实验三 单周期CPU设计

2024春季

zjx@ustc.edu.cn

实验目标

- **理解单周期LA32R CPU的结构和工作原理**
- **掌握单周期LA32R CPU的设计和调试方法**
- **掌握查看生成电路及其性能和资源使用情况**

实验内容

- **设计单周期LA32R (LoongArch-32 Reduced) CPU，可以执行如下指令：**
  - 算术/比较运算：add.w, addi.w, lu12i.w, sub.w, pcaddu12i, slt, sltu, slti, sltui, mul
  - 逻辑运算：and, or, nor, xor, andi, ori, xori
  - 移位运算：slli.w, srli.w, srai.w, sll.w, srl.w, sra.w
  - 访存：ld.w, st.w, ld.b, st.b, st.h, ld.h, ld.hu, ld.bu
  - 转移：bne, beq, b, bl, jirl, blt, bge, bltu, bgeu
- **构建包含LA32R CPU的片上系统 (System on Chip, SoC)，连接串行调试单元 (Serial Debug Unit, SDU)上板验证**

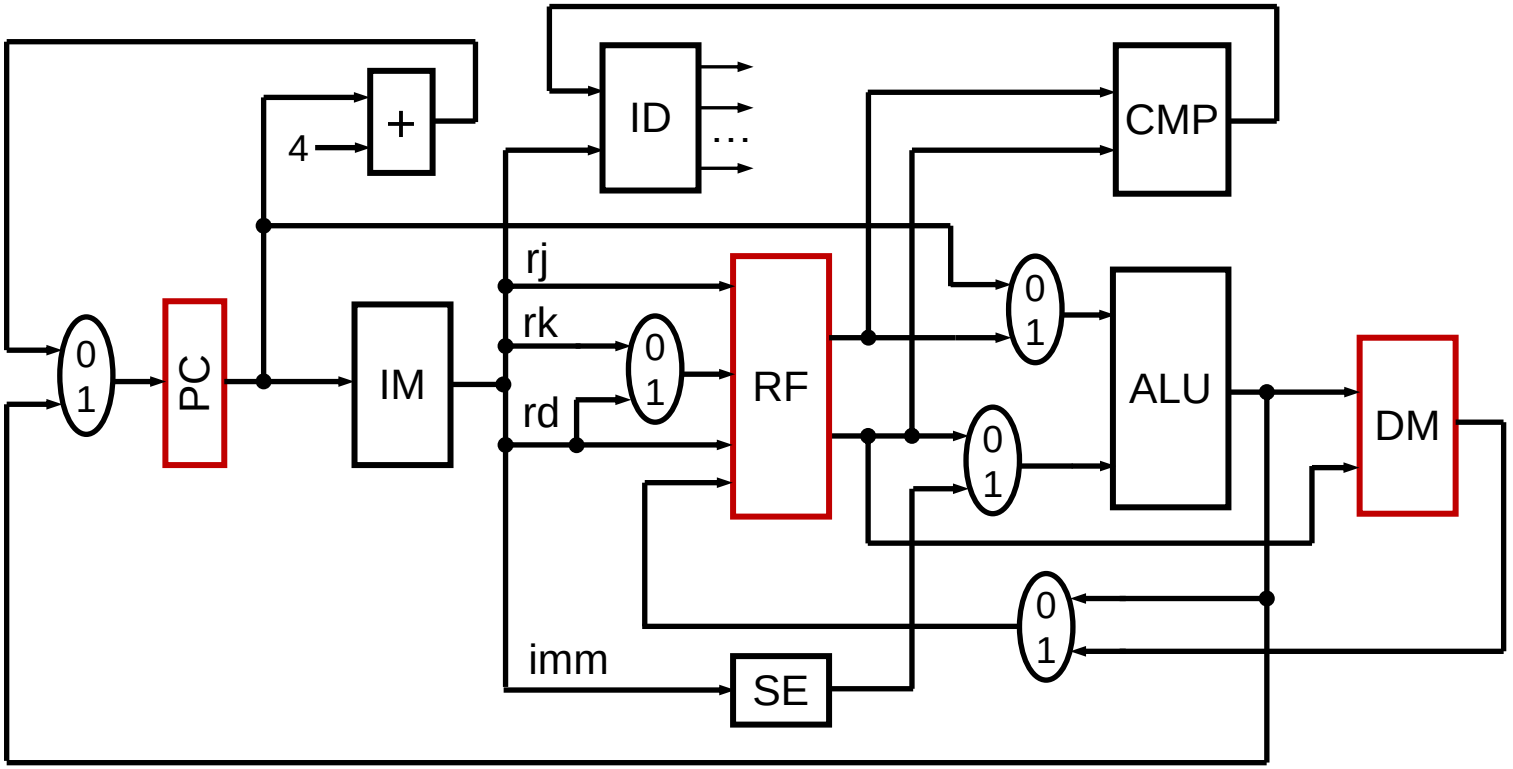
LA32R C1指令集

- add.w rd, rj, rk # rd = rj + rk
- addi.w rd, rj, si12 # rd = rj + SE(si12)
- lu12i.w rd, si20 # rd = {si20, 12'b0}
- ld.w rd, rj, si12 # rd = M[rj + SE(si12)]
- st.w rd, rj, si12 # M[rj + SE(si12)] = rd
- bne rj, rd, offs16 # if (rj != rd) pc += SE({offs16, 2'b0})

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
add.w	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	rk					rj					rd				
addi.w	0	0	0	0	0	0	1	0	1	0	si12												rj					rd				
ld.w	0	0	1	0	1	0	0	0	1	0	si12												rj					rd				
st.w	0	0	1	0	1	0	0	1	1	0	si12												rj					rd				
lu12i.w	0	0	0	1	0	1	0	si20																				rd				
bne	0	1	0	1	1	1	offs16																rj					rd				

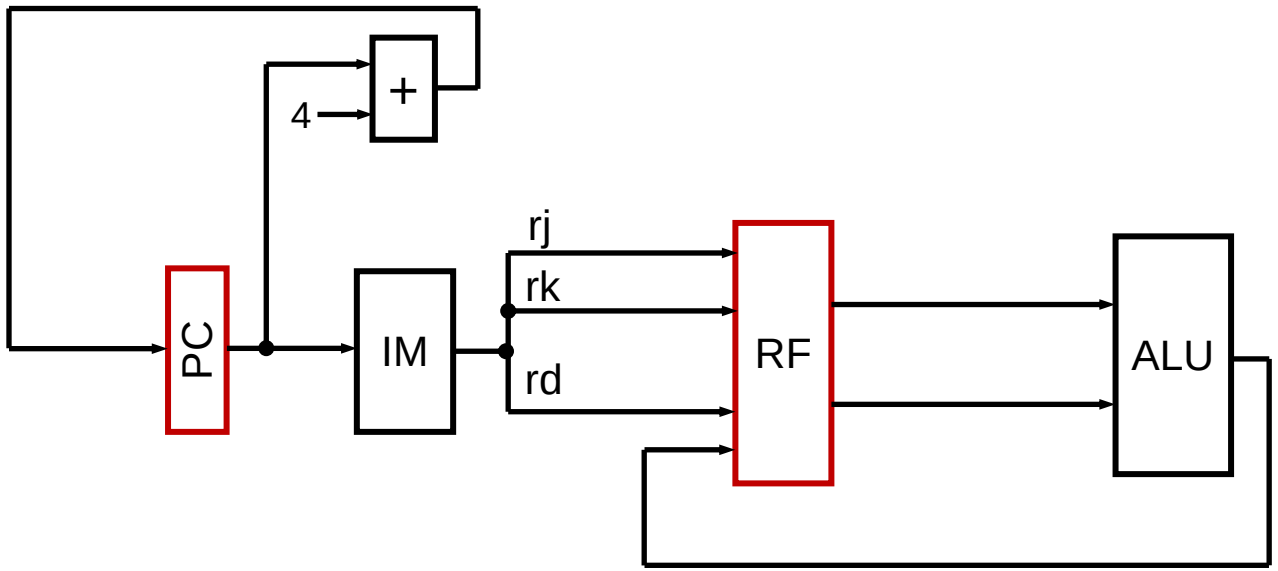
单周期数据通路

- LA32R C1-CPU的数据通路



单周期数据通路：add.w

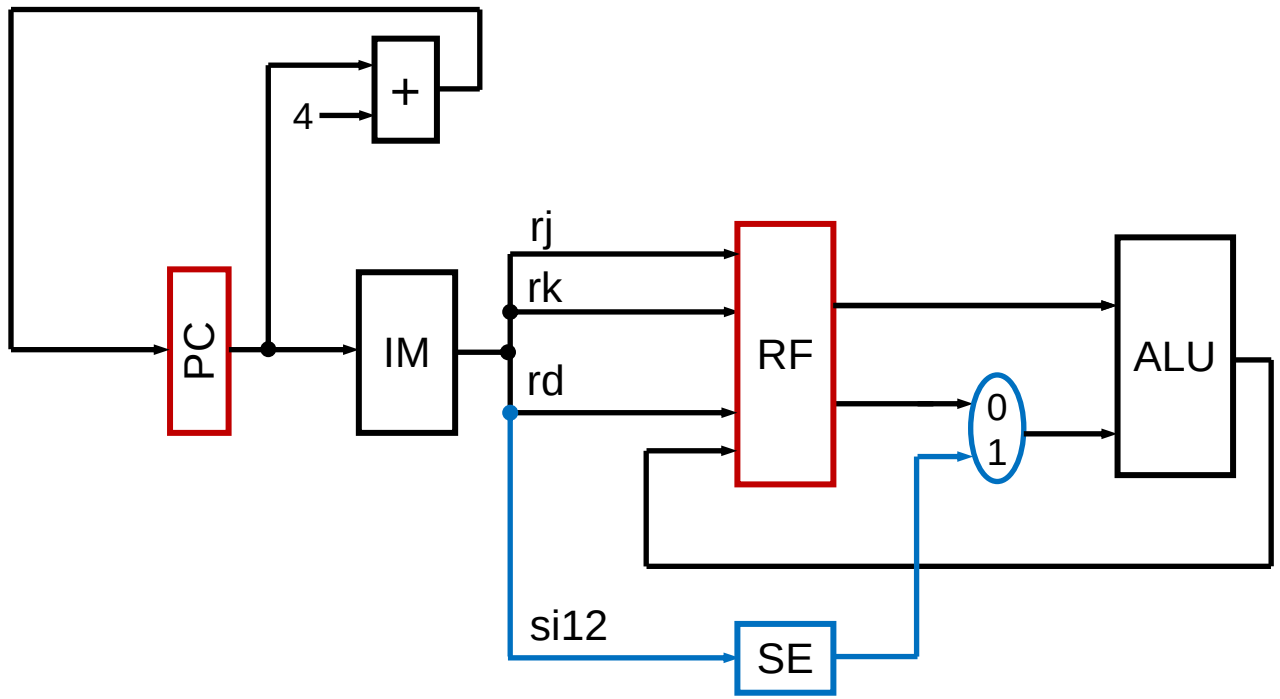
- add.w rd, rj, rk　　# rd = rj + rk, pc = pc +4



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
add.w	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	rk					rj							rd		

单周期数据通路：addi.w

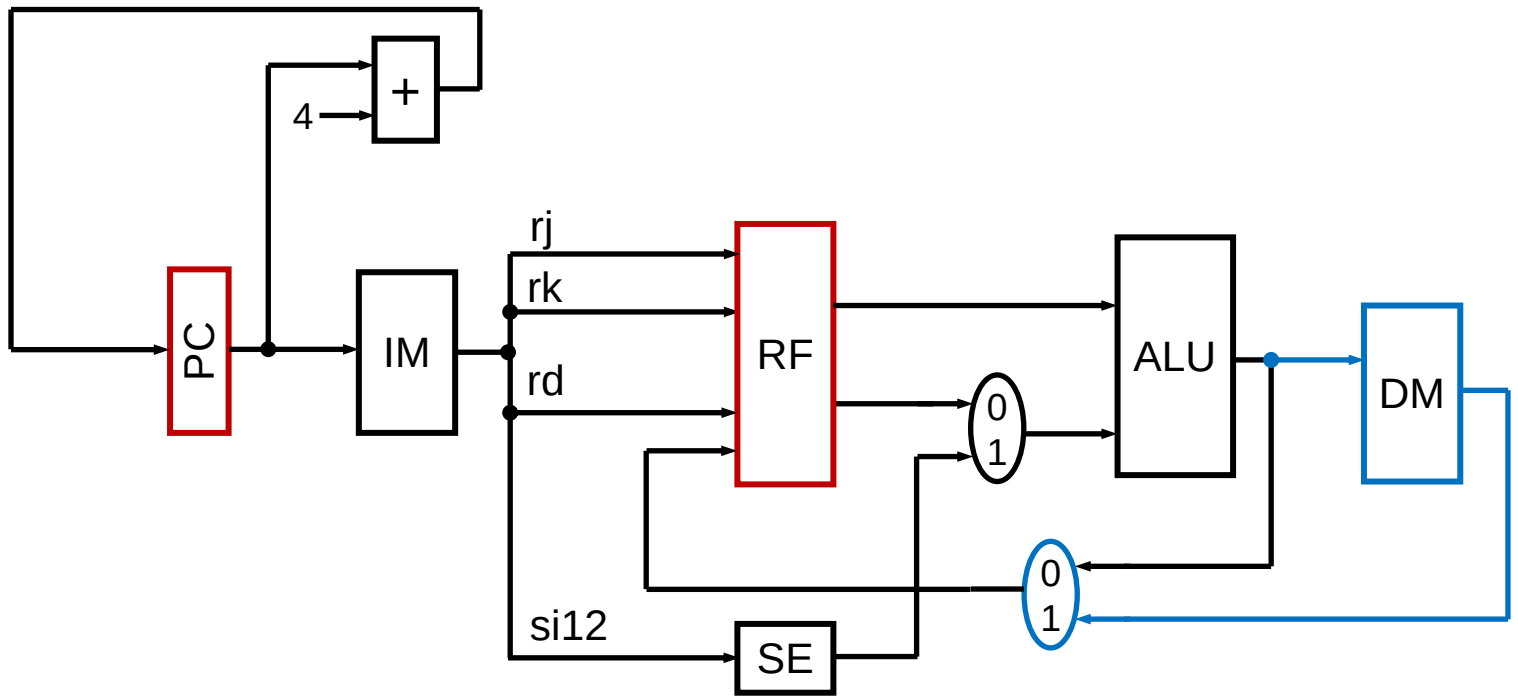
- addi.w rd, rj, si12 # rd = rj + SE(si12)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
addi.w	0	0	0	0	0	0	1	0	1	0	si12												rj					rd				

单周期数据通路：ld.w

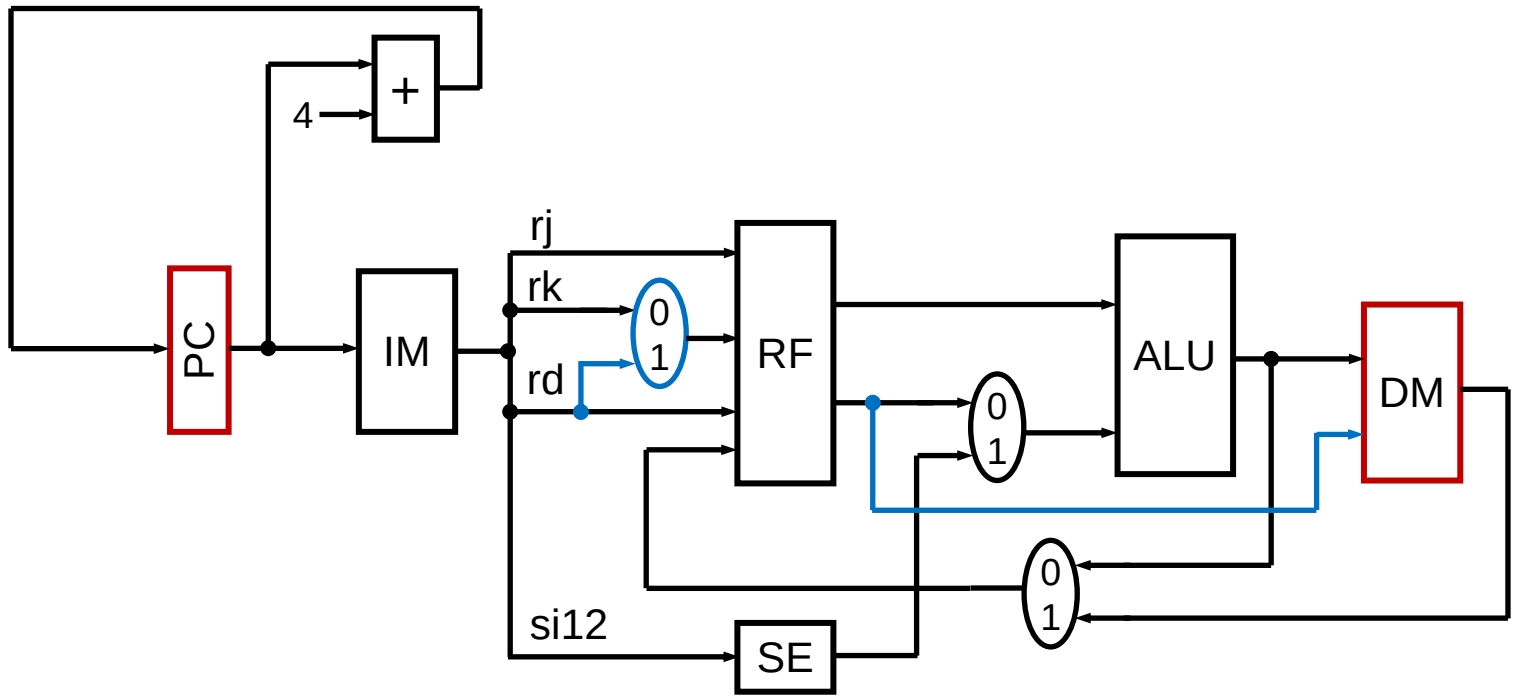
- ld.w rd, rj, si12　　# rd = M[rj + SE(si12)]



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ld.w	0	0	1	0	1	0	0	0	1	0	si12												rj					rd				

单周期数据通路：st.w

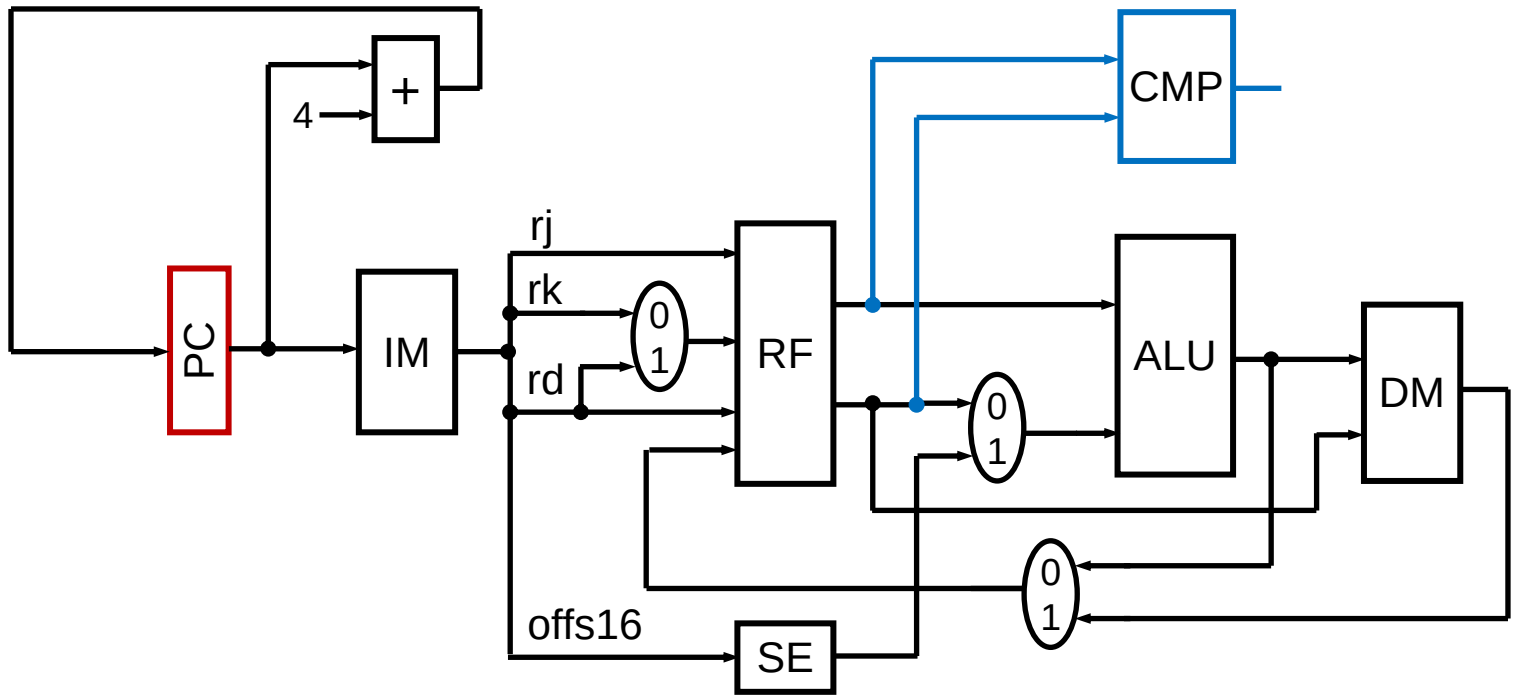
- st.w rd, rj, si12　　# M[rj + SE(si12)] = rd



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
st.w	0	0	1	0	1	0	0	1	1	0	si12												rj					rd				

单周期数据通路：bne (1)

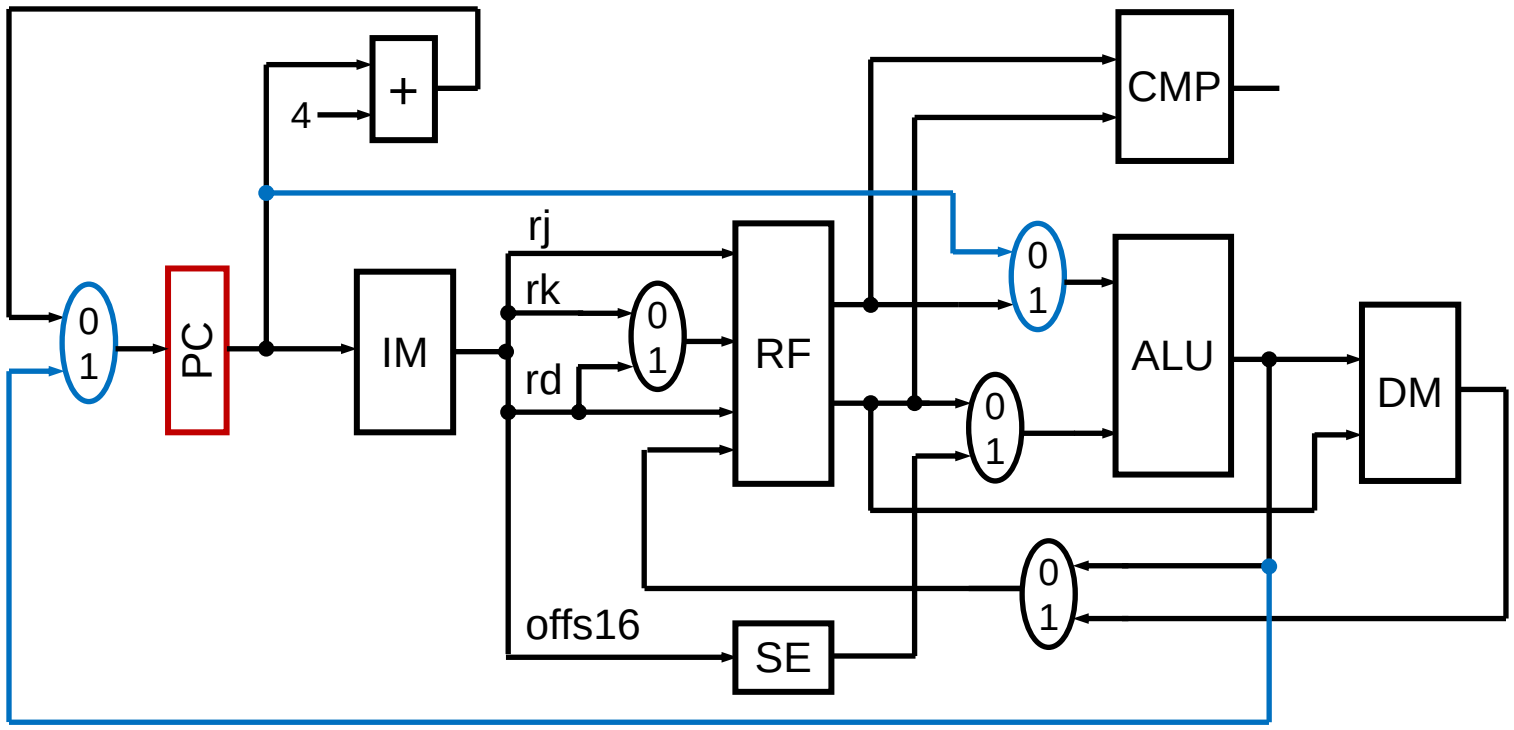
- bne rj, rd, offs16 # if (rj != rd) pc += SE({offs16, 2'b0})



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bne	0	1	0	1	1	1	offs16																rj					rd					

单周期数据通路：bne (2)

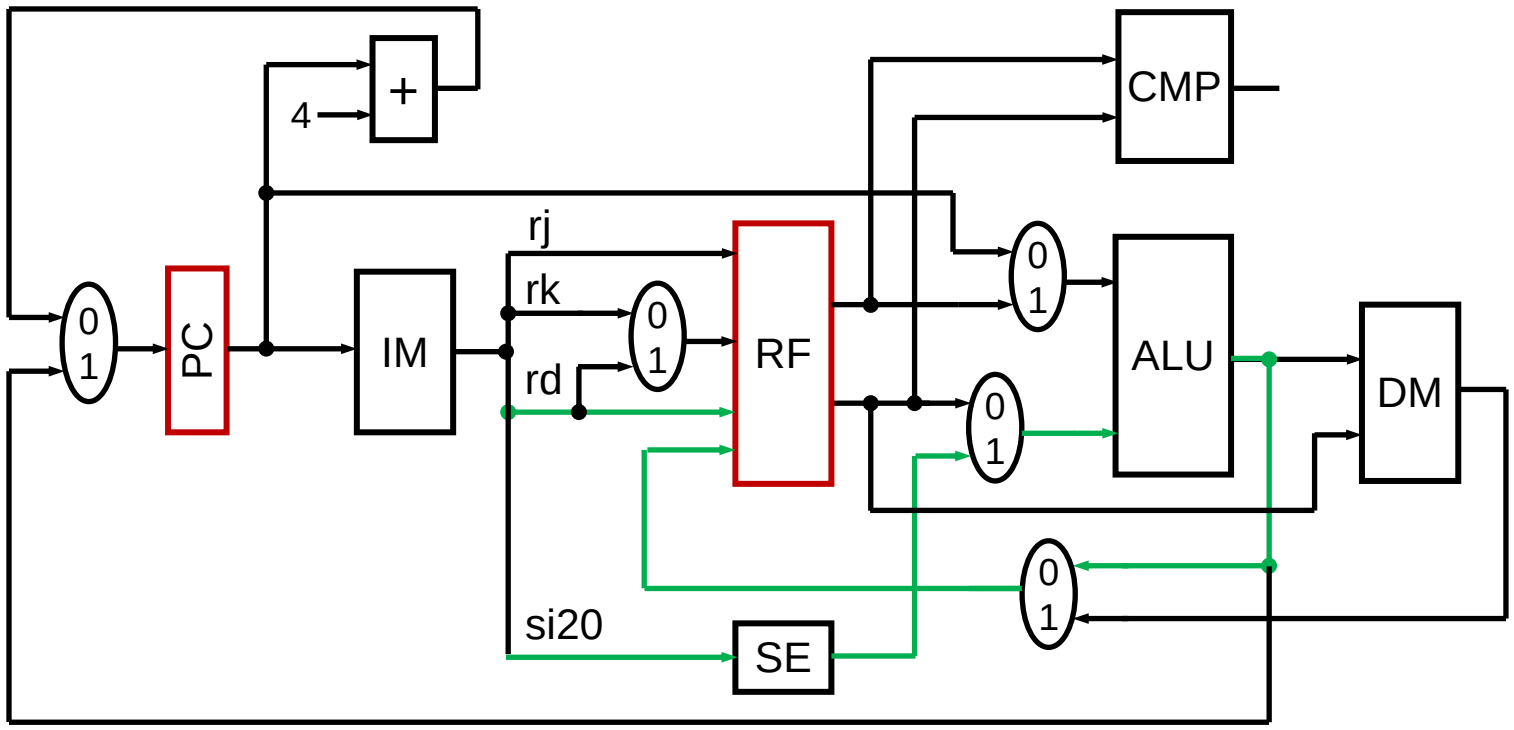
- bne rj, rd, offs16 # if (rj != rd) pc += SE({offs16, 2'b0})



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bne	0	1	0	1	1	1	offs16																rj					rd					

单周期数据通路：lu12i.w

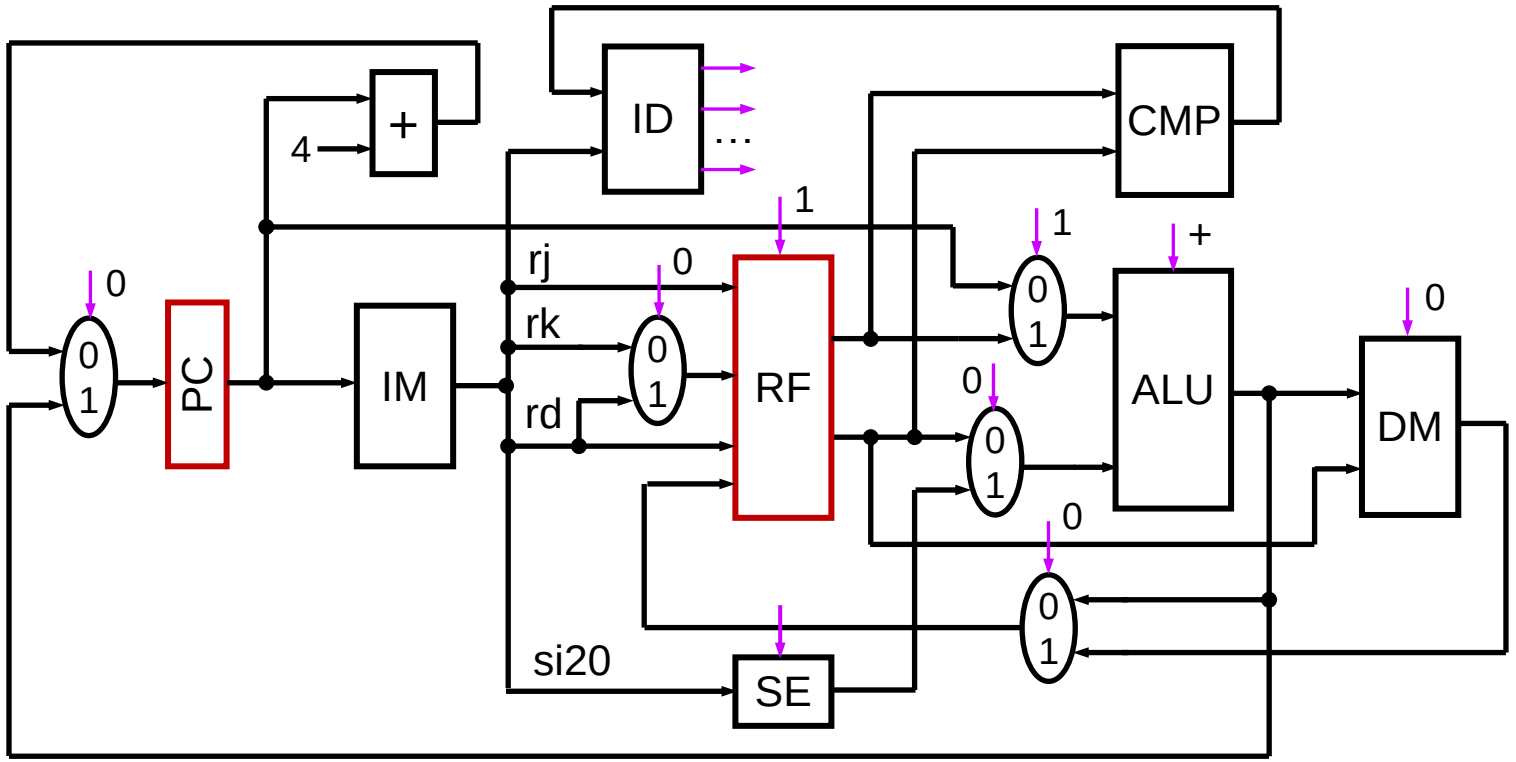
- lu12i.w rd, si20　　# rd = {si20, 12'b0}



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
lu12i.w	0	0	0	1	0	1	0	si20																				rd				

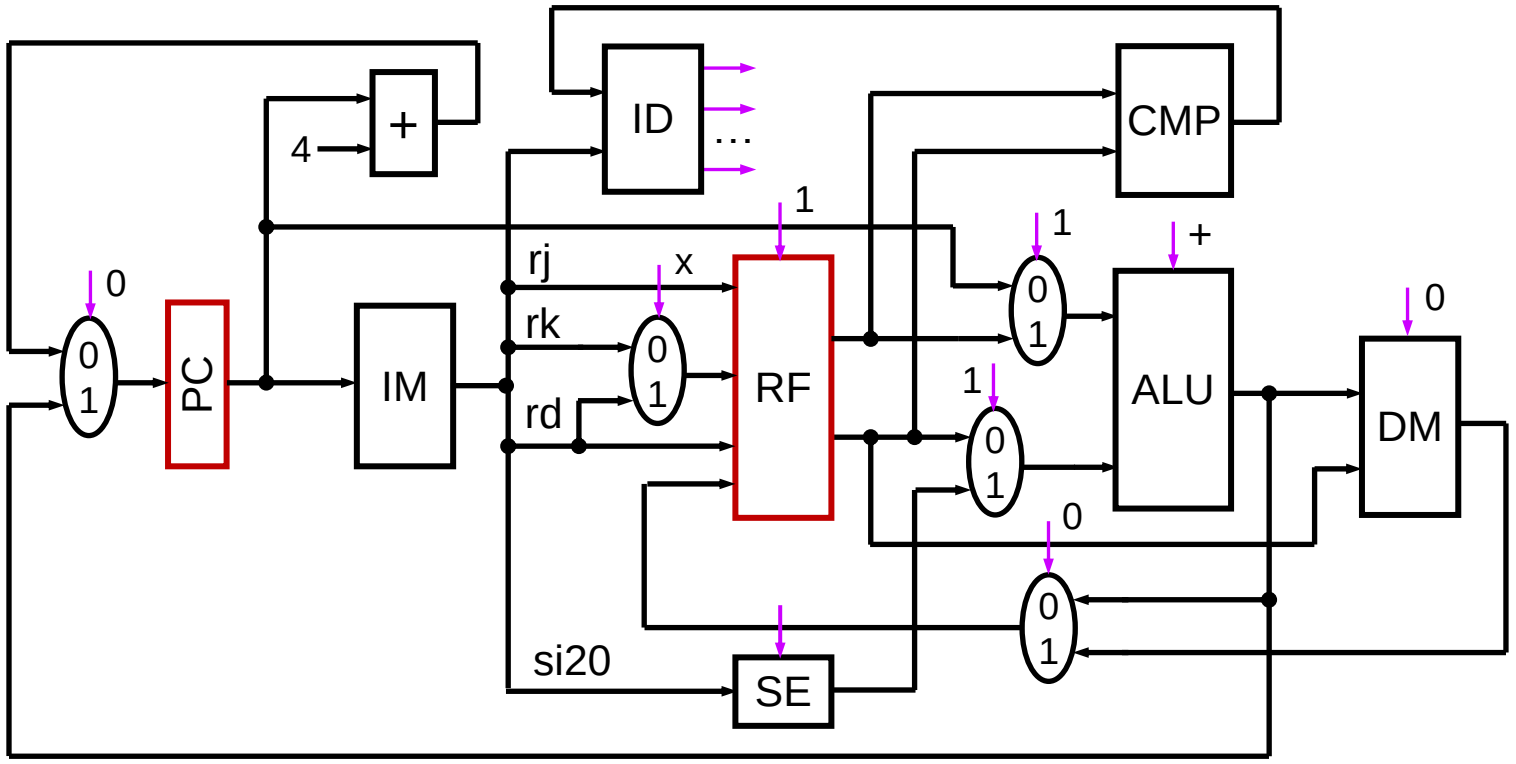
单周期控制信号：add.w

- add.w rd, rj, rk　　# rd = rj + rk



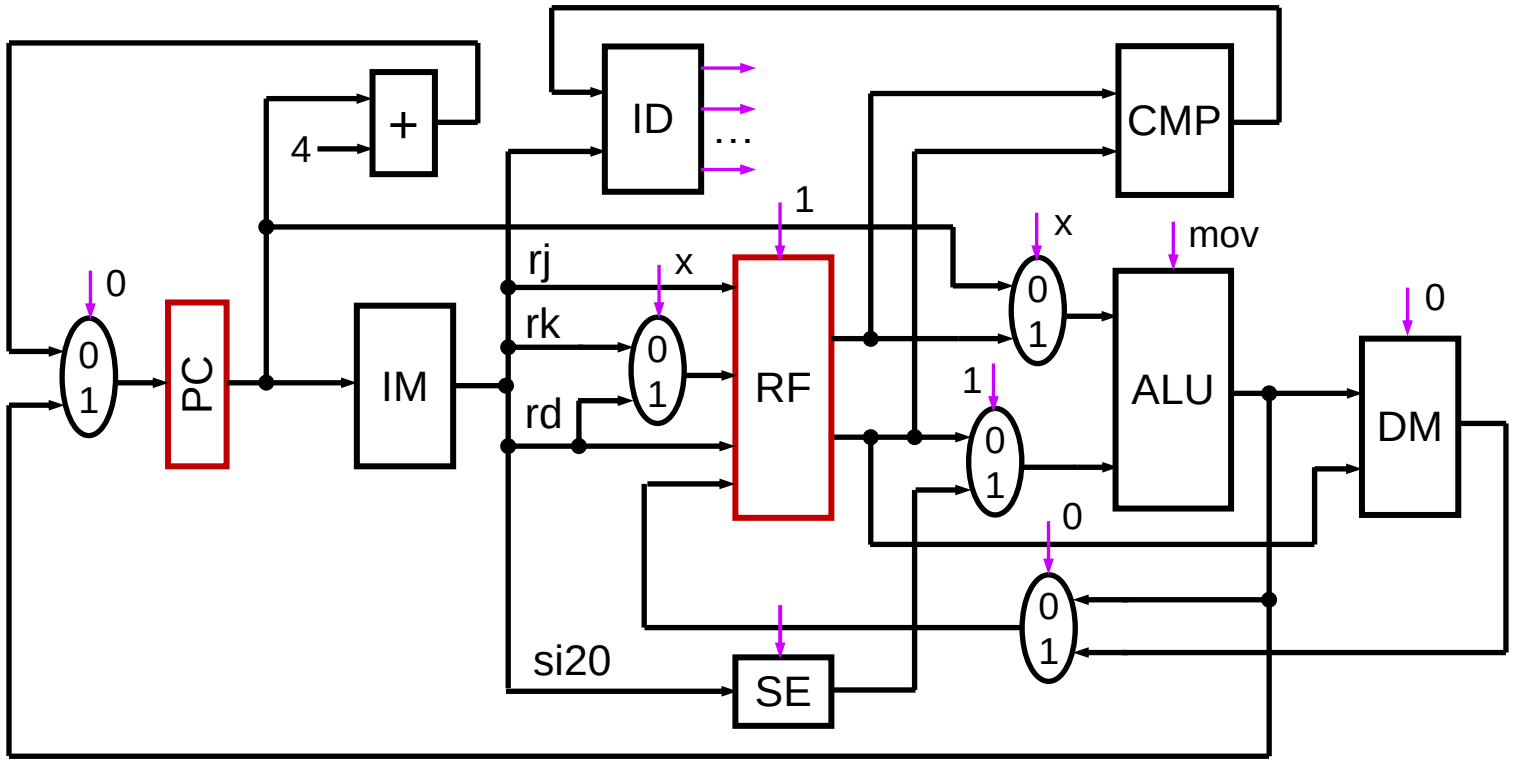
单周期控制信号：addi.w

- addi.w rd, rj, si12 # rd = rj + SE(si12)



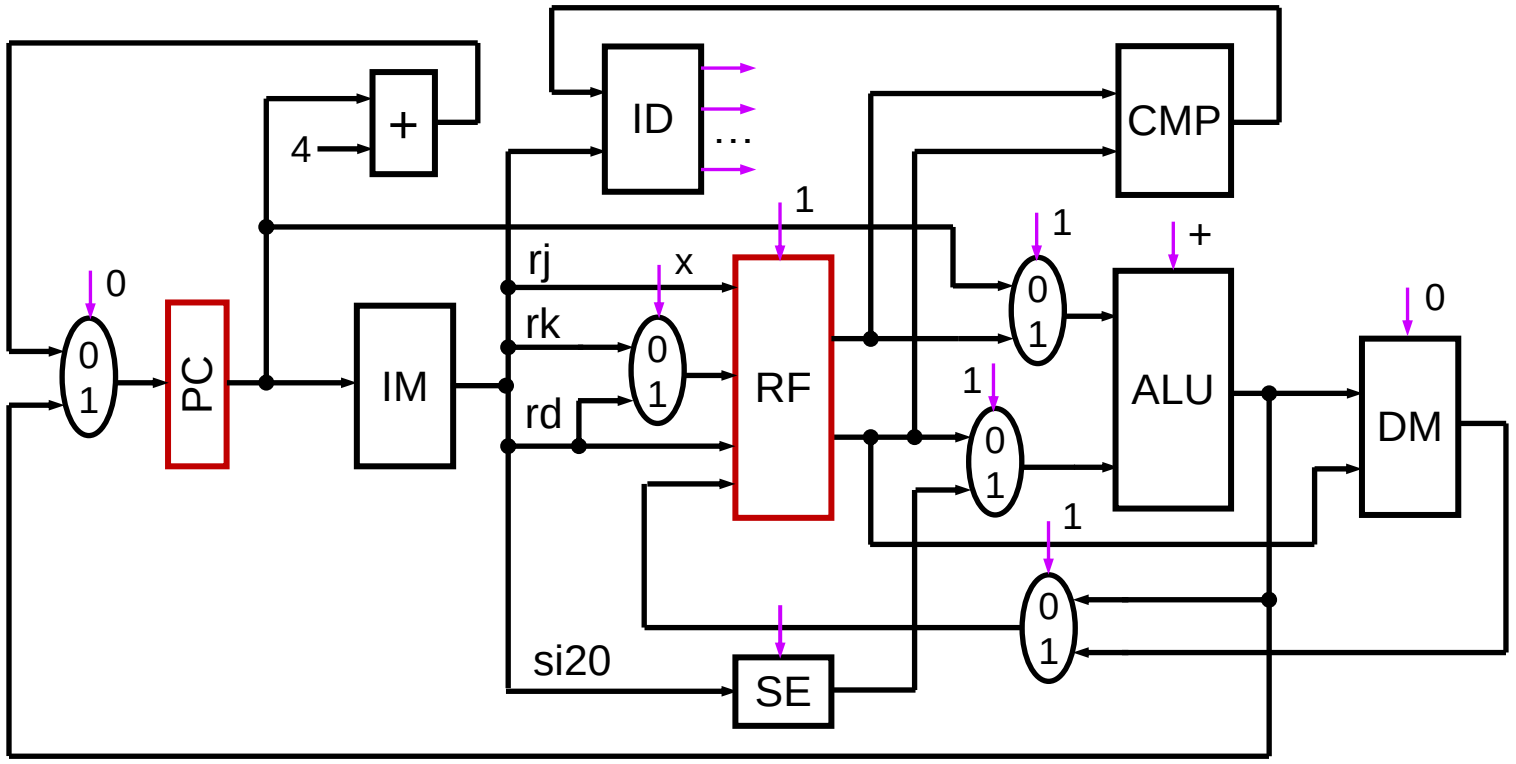
单周期控制信号：lu12i.w

- lu12i.w rd, si20 # rd = {si20, 12'b0}



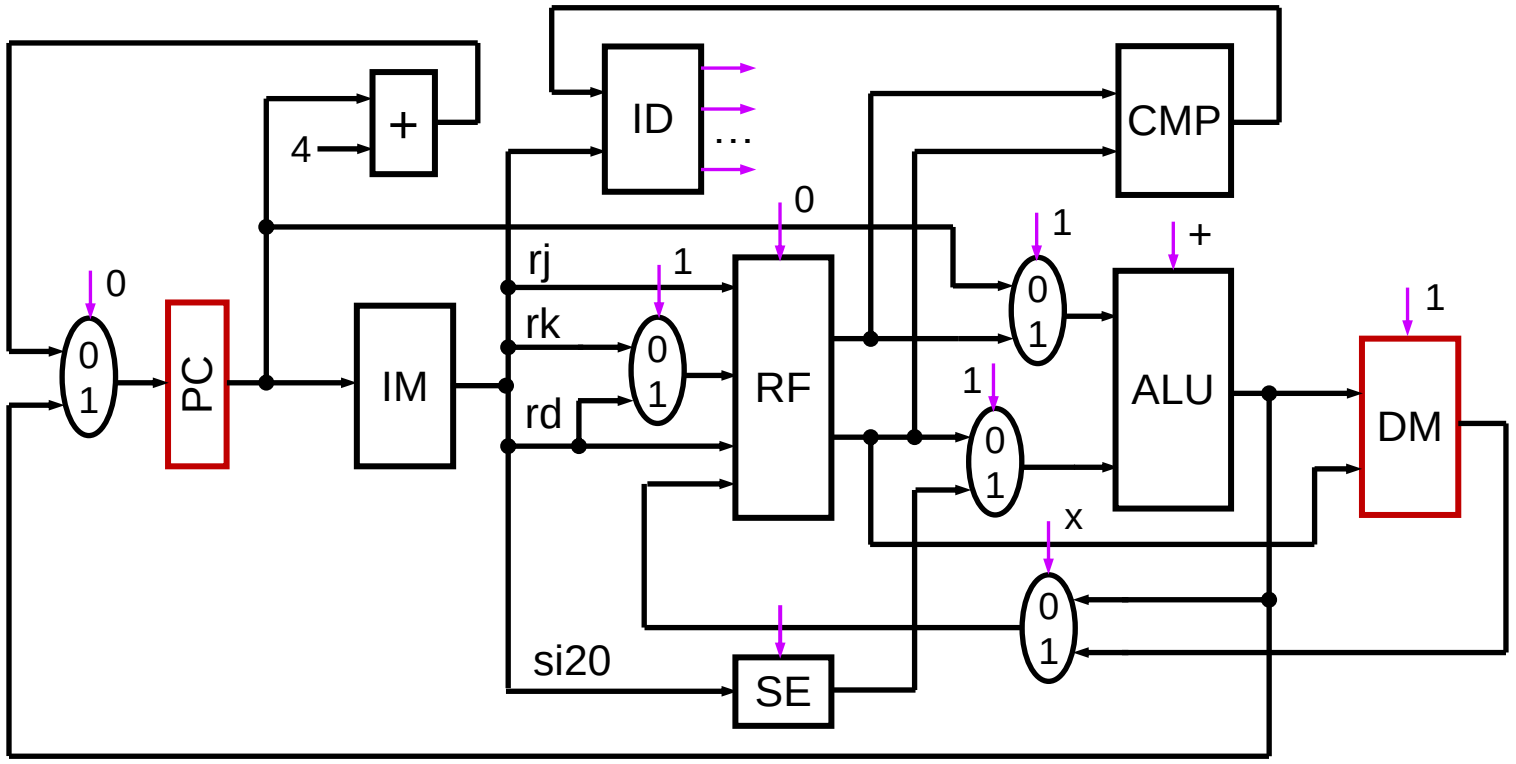
单周期控制信号：ld.w

- ld.w rd, rj, si12　　# rd = M[rj + SE(si12)]



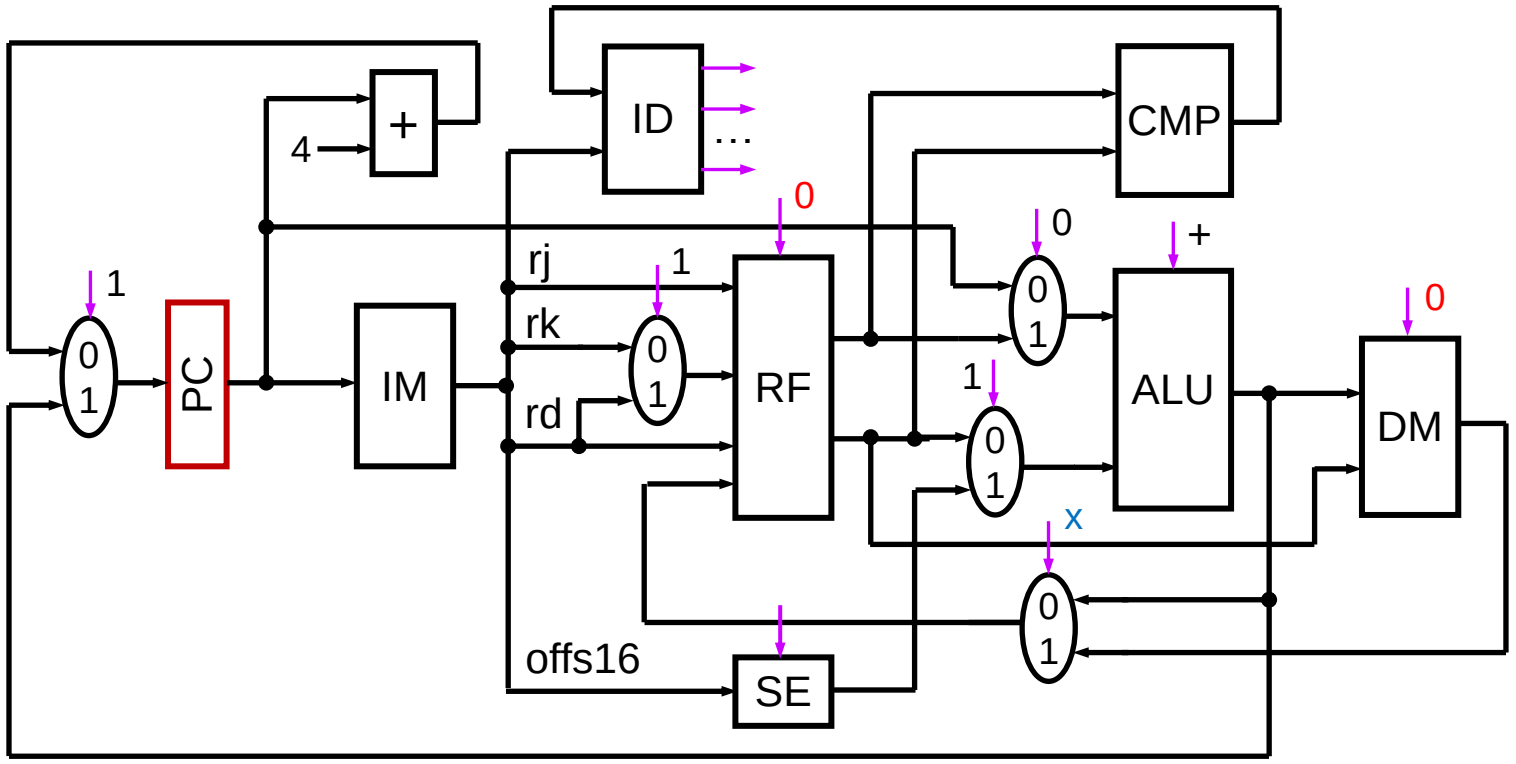
单周期控制信号：st.w

- st.w rd, rj, si12　　# M[rj + SE(si12)] = rd



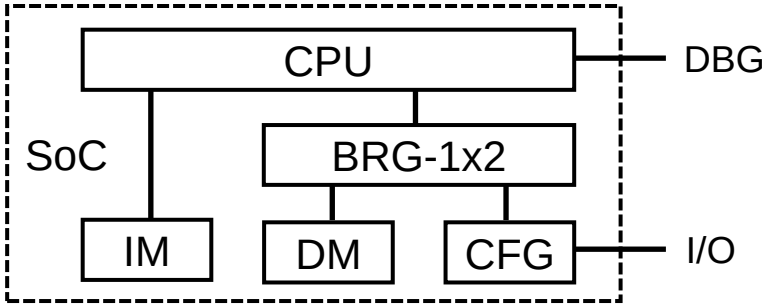
单周期控制信号：bne

- bne rj, rd, offs16 # if (rj != rd) pc += SE({offs16, 2'b0})



仿真和上板验证

- **构建片上系统 (System on Chip, SoC)**
  - IM，DM：instruction memory，data memory
  - CFG：configure registers，MMIO (Memory Mapped Input/Output)
  - BRG：bridge，连接CPU与DM和CFG
  - DBG：debug，调试接口
    - 仿真比对接口
    - 串口调试接口



- **参考资料**
  - CPU设计实战：LoongArch版：第 4 章 单周期CPU设计
    - https://bookdown.org/loongson/_book3/chapter-single-cycle-cpu.html#sec-verify-20insts-single-cycle-cpu
  - 实验开发环境：minicpu_env，mycpu_env
    - https://gitee.com/loongson-edu/cdp_ede_local

minicpu_env

```
|--miniCPU/              所实现CPU的RTL代码
|   |--minicpu_top.v     5条指令单周期CPU顶层模块
|   |--regfile.v         CPU中寄存器堆模块
|   |--tools.v           CPU中基本功能模块
|--func/                 功能验证测试程序
|   |--inst_ram.coe      测试程序对应上板用的二进制纯数据文件
|   |--inst_ram.mif      测试程序对应功能仿真用的二进制纯数据文件
|   |--inst_ram.txt      测试程序汇编代码说明
|--soc_verify/           所现的CPU的验证环境
    |--rtl/              验证用SoC设计代码目录
    |   |--soc_mini_top.v SoC的顶层文件
    |   |--CONFREG/      confreg模块，用于访问实验板上的LED灯、拨码开关等外设
    |   |--xilinx_ip/    定制的Xilinx IP，包含clk_pll、inst_ram
    |--testbench/        功能仿真验证平台
    |--run_vivado/       Vivado工程的运行目录
        |--constraints/  Vivado工程设计的约束
```

mycpu_env

```
|--myCPU/                    自己实现的CPU的RTL代码。
|--soc_verify/               自己实现的CPU的SoC系统验证环境
   |--soc_dram/              CPU对外连接distributed RAM接口时对应的验证环境。
   |   |--rtl/               SoC_Lite设计代码目录。
   |   |   |--soc_lite_top.v SoC_Lite的顶层文件。
   |   |   |--CONFREG/       confreg模块，用于访问CPU与开发板上数码管、拨码开关等外设。
   |   |   |--BRIDGE/        1×2的桥接模块， CPU的data sram接口同时访问confreg和data_ram。
   |   |   |--xilinx_ip/     定制的Xilinx IP，包含clk_pll、inst_ram、data_ram。
   |   |--testbench/         功能仿真验证平台。
   |   |   |--mycpu_tb.v     功能仿真顶层，该模块会抓取debug信息与golden_trace.txt进行比对。
   |   |--run_vivado/        Vivado工程的运行目录。
   |       |--constraints/   Vivado工程设计的约束。
   |       |--mycpu_dram_prj/ Vivado工程文件所在目录。
```

```
//mycpu_env\mycpu\mycpu_top.v

module mycpu_top (
    input  wire        clk,
    input  wire        resetn,
//inst sram interface
    output wire        inst_sram_we,
    output wire [31:0] inst_sram_addr,
    output wire [31:0] inst_sram_wdata,
    input  wire [31:0] inst_sram_rdata,
//data sram interface
    output wire        data_sram_we,
    output wire [31:0] data_sram_addr,
    output wire [31:0] data_sram_wdata,
    input  wire [31:0] data_sram_rdata,

//trace debug interface
    output wire [31:0] debug_wb_pc,
    output wire [ 3:0] debug_wb_rf_we,
    output wire [ 4:0] debug_wb_rf_wnum,
    output wire [31:0] debug_wb_rf_wdata
);
```

```
//mycpu_env\soc_verify\soc_bram\rtl\
  soc_lite_top.v

module soc_lite_top
# (parameter SIMULATION = 1'b0)
(
    input  wire        resetn,
    input  wire        clk,

//------gpio-------
    output wire [15:0] led,
    output wire [1 :0] led_rg0,
    output wire [1 :0] led_rg1,
    output wire [7 :0] num_csn,
    output wire [6 :0] num_a_g,
//    output wire [31:0] num_data,
    input  wire [7 :0] switch,
//    output wire [3 :0] btn_key_col,
//    input  wire [3 :0] btn_key_row,
    input  wire [1 :0] btn_step
);
```

|--func/	实验任务所用的功能验证测试程序。
| |--include/	功能验证测试程序共享的头文件所在目录。
| | |--sysdep.h	一些GCC通用的宏定义的头文件。
| | |--asm.h	LoongArch汇编需用到的一些宏定义的头文件，比如LEAF(x)。
| | |--regdef.h	LoongArch32 ABI下，32个通用寄存器的汇编助记定义。
| | |--cpu_cde.h	SoC_Lite相关参数的宏定义，如访问数码管的confreg的基址。
| | |--inst_test.h	各功能测试点的验证程序使用的宏定义头文件
| |--inst/	各功能测试点的汇编程序文件。
| | |--Makefile	子目录里的Makefile，会被上一级目录中的Makefile调用。
| | |--n*.S	各功能测试点的验证程序，汇编语言编写。
| |--obj/	功能验证测试程序编译结果存放目录
| | |--*	详见后面小节的说明。
| |--start.S	功能验证测试的引导代码及主函数。
| |--Makefile	编译功能验证测试程序的Makefile脚本
| |--bin.lds	编译bin.lds.S得到的结果，可被make reset命令清除
| |--convert.c	生成coe和mif文件的处理工具的C程序源码

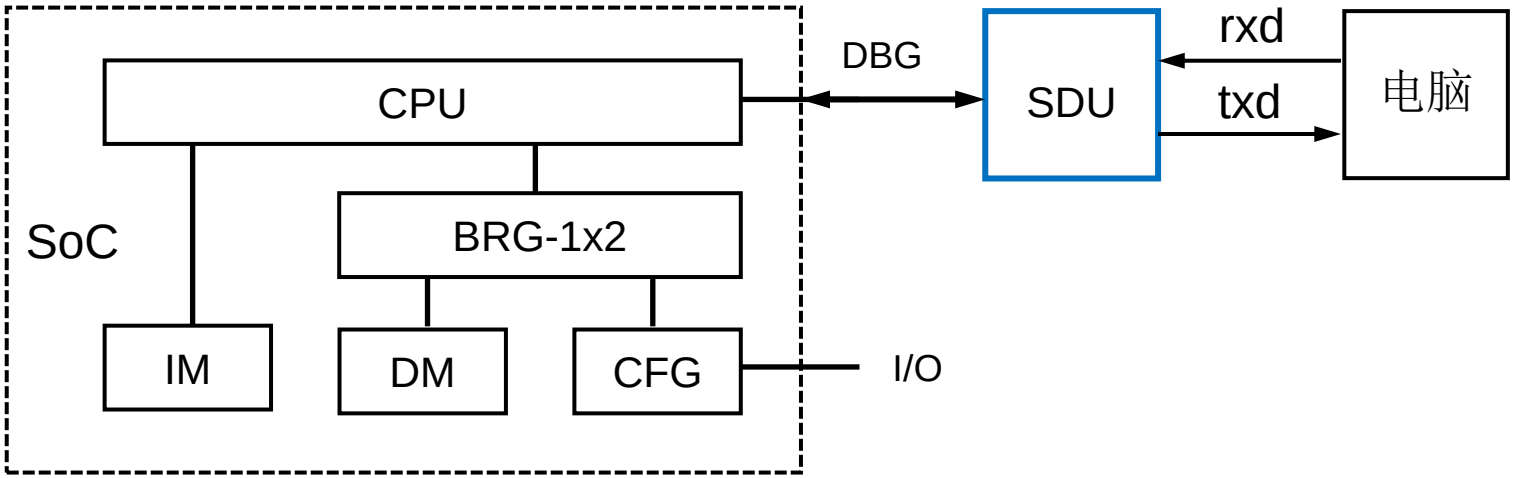
基于trace比对的仿真验证

- 先用功能正确的refCPU仿真，写寄存器堆时将PC、写地址、写数据等信息记录下来，记为golden_trace
- 后用待验证的myCPU仿真，写寄存器堆时与golden_trace中的相关信息进行比对，若不一样，则报错并停止仿真

```
|--gettrace/                生成参考trace的部分。
|   |--src/                 设计代码目录。
|   |   |--tb_top.v         仿真顶层，该模块会抓取debug信息生成到golden_trace.txt中。
|   |   |--soc_lite_top.v   SoC_Lite的顶层文件。
|   |   |--refCPU/          产生比对trace的参考处理器核设计。
|   |   |--CONFREG/         confreg模块，用于访问CPU与开发板上数码管、拨码开关等外设。
|   |   |--BRIDGE/          1×2的桥接模块，CPU的data sram接口同时访问confreg和data_ram。
|   |--gettrace.xpr         Vivado工程文件。
|   |--golden_trace.txt     运行func测试程序所生成的参考trace。需自行生成。
```

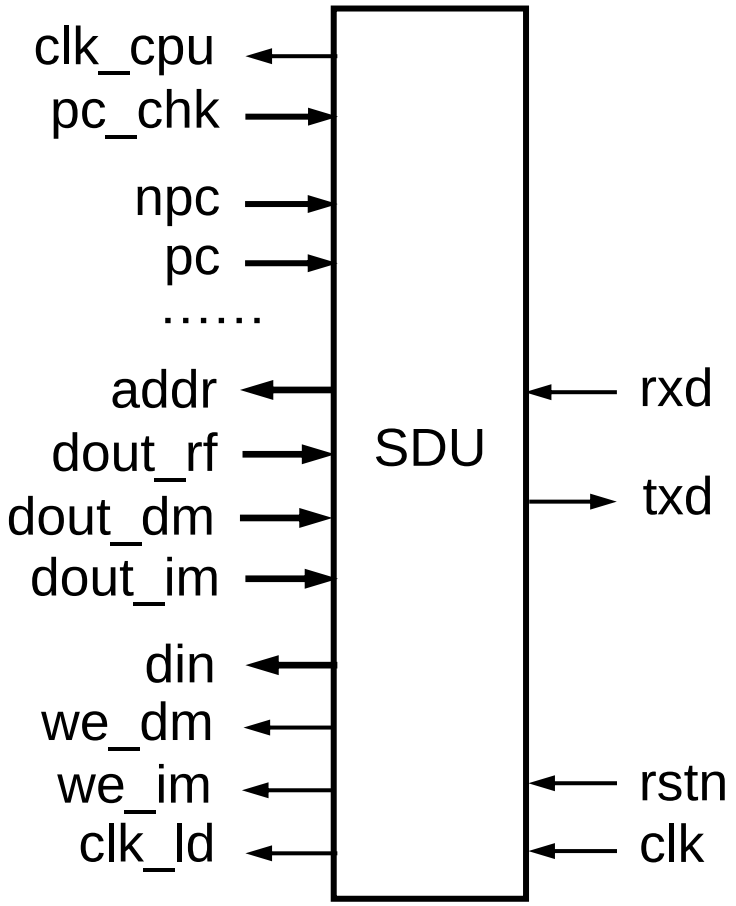
串行调试单元

- **Serial Debug Unit (SDU)，利用电脑串口对SoC进行上板验证调试**
  - 控制CPU运行方式：单步或者支持断点的连续运行
  - 查看SoC运行状态：数据通路寄存器和指令/数据存储器的内容
  - 加载指令/数据存储器：初始化IM和DM



调试接口信号

- **控制CPU运行方式**
  - 工作时钟：clk_cpu
  - 当前执行指令指针：pc_chk
- **查看SoC运行状态**
  - 数据通路：npc, pc, ir, ctl, a, b, imm, y, mdr
  - 寄存器堆：addr, dout_rf
  - 存储器：addr, dout_dm, dout_im
- **加载指令/数据存储器**
  - 写地址：addr
  - 写数据：din
  - 写使能：we_dm, we_im
  - 写时钟：clk_ld



调试命令

- **控制运行方式**
  - T：S**t**ep, CPU单步运行，运行1个时钟周期即停止
  - B：**B**reakpoint, 设置/删除/查看断点，最多可有2个断点
  - G：**G**o, CPU连续运行，遇到断点或收到停止命令H则停止
  - H：**H**alt, 停止CPU运行
- **查看运行状态**
  - P： Data**p**ath, 查看数据通路状态
  - R：**R**egister File, 查看寄存器堆内容
  - D：**D**ata Memory, 查看数据存储器内容
  - I：**I**nstruction Memory, 查看指令存储器内容
- **加载存储器**
  - LI：**L**oad **I**nstruction, 将程序加载至指令存储器
  - LD：**L**oad **D**ata, 将数据加载至数据存储器

调试命令 (续1)

- **P：Datapath, 查看数据通路状态**
  - 依次输出数据通路中npc, pc, ir, ctl, a, b, imm, y, mdr等内容
  - 电脑端显示格式：NPC = xxxx-xxxx PC = yyyy-yyyy ……
- **R：Register File, 查看寄存器堆内容**
  - addr从0递增，依次输出dout_rf，即寄存器堆32个寄存器内容
  - 电脑端显示格式：R0 = 0000-0000 R1= xxxx-xxxx ……
- **D [a]：Data Memory, 查看数据存储器内容**
  - a为16进制起始字地址，缺省时为上次查看结束地址 (复位时为0)
  - addr从a递增，依次输出dout_dm，即数据存储器地址a开始8个字
  - 电脑端显示格式： D-xxxx-xxxx: yyyy-yyyy zzzz-zzzz ……
- **I [a]：Instrction Memory, 查看指令存储器内容**
  - 其他同上，依次输出dout_im，即指令存储器地址a开始8个字
  - 电脑端显示格式： I-xxxx-xxxx: yyyy-yyyy zzzz-zzzz ……

调试命令 (续2)

- **T：Step，CPU单步运行**
  - 输出1个clk_cpu周期后，自动执行P命令，显示数据通路状态
- **B [a]：Breakpoint，设置/删除/查看断点**
  - a为可选的16进制断点，B：显示所有断点，最多可有2个断点
  - B a：如果当前断点中不存在a，且未满2个断点，则设置a断点；如果当前断点中已有a，则删除之；无论如何均显示所有断点
- **G：Go，CPU连续运行**
  - 输出1个clk_cpu周期后，检查pc_chk是否等于断点，以及是否接收到停止命令H
  - 如果结果均为否，则重复上述过程，否则执行P命令，显示数据通路状态
- **H：Halt，CPU停止运行**
  - **如果执行G命令时，CPU运行不止，H命令可强制停止**

调试命令 (续3)

- **LD 文本文件：Load Data, 加载数据存储器**
  - 数据存储在文本文件中，每行对应一个32位的16进制数据字
  - addr从0开始递增，din依次为文件一行数据，we_dm = 1，clk_ld产生一个脉冲，写入数据存储器一个数据，直至文件结束（空白行）
  - 全部写入结束后，电脑端输出：Finish
- **LI 文本文件：Load Instruction, 加载指令存储器**
  - 机器码程序存储在文本文件中，每行对应一条16进制指令码
  - addr从0开始递增，din依次为文件一行数据，we_im = 1，clk_ld产生一个脉冲，写入指令存储器一条指令，直至文件结束（空白行）
  - 全部写入结束后，电脑端输出：Finish

注意：运行新加载程序或使用新加载数据前务必复位系统！

实验要求

- **设计单周期LA32R CPU，构建SoC并上板验证**
  - 指令和数据存储器均为8K x 32位的分布式存储器
  - 查看电路资源和性能
  - 运行指令测试程序
  - 运行排序程序
- **选项：增添乘/除法运算指令**
  - mul.w, mulh.w, mulh.wu, div.w, div.wu, mod.w, mod.wu
  - 查看电路资源和性能
  - 运行指令测试程序

The End