

EEEC 417/517
Embedded Systems
Cleveland State University

Lab 3

Program Branching, Lookup Tables,
Interrupts, Timer2

Dan Simon
Rick Rarick
Summer 2018

Lab 3 Outline

1. **movwf, movf, movlw** instructions.
2. **xorlw** instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

movwf, movf, movlw instructions

1. movwf f

$(W) \rightarrow f$

Copy contents of **W** register to address **f**

2. movf f, W

$(f) \rightarrow W$

Copy contents of address **f** to **W** register

3. movlw k

$k \rightarrow W$

Copy constant (literal) to **W** register

Opcode

for

movf

movwf

movlw

Page 136

in

data sheet

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

movwf, movf, movlw

Instruction Summary:

Page 140 in data sheet

movf file_reg, F

uses the Z flag (discussed
later) to test whether

file_reg = 0

MOVF	Move f
Syntax:	[<i>label</i>] MOVF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$
Operation:	$(f) \rightarrow (\text{destination})$
Status Affected:	Z
Description:	The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected.

MOVLW	Move Literal to W
Syntax:	[<i>label</i>] MOVLW k
Operands:	$0 \leq k \leq 255$
Operation:	$k \rightarrow (W)$
Status Affected:	None
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

MOVWF	Move W to f
Syntax:	[<i>label</i>] MOVWF f
Operands:	$0 \leq f \leq 127$
Operation:	$(W) \rightarrow (f)$
Status Affected:	None
Description:	Move data from W register to register 'f'.

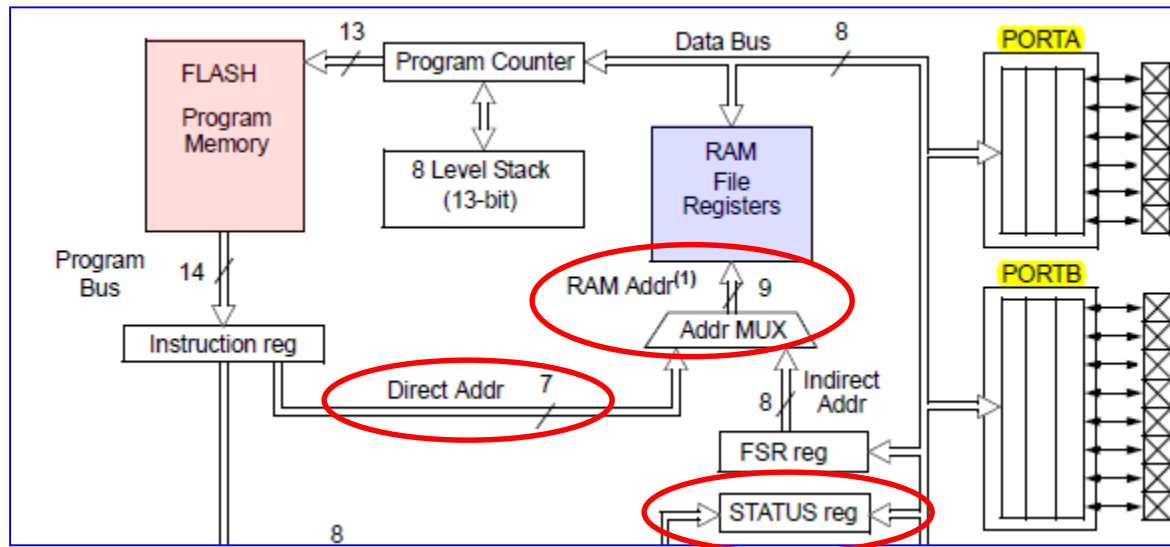
Data memory addresses in instructions

Recall: Data addresses require 9 bits, but instructions contain only 7-bit addresses (direct addressing). Higher-order bits are from the register pointer bits, STATUS<RP1:RP0>.

Later in course: indirect addressing uses the STATUS<IRP> bit.

movf **f**, d → 00 1000 **dfff ffff**

7 of the 9 address bits



Path of 2 bits from STATUS to MUX not shown

movwf, movf instructions

Instructions such as **movwf** and **movf** get the remaining two address bits from the STATUS register.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)

1 = Bank 2, 3 (100h - 1FFh)

0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0:** Register Bank Select bits (used for direct addressing)

11 = Bank 3 (180h - 1FFh)

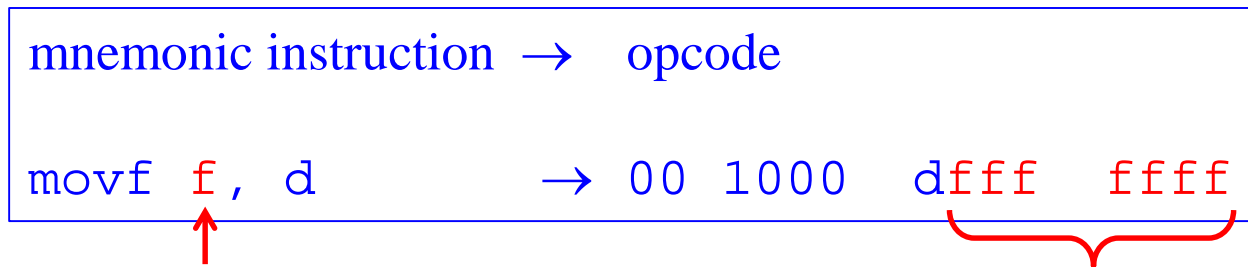
10 = Bank 2 (100h - 17Fh)

01 = Bank 1 (80h - FFh)

00 = Bank 0 (00h - 7Fh)

Each bank is 128 bytes

Unintended results related to assembly instructions



register name or address

7 of the 9 address bits

Note 1: The 'f' operand is interpreted by the assembler as an **address**.

movf PORTC, W ⇔ movf 0x07, W

movf TRISC, W ⇔ movf 0x87, W

File Address		File Address		File Address		File Address	
Indirect addr. ^(*)	00h	Indirect addr. ^(*)	128 80h	Indirect addr. ^(*)	256 100h	Indirect addr. ^(*)	384 180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h

Unintended results related to assembly instructions

movf f, d → 00 1000 dfff ffff

7 of the 9 address bits

Note 2: The assembler uses right-most 7 bits for the address.

File Address	File Address	File Address	File Address
Indirect addr.(*) 00h	128 80h	256 100h	384 180h
TMR0 01h	OPTION_REG 81h	TMR0 101h	OPTION_REG 181h
PCL 02h	PCL 82h	PCL 102h	PCL 182h
STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h
FSR 04h	FSR 84h	FSR 104h	FSR 184h
PORTA 05h	TRISA 85h	105h	185h
PORTB 06h	TRISB 86h	PORTB 106h	TRISB 186h
PORTC 07h	TRISC 87h	107h	187h

movf PORTC, W
 movf 0x07, W
 0x07 = 0 0000 0111

9 address bits

movf TRISC, W
 movf 0x87, W
 0x87 = 0 1000 0111

9 address bits

movf 0x107, W
 0x107 = 1 0000 0111

9 address bits

opcode same for all: 00 1000 0000 0111

movf instruction

movf f , d → 00 1000 dfff ffff

If bank 0 is selected:

movf	PORTC ,	W	Copies the contents of PORTC to W
movf	7 ,	W	Copies the contents of PORTC to W
movf	.7 ,	W	Copies the contents of PORTC to W
movf	TRISC ,	W	Copies the contents of PORTC to W (*)
movf	0x87 ,	W	Copies the contents of PORTC to W (*)
movf	0x107 ,	W	Copies the contents of PORTC to W (*)
movf	0x187 ,	W	Copies the contents of PORTC to W (*)

(*) --- unintended result

Register	Address (hex)	Address (binary)
PORTC	0x007	0 0000 0111
TRISC	0x087	0 1000 0111
---	0x107	1 0000 0111
---	0x187	1 1000 0111

movf instruction

movf f , d → 00 1000 dfff ffff

If bank 1 is selected:

movf	PORTC ,	W	Copies the contents of TRISC to W (*)
movf	7 ,	W	Copies the contents of TRISC to W (*)
movf	.7 ,	W	Copies the contents of TRISC to W (*)
movf	TRISC ,	W	Copies the contents of TRISC to W
movf	0x87 ,	W	Copies the contents of TRISC to W
movf	0x107 ,	W	Copies the contents of TRISC to W (*)
movf	0x187 ,	W	Copies the contents of TRISC to W (*)

(*) --- unintended result

Register	Address (hex)	Address (binary)
PORTC	0x007	0 0000 0111
TRISC	0x087	0 1000 0111
---	0x107	1 0000 0111
---	0x187	1 1000 0111

Caution: Unintended Results

- An instruction may not do what you think it is doing.
- If in doubt, observe the register of interest in a Watch window and step through the code in debug mode.
- Many errors are the result of incorrect bank selection.

movlw instruction

movlw k → 11 00xx kkkk kkkk

8-bit literal

It does not matter which bank is selected. The operand is interpreted as a literal. The assembler ignores bits to the left of the 8 bits

movlw	PORTC	Copies 7 to W
movlw	7	Copies 7 to W
movlw	.7	Copies 7 to W
movlw	B'00000111'	Copies 7 to W
movlw	TRISC	Copies 0x87 to W
movlw	0x107	Copies 7 to W
movlw	0x187	Copies 0x87 to W

0x087	=	0	1000	0111,	so	W	=	1000	0111
0x107	=	1	0000	0111,	so	W	=	0000	0111
0x187	=	1	1000	0111,	so	W	=	1000	0111

Lab 3 Outline

1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

We can use the **xor** test for **bit** equality

xor is a logical operator applied to **bits** A and B

A	B	A xor B	
0	0	0	A = B
0	1	1	A ≠ B
1	0	1	A ≠ B
1	1	0	A = B

$$A = B \Leftrightarrow (A \text{ xor } B) = 0$$

$$A \neq B \Leftrightarrow (A \text{ xor } B) = 1$$

We can use the **xor** test for **byte** equality

xor is a logical operator applied to **bytes** L and W

	Case 1
W	0001 0001
L	0001 0001
W xor L	0000 0000

$$W = L \Leftrightarrow (W \text{ xor } L) = 0$$

	Case 2
W	0001 0001
L	1001 1000
W xor L	1000 1001

$$W \neq L \Leftrightarrow (W \text{ xor } L) \neq 0$$

The `xorlw` instruction

- Computes the bitwise `xor` of a literal (constant) with the contents of the W register.
- The result of `xor` is placed in the W register:

$$(L \text{ xor } W) \rightarrow W$$

- Instruction summary from Page 142 of data sheet:

Note



XORLW	Exclusive OR Literal with W
Syntax:	<code>[label] XORLW k</code>
Operands:	$0 \leq k \leq 255$
Operation:	$(W) \text{ .XOR. } k \rightarrow (W)$
Status Affected:	Z
Description:	The contents of the W register are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register.

Opcode for xorlw

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRWF	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDTC	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

- bit 7 **IRP**: Register Bank Select bit (used for indirect addressing)
 1 = Bank 2, 3 (100h - 1FFh)
 0 = Bank 0, 1 (00h - FFh)
- bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
 11 = Bank 3 (180h - 1FFh)
 10 = Bank 2 (100h - 17Fh)
 01 = Bank 1 (80h - FFh)
 00 = Bank 0 (00h - 7Fh)
 Each bank is 128 bytes
- bit 4 **$\overline{\text{TO}}$** : Time-out bit
 1 = After power-up, CLRWD_T instruction, or SLEEP instruction
 0 = A WDT time-out occurred
- bit 3 **$\overline{\text{PD}}$** : Power-down bit
 1 = After power-up or by the CLRWD_T instruction
 0 = By execution of the SLEEP instruction
- 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/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
 (for borrow, the polarity is reversed)
 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/borrow bit (ADDWF, ADDLW, SUBLW, 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

Z-bit

Example: Suppose the result is put in the W register. Then

$$\left. \begin{array}{l} Z = 1 \Leftrightarrow W = 0 \\ Z = 0 \Leftrightarrow W \neq 0 \end{array} \right\}$$

Note opposite polarity of Z and W

The `xorlw` test for byte equality

- Example: Is $W = L$?

$$\begin{array}{r} W = 0001\ 0001 \\ L = 1001\ 1000 \\ \hline \text{xor}(L, W) = 1000\ 1001 \end{array}$$

```
movlw    B'00010001'    ; Before xorlw:  W = 0001 0001
xorlw     B'10011000'    ; After  xorlw:   W = 1000 1001
```

- Result $W \neq 0$, so STATUS bit $Z = 0$

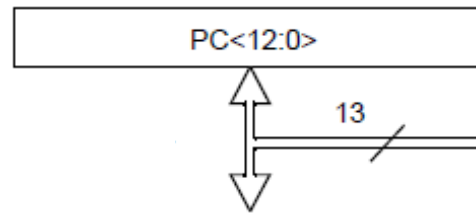
W (before xorlw)	W (after xorlw)	Z
$L = W$	$W = 0$	$Z = 1$
$L \neq W$	$W \neq 0$	$Z = 0$

- So the original $W \neq L$.

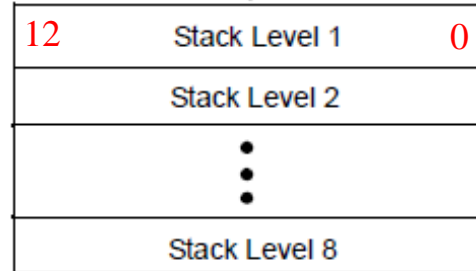
Lab 3 Outline

1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables.
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

Program Counter →

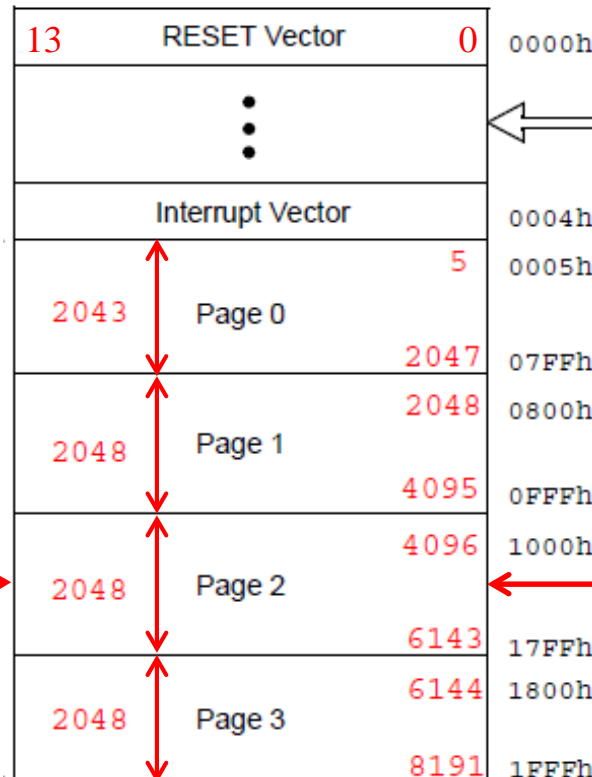


Stack Memory
(call, return)



$2^{13} = 8192 =$
 $8(1024) = 8 \text{ k}$
program memory
addresses

Program Memory
(Flash
EEPROM)

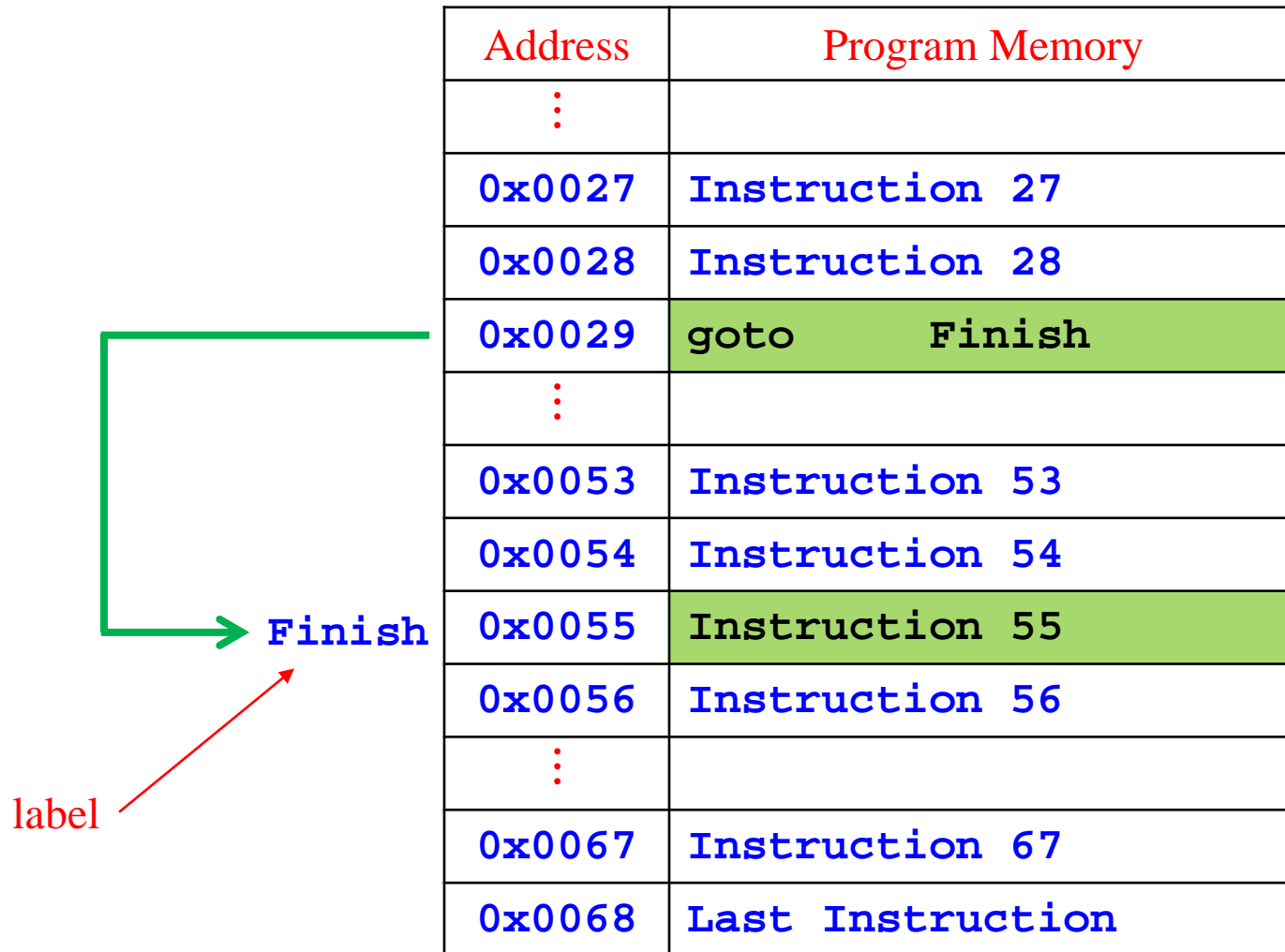


PC

14-bit registers
(instructions)

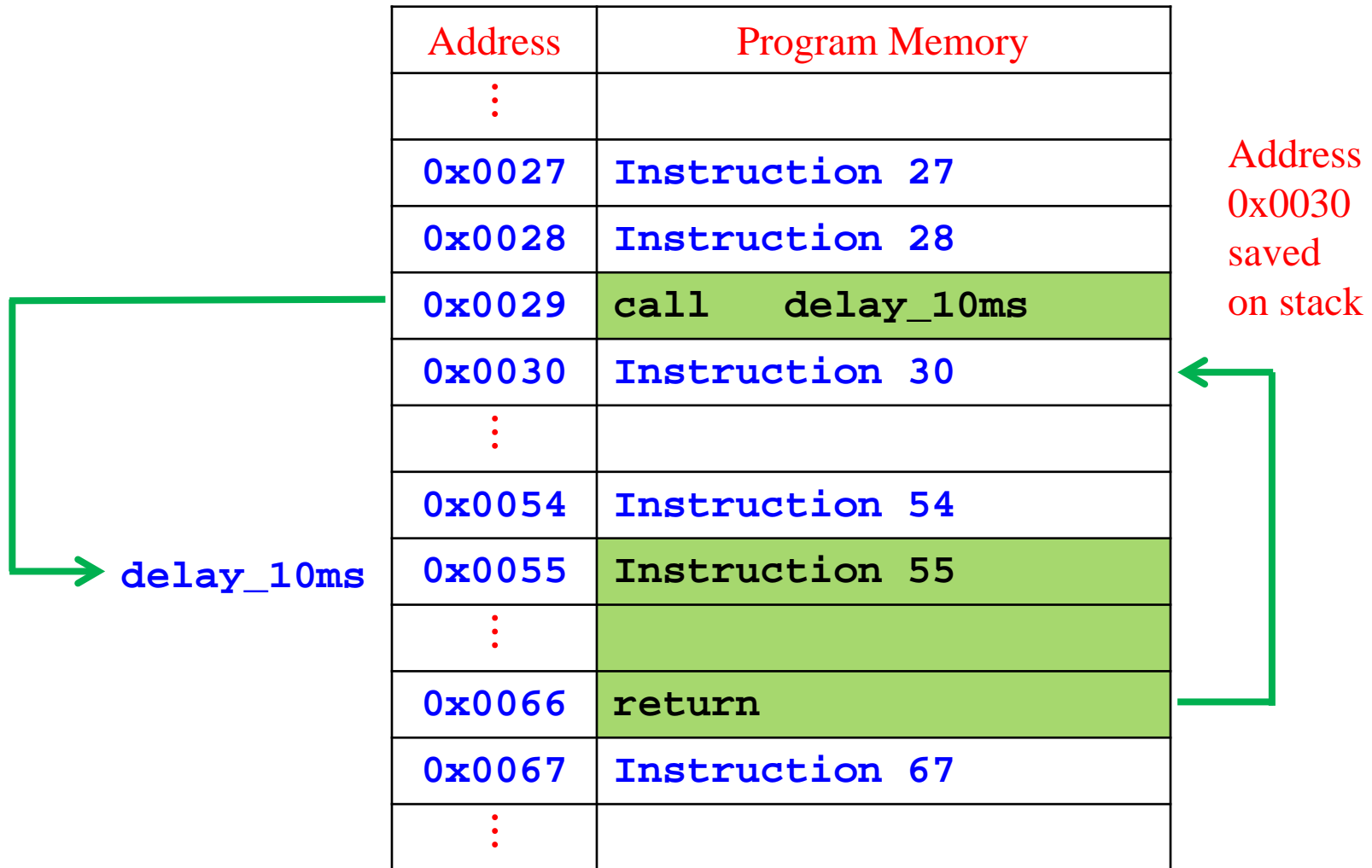
Unconditional Program Branching

The goto instruction.



Unconditional Program Branching

The `call` and `return` instructions.



Conditional Program Branching

Conditional Branching Instructions:

decfsz
incfsz
btfsc
btfss

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

bt fss conditional instruction

1. `bt fss f, b` : "Bit test f, skip if set"
means "Test bit `b` of file register `f`. If `b` is set, skip the next instruction."
2. Example: `bt fss STATUS, Z`
3. This means "Test the Z-bit of the STATUS register. If $Z = 1$, skip the instruction immediately following `bt fss` and execute the second instruction."
4. If $Z = 0$, continue with the instruction immediately following `bt fss`


Testing for byte equality and branching

```
movlw    B'00010001'    ; W = 0001 0001
xorlw    B'10011000'    ; L = 1001 1000

; Result → W,  W = 1000 1001, so Z = 0, L ≠ W

btfss    STATUS, Z      ; If Z = 1, then skip
goto     label_1         ; next instruction.
goto     label_2         ; If Z = 0, don't skip.

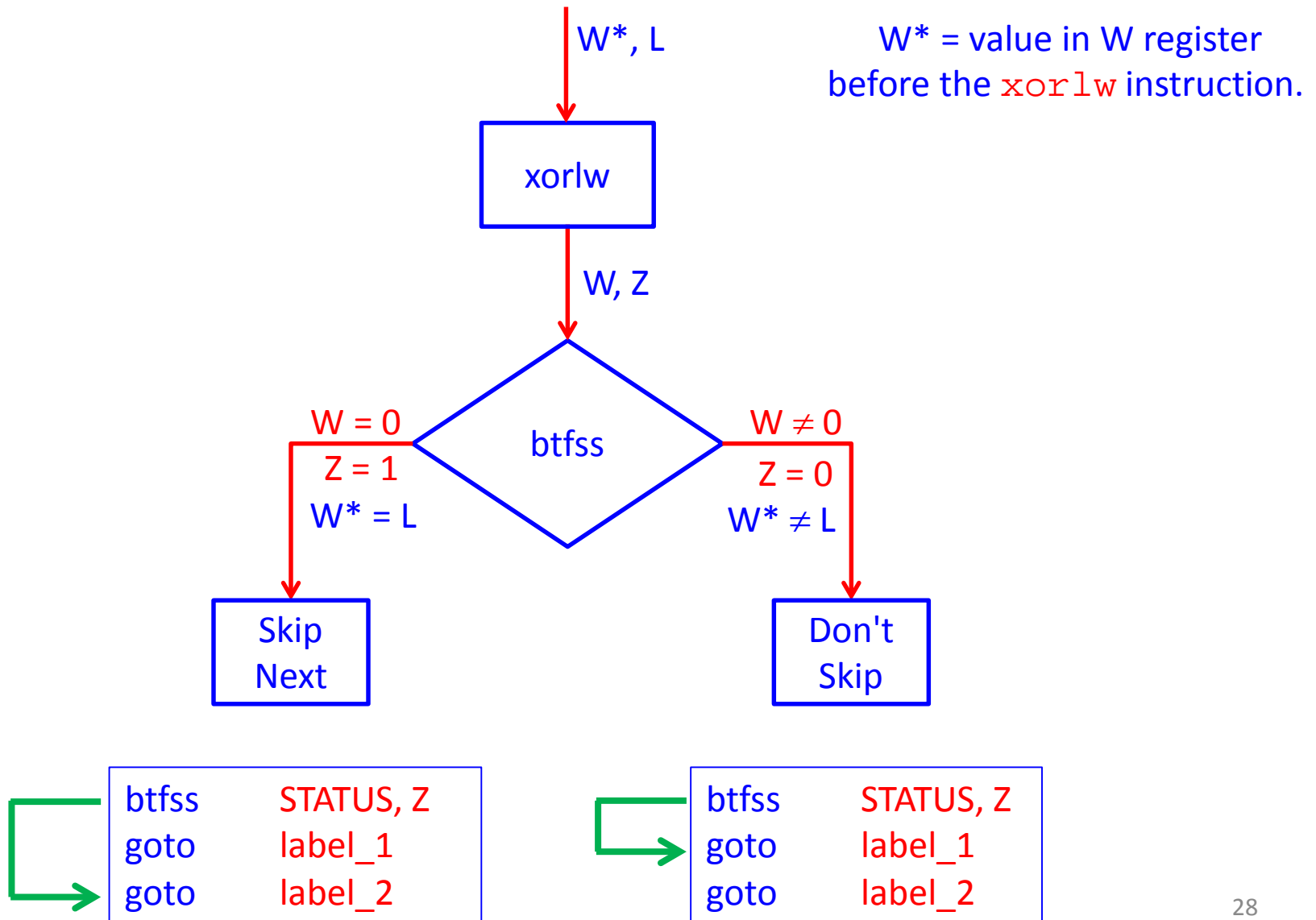
label_1   <instruction>
. . .
label_2   <instruction>
```



The above assembly code implements the standard **If-Then-Else** statement:

IF Z = 0, **THEN** goto label_1 **ELSE** goto label_2

Testing for byte equality and branching.

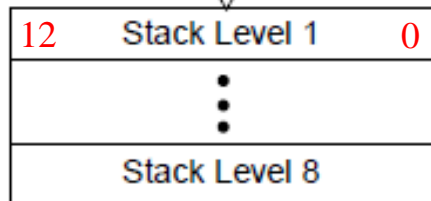
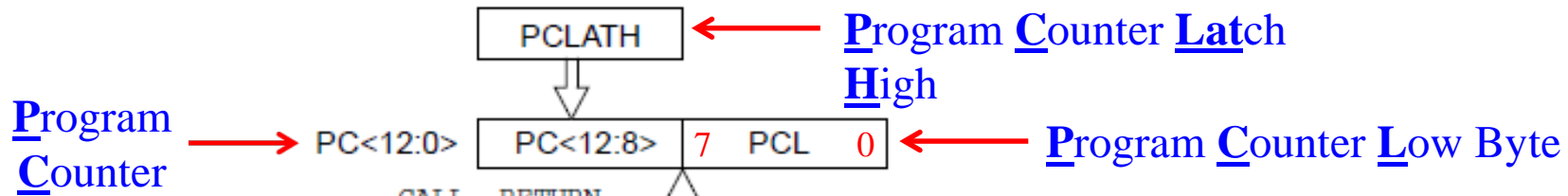


Program Control

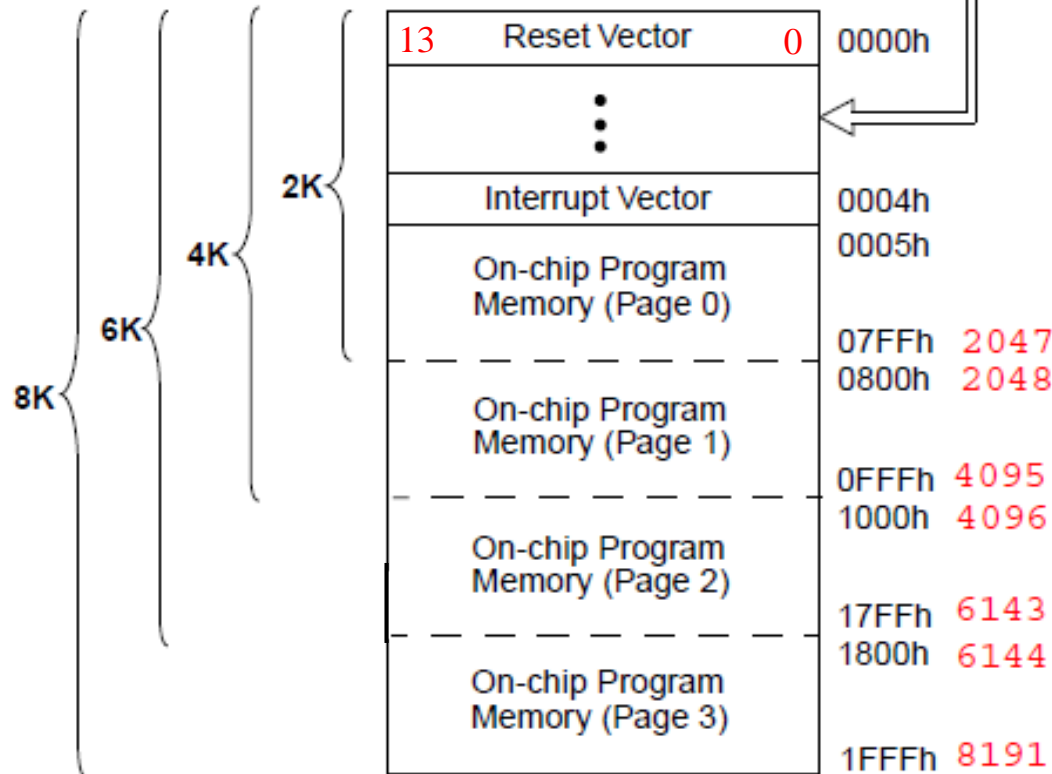
- Program control (or flow or branching) is determined by instructions such as `btfss`, `incfsz`, and `goto`.
- It is one of the more difficult aspects of assembly language programming.
- If you make sure that you understand it, your work will be easier.

Lab 3 Outline

1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup



We can change the PC register to place the program counter at different addresses.

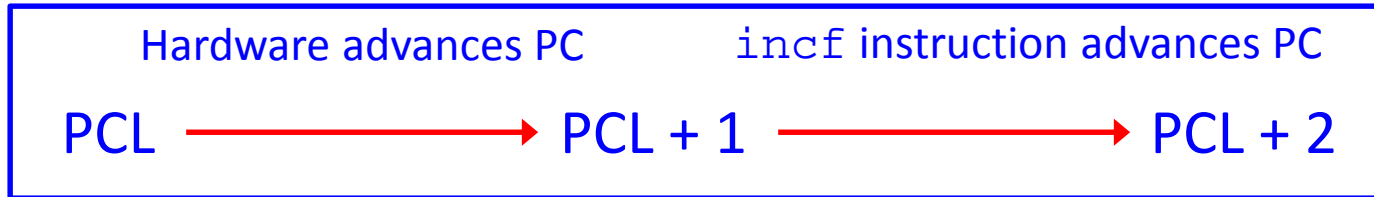


Note: Only the PC and PCLATH registers can be written to.


Program Memory

Mid-Range Ref.
Manual, p. 6-3

The `incf PCL, F` instruction



Address	Program Memory	
0x0000		
0x0001		
0x0002	<code>incf PCL, F</code>	PCL
0x0003	Instruction 1	PCL + 1
0x0004	Instruction 2	PCL + 2
0x0005		
0x0006		
0x0007		
0x0008		
0x0009		



Program Counter Arithmetic

If we want to advance the PC by 4 addresses, we set $W = 3$ and add W to the PCL register: $PCL = PCL + W + 1$.

Address	Program Memory	
0x0000		
0x0001	movlw D'3'	
0x0002	addwf PCL, F	← PCL = 2
0x0003		
0x0004		
0x0005		
0x0006		← PCL = 2 + 3 + 1
0x0007		
0x0008		
0x0009		

+1 for the addwf instruction

Application: Lookup Table

Address		Program Memory	
⋮			
0x0027	movlw	D'2'	
0x0028	call	Lookup	
0x0029	goto	0x58	
⋮			
0x0053	addwf	PCL, F	
0x0054	retlw	B'00000001'	
0x0055	retlw	B'00000010'	
0x0056	retlw	B'00000100'	
0x0057	retlw	B'00001000'	
0x0058	Instruction 58		
⋮			

PCL = PCL + W + 1

(0x0029 put on stack)

← PCL = 53

← PCL = 53 + 2 + 1

Lookup

W = 0

W = 1

W = 2

W = 3

retlw : return from subroutine with literal in W

Lookup Table Code

lab03_LookupTable.asm

```
LookupTable

    addwf    PCL, F        ; PCL = PCL + W + 1

    retlw    B'00000001'   ; W = 0
    retlw    B'00000010'   ; W = 1
    retlw    B'00000100'   ; W = 2
    retlw    B'00001000'   ; W = 3

    ; end LookupTable
```

retlw : return from subroutine with literal in W

lab03_LookupTable.asm

```
list      p = 16f877

include   "p16f877.inc"

config_1   EQU    _CP_OFF & _CPD_OFF & _LVP_OFF & _WDT_OFF
config_2   EQU    _BODEN_OFF & _PWRTE_OFF & _XT_OSC

__CONFIG   config_1 & config_2

TableSize  EQU    D'4'      ; The EQU directive assigns a literal
                             ; (constant) value to the label
                             ; 'TableSize'. The label 'TableSize'
                             ; cannot be redefined with an EQU
                             ; directive later in code.

;-----

; Allocate some General Purpose Registers for user variables. The
; values assigned to these variables can be changed in the code.

cblock    0x20

    TableIndex    ; 0x20 in data memory
    Count         ; 0x21
    CountOuter    ; 0x22
    CountInner    ; 0x23

endc
```

lab03_LookupTable.asm

```

; Continuously cycle through the first four LEDs on PORTC with a
; one-second delay.

MAIN

    call    DELAY_500ms    ; Delay 1 second
    call    DELAY_500ms

    movf    TableIndex, W    ; W = TableIndex

    call    LookupTable    ; W now contains the index for the table.
                           ; Get the PORTC entry from the table and
                           ; and return it in W.

    movwf   PORTC           ; PORTC = W

    incf    TableIndex, F    ; TableIndex = TableIndex + 1

    ; Test whether TableIndex = TableSize

    movf    TableIndex, W    ; W = TableIndex

    xorlw   TableSize        ; TableIndex xor TableSize
                           ; The result of the xorlw is placed in W.
                           ; If W = 0, TableIndex = TableSize and
                           ; Z = 1. Otherwise, W != 0,
                           ; TableIndex != TableSize, and Z = 0.

    btfss   STATUS, Z        ; Test: if Z = 1 (TableIndex = TableSize)
                           ; then skip the next instruction.

    goto    MAIN            ; If we reach this instruction, then Z = 0
                           ; (TableIndex < TableSize), so get the
                           ; next table entry for PORTC.

    clrf    TableIndex       ; If we reach this instruction, then Z = 1,
                           ; so reset TableIndex = 0

    goto    MAIN            ; Repeat until HALT or Power Down.

```

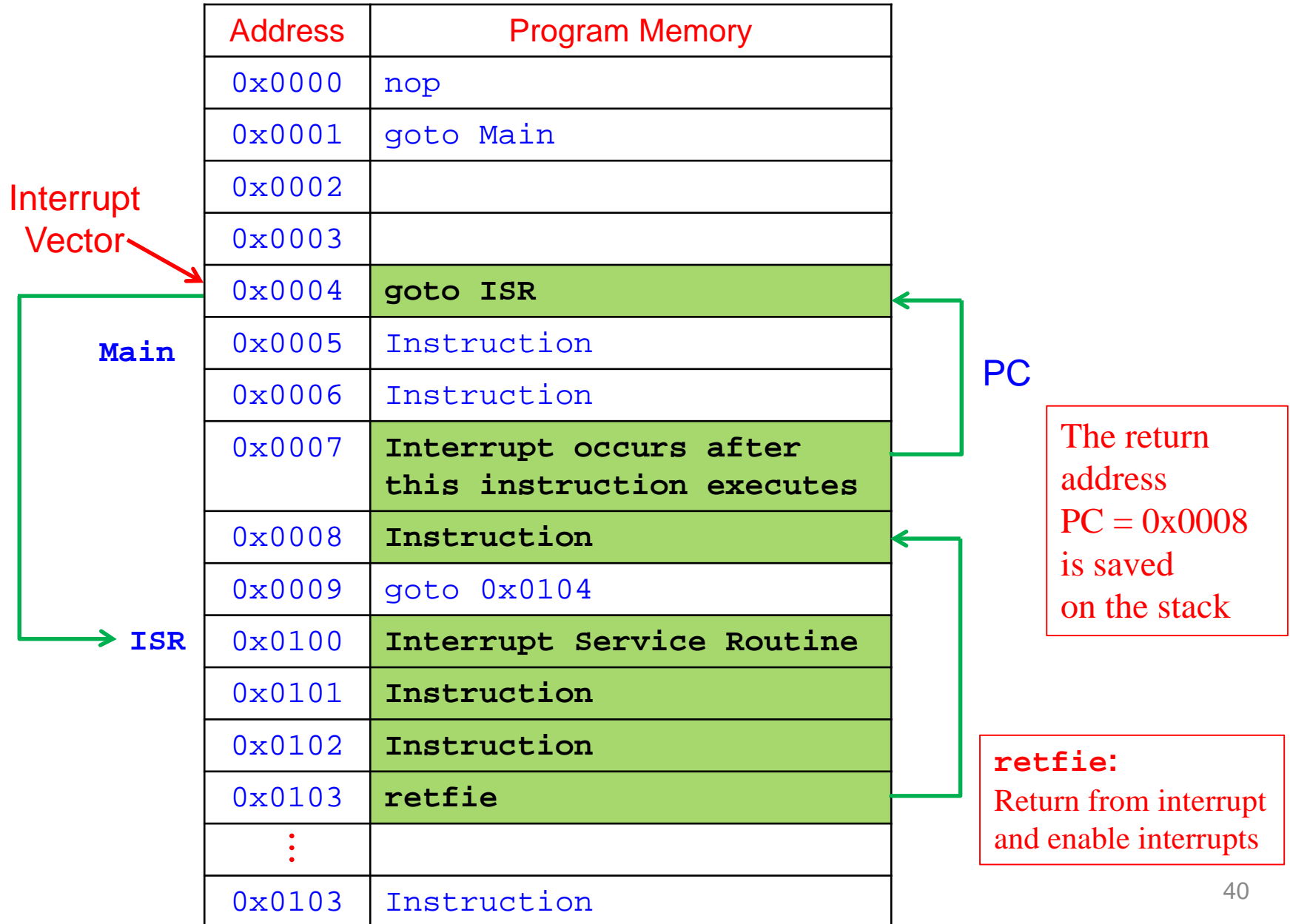
Lab 3 Outline

1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

Interrupts

1. An **interrupt** is a signal sent to the CPU controller indicating an event has occurred in one of its peripherals or in an external device that requires immediate attention.
2. When the CPU receives an interrupt signal:
 - a) Further interrupts are automatically disabled;
 - b) The address of the next instruction is saved on the **stack**;
 - c) The program counter (PC) jumps to the **interrupt vector** at address 0x0004 in program memory.
3. The CPU continues to execute instructions beginning at 0x0004 until it encounters a **retfie** (return from interrupt and enable interrupts) instruction, and the PC jumps to the address that was saved on the stack.
4. Execution continues where it left off before the interrupt.

Interrupt Program Branching



The PIC Interrupts

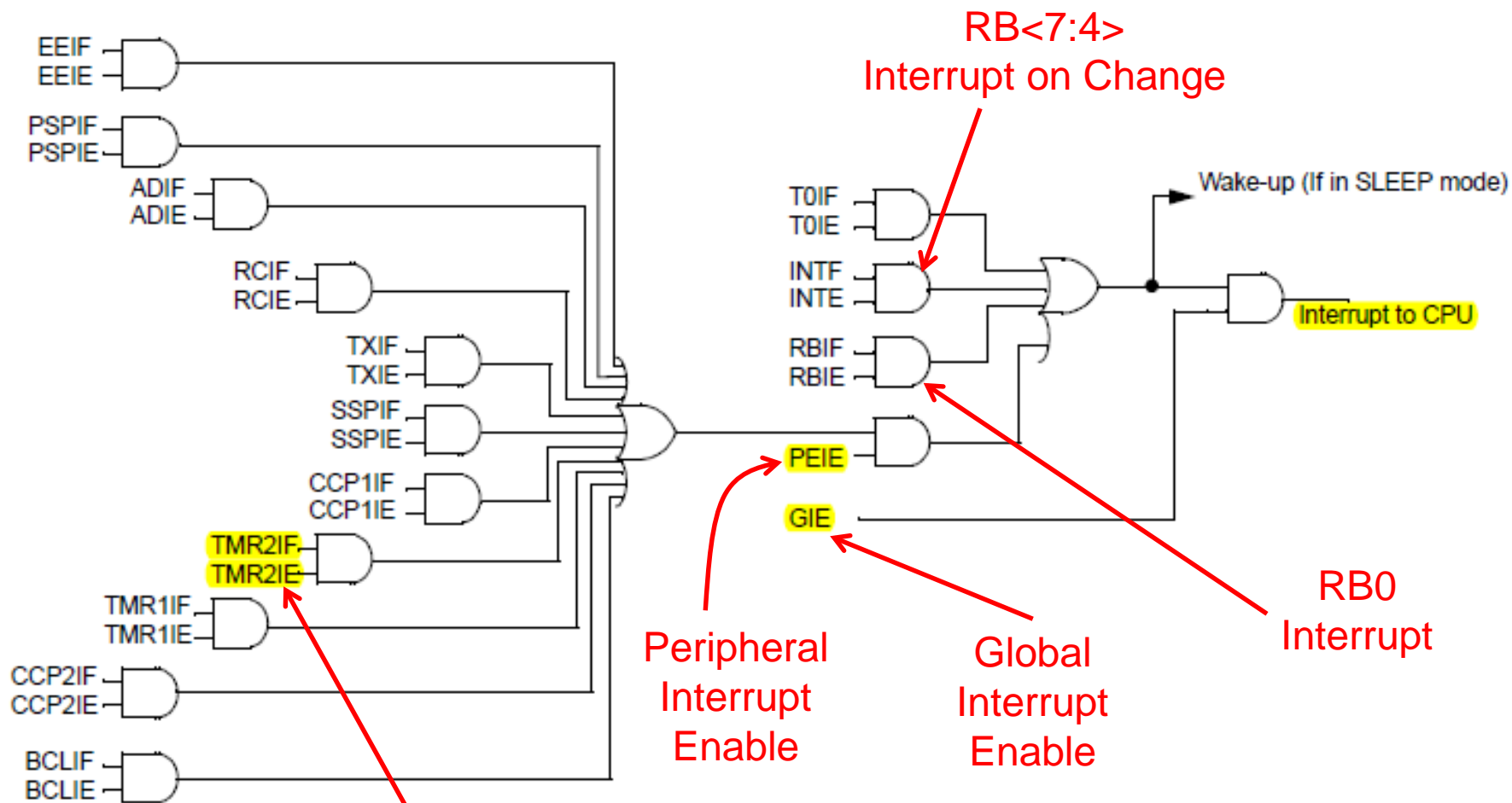
1. The PIC 16F877 has 14 interrupt sources.
2. All interrupts are disabled on reset by default.
3. Each source can be enabled by setting an enable bit denoted by xxxxIE, where the xxxx is a symbol representing the source, for example, TMR2IE – Timer2 Interrupt Enable.
4. Each source has a flag bit denoted by xxxxIF which is automatically set by hardware when an interrupt event occurs.
5. The flag bit is set even if interrupts are disabled.

The PIC Interrupt Sources

	IF Symbol	Interrupt Source
1.	EEIF	EEPROM Write Complete Interrupt
2.	PSPIF	Parallel Slave Port Read/Write Interrupt
3.	ADIF	A/D Conversion Complete Interrupt
4.	RCIF	USART Receive Interrupt
5.	TXIF	USART Transmit Interrupt
6.	SSPIF	Synchronous Serial Port (SSP) Interrupt
7.	CCP1IF	Compare Capture PWM 1 Interrupt
8.	TMR2IF	TMR2 to PR2 Match Interrupt
9.	TMR1IF	TMR1 Overflow Interrupt
10.	CCP2IF	Compare Capture PWM 2 Interrupt
11.	BCLIF	Bus Collision Interrupt (I2C Master Mode)
12.	T0IF	TMR0 Overflow Interrupt
13.	INTF	RB0/INT External Interrupt
14.	RBIF	PORTB<RB7:RB4> Change Interrupt

Interrupt Control Bits

FIGURE 12-9: INTERRUPT LOGIC



Timer2 Interrupt Flag
Timer2 Interrupt Enable

INTCON - Interrupt Control Register

The **Global Interrupt Enable (GIE)** bit and the **Peripheral Interrupt Enable (PEIE)** bit must be set in order to use any of the peripherals to the left of the PEIE bit in the previous diagram.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

bit 7

bit 0

bit 7

GIE: Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6

PEIE: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

Data sheet, p. 20

Data sheet, p. 20

Lab 3 Outline

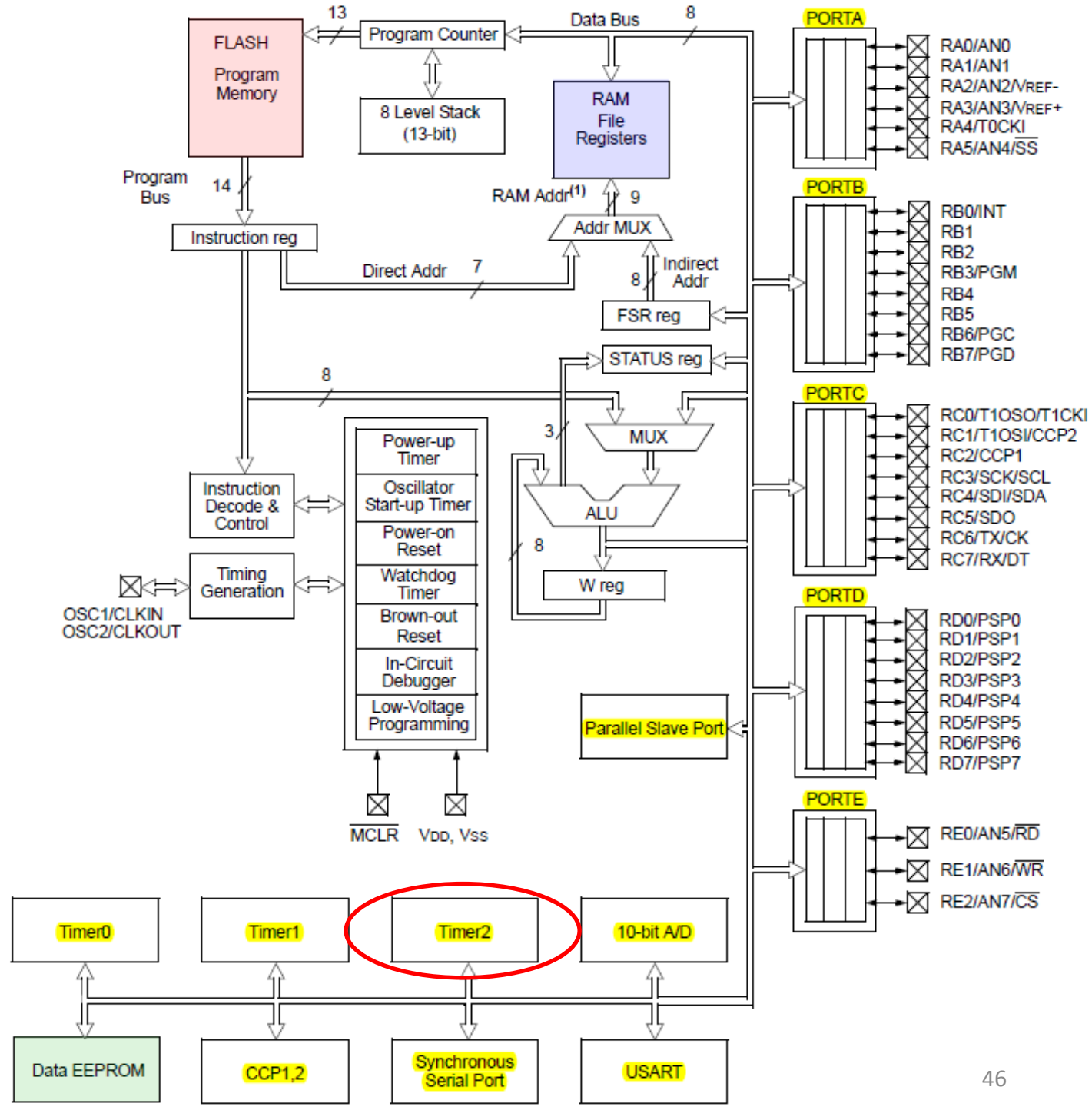
1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

PIC 16F877 Architecture

$2^{13} = 8192$
program memory
addresses
(Flash EEPROM)

$2^9 = 512$
data memory
addresses
(SRAM)

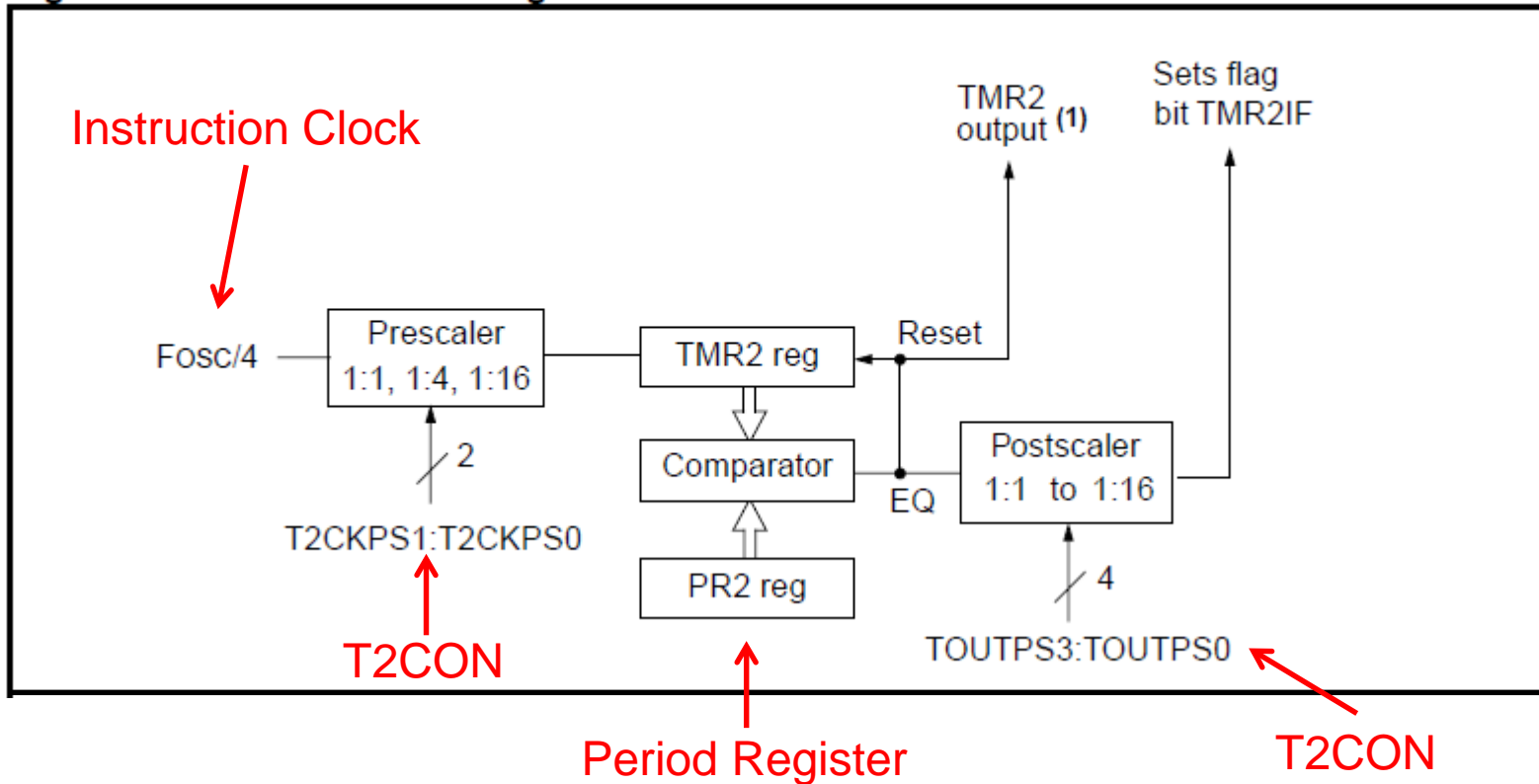
$2^8 = 256$
data memory
addresses
(EEPROM)



Timer2

Figure 13-1: Timer2 Block Diagram

Mid-Range Reference Manual, p. 13-2



1. The TMR2 register increments from 0x00 until it matches PR2 and then resets to 0x00 on the next increment from the Prescaler.
2. The Postscaler increments on each match of TMR2 and PR2.
3. $\text{Timer2 Period} = (\text{Prescaler}) (\text{PR2} + 1) (\text{Postscaler}) (4 T_{\text{osc}})$

Timer2 control registers

1. TMR2
2. T2CON
3. PR2 (Period Register)

	File Address		File Address		File Address		File Address
Indirect addr. ⁽¹⁾	00h	Indirect addr. ⁽¹⁾	128	Indirect addr. ⁽¹⁾	256	Indirect addr. ⁽¹⁾	384
TMR0	01h	OPTION_REG	80h	TMR0	100h	OPTION_REG	180h
PCL	02h	PCL	81h	PCL	101h	PCL	181h
STATUS	03h	STATUS	82h	STATUS	102h	STATUS	182h
FSR	04h	FSR	83h	FSR	103h	FSR	183h
PORTA	05h	TRISA	84h		104h		184h
PORTB	06h	TRISB	85h	PORTB	105h	TRISB	185h
PORTC	07h	TRISC	86h		106h		186h
PORTD ⁽¹⁾	08h	TRISD ⁽¹⁾	87h		107h		187h
PORTE ⁽¹⁾	09h	TRISE ⁽¹⁾	88h		108h		188h
PCLATH	0Ah	PCLATH	89h		109h		189h
INTCON	0Bh	INTCON	8Ah	PCLATH	10Ah	PCLATH	18Ah
PIR1	0Ch	PIE1	8Bh	INTCON	10Bh	INTCON	18Bh
PIR2	0Dh	PIE2	8Ch	EEDATA	10Ch	EECON1	18Ch
TMR1L	0Eh	PCON	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1H	0Fh		8Eh	EEDATH	10Eh	Reserved ⁽²⁾	18Eh
T1CON	10h		8Fh	EEADRH	10Fh	Reserved ⁽²⁾	18Fh
TMR2	11h	SSPCON2	90h		110h		190h
T2CON	12h	PR2	91h		111h		191h
SSPBUF	13h	SSPADD	92h		112h		192h
SSPCON	14h	SSPSTAT	93h		113h		193h
CCPR1L	15h		94h		114h		194h
CCPR1H	16h		95h		115h		195h
CCP1CON	17h		96h		116h		196h
RCSTA	18h	TXSTA	97h	General Purpose Register	117h	General Purpose Register	197h
TXREG	19h	SPBRG	98h	16 Bytes	118h	16 Bytes	198h
RCREG	1Ah		99h		119h		199h
CCPR2L	1Bh		9Ah		11Ah		19Ah
CCPR2H	1Ch		9Bh		11Bh		19Bh
CCP2CON	1Dh		9Ch		11Ch		19Ch
ADRESH	1Eh	ADRESL	9Dh		11Dh		19Dh
ADCON0	1Fh	ADCON1	9Eh		11Eh		19Eh
			9Fh		11Fh		19Fh
	20h		A0h		120h		1A0h
General Purpose Register	32	General Purpose Register	160	General Purpose Register	288	General Purpose Register	416
96 Bytes		80 Bytes		80 Bytes		80 Bytes	
	7Fh	accesses 70h-7Fh	EFh	accesses 70h-7Fh	16Fh	accesses 70h-7Fh	1EFh
Bank 0	127	Bank 1	255	Bank 2	383	Bank 3	511

Timer2 control registers

REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

bit 6-3 **TOUTPS3:TOUTPS0:** Timer2 Output Postscale Select bits

0000 = 1:1 Postscale

0001 = 1:2 Postscale

0010 = 1:3 Postscale

•

•

•

1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit

1 = Timer2 is on

0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1

01 = Prescaler is 4

1x = Prescaler is 16

Lab 3 settings:

T2CON = 0000 1101

Instructions:

```
movlw      B'00001101'
```

```
movwf      T2CON
```

Timer2 Timing for Lab 3

1. Prescaler = 4, PR2 = 229, Postscaler = 2

2. Timer2 interrupts every

(Prescaler) (PR2 + 1) (Postscaler) instruction cycles

$$= 4 \times 230 \times 2$$

$$= 1840$$

instruction cycles

$$\text{Timer2 Period} = (\text{Prescaler}) (\text{PR2} + 1) (\text{Postscaler}) (4 T_{\text{osc}})$$

$$= 1840 \times 1.085 \mu\text{s} = 2.00 \text{ ms}$$

3. Five Timer2 interrupts = 10 ms

A Little History



- 1955: First computer to use interrupts: Univac 1103A
- 58 × 30 feet, 15 tons
- NACA, National Advisory Committee for Aeronautics
- Wind tunnel data collection

Lab 3 Outline

1. movwf, movf, movlw instructions.
2. xorlw instruction, STATUS<Z>-bit
3. Program branching
4. Program counter arithmetic and lookup tables
5. Interrupts
6. Timer2
7. Lab 3 Hardware setup

Lab 3 Hardware setup

Same as Lab 2

End of Lab 3