

Chapter 17

Bit-Level Operations

Objectives

Upon completion of this chapter you will be able to:

- Describe what is meant by the *or*, *and*, and *exclusive or* operations,
- Use the `OI` and `OC` instructions to turn on bits,
- Use the `NI` and `NC` instructions to turn off bits,
- Use the `XI` and `XC` instructions to toggle bits,
- Use the `TM` instruction to test bits,
- Use the `OI` and `NI` instructions to change the case of a letter,
- Use the `XI` and `XC` instructions for data encryption,
- Use the `XC` instruction to swap fields,
- Use the `SLL` and `SRL` instructions to shift bits in a register,
- Use the `SLL` and `SRL` instructions to multiply a register by a power of two,
- Use PC/370's `SVC 18` to access the system date and time.

Introduction

In this chapter we will look at some of the System/370's bit level operations. Most of these are fairly specialized: they aren't needed very often, but when you *do* need them there is simply no getting by without them. In particular, we will look at the `OI`, `NI`, `XI`, `OC`, `NC`, `XC`, `TM`, `SRL`, and `SLL` instructions.

The Or, And, and Exclusive Or Operations

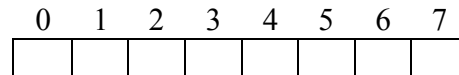
By now we know that each byte consists of eight bits, or binary digits. There are three bit-level operations. These operations are known as *Or*, *And*, and *Exclusive Or*. Each of these operations compares corresponding bits from each of the two operands. Any bit in the target operand may be changed as a result of the comparison. The result will depend on the following truth table:

	<u>OR</u> The result is 1 if either bit is 1	<u>AND</u> The result is 1 if both bits are 1	<u>EXCLUSIVE OR</u> The result is 1 if exactly one bit is 1												
Operand 1	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1
0	0	1	1												
0	0	1	1												
0	0	1	1												
Operand 2	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	1	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	1	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	1
0	1	0	1												
0	1	0	1												
0	1	0	1												
Result	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	0	1	1	1	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	1	0
0	1	1	1												
0	0	0	1												
0	1	1	0												

The corresponding Storage-and-Immediate (SI) instructions are `OI` (or immediate), `NI` (and immediate) and `XI` (exclusive or immediate).

The OI, NI, and XI Instructions

In the following discussion we will refer to bit positions by number: the standard is to number bits from left to right, beginning with zero.



Example #1: Turn on the left-most bit in the first byte of FLD.

```
OI  FLD,X'80'    or    OI  FLD,B'10000000'
```

Example #2: Turn on bit one of FLD. All other bits remain unchanged.

```
OI  FLD,X'40'    or    OI  FLD,B'01000000'
```

But note... X'99' = B'10011001' = C'r'
and... X'D9' = B'11011001' = C'R'

So Example #2 illustrates how we can change a lower case letter to upper case!

Example #3: Turn on bits zero, one, two, and three of FLD.

```
OI  FLD,X'F0'    or    OI  FLD,B'11110000'
```

So Example #3 illustrates one method by which we can remove the sign from a number following an UNPK.

You Try It...

1. Write the instruction to turn on the right-most bit in the first byte of FLD.
2. Write the instruction to turn on bits two and three of the last byte of X. (Use the length operator to point to the last byte of X.)
3. Example #2 shows how we can change a lower case letter to upper case. What if the byte in question contains a number; that is, X'F0' through X'F9'? What effect, if any, will the OI instruction as shown have on that byte?
4. Example #3 shows how we can remove the sign from a number following an UNPK. What instruction did we use before to do this? What is the length of these instructions? Why might this method (OI) be preferred over the other?

Example #4: Turn off the leftmost and rightmost bits of FLD.

```
NI  FLD,X'7E'    or    NI  FLD,B'01111110'
```

Example #5: Turn off bit four of `FLD`.

```
NI   FLD,X'F7'      or   NI   FLD,B'11110111'
```

But note... `X'FF'` = `B'11111111'` = 255
 and... `X'08'` = `B'00001000'` = 8
 and... `X'F7'` = `B'11110111'` = 247

So we could also use:

```
NI   FLD,255-8      or
NI   FLD,X'FF'-X'08' or
NI   FLD,ALLBITS-BIT4
```

where `ALLBITS EQU X'FF'`
 and `BIT4 EQU X'08'`

Example #6: Turn off bit one of `FLD`. All other bits remain unchanged. (This is the reverse of Example #2 above.)

```
NI   FLD,B'10111111' or
NI   FLD,X'BF'      or
NI   FLD,ALLBITS-X'40'
```

So Example #6 illustrates how we can change an upper case letter to lower case!

Example #7: Turn on bit seven of the rightmost byte of `FLDB`, a three-byte packed field, if it is off, otherwise turn it off.

```
XI   FLDB+2,X'01'
```

Changing the value of a field in this way (that is, turning it on if off, or turning it off if on) is sometimes referred to as **toggle**. But note...

```
and... X'0C' = B'0000110 0'
        X'0D' = B'0000110 1'
                        ↑
```

Recall that `C` represents a positive sign on a packed number, and `D` represents a negative sign on a packed number, and we see that if we `XI` the last byte with `X'01'`, we "toggle" between `C` and `D`.

So Example #7 illustrates how we can change the sign of a packed number!

You Try It...

5. Write the instruction to turn off the right-most bit in the first byte of `FLD`.
6. Write the instruction to turn off bits two and three of the last byte of `x`. (Use the length operator to point to the last byte of `x`.)
7. Example #6 shows how we can change an upper case letter to lower case. What if the byte in question contains a number; that is, `x'F0'` through `x'F9'`? What effect, if any, will the `NI` instruction as shown have on that byte?
8. Example #7 shows how we can change the sign of a packed number. We could accomplish the same thing by multiplying the number by -1. Given `PK3` contains `x'00012D'`, use both methods to change the sign.
9. Refer to the previous question. Why might this method (`XI`) be preferred over the other (`MP`)? Hint: What if we want to change the sign of `PK3` which contains `x'12345D'`?

* * * * *

The `XI` instruction has a curious property in that, if a field is `XI`'ed with a value, and the resulting field is `XI`'ed with the same value, the field returns to its original value. This property is useful in encryption programs as demonstrated in the next example.

Example #8: `XI` the letter 'R' with the character '+':

```
C'R' = X'D9' = B'11011001'
C+' = X'4E' = B'01001110'
result = B'10010111' = X'97' = C'p'
```

`XI` the result ('p') with the character '+' again:

```
C'p' = X'97' = B'10010111'
C+' = X'4E' = B'01001110'
result = B'11011001' = X'D9' = C'R'
```

We see the result ('R') is the original value. This property is useful in encryption programs!

You Try It...

10. `XI` the letter 'H' with the character '\$'. `XI` the result with the character '\$' again. Show the intermediate results.
11. `XI` the letter 'S' with the character '#'. `XI` the result with the character '#' again. Show the intermediate results.

The OC, NC, and XC Instructions

There is a corresponding SS (Storage-to-Storage) instruction for each of the above SI instructions: they are OC, NC, and XC. The function is the same as with the previous instructions, but in each case the second parameter is a field (or literal) rather than an immediate value.

Example #9: XC the letters 'PR' with the characters '+;'. (This can be thought of as encryption.)

```
C'PR' = X'D7D9' = B'1101011111011001'
C'+;' = X'4E5E' = B'0100111001011110'
      result = B'1001100110000111'
           = X'9987' = C'rg'
```

XC the result ('rg') with the characters '+;' again. (This can be thought of as decryption.)

```
C'rg' = X'9987' = B'1001100110000111'
C'+;' = X'4E5E' = B'0100111001011110'
      result = B'1101011111011001'
           = X'D7D9' = C'PR'
```

We see the result ('PR') is the original value!

Example #10: If FLDA DS CL1 has value X'10110110'
and: FLDB DS CL1 has value X'11010010'

```
XC FLDA,FLDB gives FLDA = X'01100100'
XC FLDB,FLDA (using the new FLDA)
              gives FLDB = X'10110110'
XC FLDA,FLDB (using the new FLDB)
              gives FLDA = X'11010010'
```

But note: FLDA is now equal to the "original" FLDB and FLDB is now equal to the "original" FLDA!

So Example #10 illustrates how we can use the XC instruction to "swap" the values in two fields. (The two fields must be of equal size.)

You Try It...

- Given WK2 DC CL2'HS'. XC the field WK2 with the characters '\$#' twice. Show the result after each XC.

13. Given A DC CL2 'PJ' and B DC CL2 'B4'. Use the XC instruction to swap A and B. Show all intermediate results.
14. Given X DC CL2 'R2'. Determine the results of XC X, X. What can you conclude?

Manipulating Registers: The SLL and SRL Instructions

The SLL (Shift Left Logical) and SRL (Shift Right Logical) instructions are similar to the SRP (Shift and Round Packed) instruction, except that whereas that instruction shifted the digits of a packed number to the left or right, these instructions shift the bits in a register to the left or right. Recall that the effect of the SRP was to multiply or divide the packed number by some power of ten. Likewise, the result of the SLL is to multiply the value of a register by some power of two, and the result of the SRL is to divide the value of a register by some power of two. For example:

Example #11:

LA R3, 4	R3=	<table style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0100</td> </tr> <tr> <td>00</td> <td>00</td> <td>00</td> <td>04</td> </tr> </table>	0000 0000	0000 0000	0000 0000	0000 0100	00	00	00	04	4
0000 0000	0000 0000	0000 0000	0000 0100								
00	00	00	04								
SLL R3, 3		<table style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0010 0000</td> </tr> <tr> <td>00</td> <td>00</td> <td>00</td> <td>20</td> </tr> </table>	0000 0000	0000 0000	0000 0000	0010 0000	00	00	00	20	32
0000 0000	0000 0000	0000 0000	0010 0000								
00	00	00	20								
SRL R3, 2		<table style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 0000</td> <td style="border: 1px solid black; width: 25%;">0000 1000</td> </tr> <tr> <td>00</td> <td>00</td> <td>00</td> <td>08</td> </tr> </table>	0000 0000	0000 0000	0000 0000	0000 1000	00	00	00	08	8
0000 0000	0000 0000	0000 0000	0000 1000								
00	00	00	08								

The SLL instruction above shifts all bits in register 3 to the left three positions. The net effect is to multiply that register by 2^3 , or 8, giving $4 * 8 = 32$. The SRL instruction shifts all bits in register 3 to the right 2 positions. The net effect is to multiply that register by 2^{-2} , or divide by 2^2 , or 4, giving $32 / 4 = 8$.

You Try It...

15. Execute the following instructions. Show all intermediate results.
 - LH R4, =H' 48'
 - SRL R4, 4
 - SLL R4, 2
16. Execute the following instructions. Show all intermediate results.
 - LH R4, =H' 20'
 - SRL R4, 3
 - SLL R4, 3

Sample Program: Bit-Level Operations

The following program, BITOPS.MLC, will demonstrate most of the examples discussed above. The output from the execution of the program follows the source code listing.

```

PRINT NOGEN
*****
*      FILENAME:  BITOPS.MLC      *
*      AUTHOR   :  Bill Qualls   *
*      SYSTEM   :  PC/370 R4.2   *
*      REMARKS  :  Demonstrate bit-level operations. *
*****
      START 0
      REGS
BEGIN  BEGIN
*
      WTO      'EXAMPLE #2 - Demonstrate use of OI to change'
      WTO      'lower case letter to upper case'
      WTO      LOWER
      OI       LOWER,X'40'
      WTO      LOWER
*
      WTO      'EXAMPLE #3 - Demonstrate use of OI to remove'
      WTO      'the sign from a number following an UNPK'
      UNPK    UNPACKED,=P'-12345'
      WTO      UNPACKED
      OI      UNPACKED+L'UNPACKED-1,X'F0'
      WTO      UNPACKED
*
      WTO      'EXAMPLE #6 - Demonstrate use of NI to change'
      WTO      'upper case letter to lower case'
      WTO      UPPER
      NI      UPPER,ALLBITS-X'40'
      WTO      UPPER
*
      WTO      'EXAMPLE #7 - Demonstrate use of XI to ''toggle''
      WTO      'the sign of a packed number'
      MVC     EDITED,MASK
      ED      EDITED,POSITIVE
      WTO     EDITED
      XI     POSITIVE+L'POSITIVE-1,X'01'
      MVC     EDITED,MASK
      ED      EDITED,POSITIVE
      WTO     EDITED
      XI     POSITIVE+L'POSITIVE-1,X'01'
      MVC     EDITED,MASK
      ED      EDITED,POSITIVE
      WTO     EDITED
*
      WTO      'EXAMPLE #8 - Demonstrate use of XI for'
      WTO      'encryption: once to encrypt, once to decrypt.'
      WTO      CRYPT1
      XI      CRYPT1,C'+ '    encrypt
      WTO      CRYPT1
      XI      CRYPT1,C'+ '    decrypt
      WTO      CRYPT1
*
      WTO      'EXAMPLE #9 - Demonstrate use of XC for'
      WTO      'encryption: once to encrypt, once to decrypt.'
      WTO      CRYPT2
      XC      CRYPT2,=C'+;'    encrypt
      WTO      CRYPT2
      XC      CRYPT2,=C'+;'    decrypt
      WTO      CRYPT2
*

```

(continued)

```
WTO 'EXAMPLE #10 - Demonstrate use of XC to swap'
WTO 'two values'
WTO BOTH
XC FLDA,FLDB
XC FLDB,FLDA
XC FLDA,FLDB
WTO BOTH
*
WTO 'EXAMPLE #11 - Demonstrate that SLL is same as'
WTO 'multiplying a register by a power of two, and'
WTO 'that SLR is same as dividing by a power of two.'
LA R3,4          We begin with 4
CVD R3,DBLWORD
MVC EDITED,MASK
ED EDITED,DBLWORD+5
WTO EDITED
SLL R3,3          Multiply 4 by 2^3, or 8, giving 32
CVD R3,DBLWORD
MVC EDITED,MASK
ED EDITED,DBLWORD+5
WTO EDITED
SRL R3,2          Divide 32 by 2^2, or 4, giving 8
CVD R3,DBLWORD
MVC EDITED,MASK
ED EDITED,DBLWORD+5
WTO EDITED
*
RETURN
*
LTORG
*
DBLWORD DC D'0'
MASK DC XL7'40202020212060'
EDITED DC CL7' '
POSITIVE DC PL3'+6789'
UNPACKED DC CL5' '
LOWER DC CL1'r' Lower case letter 'r'
UPPER DC CL1'T' Upper case letter 'T'
ALLBITS EQU X'FF'
CRYPT1 DC CL1'R'
CRYPT2 DC CL2'PR'
BOTH DS 0CL9
FLDA DC CL3'123'
DC CL3' '
FLDB DC CL3'AbC'
END
```

```
A:\MIN>bitops
EXAMPLE #2 - Demonstrate use of OI to change
lower case letter to upper case
r
R
EXAMPLE #3 - Demonstrate use of OI to remove
the sign from a number following an UNPK
1234N
12345
EXAMPLE #6 - Demonstrate use of NI to change
upper case letter to lower case
T
t
EXAMPLE #7 - Demonstrate use of XI to 'toggle'
the sign of a packed number
6789
6789-
6789
EXAMPLE #8 - Demonstrate use of XI for
encryption: once to encrypt, once to decrypt.
R
P
EXAMPLE #9 - Demonstrate use of XC for
encryption: once to encrypt, once to decrypt.
PR
rg
PR
EXAMPLE #10 - Demonstrate use of XC to swap
two values
123   AbC
AbC   123
EXAMPLE #11 - Demonstrate that SLL is same as
multiplying a register by a power of two, and
that SLR is same as dividing by a power of two.
4
32
8
```

You Try It...

17. Write a similar program to demonstrate your answers to all previous You Try It exercises.

Sample Program: Accessing the System Date and Time

The next program, `DATE370.MLC`, uses several of these instructions to retrieve the system date and time. This program makes use of supervisor call 18, which returns time in register 0, the year (with century) in register 1, and the day, month, and day of week indicator in register 2. These registers are then manipulated so as to return the date and time in a standard form. Meaningful comments have been used throughout. Of particular interest is the means by which the `SLL` and `SRL` instructions are used together to isolate a portion of a register. Note: `SVC 18` is discussed in PC/370's documentation. The use of `SVC 18` to obtain the system date and time is unique to PC/370.

```

PRINT NOGEN
*****
*      FILENAME:  DATE370.MLC      *
*      AUTHOR   :  Bill Qualls    *
*      SYSTEM   :  PC/370 R4.2    *
*      REMARKS  :  Demonstrate date/time functions in PC/370. *
*****
START 0
REGS
BEGIN
WTO  MESSAGE          (Before)
*
SVC  18
*
*      Supervisor call 18 returns
*      time in R0; year with century
*      in R1; day, month, and day of
*      week in R2.
*
LR   R3,R0            Put time in R3
SRL  R3,24            hhmssxx becomes 000000hh
CVD  R3,DBL           Hours only
UNPK TIME(2),DBL     Move to output
OI   TIME+1,X'F0'    Remove sign
*
LR   R3,R0            Put time in R3
SLL  R3,8             hhmssxx becomes mmssxx00
SRL  R3,24            mmssxx00 becomes 000000mm
CVD  R3,DBL           Minutes only
UNPK TIME+3(2),DBL  Move to output
OI   TIME+4,X'F0'    Remove sign
*
LR   R3,R0            Put time in R3
SLL  R3,16            hhmssxx becomes ssxx0000
SRL  R3,24            ssxx0000 becomes 000000ss
CVD  R3,DBL           Seconds only
UNPK TIME+6(2),DBL Move to output
OI   TIME+7,X'F0'    Remove sign
*
LR   R3,R0            Put time in R3
SLL  R3,24            hhmssxx becomes xx000000
SRL  R3,24            xx000000 becomes 000000xx
CVD  R3,DBL           Hundredths of seconds only
UNPK TIME+9(2),DBL Move to output
OI   TIME+10,X'F0'   Remove sign
*
CVD  R1,DBL           Year with century
UNPK DATE+6(4),DBL  Move to output
OI   DATE+9,X'F0'   Remove sign
*
LR   R3,R2            Put date in R3
SRL  R3,24            mmdww00 becomes 000000mm
CVD  R3,DBL           Month only
UNPK DATE(2),DBL    Move to output
OI   DATE+1,X'F0'   Remove sign
*

```

(continued)

```

LR      R3,R2          Put date in R3
SLL     R3,8           mmdww00 becomes ddww0000
SRL     R3,24          ddww0000 becomes 000000dd
CVD     R3,DBL         Day of month only
UNPK    DATE+3(2),DBL Move to output
OI      DATE+4,X'F0'   Remove sign
*
LR      R3,R2          Put date in R3
SLL     R3,16          mmdww00 becomes ww000000
SRL     R3,24          ww000000 becomes 000000ww
MH      R3,=H'3'       Each day of week is 3 long
A       R3,=A(DOWTBL) Displacement into table
MVC     DOW,0(R3)      Move to output
*
WTO     MESSAGE        (After)
*
RETURN
*
LTORG
*
MESSAGE DS 0CL71
DC      CL18'DATE370...Time is '
TIME    DC CL11'hh:mm:ss.xx'
DC      CL11'...Date is '
DATE    DC CL10'mm/dd/yyyy'
DC      CL18'...Day of week is '
DOW     DC CL3'ddd'
*
DBL     DS D
DOWTBL  DC C'SunMonTueWedThuFriSat'
*
END

```

```

A:\MIN>date370
DATE370...Time is hh:mm:ss.xx...Date is mm/dd/yyyy...Day of week is ddd
DATE370...Time is 08:49:54.44...Date is 01/06/1994...Day of week is Thu

```

* * * * *

The ability to turn on or turn off selected bits means we can use bits as switches. In particular, any binary condition (a condition with only two possible states) can be represented with a single bit rather than an entire byte. This can result in a substantial savings of disk space and telecommunications time and cost. Some examples of binary conditions are:

CONDITION	OFTEN REPRESENTED AS	CAN ALSO BE REPRESENTED AS
GENDER	'F' = Female 'M' = Male	0 = Female 1 = Male
TENURED	'N' = No 'Y' = Yes	0 = No 1 = Yes
CHECKING ACCOUNT TRANSACTION TYPE	'C' = Check 'D' = Deposit	0 = Check 1 = Deposit
OUT OF STOCK	' ' = No 'X' = Yes	0 = No 1 = Yes
MARKED FOR DELETION (as used in dBASE III+)	' ' = No '*' = Yes	0 = No 1 = Yes

Checking Bits: The `TM` Instructions

As we've already seen, we can use the `OI`, `NI`, and `XI` instructions to turn on or turn off bits. Of course, it doesn't do us any good to use a bit as a switch if we cannot also test the value of that bit. The `TM` (Test under Mask instruction) is used to do so. The `TM` instruction is an SI-type instruction and has the form `TM field,mask`

It is immediately followed by a `BC` (branch on condition), typically using one of the following extended mnemonics:

MNEMONIC	MEANING	BC EQUIVALENT
BO	Branch if Ones	BC 1, label
BM	Branch if Mixed	BC 4, label
BZ	Branch if Zeros	BC 8, label
BNO	Branch if Not Ones	BC 14, label
BNM	Branch if Not Mixed	BC 11, label
BNZ	Branch if Not Zeros	BC 7, label

Example #12: If the first and third bits of `FLDA` are on, then turn off the third bit. Otherwise, turn on the seventh bit.

```

                TM    FLDA, B'10100000'
                BO    OFF3RD
                OI    FLDA, B'00000010'
                B     DONE
OFF3RD EQU     *
                NI    FLDA, 255-B'00100000'
DONE EQU     *
```

Example #13: If the fifth, seventh or eighth bits of `FLDB` are on, then turn on the first bit.

```

                TM    FLDB, B'00001011'
                BZ    ALLOFF
                OI    FLDB, B'10000000'
ALLOFF EQU     *
```

There is no SS equivalent to the `TM` instruction: you can only test one byte at a time, and you can use an immediate value only. Of course, as with all SI instructions, you can use equated values. For example, to test for gender equal male, one might code:

```

                TM    INFO, MALE           where
                INFO  DS    CL1
                MALE  EQU   X'80'         First bit indicates gender
                CITIZEN EQU X'40'         Second bit indicates citizenship
                *                                           Other bits unused at this time
```

You Try It...

18. If the last bit of the first byte of `A` is on, and the first bit of the last byte of `B` is off, then turn on the first bit of the first byte of `C`.
19. Given `INFO`, `MALE`, and `CITIZEN` as defined above, and `SWITCH DC CL1 ' '`. If `INFO` indicates a female citizen, move 'Y' to `SWITCH`. Otherwise, move 'N' to `SWITCH`.

Exercises

1. True or false. Given $A \text{ DC CL2'IQ'}$ and $B \text{ DC CL2'5H'}$...
 - T F a. To turn on the last bit in the last byte of A , leaving all other bits unchanged, we code $\text{OI } A+1, X'08'$
 - T F b. To turn off the first bit in the last byte of A , leaving all other bits unchanged, we code $\text{NI } A+1, X'80'$
 - T F c. To turn off the leftmost bit in the first byte of A if it is on, and to turn on that bit if it is off, we code $\text{XI } A, X'80'$
 - T F d. To change the 'Q' in A to lower case, we code $\text{NI } A+1, B'10111111'$
 - T F e. To swap A and B we code $\text{XC } A, B$ three times.
 - T F f. The value in B may have been a result of $\text{UNPK } B, PK3$ where $PK3$ is a packed number containing $+158$.
 - T F g. $\text{OI } B+L'B-1, X'F0'$ will give B equal to CL2'58'
 - T F h. Given $\text{TM } A, X'C0'$ and BZ SKIP the branch will be taken.
 - T F i. Given $\text{TM } B+1, B'10000000'$ and BO SKIP the branch will be taken.
 - T F j. $\text{OC } A, B$ gives A equal $X'3C10'$.
 - T F k. The SRL instruction is used to multiply a register by a power of 10.
 - T F l. Given register 4 contains 10. After $\text{SLL } R4, 3$ followed by $\text{SRL } R4, 3$ register 4 still contains 10.
 - T F m. Given register 4 contains 10. After $\text{SRL } R4, 3$ followed by $\text{SLL } R4, 3$ register 4 still contains 10.

2. Complete the following tables:

	OR	AND	EXCLUSIVE OR
Operand 1	0 1 0 1	0 1 0 1	0 1 0 1
Operand 2	0 0 1 1	0 0 1 1	0 0 1 1
Result			

3. Supply the bits for MASK and the resulting bits for FLD :

a.	OI FLD, X'FC'	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">FLD</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> </tr> <tr> <td>MASK</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td>FLD</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> </table>	FLD	1	1	0	0	0	1	1	0	MASK									FLD								
FLD	1	1	0	0	0	1	1	0																					
MASK																													
FLD																													
b.	NI FLD, X'E4'	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">FLD</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> </tr> <tr> <td>MASK</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td>FLD</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> </table>	FLD	1	1	0	0	0	1	1	0	MASK									FLD								
FLD	1	1	0	0	0	1	1	0																					
MASK																													
FLD																													
c.	XI FLD, X'7A'	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">FLD</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">0</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">1</td> <td style="border: 1px solid black; text-align: center;">0</td> </tr> <tr> <td>MASK</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td>FLD</td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> <td style="border: 1px solid black; text-align: center;"> </td> </tr> </table>	FLD	1	1	0	0	0	1	1	0	MASK									FLD								
FLD	1	1	0	0	0	1	1	0																					
MASK																													
FLD																													

Exercises

d.	OI	FLD, X'B3'	FLD	1	1	0	0	0	1	1	0
			MASK								
			FLD								

e.	NI	FLD, X'A6'	FLD	1	1	0	0	0	1	1	0
			MASK								
			FLD								

f.	XI	FLD, X'AA'	FLD	1	1	0	0	0	1	1	0
			MASK								
			FLD								

g.	OI	FLD, X'0F'	FLD	1	1	0	0	0	1	1	0
			MASK								
			FLD								

4. Given the specified values for FLD (before), MASK, and FLD (after), supply the missing instruction.

a.	FLD	1	1	0	0	0	1	1	0
	MASK	1	1	1	0	0	1	0	0
	FLD	1	1	0	0	0	1	0	0

b.	FLD	1	0	0	0	1	1	0	1
	MASK	1	1	1	1	1	0	0	1
	FLD	1	1	1	1	1	1	0	1

c.	FLD	1	1	0	1	0	0	0	1
	MASK	0	0	0	0	1	1	1	1
	FLD	1	1	0	1	1	1	1	0

d.	FLD	1	1	0	1	0	1	1	1
	MASK	1	1	0	1	0	0	1	0
	FLD	1	1	0	1	0	1	1	1

e.	FLD	1	1	0	1	0	1	1	1
	MASK	0	0	1	1	0	1	0	0
	FLD	0	0	0	1	0	1	0	0

f.	FLD	1	1	0	1	0	1	1	1
	MASK	0	0	1	1	0	1	0	1
	FLD	1	1	1	0	0	0	1	0

g.	FLD	0	1	1	0	1	1	0	0
	MASK	1	0	1	0	0	1	1	1
	FLD	0	0	1	0	0	1	0	0

Exercises

5. Write the BAL code for each of the following:
 - a. If the first, second, and third bits of `FLDA` are not all on, then turn on the sixth and seventh bits.
 - b. If the fourth or fifth bit of `FLDB` is off, then turn on the first bit, otherwise turn off the last bit.
6. Based on our discussion of the use of the `XC` instruction, write a program which will encrypt a file. Write another (similar) program which will decrypt a file. You can either hard-code the "key" in the program, or read it from a file.

Note: Examples 8 and 9 of this chapter were carefully chosen to give output with printable characters. It is unlikely that your program will do so for all characters. Similarly, it is unlikely that all characters resulting from your encryption routine (which works in EBCDIC) will have a corresponding character in the ASCII character set. Therefore, make sure your encryption routine writes an EBCDIC file, and that your decryption program reads an EBCDIC file. To do so, simply omit the `OI` instruction used before the `OPEN` macro. The use of EBCDIC on the PC gives you an added level of encryption anyway! (This EBCDIC vs. ASCII consideration was mentioned in chapter 14.)

7. Use `DATE370.MLC` in this chapter to write a copy routine which returns the system date in 'mm/dd/yy' format (no century). Call your routine `DATE370.CPY`. Modify one of your existing report programs to use this routine to obtain the system date, and print that date in the headings. (`COPY` was first discussed in chapter 13.)
8. Use `DATE370.MLC` in this chapter to write a copy routine which returns the system time in 'hh:mm:ss' format (no hundredths). Call your routine `TIME370.CPY`. Modify one of your existing report programs to use this routine to obtain the system time, and print that time in the headings. (`COPY` was first discussed in chapter 13.)