计算机组成原理项目报告

开发者说明：

- > **12011528 张艺严**

  分工：CPU模块设计、测试和集成，ISA指令执行测试，uart接口实现

  贡献比：0.38

- > **12010302 陈志雄**

  分工： 测试场景1和2搭建，ISA指令执行测试

  贡献比：0.31

- > **12010426 赵思源**

  分工：测试场景1和2搭建

  贡献比：0.31

版本修改记录

- 1.0

  5.14 完成CPU模块设计，并完成集成，进行指令测试

- 2.0

  5.18 完成minisys所有指令测试，部分修改CPU结构

- 3.0

  5.29 完成uart接口的实现

CPU架构设计说明

- **CPU特性**
  - **ISA**

Minisys - A subset of MIPS32

Type	Name	funC(ins[5:0])
R	sll	00_0000
	srl	00_0010
	sllv	00_0100
	srlv	00_0110
	sra	00_0011
	srav	00_0111
	jr	00_1000
	add	10_0000
	addu	10_0001
	sub	10_0010
	subu	10_0011
	and	10_0100
	or	10_0101
	xor	10_0110
	nor	10_0111
	slt	10_1010
	sltu	10_1011

Type	Name	opC(Ins[31:26])
I	beq	00_0100
	bne	00_0101
	lw	10_0011
	sw	10_1011
	addi	00_1000
	addiu	00_1001
	slti	00_1010
	sltiu	00_1011
	andi	00_1100
	ori	00_1101
	xori	00_1110
	lui	00_1111

Type	Name	opC(Ins[31:26])
J	jump	00_0010
	jal	00_0011

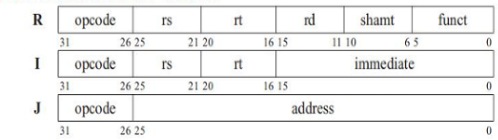
MIPS_Green_Sheet.pdf

NOTE:

Minisys is a subset of MIPS32.

The **opC** of **R-Type** instruction is **6'b00_0000**

BASIC INSTRUCTION FORMATS



- 本项目的CPU使用 **MIPS32** 这个指令集的一个子集
- CPU中的有 32 个 32-bit 位宽的寄存器用于程序执行
- 暂不支持异常处理

- **寻址空间的设计**
  - 存储结构使用的是哈佛结构，将指令空间和数据空间分离，防止出现 structure hazard
  - 指令空间和数据空间的寻址范围为0x0000_0000 ~ 0x0001_0000, 外设IO的寻址范围为0xFFFF_FC00 ~ 0xFFFF_FFFF;
  - 寻址单位为 word
- **CPI**
  - CPI 为 5，本CPU为单周期CPU且不支持流水线操作
- **CPU_TOP接口**
  - input clock_in: 从minisys开发板上的时钟Y 18获取的时钟信号
  - input FPGAReset: 与minisys开发板上的开关相绑定，用于归零PC和寄存器组
  - input start_pg: 与minisys开发板上的开关相绑定，用于启动uart串口传输模式
  - input rx: 与minisys开发板上的Y 19相绑定,用于接受串口传入的1-bit信号
  - output tx: 与minisys开发板上的V 18相绑定，用于输出从串口接收到的信号
  - input confirm：与minisys开发板上的开关相绑定，用于确认输入是否完成
  - input [2:0] index：与minisys开发板上的开关相绑定，用于控制测试场景的切换
  - input [15:0] IOReadData：与minisys开发板上的开关相绑定，用于输入数据
  - output [15:0] IOData：与minisys开发板上的led灯相绑定，用于输出需要显示的结果
  - output [5:0] control_led：与minisys开发板上的led灯相绑定，用于输出现在的控制信号的状态
  - IP核：
    - 将clock_in信号分频为23 MHZ的clock_out和10 MHZ的clock_uart

```
CPUClock clock(.clk_in1(clock_in), .clk_out1(clock_out),.clk_out2(clock_uart));
```

    - uart接口的IP核，用于执行串口通信

```
uart_bmpg_0 uart(.upg_clk_i(clock_uart),.upg_rst_i(upg_rst),.upg_rx_i(rx),.upg_clk_o(upg_clk_o)
                ,.upg_wen_o(upg_wen_o),.upg_adr_o(upg_adr_o),.upg_dat_o(upg_dat_o),.upg_done_o(upg_done_o),.upg_tx_o(tx));
```

- 子模块：

```
led LED(reset,IOWrite,writeData,IOData);

CPUClock clock(.clk_in1(clock_in), .clk_out1(clock_out),.clk_out2(clock_uart));

IFetch ifetch(Instruction, PCPlus4, linkAddr, clock_out, reset, AddrResult,linkAddr, Branch, nBranch, Jmp,
        Jal, Jr, Zero,upg_rst,upg_clk_o,upg_wen_o&(!upg_adr_o[14]),upg_adr_o[13:0],upg_dat_o,upg_done_o);

ALU alu(reset,Data1,Data2,extendedImmi,Instruction[31:26],Instruction[5:0],Instruction[10:6],PCPlus4,ALUOp,ALUSrc,IFormat,Sftmd,Signed,ALUResult,Zero,AddrResult);

Decoder decoder(reset,clock_out,confirm,index,RegWrite,Jal,MemOrIOtoReg,RegDST,Signed,Instruction[25:21],
        Instruction[20:16],Jr,Instruction[15:11],linkAddr,Instruction[15:0],ALUResult,RegWriteData,Data1,Data2,extendedImmi);

DataMemory DRAM(MemWrite,address,writeData,MemReadData,clock_out);//update

MemOrIO MOR( MemRead, MemWrite, IORead, IOWrite,AddrResult, address, MemReadData, IOReadData, RegWriteData, Data2, writeData, LEDCtrl, SwitchCtrl);

controller control(Instruction[31:26],Instruction[5:0],AddrResult[31:10],Jr,Jmp,Jal,Branch,nBranch,RegDST,MemOrIOtoReg,
        RegWrite,MemWrite,MemRead,IORead,IOWrite,ALUSrc,Sftmd,IFormat,Signed,ALUOp);
```

- **CPU内部结构**
  - led: 用来控制输出设备led灯的显示

```
module led(reset,IOWrite,writeData,IOData);
input reset; // clear the IOData
input IOWrite; // get from controller, 1 indicates the IOData should be update
input [31:0] writeData; // get from the MemOrIO, the data that should be output
output reg [15:0] IOData; // used to output the data
```

  - ifetch：用来取出指令和更新pc的值

```
output[31:0] Instruction; // the instruction fetched from this module
output reg [31:0] PCPlus4; // (pc+4) to ALU which is used by branch type instruction
output reg [31:0] linkAddr; // (pc+4) to Decoder which is used by jal instruction
input clock, reset; // Clock and reset
// from ALU
input[31:0] AddrResult; // the calculated address from ALU
input Zero; // while Zero is 1, it means the ALUresult is zero
// from Decoder
input[31:0] JrAddress; // the address of instruction used by jr instruction
// from Controller
input Branch; // while Branch is 1,it means current instruction is beq
input nBranch; // while nBranch is 1,it means current instruction is bnq
input Jmp; // while Jmp 1, it means current instruction is jump
input Jal; // while Jal is 1, it means current instruction is jal
input Jr; // while Jr is 1, it means current instruction is j

// UART Programmer Pinouts
input upg_rst_i; // UPG reset (Active High)
input upg_clk_i; // UPG clock (10MHz)
input upg_wen_i; // UPG write enable
input[13:0] upg_adr_i; // UPG write address
input[31:0] upg_dat_i; // UPG write data
input upg_done_i; // 1 if program finishe
```

  - ALU：用来执行算数运算和逻辑运算以及地址计算，并将结果输出

```
input reset;
// from Decoder
input[31:0] Data1; //the source of Ainput
input[31:0] Data2; //one of the sources of Binput
input[31:0] extendedImmi; //one of the sources of Binput
// from IFetch
input[5:0] Op; //instruction[31:26]
input[5:0] Func; //instructions[5:0]
input[4:0] Shamt; //instruction[10:6], the amount of shift bits
input[31:0] PCPlus4; //pc+4
// from Controller
input[1:0] ALUOp; //{ (R_format || I_format) , (Branch || nBranch) }
input ALUSrc; // 1 means the 2nd operand is an immediate (except beq,bne)
input IFormat; // 1 means I-Type instruction except beq, bne, LW, SW
input Sftmd; // 1 means this is a shift instruction
input Signed;
//output
output reg [31:0] ALUResult; // the ALU calculation result
output reg Zero; // 1 means the ALU_reslut is zero, 0 otherwise
output reg [31:0] AddrResult; // the calculated instruction address

wire[31:0] A,B;
wire signed [31:0] signedA,signedB; // two operands for calculation
wire[5:0] ExeCode; // use to generate ALU_ctrl. (I_format==0) ? Function_opcode : { 3'b000 , Opcode[2:0] };
wire[2:0] ALUCtl; // the control signals which affact operation in ALU directely
//wire[2:0] Sftm; // identify the types of shift instruction, equals to Function_opcode[2:0]
reg[31:0] ShiftResult; // the result of shift operation
reg[31:0] ALUOutputMux; // the result of arithmetic or logic calculation
wire[32:0] BranchAddr; // the calculated address of the instruction, Addr_Result is Branch_Addr[31:0]
```

- decoder：用来更新寄存器的值，以及从寄存器中取出值来输出

```
input reset;
input clock;
input confirm;
input [2:0] index; //clock signal generated by CPUClock module
//controller
input RegWrite; // 1 indicate write register(R,I(lw)), otherwise it's not
input Jal; // 1 indicate the instruction is "jal", otherwise it's not
input MemOrIOtoReg; // 1 indicate write data memory, otherwise it's not
input RegDST; // 1 indicate destination register is "rd"(R),otherwise it's "rt"(I)
input Signed;
//read address
input [4:0] rs; // R-format's source or I-format's source
input [4:0] rt; // R-format's source or I-format's destination
input Jr; // get the return address in register[31], 1 indicates the instruction is "jr", otherwise it's not "jr" output Jmp;
//write address
input [4:0] rd; // R-format's destination
//write data
input[31:0] linkAddr; // store the address into ra register
input [15:0] immi; // the immediate in instruction[15:0] in I-format
input[31:0] valueALU; // value comes from ALU
input[31:0] RegWriteData; // value comes from Memory or IO
//outputs
output reg [31:0] Data1; // output value 1, which goes to ALU directly or be the offset of jump
output reg [31:0] Data2; // output value 2, which can go to ALU or MemWriteData or extended immediate
output [31:0] extendedImmi;

reg [31:0] register[0:31];
reg start = 1'b0; // 1 indicate the program start,so that avoid adding value into register in the first posedge
```

- DataMemory：从内存空间中根据传入的地址取出数据并传出，以及执行数据存入内存

```
module DataMemory(MemWrite,address,writeData,MemReadData,clock);
input MemWrite;
// the address of memory unit which is to be
// read/writen
input[31:0] address;
// data to be wirten to the memory unit
input[31:0] writeData;
/*data to be read from the memory unit, in the left
screenshot its name is 'MemData' */
output[31:0] MemReadData;
input clock; // Clock signal
/* used to determine to write the memory unit or not,
in the left screenshot its name is 'WE' */
wire clk;
assign clk = !clock;

DRAM ram(.clka(clk),.wea(MemWrite),.addra(address[15:2]),.dina(writeData),.douta(MemReadData));
```

- MemOrIO：根据控制信号决定存入的数据来自外部输入或从内存中取出，以及决定存入数据是存入内存还是通过IO设备输出

```
//controller
input MemRead; // read memory, from Controller, data from register
input MemWrite; // write memory, from Controller, data from register or IO
input IORead; // read IO, from Controller
input IOWrite; // write IO, from Controller, data from register or memory
//data and output
input[31:0] AddrResult; // from alu_result in ALU
output[31:0] address; // address to Data-Memory
input[31:0] MemReadData; // data read from Data-Memory
input[15:0] IOReadData; // data read from IO,16 bits
output reg [31:0] RegWriteData; // data to Decoder(register file)
input[31:0] RegReadData; // data read from Decoder(register file)
output reg[31:0] writeData; // data to memory or I/O, m_wdata, io_wdata
output LEDCtrl; // LED Chip Select
output SwitchCtrl; // Switch Chip Select
```

- controller：分析指令并且对所有控制信号赋值，并将控制信号输出给其它模块使用

```
input[5:0] Op; // instruction[31..26], opcode
input[5:0] Func; // instructions[5..0], functionCode
input[21:0] AddrResultHigh; // From the execution unit Alu_Result[31..10]
output Jr ; // 1 indicates the instruction is "jr", otherwise it's not "jr" output Jmp;
output Jmp; // 1 indicates the instruction is "j"
output Jal; // 1 indicate the instruction is "jal", otherwise it's not
output Branch; // 1 indicate the instruction is "beq" , otherwise it's not
output nBranch; // 1 indicate the instruction is "bne", otherwise it's not
output RegDST; // 1 indicate destination register is "rd"(R),otherwise it's "rt"(I)
output MemOrIOtoReg; // 1 indicate read data from memory or IO and write it into register
output RegWrite; // 1 indicate write register(R,I(lw)), otherwise it's not
output MemWrite; // 1 indicate write data memory, otherwise it's not
output MemRead; // 1 indicates that the instruction needs to read from the
output IORead; // 1 indicates I/O read
output IOWrite; // 1 indicates I/O write
output ALUSrc; // 1 indicate the 2nd data is immidiate (I-format except "beq","bne")
output Sftmd; // 1 indicate the instruction is shift instruction
output IFormat;/* 1 indicate the instruction is I-type but isn't "beq","bne","LW" or "SW" */
output Signed; // 1 indicate the instruction use the signed value like add,sub,addi,slti,slt
output reg [1:0] ALUOp;/* if the instruction is R-type or I_format, ALUOp is 2'b10;
                          if the instruction is"beq" or "bne", ALUOp is 2'b01"
                          if the instruction is"lw" or "sw", ALUOp is 2'b00*/
```

测试说明

- 测试时特殊化操作：
  - 测试存在一个 `confirm` 拨码开关，控制（**$26**）的值，利于控制进程开始使用读入数据，例：

```
case0:  lw $v0,-928($zero)
        beq $26,$t9,case0
```

  - 读入数据后进行一个 `delay` 操作，防止抖动时 `confirm` 信号的多次变动而很快的经过了部分程序，达不到预期效果

```
delay:
    addi $a2,$a2,1
    bne $a2,$a0,delay
    addi $a3,$a3,1
    add $a2,$0,$0
    bne $a3,$a1,delay
    jr $31
```

- 读取数据操作，例：

```
lw $v0,-928($zero)
```

- 输出数据操作，例：

```
sw $t2,-912($0)
```

- 测试场景均使用循环输出结果，使灯亮度正常，例：

```
output_010: sw $t4,-912($zero)
        j output_010
```

- 测试存在一个 `reset` 拨码开关，作用为：清空所有寄存器中的值，但保留内存中的值。
- 具体场景中切换用例都需要使用 `reset` 和 `confirm` 进行处理。

测试场景一

- 通过立即数跳转，进入不同的八个用例
- 用例0：
  - 读入数据后通过移位判断是否为回文数，显示结果
- 用例1：
  - 读入一个数据，循环输出该数据时保持对 `confirm` 的确认，以便跳出第一个数的循环输出从而输出第二个数字
  - 将这两个数都存到内存中
- 用例2：
  - 从内存中读取先后两个数，**a**和**b**，读取按下列用法即可：

```
    lw $t2,0($0)
    lw $t3,4($0)
```

  - 使用 `and` 计算结果后循环输出
- 用例3：
  - 同用例2，改成使用 `or` 计算结果即可
- 用例4：
  - 同用例2，改成使用 `xor` 计算结果即可
- 用例5：
  - 从内存中读取两个数，使用 `sllv` 得到结果，再将结果覆盖 `a` 的值，循环输出结果

- 用例6：
  - 从内存中读取两个数，使用 `srlv` 得到结果，再将结果覆盖 `a` 的值，循环输出结果
- 用例7：
  - 从内存中读取两个数，使用 `srav` 得到结果，再将结果覆盖 `a` 的值，循环输出结果

测试场景二

- 通过读入一个数进行场景跳转
- 用例编号0
  - 读入第一个数，即数组大小，再根据输入的数组大小进行循环输入，将数值存到内存中。

```
    sw $v0,0($t2)  #t2为0x0000，数据在内存中的地址
    addi $t0,$t0,1
    addi $t2,$t2,4
    bne $t0,$t1,loop_000 #t0循环次数,t1输入的个数
```

- 用例编号1
  - 将输入数据视为无符号数排序
  - 双循环，冒泡排序
  - ```
    forOut_001:
       slt $t0, $s0, $s3      # 如果i<n，则t0=1
       beq $t0, $zero,dead_loop  # 如果t0=0，退出外层循环
       addi $s1, $s0, -1       # j = i - 1

     forIn_001:
       slti $t0, $s1, 0        # 如果s1<0，则t0=1
       bne $t0, $zero, exit2_001   # 如果t0!=0，跳转到exit2
    ```

    ```
        slt $t0, $t4, $t3      # 如果arr[j]<arr[j+1]，则t0=1
        beq $t0, $zero, exit2_001   # 不满足上面条件，跳转到exit2退出内层循环
        j swap_001 #满足条件就跳转到交换
        addi $s1, $s1, -1       # j--
        j forIn_001
      exit2_001:
        addi $s0, $s0, 1        # i++
        j forOut_001
    ```

- 用例编号2
  - 将输入数据转换成有符号数的补码
  - ```
        andi $t4,$t3,255 #数据8bit，t4复制t3的值
        srl $t4,$t4,7 #t4仅保留符号位
        bne $t4,$zero,negative #如果符号位不等于0，则为负数；否则为正数，补码为原码，直接存储
        sw $t3,80($t2) # 0x0000 + 80，数据集2存储的位置
    ```

    - ```
      negative:
          andi $t4,$t3,127 #拿出低7bit的数据（除去符号位）
          nor $t5,$t4,$zero #与0 nor 全部取反
          addi $t6,$t5,1
          sw $t6,80($t2)
          sw $t6,120($t2)
          j continue_010_1
      ```

- 用例编号3
    - 将2中的有符号数进行排序，与1的排序相同
- 用例编号4
    - 用数据集1中的最大值减去最小值

      ```
          addi $t2,$0,40 # 数据集1的起始
          lw $t3,0($t2) #取出最小值
          loop_100:  #循环直到能取最大值
          addi $t2,$t2,4
          addi $s0,$s0,1
          bne $s0,$t1,loop_100
          lw $t4,0($t2) #取出最大值
          sub $t5,$t4,$t3
      ```

- 用例编号5
    - 与4相同
- 用例编号6
    - 输入数据集编号（1或2或3）和该数据集中元素的下标(从0开始编址)，输出该数据的低8bit
- 用例编号7
    - 输入数据集中元素的下标，在输出设备上交替显示下面两组信息（间隔5秒）： 1）数据集0的编号（0）、用户指定的元素下标、数据集0中该下标对应的元素的低8bit 2）数据集2的编号（2）、用户指定的元素下标、数据集2中该下标对应的元素的低8bit
    - ```
      F1:   addi $t7,$zero,1000
            addi $t4,$t4,-1
            beq $t4,$zero,Type2
      Out1: sw $t5,-912($zero)
            beq $t7,$zero,F1
            addi $t7,$t7,-1
            j Out1
      ```

    - 通过不断循环几条指令来达到延时输出的效果
- 上述用例均通过了上板测试。

问题及总结

- **CPU中的问题及处理**
    - 由于我们设计中是在时钟上升沿来执行写入，所以在CPU的第一个周期的上升沿有可能会误操作(如果你的第一条指令涉及寄存器的写入)，进行一次不应该的赋值:

        - 我们在decoder中加入了一个控制信号start，在第一个时钟上升沿将其置为1，而只有在start为1时才能执行寄存器写入
    - 在进行CPU顶层集成的时候，我们连接端口时因为名字使用的不规范导致功能出现了错误，比如内存写入不成功等
        - 后来仔细检查后，修改了部分信号名字，进行了顶层连线的检查。
- **测试场景（minisys代码）的问题及处理**
    - 开始测试时输出的灯亮度极低，后修改为循环输出结果恢复正常
    - 拨码开关移动时，中间状态可能读入多种值，需要一个 `confirm` 信号决定是否程序向下进行
    - `confirm` 开关拨动时容易抖动，可能短时间内重复输入相同值，分析后添加 `delay` 模块延迟数据，成功解决问题
    - 刚开始设计的排序使用到了**栈指针**，因为没有给特定寄存器赋初值，导致栈指针存储的地址变成了输出地址。后改为不用栈解决
    - 在切换用例时没有保存数组大小，导致后续的用例进入死循环，后将数组大小存到内存中，再在用例开始时赋初值。