Shift and Rotate Instructions

Muhammad HabibUllah
habib.wattoo@nu.edu.pk

Book Chapter

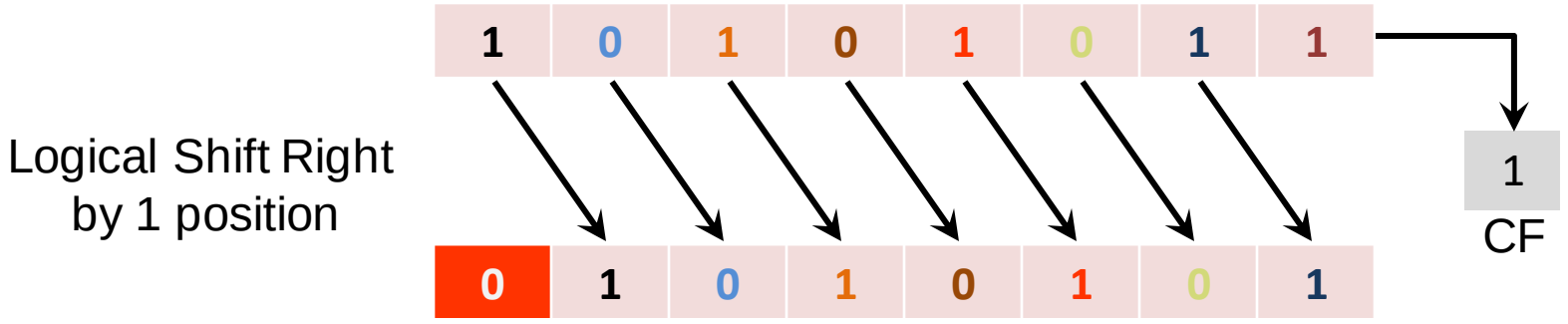
- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 7
  - Section 7.2
  - Section 7.4
- Chapter 9
  - Section 9.2

Shift and Rotate Instructions (1/2)

- Shifting an operand means moving bits right or left from their original positions inside the operand
- Rotating an operand fills the bit positions of empty end with the bits gone out on the other end of operand
- Shift and Rotate instructions affect the Carry and Overflow flags
- Two ways to shift an operand's bits
  - Logical Shift
  - Arithmetic Shift

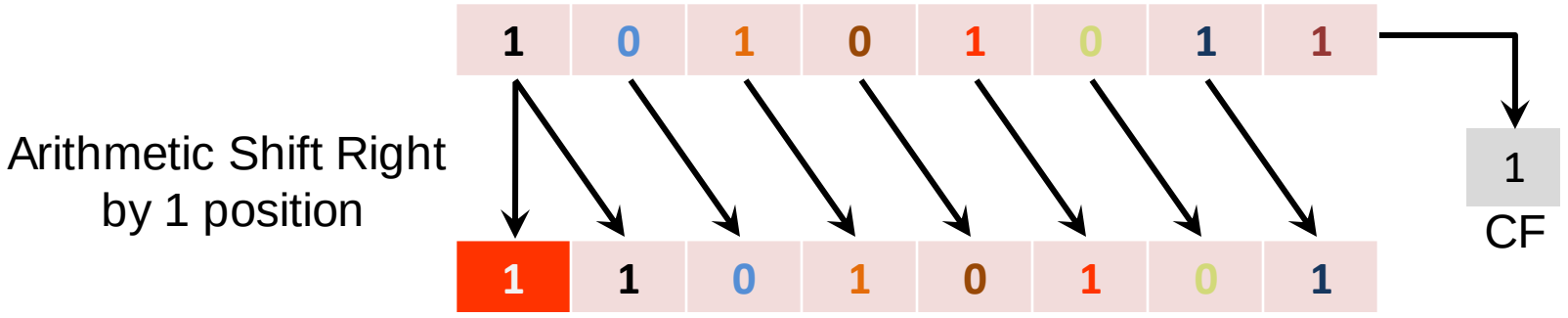
Logical Shift

- Fills the newly created bit positions with zero
- Each bit value is shifted to left or right bit position



Arithmetic Shift

- Newly created bit is filled with a copy of sign-bit
- Preserves the number's sign-bit

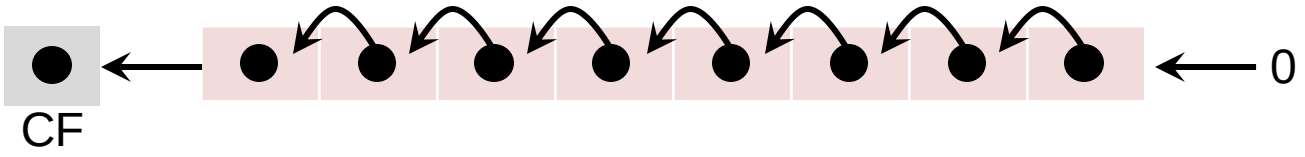


Shift and Rotate Instructions (2/2)

Instruction Mnemonic	Description
SHL	Shift Left
SHR	Shift Right
SAL	Shift Arithmetic Left
SAR	Shift Arithmetic Right
ROL	Rotate Left
ROR	Rotate Right
RCL	Rotate Carry left
RCR	Rotate Carry Right
SHLD	Double-precision Shift Left
SHRD	Double-precision Shift Right

SHL Instructions

- Performs a logical shift left on the destination operand



- Syntax for SHL is
  - `SHL dest, count`

```
SHL reg, imm8
SHL mem, imm8
SHL reg, CL
SHL mem, CL
```

Same for all shift and rotate instructions

Fast Multiplication

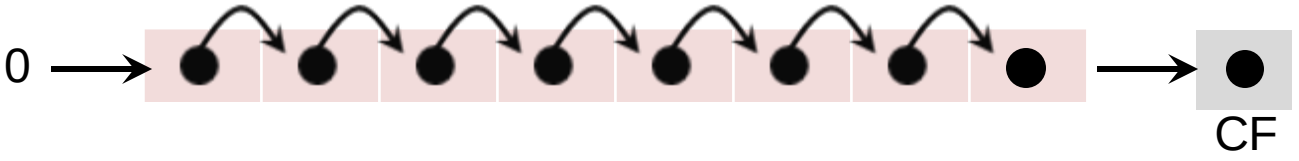
- Shifting left by 1 position generates a number which is 2 times the original operand

Before	0	0	0	0	0	0	1	0
SHL by 1 position	0	0	0	0	0	1	0	0

- Shift left by n position multiplies the operand by 2^n

SHR Instructions

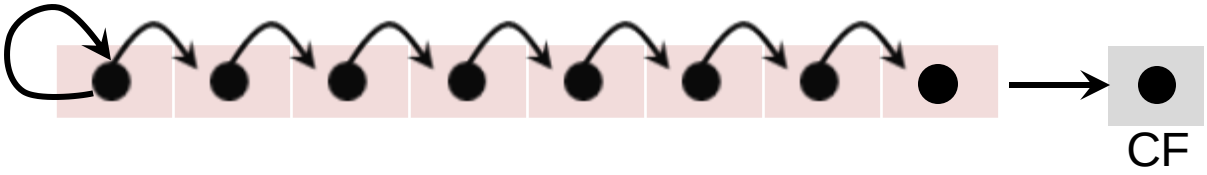
- Performs a logical shift right on the destination operand
- Highest bit position are filled with zero



- Syntax for SHL is
  - `SHR dest, count`

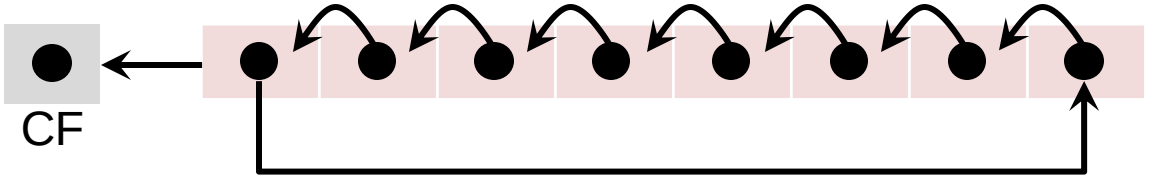
SAL and SAR Instructions

- SAL is identical to SHL
- SAR performs a right arithmetic shift on the destination operand



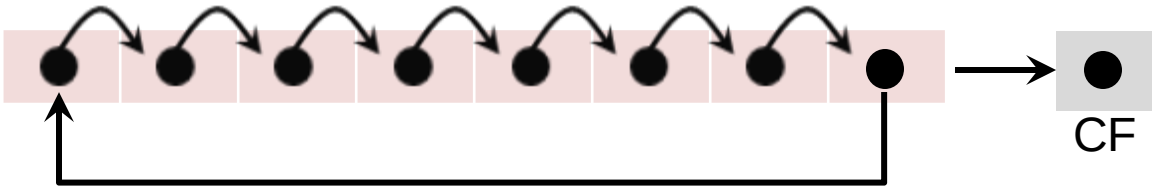
ROL Instructions

- Shifts each bit to the left
- The highest bit is copied into the CF and into the lowest bit
- No bits are lost



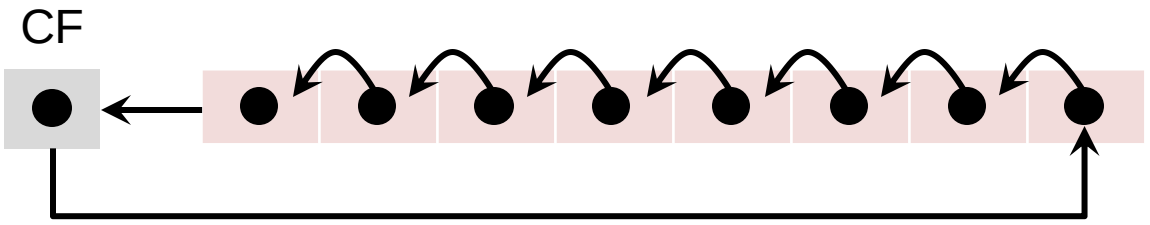
ROR Instructions

- Shifts each bit to the right
- The lowest bit is copied into the CF and into the highest bit
- No bits are lost



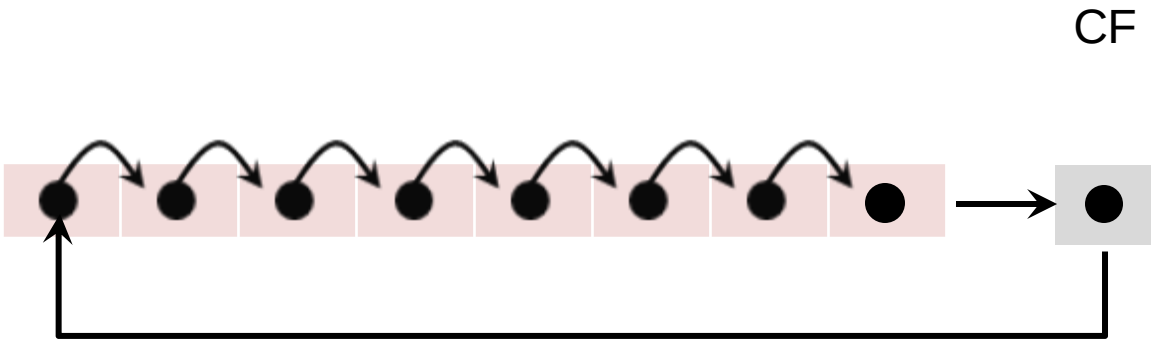
RCL Instruction

- Shifts each bit to the left
- Copies the carry flag to the LSB
- Copies the MSB into carry flag



RCR Instruction

- Shifts each bit to the right
- Copies the carry flag to the MSB
- Copies the LSB to the carry flag



SHLD Instruction

- Shifts destination operand to a given number of bits to left
- Bit positions opened up by the shift are filled by the most significant bits of the source operand
- Source operand is not affected
- Syntax is `SHLD dest, src, count`
- Operand Types

```
SHLD reg16/32, reg16/32, imm8/CL
SHLD mem16/32, reg16/32, imm8/CL
```

SHRD Instruction

- Shifts destination operand to a given number of bits to the right
- Bit positions opened up by the shift are filled by the least significant bits of the source operand
- Source operand is not affected
- Syntax is `SHRD dest, src, count`
- Operand Types

```
SHRD reg16/32, reg16/32, imm8/CL
SHRD mem16/32, reg16/32, imm8/CL
```

Integer Multiplication

- Integer multiplication in x86 can be performed as a 32-bit, 16-bit or 8-bit operation
- The default destination operand in these instructions is the accumulator register (EAX/AX/AL)

MUL Instruction (1/2)

- Used to multiply unsigned numbers
- The multiplier and multiplicand must be the same size and product is twice their size
- `MUL` instruction has a single operand which is multiplier

```
MUL reg/mem8   ;AX = AL * 8-bit reg/mem
MUL reg/mem16  ;DX:AX = AX * 16-bit reg/mem
MUL reg/mem32  ;EDX:EAX = EAX * 32-bit reg/mem
```

MUL Instruction (2/2)

- MUL sets the CF and OF if upper half of product is non-zero
- When AL is multiplied with an 8-bit value, CF and OF are set if AH is non-zero

Multiplicand	Multiplier	Product
AL	8-bit reg/mem	AX
AX	16-bit reg/mem	DX:AX
EAX	32-bit reg/mem	EDX:EAX

IMUL Instruction

- Used to multiply signed numbers
- Preserves sign by extending highest bit of lower half of the product into the upper bits of product
- Three formats of the IMUL instruction
  - One-operand format

```
IMUL reg/mem  ;AX = AL * reg/mem
```

  - Two-operand format

```
IMUL reg, reg/mem  ;AX = reg * reg/mem
```

  - Three-operand format

```
IMUL reg, reg/mem, imm  ;reg = reg/mem * imm
```

DIV Instruction

- Used to divide unsigned integers
- Performs 8-bit, 16-bit and 32-bit integer division
- Takes only one operand which is the divisor

```
DIV reg/mem8

DIV reg/mem16

DIV reg/mem32
```

Dividend	Divisor	Quotient	Remainder
AX	8-bit reg/mem	AL	AH
DX:AX	16-bit reg/mem	AX	DX
EDX:EAX	32-bit reg/mem	EAX	EDX

IDIV Instruction

- Used to divide signed numbers
- Uses same operand types as `DIV` instruction
- Before executing 8-bit division, the dividend (AX) must be completely sign-extended using
  - `CBW` (Convert Byte to Word)
  - `CWD` (Convert Word to Double-word)
  - `CDQ` (Convert Double-word to Quad-word)
- Remainder has the same size as dividend

Sign Extension Instructions

- CBW
  - Extends the sign-bit of AL into AH

```
MOV AL, -8 ;AL=1111 1000
CBW        ;AX=1111 1111 1111 1000
```

- CWD
  - Extends the sign-bit of AX into DX

```
MOV AX, -8 ; AX=1111 1111 1111 1000
CWD        ; AX=1111 1111 1111 1000
           ; DX=1111 1111 1111 1111
```

- CDQ
  - Extends the sign-bit of EAX into EDX

Extended Addition and Subtraction

- Extended precision addition/subtraction helps add/subtract numbers having almost unlimited size
- ADC helps to add two numbers using the carry flag
- SBB helps to subtract two numbers using borrow from carry flag

ADC Instruction

- Adds both a source operand and the value of CF to a destination operand
- Instruction format and limitations are same as that of ADD instruction

```
MOV DL, 0
MOV AL, 0FFh
ADD AL, 0FFh    ;AL=FEh, CF=1
ADC DL, 0       ;DL:AL=01FEh
```

SBB Instruction

- Subtracts both a source operand and value of CF from a destination operand
- Possible operands are same as for the SUB instruction

```
MOV AH, 7
MOV AL,1
SUB AL, 2          ;AL=FFh, CF=0
SBB AH, 0          ;AH:AL=07FFh
```

String Instructions (1/2)

- Five groups of instructions for processing array of **B**ytes, **W**ords and **D**ouble-words
- Called String Primitives but not limited to character arrays only
- These instructions use ESI/SI and EDI/DI registers to address memory
- Array indexes are repeated and incremented automatically

String Instructions (2/2)

Instruction	Description
MOVSB, MOVSW, MOVSD	Copy data from memory addressed by SI to memory addressed by DI
CMPSB, CMPSW, CMPSD	Compare the contents of memory addressed by SI to memory addressed by DI
SCASB, SCASW, SCASD	Compare the accumulator register (AL, AX or EAX) to the contents of memory addressed by DI
STOSB, STOSW, STOSD	Store the contents of accumulator register into memory location addressed by DI
LODSB, LODSW, LODSD	Load the contents of memory addressed by SI into the accumulator register

Direction Flag

- String instructions increment/decrement SI and DI based on the state of Direction Flag
- DF can be explicitly modified using instructions
  - `CLD` ;clear Direction Flag (forward direction)
  - `STD` ;set Direction Flag (reverse direction)

```
if(DF=0) then
    SI=SI+1
    DI=DI+1
else
    SI=SI-1
    DI=DI-1
end if
```

Value of DF	Effect on SI and DI	Address Sequence
Clear	Increment	Low-High
Set	Decrement	High-Low

Repeat Prefix

- REP (a repeat prefix) can be used just before MOVSB, MOVSW, MOVSD
- By default CX controls the number of repetitions

Prefix	Description
REP	Repeat while CX>0
REPZ, REPE	Repeat while ZF=1 and CX>0
REPNZ, REPNE	Repeat while ZF=0 and CX>0

REP Prefix

- Repeat while CX>0
- Value of CX is checked before execution of instruction
- If CX is zero, string instruction is not executed

```
while(CX≠0)
    execute the string instruction
    CX = CX – 1
end while
```

REPE, REPZ Prefixes

```
while(CX≠0)
    execute the string instruction
    CX = CX – 1
    if (ZF=0) then
        exit loop
    end if
end while
```

REPNE, REPNZ Prefixes

```
while(CX≠0)
    execute the string instruction
    CX = CX – 1
    if (ZF=1) then
        exit loop
    end if
end while
```

MOVSB, MOVSW, MOVSD (1/2)

- Copy data from memory location pointed to by SI to memory location pointed to by DI
- SI and DI are either incremented or decremented based on the value of DF
- SI/DI incremented/decremented by
  - 1 when used with MOVSB
  - 2 when used with MOVSW
  - 4 when used with MOVSD

MOVSB, MOVSW, MOVSD (2/2)

```
.data
    src DB “Hello World!”
    src_len DB $-src
    dst DB src_len DUP(?)
.code
    MOV CX, src_len
    MOV SI, OFFSET src
    MOV DI, OFFSET dst
    REP MOVSB
```

CMPSB, CMPSW, CMPSD (1/2)

- Compare memory operand pointed to by SI to memory operand pointed to by DI
- SI and DI are either incremented or decremented based on the value of DF
- SI/DI incremented/decremented by
  - 1 when used with CMPSB
  - 2 when used with CMPSW
  - 4 when used with CMPSD

CMPSB, CMPSW, CMPSD (2/2)

```
.data
    src DB “Hello World!”
    src_len DB $-src
    dst DB “Hello! World”
.code
    MOV CX, src_len
    MOV SI, OFFSET src
    MOV DI, OFFSET dst
    REPE CMPSB
```

SCASB, SCASW, SCASD (1/2)

- Compare the value in accumulator to the memory value pointed by DI
- Useful when looking for a single value in string or array
- When combined with REPNE prefix, SCAS***X*** scans until either accumulator is matched a value in memory or CX=0
- When combined with REPE prefix, string is scanned while CX>0 and value in accumulator matches each subsequent value in string

SCASB, SCASW, SCASD (2/2)

```
.data
    dst DB “Hello! World”
.code
    MOV CX, src_len
    MOV DI, OFFSET dst
    MOV AL, ‘o’
    REPNE SCASB
```

STOSB, STOSW, STOSD (1/2)

- Store the contents of accumulator in the memory addressed by DI
- When used with REP prefix, these instructions can be used to fill all elements of string with the same value

STOSB, STOSW, STOSD (2/2)

```
.data
    dst DB 5 DUP(?)
    dst_len DW $-dst
.code
    MOV CX, dst_len
    MOV DI, OFFSET dst
    MOV AL, 'H'
    REP STOSB
```

LODSB, LODSW, LODSD

- Load a byte/word from memory at SI into accumulator register
- Used to load a single value because older value is always overwritten in accumulator