

# **Appendix A:**

## **Improving Code Size With the MPLAB C18 Compiler**

# **Suggestion #1**

*Carefully select command-line options*

# Command-Line Options

## LFSR Use

MPLAB-C18's **-lfsr** switch enables use of the LFSR instruction

Currently, MPLAB-C18 assumes that LFSR shouldn't be used without the **-lfsr** switch given

The switch should always be used when it is known that the LFSR errata doesn't exist on the targeted part

# **Command-Line Options**

## **Optimization**

All of MPLAB-C18's optimizations currently target code size

Optimizations should be enabled for smallest code size

**NOTE:** Optimizations may interfere with MPLAB debugging

# Command-Line Options

## Memory Model

MPLAB-C18 has two memory models:

- ms**: small memory model (pointers to program memory are 16-bits wide)
- m1**: large memory model (pointers to program memory are 24-bits wide)

Use **-ms** whenever possible

## **Suggestion #2**

*Select appropriate storage class for data*

# Command-Line Options

## Data Storage Class

Default storage class for parameters and local variables is **auto**

Parameters are passed on the software stack

Locals are located on the software stack

# Using auto Variables

Example - calculate the expression (a + b):

<code>movlw</code>	<code>offset(a)</code>
<code>movff</code>	<code>PLUSW2, tmp</code>
<code>movlw</code>	<code>offset(b)</code>
<code>movf</code>	<code>PLUSW2</code>
<code>addwf</code>	<code>tmp</code>

**6 program words**  
**(not counting prolog/epilog)**



# Command-Line Options

## Data Storage Class

C also provides for `static` local variables

MPLAB-C18 extends C with `static` parameters  
(available in v1.10 and later)

For example:

```
char add( static char a, static char b )  
{  
    static char result;  
    result = a + b;  
    return result;  
}
```

# Using static Variables

Example - calculate the expression (a + b):

```
movlb      b*  
movf       b  
addwf      a
```

\*likely target for optimization

**3 program words**  
**(no prolog/epilog required)**

# **static Gotchas**

## **Gotcha #1 - Reentrant code**

*Variables may overwrite themselves*

Recursion (function calls itself)

Function called (directly or indirectly) from main() and an ISR.

# **static Gotchas**

## **Gotcha #2 - Function pointers**

*Address of parameters not known at compile time*

**Function pointers may not be used with functions containing static parameters**

# static Gotchas

## Gotcha #3 - Matching declarations

*All declarations must use explicit storage class if not all files are compiled with the same default*

Example:

```
char add( char a, char b );
```

Will only work if the default storage class is identical in both the declaring and defining files.

# **static Gotchas**

What if one of the “static Gotchas” applies to your code?

Best case: use `-o1` on all files and explicit `auto` storage class as needed.

Intermediate case: Use `-o1` on as many files as possible and explicit storage classes as needed.

Worst case: Don't use `-o1`, but use explicit `static` storage class as much as possible.

# Command-Line Options

## Data Storage Class

MPLAB-C18 v2.0 and later extends C with the **overlay** storage class for local variables

Behaves identically to the **static** storage class, except:

RAM locations are overlaid by the linker when possible based on a call tree analysis

Default storage class can be set to **overlay** using the **-sco** option

## **Suggestion #3**

*Choose smallest data type possible*



# MPLAB-C18 Data Types

Type	Min Value	Max Value
unsigned char	0	255
signed char	-128	127
unsigned int	0	65,535
signed int	-32,768	32,767
unsigned short long	0	16,777,215
signed short long	-8,388,608	8,388,607
unsigned long	0	4,294,967,295
signed long	-2,147,483,648	2,147,483,647

# Using Appropriate Data Types

$$c = a + b$$

**char:**

```
MOVLB    b
MOVF      b,0,1
ADDWF     a,0,1
MOVWF     c,1
```

**(4 words)**

**int:**

```
MOVLB    a
MOVF      b,0,1
ADDWF     a,0,1
MOVWF     c,1
MOVF      high(b),0,1
ADDWFC    high(a),0,1
MOVWF     high(c),1
```

**(7 words)**

## **Suggestion #4**

*Use access RAM for your variables*

# Variable Allocation

## Using Access RAM

MPLAB-C18 allows for efficient use of unbanked RAM with the **near** type specifier

RAM variables will default to **near** by using the **-oa** option

Compiler won't emit **movlb** instructions for accessing these variables

# Variable Allocation

## Using Access RAM

Use the **near** specifier for the most frequently accessed variables

Gotcha: as with **static** and **overlay**, prototypes must match definitions

## **Suggestion #5**

*Keep definitions in same file with references*

# Variable Allocation

## Defining Variables

MPLAB-C18 can be more aggressive optimizing variables in the files where they are defined.

Source code:

```
char a, b, c;  
void foo( void )  
{  
    c = a + b;  
}
```

Machine code:

```
MOVLB      b  
MOVF       b, 0, 1  
ADDWF      a, 0, 1  
MOVWF      c, 1
```

(4 words)

# Variable Allocation

## Defining Variables

MPLAB-C18 must be more conservative with externally-defined variables

### Source code:

```
extern char a, b, c;  
void foo( void )  
{  
    c = a + b;  
}
```

### Machine code:

```
MOVLB    b  
MOVF     b, 0, 1  
MOVLB    a  
ADDWF    a, 0, 1  
MOVLB    c  
MOVWF    c, 1
```

(6 words)



## **Suggestion #6**

*Use #pragma varlocate*

# Using #pragma varlocate

Use #pragma varlocate to tell the compiler what bank a variable is located in

## Source code:

```
extern char a, b, c;
void foo( void )
{
    c = a + b;
}
```

## Machine code:

```
MOVLB      b
MOVF       b,0,1
MOVLB      a
ADDWF      a,0,1
MOVLB      c
MOVWF      c,1
```

(6 words)

# Using #pragma varlocate

Improves MPLAB-C18 banking optimizer

## Source code:

```
#pragma varlocate 3 a, b, c
extern char a, b, c;
void foo( void )
{
    c = a + b;
}
```

## Machine code:

```
MOVLB      b
MOVF       b,0,1
ADDWF      a,0,1
MOVWF      c,1
```

(4 words)

# Using `#pragma varlocate`

Gotcha: *has no impact on how variables are actually allocated*

## **Suggestion #7**

*Replace Common Expressions With Variables*

# Common Sub-Expression Elimination

Applies to all types of expressions

Source code:

```
MY_STRUCT s[10];  
for(i=0; i<10; i++)  
{  
    s[i].a = i;  
    s[i].b = 34;  
}
```

Code size:

```
10 words to calculate s[i]  
2 words to assign i  
10 words to calculate s[i]  
3 words to assign 34  
  
= 25 words total
```

# Common Sub-Expression Elimination

Source code:

```
MY_STRUCT s[10];
MY_STRUCT *p = &(s[0]);

for(i=0; i<10; i++)
{
    p->a = i;
    p->b = 34;
    p++;
}
```

Code size:

0	calculate s[i]
6	assign i
7	assign 34
4	increment p

= 17 words total

## **Suggestion #8**

*Don't Use a Variable When a Constant Will Do*



# Constant Evaluations

Pre-calculate all values that can be determined at compile-time.

**Original source:**

```
a = 2;  
b = 17 + 52 * a;  
c = b;
```

**Transformed source:**

```
c = 121;
```

**Appendix B:**

**PIC18FXXX**

**Instruction**

**Set and PIC16/17**

**Migration**

# PIC18 Architecture

## ALU: Status Register

### STATUS Register Format

bit 7							bit 0
-	-	-	N	OV	Z	DC	C

### Bit definitions

<b>N</b>	<b>N</b> egative/ <b>P</b> ositive	<i>ALU result is negative</i>
<b>OV</b>	<b>O</b> Verflow	<i>2's Complement Overflow occurred</i>
<b>Z</b>	<b>Z</b> ero	<i>Result is zero</i>
<b>DC</b>	<b>D</b> igit <b>C</b> arry / <b>!</b> Borrow	<i>Carry/borrow from lower nibble</i>
<b>C</b>	<b>C</b> arry / <b>!</b> Borrow	<i>Carry/borrow from upper nibble</i>

# PIC18 Architecture

## Data Memory

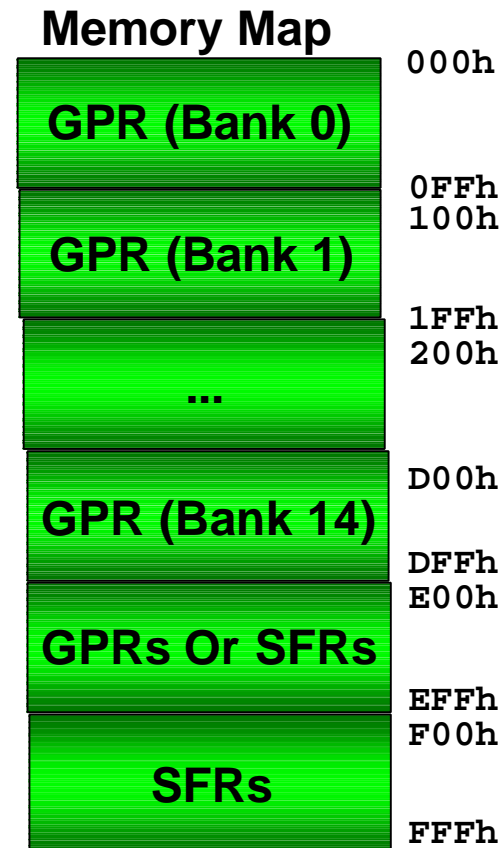
Up to 16 banks of  
256 bytes of SRAM

Unused banks read  
'00h'

Bank selected by  
BSR<3:0>

Linear access

SFR are located in  
Bank 14 and/or 15



# PIC18 Architecture

## Accessing Data Memory

Select a bank

BSR<3:0> contains bank

Instruction with 8-bit  
address as operand

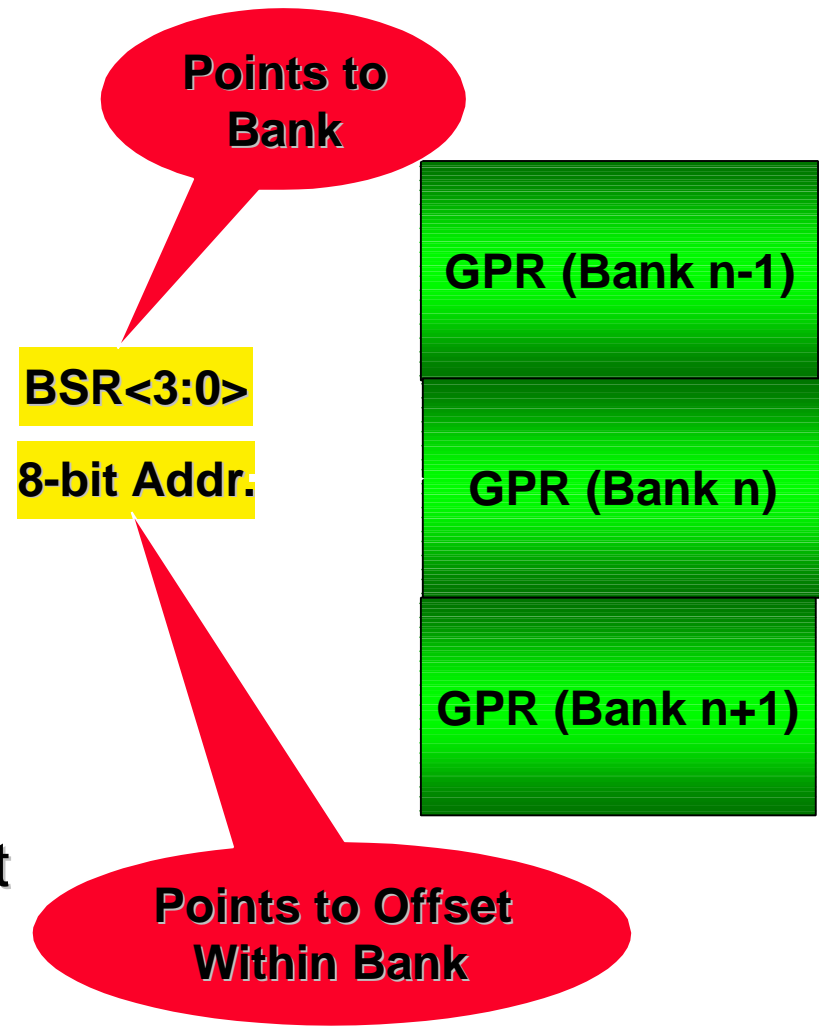
“BANKED” bit

MPASM assembler tip

12-bit Register address

Use BANKSEL directive

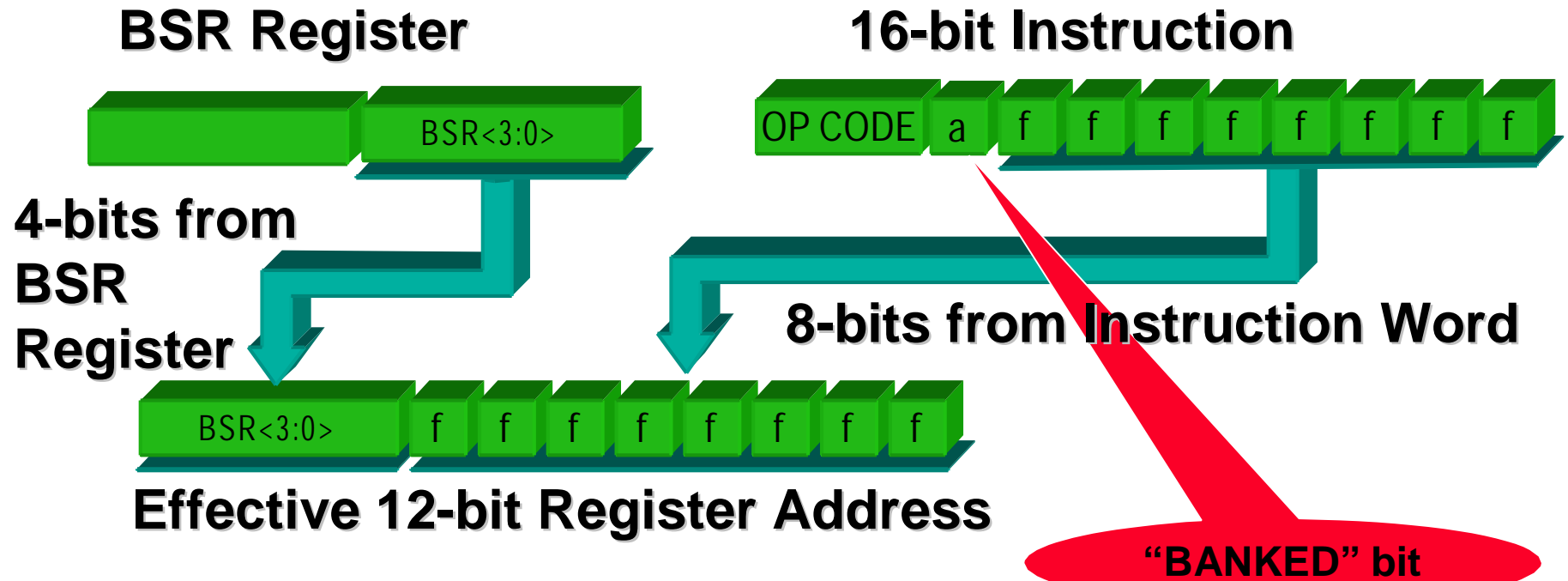
Let MPASM assembler set  
“BANKED” bit



# PIC18 Architecture

## Accessing Data Memory

Instruction Format Example:



# PIC18 Architecture

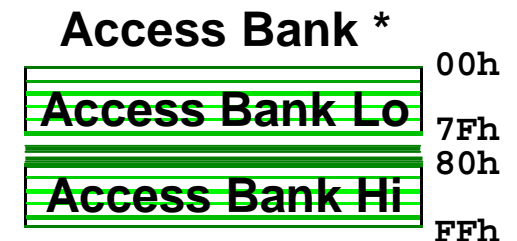
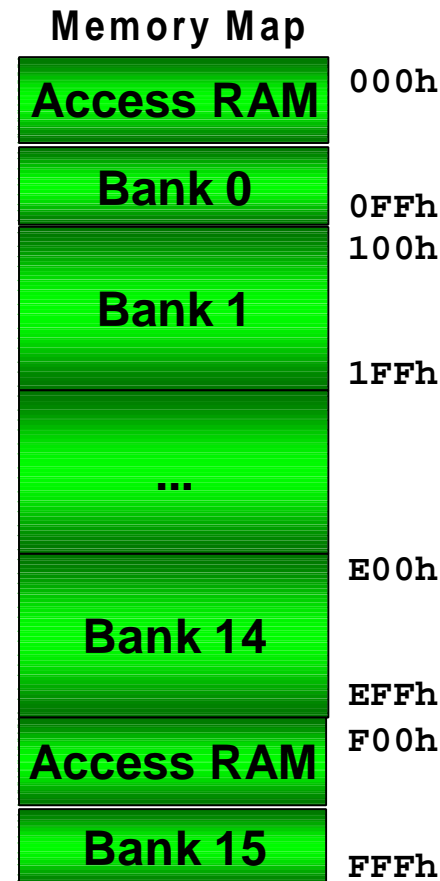
## Access Bank

256 bytes of non-banked memory

Fast access to frequently used registers (SFRs and GPRs)

Size of Access Bank depends on device

e.g. PIC18FXX2:  
128/128;  
PIC18FXX8: 96/160



\* Note: Check your device datasheet

# PIC18 Architecture

## Accessing Access Bank

Instruction with 8-bit  
address as operand

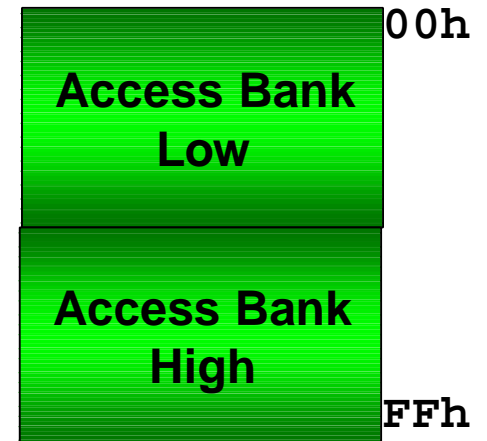
Special “**ACCESS**” bit

MPASM assembler tip

12-bit Register address

Let MPASM assembler  
set “**ACCESS**” bit

8-bit Add.

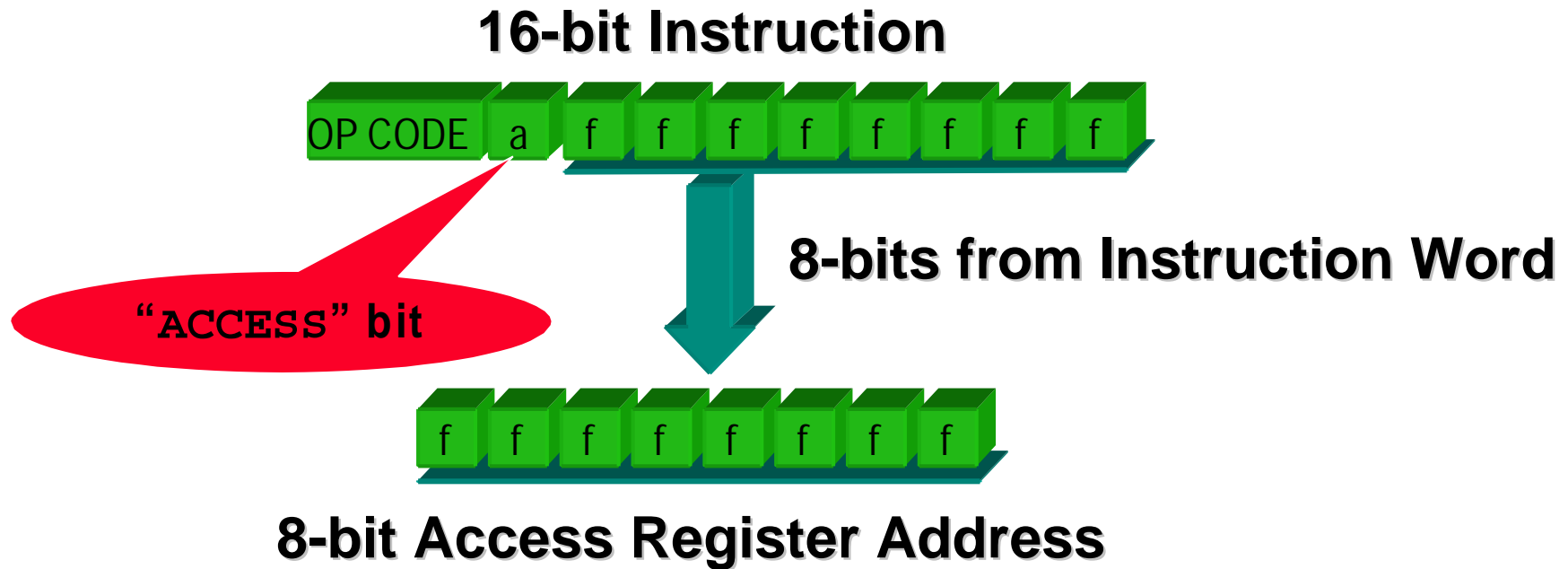




# PIC18 Architecture

## Accessing Access Bank

Instruction Format Example:



# PIC18 Architecture

## Program Memory Storage Scheme

Little-Endian Format

Instruction	Opcode	Memory	Address
...			00007h
MOVLW 55h	0E55h	55h	00008h
		0Eh	00009h
GOTO 06h	EF03h, F000h	03h	0000Ah
		EFh	0000Bh
		00h	0000Ch
		F0h	0000Dh

# **PIC18 Instructions**

## **Instruction Features**

Upward compatible with PIC16, PIC17,

16-bit Instruction width

Instruction fetches are 16-bit wide

Fetch and Execution is overlapped

Single Cycle 8 x 8 Multiply

Generates compact code

Most Instructions are Orthogonal

# **PIC18 Instructions**

## **Instruction Features**

**Most Instructions are Single Word**

71 Single Word; 4 Double Word

**Most Instructions are Single Cycle**

17 are Double Cycle

18 conditional branch/skips are 1, 2 (or 3)

**Register to Register transfer instruction**

**Powerful bit manipulation**

Available for entire data memory region



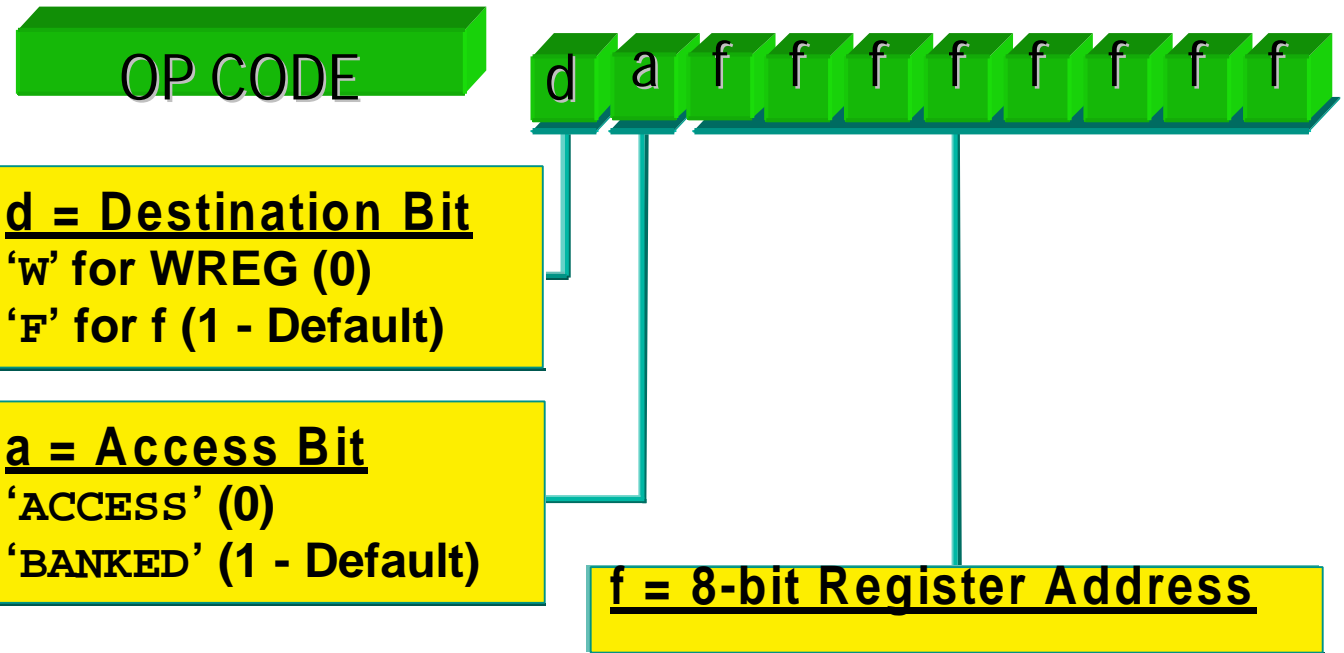
# PIC18 Instruction

## Byte Oriented Operation

### Byte-Oriented Operations

ADDWF	f [,d [,a]]
ADDWFC	f [,d [,a]]
ANDWF	f [,d [,a]]
CLRF	f [,a]
COMF	f [,d [,a]]
CPFSEQ	f [,a]
CPFSGT	f [,a]
CPFSLT	f [,a]
DECF	f [,d [,a]]
DECFSZ	f [,d [,a]]
DCFSNZ	f [,d [,a]]
INCF	f [,d [,a]]
INCFSZ	f [,d [,a]]
INFSNZ	f [,d [,a]]
IORWF	f [,d [,a]]
MOVF	f [,d [,a]]
MOVFF	fs, fd
MOVWF	f [,a]

### 16-bit Instruction for Byte Oriented Operations



### Example:

**ADDWF**      *f [,d [,a]]*

**ADDWF**      Count

**MOVFF** *fs, fd*

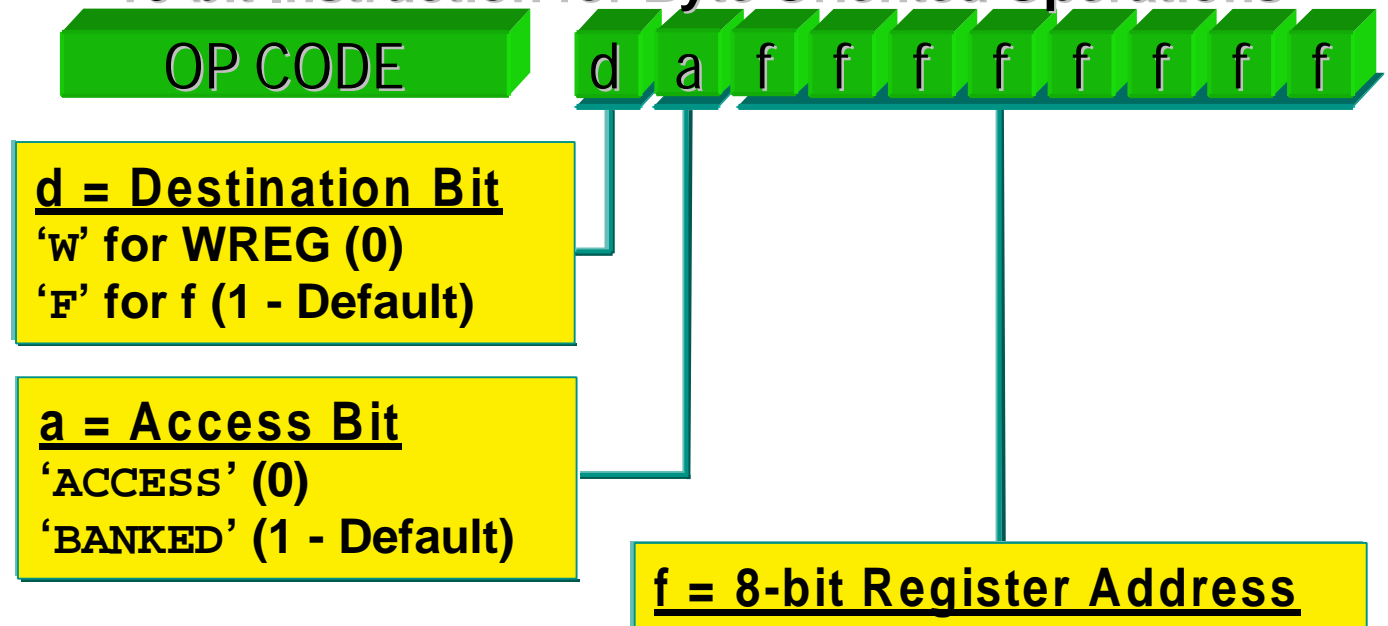
**MOVFF** Source, Dest

# PIC18 Instructions

## Byte Oriented Operation

Byte-Oriented Operations	
MULWF	f [,a]
NEGF	f [,a]
RLCF	f [,d [,a]]
RLNCF	f [,d [,a]]
RRCF	f [,d [,a]]
RRNCF	f [,d [,a]]
SETF	f [,a]
SUBFWB	f [,d [,a]]
SUBWF	f [,d [,a]]
SUBWFB	f [,d [,a]]
SWAPF	f [,d [,a]]
TSTFSZ	f [,a]
XORWF	f [,d [,a]]

### 16-bit Instruction for Byte Oriented Operations



**Example:**

*SUBWF*      *f [,d [,a]]*  
 SUBWF      Value, W

# PIC18 Instructions

## Byte Oriented Operation

Perform Multi-byte (4 byte) increment:  
"Count32++"

...

```
movlw      01h
```

```
addwf      Count32, F      ; Inc  LSB by '1'
```

```
clrf       WREG            ; Pass  the carry
```

```
addwfc     Count32+1, F    ; to LOW  MSB
```

```
addwfc     Count32+2, F    ; to HIGH LSB
```

```
addwfc     Count32+3, F    ; to HIGH MSB
```

...

# PIC18 Instructions

## Byte Oriented Operation

### Bit-Oriented Operations

BCF	f, b [,a]
BSF	f, b [,a]
BTG	f, b [,a]
BTFSC	f, b [,a]
BTFSS	f, b [,a]

### 16-bit Instruction for Bit Oriented Operations



b = 3-Bit Address  
(Bit Number)

a = Access Bit  
'ACCESS' (0)  
'BANKED' (1 - Default)

f = 8-bit Register Address

Example:

<i>BTFSC</i>	<i>f, b [,a]</i>
BTFSC	STATUS, C



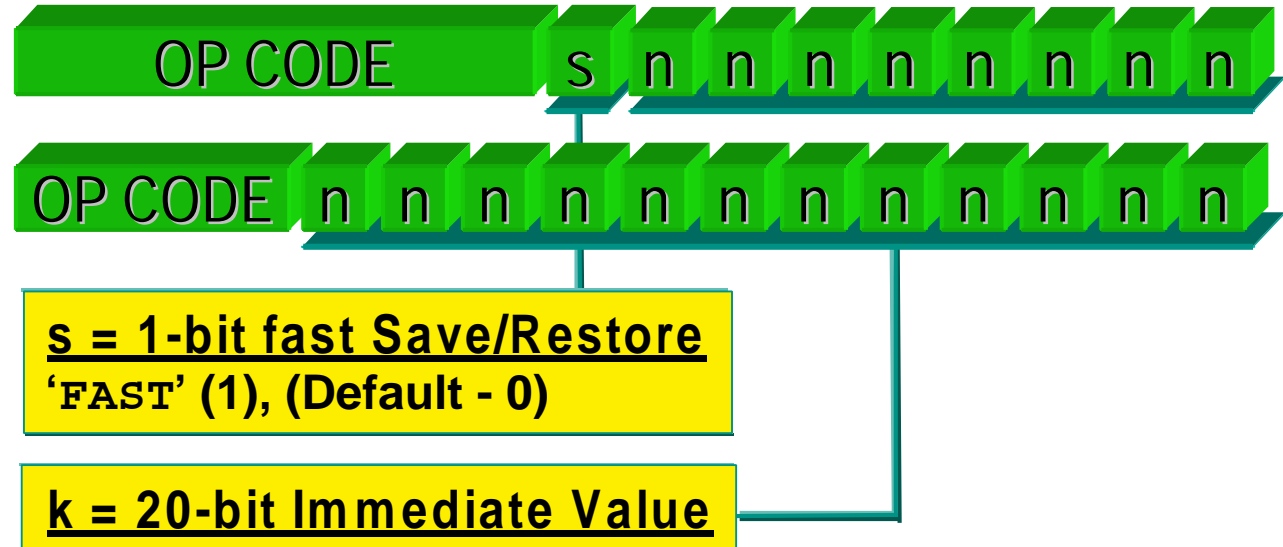
# PIC18 Instructions

## Control Operations

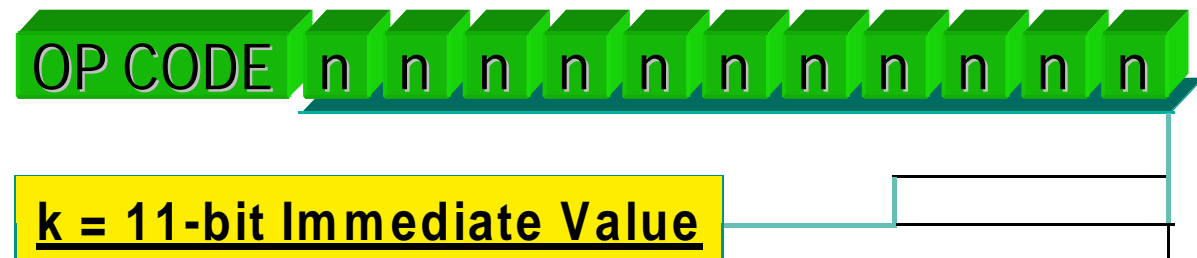
### Control Operations

BC	n
BN	n
BNC	n
BNN	n
BNOV	n
BNZ	n
BOV	n
BRA	n
BZ	n
CALL	n [,s]
GOTO	n
RCALL	n
RETFIE	[s]
RETURN	[s]

### 16-bit Instruction for CALL and GOTO



### 16-bit Instruction for RCALL and BRA



# PIC18 Instructions

## Control Operations

Control Operations	
BC	n
BN	n
BNC	n
BNN	n
BNOV	n
BNZ	n
BOV	n
BRA	n
BZ	n
CALL	n [,s]
GOTO	n
RCALL	n
RETFIE	[s]
RETURN	[s]

(Un)Conditional branches  
spans -128 through +127  
Instructions

**CALL** and **GOTO** contain full  
21-bit address

Provides Linear access to  
2MB

**RCALL** spans -1024 through  
1023 Instructions

# PIC18 Instructions

## Control Operations

### Control Operations

CLRWDT  
DAW  
NOP  
POP  
PUSH  
RESET  
SLEEP

16-bit Instruction for CLRWDT



OP CODE

16-bit Instruction for DAW



OP CODE

# PIC18 Instructions

## Control Operations = Example #1

### Handling Interrupt

```
org          00008h  
  
bra          HighISR  
  
...  
  
HighISR:  
  
...  
  
retfie      FAST  
  
...
```

# PIC18 Instructions

## Control Operations = Example #2

...

```
btfscl    PORTB, RB6    ; Is RB6 low ?  
bra       $-2           ; No.  Wait...  
...         ; Yes.
```

# PIC18 Instructions

## Literal Operations

### Literal Operations

ADDLW	k
ANDLW	k
IORLW	k
LFSR	f, k
MOVLB	k
MOVLW	k
MULLW	k
RETLW	k
SUBLW	k
XORLW	k

Example:

*MOVLW*      *k*  
*MOVLW*      5Ah

### 16-bit Instruction for LFSR



f = 2-bit FSR Selector  
 FSR0, FSR1 or FSR2

k = 8-bit Immediate Value

### 16-bit Instruction for Other Literal Operations



k = 8-bit Immediate Value

*LFSR*      *f, k*  
*LFSR*      FSR0, 400h

# PIC18 Instructions

## Literal Operations = Example

### Immediate Operation

...

`movlw 55h`

`movwf PORTB`

...

### Indirect Operation

...

`lfsr FSR0, 400h`

`movwf INDF0 ; *FSR0`

`movwf POSTINC0 ; *FSR0++`

`movwf POSTDEC0 ; *FSR0--`

`movwf PREINC0 ; *++FSR0`

`movwf PLUSW0 ; FSR0[WREG]`

# PIC18 Instructions

## Data Program Operation

### Control Operations

TBLRD\*  
TBLRD\*+  
TBLRD\*-  
TBLRD+\*  
TBLWT\*  
TBLWT\*+  
TBLWT\*-  
TBLWT+\*

16-bit Instruction for TBLRD\*/TBLWT\*

OP CODE

Example:

*TBLRD\**  
*TBLRD\*+*



# PIC18 Instructions

**Data**    Program Operation = Example

```
movlw      upper(LookUpTable)    ; Load look-up
movwf      TBLPTRU                ; table
movlw      high(LookupTable)      ; address
movwf      TBLPTRH
movlw      low(LookupTable)
movwf      TBLPTL
tblrd*+                                ; Read it.
```



# **PIC16C/FXXX to PIC18FXXXX Source Code Conversion Tips**



# **PIC16F/CXXX to PIC18FXXXX Assembly Compatibility**

C source code on most PIC16C/FXXX platforms will directly port to PIC18FXXXX

Compatible Assembly code source except:

- Absolute constants used for program memory

- Computed GOTO (addwf PCL,F)

- RAM requirements above 256 bytes are selected by BSR not RP0 and RP1 bits

- FSR is 12 bits wide, also includes auto increment

Double check immediate constants when initializing peripherals



# Code Conversion Tip

## Data Memory Access

```
bsf STATUS,RP0
```

```
bcf STATUS,RP0
```

These instructions can be ignored because bits 7,6,5 in STATUS register are unused

For devices with less than 256 bytes of RAM, it is not necessary to be concerned with RAM locations. Why is this the case?

Most memory accesses can be done in Access Bank

Assembler will automatically select “a” bit when applicable

Address locations now use 12-bit values.

Set the BSR if you need to.



# Code Conversion Tip

## PCLATU, PCLATH

CALL, GOTO instructions write directly to the program counter.

Operations to the PC latches before a CALL or GOTO will be ignored.

Program addresses are now BYTE addresses

If labels are used, then any moves to PCLATH are still OK

If absolute values are used, then they must be modified

Example      Goto \$+1      ; PIC16CXXX  
                 Goto \$+2      ; PIC18FXXX



# Code Conversion Tip

## Conditional GOTO, Tables

```
Code    movlw    HIGH Table ;(Table must be a label)
        movwf    PCLATH
        movlw    offset
        call     Table

.....

Table   addwf    PCL
        retlw    'A'
        retlw    'B'
```

.....

What's wrong with this code?

# Code Conversion Tip

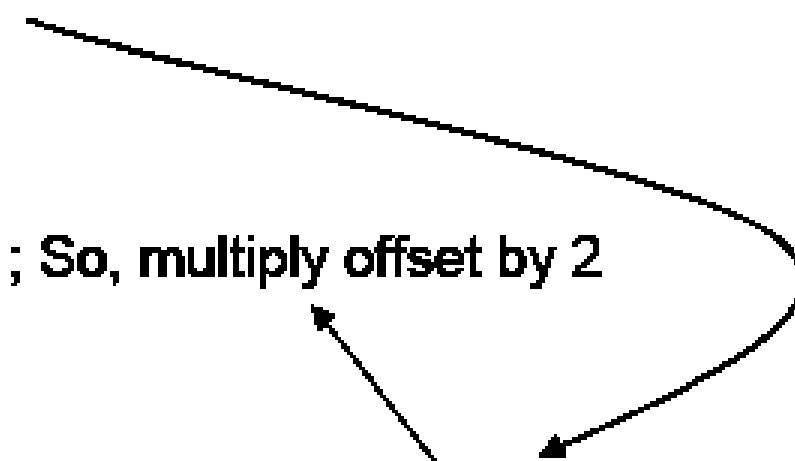
## Conditional GOTO, Tables

```
Code    movlw    HIGH Table
        movwf    PCLATH
        bcf      STATUS,C
        rlnsf    offset,W    ; So, multiply offset by 2
        call     Table
```

.....

```
Table   addwf    PCL          ; On PIC18, this is a BYTE address
        retlw    'A'
        retlw    'B'
```

.....



# Code Conversion Tip

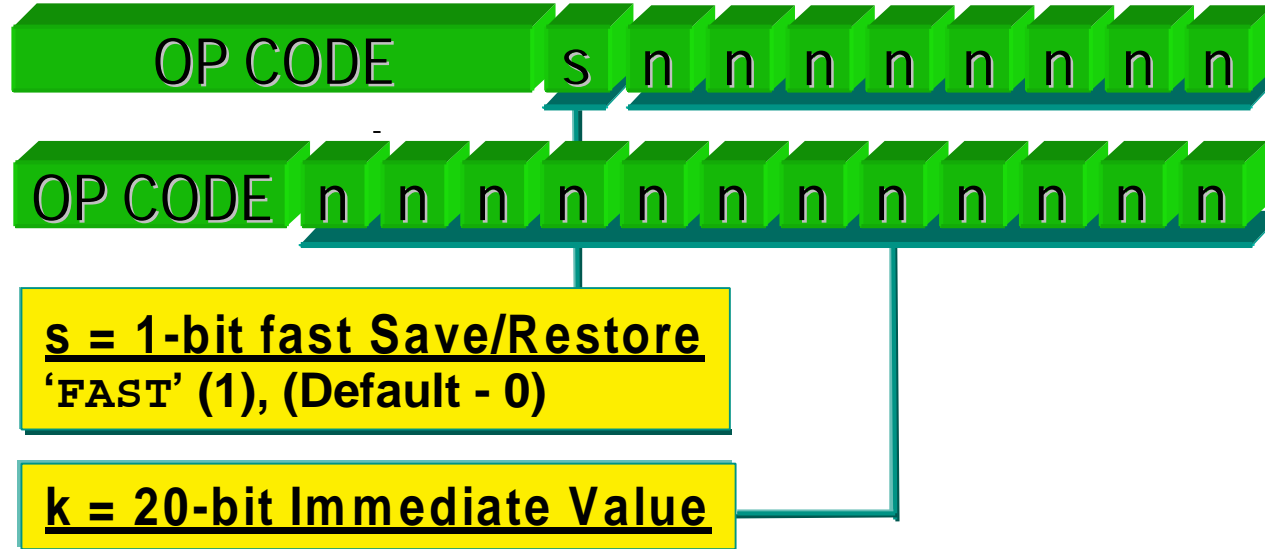
- Be particularly careful about loading registers  
    **movlw   B'00100110'**  
    **movwf   register**
- Most registers are compatible, but there are differences
- Use of symbolic bit names is safest

**d**

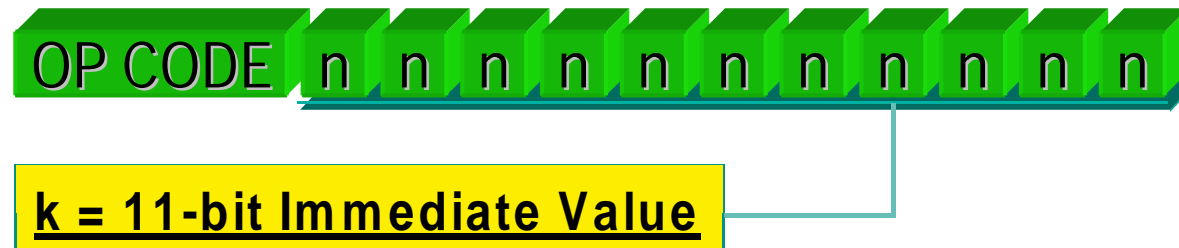


# Conversion Tip

## 16-bit Instruction for CALL and GOTO



## 16-bit Instruction for RCALL and BRA





# **Appendix C:**

## **PIC18FXXX**

### **Flash**

### **Programming**

### **Tips**

---

# **PIC18F FLASH**

## **Program Memory Reads and Writes**

**READs performed on bytes**

**Can READ entire user Program Memory of up to 2M plus:**

User ID locations 200000h-200007h

CONFIG registers 300000h-30000Dh

Device ID registers 3FFFFEh,3FFFFFFh

**To READ Program Memory:**

Load TBLPTRU,TBLPTRH,TBLPTRL

Execute one of the TBLRDs

TBLRD\*, TBLRD\*+, TBLRD+\*, TBLRD\*-

result in TABLAT

# PIC18F ARCHITECTURE

## 18F Addressable Memory is divided into:

### USER MEMORY:

Up to 128 Kbytes  
internal

Up to 2 Mbytes external

### USER IDs:

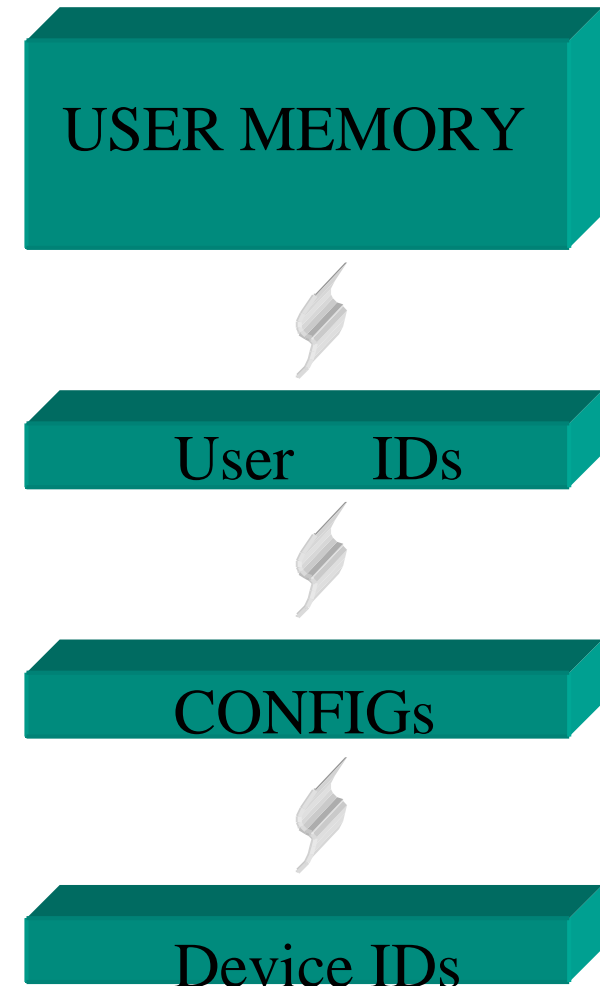
8 modifiable bytes

### CONFIGs:

Device settings, code  
protects, etc

### DEVICE IDs:

Part and rev. signature





# PIC18F FLASH

## Program Memory Reads and Writes

### **ERASING User memory (USER MODE):**

Performed on 64 bytes (32 words)

Load TBLPTRU,TBLPTRH,TBLPTRL

TBLPTR 6 LSBs are don't cares

Configure EECON1

Disable interrupts

Perform programming sequence

Start ERASE

Internally timed, NO CODE EXECUTION

Re-enable interrupts



# PIC18F FLASH

## Program Memory Reads and Writes

### **WRITES to User memory (USER MODE):**

Performed on 8 bytes (4 words)

Load TBLPTRU,TBLPTRH,TBLPTRL

Load 8 bytes into write buffers by 8 table write instructions

TBLWT\*,TBLWT\*+,TBLWT\*-,TBLWT+\*

Configure EECON1

Disable interrupts

Perform programming sequence

Start WRITE

Internally timed, NO CODE EXECUTION

Re-enable interrupts



# PIC18F ARCHITECTURE

## 18F Internal User Memory is separated by:

### PANELS:

Define internal cell grouping boundaries

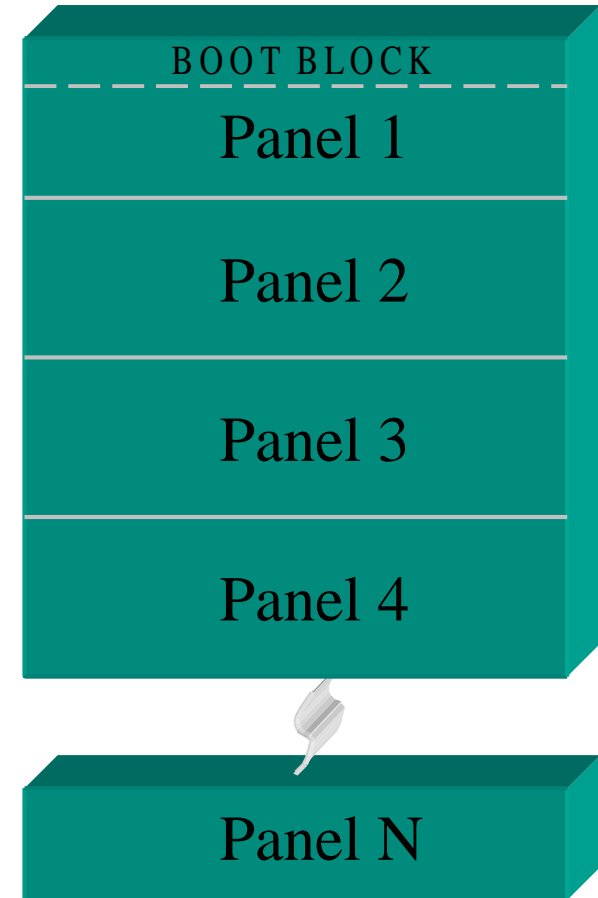
Always 8 Kbytes  
(4 Kwords)

### BLOCKS:

Define Code Protect boundaries

Minimum 512 bytes

Could be 16 Kbytes  
(18F8720)

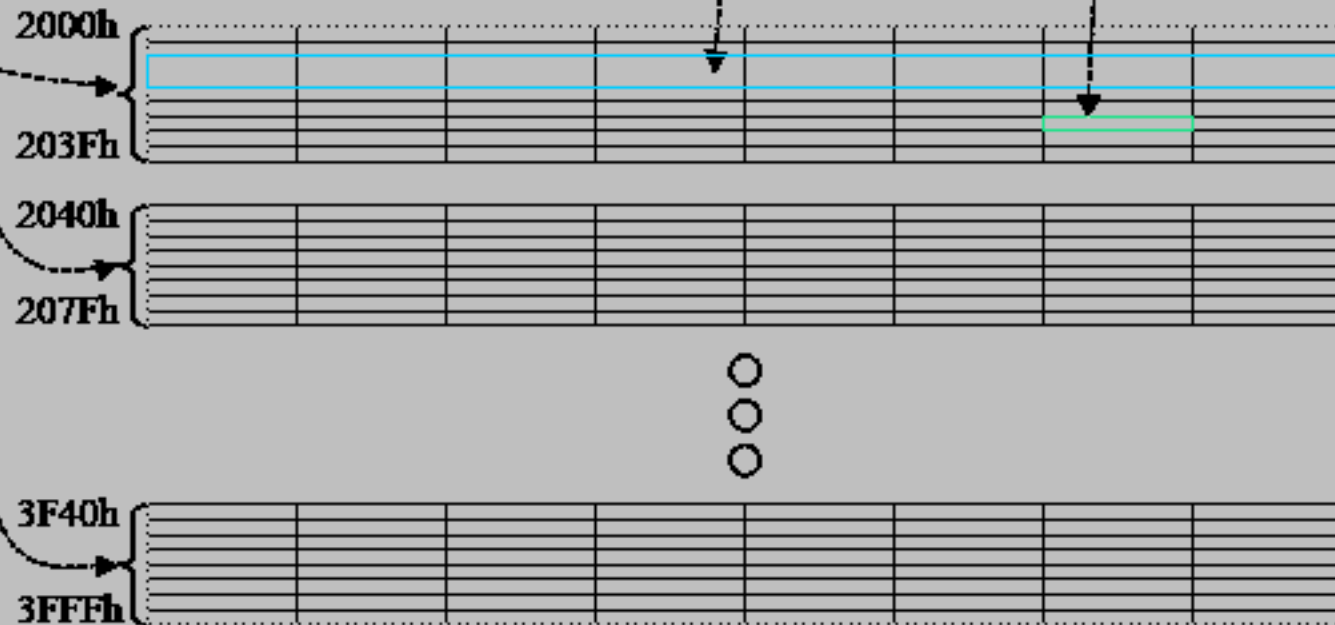


# PIC18F ARCHITECTURE

- **18F Internal User Memory Panel:**

- **SINGLE PANEL (8K bytes):**

ERASE  
Boundaries  
(64 bytes)



WRITE  
Boundary  
(8 bytes)

READ  
Boundary  
(1 byte)



# PIC18F ARCHITECTURE

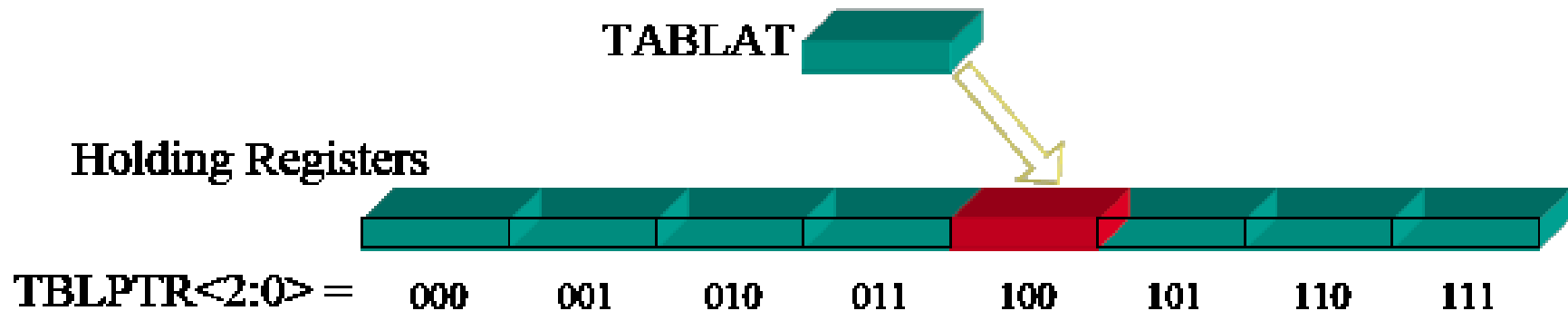
## 18F Holding Registers:

USER - 8 bytes for *entire code memory (single-panel programming)*

ICSP programming - 8 bytes for *each panel (multi-panel use)*

Loaded by TBLWT\* instruction.

TBLWT\* instruction moves contents of TABLAT to a holding register. The last three bits of TBLPTR determine which holding register. TBLPTR<20:3> are don't cares.



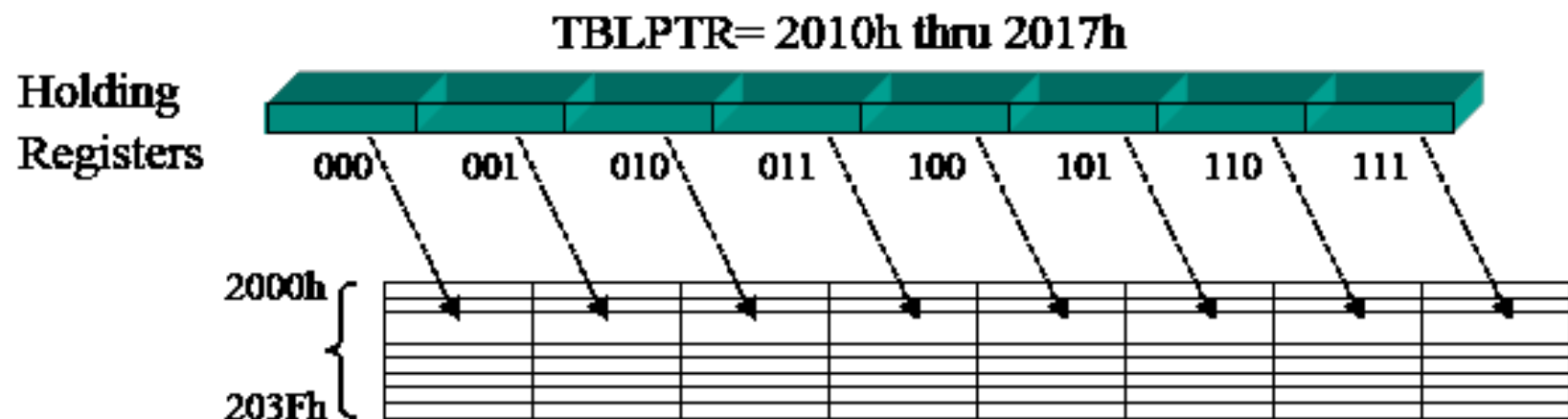
# PIC18F ARCHITECTURE

## 18F Holding Registers (cont):

Writes to Program Memory (details later)

TBLPTR <20:3> determines which 8 bytes of internal user memory Holding Registers will write to

In example, TBLPTR could be in range of 2010h - 2017h, and the holding registers will write same 8 bytes.



# PIC18F EECON1

## EECON1 REGISTER

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS		FREE	WRERR	WREN	WR	RD
bit7	6	5	4	3	2	1	bit0

- bit 7: **EEPGD**: FLASH Program or Data EEPROM Memory Select Bit  
1 = Access Program Flash memory  
0 = Access Data EEPROM memory
- bit 6: **CFGS**: FLASH Program/Data EE or Configuration Select bit  
1 = Access Configuration registers  
0 = Access Program Flash or Data EEPROM memory
- bit 5: **Unimplemented**: Read as '0'
- bit 4: **FREE**: FLASH Row Erase Enable bit  
1 = Erase the program memory row addressed by TBLPTR on next WR command (cleared on erase completion)  
0 = Perform write only

# PIC18F EECON1

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS		FREE	WRERR	WREN	WR	RD
bit 7	6	5	4	3	2	1	bit 0

- bit 3: WRERR: FLASH and EEPROM Error Flag Bit**  
 1 = A write operation is prematurely terminated (RESET)  
 0 = The write operation completed  
 Note: When WRERR occurs, EEGD and CFGS are not cleared.
- bit 2: WREN: FLASH and EEPROM Write Enable Bit**  
 1 = Allows write cycles  
 0 = Inhibits erases or writes to FLASH and EEPROM
- bit 1: WR: Write Control Bit**  
 1 = Initiates FLASH erase or write or EEPROM erase/write  
 0 = The write or erase operation is complete
- bit 0: RD: Read Control Bit**  
 1 = Initiates an EEPROM read  
 0 = Does not initiate an EEPROM read

# PIC18F REQUIRED SEQUENCE

**WRITE and ERASE of internal user memory require six instructions as shown below:**

**Makes accidental writes and erases highly improbable.**

**First instruction following the WR bit set must be NOP. This instruction was pre-fetched and must be discarded.**

<code>movlw</code>	<code>55h</code>
<code>movwf</code>	<code>EECON2</code>
<code>movlw</code>	<code>AAh</code>
<code>movwf</code>	<code>EECON2</code>
<code>bsf</code>	<code>EECON1,WR</code>
<code>nop</code>	

# PIC18F READ

**READs performed on bytes**

**Can READ entire user Program Memory of up to 2M plus:**

User ID locations 200000h-200007h

CONFIG registers 300000h-30000Dh

Device ID registers 3FFFFFFEh,3FFFFFFh

**To READ Program Memory:**

Load TBLPTRU,TBLPTRH,TBLPTRL

Execute one of the TBLRDs

TBLRD\*, TBLRD\*+, TBLRD+\*, TBLRD\*-

result in TABLAT *next instruction cycle*

# PIC18F READ

## READ Code example:

```
; Load Table Pointer
    movlw    UPPER(TBL_ADDR)
    movwf    TBLPTRU
    movlw    HIGH(TBL_ADDR)
    movwf    TBLPTRH
    movlw    LOW(TBL_ADDR)
    movwf    TBLPTRL
    tblrd*
    movff    TABLAT, INDF0
```

# PIC18F ERASE

## ERASING User memory:

Performed on 64 bytes (32 words)

Load TBLPTRU,TBLPTRH,TBLPTRL

Configure EECON1

Disable interrupts

Perform programming sequence

Start erase (Set WR bit)

*Internally timed 2 mS (typical)*

PROCESSOR 'HALTS', NO CODE EXECUTION

TBLPTR 6 LSBs are don't cares



Re-enable interrupts



# PIC18F ERASE

## ERASE User Memory Code Example:

```
; Load Table Pointer
    bsf      EECON1,EEPGD
    bcf      EECON1,CFGFS
    bsf      EECON1,WREN
    bsf      EECON1,FREE
    bcf      INTCON,GIE
    movlw    55h
    movwf    EECON2
    movlw    AAh
    movwf    EECON2
    bsf      EECON1,WR
    nop
    bsf      INTCON,GIE
    bcf      EECON1,WREN
```



# PIC18F WRITE

## **WRITES to User memory:**

Performed on 8 bytes (4 words)

Load TBLPTRU,TBLPTRH,TBLPTRL

Load 8 bytes into write buffers by 8 table write instructions

TBLWT\*,TBLWT\*+,TBLWT\*-,TBLWT+\*

Configure EECON1

Disable interrupts

Perform programming sequence



# PIC18F WRITE

## WRITES to User memory (cont):

Start write (set WR bit)

*Internally timed 2 mS*

PROCESSOR 'HALTS', NO CODE EXECUTION

TBLPTR 3 LSBs are don't cares



Re-enable interrupts



McCOMB  
UNIVERSITY  
OF CALIFORNIA

# PIC18F WRITE

**What's wrong with this?**

```
; GIVEN:
; FSR0    ->  points to first of 8 bytes of a buffer
;           that will be written
; TBLPTR  ->  points to first byte of 8 byte block in
;           internal user memory
; COUNTER =   8
; WRITEIT =   Correct programming macro

WRITE_TO_HREGS
    movff  POSTINC0,TABLAT        ; load Holding Regs
    TBLWT*+                       ;
    decfsz COUNTER                ;
    bra    WRITE_TO_HREGS        ;
    WRITEIT                       ; Write Holding Regs
                                   ; to user memory
```

**After the last TBLWT\*+, TBLPTR would be pointing to the next 8 bytes block !**



# **Appendix D:**

## **PIC18FXXX**

### **Peripheral**

### **Configuration**

### **Spreadsheet**

# **Spreadsheet Basics**

*PIC18Fxxx Peripheral Configuration.xls*

*Spreadsheet based on Microsoft Excel*

*Calculates period, baud rate, operating frequency for the following peripherals:*

*TMR0, TMR1, TMR2 and TMR3 period*

*PWM / CCP0 through PWM / CCP4 frequency*

*A/D conversion period*

*UART Baud Rate*

*Contains reference map for Special Function Registers, Pinouts and Instruction Set*



MICROCHIP

# PWM Configuration Example

Microsoft Excel - PIC18Fxxx Peripheral Configuration.xls

File Edit View Insert Format Tools Data Window Help

Arial 10 B I U Insert Hyperlink

K5

Calculate the Period given PR2										Calculate the Period Register Value needed given a known PWM period					
Place cursor here for CCP1CON bit definitions	Operating Freq. (from Main Page)	Freq. OVERRIDE (this page only)	Frequency Used (this page only)	TMR2 Prescale Setting (1, 4, or 16)	PR2 Value	Period (us)	Frequency	Bits of PWM Resolution	Period Desired (us)	Equivalent frequency	Calculated PR2 Value	Percent Error PR2 Rounded down	PR2 Rounded Down Frequency	Percent Error PR2 Rounded Up	PR2 Rounded Up Frequency
ENTER DATA IN GREEN CELLS ONLY	40,000,000	33,333,333	33,333,333	1	0	0.1	10000000	2.0	11	90,909	90.67	-0.7%	91,575	0.4%	90,580
					1	0.2	5000000	3.0							
					2	0.3	3333333	3.6							
					3	0.4	2500000	4.0							
					4	0.5	2000000	4.3							
					5	0.6	1666667	4.6							
					6	0.7	1428571	4.8							
					7	0.8	1250000	5.0							
					8	0.9	1111111	5.2							
					9	1	1000000	5.3							
					10	1.1	909091	5.5							
					11	1.2	833333	5.6							
					12	1.3	769231	5.7							
					13	1.4	714286	5.8							
					14	1.5	666667	5.9							
					15	1.6	625000	6.0							
					16	1.7	588235	6.1							
					17	1.8	555556	6.2							
					18	1.9	526316	6.2							
					19	2	500000	6.3							
					20	2.1	476190	6.4							
					21	2.2	454545	6.5							
					22	2.3	434783	6.5							
					23	2.4	416667	6.6							
					24	2.5	400000	6.6							
					25	2.6	384615	6.7							

**FIGURE 14-3: SIMPLIFIED PWM BLOCK DIAGRAM**

**Note:** 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

THIS TRANSMITTAL AND ACCOMPANYING DOCUMENTS ARE INTENDED ONLY AS SUGGESTION. No representation or warranty is given and no liability is assumed by Microchip Technology Inc. with respect to accuracy or use of such information, or infringement of patents arising from such use or otherwise..

TMR1 TMR2 TMR3 PWM A-D USART Reference - SFR Map Reference - Pinouts Reference - Instruction set

Ready