北京科技大学 计算机与通信工程学院

代码阅读报告

课程名称： 计算机组成原理课程设计

学生姓名： 张宝丰

专　　业： 计算机科学与技术

班　　级： 计 1703

学　　号： 41724081

指导教师： 张磊

报告成绩：

实验地点： 机电楼 320

实验时间： 2019 年 11 月 16 日----2019 年 12 月 15 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 1703

姓名：　张宝丰　　学号：41724081　　实验日期：2019 年　11 月　16 日

一、课设目的与要求

- 学会处理器的设计方法：单周期或多周期或流水线。
- 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。
- 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。

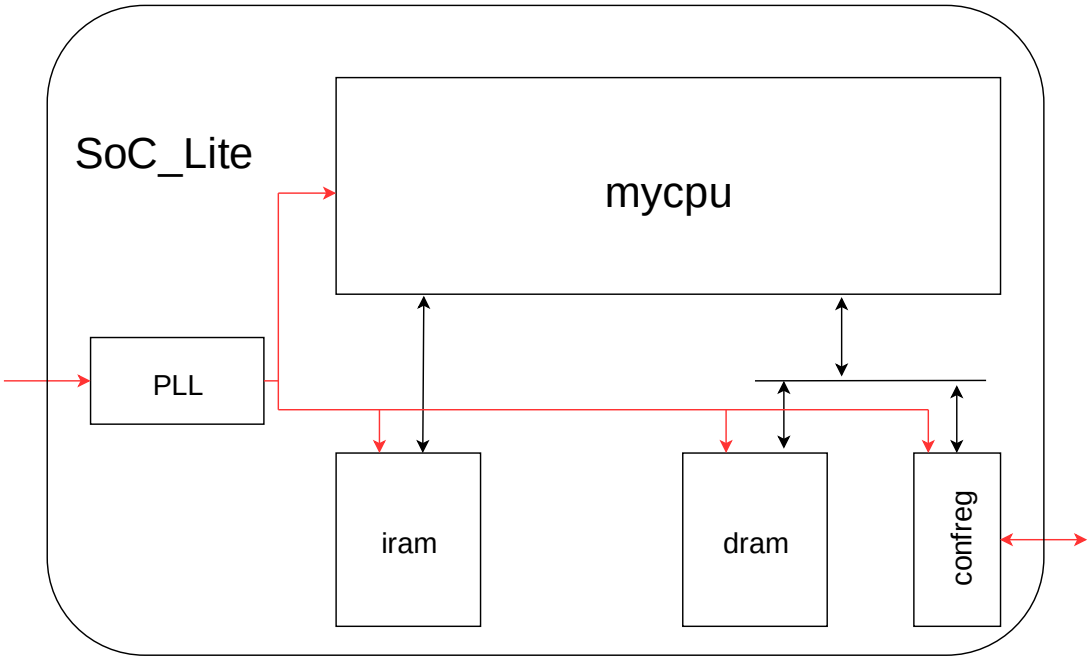
OS：Win10 64 位

Software：Vivado2018.3 开发工具

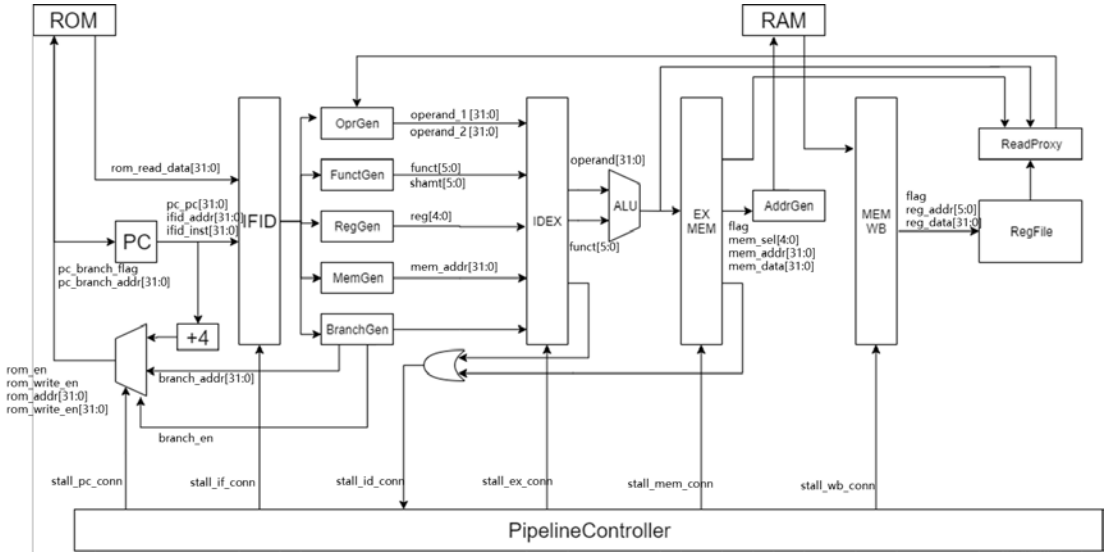
VirtualBox 虚拟机+Ubuntu16.04.6

三、设计过程与结果分析

1 TinyMIPS 总体结构框图（自己动手重新画图，推荐画图工具：gliffy 或 draw.io 或者其它）



2 数据通路图（自己动手重新画图，需要将每个信号的位数在图上有所显示）



3 跳转型指令（J）的设计过程

说明：此节及之后的所有指令的 6 位操作码均已在/define/opcode.v 中定义，所有指令的设计标准完全依照《“系统能力培养大赛 MIPS”指令系统规范》

3.1 指令格式

目标：无条件跳转，在距离当前地址 256MB 的范围内跳转。
操作：

```
I:
I+1: PC ← PC_GPRLEN-1..28 || instr_index || 0^2
```

3.2 实现代码

J 指令在 ID 级就完成对 PC 的修改，直接在 ID 级的`BranchGen.v`中添加下面的代码：

```
`OP_J: begin
    branch_flag <= 1;
    branch_addr <= {addr_plus_4[31:28], jump_addr, 2'b00};
end
```

3.3 仿真测试

仿真测试的步骤同 A07 说明，生成汇编代码时，我利用了 Uranus 项目中的 assembler，输入下面指令，直接在 windows 环境下根据 inst_rom.txt 中的汇编指令生成 inst_rom.bin：

```
python mips_asm.py inst_rom.txt
```

还可以把这条指令写成.bat 批处理文件，一键汇编。

针对 J 指令的测试汇编：

```
24 08 00 01     // addiu $8, $0, 1
25 08 00 02     // addiu $8, $8, 2
08 00 00 06     // j 0x6
00 00 00 00     // nop
```

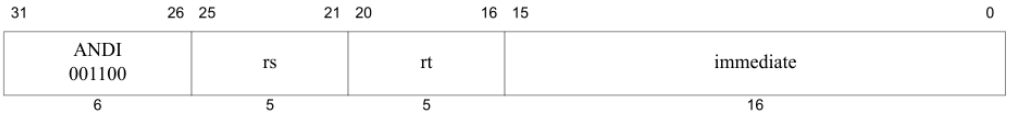
25 08 00 03　　// addiu $8, $8, 3
25 08 00 04　　// addiu $8, $8, 4
25 08 00 05　　// addiu $8, $8, 5
25 08 00 06　　// addiu $8, $8, 6

当 J 指令正常运行时，PC 应当在执行完 j 0x6 之后跳转到 addiu $8, $8 ,5 并继续执行，下面是仿真波形：

从红框中可以看到，PC 在 J 指令结束后，执行了一条 nop 指令，然后跳转到了 2508005 这条指令，正是 addiu $8, $8 ,5，说明 J 指令功能正确。

4 算数型指令（ANDI）的设计过程

4.1 指令格式



格式：ANDI rt, rs, immediate
目标：与立即数——和常数进行“逻辑与”的位运算
描述：GPR[rt] <- GPR[rs] and zero_extend(immediate)

4.2 实现代码

ANDI 指令需要读一个寄存器 rs，写一个寄存器 rt，据此修改/stage/id/RegGen.v

```
    // 生成寄存器读地址
    `OP_ADDIU begin
        reg_read_en_1 <= 1;
        reg_read_en_2 <= 0;
        reg_addr_1 <= rs;
        reg_addr_2 <= 0;
      end
…
    // 生成寄存器写地址
    `OP_ADDIU: begin
        reg_write_en <= 1;
        reg_write_addr <= rt;
      end
```

之后修改/stage/id/OperandGen.v。考虑 ANDI 指令的操作数，ANDI 的第一个操作数为 rs 寄存器的值，第二个操作数是零扩展的立即数，在给出的 CDE 中并没有实现零扩展，因此我首先实现零扩展的信号：

```
wire[`DATA_BUS] zero_ext_imm = {16'b0, imm};
```

接下来实现两个操作数的生成：

```
// 生成操作数 1
    `OP_ANDI: begin
        operand_1 <= reg_data_1;
      end
…
// 生成操作数 2
    `OP_ANDI: begin
        operand_2 <= zero_ext_imm;
      end
```

由于 ANDI 是 I 型指令，需要根据操作码给出功能码，在 id/FunctGen.v 中加入给出功能码的语句：

```
        `OP_ANDI: funct <= `FUNCT_AND;
```

下一步是在 ex/EX.v 中实现`FUNCT_AND 相应的运算操作，原始代码中已经实现如下：

```
        `FUNCT_AND: result <= operand_1 & operand_2;
```

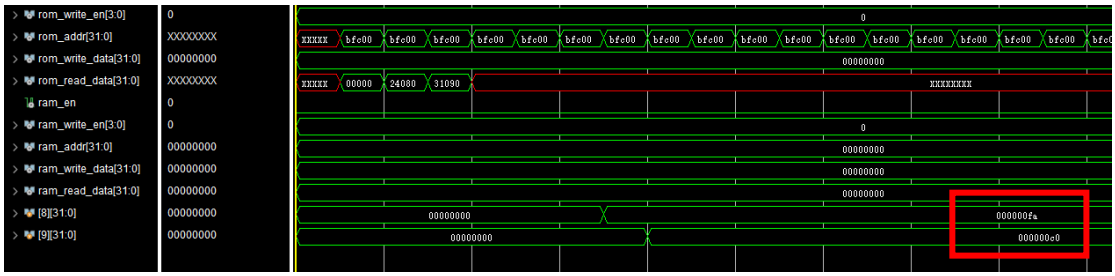
result 信号会在流水线上传递下去，一直到 WB 级写回 rt 寄存器，这样 ANDI 指令就扩展完了。

4.3 仿真测试

对于 andi 指令测试汇编：

```
    24 08 00 fa     // addiu $8,$0,0xFA
    31 09 00 c1     // andi   $9,$8,0xC1
```

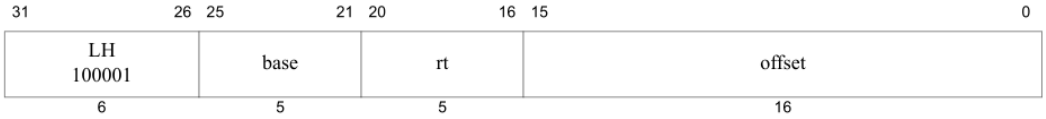
若指令功能正常，运行结束后，寄存器$9 的值应为 0xC0，下面是仿真波形：



可以看到，在正常的 4 周期延迟后（id, ex, mem, wb——4 级流水线），$9 正确地变为 0xc0，说明 andi 功能正确

5 访存型指令（LH）的设计过程

4.1 指令格式



格式：LH rt, offset(base)
目标：加载半字——从内存中加载半字，并作为一个有符号数处理
描述：GPR[rt] <- memory[GPR[base] + offset]
操作：

```
vAddr ← sign_extend(offset) + GPR[base]
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (CCA, HALFWORD, pAddr, vAddr, DATA)
byte ← vAddr_1..0 xor (BigEndianCPU || 0)
GPR[rt] ← sign_extend(memword_15+8*byte..8*byte)
```

4.2 实现代码

LH 指令涉及全部的 5 级流水线，我们首先考虑 ID 级。
LH 指令需要读取一个寄存器 base(rs)，将访存的结果写入一个寄存器 rt，据此修改 id/RegGen.v：

```
// generate read address
`OP_LH: begin
    reg_read_en_1 <= 1;
    reg_read_en_2 <= 0;
    reg_addr_1 <= rs;
    reg_addr_2 <= 0;
  end
…
// generate write address
  `OP_LH: begin
    reg_write_en <= 1;
    reg_write_addr <= rt;
  end
```

LH 要访问内存的地址是 signed_ext(offset) + GPR(base)，因此传递给 ex 级的两个操作数分别是 base 寄存器的值和符号扩展的立即数，据此修改 id/OperandGen.v：

```
// generate operand_1
`OP_LH: begin
    operand_1 <= reg_data_1;
```

```
        end
    …
    // generate operand_2
    `OP_LH: begin
            operand_2 <= sign_ext_imm;
        end
```

接下来是访存信号的生成。LH 指令需要读内存，故 mem_read_flag <= 1；LH 是加载半字，因此 mem_sel <= 4’b0011，据此修改 id/MemGen.v

```
…
`OP_LH: mem_read_flag <= 1;
…
`OP_LH: mem_sign_ext_flag <= 1;
…
`OP_LH: mem_sel <= 4'b0011;
…
```

LH 的两个操作数进行的是无符号加运算，功能码是`FUNCT_ADDU，据此修改 FunctGen.v：

```
`OP_LH: funct <= `FUNCT_ADDU;
```

接下来需要考虑的是访存级 MEM 的扩充，我在阅读源码时发现，MEM.v 只实现了字（word，4 个字节）和字节（Byte）的操作，并未实现半字（half-word）的操作，因此需要对 id/MEM.v 进行扩充如下：

```
    // generate ram_write_sel signal
    …
    else if(mem_sel_in == 4'b0011) begin    // half-word
        case (address[1:0])
            2'b00: ram_write_sel <= 4'b0011;
            2'b10: ram_write_sel <= 4'b1100;
            default: ram_write_sel <= 4'b0000;
        endcase
    end
    …

    // generate ram_write_data signal
    else if(mem_sel_in == 4'b0011) begin
        case (address[1:0])
            2'b00: ram_write_data <= mem_write_data;
            2'b10: ram_write_data <= mem_write_data << 16;
            default: ram_write_data <= 0;
        endcase
    end
```

类似地修改 WB 级的 WB.v，使之支持半字写回寄存器：

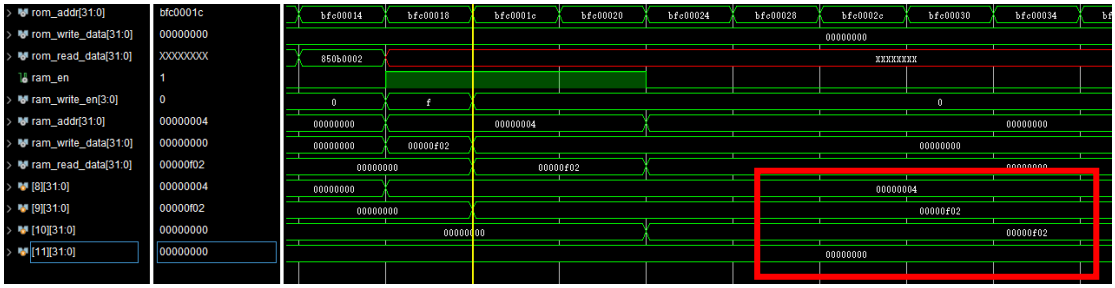
```
else if (mem_sel == 4'b0011) begin
    case(address[1:0])
        2'b00: result_out <= mem_sign_ext_flag ? {{16{ram_read_data[15]}}, ram_read_data[15:0]} : {16'b0, ram_read_data[15:0]};
        2'b10: result_out <= mem_sign_ext_flag ? {{16{ram_read_data[31]}}, ram_read_data[31:16]} : {16'b0, ram_read_data[31:16]};
        default: result_out <= 0;
      endcase
    end
```

4.3 仿真测试

仿真汇编代码：

```
24 08 00 04     // addiu $8,$0,0x4
24 09 0f 02     // addiu $9,$0,0x0F02
ad 09 00 00     // sw    $9,0($8)
85 0a 00 00     // lh    $10,0($8)
85 0b 00 02     // lh    $11,2($8)
```

仿真波形



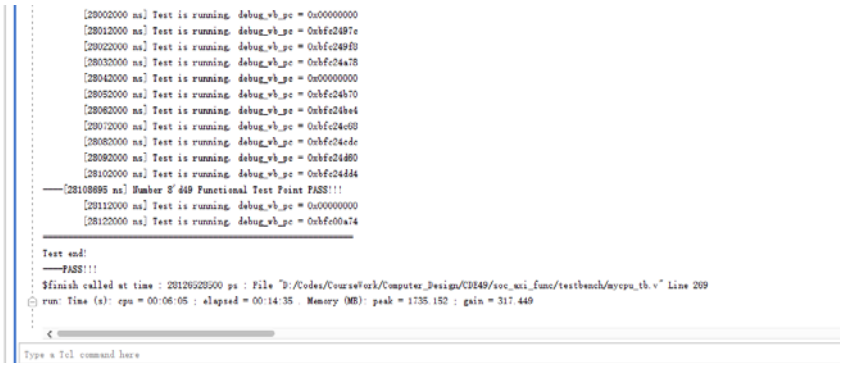
这里 LH 指令确实将 RAM 中的数据分别加载到了$10 和$11，但是却应用的是小端存储（低地址存储在低字节），MIPS 应当是大端存储，给出的 CDE 中的实现逻辑都是小端存储，我为了逻辑一致也实现的是小端存储。后经查阅参考书《自己动手写 CPU》，发现书中有关 MEM 和 WB 的 mem_sel 信号与给出的 CDE 恰好相反，这一点我认为应当注意。

四：结论（讨论）

1、结论

在这次计算机组成原理课程设计中，我结合指导书和参考书目，更加深入地理解了计算机组成原理，对 CPU 的构成有了更加深入地认识；同时结合给出的基本代码，扩展了 17 条 MIPS32 指令，大大提升了动手能力；此外，我还学习了使用 vivado 进行仿真测试的方法，为今后从事硬件设计相关工作打下了良好的基础。4 周的时间，可以说收获颇丰。

我扩展了考试范围内的所有 17 条指令，包括 SLTI、SLTIU、ANDI、NOR、ORI、XORI、SRA、SRL、BGEZ、BGTZ、BLEZ、BLTZ、J、JR、LH、LHU、SH，并通过了全部的仿真测试，代码已在 CG 平台提交，这里附上全部测试点通过的截图：



2、讨论

课程当中实现的 CPU 还只是一个非常基本的模型，只包含流水线、流水线暂停、数据前递、分支延迟槽的机制，异常、自陷等高级功能尚未实现，还有较为复杂的乘除法指令也没有实现，这些可以在今后继续动手实现。此外，我认为计组课设对于计科的同学们来说是一门极为重要的核心课程，对于同学们理解现代 CPU 构造有着莫大的帮助，如此重要的课程应当有更多的课时，并且提前进行，比如可以把这门课程提前到刚刚学完计组的大二暑假，这样既能及时应用知识，也能减轻同学们在大三上学期过重的实验压力。

五、附注

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：	