ECE 354 – Computer Systems Lab II

Microcontroller Architecture

Tilman Wolf

UMass Amherst

Introduction Lab

- Difference between IDE and ICD?
- Steps to change code?
- Demo board
- MPLAB software

Lab Hours

- The Marston 228 Lab is open:
  - Mondays 2:30 - 7:00
    (TAs: Avi (2:30 - 4:00), Barron (4:00 - 7:00))
  - Tuesdays 4:00 - 7:00
    (TA: Avi)
  - Wednesdays 2:30 - 7:00
    (TAs: Avi (2:30 - 4:00), Barron (4:00 - 7:00))
  - Thursdays 4:00 - 7:00
    (TA: Matt)
  - Fridays 1:00 - 4:00
    (TA: Matt)

Outline

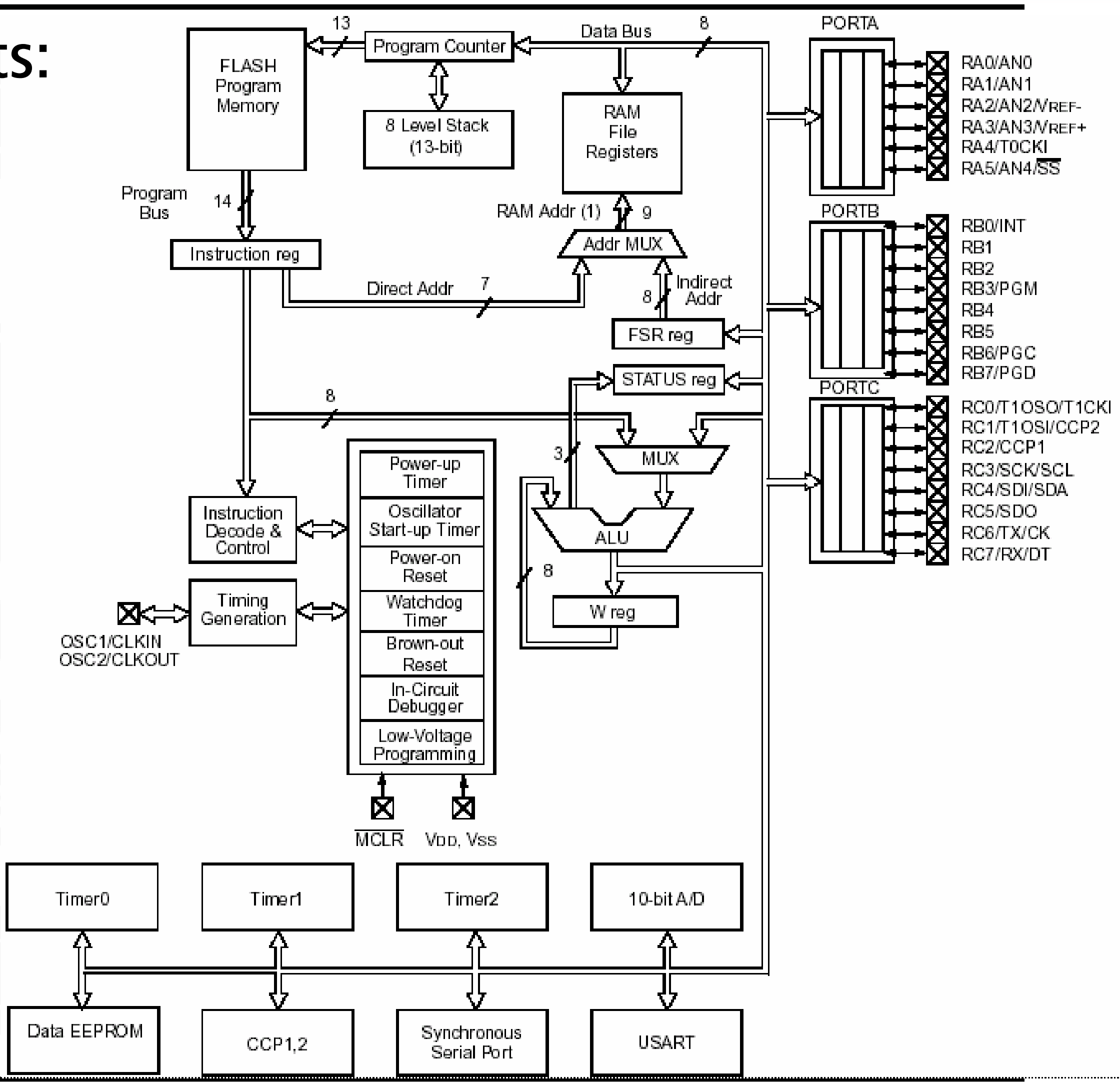
- Microcontroller Architecture (PIC16F877)
  - Basic Architecture
  - Memories
  - Registers
  - Instructions
- Serial Communication
  - UART
  - Tx and Rx
- Lab 1
- Assembly Tutorial

Computer Architecture

- What is the minimum a computer needs?
  - Memory (instruction, data, or combined)
  - Processor/ALU
  - I/O
- What are the basic processing steps?
  - Instruction fetch
  - Instruction decode
  - Memory read
  - ALU operation
  - Write-back
- Details depend on particular architecture

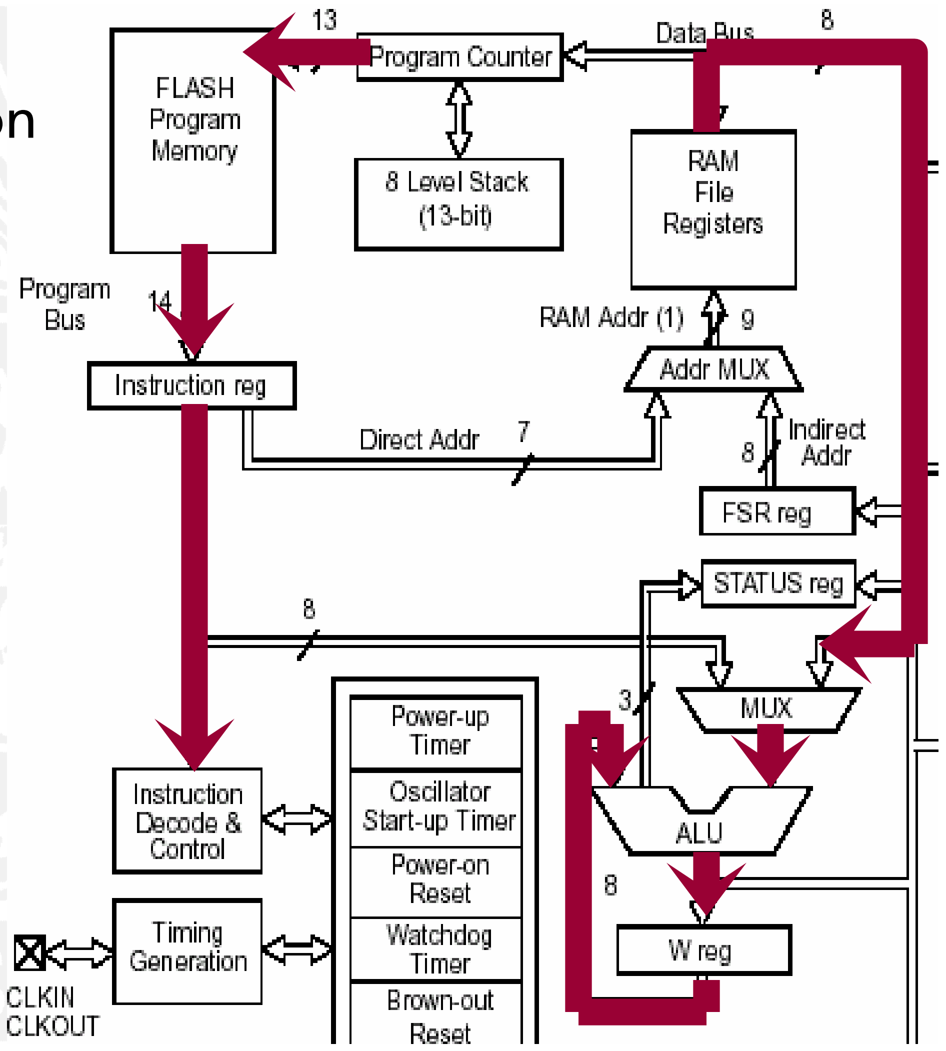
PIC16F877 Architecture

- Main PIC components:
  - ALU
  - Program memory
  - Register file
  - Program counter
  - Status register
  - Clock
  - I/O ports
  - Timers
  - A/D converter
  - USART



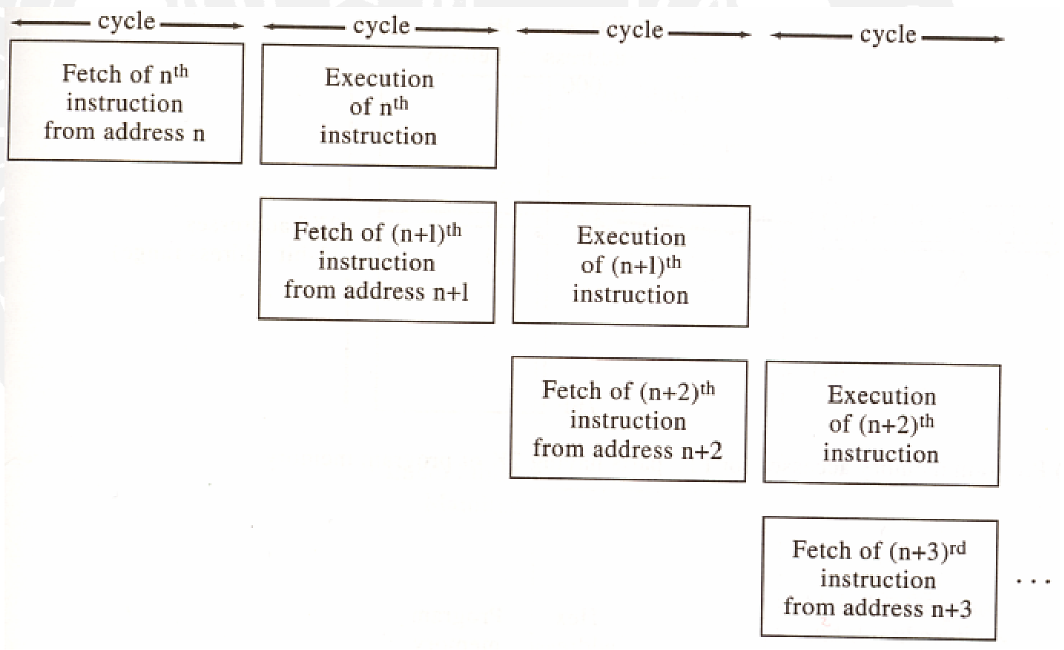
Processing on PIC

- Basic Steps:
  - Determine current instruction based on program counter
  - Load instruction into instruction register
  - Decode instruction
  - Read register
  - Perform ALU operation
  - Write back result
- Additional data paths for
  - Change in program counter
  - Immediate values
  - I/O operation



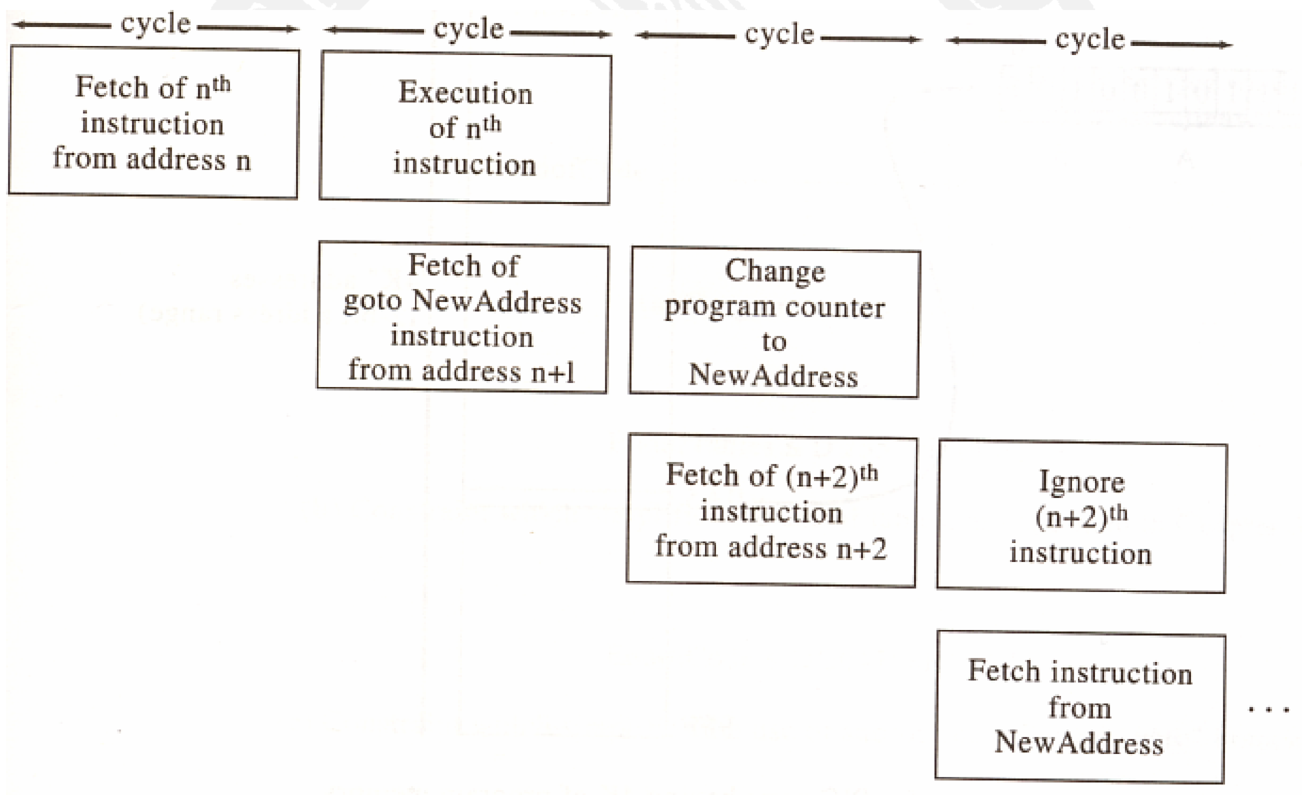
Pipelining

- Harvard architecture
  - Separate program and data memory
- Observation
  - Program memory is idle while data memory is in use
  - Accesses could be interleaved
- Pipelining:
  - 2 stages


cycle
cycle
cycle
cycle
Fetch of nth instruction from address n
Execution of nth instruction
Fetch of (n+1)th instruction from address n+1
Execution of (n+1)th instruction
Fetch of (n+2)th instruction from address n+2
Execution of (n+2)th instruction
Fetch of (n+3)rd instruction from address n+3
. . .

Pipeline Stalls

- Causes for pipeline stalls
  - Control dependencies
  - Data dependencies

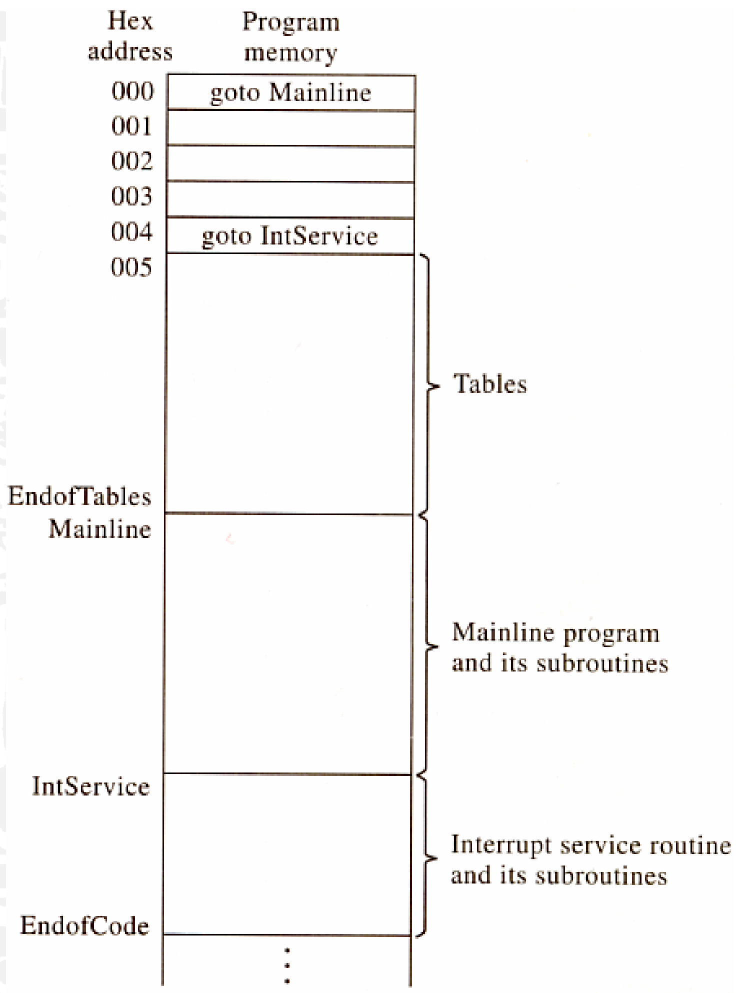

cycle
cycle
cycle
cycle
Fetch of nth instruction from address n
Execution of nth instruction
Fetch of goto NewAddress instruction from address n+1
Change program counter to NewAddress
Fetch of (n+2)th instruction from address n+2
Ignore (n+2)th instruction
Fetch instruction from NewAddress
. . .

Instruction Memory

- How many bits do we need to address memory?
  - E.g., how many bits do we need to address 4kbit of memory?
- Well, it depends…
  - What is the smallest unit that we need to address?
  - E.g., 8-bit addressable: 4kb/8b = 512 words, requires $\log_2(512)$ bits
- Instruction memory on PIC16F877
  - 8K instructions
  - Instruction size: 14 bits
  - How many address bits do we need?

Instruction Memory

- Memory map
- Special instructions
  - 0x000 start of program is single goto instruction
  - 0x004 goto to interrupt service routine
- Memory map is created by IDE software when project is built

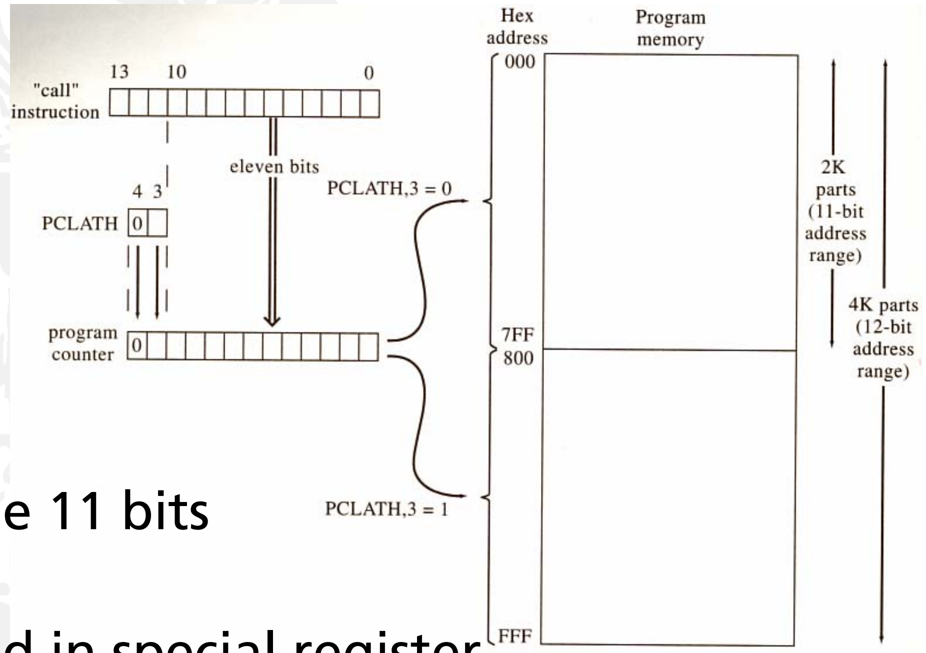


(a) Program memory map

UMass Amherst

Call Instructions

- PIC designers needed to save bits wherever possible
  - Lots of “hacks” necessary to exploit full functionality
- Example: call instruction
  - Basically changes program counter
  - There are also other effect that we’ll discuss later
- Program counter is 13 bits
  - Call function can only provide 11 bits (why?)
  - Two additional bits are stored in special register
- Calls within current 2K instruction block are “cheaper”
  - Why?



Tilman Wolf

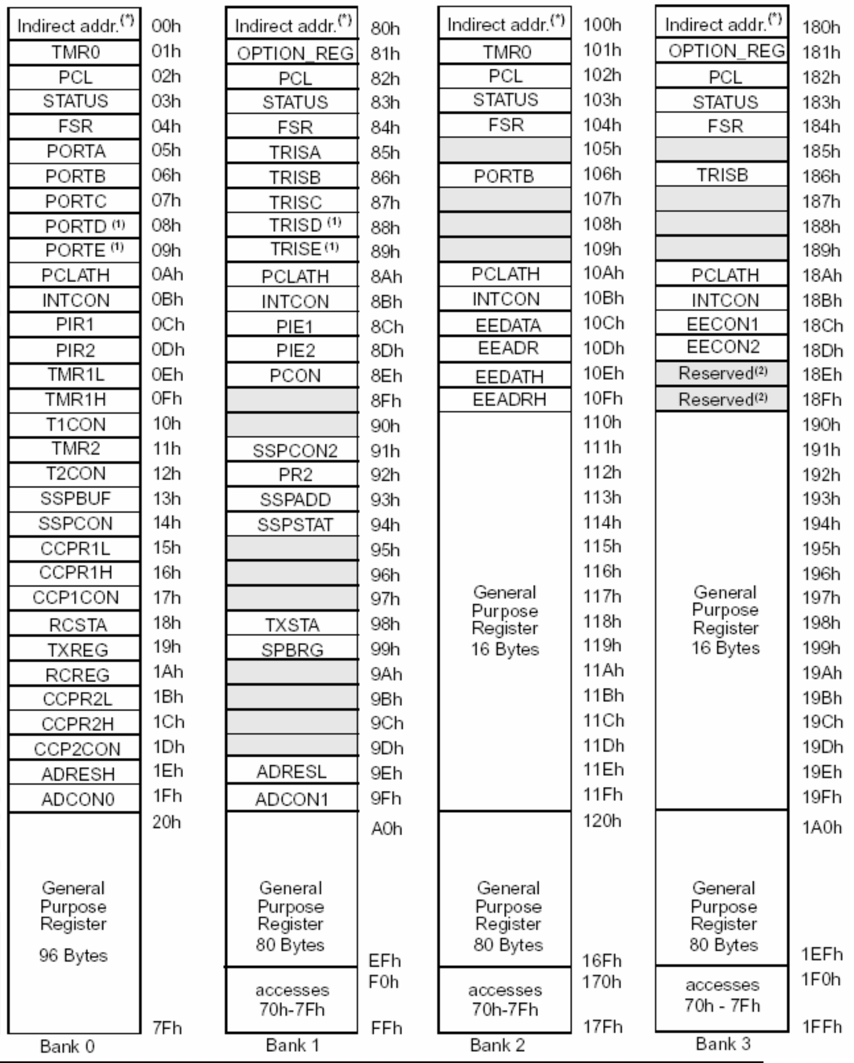
UMass Amherst

Reset

- What happens on RESET?
- Two possible causes for RESET
  - Power applied to 16F877
  - MCLR (master clear) asserted active low
- PC automatically cleared to 0x000
  - Reset vector stored at 0x000
  - Program counter jumps to actual program start
- Program may start at 0x005 or higher address

Register File

- Registers are data memory
  - Most registers are general-purpose
  - Some are special-purpose
- Each register holds 8-bit value
- Registers are separated into banks
  - 128 registers per bank
  - PIC16F877 has 4 banks
- Why use banks?



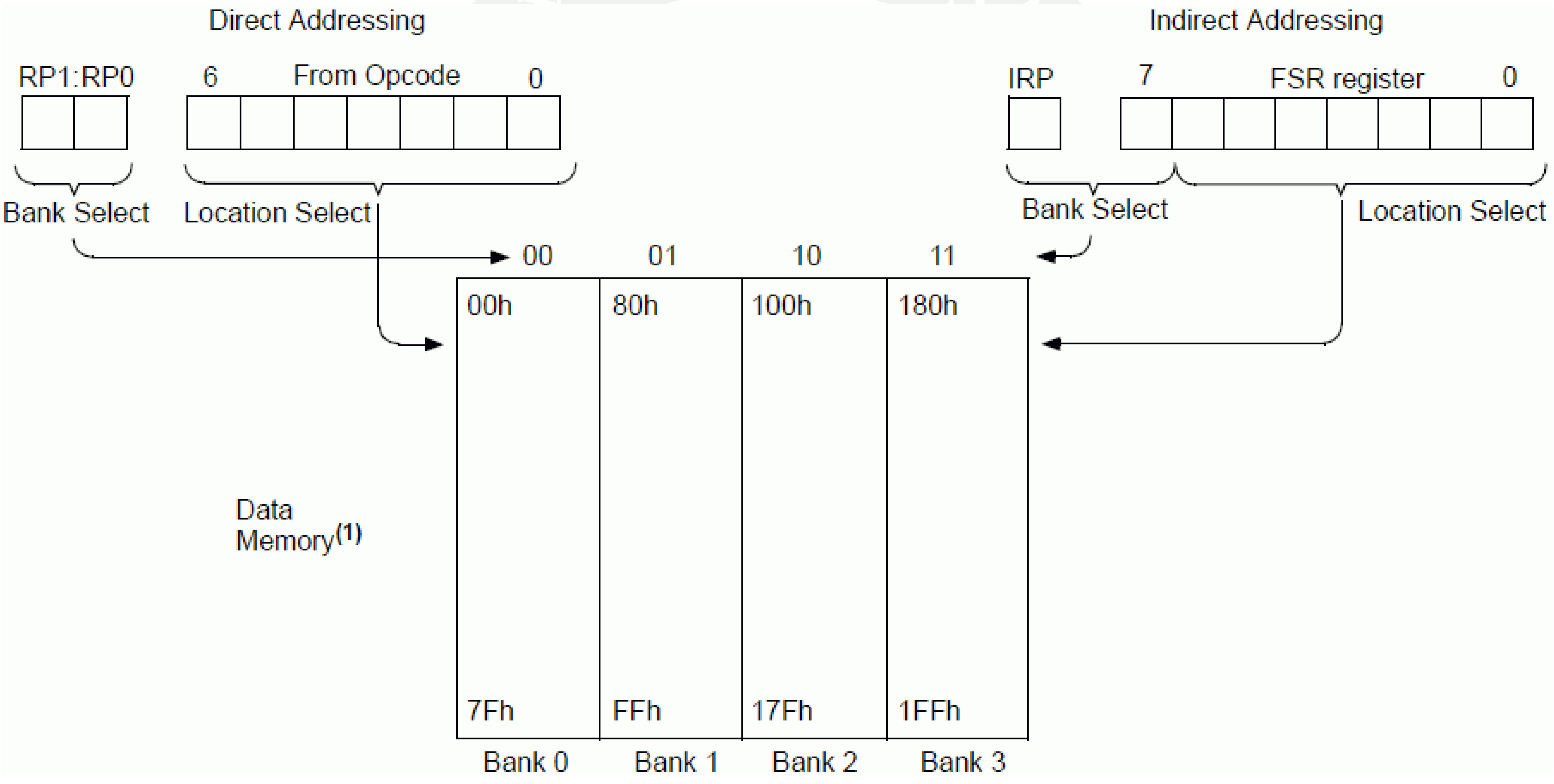
Bank 0		Bank 1		Bank 2		Bank 3	
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	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		A0h		120h		1A0h
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		FFh		17Fh		1FFh

Tilman Wolf

UMassAmherst

Register File Addressing

- “Bank Select” bits choose bank (2 bits)

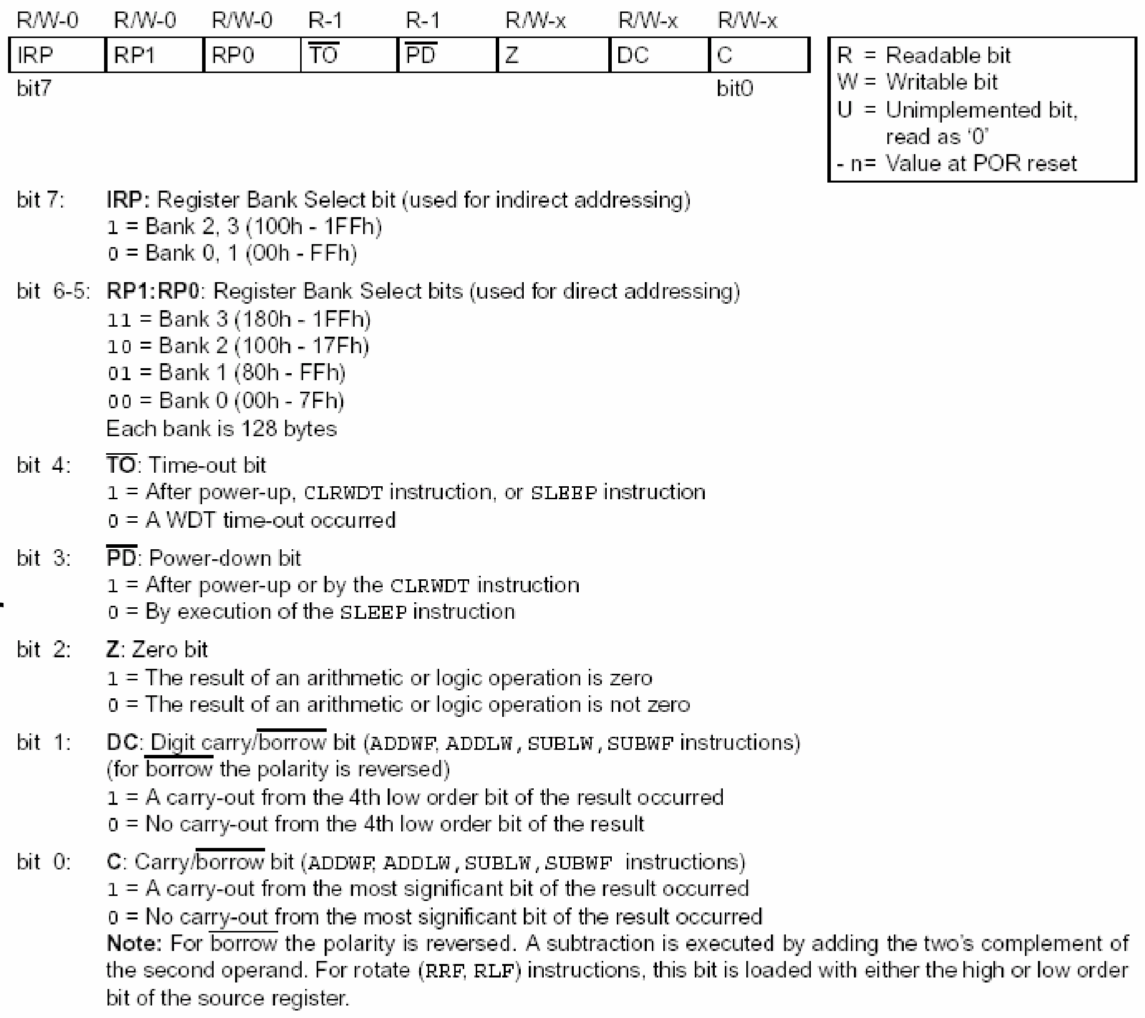


CPU Registers

- Special registers
  - Working register
  - STATUS register
  - FSR (File Select Register)
  - INDF register
  - Program counter (12 bits)
    - PCLATH (Program Counter Latch) (4 bits)
    - PCL (8 bits)
  - Eight-level stack
- We'll discuss details in other lectures
  - For Lab 1: Working register and STATUS register

STATUS Register

- Status bits
  - Bit 0: Carry
  - Bit 1: Digit carry
  - Bit 2: Zero result
  - Bits 3 & 4: Use at power-up and sleep
  - Bit 5 & 6: bank select
  - Bit 7: bank select for indirect addressing



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
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n= Value at POR reset

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, `CLRWDT` 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 `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

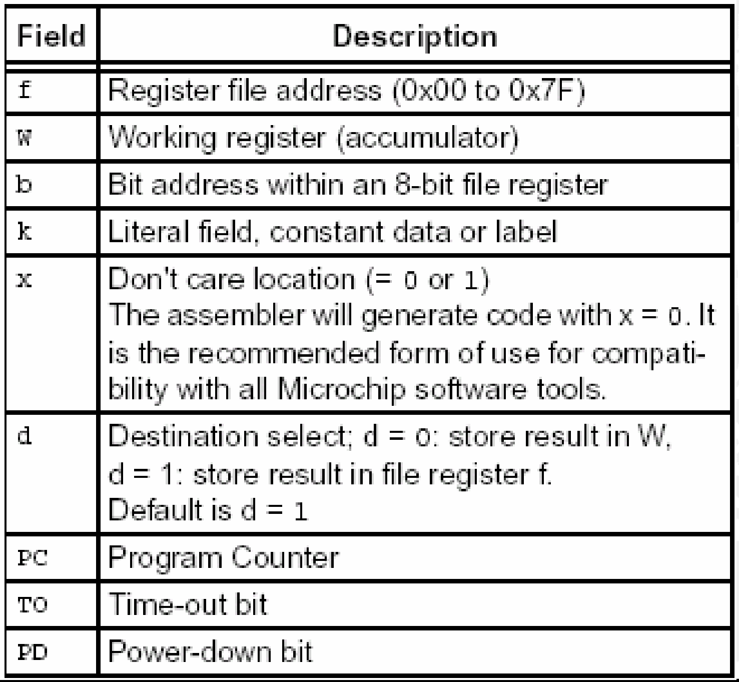
bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
(for $\overline{\text{borrow}}$ 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

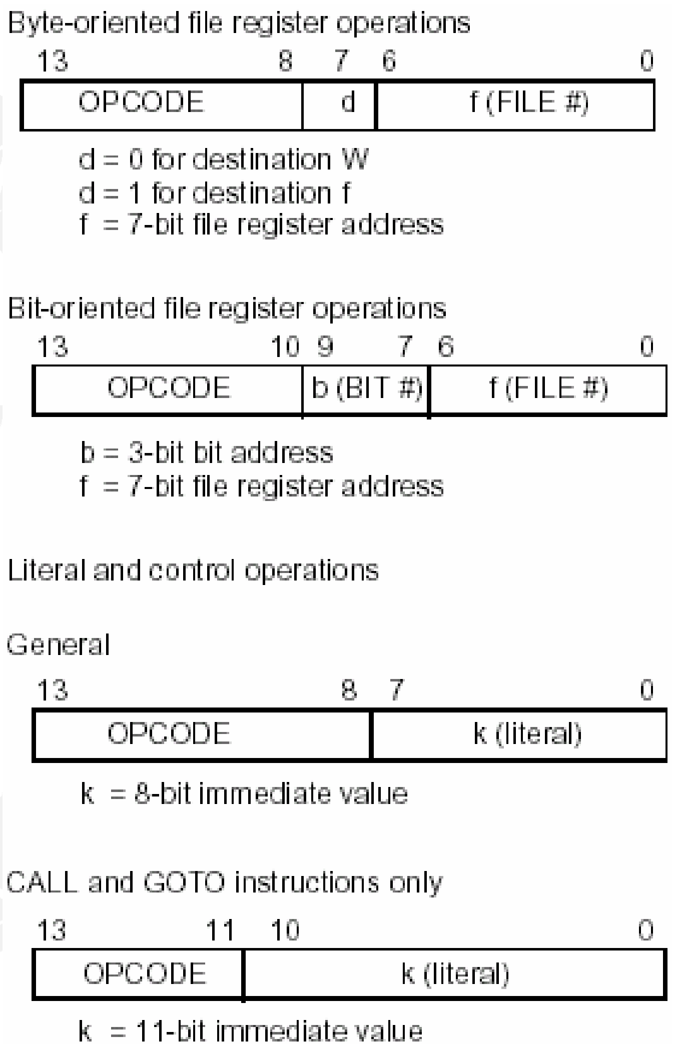
bit 0: **C**: Carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{\text{borrow}}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Instructions

- Instruction format:
  - OPCODE determines instrucion
  - Registers, bits, literals depend on OPCODE
- OPCODE fields:

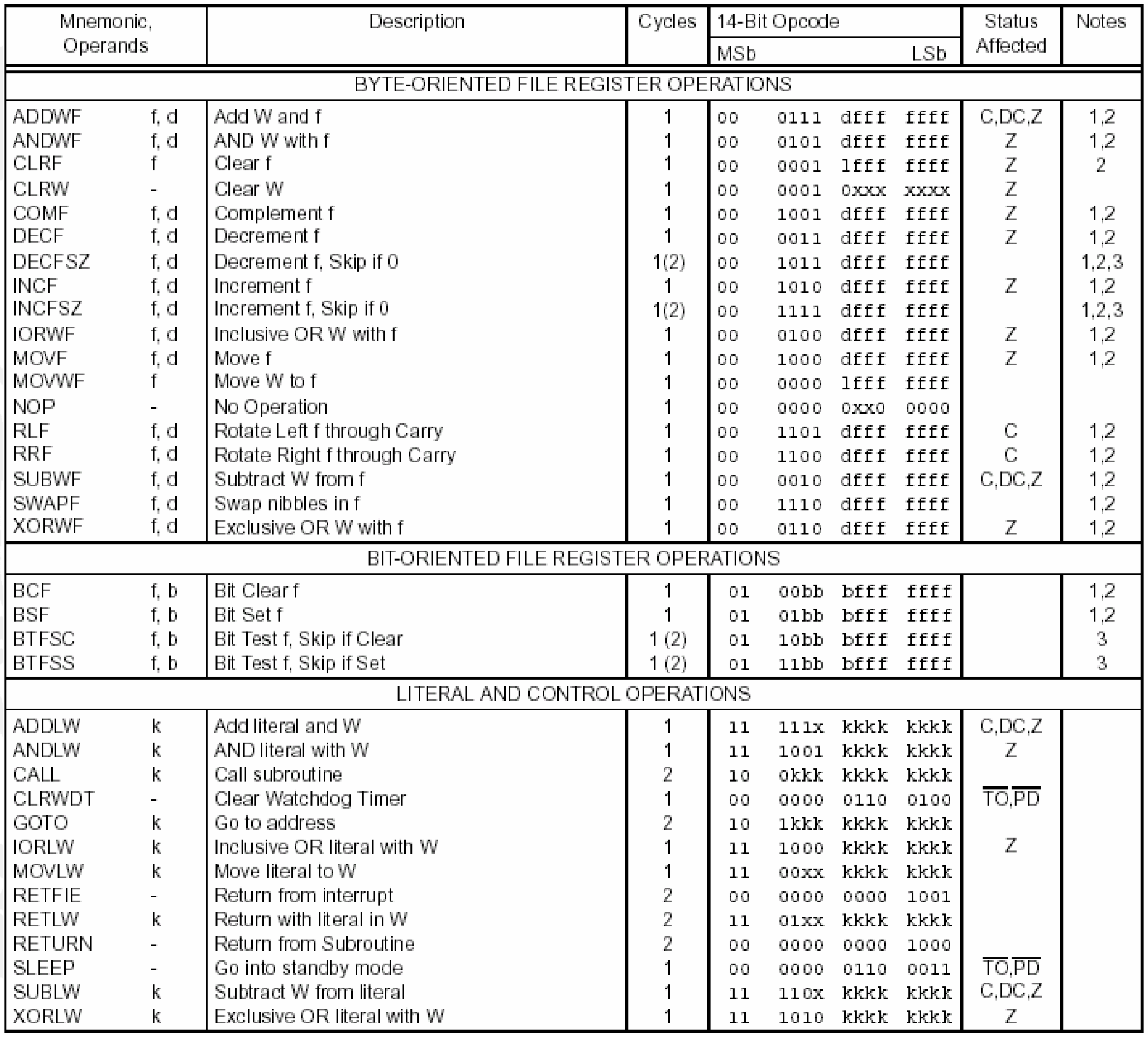


Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
PC	Program Counter
TO	Time-out bit
PD	Power-down bit


Byte-oriented file register operations
13 8 7 6 0
OPCODE
d
f (FILE #)
d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address
Bit-oriented file register operations
13 10 9 7 6 0
OPCODE
b (BIT #)
f (FILE #)
b = 3-bit bit address
f = 7-bit file register address
Literal and control operations
General
13 8 7 0
OPCODE
k (literal)
k = 8-bit immediate value
CALL and GOTO instructions only
13 11 10 0
OPCODE
k (literal)
k = 11-bit immediate value

Instruction Set

- 35 instructions
  - OPCODE
  - Letters indicate format
    - F, W
  - Z indicates conditional execution
- More details
  - Datasheet pp. 139-144
  - Peatman pp. 25, 27-28



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	lfff	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	lfff	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	

Assembler

- Creating instructions “by hand” is difficult
- Binary code specifies op-code and values of operands
  - “11 1110 1000 0111” adds 135 to working register
- Assembler translates “readable” code into binary
  - “ADDLW 135” means “add literal 135 to working register”
  - Assembler converts this to “11 1110 1000 0111”
- Other convenient features
  - Labels for branches and jumps (e.g., “bug” and “start”)
  - Register addresses can be named (e.g., “c1 equ 0x0c”)

Lab 1

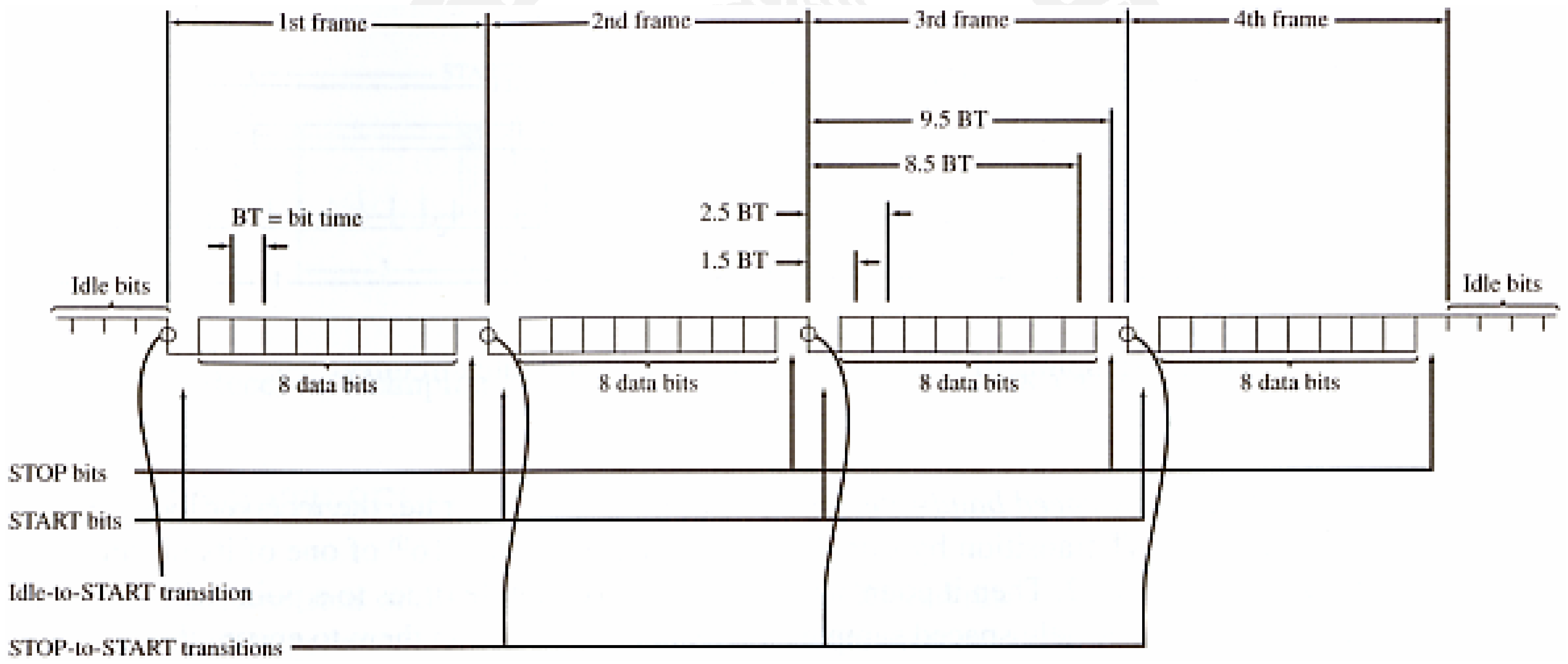
- Connect PIC to terminal
  - PIC in stand-alone mode using USART interface
  - Mostly software development
- Required functionality:
  - Three bits of port A connected to switches
  - Value of port A is shown on LEDs on port B
  - PIC sends “Number?” to terminal
  - User presses key, value is sent to PIC and echoed
  - If user presses ‘0’-’7’, PIC compares value to port A and sends response to terminal: “equal” or “not equal”
  - Send response every time switch values change
  - Repeat with user input

UART/USART

- "<u>U</u>niversal <u>S</u>ynchronous/<u>A</u>synchronous <u>R</u>eceiver/<u>T</u>ransmitter"
- Serial data communication between PIC and Terminal
- Two cables (receive and transmit)
- Each 1-byte character is transmitted separately
  - Start, 8 data bits, 1 parity bit, Stop
- We use asynchronous mode
  - Sender uses local clock
- Baud rate specifies speed of transmission

Synchronization

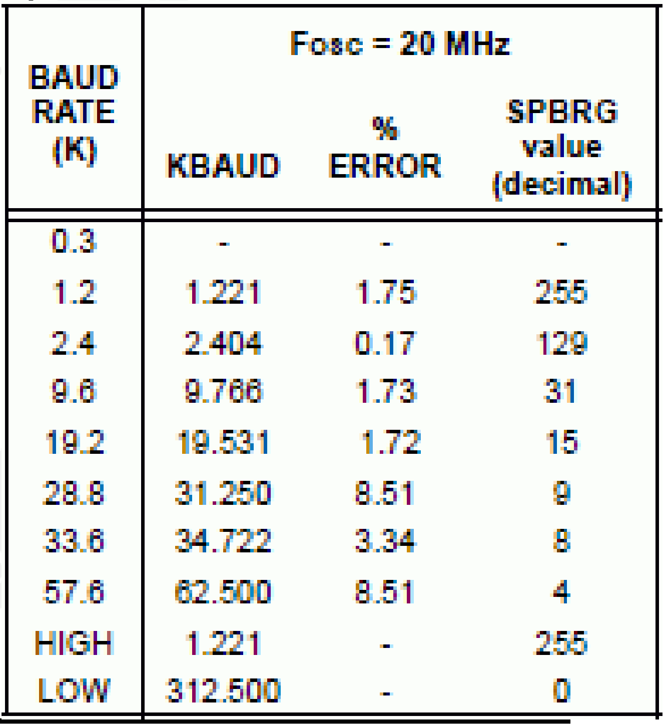
- Clocks on sender and receiver are never exactly in sync
  - Requires synchronization of receiver
  - High-low transition signals frame boundary



UMass Amherst

UART on PIC16F877

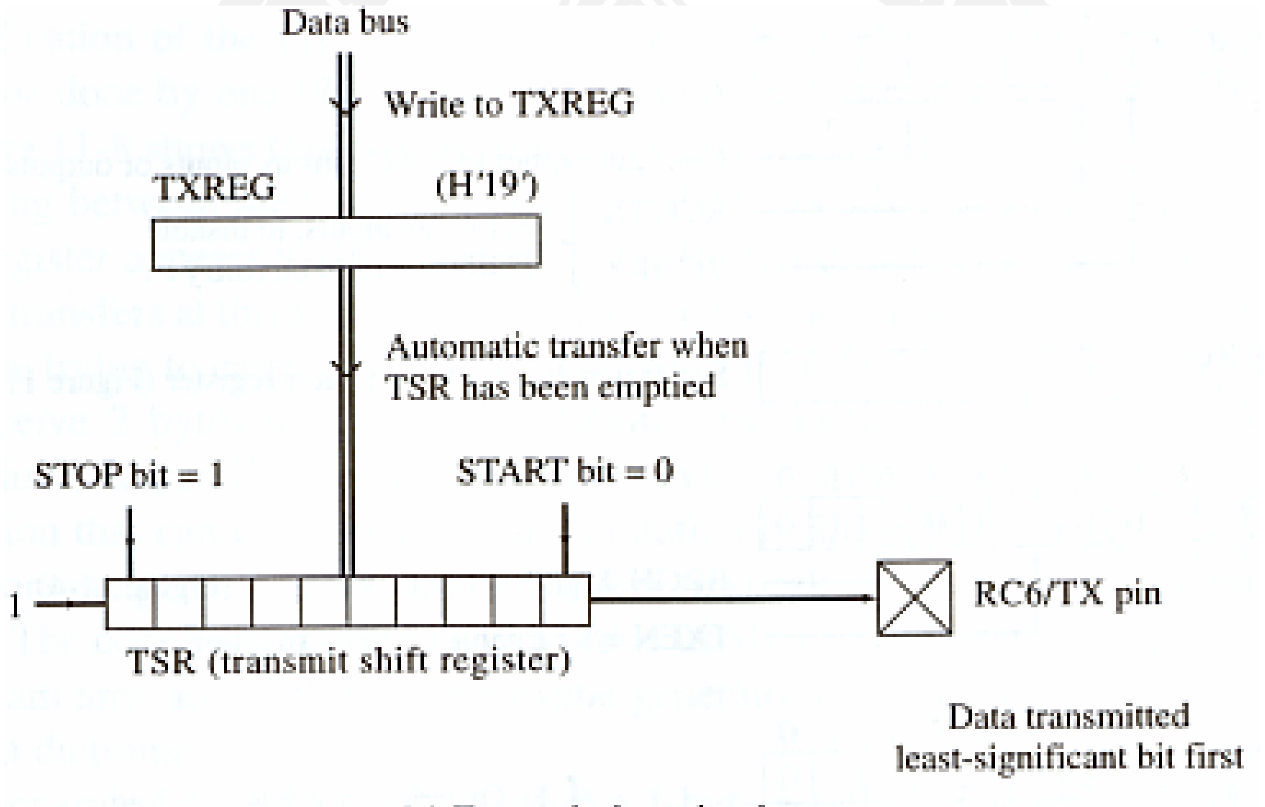
- Several registers involved
- Control/status registers:
  - TXSTA (transmit status and control register)
  - RCSTA (receive status and control register)
  - Configurations: enable bit, 8/9 bit, buffer full, etc.
- Baud rate generator:
  - SPBGR
  - Data sheet table 10-3 shows value for different clock and baud rates
- Data registers:
  - TXREG and RCREG
- Tx and Rx completion:
  - PIR1<4> and PIR1<5>
    set when TXREG clears and RCREG is filled
- Optional: enable bit in PIE1 for interrupt



BAUD RATE (K)	Fosc = 20 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-
1.2	1.221	1.75	255
2.4	2.404	0.17	129
9.6	9.766	1.73	31
19.2	19.531	1.72	15
28.8	31.250	8.51	9
33.6	34.722	3.34	8
57.6	62.500	8.51	4
HIGH	1.221	-	255
LOW	312.500	-	0

UART Transmission

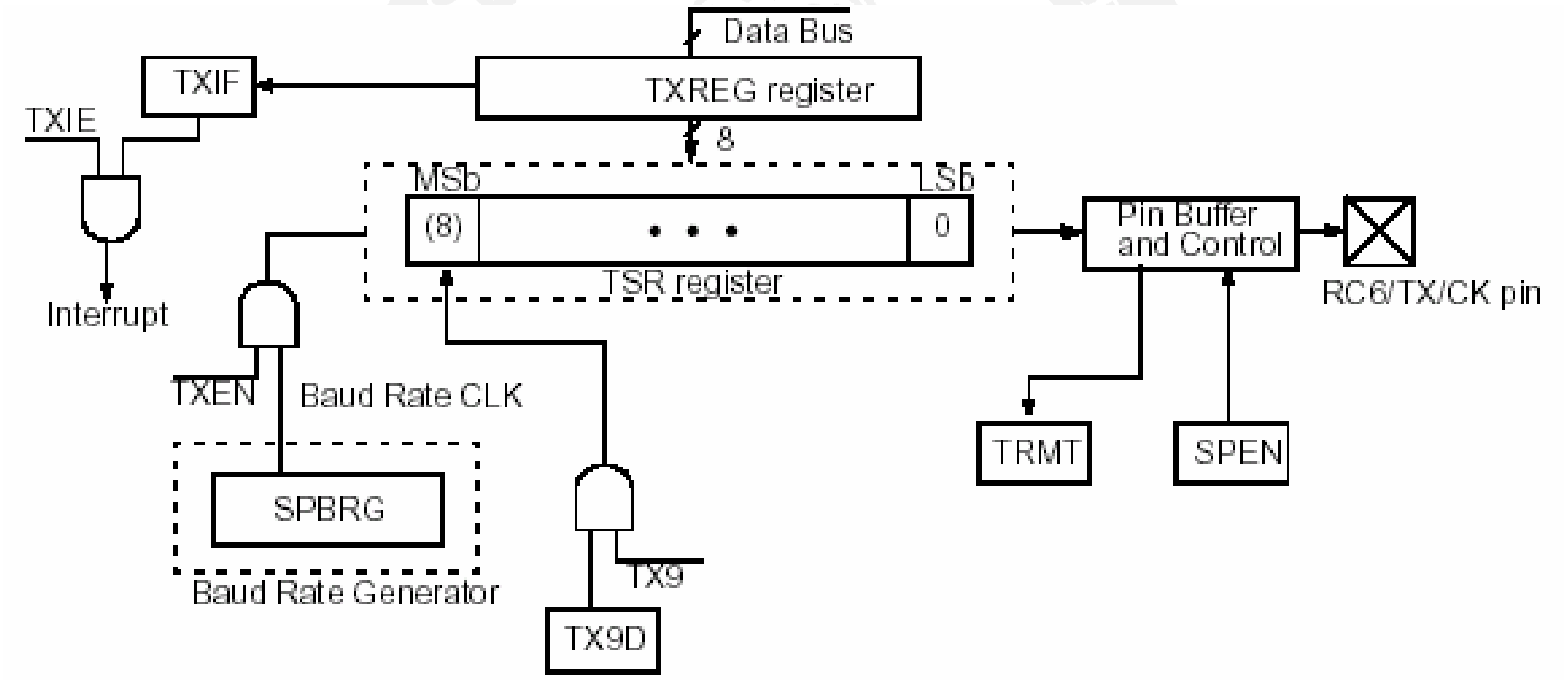
- TXREG is accessed from program
  - Need to check if empty before writing next value (TXSTA)


Data bus
Write to TXREG
TXREG
(H'19')
Automatic transfer when TSR has been emptied
STOP bit = 1
START bit = 0
1
RC6/TX pin
TSR (transmit shift register)
Data transmitted least-significant bit first

(a) Transmit data circuit

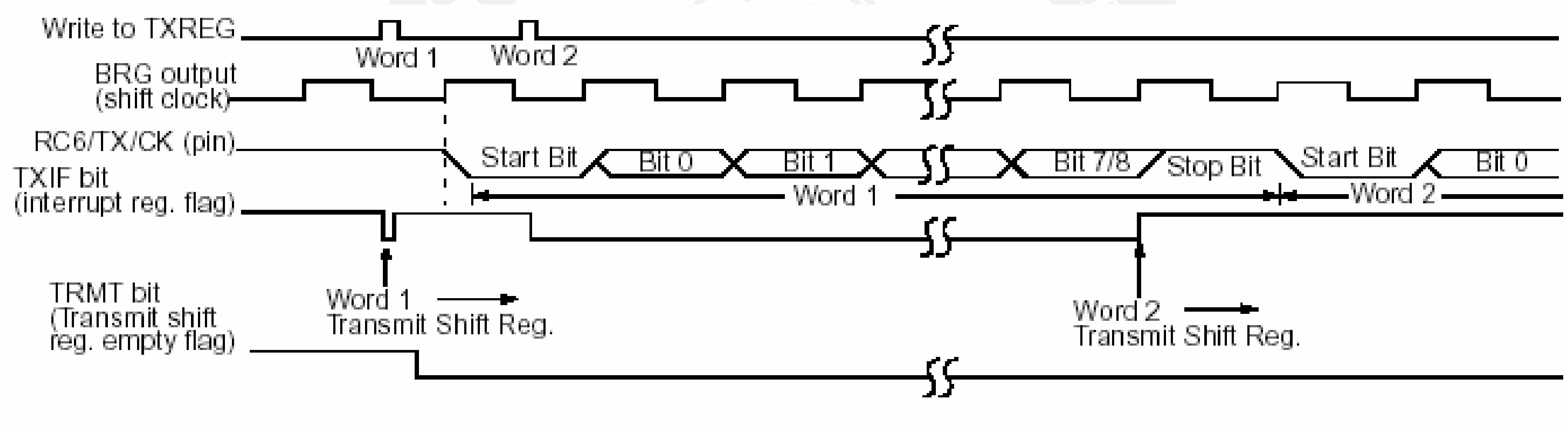
UART Transmission

- Actually, it's more complicating
  - Ensure all related registers are configured correctly



UART Tx Signals

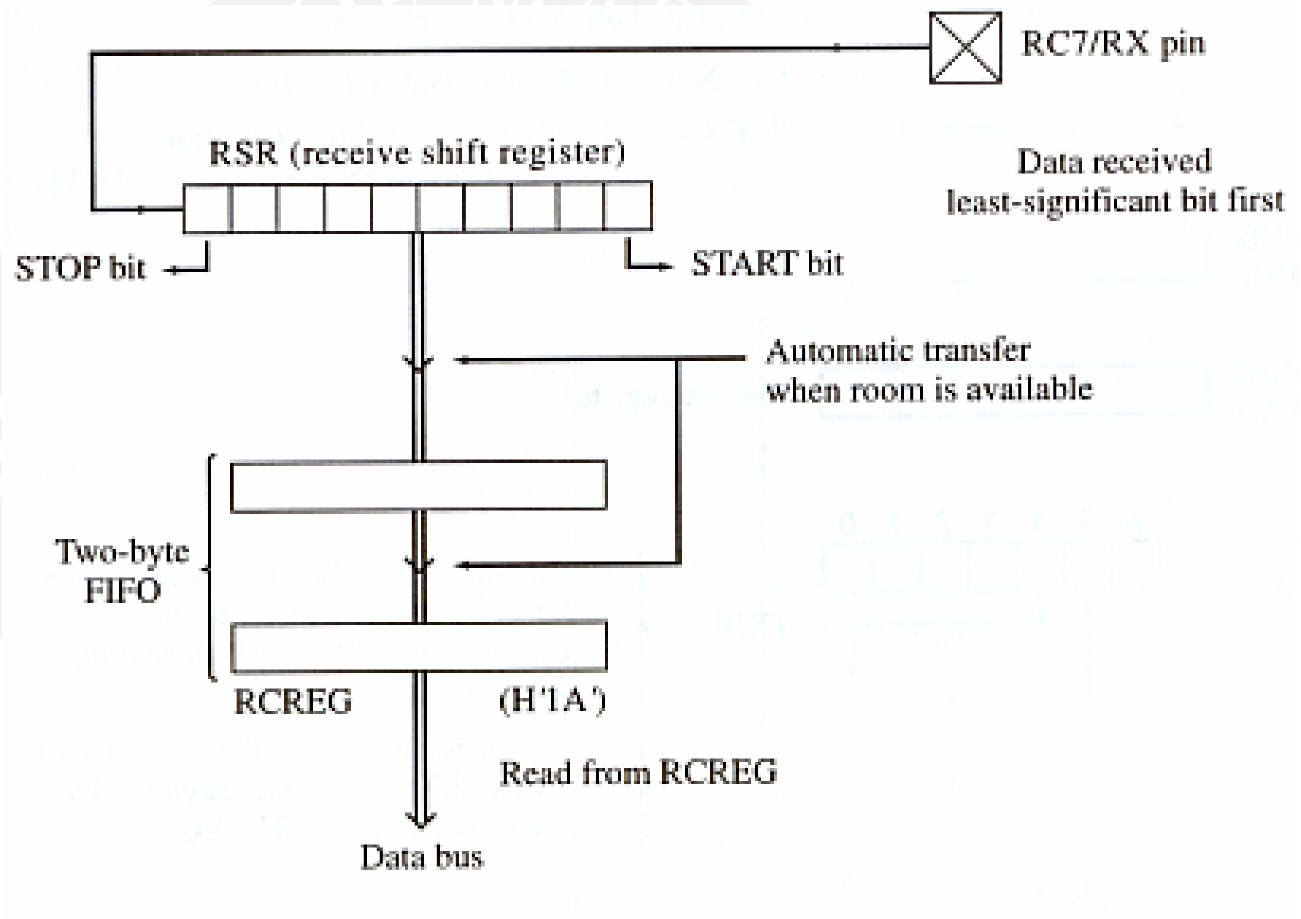
- Example for two transmissions



Note: This timing diagram shows two consecutive transmissions.

UART Receiving

- 2-byte FIFO



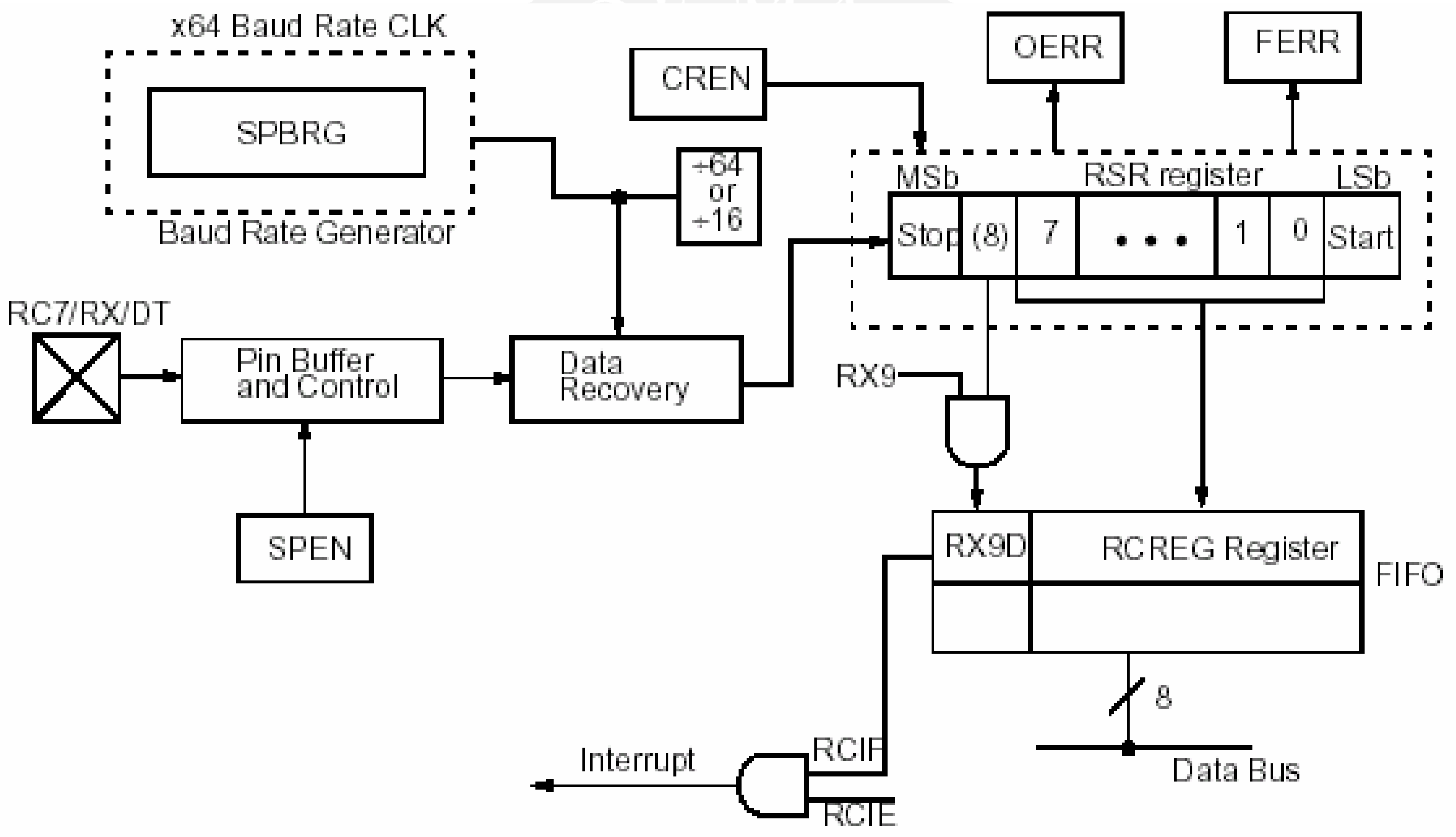
(b) Receive data circuit

Tilman Wolf

UMass Amherst

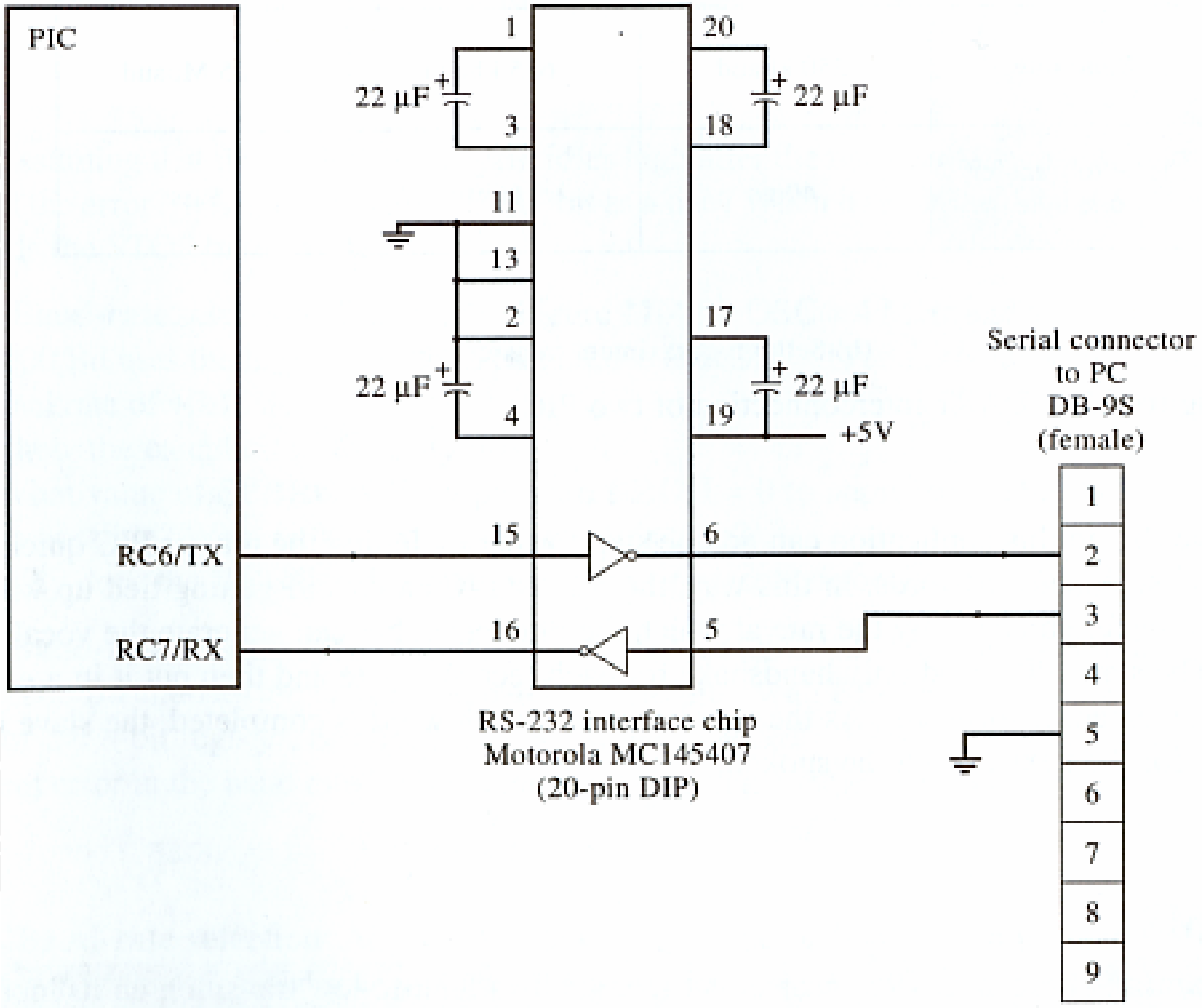
UART Receiving

- Details:



Hardware Setup

- Requires MAX232 driver
  - PIC: 0/+5V
  - RS-232 interface: ±10V
- See Peatman Ch. 11



UMass Amherst

Lab 1

- BEFORE YOU START: READ!
  - Lab Assignment
  - Data sheet pp. 29 – 33 (port I/O)
  - Data sheet pp. 95 – 104 (UART)
  - Peatman, chapter 11 (UART)
  - MAX232 data sheet
- Quiz will have a few simple questions regarding lab
- Think about how you want to split work
- Think about steps to take to get it working
- Start working early!
  - Lab schedule starts today