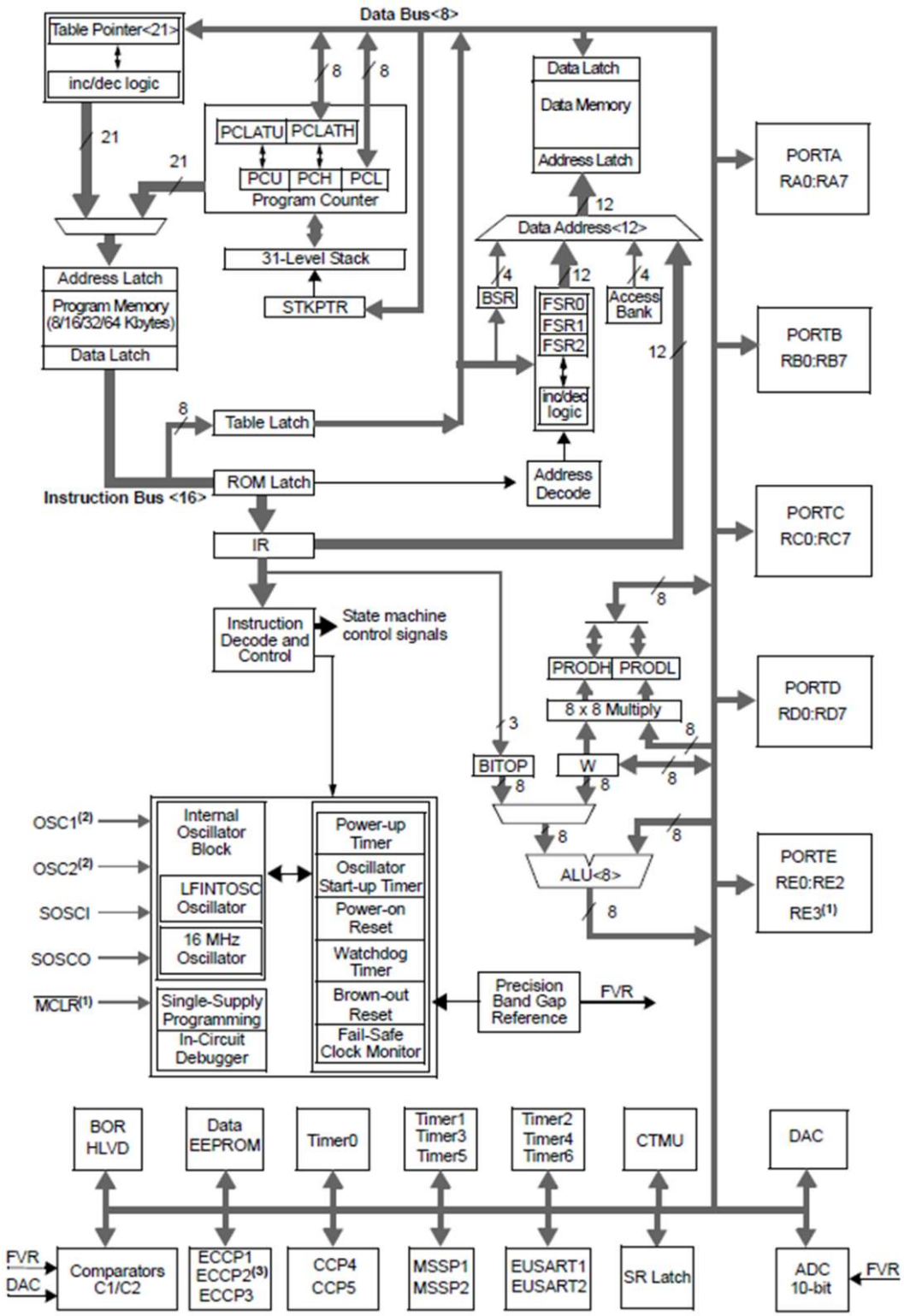
FIB UPC UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

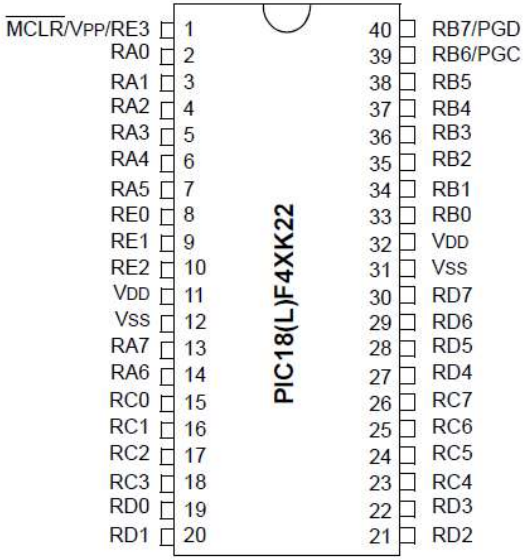
Introduction to PIC18 architecture

Dpt. Enginyeria de Sistemes, Automàtica i Informàtica Industrial

PIC18 Architecture
18F45K22



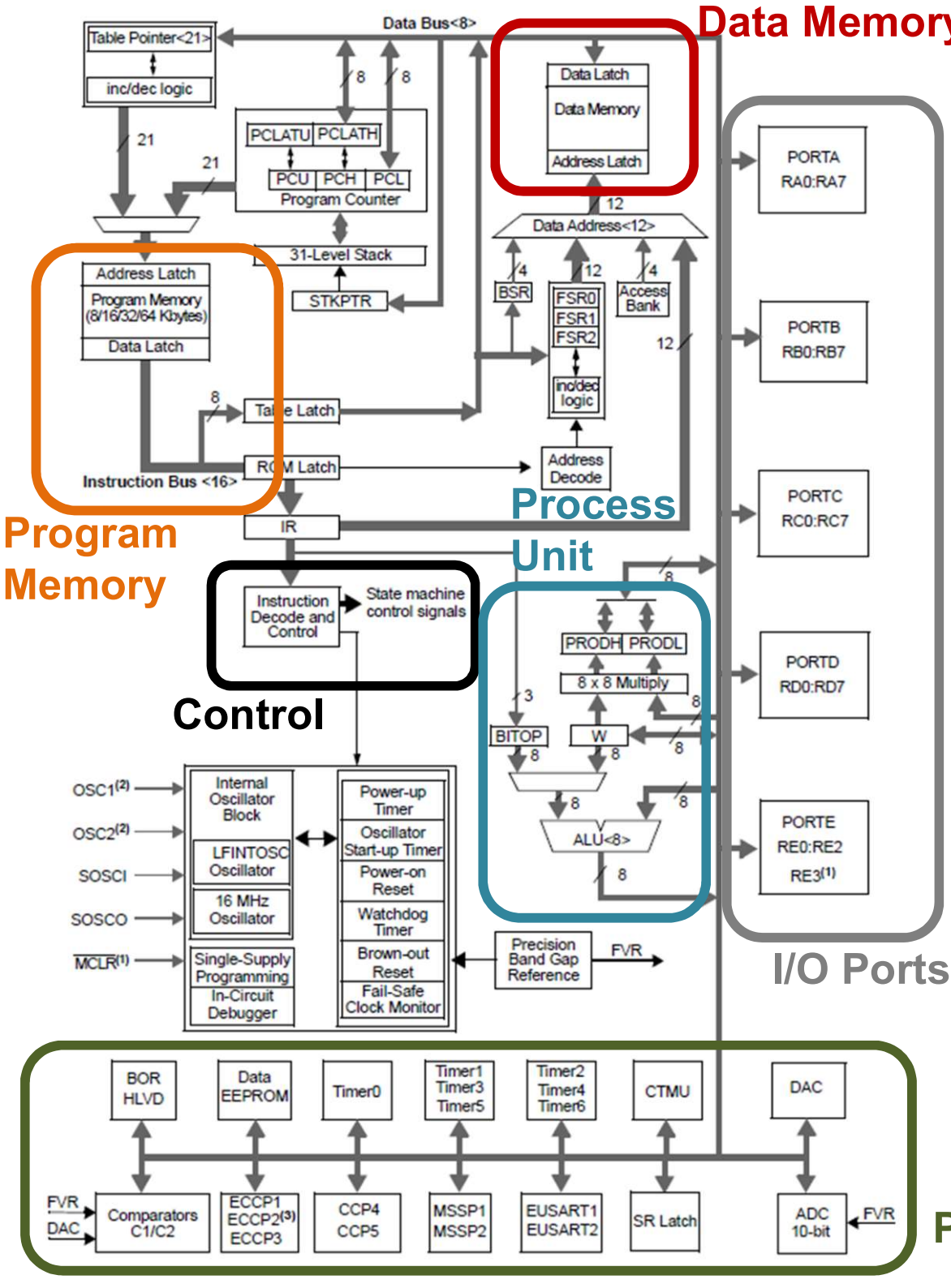
40-PIN PDIP DIAGRAM



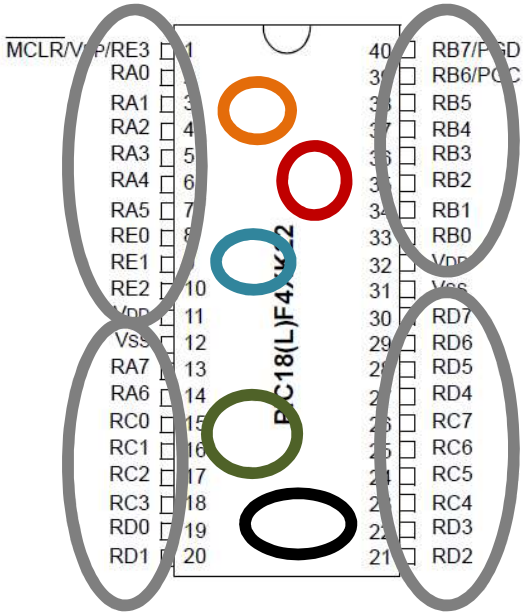
chip pinout:

which pin is connected to each function?

PIC18 Architecture
18F45K22



40-PIN PDIP DIAGRAM



System on chip

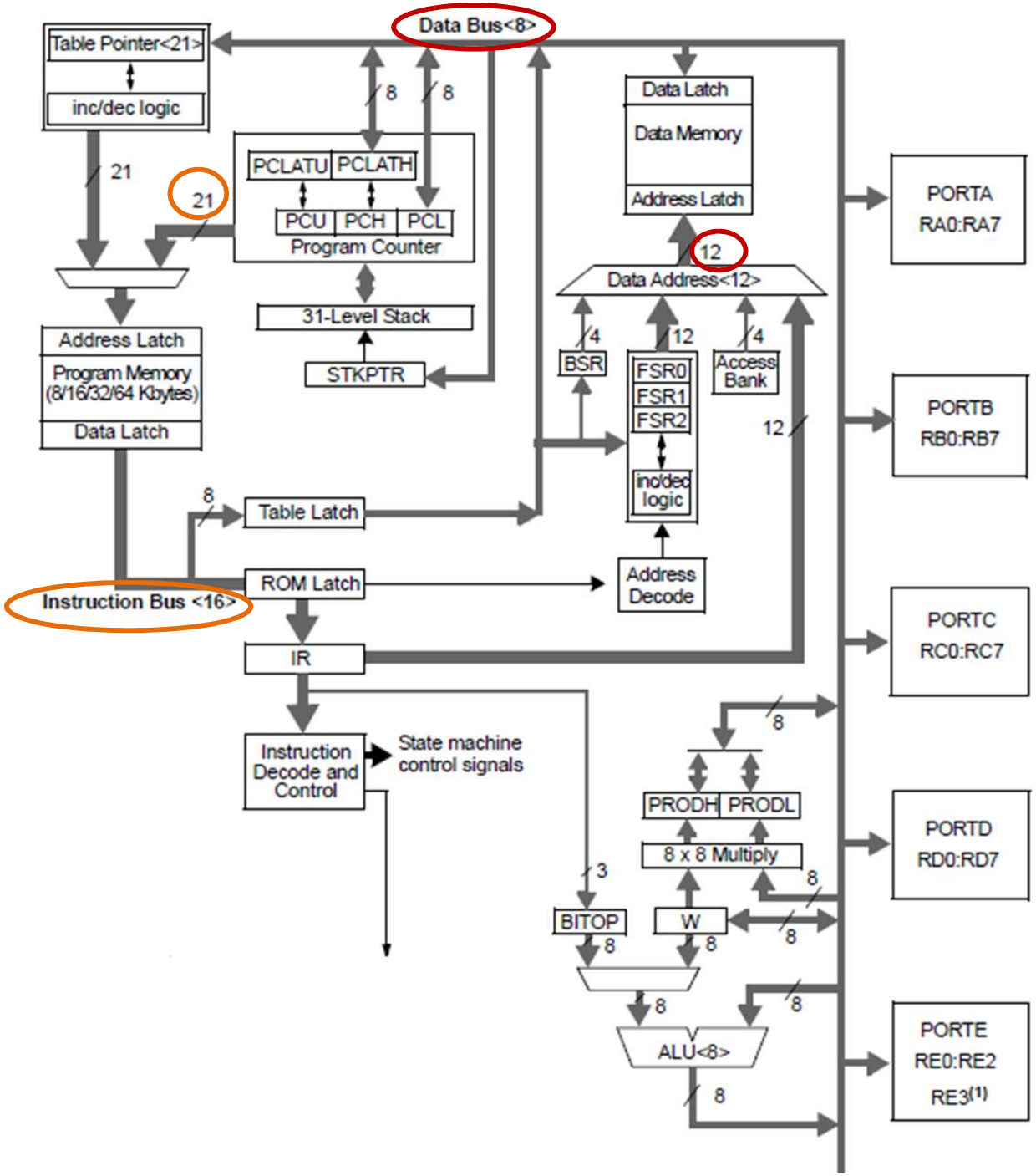
Laboratori: 18F45K22

TABLE 1-1: DEVICE FEATURES

Features	PIC18F23K22 PIC18LF23K22	PIC18F24K22 PIC18LF24K22	PIC18F25K22 PIC18(L)F25K22	PIC18F26K22 PIC18LF26K22	PIC18F43K22 PIC18LF43K22	PIC18F44K22 PIC18LF44K22	PIC18F45K22 PIC18LF45K22	PIC18F46K22 PIC18LF46K22
Program Memory (Bytes)	8192	16384	32768	65536	8192	16384	32768	65536
Program Memory (Instructions)	4096	8192	16384	32768	4096	8192	16384	32768
Data Memory (Bytes)	512	768	1536	3896	512	768	1536	3896
Data EEPROM Memory (Bytes)	256	256	256	1024	256	256	256	1024
I/O Ports	A, B, C, E[1]	A, B, C, E[1]	A, B, C, E[1]	A, B, C, E[1]	A, B, C, D, E	A, B, C, D, E	A, B, C, D, E	A, B, C, D, E
Capture/Compare/PWM Modules (CCP)	2	2	2	2	2	2	2	2
Enhanced CCP Modules (ECCP) - Half Bridge	2	2	2	2	1	1	1	1
Enhanced CCP Modules (ECCP) - Full Bridge	1	1	1	1	2	2	2	2
10-bit Analog-to-Digital Module (ADC)	2 Internal 17 Input	2 Internal 17 Input	2 Internal 17 Input	2 Internal 17 Input	2 Internal 28 Input	2 Internal 28 Input	2 Internal 28 Input	2 Internal 28 Input
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN 28-pin UQFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN 28-pin UQFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 40-pin UQFN 44-pin QFN 44-pin TQFP	40-pin PDIP 40-pin UQFN 44-pin QFN 44-pin TQFP	40-pin PDIP 40-pin UQFN 44-pin QFN 44-pin TQFP	40-pin PDIP 40-pin UQFN 44-pin QFN 44-pin TQFP
Interrupt Sources	33							
Timers (16-bit)	4							
Serial Communications	2 MSSP, 2 EUSART							
SR Latch	Yes							
Charge Time Measurement Unit Module (CTMU)	Yes							
Programmable High/Low-Voltage Detect (HLVD)	Yes							
Programmable Brown-out Reset (BOR)	Yes							
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Overflow, Stack Underflow (PWRT, OST), MCLR, WDT							
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled							
Operating Frequency	DC - 64 MHz							

Note 1: PORTE contains the single RE3 read-only bit.

PIC18 Architecture. Data/Program buses



Program Memory Bus:

21 bit address bus
16 bit wide instruction

(32KB out of 2MB max)

Data Memory Bus:

12 bit address bus
8 bit wide data

(1536B out of 4096B max)

Separation of Data Memory and Program Memory

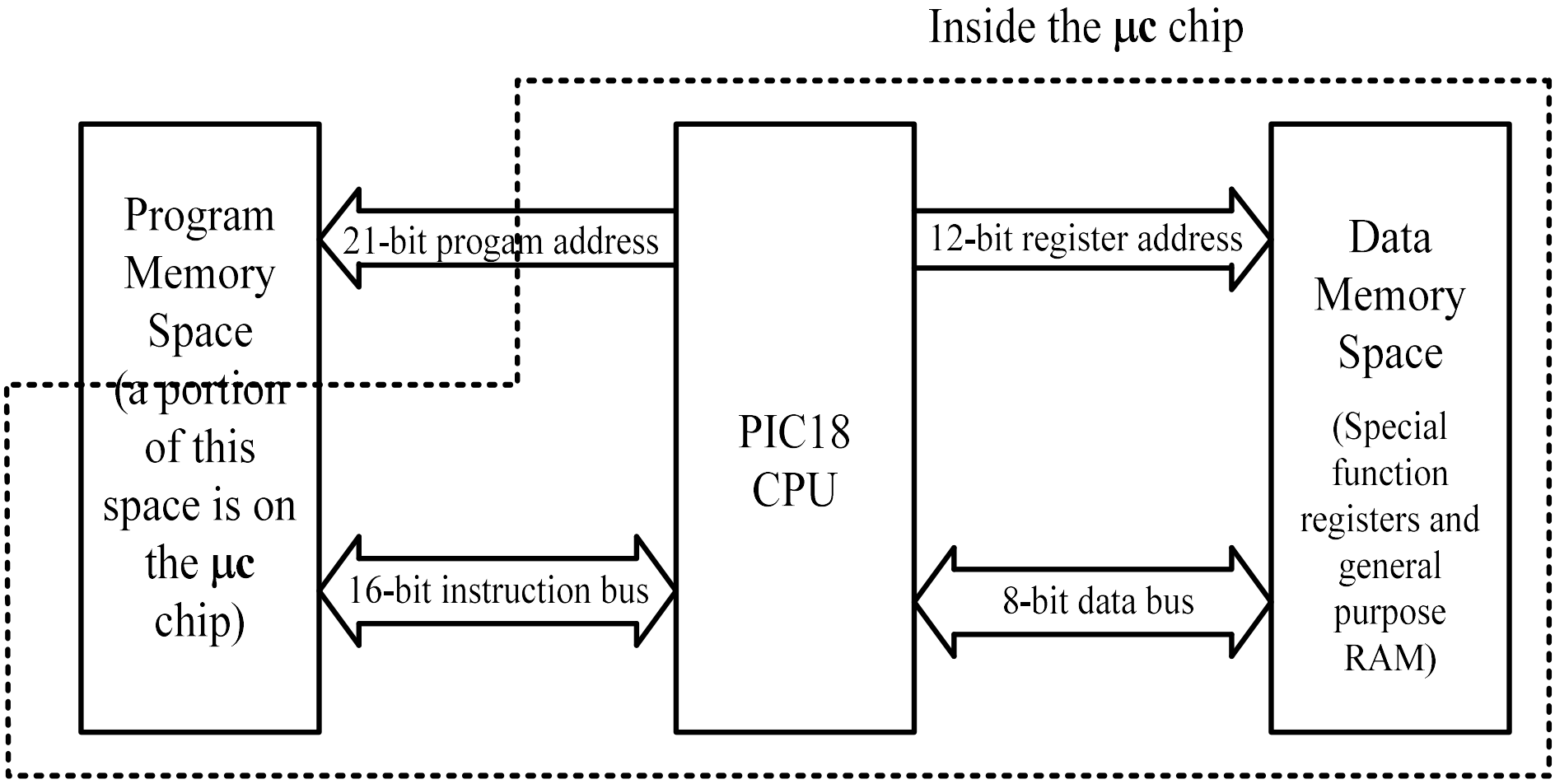


Figure 1.3 The PIC18 memory spaces

Types of semiconductor memory

Main classification

- **Volatile:** loses data when power is turned off

  Typical volatile: **Random-Access Memory (RAM)**. It's a Read/Write memory.

- **Non-volatile:** keeps stored data when power is turned off

  Typical non-volatile: **Read-Only Memory (ROM)**. Can only be read but not written by the processor

Types of RAM:

- **Dynamic random-access memory (DRAM):** need periodic refresh
- **Static random-access memory (SRAM):** no periodic refresh is required

Types of ROM:

- **Mask-programmed read-only memory (MROM):** programmed when being manufactured
- **Programmable read-only memory (PROM):** can be programmed by the end user

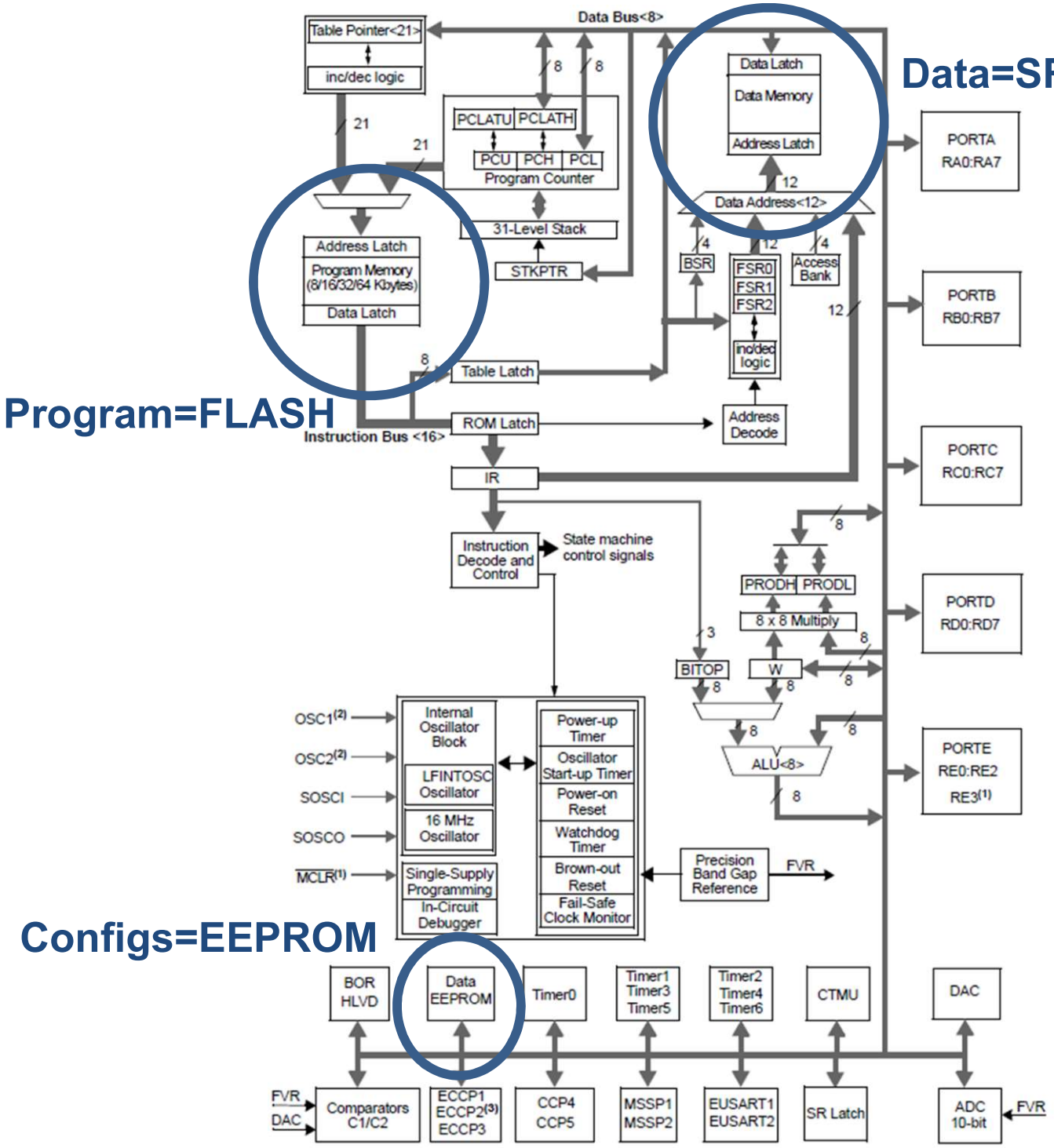
Types of semiconductor memory

Types of ROM:

- **Erasable programmable ROM (EPROM)**
  - electrically programmable many times
  - erased by ultraviolet light (through a window)
  - erasable in bulk (whole chip in one erasure operation)

- **Electrically erasable programmable ROM (EEPROM)**
  - electrically programmable many times
  - electrically erasable many times
  - can be erased one location, one row, or whole chip in one operation

- **Flash memory**
  - electrically programmable many times
  - electrically erasable many times
  - can only be erased in bulk (either a block or the whole chip)



Memory technologies on PIC18F

Program Memory Organization

- The program counter (PC) is 21-bit long, which enables the user program to access up to 2 MB of program memory.

- The PIC18 has a 31-entry return address stack to hold the return address for subroutine call.

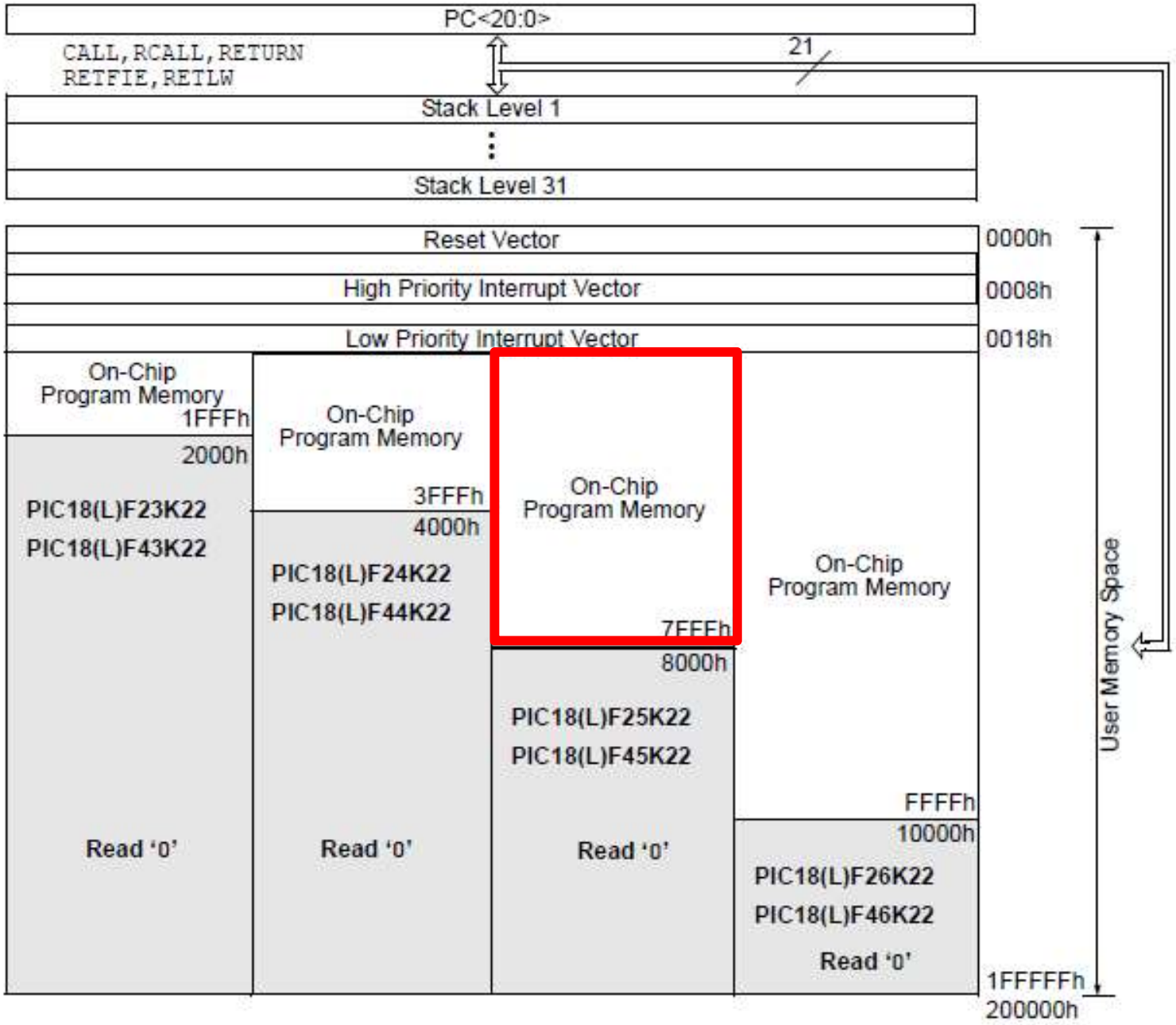
- After power-on, the PIC18 starts to execute instructions from address 0.

- The location at address 0x08 is reserved for high-priority interrupt service routine.

- The location at address 0x18 is reserved for low-priority interrupt service routine.

- Up to 32KB of program memory is inside the MCU chip. (Accessing a location beyond the upper boundary of the physically implemented memory will return all ‘0’s)

Program Memory Organization



Instructions in program memory

- The program memory is addressed in bytes.
- Instructions are stored as **two bytes or four bytes** in program memory.
- The Least Significant Byte of an instruction word is always stored in a program memory location with an even address (LSb = 0).
- To maintain alignment with instruction boundaries, the PC increments in steps of 2 and the LSb will always read '0'.

Program Memory
Byte Locations →

Instruction 1: MOVLW 055h
Instruction 2: GOTO 0006h

Instruction 3: MOVFF 123h, 456h

LSb = 1	LSb = 0	Word Address ↓
		000000h
		000002h
		000004h
		000006h
0Eh	55h	000008h
EFh	03h	00000Ah
F0h	00h	00000Ch
C1h	23h	00000Eh
F4h	56h	000010h
		000012h
		000014h

2-Word instructions

- The standard PIC18 instruction set has four two-word instructions:
  CALL, MOVFF, GOTO and LFSR.
- In all cases, the second word of the instructions always has '1111' as its four Most Significant bits
- If the instruction is executed in proper sequence, immediately after the first word, the data in the second word is accessed and used by the instruction sequence. If the first word is skipped for some reason and the second word is executed by itself, a NOP is executed instead

CASE 1:

Object Code	Source Code
0110 0110 0000 0000	TSTFSZ REG1 ; is RAM location 0?
1100 0001 0010 0011	MOVFF REG1, REG2 ; Yes,skip this word
1111 0100 0101 0110	; Execute this word as a NOP
0010 0100 0000 0000	ADDWF REG3 ; continue code

CASE 2:

Object Code	Source Code
0110 0110 0000 0000	TSTFSZ REG1 ; is RAM location 0?
1100 0001 0010 0011	MOVFF REG1, REG2 ; No, execute this word
1111 0100 0101 0110	; 2nd word of instruction
0010 0100 0000 0000	ADDWF REG3 ; continue code

PIC18 Data Memory

- Implemented in SRAM and consists of **general-purpose registers** (GPR) and **special-function registers** (SFR). Both are referred to as data registers.

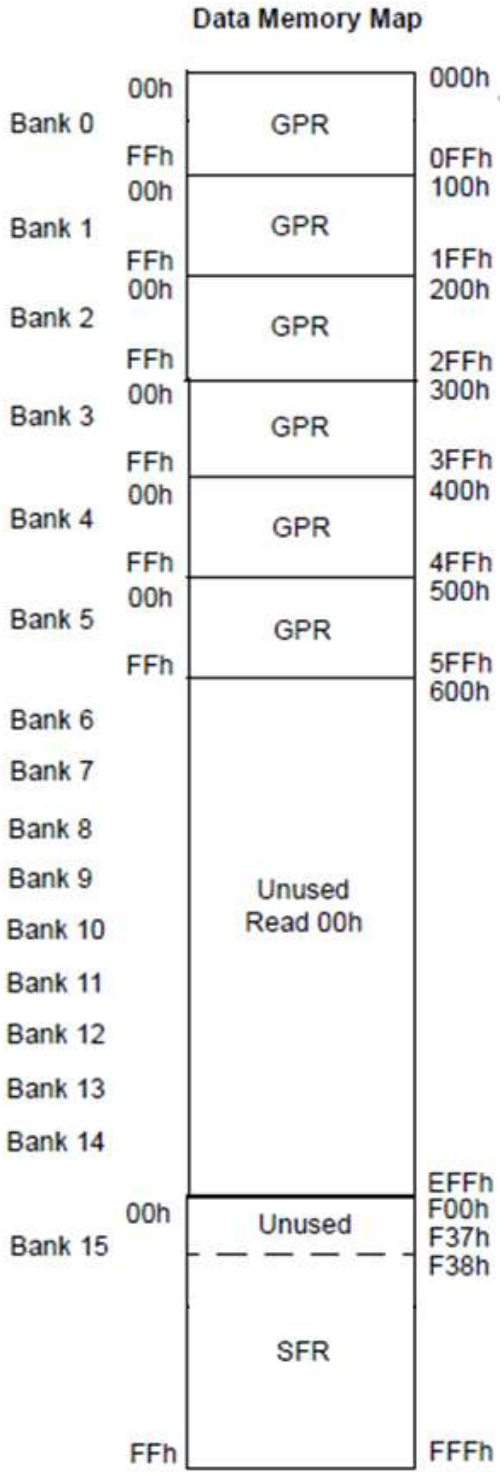
- A PIC18 MCU may have up to 4096 bytes of data memory.

- General-purpose registers (GPR) are used to hold dynamic data.

- Special-function registers (SFR) are used to control the operation of peripheral functions.

Banked RAM

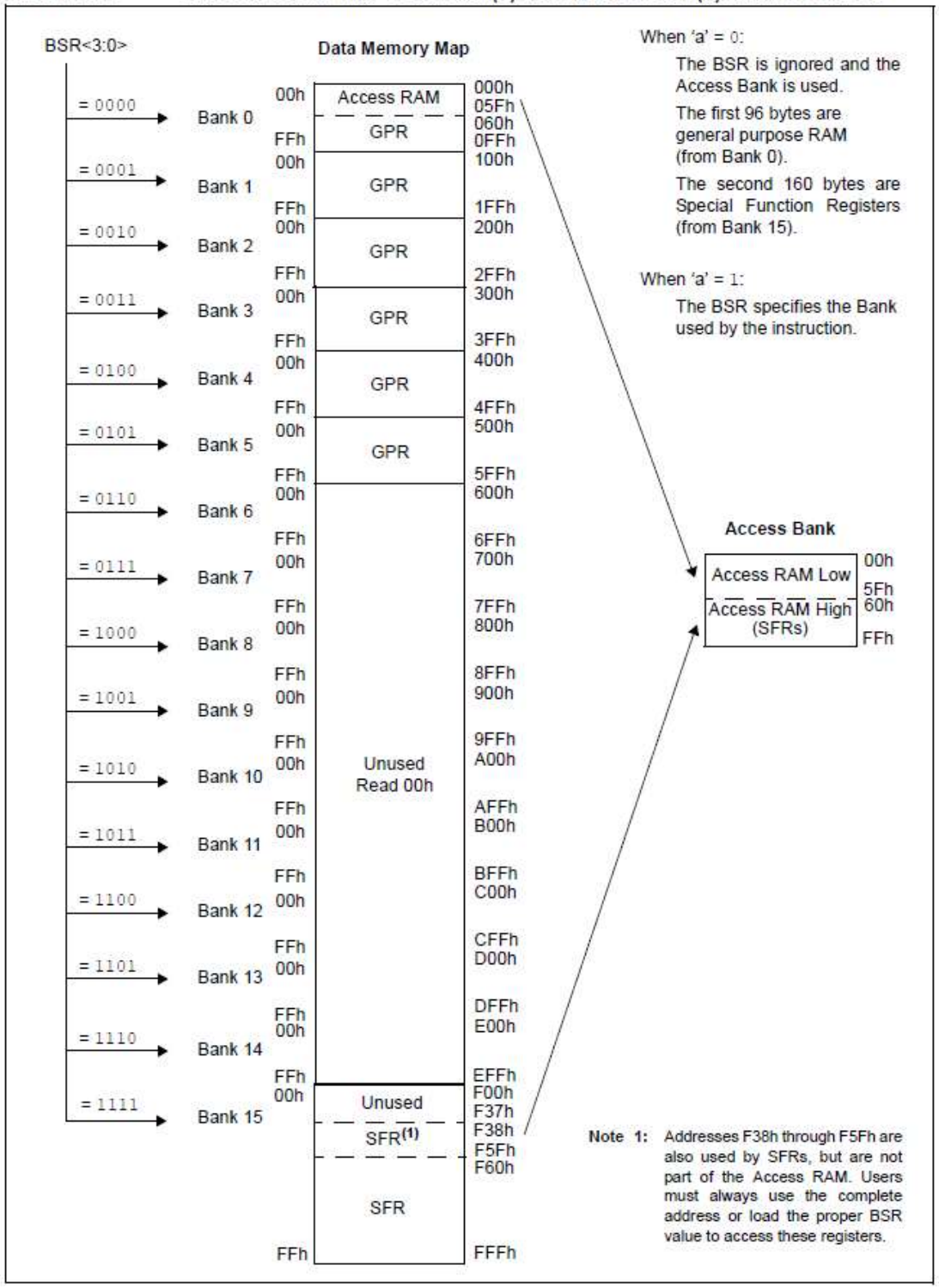
- Most instructions use 8 bits to specify a data register (**f** field).
- Eight bits can specify only 256 registers. This limitation forces the PIC18 to **divide data registers into banks**.
- PIC18F45K22 implements **seven banks:**
  - 6 GPR-based for a total of 6*256B = 1,5KB
  - 1 SFR-based (200B implemented out of 256 possible)



Banked RAM

- Only one bank is active at a time. When operating on a data register in a different bank, bank switching is needed.
  The active bank is specified by the **BSR register** (Bank Select Register)
- Bank switching incurs overhead and may cause program errors.
- **Access bank** is created to minimize the problems of bank switching.
- Access bank consists of the lowest **96 bytes in general-purpose registers** and the highest **160 bytes of special function registers**.
- When operands are in the access bank, no bank switching is needed.

FIGURE 5-7: DATA MEMORY MAP FOR PIC18(L)F25K22 AND PIC18(L)F45K22 DEVICES

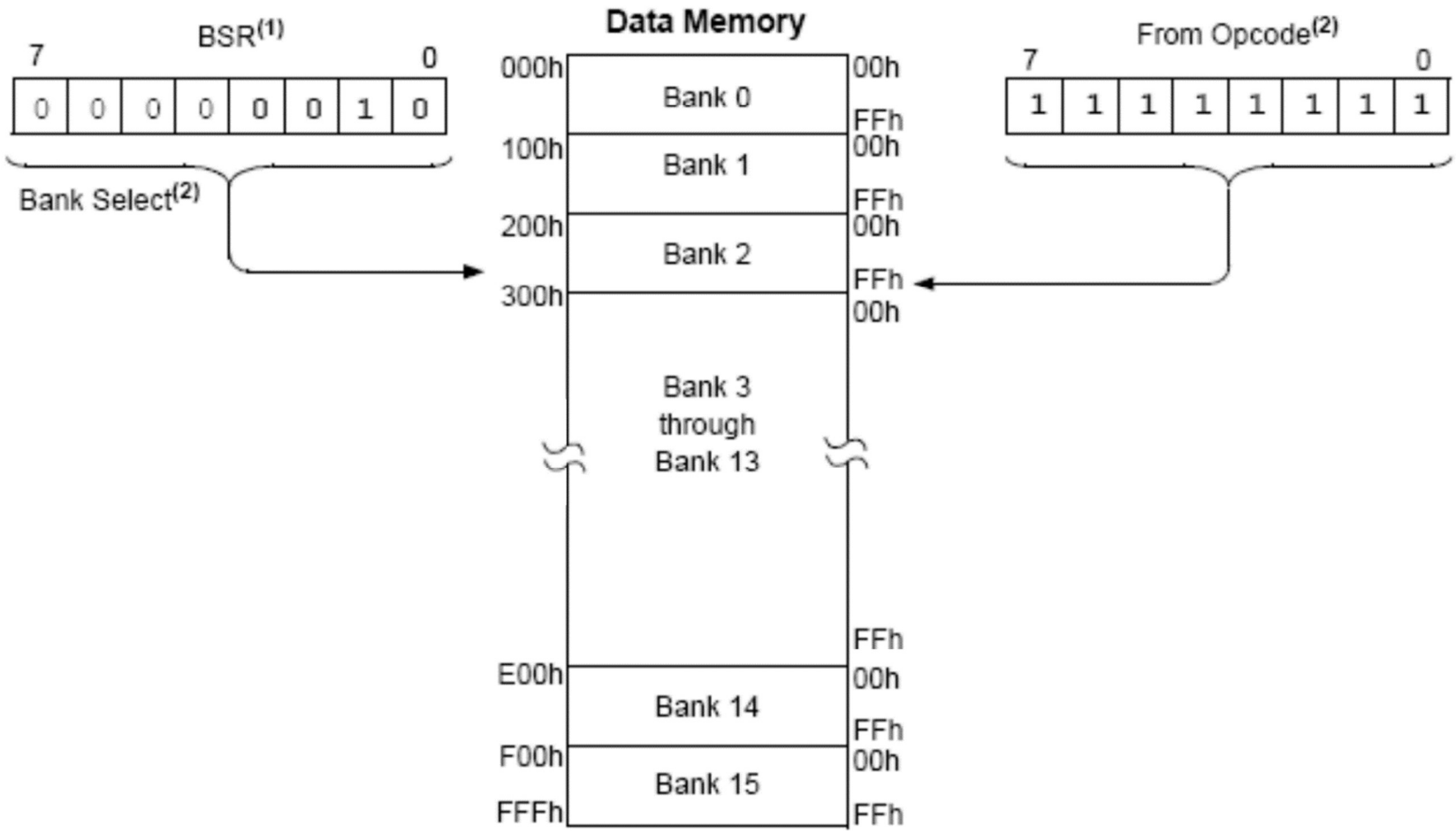


RAM is divided into 256B Banks.

Two addressing modes: Banked (BSR) and Access

Bank Select Register

FIGURE 5-6: USE OF THE BANK SELECT REGISTER (DIRECT ADDRESSING)



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.

(The BSR can be loaded directly by using the MOVLB instruction)

Selecting Access Bank or BSR

- To indicate use of Access Bank or BSR we use the ‘a’ parameter in many instructions

```
movwf f,a
```

When 'a' = 0:

The BSR is ignored and the Access Bank is used.

The first 96 bytes are general purpose RAM (from Bank 0).
The remaining 160 bytes are Special Function Registers (from Bank 15).

When 'a' = 1:

The BSR specifies the bank used by the instruction

Examples of the Use of Access Bank

- The following examples are instructions with 3 parameters, e.g.: **addwf f,d,a**

- Arguments ‘d’ and ‘a’ can be 0 or 1. For the sake of understanding, in the slides we’ll use the following labels:
  - In ‘d’ (destination): **F** (1) indicates file, **W** (0) indicates WREG register
  - In ‘a’ (bank addressing): **A** (0) indicates Access Bank, **BANKED** (1) indicates BSR usage

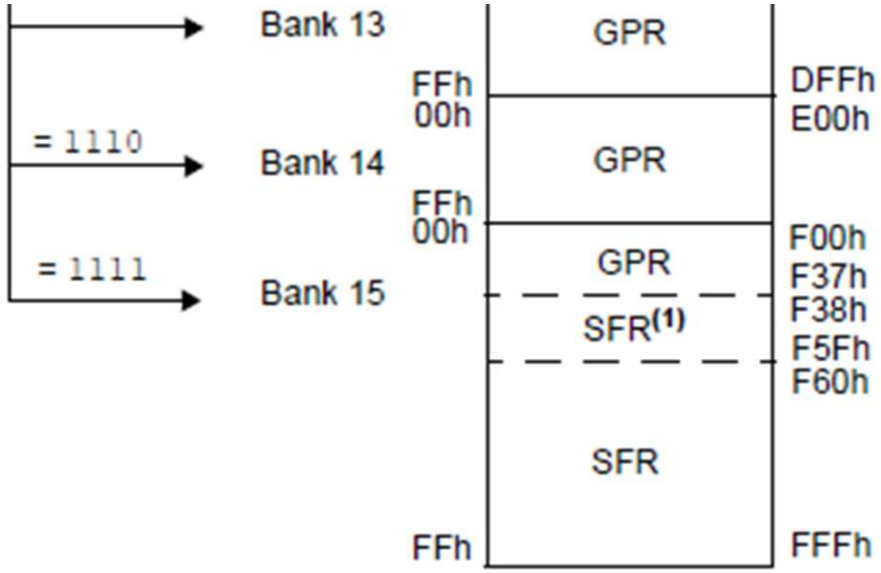
```
addwf 0x20,F,A          ;add WREG with the data register at 0x20 in access bank
                        ;and store the sum at 0x20 in access bank.

subwf 0x30,F,BANKED ;subtract the value of WREG from the data register
                    ;0x30 in the bank specified by the current contents
                    ;of the BSR register. The difference is stored in
                    ;data register 0x30 in the same bank.

addwf 0x40,W,A    ;add the WREG register with data register at 0x40 in
                  ;access bank and leaves the sum in WREG.
```

Special Function Registers

- The group of registers from 0xF38 to 0xFFF are dedicated to the general control of MCU operation and peripherals. Only registers placed between 0xF60 and 0xFFF are mapped in the access bank.

- The WREG register is involved in the execution of many instructions.

- The STATUS register holds the status flags for the instruction execution .



Note 1: Addresses F38h through F5Fh are also used by SFRs, but are not part of the Access RAM. Users must always use the complete address or load the proper BSR value to access these registers.

SFR map for our PIC18F45K22

TABLE 5-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18(L)F2X/4XK22 DEVICES

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FD7h	TMR0H	FAFh	SPBRG1	F87h	—(2)	F5Fh	CCPR3H
FFEh	TOSH	FD6h	TMR0L	FAEh	RCREG1	F86h	—(2)	F5Eh	CCPR3L
FFDh	TOSL	FD5h	T0CON	FADh	TXREG1	F85h	—(2)	F5Dh	CCP3CON
FFCh	STKPTR	FD4h	—(2)	FACh	TXSTA1	F84h	PORTE	F5Ch	PWM3CON
FFBh	PCLATU	FD3h	OSCCON	FABh	RCSTA1	F83h	PORTD(3)	F5Bh	ECCP3AS
FFAh	PCLATH	FD2h	OSCCON2	FAAh	EEADRH(4)	F82h	PORTC	F5Ah	PSTR3CON
FF9h	PCL	FD1h	WDTCON	FA9h	EEADR	F81h	PORTB	F59h	CCPR4H
FF8h	TBLPTRU	FD0h	RCON	FA8h	EEDATA	F80h	PORTA	F58h	CCPR4L
FF7h	TBLPTRH	FCFh	TMR1H	FA7h	EECON2(1)	F7Fh	IPR5	F57h	CCP4CON
FF6h	TBLPTRL	FCEh	TMR1L	FA6h	EECON1	F7Eh	PIR5	F56h	CCPR5H
FF5h	TABLAT	FCDh	T1CON	FA5h	IPR3	F7Dh	PIE5	F55h	CCPR5L
FF4h	PRODH	FCCh	T1GCON	FA4h	PIR3	F7Ch	IPR4	F54h	CCP5CON
FF3h	PRODL	FCBh	SSP1CON3	FA3h	PIE3	F7Bh	PIR4	F53h	TMR4
FF2h	INTCON	FCAh	SSP1MSK	FA2h	IPR2	F7Ah	PIE4	F52h	PR4
FF1h	INTCON2	FC9h	SSP1BUF	FA1h	PIR2	F79h	CM1CON0	F51h	T4CON
FF0h	INTCON3	FC8h	SSP1ADD	FA0h	PIE2	F78h	CM2CON0	F50h	TMR5H
FEFh	INDF0(1)	FC7h	SSP1STAT	F9Fh	IPR1	F77h	CM2CON1	F4Fh	TMR5L
FEEh	POSTINC0(1)	FC6h	SSP1CON1	F9Eh	PIR1	F76h	SPBRGH2	F4Eh	T5CON
FEDh	POSTDEC0(1)	FC5h	SSP1CON2	F9Dh	PIE1	F75h	SPBRG2	F4Dh	T5GCON
FECh	PREINC0(1)	FC4h	ADRESH	F9Ch	HLVDCON	F74h	RCREG2	F4Ch	TMR6
FEBh	PLUSW0(1)	FC3h	ADRESL	F9Bh	OSCTUNE	F73h	TXREG2	F4Bh	PR6
FEAh	FSR0H	FC2h	ADCON0	F9Ah	—(2)	F72h	TXSTA2	F4Ah	T6CON
FE9h	FSR0L	FC1h	ADCON1	F99h	—(2)	F71h	RCSTA2	F49h	CCPTMRS0
FE8h	WREG	FC0h	ADCON2	F98h	—(2)	F70h	BAUDCON2	F48h	CCPTMRS1
FE7h	INDF1(1)	FBFh	CCPR1H	F97h	—(2)	F6Fh	SSP2BUF	F47h	SRCON0
FE6h	POSTINC1(1)	FBEh	CCPR1L	F96h	TRISE	F6Eh	SSP2ADD	F46h	SRCON1
FE5h	POSTDEC1(1)	FBDh	CCP1CON	F95h	TRISD(3)	F6Dh	SSP2STAT	F45h	CTMUCONH
FE4h	PREINC1(1)	FBCh	TMR2	F94h	TRISC	F6Ch	SSP2CON1	F44h	CTMUCONL
FE3h	PLUSW1(1)	FBBh	PR2	F93h	TRISB	F6Bh	SSP2CON2	F43h	CTMUICON
FE2h	FSR1H	FBAh	T2CON	F92h	TRISA	F6Ah	SSP2MSK	F42h	VREFCON0
FE1h	FSR1L	FB9h	PSTR1CON	F91h	—(2)	F69h	SSP2CON3	F41h	VREFCON1
FE0h	BSR	FB8h	BAUDCON1	F90h	—(2)	F68h	CCPR2H	F40h	VREFCON2
FDFh	INDF2(1)	FB7h	PWM1CON	F8Fh	—(2)	F67h	CCPR2L	F3Fh	PMD0
FDEh	POSTINC2(1)	FB6h	ECCP1AS	F8Eh	—(2)	F66h	CCP2CON	F3Eh	PMD1
FDDh	POSTDEC2(1)	FB5h	—(2)	F8Dh	LATE(3)	F65h	PWM2CON	F3Dh	PMD2
FDCh	PREINC2(1)	FB4h	T3GCON	F8Ch	LATD(3)	F64h	ECCP2AS	F3Ch	ANSELE
FDBh	PLUSW2(1)	FB3h	TMR3H	F8Bh	LATC	F63h	PSTR2CON	F3Bh	ANSELD
FDAh	FSR2H	FB2h	TMR3L	F8Ah	LATB	F62h	IOCB	F3Ah	ANSELC
FD9h	FSR2L	FB1h	T3CON	F89h	LATA	F61h	WPUB	F39h	ANSELB
FD8h	STATUS	FB0h	SPBRGH1	F88h	—(2)	F60h	SLRCON	F38h	ANSELA

Not in ACCESS BANK !!!

Note
1: Not a physical register.
2: Unimplemented registers are read as '0'.
3: These registers are implemented only on 40/44-pin devices.

Status register

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative (ALU MSB = 1).
1 = Result was negative
0 = Result was positive

bit 3 **OV:** Overflow bit
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude which causes the sign bit (bit 7 of the result) to change state.
1 = Overflow occurred for signed arithmetic (in this arithmetic operation)
0 = No overflow occurred

bit 2 **Z:** Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

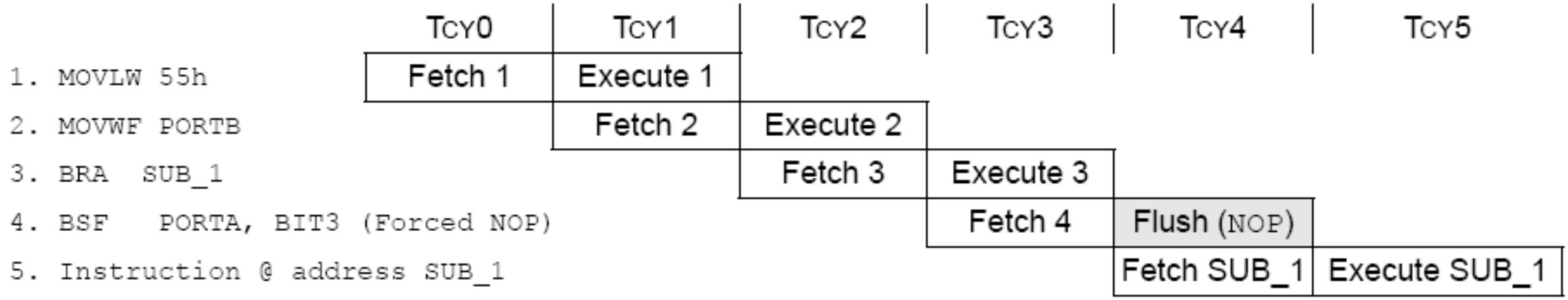
bit 1 **DC:** Digit Carry/$\overline{\text{Borrow}}$ bit(1)
For ADDWF, ADDLW, SUBLW and SUBWF instructions:
1 = A carry-out from the 4th low-order bit of the result occurred
0 = No carry-out from the 4th low-order bit of the result

bit 0 **C:** Carry/$\overline{\text{Borrow}}$ bit(2)
For ADDWF, ADDLW, SUBLW and SUBWF instructions:
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

The PIC18 Pipelining

- The PIC18 divide most of the instruction execution into two stages: **instruction fetch** and **instruction execution**.

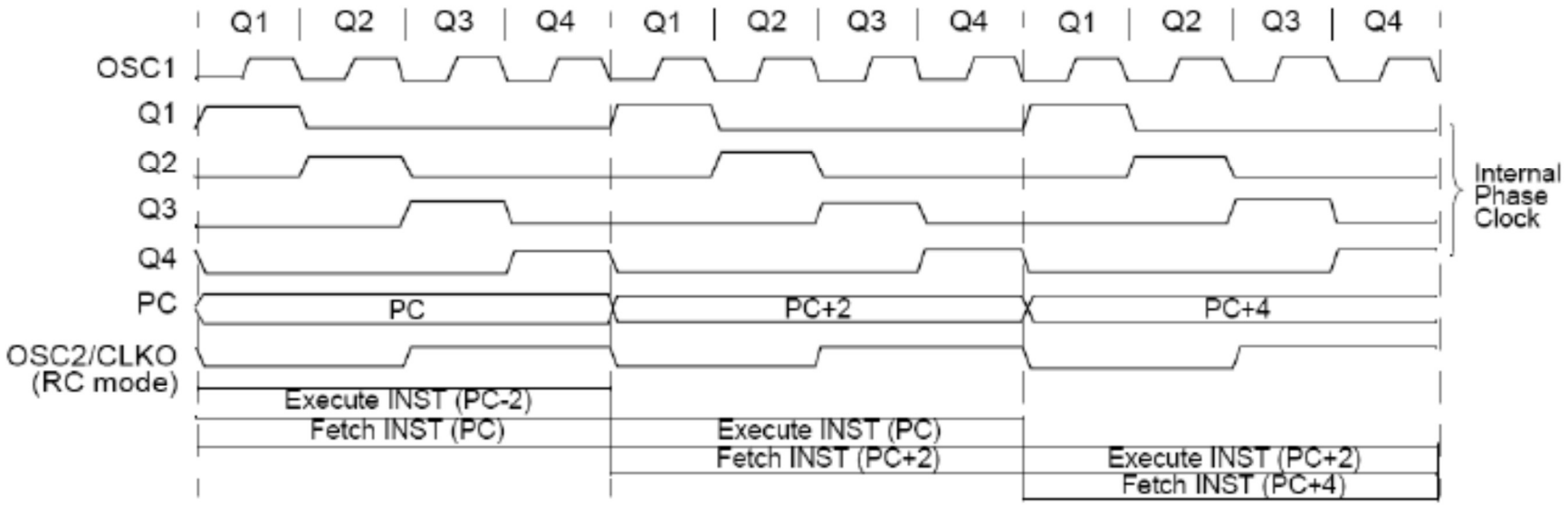
- Up to two instructions are overlapped in their execution. One instruction is in fetch stage while the second instruction is in execution stage. Because of pipelining, each instruction appears to take one instruction cycle to complete.



- All single-word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles with the additional instruction cycle(s) executed as a NOP.

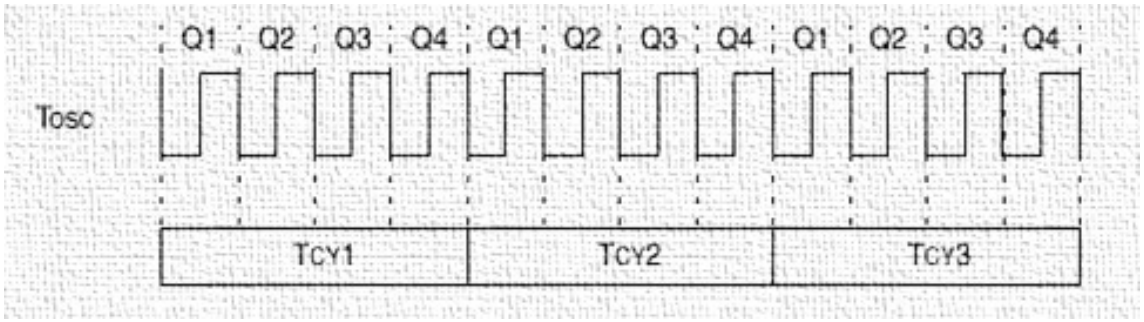
- The double-word instructions execute in two instruction cycles.

The PIC18 Pipelining



f_{OSC} = 8 MHz → T_{OSC} = 125 ns → T_{cyc} = 500 ns

f_{OSC_MAX} =20 MHz → T_{OSC} = 50 ns → T_{cyc} = 200 ns



Instruction subcycles (Q-cycles)

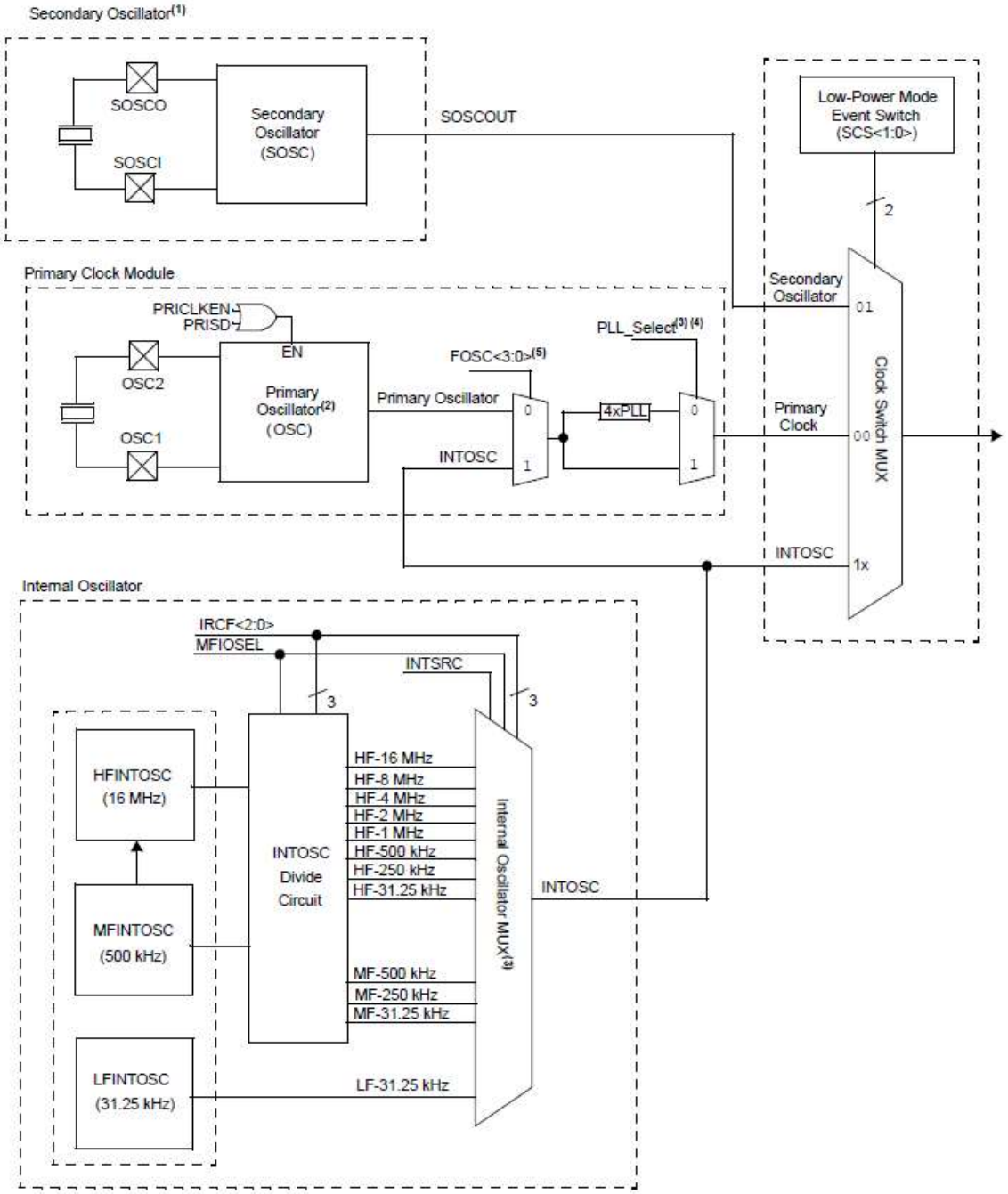
Q1. instruction decode subcycle

Q2. read data subcycle

Q3. process subcycle

Q4. write data subcycle

PIC18F45K22 system clock diagram



PIC18 Addressing Modes

4 addressing modes:

- **Inherent**: don't need any argument

```
sleep, reset
```

- **Literal**: require an explicit argument in the opcode

```
movlw 0x30      ; load 0x30 into WREG
```

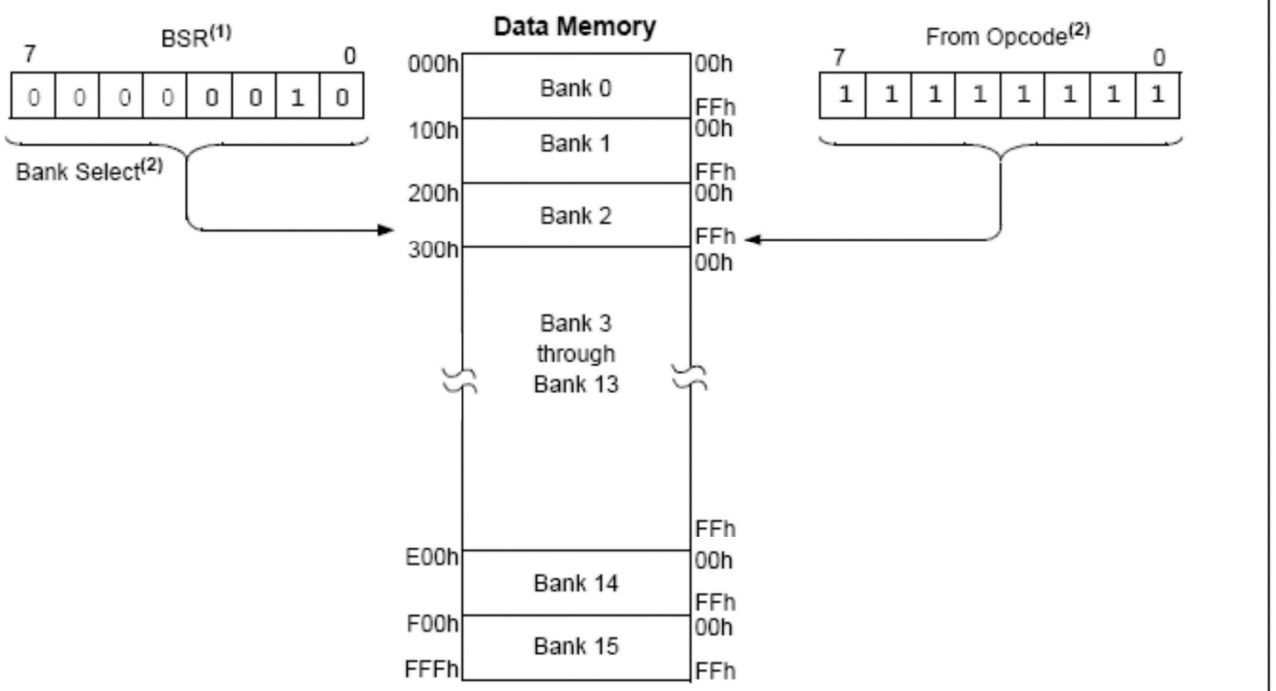
- **Direct**: specifies source and/or destination address

```
movwf 0x20      ; the value 0x20 is
                ; register direct mode
```

- **Indirect**: Access a location without giving a fixed address in the instruction. Use FSR register as pointers.

Direct addressing mode

FIGURE 5-6: USE OF THE BANK SELECT REGISTER (DIRECT ADDRESSING)



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.

ADDWF f, d, a

The destination of the operation's results is determined by the destination bit 'd'.

'd' = 1, the results are stored back in the source register, overwriting its original contents.

'd' = 0, the results are stored in the W register.

Instructions without the 'd' argument have a destination that is implicit in the instruction.

Indirect addressing mode

- File Select Registers (FSRx) are used as pointers to the actual data register.
- The registers for Indirect Addressing are also implemented with Indirect File Operands that permit automatic manipulation of the pointer value with auto-incrementing, auto-decrementing or offsetting with another value.
- 5 Modes: INDF POSTDEC POSTINC PREINC PLUSW

EXAMPLE 5-5: HOW TO CLEAR RAM (BANK 1) USING INDIRECT ADDRESSING

```
        LFSR    FSR0, 100h ;
NEXT    CLRF    POSTINC0   ; Clear indirect
                           ; register then
                           ; inc FSR pointer
        BTFSS   FSR0H, 1   ; All done with
                           ; Bank1?
        BRA     NEXT       ; NO, clear next
CONTINUE                   ; YES, continue
```

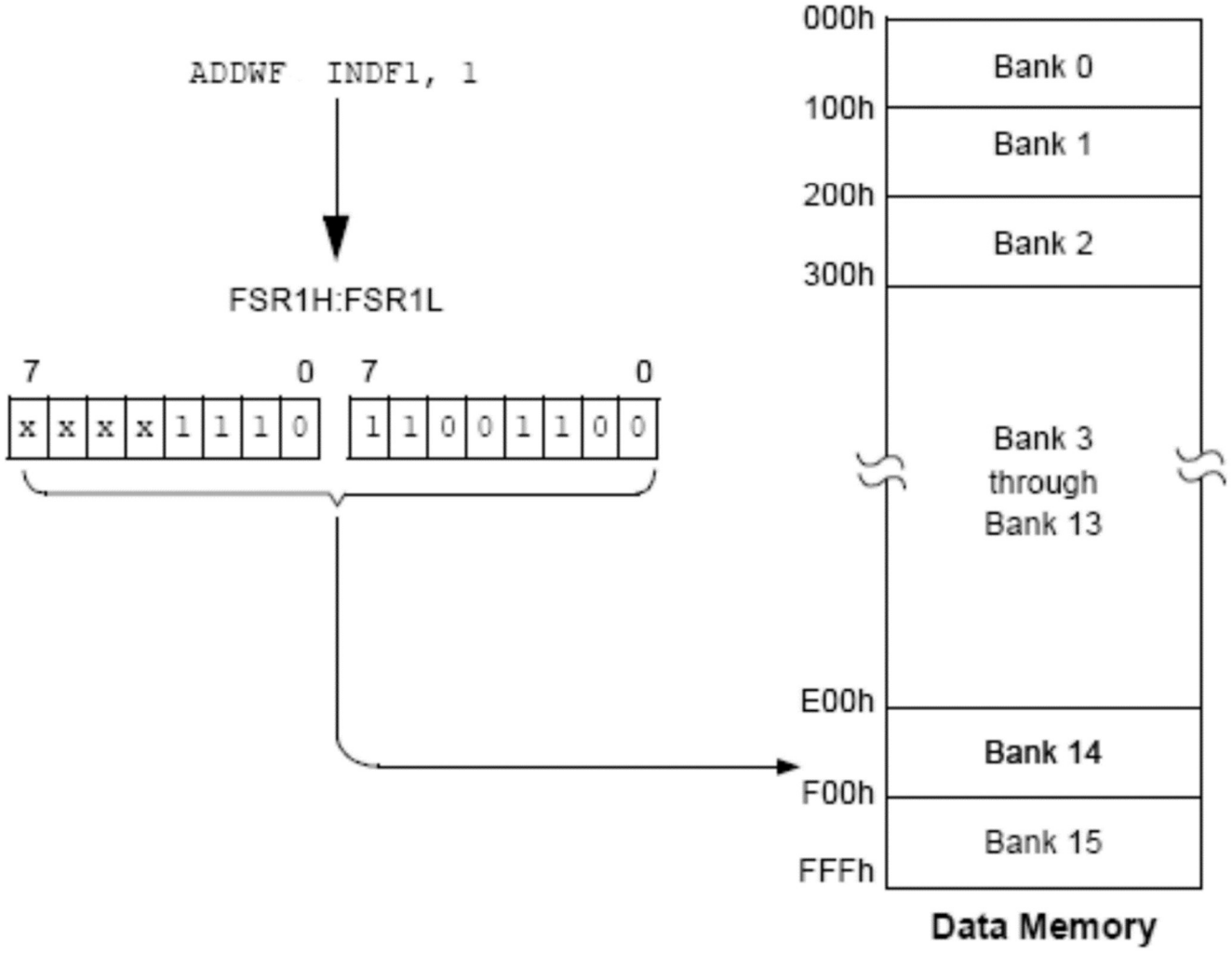
Indirect addressing mode

Using an instruction with one of the indirect addressing registers as the operand....

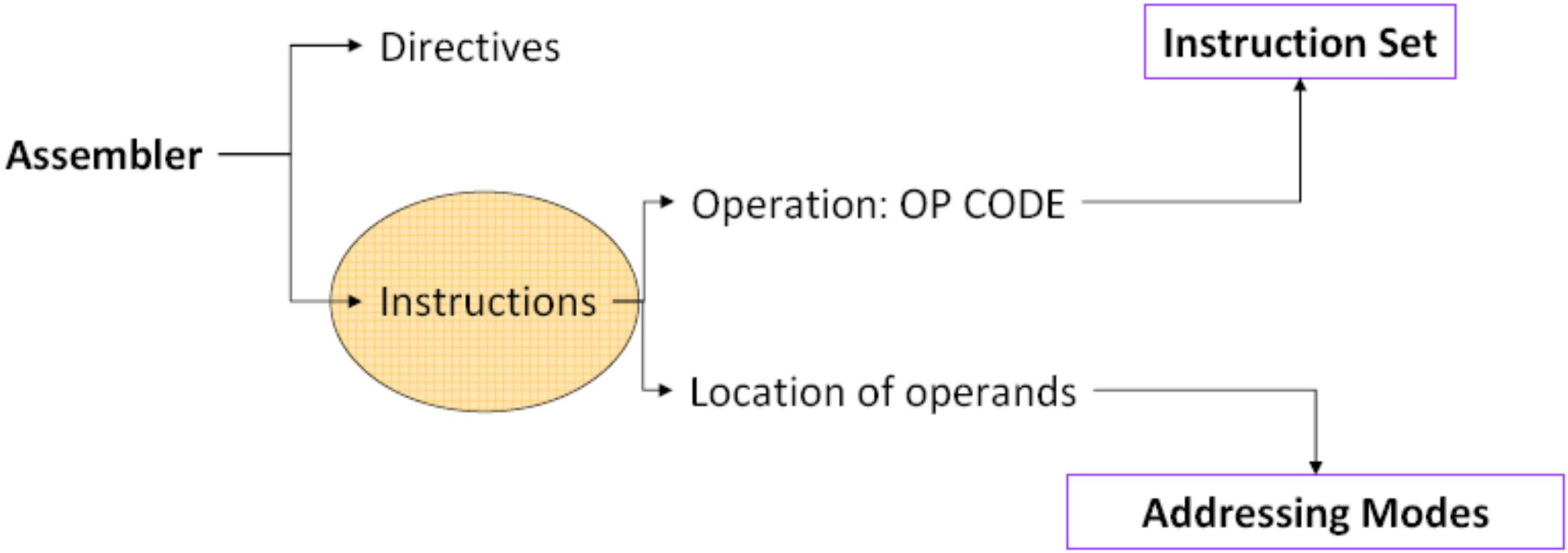
...uses the 12-bit address stored in the FSR pair associated with that register....

...to determine the data memory location to be used in that operation.

In this case, the FSR1 pair contains ECCh. This means the contents of location ECCh will be added to that of the W register and stored back in ECCh.



Fundamentals of assembler



Assembler directives

There are six basic types of directives provided by the assembler.

- Control Directives
- Conditional Assembly Directives
- Data Directives
- Listing Directives
- Macro Directives
- Object File Directives

Assembler directives. Examples

#define <name> [<string>] #define PORTA 80

This directive defines a text substitution string. Whenever <name> is encountered in the assembly code, <string> will be substituted.

#include <include_file> #include <p18f2525.inc>

This directive includes additional source file. The specified file is read in as source code. The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement.

[<label>] org <expr> Reset ORG 0000h

This directive sets the program origin for subsequent code at the address defined in <expr>.

Programs. Begin & end

START

We fix the location of instructions in memory by the directive ORG

```
org 0x0000h
bra main
```

STOP

Program can NOT wander around any memory location.
Execution must be limited to program lines written by user.

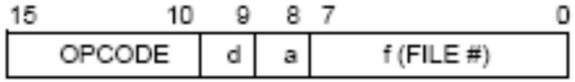
```
loop
 bra loop
end
```

Usual classification of instructions

- Data Transfer Movement (Move)
- Data Modification (Clear, Inc, Dec)
- Rotation Bits (Shift, Rotate)
- Arithmetic (Add, Sub, Mult, Div)
- Logic (And, Or, Xor)
- Bitwise (Set bit, Clear bit, Jump if bit set, Jump if bit clear)
- Branching Control (Jump, Conditional jumps)
- Stack (Push, Pull)
- Subroutines (Call, Return)
- Interrupt (Int. Retfie)

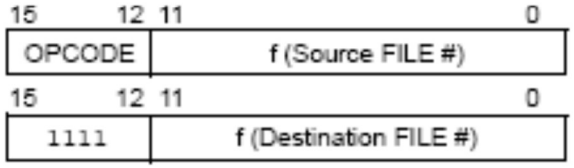
Microchip classification of instructions

Byte-oriented file register operations



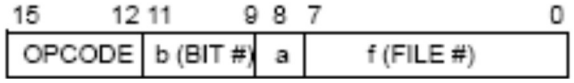
d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Byte to Byte move operations (2-word)



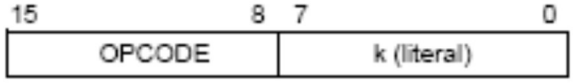
f = 12-bit file register address

Bit-oriented file register operations



b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Literal operations



k = 8-bit immediate value

Example Instruction

```
ADDWF MYREG, W, B
```

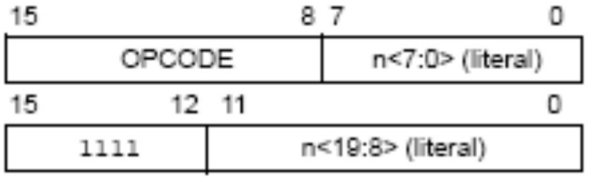
```
MOVFF MYREG1, MYREG2
```

```
BSF MYREG, bit, B
```

```
MOVLW 7Fh
```

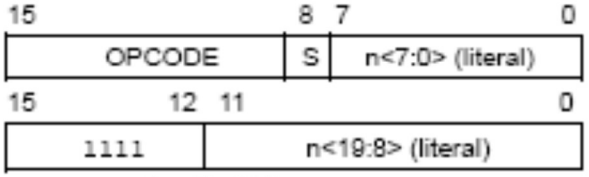
Control operations

CALL, GOTO and Branch operations



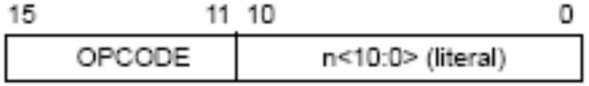
n = 20-bit immediate value

```
GOTO Label
```

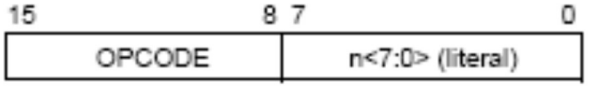


S = Fast bit

```
CALL MYFUNC
```



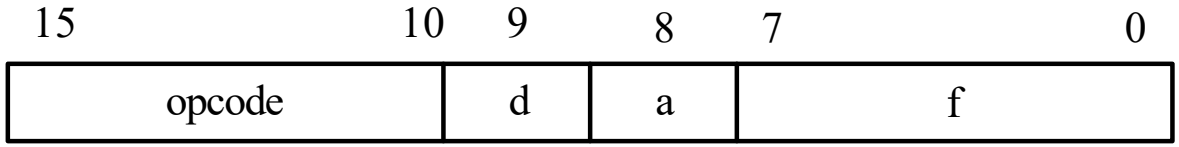

```
BRA MYFUNC
```




```
BC MYFUNC
```

Instruction Format

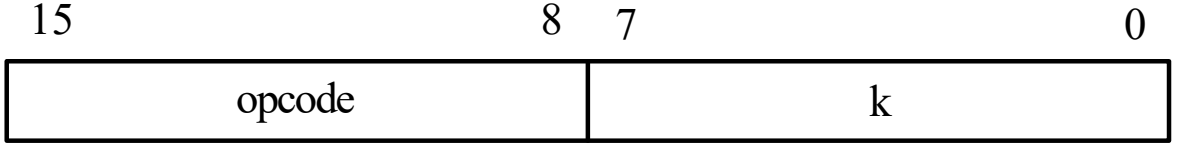
Byte-oriented operations



d = 0 for result destination to be WREG register.
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Literal operations

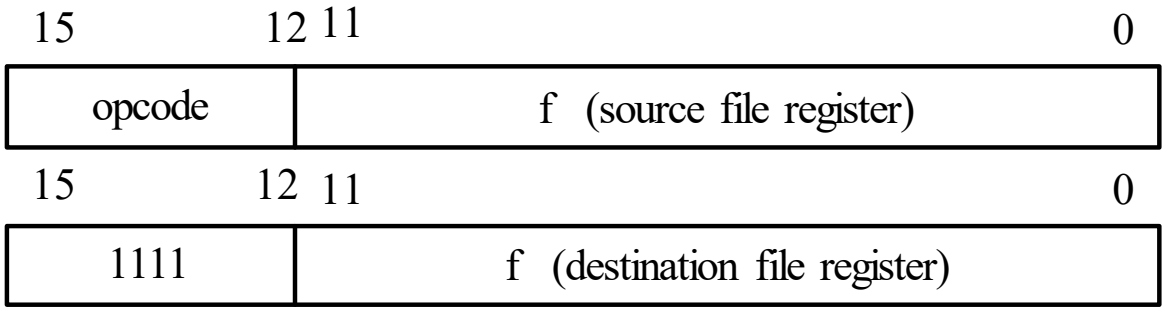
- A literal is a number to be operated on directly by the CPU



k = 8-bit immediate value

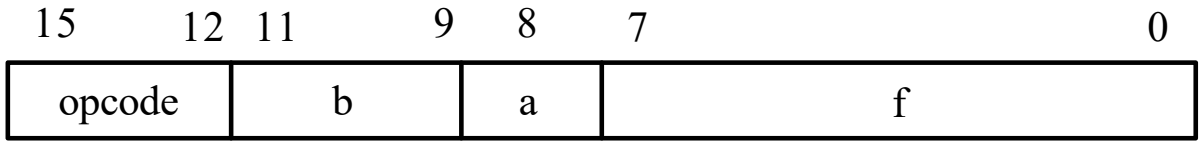
Instruction Format

Byte-to-byte move operations (2 words)



f = 12-bit file register address

Bit-oriented file register operations

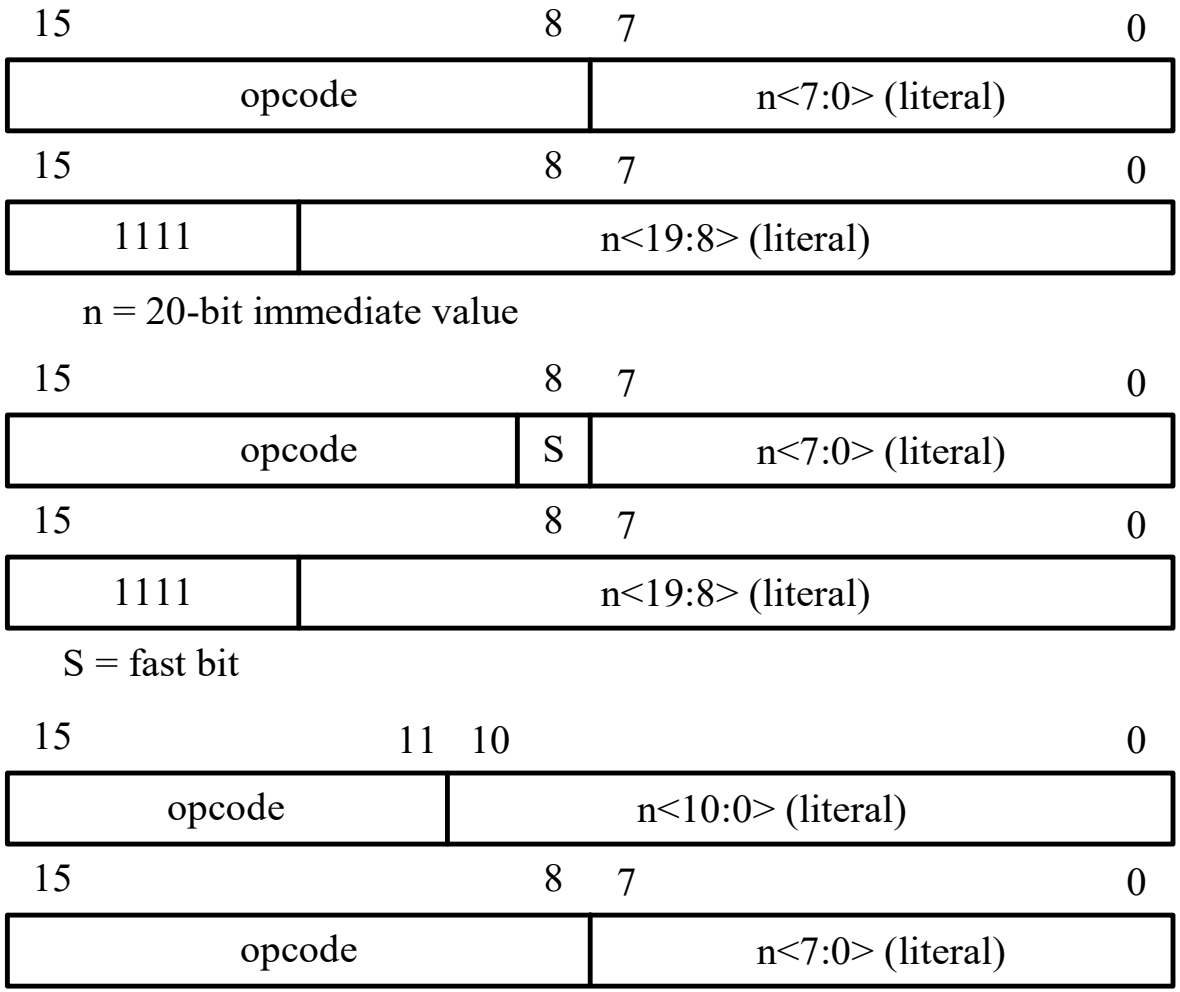


b = 3-bit position of bit in the file register (f).
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Instruction Format

Control operations

- These instructions are used to change the program execution sequence and making subroutine calls.



Instruction Set (1/3)

TABLE 25-2: PIC18(L)F2X/4XK22 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and CARRY bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to f_d (destination)	2	1100 1111	ffff ffff	ffff ffff	ffff ffff	None	1st word 2nd word
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	1, 2
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	

Instruction Set (2/3)

TABLE 25-2: PIC18(L)F2X/4XK22 INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	k, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	k	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Instruction Set (3/3)

TABLE 25-2: PIC18(L)F2X/4XK22 INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word	2	1110	1110	00ff	kkkk	None	
				1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, 'd' = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are two-word instructions. The second word of these instructions will be executed as a `NOP` unless the first word of the instruction retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

Assembler instructions

ADDLW — **ADD Literal to W**

Syntax: ADDLW k

Operands: $0 \le k \le 255$

Operation: (W) + k → W

Status Affected: N, OV, C, DC, Z

Encoding:

0000	1111	kkkk	kkkk

Description: The contents of W are added to the 8-bit literal 'k' and the result is placed in W.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `ADDLW 15h`

Before Instruction
W = 10h
After Instruction
W = 25h

ADDWF — **ADD W to f**

Syntax: ADDWF f {,d {,a}}

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (W) + (f) → dest

Status Affected: N, OV, C, DC, Z

Encoding:

0010	01da	ffff	ffff

Description: Add W to register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

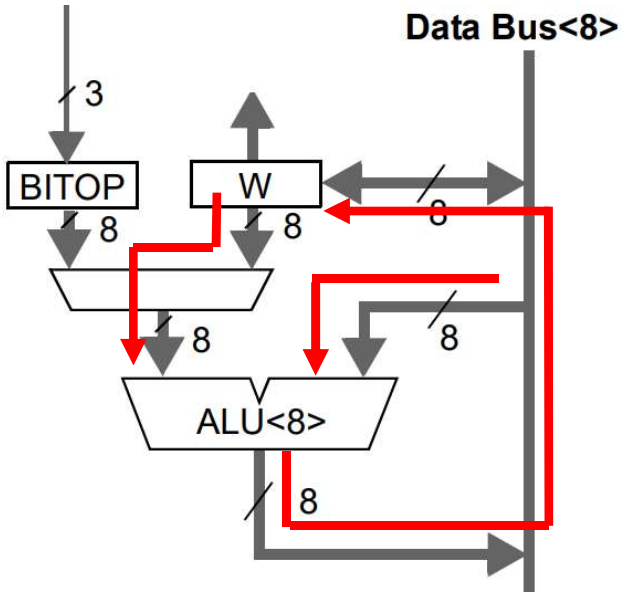
Cycles: 1

f/W distinction

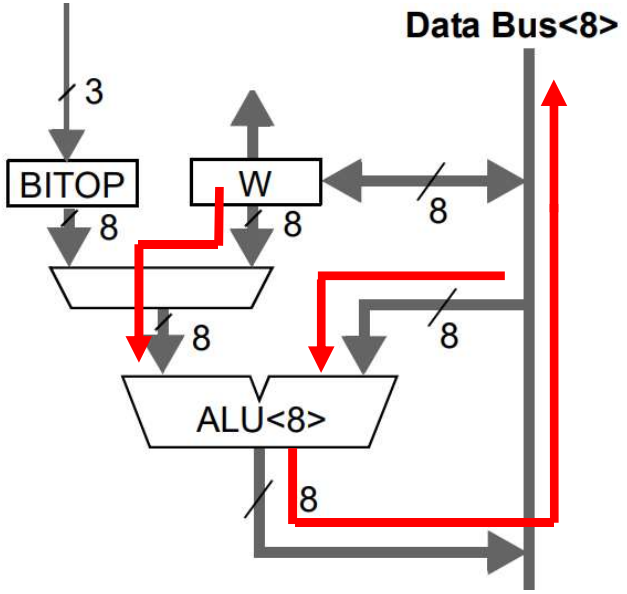
Accumulator architecture.

Examples:

```
addwf NUM1,0,0   ;add WREG plus NUM1 in AccesBank
                 ;put result to WREG
```




```
addwf NUM1,1,0   ;add WREG plus NUM1 in AccesBank
                 ;put result back to NUM1
```



Case issue

- The PIC18 instructions can be written in either uppercase or lowercase.

- MPASM allows the user to include “p18Fxxxx.inc” file to provide register definitions for the specific processor.

- All special function registers and bits are defined in uppercase.

- The convention followed in this text is: using **lowercase** for instructions and directives, using **uppercase** for special function registers.

Assembly program template

```
        org     0x0000   ;  program starting address after power on reset
        goto    start

        org     0x08
        ...              ;  isr code here
        retfie           ;  high-priority interrupt service routine

        org     0x18
        ...              ;  isr code here
        retfie           ;  low-priority interrupt service routine

start   …                ;  your program here
        …
        end
```

Sample program 1

Example Write a program that adds the three numbers stored in data registers at 0x20, 0x30, and 0x40 and places the sum in data register at 0x50.

Algorithm:

Step 1
Load the number stored at 0x20 into the WREG register.
Step 2
Add the number stored at 0x30 and the number in the WREG register and leave the sum in the WREG register.
Step 3
Add the number stored at 0x40 and the number in the WREG register and leave the sum in the WREG register.
Step 4
Store the contents of the WREG register in the memory location at 0x50.

Sample program 1

The program that implements this algorithm is as follows:

```
#include <p18F8720.inc>          ; can be other processor
        org     0x00
        goto    start
        org     0x08
        retfie
        org     0x18
        retfie
start   movf    0x20,W,A         ; WREG ← [0x20]
        addwf   0x30,W,A         ; WREG ← [0x20] + [0x30]
        addwf   0x40,W,A         ; WREG ← [0x20] + [0x30] + [0x40]
        movwf   0x50,A           ; 0x50 ← sum (in WREG)
        end
```

Sample program 2

Example Write a program to add two 24-bit numbers stored at 0x10~0x12 and 0x13~0x15 and leave the sum at 0x20~0x22.

Solution:

```
            #include <p18F8720.inc>
            org     0x00
            goto    start
            org     0x08
            retfie
            org     0x18
            retfie
start       movf    0x10,W,A        ; WREG ← [0x10]
            addwf   0x13,W,A        ; WREG ← [0x13] + [0x10]
            movwf   0x20,A          ; 0x20 ← [0x10] + [0x13]
            movf    0x11,W,A        ; WREG ← [0x11]
            addwfc  0x14,W,A        ; WREG ← [0x11] + [0x14] + C flag
            movwf   0x21,A          ; 0x21 ← [WREG]
            movf    0x12,W,A        ; WREG ← [0x12]
            addwfc  0x15,W,A        ; WREG ← [0x12] + [0x15] + C flag
            movwf   0x22,A          ; 0x22 ← [WREG]
            end
```

Changing the Program Counter

- Microcontroller executes instruction sequentially in normal condition.
- PIC18 has a 21-bit program counter (PC) which is divided into three registers: PCL, PCH, and PCU.
- PCL can be accessed directly. However, PCH and PCU are not directly accessible.
- One can access the values of PCH and PCU indirectly by accessing the PCLATH and PCLATU.
- Reading the PCL will cause the values of PCH and PCU to be copied into the PCLATH and PCLATU.
- Writing the PCL will cause the values of PCLATH and PCLATU to be written into the PCH and PCU.
- In normal program execution, the PC value is incremented by either 2 or 4.
- To implement a program loop, the processor needs to change the PC value by a value other than 2 or 4.

Instructions for Changing Program Counter

BRA n: jump to the instruction with address equal to PC+2+n

B_{CC} n: jump to the instruction with address equal to PC+2+n if the condition code CC is true.

CC can be any one of the following:

C: if C (Carry) flag is set to 1

N: if N (Negative) flag is set to 1 which indicates that the previous operation result was negative

NC: if C flag is 0

NN: if N flag is 0 which indicates non-negative condition

OV: if OV flag is 1 indicating previous operation caused an overflow

Z: if Z flag is 1 which indicates the previous operation result was zero

NOV: if OV (Overflow) flag is 0 indicating there is no overflow condition

NZ: if Z (Zero) flag is 0 indicating previous operation result was not zero

GOTO n: jump to address represented by **n**

The destination of a **branch** or **goto** instruction is normally specified by a label.

Instructions for Changing Program Counter

```
cpfseq    f,a      ; compare register f with WREG, skip if equal
cpfsgt    f,a      ; compare register f with WREG, skip if equal
cpfslt    f,a      ; compare register f with WREG, skip if less than
decfsz    f,d,a    ; decrement f, skip if 0
dcfsnz    f,d,a    ; decrement f, skip if not 0
incfsz    f,d,a    ; increment f, skip if 0
infsnz    f,d,a    ; increment f, skip if not 0
tstfsz    f,a      ; test f, skip if 0
btfsc     f,b,a    ; test bit b of register f, skip if 0
btfss     f,b,a    ; test bit b of register f, skip if 1
```

Instructions for Inc/Decrementing file registers

Instructions for changing register value by 1:

```
incf    f,d,a
decf    f,d,a
```

If then else. Example

```
        btfsc STATUS,Z          ; flag Z test
        goto Zset
Zclear  . . .                   ; code for Z=0
        goto Zdone
Zset    . . .                   ; code for Z=1
Zdone   . . .                   ; We're done
```

Loops. Example1

loop that execute n times

Example 1

```
i_cnt    equ     PRODL ; use PRODL as loop count
         clrf    i_cnt,A
i_loop   …
         …               ; i_cnt is incremented in the loop
         movlw   n
         cpfseq  i_cnt,A  ; compare i_cnt with WREG and skip if equal
         goto    i_loop   ; executed when i_cnt ≠ loop limit
```

Loops. Example2

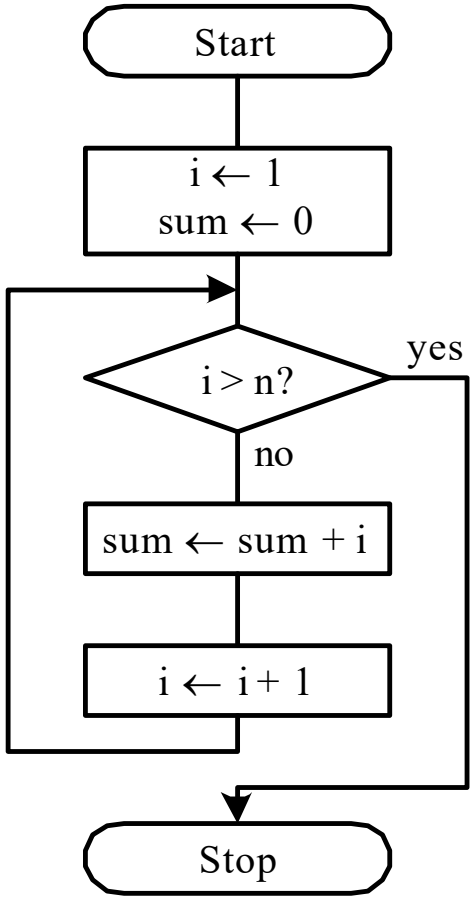
```
n       equ     20          ; n has the value of 20
lp_cnt  set     0x10        ; assign file register 0x10 to lp_cnt
        …
        movlw   n
        movwf   lp_cnt      ; prepare to repeat the loop for n times
loop    …                   ; program loop
        …                   ;       “
        decfsz  lp_cnt,F,A  ; decrement lp_cnt and skip if equal to 0
        goto    loop        ; executed if lp_cnt ≠ 0
```

Sample program 3

Write a program to compute 1 + 2 + 3 + … + n and save the sum at 0x00 and 0x01.

Solution:

1. Program logic



Flowchart for computing 1+2+…+n

Sample program 3

```
            #include <p18F8720.inc>
            radix       dec
n           equ         D'50'
sum_hi      set         0x01        ; high byte of sum
sum_lo      set         0x00        ; low byte of sum
i           set         0x02        ; loop index i
            org         0x00        ; reset vector
            goto        start
            org         0x08
            retfie
            org         0x18
            retfie
start       clrf        sum_hi,A    ; initialize sum to 0
            clrf        sum_lo,A    ;               "
            clrf        i,A         ; initialize i to 0
            incf        i,F,A       ; i starts from 1
sum_lp      movlw       n           ; place n in WREG
            cpfsgt      i,A         ; compare i with n and skip if i > n
            bra         add_lp      ; perform addition when i ≤ 50
            bra         exit_sum    ; it is done when i > 50
add_lp      movf        i,W,A       ; place i in WREG
            addwf       sum_lo,F,A  ; add i to sum_lo
            movlw       0
            addwfc      sum_hi,F,A  ; add carry to sum_hi
            incf        i,F,A       ; increment loop index i by 1
            bra         sum_lp
exit_sum    nop
            bra         exit_sum
            end
```

Sample program: 1 sec. delay

```
lp_cnt0     equ     0x20
lp_cnt1     equ     0x21
lp_cnt2     equ     0x22
            movlw   D'10'
            movwf   lp_cnt0,A
loop0       movlw   D'200'
            movwf   lp_cnt1,A
loop1       movlw   D'250'
            movwf   lp_cnt2,A
loop2       nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            nop                 ; 1 instruction cycle
            decfsz  lp_cnt2,F,A ; 1 instruction cycle (2 when [lp_cnt1] = 0)
            goto    loop2       ; 2 instruction cycle
            decfsz  lp_cnt1,F,A
            goto    loop1
            decfsz  lp_cnt0,F,A
            goto    loop0       ; 1 sec delay. (assume instruction clock period is 100ns, Fosc = 40 MHz):
```