Instruction Set:

An instruction length may range between 1 and 6 bytes. The instruction will contain the codes for the operation to be performed, the addressing modes, the registers and immediate data as required. It is categorized as

1) Data transfer Group
2) Arithmetic Group
3) Logical Group
4) Control transfer Group
5) Miscellaneous Instruction Group.

Data transfer Group:

It performs data movement between registers, registers and memory, register and immediate data, memory and immediate data, between two memory locations, between I/O port & registers, & between stack & memory / register. Both 8-bit & 16-bit data transfers are provided.

Syntax: **MOV dest, src.** Move byte or word

copies the byte or word from source operand to destination.

Byte Register (8 bit) — Word Register (16 bit)

i) MOV RBD, RBS — (RBD) ← (RBS)

ii) MOV RWD, RWS — (RWD) ← (RWS)

iii) MOV RB, DADDR — (RB) ← (EA)

Effective Address → EA

iv) MOV RB, DATA8 — (RB) ← (DATA8)

v) MOV SR, RW — (SR) ← (RW)

vi) MOV DADDR, SR — (EA) ← (SR)

Syntax: **POP dest** pop word off stack

Increment SP by 2 to point to new stack top.

PUSH src | Push word onto stack

Decrement SP by 2 and transfers one word from source to stack top.

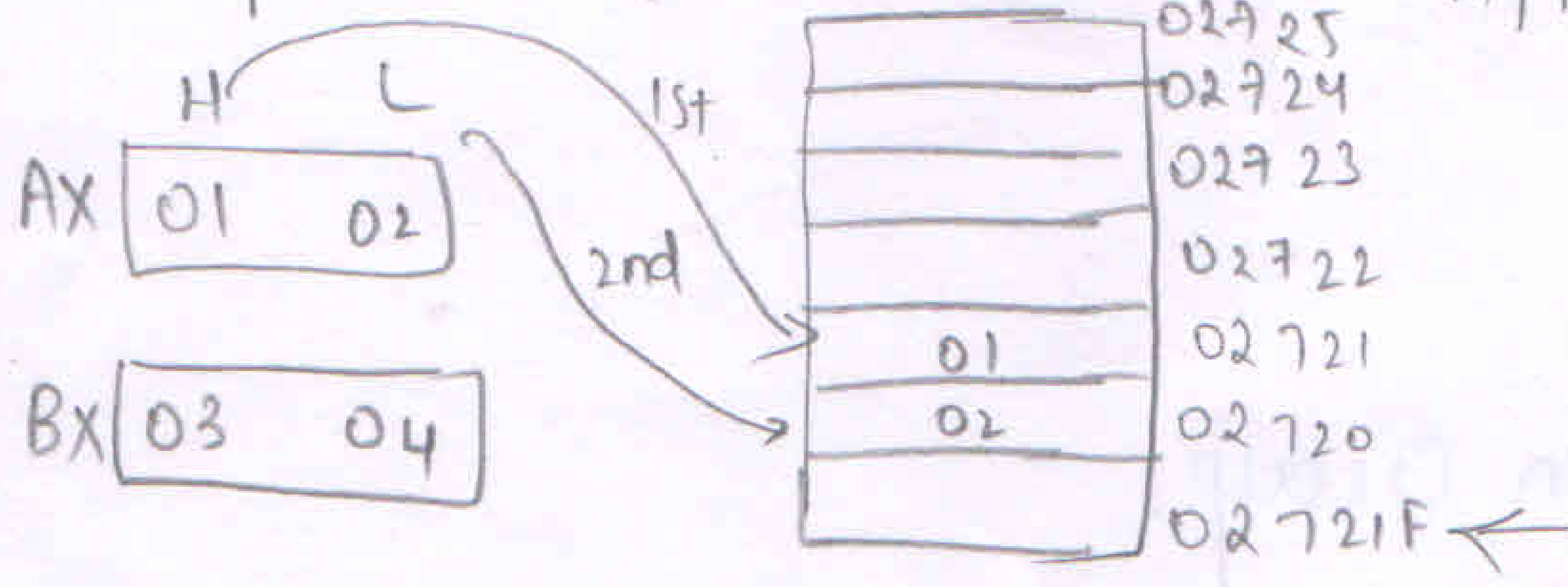
i) PUSH RW

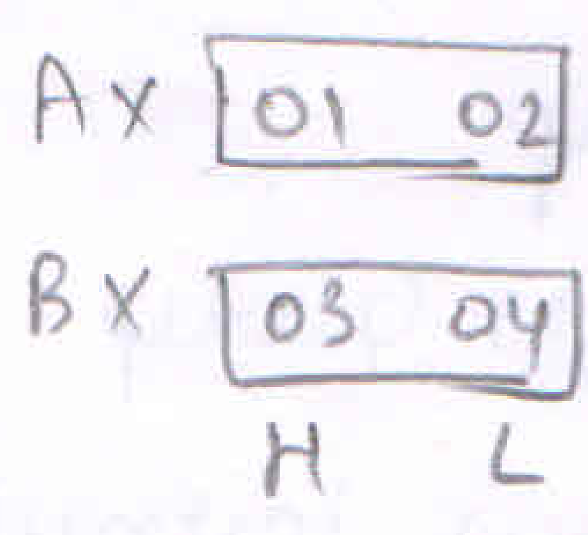
$(SP) \leftarrow (SP) - 2$, $((SP)) \leftarrow (RW)$

ii) PUSH DADDR

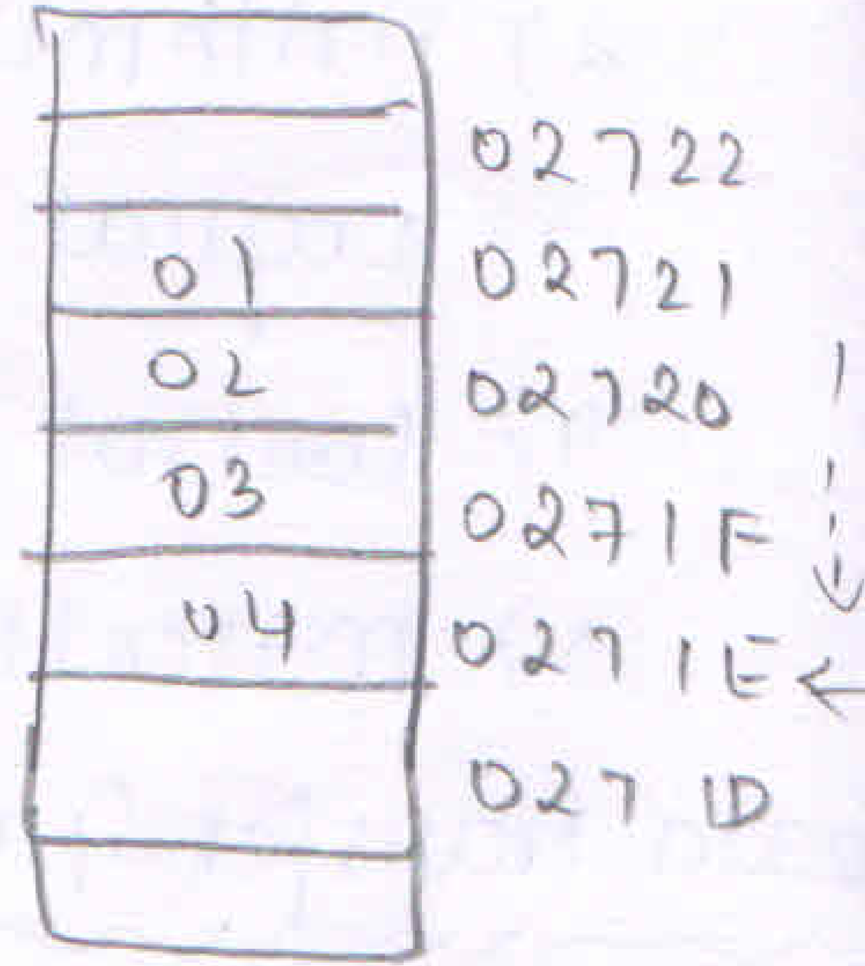
$(SP) \leftarrow (SP) - 2$ $((SP)) \leftarrow (EA)$

Before PUSH

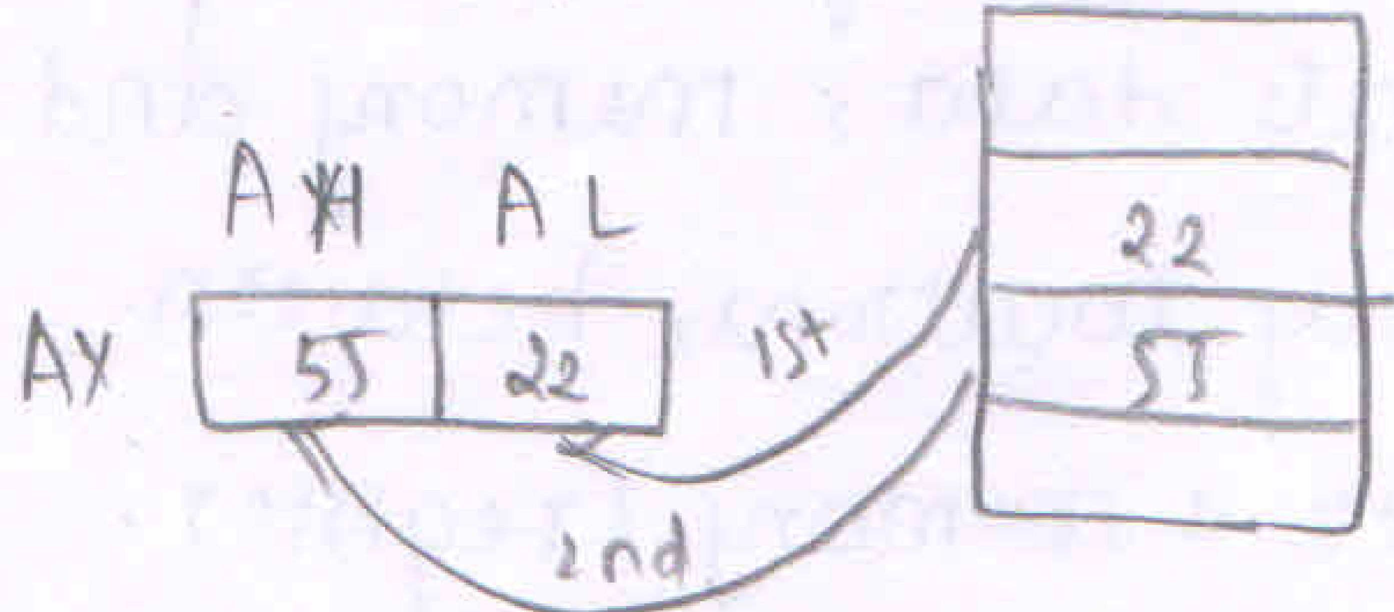


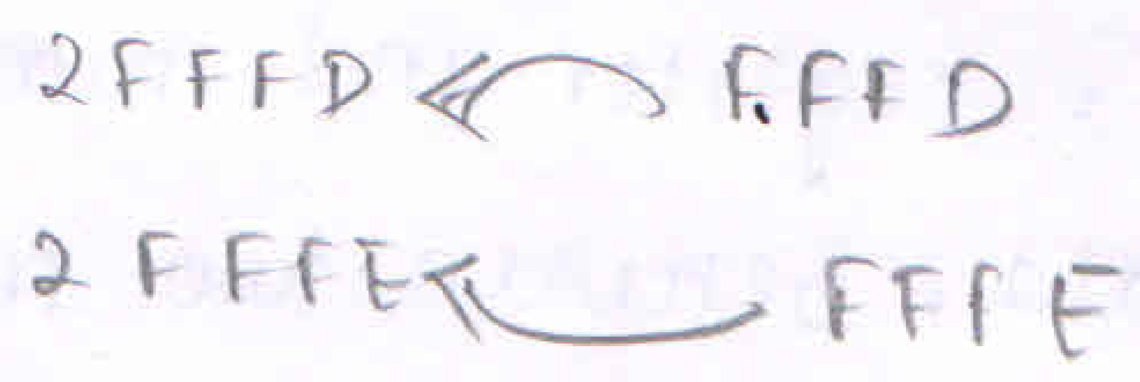


After PUSH



Eg. POP AX
POP BX





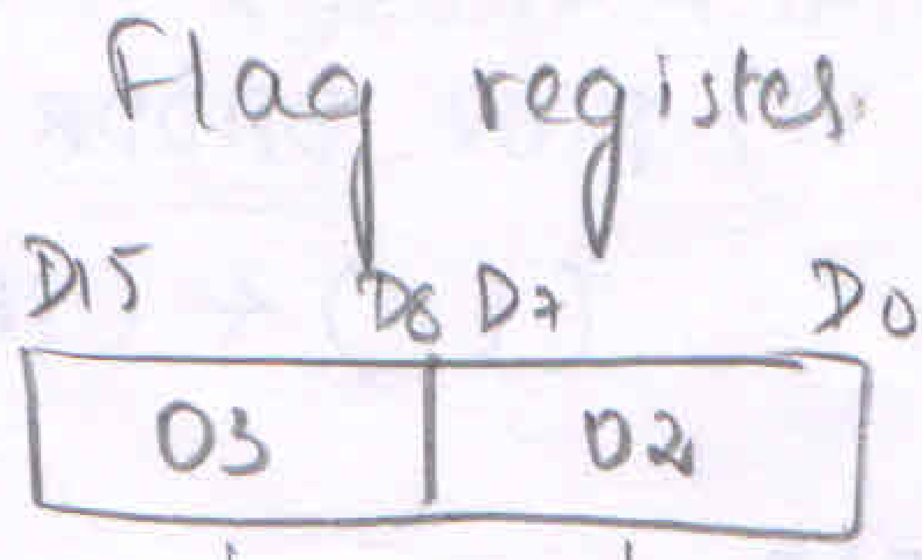
Syntax: PUSH F (Push flag to stack)

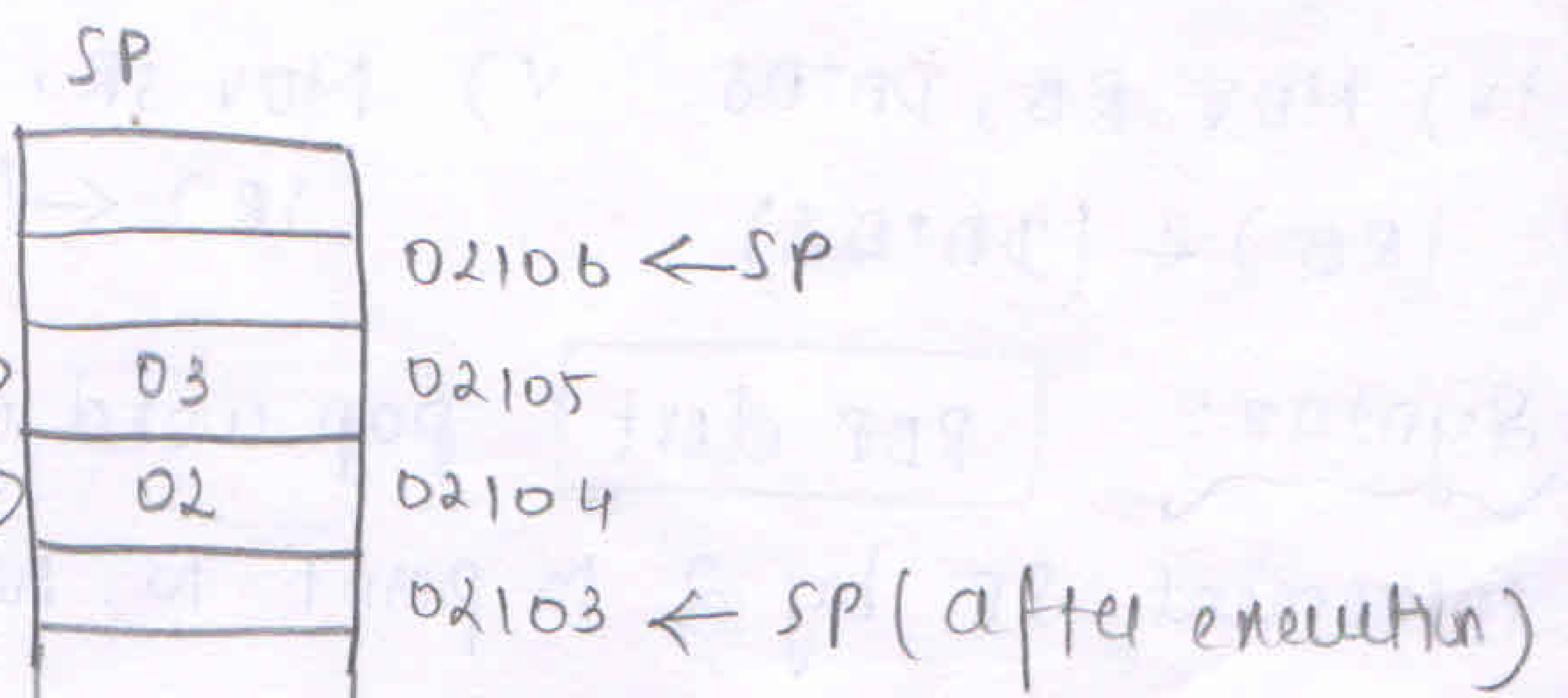
Instruction moves the flag reg to stack. First upper byte then lower byte is pushed on to it. The SP decremented by 2, for each push operation.

$(SP) \leftarrow (SP) - 2$, $((SP)) \leftarrow (SFR)$

Flag register

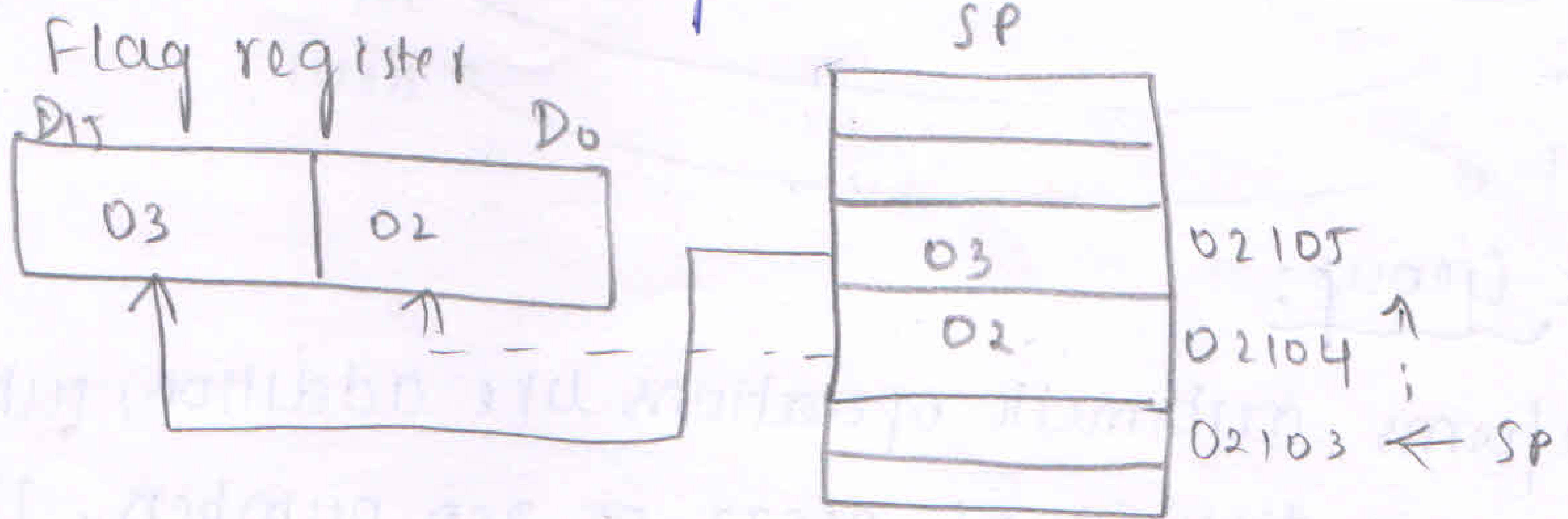
SP





syntax : POPF - pop flags from stack

Loads flag register completely from content of m/y location. The SP is incremented by 2 for each pop operation. First lower order then higher order.



(SFR) ← ((SP)), (SP) ← (SP) + 2

syntax: LEA reg, mem load effective address.

Transfer offset address of 'src' to destination reg.

Eg: LEA RW, DADDR

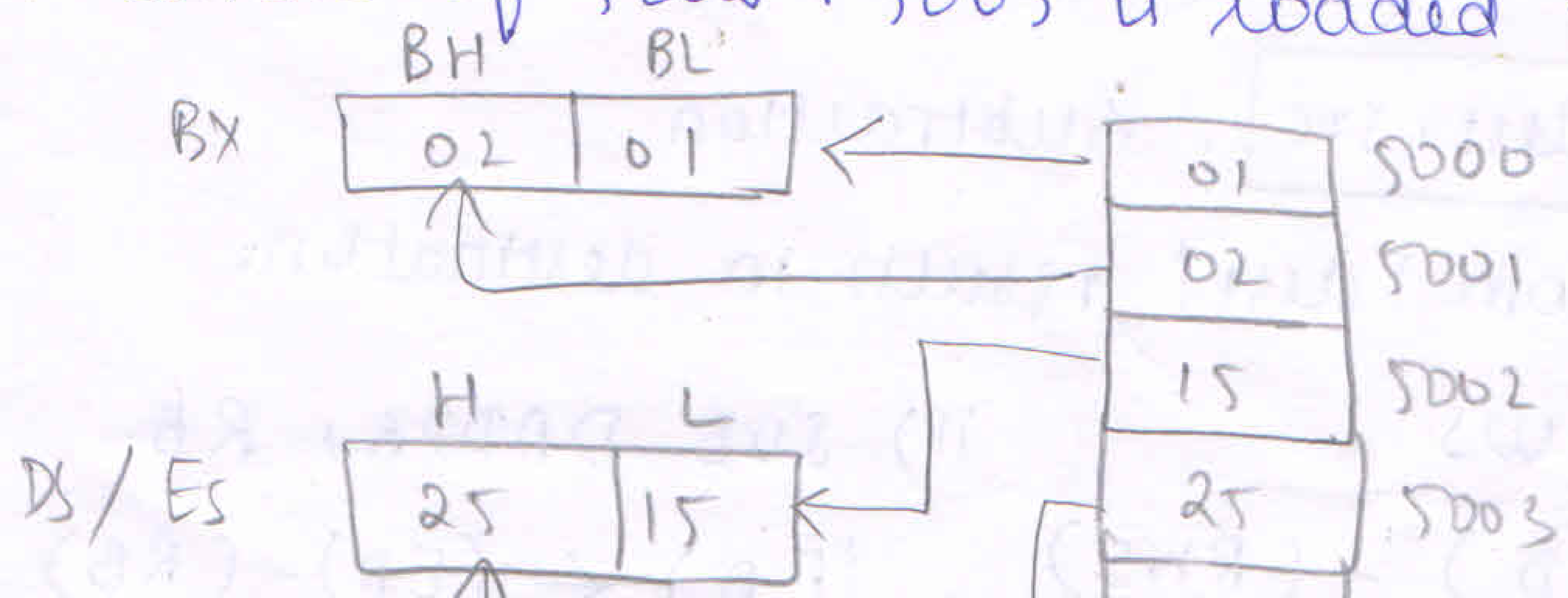
(RW) ← DEA.

syntax: LES reg16, mem load pointer using ES

(RW) ← (EA), (ES) ← (EA) + 2

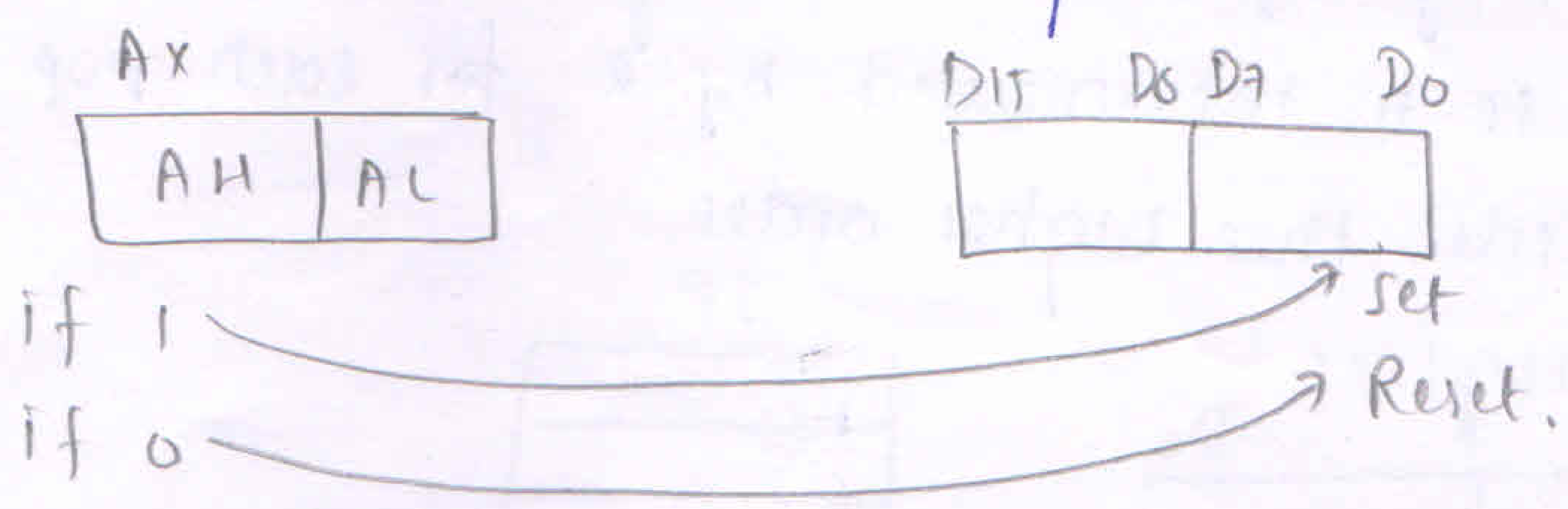
LES RW, DADDR.

Eg: LES BX, 5000H. Loads content of 5000 & 5001 to BX and content of 5002 & 5003 is loaded to ES register.



syntax: SAHF (Store all lower byte of flag reg)

Set / reset condition code flags in lower byte of reg.



Arithmetic Group:

Performs arithmetic operations like addition, subtraction, multiplication + division of ASCII or BCD numbers. The increment + decrement operations also included in it.

syntax: ADD dest, src Addition

Adds 'src' + 'dest' + replaces the original contents of 'dest'

i) ADD RBD, RBS
(RBD) ← (RBD) + (RBS)

ii) ADD DADDR, RD
(EA) ← (EA) + (RB)

syntax: ADC dest, src Add with carry

Sums 2 binary operands results in destination. If CF is set, 1 is added to destination.

i) ADC RB, DADDR
(RB) ← (EA) + (RB) + (CF)

ii) ADC AL, DATA8
(AL) ← (AL) + (DATA8) + C

syntax: SUB dest, src subtraction

Subtract 'src' from 'dest' results in destination.

1) SUB RWD, RWS
(RWD) ← (RWD) − (RWS)

ii) SUB DADDR, RB
(EA) ← (EA) − (RB)

Syntax: **SBB dest, src** - subtract with borrow

subtract source from dest + subtracts 1 extra if CF set.

i) SBB RB, DADDR

(RB) ← (RB) - (EA) - (CF).

Syntax: **MUL src** unsigned multiply

Source may be register or memory. If 'src' is byte → AL used. Result → AX.

If 'src' is word → AX x 'src' + DX : AX → results.

Eg: MUL RB

(AX) ← (AL) x (RB)

MUL RW

(DX) (AX) ← (AX) x (RW)

Syntax: **IMUL src** signed multiply

Multiplication of 'src' result in acc. 'src' byte → AL result → AX. If 'src' word → AX x 'src' result → DX : AX

Eg: IMUL RW

(DX) (AX) ← (AX) x (RW)

Syntax: **DIV src** Divide

If 'src' Byte → AX ÷ 'src' quotient in AL, remainder AH.

if 'src' word → (DX : AX) ÷ 'src' quotient in AX, remainder DX.

Eg: DIV RW

(DX) (AX) ← (DX) (AX) / (RW)

0 → +ve sign

1 → -ve sign

Syntax: **IDIV src** signed integer division

if 'src' Byte → AX ÷ 'src'

if 'src' word → (DX : AX) ÷ 'src'

Syntax: SBB dest, src – subtract with borrow

subtract source from dest + subtracts 1 extra if CF set.

i) SBB RB, DADDR

(RB) ← (RB) – (EA) – (CF).

Syntax: MUL src unsigned multiply

Source may be register or memory. If 'src' is byte → AL used. Result → AX.

If 'src' is word → AX x 'src' + DX : AX → results.

Eg: MUL RB

(AX) ← (AL) x (RB)

MUL RW

(DX) (AX) ← (AX) x (RW)

Syntax: IMUL src signed multiply

Multiplication of 'src' result in acc. 'src' byte → AL

result → AX. If 'src' word → AX x 'src' result → DX : AX

Eg: IMUL RW

(DX) (AX) ← (AX) x (RW)

Syntax: DIV src Divide

If 'src' Byte → AX ÷ 'src' quotient in AL, remainder AH.

if 'src' word → (DX : AX) ÷ 'src' quotient in AX, remainder DX.

Eg: DIV RW

(DX) (AX) ← (DX) (AX) / (RW)

0 → +ve sign

1 → -ve sign

Syntax: IDIV src signed integer division

if 'src' Byte → AX ÷ 'src'

if 'src' word → (DX : AX) ÷ 'src'

Syntax: INC dest Increment.

Add 1 to destination. Dest may be memory or register.

Eg: INC RB INC DADDR

(RB) ← (RB)+1. (EA) ← (EA)+1.

Syntax: DEC dest Decrement

Subtract 1 from destination.

Eg: DEC RB

(RB) ← (RB) −1.

Syntax: CMP dest, src Compare

Subtract the source from destination but does not save result.

Eg: CMP RBD, RBS

(RBD) − (RBS)

Syntax: CBW - Convert Byte to word

Byte to be converted must be in AL

Result will be in AX.

Before	After (AH AL)
AL = 90 H	FF 90 (AX)
AL = 53 H	00 53 (AX)

Syntax: AAA ASCII adjust for addition

It is executed after ADD instruction that adds 2 ASCII.

It gives unpacked decimal digit.

AL<9 AF=0 it adds 06H to AL

AL<9 AF=1 it adds 06H to AL ↑ AH by 1

AH=23 AL=67 <9 AF=0 AH=0 AL=07

Logical Group:

The logical operations such as AND, OR, NOT, XOR performed. Rotate, shift & TEST also included in it.

syntax: `AND dest, src` logical AND

Results in dest

Eg: AND 3F0F, 0008H.

3F0F → 0011 1111 0000 1111
0008 → 0000 0000 0000 1000

0000 0000 0000 1000 → 0008H

syntax: `OR dest, src` logical OR

Eg: OR RBD, RBS

(RBD) ← (RBD) OR (RBS)

syntax: `XOR dest, src` E-XOR

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Eg: XOR RBD, RBS

(RBD) ← (RBD) XOR (RBS)

syntax: `TEST dest, src` Test for Bit pattern.

Logical compare instruction. Results not saved in dest.

Each bit of result bit is set to 1, if corresponding bits of both operands are 1, else bit is reset 0. AND operation will be performed,

Eg: TEST RBD, RBS

(RBD) AND (RBS)

Syntax: NOT dest   1's compliment

logical NOT. Inverts the bits of 'dest'

Eg: NOT AX

NOT [5000H]

If AX = 200FH.

AX ⇒ 0010 0000 0000 1111

↓ ↓ ↓ ↓

1101 1111 1111 0000

D F F 0.

Shift Instruction:

Syntax: SHL/SAL dest, sourc (Shift logical / Arithmetic left)

This instruction shift the opeand bit by bit to left & insert zeeos in least significant bit (LSB). The shift opeation through caey.

Eg: SHL AX, 2

if AX = ACA5

opeand 1010 1100 1010 0101

SHL 1 1010110010100 1010 ← inserted.

SHL 2 0101 100101001 0100 ← inserted

Syntax: SHR dest, sour   shift logical Right

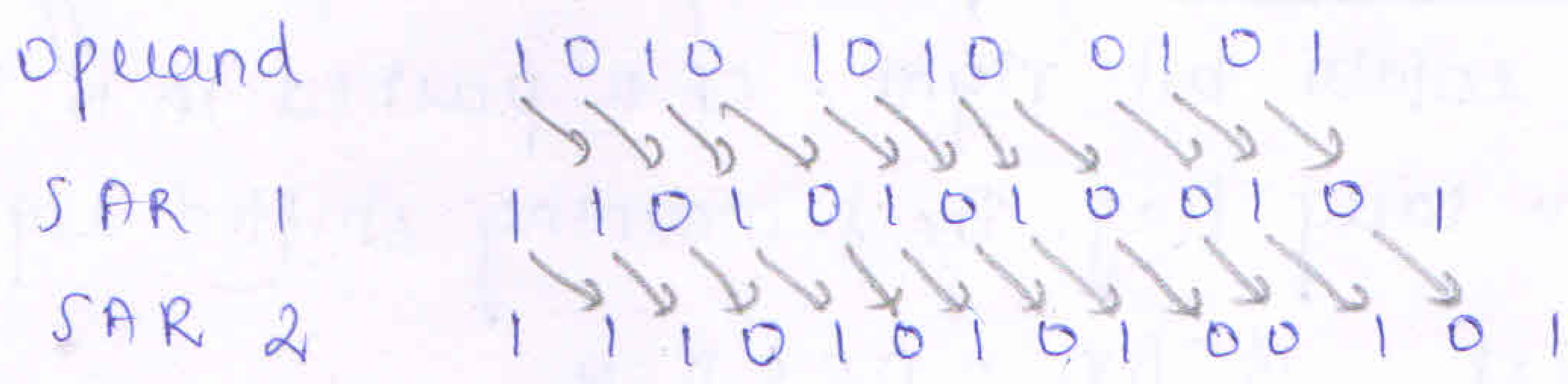
This instruction shift the opeand bit by bit to right & insert zeeos in Most significant bits (MSB).

Syntax: | SAR dest, sour | Shift Arithmetic Right

This instruction performs right shift on the operand by insert 1 in MSB. Its through carry flag

Eg: SAR BX, 2

if BX = AA5



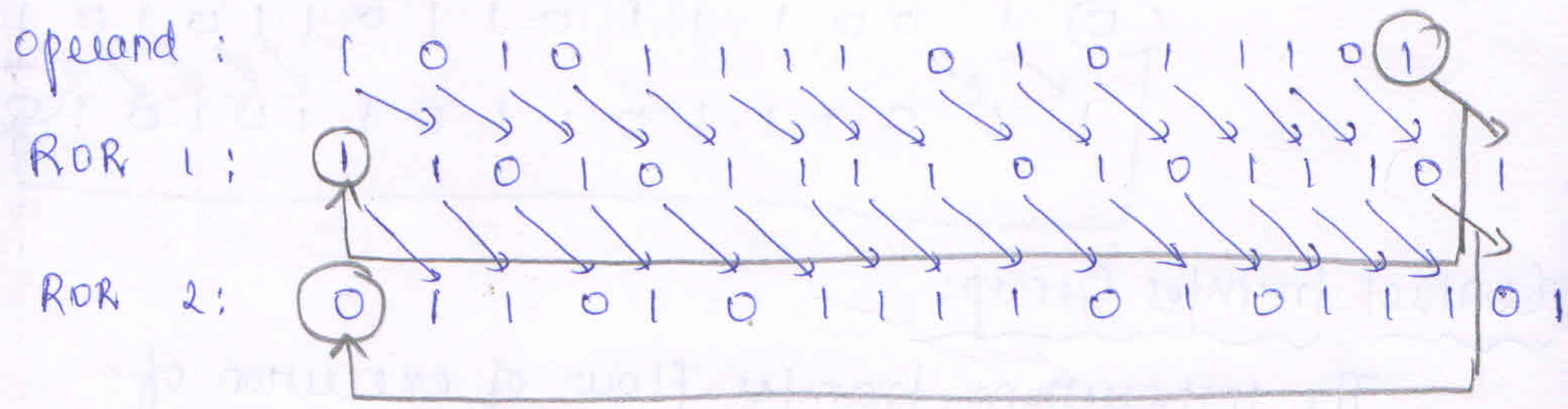
Immediate operand not allowed in any shift instructions.

Rotate instructions:

Syntax: | ROR dest, source | Rotate right without carry.

The instructions rotate the contents of bits right, excluding carry. The LSB pushed into CY flag & simultaneously transferred into MSB position. The remaining bits shifted right by specified positions.

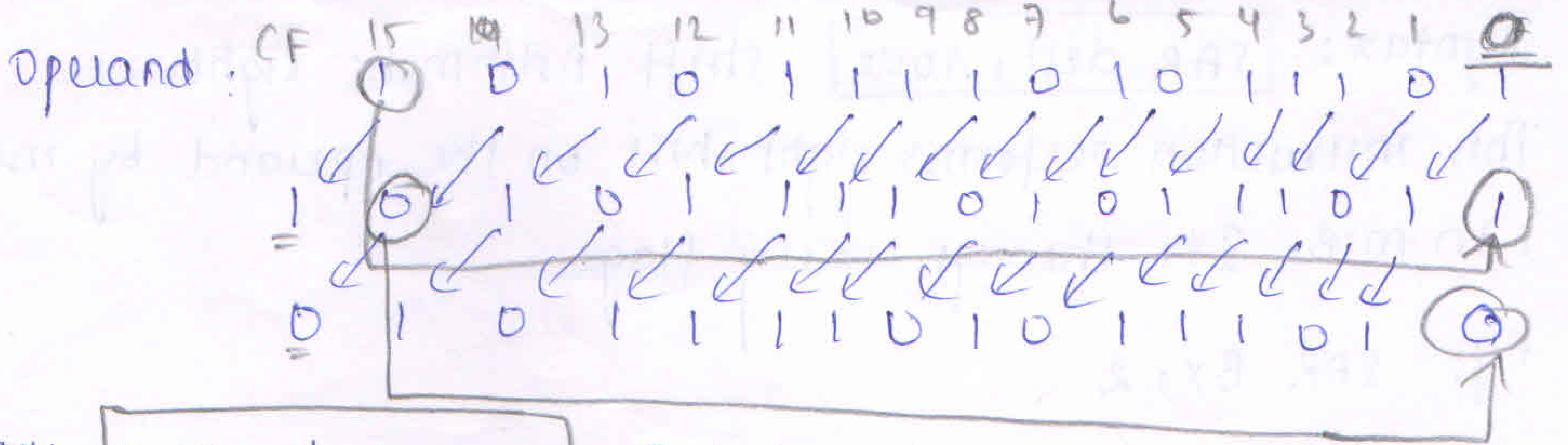
Eg: ROR BX, 2 if BX = AF5D H.



Syntax: | ROL dest, Source | Rotate left without carry

The instructions rotate the bits left, excluding carry,

The MSB pushed into CY flag

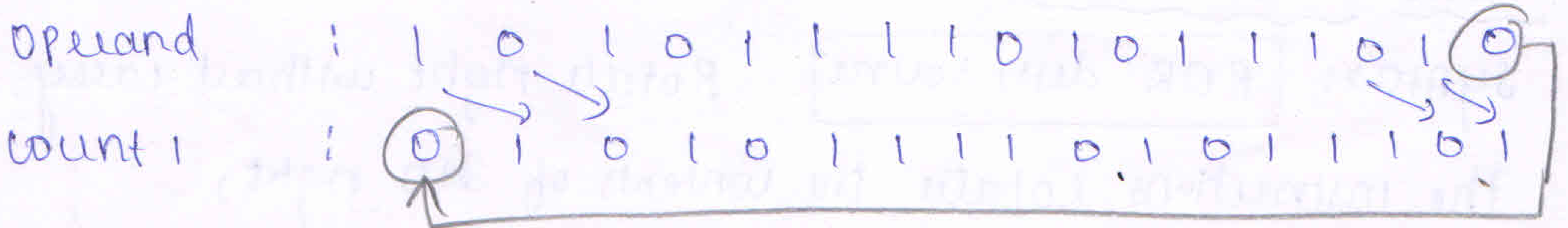


Syntax: RCR dest, sour Rotate right through carry

The instructions rotate bit right. CF is pushed in to MSB, LSB pushed into carry flag. The remaining shifted right

E.g: RCR BX, 1 if BX = AF5D H.

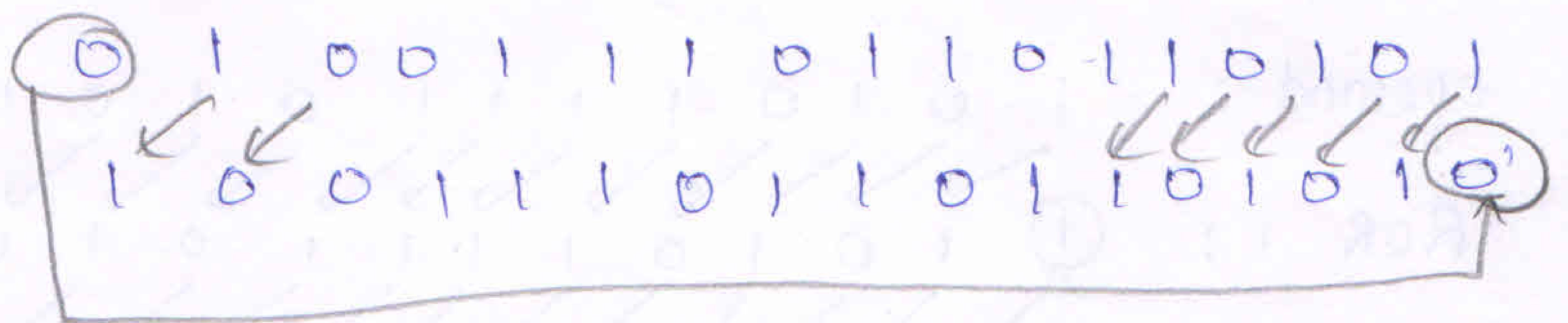
Bit position : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 CF



Syntax RCL dest, sour Rotate left through carry

Eg: RCL BX, 1 if BX = 9DB5 H.

Bit position : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 CF



Control transfer Group:

The instructions transfer flow of execution of program to a new address specified in instruction directly or indirectly. 2 types

Unconditional control transfer:

The control is transferred to specified location independent of any status or condition.

Eg: CALL, RET, INT N, INT O, JMP, LOOP, IRET. etc...

Conditional control transfer:

The control is transferred to specified location provided the result of previous operand satisfies condition.

Eg: JZ, JS, JO, JP, JC etc...

Syntax: | CALL PROC-NAME | procedure call.

This instruction used to call a subroutine (procedure) from main program. 2 types of procedure 1) NEAR CALL 2) FAR CALL

In near CALL → pushes IP reg. on stack.

In far CALL → pushes IP + CS both on stack

Near CALL: It is call to sub program which is same CS as CALL instruction.

Eg for far call:
CALL RW
((SP)) ← (IP), (SP) ← (SP) - 2, (IP) ← RW

Eg for near call:
CALL PROC_NAME
((SP)) ← (IP), (SP) ← (SP) - 2, (IP) ← (IP) + DISP.

Near call operation: i) SP ↓ 2

ii) copies offset of next instruction after CALL on STACK.

iii) It loads IP with offset of ...

Far CALL operation: i) SP ↓ 2

ii) copies content of CS

iii) SP ↓ 2

iv) copies offset of instruction after CALL to stack

v). It loads CS with segment of program

vi) IP with offset of 1st instruction.

Far call: It is call to sub program which is in diff segment contains call instruction.

Syntax: RET Return from procedure

Eg: RET nBytes

Transfers control from a procedure back to instruction address saved on stack.

* RET 1byte

(IP) ← ((SP)), (SP) ← (SP) + 2 for NEAR CALLS.

Returns from subroutine in current segment.

Syntax: INT N - Interrupt Type N

When INT N instruction executed the type byte N is multiplied by 4. IF enabled.

Eg: INT 20
↓
Type N ⇒ 20 * 4 = 80 H

IP : CS ⇒ 0000 : 0080 H.

Syntax: INT 0 Interrupt on overflow

This command executed when OF is set. It generates INT 4 which causes the code addressed by 0000:0010H to be executed.

Syntax: IRET Return from ISR

When an interrupt service routine is to be called before transferring control. IP, CS stored on stack to indicate the location where the execution is to be continued, after ISR is executed. IRET is executed. IP, CS values retreived from stack to continue execution of main program.

Syntax: JMP LABEL unconditional jump.

Unconditionally transfer control to 'LABEL'.

Eg: JMP LABEL

(IP) ← (IP) + DISP (displacement)

Jump directs to program memory locations identified by label. The DISP added to IP will computed as 8bit or 16 bit signed number.

J <COND> LABEL - conditional Jump

Conditional jump occurs after an arithmetic/logic or compare instructions,

mnemonics	Meaning	Jump condition.
JZ	Jump if zero	ZF=1
JNZ	Jump if non zero	ZF=0
JC	Jump carry	CF=1

JO Jump if overflow OF=1

JP Jump parity PF=1

syntax: LOOP LABEL Decrement CX & loop if $CX \neq 0$.

Decrement CX by 1 & transfers control to label if $CX \neq 0$.

$(CX) \leftarrow (CX) - 1$ if $CX \neq 0$.

then $(IP) \leftarrow (IP) + DISP8$.

Miscellaneous instruction group:

* CLC - clear carry → clear carry flag $(CF) \leftarrow 0$.
* CMC - complement carry → Toggles the CF $(CF) \leftarrow (\overline{CF})$
* CLI - clear interrupt flag → $(IF) \leftarrow 0$.
* STC - set carry → set CF=1 → $(CF) \leftarrow 1$
* STI - set interrupt flag $(IF) \leftarrow 1$.
* HLT - Halt CPU. The processor enters halt state.
* NOP - No operation. It do-nothing instruction.
* ESC - Escape. The CPU treats as NOP but places the contents of m/y location on bus.
* WAIT - wait for TEST i/p pin to go low.

  It holds operation of processor with current status till logic levels on $\overline{TEST}$ pin goes low.
* LOCK - Lock Bus

  It used to avoid 2 processor from updating