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： 康晟毓

专　　业： 计算机科学与技术

班　　级： 计 175

学　　号： 41724178

指导教师：

报告成绩：

实验地点： 机电楼

实验时间： 年　月　日----　　年　月　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：计科 175

姓名：康晟毓　　学号：41724178　　实验日期：　2019 年　11 月　1　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。

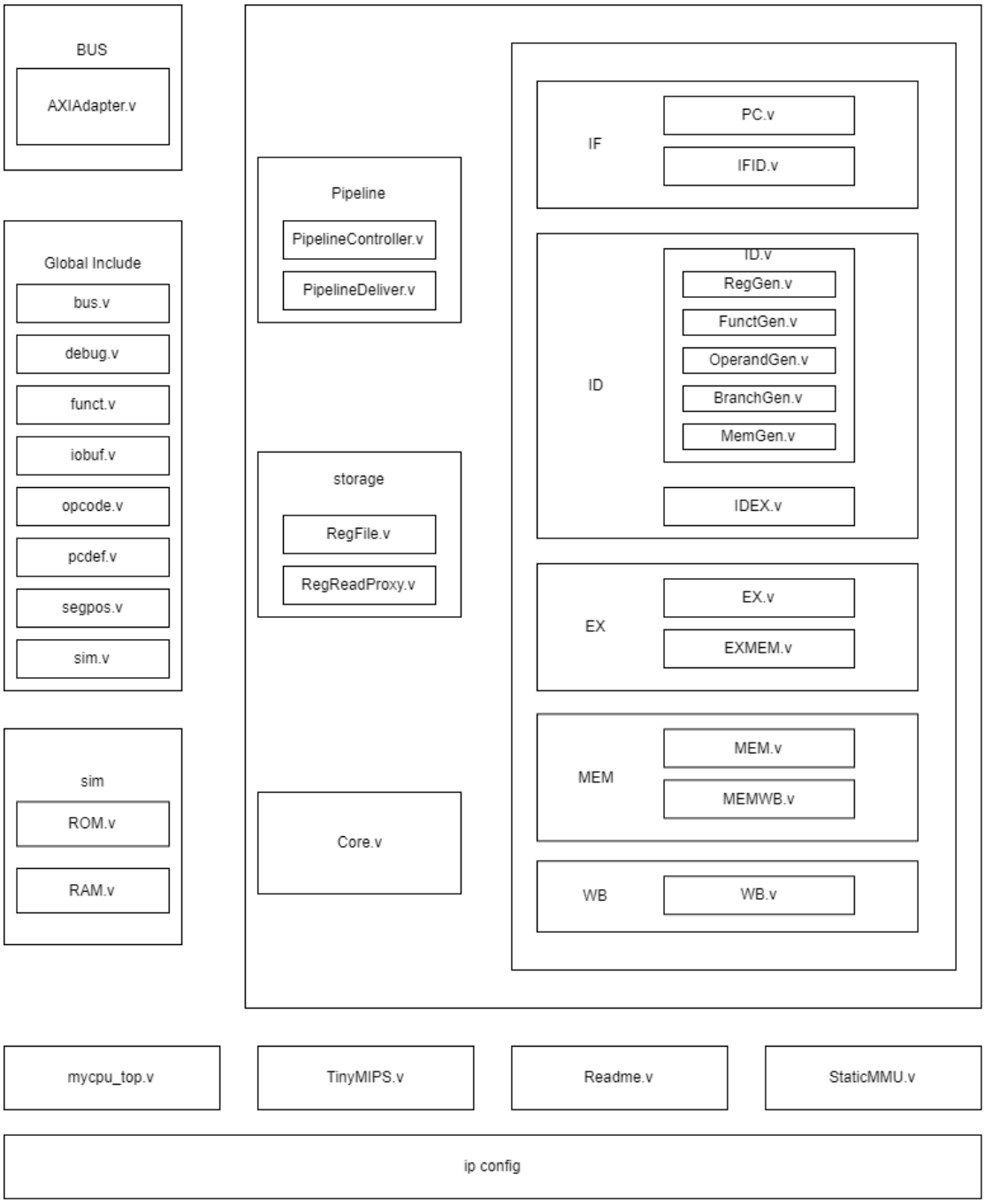
OS：Win10 64 位

Software：Vivado2018.3 开发工具

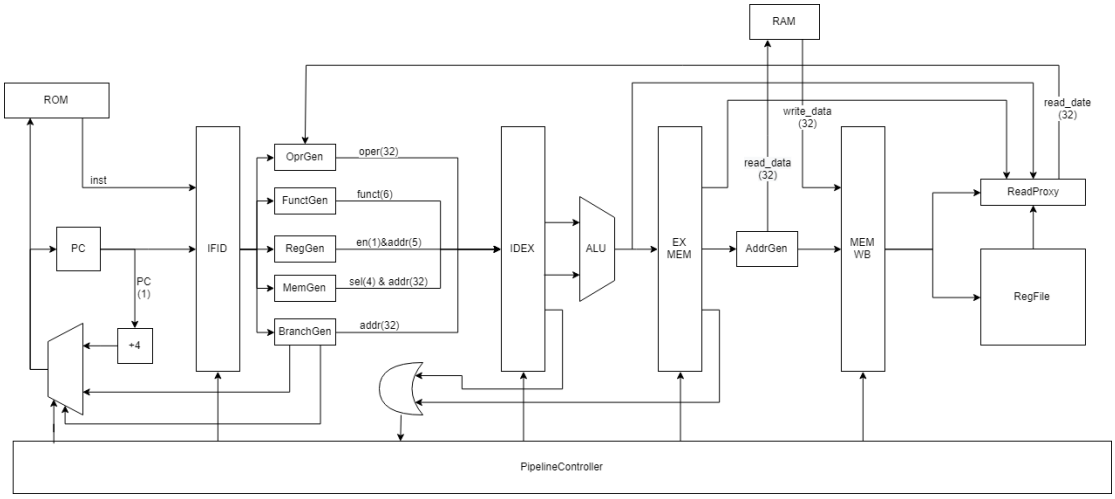
VirtualBox 虚拟机+Ubuntu16.04.6

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）



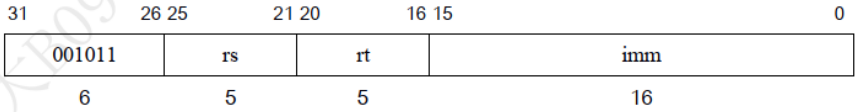
2 数据通路图（自己动手重新画图，需要将每个信号的位数在图上有所显示）



3 SLTIU 指令的设计过程

3.1 指令格式

3.3.10 SLTIU



汇编格式： SLTIU rt, rs, imm

功能描述： 将寄存器 rs 的值与**有符号扩展**至 32 位的立即数 imm 进行无符号数比较，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。

操作定义：
```
if (0||GPR[rs]31..0) < Sign_extend(imm) then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif
```

3.2 实现代码

因为 SLTIU 指令不是特殊指令，所以要在 opcode.v 中先设置指令对应的操作码。SLTIU 读取了 rs 和 imm 的内容，写入了 rt 的内容，所以需要在 RegGen.v 和 OperandGen.v 中进行设置。其中还用到了 SLTU 的操作，所以需要修改 FunctGen.v 对应的代码。

Opcode.v:

```
`define OP_SLTIU        6'b001011
```

RegGen.v:

```
20    always @(*) begin
21      case (op)
22        // arithmetic & logic (immediate)
23        `OP_ADDIU, `OP_SLTIU,
24        // memory accessing
25        `OP_LB, `OP_LW, `OP_LBU: begin
26          reg_read_en_1 <= 1;
27          reg_read_en_2 <= 0;
28          reg_addr_1 <= rs;
29          reg_addr_2 <= 0;
30        end
31        // branch
```

```
`OP_LB, `OP_LBU, `OP_LW, `OP_SLTIU: begin
  reg_write_en <= 1;
  reg_write_addr <= rt;
end
```

functGen.v:

```
`OP_SLTIU: funct <= `FUNCT_SLTU;
```

operandGen.v

```
always @(*) begin
  case (op)
    // immediate
    `OP_ADDIU, `OP_LUI, `OP_SLTIU,
    // memory accessing
    `OP_LB, `OP_LW, `OP_LBU, `OP_SB, `OP_SW: begin
      operand_1 <= reg_data_1;
    end
// arithmetic & logic (immediate)
`OP_ADDIU, `OP_SLTIU,
// memory accessing
`OP_LB, `OP_LW, `OP_LBU, `OP_SB, `OP_SW: begin
  operand_2 <= sign_ext_imm;
```

3.3 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）

```
42657    /home/nscscc/func/inst/n10_sltiu.S:10
42658    bfc262f8:   3c08c255    lui t0,0xc255
42659    bfc262fc:   2508e3d8    addiu   t0,t0,-7208
42660    bfc26300:   3c030000    lui v1,0x0
42661    bfc26304:   24630001    addiu   v1,v1,1
42662    bfc26308:   2d02f469    sltiu   v0,t0,-2967
42663    bfc2630c:   14430838    bne v0,v1,bfc283f0 <inst_error>
42664    bfc26310:   00000000    nop
```

程序功能：

T0 = C2550000H

T0 = T0 - 7208 = C254E3D8H

V1 = 0

V1 = v1 + 1 = 1

If (t0 < -2967)

　　V0 = 1

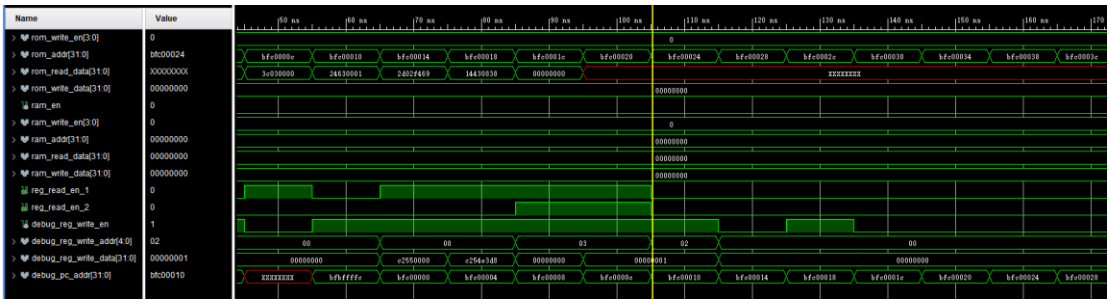
Else

　　V0 = 0

If (v0 != v1)

　　报错

测试波形

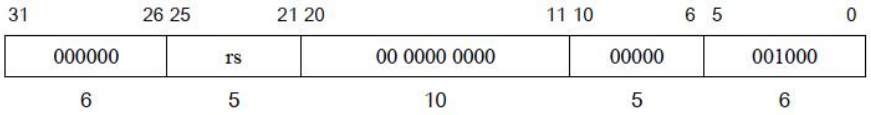


Pc 初值为 bfc262f8，因为 C254E3D8H < -2967，所以 xdata = 1，符合结果，说明指令设计正确。

4 JR 指令的设计过程

4.1 指令格式

3.6.11 JR



31	26 25	21 20	11 10	6 5	0
000000	rs	00 0000 0000	00000	001000	
6	5	10	5	6	

汇编格式： JR rs

功能描述： 无条件跳转。跳转目标为寄存器 rs 中的值。

操作定义： I: temp ← GPR[rs]

I+1: PC ← temp

例 外： 无

4.2 实现代码

JR 是特殊指令，所以需要在 funct.v 中设置它的功能码。JR 是跳转指令，所以需要在 BranchGen.v 中的 special 部分进行处理。因为 JR 是无条件跳转的，所以只要检测到是 JR 命令，就对跳转标志和跳转的地址进行设置。

Funct.v:

```
`define FUNCT_JR        6'b001000
```

BranchGen.v:

```
else if (funct == `FUNCT_JR) begin
    branch_flag <= 1;
    branch_addr <= reg_data_1;
end
```

4.3 仿真测试

```
79983    /home/nscscc/func/inst/n20_jr_ds.S:11
79984    bfc483ec:    3c15bfc5    lui s5,0xbfc5
79985    bfc483f0:    26b58408    addiu   s5,s5,-31736
79986    bfc483f4:    25168123    addiu   s6,t0,-32477
79987    bfc483f8:    02a00008    jr  s5
79988    bfc483fc:    25098123    addiu   t1,t0,-32477
79989    bfc48400:    100000c3    b   bfc48710 <inst_error>
79990    bfc48404:    00000000    nop
79991    bfc48408:    153600c1    bne t1,s6,bfc48710 <inst_error>
79992    bfc4840c:    00000000    nop
```

程序功能

S5 = BFC50000H

S5 = S5 – 31726 = BFC48408H

S6 = T0 -32477 = 800C8123H

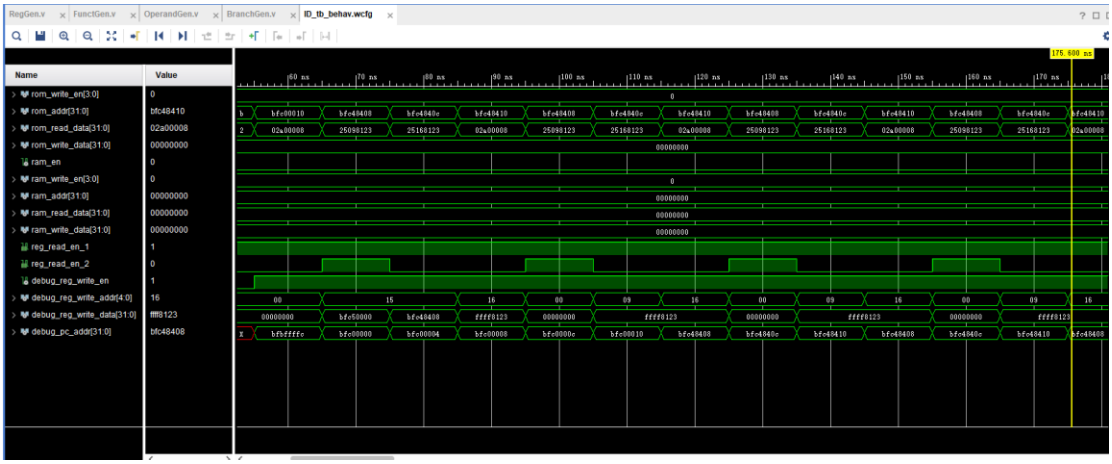
JR S5(BFC48408H)

T1 = T0 + (-32477) = 800C8123H

If (T1 != S6)

报错

测试波形



手动推算可得到，应当在 PC = BFC483F8H 时进行 JR 指令，跳转到 BFC48408H 处。由于延迟槽的存在，还需要执行 BFC483FCH 的指令，随后 PC 直接跳转为 BFC48408H，程序实现结果符合预期，说明设计正确

5 SH 指令的设计过程

5.1 指令格式

3.9.9 SH

31	26 25	21 20	16 15	0
101001	base	Rt	offset	
6	5	5	16	

汇编格式： SH rt, offset(base)

功能描述： 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。

操作定义：
vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_0 \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
datahalf ← $GPR[rt]_{15..0}$
StoreMemory(CCA, HALFWORD, datahalf, pAddr, vAddr, DATA)

例　外： 地址最低 1 位不为 0，触发地址错例外

5.2 实现代码

SH 指令是访存类指令，所以需要在 opcode.v 中设置操作码，对 operGen, regGen, MemGen, FucntGen 都进行修改。

Opcode.v

```
`define OP_SH          6'b101001
```

operateGen.v

```
`OP_LB, `OP_LW, `OP_LBU, `OP_SB, `OP_SW, `OP_SH, `OP_LH, `OP_LHU: begin
  operand_1 <= reg_data_1;
```

regGen.v

```
`OP_SH,
// r-type
`OP_SPECIAL: begin
  reg_read_en_1 <= 1;
  reg_read_en_2 <= 1;
  reg_addr_1 <= rs;
  reg_addr_2 <= rt;
```

MemGen.v

```
always @(*) begin
  case (op)
    `OP_SB, `OP_SW, `OP_SH: mem_write_flag <= 1;
    default: mem_write_flag <= 0;
  endcase
end
```

FunctGen.v

```
`OP_SLTIU: funct <= `FUNCT_SLTU;
`OP_LB, `OP_LBU, `OP_LW, `OP_LH, `OP_LHU, `OP_SH,
`OP_SB, `OP_SW, `OP_ADDIU: funct <= `FUNCT_ADDU;
`OP_ANDI: funct <= `FUNCT_AND;
```

因为 SH 是对半字操作，所以还需要修改 MEM 和 WB

MEM.v

```
if (mem_sel_in == 4'b0011) begin   // half
  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
```

WB.v

```
else if (mem_sel == 4'b0011) begin
  case(address[1:0])
    2'b00: result_out <= mem_sign_ext_flag ? {{24{ram_read_data[8]}}, ram_read_data[8:0]} : {24'b0, ram_read_data[8:0]};
    2'b10: result_out <= mem_sign_ext_flag ? {{24{ram_read_data[31]}}, ram_read_data[31:16]} : {24'b0, ram_read_data[31:16]};
  endcase
end
```

5.3 仿真测试

```
38582    bfc22660 <n64_sh_test>:
38583    /home/nscscc/func/inst/n64_sh.S:7
38584    bfc22660:   26100001    addiu   s0,s0,1
38585    /home/nscscc/func/inst/n64_sh.S:8
38586    bfc22664:   24120000    li  s2,0
38587    /home/nscscc/func/inst/n64_sh.S:10
38588    bfc22668:   3c0ae160    lui t2,0xe160
38589    bfc2266c:   254a8848    addiu   t2,t2,-30648
38590    bfc22670:   3c090e1a    lui t1,0xe1a
38591    bfc22674:   25293600    addiu   t1,t1,13824
38592    bfc22678:   3c08800d    lui t0,0x800d
38593    bfc2267c:   250852a4    addiu   t0,t0,21156
38594    bfc22680:   3c033601    lui v1,0x3601
38595    bfc22684:   24638848    addiu   v1,v1,-30648
38596    bfc22688:   ad0a3adc    sw  t2,15068(t0)
38597    bfc2268c:   a5093ade    sh  t1,15070(t0)
38598    bfc22690:   25040004    addiu   a0,t0,4
38599    bfc22694:   2505fffc    addiu   a1,t0,-4
38600    bfc22698:   ac843adc    sw  a0,15068(a0)
38601    bfc2269c:   aca53adc    sw  a1,15068(a1)
38602    bfc226a0:   8d023adc    lw  v0,15068(t0)
38603    bfc226a4:   8ca43adc    lw  a0,15068(a1)
38604    bfc226a8:   8c853adc    lw  a1,15068(a0)
38605    bfc226ac:   8ca63adc    lw  a2,15068(a1)
38606    bfc226b0:   144309dc    bne v0,v1,bfc24e24 <inst_error>
38607    bfc226b4:   00000000    nop
```

代码功能：

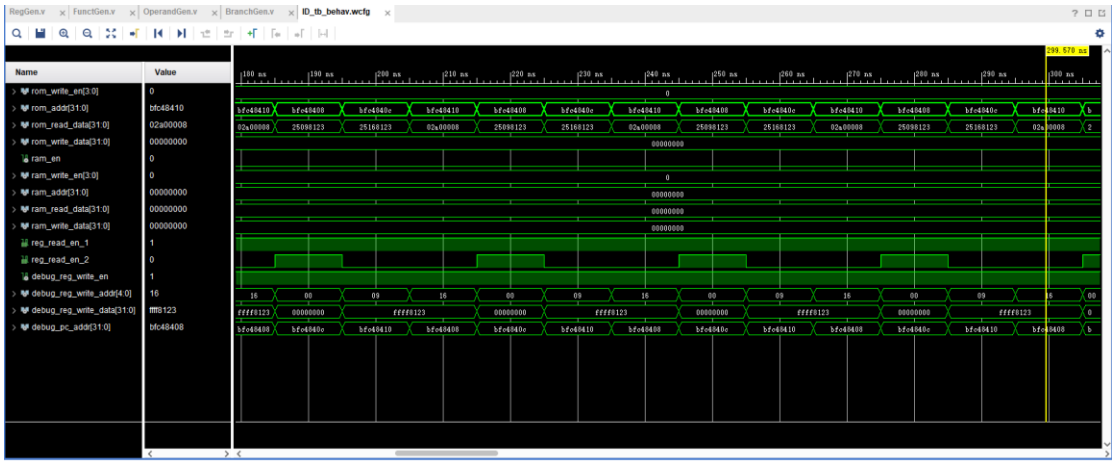
BFC22688H: [T0 + 15058H] <= T2

BFC2268CH: [T0 + 15070H] <= T1 低半字

求 a0 和 a1 的值

BFC226A0H: V0 <= [T0 + 15068H]

测试波形：



手推可以得到，内存对应的内容分辨如下。

15068: 48H

15069: 88H

15070: 00H

15071: 36H

执行 PC = BFC226A0H 的命令时，V0 应当为 36008848H，和波形显示的结果一样，符合预期，说明指令设计正确。

四：结论

1、结论

课设期间完成的主要工作：

跟随学长的讲解对 TinyMips 的实现过程。其中主要是汇编语言和流水线的关联等内容。其中课设期间完成的主要工作：跟随学长的讲解对 TinyMips 的实现过程。其中包含汇编语言和流水线的关联，如何实现流水线，流水线的实现细节，冲突、写后读等问题的处理。最后学长讲解了指令的拓展方法并让我们动手操作。

通过课程设计过程所取得的收获和能力的达成情况：

了解并实现了 TinyMips 中指令拓展的方法，熟悉了流水线的工作原理，掌握了 Verilog 语言和汇编语言的程序设计，对课上所学的理论知识有了更深刻的了解。同时增强了自己的动手能力和解决问题的能力，也让自己对 cpu 的理解达到了一个新的层次。

2、讨论

存在的问题及可能的改进方向：

无

五、教师评审

教师评语	实验成绩
签名： 日期：	