

Introduction to Embedded Microcomputer Control

Topics

- Number Systems (decimal, binary, octal, hexadecimal)
- Assembly Language Programming
 - Instruction Set [Reduced Instruction Set Computer (RISC)]
- Code Simulation and Debugging
- Exercises
- Microcontrollers from MicroChip Inc.
- Microcontroller Architecture
 - Central Processing Unit (CPU)
 - Arithmetic Logic Unit (ALU)
 - Memory Organization
 - Input / Output (I / O)
 - Interrupts

Decimal Number System

- Decimal – 10 digits \Rightarrow 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Each place (left to right) corresponds to a power of 10.
 - Read left to right
 - Can call the leftmost digit the most significant digit

10^5	10^4	10^3	10^2	10^1	10^0
100,000	10,000	1,000	100	10	1
8	7	4	0	3	1

most significant digit

least significant digit

$$8 * 10^5 + 7 * 10^4 + 4 * 10^3 + 0 * 10^2 + 3 * 10^1 + 1 * 10^0 = 874,031_{\text{dec}}$$

Binary Number System

- Binary – 2 digits \Rightarrow 0, 1
 - Each place corresponds to a power of 2.
 - Read left to right

There are 10 kinds of people in this world - - those that understand binary and those that don't.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	1	0	0	1	1

most significant digit

least significant digit

$$11010011_{\text{bin}} = 1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 211_{\text{dec}}$$

Octal

- Octal – 8 digits \Rightarrow 0, 1, 2, 3, 4, 5, 6, 7
 - Each place corresponds to a power of 8.
 - Each octal digit is equivalent to 3 binary digits.

2^8 256	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
1	1	1	0	1	0	0	1	1
8^2 64			8^1 8			8^0 1		
7			2			3		

$$723_{\text{oct}} = 7 * 8^2 + 2 * 8^1 + 3 * 8^0 = 467_{\text{dec}}$$

Hexidecimal (Hex)

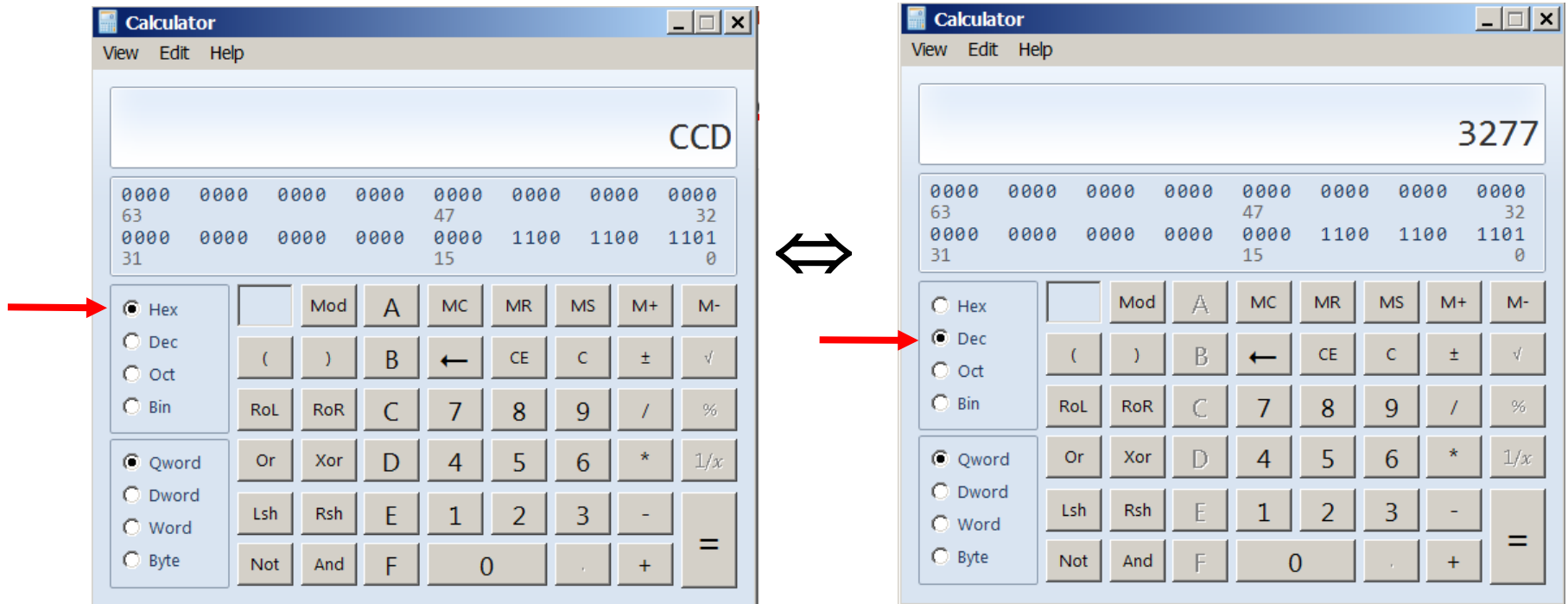
- Hexidecimal – 16 digits \Rightarrow
 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
 - Each place corresponds to a power of 16.
 - Each hex digit is equivalent to 4 binary digits.

2^{15} 65536	2^{14} 16384	2^{13} 8192	2^{12} 4096	2^{11} 2048	2^{10} 1024	2^9 512	2^8 256	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
1	0	0	1	1	1	1	1	1	1	0	1	0	0	1	1
16^3 4096				16^2 256				16^1 16				16^0 1			
9				F				C				3			

$$\begin{aligned}
 9FC3_{\text{hex}} &= 9 * 16^3 + F * 16^2 + C * 16^1 + 3 * 16^0 + 2 * 8^1 + 3 * 8^0 = 40,899_{\text{dec}} \\
 &= 9 * 4096 + 15 * 256 + 13 * 16 + 3 * 1
 \end{aligned}$$

Microsoft Windows Calculator

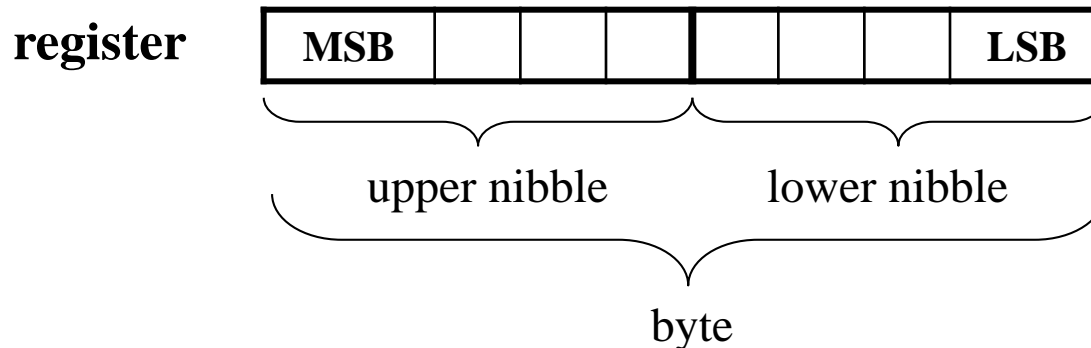
Does number system conversion (programmer view)



Note: digits are grayed

Data Storage

- In the microcontroller, data is stored in an 8 bit binary register
- Terminology
 - One digit \Rightarrow bit (**B**inary dig**IT**)
 - Four bits \Rightarrow nibble
 - Eight bits \Rightarrow byte (**B**inar**Y** Tupl**E**)
 - Sixteen bits \Rightarrow word (2 registers)
 - MSB (Most Significant Bit) \Rightarrow left most bit in register $2^7 = 128$
 - LSB (Least Significant Bit) \Rightarrow right most bit in register $2^0 = 1$



Data Interpretation

Binary data in a register can mean different things depending on the use in the program.

In Assembly programming, you (the programmer) supply the meaning.

- Numeric (binary number)
 - Unsigned
 - Sign Magnitude
 - 2's Complement
- Logical
 - Bitwise $0 \Rightarrow \text{False}$ $1 \Rightarrow \text{True}$ $110010001 \Rightarrow \text{TTFFTFFFT}$
 - Register $00000000 \Rightarrow \text{False}$ $\text{non-zero} \Rightarrow \text{True}$
- ASCII (character representation) 7 bits 128 characters
 - $01000001 \Rightarrow \text{A}$ $01110101 \Rightarrow \text{u}$ $01001110 \Rightarrow \text{N}$

Aside: in C, you tell the compiler the meaning by a declaration. There is nothing equivalent to this in Assembly.

Numeric Data

- Unsigned

- Range $0 \leftrightarrow 255$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	1

211

- Sign Magnitude

- Typically not used
- Range $-127 \leftrightarrow +127$

sign bit	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	1

- 83

sign bit: 1 \Rightarrow negative 0 \Rightarrow positive

- 2's Complement

- Most often used
- Positive & negative numbers
- Range $-128 \leftrightarrow +127$

ind	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	0	1	1

- 45

Making a Number Negative


To represent a negative number in 2's complement form :

take the positive number in binary	45	\Rightarrow	0 0 1 0 1 1 0 1	
complement it (change 0 to 1 & 1 to 0)			1 1 0 1 0 0 1 0	
add 1			1 1 0 1 0 0 1 1	$\Rightarrow -45$

take the positive number in binary	1	\Rightarrow	0 0 0 0 0 0 0 1	
complement it (change 0 to 1 & 1 to 0)			1 1 1 1 1 1 1 0	
add 1			1 1 1 1 1 1 1 1	$\Rightarrow -1$

To determine what the negative number in 2's complement form is :

take the negative number in binary	1 1 1 1 1 1 0 1	
complement it (change 0 to 1 & 1 to 0)	0 0 0 0 0 0 1 0	
add 1	0 0 0 0 0 0 1 1	\Rightarrow was -3



**Be careful with the Windows calculator & negative numbers.
It assumes much more than 8 bits.**

2's Complement Arithmetic

3	0	0	0	0	0	0	1	1	increment
2	0	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	1	↑
0	0	0	0	0	0	0	0	0	zero
-1	1	1	1	1	1	1	1	1	↓
-2	1	1	1	1	1	1	1	0	
-3	1	1	1	1	1	1	0	1	decrement

Rules

addition

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

subtraction

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$0 - 1 = 1 \text{ borrow } 1$$

$$1 - 1 = 0$$

$$\begin{array}{r} 00000011 \\ 00001010 \\ \hline 00001101 \end{array} \quad \begin{array}{r} 3 \\ + 10 \\ \hline 13 \end{array}$$

$$\begin{array}{r} 00000011 \\ 11110110 \\ \hline 11111001 \end{array} \quad \begin{array}{r} 3 \\ + (-10) \\ \hline -7 \end{array}$$

$$\begin{array}{r} 00000011 \\ 00001010 \\ \hline 11111001 \end{array} \quad \begin{array}{r} 3 \\ - 10 \\ \hline -7 \end{array}$$

$$\begin{array}{r} 00001010 \\ 00000011 \\ \hline 00000111 \end{array} \quad \begin{array}{r} 10 \\ - 3 \\ \hline 7 \end{array}$$

Care must be taken to insure values are within range.

~~$$\begin{array}{r} 00110011 \\ 01111110 \\ \hline 10110001 \end{array} \quad \begin{array}{r} 51 \\ + 126 \\ \hline -79 \end{array}$$~~

Logical Data Representation

- Bitwise
 - Each bit in a register can represent a logical true or false
 - Interpret each bit $0 \Rightarrow \text{False}$ $1 \Rightarrow \text{True}$
 - Terminology: $0 \Rightarrow$ say bit is “cleared”
 $1 \Rightarrow$ say bit is “set”
- Register
 - If the result of any numeric or logical operation in the microcontroller is zero, a bit in a special internal register is set.
 - Special register is called the STATUS register
 - Bit is called the zero bit (abbreviated “z”) (bit # 2 in register)
 - 0 result \Rightarrow $\text{STATUS},z = 1$ ($\text{STATUS},2 = 1$)

Logical Operations

- The microcontroller can perform all standard logical operations
 - AND, OR, XOR (Exclusive OR), NOT (called “complement”)
- Operations are preformed bitwise

$$(\text{“W” register}) = (\text{“W” register}) \bullet (\text{“Temp” register})$$

W	1	1	0	0	1	1	1	0
	&	&	&	&	&	&	&	&
Temp	0	1	0	0	1	0	1	1
	⇓	⇓				⇓	⇓	
W	0	1	0	0	1	0	1	0

STATUS_z = 0

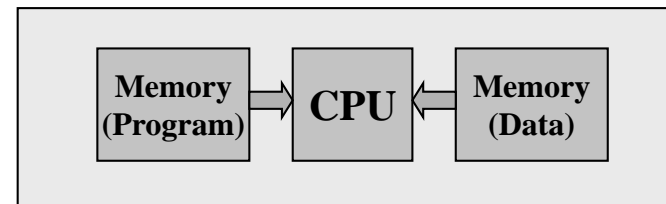
Aspects of Embedded Programming

There are several aspects to programming an embedded MicroChip microcontroller which makes it different from other programming.

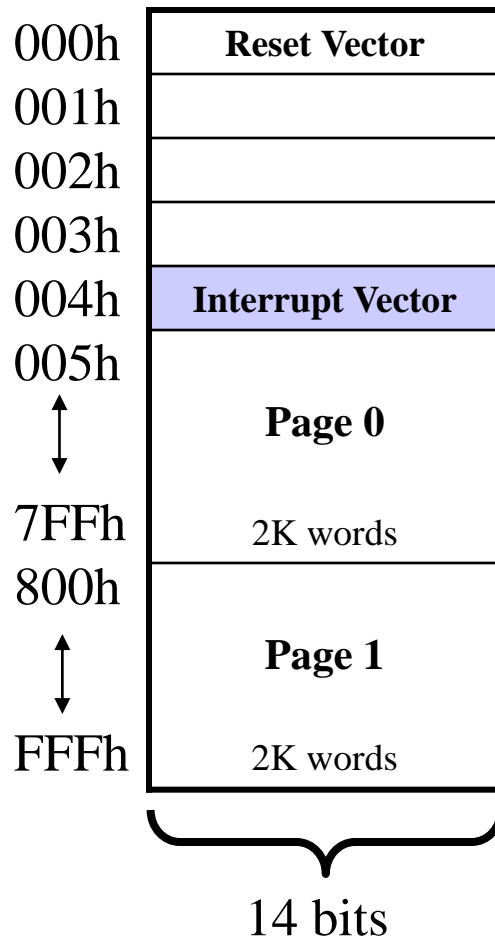
- Program and Data are in separate memory locations
- Input / Output pins have multiple uses which can be changed within a program (by manipulating special function registers)
- Limited memory addressing
 - Program memory is split into pages
 - Data memory is split into banks
- Data memory is memory mapped so the same instructions used for internal memory are used for input / output
- Program execution cannot halt – the processor must always be doing something
- Use of interrupts

MicroChip PIC16F74

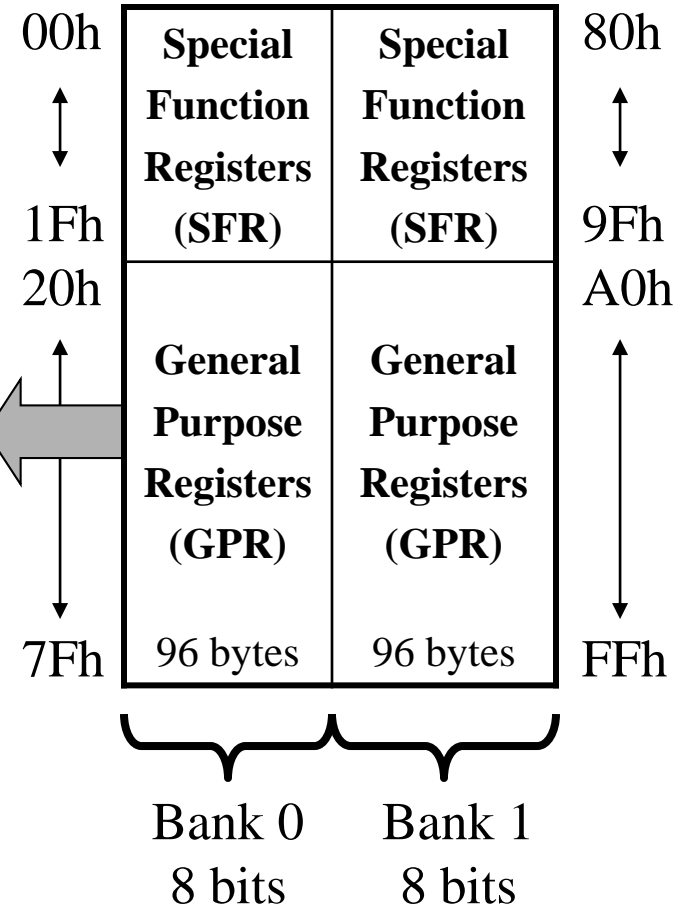
Harvard Architecture



Program Memory



Data Memory



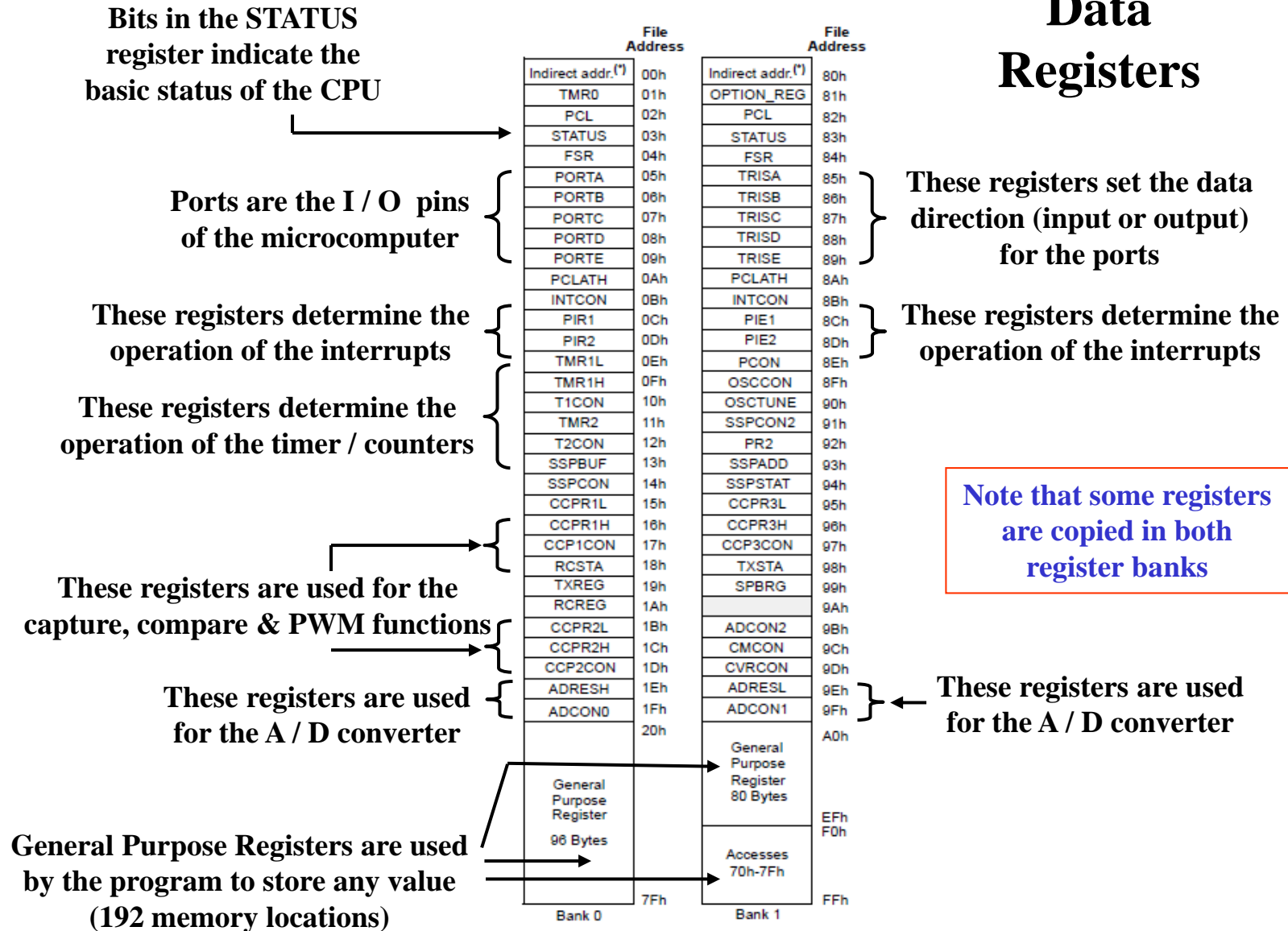
Program Memory Use

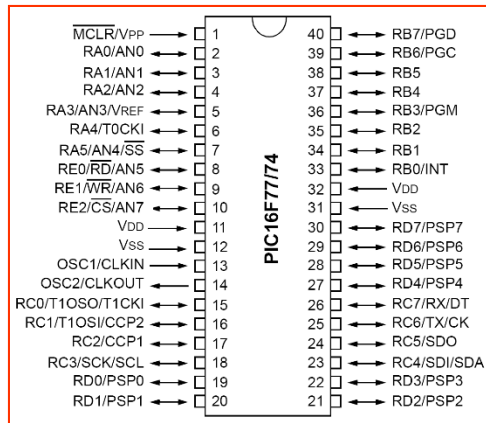
- Holds the program that runs the microcomputer.
- Typically changed only if the program does not work or to upgrade the program.
- You would typically program memory locations sequentially.
- When the device resets or powers up, execution starts at program memory location 000h
- Requires a programmer to program (a device that connects to the PC serial port into which you insert the chip) - - you typically say that you “burn the chip”
- Flash memory has to be erased electronically before being reprogrammed. This is done automatically.
- The EPROM part (PIC16C74 device in the lab with a window in the top) has to be erased under a UV light before being reprogrammed.

Data Memory Use

- Data memory is written to and read from under program control - - typically many times during program execution
- General Purpose Registers (GPR) hold your data. You typically assign a name to the GPR which indicates the data stored there (for better program readability).
- Data stored in the Special Function Registers (SFR) determines how the microcomputer behaves. Typically the first thing that a program does is write the proper data to the SFR - - called “initialization”
- When the device is reset or powered up, all data memory is set to some initial default value
 - GPR is set equal to all zeroes
 - Different SFR have different initial values
 - It is poor programming practice to rely on the initial values of data memory. You typically set it to the proper value even if this is the default.
- There are 2 additional banks of data memory used for in-circuit programming of the Flash memory. We will not use these.

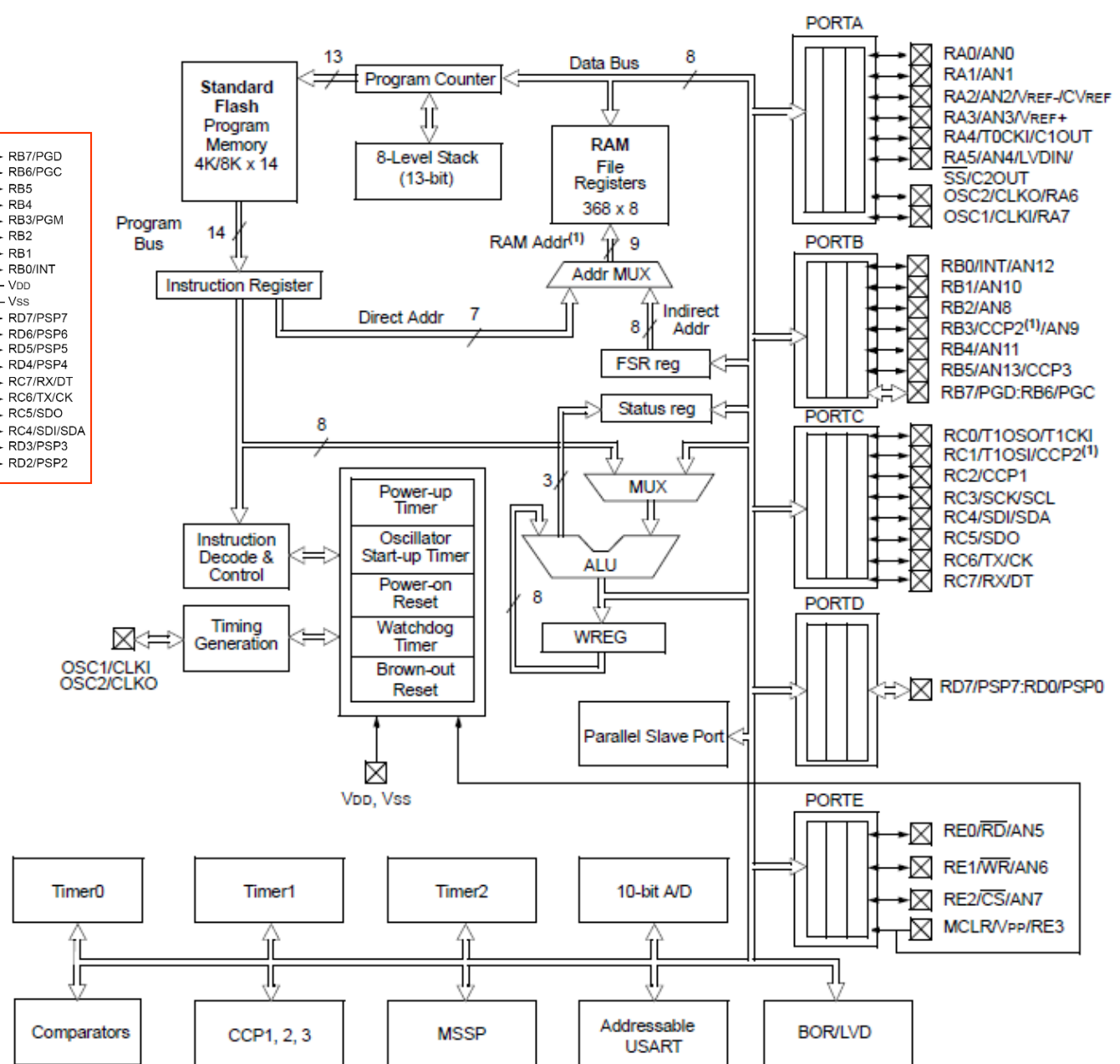
Data Registers





PIC16F747 Schematic

Note that the port pins can have different functions. R## indicates digital I/O (such as RB1). AN# indicates analog input to an A/D (such as AN2).



Determine the Function of the Digital Port Pins

- For general purpose digital I/O on Ports B, C & D you have to set the data direction - - input or output
- Set data direction in TRIS registers (tri-state)
 - Writing a 1 in a TRIS bit location indicates that the corresponding port pin is an input
 - Writing a 0 in a TRIS bit location indicates that the corresponding port pin is an output

PORTB	⇔	TRISB
PORTC	⇔	TRISC
PORTD	⇔	TRISD

Determine the Function of the Analog / Digital Ports

- For Ports A & E you have to determine if the pins are analog or digital and if digital, set the data direction - - input or output (analog is only input)
- Determine Analog or Digital using the register

ADCON1

- If Analog, determine the operation of the A/D using register

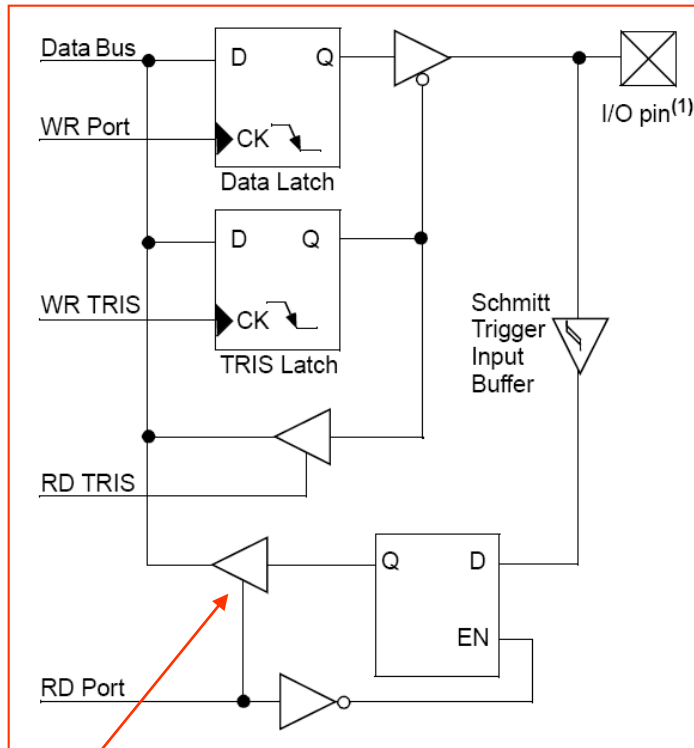
ADCON0

- If Digital, set the data direction in TRIS registers

PORTA	⇔	TRISA
PORTE	⇔	TRISE

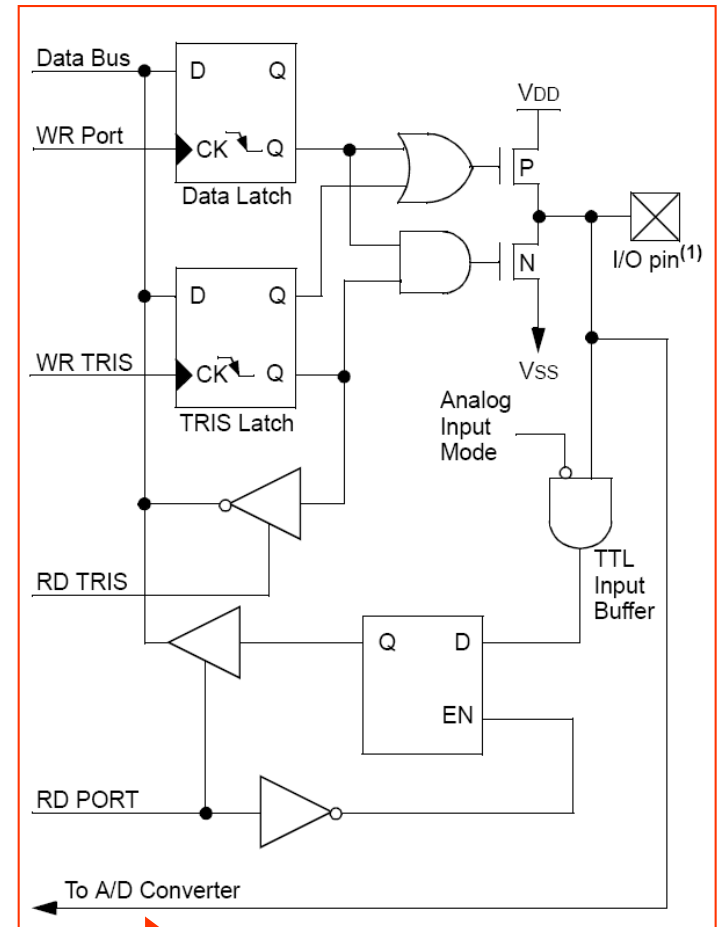
Port Block Diagrams

Port D



This is a “tri-state” buffer. Digital signals pass from input to output when the control is enabled. Otherwise, the output “floats” (like an open switch).

Port A



Note: if input is analog.

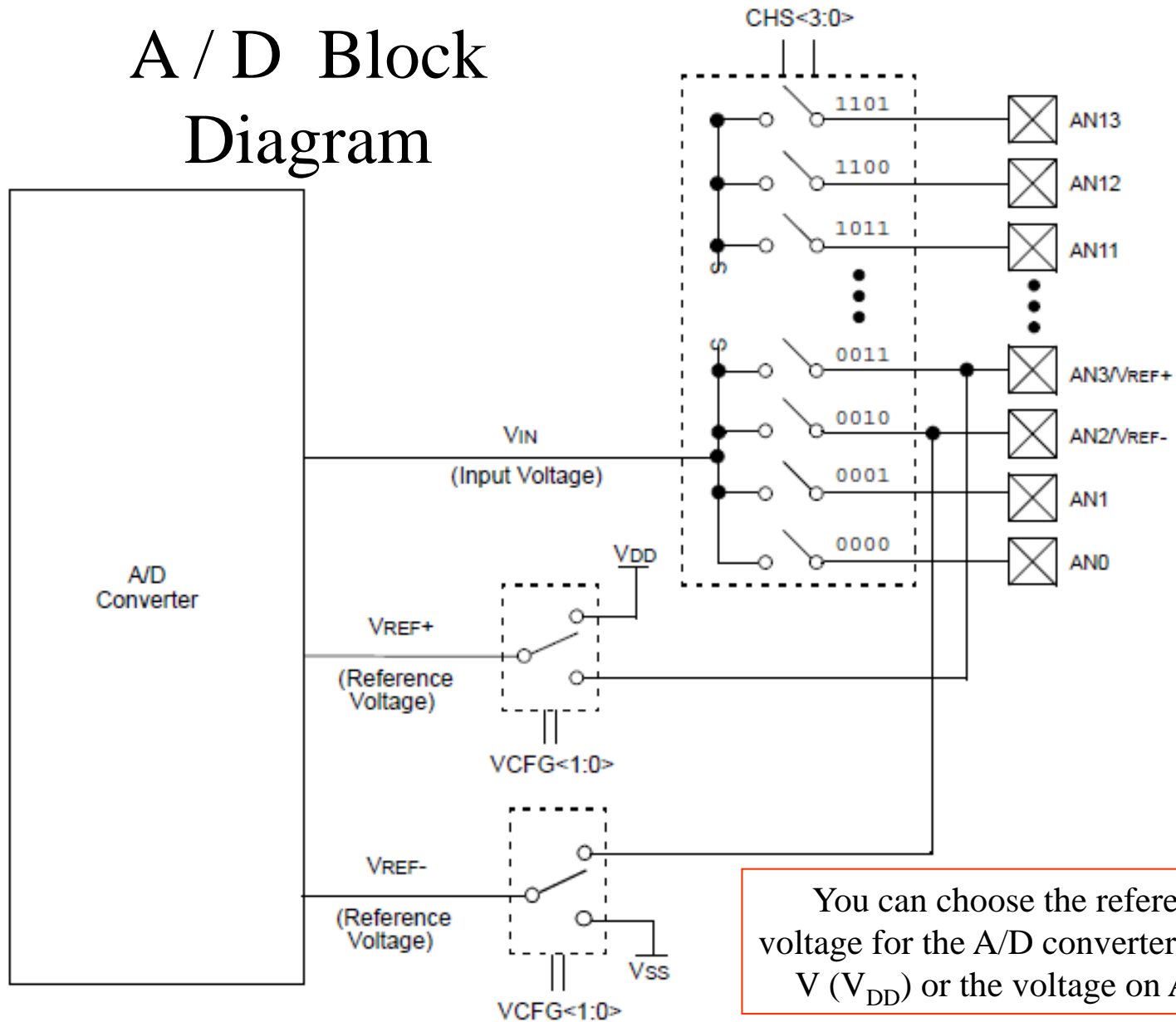
Digital Data Direction

TRISA	-	-	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0
-------	---	---	--------	--------	--------	--------	--------	--------

TRISB								
TRISC	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0
TRISD								

- In TRISA, TRISB, TRISC & TRISD registers, all bits set the digital data direction.
- A “1” in a bit means that that bit is an input.
- A “0” in a bit means that that bit is an output.

A / D Block Diagram



You can choose the reference voltage for the A/D converter to be 5 V (V_{DD}) or the voltage on AN3.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7						bit 0	

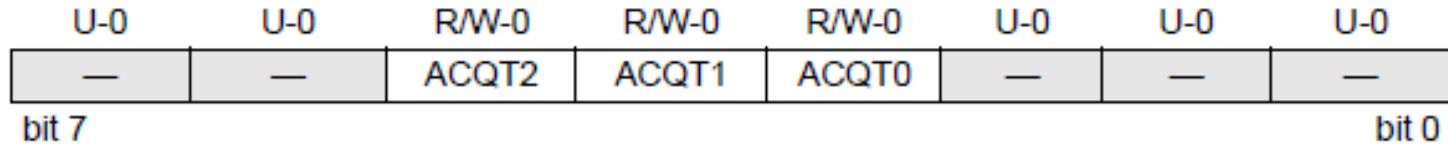
- bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six Least Significant bits of ADRESL are read as '0'.
- bit 6 **ADCS2:** A/D Clock Divide by 2 Select bit
1 = A/D clock source is divided by two when system clock is used
0 = Disabled
- bit 5 **VCFG1:** Voltage Reference Configuration bit 1
0 = VREF- is connected to VSS
1 = VREF- is connected to external VREF- (RA2)
- bit 4 **VCFG0:** Voltage Reference Configuration bit 0
0 = VREF+ is connected to VDD
1 = VREF+ is connected to external VREF+ (RA3)
- bit 3-0 **PCFG<3:0>:** A/D Port Configuration bits

ADCON1 Register

	AN13	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	D	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D	D

Legend: A = Analog input, D = Digital I/O

ADCON2 Register



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

bit 5-3 **ACQT<2:0>:** A/D Acquisition Time Select bits

000 = 0⁽¹⁾

001 = 2 TAD

010 = 4 TAD

011 = 6 TAD

100 = 8 TAD

101 = 12TAD

110 = 16 TAD

111 = 20 TAD

Note 1: If the A/D clock source is selected as RC, a time of T_{CY} is added before the A/D clock starts. This allows the `SLEEP` instruction to be executed.

bit 2-0 **Unimplemented:** Read as '0'

ADCON0 Register

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	ADON
bit 7							bit 0
bit 7-6	ADCS1:ADCS0: A/D Conversion Clock Select bits <u>If ADCS2 = 0:</u> 000 = Fosc/2 001 = Fosc/8 010 = Fosc/32 011 = FRC (clock derived from an RC oscillation) <u>If ADCS2 = 1:</u> 00 = Fosc/4 01 = Fosc/16 10 = Fosc/64 11 = FRC (clock derived from an RC oscillation)						
bit 5-3	CHS<2:0>: Analog Channel Select bits 0000 = Channel 00 (AN0) 0001 = Channel 01 (AN1) 0010 = Channel 02 (AN2) 0011 = Channel 03 (AN3) 0100 = Channel 04 (AN4) 0101 = Channel 05 (AN5) ⁽¹⁾ 0110 = Channel 06 (AN6) ⁽¹⁾ 0111 = Channel 07 (AN7) ⁽¹⁾ 1000 = Channel 08 (AN8) 1001 = Channel 09 (AN9) 1010 = Channel 10 (AN10) 1011 = Channel 11 (AN11) 1100 = Channel 12 (AN12) 1101 = Channel 13 (AN13) 111x = Unused Note 1: Selecting AN5 through AN7 on the 28-pin product variant (PIC16F737 and PIC16F767) will result in a full-scale conversion as unimplemented channels are connected to VDD.						
bit 2	GO/DONE: A/D Conversion Status bit 1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle. This bit is automatically cleared by hardware when the A/D conversion has completed. 0 = A/D conversion completed/not in progress						
bit 1	CHS<3>: Analog Channel Select bit (see bit 5-3 for bit settings)						
bit 0	ADON: A/D Conversion Status bit 1 = A/D converter module is operating 0 = A/D converter is shut-off and consumes no operating current						

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	— ⁽¹⁾	TRISE2	TRISE1	TRISE0
bit 7							bit 0

TRISE Register

bit 7 **Parallel Slave Port Status/Control bits:**

IBF: Input Buffer Full Status bit

- 1 = A word has been received and is waiting to be read by the CPU
- 0 = No word has been received

bit 6 **OBF:** Output Buffer Full Status bit

- 1 = The output buffer still holds a previously written word
- 0 = The output buffer has been read

bit 5 **IBOV:** Input Buffer Overflow Detect bit (in Microprocessor mode)

- 1 = A write occurred when a previously input word has not been read (must be cleared in software)
- 0 = No overflow occurred

bit 4 **PSPMODE:** Parallel Slave Port Mode Select bit

- 1 = Parallel Slave Port mode
- 0 = General Purpose I/O mode

Be
careful

bit 3 **Unimplemented:** Read as '1'⁽¹⁾

Note 1: RE3 is an input only. The state of the TRISE3 bit has no effect and will always read '1'.

bit 2 **PORTC Data Direction bits:**

TRISE2: Direction Control bit for pin RE2/ $\overline{\text{CS}}$ /AN7

- 1 = Input
- 0 = Output

bit 1 **TRISE1:** Direction Control bit for pin RE1/ $\overline{\text{WR}}$ /AN6

- 1 = Input
- 0 = Output

bit 0 **TRISE0:** Direction Control bit for pin RE0/ $\overline{\text{RD}}$ /AN5

- 1 = Input
- 0 = Output

In TRISE, for general purpose I/O bits 3 – 7 should be set to 0. Bits 0 – 2 set the digital data direction.

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, **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/borrow bit (**ADDWF**, **ADDLW**, **SUBLW**, **SUBWF** instructions)
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

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.

STATUS Register

Limited Memory Addressing

- Program memory is split into pages.
 - In PIC16F74, Page 0 is 2K words holding approximately 2000 Assembly instructions
 - Care must be taken if an Assembly program goes over a page boundary.
 - If your Assembly programs exceed 2000 instructions for this course, you should really try to reduce your code.
 - C compiler takes care of paging
- Data memory is split into banks.
 - When you write to a register, you have to insure that you are pointing to the correct bank (the bank in which the register is)
 - Banks are set in the STATUS register with bits RP0 & RP1
 - Assembler does not check if you are pointing to the proper bank but gives you a warning if you try to access a register not in bank 0
 - C compiler does bank switching

Assembly Language Programming

- Assembly language programs a microcomputer at the lowest possible level
 - Each Assembly language instruction corresponds to one machine instruction
- Requires software called an “Assembler”
 - The Assembler converts the Assembly instruction into the binary code that is actually entered into the microcomputer’s program memory
- Microcomputer Assembly programs typically has 3 parts
 1. Directives to tell the assembler what to do
 2. Initialization code which sets up the registers to perform specific functions
 3. Code that is executed to perform a task

Assembler Directives

- Used to tell the Assembler what to do
- Not programmed onto the chip

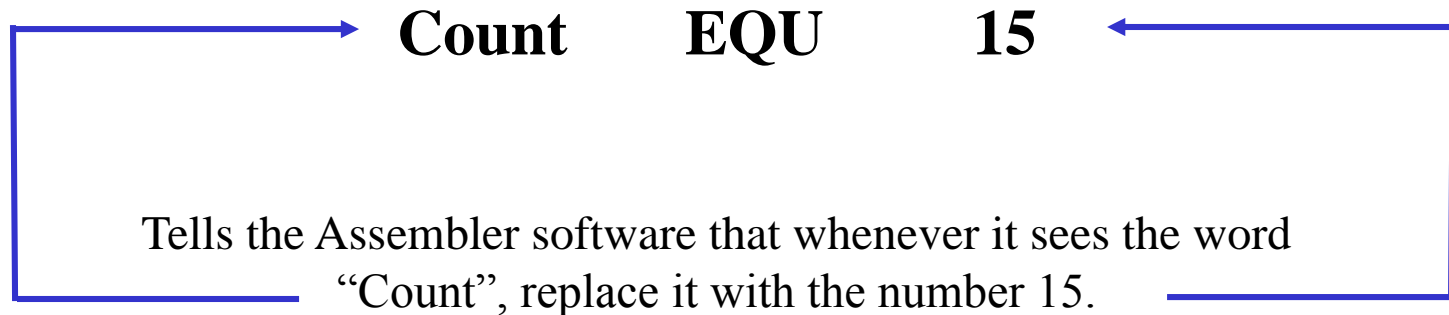
ORG ⇒ sets the next program location in program memory

EQU ⇒ defines a constant for the Assembler

END ⇒ defines the end of the program

_ _ CONFIG ⇒ determines the configuration fuses

Example of a Directive



The assembler is case sensitive, register names & instructions should be upper case (like **EQU** above).

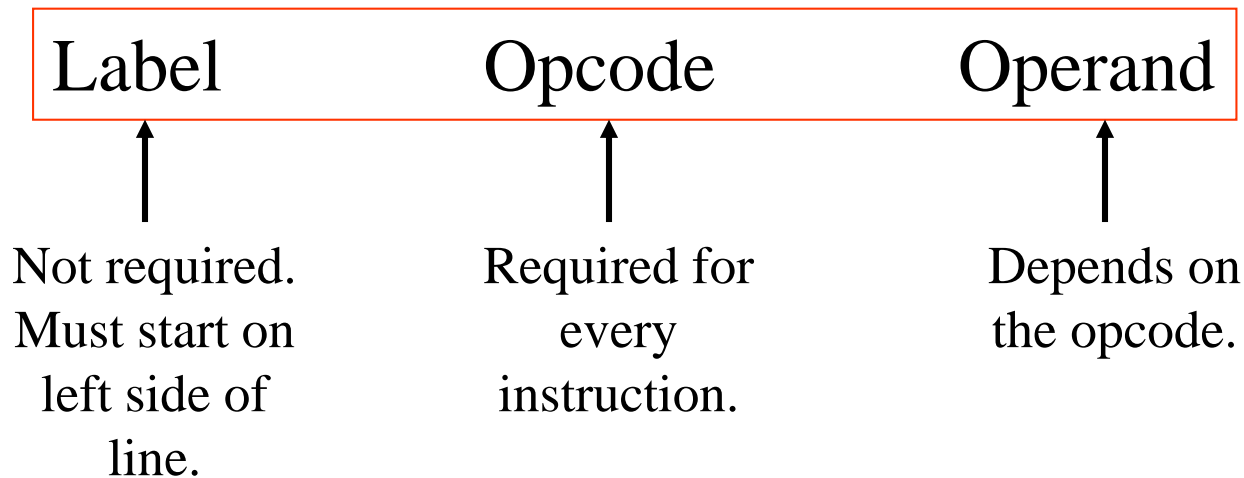
For easier readability, I recommend that constants you define have upper case & lower case letters.

Radix Specification

Type	Syntax	Example
Decimal	D' <digits>'	D' 100' 100
Hexadecimal	H' <hex_digits>' 0x<hex_digits>	H' 9f' 9fh 0x9f
Octal	O' <octal_digits>'	O' 777'
Binary	B' <binary_digits>'	B' 00111001' ←
ASCII	' <character>' A' <character>'	' C' A' C'

Hexadecimal	<hex_digits>h	02h or 0FDh ←
Decimal	digits	2 or 253 ←

Assembly Language Instruction



For readability, the label is usually placed on a separate line.

The Assembler ignores whitespace.

The label, opcode and operand can be separated by 1 or more spaces.

Assembly Instruction Set

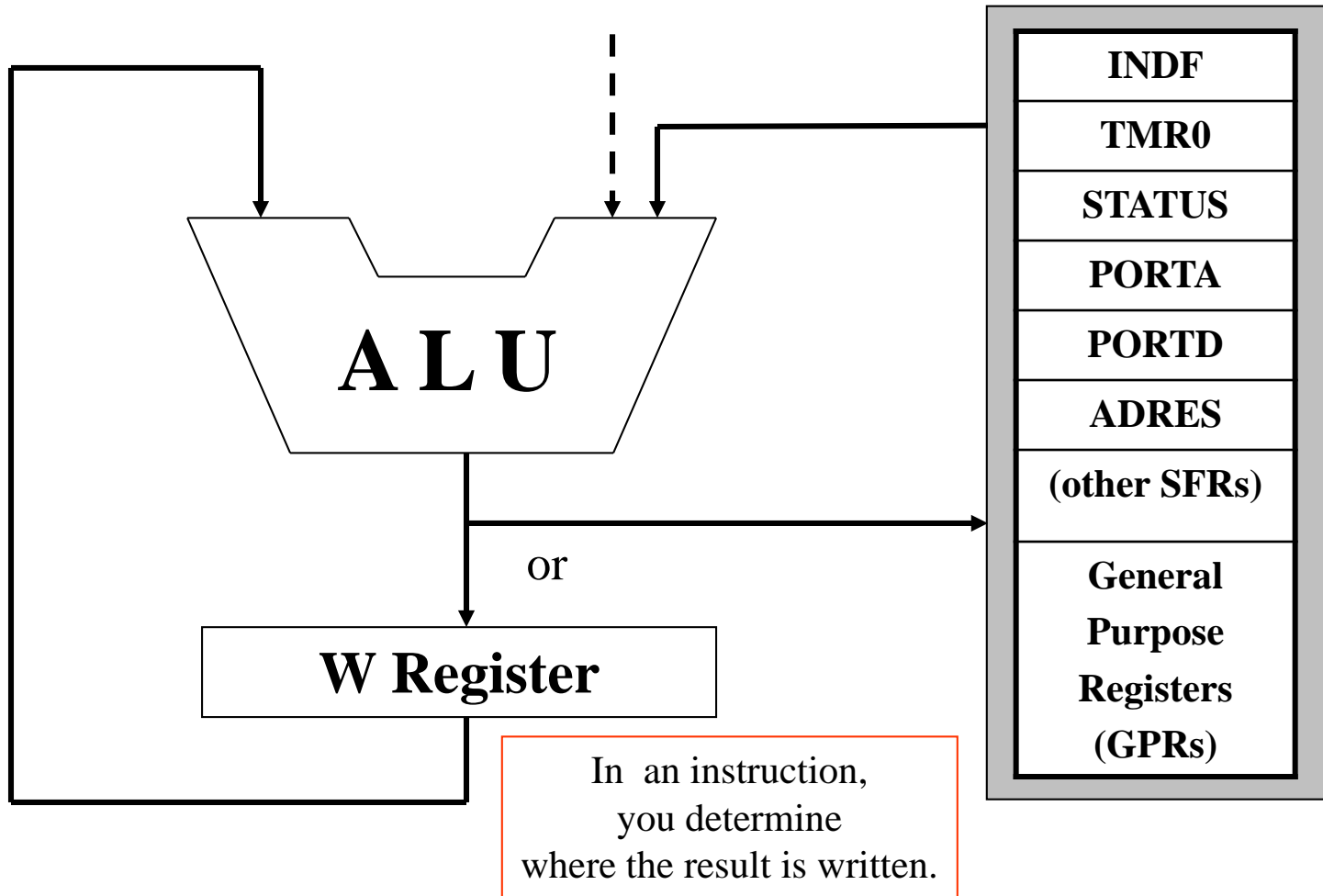
The total
number of
Assembly
instructions is
35, but they
fall into
certain
categories.

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	

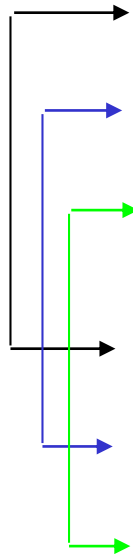
Arithmetic Logic Unit (ALU)

that is, a binary byte

literal



Logical Instructions

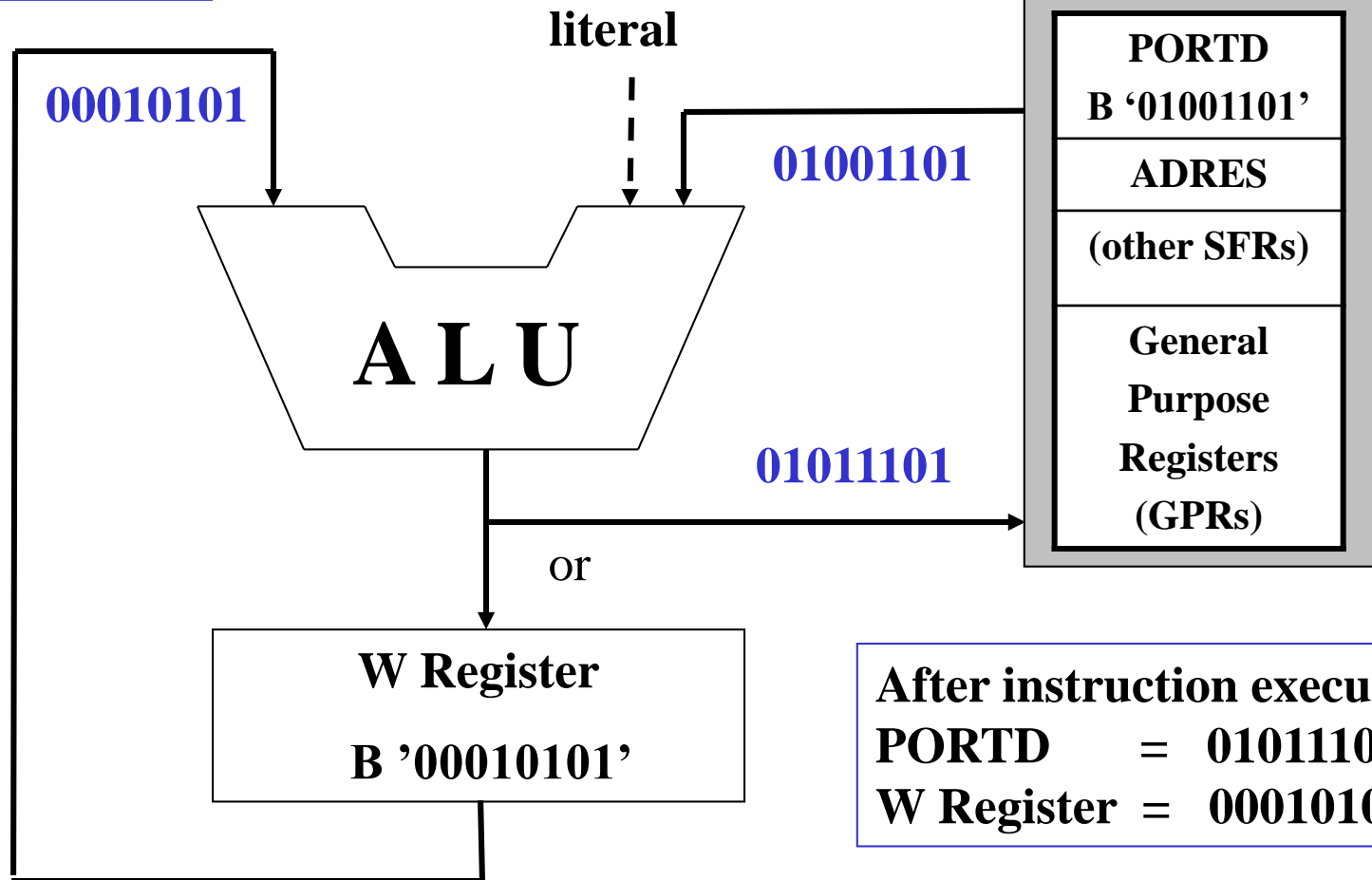
- 
- ANDWF – bitwise logical AND of W register and another register
 - IORWF – bitwise logical inclusive OR (normal OR) of W register and another register
 - XORWF – bitwise logical exclusive OR of W register and another register
 - ANDLW – bitwise logical AND of W register and a literal (a number included in the instruction)
 - IORLW – bitwise logical inclusive OR of W register and a literal
 - XORLW – bitwise logical exclusive OR of W register and a literal
 - COMF – bitwise logical not (complement) of a register

Label	ANDWF	PORTD,F
Label2	ANDLW	B'00011000'

Example:

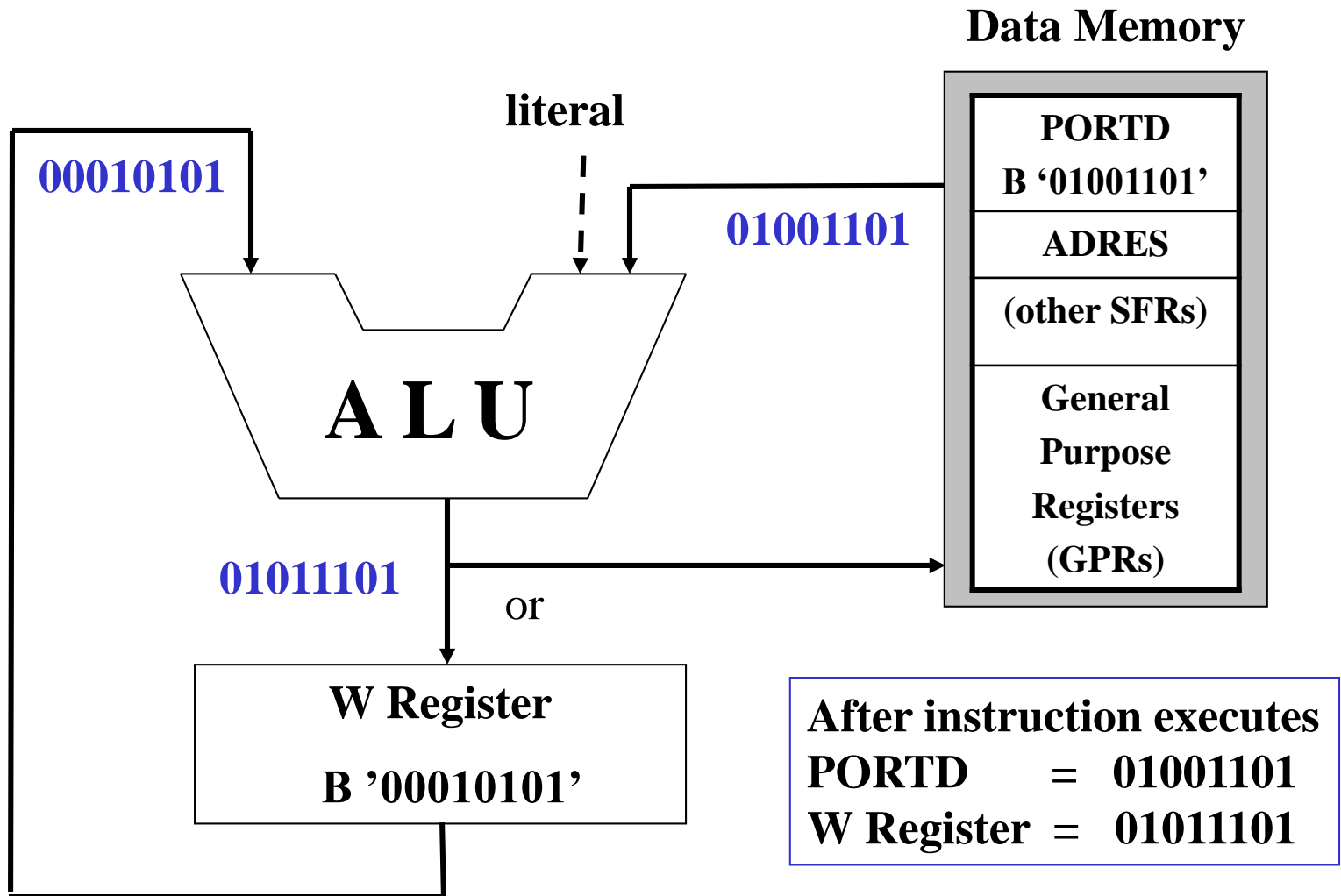
IORWF PORTD,F

No label



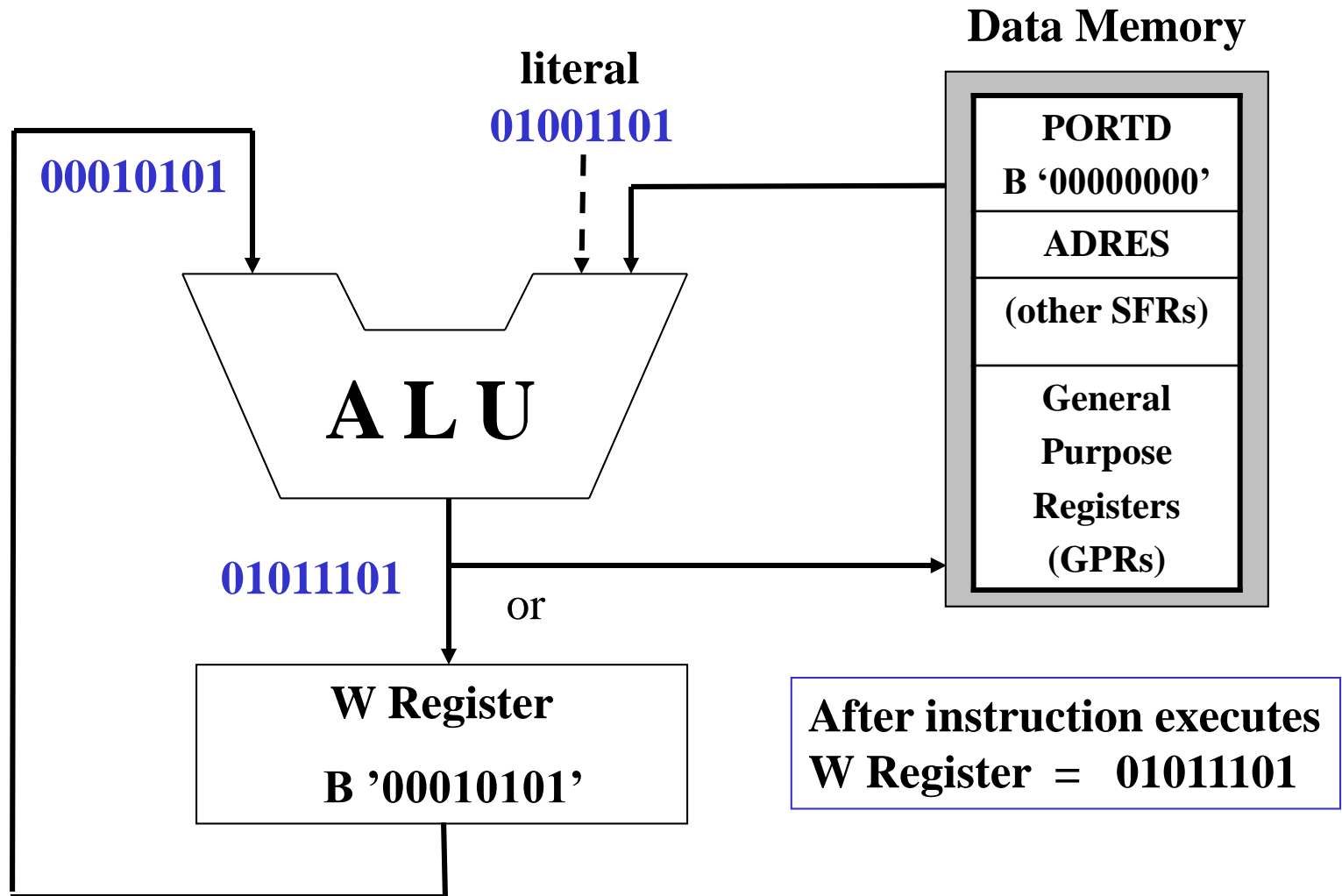
Example:

IORWF PORTD,W



Example:

IORLW B '01001101'



Encoding

**Program Memory
contains binary
digits.**

**The Assembler
converts an
Assembly language
instruction into
binary so instruction
can be written to
Program Memory.**

**Assembler \Rightarrow
each instruction has
one unique binary
representation.**

IORLW

Inclusive OR Literal with W

Syntax: `[label] IORLW k`

Operands: $0 \leq k \leq 255$

Operation: $(W).OR. k \rightarrow W$

Status Affected: Z

Encoding:

11	1000	kkkk	kkkk
----	------	------	------

Description: The contents of the W register is OR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

IORWF

Inclusive OR W with f

Syntax: `[label] IORWF f,d`

Operands: $0 \leq f \leq 127$
 $d \in [0,1]$

Operation: $(W).OR. (f) \rightarrow \text{destination}$

Status Affected: \bar{Z}

Encoding:

00	0100	dfff	ffff
----	------	------	------

Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Arithmetic Instructions

- • ADDWF – Add the W register to another register
- • ADDLW – Add a literal (a constant) the W register
- • SUBWF – Subtract the W register from another register
 - 2's complement subtract
- • SUBLW – Subtract the W register from a literal
 - 2's complement subtract

Increment & Decrement

- INCF – Increment a register
- DECF – Decrement a register
- INCFZ – Increment a register & skip the next instruction if the result is zero
- DECFZ – Decrement a register & skip the next instruction if the result is zero

DECFZ Instruction

Label DECFZ Count,F

Count	0	0	0	0	0	0	1
-------	---	---	---	---	---	---	---

Count	0	0	0	0	0	0	0
-------	---	---	---	---	---	---	---

next instruction after DECFZ instruction is **NOT** executed

Count	0	0	0	0	0	1	0	0
-------	---	---	---	---	---	---	---	---

Count	0	0	0	0	0	0	1	1
-------	---	---	---	---	---	---	---	---

next instruction after DECFZ instruction is executed

Move Instructions

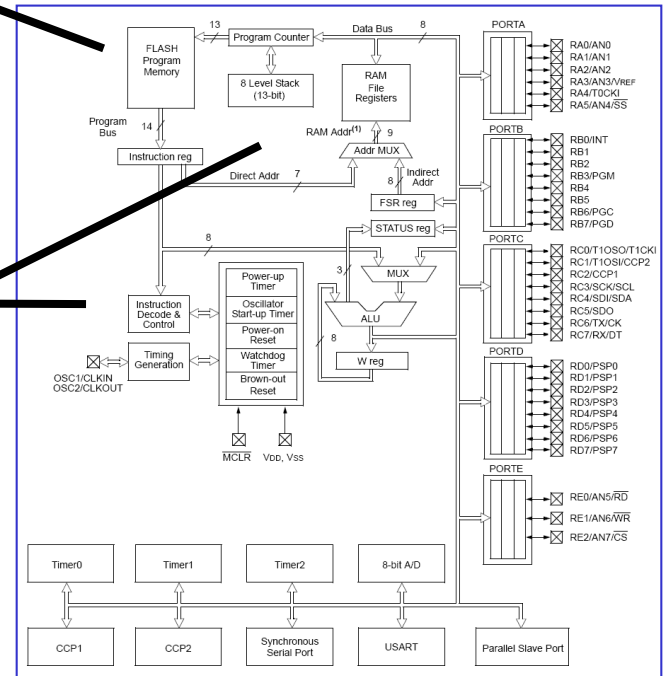
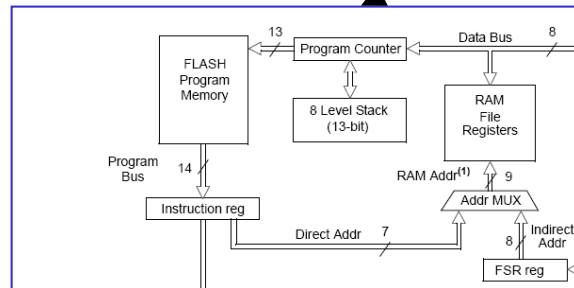
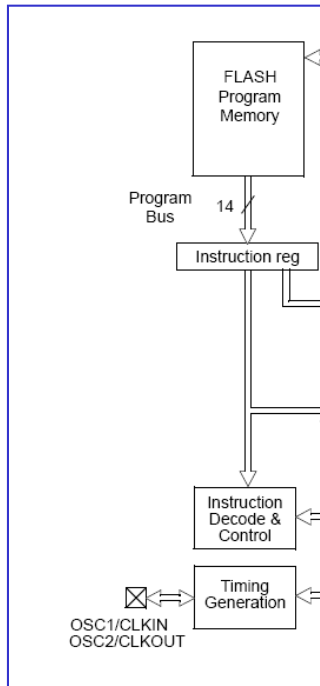
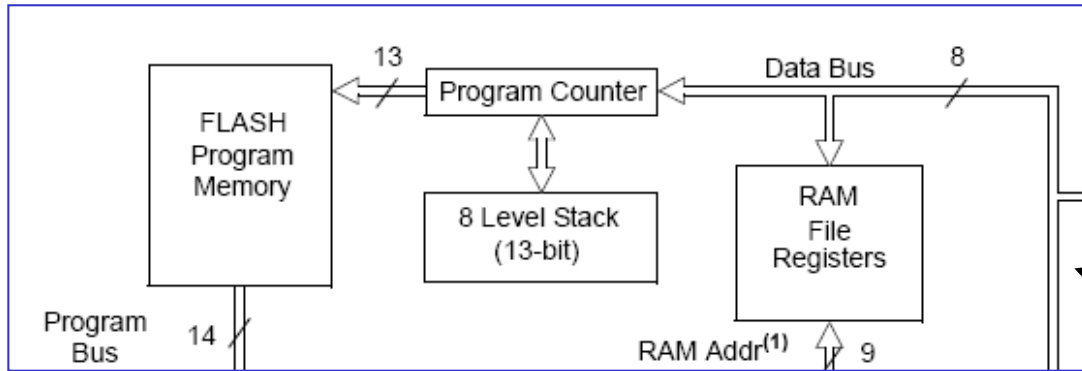
- MOVWF – Move the W register to another register
- MOVF – Move a register to the W register (or itself)
- MOVLW – Move a literal to the W register
- RLF – Rotate the bits of a register left through the carry bit
- RRF – Rotate the bits of a register right through the carry bit
- SWAPF – Swap (exchange) the nibbles (4 bits) of a register
- NOP – Do nothing (except waste a clock cycle)
- SLEEP – Put the microcomputer into low power mode (several things can “wake up” the microcomputer)

Bit Instructions

- BSF – Set (make = 1) a bit in a register
- BCF – Clear (make = 0) a bit in a register
- CLRF – Clear all the bits in a register
- CLRW – Clear all the bits in the W register

- BTFSC – Check a bit in a register & skip the next instruction if the bit is clear (zero)
- BTFSS – Check a bit in a register & skip the next instruction if the bit is set (1)

CPU Program Counter



Jumps & Subroutine Processing

- GOTO – Set **Program Counter** to new address (that instruction will be executed next)
- CALL – Set **Program Counter** to new address (that instruction will be executed next)
 - Put address after the CALL instruction into the **Stack** (called “Pushing the Stack”)
- RETURN – Set **Program Counter** to address at the top of the **Stack** (that instruction will be executed next)
 - Move all next instructions (if any) on the **Stack** up one level (called “Popping the Stack”)
- RETLW –RETURN & put a byte into the W Register
- RETFIE – RETURN from an interrupt subroutine (does return & re-enables the interrupt bit)

Program Memory

000h	<i>Reset Vector</i>		
001h	movlw	0FFh	
002h	movwf	PORTD	
003h			
004h	<i>Interrupt Vector</i>		
005h			
006h	goto	Loop	
007h			
008h	call	Timer	
009h			
00Ah			
00Bh	Loop	addlw	06h
00Ch			
00Dh	Timer	andwf	PORTA,F
00Eh			
00Fh		return	
010h			

Program Counter 13 bits

Start at 0 0 0 hex

Increments after executing instruction

Program Counter

**Use
Assembler
ORG
directive**

Stack 8 deep 13 bits wide

Program Memory

000h	<i>Reset Vector</i>
001h	
002h	
003h	
004h	<i>Interrupt Vector</i>
005h	
006h	goto Loop
007h	
008h	call Timer
009h	
00Ah	
00Bh	Loop addlw 06h
00Ch	
00Dh	Timer andwf PORTA,F
00Eh	
00Fh	return
010h	

Program Counter & Stack

Loop	equ	00Bh
Timer	equ	00Fh

00B

Program Counter 13 bits

Stack 8 deep 13 bits wide

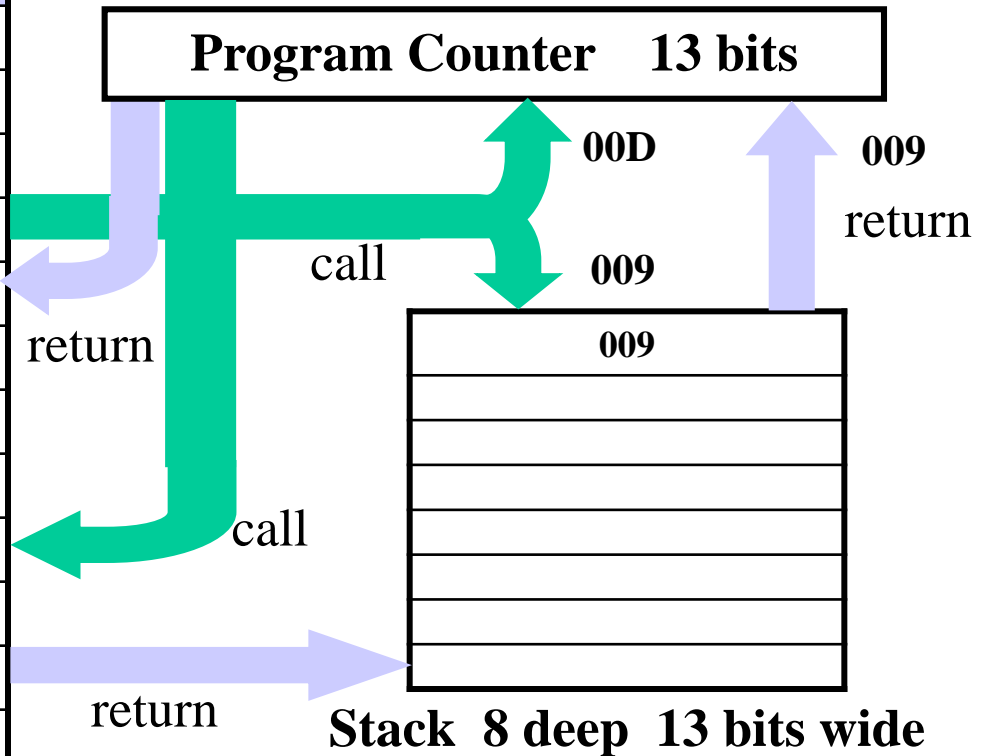
Program Memory

000h	<i>Reset Vector</i>
001h	
002h	
003h	
004h	<i>Interrupt Vector</i>
005h	
006h	goto Loop
007h	
008h	call Timer
009h	
00Ah	
00Bh	Loop addlw 06h
00Ch	
00Dh	Timer andwf PORTA,F
00Eh	
00Fh	return
010h	

Subroutine Processing

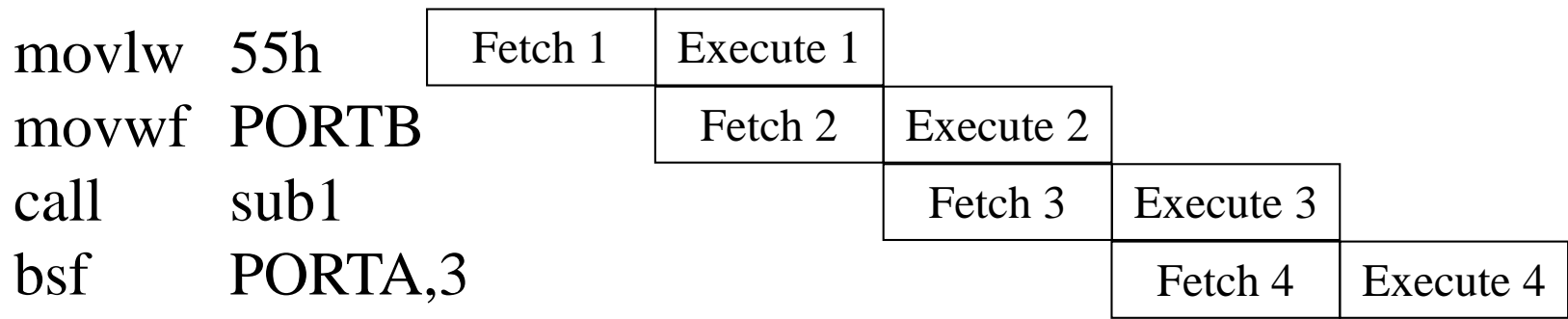
```

Loop    equ    00Bh
Timer    equ    00Fh
  
```



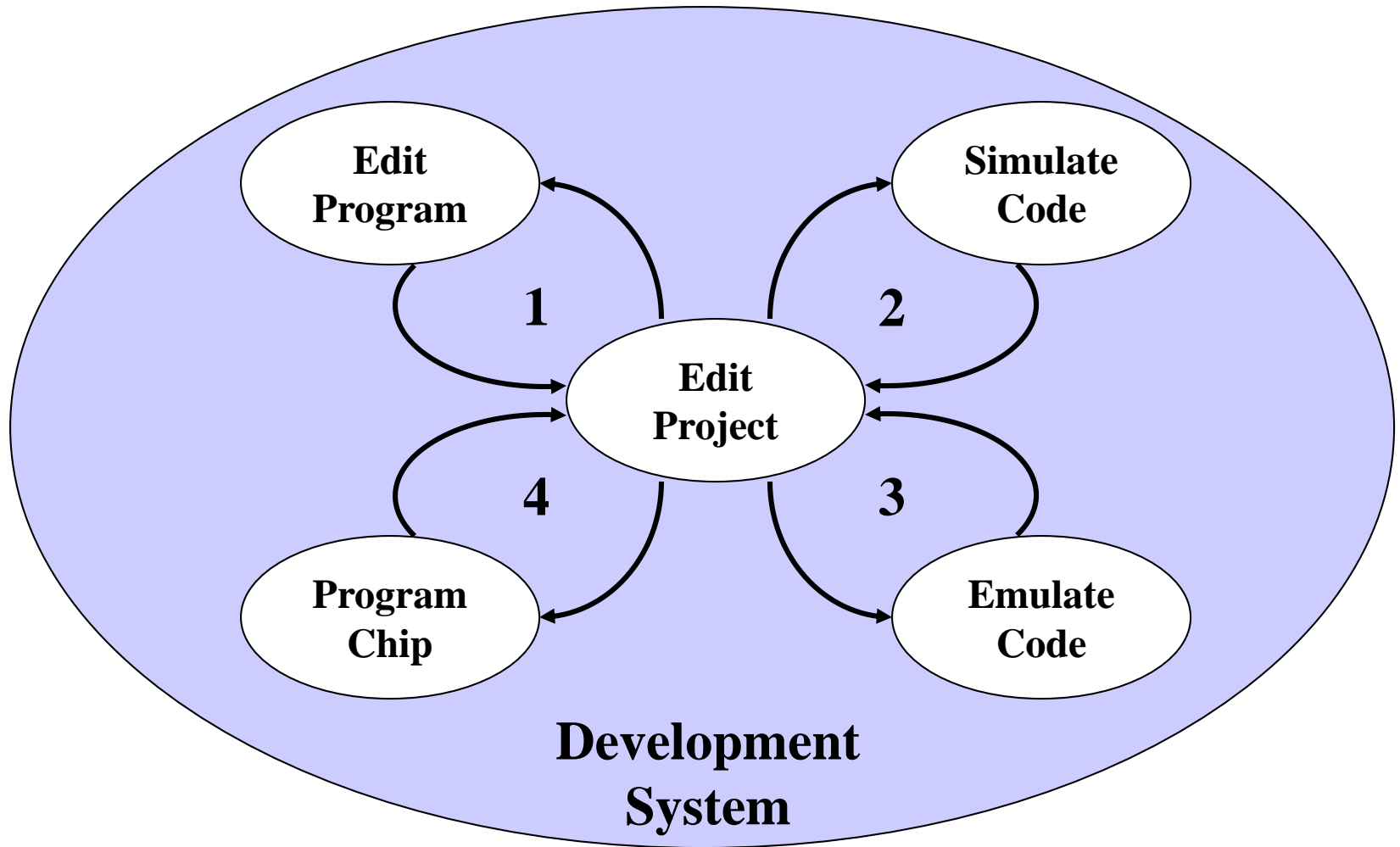
Pipelining

- Instructions are fetched and executed sequentially
- Pipelined architecture overlaps fetch and execution, making single cycle instructions possible

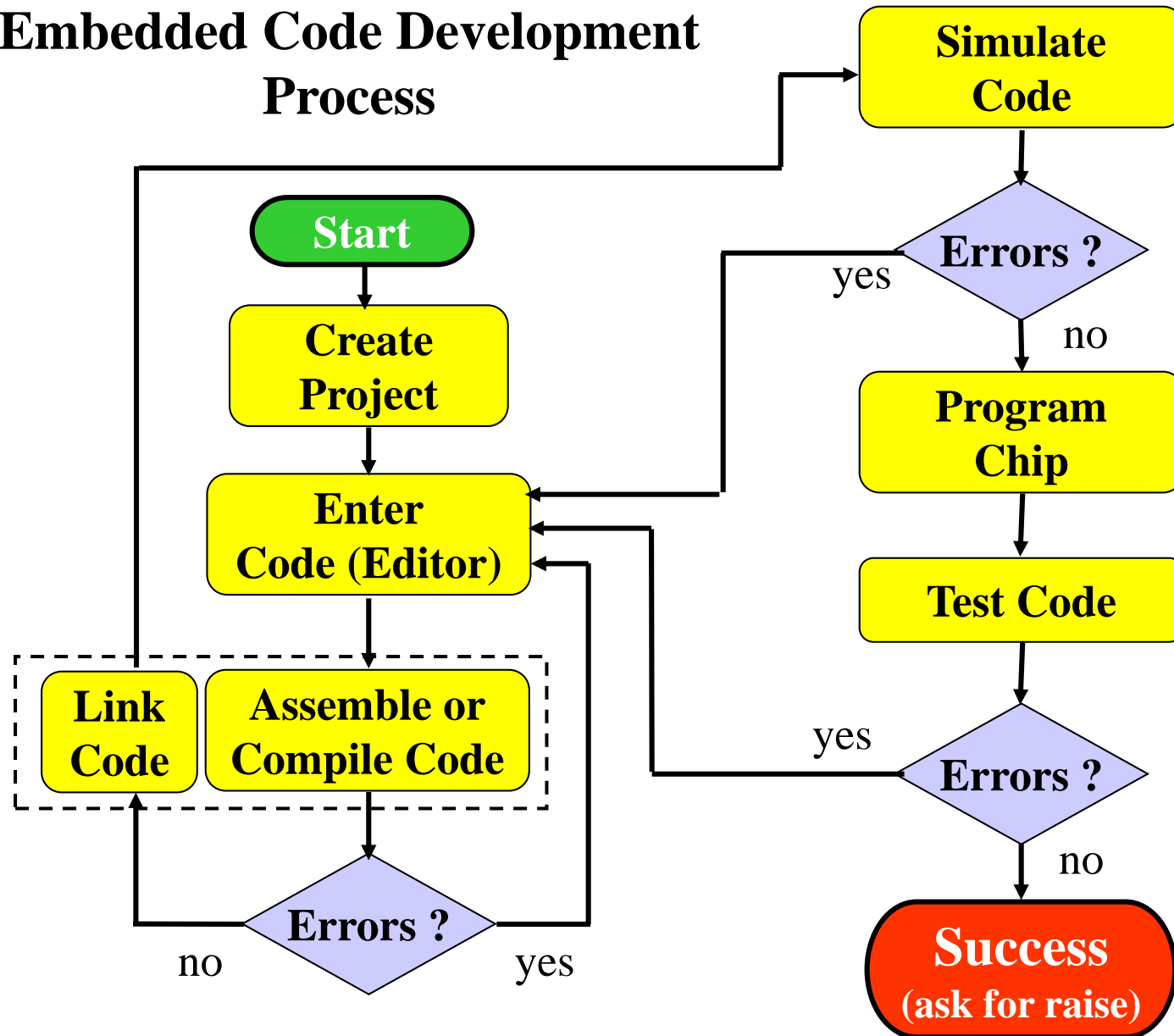


- Any program branch such as *goto* or *call* will take 2 cycles

Microcomputer Development System



Embedded Code Development Process



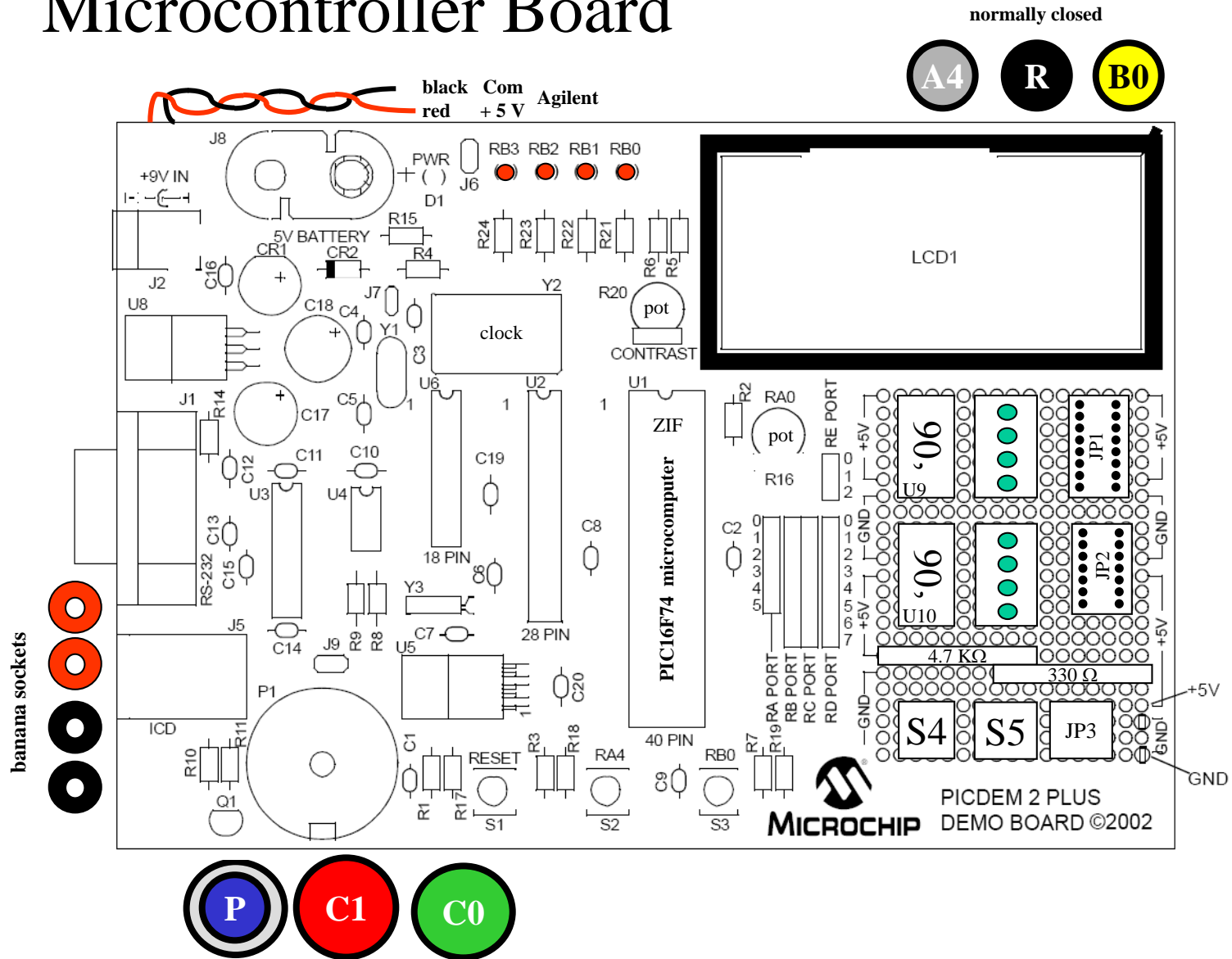
Debugging Code in a Code Simulator

- Break Points
 - Conditionally Stops Processing
- Trace Points
 - Collects Code Execution
- Stimulus
 - Simulate the Application of Signals

Microcomputer Exercise

- Learn Microcomputer Development System
 - Code Editor
 - Code Simulator & Debugger
 - Device Programmer
- Learn Assembly Programming with 3 simple programs
 - Counter – counts the number of button presses
 - ADConverter – reads an analog value on a potentiometer and outputs the value on a series of LEDs
 - Timer – long timer which increments the LEDs slowly (once per second)
- Learn Code Development Process by doing it 3 times

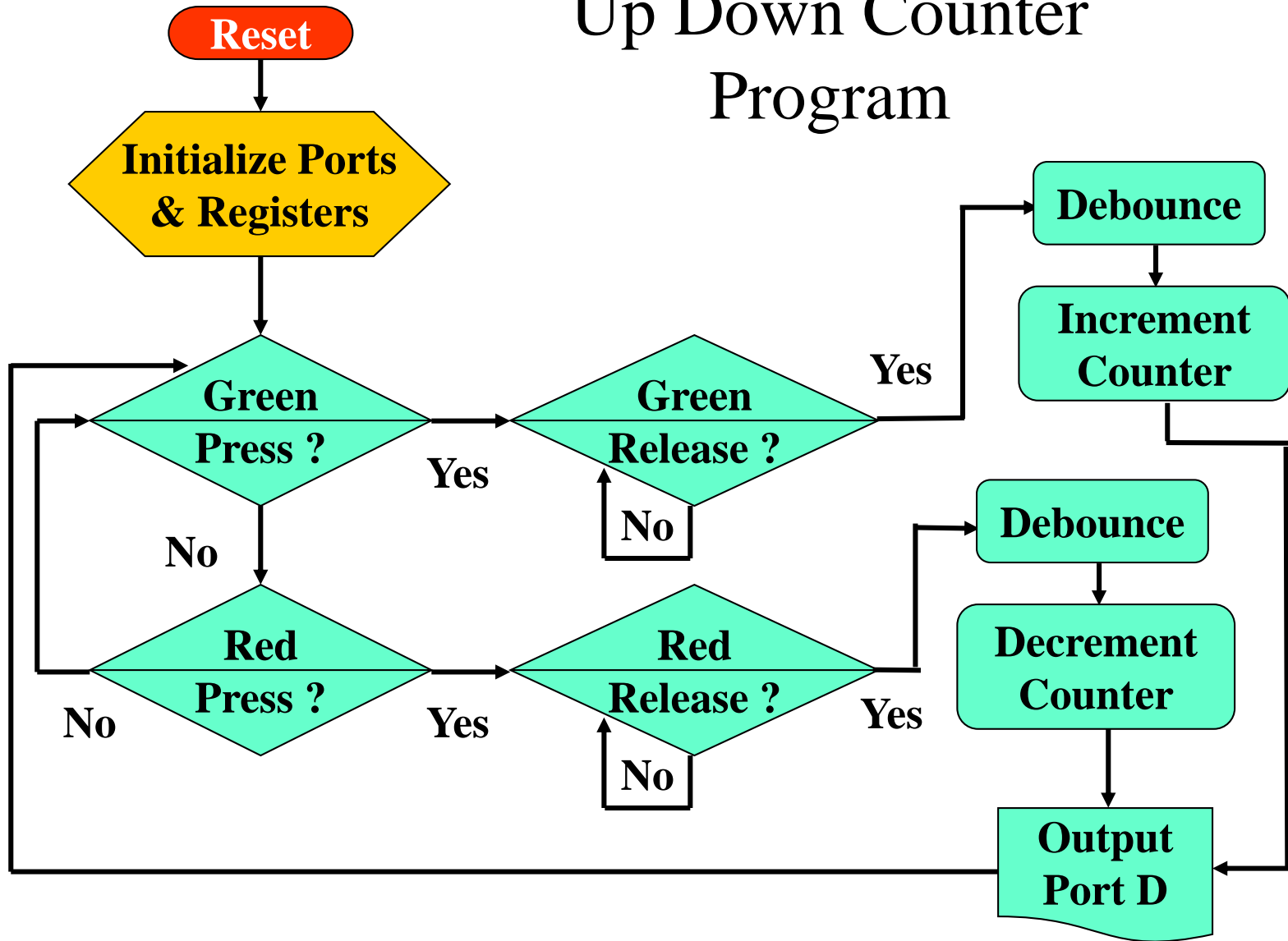
Microcontroller Board



PM3 Device Programmer



Up Down Counter Program



UpDown Counter Program - Initialization

```
clrf    PORTD    ; Clear Port D output latches
clrf    PORTC    ; Clear Port C output latches
bsf     STATUS,RP0 ; Set bit in STATUS register for bank 1
movlw   B'11111111' ; move hex value FF into W register
movwf   TRISC    ; Configure Port C as all inputs
clrf    TRISD    ; Configure Port D as all outputs
bcf     STATUS,RP0 ; Clear bit in STATUS register for bank 0
```

UpDown Counter Program - Counting

waitPress

btfsc	PORTC,0	; see if green button pressed
goto	GreenPress	; green button is pressed
btfsc	PORTC,1	; see if red button pressed
goto	RedPress	; red button is pressed
goto	waitPress	; keep checking

GreenPress

btfss	PORTC,0	; see if green button still pressed
goto	waitPress	; noise - keep checking

GreenRelease

btfsc	PORTC,0	; see if green button released
goto	GreenRelease	; no - keep waiting
call	SwitchDelay	; let switch debounce
goto	IncCount	; increment the counter

UpDown Counter Program - Continued

RedPress

```
btfss    PORTC,1    ; see if red button still pressed  
goto     waitPress ; noise - keep checking
```

RedRelease

```
btfsc    PORTC,1    ; see if red button released  
goto     RedRelease ; no - keep waiting  
call     SwitchDelay ; let switch debounce  
decf     Count,F    ; decrement count - store in register  
goto     outCount   ; output the count on the LEDs
```

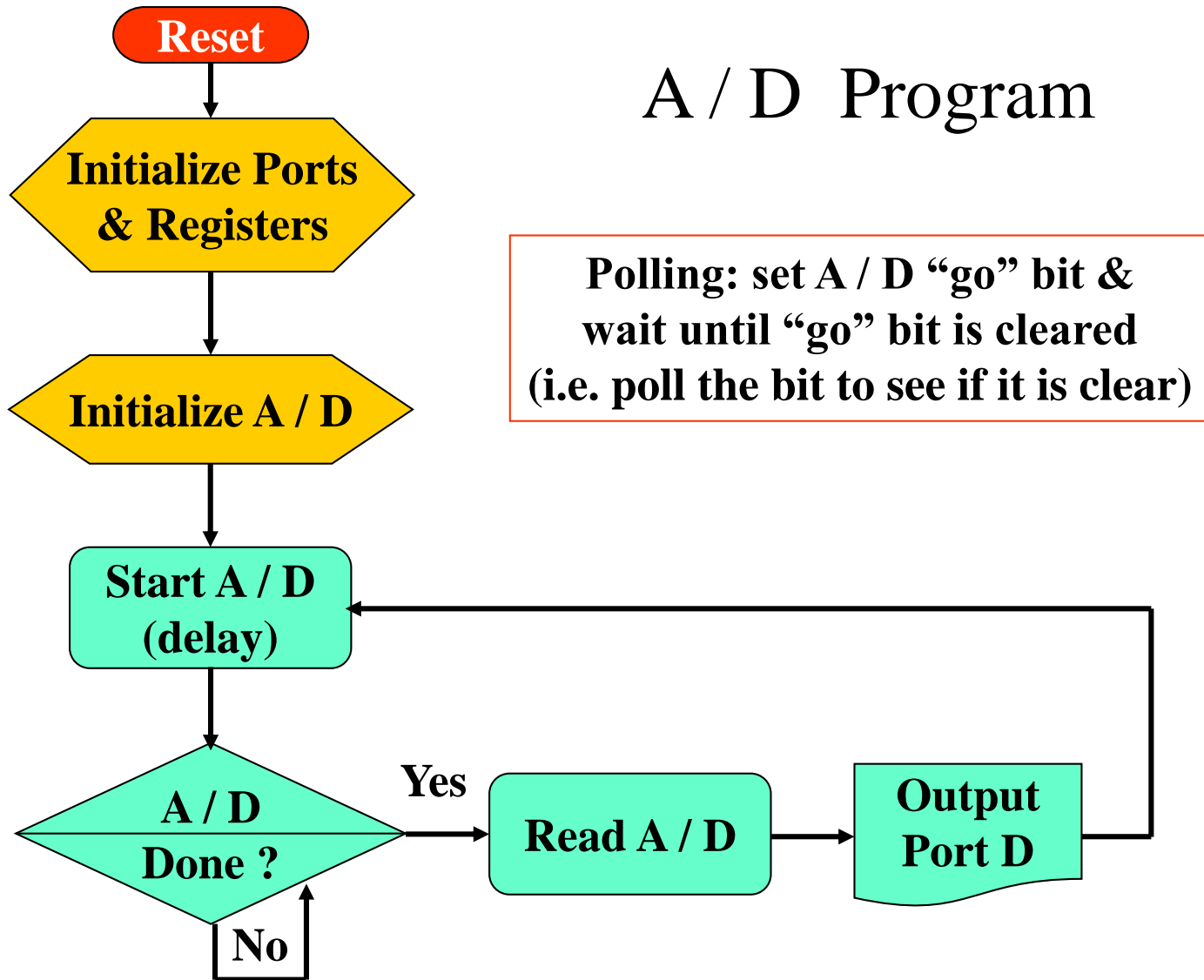
IncCount

```
incf     Count,F    ; increment count - store in register
```

OutCount

```
movf     Count,W    ; move the count to the W register  
movwf    PORTD      ; display count on port D  
goto     waitPress  ; wait for next button press
```


A / D Program



AtoDPolled Program - Initialization

initAD

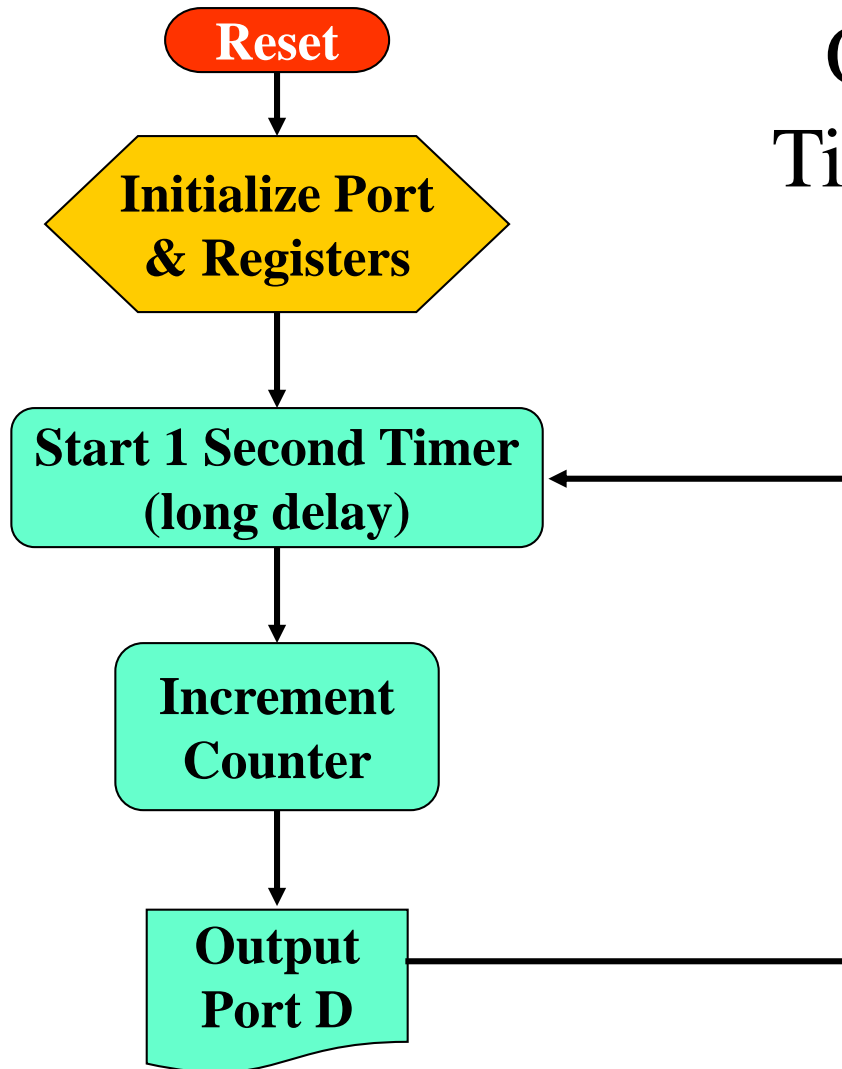
```
bsf      STATUS,RP0      ; select register bank 1
movlw    B'00000100'     ; RA0,RA1,RA3 analog inputs, all other digital
movwf    ADCON1          ; move to special function A/D register
bcf      STATUS,RP0      ; select register bank 0
movlw    B'01000001'     ; select 8 * oscillator, analog input 0, turn on
movwf    ADCON0          ; move to special function A/D register
return
```

AtoDPolled Program – Read AtoD

waitLoop

btfsc	ADCON0,GO	; check if A/D is finished
goto	waitLoop	; loop right here until A/D finished
btfsc	ADCON0,GO	; make sure A/D finished
goto	waitLoop	; A/D not finished, continue to wait
movf	ADRESH,W	; get A/D value
movwf	PORTD	; display on LEDs
bsf	ADCON0,GO	; restart A/D conversion
goto	waitLoop	; return to loop

One Second Timer Program



How to design a long counter

- Determine instruction cycle

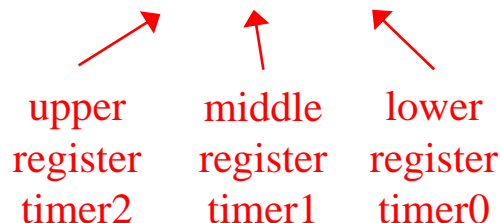
- Microchip instruction cycle is oscillator $\div 4$
- Microchip instruction cycle is oscillator $\div 4$
- We use 4 MHz oscillator \Rightarrow 1 MHz (1 μ sec) instruction cycle

• Code delay

```

    decfsz    Timer0, F    ; Delay loop    1 or 2 cycles
    goto      delay        2 cycles
    
```

- Each loop = 3 μ sec
- 1 sec \Rightarrow 333,333 loops
- 333,333 = 5 16 15



 upper register timer2 middle register timer1 lower register timer0

Programmed 6 16 15

Timer Program - Timing

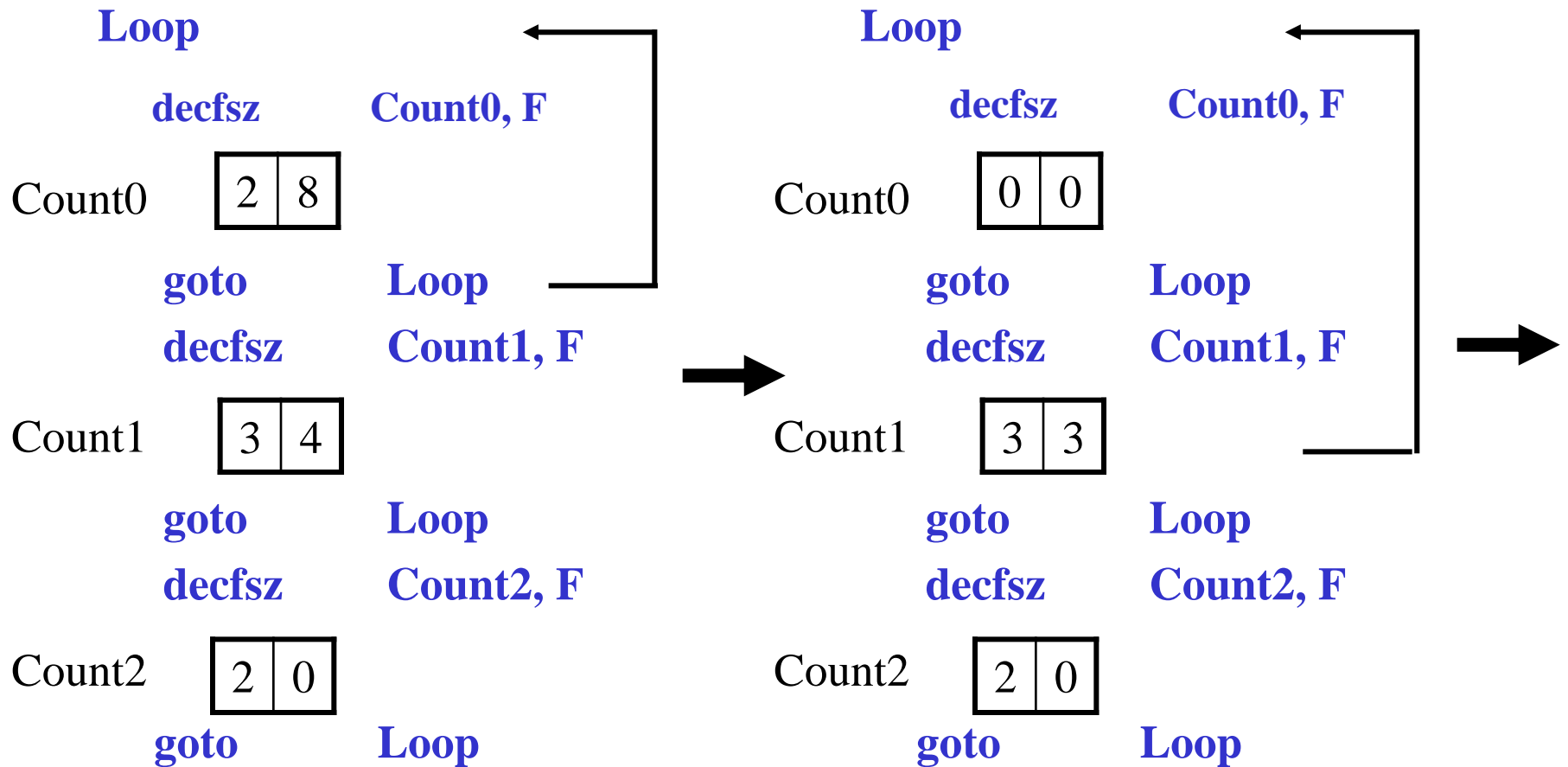
timeLoop

movlw	06h	; get most significant hex value + 1
movwf	Timer2	; store it in count register
movlw	16h	; get next most significant hex value
movwf	Timer1	; store it in count register
movlw	15h	; get least significant hex value
movwf	Timer0	; store it in count register

delay

decfsz	Timer0, F	; Delay loop
goto	delay	
decfsz	Timer1, F	; Delay loop
goto	delay	
decfsz	Timer2, F	; Delay loop
goto	delay	

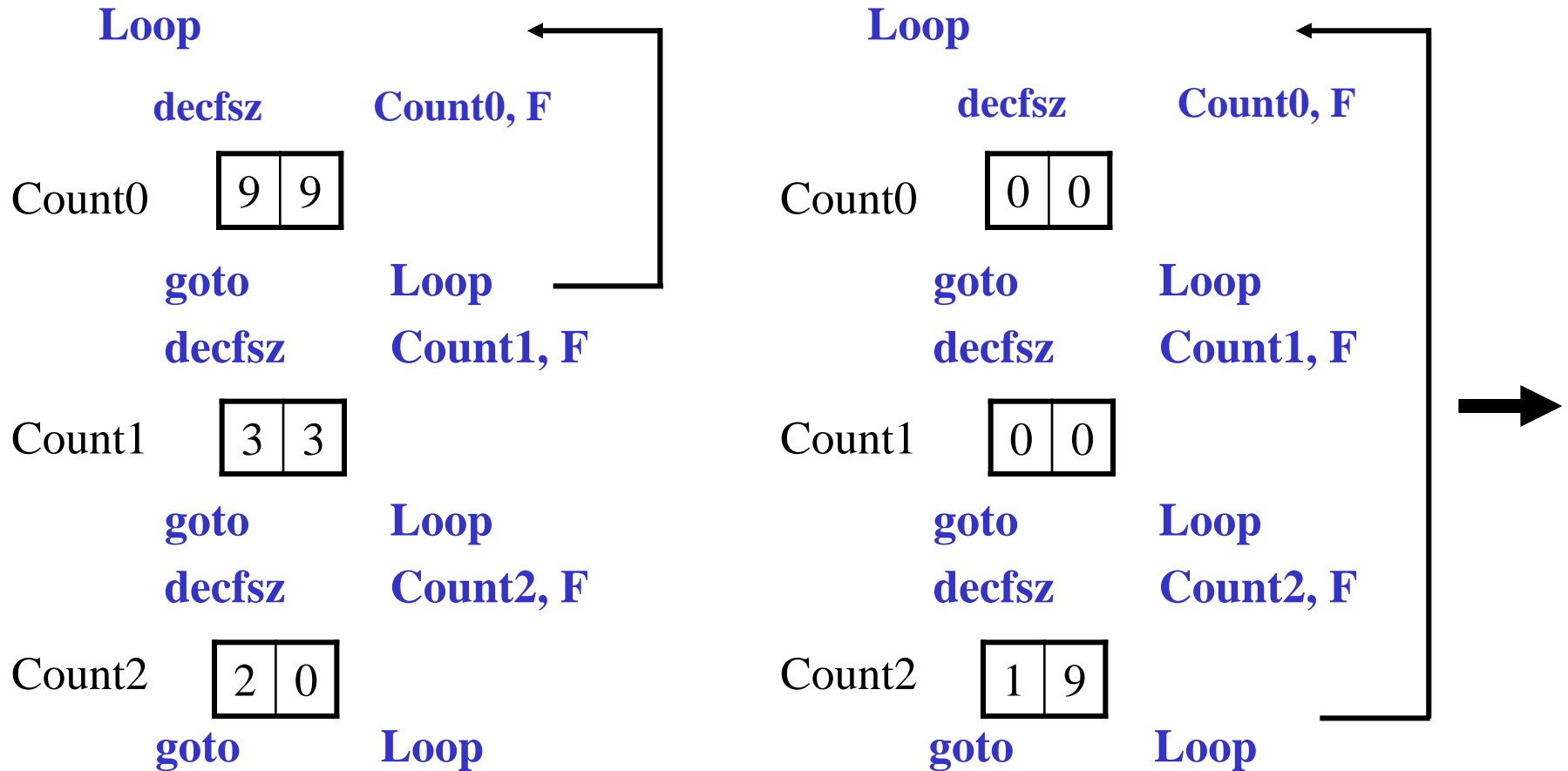
Decimal Timer for 203,428



203,428

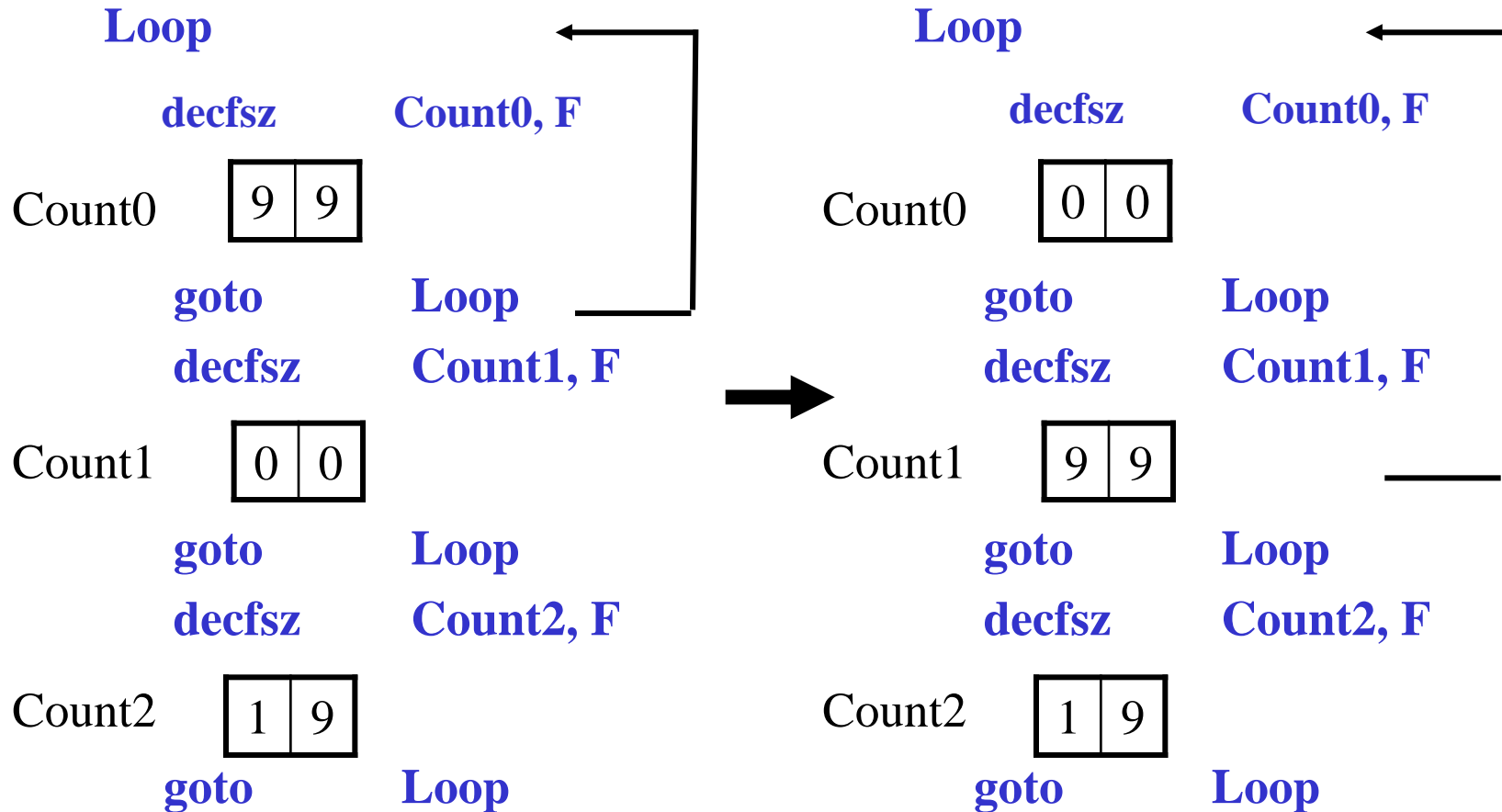
Has an extra count

Decimal Timer Continued



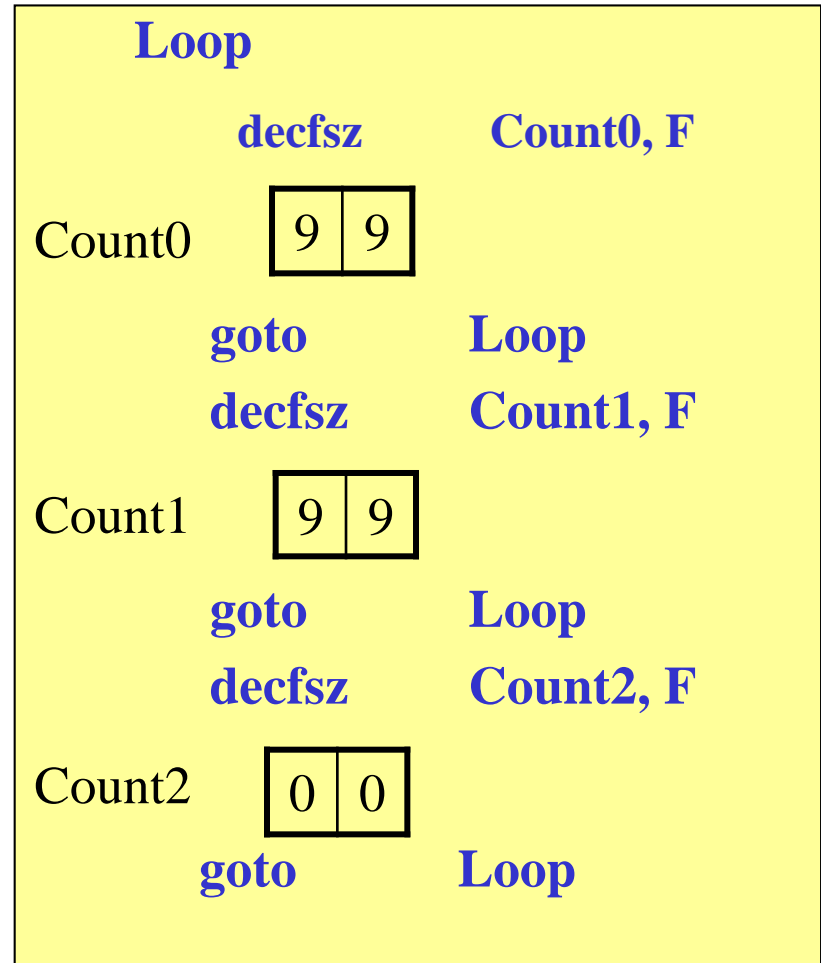
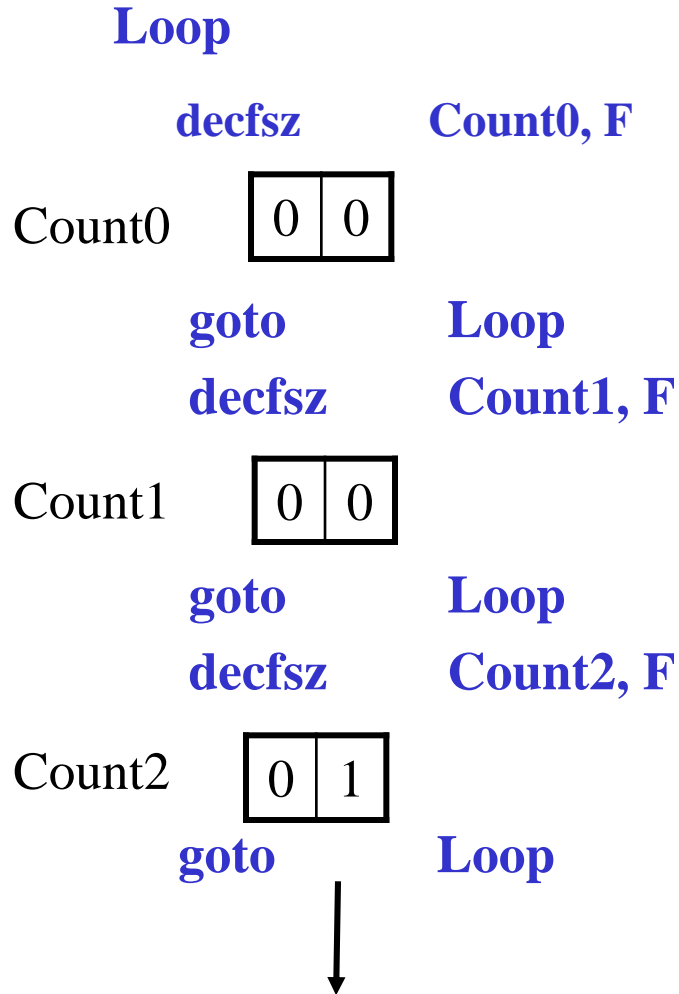
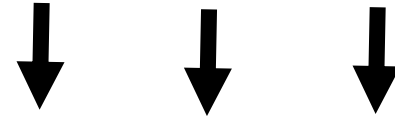
203,399

Decimal Timer Continued



19,099 An Extra Count

Never Get Here



Three 2 digit registers

2	0	3	4	2	8
---	---	---	---	---	---

⋮

2	0	3	4	0	0
---	---	---	---	---	---

2	0	3	3	9	9
---	---	---	---	---	---

⋮

2	0	3	3	0	0
---	---	---	---	---	---

2	0	3	2	9	9
---	---	---	---	---	---



2	0	3	0	0	0
---	---	---	---	---	---

2	0	2	9	9	9
---	---	---	---	---	---

⋮

2	0	2	9	0	0
---	---	---	---	---	---

2	0	2	8	9	9
---	---	---	---	---	---

⋮

2	0	0	0	0	0
---	---	---	---	---	---

1	9	9	9	9	9
---	---	---	---	---	---



1	0	0	0	0	0
---	---	---	---	---	---

0	9	9	9	9	9
---	---	---	---	---	---

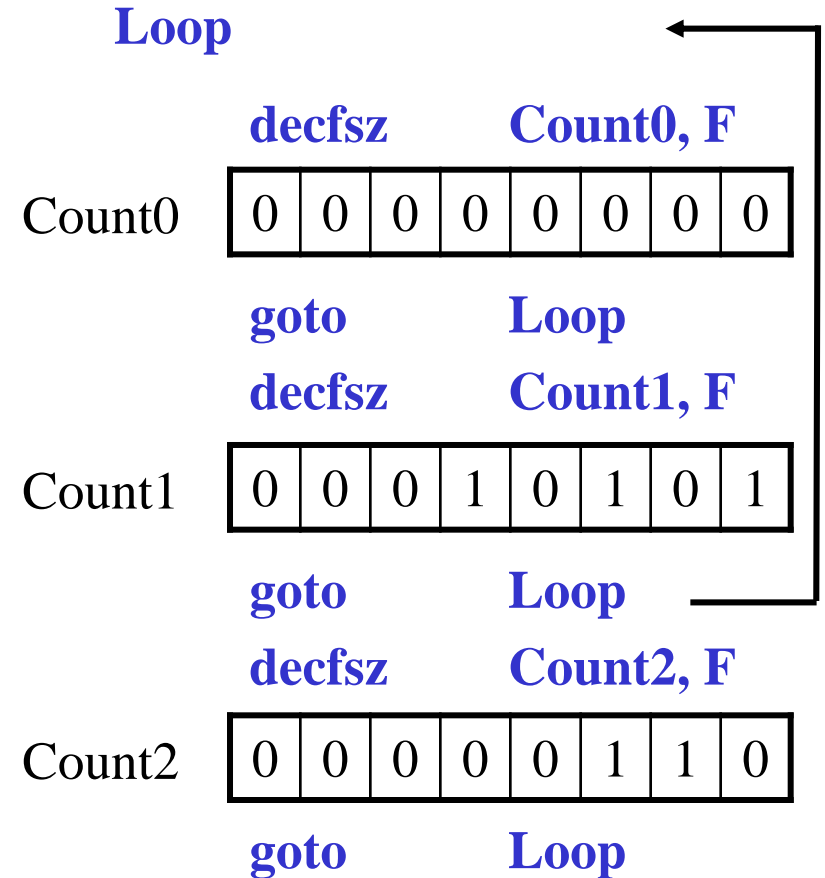
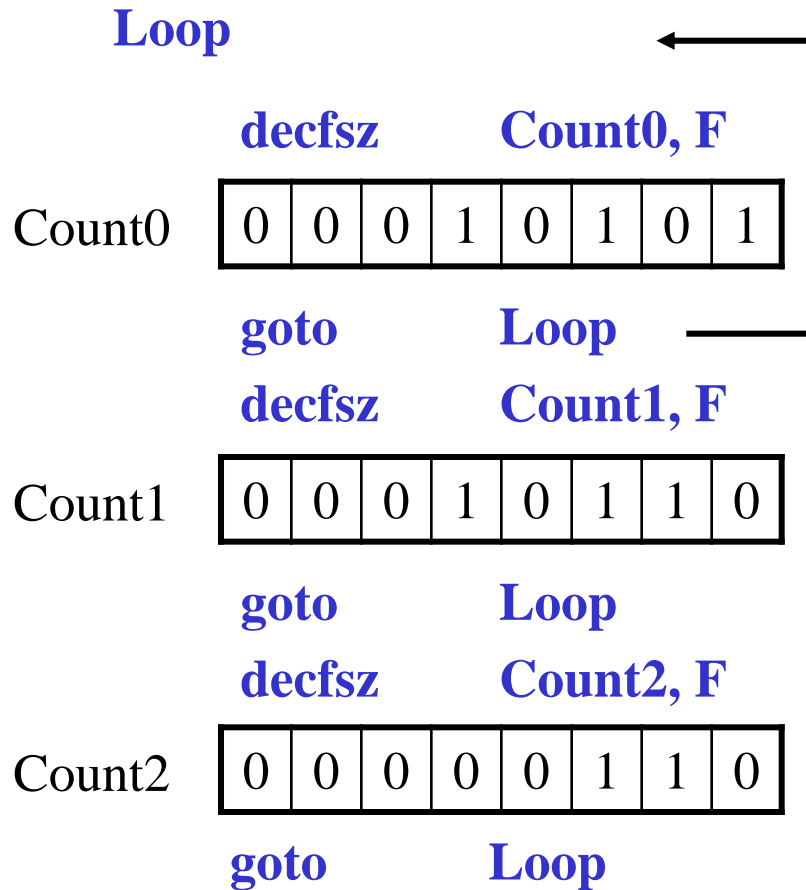
⋮

1	0	0	0	0	0
---	---	---	---	---	---

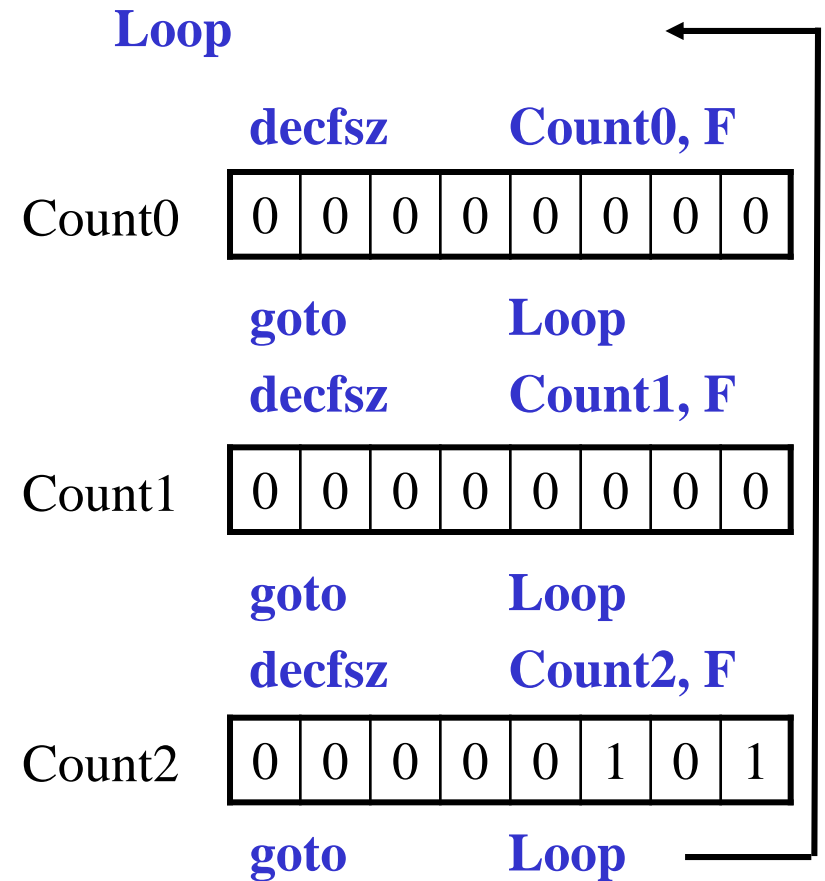
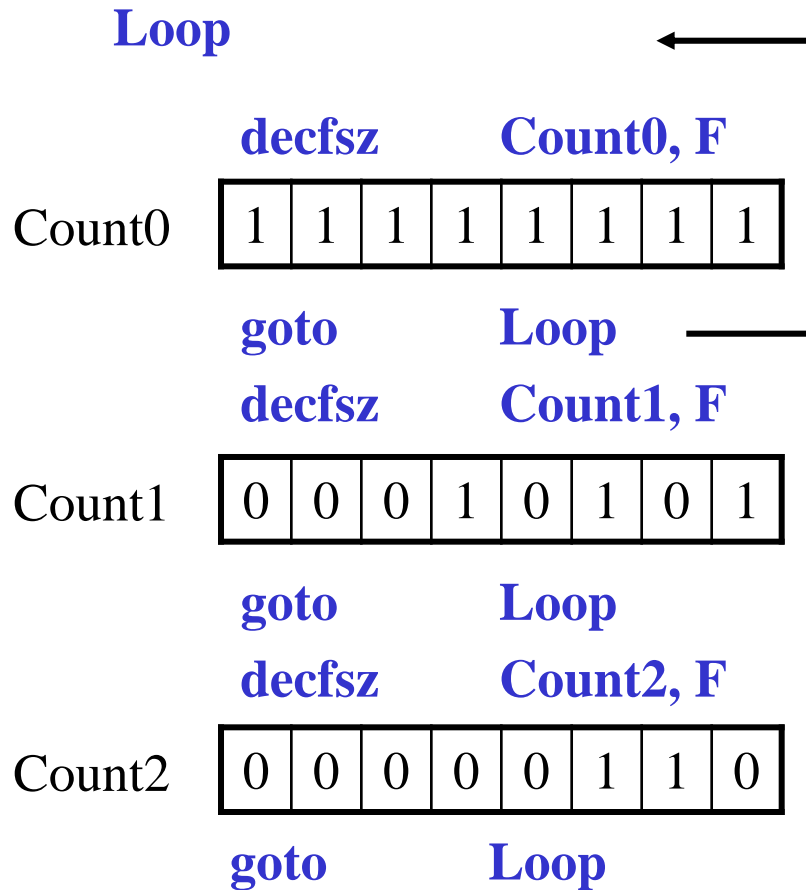
0	0	9	9	9	9
---	---	---	---	---	---

**Algorithm does
NOT continue after
upper 2 digits = 0 0**

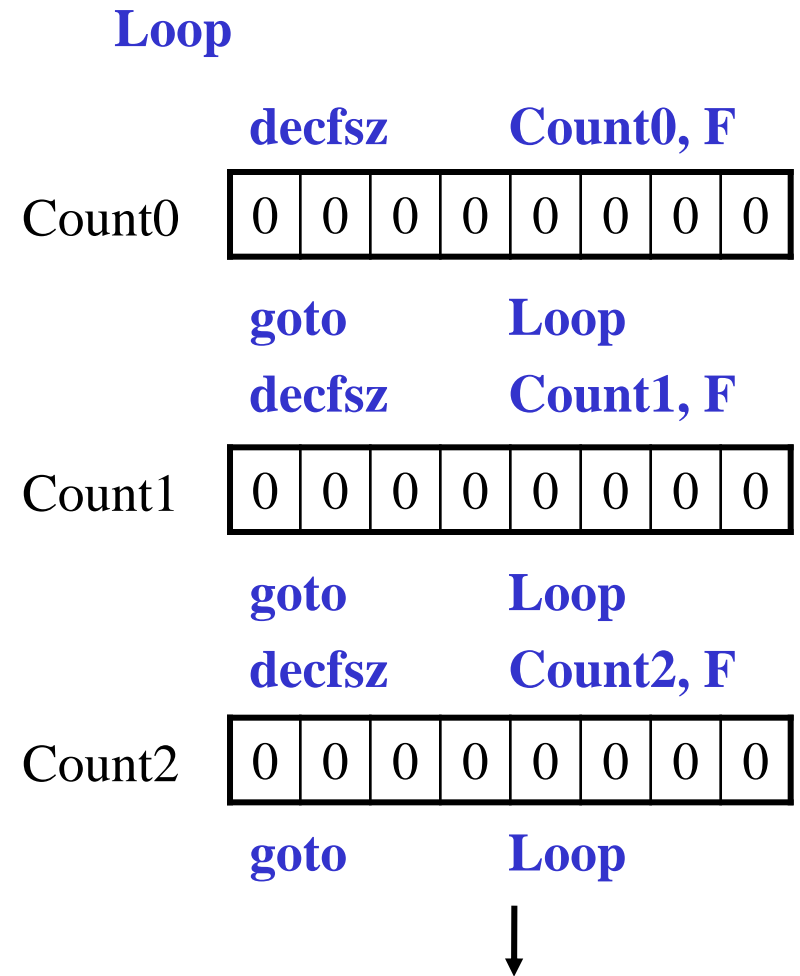
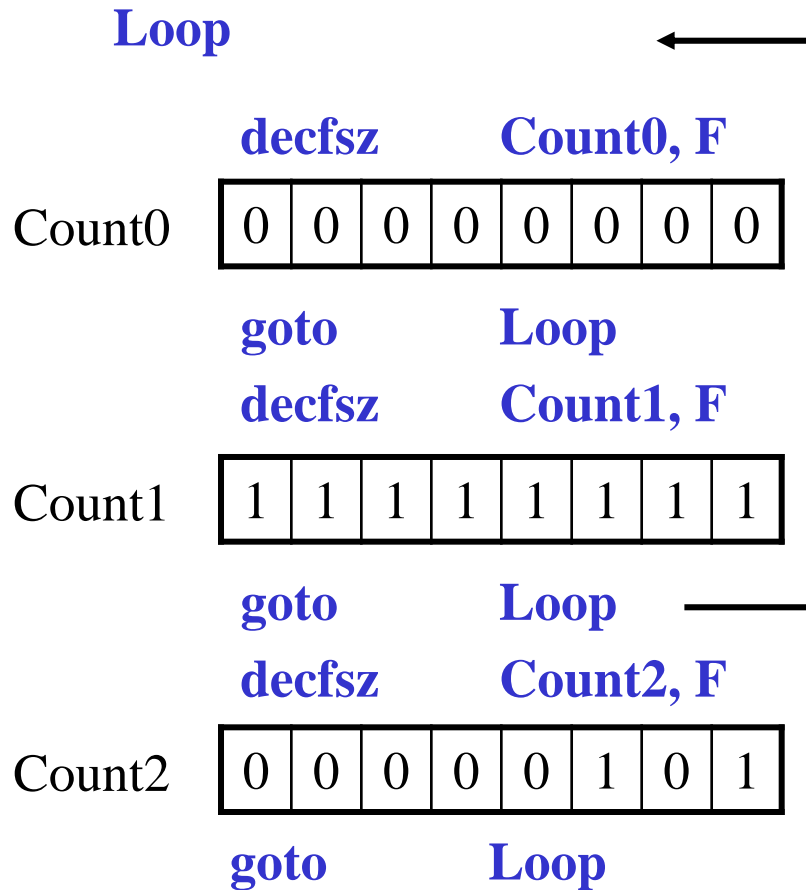
Software Timer



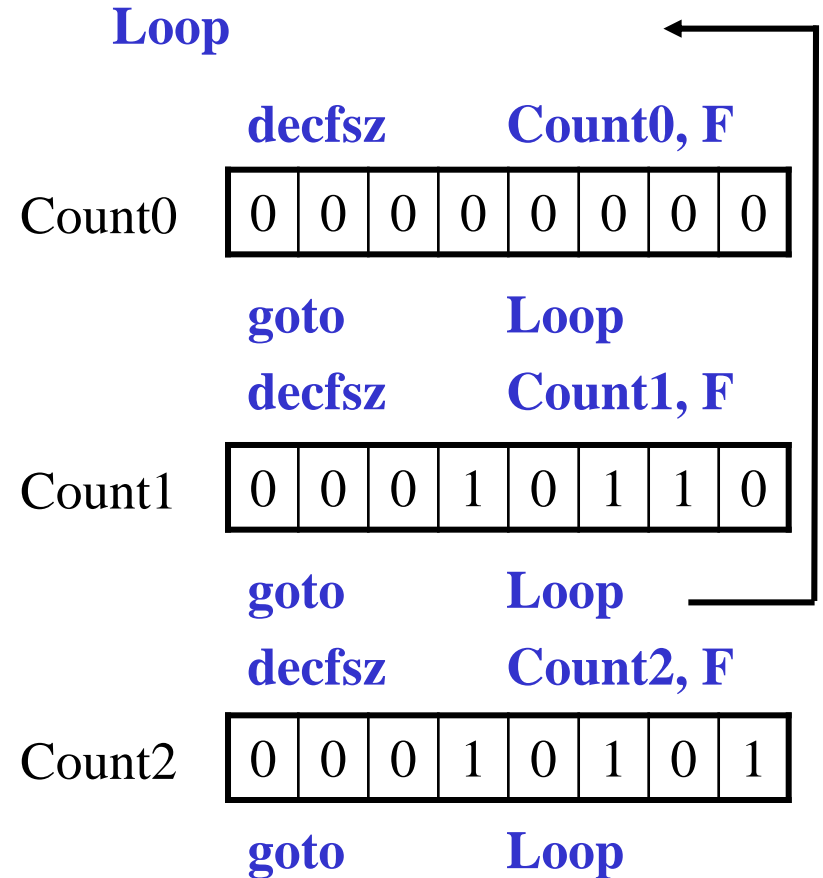
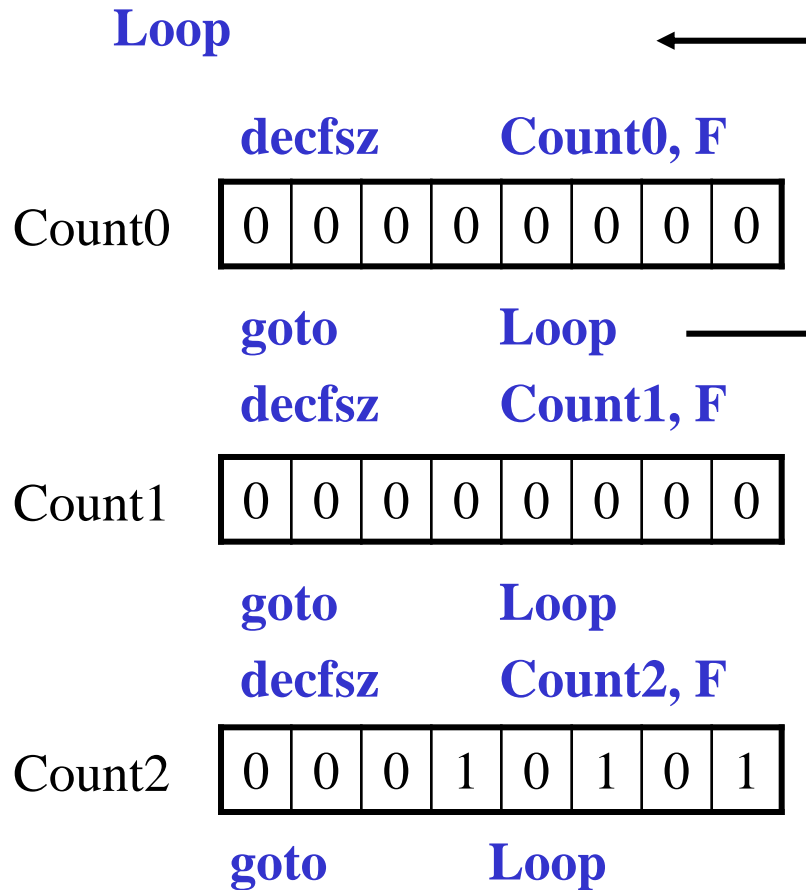
Software Timer Continued



Software Timer Continued

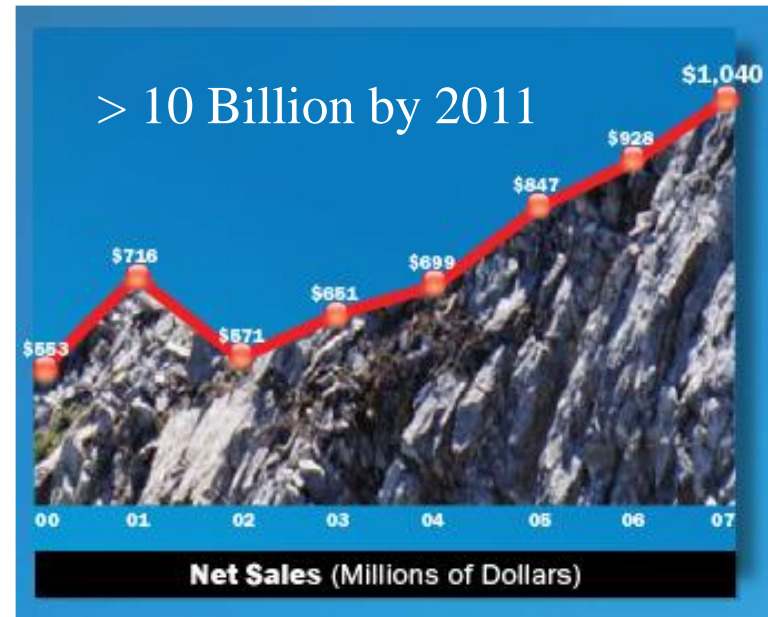


Software Timer Continued



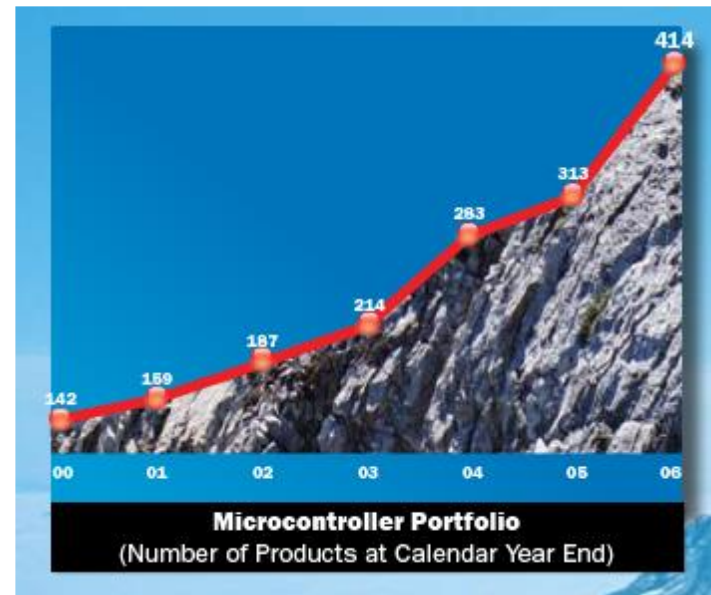
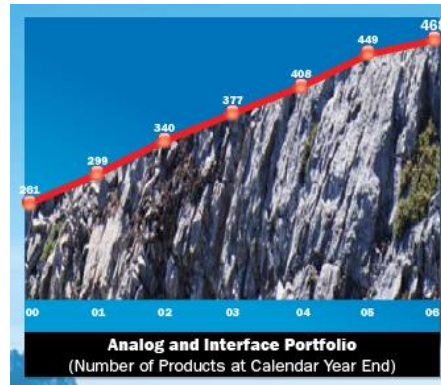
MicroChip Inc.

- Headquarters in Chandler, AZ (www.microchip.com)
- Went from # 20 in microcomputers in 1990 to # 2 (behind Motorola) in 2000. Now # 1 in 8-bit microcontrollers
 - Started in 1989
 - IPO in March 1993
 - 5000+ employees worldwide
 - Revenue: \$1 to \$2 billion (USD) per year
- Innovative Marketing - - changed the way industry sells microcomputers
 - Sell microcomputers & give away development software
 - Give away free samples of their parts (to anyone)
 - Free, very-capable technical support
 - In-house process development, mask generation and fabrication
 - Manufacturing facilities:
 - Chandler (Tempe), Arizona
 - Gresham, Oregon
 - Chachoengsao, Thailand

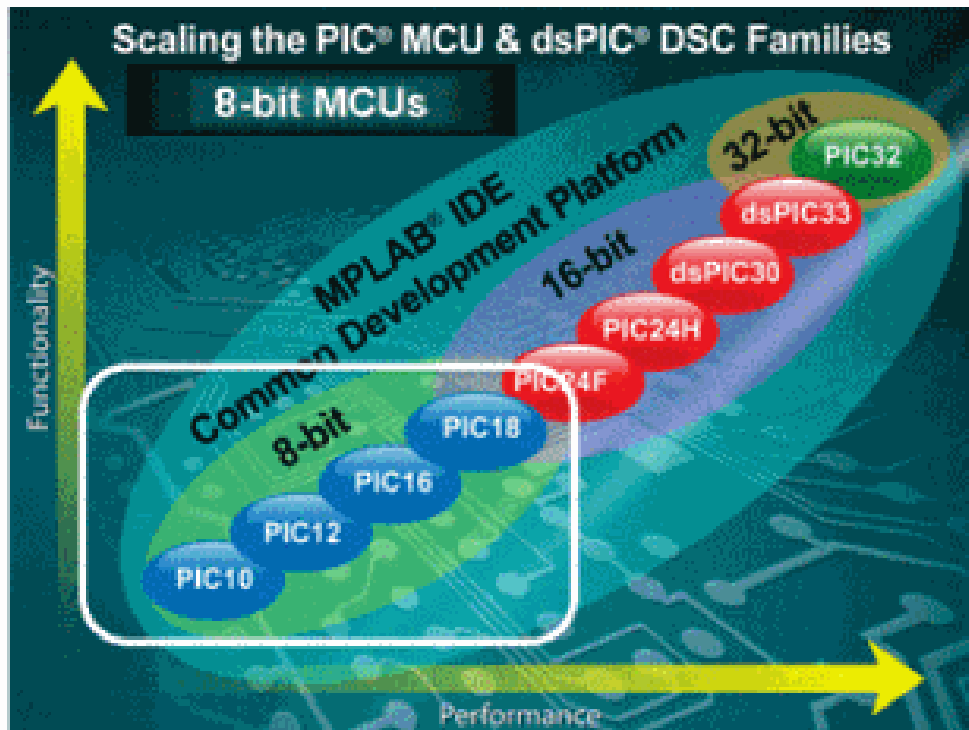


MicroChip Niche

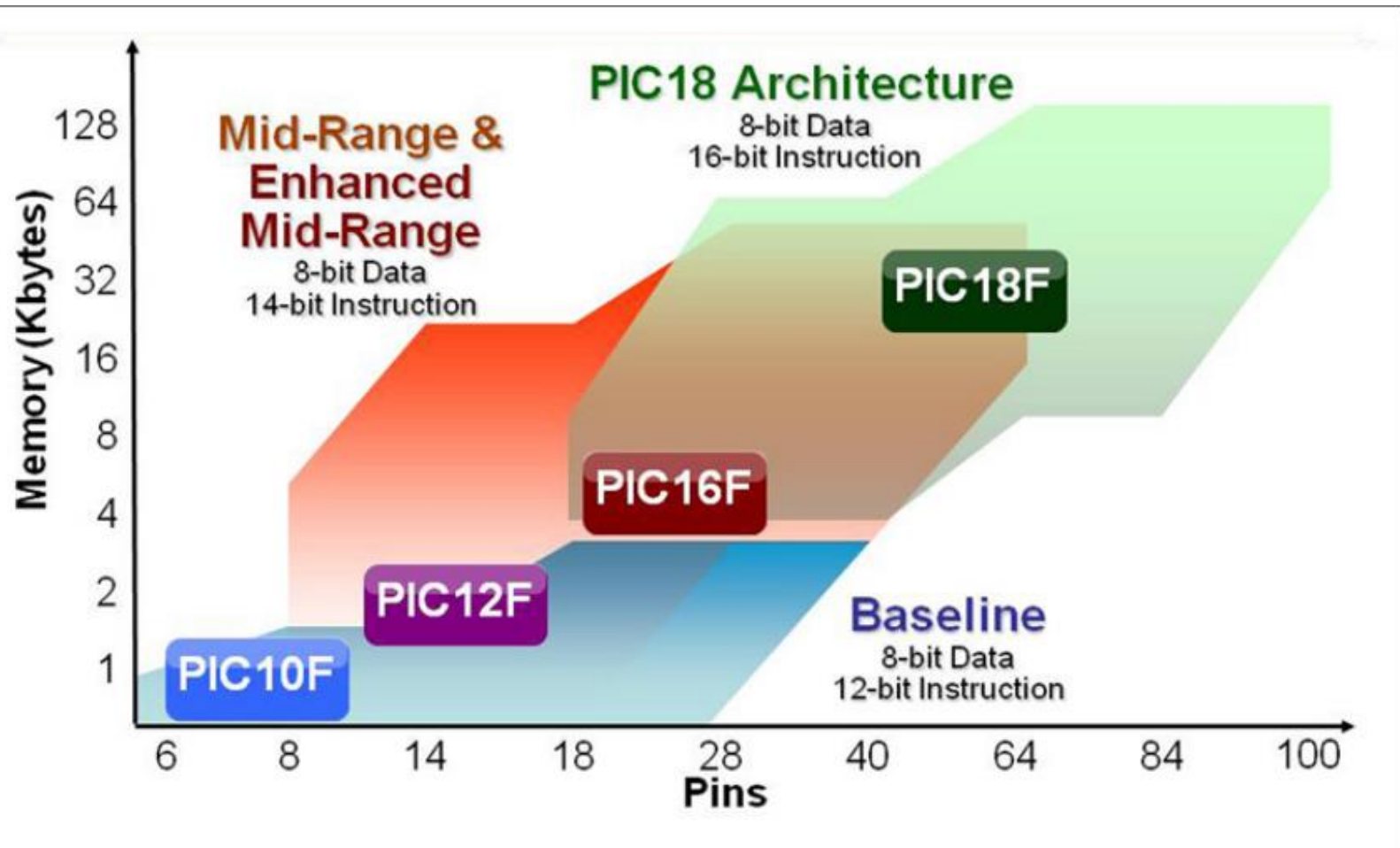
- Very broad product line – mostly very low cost devices
 - Most devices have 8 bit data memory size
 - Program memory size is 12, 14 or 16 bits – optimized for instruction set
 - 6 pin package – 80 pin package
- Ideal Mechatronic microcomputer – major application
 - Internal A/D (Analog-to-Digital) converters
 - Internal PWM (Pulse Width Modulation)
 - High speed (to minimize control phase lags)
 - Internal USART (Universal Synchronous Asynchronous Receiver Transmitter) to communicate with PCs
 - Internal counters and timers



MicroChip Product Line



MicroChip Product Line Architecture Overview

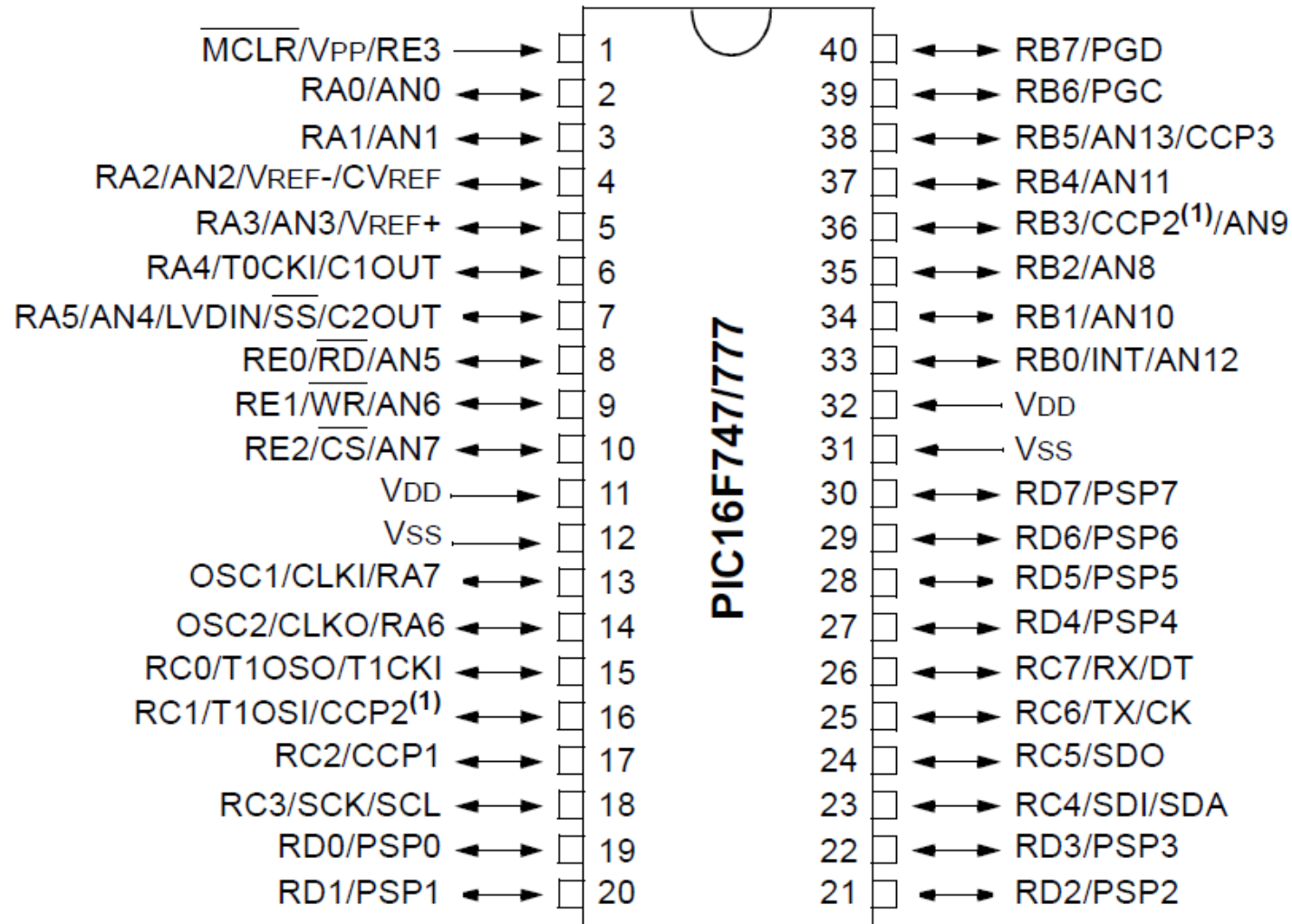


MicroChip PIC16F747 Device

High performance RISC CPU

- Only 35 single word Assembler instructions to learn
- All single cycle instructions (except program branches which are 2 cycle)
- Operating speed: DC - 20 MHz clock input (we use 4 MHz initially)
DC - 200 ns instruction cycle (we use 1 msec)
- 4K x 14 words of FLASH Program Memory
- 192 x 8 bytes of Data Memory (RAM)
- Interrupt capability (for 17 different sources of interrupts)
- Eight level deep hardware stack
- Direct, Indirect and Relative Addressing modes
- Processor can read program memory (used to remotely program device)

MicroChip PIC16F747 Device



Special PIC16F747 Features

- Power-on Reset (POR) (can reset the processor with power on)
- Power-up Timer (PWRT) (allows external circuitry to turn on before program starts)
- Oscillator Start-up Timer (OST) (allows oscillator clock to turn on and stabilize before program starts)
- Fail-Safe Clock Monitor for protecting applications against crystal failure
- • Two-Speed Start-up mode for immediate code execution
- Watchdog Timer (WDT) with its own on-chip oscillator for reliable operation (protects against program bugs) [We will not use this]
- Programmable code protection (microcomputer can be set so your competitors cannot read your program from the chip) [We will not use this]
- Power saving SLEEP mode
- Several oscillator options [We use a high speed crystal oscillator]
- Brown-out detection circuitry for Brown-out Reset (BOR)
- In-Circuit Serial Programming via two pins (to update customer's program remotely)

Many of these features are set with “configuration fuses”

Some are NON – REVERSABLE.

PIC16F747 Internal Peripherals

- 36 Digital Input / Output Pins [Ports A, B, C, D & E]
- 3 Internal Timer / Counters with Different Features
 - Timer0: 8-bit timer/counter with 8-bit prescaler
 - Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external clock
 - Timer2: 8-bit timer/counter with prescaler, postscaler & 8-bit period register (counts up until count matches register then resets - for programmed clock)
- 3 Capture, Compare, PWM (Pulse Width Modulation) modules
 - Capture is 16-bit, max. resolution is 12.5 ns (value of Timer1 is saved in 2 registers when external digital input occurs)
 - Compare is 16-bit, max. resolution is 200 ns (when value of Timer1 equals value in 2 registers an external digital signal is output)
 - PWM max. resolution is 10-bit (used for control output)
- 10-bit, 14-channel Analog-to-Digital converter
- Supports Several Communication Protocols
 - Synchronous Serial Port (SSP) with SPI (Master mode) and I²C (Slave mode)
 - Universal Synchronous Asynchronous Receiver Transmitter (USART) to communicate with PCs (via RS232 protocol for example)
 - Parallel Slave Port (PSP), 8-bits wide with external RD (read), WR (write) and CS (chip select) controls

PIC16F747 Electrical Characteristics

- Low power, high speed CMOS (Complementary Metal Oxide Semiconductor) FLASH technology
- Fully static design (RAM does not require refresh to hold data)
- Wide operating voltage range: 2.0 V to 5.5 V
- High Sink/Source Current Capability from I / O pins: 25 mA
- Industrial temperature range: - 40⁰ C to 85⁰ C
- Low power consumption:
 - Less than 2 mA typical @ 5V, 4 MHz
 - Less than 1 μ A typical standby current (in sleep mode)



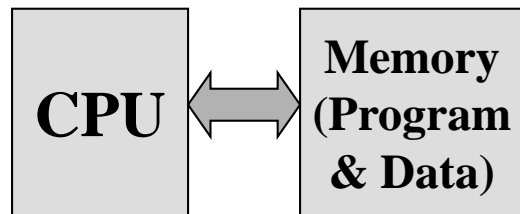
Be careful of 5.5 V limit

Microcomputer Architecture

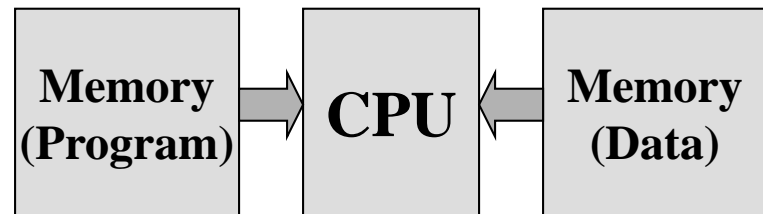
Microcomputers have one of two basic architectures for their internal memory.

- von-Neumann
 - One memory for both instruction and data memory
- Harvard
 - Two separate spaces – one for instruction & one for data memory

von-Neumann

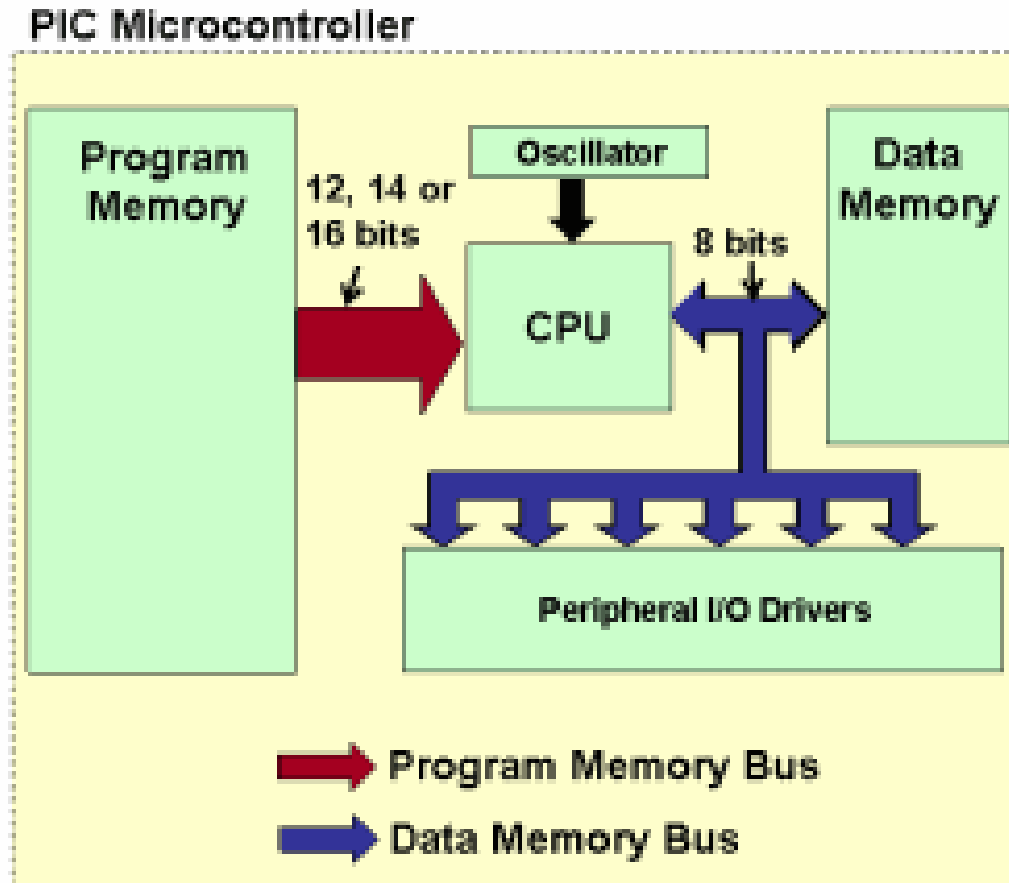


Harvard

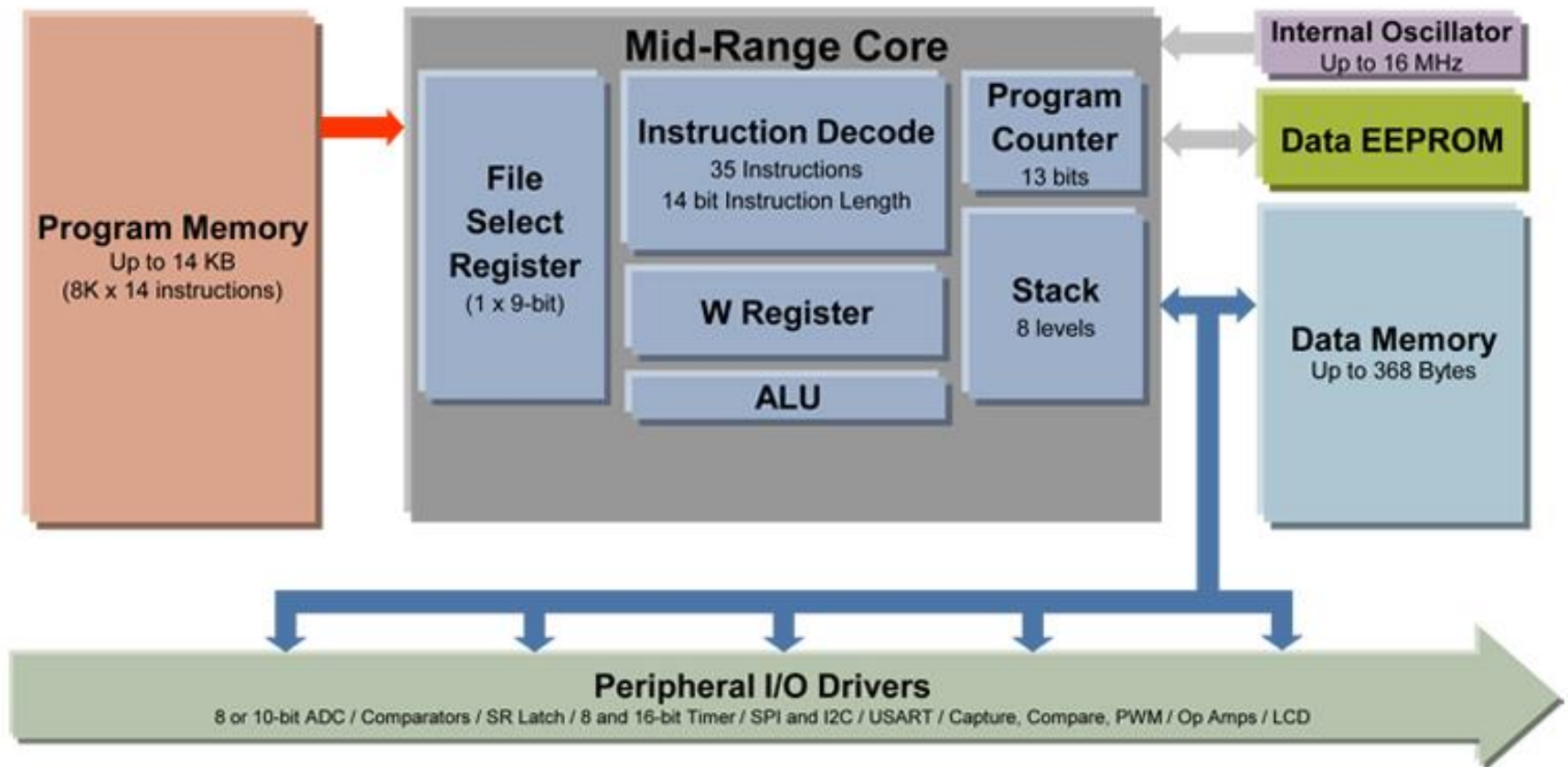


The distinction also refers to the type of memory bus used.

MicroChip Harvard Architecture



MicroChip Mid Range



von-Neumann Architecture Features

- Traditional architecture used in microcomputers and computers
- CPU fetches instructions and data from same memory space
 - Limits operating bandwidth (CPU speed)
 - Memory bus has been called the “von-Neumann bottleneck”
 - “Caching” tries to circumvent bottleneck
- Only one type of memory is needed
 - typically RAM (Random Access Memory) since data is meant to be changed (although a PC bios is typically not RAM – limited use)
 - Compact memory use
- Program and data memory addresses are the same size
- Instructions and data can be easily brought out on I / O pins
- Since instructions and data have same length, two or more memory locations may be required for some instructions.
- Data can be executed as if it were an instruction (at the CPU level)

Harvard Architecture Features

- Newer architecture meant to alleviate von-Neumann bottleneck
- CPU fetches instructions from one memory space and data from another memory space
 - Increased throughput since instructions and data arrive simultaneously
 - Pipelining (caching) is easier
- Program and data memory can be different types
 - RAM for data memory
 - ROM (Read Only Memory) for instructions – much lower cost and higher density than RAM
 - OTP (One Time Programmable) ROM parts (very low cost)
 - EPROM (Erasable Programmable Read Only Memory) – program and can erase with ultraviolet light and then reprogram
 - EEPROM (Electrically Erasable Programmable Read Only Memory) or Flash Memory - program and can erase electronically (by setting voltage levels) and then reprogram
- Program and data memory addresses can be different sizes
 - Can have much more program memory than data memory
 - Permits VLIW (Very Long Instruction Word)
- Instructions and data cannot be easily brought out on I / O pins

Complexity of Instruction Set

- CISC – Complex Instruction Set Computer
 - One instruction is decoded and requires multiple machine cycles to execute
 - Smaller program memory size
 - Slower program execution
- RISC – Reduced Instruction Set Computer
 - One instruction requires one machine cycle to execute
 - Larger program memory size
 - Faster program execution
 - Less instructions to remember

von-Neuman CISC microcomputers

Zilog Z8	45 instructions
Intel 8051	40 instructions
Motorola HC11	109 instructions

MicroChip Microcomputers achieve their high-performance from several advanced architectural features

- Harvard Architecture
- RISC – Reduced Instruction Set Computer
- All instructions are single word
- VLIW – Very Long Instruction Words
- Very Fast Single Machine Cycle Instructions ($\text{clock} \div 4$)
- Instruction Pipelining
- Multi-Purpose Registers to Store Data (RAM)
- PROM, EPROM or Flash Memory to Store Program
- Orthogonal Program Set – Same Instructions Work on Any Register or Any I / O port