Addressing Modes, Table, User Stack

1

1

1. Address Mode
2. Table
3. Stack and Subroutine Call

2

2

1. **Address Mode**
2. Table
3. Stack and Subroutine Call

3

3

Introduction

- Data (source and destination) could be in
  - Register
  - Memory (data or program)
  - Immediate values (part of instruction)
- PIC18 provides 4 addressing modes
  - (The waysof accessing data is called *addressing mode*.)
  - Immediate
  - Direct
  - Register indirect
    - Indexed-ROM

4

4

Immediate and Direct Addressing mode

Add. Code

0000 0E56 MOVLW 0x56

0002 6E40 MOVWF 0x40, ACCESS

0004 C040 MOVFF 0x40, 0x50

0006 F050

0008 EF00 GOTO 0

000A F000

56 →WREG

WREG → loc 40H

(Loc 40H)→loc 50H

5

5

Immediate and Direct Addressing mode

What is the difference between

```
INCF fileReg, W
INCF fileReg, F
```

6

6

SFR Registers and their addresses

- Can be access by Their name
- Their address

- Which is easier to remember?

```
MOVWF PORTB
MOVWF 0xF81
```

Address	Symbol Name	Value
FE8	WREG	0x56
F80	PORTA	0x00
F81	PORTB	0x00
F83	PORTD	0x00
F82	PORTC	0x00
F89	LATA	0x00
F8A	LATB	0x00
F8B	LATC	0x00
F8C	LATD	0x00
F92	TRISA	0x7F
F93	TRISB	0xFF
F94	TRISC	0x00
F95	TRISD	0xFF
	INDF0	ed Memory
	INDF1	ed Memory
FE9	FSR0	0x0000
FE9	FSR0L	0x00
FEA	FSR0H	0x00
FE1	FSR1L	0x00
FE2	FSR1H	0x00

7

7

SFR Registers and their addresses

SFR addresses are started at F80H and the last location is the address FFFH

In Listing 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

```
0000 0E56 MOVLW 0x56          ; MOVLW 56H
0002 6E81 MOVWF 0xf81, ACCESS ; MOVWF PORTB
```

8

8

Register indirect Addressing mode

A register is used as a pointer to the data RAM location.

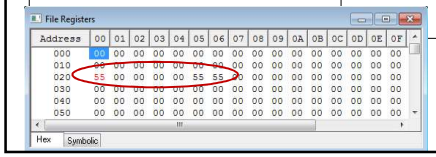
In PIC18 Three 12-bit Registers are used (from 0 to FFFh)

FSR0, FSR1, FSR2, Each register is associated with INDFx (shadow registers for using FSRn)

MOVFW INDF0 (move the WREG to the location pointed by FSR0)

```
cblock 0x020
   data_table: 5
   R1
   R2
endc
```

```
Main ORG          0x0000
     LFSR   0, data_table ; FSR0=020h
     movlw  0x55
     movwf  R1
     movwf  R2
     movwf  INDF0
Here goto here
```



9

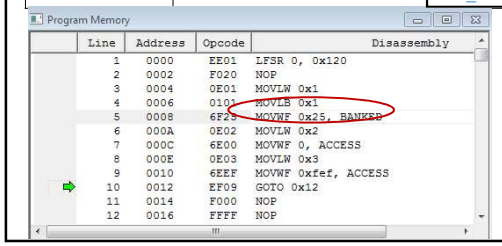
9

Register indirect Addressing mode

```
cblock 0x000
   R2
endc
cblock 0x120
   data_table: 5
   R1
endc
```

```
Main ORG          0x0000
     LFSR   0, data_table ; FSR0=120h
     movlw  01H
     movlb  high R1
     movwf  R1
     movlw  02H
     movwf  R2
     movlw  03H
     movwf  INDF0
Here goto here
```

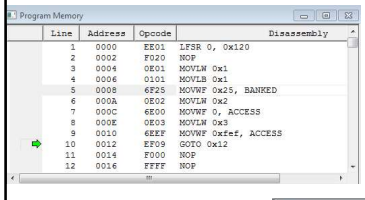
R1 address = 125H
R2 address = 000H
data_table address = 120H

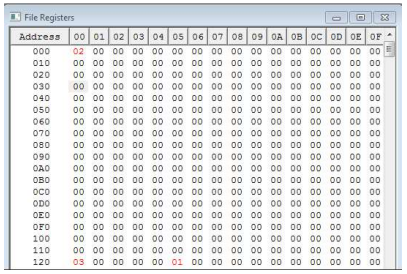


10

10

Register indirect Addressing mode





11

11

1. Address Mode
2. **Table**
3. Stack and Subroutine Call

12

12

Tables

Indirect addressing makes accessing data dynamic. That is why we can implement array in high level language.

Looping is possible to increment the address in FSR with

Instructions	
XXXX INDFn	After accessing fileREG pointed by FSFn, the FSRn is unchanged.
XXXX POSTINCn	After accessing fileREG pointed by FSFn, the FSRn is incremented
XXXX PREINCn	Before accessing fileREG pointed by FSFn, the FSRn is incremented.
XXXX POSTDECn	After accessing fileREG pointed by FSFn, the FSRn is decremented
XXXX PLUSWn	Address = WREG + FSRn

XXXX stays for some instructions

13

13

Example

Write a program to copy the value 55H into RAM locations 40h to 45h using

- Direct addressing mode
- Register indirect addressing mode
- A loop

14

14

Example

Solution B

```
MOVLW 55H
LFSR 0,0x40
MOVWF POSTINC0
MOVWF POSTINC0
MOVWF POSTINC0
MOVWF POSTINC0
MOVWF POSTINC0
```

15

15

Example

Solution C

```
        LFSR    0, 0x040
        MOVLW   0x5
        MOVWF   COUNT
        MOVLW   0x55
B1      MOVWF   POSTINC0
        DECF    COUNT,F
        BNZ     B1
```

Of course C is much better if the table size is large

16

16

Example

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

```
        COUNTREG   EQU   0x10
        CNTVAL     EQU   D'16'
        MOVLW      CNTVAL
        MOVWF      COUNTREG
        LFSR       1,0x60
B3      CLRF       POSTINC1
        DECF       COUNTREG,F
        BNZ        B3
```

17

17

Example

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

```
        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
```

18

18

Example

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.

```
        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
B5      ADDWF      POSTINC0, W
        BNC        OVER
        INCF       H_BYTE,F
OVER    DECF       COUNTREG,F
        BNZ        B5
        MOVWF      L_BYTE
```

Address	Data
040H	7D
041H	EB
042H	C5
043H	5B

19

19

Example

- Write a program to add the following multi-byte BCD numbers and save the result at location 60H. 12896577 +23647839

```
  COUNTREG      EQU 0x20
  CNTVAL   EQU D'4'
  MOVLW         CNTVAL
  MOVWF         COUNTREG
  LFSR          0,0x30
  LFSR          1,0x50
  LFSR          2,0x60
  BCF      STATUS,C
B3   MOVF       POSTINC0,W
  ADDWFC   POSTINC1,W
  DAW
  MOVWF         POSTINC2
  DECF          COUNTREG,F
  BNZ      B3
```

Address	Data
030H	77
031H	65
032H	89
033H	12
050H	39
051H	78
052H	64
053H	23

20

20

Fixed Data in Program Space

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"
```

21

21

Fixed Data in Program Space

- Program conter 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.
- 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
  - A register to store the read byte **TBLLAT**: keeps the data byte once it is fetched into the CPU

22

22

Fixed Data in Program Space

- Is there any instruction to load 21 bits

```
MOVLW  low   MYDATA
MOVWF        TBLPTRL
MOVLW  high  MYDATA
MOVWF        TBLPTRH
MOVLW  upper MYDATA
MOVWF        TBLPTRU
```

- A register to store the read byte **TBLAT**: keeps the data byte once it is fetched into the CPU. Use instruction TBLRD* to read the data from ROM (put it in TBLLAT)

TBLRD*	Table Read	After Read, TBLPRTR stays the sam
TBLRD*+	Table Read with Post-inc	Reads and inc. TBLPTR
TBLRD*-	Table Read with Post-dec	Reads and dec TBLPTR
TBLRD+*	Table Read with pret-inc	Increments TBLPTR and then reads

23

23

Example

Write a program to read a byte in MYDATA, and then put it to PORTD

```
000000                  00004 Main    ORG     0x0000
000000 6A95             00005                 CLRF    TRISD
000002 0E18             00006                 MOVLW   low MYDATA
000004 6EF6             00007                 MOVWF   TBLPTRL
000006 0E00             00008                 MOVLW   high MYDATA
000008 6EF7             00009                 MOVWF   TBLPTRH
00000A 0E00             00010                 MOVLW   upper MYDATA
00000C 6EF8             00011                 MOVWF   TBLPTRU
00000E 0008             00012                 TBLRD*
000010 50F5             00013                 MOVF    TABLAT, W
000012 6E83             00014                 MOVWF   PORTD
                        00015
000014 EF0A F000        00016                 here    goto    here
                        00017
000018 0041             00018 MYDATA  DB "A"
                        00019
                        00020                 END
```

24

24

Example

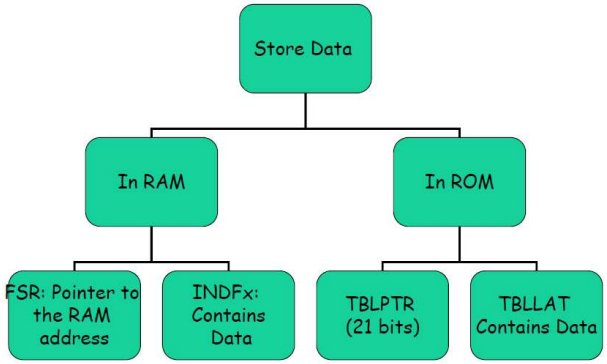
Assume that ROM space starting at 250H contains "Embedded System“, write a program to send all characters to PORTD one byte at a time

```
000000                  00004 Main    ORG  0x0000
000000 6A95             00005         CLRF    TRISD
000002 0E50             00006         MOVLW   low MYDATA
000004 6EF6             00007         MOVWF   TBLPTRL
000006 0E02             00008         MOVLW   high YDATA
000008 6EF7             00009         MOVWF   TBLPTRH
00000A 0E00             00010         MOVLW   upper MYDATA
00000C 6EF8             00011         MOVWF   TBLPTRU
00000E 0009             00012 B7      TBLRD*+
000010 50F5             00013         MOVF    TABLAT,W
000012 E002             00014         BZ      EXIT
000014 6E83             00015         MOVWF   PORTD
000016 D7FB             00016         BRA     B7
000018 EF0C F000        00017 EXIT     GOTO    EXIT
000250                  00018         ORG     0x250
000250 6D45 6562 6464   00019 MYDATA  DB "Embedded System",0
       6465 5320 7379
       6574 006D
                        00020               END
```

25

25

Summary



26

26

Look-Up table

- 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?**
- Store the function f(x) in a table (RAM or ROM)
- We can get the base address of the table
- Input *x* provides the displacement

27

27

Look-Up table with RETLW

- RETLW (Return from subroutine with Literal to WREG) will provide the desired look-up table element in WREG
- Before execute RETLW, we need to a fixed value (displacement) to the PCL to index into the look-up table. So PC=PC+disp

  And the corresponding RETLW is at the PC+disp.

28

28

Example

Write a program to get x2. Use look-up table instead of a mutliply instruction.

```
Main
        ORG    0x0000
       MOVLW    d'4'
       call     XSQR_TABLE
  EXIT GOTO EXIT
```

```
XSQR_TABLE
        MULLW    0x2
        MOVFF   PRODL, WREG
        ADDWF   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
```

Since RETLW occupies two bytes, we need MULLW 02.

Any potential problem ? ADD does not affect the whole PC.

29

29

Example

Write a program to get x2. Use look-up table instead of a mutliply instruction. Use **TBLAT**

```
    Input EQU 0
Main      ORG   0x0000
          MOVLW   d'9'
          MOVWF    input
          call    XSQR
EXIT      GOTO EXIT
XSQR    MOVLW   low XSQR_TABLE
        MOVWF   TBLPTRL
        MOVLW   high XSQR_TABLE
        MOVWF   TBLPTRH
        MOVLW   upper XSQR_TABLE
        MOVWF   TBLPTRU
        MOVF    input, W
        ADDWF   TBLPTRL, F
       MOVLW    0
       ADDWFC   TBLPTRH
       ADDWFC   TBLPTRU
       TBLRD*
       MOVF     TABLAT, W
        RETURN
```

```
ORG    0x0FE
XSQR_TABLE
 db D'0', D'1',D'4',D'9',
 db D'16',D'25',D'36',D'49',
 db D'64',D'81'
END
```

30

30

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

31

31

Example

Write a program to get x2. Use look-up table instead of a mutliply instruction. Use **FSR.** FSR provide the base addr. W provide disp.

```
table   equ 0
Main    ORG    0x0000
        LFSR  0, table
        MOVLW 0
        MOVWF POSTINC0
        MOVLW d'1'
        MOVWF POSTINC0
        MOVLW d'4'
        MOVWF POSTINC0
        MOVLW d'9'
        MOVWF POSTINC0
        MOVLW d'16'
        MOVWF POSTINC0
        MOVLW d'25'
        MOVWF POSTINC0
        MOVLW d'36'
        MOVWF POSTINC0
        MOVLW d'49'
        MOVWF POSTINC0
        MOVLW d'64'
        MOVWF POSTINC0
        MOVLW d'81'
        MOVWF POSTINC0
  ;for initialization of the table in RAM
```

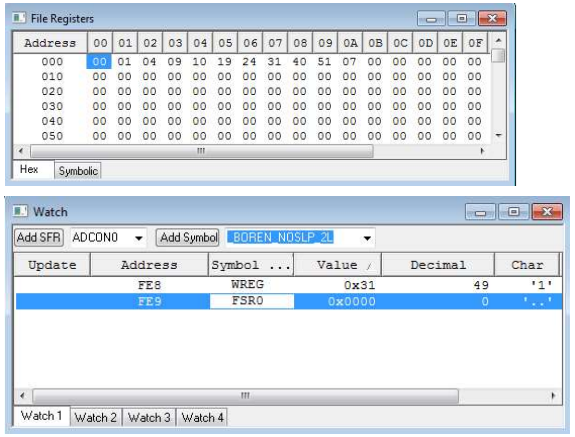
```
      MOVLW d'7'
      call  XSQR
EXIT GOTO EXIT

XSQR   LFSR     0, table
       MOVFF    PLUSW0, WREG
       RETURN
       END
```

32

32

Example



33

33

Review: bit-addressability

- Often need to access individual bit of the port instead of the entire 8 bits.
- PIC18 provides instructions that alter individual bits without altering the rest of the bits in the port.
- Most common bit-oriented instructions:

Instructions		Function
bsf	fileReg, bit	Bit Set fileReg
bcf	fileReg, bit	Bit Clear fileReg
btg	fileReg, bit	Bit Toggle fileReg
btfsc	fileReg, bit	Bit test fileReg, skip if clear
btfss	fileReg, bit	Bit test fileReg, skip if set

34

34

1. Address Mode
2. Table
3. **Stack and Subroutine Call**

35

35

Stack and Subroutine Call

- In modern computers, we need a stack to store the return address during subroutine call. Besides, we need to save the register values that will be modified by the subroutine.
- In the PIC18, we only have a hardware stack for storing return address.
- How to solve this problem.

36

36

Stack and Subroutine Call

```
ascii_l   EQU 0x0 ;input parameter
ascii_h   EQU 0x1 ;input parameter
out_bcd   EQU 0x2 ;output value
local_var EQU 0x3

Main program

… …
… …
CALL SUB1
… …
… …
CALL SUB1
… …
```

```
SUB1
  … …
  this routine may modify
  WREG, STATUS, BSR
  … …
RETURN
```


Stack and Subroutine Call

Use some locations to store the affected registers

```
ascii_l   EQU 0x0 ;input parameter
ascii_h   EQU 0x1 ;input parameter
out_bcd   EQU 0x2 ;output value
local_var EQU 0x3
W_TEMP_SUB1     EQU 0x4
Flag_TEMP_SUB1 EQU 0x5
BSR_TEMP_SUB1   EQU 0x6

Main program

… …
… …
CALL SUB1
… …
… …
CALL SUB1
… …
```

```
SUB1
 MOVWF W_TEMP_SUB1
 MOVFF STATUS, STATUS_TEMP) SUB1
 MOVFF BSR, BSR_TEMP_SUB1
 … …
 this routine may modify
 WREG, STATUS, BSR
 … …
 MOVFF BSR_TEMP_SUB1, BSR
 MOVFF STATUS_TEMP_SUB1, STATUS
 MOVF W_TEMP_SUB1,W
RETURN
```

Waste resource if we have many routines.
Also, the routines cannot be re-entrence

Stack and Subroutine Call

We can use FSR to implement a user stack.

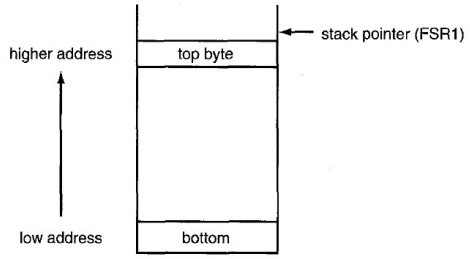


Figure 4.2 ■ The PIC18 Software Stack

Stack and Subroutine Call

We can use FSR to implement a user stack.

Use LFSR to define a stack

```
LFSR FSR1, 0x500
```

The following instruction will push the STATUS register onto the stack:

```
MOVFF STATUS,PREINC1
```

The following instruction sequence will pull the top byte of the stack onto the STATUS register:

```
MOVFF POSTDEC1,STATUS
```

Stack and Subroutine Call

```
stack   equ 0x200
R0      equ 0
R1      equ 1
R2      equ 2
Main    ORG     0x0000
        LFSR  0, stack          ; define a user stack
        MOVLW 0x00
        MOVWF R0
        MOVLW 0x10
        MOVWF R1
        MOVLW 0x20
        MOVWF R2
        MOVLW 0x99
        MOVFF R0,   PRETINC0    ; push R0 to stack   FSR0=0201
        MOVFF R1,   PRETINC0    ; push R1 to stack   FSR0=0202
        MOVFF R2,   PRETINC0    ; push R2 to stack   FSR0=0203
        MOVFF WREG, PRETINC0    ; push WREG to stack FSR0=0204

        MOVLW 0x71              ; Now program change W, R0, R1, R2
        MOVWF R0                ;
        MOVWF R1                ;
        MOVWF R2                ;
                                ; restore the value from stack
        MOVFF POSTDEC0, WREG    ; FSR0=0203
        MOVFF POSTDEC0, R0      ; FSR0=0202
        MOVFF POSTDEC0, R1      ; FSR0=0201
        MOVFF POSTDEC0, R2      ; FSR0=0200

        EXIT GOTO EXIT

 END
```

41

41

Stack and Subroutine Call

Use stacks store the affected registers

```
ascii_1   EQU 0x0 ;input parameter
ascii_h   EQU 0x1 ;input parameter
out_bcd   EQU 0x2 ;output value
local_var EQU 0x3
stack     EQU 0x200; define a stack
```

Main program

```
    LFSR  0, stack
… …
… …
CALL SUB1
… …
… …
CALL SUB1
… …
```

SUB1

```
MOVFF WREG, PREINC0
MOVFF STATUS, PREINC0
MOVFF BSR, PREINC0
… …
```

this routine may modify WREG, STATUS, BSR

```
… …
MOVFF POSTDEC0, BSR
MOVFF POSTDEC0, STATUS
MOVFF POSTDEC0,WREG
```

RETURN

In general, modern computer systems use stack to pass parameters and to handle local variables. But the procedure is very complicated. If you are interested in this issue, read PIC Microcontroller: An Introduction to Software & Hardware Interfacing Chapter 4.

42

42