实验零—单周期 CPU 设计

姓名：汤尧 学号：3200106252 学院：求是学院云峰学园

课程名称：计算机系统II 同组学生姓名：无

实验时间：周四 3，4，5 节课 实验地点：紫金港机房 指导老师：卢立

一、 实验目的和要求

1. 回顾单周期 CPU 数据通路和控制器设计过程
2. 回顾程序执行过程
3. 为之后搭建流水线 CPU 打下基础

二、 实验内容和原理

2.1 实验内容

1. 完成 32 位单周期 CPU 数据通路设计
2. 完成单周期 CPU 控制模块设计，结合实验 10 数据通路，搭起 32 位单周期 CPU
3. 实现指令
4. 通过仿真测试和上板验收

2.2 设计模块

2.2.1 概述

单周期 CPU 主要特征是在一个周期内完成一条指令，也就意味着其 CPI(cycle per instruction) 是 1。考虑到不同的指令都需要在一个周期内完成，因而单周期 CPU 时钟频率是以执行最慢的那条指令作为基准，这也是单周期 CPU 最大的缺陷之一。

我们可以把单周期 CPU 分成数据通路和控制单元两个模块，本次实验将完成数据通路模块。

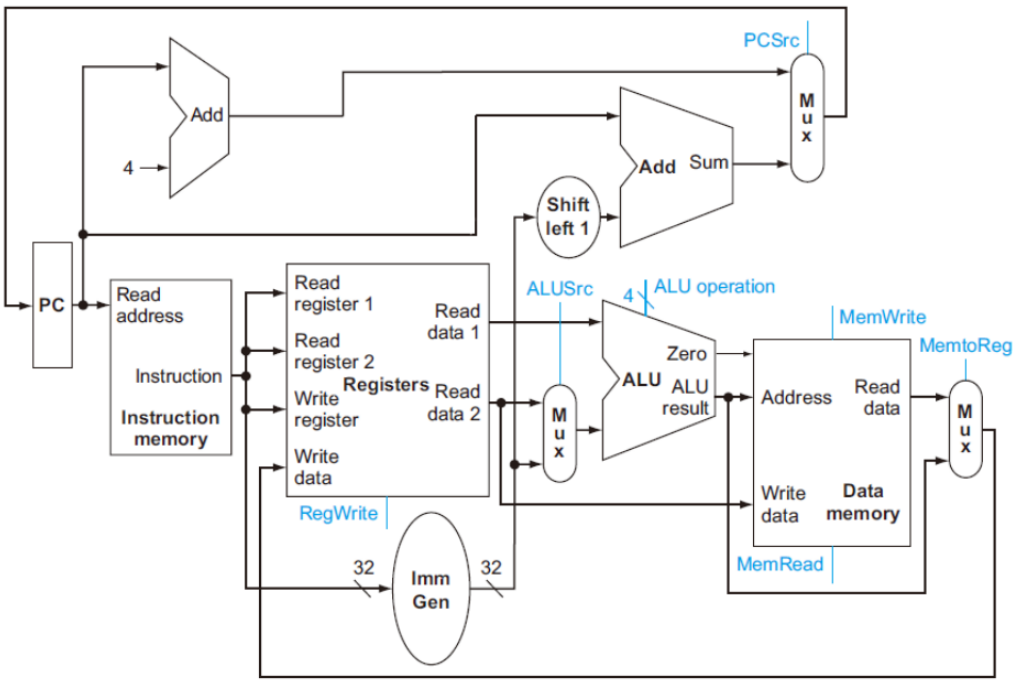
2.2.2 数据通路设计

为了之后能和流水线 CPU 进行衔接，我们把单周期 CPU 数据通路划分成 5 个阶段(stage):

1. instruction fetch (IF)
2. instruction decode (ID)
3. execution (EX)
4. memory access (MEM)
5. register write back (WB)

- 获取指令阶段 (IF)
  - 从 instruction memory 中获取 32 位指令
  - 令当前的 PC(program counter) 增加 4（PC = PC + 4，使当前 PC 指向下一条 32 位的指令，instruction memory 是 byte-addressing，所以是 +4 而不是 +32）
- 指令译码阶段 (ID)
  - 拿到指令之后，我们需要知道指令能提供各种信息包括 opcode, rs1, rs2, rd, imm 等等，所以需要对指令进行译码，提取信息。
  - 译码完成后，我们可以通过拿到的 rs1, rs2 读取相应的寄存器值
- 执行阶段 (EX)
  - 执行一些运算操作，包括 +，-，*，/，逻辑运算或左移、右移等等
  - 跳转地址的计算也可以在这个阶段执行
- 访存阶段 (MEM)
  - 从 data memory 中拿到数据
- 写回阶段 (WB)
  - 把计算后获得的数据或者从 data memory load 的数据写回到寄存器中

2.2.3 数据通路图



2.2.4 控制单元

控制单元，也称译码器，它的作用是解码指令，发出信号，告诉 Datapath 应该执行什么操

作。

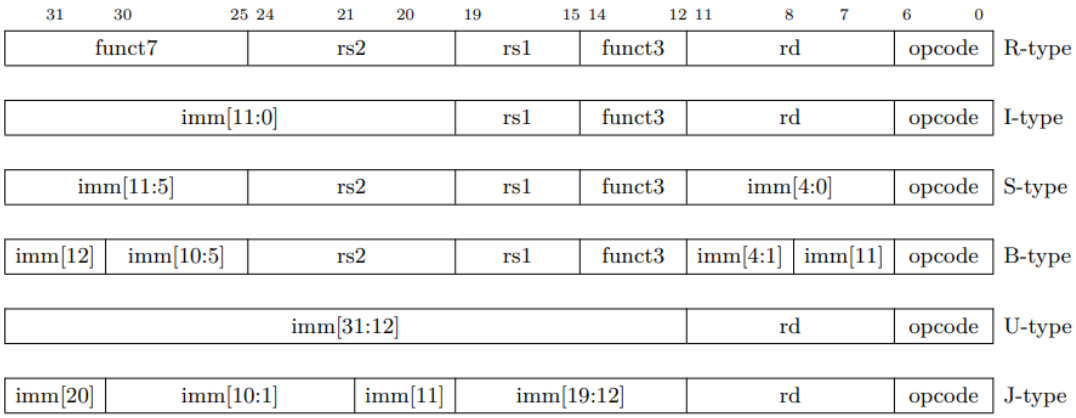
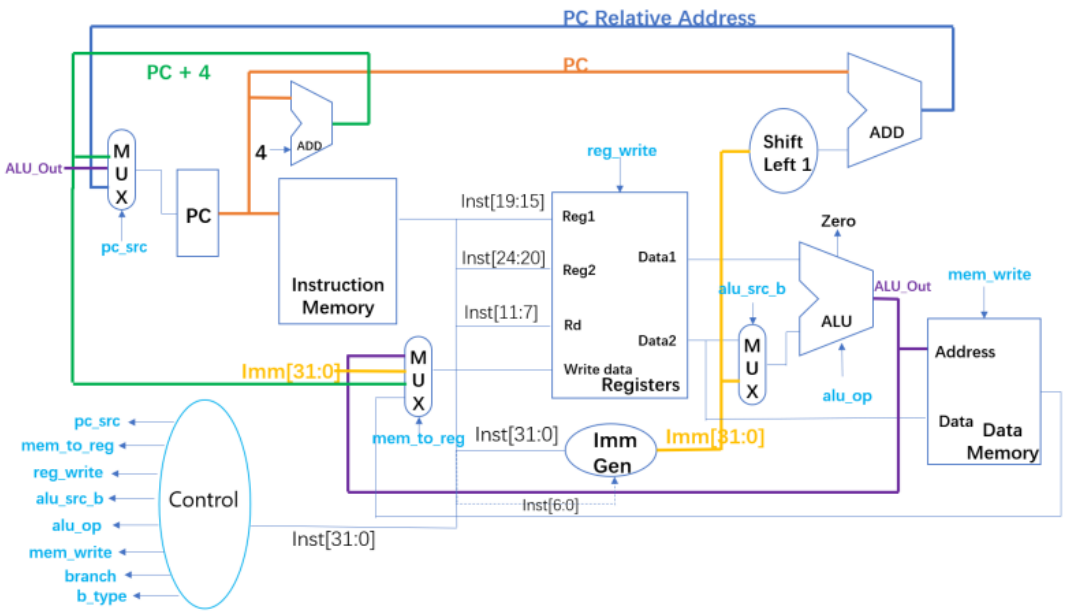


图 1: RISC-V base instruction formats

2.2.5 单周期 CPU 参考设计



2.2.6 实现指令

I 型： addi,slli,slti,sltiu,xori,srli,ori,andi,srai,lw,jalr

R 型：add, sll, slt, sltu, xor, srl, or, and, sra

U 型：lui, auipc

S 型：sw

B 型：beq, bne

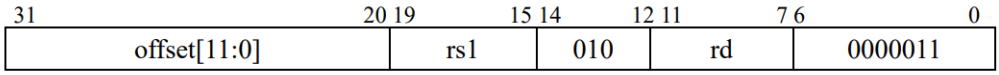
J 型：jal

lw rd, offset(rs1) x[rd] = sext(M[x[rs1] + sext(offset)][31:0])

字加载 *(Load Word)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取四个字节，写入 x[*rd*]。对于 RV64I，结果要进行符号位扩展。

压缩形式：**c.lwsp** rd, offset; **c.lw** rd, offset(rs1)



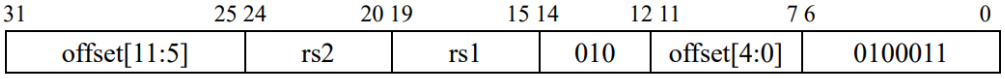
---

sw rs2, offset(rs1) M[x[rs1] + sext(offset) = x[rs2][31:0]

存字*(Store Word)*. S-type, RV32I and RV64I.

将 x[*rs2*]的低位 4 个字节存入内存地址 x[*rs1*]+*sign-extend(offset)*。

压缩形式：**c.swsp** rs2, offset; **c.sw** rs2, offset(rs1)



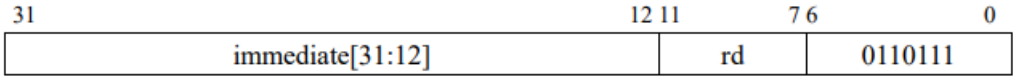
---

lui rd, immediate x[rd] = sext(immediate[31:12] << 12)

高位立即数加载 *(Load Upper Immediate)*. U-type, RV32I and RV64I.

将符号位扩展的 20 位立即数 *immediate* 左移 12 位，并将低 12 位置零，写入 x[*rd*]中。

压缩形式：**c.lui** rd, imm



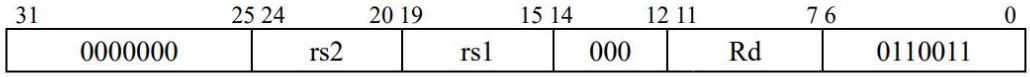
---

add rd, rs1, rs2 x[rd] = x[rs1] + x[rs2]

加 *(Add)*. R-type, RV32I and RV64I.

把寄存器 x[*rs2*]加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.add** rd, rs2; **c.mv** rd, rs2



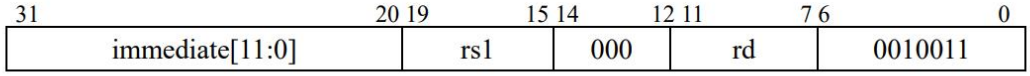
---

addi rd, rs1, immediate x[rd] = x[rs1] + sext(immediate)

加立即数*(Add Immediate)*. I-type, RV32I and RV64I.

把符号位扩展的立即数加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.li** rd, imm; **c.addi** rd, imm; **c.addi16sp** imm; **c.addi4spn** rd, imm

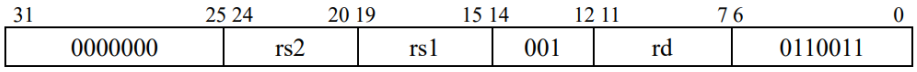


---

sll rd, rs1, rs2 $x[rd] = x[rs1] \ll x[rs2]$

逻辑左移(Shift Left Logical). R-type, RV32I and RV64I.

把寄存器 x[*rs1*]左移 x[*rs2*]位，空出的位置填入 0，结果写入 x[*rd*]。x[*rs2*]的低 5 位（如果是 RV64I 则是低 6 位）代表移动位数，其高位则被忽略。



31–25	24–20	19–15	14–12	11–7	6–0
0000000	rs2	rs1	001	rd	0110011

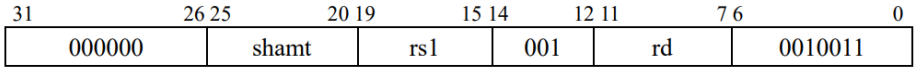
---

slli rd, rs1, shamt $x[rd] = x[rs1] \ll shamt$

立即数逻辑左移(Shift Left Logical Immediate). I-type, RV32I and RV64I.

把寄存器 x[*rs1*]左移 *shamt* 位，空出的位置填入 0，结果写入 x[*rd*]。对于 RV32I，仅当 *shamt*[5]=0 时，指令才是有效的。

压缩形式：**c.slli** rd, shamt



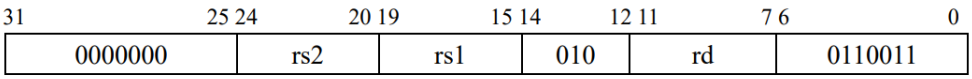
31–26	25–20	19–15	14–12	11–7	6–0
000000	shamt	rs1	001	rd	0010011

---

slt rd, rs1, rs2 $x[rd] = (x[rs1] <_s x[rs2])$

小于则置位(Set if Less Than). R-type, RV32I and RV64I.

比较 x[*rs1*]和 x[*rs2*]中的数，如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。



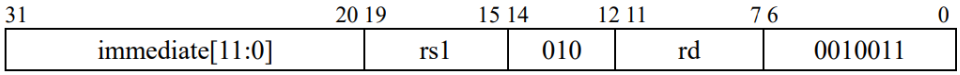
31–25	24–20	19–15	14–12	11–7	6–0
0000000	rs2	rs1	010	rd	0110011

---

slti rd, rs1, immediate $x[rd] = (x[rs1] <_s sext(immediate))$

小于立即数则置位(Set if Less Than Immediate). I-type, RV32I and RV64I.

比较 x[*rs1*]和有符号扩展的 *immediate*，如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。



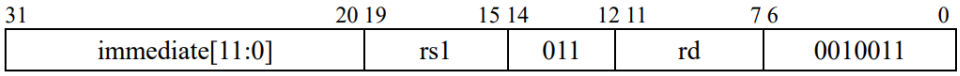
31–20	19–15	14–12	11–7	6–0
immediate[11:0]	rs1	010	rd	0010011

---

sltiu rd, rs1, immediate $x[rd] = (x[rs1] <_u sext(immediate))$

无符号小于立即数则置位(Set if Less Than Immediate, Unsigned). I-type, RV32I and RV64I.

比较 x[*rs1*]和有符号扩展的 *immediate*，比较时视为无符号数。如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。



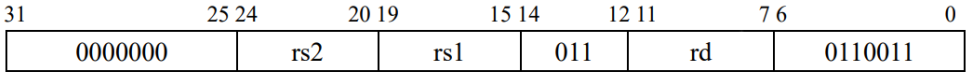
31–20	19–15	14–12	11–7	6–0
immediate[11:0]	rs1	011	rd	0010011

---

sltu rd, rs1, rs2 $x[rd] = (x[rs1] <_u x[rs2])$

无符号小于则置位(Set if Less Than, Unsigned). R-type, RV32I and RV64I.

比较 x[*rs1*]和 x[*rs2*]，比较时视为无符号数。如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。



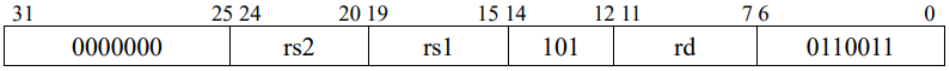
31–25	24–20	19–15	14–12	11–7	6–0
0000000	rs2	rs1	011	rd	0110011

---

srl rd, rs1, rs2 $x[rd] = (x[rs1] \gg_u x[rs2])$

逻辑右移(Shift Right Logical). R-type, RV32I and RV64I.

把寄存器 x[*rs1*]右移 x[*rs2*]位，空出的位置填入 0，结果写入 x[*rd*]。x[*rs2*]的低 5 位（如果是 RV64I 则是低 6 位）代表移动位数，其高位则被忽略。



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	101	rd	0110011

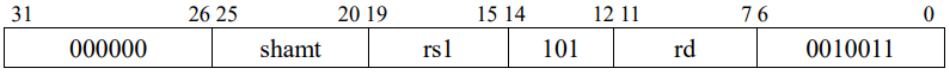
---

srli rd, rs1, shamt $x[rd] = (x[rs1] \gg_u shamt)$

立即数逻辑右移(Shift Right Logical Immediate). I-type, RV32I and RV64I.

把寄存器 x[*rs1*]右移 *shamt* 位，空出的位置填入 0，结果写入 x[*rd*]。对于 RV32I，仅当 *shamt*[5]=0 时，指令才是有效的。

压缩形式：**c.srli** rd, shamt



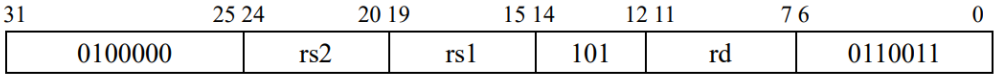
31 26	25 20	19 15	14 12	11 7	6 0
000000	shamt	rs1	101	rd	0010011

---

sra rd, rs1, rs2 $x[rd] = (x[rs1] \gg_s x[rs2])$

算术右移(Shift Right Arithmetic). R-type, RV32I and RV64I.

把寄存器 x[*rs1*]右移 x[*rs2*]位，空位用 x[*rs1*]的最高位填充，结果写入 x[*rd*]。x[*rs2*]的低 5 位（如果是 RV64I 则是低 6 位）为移动位数，高位则被忽略。



31 25	24 20	19 15	14 12	11 7	6 0
0100000	rs2	rs1	101	rd	0110011

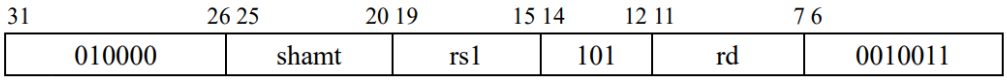
---

srai rd, rs1, shamt $x[rd] = (x[rs1] \gg_s shamt)$

立即数算术右移(Shift Right Arithmetic Immediate). I-type, RV32I and RV64I.

把寄存器 x[*rs1*]右移 *shamt* 位，空位用 x[*rs1*]的最高位填充，结果写入 x[*rd*]。对于 RV32I，仅当 *shamt*[5]=0 时指令有效。

压缩形式：**c.srai** rd, shamt



31 26	25 20	19 15	14 12	11 7	6 0
010000	shamt	rs1	101	rd	0010011

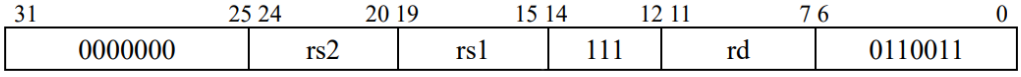
---

and rd, rs1, rs2 x[rd] = x[rs1] & x[rs2]

与 (And). R-type, RV32I and RV64I.

将寄存器 x[*rs1*]和寄存器 x[*rs2*]位与的结果写入 x[*rd*]。

压缩形式：**c.and** rd, rs2



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	111	rd	0110011

---

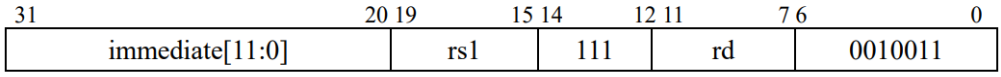
andi rd, rs1, immediate

x[rd] = x[rs1] & sext(immediate)

与立即数 *(And Immediate)*. I-type, RV32I and RV64I.

把符号位扩展的立即数和寄存器 x[*rs1*]上的值进行位与，结果写入 x[*rd*]。

压缩形式：**c.andi** rd, imm



---

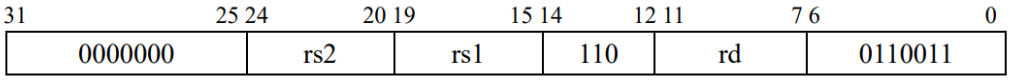
or rd, rs1, rs2

x[rd] = x[rs1] | $x[rs2]$

取或*(OR)*. R-type, RV32I and RV64I.

把寄存器 x[*rs1*]和寄存器 x[*rs2*]按位取或，结果写入 x[*rd*]。

压缩形式：**c.or** rd, rs2



---

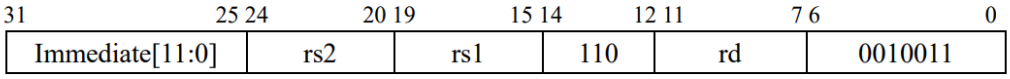
ori rd, rs1, immediate

x[rd] = x[rs1] | sext(immediate)

立即数取或*(OR Immediate)*. R-type, RV32I and RV64I.

把寄存器 x[*rs1*]和有符号扩展的立即数 *immediate* 按位取或，结果写入 x[*rd*]。

压缩形式：**c.or** rd, rs2



---

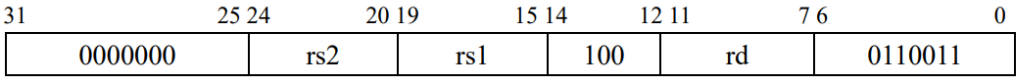
xor rd, rs1, rs2

x[rd] = x[rs1] ^ x[rs2]

异或*(Exclusive-OR)*. R-type, RV32I and RV64I.

x[*rs1*]和 x[*rs2*]按位异或，结果写入 x[*rd*]。

压缩形式：**c.xor** rd, rs2



---

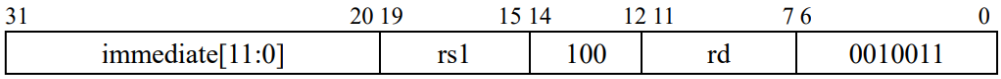
xori rd, rs1, immediate

x[rd] = x[rs1] ^ sext(immediate)

立即数异或*(Exclusive-OR Immediate)*. I-type, RV32I and RV64I.

x[*rs1*]和有符号扩展的 *immediate* 按位异或，结果写入 x[*rd*]。

压缩形式：**c.xor** rd, rs2



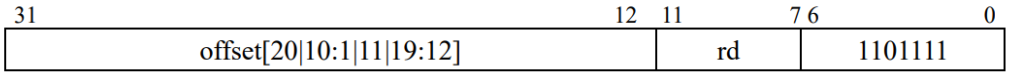
---

jal rd, offset　　x[rd] = pc+4; pc += sext(offset)

跳转并链接 *(Jump and Link)*. J-type, RV32I and RV64I.

把下一条指令的地址(*pc*+*4*)，然后把 *pc* 设置为当前值加上符号位扩展的 *offset*。*rd* 默认为 x1。

压缩形式：**c.j** offset; **c.jal** offset



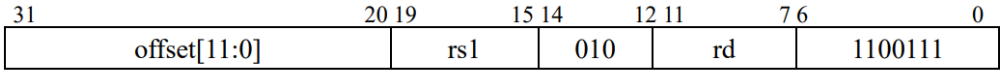
---

jalr rd, offset(rs1)　　t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t

跳转并寄存器链接 *(Jump and Link Register)*. I-type, RV32I and RV64I.

把 *pc* 设置为 x[*rs1*]+*sign-extend*(*offset*)，把计算出的地址的最低有效位设为 0，并将原 *pc*+*4* 的值写入 f[*rd*]。*rd* 默认为 x1。

压缩形式：**c.jr** rs1; **c.jalr** rs1



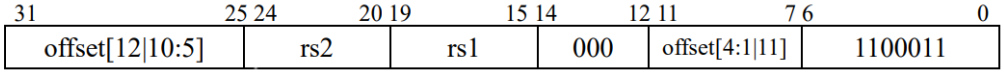
---

beq rs1, rs2, offset　　if (rs1 == rs2) pc += sext(offset)

相等时分支 *(Branch if Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

压缩形式：**c.beqz** rs1, offset



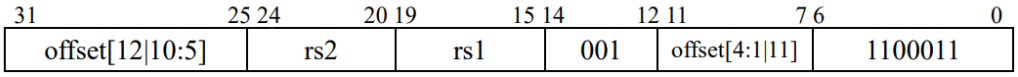
---

bne rs1, rs2, offset　　if (rs1 ≠ rs2) pc += sext(offset)

不相等时分支 *(Branch if Not Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值不相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

压缩形式：**c.bnez** rs1, offset

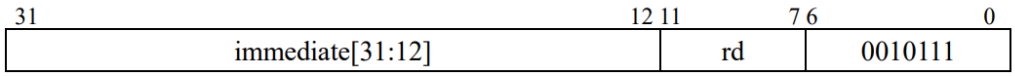


---

auipc rd, immediate　　x[rd] = pc + sext(immediate[31:12] << 12)

PC 加立即数 *(Add Upper Immediate to PC)*. U-type, RV32I and RV64I.

把符号位扩展的 20 位（左移 12 位）立即数加到 *pc* 上，结果写入 x[*rd*]。



三、 主要仪器设备

- HDL：Verilog、SystemVerilog
- IDE：Vivado
- 开发板：NEXYS A7

四、 操作方法与实验步骤

4.1 操作方法

完成单周期 CPU 设计，并将其通过 verilog 电路实现，Run Simulation 进行仿真并观察其波动情况。

4.2 实验步骤

4.2.1 数据通路设计

1. 根据 lab10-1 中给出的参考数据通路图完成设计
2. 根据提供的测试文件进行仿真测试
3. 提高提供的测试环境进行上板验证

4.2.2 控制模块设计

1. 完成控制单元模块设计
2. 结合数据通路，搭起单周期 CPU
3. 使用 lab10-1 提供的仿真代码文件进行仿真测试
4. 使用 lab10-1 提供的测试环境进行上板验证

五、 实验结果与分析

5.1 主要代码片段

entend 模块：根据指令解码，选择不同数字拓展类型。

```
always @(*)
    case (inst[6:0])
        7'b0010011: extend <= Immgen;//addi R
        7'b0110011: extend <= Immgen;//add
        7'b0000011: extend <= Immgen;//lw I
        7'b0100011: extend <= Smmgen;//sw S
        7'b1100011: extend <= Bmmgen;//beq B
        7'b1100111: extend <= Immgen;//jalr I
        7'b1101111: extend <= Jmmgen;//jal J
        7'b0110111: extend <= Ummgen;//lui U
        7'b0010111: extend <= Ummgen;//lui U
        default: extend = 0;
    endcase
```

pc 变化，每个时钟周期内，根据 rst 信号选择 pc 为 pc+4 或是初始地址。

```
50      always @(posedge clk or posedge rst) begin
51          if(rst) begin
52          pcc <= 32'b0;
53          addrr <= 32'h00000004;
54          end
55          else begin
56          pcc <= nextpcc;
57          addrr <= nextaddr;
58          end
59      end
```

多路选择器模块，根据 Control 模块传来的信号选择输出。

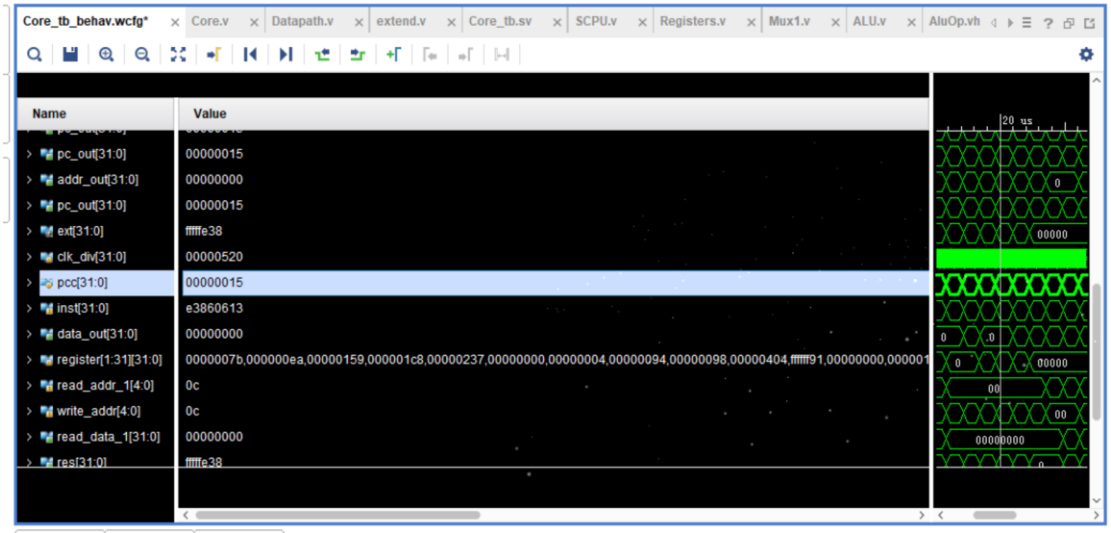
```
25  input b_byte,
26  input [31:0] I0,
27  input [31:0] I1,
28  input [31:0] I2,
29  input [31:0] I3,
30  input zero,
31   // 2'b00 表示写回rd的数据来自ALU, 2'b01表示数据来自imm,
32   //2'b10表示数据来自pc+4, 2'b11 表示数据来自data memory
33  output [31:0]o
34      );
35      reg [31:0] oo;
36      always @(*)
37      case(s)
38      2'b00: oo <= I0;
39      2'b01: oo <= I1;
40      2'b10: oo <= I2;
41      2'b11: if((b_byte & zero == 0)&(b_byte|zero == 1)) oo <= I3; else oo <= I0;
42      endcase
43      assign o=oo;
44  endmodule
```

control 模块，将指令解码，改变控制信号。

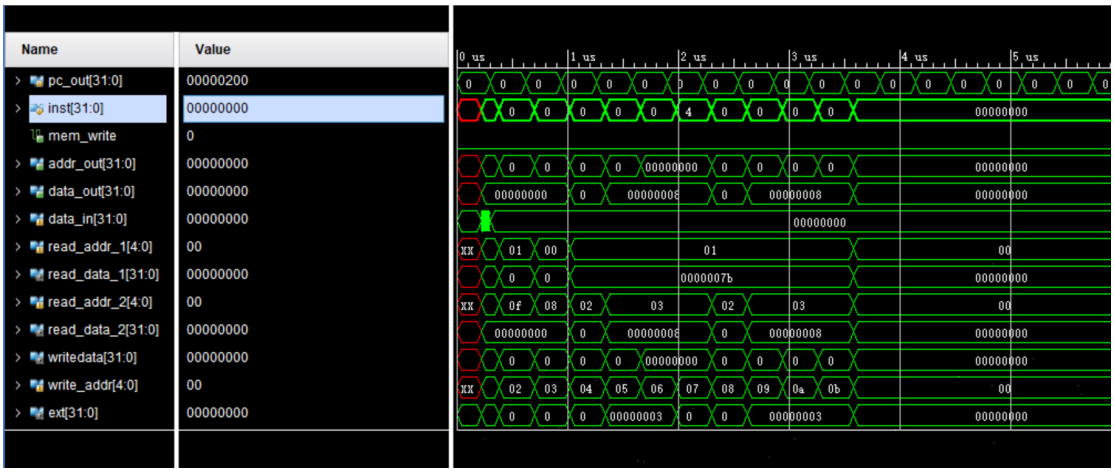
```
case (op_code)
7'b0110111: begin alu_src_b = 1'b1; mem_to_reg = 2'b00; end // LUI  U
7'b0010011: begin alu_src_b = 1'b1; pc_src = 2'b00; //addi,ori,slti,andi,xori,slli,sltu
    if(funct3 == 3'b000) alu_op = ADD;
    if(funct3 == 3'b001) alu_op = SLL;
    if(funct3 == 3'b010) alu_op = SLT;
    if(funct3 == 3'b011) alu_op = SLTU;
    if(funct3 == 3'b100) alu_op = XOR;
    if(funct3 == 3'b101) alu_op = SRL;
    if(funct3 == 3'b110) alu_op = OR;
    if(funct3 == 3'b111) alu_op = AND;
    end // ADDI R
7'b0110011: begin  alu_src_b = 1'b0; mem_to_reg = 2'b00;//add or and slt sll xor,sltiu
    if(funct3 == 3'b000) alu_op = ADD;
    if(funct3 == 3'b001) alu_op = SLL;
    if(funct3 == 3'b010) alu_op = SLT;
    if(funct3 == 3'b011) alu_op = SLTU;
    if(funct3 == 3'b100) alu_op = XOR;
    if(funct3 == 3'b101) alu_op = SRL;
    if(funct3 == 3'b110) alu_op = OR;
    if(funct3 == 3'b111) alu_op = AND;
    end
7'b0000011: begin pc_src = 2'b00; reg_write = 1'b1; mem_to_reg = 2'b11; end // LW   I
7'b0100011: begin pc_src = 2'b00; reg_write = 1'b0; mem_to_reg = 2'b00; end // SW   S
7'b0010111: begin pc_src = 2'b10; reg_write = 1'b1; alu_src_b = 1'b1; alu_op = ADD; end//auipc
7'b1100111: begin pc_src = 2'b10; reg_write = 1'b1; alu_src_b = 1'b1;  mem_to_reg = 2'b10;alu_op = ADD; end//jalr
7'b1101111: begin pc_src = 2'b10; alu_op = ADD; alu_src_b = 1'b1; mem_to_reg = 2'b10; branch = 1;end // JAL
7'b1100011: begin alu_op = SUB; pc_src = 2'b11; alu_src_b = 1'b0; mem_to_reg = 2'b10;
    if(funct3 == 000)  b_type = 0;//beq
    else if(funct3 == 001) b_type = 1;end // bne
```

5.2 仿真模拟

首先 rom 连接 lab_10.coe, lw,sw,addi,bne 指令成功后的波形变化。



自己编写的测试数据，验证 bonus 的指令。



上板验证，将 pc 当前指令地址，inst 当前指令值， ld/st addr 访问访存的地址，reg_write_data 寄存器写回值在四个 debug 输出信号中输出。和汇编指令对应验证 CPU 正常运行。