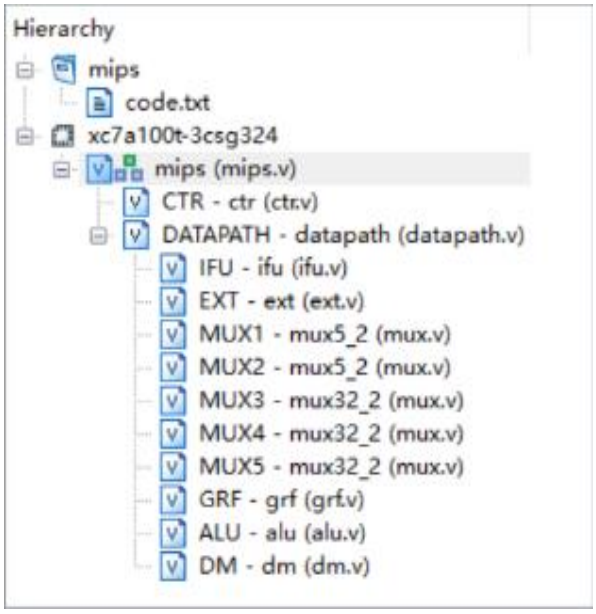
计算机组成原理实验报告

一、 CPU 设计文档

（一） 总体设计



图表 1 模块设计

```
21  module mips(
22      input clk,
23      input reset
24      );
25     wire ctr_jr, ctr_jal, ctr_nPCsel, ctr_MemWrite, ctr_RegWrite, ctr_MemtoReg, ctr_ALUSrc, ctr_RegDst;
26     wire [1:0] ctr_EXTop;
27     wire [2:0] ctr_ALUctr;
28     wire [31:0] ctr_instruction;
29
30     ctr CTR (
31        .jr(ctr_jr),
32        .jal(ctr_jal),
33        .nPCsel(ctr_nPCsel),
34        .MemWrite(ctr_MemWrite),
35        .RegWrite(ctr_RegWrite),
36        .MemtoReg(ctr_MemtoReg),
37        .ALUSrc(ctr_ALUSrc),
38        .RegDst(ctr_RegDst),
39        .EXTop(ctr_EXTop),
40        .ALUctr(ctr_ALUctr),
41
42        .instruction(ctr_instruction)
43        );
44
45     datapath DATAPATH (
46        .clk(clk),
47        .reset(reset),
48        .datapath_jr(ctr_jr),
49        .datapath_jal(ctr_jal),
50        .datapath_nPCsel(ctr_nPCsel),
51        .datapath_MemWrite(ctr_MemWrite),
52        .datapath_RegWrite(ctr_RegWrite),
53        .datapath_MemtoReg(ctr_MemtoReg),
54        .datapath_ALUSrc(ctr_ALUSrc),
```

图表 2 总体设计

```
55          .datapath_RegDst(ctr_RegDst),
56          .datapath_EXTop(ctr_EXTop),
57          .datapath_ALUctr(ctr_ALUctr),
58
59          .datapath_instruction(ctr_instruction)
60          );
61
62  endmodule
63
```

图表 3 总体设计

（二） 数据通路设计

1. datapath（数据通路）

```
21  module datapath(
22     input clk,
23     input reset,
24     input datapath_jr,
25     input datapath_jal,
26     input datapath_nPCsel,
27     input datapath_MemWrite,
28     input datapath_RegWrite,
29     input datapath_MemtoReg,
30     input datapath_ALUSrc,
31     input datapath_RegDst,
32     input [1:0] datapath_EXTop,
33     input [2:0] datapath_ALUctr,
34     output [31:0] datapath_instruction
35      );
36     wire alu_Zero;
37     wire [4:0] writereg_temp, writereg, rt, rd;
38     wire [31:0] alu_Result, dm_dout, ext_immed32, grf_readdata1, grf_readdata2, mux32_2_C, ifu_PC, ifu_PC4, data, data_temp, ALU
39
40     ifu IFU (
41        .instruction(datapath_instruction),
42        .PC(ifu_PC),
43        .PC4(ifu_PC4),
44
45        .clk(clk),
46        .reset(reset),
47        .immed32(ext_immed32),
48        .immed26(datapath_instruction[25:0]),
49        .reg1data(grf_readdata1),
50        .jal(datapath_jal),
51        .jr(datapath_jr),
52        .nPCsel(datapath_nPCsel),
53        .Zero(alu_Zero)
54        );
55
```

图表 4datapath

```
56     ext EXT (
57        .immed32(ext_immed32),
58
59        .immed16(datapath_instruction[15:0]),
60        .EXTop(datapath_EXTop)
61        );
62
63     mux5_2 MUX1 (
64        .A5(datapath_instruction[20:16]),
65        .B5(datapath_instruction[15:11]),
66        .option(datapath_RegDst),
67        .C5(writereg_temp)
68        );
69
70     mux5_2 MUX2 (
71        .A5(writereg_temp),
72        .B5(5'h1f),
73        .option(datapath_jal),
74        .C5(writereg)
75        );
76
77     mux32_2 MUX3 (
78        .A32(grf_readdata2),
79        .B32(ext_immed32),
80        .option(datapath_ALUSrc),
81        .C32(ALU_B)
82        );
83
84     mux32_2 MUX4 (
85        .A32(alu_Result),
86        .B32(dm_dout),
87        .option(datapath_MemtoReg),
88        .C32(data_temp)
89        );
```

图表 5datapath

```
90    mux32_2 MUX5 (
91       .A32(data_temp),
92       .B32(ifu_PC4),
93       .option(datapath_jal),
94       .C32(data)
95       );
96
97    grf GRF (
98       .readdata1(grf_readdata1),
99       .readdata2(grf_readdata2),
100
101      .clk(clk),
102      .reset(reset),
103      .PC(ifu_PC),
104      .writedata(data),
105      .writereg(writereg),
106      .instruction(datapath_instruction),
107      .regwrite(datapath_RegWrite)
108      );
109
110   alu ALU (
111      .Result(alu_Result),
112      .Zero(alu_Zero),
113
114      .A(grf_readdata1),
115      .B(ALU_B),
116      .ALUOp(datapath_ALUctr)
117      );
118
119   dm DM (
120      .dout(dm_dout),
121
122      .clk(clk),
123      .reset(reset),
124      .PC(ifu_PC)
```

图表 6datapath

```
124         .PC(ifu_PC),
125         .MemWrite(datapath_MemWrite),
126         .ADDR(alu_Result),
127         .din(grf_readdata2)
128         );
129
130   endmodule
131
```

图表 7datapath

1）端口说明

表格 1datapath 端口说明

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号 1：复位 0：无效
datapath_jr	I	控制器发来 jr 信号 1：jr 指令 0：无效
datapath_jal	I	控制器发来 jal 信号 1：jal 指令 0：无效
		控制器发来 nPCsel 信号

datapath_nPCsel	I	1：跳转指令 0：无效
datapath_MemWrite	I	控制器发来 MemWrite 信号 1：写内存 0：无效
datapath_RegWrite	I	控制器发来 RegWrite 信号 1：写寄存器 0：无效
datapath_MemtoReg	I	控制器发来 MemtoReg 信号 1：内存输出有效内容 0：无效
datapath_ALUSrc	I	控制器发来 ALUSrc 信号 1：32 位立即数有效 0：无效
datapath_RegDst	I	控制器发来 RegDst 信号 1：写入 rd 寄存器 0：写入 rt 寄存器
datapath_EXTop[1:0]	I	控制器发来 EXTop 信号 00：无符号扩展 01：低 16 位补 0 10：有符号扩展 11：有符号扩展后逻辑左移两位
datapath_ALUctr[2:0]	I	控制器发来 ALUctr 信号 控制信号 000：加运算 001：减运算 010：或运算 011：输出写入数据 2

		100:异或运算
datapath_instruction[31:0]	0	发给控制器的指令

2） 功能定义

表格 2datapath 功能定义

序号	功能名称	功能描述
1	连接基本模块	通过 datapath，以声明中间变量和实例化引用的方式连接各基础模块

2. ifu（取指令单元）

```
21  module ifu(
22     input clk,
23     input reset,
24     input [31:0] immed32,
25     input [25:0] immed26,
26     input [31:0] reg1data,
27     input jal,
28     input jr,
29     input nPCsel,
30     input Zero,
31     output [31:0] instruction,
32     output reg [31:0] PC,
33     output [31:0] PC4
34      );
35     reg [31:0] instructions [1023:0];
36     integer i;
37     assign PC4 = PC + 4;
38
39     initial begin
40        PC = 32'h00003000;
41        for (i = 0; i < 1024; i = i + 1)
42           instructions[i] = 0;
43        $readmemh("code.txt", instructions);
44     end
45     assign instruction = instructions[PC[11:2]];
46
47     always @(posedge clk) begin
48        if (reset) begin
49           PC <= 32'h00003000;
50        end
51        else begin
52           if (jr)
53              PC <= reg1data;
54           else begin
```

图表 8ifu

```
55              if (jal)
56                  PC <= {PC[31:28], immed26, 2'b0};
57              else begin
58                  if (nPCsel == 1 && Zero == 1)
59                      PC <= PC + 4 + immed32;
60                  else
61                      PC <= PC + 4;
62              end
63          end
64      end
65  end
66
67  endmodule
68
```

图表 9ifu

1）端口说明

表格 3ifu 端口说明

信号名	方向	描述
jr	I	控制器发来 jr 信号 1：jr 指令 0：无效
reset	I	复位信号 1：复位 0：无效
clk	I	时钟信号
jal	I	控制器发来 jal 信号 1：jal 指令 0：无效
nPCsel	I	控制器发来 nPCsel 信号 1：跳转指令 0：无效
Zero	I	alu 发来 Zero 信号 1：alu 两输入相等 0：alu 两输入不等
immed32[31:0]	I	ext 发来 32 位立即数信号
immed26[25:0]	I	26 位立即数信号

reg1data[31:0]	I	grf 发来 reg1 的值
instruction[31:0]	O	输出的指令
PC[31:0]	O	输出的当前 PC
PC4[31:0]	O	输出的当前 PC+4

2) 功能定义

表格 4 ifu 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，PC 被置为 0x00000000
2	取指令	根据 PC 从 IM 中取出指令
3	计算下一条指令地址	PC←PC+4 || PC←reg1data || PC←PC + 4 + immed32 || PC←{PC[31:28], immed26, 2'b0}

3. grf（通用寄存器组）

```
21  module grf(
22      input clk,
23      input reset,
24      input [31:0] writedata,
25      input [4:0] writereg,
26      input [31:0] instruction,
27      input regwrite,
28      input [31:0] PC,
29      output [31:0] readdata1,
30      output [31:0] readdata2
31       );
32      reg [31:0] regs [31:0];
33      integer i;
34      assign readdata1 = regs[instruction[25:21]];
35      assign readdata2 = regs[instruction[20:16]];
36      initial begin
37          for (i = 0; i < 32; i = i + 1) begin
38              regs[i] = 0;
39          end
40      end
41
42      always @(posedge clk) begin
43          if (reset) begin
44              for (i = 0; i < 32; i = i + 1) begin
45                  regs[i] <= 0;
46              end
47          end
48          else begin
49              if (regwrite) begin
50                  regs[writereg] <= writedata;
51                  $display("@%h: $%d <= %h", PC, writereg, writedata);
52              end
53              regs[0] <= 0;
54          end
55      end
```

图表 10grf

1）端口说明

表格 5grf 端口说明

信号名	方向	描述
writedata[31:0]	I	写入的 32 位数据
reset	I	复位信号 1：复位 0：无效
clk	I	时钟信号
instruction[31:0]	I	指令
writereg[4:0]	I	写寄存器编号
regwrite	I	写控制信号 1：写入

		0：无效
readdata1[31:0]	0	32 位寄存器 1 输出
readdata2[31:0]	0	32 位寄存器 2 输出

2） 功能定义

表格 6grf 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，32 个寄存器被置为 0x00000000
2	写寄存器	写寄存器控制信号有效时，把 32 位数据写入寄存器
3	读寄存器	根据输入的地址读出两个寄存器中的值

4. alu（算术逻辑单元）

```
21  module alu(
22     input [31:0] A,
23     input [31:0] B,
24     input [2:0] ALUOp,
25     output [31:0] Result,
26     output Zero
27      );
28     assign Zero = (A == B) ? 1 : 0;
29     assign Result = (ALUOp == 0) ? (A + B) : ((ALUOp == 1) ? (A - B) : ((ALUOp == 2) ? (A | B) : ((ALUOp == 3) ? B : (A ^ B))));
30
31  endmodule
32
```

图表 11ALU

1） 端口说明

表格 7alu 端口说明

信号名	方向	描述
A[31:0]	I	32 位写入数据 1
B[31:0]	I	32 位写入数据 2
ALUOp[2:0]	I	控制信号 000：加运算 001：减运算 010：或运算

		011：输出写入数据 2 100:异或运算
Zero	0	相等信号 1：相等 0：不相等
Result[31:0]	0	32 位输出数据

2）功能定义

表格 8alu 功能定义

序号	功能名称	功能描述
1	加运算	A+B
2	减运算	A-B
3	或运算	A|B
4	输出写入数据 2	B
5	异或运算	A^B
6	比较运算	A==B

5. dm（数据存储器）

```
21  module dm(
22     input clk,
23     input reset,
24     input MemWrite,
25     input [31:0] ADDR,
26     input [31:0] din,
27     input [31:0] PC,
28     output [31:0] dout
29      );
30     wire [9:0] addr;
31     assign addr = ADDR[11:2];
32     reg [31:0] memory[1023:0];
33     integer i;
34     assign dout = MemWrite ? din : memory[addr];
35
36     initial begin
37        for (i = 0; i < 1024; i = i + 1) begin
38           memory[i] = 0;
39        end
40     end
41
42     always @(posedge clk) begin
43        if (reset) begin
44           for (i = 0; i < 1024; i = i + 1) begin
45              memory[i] <= 0;
46           end
47        end
48        else begin
49           if (MemWrite) begin
50              memory[addr] <= din;
51              $display("@%h: *%h <= %h", PC, ADDR, din);
52           end
53        end
54     end
```

图表 12dm

1）端口说明

表格 9dm 端口说明

信号名	方向	描述
ADDR [31:0]	I	32 位写入内存地址
din[31:0]	I	32 位写入数据
clk	I	时钟信号
reset	I	复位信号 1：复位 0：无效
PC[31:0]	I	当前 PC
MemWrite	I	写内存控制信号 1：写入 0：无效
dout[31:0]	O	32 位输出数据

2）功能定义

表格 10dm 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，内存和读出内存的寄存器被置为 0x00000000
2	写内存	写内存控制信号有效时，根据输入的地址写入 32 位数据
3	读内存	根据输入的地址读出内存数据

6. ext（位扩展器）

```
21  module ext(
22     input [15:0] immed16,
23     input [1:0] EXTop,
24     output [31:0] immed32
25      );
26     assign immed32 = (EXTop == 0) ? {16'b0, immed16} :
27           ((EXTop == 1) ? {immed16, 16'b0} :
28           ((EXTop == 2) ? ((immed16[15]) ? {{16{1'b1}}, immed16} :
29           {{16'b0}, immed16}) : ((immed16[15]) ? {{14{1'b1}}, immed16, 2'b0} :
30           {14'b0, immed16, 2'b0})));
31
32  endmodule
33
```

图表 13ext

1）端口说明

表格 11ext 端口说明

信号名	方向	描述
immed16[15:0]	I	16 位写入立即数
EXTop[1:0]	I	扩展控制信号 00：无符号扩展 01：低 16 位补 0 10：有符号扩展 11：有符号扩展后逻辑左移两位

Immed32[31:0]	0	32 位输出立即数

2）功能定义

表格 12ext 功能定义

序号	功能名称	功能描述
1	无符号扩展	高 16 位补 0
2	低 16 位补 0	低 16 位补 0
3	有符号扩展	Immed[15]为 1 时高 16 位补 1，为 0 时高 16 位补 0
4	有符号扩展后逻辑左移两位	Immed[15]为 1 时高 16 位补 1，为 0 时高 16 位补 0，再左移两位，溢出舍去，低 2 位补 0

7. mux（多路选择器）

```
21  module mux5_2(
22      input [4:0] A5,
23      input [4:0] B5,
24      input option,
25      output [4:0] C5
26       );
27      assign C5 = option ? B5 : A5;
28
29  endmodule
30
31  module mux32_2(
32      input [31:0] A32,
33      input [31:0] B32,
34      input option,
35      output [31:0] C32
36       );
37
38      assign C32 = option ? B32 : A32;
39
40  endmodule
41
```

图表 14mux

1）端口说明

表格 13mux 端口说明

信号名	方向	描述
A5[4:0]	I	5 位输入 A5
B5[4:0]	I	5 位输入 B5
option	I	选择控制信号 1：输出 B5 0：输出 A5
C5[4:0]	O	5 位输出 C5
A32[31:0]	I	32 位输入 A32
B32[31:0]	I	32 位输入 B32
option	I	选择控制信号 1：输出 B32 0：输出 A32
C32[31:0]	O	32 位输出 C32

2）功能定义

表格 14mux 功能定义

序号	功能名称	功能描述
1	5 位输入 2 选 1	option 为 1 输出 B5，为 0 输出 A5
2	32 位输入 2 选 1	option 为 1 输出 B32，为 0 输出 A32

（三） 控制器设计

```
29  module ctr(
30     input [31:0] instruction,
31     output jr,
32     output jal,
33     output [2:0] ALUctr,
34     output [1:0] EXTop,
35     output nPCsel,
36     output MemWrite,
37     output RegWrite,
38     output MemtoReg,
39     output ALUSrc,
40     output RegDst
41      );
42     assign jr = (instruction[31:26] == 6'b0 && instruction[5:0] == 6'b001000) ? 1 : 0;
43     assign jal = (instruction[31:26] == `jal) ? 1 : 0;
44     assign ALUctr = (instruction[31:26] == `xori) ? 4 :
45           ((instruction[31:26] == `lui) ? 3 : ((instruction[31:26] == `ori) ? 2 :
46           ((instruction[31:26] == 0 && instruction[5:0] == 6'b100011) ? 1 : 0)));
47     assign EXTop = (instruction[31:26] == `beq) ? 3 :
48           ((instruction[31:26] == `lw || instruction[31:26] == `sw) ? 2 :
49           ((instruction[31:26] == `lui) ? 1 : 0));
50     assign nPCsel = (instruction[31:26] == `beq) ? 1 : 0;
51     assign MemWrite = (instruction[31:26] == `sw) ? 1 : 0;
52     assign RegWrite = (instruction[31:26] == `ori || instruction[31:26] == `lw || instruction[31:26] == `lui
53           || instruction[31:26] == `jal || instruction[31:26] == `xori
54           || (instruction[31:26] == 0 && instruction[5:0] == 6'b100001)
55           || (instruction[31:26] == 0 && instruction[5:0] == 6'b100011)) ? 1 : 0;
56     assign MemtoReg = (instruction[31:26] == `lw) ? 1 : 0;
57     assign ALUSrc = (instruction[31:26] == `ori || instruction[31:26] == `lw
58           || instruction[31:26] == `sw || instruction[31:26] == `lui) ? 1 : 0;
59     assign RegDst = ((instruction[31:26] == 0 && instruction[5:0] == 6'b100001)
60           || (instruction[31:26] == 0 && instruction[5:0] == 6'b100011)) ? 1 : 0;
61
62  endmodule
63
```

图表 15ctr

1. 端口说明

表格 15ctr 端口说明

信号名	方向	描述
instruction[31:0]	I	32 位指令
RegDst	O	grf 写寄存器决定信号 1：rd 0：rt
ALUSrc	O	alu 输入数据 B 决定信号 1：32 位立即数 0：GRF 寄存器 2 输出值
MemtoReg	O	grf 写入数据决定信号 1：DM 输出数据 0：ALU 输出数据
RegWrite	O	grf 写寄存器信号 1：写寄存器 0：无效

MemWrite	0	写内存 dm 信号 1：写入内存 0：无效
nPC_sel	0	跳转信号 Zero 为 1 时： 1：跳转 0：无效 Zero 为 0 时： 无效
EXTop[1:0]	0	扩展控制信号 00：无符号扩展 01：低 16 位补 0 10：有符号扩展 11：有符号扩展后逻辑左移两位
ALUctr[2:0]	0	alu 控制信号 000：加运算 001：减运算 010：或运算 011：输出写入数据 2 100:异或运算
jr	0	jr 信号 1：jr 指令 0：无效
jal	0	jal 信号 1：jal 指令 0：无效

2. 真值表

表格 16ctr 真值表

func	100001	100011	001000								
op	000000	000000	000000	001101	100011	101011	000100	001111	000000	001110	000011
	addu	subu	jr	ori	lw	sw	beq	lui	nop	xori	jal
RegDst	1	1	0	0	0	x	x	0	0	0	0
ALUSrc	0	0	0	1	1	1	0	1	0	1	0
MemtoReg	0	0	0	0	1	x	x	x	0	0	0
RegWrite	1	1	0	1	1	0	0	1	0	1	1
MemWrite	0	0	0	0	0	1	0	0	0	0	0
nPC_sel	0	0	0	0	0	0	1	0	0	0	0
EXTop	x	x	00	00	10	10	11	01	00	00	00
ALUctr	000	001	000	010	000	000	x	011	000	100	000
jr	0	0	1	0	0	0	0	0	0	0	0
jal	0	0	0	0	0	0	0	0	0	0	1

（四） 测试程序

（一） 测试程序

```
.data

.text
```

```
#测试 ori 指令
#第三个立即数是无符号扩展，不存在负数的情况
ori $a0, $0, 123    #测试与 0 进行 or 运算
ori $a1, $a0, 456     #测试两个非 0 数的 or 运算

#测试 lui 指令
lui $a2, 123         #测试，构造正数
lui $a3, 0xffff        #测试，构造负数

#测试 addu 指令
#无符号相加，不存在负数情况
addu $s0, $a0, $a2    #测试正数相加

#测试 subu 指令
#无符号相减，不存在负数情况
subu $s1, $a2, $a0    #测试正数相减

#测试 sw 指令
ori $t0, $t0, 0x0000     #构造 0
sw $a0, 0($t0)          #测试 0111_1011 存入内存
sw $a1, 4($t0)          #测试 1_1111_1011 存入内存
```

```
    #测试 lw 指令
    lw $s0, 0($t0)          #测试内存写寄存器
    lw $s1, 4($t0)          #测试内存写寄存器

    #测试 beq 指令
    ori $a0, $0, 1          #构造 1
    ori $a1, $0, 2          #构造 2
    ori $a2, $0, 1          #构造 1
    beq $a0, $a1, loop1     #测试不跳转
    nop
    beq $a0, $a2, loop2     #测试跳转
    nop

loop1:ori $a0, $0, 0
loop2:ori $a1, $0, 0

    #测试 jal,jr 指令
    ori $s2, $2, 1
    ori $s3, $2, 2
    addu $s4, $s2, $s3
    jal loop
    nop
```

```
   jal end
   nop

   ori $s2, $2, 3
loop:  ori $s3, $2, 4
   jr $ra
end: nop
```

（二） 机器码

```
3404007b
348501c8
3c06007b
3c07ffff
00868021
00c48823
35080000
ad040000
ad050004
8d100000
8d110004
34040001
34050002
34060001
```

```
10850003
00000000
10860002
00000000
34040000
34050000
34520001
34530002
0253a021
0c000c1c
00000000
0c000c1e
00000000
34520003
34530004
03e00008
00000000
```

（三） 期望输出

```
$a0 0x00000001
$a2 0x00000001
$a3 0xffff0000
$s0 0x0000007b
```

$s1 0x000001fb

$s2 0x00000001

$s3 0x00000004

$ra 0x00003068

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x0000007b	0x000001fb	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

图表 16MARS 输出

$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000001
$a1	5	0x00000000
$a2	6	0x00000001
$a3	7	0xffff0000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x0000007b
$s1	17	0x000001fb
$s2	18	0x00000001
$s3	19	0x00000004
$s4	20	0x00000003
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003068
pc		0x0000307c
hi		0x00000000
lo		0x00000000

图表 17MARS 输出

(四) Verilog 输出

```
@00003000: $ 4 <= 0000007b

@00003004: $ 5 <= 000001fb

@00003008: $ 6 <= 007b0000

@0000300c: $ 7 <= ffff0000

@00003010: $16 <= 007b007b

@00003014: $17 <= 007aff85
```

```
@00003018: $ 8 <= 00000000
@0000301c: *00000000 <= 0000007b
@00003020: *00000004 <= 000001fb
@00003024: $16 <= 0000007b
@00003028: $17 <= 000001fb
@0000302c: $ 4 <= 00000001
@00003030: $ 5 <= 00000002
@00003034: $ 6 <= 00000001
@0000304c: $ 5 <= 00000000
@00003050: $18 <= 00000001
@00003054: $19 <= 00000002
@00003058: $20 <= 00000003
@0000305c: $31 <= 00003060
@00003070: $19 <= 00000004
@00003064: $31 <= 00003068
```

（五） 结论

期望输出与实际输出相同。

二、 思考题

（一） 根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

输入 DM 的 addr 位宽为 10 位， 取得是 32 位 addr 的 2-11 位。直接声明[11:2]addr 则能更直观的提醒取得是 11：2 这 10 位，实际上与 9：0 没有区别。

从 alu 输出的 result 得到。

（二） 在相应的部件中，reset 的优先级比其他控制信号（不包括 clk 信号）都要高，且相应的设计都是同步复位。清零信号 reset 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

对 ifu、dm、grf 进行复位。这些部件有寄存器或 ram，清零时需要将这些寄存器或 ram 清空，来初始化 cpu。

（三） 列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

利用 if-else、case 完成操作码和控制信号的值之间的对应，如

```
case(op)
6’b000011: begin
            jal = 1;
            ……
             End
endcase
```

利用 assign 语句完成操作码和控制信号的值之间的对应，如

```
assign jr = (instruction[31:26] == 6’b0 && instruction[5:0] == 6’b001000) ? 1 : 0;
```

利用宏定义，如

```
`define beq 6'b000100

assign nPCsel = (instruction[31:26] == `beq) ? 1 : 0;
```

（四） 根据你所列举的编码方式，说明他们的优缺点。

if-else、case 方式对每个指令产生控制信号的过程简洁明了，易于添加新指令，但对每个信号的所有产生情况不够清晰；assign 对每个信号的所有产生情况集中到一起，但对一个指令所产生的所有信号不清晰，难以 debug；宏定义使得上述两种方式更加清晰。

（五） C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分 。

溢出即计算结果超出 32 位，对于支持 c 语言的 mips 指令，因 c 语言不处理溢出，addi 与 addiu 的结果相同，都看作没有溢出，即 addi 无需判断是否溢出并处理 overflow。同理 add 与 addu 等价，都不处理溢出所得 overflow 值。

（六） 根据自己的设计说明单周期处理器的优缺点。

优点：每一条指令执行过程清晰，直观；

缺点：各个元件功能单一，不能复用，造成芯片面积大、成本高的缺陷。处理 n 条指令至少需要 n 个周期，周期取决于最长数据通路的耗时，耗时较长。

（七） 简要说明 jal、jr 和堆栈的关系。

jal 跳转至函数，jr 跳转回原程序。函数调用时，$s0-$s7 八个寄存器为受保护寄存器，即在函数调用前后，这八个寄存器中的数值不能被修改，若函数对$s0-$s7 有修改，则需要用栈将原来值保存起来。当函数进行递归调用时，程序执行符合栈的先进后出模式，在执行 jal 后进栈，jr 返回后出栈，以保证受保护寄存器的值不变。