

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

第四讲 五级流水线设计

2004.3.6

主要内容

- 数据通路功能设计
- 优化设计

ARM9流水线设计

The diagram illustrates the ARM9 pipeline design. It consists of five stages, each represented by a blue box with yellow text:

- 取指 (Fetch)
- 译码 (Decode)
- 执行 (Execute)
- 访存 (Access)
- 写回 (Write Back)

Below the stages, there are four green squares representing instructions. A large, stylized arrow points from the first two stages to the last two stages, indicating the flow of the pipeline. At the bottom, a clock signal waveform is shown, consisting of a series of pulses.

ARM9流水线设计

ARM	Thumb		
PC	PC	Fetch	Instruction fetched from memory
PC - 4	PC - 2	Decode	Registers used in instruction are decoded
PC - 8	PC - 4	Execute	Shift and ALU operation
PC - 12	PC - 6	Memory	Data access to/from memory
PC - 16	PC - 8	Writeback	Writeback to register bank

ARM指令概述

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR für			SPSR für	SPSR und

Datapath
RegisterFile

Controler

ARM指令概述

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
Cond	0	0	0	0	0	0	0	Opcode	S	Rn	Rd	Operand 2															Data Processing / PSR Transfer				
Cond	0	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm	Multiply													
Cond	0	0	0	0	0	1	U	A	S	RdHi	RdLo	Rn	1	0	0	1	Rm	Multiply Long													
Cond	0	0	0	0	1	0	B	0	0	Rn	Rd	0	0	0	0	1	0	0	1	Rm	Single Data Swap										
Cond	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	0	0	1	Rn	Branch and Exchange									
Cond	0	0	0	0	P	U	0	W	L	Rn	Rd	0	0	0	0	1	S	H	1	Rm	Halfword Data Transfer: register offset										
Cond	0	0	0	0	P	U	1	W	L	Rn	Rd	Offset	1	S	H	1	Offset	Halfword Data Transfer: immediate offset													
Cond	0	1	1	P	U	B	W	L	Rn	Rd	Offset																Single Data Transfer				
Cond	0	1	1														1											Undefined			
Cond	1	0	0	P	U	S	W	L	Rn	Register List																	Block Data Transfer				
Cond	1	0	1	L	Offset																							Branch			
Cond	1	1	0	P	U	N	W	L	Rn	CRd	CP#	Offset															Coprocessor Data Transfer				
Cond	1	1	1	0	CP	Opc	CRn	CRd	CP#	CP	0	CRm															Coprocessor Data Operation				
Cond	1	1	1	0	CP	Opc	L	CRn	Rd	CP#	CP	1	CRm															Coprocessor Register Transfer			
Cond	1	1	1	1	Ignored by processor																							Software Interrupt			
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

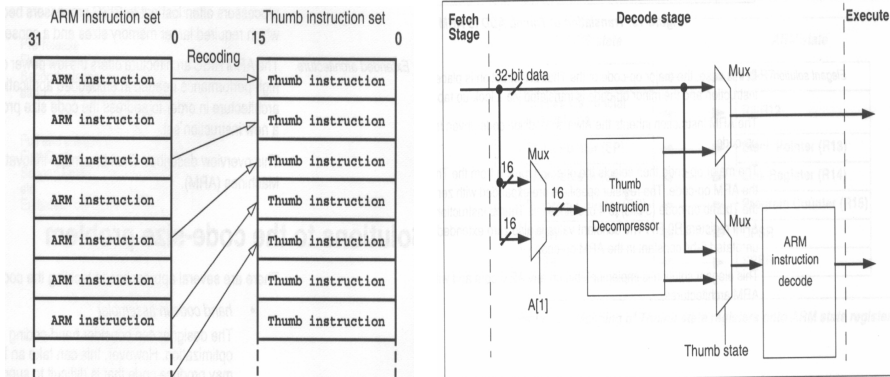
Thumb指令概述

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	Op	Offset5					Rs		Rd					Move shifted register	
2	0	0	0	1	1	Op	Rn/offset3			Rs		Rd					Add/subtract	
3	0	0	1	Op	Rd		Offset8										Move/compare/add /subtract immediate	
4	0	1	0	0	0	0	Op		Rs		Rd					ALU operations		
5	0	1	0	0	0	1	Op	H1	H2	Rs/Hs		Rd/Hd					Hi register operations /branch exchange	
6	0	1	0	0	1	Rd		Word8									PC-relative load	
7	0	1	0	1	L	B	0	Ro		Rb		Rd					Load/store with register offset	
8	0	1	0	1	H	S	1	Ro		Rb		Rd					Load/store sign-extended byte/halfword	
9	0	1	1	B	L	Offset5					Rb		Rd					Load/store with immediate offset
10	1	0	0	0	L	Offset5					Rb		Rd					Load/store halfword
11	1	0	0	1	L	Rd		Word8									SP-relative load/store	
12	1	0	1	0	SP	Rd		Word8									Load address	
13	1	0	1	1	0	0	0	0	S	SWord7							Add offset to stack pointer	
14	1	0	1	1	L	1	0	R	Rlist								Push/pop registers	
15	1	1	0	0	L	Rb		Rlist									Multiple load/store	
16	1	1	0	1	Cond			Soffset8					Rd					Conditional branch
17	1	1	0	1	1	1	1	1	Value8								Software interrupt	
18	1	1	1	0	0	Offset11					Rd							Unconditional branch
19	1	1	1	1	H	Offset					Rd							Long branch with link
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		

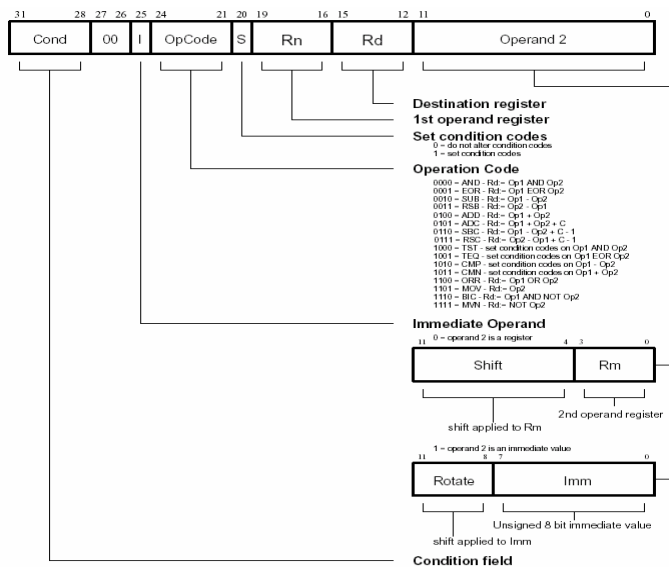
Undefined
R0
R1
R2
R3
R4
R5
R6
R7
SP_und
LR_und
PC

CPSR
SPSR_und

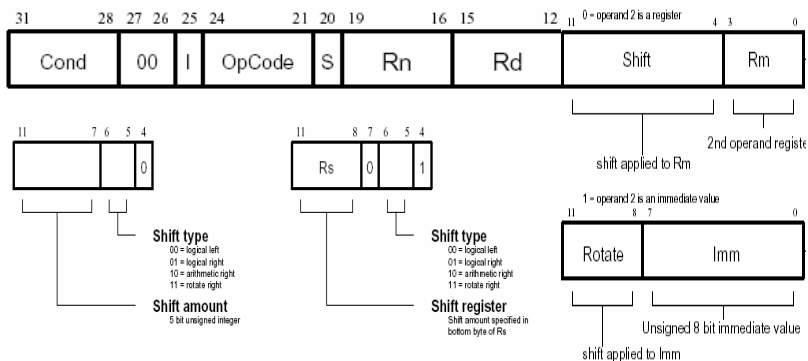
ARM与Thumb的统一



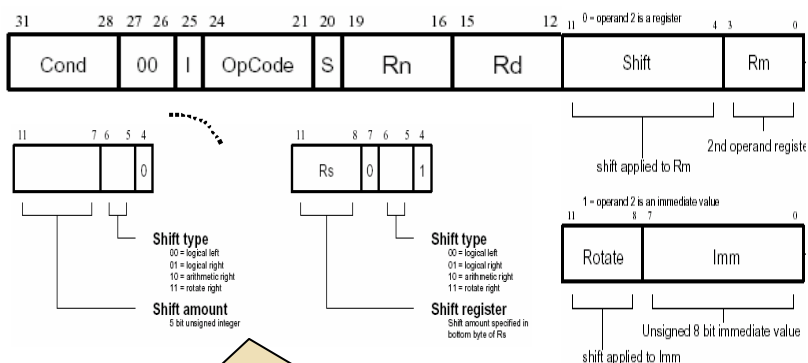
数据处理指令



数据处理指令



数据处理指令

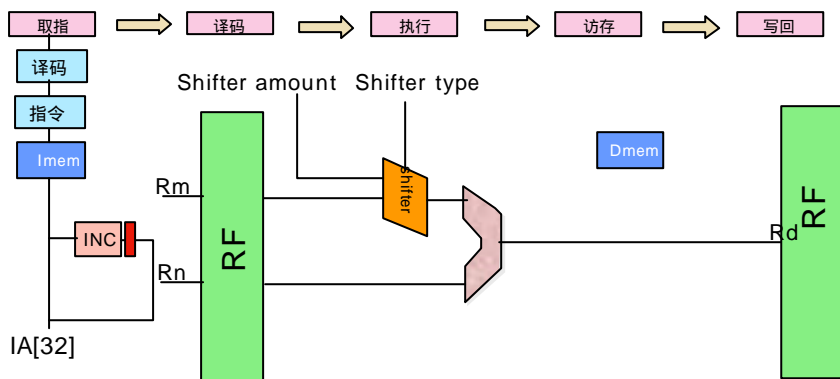


$Rd \leftarrow Rn [\text{opcode}] (Rm [\text{shift type}] \text{Shift amount});$
 $PC \leftarrow PC + 4;$

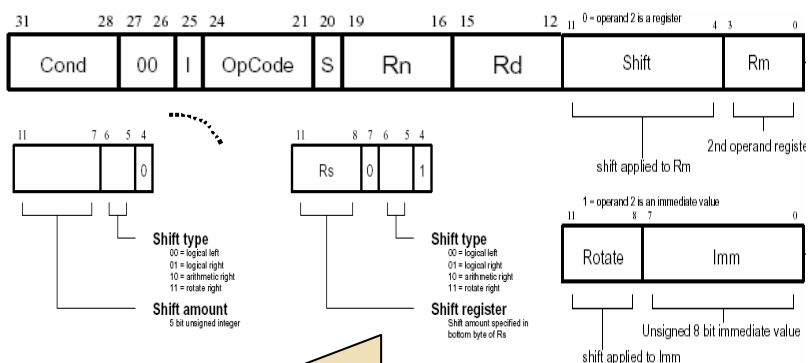
数据处理指令

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

$PC \leftarrow PC + 4;$ 物理寄存器级描述



数据处理指令

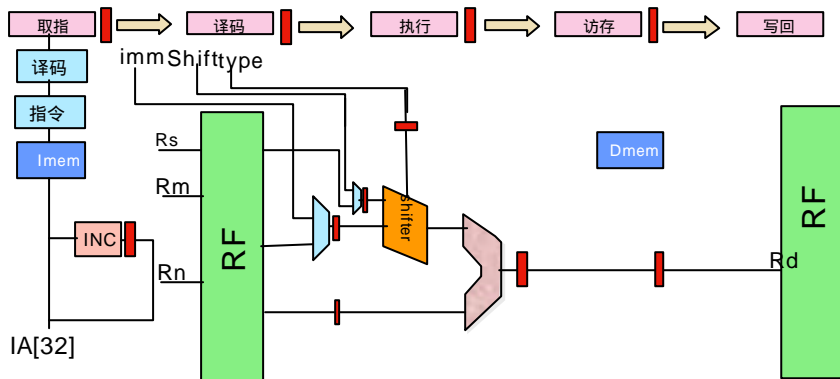


$Rd \leftarrow Rn$ [opcode] (Rm [shift type] Rs);
 $PC \leftarrow PC + 4$;

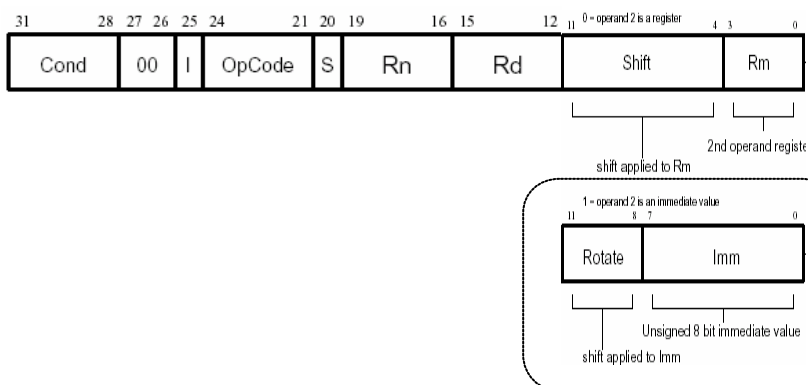
数据处理指令:时序设计

$Rd \leftarrow Rn$ [opcode] (Rm [shift type] Rs);

$PC \leftarrow PC + 4$; 物理寄存器级描述



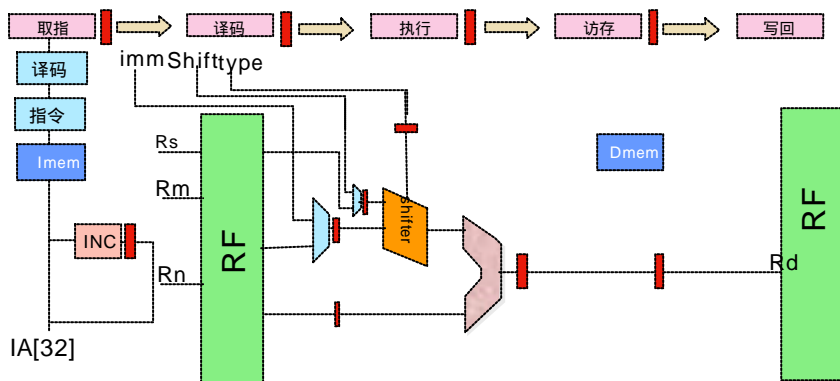
数据处理指令



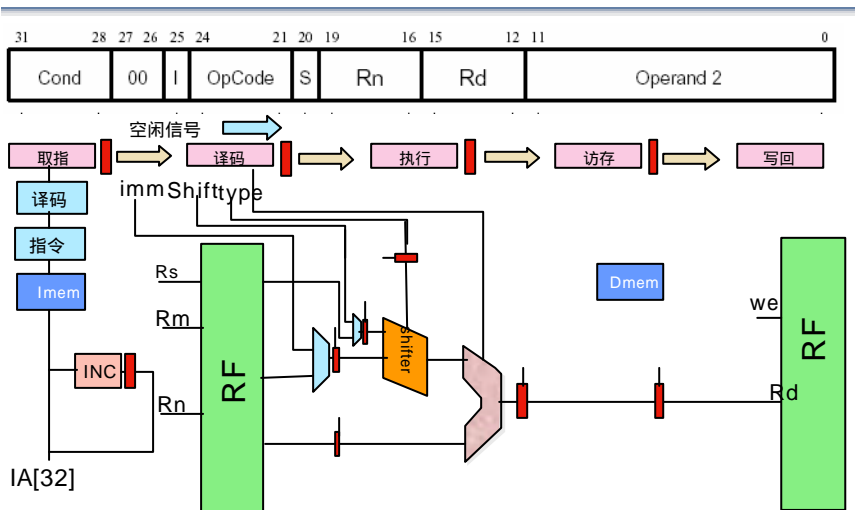
$Rd \leftarrow Rn [\text{opcode}] (\text{imm} [\text{shift type}] \text{Rotate});$
 $PC \leftarrow PC + 4;$

数据处理指令: Datapath不用修改

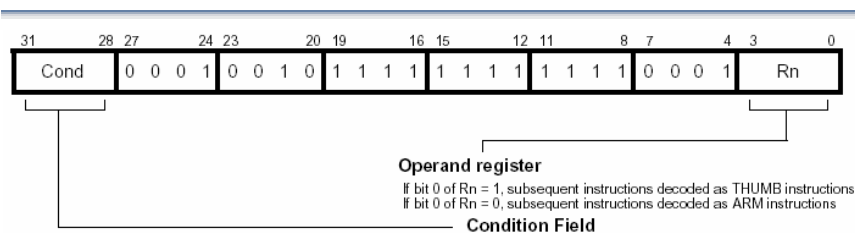
$Rd \leftarrow Rn [\text{opcode}] (\text{imm} [\text{shift type}] \text{Rotate});$
 $PC \leftarrow PC + 4;$ 物理寄存器级描述



条件执行



转移指令



Logical Register Transfer:

PC ← Rm;

If Rm[0]=1 then

Change into JBCore16;

Else

Change into JBCore32;

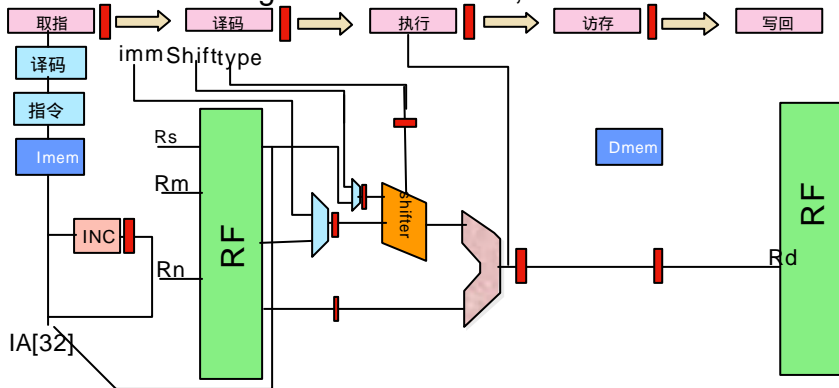
End if

转移转换指令（32/16位转换）

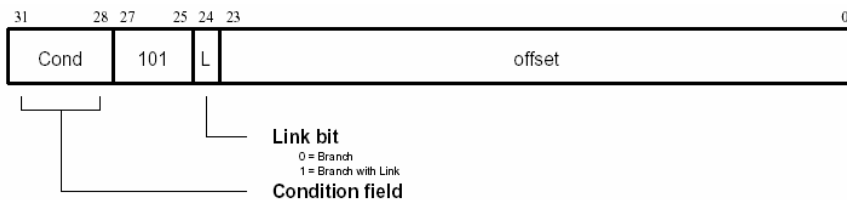
物理寄存器级描述:

PC ← Rm; If Rm[0]=1 then Change into JBCore16;

Else Change into JBCore32; End if



Branch/Branch with link



Logical Register Transfer:

$R14 \leftarrow PC; (\text{if } L=1)$

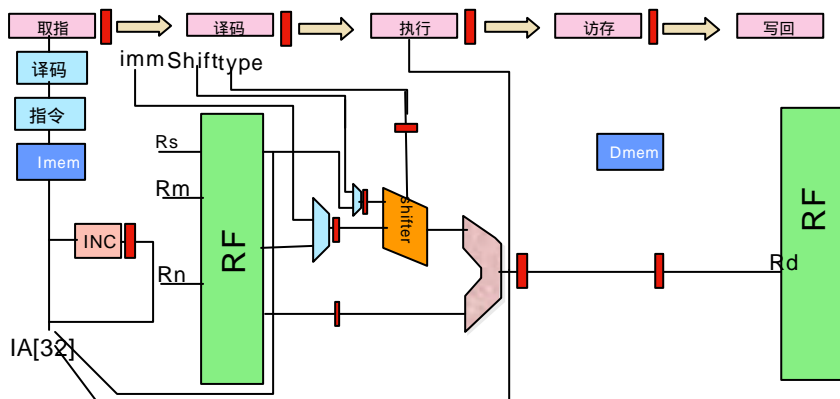
$PC \leftarrow PC + \text{Signed EXT}(\text{offset} * 4)$

Branch/Branch with link

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

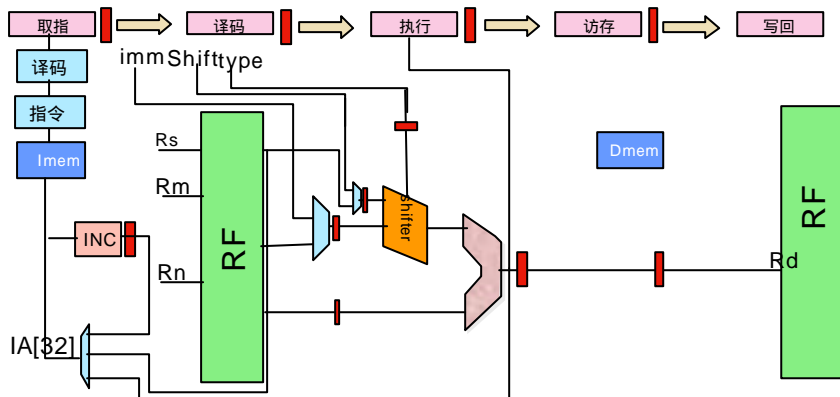


Branch/Branch with link

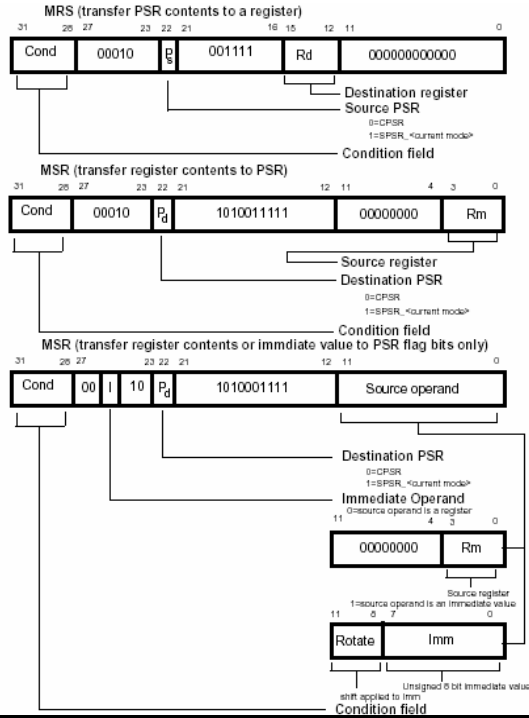
Logical Register Transfer:

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

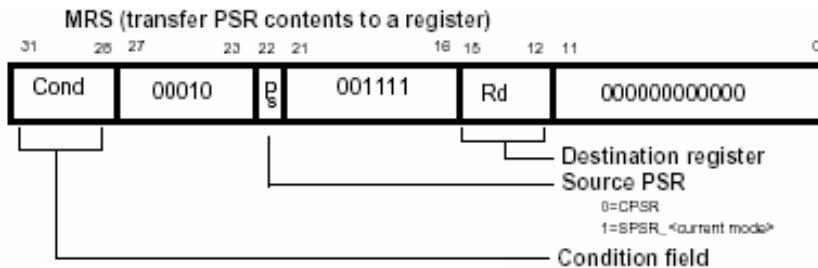
物理寄存器级描述: 控制器保存PC



PSR Transfer:



PSR Transfer (1)



当R=0

Logical Register Transfer:

Rd<-CPSR;

PC<-PC+4;

当R=1

Logical Register Transfer:

Rd<-SPSR;

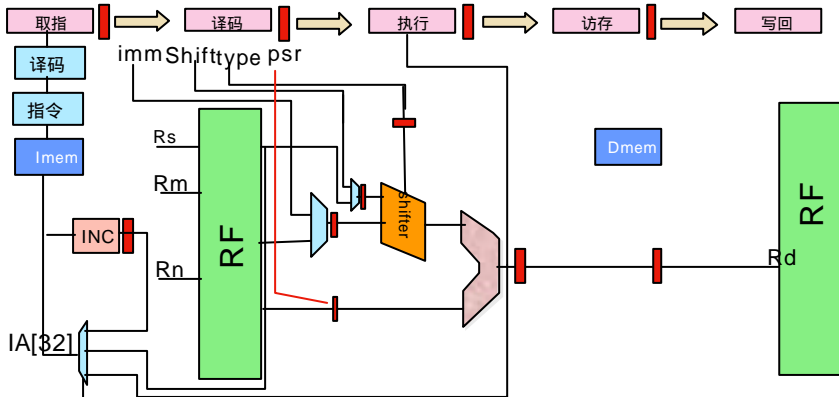
PC<-PC+4;

PSR Transfer (1)

当R=0 Logical Register Transfer:

$R_d \leftarrow \text{CPSR}; PC \leftarrow PC + 4;$

物理寄存器级描述:

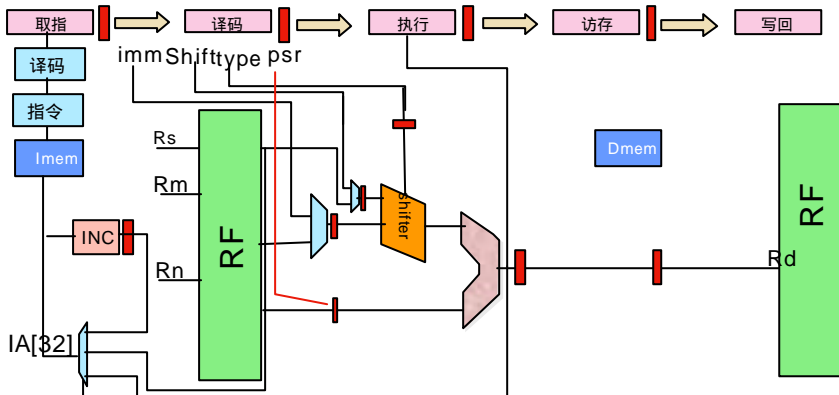


PSR Transfer (1)

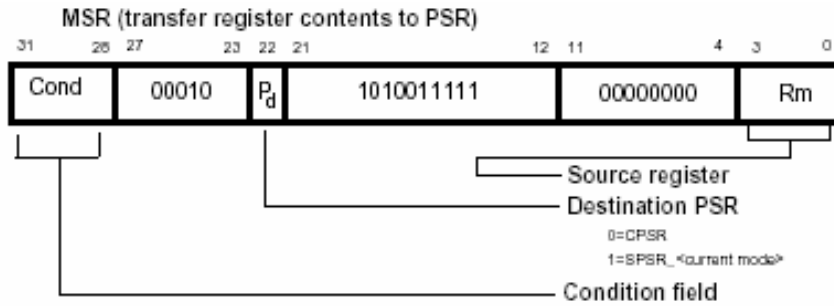
当R=1 Logical Register Transfer:

$R_d \leftarrow \text{SPSR}; PC \leftarrow PC + 4;$

物理寄存器传输级:



PSR Transfer (2)



当R=0

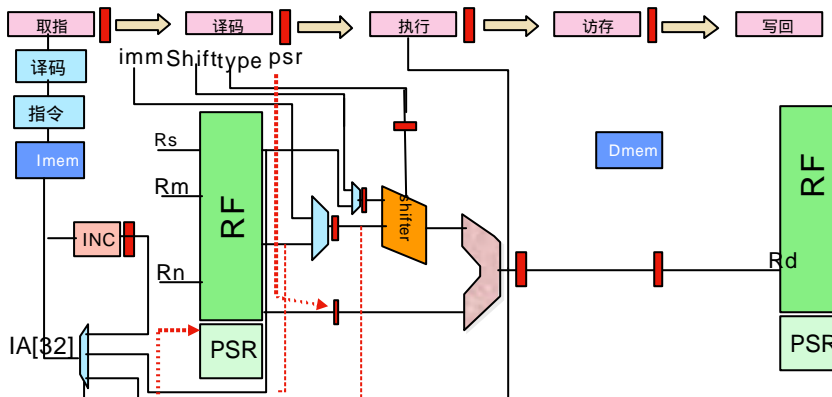
Logical Register Transfer: CPSR← Rm;
PC←PC+4;

当R=1

Logical Register Transfer: SPSR← Rm;
PC←PC+4;

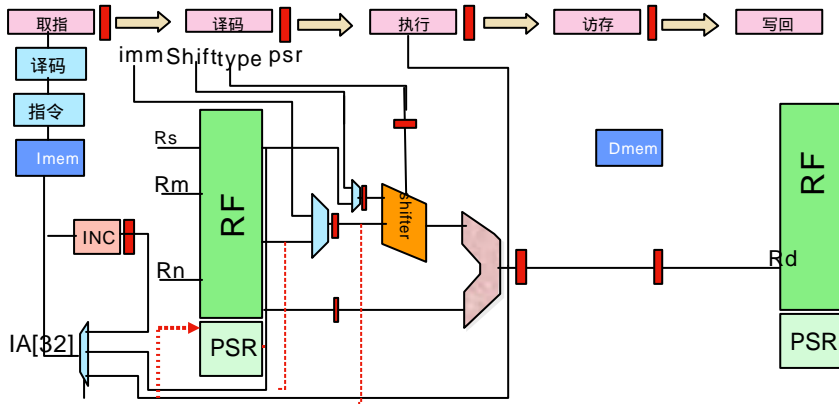
PSR Transfer (2)

Logical Register Transfer: 当R=0 , CPSR← Rm;
当R=1 , SPSR← Rm; PC←PC+4;



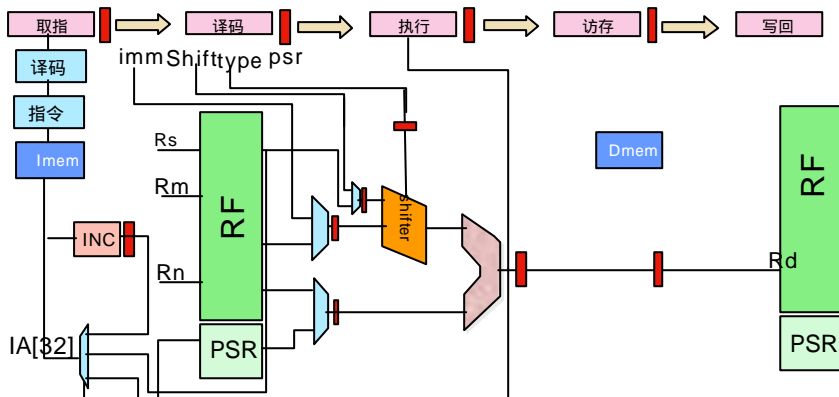
PSR Transfer (2) 哪种方法最好?

Logical Register Transfer: 当R=0, CPSR←Rm;
当R=1, SPSR←Rm; PC←PC+4;

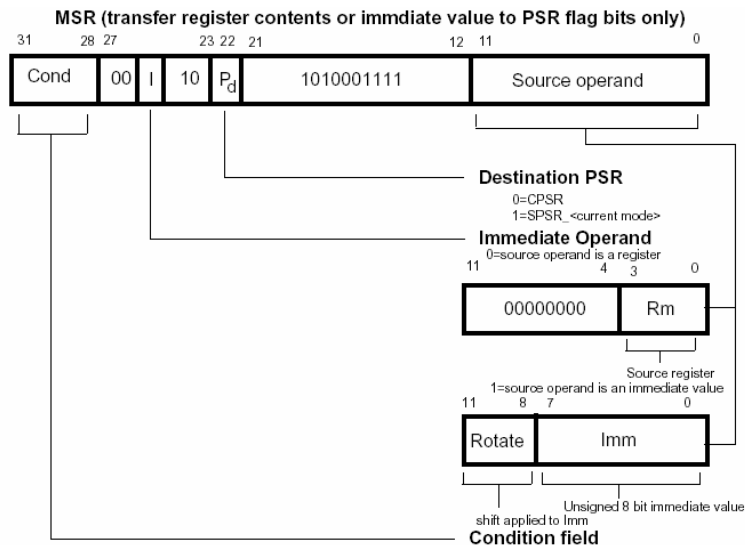


PSR Transfer (2)

Logical Register Transfer: 当R=0, CPSR←Rm;
当R=1, SPSR←Rm; PC←PC+4;

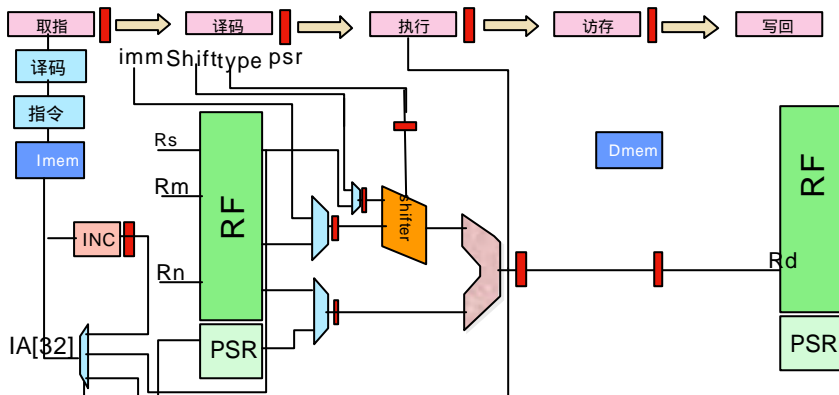


PSR Transfer (3) 只改变Flag



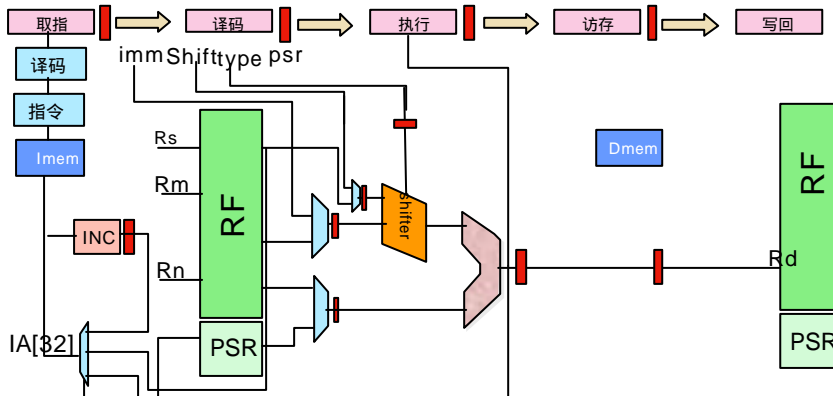
PSR Transfer (3) 只改变Flag

Logical Register Transfer: 当R=0, CPSR<- Rm;
当R=1, SPSR<- Rm; PC<-PC+4;

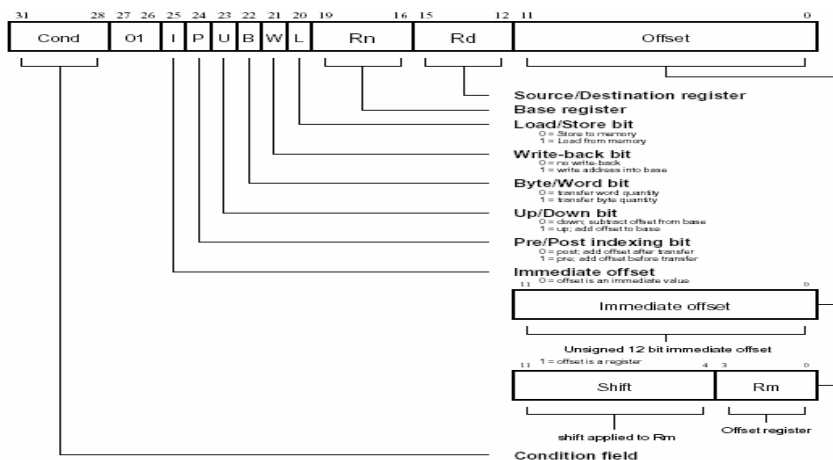


PSR Transfer (3) 只改变Flag

Logical Register Transfer: $\text{CPSR}[\text{flag}] \leftarrow \text{imm}[\text{Rotate}] \text{shift};$
 $\text{SPSR}[\text{flag}] \leftarrow \text{imm}[\text{Rotate}] \text{shift} \ll \text{PC} + 4;$



Single Data Transfer (LDR, STR)



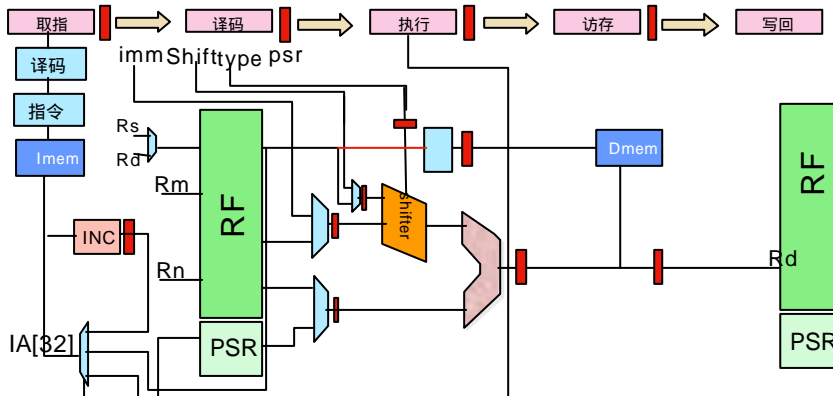
Load/Store register offset

当L=0(store)

Logical Register Transfer:

$R_d \rightarrow \text{MEM}(R_n + R_m [\text{shift}] \text{ShiftImmediate});$

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



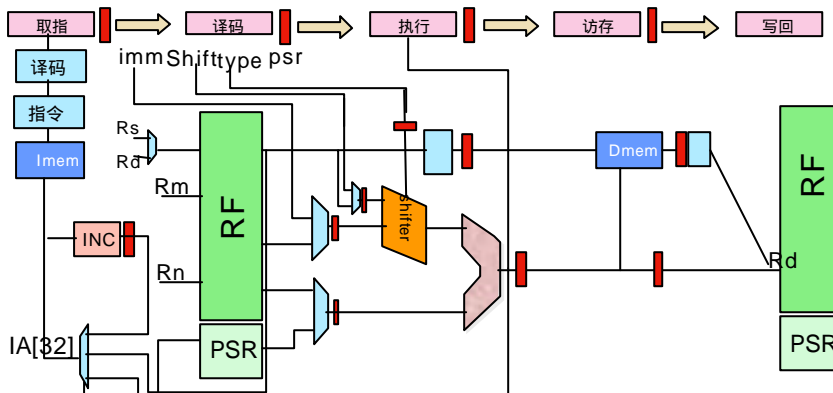
Load/Store register offset

当L=1(load)

Logical Register Transfer:

$R_d \leftarrow \text{MEM}(R_n + R_m [\text{shift}] \text{Shift immediate});$

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



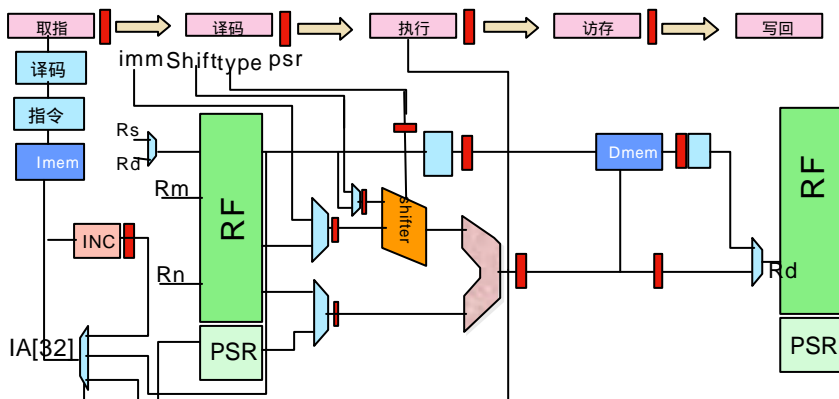
Load/Store register offset

当L=1(load)

Logical Register Transfer:

$R_d \leftarrow \text{MEM}(R_n + R_m [\text{shift}] \text{ Shift immediate});$

$PC \leftarrow PC + 4;$



Load/Store register offset

- P = 0: 基址与偏移量相加在传输之后 ;
- 1: 基址与偏移量相加在传输之前 ;
- U = 0: 基址 = 基址 - 偏移量
- 1: 基址 = 基址 + 偏移量
- B = 0: 传送整个字 ;
- 1: 传送单个字节。
- W = 0: 基址不写回 ;
- 1: 基址写会。
- Rn: 基址寄存器 ;
- Rd: 源 / 目的寄存器

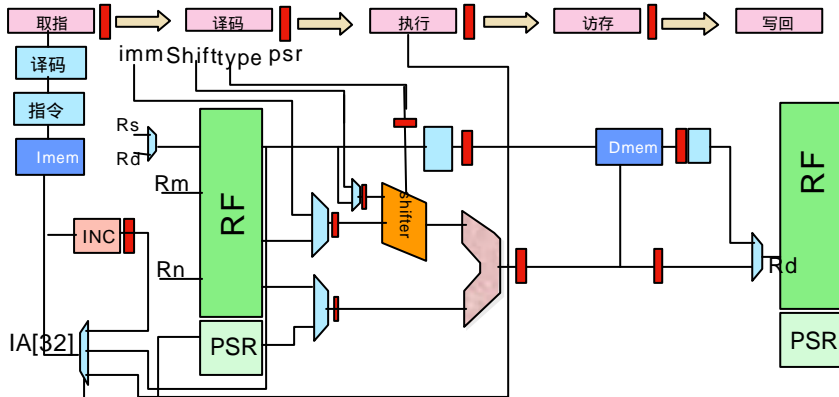
Load/Store byte register offset

当L=1(load)

Logical Register Transfer:

$R_d \leftarrow \text{MEM}(R_n + R_m [\text{shift}] \text{ Shift immediate});$

$PC \leftarrow PC + 4;$ 取数据总线的低8位，高位填0：如何实现。



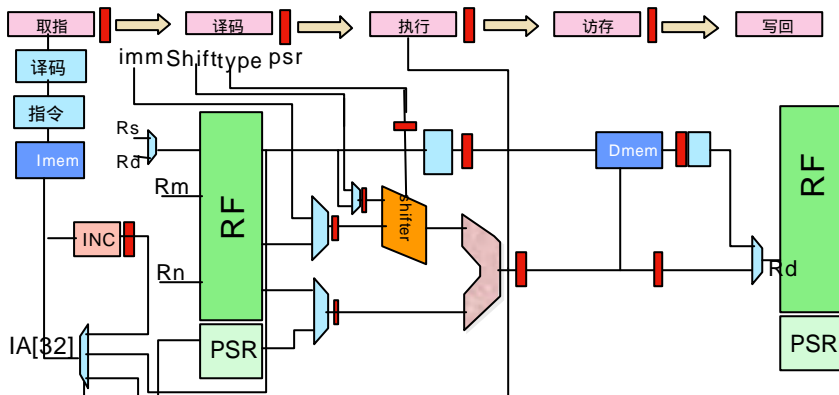
Load/Store byte register offset

当L=0(store)

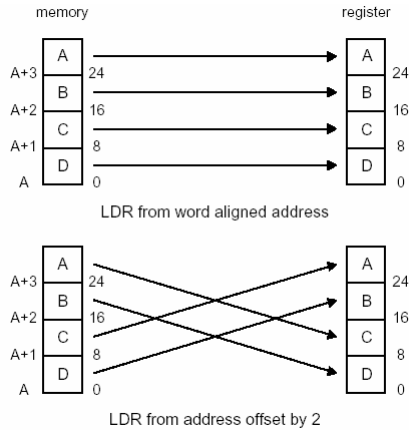
Logical Register Transfer:

$R_d \rightarrow \text{MEM}(R_n + R_m [\text{shift}] \text{ Shift immediate});$

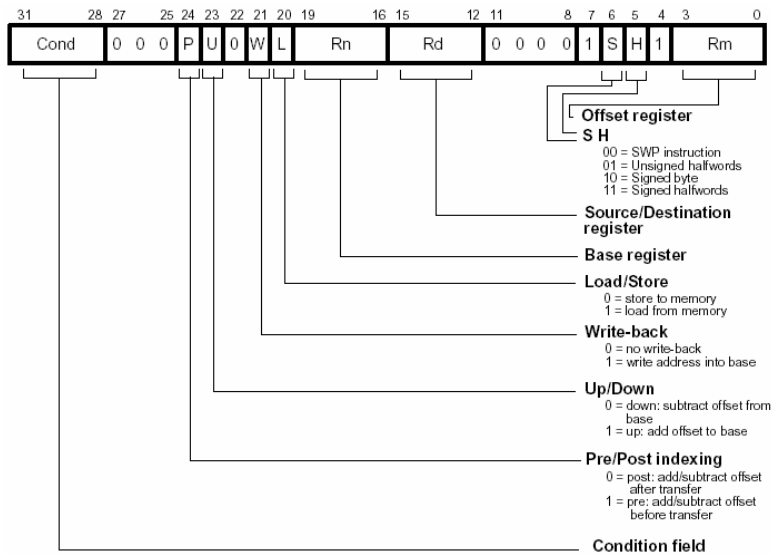
$PC \leftarrow PC + 4;$ 用寄存器的低8位重复填充32为总线，内存负责存储相应的字节操作。



Load/Store word register offset

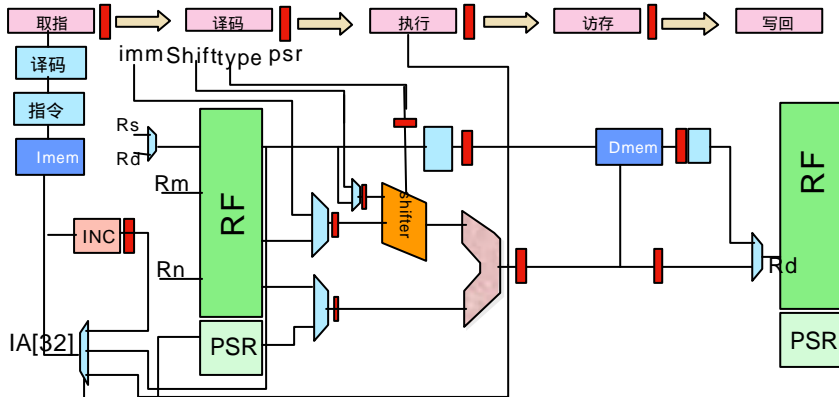


Halfword and Signed Data Transfer

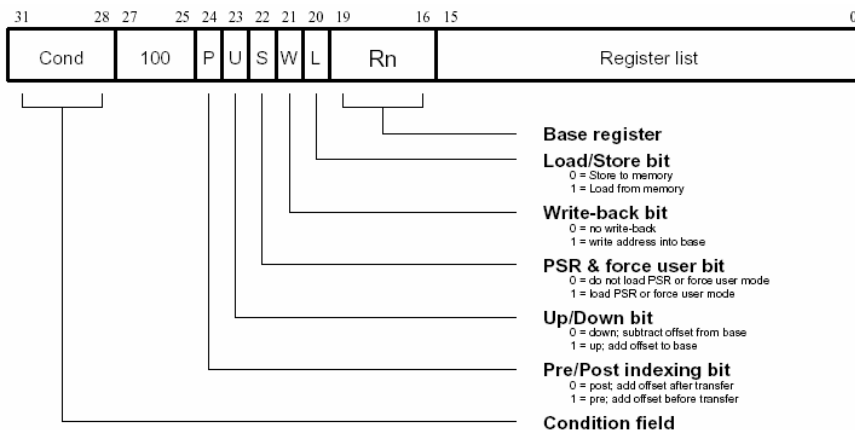


Halfword and Signed Data Transfer

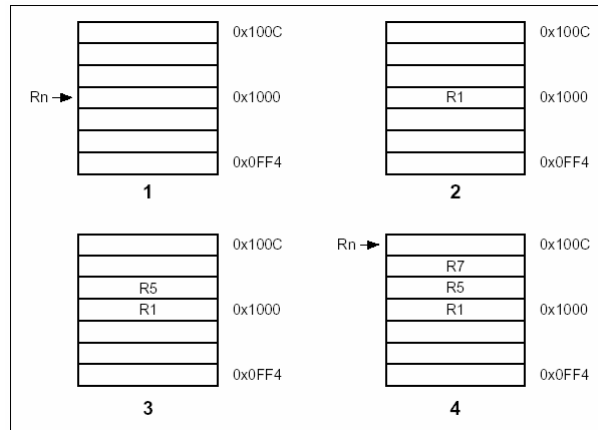
Load符号字节/半字时，高位扩展；无符号数高位补0



Block Data Transfer (LDM, STM)



Block Data Transfer (LDM, STM)



Block Data Transfer (LDM, STM)

The transfer addresses are determined by the contents of the base register (R_n), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R_{15} (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of $R1$, $R5$ and $R7$ in the case where $R_n=0x1000$ and write back of the modified base is required ($W=1$).

Block Data Transfer (LDM, STM)

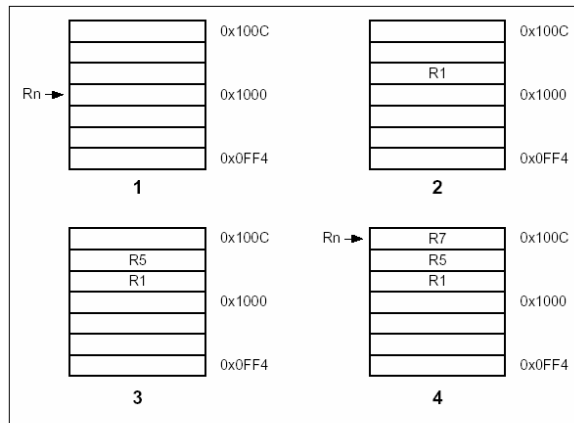


Figure 4-20: Pre-increment addressing

Block Data Transfer (LDM, STM)

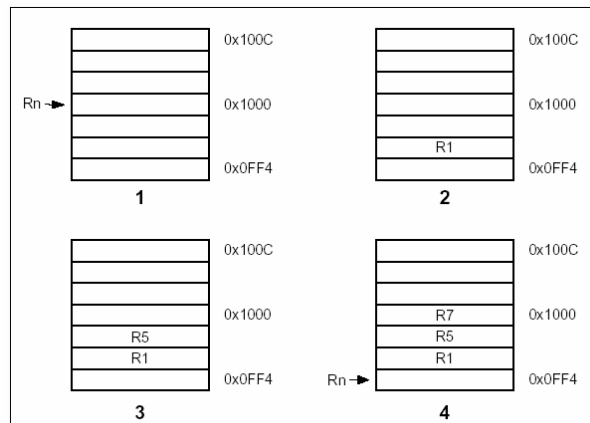


Figure 4-21: Post-decrement addressing

Block Data Transfer (LDM, STM)

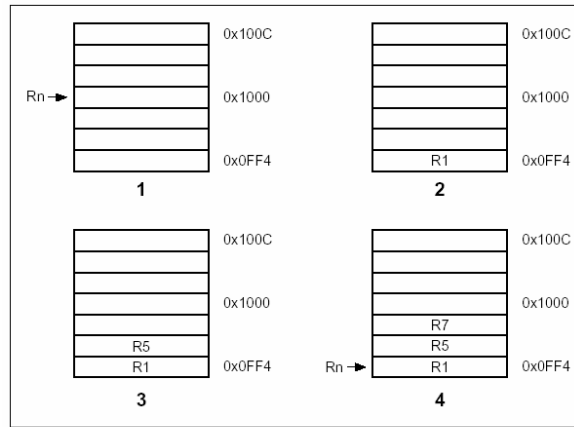


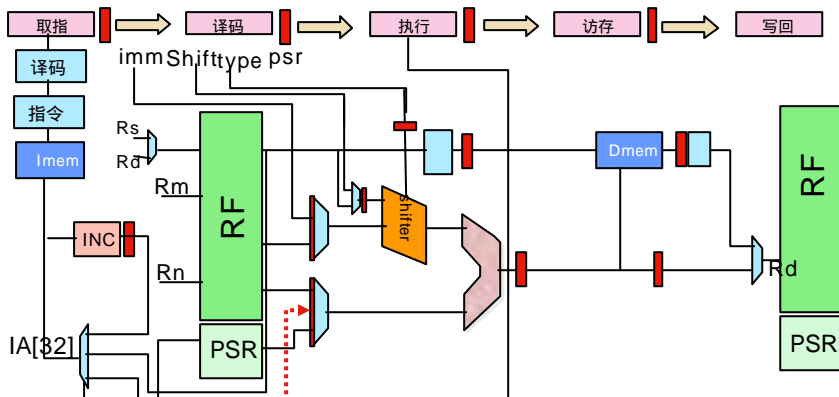
Figure 4-22: Pre-decrement addressing

Block Data Transfer (LDM, STM)

Logical Register Transfer: 当L=0(store)

Temp \leftarrow Rn; When offset[I]='1' then Ri \rightarrow Temp(+/-4); Temp \leftarrow Temp +/- 4;

(Rn \leftarrow Rn +/- ldstnum;); PC \leftarrow PC+4;



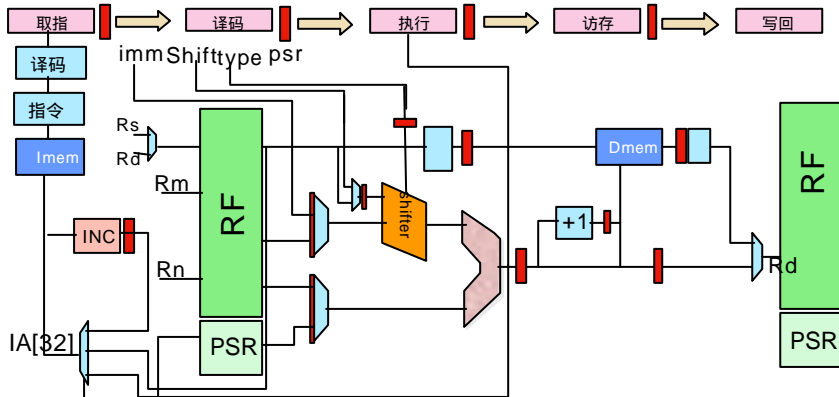
Block Data Transfer (LDM, STM)

Logical Register Transfer: 当L=0(store)

Temp \leftarrow -Rn; When offset[I]='1' then Ri \rightarrow Temp(+/-4); Temp \leftarrow -Temp+/-

4;

(Rn \leftarrow -Rn+/-ldstnum;) ; PC \leftarrow -PC+4;

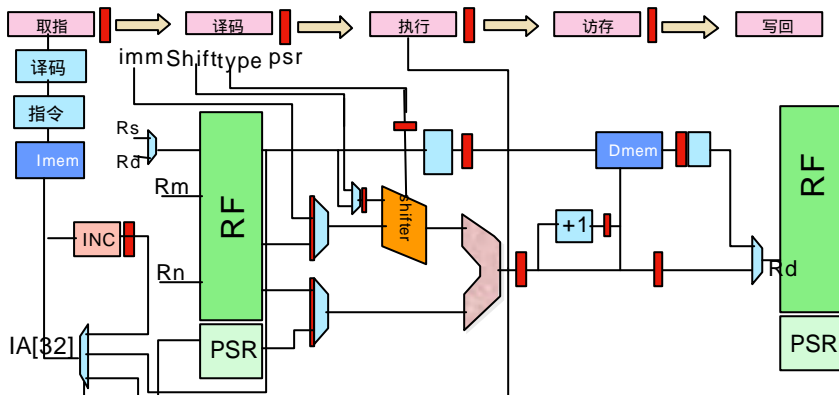


Block Data Transfer (LDM, STM)

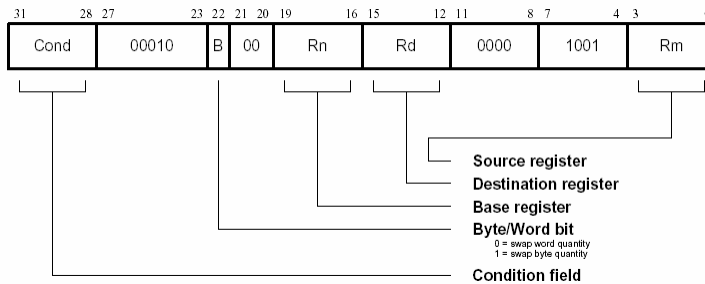
当L=1(Load)Logical Register Transfer:

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

(Rn \leftarrow -Rn+/-ldstnum;) PC \leftarrow -PC+4;



Single Data Swap (SWP)

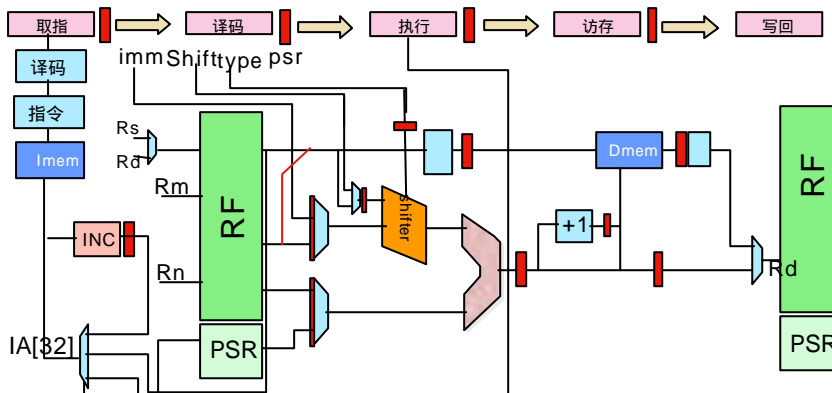


The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together

SWAP Rd,Rm,[Rn]; [Rn]->Rd; Rm->[Rn]

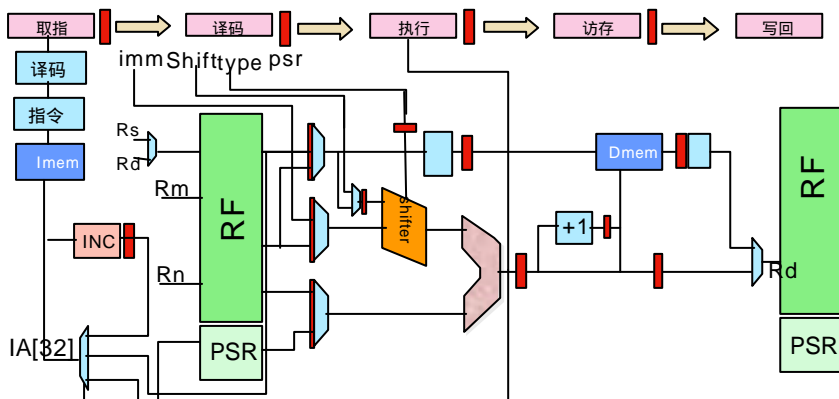
Single Data Swap (SWP)

SWAP Rd,Rm,[Rn]; [Rn]->Rd; Rm->[Rn]



Single Data Swap (SWP)

SWAP Rd,Rm,[Rn]; [Rn]->Rd; Rm->[Rn]



数据通路优化:解决相关

情况1

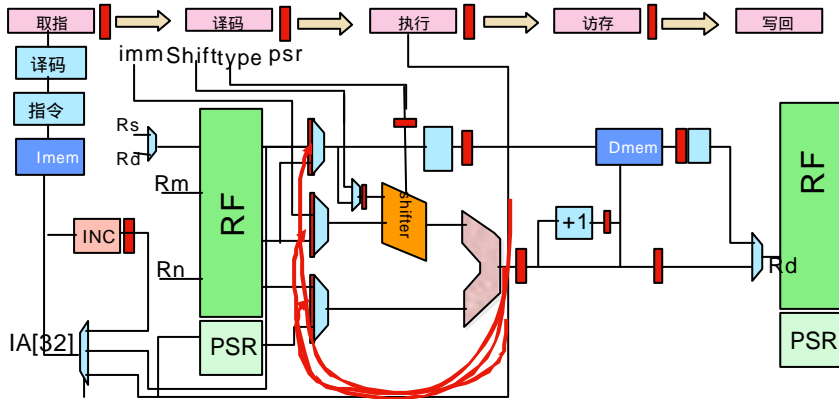
```
sub R2, R1, R3 ; and R12, R2, R5 ; or R13, R6, R2 ;
add R14, R2, R2 ; sw R15, 100(R2)
```



数据通路优化:解决相关

情况1

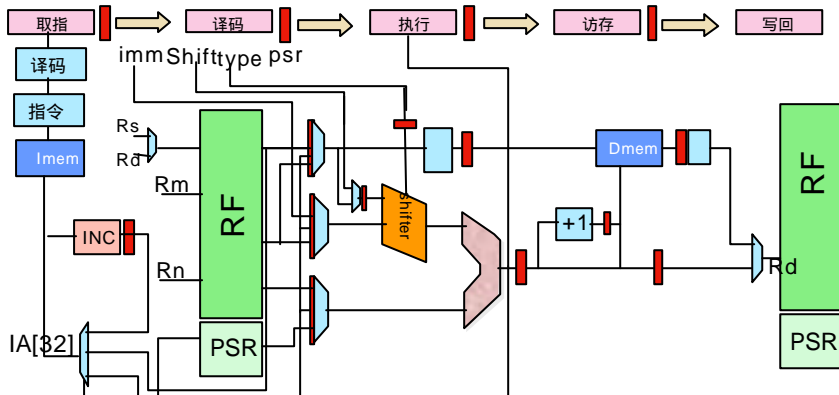
sub R2, R1, R3 ; and R12, R2, R5 ; or R13, R6, R2 ;
add R14, R2, R2 ; sw R15, 100(R2)



数据通路优化:解决相关

情况1

sub R2, R1, R3 ; and R12, R2, R5 ; or R13, R6, R2 ;
add R14, R2, R2 ; sw R15, 100(R2)



数据通路优化:解决相关

情况3

- lw R1, 0(R2)
- sub R4, R1, R5
- and R6, R1, R7
- or R8, R1, R9

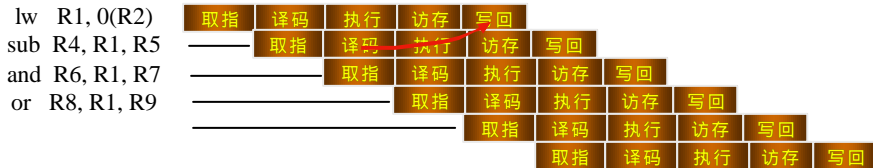
lw R1, 0(R2)

sub R4, R1, R5

and R6, R1, R7

or R8, R1, R9

取指	译码	执行	访存	写回					
取指	译码	执行	访存	写回					
	取指	译码	执行	访存	写回				
		取指	译码	执行	访存	写回			
			取指	译码	执行	访存	写回		
				取指	译码	执行	访存	写回	
					取指	译码	执行	访存	写回



数据通路优化:解决相关

情况3

- lw R1, 0(R2)
- sub R4, R1, R5
- and R6, R1, R7
- or R8, R1, R9

lw R1, 0(R2)

sub R4, R1, R5

and R6, R1, R7

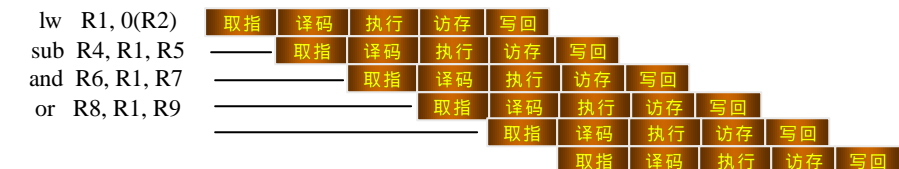
or R8, R1, R9

lw R1, 0(R2)

sub R4, R1, R5

and R6, R1, R7

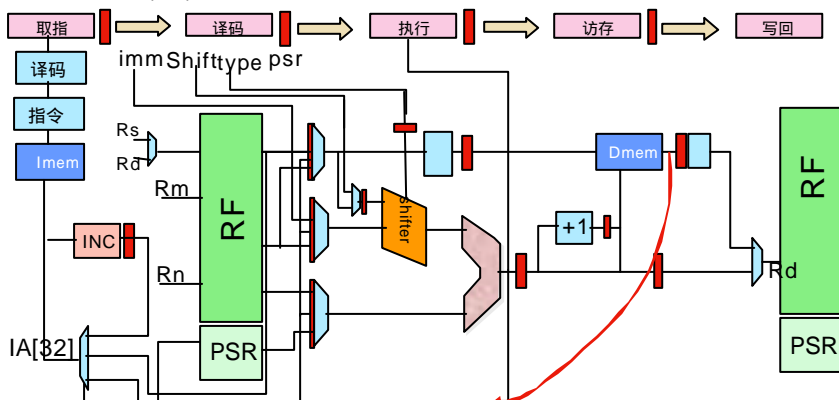
or R8, R1, R9



数据通路优化:解决相关

情况3

- lw R1, 0(R2)
- sub R4, R1, R5
- and R6, R1, R7
- or R8, R1, R9



数据通路优化:解决相关

情况3

- lw R1, 0(R2)
- sub R4, R1, R5
- and R6, R1, R7
- or R8, R1, R9

