

CS680 Linux Kernel Programming, Spring 2017

Adrew Sohn

Computer Science Department

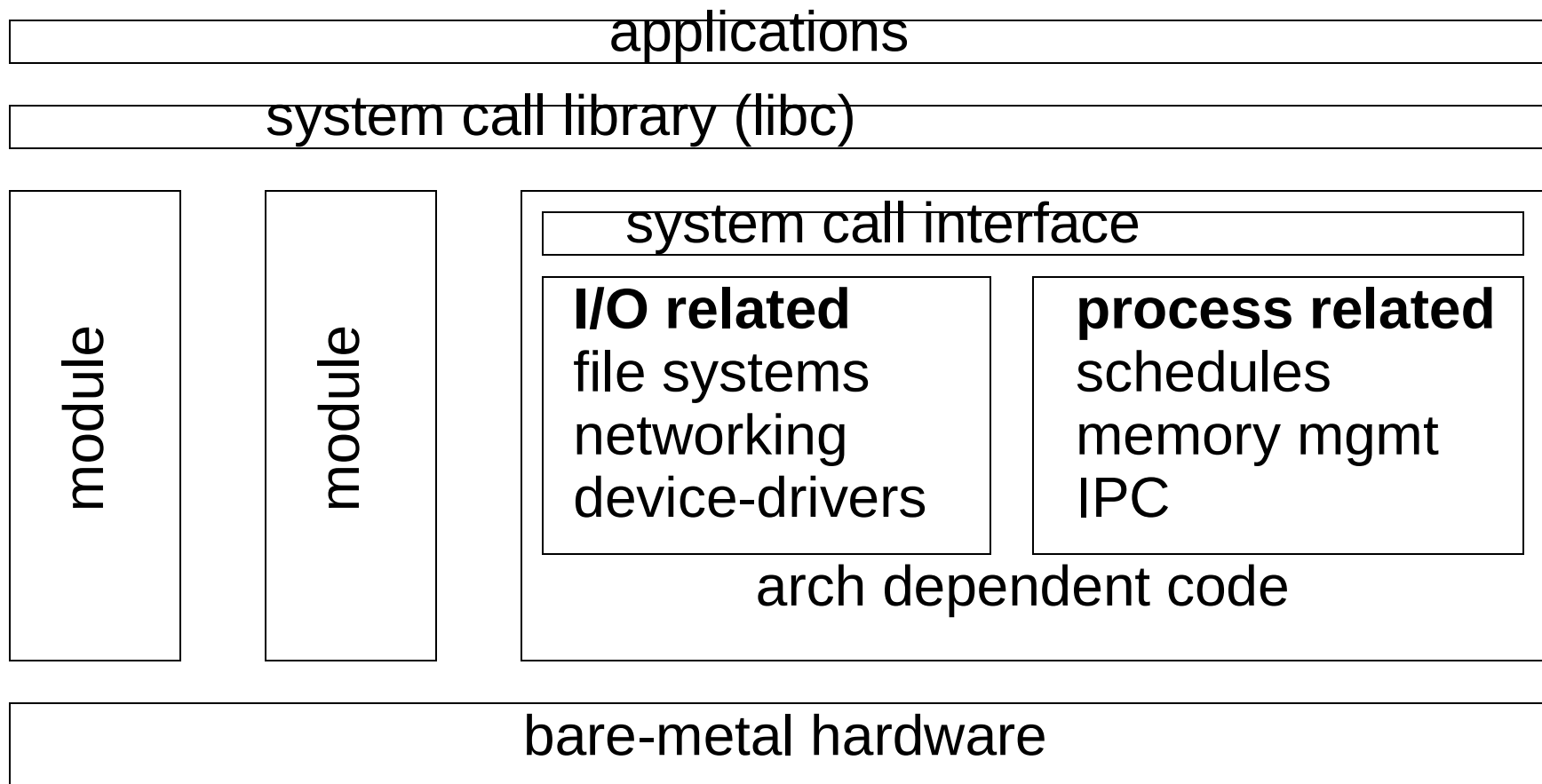
New Jersey Institute of Technology

kernel version 4.10 as of 1/16/2017

1. LAMP, virtualization, datacenter computing infrastructure, Review of Intel architecture based on Intel 64 and IA-32 Architectures Software Developer's Manual (4684 pages!), Intel and AT&T Linux assembly, assembly in-line programming
2. Setting up LXR (Linux cross referencer) on your laptop and in-class demonstration, compiling the kernel, Module programming, Booting - machine BIOS, disk MBR, Grub Linux loader, preliminary setup (setup(), startup_32() 1-compressed and 2-decompressed)
3. Booting continues, Overview of kernel startup and initialization (start_kernel())
4. Memory - overview, BIOS e820 mem map, memblock, segmentation
5. Memory - mapping, paging (get_free_pages),
6. **Test 1: 10-11:30am, Tue, 2/14/2017 (week5)**
Memory - caching (kmallocc) and process address space (vmalloc)
7. Process - thread union, kernel stack, task and thread struct, PID0 (swapper)
8. Process - PID1 (init), PID2 (kthreadd), PID3 (softirqd), PID4 (migrationd)
9. Process - process scheduling schedule() and process switching context_switch()
Interrupts - exceptions do_trap() and hard interrupts do_IRQ()
10. **Test 2: 10-11:30am, Tue, 3/28/2017 (week10)**
Interrupts - soft interrupts do_softirq(), ksoftirqd, timer interrupts run_timer_softirq()
11. File system - virtual file system, registering, mounting, Block IO,
12. File system - IO scheduler, device driver, vfs_read(), Ext4 example
13. Networking - receiving packets: NIC, ISR, Softirq, IP, TCP, Inet, BSD, User, sys_send()
14. Networking - sending packets: User, BSD, Inet, TCP, IP, Softirq, Qdisc, ISR, NIC, sys_recv()
15. **Final exam (week15): See the registrar's page: <http://www.njit.edu/registrar>**

Chapter 1 Introduction to the Linux Kernel

1. LAMP, cloud computing, data center technologies, data analytics, big data, cognitive computing, etc.
2. Virtualization and virtual machines (kvm, xen, virtualbox, ...)
3. Intel Architecture
4. Assembly - AT&T (Motolora) and intel



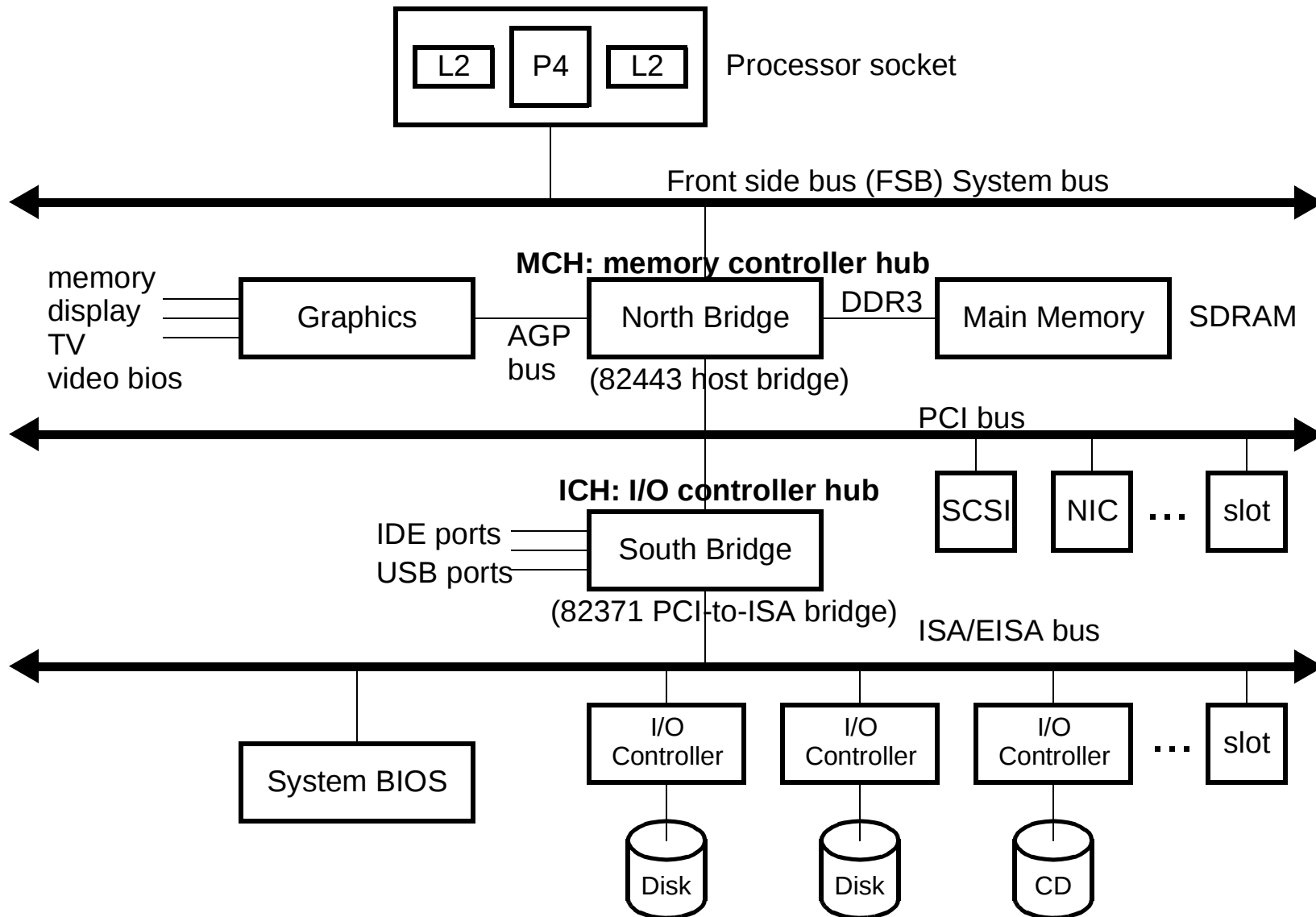
LAMP: Linux Apache my(maria)Sql PHP/Python

Have's	Have Nots
Closed	Open
\$\$\$	\$000
Microsoft	LAMP
Windows	Linux
IIS	Apache
SQL	My(Maria)SQL
.net	PHP/Python
GUI	CLI
IDE	emacs/vi
C#	C
powershell (win10 supports bash)	bash

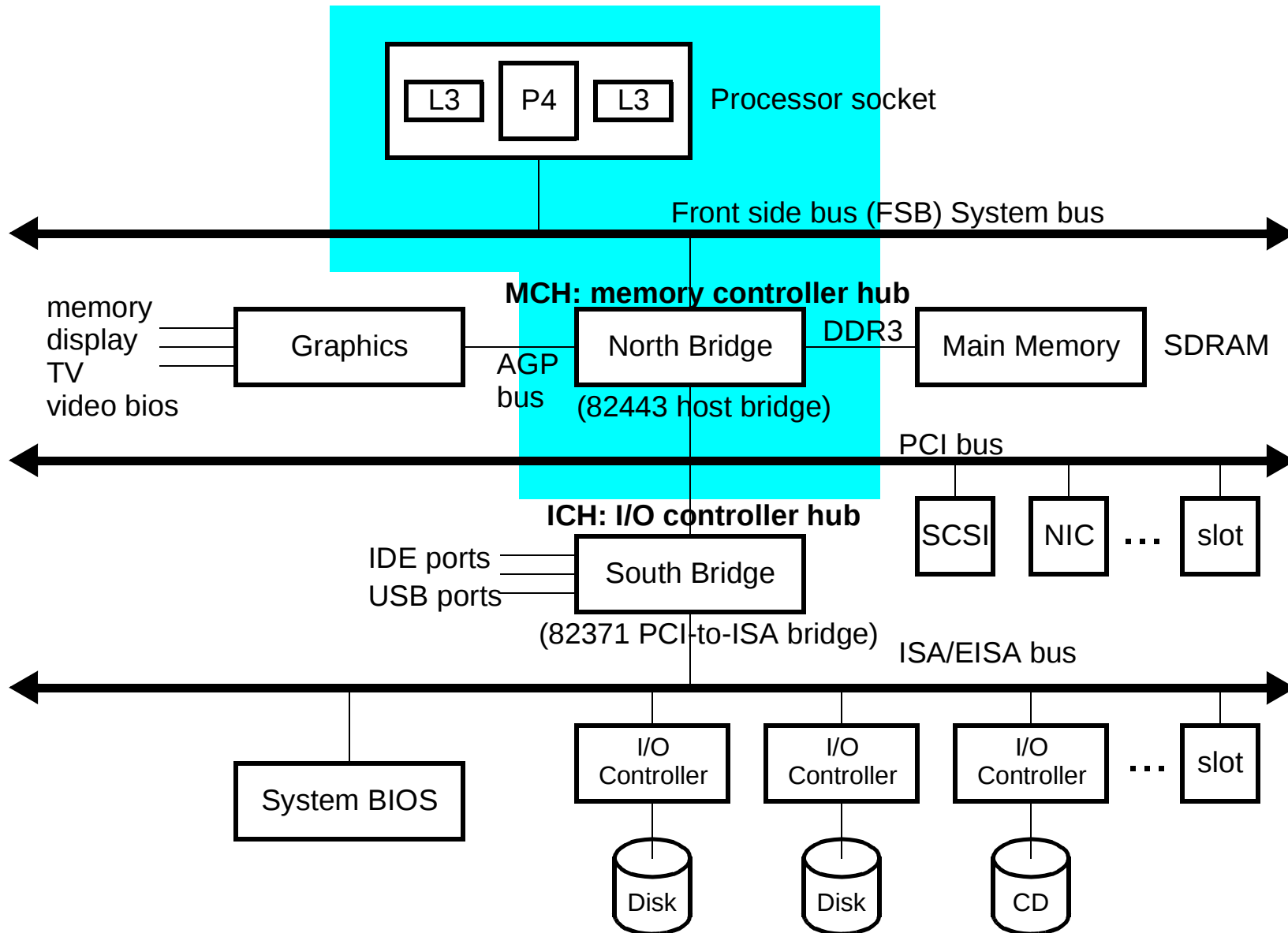
Virtualization: Hypervisor and Virtual Machines

- KVM - Kernel based VM, starting 2.6.20
- VMware (EMC) - vSphere Hypervisor, free version
- Xen (Citrix) - Express version free
- Microsoft - Hyper-V hypervisor
- Oracle (Sun) - Virtual Box
- UML - user mode linux
- BSD - bhybe “Bee Hyve”

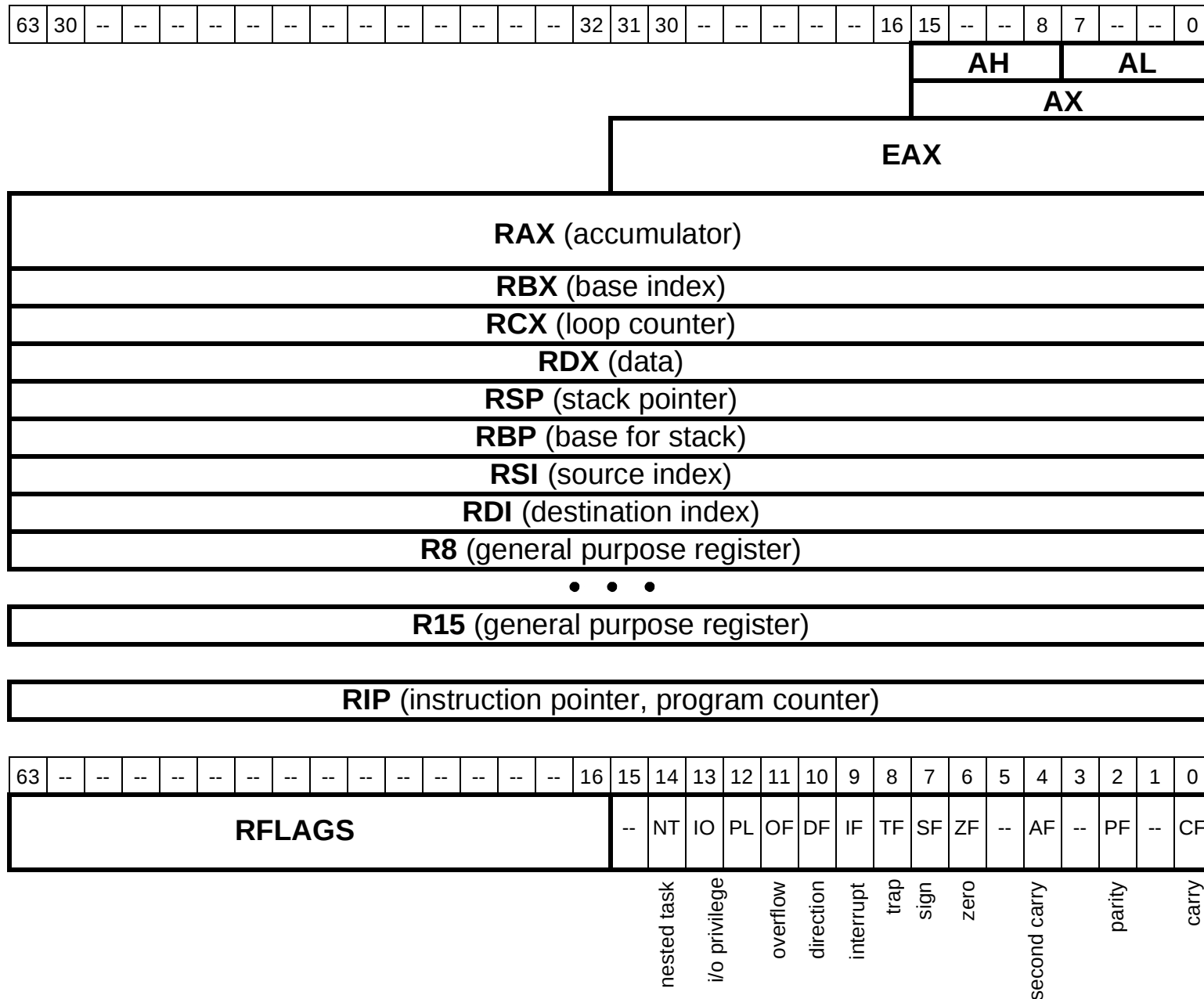
Old i386-based motherboard architecture



Modern motherboard - Platform Controller Hub (PCH)



General Purpose, Pointer, Index, Instruction Registers



Control registers

63	-	-	-	-	-	-	-	32	31	-	-	-	-	-	-	0
prcr status, operating mode, sys contrl, flags																CR0
CR1																CR1
CR2 - page fault linear address																CR2
CR3 - page directory base register																CR3
CR4																CR4
CR8 - task priority register																CR8
EFER (Extended Feature Enable Register)																EFER

IDTR interrupt descriptor table register

GDTR global descriptor table register

Segment registers

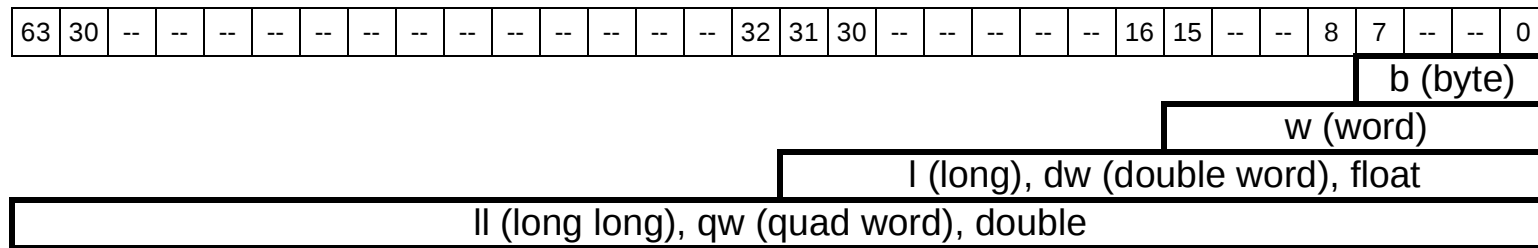
16 bits																
CS - code																CS
SS - stack																SS
DS - data																DS
ES - extra																ES
FS - extra																FS
GS - extra																GS

Status register

31	--	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
--	--	--	B	C3	ST	ST	ST	C2	C1	C0	IR	--	EU	PE	OE	ZE	DE	IE
			store the results of a boolean test on a float													zero divide error		invalid operation error
			busy	condition code	stack pointer	stack pointer	stack pointer	condition code	condition code	condition code	interrupt request		underflow	precision	overflow		divide error	
			counting how many registers are currently NOT in use. first initialize to 0, decrement on each push, increment on each pop.															

Linux Assembly Basics

Operator size:



Syntax:

	Linux/AT&T	Intel
register naming	%eax	eax
source/destination order	movl source, destination	mov destination, source
constant/immediate value	movl \$abc, %eax	mov eax, abc
hexa decimal	movl \$0xd12f, %eax	mov eax, d12fh
--	--	--

Examples - see <http://ibiblio.org> inline assembly how-to

Linux Assembly In-lining

Format

```
_asm__volatile_  
    asm instructions  
    :output  
    :input  
    :clobbered registers  
)
```

volatile - tell the compiler not to optimize

Constraints

- m: memory operand
- o: memory operand with offset
- r: register

Modifiers

- “=” write-only
- “&” early clobber operand which is modified before the instruction is finished using the input
- & for unique reg
- \$ - immediate value
- % - variable identifier
- %% - register name

Example - *test* (see IBM developerworks)

```

int test ()
{
    int x=10, y;
    printf("before: x=%x, y=%x",x,y);
    asm (
        "movl %1, %%eax;
         movl %%eax, %0;"
        : "=r" (y)          /* output */
        : "r" (x)           /* input */
        : "%eax"            /* clobbered register*/
    );
    printf("after: x=%x, y=%x",x,y);}

```

%% - to use register %eax inside "asm", %eax should be preceded by one more %, %%eax. %0 and %1 are to identify variables. Anything with single % will be treated as input/output operands and not as registers.

Example - *strcpy* (see ibiblio.org)

“Hello, World!” => “H=48 e=65 l=6C ... l=6C d=64 !=21”

```
static inline char *strcpy (char *dst, const *src)
{
    int tsrc, tdst;
    __asm__ __volatile__ (
        "1: \tlodsb\n\t"          ;(ESI) --> EAX
        "stosb\n\t"              ;EAX --> (EDI)
        "testb %%al, %%al\n\t"    ;AND-ing al & al, set Eflags ZF bit (bit6) to 0/1
        "jne 1b"                 ;jump to 1b if not equal to 1; b=backward,1=label
    :
    : "=&S" (tsrc),              ESI ;parameter 0    &=>early clobber
      "=&D" (tdst),              EDI ;parameter 1
      "0" (src),                 ;parameter 2
      "1" (dst),                 ;parameter 3
      : "memory"
    );
    return dst;
}
```

output : “=&S” (tsrc), ESI ;parameter 0 &=>early clobber
 “=&D” (tdst), EDI ;parameter 1

input : “0” (src), ;parameter 2
 “1” (dst), ;parameter 3
 : “memory”

d =	01100100
&	01100100
	01100100
not zero --> ZF bit=0	

null =	00000000
&	00000000
	00000000
zero --> ZF bit=1	

Linux Assembly

```
.global  _start

.text
_start:
    movl    $4, %eax        ;syscall number - sys_write
    movl    $1, %ebx        ;file descriptor - stdout; $0 stdin
    movl    $msg, %ecx       ;msg itself
    movl    $len, %edx       ;msg length
    int     $0x80           ;interrupt

    movl    $1, %eax        ;syscall number - sys_exit
    int     $0x80           ;interrupt

.data
msg:
    .ascii  "Hello, World and Welcome to the Linux Kernel!\n"
    len =   . - msg
```

hello.asm

Commands to generate an executable file:

1. `as -o hello.o hello.asm`
2. `ld -o hello hello.o`

Essentials for Module Programming

`module_init()`
`module_exit()`

`/sbin/insmod`
`/sbin/rmmod`
`/sbin/lsmmod`
`/sbin/modprobe`

messages at `/var/log/messages`

Hello World Module Programming (see tldp.org)

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
static int hello_data __initdata=3;

static int hello_init(void)
{
    printk(KERN_INFO "Loading Hello module - Hello, World!\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_INFO "Exiting Hello module - Goodbye, World!\n");
}

module_init(hello_init)
module_exit(hello_exit)
```