Logical and Bit Operations

Chapter 9

S. Dandamudi

Outline

- Logical instructions
  - AND
  - OR
  - XOR
  - NOT
  - TEST
- Shift instructions
  - Logical shift instructions
  - Arithmetic shift instructions
- Rotate instructions
  - Rotate without carry
  - Rotate through carry
- Logical expressions in high-level languages
  - Representation of Boolean data
  - Logical expressions
  - Bit manipulation
  - Evaluation of logical expressions
- Bit instructions
  - Bit test and modify instructions
  - Bit scan instructions
- Illustrative examples

Logical Instructions

- Logical instructions operate on bit-by-bit basis
- Five logical instructions:
  - AND
  - OR
  - XOR
  - NOT
  - TEST
- All logical instructions affect the status flags

Logical Instructions (cont'd)

- Since logical instructions operate on a bit-by-bit basis, no carry or overflow is generated
- Logical instructions
  - Clear carry flag (CF) and overflow flag (OF)
  - AF is undefined
- Remaining three flags record useful information
  - Zero flag
  - Sign flag
  - Parity flag

Logical Instructions (cont'd)

AND instruction

- Format

```
and     destination,source
```

- Usage
  - To support compound logical expressions and bitwise AND operation of HLLs
  - To clear one or more bits of a byte, word, or doubleword
  - To isolate one or more bits of a byte, word, or doubleword

Logical Instructions (cont'd)

OR instruction

- Format

```
or     destination,source
```

- Usage
  - To support compound logical expressions and bitwise OR operation of HLLs
  - To set one or more bits of a byte, word, or doubleword
  - To paste one or more bits of a byte, word, or doubleword

Logical Instructions (cont'd)

XOR instruction

- Format

```
xor     destination,source
```

- Usage
  - To support compound logical expressions of HLLs
  - To toggle one or more bits of a byte, word, or doubleword
  - To initialize registers to zero
    - Example: `xor AX,AX`

Logical Instructions (cont'd)

NOT instruction

- Format

```
not     destination
```

- Usage
  - To support logical expressions of HLLs
  - To complement bits
    - Example: 2's complement of an 8-bit number

```
not     AL
inc     AL
```

Logical Instructions (cont'd)

TEST instruction

- Format

```
test    destination,source
```

  - TEST is a non-destructive AND operation
    - Result is not written in **destination**
    - Similar in spirit to **cmp** instruction
- Usage
  - To test bits
    - Example:

```
test    AL,1
jz      even_number    ; else odd number
```

Shift Instructions

- Two types of shift instructions
  - Logical shift instructions
    - **`shl (SHift Left)`**
    - **`shr (SHift Right)`**
    - Another interpretation:
      - Logical shift instructions work on unsigned binary numbers
  - Arithmetic shift instructions
    - **`sal (Shift Arithmetic Left)`**
    - **`sar (Shift Arithmetic Right)`**
    - Another interpretation:
      - Arithmetic shift instructions work on signed binary numbers

Shift Instructions (cont'd)

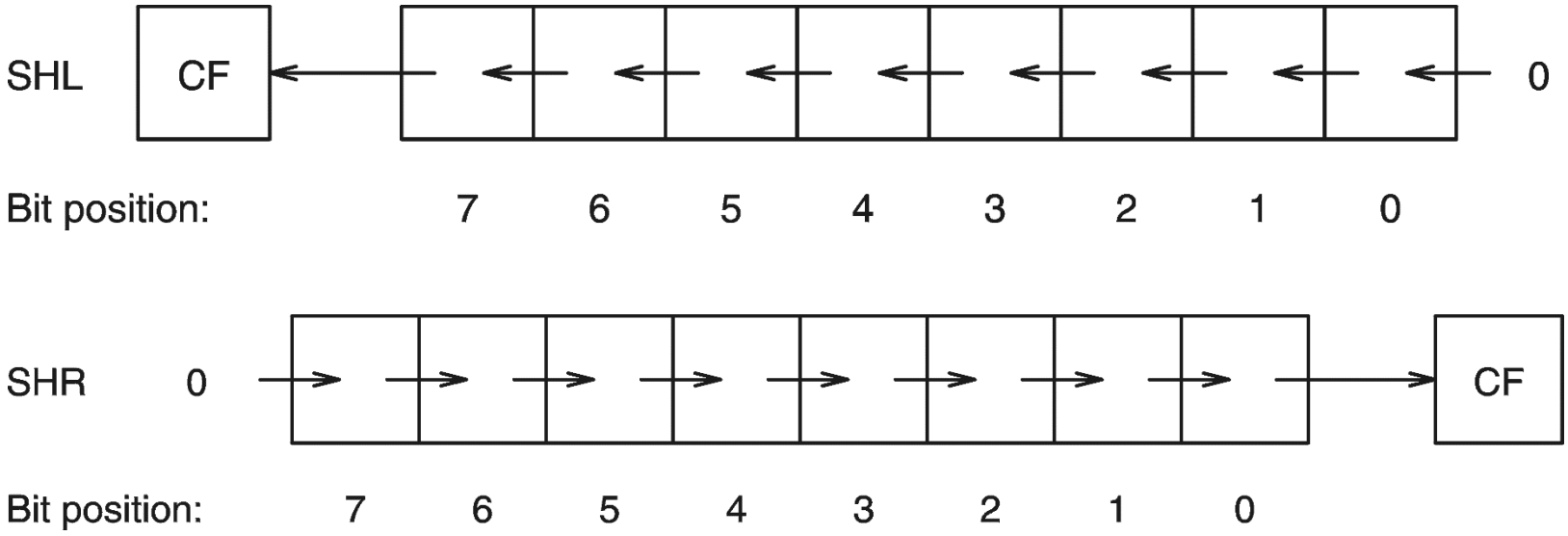
- Effect on flags
  - Auxiliary flag (AF): undefined
  - Zero flag (ZF) and parity flag (PF) are updated to reflect the result
  - Carry flag
    - Contains the last bit shifted out
  - Overflow flag
    - For multibit shifts
      - Undefined
    - For single bit shifts
      - OF is set if the sign bit has changed as a result of the shift
      - Cleared otherwise

Logical Shift Instructions

- General format

```
shl     destination,count
shr     destination,count
```

`destination` can be an 8-, 16-, or 32-bit operand located either in a register or memory



Logical Shift Instructions (cont'd)

- Two versions

```
shl/shr    destination,count
shl/shr    destination,CL
```

  - First format directly specifies the count value
    - Count value should be between 0 and 31
    - If a greater value is specified, Pentium takes only the least significant 5 bits as the count value
  - Second format specifies count indirectly through CL
    - CL contents are not changed
    - Useful if count value is known only at the run time as opposed at assembly time
      - Ex: Count is received as an argument in a procedure call

Logical Shift Instructions (cont'd)

- Usage
  - Bit manipulation

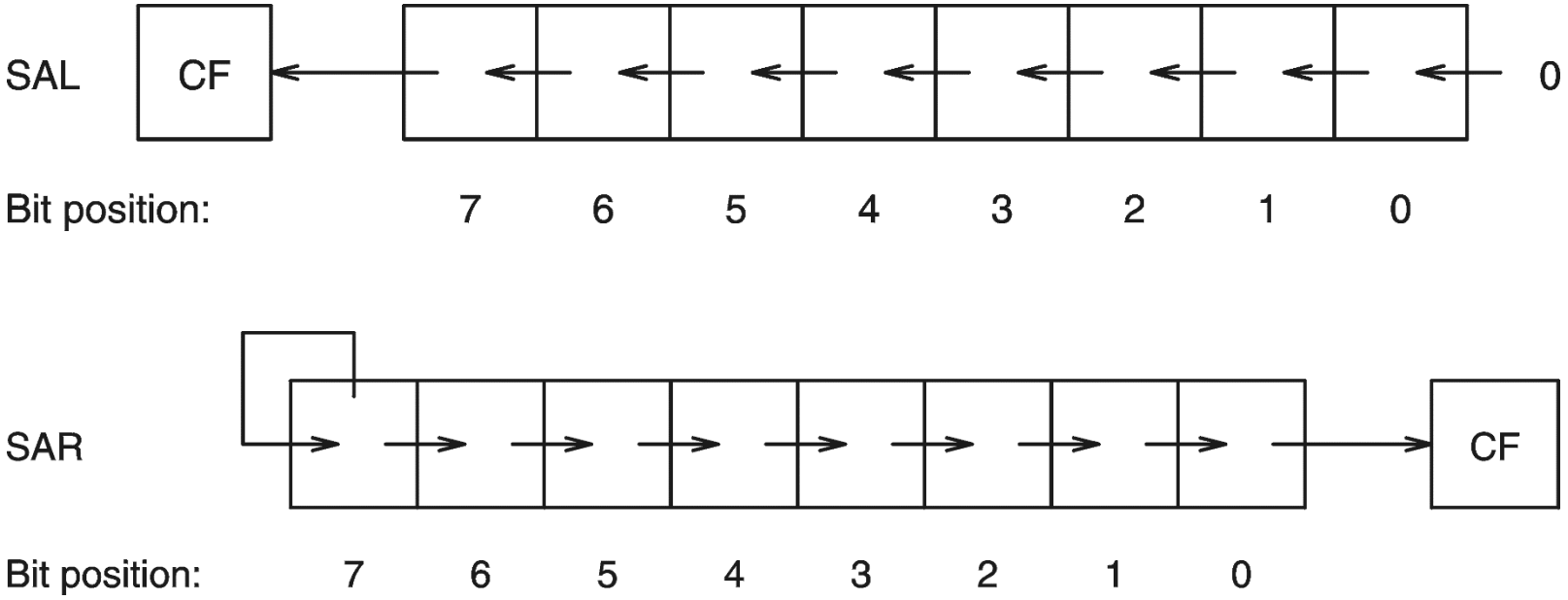
```
; AL contains the byte to be encrypted
mov    AH,AL
shl    AL,4      ; move lower nibble to upper
shr    AH,4      ; move upper nibble to lower
or     AL,AH     ; paste them together
; AL has the encrypted byte
```

  - Multiplication and division
    - Useful to multiply (left shift) or divide (right shift) by a power of 2
    - More efficient than using multiply/divide instructions

Arithmetic Shift Instructions

- Two versions as in logical shift

```
sal/sar     destination,count
sal/sar     destination,CL
```



Double Shift Instructions

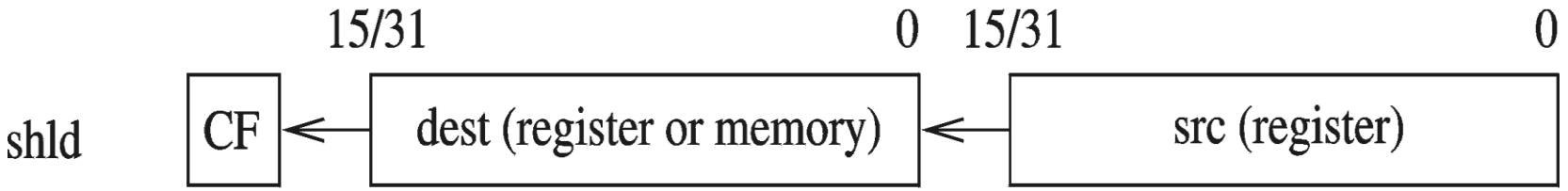
- Double shift instructions work on either 32- or 64-bit operands
- Format
  - Takes three operands

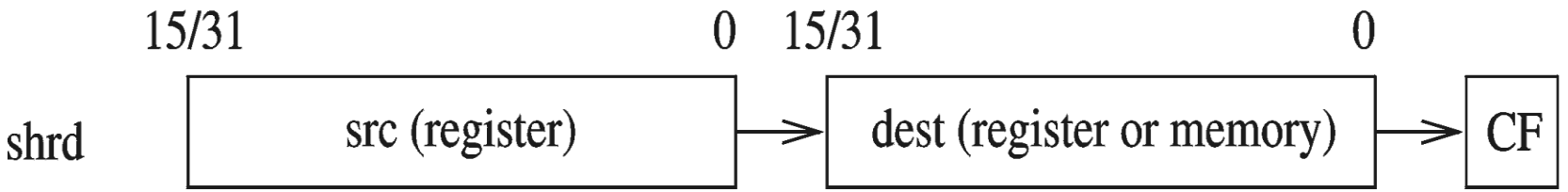
```
shld    dest,src,count  ; left-shift
shrd    dest,src,count  ; right-shift
```

  - **`dest`** can be in memory or register
  - **`src`** must be a register
  - **`count`** can be an immediate value or in CL as in other shift instructions

Double Shift Instructions (cont'd)

- **`src`** is not modified by the doubleshift instruction
- Only **`dest`** is modified
- Shifted out bit goes into the carry flag





Rotate Instructions

- A problem with the shift instructions
  - Shifted out bits are lost
  - Rotate instructions feed them back

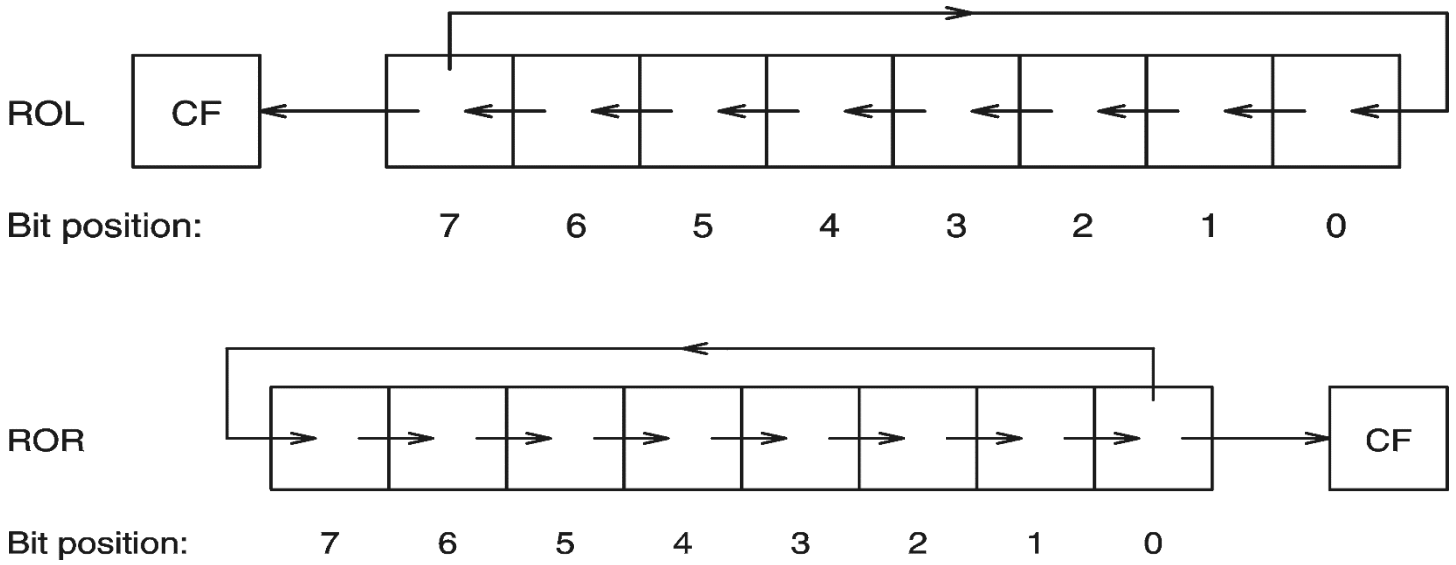
- Two types of rotate instructions
  - Rotate without carry
    - Carry flag is not involved in the rotate process
  - Rotate through carry
    - Rotation involves the carry flag

Rotate Without Carry

- General format

```
rol     destination,count
ror     destination,count
```

count can be an immediate value or in CL (as in shift)

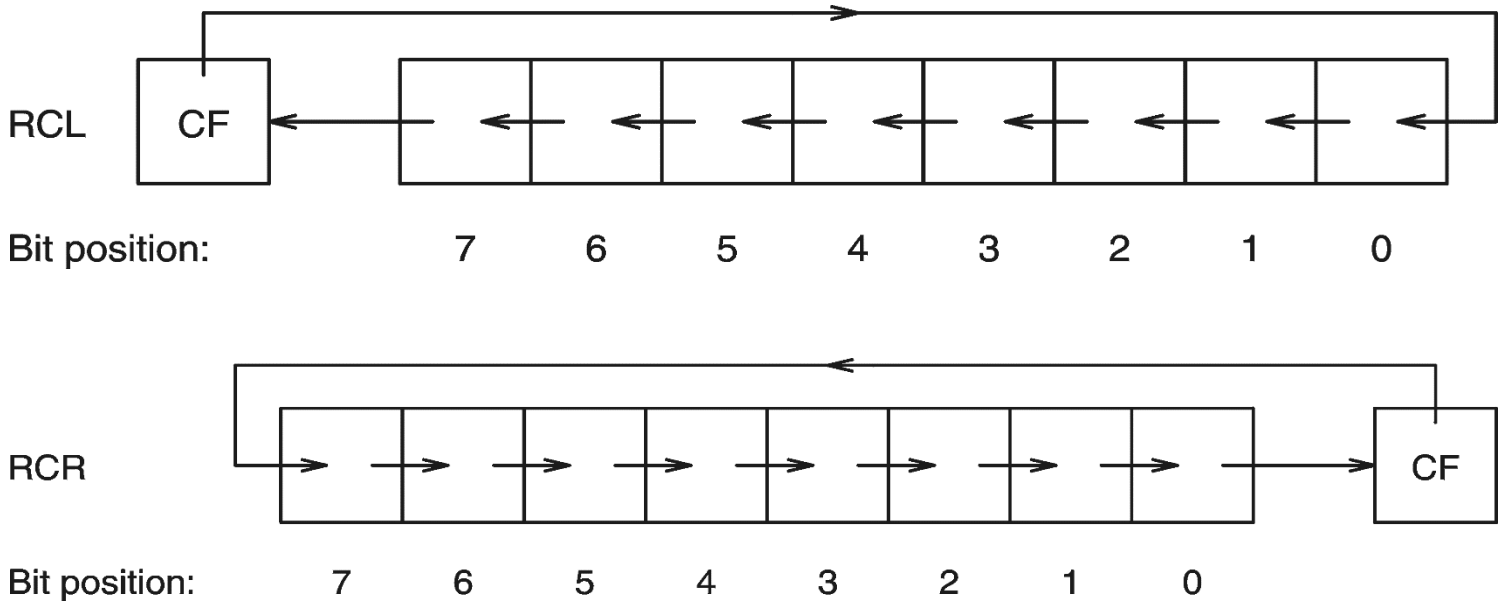


Rotate Through Carry

- General format

```
rcl     destination,count
rcr     destination,count
```

count can be an immediate value or in CL (as in shift)



Rotate Through Carry (cont'd)

- Only two instructions that take CF into account
  - This feature is useful in multiword shifts
- Example: Shifting 64-bit number in EDX:EAX
  - Rotate version

```
    mov     ECX,4         ; 4 bit shift
shift_left:
    shl     EAX,1       ; moves leftmost bit of EAX to CF
    rcl     EDX,1       ; CF goes to rightmost bit of EDX
    loop    shift_left
```

  - Double shift version

```
    shld    EDX,EAX,4 ; EAX is unaffected by shld
    shl     EAX,4
```

Logical Expressions in HLLs

- Representation of Boolean data
  - Only a single bit is needed to represent Boolean data
  - Usually a single byte is used
    - For example, in C
      - All zero bits represents *false*
      - A non-zero value represents *true*
- Logical expressions
  - Logical instructions AND, OR, etc. are used
- Bit manipulation
  - Logical, shift, and rotate instructions are used

Evaluation of Logical Expressions

- Two basic ways
  - Full evaluation
    - Entire expression is evaluated before assigning a value
    - PASCAL uses full evaluation
  - Partial evaluation
    - Assigns as soon as the final outcome is known without blindly evaluating the entire logical expression
    - Two rules help:
      - **`cond1 AND cond2`**
        - If **`cond1`** is false, no need to evaluate **`cond2`**
      - **`cond1 OR cond2`**
        - If **`cond1`** is true, no need to evaluate **`cond2`**

Evaluation of Logical Expressions (cont'd)

- Partial evaluation
  - Used by C
- Useful in certain cases to avoid run-time errors
- Example

```
if ((X > 0) AND (Y/X > 100))
```

  - If x is 0, full evaluation results in divide error
  - Partial evaluation will not evaluate **`(Y/X > 100)`** if **`X = 0`**
- Partial evaluation is used to test if a pointer value is NULL before accessing the data it points to

Bit Instructions

- Bit Test and Modify Instructions
  - Four bit test instructions
  - Each takes the position of the bit to be tested

Instruction	Effect on the selected bit
bt (Bit Test)	No effect
bts (Bit Test and Set)	selected bit ← 1
btr (Bit Test and Reset)	selected bit ← 0
btc (Bit Test and Complement)	selected bit ← NOT(selected bit)

Bit Instructions (cont'd)

- All four instructions have the same format
- We use **`bt`** to illustrate the format

```
bt      operand,bit_pos
```

  - **`operand`** is word or doubleword
    - Can be in memory or a register
  - **`bit_pos`** indicates the position of the bit to be tested
    - Can be an immediate value or in a 16- or 32-bit register

- Instructions in this group affect only the carry flag
    - Other five flags are undefined following a bit test instruction

Bit Scan Instructions

- These instructions scan the operand for a 1 bit and return the bit position in a register
- Two instructions

```
bsf     dest_reg,operand      ;bit scan forward
bsr     dest_reg,operand      ;bit scan reverse
```

  - **operand** can be a word or doubleword in a register or memory
  - **dest_reg** receives the bit position
    - Must be a 16- or 32-bit register
  - Only ZF is updated (other five flags undefined)
    - ZF = 1 if all bits of operand are 0
    - ZF = 0 otherwise (position of first 1 bit in dest_reg)

Illustrative Examples

- Example 1
  - Multiplication using shift and add operations
    - Multiplies two unsigned 8-bit numbers
      - Uses a loop that iterates 8 times
- Example 2
  - Same as Example 1 (efficient version)
    - We loop only for the number of 1 bits
      - Uses bit test instructions
- Example 3
  - Conversion of octal to binary