EC325 Microprocessors Branching and Looping

Yasser F. O. Mohammad

REMINDER 1:Explicit Size Declaration

- The problem:
  - mov [ebx], 0
- The solution:
  - mov BYTE PTR [ebx],0
  - mov WORD PTR [ebx],0
  - mov DWORD PTR [ebx],0

REMINDER 2: Addition and Subtraction

- add destination, source
  - Dest=dest+source
- sub destination, source
  - Dest=dest-source
- inc operand
  - operand=operand+1
- dec operand
  - operand=operand-1
- neg operand
  - Operand=-operand (2's complement)
- Why 2's complement???
- Careful: SF does not mean sign if the inputs are unsigned

REMINDER :Signed Multiplication IMUL

- imul source
  - AX=AL*operand ; if byte
  - DX:AX=AX*operand ; if word
  - EDX:EAX=EAX*operand ; if dword
  - CF, OF are set if the high order half is significant
- imul register, source
  - register=register*source
  - CF, OF are set if the result cannot fit into *register*
- imul register, source, immediate
  - register=source*immediate
  - CF, OF are set if the result cannot fit into *register*

REMINDER 4: Signed Division

- idiv divisor
  - Same as idiv but quotient takes the sign of the operation
  - Sign of the remainder = sign of dividend
  - Sign of quotient is negative iff sign of dividend and divisor are different

Before	*Instruction executed*	*After*				
EDX: 00 00 00 00 EAX: 00 00 00 64 EBX: 00 00 00 OD	`div ebx`	EDX	00	00	00	09
		EAX	00	00	00	07
DX: 00 00 AX: 00 64 CX: 00 0D	`idiv cx`	DX	00	09		
		AX	00	07		
AX: 00 64 byte at Divisor: 0D	`div Divisor`	AX	09	07		

REMINDER 5: Carry Flag Control

Instruction	Operation	Clock Cycles	Number of Bytes	Opcode
`clc`	clear carry flag (CF := 0)	2	1	F8
`stc`	set carry flag (CF := 1)	2	1	F9
`cmc`	complement carry flag (if CF = 0 then CF := 1 else CF := 0)	2	1	F5

What is it all about?

- Goto
- IF …. THEN ….. ELSE …. END IF
- WHILE
- FOR

Unconditional Jmp

```
jmp     quit
          .
          .
quit:   INVOKE ExitProcess, 0
          .
          .
```

- Jmp statement
- Jmp offset
  - Offset = register, or memory location (signed)
  - Offset is added to the address of *next* instruction
- Jmp Types:
  - Relative Jump = Interasegment Jump = changes EIP
  - Far Jump = Intersegment Jump = changes CS, EIP
  - Task Switch = Jump to a different task (privileged)

Offset Type	Offset Size	Maximum offset
Relative short	4 bytes	-2147483648 → 2147483647
Relative near	Single byte	-128→127
Register indirect	4 bytes	-2147483648 → 2147483647
Memory indirect	4 bytes	-2147483648 → 2147483647

Why do we need relative short jmp?

Unconditional Jump Notes

- Address are cyclic
  - 0FFFFH +2=0001H
- Relative short maximum displacement
  - Before 80386: ±32K
  - Since 80386:
    - Real mode: ±32K
    - Protected mode: ±4G

Unconditional Jump Example

SHORT JUMP: Before 80386 (using IP)

Opcode

E B	Disp

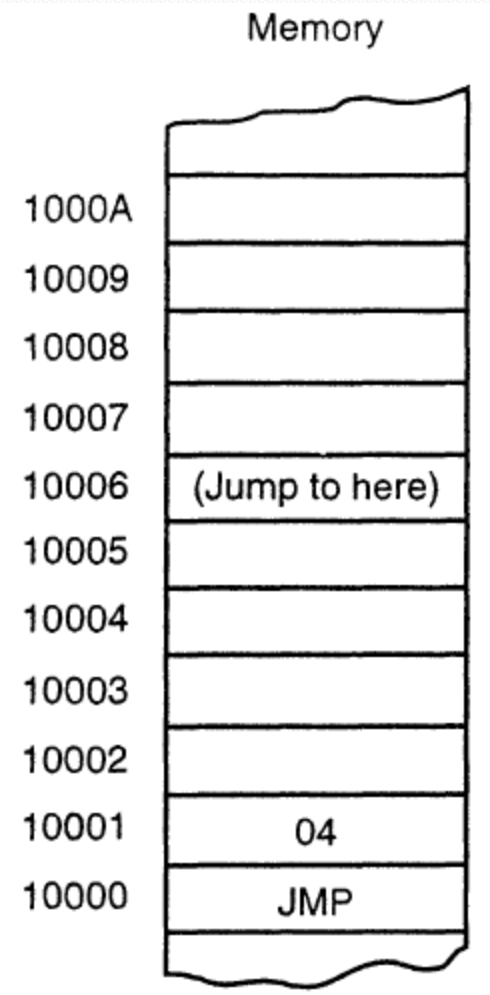
CS = 1000H
IP = 0002H
New IP = IP + 4
New IP = 0006H

Opcode

E 9	Disp Low	Disp High

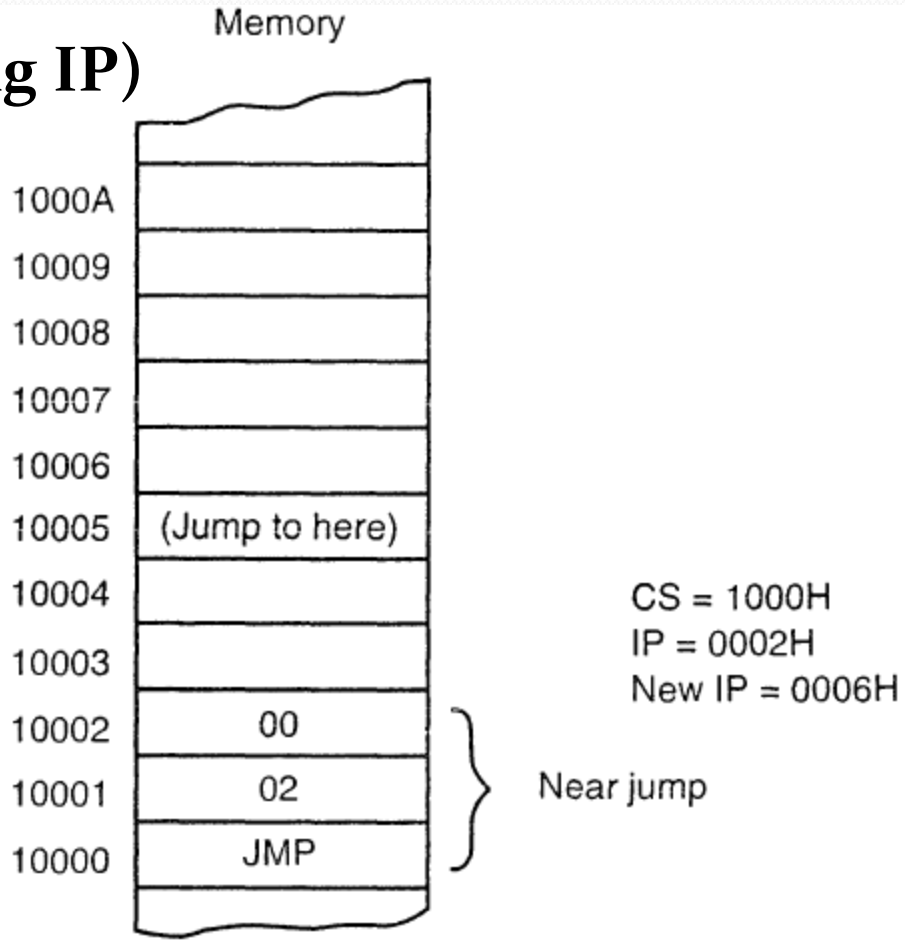
Opcode

E A	IP Low	IP High	CS Low	CS High



Unconditional Jmp Example

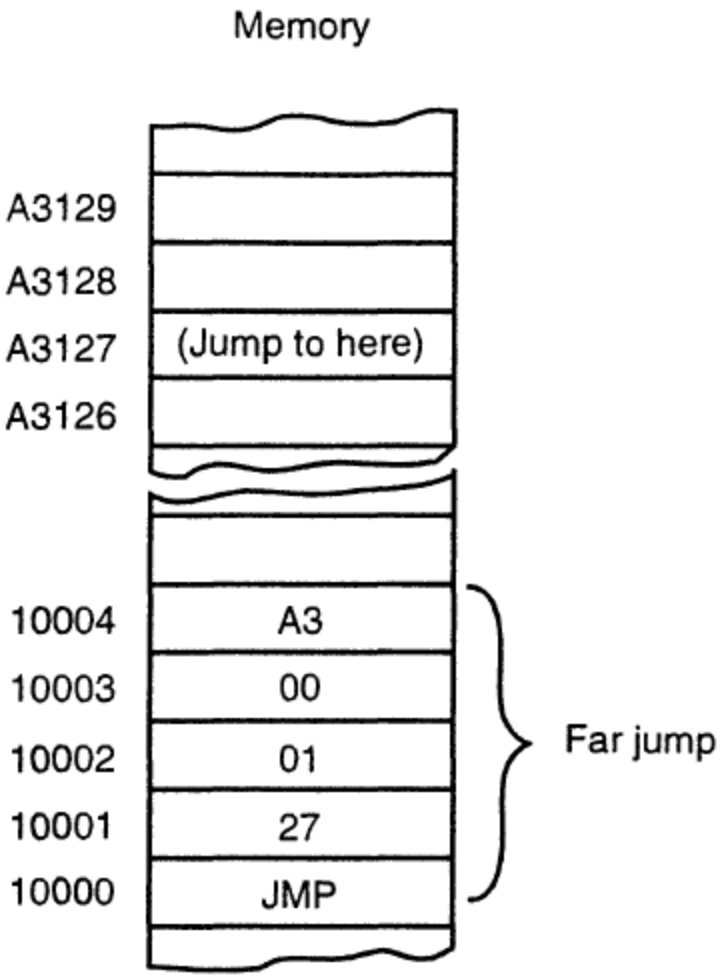
NEAR JUMP: Before 80386 (using IP)



Unconditional Jmp Example

FAR JUMP: Before 80386 (using IP)

- JMP FAR PTR Address



Jmp miscellaneous info

- Jmp $+4
  - Jump 4 bytes after next instruction

Conditional Jump

- J* targetStatement

- * identifies the condition to take the jump

Reading/Writing in DOS*

- Reading one character to AL
  - MOV AH, 1
  - INT 21H
- Writing one character from DL
  - MOV AH, 2
  - INT 21H

Conditional JMP EXAMPLE

```
                        ;A program that reads 1, 2, or 3 from the keyboard
                        ;if a 1, 2, or 3 is typed, a 1, 2, or 3 is displayed.
                        ;
                               .MODEL SMALL              ;select SMALL model
0000                           .DATA                     ;start of DATA segment
0000  0030 R            TABLE  DW     ONE                ;define lookup table
0002  0034 R                   DW     TWO
0004  0038 R                   DW     THREE
0000                           .CODE                     ;start of CODE segment
                               .STARTUP                  ;start of program
0017                    TOP:
0017  B4 01                    MOV    AH,1               ;read key into AL
0019  CD 21                    INT    21H

001B  2C 31                    SUB    AL,31H             ;convert to binary
001D  72 F8                    JB     TOP                ;if below '1' typed
001F  3C 02                    CMP    AL,2
0021  77 F4                    JA     TOP                ;if above '3' typed

0023  B4 00                    MOV    AH,0               ;double to 0, 2, or 4
0025  03 C0                    ADD    AX,AX
0027  BE 0000 R                MOV    SI,OFFSET TABLE    ;address lookup table
002A  03 F0                    ADD    SI,AX              ;form lookup address
002C  8B 04                    MOV    AX,[SI]            ;get ONE, TWO, or THREE
002E  FF E0                    JMP    AX                 ;jump address
0030                    ONE:
0030  B2 31                    MOV    DL,'1'             ;load '1' for display
0032  EB 06                    JMP    BOT                ;go display '1'
0034                    TWO:
0034  B2 32                    MOV    DL,'2'             ;load '2' for display
0036  EB 02                    JMP    BOT                ;go display '2'
0038                    THREE:
0038  B2 33                    MOV    DL,'3'             ;load '3' for display
003A                    BOT:
003A  B4 02                    MOV    AH,2               ;display number
003C  CD 21                    INT    21H
                               .EXIT                     ;exit to DOS
                               END                       ;end of file
```

Conditional Jumps

Assembly Language	*Condition Tested*	*Operation*
JA	Z = 0 and C = 0	Jump if above
JAE	C = 0	Jump if above or equal
JB	C = 1	Jump if below
JBE	Z = 1 or C = 1	Jump if below or equal
JC	C = 1	Jump if carry set
JE or JZ	Z = 1	Jump if equal or jump if zero
JG	Z = 0 and S = O	Jump if greater than
JGE	S = O	Jump if greater than or equal
JL	S <> O	Jump if less than
JLE	Z = 1 or S <> O	Jump if less than or equal
JNC	C = 0	Jump if no carry
JNE or JNZ	Z = 0	Jump if not equal or jump if not zero
JNO	O = 0	Jump if no overflow
JNS	S = 0	Jump if no sign
JNP or JPO	P = 0	Jump if no parity or jump if parity odd
JO	O = 1	Jump if overflow set
JP or JPE	P = 1	Jump if parity set or jump if parity even
JS	S = 1	Jump if sign is set
JCXZ	CX = 0	Jump if CX is zero
JECXZ	ECX = 0	Jump if ECX is zero

Comparing things

- CMP source1, source2
  - source1-source2
  - Adjusts Flags
- Usually used before conditional jumps
- Immediates comes next
- Has relative short, relative near jumps

Interpreting Flags

	operand1	operand2	difference	flags				interpretation	
				CF	OF	SF	ZF	signed	unsigned
1	3B	3B	00	0	0	0	1	*op1=op2*	*op1=op2*
2	3B	15	26	0	0	0	0	*op1>op2*	*op1>op2*
3	15	3B	DA	1	0	1	0	*op1<op2*	*op1<op2*
4	F9	F6	03	0	0	0	0	*op1>op2*	*op1>op2*
5	F6	F9	FD	1	0	1	0	*op1<op2*	*op1<op2*
6	15	F6	1F	1	0	0	0	*op1>op2*	*op1<op2*
7	F6	15	E1	0	0	1	0	*op1<op2*	*op1>op2*
8	68	A5	C3	1	1	1	0	*op1>op2*	*op1<op2*
9	A5	68	3D	0	1	0	0	*op1<op2*	*op1>op2*

Conditional Jump Instructions

Appropriate for use after comparison of unsigned operands

mnemonic	description	flags to jump	opcode short	opcode near
`ja`	jump if above	CF=0 and ZF=0	77	OF 87
`jnbe`	jump if not below or equal			
`jae`	jump if above or equal	CF=0	73	OF 83
`jnb`	jump if not below			
`jb`	jump if below	CF=1	72	OF 82
`jnae`	jump if not above or equal			
`jbe`	jump if below or equal	CF=1 or ZF=1	76	OF 86
`jna`	jump if not above			

(continued)

Conditional Jump Instructions 2

Appropriate for use after comparison of signed operands

mnemonic	description	flags to jump	opcode short	opcode near
`jg`	jump if greater	SF=OF and ZF=0	7F	OF 8F
`jnle`	jump if not less or equal			
`jge`	jump if greater or equal	SF=OF	7D	OF 8D
`jnl`	jump if not less			
`jl`	jump if less	SF≤OF	7C	OF 8C
`jnge`	jump if not greater or equal			
`jle`	jump if less or equal	SF≤OF or ZF=1	7E	OF 8E
`jng`	jump if not greater			

Conditional Jump Instructions 3

Other conditional jumps

mnemonic	description	flags to jump	opcode short	opcode near
je	jump if equal	ZF=1	74	OF 84
jz	jump if zero			
jne	jump if not equal	ZF=0	75	OF 85
jnz	jump if not zero			
js	jump if sign	SF=1	78	OF 88
jns	jump if not sign	SF=0	79	OF 89
jc	jump if carry	CF=1	72	0F 82
jnc	jump if not carry	CF=0	73	0F 83
jp	jump if parity	PF=1	7A	OF 8A
jpe	jump if parity even			
jnp	jump if not parity	PF=0	7B	OF 8B
jpo	jump if parity odd			
jo	jump if overflow	OF=1	70	OF 80
jno	jump if not overflow	OF=0	71	OF 81

Example Conditional Jump 1

```
if value < 10
then
        add 1 to smallCount;
else
        add 1 to largeCount;
end if;
```

```
            cmp   ebx, 10          ; value < 10 ?
            jnl   elseLarge
            inc   smallCount       ; add 1 to small_count
            jmp   endValueCheck
elseLarge:  inc   largeCount       ; add 1 to large_count
endValueCheck:
```

Example Conditional Jump 2

```
if (total ≥ 100) or (count = 10)
then
        add value to total;
end if;
```

```
            cmp   total, 100      ; total >= 100 ?
            jge   addValue
            cmp   cx, 10          ; count = 10 ?
            jne   endAddCheck
addValue:   mov   ebx, value      ; copy value
            add   total, ebx      ; add value to total
endAddCheck:
```

Example Conditional Jump 3

```
if (count > 0) and (ch = backspace)
then
        subtract 1 from count;
end if;
```

```
        cmp   cx, 0             ; count > 0 ?
        jng   endCheckCh
        cmp   al, backspace     ; ch a backspace?
        jne   endCheckCh
        dec   count             ; subtract 1 from count
endCheckCh:
```

While Loop Using Jump

```
while continuation condition loop
   ... { body of loop }
end while;
```

```
while:          .               ; code to check Boolean expression
                .
                .
body:           .               ; loop body
                .
                .
                jmp   while     ; go check condition again
endWhile:
```

Example While Loop 1

```
while (sum < 1000) loop
        ... { body of loop }
end while;
```

```
whileSum:     cmp    sum, 1000       ; sum < 1000?
              jnl    endWhileSum     ; exit loop if not
               .                     ; body of loop
               .
               .
              jmp    whileSum        ; go check condition again
endWhileSum:
```

Example While Loop 2

```
x := 0;
twoToX := 1;
while twoToX < number
        multiply twoToX by 2;
        add 1 to x;
end while;
subtract 1 from x;
```

```
            mov   cx, 0         ; x := 0
            mov   eax, 1        ; twoToX := 1
whileLE:    cmp   eax, number   ; twoToX <= number?
            jnle  endWhileLE    ; exit if not
body:       add   eax, eax      ; multiply twoToX by 2
            inc   cx            ; add 1 to x
            jmp   whileLE       ; go check condition again
endWhileLE:
            dec   cx            ; subtract 1 from x
```

Example While Loop 3

```
while (sum < 1000) or (flag = 1) loop
        ... { body of loop }
end while;
```

```
whileSum:    cmp    eax, 1000        ; sum < 1000?
             jl     body             ; execute body if so
             cmp    dh,1             ; flag = 1?
             jne    endWhileSum      ; exit if not
body:         .                      ; body of loop
              .
              .
             jmp    whileSum         ; go check condition again
endWhileSum:
```

For Loop using JUMP

```
prompt for tally of numbers;
input tally;
sum := 0
for count := 1 to tally loop
        prompt for number;
        input number;
        add number to sum;
end for;
```

```
                output prompt1          ; prompt for tally
                input  value, 20        ; get tally (ASCII)
                atoi   value            ; convert to 2's complement
                mov    tally, ax        ; store tally

                mov    edx, 0           ; sum := 0
                mov    bx, 1            ; count := 1

forCount:       cmp    bx, tally        ; count <= tally?
                jnle   endFor           ; exit if not
                output prompt2          ; prompt for number
                input  value, 20        ; get number (ASCII)
                atod   value            ; convert to 2's complement
                add    edx, eax         ; add number to sum
                inc    bx               ; add 1 to count
                jmp    forCount         ; repeat
endFor:
```

Until using JUMP

```
count := 0;
until (sum > 1000) or (count = 100) loop
        ... { body of loop }
        add 1 to count;
end until;
```

```
            mov    cx, 0        ; count := 0
until:       .                  ; body of loop
             .
             .
            inc    cx           ; add 1 to count
            cmp    sum, 1000    ; sum > 1000 ?
            jg     endUntil     ; exit if sum > 1000
            cmp    cx, 100      ; count = 100 ?
            jne    until        ; continue if count not = 100
endUntil:
```

Endless loop with a break

```
forever loop
        .
        .
        .
        if (response = 's') or (response = 'S')
        then
                exit loop;
        end if;
        .
        .
        .
end loop;
```

```
forever:        .
                .
                .
                cmp     al, 's'         ; response = 's'?
                je      endLoop         ; exit loop if so
                cmp     al, 'S'         ; response = 'S'?
                je      endLoop         ; exit loop if so
                .
                .
                .
                jmp     forever         ; repeat loop body
endLoop:
```

Conditional Set Instructions*

Assembly Language	*Condition Tested*	*Operation*
SETB	C = 1	Set if below
SETAE	C = 0	Set if above or equal
SETBE	Z = 1 or C = 1	Set if below or equal
SETA	Z = 0 and C = 0	Set if above
SETE or SETZ	Z = 1	Set if equal or set if zero
SETNE or SETNZ	Z = 0	Set if not equal or set if not zero
SETL	S <> O	Set if less than
SETLE	Z = 1 or S <> O	Set if less than or equal
SETG	Z = 0 and S = O	Set if greater than
SETGE	S = O	Set if greater than or equal
SETS	S = 1	Set if sign (negative)
SETNS	S = 0	Set if no sign (positive)
SETC	C = 1	Set if carry
SETNC	C = 0	Set if no carry
SETO	O = 1	Set if overflow
SETNO	O = 0	Set if no overflow
SETP or SETPE	P = 1	Set if parity or set if parity even
SETNP or SETPO	P = 0	Set if no parity or set if parity odd

LOOP instruction

- loop statement
  - Statetement must be short distance from the instruction (-128→ 127 bytes)
  - Does the following:
    - ECX=ECX-1
    - If ECX==0 then continue to next statement
    - If ECX ≠ 0 then jump to *statement*
  - Similar to a high level For-Loop with count in ECX

```
for(     ; ECX>0;  ECX--){
    .
    .
}
```

Example Loop 1

```
for count := 20 downto 1 loop
        ... { body of loop }
end for;
```

```
            mov   ecx, 20        ; number of iterations
forCount:    .                   ; body of loop
             .
             .
            loop  forCount       ; repeat body 20 times
```

Example Loop 2

- Count is stored in memory location *number*

```
            mov   ecx, number    ; number of iterations
forIndex:    .                   ; body of loop
             .
             .
            loop  forIndex       ; repeat body number times
```

- *What is wrong?*

Example Loop 2 Corrected

- Count is stored in memory location *number*

```
            mov   ecx, number    ; number of iterations
            cmp   ecx, 0         ; number = 0 ?
            je    endFor         ; skip loop if number = 0
forIndex:    .                   ; body of loop
             .
             .
             loop  forIndex      ; repeat body number times
endFor:
```

Example Loop 2 Another Correction

- Count is stored in memory location *number*

```
            mov   ecx, number    ; number of iterations
            jecxz endFor         ; skip loop if number = 0
forIndex:     .                  ; body of loop
              .
              .
            loop  forIndex       ; repeat body number times
endFor:
```

Large For Loops using JUMP

```
for counter := 50 downto 1 loop
      ... { body of loop }
end for;
```

```
            mov   ecx, 50        ; number of iterations
forCounter:  .                   ; body of loop
             .
             .
            dec   ecx            ; decrement loop counter
            jecxz endFor         ; exit if counter = 0
            jmp   forCounter     ; otherwise repeat body
endFor:
```

Forward For Loop

```
for index := 1 to 50 loop
        ...{ loop body using index }
end for;
```

```
        mov    ebx, 1       ; index := 1
        mov    ecx, 50      ; number of iterations for loop
forNbr:   .
          .                 ; use value in EBX for index
          .
        inc    ebx          ; add 1 to index
        loop   forNbr       ; repeat
```

Loop instruction variants

- loope/loopz statement
  - Does the following:
    - ECX=ECX-1
    - If ECX==0 AND ZF==1 then continue to next statement
    - If ECX ≠ 0 OR ZF==0 then jump to *statement*

- loopne/loopnz statement
  - Does the following:
    - ECX=ECX-1
    - If ECX==0 OR ZF==1 then continue to next statement
    - If ECX ≠ 0 AND ZF==0 then jump to *statement*

Why Loop variants?

```
for year := 10 downto 1 until balance = 0 loop
        ... { body of loop }
end for;
```

```
            mov     ecx, 10 ; maximum number of iterations
forYear:     .              ; body of loop
             .|
             .
            cmp     ebx, 0  ; balance = 0 ?
            loopne  forYear ; repeat 10 times if balance not 0
```

Putting It All Together

- Read a set of nonzero numbers until you read a zero.
- Calculate their average.
- Print all numbers over the average.

```
nbrElts := 0;              { input numbers into array }
get address of first item of array;

while (number from keyboard > 0) loop
      convert number to 2's complement;
      store number at address in array;
      add 1 to nbrElts;
      get address of next item of array;
end while;

sum := 0;                  { find sum and average }
get address of first item of array;

for count := nbrElts downto 1 loop
      add doubleword at address in array to sum;
      get address of next item of array;
end for;

average := sum/nbrElts;
display average;

get address of first item of array;  { list big numbers }

for count := nbrElts downto 1 loop
      if doubleword of array > average
      then
            convert doubleword to ASCII;
            display value;
      end if;
      get address of next item of array;
end for;
```

Big Number Printer in Assembly

```
; input a collection of numbers
; report their average and the numbers which are above average
; author:  R. Detmer
; date:  revised 9/97

.386
.MODEL FLAT

INCLUDE io.h

ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

cr          EQU    0dh   ; carriage return character
Lf          EQU    0ah   ; linefeed character
maxNbrs     EQU    100   ; size of number array

.STACK      4096

.DATA
directions  BYTE     cr, Lf, 'You may enter up to 100 numbers'
            BYTE     ' one at a time.',cr,Lf
            BYTE     'Use any negative number to terminate
                     input.',cr,Lf,Lf
            BYTE     'This program will then report the average and
                     list',cr,Lf
            BYTE     'those numbers which are above the
                     average.',cr,Lf,Lf,Lf,0
prompt      BYTE     'Number?  ',0
number      BYTE     20 DUP (?)
nbrArray    DWORD    maxNbrs DUP (?)
nbrElts     DWORD    ?
avgLabel    BYTE     cr,Lf,Lf,'The average is'
outValue    BYTE      11 DUP (?), cr,Lf,0
aboveLabel  BYTE      cr,Lf,'Above average:',cr,Lf,Lf,0

.CODE
_start:
; input numbers into array

    output directions     ; display directions
    mov    nbrElts,0      ; nbrElts := 0
    lea    ebx,nbrArray   ; get address of nbrArray
```

```
whilePos:   output prompt        ; prompt for number
            input  number,20     ; get number
            atod   number        ; convert to integer
            jng    endWhile      ; exit if not positive
            mov    [ebx],eax     ; store number in array
            inc    nbrElts       ; add 1 to nbrElts
            add    ebx,4         ; get address of next item of array
            jmp    whilePos      ; repeat
endWhile:

; find sum and average

            mov    eax,0         ; sum := 0
            lea    ebx,nbrArray  ; get address of nbrArray
            mov    ecx,nbrElts   ; count := nbrElts

            jecxz  quit          ; quit if no numbers
forCount1:  add    eax,[ebx]     ; add number to sum
            add    ebx,4         ; get address of next item of array
            loop   forCount1     ; repeat nbrElts times

            cdq                  ; extend sum to quadword
            idiv   nbrElts       ; calculate average
            dtoa   outValue,eax  ; convert average to ASCII
            output avgLabel      ; print label and average
            output aboveLabel    ; print label for big numbers
```

```
forCount2:  cmp    [ebx],eax         ; doubleword > average ?
            jng    endIfBig          ; continue if average not less
            dtoa   outValue,[ebx]    ; convert value from array to
                                     ; ASCII
            output outValue          ; display value
endIfBig:
            add    ebx,4             ; get address of next item of array
            loop   forCount2         ; repeat

quit:       INVOKE ExitProcess, 0     ; exit with return code 0

PUBLIC _start                        ; make entry point public
            END                      ; end of source code
```