2023秋P4·CPU设计文档

Part 1 CPU设计草稿

1.1 总体概述

1.1.1 CPU类型

本CPU为**单周期MIPS - CPU**，目前由**Verilog**的**ISE**工具链实现。

1.1.2 CPU结构

CPU包含**GRF、EXT、ALU、DM、IFU、Controller**六个模块。

> 相较于**Logisim**，在**ISE**中能够快速简捷地对输入的32位指令进行解析，因此不需要单独的**Splitter**模块。
> `All_MUXs.v`文件集成了3个多路选择器，由于功能结构简单，这里不算做单个模块。

1.1.3 CPU支持的指令集

{**add, sub, ori, lw, sw, beq, lui, jal, jr, nop**}

> 要求中字面上是add和sub指令，是为了方便以后扩展功能。但是实际上本次开发只需考虑无符号数即可，更接近于addu和subu指令。

1.2 模块设计

1.2.1 GRF

1.2.1.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号：1，允许写入；0，不可写入
A1	I	5	读出数据的地址1
A2	I	5	读出数据的地址2
A3	I	5	写入数据的地址
WD	I	32	写入数据
RD1	O	32	读出数据1
RD2	O	32	读出数据2

1.2.1.2 功能定义

- 读数据
  - 读数据1：根据地址A1，读出对应的数据RD1
  - 读数据2：根据地址A2，读出对应的数据RD2
- 写数据
  - 写数据：WE为1、reset不为1且时钟上升来临时，根据地址A3，写入数据WD

    - 会按照教程所要求格式输出写入信息
- 同步复位
    - 当reset为1时，将所有寄存器清零

1.2.1.3 模块代码

```
`timescale 1ns / 1ps
module GRF (
    input         clk,
    input         reset,
    input         WE,
    input  [ 4:0] A1,
    input  [ 4:0] A2,
    input  [ 4:0] A3,
    input  [31:0] WD,
    input  [31:0] pc,
    output [31:0] RD1,
    output [31:0] RD2
);
  reg     [31:0] regfile[0:31];
  integer        i = 0;
  always @(posedge clk) begin
    if (reset == 1'b1) begin
      for (i = 0; i < 32; i = i + 1) begin
        regfile[i] <= 32'd0;
      end
    end else begin
      if (WE == 1'b1) begin
        if (A3 != 5'd0) begin
          regfile[A3] <= WD;
          //display for test
          $display("@%h: $%d <= %h", pc, A3, WD);
        end else begin
          regfile[0] <= 32'd0;
        end
      end
    end
  end
  assign RD1 = regfile[A1];
  assign RD2 = regfile[A2];
endmodule
```

1.2.2 EXT

1.2.2.1 端口定义

端口名	方向	位宽	信号描述
dataIn	I	16	输入数据,16位立即数
extOp	I	1	扩展操作类型：1，符号扩展；0，无符号扩展
dataOut	O	32	输出数据,32位立即数

1.2.2.2 功能定义

- 位扩展

- 将16位的立即数扩展为32位的立即数，提供符号扩展和无符号扩展两种方式

1.2.2.3 模块代码

```
`timescale 1ns / 1ps
module EXT (
    input      [15:0] dataIn,
    input             extOp,
    output reg [31:0] dataOut
);
  always @(*) begin
    if (extOp == 1'b0) begin
      dataOut = {{16{1'b0}}, dataIn};
    end else begin
      dataOut = {{16{dataIn[15]}}, dataIn};
    end
  end
endmodule
```

1.2.3 ALU

1.2.3.1 端口定义

端口名	方向	位宽	信号描述
A	I	32	第一个操作数
B	I	32	第二个操作数
aluOp	I	3	ALU操作类型：000，加；001，减；010，按位与；011，按位或；100，判断相等
R	O	32	运算结果
zero	O	1	相等状态码：1，相等；0，不相等

1.2.3.2 功能定义

- 算术运算
  - 加法：R = A+B
  - 减法：R = A-B
- 位运算
  - 按位与：R = A&B
  - 按位或：R = A|B
- 判断相等
  - 判断A和B是否相等，相等时zero为1，不相等时zero为0；**此时R输出端保持原值**

1.2.3.3 模块代码

```
`timescale 1ns / 1ps
module ALU (
    input      [31:0] A,
    input      [31:0] B,
    input      [ 2:0] aluOp,
    output reg [31:0] R,
    output            zero
```

```
);
  always @(*) begin
    case (aluOp)
      3'd0: R = A + B;
      3'd1: R = A - B;
      3'd2: R = A & B;
      3'd3: R = A | B;
      3'd4: R = R;
      default: R = R;
    endcase
  end
  assign zero = (A == B) ? 1 : 0;
endmodule
```

1.2.4 DM

1.2.4.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号：1，允许写入；0，不可写入
RE	I	1	读使能信号：1，允许读出；0，不可读出
A	I	32	读写地址：输入值是按字节为单位；有效范围（以字节为单位）是0x0000-0x2fff，共3072字；需要利用以字为单位的地址进行整字存取
WD	I	32	写入数据
RD	O	32	读出数据

1.2.4.2 功能定义

- 读数据
  - 读数据：RE为1时且时钟上升来临时，根据地址A，读出数据RD;否则读出0。
- 写数据
  - 写数据：WE为1时且时钟上升来临时，根据地址A，写入数据WD；否则保持原值。
  - 会按照教程所要求格式输出写入信息
- 同步复位
  - 当reset为1时，将所有内容清零

1.2.4.3 模块代码

```
`timescale 1ns / 1ps
module DM (
    input         clk,
    input         reset,
    input         WE,
    input         RE,
    input  [31:0] A,
    input  [31:0] pc,
    input  [31:0] WD,
```

```
    output [31:0] RD
);
  reg     [31:0] ram   [0:3071];
  // real address
  wire    [11:0] addr;
  integer        i = 0;
  assign addr = A[13:2];
  always @(posedge clk) begin
    if (reset == 1'b1) begin
      for (i = 0; i < 3072; i = i + 1) begin
        ram[i] <= 32'd0;
      end
    end else begin
      if (WE == 1'b1) begin
        ram[addr] <= WD;
        // display for test
        $display("@%h: *%h <= %h", pc, A, WD);
      end else begin
        ram[addr] <= ram[addr];
      end
    end
  end
  assign RD = (RE == 1'b1) ? ram[addr] : 32'b0;
endmodule
```

1.2.5 IFU

1.2.5.1 端口定义

端口名	方向	位宽	信号描述
clk	I	1	时钟信号
reset	I	1	同步复位信号，复位使PC的值为0x3000（起始地址）
zero	I	1	相等状态码：1，相等；0，不相等
offset	I	16	跳转偏移量
instr_index	I	26	跳转目标地址
rsIn	I	32	GRF[rs]读出的值
jumpOp	I	3	跳转操作类型：000，无跳转；001，beq；010；jal；011，jr
nInstr	O	32	下一条32位指令数据
pc	O	32	PC值，用于显示当前指令的地址(按字节)

> 该部分被分为**pc、npc、im三个子模块**,本设计将其写在同一个文件中，用注释区分。

1.2.5.2 功能定义

- 同步复位
  - 当reset为1时，将PC的值置为0x3000（起始地址）
- 计算下一条指令地址（调整PC值）
  - 当beq指令有效时（zero == 1 && jumpOp == 001），PC = PC + 4 + offset * 4
  - 当jal指令有效时（jumpOp == 010），PC = pc[31:28] || instr_index || 00
  - 当jr指令有效时（jumpOp == 011），PC = rsIn

    - 否则，PC = PC + 4
- 取指令
    - 根据PC的值，从IM中读出下一条指令nInstr

1.2.5.3 模块代码

```
`timescale 1ns / 1ps
module IFU (
    input             clk,
    input             reset,
    input             zero,
    input      [15:0] offset,
    input      [25:0] instr_index,
    input      [31:0] rsIn,
    input      [ 2:0] jumpOp,
    output reg [31:0] nInstr,
    output reg [31:0] pc
);
  initial begin
    pc = 32'h00003000;
  end
  reg [31:0] rom  [0:4095];
  reg [31:0] pc_;
  reg [11:0] addr;

  // Sub-module1 npc
  reg [31:0] npc;
  always @(*) begin
    case (jumpOp)
      3'd0: begin
        // none
        npc = pc + 4;
      end
      3'd1: begin
        //beq
        if (zero == 1'b1) begin
          npc = pc + 4 + ({{16{offset[15]}}, offset} << 2);
        end else begin
          npc = pc + 4;
        end
      end
      3'd2: begin
        //jal
        npc = {pc[31:28], instr_index, {2{1'b0}}};
      end
      3'd3: begin
        //jr
        npc = rsIn;
      end
      default: npc = pc + 4;
    endcase
  end

  // Sub-module2 pc
  always @(posedge clk) begin
    if (reset == 1'b1) begin
      pc <= 32'h00003000;
    end else begin
```

```
        pc <= npc;
      end
    end

    // Sub-module3 im
    initial begin
      $readmemh("code.txt", rom);
    end
    always @(*) begin
      pc_    = pc - 32'h00003000;
      addr   = pc_[13:2];
      nInstr = rom[addr];
    end
  endmodule
```

1.2.6 Controller

1.2.6.1 端口定义

端口名	方向	位宽	信号描述
op	I	6	操作码
fn	I	6	功能码
regDst	O	2	寄存器堆写入地址选通信号：00，选用rt为地址写入；01，选用rd为地址写入；10，选用$ra为目标写入
aluSrc	O	1	ALU第二个操作数（B）选通信号：1，选用立即数；0，选用寄存器堆读出的数据
aluOp	O	3	ALU操作类型：000，加；001，减；010，按位与；011，按位或；100，判断相等
memToR	O	2	数据写入寄存器堆时的数据来源：00，来自ALU运算结果；01，来自DM；10，来自16位立即数；11，写入pc+4的值
memWrite	O	1	DM写使能信号：1，允许写入；0，不可写入
memRead	O	1	DM读使能信号：1，允许读出；0，不可读出
regWrite	O	1	寄存器堆写使能信号：1，允许写入；0，不可写入
extOp	O	1	扩展操作类型：1，符号扩展；0，无符号扩展。仅负责ALU两个操作数的符号扩展
jumpOp	O	3	跳转操作类型：000，无跳转；001，beq ; 010 , jal ; 011 , jr

1.2.6.2 功能定义

- 控制信号生成
  - 识别指令的操作码和功能码，根据指令的不同，生成不同的控制信号

1.2.6.3 指令识别对照表

op[5:0]、fn[5:0]与九种指令（除nop外）的对应关系如下表所示：

op[5:0]	fn[5:0]	add	sub	ori	lw	sw	beq	lui	jal	jr
000000	100000	1	0	0	0	0	0	0	0	0
000000	100010	0	1	0	0	0	0	0	0	0

op[5:0]	fn[5:0]	add	sub	ori	lw	sw	beq	lui	jal	jr
001101	x	0	0	1	0	0	0	0	0	0
100011	x	0	0	0	1	0	0	0	0	0
101011	x	0	0	0	0	1	0	0	0	0
000100	x	0	0	0	0	0	1	0	0	0
001111	x	0	0	0	0	0	0	1	0	0
000011	x	0	0	0	0	0	0	0	1	0
000000	001000	0	0	0	0	0	0	0	0	1

> 对于nop，指令码为0x00000000，op[5:0]为000000，fn[5:0]为000000，均为0，故不在表中列出。

1.2.6.4 控制信号生成对应表

所有控制信号与指令识别子模块的输出信号的对应关系如下表所示（展示的是每个信号为1时所需要的控制信号情况）：

信号名	regDst	aluSrc	aluOp[2:0]	memToR[1:0]	memWrite	memRead	regWrite	extOp	jumpOp[2:0]
add	01	0	000	00	0	0	1	x	000
sub	01	0	001	00	0	0	1	x	000
ori	00	1	011	00	0	0	1	0	000
lw	00	1	000	01	0	1	1	1	000
sw	x	1	000	x	1	0	0	1	000
beq	x	0	100	x	0	0	0	x	001
lui	00	x	x	10	0	0	1	x	000
jal	10	x	x	11	0	0	1	x	010
jr	x	x	x	x	0	0	0	x	011

1.2.6.5 模块代码

```
`timescale 1ns / 1ps
module Controller (
    input      [5:0] op,
    input      [5:0] fn,
    output reg [1:0] regDst,
    output reg       aluSrc,
    output reg [2:0] aluOp,
    output reg [1:0] memToR,
    output reg       memWrite,
    output reg       memRead,
    output reg       regWrite,
    output reg       extOp,
    output reg [2:0] jumpOp
);
  always @(*) begin
```

```
      case (op)
        6'b000000: begin
          //R-type
          regDst   = 2'b01;
          aluSrc   = 0;
          memRead  = 0;
          memWrite = 0;
          extOp    = extOp;
          case (fn)
            6'b100000: begin
              //add
              aluOp    = 3'b000;
              memToR   = 2'b00;
              regWrite = 1;
              jumpOp   = 3'b000;
            end
            6'b100010: begin
              //sub
              aluOp    = 3'b001;
              memToR   = 2'b00;
              regWrite = 1;
              jumpOp   = 3'b000;
            end
            6'b001000: begin
              //jr
              aluOp    = aluOp;
              memToR   = memToR;
              regWrite = 0;
              jumpOp   = 3'b011;
            end
            default: begin
              regDst   = 0;
              aluSrc   = 0;
              aluOp    = 0;
              memToR   = 0;
              memWrite = 0;
              memRead  = 0;
              regWrite = 0;
              extOp    = 0;
              jumpOp   = 0;
            end
          endcase
        end
        6'b100011: begin
          //lw
          regDst   = 2'b00;
          aluSrc   = 1;
          aluOp    = 3'b000;
          memToR   = 2'b01;
          memWrite = 0;
          memRead  = 1;
          regWrite = 1;
          extOp    = 1;
          jumpOp   = 3'b000;
        end
        6'b101011: begin
          //sw
          regDst   = regDst;
          aluSrc   = 1;
          aluOp    = 3'b000;
```

```
          memToR   = memToR;
          memWrite = 1;
          memRead  = 0;
          regWrite = 0;
          extOp    = 1;
          jumpOp   = 3'b000;
        end
        6'b000100: begin
          //beq
          regDst   = regDst;
          aluSrc   = 0;
          aluOp    = 3'b100;
          memToR   = memToR;
          memWrite = 0;
          memRead  = 0;
          regWrite = 0;
          extOp    = extOp;
          jumpOp   = 3'b001;
        end
        6'b001101: begin
          //ori
          regDst   = 2'b00;
          aluSrc   = 1;
          aluOp    = 3'b011;
          memToR   = 2'b00;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = 0;
          jumpOp   = 3'b000;
        end
        6'b001111: begin
          //lui
          regDst   = 2'b00;
          aluSrc   = aluSrc;
          aluOp    = aluOp;
          memToR   = 2'b10;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = extOp;
          jumpOp   = 3'b000;
        end
        6'b000011: begin
          //jal
          regDst   = 2'b10;
          aluSrc   = aluSrc;
          aluOp    = aluOp;
          memToR   = 2'b11;
          memWrite = 0;
          memRead  = 0;
          regWrite = 1;
          extOp    = extOp;
          jumpOp   = 3'b010;
        end
        default: begin
          regDst   = 0;
          aluSrc   = 0;
          aluOp    = 0;
          memToR   = 0;
```

```
            memWrite = 0;
            memRead  = 0;
            regWrite = 0;
            extOp    = 0;
            jumpOp   = 0;
          end
        endcase
      end
    endmodule
```

Part 2 测试方案

本次进行的测试方法是Mars对拍测试。

构造一段MIPS程序，输入Mars中运行；
利用Mars导出机器码，形成code.txt文件，用来载入自己的cpu的IM中，仿真运行；
比对两者按统一格式输出的内容。即如下两种格式：

```
//grf
$display("@%h: $%d <= %h", WPC, Waddr, WData);
//dm
$display("@%h: *%h <= %h", pc, addr, din);
```

注意:

- 需要用到魔改版的Mars来获取标准的输出
- 这里统一规定：对于0号寄存器的写入不进行输出

2.1 测试代码与结果1

> 这是P3用到的测试代码，用来检查原有功能是否仍然正确。

- 测试代码：

```
ori $t1, $zero, 0
ori $t2, $t1, 5
ori $t3, $zero, 7
ori $a0, $zero, 9
ori $a1, $zero, 13
add $t1, $t1, $t3
add $a0, $a0, $t3
sub $a1, $a1, $t3
beq $t1, $t2, next
add $t1, $t1, $t3
beq $t1, $t2, next
next:sub $t1, $t1, $t3
lui $t4, 0xfff
lui $t6, 0x123
ori $t2, $zero, 4
sw $t6, 8($t2)
sw $t4, 0x14($t2)
lw $t5, 0x14($t2)
lw $t7, 8($t2)
```

- Mars运行结果/CPU运行结果：

```
@00003000: $ 9 <= 00000000
@00003004: $10 <= 00000005
@00003008: $11 <= 00000007
@0000300c: $ 4 <= 00000009
@00003010: $ 5 <= 0000000d
@00003014: $ 9 <= 00000007
@00003018: $ 4 <= 00000010
@0000301c: $ 5 <= 00000006
@00003024: $ 9 <= 0000000e
@0000302c: $ 9 <= 00000007
@00003030: $12 <= 0fff0000
@00003034: $14 <= 01230000
@00003038: $10 <= 00000004
@0000303c: *0000000c <= 01230000
@00003040: *00000018 <= 0fff0000
@00003044: $13 <= 0fff0000
@00003048: $15 <= 01230000
```

2.2 测试代码与结果2

> 这是一份覆盖比较全面的代码

```
ori $0, $0, 136
ori $1, $0, 16
ori $2, $0, 68
ori $3, $0, 120
ori $4, $0, 120
ori $5, $0, 36
ori $6, $0, 120
ori $7, $0, 136
ori $8, $0, 136
ori $9, $0, 156
ori $10, $0, 140
ori $11, $0, 84
ori $12, $0, 48
ori $13, $0, 36
ori $14, $0, 0
ori $15, $0, 84
ori $16, $0, 52
ori $17, $0, 132
ori $18, $0, 48
ori $19, $0, 92
ori $20, $0, 16
ori $21, $0, 128
ori $22, $0, 64
ori $23, $0, 76
ori $24, $0, 72
ori $25, $0, 120
ori $26, $0, 104
ori $27, $0, 92
ori $28, $0, 84
ori $29, $0, 0
ori $30, $0, 108
ori $31, $0, 52
```

```
sub $2,$2,$2
sw $3,4($2)
ori $17,$11,156
ori $8,$30,72
jal Test1
nop

Back1:
lui $9,92
lui $4,132
ori $15,$4,80
jal Test2
nop

Back2:
sub $15,$15,$15
lw $17,36($15)
ori $0,$23,64
lui $7,112
jal Test3
nop

Back3:
sub $10,$29,$16
sub $23,$23,$23
lw $5,72($23)
lui $7,124
jal Test4
nop

Back4:
lui $26,4
lui $9,44
add $3,$1,$13
jal Test5
nop

Back5:
sub $0,$0,$0
lw $2,100($0)
ori $7,$7,108
ori $17,$28,28
jal Test6
nop

Back6:
sub $30,$17,$29
sub $5,$5,$5
lw $28,64($5)
sub $4,$4,$4
lw $2,80($4)
jal Test7
nop

Back7:
sub $14,$14,$14
lw $26,116($14)
add $21,$3,$1
ori $29,$18,8
jal Test8
```

```
nop

Back8:
sub $22,$25,$25
sub $16,$16,$16
sw $22,84($16)
lui $0,0
jal Test9
nop

Back9:
lui $30,12
sub $18,$18,$18
sw $7,44($18)
nop
jal Test10
nop

Back10:
sub $23,$3,$19
nop
add $18,$5,$6
jal Test11
nop

Back11:
sub $4,$9,$3
lui $27,56
sub $17,$17,$17
lw $0,52($17)
jal Test12
nop

Back12:
sub $8,$8,$8
sw $0,32($8)
add $0,$14,$21
sub $17,$17,$17
lw $16,108($17)
jal Test13
nop

Back13:
nop
sub $3,$3,$3
sw $16,100($3)
sub $1,$2,$21
jal Test14
nop

Back14:
nop
add $12,$7,$5
sub $2,$2,$2
lw $24,80($2)
jal Test15
nop

jal End
```

```
Test1:
nop
sub $13,$17,$7
ori $23,$3,152
ori $21,$11,0
add $17,$6,$18
nop
lui $16,44
beq $22,$12,EndTest1
nop
nop
sub $26,$26,$26
lw $30,104($26)
sub $0,$0,$0
lw $12,24($0)
sub $25,$6,$14
sub $0,$21,$19
beq $7,$2,EndTest1
nop
nop
sub $29,$29,$29
sw $14,104($29)
lui $15,112
add $0,$26,$0
sub $23,$23,$23
lw $8,104($23)
sub $9,$28,$26
sub $7,$7,$7
lw $1,124($7)
sub $23,$11,$11
EndTest1:
jr $ra
nop


Test2:
sub $2,$2,$2
sub $17,$17,$17
ori $17,$17,10
sub $1,$1,$1
ori $1,$1,1
SubTest2:
add $2,$2,$1
beq $2,$17,EndTest2
nop
sub $26,$3,$25
sub $5,$5,$5
sw $30,32($5)
nop
nop
sub $3,$19,$7
sub $20,$10,$12
nop
sub $6,$6,$6
sw $18,48($6)
sub $3,$3,$3
lw $11,56($3)
sub $28,$28,$28
lw $24,156($28)
lui $0,48
```

```
ori $19,$13,4
beq $7,$11,SubTest2
nop
nop
beq $30,$20,SubTest2
nop
nop
sub $29,$29,$29
sw $22,44($29)
sub $29,$29,$29
sw $14,96($29)
sub $28,$28,$28
lw $11,124($28)
ori $24,$3,28
sub $22,$22,$22
lw $30,12($22)
ori $21,$16,48
EndTest2:
jr $ra
nop


Test3:
sub $21,$21,$21
sw $18,112($21)
add $3,$2,$27
ori $7,$16,36
nop
add $8,$7,$11
sub $25,$25,$25
lw $0,40($25)
add $12,$6,$18
nop
sub $27,$27,$27
sw $21,108($27)
ori $14,$11,100
sub $14,$14,$14
lw $5,148($14)
lui $30,20
beq $11,$30,EndTest3
nop
nop
sub $22,$22,$22
lw $8,112($22)
beq $7,$26,EndTest3
nop
nop
sub $12,$28,$18
lui $13,144
sub $23,$23,$3
beq $1,$4,EndTest3
nop
nop
lui $21,136
EndTest3:
jr $ra
nop


Test4:
```

```
sub $5,$5,$5
sw $28,52($5)
nop
ori $19,$4,84
add $11,$1,$28
add $20,$23,$16
ori $25,$0,80
sub $20,$20,$20
lw $7,24($20)
nop
sub $14,$14,$14
sw $25,0($14)
sub $9,$9,$23
sub $28,$25,$1
nop
add $11,$20,$11
sub $13,$10,$14
sub $20,$20,$20
sw $29,24($20)
sub $28,$28,$22
sub $23,$23,$23
lw $14,80($23)
sub $8,$13,$10
sub $30,$20,$25
sub $23,$5,$3
EndTest4:
jr $ra
nop


Test5:
sub $28,$28,$28
lw $6,88($28)
add $19,$3,$11
sub $8,$8,$8
sw $27,68($8)
add $28,$28,$13
sub $24,$24,$24
sw $16,100($24)
sub $22,$5,$26
sub $4,$4,$4
lw $8,80($4)
lui $29,136
sub $26,$30,$26
ori $23,$2,124
sub $19,$24,$13
lui $28,76
beq $4,$15,EndTest5
nop
nop
ori $6,$22,64
beq $5,$21,EndTest5
nop
nop
sub $11,$11,$11
lw $8,144($11)
sub $21,$21,$21
lw $2,144($21)
beq $27,$17,EndTest5
nop
```

```
nop
ori $0,$19,88
beq $10,$30,EndTest5
nop
nop
EndTest5:
jr $ra
nop


Test6:
sub $24,$24,$24
sub $26,$26,$26
ori $26,$26,10
sub $3,$3,$3
ori $3,$3,1
SubTest6:
add $24,$24,$3
beq $24,$26,EndTest6
nop
nop
nop
nop
sub $30,$30,$30
lw $21,88($30)
sub $11,$10,$25
sub $10,$13,$6
ori $10,$16,24
sub $17,$17,$17
lw $5,132($17)
lui $9,52
add $28,$0,$0
nop
add $6,$4,$9
lui $9,108
sub $6,$6,$6
sw $4,4($6)
sub $28,$28,$28
sw $19,88($28)
add $1,$14,$8
sub $20,$20,$20
sw $14,152($20)
sub $20,$20,$20
sw $8,56($20)
nop
add $12,$14,$13
EndTest6:
jr $ra
nop


Test7:
sub $12,$12,$12
sub $20,$20,$20
ori $20,$20,10
sub $15,$15,$15
ori $15,$15,1
SubTest7:
add $12,$12,$15
beq $12,$20,EndTest7
```

```
nop
sub $29,$29,$29
lw $5,40($29)
ori $29,$19,128
sub $16,$16,$16
sw $2,72($16)
ori $27,$30,0
sub $4,$4,$4
sw $22,12($4)
ori $0,$23,12
add $6,$29,$14
nop
lui $9,12
sub $27,$24,$7
sub $21,$23,$8
sub $13,$13,$13
lw $3,76($13)
sub $6,$6,$6
lw $28,28($6)
sub $1,$29,$25
lui $22,88
sub $8,$8,$8
lw $29,60($8)
beq $16,$7,SubTest7
nop
nop
nop
beq $22,$5,SubTest7
nop
nop
nop
EndTest7:
jr $ra
nop


Test8:
sub $20,$20,$20
sub $10,$10,$10
ori $10,$10,10
sub $8,$8,$8
ori $8,$8,1
SubTest8:
add $20,$20,$8
beq $20,$10,EndTest8
nop
lui $15,80
sub $21,$4,$27
add $17,$15,$7
nop
add $27,$18,$22
nop
add $12,$6,$9
lui $1,4
add $12,$29,$1
nop
add $28,$24,$23
nop
sub $30,$30,$30
lw $9,156($30)
```

```
add $28,$14,$21
lui $23,72
lui $29,120
ori $24,$0,60
sub $7,$7,$7
sw $11,40($7)
add $21,$15,$18
beq $17,$10,SubTest8
nop
nop
EndTest8:
jr $ra
nop


Test9:
sub $10,$10,$10
sub $8,$8,$8
ori $8,$8,10
sub $26,$26,$26
ori $26,$26,1
SubTest9:
add $10,$10,$26
beq $10,$8,EndTest9
nop
add $5,$9,$19
sub $2,$23,$4
nop
sub $7,$7,$7
lw $7,40($7)
nop
sub $20,$16,$6
lui $5,44
lui $16,112
nop
beq $30,$11,SubTest9
nop
nop
beq $29,$13,SubTest9
nop
nop
sub $12,$14,$0
add $11,$27,$22
sub $12,$12,$12
lw $6,108($12)
add $21,$11,$20
lui $27,72
nop
sub $14,$27,$0
nop
sub $21,$21,$21
sw $18,24($21)
EndTest9:
jr $ra
nop


Test10:
ori $23,$27,108
add $5,$2,$9
```

```
ori $21,$23,84
sub $22,$22,$22
lw $21,124($22)
sub $17,$20,$10
add $30,$21,$21
sub $21,$21,$21
sw $22,88($21)
sub $29,$29,$29
sw $9,40($29)
beq $21,$19,EndTest10
nop
nop
sub $22,$22,$22
sw $26,52($22)
nop
sub $26,$26,$26
sw $29,68($26)
lui $5,148
ori $29,$1,76
sub $29,$29,$29
lw $22,116($29)
sub $23,$23,$23
sw $29,36($23)
ori $6,$5,132
lui $28,140
ori $28,$3,56
add $23,$2,$26
EndTest10:
jr $ra
nop


Test11:
sub $17,$17,$17
sw $3,96($17)
sub $12,$12,$12
sw $10,124($12)
add $5,$5,$5
sub $19,$19,$19
sw $9,152($19)
sub $1,$1,$1
lw $7,72($1)
sub $12,$12,$12
sw $27,148($12)
lui $21,72
nop
add $11,$24,$28
beq $8,$8,EndTest11
nop
nop
nop
beq $23,$0,EndTest11
nop
nop
ori $17,$1,116
sub $5,$24,$19
add $20,$9,$9
add $20,$5,$30
nop
beq $8,$3,EndTest11
```

```
nop
nop
beq $25,$10,EndTest11
nop
nop
nop
EndTest11:
jr $ra
nop


Test12:
sub $22,$22,$22
sub $29,$29,$29
ori $29,$29,10
sub $21,$21,$21
ori $21,$21,1
SubTest12:
add $22,$22,$21
beq $22,$29,EndTest12
nop
add $20,$25,$11
lui $27,72
add $18,$15,$14
sub $24,$0,$10
add $10,$11,$13
lui $7,120
add $25,$30,$12
sub $0,$0,$0
lw $10,76($0)
beq $30,$24,SubTest12
nop
nop
lui $4,108
sub $28,$28,$28
sw $11,8($28)
beq $21,$9,SubTest12
nop
nop
ori $13,$13,40
nop
add $2,$7,$16
lui $11,140
sub $23,$25,$5
sub $3,$30,$23
sub $30,$28,$25
sub $28,$6,$13
EndTest12:
jr $ra
nop


Test13:
ori $30,$2,144
sub $11,$11,$11
sw $23,92($11)
ori $20,$28,52
sub $17,$17,$17
sw $15,56($17)
lui $1,8
```

```
sub $8,$8,$8
lw $1,92($8)
lui $0,100
beq $11,$23,EndTest13
nop
nop
sub $6,$27,$18
sub $8,$8,$8
lw $3,36($8)
nop
sub $26,$26,$26
lw $21,92($26)
lui $3,132
nop
beq $3,$11,EndTest13
nop
nop
sub $25,$25,$25
sw $14,0($25)
sub $15,$15,$15
lw $21,12($15)
beq $23,$7,EndTest13
nop
nop
sub $17,$17,$17
lw $23,84($17)
ori $29,$15,68
EndTest13:
jr $ra
nop


Test14:
nop
sub $6,$20,$5
ori $20,$17,72
sub $0,$0,$0
sw $30,60($0)
sub $9,$9,$9
lw $12,12($9)
ori $28,$23,56
sub $7,$7,$7
lw $28,20($7)
nop
ori $20,$5,124
beq $8,$30,EndTest14
nop
nop
ori $29,$24,148
beq $17,$14,EndTest14
nop
nop
sub $2,$2,$2
sw $3,16($2)
sub $4,$7,$7
sub $24,$24,$24
lw $28,40($24)
beq $8,$15,EndTest14
nop
nop
```

```
ori $24,$3,24
lui $0,156
beq $25,$27,EndTest14
nop
nop
sub $5,$20,$30
EndTest14:
jr $ra
nop


Test15:
sub $17,$17,$17
sub $14,$14,$14
ori $14,$14,10
sub $30,$30,$30
ori $30,$30,1
SubTest15:
add $17,$17,$30
beq $17,$14,EndTest15
nop
ori $13,$26,100
sub $21,$21,$21
sw $11,56($21)
sub $6,$13,$9
sub $1,$3,$12
add $27,$0,$21
nop
sub $23,$23,$23
sw $8,132($23)
ori $2,$8,96
sub $7,$7,$7
lw $29,16($7)
add $21,$15,$18
add $4,$10,$4
sub $22,$8,$5
sub $22,$22,$22
lw $15,100($22)
lui $12,0
beq $15,$23,SubTest15
nop
nop
sub $11,$11,$3
beq $0,$12,SubTest15
nop
nop
sub $8,$8,$8
sw $12,140($8)
add $18,$15,$7
ori $21,$1,108
EndTest15:
jr $ra
nop


End:
```

- Mars运行结果/CPU运行结果：

> 篇幅原因，只展示前30行。

```
@00003004: $ 1 <= 00000010
@00003008: $ 2 <= 00000044
@0000300c: $ 3 <= 00000078
@00003010: $ 4 <= 00000078
@00003014: $ 5 <= 00000024
@00003018: $ 6 <= 00000078
@0000301c: $ 7 <= 00000088
@00003020: $ 8 <= 00000088
@00003024: $ 9 <= 0000009c
@00003028: $10 <= 0000008c
@0000302c: $11 <= 00000054
@00003030: $12 <= 00000030
@00003034: $13 <= 00000024
@00003038: $14 <= 00000000
@0000303c: $15 <= 00000054
@00003040: $16 <= 00000034
@00003044: $17 <= 00000084
@00003048: $18 <= 00000030
@0000304c: $19 <= 0000005c
@00003050: $20 <= 00000010
@00003054: $21 <= 00000080
@00003058: $22 <= 00000040
@0000305c: $23 <= 0000004c
@00003060: $24 <= 00000048
@00003064: $25 <= 00000078
@00003068: $26 <= 00000068
@0000306c: $27 <= 0000005c
@00003070: $28 <= 00000054
@00003074: $29 <= 00000000
@00003078: $30 <= 0000006c
```

2.3 结论

经对比内容一致，本测试通过，说明CPU的基本功能实现正确。

Part 3 思考题解答

- **阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0]？。**

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

这个 addr 信号来自于 ALU 的输出R端口 32 位中的[11:2]这10位。
我们设计的 DM 模块是以 **字** 为单位的。在数据通路连接中，很容易发现我们通过ALU 运算出的地址以 **字节** 为单位,接

入 DM 的地址A端口。1个字相当于4个字节，因此需要将32位的地址右移两位（/4）后再取低10位。

- **思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。**

  第一种
  用 `always@(*) + case` 语句实现**指令对应的控制信号如何取值。**
  代码示例参见上面的Controller模块的代码展示。

  第二种
  用`assign`语句对每个控制信号进行赋值，实现**控制信号每种取值所对应的指令**，这种方法与P3推荐的控制器设计方法有异曲同工之妙。
  代码示例如下：

```
//and gates
assign beq = (op == 6'b000100);
//or gates
assign aluSrc = ori | lw | sw ;
```

  我认为，第一种方法较之第二种的优点是**直观并且可扩展性强。**每个指令对应什么样的控制信号一目了然。每次新增指令时，只需要考虑好各信号取值，用简单的赋值语句添加即可。而第二种方法每次需要考虑逻辑关系，要重新计算或逻辑的表达式。
  第二种方法较之第一种**代码量小，更加简洁。**但是逻辑可能没有那么清晰。

- **在相应的部件中，复位信号的设计都是同步复位，这与 P3 中的设计要求不同。请对比同步复位与异步复位这两种方式的 reset 信号与 clk 信号优先级的关系。**

  异步复位，两者同优先级。clk 或是 reset 都能引起敏感变化，两者也没有很明显的关系；
  同步复位，只有 clk 才能引起敏感变化，reset 信号只是在 clk 信号为 上升沿 的时候才会起作用，因此 clk 信号优先级高于 reset 信号。

- **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。**

  add的操作过程是：

```
temp ← (GPR[rs]31||GPR[rs]) + (GPR[rt]31||GPR[rt])

if temp32 ≠ temp31 then

SignalException(IntegerOverflow)

else

GPR[rd] ← temp 31..0

endif
```

  addu的操作过程是：

```
GPR[rd] ← GPR[rs] + GPR[rt]
```

忽略溢出，也就是add操作只考虑GPR[rd] ← temp 31..0 ,temp 31..0是temp的低32位，由计算方法可知仅取决于GPR[rs]31||GPR[rs]和GPR[rt]31||GPR[rt]的低32位，与GPR[rs]31和GPR[rt]31无关。也就是GPR[rs] + GPR[rt]。和addu的操作过程一致。
对于addi和addiu同理，只是其中一个操作数变成了立即数而已。