The University of Texas at Arlington

Lecture 3
PIC Assembly

CSE@UTA

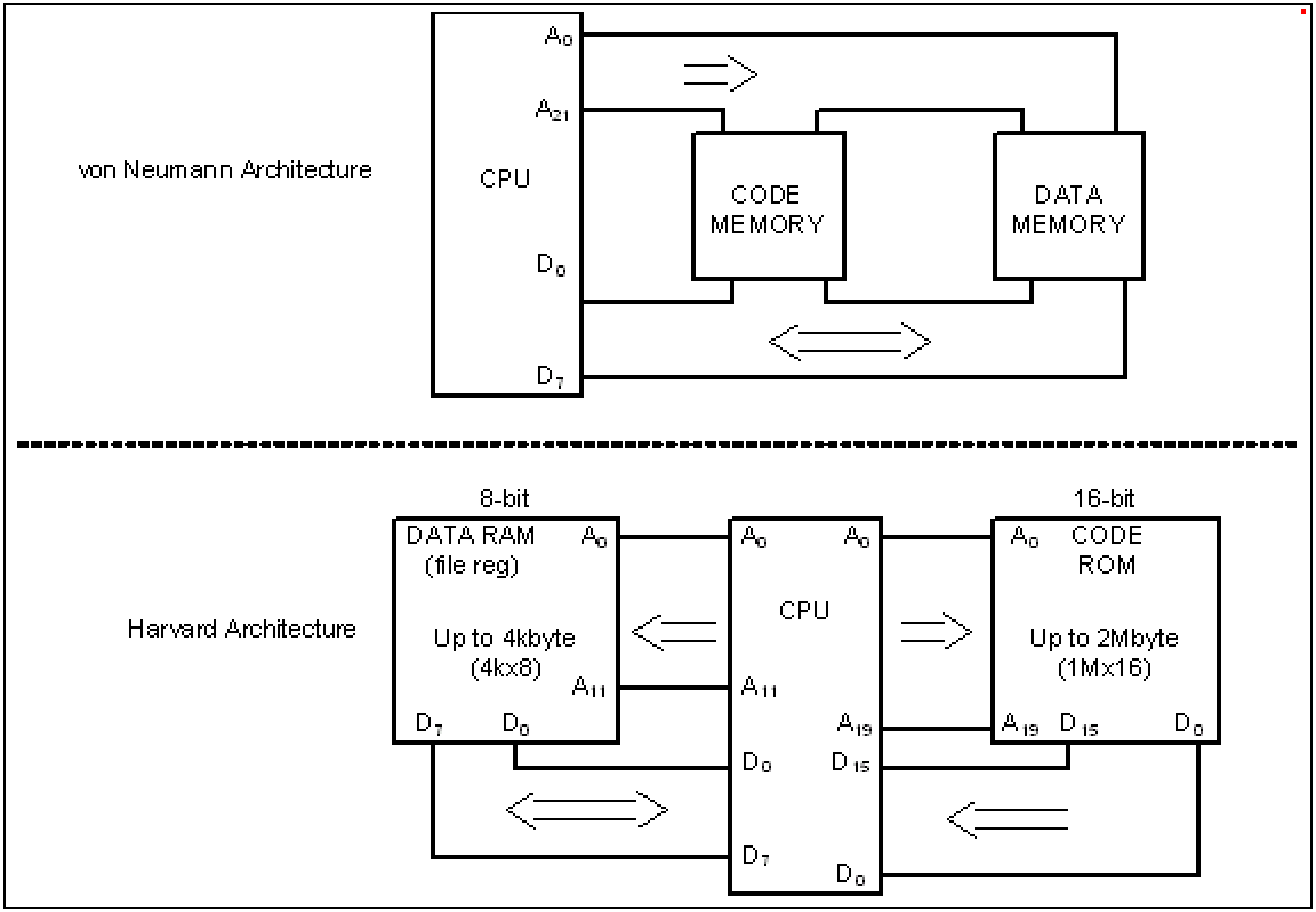
CSE 3442/5442
Embedded Systems I

Based heavily on slides by Dr. Gergely Záruba and Dr. Roger Walker

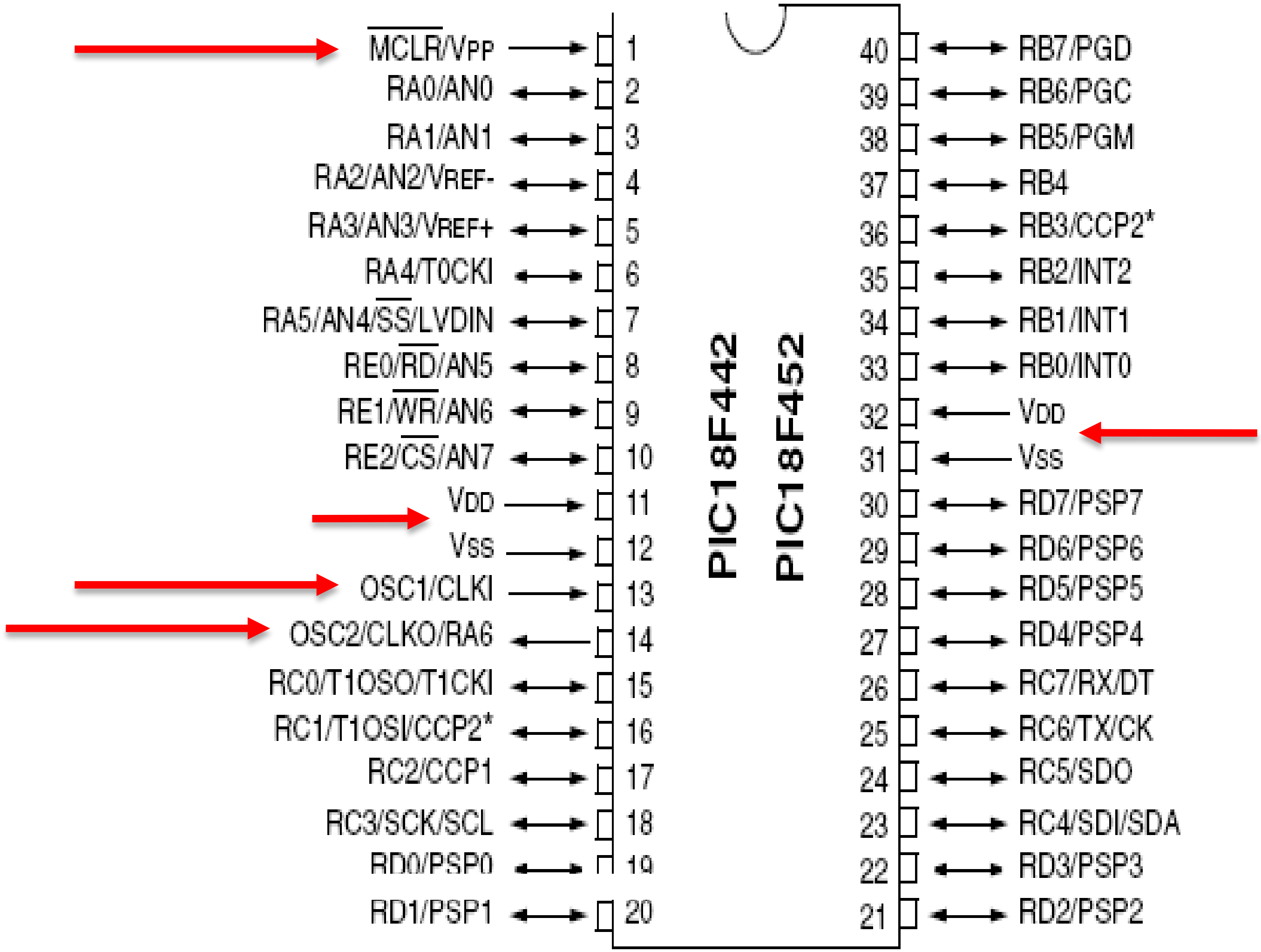
RISC: Reduced Instruction Set Computer

1. Fixed instruction size (2 and 4 bytes in PIC ; ADD, GOTO)
2. Many registers (no need for large stack)
3. Small instruction set – longer code
4. Small clock cycle/instruction
5. Usually Harvard architecture
6. No microcoding; instructions are internally hardwired – can result in 50% reduction in the number of transistors
7. No cross operations between GFR registers

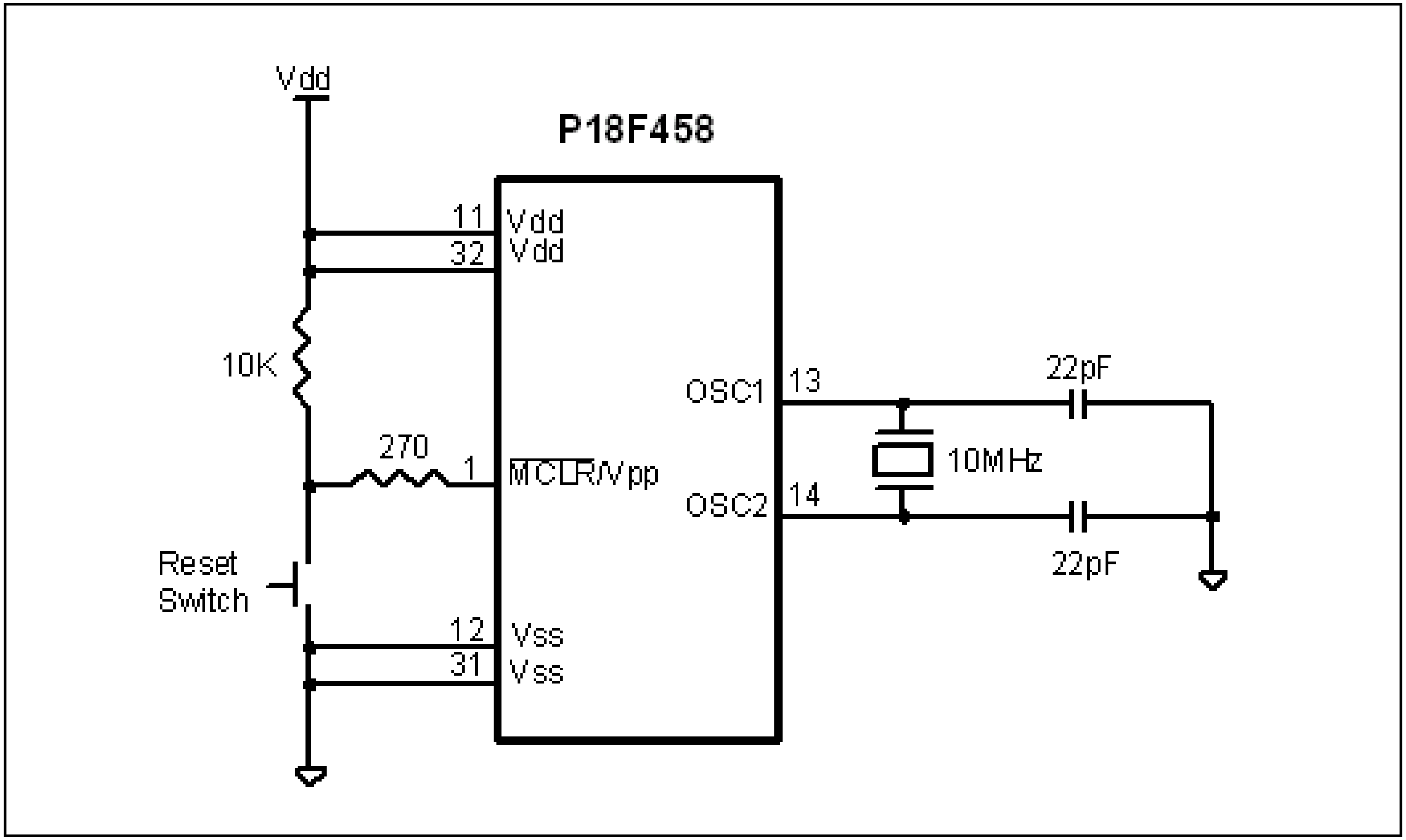
PIC uses Harvard Architecture



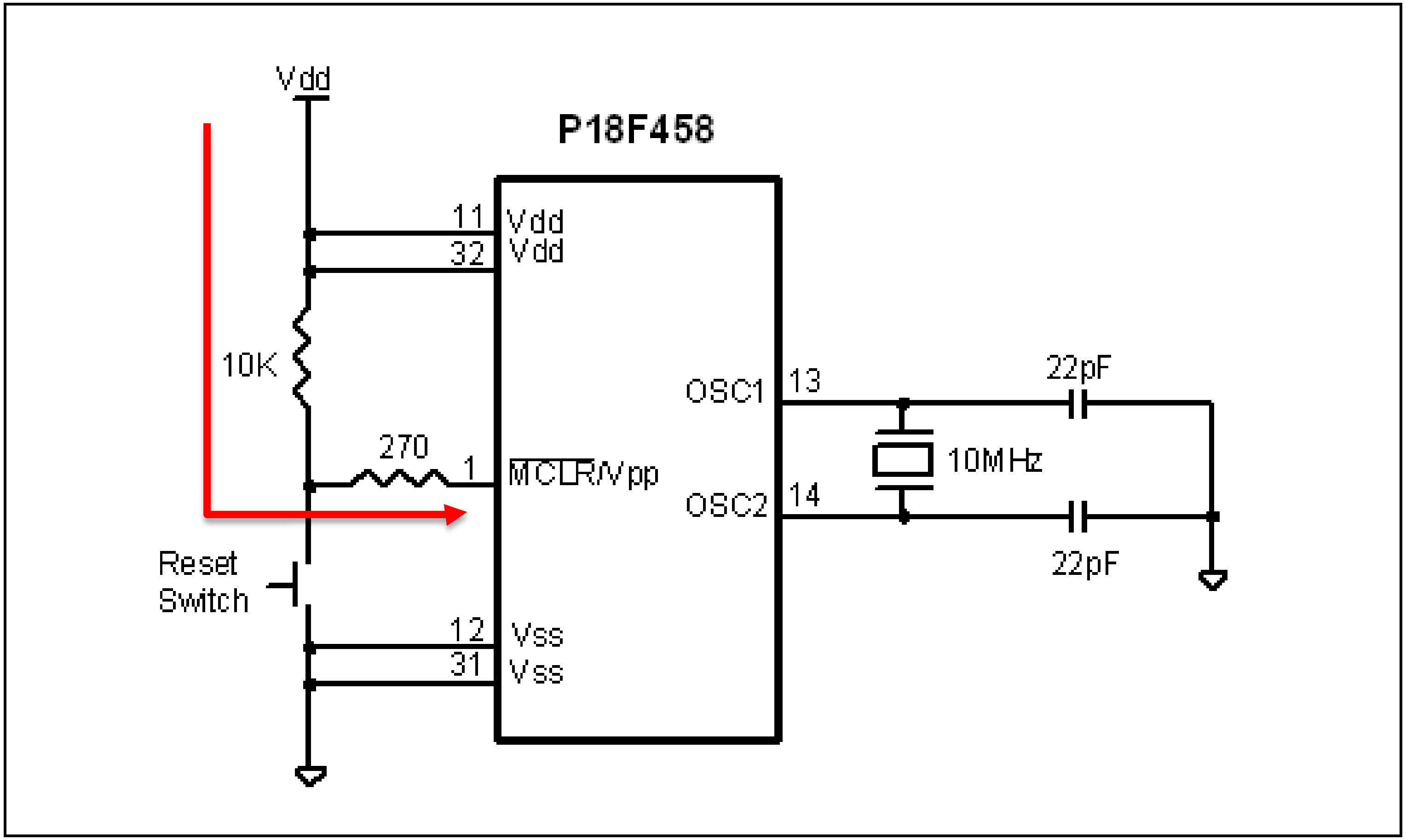
PIC18F452 Pin Diagram



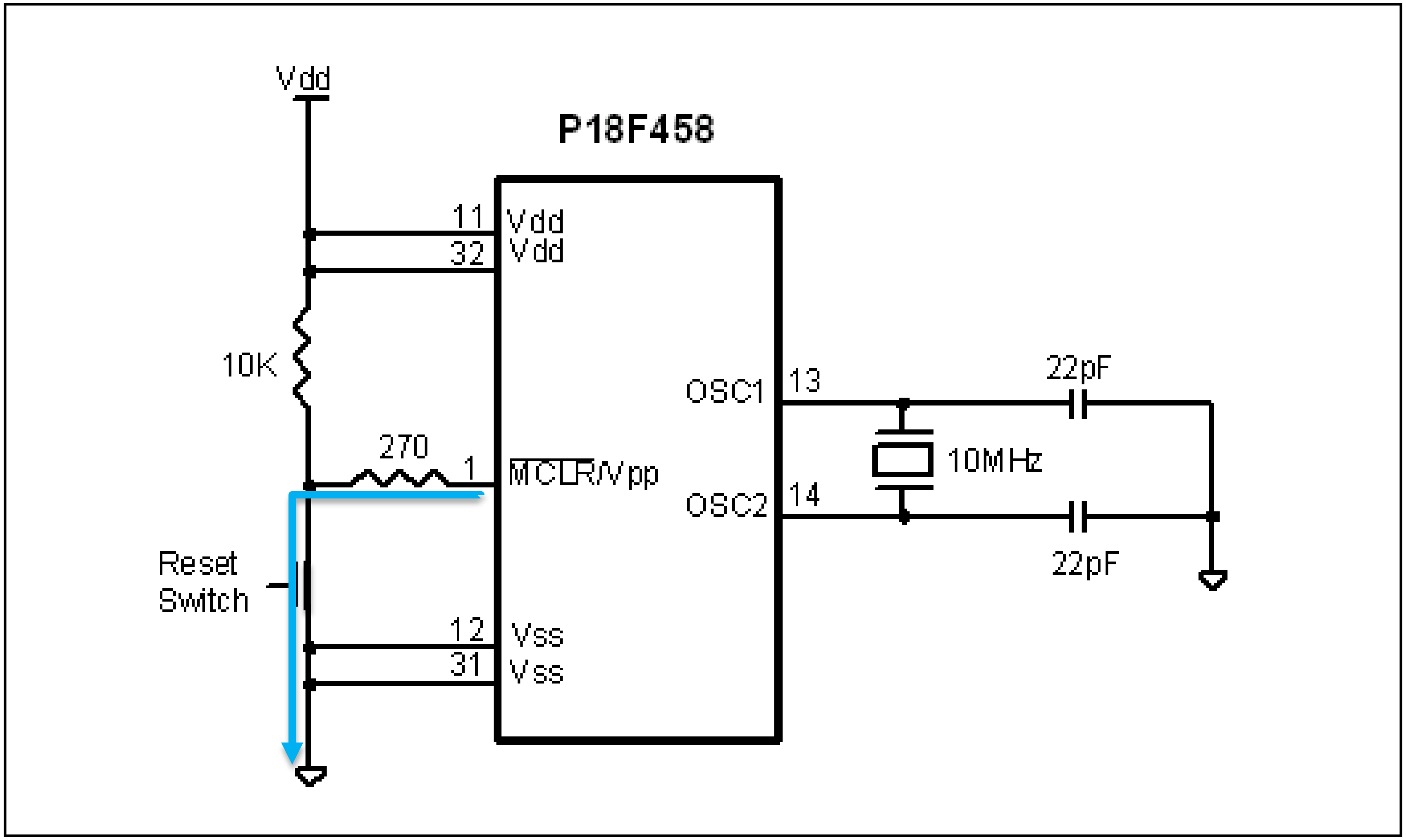
Example - Powering Up PIC18F458



Example - Powering Up PIC18F458



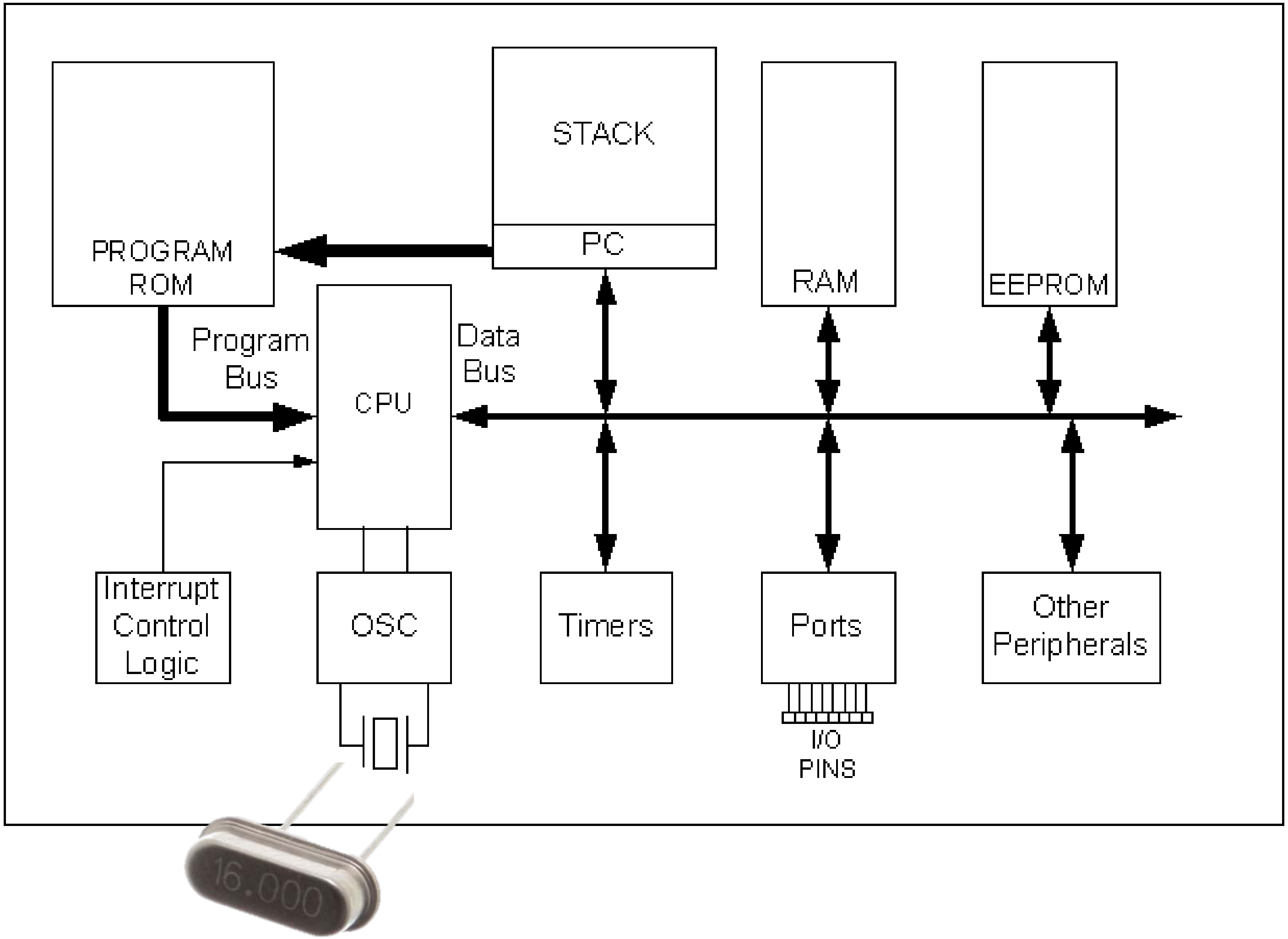
Example - Powering Up PIC18F458



Programs in ROM

- When PIC is powered up (VCC applied to Reset Pin – Chapter 8), the micro-controller begins executing instruction at location 00000h (Reset Vector).
- Use **ORG** statement for this instruction in your code (if programming in assembly).
  - C compiler takes care of creating assembly code having this.

Recap


PROGRAM
ROM
STACK
PC
RAM
EEPROM
Program
Bus
CPU
Data
Bus
Interrupt
Control
Logic
OSC
Timers
Ports
Other
Peripherals
I/O
PINS
16.000

Recap

- Register
  - A place inside the PIC that can be written to, read from, or both (8-bit numbers)

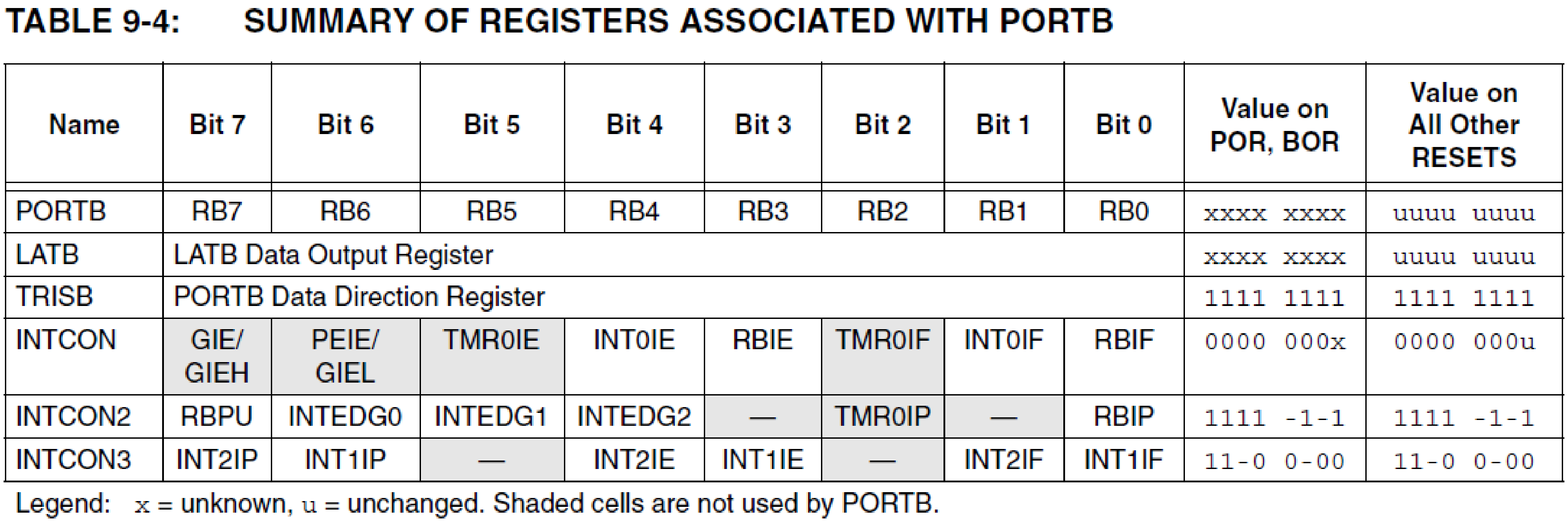
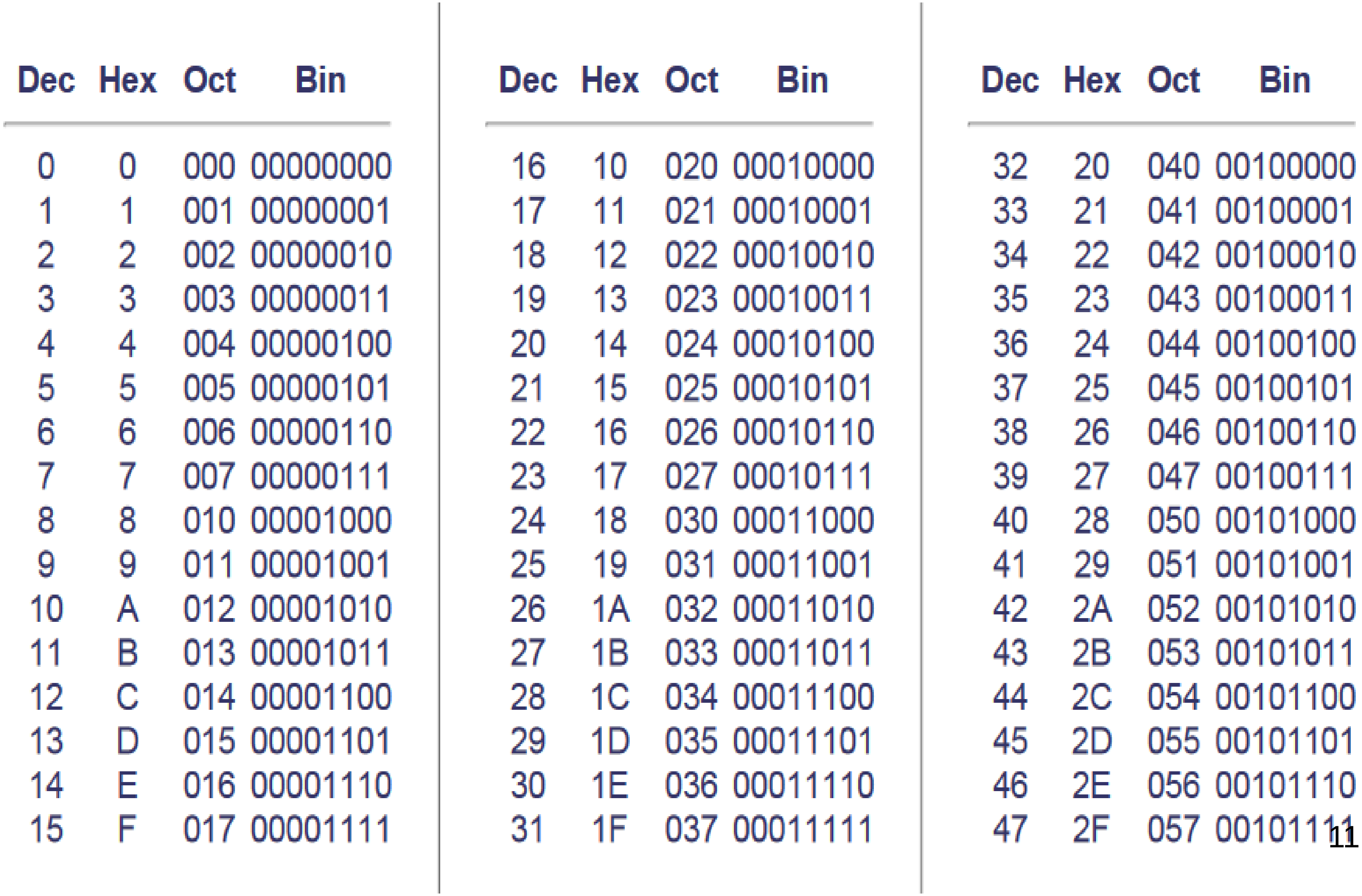


TABLE 9-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
LATB	LATB Data Output Register								xxxx xxxx	uuuu uuuu
TRISB	PORTB Data Direction Register								1111 1111	1111 1111
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	1111 -1-1	1111 -1-1
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	11-0 0-00

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

Dec, Hex, Bin



Dec	Hex	Oct	Bin
0	0	000	00000000
1	1	001	00000001
2	2	002	00000010
3	3	003	00000011
4	4	004	00000100
5	5	005	00000101
6	6	006	00000110
7	7	007	00000111
8	8	010	00001000
9	9	011	00001001
10	A	012	00001010
11	B	013	00001011
12	C	014	00001100
13	D	015	00001101
14	E	016	00001110
15	F	017	00001111
16	10	020	00010000
17	11	021	00010001
18	12	022	00010010
19	13	023	00010011
20	14	024	00010100
21	15	025	00010101
22	16	026	00010110
23	17	027	00010111
24	18	030	00011000
25	19	031	00011001
26	1A	032	00011010
27	1B	033	00011011
28	1C	034	00011100
29	1D	035	00011101
30	1E	036	00011110
31	1F	037	00011111
32	20	040	00100000
33	21	041	00100001
34	22	042	00100010
35	23	043	00100011
36	24	044	00100100
37	25	045	00100101
38	26	046	00100110
39	27	047	00100111
40	28	050	00101000
41	29	051	00101001
42	2A	052	00101010
43	2B	053	00101011
44	2C	054	00101100
45	2D	055	00101101
46	2E	056	00101110
47	2F	057	00101111

Assembler/Compiler Data Formats

- Data Byte Representation
  - hex, decimal, binary, ASCII

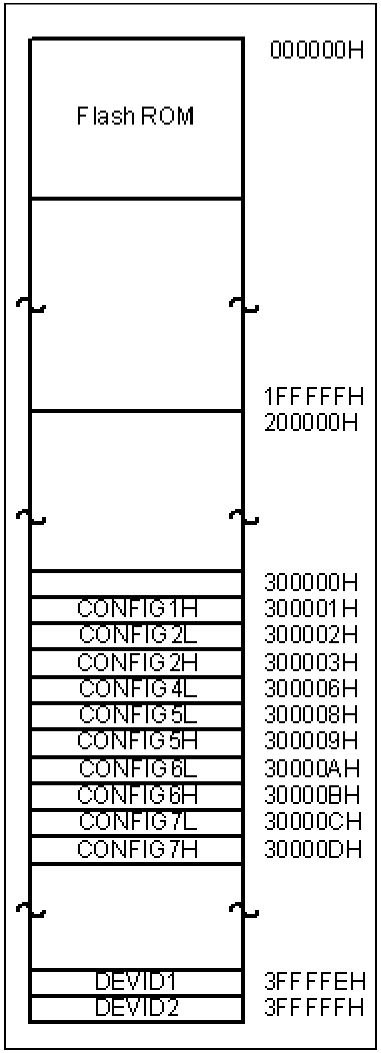
Format	.ASM	.C
Decimal	D’127’ or .127	127
Hex	07F or H’07F’ or 07FH or 0x7F	0x7F
Binary	b‘01111111’	0b01111111

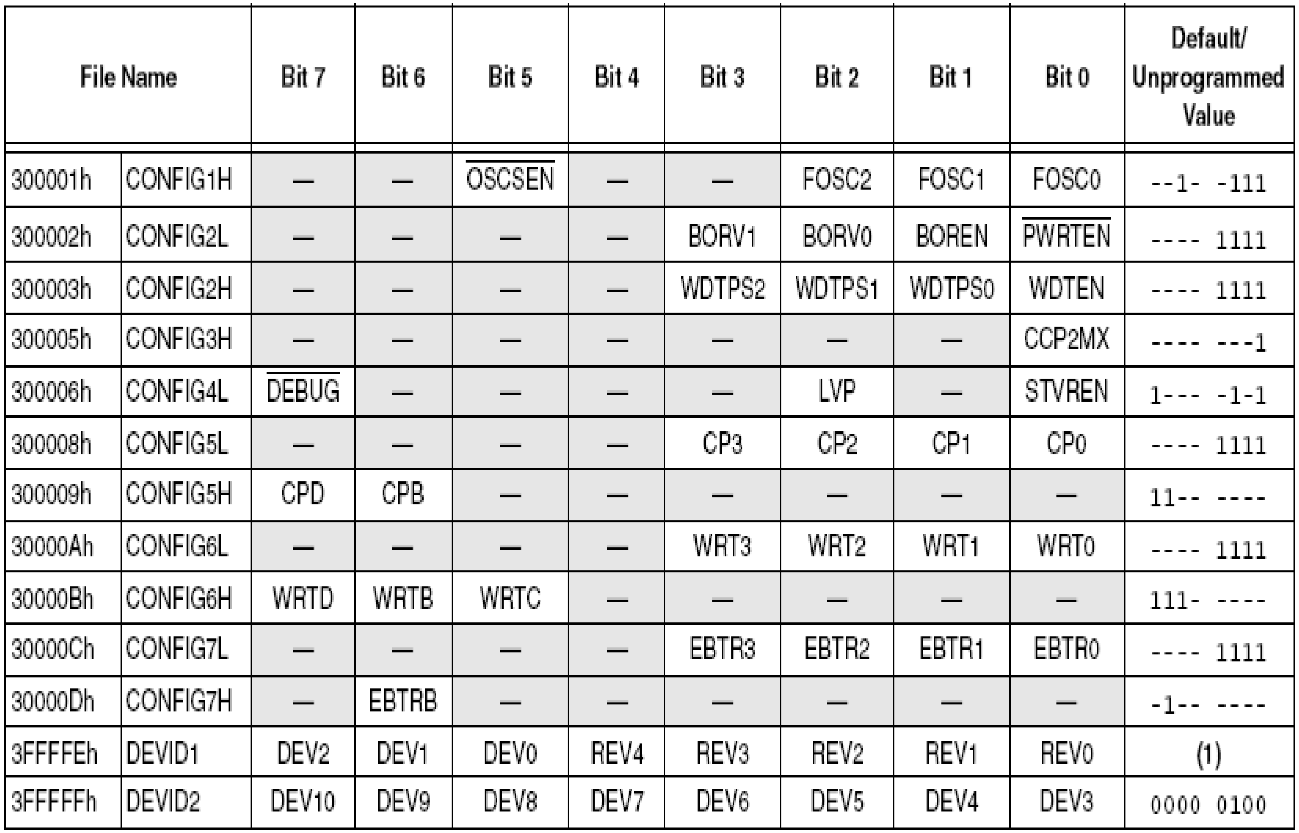
Assembler/Compiler Directives

- **Instructions** (MOVLW, ADDLW, etc.) tell CPU what to do
- **Directives** give directions to the Assembler/Compiler
  - "pseudo-instructions"

- Assembler directives:
  - **EQU** (defining constants), (**SET** is similar but can be reset)
  - **ORG** (origin - explicit address offset operand must be hex)
  - **END** (tells assembler that this is end of code)
  - **LIST** (indicates specific controller, e.g., LIST P=18F452)
  - **#include** (to include libraries associated)
  - **_config directives** – tell assembler what the configuration (stored at 300000H) bits of the target PIC should be
  - **radix** (e.g., radix dec will change to decimal notation; default is hex)

Configuration Registers





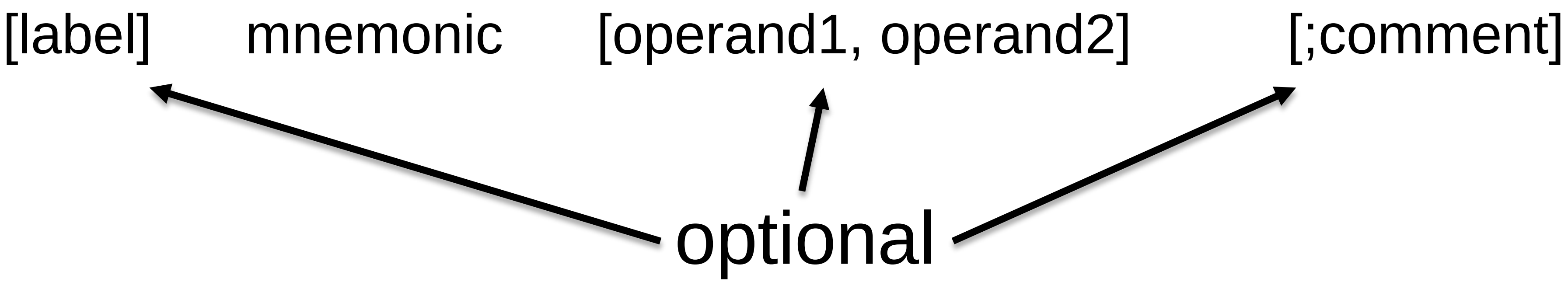
File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default/ Unprogrammed Value
300001h	CONFIG1H	—	—	$\overline{\text{OSCSEN}}$	—	—	FOSC2	FOSC1	FOSC0	--1- -111
300002h	CONFIG2L	—	—	—	—	BORV1	BORV0	BOREN	$\overline{\text{PWRTEN}}$	---- 1111
300003h	CONFIG2H	—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN	---- 1111
300005h	CONFIG3H	—	—	—	—	—	—	—	CCP2MX	---- ---1
300006h	CONFIG4L	$\overline{\text{DEBUG}}$	—	—	—	—	LVP	—	STVREN	1--- -1-1
300008h	CONFIG5L	—	—	—	—	CP3	CP2	CP1	CP0	---- 1111
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	—	11-- ----
30000Ah	CONFIG6L	—	—	—	—	WRT3	WRT2	WRT1	WRT0	---- 1111
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—	111- ----
30000Ch	CONFIG7L	—	—	—	—	EBTR3	EBTR2	EBTR1	EBTR0	---- 1111
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	—	-1-- ----
3FFFFEh	DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0	(1)
3FFFFFh	DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	0000 0100

Table 19-1 from Data Sheet

Assembly Language Structure

[label]   mnemonic   [operand1, operand2]   [;comment]

Assembly Language Structure



[label] mnemonic [operand1, operand2] [;comment]

optional

- **Label**: Can now refer to a line of code by name
- **Mnemonic** (instruction): ADDLW, BNZ, etc.
- **Operand(s)**: Literal, file register location, variable that is manipulated, used, or acted upon
- **Comment**: starts with ; and is ignored by assembler

Instruction Format

Literal operations

15 8	7 0
OPCODE	k (literal)

MOVLW 0x7F

k = 8-bit immediate value

Byte-oriented file register operations

Example Instruction

15 10	9	8	7 0
OPCODE	d	a	f (FILE #)

ADDWF MYREG, W, B

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

Opcode

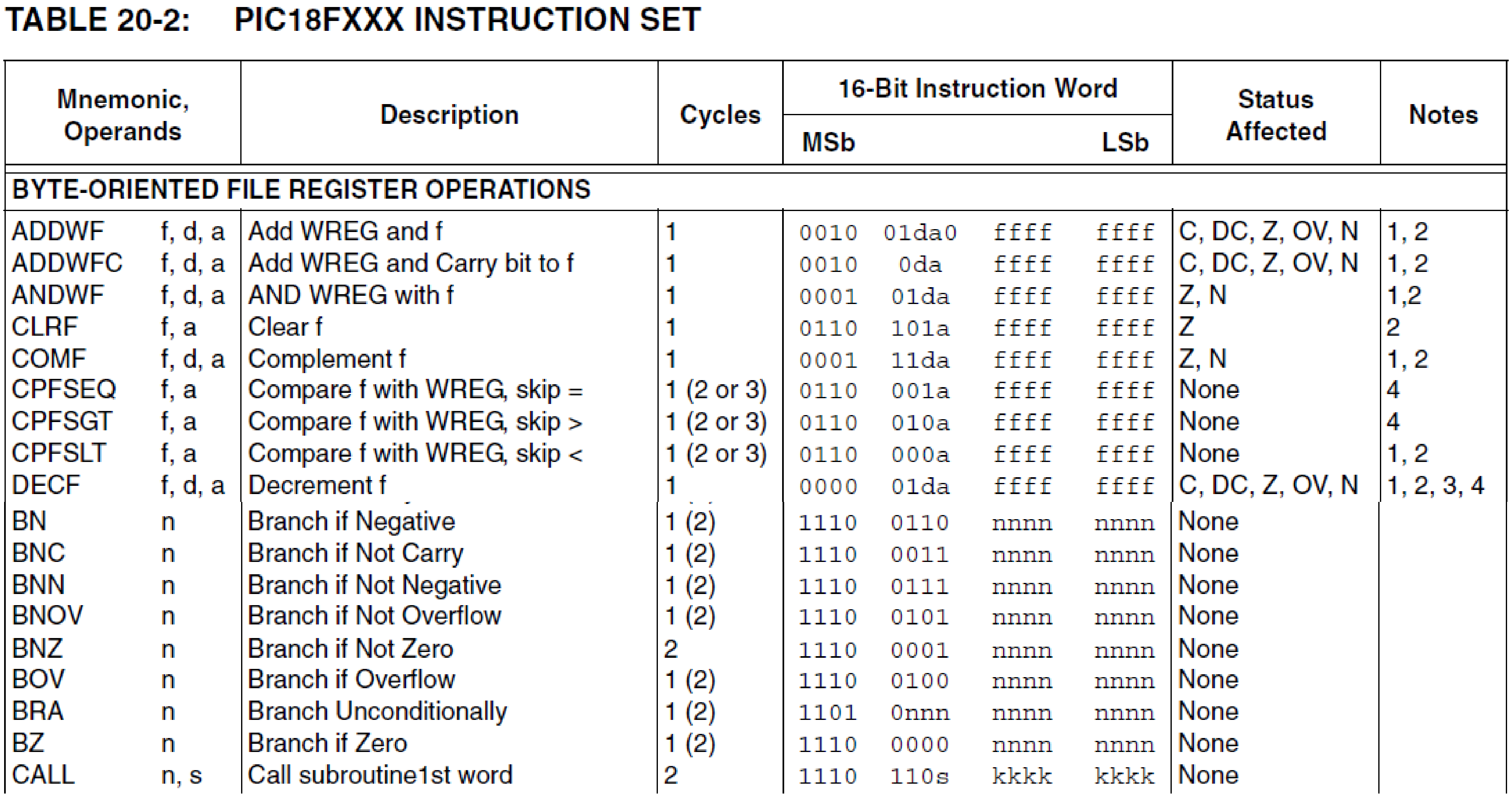


TABLE 20-2: PIC18FXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER 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
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	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	

Opcode

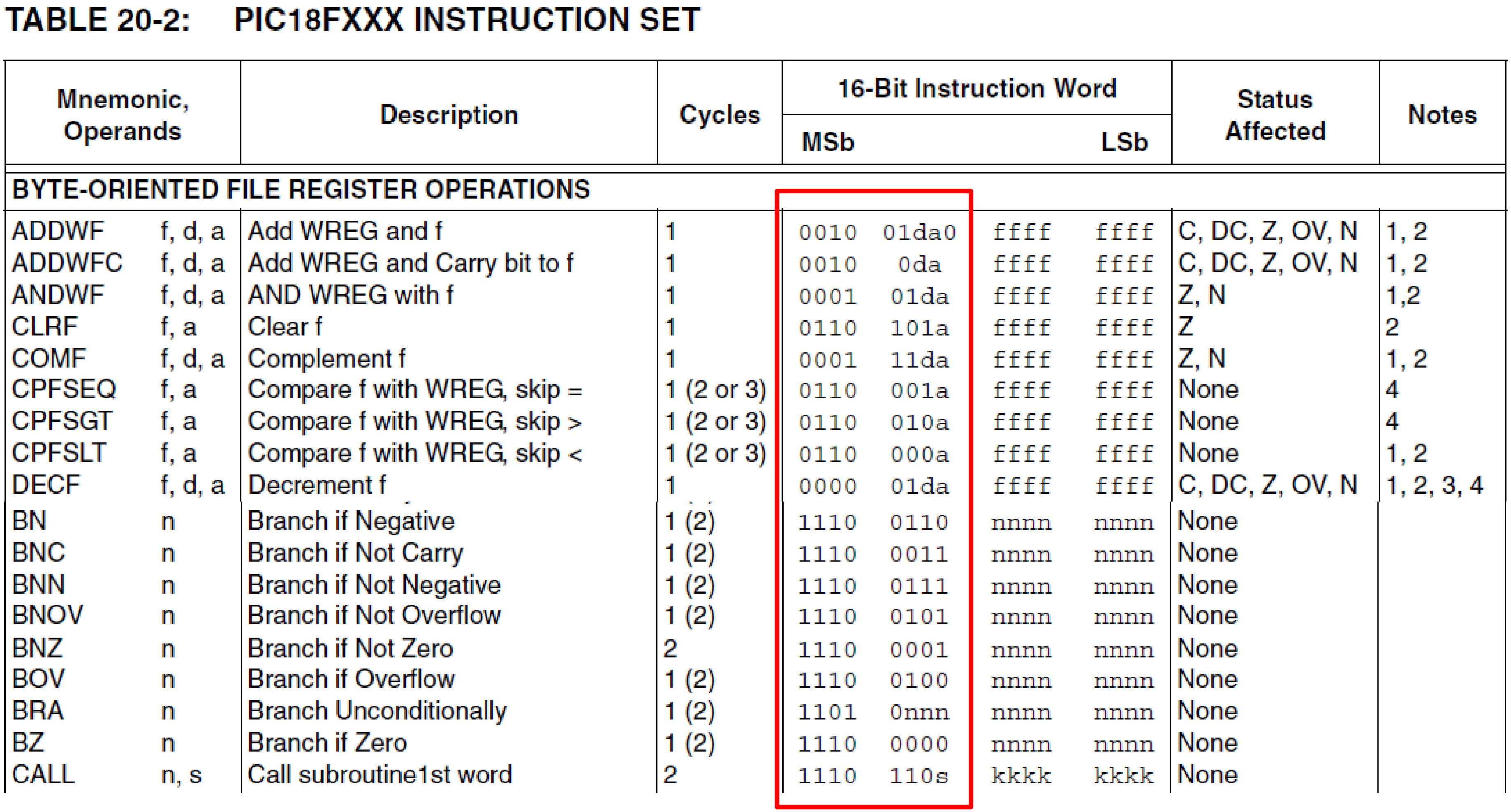


TABLE 20-2: PIC18FXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER 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
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	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	

Assembly Programming sample

```
SUM     EQU     0F7H

        ORG     0H

HERE    MOVLW   0
        MOVWF   SUM
        MOVLW   25H        ;25H → WREG
        ADDLW   0x34       ;+ 34H
        ADDLW   11H        ;+ 11H
        ADDLW   0C1H       ;+ C1H
        ADDLW   25         ;+25H
        ADDLW   D'18'      ;+ 18 decimal
        ADDLW   B'00000110‘ ;+6 dec
        MOVWF   SUM

        MOVLW   SUM
        GOTO    HERE

END
```

Assembly Assembled an Linked

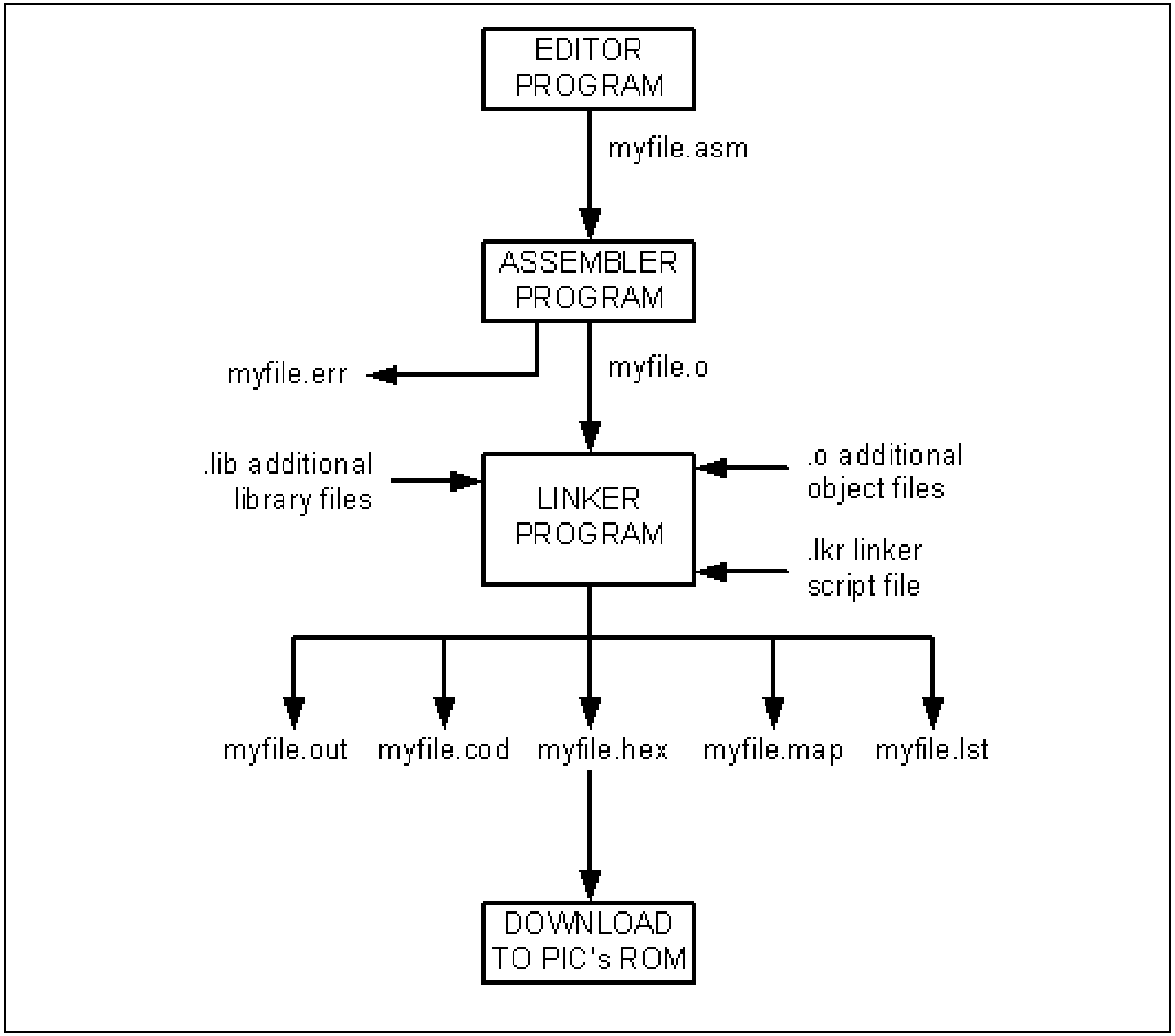
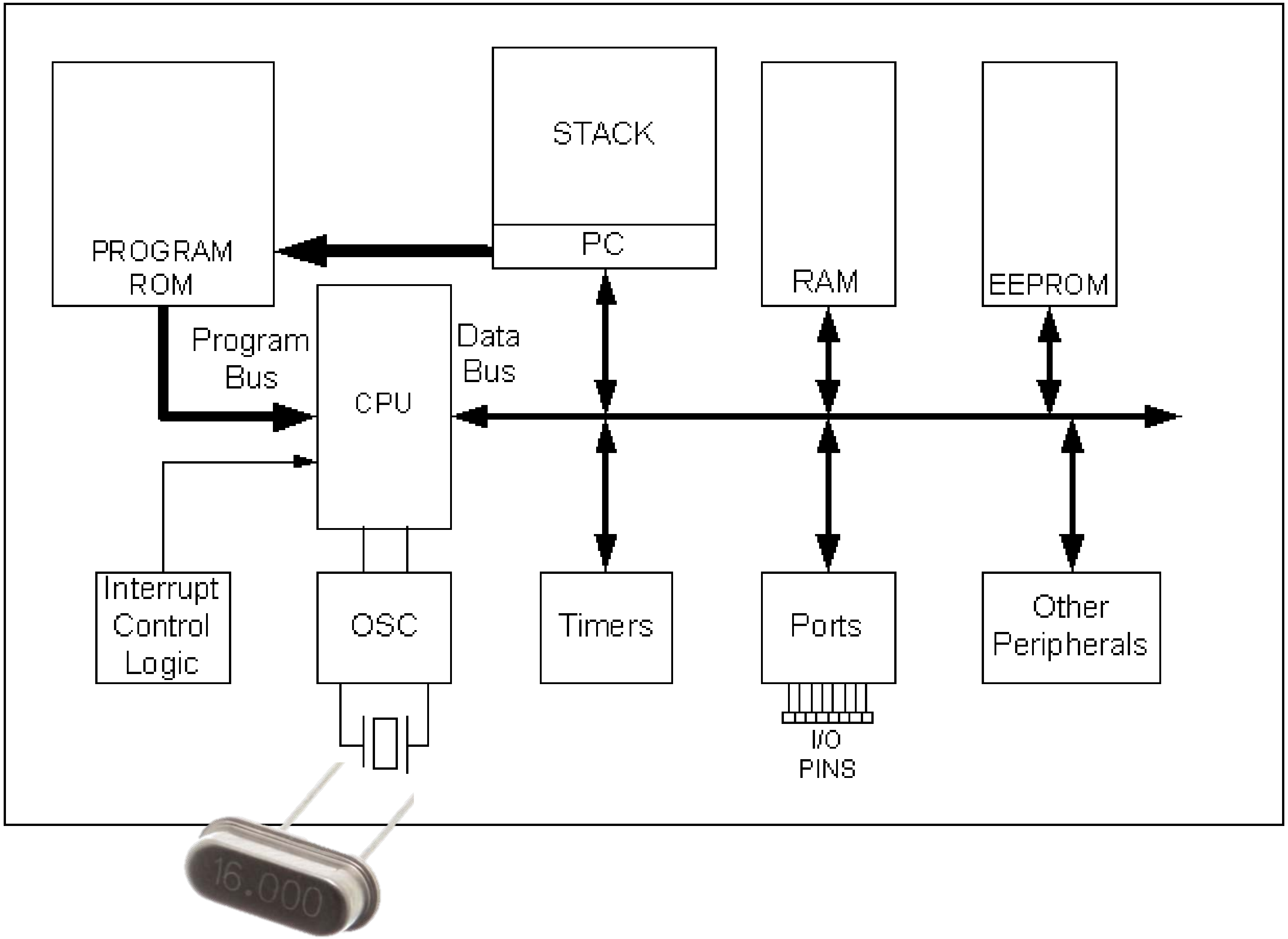
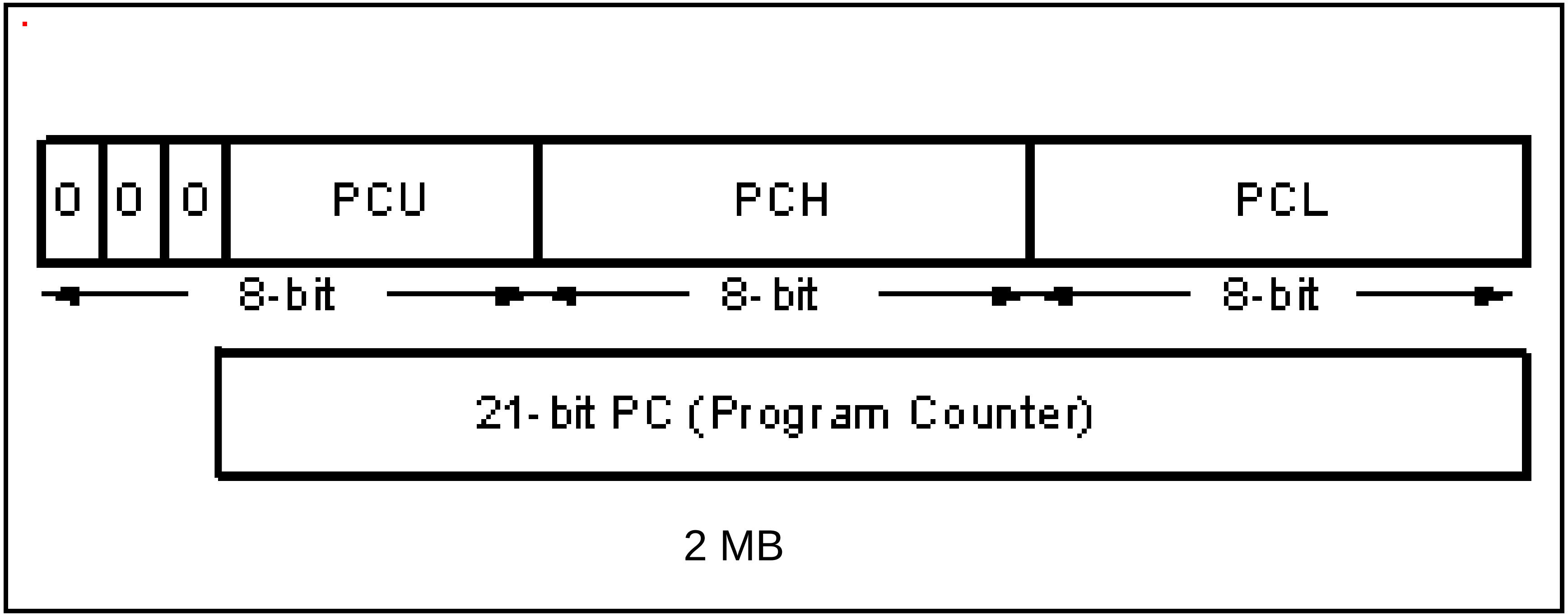


Figure 1-2. Simplified View of a PIC Microcontroller



PIC18 Program Counter



2 MB

Figure 2-9.

21-bit → 000000 to 1FFFFF addresses

PIC18 On-Chip Program ROM Address Range

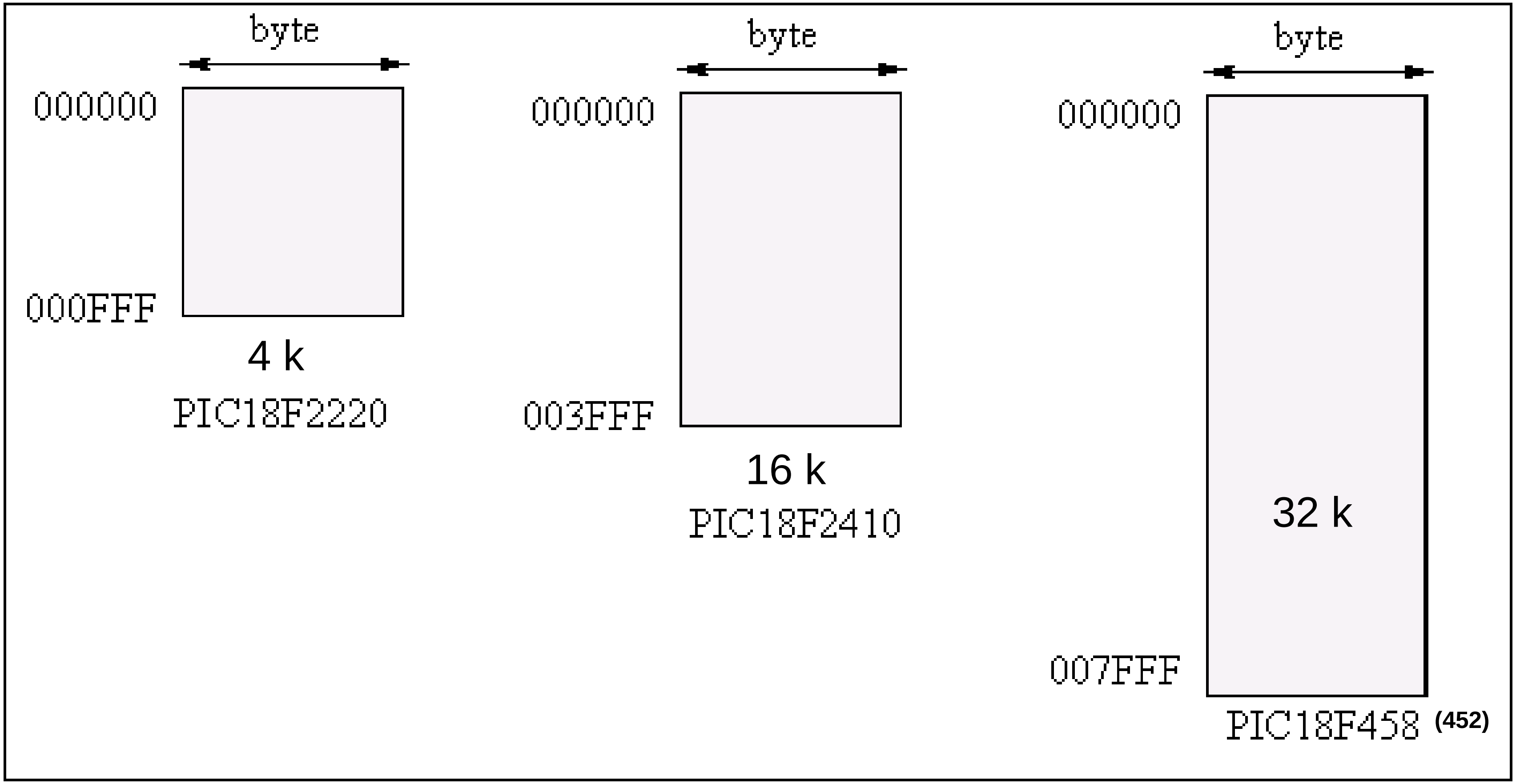
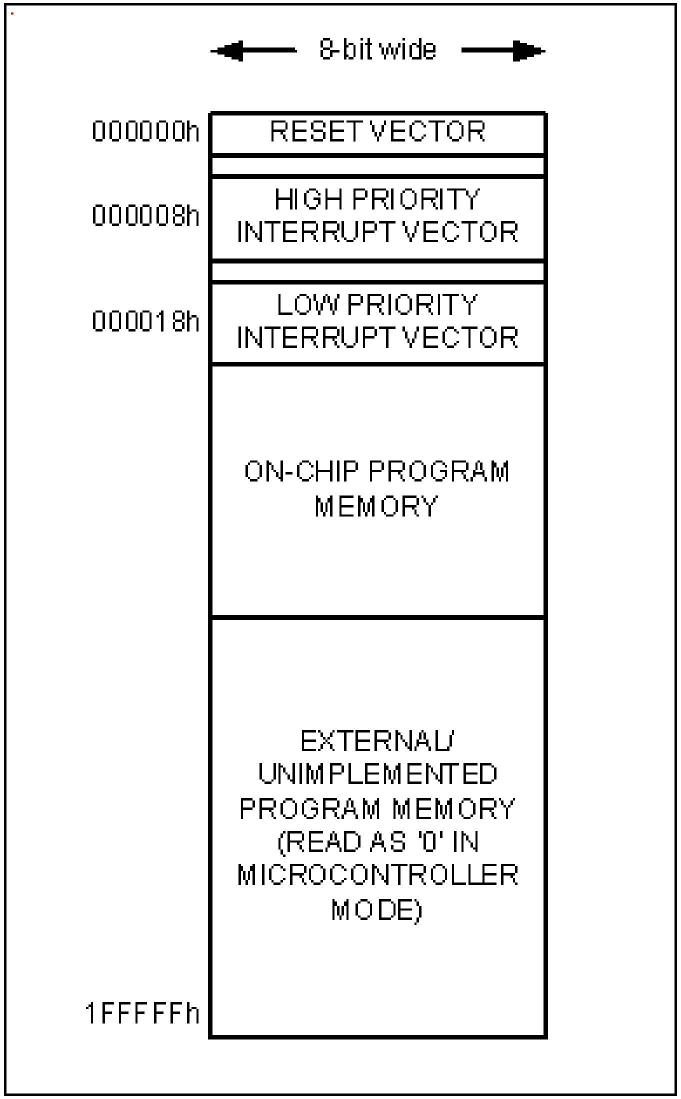


Figure 2-10

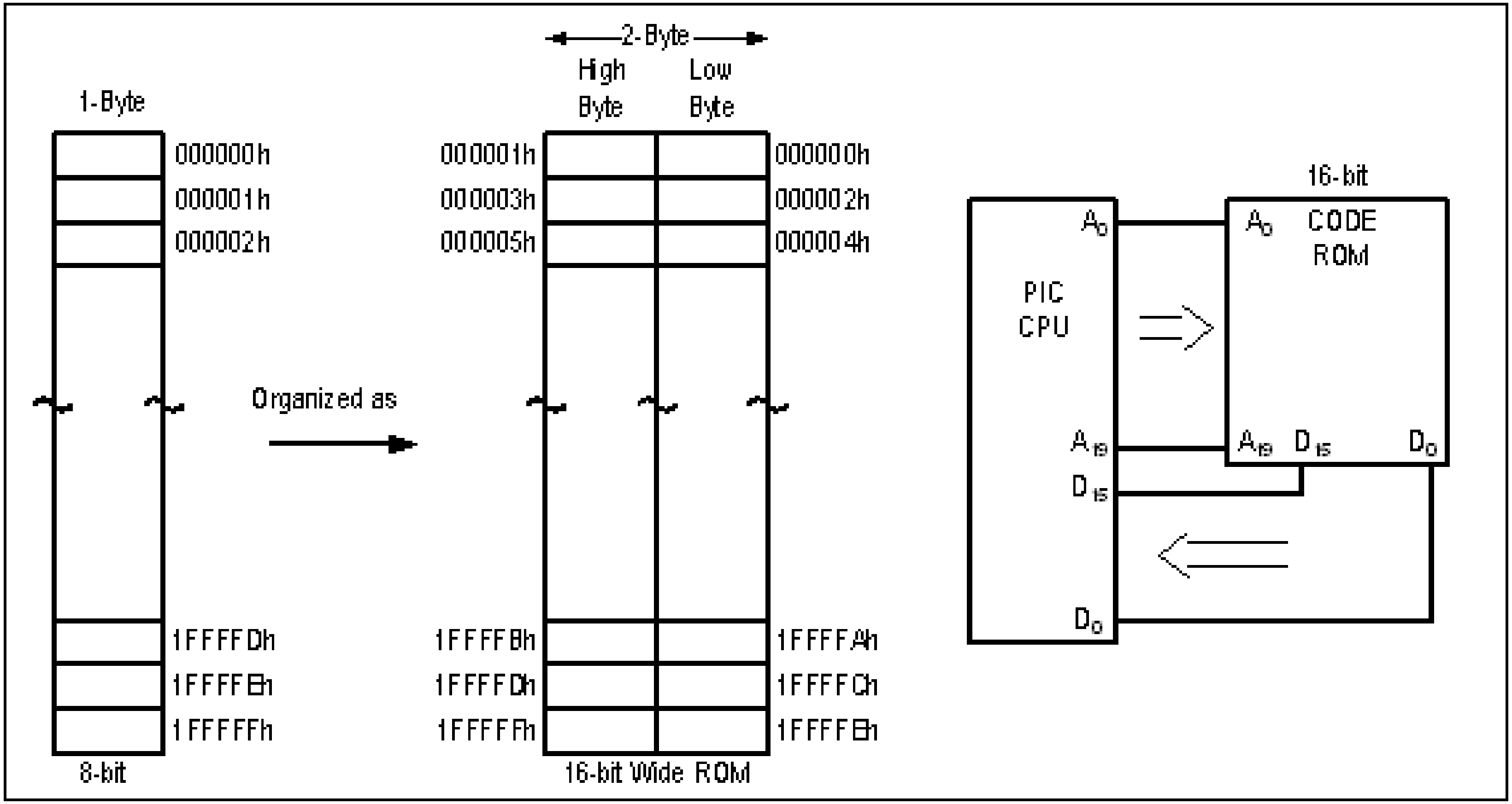
PIC18 Program ROM Space


8-bit wide
000000h
RESET VECTOR
000008h
HIGH PRIORITY INTERRUPT VECTOR
000018h
LOW PRIORITY INTERRUPT VECTOR
ON-CHIP PROGRAM MEMORY
EXTERNAL/ UNIMPLEMENTED PROGRAM MEMORY (READ AS '0' IN MICROCONTROLLER MODE)
1FFFFFh

Assembly Programming sample

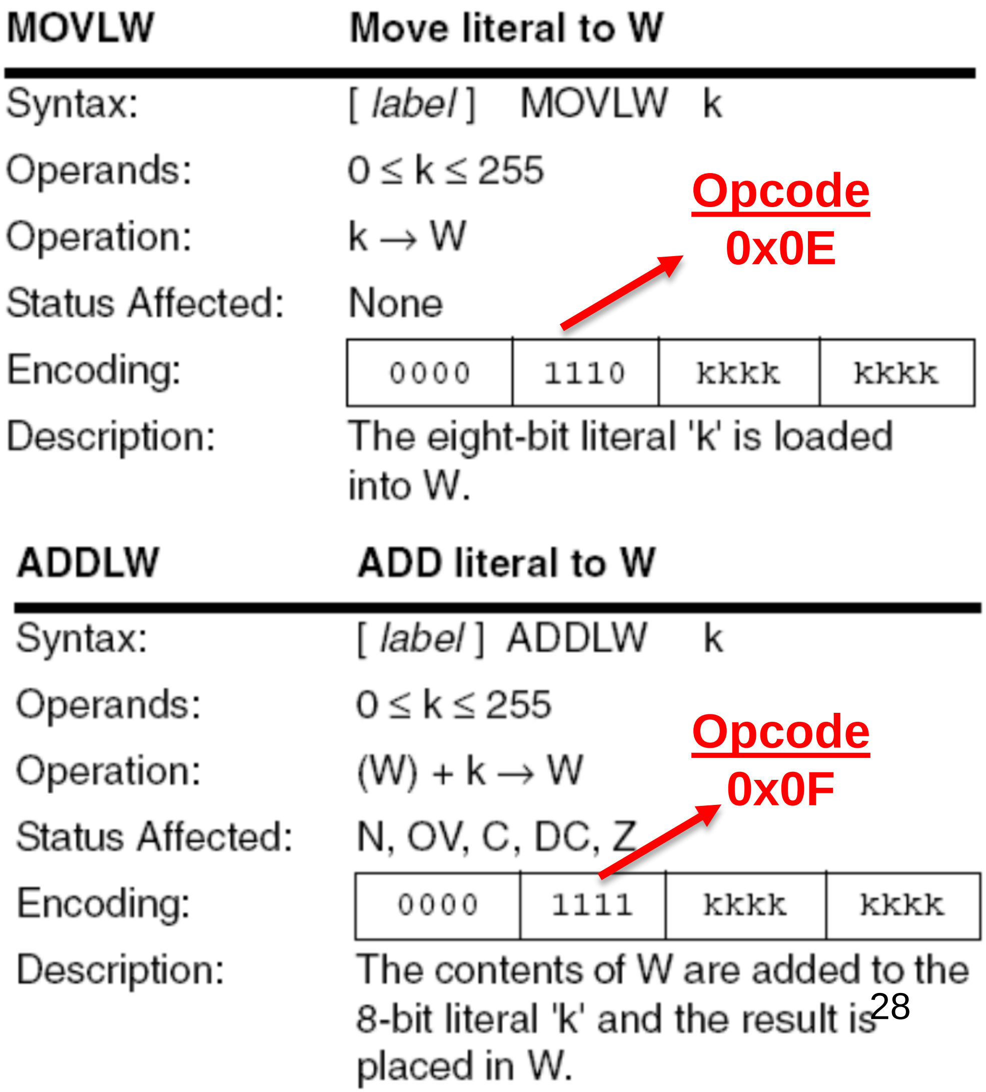
```
      SUM   EQU    0F7H

0x00        ORG    0H

0x00  HERE  MOVLW  0
0x02        MOVWF  SUM
0x04        MOVLW  25H          ;25H → WREG
0x06        ADDLW  0x34         ;+ 34H
0x08        ADDLW  11H          ;+ 11H
0x0A        ADDLW  0C1H         ;+ C1H
0x0C        ADDLW  25           ;+25H
…           ADDLW  D'18'        ;+ 18 decimal
            ADDLW  B'00000110‘ ;+6 dec
            MOVWF  SUM

            MOVLW  SUM
            GOTO   HERE

      END
```

PIC18 Program ROM Width



ROM Contents

0x00	**MOVLW**	**25H**
0x02	**ADDLW**	**34H**
0x04	**ADDLW**	**11H**
0x06		
0x08		
…		



MOVLW **Move literal to W**

Syntax: [*label*] MOVLW k

Operands: $0 \leq k \leq 255$

Operation: $k \rightarrow W$

Status Affected: None

Encoding:

0000	1110	kkkk	kkkk

Description: The eight-bit literal 'k' is loaded into W.

Opcode

0x0E

ADDLW **ADD literal to W**

Syntax: [*label*] ADDLW k

Operands: $0 \leq k \leq 255$

Operation: $(W) + k \rightarrow W$

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

Encoding:

0000	1111	kkkk	kkkk

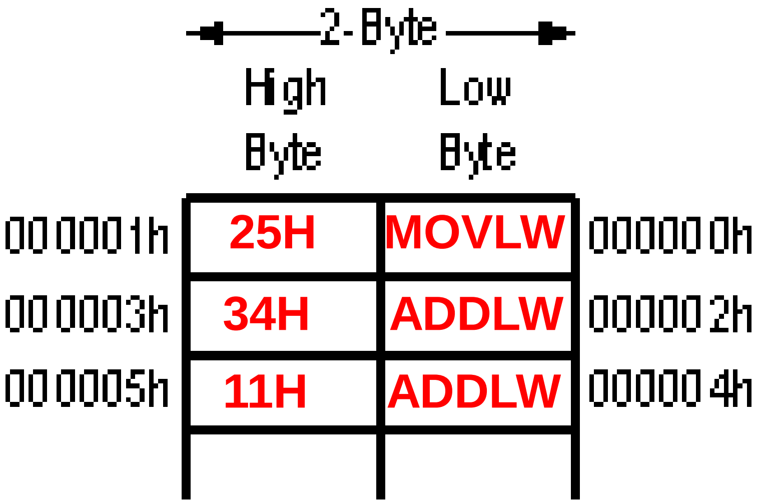
Description: The contents of W are added to the 8-bit literal 'k' and the result is placed in W.

Opcode

0x0F

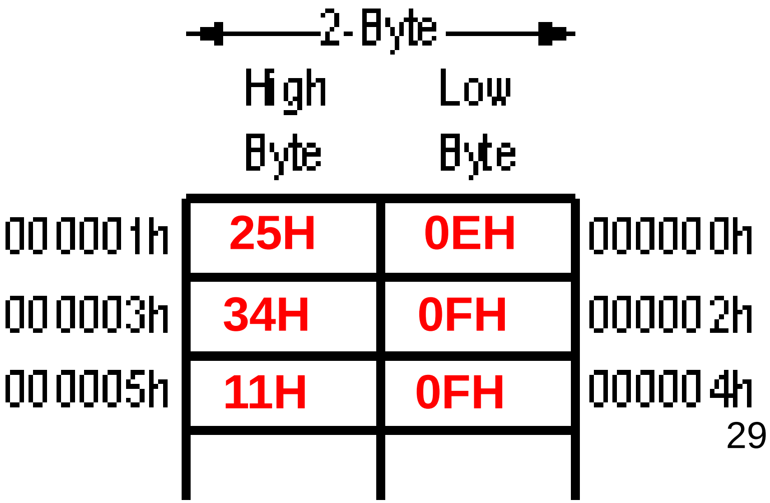
ROM Contents

0x00	**MOVLW**	**25H**
0x02	**ADDLW**	**34H**
0x04	**ADDLW**	**11H**
0x06		
0x08		
…		



2-Byte

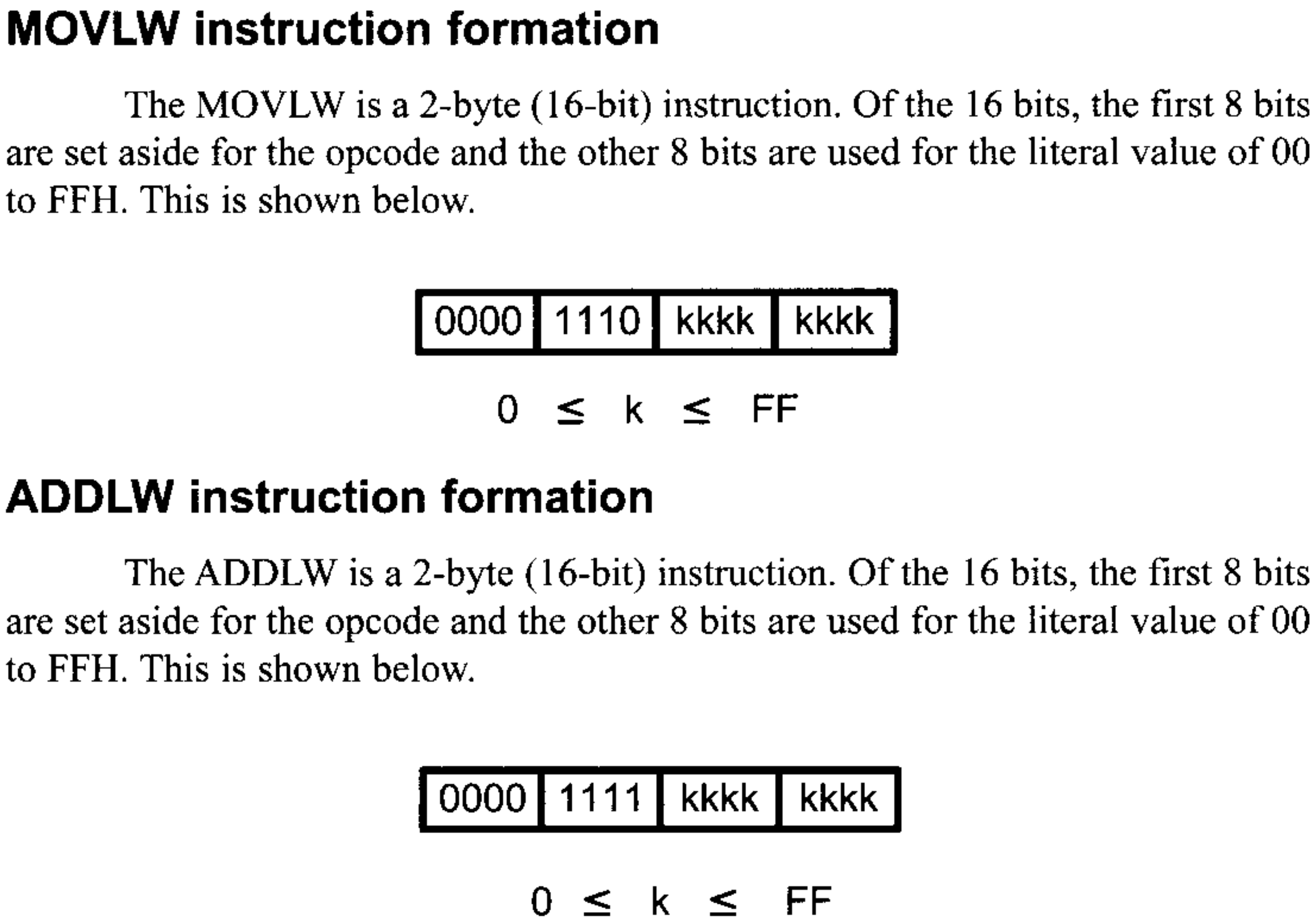
	High Byte	Low Byte	
000001h	**25H**	**MOVLW**	000000h
000003h	**34H**	**ADDLW**	000002h
000005h	**11H**	**ADDLW**	000004h



2-Byte

	High Byte	Low Byte	
000001h	**25H**	**0EH**	000000h
000003h	**34H**	**0FH**	000002h
000005h	**11H**	**0FH**	000004h

ROM Contents



MOVLW instruction formation

The MOVLW is a 2-byte (16-bit) instruction. Of the 16 bits, the first 8 bits are set aside for the opcode and the other 8 bits are used for the literal value of 00 to FFH. This is shown below.

0000	1110	kkkk	kkkk

0 ≤ k ≤ FF

ADDLW instruction formation

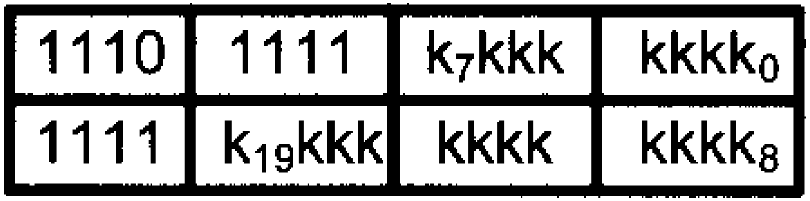
The ADDLW is a 2-byte (16-bit) instruction. Of the 16 bits, the first 8 bits are set aside for the opcode and the other 8 bits are used for the literal value of 00 to FFH. This is shown below.

0000	1111	kkkk	kkkk

0 ≤ k ≤ FF

GOTO and the PC

- GOTO, 4 byte instruction:

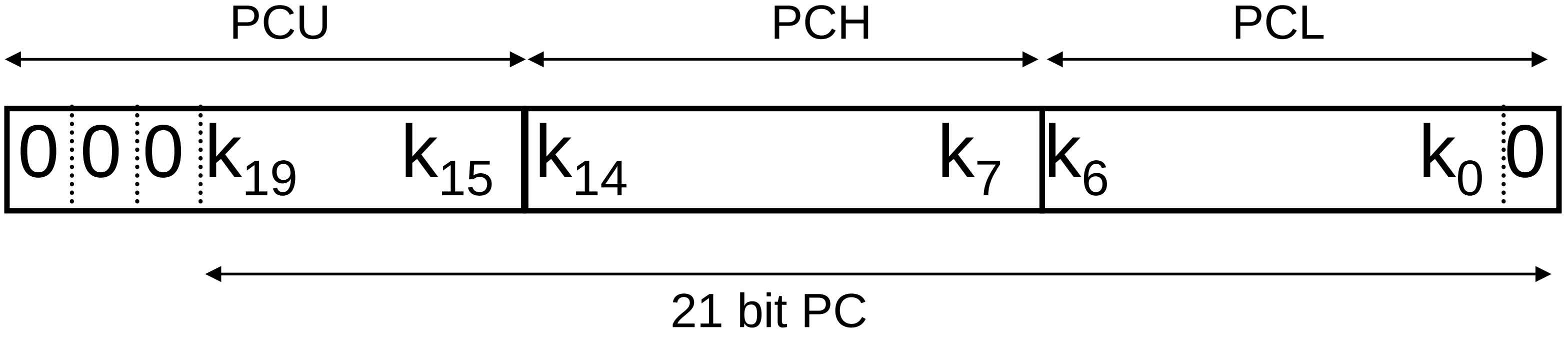


1110	1111	$k_7$kkk	kkkk$_0$
1111	k_{19}kkk	kkkk	kkkk$_8$

$0 \leq k \leq$ FFFFF

0E	07
0F	EF
10	00
11	F0

little endian!



Assembly Programming sample

```
        SUM     EQU     0F7H

0x00            ORG     0H

0x00    HERE    MOVLW   0
0x02            MOVWF   SUM
0x04            MOVLW   25H         ;25H → WREG
0x06            ADDLW   0x34        ;+ 34H
0x08            ADDLW   11H         ;+ 11H
0x0A            ADDLW   0C1H        ;+ C1H
0x0C            ADDLW   25          ;+25H
…               ADDLW   D'18'       ;+ 18 decimal
                ADDLW   B'00000110‘ ;+6 dec
                MOVWF   SUM

                MOVLW   SUM
                GOTO    HERE        ;GOTO 0x00

        END
```

Note

- Chapter 2 for more details of Assembly and Architecture
- Start reading Chapter 3
  - Branching