

汇编2018

注：大二小学期资料

<https://cloud.tsinghua.edu.cn/f/f99bbdbd3f9e4df89477/>

大纲

- 1. 整数与浮点数
- 2. X86汇编语言与C语言
- 3. MIPS32

*由于这学期重点放在64位上，所以之前的题目涉及到很多的32位的题目都不太有用了。

1. 整数与浮点数

- A. 大小， 比特， 字节， 字
- B. 机器字长， 小端（ 大端 ） 排布
- C. 整数：补码
- D. 浮点数：规格化， 非规格化

2. X86汇编语言与C语言

- E. 操作类型：算逻操作，传递数据，顺序转移
- F. 数据格式
- G. 寄存器位置
- H. 数据传送、寻址模式
- I. 整数计算指令
- J. 条件码、比较指令、测试指令、跳转指令
- K. 栈，参数传递
- L. 数组与结构

3. MIPS

- M. MIPS指令集特点
- N. MIPS汇编指令
- O. 异常处理
- P. 虚存与MIPS32内存管理

A.大小, 比特, 字节, 字

- 大小 : 1K, 1M, 1G, 1T, 1P
- 比特 : 1Bit
- 字节 : 1Byte = 8Bit
- 字 : 1Word = 2Byte = 16Bit
- 32位字
- 64位字

B. 机器字长， 小端（ 大端 ）排布

- **机器字长**：一次整数运算所能处理的位数， 数据地址的长度
- **32位字长**：4GB地址
- **64位字长**：48位宽的地址256TB
- **最小单位**：Byte
- **小端排布**：低对低、高对高
- 数值0x01234567， 地址0x100

	0x100	0x101	0x102	0x103	
	67	45	23	01	

C. 整数的表示：补码

- 无符号数和有符号数
- 补码：反码+1
- 补码可以直接相加

D. 浮点数的表示：规格化，非规格化

- 表示: $(-1)^S * M * 2^E$
- E与exp的关系 (bias), M与frac的关系
- **32位浮点数** : E (8位), M (23位)
- **64位浮点数** : E (11位), M (52位)
- **规格化**浮点数 : $\text{exp} \neq \text{全0/全1}$, $E = \text{exp} - \text{bias}$, $M = 1.\text{frac}$
- **非规格化**浮点数 : $\text{exp} = \text{全0}$, $E = -\text{bias} + 1$, $M = 0.\text{frac}$
- **特殊值** : $\text{exp} = \text{全1}$, $\text{frac} = \text{全0}$ 即无穷, $\text{frac} \neq \text{全0}$ 即NaN

D. 浮点数的表示 Cont'

已知 `int x = ...; int y = ...; float f = ...; double d = ...; unsigned int ux = x; unsigned int uy = y;`

判断下面的等式（或不等式或推导）是否成立。

- | | | |
|--|---|---|
| (1) <code>x == (int)(float) x</code> | 错 | Float的frac只有23位, int32位, 可能产生精度丢失
而double的52位则不会 |
| (2) <code>x == (int)(double) x</code> | 对 | |
| (3) <code>f == (float)(double) f</code> | 对 | |
| (4) <code>d == (float) d</code> | 错 | 精度丢失 |
| (5) <code>d > f -> -f > -d</code> | 对 | |
| (6) <code>(d+f)-d == f</code> | 错 | |
| (7) <code>(x>y) == (-x<-y)</code> | 错 | $X = -2^{31}$ |
| (8) <code>(x -x)>>31 == -1</code> | 错 | $X = 0$ |
| (9) <code>~x+~y == ~(x+y)</code> | 错 | |
| (10) <code>(int)(ux-uy) == -(y-x)</code> | 对 | 有无符号不影响运算 |

E. 算逻操作, 传递数据, 顺序转移

- 算逻操作

- add, sub, imul, sal, sar, xor
- inc, dec, neg, not
- leaq

- 传递数据

- mov

- 顺序转移

- jmp
- call, ret

F. 数据格式

- B, **Byte**, 1byte
- W, **Word**, 2byte
- L, **Long**, 4byte
- Q, **Quadword**, 8byte

G. 寄存器位置

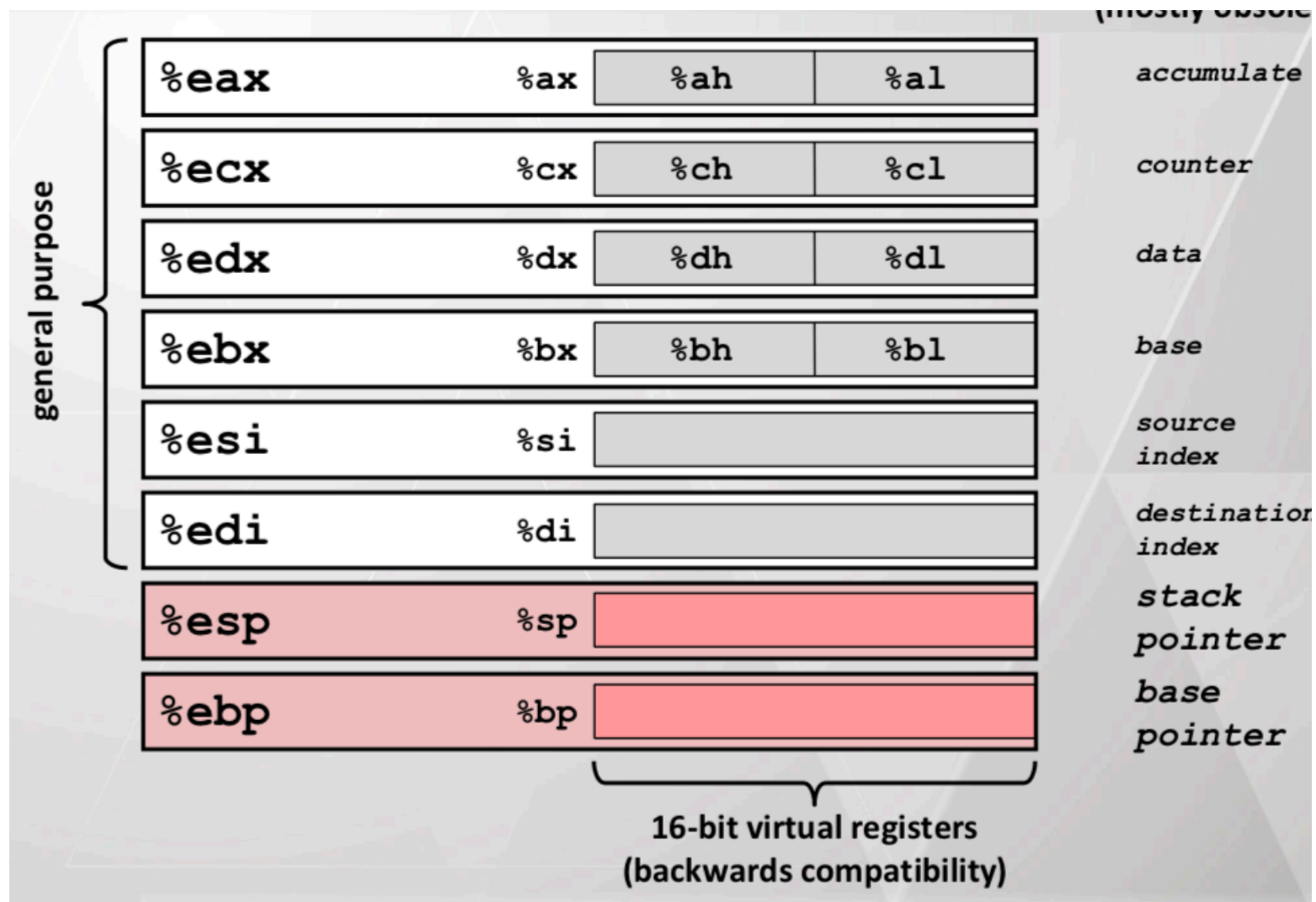
- 32与64位的不同

%rax	%eax	%r8	%r8d
%rdx	%edx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rbx	%ebx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

64位

可以通过%eax访问低32位

G. 寄存器位置 Cont'



32位

通过%al可以访问低8位

H. 数据传送，寻址模式

- 数据传送指令mov : movl, movq, movb ...
- AT&T : mov Source, Dest (后边是目标)
- **三种数据类型** :
 - 1. 立即数, \$0x400, \$-533
 - 2. 寄存器, %rax, %r13, %al
 - 3. 内存, (%rax), 16(%rsp)
- **寻址模式** : 基址寄存器, 索引寄存器, 比例因子, 偏移量
 - 变址寻址 $D(Rb, Ri, S) = Mem[Reg[Rb] + Reg[Ri] * S + D]$
- **地址计算指令**leaq

I. 整数计算指令

双操作数指令

addq Src, Dest **Dest = Dest + Src**

subq Src, Dest **Dest = Dest - Src**

imulq Src, Dest **Dest = Dest * Src**

salq Src, Dest **Dest = Dest << Src**

sarq Src, Dest **Dest = Dest >> Src**

shrq Src, Dest **Dest = Dest >> Src**

xorq Src, Dest **Dest = Dest ^ Src**

andq Src, Dest **Dest = Dest & Src**

orq Src, Dest **Dest = Dest | Src**

**Signed Multiply
与shll等价**

算术右移

逻辑右移

I. 整数计算指令 Cont'

单操作数指令

`incq Dest`

$Dest = Dest + 1$

`decq Dest`

$Dest = Dest - 1$

`negq Dest`

$Dest = - Dest$

`notq Dest`

$Dest = \sim Dest$

以及使用leaq用于整数运算

J. 条件码、比较指令、测试指令、跳转指令

- **条件码**：CF, ZF, SF, OF
 - CF, 进位, 无符号数的溢出
 - ZF, 测零, 是零则为1
 - SF, 负数, 是负数则为1
 - OF, 补码运算溢出, 带符号数运算的溢出, 即正正得负, 负负得正等
 - 算术指令设置
- **比较指令cmp**：cmp b, a 类似于计算a-b但不修改具体值
- **测试指令test**：test a, b 计算a&b, 多用于test a&a = a
- **跳转指令j**：
 - 无条件跳转jmp,
 - 有符号je, jne, js, jns, jg, jge, jl, jle
 - 无符号ja, jb

J. 条件码、比较指令、测试指令、跳转指令 Cont'

- X86-32位体系结构中的条件跳转指令 jg 是用于符号数比较还是无符号数比较的?
- 其产生跳转的成立条件是 $\sim(SF \wedge OF) \& \sim ZF$ 为真, 请解释为何是这一条件。
- 有符号比较
- 先回答ZF
- 再根据SF和OF分情况讨论

K. 栈， 参数传递

- **栈**：与栈顶寄存器%rsp
- **函数调用**：在栈中压入返回地址
- **参数**：前6个保存在%rdi, %rsi, %rdx, %rcx, %r8, %r9 中， 其余从右到左压栈

Registers

■ First 6 arguments

%rdi
%rsi
%rdx
%rcx
%r8
%r9

■ Return value

%rax

Stack

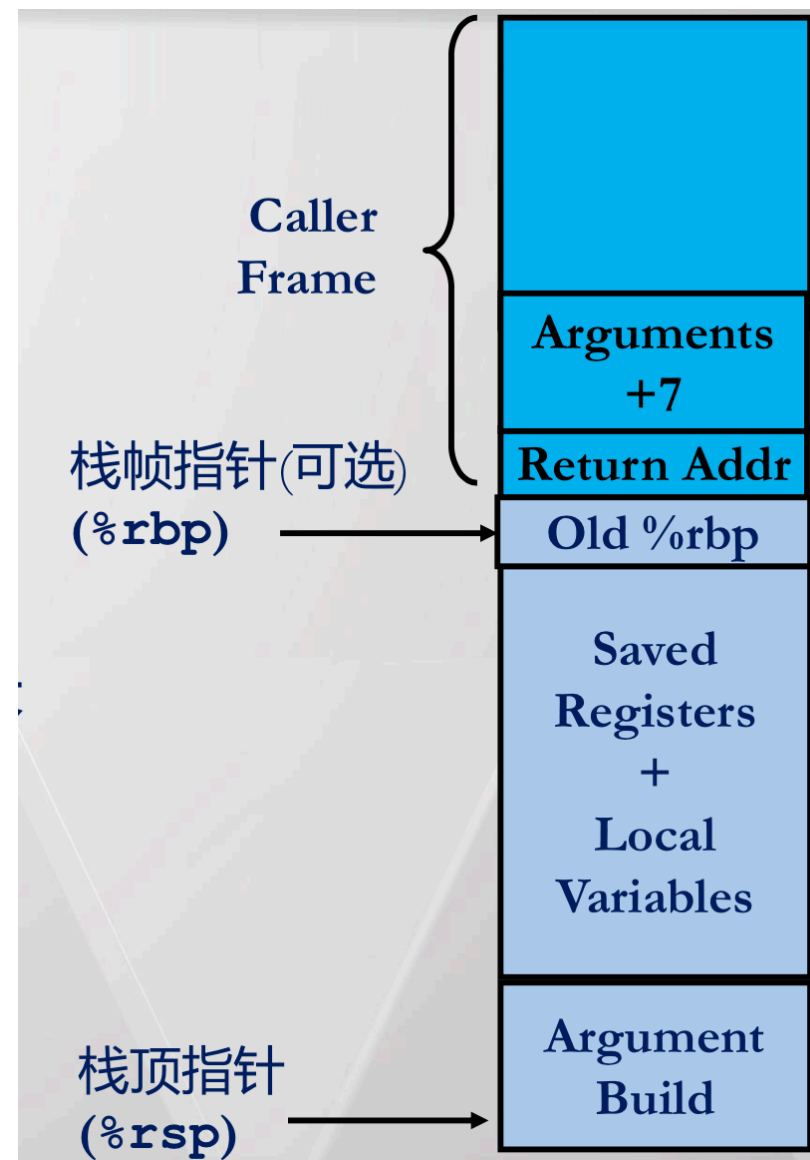
大于6个的参数使用栈传递，从右往左压栈

...
Arg <i>n</i>
...
Arg 8
Arg 7

- Only allocate stack space when needed

K. 栈, 参数传递 Cont'

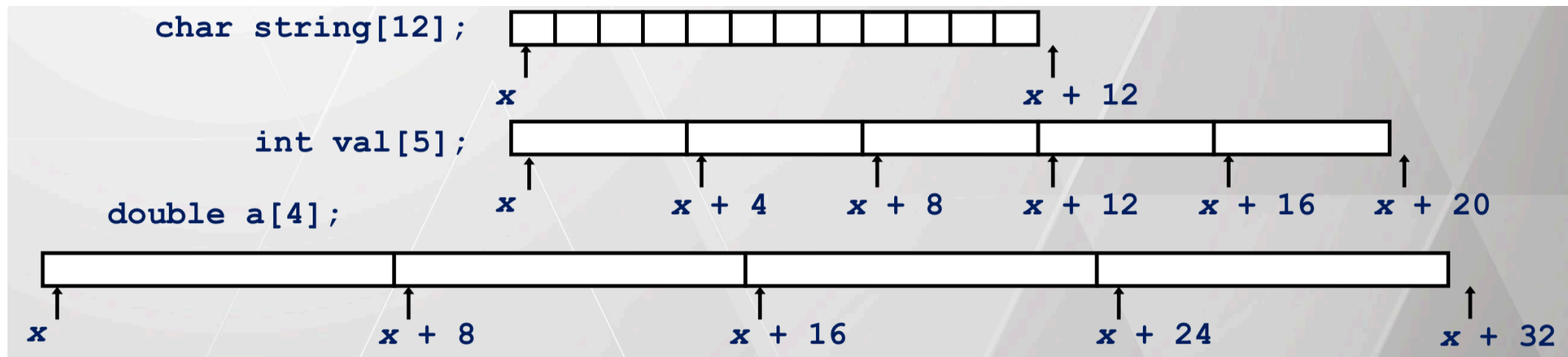
- **栈的使用情况**：栈帧的叠加
 - 调用参数
 - 返回地址
 - 寄存器保存值
 - 局部变量
 - ...
- **栈的方向**：往下减, 往上加
- **寄存器使用惯例**：
 - 被调用者保存 (子函数要压栈)
 - 调用者保存 (父过程保存)
- **32位和64位区别**：参数全压栈, +4/+8



L. 数组与结构

- 数组

- 连续存储
- 如何访问数组位置中的某个值（通过循环变量和leaq指令）



L. 数组与结构 Cont'

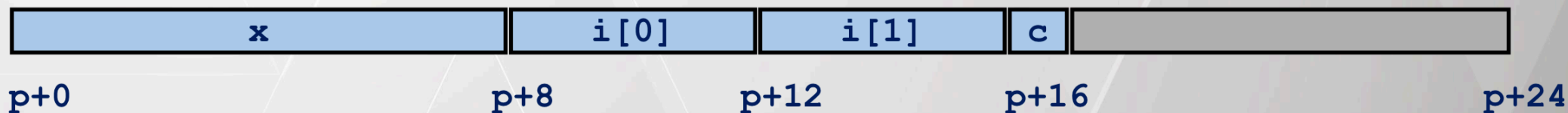
- 结构

- 连续分配内存区域
- 存储位置对齐
 - 每个元素有自己的对齐要求（ 64位：char:1, int:4, float:4, double:8, long int:8 ）
 - *** 特例：linux x86-32 double为4位 ***
 - 结构整体的对齐要求取最大的

L. 数组与结构 Cont'

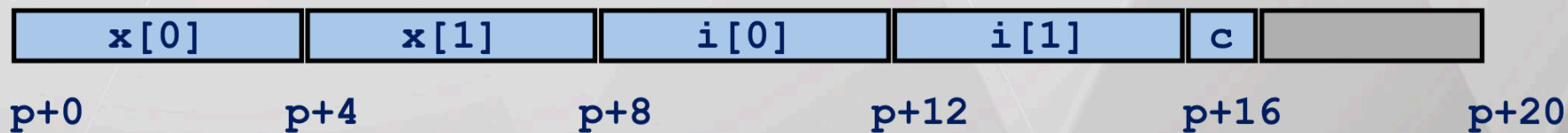
```
struct S2 {  
    double x;  
    int i[2];  
    char c;  
} *p;
```

p必须是8的整数倍:



```
struct S3 {  
    float x[2];  
    int i[2];  
    char c;  
} *p;
```

p必须是4的整数倍



L. 数组与结构 Cont'

1) (4 分) 已知一个 c 语言结构类型的定义如下, 请问在 X86 32 位 Linux 系统下变量 p 所占的空间大小是多少, 对齐的要求如何?

(2 分) + (2 分)

24 字节; 4 对齐。

```
struct S1 {  
    char c;  
    int i[2];  
    my_struct v;  
} p;
```

```
typedef TagStruct {  
    int k[2];  
    char c2;  
} my_struct;
```

M. MIPS指令集特点

- **流水线结构**：Fetch, Read, ALU, Memory, Write
- **指令集特点**（ 超多 ）
 - 指令长度（ 定长 ）
 - 指令中的内存操作数（ 没有 ）
 - 计算指令操作数（ 3个 ）
 - 寄存器个数（ 32个 ）
 - 条件码（ 无 ）
 - 访存操作（ load, store ）
 - 半字或者字节操作（ 无 ）
 - 针对栈的操作（ 无 ）
- **Branch Delay Slot**：跳转延迟槽（ 用于优化流水线效果 ）

N. MIPS汇编指令

- `addu $1, $2, $3` ($\$1 = \$2 + \$3$)
 - 与AT&T风格相反，目标寄存器在左侧
- `jal printf #` 函数跳转指令
- `nop #` 用于填充延迟槽，与上一条指令同时执行
- `lw $1, offset($2) #` 访存指令
 - 支持load不同的类型（b, h, w, d），以及带符号/不带符号扩展

N. MIPS汇编指令 Cont'

寄存器编号	名字	习惯用途
0	zero	值永远是0
1	at	保留给汇编器使用（编程时避免使用）
2-3	v0,v1	过程调用返回值
4-7	a0-a3	传参用寄存器
8-15	t0-t7	调用者保存寄存器
24, 25	t8,t9	
16-23	s0-s7	被调用者保存寄存器
26, 27	k0,k1	保留给异常处理使用
28	gp	全局指针(Global Pointer)——因为MIPS指令的立即数域宽度有限，gp寄存器可以作为基址寄存器进行load/store寻址
29	sp	栈顶寄存器
30	fp	栈帧寄存器（frame pointer）
31	ra	保存过程调用的返回地址

MIPS32寄存器命名与使用惯例

N. MIPS汇编指令 Cont'

- 算术

- ADDU 无符号整数相加
- ADDIU 无符号整数相加 (立即数)

- 分支

- BEQ branch on equal
- J jump

- 访存

- SW 保存一个word
- LL/SC 加锁的访存

- 常用

- LUI 加载上16位的立即数
- ORI 或上一个16位的立即数

N. MIPS汇编指令 Cont'

li	\$3, -5	⇒	addiu	\$3, \$0, -5
li	\$4, 0x8000	⇒	ori	\$4, \$0, 0x8000
li	\$5, 0x120000	⇒	lui	\$5, 0x12
li	\$6, 0x12345	⇒	lui	\$6, 0x1
			ori	\$6, \$6, 0x2345

lw	\$2, (\$3)	⇒	lw	\$2, 0(\$3)
lw	\$2, 8+4(\$3)	⇒	lw	\$2, 12(\$3)
lw	\$2, addr	⇒	lui	at, %hi(addr)
			lw	\$2, %lo(addr)(at)
sw	\$2, addr(\$3)	⇒	lui	at, %hi(addr)
			addu	at, at, \$3
			sw	\$2, %lo(addr)(at)

MIPS指令转换

O. 异常处理

- **异常**：打断程序正常的运行流程，且会引起运行态改变的事件
 - 用户态 - 内核态 - 用户态
- **同步异常**
 - Traps：故意引起的，返回到下一条指令
 - Faults：无意引起的，可恢复，重新执行当前指令
 - Aborts：无意引起的，不可恢复，程序退出
- **异步异常**（中断）
 - 由外部事件引起，往往是外设触发，重新执行或下一条指令
 - IO中断, Hard Reset, Soft Reset

O. 异常处理 Cont

- 异常处理基本过程

- 保存现场
- 判断不同的异常（ 查询cause寄存器 ）
- 构造异常处理内存空间（ 保留通用寄存器、堆栈存储区 ）
- 处理
- 返回（ 恢复寄存器， 清零cause， 将status相关位“置1”以中断 ）

- 异常返回

- 核心态 → 用户态
- Eret指令，“status转换”和“指令返回”同时完成

O. 异常处理 Cont'

4) 请分别举出一个同步异常、异步异常的例子；以及举出两种情况，在这些情况下异常返回不是回到 EPC 记录的地址（5 分）。

- 同步异常

- Traps：故意引起的，返回到下一条指令【断点，系统调用】
- Faults：无意引起的，可恢复，重新执行当前指令【页缺失】
- Aborts：无意引起的，不可恢复，程序退出【parity error, machine check】

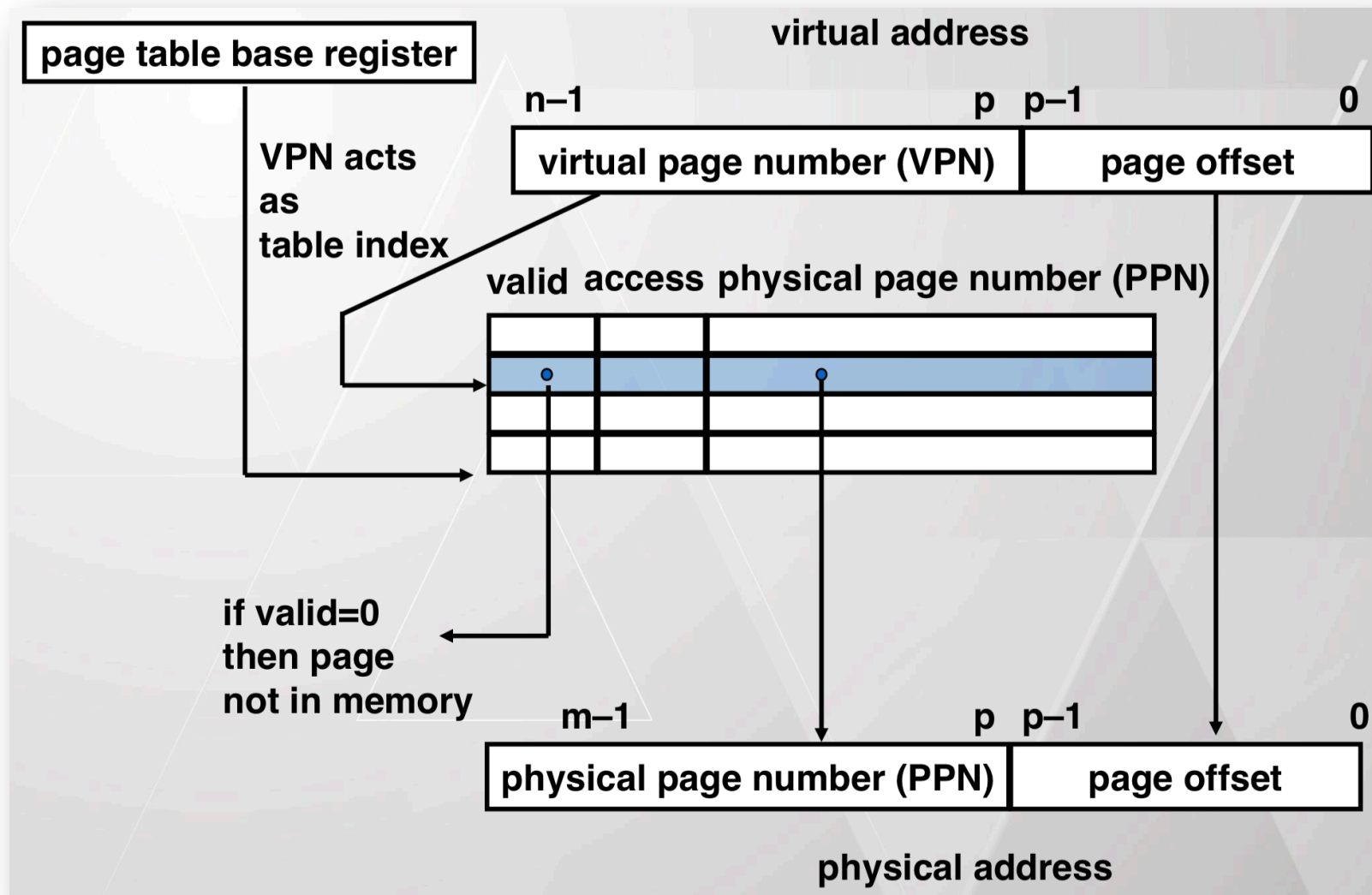
- 异步异常（中断）

- 由外部事件引起，往往是外设触发，重新执行或下一条指令
- 【IO中断，Hard Reset, Soft Reset】

P. 虚存与MIPS32内存管理

- **虚存的作用**（ 3点 ）
 - 1. 将内存memory作为硬盘disk的缓存
 - 2. 简化程序的内存管理， 是多个进程同时进行都拥有自己的空间
 - 3. 存储保护功能， 控制不同的访问权限
- **异常**：页缺失(Page Fault)
- **虚拟地址与实际物理地址的转换**
- **TLB快表**

P. 虚存与MIPS32内存管理 Cont'

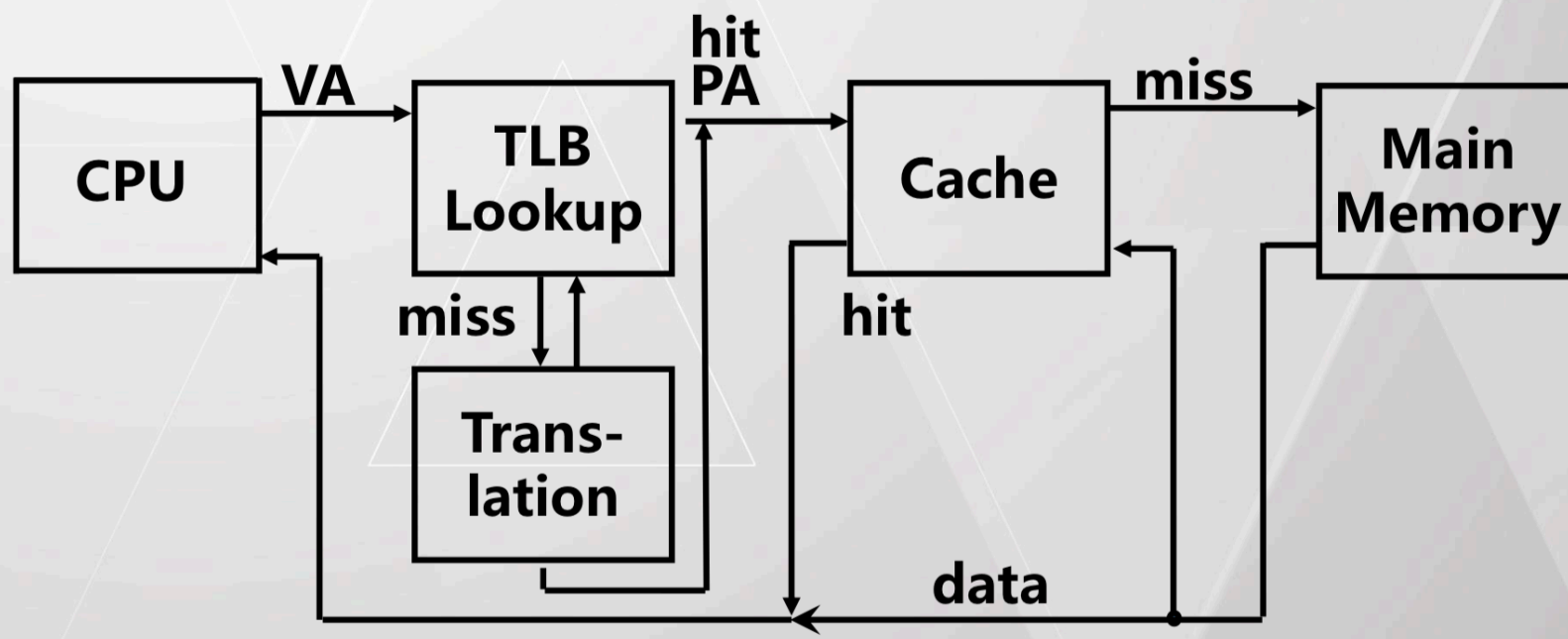


虚拟地址与
实际地址的
转换

P. 虚存与MIPS32内存管理 Cont'

“Translation Lookaside Buffer” (TLB)

- 处理器内存管理单元中的硬件部件
功能是将虚存页地址转换为物理内存地址
- 存储了内存中页表的一个子集



TLB快表
工作流程

P. 虚存与MIPS32内存管理 Cont'

任何的处理器地址的访问（包括指令与数据）都需要经过MMU（内存管理单元）的地址转换，即程序（虚）地址→物理地址

有一些嵌入式处理器没有MMU，但是一般也经过地址转换(fixed mapping)

用户态\核心态\调试态*

用户态下某些指令是非法的，且访存地址有限(0x8000 0000以下)

Kuseg——用户态可以使用的地址，需经过MMU转换

Kseg0——最高位清零后就是物理地址
Cacheable

Kseg1——最高三位清零后就是物理地址
Uncached；启动后可使用，系统的启动地址就在这段

Kseg2——核心态使用，需经过MMU转换

0xC000 0000

0xA000 0000

0x8000 0000

0x0000 0000

Mapped(kseg2)

1 GB

Unmapped uncached(kseg1)

512 MB

Unmapped cached(kseg0)

512 MB

32-bit user space
kuseg

2 GB

P. 虚存与MIPS32内存管理

- 虚存设计中，内存管理单元（MMU）的功能是将 虚拟地址 转换为 物理地址。在这一过程中，非映射段 地址保持不变。