**北京科技大学 计算机与通信工程学院**

**课程设计报告**

**课程名称**： 计算机组成原理课程设计

**学生姓名**： 张涵棋

**专 业**： 计算机科学与技术

**班 级**： 计192

**学 号**： 41924076

**指导教师**： 张磊

**报告成绩**：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**实验地点**： 机电楼304

**实验时间**：2021年9月11日----2021年11月20日

**北京科技大学实验报告**

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级：计192

姓名： 张涵棋 学号： 41924076 实验日期： 2021 年 12月 17日

**一、课设目的与要求**

* 学会处理器的设计方法：单周期/流水线。
* 掌握处理器设计过程中指令扩展的方法。
* 能够运用现代工具独立实现一个完整的处理器。
* 了解处理器功能测试的方法：仿真测试及FPGA测试。
* 了解处理器性能测试的方法。
* 计算机系统观的建立，对所设计的处理器在整个计算机系统中的位置有所了解。

**二、实验设备（环境）及要求**

龙芯实验箱一体化实验平台/CG平台/流水线处理器关键技术虚拟仿真实验平台

OS：Win10 64位

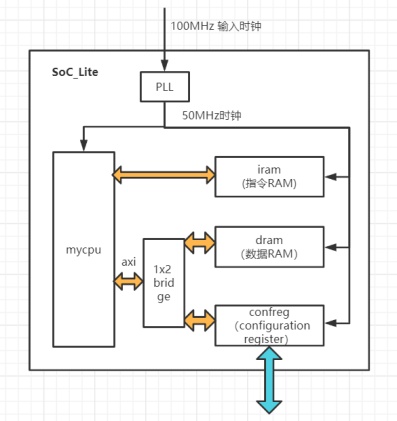
Software：Vivado2018.3开发工具

VirtualBox虚拟机+Ubuntu16.04.6（正文用五号宋体）

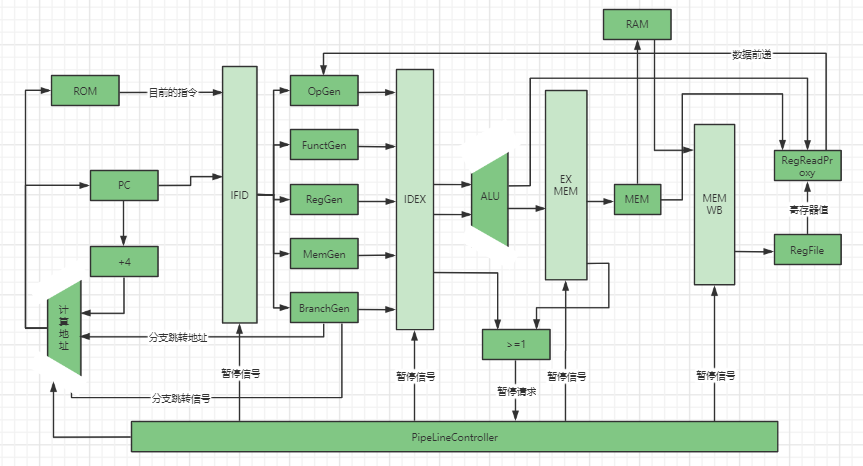
**三、设计过程与结果分析**

**Part 1 验证实验，代码阅读分析**

**1 TinyMIPS\_Extend总体结构框图**



硬件结构图



数据通路图

**2 ADDIU指令（典型指令）的设计过程**

**2.1指令格式**

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| 001001 | Rs | Rt | Imm |

ADDIU是I型指令。功能是将Rs的值与有符号扩展的Imm相加，结果送入Rt。

**2.2实现代码（需有恰当的注释）**

**2.2.1 操作码宏定义**

|  |
| --- |
| ‘define OP\_ADDIU 6'b001001 |

**2.2.2 在ID的RegGen模块，生成访问寄存器的信号**

|  |
| --- |
| case (op)        `OP\_ADDIU: begin          reg\_read\_en\_1 <= 1;//给regfile的使能信号1是有效          reg\_read\_en\_2 <= 0;//使能信号2无效，因为其中一个源操作数是立即数          reg\_addr\_1 <= rs;//给regfile的地址1是rs，即从rs中读出操作数          reg\_addr\_2 <= 0;  end |

**2.2.3在ID的FunctGen模块，生成功能码**

|  |
| --- |
| case (op)        `OP\_ADDIU: funct <= `FUNCT\_ADDU;//生成加法运算的功能码  endcase |

**2.2.4 ID的OperandGen，取操作数**

|  |
| --- |
| wire[`DATA\_BUS] sign\_ext\_imm = {{16{imm[15]}}, imm};//有符号拓展  case (op)//取操作数1      `OP\_ADDIU: begin          operand\_1 <= reg\_data\_1;        end  case (op)//取操作数2        `OP\_ADDIU: begin          operand\_2 <= sign\_ext\_imm;  end |

**2.2.5 在EX，进行运算**

|  |
| --- |
| //将处理后的操作数2与操作数1相加，得到result\_sum  wire[`DATA\_BUS] result\_sum = operand\_1 + operand\_2\_mux;  //然后进行加法运算  case (funct)  `FUNCT\_ADD: result <= result\_sum;  end |

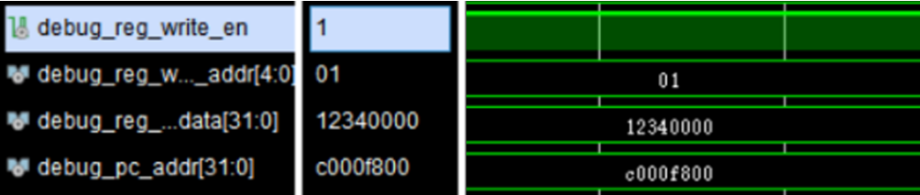
**2.2.6 WB级的RegFile模块，将结果写回到寄存器rt中**

|  |
| --- |
| //当写使能与目标寄存器合法的时候，将数据写回（ADDIU写回到寄存器rt中）      else if (write\_en && |write\_addr) begin        registers[write\_addr] <= write\_data;      end |

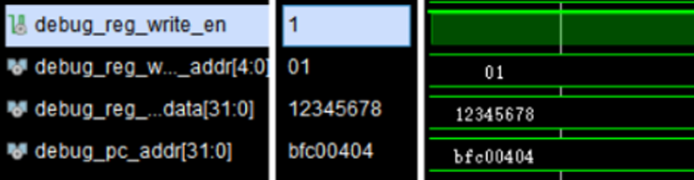
**2.3仿真测试**（需列出指令的测试波形以及程序段的测试结果并分别说明）

**2.3.1指令的测试波形**

执行指令lui $8, 0x1234的仿真结果如下：



然后再执行指令addiu $8,$8,0x5678的仿真结果如下：

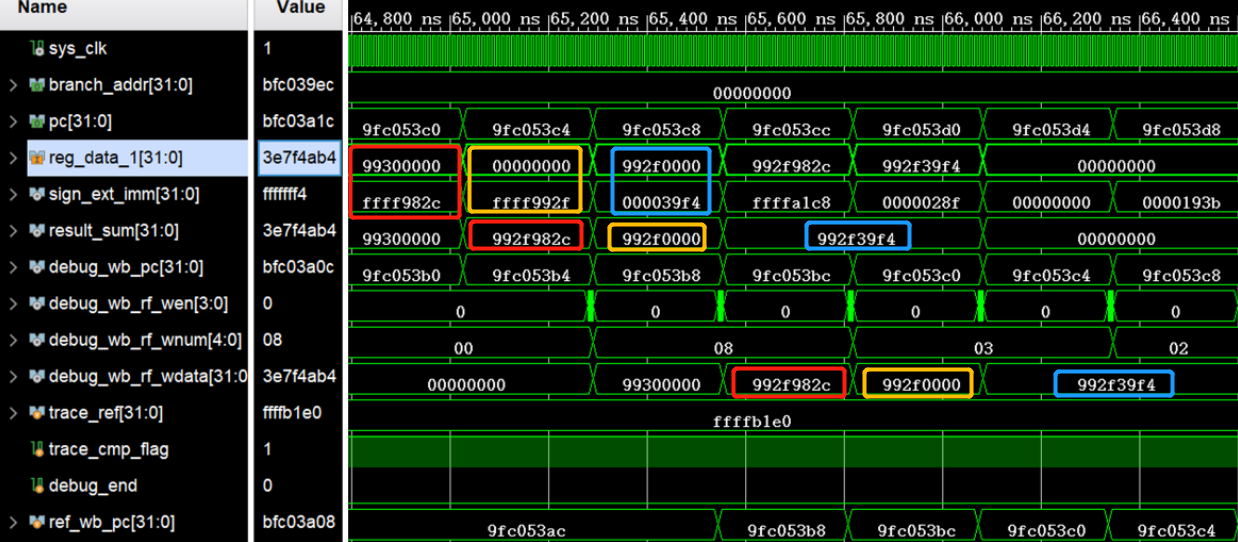


说明：首先执行指令lui $8, 0x1234，把0x1234写入8号寄存器高16位，看到debug\_reg\_...data[31:0]信号为0x12340000，说明将该数写入寄存器了。

然后，执行指令addiu $8,$8,0x5678，把0x5678和8号寄存器相加，结果写入8号寄存器。看到debug\_reg\_...data[31:0]信号为0x12345678，说明将该数写入寄存器了。

从而证明ADDIU指令是正确的。

**2.3.2程序段的测试结果**



说明：由仿真可以看出，第n个PC的Reg\_data\_1和 sign\_ext\_imm ，它们之和在第n+1个PC的result\_sum[31:0]（EX级）中体现，并且在第n+3个pc的时候被写回（WB级）。

**2.4 FPGA测试**

上板结果与仿真结果一样，正确。如下图。



**Part 2 扩展实验（在TinyMIPS\_Extend基础上扩展指令，运算类、跳转类、访存类均有）**

**总述:共实现了“19+异常”条指令扩展，分别是：**

**slti,sltiu,j,add,addi,sub,nor,xori,sra,bgez,bgtz,blez,bltz,bltzal,bgezal,lh,lhu,sh,srl,ri异常等指令**

**其中指令SLTI,BGEZ,SH的设计实现过程如下：**（三类指令分别扩展一条以上）

（最后仿照小组其他同学的代码写了ft,soft\_int，但没有试，可能不太正确）

**1 指令格式**

* 1. **运算类：SLTI**

31 26 25 21 20 16 15 11 10 6 5 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs | rt | rd | 00000 | 101011 |

汇编格式：SLTI rt, rs, imm

* 1. **跳转类：BGEZ**

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| 000001 | rs | 00001 | offset |

汇编格式：BGEZ rs, offset

**1.3访存类：SH**

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| 101001 | base | rt | offset |

汇编格式：SH rt, offset(base)

**2分析指令功能及执行过程，画出数据通路图**

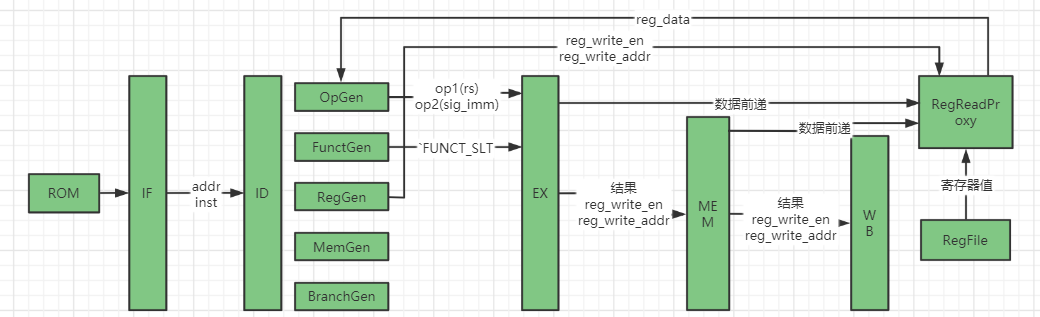
**2.1 SLTI**

1）功能：将寄存器 rs 的值与有符号扩展至 32 位的立即数 imm 进行有符号数比较，如果寄存器 rs 中的值小，则寄存器 rt 置 1；否则寄存器 rt 置 0。

2）执行过程：

取出SLTIU指令。译码（Rs读使能，Rt写使能，生成funct码，获取rs为操作数1，立即数符号扩展为操作数2）。执行（将操作数1小于2的标志零扩展，作为结果）。写回（运算结果写入rt）

3）数据通路图：



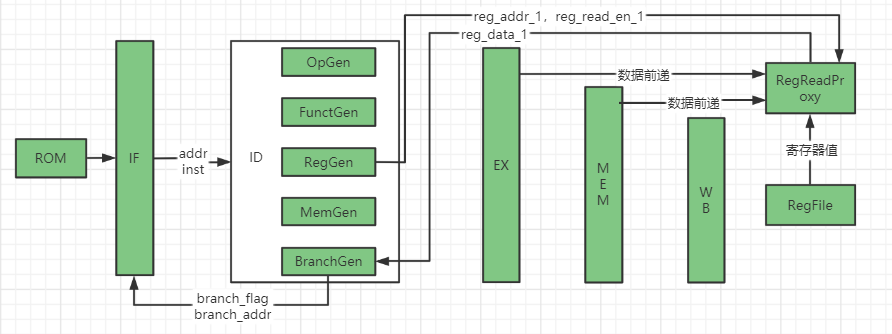
**2.2BGEZ**

1）功能：如果寄存器 rs 的值大于等于 0 则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

2）执行过程：

取出BGEZ指令。译码（Rs和rt读使能，生成funct码，产生分支标志、分支地址、延迟槽标志）。

1. 数据通路图：

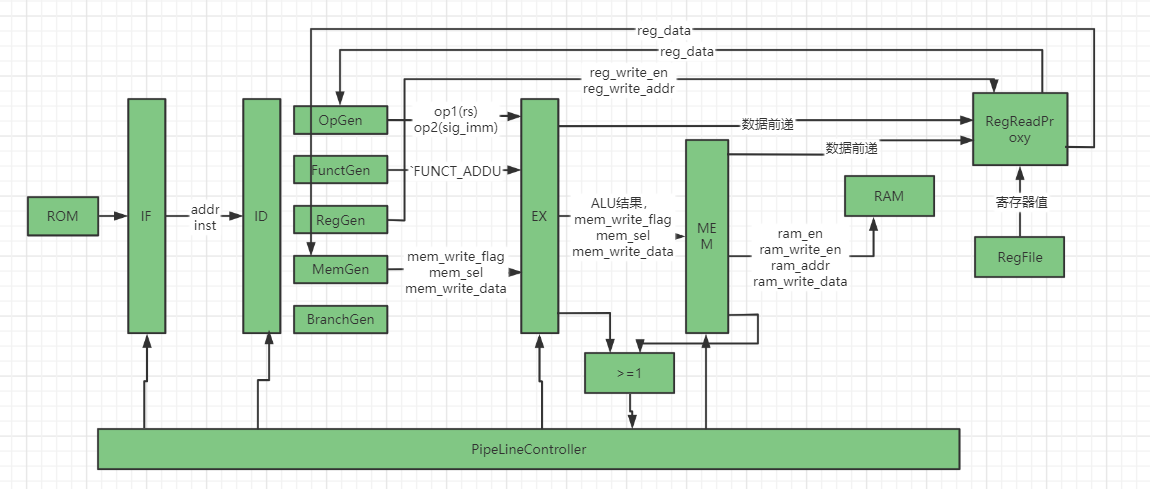


**2.3SH**

1）功能：将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是2的整数倍则触发地址错例外，否则据此虚地址将 rt 寄存器的低半字存入存储器中。2）执行过程：

取出SH指令。译码（Rt、rs读使能读地址，生成funct码，获取rs为操作数1，立即数符号扩展为操作数2。内存写使能，mem\_sel后两位为1，要写入内存的数据是rt）。执行（操作数1、2相加）。写回（ALU结果作为写地址，根据地址判断写入的半字是低/高位，写入的数据为rt）

3）数据通路图：



**3 代码实现**

**2.1 SLTI**

1）opcode.v

`define OP\_SLTI 6'b001010

2）RegGen.v(我只将相关代码合并列出)

reg\_read\_en\_1 <= 1;//一个源操作数

reg\_read\_en\_2 <= 0;//另一个操作数是imm，不用读寄存器

reg\_addr\_1 <= rs;//源操作数是rs

reg\_addr\_2 <= 0;

reg\_write\_en <= 1;//写使能

reg\_write\_addr <= rt;//目的操作数是rt

3）functGen.v

`OP\_SLTI: funct <= `FUNCT\_SLT;//生成funct码

4）operandGen.v，生成两个操作数(我只将相关代码合并列出)

operand\_1 <= reg\_data\_1;

operand\_2 <= sign\_ext\_imm;//符号扩展

1. EX.v

wire operand\_1\_lt\_operand\_2 = funct == `FUNCT\_SLT ? // 操作数1<2的标志

((operand\_1[31] && !operand\_2[31]) ||// op1负数 & op2正数

(!operand\_1[31] && !operand\_2[31] && result\_sum[31]) ||// op12 都正数, op1 - op2 负数

(operand\_1[31] && operand\_2[31] && result\_sum[31]))// op12都负数, op1 - op2负数

: (operand\_1 < operand\_2);

`FUNCT\_SLT: result <= {31'b0, operand\_1\_lt\_operand\_2};//比较结果

**2.2BGEZ**

1. opcode.v

`define OP\_BGEZ 6'b000001

2）RegGen.v(我只将相关代码合并列出)

reg\_read\_en\_1 <= 1;//源操作数1

reg\_read\_en\_2 <= 1;//源操作数2

reg\_addr\_1 <= rs;//源操作数是rs

reg\_addr\_2 <= rt;//源操作数是rt

3）BranchGen.v（只列出主要代码）

wire[`DATA\_BUS] sign\_ext\_imm\_sll2 = {{14{inst[15]}}, inst[15:0], 2'b00};//有符号的扩展立即数

if (inst[20:16] == 5'b00001 || inst[20:16] == 5'b10001) begin

if (reg\_data\_1[31] == 1'b0) begin

branch\_flag <= 1;//分支标志

branch\_addr <= addr\_plus\_4 + sign\_ext\_imm\_sll2;//分支地址

......

next\_inst\_delayslot\_flag <= 1;//延迟槽标志

**2.3 SH**

1）opcod.v

|  |
| --- |
| `define OP\_SH 6'b101001 |

2）RegGe.v

|  |
| --- |
| reg\_read\_en\_1 <= 1;//读使能1          reg\_read\_en\_2 <= 1;//读使能2          reg\_addr\_1 <= rs;//源操作数1是rs          reg\_addr\_2 <= rt;//源操作数2是rt |

3）FunctGen.v

|  |
| --- |
| `OP\_SH: funct <= `FUNCT\_ADDU;//生成funct码 |

4）OperandGen.v

|  |
| --- |
| wire[`DATA\_BUS] sign\_ext\_imm = {{16{imm[15]}}, imm};//符号扩展  operand\_1 <= reg\_data\_1;//操作数1是rs寄存器中的值  operand\_2 <= sign\_ext\_imm;//操作数2是符号位扩展后的立即数 |

5）MemGen.v

|  |
| --- |
| mem\_write\_flag <= 1;//内存写使能   mem\_sel <= 4'b0011;//后两位为1，说明数据在低位。   mem\_write\_data <= reg\_data\_2;//要写入内存的数据 |

6）MEM.v

|  |
| --- |
| if (mem\_sel\_in == 4'b0011) begin   //写信号          case (address[1:0])            2'b00: ram\_write\_sel <= 4'b0011;//00时写低位            2'b10: ram\_write\_sel <= 4'b1100;//10时写高位            default: ram\_write\_sel <= 4'b0000;          endcase        end   if (mem\_sel\_in == 4'b0011) begin//写数据          case (address[1:0])            2'b00: ram\_write\_data <= mem\_write\_data;//低位数据            2'b10: ram\_write\_data <= mem\_write\_data << 16;//高位数据          endcase        end |

**4 指令功能测试**（需有测试激励，仿真波形及分析，测试部分可将所有指令放在一起做。）

**4.1 SLTI**

addiu $10,$0,0x0008

slti $11,$10,0x0005

slti $11,$10,0x000b

addiu $12,$0,0x0009

slti $11,$12,0x8000



分析：

1. 立即数 8写入$10。
2. slti $11,$10,0x0003，$10的值8>立即数3，所以0写入$11。

（3）slti $11,$10,0x000a,$10的值8<立即数a，向$11写入1。

（4）向$12写入1。

（5） $11,$12,0x8000，其中0x8000符号扩展是负数，$12的值9>负数，所以向$12写入0。

**4.2 BGEZ**

addiu   $10,$0,0x0001

bgez    $10,zhq

nop

addiu    $10,$0,0x0008

zhq: addiu    $10,$0,0x0003



pc=bfc0008 执行延迟槽的nop。因为寄存器10的值大于等于0，所以执行之前的bgez，跳转到pc=bfc0014即zhq。

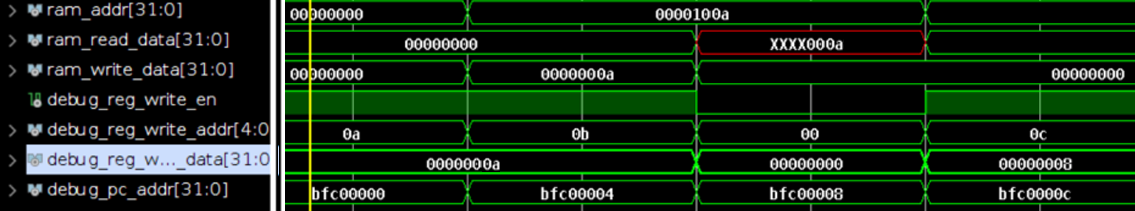
**4.3 SH**

addiu $10,$0,0x000a

addiu $11,$0,0x000a

sh $11,0x1000($10)

lh $12,0x1000($10)

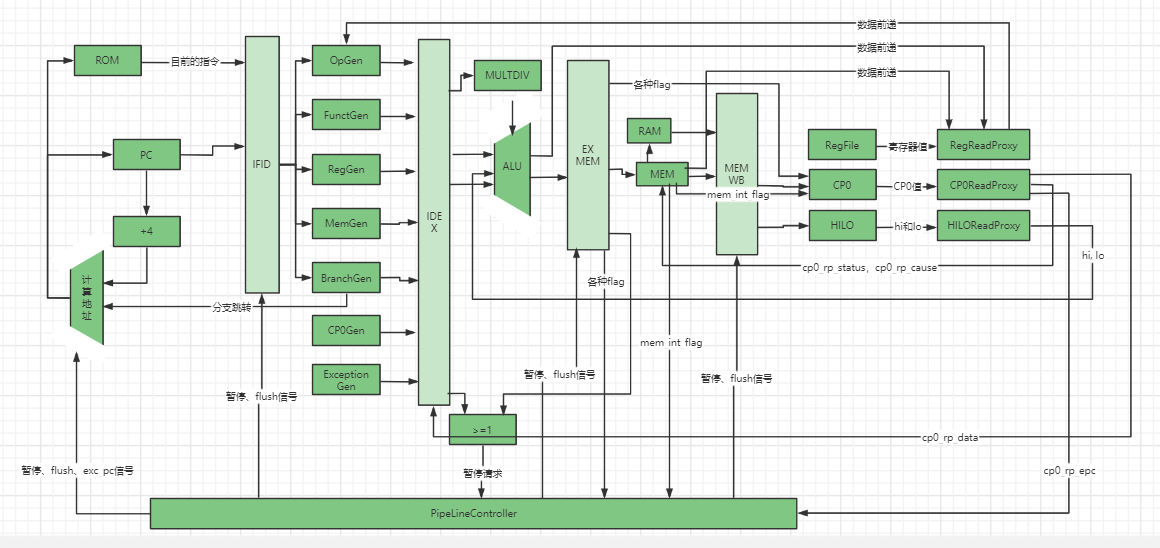


pc=0xbfc0000，向$10写入0x000a。pc=0xbfc0004，向$11写入0x000a。pc=0xbfc0008，将$11的内容（值a）写入0x100a地址的内存。pc=0xbfc000c，将0x100a地址的内存读入$11，看到值确实为a。

**Part 3 创新实验（演示系统介绍）实验分组编号：2（填写实验箱分组中组号）**

**系统概述：实现了什么功能，扩展了哪些内容，如何测试，有何发现。**

**1 系统总体架构图**



**2 设计过程**

**2.1HILO**

HILO相关指令有MFHI、MFLO、MTHI、MTLO，功能是在 WB 阶段写入 HILO 寄存器，或者在 EX 阶段读HILO 寄存器。

1）/core/storage 添加HILO.v。

2）采用数据前递解决RAW数据相关问题，/core/storage增加 HILOReadProxy.v，将mem、wb和原有HILO的值整合后传给EX。

3）典型读HILO指令：MFHI。在ex.v中读HILO寄存器：`FUNCT\_MFHI: result <= hi\_in;，

4）典型写HILO指令：MTLO。在ex.v中处理HILO写使能，HI和LO分别的写数据。

always @(posedge clk) begin

if(rst) begin

hi <= 0;

lo <= 0;

end else if (write\_en) begin

hi <= hi\_i;

lo <= lo\_i;

end

end

**2.2乘除法**

1. 使用IP核DivGen和MultGen进行乘除法运算。
2. 除法无法在一个周期内结束，所以除法没有计算完之前要在ex.v暂停流水。

stall\_request <= !mult\_div\_done;

1. 乘除结果写入HILO寄存器。在ex.v：

hilo\_write\_en <= 1;

hi\_out <= mult\_div\_result[63:32];

lo\_out <= mult\_div\_result[31: 0];

**2.3CP0**

1. 在storage增加CP0.v模块。包括5个寄存器：

reg[`DATA\_BUS] reg\_badvaddr，记录最近一次地址例外的虚地址

reg[ 32 : 0] reg\_count，处理器内部定时器

reg[`DATA\_BUS] reg\_status，记录处理器状态

reg[`DATA\_BUS] reg\_cause，记录最后一次例外原因

reg[`DATA\_BUS] reg\_epc，记录例外处理后开始执行的指令的 PC

1. 采用数据前递解决RAW数据相关问题，storage增加 CP0ReadProxy.v，将mem、wb和原有CP0的值整合后，将EPC值传给PipeLineController，DATA值传给IDEX，STATUS和CAUSE值传给MEM。

**2.4异常指令**

1. **在cp0def.v中定义了7个例外类型**

`define CP0\_EXCCODE\_INT 8'h00

`define CP0\_EXCCODE\_ADEL 8'h04

`define CP0\_EXCCODE\_ADES 8'h05

`define CP0\_EXCCODE\_SYS 8'h08

`define CP0\_EXCCODE\_BP 8'h09

`define CP0\_EXCCODE\_RI 8'h0a

`define CP0\_EXCCODE\_OV 8'h0c

1. **CP0中的处理（只列出了最主要的部分）**
2. CP0的Status寄存器的EXL位（即第1位）负责记录当前指令是否是处于例外状态。在CP0.v中，当有异常flag时EXL位置1，异常处理返回时置0。其中异常flag是否树立取决于当前是否正在处理其他异常。
3. 在CP0.v中计算EPC，即异常返回地址。如果发生例外的指令不在分支延迟槽，EPC等于发生例外的指令的 PC。否则EPC等于发生例外的指令的 PC-4

assign exc\_epc = delayslot\_flag ? current\_pc\_addr - 4 : current\_pc\_addr;

1. 如果发生例外的指令在延迟槽中，就让CP0的CAUSE寄存器的BD位为1。

reg\_cause[31] <= delayslot\_flag;

（4）在CP0.v中，寄存器reg\_cause的EXCCODE项根据各级产生并传递来的flag的优先级，记录当前的例外类型。（只列出有关代码）

if (int\_flag) begin reg\_cause[6:2] <= `CP0\_EXCCODE\_INT; end//INT例外

else if (adel\_flag && current\_pc\_addr[1:0] != 2'b00) begin

reg\_cause[6:2] <= adel\_flag ? `CP0\_EXCCODE\_ADEL : 0; end//ADEL例外

else if (ri\_flag) begin

reg\_cause[6:2] <= `CP0\_EXCCODE\_RI; end//RI例外

else if (break\_flag || syscall\_flag || overflow\_flag) begin

reg\_cause[6:2] <= break\_flag ? `CP0\_EXCCODE\_BP : syscall\_flag ? `CP0\_EXCCODE\_SYS :

overflow\_flag ? `CP0\_EXCCODE\_OV : 0; end//BP、SYS、OV例外

else if (adel\_flag || ades\_flag) begin//ADEL、ADES例外

reg\_cause[6:2]<=adel\_flag ? `CP0\_EXCCODE\_ADEL:ades\_flag ? `CP0\_EXCCODE\_ADES : 0;end

1. 如果有异常flag，会在PiplineControlloer里清空流水线。

assign flush = stall\_all ? 0 : (int\_flag || eret\_flag || syscall\_flag || break\_flag || overflow\_flag || adel\_flag || ades\_flag || ri\_flag) ? 1 : 0;

（5）异常地址跳转也在PiplineControlloer中实现：

always @(\*) begin

if (eret\_flag) begin exc\_pc <= cp0\_epc; end//异常返回地址

else if (int\_flag || syscall\_flag || break\_flag || overflow\_flag || adel\_flag || ades\_flag || ri\_flag) begin exc\_pc <= 32'hbfc0\_0380; end//异常处理地址

else begin exc\_pc <= 32'hbfc0\_0000; end end

1. **例外的实现**
2. **中断例外**
3. MEM.v

如果收到没被屏蔽的中断，就会触发中断例外。首先在MEM.v中判断int\_flag是否树立。首先判断STATUS的IE项全局中断允许，EXL项当前不在异常级，不是流水线暂停。如果成立，再根据STATUS的IM7-IM0是否屏蔽相应中断，和CAUSE的IP7-0是否对应外部中断发生，最终决定是否树立 int\_flag。

assign int\_flag = (cp0\_status\_in[0] && !cp0\_status\_in[1] && !stall\_all) ?

|(cp0\_cause\_in[15:8] & cp0\_status\_in[15:8]) : 0;

1. CP0中

首先置STATUS的EXL为异常级。

else if (int\_flag ......) begin reg\_status[1] <= 1;

CAUSE的BD位为记录例外指令是否在延迟槽，EXCCODE项记录例外类型为`CP0\_EXCCODE\_INT。

if (int\_flag) begin

reg\_cause[31] <= delayslot\_flag;

reg\_cause[6:2] <= `CP0\_EXCCODE\_INT;

EPC记录返回地址。

else if (int\_flag......) begin reg\_epc <= exc\_epc;

3）PiplineControlloer中清空流水线，并跳转到异常处理地址。

assign flush = stall\_all ? 0 : (int\_flag ......) ? 1 : 0;

exc\_pc <= 32'hbfc0\_0380;

**后续例外的CP0和PiplineControlloer基本类似，不再赘述**

1. **整数溢出例外**

ADD、ADDI、SUB 指令在EX溢出会引发整型溢出例外。

EX.v输出overflow\_flag

assign overflow\_flag\_out = (funct == `FUNCT\_ADD || funct == `FUNCT\_SUB) && (operand\_1[31] == operand\_2\_mux[31] && operand\_1[31] != result\_sum[31]);

1. **地址不对齐例外**

以adel\_flag为例：

如果是读半字，且访问地址不对齐半字边界，会触发例外。

assign adel\_flag\_out = load\_adel\_flag || pc\_adel\_flag;

if (mem\_read\_flag\_in) if (mem\_sel\_in == 4'b0011) if (result[0] != 1'b0) load\_adel\_flag <= 1;

如果是读字，且访问地址不对齐字边界，会触发例外。

if (mem\_read\_flag\_in) if (mem\_sel\_in == 4'b1111) if (result[1:0] != 2'b00) load\_adel\_flag <= 1;

取指 PC 不对齐于字边界，会触发例外。

if (current\_pc\_addr\_in[1:0] != 2'b00) begin pc\_adel\_flag <= 1;

ades\_flag是写不对齐，实现过程同理，但不用判断取值PC对齐。

1. **系统调用例外**

ID级的ExceptionGen.v通过指令译码判断是否有syscall例外。然后将flag一路传到EXMEM，然后传给CP0。

assign syscall\_flag = (op == `OP\_SPECIAL && funct == `FUNCT\_SYSCALL) ? 1 : 0;

1. **断点调用例外**

ID级的ExceptionGen.v通过指令译码判断是否有break例外。flag传递同上。

assign break\_flag = (op == `OP\_SPECIAL && funct == `FUNCT\_BREAK) ? 1 : 0;

1. **保留指令例外**

如果`OP\_SPECIAL，但不是已定义的funct，或者根本不是已定义的op，就触发例外。

default: begin ri\_flag <= 1; end

**3 实验现象及分析**

1. **功能测试**

实现了87个功能点，测试结果如下：



**2）性能测试：**

因为我没有完成所有功能点，所以我对我组**其他同学**实现了89个功能点的CPU进行了多次性能测试。

首先，使用vivado2019.2打开mycpu.xpr ，导入cpu文件。然后更换IP核的coe文件并仿真。最后更换IP的coe文件为allbench对应的coe文件，生成bit流后上板。在板上依次进行10个实验。

|  |  |
| --- | --- |
|  |  |
| 我组10个功能测试照片 | 我组功能测试结果统计图 |

与其他组对比，发现没有进行优化时，各组结果都是如此。

此外，为了获取最高的性能得分，我们尝试对cpu\_clk 进行修改，使其为 myCPU 支持的最高频率。功能测试和性能测试都调用了xilinx 的PLL IP生成2个时钟：sys\_clk是外部计时器时钟，固定 100MHz。cpu\_clk是SoC 全局时钟，可以根据 CPU 频率调整。仿真时 cpu\_clk 固定为 91MHz。

**理论上，cpu\_clk频率越高，执行相同程序，cpu 外部计数器的周期数越少。如果不更改gs132的cpu\_clk频率，则计算出的测试成绩越高。**

**我们尝试调整cpu\_clk为50、51、52、53、54、55、60MHz进行上板性能测试**，发现cpu\_clk=50、51的时候，上板测试成绩都是0.923没有变。cpu\_clk>51时，发现有些测试程序在绿灯亮起后数码管仍然显示00000000，如下图：



而其他程序在绿灯亮起后数码管显示的cpu 外部计数器的周期数确实减少了。因此，我初步认为cpu上板时能够支持的最高cpu\_clk为51MHz。但结果正确性还有待考察。

**3）创新代码的学习**

在小组作业中，我没有写创新代码，但是对小组其他同学的创新代码都进行了学习与分析。

以基于汇编语言的SCAN-流水灯设计为例。

|  |  |
| --- | --- |
| kseg0\_kseg1: ;循环主体  jal toleft ;来到toleft函数  nop  kkk:  lui t4,0x100;100写到t4高位  addu s3,s3,t4;s3<-s3+t4  sw s3,0(s1);向NUM\_ADDR写七段数码管值，新值是原值高位+1  jal toright;来到toright  nop  jjj:  addiu s3,s3,1;s3<-s3+1  sw s3,0(s1);向NUM\_ADDR写s3内容（七段数码管值），新值是原值低位+1 | toleft:;向左移动  SLL t7,t7,1;led的值t7逻辑左移1位  xor t9,t8,t7;t9<-t8（即0xffff） xor t7  sw t9, 0(s7);向LED\_ADDR写t9内容  jal wait\_1s;等一秒  nop  lw a3,0(s4);读出BTN\_KEY\_ADDR，存到a3  LI (t5,0x8000)  beq a3,t5,toright;如果a3=t5就转移到toright  nop  bne t7,s8,toleft;如果t7不等于s8就转移回到toleft  nop  J kkk;转到主循环的kkk  nop |
| toright:  SRL t7,t7,1;led的值t7逻辑右移1位  xor t9,t8,t7;t9<-t8（即0xffff） xor t7  sw t9, 0(s7);向LED\_ADDR写t9内容  jal wait\_1s;等一秒  nop  lw a3,0(s4);读出BTN\_KEY\_ADDR，存到a3  LI (t5,0x1)  beq a3,t5,toleft;如果a3=t5就转移到toleft  nop  bne t7,s6,toright;如果t7不等于s6就转移  nop  J jjj;转到主循环的jjj  nop | 总结：  首先在主循环中，分别有数码管显示得分高位+1、低位+1、跳转到toleft、跳转到toright的4段代码。首先执行toleft。  在toleft中，首先把led左移一位并输出到LED的地址。然后等一秒。然后读键盘，如果按下相应按钮，就跳到toright。如果没按键盘，再判断如果没到边界就继续左移，如果到边界了就回到主循环，在主循环中先让数码管显示的得分+1，然后来到toright。  在toright中和toleft同理，不再赘述。 |

此外，也学习了其他队友设计的基于开源模块将LCD显示屏向SOC的封装、基于汇编语言的SCAN-流水灯设计、 一维冰壶游戏设计等代码。因为篇幅限制，不再叙述。

**四：结论（讨论）**

**1、结论**（实验总结）

在本学期计组课设中，首先，在写cpu指令和异常处理的过程中，我完成了87个功能点，没通过soft\_int和ft的测试，但小组其他有两位同学各自独立完成了全部89个功能点。

其次，我对完成了89个功能点的同学的cpu进行了性能测试，在cpu\_clk=50MHZ时得到性能分0.923，为了进一步测试cpu的最高频率，我尝试了50以上的6个频率，发现最高能正常完成性能测试的频率是51MHZ，但因为时间不足，这一结论的正确性还有待考察。

第三，我对两套游戏与功能设计提出了设计功能的意见，确保功能的有序进行，并在代码实现后认真阅读并分析了几位同学的创新代码、硬件逻辑等。

第四，答辩中我完成了PPT中内容的撰写。

**2、讨论**（问题归纳，课程建议等）

在实验中，我遇到的最大的问题是时间不够，因为每跑一次都很慢，所以需要写很长一段代码，认真检查错误后，再跑仿真。

第二个问题是soft\_int等一两个指令的实现比较难，我最终没有彻底实现。于是在队友实现了以后学习了队友的代码，对这些指令有了一点新的认识。

对于本课程，我认为设计非常合理，没有特别大的建议。只是希望如果可以提前点确定个人验收时候的规则就更好啦！

**北京科技大学实验报告**

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级：计192

姓名： 张涵棋 学号： 41924076 实验日期： 2021 年 12月 17日

**五、教师评审**

|  |  |
| --- | --- |
| **教师评语** | **实验成绩** |
| （虽然课设主要侧重于验证问题，但是建议各位老师从解决“工程技术问题”，特别是“复杂工程问题”的角度去评审学生课设过程及代码阅读报告，主要内容包括提出问题、分析问题、解决问题及验证问题。**要有较详细的评审意见**。）  签名：  日期： |  |