_RES<<2) + PCp4;

wire [31:0] PC_SRC_SEL;
  wire [31:0] NEXT_PC;
//wire [31:0] PCp4;
//wire [31:0] CURR_PC;  //declared above;
wire PC_SRC;
assign PC_SRC = BRANCH & ZERO;
assign PCp4[31:0] = CURR_PC[31:0] + 4;
//serval MUX below
assign PC_SRC_SEL[31:0] = PC_SRC ? BEQ_ADDRESS[31:0]: PCp4[31:0];
assign NEXT_PC[31:0] = JUMP ? JUMP_ADDRESS[31:0] : PC_SRC_SEL[31:0];
```

nextPC.v

7 信号线和模块实例化

对连线系统的命名有助于连线不重不漏，也便于实例化模块。

```
//                    main control signal                    //
  wire REG_DST,
       JUMP,
       BRANCH,
       MEM_READ,
       MEM_TO_REG,
       MEM_WRITE,
       ALU_SRC,
       REG_WRITE,
       ZERO;  //generated by ALU
  wire [1:0]  ALU_OP;
  wire [3:0]  ALU_CTR;

//                    data bus                    //

  wire [31:0] REG_DATA_1;
  wire [31:0] REG_DATA_2;
    //reg read output
```

```
wire  [4:0]   REG_WRITE_ADDRESS;
  //reg write address, the data will be from the MEM_REG_MUX
wire  [31:0] ALU_RES;
wire  [31:0] EXTENDED_RES;
wire  [31:0] MEM_READ_DATA;   //address specified by alu result

wire  [31:0] MEM_REG_MUX;
  //to be written into reg(memomy or alu result)
wire  [31:0] REG_ALU_MUX;
  //which one to be used by ALU as the 2nd src(rt or imm)
```

wire.v

```
Pc mainPC(
      .clock_in(CLOCK_IN),
      .nextPC(NEXT_PC),
      .currPC(CURR_PC),
    .rst(RESET)
    );
Ctr mainCtr(
    .opCode(INSTR[31:26]),
    .regDst(REG_DST),
    .aluSrc(ALU_SRC),
    .memToReg(MEM_TO_REG),
    .regWrite(REG_WRITE),
    .memRead(MEM_READ),
    .memWrite(MEM_WRITE),
    .branch(BRANCH),
    .aluOp(ALU_OP),
    .jump(JUMP)
);
AluCtr mainAluCtr (
    .aluOp(ALU_OP),
    .funct(INSTR[5:0]),
    .aluCtr(ALU_CTR)
);
Alu mainAlu (
    .input1(REG_DATA_1),
    .input2(REG_ALU_MUX),
    .aluCtr(ALU_CTR),
    .zero(ZERO),
    .aluRes(ALU_RES)
);
signExt mainSignExt (
    .inst(INSTR[15:0]),
    .data(EXTENDED_RES)
);
dataMemory mainDataMemory (
    .clock_in(CLOCK_IN),
    .address(ALU_RES),
    .writeData(REG_DATA_2),
    .readData(MEM_READ_DATA),
    .memWrite(MEM_WRITE),
    .memRead(MEM_READ)
);
Register mainRegister (
    .clock_in(CLOCK_IN),
    .regWrite(REG_WRITE),
    .readReg1(INSTR[25:21]),
    .readReg2(INSTR[20:16]),
    .writeReg(REG_WRITE_ADDRESS),
    .writeData(MEM_REG_MUX),
        .reset(RESET),
        .readData1(REG_DATA_1),
        .readData2(REG_DATA_2),
        .reg1(reg1),
        .reg2(reg2)
    );
endmodule
```

instances.v

8 ModelSim 仿真

```
00001000000000000000000000000010
00000000000000000000000000000000
10001100000000000100000000000000
10001100000000100000000000000100
00000000001000100000100000100000
00000000001000100001000000100000
00010000000000001111111111111101
```

```
j 2
nop
lw   $1, (0)$0;
lw   $2, (4)$0;
AGAIN:   ADD $1, $1, $2;
         ADD $2, $1, $2;
beq $0, $0, AGAIN;
```

Figure 4: 二进制指令

Figure 5: MIPS 指令

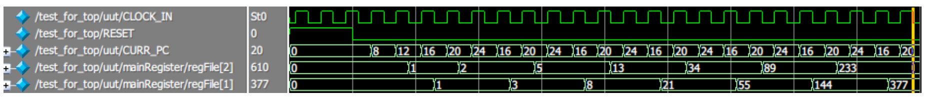

Figure 6: ModelSim 仿真波形

9 上板输入输出实现方案

9.1 IO Scheme

SW[2:0]	MODE
000	led[5:3] 为 $2,led[2:0] 为 $1
001	led 显示 $1
011	led 显示 $2
111	led 显示 PC

设计使用 8 个 LED 作为输出，4 个开关和 1 个复位式按钮作为输入，具体功能见下图。因板上接口限制，将只对输出寄存器 $1,$2 和 PC, 通过 SW[2:0] 选择输出模式，见左图。

9.2 IO 控制逻辑

```
////            GENRERATING SLOW CLOCK            ////
    wire CLOCK_IN;
    reg [26:0] Buffer = 0;
    always@ (posedge CLOCK) Buffer = Buffer + 1;
    assign CLOCK_IN = FAST ? CLOCK : Buffer[26];

```

IO Port	Function
LED[7]	查看 clock 信号
LED[6]	查看 reset 信号
LED[5:0]	查看寄存器数值
SW[3]	调整时钟快/慢模式
SW[2:0]	设置寄存器数值输出模式
Button	输入 reste 信号

Table 1: IO Scheme

```verilog
////                IO MODE SEL                              ////
   assign LED[7] = RESET;
   assign LED[6] = CLOCK_IN;
   wire [5:0] OUTPUT;
   assign LED[5:0] = OUTPUT;
   wire [31:0] reg1;
   wire [31:0] reg2;
   wire [5:0]  reg12;
   assign reg12[5:3] = reg2;
   assign reg12[2:0] = reg1;
   wire [31:0] CURR_PC_IO;
   assign CURR_PC_IO = CURR_PC>>2;
   wire [5:0] TEMP1;
   wire [5:0] TEMP2;
   assign TEMP1 = MODE[0] ? reg1:reg12;
   assign TEMP2 = MODE[1] ? reg2:TEMP1;
   assign OUTPUT = MODE[2]? CURR_PC_IO:TEMP2;
```

IO.v

9.3 User Constraint File

```
NET "CLOCK"    LOC = C9;
NET "LED[0]"   LOC = F12;
NET "LED[1]"   LOC = E12;
NET "LED[2]"   LOC = E11;
NET "LED[3]"   LOC = F11;
NET "LED[4]"   LOC = C11;
NET "LED[5]"   LOC = D11;
NET "LED[6]"   LOC = E9;
NET "LED[7]"   LOC = F9;
NET "MODE[0]"  LOC = L13;
NET "MODE[1]"  LOC = L14;
NET "MODE[2]"  LOC = H18;
NET "FAST"     LOC = N17;
NET "RESET"    LOC = K17 | IOSTANDARD = LVTTL | PULLDOWN;
```

top.ucf