计算机学院学科基础课

计算机组成原理

单周期数据通路
一般性设计方法

高小鹏

北京航空航天大学计算机学院

必需的部件

- PC、NPC、IM

名称	功能	输入	输出
PC	指向指令存储器	NPC[31:2]	PC[31:2]
		Clk	
		Reset	
NPC	计算下一个PC值	Imm[15:0]	NPC[31:2]
		Br	
		Zero	
IM	指令存储器	PC[31:2]	IM[31:0]

部件描述与HDL建模

- 示例：PC

4.1.1. 基本描述

PC 模块的主要功能是将 NPC[31:0]的值保存并输出。PC 的各种取值将根据所执行的指令、外部状态(中断)及处理器控制器的当前状态的不同，由数据通路其他部件生成。

4.1.2. 模块接口

表 4-1 PC 接口信号定义

信号名	方向	描述
Clk	I	MIPS-C 处理器时钟
Reset	I	复位信号
NextPC[31:0]	I	下一个 PC 值
PCWr	I	PC 写使能
PC[31:0]	O	PC 输出

部件描述与HDL建模

- 示例：PC

4.1.3. 功能定义

PC 模块的核心是一个寄存器。该寄存器在 PCWr 有效时将 NextPC[31:0]锁存并输出。

表 4-2 PC 功能需求定义

编号	功能名称	功能描述
1	初始化	当 Reset 信号有效后，PC 输出 0xBFC00000。
2	PC 更新	当时钟上升沿到来时，PCWr 有效则将 NPC 写入 PC 内部，并且从 PC 端口输出。

部件描述与HDL建模

- 示例：PC

```
16  `timescale  1ns/1ns
17
18  module PC( CLK_I, Reset_I, Addr_I, PCWrite_I, PC_O ) ;
19      input               CLK_I;          // system clock
20      input               Reset_I;        // reset signal
21      input [31:0]        Addr_I;         // next PC
22      input               PCWrite_I;      // write enable
23      output[31:0]        PC_O;           // PC output
24
25
26      /*  internal reg and wire   */
27      reg [31:0]  addr ;                  // latch the address
28
29      /* read register */
30      assign PC_O = addr;
31
32      always@ ( posedge CLK_I or posedge Reset_I )
33      begin
34          if(Reset_I)
35              addr <= 'hBFC00000;
36          else if( PCWrite_I )
37              addr <= Addr_I;
38      end
39
40  endmodule
```

必需的部件：RF

- RF：寄存器文件
  - 32个寄存器；0号寄存器永远为0

名称	功能	输入	输出
RF	寄存器文件	RS1[4:0]	RD1[31:0]
		RS2[4:0]	RD2[31:0]
		RD[4:0]	
		WData[31:0]	
		RegWr	
		Clk	
		Reset	

必需的部件：ALU、DM

- ALU：各类运算、地址计算
- DM：数据存储器

名称	功能	输入	输出
ALU	加/减/或	A[31:0]	ALU[31:0]
		B[31:0]	
DM	数据存储器	Ad[31:2]	DM[31:0]
		WrData[31:0]	
		DMWr	
		Clk	

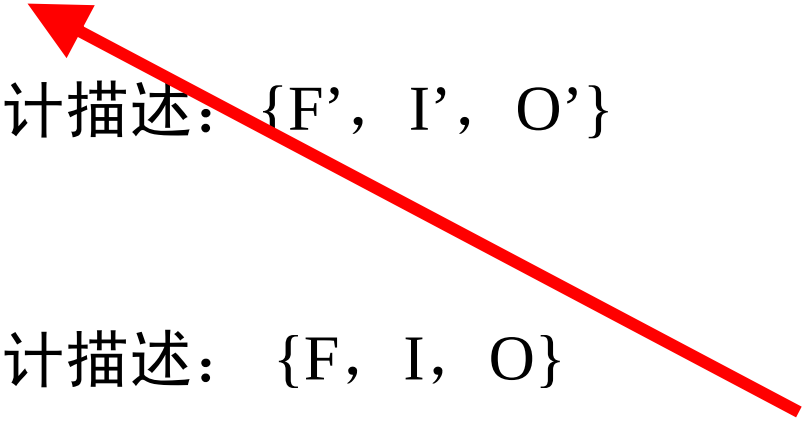
数据通路设计表格

- 表格记录了部件输入端的输入来源
  - 忽略控制类信号
  - 只保留数据类信号

指令	NPC	PC	IM	RF		ALU		DM
				WData	RD	A	B	

单指令数据通路构造的一般性方法

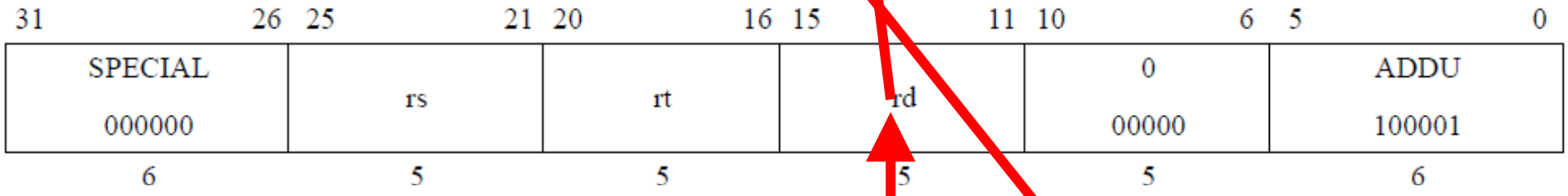
- S1：阅读每条指令→发现所有的新增需求
- S2：对每个新增需求（2种处理方法）
  - 合并至已有部件
    - 修改已有部件设计描述：{F’，I’，O’}
  - 需要新增部件
    - 建立新增部件设计描述：{F，I，O}
- S3：对每个部件设置输入来源



原则：
- 来源相同/相近
- 目的相同/相近

ADDU

指令	NPC	PC	IM	RF		ALU		DM
				WData	RD	A	B	
ADDU		NPC	PC	ALU	IM[15:11]	RF.RD1	RF.RD2	



Operation:

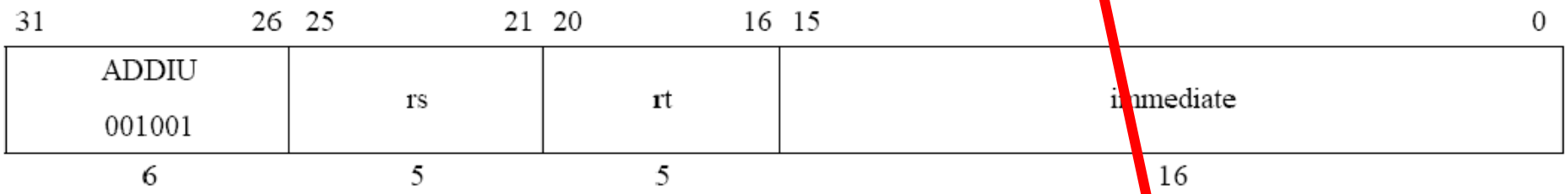
```
temp ← GPR[rs] + GPR[rt]
GPR[rd] ← temp
```

RTL

R[rd]←R[rs]+R[rt]
PC←PC+4

ADDIU

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU									



Operation:

```
temp ← GPR[rs] + sign_extend(immediate)
GPR[rt]← temp
```

RTL

R[rt]←R[rs]+sign_ext(imm16)

PC←PC+4

S_EXT：新增的部件

- S_EXT：有符号扩展

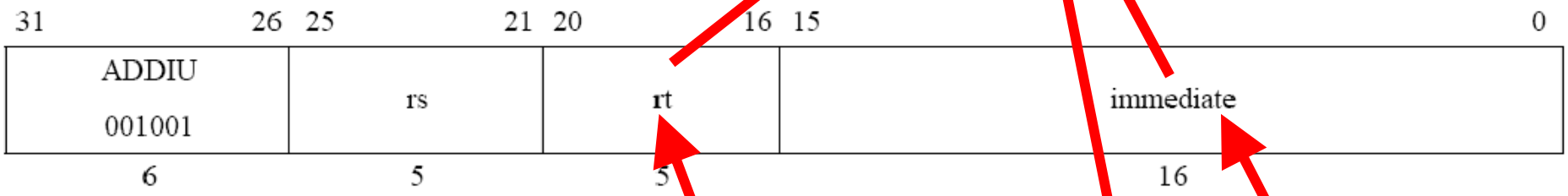
名称	功能	输入	输出
S_EXT	将16位补码扩展为32位补码	Imm[15:0]	S_EXT[31:0]

HDL建模：sign_ext.v

```
module SignEXT( Imm, S_Ext ) ;
    。 。 。
    assign S_Ext = {16{Imm[15]}, Imm} ;
end module
```

ADDIU

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	



Operation:

```
temp ← GPR[rs] + sign_extend(immediate)
GPR[rt]← temp
```

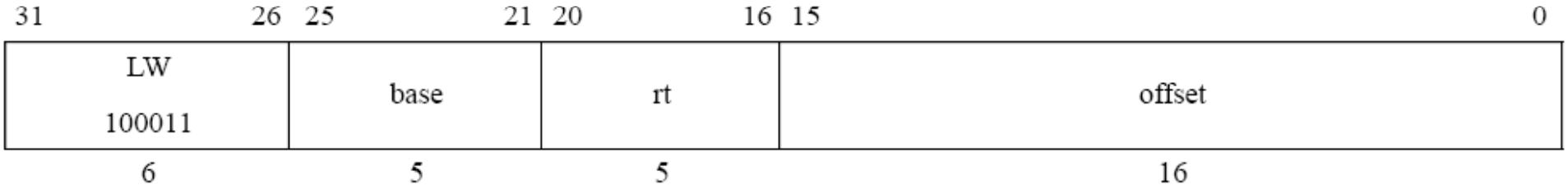
RTL

R[rt]←R[rs]+sign_ext(imm16);

PC←PC+4

LW

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW									

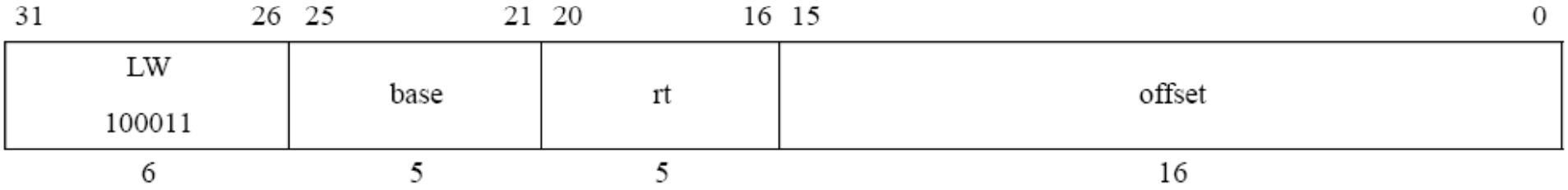



```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 then
    SignalException(AddressError)
endif
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
memword← LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
GPR[rt]← memword
```

????

LW

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW									

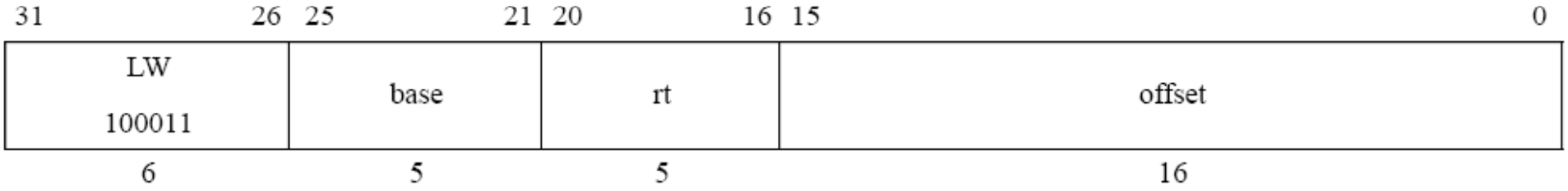



```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 then
    SignalException(AddressError)
endif
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
memword← LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
GPR[rt]← memword
```

vAddr：虚拟地址
全部忽略！

LW：改写Operation

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW									

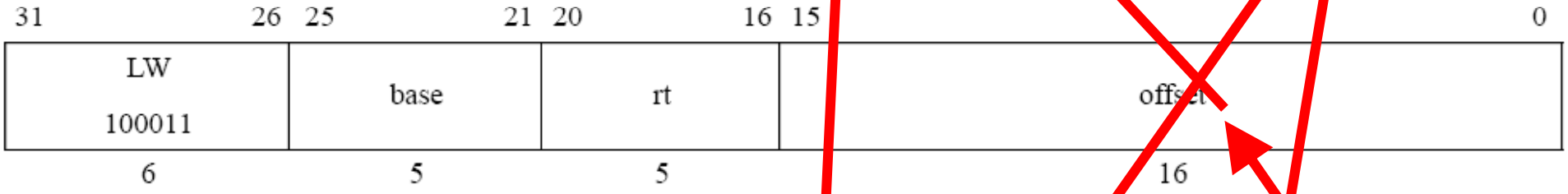



```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 then
    SignalException(AddressError)
endif
(pAddr, CCA)← AddressTranslation (vAddr, DATA, LOAD)
memword← LoadMemory (CCA, WORD, pAddr, vAddr, DATA)
GPR[rt]← memword
```

```
Addr ← sign_extend(offset) + GPR[base]
memword ← Memory[Addr]
GPR[rt] ← memword
PC ← PC + 4
```

LW：改写Operation

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	




```
Addr ← sign_extend(offset) + GPR[base]
memword ← Memory[Addr]
GPR[rt] ← memword
PC ← PC + 4
```

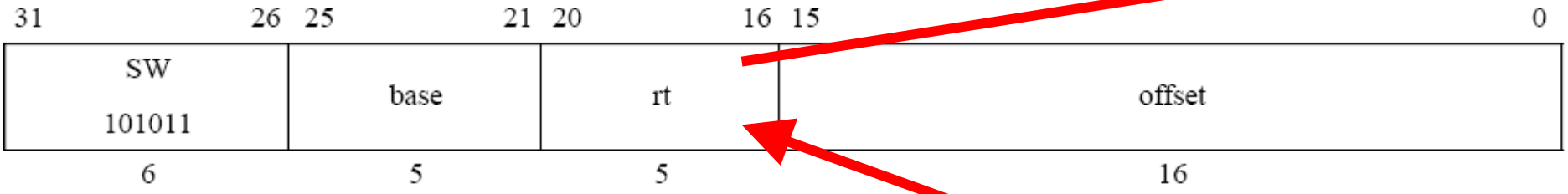
RTL

R[rt]←MEM[R[rs]+sign_ext(imm16)]

PC←PC+4

SW

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
SW		NPC	PC			IM[15:0]	RF.RD1	S_EXT	RF.RD2



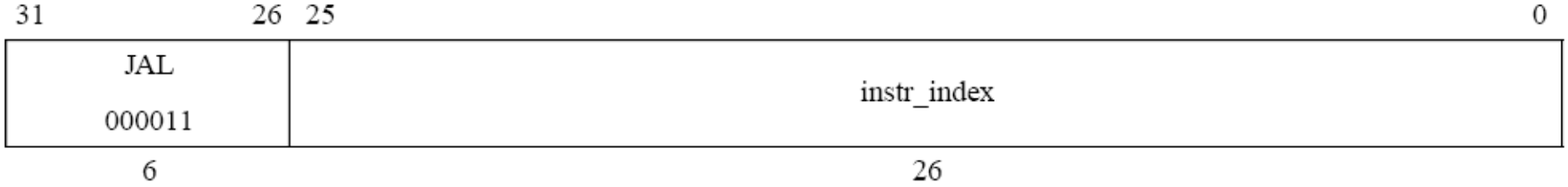
RTL描述

MEM[R[rs]+sign_ext(imm16)] ←R[rt]

PC←PC+4

JAL：跳转并链接

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
SW		NPC	PC			IM[15:0]	RF.RD1	S_EXT	RF.RD2
JAL									



Operation:

```
I:   GPR[31]← PC + 8
I+1:PC      ← PC_GPRLEN-1..28 || instr_index || 0^2
```

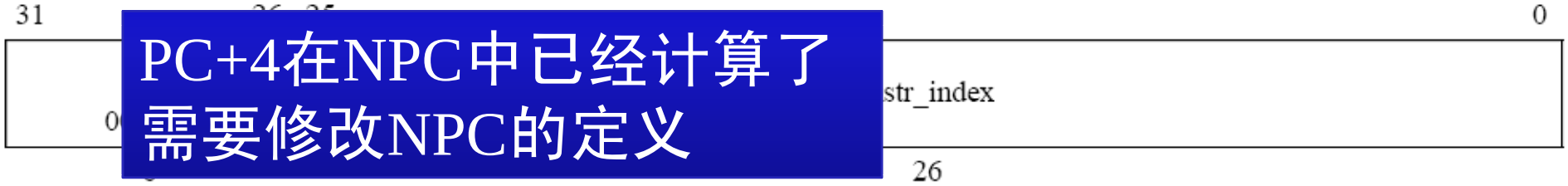
RTL描述

R[31] ← PC + 4

PC ← PC[31:28] || instr_index || 00

JAL：跳转并链接

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
SW		NPC	PC			IM[15:0]	RF.RD1	S_EXT	RF.RD2
JAL									



PC+4在NPC中已经计算了
需要修改NPC的定义

Operation:

PC计算方法发生变化
需要修改NPC的定义

RTL描述

R[31] ← PC + 4

PC ← PC[31:28] || instr_index || 00

修改：NPC的部件定义、HDL建模

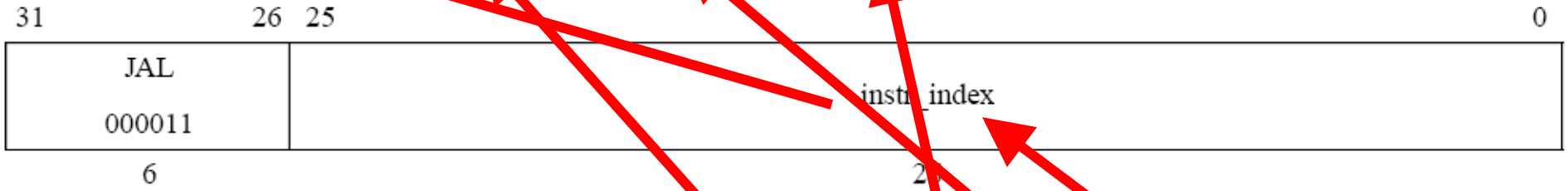
- 需要修改输入：Imm[15:0] → Imm[25:0]
- 需要增加输出：PC4[31:0]
- Br不合适了，用更通用的Op[1:0]代替
  - 3个功能，至少需要2位控制信号
  - Op：控制器要根据指令输出对应的编码
- 需要重新修改：npc.v

Op编码	编码含义
00	PC + 4
01	BEQ指令
10	JAL指令
11	未定义

名称	功能	输入	输出
NPC	1、计算下一个PC值 2、输出PC+4	Imm[25:0]	NPC[31:2]
		Op[1:0]	PC4[31:0]
		Zero	

JAL：跳转并链接

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
SW		NPC	PC			IM[15:0]	RF.RD1	S_EXT	RF.RD2
JAL	IM[25:0]	NPC.NPC	PC	NPC.PC4	0x1F				



Operation:

```
I:   GPR[31]← PC + 8
I+1:PC      ← PC_GPRLEN-1..28 || instr_index || 0^2
```

RTL描述

R[31] ← PC + 4

PC ← PC[31:28] || instr_index || 00

多指令数据通路合并

指令	NPC	PC	IM	RF		S_EXT	ALU		DM
				WData	RD		A	B	
ADDU		NPC.NPC	PC	ALU	IM[15:11]		RF.RD1	RF.RD2	
ADDIU		NPC.NPC	PC	ALU	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
LW		NPC.NPC	PC	DM	IM[20:16]	IM[15:0]	RF.RD1	S_EXT	
SW		NPC.NPC	PC			IM[15:0]	RF.RD1	S_EXT	RF.RD2
JAL	IM[25:0]	NPC.NPC	PC	NPC.PC4	0x1F				
合并	IM[25:0]	NPC.NPC	PC	ALU| DM | NPC.PC4	IM[15:11]| IM[20:16]| 0x1F	IM[15:0]	RF.RD1	RF.RD2| S_EXT	RF.RD2

- 合并：垂直方向归并，去除相同项
- 增加MUX：输入源多余1个的需设置MUX
  - 需要MUX部件描述及HDL建模
  - MUX控制信号由控制器产生

数据通路设计的一般性方法

单指令数据通路构造

```
for each 指令
    for each 新增需求
        case 可以合并至已有部件：
            修改部件设计描述、HDL建模：{F′，I′，O′}
        case 需要新增部件：
            建立新部件设计描述、HDL建模：{F，I，O}
            增加新部件

    for each 部件
        设置输入来源
```

多数据通路综合

```
按垂直方向合并数据通路，并去除相同项
for each 输入来源多余1个的输入端
    部署1个MUX(MUX的输入规模为输入来源数)
    MUX设计定义、HDL建模
```

系统实现

HDL建模：连接所有的部件及所有的MUX

数据通路设计的一般性方法

固定复杂度(单指令，对每条指令理解正确)

```
for each 指令
    for each 新增需求
        case 可以合并至已有部件：
            修改部件设计描述、HDL建模：{F′，I′，O′}
        case 需要新增部件：
            建立新部件设计描述、HDL建模：{F，I，O}
            增加新部件

    for each 部件
        设置输入来源
```

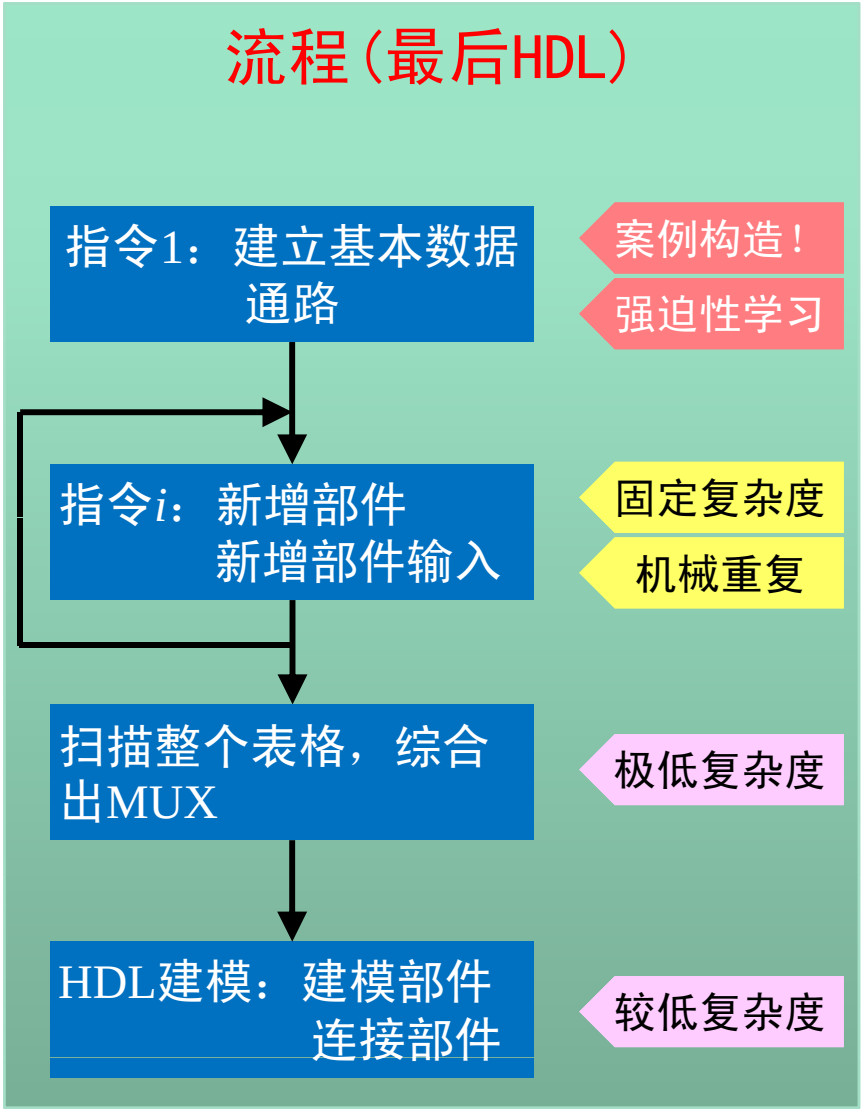
极低复杂度

```
按垂直方向合并数据通路，并去除相同项
for each 输入来源多余1个的输入端
    部署1个MUX(MUX的输入规模为输入来源数)
    MUX设计定义、HDL建模
```

较低复杂度

HDL建模：连接所有的部件及所有的MUX

数据通路设计的一般性方法



VerilogHDL工程注意事项

- 文件层次清晰
- 文件名易懂
- 模块名易懂
- 端口命名易懂
- 要有注释