Conditional Processing

Muhammad Afzaal
m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 6
  - Section 6.1
  - Section 6.3

Boolean and Comparison Instructions

- Four basic Boolean operations
  - AND
  - OR
  - XOR
  - NOT
- Boolean operations can be carried out at bit-level

AND Instruction (1/2)

- Performs a bitwise AND operation between matching bits in two operands
- Syntax is `AND dest, src`
- The permitted operand combinations are
  - `AND reg, reg`
  - `AND reg, mem`
  - `AND reg, imm`
  - `AND mem, reg`
  - `AND mem, imm`

x	y	x^y
0	0	0
0	1	0
1	0	0
1	1	1

AND Instruction (2/2)

- Can be used for bit masking
  - ... We can clear certain bits from a value by `AND`ing those bits with 0
- Difference of only 1 bit (bit number 5) in lower case and upper case alphabets
  - By `AND`ing any smaller alphabet with 11011111, it can be converted to upper case
- Always clears OF and CF
- Modifies SF, ZF, PF

OR Instruction (1/2)

- Performs a Boolean OR operation between matching bits of two operands
- Syntax is `OR dest, src`
- Permitted operand combinations are same as of AND
  - `OR reg, reg`
  - `OR reg, mem`
  - `OR reg, imm`
  - `OR mem, reg`
  - `OR mem, imm`

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

OR Instruction (2/2)

- Useful when we need to set one or more bits in an operand without affecting other bits
- Always clears OF and CF
- Modifies the values of SF, ZF and PF
- If a number is ORed with itself, values of ZF and SF indicate following information

ZF	SF	Value of operand is …
Clear	Clear	Greater than zero
Set	Clear	Equal to zero
Clear	Set	Less than zero

XOR Instruction (1/2)

- Performs a Boolean exclusive-OR between matching bits in two operands
- Syntax is `XOR dest, src`
- Operand combinations are same as of `AND` and `OR`
  - `XOR reg, reg`
  - `XOR reg, mem`
  - `XOR reg, imm`
  - `XOR mem, reg`
  - `XOR mem, imm`

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Instruction (2/2)

- A bit XORed with 0 retains its value, and a bit XORed with 1 is toggled
- XOR reverses itself when applied twice to the same operand
  - This property makes XOR ideal for a simple form of symmetric encryption
- Always clears OF and CF
- Modifies SF, ZF and PF

NOT Instruction

- Toggles all bits in an operand
- Result is called one's complement
- NOT takes only one operand with following formats
  - `NOT reg`
  - `NOT mem`
- `NOT` **does not affect any flags**

CMP Instruction (1/2)

- Comparison is performed to replicate conditional branching/loops
- CMP instruction performs an implied subtraction of a source operand from a destination operand
- Neither operand is changed
- Syntax is `CMP dest, src`
- Uses the same operand combinations as of `AND` and `OR`
- CMP instruction affects the flags in following way
  - Changes OF, SF, ZF, CF, AF and PF

CMP Instruction (2/2)

- when compared two unsigned operands

CMP Results	ZF	CF
Dest < Src	0	1
Dest > Src	0	0
Dest = Src	1	0

- When compared two signed operands then

CMP Results	Flags
Dest < Src	SF ≠ OF
Dest > Src	SF = OF
Dest = Src	ZF = 1

Setting/Clearing Individual Flags

- Zero Flag (ZF)
  - *To set, `AND` an operand with 0*
  - *To clear, `OR` an operand with 1*
- Sign Flag (SF)
  - To set, `OR` MSB of an operand with 1
  - To clear, `AND` MSB of an operand with 0
- Carry Flag (CF)
  - To set, use `STC` instruction
  - To clear, use `CLC` instruction
- Overflow Flag (OF)
  - To set, add two positive values that produce negative sum
  - To clear, `OR` an operand with 0

Conditional Jumps (1/2)

- No high-level logic structures such as if-then-else in IA-32 instruction set
- In assembly language, a combination of comparisons and jumps can be used to implement any logic structure
- First, an operation such as CMP, AND or SUB etc. modifies the status flags
- Second, a conditional jump instruction tests the flags and causes a branch to new address

Conditional Jumps (2/2)

- Example 1

```
CMP al, 0
JZ L1 ;jump if ZF=1
.
.
.
L1:
.
.
```

- Example 1

```
CMP al, 0
JNZ L1 ;jump if ZF=0
.
.
.
L1:
.
.
```

J*COND* Instruction

- A conditional jump instruction branches to a destination label when a status flag condition is true
- If the flag condition is false, the instruction immediately after the conditional jump is executed
- Syntax is `JCOND dest`
- *COND* refers to a flag identifying the state of one or more flags

How to Use Conditional Jump?

- First an arithmetic/logic operation is performed
- After this operation, status flags in EFLAGS register are set/cleared
- Now jump instruction can be performed based upon the value of flags

Types of Conditional Jump Instructions

- The conditional jump instructions can be divided into four groups
  - Jumps based on specific flag values
  - Jumps based on equality between operands or the value of CX
  - Jumps based on comparisons of unsigned operands
  - Jumps based on comparisons of signed operands

Jumps Based on Flag Values

Mnemonic	Description	Flags/Registers
JZ	**J**ump if **Z**ero	ZF = 1
JNZ	**J**ump if **N**ot **Z**ero	ZF = 0
JC	**J**ump if **C**arry	CF = 1
JNC	**J**ump if **N**ot **C**arry	CF = 0
JO	**J**ump if **O**verflow	OF = 1
JNO	**J**ump if **N**ot **O**verflow	OF = 0
JS	**J**ump if **S**igned	SF = 1
JNS	**J**ump if **N**ot **S**igned	SF = 0
JP	**J**ump if **P**arity	PF = 1
JNP	**J**ump if **N**ot **P**arity	PF = 0

Jumps Based on Equality

- Jumps based on equality are taken either
  - Two operands are compared with CMP instruction or
  - Based on the value of CX or ECX

Mnemonic	Description
JE	**J**ump if **E**qual
JNE	**J**ump if **N**ot **E**qual
JCXZ	**J**ump if **CX** = 0
JECXZ	**J**ump if **ECX** = 0

Jumps Based on Unsigned Comparisons

`CMP unsigned_dest, unsigned_src`

`Junsigned`

Mnemonic	Description
JA	**J**ump if **A**bove (if dest > src)
JNBE	**J**ump if **N**ot **B**elow or **E**qual (same as JA)
JAE	**J**ump if **A**bove or **E**qual (if dest ≥ src)
JNB	**J**ump if **N**ot **B**elow (same as JAE)
JB	**J**ump if **B**elow (if dest < src)
JNAE	**J**ump if **N**ot **A**bove or **E**qual (same as JB)
JBE	**J**ump if **B**elow or **E**qual (if dest ≤ src)
JNA	**J**ump if **N**ot **A**bove (same as JBE)

Jumps Based on Signed Comparisons

```
CMP signed_dest, signed_src
Jsigned
```

Mnemonic	Description
JG	**J**ump if **G**reater (if dest > src)
JNLE	**J**ump if **N**ot **L**ess than or **E**qual (same as JG)
JGE	**J**ump if **G**reater than or **E**qual (if dest ≥ src)
JNL	**J**ump if **N**ot **L**ess (same as JGE)
JL	**J**ump if **L**ess (if dest < src)
JNGE	**J**ump if **N**ot **G**reater than or **E**qual (same as JL)
JLE	**J**ump if **L**ess than or **E**qual (if dest ≤ src)
JNG	**J**ump if **N**ot **G**reater (same as JLE)

Conditional Loop Instructions (1/2)

- `LOOPZ` (LOOP if Zero)
  - Works just like `LOOP` with one additional condition that ZF must be set in order to transfer control to destination label
- `LOOPE` (LOOP if Equal)
  - Equivalent to LOOPZ
- LOOPZ and LOOPE perform the following tasks when executed
  - ECX = ECX – 1
  - If ECX > 0 and ZF = 1, jump to destination

Conditional Loop Instructions (2/2)

- LOOPNZ (LOOP if Not Zero)
  - Counterpart of LOOPZ
- LOOPNE (LOOP if Not Equal)
  - Same as LOOPNZ
- LOOPNZ and LOOPNE perform the following tasks when executed
  - ECX = ECX – 1
  - If ECX > 0 and ZF = 0, jump to destination