Project3 Part4实验报告

——用VerilogHDL完成单周期处理器开发

55201128 胡剑铮

一、实验内容及要求

1.设计的处理器应支持指令集MIPS-Lite3：addu，subu，ori，lw，sw，beq，lui，addi，addiu， slt，j，jal，jr。其中addu, subu, addi, addiu可以不支持实现溢出

2.处理器为单周期设计

二、 设计要求

3. 单周期处理器由 datapath(数据通路)和 controller(控制器)组成。

a) 数据通路由如下 module组成：PC(程序计数器)、NPC(NextPC计算单元)、

GPR (通用寄存器组，也称为寄存器文件、寄存器堆)、ALU(算术逻辑单

元)、EXT(扩展单元)、IM(指令存储器)、DM(数据存储器)。

b) IM：容量为 4KB(32bit×1024 字)。

c) DM：容量为 4KB(32bit×1024 字)。

1. 具体设计及实现
2. 总体结构

总体结构代码如下图所示，各wire变量与其同名的模块接口参数连接。

整体结构由控制器controller，运算器ALU，数据存储器DM，位扩展组件EXT，寄存器堆GPR，指令存储器IM，程序计数器PC运算组件构成。其中ALU的输入端和PC运算组件的输出端各设一个多路选择器，分别为mux\_ALU和mux\_instr，GPR的输入端设两个多路选择器，分别用于读和写寄存器堆时的参数选择，为mux\_GPR和mux\_write。PC运算组件主体为用于计算下一个PC的值的NPC。

1. module main;
2. wire[15:0] imm;
3. wire[31:2] next\_pc, pc;
4. wire[4:0] rs, rt, rd, A1, A2, A3;
5. wire[11:2] add\_dm;
6. wire[31:0] dm\_in, dm\_out, im\_out, sum, alu\_out, ex\_imm, b;
7. wire[31:0] read\_data1, read\_data2, write\_data;
8. wire[11:2] add\_im;
9. wire[5:0] func,op;
10. wire[1:0] aluop;
11. wire[25:0] im\_outs;
12. wire[31:28] pc\_s;
13. wire[31:2] pc\_plus, instr;
14. wire zero, write\_reg, alu\_sel, ifbeq;
15. wire in, out, dest, extop, sign, set\_GPR, J, JR, JAL;
16. reg clk,reset;
18. controller controller(.func(func), .op(op), .aluop(aluop), .write\_reg(write\_reg),
19. .alu\_sel(alu\_sel), .ifbeq(ifbeq), .in(in), .out(out), .dest(dest),
20. .extop(extop), .sign(sign), .set\_GPR(set\_GPR), .J(J), .JAL(JAL), .JR(JR));
21. ALU ALU(.a(read\_data1), .b(b), .op(aluop), .sum(alu\_out));
22. NPC NPC(.ifbeq(ifbeq), .alu\_out(alu\_out), .now\_pc(pc), .next\_pc(next\_pc), .offset(imm),
23. .J(J), .instr(instr));
24. PC PC(.next\_pc(next\_pc), .reset(reset), .clk(clk), .pc(pc));
25. IM IM(.add\_im(add\_im), .im\_out(im\_out));
26. mux\_GPR mux\_GPR(.rt(rt), .rd(rd), .dest(dest), .A3(A3));
27. GPR GPR(.read\_reg1(A1), .read\_reg2(A2), .write\_reg(A3), .write\_data(write\_data),
28. .JAL(JAL), .op(write\_reg), .set\_GPR(set\_GPR), .clk(clk),
29. .read\_data1(read\_data1), .read\_data2(read\_data2), .pc\_plus(pc\_plus));
30. EXT EXT(.extop(extop), .sign(sign), .a(imm), .res(ex\_imm));
31. mux\_ALU mux\_ALU(.read\_data2(read\_data2), .ex\_imm(ex\_imm),
32. .alu\_sel(alu\_sel), .b(b));
33. DM DM(.add\_dm(add\_dm), .dm\_in(dm\_in), .clk(clk),
34. .in(in), .out(out), .dm\_out(dm\_out));
35. mux\_write mux\_write(.alu\_out(alu\_out), .out(out),
36. .dm\_out(dm\_out), .write\_data(write\_data));
37. mux\_instr mux\_instr(.im\_outs(im\_outs), .pc\_s(pc\_s), .JR(JR),
38. .instr(instr), .read\_data1(read\_data1));
40. assign rs = im\_out[25:21];
41. assign rt = im\_out[20:16];
42. assign rd = im\_out[15:11];
43. assign op = im\_out[31:26];
44. assign func = im\_out[5:0];
45. assign imm = im\_out[15:0];
46. assign A1 = rs;
47. assign A2 = rt;
48. assign add\_im = pc[11:2];
49. assign add\_dm = alu\_out[11:2];
50. assign dm\_in = read\_data2;
51. assign im\_outs = im\_out[25:0];
52. assign pc\_s = pc[31:28];
53. assign pc\_plus = pc + 1;
55. initial
56. begin
57. clk = 0;
58. reset = 1;
59. #6 reset = 0;
60. #100 $finish;
61. end
63. always
64. begin
65. #5 clk = ~clk;
66. end
68. endmodule
69. 算数逻辑单元ALU
70. `timescale 1ns / 1ps
72. module ALU(
73. input[31:0] a, b,
74. input[1:0] op,
75. output reg[31:0] sum
76. );
78. always @(a or b or op)
79. begin
80. **if** (op == 3)
81. begin
82. sum = a + b;
83. end
84. **else** **if** (op == 2)
85. begin
86. sum = a + (~b) + 1;
87. end
88. **else**
89. begin
90. sum = a | b;
91. end
92. end
93. endmodule

根据2bit位的运算指令参数op决定执行哪种运算，而运算参数由多路选择器mux\_ALU决定：

1. `timescale 1ns / 1ps
3. module mux\_ALU(
4. input[31:0] read\_data2, ex\_imm,
5. input alu\_sel,
6. output reg[31:0] b
7. );
9. always @(read\_data2 or ex\_imm)
10. begin
11. **if** (alu\_sel == 1) b = ex\_imm;
12. **else** b = read\_data2;
13. end
14. endmodule

若alu\_sel为1则第二个运算参数为扩展立即数，否则为从寄存器堆中读取的read\_data2。

1. 扩展单元EXT
2. `timescale 1ns / 1ps
4. module EXT(
5. input extop, sign,
6. input[15:0] a,
7. output reg[31:0] res
8. );
10. always @(a or sign or extop)
11. begin
12. **if** (extop == 0) res = a << 16;
13. **else** **if** (extop == 1 && sign == 0) res = a;
14. **else** res = $**signed**(a);
15. end
16. endmodule

若extop为0则扩展至高16位（lui命令），若sign为1则扩展位填充为符号位（LW，SW命令）。

1. 寄存器堆GPR
2. `timescale 1ns / 1ps
4. module GPR(
5. input[4:0] read\_reg1, read\_reg2, write\_reg,
6. input[31:0] write\_data,
7. input op, clk, set\_GPR, JAL,
8. input[31:2] pc\_plus,
9. output reg[31:0] read\_data1, read\_data2
10. );
11. reg[31:0] gpr[31:0];
13. always @(read\_reg1 or read\_reg2)
14. begin
15. read\_data1 = gpr[read\_reg1];
16. read\_data2 = gpr[read\_reg2];
17. end
19. always @(posedge clk)
20. begin
21. **if** (op == 1)
22. begin
23. **if** (set\_GPR == 0)
24. **if** (JAL == 1) gpr[31] = pc\_plus;
25. **else** gpr[write\_reg] = write\_data;
26. **else**
27. **if** (write\_data[31] == 1) gpr[write\_reg] = 1;
28. **else** gpr[write\_reg] = 0;
29. end
30. end
32. initial
33. begin: seq\_blk\_a
34. integer i;
35. **for** (i = 0; i < 32; i += 1) gpr[i] = 0;
36. end
37. endmodule

由read\_reg1和read\_reg2确定读取的寄存器序号，op为1时写寄存器，若为JAL指令则将(PC+4)存入GPR[31]中，若set\_GPR为1则寄存器根据write\_data的正负置位（slt指令）。

同时，GPR输入端设两个多路选择器，一个是确定读参数的mux\_GPR，一个是

mux\_GPR：

1. `timescale 1ns / 1ps
3. module mux\_GPR(
4. input[4:0] rt, rd,
5. input dest,
6. output reg[4:0] A3
7. );
9. always @(rt or rd or dest)
10. **if** (dest == 1) A3 = rd; **else** A3 = rt;
11. endmodule

mux\_write：由out信号决定使用DM的输出还是ALU的输出。

1. `timescale 1ns / 1ps
3. module mux\_write(
4. input out,
5. input[31:0] alu\_out,
6. input[31:0] dm\_out,
7. output reg[31:0] write\_data
8. );
10. always @(alu\_out or dm\_out or out)
11. **if** (out == 1) write\_data = dm\_out; **else** write\_data = alu\_out;
12. endmodule
13. 数据存储器DM

在时钟上升沿@posedge clk根据信号输入或输出DM。

1. `timescale 1ns / 1ps
3. module DM(
4. input[11:2] add\_dm,
5. input[31:0] dm\_in,
6. input clk, in, out,
7. output reg[31:0] dm\_out
8. );
9. reg[31:0] dm[1023:0];
11. always @(add\_dm or out)
12. **if** (out) dm\_out = dm[add\_dm];
14. always @(posedge clk)
15. begin
16. **if** (in == 1) dm[add\_dm] = dm\_in;
17. **if** (out == 1) dm\_out = dm[add\_dm];
18. end
20. initial
21. begin: seq\_blk\_a
22. integer i;
23. **for** (i = 0; i < 1024; i += 1) dm[i] = 0;
24. dm[1] = 1;
25. dm[2] = 1;
26. end
27. endmodule
28. PC及其运算组件NPC

PC：由NPC的输出next\_pc决定下一个PC的值，当reset信号为高电平时清零。

1. `timescale 1ns / 1ps
3. module PC(
4. input[31:2] next\_pc,
5. input reset, clk,
6. output reg[31:2] pc
7. );
9. always @(posedge clk)
10. begin
11. **if** (reset == 1) pc = 30'h00003000;
12. **else** pc = next\_pc;
13. end
14. endmodule

NPC：执行J指令时直接将立即数赋给next\_pc，ifbeq指令时根据ALU的输出确定是否跳转，其余情况PC正常+1。

1. `timescale 1ns / 1ps
3. module NPC(
4. input ifbeq,
5. input[31:0] alu\_out,
6. input[31:2] now\_pc,
7. input[15:0] offset,
8. input J,
9. input[31:2] instr,
10. output reg[31:2] next\_pc
11. );
13. reg[29:0] tmp;
14. always @(ifbeq or alu\_out or now\_pc or offset or J or instr)
15. begin
16. **if** ((ifbeq == 0) && (J == 0)) next\_pc = now\_pc + 1;
17. **else** **if** (J == 1) next\_pc = instr;
18. **else** **if** (alu\_out == 0)
19. begin
20. tmp = $**signed**(offset);
21. next\_pc = tmp + now\_pc;
22. end
23. **else** next\_pc = now\_pc + 1;
24. end
25. endmodule

多路选择器mux\_instr确定生成地址由寄存器堆读取的数直接指定（JR）还是由立即数和PC拼凑得到（J）。

1. `timescale 1ns / 1ps
3. module mux\_instr(
4. input[25:0] im\_outs,
5. input[31:28] pc\_s,
6. input[31:0] read\_data1,
7. input JR,
8. output reg[31:2] instr
9. );
11. always @(im\_outs or pc\_s or read\_data1 or JR)
12. begin
13. **if** (JR == 1) instr = read\_data1[31:2];
14. **else** instr = {pc\_s, im\_outs, 2'b00};
15. end
16. endmodule
17. 指令存储器IM
18. `timescale 1ns / 1ps
20. module IM(
21. input[11:2] add\_im,
22. output reg[31:0] im\_out
23. );
24. reg[31:0] im[1023:0];
26. always @add\_im
27. begin
28. im\_out = im[add\_im];
29. end
31. endmodule
32. 控制器controller

主要负责将指令转换成各种信号，从而使各模块正常运行并完成相应指令功能。

1. `timescale 1ns / 1ps
3. module controller(
4. input[5:0] func,op,
5. output reg[1:0] aluop,
6. output reg write\_reg, alu\_sel, ifbeq, in, out, dest,
7. output reg extop, sign, set\_GPR, J, JAL, JR
8. );
9. reg addu, subu, ori, lw, sw, beq, lui, addi, slt, j, jal, jr;
11. always @(func or op)
12. begin
13. addu = 0; addi = 0; subu = 0; ori = 0;
14. lw = 0; sw = 0; beq = 0; lui = 0;
15. slt = 0; j = 0; jal = 0; jr = 0; extop = 1; sign = 1; set\_GPR = 0;
17. **if** ((op == 6'b0) && (func == 6'b100001)) addu=1;
18. **else** **if** ((op == 6'b0) && (func == 6'b100011)) subu=1;
19. **else** **if** ((op == 6'b001101)) ori=1;
20. **else** **if** ((op == 6'b100011)) lw=1;
21. **else** **if** ((op == 6'b101011)) sw=1;
22. **else** **if** ((op == 6'b000100)) beq=1;
23. **else** **if** ((op == 6'b001111)) lui=1;
24. **else** **if** ((op == 6'b001000) || (op == 6'b001001)) addi=1;
25. **else** **if** ((op == 6'b000000) || (func == 6'b101010)) slt=1;
26. **if** (op == 6'b000010) j=1;
27. **if** (op == 6'b000011) jal=1;
28. **if** ((op == 6'b000000) && (func == 6'b001000)) jr=1;
29. **if** (lui == 1) sign=0;
30. **if** (lui == 1) extop=0;
31. **if** ((addu == 1) || (subu == 1) || (slt == 1)) dest=1;
32. **if** ((addu == 1) || (subu == 1) || (lui == 1) || (lw == 1) || (ori == 1)
33. || (addi == 1) || (jal == 1) || (slt == 1)) write\_reg=1;
34. **if** ((lui == 1) || (ori == 1) || (lw == 1) || (sw == 1) || (addi == 1)) alu\_sel=1;
35. **if** (beq == 1) ifbeq=1;
36. **if** (sw == 1) in=1;
37. **if** (lw == 1) out=1;
38. **if** ((addu == 1) || (lw == 1) || (sw == 1) || (addi == 1)) aluop=3;
39. **if** ((subu == 1) || (beq == 1) || (slt == 1)) aluop=2;
40. **if** (ori == 1) aluop=1;
41. **if** (slt == 1) set\_GPR=1;
42. **if** ((j == 1) || (jal == 1) || (jr == 1)) J=1;
43. **if** (jal == 1) JAL=1;
44. **if** (jr == 1) JR=1;
45. end
46. endmodule

四、 程序测试

LUI $t1, 0

ORI $t1, 1 # $t1 赋值为 1, 表示循环变量 i

LUI $t2, 0

ORI $t2, 11 # $t2 赋值为 11, 表示循环的上限 n + 1

LUI $t3, 0

ORI $t3, 0 # $t3 赋值为 0，用来储存结果 ans

LUI $t4, 0

ORI $t4, 1 # $t4 赋值为 1, 表示循环的增量 delta

loop\_begin:

BEQ $t1, $t2, loop\_end # if i >= n + 1 goto loop\_end

ADDU $t3, $t3, $t1 # ans = ans + i

ADDU $t1, $t4, $t1 # i = i + delta

BEQ $zero, $zero, loop\_begin # goto loop\_begin

loop\_end:

SW $t3, 8($zero) # Mem[8] = ans

通过MARS将以上程序转成机器码：

3c090000

35290001

3c0a0000

354a000b

3c0b0000

356b0000

3c0c0000

358c0001

112a0003

01695821

01894821

1000fffc

ac0b0008

再将以上机器码存于code.txt，再读进IM后对其进行测试得到结果。

该测试结果与预期结果相同。