Microprocessor based Digital Systems

Assembler Programming

Guillermo Carpintero

Universidad **Carlos III** de Madrid

Programming Languages

Allow us to express the sequence of operations that we need the processor to perform

Machine Code 10010100111101

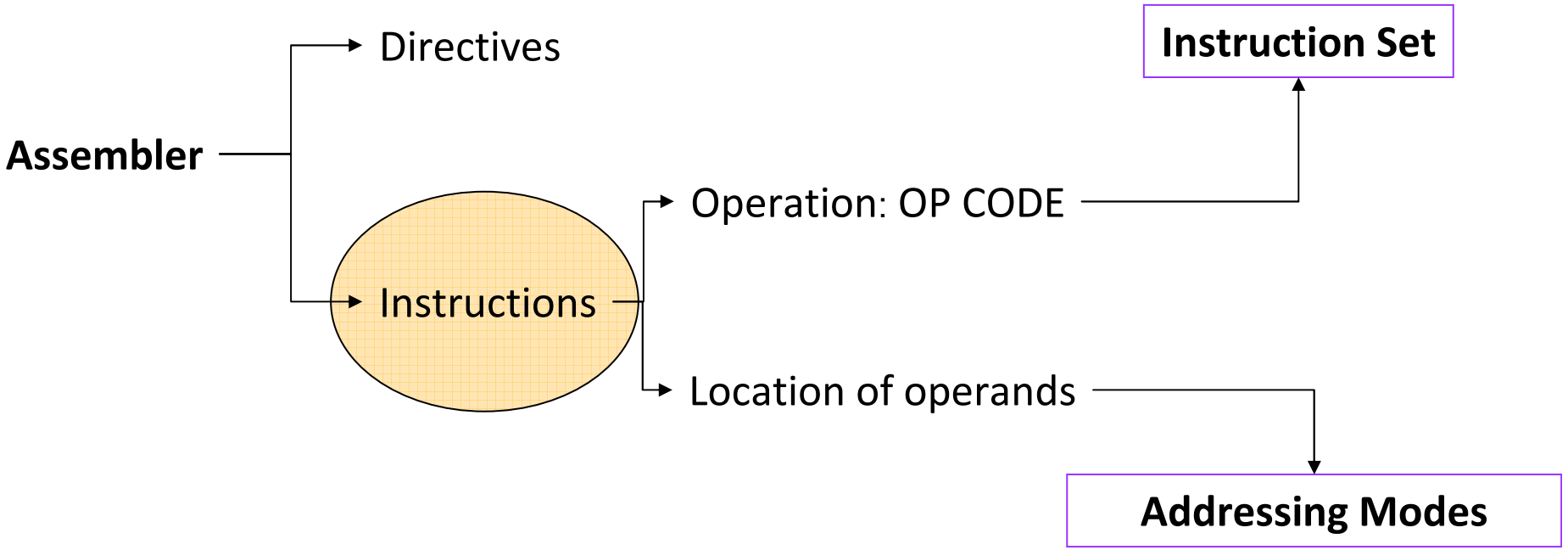
Assembler add A,B

- Based on mnemonics
- One-to-one correspondence to machine codes

High level A = A + B

- Close to formal language structures A += 1

Fundamentals of Assembler



Fundamentals of Assembler

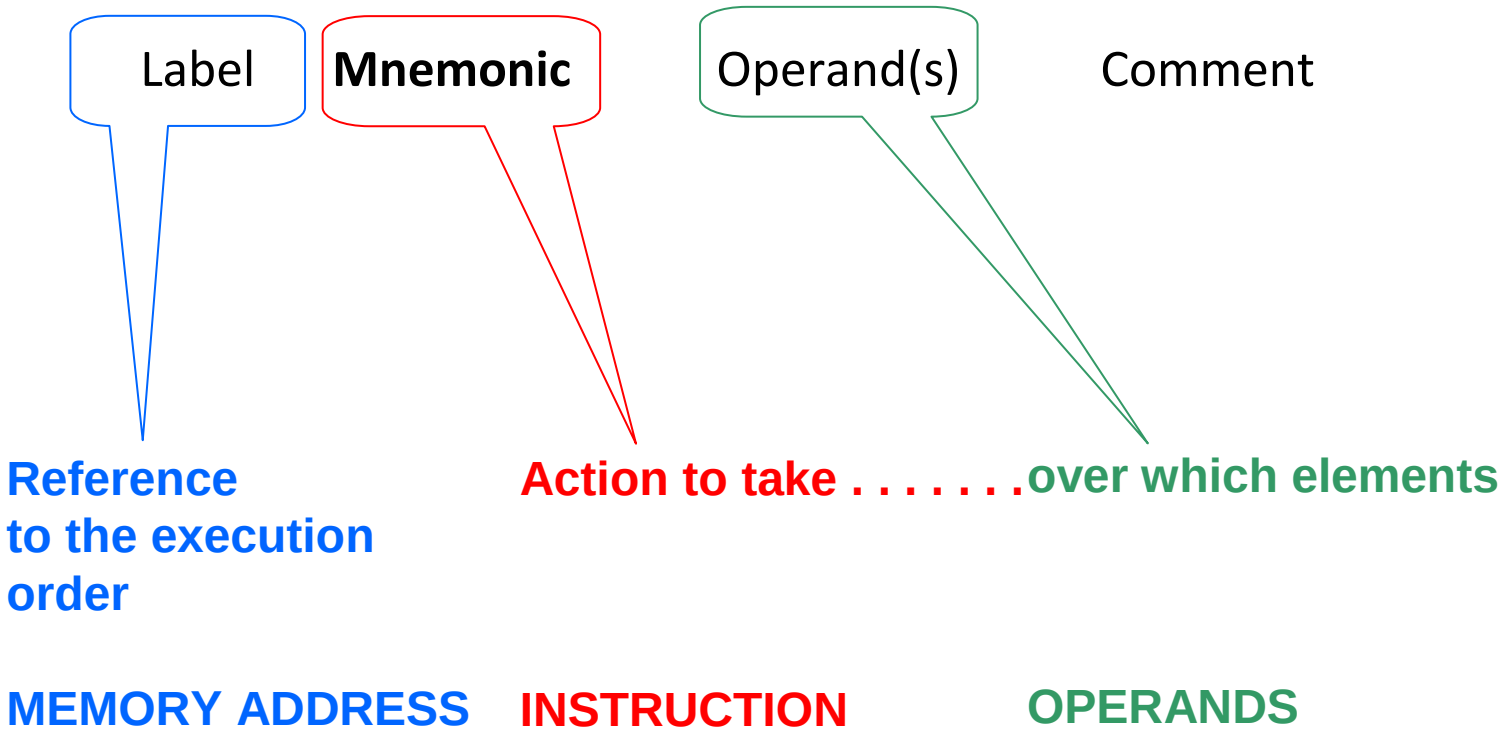
Assembler is a range of SIMPLE orders . . . .

. . . . . which can be executed in the data path.

Fundamentals of Assembler

The Assembler sentences

Each line of the source file may consist of up to four fields:



Assembler Instructions

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)
Boolean	(Set bit, Clear bit, Jump if bit set, Jump if bit clear)
Branching	Control (Jump, **Conditional jumps**) Subroutine (Push, Pull) Interrupt (Int. Return)

Figura del “PIC18F2525/2620/4525/4620 Data Sheet”
Con permiso de MICROCHIP

Assembler Instructions

Manufacturer tables in Datasheet

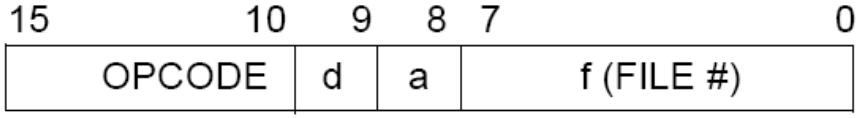
Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
BYTE-ORIENTED OPERATIONS						
ADDWF	f, d, a	Add WREG and f	1	0010 01da0 ffff ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and CARRY bit to f	1	0010 0da ffff ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG 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 1st word f_d (destination) 2nd word	2	1100 ffff ffff ffff 1111 ffff ffff ffff	None	
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	

Figura del "PIC18F2525/2620/4525/4620 Data Sheet"
Con permiso de MICROCHIP

Assembler Instructions

MICROCHIP Classification of Instructions

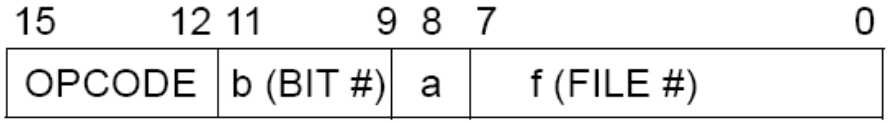
Byte-oriented file register operations



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

MOVWF	**Move W to f**
Syntax:	MOVWF f {,a}
Operands:	0 ≤ f ≤ 255 a ∈ [0,1]
Operation:	(W) → f
Status Affected:	None
Encoding:	0110 111a ffff ffff

Bit-oriented file register operations

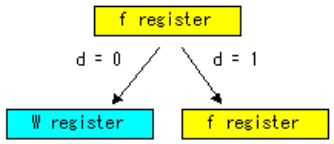


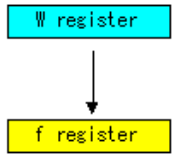
b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

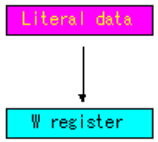
BCF	**Bit Clear f**
Syntax:	BCF f, b {,a}
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [0,1]
Operation:	0 → f<b>
Status Affected:	None
Encoding:	1001 bbba ffff ffff

Assembler Instructions

http://hobby_elec.piclist.com/

MOVF	Move f
Form	**[label]▵MOVF▵f, d** (label is omitable, ▵ shows SPACE code)
Operands	f : Register file addesss (00(00h) to 127(7Fh)) d : Destination select (0 or 1)
Operation	It moves (copy) the contents of the f register. d = 0 : store result in W d = 1 : store result in f Seemed to move from the f register to the f register no meaning, but its being used in the purpose to set a flag. 
Flag	When the result is 0, it sets 1 to the Z flag. When the result is not 0, it sets 0 to the Z flag.
Instruction cycles	1 cycle

MOVWF	Move W to f
Form	**[label]▵MOVWF▵f** (label is omitable, ▵ shows SPACE code)
Operands	f : Register file addesss (00(00h) to 127(7Fh))
Operation	It moves (copy) the contents of the W register to the f register. 
Flag	No change
Instruction cycles	1 cycle

MOVLW	Move literal to W
Form	**[label]▵MOVLW▵k** (label is omitable, ▵ shows SPACE code)
Operands	k : literal field (00(00h) to 255(FFh))
Operation	It sets literal data to the W register. 
Operation	No change
Instruction cycles	1 cycle

NOP	No Operation
Form	**[label]▵NOP** (label is omitable, ▵ shows SPACE code)
Operands	None
Operation	It is moved to the next instruction without processing anything. This instruction is used when it adjusts a processing time as the timer processing, and so on. 
Flag	No change
Instruction cycles	1 cycle

Assembler Instructions

Operands, and the use of LABELS

Literals	**MOVLW**	**0x**F0	hexadecimal
		0d10	decimal
		0b10100101	binary
a, d, f	**MOVWF**	**PORTA**, **A**	
		BANKED	
	BCF	**PORTA**, **BIT5**, A	

What is the value of

- **PORTA**, if it has to indicate the address of the Port A?
- **BIT5**, if it has to indicate that it is the bit 5 we want to clear?

Assembler Instructions

Write a program (CLEANRAM) that clears N positions of RAM memory starting from address M.

N is stored in register W
M is stored in FSR0

You will need:

Code: Write the code! (**HOMEWORK**)

Start: Tell de machine how to locate the code to execute

Stop: Tell the machine what to do when it has finished

Assembler Instructions

CODE

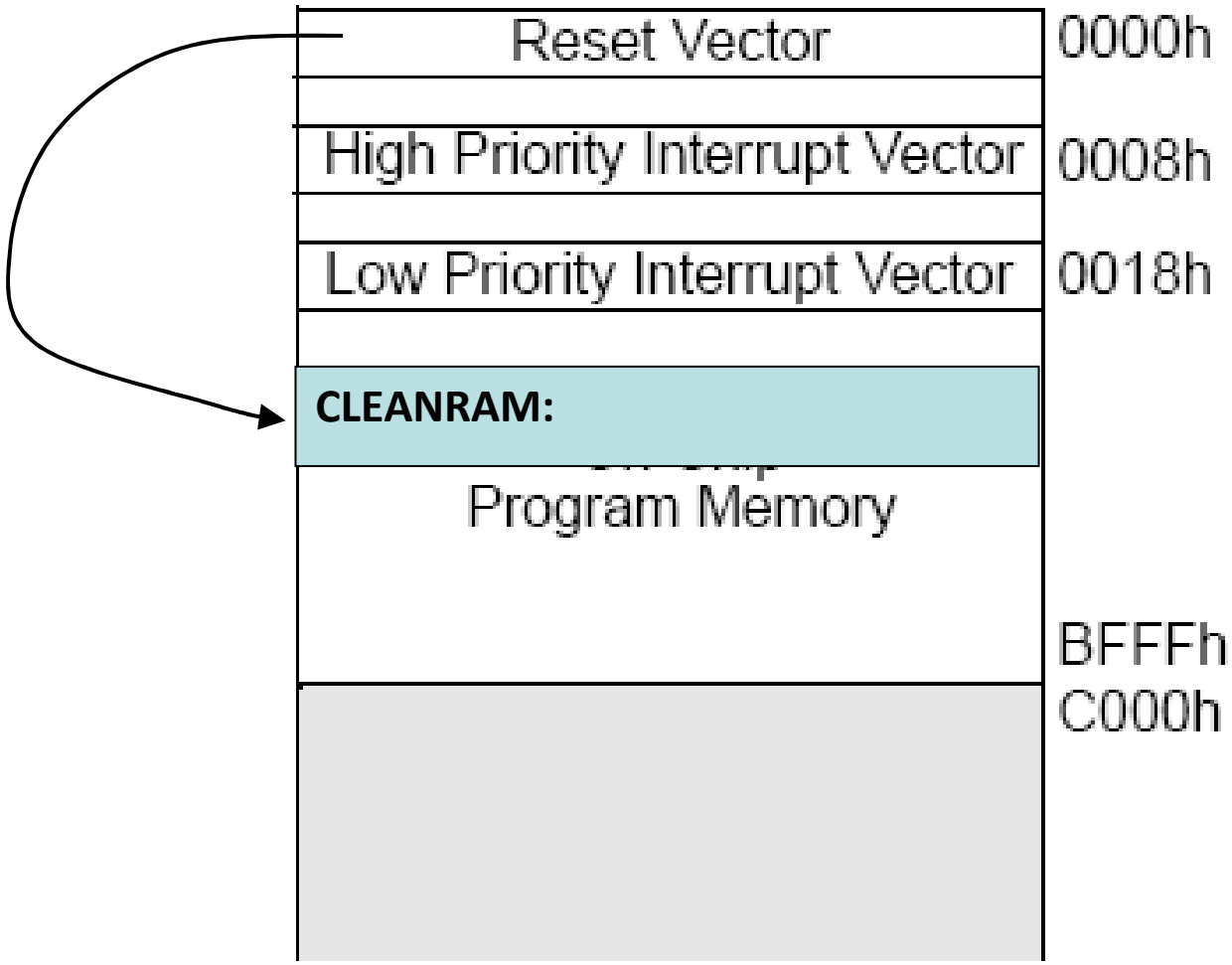
CLEANRAM:

Instructions

(Homework for next day)

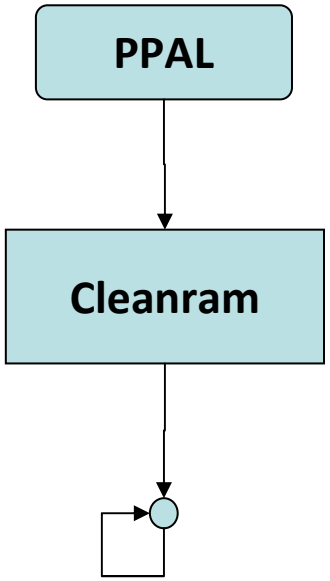
Assembler Instructions

START



Assembler Instructions

STOP



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

Figura del "PIC18F2525/2620/4525/4620 Data Sheet"
Con permiso de MICROCHIP

Assembler Instructions

BRANCHING instructions. UNCONDITIONAL

BRA	**Unconditional Branch**
Syntax:	BRA n
Operands:	-1024 ≤ n ≤ 1023
Operation:	(PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	1101 | 0nnn | nnnn | nnnn
Description:	Add the 2's complement number '2n' to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is a two-cycle instruction.
Words:	1
Cycles:	2

GOTO	**Unconditional Branch**
Syntax:	GOTO k
Operands:	0 ≤ k ≤ 1048575
Operation:	k → PC<20:1>
Status Affected:	None
Encoding: 1st word (k<7:0>) 2nd word(k<19:8>)	1110 | 1111 | $k_7$kkk | kkkk$_0$ 1111 | k_{19}kkk | kkkk | kkkk$_8$
Description:	GOTO allows an unconditional branch anywhere within entire 2-Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.
Words:	2
Cycles:	2

Assembler Instructions

START

We fix the location of instructions in memory by **ASSEMBLER DIRECTIVE** ORG

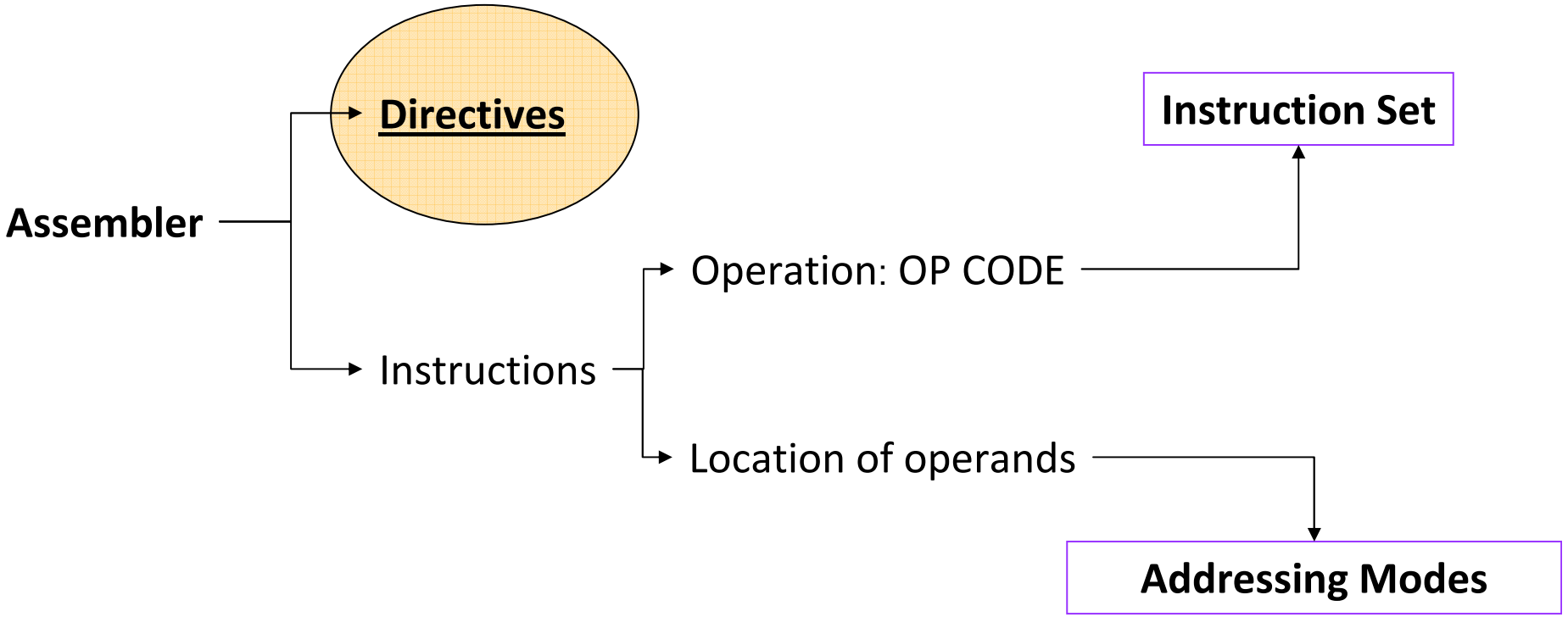
```
ORG 0x0000h
BRA CLEANRAM
```

STOP

Infinite loop

```
End       BRA       End
```

Fundamentals of Assembler



CODE Assembler Directives

#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>.

DATA Assembler Directives

<label> equ <expr>
The equ directive defines a constant. Wherever the label appears in the program, the assembler will replace it with <expr>.

[<label>] data <expr>,[,<expr>, . . ., <expr>]
[<label>] data "<text_string>"[, "<text_string>"] **data "hola mundo"**
This directive initializes one or more words of program memory with data. The data may be in the form of constants, relocatable or external labels, or expressions of any of the above. Each expr is stored in one word. The data may also consist of ASCII character strings, <text_string>, enclosed in single quotes for one character or double quotes for strings. Single character items are placed into the low byte (higher address) of the word, while strings are packed two bytes into a word.

[<label>] db <expr>[, <expr>, . . ., <expr>] **DB** **'1',1,0x03**
This directive reserves program memory words with packed 8-bit values. Multiple expressions continue to fill bytes consecutively until the end of expressions. Should there be an odd number of expressions, the last byte will be zero

[<label> dw <expr> [, <expr>, . . ., <expr>] **DW** **'1',1,0x03**
This directive reserves program memory words for data, initializing that space to specific values.

Figura del “PIC18F2525/2620/4525/4620 Data Sheet”
Con permiso de MICROCHIP

Assembler Instructions

Most Important Instructions:

BRANCHING instructions. CONDITIONAL

Over the ALU status FLAGS

BC	**Branch if Carry**
Syntax:	BC n
Operands:	-128 ≤ n ≤ 127
Operation:	if CARRY bit is ‘1’ (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	1110 0010 nnnn nnnn

BN	**Branch if Negative**
Syntax:	BN n
Operands:	-128 ≤ n ≤ 127
Operation:	if NEGATIVE bit is ‘1’ (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	1110 0110 nnnn nnnn