**Seminar 3 – Bitwise instructions**

## Boolean instructions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operand1 – op1** | **Operand2 – op2** | **And op1, op2** | **Or op1, op2** | **Xor op1, op2** | **Not op1** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Op1: reg, mem

Op2: reg, mem, imm

Op2 = op1 relatie op2

The operands can be 8, 16, or 32 bits, and they must be the same size.

* General rules:

X can have value 1 or value 0

**X or 0 = X**

**X or 1 = 1**

**X and 0 = 0**

**X and 1 = X**

**Examples:**

Data segment: a db 0000.1111b

Code segment:

and byte[a], 0000.0101b ; => 0000.1111b and 0000.0101b => a=0000.0101b

or byte[a], 0000.1101b ; => 0000.0101b or 0000.1101b =>a=0000.1101b

xor byte[a], 0000.0110b ; => 0000.1101b xor 0000.0110b =>a=0000.1011b

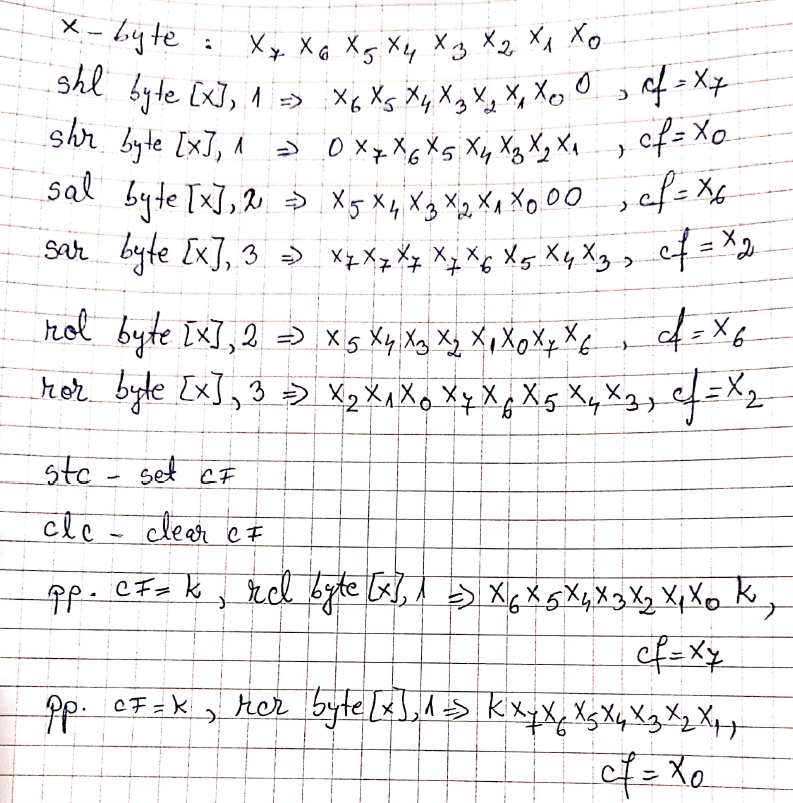
not byte[a] ; => not 0000.1011b =>a=1111.0100b

## Shift instructions

|  |  |  |
| --- | --- | --- |
| SHL *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The SHL **(shift logic left)** instruction performs a logical left shift on the destination operand, filling the lowest bit with 0. The highest bit is moved to the Carry flag, and the bit that was in the Carry flag is discarded. | mov dl, 0000.1111b  shl dl, 3 ; dl = 0111.1000b |
| SHR *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The SHR **(shift logic right)** instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0.  The lowest bit is copied into the Carry flag, and the bit that was previously in the Carry flag is lost. | mov eax, 12340000h  shr eax, 16 ; eax = 00001234h |
| SAL *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The SAL **(shift arithmetic left)** instruction works the same as the SHL instruction. For each shift count, SAL shifts each bit in the destination operand to the next highest bit position. The lowest bit is assigned 0.  The highest bit is moved to the Carry flag, and the bit that was in the Carry flag is discarded. | mov al, 1110.0011b  sal al, 1 ; al=1100.0110b |
| SAR *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The SAR **(shift arithmetic right)** instruction performs a logical right shift on the destination operand, filling all the positions with the most significant bit (bit sign).  The lowest bit is copied into the Carry flag, and the bit that was previously in the Carry flag is lost. | mov al, 1110.0011b  sar al, 1 ; al=1111.0001b |

## Rotate instructions

|  |  |  |
| --- | --- | --- |
| ROL *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The ROL **(rotate left)** instruction shifts each bit to the left.  The highest bit is copied into the Carry flag and the lowest bit position. | mov bl, 0111.1000b  rol bl, 4 ; bl = 1000.0111b |
| ROR *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The ROR **(rotate right)** instruction shifts each bit to the right and copies the lowest bit into the Carry flag and the highest bit position. | mov al, 1010.1110b  ror al, 3 ; al=1100.0101 |
| RCL *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The RCL **(rotate carry left)** instruction shifts each bit to the left, copies the Carry flag to the LSB, and copies the MSB into the Carry flag. | mov al, 0  stc  rcl al, 1 ; al=0000.0001b, cf = 0  rcl al, 2 ; al=0000.0100b |
| RCR *destination, count*  *destination: reg, mem*  *count: imm8, or CL reg* | The RCR **(rotate carry right)** instruction shifts each bit to the right, copies the Carry flag into the MSB, and copies the LSB into the Carry flag. | mov bl, 0  stc  rcr bl, 1 ; bl=1000.0000b, cf = 0  rcr bl, 2 ;bl=0010.0000b |



## Prob1:

Byte A – given: a7 a6 a5 a4 a3 a2 a1 a0

Compute byte B:

Bitis 0-2 B = not bits 0-2 A

Bits 3-4 B = 1

Bits 5-7 B = bits 2-4 A

* b7 b6 b5 b4 b3 b2 b1 b0 = a4 a3 a2 1 1 not a2 not a1 not a0

Eg, if A = 1111.1111b = FFh => B = 1111.1000 = F8h

## Data segment a db 1111.1111b b db 0

Code segment

Mov al, [a] ; al= a7 a6 a5 a4 a3 a2 a1 a0

Not al ; al =not (a7 a6 a5 a4 a3 a2 a1 a0)

And al, 0000.0111b ; al = 00000 not a2 not a1 not a0

Or byte[b], al ; byteb = 00000 not a2 not a1 not a0

Or byte[b], 00011000b ; byteb = 00011 not a2 not a1 not a0

Mov bl, [a] ; bl = a7 a6 a5 **a4 a3 a2** a1 a0

And bl, 00011100b ; bl = 000 a4 a3 a2 00

Shl bl, 3 ; bl=a4 a3 a2 00000

Or byte[b], bl ; b=a4 a3 a2 11 not a2 not a1 not a0

## Prob2:

**Generate value 13 (decimal) in bl (initial bl=0) with shift and rotate instructions.**

**Bl = 0000.0000 => bl = 0000.1101b**

Stc ; cf=1

Rcl bl, 1 ; bl = 0000.0001b, cf = 0

Stc ; cf =1

Rcl bl, 1 ; bl = 0000.0011b, cf =0

Rcl bl,1 ; bl = 0000.0110, cf =0

Stc

Rcl bl, 1 ; bl = 0000.1101b

**Pb 3. Se se insumeze bitii de pe pozitia 3 5 din a cu bitii de pe pozitia 2 4 din b, a si b de tip byte., rez = biti a+ biti b**

a7 a6 a5 a4 a3 a2 a1 a0

b7 b6 b5 b4 b3 b2 b1 b0

mov al, [a]

mov bl, [b]

shr al, 3 ; al = 000a7a6a5a4a3

shr bl, 2 ; bl = 00b7b6b5b4b3b2

and al, 00000111b ; al = 00000a5a4a3

and bl, 00000111b ; bl = 00000b4b3b2

add al, bl

**Secvente de instructiuni:**

**Mov al, 10**

**Jmp Scadere ; salt la eticheta Scadere**

**Adunare:**

**Add al, 5**

**Jmp final**

**Scadere:**

**Sub al, 6**

**Jmp Adunare**

**Final:**

**Scadere, adunare – etichete de cod**

## CMP and Jcond

The CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand. Neither operand is modified:

**Syntax: CMP *destination, source*** *; destination-source is performed*

*destination: reg, mem*

*source: reg, mem, imm*

*destination and source – same dimensions*

The CMP instruction changes the Overflow, Sign, Zero, Carry, Auxiliary Carry, and Parity flags according to the value the destination operand would have had if actual subtraction had taken place.

Usually a CMP instruction is used follows by a conditional jump.

The condition is set by the relation between *destination* and *source*.

***Cmp destionation, source***

***Jcc – jump if destionation is in relations (set by jcc) with source.***

These jumps checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand.

* A condition code (cc) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, the jump is not performed and execution continues with the instruction following the Jcc instruction.

**Jcc — Jump if Condition Is Met**

**The conditions for each Jcc mnemonic are given in the "{description}" column of the table on the preceding page. The terms "less" and "greater" are used for comparisons of signed integers and the terms "above" and "below" are used for unsigned integers.**

|  |  |
| --- | --- |
| **Mnemonic** | **Description** |
| JA name\_for\_label | Jump if above (CF=0 and ZF=0). |
| JAE name\_for\_label | Jump if above or equal (CF=0). |
| JB name\_for\_label | Jump if below (CF=1). |
| JBE name\_for\_label | Jump if below or equal (CF=1 or ZF=1). |
| JC name\_for\_label | Jump if carry (CF=1). |
| JCXZ name\_for\_label | Jump if CX register is 0. |
| JECXZ name\_for\_label | Jump if ECX register is 0. |
| JE name\_for\_label | Jump if equal (ZF=1). |
| JG name\_for\_label | Jump if greater (ZF=0 and SF=OF). |
| JGE name\_for\_label | Jump if greater or equal (SF=OF). |
| JL name\_for\_label | Jump if less (SF<>OF). |
| JLE name\_for\_label | Jump if less or equal (ZF=1 or SF<>OF). |
| JNA name\_for\_label | Jump if not above (CF=1 or ZF=1). |
| JNAE name\_for\_label | Jump if not above or equal (CF=1). |
| JNB name\_for\_label | Jump if not below (CF=0). |
| JNBE name\_for\_label | Jump if not below or equal (CF=0 and ZF=0). |
| JNC name\_for\_label | Jump if not carry (CF=0). |
| JNE name\_for\_label | Jump if not equal (ZF=0). |
| JNG name\_for\_label | Jump if not greater (ZF=1 or SF<>OF). |
| JNGE name\_for\_label | Jump if not greater or equal (SF<>OF). |
| JNL name\_for\_label | Jump if not less (SF=OF). |
| JNLE name\_for\_label | Jump if not less or equal (ZF=0 and SF=OF). |
| JNO name\_for\_label | Jump if not overflow (OF=0). |
| JNP name\_for\_label | Jump if not parity (PF=0). |
| JNS name\_for\_label | Jump if not sign (SF=0). |
| JNZ name\_for\_label | Jump if not zero (ZF=0). |
| JO name\_for\_label | Jump if overflow (OF=1). |
| JP name\_for\_label | Jump if parity (PF=1). |
| JPE name\_for\_label | Jump if parity even (PF=1). |
| JPO name\_for\_label | Jump if parity odd (PF=0). |
| JS name\_for\_label | Jump if sign (SF=1). |
| JZ name\_for\_label | Jump if zero (ZF = 1). |

a) Mov al, 10 ; al, -10

Mov bl, 20 bl, -20

If **al** este mai mare decat **bl** atunci rez = al-bl

Altfel rez = bl-al

Cmp al, bl

JAE then\_etic ; JGE

JB then else\_etic ;JL

Then\_etic:

Sub al, bl

Mov [rez], al

Jmp final

Else\_etic:

Sub bl, al

Mov [rez], bl

Final:

**b) if a mod 2 = 0 then rez = a\*3, a –byte**

**c) if c>d then c=c-1**

**else c=c+2**

**c, d sunt numere intregi**

**c - > al**

**d-> bl**

**mov al, [c]**

**mov bl, [d]**

**cmp al, bl**

**JG then\_eticheta**

**JNG else\_eticheta ; jmp else\_eticheta**

**then\_eticheta:**

**sub al, 1**

**jmp final**

**else\_eticheta:**

**add al, 2**

**final:**

**; verificare daca la adunare avem depasire**

**a+b, a si b sunt quad**

**mov eax, dword[a+0]**

**mov edx, dword[a+4]**

**mov ecx, dowrd[b+4]**

**mov ebx, dword[b+0]**

**; edx:eax+**

**;ecx:ebx**

**Add eax, ebx**

**; verific daca CF=1**

**JC sumalastangacucarry ; jump if caary is set**

**JNC sumastangafaracarry ; jump if caary is not set**

**sumalastangacucarry:**

**adc edx, ecx**

**sumastangafaracarry**

**add edx, ecx**

|  |  |
| --- | --- |
| **b) if a mod 2 = 0 then rez = a\*3**  unsigned  mov al, [a]  mov ah, 0  mov bl, 2  div bl ;ax/bl = al cat si ah rest  cmp ah, 0  JE ramura\_then  Jmp final ; JNE final  ramura\_then:  mov al,[a]  mov bl, 3  mul bl ; ax=a\*3  mov [rez], ax  final: | **Signed**  mov al, [a]  cbw  mov bl, 2  idiv bl ;ax/bl = al cat si ah rest  cmp ah, 0  JE ramura\_then  Jmp final ; JNE final  ramura\_then:  mov al,[a]  mov bl, 3  imul bl ; ax=a\*3  mov [rez], ax  final: |

## Probl 3. (unsigned and signed)

E = a\*2

F = c/2

a – byte, b – word, c – word

if e<f, then rez = f/2

else rez = e\*2

Produce division and multiply operations without using mul/imul and div/idiv

|  |  |
| --- | --- |
| unsigned  Mov al, [a]  Shl al, 1 ; al = a\*2  Cbw ; ax=a\*2  Mov bx, [c]  Shr bx, 1 ; bx =c/2  Cmp ax, bx  JB then  JAE else ; jmp else  Then:  Shr bx, 1 ; bx=f/2  Jmp final  Else:  Shl ax, 1 = ax =e\*2  Final | signed  Mov al, [a]  Shl al, 1 ; al = a\*2  Cbw ; ax=a\*2  Mov bx, [c]  Sar bx, 1 ; bx =c/2  Cmp ax, bx  JL then  JGE else ; jmp else  Then:  Sar bx, 1 ; bx=f/2  Jmp final  Else:  Shl ax, 1 = ax =e\*2  Final |

## Multiplications and divisions using bitwise instructions

