ARM体系结构

李曦

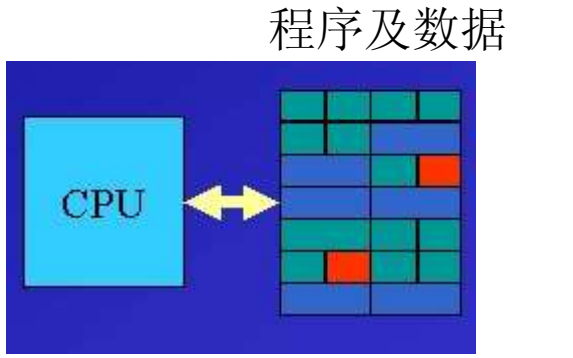
llxx@ustc.edu.cn

内容提要

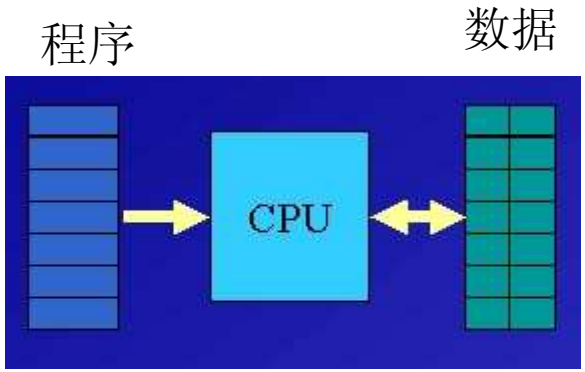
- **ARM**体系结构概览
  - 嵌入式微处理器体系结构
  - **ARM**历史
  - **ARM**体系结构特征
  - **ARM**片上总线**AMBA**
  - **ARM**对调试的支持
  - 虚存管理
- **ARM**编程模型
  - **ARM**微处理器的工作状态
  - **ARM**体系结构的存储器模式
  - **ARM**微处理器的操作模式
  - **ARM**体系结构的寄存器组织
  - **ARM**微处理器的异常状态

计算机体系结构

- 冯.诺依曼结构（von Neumann arch）
  - 系统组成：ALU，Controller，Memory，Input，Output
  - 程序指令和数据放在同一存储器的不同物理位置，CPU通过同一条总线访问程序和数据，程序指令和数据的宽度是相同的
  - 存储程序(stored program)：程序以数字形式存在，可以与数据一样被读写



- 哈佛体系结构（Harvard architecture）
  - 程序与数据有单独的存储器，它们通过不同的总线来访问
    - 首先到程序指令存储器中读取程序指令，解码后得到数据地址，再到相应的数据存储器中读取数据，执行指令。在执行一条指令的同时可以预先读取下一条指令
  - 指令和数据可以有不同的数据宽度
    - 如PIC的程序指令是14位，而数据是8位



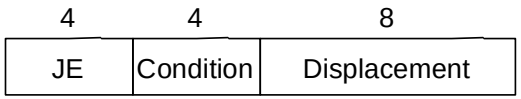
- 指令集体系结构
  - CISC、RISC、VLIW

复杂指令集计算机CISC

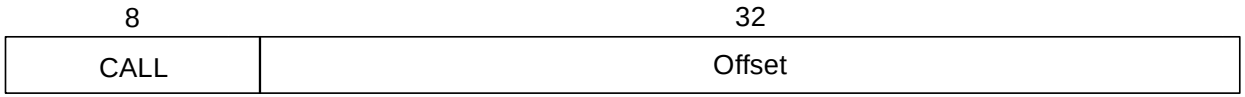
- 背景：
  - 存储资源紧缺，设置一些功能复杂的指令以减少完成一个任务所需的指令数目
  - 增强指令功能，强调编译优化，通过减少程序中指令数达到提高运行速度的目的
- 特点：
  - 指令格式不固定，指令可长可短，操作数可多可少
  - 寻址方式复杂多样，操作数可来自寄存器，也可来自存储器
  - 使用微代码。指令集存储在控存里（比主存的速度快很多）
  - 允许设计师实现CISC体系机器的向上相容
    - 新的系统可以使用一个包含早期系统的指令超集
  - 微程式指令的格式与高阶语言相匹配，因而编译器的设计较简单
  - CPI > 5，指令越复杂，CPI越大
  - 中断响应时间难于预期，实时性不好

X86指令格式

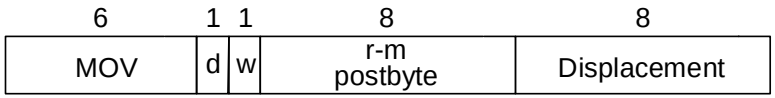
a. JE EIP + displacement



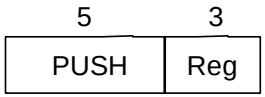
b. CALL



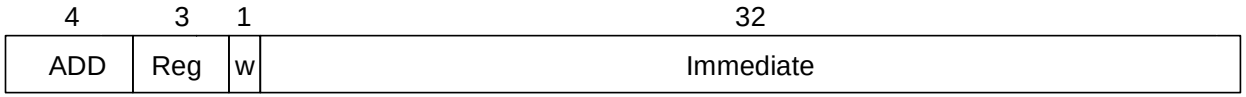
c. MOV EBX, [EDI + 45]



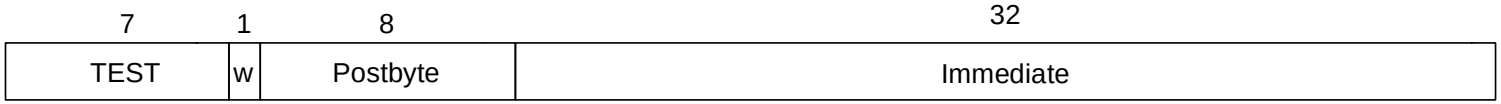
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



Example (8051)

```
007B:   0581    INC SP
007D:   18      DEC R0
007E:   EB      MOV A,R3
007F:   C6      XCH A,@R0
0080:   C9      XCH A,R1
0081:   08      INC R0
0082:   CA      XCH A,R2
0083:   C6      XCH A,@R0
0084:   CA      XCH A,R2
0085:   08      INC R0
0086:   F6      MOV @R0,A
0087:   7BFF    MOV R3,#FF
0089:   11A7    ACALL 00A7
008B:   D0E0    POP ACC
```

t	dt	PC	Source
-15	2	033A	MOV SCON,#52
-13	2	033D	MOV TMOD,#20
-11	2	0340	MOV TCON,#69
-9	2	0343	MOV TH1,#F3
-7	1	0346	MOV R3,#05
-6	1	0348	MOV R2,#03
-5	1	034A	MOV R1,#1F
-4	2	034C	LCALL ?PRINTF1
-2	2	0079	MOV R0,SP
0	0	007B	INC SP

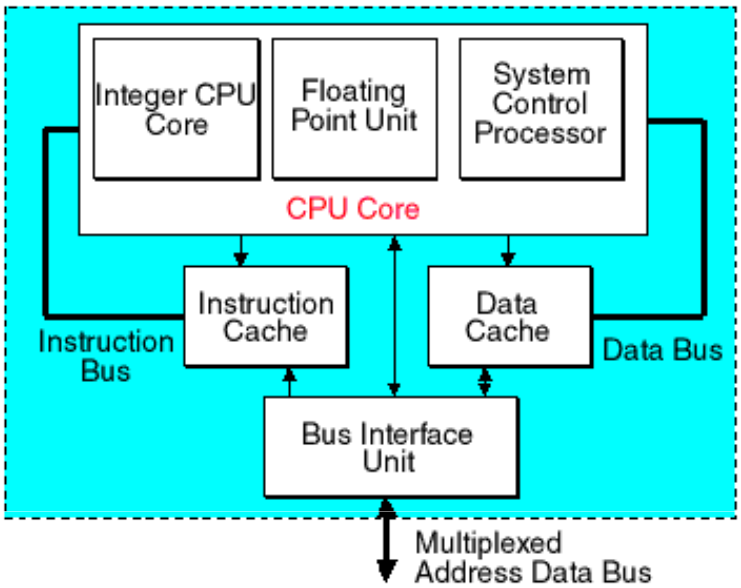
CISC的缺陷

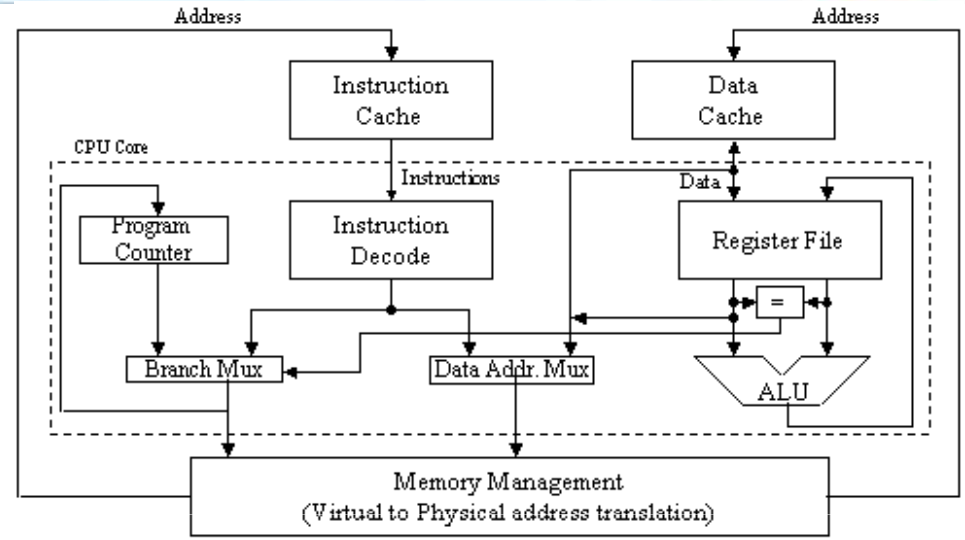
- 指令使用频度不均衡：
  - “80~20”理论
    - 80%的计算任务只需要调用处理器20%的指令就能完成
    - 扩充的复杂指令往往是低频度指令
- 大量复杂指令的控制逻辑不规整，不适于VLSI工艺
  - 微程序的使用反而制约了速度提高（微码的存控速度比CPU慢5-10倍）
- CISC指令的格式长短不一，需要不同的时钟周期来完成。
  - 执行较慢的指令影响系统的性能。
  - 不利于采用先进的指令级并行技术
- 软硬功能分配
  - 可划分的粒度变大了，灵活性小了

典型指令使用频度

指令类型	使用指令使用频度
数据传送类	43%
转/跳控制类	23%
算术运算类	15%
比较类	13%
逻辑运算类	5%
其他	1%

RISC基本设计思想





- 精简指令集：保留最基本指令(运算指令、加载、存储指令和转移指令)，减小CPI:
  CPUtime=IC * CPI * CC　IC—程序中指令数　CPI—每条指令执行周期数　CC—时钟周期
- 复杂指令可通过对简单基本的指令组合而成
- 每条指令的长度相同，大部分指令可以在一个机器周期里完成，CPI=1
- 采用多级指令流水线结构,处理器在同一时间内可执行多条指令，CPI<1
- 采用加载(Load)/存储(Store)结构，统一存储器访问方式，只允许Load和Store指令执行存储器操作，其余指令均对寄存器操作。
- 大大增加通用寄存器的数量，ALU只与寄存器文件直接连接。
- 采用高速缓存(cache)结构，并基于哈佛结构

MIPS指令格式

- 100余条指令（Hennessy中33条），共32个通用寄存器
- 指令格式：定长32位
  - R-type：arithmetic instruction
  - I-type：data transfer
  - J-type：branch instruction(conditional & unconditional)

R-type

op(6 bits)	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)

I-type

op(6 bits)	rs(5 bits)	rt(5 bits)	addr/immediate(16 bits)

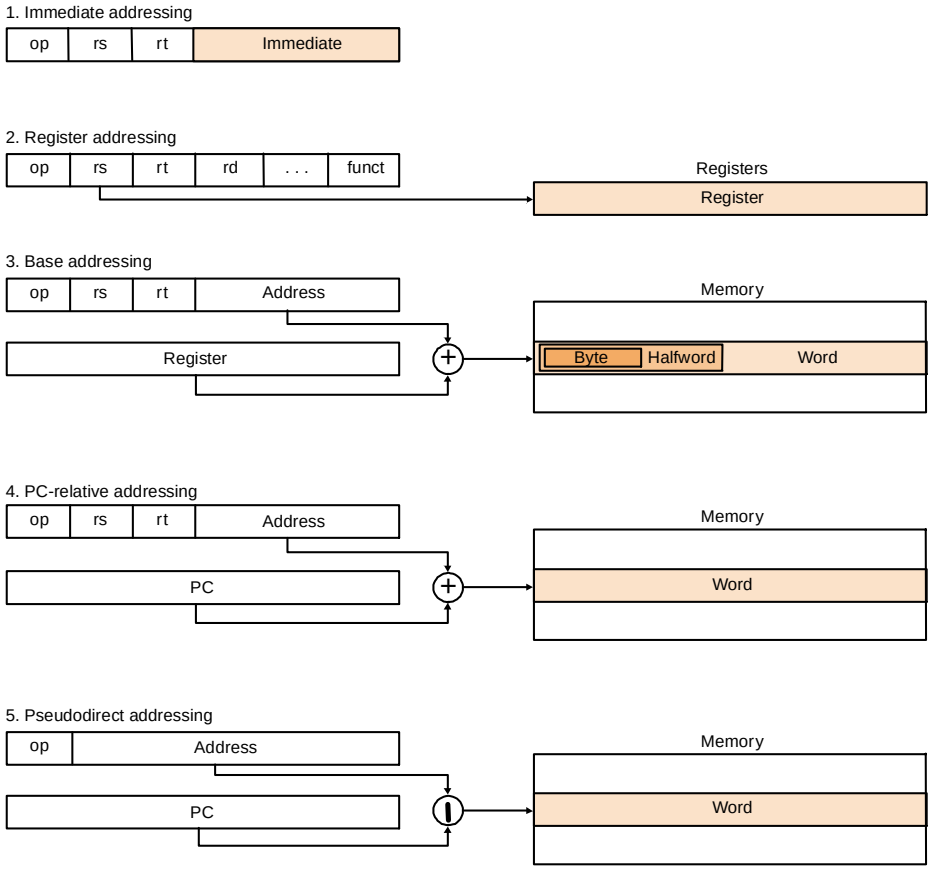
J-type

op(6 bits)	rs(5 bits)	rt(5 bits)	addr(16 bits)

op(6 bits)	addr(26 bits)

MIPS寻址模式

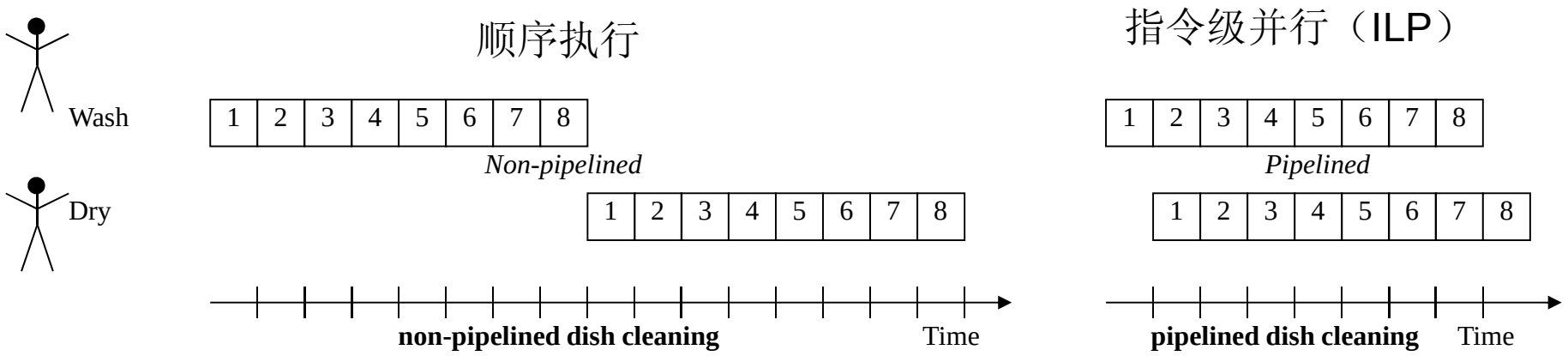
- 寄存器寻址：R-type
- 基址寻址：I-type
- 立即寻址
- 相对寻址
- 伪直接寻址（pseudodirect addressing）
  - 26位形式地址左移2位，与PC的高4位拼接



RISC CPU Example (ARM)

```
0x00008110      mov     r6,#1
0x00008114      add     r0,r6,r6,lsl #1
0x00008118      add     r6,r0,#1
0x0000811c      cmp     r6,r10
0x00008120      ble     0x8114  ; (shell_sort + 0x14)
0x00008124      mov     r1,r6
0x00008128      mov     r0,#3
0x0000812c      bl      __rt_sdiv
0x00008130      mov     r6,r0
0x00008134      add     r8,r6,#1
0x00008138      b       0x817c  ; (shell_sort + 0x7c)
0x0000813c      ldr     r9,[r5,r8,lsl #2]
0x00008140      mov     r4,r8
0x00008144      b       0x8154  ; (shell_sort + 0x54)
0x00008148      ldr     r0,[r5,r7,lsl #2]
```

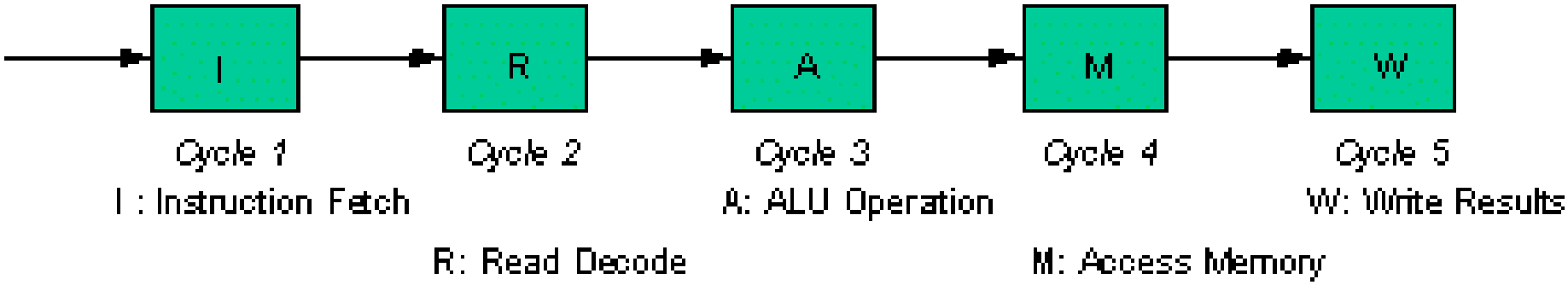
流水线技术



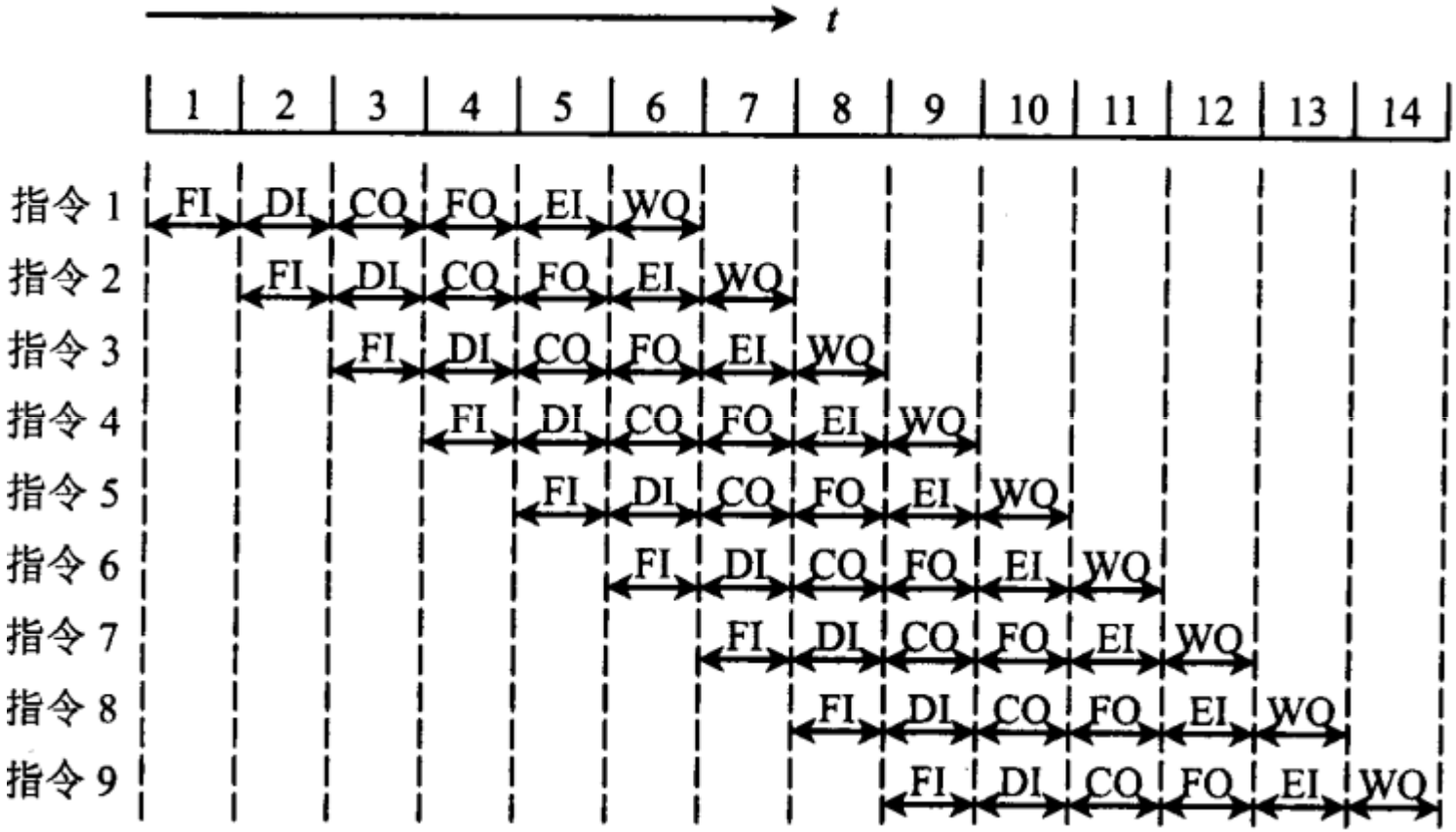
- 流水过程由多个相互联系的子过程组成，每个子过程称为流水线的“级”或“段”。
  - 一条流水线的段数被称为流水线的深度，或“流水深度”。
- 各个功能段所需时间尽量相等
  - 否则，消耗时间长的功能段将成为流水线的瓶颈，会造成流水线的阻塞或断流。
  - 每个功能段所需的时间一般为一个机器周期（cycle） 。

Pipelining of Instructions

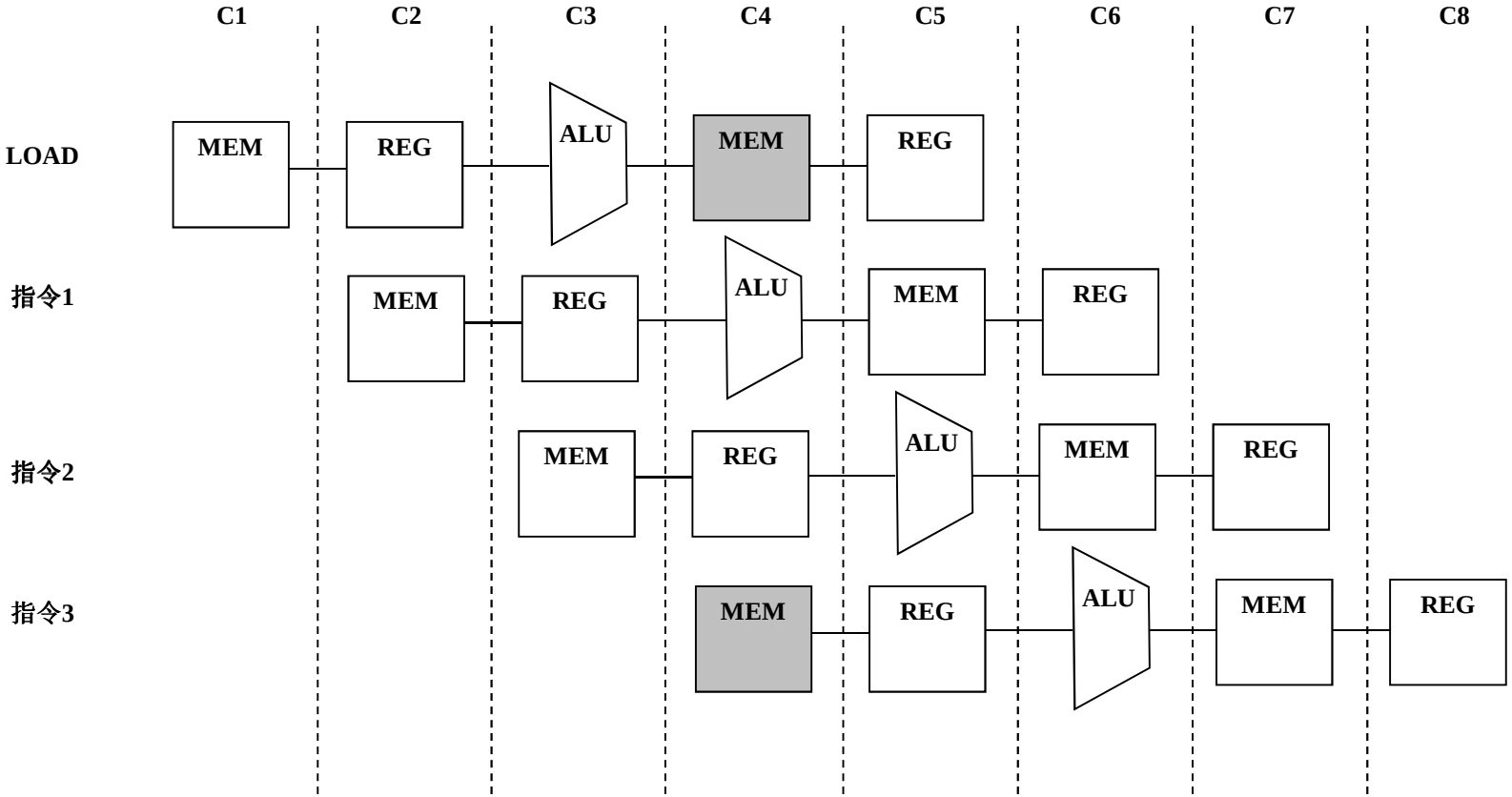
- RISC processors aim to execute one instruction per clock cycle。
- RISC processors break the complete fetched, decode and execution process into several stages。
- At any one time there will be an instruction at each stage, so for a five stage pipeline there will be five instructions in flight。



指令六级流水时序(时空图)

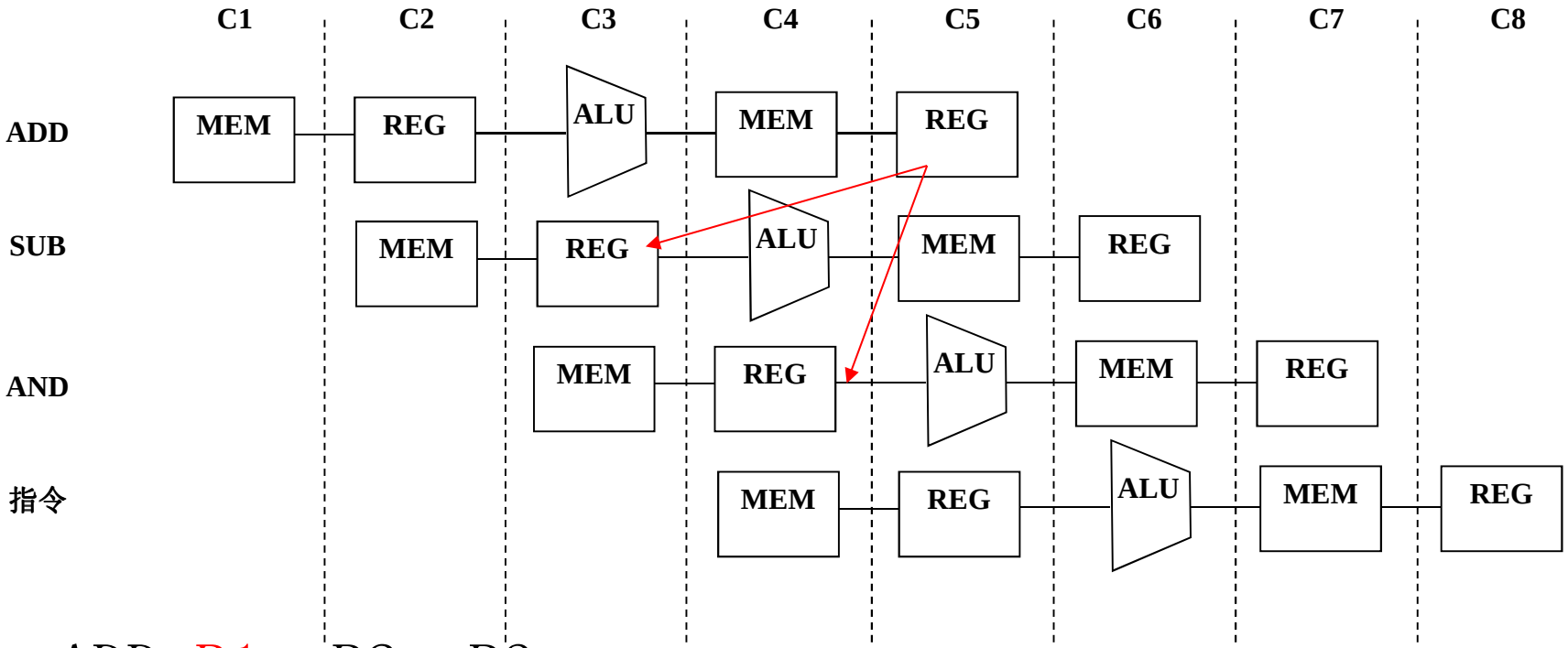


结构相关



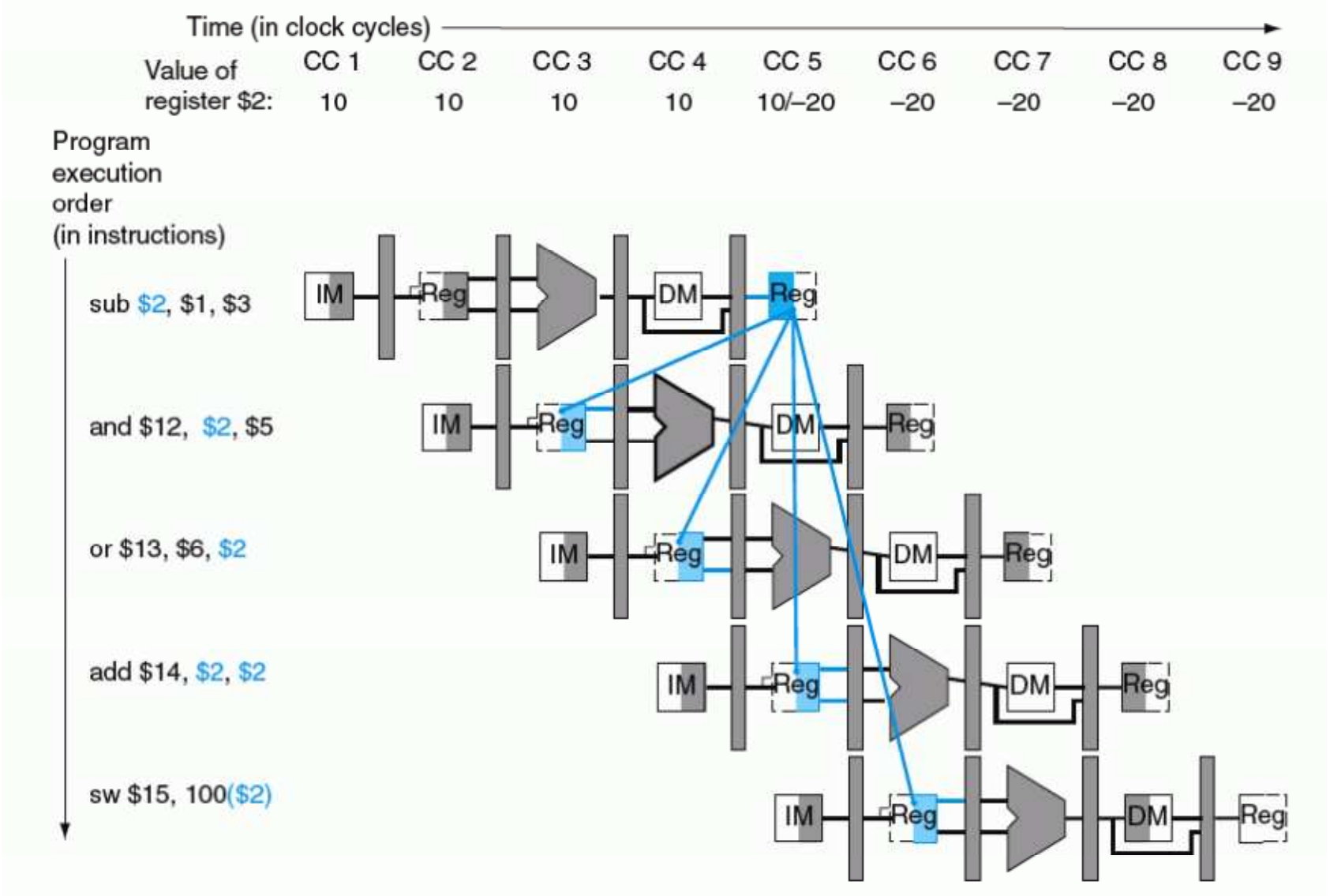
- 典型的结构相关由访存造成。

数据相关



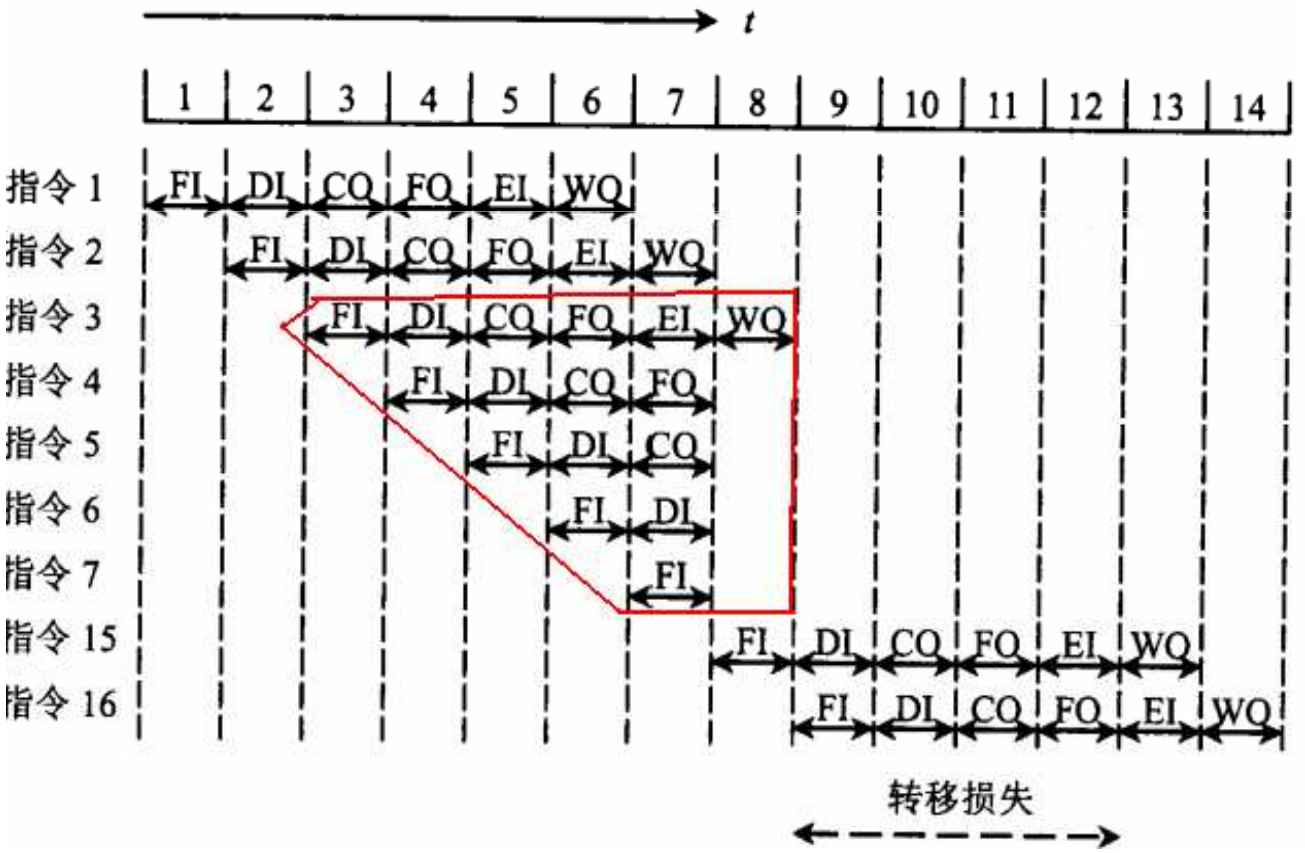
- ADD R1, R2, R3
- SUB R4, R1, R5
- AND R6, R1, R7

data hazard(RAW)

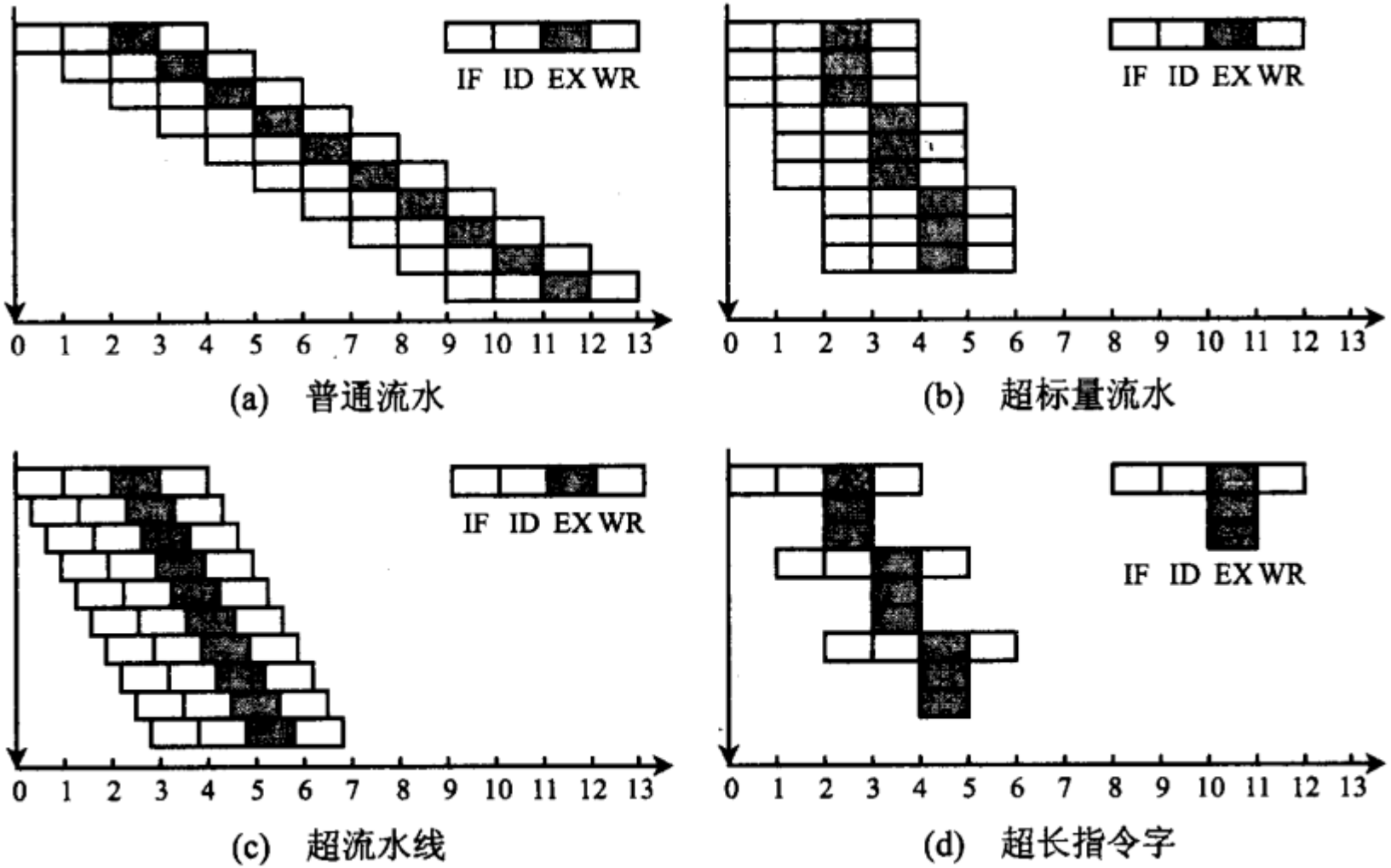


分支相关

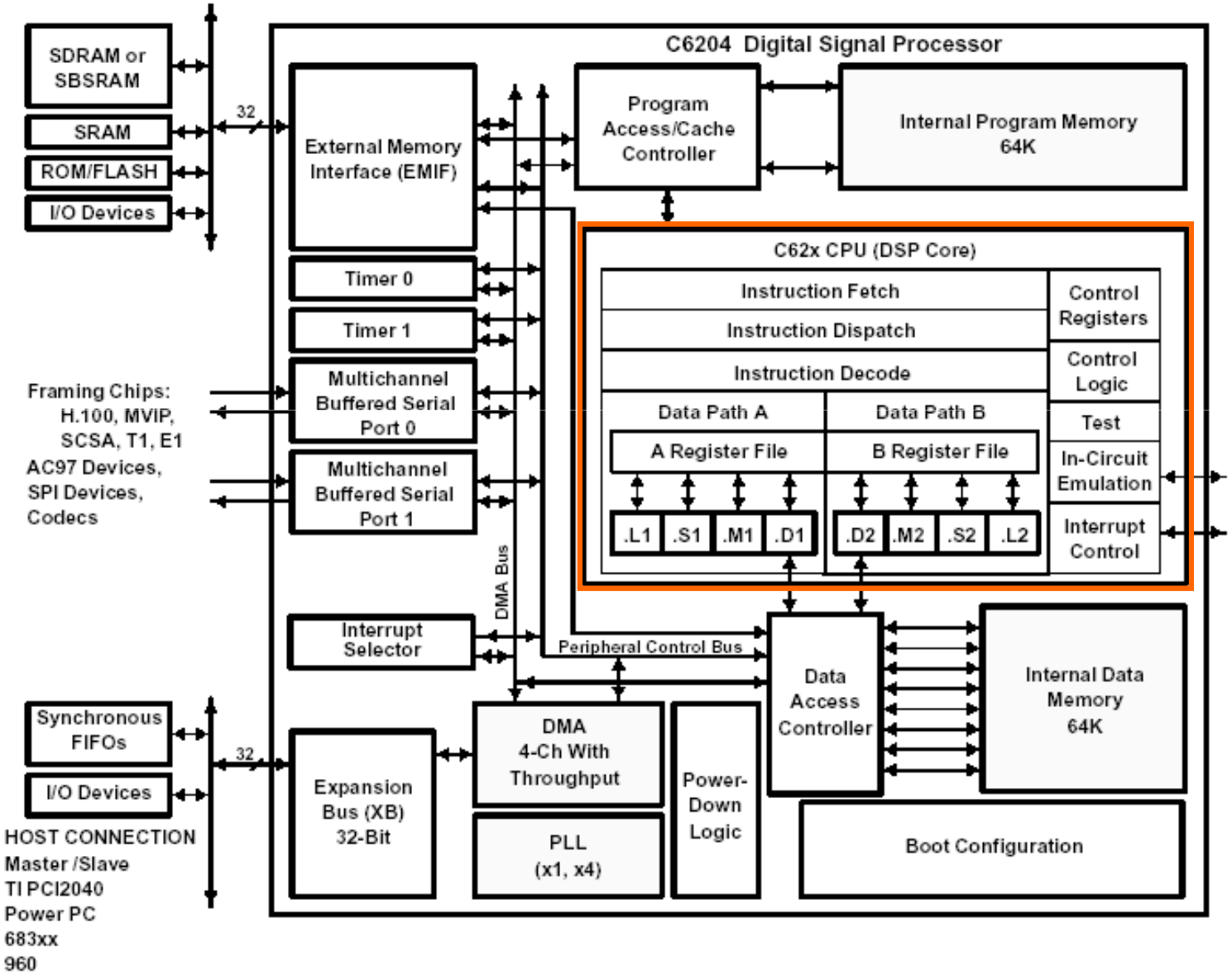
- 假设指令3是一条条件转移指令
  - 即指令3必须待指令2的结果出现后(第7个时间单元)，才能决定下一条指令是4(条件不满足)还是15(条件满足)。



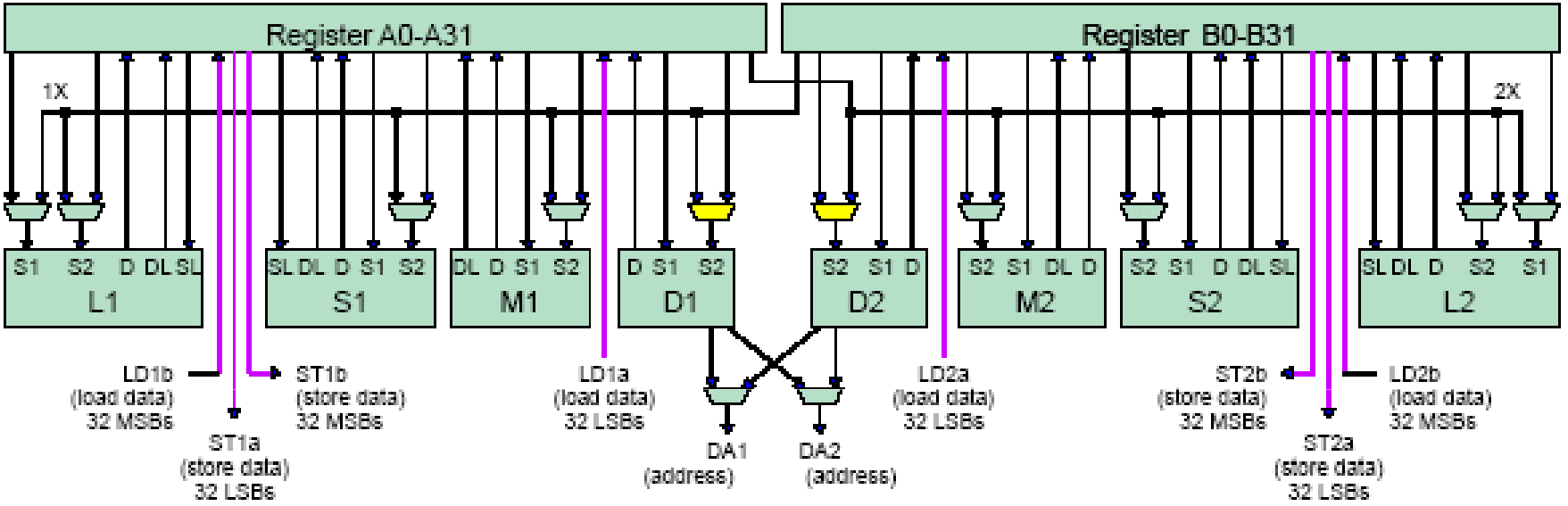
四种流水技术比较


IF ID EX WR
0 1 2 3 4 5 6 7 8 9 10 11 12 13
(a) 普通流水
IF ID EX WR
0 1 2 3 4 5 6 7 8 9 10 11 12 13
(b) 超标量流水
IF ID EX WR
0 1 2 3 4 5 6 7 8 9 10 11 12 13
(c) 超流水线
IF ID EX WR
0 1 2 3 4 5 6 7 8 9 10 11 12 13
(d) 超长指令字

TMS320C64x：VLIW

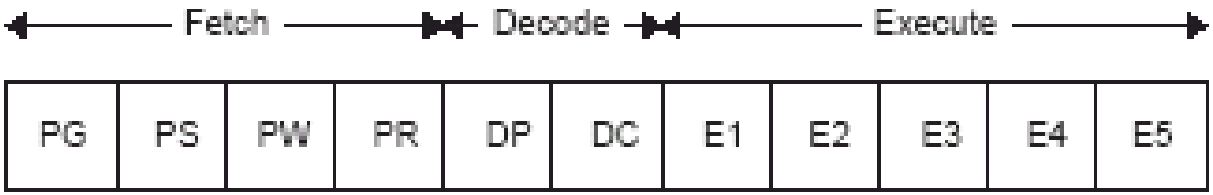


TMS320C64x：数据通路



- .M单元主要完成乘法运算，.D单元是唯一能产生地址的功能单元
- .L与.S是主要的算术逻辑功能单元（ALU）
- Two general-purpose register files (A and B) 64×32bit
- Eight functional units (.L1, .L2, .S1, .S2, .M1, .M2, .D1, and .D2)
- Two load-from-memory paths (LD1 and LD2) 64bit
- Two store-to-memory paths (ST1 and ST2) 64bit
- Two register file cross paths (1X and 2X)
- Two data address paths (DA1 and DA2)

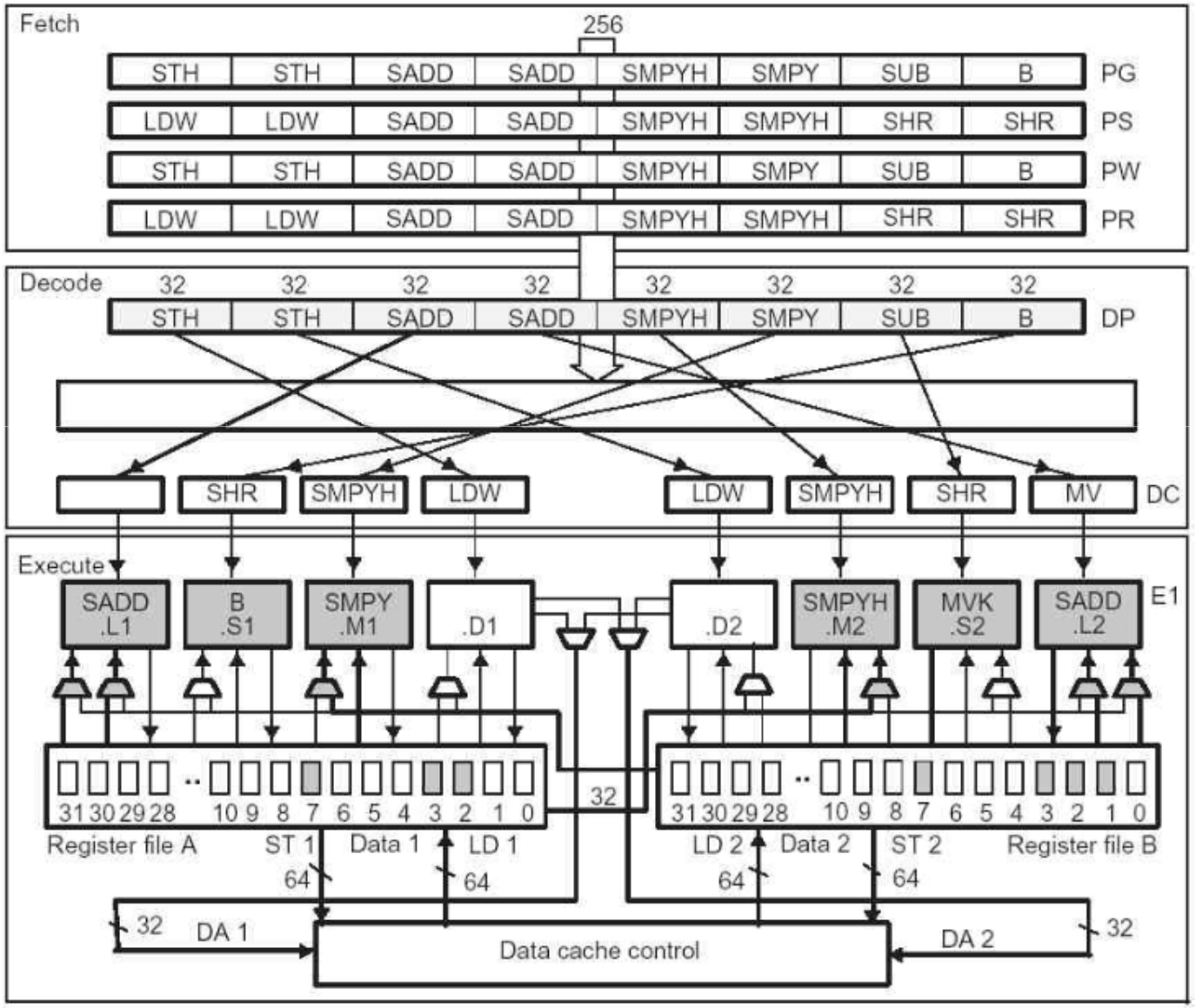
TMS320C64x：流水线



		Instruction Type					
		Single Cycle	16 X 16 Single Multiply/ C64x .M Unit Non-Multiply	Store	C64x Multiply Extensions	Load	Branch
Execution phases	E1	Compute result and write to register	Read operands and start computations	Compute address	Reads operands and start computations	Compute address	Target-code in PG‡
	E2		Compute result and write to register	Send address and data to memory		Send address to memory	
	E3			Access memory		Access memory	
	E4				Write results to register	Send data back to CPU	
	E5					Write data into register	
Delay slots		0	1	0†	3	4†	5‡

- PG: Program addr generate
- PS: Program address send
- PW: Program access ready wait
- PR: Program fetch packet receive
- DP: Instruction dispatch
- DC: Instruction decode

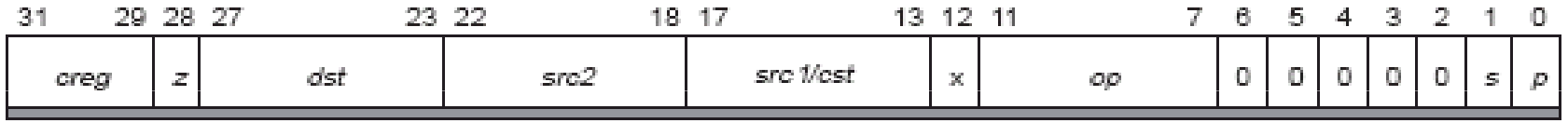
某时刻的流水线


Fetch
256
STH STH SADD SADD SMPYH SMPY SUB B PG
LDW LDW SADD SADD SMPYH SMPYH SHR SHR PS
STH STH SADD SADD SMPYH SMPY SUB B PW
LDW LDW SADD SADD SMPYH SMPYH SHR SHR PR
Decode
32 32 32 32 32 32 32 32
STH STH SADD SADD SMPYH SMPY SUB B DP
SHR SMPYH LDW LDW SMPYH SHR MV DC
Execute
SADD .L1
B .S1
SMPY .M1
.D1
.D2
SMPYH .M2
MVK .S2
SADD .L2
E1
31 30 29 28 10 9 8 7 6 5 4 3 2 1 0
32
Register file A
ST 1
Data 1
LD 1
LD 2
Data 2
ST 2
Register file B
64
64
64
64
32
DA 1
Data cache control
DA 2
32

TMS320C64x指令字

Operations on the .L unit

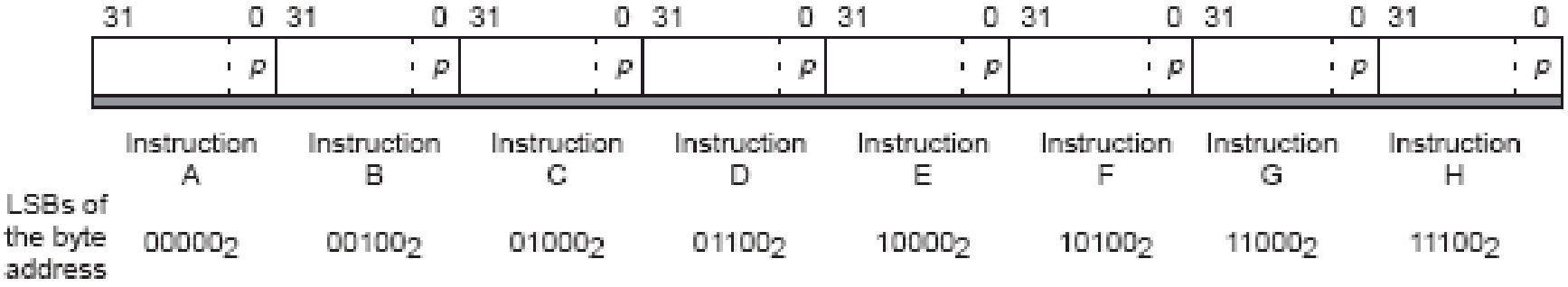
Operations on the .M unit

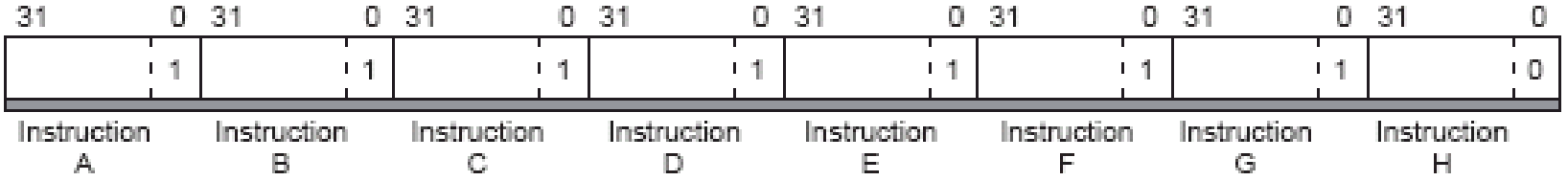


Operations on the .D unit

Operations on the .S unit

取指包 vs. 执行包

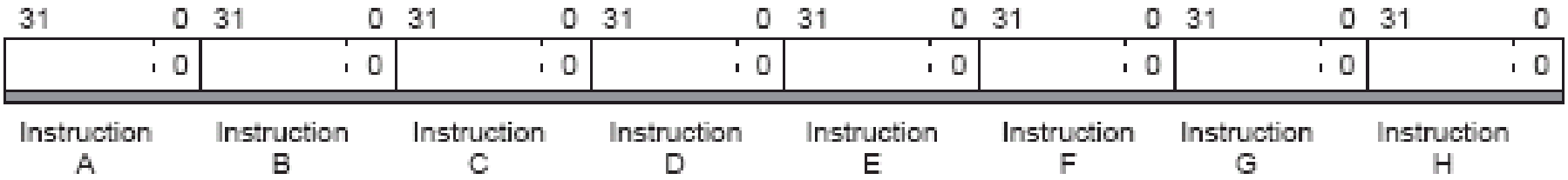




results in this execution sequence:

Cycle/Execute Packet	Instructions							
1	A	B	C	D	E	F	G	H

不可并行执行包



results in this execution sequence:

Cycle/Execute Packet	Instructions
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

ARM公司的发展历程

- ARM = Advanced RISC Machines
  - 1985年4月26日，第一个ARM原型在英国剑桥的Acorn计算机有限公司诞生，由美国加州San Jose VLSI技术公司制造。
  - 20世纪80年代后期，ARM很快开发成Acorn的台式机产品，形成英国的计算机教育基础。
- 1990年成立ARM公司
- 20世纪90年代，ARM32位嵌入式处理器占据了低功耗、低成本和高性能的嵌入式系统应用领域的领先地位。
  - 1999年因移动电话火爆市场，其32位RISC处理器占市场份额超过了50%
  - 2001年初，ARM公司的32位RISC处理器市场占有率超过了75%。
- ARM公司是IP供应商，是设计公司。
  - 由合作伙伴公司来生产各具特色的芯片。
  - Also develop technologies to assist with the design-in of the ARM architecture
    - Software tools, boards, debug hardware, application software, bus architectures, peripherals etc

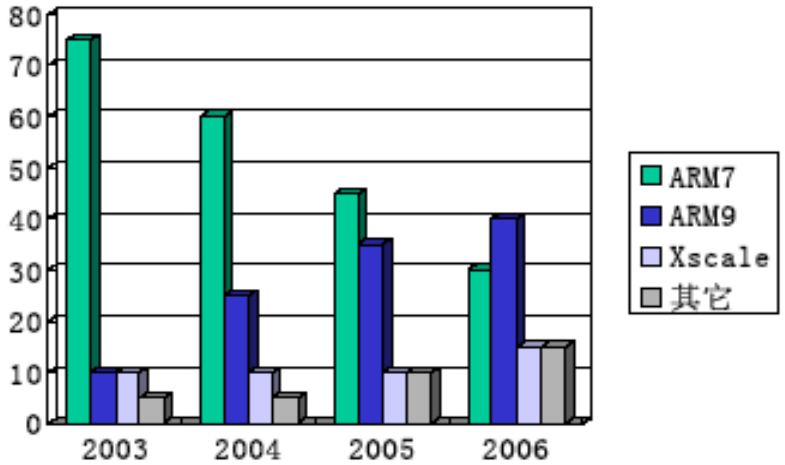
ARM Partnership Model

ARM Powered Products

ARM产品

- 1991，ARM推出第一款RISC嵌入式微处理器核 ARM6
- 1993，ARM推出ARM7核。
  - 从ARM7开始，ARM核被普遍认可和广泛使用
- 1995，ARM的Thumb扩展指令集结构为16位系统增加了32位的性能,提供业界领先的代码密度
- 1995年，StrongARM问世

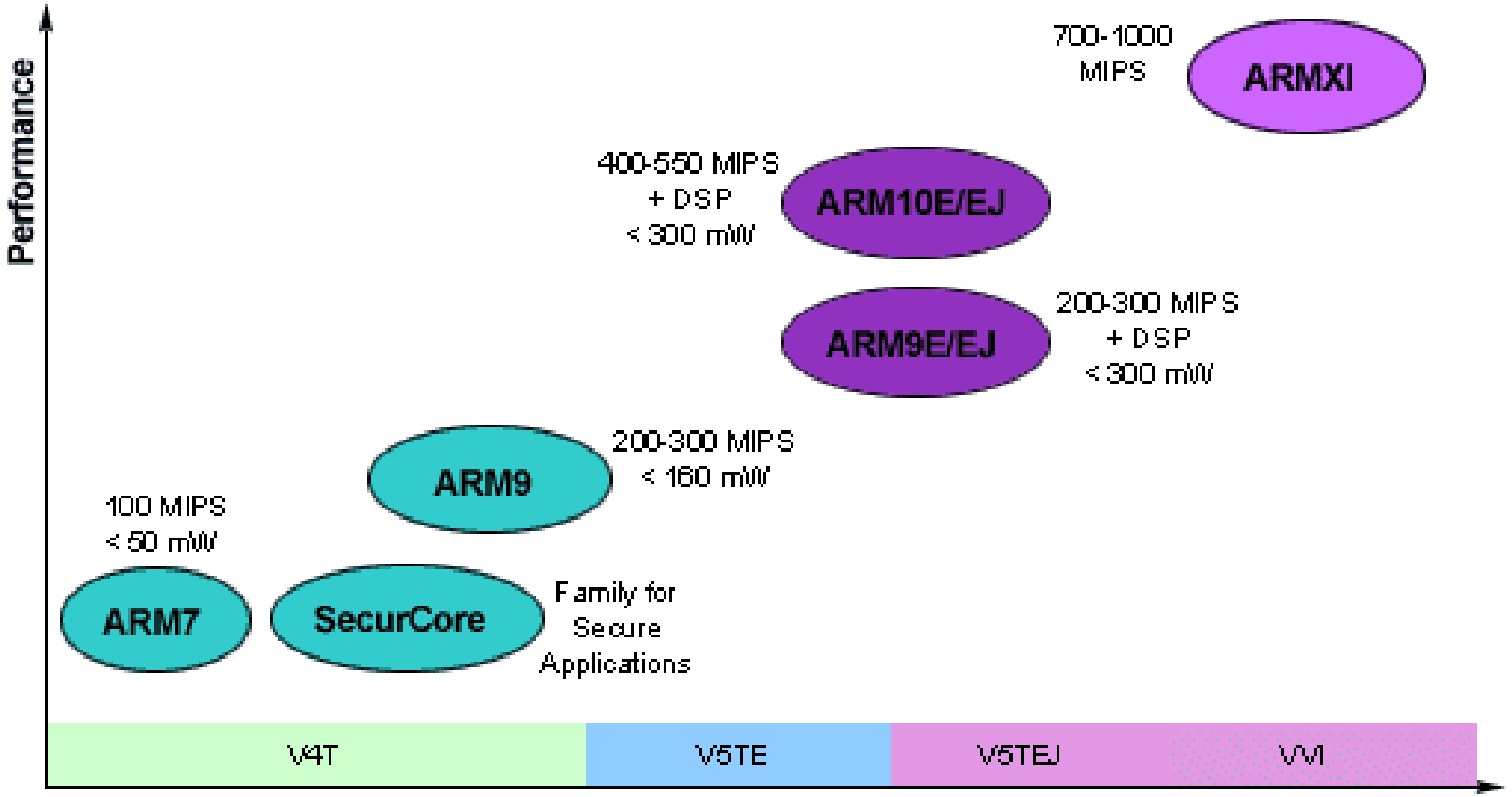
XScale是下一代StrongARM芯片的发展基础
ARM10TDMI是ARM处理器核中的高端产品
ARM11是ARM家族中性能最强的一个系列



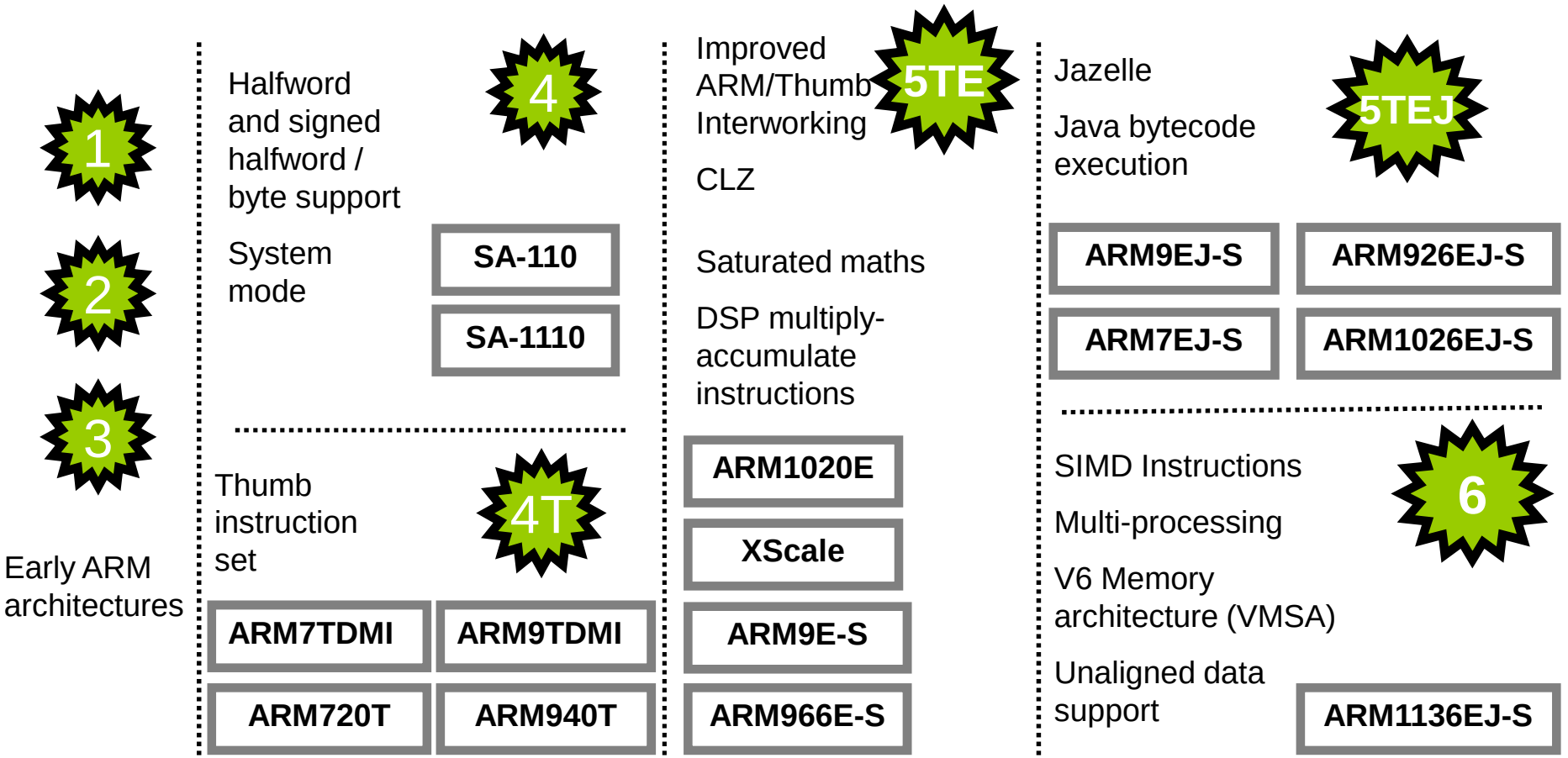
ARM核

	Cache Size (Inst/Data)	Tightly Coupled Memory	Memory Manage -ment	AHB Bus Interface	Thumb	DSP	Jazelle	Clock MHz **
Embedded Cores								
ARM7TDMI	No	No	No	Yes*	Yes	No	No	133
ARM7TDMI-S	No	No	No	Yes*	Yes	No	No	100-133
ARM7EJ-S	No	No	No	Yes*	Yes	Yes	Yes	100-133
ARM966E-S	No	Yes	No	Yes	Yes	Yes	No	230-250
ARM940T	4K/4K	No	MPU	Yes*	Yes	No	No	180
ARM946E-S	Variable	Yes	MPU	Yes	Yes	Yes	No	180-210
ARM1026EJ-S	Variable	Yes	MMU+MPU	dual AHB	Yes	Yes	Yes	266-325
Platform Cores								
ARM720T	8K unified	No	MMU	Yes	Yes	No	No	100
ARM920T	16K/16K	No	MMU	Yes*	Yes	No	No	250
ARM922T	8K/8K	No	MMU	Yes*	Yes	No	No	250
ARM926EJ-S	Variable	Yes	MMU	dual AHB	Yes	Yes	Yes	220-250
ARM1020E	32K/32K	No	MMU	dual AHB	Yes	Yes	No	325
ARM1022E	16K/16K	No	MMU	dual AHB	Yes	Yes	No	325
ARM1026EJ-S	Variable	Yes	MMU+MPU	dual AHB	Yes	Yes	Yes	266-325
Secure Applications								
SC100	No	No	MPU	No	Yes	No	No	80
SC110	No	No	MPU	No	Yes	No	No	80
SC200	Optional	No	MPU	No	Yes	Yes	Yes	110
SC210	Optional	No	MPU	No	Yes	Yes	Yes	110
Intel ARM-based Processors								
StrongARM	16K/8K	No	MMU	N/A	No	No	No	206
Intel XScale	32K/32K	No	MMU	N/A	Yes	Yes	No	400

ARM微处理器性能一览



Development of the ARM Architecture



V1和V2版架构

- V1版架构只在原型机ARM1出现过,其基本性能：
  - 基本的数据处理指令(无乘法)
  - 字节、半字和字的LOAD/STORE指令
  - 转移指令,包括子程序调用及链接指令
  - 软件中断指令
  - 寻址空间：64M字节(26位寻址空间)
- V2版对V1版进行了扩展,如ARM2与ARM3(V2a版)架构,增加了以下功能：
  - 乘法和乘加指令
  - 支持协处理器操作指令
  - 快速中断模式
  - SWP/SWPB的最基本存储器与寄存器交换指令

V3版架构

- 把寻址空间增至32位(4G字节)
- 增加了当前程序状态寄存器CPSR(Current Program Status Register)和程序状态保存寄存器SPSR(Saved Program Status Register)以便于异常(Exception)的处理。
- 增加了中止(Abort)和未定义二种处理器模式。
- 指令集变化如下：
  - 增加了MRS/MSR指令,以访问新增的CPSR/SPSR寄存器
  - 增加了从异常处理返回的指令功能
  - T —— Thumb状态：16位指令执行时可以无开销地扩展成32位指令格式
  - M —— 长乘法支持:（32*32=>64或者32*32+64=>64）
- ARM6就采用该版架构。

V4版架构

- V4版架构是目前应用最广的ARM体系结构。
- 对V3版架构进行了进一步扩充,有的还引进了16位的Thumb指令集,使ARM使用更加灵活。
- D —— 对调试的支持（Debug）,JTAG接口，片上调试
- I —— 嵌入的ICE（In Circuit Emulation）,允许对电路内部嵌入的ARM内核进行访问和控制，片上断点
- 指令集中增加了以下功能：
  - 符号化和非符号化半字及符号化字节的存/取指令
  - 增加了16位Thumb指令集
  - 完善了软件中断SWI指令的功能
  - 处理器系统模式引进特权方式时使用用户寄存器操作
  - 把一些未使用的指令空间捕获为未定义指令
- ARM7、部分ARM9和StrongARM都采用该版架构。

V5版架构

- 在V4版基础上增加了一些新的指令，主要提升了ARM和Thumb指令的交互工作能力。
- E —— DSP指令支持。
- J —— Java指令支持。
- 新增指令有：
  - 带有链接和交换的转移BLX指令
  - 计数前导零CLZ指令
  - BRK中断指令，软件断点
  - 增加了信号处理指令(V5TE版)
  - 为协处理器增加更多可选择的指令
- ARM10和XScale都采用该版架构。

V6版架构

- 在V5版基本上增加了增加了媒体指令(SIMD, FFT, MPEG4)，可以使音频/视频处理性能提高4倍。
- 提升了性能与功耗比，适合使用电池供电的便携式设备。
- 多处理器支持(multiprocessing)
- 改善的异常和中断处理能力
- 新的内存管理机制
- 混序数据支持
- ARM11采用该版架构。

ARM产品与体系结构

	Thumb	DSP	Jazelle	Media	TrustZone	Thumb-2	
v4							StrongARM
v4T	*						ARM7T, ARM9
v5T	*						ARM10T，XScale
v5TE	*	*					ARM9E, ARM10E
v5TEJ	*	*	*				ARM7EJ,ARM9EJ, ARM10EJ
v6	*	*	*	*			ARM1136J(F)-S
v6Z	*	*	*	*	*		
v6T2	*	*	*	*		*	ARM1156T2(F)-S

ARM处理器的特点

- ARM处理器本身是32位设计，但也配备16位指令集。
  - 16位指令占用的存储器空间节省35%。
- 条件执行
  - 所有的指令都可根据前面的执行结果决定是否被执行，提高了指令的执行效率。
- 可用Load/Store指令批量传输数据。
- 可在一条数据处理指令中同时完成逻辑处理和移位。
- 支持基于ARM处理器核的高集成SOC调试

ARM7微处理器系列

- 三级流水线结构。最高可达130MIPS。
  - 取指、译码、执行
- 对操作系统的支持广泛
  - 包括Windows CE、Linux、Palm OS等。
- 指令系统与ARM9系列、ARM10E系列兼容
- ARM7TMDI是目前使用最广泛的32位嵌入式RISC处理器，属低端ARM处理器核。
  - Samsung公司的S3C44B0

ARM9微处理器系列

- 5级流水线，哈佛结构，最高可达300MIPS。
  - 取指->指令译码->执行->数据缓存->写回
- 支持32位的高速AMBA总线接口。
- 全性能的MMU，支持Windows CE、Linux、Palm OS等多种主流嵌入式操作系统。
- 支持数据Cache和指令Cache，具有更高的指令和数据处理能力。
- ARM9系列主要应用于无线设备、仪器仪表、安全系统、机顶盒、高端打印机、数字照相机和数字摄像机等。

ARM流水线

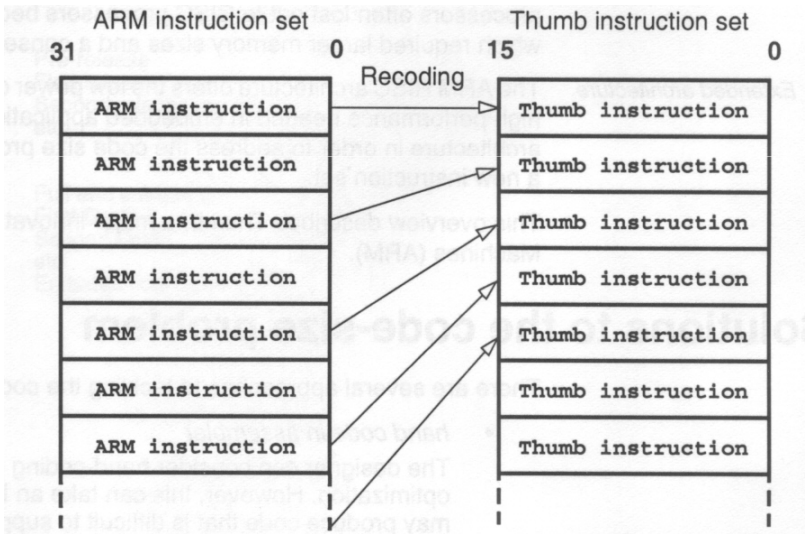
ARM7	预取（Fetch）	译码（Decode）	执行（Execute）					
ARM9	预取（Fetch）	译码（Decode）	执行（Execute）	访存（Memory）	写回（Write）			
ARM10	预取（Fetch）	发射（Issue）	译码（Decode）	执行（Execute）	访存（Memory）	写回（Write）		
ARM11	预取（Fetch）	预取（Fetch）	发射（Issue）	译码（Decode）	转换（Snny）	执行（Execute）	访存（Memory）	写回（Write）

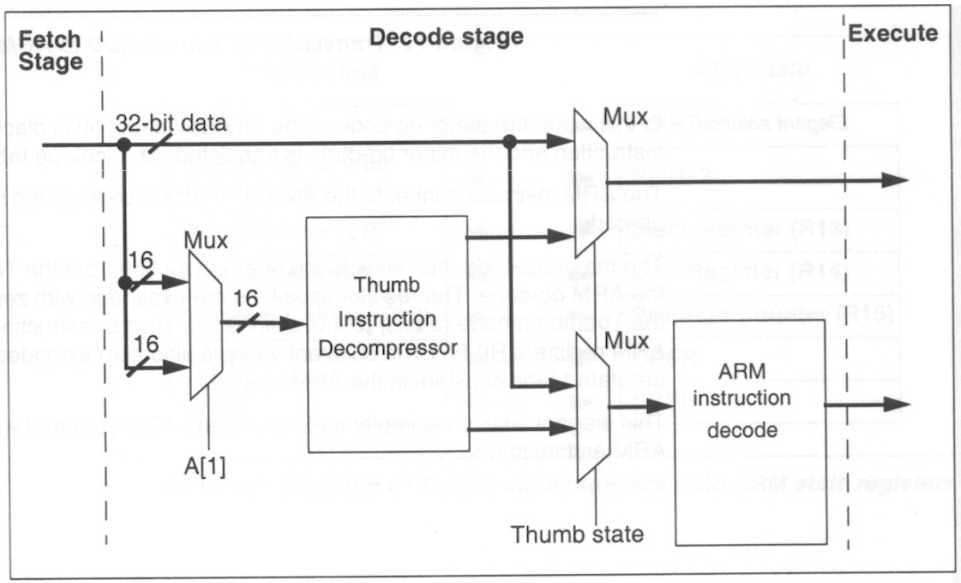
ARM11流水线分支预测

- 分支预测
  - 动态预测
    - 64个4状态跳转地址缓存器BTAC（4-state branch target address cache）保存最近发生的跳转地址。
      - StronglyTaken，WeaklyTaken，Strongly not Taken，Weakly not Taken
  - 静态预测
    - 如果BTAC中没有记录，从跳转的方式判断是否执行。
      - 如果是向回跳转，大多数情况是遇到一个循环，则假设这条指令被执行。
      - 如果是向前跳转，则假设这条指令不被执行。
  - 动态预测和静态预测的组合使ARM11能达到85%的预测正确性
    - 每一个正确的预测，减少5个时钟周期的等待时间。
- 单发射，多执行部件（ALU、MAC、LS）
  - 消除指令间等待，类VLIW体系
- 乱序执行

Thumb体系结构

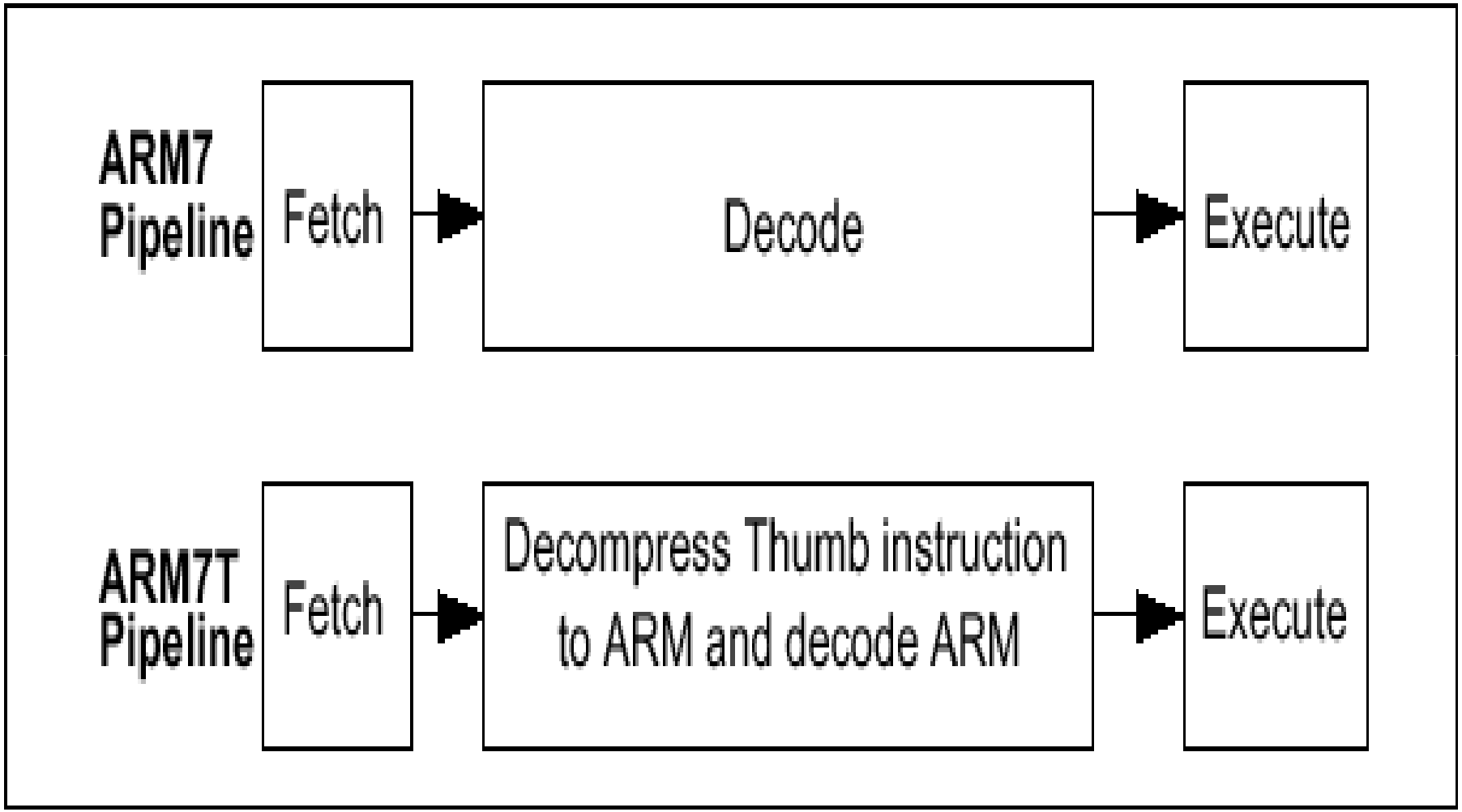
- Thumb指令集（‘T’)
  - ◆32位ARM指令集的子集，按16位指令重新编码





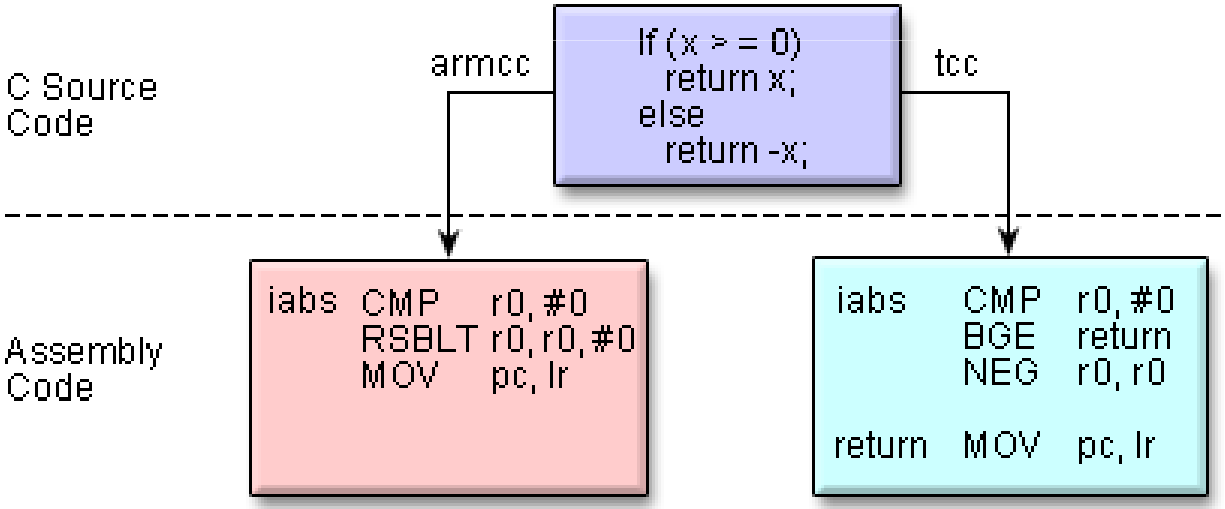
- ◆代码尺寸小（up to 40 % compression)
- ◆简化设计

Thumb的技术实现


ARM7
Pipeline
Fetch
Decode
Execute
ARM7T
Pipeline
Fetch
Decompress Thumb instruction
to ARM and decode ARM
Execute

指令集选择

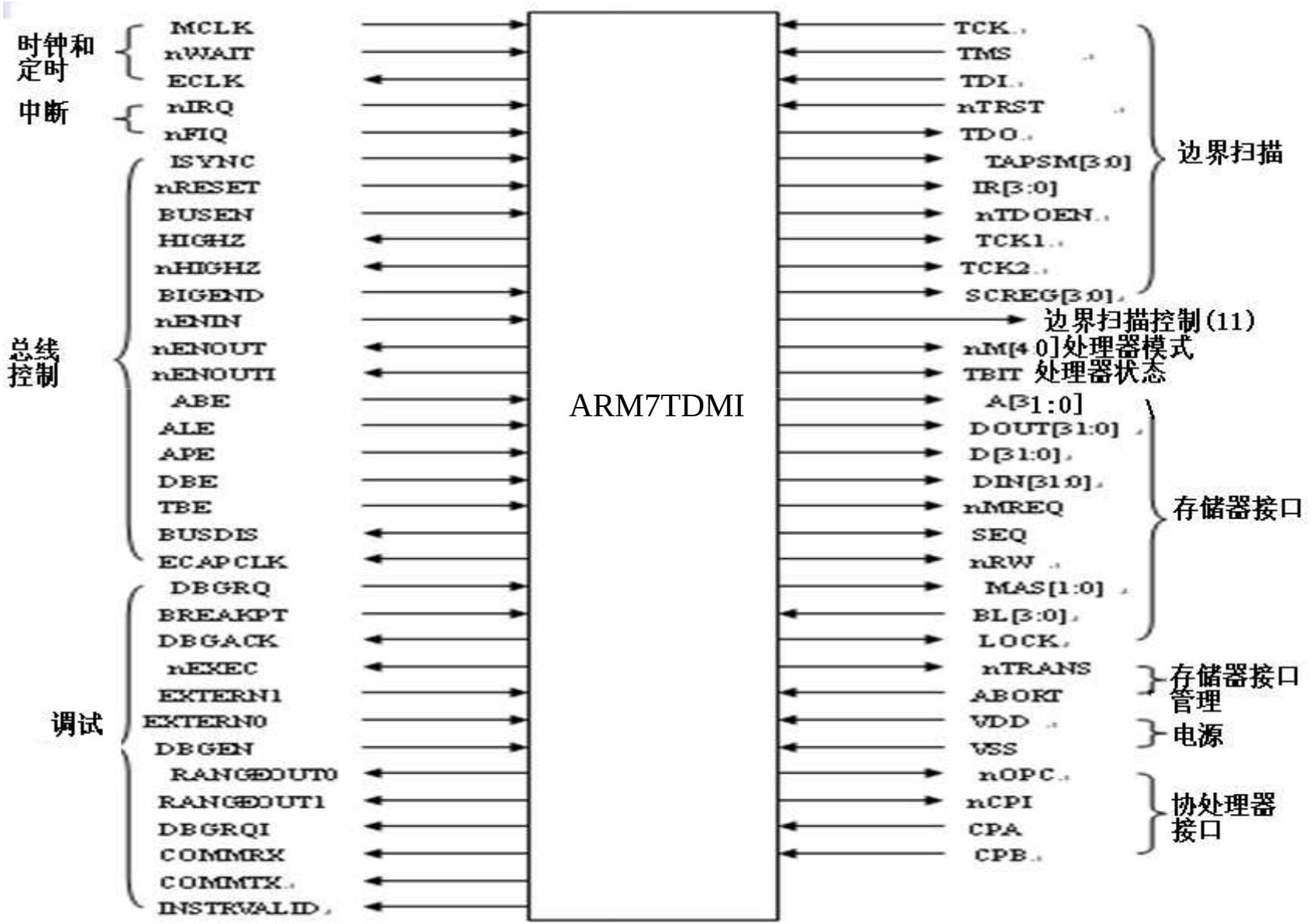
- 若对系统的性能有较高要求，应使用**32**位的存储系统和**ARM**指令
- 若对系统的成本及功耗有较高要求，则应使用**16**位的存储系统和**Thumb**指令集。
- 若两者结合使用，充分发挥其各自的优点，会取得更好的效果。



Comparison

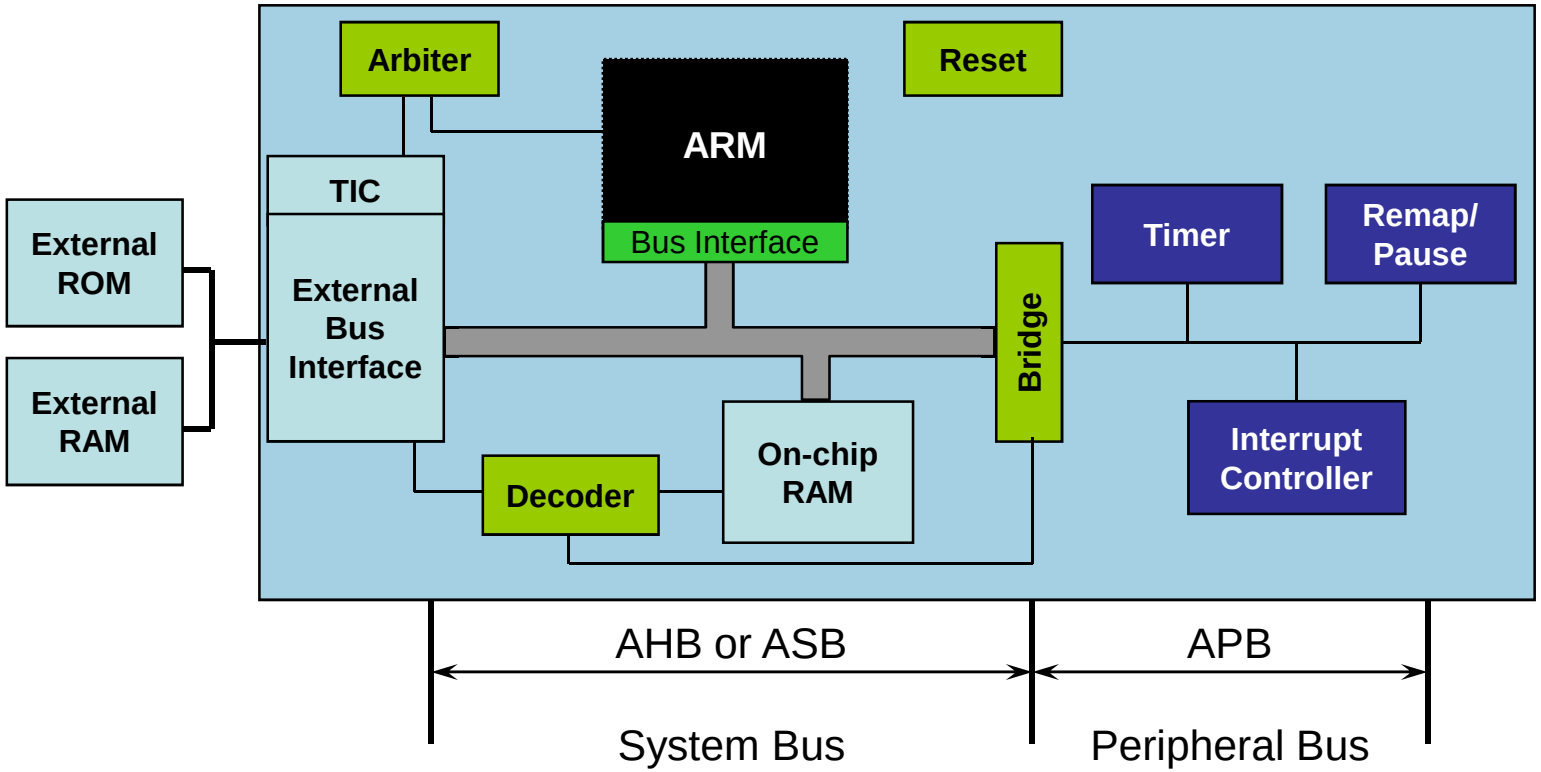
Code	Instruction	Size (Bytes)	Normalized
ARM	3	12	1.0
Thumb	4	8	0.67

ARM7TDMI功能信号图



ARM片上总线AMBA

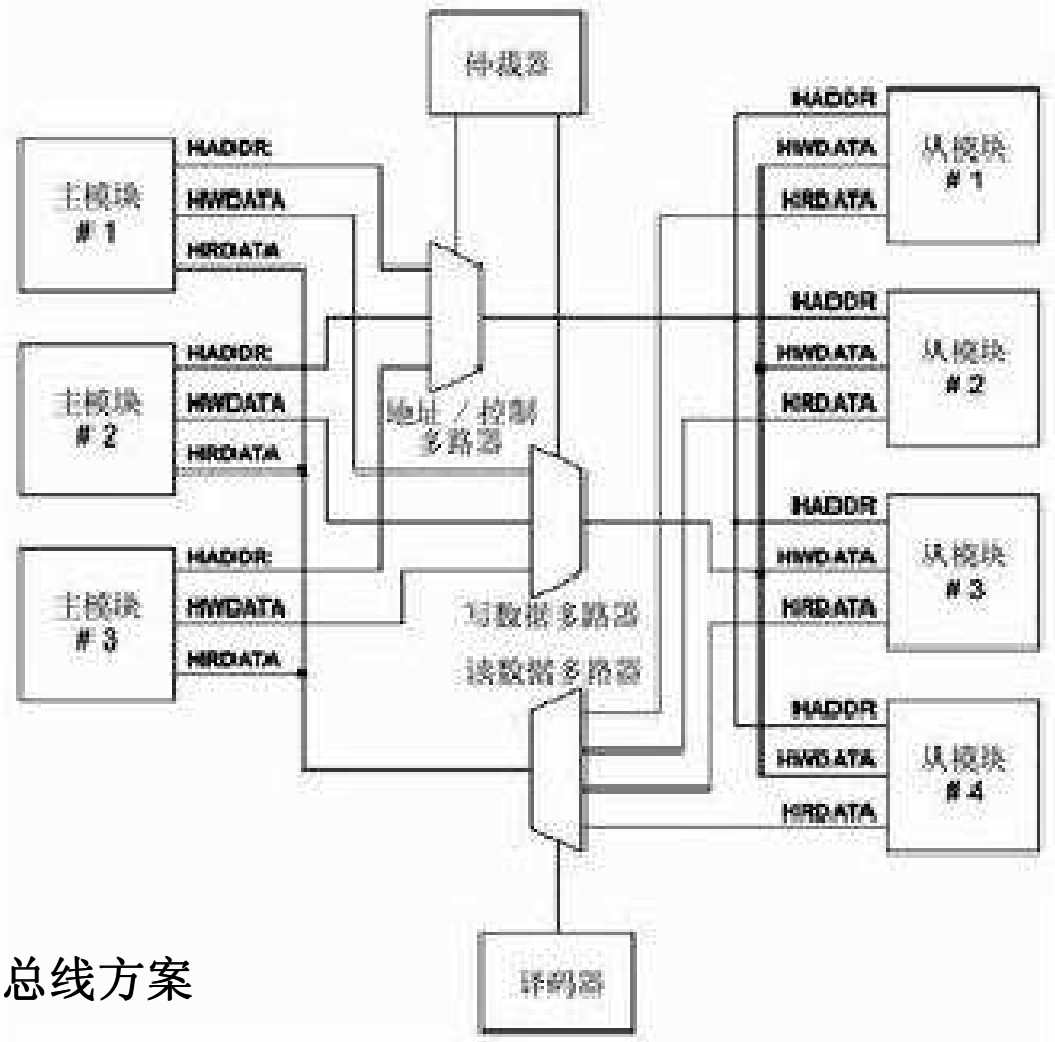
- On-Chip Bus：用于支持IP集成
  - IBM：CoreConnect；Silicore Corp：Wishbone
  - ARM：AMBA(Advanced Microcontroller Bus Arch)



AHB主要特性

- 多控制器；分段传输；
  - 单周期总线控制权移交；
- **32～128**位总线宽；
  - 访问空间限制在**32**位；
  - 支持字节、半字和字传输，突发传输模式最大**16**字节；
- 支持仲裁、**REQ**、**GNT**和**LOCK**；
- 访问保护机制
  - 区别特权访问和无特权访问模式，或指令和数据提取等；
- 提供为较慢设备使用而扼制数据流的机制；
- **AHB**总线支持分离传输模式。

AHB(Advanced High performance Bus)



中心多路器总线方案

图1 AHB总线互连结构图

APB主要特征

- 低性能、低功率外围总线
- **32**位地址空间；
- **32**位数据总线；
- 单控制器（**APB**桥）
- 简单，只有**4**个控制信号
  - PSELx，PENABLE，PADDR、PWRITE
- 两个时钟周期传输
- 无需等待周期和回应信号
- 分开读和写数据总线

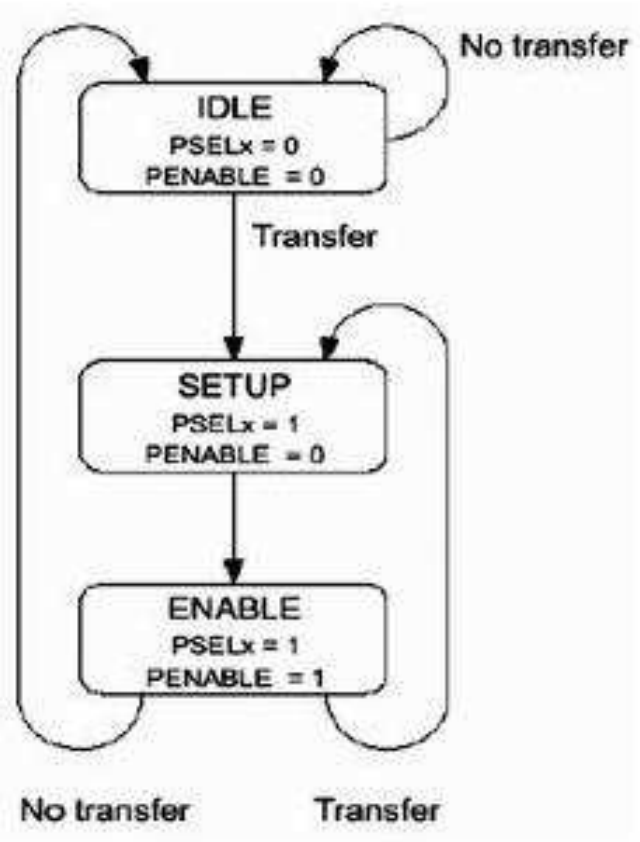
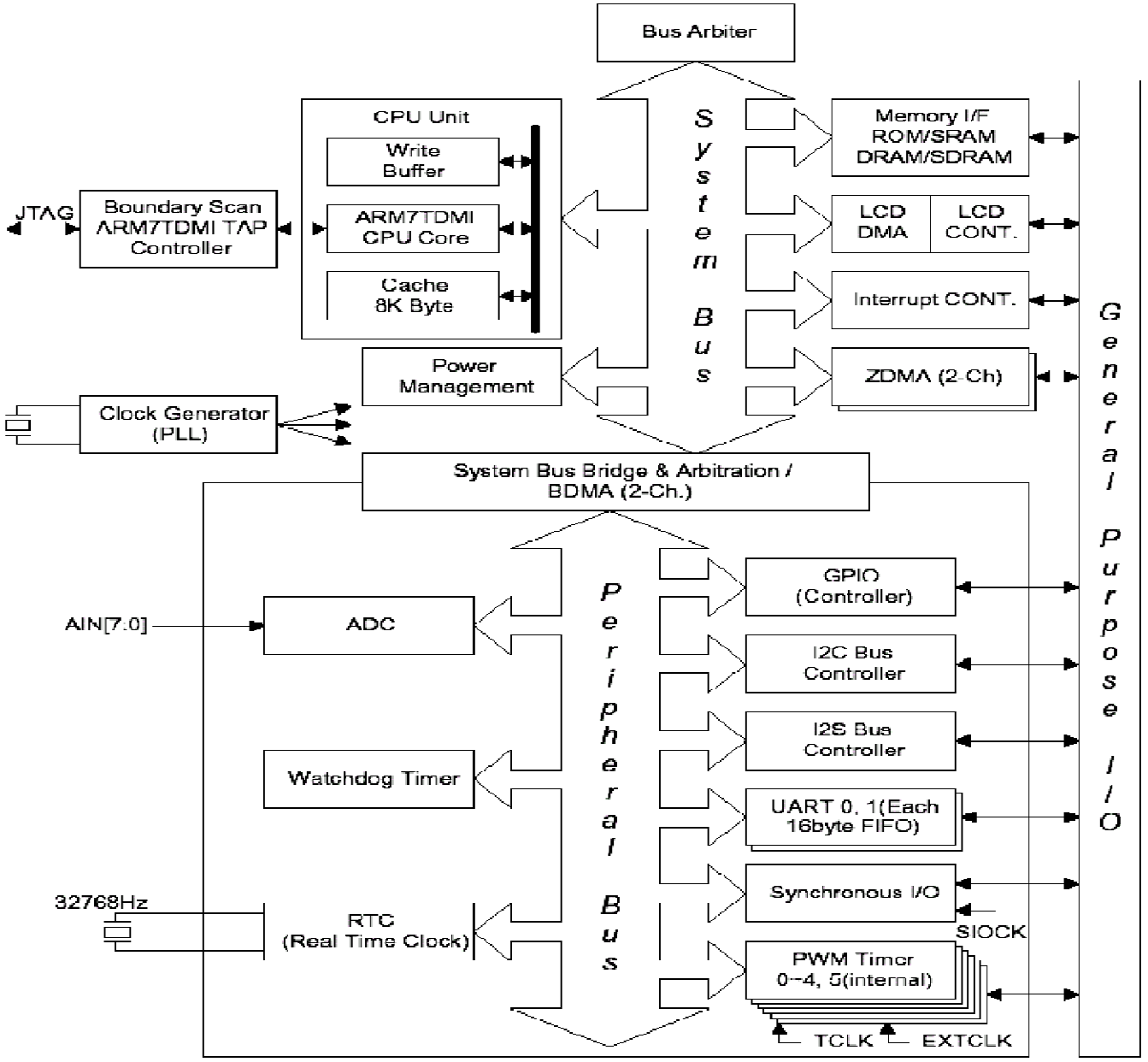


图2 APB传输状态图

S3C44B0X微处理器体系结构框图

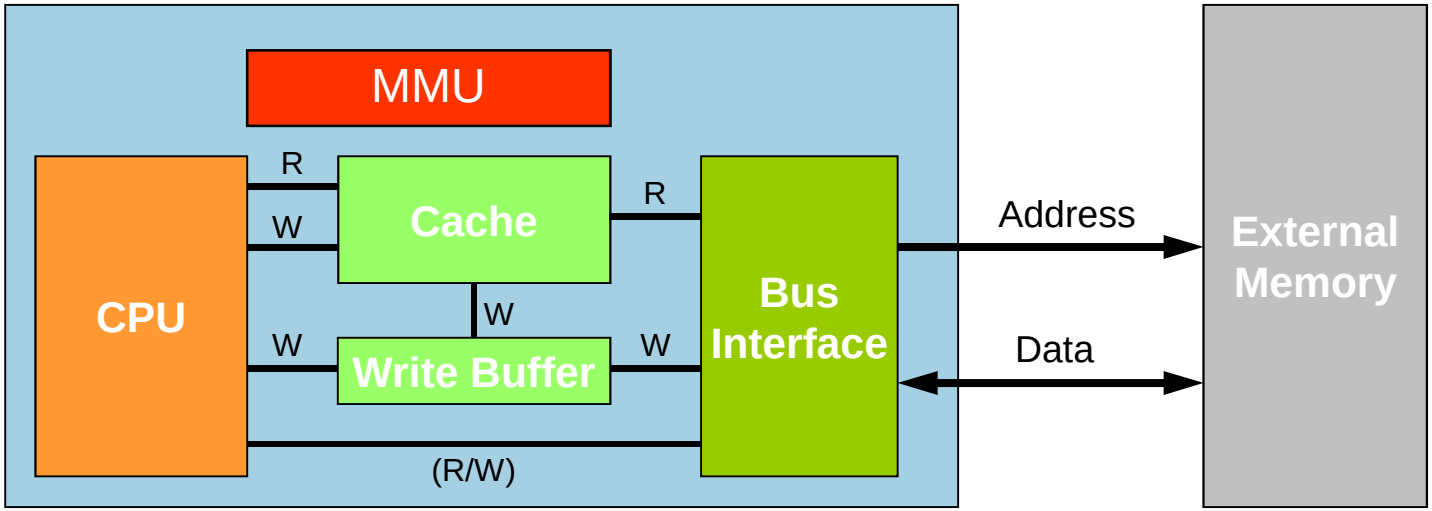


存储器

- SRAM：一bit大约需要六个晶体管，读写速度较快。
- DRAM：一bit大约需要一个晶体管和一个电容，需要刷新。
- SDRAM：与CPU频率同步，共享一个时钟周期。
- FLASH存储器：既具有SRAM读写的较快的访问速度，又具有ROM在断电后信息不丢失的特点。由于FLASH不需要存储电容器，故其集成度更高，制造成本低于DRAM。
  - NOR技术Flash
    - 拥有独立的数据总线和地址总线,能快速随机读取
    - 重新编程之前需要对块或整片进行预编程和擦除操作
      - 擦除和编程时间长，不适于纯数据存储和文件存储的应用
  - NAND技术Flash
    - 具有快编程和快擦除的功能，其块擦除时间比NOR技术快一百多倍
    - 数据、地址采用同一总线。随机读取速度慢且不能按字节随机编程。
    - 芯片尺寸小，成本(bit cost)低，可取代硬盘或其他大容量存储设备

高效的存储器系统

- MMU支持虚拟/物理地址映射，满足操作系统内存管理需要
- Cache和Write Buffer提高系统存储器访问性能
- MMU中的地址转换页表和Cache的内容需要按需更新
  - 带有一定的随机性，造成某些实时不确定性
  - ARM的MMU和Cache设计允许部分随机内存锁定，以提高实时性能



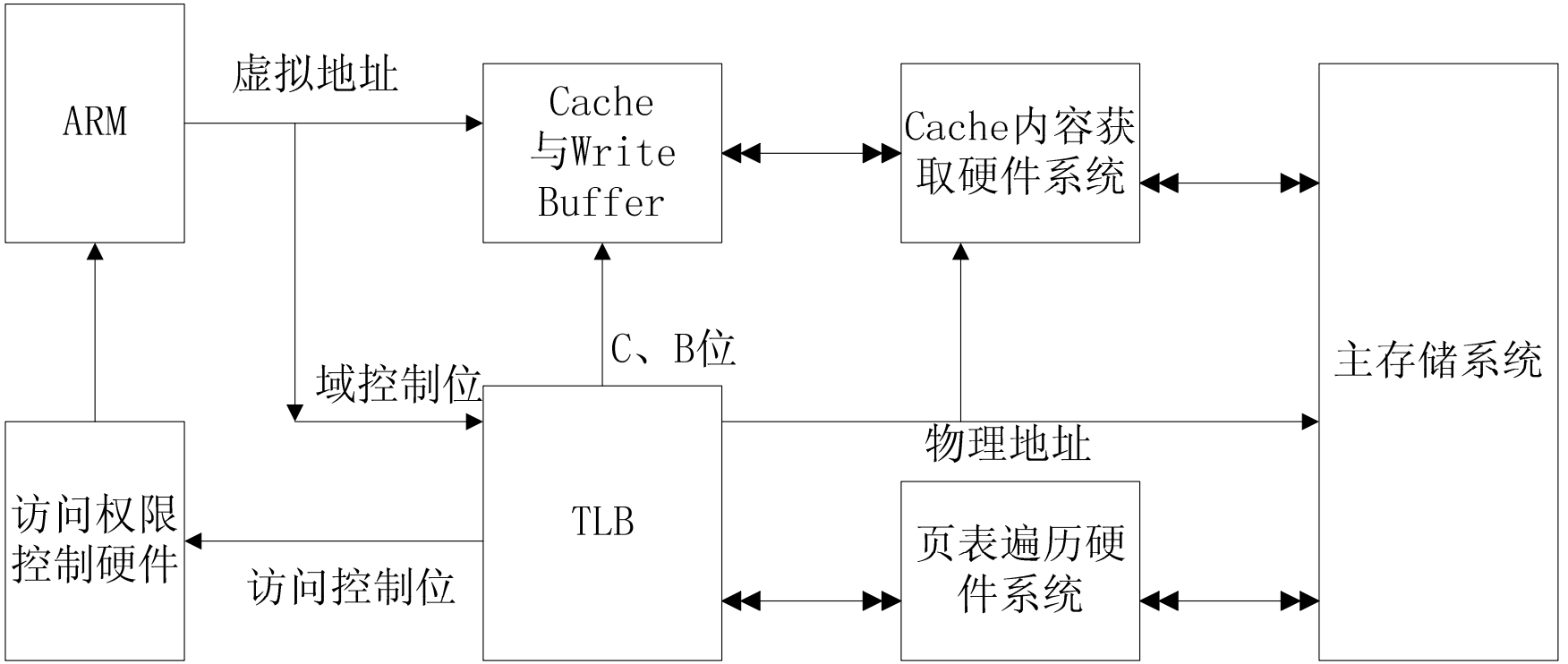
MMU（Memory Management Unit）

- 用于管理虚拟内存系统的硬件。
- 两个主要功能：
  - 将虚地址转换成物理地址
    - MMU关掉时，虚地址直接输出到物理地址总线
  - 控制存储器的存取权限
- TLB，Translation Lookaside Buffers
  - 在MMU中
  - 存放从虚拟地址到物理地址的匹配表
    - 内容包括：虚址及其对应的物理地址，权限，域和映射类型。

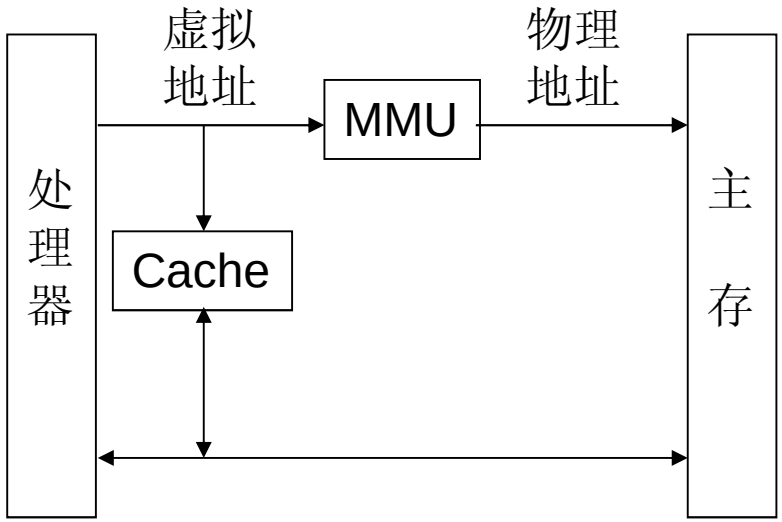
MMU使能

- 当CPU对一虚拟地址进行存取时
  - 首先搜索TLB表以查找对应的物理地址等信息
  - 如果没有查到，则查找translation table（称为Translation Table Walk，TTW）。
  - 经过TTW过程后，将查到的信息保存到TLB。然后根据TLB表项的物理地址进行读写。
- MMU可以锁定某些TLB表项，以提高特定地址的变换速度

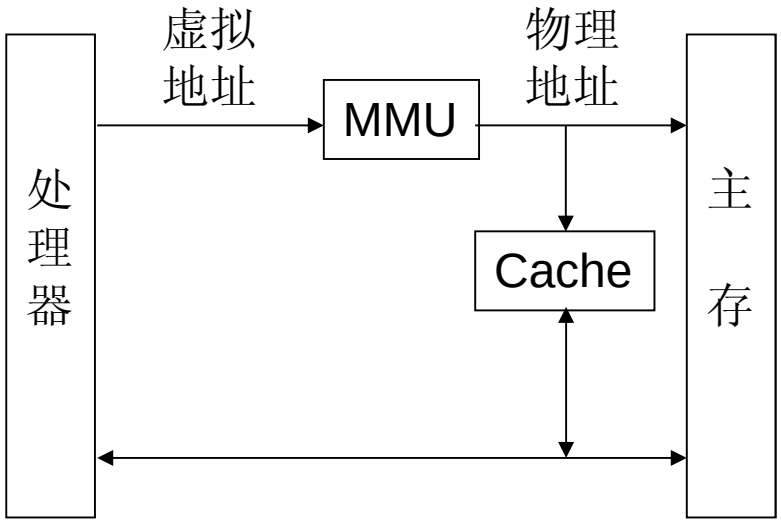
使能MMU时存储访问过程



逻辑Cache和物理Cache



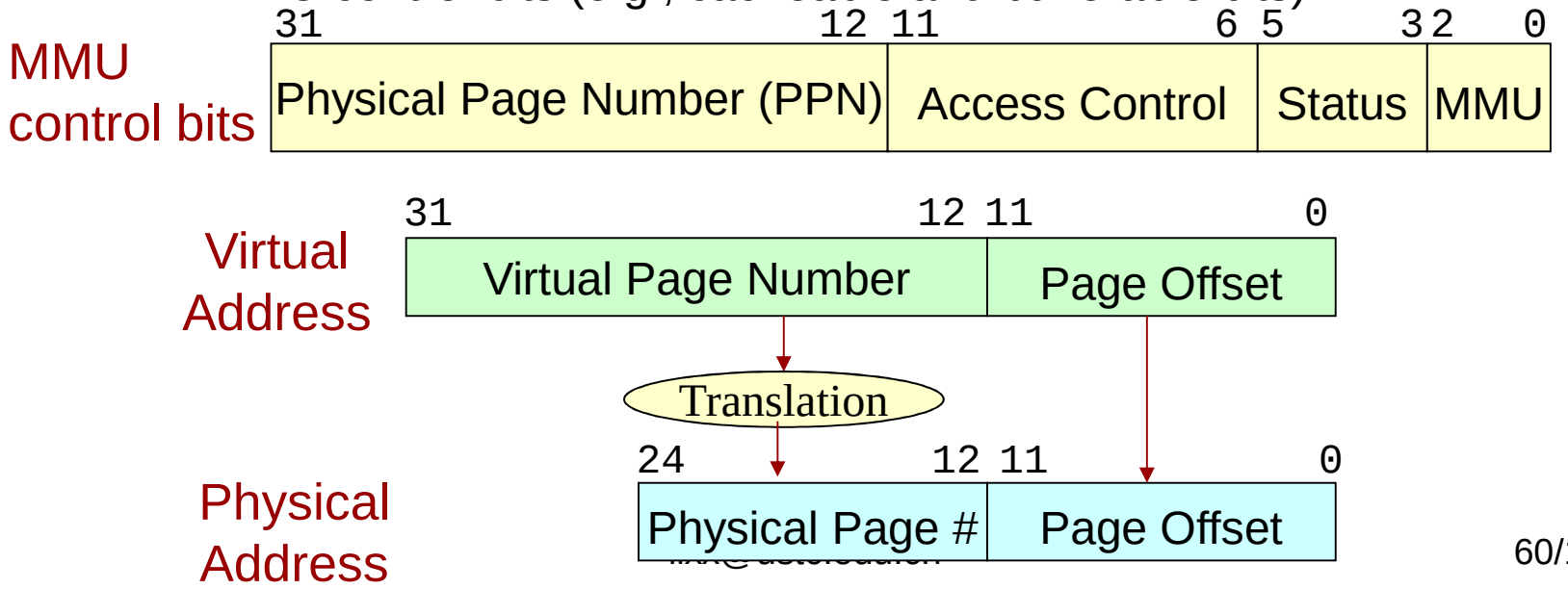
逻辑Cache



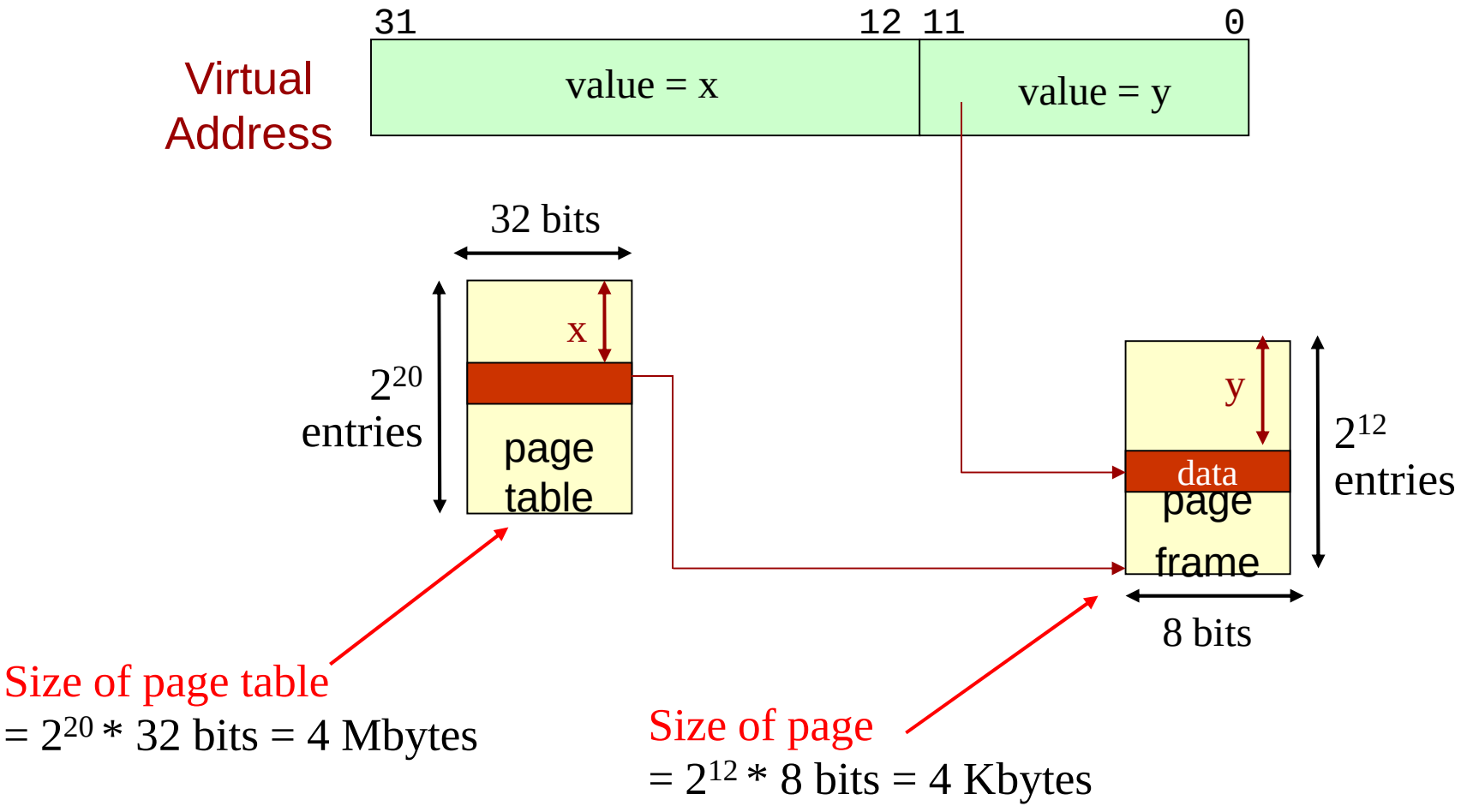
物理Cache

内存虚实映射

- 一个虚拟地址由虚拟页号和页内偏移量两部份组成。
  - 如果页的大小是 4KB，则虚拟地址的0至11位是偏移量，第12位以上是虚拟页号。
- MMU使用页表将虚拟页编号转换到物理的页，并根据页内偏移量访问物理页的正确偏移处 。
  - 每一个页表项(PTE)包括以下信息：
    - 有效标志：表示页表本条目是否有效
    - 本页表条目描述的物理页编号
    - 访问控制位(是否可以写?)
    - MMU control bits (e.g., cacheable and bufferable bits)



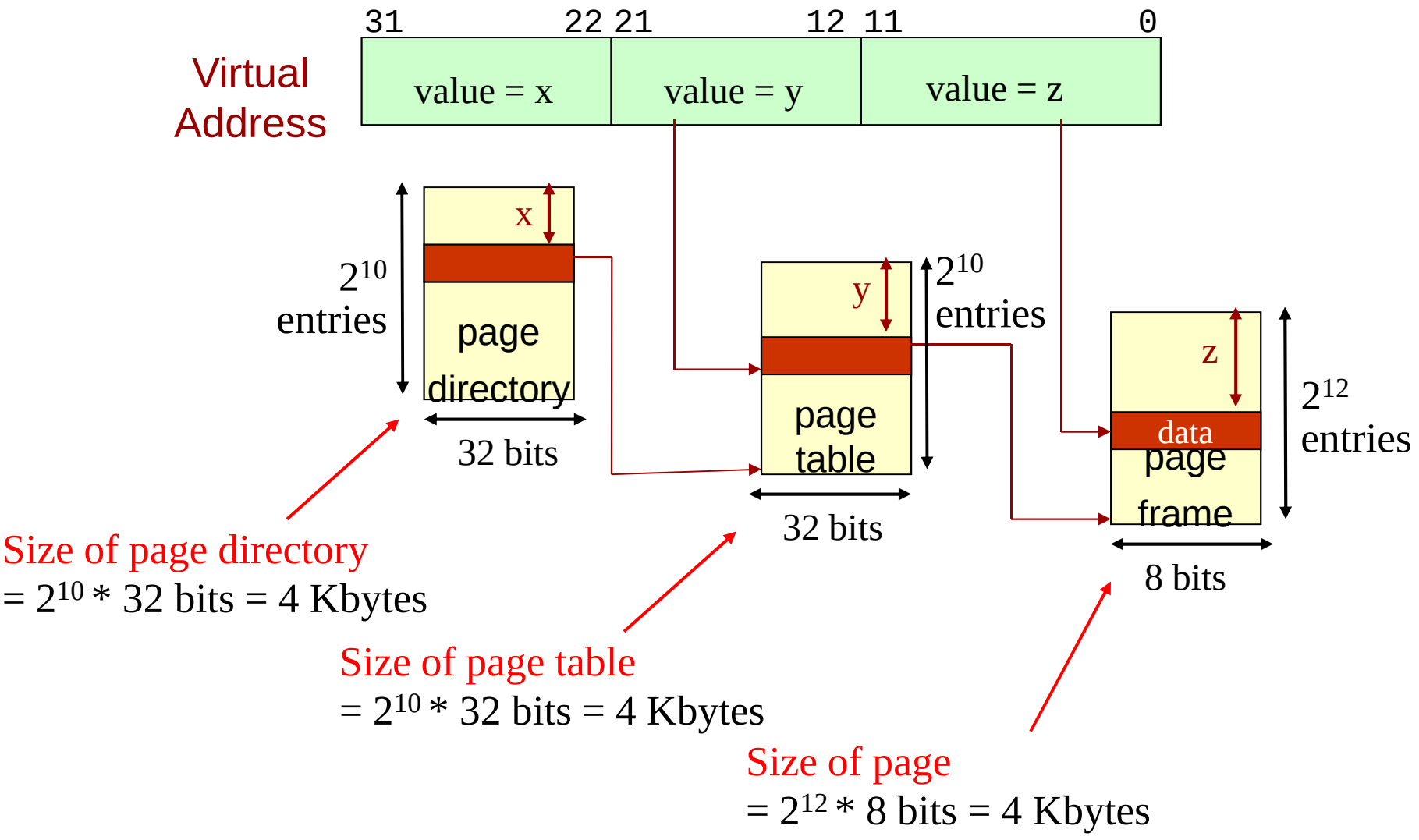
Example: Single-Level Page Table



Single-Level Page Table

- Assumptions
  - 32-bit virtual addresses
  - 4 Kbyte page size = 2^{12} bytes
  - 32-bit address space
- How many virtual page numbers?
  - $2^{32} / 2^{12} = 2^{20}$ = 1,048,576 virtual page numbers = number of entries in the page table
- If each page table entry occupies 4 bytes, how much memory is needed to store the page table?
  - 2^{20} entries * 4 bytes = 2^{22} bytes = 4 Mbytes

Example: Two-level Page Table



Two-Level Page Table

- Assumptions
  - 2^{10} entries in page directory (= max number of page tables)
  - 2^{10} entries in page table
  - 32 bits allocated for each page directory entry
  - 32 bits allocated for each page table entry
- How much memory is needed?
  - Page table size = 2^{10} entries * 32 bits = 2^{12} bytes = 4 Kbytes
  - Page directory size = 2^{10} entries * 32 bits = 2^{12} bytes = 4 Kbytes

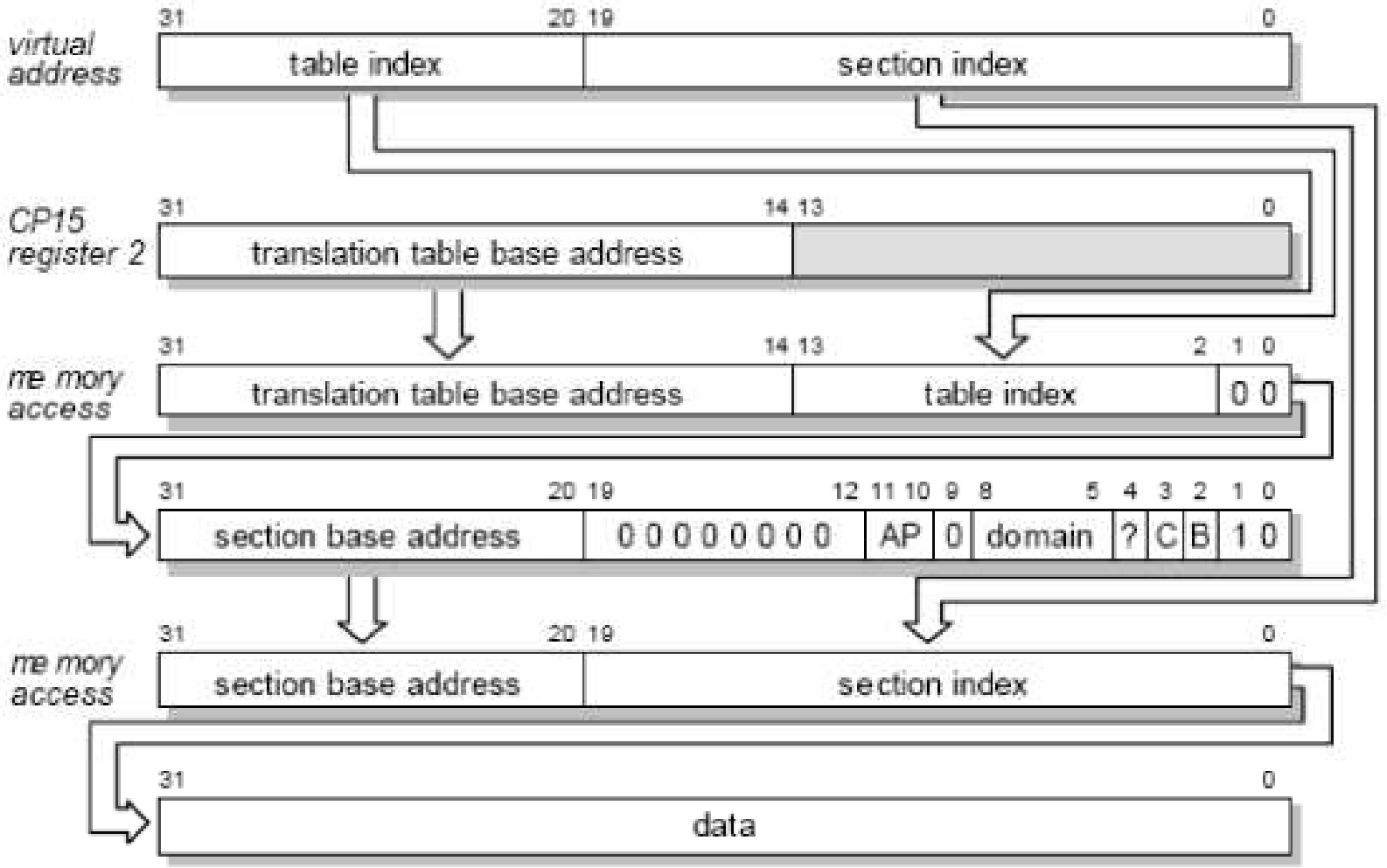
Two-Level Page Table

- Small (typical) system
  - One page table might be enough
    - Page directory size + Page table size = 8 Kbytes of memory would suffice for virtual memory management
  - How much *physical* memory could *this one page table* handle?
    - Number of page tables * Number of page table entries * Page size = $1 * 2^{10} * 2^{12}$ bytes = 4 Mbytes
- Large system
  - You might need the maximum number of page tables
    - Max number of page tables * Page table size = 2^{10} directory entries * 2^{12} bytes = 2^{22} bytes = 4 Mbytes of memory would be needed for virtual memory management
  - How much *physical* memory could *these 2^{10} page tables* handle?
    - Number of page tables * Number of page table entries * Page size = $2^{10} * 2^{10} * 2^{12}$ bytes = 4 Gbytes

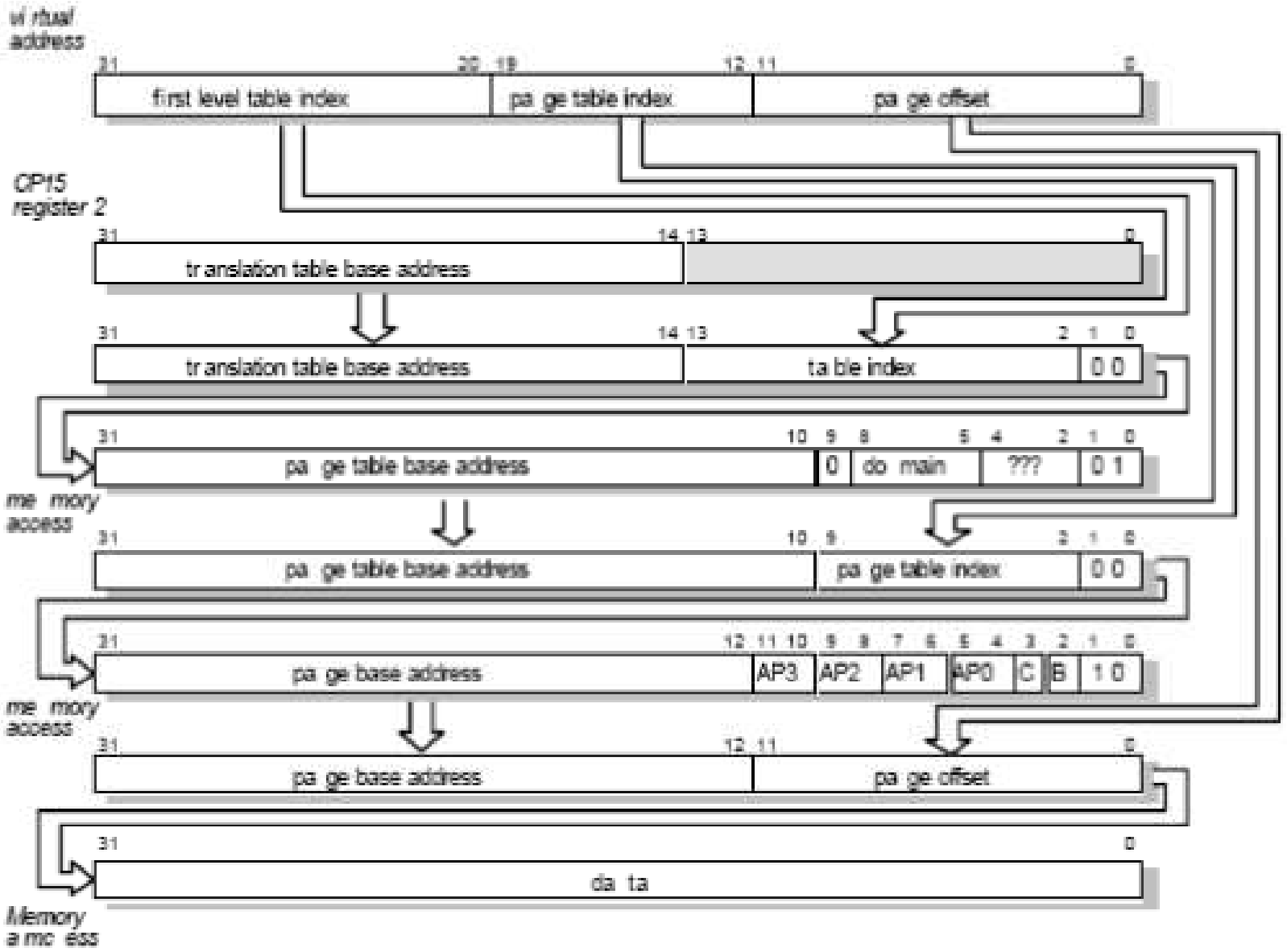
ARM MMU

- Use 2 level page table with TLB
- Memory granularity: 3 options supported
  - 分区模式：1MB sections
  - 分页模式
    - Large pages (64KBytes) - 一次访问16KB
    - Small pages (4KBytes) - 一次访问1KB
- when virtual address not mapped or permission check fails：Puts processor in Abort Mode

Section translation sequence



Small page translation sequence



访问控制

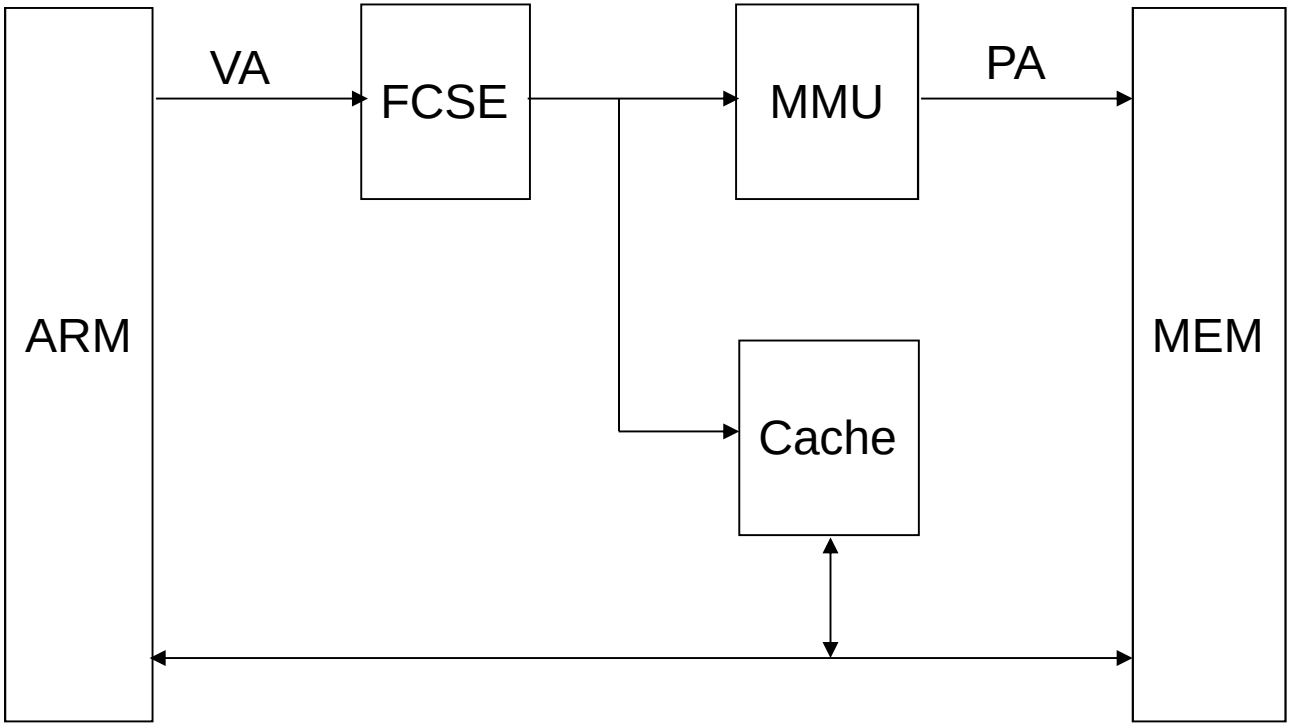
AP	S R	特权级时访问权限	用户级时访问权限
0b00	0 0	没有访问特权	没有访问特权
0b00	1 0	只读	没有访问特权
0b00	0 1	只读	只读
0b00	1 1	不可预知	不可预知
0b01	X X	读/写	没有访问特权
0b10	X X	读/写	只读
0b11	X X	读/写	读/写

快速上下文切换技术FCSE

- FCSE（Fast Context Switch Extension）
  - 通过修改系统中不同进程的虚拟地址，避免在进行进程间切换时造成的虚拟地址到物理地址的重映射，从而提高系统的性能。
- FCSE位于CPU和MMU之间，其责任就是将不同进程使用的相同虚拟地址映射为不同的虚拟空间，使得在上下文切换时无需重建TLB等。

FCSE

- ARM将4GB虚空间划分为128个进程块
  - 每个进程块32M，可以包含一个程序



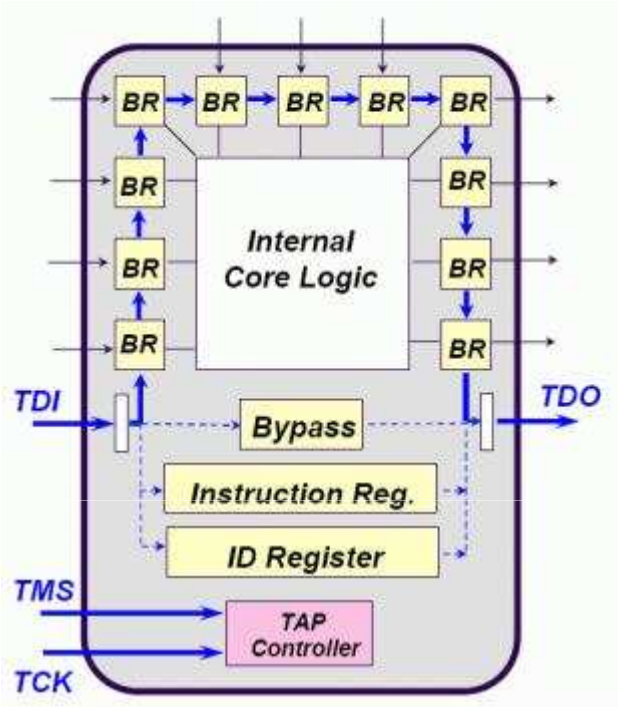
ARM对系统测试与调试的支持

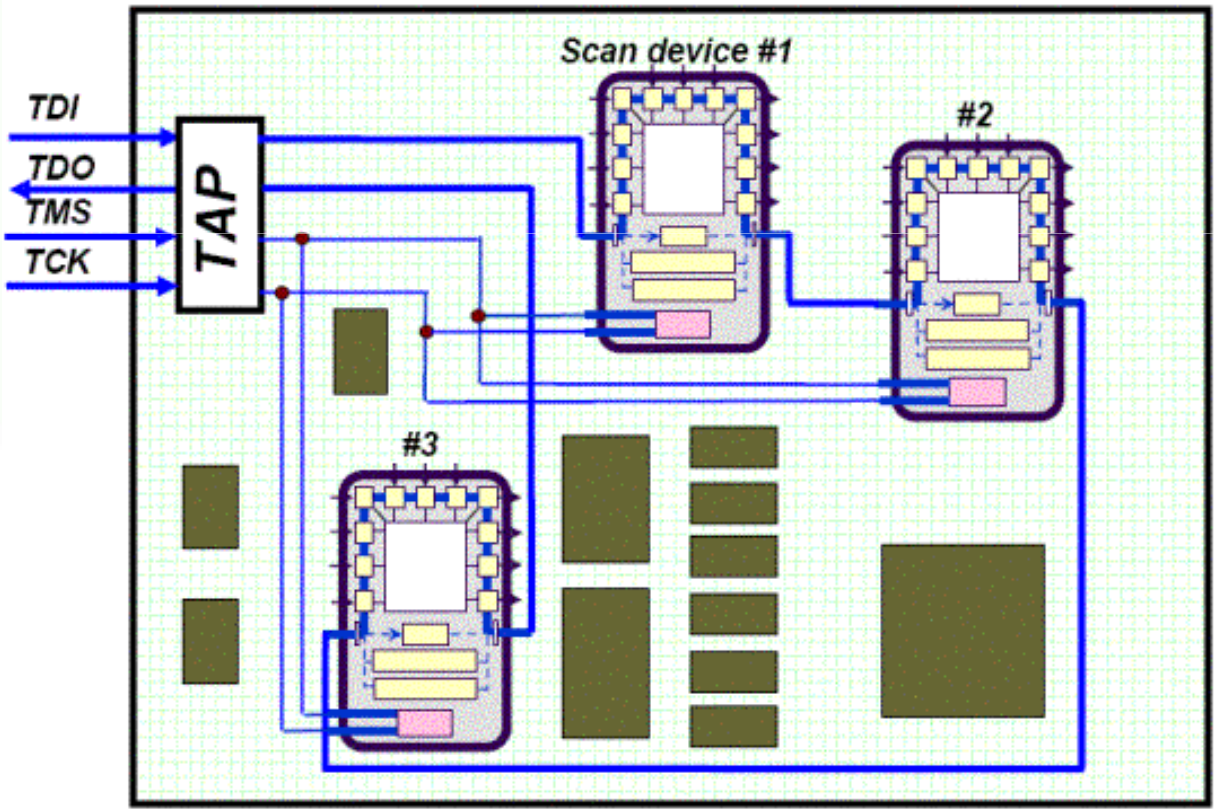
- 嵌入式ICE-RT：在线CPU仿真器，支持应用软件调试
  - 支持软件断点和硬件断点设置
  - 设置复杂的断点触发条件
  - 实时跟踪目标程序运行
  - 提供shadow RAM，实时查看内存和变量
- 含
  - JTAG (Joint Test Action Group)
  - 嵌入式跟踪宏单元(ETM，Embedded Trace Macrocell)
    - 在JTAG的基础上，增加了硬件断点
    - 设计成驻留在ARM处理器上，用以监控内部总线，并能以核速度无妨碍地跟踪指令和数据的访问。

JTAG

- IEEE1149.1标准，对PCB中的集成电路进行互连完整性测试
  - 检测开路和短路
  - 只需将JTAG接口同JTAG仿真器硬件连接，利用相应的调试工具通过边界扫描便可以被测试。
- JTAG接口定义：
  - TMS： 测试模式选择（Test Mode Select）
    - 通过TMS信号控制JTAG状态机的状态
  - TCK ：JTAG的时钟信号
  - TDI：数据输入信号
  - TDO ： 数据输出信号
  - nTRST：JTAG复位信号
    - 复位JTAG的状态机和内部的宏单元（Macrocell）
- 利用ARM处理器中的调试模块的功能，通过其JTAG边界扫描口来与仿真器连接，对芯片的内部总线，I/O口等内部工作状态信息进行监控，从而达到调试的目的。

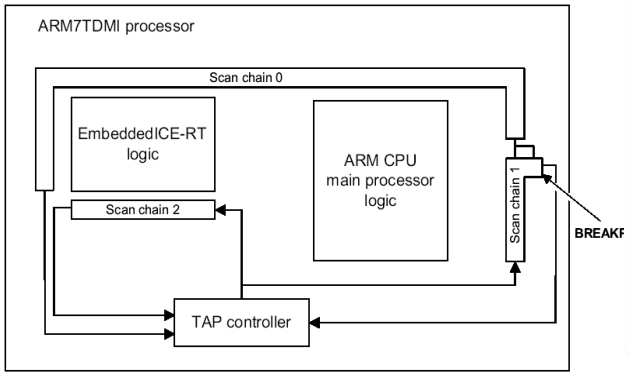
JTAG结构

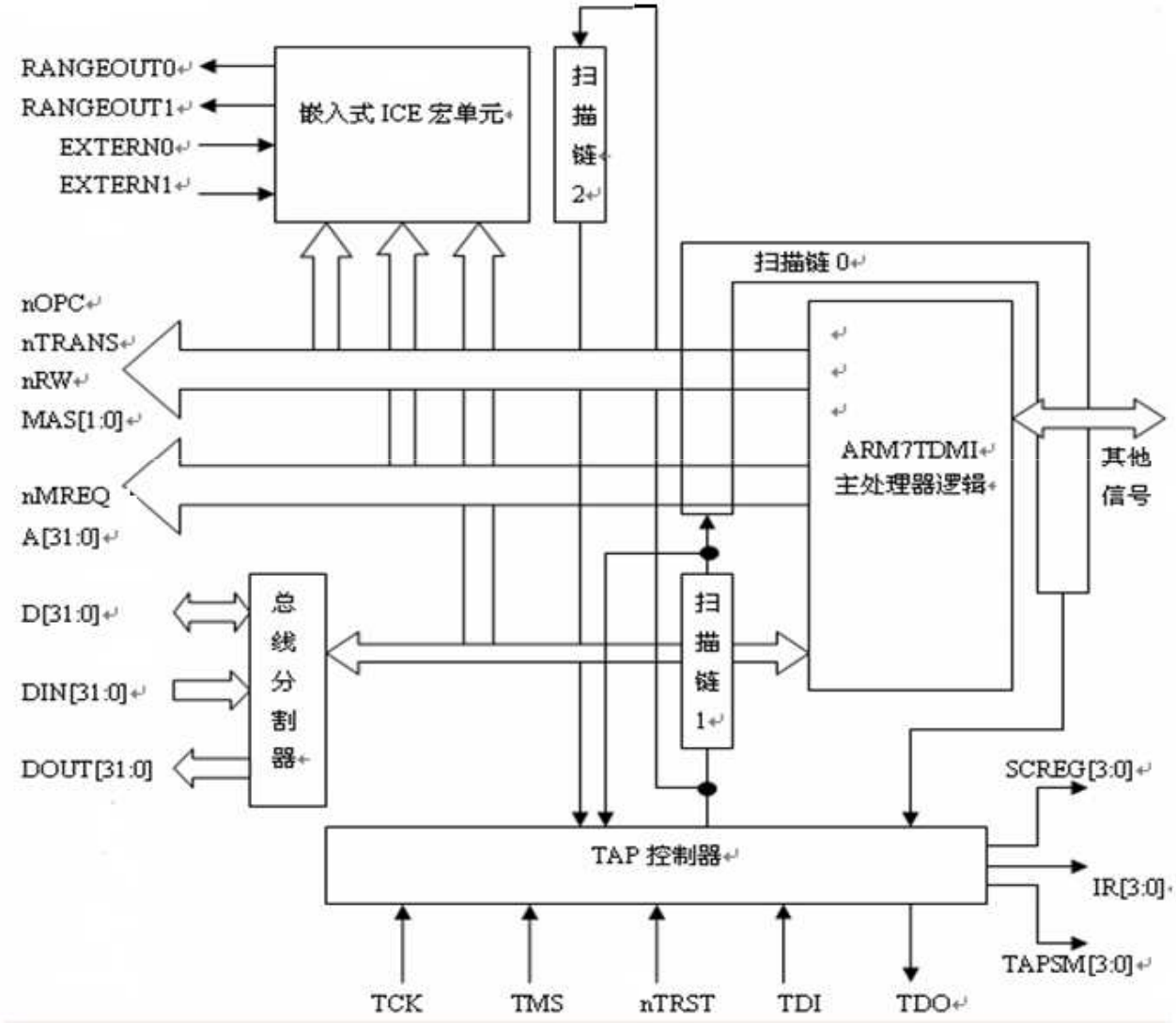

BR
Internal
Core Logic
TDI
TDO
Bypass
Instruction Reg.
ID Register
TMS
TAP
Controller
TCK


Scan device #1
#2
#3
TDI
TDO
TMS
TCK
TAP

ARM处理器的JTAG标准链

- Scan Chain 0、1、2





Scan Chain 0、1、2

- **Scan Chain0**
  - 有**113**个扫描单元，包括**ARM**核的所有**I/O**、地址数据总线和输入输出控制信号。
  - 可以通过这条链得到**ARM7TDMI**所有的内核信息。
- **Scan Chain1**
  - 有**33**个扫描单元，包括**ARM**核的数据总线和一个断点控制信号。
  - 可以控制**ARM**核执行指定的指令，从而实现对**ARM**的内部寄存器、协处理器以及外部存储器的读写操作。
- **Scan Chain2**
  - 有**38**个扫描单元，通过控制**EmbeddedICE**宏单元，实现对**ARM**执行指令的断点、观察点控制。
  - 通过对**EmbeddedICE**的控制，对**EmbeddedICE**中寄存器的读取，可以获得**ARM**内核的状态，为程序设置断点以及读取**Debug**通信通道。

内容提要

- **ARM体系结构概览**
- **ARM编程模型**
  - **ARM微处理器的工作状态**
  - **ARM体系结构的存储器模式**
  - **ARM微处理器的操作模式**
  - **ARM体系结构的寄存器组织**
  - **ARM微处理器的异常状态**

4.1 ARM微处理器的工作状态

- ARM7TDMI有两种工作状态:
  - ARM - 32-bit，按字排列的ARM指令集
  - Thumb -16-bit，按半字排列的Thumb指令集
    - 可能通过BX指令(分支和交换指令）在ARM状态和Thumb状态之间切换

```
;从Arm状态切换到Thumb状态
    LDR     R0,=Lable+1
    BX      R0
;从Thumb状态切换到ARM状态
    LDR     R0,=Lable
    BX      R0
```

跳转地址标号

地址最低位为1，表示切换到Thumb状态

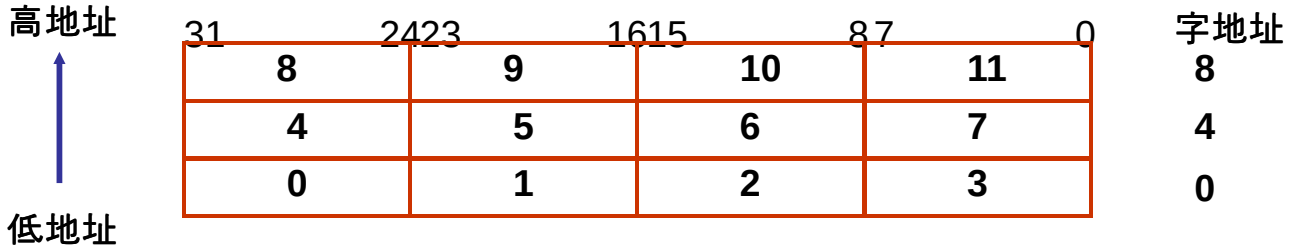
地址最低位为0，表示切换到ARM状态

4.1 ARM微处理器的工作状态

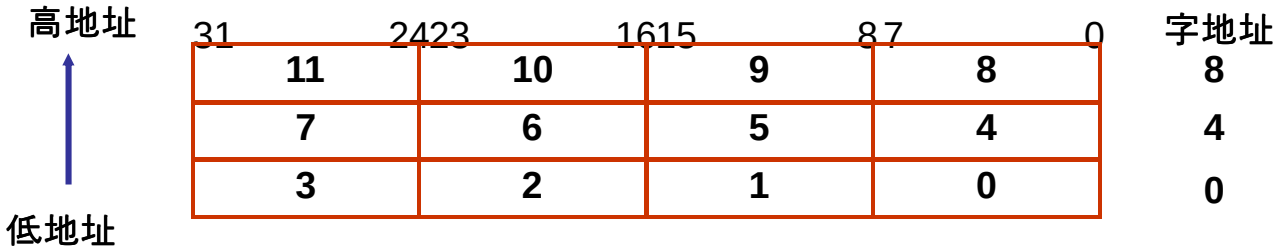
- 在程序执行的过程中，处理器可以在两种状态下切换。需要强调的是：
  - **ARM**和**Thumb**之间状态的切换不影响处理器的模式或寄存器的内容。
  - **ARM**处理器在开始执行代码时，只能处于**ARM**状态。

4.2 ARM体系结构的存储器模式

- 大端模式
  - ◆最高位字节保存在最低位地址
  - ◆字由最低位字节的字节地址寻址

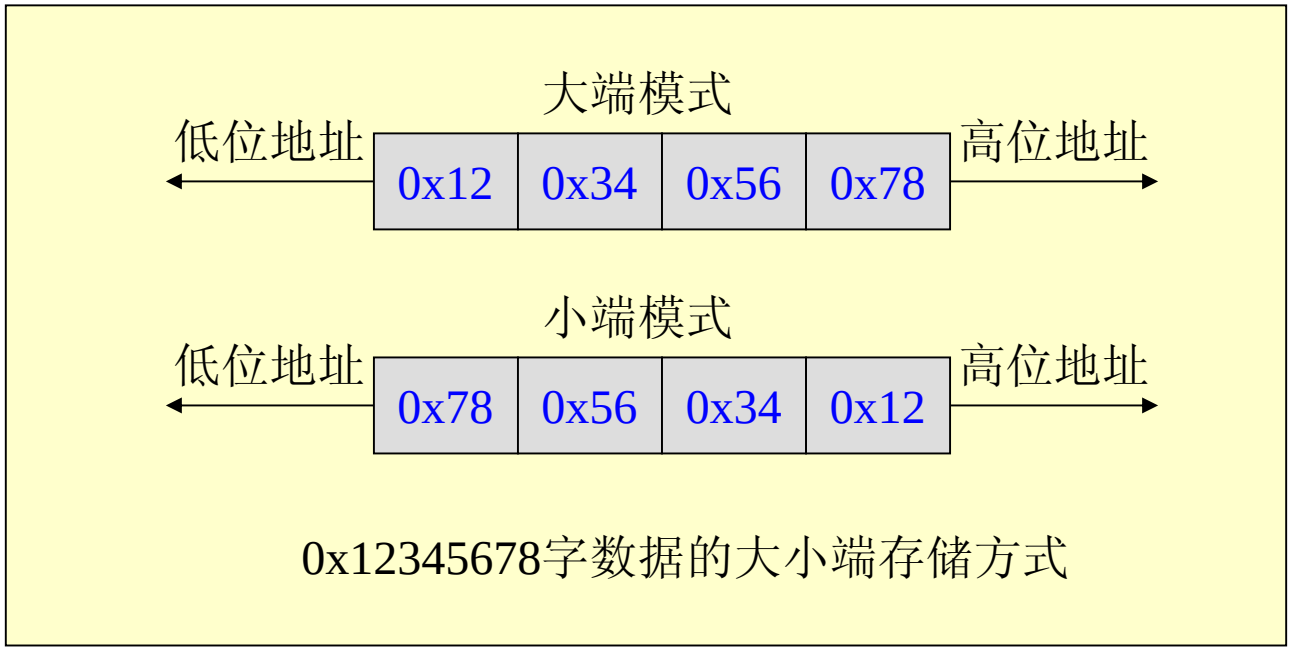


- 小端模式
  - ◆最低位字节保存在最低位地址
  - ◆字由最低位字节的字节地址寻址



存储模式选择

- 小端模式是ARM处理器的默认形式
  - 一个基于ARM内核的芯片可以只支持大端模式或小端模式，也可以两者都支持。
- 一般使用芯片的引脚来配置
  - 在ARM指令集中不包含任何直接选择大小端的指令



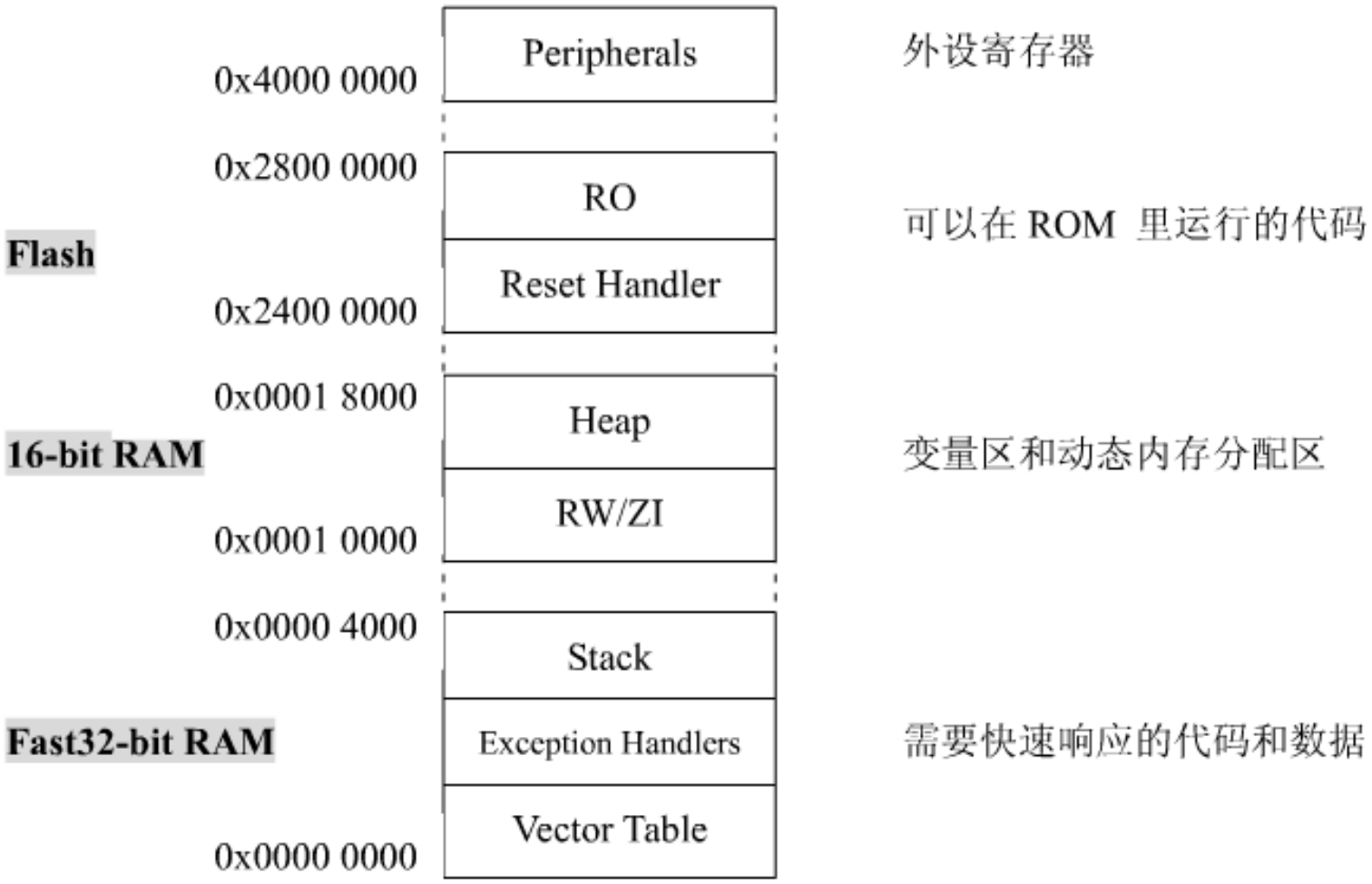
0x12345678字数据的大小端存储方式

ARM地址空间与存储器组织

- 寻址空间为4GB
  - 使用单一平面的2^{32}个8位字节地址空间。
    - 字节地址为无符号数：0到$2^{32}-1$。
  - 如果地址向上或向下溢出地址空间，通常会发生翻转。
    - 如果在取指操作时地址发生溢出，只要没有执行预取的无效指令，就不会导致异常。
- I/O空间
  - 存储器映像方式
    - 通常，存储器映射的I/O位置没有cache和缓冲区。

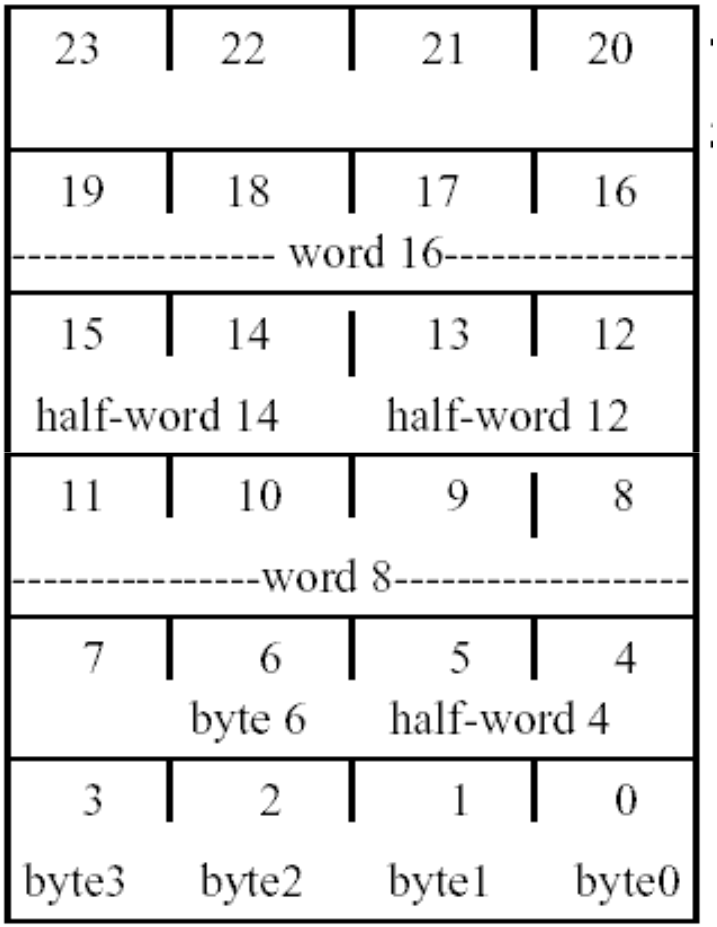
存储器组织

典型的地址空间分配


0x4000 0000
Peripherals
外设寄存器
0x2800 0000
RO
Flash
Reset Handler
0x2400 0000
可以在 ROM 里运行的代码
0x0001 8000
Heap
16-bit RAM
RW/ZI
0x0001 0000
变量区和动态内存分配区
0x0000 4000
Stack
Fast32-bit RAM
Exception Handlers
Vector Table
0x0000 0000
需要快速响应的代码和数据

数据类型

- 3种：字节（8位）、半字（16位）、字（32位）
  - 无符号数：N位数据使用正常的二进制格式表示，范围为0～2^N-1的非负整数；
  - 有符号数：N位数据值使用补码表示，范围为-2^{N-1}～+2^{N-1}-1的整数；
- 字对齐
  - 字需要4字节对齐
    - 地址的低两位为00
  - 半字需要2字节对齐
    - 地址的最低位为0



非对齐的数据访问操作

- 对于Load/Store操作，如果是非对齐的数据访问操作，系统定义了三种可能的结果：
  - 执行的结果不可预知
  - 忽略字单元地址的低两位的值，即访问地址为(address AND 0XFFFFFFFC)的字单元；忽略半字单元地址的最低位的值，即访问地址为(address AND 0XFFFFFFFE)的半字单元。
  - 忽略字单元地址的低两位的值；或忽略半字单元地址的最低位的值；由存储系统实现这种忽略。也就是说，这时该地址值原封不动地送到存储系统。
- 当发生非对齐的数据访问时，到底采用上述三种方法中的哪一种，是由各指令指定的。

4.3 ARM微处理器的工作模式

- ARM7TDMI处理器有7种操作模式:
  - 用户模式(usr): 正常的程序执行模式
  - 系统模式(sys): 运行具有特权的操作系统任务
  - 快速中断模式(fiq): 高速数据传输或通道处理
  - 中断模式(irq): 用于通用中断处理
  - 管理员模式(svc): 操作系统使用的保护模式
  - 数据访问中止模式(abt): 当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护
  - 未定义模式(und): 当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

操作模式分类

工作模式	功能	M[4:0]
用户模式（usr）	正常的程序执行状态	10000
快速中断模式（fiq）	用于高速数据传输或通道处理	10001
外部中断模式（irq）	用于通用的中断处理	10010
管理模式（svc）	操作系统的保护模式	10011
中止模式(abt)	用于虚拟存储及存储保护	10111
未定义指令模式（und）	用于支持硬件协处理器的软件仿真	11011
系统模式（sys）	运行特权级的操作系统任务	11111

异常模式

特权模式

用户模式和特权模式

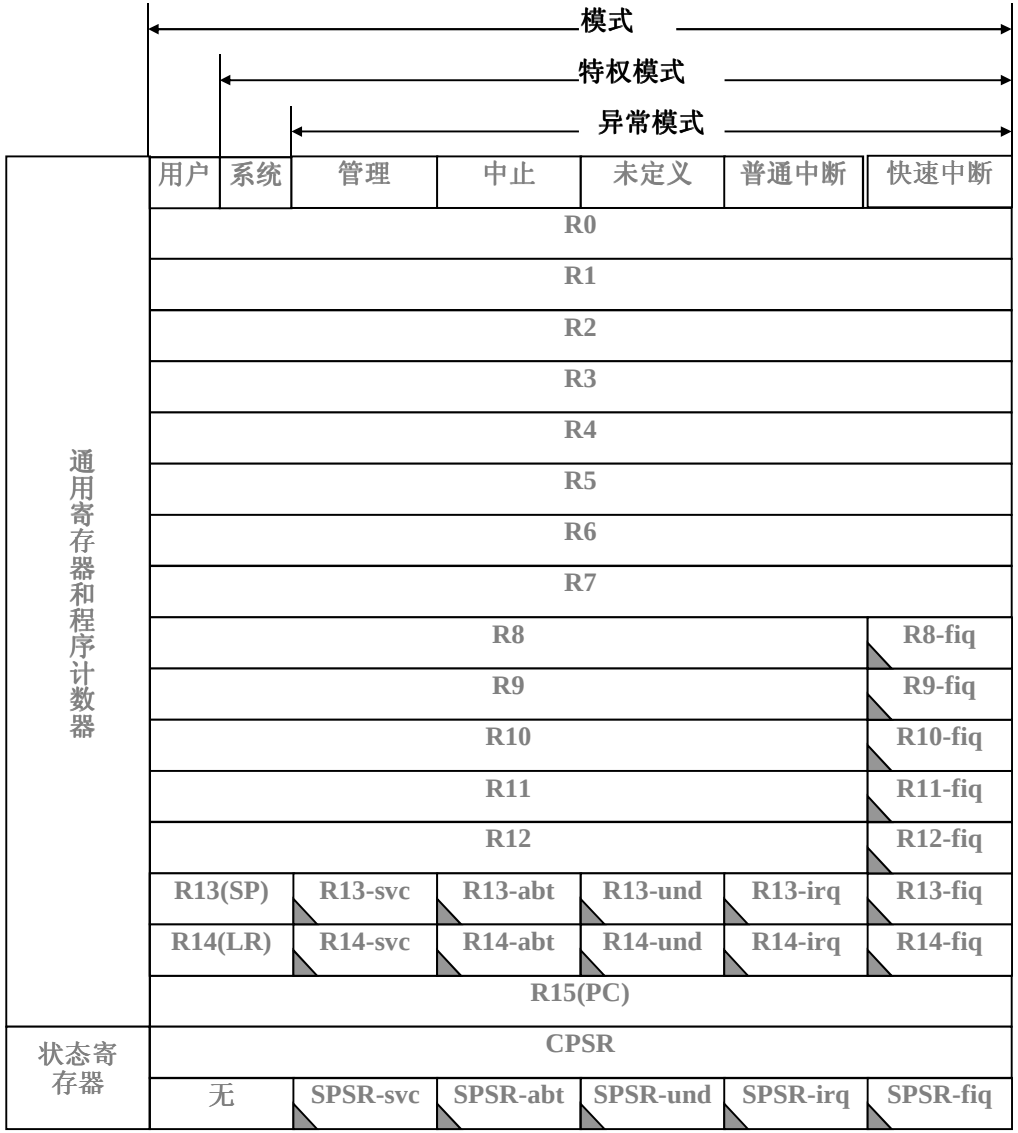
- 特权模式
  - 除了用户模式之外的其他6种处理器模式
  - 特权模式下，程序可以访问所有的系统资源，也可以进行处理器模式的切换。
- 异常模式
  - 特权模式中，除系统模式外，其他5种模式
- 用户模式
  - 不能够访问一些受操作系统保护的系统资源
  - 不能直接进行处理器模式的切换
    - 当需要进行处理器模式切换时，应用程序可以产生异常处理，在异常处理中进行处理器模式的切换。

模式切换

- 用户模式
  - 通过修改CPSR进入
- 异常模式
  - 外部中断、异常处理
  - 当应用程序发生异常时，进入相应的异常模式
  - 每一种异常模式下都有一组寄存器，可以保证在进入异常模式时，用户模式下的寄存器不被破坏。
- 管理模式
  - 复位、软中断
- 系统模式：主要供操作系统任务使用。
  - 通常操作系统的任务需要访问所有的系统资源
  - 该任务仍然使用用户模式的寄存器组，而不是使用异常模式下相应的寄存器组，保证当异常中断发生时任务状态不被破坏。
  - 通过修改CPSR进入

4.4 ARM体系结构的寄存器组织

- 37寄存器
  - 31个通用32位寄存器(包括程序计数器PC)
    - 未分组寄器R0-R7:
      - ✓ 指向同一个物理寄存器
    - 分组寄存器R8-R14
      - ✓ 命名规则：R13_<mode>、R14_<mode>
        - » Mode：usr、fiq、irq、svc、abt、und。
    - 程序计数器PC（R15）
  - 6 个状态寄存器

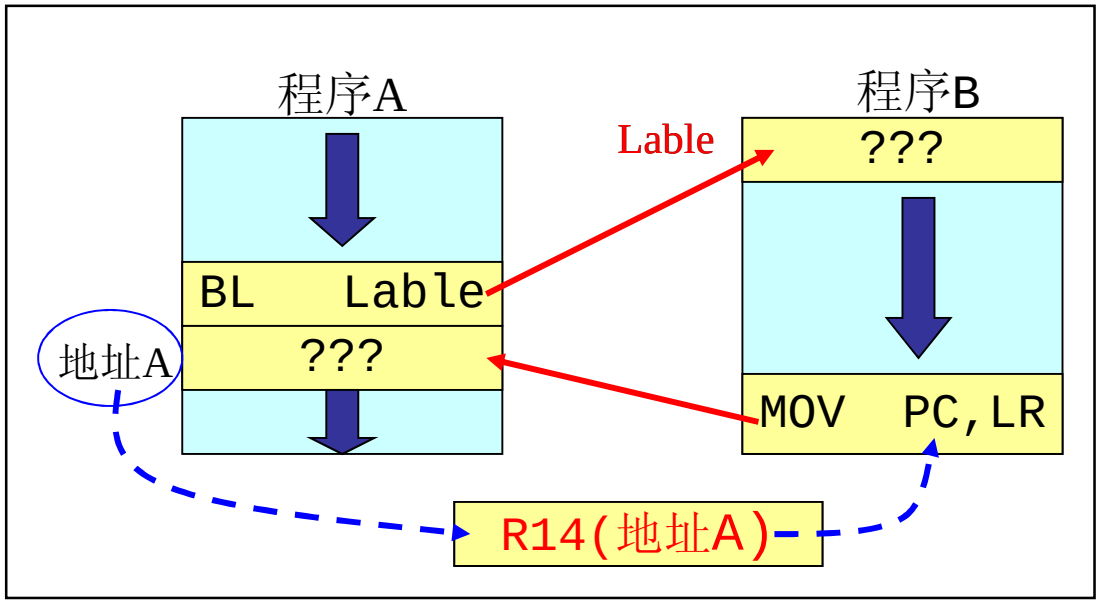


4.4 ARM体系结构的寄存器组织

- R0 到 R15 可以直接访问
- R0 到 R14 是通用寄存器
- R13: 堆栈指针 (sp) (通常)
  - ◆每种处理器模式都有单独的堆栈,在用户应用程序的初始化部分，一般都要初始化每种模式下的R13，使其指向该运行模式的栈空间
  - ◆Thumb中，某些指令强制性的要求使用R13作为堆栈指针

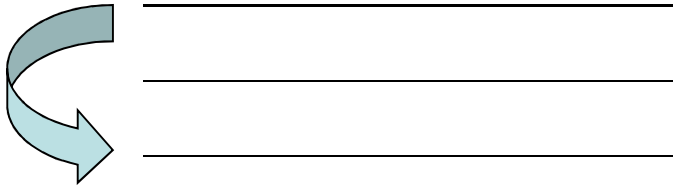
4.4 ARM体系结构的寄存器组织

- R14：子程序链接寄存器（Subroutine Link Register，LR）
  - 当执行BL子程序调用指令时，R14←R15（程序计数器PC）。
  - 与之类似，当发生中断或异常时，对应的分组寄存器R14_svc、R14_irq、R14_fiq、R14_abt和R14_und用来保存R15的返回值。
  - 其他情况下，R14用作通用寄存器。



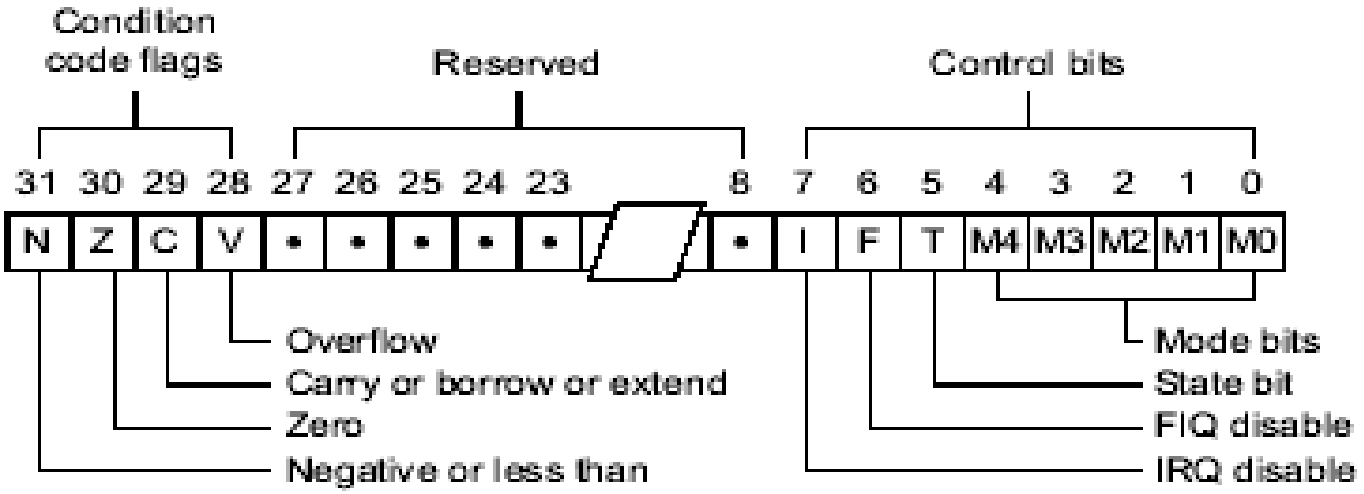
4.4 ARM体系结构的寄存器组织

- **R15：程序计数器 (PC)**
  - **R15**虽然也可用作通用寄存器，但一般不这么使用，否则程序的执行结果可能未知。
  - 当向**R15**中写入一个地址值时，程序将跳转到该地址执行
    - 在**ARM**状态下，位**[1:0]**为**00**；在**Thumb**状态下，位**[0]**为**0** 。
      - 由于在**ARM**状态下指令总是是字对齐的，所以**R15**值的第**0**位和第**1**位总为**0**，**PC**［**31**：**2**］用于保存地址。
  - 由于**ARM**体系结构采用了多级流水线技术，对于**ARM**指令集而言，**PC**总是指向当前指令的下两条指令的地址，即**PC**的值为当前指令的地址值加**8**个字节。



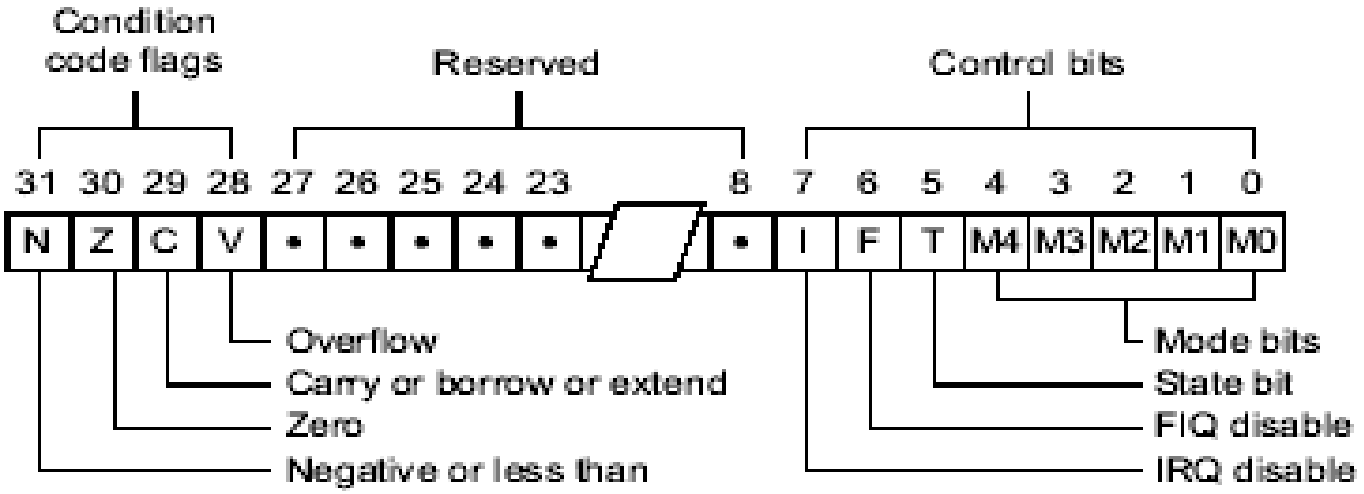
4.4 ARM体系结构的寄存器组织

- **CPSR – Current Program Status Register**，当前程序状态寄存器
  - **CPSR**可在任何运行模式下被访问，它包括条件标志位、中断禁止位、当前处理器模式标志位，以及其他一些相关的控制和状态位。



程序状态寄存器CPSR

- N, Z, C and V 条件码标志
  - 可以在处理器中作为数学和逻辑操作改变
  - 可以被所有的指令测试，以决定指令是否被执行
  - N : Negative. Z : Zero. C : Carry. V : oVerflow
- I and F 位是中断禁止位
- M0, M1, M2, M3 and M4 位是模式位



在**ARM**状态下，绝大多数的指令都是有条件执行的。
在**Thumb**状态下，仅有分支指令是有条件执行的。
条件码标志各位的具体含义：

标志位	含　义
N	当用两个补码表示的带符号数进行运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零；
Z	Z=1 表示运算的结果为零；Z=0表示运算的结果为非零；
C	可以有4种方法设置C的值： －加法运算（包括比较指令CMN）：当运算结果产生了进位时（无符号数溢出），C=1，否则C=0。 －减法运算（包括比较指令CMP）：**运算产生借位（无符号数溢出），C=0**，否则C=1 －对于包含移位操作的非加/减运算指令，C为移出值的最后一位。 －对于其他的非加/减运算指令，C的值通常不改变。
V	可以有2种方法设置V的值： －对于加/减法运算指令，当操作数和运算结果为二进制的补码表示的带符号数时，V=1表示符号位溢出。 －对于其他的非加/减运算指令，V的值通常不改变。
Q	在ARM v5及以上版本的E系列处理器中，用Q标志位指示增强的DSP运算指令是否发生了溢出。在其他版本的处理器中，Q标志位无定义。

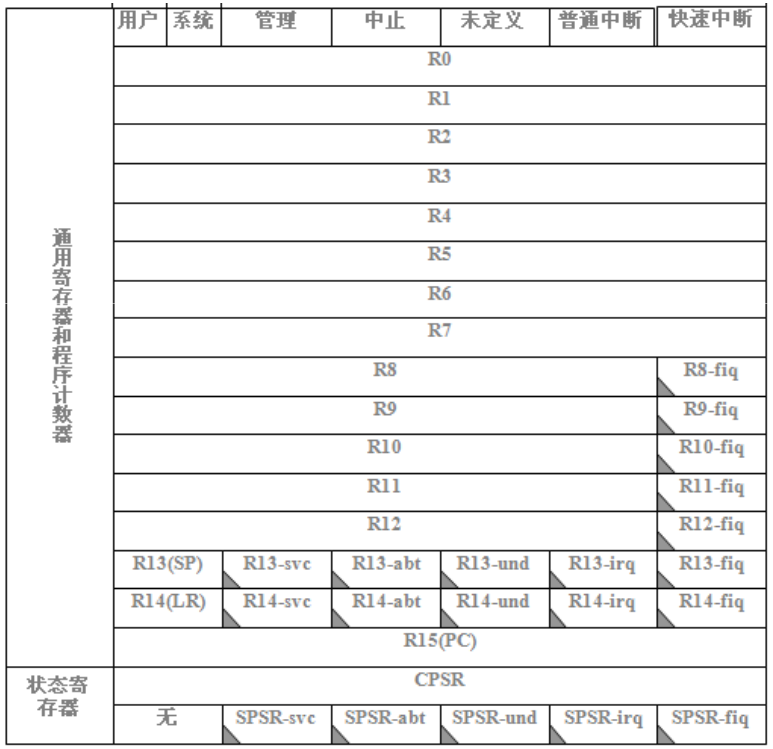
程序状态寄存器CPSR的模式位

M[4:0]	Mode	Visible Thumb-state registers	Visible ARM-state registers
10000	User	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR
10001	FIQ	r0–r7, SP_fiq, LR_fiq, PC, CPSR, SPSR_fiq	r0–r7, r8_fiq–r14_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	r0–r7, SP_irq, LR_irq, PC, CPSR, SPSR_irq	r0–r12, r13_irq, r14_irq, PC, CPSR, SPSR_irq
10011	Supervisor	r0–r7, SP_svc, LR_svc, PC, CPSR, SPSR_svc	r0–r12, r13_svc, r14_svc, PC, CPSR, SPSR_svc
10111	Abort	r0–r7, SP_abt, LR_abt, PC, CPSR, SPSR_abt	r0–r12, r13_abt, r14_abt, PC, CPSR, SPSR_abt
11011	Undefined	r0–r7, SP_und, LR_und, PC, CPSR, SPSR_und	r0–r12, r13_und, r14_und, PC, CPSR, SPSR_und
11111	System	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR

并不是所有的运行模式位的组合都是有效地，其他的组合结果会导致处理器进入一个不可恢复的状态。

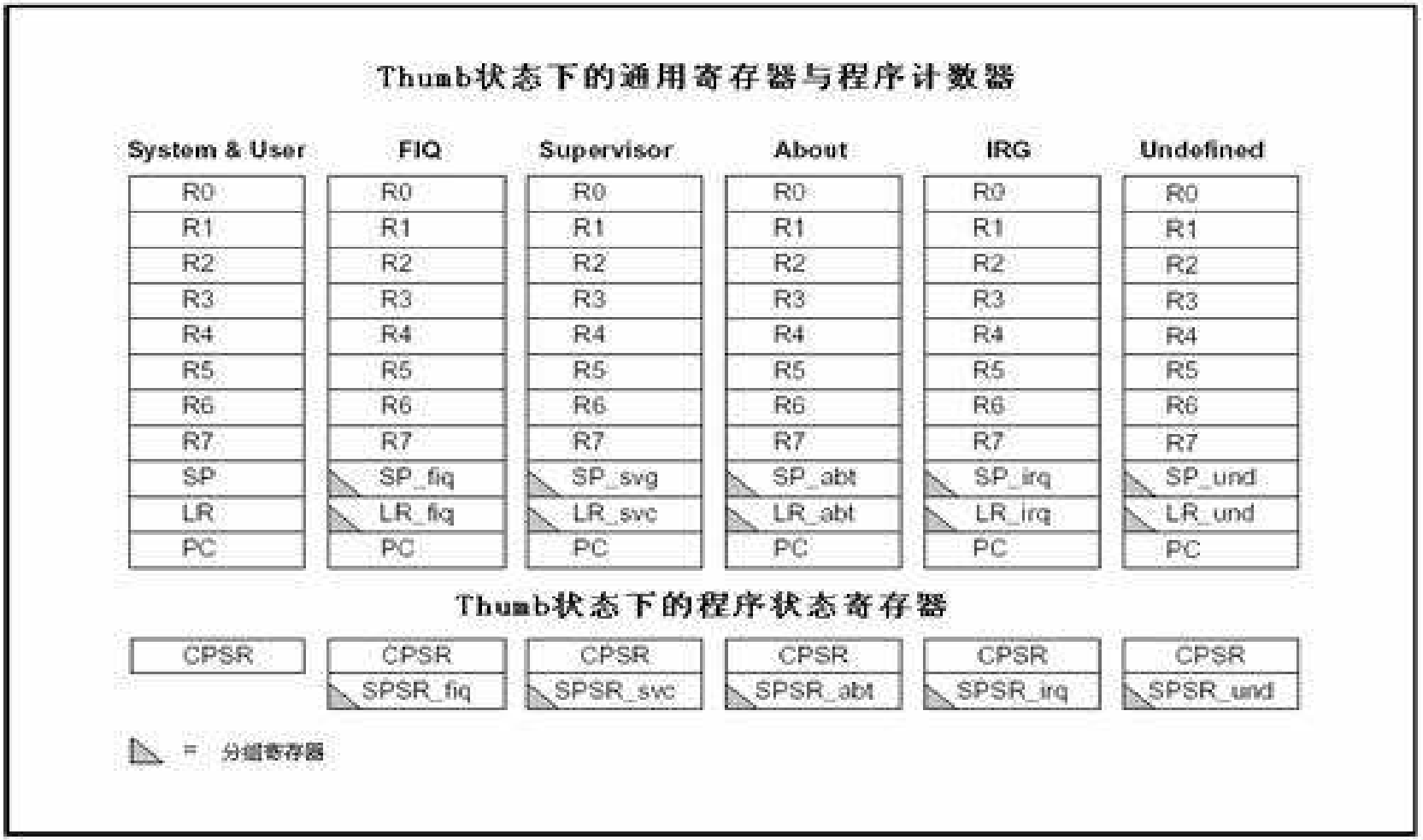
4.4 ARM体系结构的寄存器组织

- **5个SPSRs--Saved Program Status Register** 程序状态保存寄存器
  - 当异常发生时，**SPSR**用于保存**CPSR**的当前值，从异常退出时则可由**SPSR**来恢复**CPSR**。
  - 由于用户模式和系统模式不属于异常模式，它们没有**SPSR**，当在这两种模式下访问**SPSR**，结果是未知的。

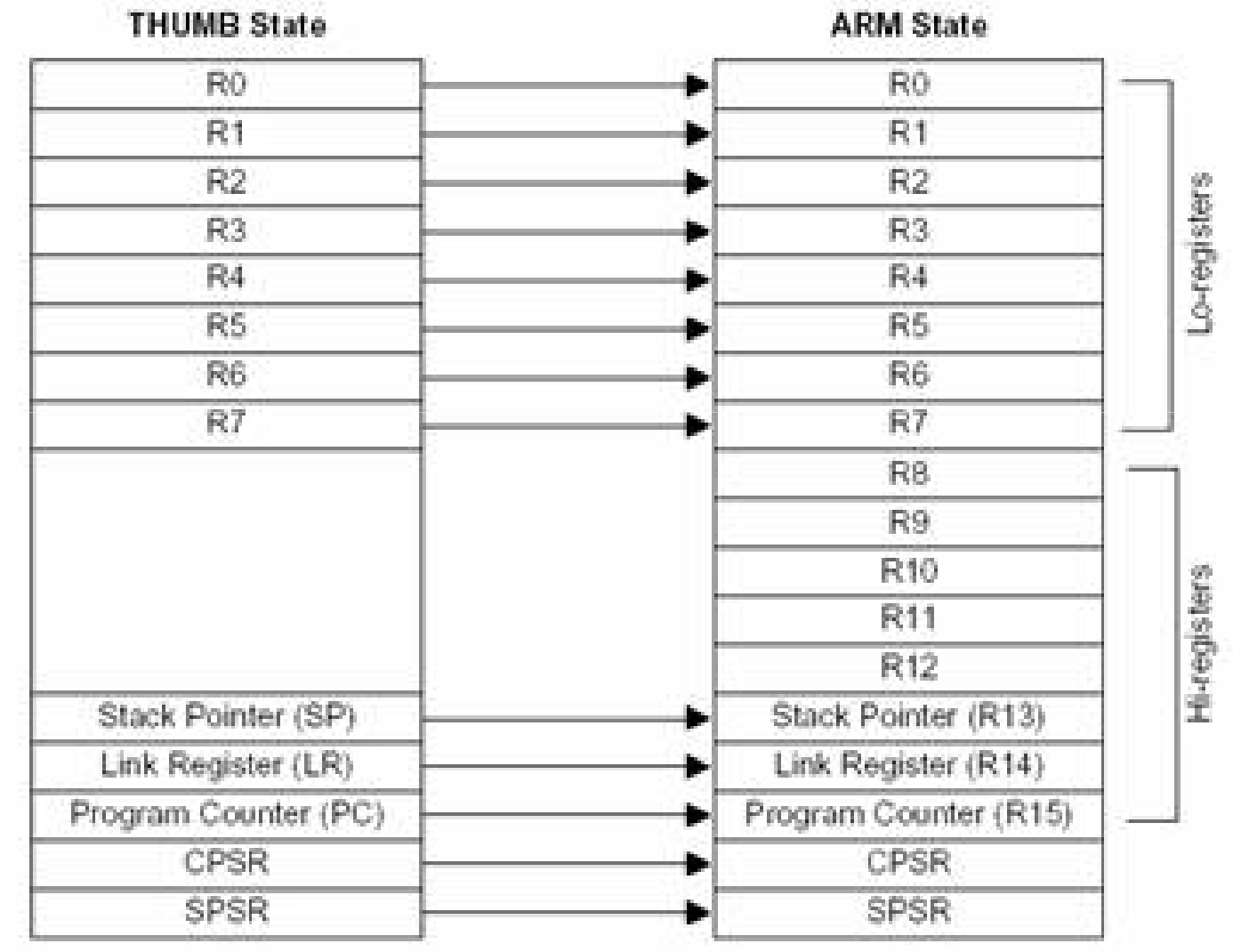


Thumb状态下的寄存器组织

Thumb状态下可以直接访问8个通用寄存器（R7～R0）、程序计数器（PC）、堆栈指针（SP）、连接寄存器（LR）和CPSR。同时，在每一种特权模式下都有一组SP、LR和SPSR。



Thumb寄存器与ARM寄存器关系



访问THUMB状态下的高位寄存器

- 在**Thumb**状态下，**Hi-registers**（高位寄存器）**R8～R15**并不是标准寄存器集的一部分，但可使用汇编语言程序受限制的访问这些寄存器，将其用作快速的暂存器。
  - 使用带特殊变量的**MOV**指令，数据可以在低位寄存器和高位寄存器之间进行传送；
  - 高位寄存器的值可以使用**CMP**和**ADD**指令进行比较或加上低位寄存器中的值。

4.5 异常(exceptions)

- 异常类型（7种）
  - 复位
  - 数据中止
  - FIQ
  - IRQ
  - 预取中止
  - 未定义指令
  - 软中断

User32	Fiq32	Supervisor32	Abort32	IRQ32	Undefined32
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
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13(SP)	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14(LR)	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

异常类型	具体含义
复位	当处理器的复位电平有效时，产生复位异常，程序跳转到复位异常处理程序处执行。
未定义指令	当ARM处理器或协处理器遇到不能处理的指令时，产生未定义指令异常。可使用该异常机制进行软件仿真。
软件中断	该异常由执行SWI指令产生，可用于用户模式下的程序调用特权操作指令。可使用该异常机制实现系统功能调用。
指令预取中止	若处理器预取指令的地址不存在，或该地址不允许当前指令访问，存储器会向处理器发出中止信号，产生指令预取中止异常。
数据中止	若处理器数据访问指令的地址不存在，或该地址不允许当前指令访问时，产生数据中止异常。
IRQ（外部中断请求）	当处理器的外部中断请求引脚有效，且CPSR中的I位为0时，产生IRQ异常。系统的外设可通过该异常请求中断服务。
FIQ（快速中断请求）	当处理器的快速中断请求引脚有效，且CPSR中的F位为0时，产生FIQ异常。

异常优先级

从高到低:

(1) Reset (highest priority)

(2) Data abort

(3) FIQ

(4) IRQ

(5) Prefetch abort

(6) 未定义指令, Software interrupt

异常向量，操作模式

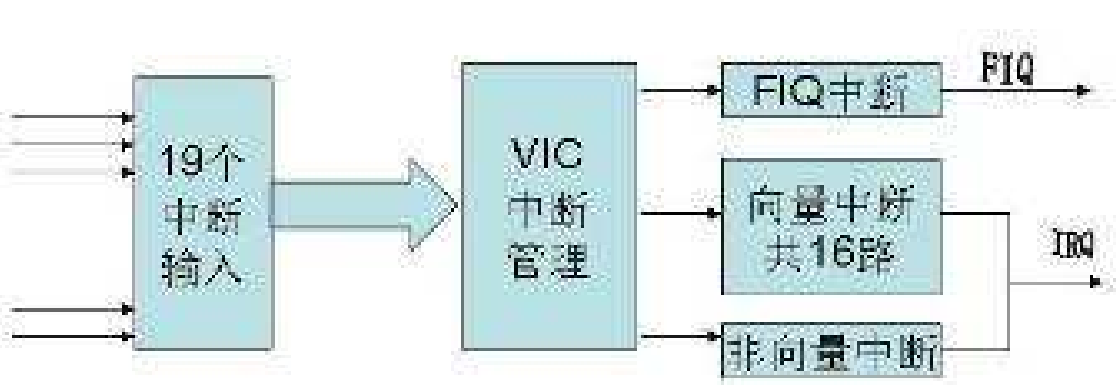
Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	*Reserved*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

每个中断只占据向量表中1个字的存储空间，只能放置一条转移指令。

FIQ位于异常向量表的最后位置，因此可紧接异常向量表书写FIQ的ISR，而不必进行程序跳转操作，避免了刷新指令流水线和高速缓存。

ARM的向量中断和非向量中断

- 向量中断：不同的中断有不同的ISR
  - 由硬件提供ISR地址，直接对PC赋值(此处通常是转移指令)
- 非向量中断：各中断源共享一个ISR
  - 由软件提供ISR地址
  - 在ISR中读INTR识别中断源，并判断优先级



对异常的响应-1

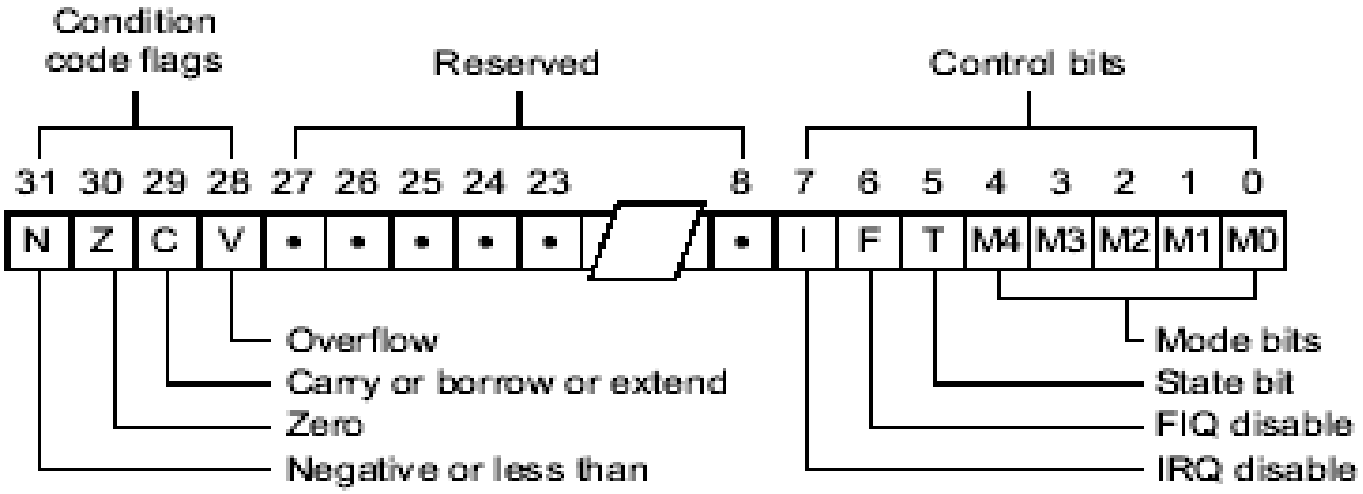
- 保存断点
  - 在相应的链接寄存器LR(r14)中保存下一条指令的地址，以便程序在处理异常返回时能从正确的位置重新开始执行
- 保存现场
  - 将CPSR复制到相应的SPSR中
- 设置工作模式
  - 根据异常类型，强制设置CPSR的运行模式位
- 转向异常处理程序
  - 强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

- 如果异常发生时处于Thumb状态，则当异常向量地址加载入PC时，处理器自动切换到ARM状态。

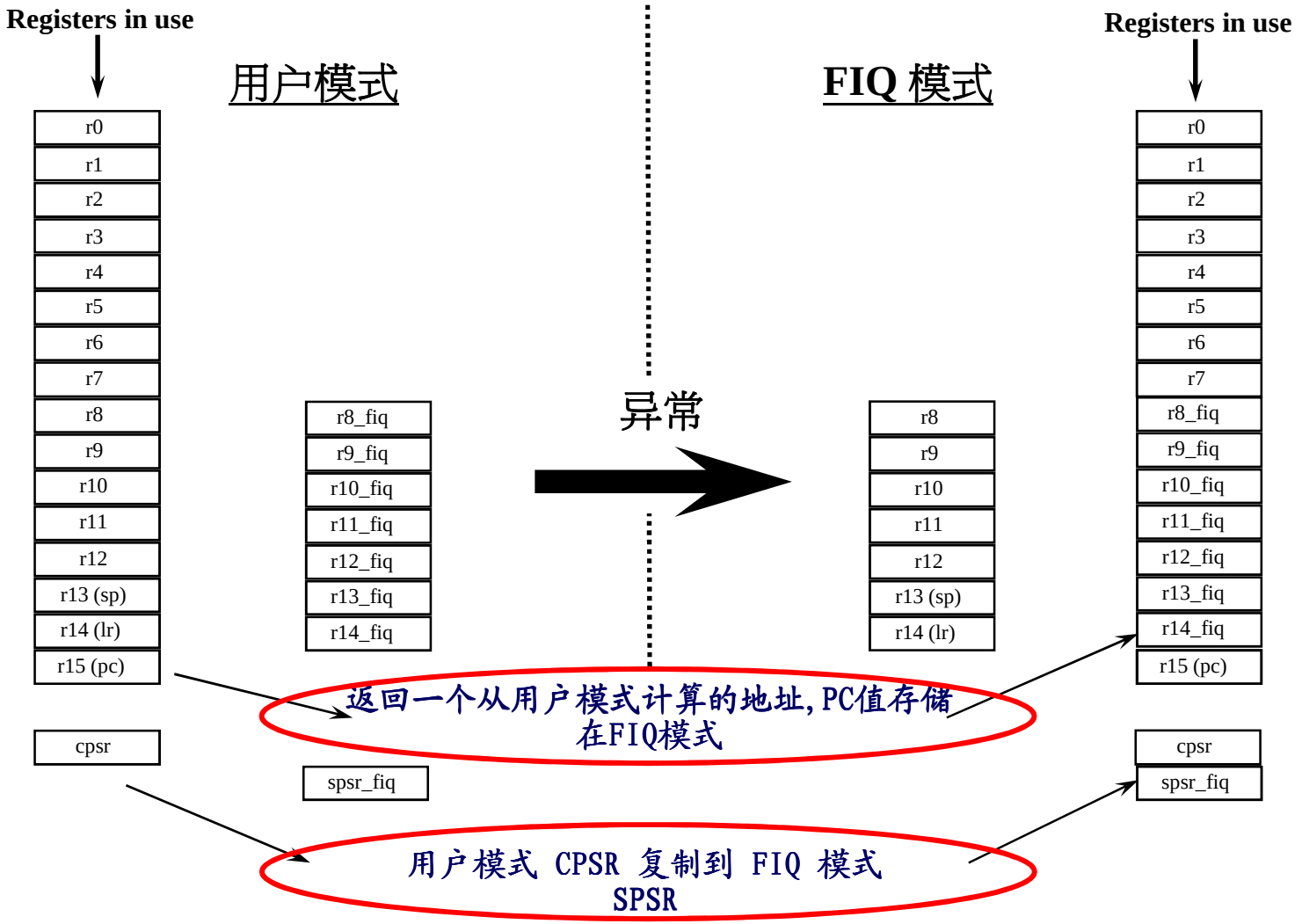
对异常的响应-2

ARM微处理器对异常的响应过程用伪码可以描述为：

```
1. R14_<Exception_Mode>=Return Link
2. SPSR_<Exception_Mode>=CPSR
3. CPSR[4:0]=Exception Mode Number
4. CPSR[5]=0                    ;当运行于Thumb状态时
5. CPSR[6]=1                    ;禁止新的IRQ中断
6. IF<Exception_Mode>==Reset or FIQ then
7.    CPSR[7]=1; 当响应FIQ异常时，禁止新的FIQ异常
8. PC=Exception Vector Address
```



例: 用户模式到 FIQ模式*



从异常的返回

- 将LR寄存器中的值减去相应的偏移量送到PC中
- 将 SPSR 复制回 CPSR
- 清除禁止中断标志，如果它被设置成使能
- *所有修改过的用户寄存器必须从处理程序的保护堆栈中恢复（即出栈）。*

- 可以认为程序总是从复位异常处理程序开始执行的，因此复位异常处理程序不需要返回。

异常返回的原子操作

- **CPSR和PC的恢复不能独立完成。**
  - 如果先恢复**CPSR**，则保存返回地址的当前异常模式的**r14**就不能再访问了；如果先恢复**PC**，异常处理程序将失去对指令流的控制，使得**CPSR**不能恢复。
  - **ARM**提供了两种返回处理机制，使上述两步作为一条指令的一部分同时完成。
    - 当返回地址保存在当前异常模式的**r14**时，使用其中一种机制；
    - 当返回地址保存在堆栈时使用另一种机制。

1、SUBS PC, R14_fiq, #4　　含'S'指令

2、指令示例：

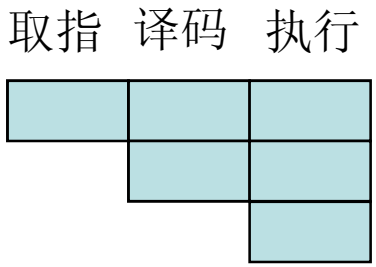
STMFD R13!，{R0，R4-R12，LR} ；将寄存器列表中的寄存器**R0，R4**到**R12，LR**存入堆栈。

LDMFD R13!,{R0，R4-R12，PC}^ ；将堆栈内容恢复到寄存器**R0，R4**到**R12，PC**，同时**SPSR**复制到**CPSR**

{^}为可选后缀，当指令为LDM且寄存器列表中包含R15，选用该后缀时表示：除了正常的数据传送之外，还将SPSR复制到CPSR。

返回断点

- SWI和UDEF
  - 在译码段产生(提前，减少转移开销)
    - 异常由当前指令产生，PC还未更新（＝PC+4）
  - 返回当前指令的下一条指令，直接返回PC
- FIQ、IRQ
  - 在执行段后响应
    - 指令已完成，PC已更新（＝PC+8）
  - 返回当前指令的下一条指令，应PC-4
- PABT
  - 在取指段产生
    - 指令未完成，PC未更新（＝PC+4）
  - 返回当前指令，应PC-4
- DABT
  - 在执行段产生
    - PC已更新（＝PC+8）
  - 返回当前指令，应PC-8



进入/退出异常概述

	Return Instruction	Previous State ARM R14_x	Previous State THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	-	-	4

注意：

1、在此PC应是具有预取中止的BL/SWI/未定义指令所取的地址。

2、在此PC是从FIQ或IRQ取得不能执行的指令的地址。

3、在此PC是产生数据中止的加载或存储指令的地址。

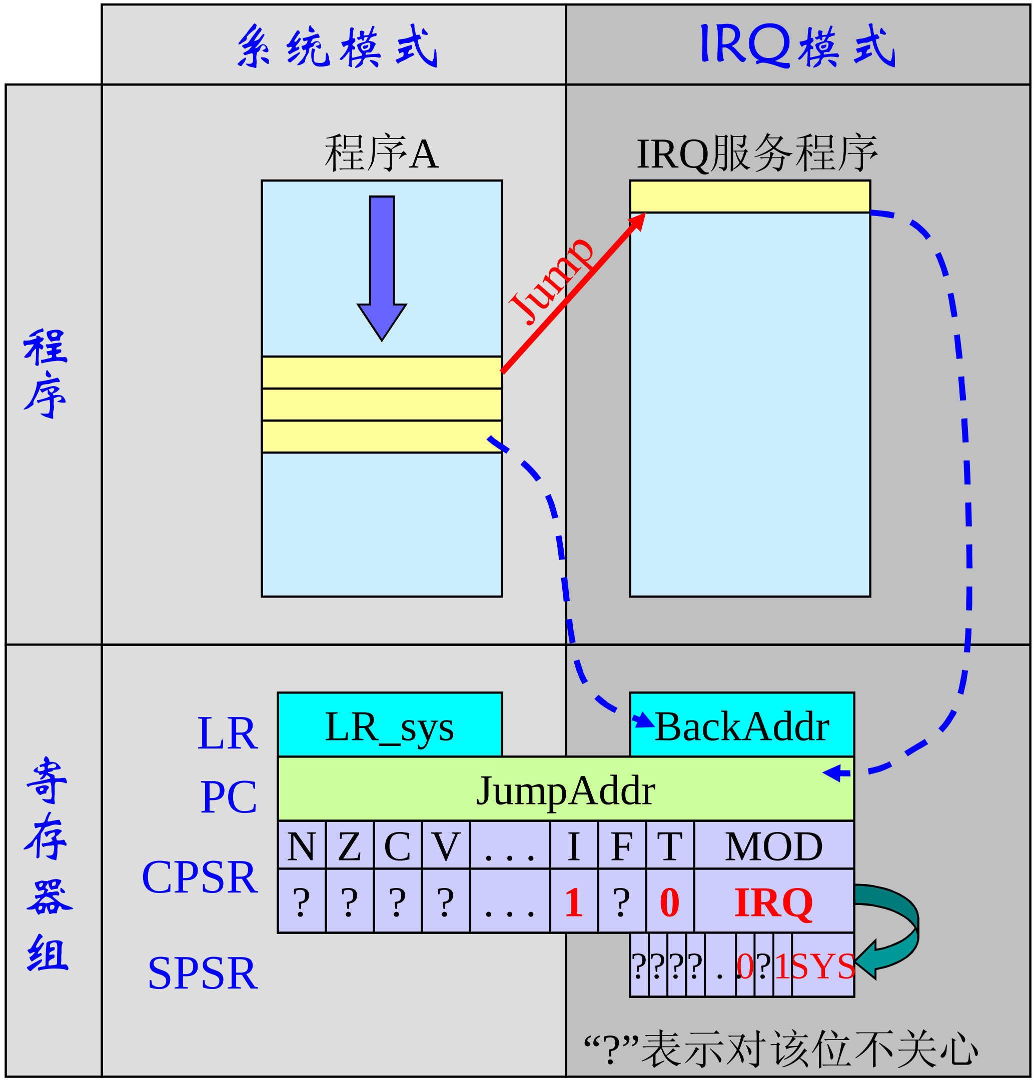
4、系统复位时，保存在**R14_svc**中的值是不可预知的。

图示：进入异常过程

2. 用户程序运行时发生IRQ中断，硬件完成以下动作：

程序A在系统模式下运行用户程序，假定当前处理器状态为Thumb状态，允许IRQ中断；

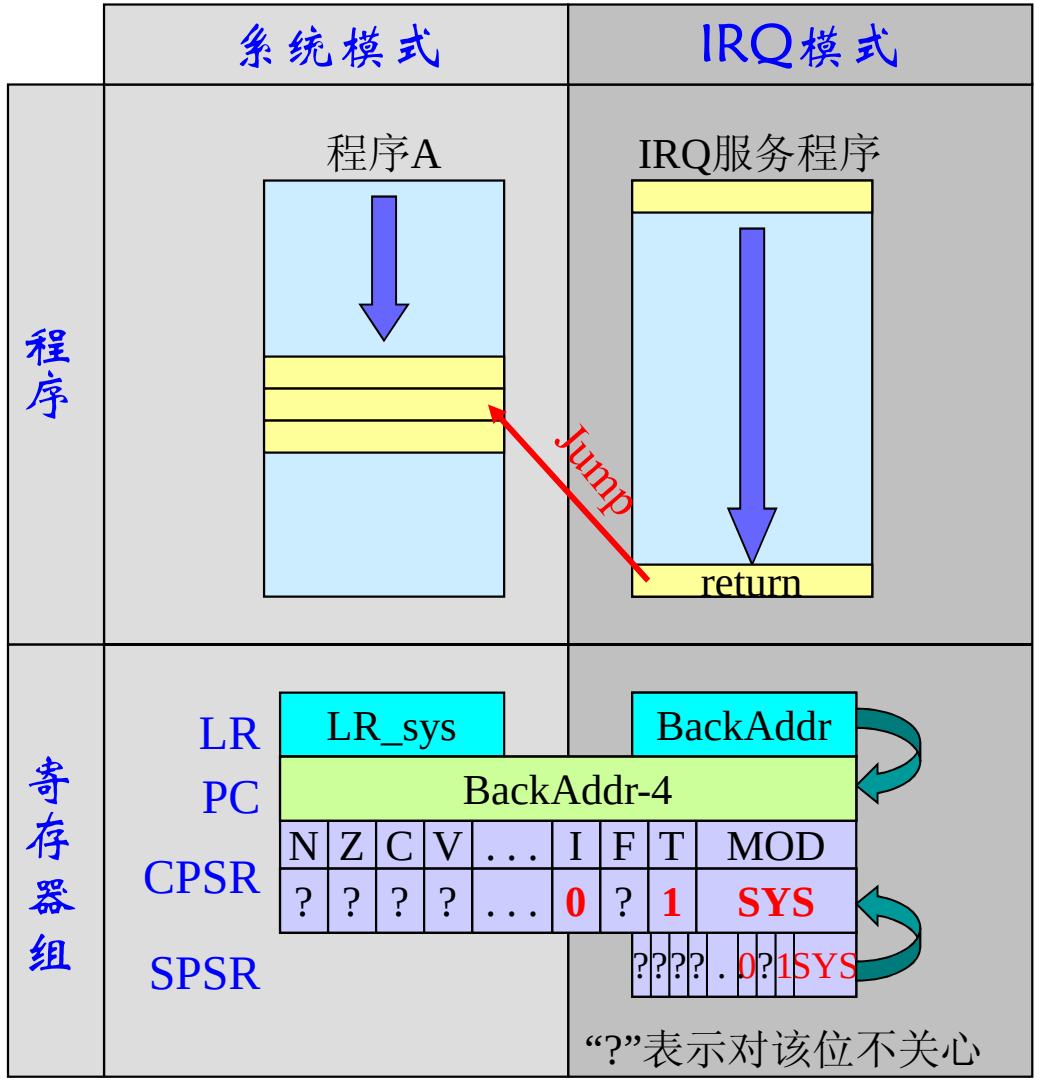
- 将CPSR的内容存入IRQ模式的SPSR寄存器
- 置位I位（禁止IRQ中断）
- 清零T位（进入ARM状态）
- 设置MOD位，切换处理器模式至IRQ模式
- 将下一条指令的地址存入IRQ模式的LR寄存器
- 将跳转地址存入PC，实现跳转



图示：退出异常过程

在异常处理结束后，异常处理程序完成以下动作：

- 将SPSR寄存器的值复制回CPSR寄存器；
- 将LR寄存的值减去一个常量后复制到PC寄存器，跳转到被中断的用户程序。



FIQ（Fast Interrupt Request）

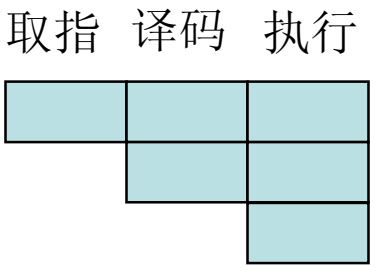
- **FIQ**异常是为了支持数据传输或者通道处理而设计的。
  - 可由外设通过对处理器上的**nFIQ**引脚输入低电平产生**FIQ**。处理器会在指令执行时检查**FIQ**的输入。
- **FIQ**优先级最高，但中断地址在向量表的最低位置，可直接接其**ISR**，避免转移，快速响应
- 不管是在**ARM**状态还是在**Thumb**状态下进入**FIQ**模式，**FIQ**处理程序均会执行以下指令从**FIQ**模式返回：
  - **SUBS PC,R14_fiq ,#4**
    - 该指令将寄存器**R14_fiq**的值减去**4**后，复制到程序计数器**PC**中，从而实现从异常处理程序中的返回，同时将**SPSR_mode**寄存器的内容复制到当前程序状态寄存器**CPSR**中。

IRQ中断处理

- When an IRQ is detected, ARM7 performs the following:
  - 1. Saves the address of the next instruction to be executed plus 4 in R14_irq; saves CPSR in SPSR_irq
  - 2. Forces M[4:0]=10010 (IRQ mode) and sets the I bit in the CPSR
  - 3. Forces the PC to fetch the next instruction from address 0x00000018
- To return normally from IRQ
  - use SUBS PC,R14_irq,#4, which will restore both the PC and the CPSR and resume execution of the interrupted code.

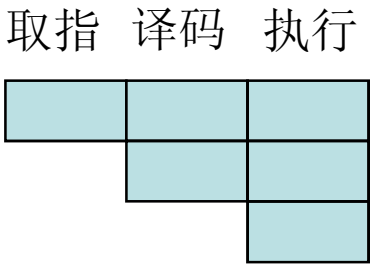
预取指中止

- 当发生预取中止时，ARM7TDMI内核将预取的指令标记为无效，但在指令到达流水线的执行阶段时才进入异常。
  - 如果指令在流水线中因为发生分支而没有被执行，中止将不会发生。（？）
- 在处理中止的原因之后，恢复PC和CPSR并重试被中止的指令



数据中止

- 当发生数据中止后，根据产生数据中止的指令类型作出不同的处理
  - LDR、STR：回写到被修改的基址寄存器（中止处理程序必须注意这一点）。
  - SWP：中止好像没有被执行过一样（中止必须发生在SWP指令进行读访问时）。
- 在修复产生中止的原因后，重试被中止的指令



软件中断指令

- 使用软件中断(SWI)指令可以进入管理模式，通常用于请求一个特定的管理函数。
- SWI处理程序读取操作码，以提取SWI函数编号。
- SWI处理程序通过执行下面的指令返回：

  MOVS　PC, R14_svc

  – 这个动作恢复了PC，并返回到SWI之后的指令。

未定义的指令

- 当ARM7TDMI遇到一条自己和系统内任何协处理器都无法处理的指令时，内核执行未定义指令陷阱。
  - 软件可使用这一机制通过模拟未定义的协处理器指令来扩展ARM指令集。
- 在模拟处理了失败的指令后，陷阱程序执行下面的指令返回：

```
MOVS    PC, R14_svc
```

  - 这个动作恢复了PC，并返回到未定义指令之后的指令。

复位

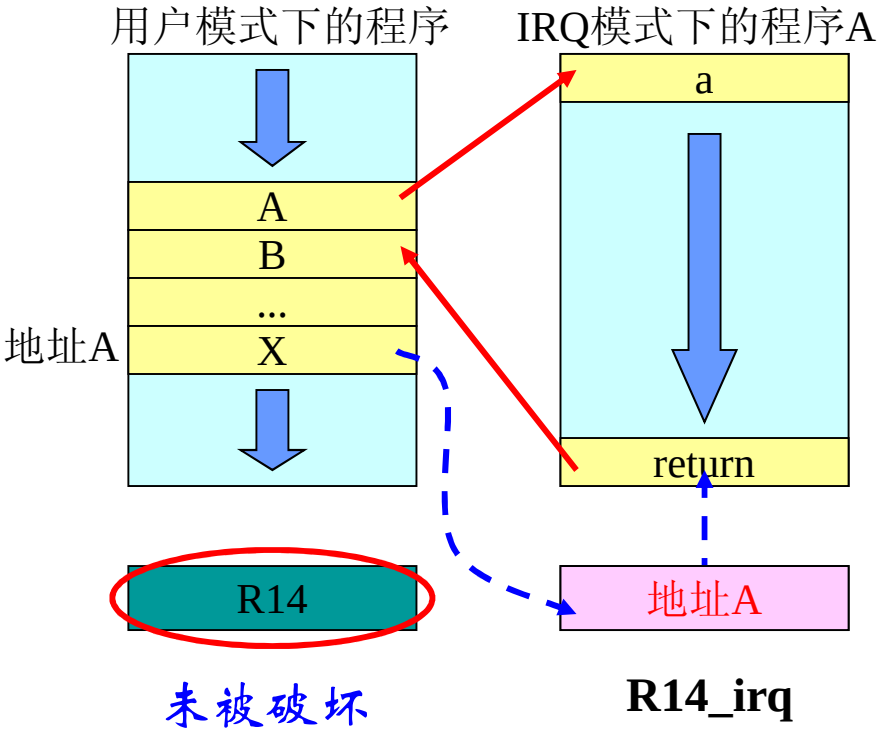
- 当nRESET信号被拉低时（一般外部复位引脚电平的变化和芯片的其它复位源会改变这个内核信号），ARM7TDMI处理器放弃正在执行的指令。
  - 在复位后，除PC和CPSR之外的所有寄存器的值都不确定。
- 当nRESET信号再次变为高电平时，ARM处理器执行下列操作：
  1. 强制CPSR中的M[4:0]变为b10011（管理模式）；
  2. 置位CPSR中的I和F位；
  3. 清零CPSR中的T位；
  4. 强制PC从地址0x00开始对下一条指令进行取指；
  5. 返回到ARM状态并恢复执行。

异常嵌套

- ARM进入异常时硬件禁止新的中断
  - 防止不受控制的异常嵌套
- R14寄存器使用
  - 当发生异常嵌套时，这些异常之间可能会发生R14寄存器冲突。
  - 例如：如果用户在用户模式下执行程序时发生了IRQ中断，用户模式寄存器不会被破坏。但是如果允许在IRQ模式下的中断处理程序重新使能IRQ中断，并且发生了嵌套的IRQ中断时，外部中断处理程序保存在R14_irq中的任何值都将被嵌套中断的返回地址所覆盖。

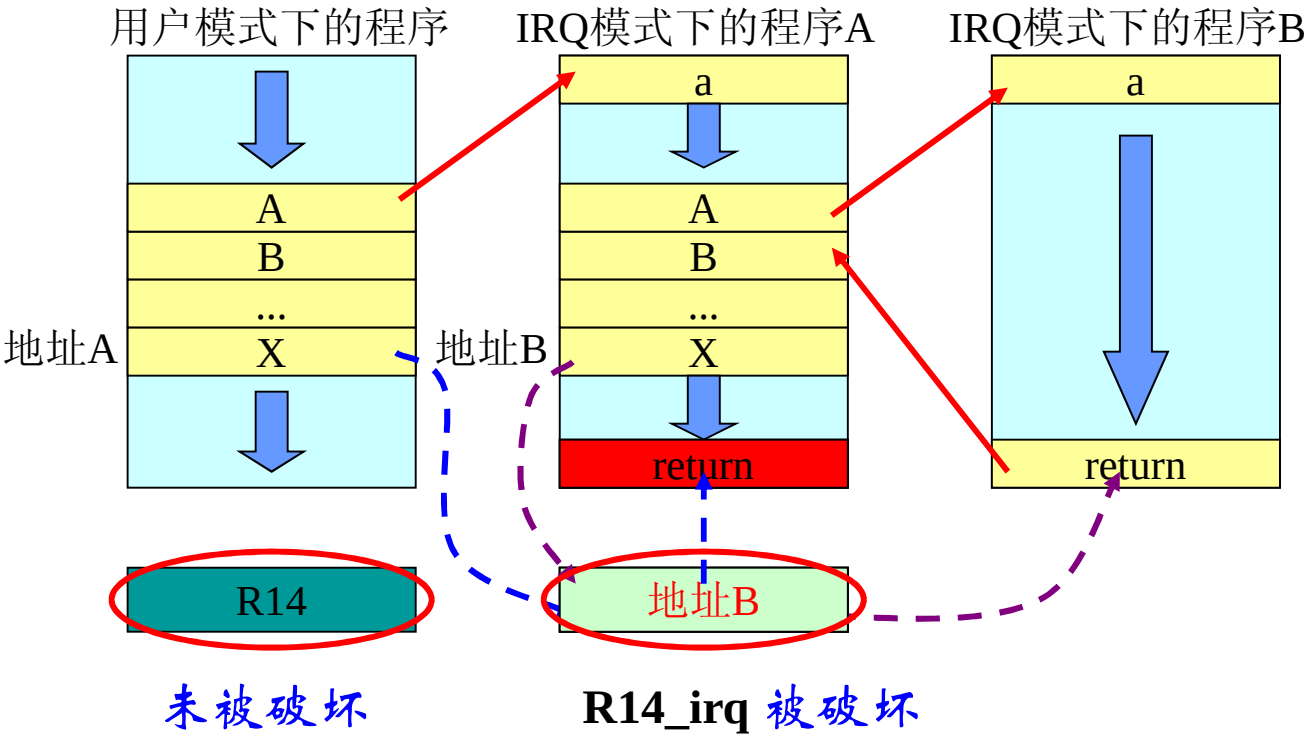
- 未嵌套时

3. IRQ服务程序A执行完毕，将R14_irq寄存器的内容减去某个常量后存入PC，返回之前被中断的程序；



- 允许嵌套时

6. 在程序B返回到程序A，然后在返回到用户模式下被中断的程序时，发生错误，将不能正确返回；



解决办法是确保R14的对应版本在发生中断嵌套时不再保存任何有意义的值（将R14入栈），或者切换到其它处理器模式下。

作业

- 影响流水线处理器性能发挥的主要因素有哪些？
- 分析ARM处理器不同工作模式的作用，举例说明切换过程。
- 分析ARM处理器的中断响应时间。


Thank You

控制位

CPSR的低**8**位（包括**I**、**F**、**T**和**M[4**：**0]**）称为控制位，当发生异常时这些位可以被改变。如果处理器运行特权模式，这些位也可以由程序修改。

— 中断禁止位**I**、**F**

I=1 禁止**IRQ**中断；**F=1** 禁止**FIQ**中断。

— **T**标志位：该位反映处理器的运行状态。

对于**ARM**体系结构**v5**及以上的版本：

T系列处理器，当该位为**1**时，程序运行于**Thumb**状态，否则运行于**ARM**状态。

非**T**系列处理器，当该位为**1**时，执行下一条指令以引起未定义指令异常；当该位为**0**时，表示运行于**ARM**状态。

保留位

CPSR中的其余位为保留位，保留位将用于ARM版本的扩展。