

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

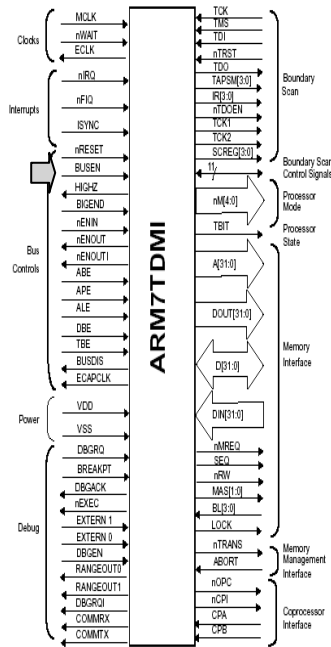
- Thumb状态
- Thumb指令集
- 三级流水数据通路设计

ARMTDMI

Open

BUSEN

Data bus configuration, determines whether the bidirectional data bus, **D[31:0]**, or the unidirectional data busses, **DIN[31:0]** and **DOUT[31:0]**, are to be used for transfer of data between the processor and memory. When **BUSEN** is LOW, the bidirectional data bus, **D[31:0]** is used. In this case, **DOUT[31:0]** is driven to value 0x00000000, and any data presented on **DIN[31:0]** is ignored. When **BUSEN** is HIGH, the bidirectional data bus, **D[31:0]** is ignored and must be left unconnected. Input data and instructions are presented on the input data bus, **DIN[31:0]**, output data appears on **DOUT[31:0]**.



ARMTDMI

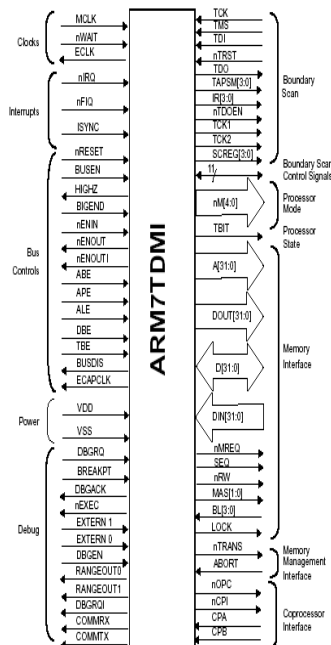
Open

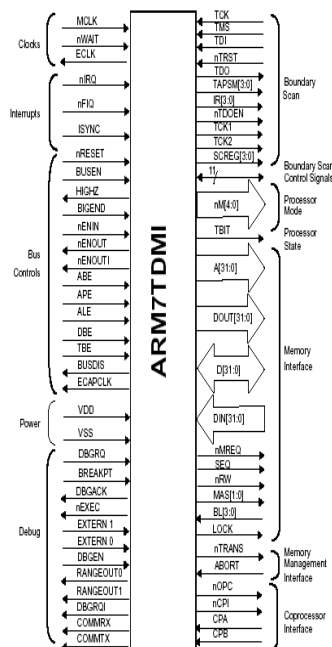
ISYNC

Synchronous interrupts. LOW indicates that the **nIRQ** and **nFIQ** inputs are to be synchronised by the ARM core. When HIGH disables this synchronisation for inputs that are already synchronous.

LOCK

Locked operation. When **LOCK** is HIGH, the processor is performing a "locked" memory access, and the memory controller must wait until **LOCK** goes LOW before allowing another device to access the memory. **LOCK** changes while **MCLK** is HIGH, and remains HIGH for the duration of the locked memory accesses. It is active only during the data swap (SWP) instruction.





ARMTDMI

nMREQ

Not memory request.

This signal, when LOW, indicates that the processor requires memory access during the following cycle.

nOPC

Not op-code fetch.

When LOW this signal indicates that the processor is fetching an instruction from memory; when HIGH, data (if present) is being transferred.

nRESET

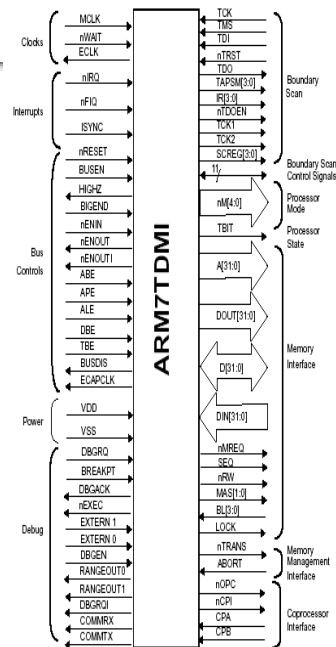
Not reset.

This is a level sensitive input signal which is used to start the processor from a known address. A LOW level will cause the instruction being executed to terminate abnormally.

nRW

Not read/write.

When HIGH this signal indicates a processor write cycle; when LOW, a read cycle.

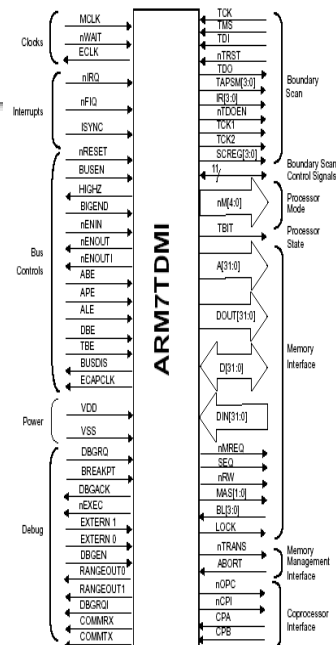


ARMTDMI

nWAIT

Not wait.

When accessing slow peripherals, ARM7TDMI can be made to wait for an integer number of **MCLK** cycles by driving **nWAIT** LOW. Internally, **nWAIT** is ANDed with **MCLK** and must only change when **MCLK** is LOW. If **nWAIT** is not used it must be tied HIGH.

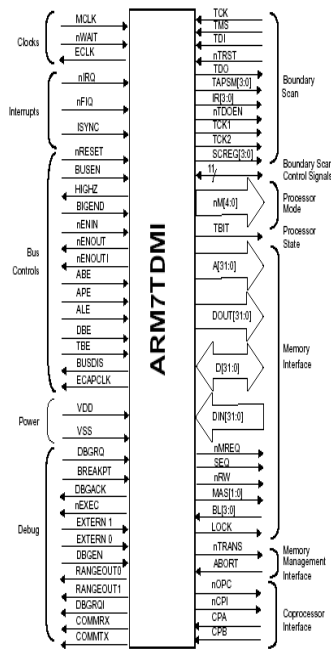


ARMTDMI

SEQ

Sequential address.

This output signal will become HIGH when the address of the next memory cycle will be related to that of the last memory access. The new address will either be the same as the previous one or 4 greater in ARM state, or 2 greater in THUMB state. The signal becomes valid during phase 1 and remains so through phase 2 of the cycle before the cycle whose address it anticipates. It may be used, in combination with the low-order address lines, to indicate that the next cycle can use a fast memory mode (for example DRAM page mode) and/or to bypass the address translation system.



ARMTDMI

VDD

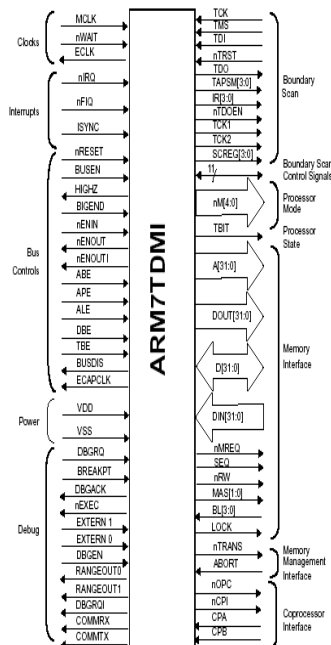
Power supply.

P These connections provide power to the device.

VSS

Ground.

P These connections are the ground reference for all signals.



Thumb状态

From the programmer's point of view, the ARM7TDMI can be in one of two states:

ARM state which executes 32-bit, word-aligned ARM instructions.

THUMB state which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

Note Transition between these two states does not affect the processor mode or the contents of the registers.

Entering THUMB state

- Entry into THUMB state can be achieved by executing a BX instruction with the state
 - bit (bit 0) set in the operand register.
- Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM state

- Entry into ARM state happens:
- 1 On execution of the `BX` instruction with the state bit clear in the operand register.
- 2 On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.).
- In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

Thumb寄存器

THUMB State General Registers and Program Counter

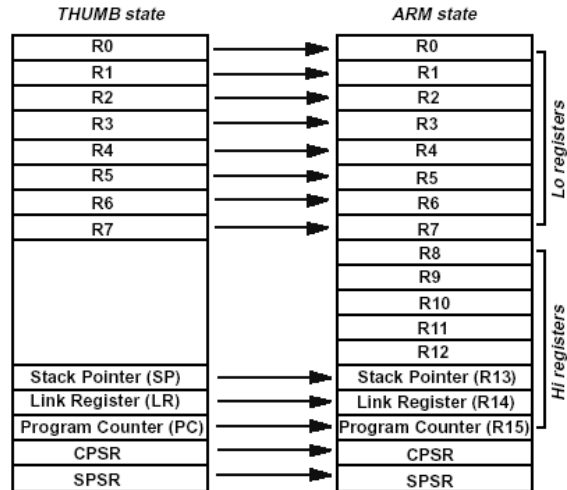
System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

THUMB State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

Thumb寄存器



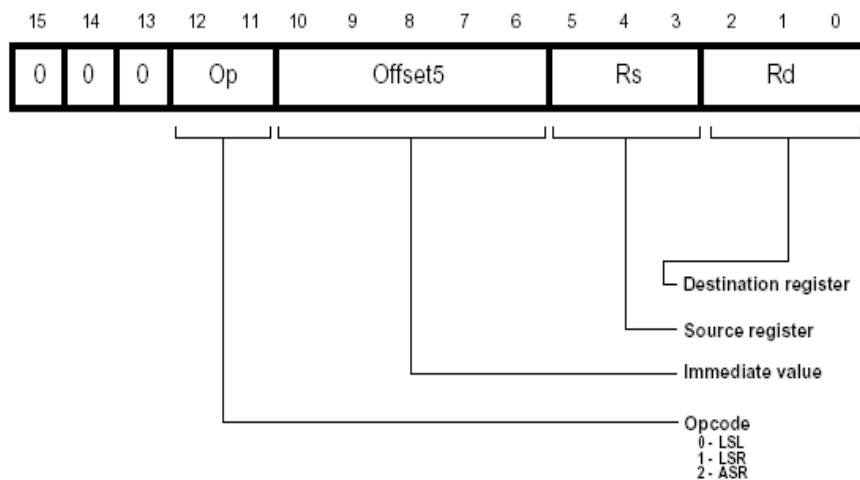
Thumb寄存器

- In THUMB state, registers R8-R15 (the Hi registers) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.
- A value may be transferred from a register in the range R0-R7 (a Lo register) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions.

Thumb寄存器

	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	I	Op	Rn/offset3			Rs			Rd			Add/subtract	
3	0	0	1	Op	Rd				Offset8								Move/compare/load /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					Softset8							Conditional branch	
17	1	1	0	1	1	1	1	1	Value8								Software interrupt	
18	1	1	1	0	0	Offset11												Unconditional branch
19	1	1	1	1	H	Offset												Long branch with link
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		

Thumb指令集：Format 1: move shifted register



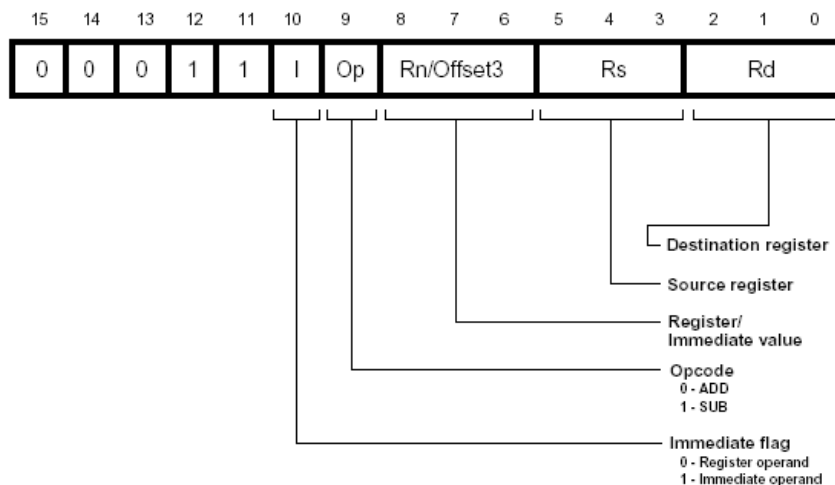
Thumb指令集：Format 1: move shifted register

OP	THUMB assembler	ARM equivalent	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

Examples

LSR R2, R5, #27 ; Logical shift right the contents
 ; of R5 by 27 and store the result in R2.
 ; Set condition codes on the result.

Format 2: add/subtract



Format 2: add/subtract

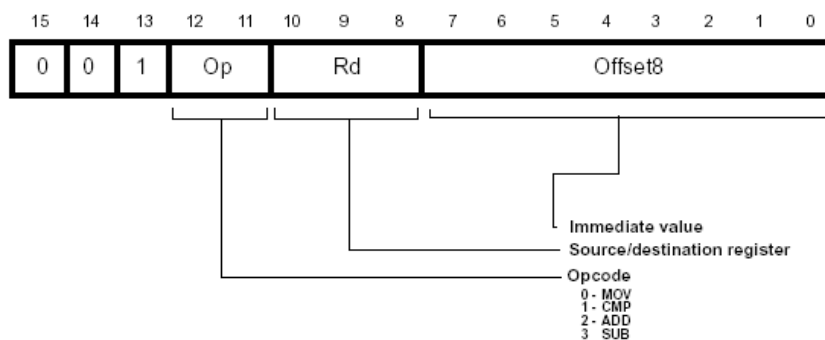
Op	I	THUMB assembler	ARM equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

Examples

ADD R0, R3, R4 ; $R0 := R3 + R4$ and set condition codes on the result.

SUB R6, R2, #6 ; $R6 := R2 - 6$ and set condition codes.

move/compare/add/subtract immediate



The instructions in this group perform operations between a Lo register and an 8-bit immediate value.

move/compare/add/subtract immediate

Op	THUMB assembler	ARM equivalent	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

Examples

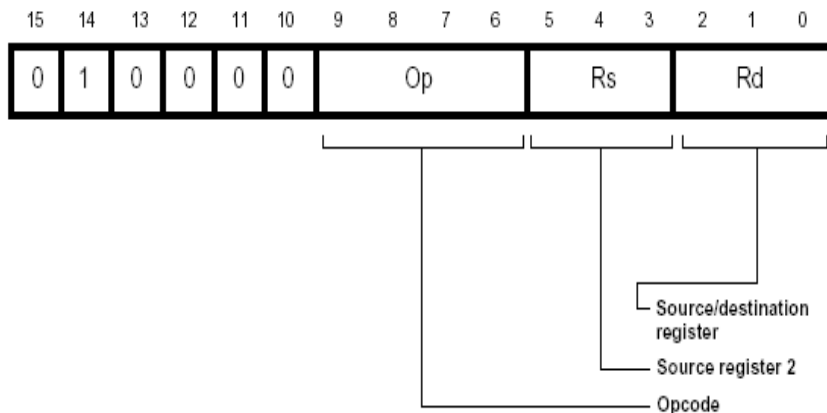
MOV R0, #128 ; R0 := 128 and set condition codes

CMP R2, #62 ; Set condition codes on R2 - 62

ADD R1, #255 ; R1 := R1 + 255 and set condition codes

SUB R6, #145 ; R6 := R6 - 145 and set condition codes

ALU operations



ALU operations

OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd := Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd := Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = -Rs

ALU operations

OP	THUMB assembler	ARM equivalent	Action
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVNS Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

ALU operations

Examples

EOR R3, R4 ; R3 := R3 EOR R4 and set condition codes

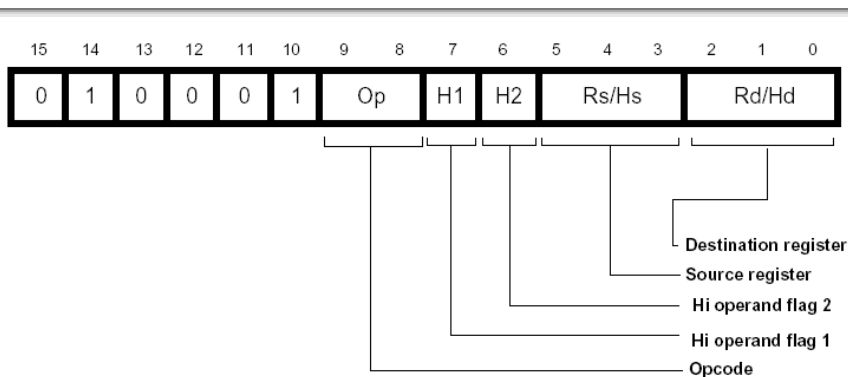
ROR R1, R0 ; Rotate Right R1 by the value in R0, store
; the result in R1 and set condition codes

NEG R5, R3 ; Subtract the contents of R3 from zero,
; store the result in R5. Set condition codes
; ie R5 = -R3

CMP R2, R6 ; Set the condition codes on the result of
; R2 - R6

MUL R0, R7 ; R0 := R7 * R0 and set condition codes

Hi register operations/branch exchange



Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15

Hi register operations/branch exchange

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

Hi register operations/branch exchange

The BX instruction

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

Bit 0 = 0 causes the processor to enter ARM state.

Bit 0 = 1 causes the processor to enter THUMB state.

Note The action of H1 = 1 for this instruction is undefined, and should not be used.

Hi register operations/branch exchange

Hi register operations

ADD PC, R5 ; PC := PC + R5 but don't set the
; condition codes.

CMP R4, R12 ; Set the condition codes on the
; result of R4 - R12.

MOV R15, R14 ; Move R14 (LR) into R15 (PC)
; but don't set the condition codes,
; eg. return from subroutine.

Hi register operations/branch exchange

Branch and exchange

; Switch from THUMB to ARM state.

ADR R1,outofTHUMB ; Load address of outofTHUMB into R1.

MOV R11,R1

BX R11 ; Transfer the contents of R11 into the PC.

; Bit 0 of R11 determines whether

; ARM or THUMB state is entered, ie.

; ARM state here.

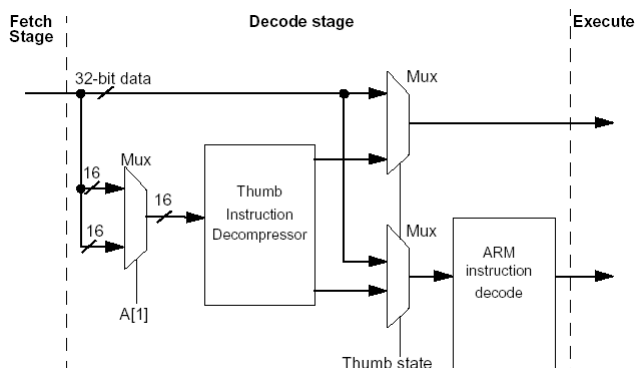
...

ALIGN

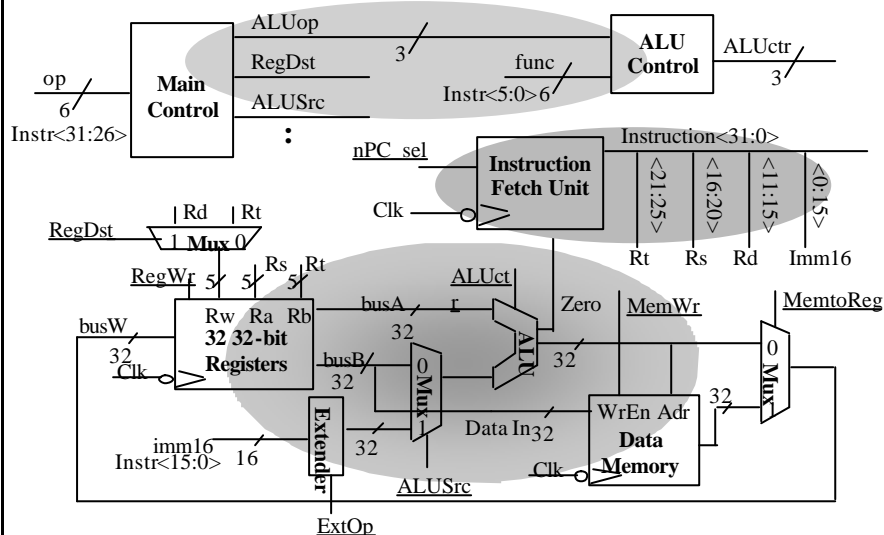
CODE32

outofTHUMB ; Now processing ARM instructions...

Thumb与ARM共享数据通路



ARM流水线设计:单周期流水





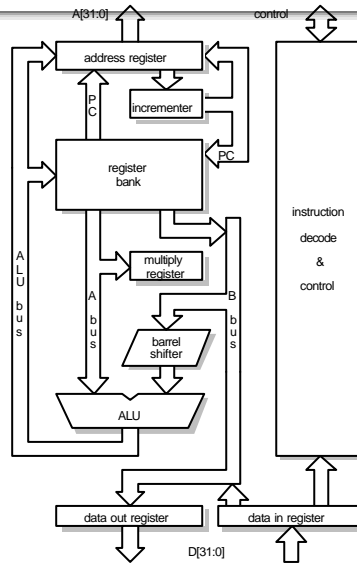
- _____



ARM7三级流水

■ ARM7

- 两个主要的模块：数据通路和译码器
- 寄存器堆 (r0 to r15)
- 两个读口分别对应A- bus/ B-bus
- 一个写口，来自ALU- bus
- 另外的读写口，对程序计数器r15操作
- 桶式移位器Barrel shifter / ALU
- 地址寄存器/增加器
 - 单个内存端口
 - 保存PC地址 (with increment) 或操作数地址

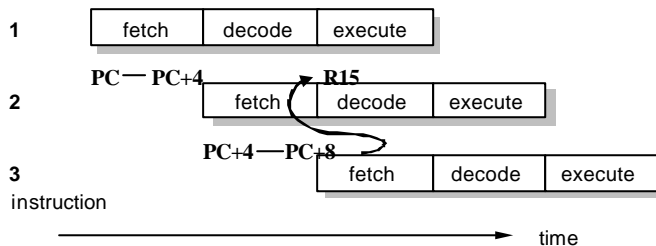


ARM7三级流水

■ ARM7 (续)

■ 流水线: 3 Stage pipeline

- **Fetch** : fetch instruction code from memory into the instruction pipeline
- **Decode** : instruction decoded to obtain control signals for the datapath ready for the next stage
- **Execute** : instruction “owns” the datapath - register read; shifting; ALU results generated and write - back

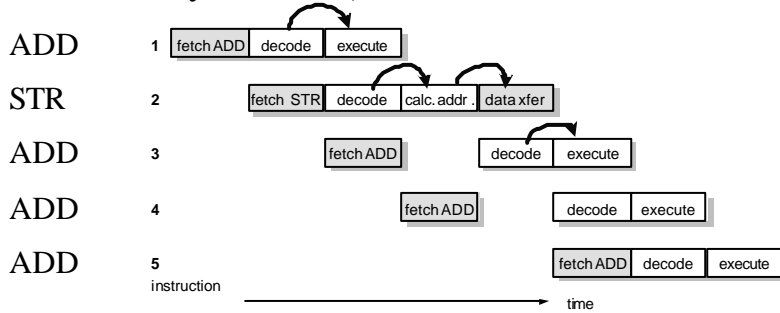


ARM7三级流水

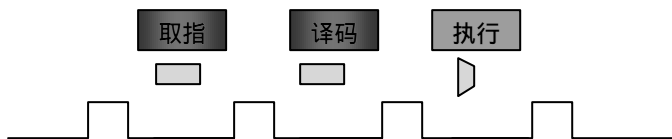
■ ARM7(cont'd)

■ Multi-cycle operation

- Single cycle throughput for almost simple data processing instruction
- Multi-cycle for mul, load/store



流水段的功能设计



分配给每个流水段的部件时延小于时钟周期

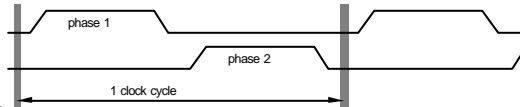
流水线相关

流水线断流会导致其效率大大降低,流水段越多,其损失越大.P4的Netburst结构.

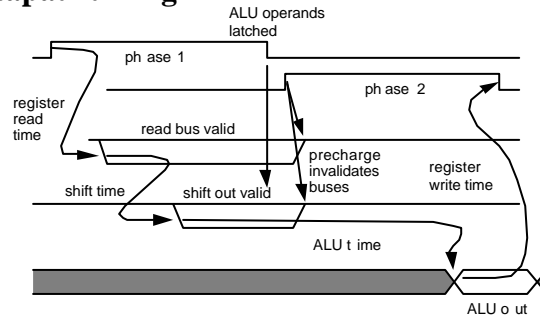
ARM7三级流水实现：

■ ARM7(cont'd)

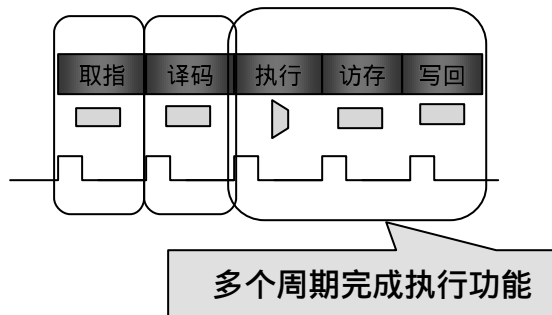
■ 2 Phase Non-overlapping clocking scheme



■ Datapath timing

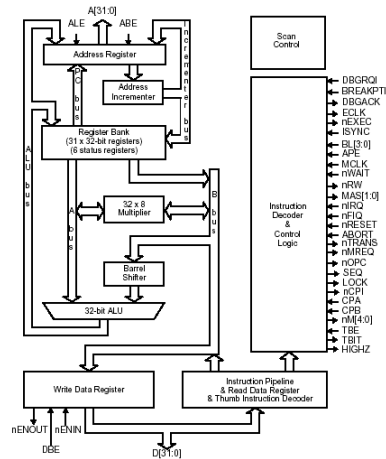


多周期执行



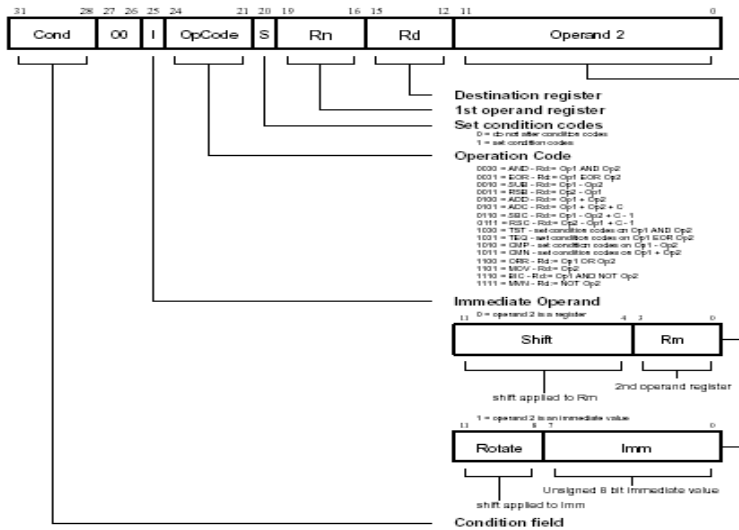
与五级流水的区别：执行部件执行时，其它部件停顿

ARM7的设计



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	1	Opcode			S	Rn		Rd		Operand 2															Data Processing / PSR Transfer				
Cond		0	0	0	0	0	0	A	S	Rd		Rn		Rs		1	0	0	1	Rm		Multiply										
Cond		0	0	0	0	1	U	A	S	RdHi		RdLo		Rn		1	0	0	1	Rm		Multiply Long										
Cond		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	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn		Branch and Exchange						
Cond		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	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	

ARM7的设计：数据处理指令

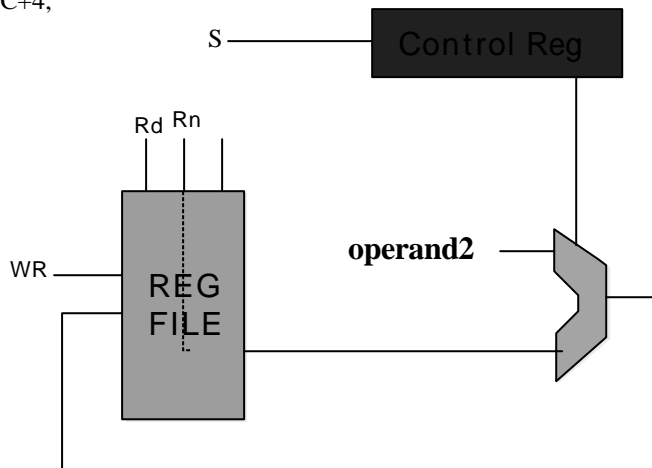


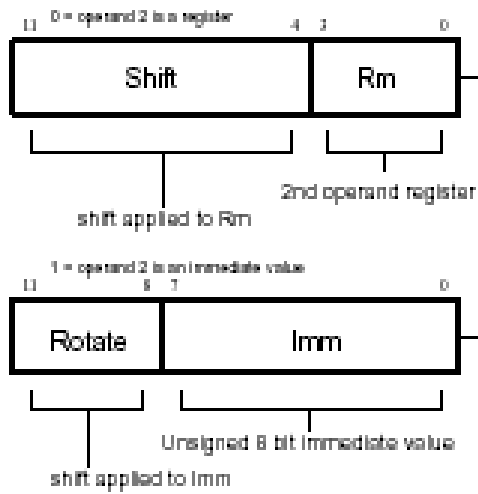
ARM7的设计：数据处理指令

Logical Register Transfer:

Rd ← Rn [opcode] (Rm [shift] ShiftImmediate);

PC ← PC + 4;



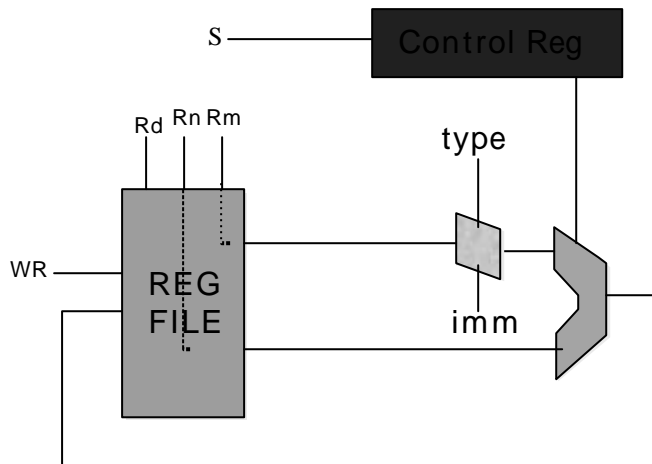


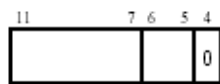
operand2

Logical Register Transfer:

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

$PC \leftarrow PC + 4$;





Shift type
 00 = logical left
 01 = logical right
 10 = arithmetic right
 11 = rotate right

Shift amount
 5 bit unsigned integer



Shift type
 00 = logical left
 01 = logical right
 10 = arithmetic right
 11 = rotate right

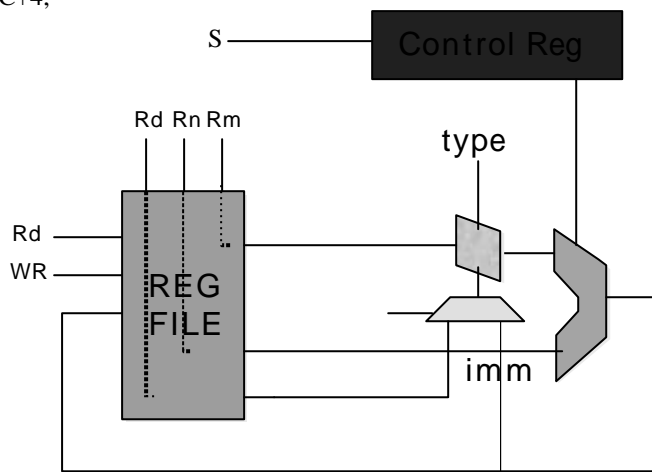
Shift register
 Shift amount specified in bottom byte of Rs

数据处理

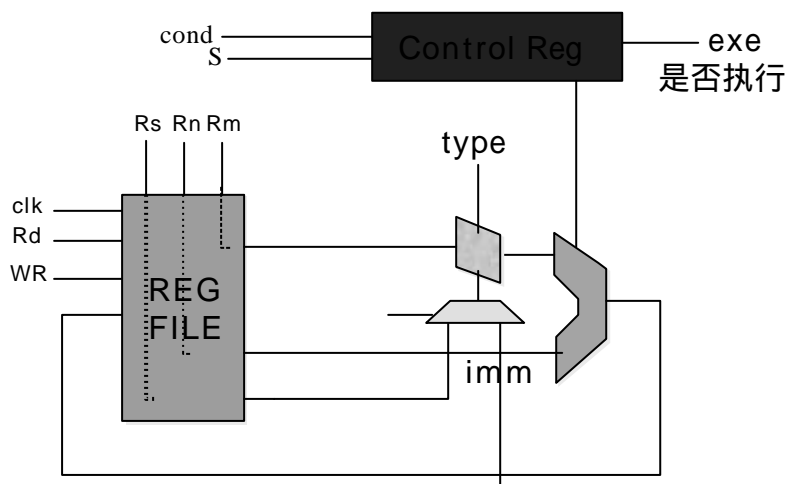
Logical Register Transfer:

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

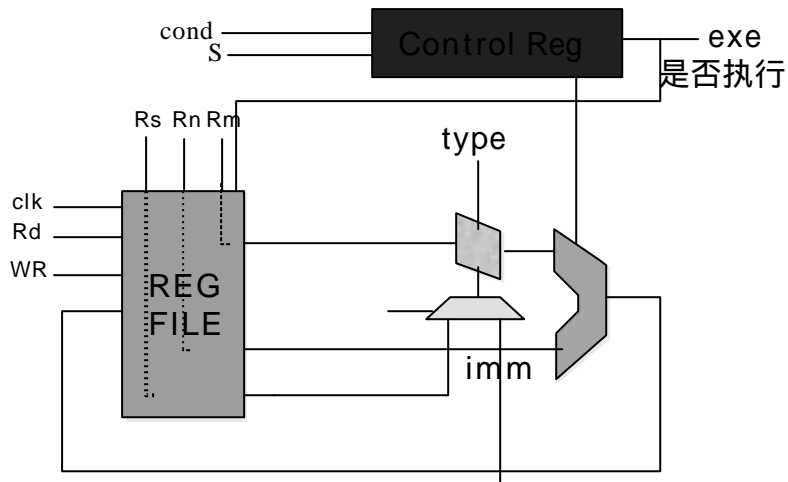
$PC \leftarrow PC + 4;$



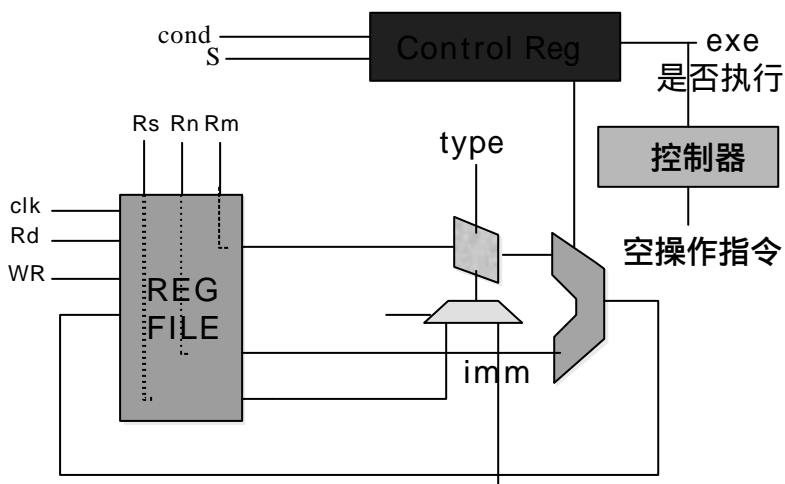
条件执行



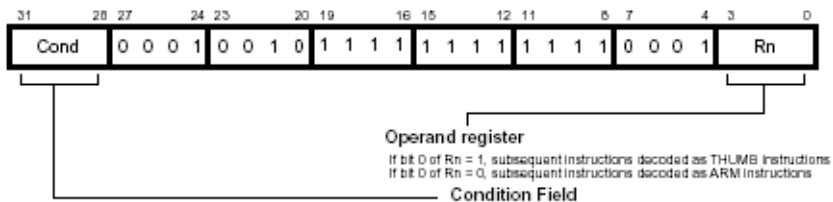
如何实现条件执行：方法1



如何实现条件执行：方法2



转移指令



If bit 0 of Rn = 1, subsequent instructions
 decoded as THUMB instructions
 If bit 0 of Rn = 0, subsequent instructions
 decoded as ARM instructions

转移指令

Logical Register Transfer:

PC \leftarrow Rm;

If Rm[0]=1 then

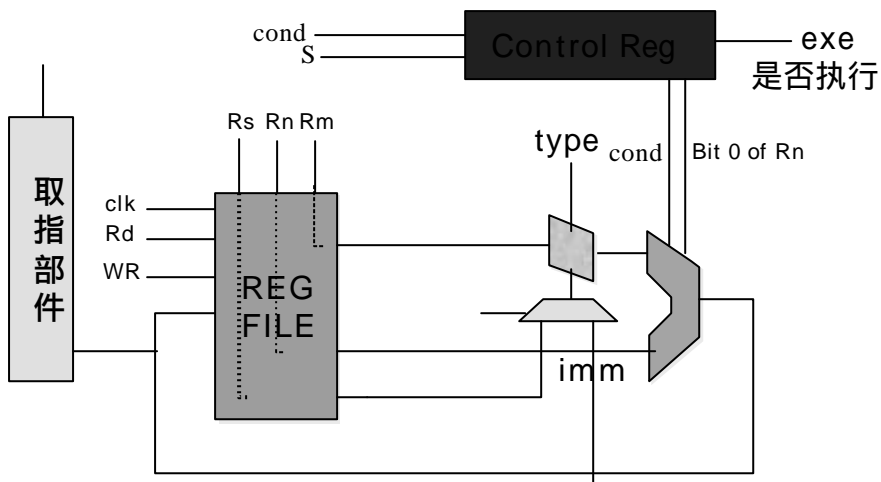
Change into JBCore16;

Else

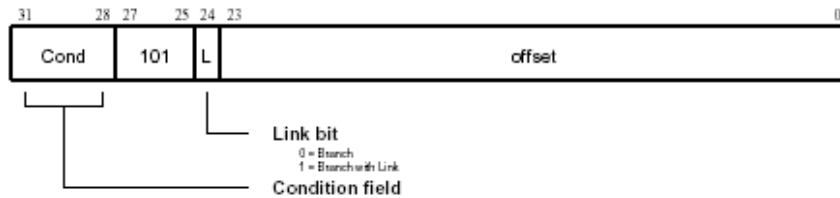
Chang into JBCore32;

End if

转移指令：PC \leftarrow -Rn



Branch with Link

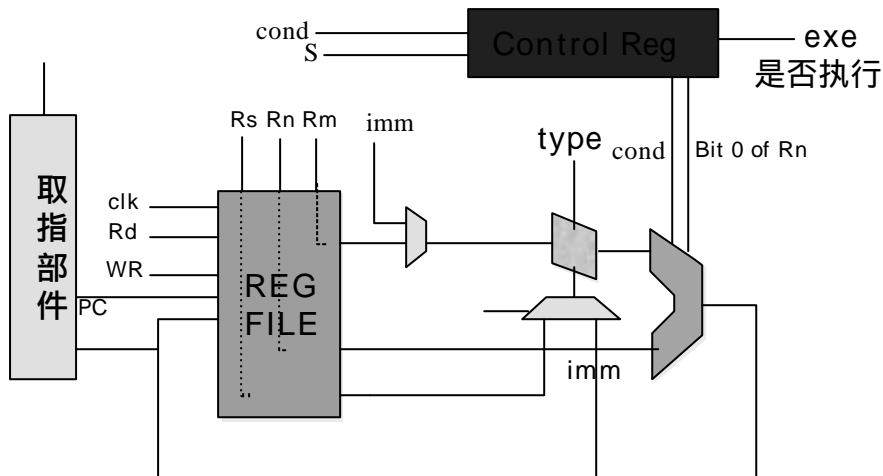


The link bit

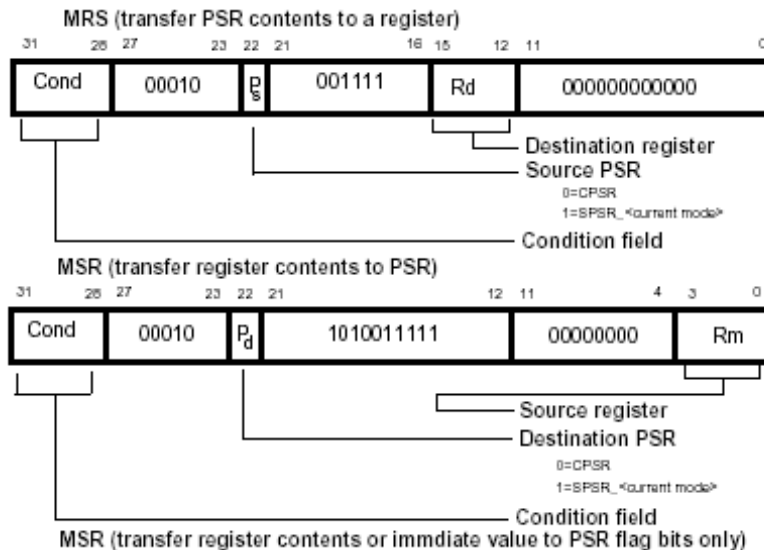
Branch with Link (BL) writes the old PC into the link register (R14) of the current bank.

The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

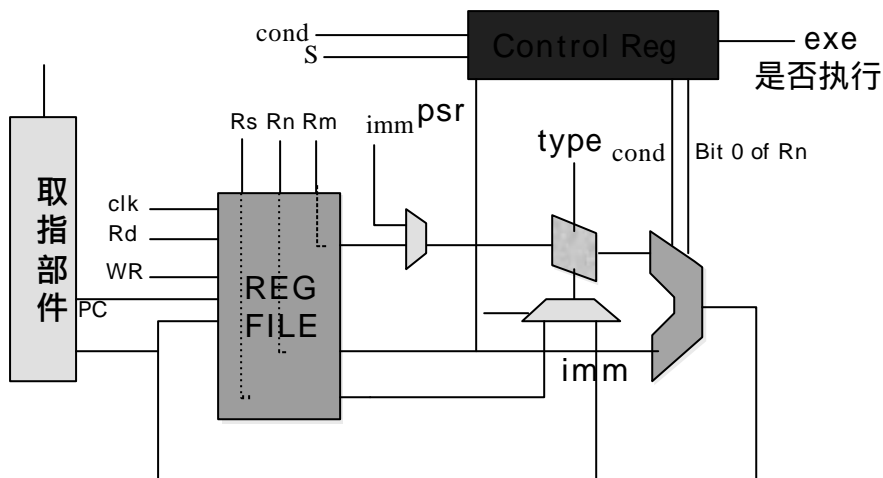
oldPC → R14; imm --> PC



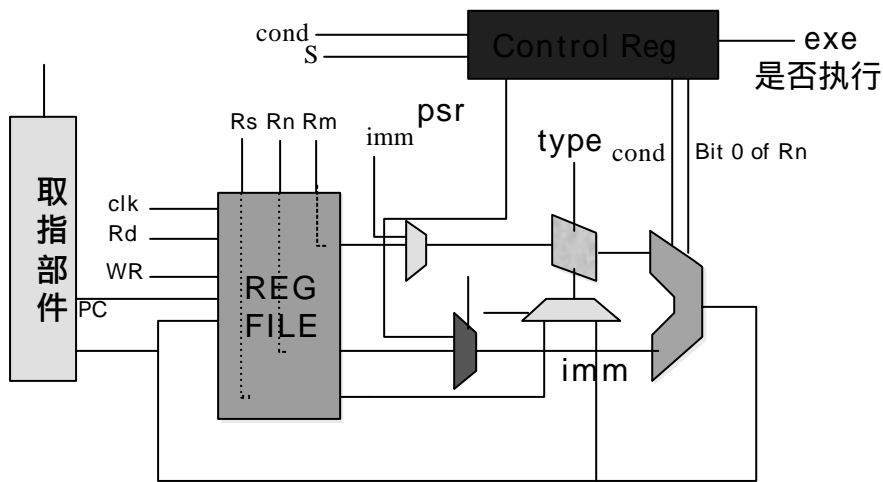
PSR Transfer (MRS, MSR)



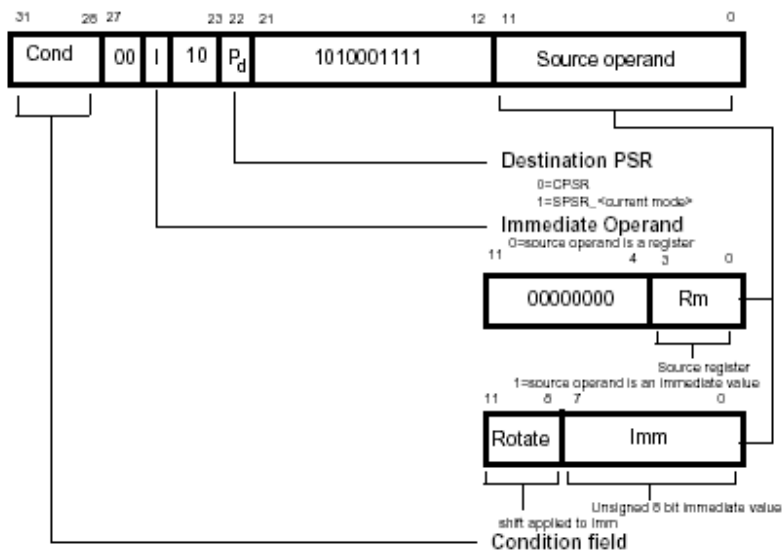
PSR->Rd



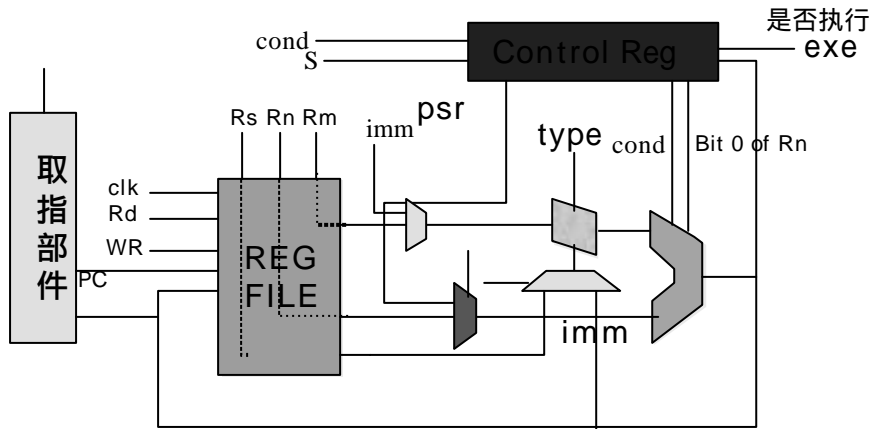
PSR->Rd



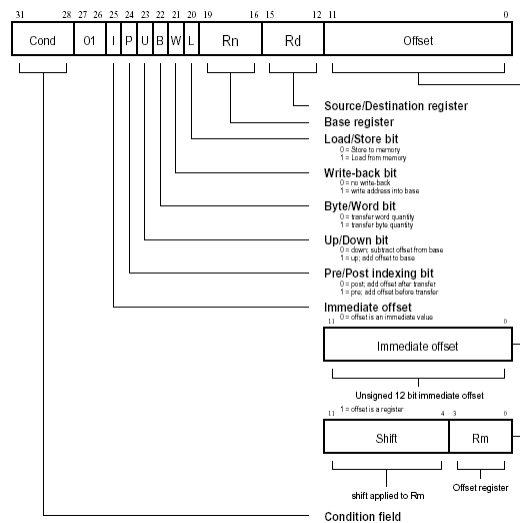
Psr<-source



Psr<-source



Single Data Transfer (LDR, STR)



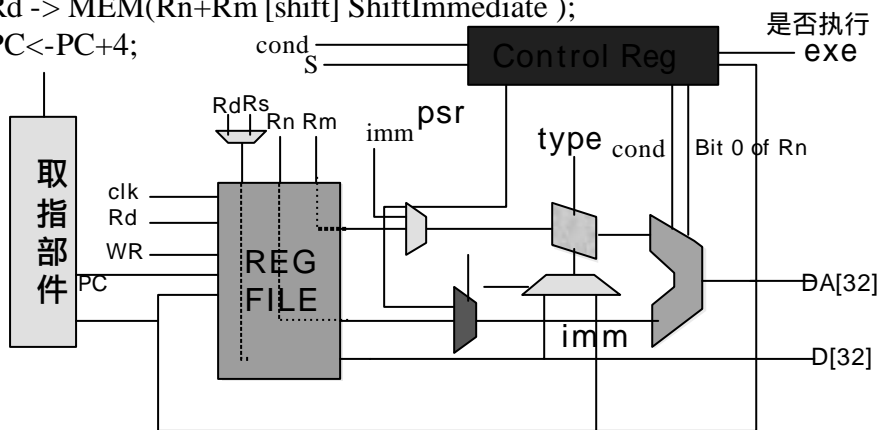
Load/Store register offset

当L=0(store)

Logical Register Transfer:

$Rd \rightarrow MEM(Rn + Rm \text{ [shift] ShiftImmediate});$

$PC \leftarrow PC + 4;$



Load/Store register offset

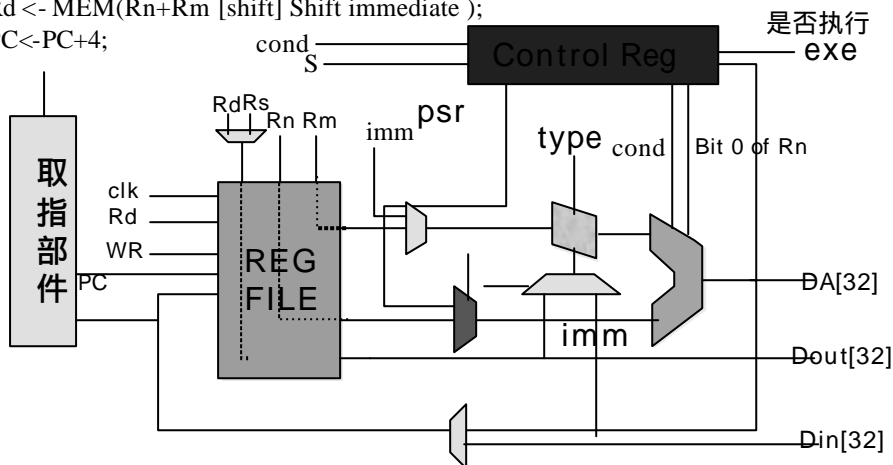
当L=1(load)

能否实现寄存器移位量？

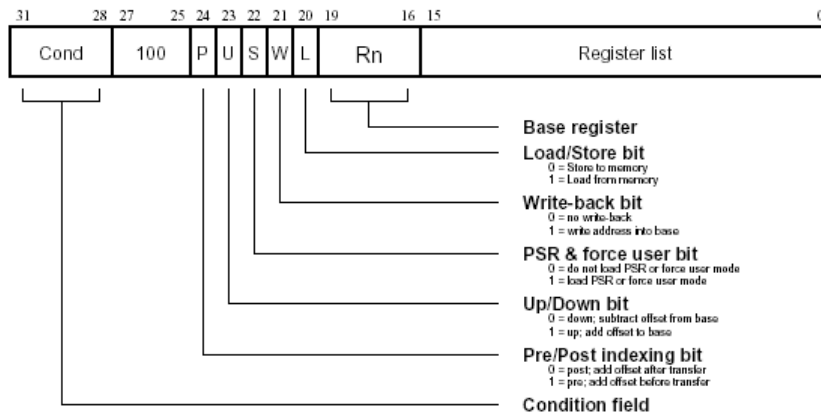
Logical Register Transfer:

$Rd \leftarrow MEM(Rn + Rm \text{ [shift] Shift immediate});$

$PC \leftarrow PC + 4;$



Block Data Transfer (LDM, STM)



Block Data Transfer (LDM, STM)

当L=0(store)

Logical Register Transfer:

Temp ← Rn;

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

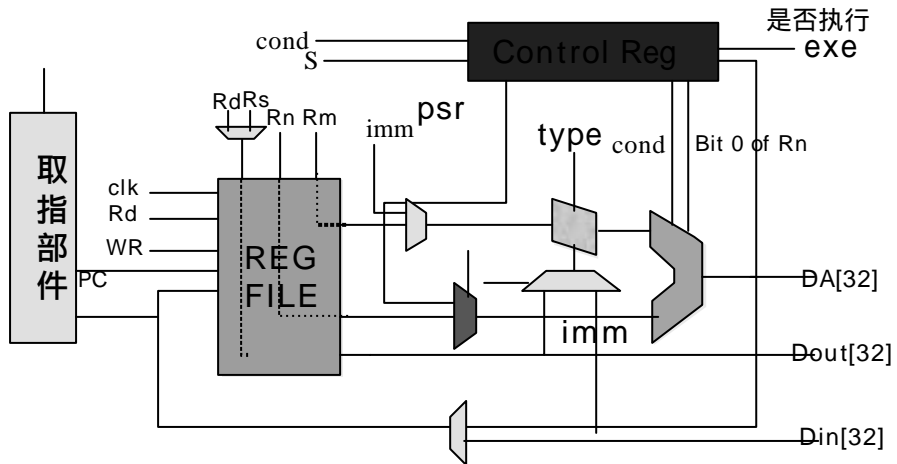
Temp ← Temp +/- 4;

(Rn ← Rn +/- ldstnum;)

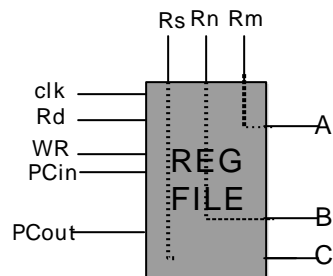
PC ← PC + 4;

Block Data Transfer (LDM, STM)

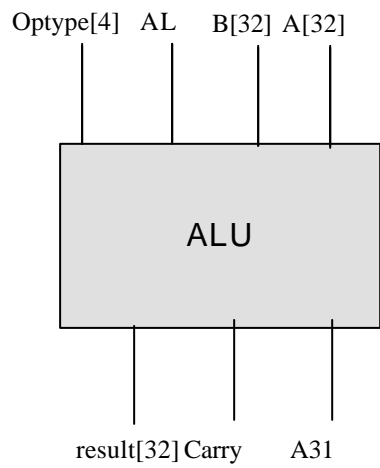
MEM[Rn]<-R list



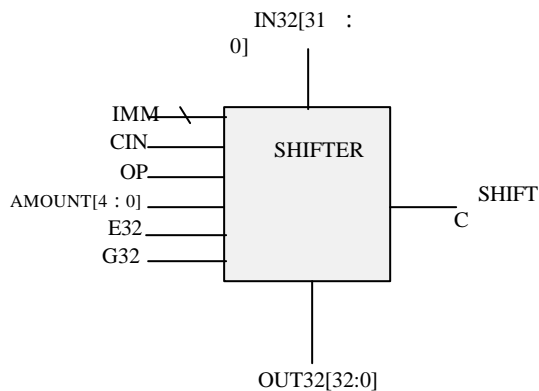
模块设计：寄存器堆



ALU设计

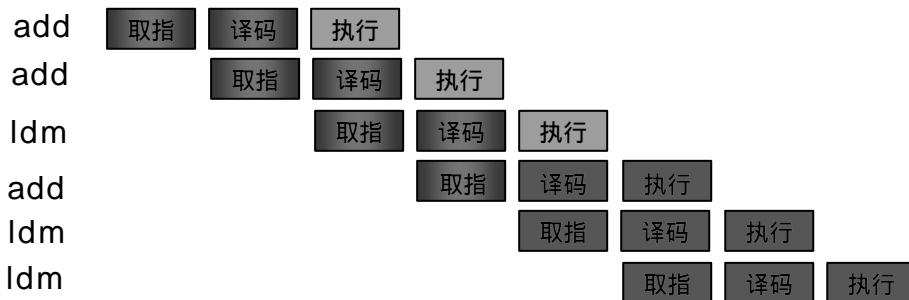


Shifter设计



模块设计文档

控制器的设计：多周期指令实现



Ldm发生数据失效如何处理？

ADD r0,r1

Sub r2,r0

三级流水中是否会正确执行？