6. Branch Instructions
Branch-related Addressing Modes

• Defined as the way in which a branch operand is specified. Supported modes are:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Format</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrasegment Direct</td>
<td>Label±Constant Expression</td>
<td>error+6 start_loop</td>
</tr>
<tr>
<td>Intrasegment Indirect</td>
<td>Same as Data-related except cannot be immediate mode</td>
<td>[bx]; table[si+8]</td>
</tr>
<tr>
<td>Intersegment Direct</td>
<td>Label±Constant Expression</td>
<td>far_exit abort-8</td>
</tr>
<tr>
<td>Intersegment Indirect</td>
<td>Same as Data-related except cannot be immediate or register mode</td>
<td>[bx]; table[si+8]</td>
</tr>
</tbody>
</table>

• Intrasegment Direct – New IP is sum of current IP plus displacement given by instruction; no CS change:

• Intrasegment Indirect – New IP retrieved from register or memory location determined using any data-related addressing mode, except immediate; no CS change. Refer to data-related addressing mode diagrams.
Branch-related Addressing Modes – Cont’d

- Intersegment Direct – Both new IP and new CS given by instruction:

- Intersegment Indirect – New IP and new CS retrieved from two consecutive word locations determined using any data-related addressing mode, except immediate or register. Refer to data-related addressing mode diagrams.
Unconditional Branch Instructions

These instructions transfer execution control to another point of the program altering IP and, in some cases, CS.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic and Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrasegment direct short branch</td>
<td>jmp short opr</td>
<td>(ip)←(ip)+sign extended 8-bit displacement</td>
</tr>
<tr>
<td>Intrasegment direct near branch</td>
<td>jmp near ptr opr</td>
<td>(ip)←(ip)+16-bit displacement</td>
</tr>
<tr>
<td>Intrasegment indirect branch</td>
<td>jmp opr16</td>
<td>(ip)← (EA of opr)</td>
</tr>
<tr>
<td>Intersegment direct (far) branch</td>
<td>jmp far ptr opr</td>
<td>(ip)←EA of opr (cs)←Segment of opr</td>
</tr>
<tr>
<td>Intersegment indirect branch</td>
<td>jmp opr32</td>
<td>(ip)←(EA of opr) (cs)←(EA of opr + 2)</td>
</tr>
</tbody>
</table>

Flags: Not affected

Addressing Modes: Refer to branch-related addressing modes table.
EXAMPLE:

extrn action2: near
extrn action3: far

.model medium

.data
    vector2     dw     action2
    vector3     dd     action3

.code
    ...
    ...
    ;Intrasegment direct short
    jmp short action1
    ...
    ...
    action1:
    ...
    ...
    ;Intrasegment direct near
    jmp near ptr action2
    ...
    ...
    ;Intrasegment indirect
    jmp vector2
    ...
    ...
    ;Intersegment direct far
    jmp far ptr action3
    ...
    ...
    ;Intersegment indirect
    jmp vector3
    ...
    ...

Conditional Branch Instructions

These instructions transfer execution control to another point depending on the conditional flags.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic and Format</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch on zero, or equal</td>
<td>jz opr</td>
<td>zf = 1</td>
</tr>
<tr>
<td></td>
<td>je opr</td>
<td></td>
</tr>
<tr>
<td>Branch on non-zero, or not equal</td>
<td>jnz opr</td>
<td>zf = 0</td>
</tr>
<tr>
<td></td>
<td>jne opr</td>
<td></td>
</tr>
<tr>
<td>Branch on sign set</td>
<td>js opr</td>
<td>sf = 1</td>
</tr>
<tr>
<td>Branch on sign clear</td>
<td>jns opr</td>
<td>sf = 0</td>
</tr>
<tr>
<td>Branch on overflow</td>
<td>jo opr</td>
<td>of = 1</td>
</tr>
<tr>
<td>Branch on no overflow</td>
<td>jno opr</td>
<td>of = 0</td>
</tr>
<tr>
<td>Branch on even parity, or parity set</td>
<td>jpe opr</td>
<td>pf = 1</td>
</tr>
<tr>
<td></td>
<td>jp opr</td>
<td></td>
</tr>
<tr>
<td>Branch on odd parity, or parity clear</td>
<td>jpo opr</td>
<td>pf = 0</td>
</tr>
<tr>
<td></td>
<td>jnp opr</td>
<td></td>
</tr>
<tr>
<td>Branch on below, or not above or equal (unsigned)</td>
<td>jb opr</td>
<td>cf = 1</td>
</tr>
<tr>
<td></td>
<td>jnae opr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jc opr</td>
<td></td>
</tr>
<tr>
<td>Branch on not below, or above or equal (unsigned)</td>
<td>jnb opr</td>
<td>cf = 0</td>
</tr>
<tr>
<td></td>
<td>jae opr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jnc opr</td>
<td></td>
</tr>
</tbody>
</table>
## Cond. Branch Instr. - Cont’d

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic and Format</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch on below or equal, or not above (unsigned)</td>
<td>jbe opr</td>
<td>cf</td>
</tr>
<tr>
<td></td>
<td>jna opr</td>
<td></td>
</tr>
<tr>
<td>Branch on not below or equal, or above (unsigned)</td>
<td>jnbe opr</td>
<td>cf</td>
</tr>
<tr>
<td></td>
<td>ja opr</td>
<td></td>
</tr>
<tr>
<td>Branch on less, or not greater or equal (signed)</td>
<td>jl opr</td>
<td>sf</td>
</tr>
<tr>
<td></td>
<td>jnge opr</td>
<td></td>
</tr>
<tr>
<td>Branch on not less, or greater or equal (signed)</td>
<td>jnl opr</td>
<td>sf</td>
</tr>
<tr>
<td></td>
<td>jge opr</td>
<td></td>
</tr>
<tr>
<td>Branch on less or equal, or not greater (signed)</td>
<td>jle opr</td>
<td>(sf</td>
</tr>
<tr>
<td></td>
<td>jng opr</td>
<td></td>
</tr>
<tr>
<td>Branch on not less or equal, or greater (signed)</td>
<td>jnle opr</td>
<td>(sf</td>
</tr>
<tr>
<td></td>
<td>jg opr</td>
<td></td>
</tr>
</tbody>
</table>

Note: In all cases, if the branch is taken, new (ip) = (ip) + sign extended 8-bit displacement.

Flags: Not affected.

Addressing Modes: opr must represent a label that is within -128 to 128 bytes of the instruction.
EXAMPLE:

; if (b >= c) a = 0
L2: mov ax, b
    cmp ax, c
    jnae L3
    mov a, 0
    jmp L4

; else a = c
L3: mov ax, c
    mov a, ax

; if (a & 1) b = 1
L4: test a, 1
    jz L5
    mov b, 1

; while (a-- < 5) b++
L5: jmp L7
L6: inc b
    dec a
L7: cmp a, 5
    jb L6
Branch Instructions Application
Example

ESCP equ 1bh ;define ASCII code for ESC key

.model small

.stack 200h

.data
   old_key db ? ;remember last pressed key

.code

start:
   mov ax,@data ;set-up ds to access data
   mov ds,ax

read_key:
   mov ah,0 ;use BIOS interrupt 16h, service 0
   int 16h ;to read input from Keyboard
   cmp al,ESCP ;ESC key pressed?
   je exit ;if so, exit program

   cmp al,'9' ;al > 9?
   ja read_key ;if so, don't print it

   cmp al,'0' ;al < 0?
   jb read_key ;if so, don't print it

   cmp al,old_key ;Same key pressed?
   je read_key ;if so, don't print it
   mov old_key,al ;otherwise, store entered key into dram

   mov ah,0eh ;use BIOS interrupt 10h, service 0eh
   int 10h ;to print character on screen

   jmp read_key ;re-start the whole thing

exit: ;we get here only if ESC key pressed
   mov ax,4c00h ;use DOS interrupt 21h,service 0
   int 21h ;to terminate program

end start
Loop Instructions

These instructions transfer execution control to another point depending on the value of the \( cx \) register and/or conditional flags.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic and Format</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>( loop \ opr )</td>
<td>( (cx)!=0 )</td>
</tr>
<tr>
<td>Loop while zero, or equal</td>
<td>( loopz \ opr )</td>
<td>( zf =1 ) and ( (cx)!=0 )</td>
</tr>
<tr>
<td></td>
<td>( loope \ opr )</td>
<td>( (cx)!=0 )</td>
</tr>
<tr>
<td>Loop while nonzero or not equal</td>
<td>( loopnz \ opr )</td>
<td>( zf=0 ) and ( (cx)!=0 )</td>
</tr>
<tr>
<td></td>
<td>( loopne \ opr )</td>
<td>( (cx)!=0 )</td>
</tr>
<tr>
<td>Branch on CX zero</td>
<td>( jcxz \ opr )</td>
<td>( (cx)=0 )</td>
</tr>
</tbody>
</table>

Note: All except \( jcxz \), decrement \( cx \) without affecting flags before checking the Test Condition. In all cases, if the branch is taken, \( (ip) = (ip) + \) sign extended 8-bit displacement.

Flags: Not affected.

Addressing Modes: \( opr \) must represent a label that is within -128 to 128 bytes of the instruction.
EXAMPLE:

;add 4 (5 times)
    mov ax, 0
    mov cx, 5
L1: add ax, 4
    loop L1

;search for '//' on "buffer"
    mov si, -1
    mov cx, 80
L2: inc si
    cmp buffer[si], '
    loopne L2
    je L1

;skip L3 if CX=0
    jcxz L4

;wait for flag bit 0 = 1
;maximum wait is cx times
L3: test flag, 01h
    loopz L3
    jnz L1

;Delay 64K Times
    mov cx, 0
L4: loop L4
Branch Operations Exercise

Write an assembly program that will perform the following sequence:

1. Add variable \( a \) to variable \( b \).

2. If \( a \geq 0 \), assign 1 to \( c \), otherwise, assign 0.

3. Decrement \( b \).

4. If \( b > 0 \), go to step 1.

5. If \( c = 1 \), add \( a \) to \( b \).

6. If \( c = 0 \), subtract \( a \) from \( b \).

7. Repeat the above sequence 5 times.

*Variable \( c \) should be 8-bit wide; all other variables, 16-bit.