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:

<table>
<thead>
<tr>
<th>OR</th>
<th>AND</th>
<th>EXCLUSIVE OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>The result is 1 if either bit is 1</td>
<td>The result is 1 if both bits are 1</td>
<td>The result is 1 if exactly one bit is 1</td>
</tr>
</tbody>
</table>

Operand 1

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Result

| 0 | 1 | 1 | 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.

\[
\text{OI FLD,X'80' or OI FLD,B'10000000'}
\]

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

\[
\text{OI FLD,X'40' or OI FLD,B'01000000'
}
\]

But note...

\[
\text{X'99' = B'10011001' = C'r'}
\]

and...

\[
\text{X'D9' = B'11011001' = C'R'}
\]

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.

\[
\text{OI FLD,X'F0' or OI FLD,B'11110000'
}
\]

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.

\[
\text{NI FLD,X'7E' or NI FLD,B'01111110'
}\]
Example #5: Turn off bit four of FLD.

\[
\text{NI FLD,X}'F7' \quad \text{or} \quad \text{NI FLD,B}'11110111'\\
\]

But note...
\[\text{X}'FF' = B'11111111' = 255\]
and...
\[\text{X}'08' = B'00001000' = 8\]
and...
\[\text{X}'F7' = B'11110111' = 247\]

So we could also use:

\[
\begin{align*}
\text{NI FLD,255-8} & \quad \text{or} \\
\text{NI FLD,X}'FF'-X'08' & \quad \text{or} \\
\text{NI FLD,ALLBITS-BIT4} &
\end{align*}
\]

where
\[
\begin{align*}
\text{ALLBITS} & \quad \text{EQU} \quad \text{X}'FF' \\
\text{BIT4} & \quad \text{EQU} \quad \text{X}'08'
\end{align*}
\]

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

\[
\begin{align*}
\text{NI FLD,B}'10111111' & \quad \text{or} \\
\text{NI FLD,X}'BF' & \quad \text{or} \\
\text{NI FLD,ALLBITS-X}'40'
\end{align*}
\]

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.

\[
\text{XI FLDB+2,X}'01'
\]

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 toggling. But note...

\[
\begin{align*}
\text{X}'0C' & = B'00001100' \\
\text{X}'0D' & = B'00001101'
\end{align*}
\]

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'0F0'` through `X'0F9'`? 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 '+':

\[
\begin{align*}
\text{C'}R' &= \text{X'}D9' = B'11011001' \\
\text{C'}+ &= \text{X'}4E' = B'01001110' \\
\text{result} &= B'10010111' = X'97' = \text{C'}p'
\end{align*}
\]

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

\[
\begin{align*}
\text{C'}p' &= \text{X'}97' = B'10010111' \\
\text{C'}+ &= \text{X'}4E' = B'01001110' \\
\text{result} &= B'11011001' = X'D9' = \text{C'}R'
\end{align*}
\]

*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.)

\[
\begin{align*}
C'PR' &= X'D7D9' = B'1101011111011001' \\
C'+;' &= X'4E5E' = B'0100111001011110' \\
\text{result} &= B'1001100110000111' \\
&= X'9987' = C'rg'
\end{align*}
\]

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

\[
\begin{align*}
C'rg' &= X'9987' = B'1001100110000111' \\
C'+;' &= X'4E5E' = B'0100111001011110' \\
\text{result} &= B'1101011111011001' \\
&= X'D7D9' = C'PR'
\end{align*}
\]

*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...**

12. Given **WK2 DC CL2 'HS'**. **XC** the field **WK2** with the characters '§#' twice. Show the result after each **XC**.
CHAPTER 17
BIT LEVEL OPERATIONS

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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R3,4</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA R3,4</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>4</td>
</tr>
<tr>
<td>SLL R3,3</td>
<td>0000 0000 0000 0000 0000 0000 0010 0000</td>
<td>32</td>
</tr>
<tr>
<td>SRL R3,2</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>8</td>
</tr>
</tbody>
</table>

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 \( 4 \times 8 = 32 \). 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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R4,4</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LH R4,=H'48'</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>4</td>
</tr>
<tr>
<td>SRL R4,4</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>32</td>
</tr>
<tr>
<td>SLL R4,2</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>8</td>
</tr>
</tbody>
</table>

16. Execute the following instructions. Show all intermediate results.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R4,2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LH R4,=H'20'</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>4</td>
</tr>
<tr>
<td>SRL R4,3</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>32</td>
</tr>
<tr>
<td>SLL R4,3</td>
<td>0000 0000 0000 0000 0000 0000 0000 0100</td>
<td>8</td>
</tr>
</tbody>
</table>

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

WTO 'EXAMPLE #2 - Demonstrate use of OI to change'
WTO 'lower case letter to upper case'
OI LOWER
WTO LOWER

WTO 'EXAMPLE #3 - Demonstrate use of OI to remove'
WTO 'the sign from a number following an UNPK'
UNPK UNPACKED,-P'-12345'
OI UNPACKED
WTO UNPACKED

WTO 'EXAMPLE #6 - Demonstrate use of NI to change'
WTO 'upper case letter to lower case'
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)
```
EXAMPLE #10 - Demonstrate use of XC to swap

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

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.'

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 '0C19
FLDA DC 'CL3'123'
DC 'CL3' '
FLDB DC 'CL3'AbC'
END
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-
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.
PP
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 SRL is same as dividing by a power of two.
\[ 4 \times 32 = 8 \]

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.
**CHAPTER 17
BIT LEVEL OPERATIONS**

---

```
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 hhmmssxx 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 hhmmssxx 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 hhmmssxx 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 hhmmssxx 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 mmddwxxx becomes 000000mm
CVD R3, DBL Month only
UNPK DATE(2), DBL Move to output
OI DATE+1, X'F0' Remove sign

(continued)
```
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 substantial savings of disk space and telecommunications time and cost. Some examples of binary conditions are:

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

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
Checking Bits: The \textit{TM} Instructions

As we've already seen, we can use the \texttt{OI}, \texttt{NI}, and \texttt{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 \texttt{TM} (Test under Mask instruction) is used to do so. The \texttt{TM} instruction is an SI-type instruction and has the form \texttt{TM field,mask}

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

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>MEANING</th>
<th>BC EQUIVALENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BO</td>
<td>Branch if Ones</td>
<td>BC 1,label</td>
</tr>
<tr>
<td>BM</td>
<td>Branch if Mixed</td>
<td>BC 4,label</td>
</tr>
<tr>
<td>BZ</td>
<td>Branch if Zeros</td>
<td>BC 8,label</td>
</tr>
<tr>
<td>BNO</td>
<td>Branch if Not Ones</td>
<td>BC 14,label</td>
</tr>
<tr>
<td>BNM</td>
<td>Branch if Not Mixed</td>
<td>BC 11,label</td>
</tr>
<tr>
<td>BNZ</td>
<td>Branch if Not Zeros</td>
<td>BC 7,label</td>
</tr>
</tbody>
</table>

**Example #12:** If the first and third bits of \texttt{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 \texttt{FLDB} are on, then turn on the first bit.

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

There is no SS equivalent to the \texttt{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.
CHAPTER 17
BIT LEVEL OPERATIONS

Exercises

1. True or false. Given A DC CL2’IQ’ and B 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 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 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 XI A,X’80’
   T  F d. To change the ‘Q’ in A to lower case, we code NI A+1,B’10111111’
   T  F e. To swap A and B we code XC A,B three times.
   T  F f. The value in B may have been a result of UNPK B,PK3 where PK3 is a packed number containing +158.
   T  F g. OI B+L’B-1,X’F0’ will give B equal to CL2’58’
   T  F h. Given TM A,X’C0’ and BZ SKIP the branch will be taken.
   T  F i. Given TM B+1,B’10000000’ and BZ SKIP the branch will be taken.
   T  F j. 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 SLL R4,3 followed by SRL R4,3 register 4 still contains 10.
   T  F m. Given register 4 contains 10. After SRL R4,3 followed by SLL R4,3 register 4 still contains 10.

2. Complete the following tables:

<table>
<thead>
<tr>
<th>OR</th>
<th>AND</th>
<th>EXCLUSIVE OR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>
   

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

   a. OI FLD,X’FC’

   b. NI FLD,X’E4’

   c. XI FLD,X’?A’
### Exercises

d.  **OI** FLD, X'B3'

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


e.  **NI** FLD, X'A6'

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


f.  **XI** FLD, X'AA'

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


g.  **OI** FLD, X'0F'

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


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

a.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


b.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


c.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>


d.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


e.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


f.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>


g.  

<table>
<thead>
<tr>
<th>FLDS</th>
<th>Masks</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MFDS</th>
<th>FLDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 2009 by Bill Qualls – All Rights Reserved
Chapter 17

Bit Level Operations

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.)