北京邮电大学
Beijing University of Posts and Telecommunications

嵌入式系统

ARM基础

邝 坚

嵌入式系统与网络通信研究中心

北京邮电大学 计算机学院/软件学院

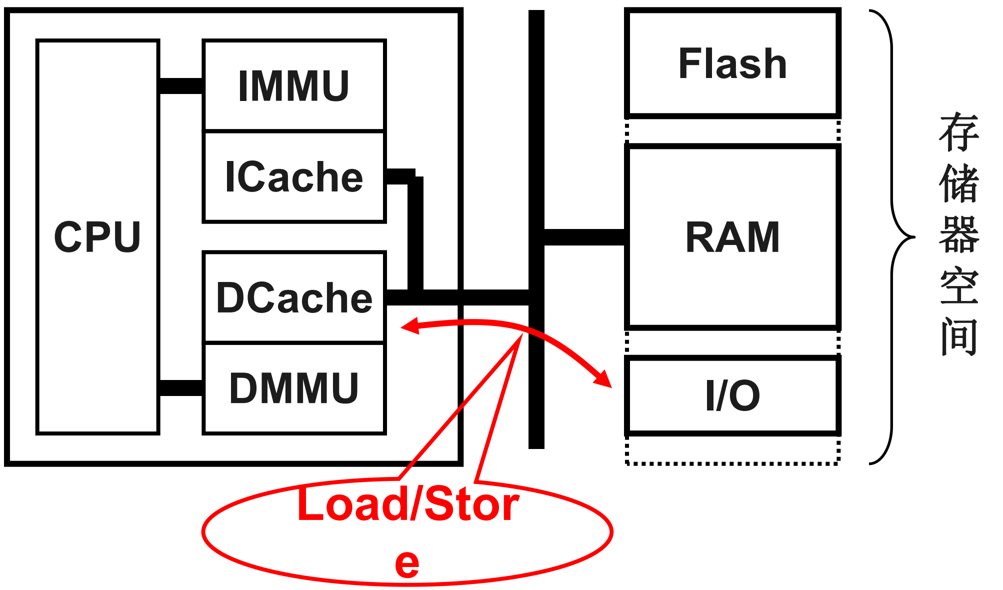
RISC Architecture Review

典型的RISC结构特征

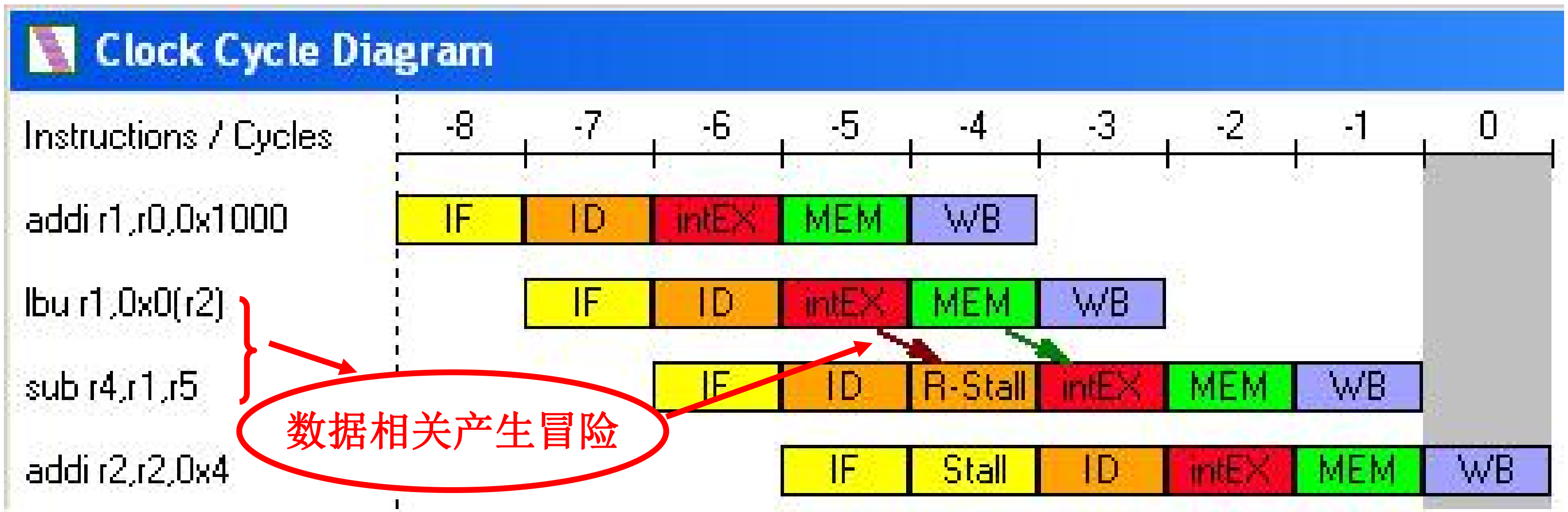
- 目前绝大多数嵌入式处理器是**RISC**系统结构
  - 单一指令长度
  - 较少的寻址方式（一般≤**5**）
  - **LOAD/STORE**指令不会与算术运算混在一起
  - 每条指令最多有一个存储器操作数**(LOAD/STORE)**
  - 整数寄存器有**32**个以上
  - 浮点寄存器（如果有）有**16**个以上

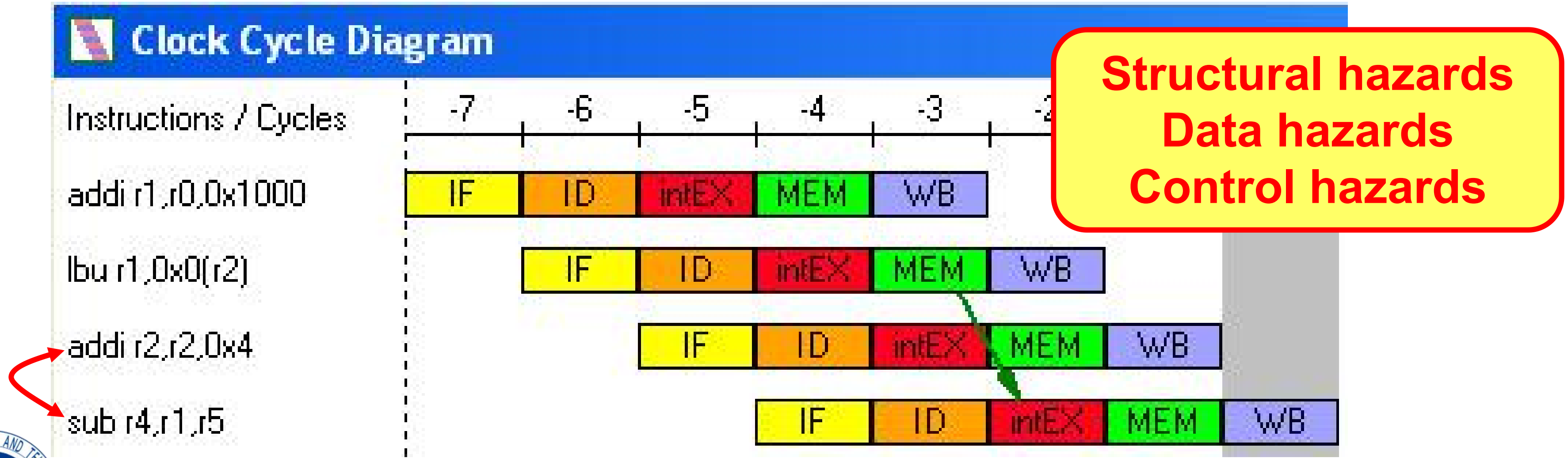
设计原则-01

- Cache一致性保证(1)
  - RISC处理器往往没有I/O地址空间，因此I/O设备会占用一部分存储器地址空间。由于CPU访问存储器与访问I/O的方式相同(Load/Store) – **I/O地址区域永远不要设置Cache Enable！**



没有流水就没有RISC，但是…

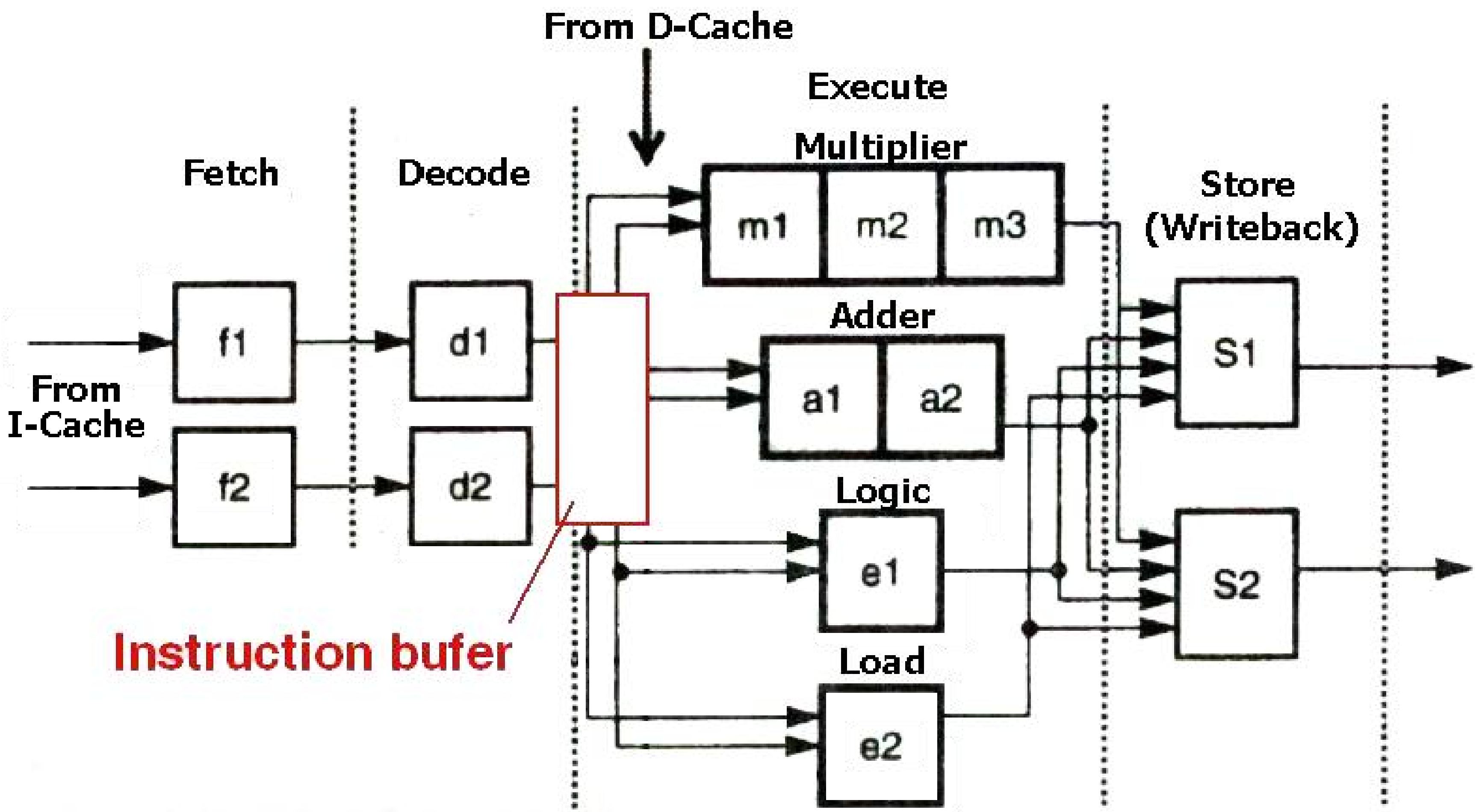




设计原则-02

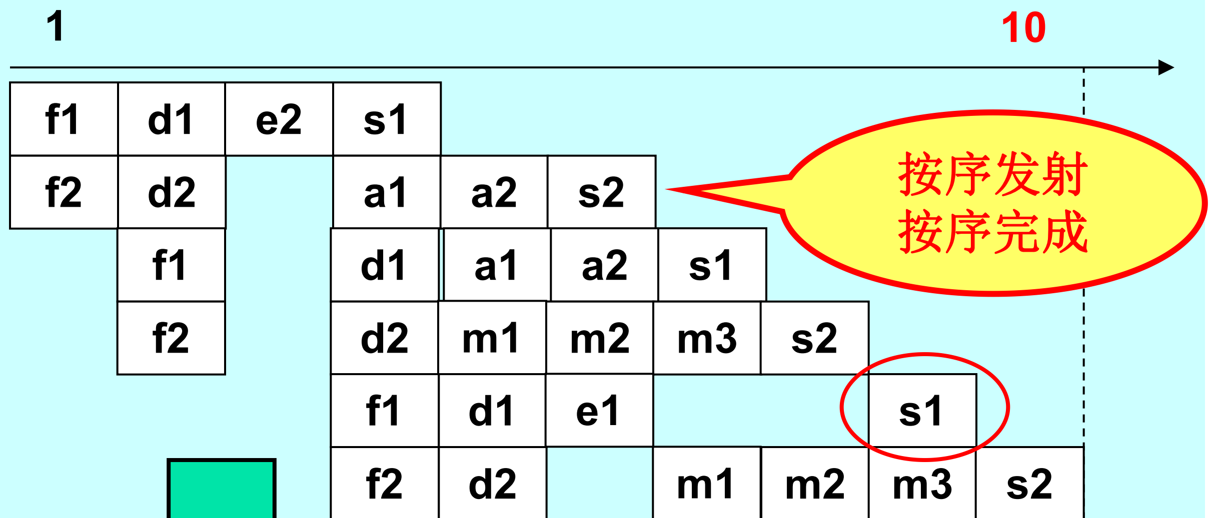
- 代码运行高效的原因之一是尽可能避免“冒险”，其前提，是**了解处理器的系统结构**
  - 指令之间的数据相关性是“数据冒险”的原因，**降低指令之间的数据相关性，将有效减少“数据冒险”产生的几率**；
  - 相对而言，因条件转移指令产生的“控制冒险”对代码运行的效率影响更大。**了解目标处理器针对“控制冒险”的预测机制，并依建议实施，将能保证代码高效的执行**。

Superscalar

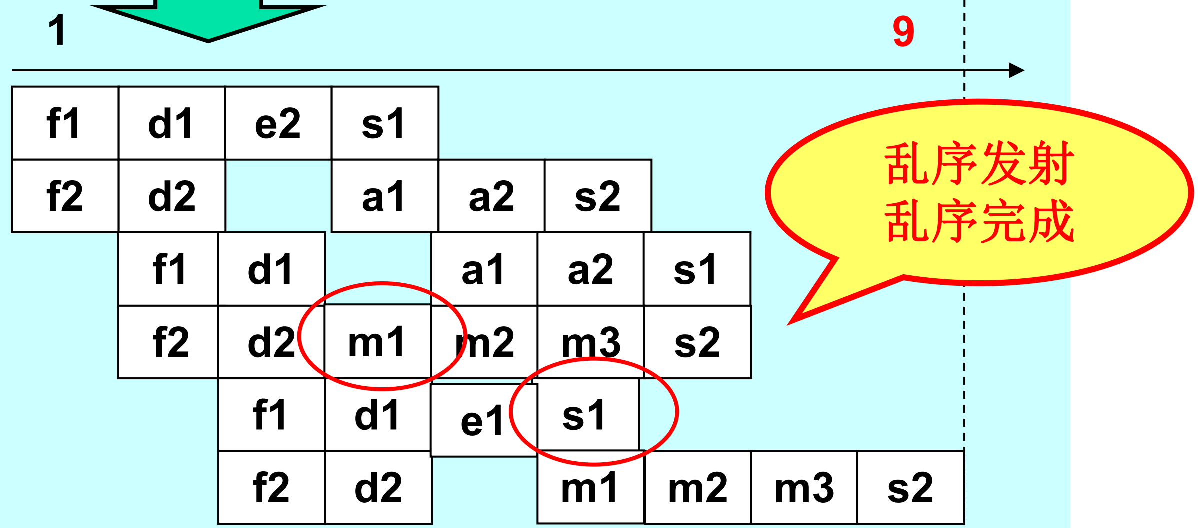


乱序(Out of Order)的确能提高代码运行效率

```
LOAD   R1,M(A)
ADD    R2,R2,R1
ADD    R3,R3,R4
MUL    R4,R4,R5
NEG    R6,R6
MUL    R6,R6,R7
```




```
LOAD   R1,M(A)
ADD    R2,R2,R1
ADD    R3,R3,R4
MUL    R4,R4,R5
NEG    R6,R6
MUL    R6,R6,R7
```



设计原则-03

- **要防止CPU对I/O区域访问时的“乱序(Out of order)”操作**
  - 对I/O的读写操作(Load/Store)往往有严格此序要求。例如：先读(Load)状态寄存器，判断其内容是否有效后，再写入(Store)数据寄存器。
  - <u>“乱序”有可能打乱多次读写(Load/Store)的次序，造成对I/O的误操作或操作失败</u>。
  - 允许“乱序”的处理器都有相应的防止“乱序”发生的机制。

How about ARM

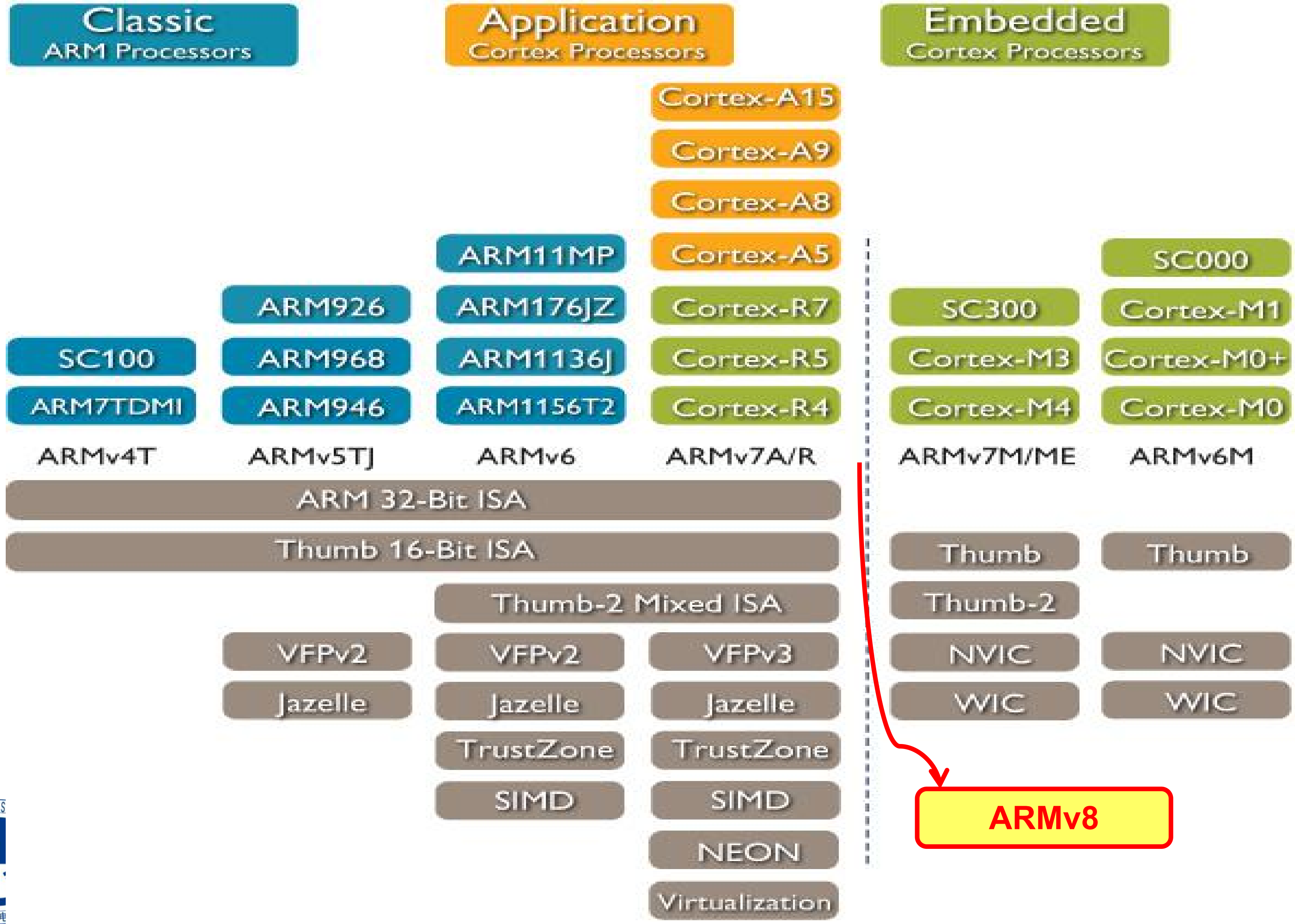
ARM® – Advanced RISC Machines

- Founded 27th Nov 1990 (Acorn、Apple、VLSI), in a barn.
- The world's leading semiconductor IP company.
- Data:
  - Originally 12 employees, now >2,000.
  - Over 20 billion ARM technology based chips shipped to date.
  - About 750 processor licenses sold to more than 250 companies.
  - Millions of developers; billions of users.
  - From 8 Partners to 400 Partners at 2010.

ARM Technology

- **From processor and multimedia IP to software**
  - **Processor IP – Design of the brain of the chip**
  - **Physical IP – Design of the building blocks of the chip**
  - **Software development tools**

ARM Architecture Overview



ARM Architecture Version

- **ARMv4**
  can be considered a 32-bit ISA operating in a 32-bit address space. Implementations include some members of the ARM7™ core family and Intel StrongARM® processors.
- **ARMv4T**
  Added the 16-bit Thumb instruction set while retaining all the benefits of a 32-bit system - 35% codesize saving - ARM7TDMI®, ARM9TDMI, **ARM920T**, ARM940T.

ARM Architecture Versions

- **ARMv5TE**
  Introduced improvements to the Thumb architecture, along with ARM 'Enhanced' DSP instruction set extensions.
- **ARMv5TEJ**
  Added the Jazelle extension to support Java acceleration technology – 8x & 80% reduction in power consumption - ARM926EJ-S™ and ARM968E-S™.

ARM Architecture Versions

- **ARMv6**
  Introduced an array of new features including the Single Instruction Multiple Data (SIMD) operations, where the extensions increase performance by up to four times. In addition Thumb-2 and TrustZone technologies were introduced as variants of the ARMv6 - ARM1176JZ and ARM1136EJ

ARM Architecture Versions

- **ARMv6M**
  Designed for low-cost, high-performance devices providing a 32-bit powerful solution in a marketplace previously dominated by 8-bit devices. 16-bit Thumb only - Cortex-M0/M0+ and Cortex-M1

ARM Architecture Versions

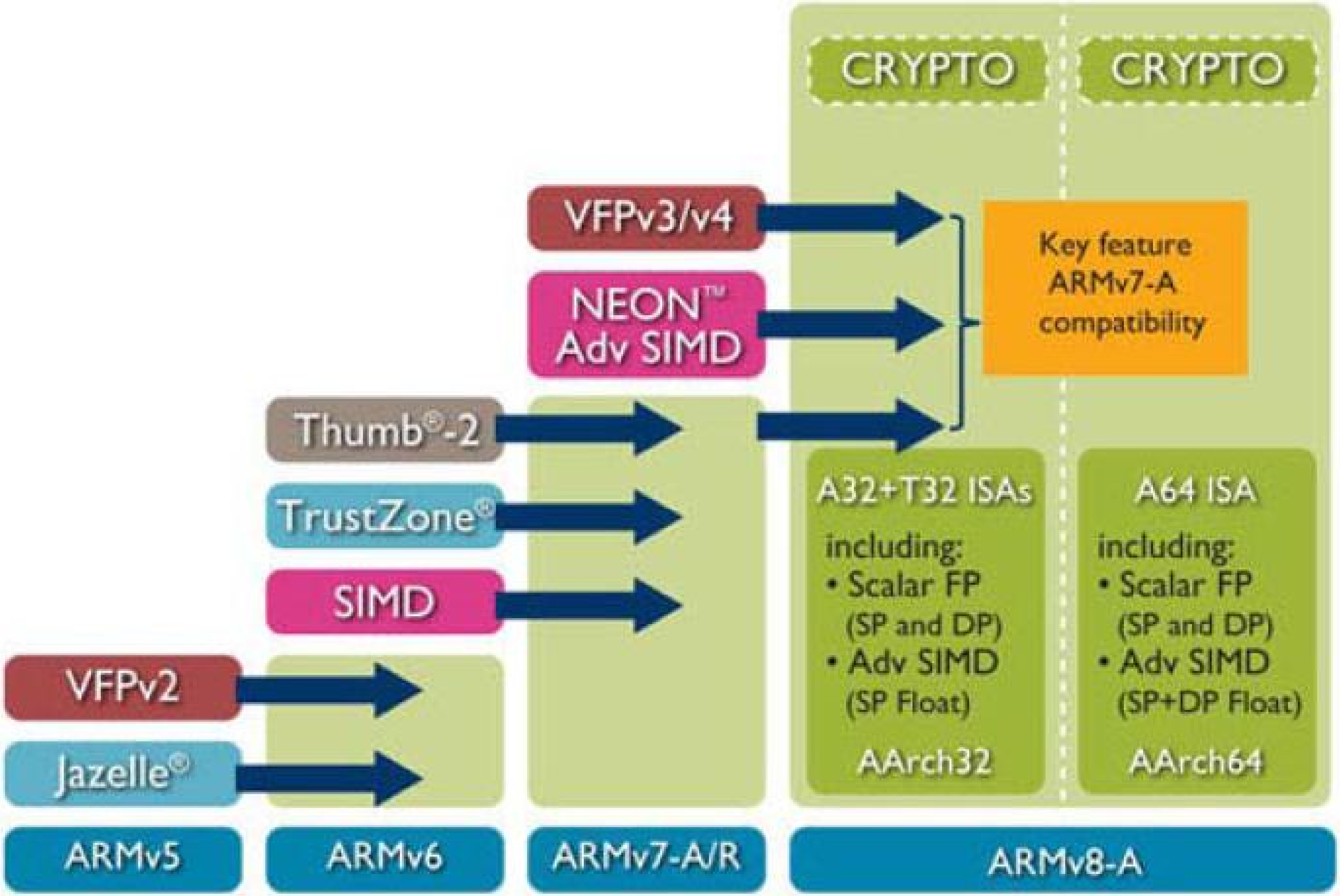
- **ARMv7**
  All Cortex processors except Cortex-M0/1. 3 profiles:
  - Cortex-A (Applications) - Cortex-A17/A15/A12/A9/A8/A5
    - Virtual addressing (MMU) for running Windows, Linux, etc
    - Cache for high performance
    - Often heavy focus on 3rd party applications requiring large memory
  - Cortex-R (Real-time and Control) - Cortex-R7/R5/R4
    - No virtual address capability, typically run RTOS
    - Microarchitecture focus on on fast response to interrupts
    - ECC cache options, lock-stop, error tolerance
  - Cortex-M (Microcontroller) - Cortex-M3
  - Cortex-EM (Microcontroller) - Cortex-M7/M4
    - Extremely small gate count and low power
    - Fast and deterministic exception handing
    - No cache, no virtual address capability

ARM Architecture Versions

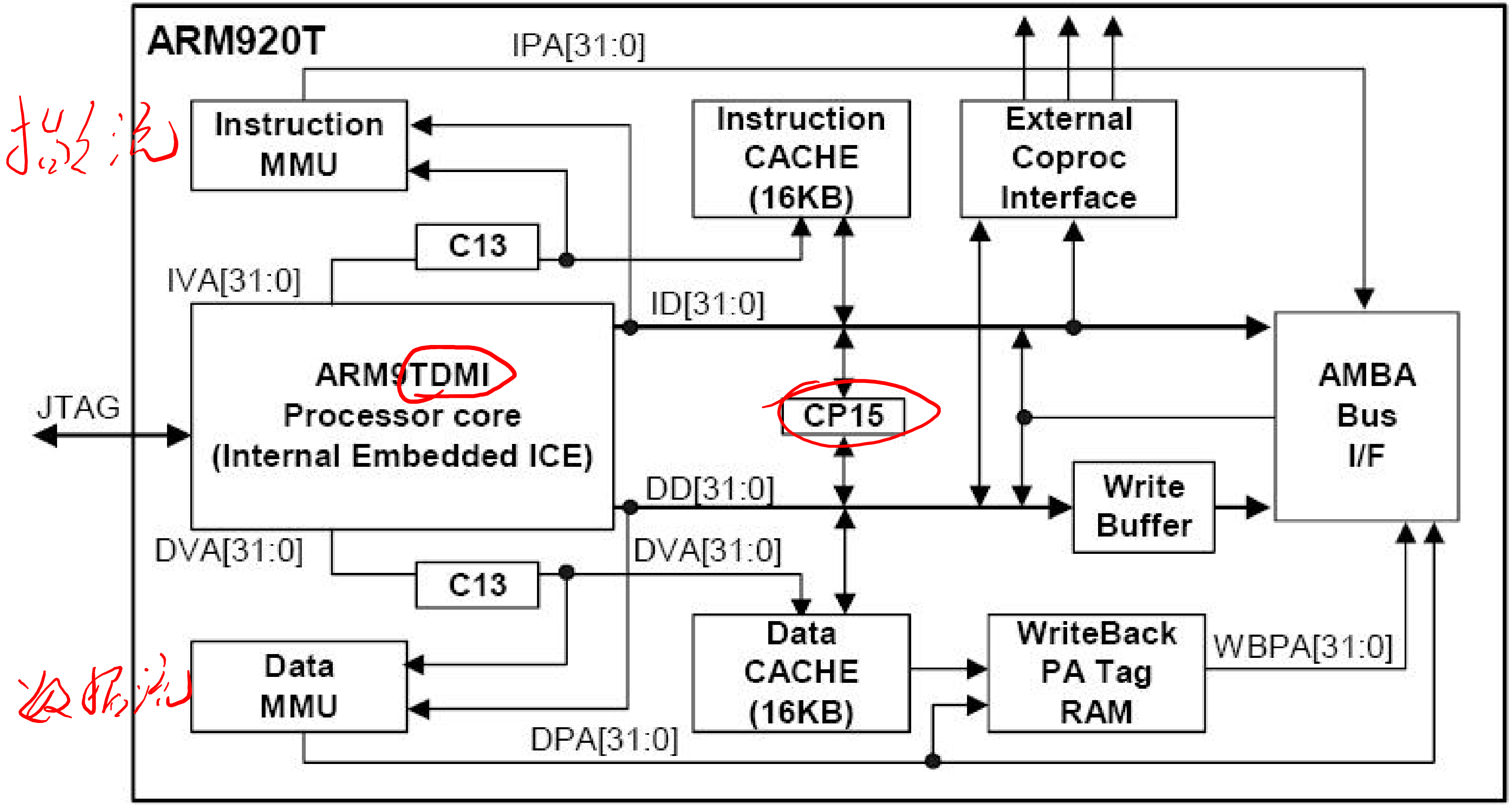
- **ARMv8**
  It adds a 64-bit architecture, named "AArch64", and a new "A64" instruction set. AArch64 provides user-space compatibility with ARMv7-A ISA, the 32-bit architecture, therein referred to as "AArch32" and the old 32-bit instruction set, now named "A32".
- The Thumb instruction sets are referred to as "T32" and have no 64-bit counterpart.
- ARMv8-A allows 32-bit applications to be executed in a 64-bit OS, and a 32-bit OS to be under the control of a 64-bit hypervisor.

ARMv8-A	64/32	ARM Cortex-A53, ARM Cortex-A57, ARM Cortex-A72
ARMv8.1-A	64/32	NA
ARMv8-R	32	NA

ARM Architecture Versions



ARM Core Sample – ARM920T



T: Thumb **D**: 在片调试(Debug)功能，允许处理器响应调试请求暂停；
M: 增强型乘法器，产生全64位结果； **I**: 嵌入式ICE硬件，提供片上断点和调试点支持。

ARM920T Core

- 协处理器(Coprocessor)
  - CP15 – 系统控制功能。内部寄存器(16个)用于配置和控制cache、MMU、时钟模式等
  - CP14 – Debug控制器
  - CP13 - CP0 – 可通过“external coprocessor interface”挂接
- R13
  - CP15中的一个寄存器，用于协助Core与Cache/MMU之间进行地址转换

虚拟地址 → 实地址 → cache 地址

MMU cache 映射

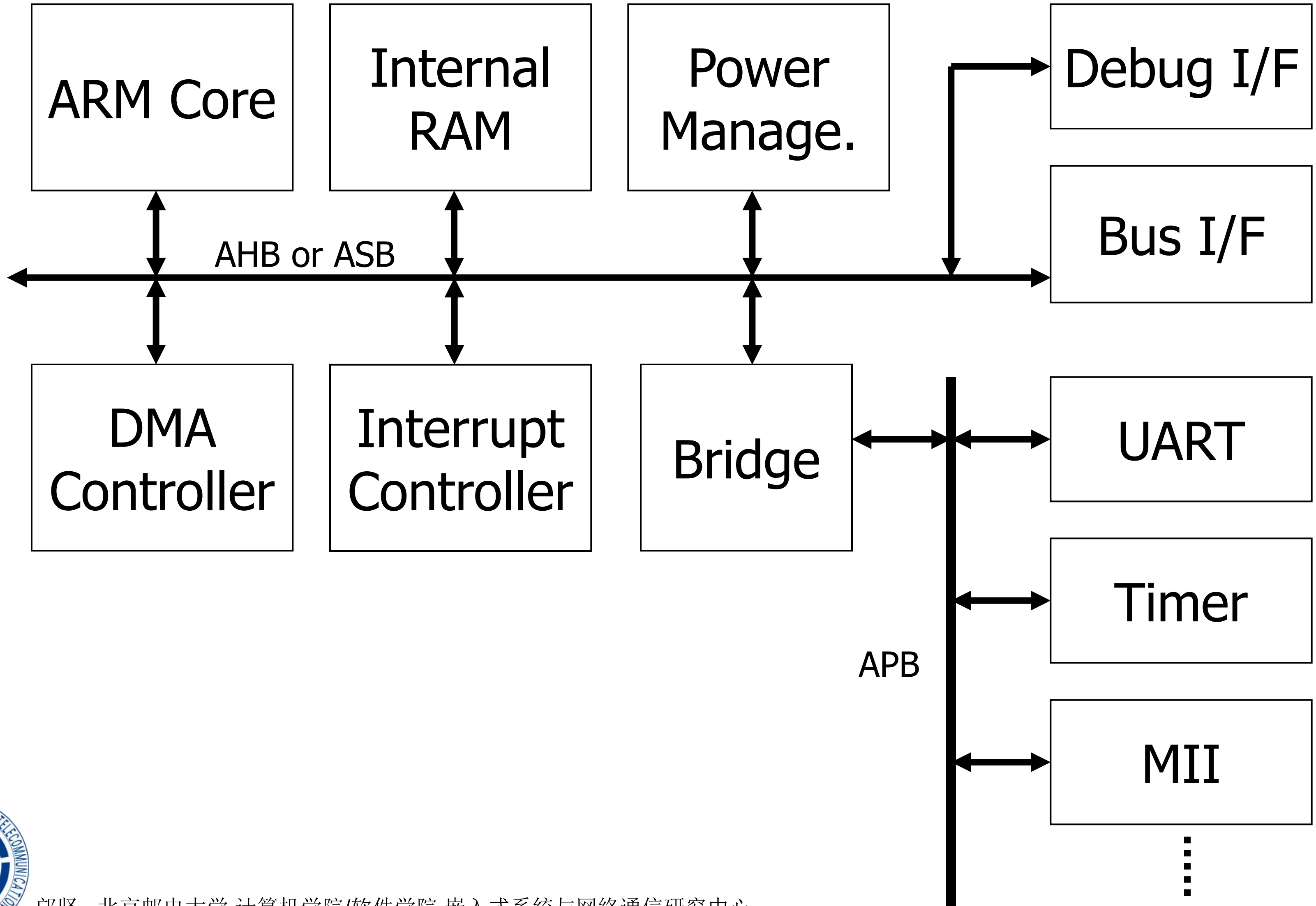
ARM920T Core

- Cache
  - 虚地址Cache，64路相联存储
  - 16KB ICache，16KB DCache
  - 512行，每行8个word(32bit)
  - 可锁定64个word
  - Pseudo-random或round-robin替换算法
- PA TAG RAM
  - 保存被写回的DCache数据的物理地址
- Write Back Buffer
  - 被写回的数据缓冲。指令可以不用等待写入RAM动作完成
- MMU
  - 支持16个地址域(domain)
  - ITLB/DTLB各16个表项

ARM AMBA bus

- AMBA – Advanced Microcontroller Bus Architecture。公开标准，用于连接和管理SoC系统的不同功能模块
  - Advanced eXtensible Interface (AXI)
  - Advanced High-performance Bus (AHB)
  - Advanced System Bus (ASB)
  - Advanced Peripheral Bus (APB)
  - Advanced Trace Bus (ATB).

AMBA Sample



ARM Core
Internal RAM
Power Manage.
Debug I/F
Bus I/F
AHB or ASB
DMA Controller
Interrupt Controller
Bridge
UART
Timer
APB
MII

ARM Pipelining

ARM7TDMI 3段

取指	Thumb 解压	译码	Reg read	Shift ALU	Reg Write

ARM9TDMI 5段

取指	Reg R 译码	Shift/ALU	MEM	Reg W 写回

ARM10TDMI 6段

转移预测			Add Cal	MEM	data W 写回
取指	发射	Reg R 译码	Shift ALU/乘法	乘法器 部分积加	Reg W 写回

Endian Mode

符合阅读习惯

- Big-Endian（大端模式）

最高位 MSB　　　低位 最低位 LSB

0x12345678

12	34	56	78

0x1234

12	34		

低地址 → 高地址

- Little-Endian（小端模式）

MSB　　　LSB

0x12345678

78	56	34	12

0x1234

34	12		

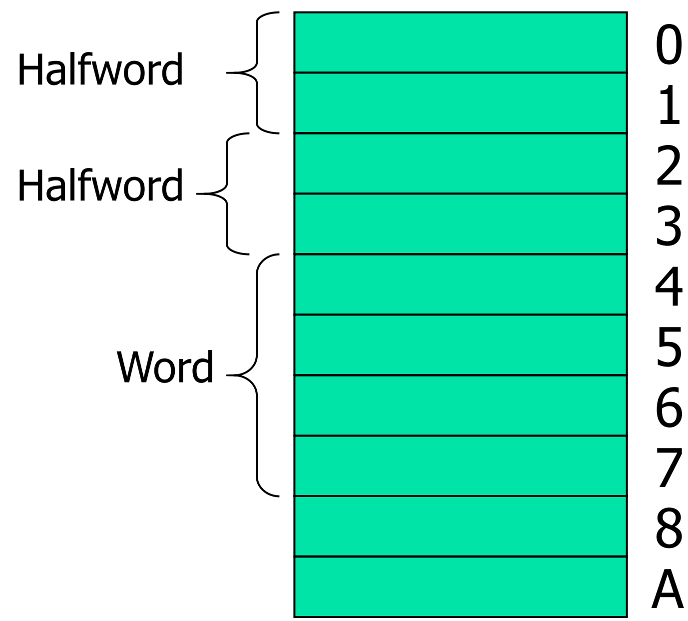
低地址　　　高地址

提示：

处理器的大小端模式可能会影响某些代码的编写方式，例如**TCP/IP**网络协议栈，因为**Ethernet**以大端模式传输数据。

Data & Instruction alignment

- 数据类型
  - Byte：8位
  - Halfword：16位（2字节边界对准）
  - Word：32位（4字节边界对准）
- 指令
  - ARM 32位指令
    - 4字节边界对准
  - Thumb 16位指令
    - 2字节边界对准



设计原则-04

- RISC处理器往往有数据对准要求，**养成好的结构数据定义(对准)习惯很有必要**。即使处理器没有数据对准要求，如X86，一个好的结构数据定义也会提高代码执行效率。

偶地址访问一个字快

```
typedef struct
{
  char   c;
  short  s;
  int    i;
}sample_srtuc
```

→

```
typedef struct
{
  char   c, recv;
  short  s;
  int    i;
}sample_srtuc
```

∵偶→D:低8位
奇→D:高8位
访问偶地址，
一个周期可同
取一字

Operation Mode

特权模式

异常模式

- User (usr): ARM正常程序执行模式
- System (sys): 操作系统的特权用户模式
- FIQ (fiq): 响应快速中断时的处理模式
- IRQ (irq): 响应普通中断时的处理模式
- Supervisor (svc): 操作系统保护模式 ← 软件通过系统调用 访问操作系统
- Abort mode (abt): 虚拟存储或存储器保护 — 存取数据出现问题
- Undefined (und): 硬件协处理器的软件仿真 不支持 非法访问

- 特权模式(Privileged Mode)。程序可以访问所有的系统资源，也可以通过软件控制进行处理器模式的切换。
- 用户模式下不能软件进行模式改变，也不能访问被保护的系统资源。
- 外部中断或异常可进入对应模式

ARM State & GPRs

Thumb

不分组寄存器

分组寄存器

SP堆栈寄存器

LR链接寄存器

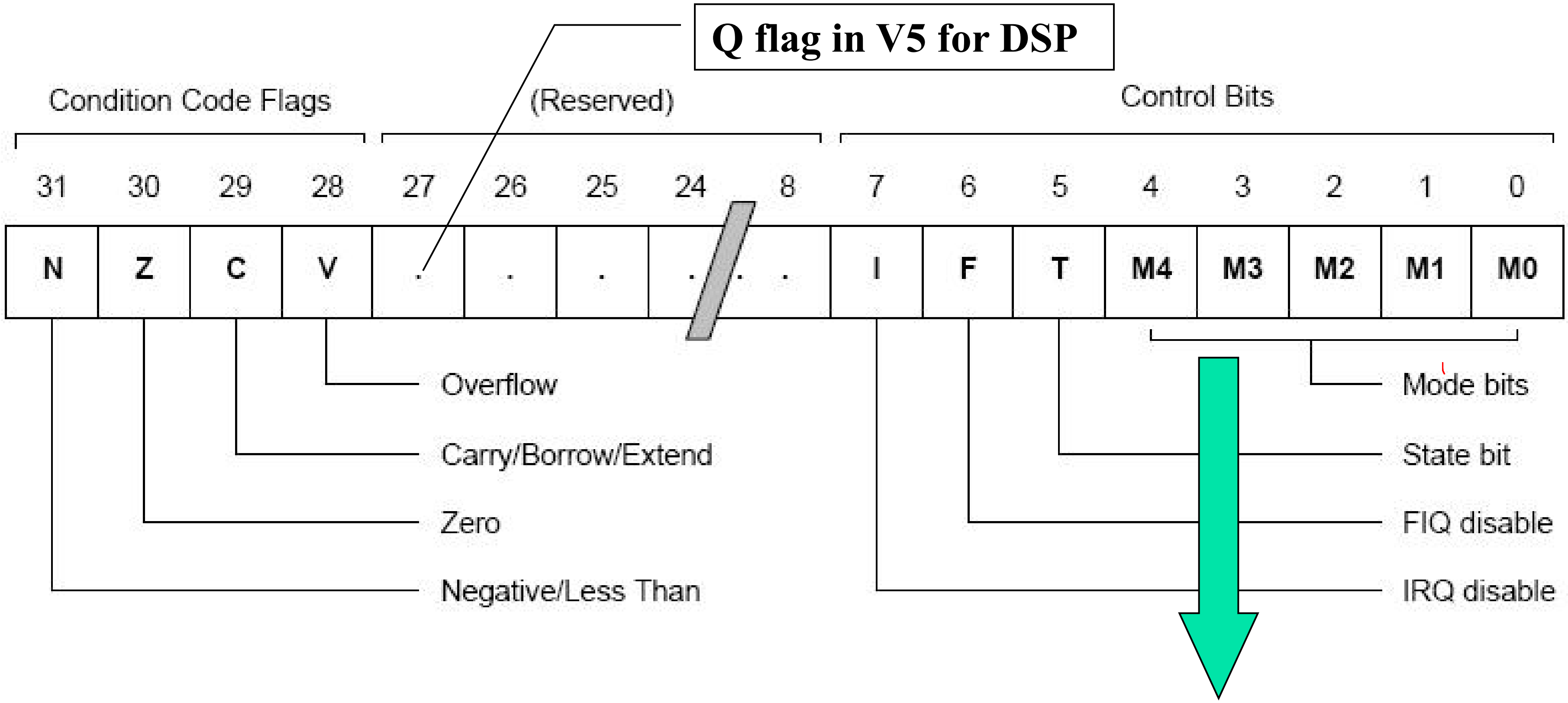
System & User	FIQ	Supervisor	About	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

◣ : Banked寄存器

ARM State Program Status Register

System & User	FIQ	Supervisor	About	IRQ	Undefined
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

CPSR/SPSR(Page: A2-9 ~ 12)



10000	10001	10010	10011	10111	11011	11111
User	FIQ	IRQ	Super	Abort	Undef	Sys

Exception

异常服务程序入口

类型	模式	低端地址	*高端地址
复位	Super	0x00000000	0xFFFF0000
未定义指令	Undef	0x00000004	0xFFFF0004
软件中断	Super	0x00000008	0xFFFF0008
预取指令终止	Abort	0x0000000C	0xFFFF000C
数据终止	Abort	0x00000010	0xFFFF0010
(保留)	(Resv)	0x00000014	0xFFFF0014
IRQ	IRQ	0x00000018	0xFFFF0018
FIQ	FIQ	0x0000001C	0xFFFF001C

Exception

- 异常处理
  - 出现时：PC→R14、CPSR→SPSR、中断被禁止
  - 返回时：SPSR→CPSR、R14→PC、中断允许，如果之前被禁止
- 异常优先级

1（高）	复位
2	数据终止
3	FIO
4	IRQ
5	预取终止
6（低）	未定义/软中断

Exception

- **Reset**：复位信号输入。
- **Undefined instructions**：执行未定义指令、协处理器未应答。
- **Software interrupt**：SWI指令。
- **Prefetch abort**：取指令存储器终止。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，存储器发出存储器终止(Abort)信号；BKPT指令(V5)。
- **Data about**：数据访问存储器终止。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，存储器发出存储器终止(Abort)信号。

Exception Return

	返回指令	以前的状态		注意
		ARM R14_x	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。

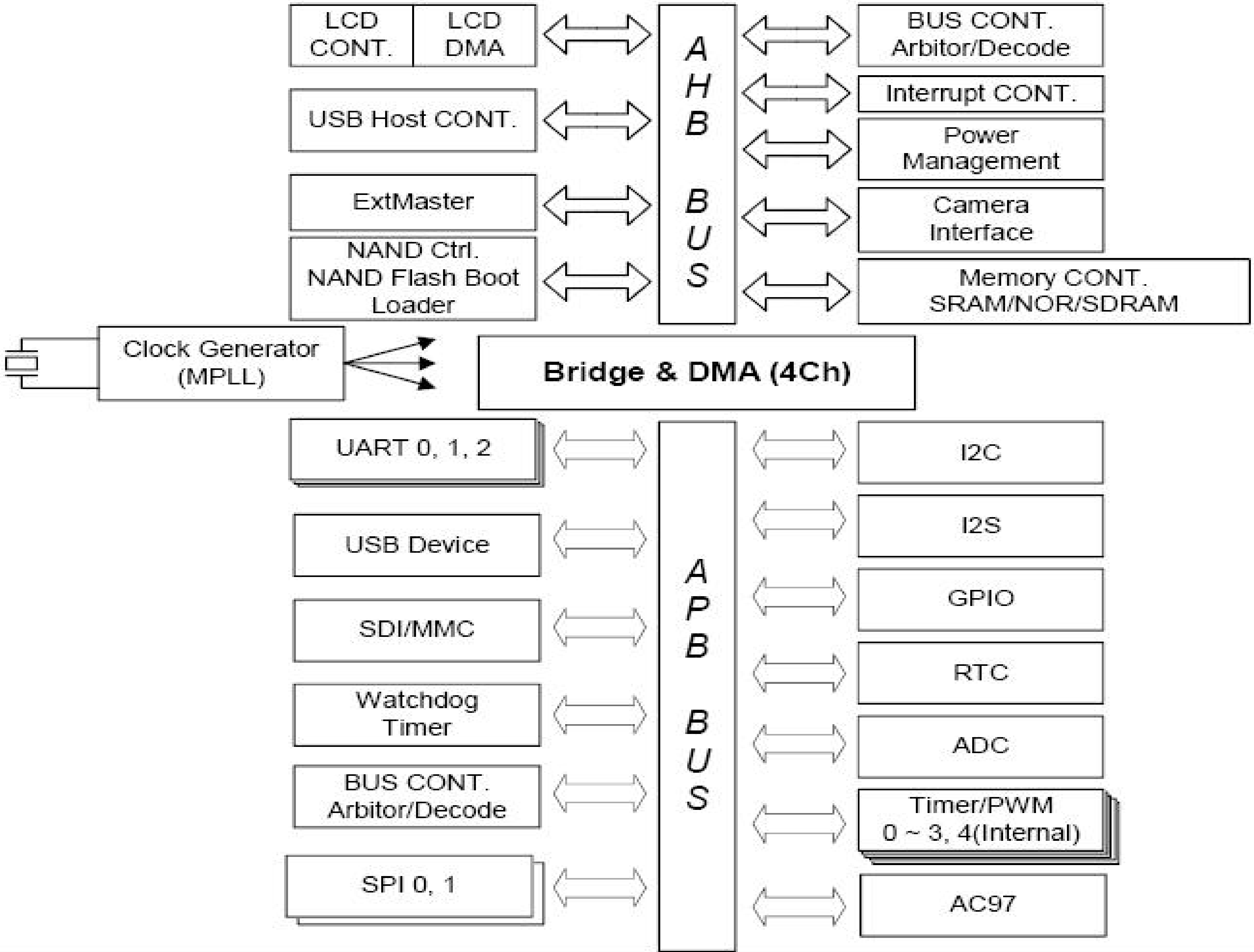
2．在此 PC 是从 FIQ 或 IRQ 取得不能执行的指令的地址。

3．在此 PC 是产生数据中止的加载或存储指令的地址。

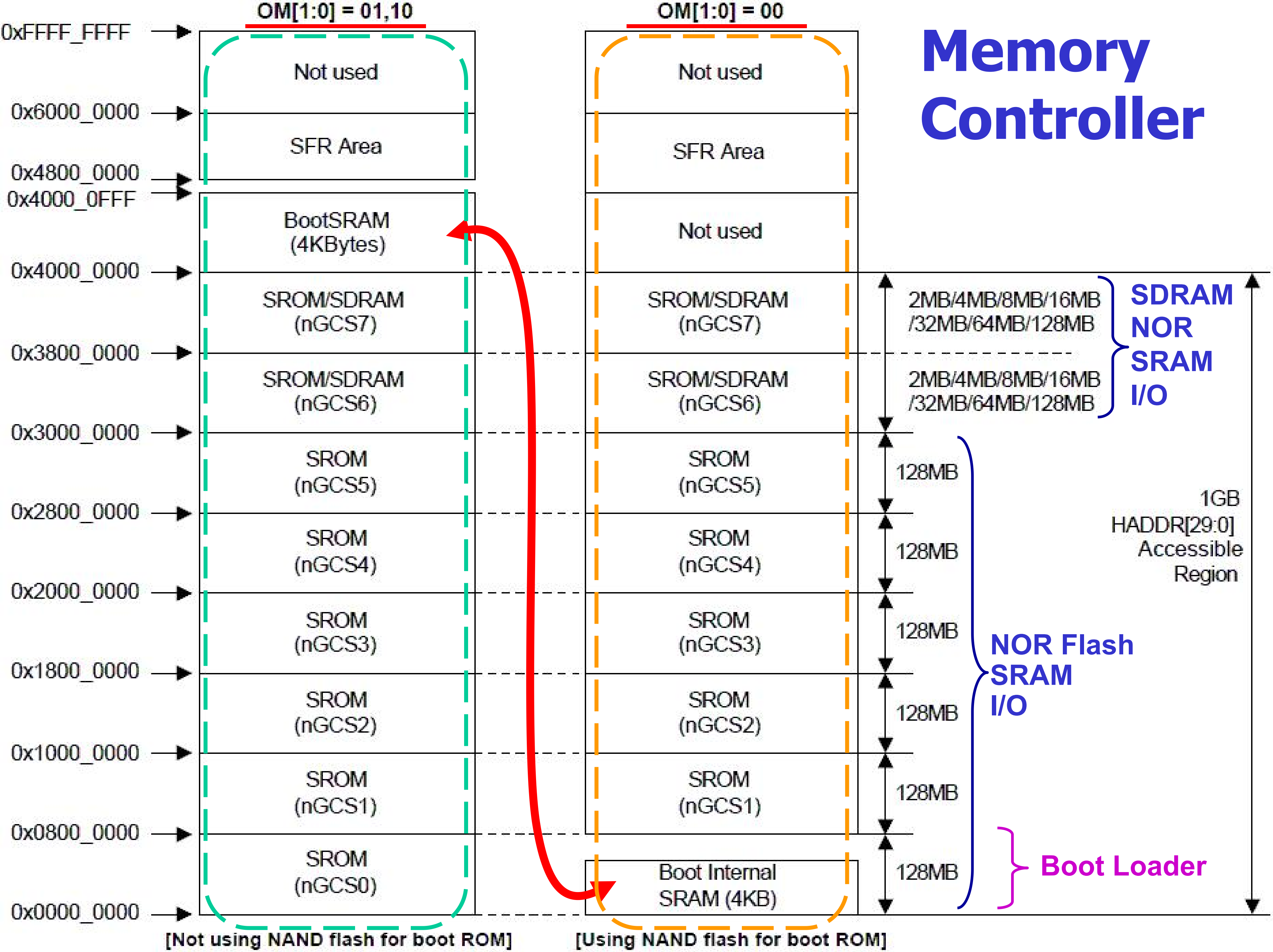
4．系统复位时，保存在 R14_svc 中的值是不可预知的。

S3C2440

Block Diagram (Peripheral only)



Memory Controller



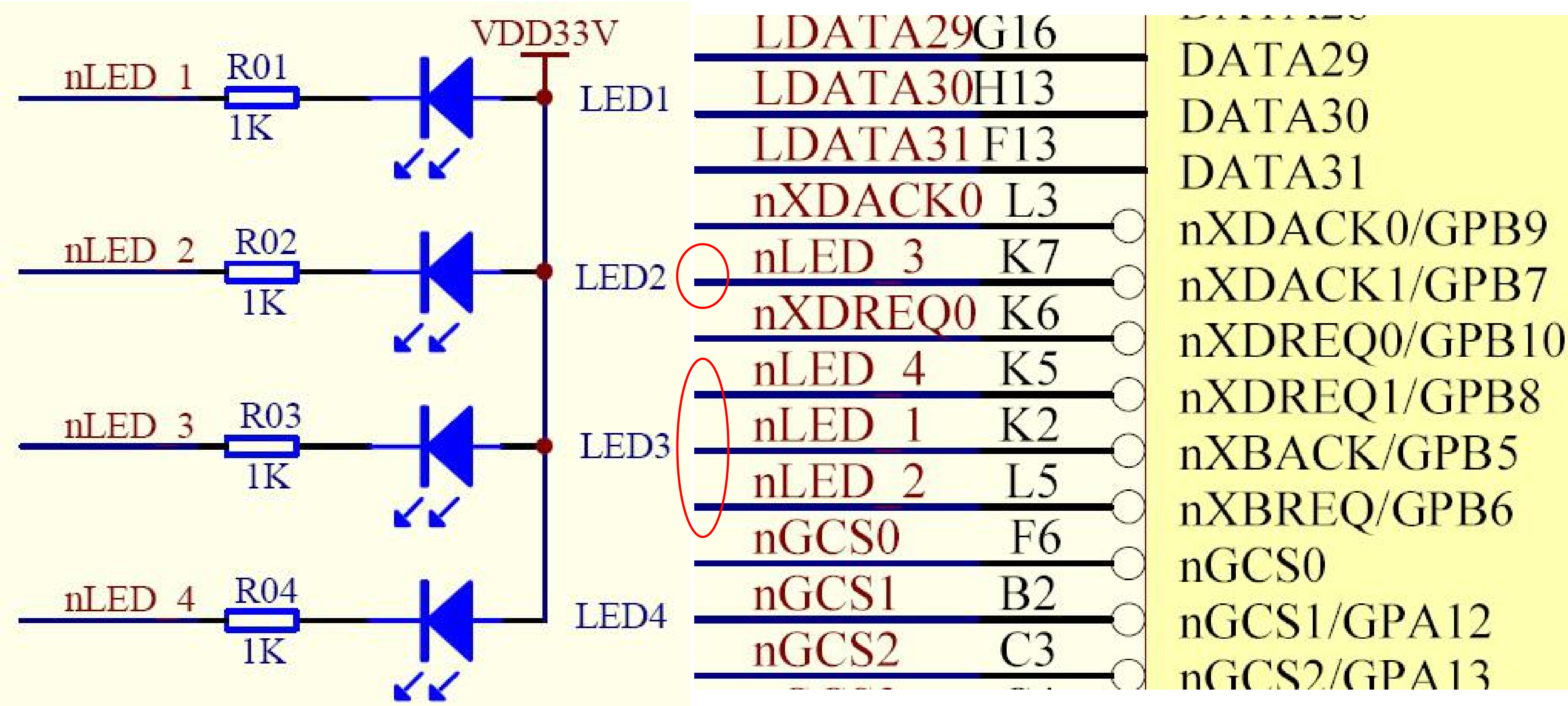
S3C2440 I/O Ports

- The S3C2440has 130 multi-functional input/output port pins - Input/Output, Device function pin, Interrupt.
- User have to define which function of each pin is used before starting the main program.
- If a pin is not used for multiplexed functions, the pin can be configured as GPIO ports.
- The ports are:
  - Port A (GPA): 25-output port
  - Port B (GPB): 11-input/output port
  - Port C (GPC): 16-input/output port
  - Port D (GPD): 16-input/output port
  - Port E (GPE): 16-input/output port
  - Port F (GPF): 8-input/output port
  - Port G (GPG): 16-input/output port
  - Port H (GPH): 9-input/output port
  - Port J(GPJ): 13-input/output port

Multi-functions example

	GPIO	Device Func/Int		
Port B		Selectable Pin Functions		
GPB10	Input/output	nXDREQ0	–	DMA
GPB9	Input/output	nXDACK0	–	DMA
GPB8	Input/output	nXDREQ1	–	DMA
GPB7	Input/output	nXDACK1	–	DMA
GPB6	Input/output	nXBREQ	–	BUS
GPB5	Input/output	nXBACK	–	BUS
GPB4	Input/output	TCLK0		External CLK
GPB3	Input/output	TOUT3	–	Timer
GPB2	Input/output	TOUT2		Timer
GPB1	Input/output	TOUT1	–	Timer
GPB0	Input/output	TOUT0	–	Timer

I/O Port Example



PORT CONTROL

- **Port configuration register (GPACON-GPJCON)** - to determine which function is selected for each pin.
- **Port data register (GPADAT-GPJDAT)**
- **Port pull-up register (GPBUP-GPJUP)**

Port control example

Register	Address	R/W	Description	Reset Value
GPBCON	0x56000010	R/W	Configure the pins of port B	0x0
GPBDAT	0x56000014	R/W	The data register for port B	Undefined
GPBUP	0x56000018	R/W	Pull-up disable register for port B	0x0
Reserved	0x5600001C	–	Reserved	Undefined

GPBCON	Bit	Description	
GPB10	[21:20]	00 = Input 10 = nXDREQ0	01 = Output 11 = reserved
GPB9	[19:18]	00 = Input 10 = nXDACK0	01 = Output 11 = reserved
GPB8	[17:16]	00 = Input 10 = nXDREQ1	01 = Output 11 = Reserved
GPB7	[15:14]	00 = Input 10 = nXDACK1	01 = Output 11 = Reserved
GPB6	[13:12]	00 = Input 10 = nXBREQ	01 = Output 11 = reserved
GPB5	[11:10]	00 = Input 10 = nXBACK	01 = Output 11 = reserved

LEDs

Port control example

GPBDAT	Bit	Description
GPB[10:0]	[10:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

GPBUP	Bit	Description
GPB[10:0]	[10:0]	0: the pull up function attached to to the corresponding port pin is enabled. 1: the pull up function is disabled.

Leds control code

```
ldr r0,=0x56000010 /* GPBCON                */
ldr r1,=0x15400    /* 17-10bit=01010101b */
str r1,[r0]        /* GPB5-8 output       */
ldr r0,=0x56000018 /* GPBUP               */
ldr r1,=0x0
str r1,[r0]        /* All pullup          */
ldr r0,=0x56000014 /* GPBDAT              */
ldr r1,=0x1E0      /* 8-5bit=1111b        */
str r1,[r0]        /* all leds off        */
ldr r1,=0x0
str r1,[r0]        /* all leds on         */
```

本段代码的编写是否有问题？

设计原则-05

- **该做的一定要做到位！不该做的一定不要碰！**
  - 概要设计时就要有完备的功能描述，实施和测试过程中不要有遗漏
  - I/O都是临界资源。多道程序环境下，对临界资源的操作要有规有矩
    - 代码可重入(Reentrance)
    - 互斥操作保证，甚至关中断
- **注意I/O初始化和操作的次序要求**

Leds control code

```
#define rGPBCON     (*(volatile unsigned *)\
          0x56000010) //Port B control
#define rGPBDAT     (*(volatile unsigned *)\
          0x56000014) //Port B data
#define rGPBUP      (*(volatile unsigned *)\
          0x56000018) //Pull-up control B
//-------------------------------------------
void Led_Display(int data)
{
  rGPBDAT = (rGPBDAT & ~(0xf<<5)) |
            ((~data & 0xf)<<5);
}
```

这段代码还有问题吗？

Practise

Tools & LEDs Control

Practise-2

- 设计汇编代码及C代码完成对板上4个发光二极管的各自不同周期的闪烁控制功能。
  - 要求有良好的代码编制习惯 – 设计原则
  - 代码注释≥30%
- 注解：Port初始化、LEDs控制函数、Clock
- 重点：Clock激励