第6章　程序控制指令

罗文坚
中国科大 计算机学院

http://staff.ustc.edu.cn/~wjluo/mcps/

1

本章内容

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

2

转移指令

- 无条件转移指令
  - JMP
- 条件转移指令和条件设置指令
  - 条件转移指令：JA，JAE，JBE，……
  - 条件设置指令：SETA，SETAE，SETB，……
- 循环指令
  - LOOP，LOOPE，LOOPZ，LOOPNE，LOOPNZ

3

转移类型与寻址方式

- 段内转移：同一个段，只改变IP/EIP/RIP
  - near类型：16位，或32位，或64位偏移量（64位模式，实际是40位）
  - short类型：8位（是near类型的一个特例）
- 段间转移：不同段，改变CS: IP/EIP/RIP
  - far类型
- 直接寻址：标号地址、立即数
- 间接寻址：目标地址在REG或MEM中

4

无条件转移指令JMP

- JMP指令：无条件将程序转移到指令指定的目的操作数。
  - 不记录返回地址信息。
- JMP指令可以实现段内转移和段间转移。
- JMP指令的操作数可以是立即数、通用寄存器、存储器地址。

5

无条件转移指令JMP—段内转移

寻址方式	操作数类型		操作数的使用方式	指令实例
直接	标号	1字节立即数	加入IP/EIP/RIP	JMP SHORT START
		2字节立即数		JMP START1
		4字节立即数		JMP START2
	$	立即数		JMP $+2
间接	寄存器操作数		送入IP/EIP/RIP	JMP BX 或 JMP EBX JMP RBX
	存储器操作数			JMP JTABLE[BX]

- START和START1、START2是转移目的标号（符号地址， NEAR类型）
- START指示的目的地址与当前地址间的转移范围在-128~+127个字节内。
- JTABLE是变量，类型为WORD（实模式），DWORD（保护模式）、QWORD（保护模式）。

6

无条件转移指令JMP—段间转移

寻址方式	操作数类型		操作数的使用方式	指令实例
直接	标号	4字节立即数	送入CS和IP/EIP/RIP	JMP START3
		6字节立即数		
		10字节立即数		
间接	存储器操作数		送入CS和IP/EIP/RIP	JMP JTABLE1[BX]

- START3是标号，类型是FAR。
- JTABLE1是变量，类型为DWORD（实模式），FWORD（保护模式）、TWORD（64模式）。

7

无条件转移指令JMP

1. 对于位移量为8位的短转移，在标号前可以加说明符SHORT，也可以省略不写。
2. 对于位移量位16位的近转移，在标号前可以加说明符NEAR PTR，也可以省略不写。
3. 默认情况下，代码标号（标号后跟单个冒号）有一个局部域，对其所在过程内的语句可见，这阻止了跳转或循环语句转移到当前过程之外的标号。
4. 少数情况下，如果必须将控制转移到当前过程之外的标号处，标号必须被声明为全局的。声明全局标号，要在标号后跟两个冒号。

8

Example1

- 例，JMP指令的使用，段内跳转。

```
        XOR  BX, BX
START:  MOV AX, 1
        ADD AX, BX
        JMP SHORT NEXT
        ……
NEXT:   MOV BX, AX
        JMP START
```

9

Example2

- 例，JMP指令的使用，远跳转。

```
        EXTRN UP: FAR
        XOR  BX, BX
START:  MOV AX, 1
        ADD AX, BX
        ……
        JMP FAR PTR NEXT
        ……
        JMP UP
```

10

Example3

- 例，全局标号和局部标号的使用。

```
MAIN PROC
      JMP L2 ;错误!
 L1:: ……    ;全局标号
      ……
      RET
MAIN ENDP

SUB PROC
 L2: ……     ;局部标号
     JMP L1  ;正确
      RET
SUB ENDP
```

11

条件转移指令

- 条件转移指令共计21条，这些指令根据上一条指令执行后处理器的状态标志，确定程序的执行方向。
- 转移范围：
  - 对于16位微机，均为短转移：目的地址必须在当前段内，且与下一条指令的第一个字节的距离在－128～127内。
  - 对于80386以上微处理器，为近转移（±32KB范围）。
  - 在Pentium4的64位模式下，为近转移（±2GB范围）
- 均为直接转移：使用标号地址，机器码中为相对位移量disp。
- 条件转移指令不影响状态位。

12

条件转移指令

- 条件转移指令分为两类：
  - 直接标志转移：这类指令在助记符中直接给出标志状态的测试条件，如jc、jnc、jz、jnz。
  - 间接标志转移：这类指令在助记符中不直接给出标志状态的测试条件，但仍以某一个或某几个标志的状态作为测试条件。
    - 无符号数：
      - JA：高于/不低于等于，JB：低于/不高于等于，…
    - 有符号数：
      - JG：大于/不小于等于，JL：小于/不大于等于，…

13

条件转移指令

单标志位	
助记符	测试条件
JAE/JNB	CF=0
JB/JNAE	CF=1
JC	CF=1
JNC	CF=0
JE/JZ	ZF=1
JNE/JNZ	ZF=0
JNO	OF=0
JO	OF=1
JNP/JPO	PF=0
JP/JPE	PF=1
JNS	SF=0
JS	SF=1

多标志位，无符号数	
助记符	测试条件
JA/JNBE	(CF∨ZF)=0
JBE/JNA	(CF∨ZF)=1

多标志位，带符号数	
JGE/JNL	(SF ⊕ OF) = 0
JL/JNGE	(SF ⊕ OF) = 1
JG/JNLE	((SF⊕OF)∨ZF)= 0
JLE/JNG	((SF⊕OF)∨ZF)= 1

CX/ECX/RCX	
JCXZ	CX=0
JEXC	ECX=0
JRCX	RCX=0

14

条件设置指令

- 83086以上CPU含有条件设置指令。
  - 条件设置指令的功能：根据对条件进行测试的结果，或者把一个字节设置为01H，或者把该字节清除为00H。
  - 有近20条条件设置指令，格式类似。
- 以SETC为例：
  - 格式：SETC REG8/MEM8
  - 功能：如果进位标志位1，则REG8/MEM8置为1，否则为0。
- 条件转移指令要测试的条件可以由条件设置指令来建立。

15

条件设置指令

单标志位	
助记符	测试条件
SETAE	C=0
SETB	C=1
SETC	C=1
SETE或SETZ	Z=1
SETGE	S=0
SETL	SF!=OF
SETNC	C=0
SETNE或SETNZ	Z=0
SETNO	O=0
SETNS	S=0
SETNP或SETPO	P=0
SETO	O=1

单标志位	
助记符	测试条件
SETP或SETPE	P=1
SETS	S=0

多标志位，无符号数	
助记符	测试条件
SETA	C=0且Z=0
SETBE	C=1或Z=1

多标志位，带符号数	
SETG	Z=0且SF=OF
SETLE	Z=1或SF!=OF

16

Example 1

- 例，统计EAX中的8个十六进制数中有多少个0。

```
        MOV BL, 0H
        MOV CX,8     ; 8个十六进制数
AGAIN: TEST AL,0FH;测试低4位二进制数是否为0
        JNZ NEXT     ;不为零，则继续测试下一个4位
        INC BL       ;为零，计数器BL加1
NEXT:   ROR EAX, 4 ；循环右移4位
        LOOP AGAIN
```

17

Example 2

- 例，统计EAX中的8个十六进制数中有多少个0。

```
        MOV BL, 0H
        MOV CX,8     ; 8个十六进制数
AGAIN: TEST AL,0FH;测试低4位二进制数是否为0
        SETZ BH      ; 为零，则置BH为1，否则为0
        ADD BL, BH  ;为零，计数器BL加1
        ROR EAX, 4 ；循环右移4位
        LOOP AGAIN
```

18

循环控制指令

- LOOP指令

格式	LOOP DEST	
功能	8086~80286	CX ← CX-1，CX不为0，则转移到DEST，否则顺序执行。
	80386~Core2	循环计数用CX（16位指令模式）或ECX（32位指令模式）； LOOPW使用CX； LOOPD使用ECX。
	64位模式	循环计数用RCX
标志	不影响状态位。状态位并不受LOOP指令中的“CX－1”的影响。因此，ZF=1时，CX未必为0。	

19

循环控制指令

- 条件循环指令：
  - 为零(相等)循环：LOOPE/LOOPZ DEST
    - CX≠0且ZF=1时，转到DST所指指令。
  - 非零(不相等)循环：LOOPNE/LOOPNZ DEST
    - CX≠0且ZF=0时，转到DST所指指令。
- 注意：
  - 8086~80286，使用CX；80386~Core2，16位指令模式使用CX；32位指令模式，使用ECX；64位模式，使用RCX。
  - 类似于LOOP，也有LOOPEW、LOOPED、LOOPNEW、LOOPNED指令。

20

循环控制指令

- 通常的循环控制

```
        MOV  CX , N
BEGIN:  ....
        ....        循环体
        ....
        DEC  CX
        JNZ  BEGIN  ⇒ LOOP BEGIN
```

21

Example

- 例，两个存储块，对应位置的数据分别相加。

```
.MODEL SMALL                    MOV CX, 100
.DATA                           MOV SI, OFFSET BLOCK1
BLOCK1 DW 100 DUP(?)            MOV DI, OFFSET BLOCK2
BLOCK2 DW 100 DUP(?)      L1:   LODSW
.CODE                           ADD AX, ES:[DI]
.STARTUP                        STOSW
    MOV AX, DS                  LOOP L1
    MOV ES, AX            .EXIT
    CLD                   END
```

22

本章内容

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

23

控制汇编语言程序的流程

- 伪指令：
  - .IF、.ELSE、.ELSEIF、.ENDIF
  - .REPEAT~.UNTIL
  - .WHILE~.ENDW

24

.IF语句

例，测试AL的内容是否在‘A’~‘F’之间。

```
.IF AL>=`A` && AL<=`F`
      SUB AL, 7
.ENDIF
SUB 30H
```

```
      CMP AL, 41H
      JB LATER
      CMP AL, 46H
      JA LATER
      SUB AL, 7
LATER:
      SUB AL, 30H
```

25

.IF语句

- .IF语句的格式：

```
.IF 表达式1
  （汇编语言语句组1）
.ELSEIF 表达式2
  （汇编语言语句组2）
.ELSEIF 表达式3
  （汇编语言语句组3）
  ……
.ELSE
  （汇编语言语句组n）
.ENDIF
```

26

.IF语句

- 用于.IF语句的关系运算符
  1) ==，等于或相同
  2) !=，不等于
  3) >，>=，<，<=
  4) &，位测试
  5) !，逻辑“非”
  6) &&，逻辑“与”
  7) ||，逻辑“或”
  8) |，或

27

本章内容

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

28

过程

- 过程、子程序、函数是程序的重要组成部分。
- 过程的调用指令：CALL
  – 将其后指令的地址（返回地址）压入堆栈
- 过程的返回指令：RET
  – 从堆栈中弹出返回地址
- 过程的定义：以PROC开始，以ENDP结束。

29

过程的定义

- 近过程
  – 段内调用

```
SUMS PROC NEAR
   ADD AX, BX
   ADD AX, CX
   ADD AX, DX
   RET
SUMS ENDP
```

- 远过程
  – 段间调用

```
SUMS1 PROC FAR
   ADD AX, BX
   ADD AX, CX
   ADD AX, DX
   RET
SUMS1 ENDP
```

30

CALL指令

- 近CALL调用
  - 段内调用，将下一条指令的偏移地址（IP/EIP/RIP）压入堆栈。
  - 例，CALL SORT；//设SORT是近过程
- 远CALL调用
  - 段间调用，将将下一条指令的段基址（CS）和偏移地址（ IP/EIP/RIP ）压入堆栈。
  - 例，CALL COS；//设COS是远过程

31

CALL指令

- CALL指令也可以使用寄存器操作数
  - 例，CALL BX，其功能是将IP入栈，并跳转到当前代码段以BX内容为偏移地址的地方继续执行。
- CALL指令也可以使用间接存储器寻址的操作数
  - 例，CALL TABLE [4*EBX]，从数据段中以TABLE [4*EBX]寻址的存储单元得到的数据，作为过程的起始地址。

32

Example1

```
TABLE DW ？；过程ZERO的起始地址
      DW ？；过程ONE的起始地址
      DW ？；过程TWO的起始地址

      CALL TABLE[2*EBX]
```

33

RET指令

- RET指令
  - 近返回：从栈顶取出偏移地址。
  - 远返回：从栈顶取出段基址和偏移地址。
- 带参数的RET指令
  - 格式：RET n
  - 功能：从栈顶弹出返回地址后，将堆栈指针（SP）的内容加上一个数值n。
  - 用途：调用过程前先把参数压入堆栈，如果返回时要丢弃这些参数，可以采用这种形式。非常适用于那些用C/C++或PASCAL调用规则的系统。

34

Example2

```
MOV AX, 30
MOV BX, 40
PUSH AX；堆栈参数1
PUSH BX；堆栈参数1
CALL ADDM
```

```
ADDM PROC NEAR
     PUSH BP
     MOV BP, SP
     MOV AX, [BP+4]
     ADD AX, [BP+6]
     POP BP
     RET 4
ADDM EDNP
```

35

本章内容

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

36

中断概述

- 中断的产生
  - 硬件产生（Hardware-generated），外部中断
    - NMI引脚
    - INTR引脚（可屏蔽中断）
  - 软件产生（Software-generated），内部中断
    - 用于解决CPU在运行过程中发生的一些意外情况。
    - 例如，除零或商溢出。
- 通常，内部中断称为异常。
- 任何类型的中断都是通过调用中断服务程序（ISP，Interrupt Service Procedure）来使当前程序暂停执行。

37

中断及中断返回指令

- CPU每响应一次中断：
  1. 不但要像过程调用指令那样，把CS和IP（或EIP/RIP）寄存器的值（即断点）送入堆栈保存，而且还要将标志寄存器的值入栈保护，以便在中断服务程序执行完后，能够正确恢复CPU的状态。
  2. 根据中断类型号（0～255），找到中断服务程序的入口地址，转相应的中断服务程序。
  3. 中断服务程序结束后，通过中断返回指令IRET，从堆栈中恢复中断前CPU的状态和断点，返回原来的程序继续执行。

38

中断向量

- 中断向量共有256个，每个中断向量保护一个中断服务程序的入口地址（段基址和偏移量）。
- 微处理器按实模式操作时，中断向量（Interrupt Vector）是4个字节的数据，存放在存储器的第一个1024单元。
- 在保护模式下，用中断描述符表代替向量表，每个中断用8个字节的中断描述符说明。

39

中断向量

- 前32个中断向量是Intel保留的，其余的中断向量是用户可用。
- 类型0～类型4中断：
  - 类型0：除法错中断（除数为0或商超过了寄存器能容纳的范围，自动产生）
  - 类型1：单步中断
  - 类型2：不可屏蔽中断
  - 类型3：断点中断（断点可以设置在程序中的任何地方，设置方法是插入一条INT 3指令）
  - 类型4：溢出中断（若溢出标志OF置1，可由INTO指令产生类型为4的中断）

40

中断向量

```
#include <conio.h>
#include <stdio.h>
int   main()
{
        int   a=1,b=0;
        printf( "Division   by   zero:%d\n ",a/b);
        getchar();
        return   0;
}
int   main()
{
        double   x=1.0,y=-1.0,z=0.0;
        printf( "division   by   zero:%f   %f\n ",x/z,y/z);
        getchar();
        return   0;
}
```

为什么整数除0会发生异常?

为什么浮点数除0不会出现异常?

浮点运算中，一个有限数除以0，结果为正无穷大（负无穷大）

问题一：为什么整除int型会产生错误？是什么错误？

二：用double型的时候结果为1.#INF00和-1.#INF00，作何解释???

41

中断及中断返回指令

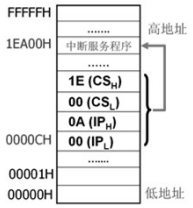
- INT指令
- INT3指令
- INTO指令
- IRET、IRETD、IRETQ指令

42

INT指令

- INT n （n为中断类型码）
  - n 为中断类型号，可以为0~255。INT n可以在编程时安排在程序中的任何位置上。

➢ 以实模式为例，其功能为：
SP ← SP－2 ；标志寄存器入栈
[SP+1]:[SP] ← FR
SP ← SP－2 ；断点地址入栈
[SP+1]:[SP] ← CS
SP ← SP－2
[SP+1]:[SP] ← IP
TF ← 0 ；禁止单步
IF ← 0 ；禁止中断
IP ← [n×4＋1]：[n×4] ；转向中断服务程序
CS ← [n×4＋3]：[n×4＋2]



43

INT n 指令

- INT n （n为中断类型码）
  - 原则上讲，用INT n指令可以调用所有256个中断，尽管其中有些中断是硬件触发的。
- 程序中需要调用某一类型的中断服务程序时，可通过插入INT n指令来实现；也可利用INT n指令来调试各种中断服务程序。
  - 例如，可用INT 0让CPU执行除法出错中断服务程序，而不必运行除法程序；可用INT 2指令执行NMI中断服务程序，从而不必在NMI引脚上加外部信号，就可对NMI子程序进行调试。
- 功能调用：
  - INT 16：BIOS服务
  - INT 21：DOS服务

44

INT3、INTO指令

- INT3指令
  - 格式：INT3
  - 功能：同“INT 3”
- INTO指令
  - 格式：INTO
  - 功能：同“INT 4”
  - 当带符号数进行算术运算时，如果OF＝1，可由INTO产生溢出中断处理；若OF=0，则INTO指令不产生中断。如果程序中无INTO，溢出异常被忽略。
  - 因此，有符号数加减运算后，必须使用INTO，一旦溢出就能及时向CPU提出中断请求，如显示出错信息。溢出中断处理完后，CPU将不返回原程序继续执行，而是把控制权交给操作系统。

45

IRET指令

- IRET：实模式中断返回
  - 总是被安排在中断服务程序的出口处。
  - 当IRET执行后，首先从堆栈中依次弹出程序断点（送入IP和CS），接着弹出标志寄存器；然后按CS:IP的值使CPU返回断点继续执行。
- IRET相当于：
  - 先RET，再POPF。
- 保护模式：IRETD
- 64位模式：IRETQ

46

中断服务程序

- 例，累加DI、SI、BP和BX内容的中断服务程序。

```
INTS PROC FAR USES AX
      ADD AX, BX
      ADD AX, BP
      ADD AX, DI
      ADD AX, SI
      IRET
INTS EDNP
```

47

中断控制

- 硬件产生的外部中断有两个来源：
  - NMI引脚
  - INTR引脚（可屏蔽中断）
- 控制INTR引脚的指令有两条：STI、CLI。
- STI指令
  - 格式：STI
  - 功能：设置中断允许标志，将IF置1，允许INTR输入。
- CLI指令
  - 格式：CLI
  - 功能：清除中断允许标志，将IF清零，禁止INTR输入。

48

PC机的中断

- 早期的PC机是基于8086/8088的系统， Intel保留的中断只包含0~4号中断。
  - 早期的16位微机使用中断向量表。
- Windows平台上访问保护模式中断结构，要通过Microsoft提供的内核调用功能而不能直接寻址。
  - 保护模式中断使用中断描述符表。

49

本章内容

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

50

机器控制及其他指令

- 控制进位标志
  - STC：将CF置1
  - CLC：将CF清0
  - CMC：将CF取反
- WAIT指令
  - 监控8086上的硬件引脚TEST#、286和386上的硬件引脚BUSY#。80486~Core2没有相应引脚。
  - 如果WAIT指令执行时，BUSY#=1，则继续执行下一条指令；如果BUSY#=0，则微处理器要等待，直到BUSY#=1。

51

机器控制及其他指令

- HLT指令
  - CPU暂停，直到有复位（Reset）信号或外部中断请求时退出暂停状态。
    - 在RESET上加复位信号。
    - 在NMI引脚上出现中断请求。在允许中断的情况下，在INTR上出现中断请求信号。
    - 出现DMA操作。
  - 在程序中，通常用HLT指令来等待中断的出现。
  - 因为DOS和Windows大量使用中断，HLT并不停机。
- NOP指令
  - 这是一条单字节指令，执行时需耗费3个时钟周期的时间，但不完成任何操作。

52

机器控制及其他指令

- LOCK前缀
  - 封锁总线指令，禁止其他主控设备使用总线。
  - 是一种前缀，可加在任何指令的前端，用来维持总线封锁引脚LOCK#有效。
  - 例，LOCK: MOV AL, [SI]
- ESC指令
  - 转义指令，从微处理器向浮点协处理器传递指令。
  - 协处理器从ESC指令获得其操作码，并开始执行协处理器指令。
  - ESC从来不在程序中出现。当协处理器指令出现时，汇编程序把它们看做是协处理器的ESC。

53

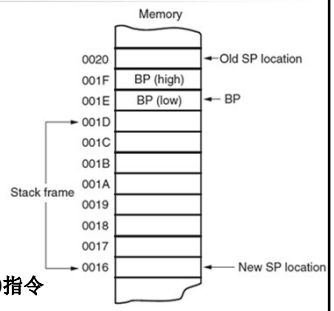
机器控制及其他指令

- BOUND指令
  - 格式：BOUND reg, src
  - （80186以上CPU）检查数组边界，reg是16位或32为寄存器，src为内存中的两个字或双字，是被检查数组的上限和下限。该指令将reg中的值与src中的值进行比较，若reg的值在src的上下限之间，则继续执行下一条指令，否则产生5号中断。
  - 注意：该中断的返回地址是BOUND指令的地址。
  - 例，BOUND SI, DATA
    - 下界在存储单元DATA，上界在存储单元DATA+2。

54

机器控制及其他指令

- ENTER和LEAVE指令
  - 80186以上支持。
  - ENTER 指令（使用时）通常是过程中的第一条指令，用于为过程建立新的堆栈帧。
  - 在过程的末尾（就在RET 指令的前面），使用 LEAVE 指令释放堆栈帧，恢复SP和BP。
- 例，实模式下ENTER 8，0指令
  相当于：PUSH BP
  MOV BP, SP
  SUB SP, 8



55

机器控制及其他指令

- ENTER指令格式：ENTER data16, data8
  - ENTER指令通常是进入过程时要执行的第一条指令，为过程创建堆栈帧。第1个操作数（大小操作数）指定堆栈帧的大小（即堆栈上给过程分配的动态存储空间字节数）。第2个操作数（嵌套层数操作数）给出过程的词法嵌套层级（0~31）。这两个操作数都是立即数。
  - 嵌套层级确定要从前面的帧复制到新堆栈帧“显示区”的堆栈帧指针数。若嵌套层级为0，则处理器将帧指针BP/EBP压入堆栈，将当前堆栈指针ESP复制到BP/EBP，并将当前堆栈指针值减去大小操作数中的值之后的结果加载到SP/ESP。如果嵌套层级大于或等于1，则处理器在调整堆栈指针之前，先将其它帧指针压入堆栈。这些额外的帧指针为被调用过程访问堆栈上的其它嵌套帧提供访问点。
    - 不妨课后调研。

56

机器控制及其他指令

- LEAVE指令格式：LEAVE

```
SORT PROC
    ENTER 8, 0
    ….
    LEAVE
    RET
    ENDP
```

```
SORT PROC
    PUSH BP
    MOV BP, SP
    SUB SP, 8
    ….
    MOV SP, BP
    POP BP
    RET
    ENDP
```

57

本章小结

- 转移指令
- 控制汇编语言程序的流程
- 过程
- 中断概述
- 机器控制及其他指令

- 掌握各类指令的格式、用法！

58

作业

- 习题11，习题25，习题27，习题41，习题47。
- （补充题1）下列程序段执行完以后，程序转移分别到了哪里？

```
程序段1
MOV AX, 147BH
MOV BX, 80DCH
ADD AX, BX
JNO L1
JNC L2
```

```
程序段2
MOV AX, 99D8H
MOV BX, 9847H
SUB AX, BX
JNC L3
JNO L4
```

59