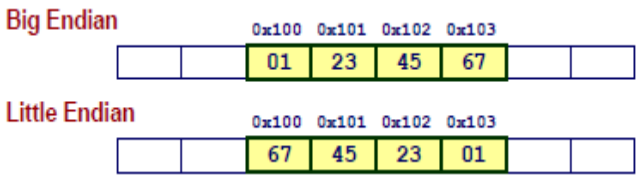
C Data Type	Intel IA32	x86-64
char	1	1
short	2	2
int	4	4
long	4	8
long long	8	8
float	4	4
double	8	8
long double	10/12	10/16
pointer	4	8

0x1234567



- Big Endian: Sun, PPC Mac, Internet
  - Least significant byte has highest address
- Little Endian: x86
  - Least significant byte has lowest address

float $(-1)^s M\ 2^E$

Special Values

- **Condition: exp** = 111…1
- **Case: exp** = 111…1, **frac** = 000…0
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞

"Normalized" Values

- **When: exp ≠ 000…0 and exp ≠ 111…1**
- **Exponent coded as a *biased* value: *E* = *Exp* – *Bias***
  - *Exp*: unsigned value exp
  - *Bias* = 2^{k-1} - 1, where *k* is number of exponent bits
    - Single precision: 127 (Exp: 1…254, E: -126…127)
    - Double precision: 1023 (Exp: 1…2046, E: -1022…1023)

Denormalized Values

- **Condition:** exp = 000…0
- **Exponent value: *E* = 1 – *Bias***
  - (instead of *E* = 0 – *Bias*)
- **Significand coded with implied leading 0: *M* = 0.xxx…x_2**
  - **xxx…x**: bits of **frac**

z

addl	Src,Dest	Dest = Dest + Src
subl	Src,Dest	Dest = Dest – Src
imull	Src,Dest	Dest = Dest * Src
sall	Src,Dest	Dest = Dest << Src
sarl	Src,Dest	Dest = Dest >> Src
shrl	Src,Dest	Dest = Dest >> Src
xorl	Src,Dest	Dest = Dest ^ Src
andl	Src,Dest	Dest = Dest & Src
orl	Src,Dest	Dest = Dest | Src

incl	Dest	Dest = Dest + 1
decl	Dest	Dest = Dest – 1
negl	Dest	Dest = – Dest
notl	Dest	Dest = ~Dest

ja	Above (unsigned)
jb	Below (unsigned)

CF	Carry Flag (for unsigned)	SF	Sign Flag (for signed)
ZF	Zero Flag	OF	Overflow Flag (for signed)

- ZF set when a&b == 0
- SF set when a&b < 0

- **testl/testq** *Src2, Src1*

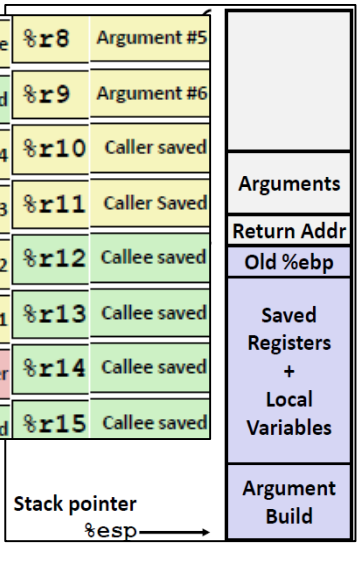
  testl b,a like computing **a&b** without setting destination

- **cmpl/cmpq** *Src2, Src1*
- **cmpl b,a** like computing **a-b** without setting destination

  看 a 是否？b
- CF set if carry out from most significant bit (used for unsigned comp
- ZF set if **a == b**
- SF set if **(a-b) < 0** (as signed)
- OF set if two's-complement (signed) overflow

```
(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)
```

%rax	Return value	%r8	Argument #5
%rbx	Callee saved	%r9	Argument #6
%rcx	Argument #4	%r10	Caller saved
%rdx	Argument #3	%r11	Caller Saved
%rsi	Argument #2	%r12	Callee saved
%rdi	Argument #1	%r13	Callee saved
%rsp	Stack pointer	%r14	Callee saved
%rbp	Callee saved	%r15	Callee saved



Program symbols are either strong or weak

- ***Strong***: procedures and initialized globals
- ***Weak***: uninitialized globals

不能有多个 srtong,挑 strong 不挑 weak,多个 weak 随机挑

Specific Cases of Alignment (IA32)

- 1 byte: `char`, ...
  - no restrictions on address
- 2 bytes: `short`, ...
  - lowest 1 bit of address must be 0_2
- 4 bytes: `int`, `float`, `char *`, ...
  - lowest 2 bits of address must be 00_2
- 8 bytes: `double`, ...
  - Windows (and most other OS's & instruction sets):
    - lowest 3 bits of address must be 000_2
  - Linux:
    - lowest 2 bits of address must be 00_2
    - i.e., treated the same as a 4-byte primitive data type
- 12 bytes: `long double`
  - Windows, Linux:
    - lowest 2 bits of address must be 00_2

Specific Cases of Alignment (x86-64)

- 1 byte: `char`, ...
  - no restrictions on address
- 2 bytes: `short`, ...
  - lowest 1 bit of address must be 0_2
- 4 bytes: `int`, `float`, ...
  - lowest 2 bits of address must be 00_2
- 8 bytes: `double`, `char *`, ...
  - Windows & Linux:
    - lowest 3 bits of address must be 000_2
- 16 bytes: `long double`
  - Linux:
    - Lowest 3 bits of address must be 000_2

Satisfying Alignment with Structures

- **Within structure:**
  - Must satisfy each element's alignment requirement
- **Overall structure placement**
  - Each structure has alignment requirement **K**
    - **K** = Largest alignment of any element
  - Initial address & structure length must be multiples of **K**

```
struc
  cha
  int
  dou
} *p;
```

Global symbols

- Symbols defined by module *m* that can be referenced by other modules.
- E.g.: non-`static` C functions and non-`static` global variables.

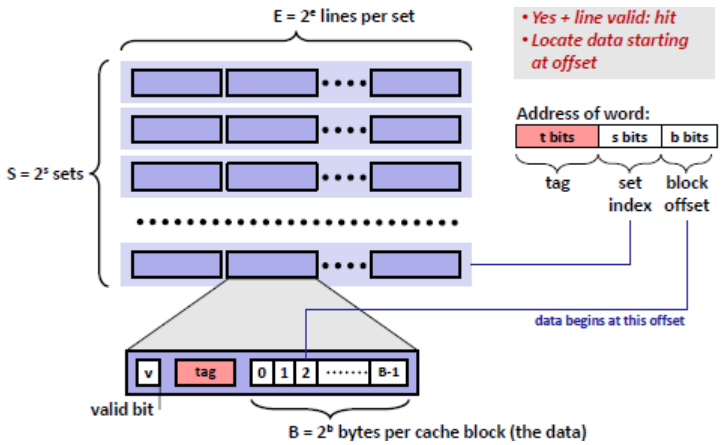
External symbols

- Global symbols that are referenced by module *m* but defined by some other module.

Local symbols

- Symbols that are defined and referenced exclusively by module *m*.
- E.g.: C functions and variables defined with the `static` attribute.
- **Local linker symbols are *not* local program variables**

Local program var> static local var> global var

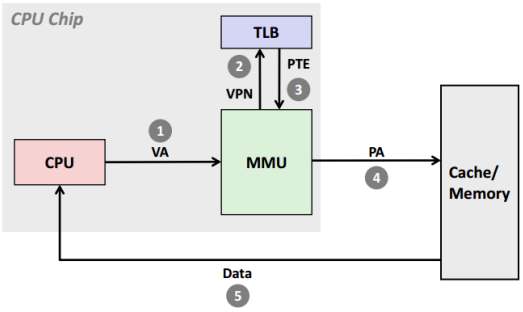


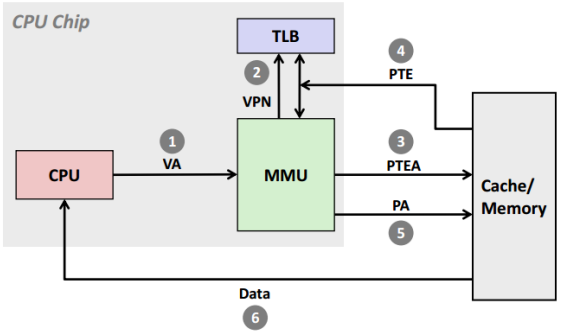
Readers:

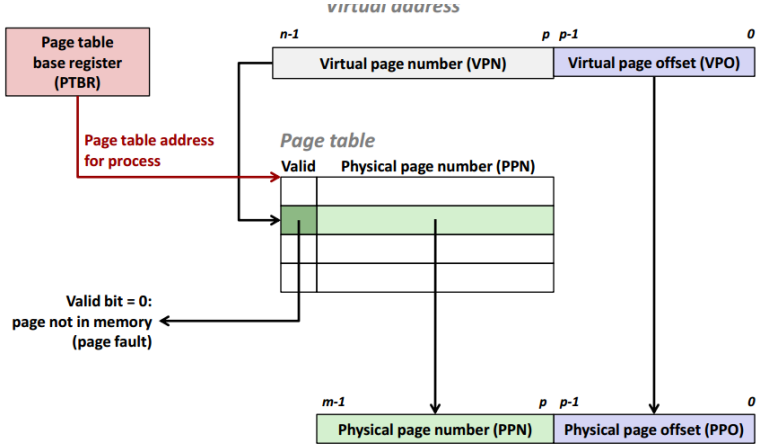
```
int readcnt;    /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
  while (1) {
    P(&mutex);
    readcnt++;
    if (readcnt == 1) /* First in */
      P(&w);
    V(&mutex);

    /* Reading happens here */

    P(&mutex);
    readcnt--;
    if (readcnt == 0) /* Last out */
      V(&w);
    V(&mutex);
  }
}
```

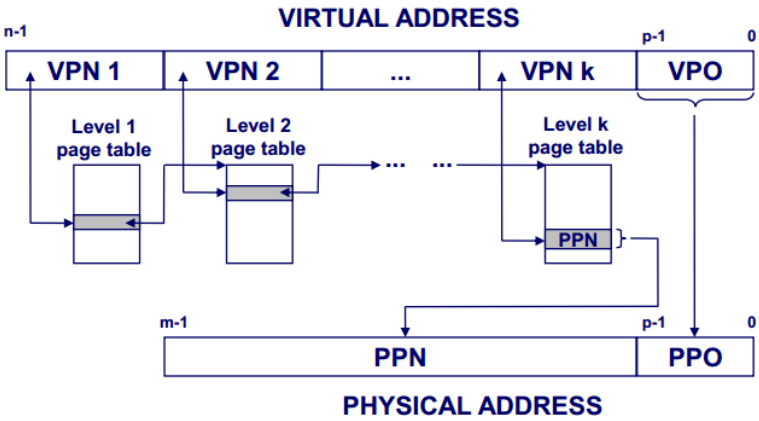
Writers:

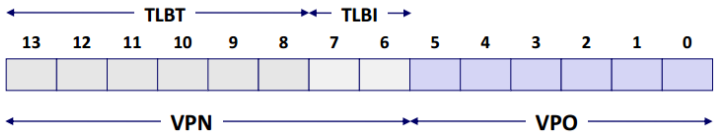
```
void writer(void)
{
  while (1) {
    P(&w);

    /* Writing here */

    V(&w);
  }
}
```











Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0