Short vs. Near Jumps

- Jumps use relative addressing
  - assembler computes an offset from address of current instruction.
  - produces relocatable code
- SHORT jumps use 8-bit offsets
  - target label within -128 bytes to +127 bytes
- NEAR jumps use 32-bit offsets
  - target label within $-2^{32}$ bytes to $+2^{32} - 1$ bytes
SHORT JUMPS VS NEAR JUMPS

• Some assemblers determine SHORT vs NEAR jumps automatically, but some do not.
• explicitly specify SHORT jumps
  • jmp SHORT somewhere
• explicitly specify NEAR jumps
  • jge NEAR somewhere

Short Jumps vs. Near Jumps

• Some assemblers determine SHORT vs NEAR jumps automatically, but some do not.
• explicitly specify SHORT jumps
  jmp SHORT somewhere
• explicitly specify NEAR jumps
  jge NEAR somewhere

/ File: jmp.exe
/* demonstrating near and short jumps */

; section .text
 global _start
_start: nop
 ; initialize
start: mov eax, 17  ; eax = 17
 cmp eax, 43  ; 17 - 43 is ...
 jmp exit  ; exit if 17 >= 42
 jge near exit
 jmp short exit
 jmp near exit

exit: mov edx, 0  ; exit code. Oneway
 mov eax, 1  ; Exit
 int 0x80  ; Call kernel.
Logical (Bit Manipulation) Instructions

- **AND:** used to clear bits (store 0 in the bits):
  - To clear the lower 4 bits of the AL register:
    ```
    AND AL, F0h
    0001 0000
    0000 0110
    1101 0000
    ```

- **OR:** used to set bits (store 1 in the bits):
  - To set the lower 4 bits of the AL register:
    ```
    OR AL, 0Fh
    0000 1111
    0001 0000
    1101 1111
    ```

- **NOT:** flip all the bits

- **Shift and Rotate instructions** move bits around
### Instruction Set Reference

#### ADD—Logical Addition

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add</td>
<td>A</td>
<td>B</td>
<td>A + B</td>
</tr>
<tr>
<td>2</td>
<td>Add</td>
<td>B</td>
<td>A</td>
<td>B + A</td>
</tr>
<tr>
<td>3</td>
<td>Add</td>
<td>A</td>
<td>C</td>
<td>A + C</td>
</tr>
<tr>
<td>4</td>
<td>Add</td>
<td>B</td>
<td>C</td>
<td>B + C</td>
</tr>
<tr>
<td>5</td>
<td>Add</td>
<td>A</td>
<td>D</td>
<td>A + D</td>
</tr>
<tr>
<td>6</td>
<td>Add</td>
<td>B</td>
<td>D</td>
<td>B + D</td>
</tr>
<tr>
<td>7</td>
<td>Add</td>
<td>A</td>
<td>E</td>
<td>A + E</td>
</tr>
<tr>
<td>8</td>
<td>Add</td>
<td>B</td>
<td>E</td>
<td>B + E</td>
</tr>
</tbody>
</table>

### Instruction Set Reference

#### OR—Logical Inclusive OR

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OR</td>
<td>A</td>
<td>B</td>
<td>A OR B</td>
</tr>
<tr>
<td>2</td>
<td>OR</td>
<td>B</td>
<td>A</td>
<td>B OR A</td>
</tr>
<tr>
<td>3</td>
<td>OR</td>
<td>A</td>
<td>C</td>
<td>A OR C</td>
</tr>
<tr>
<td>4</td>
<td>OR</td>
<td>B</td>
<td>C</td>
<td>B OR C</td>
</tr>
<tr>
<td>5</td>
<td>OR</td>
<td>A</td>
<td>D</td>
<td>A OR D</td>
</tr>
<tr>
<td>6</td>
<td>OR</td>
<td>B</td>
<td>D</td>
<td>B OR D</td>
</tr>
<tr>
<td>7</td>
<td>OR</td>
<td>A</td>
<td>E</td>
<td>A OR E</td>
</tr>
<tr>
<td>8</td>
<td>OR</td>
<td>B</td>
<td>E</td>
<td>B OR E</td>
</tr>
</tbody>
</table>

### Instruction Set Reference

#### NOT—One's Complement Negate

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Operand 1</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOT</td>
<td>A</td>
<td>~A</td>
</tr>
<tr>
<td>2</td>
<td>NOT</td>
<td>B</td>
<td>~B</td>
</tr>
<tr>
<td>3</td>
<td>NOT</td>
<td>C</td>
<td>~C</td>
</tr>
<tr>
<td>4</td>
<td>NOT</td>
<td>D</td>
<td>~D</td>
</tr>
<tr>
<td>5</td>
<td>NOT</td>
<td>E</td>
<td>~E</td>
</tr>
</tbody>
</table>

### Instruction Set Reference

#### NOP—No Operation

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Operation</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>XORI</td>
<td>X (or) X</td>
</tr>
</tbody>
</table>
**Figure 7.7. SHL/ISAL Instruction Operation**

**Figure 7.8. SHR Instruction Operation**
### Initial State (Positive Operand) | Operands | CF
---|---|---
| | | X

After 1-bit SAR Instruction:

| | | 1 |
---|---|---

### Initial State (Negative Operand) | Operands | CF
---|---|---
| | | X

After 1-bit SAR Instruction:

| | | 1 |
---|---|---

### Figure 7-8. SAR Instruction Operation

---

<table>
<thead>
<tr>
<th>INSTRUCTION SET REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLLSR/ROLN/ROLN—Rotate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>ROLLSR</td>
<td>Rotate left, sign-extend one bit</td>
</tr>
<tr>
<td>010</td>
<td>ROLN</td>
<td>Rotate left, no sign-extend</td>
</tr>
<tr>
<td>001</td>
<td>ROLSR</td>
<td>Rotate right, sign-extend one bit</td>
</tr>
<tr>
<td>000</td>
<td>ROLN</td>
<td>Rotate right, no sign-extend</td>
</tr>
</tbody>
</table>

### INSTRUCTION SET REFERENCE

#### ROLLSR/ROLN/ROLN—Rotate

**Description:**

- **ROLLSR**: Rotate left, sign-extend one bit.
- **ROLN**: Rotate left, no sign-extend.
- **ROLSR**: Rotate right, sign-extend one bit.
- **ROLN**: Rotate right, no sign-extend.

**Usage:**

- **ROLLSR**: Used to rotate the leftmost bit of an operand to the rightmost position and append the sign bit to the left.
- **ROLN**: Used to rotate the leftmost bit of an operand to the rightmost position without sign-extend.
- **ROLSR**: Used to rotate the rightmost bit of an operand to the leftmost position and append the sign bit to the right.
- **ROLN**: Used to rotate the rightmost bit of an operand to the leftmost position without sign-extend.

**Flags:**

- **CF**: Carry flag
- **PF**: Parity flag
- **AF**: Auxiliary carry flag
- **ZF**: Zero flag
- **SF**: Sign flag
- **OF**: Overflow flag

**Implementation:**

- The **ROLLSR** instruction rotates the leftmost bit of the operand to the rightmost position and appends the sign bit to the left. It sets the carry flag if the sign bit is flipped.
- The **ROLN** instruction rotates the leftmost bit of the operand to the rightmost position without sign-extend. It does not set the carry flag.
- The **ROLSR** instruction rotates the rightmost bit of the operand to the leftmost position and appends the sign bit to the right. It sets the carry flag if the sign bit is flipped.
- The **ROLN** instruction rotates the rightmost bit of the operand to the leftmost position without sign-extend. It does not set the carry flag.

### Additional Information

- **INSTRUCTION SET REFERENCE**

- **ROLLSR/ROLN/ROLN—Rotate**

#### INSTRUCTION SET REFERENCE

**Description:**

- **ROLLSR**: Rotate left, sign-extend one bit.
- **ROLN**: Rotate left, no sign-extend.
- **ROLSR**: Rotate right, sign-extend one bit.
- **ROLN**: Rotate right, no sign-extend.

**Usage:**

- **ROLLSR**: Used to rotate the leftmost bit of an operand to the rightmost position and append the sign bit to the left.
- **ROLN**: Used to rotate the leftmost bit of an operand to the rightmost position without sign-extend.
- **ROLSR**: Used to rotate the rightmost bit of an operand to the leftmost position and append the sign bit to the right.
- **ROLN**: Used to rotate the rightmost bit of an operand to the leftmost position without sign-extend.

**Flags:**

- **CF**: Carry flag
- **PF**: Parity flag
- **AF**: Auxiliary carry flag
- **ZF**: Zero flag
- **SF**: Sign flag
- **OF**: Overflow flag

**Implementation:**

- The **ROLLSR** instruction rotates the leftmost bit of the operand to the rightmost position and appends the sign bit to the left. It sets the carry flag if the sign bit is flipped.
- The **ROLN** instruction rotates the leftmost bit of the operand to the rightmost position without sign-extend. It does not set the carry flag.
- The **ROLSR** instruction rotates the rightmost bit of the operand to the leftmost position and appends the sign bit to the right. It sets the carry flag if the sign bit is flipped.
- The **ROLN** instruction rotates the rightmost bit of the operand to the leftmost position without sign-extend. It does not set the carry flag.

### Additional Information

- **INSTRUCTION SET REFERENCE**

- **ROLLSR/ROLN/ROLN—Rotate**
Figure 7.11. ROL, ROR, RCL, and RCR Instruction Operations
Example Using AND, OR, & SHL

- Copy bits 4-7 of BX to bits 8-11 of AX
  - AX = 0110 1011 1001 0110
  - BX = 1101 0011 1100 0001

1. Clear bits 8-11 of AX & all but bits 4-7 of BX using AND instructions
   - AX = 0110 0000 1001 0110
   - BX = 0000 0000 1100 0000

2. Shift bits 4-7 of BX to the desired position using a SHL instruction
   - AX = 0110 0000 1001 0110
   - BX = 0000 1100 0000 0000

3. "Copy" bits 4-7 of BX to AX using an OR instruction
   - AX = 0110 1100 1001 0110
   - BX = 0000 1100 0000 0000

More Arithmetic Instructions

- NEG: two's complement negation of operand
- MUL: unsigned multiplication
  - Multiply AL with r/m8 and store product in AX
  - Multiply AX with r/m16 and store product in DX:AX
  - Multiply EAX with r/m32 and store product in EDX:EAX
  - Immediate operands are not supported.
  - CF and OF cleared if upper half of product is zero.
- IMUL: signed multiplication
  - Use with signed operands
  - More addressing modes supported
- DIV: unsigned division
Indexed Addressing Modes
Indexed Addressing Modes

- Operands of the form: [ESI + ECX*4 + DISP]
- ESI = Base Register
- ECX = Index Register
- 4 = Scale factor
- DISP = Displacement
- The operand is in memory
- The address of the memory location is ESI + ECX*4 + DISP
Typical Uses for Indexed Addressing

- Base + Displacement
  - access character in a string or field of a record
  - access a local variable in function call stack
- Index*Scale + Displacement
  - access items in an array where size of item is 2, 4 or 8 bytes
- Base + Index + Displacement
  - access two dimensional array (displacement has address of array)
  - access an array of records (displacement has offset of field in a record)
- Base + (Index*Scale) + Displacement
  - access two dimensional array where size of item is 2, 4 or 8 bytes
References

- Some figures and diagrams from IA-32 Intel Architecture Software Developer's Manual, Vols 1-3
  <http://developer.intel.com/design/Pentium4/manuals/>