

嵌入式微处理器系统

崔光佐

普适计算与应用实验室

北京大学现代教育技术中心

<http://www.uclab.org>



第二篇 ARM微处理器体系机构

第三讲 Thumb指令与三级流水设计

2004.2.21

主要内容

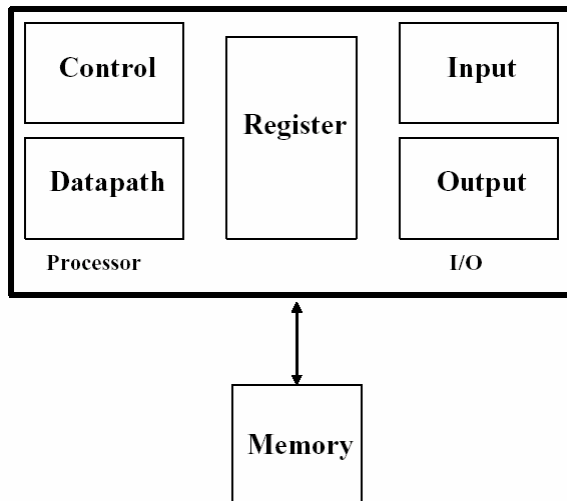
控制器设计概述

ARM9*译码器

ARM9*控制器

完整的ARM9*处理器

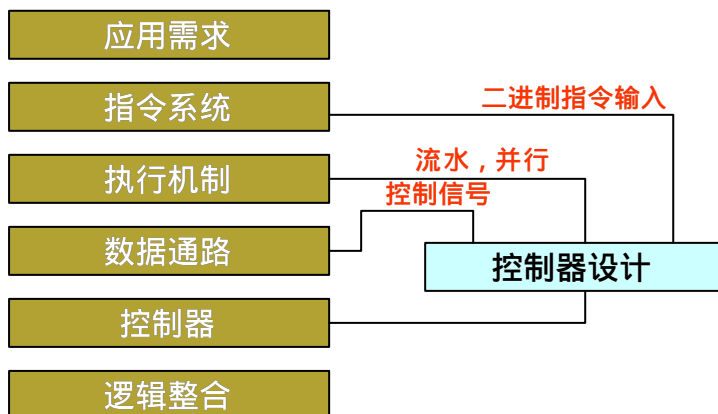
控制器设计概述:处理器总体结构



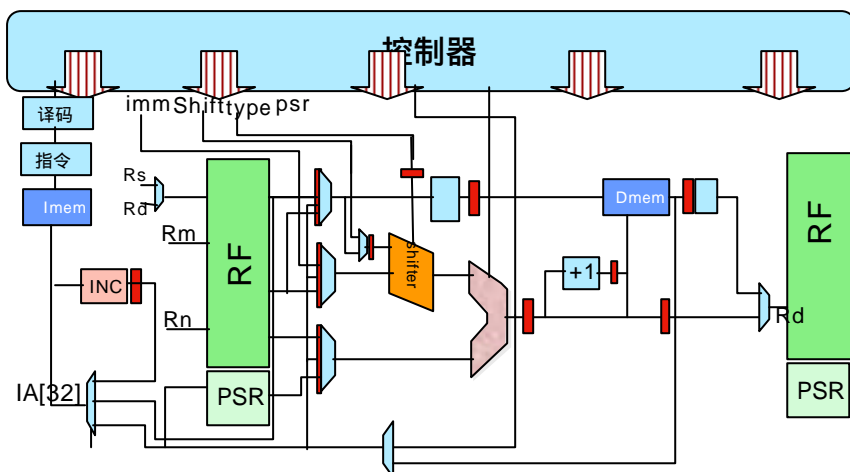
控制器设计概述：控制器作用

- 选择适当的操作完成指令的功能 (ALU, read, write, 等等)
- 控制数据的流动 (多路选择器, 输入等)
- 来自32位指令的信息
- 中断的处理

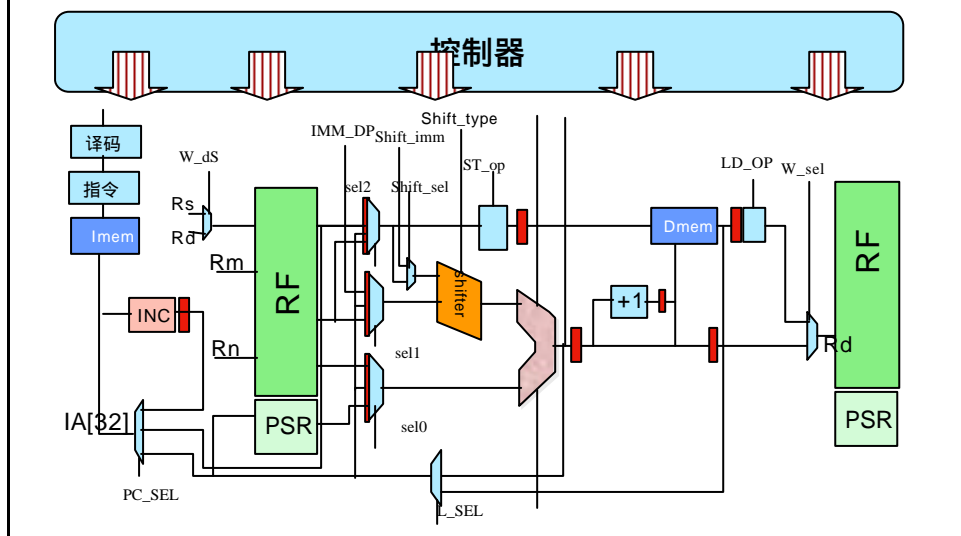
控制器设计概述



控制器设计概述：流水线寄存器



控制器设计概述：多功能模块操作选择



流水线信息控制：产生、发送（时间、模块）

对于五级流水线，控制器需要控制每一个流水段，为每个流水段产生相应的控制信号：

取指令，PC增加

指令译码，访问寄存器

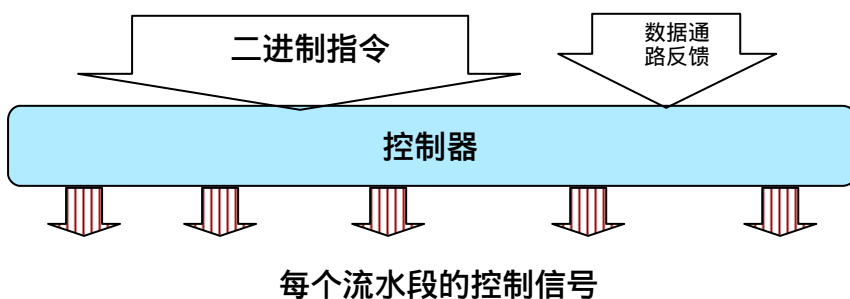
执行，控制每个部件，部件管理

访存

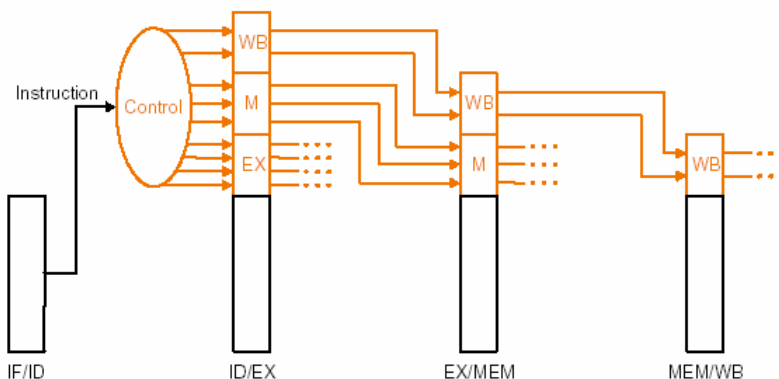
写回

•控制信号的传送：流水

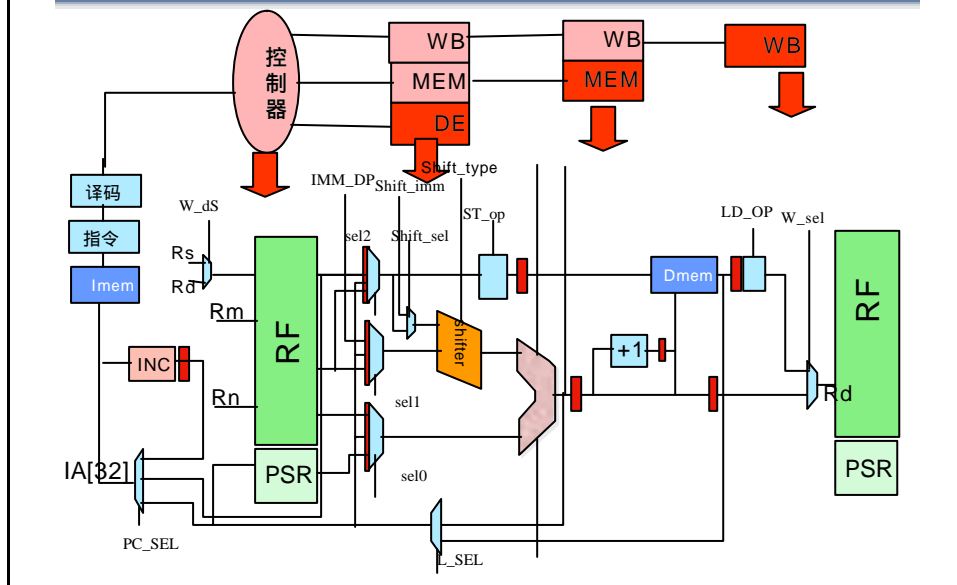
流水线控制器设计



控制器设计概述：控制信号的流水

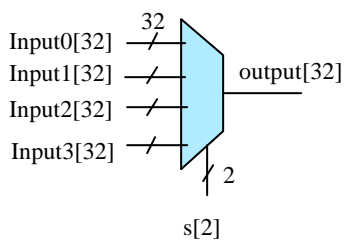


控制器设计概述：信号流水发送



数据通路中的常用部件

常用部件：多路选择器。

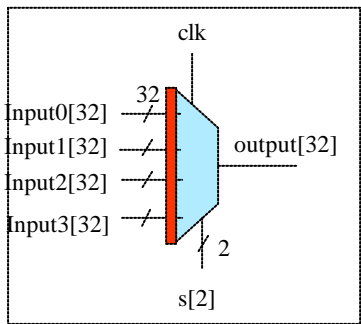


S	output
00	input0
01	input1
10	input2
11	input3

$I[k]$ 表示二进制指令的第k位

数据通路中的常用部件

常用部件：时钟控制多路选择器。



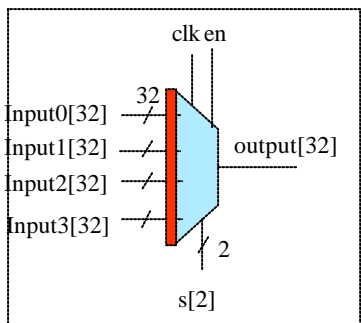
结构图

clk	S	output
1	00	input0
1	01	input1
1	10	input2
1	11	input3
0	xx	output

功能表

数据通路中的常用部件

常用部件：带使能端的时钟控制多路选择器。

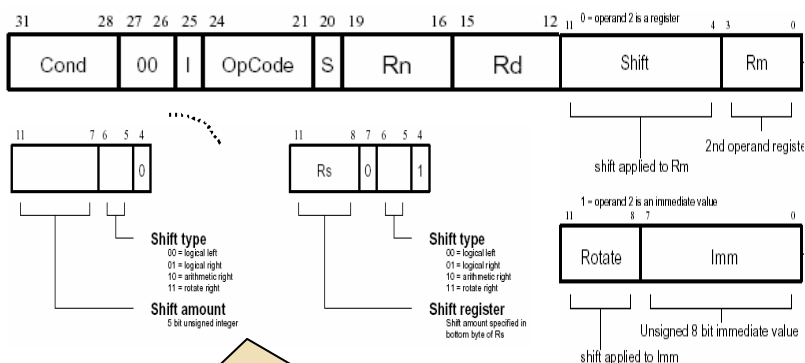


结构图

en	clk	s	output
1	1	00	input0
1	1	01	input1
1	1	10	input2
1	1	11	input3
0	x	xx	output
x	0	xx	output

功能表

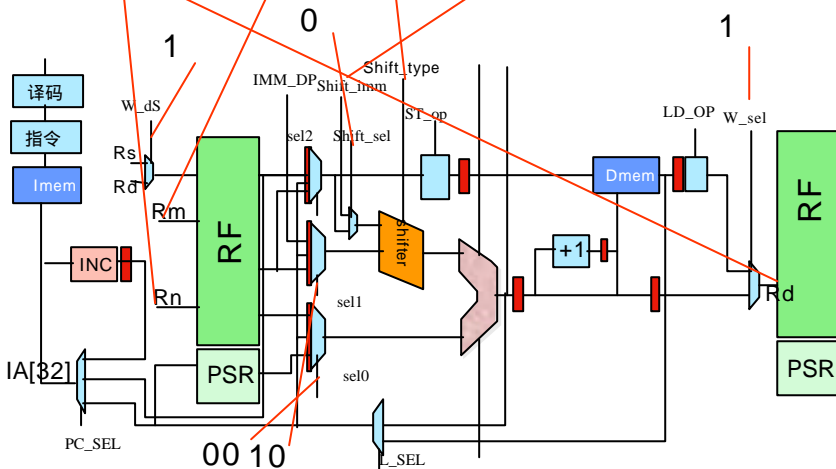
数据处理指令分析



$Rd \leftarrow Rn [\text{opcode}] (Rm [\text{shift type}] \text{Shift amount});$
 $PC \leftarrow PC + 4;$

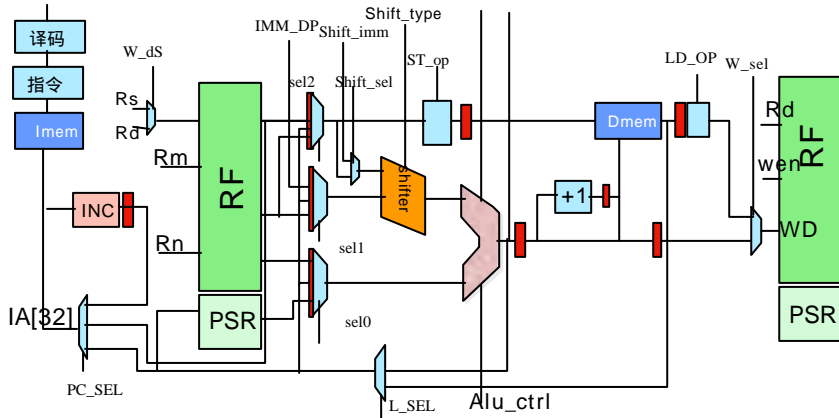
数据处理指令

- $Rd \leftarrow Rn [\text{opcode}] (Rm [\text{shift type}] \text{Shift amount});$
- $PC \leftarrow PC + 4$ 物理寄存器级描述

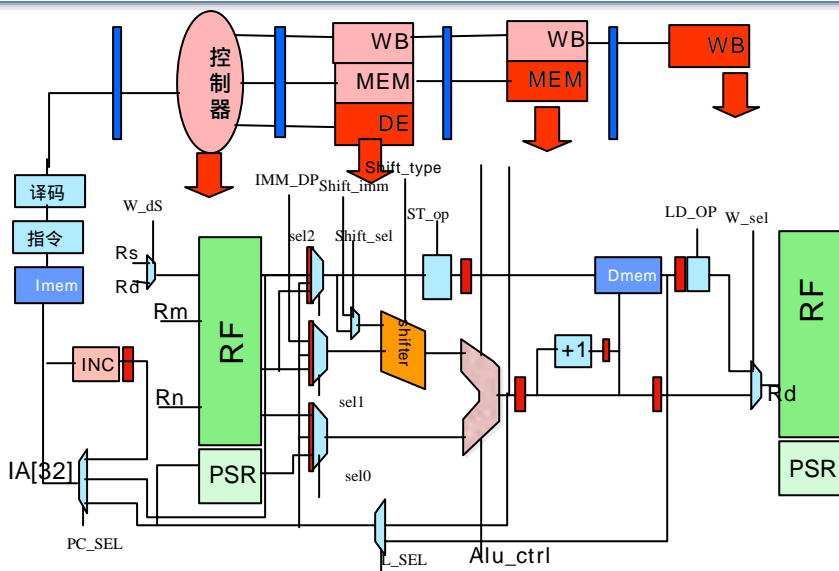


$Rd \leftarrow Rn [\text{opcode}] (Rm [\text{shift type}] \text{ Shift amount});$
 $PC \leftarrow PC + 4;$ 物理寄存器级描述

- Decode: Rn 、 Rm 、shift amount 采用硬连线, $PC_SEL = "00"$
- EXE: $sel1 = "10"$, $sel0 = "00"$, $shift_sel = 0$, $shift_type = I[5,6]$, $ALU_ctrl = i[21-24]$
- MEM:
- WB: $Rd = I[12-15]$, $wen = "1"$, $W_sel = "1"$;



数据处理指令

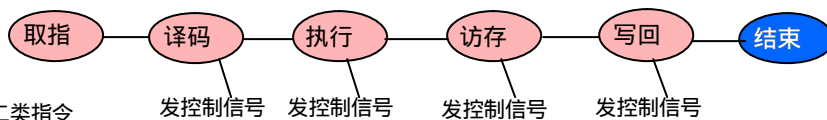


控制器设计概述

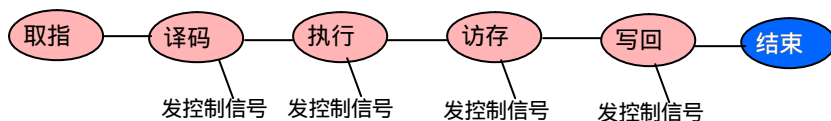
- 分析每一条指令的控制流，得出不同流水段的控制信号；
- 根据每条指令的控制信号分析结果，得出每条指令的控制流图；状态转换图；
- 综合所有指令的状态图和控制信号，得出控制器的总的状态图；

控制器设计概述

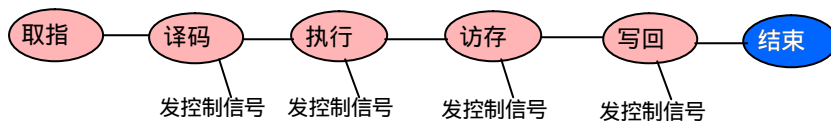
第一类指令



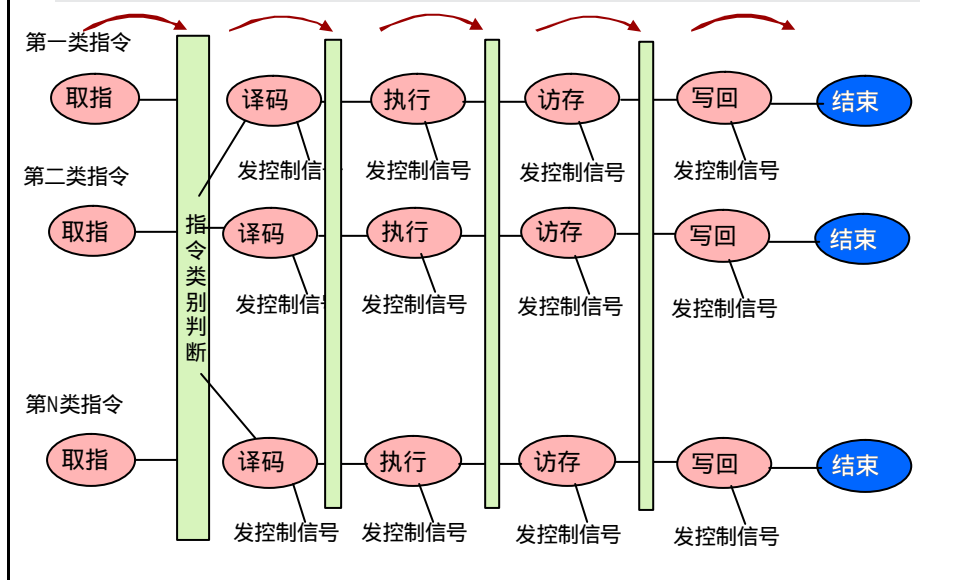
第二类指令



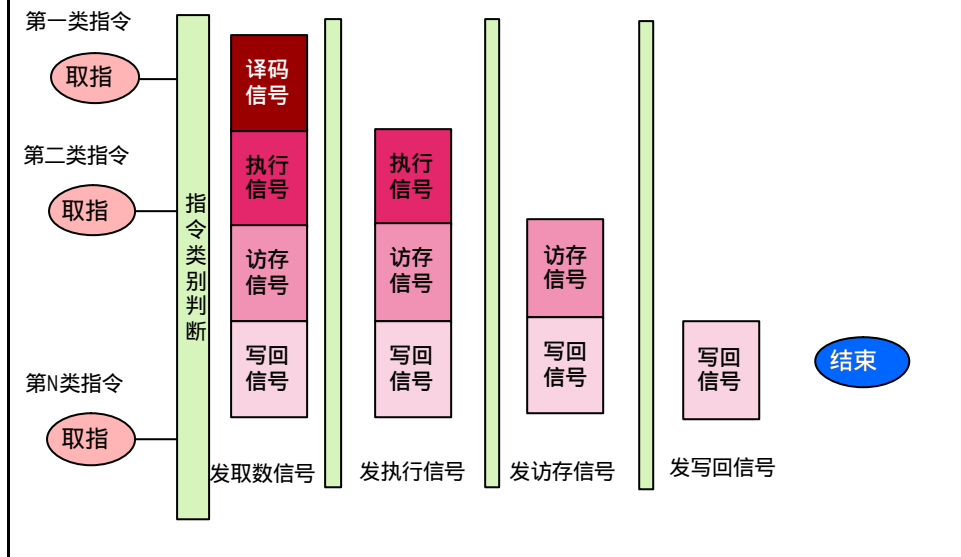
第N类指令



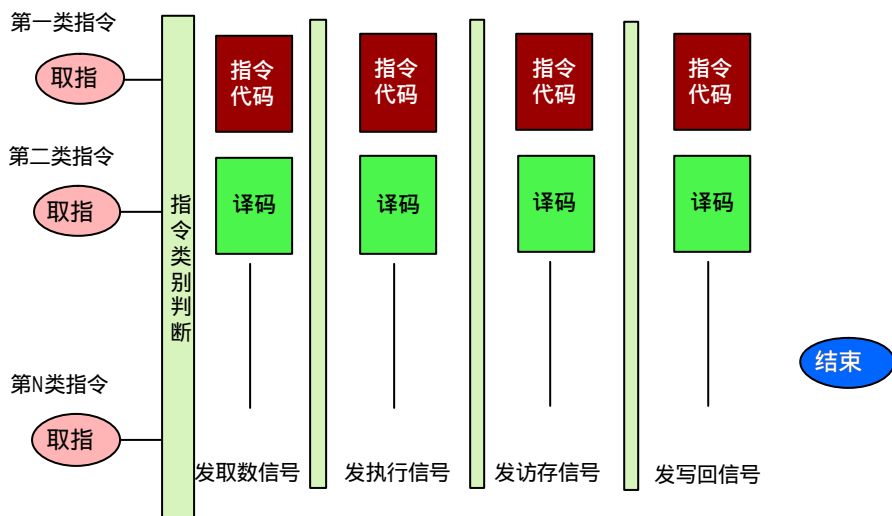
控制器综合：控制信号流水



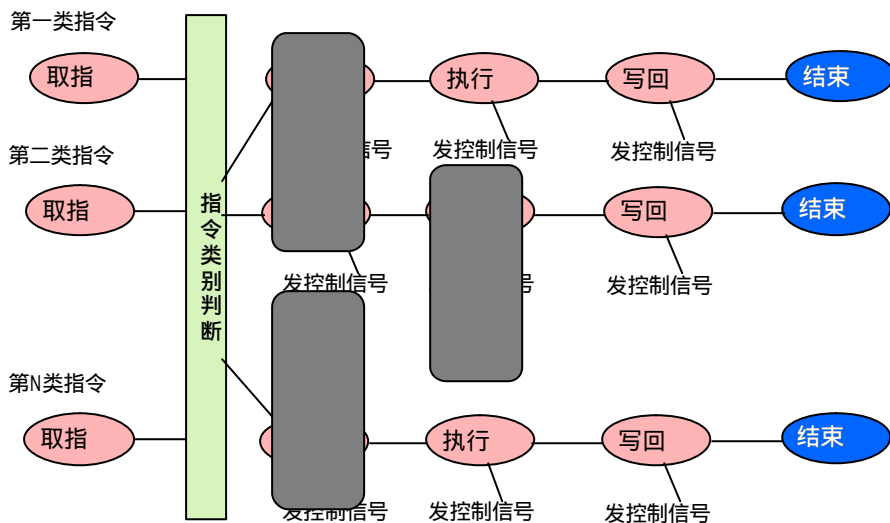
控制器综合：流水信号产生



流水信号产生：另一类方法



控制器优化：状态合并，提取相同信号

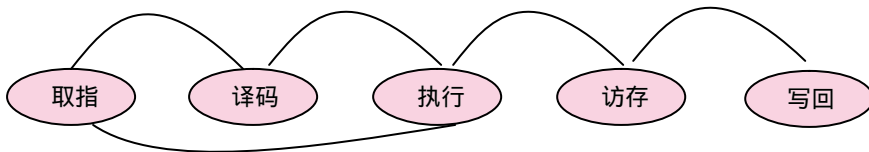


转移指令

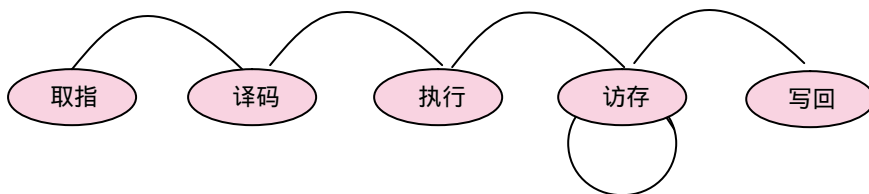
BE R1



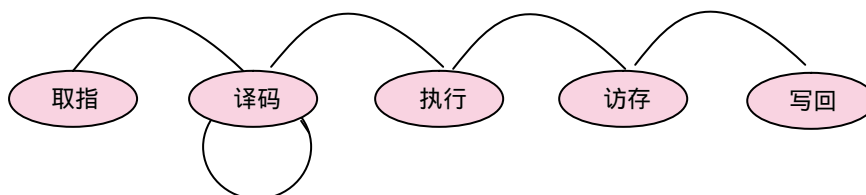
BL #100



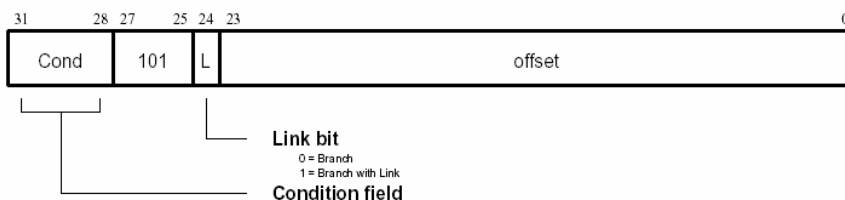
多周期指令：LDM R0,{R1,R2,R7}



多周期指令：STM R0,{R1,R2,R7}



控制器：流水线互锁—转移指令



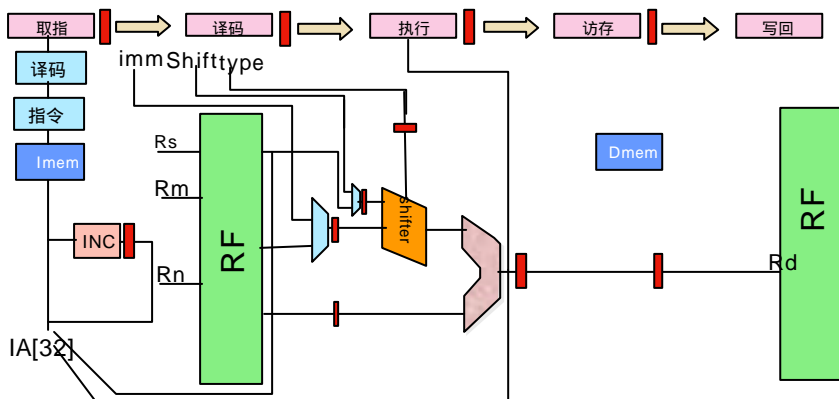
Logical Register Transfer:
 $R14 \leftarrow -PC; (\text{if } L=1)$
 $PC \leftarrow -PC + \text{Signed EXT}(\text{offset} * 4)$

转移指令

Logical Register Transfer:

$R14 \leftarrow -PC; (if L=1); PC \leftarrow -PC + Signed\ EXT(offset * 4)$

物理寄存器级描述: 控制器保存PC



转移指令

Logical Register Transfer:

$R14 \leftarrow -PC; (if L=1); PC \leftarrow -PC + Signed\ EXT(offset * 4)$

物理寄存器级描述: 控制器保存PC

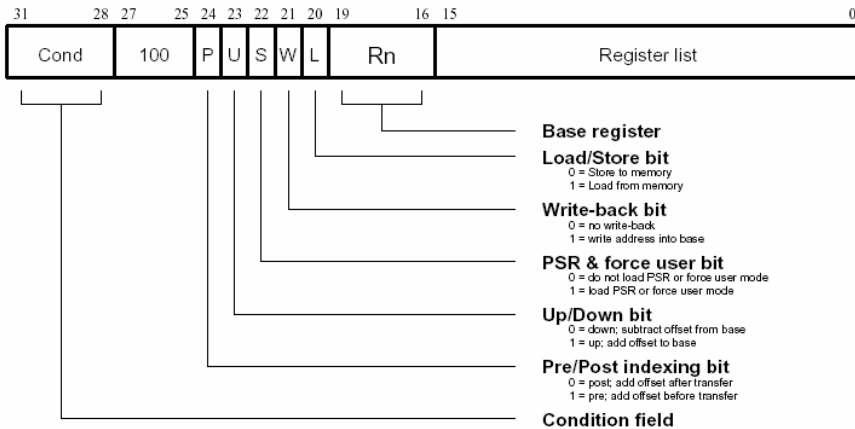


如何优化？硬件复杂度，停滞周期

LDM指令：分析

当L=1(Load)Logical Register Transfer:

Temp<-Rn; When offset[I]='1' then Ri<-MEM[Temp(+/-4)]; Temp<-Temp+/-4;
(Rn<-Rn+/- -ldstnum;) PC<-PC+4;



LDM指令：分析

当L=1(Load)Logical Register Transfer:

Temp<-Rn; When offset[I]='1' then Ri<-MEM[Temp(+/-4)]; Temp<-Temp+/-4;
(Rn<-Rn+/- -ldstnum;) PC<-PC+4;

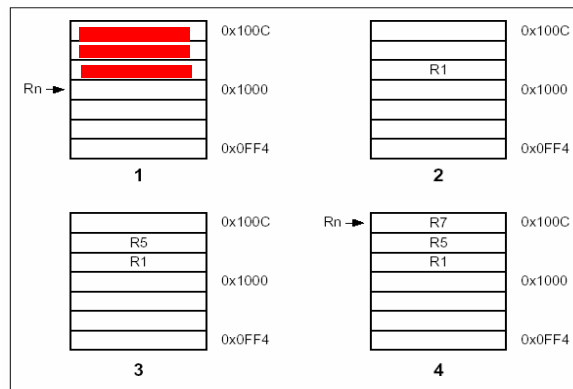
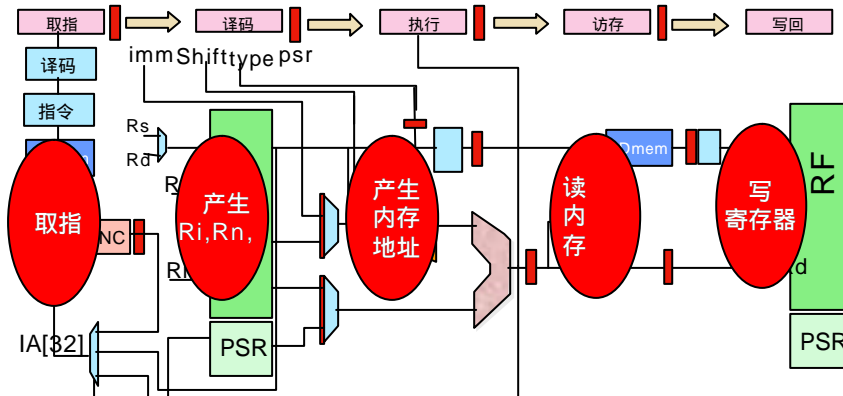


Figure 4-20: Pre-increment addressing

LDM指令：分析,装入第一个寄存器

当L=1(Load)Logical Register Transfer:

Temp<-Rn; When offset[I]='1' then Ri<-MEM[Temp(+/-4)]; Temp<-Temp+/-4;
(Rn<-Rn/+/-ldstnum;) PC<-PC+4;



LDM指令：分析

STMED SP!, {R0-R2, R14} ; Save R0 to R3 to use as workspace

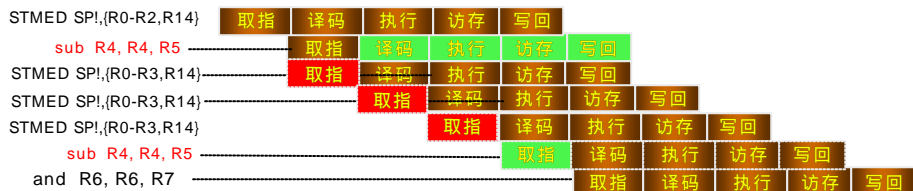
; and

R14 for returning. 假设寄存器地址都在译码阶段产生

sub R4, R1, R5

and R6, R1, R7

Name	Stack	Other	L bit	P bit	U bit
pre-increment load	LDMED	LDMIB	1	1	1

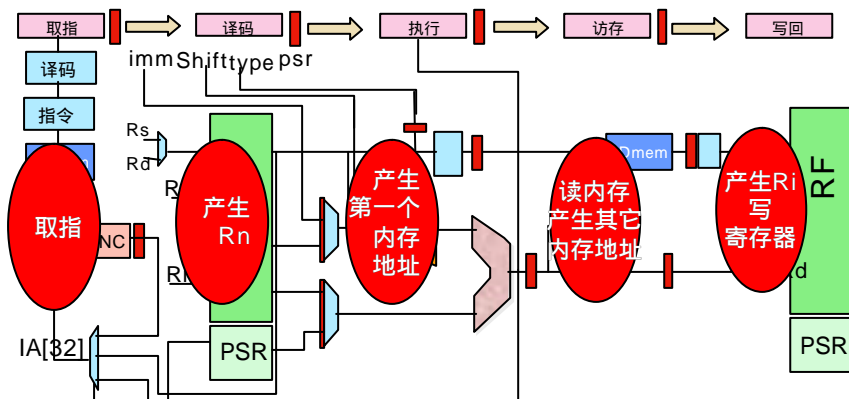


讨论：寄存器地址在何时产生？

当L=1(Load)Logical Register Transfer:

Temp<-Rn; When offset[I]='1' then Ri<-MEM[Temp(+/-4)]; Temp<-Temp+/-4;

(Rn<-Rn+/--ldstnum;) PC<-PC+4;



LDM指令：优化

STMED SP!,{R0-R2,R14} ; Save R0 to R3 to use as workspace

; and R14 for returning.

sub R4, R1, R5

and R6, R1, R7

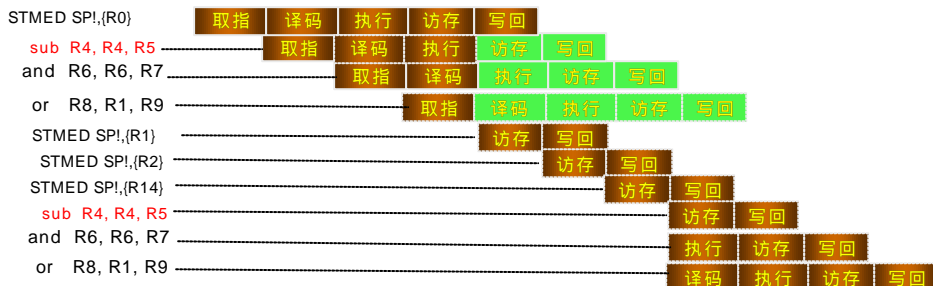
or R8, R1, R9

其余的寄存器地址在写回阶段产生

其余的内存地址在访存阶段产生

写回寄存器在第一次访存时完成

Name	Stack	Other	L bit	P bit	U bit
pre-increment load	LDMED	LDMIB	1	1	1



相关解决

相关检测：列出指令中所有相关的情形；
根据指令的操作和操作数可检测所有具体相邻指令的相关；

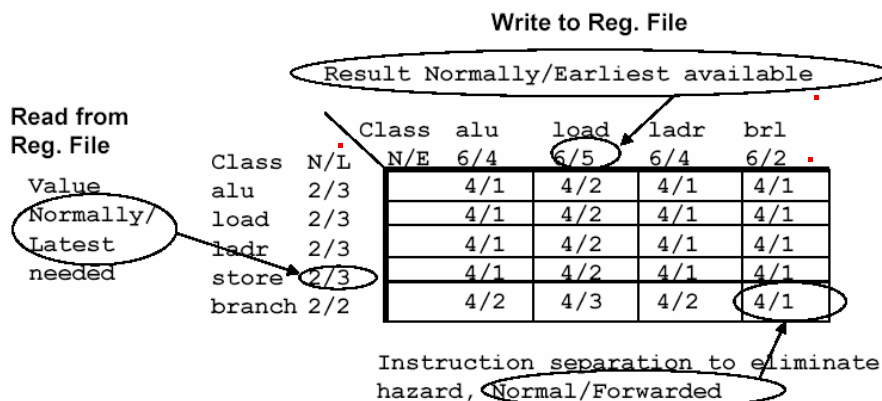
校正措施：

后面的相关指令可以停滞一段时间，等待前面的指令完成；
运算的结果前递给后面的指令，不必写入寄存器之后读取；

相关分析

- 考虑每一对指令的相关情况
- 通常数据在写入寄存器之后才有效
- 但可以将运算的结果提前发送给需要的指令
 - 阶段3的运算结果，阶段4的访存结果
- 操作数通常在第二阶段读取
- 操作数可在最后需要时接收前递的结果
- 阶段3的ALU操作以及地址的修改；阶段4存储的寄存器，阶段2需要转移目标

相关分析



- Latest needed stage 3 for store is based on address modifier register. The stored value is not needed until stage 4
- Store also needs an operand from ra. See Text Tbl 5.1

流水线相关处理

控制器检测到相关发生时，可使流水线停滞，以使前面的指令继续执行，而后面的指令禁止执行。

可以简单地通过关掉时钟的方法使一个流水段停滞，其输出不变，但如何处理空出的流水段呢？

假如流水段1和2停滞，流水段3如何处理呢？

插入NOP操作可以简单地解决

流水线相关处理

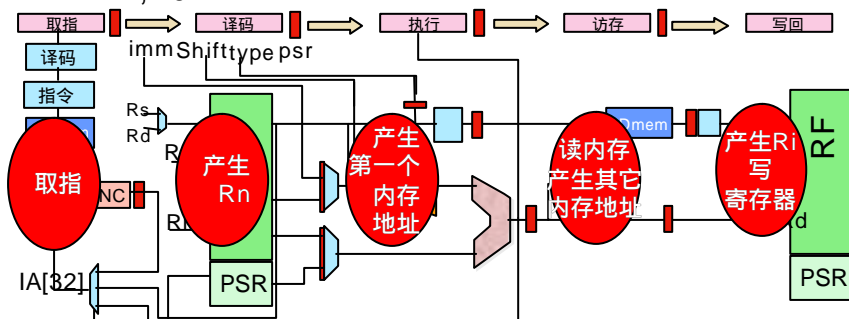
RAW相关：

一个寄存器写操作，后跟一个寄存器读操作。

BL [R7]

ADD R2,R0

ADD R2,R0



举例：ALU操作相关检测以及处理

- 1: OP Ra,Rb,Rc
- 2: OP Ra,Rb,Rc
- If $((OP1 \in ALU \wedge OP2 \in ALU) \wedge ((Rb2=Ra1) \vee (Rc2=Ra1)))$
- Then 发生数据相关，并进行处理
- 或停滞流水线，或采用前递
- End if;
- If (OP1=Branch)
- Then 发生控制相关
- 或停滞流水线，或采用前递
- Endif;

相关检测

- 对于ALU指令的阶段3依赖于另一个指令的阶段4的情况，检测等式描述如下：
- $alu4 = alu3 \quad ((ra4 = rb3) \quad X \quad Z4:$
- $(ra4 = rc3) \quad .imm3 \quad Y \quad Z4):$
- 但相关检测时，rb和rc在阶段3必须是有效的；
- Multiplexers must be put on the X and Y inputs to the ALU so that Z4 or Z5 can replace either X3 or Y3 as inputs

前递处理后指令序列的限制

(1) Branch delay slot

- The instruction after a branch is always executed, whether the branch succeeds or not.

```
br r4
add ...
...
```

(2) Load delay slot

- A register loaded from memory cannot be used as an operand in the next instruction.
- A register loaded from memory cannot be used as a branch target for the next two instructions.

```
ld r4, 4(r5)
nop
neg r6, r4

ld r0, 1000
nop
nop
br r0
```

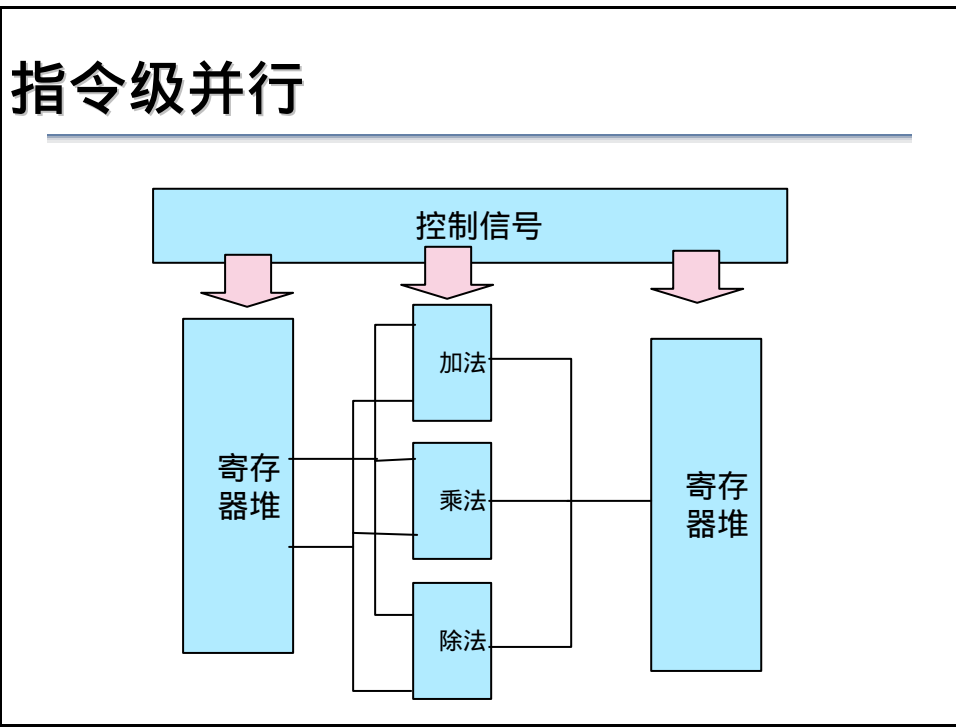
(3) Branch target

- Result register of ALU or ldr instruction cannot be used as branch target by the next instruction.

```
not r0, r1
nop
br r0
```

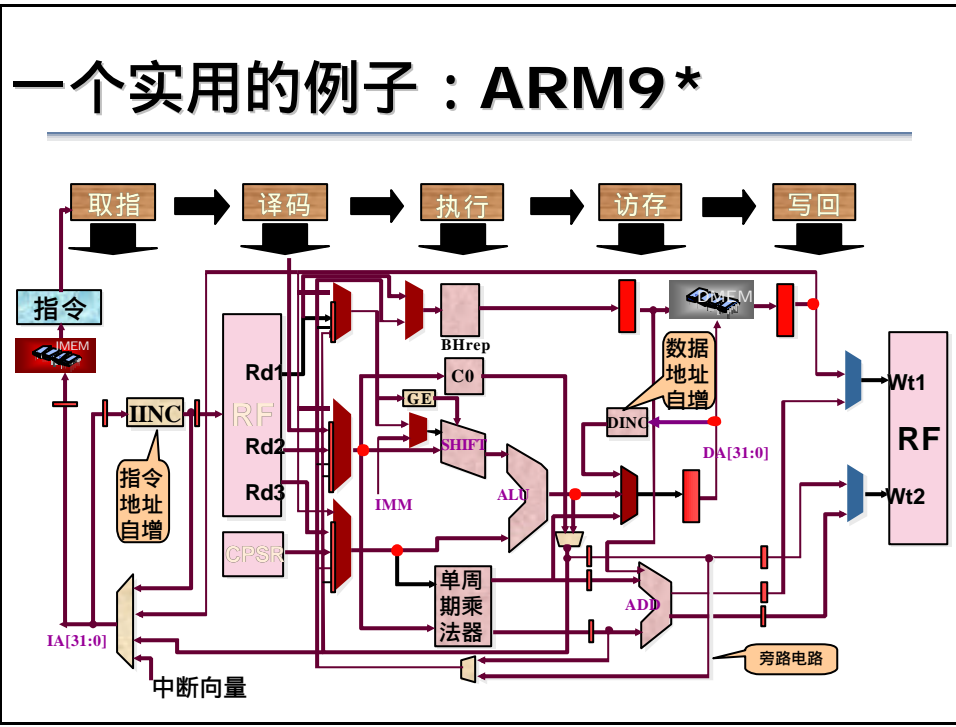

指令级并行

The diagram illustrates the architecture for Instruction-Level Parallelism (ILP). At the top, a horizontal light blue box labeled "控制信号" (Control Signal) has three pink arrows pointing downwards to three separate processing units. These units are arranged vertically in the center: "加法" (Addition), "乘法" (Multiplication), and "除法" (Division). To the left of these units is a vertical light blue box labeled "寄存器堆" (Register File), and to the right is another vertical light blue box labeled "寄存器堆" (Register File). Lines connect the left register file to each of the three processing units. Similarly, lines connect each of the three processing units to the right register file, indicating that multiple instructions can be executed simultaneously on different data paths.



一个实用的例子：ARM9*

The diagram illustrates the ARM9 architecture, showing the flow of instructions through five stages: 取指 (Instruction Fetch), 译码 (Instruction Decode), 执行 (Execute), 访存 (Memory Access), and 写回 (Write Back). Key components include the Instruction Memory (IMEM), Register File (RF), ALU, and various control units. Callouts highlight specific features like '指令地址自增' (Instruction Address Increment), '数据地址自增' (Data Address Increment), and '旁路电路' (Bypass Circuit).

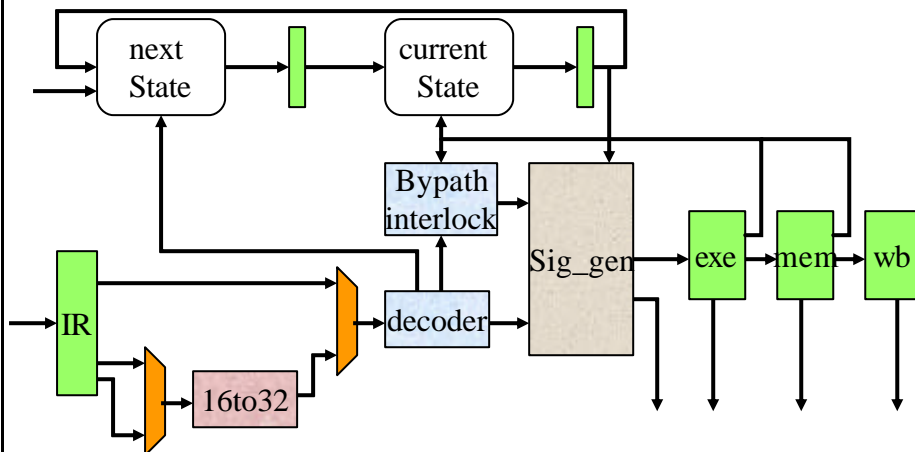


ARM9*的数据通路

数据通路结构和模块设计

数据通路控制信号说明

ARM* 的控制器设计



ARM9*控制器

■ 控制信号描述