Project3 Part5, 6, 7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. 处理器支持 MIPS-C4 指令集。

MIPS-C4={ LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、

SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、

SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、

XORI、LUI、SLTI、SLTIU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、 J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO }。

2. 所有运算类指令均不支持溢出。

3. 所有指令均不支持异常。

4. 处理器为多周期设计。

二、 设计要求

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

a) 数据通路包括如下 module：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)等。

2. 多周期数据通路包括 PC、NPC、IM、DM 这 4 个独立模块。其中：

a) IM：容量为 4KB(32bit×1024 字)。

b) DM：容量为 4KB(32bit×1024 字)。

3. 层次及模块实例化命名满足：

a) 本 project 的顶层设计文件命名：mips.v。

b) PC 必须被实例化命名：U\_PC。下面代码为示例。

pc U\_PC(…) ; // 实例化 PC(程序计数器)

c) 指令存储器必须被实例化命名：U\_IM。

d) 数据存储器必须被实例化命名：U\_DM。

e) 寄存器文件必须被实例化命名：U\_RF。

7. datapath 中的每个 module 都由一个独立的 VerilogHDL 文件组成。

a) 所有 mux（包括不同位数、不同端口数等）都建模在一个 mux.v 中。

可以有多个 module。

9. PC 复位后初值为 0x0000\_3000，目的是与 MARS 的 Memory Configuration

相配合。

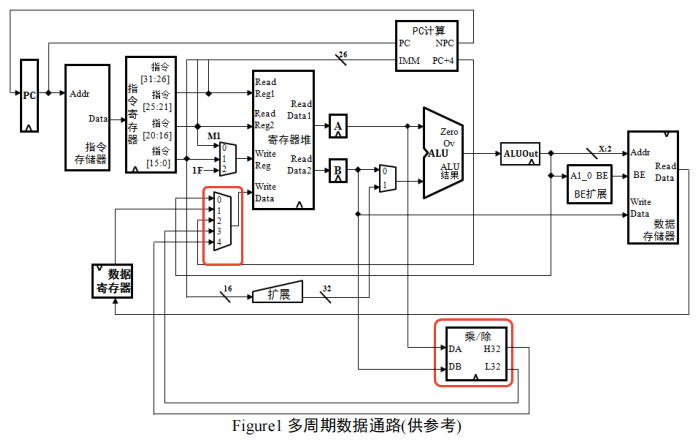
三、具体设计及实现

1. 总体结构

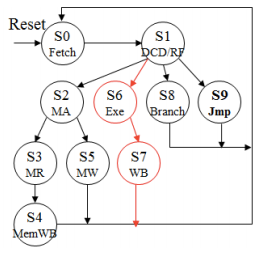
总体结构代码如下图所示，各wire变量与其同名的模块接口参数连接。

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。

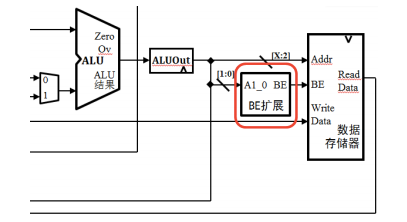
总体实现框架类似下图（但不完全相同）：



关于多周期：采用下图所示的状态机



同时，采用下图所示的扩展组件，用于满足各指令不同的扩展需求。



1. `include "testbench.v"
2. `include "alu.v"
3. `include "be.v"
4. `include "controller.v"
5. `include "dm.v"
6. `include "dmext.v"
7. `include "ext.v"
8. `include "gpr.v"
9. `include "im.v"
10. `include "muldiv.v"
11. `include "npc.v"
12. `include "pc.v"
13. `include "sgate.v"
14. `include "SGateMUX.v"
15. `include "ALUMUX.v"
16. `include "GPRWriteRegMUX.v"
17. `include "GPRWriteDataMUX.v"
19. module mips(clk, reset);
21. input clk, reset;
23. wire ALUzero, isj, isjr, RegWrite, DMwe, MUX3choose, SLTop, muldivWE;
24. wire [1:0] EXTop, MUX1choose, MUX4choose, SGateOP;
25. wire [2:0] MUX2choose, BranchOP, DMop, muldivOP;
26. wire [3:0] BEout, ALUop;
27. wire [4:0] GPRWriteReg, SGate\_s;
28. wire [31:0] pc\_in, pc\_out, NPCout, instr, HI, LO;
29. wire [31:0] GPRWriteData, ReadData1, ReadData2, EXTout, ALUin, IR\_wire;
30. wire [31:0] ALUout\_wire, DMout\_wire, DMEXTout, SLTin, SLTout, SGateout;
31. reg [31:0] IR, A, B, ALUout, DMout;
33. wire PC\_in, IR\_in, A\_in, B\_in, ALUout\_in, DMout\_in;
35. controller controller\_i(clk, reset, instr, RegWrite, DMwe, ALUop, EXTop, MUX1choose, MUX2choose, MUX3choose, MUX4choose,
36. isj, isjr, BranchOP, SLTop, PC\_in, IR\_in, A\_in, B\_in, ALUout\_in, DMout\_in, DMop, SGateOP,
37. muldivOP, muldivWE);
38. //输入：时钟，重置，指令；输出：全部信号
39. pc\_module U\_PC(pc\_in, clk, PC\_in, reset, pc\_out);
40. //输入：新PC，时钟，写信号，重置；输出：新PC
41. im\_4k U\_IM(pc\_out[11:2], IR\_wire);
42. //输入：地址；输出：指令
43. npc\_module U\_NPC(pc\_out, instr[25:0], A, isj, isjr, BranchOP, ALUzero, SGateout[31], pc\_in, NPCout);
44. //输入：各种可能影响PC值的信号；输出：NPC，PC+4
45. assign instr = IR;
47. GPRWriteRegMUX MUX1(instr[20:16], instr[15:11], 5'h1F, MUX1choose, GPRWriteReg);
48. //输入：指令中可能指代写寄存器编号的位置；输出：写寄存器编号
49. GPRWriteDataMUX MUX2(ALUout, DMout, NPCout, SLTout, HI, LO, MUX2choose, GPRWriteData);
50. //输入：可能写入寄存器的所有结果；输出：写入寄存器的数据
52. gpr\_32 U\_RF(GPRWriteData, RegWrite, clk, reset, instr[25:21], instr[20:16], GPRWriteReg, ReadData1, ReadData2);
53. //输入：需写入的数据，写信号，时钟，重置，两个读寄存器编号，一个写寄存器编号；输出：读出的两个数据
55. ext\_module U\_EXT(instr[15:0], EXTop, EXTout);
56. //输入：指令，扩展方式；输出：扩展后数值
58. ALUMUX MUX3(B, EXTout, MUX3choose, ALUin);
59. //输入：B和EXTout；输出：送入ALU的值
61. alu U\_ALU(A, ALUin, ALUop, ALUout\_wire);
62. //输入：A, B, op；输出：运算结果
63. SGateMUX MUX4(instr[10:6], A[4:0], 5'h00, MUX4choose, SGate\_s);
64. //输入：可能成为位移位数的几个值；输出：位移位数
65. shift\_gate U\_SGate(ALUout\_wire, SGateOP, SGate\_s, SGateout);
66. //输入：ALU运算结果，位移方式，位移位数；输出：位移结果
68. assign ALUzero = (SGateout == 32'h00000000);
69. muldiv\_module U\_MULDIV(A, B, muldivOP, clk, muldivWE, HI, LO);
70. //输入：A, B, op, clk, 写使能；输出：HI, LO
72. dm\_4k U\_DM(ALUout[12:2], BEout, B, DMwe, clk, DMout\_wire);
73. //输入：地址，be，输入数据，写使能，时钟；输出：DM输出
74. be\_module U\_BE(ALUout[1:0], DMop[1:0], BEout);
75. //输入：ALUout, DMop，输出：be
76. DMext U\_DMext(DMout\_wire, ALUout[1:0], DMop, DMEXTout);
77. //输入：DM输出，ALU输出，DMop；输出：扩展后结果
79. always @ (posedge clk)
80. begin
81. **if** (reset)
82. begin
83. IR <= 32'h00000000;
84. A <= 32'h00000000;
85. B <= 32'h00000000;
86. ALUout <= 32'h00000000;
87. DMout <= 32'h00000000;
88. end
89. **else**
90. begin
91. **if** (IR\_in) IR <= IR\_wire;
92. **if** (A\_in) A <= ReadData1;
93. **if** (B\_in) B <= ReadData2;
94. **if** (ALUout\_in) ALUout <= SGateout;
95. **if** (DMout\_in) DMout <= DMEXTout;
96. end
97. end
99. endmodule
100. 控制器

输入：时钟，重置，指令

输出：全部信号

1. module controller(clk, reset, instr, RegWrite, DMwe, ALUop, EXTop, MUX1choose, MUX2choose, MUX3choose, MUX4choose,
2. isj, isjr, BranchOP, SLTop, PC\_in, IR\_in, A\_in, B\_in, ALUout\_in, DMout\_in, DMop, SGateOP,
3. muldivOP, muldivWE);
5. input clk, reset;
6. input [31:0] instr;
7. output RegWrite, DMwe;
8. output [3:0] ALUop;
9. output [1:0] EXTop, MUX1choose;
10. output [2:0] MUX2choose;
11. output MUX3choose;
12. output [1:0] MUX4choose;
13. output isj, isjr;
14. output [2:0] BranchOP;
15. output SLTop, PC\_in, IR\_in, A\_in, B\_in, ALUout\_in, DMout\_in;
16. output [2:0] DMop;
17. output [1:0] SGateOP;
18. output [2:0] muldivOP;
19. output muldivWE;
21. wire [5:0] op, opp;
23. wire is\_add, is\_addu, is\_sub, is\_subu, is\_and, is\_or, is\_xor, is\_nor;
24. wire is\_addi, is\_addiu, is\_andi, is\_ori, is\_xori;
25. wire is\_lw, is\_sw, is\_lb, is\_lbu, is\_sb, is\_lh, is\_lhu, is\_sh, is\_lui;
26. wire is\_slt, is\_sltu, is\_slti, is\_sltiu;
27. wire is\_beq, is\_bne, is\_bgez, is\_bgtz, is\_blez, is\_bltz;
28. wire is\_j, is\_jal, is\_jalr, is\_jr;
29. wire is\_mult, is\_multu, is\_div, is\_divu, is\_mfhi, is\_mflo, is\_mthi, is\_mtlo;
30. wire RegWrite, DMwe, MUX3choose, isj, isjr, SLTop, muldivWE;
31. wire PC\_in, IR\_in, A\_in, B\_in, ALUout\_in, DMout\_in;
32. wire [1:0] EXTop, MUX1choose, MUX4choose, SGateOP;
33. wire [2:0] MUX2choose, BranchOP, DMop, muldivOP;
34. wire [3:0] ALUop;
36. parameter Fetch = 4'h0, RF = 4'h1, MA = 4'h2, MR = 4'h3;
37. parameter MemWB = 4'h4, MW = 4'h5, Exe = 4'h6, WB = 4'h7;
38. parameter Branch = 4'h8, Jmp = 4'h9;
40. reg [3:0] nowState;
42. initial
43. begin
44. nowState <= Fetch;
45. end
47. always @ (posedge clk)
48. begin
49. **if** (reset) nowState <= Fetch;
50. **else**
51. begin
52. **case**(nowState)
53. Fetch:
54. begin
55. nowState <= RF;
56. end
57. RF:
58. begin
59. **if** (is\_lw || is\_sw || is\_lb || is\_lbu || is\_sb || is\_lh || is\_lhu || is\_sh) nowState <= MA;
60. **else** **if** (is\_beq || is\_bne || is\_bgez || is\_bgtz || is\_blez || is\_bltz) nowState <= Branch;
61. **else** **if** (is\_j || is\_jal || is\_jalr || is\_jr) nowState <= Jmp;
62. **else** nowState <= Exe;
63. end
64. MA:
65. begin
66. **if** (is\_lw || is\_lb || is\_lbu || is\_lh || is\_lhu) nowState <= MR;
67. **else** nowState <= MW; //sw
68. end
69. MR: nowState <= MemWB;
70. MemWB: nowState <= Fetch;
71. MW: nowState <= Fetch;
72. Exe: nowState <= WB;
73. WB: nowState <= Fetch;
74. Branch: nowState <= Fetch;
75. Jmp: nowState <= Fetch;
76. endcase
77. end
78. end
80. assign op = instr[31:26];
81. assign opp = instr[5:0];
83. assign is\_add = ( (op == 6'b000000) && (opp == 6'b100000) );
84. assign is\_addu = ( (op == 6'b000000) && (opp == 6'b100001) );
85. assign is\_sub = ( (op == 6'b000000) && (opp == 6'b100010) );
86. assign is\_subu = ( (op == 6'b000000) && (opp == 6'b100011) );
87. assign is\_and = ( (op == 6'b000000) && (opp == 6'b100100) );
88. assign is\_or = ( (op == 6'b000000) && (opp == 6'b100101) );
89. assign is\_xor = ( (op == 6'b000000) && (opp == 6'b100110) );
90. assign is\_nor = ( (op == 6'b000000) && (opp == 6'b100111) );
92. assign is\_addi = (op == 6'b001000);
93. assign is\_addiu = (op == 6'b001001);
94. assign is\_andi = (op == 6'b001100);
95. assign is\_ori = (op == 6'b001101);
96. assign is\_xori = (op == 6'b001110);
98. assign is\_lui = (op == 6'b001111);
100. assign is\_slt = ( (op == 6'b000000) && (opp == 6'b101010) );
101. assign is\_sltu = ( (op == 6'b000000) && (opp == 6'b101011) );
102. assign is\_slti = (op == 6'b001010);
103. assign is\_sltiu = (op == 6'b001011);
105. assign is\_j = (op == 6'b000010);
106. assign is\_jal = (op == 6'b000011);
107. assign is\_jalr = ( (op == 6'b000000) && (opp == 6'b001001) );
108. assign is\_jr = ( (op == 6'b000000) && (opp == 6'b001000) );
110. assign is\_lw = (op == 6'b100011);
111. assign is\_sw = (op == 6'b101011);
112. assign is\_lb = (op == 6'b100000);
113. assign is\_lbu = (op == 6'b100100);
114. assign is\_lh = (op == 6'b100001);
115. assign is\_lhu = (op == 6'b100101);
116. assign is\_sb = (op == 6'b101000);
117. assign is\_sh = (op == 6'b101001);
119. assign is\_sll = ( (op == 6'b000000) && (opp == 6'b000000) );
120. assign is\_srl = ( (op == 6'b000000) && (opp == 6'b000010) );
121. assign is\_sra = ( (op == 6'b000000) && (opp == 6'b000011) );
122. assign is\_sllv = ( (op == 6'b000000) && (opp == 6'b000100) );
123. assign is\_srlv = ( (op == 6'b000000) && (opp == 6'b000110) );
124. assign is\_srav = ( (op == 6'b000000) && (opp == 6'b000111) );
126. assign is\_beq = (op == 6'b000100);
127. assign is\_bne = (op == 6'b000101);
128. assign is\_bgez = ( (op == 6'b000001) && (instr[20:16] == 5'b00001) );
129. assign is\_bgtz = (op == 6'b000111);
130. assign is\_blez = (op == 6'b000110);
131. assign is\_bltz = ( (op == 6'b000001) && (instr[20:16] == 5'b00000) );
133. assign is\_mult = ( (op == 6'b000000) && (opp == 6'b011000) );
134. assign is\_multu = ( (op == 6'b000000) && (opp == 6'b011001) );
135. assign is\_div = ( (op == 6'b000000) && (opp == 6'b011010) );
136. assign is\_divu = ( (op == 6'b000000) && (opp == 6'b011011) );
137. assign is\_mfhi = ( (op == 6'b000000) && (opp == 6'b010000) );
138. assign is\_mflo = ( (op == 6'b000000) && (opp == 6'b010010) );
139. assign is\_mthi = ( (op == 6'b000000) && (opp == 6'b010001) );
140. assign is\_mtlo = ( (op == 6'b000000) && (opp == 6'b010011) );
142. assign RegWrite =   (nowState == WB && (!is\_mult && !is\_multu && !is\_div && !is\_divu && !is\_mthi && !is\_mtlo)) ||
143. (nowState == MemWB) || (nowState == Jmp && (is\_jal || is\_jalr));
144. assign DMwe = (nowState == MW);
146. assign ALUop =  (is\_or || is\_ori) ? 4'b0001 : //or
147. (is\_sub || is\_subu || is\_beq || is\_bne) ? 4'b0010 : //sub
148. (is\_add || is\_addu || is\_addi || is\_addiu || is\_lw || is\_sw || is\_lb || is\_lbu || is\_sb || is\_lh || is\_lhu || is\_sh) ? 4'b0011 : //add
149. (is\_and || is\_andi) ? 4'b0100 : //and
150. (is\_xor || is\_xori) ? 4'b0101 : //xor
151. (is\_nor) ? 4'b0110 : //nor
152. (is\_slt || is\_slti) ? 4'b0111 : //signed compare
153. (is\_sltu || is\_sltiu) ? 4'b1000 : //unsigned compare
154. (is\_bgez || is\_bgtz || is\_blez || is\_bltz) ? 4'b1001 : //output A
155. 4'b0000; //output B
157. assign EXTop =  (is\_addi || is\_addiu || is\_lw || is\_sw || is\_slti || is\_sltiu || is\_lb || is\_lbu || is\_sb || is\_lh || is\_lhu || is\_sh) ? 2'b00 : //signed
158. (is\_andi || is\_ori || is\_xori) ? 2'b01 : //unsigned
159. 2'b10; //high(lui)
161. assign MUX1choose = (is\_addi || is\_addiu || is\_ori || is\_lw || is\_lb || is\_lh || is\_lbu || is\_lhu || is\_lui || is\_slti || is\_sltiu
162. || is\_andi || is\_xori) ? 2'b00 : //rt
163. (is\_add || is\_addu || is\_sub || is\_subu || is\_and || is\_or || is\_xor || is\_nor || is\_slt || is\_sltu || is\_jalr
164. || is\_sll || is\_srl || is\_sra || is\_sllv || is\_srlv || is\_srav || is\_mfhi || is\_mflo) ? 2'b01 : //rd
165. (is\_jal) ? 2'b10 : //5'b11111
166. 2'b11;
168. assign MUX2choose = (is\_add || is\_addu || is\_sub || is\_subu || is\_ori || is\_addi || is\_addiu || is\_lui
169. || is\_and || is\_or || is\_xor || is\_nor || is\_andi || is\_xori
170. || is\_slt || is\_sltu || is\_slti || is\_sltiu
171. || is\_sll || is\_srl || is\_sra || is\_sllv || is\_srlv || is\_srav) ? 3'b000 : //ALUout
172. (is\_lw || is\_lb || is\_lh || is\_lbu || is\_lhu) ? 3'b001 : //DMout
173. (is\_jal || is\_jalr) ? 3'b010 : //PC+4
174. (is\_mfhi) ? 3'b100 : //HI
175. (is\_mflo) ? 3'b101 : //LO
176. 3'b011; //SLTout
178. assign MUX3choose = (is\_andi || is\_ori || is\_xori || is\_lui || is\_addi || is\_addiu || is\_lw || is\_sw || is\_slti || is\_sltiu
179. || is\_lb || is\_lbu || is\_sb || is\_lh || is\_lhu || is\_sh); //imm
181. assign MUX4choose = (is\_sll || is\_srl || is\_sra) ? 2'b00 : //imm
182. (is\_sllv || is\_srlv || is\_srav) ? 2'b01 : //GPR[rs]
183. 2'b10; //5'b00000
185. assign isj = (is\_j || is\_jal);
186. assign isjr = (is\_jr || is\_jalr);
187. assign BranchOP =   (is\_beq || is\_bne || is\_bgtz || is\_blez) ? op[2:0] :
188. (is\_bgez || is\_bltz) ? {2'b01, instr[16]} :
189. 3'b000;
191. assign SLTop = (is\_sltu || is\_sltiu); // is unsigned?
193. assign PC\_in = (nowState == MemWB) || (nowState == MW) || (nowState == WB) || (nowState == Branch) || (nowState == Jmp);
194. assign IR\_in = (nowState == Fetch);
195. assign A\_in = (nowState == RF);
196. assign B\_in = (nowState == RF);
197. assign ALUout\_in = (nowState == MA) || (nowState == Exe);
198. assign DMout\_in = (nowState == MR);
200. assign DMop[2] = (is\_lbu || is\_lhu); //u?
201. assign DMop[1:0] = (is\_lb || is\_sb || is\_lbu) ? 2'b00 :
202. (is\_lh || is\_sh || is\_lhu) ? 2'b01 :
203. 2'b11; //b,h,w
205. assign SGateOP = opp[1:0];
207. assign muldivOP = ((op == 6'b000000) && (opp[5:3] == 3'b011)) ? opp[2:0] : //mult, div
208. (is\_mthi) ? 3'b100 : //mthi
209. (is\_mtlo) ? 3'b101 : //mtlo
210. 3'b000;
212. assign muldivWE = (nowState == Exe && (is\_mult || is\_multu || is\_div || is\_divu || is\_mthi || is\_mtlo));
214. endmodule
215. 算数逻辑单元ALU

输入：A, B, op

输出：运算结果

1. module alu(A,B,F,C);
2. input [31:0] A, B;
3. input [3:0] F;
4. output [31:0] C;
6. wire [31:0] C;
7. parameter op\_or = 4'b0001, op\_sub = 4'b0010, op\_add = 4'b0011, op\_and = 4'b0100;
8. parameter op\_xor = 4'b0101, op\_nor = 4'b0110, op\_scomp = 4'b0111, op\_ucomp = 4'b1000;
9. parameter op\_A = 4'b1001;
11. assign  C = (F == op\_or) ? (A|B) :
12. (F == op\_sub) ? (A-B) :
13. (F == op\_add) ? (A+B) :
14. (F == op\_and) ? (A&B) :
15. (F == op\_xor) ? (A^B) :
16. (F == op\_nor) ? (~(A|B)) :
17. (F == op\_scomp) ? {31'b0, ($**signed**(A)<$**signed**(B))} :
18. (F == op\_ucomp) ? {31'b0, ($unsigned(A)<$unsigned(B))} :
19. (F == op\_A) ? A : B;
21. endmodule
22. be字节使能计算模块

输入：ALUout, DMop

输出：be

1. module be\_module(ALUout, DMop, BEout);
2. input [1:0] ALUout, DMop;
3. output [3:0] BEout;
5. wire [3:0] BEout;
7. assign BEout[3] = (DMop == 2'b11) || (DMop == 2'b01 && ALUout[1] == 1'b1) || (DMop == 2'b00 && ALUout == 2'b11);
8. assign BEout[2] = (DMop == 2'b11) || (DMop == 2'b01 && ALUout[1] == 1'b1) || (DMop == 2'b00 && ALUout == 2'b10);
9. assign BEout[1] = (DMop == 2'b11) || (DMop == 2'b01 && ALUout[1] == 1'b0) || (DMop == 2'b00 && ALUout == 2'b01);
10. assign BEout[0] = (DMop == 2'b11) || (DMop == 2'b01 && ALUout[1] == 1'b0) || (DMop == 2'b00 && ALUout == 2'b00);
12. endmodule
13. DM数据存储器

输入：地址，be，输入数据，写使能，时钟

输出：DM输出

1. module dm\_4k(addr, be, din, we, clk, dout);
2. input [12:2] addr;
3. input [3:0] be;
4. input [31:0] din;
5. input we, clk;
6. output [31:0] dout;
8. wire [31:0] dout;
9. reg [31:0] dm[2047:0];
11. assign dout = dm[addr];
13. integer i;
15. initial
16. begin
17. **for**(i=0; i<2048; i=i+1)
18. dm[i]=32'h00000000;
19. end
21. always @(posedge clk)
22. **if** (we)
23. begin
24. **case**(be)
25. 4'b1111: dm[addr] <= din;
26. 4'b0011: dm[addr][15:0] <= din[15:0];
27. 4'b1100: dm[addr][31:16] <= din[15:0];
28. 4'b0001: dm[addr][7:0] <= din[7:0];
29. 4'b0010: dm[addr][15:8] <= din[7:0];
30. 4'b0100: dm[addr][23:16] <= din[7:0];
31. 4'b1000: dm[addr][31:24] <= din[7:0];
32. endcase
33. end
35. integer handle;
37. initial
38. begin
39. #9980000
40. handle = $fopen("dm.txt","w");
41. **for**(i=0; i<32; i=i+1)
42. $fwrite(handle, "dm[%d]=%h\n", i, dm[i]);
43. $fclose(handle);
44. end
46. endmodule
47. 数据存储器扩展单元DMEXT

输入：DM输出，ALU输出，DMop

输出：扩展后结果

1. module DMext(memword, ALUout, DMop, dataout);
3. input [31:0] memword;
4. input [1:0] ALUout;
5. input [2:0] DMop;
6. output [31:0] dataout;
8. reg [31:0] dataout;
10. always
11. begin
12. #5
13. **case**(DMop[1:0])
14. 2'b00: begin
15. **case**(ALUout)
16. 2'b00: dataout[7:0] <= memword[7:0];
17. 2'b01: dataout[7:0] <= memword[15:8];
18. 2'b10: dataout[7:0] <= memword[23:16];
19. 2'b11: dataout[7:0] <= memword[31:24];
20. endcase
21. **if** (DMop[2]) dataout[31:8] <= 24'b0;
22. **else** dataout[31:8] <= {24{dataout[7]}};
23. end
24. 2'b01: begin
25. **case**(ALUout[1])
26. 1'b0: dataout[15:0] <= memword[15:0];
27. 1'b1: dataout[15:0] <= memword[31:16];
28. endcase
29. **if** (DMop[2]) dataout[31:16] <= 16'b0;
30. **else** dataout[31:16] <= {16{dataout[15]}};
31. end
32. 2'b11: dataout <= memword;
33. endcase
34. end
36. endmodule
37. 指令数扩展单元EXT

输入：指令，扩展方式

输出：扩展后数值

1. module ext\_module(A,EXTop,B);
2. input [15:0] A;
3. input [1:0] EXTop;
4. output [31:0] B;
5. wire [31:0] B;
7. parameter op\_signed = 2'b00;
8. parameter op\_unsigned = 2'b01;
9. parameter op\_high = 2'b10;
11. assign  B = (EXTop==op\_signed) ? {{16{A[15]}},A} :
12. (EXTop==op\_unsigned) ? {16'h0000,A} :
13. (EXTop==op\_high) ? {A,16'h0000} :
14. 32'bz;
16. endmodule
17. 寄存器堆GPR

输入：需写入的数据，写信号，时钟，重置，两个读寄存器编号，一个写寄存器编号

输出：读出的两个数据

1. module gpr\_32(Wd, RegWrite, Clk, Reset, A1, A2, A3, Rd1, Rd2);
3. input [31:0] Wd;
4. input RegWrite, Clk, Reset;
5. input [4:0] A1, A2, A3;
6. output [31:0] Rd1, Rd2;
8. wire [31:0] Rd1, Rd2;
9. reg [31:0] gpr[31:0];
11. assign Rd1 = gpr[A1];
12. assign Rd2 = gpr[A2];
14. integer i;
16. always @(posedge Clk)
17. begin
18. **if** (Reset) begin
19. **for**(i=0; i<32; i=i+1)
20. gpr[i] <= 32'h00000000;
21. end
22. **else** **if** (RegWrite) gpr[A3] <= Wd;
23. end
25. integer handle;

28. initial
29. begin
30. #9990000
31. handle = $fopen("gpr.txt","w");
32. **for**(i=0; i<32; i=i+1)
33. $fwrite(handle, "gpr[%d]=%h\n", i, gpr[i]);
34. $fclose(handle);
35. end
36. //for output
38. endmodule
39. 指令存储器IM

输入：地址

输出：指令

1. module im\_4k(addr,dout);
2. input [11:2] addr;
3. output [31:0] dout;
5. reg [31:0] im[1023:0];
6. wire [31:0] dout;
8. assign dout = im[addr[11:2]];
10. initial begin
11. $readmemh("code.txt", im, 0, 1023);
12. end
14. endmodule
15. 乘除运算单元MULDIV

输入：A, B, op, clk, 写使能

输出：HI, LO

1. module muldiv\_module(DA, DB, op, clk, we, HI, LO);
3. input [31:0] DA, DB;
4. input [2:0] op;
5. input clk, we;
6. output [31:0] HI, LO;
8. reg [31:0] HI, LO;
10. always @ (posedge clk)
11. begin
12. **if** (we)
13. **case**(op)
14. 3'b000: {HI, LO} <= $**signed**(DA) \* $**signed**(DB); //mult
15. 3'b001: {HI, LO} <= $unsigned(DA) \* $unsigned(DB); //multu
16. 3'b010:
17. begin
18. LO <= $**signed**(DA) / $**signed**(DB);
19. HI <= $**signed**(DA) % $**signed**(DB);
20. end //div
21. 3'b011:
22. begin
23. LO <= $unsigned(DA) / $unsigned(DB);
24. HI <= $unsigned(DA) % $unsigned(DB);
25. end //divu
26. 3'b100: HI <= DA; //write\_HI
27. 3'b101: LO <= DA; //write\_LO
28. endcase
29. end
31. endmodule
32. 位移部件多路选择器SGateMUX

输入：可能成为位移位数的几个值

输出：位移位数

1. module SGateMUX(in0, in1, in2, choose, dout);
2. input [4:0] in0, in1, in2;
3. input [1:0] choose;
4. output [4:0] dout;
6. wire [4:0] dout;
7. assign dout =   (choose==2'b00) ? in0 :
8. (choose==2'b01) ? in1 :
9. (choose==2'b10) ? in2 :
10. 5'b00000;
11. endmodule
12. ALU输入多路选择器ALUMUX

输入：B和EXTout

输出：送入ALU的值

1. module ALUMUX(in0, in1, choose, dout);
2. input [31:0] in0, in1;
3. input choose;
4. output [31:0] dout;
6. wire [31:0] dout;
7. assign dout = (choose) ? in1 : in0;
8. endmodule
9. 写寄存器编号多路选择器GPRWriteRegMUX

输入：指令中可能指代写寄存器编号的位置

输出：写寄存器编号

1. module GPRWriteRegMUX(in0, in1, in2, choose, dout);
2. input [4:0] in0, in1, in2;
3. input [1:0] choose;
4. output [4:0] dout;
6. wire [4:0] dout;
7. assign dout =   (choose==2'b00) ? in0 :
8. (choose==2'b01) ? in1 :
9. (choose==2'b10) ? in2 :
10. 5'b00000;
11. endmodule
12. 写寄存器数据多路选择器GPRWriteDataMUX

输入：可能写入寄存器的所有结果

输出：写入寄存器的数据

1. module GPRWriteDataMUX(in0, in1, in2, in3, in4, in5, choose, dout);
2. input [31:0] in0, in1, in2, in3, in4, in5;
3. input [2:0] choose;
4. output [31:0] dout;
6. wire [31:0] dout;
7. assign dout =   (choose==3'b000) ? in0 :
8. (choose==3'b001) ? in1 :
9. (choose==3'b010) ? in2 :
10. (choose==3'b011) ? in3 :
11. (choose==3'b100) ? in4 :
12. in5;
13. endmodule
14. NPC运算单元

输入：各种可能影响PC值的信号

输出：NPC，PC+4

1. module npc\_module(pcin, imm, fromrs, isj, isjr, BranchOP, alu\_zero, alu\_sign, npc, pcout);
2. input [31:0] pcin;
3. input [25:0] imm;
4. input [31:0] fromrs;
5. input isj, isjr;
6. input [2:0] BranchOP;
7. input alu\_zero, alu\_sign;
8. output [31:0] npc;
9. output [31:0] pcout;
10. wire [31:0] npc, pcout;
12. wire [31:0] EXT\_out;
13. ext\_module ext\_(imm[15:0], 2'b00, EXT\_out);
15. assign npc =    (isjr) ? fromrs :
16. (isj) ? {pcin[31:28],imm,2'b00} :
17. (BranchOP == 3'b100 && alu\_zero) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //beq
18. (BranchOP == 3'b101 && !alu\_zero) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //bne
19. (BranchOP == 3'b011 && !alu\_sign) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //bgez
20. (BranchOP == 3'b111 && !alu\_sign && !alu\_zero) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //bgtz
21. (BranchOP == 3'b110 && (alu\_sign || alu\_zero)) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //blez
22. (BranchOP == 3'b010 && alu\_sign) ? (pcin + 32'h00000004 + {EXT\_out,2'b00}) : //bltz
23. (pcin + 32'h00000004);
24. assign pcout = pcin + 32'h00000004;
26. endmodule
27. PC模块

输入：新PC，时钟，写信号，重置

输出：新PC

1. module pc\_module(din, clock, pc\_in, reset, dout);
2. input [31:0] din;
3. input clock, pc\_in, reset;
4. output [31:0] dout;
5. wire [31:0] dout;
6. reg [31:0] pc;
8. assign dout = pc;
10. always @(posedge clock)
11. **if** (reset) pc <= 32'h00003000;
12. **else** **if** (pc\_in) pc <= din;
14. endmodule
15. 位移模块SGate

输入：ALU运算结果，位移方式，位移位数

输出：位移结果

1. module shift\_gate(ALUout, SGateOP, s, dataout);
3. input [31:0] ALUout;
4. input [1:0] SGateOP;
5. input [4:0] s;
6. output [31:0] dataout;
8. reg [31:0] dataout;
10. always
11. begin
12. #5
13. **case**(SGateOP)
14. 2'b00: dataout <= (ALUout << s);
15. 2'b01: dataout <= ALUout;
16. 2'b10: dataout <= (ALUout >> s);
17. 2'b11: dataout <= ($**signed**(ALUout) >>> s);
18. endcase
19. end
21. endmodule
22. 程序测试

测试数据由MARS从汇编指令转换得到：

3c01ffff

3421fff5

00014025

34090064

01090018

00005010

00005812

01090019

00006010

00006812

0128001a

00007010

00007812

0109001b

00008010

00008812

01000011

01200013

00009010

00009812

共20条指令，程序正常运行。运行完毕时的GPR寄存器堆内数值：

gpr[0]=00000000

gpr[1]=fffffff5

gpr[2]=00000000

gpr[3]=00000000

gpr[4]=00000000

gpr[5]=00000000

gpr[6]=00000000

gpr[7]=00000000

gpr[8]=fffffff5

gpr[9]=00000064

gpr[10]=ffffffff

gpr[11]=fffffbb4

gpr[12]=00000063

gpr[13]=fffffbb4

gpr[14]=00000001

gpr[15]=fffffff7

gpr[16]=00000055

gpr[17]=028f5c28

gpr[18]=fffffff5

gpr[19]=00000064

gpr[20]=00000000

gpr[21]=00000000

gpr[22]=00000000

gpr[23]=00000000

gpr[24]=00000000

gpr[25]=00000000

gpr[26]=00000000

gpr[27]=00000000

gpr[28]=00000000

gpr[29]=00000000

gpr[30]=00000000

gpr[31]=00000000

运行结束生成的VCD文件如图：

