计算机设计与实践

单周期CPU设计与实现

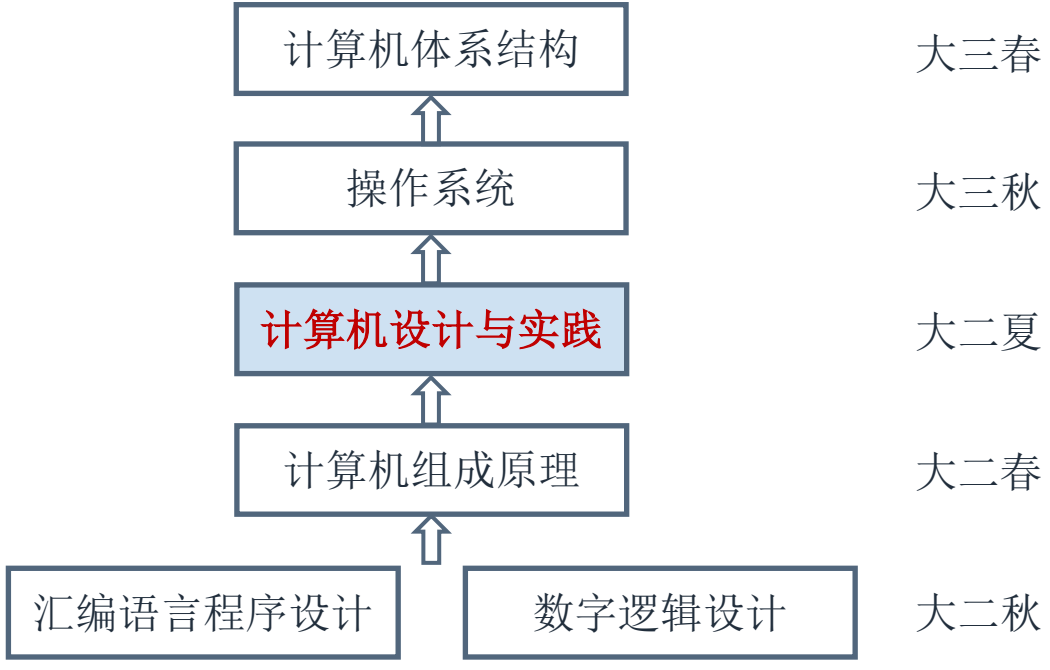
马世禹

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

目录

课程基本信息

- 开课学期：2夏 （2021夏）
- 课程学分：3.5
- 总学时：56学时 = 4学时理论 + 52学时实验
- 实验项目：24条指令的RISC-V CPU设计与实现



群名称:2021夏计算机设计与实践19...
群　号:273662926

课程目标

1. **综合**运用《数字逻辑设计》、《计算机组成原理》等基础课程知识，系统掌握计算机硬件系统设计与实现的**工程**方法；
2. 锻炼对计算机硬件系统的**分析**、**设计**和**创新**能力；
3. 通过计算机部件、CPU，再到SoC的逐步实现，锻炼**解决复杂工程**问题的能力;
4. 进一步掌握Verilog语言和EDA工具的使用，提升计算机硬件系统的**开发**、**调试**能力;

课程基本信息

- 总学时：56学时 = 4学时理论 + 52学时实验
- 实验项目：24条指令的RISC-V CPU设计与实现

	教学内容	学时
单周期CPU	单周期CPU设计理论	2
	RISC-V汇编语言程序设计	4
	单周期CPU设计与实现	20
流水线CPU	流水线CPU设计理论	2
	流水线CPU设计与实现	20
	外设I/O和课程总结	8

教学内容	学时
数据通路-取指/译码	4
数据通路-执行/访存	4
控制器	4
仿真	4
开发板实现	4

教学内容	学时
理想流水线	8
停顿机制	4
数据前推	4
开发板实现	4

从代码到实现

```
li a0, 5
ecall
mv t3, a0
# prints a newline
li a0, 11
li a1, '\n'
ecall
# prints msg2
li a0, 4
la a1, msg2
ecall
```

汇编器
RARS

```
0x00000597
0x07c58593
0x00000073
0x00500513
0x00000073
0x00050e13
0x00b00513
```

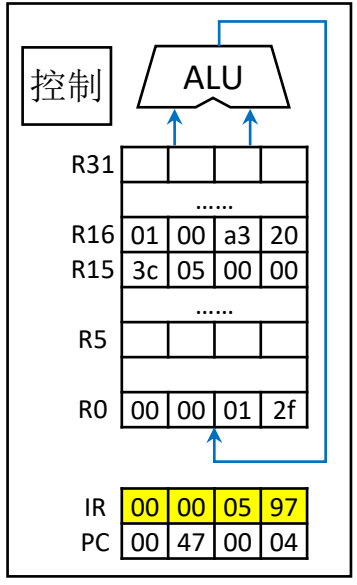
加载

主存

……			
97	05	00	00
93	85	c5	07
73	00	00	00
……			
13	05	b0	00
……			

执行

CPU



RISC-V汇编程序 → 机器码 → 指令

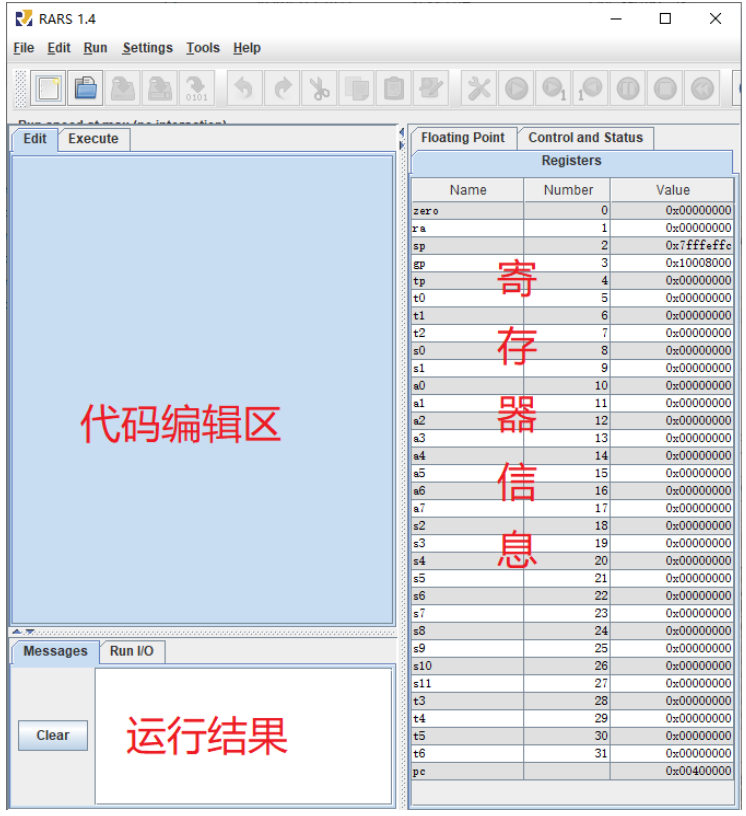
项目内容

1. 基于miniRV-1指令集，编写汇编程序，用RARS汇编器翻译成机器码，并在Logisim上运行；
2. 使用Verilog语言，分别设计单周期和流水线RISC-V CPU；
3. 在自己设计的CPU上，运行1.中的自编程序；
4. 基于Trace方法进行测试；
5. 添加拨码开关、LED、数码管的外设，形成SoC (System-On-Chip)；
6. 编写程序，对外设进行测试。

相关工具

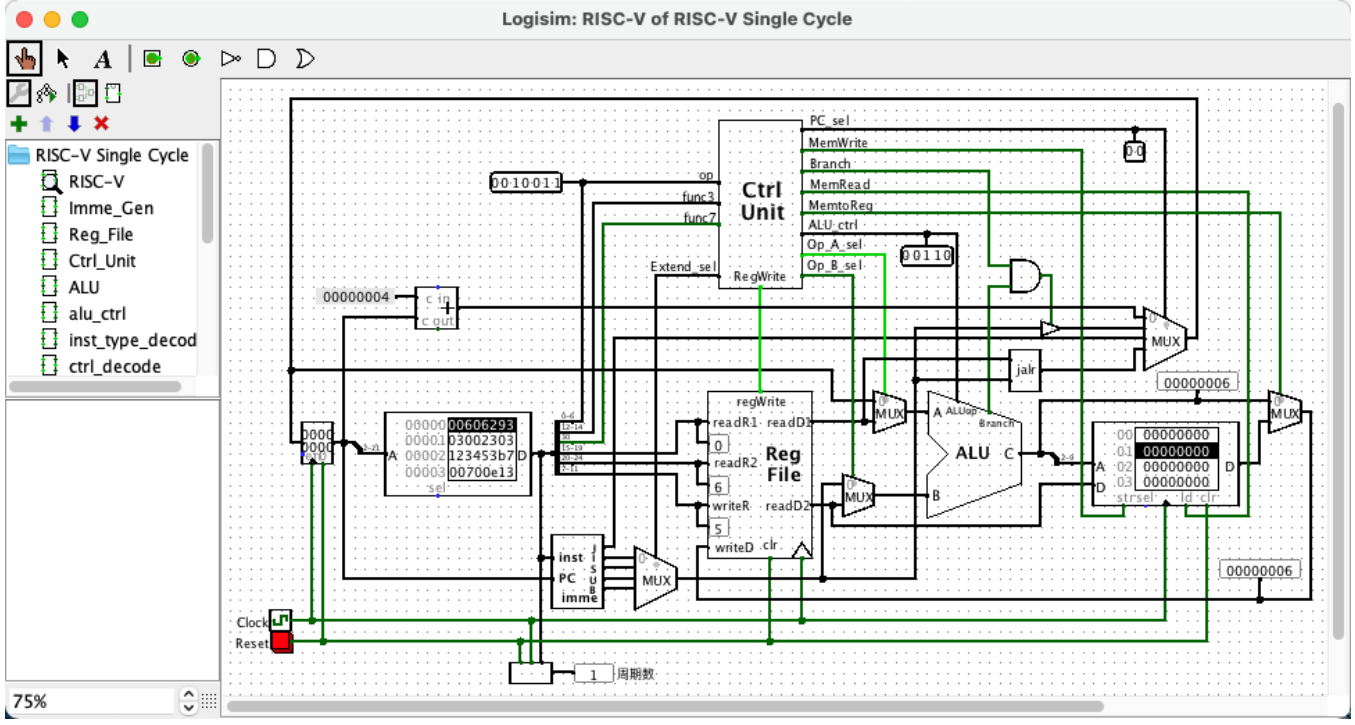
1. RARS

汇编程序IDE：编辑器+汇编器



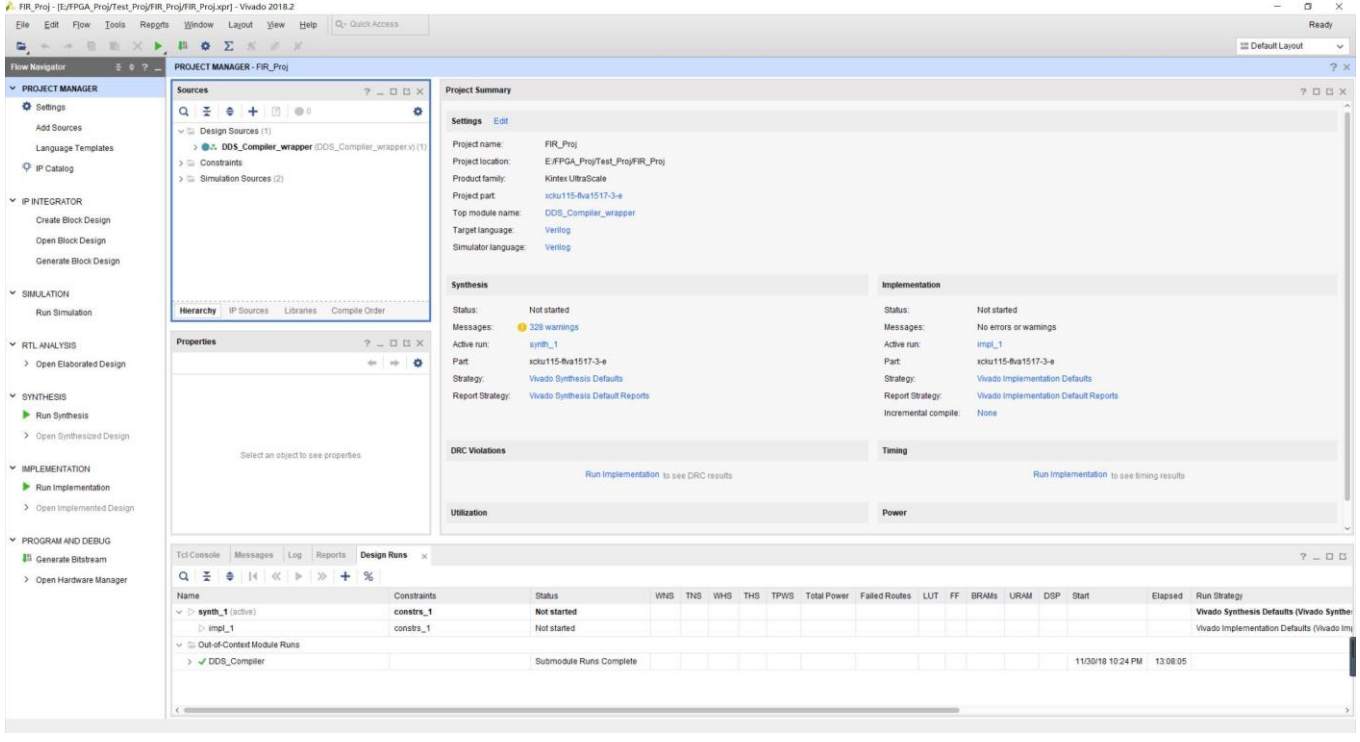
相关工具

2. Logisim



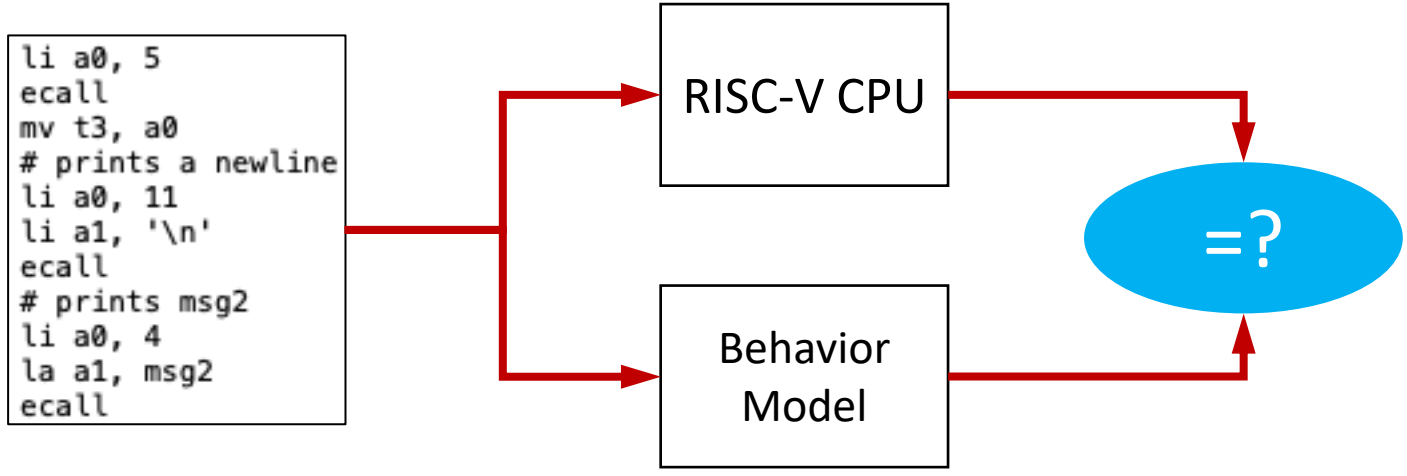
相关工具

3. Vivado



相关工具

4. Trace比对方法



评分标准：代码和报告(60%) + 验收(20%) + 作业(20%)

一、基础分：

中等：完成单周期和流水CPU的仿真、下载，正确运行要求的汇编程序

良好：在流水CPU的基础上，完成SoC的下载及正确运行要求的测试程序

优秀：在良好的基础上进行扩展或优化

扩展或优化内容包括（但不限于）：

1、编写汇编器；

2、优化电路逻辑表达式，在电路设计上优化系统性能（提高主频、降低功耗等）；

3、为其他外设（如键盘、UART串口等）设计接口电路，可以进行正确的IO操作；

4、设计实现分支预测、超标量、多核、乱序等进阶功能。

二、附加分：

按照每档要求的基础上完成要求以外的工作可获得附加分，如：编写自己的汇编程序，比较不同实现方案在资源使用、功耗、频率等性能上的差异…

参考资料

- 《计算机组成与设计: 硬件/软件接口 (RISC-V版)》
  —— David Patterson, John Hennessy

- riscv/riscv-cores-list:
  https://github.com/riscv/riscv-cores-list

RISC-V Cores and SoC Overview

This document captures the status of various cores and SoCs that endeavor to implement the RISC-V specification. Note that none of these cores/SoCs have passed the in-development RISC-V compliance suite.

Please add to the list and fix inaccuracies - see our CONTRIBUTING file for details.

Cores

Name	Supplier	Links	Capability	Priv. spec	User spec	Pri Lan
Avispado	SemiDynamics	Website	RV64	1.10	RV64GC, 2.2, multicore, V-ready	Syster
Atrevido	SemiDynamics	Website	RV64	1.10	RV64GC, 2.2, multicore, V-ready	Syster
RV32EC_P2	IQonIC Works	Website	RV32	1.11	RV32E[M]C/RV32I[M]C	Syster

目录

miniRV-1指令系统

- miniRV-1是32位整型RISC-V指令集（RV32I）的子集
  - 指令数：24条
  - 寄存器：32个32位通用寄存器
  - 指令长度：32位定长
  - 寻址方式：4种

miniRV-1寄存器组

寄存器	助记符	释义
x0	zero	常数0，永远只返回0（只读不可写）
x1	ra	存放调用子程序(函数)时的返回地址（Return Address）
x2	sp	堆栈指针。对它的调整必须显式地通过指令来实现，硬件不支持堆栈指针的调整（Stack Pointer）
x3	gp	全局指针。某些运行时系统用来为static或extern变量提供简单的访问方式（Global Pointer）
x4	tp	线程指针。多线程应用程序通过该寄存器来访问某个线程的本地数据（Thread Pointer）
x5~x7	t0~t2	存放临时变量，子程序(函数)使用时不保存这些寄存器的值，因此调用后它们的值会被破坏

miniRV-1寄存器组

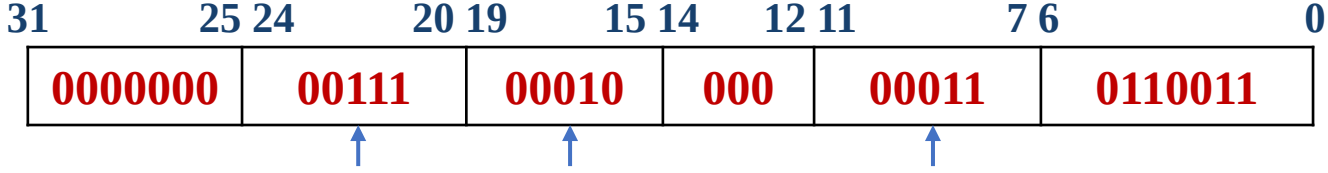
寄存器	助记符	释义
x8	s0 / fp	子程序用该寄存器做堆栈帧指针。子程序(函数)必须在返回前恢复这些寄存器的值以保证其没有变化 (Saved Register / Frame Pointer)
x9	s1	子程序用寄存器。子程序(函数)必须在返回前恢复这些寄存器的值以保证其没有变化 (Saved Register)
x10~x11	a0~a1	存放子程序(函数)调用时的整型参数或返回值
x12~x17	a2~a7	存放子程序(函数)调用时的整型参数
x18~x27	s2~s11	子程序用寄存器。子程序(函数)必须在返回前恢复这些寄存器的值以保证其没有变化 (Saved Register)
x28~x31	t3~t6	存放临时变量，子程序(函数)使用时不保存这些寄存器的值，因此调用后它们的值会被破坏

miniRV-1指令的寻址方式

➢ 寄存器寻址

操作数存放在寄存器中，指令里存放寄存器地址

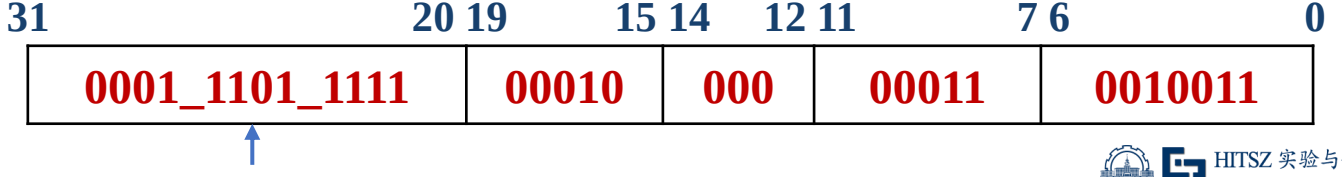
e.g. `add x3, x2, x7`



➢ 立即数寻址

指令中的操作数可以是12位的二进制立即数

e.g. `addi x3, x2, 0x1DF`

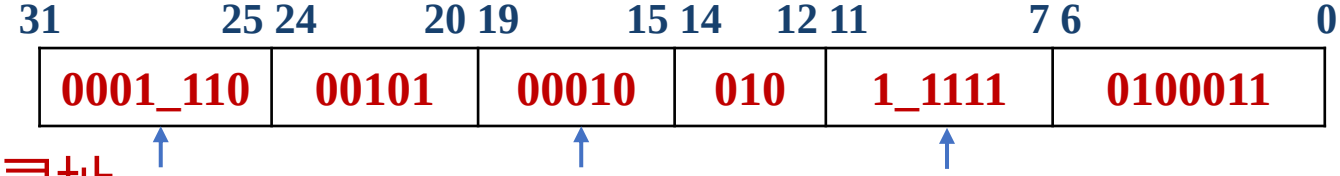


miniRV-1指令的寻址方式

➢ 变址寻址

操作数存放在数据存储器中，其有效地址由2部分组成：基地址存放在寄存器中，偏移地址则是一个12位的立即数

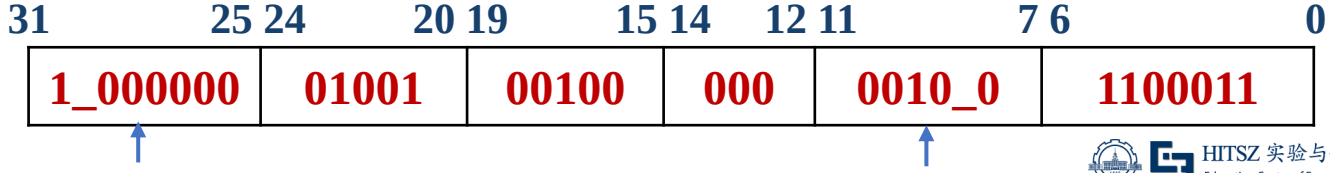
e.g. `sw x5, 0x1DF(x2)`



➢ PC相对寻址

操作数是当前的PC值加上经过符号扩展的12位偏移地址的值

e.g. `beq x4, x9, 0x802`

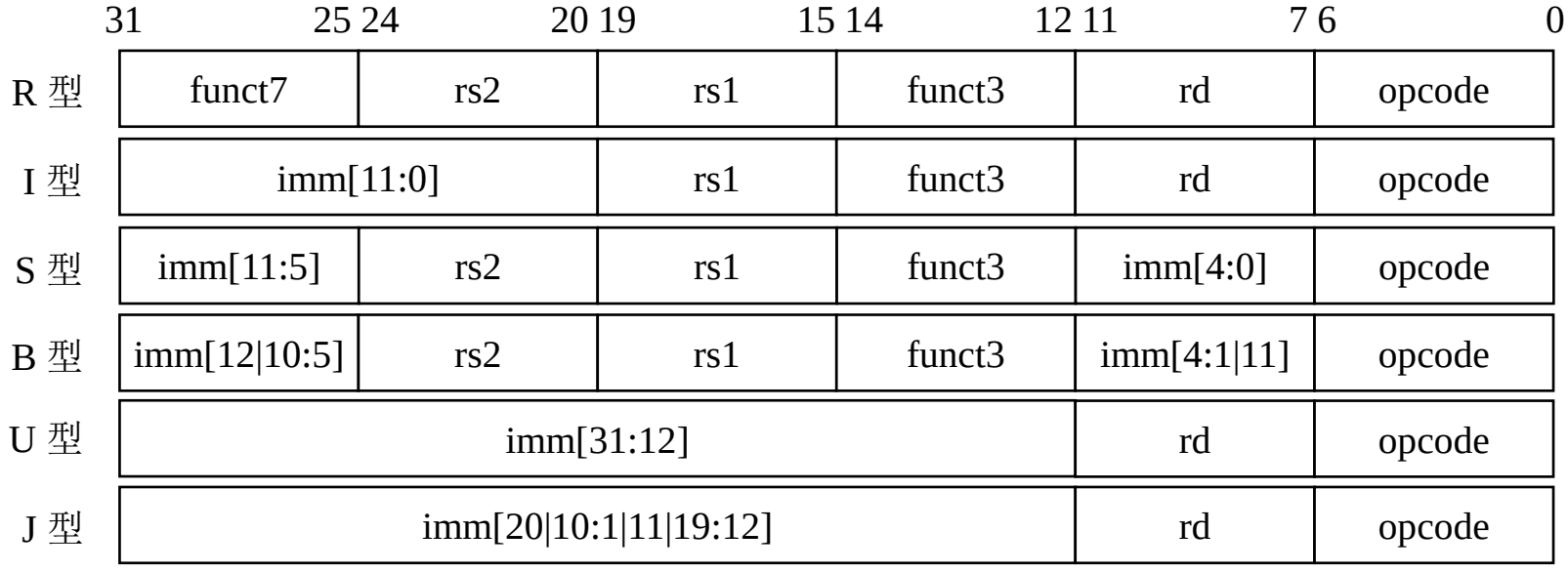


miniRV-1指令目录

- 算术运算指令 (3) —— add, addi, sub, lui
- 逻辑运算指令 (7) —— and, andi, or, ori, xor, xori
- 移位运算指令 (6) —— sll, slli, srl, srli, sra, srai
- 数据加载与存储指令 (2) ——lw, sw
- 条件转移指令 (4) —— beq, bne, blt, bge
- 无条件转移指令 (2) —— jal, jalr

miniRV-1指令格式

- 共6种类型的指令格式



miniRV-1指令格式 —— R型指令 (8)

助记符	指令格式						使用语法	操作及解释
BIT#	31..25	24..20	19..15	14..12	11..7	6..0		(r)表示寄存器r的值; Mem[addr]表示地址为addr的存储单元的值; sext(a)表示a的符号扩展
R-类型	funct7	rs2	rs1	funct3	rd	opcode		
add	0000000	rs2	rs1	000	rd	0110011	add rd, rs1, rs2	(rd) ← (rs1) + (rs2)
sub	0100000	rs2	rs1	000	rd	0110011	sub rd, rs1, rs2	(rd) ← (rs1) - (rs2)
and	0000000	rs2	rs1	111	rd	0110011	and rd, rs1, rs2	(rd) ← (rs1) & (rs2)
or	0000000	rs2	rs1	110	rd	0110011	or rd, rs1, rs2	(rd) ← (rs1) | (rs2)
xor	0000000	rs2	rs1	100	rd	0110011	xor rd, rs1, rs2	(rd) ← (rs1) ^ (rs2)
sll	0000000	rs2	rs1	001	rd	0110011	sll rd, rs1, rs2	(rd) ← (rs1) << (rs2)
srl	0000000	rs2	rs1	101	rd	0110011	srl rd, rs1, rs2	(rd) ← (rs1) >> (rs2), 逻辑右移
sra	0100000	rs2	rs1	101	rd	0110011	sra rd, rs1, rs2	(rd) ← (rs1) >> (rs2), 算术右移

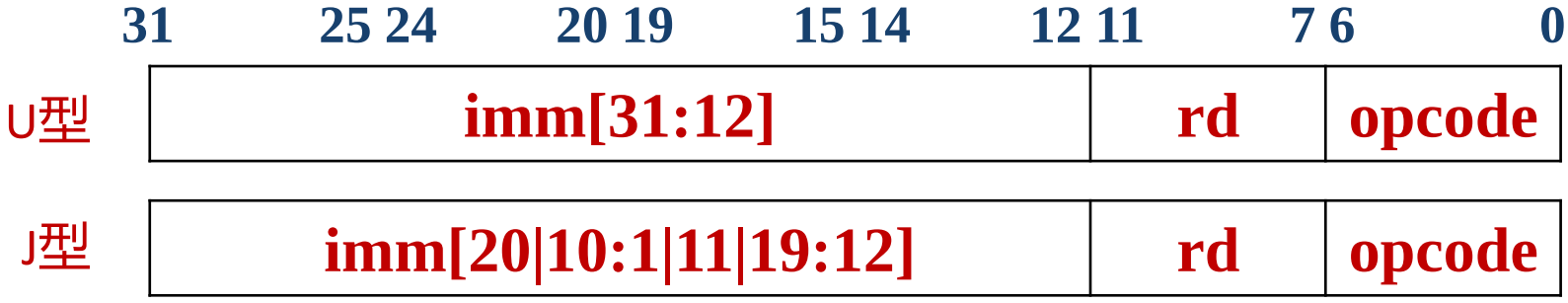
miniRV-1指令格式 —— I型指令 (9)

助记符	指令格式						使用语法	操作及解释
BIT#	31..25	24..20	19..15	14..12	11..7	6..0		(r)表示寄存器r的值; Mem[addr]表示地址为addr的存储单元的值; sext(a)表示a的符号扩展
I-类型	imm[11:0]		rs1	funct3	rd	opcode		
addi	imm[11:0]		rs1	000	rd	0010011	addi rd, rs1, imm	(rd) ← (rs1) + sext(imm)
andi	imm[11:0]		rs1	111	rd	0010011	andi rd, rs1, imm	(rd) ← (rs1) & sext(imm)
ori	imm[11:0]		rs1	110	rd	0010011	ori rd, rs1, imm	(rd) ← (rs1) | sext(imm)
xori	imm[11:0]		rs1	100	rd	0010011	xori rd, rs1, imm	(rd) ← (rs1) ^ sext(imm)
slli	0000000	shamt	rs1	001	rd	0010011	slli rd, rs1, shamt	(rd) ← (rs1) << shamt
srli	0000000	shamt	rs1	101	rd	0010011	srli rd, rs1, shamt	(rd) ← (rs1) >> shamt, 逻辑右移
srai	0100000	shamt	rs1	101	rd	0010011	srai rd, rs1, shamt	(rd) ← (rs1) >> shamt, 算术右移
lw	imm[11:0]		rs1	010	rd	0000011	lw rd, offset(rs1)	(rd) ← sext(Mem[(rs1) + sext(offset)][31:0])
jalr	imm[11:0]		rs1	000	rd	1100111	jalr rd, offset(rs1)	t ← (pc) + 4; (pc) ← ((rs1) + sext(offset)) & ~1; (rd) ← t

miniRV-1指令格式 —— S型、B型指令 (1+4)

助记符	指令格式						使用语法	操作及解释
BIT#	31..25	24..20	19..15	14..12	11..7	6..0		(r)表示寄存器r的值; Mem[addr]表示地址为addr的存储单元的值; sext(a)表示a的符号扩展
S-类型	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode		
sw	imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw rs2, offset(rs1)	Mem[(rs1) + sext(offset)] ← (rs2)[31:0]
B-类型	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode		
beq	imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	beq rs1, rs2, offset	if ((rs1) = (rs2)) (pc) ← (pc) + sext(offset)
bne	imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	bne rs1, rs2, offset	if ((rs1) ≠ (rs2)) (pc) ← (pc) + sext(offset)
blt	imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	blt rs1, rs2, offset	if ((rs1) < (rs2)) (pc) ← (pc) + sext(offset), 有符号比较
bge	imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	bge rs1, rs2, offset	if ((rs1) ≥ (rs2)) (pc) ← (pc) + sext(offset), 有符号比较

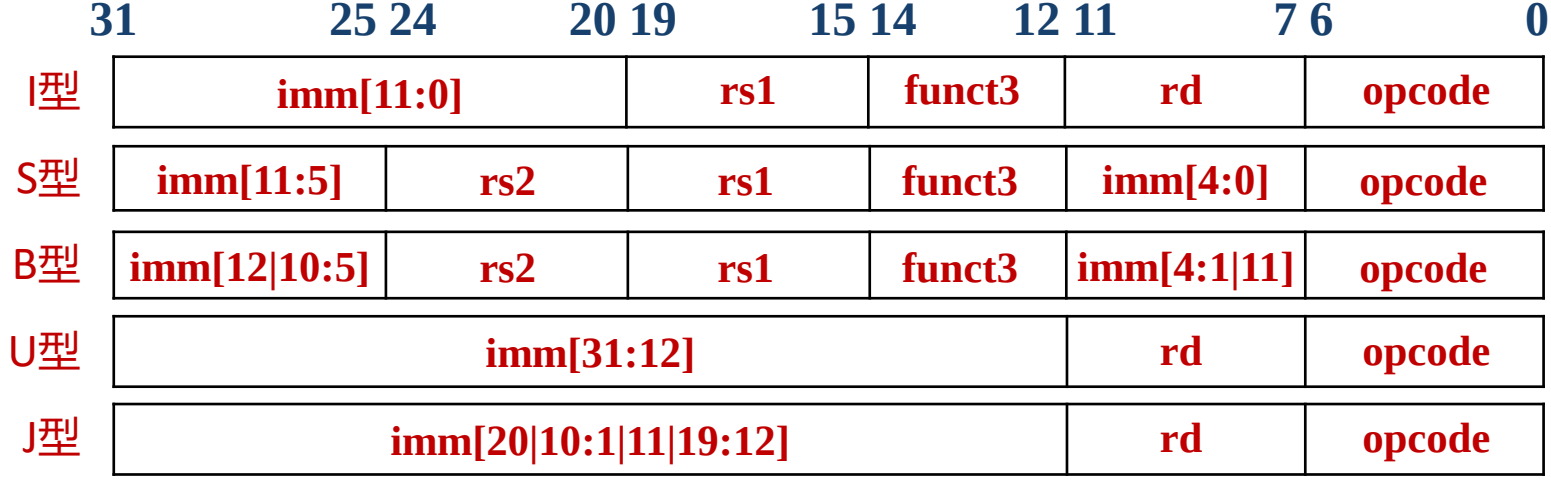
miniRV-1指令格式 —— U型、J型指令 (1+1)



BIT#	31..25	24..20	19..15	14..12	11..7	6..0		(r)表示寄存器r的值; Mem[addr]表示地址为addr的存储单元的值;
U-类型	imm[31:12]				rd	opcode		
lui	imm[31:12]				rd	0110111	lui rd, imm	(rd) ← sext(imm[31:12] << 12)
J-类型	imm[20|10:1|11|19:12]				rd	opcode		
jal	imm[20|10:1|11|19:12]				rd	1101111	jal rd, offset	(rd) ← pc + 4; pc += sext(offset)

miniRV-1指令格式 —— special for RISC-V

B型、J型指令的立即数被“切割”和“重排”，是为了**使得拥有共同数据通路的指令的编码拥有尽可能多的bit是重叠的**，从而简化数据通路和控制逻辑。



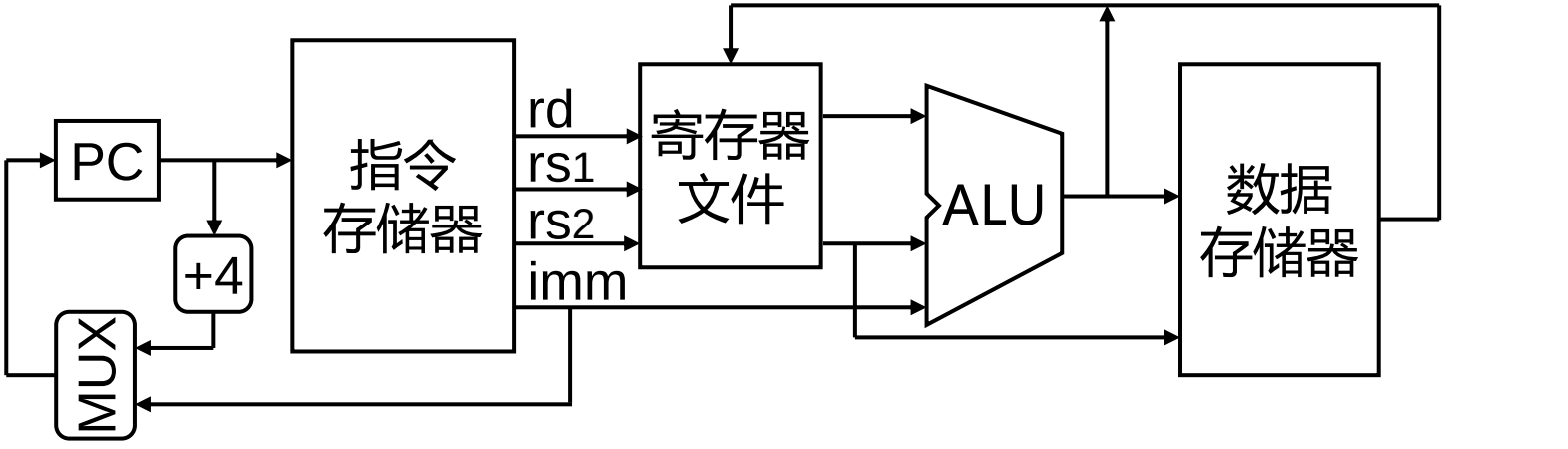
目录

CPU的基本功能

- 指令控制 —— 控制指令执行顺序
- 操作控制 —— 根据指令的功能，产生相应的控制信号
- 时序控制 —— 对各种操作进行时间上的控制
- 数据加工 —— 对数据进行算术/逻辑运算或其他处理
- 端口访问 —— 对来自存储器或I/O端口的数据进行访问
- 中断处理 —— 处理运行过程中的异常情况

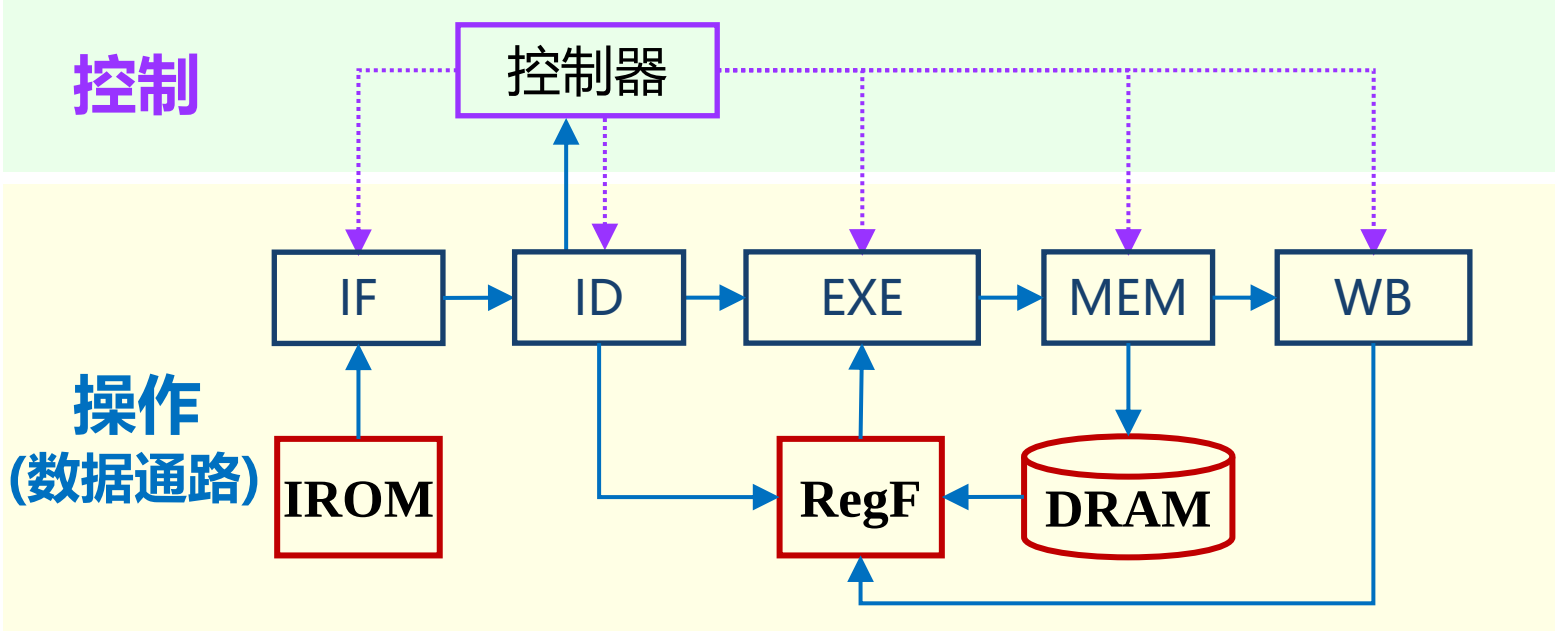
CPU执行指令的基本过程

- 从**数据流**的角度看指令执行过程



CPU执行指令的基本过程

- 从**控制流**的角度看指令执行过程



CPU = 数据通路 + 控制逻辑

CPU的基本结构

- 从结构上，CPU包含**数据通路**和**控制逻辑**
  - 数据通路
    - 数据所流经路径上的所有部件构成的通路，是CPU完成数据处理的物理基础
    - 包括PC、指令存储器、寄存器堆、ALU，etc
  - 控制逻辑
    - 控制运行时数据通路的具体功能，主要指控制器

目录

工程化方法设计CPU

CPU = 数据通路 + 控制逻辑

构造功能部件

构造一组必要的数据通路功能部件：对于每个功能部件，都需要定义其功能及接口信号，并用HDL描述其内部具体的电路行为或结构。

构造指令级别的数据通路与控制信号取值

根据指令操作语义，构造指令级别的数据通路与相关功能部件的控制信号取值。

综合数据通路

当所有指令级别的数据通路构造完毕后，将其综合为完整数据通路。

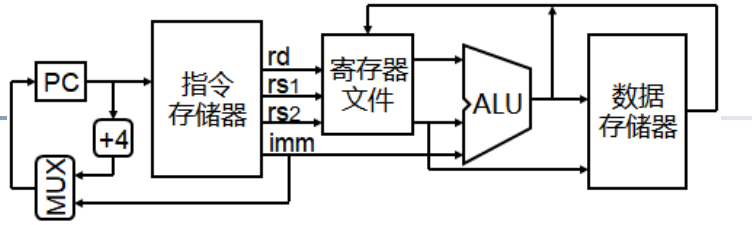
综合控制器

将全部指令对应的控制信号取值均建立后，合并所有的控制信号取值，形成控制信号矩阵，并生成相应的控制信号表达式。

生成工程项目

将第3、第4步的结果翻译为HDL语言，并在顶层工程文件中组装数据通路与控制器。

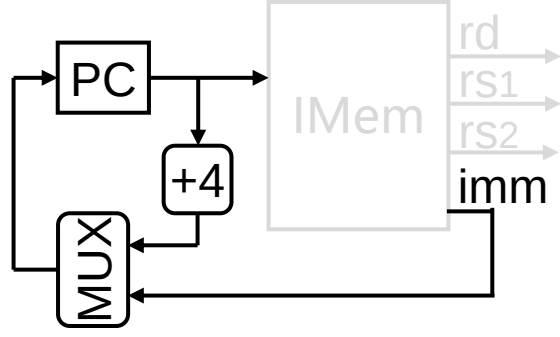
数据通路的主要功能部件



- ✓ **PC**（Program Counter）
- ✓ 指令存储器（Instruction Memory，**IROM**）
  - 只读：输入地址，输出指令
- ✓ 数据存储器（Data Memory，**DRAM**）
  - 读操作：输入地址，输出数据
  - 写操作：输入地址、数据
- ✓ 寄存器文件（Register File，**RF**）
  - 读操作：输入2个寄存器号，输出2个32bit的寄存器值
  - 写操作：输入1个寄存器号、待写入的32bit数据
- ✓ **ALU**（Arithmetic & Logic Unit）
  - 算术、逻辑、移位、比较运算：输入2个操作数，输出运算结果

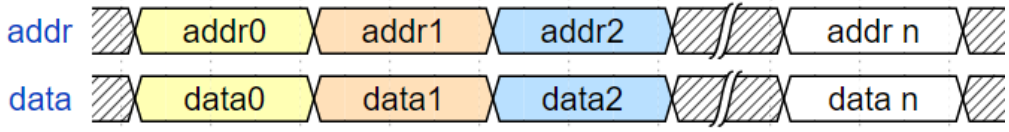
构造功能部件 —— PC设计

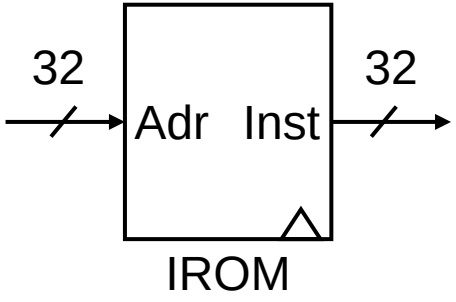
- PC是一个32bit寄存器，存储着当前指令的地址
  - 对于32位RISC CPU，指令均为32bit定长，即每条指令4Byte
  - 因此PC的BIT1和BIT0恒为0，故也可使用30bit寄存器
- CPU复位后PC的初始值即为首条指令的地址，如0x0000_0000H
- 执行时，需不停地更新PC的值
  - 默认情况下：PC <- PC + 4
  - 若遇到分支指令：PC <- imm



构造功能部件 —— IROM设计

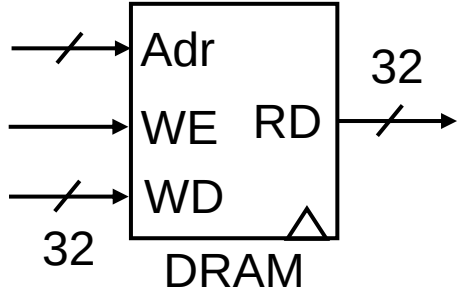
- IROM是一个单端口的只读存储器
  - 输入地址：PC
  - 输出数据：指令
- IROM的读取操作其实是组合逻辑





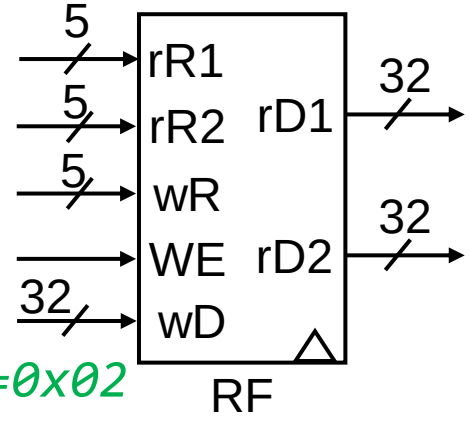
构造功能部件 —— DRAM设计

- DRAM是一个按地址访问的存储器，可读可写
  - 读操作：读使能有效，输入地址，输出数据，
  - 写操作：写使能有效，输入地址、数据
- DRAM的读写逻辑：
  - WE=0：RD <- DRAM[Adr]
  - WE=1：DRAM[Adr] <- WD



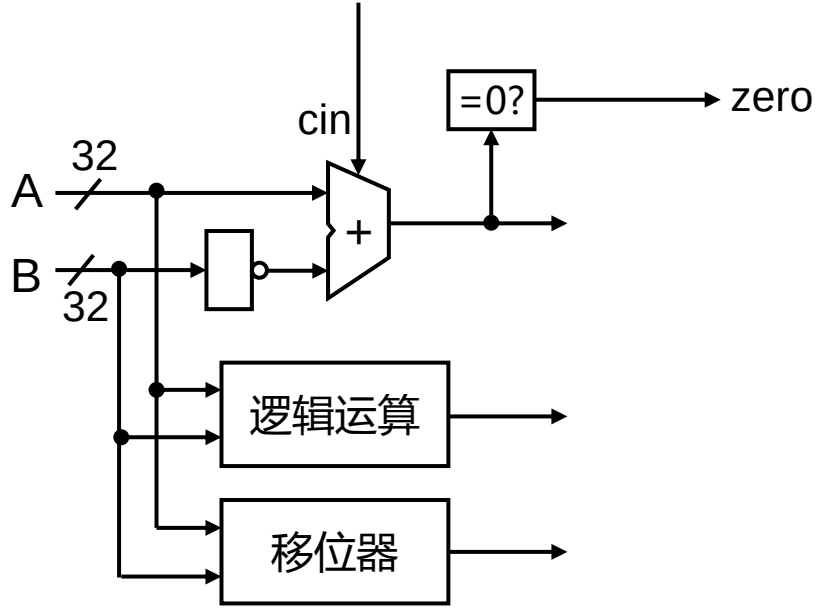
构造功能部件 —— RF设计

- RF包含32个32bit寄存器
  - 1条指令最多有2个源寄存器(rs1、rs2)、1个目标寄存器(rd)
  - 因此，RF需要3个“地址”端口、3个“数据”端口
  - 读操作：读使能有效，输入寄存器地址，输出寄存器存储的值
  - 写操作：写使能有效，输入寄存器地址、数据
- RF的读写逻辑：
  - WE=0: rD1<-REG[rR1]， rD2<-REG[rR2]
  - WE=1: REG[wR] <- wD
  - **e.g.** **add** *x2,x9,x5* *#rR1=0x09,rR2=0x05;wR=0x02*



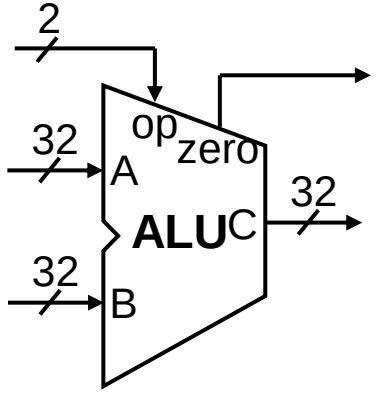
构造功能部件 —— ALU设计

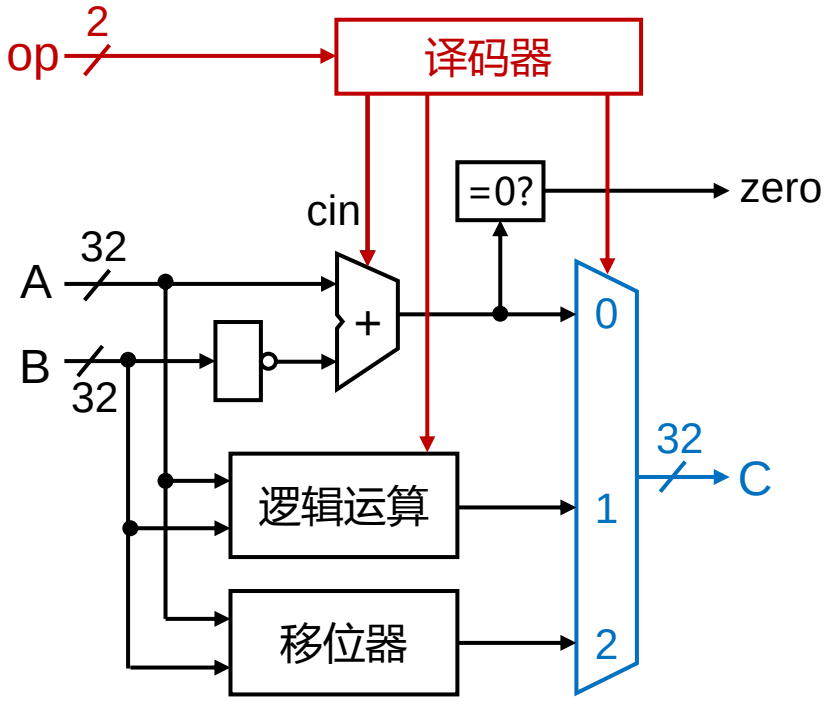
- ALU的功能：
  - 算术运算（add,sub）
    - ✓ 加法器、补码逻辑
  - 逻辑运算（and,or,etc）
    - ✓ 按位操作
  - 移位运算（sll,srl,etc）
    - ✓ 移位器
  - 比较运算（slt,sltu,etc）
    - ✓ 复用sub，判断标志位



构造功能部件 —— ALU设计

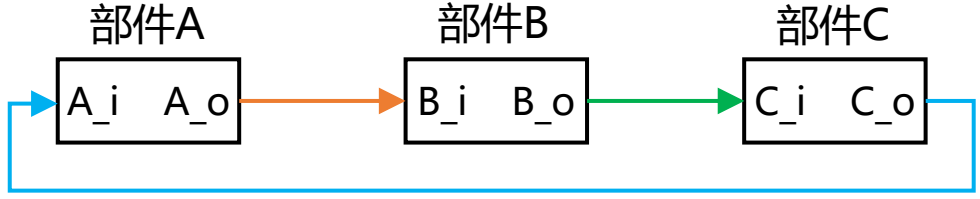
- 集成各项功能：
  - 多路选择器选择ALU输出
  - 译码器选择具体运算





构造数据通路

- 数据通路本质上是一个功能部件和连接关系的集合
  - 集合的元素：部件、部件的输入输出信号之间的连接关系
  - E.g. 3个部件的连接关系



  - 连接关系集合：{<C.C_o, A.A_i>, <A.A_o, B.B_i>, <B.B_o, C.C_i>}
  - 表格表示法：

部件A	部件B	部件C
A_i	B_i	C_i
C.C_o	A.A_o	B.B_o

部件的输入信号/引脚

由其他部件产生，并输入到当前引脚的信号

构造数据通路

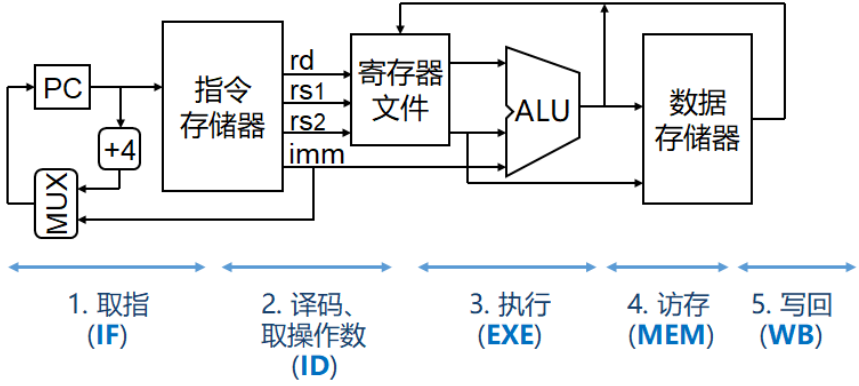
- 每条指令对应的部件和连接关系都不同
- 为了方便数据通路的综合，采用表格驱动设计的方法
  - E.g. 某数据通路包含PC、NPC、IROM、RF、ALU、DRAM的部件
  - 建立数据通路表：

所属单元	取指单元					译码单元					执行单元		存储单元	
部件	PC	NPC			IROM	RF				S_EXT	ALU		DRAM	
输入信号	din	PC	Imm	RA	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin

第1-2行：数据通路的功能部件及其所属单元
第3行：　各功能部件的输入信号/引脚
第4+行：　由其他部件产生，并连接到当前输入引脚的信号

构造数据通路

必要的功能部件

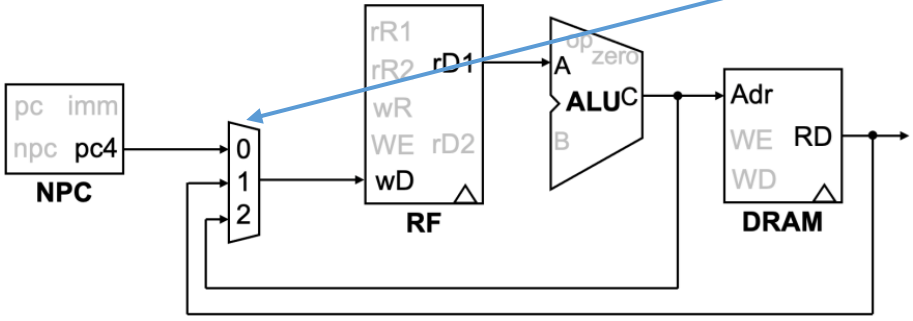


构造数据通路

所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				

综合数据通路

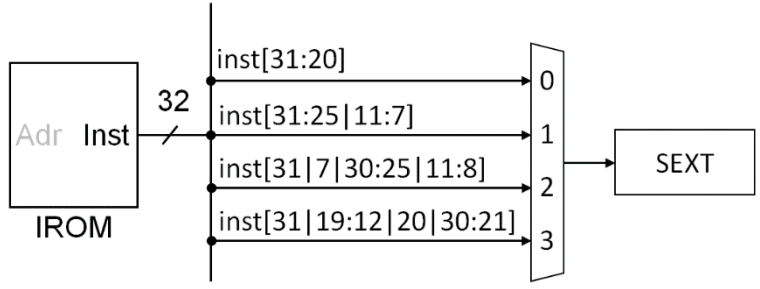
所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				
完整	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C DRAM.rd NPC.pc4	IROM.inst[31:7]	RF.rD1	RF.rD2 SEXT.ext	ALU.C	RF.rD2



综合方法1：使用多路选择器

综合数据通路

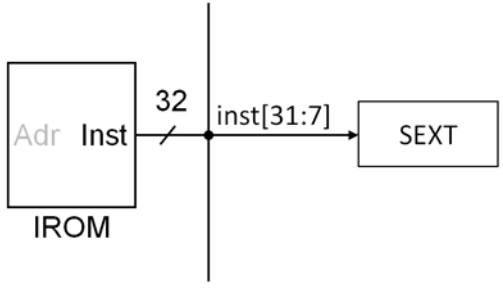
所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				
完整	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C DRAM.rd NPC.pc4	IROM.inst[31:7]	RF.rD1	RF.rD2 SEXT.ext	ALU.C	RF.rD2



综合方法2：增加输入信号

综合数据通路

所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				
完整	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C DRAM.rd NPC.pc4	IROM.inst[31:7]	RF.rD1	RF.rD2 SEXT.ext	ALU.C	RF.rD2



综合方法2：增加输入信号

构造控制单元

- 控制单元：根据指令的操作码 (opcode)和功能码 (funct3/funct7)，产生执行指令所需的**控制信号**

- 控制信号
  - 操作选择信号：指令执行时，选择部件要完成的具体操作
    - 多功能部件 (NPC、SEXT、ALU、存储器等)
  - 多路选择信号：对多输入源进行选择
    - 多路选择器
  - 分支控制信号：控制分支指令是否跳转
    - ALU

添加控制信号

- 在数据通路表中新增控制信号：

所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				
完整	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C DRAM.rd NPC.pc4	IROM.inst[31:7]	RF.rD1	RF.rD2 SEXT.ext	ALU.C	RF.rD2
操作选择信号	-	npc_op		-	-	-	rf_we		sext_op	alu_op		-	dram_we
多路选择信号	-	-		-	-	-	-	wd_sel	-	-	alub_sel	-	-
分支控制信号	-	-		-	-				-	branch		-	

确定控制信号取值

- 建立控制信号取值表：

指令	npc_op	rf_we	wd_sel	sext_op	alu_op	alub_sel	branch	dram_we

- 逐条指令分析，确定其控制信号取值：

指令	npc_op	rf_we	wd_sel	sext_op	alu_op	alub_sel	branch	dram_we
add	pc+4	1	'b00	X	ADD	0	0	0
sub	pc+4	1	'b00	X	SUB	0	0	0
ori	pc+4	1	'b00	'b00	OR	1	0	0
lw	pc+4	1	'b01	'b00	ADD	1	0	0
sw	pc+4	0	'bXX	'b01	ADD	1	0	1
beq	'b01	0	'bXX	'b10	SUB	0	1	0
jal	'b10	1	'b10	'b11	X	X	0	0

综合控制信号

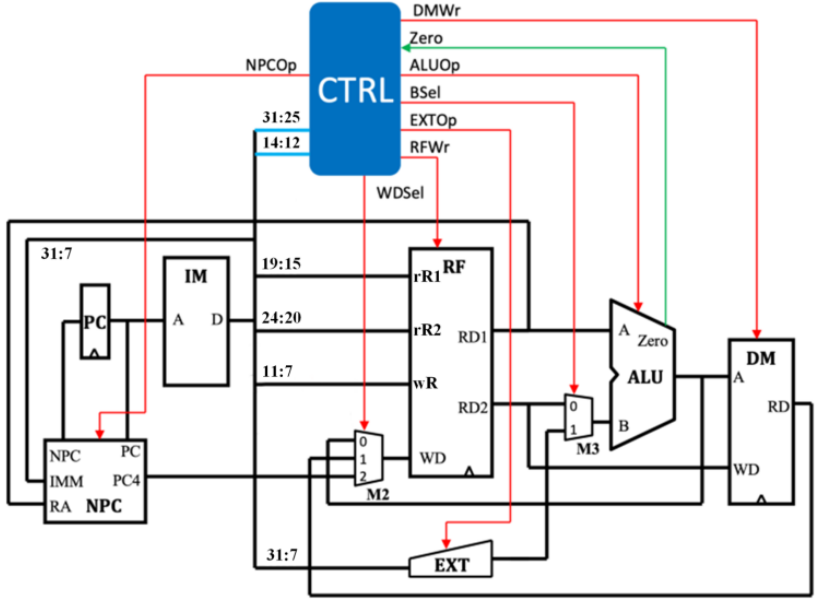
- 根据控制信号取值表，化简控制信号的逻辑表达式
- 根据表达式，设计控制单元
- 在数据通路图中添加控制单元

输入：opcode、funct3/funct7

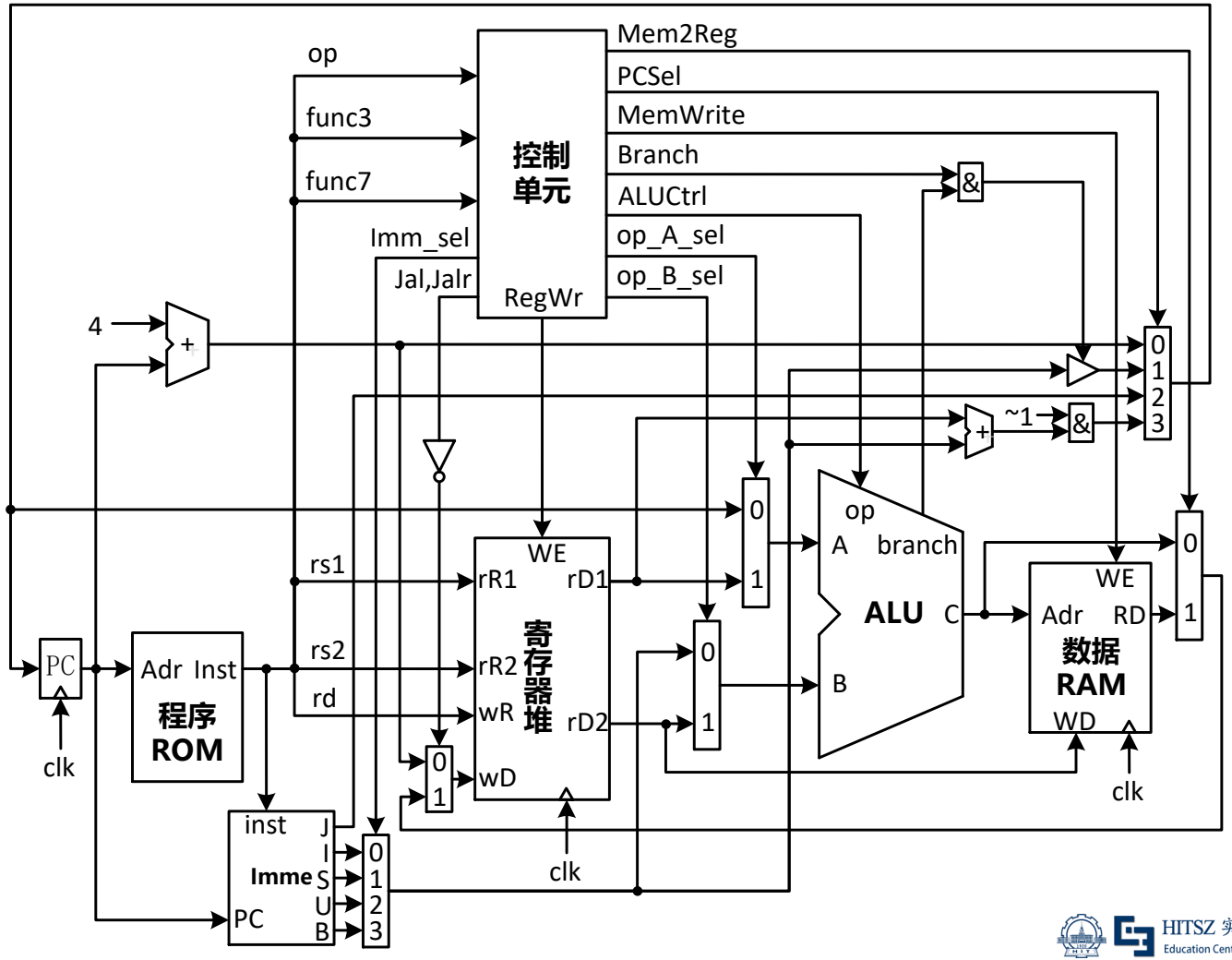
分支控制信号

输出：操作选择信号

多路选择信号



完整数据通路(示例)


op
func3
func7
Imm_sel
Jal,Jalr
控制单元
RegWr
Mem2Reg
PCSel
MemWrite
Branch
ALUCtrl
op_A_sel
op_B_sel
4
+
~1
&
0
1
2
3
PC
clk
Adr Inst
程序ROM
rs1
rs2
rd
WE
rR1
rD1
寄存器堆
rR2
wR
rD2
wD
op
A
branch
ALU
C
B
WE
Adr
RD
数据RAM
WD
inst
Imme
PC
J
I
S
U
B

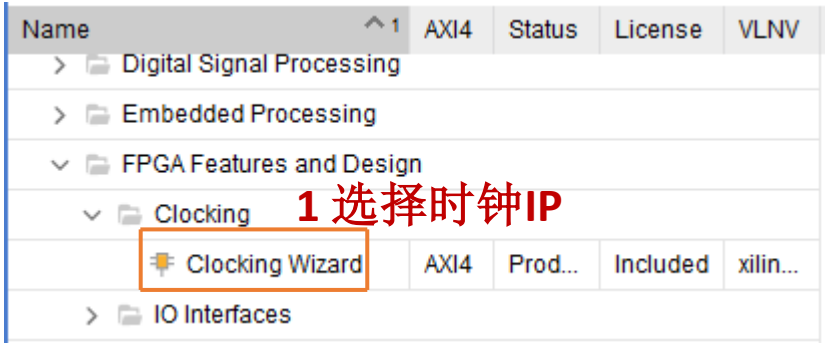
目录

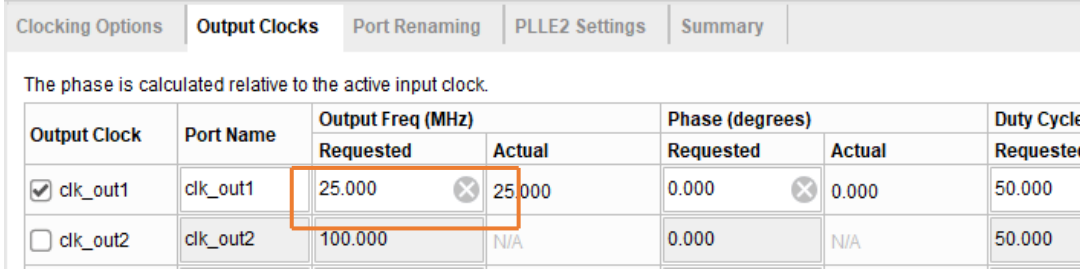
实现方法

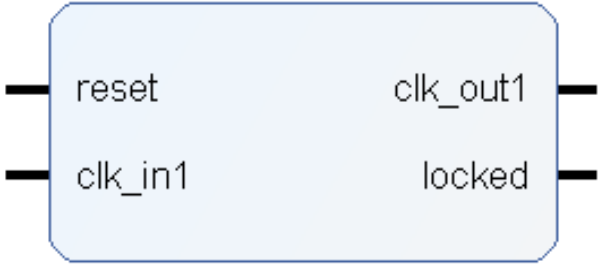
```
module miniRV(rst_i, clk_i, debug信号);
    input rst_i;            // 板上的Reset信号，低电平复位
    input clk_i;            // 板上的100MHz时钟信号
```

模 块	文 件	备 注
顶层模块	**miniRV.v**	**实例化、连接各部件**
时钟	**cpuclk.v**	**系统时钟 (25MHz)**
存储器模块	**prgrom.v**	**指令存储器 (64KB)**
	dmem.v	**数据存储器 (64KB)**
取指模块	*ifetch.v*	
译码模块 (含寄存器组)	*idecode.v*	
执行模块	*execute.v*	
控制器模块	*control.v*	

时钟模块实现 —— 使用IP核

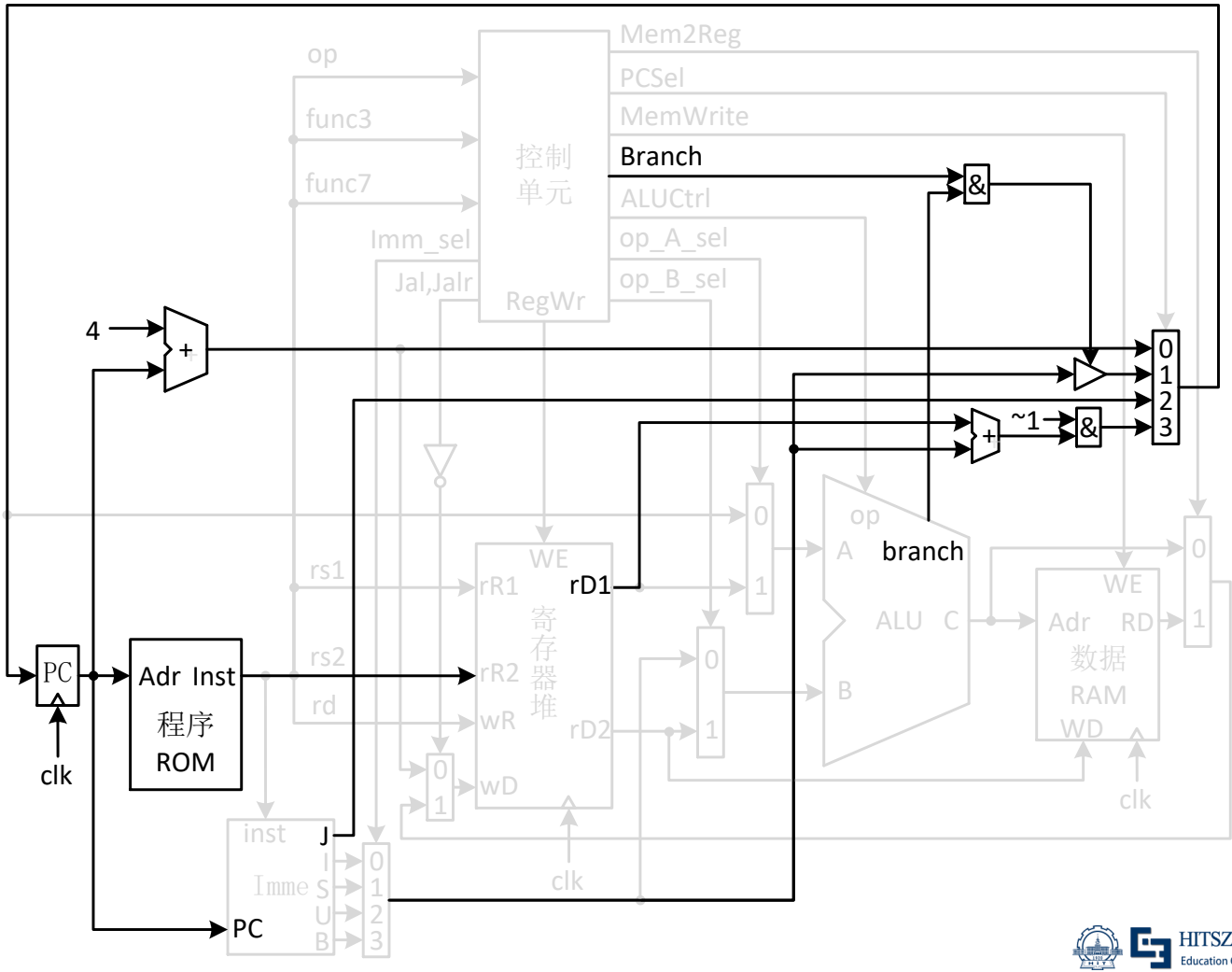






3 选择输出时钟频率

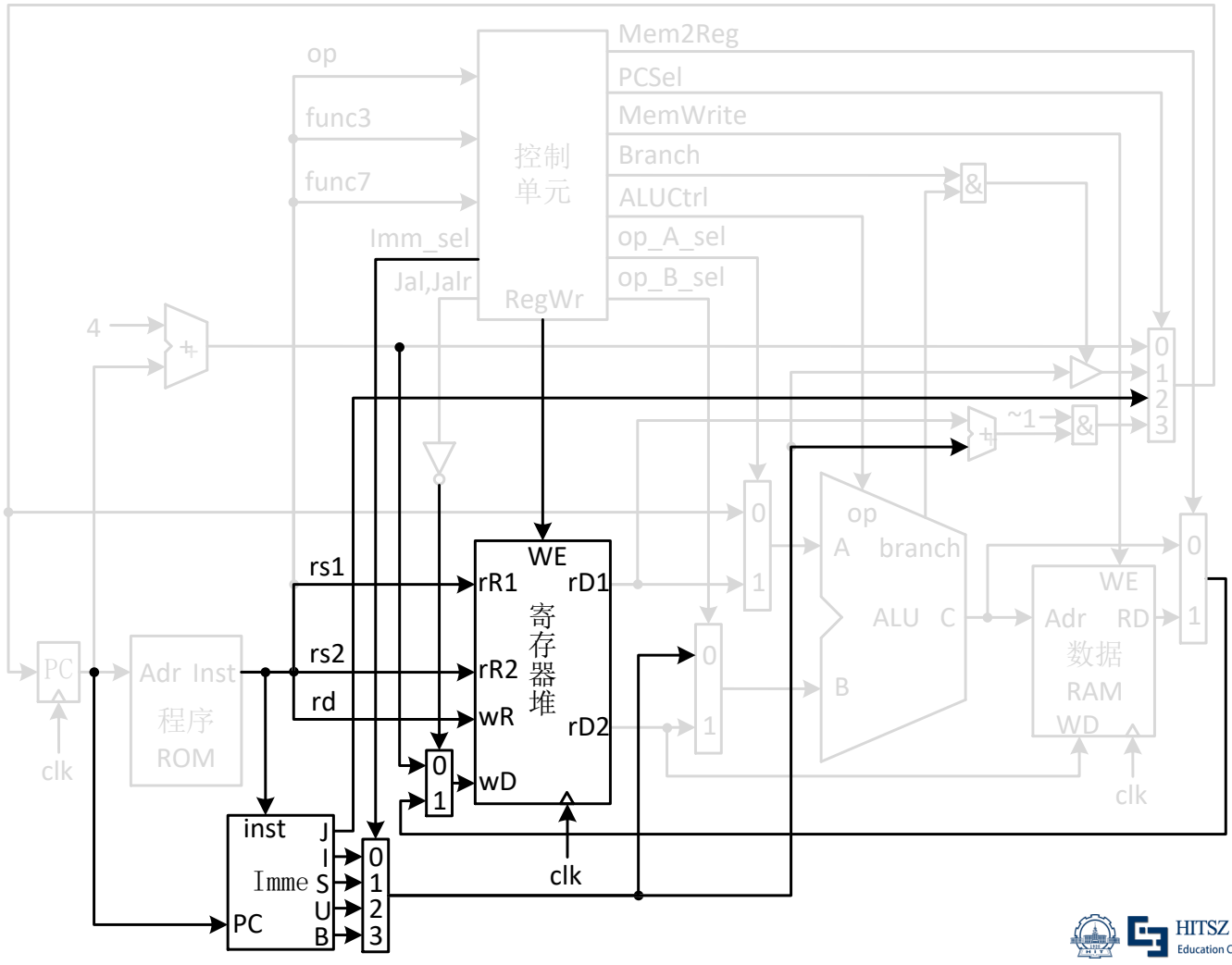
取指模块实现


op
func3
func7
Imm_sel
Jal,Jalr
控制单元
RegWr
Mem2Reg
PCSel
MemWrite
Branch
ALUCtrl
op_A_sel
op_B_sel
&
4
+
0
1
2
3
~1
&
+
PC
clk
Adr Inst
程序
ROM
rs1
rs2
rd
WE
rR1
rR2
wR
wD
寄存器堆
rD1
rD2
0
1
op
A
branch
ALU C
B
WE
Adr RD
数据
RAM
WD
clk
inst
J
Imme
I
S
U
B
0
1
2
3
clk
PC

取指模块实现

- 使用Distributed Memory IP核建立IROM
- 利用PC，从IROM中取出指令
- 对PC值进行+4
- 完成几种跳转指令的PC修改功能
  - 要点：根据其他单元的关键信号，用ALU计算结果or立即数修改PC
- 最终修改PC值
  - 要点：使用多路选择器

译码模块实现


op
func3
func7
Imm_sel
Jal,Jalr
控制单元
RegWr
Mem2Reg
PCSel
MemWrite
Branch
ALUCtrl
op_A_sel
op_B_sel
4
~1
rs1
rs2
rd
WE
rR1
rR2
wR
wD
rD1
rD2
寄存器堆
clk
PC
Adr Inst
程序ROM
inst
Imme
J
I
S
U
B
op
A
branch
ALU
C
B
WE
Adr
RD
数据RAM
WD
0
1
2
3

译码模块实现

- 分解指令中的各个字段 —— 按照指令格式，分别取出各个字段
  - ✓ Tips：用**assign**语句、位选择符**[]**、位拼接符**{}**实现

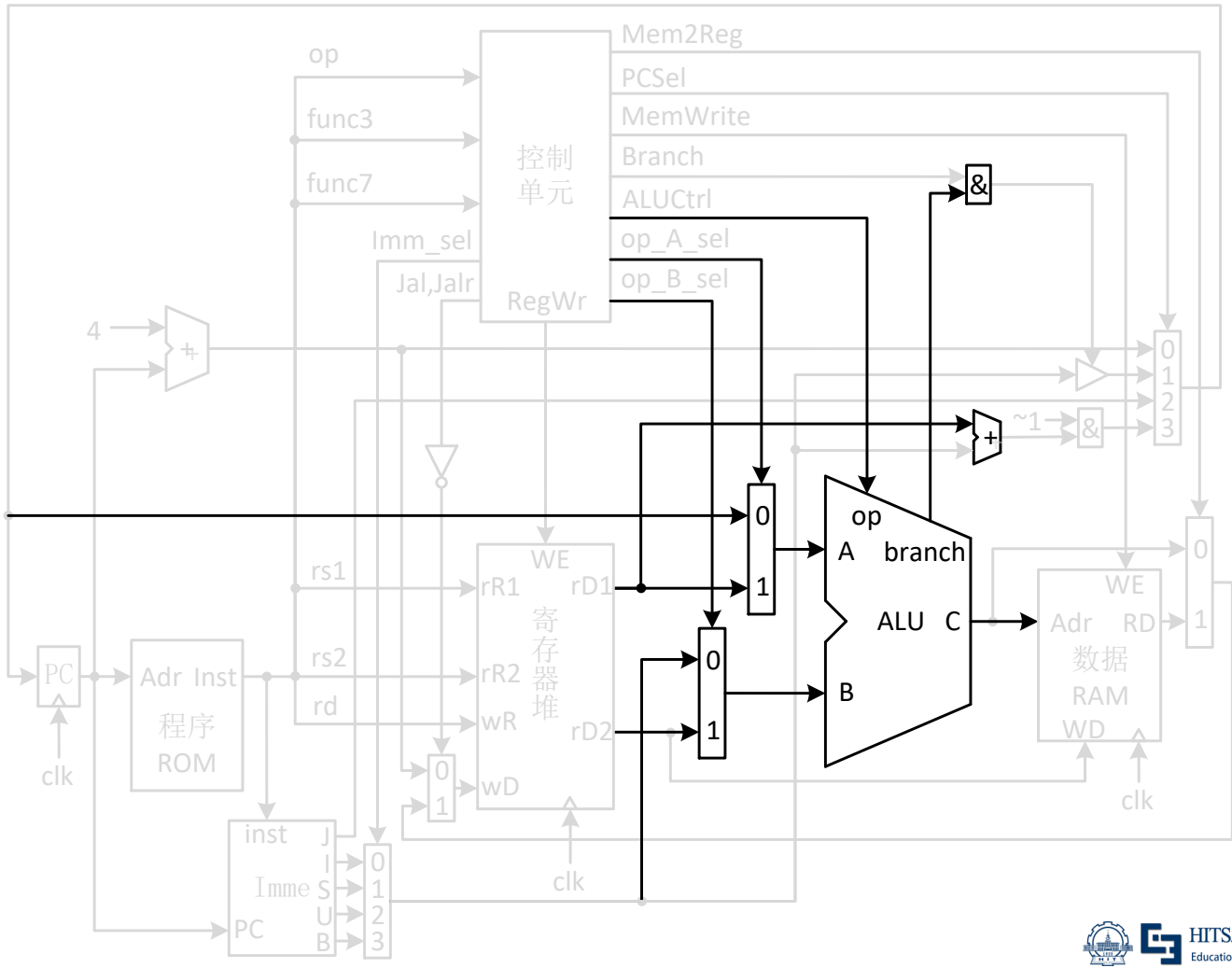
	31	25 24	20 19	15 14	12 11	7 6	0
R 型	funct7	rs2	rs1	funct3	rd	opcode	
I 型	imm[11:0]		rs1	funct3	rd	opcode	
S 型	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	
B 型	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	
U 型	imm[31:12]				rd	opcode	
J 型	imm[20|10:1|11|19:12]				rd	opcode	

译码模块实现

- 分解指令中的各个字段
- 建立寄存器文件RF
- 对寄存器文件进行读写操作
  - 要点：根据分解指令得到的寄存器号和来自其他单元的关键信号，读写相应寄存器
- 对立即数进行符号扩展
  - 要点：lui、jal指令的立即数是20位，其余是12位

    不同类型的指令，其立即数扩展操作不同

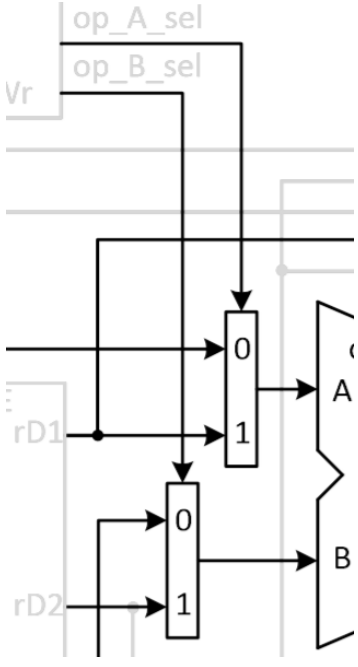
执行模块实现


op
func3
func7
控制单元
Mem2Reg
PCSel
MemWrite
Branch
ALUCtrl
op_A_sel
op_B_sel
Imm_sel
Jal,Jalr
RegWr
4
+
&
~1
0
1
2
3
op
A
branch
ALU
C
B
rs1
rs2
rd
WE
rR1
rR2
wR
wD
rD1
rD2
寄存器堆
PC
Adr
Inst
程序ROM
clk
inst
Imme
J
I
S
U
B
Adr
RD
数据RAM
WD

执行模块实现

- 选择运算数据
  - 要点：根据控制器的操作数选择信号，选择不同的源操作数：

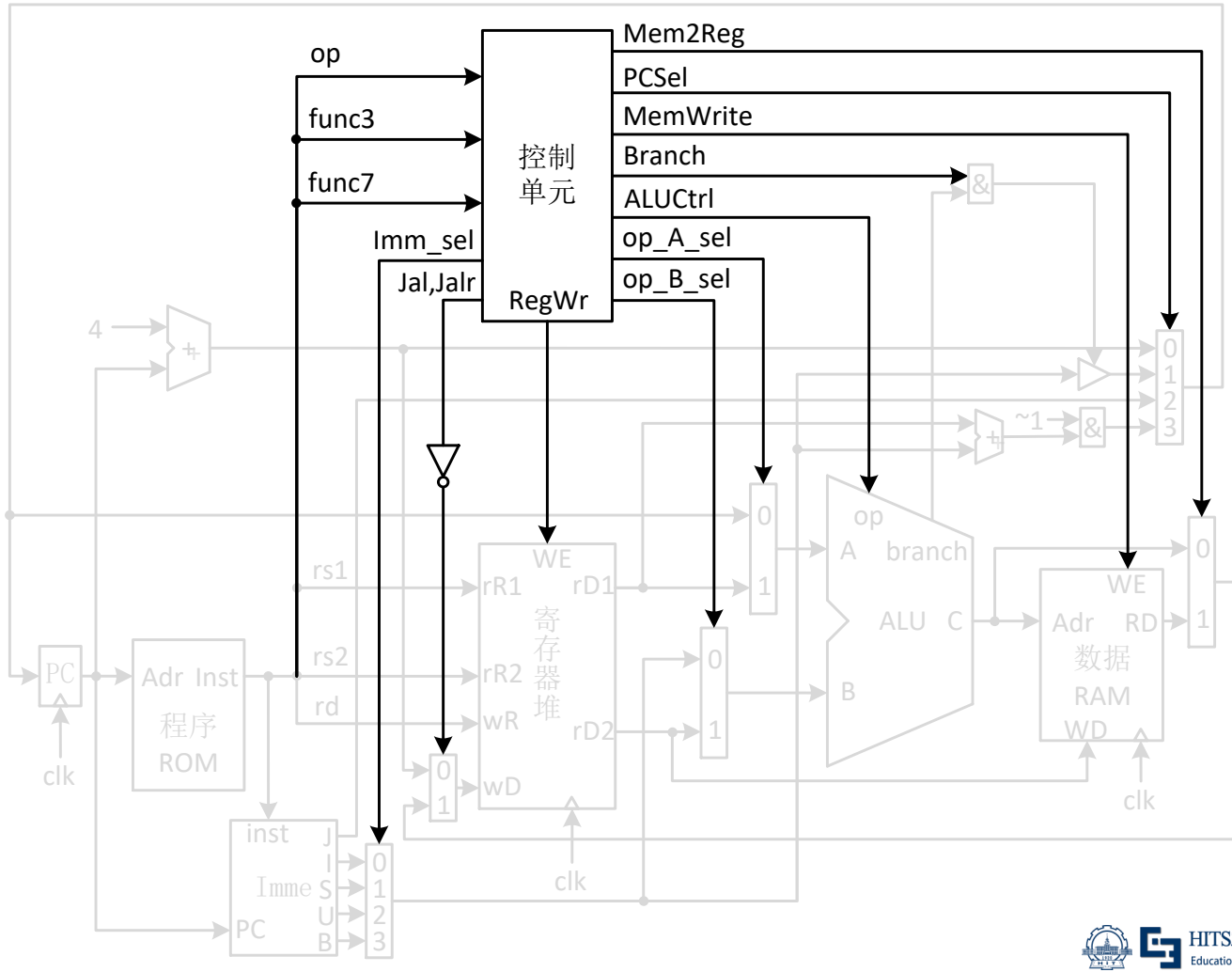
```
input [31:0] rd_dat1_i;   // (rs1)
input [31:0] rd_dat2_i;   // (rs2)
input [31:0] npc_i;       // Next PC
input [31:0] imm_i;       // 扩展后的立即数

assign op_A = op_A_sel ? npc_i : rd_dat1_i;
assign op_B = op_B_sel ? imm_i : rd_dat2_i;
```



执行模块实现

- 选择运算数据
- 完成算术、逻辑、移位运算
  - 要点：调用设计好的ALU模块；用控制器的op信号选择运算功能
- 完成比较运算的PC值计算
  - 要点：B型、J型指令的立即数在符号扩展后，需要加上PC的值
- 完成运算结果输出（含比较后赋值）
  - 要点：根据控制器的ALUCtrl等控制信号，输出相应的运算结果

控制模块实现


op
func3
func7
Imm_sel
Jal,Jalr
控制单元
RegWr
Mem2Reg
PCSel
MemWrite
Branch
ALUCtrl
op_A_sel
op_B_sel
4
PC
clk
Adr Inst
程序ROM
rs1
rs2
rd
WE
rR1
rR2
wR
wD
rD1
rD2
寄存器堆
clk
op
A
branch
ALU
C
B
~1
WE
Adr
RD
数据RAM
WD
clk
inst
Imme
PC
J
I
S
U
B

控制器实现

- 考察每条指令在数据通路中的执行过程和涉及到的控制信号的取值
- 根据指令和控制信号的关系，写出每个控制信号之间的逻辑表达式
- 必要时，使用卡诺图化简逻辑表达式
- 根据逻辑表达式，用Verilog HDL实现相应的控制逻辑
- 综合所有控制逻辑，形成控制器

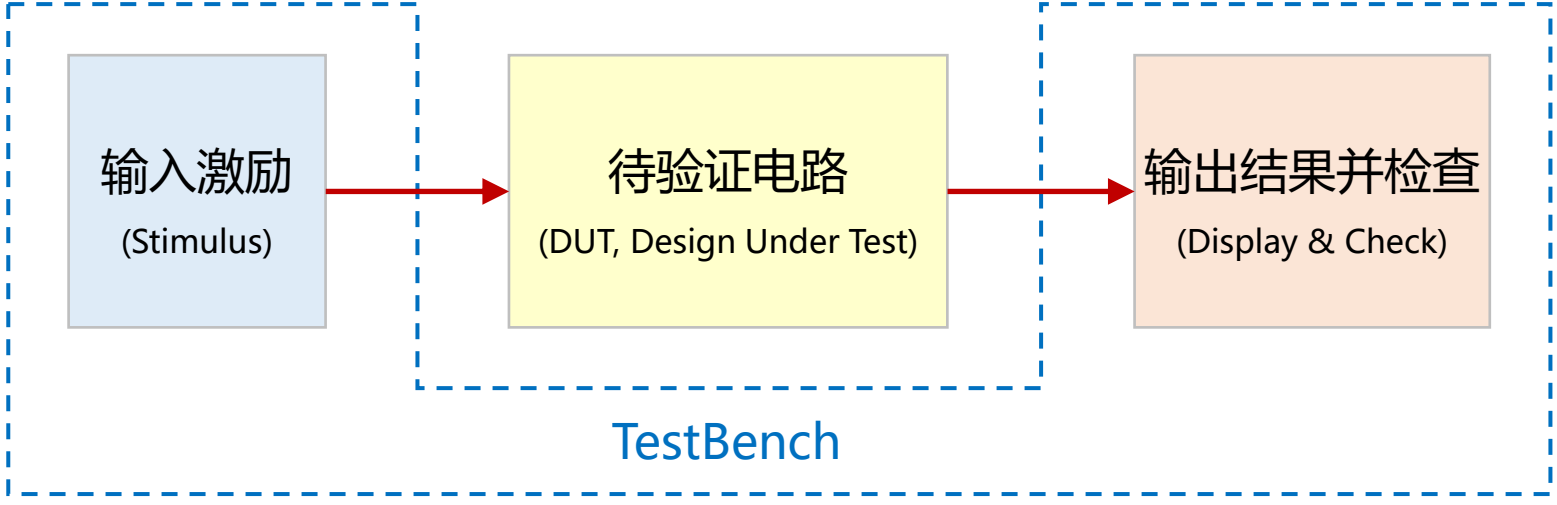
控制器实现

- 要点：
  - 根据控制信号取值表，为每个需要输出到其他单元的控制信号写出相应的逻辑表达式
  - 表达式的输入：opcode、funct3、funct7

指令	npc_op	rf_we	wd_sel	sext_op	alu_op	alub_sel	branch	dram_we
add	pc+4	1	'b00	X	ADD	0	0	0
sub	pc+4	1	'b00	X	SUB	0	0	0
ori	pc+4	1	'b00	'b00	OR	1	0	0
lw	pc+4	1	'b01	'b00	ADD	1	0	0
sw	pc+4	0	'bXX	'b01	ADD	1	0	1
beq	'b01	0	'bXX	'b10	SUB	0	1	0
jal	'b10	1	'b10	'b11	X	X	0	0

数字电路的功能验证

- 如何对数字电路的功能进行验证？ —— 时序仿真
- 仿真 —— 基于软件模拟 (而非电路实测) 来验证电路功能的方法



CPU的功能验证

- CPU也是数字电路，也可采用相同的验证方法，但效率太低
  - 借助外设输出
  - 出错点通过测试程序的逻辑路径传递很远之后才能被发现
- 改进：
  - 用**一段指令序列**作为输入激励，通过**观察指令执行结果**来验证CPU功能
  - 验证效率大幅提高，但难以定位错误
- 更好的方法：基于Trace比对的功能验证

CPU的功能验证 —— 基于Trace比对

- Trace：CPU执行指令序列时产生的信息 (PC、写寄存器的信息，etc)
- 基于Trace比对的验证方法：
  - ① 用已知功能正确的CPU运行测试程序，记录Trace0 (Golden Trace)
  - ② 用待验证CPU运行测试程序，产生Trace1
  - ③ 将Trace1与Trace0进行实时比对，如果出现不同，立即报错并停止
- 特殊情况：
  - Store指令 —— 后续相关的Load指令写入寄存器的值与GT不同

作业 (DDL: 6月29日 23:59)

- 完成数据通路和控制信号的构造、综合

所属单元	取指单元				译码单元					执行单元		存储单元	
部件	PC	NPC		IROM	RF				SEXT	ALU		DRAM	
输入信号	din	PC	imm	adr	rR1	rR2	wR	wD	din	A	B	adr	wdin
add	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
sub	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C		RF.rD1	RF.rD2		
ori	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	ALU.C	IROM.inst[31:20]	RF.rD1	SEXT.ext		
lw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]		IROM.inst[11:7]	DRAM.rd	IROM.inst[31:20]	RF.rD1	SEXT.ext	ALU.C	
sw	NPC.npc	PC.pc		PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31:25|11:7]	RF.rD1	SEXT.ext	ALU.C	RF.rD2
beq	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]			IROM.inst[31|7|30:25|11:8]	RF.rD1	RF.rD2		
jal	NPC.npc	PC.pc	SEXT.ext	PC.pc			IROM.inst[11:7]	NPC.pc4	IROM.inst[31|19:12|20|30:21]				
完整	NPC.npc	PC.pc	SEXT.ext	PC.pc	IROM.inst[19:15]	IROM.inst[24:20]	IROM.inst[11:7]	ALU.C DRAM.rd NPC.pc4	IROM.inst[31:7]	RF.rD1	RF.rD2 SEXT.ext	ALU.C	RF.rD2
操作选择信号	-	npc_op		-	-	-	rf_we		sext_op	alu_op		-	dram_we
多路选择信号	-	-		-	-	-	-	wd_sel	-	-	alub_sel	-	-
分支控制信号	-	-		-	-				-	branch		-	

- 完成控制信号的取值表

指令	npc_op	rf_we	wd_sel	sext_op	alu_op	alub_sel	branch	dram_we
add	pc+4	1	'b00	X	ADD	0	0	0
sub	pc+4	1	'b00	X	SUB	0	0	0
ori	pc+4	1	'b00	'b00	OR	1	0	0
lw	pc+4	1	'b01	'b00	ADD	1	0	0
sw	pc+4	0	'bXX	'b01	ADD	1	0	1
beq	'b01	0	'bXX	'b10	SUB	0	1	0
jal	'b10	1	'b10	'b11	X	X	0	0

谢　谢!