



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

---

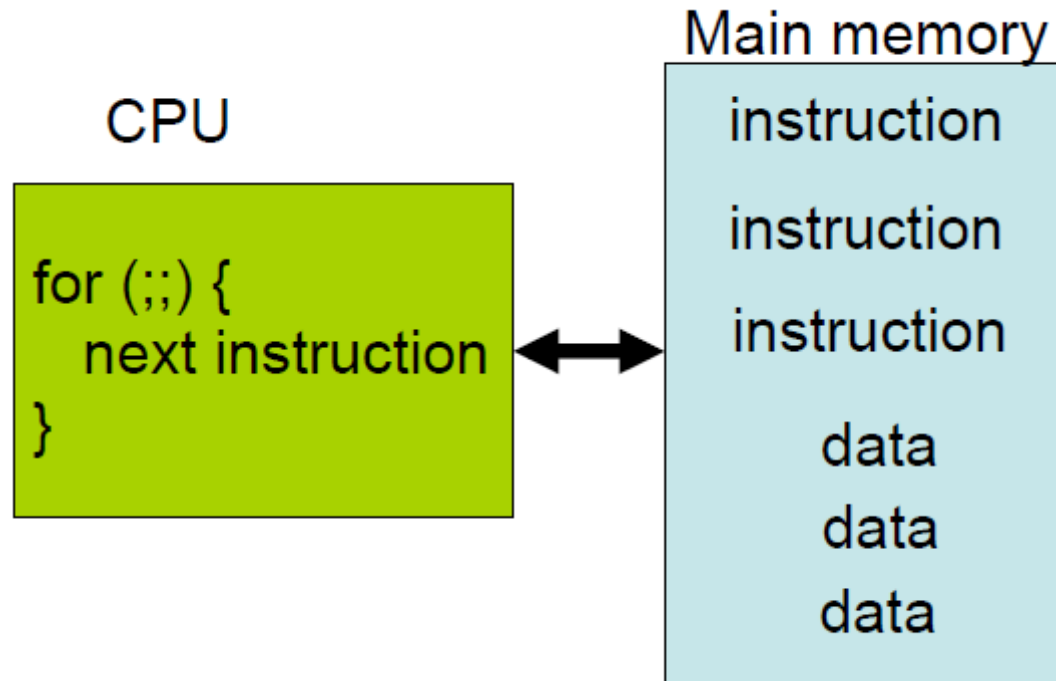
- ◆ The stored program computer
- ◆ X86 implementation
- ◆ Registers and Memory
- ◆ Stack memory + operations
- ◆ Example Program
- ◆ From C to running program
- ◆ Homework





# The stored program computer

---



- ◆ Memory holds instructions and data
- ◆ CPU interpreter of instructions



# ABI: Application Binary Interface

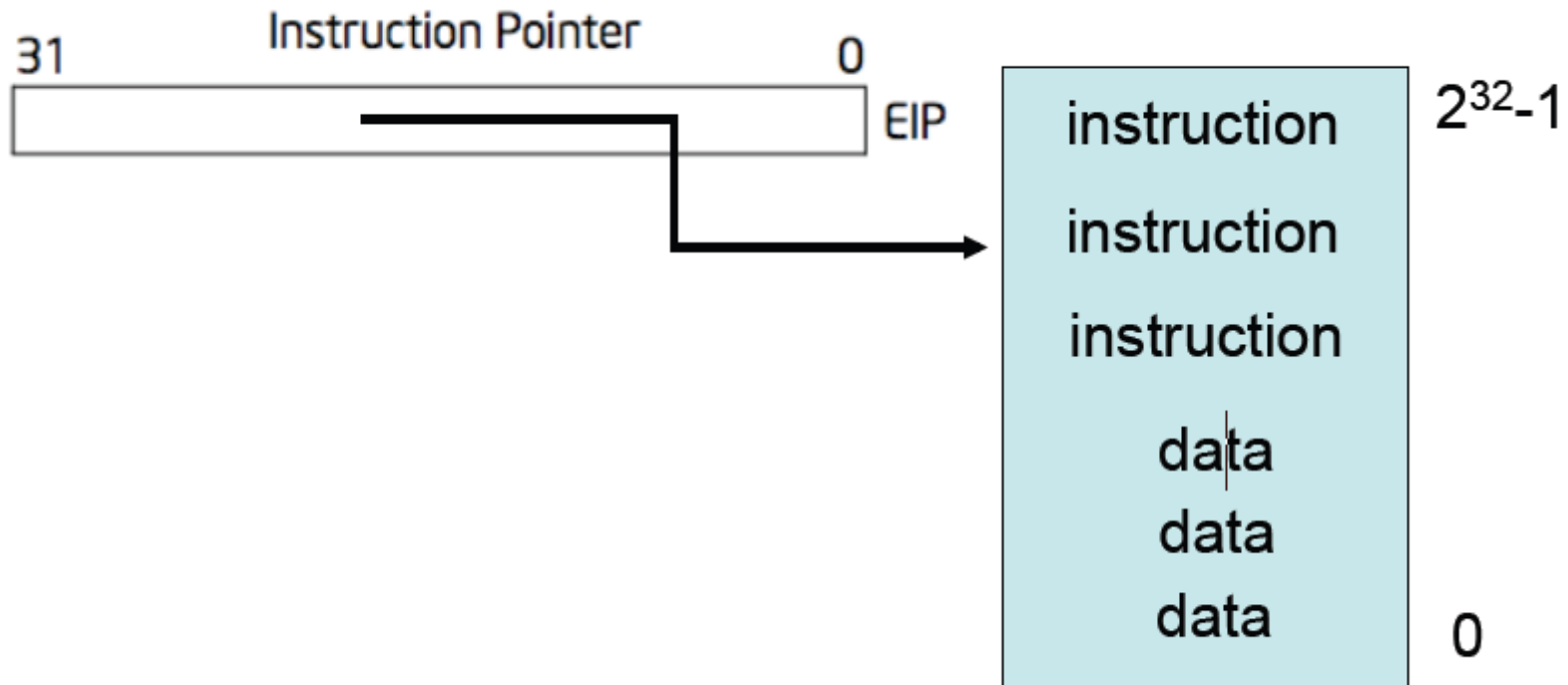
---

- ◆ 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
	AH	AL	
	BH	BL	
	CH	CL	
	DH	DL	
	BP		
	SI		
	DI		
	SP		

16-bit 32-bit

AX EAX 累加器(Accumulator)  
BX EBX 基地址寄存器(Base Register)  
CX ECX 计数寄存器(Count Register)  
DX EDX 数据寄存器(Data Register)  
EBP 堆栈基指针(Base Pointer)  
ESI 变址寄存器(Index Register)  
EDI  
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

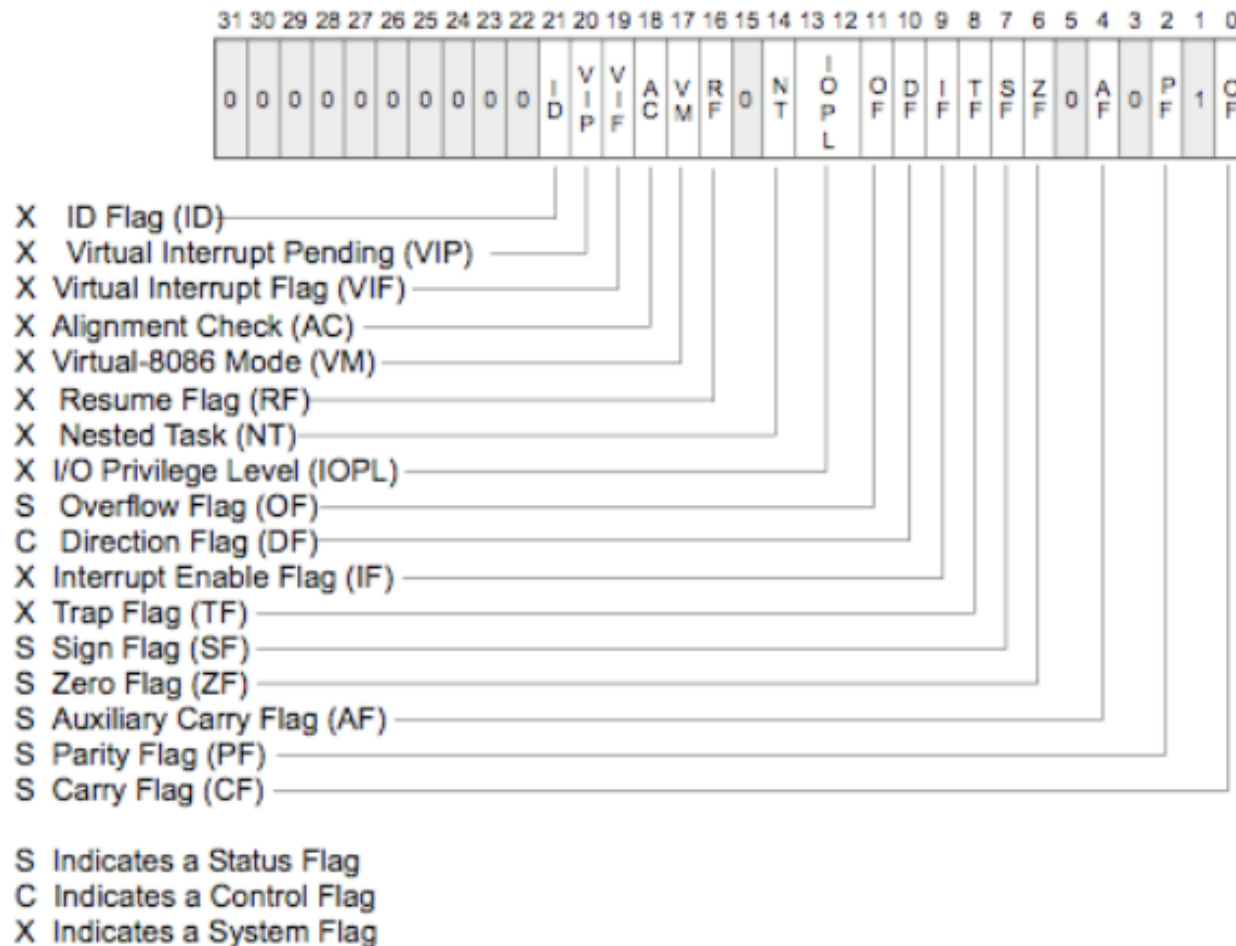
---

- ◆ **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。





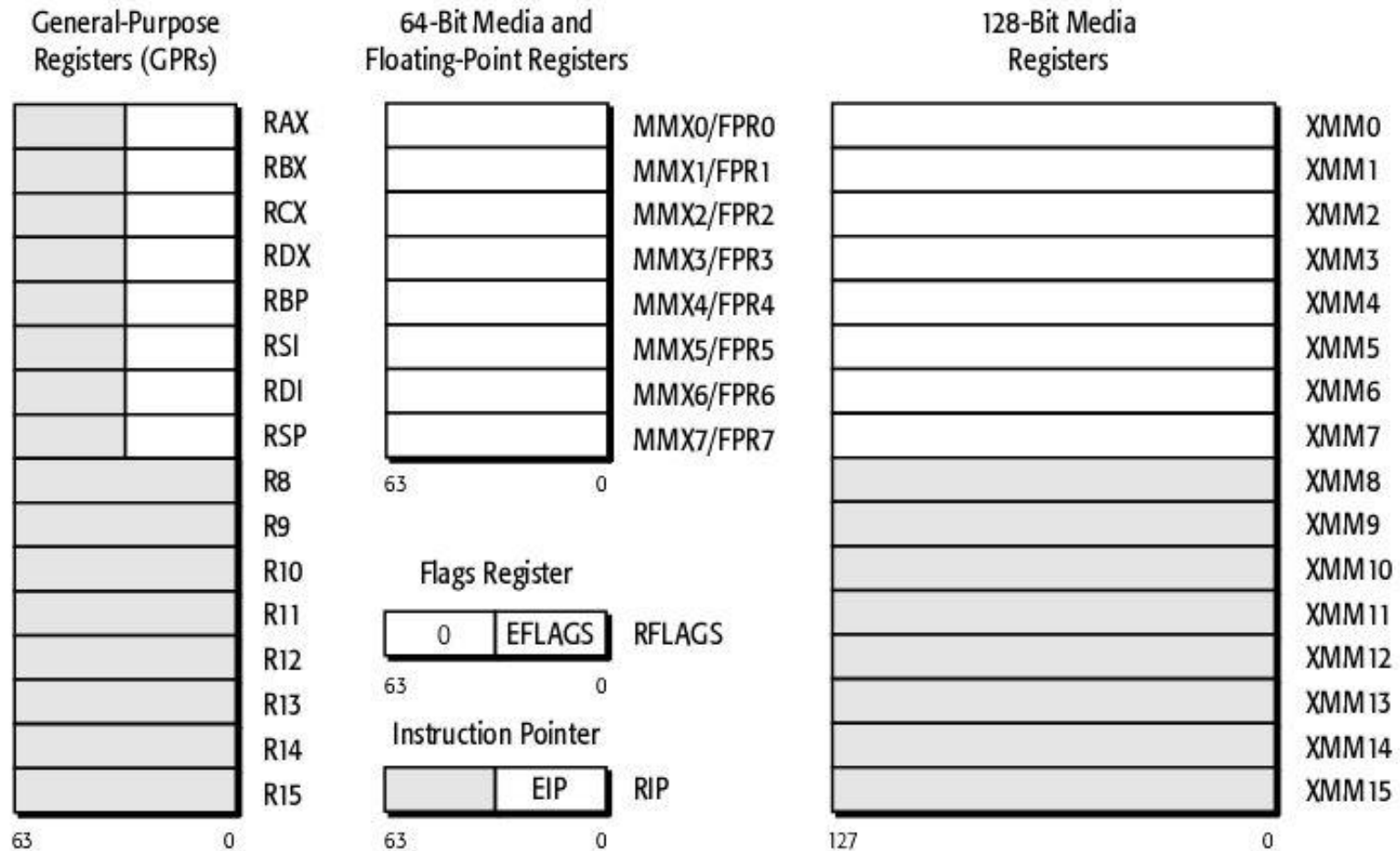
# EFLAGS register









# X86\_64 Registers



-  Legacy x86 registers, supported in all modes
-  Register extensions, supported in 64-bit mode

Application-programming registers also include the 128-bit media control-and-status register and the x87 tag-word, control-word, and status-word registers



# Memory: more work space

---

<code>movl %eax, %edx</code>	<code>edx = eax;</code>	<i>register mode</i>
<code>movl \$0x123, %edx</code>	<code>edx = 0x123;</code>	<i>immediate</i>
<code>movl 0x123, %edx</code>	<code>edx = *(int32_t*)0x123;</code>	<i>direct</i>
<code>movl (%ebx), %edx</code>	<code>edx = *(int32_t*)ebx;</code>	<i>indirect</i>
<code>movl 4(%ebx), %edx</code>	<code>edx = *(int32_t*)(ebx+4);</code>	<i>displaced</i>

- Memory instructions: MOV, PUSH, POP, etc
- Most instructions can take a memory address
- b,w,l,q分别代表8位, 16位,32位和64位





# Stack memory + operations

---

## Example instruction   What it does

<code>pushl %eax</code>	<code>subl \$4, %esp</code> <code>movl %eax, (%esp)</code>
<code>popl %eax</code>	<code>movl (%esp), %eax</code> <code>addl \$4, %esp</code>
<code>call 0x12345</code>	<code>pushl %eip (*)</code> <code>movl \$0x12345, %eip (*)</code>
<code>ret</code>	<code>popl %eip (*)</code>

<code>enter</code>	<code>pushl %ebp</code> <code>movl %esp, %ebp</code>
<code>leave</code>	<code>movl %ebp, %esp</code> <code>popl %ebp</code>

- 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  
...
```

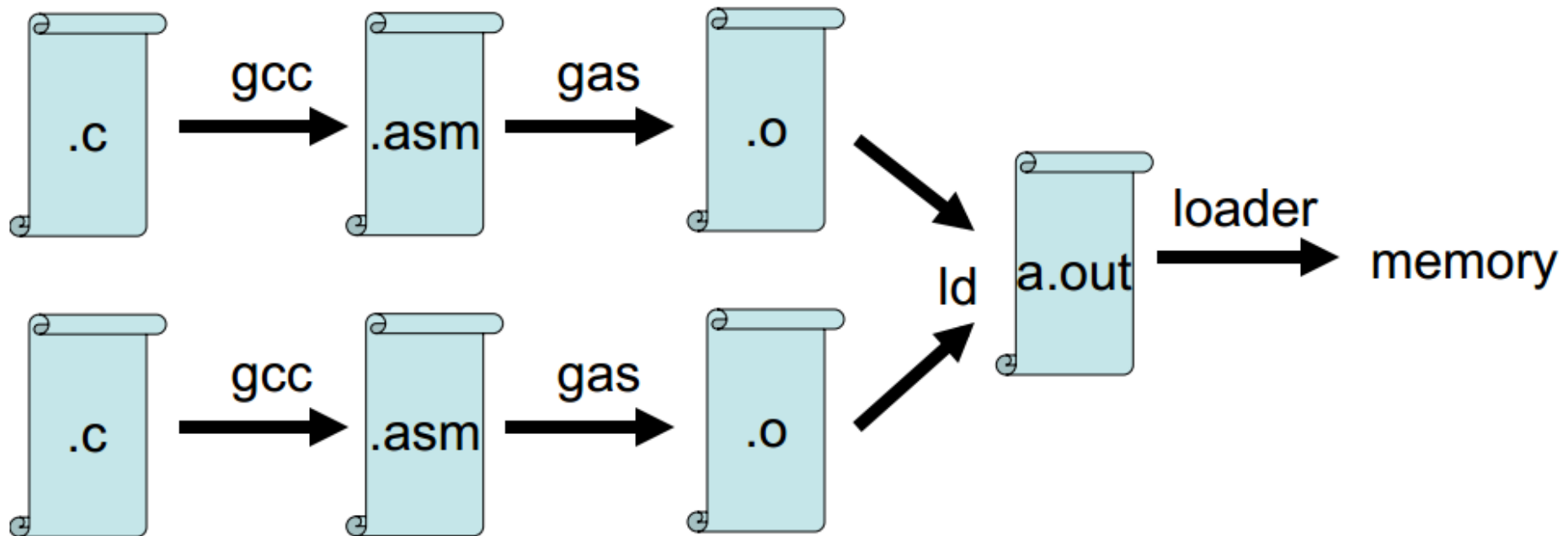
请画出当前堆栈状态，表明当前栈顶和栈基地址位置





# From C to running program

---





# Example

```
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

```
1      ...
2      g:
3          pushl    %ebp
4          movl     %esp, %ebp
5          movl     8(%ebp), %eax
6          addl     $8, %eax
7          popl     %ebp
8          ret
9      ...
10     main:
11         pushl    %ebp
12         movl     %esp, %ebp
13         pushl    $8
14         call     g
15         subl     $8, %eax
16         movl     %ebp, %esp
17         popl     %ebp
18         ret
19     ...
```

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]





# 实验一：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
- ◆ 实验报告要求：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





# 64位环境的注意事项

---

- ◆ //在64位环境下编译成32位的汇编
  - `gcc -S -o ccode32.s ccode.c -m32`
- ◆ //链接时会缺少构建32 位可执行程序缺少的包，使用以下指令安装：
  - `sudo apt-get install libc6-dev-i386`
- ◆ //编译链接成32位的可执行文件
  - `gcc -o ccode32 ccode.c -m32`





“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>