80X86 ISA & PROGRAMMING

LOGICAL INSTRUCTIONS

AND DESTINATION, SOURCE

- Logical ANDs each bit in the source with the corresponding bit in the destination

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

- ||ly OR XOR

NOT DESTINATION

- Complements each bit in the destination and stores the result back into the destination

- No Flags Affected

```
NOT AL
NOT BX
NOT BYTEPTR[SI]
```

NEG DESTINATION

- Does 2's complement on the data in the destination and stores the result back into the destination
- Cannot find 2's complement of -128 (8-bit) or -32,768 (16-bit)
- OF set indicating operation could not be done
- All Flags Affected

```
NEG     AL
NEG     BX
NEG     BYTEPTR[SI]
```

8086-80486

STRING INSTRUCTIONS

STRING INSTRUCTIONS

- 80x86 is equipped with special instructions to handle string operations
- String: A series of data words (or bytes) that reside in consecutive memory locations
- Operations: move, scan, compare

MOVS/MOVSB/MOVSW/MOVSD

- Copies a byte or word or double-word from a location in the data segment to a location in the extra segment
- Source – DS:SI
- Destination – ES:DI
- No Flags Affected
- For multiple-byte or multiple-word moves, the count to be in CX register
- Byte transfer, SI or DI increment or decrement by 1
- Word transfer, SI or DI increment or decrement by 2
- Double word transfer SI or DI increment or decrement by 4

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

Selects auto increment D=0

auto decrement D=1

operation for the DI & SI registers during string Ops

D is used only with strings

THE DIRECTION FLAG

COPY A BLOCK OF DATA FROM ONE MEMORY AREA TO ANOTHER MEMORY AREA- 50 DATA

```
.model tiny
.386
.data
array1    db      0a_h ,bc_h ,de_h ,0f5_h ,11_h, 56_h ,78_h  ,0ff_h ,0ff_h ,23_h 4ah, …
array2    db      50 dup(0)
.code
startup
          mov     cx,32_h
          lea     si,array1
          lea     di,array2

          cld
  rep     movsb
```

8086-80486

STRING INSTRUCTIONS

STRING INSTRUCTIONS

- 80x86 is equipped with special instructions to handle string operations
- String: A series of data words (or bytes) that reside in consecutive memory locations
- Operations: move, scan, compare

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

Selects auto increment D=0

auto decrement D=1

operation for the DI & SI registers during string Ops

D is used only with strings

THE DIRECTION FLAG

LODS

- Loads AL or AX or EAX with the data stored at the data segment
- Offset address indexed by SI register
- After loading contents of SI inc if D = 0 & dec if D = 1

- LODSB ; AL = DS:[SI]; SI = SI ±1
- LODSW ; AX = DS:[SI]; SI = SI ±2
- LODSD ; EAX = DS:[SI]; SI = SI±4

- LODS affects no FLAGs

STOS

- Stores AL, or AX or EAX into the extra segment memory at offset address indexed by DI register
- After storing contents of DI inc if D = 0 / dec if D = 1

- STOSB ; ES:[DI] = AL DI = DI ±1
- STOSW ; ES:[DI] = AX DI = DI ± 2
- STOSD ; ES:[DI] = EAX DI = DI ± 4

- STOS affects no FLAGs

Write an ALP to fill a set of 100 memory locations starting at displacement 'DIS1' with the value $F6_H$

```
.MODEL TINY
.DATA
DAT1      DB   100 DUP(?)
.CODE
.STARTUP
        MOV       DI, OFFSET DAT1
        MOV       AL, 0F6H
        MOV       CX, 100
        CLD
        REP  STOSB
.EXIT
END
```

SCAS

- Compares the AL with a byte of data in memory
- Compares the AX with a word of data in memory
- Compares the EAX with a double word of data in memory
- Memory is ES: DI
- Operands not affected - flags affected (subtraction)

- SCASB
- SCASW
- SCASD
- Can be used with prefix
- REPNE SCASB

Write an ALP to find the displacement at which the data $0D_H$ is present from an array of data stored from location DAT1. The number of bytes of data in the array is 80.

```
.MODEL TINY
.DATA
DAT1      DB   80 DUP(?)
.CODE
.STARTUP
        MOV      DI, OFFSET DAT1
        MOV      AL, 0D_H
        MOV      CX, 80
        CLD
        REPNE  SCASB
.EXIT
END
```

- Scanning is repeated as long as bytes are not equal & the end of the string not reached
- If $0D_H$ is found DI will point to the next address

CMPS

- Compares a byte in one string with a byte in another string or a word /dword in one string with a word/dword in another string
- DS: SI with ES: DI

- Flags affected
- Direction flag used for auto increment or decrement
- Can be used with Prefix

PROGRAM SEGMENT

```
MOV     SI, OFFSET STRING FIRST
MOV     DI, OFFSET STRING SECOND
CLD
MOV     CX, 100
REPE    CMPSB
```

- Repeat until end of string(CX≠0)

or until compared bytes are not equal

CMPSB

Source		Destination	
1000	67	2000	67
1001	56	2001	56
1002	4A	2002	4A
1003	67	2003	67
1004	AA	2004	AA

CMPSB

Source		Destination	
1000	67	2000	67
1001	56	2001	56
1002	67	2002	4A
1003	67	2003	67
1004	AA	2004	AA

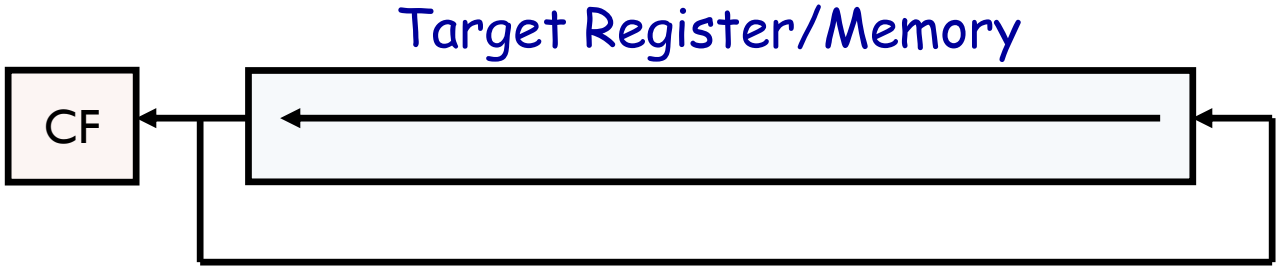
8086-80486

ROTATE OPERATIONS

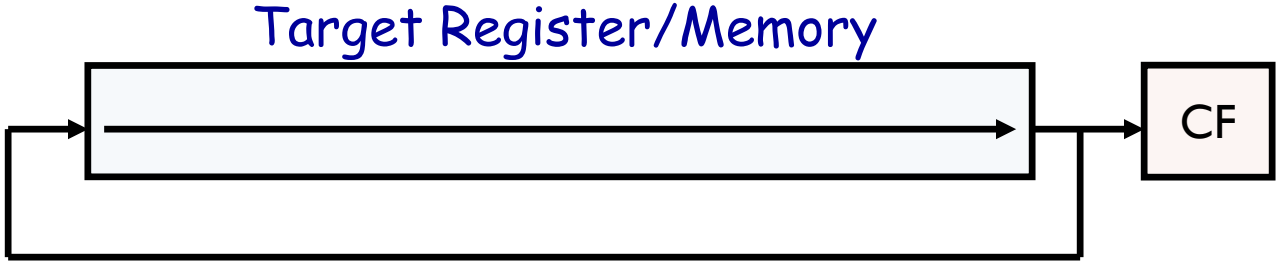
ROL DESTINATION, COUNT
ROR DESTINATION, COUNT

- USE CL FOR COUNT greater than 1 (***if 80386 count greater than 1 can be specified directly)***
- rol ax,1
- ror byteptr[si], 1
- mov cl,04h
- rol ax,cl
- ror byteptr[si], cl
- rol ecx,12

ROL



ROR



Flags Affected : CF

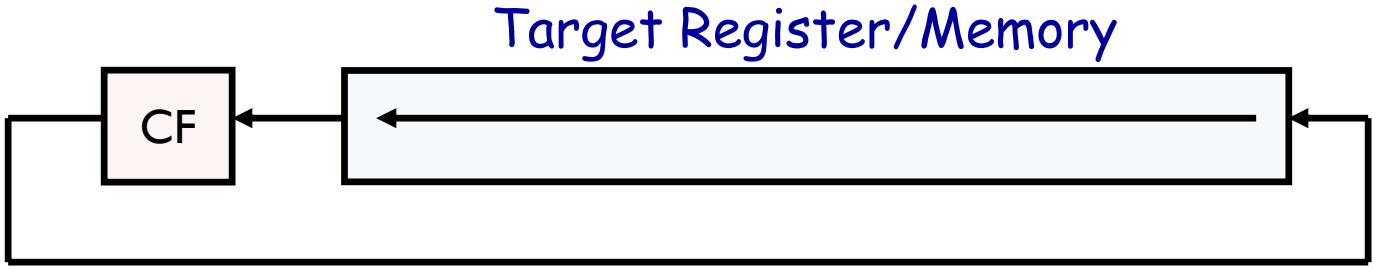
OF – If MSB changes – single bit rotate

OF-0

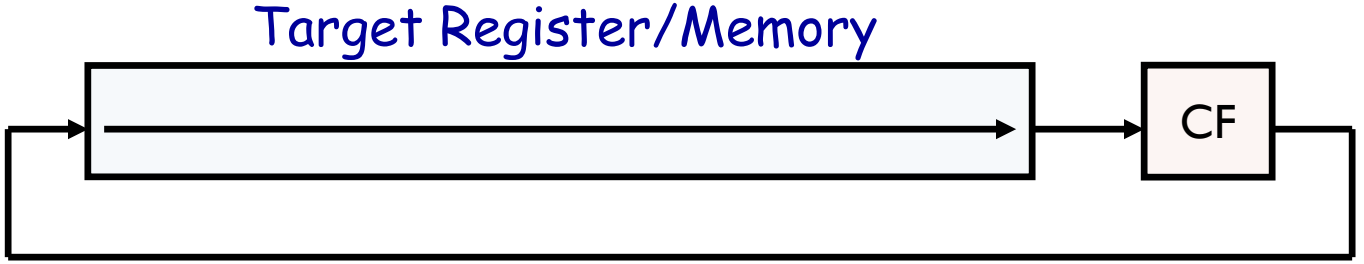
EXAMPLE

- ROL BL, 04
- Swap Nibbles
- ROR WORD PTR [BX], 04

RCL



RCR



WRITE A PROGRAM THAT COUNTS THE NUMBER OF 1'S IN A BYTE IN LOCATION DATA1 AND WRITES IT INTO LOCATION RES1

```
.Model Tiny
.data
DATA1   DB  0A7H
RES1    DB  ?
.code
.startup
        SUB     BL, BL          ;clear BL - no. of 1s
        MOV     DL, 8           ;rotate total of 8 times
        MOV     AL,DATA1
AGAIN:  ROL     AL,1            ;rotate it once
        JNC     NEXT            ;check for 1
        INC     BL              ;if CF=1 then inc count
NEXT:   DEC     DL              ;go through this 8 times
        JNZ     AGAIN           ;if not finished go back
        MOV     RES1,BL
.exit
end
```

8086-80486 ISA & PROGRAMMING

SHIFT OPERATIONS

SAL/SHL

- SAL/SHL or two mnemonics for the same operation
- Shifts each bit in the specified destination some number of bit positions to the left
- As bit shifted out of LSB 0 is put in LSB, MSB shifted to CF
- SAL/SHL destination, count
- Count in CL if count greater than 1 (except in 386 and above)

EXAMPLES

- SAL BX, 1
- MOV CL, 02
- SAL BP, CL
- SAL BYTE PTR[BX], 1
- SAL EAX,12

- Flags Affected : CF, ZF,SF
- OF – If MSB changes – single bit rotate
- PF - Affected – but has meaning only if 8-bit operation
- AF - Undefined

SHR

- Shift Operand bits right. Put Zero in MSBs
- SHR Destination, Count

- SHR BP, 1

- MOV BL, AL
- AND BL, 0FH
- MOV CL, 04H
- SHR AL, CL
- MOV BH, AL

SAR

- Shift Operand Bits Right. New MSB == OLD MSB
- SAR Destination, Count

```
SAR     DI, 1
MOV     CL, 02H
SAR     WORD PTR[BP], CL
```

8086-80486 –ISA & PROGRAMMING

MULTIPLY & DIVIDE

MUL/IMUL SOURCE

- Source times AL
- Source times AX
- Source times EAX
- Source can be a register or memory location

MUL/IMUL SOURCE

- Result for Byte multiplication in AX
- Result for Word multiplication in DX :AX
- Result for Dword Multiplication EDX:EAX
- CF and OF zero if MSB/MSW/MSD zero
- AF,PF,SF,ZF –undefined
- CBW/CWD

CBW/CWD

- Convert Byte to Word
- 80 – FF80
- 27- 0027

- Convert Word to Double Word
- 5643 – 00005643
- 9100- FFFF9100

```
.MODEL TINY
.DATA
MULTIPLICAND    DW      2040H
MULTIPLIER      DW      2000H
PRODUCT1        DW      ?
PRODUCT2        DW      ?
.CODE
.STARTUP
                MOV     AX, MULTIPLICAND
                MUL     MULTIPLIER
                MOV     PRODUCT1, AX
                MOV     PRODUCT2,  DX
.EXIT
END
```

SPECIAL IMUL

- Available only from 80186
- IMUL Dest, Source, Immediate Data

DIV/ IDIV SOURCE

- WORD ÷ BYTE
- DWORD ÷ WORD
- QWORD ÷ DWORD

DIV/ IDIV SOURCE

- Word ÷ Byte
- Word in AX, Byte in Register (or) Memory location
- AL- quotient AH- reminder
- DWORD ÷ WORD
- DWORD in DX :AX, Word in Register (or) Memory Location
- AX- Quotient DX- Reminder
- QWORD ÷ DWORD
- QWORD in EDX : EAX, DWord in Register (or) Memory Location
- EAX- Quotient EDX- Reminder
- All Flags undefined
- Sign of remainder same as dividend – In Case of signed ops

8086-80486 –ISA & PROGRAMMING

BRANCH OPERATIONS

Jump Instructions

Conditional Jump and Unconditional Jump instructions

Unconditional jump Instructions

Near jump or intra segment jump

Far or intersegment jump

Near and Far jumps are further divided into Direct or Indirect

Direct - Destination address specified as a part of the instruction

Indirect - Destination address specified in a register or memory location

Direct near jump

- Near type fetched from anywhere in the current code segment
- Adds the displacement contained in the instruction to the contents of IP
- Signed displacement - forward or backward

Short jump

Displacement 8 –bits
+127 to -128 locations.

Near Jump

Displacement 16 bits
+32, 767 to – 32, 768 locations

0100_H	back:	add	al, 03_H
0102_H		nop	
0103_H		nop	
0104_H		jmp	back
0106_H			

- 6 is the displacement FA_H

```
0100        EB                  JMP  THER
0102        90                  NOP
0103        90                  NOP
0104                  THER:     MOV  AX, 0000H
```

Displacement = 02_H

Intra segment Indirect

Register indirect

JMP BX

Indirect Memory Addressing

JMP WORD PTR[BX]

Intersegment Direct

JMP Offset Base

Absolute branch

IP = Offset
CS = Base

INTERSEGMENT INDIRECT

Memory Addressing

JMP DWORD PTR [BX]

Conditional Jump Instructions

Always of kind SHORT

JC/JNC	➔	Carry
JZ/JNZ	➔	Zero
JP/JNP	➔	Parity
JS/JNS	➔	Sign
JO/JNO	➔	Overflow
JCXZ	➔	CX =0

8086 Conditional Jump Instructions

Unsigned numbers:

JA
JAE
JB
JBE

Comp Operands	CF	ZF
Dest > Src	0	0
Dest = Src	0	1
Dest < Src	1	0

8086 Conditional Jump Instructions

Signed numbers:

JG
JGE
JL
JLE

Comp Operands	CF	SF,OF
Dest > Src	0	SF = OF
Dest = Src	0	X
Dest < Src	1	SF ≠ OF

8086-80486 –ISA & PROGRAMMING

LOOP OPERATIONS

LOOP

Jump to a specified label if CX ≠0 after auto- dec of CX
Used to repeat a series of instructions some number of times.
The no. of times sequence is to be repeated is loaded in CX
Each time Loop inst executes CX automatically decs by 1

LOOP X1 

```
DEC  CX
JNZ  X1
```

IF CX ≠ '0' execution will jump to destination

Specified by a label in the instruction

If CX = 0 after auto dec execution will go on to the next instruction after LOOP

Destination address is of type SHORT

LOOP affects no FLAGs

Add a data in one block of memory with data in another block of memory using LOOP

Size of block -100 words

$Y_1 = X_1 + Y_1$

$Y_2 = X_2 + Y_2$

……

$Y_{n=} = X_n + Y_n$

```
.Model Tiny
.data
BLOCK 1     DW          100 DUP(?)
BLOCK 2     DW          100 DUP(?)
COUNT       DW          100
.code
.startup
            CLD
            MOV         CX, COUNT
            MOV         SI, OFFSET BLOCK1
            MOV         DI, OFFSET BLOCK 2
X1:         LODSW
            ADD         AX, [DI]
            STOSW
            LOOP        X1
.EXIT
END
```

Conditional LOOPs

LOOPE/LOOPZ (LOOP while equal)
LOOP while CX ≠ 0 and ZF = 1

Each time the LOOP instr executes - CX decremented
If CX ≠ 0 & ZF = 1 execution will jump to destn specified

If CX = 0 after auto decrement or ZF = 0 execution will go to the next inst

```
        MOV     BX, OFFSET ARRAY
        DEC     BX
        MOV     CX, 100
NEXT:   INC     BX
        CMP     [BX], OFF_H
        LOOPE   NEXT
```

LOOPNE/LOOPNZ

Loop While CX $\neq$ 0 and ZF = 0

```
        MOV     BX, OFFSET ARRAY
        DEC     BX
        MOV     CX, 100
NEXT:   INC     BX
        CMP     [BX], 0D_H
        LOOPNE  NEXT
```

8086-80486- ISA & Programming

Stack

PUSH & POP

- Store and retrieve data from LIFO stack memory
- 2/4 bytes involved
- Whenever 16-bit data pushed into stack
- MSB moves into memory [SP-1]
- LSB moves into memory [SP-2]

- Contents of SP register decremented by 2

Push

- Push data from
  - Registers/Segment Register
  - Memory
  - Flag Register

Push

- PUSH AX
- PUSH EBX
- PUSH DS
- PUSH WORD PTR[BX]
- PUSHF
- PUSHFD
- PUSHA
- PUSHAD
- PUSH 8-imm
- PUSH 16-imm
- PUSHD 32-imm

Example- PUSH operation

- PUSH AX
- PUSH BX
- PUSH SI
- PUSH WORD PTR[BX]
- PUSHF

70050	
$7004F_H$	AH
$7004E_H$	AL
$7004D_H$	BH
$7004C_H$	BL
$7004B_H$	$SI_{(High)}$
$7004A_H$	$SI_{(Low)}$
70049_H	$Mem_{(high)}$
70048_H	$Mem_{(low)}$
70047_H	$FLR_{(high)}$
70046_H	$FLR_{(low)}$

$SP \leftarrow SP-2$ $[004E_H]$

$7004E_H \leftarrow AX$

$SP \leftarrow SP-2$ $[004C_H]$

$7004C_H \leftarrow BX$

$SP \leftarrow SP-2$ $[004A_H]$

$7004A_H \leftarrow SI$

$SP \leftarrow SP-2$ $[0048_H]$

$70048_H \leftarrow MEM$

$SP \leftarrow SP-2$ $[0046_H]$

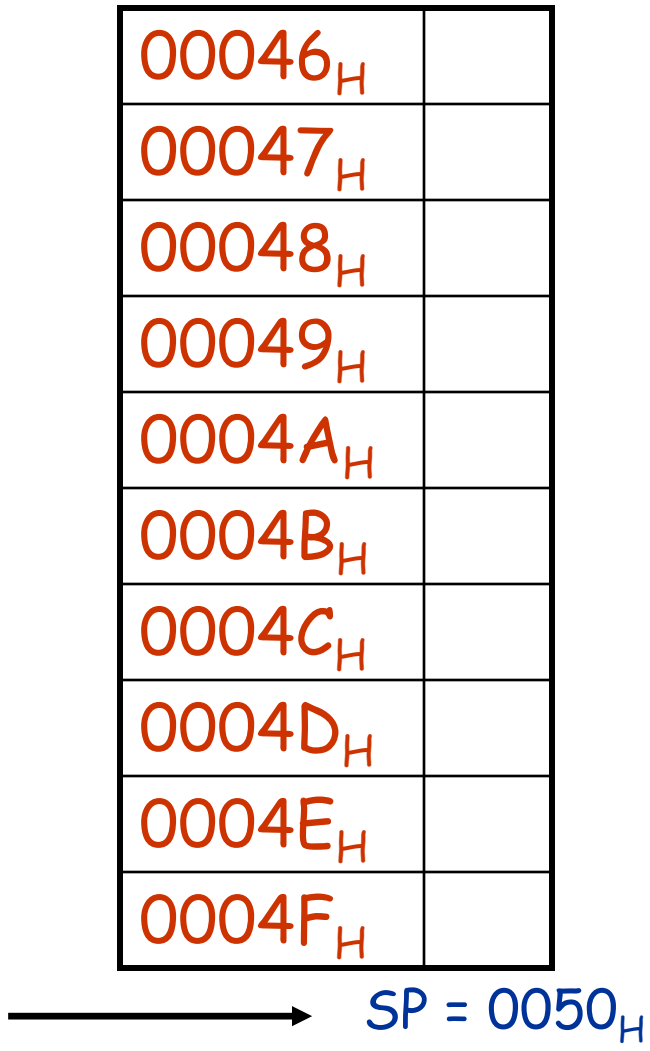
$70046_H \leftarrow FLAGS$

$SP:0050_H$ $SS:7000_H$

POP

- POP performs inverse of PUSH
- Takes data from stack to register, Memory
- Data popped in 16 bits/32 bits
- First byte from stack to lower register
- Second byte to higher register
- SP = SP+2

```
POP     AX
POP     CX
POP     WORD PTR[BX]
```


00046H
00047H
00048H
00049H
0004AH
0004BH
0004CH
0004DH
0004EH
0004FH
SP = 0050H

PUSH & POP Example

- AX – 3456_H AX- $7FDC_H$
- BX – $12AB_H$ BX – $12AB_H$
- CX – $7FDC_H$ CX – 3456_H
- SS:SP – 7000:0050

7004A	DC
7004B	7F
7004C	AB
7004D	12
7004E	56
7004F	34
70050	

- PUSH AX
- PUSH BX
- PUSH CX
- POP AX
- POP BX
- POP CX

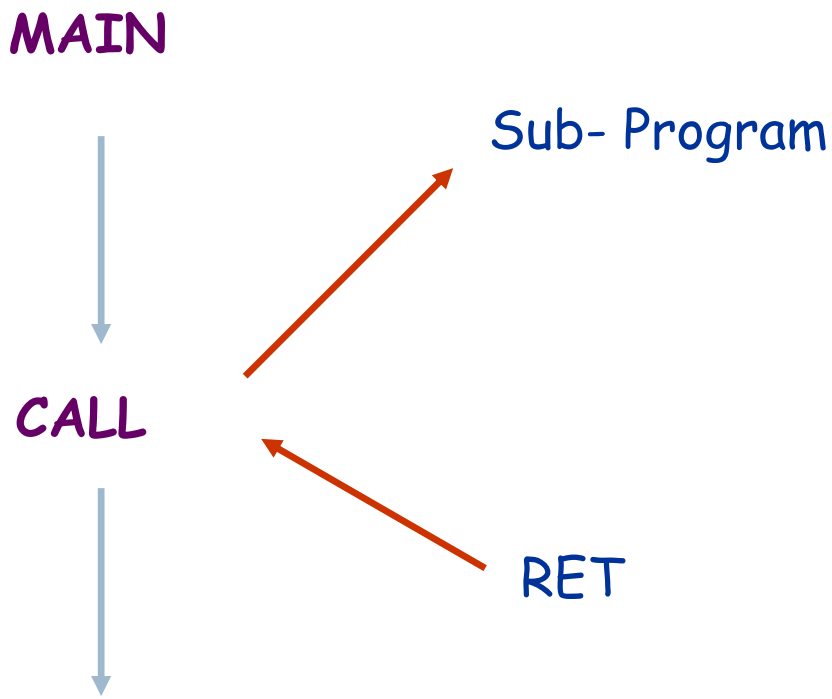
8086-80486

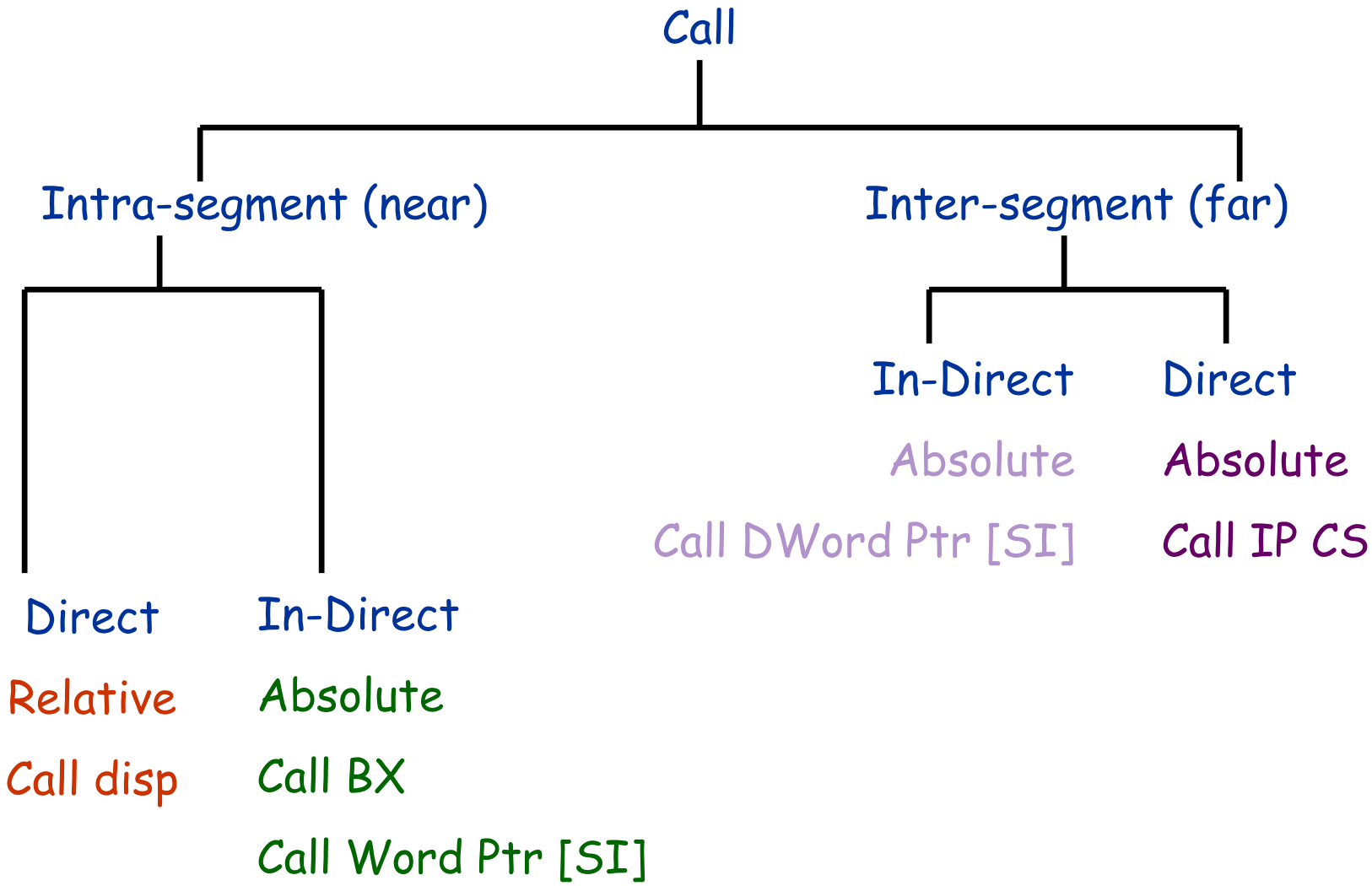
Subroutines

Subroutines

- CALL Instruction
- CALL instruction in the main line program loads the IP& in some cases also CS registers - the starting address of the procedure
- Next instruction fetched will be the first inst of the procedure

RET instruction at the procedure end sends the execution Back to mail line program




Call
Intra-segment (near)
Inter-segment (far)
Direct
Relative
Call disp
In-Direct
Absolute
Call BX
Call Word Ptr [SI]
In-Direct
Absolute
Call DWord Ptr [SI]
Direct
Absolute
Call IP CS

Subroutines – Call & RET

- CALL Stores the address of the instruction after call into stack (return address)

near CALL or far CALL

(IP saved) (CS and IP saved)

- RET inst retrieves the next address after CALL

Back to IP or (IP and CS)

DIRECT INTERSEGMENT FAR CALL

- Both CS and IP needs to be changed

  CALL IPL IPH CSL CSH

- INDIRECT INTERSEGMENT FAR CALL
  - Two words taken From Memory
  - First word - IP , Second word - CS

  CALL DWORD PTR [BX]

In Assembly language programming

Procedures (subroutines) begins with PROC directive
and ends with ENDP directive

PROC directive is followed by the type of procedure:
NEAR or FAR

CALL instruction links to the procedure
RET instruction returns from the procedure

```
        CALL        SUMS
                    _____
                    _____


SUMS        PROC    NEAR
            ADD     AX, BX
            ADD     AX, CX
            ADD     AX, DX
            RET
SUMS        ENDP
```

```
.Model Tiny
.data
dat1            db      '1', '2', '3', '4'……
res             db      100 dup(0)
stack1          dw      100 dup(?)
top_stack1      label   word
.code
.startup
                lea     sp,top_stack1
                lea     si,dat1
                lea     di,res
                mov     cx,100
x1:             lodsb
                call    mask
                loop    x1
.exit
```

```
mask        proc  near
            and   al,0fh
            stosb
            ret
mask        endp
end
```

STACK

- Call SUMS

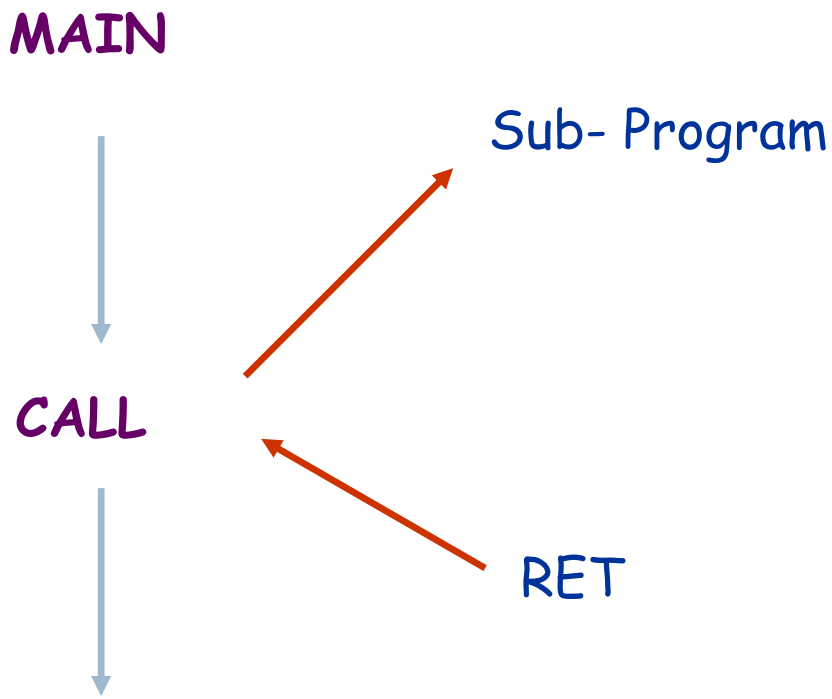
- PUSH AX
- PUSH BX
- ADD AX,BX
- RET

Will Return be to the correct address ?

8086-80486

Subroutines & Stacks

RET instruction at the procedure end sends the execution Back to mail line program



```
.Model Tiny
.data
dat1            db      '1', '2', '3', '4
res             db      4 dup(0)
stack1          dw      4 dup(?)
top_stack1      label   word
.code
.startup
                lea     sp,top_stack1
                lea     si,dat1
                lea     di,res
                mov     cx,4
x1:             lodsb
                call    mask1
                loop    x1
.exit
```

```
mask1       proc   near
            and    al,0f_h
            stosb
            ret
mask1       endp
end
```

0100	8D
0101	26
0102	2E
0103	01
0104	8D
0105	36
0106	1E
0107	01
0108	8D
0109	3E
010A	22
010B	01
010C	B9
010D	04
010E	00
010F	AC

CS,DS,SS,ES - 0B3E
IP - 0100

LEA SP, [012E]

LEA SI, [011E]

LEA DI, [0122]

MOV CX,0004

LODSB

0110	E8
0111	06
0112	00
0113	E2
0114	FA
0115	B4
0116	4C
0117	CD
0118	21
0119	24
011A	0F
011B	AA
011C	C3
011D	
011E	31
011F	32

CALL 0119

LOOP 010F

MOV AH,4C

INT 21

AND AL,0F

STOSB

RET

0122	**33**
0121	34
0122	0
0123	0
0124	0
0125	0
0126	X
0127	X
0128	X
0129	X
012A	X
012B	X
012C	X
012D	X
012E	
011F	

TOP_STACK

SP – 012E

SI – 011E

DI – 0123

CX - 0004

AL - 31

AL - 01

IP – 011F

0122	33
0121	34
0122	01
0123	0
0124	0
0125	0

0126	X
0127	X
0128	X
0129	X
012A	X
012B	X
012C	13
012D	01
012E	
011F	

8086-80486

SUBROUTINES- PARAMETER PASSING

PASSING PARAMETERS TO PROCEDURES

- Parameters are data values or addresses made available for the procedures for use
- Passed by
  - Registers
  - Dedicated memory locations accessed by name
  - With pointers passed in registers
  - With stack

EX: BCD to Binary Conversion

```
BCD_BIN     PROC    NEAR
            PUSHF
            PUSH    BX
            PUSH    CX
            MOV     BL , AL
            AND     BL , 0F_H
            AND     AL , 0F0_H
            MOV     CL , 04
            ROR     AL, CL
            MOV     BH , 0A_H
            MUL     BH
            ADD     AL, BL
            POP     CX
            POP     BX
            POPF
            RET
BCD_BIN     ENDP
```

BCD – 16

Bin/hex -10_h

(01 * $0a_h$) + 6

```
MOV   AL,  BCDINPUT
CALL  BCD_BIN
```

Using reg

```
BCD_BIN     PROC  NEAR
            MOV   AL, BCDINPUT
```

Using Mem

```
            MOV   SI,  OFFSET  BCDINPUT
            CALL   BCD_BIN

BCD_BIN     PROC  NEAR
            MOV   AL, [SI]
```

Using Pointers

```
        MOV   AX,  30
        PUSH  AX
        CALL  BCD_BIN

BCD_BIN PROC  NEAR
        PUSH  BP
        MOV   BP, SP
        MOV   AX, [BP+4]
        …..
        POP   BP
        RET
```

Using stack