Unit-4

Arithmetic, Logic, String and Bitwise Operations

Arithmetic operations

- **add src, dest**

  addb $10, %al ---adds the immediate value 10 to the 8-bit AL register

  addw %bx, %cx --- adds the 16-bit value of the BX register to the CX register

  addl data, %eax --- adds the 32-bit integer value at the data label to EAX

- **adc src, dest ---Add with carry**

sub src,dest

- source value is subtracted from the destination value, with the result stored in the destination operand location.
- The source and destination operands can be 8-, 16-, or 32-bit registers or values stored in memory.

sbb src,dest --subtract with borrow

inc dest

dec dest

- The INC and DEC instructions are used to increment (INC) and decrement (DEC) an unsigned integer where destination can be an 8, 16, or 32-bit register or value in memory.

- **neg dest ----change the sign of dest –**
  - forms the 2's complement of the specified destination
  - As destination is modified after execution, it can't be immediate value.

- **cmp src(operand1) , dest(operand2)**

  - The CMP instruction compares the second operand with the first operand.
  - It performs a subtraction operation on the two operands behind it as (operand2 – operand1)
  - Neither of the operands is modified, but the EFLAGS register is modified.

Multiplication and division of integers:

- **Unsigned multiplication:**

  mul src

Source Operand Size	Destination Operand	Destination Location
8 bits	AL	AX
16 bits	AX	DX:AX
32 bits	EAX	EDX:EAX

- **Signed multiplication:**

  imul src

  imul %dx ---------- (dx x ax)---result is in DX:AX

  imul src, destreg

  imul %dx, %ax ---- result will be in DX:AX

  imul imm, src, destreg

  imul $10, %ebx, %eax

  - where imm (multiplier) is an immediate value, source is a 16 or 32-bit register or value in memory, and destination must be a general-purpose register.

Multiplication and division of integers:

- **Unsigned division:**

  div src

Dividend	(Divisor or src)	Quotient	Remainder
AX (16 bit)	8 bits	AH	AL
DX:AX (32 bit)	16 bits	DX	AX
EDX:EAX (64 bit)	32 bits	EDX	EAX

- **Signed division:**
  idiv src

BCD numbers:

- DAA (decimal adjust after addition)—packed BCD addition
- DAS (decimal adjust after subtraction)—packed BCD subtraction

- **Unpacked BCD arithmetic:**

  aaa --ASCII adjust after addition

  aas --ASCII adjust after subtraction

  aam --ASCII adjust after multiplication

  aad --ASCII adjust before division

Logic operartion instructions:

- Logic instructions:

  and src , dest

  or src , dest

  xor src , dest

  not dest

- Source can be an 8, 16, or 32-bit immediate value, register, or value in memory
- Destination can be an 8, 16, or 32-bit register or value in memory
- cannot use memory values for both the source and destination

Shift and rotate instructions :

- Shift operations for signed numbers are also known as arithmetic shifts
- Shift operations for unsigned numbers are known as logical shifts

- **sal –Shift arithmetic left**

  sal destination --- shifts the destination value left one position

  sal %cl, destination -- shifts the destination value left by the number of times specified in the CL register

  sal shifter, destination--- shifts the destination value left the number of times indicated by the shifter value

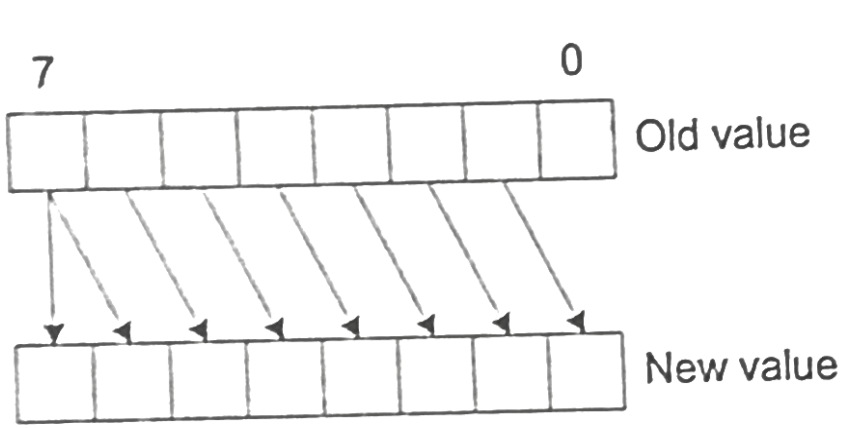
- The destination operand can be an 8, 16, or 32-bit register or value in memory.

- **shl –Shift logical left**

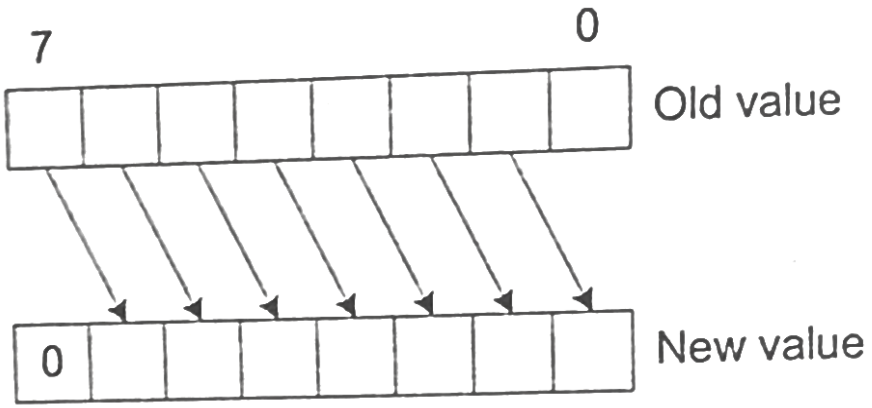
- **Left shift operations for both signed and unsinged numbers are identical.**

- SAR- Shift arithmetic right
  - The SAR instruction either clears or sets the bits emptied by the shift, depending on the sign bit (MSB/Old MSB) of the integer.

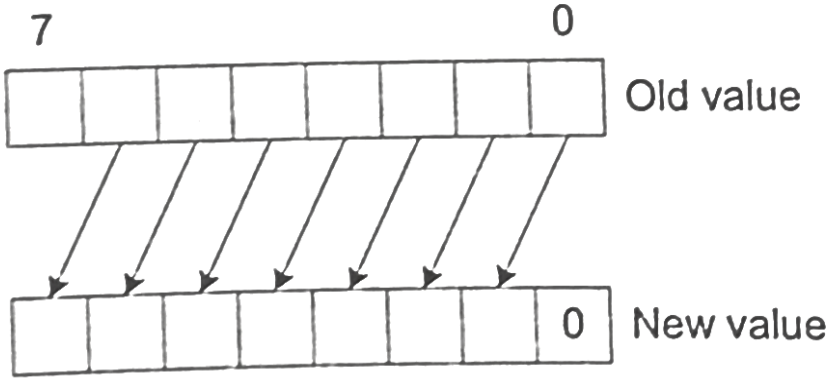
- SHR- **Shift logical Right**
  - The SHR instruction clears the bits emptied by the shift.



(a) Arithmetic right shift



(b) Logical right shift



(c) Arithmetic or logical left shift

Figure 5.2: Various shift instructions.

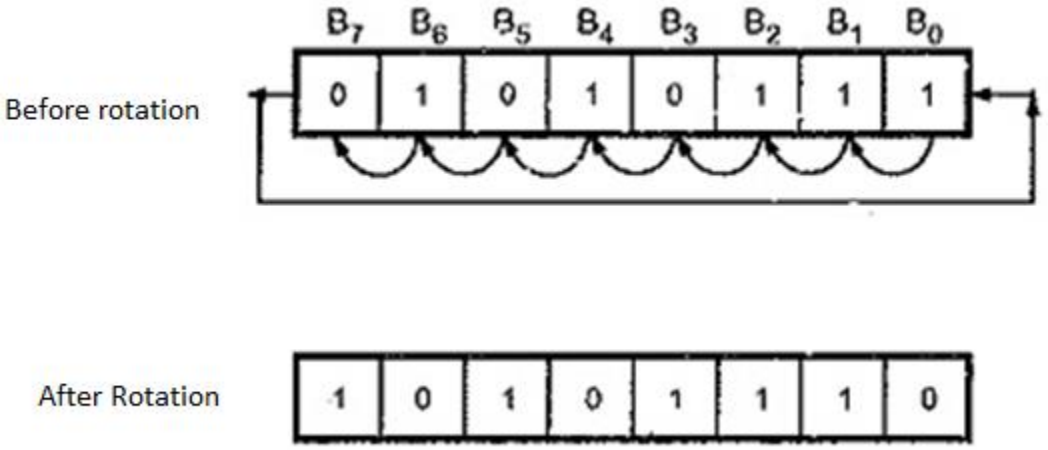
Rotate instructions:

- The rotate instructions perform just like the shift instructions, except the overflow bits are pushed back into the other end of the value instead of being dropped.
  - **ROL destination** --- rotates the destination value left one position
  - **ROL %cl, destination** -- rotates the destination value left by the number of times specified in the CL register
  - The destination operand can be any register or memory location of size 8,16 or 32 bit

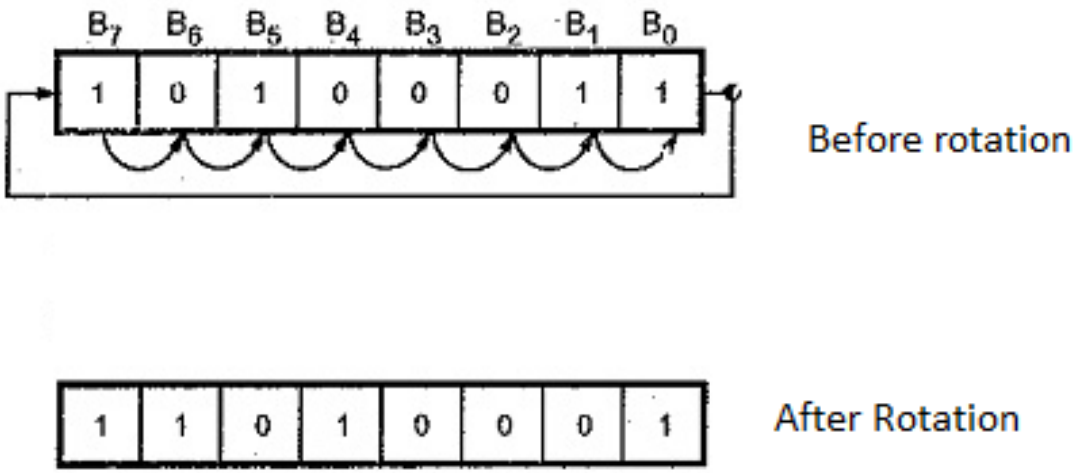
e.g. ror dest

- A single operand that is shifted once in the indicated direction
- Two operands: ROR %cl, %ax
  - The %cl register to indicate the number of times to rotate the destination operand
- Two operands: ROR $0x02,%cx
  - An immediate value to indicate the number of times to rotate the destination operand

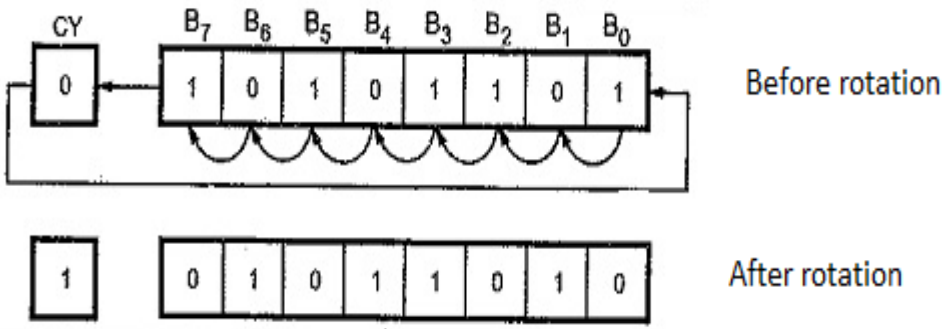
- rol count, dest



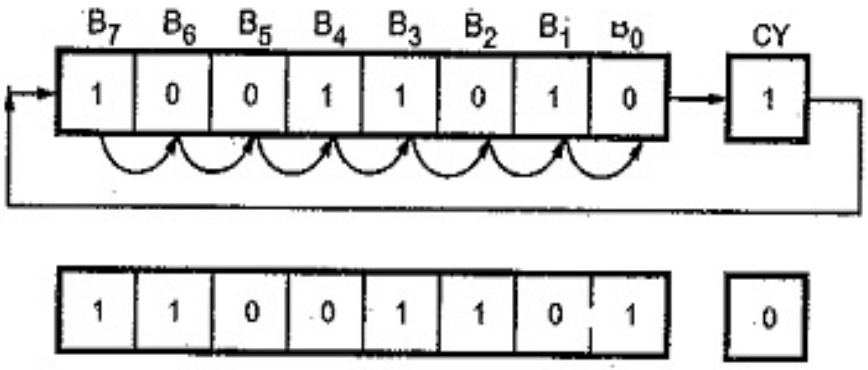
- ror count, dest



- rcl count, dest
  - Rotate left with carry



- rcr count, dest
  - Rotate right with carry



String Attributes:

- **Address:**
  - Address of string provides the address in memory of the first or last element of the string.

- **Size of each elements**
  - Size of an element to be 8, 16 or 32 bits.
- **Number of elements**
  - Can have any possible valued.
- **Direction for address adjustment**
  - It can be upward or downwards (i.e. address is incremented or decremented)
  - It is decided by the direction flag

- If **DF=0** string is processed from lower address to higher address i.e address is auto incremented
- If **DF=1** string is processed from Higher address to lower address i.e address is auto decremented

String instructions

- ***The MOVS instruction:***
- To move string data from one memory location to another.
- It copies a byte, a word or a long word from a source string to the destination string.

MOVSB: Moves a single byte

MOVSW: Moves a word (2 bytes)

MOVSL: Moves a doubleword (4 bytes)

- How to load the ESI and EDI values?

movl $str1, %esi

  - This instruction moves the 32-bit memory location of the label str1 to the ESI register.

movl $output, %edi

  - This instruction moves the 32-bit memory location of the output label to the EDI register.

leal output, %edi **----**loads the effective address of an object

  - loads the 32-bit memory location of the output label to the EDI register.

String instructions

#example of the MOVS instructions

```
.section .data
str11:
.ascii “Welcome to DKTE.\n”
.section .bss
.lcomm output, 17
.section .text
.globl _start
        _start:
        nop
```

String instructions

leal str1, %esi

leal output, %edi

movsb

movsw

movsl

movl $1, %eax

movl $0, %ebx

int $0x80

```
(gdb) s
13          movsb
(gdb) s
14          movsw
(gdb) x/s &output
0x80490b0 <output>:      "W"
(gdb) s
15          movsl
(gdb) x/s &output
0x80490b0 <output>:      " Wel "
(gdb) s
17          movl $1, %eax
(gdb) x/s &output
0x80490b0 <output>:      "Welcome"
(gdb)
```

- **CLD :**clear the DF flag
- If DF=0
  - the ESI and EDI registers are incremented after each MOVS instruction

- **STD** :set the DF flag
- If DF=1
  - ESI and EDI registers are decremented after each MOVS instruction

String instructions

- **Comparing Strings:** is used to compare string values.
  - Compares an element of source string with an element of destination string and sets the flags accordingly.

  - CMPSB: Compares a byte value
  - CMPSW: Compares a word (2 bytes) value
  - CMPSL: Compares a double word (4 bytes) value

String instructions

- **Scanning Strings:**
  - to scan a string for a specific character or character sequence
  - is used to scan strings for one or more search characters.
  - The EDI register must contain the memory address of the string to scan.

  - **SCASB: Compares a byte in memory with the AL register value**
  - **SCASW: Compares a word in memory with the AX register value**
  - **SCASL: Compares a doubleword in memory with the**

String instructions

- **LODS:**
  - The LODS instruction is used to move a string value in memory into the EAX register
  - The ESI register must contain the memory address of the location of the string to load.

- **LODSB** :
  - instruction moves (loads) a byte into the AL register,
- **LODSW:**
  - instruction moves(loads) a word into the AX register,
- **LODSL:**
  - instruction moves(loads) a doubleword into the EAX register.

String instructions

- **STOS:**
  - Moves string back into memory
  - The STOS instruction uses an destination operand of the EDI register.

  - **STOSB** --store a byte
  - **STOSW** ----store a word
  - **STOSL ---**store a double word

```
#    Example of using the STOS and LODS instruction
.section .data
        str1:
        .ascii  “A“
        .section .bss
                .lcomm block1, 50
.section .text
        .globl _start
        _start:
                nop
                leal str1, %esi
                leal block1, %edi
                movl $15, %ecx
                cld
                lodsb
                rep stosb
                movl $1, %eax
                movl $0, %ebx
                int $0x80
```

String instructions

- ***The REP prefix:***
  - to repeat a string instruction a specific number of times, by the value in the ECX register
  - similar to using a loop, but without the extra LOOP instruction.
  - The **REP** instruction repeats the string instruction immediately following it until the value in the ECX register is zero.

String instructions

#Moving a string byte by byte , example of the REP instruction

```
.section .data
value1:
.ascii “Welcome to DKTE.\n”
.section .bss
.lcomm output, 17
.section .text
.globl _start
_start: nop
        leal value1, %esi
        leal output, %edi
```

String instructions

```
movl $23, %ecx
cld
rep movsb
movl $1, %eax
movl $0, %ebx
int $0x80
```

```
x/s &output
```

String instructions

- Prefix instructions:

String instructions	Can be used with				
	rep	repe	repz	repne	repnz
movs	√				
cmps		√	√	√	√
scas		√	√	√	√
lods	√				
Stos	√				

Bit oriented instructions

- Test and modify individual bits in the operands
- Operates on individual bits and set the conditional flag to the bit values.

- **Bit testing and modification:**

bt bitoffset, dest

- Copies the specified bit from the dest operand to the carry flag.(CF)

```
movl $0xff,%eax
movl $0x02,%ecx
bt %ecx, %eax
```

eax =1111 1111

--bt instruction copies the index 2 value specified by ecx register to the CY i.e 1 (shown by RED color)

btr bitoffset, dest

– Copies the specified bit from the dest operand to the carry flag.(CF) and at the same time **resets that bit to 0**

```
movl $0xff,%eax
movl $0x02,%ecx
btr %ecx, %eax
```

after btr %ecx, %eax

eax =1111 1011

--btr instruction copies the index 2 value specified by ecx register to the CY i.e 1 and at the same time resets that bit to 0 (shown by RED color)

bts bitoffset, dest

– Copies the specified bit from the dest operand to the carry flag.(CF) and at the same time **sets that bit to 1**

movl $0xfb,%eax

movl $0x02,%ecx

bts %ecx, %eax

after bts %ecx, %eax

eax =1111 1111

--btr instruction copies the index 2 value specified by ecx register to the CY i.e 0 and at the same time sets that bit to 1 (shown by RED color)

btc bitoffset, dest

– Copies the specified bit from the dest operand to the carry flag.(CF) and at the same time **compliments that bit.**

```
movl $0xff,%eax
movl $0x02,%ecx
btr %ecx, %eax
```

after btc %ecx, %eax

eax =1111 1011

--btc instruction copies the index 2 value specified by ecx register to the CY i.e 1 and at the same time it compliments that bit (shown by RED color)

Searching for a set bit:

bsf src, dest

bsr src ,dest

- src can be 16 bit or 32 bit register or memory location and
- dest can be 16 bit or 32 bit register

- **bsf src, dest ------bit scan forward**
  - **This instruction is used to return index of least significant bit that is set to 1 in the operand.**

    movl $0x18,%eax

    bsf %eax, %ecx

Here eax = 0001 1000 ---- here index of least significant bit that is set to 1 is 3 and is returned to ecx

- **bsr src ,dest --bit scan reverse**
  - **Is used to return index of the most significant bit that is set to 1 in the src operand.**

```
movl $0x2C,%eax
bsr %eax, %ecx
```

Here eax = 0010 1100 ---- here index of **most significant bit** that is set to 1 is 5 and is returned to ecx

Thank You