PIC Microcontroller and Embedded Systems
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq
The Islamic Uni. Of Gaza
Chapter 6: Bank Switching, Table processing, Macros and Modules
Objective

- List all addressing modes of PIC18 uCs
- Contrast and compare the addressing modes
- Code PIC18 instructions to manipulate a lookup table.
- Access fixed data residing in ROM space.
- Discuss how to create macros and models, and its advantages.
- Discuss how to access the entire 4kB of RAM
- List address for all 16 banks of the PIC18
- Discuss bank switching for the PIC18
Outlines

- Immediate and Direct Addressing mode
- Register indirect Addressing mode
- Lookup table and table processing
- Bit addressability of data RAM
- Bank switching
- Checksum and ASCII subroutines
- Macros and models
Introduction

- Data could be in
  - A register
  - In memory
  - Provided as an immediate values

- PIC18 provides 4 addressing modes
  - Immediate
  - Direct
  - Register indirect
  - Indexed-ROM
Section 6.1: Immediate and Direct Addressing mode

- In immediate addressing mode, the operands come after the opcode
  - `MOVW 0x25`
  - `SUBW D'34'`
  - `ADWLW 0x86`

- In direct addressing mode, the operand data is in a RAM location whose address is known and given as a part of the instruction.
Figure 6-1. MOVFF and MOVWF Direct Addressing Opcode

MOVLW 0X56
MOVWF 0X40
MOVFF 0X40,50H
Figure 6-1. MOVFF and MOVWF Direct Addressing Opcode

MOVLW 0X56
MOVWF 0X40
MOVFF 0X40,50H

A – bank accessed for operation
A = 0, use default access bank
A = 1, use bank pointed to by
BSR (Bank Selector Register)
Immediate and Direct Addressing mode

- What is the difference between
  - INCF fileReg, W
  - INCF fileReg, F
- What is the default destination?
- What is the difference between DECFSZ and DECF?
  - Operation
  - Branch
SFR Registers and their addresses

- Can be accessed by
  - Their name
  - Their address
- Which is easier to remember?

- `MOVWF PORTB`
- `MOVWF 0xF81`
SFR Registers and their addresses

Remember
- SFR addresses is started at F80h and the last location has the address FFFh

Notes
- In .lst file, you will see that the SFR names are replaced with their addresses.
- The WREG register is one of the SFR registers and has address FE8h.
Section 6.2: Register indirect Addressing mode

- A register is used as a pointer to the data RAM location.
- Three 12-bit Registers are used (from 0 to FFFh)
  - FSR0
  - FSR1
  - FSR2
- Each register is associated with INDFx
- Syntax
  - \texttt{LFSR n, data} \rightarrow \texttt{LFSR 1,95Eh} \rightarrow \text{needs 2 cycles}

FSR means file Select register
Advantages of Register indirect Addressing mode

- It makes accessing data dynamic
- Looping is possible to increment the address
  - Not possible in direct addressing mode
  - Example
    - INCF FSR2L
Example 6-2

- Write a program to copy the value 55H into RAM locations 40h to 45h using
  
  A. Direct addressing mode
  
  B. Register indirect addressing mode
  
  C. A loop

Solution A

MOVLW 0x55
MOVWF 0x40
MOVWF 0x41
MOVWF 0x42
MOVWF 0x43
MOVWF 0x44
Example 6-2 (cont.)

Solution B

```
MOVLW 55H
LFSR 0,0x40
MOVWF INDF0
INCF FSR0L,F
MOVWF INDF0
INCF FSR0L,F
MOVWF INDF0
INCF FSR0L,F
MOVWF INDF0
```

Solution C

```
COUNT EQU 0x10
MOVLW 0x5
MOVWF COUNT
LFSR 0,0x40
MOVLW 0x55
B1
MOVWF INDF0
INCF FSR0L,F
DECF COUNT,F
BNZ B1
```
Auto increment option for FSR

- Normal increment can cause problem since it increments 8-bit
  - INC FSR0L, F

- Auto increment and auto decrement solve the problem
  - They doesn’t affect the status flag
### PIC18 auto increment/decrement of FSRn

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLRF INDFn</td>
<td>After clearing fileReg pointed by FSRn, the FSRn stays the same</td>
</tr>
<tr>
<td>CLRF POSTINCn</td>
<td>After clearing fileReg pointed by FSRn, the FSRn is incremented (like x++)</td>
</tr>
<tr>
<td>CLRF PREINCN</td>
<td>The FSRn is incremented, then fileReg pointed to by FSRn is cleared (like ++x)</td>
</tr>
<tr>
<td>CLRF POSTDECDn</td>
<td>After clearing fileReg pointed by FSRn, the FSRn is decremented (like x++)</td>
</tr>
<tr>
<td>CLRF PLUSWN</td>
<td>Clears fileReg pointed by FSRn + WREG, and FSRn W are unchanged</td>
</tr>
</tbody>
</table>
Example 6-4

- Write a program to clear 16 RAM location starting at location 60H using Auto increment.

- Note: there are two identical mistakes in your book, pp 202. The right correction is FSR1=60H (not 40H)

Solution

COUNTREG EQU 0x10
CNTVAL EQU D'16'

MOVLW CNTVAL
MOVWF COUNTREG
LFSR 1,0x60

B3

CLRF POSTINC1
DECF COUNTREG,F
BNZ B3
Example 6-5

Write a program to copy a block of 5 bytes of data from location starting at 30H to RAM locations starting at 60H.

Solution

```
COUNTREG    EQU 0x10
CNTVAL      EQU D'5'

MOVLW      CNTVAL
MOVWF      COUNTREG
LFSR 0, 0x30
LFSR 1, 0x60
B3
MOVF      POSTINC0,W
MOVWF      POSTINC1
DECF      COUNTREG,F
BNZ        B3
```
Example 6-6

Assume that RAM locations 40-43H have the following hex data. Write a program to add them together and place the result in locations 06 and 07.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>040H</td>
<td>7D</td>
</tr>
<tr>
<td>041H</td>
<td>EB</td>
</tr>
<tr>
<td>042H</td>
<td>C5</td>
</tr>
<tr>
<td>043H</td>
<td>5B</td>
</tr>
</tbody>
</table>

Solution

COUNTREG EQU 0x20
L_BYTE EQU 0x06
H_BYTE EQU 0x07
CNTVAL EQU 4

MOVLW CNTVAL
MOVWF COUNTREG

LFSR 0,0x40
CLRF WREG
CLRF H_BYTE

ADDWF POSTINC0, W
BNC OVER

DECF COUNTREG,F
BNZ B5

MOVWF L_BYTE
The PIC uCs

Example 6-7

- Write a program to add the following multi-byte BCD numbers and save the result at location 60H.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>030H</td>
<td>77</td>
</tr>
<tr>
<td>031H</td>
<td>65</td>
</tr>
<tr>
<td>032H</td>
<td>89</td>
</tr>
<tr>
<td>033H</td>
<td>12</td>
</tr>
<tr>
<td>050H</td>
<td>39</td>
</tr>
<tr>
<td>051H</td>
<td>78</td>
</tr>
<tr>
<td>052H</td>
<td>64</td>
</tr>
<tr>
<td>053H</td>
<td>23</td>
</tr>
</tbody>
</table>

12896577 + 23647839

Solution

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTREG EQU 0x20</td>
<td></td>
</tr>
<tr>
<td>CNTVAL EQU D'4'</td>
<td></td>
</tr>
<tr>
<td>MOVLW CNTVAL</td>
<td></td>
</tr>
<tr>
<td>MOVWF COUNTREG</td>
<td></td>
</tr>
<tr>
<td>LFSR 0,0x30</td>
<td></td>
</tr>
<tr>
<td>LFSR 1,0x50</td>
<td></td>
</tr>
<tr>
<td>LFSR 2,0x60</td>
<td></td>
</tr>
<tr>
<td>BCF STATUS,C</td>
<td></td>
</tr>
<tr>
<td>MOVF POSTINC0,W</td>
<td></td>
</tr>
<tr>
<td>ADDWF POSTINC1,W</td>
<td></td>
</tr>
<tr>
<td>DAW</td>
<td></td>
</tr>
<tr>
<td>MOVWF POSTINC2</td>
<td></td>
</tr>
<tr>
<td>DECF COUNTREG,F</td>
<td></td>
</tr>
<tr>
<td>BNZ B3</td>
<td></td>
</tr>
</tbody>
</table>
Section 6.3: Lookup table and table processing

- Beside instructions, ROM has enough space to store fixed data

- DB directive, which means Define Byte, is widely used to allocate ROM program memory in byte-sized chunks

- Use single quotes ('') for a single character or double quotes (") for a string
  - Org 0x500
  - DATA1 DB 0x39
  - DATA2 DB 'z'
  - DATA3 DB "Hello All"

- ROM address must be even
Reading table elements in PIC18

- Program counter is 21-bit, which is used to point to any location in ROM space.
- How to fetch data from the code space?
  - Known as a **table processing**: register indirect ROM addressing mode.
  - There are table read and table write instructions
Reading table elements in PIC18

- To read the fixed data byte
  - We need an address pointer: TBLPTR
    - Points to data to be fetched
    - 21 bits as the program counter!!
    - Divided into 3 registers: TBLPTRL, TBLPTRH, TBLPTRU (all parts of SFR)
    - Is there any instruction to load 21 bits (as LFSR)?
  - A register to store the read byte
    - TBLLAT: keeps the data byte once it is fetched into the CPU
Auto increment option for TBLPTR

- Can you use the following instruction
  - INCF TBLPTR, f
- Cause Problem
- Example: Assume that ROM space starting at 250H contains "Embedded System", write a program to send all characters to PORTB one byte at a time

<table>
<thead>
<tr>
<th>TBLRD*</th>
<th>Table Read</th>
<th>After Read, TBLPRTR stays the same</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBLRD*+</td>
<td>Table Read with Post-inc</td>
<td>Reads and inc. TBLPTR</td>
</tr>
<tr>
<td>TBLRD*-</td>
<td>Table Read with Post-dec</td>
<td>Reads and dec TBLPTR</td>
</tr>
<tr>
<td>TBLRD++</td>
<td>Table Read with pret-inc</td>
<td>Increments TBLPTR and then reads</td>
</tr>
</tbody>
</table>
Example 6.10a

```
RCOUNT EQU 0x20
CNTVAL EQU 0x0F

ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL

MOVLW 0x02
MOVWF TBLPTRH

MOVLW CNTVAL
MOVWF RCOUNT

CLRF TRISB

B6 TBLRD*
MOVFF TABLAT,PORTB
INCF TBLPTRL,F
DECF RCOUNT,F
BNZ B6
HERE GOTO HERE

ORG 0x250
MYDATA DB "Embedded System"
END
```
<table>
<thead>
<tr>
<th>Address</th>
<th>00</th>
<th>02</th>
<th>04</th>
<th>06</th>
<th>08</th>
<th>OA</th>
<th>OC</th>
<th>OE</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>01A0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>01B0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>01C0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>01D0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>01E0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>01F0</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0200</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0210</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0220</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0230</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0240</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0250</td>
<td>6D45</td>
<td>6562</td>
<td>6464</td>
<td>6465</td>
<td>5320</td>
<td>7379</td>
<td>6574</td>
<td>006D</td>
<td>Embedded System</td>
</tr>
<tr>
<td>0260</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0270</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0280</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0290</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
</tbody>
</table>
Why don’t you see the “Embedded System”
Example 6.10b

```
ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL
MOVLW 0x02
MOVWF TBLPTRH
CLRF TRISB
B7 TBLRD*
MOVF TABLAT,W
BZ EXIT
 MOVWF PORTB
INCF TBLPTRL,F
BRA B7
EXIT GOTO EXIT
```

```
ORG 0x250
MYDATA DB "Embedded System",0
END
```
Example 6.11: Auto increment

ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL
MOVLW 0x02
MOVWF TBLPTRH
CLRF TRISB
B7 TBLRD*+
MOVF TABLAT,W
BZ EXIT
MOVWF PORTB
BRA B7
EXIT GOTO EXIT

ORG 0x250
MYDATA DB "Embedded System",0
END
Summery

The PIC uCs

Store Data

In RAM
- FSR: Pointer to the RAM address
- INDFx: Contains Data

In ROM
- TBLPTR (21 bits)
- TBLLAT Contains Data
Look-Up table and RETLW

- Used to access elements of a frequently used with minimum operations
- Example: $x^2$
- We can use a look-up table instead of calculating the values **WHY?**
- We need to a fixed value to the PCL to index into the look-up table
- **RETLW** (Return Literal to $W$) will provide the desired look-up table element in WREG
Example 6-14

- Write a program to get the x value from PORT B and send $x^2$ to port C.
- Use look-up table instead of a multiply instruction.
- Use PB3-PB0

```
ORG 0
SETF TRISB
CLRF TRISC
B1 MOVF PORTB,W
ANDLW 0x0F
CALL XSQR_TABLE
MOVWF PORTC
BRA B1

XSQR_TABLE
MULLW 0x2
MOVFF PRODL, WREG
ADDF W PCL
RETLW D'0'
RETLW D'1'
RETLW D'4'
RETLW D'9'
RETLW D'16'
RETLW D'25'
RETLW D'36'
RETLW D'49'
RETLW D'64'
RETLW D'81'
END
```
Example 6-14

<table>
<thead>
<tr>
<th>Address</th>
<th>00</th>
<th>02</th>
<th>04</th>
<th>06</th>
<th>08</th>
<th>0A</th>
<th>0C</th>
<th>0E</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>6893</td>
<td>6A94</td>
<td>5081</td>
<td>0B0F</td>
<td>EC08</td>
<td>F000</td>
<td>6E82</td>
<td>D7FA</td>
<td>.h.j.P...</td>
</tr>
<tr>
<td>0010</td>
<td>0D02</td>
<td>CFF3</td>
<td>FFE8</td>
<td>26F9</td>
<td>0C00</td>
<td>0C01</td>
<td>0C04</td>
<td>0C09</td>
<td>.......... &amp;</td>
</tr>
<tr>
<td>0020</td>
<td>0C10</td>
<td>OC19</td>
<td>OC24</td>
<td>OC31</td>
<td>OC40</td>
<td>OC51</td>
<td>FFFF</td>
<td>FFFF</td>
<td>.......... $</td>
</tr>
<tr>
<td>0030</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0040</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0050</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0060</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0070</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0080</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>0090</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
<tr>
<td>NNAN</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>FFFF</td>
<td>...........</td>
</tr>
</tbody>
</table>
Accessing a look-up table in RAM

- Store data in a continue location
- Using FSR as a pointer and the working register as an index
- For example:
  - MOVFF PLUS2 , PortD
  - Will copy data from location pointed by FSR2+WREG into PortD
Example 6-15: $X^2$

ORG 0
MOVLW 0
MOVWF 40H
MOVLW 1
MOVWF 41H
MOVWF 42H
MOVLW .9
MOVWF 43H
MOVLW .16

SETF TRISC
CLRF TRISD
LFSR 2,0x40

B1 MOVF PORTC,W
ANDLW B'00000111'
MOVFF PLUSW2,PORTD
BRA B1
END
Example 6-15

### File Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>030</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>00</td>
<td>00</td>
<td>04</td>
<td>09</td>
<td>10</td>
<td>19</td>
<td>24</td>
<td>31</td>
<td>40</td>
<td>51</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........ $1 @Q</td>
<td></td>
</tr>
<tr>
<td>050</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>060</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>........</td>
<td></td>
</tr>
</tbody>
</table>

Hex | Symbolic
### Example 6-15

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0018</td>
<td>0E24</td>
<td>MOV LW 0x24</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>6E46</td>
<td>MOVWF 0x46, ACCESS</td>
</tr>
<tr>
<td>15</td>
<td>001C</td>
<td>0E31</td>
<td>MOV LW 0x31</td>
</tr>
<tr>
<td>16</td>
<td>001E</td>
<td>6E47</td>
<td>MOVWF 0x47, ACCESS</td>
</tr>
<tr>
<td>17</td>
<td>0020</td>
<td>0E40</td>
<td>MOV LW 0x40</td>
</tr>
<tr>
<td>18</td>
<td>0022</td>
<td>6E48</td>
<td>MOVWF 0x48, ACCESS</td>
</tr>
<tr>
<td>19</td>
<td>0024</td>
<td>0E51</td>
<td>MOV LW 0x51</td>
</tr>
<tr>
<td>20</td>
<td>0026</td>
<td>6E49</td>
<td>MOVWF 0x49, ACCESS</td>
</tr>
<tr>
<td>21</td>
<td>0028</td>
<td>6894</td>
<td>SETF 0xf94, ACCESS</td>
</tr>
<tr>
<td>22</td>
<td>002A</td>
<td>6A95</td>
<td>CLR F 0xf95, ACCESS</td>
</tr>
<tr>
<td>23</td>
<td>002C</td>
<td>EE20</td>
<td>LFSR 0x2, 0x40</td>
</tr>
<tr>
<td>24</td>
<td>002E</td>
<td>F040</td>
<td>NOP</td>
</tr>
<tr>
<td>25</td>
<td>0030</td>
<td>5082</td>
<td>MOV F 0xf82, W, ACCESS</td>
</tr>
<tr>
<td>26</td>
<td>0032</td>
<td>OB07</td>
<td>ANDLW 0x7</td>
</tr>
<tr>
<td>27</td>
<td>0034</td>
<td>CFDB</td>
<td>MOVFF 0xfdb, 0xf83</td>
</tr>
<tr>
<td>28</td>
<td>0036</td>
<td>FF83</td>
<td>NOP</td>
</tr>
<tr>
<td>29</td>
<td>0038</td>
<td>D7FB</td>
<td>BRA 0x30</td>
</tr>
<tr>
<td>30</td>
<td>003A</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
<tr>
<td>31</td>
<td>003C</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
<tr>
<td>32</td>
<td>003E</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
<tr>
<td>33</td>
<td>0040</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
<tr>
<td>34</td>
<td>0042</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
</tbody>
</table>
Example 6-16

- Write a program to get the x value from PORT B and send $x^2 + 2x + 3$ to port C.
- Use look-up table instead of a multiply instruction.
- Use PB3-PB0

```assembly
ORG 0
SETF TRISB
CLRF TRISC
B1 MOVF PORTB, W
ANDLW 0x0F
CALL XSQR_TABLE
MOVWF PORTC
BRA B1

XSQR_TABLE
MULLW 0x2
MOVFF PRODL, WREG
ADDWF PCL
RETLW D'3'
RETLW D'6'
RETLW D'11'
RETLW D'18'
RETLW D'27'
RETLW D'38'
RETLW D'51'
RETLW D'66'
RETLW D'83'
RETLW D'102'
END
```

The PIC uCs
Section 6.4: bit addressability of data RAM

- One of the basic feathers of the PIC18 is the bit addressability of RAM.
  - Bit-addressable instructions
    - Use only direct addressing mode
  - Byte-addressable instructions
**Status Register Bit-addressability**

- You can access any bit of the status register by their name.
- **Examples**
  - `BCF STATUS, C`
  - `BTFSS STATUS, Z`
Section 6.5: Bank switching in the PIC18

- PIC18 has maximum of 4K of RAM
  - Not all the space used.
  - The fileReg is divided into 16 banks of 256B each
  - Every PIC18 has the access bank (the first 128B of RAM + SFR)
  - Most PIC18 that access the data space in RAM has the ability to access any bank through setting an optional operand, called A
- Example: `MOVWF myReg, A`
  - If 0 it access the default bank (default)
  - If 1, it uses the bank selection register (BSR) to select the bank
Figure 6-3. Data RAM Registers
The BSR register and bank switching

- It is 8-bit register
  - 4 bits are used → 16 banks
  - Banks 0 (from 00 to FF)
  - Banks 1 (from 100 to 1FF)
  - Banks 2 (from 200 to 2FF)
  - ...
  - Banks F (from F00 to FFF) (includes SFR)
- Upon power-on reset, BSR is equal to 0 (default value)
Note 1: The Access RAM bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

2: The MOVFF instruction embeds the entire 12-bit address in the instruction.
A Bit in the Instruction Field for INCF F, D, A

- Two things must be done
  1. Load BSR with desired bank

```assembly
MYREG EQU 0x40
MOVLB 0x2
MOVLW 0
MOVWF MYREG
INCF MYREG, F, 1
INCF MYREG, F, 1
```

The PIC uCs

- D = F, destination is fileReg
- D = W, destination is WREG

- A = 0, use default access bank
- A = 1, use bank pointed to by BSR (Bank Selector Register)

- 0 ≤ f ≤ FF
Example 6-25

- Write a program to copy the value 55H into RAM locations 340h to 345h using
  A. Direct addressing mode
  B. A loop

Solution (A)

MOVLB 0x3
MOVLW 0x55
MOVWF 0x40, 1
MOVWF 0x41, 1
MOVWF 0x42, 1
MOVWF 0x43, 1
MOVWF 0x44, 1
MOVWF 0x44, 1
Example 6-25

Write a program to copy the value 55H into RAM locations 340h to 345h using

A. Direct addressing mode
B. A loop

Solution (B)

COUNT EQU 0x10  
MOVLB 0x3  
MOVLW 0x6  
MOVWF COUNT  
LFSR 0,0x340  
MOVLW 0x55  

B1

MOVWF INDF0,0  
INCF FSR0L  
DECF COUNT,F,0  
BNZ B1
Section 6.6: Checksum and ASCII subroutines

To ensure the integrity of ROM contents, every system must perform a checksum calculation.
- Corruption (caused by current surge)

To calculate the checksum byte
1. Add the bytes and drop the carries
2. Take the 2's complement of the total sum

To perform the checksum operation
1. Add all bytes, including the checksum byte
2. The result must be zero, else error
Example 6-29

- Find the checksum byte
  - $25H$
  - $+ 62H$
  - $+ 3FH$
  - $+ 52H$
  - $\underline{118H}$ (Drop the carry bit)
  - The 2's comp. is $E8$

- Perform the checksum
  - $25H$
  - $+ 62H$
  - $+ 3FH$
  - $+ 52H$
  - $+ \underline{E8H}$
  - $200$ (Drop the carry)
Example 6-29

If the second byte 62H has been changed into 22H. Show how the checksum method detects the error.

\[ 25H + 22H + 3FH + 52H + E8H + E8H = 1COH \] (Drop the carry bit)
Section 6.6: Checksum

AM_ADDR EQU 40H
COUNTREG EQU 0x20
CNTVAL EQU 4
CNTVAL1 EQU 5

ORG 0
CALL COPY_DATA
CALL CAL_CHKSUM
CALL TEST_CHKSUM
BRA $
COPY_DATA

MOVLW  low(MYBYTE)
MOVWF  TBLPTRL
MOVLW  high(MYBYTE)
MOVWF  TBLPTRH
MOVLW  upper(MYBYTE)
MOVWF  TBLPTRTRU
LFSR    0, RAM_ADDR
C1      TBLRD*+

MOVF    TABLAT,W
BZ EXIT
MOVWF   POSTINCO
BRA     C1
EXIT    RETURN
CAL_CHKSUM

MOVLW CNTVAL
MOVWF COUNTREG
LFSR 0,RAM_ADDR
CLRF WREG
C2 ADDWF POSTINC0,W
DECF COUNTREG,F
BNZ C2
XORLW 0xFF
ADDLW 1
MOVWF POSTINC0
RETURN

TEST_CHKSUM

MOVLW CNTVAL1
MOVWF COUNTREG
CLRF TRISB
LFSR 0,RAM_ADDR
CLRF WREG
C3 ADDWF POSTINC0,W
DECF COUNTREG,F
BNZ C3
XORLW 0x0
BZ G_1
MOVLW 'B'
MOVWF PORTB
RETURN

G_1 MOVLW 'G'
MOVWF PORTB
RETURN
Section 6.7: Macros and models

- Dividing a program into several models allows us to use models in other application.
  - Reduce time
  - Reduce of errors
- Increase the code size every time it invoked

MACRO Syntax

```
MACRO dummy1, dummy2 ...
..........
ENDM
```

Unique Body
Example:

- Write a delay macro and a MOVLF macro.

```c
#include P18F458.INC

NOEXPAND

DELAY_1 MACRO V1, TREG

LOCAL BACK

MOVLW V1

MOVWF TREG

BACK NOP NOP NOP DECF TREG,F

BNZ BACK

ENDM
```

Local declaration
Example, cont.

`MOVLF MACRO K, MYREG
  MOVLW K
  MOVWF MYREG
ENDM`

`ORG 0
  CLRF TRISB
  OVER MOVLF 0x55, PORTB
  DELAY_1 0x200, 0x10
  MOVLF 0xAA, PORTB
  DELAY_1 0x200, 0x10
  BRA OVER
END`
Figure 6-11. List File with NOEXPAND Option for Program 6-4

```assembly
00001 ;Program 6-4: toggling Port B using macros
00002   #include P18F458.INC
00003   NOEXPAND
00004 ;-----------------------sending data to fileReg macro
00005   MOVFL MACRO K, MYREG
00006   MOVLU  K
00007   MOVWF  MYREG
00008   ENDM
00009
00010 ;--------------------------------time delay macro
00011   DELAY_1 MACRO V1, TREG
00012   LOCAL  BACK
00013   MOVLU  V1
00014   MOVWF  TREG
00015   BACK  NOP
00016   NOP
00017   NOP
00018   NOP
00019   DECFF TREG,T
00020   BNZ   BACK
00021   ENDM
00022
00023 ;---------------------------------program starts
00000  00024   ORG    0
00025   6A93
00026   CLRFB TRISB ;Port B as an output
00027   MOVLF  0x55, PORTB
00028   MOVLF  0x200, PORTB
00029   MOVLF  0xAA, PORTB
00030   BRA   OVER
00031   ENDM
```

The PIC uCs
Figure 6-12. List File with EXPAND Option for Program 6-4

```
00001 ;Program 6-4: toggling Port B using macros
00002     #include P18F458.INC
00003     EXPAND
00004 ;-------------------sending data to fileReg macro
00005     MOVLF MACRO K, MYREG
00006     MOVLW K
00007     MOVWF MYREG
00008 ENDM
00009
00010 ;-----------------------------time delay macro
00011     DELAY_1 MACRO V1, TREG
00012     LOCAL BACK
00013     MOVLW V1
00014     MOVWF TREG
00015     BACK ND
00016     ND
00017     ND
00018     ND
00019     DECFSZ TREG, r
00020     BNZ BACK
00021     ENDM
00022
```
Figure 6-12. List File with EXPAND Option for Program 6-4 (cont.)

```
000000 00024  ORG 0
000000 6A93 00025  CLRF TRISB ;Port B as an output
000026 OVER MOVLF 0x55, PORTB
000002 0E55 M  MOVWL 0x55
000004 6E81 M  MOVWF PORTB
        00027 DELAY_1 0x200, 0x10

0000  M  LOCAL BACK
000006 0E00 M  MOVWL 0x200
000008 6E10 M  MOVWF 0x10
00000A 0000 M  BACK NOP
00000C 0000 M  NOP
00000E 0000 M  NOP
000010 0000 M  NOP
000012 0610 M  DECF 0x10, F
000014 E1FA M  BNZ BACK
        00028 MOVLF 0xAA, PORTB
000016 0EAA M  MOVWL 0xAA
000018 6E81 M  MOVWF PORTB
        00029 DELAY_1 0x200, 0x10

0000  M  LOCAL BACK
00001A 0E00 M  MOVWL 0x200
00001C 6E10 M  MOVWF 0x10
00001E 0000 M  BACK NOP
000020 0000 M  NOP
000022 0000 M  NOP
000024 0000 M  NOP
000026 0610 M  DECF 0x10, F
000028 E1FA M  BNZ BACK
00002A D7EB 00030  BRA OVER
        00031 END
```
Chapter 6: Summary

Next: Chapter 9
Arithmetic, logic
Instruction and programs