Logic and Control Instructions

Objectives

- **Control transfer operations**
- **Logical comparisons**
- **Logical and bit-wise operations**
- **Program organization**

Control Transfer Operations

A *transfer of control* is a way of altering the order in which statements are executed.

- ***Unconditional transfer*** -- branches to a new location in all cases -- JMP, LOOP, CALL
- ***Conditional transfer*** -- branches if a certain condition is true. The CPU interprets true/false conditions based on the content of the CX and Flags registers -- JZ, JE, JNZ, JNE, JC, JNC

Compare Operations

- **CMP**
- **TEST**

Logical and Bit-wise Operations

Logical operations --

- AND, OR, NOT, XOR

Shift and rotate --

- SAR/SHR
- SAL/SHL
- RCR/ROR
- RCL/ROL

JMP Instruction

[label:]	**JMP**	**[option]**	**destination label**

```
JMP Label               ;in current
                         segment
JMP NEAR PTR Label;near: in current
                               segment
JMP SHORT Label    ;in current seg
JMP FAR PTR Label ;to different seg
```

Example

```
0100 B4 02  Start: MOV AH,2
0102 B2 41         MOV DL,’A’
0104 CD 21         INT 21H
0106 EB F8         JMP Start
0108 ...
```

`EB:` short jump

`E9:` near jump

Instructions Addressing

- ***Short* address --**
  - limited to a distance of -128 to 127 bytes of instructions, 1 byte offset
- ***Near* address --**
  - limited to a distance of -32,768 to 32,767 bytes of instructions within the same segment, 1-2 words offset
- ***Far* address --**
  - over 32K or another segment

Distance Rules

Instruction	Short -128 to 127 Same segment	Near -32K to 32K Same segment	Far Over 32K Another segment
JMP	yes	yes	yes
Jcond	yes	yes (386+)	no
LOOP	yes	no	no
CALL	n/a	yes	yes

Example

```
Label1:   JMP SHORT Label2
            :
            :
Label2    JMP Label1
```

```
        PAGE 60, 123
TITLE   JUMP program
        .MODEL SMALL
        .CODE
        ORG     100H
;-----------------------------------------
Main    PROC    NEAR
        MOV     AX,01   ;
        MOV     BX,01   ;
        MOV     CX,01   ;

A20:    ADD AX,01       ;Add 01 to AX
        ADD BX,AX       ;Add AX to BX
        SHL CX,1        ;Double CX
        JMP A20         ;Repeat at label A20
Main    ENDP            ;end of procedure
        END             ;end of program
```

```
                       PAGE 60, 123
               TITLE   JUMP program
                       .MODEL SMALL
0000                   .CODE
                       ORG    100H
               ;-----------------------------------------
0100           Main    PROC    NEAR
0100  B8 0001           MOV    AX,01   ;
0103  83 C3 01          MOV    BX,01   ;
0106  B9 0001           MOV    CX,01   ;

0109  83 C0 01 A20:     ADD AX,01       ;Add 01 to AX
010C  03 D8             ADD BX,AX       ;Add AX to BX
010E  D1 E1             SHL CX,1        ;Double CX
0110  EB F7             JMP A20         ;Repeat at label A20
0112           Main     ENDP            ;end of procedure
                        END             ;end of program
```

LOOP Instruction

- repeat a block of statements with a specific number of times
- CX register is automatically used as a counter and decremented each time the loop repeats
- does not change flag
- destination must be (short) -128 to 127 bytes from the current location

LOOP Instruction

[label:]	LOOP	destination label

- The LOOP instruction subtract 1 from CX register
- if CX is not zero, control transfer to destination
- LOOPE/LOOPZ, LOOPNE/LOOPNZ
- LOOPW, LOOPD (386) uses the 32-bit ECX register

Example

```
              MOV CX,5  ;initialized CX
Start:
                :
                .
              LOOP Start;jump to Start
```

Flag Register

- Some instructions, when executed, change the status of the flags
- Different instructions effect different flags
- Some instructions effect more than one flag, and some do not effect any flags
- There are instructions that test the flags and base their actions on the status of those flags, e.g., Conditional jump instructions

Flags Register

O = Overflow -- indicate overflow of the left most bit following arithmetic

D = Direction -- determine left or right direction for moving or comparing data

I = Interrupt -- indicate that all interrupts to be processed or ignored

T = Trap -- permit operation of the processor in single-step-mode

S = Sign -- indicate the resulting sign of an arithmetic operation, 0 (negative), 1 (positive)

Z = Zero -- indicate the resulting sign of an arithmetic or comparison operation, 0 (nonzero), 1 (zero) result

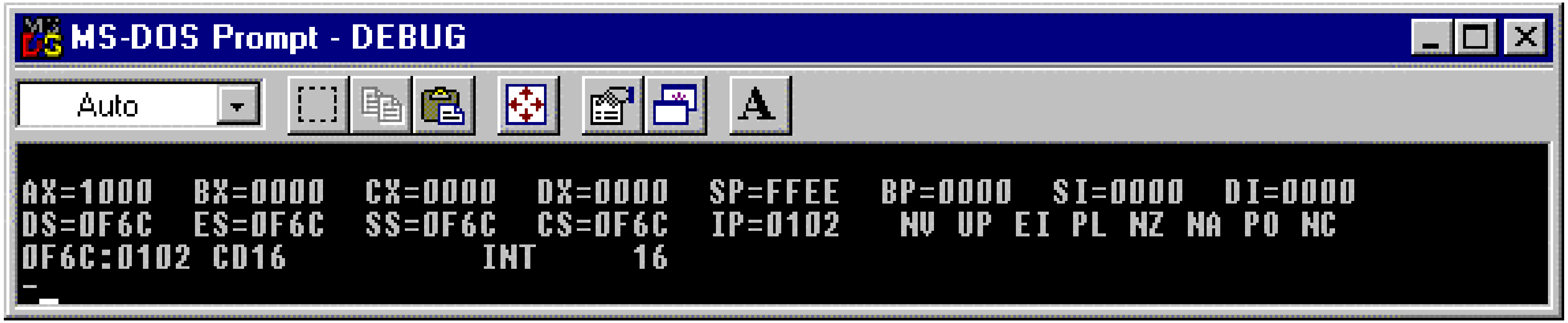
A = Auxiliary carry -- contain a carry out of bit 3 on 8-bit data

P = Parity -- indicate even or odd parity of a low-order 8-bit data operation

C = Carry -- contain the leftmost bit

x = undefined

Flags Registers



```
AX=1000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F6C  ES=0F6C  SS=0F6C  CS=0F6C  IP=0102   NV UP EI PL NZ NA PO NC
0F6C:0102 CD16          INT     16
-
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	O	D	I	T	S	Z	x	A	x	P	x	C

CF -- Carry Flag

- **Is set when the result of an *unsigned* arithmetic operation is too large to fit into the destination, or shift and rotate operations**

  Example: `AL = FFH`

  ```
  ADD AL,01H
  ```

  will set CF = 1, but **`INC AL`** will not set CF.

- **JC and JNC use this flag**

Example

- Suppose `AH = 00H, AL = FFH`

```
. . .    ;AX = 0000 0000 1111 1111
ADD  AX,1;AX = 0000 0001 0000 0000
          CF is set to 0
```

- The carry flag contains the borrow after a subtraction.

Example: `AL = 00H`

```
SUB AL,1;AL = 1111 1111 and CF = 1
```

CF -- Carry Flag

- Several other instructions effect the carry flag: CMP, TEST, SHL, SHR, etc.
- Two instructions explicitly change the carry flag:
  - **CLC: clear CF to 0**
  - **STC: set CF to 1**

AF -- Auxiliary Flag

- Is set when an operation causes a carry from bit 3 to bit 4 (or borrow from bit 4 to bit 3) of operand.

Example: `AL = 9BH = 1001 1011`

```
ADD  AL,7 ; AL = A2H = 1010  0010
```

CF = 0

AF = 1

ZF -- Zero Flag

- Effected by arithmetic and logic operations and the **`CMP`** operation
- Set to 1 if result of operation is zero; otherwise it is reset to 0.
- Used by conditional jumps such as JZ, JE, JNZ and JNE

SF -- Sign Flag

- Set according to the sign of the result of an arithmetic operation.
- If result of last operation is negative, then SF is set to 1; otherwise, SF is set to 0
- Use it only when doing signed arithmetic.
- Used by conditional jumps such as JS, JL, JNS, JGE

OF -- Overflow Flag

- Effected when signed numbers are added or subtracted.
- An overflow indicates that the result does not fit in the destination operand.

Example: `ADD AL, BL`

If result is not in (-128, 127), an overflow occurs.

- Use Overflow with signed arithmetic.

  JO, JNO among others

Other Flags

- PF (Parity Flag)
- TF (Trap Flag)
- IF (Interrupt Flag)

Example

Before: **`DL = 12H`**

```
        ADD  DL, 33H
```

After: **`DL = 45H`**

CF= 0 , ZF= 0, SF = 0, OF = 0

Example

Before: **`DL = F3H`**

```
ADD DL,F6H
```

After: **`DL = E9H`**

CF = 1, ZF = 0, SF=1, OF=0

- Two interpretations: Signed or Unsigned
  - **Unsigned: Ignore SF and OF**
  - **Signed: Ignore CF**

Unsigned Operation

Hex	Binary	Interpretation
	(Unsigned)	Decimal
F3H	1111 0011	243
F6H	1111 0110	246
1 E9H	11110 1001	489

Result: a sum of E9H and a carry out of 1, set CF=1

Unsigned Operation

Hex	Binary (2's complement)	Interpretation Decimal
F3H	1111 0011	-13
F6H	1111 0110	-10
1 E9H	11110 1001	-23

In 2's complement addition, carry is discarded
result: DL = E9H which is interpreted as -23
SF = 1, CF = 0, OF = 0

CMP Instruction

[label:]	CMP	reg/mem,reg/mem/imd

- Compare two numeric data fields
- Effects the flags: ZF, SF, CF, AF, OF, PF

Example:

```
          CMP DX,10
          JE  P50
           ...      ;continue if not equal
     P50:  ...      ;Jump point if DX is zero
```

- JE tests only the ZF flag

CMP Instruction

CMP subtracts the second operand from the first and sets the flags accordingly

- **if result is equal to 0 set ZF to 1**

  => two operands are equal
- **if result is positive set SF to 0 or CF to 0**

  => first operand is greater than second operand
- **if result is negative set SF to 1 or CF to 0**

  => first operand is less than second operand

CMP Unsigned Operands

CMP Results	CF	ZF
Destination < Source	1	0
Destination = Source	0	1
Destination > Source	0	0

CMP Signed Operands

CMP Results	ZF	SF, OF
Destination < Source	?	SF <> OF
Destination = Source	1	?
Destination > Source	0	or SF = OF

Conditional Jump Instructions

`[label:]`	`Jcond`	`Short address`

- Transfer control depending on status of the flags register
- test one or more of the following flag bits:

  SF ZF CF PF OF
- If the condition under test is true, then branch to Label; otherwise, the next sequential instruction (immediately following the jump) is executed

Conditional Jump Instructions

```
A20:
    ...
    DEC CX ;decrement CX
    JNZ A20;Jump if ZF = 0
    ...
```

Signed and Unsigned Data

Example: `CX=11000110, DX=00010110`

```
        MOV AX, 0
        CMP CX, DX
        JE  P50    ;jump if ZF = 1
        MOV AX, 1
   P50: ...
```

What is the contents of AX?

Jump for Unsigned Data

- Using Above and Below

– **JE/JZ**	**Jump if equal/jump if zero**
– **JNE/JNZ**	**Jump if not equal**
– **JA**	**Jump if above**
– **JAE**	**Jump if above or equal**
– **JB**	**Jump if below**
– **JBE**	**Jump if below or equal**

- Test the ZF and/or the CF flag bits.

Jump for Signed Data

- Using greater and less
  - **JE/JZ ZF**
  - **JNE/JNZ ZF**
  - **JG jump if greater than (OF, SF)**
  - **JGE jump if greater or equal (OF, SF, ZF)**
  - **JL jump if less than (OF, SF)**
  - **JLE jump if less than or equal (OF, SF, ZF)**

Test the ZF, SF and/or OF flag bits.

Arithmetic Test

– JCXZ	jump if CX is zero
– JC, JNC	CF
– JO, JNO	OF
– JP/JPE, JNP/JPO	PF
– JS, JNS	SF

Jumps Based on General Comparisons

Mnemonic	ZF	CF	PF	CX
JZ	1			
JE	1			
JNZ	0			
JNE	0			
JC		1		
JNC		0		
JCXZ				0
JP			1	
JNP			0	

Jumps Based on Unsigned Comparisons

Mnemonic	ZF	CF
JA	0	0
JNBE	0	0
JAE	0	
JNB	0	
JB		1
JNAE		1
JBE	1	or 1
JNA	1	or 1

Jumps Based on Signed Comparisons

Mnemonic		SF
JG	0	0
JNLE	0	0
JGE		=OF
JNL		=OF
JL		¹OF
JNGE		¹OF
JLE	1	or ¹OF
JNG	1	or ¹OF
JS		1
JNS		0

Logical and Bit-wise Operations

Logical operations --

- AND, OR, NOT, XOR

Shift and rotate --

- SAR/SHR
- SAL/SHL
- RCR/ROR
- RCL/ROL

Logical Operations

`[label:]`	`operation`	`reg/mem,reg/mem/imm`

`[label:]`	`NOT`	`reg/mem`

- **AND**
- **OR**
- **XOR**
- **NOT**
- **They effect the ZF, SF and PF**

Examples

```
                  AND  OR  XOR  TEST
Operand 1:       0101 0101 0101 0101
Operand 2:       0011 0011 0011 0011
                 --------------------
Rslt in Operand 1:0001 0111 0110 0101
Result:                          0001
```

OR Operation

- May be used to test if a register is zero

```
OR DX,DX   ;set ZF and SF
JZ ...
```

- May be used to test the sign of a register

```
OR DX,DX
JS ...
```

- Better use the `CMP` instruction for the above

AND Operation

- May be used to test for a specific bit

```
MOV BL,00001000
AND BL,AL      ;The result is equal
                to 4th bit of AL
JZ  ...
```

- Another way

```
AND AL, 00001000
JZ  ...
```

AND Operation

- May be used to Clear a register

```
AND BL,0H
```

- May be used to mask some bits of a register

```
AND BL,0FH      ;zeros left 4 bits
AND BL,11000011B;Zeros the middle 4
                 bits
AND BL,11111101 ;Zeros the second bit
```

TEST Operation

Performs the same function as the AND but does not modify the destination register

```
TEST AL,00001000;is the 4th bit of
JZ   ...            ;AL 0 ?
TEST BL,00000001;Does BL contain an
JNZ  ...             ;odd value ?
TEST CL,11110000;Are any of the 4
                 leftmost
JNZ  ...            ; bits in CL nonzero?
```

Shift Instructions

- Used to position or move numbers to the left or to the right within a register or a memory location
- They also perform simple arithmetic (multiply or divide by 2^n).
- Shift up to 8 bits in a byte, 16 bits in a word, 32 bits in a double word (386 and later)
- Shift Logically (unsigned) or arithmetically(signed data).

Shift Instructions

`[label:]`	`shift`	`register/memory,CL/immediate`

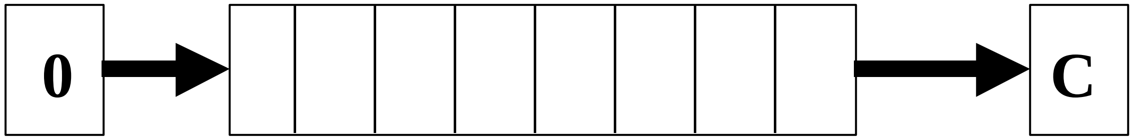
- 1st operand is data to be shifted.
- 2nd operand is the number of shifts
- register: can be any register except segment register
- for 8086, immediate value must be 1.
  - **Use CL if need to shift by more than one bit.**
- for later processors, immediate value can be any positive integer up to 31.

Shift Instructions

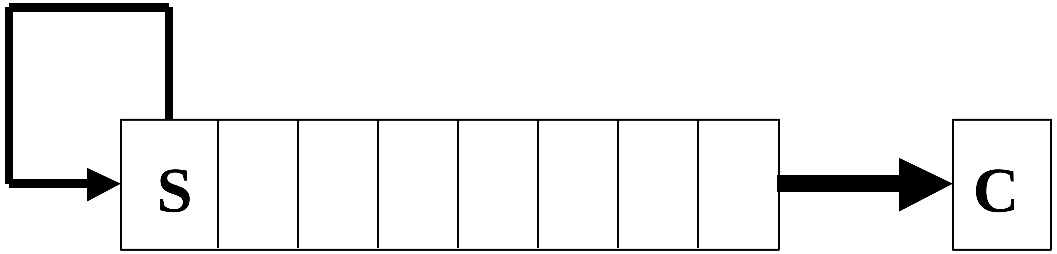
- Shift right
  - **SHR: Logical shift right**
  - **SAR: Arithmetic shift right**
- Shift left
  - **SHL: Logical shift left**
  - **SAL: Arithmetic shift left**

Shift Right

SHR:



SAR:



SHR

Instruction	Binary	Decimal	CF
MOV AL,10110011B	1011 0011	179	-
SHR AL,01	0101 1001	89	1
MOV CL,02			
SHR AL,CL	0001 0110	22	0
(80286+)			
SHR AL,02			

SAR

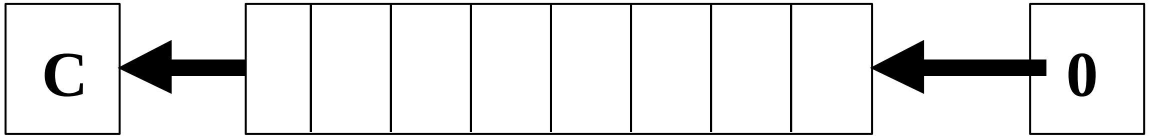
Instruction	Binary	Decimal	CF
MOV AL,10110011B	1011 0011	-77	-
SAR AL,01	1101 1001	-39	1
MOV CL,02			
SAR AL,CL	1111 0110	-10	0
(80286+)			
SAR AL,02			

SHR and SAR

- Right shifts are especially useful for halving values: i.e. integer division by 2
  - **Right shift by 2 bits => divide by 4**
  - **Right shift by 3 bits => divide by 8 etc.**
  - **SHR: for unsigned numbers**
  - **SAR: for Signed numbers**
- Much faster than the divide instruction.
- Shifting by 1 bit, the remainder is in CF.

Shift Left

SHL & SAL:



SHL

Instruction	Binary	Decimal	CF
MOV AL,00001101B	0000 1101	13	-
SHL AL,01	0001 1010	26	0
MOV CL,02			
SHL AL,CL	0110 1000	104	0
(80286+)			
SHL AL,02	1010 0000	160	1

SAL

Instruction	Binary	Decimal	CF
MOV AL,11110110B	1111 0110	-10	-
SAL AL,01	1110 1100	-20	1
MOV CL,02			
SAL AL,CL	1011 0000	-80	1
(80286+)			
SAL AL,02	1100 0000	-64	0

SHL and SAL

- **SHL and SAL are identical**
- **SHL for unsigned and SAL for signed**
- **can be used to double numbers.**
- **Each bit shift to the left , double the value**
  - shifting left by 2 bits = multiply by 4 etc.
- **Note: if after a left shift CF=1**
  - size of the register/memory location is not large enough for the result.

Example

A code segment that will multiply AX by 10

Assume the number N is the content of AX

```
SHL   AX, 1              ; AX = 2*N
MOV   BX, AX             ; save in BX
SHL   AX, 2              ; AX = 8*N
ADD   AX, BX             ; AX = 2*N+8*N
```

Rotate Instructions

- **Rotate binary data in a memory location or a register either from one end to the other, or through the CF.**
- **Used mostly to**
  - inspect specific bits
  - shift numbers that are wider that register size (I.e. wider that 16 bits in 8086/286).

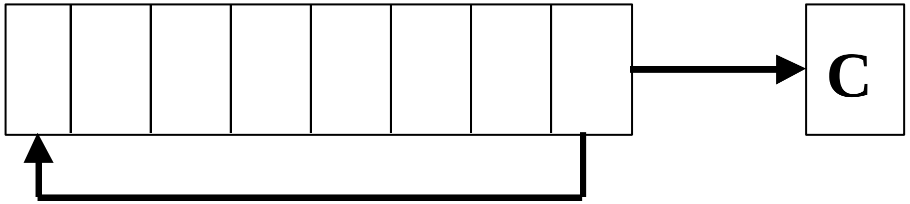
Rotate Instructions

`[label:]`	`rotate`	`register/memory,CL/immediate`

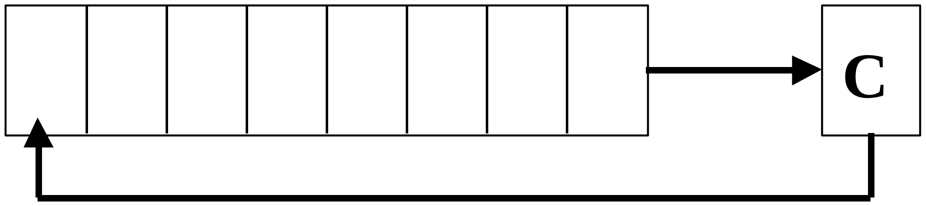
- **Rotate Right**
  - ROR
  - RCR
- **Rotate Left**
  - ROL
  - RCL

Rotate Right

ROR:

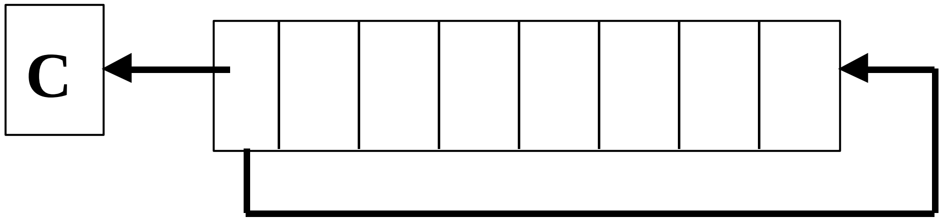


RCR:

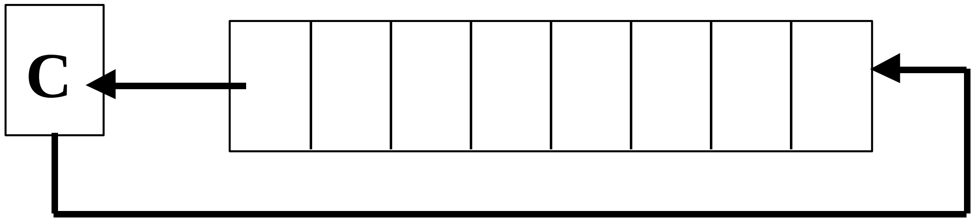


Rotate Left

ROL:



RCL:



Example

Instruction	Binary	CF	
MOV BL,10110100B	1011 0100	-	
ROR BL,01	0101 1010	0	
MOV CL,02			
ROR BL,CL	1001 0110	1	
MOV BL,10110100B	1011 0100	1	
RCR BL,01	1101 1010	0	
MOV CL,02			
RCR BL,CL	0011 0110	1	

Example

- **Rotate instructions are often used to shift wide numbers to the left or right.**

  Example: Assume a 48-bit number is stored in registers DX, BX, AX

  write a code segment to shift number to the left by one position:

```
SHL AX,1
RCL BX,1
RCL DX,1
```

Procedures

Syntax

```
proc-name       PROC [NEAR/FAR]
                ...
                ...
                RET
                ENDP
```

- NEAR indicates that the procedure is to be called from within current segment (default)
- FAR indicates that the procedure is to be called from other segments

Calling Procedure

[label:]	CALL	Proc-name
[label:]	RET	[Pop-value]

- **The CALL transfers control to the called procedure:**
  - save the current IP in the stack
  - load the IP with the address of the called procedure
- **The RET instruction returns control to the calling procedure.**
  - Restores the IP with the saved address
- **In general, the RET instruction is the last instruction in a procedure.**

CALL Procedure -- NEAR

CALL to a NEAR procedure

- push IP on top of the stack. The IP at the time of call contains the offset of the next instruction (after the CALL instruction)
- Load the offset of the first instruction of the called procedure into the IP register.

RETURN from a NEAR procedure

- Pop the top of the stack into the IP register

Instruction Queue is also saved/restored on CALL/RET

CALL Procedure -- FAR

CALL to a FAR procedure

push CS registers on top of the stack.

Load the CS register with address of new segment.

Push IP on top of the stack.

Load IP register with offset of the called procedure

RETURN from a FAR procedure

Pop the top of the stack into the IP register

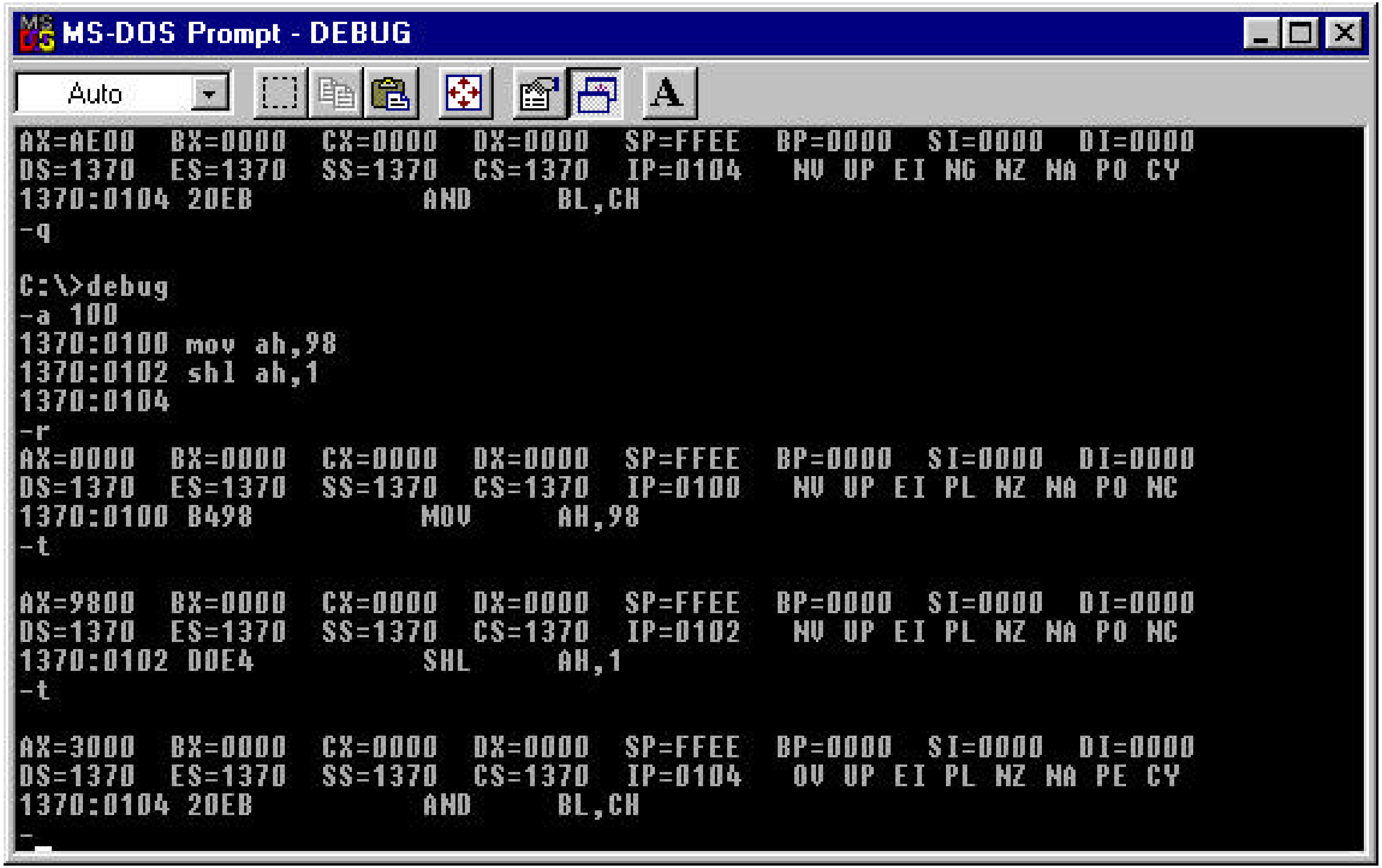
Pop the top of the stack into CS register

FAR procedure will be dealt with in Chapter 23.

```
                         .Model SMALL
OFFSET                   .Stack 64
                         .Data
                         .Code
0100                     MAIN          PROC   FAR
0100                     MOV    DL, 'O'
0103                     CALL   DISP
0105                     MOV    DL, 'K'
0107                     CALL   DISP
          MAIN           ENDP
0109      DISP           PROC   NEAR
0109                     MOV    AH, 2
010B                     INT    21H
010D                     RET
          DISP           ENDP
                         END    MAIN
```

Effect of program execution on stack

- **Initialization: . STACK size**
  - Each procedure call requires at least one word on stack to save current IP (for NEAR procedures)



```
AX=AE00  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1370  ES=1370  SS=1370  CS=1370  IP=0104   NV UP EI NG NZ NA PO CY
1370:0104 20EB          AND     BL,CH
-q

C:\>debug
-a 100
1370:0100 mov ah,98
1370:0102 shl ah,1
1370:0104
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1370  ES=1370  SS=1370  CS=1370  IP=0100   NV UP EI PL NZ NA PO NC
1370:0100 B498          MOV     AH,98
-t

AX=9800  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1370  ES=1370  SS=1370  CS=1370  IP=0102   NV UP EI PL NZ NA PO NC
1370:0102 D0E4          SHL     AH,1
-t

AX=3000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1370  ES=1370  SS=1370  CS=1370  IP=0104   OV UP EI PL NZ NA PE CY
1370:0104 20EB          AND     BL,CH
-
```