中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

How the Computer Works

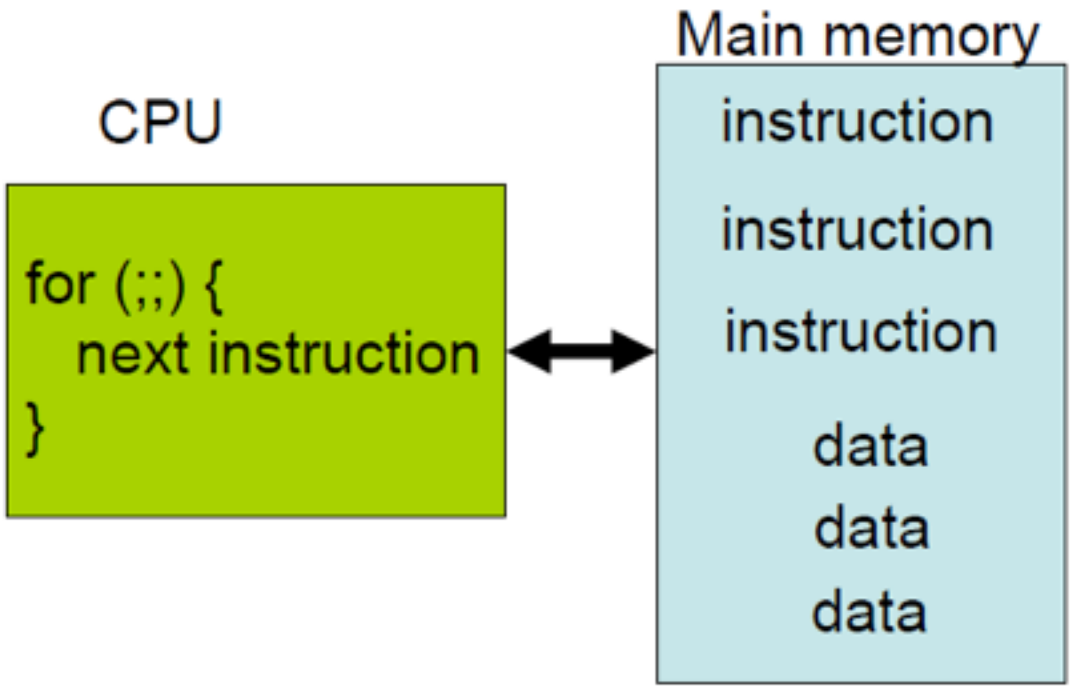
based on X86/Linux

孟宁
电话：0512-68839303
E-mail：mengning@ustc.edu.cn
主页：http://staff.ustc.edu.cn/~mengning
地址：苏州工业园区独墅湖高等教育区仁爱路188号507室

Agenda

- The stored program computer
- X86 implementation
- Registers and Memory
- Stack memory + operations
- Example Program
- From C to running program
- Homework

The stored program computer

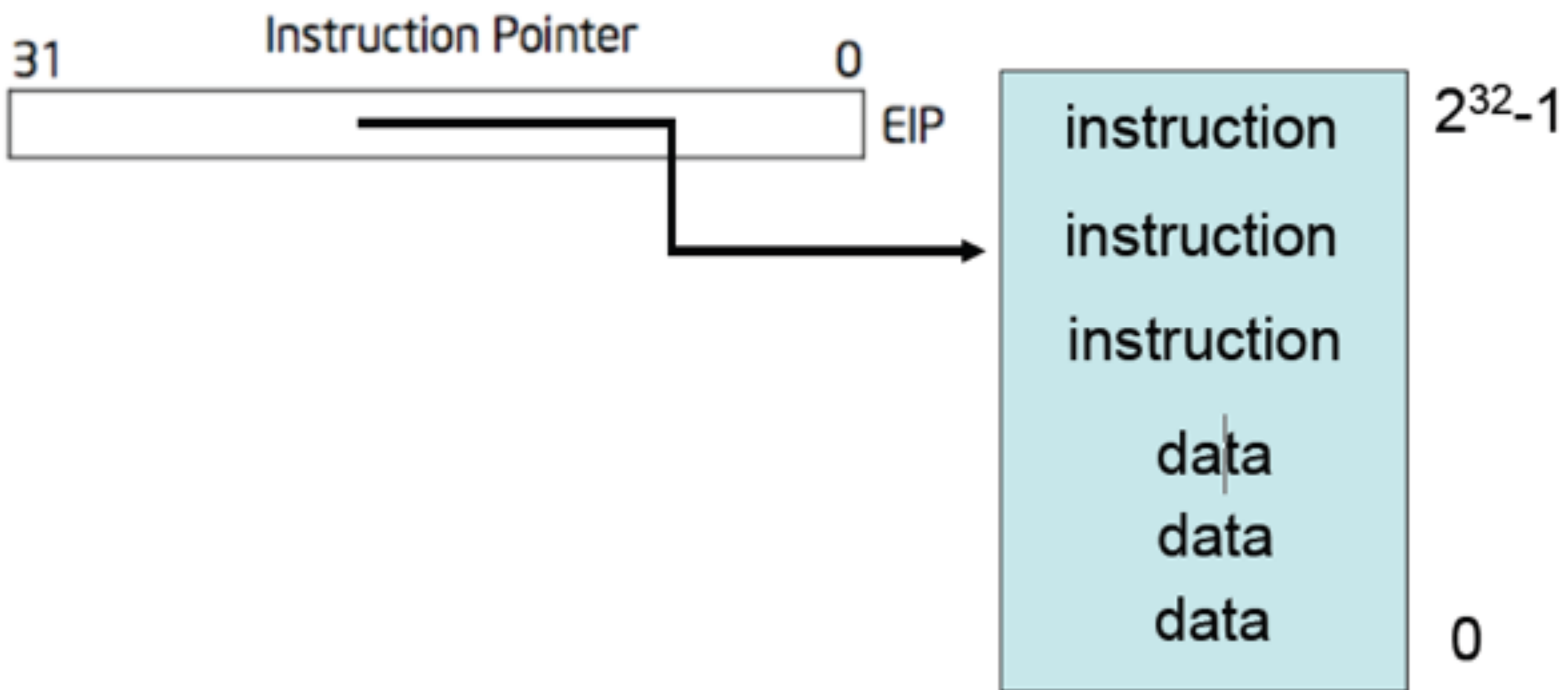


- Memory holds instructions and data
- CPU interpreter of instructions

ABI：Application Binary Interface

- Instructions encoding
- Registers convention in Instructions
- Most instructions can take a Memory address

X86 implementation



- EIP is incremented after each instruction
- Instructions are different length
- EIP modified by CALL, RET, JMP, and conditional JMP

Registers for work space

General-Purpose Registers

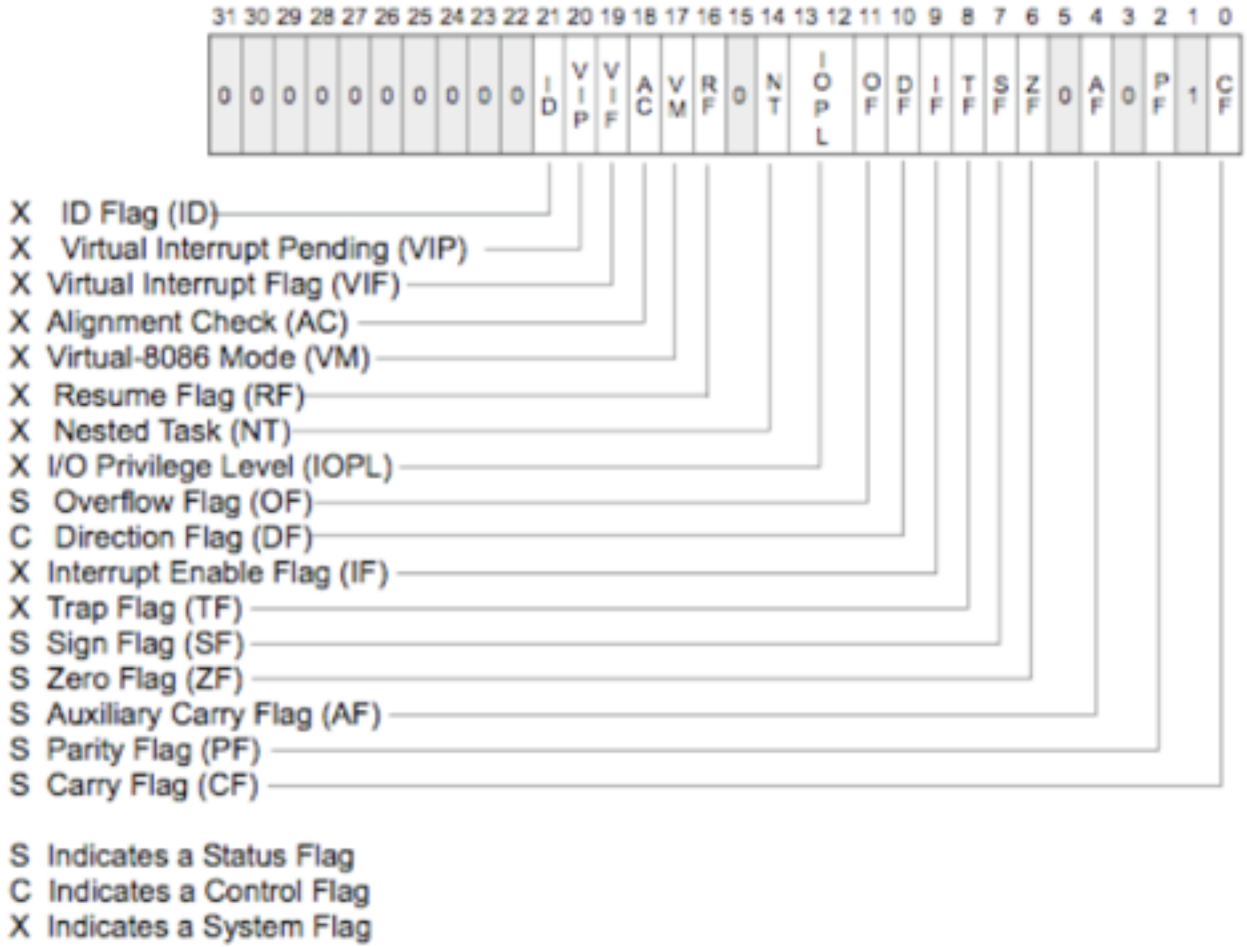
31 … 16	15 … 8	7 … 0	16-bit	32-bit	
	AH	AL	AX	EAX	累加器(Accumulator)
	BH	BL	BX	EBX	基地址寄存器(Base Register
	CH	CL	CX	ECX	计数寄存器(Count Register)
	DH	DL	DX	EDX	数据寄存器(Data Register)
	BP			EBP	堆栈基指针(Base Pointer)
	SI			ESI	变址寄存器(Index Register)
	DI			EDI	变址寄存器(Index Register)
	SP			ESP	堆栈顶指针(Stack Pointer)

- 8, 16, and 32 bit versions
- By convention some registers for special purposes
- Example: ADD EAX, 10
- Other instructions: SUB, AND, etc.

Segment Register

- CS——代码段寄存器(Code Segment Register)，其值为代码段的段值；
- DS——数据段寄存器(Data Segment Register)，其值为数据段的段值；
- ES——附加段寄存器(Extra Segment Register)，其值为附加数据段的段值；
- SS——堆栈段寄存器(Stack Segment Register)，其值为堆栈段的段值；
- FS——附加段寄存器(Extra Segment Register)，其值为附加数据段的段值；
- GS——附加段寄存器(Extra Segment Register)，其值为附加数据段的段值。

EFLAGS register



Memory: more work space

movl %eax, %edx	edx = eax;	*register mode*
movl $0x123, %edx	edx = 0x123;	*immediate*
movl 0x123, %edx	edx = *(int32_t*)0x123;	*direct*
movl (%ebx), %edx	edx = *(int32_t*)ebx;	*indirect*
movl 4(%ebx), %edx	edx = *(int32_t*)(ebx+4);	*displaced*

- Memory instructions: MOV, PUSH, POP, etc
- Most instructions can take a memory address
- b,w,l分别代表8位，16位和32位

Stack memory + operations

Example instruction	What it does
pushl %eax	subl $4, %esp movl %eax, (%esp)
popl %eax	movl (%esp), %eax addl $4, %esp
call 0x12345	pushl %eip (*) movl $0x12345, %eip (*)
ret	popl %eip (*)

enter	pushl %ebp movl %esp,%ebp
leave	movl %ebp,%esp popl %ebp

- Stack grows down
- Use to implement procedure calls

More memory

- 80386: 32 bit data and bus addresses
- Now: the transition to 64 bit addresses
- Backwards compatibility:
  - Boots in 16-bit mode, and switches to protected mode with 32-bit addresses
- 80386 also added virtual memory addresses
  - Segment registers are indices into a table
  - Page table hardware

练习

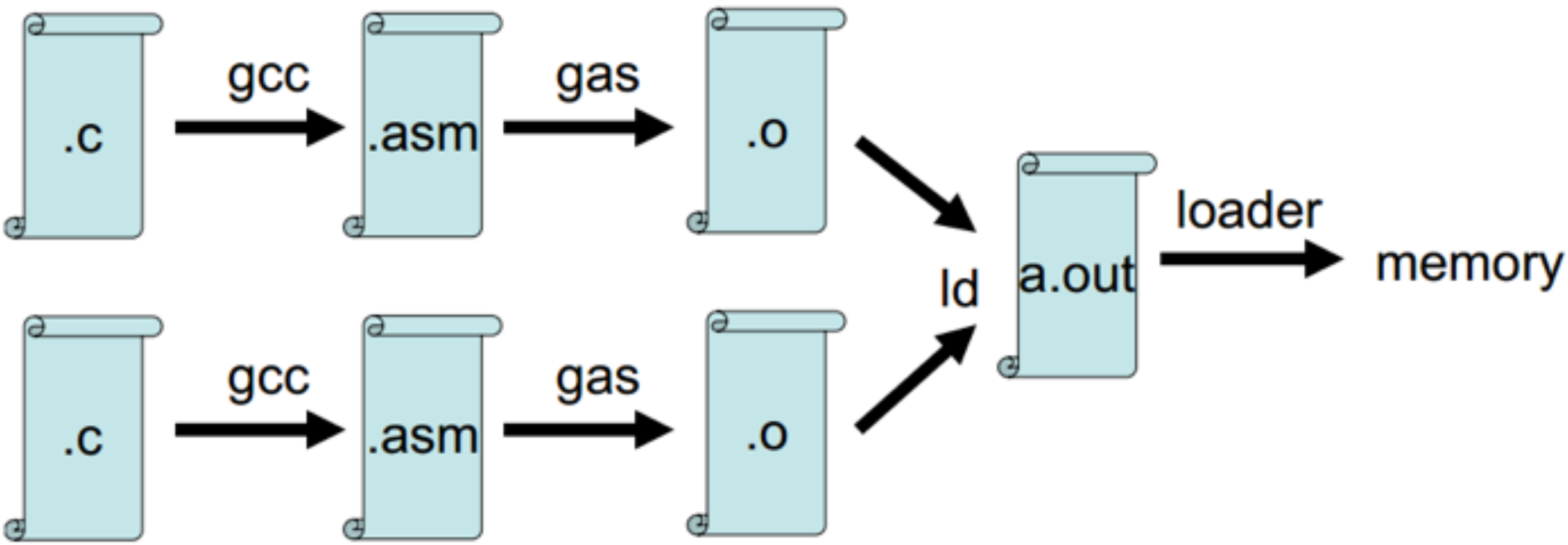
```
...
pushl    $8
movl     %esp, %ebp
subl     $4, %esp
movl     $8, (%esp)
...
```

```
...
pushl    $8
movl     %esp, %ebp
pushl    $8
...
```

```
...
pushl    $8
movl     %esp, %ebp
pushl    %esp
pushl    $8
addl     $4, %esp
popl     %esp
...
```

请画出当前堆栈状态，表明当前栈顶和栈基地址位置

From C to running program



Example

```c
int g(int x)
{
        return x+3;
}

int f(int x)
{
        return g(x);
}

int main(void)
{
        return f(8)+1;
}
```

```
...
g:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        addl    $3, %eax
        popl    %ebp
        ret
...
f:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $4, %esp
        movl    8(%ebp), %eax
        movl    %eax, (%esp)
        call    g
        movl    %ebp,%esp
        popl    %ebp
        ret
...
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $4, %esp
        movl    $8, (%esp)
        call    f
        addl    $1, %eax
        movl    %ebp,%esp
        popl    %ebp
        ret
...
```

练习

```
        ...
        g:
                pushl   %ebp
                movl    %esp, %ebp
                movl    8(%ebp), %eax
                addl    $8, %eax
                popl    %ebp
                ret
        ...
        main:
                pushl   %ebp
                movl    %esp, %ebp
                pushl   $8
                call    g
                subl    $8, %eax
                movl    %ebp,%esp
                popl    %ebp
                ret
        ...
```

它对应的C代码?

Summary

- 对能看得见的结构部分（例如，通用寄存器，控制寄存器，状态寄存器，中断或者例外寄存器）的使用方法，或者说编程模式，就是所谓的ABI。
- 掌握一个CPU的ABI，或者说编程界面，是一个基本功。是必须的。浅显说，就是那些寄存器的用法，分布和使用约定。
- 定义一个处理器的ABI，也是做编译器设计的第一个环节。笔者曾经设计过一个网络处理器的GCC的后端target。设计的第一个事情其实就是设计寄存器的约定。(by陈怀临)

思考

- 计算机是怎样工作的？[单任务]
- 计算机是怎样工作的？[多任务]

实验：计算机是怎样工作的?

- 实验：请使用Example的c代码分别生成.cpp,.s,.o和ELF可执行文件，并加载运行，分析.s汇编代码在CPU上的执行过程
- 要求：通过实验解释单任务计算机是怎样工作的，并在此基础上讨论分析多任务计算机是怎样工作的。
- gcc用法参考（*表示文件名）
  - gcc –E –o *.cpp *.c
  - gcc –x cpp-output –S –o *.s *.cpp
    - gcc –S –o *.s *.c
  - gcc –x assembler –c *.s -o *.o
    - gcc –c *.c -o *.o
    - as –o *.o *.s
  - gcc –o * *.o
    - gcc –o * *.c

“世上无难事

只要肯登攀”

--毛泽东

谢谢大家！

参考资料：
http://pdos.csail.mit.edu/6.828/2011/lec/l-x86.pdf
http://www.tektalk.org/2011/12/11