

计算机科学与技术专业课程

计算机组成

CPU形式建模综合方法 ——单周期数据通路

高小鹏

北京航空航天大学计算机学院
系统结构研究所

必需的部件

- PC、NPC、IM
- Br: 1~B类指令

Q
为什么IM省略了地址最低2位?

名称	功能	输入	输出
PC	指向指令存储器	NPC[31:0]	PC[31:0]
		Clk	
		Reset	
NPC	计算下一个PC值	PC[31:0]	NPC[31:0]
		Imm[15:0]	
		Br	
		Zero	
IM	指令存储器	Ad[31:2]	DO[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
<u>Clk</u>	I	MIPS-C 处理器时钟
Reset	I	复位信号
<u>NextPC[31:0]</u>	I	下一个 PC 值
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 Clk上升沿 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, 将NextPC 写入 PC 内部, 并且从 PC 端口输出。

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	A1 [4:0]	RD1 [31:0]
		A2 [4:0]	RD2 [31:0]
		A3 [4:0]	
		WD [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]	C[31:0]
		B[31:0]	
		Op[X:0]	
DM	数据存储器	Ad[31:2]	DO[31:0]
		DIn[31:0]	
		DMWr	
		Clk	

数据通路设计表格

- 基于部件的数据通路设计思路：建立功能部件之间的输入信号/输出信号的连接关系
- 表格：记录了部件**输入信号**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	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B	Ad	Din

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: 对每个部件的输入信号设置输入来源

原则:

- ◆ 来源相同/相近
- ◆ 目的相同/相近

ADDU

指令	NPC		PC	IM	RF				ALU		D
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B	Ad
addu	PC.PC		NPC.NPC	PC.PC							

□ NPC.PC: 固定连接

- ◆ 为了计算下一个地址，必须输入PC值

□ PC.NPC: 固定连接

- ◆ 由于NPC承担了下一个PC值的计算，因此PC只是存储NPC的计算结果而已

□ IM.Ad: 固定连接

- ◆ IM的地址只能来自PC

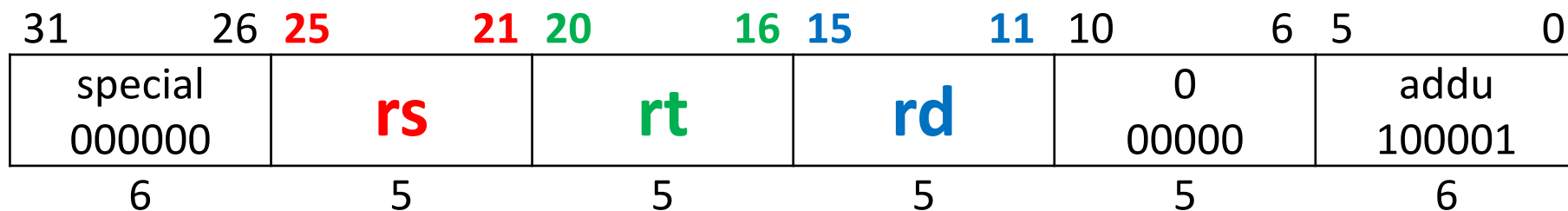
RTL

$R[rd] \leftarrow R[rs] + R[rt]$

$PC \leftarrow PC + 4$

ADDU

指令	NPC		PC	IM	RF				ALU	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B
addu	PC.PC		NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11]			



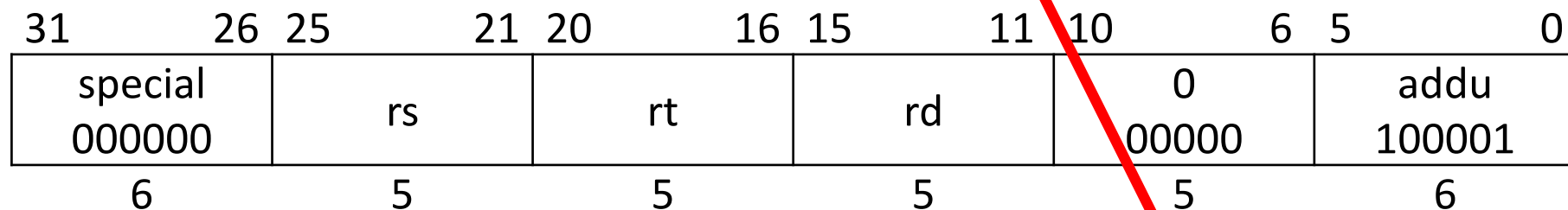
- A1: rs的寄存器编号
 - ◆ rs寄存器的编号对应IM[25:21]
- A2/A3: 同理

RTL

$R[\text{rd}] \leftarrow R[\text{rs}] + R[\text{rt}]$
 $PC \leftarrow PC + 4$

ADDU

指令	NPC		PC	IM	RF				ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B	Ad	Din
addu	PC.PC		NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11]	ALU.C	RF.RD1	RF.RD2		



□ 前提

- ◆ rd的值来自加法运算
- ◆ ALU具有加法功能

□ 结论：RF回写数据来自ALU

RTL

$$R[rd] \leftarrow R[rs] + R[rt]$$

$$PC \leftarrow PC + 4$$

ADDU

指令	NPC		PC	IM	RF				ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B	Ad	Din
addu	PC.PC		NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11]	ALU.C	RF.RD1	RF.RD2		

31	26	25	21	20	16	15	11	10	6	5	0																			
special 000000						rs					rt					rd					0 00000					addu 100001				
6						5					5					5					5					6				

- ADDU对2个寄存器进行加法，因此ALU的A和B自然就对应RF的2个数据输出

T
完全可以交换A和B的输入来源。
固定关系更利于降低复杂度

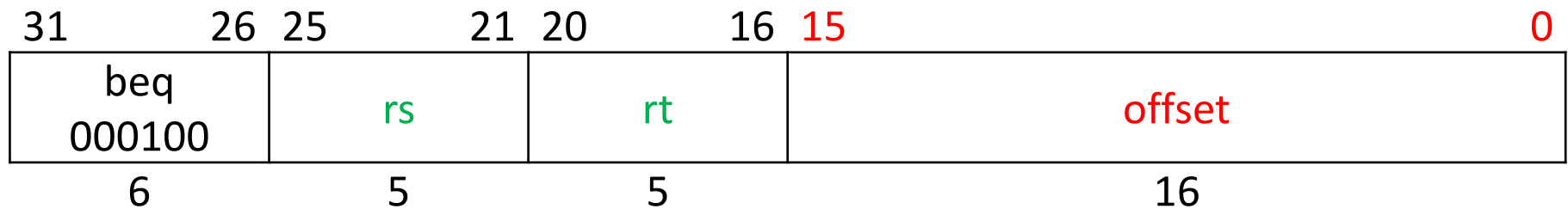
RTL

$$R[rd] \leftarrow R[rs] + R[rt]$$

$$PC \leftarrow PC + 4$$

BEQ

指令	NPC		PC	IM	RF				ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	A	B	Ad	Din
beq	PC.PC	IM[15:0]	NPC.NPC	PC.PC	IM[25:21]	IM[20:16]			RF.RD1	RF.RD2		



- 要点：PC计算涉及到立即数；借用ALU减法实现比较

```

if ( R[rs] == R[rt] )
    PC ← PC + 4 + sign_ext(imm16) || 00)
else
    PC ← PC + 4
    
```

ADDIU

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	D
addiu													

- Sign_ext(): 这是一个新的计算需求
 - ◆ 在原有的数据通路中无法满足
- S_EXT: 对应的新增功能部件

RTL

$R[rt] \leftarrow R[rs] + \text{sign_ext}(\text{imm16})$

$PC \leftarrow PC + 4$

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]	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	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
addiu	PC.PC		NPC.NPC	PC.PC	IM[25:21]		IM[20:16]	ALU.C	IM[15:0]	RF.RD1	S_EXT.Ext		

31	26	25	21	20	16	15	0
addiu		rs		rt		immediate	
001001							

- 由于不需要读第2个寄存器（即rt），因此RF的A2就无需输入

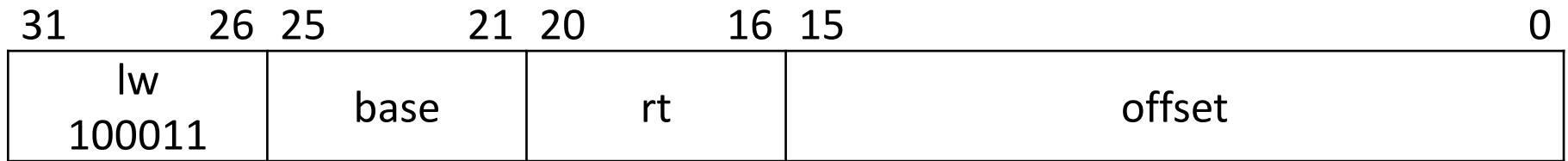
RTL

$R[rt] \leftarrow R[rs] + \text{sign_ext}(\text{imm16})$

$PC \leftarrow PC + 4$

LW

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
lw	PC.PC		NPC.NPC	PC.PC	IM[25:21]		IM[20:16]	DM.DO	IM[15:0]	RF.RD1	S_EXT.Ext	ALU.C	



- 新增DM功能部件
- 由于不是写存储器操作，因此DM.Din无需连接

RTL

$R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})]$

$PC \leftarrow PC + 4$

SW

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
sw	PC.PC		NPC.NPC	PC.PC	IM[25:21]	IM[20:16]			IM[15:0]	RF.RD1	S_EXT.Ext	ALU.C	RF.RD2

31	26	25	21	20	16	15	0
lw 101011		base		rt		offset	

- 注意：rs和rt都是应该被读出的！

RTL

$\text{MEM}[\text{R}[\text{rs}] + \text{sign_ext}(\text{imm16})] \leftarrow \text{R}[\text{rt}]$

$\text{PC} \leftarrow \text{PC} + 4$

JAL

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
jal													



- PC+4：在NPC中已经计算完成，因此只需要修改NPC的接口，增加PC+4输出即可
- PC计算：计算方法发生变化，需要修改NPC的接口与功能

RTL

$R[31] \leftarrow PC + 4$

$PC \leftarrow PC[31:28] \parallel instr_index \parallel 00$

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	PC[31:2]	NPC[31:2]
		Imm[25:0]	PC4[31:0]
		Op[1:0]	
		Zero	

JAL

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
jal	PC.PC	IM[25:0]	NPC.NPC	PC.PC			0x1F	NPC.PC4					



- RF.A3: 由于R[31]是默认的, 因此需要直接表示为0x1F
- RF.WD: 写入数据来自NPC输出的PC+4

RTL

$$R[31] \leftarrow PC + 4$$

$$\text{PC} \leftarrow \text{PC}[31:28] \parallel \text{instr_index} \parallel 00$$

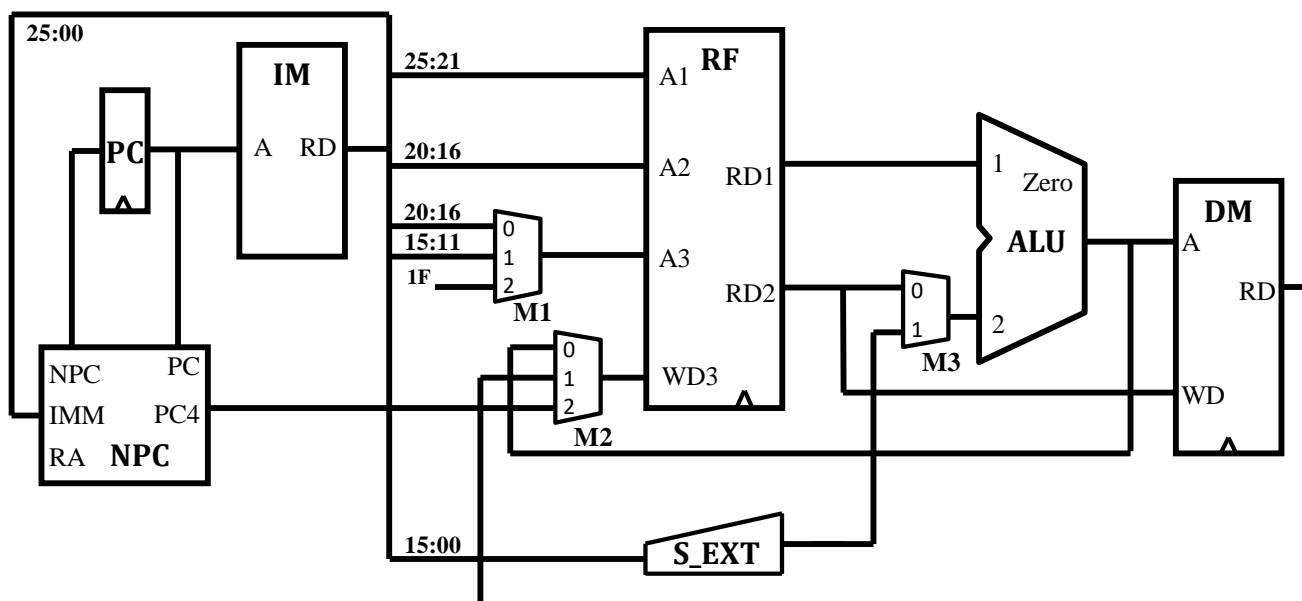
合并数据通路

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
addu	PC.PC		NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11]	ALU.C		RF.RD1	RF.RD2		
addiu	PC.PC		NPC.NPC	PC.PC	IM[25:21]		IM[20:16]	ALU.C	IM[15:0]	RF.RD1	S_EXT.Ext		
beq	PC.PC	IM[15:0]	NPC.NPC	PC.PC	IM[25:21]	IM[20:16]				RF.RD1	RF.RD2		
lw	PC.PC		NPC.NPC	PC.PC	IM[25:21]		IM[20:16]	DM.DO	IM[15:0]	RF.RD1	S_EXT.Ext	ALU.C	
sw	PC.PC		NPC.NPC	PC.PC	IM[25:21]				IM[15:0]	RF.RD1	S_EXT.Ext	ALU.C	RF.RD2
jal	PC.PC	IM[25:0]	NPC.NPC	PC.PC			0x1F	NPC.PC4					
合并	PC.PC	IM[25:0]	NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11] IM[20:16] 0x1F	ALU.C DM.DO NPC.PC4	IM[15:0]	RF.RD1	RF.RD2 S_EXT.Ext	ALU.C	RF.RD2

- 合并：垂直方向归并
- 增加MUX：对于输入来源为2个以上的，需在输入信号前部署MUX
 - 引入MUX，就必然引入MUX控制信号；MUX控制信号由控制器产生

数据通路的两种表示方式

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
合并	PC.PC	IM[25:0]	NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11] IM[20:16] 0x1F	ALU.C DM.DO NPC.PC4	IM[15:0]	RF.RD1	RF.RD2 S_EXT.Ext	ALU.C	RF.RD2



- 从形式建模的数据通路可以很容易的构造出图形化的数据通路

从设计模型到VerilogHDL

指令	NPC		PC	IM	RF				S_EXT	ALU		DM	
	PC	Imm	NPC	Ad	A1	A2	A3	WD	Imm	A	B	Ad	Din
合并	PC.PC	IM[25:0]	NPC.NPC	PC.PC	IM[25:21]	IM[20:16]	IM[15:11] IM[20:16] 0x1F	ALU.C DM.DO NPC.PC4	IM[15:0]	RF.RD1	RF.RD2 S_EXT.Ext	ALU.C	RF.RD2

```
wire      [31:0]      RD2 ;
wire      [31:0]      Ext ;
wire      [31:0]      ALU_B ;
```

```
RF        U_RF( ... , RD2 , ... ) ;           // 实例化寄存器堆
EXT       U_EXT( ... , Ext , ... ) ;          // 实例化扩展单元
ALU       U_ALU( ... , ALU_B , ... ) ;        // 实例化ALU
```

```
// 实例化ALU的B输入端MUX
```

```
MUX32_2_1 U_MUX_ALUB( RD2 , Ext , ALU_B , ALUBSrc ) ;
```

ALUBSrc: 属于生成出的信号；理论上可以自动生成

小节：开发过程

- 基本过程：形式建模，独立构造，简单综合，快速转换



设计输入

①形式建模

Cycle	Stage	Operation	Component		Signal
			IR	IRWr: 1	
Cycle1	Fetch (fetch instruction)	$IR \leftarrow IM[PC]$ $PC \leftarrow PC+4$	IR	IRWr: 1	ADD
			ALU	ALUOp: ADD	E
			PC	PCWr: 1	SE DD
Cycle2	RF (Read operator)	$A \leftarrow RF[rs]$ $EXT(Imm16)$	EXT	EXTOp: SE	ADD
Cycle3	MA (calculate DM address)	$ALUOut \leftarrow A + EXT$	ALU	ALUOp: ADD	
Cycle4	MR (read DM)	$DR \leftarrow DM[ALUOut]$			
Cycle5	MemWB (write DR to RF)	$RF[rt] \leftarrow DR$	RF	RFWr: 1	

②独立构造

③简单综合

设计输出

HDL文件

④快速转换

指令\部件	PC		NPC		IM	RF				EXT	ALU		DM
	DI	PC	Imm	RS	A	A1	A2	A3	WD		A	B	A
add	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2	
sub	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2	
ori	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[15:0]	RF.RD1	EXT.Ext	
lw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C
sw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]		IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C
slt	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2	
beq	NPC.NPC	PC.DO	IM.D[15:0]		PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2	
jal	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO			0x1F	NPC.PC4				
jr	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]							

部件	PC		NPC		IM	RF				EXT	ALU		DM
输入信号	DI	PC	Imm	RS	A	A1	A2	A3	WD		A	B	A
	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11] IM.D[20:16] 0x1F	ALU.C DM.RD NPC.PC4	IM.D[15:0]	RF.RD1	RF.RD2 EXT.Ext	ALU.C

小节：复杂度

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

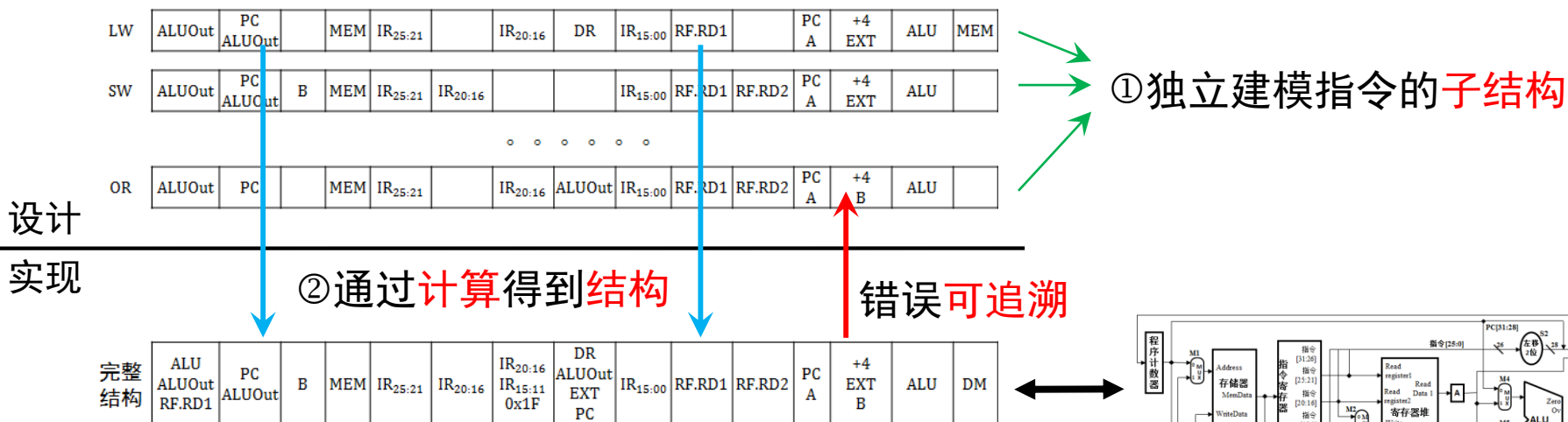
```
for each 输入来源多余1个的输入端
  部署1个MUX (MUX的输入规模为输入来源数)
  MUX设计定义、HDL建模
```

较低复杂度

HDL建模：连接所有的部件及所有的MUX

小节：优势

- 工程化方法：CPU开发复杂度低



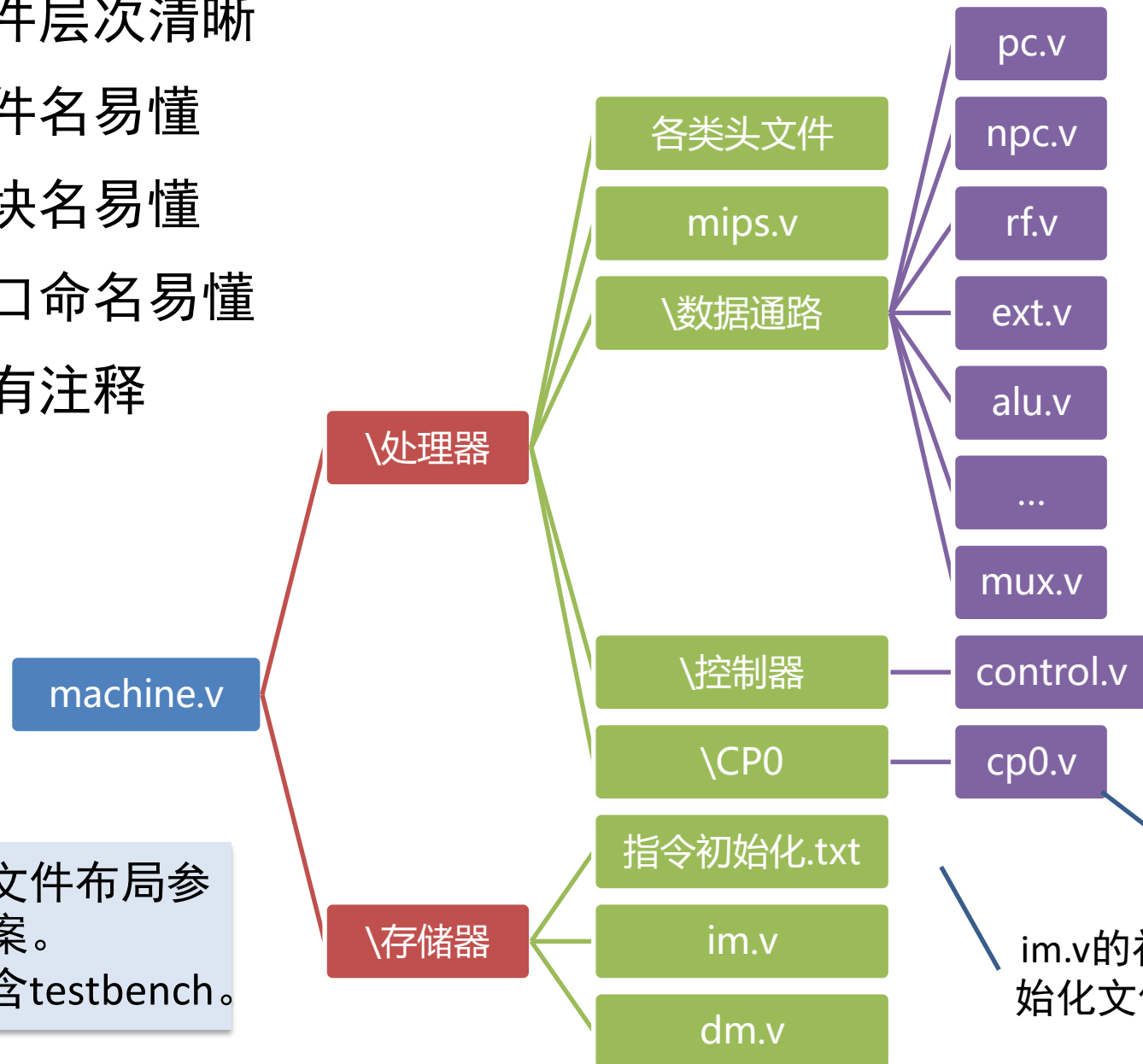
优势：

- 1.设计：与实现解耦
- 2.综合：指令集整体综合，可计算
- 3.排错：错误可追溯



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



工程文件布局参考方案。
不包含testbench。

与中断/异常相关（P7、P8）

im.v的初始化文件

忠告

- ❑ 方法的威力体现在数十条指令规模的CPU上
 - ◆ 简单问题不需要方法
- ❑ 应该努力学习在抽象模型层次上思考与工作
 - ◆ 图很直观，更适合理解，用于**频繁的细节性**设计则非常低效
- ❑ 在模型层面上完成设计后，再开始编码（实现）
 - ◆ 在模型层面的任何努力都将获得回报
 - ◆ 急于编码是不成熟的表现
 - 注意：这与基本的编码训练是无关的2个问题！

T

求解复杂问题时，应努力寻找抽象模型
从**模型层次**开始设计，可避免**实现细节**带给设计的**干扰**