EEC 417/517
Embedded Systems
Cleveland State University

Lab 9

Switch Debouncing,

EEPROM Data Memory,

Flash EEPROM Program Memory,

Indirect Addressing

Dan Simon

Rick Rarick

Spring 2018

Lab 9 Outline

1. **Switch Debouncing**
2. EEPROM Data Memory
3. Flash EEPROM Program Memory
4. Indirect Addressing

Key (or Switch) Debouncing

- Computer Keyboard
- Cell phones
- TV remote
- Garage door opener
- Anything with a keypad or buttons

Radio Shack TRS-80 (Model I)

Catalog:	26-1001
Released:	August 1977
Price:	US $599.95 (with monitor)
How Many:	200,000 (1977-1981)
CPU:	Zilog Z-80A, 1.77 MHz
RAM:	4K, 16K max*
Ports:	Cassette I/O, video, Expansion connector*
Display:	12-inch monochrome monitor 64 X 16 text
Expansion:	External Expansion Interface*
Storage:	Cassette storage*
OS:	BASIC in ROM*

* Additional capabilties with Expansion Interface

Switch Debouncing

1. TRS-80 – Radio Shack, 1977

2. Managers decided to produce 3,500 units

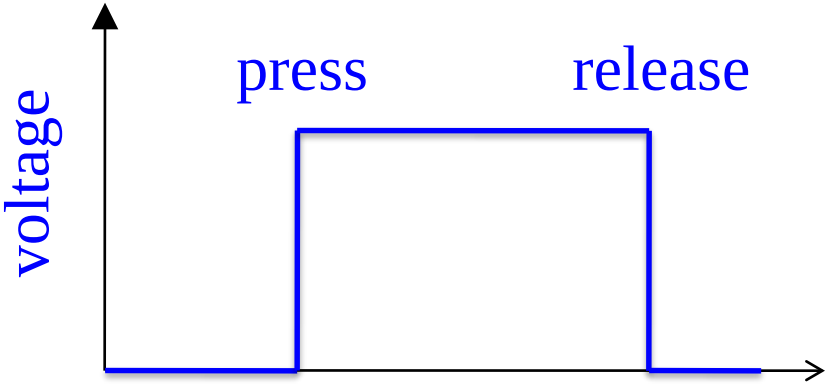
3. Radio Shack sold 65,000 units the first five months
   250,000 units in its three years of production

4. Keyboard used mechanical switches that bounced, resulting in multiple letters being typed accidentally

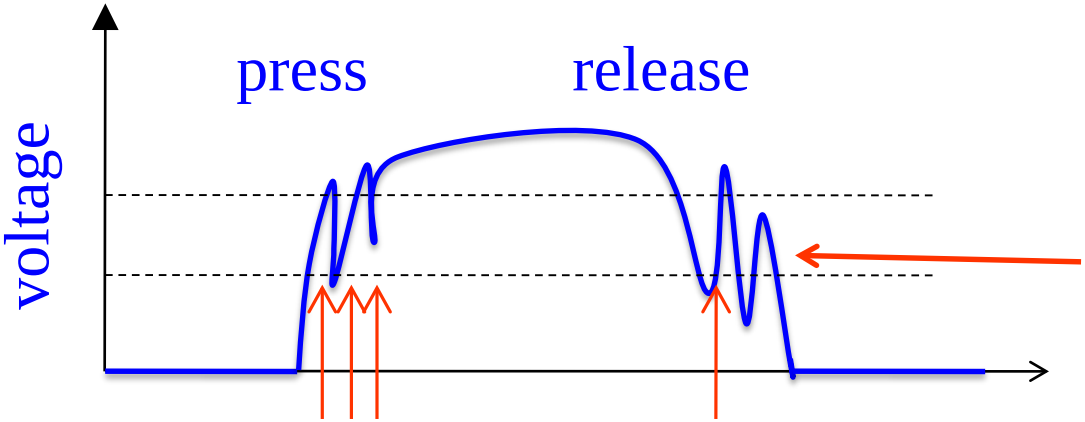
5. A keyboard debounce “tape” was distributed as a fix
   a) Ignored multiple key contact closures
   b) Slowed down keyboard polling

6. The keyboard hardware was also changed to be less vulnerable to bounce.

Switch Bouncing



Ideal World:
(No Bounce)

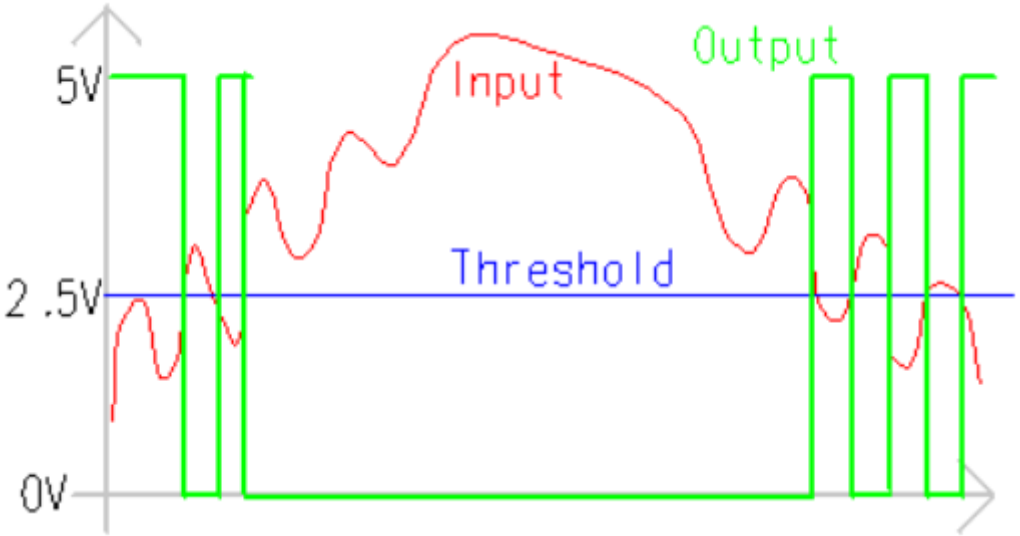


Real World:

High frequency
key bounce

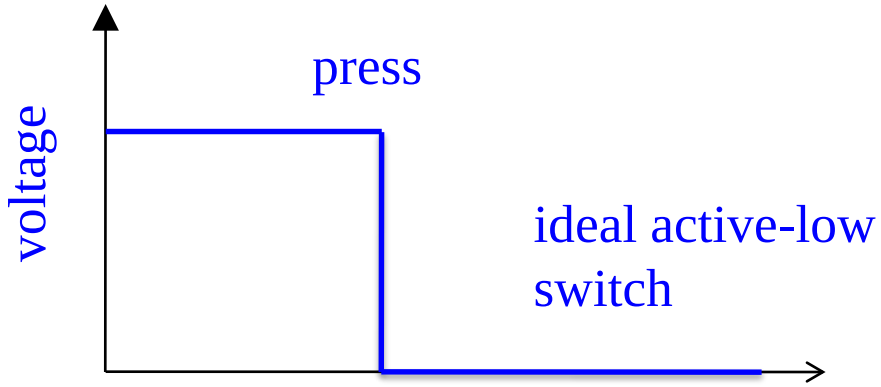
Four key presses detected!

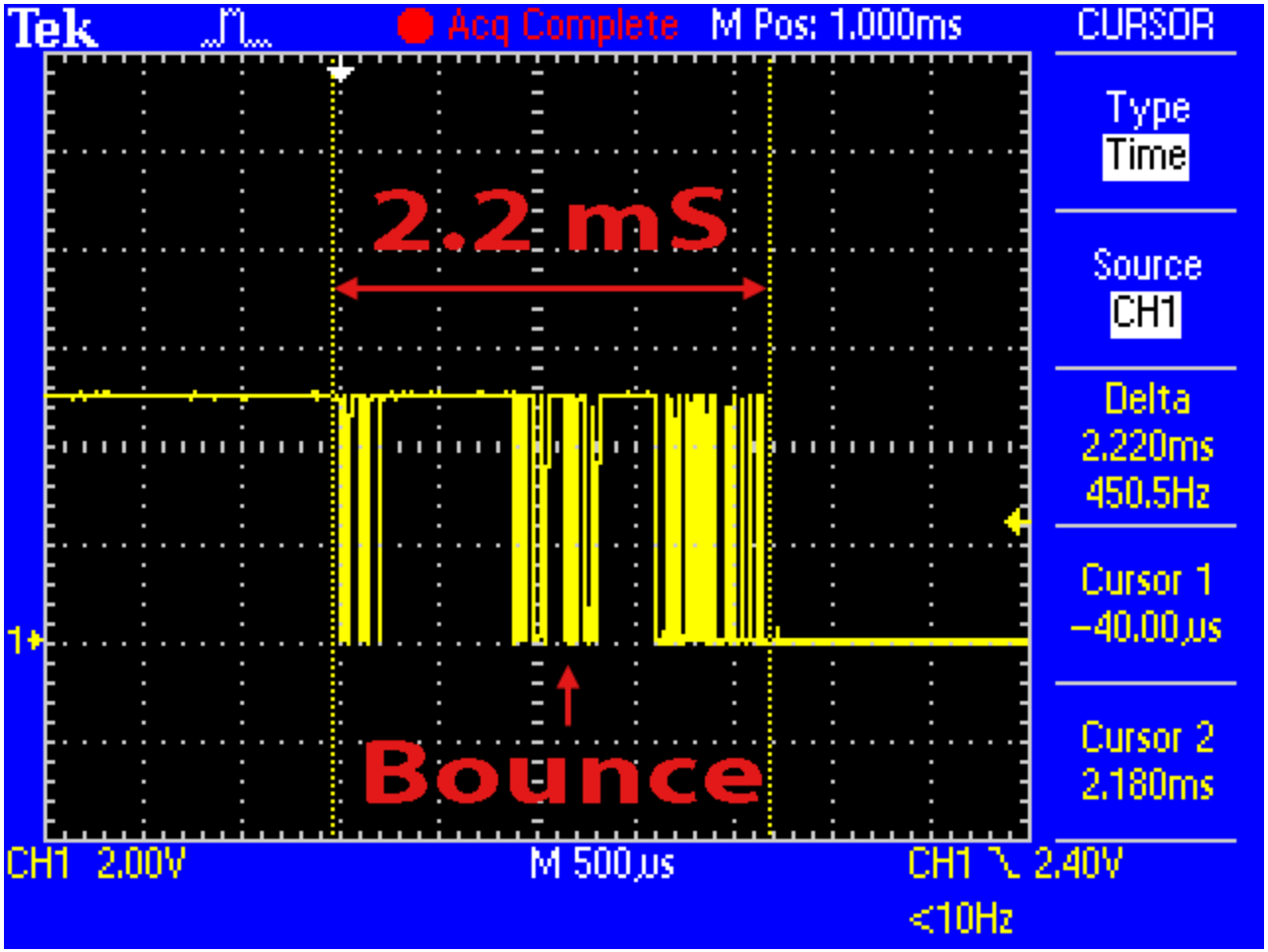
Switch Bouncing



High frequency
key bounce

Switch Bouncing





Switch Debouncing

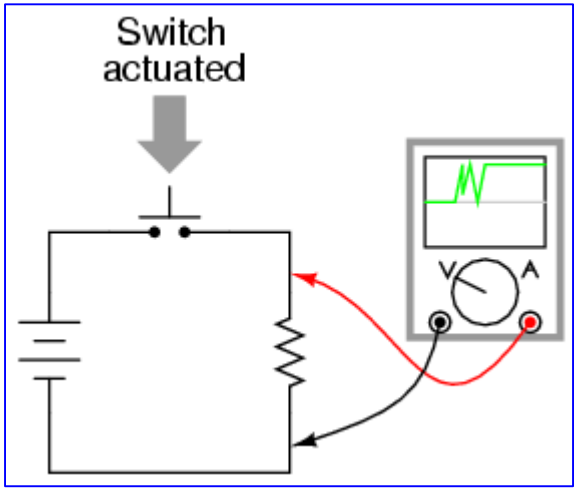
1. Software solutions
   a) Disable interrupts for *T* seconds
   b) Check for *n* consecutive stable switch readings

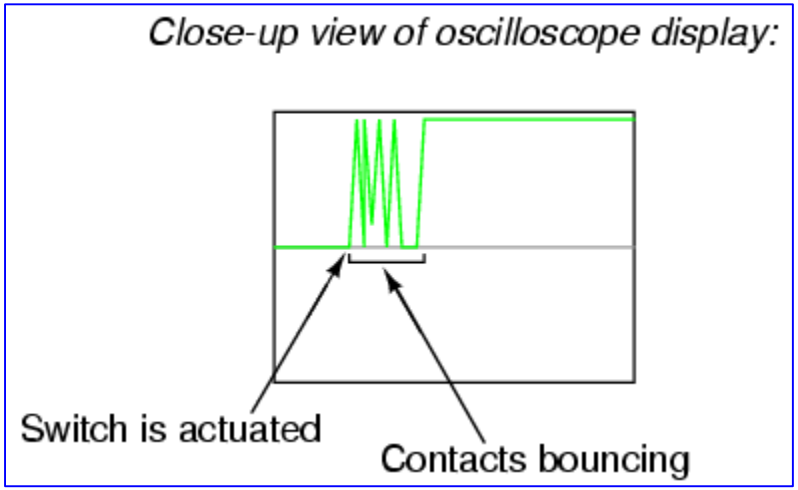
2. Analog circuit solution: low pass filter

3. Digital circuit solution: NAND gates

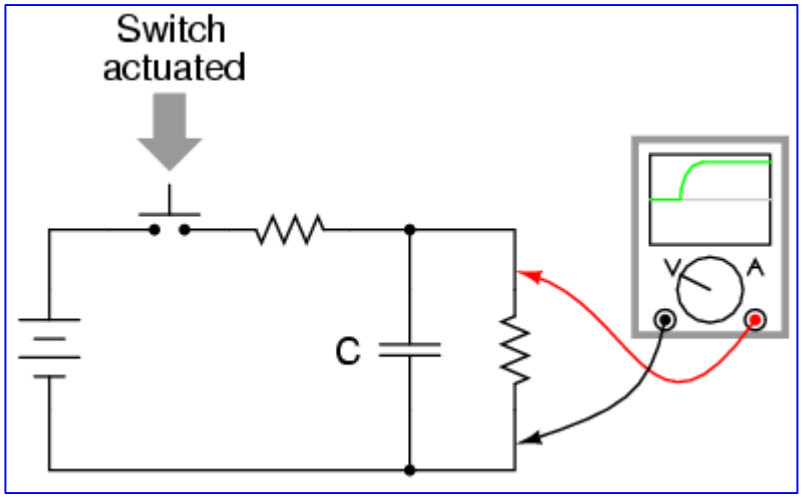
4. See www.ganssle.com/debouncing.pdf
   for more details

Low Pass Filter Debouncing





Low Pass Filter



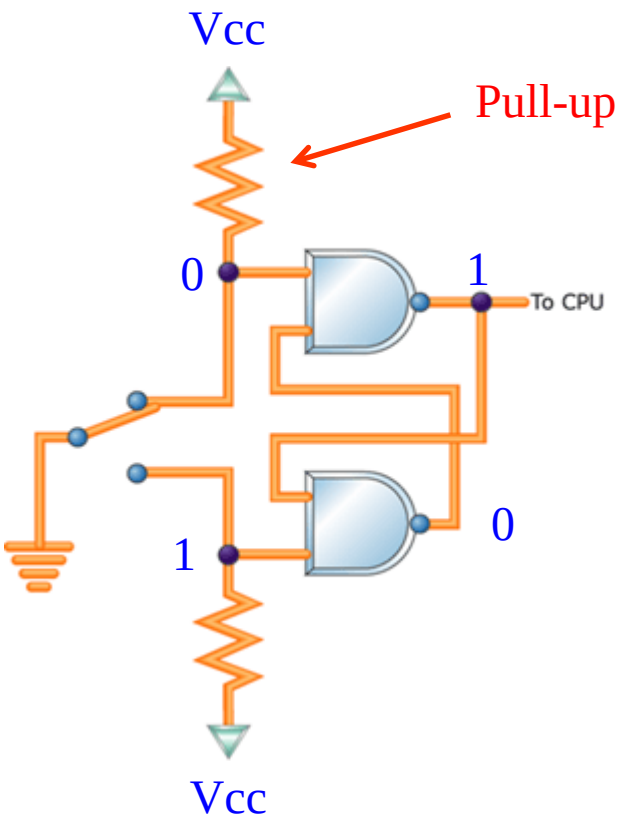
$$\mathbf{Z}(\omega) = \frac{1}{j\omega C}$$

ω large $\Rightarrow$ small

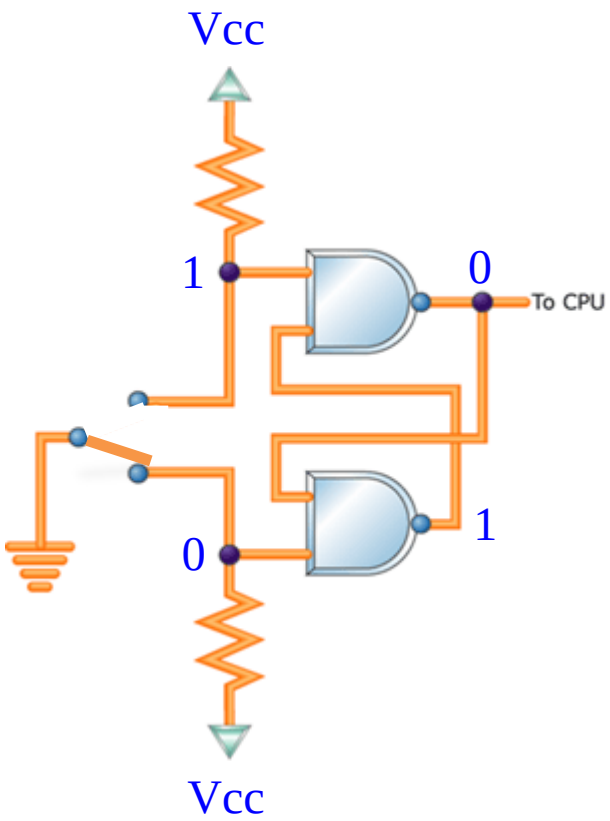
NAND Gate Debouncing
Set-Reset (SR) Latch

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

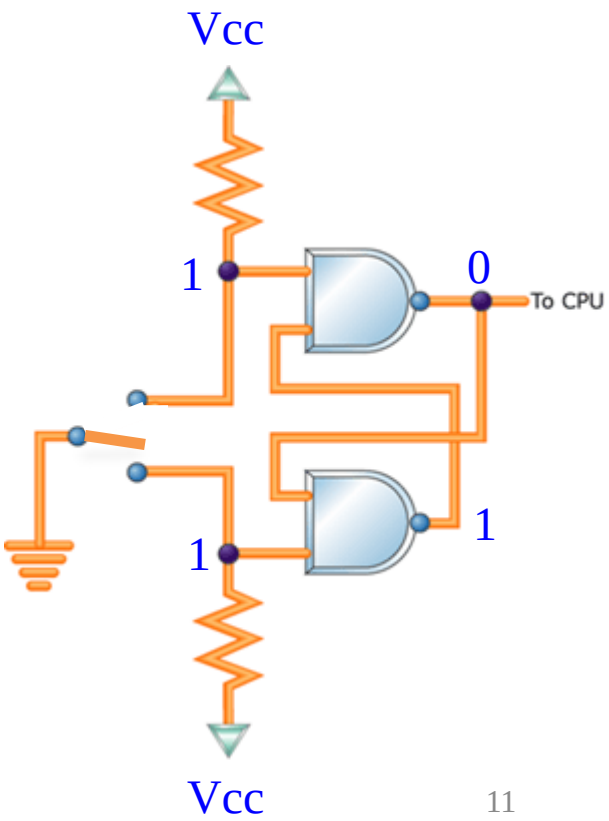
Set



Reset



Bounce



lab09.asm – Software Debounce

Debounce the RB0 button.

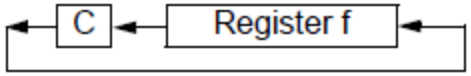
Every time the RB0 button is pressed, PORTC rotates left one bit, and the next PORTC LED should turn on:

1) 0000 0001
2) 0000 0010
…
8) 1000 0000

PORTC connected to LEDs

lab09a.asm – Software Debounce

Every time the RB0 button is pressed, PORTC rotates left one bit.

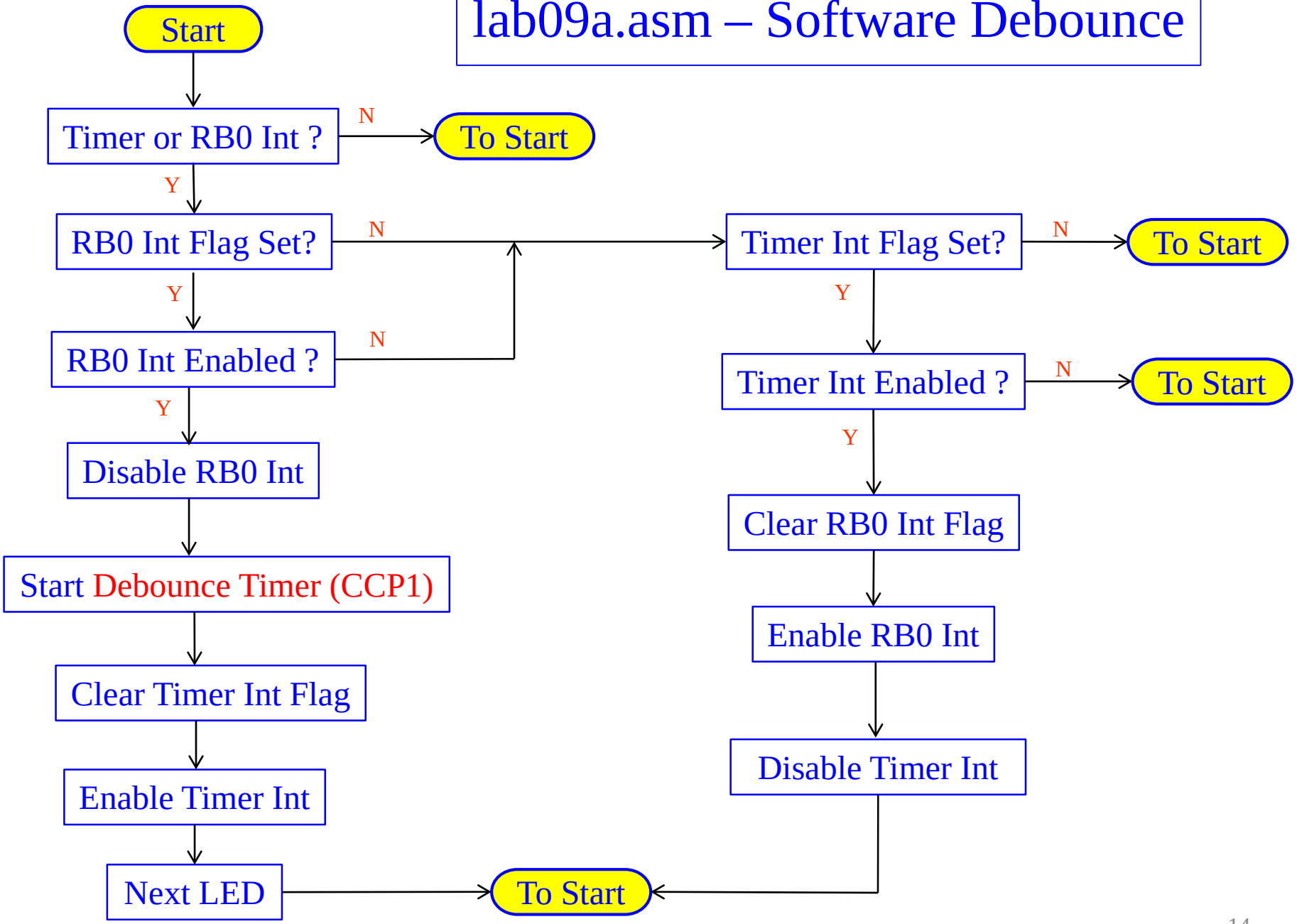
RLF	**Rotate Left f through Carry**
Syntax:	[*label*] RLF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is stored back in register 'f'. 

Rotate left:
0000 0001

Animation: http://www.pictutorials.com/RRF-RLF.htm

```
bcf     STATUS, C       ; Carry bit must be set and cleared
rlf     PORTC,  F       ; manually in code.
movf    PORTC,  F       ; Test whether PORTC = 0.
btfsc   STATUS, Z       ; If PORTC = 0, set PORTC = 1.
incf    PORTC,  F       ; Otherwise, skip and return to Main
return
```

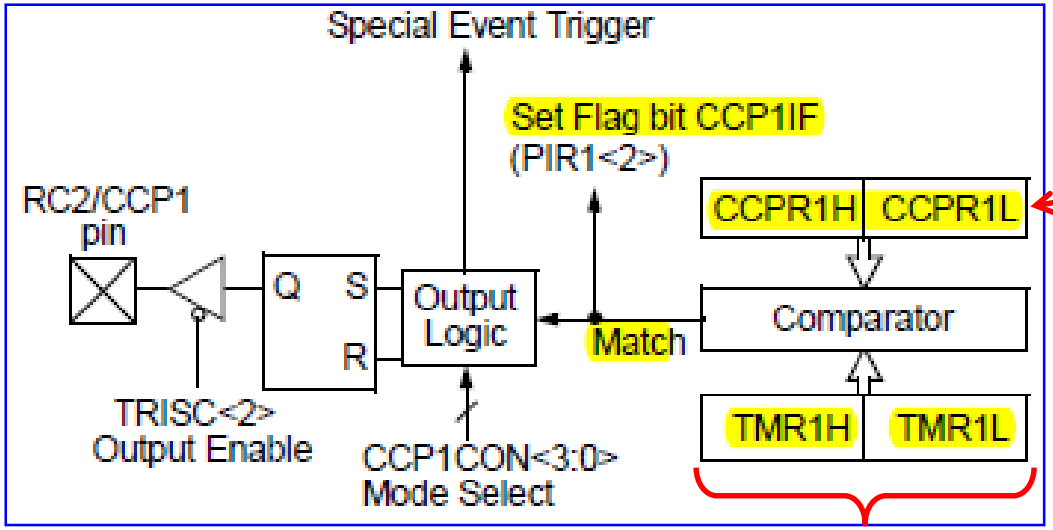
lab09a.asm – Software Debounce



CCP1 - Compare Mode (Used as 16-bit Timer)

$2^{16} = 65536$

CCP1



= 50,000

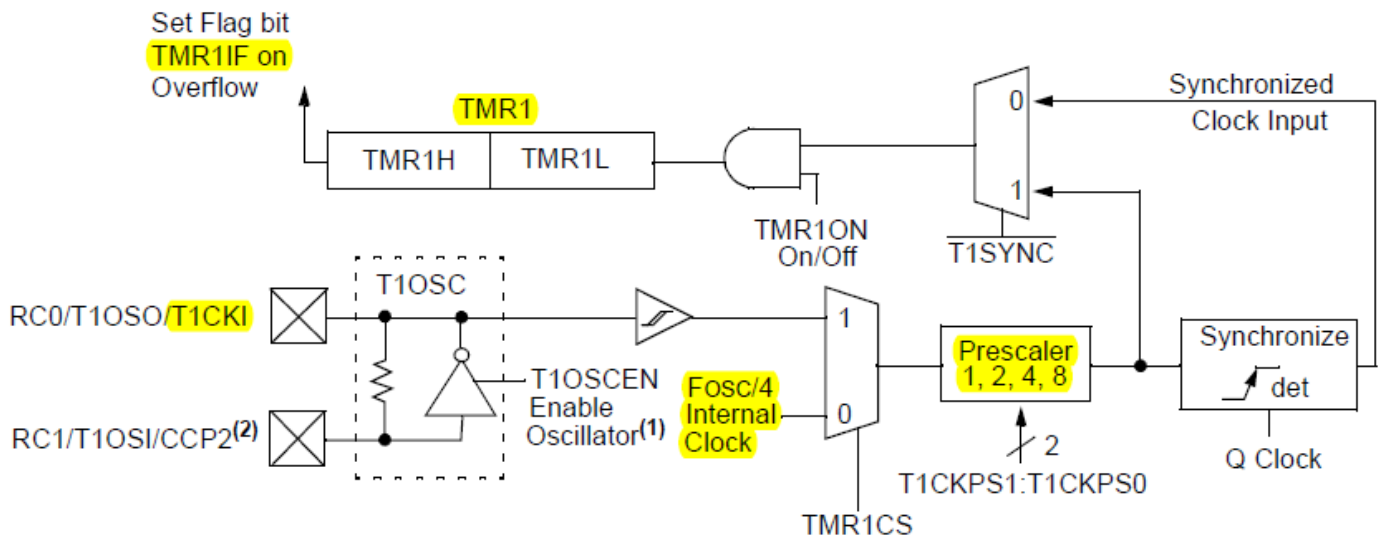
TMR1

For lab09, match when TMR1 = CCPR1H : CCPR1L = 50,000

Datasheet p. 60

Timer1

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

If Prescale = 8, then TMR1 will equal 50,000 ticks after

(8)(50,000) = 400,000 instruction cycles =

(400,000)(1.085 μs) = 0.434 sec = debounce time

Datasheet p. 52

lab09a.asm – Software Debounce

REGISTER 8-1: CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS: 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

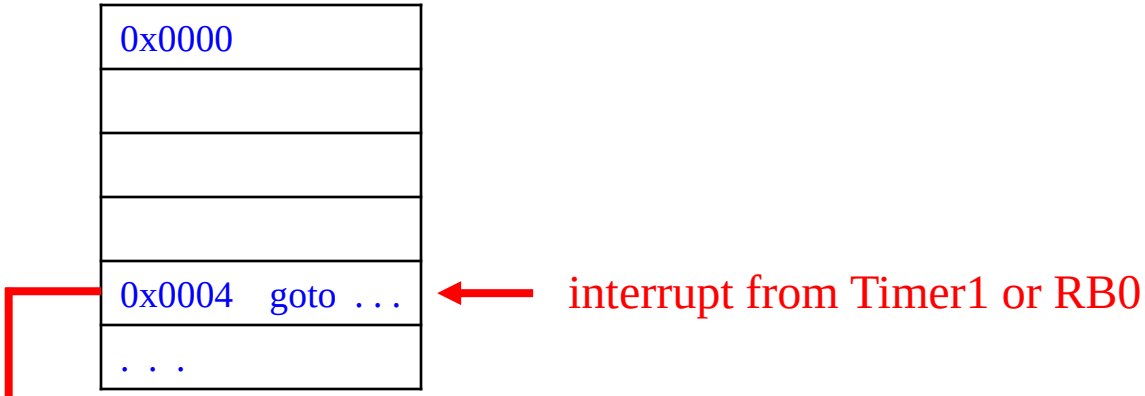
bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

lab09a.asm – Software Debounce

```
INIT
        banksel  INTCON
        bsf      INTCON, GIE
        bsf      INTCON, PEIE
        bsf      INTCON, INTE      ; Enable the RB0/INT interrupt

        clrf     PORTC             ; Initialize the LEDs to all off

        movlw    B'00110001'       ; Enable Timer1 with a 1:8 prescale
        movwf    T1CON

        movlw    0xC3              ; Set the Timer1 match to occur after
        movwf    CCPR1H            ; 0xC350 = 50000 ticks
        movlw    0x50              ; (50000 x 8 = 400,000 cycles = 0.434
        movwf    CCPR1L            ; sec @ 3.6864 MHz)

        movlw    B'00001010'       ; 1010 -> Compare mode, generate a CCP1
        movwf    CCP1CON           ; interrupt on match

        banksel  TRISB             ; Use RB0 as an interrupt
        movlw    B'00000001'
        movwf    TRISB

        clrf     TRISC             ; The PORTC pins are all outputs for LEDs
        bcf      OPTION_REG, INTEDG ; Interrupt on the falling edge of RB0
```

lab09a.asm




```
; Interrupt Service Routine

INT_SVC
        push

        btfsc   INTCON, INTF    ; Check for an RB0/INT interrupt
        call    RB0Int

        btfsc   PIR1, CCP1IF    ; Check for a CCP1 interrupt
        call    CCP1Int         ; 0.434 second timer delay

        pop
        retfie
```

RB0Int

```
; This routine disables any further interrupts from RB0, starts  ; the CCP1 debounce timer,
enables CCP1 timer interrupts, and
; turns on next LED.

banksel     INTCON
btfss       INTCON, INTE        ; Don't check for an RB0/INT interrupt,
return                          ; unless the RB0/INT interrupt is enabled

bcf         INTCON, INTF        ; Clear the RB0/INT interrupt flag
; bcf       INTCON, INTE        ; Disable the RB0/INT interrupt
; clrf      TMR1H               ; Reset the Timer1 registers
; clrf      TMR1L
; bcf       PIR1, CCP1IF        ; Clear CCP1 interrupt flag
; banksel   PIE1
; bsf       PIE1, CCP1IE        ; Enable the CCP1 interrupt

banksel     STATUS              ; Turn on the next LED
bcf         STATUS, C
rlf         PORTC, F
movf        PORTC, F ; Test whether PORTC = 0.
btfsc       STATUS, Z           ; If PORTC = 0, set PORTC = 1.
incf        PORTC, F ; Otherwise, skip and return to Main.

return
```

Debounce

Rotate

lab09a.asm

```
; This routine is entered after the debounce timer delay of about
; 434 ms. The timer interrupts are then disabled and RB0
; interrupts are enabled.

CCP1Int

        banksel  PIE1              ; Don't check for a CCP1 interrupt
        btfss    PIE1, CCP1IE      ; unless the CCP1 interrupt is enabled
        return

        banksel  INTCON
        bcf      INTCON, INTF      ; Clear the RB0/INT interrupt flag
        bsf      INTCON, INTE      ; Enable the RB0/INT interrupt

        banksel  PIE1
        bcf      PIE1, CCP1IE      ; Disable the CCP1 interrupt
        return
```

Lab 9 Outline

1. Switch Debouncing
2. **EEPROM Data Memory**
3. Flash EEPROM Program Memory
4. Indirect Addressing

PIC Memory Sizes and Types

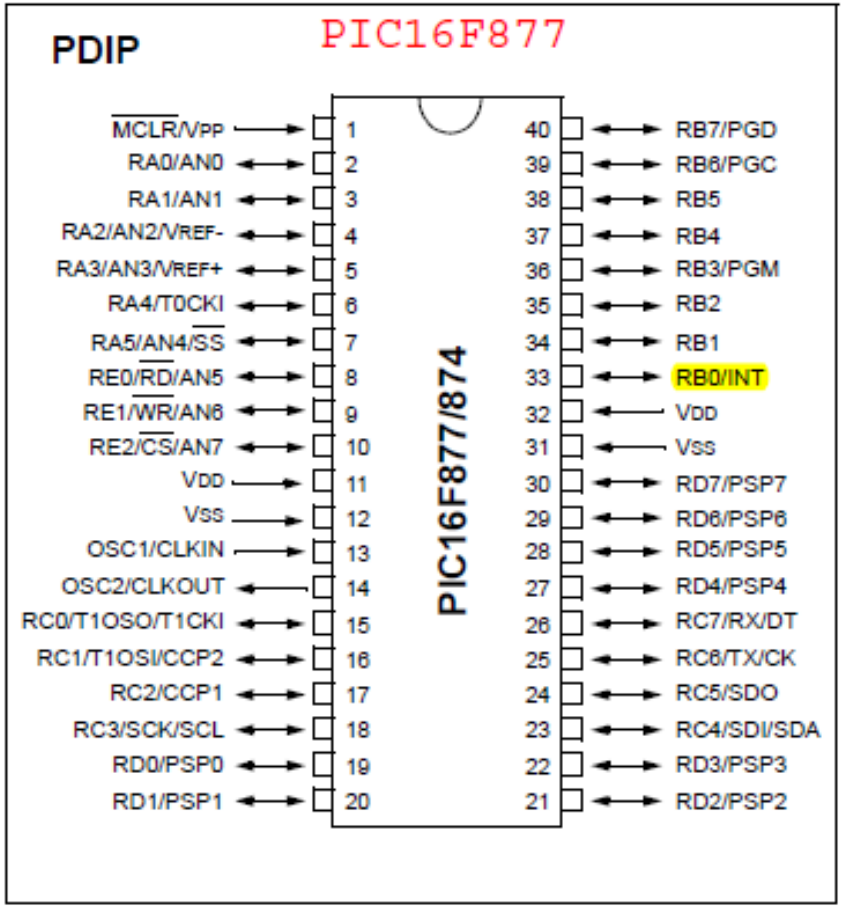
Devices Included in this Data Sheet:

- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)

Pin Diagram



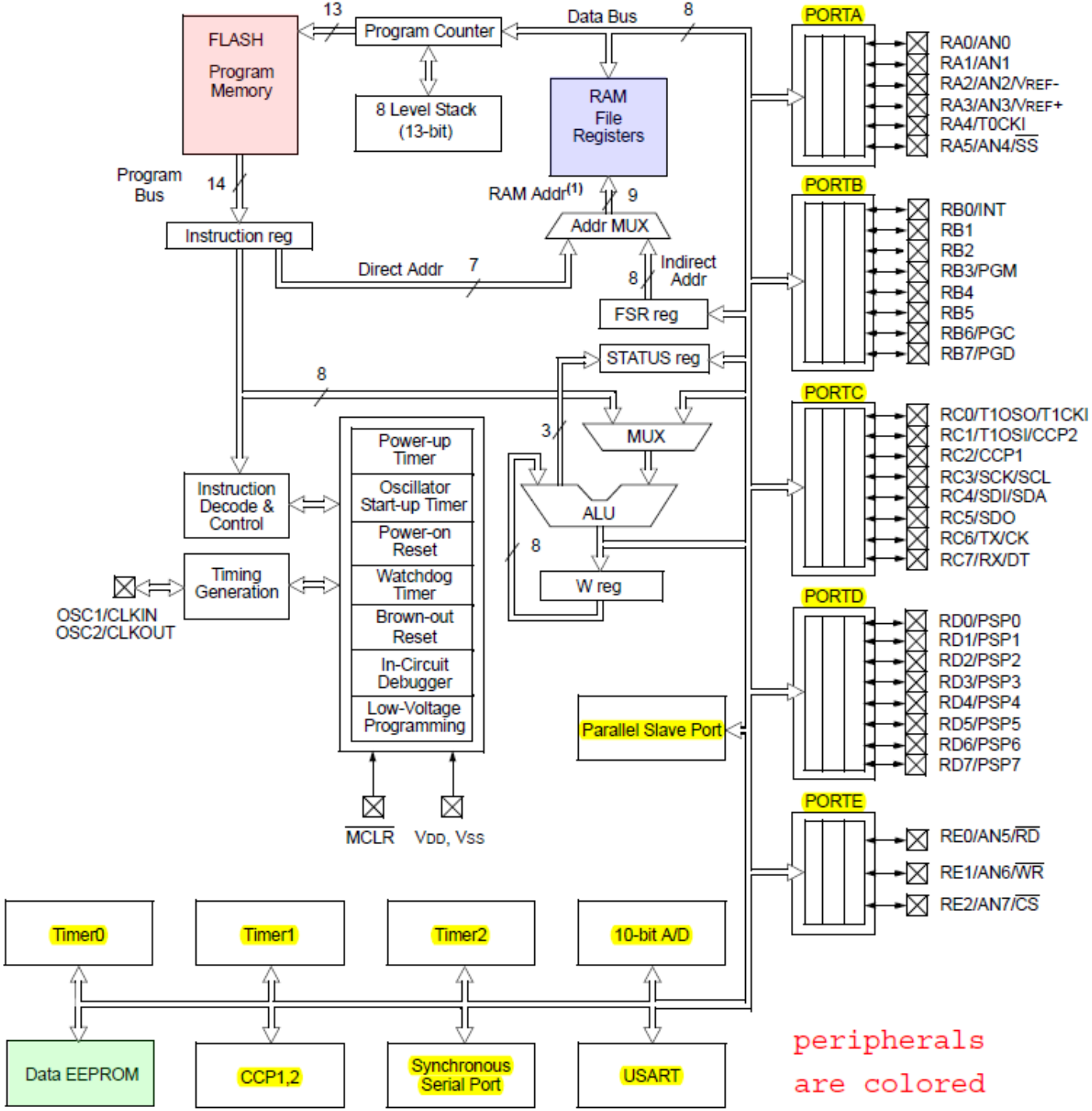
Flash Program Memory = Flash EEPROM

PIC 16F877 Architecture

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

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

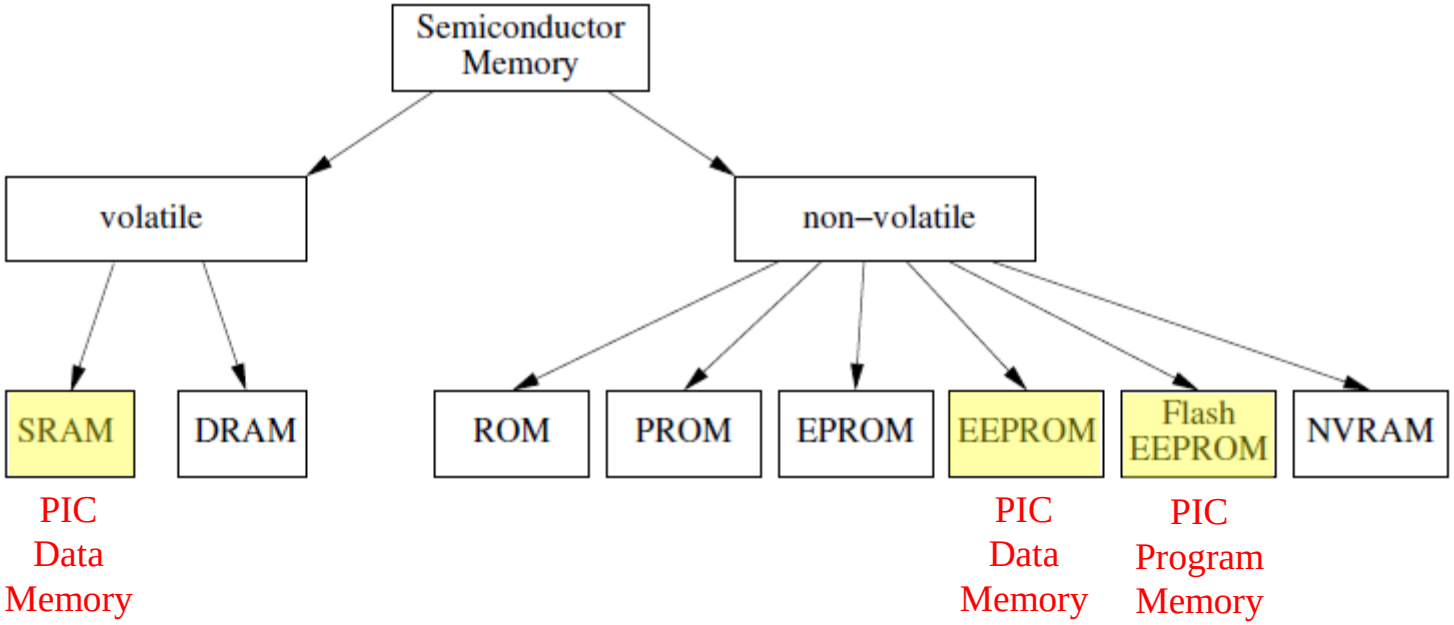
$2^{8} = 256$
data memory
addresses
(EEPROM)



peripherals
are colored

Data sheet p. 6

Basic types of semiconductor memory



1. SRAM: Static Random Access Memory
2. DRAM: Dynamic Random Access Memory
3. ROM: Read-Only Memory
4. PROM: Programmable Read-Only Memory
5. EPROM: Erasable programmable Read-Only Memory (UV)
6. EEPROM: Electrically erasable programmable ROM
7. Flash EEPROM (Individual addresses not erasable, only in blocks)

More information: Introduction to Microcontrollers (Vienna Univ).pdf, p. 23

EEPROM and Flash EEPROM

1. EEPROM: Nonvolatile, rewritable, cannot erase in blocks (slow erase).

2. Flash EEPROM: Nonvolatile, rewritable, can erase in blocks (fast erase).

3. EEPROM Data Memory Uses:
   a) Lookup tables for math functions, etc.
   b) User IDs and passwords
   c) Calibration data for instruments

4. Flash EEPROM Program Memory Uses:
   a) Extend data memory
   b) Self-modifying code

EEPROM Registers

Access to both Data EEPROM and Flash EEPROM (Program Memory) is through a set of six registers:

EEDATA
EEADR } low byte for data EEPROM

EEDATH
EEADRH } high byte for Flash EEPROM

EECON1
EECON2 } control

	File Address		File Address		File Address		File Address
			128		256		384
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	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
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h	General Purpose Register 16 Bytes	117h	General Purpose Register 16 Bytes	197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
	20h 32		A0h 160		120h 288		1A0h 416
General Purpose Register 96 Bytes		General Purpose Register 80 Bytes		General Purpose Register 80 Bytes		General Purpose Register 80 Bytes	
			EFh		16Fh		1EFh
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh 127		FFh 255		17Fh 383		1FFh 511
Bank 0		Bank 1		Bank 2		Bank 3	

Accessing EEPROM Memory

1. Data EEPROM requires 8 bits for addressing and 8 bits for data (256 bytes):

   a. **EEPROM Address** – EEADR

   b. **EEPROM Data** – EEDATA

2. Flash EEPROM (Program Memory) requires 14 bits for data and 13 bits for addressing (8192 words).

   a. **Flash EEPROM Data** – EEDATH : EEDATA

   b. **Flash EEPROM Address** – EEADRH : EEADR

REGISTER 4-1: EECON1 REGISTER (ADDRESS 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit

1 = Accesses program memory
0 = Accesses data memory
(This bit cannot be changed while a read or write operation is in progress)

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **WRERR**: EEPROM Error Flag bit

1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit

1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR**: Write Control bit

1 = Initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD**: Read Control bit

1 = Initiates an EEPROM read. (RD is cleared in hardware. The RD bit can only be set (not cleared) in software.)
0 = Does not initiate an EEPROM read

Data sheet p. 42

Example: Read data from EEPROM address 0xC3 (= 195).

Only need one byte for the EEPROM address (EEADR),
and one byte for the EEPROM data (EEDATA).

```
movlw           0xC3            ; Data EEPROM address to read from.

banksel         EEADR           ; Bank 2

movwf           EEADR           ; EEADR = 0xC3

banksel         EECON1          ; Bank 3

bcf             EECON1, EEPGD   ; Access data EEPROM memory

bsf             EECON1, RD      ; Start read operation from EEPROM
                                ; address into data buffer EEDATA.

banksel         EEDATA          ; Bank 2

movf            EEDATA, W       ; W = EEDATA for transfer to data
                                ; memory
```

Example: Write 0xBE to EEPROM address 0xC3

```
banksel  EECON1            ; Bank 3
btfsc    EECON1, WR        ; Wait for WR = 0 (previous write complete)
goto     $-1               ; PC = PC – 1 ($ = current PC, see MPASM Guide)

movlw    0xC3              ; EEPROM address to write to.
banksel  EEADR             ; Bank 2
movwf    EEADR             ; EEADR = 0xC3
movlw    0xBE              ; Data to write to EEPROM
movwf    EEDATA            ; EEDATA = 0xBE
banksel  EECON1            ; Bank 3
bcf      EECON1, EEPGD     ; Access data EEPROM memory
bsf      EECON1, WREN      ; Enable writes to EEPROM
bcf      INTCON, GIE       ; No interrupts while writing

movlw    0x55              ; These five instructions are required for
movwf    EECON2            ; every write to EEPROM to prevent inadvertent
movlw    0xAA              ; write operations. See page 43 in data sheet.
movwf    EECON2            ; EECON2 is no a physical register
bsf      EECON1, WR        ; Start write operation

bsf      INTCON, GIE       ; Enable interrupts
bcf      EECON1, WREN      ; Disable writes
```

Note: 0x55 = 0101 0101 and 0xAA = 1010 1010

Lab 9 Outline

1. Switch Debouncing
2. EEPROM Data Memory
3. **Flash EEPROM Program Memory**
4. Indirect Addressing

Accessing Program Memory

- Program memory can be used for storing data or for self-modifying code.
- Program memory: Available data memory = 368 bytes ;
  Available program memory that can be used for data = 8192 – (program size) 14-bit words
- Self-modifying code, such as a bootloader.
- A bootloader is a program that runs in the microcontroller and receives new program information (such as a firmware update) externally via some communication means and writes that information to the program memory of the processor.

Example: Read Flash EEPROM address 0x1EA0

```
movlw       0xA0          ; Low byte of Flash EEPROM address
banksel     EEADR         ; Bank 2
movwf       EEADR         ; EEADR = 0xA0
movlw       0x1E          ; High byte of Flash EEPROM address
movwf       EEADRH        ; EEADRH = 0x1E

banksel     EECON1        ; Bank 3
bsf         EECON1, EEPGD ; Access Program memory
bsf         EECON1, RD    ; Start read operation
nop                       ; Two NOPs required. See page 44,
nop                       ; in data sheet

banksel     EEDATA        ; Bank 2
movf        EEDATA, W     ;
movwf       DATAL         ; DATAL = EEDATA
movf        EEDATH, W     ;
movwf       DATAH         ; DATAH = EEDATH
```

Example: Write the instruction `goto 0x03A2` (0x2BA2) to address 0x1EA0

```
movlw    0xA0              ; Low byte of Flash EEPROM address
banksel  EEADR             ; Bank 2
movwf    EEADR             ; EEADR = 0xA0
movlw    0x1E              ; High byte of Flash EEPROM address
movwf    EEADRH            ; EEADRH = 0x1E

movlw    0xA2              ;
movwf    EEDATA            ; EEDATA = 0xA2
movlw    0x2B              ;
movwf    EEDATH            ; EEDATH = 0x2B

banksel  EECON1            ; Bank 3
bsf      EECON1, EEPGD     ; Access Program memory
bsf      EECON1, WREN      ; Enable writes to Flash EEPROM
bcf      INTCON, GIE       ; Disable interrupts

movlw    0x55              ; These seven instructions are required for
movwf    EECON2            ; every write to Flash EEPROM.
movlw    0xAA              ;
movwf    EECON2            ;
bsf      EECON1, WR        ; Start write operation
nop                        ;
nop                        ;

bsf      INTCON, GIE       ; Enable interrupts
bcf      EECON1, WREN      ; Disable writes
```

Lab 9 Outline

1. Switch Debouncing
2. EEPROM Data Memory
3. Flash EEPROM Program Memory
4. **Indirect Addressing**

Direct/Indirect Data Addressing

Data Addressing Modes: Direct and Indirect

Example: Suppose that we want to store the value `0x1C` in `Var1` at address `0x21`.

Direct Addressing: Target address of data is put in the instruction:

```
movlw       0x1C        ;     W = 0x1C
movwf       Var1        ; Var1 = 0x1C
```

The contents of W are written directly into the address location of Var1.

Indirect Addressing: This is another way to do the same thing. We put the address we want to write to (or read from) in the **File Select Register** (FSR) which is called a "pointer". (It "points to" the address we want to write to).

We then write to a pseudo-register called INDF (Indirect File) which writes to the address in FSR.

Indirect Addressing

Example: Store the value `0x1C` in `Var1` at address `0x21`.

Data Memory

INDF	00h
TMR0	01h
PCL	02h
STATUS	03h
FSR = 0x00	04h
PORTA	05h
PORTB	06h
ADCON0	1Fh
	20h
Var1 = 00h	21h
	22h
	23h
	24h
	7Fh

Bank 0

First, copy the address of `Var1` to `FSR`.

```
movlw  Var1
movwf  FSR
```

Why do these instructions copy 0x21 to Var1 instead of 0x00?

Data Memory

INDF	00h
TMR0	01h
PCL	02h
STATUS	03h
FSR = 0x21	04h
PORTA	05h
PORTB	06h
ADCON0	1Fh
	20h
Var1 = 00h	21h
	22h
	23h
	24h
	7Fh

Bank 0

Indirect Addressing

Example: Store the value `0x1C` in `Var1` at address `0x21`.

Data Memory

INDF	00h
TMR0	01h
PCL	02h
STATUS	03h
FSR = 0x21	04h
PORTA	05h
PORTB	06h
ADCON0	1Fh
	20h
Var1 = 00h	21h
	22h
	23h
	24h
	7Fh

Bank 0

Then write `0x1C` to `INDF`.

```
movlw  0x1C
movwf  INDF
```

Whatever is written to INDF will be written to the address contained in FSR. Same for reading.

Data Memory

INDF	00h
TMR0	01h
PCL	02h
STATUS	03h
FSR = 0x21	04h
PORTA	05h
PORTB	06h
ADCON0	1Fh
	20h
Var1 = 0x1C	21h
	22h
	23h
	24h
	7Fh

Bank 0

Indirect Addressing

Indirect addressing is also used for a common programming technique called "**pointer arithmetic**."

Here is an good animation illustrating pointer arithmetic using indirect addressing:

http://www.pictutorials.com/Indirect_Addressing_Exam.htm

Indirect Addressing

Example: Suppose that we want to store the value `0x1C` in `Var2` at address `0x0121 = 0001 0010 0001 = D'289'`

```
movlw  Var2          ;   W = 0010 0001 = 0x21, not 0x0121
movwf  FSR           ; FSR = 0010 0001 = 0x21
movlw  0x1C
movwf  INDF
```

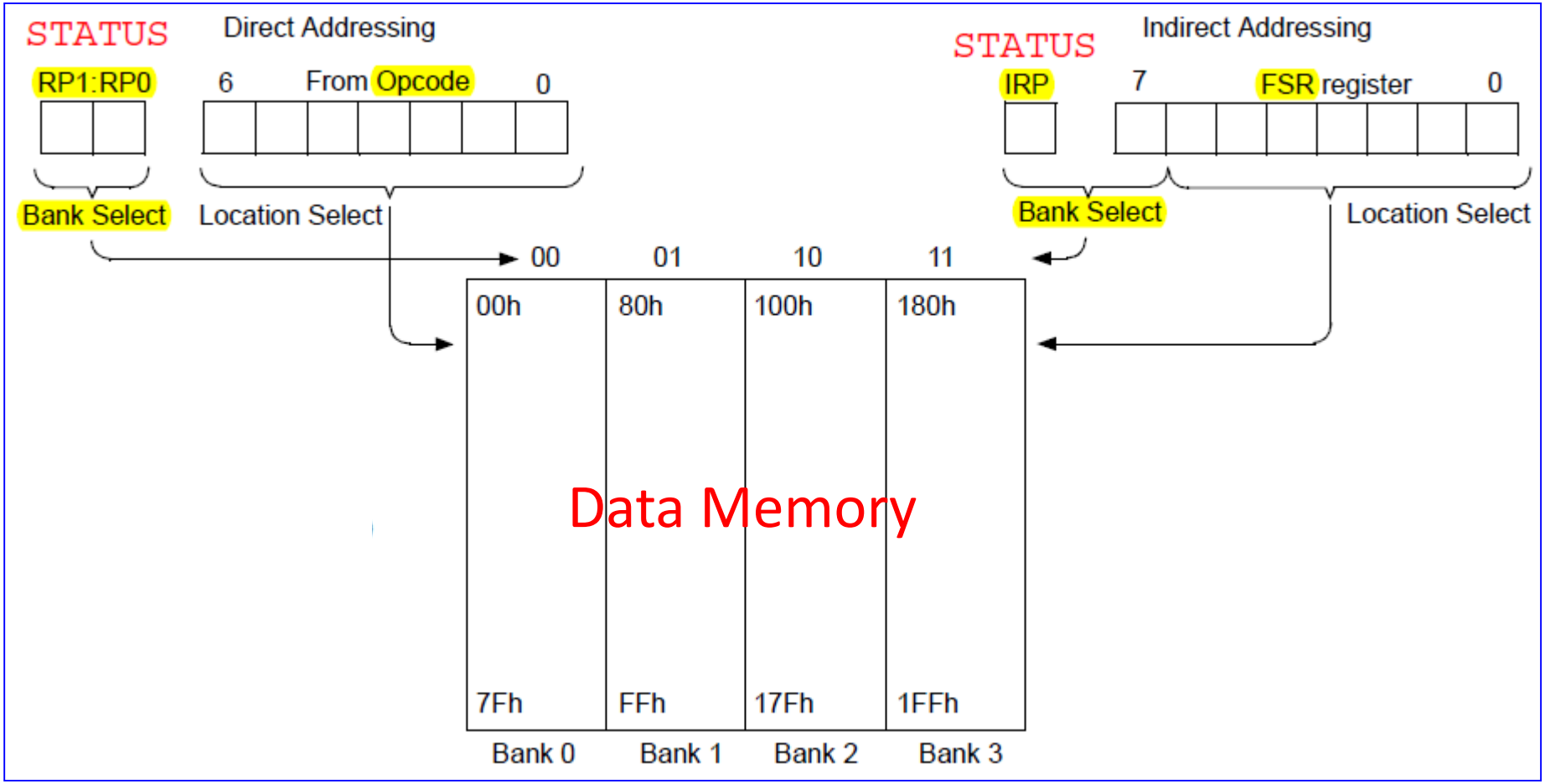
This will write `0x1C` to `0x21` because `FSR` is an 8-bit register, so `FSR` contains `0x21` instead of `0x0121`. We need 9 bits for addresses in data memory ($2^9 = 512$).

Solution: Indirect addressing uses the Indirect Register Pointer (IRP) bit in the STATUS register to get the 9th address bit:

```
movlw  Var2          ;   W = 0010 0001 = 0x21, not 0x121
movwf  FSR           ; FSR = 0010 0001 = 0x21
bsf    STATUS, IRP   ; Assembler will use 1 0010 0001
movlw  0x1C          ; for the address of Var2.
movwf  INDF          ; Var2 = 0x1C
```

Indirect Addressing

Indirect addressing uses the Indirect Register Pointer (IRP) bit in the STATUS register to get the 9th address bit.



Indirect Addressing

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	$\overline{TO}$	$\overline{PD}$	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)

We can also use the bankisel (bank indirect select) assembly directive to select the correct value for the IRP bit:

```
movlw       Var2    ;   W = 0010 0001 = 0x21, not 0x0121
movwf       FSR     ; FSR = 0010 0001 = 0x21
bankisel    Var2    ; Assembler will use 1 0010 0001
movlw       0x1C    ; for the address of Var2.
movwf       INDF    ; Var2 = 0x1C
```

Indirect Addressing

More efficient code

1. Initialize consecutive memory addresses
2. Prevent switching between banks

Prevent Bank Switching

Direct Addressing

```
banksel     Reg2
read        Reg2
⋮
banksel     PORTC
write       PORTC
⋮

banksel     Reg2
read        Reg2
⋮

banksel     PORTC
write       PORTC
⋮
```

Indirect Addressing

```
banksel     PORTC
bankisel    Reg2
FSR = address of Reg2

read        INDF
write       PORTC
⋮

read        INDF
write       PORTC
⋮

read        INDF
write       PORTC
⋮
```

Initialize consecutive addresses

Direct Addressing

```
clrf   Var1
clrf   Var2
…
clrf   VarN
```

N instructions

Indirect Addressing

```
        movlw   Var1
        movwf   FSR
Next:   clrf    INDF
        incf    FSR, F
        movf    FSR, W
        sublw   VarN+1
        btfss   STATUS, Z
        goto    Next
```

8 instructions

End Lab 9