TERMS USED IN COMPARISON

4.2 Terms used in comparison

Above and Below used for comparing Unsigned numbers. Greater than and less than used when comparing signed numbers. All Intel microprocessors use this convention.

Accordingly, all the following statements are true.

- 95H is above 65H  Unsigned comparison - True
- 95H is less than 65H  Signed comparison – True (as 95H is negative, 65H is positive)
- 65H is below 95H  Unsigned comparison - True
- 65H is greater than 95H  Signed comparison - True

4.2.1 Jump based on multiple flags

Conditional Jumps based on multiple flags are used after a CMP (compare) instruction.

JBE / JNA instruction

‘Jump if Below or Equal’ or ‘Jump if Not Above’

<table>
<thead>
<tr>
<th>Jump if</th>
<th>No Jump if</th>
<th>Ex.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy = 1 OR Z= 1</td>
<td>Cy = 0 AND Z = 0</td>
<td>CMP BX, CX</td>
</tr>
<tr>
<td>Below OR Equal</td>
<td>Surely Above</td>
<td>JBE BX_BE</td>
</tr>
</tbody>
</table>
### 4.2.2 BX_BE (BX is Below or Equal) is a symbolic location

### 4.2.3 JNBE / JA instruction

‘Jump if Not (Below or Equal)’ or ‘Jump if Above’

<table>
<thead>
<tr>
<th>Jump if</th>
<th>No Jump if</th>
<th>Ex.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy = 0 AND Z = 0</td>
<td>Cy = 1 OR Z = 1</td>
<td>CMP BX, CX</td>
</tr>
<tr>
<td>Surely Above</td>
<td>Below OR Equal</td>
<td>JBE BX_BE</td>
</tr>
</tbody>
</table>

### 4.2.4 JLE / JNG instruction

‘Jump if Less than OR Equal’ or ‘Jump if Not Greater than’

<table>
<thead>
<tr>
<th>Jump if</th>
<th>No Jump if</th>
</tr>
</thead>
<tbody>
<tr>
<td>[(S=1 AND V=0) OR (S=0 AND V=0)] OR Z=1</td>
<td>[S=0 AND V=0] OR (S=1 AND V=1) AND Z=0</td>
</tr>
<tr>
<td>[(surely negative) or (wrong answer positive!)] OR Equal</td>
<td>[(surely positive) or (wrong answer negative!)] and not equal</td>
</tr>
</tbody>
</table>

i.e. \([S \text{ XOR } V=1] \text{ OR } Z=1\)  
i.e.\([S \text{ XOR } V=0] \text{ AND } Z=0\)
### JNLE / JG instruction

‘Jump if Not (Less than OR Equal)’ or ‘Jump if Greater than’

<table>
<thead>
<tr>
<th>Jump if</th>
<th>No Jump if</th>
</tr>
</thead>
<tbody>
<tr>
<td>(((S=0 \text{ AND } V=0) \text{ OR } (S=1 \text{ AND } V=1)) \text{ AND } Z=0)</td>
<td>(((S=1 \text{ AND } V=0) \text{ OR } (S=0 \text{ AND } V=1)) \text{ OR } Z=1)</td>
</tr>
<tr>
<td>[(surely positive) or (wrong answer negative!) and not equal]</td>
<td>[(surely negative) or (wrong answer positive!) or equal]</td>
</tr>
<tr>
<td>i.e. S XOR V=0 AND Z=0</td>
<td>i.e. S XOR V=1 OR Z=1</td>
</tr>
</tbody>
</table>

### JL / JNGE instruction

‘Jump if Less than’ or ‘Jump if NOT (Greater than or Equal)’

<table>
<thead>
<tr>
<th>Jump if</th>
<th>No Jump if</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[S=1 \text{ AND } V=0] \text{ OR } [S=0 \text{ AND } V=1]] (surely negative) or (wrong answer positive!)</td>
<td>[[S=0 \text{ AND } V=0] \text{ OR } [S=1 \text{ AND } V=1]] (surely positive) or (wrong answer negative!)</td>
</tr>
<tr>
<td>i.e. S XOR V=1</td>
<td>i.e. S XOR V=0</td>
</tr>
</tbody>
</table>

Note: When S=1, result cannot be 0
4.2.5 JNL / JGE instruction

‘Jump if Not Less than’ or ‘Jump if Greater than OR Equal’

\[
\begin{align*}
\text{Jump if} & \quad \text{No Jump if} \\
[S=0 \ \text{AND} \ V=0] \ \text{OR} \ (S=1 \ \text{AND} \ V=1) & \quad [S=1 \ \text{AND} \ V=0] \ \text{OR} \ (S=1 \ \text{AND} \ V=1) \\
\text{(surely positive) or (wrong answer negative!)} & \quad \text{(surely negative) or (wrong answer positive!)}
\end{align*}
\]

i.e. \( S \ \text{XOR} \ V = 0 \) \quad \text{i.e.} \( S \ \text{XOR} \ V = 1 \)

Note: When \( S = 0 \), result can be \( \geq 0 \)

Unconditional Jump instruction

Unconditional Jump Instruction

\[\downarrow \quad \downarrow\]

Near Jump or Intra segment Jump \quad Far Jump or Inter segment Jump

(Jump within the segment) \quad (Jump to a different segment)

Near Unconditional Jump instruction

\[\downarrow \quad \downarrow\]

Direct Jump (common) \quad Indirect Jump (uncommon)

2-bytes Short Jump (EB r8) \quad 3-bytes Long Jump (E9 r16) \quad 2 or more bytes
Range: \( \pm 2^7 \)  
Range: \( \pm 2^{15} \)  
Starting with FFH  
Range: complete segment

Three Near Jump and two Far Jump instructions have the same mnemonic JMP, but they have different opcodes

### 4.2.5 Short Jump Instruction

2 byte (EB r8) instruction with Range: -128 to +127 bytes

*For Backward jump:* Assembler knows the quantum of jump. Generates Short Jump code if \( \leq 128 \) bytes is the required jump. Generates code for Long Jump if \( >128 \) bytes is the required jump.

*For Forward jump:* Assembler doesn’t know jump quantum in pass 1. Assembler reserves 3 bytes for the forward jump instruction. If jump distance turns out to be \( >128 \) bytes, the instruction is coded as E9 r16 (E9H = Long jump code). If jump distance becomes \( \leq 128 \) bytes, the instruction is coded as EB r8 followed by code for NOP (E8H = Short jump code).
4.2.5 SHORT Assembler Directive

Assembler generates only 2 byte Short Jump code for forward jump, if the SHORT assembler directive is used.

JMP SHORT       SAME

Programmer should ensure that
the Jump distance is ≤127 bytes

SAME: MOV CX, DX

Long Jump instruction

3-byte (E9 r16) instruction with Range: -32768 to +32767 bytes

Long Jump can cover entire 64K bytes of Code segment

CS:0000H
CS:8000H

Long Jump can handle it as jump
quantum is ≤32767

FRWD = CS:FFFFH

JMP FRWD
Long Jump can handle it as jump quantum is <= 32767

\[
\begin{align*}
\text{BKWD} &= \text{CS:0000H} \\
\text{CS:8000H} &\quad \text{JMP BKWD} \\
\text{FRWD} &= \text{CS:FFFFH}
\end{align*}
\]

### 4.2.6 Long Jump or Short Jump?

**Can be treated as a small (20H) Backward Branch!**

- CS:0000H
- CS:0010H
- FRWD = CS:FFF0H
- CS:FFFFH

Jump distance =FFE0H. Too very long forward jump.

**Can be treated as a small (20H) Forward Branch!**

- BKWD = CS:0010H
- CS:FFF0H
- CS:FFFFH

Jump distance =FFE0H. Too very long backward jump.
4.2.7 Intra segment indirect Jump

It is also called Near Indirect Jump. It is not commonly used.

Instruction length: 2 or more bytes  Range: complete segment

Ex.1: JMP DX

If DX = 1234H, branches to CS:1234H. 1234H is not signed relative displacement.

Ex. 2: JMP wordptr 2000H[BX]

<table>
<thead>
<tr>
<th>If BX contents is 1234H</th>
<th>DS:3234H 5678H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branches to CS:5678H</td>
<td>DS:3236H AB22H</td>
</tr>
</tbody>
</table>

Far Jump instruction

4.2.8 Far Jump

Direct Jump (common)  Indirect Jump (uncommon)

5 bytes, opcode EA, 2 byte offset,  2 or more bytes,
2 byte segment value  starting with opcode FFH

Range: anywhere in memory  Range: anywhere in memory

As stated earlier, three Near Jump and two Far Jump instructions have the same mnemonic JMP but different opcodes.
4.2.9 Inter segment Direct Jump instruction

Also called Far Direct Jump. It is the common inter segment jump scheme
It is a 5 byte instruction. 1 byte opcode (EAH), 2 byte offset value, 2 byte segment value
Ex. JMP Far ptr LOC

4.2.10 Inter segment Indirect Jump instruction

Also called Far Indirect Jump. It is not commonly used. Instruction length depends on the way
jump location is specified. It can be a minimum of 2 bytes.

Ex. JMP DWORD PTR 2000H[BX]

If BX contents is 1234H branch takes place to location ABCDH:5678H. It is a 4-byte instruction.

```
| DS:3234H | 5678H |
| DS:3236H | ABCDH |
```

Iteration Instructions

Iteration instructions provide a convenient way to implement loops in a program

```
LOOP    LOOPZ or LOOPE    LOOPNZ or LOOPNE    JCXZ
```