计算机体系结构基础

胡伟武、苏孟豪

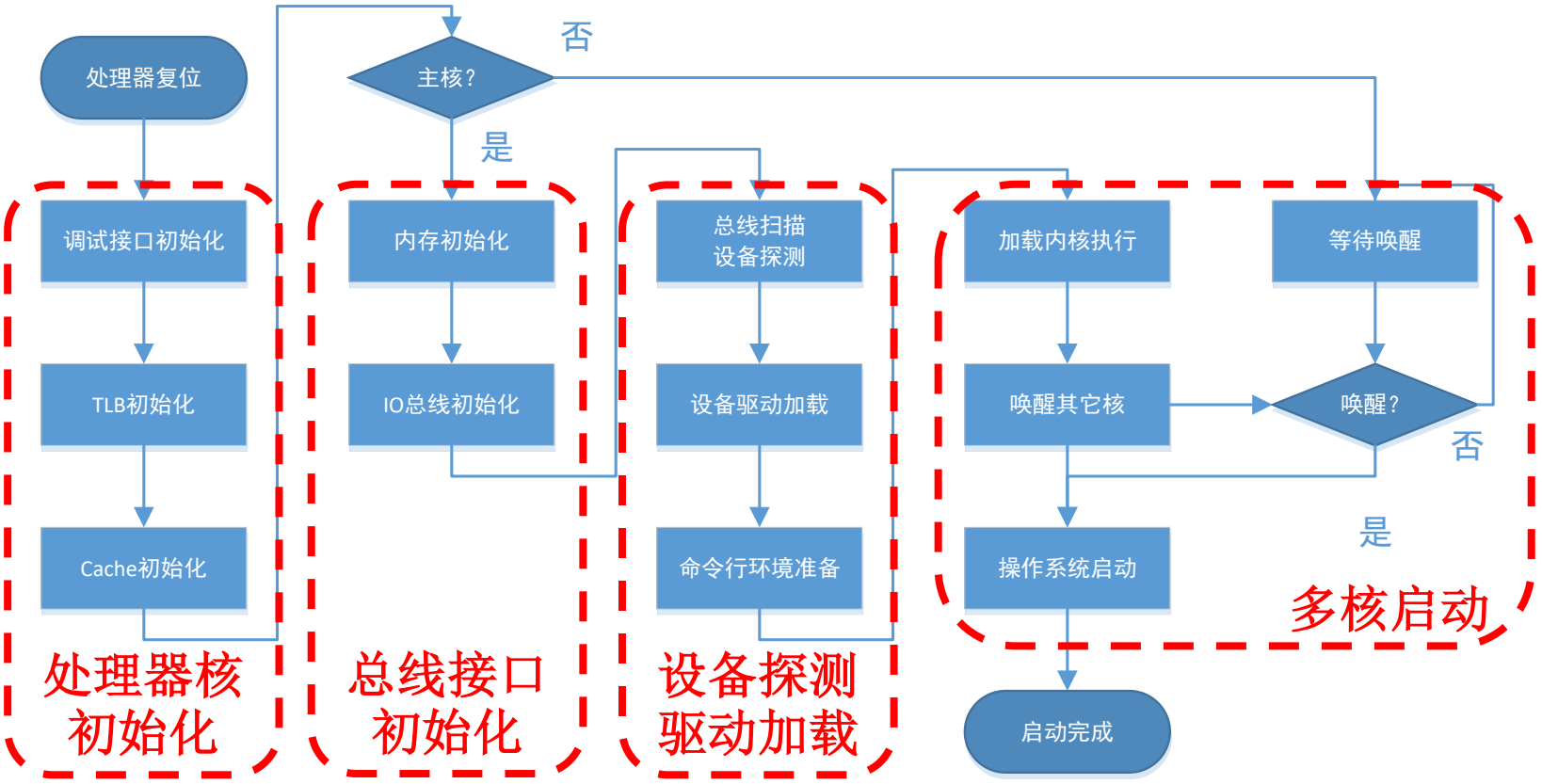
第09章：计算机系统启动过程分析

- 处理器核初始化
- 总线接口初始化
- 设备探测及驱动加载
- 多核启动过程

一句话要点

- 系统启动
  - 从复位到系统可用
- 初始化是什么
  - 将系统各种寄存器状态从不确定设置为确定，将一些模块状态从无序强制为有序的一个过程
- 什么东西需要初始化
  - **CPU**、内存、各类**IO**接口
- 怎么初始化
  - 按照从核内到核外，从片内到片外的次序进行

系统启动过程示意图


处理器复位
主核？
否
是
调试接口初始化
TLB初始化
Cache初始化
处理器核
初始化
内存初始化
IO总线初始化
总线接口
初始化
总线扫描
设备探测
设备驱动加载
命令行环境准备
设备探测
驱动加载
加载内核执行
唤醒其它核
操作系统启动
等待唤醒
唤醒？
否
是
多核启动
启动完成

提纲

- 处理器核初始化
- 总线接口初始化
- 设备探测及驱动加载
- 多核启动过程

处理器复位

- 从芯片引脚输入电平信号，将处理器内部的部分寄存器等状态置为预设值
- 开始取指（MIPS处理器的第一条指令为0xBFC00000）
- 复位输入是一个硬件初始化动作，至少需要完成最基本的芯片状态初始化（保证第一条指令取指执行）
  - 控制寄存器状态初始化，如处于核心态
  - 核内程序计数器（PC）初始化为0xBFC00000
  - 片内对于0xBFCxxxxx段地址的路由通路初始化
  - 清空流水线有效位避免混乱
  - 为什么不初始化更多硬件（如内存、Cache、TLB、寄存器）？

首条指令

- 初始化CP0（0号协处理器，用于控制处理器行为）
  - 状态寄存器Status
  - 原因寄存器Cause
- 初始化软件约定使用的通用寄存器
  - 栈指针：BIOS用的堆栈
  - 全局指针：BIOS用的地址空间
  - 内核约定物理地址240MB-256MB之间的空间给BIOS用

```
mtc0    zero, COP_0_STATUS_REG
mtc0    zero, COP_0_CAUSE_REG
li      t0, SR_BOOT_EXC_VEC
mtc0    t0, COP_0_STATUS_REG
la      sp, stack
la      gp, _gp
```

```
mtc0    zero,c0_sr
mtc0    zero,c0_cause
lui     t0,0x40
mtc0    t0,c0_sr
lui     sp,0x8f90
addiu   sp,sp,-16384
lui     gp,0x8f9a
addiu   gp,gp,-8192
```

外部调试接口初始化

- 外部调试接口本身不是处理器核初始化的内容，但尽早完成初始化能够开始人机交互，方便软件调试
- 输出
  - 蜂鸣器
  - 数码管显示
- 输入输出
  - 串口控制器
- 显示器这样的设备相比之下是比较复杂的接口，在启动的后期才会使用

串口初始化

- 建立主机与被调试机之间的通信协议
  - 主要的配置是串口波特率
  - 其中GS3_UART_BASE是串口控制器内部的寄存器

```
LEAF(initserial)
        li   a0, GS3_UART_BASE
        li       t1, 128
        sb       t1, 3(a0)
        li       t1, 0x12
        sb       t1, 0(a0)
        li       t1, 0x0
        sb       t1, 1(a0)
        li       t1, 3
        sb       t1, 3(a0)
        #srl     t1, t1, 0x8
        li       t1, 0
        sb       t1, 1(a0)
        li       t1,71
        sb       t1,2(a0)
        jr       ra
        nop
END(initserial)
```

加载串口设备基地址

配置串口波特率分频，当串口控制器输入频率为33MHz时，将串口通讯速率设置在115200，分频的方式为

33,000,000 / 16 / 0x12 = 114583

由于串口通信有固定的格式，只要两端的速率保持在一定范围之内就可以保证传输的正确性

TLB初始化

- **TLB用于虚实地址转换**
- **从TLB来看，MIPS地址空间，分为非映射与映射**
  - **0x80000000 – 0x8FFFFFFF：Cached/Unmapped内存空间**
  - **0x9FC00000 – 0x9FCFFFFF：Cached/Unmapped取指空间**
  - **0xB0000000 – 0xBFFFFFFF：Uncached/Unmapped IO空间**
  - **对于BIOS，基本使用非映射空间，无需TLB转换**
- **对于大地址空间的使用，会借助TLB映射**
  - **0xC0000000 -> 0x40000000：对应显存等大IO空间**

TLB初始化代码

- 逐一清空每一个TLB项
  - 需要使用时再填入正确映射（通过TLB Refill例外）

```
LEAF(CPU_TLBClear)
        li      a3, 0                   循环控制变量
        li      a2, 64
        li      a2, PG_SIZE_4K          设置页大小
        MTC0    a2, COP_0_TLB_PG_MASK
1:
        MTC0    zero, COP_0_TLB_HI      对TLB_HI寄存器初始化，对应于VPN
        MTC0    zero, COP_0_TLB_LO0     对TLB_LO0寄存器初始化，对应于PFN0
        MTC0    zero, COP_0_TLB_LO1     对TLB_LO1寄存器初始化，对应于PFN1
        mtc0    a3, COP_0_TLB_INDEX     当前处理的TLB项索引号
        addiu   a3, 1
        tlbwi                           操作TLB表
        bne      a3, a2, 1b
        nop

        jr       ra
        nop
END(CPU_TLBClear)
```

Cache初始化

- 复位后BIOS刚开始启动时，处理器从非缓存空间开始执行
  - 上百拍完成一条指令的的取指和执行
  - Cache初始化后，跳转到Cache空间执行，全速流水
  - 尽早完成Cache初始化，使用Cache地址执行能够大大提升启动速度
- 从Cache来看，MIPS地址空间，分为非缓存与缓存
  - 0x80000000 – 0x8FFFFFFF：Cached/Unmapped内存空间
  - 0x9FC00000 – 0x9FCFFFFF：Cached/Unmapped取指空间
  - 0xB0000000 – 0xBFFFFFFF：Uncached/Unmapped IO空间
  - Cached/Unmapped取指空间和Uncached/Unmapped IO空间的物理地址是相同的，Cache失效会自动到IO空间去取指

一级Cache初始化代码

```
LEAF(godson2_cache_init)
    lui     a0, 0x8000
    li      a2, (1<<14)

    mtc0    $0, CP0_TAGHI
    mtc0    $0, CP0_TAGLO
    li      a1, 0x22
    addu    v0, $0, a0
    addu    v1, a0, a2
1:
    slt     a3, v0, v1
    beq     a3, $0, 2f
    nop
     mtc0    a1, CP0_ECC
    cache   Index_Store_Tag_D, 0x0(v0)
    cache   Index_Store_Tag_D, 0x1(v0)
    cache   Index_Store_Tag_D, 0x2(v0)
    cache   Index_Store_Tag_D, 0x3(v0)
    mtc0    zero, CP0_ECC
    cache   Index_Store_Tag_I, 0x0(v0)
    cache   Index_Store_Tag_I, 0x1(v0)
    cache   Index_Store_Tag_I, 0x2(v0)
    cache   Index_Store_Tag_I, 0x3(v0)
    b       1b
    daddiu   v0, v0, 0x20
2:
    jr      ra
    nop
END(godson2 cache init)
```

基地址
64KB/4路，为Index的实际数量

对TAGHI进行初始化
对TAGLO进行初始化
A1寄存器用于存放ECC校验码，全0时，ECC位为0x22

根据V0和V1的值判断是否已经完成所有项的遍历

数据Cache采用ECC检验，初始化为0x22

对4路数据Cache分别进行写TAG操作

指令Cache采用奇偶检验，初始化为0

对4路指令Cache分别进行写TAG操作

开Cache执行

- **Cache初始化仅仅是使Cache可用**
  - **要使用Cache必须开Cache，并使用Cache地址操作**
  - **注意0xA0000000段和0x80000000段的物理地址是相同的**

```
## enable kseg0 cachablilty####
        mfc0    t6, CP0_CONFIG
        ori     t6, t6, 7
        xori    t6, t6, 4
        mtc0    t6, CP0_CONFIG

#jump to cached kseg0 address
        PRINTSTR("Jump to 9fc\r\n")
        lui     t0, 0xdfff
        ori     t0, t0, 0xffff
        bal     1f
        nop
1:
        and     ra, ra, t0
        addiu   ra, ra, 16
        jr      ra
        nop
2:
```

开Cache执行
使能0x80000000 – 0x9FFFFFFF的Cache功能

将程序地址跳转到0x9FC00000段执行
BIOS执行到该位置时，应该在Uncache的取指执行，所在的地址段为0xBFC00000 – 0xBFCFFFFF。

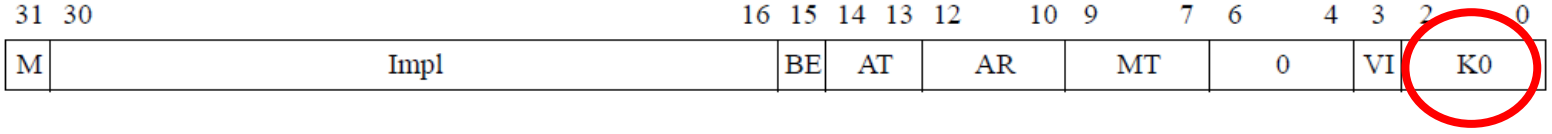
BAL指令将会将延迟槽后的指令地址填入RA寄存器，正常情况下为0xBFCxxxxx。

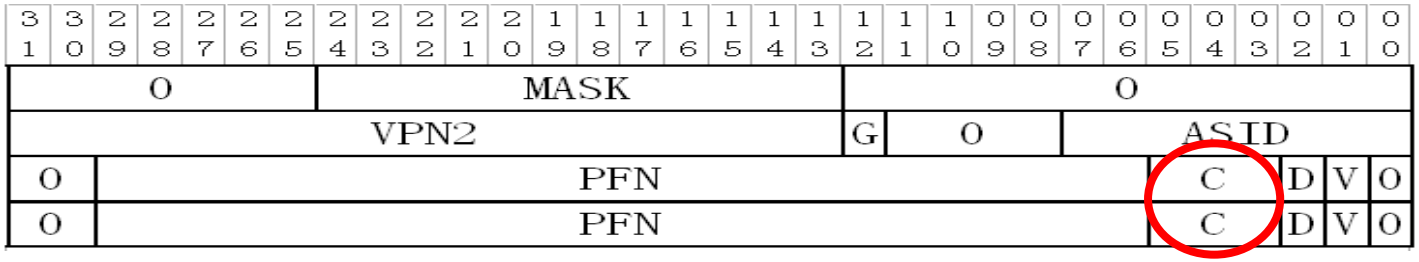
0xBFCx & 0xDFFF = 0x9FCx，地址变换为Cached
将记录的地址再加16，实际上是原指令后的第4条，也即标号2处的指令
跳转到该指令的Cache空间处

Cache算法配置

- 三位决定Cache算法
  - 对于kseg0段（0x80000000段），由Config寄存器的k0域决定
  - 其它部分由TLB表项决定
  - 2、3分别表示uncached和cached，其余6个值自己定

Figure 8-23 Config Register Format





提纲

- 处理器核初始化
- 总线接口初始化
- 设备探测及驱动加载
- 多核启动过程

内存接口初始化

- 到此为止，BIOS从 Flash读程序，写IO或控制寄存器器
  - 相比Flash设备，内存接口的访问性能大大提升
- 内存接口的初始化与核内部件的初始化的差别
  - Cache/TLB的初始化主要是将内容设置为无效
  - 内存接口的初始化针对接口控制
    - 通过内存的SPD（并非必需），获取内存大小，类型，频率，延迟等各种信息
    - 根据所得到的内存信息对控制器和内存进行设置
    - 可能还有对于时序配合的信号训练
    - 对内容并不关心（ECC内存除外，不初始化内容会导致ECC错）

内存控制器配置代码

- 将预先定义好的值写入内存控制器的相应寄存器中
- 内存控制器将自动对内存进行初始化配置
  - 主要设置延迟、匹配阻抗等
- 初始配置之后，会再对内存信号进行训练

```
ddr2_config:
    daddu   a2, a2, s0
    dli     t1, DDR_PARAM_NUM
    daddiu  v0, t8, 0x0
1:
    ld      a1, 0x0(a2)
    sd      a1, 0x0(v0)
    daddiu  t1, t1, -1
    daddiu  a2, a2, 0x8
    daddiu  v0, v0, 0x8
    bnez    t1, 1b
    nop
```

a2为调用该程序时传入的参数，与s0之和用于表示初始化参数在FLASH中的基地址
t1用于表示内存参数的个数
t8为调用该程序时传入的参数，用于表示内存控制器的寄存器基地址

初始化的过程就是从FLASH中取数再写入内存控制器中的寄存器的过程

IO总线初始化

- 根据不同IO总线的需求进行针对性的初始化
- 通过初始化配置消除与具体实现相关的总线特性，保证软件兼容性
  - 信号定义，硬件实现各不相同：HyperTransport、PCIE等
  - 软件协议兼容PCI：配置空间、IO空间及Memory空间
- 龙芯3号中使用HyperTransport接口
  - 地址划分：确定配置访问、IO访问及Memory访问的地址
  - 设定DMA访问地址
  - 对总线频率、宽度进行重新设置

提纲

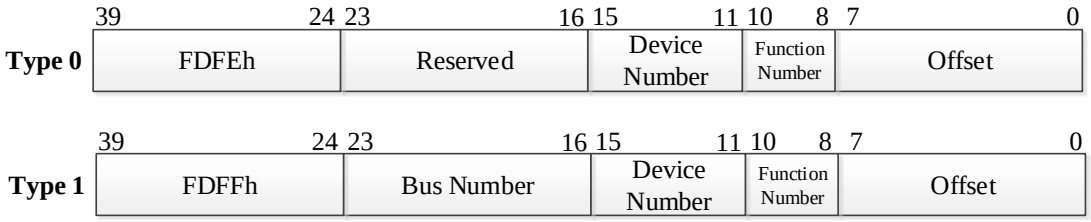
- 处理器核初始化
- 总线接口初始化
- 设备探测及驱动加载
- 多核启动过程

PCI协议下的工作流程

- 上世纪九十年代提出，软件结构被PCIE和HT等继承
  - 配置空间、IO空间、Memory空间
- 通过配置空间探测总线设备
  - 配置空间大部分情况下是对设备的属性进行刻画
  - 在完成设备探测之后基本不再使用配置空间
- 对总线上的各个设备所需的地址空间进行分配
  - IO、Memory空间是真正使用设备功能时的地址空间
  - 对于IO和内存统一编址的CPU，两者区别不大
- 使用分配好的空间对各个设备进行控制
  - 主要是驱动程序加载
- 特点：
  - 总线设备发生变化时无需修改软件
  - 同一总线支持多个相同设备不会导致冲突

设备探测

- 使用配置空间：以HyperTransport的配置空间为例
  - 通过总线号、设备号及功能号可以遍历总线设备
  - 高位地址访问不同设备，Offset是设备内地址偏移



  - 通过规定的寄存器得到设备的唯一标识、设备类型等
    - Device ID、Vendor ID
  - 通过基址寄存器（BAR）获取并对设备空间进行配置

设备配置空间寄存器分布

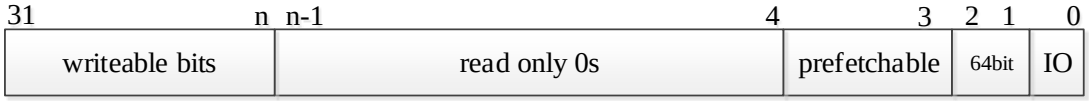
- 一般放在IO控制器上（如PCIE控制器）
  - 所有设备的空间分布都一致

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h 14h 18h 1Ch 20h 24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

Note: *Shaded registers contain minimum-required read-write bits. Other registers are read-only or contain only device-dependent bits.*

地址空间分配

- 基址寄存器（BAR）
  - 向该寄存器写入全1，读出再判断0的个数来获取空间需求
  - 再向该寄存器写入分配好的基地址供设备进行命中判断



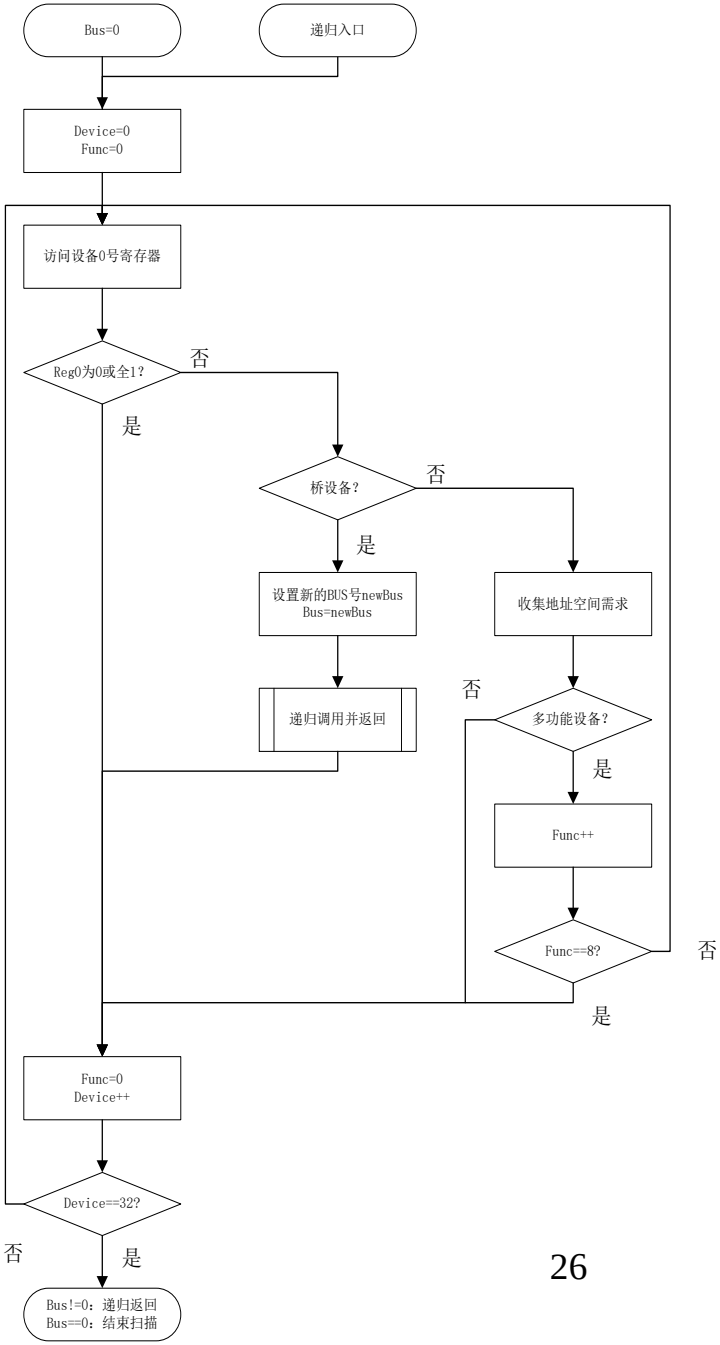
- 每个设备地址空间为2的幂次方对齐
- 先收集所有的设备空间信息
  - 按照大小排序
  - 根据顺序分配基地址，以减少空间碎片

PCI设备的探测

1. 将初始总线号、初始设备号、初始功能号设为0；
2. 使用当前的总线号、设备号、功能号组成一个配置空间地址，这个地址的构成如图9.3所示，使用该地址，访问其0号寄存器，检查其设备号；
3. 如果读出全1或全0，表示无设备；
4. 如果该设备为有效设备，检查每个BAR所需空间大小，并收集相关信息；
5. 检测其是否为多功能设备，如果是则将功能号加1重复扫描，执行第2步；
6. 如果该设备为桥设备，则给该桥配置一个新的总线号，再使用该总线号，从设备号0、功能号0开始递归调用，执行第2步；
7. 如果设备号非31，则设备号加1，继续执行第2步；如果设备号为31，且总线号为0，表示扫描结束；如果总线号非0，则退回上一层递归调用；

PCI设备探测过程

- 对多功能设备扫描FUNC
  - 可以有更多的地址空间BAR
- 对桥设备递归调用该过程
  - 如南桥上有很多设备
- 遍历整个总线获得所有设备信息
  - 同时对设备地址空间大小进行排序
- 再次遍历对每个设备空间进行配置
  - 把设备地址空间的信息传递给设备驱动程序，设备驱动程序使用配置好的设备地址空间



地址空间分配示例

- 设备内部的地址空间，CPU通过Uncache或IO指令访问
- 配置空间扫描后得到的设备及空间需求

设备号	名称	BAR号	大小
1	USB控制器	0	4KB
2	显示控制器	0	128MB
		1	64KB
3	网卡控制器	0	4KB
		1	16KB

- 最终的地址空间分配

设备号	名称	BAR号	大小	起始地址	结束地址
1	USB控制器	0	4KB	0x48015000	0x48015FFF
2	显示控制器	0	128MB	0x40000000	0x47FFFFFF
		1	64KB	0x48000000	0x4800FFFF
3	网卡控制器	0	4KB	0x48014000	0x48014FFF
		1	16KB	0x48010000	0x48013FFF

提纲

- 处理器核初始化
- 总线接口初始化
- 设备探测及驱动加载
- 多核启动过程

多核初始化

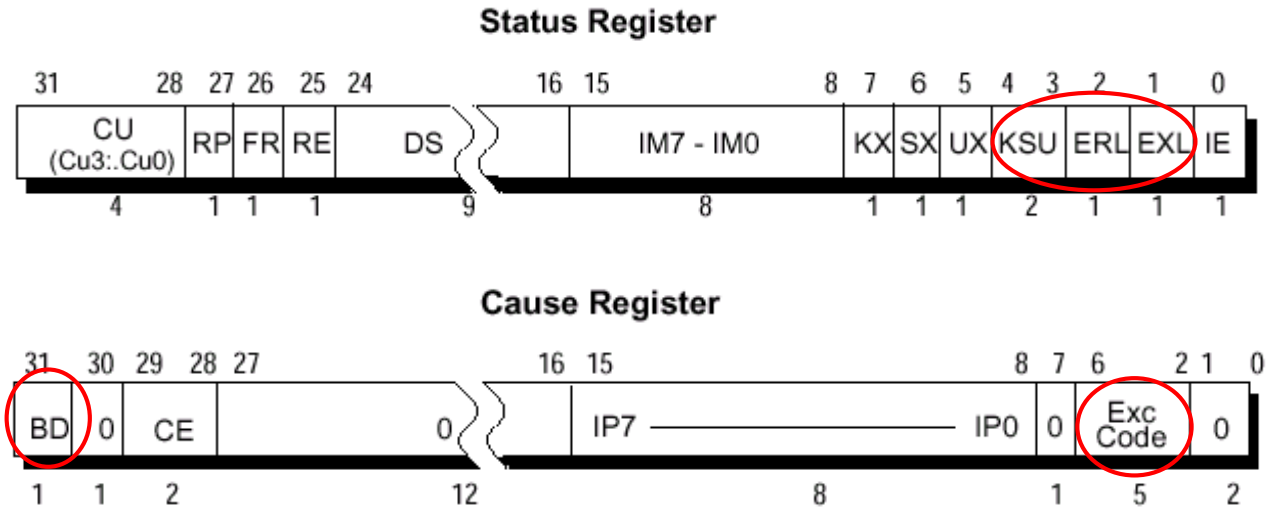
- 一个核执行主流程，其它核仅负责私有部件的初始化
  - 主核：核内私有部件、片内共享部件、总线接口
  - 从核：核内私有部件
- 通过核间通信机制进行同步
  - 串口初始化后，通知从核，开始打印输出
  - 共享Cache初始化后，通知从核，开Cache执行
  - 内存初始化后，通知从核，可以读写内存
- 可以通过并行化共享部件初始化的方法来加速启动
  - 每个核初始化一部分共享Cache
  - 每个核执行不同的初始化：Cache、内存、接口

核间通信机制

- 核间中断机制，以龙芯3A为例
  - 每个核有32个不同的中断可供软件使用
  - 可以约定为不同核产生的中断，或者对应不同事件
- 信箱寄存器（查询方式）
  - 多个寄存器供传递参数及特殊事件

名称	读写权限	描述
IPI_Status	R	32位状态寄存器，被置1且对应位使能情况下，处理器核中断线被置位
IPI_Enable	RW	32位使能寄存器，控制对应中断位是否有效
IPI_Set	W	32位置位寄存器，往对应的位写1，则对应的STATUS寄存器位被置1
IPI_Clear	W	32位清除寄存器，往对应的位写1，则对应的STATUS寄存器位被清0
MailBox0	RW	缓存寄存器，供传递参数使用
MailBox01	RW	缓存寄存器，供传递参数使用
MailBox02	RW	缓存寄存器，供传递参数使用
MailBox03	RW	缓存寄存器，供传递参数使用

Status和Cause寄存器



多核唤醒

- 内核启动过程中，会将从核逐一加入系统管理
  - 这一过程称之为“唤醒”
- 唤醒中主从核的运行
  - 主核：将从核需要运行的程序指针、参数写入信箱寄存器。等待从核被唤醒
  - 从核：轮询信箱寄存器，发现非零值则跳转执行。执行完相关的程序后，通过信箱寄存器通知主核
  - 主核：收到信箱寄存器通知后，继续下一个核的唤醒或其它操作

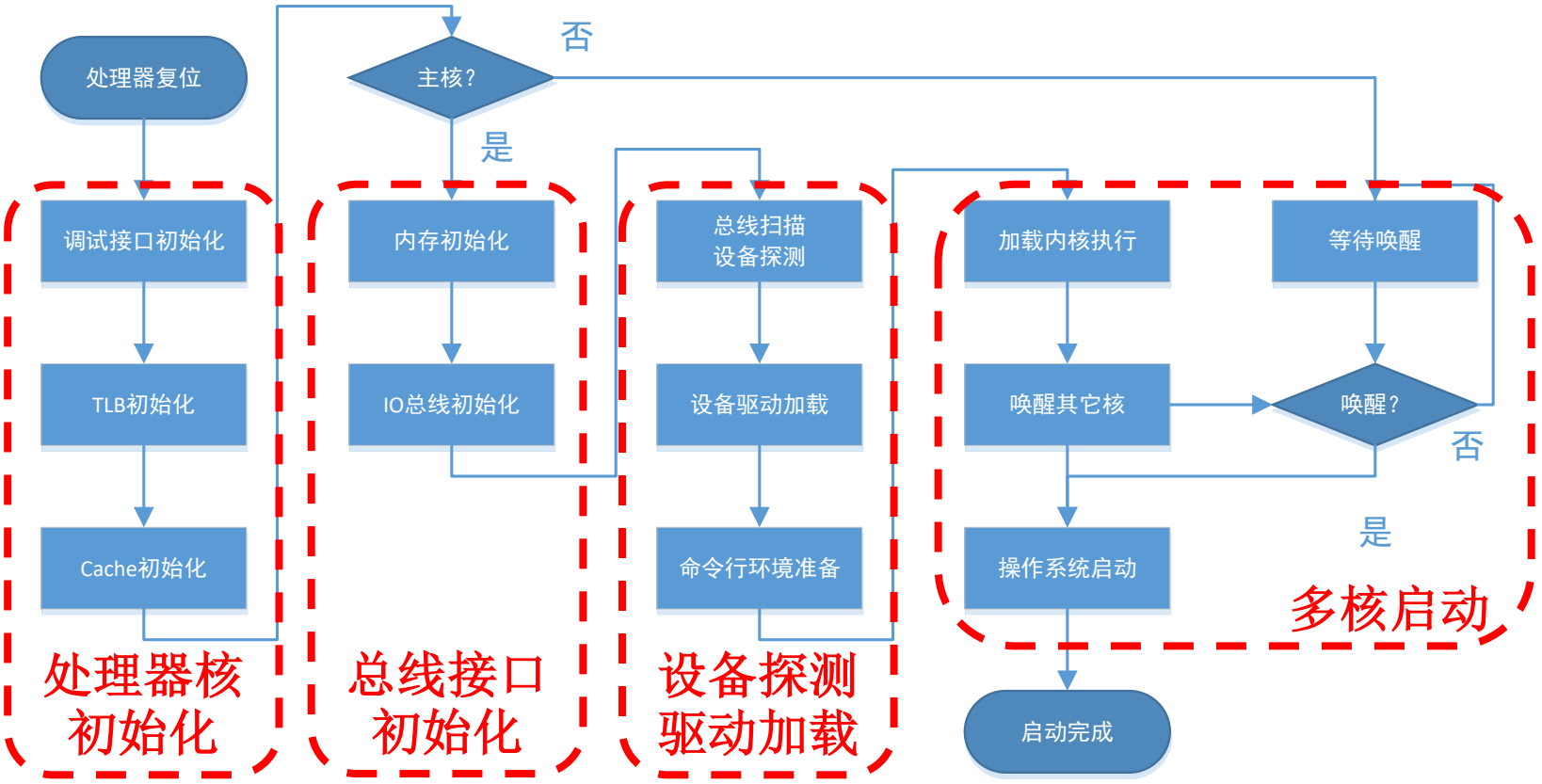
小结

- CPU刚上电时，硬件只对必要的状态进行复位
  - 除了从PC到BIOS接口有一丝光亮，其它都是漆黑一片
  - 从0xBFC00000取第一条指令时要绕过Cache和TLB
- 软件初始化必要的调试接口：指示灯、蜂鸣器、串口
  - 从SPI接口（最高33MHz）取指，每拍一位，32拍才取回一条指令，CPU要1000拍才执行一条指令；CPU内部有些朦胧的光亮
  - Load/Store指令访问IO接口的控制寄存器与访问内存有不同的含义
- 软件初始化CPU内部
  - 先初始化Cache，CPU可以高速运行；再打开TLB，访问地址空间更大
  - CPU内部一片光明，但外部还是漆黑一片
- 软件初始化内存控制器
  - 通过I2C总线读取内存条信息
  - 内存控制器参数太多，主要是从BIOS空间读取内存参数并写入内部控制寄存器
  - DDR4 3200每传输一位数据只有0.3ns，电信号只能在主板上行进4-5cm
  - 内存是CPU的后花园，这时CPU通往后花园的路已经打通

小结

- 软件初始化IO接口
  - PCI协议：上世纪九十年代提出，软件结构被PCIE和HT等继承
  - 配置空间：通过配置空间探测总线设备，配置空间大部分情况下是对设备的属性进行刻画；CPU通过对设备空间的扫描发现所有IO控制器；
  - IO空间、Memory空间：驱动程序真正使用设备功能时的地址空间；对于IO和内存统一编址的CPU，两者区别不大；
  - 总线设备发生变化时无需修改软件，同一总线支持多个相同设备不会导致冲突
  - Intel的厉害之处：PCI协议使得硬件能自动识别IO设备并加载驱动程序；UEFI协议使得最新的CPU可以运行十年前的操作系统；X86指令系统使得最新的CPU可以运行几十年前的应用程序
- 唤醒其它处理器核，进入多核状态
  - 多核之间通过信箱等方式通信
  - 信箱可以是内存单元，也可以是CPU内部寄存器
- 至此，CPU内部光明一片，对外四通八达

小 结


处理器复位
主核？
否
是
调试接口初始化
TLB初始化
Cache初始化
处理器核
初始化
内存初始化
IO总线初始化
总线接口
初始化
总线扫描
设备探测
设备驱动加载
命令行环境准备
设备探测
驱动加载
加载内核执行
唤醒其它核
操作系统启动
等待唤醒
唤醒？
否
是
多核启动
启动完成