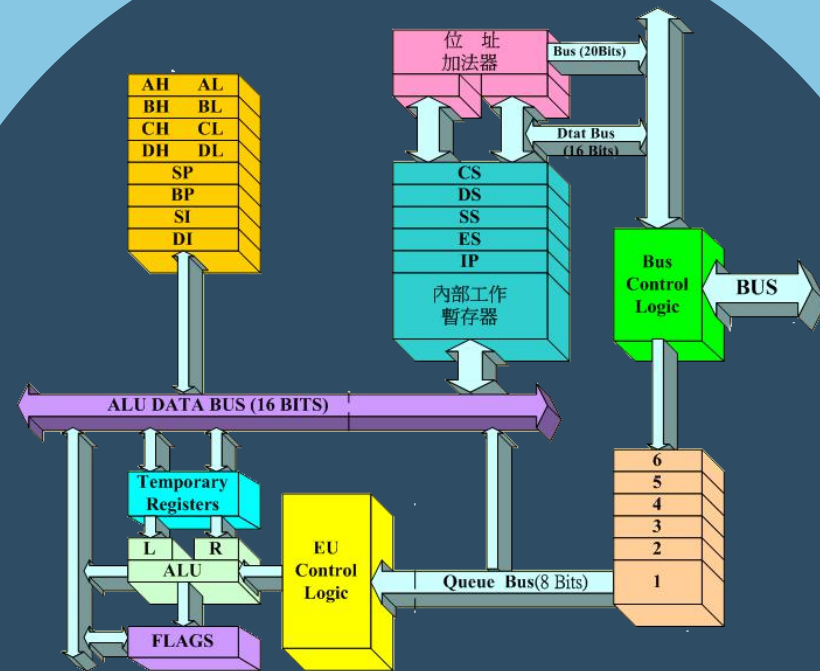


call指令和ret指令

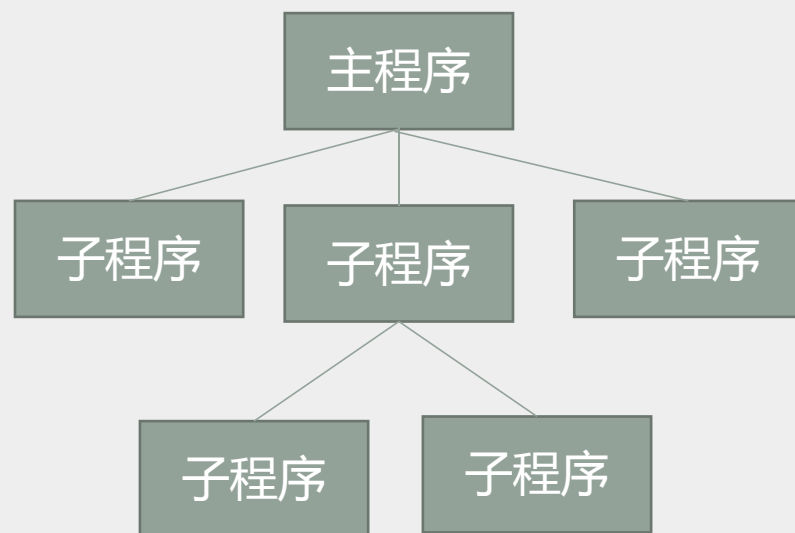
贺利坚 主讲



汇编语言程序设计
Assembly Language

模块化程序设计

```
#include <stdio.h>
int cube(int x);
int main()
{
    printf("%d\n",cube(2));
    return 0;
}
int cube(int x)
{
    int f;
    f=x*x;
    f=f*x;
    return f;
}
```



🖥️ 调用子程序：call指令

🖥️ 返回：ret 指令

🖥️ 示例

```
mov ax, 0
call s
mov ax, 4c00h
int 21h
```

```
s: add ax, 1
ret
```

🖥️ 实质：流程转移指令，它们都修改IP，或同时修改CS和IP

call 指令

💻 字面意思：调用子程序

💻 实质：流程转移

📁 call指令实现转移的方法和 jmp 指令的原理相似

💻 格式：call 标号

call 标号

💻 CPU执行call指令，进行两步操作：

- (1) 将当前的 IP 或 CS和IP 压入栈中；
- (2) 转移到标号处执行指令。

(1) $(sp) = (sp) - 2$
 $((ss)*16+(sp)) = (IP)$
(2) $(IP) = (IP) + 16\text{位位移}$

💻 call 标号

- 📁 16位位移=“标号”处的地址 - call指令后的第一个字节的地址；
- 📁 16位位移的范围为 -32768~32767，用补码表示；
- 📁 16位位移由编译程序在编译时算出。

```
mov ax, 0  
call s  
mov ax, 4c00h  
int 21h
```

```
s: add ax, 1  
ret
```

相当于：

```
push IP  
jmp near ptr 标号
```

指令“call far ptr 标号”实现的是段间转移

🖥️ CPU执行“call far ptr 标号”时的操作

(1) $(sp) = (sp) - 2$

$((ss) \times 16 + (sp)) = (CS)$

$(sp) = (sp) - 2$

$((ss) \times 16 + (sp)) = (IP)$

(2) $(CS) = \text{标号所在的段地址}$

$(IP) = \text{标号所在的偏移地址}$

🖥️ “call far ptr 标号”相当于

push CS

push IP

jmp far ptr 标号

⌚ “call 标号”类似“jmp near ptr 标号”，对应的机器指令中为相对于当前IP的转移位移，而不是转移的目的地址，**实现段内转移**。
⌚ 指令“call far ptr 标号”**实现的是段间转移**！

```
mov ax, 0  
call far ptr s
```

.....


```
mov ax, 4c00h  
int 21h
```

```
s: add ax, 1  
ret
```





转移地址在寄存器中的call指令


指令格式

 call 16位寄存器


功能


 $(sp) = (sp) - 2$

 $((ss)*16+(sp)) = (IP)$

 $(IP) = (16\text{位寄存器})$

相当于进行

 push IP

 jmp 16位寄存器

```
mov ax, 0
```

```
call ax
```

```
.....
```

```
mov ax, 4c00h
```

```
int 21h
```

转移地址在内存中的call指令

 call word ptr 内存单元地址

相当于：

push IP

jmp word ptr 内存单元地址

```
mov sp,10h
```

```
mov ax,0123h
```

```
mov ds:[0],ax
```

```
call word ptr ds:[0]
```

执行后，(IP)=0123H，(sp)=0EH

 call **d**word ptr 内存单元地址

相当于

push CS

push IP

jmp **d**word ptr 内存单元地址

```
mov sp,10h
```

```
mov ax,0123h
```

```
mov ds:[0],ax
```

```
mov word ptr ds:[2],0
```

```
call dword ptr ds:[0]
```

执行后，(CS)=0，(IP)=0123H，(sp)=0CH

低地址放偏移地址

高地址放段地址

返回指令：ret 和 retf

	ret指令	retf指令
功能	用栈中的数据，修改IP的内容，从而实现近转移；	用栈中的数据，修改CS和IP的内容，从而实现远转移；
相当于	pop IP	pop IP pop CS
举例	<pre>1 assume cs:codesg,ss:stack 2 stack segment 3 db 16 dup (0) 4 stack ends 5 codesg segment 6 mov ax,4c00h 7 int 21h 8 start: mov ax,stack 9 mov ss,ax 10 mov sp,16 11 mov ax,0 12 push ax 13 mov bx,0 14 ret 15 codesg ends 16 end start</pre>	<pre>1 assume cs:codesg,ss:stack 2 stack segment 3 db 16 dup (0) 4 stack ends 5 codesg segment 6 mov ax,4c00h 7 int 21h 8 start: mov ax,stack 9 mov ss,ax 10 mov sp,16 11 mov ax,0 12 push cs 13 push ax 14 mov bx,0 15 retf 16 codesg ends 17 end start</pre>

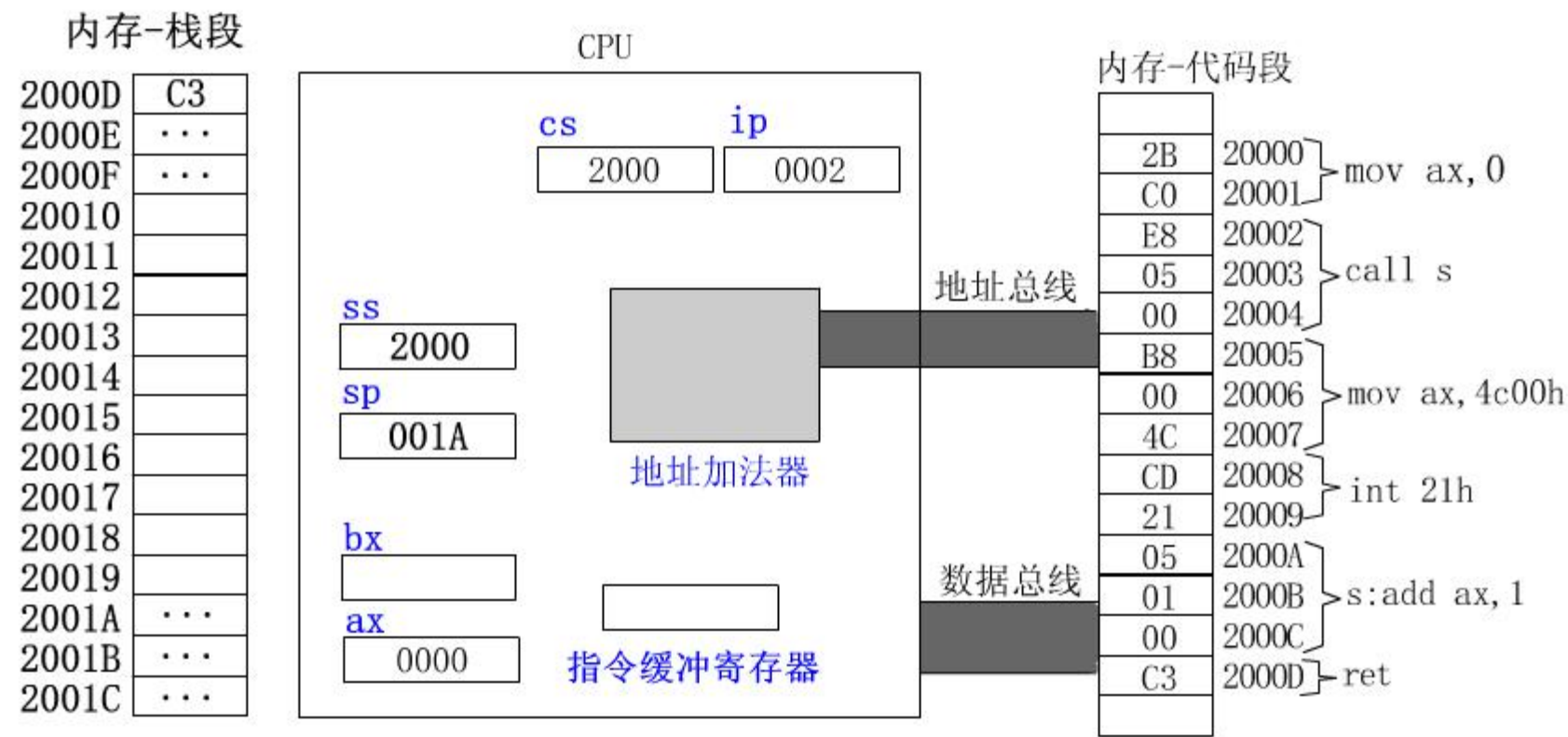
演示

CALL指令和 RET指令执行的 过程

```
mov ax, 0
call s
mov ax, 4c00h
int 21h

s: add ax, 1
ret
```

依据位移进行转移的call指令



依据位移进行转移的call指令