MICROCHIP MASTER'S

618 ICD

Audio Spectrum Analyzer
PIC18FXXX Hands On Workshop

MPLAB® IDE V6.0

MPLAB ICD 2

MPLAB C18

PIC18FXXX Hands On Workshop Agenda

- PIC18FXXXX architecture, peripherals and special features
- PICmicro® product overview including future products
- PIC18FXXXX development tool overview
- Audio Spectrum Analyzer Demo Board design
- Lab 1 - Install MPLAB 6.0, MPLAB ICD 2, MPLAB C18, Demo Board, Create Project, Compile and Run, Display Message
- Lab 2 - Develop a traffic light
- Lab 3 - A/D Sampling ISR, Fill A/D sample buffer
- Lab 4 - Apply DFT to A/D sample buffer, scale and display DFT results.
- Lab 5 - Extra credit- Add Automatic Gain Control

PIC18FXXX Workshop Appendix A-D

- The following Appendix topics are available for your reference, but will not be presented today:
  - Appendix A: Optimizing C source code for compiler efficiency
  - Appendix B: PIC18FXXXX Instruction Set, PIC16/17 migration
  - Appendix C: PIC18FXXXX Flash Programming Tips
  - Appendix D: PIC18FXXXX Peripheral Calculation Spreadsheet

Microchip Technology Inc.

Company Overview

Corporate Overview

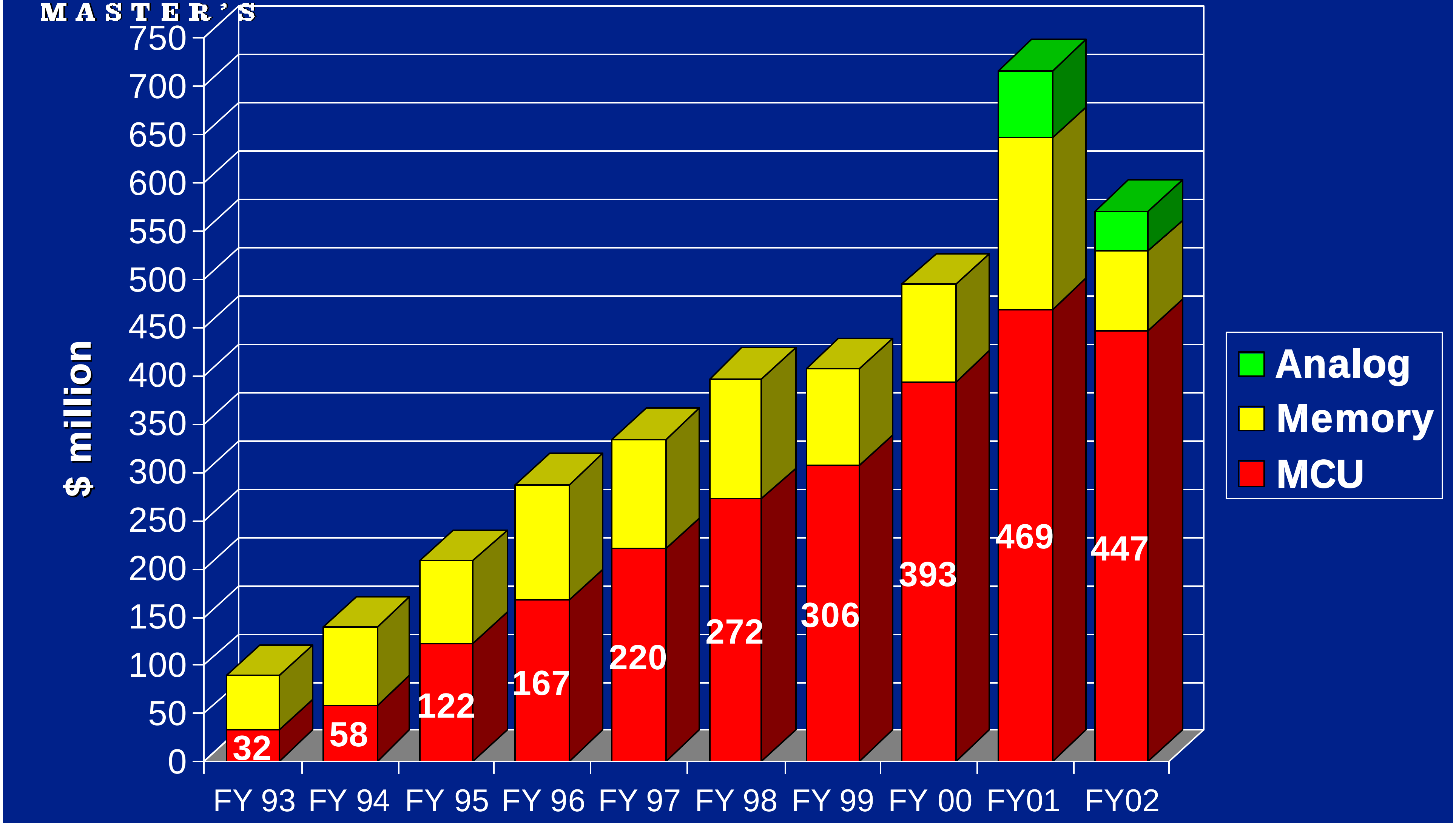
- Leading semiconductor manufacturer:
  - of high-performance, field-programmable 8-bit & 16-bit RISC Microcontrollers
  - of Analog & Interface products
  - of related Memory products
  - for high-volume embedded control applications
- $572 million in product sales in FY02
- More than 3,000 employees
- Headquartered near Phoenix in Chandler, AZ

“The Silicon Desert”

History of the PICmicro® Microcontroller

1989	Pioneered field-programmable MCU: PIC16C5X family
1990	Shipped 1 millionth OTP PICmicro® device
1991	Introduced MPLAB® IDE -- the world’s first Windows 3.0 based development system
1992	Offered ROM program memory to PICmicro customer base
1994	Introduced *Enhanced* FLASH PICmicro MCUs
1996	Introduced the world’s first 8-pin microcontrollers
	Ranked #5 in 8-bit MCU market share
1997	Achieved #2 ranking in 8-bit MCU market share
1999	Introduced PIC18CXXX enhanced core architecture
	Shipped 1 billionth PICmicro MCU
2000	Announced comprehensive FLASH PICmicro product roadmap
2001	Shipped 200,000th development system
2002	Shipped 2 billionth PICmicro MCU

Annual Net Sales Growth



Worldwide Manufacturing Locations

Washington

Fab 3
710K sq feet

Arizona Corp. HQ

Fab 1
270 K sq feet

Fab 2
178 K sq feet

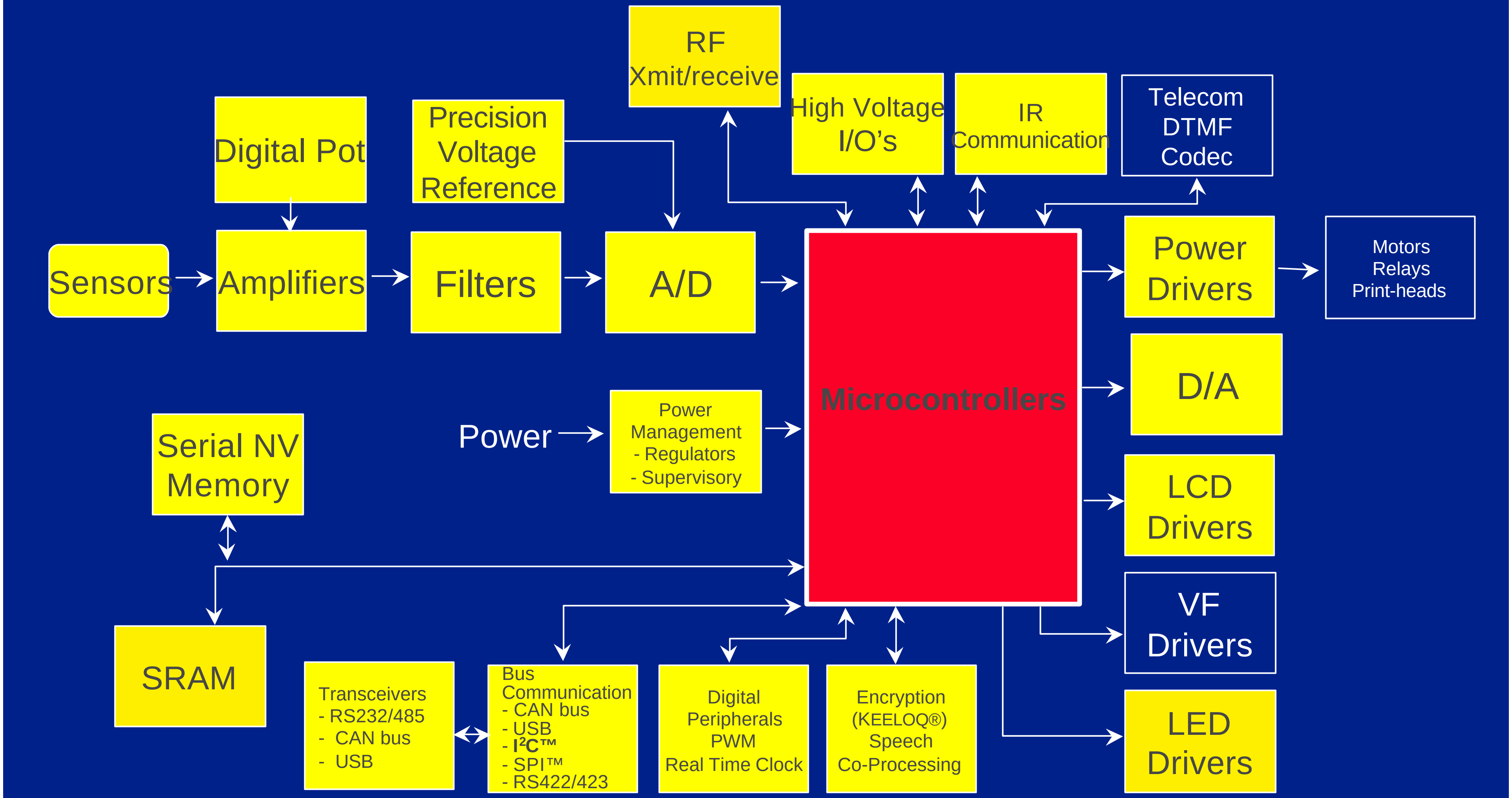
Shanghai Assembly & Test

80 K sq feet

Bangkok Assembly & Test Facility

190K sq feet

Existing PICmicro® MCU Core and Peripheral Blocks



RF
Xmit/receive

High Voltage
I/O's

IR
Communication

Telecom
DTMF
Codec

Digital Pot

Precision
Voltage
Reference

Sensors

Amplifiers

Filters

A/D

Microcontrollers

Power
Drivers

Motors
Relays
Print-heads

D/A

Serial NV
Memory

Power

Power
Management
- Regulators
- Supervisory

LCD
Drivers

VF
Drivers

SRAM

Transceivers
- RS232/485
- CAN bus
- USB

Bus
Communication
- CAN bus
- USB
- I²C™
- SPI™
- RS422/423

Digital
Peripherals
PWM
Real Time Clock

Encryption
(KEELOQ®)
Speech
Co-Processing

LED
Drivers

Microcontroller Market Pyramid

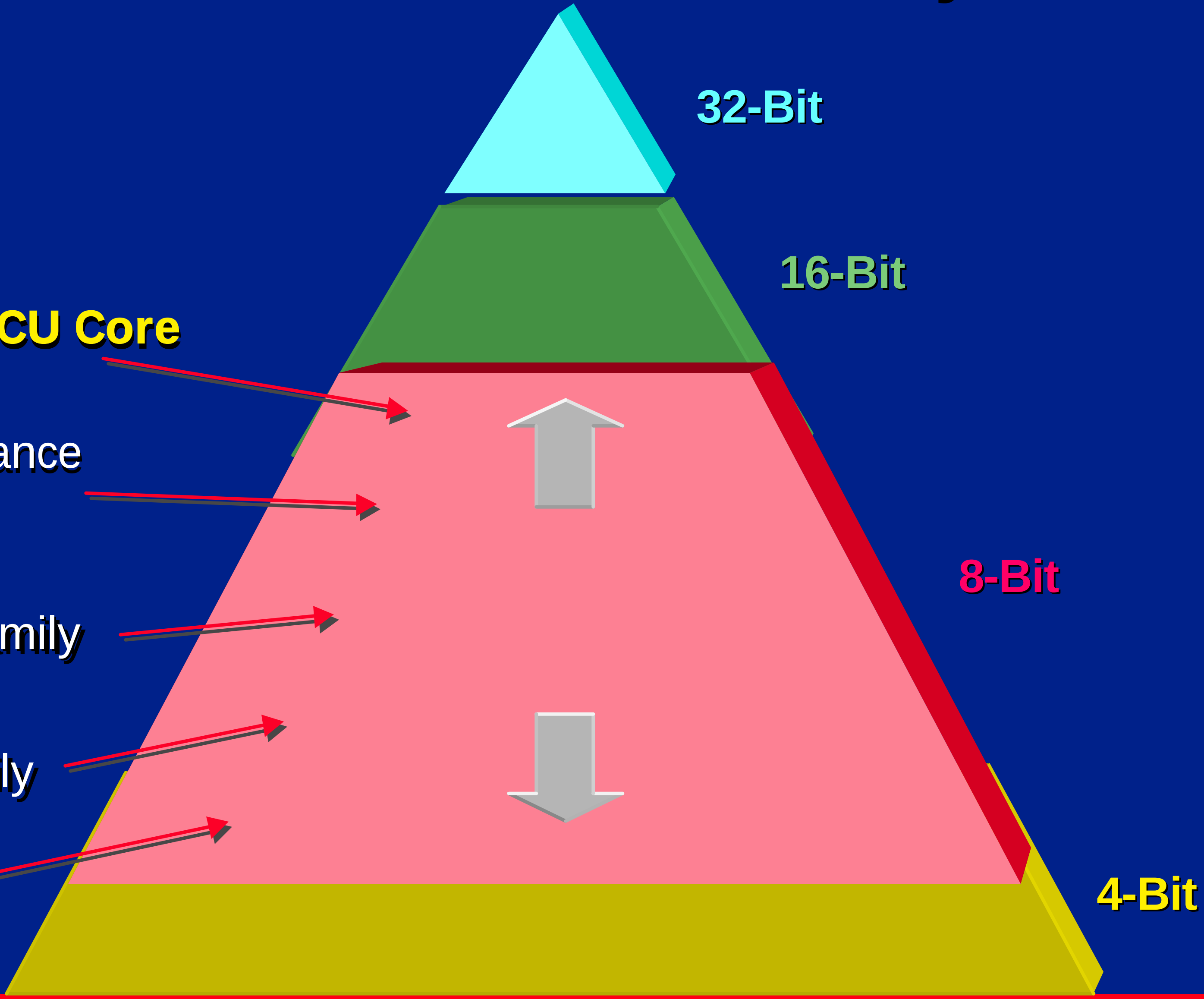
PIC18CXXX
Enhanced MCU Core

PIC17CXXX
High-Performance Family

PIC16CXX
Mid-Range Family

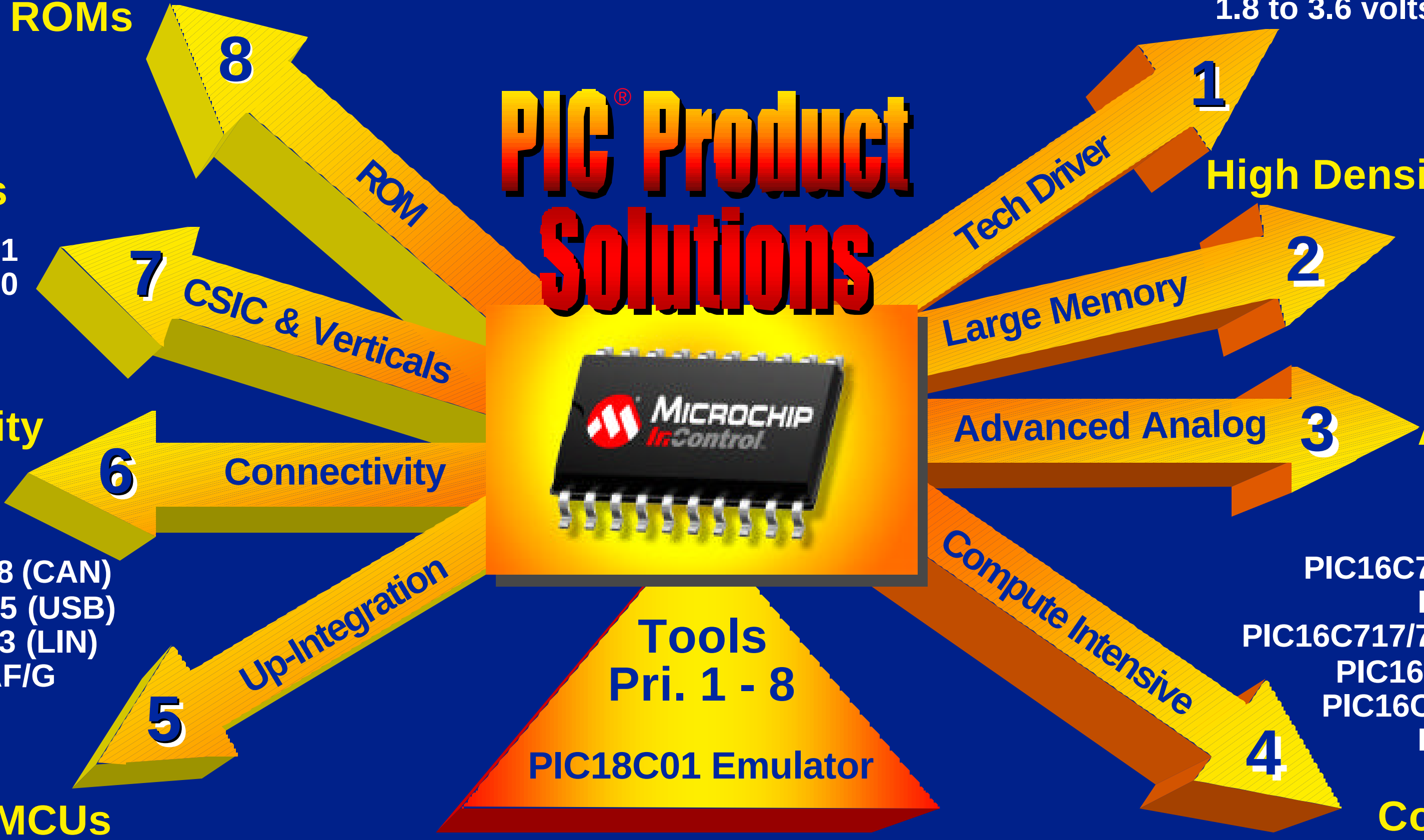
PIC16C5X
Baseline Family

PIC12CXXX
8-Pin Family



MICROCHIP
MASTER'S

PICmicro® Strategic Directions



ROMs

CSICs & Verticals

HCS101/201
HCS365/370
HCS412

Connectivity RF and Wired

PIC18F458/258 (CAN)
PIC16C745/765 (USB)
PIC16C432/433 (LIN)
rfPIC12C509AF/G

8-Pin PIC MCUs High Integration

PIC12F629
PIC12F675

New Process Development

2.0 to 5.5 volts - 0.4 micron
1.8 to 3.6 volts - 0.18 micron
H.V. Foundry

High Density Memory ROMless, FLASH

PIC18C801
PIC18F8720

Adv. Mixed Signal, HV or HI

PIC16C773/774 (12 bit)
PIC16C712/716
PIC16C717/770/771 (12 bit)
PIC16C432/433 (LIN)
PIC16C925/926 (LCD)
PIC16C781/782

Compute Intensive

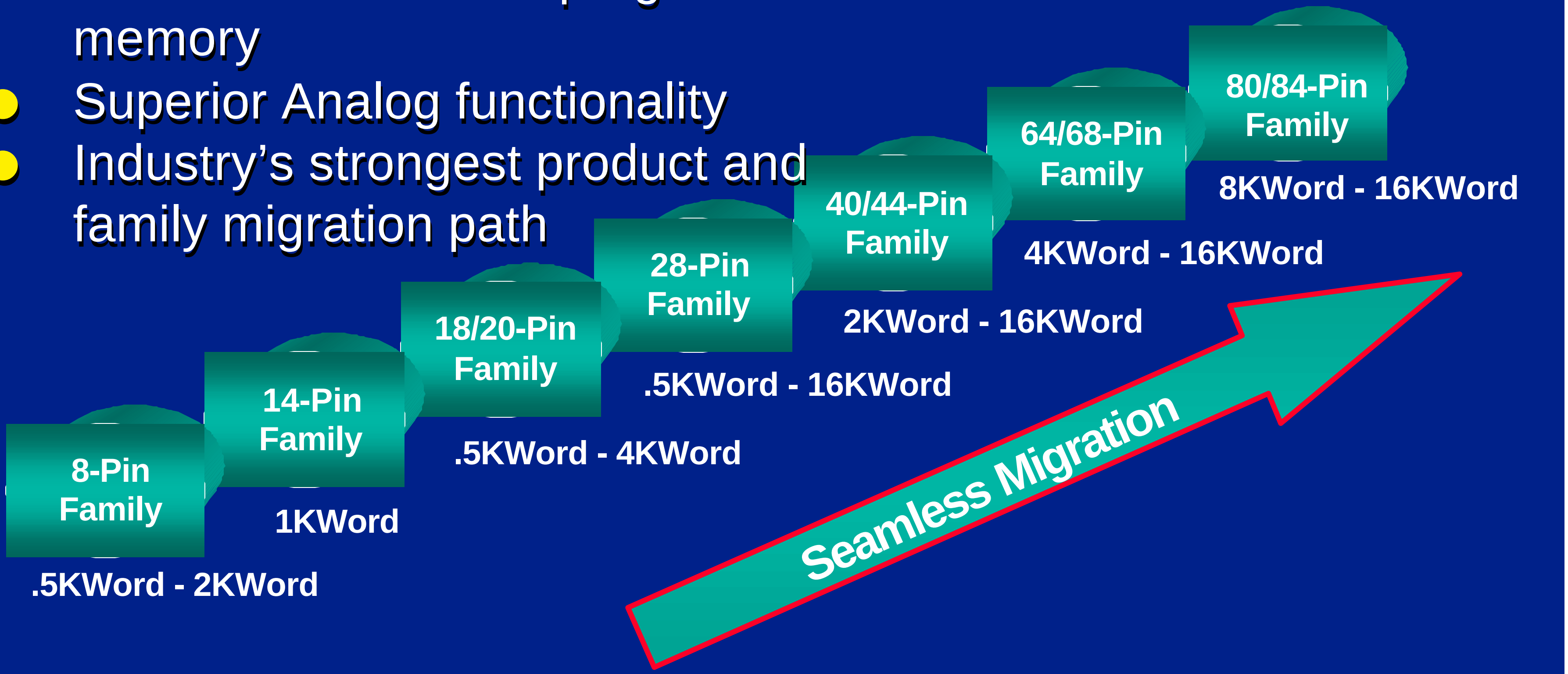
PIC18F452/442/252/242
dsPICä 30F

Development Tools: Whole Product

PICmicro® MCU Product Migration Path
Today

159 Products

- *Enhanced* FLASH, OTP (EPROM), EEPROM and ROM program memory
- Superior Analog functionality
- Industry’s strongest product and family migration path



8-Pin Family

.5KWord - 2KWord

14-Pin Family

1KWord

18/20-Pin Family

.5KWord - 4KWord

28-Pin Family

.5KWord - 16KWord

40/44-Pin Family

2KWord - 16KWord

64/68-Pin Family

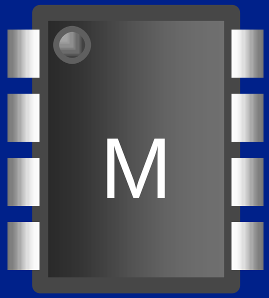
4KWord - 16KWord

80/84-Pin Family

8KWord - 16KWord

Seamless Migration

PICmicro® 8-Pin Families



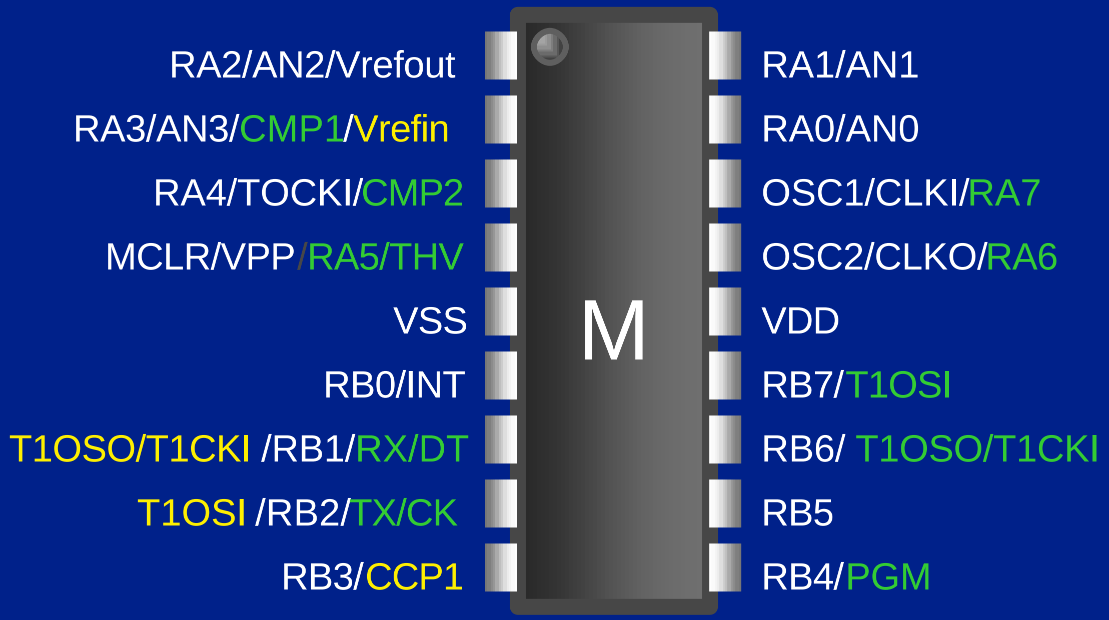
VDD	VSS
GP5/OSC1/CLKIN	GP0/AN0
GP4/OSC2/AN3/CLKOUT	GP1/AN1/Vref
GP3/MCLR/VPP	GP2/TOCKI/AN2/INT

PIC12C508A
PIC12C509A
PIC12CE518
PIC12CE519

PIC12C672
PIC12C671
PIC12CE673
PIC12CE674

PIC12F629
PIC12F675

PICmicro® 18-Pin Families



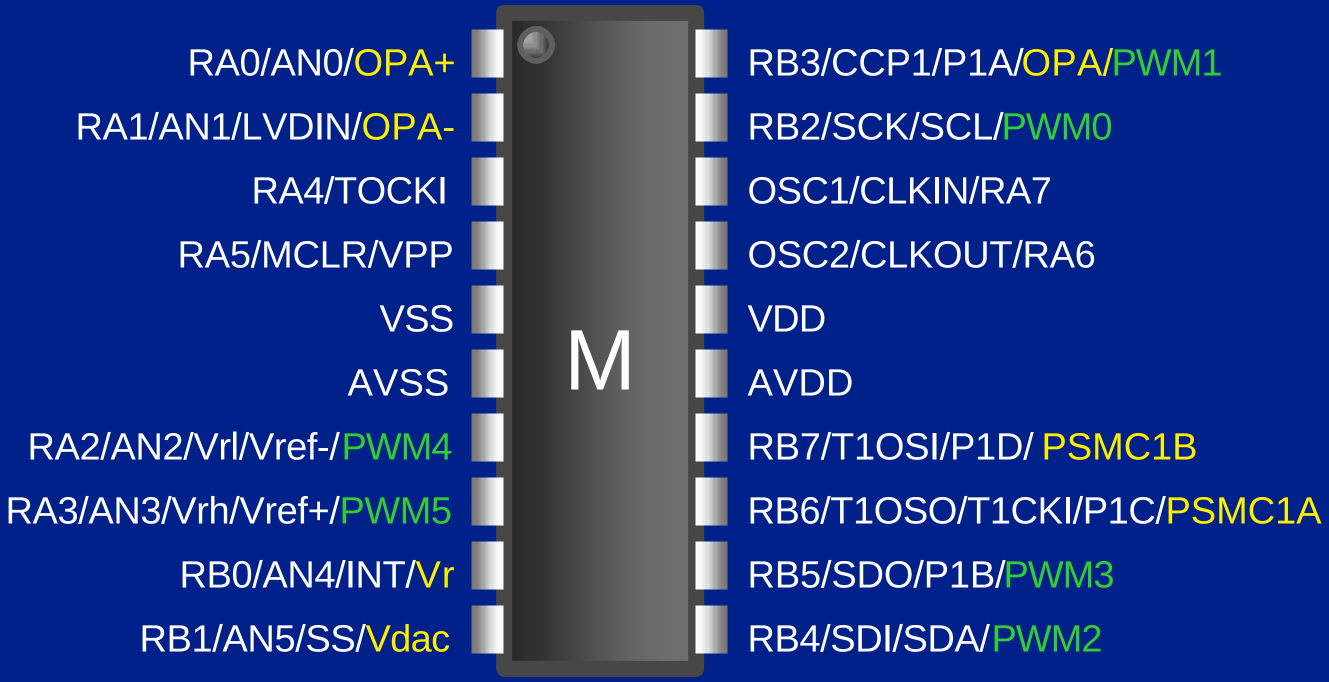
RA2/AN2/Vrefout		RA1/AN1
RA3/AN3/CMP1/Vrefin		RA0/AN0
RA4/TOCKI/CMP2		OSC1/CLKI/RA7
MCLR/VPP/RA5/THV		OSC2/CLKO/RA6
VSS	M	VDD
RB0/INT		RB7/T1OSI
T1OSO/T1CKI /RB1/RX/DT		RB6/ T1OSO/T1CKI
T1OSI /RB2/TX/CK		RB5
RB3/CCP1		RB4/PGM

PIC16CR620A
PIC16C620A
PIC16C621A
PIC16C622A
PIC16CE623
PIC16CE624
PIC16CE625

PIC16C710
PIC16C711
PIC16C712
PIC16C715
PIC16C716
PIC16F87
PIC16F88

PIC16F627
PIC16F628
PIC16F84A
PIC16F818
PIC16F819

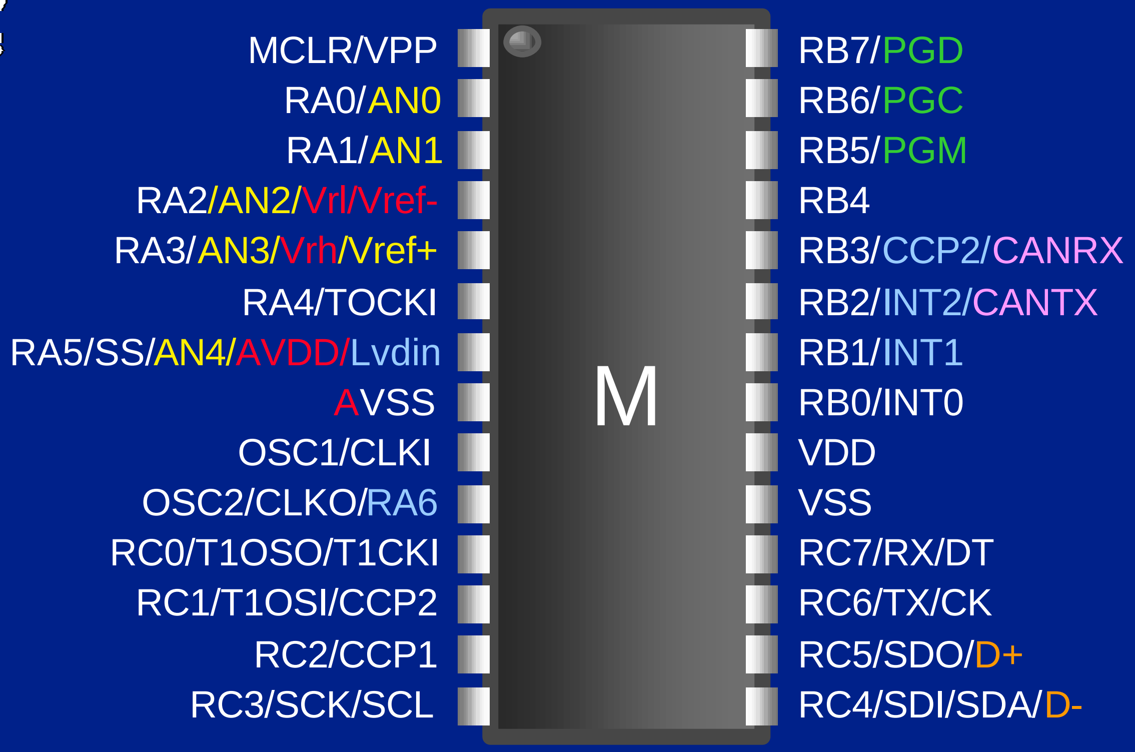
PICmicro® 20-Pin Families



RA0/AN0/OPA+	RB3/CCP1/P1A/OPA/PWM1
RA1/AN1/LVDIN/OPA-	RB2/SCK/SCL/PWM0
RA4/TOCKI	OSC1/CLKIN/RA7
RA5/MCLR/VPP	OSC2/CLKOUT/RA6
VSS	VDD
AVSS	AVDD
RA2/AN2/Vrl/Vref-/PWM4	RB7/T1OSI/P1D/ PSMC1B
RA3/AN3/Vrh/Vref+/PWM5	RB6/T1OSO/T1CKI/P1C/PSMC1A
RB0/AN4/INT/Vr	RB5/SDO/P1B/PWM3
RB1/AN5/SS/Vdac	RB4/SDI/SDA/PWM2

PIC16C717	**PIC16C781**	**PIC18F1320**
PIC16C770	**PIC16C782**	**PIC18F1220**
PIC16C771		

PICmicro® 28-Pin Families



MCLR/VPP	RB7/PGD
RA0/AN0	RB6/PGC
RA1/AN1	RB5/PGM
RA2/AN2/Vrl/Vref-	RB4
RA3/AN3/Vrh/Vref+	RB3/CCP2/CANRX
RA4/TOCKI	RB2/INT2/CANTX
RA5/SS/AN4/AVDD/Lvdin	RB1/INT1
AVSS	RB0/INT0
OSC1/CLKI	VDD
OSC2/CLKO/RA6	VSS
RC0/T1OSO/T1CKI	RC7/RX/DT
RC1/T1OSI/CCP2	RC6/TX/CK
RC2/CCP1	RC5/SDO/D+
RC3/SCK/SCL	RC4/SDI/SDA/D-

PIC16CR63
PIC16C62B
PIC16C63A
PIC16C66
PIC16C642

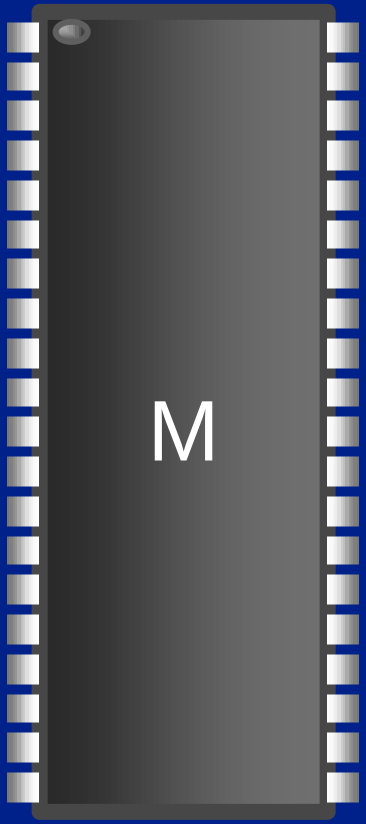
PIC16CR72
PIC16C72A
PIC16C73B
PIC16C76
PIC16C773
PIC16C745

PIC16F73
PIC16F76
PIC16F870
PIC16F872
PIC16F873/A
PIC16F876/A

PIC18F242
PIC18F252
PIC18F2450
PIC18F2550
PIC18F2220
PIC18F2320

PIC18F248
PIC18F258
PIC18C242
PIC18C252

PICmicro® 40-Pin Families



MCLR/VPP	RB7/PGD/KBI3
RA0/AN0	RB6/PGC/KBI2
RA1/AN1	RB5/PGM/KBI1
RA2/AN2/Vrl/Vref-	RB4/KBI0
RA3/AN3/Vrh/Vref+	RB3/CCP2/CANRX
RA4/TOCKI	RB2/INT2/CANTX
RA5/SS/ AN4/AVDD/Lvdin	RB1/INT1
RE0/RD/AN5	RB0/INT0
RE1/WR/AN6	VDD
RE3/CS/AN7	VSS
AVDD	RD7/PSP7/PD
AVSS	RD6/PSP6/PC
OSC1/CLKI	RD5/PSP5/PB
OSC2/CLKO/RA6	RD4/PSP4/ECC/PA
RC0/T1OSO/T1CKI	RC7/RX/DT
RC1/T1OSI/CCP2	RC6/TX/CK
RC2/CCP1	RC5/SDO/D+
RC3/SCK/SCL	RC4/SDI/SDA/D-
RD0/PSP0/C1IN+	RD3/SPS3/C2IN-
RD1/PSP1/C1IN-	RD2/PSP2/C2IN+

PIC16CR65
PIC16C65B
PIC16C67
PIC16C662

PIC16C74B
PIC16C77
PIC16C774
PIC16C765

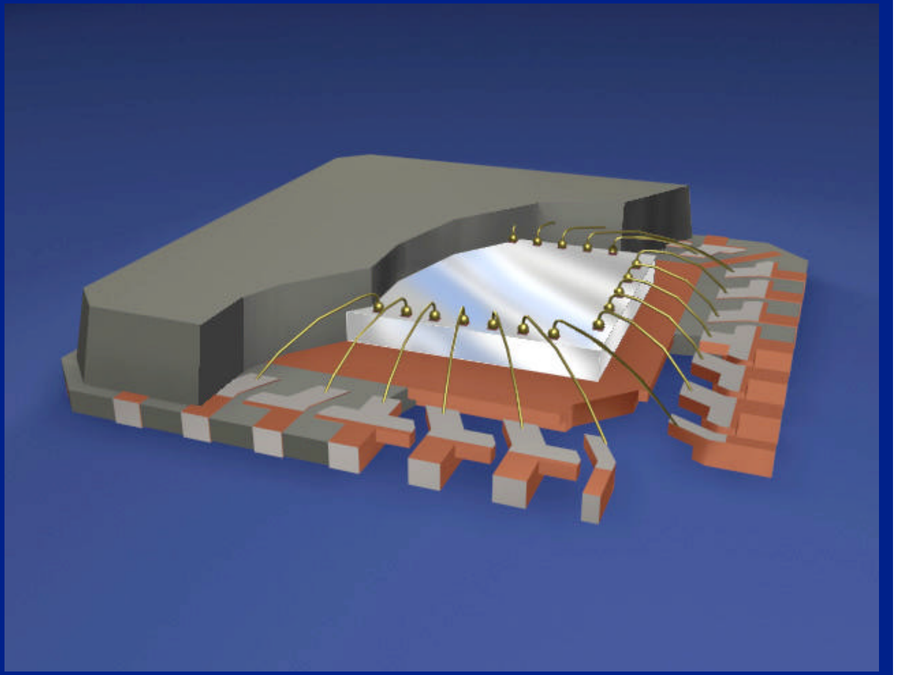
PIC16F74
PIC16F77
PIC16F871
PIC16F874/A
PIC16F877/A

PIC18F442
PIC18F452
PIC18F4450
PIC18F4550
PIC18F4220
PIC18F4320

PIC18F448
PIC18F458
PIC18C442
PIC18C452

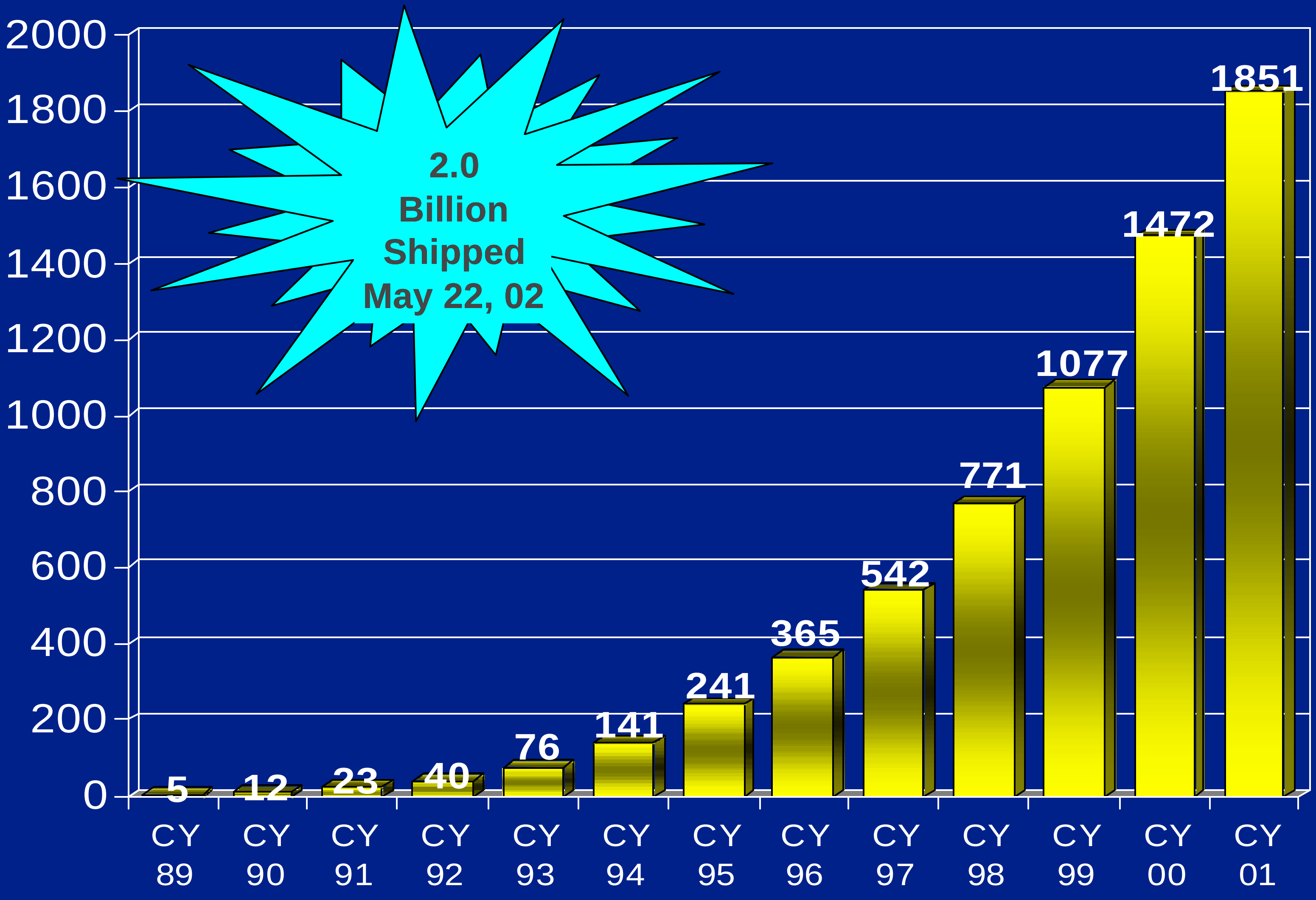
Quad Flat No Lead (QFN)

- Moving into a JEDEC standard environment
- JEDEC is naming them:
  - QFN (ala 28/40 lead)
  - Quad Flat No Lead
  - DFN (ala 8 lead)
  - Dual Flat No Lead
- MCHP package ordering names will not change
  - /ML and /MF



Cumulative PICmicro® Shipment

(Millions of Units)



2.0 Billion Shipped May 22, 02

Thousands of Customers

Consumer

- Black & Decker
- Coleman
- Genie
- Goldstar
- Hamilton Beach
- JVC
- Mitsubishi
- Panasonic
- Philips
- Samsung
- Sanyo
- Sega
- Sony
- Sunbeam
- Toshiba
- Whirlpool

Automotive

- BMW
- Ford
- Delphi
- Honda
- JCI
- Lear
- Lexus
- Mercedes/Benz
- Nissan
- Robert Bosch
- Sagem
- Siemens/VDO
- Stribel
- Toyota
- TRW
- Valeo

Office Automation

- Alps
- Apple Computer
- Conner
- Compaq
- DEC
- Dell Computer
- Hewlett Packard
- IBM
- Logitech
- Microsoft
- Mitsumi
- NCR
- Panasonic
- Quantum
- Texas Instruments

Telecom

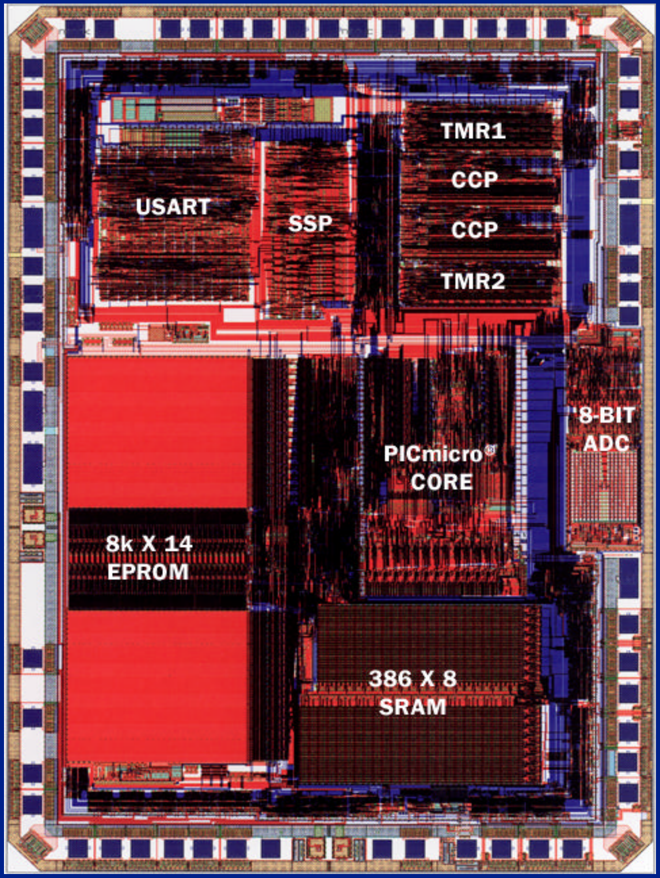
- Codex
- Ericsson
- Kyocera
- Motorola
- Nokia
- Northern Telecom
- Pacific Monolithics
- Pulsecomm
- Qualcomm
- Rockwell
- Sagem
- Samsung
- Siemens
- UDS

Industrial

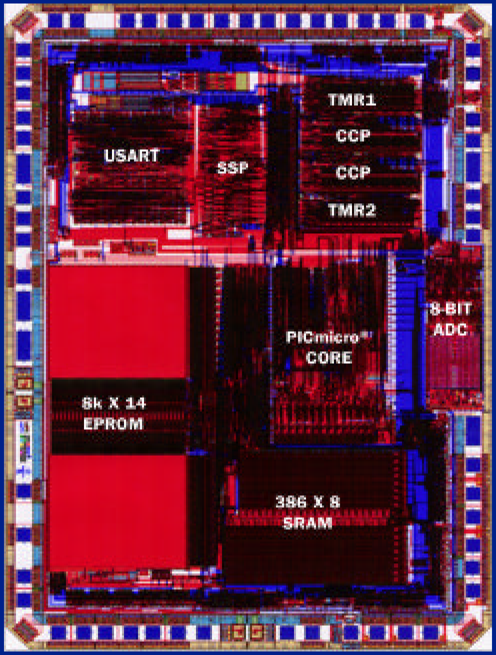
- Allen-Bradley
- American Sensors
- Banner
- Code Alarm
- Foxboro
- General Electric
- Honeywell
- ILCO-Unican
- Invensys
- Pitney Bowes
- Tandy
- United Technologies
- Wayne Systems
- Whirlpool

Process Technology Advancements

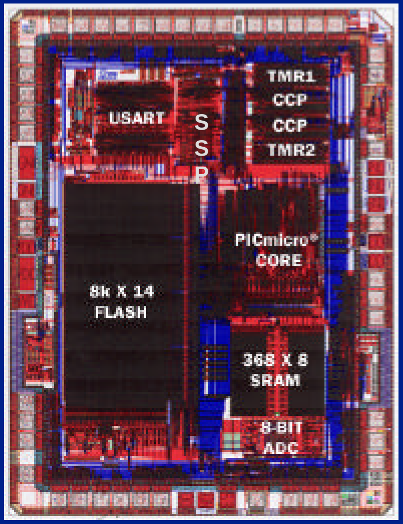
PIC16C77 (0.9µ)



PIC16C77 (0.7µ)*



PIC16F77 (0.5µ)



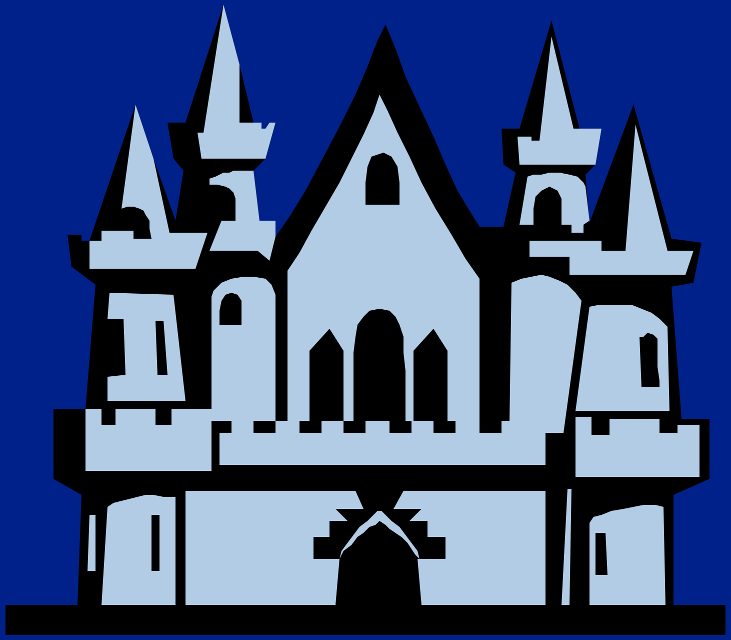
*** Equivalent device**

MICROCHIP MASTER'S

Worldwide 8-bit Microcontroller Market Share - Units

No.	1990	1991	1992	1993	1994	1995/96	1997-00
Rank	Rank	Rank	Rank	Rank	Rank	Rank	Rank
1	Motorola	Motorola	Motorola	Motorola	Motorola	Motorola	Motorola
2	Mitsubishi	Mitsubishi	Mitsubishi	Mitsubishi	Mitsubishi	Mitsubishi	**Microchip**
3	NEC	NEC	Intel	NEC	NEC	SGS-Thomson	NEC
4	Intel	Intel	NEC	Hitachi	Philips	NEC	Hitachi
5	Hitachi	Hitachi	Philips	Philips	Intel	**Microchip**	ST-Micro
6	Philips	Philips	Hitachi	Intel	**Microchip**	Philips	Infineon
7	Matsushita	Matsushita	Matsushita	SGS	Zilog	Zilog	Mitsubishi
8	National	SGS-Thomson	SGS	**Microchip**	SGS	Hitachi	Philips
9	Siemens	Siemens	National	Matsushita	Matsushita	Fujitsu	Toshiba
10	TI	TI	TI	Toshiba	Hitachi	Intel	Atmel
11	Sharp	National	Zilog	National	Toshiba	Siemens	Zilog
12	Oki	Toshiba	Toshiba	Zilog	National	Toshiba	Fujitsu
13	Toshiba	Sony	Siemens	TI	TI	Matsushita	Matsushita
14	SGS-Thomson	Sharp	**Microchip**	Siemens	Ricoh	TI	Realtek
15	Zilog	Oki	Sharp	Sharp	Fujitsu	National	Samsung
16	Matra MHS	Zilog	Sanyo	Oki	Siemens	Temic	National
17	Sony	**Microchip**	Matra MHS	Sony	Sharp	Sanyo	Sanyo
18	Fujitsu	Matra MHS	Sony	Sanyo	Oki	Ricoh	Elan
19	AMD	Fujitsu	Oki	Fujitsu	Sony	Oki	TI
20	**Microchip**	Sanyo	Fujitsu	AMD	Temic	Sharp	Sony

Based on unit shipment volume 1990-2000, Source: Dataquest, July 2001



PIC18 Architecture And Peripherals

PIC18 Architecture
Features

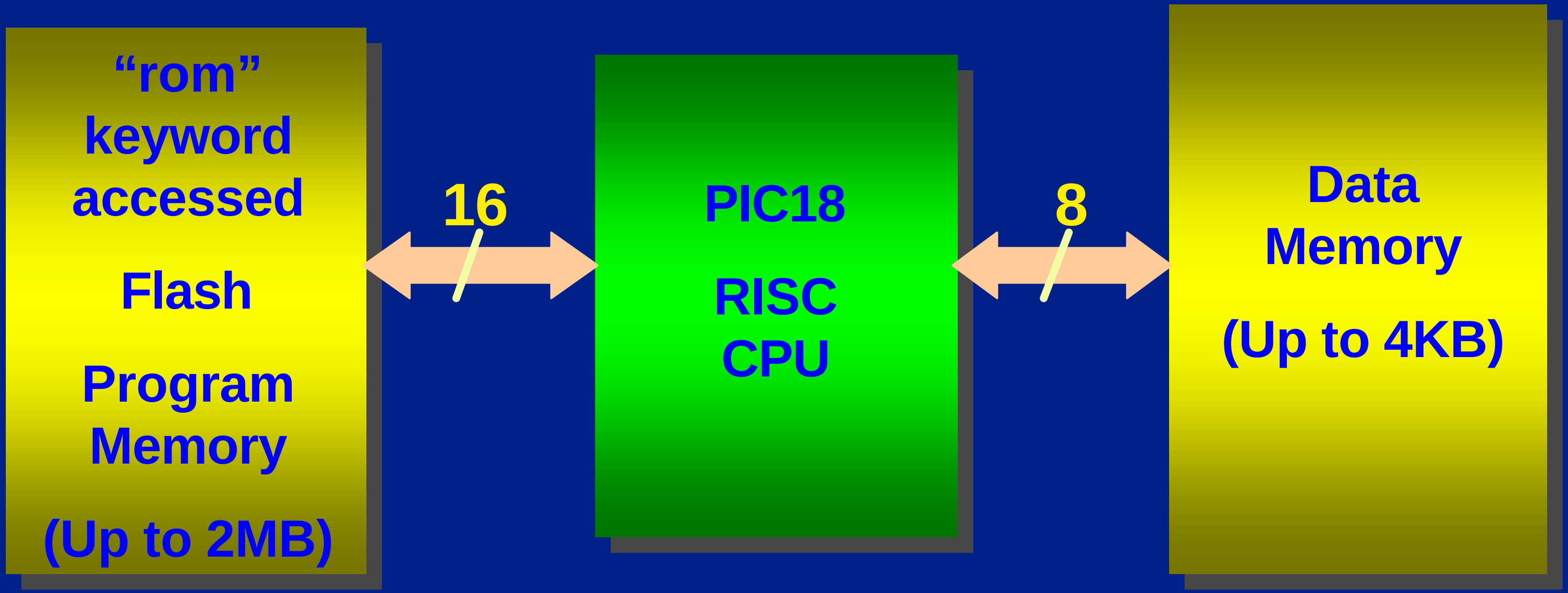
- High Performance 8-bit RISC CPU
- 40 MHz / 10 MIPs sustained operation
- 2.0V to 5.5V operation
- Linear Program Memory addressing to 2MB
- Linear Data Memory addressing to 4KB
- 3 Data Pointers with 5 addressing modes
- Relative conditional branch instructions

PIC18 Architecture
Features (Continued)

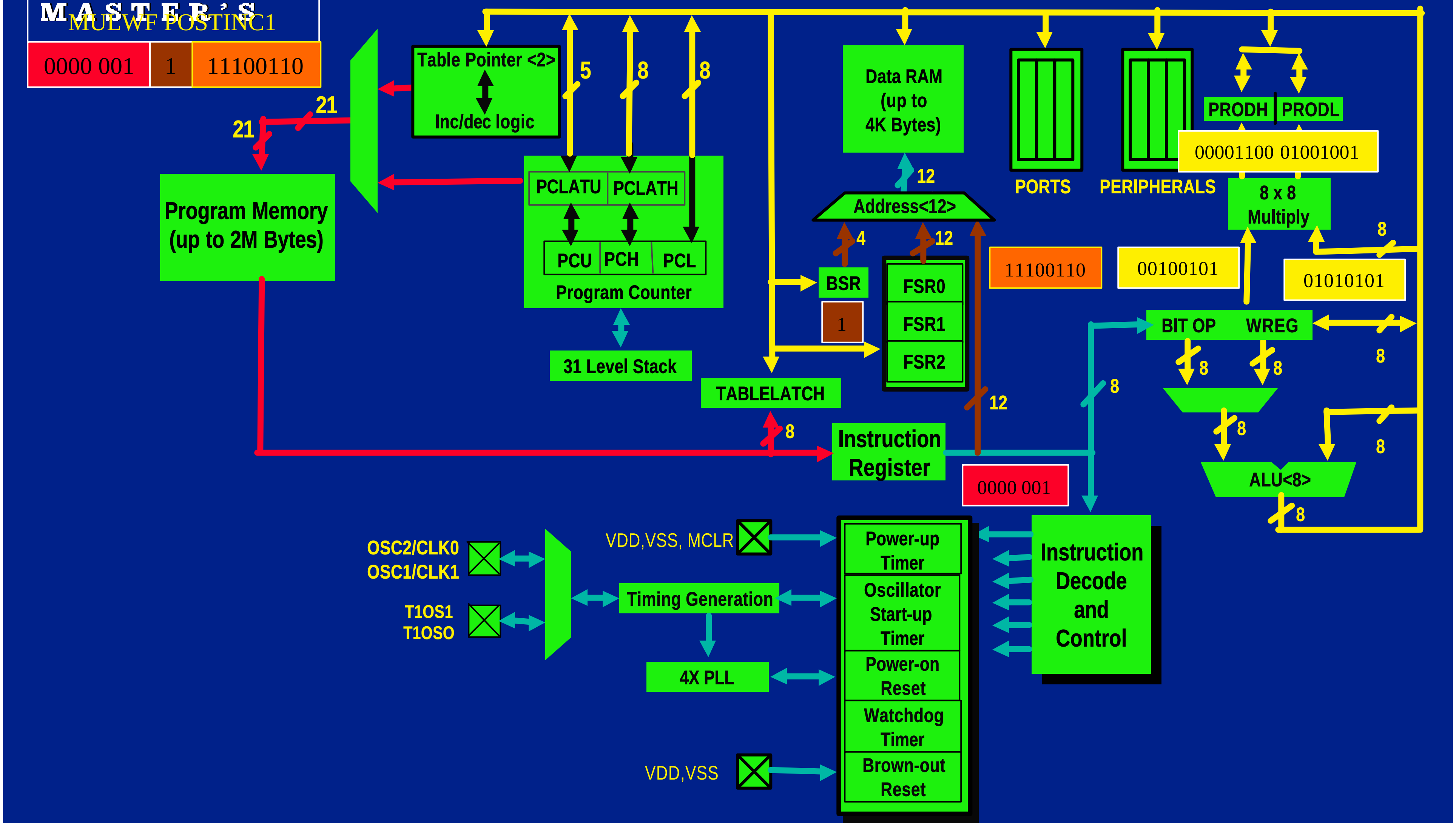
- Up to 10MIPS @ 10MHz with 4X PLL
- Enhanced Flash memory
  - 2 Seconds Programming Time
  - Low Cost MPLAB-ICD-II Support
  - Flexible Program Memory Protection
- And Many More...

PIC18 Architecture
Harvard Architecture

- Separate memory spaces for instructions and data
  - Increased throughput
  - Different program and data bus widths are possible



PIC18 Block Diagram



PIC18 Architecture

Oscillator

- Various oscillator modes

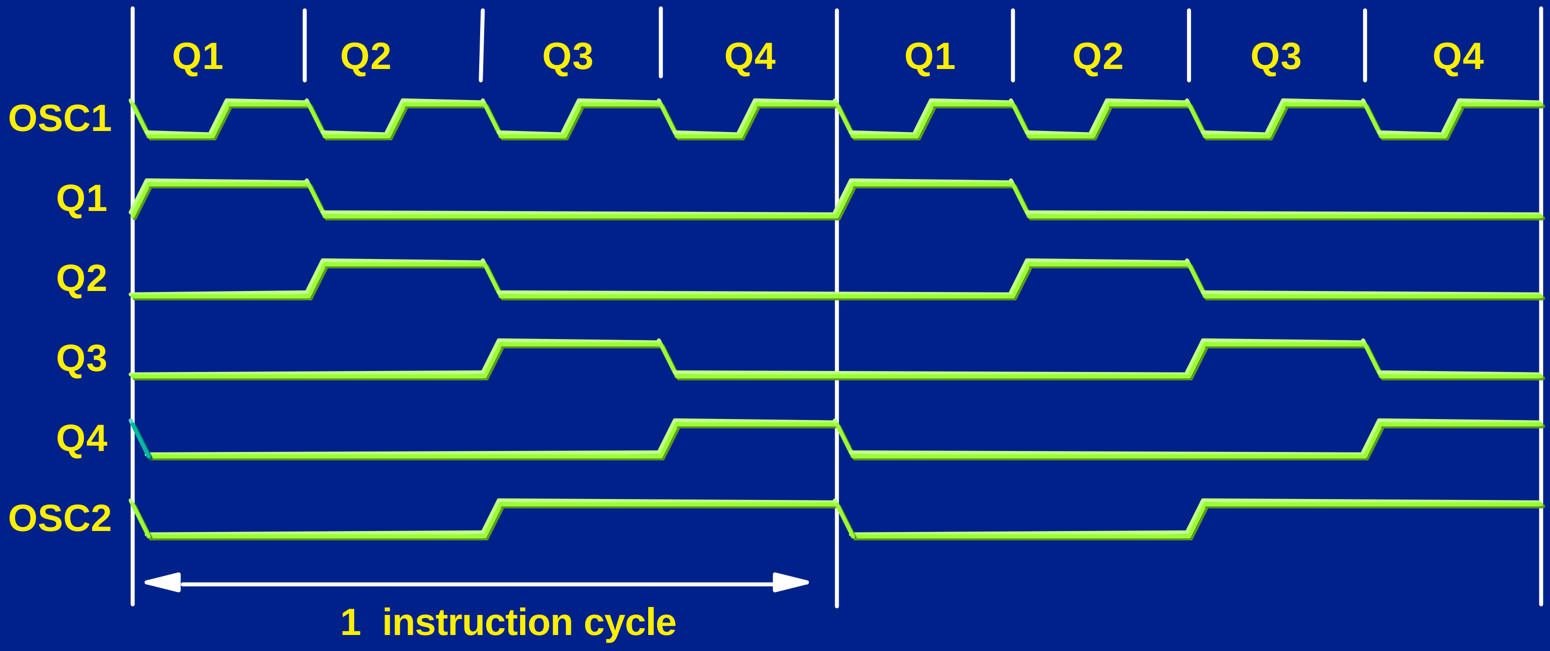
LP	Low Power Crystal (200KHz max)
XT	Crystal/Resonator (4MHz max)
HS	High Speed Crystal/Resonator (40MHz max)
HS + PLL	HS + 4X PLL (10MHz max)
RC	External RC (4MHz max)
RCIO	RC with OSC2 as I/O (4MHz max)
EC	External Clock (40MHz max)
ECIO	EC with OSC2 as I/O (40MHz max)
INTOSC	Internal RC Oscillator (30/500 kHz, 1/4/8 MHz)

Secondary Oscillator Mode

Modes selected by Configuration registers

PIC18 Architecture
Clocking Scheme

- Instruction cycle = 1/4 of clock input frequency
- 100 ns Instruction cycle at 40 MHz clock



PIC18 Architecture

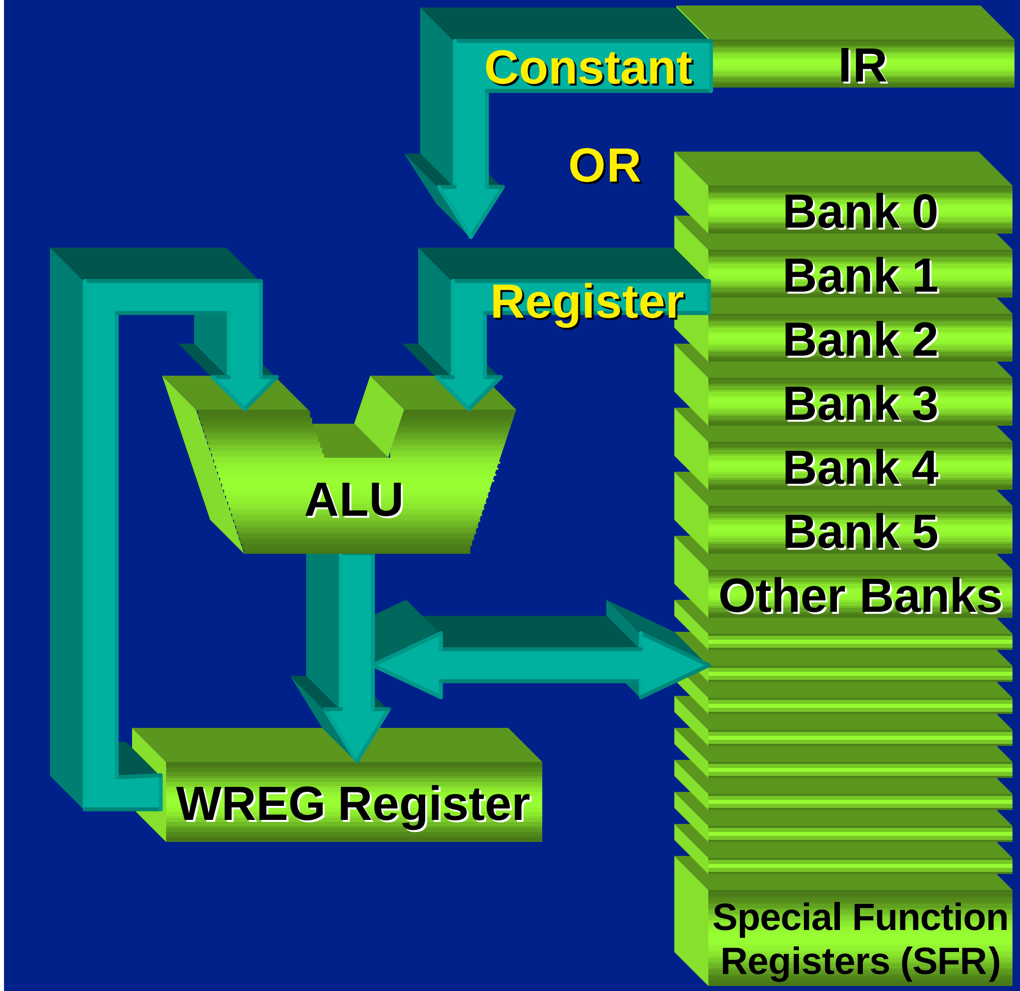
Instruction Pipeline

- Allows overlap of fetch and execution
- Makes single cycle execution
- Program branches (e.g. **GOTO**, **CALL** or Write to PC) take two or three cycles

	T_{CY0}	T_{CY1}	T_{CY2}	T_{CY3}	T_{CY4}
1. MOVWF PORTB	Fetch 1	Exec. 1			
2. RCALL SUB_1		Fetch 2	Exec. 2		
2b. *Forced NOP*			Fetch 2b	Forced NOP	
3. BSF PORTA, RA3				Fetch SUB_1	Exec. SUB_1
					Fetch SUB_1+1

PIC18 Architecture

ALU



- Operates on WREG and a Register or Constant
- Multi-Byte calculation using **ADDWFC** etc.

PIC18 Architecture

8 x 8 Hardware Multiplier

- Single Cycle Hardware Multiplier
- Performs
  - WREG X Register
  - WREG X Constant
- 16-bit result stored in PRODH:PRODL
- Integer arithmetic operation
- Unsigned operation

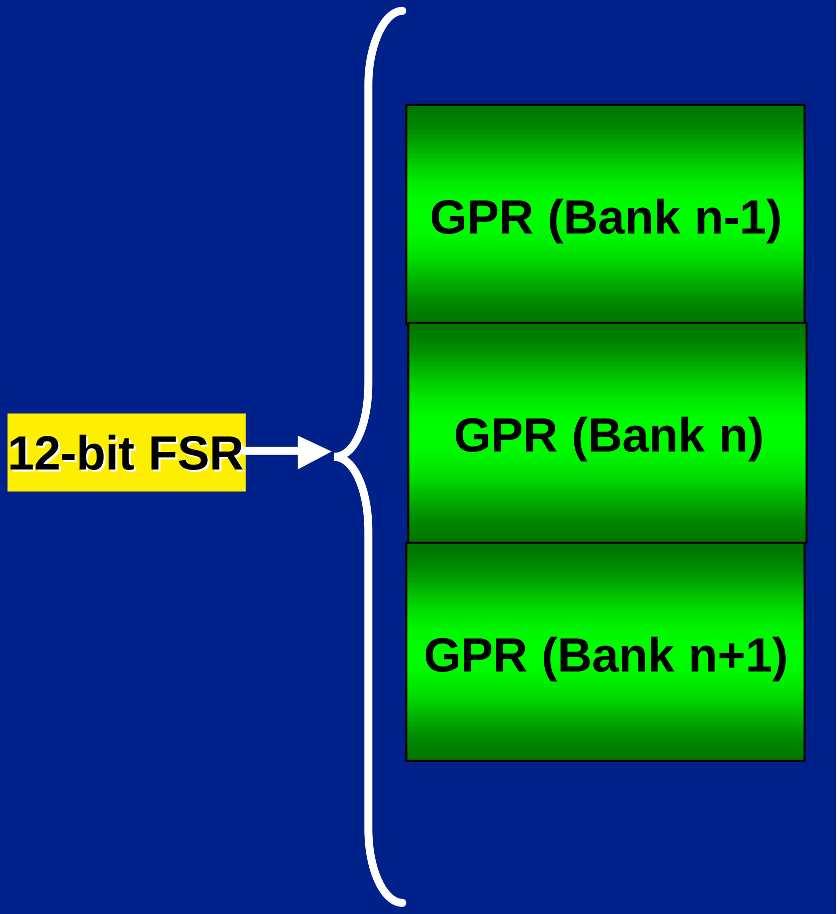
PIC18 Architecture
Computation Performance

Function	Prog Words (estimated)	RAM (estimated)	Max Time (uS) @ 10MIPS
8 x 8 unsigned multiply	1	-	0.1
16 X 16 unsigned multiply	30	7	3
16 X 16 signed multiply	40	8	4
32 x 32 signed multiply	140	18	15
32 / 16 signed divide	450	9	42
Float Add (IEEE 32bit)	320	12	7
Float Mul (IEEE 32bit)	350	13	10
Float Div (IEEE 32bit)	130	14	32
Sqrt (32bit)	320	10	57
Sin (32bit)	420	11	241

PIC18 Architecture

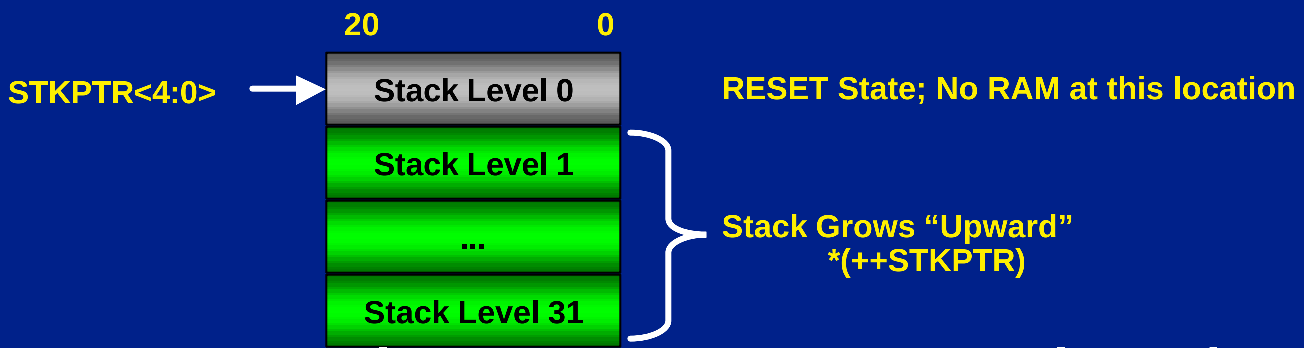
Indirect Access

- Indirect Addressing
  - Three 12-bit FSRs
  - FSRnH:FSRnL (0 ≤ n ≤ 2)
- Linear access to 4KB
- Special Instruction to load FSRn in 2 cycles
- De-reference operations
  - Unchanged
  - Pre/Post Increment
  - Post Decrement
  - Indexed by WREG (signed)



PIC18 Architecture
Stack Memory

- Hardware stack - 31 levels deep
  - Separate memory, pointed by STKPTR
  - Used by **`CALL`**, **`RCALL`**, INT, **`RETURN`**, **`RETFIE`**

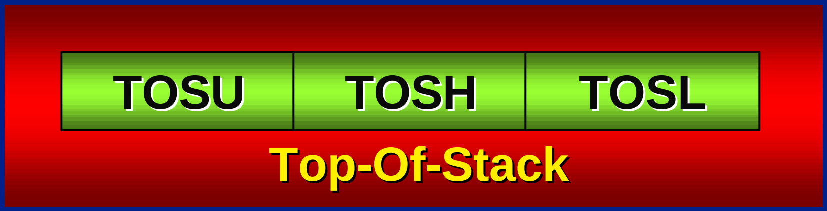


- Software stack uses FSRn, not hardware stack
  - Uses general purpose RAM, pointed by FSRn
  - Used to store local variables for re-entrant functions

PIC18 Architecture

Accessing HW Stack

- 5-bit Stack Ptr addresses 21-bit wide stack
- Top-Of-Stack = TOSU:TOSH:TOSL
  - Readable & Writeable => RTOS Friendly
- **PUSH** puts current PC on Top-Of-Stack
- **POP** discards Top-Of-Stack
- When enabled, Stack OV resets the device
- Stack Underflow returns 00000h



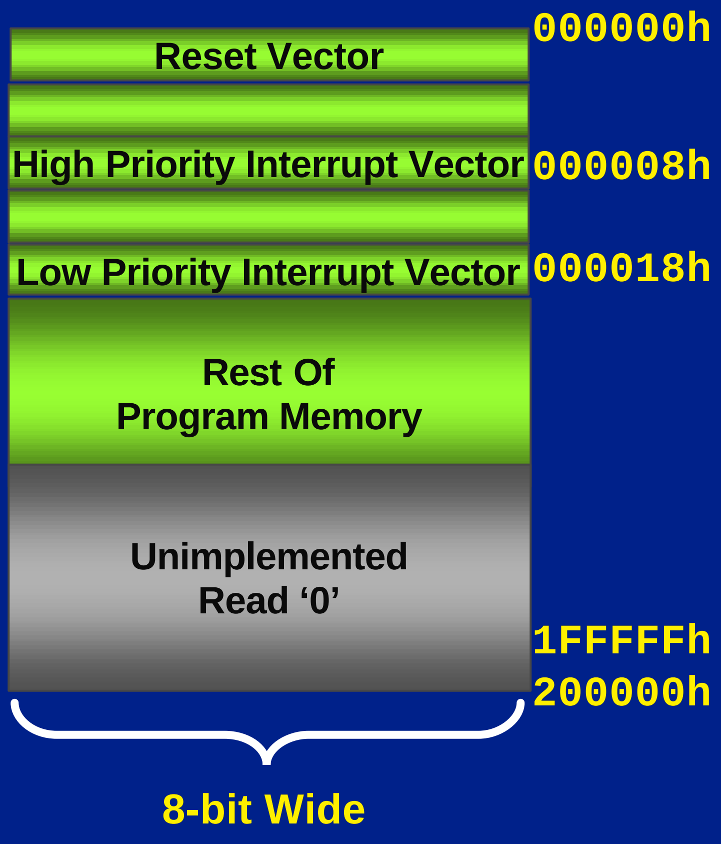
TOSU | TOSH | TOSL

Top-Of-Stack

PIC18 Architecture

Program Memory

- Up to 2M x 8 in size*
- Linear access
- Two Interrupt Vectors
- Self programmable*
- Programmable over entire voltage range
- Flexible Code Protection Modes*
- 100 K erase/writes (typical)*
- > 40 years retention (typical)



000000h

Reset Vector

High Priority Interrupt Vector 000008h

Low Priority Interrupt Vector 000018h

Rest Of
Program Memory

Unimplemented
Read ‘0’

1FFFFFh

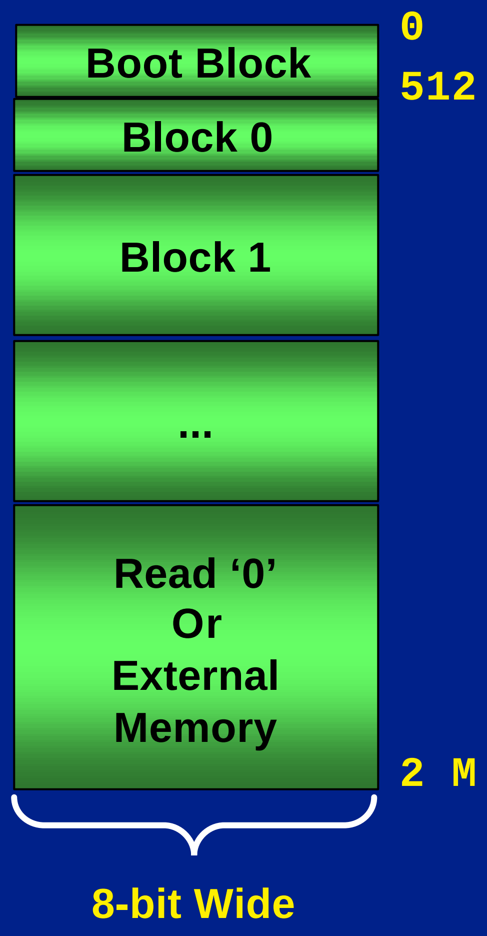
200000h

8-bit Wide

* Note: Check your device datasheet

PIC18 Architecture
Program Memory Organization

- Divided into blocks
- 512 bytes of Boot block*
- Block size varies by device
  - 8KB on PIC18F452
- Blocks erased in bulk or 64* bytes
  - Bulk erase in ICSP™ programming mode (4.5 - 5.5V)
- Code protection by block
- Internal Read/Write protection by block

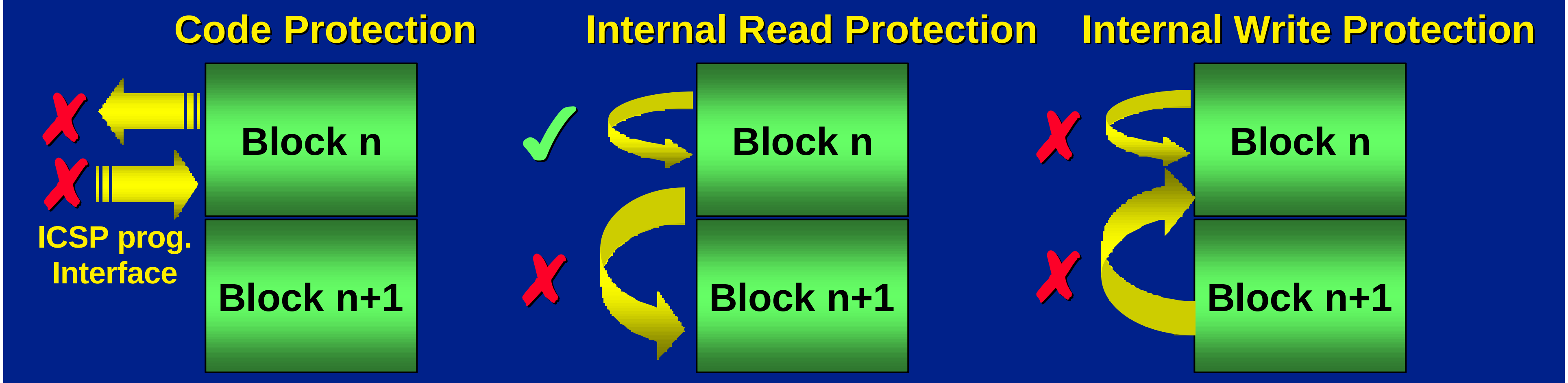


* Note: Check your device datasheet

PIC18 Architecture

Program Memory : Protection

Three types of Protection Scheme:



Code Protection

Block n

Block n+1

ICSP prog. Interface

Internal Read Protection

Block n

Block n+1

Internal Write Protection

Block n

Block n+1

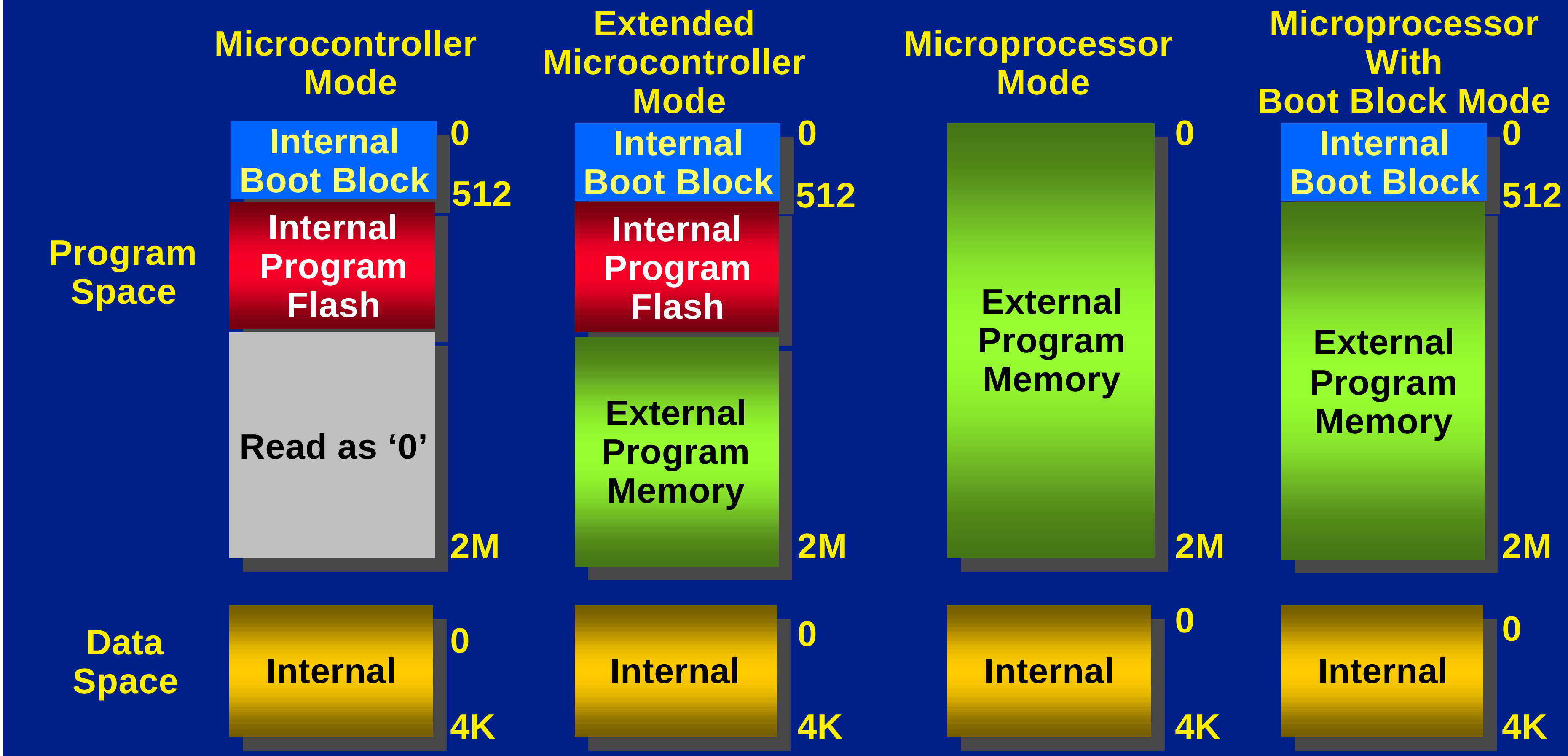
ICSP programming mode
Read and Write disabled

Reads from same block OK,
reads from other blocks disabled

Self Write to this block are disabled

PIC18 Architecture
Program Memory Modes

Four Modes:



Microcontroller Mode

Extended Microcontroller Mode

Microprocessor Mode

Microprocessor With Boot Block Mode

Program Space

Internal Boot Block

Internal Program Flash

Read as ‘0’

External Program Memory

0

512

2M

Data Space

Internal

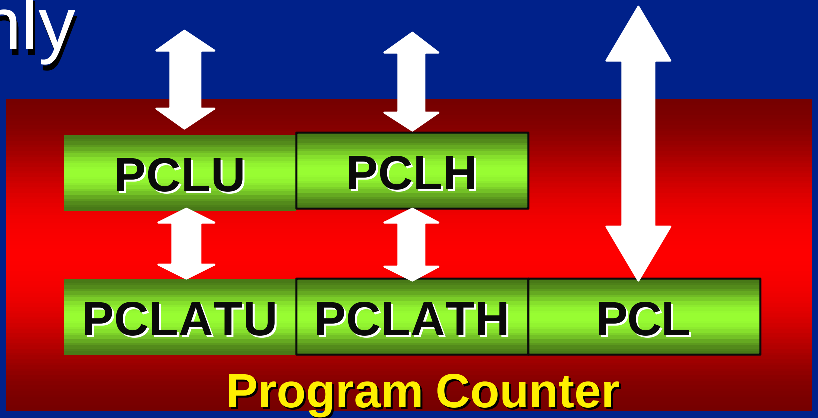
0

4K

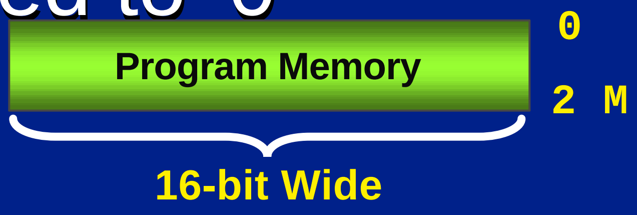
Note: Check your device datasheet

PIC18 Architecture
Accessing Program Memory

- 21-bit Divided into PCU:PCH:PCL
  - PCL is readable/writeable
  - PCU:PCH is readable/writeable via shadow registers only



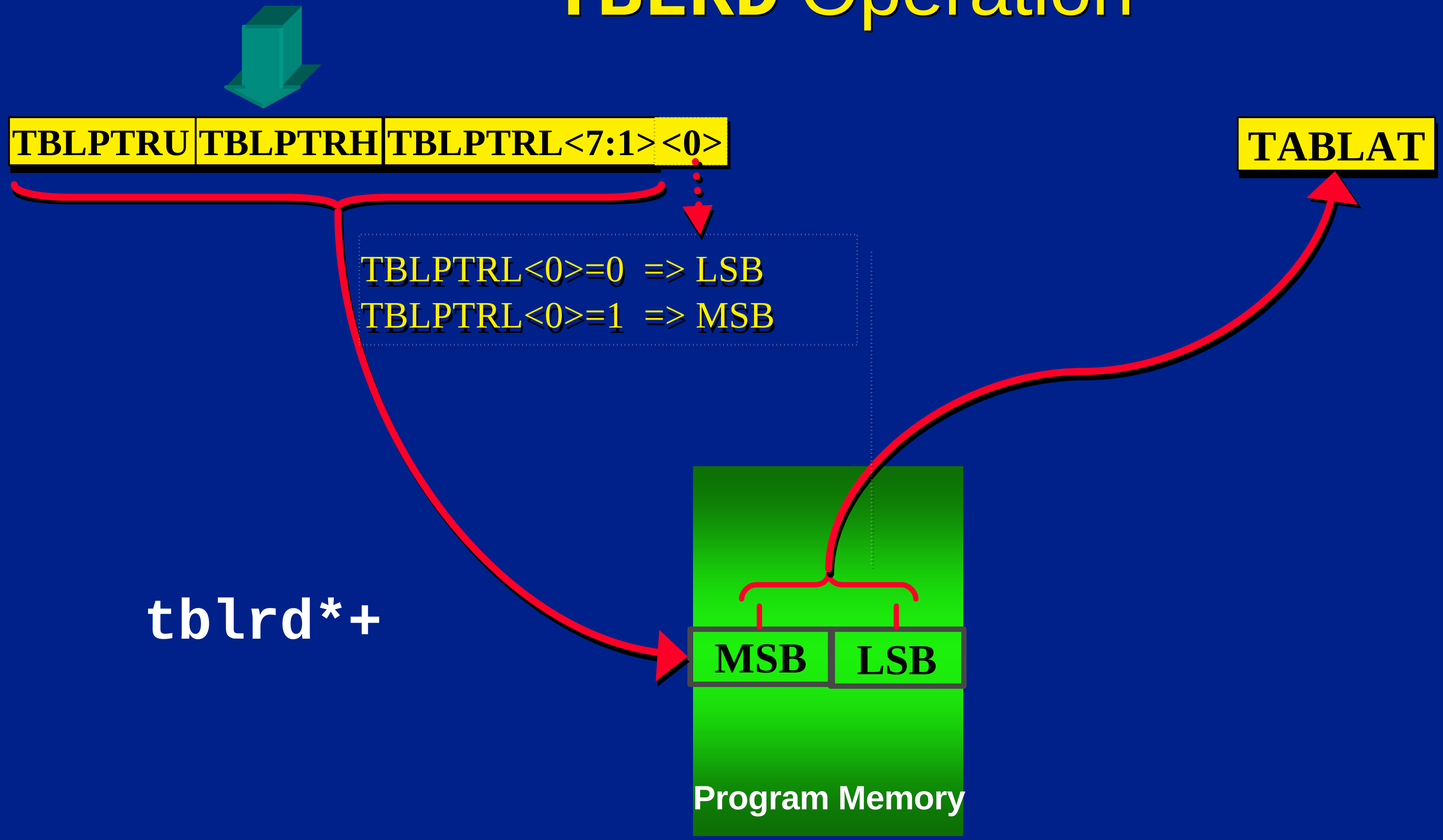
- PCL<0> is forced to ‘0’



PIC18 Architecture

Reading Program Memory

TBLRD Operation

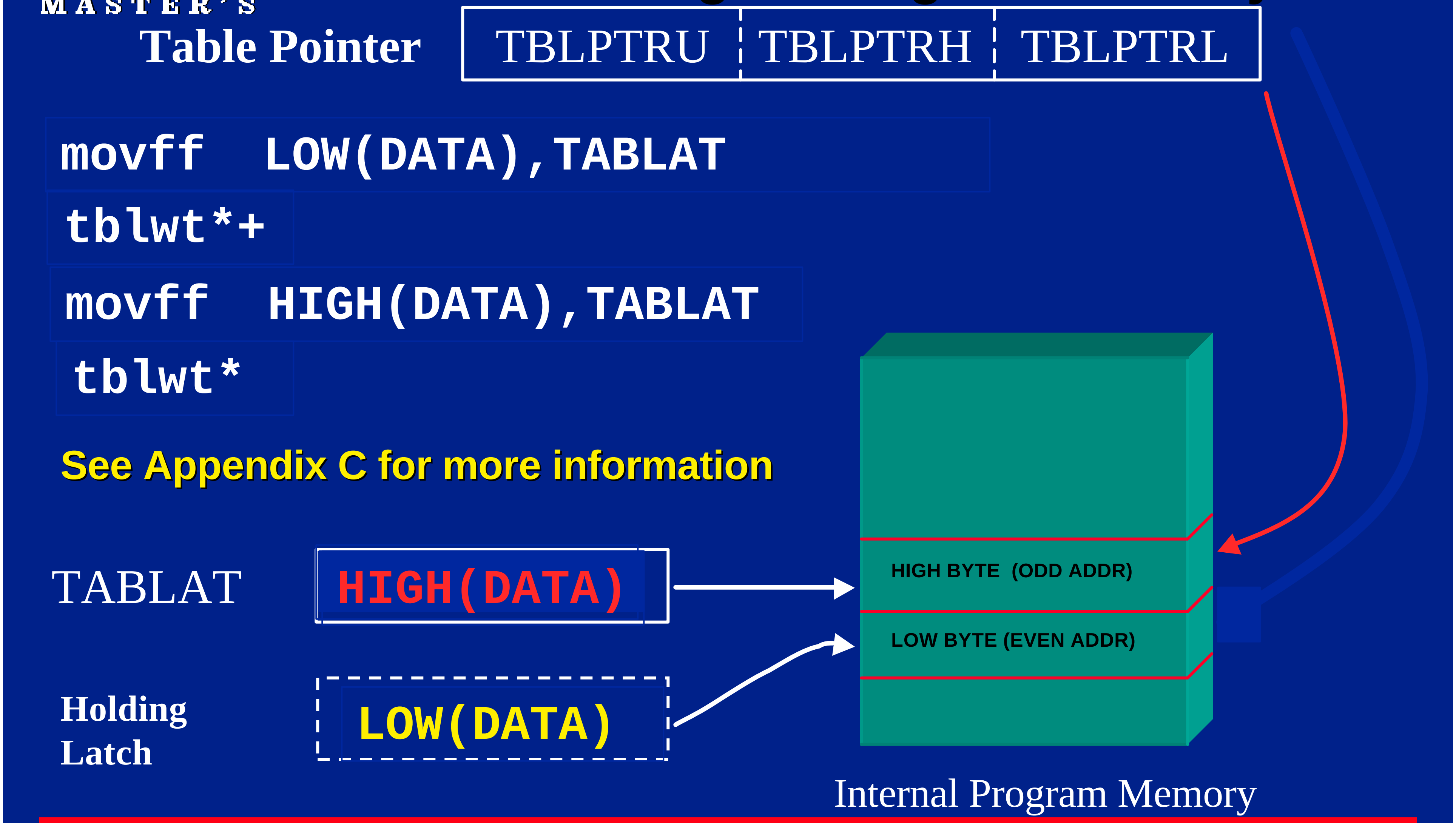


TBLPTRL<0>=0 => LSB
TBLPTRL<0>=1 => MSB

```
tblrd*+
```

PIC18 Architecture

Writing to Program Memory



```
movff  LOW(DATA),TABLAT
tblwt*+
movff  HIGH(DATA),TABLAT
tblwt*
```

See Appendix C for more information

PIC18 Architecture

Accessing Program Memory (Cont.)

- **TBLPTR** is used to address program memory
  - Divided in TBLPTRU:TRBLPTRH:TBLPTRL
- **TBLRD** is used to read a byte
- **TBLWT** is used to load write buffer
  - EECON1 register controls actual write cycle
  - Protected against “run-away” code
- Erase block size 32 or 64 bytes*
- 8 bytes written at a time

* Note: Check your device datasheet

Table Pointer Operations

- To enhance flexibility of table operations, the TBLPTR automatically increment and decrement during read/write operations
- PIC18 devices have 4 modify modes for TBLPTR

`tblwt*`	`tblrd*`	no change
`tblwt*+`	`tblrd*+`	auto post increment
`tblwt*-`	`tblrd*-`	auto post decrement
`tblwt+*`	`tblrd+*`	auto pre increment

PIC18 Architecture

Data EEPROM

- Size ranges from 64 to 1024 bytes
- 1 M erase/write cycles (typical)
- > 40 years retention (typical)
- Read and Written at byte boundary
  - Automatic Erase-Before-Write
- Protection against “run-away” code
- Code Protection And Internal Write Protection
- Accessed via EEADR, EEDATA and EECONn registers

PIC18 Architecture

Configuration

- Configuration Registers at **300000h**
- Bit(s) enable/define mode(s)
- Written one byte at a time
- Writeable in all modes
  - Special “Configuration Write Protect” bit
- Most bits can be written to either ‘**1**’ or ‘**0**’
  - Code, Read and Write Protection bits can be written ‘**1**’ -> ‘**0**’ only
  - Bulk Erase required to reset Code, Read and Write Protection bits to a ‘**1**’

Specifying Configuration Information in Source File

- Create “config.asm” file and include in project:

```
#include p18f452.inc
__CONFIG _CONFIG1L,0xFF
__CONFIG _CONFIG1H,_OSCS_OFF_1H&_HSPLL_OSC_1H
__CONFIG _CONFIG2L,_BOR_OFF_2L&_BORV_20_2L&_PWRT_OFF_2L
__CONFIG _CONFIG2H,_WDT_OFF_2H&_WDTPS_128_2H
__CONFIG _CONFIG3L,0xFF
__CONFIG _CONFIG3H,_CCP2MX_OFF_3H
__CONFIG _CONFIG4L,_STVR_ON_4L&_LVP_OFF_4L&_DEBUG_OFF_4L
__CONFIG _CONFIG4H,0xFF
__CONFIG _CONFIG5L,_CP0_OFF_5L&_CP1_OFF_5L&_CP2_OFF_5L&_CP3_OFF_5L
__CONFIG _CONFIG5H,_CPB_OFF_5H&_CPD_OFF_5H
__CONFIG _CONFIG6L,_WRT0_OFF_6L&_WRT1_OFF_6L&_WRT2_OFF_6L&_WRT3_OFF_6L
__CONFIG _CONFIG6H,_WRTC_OFF_6H&_WRTB_OFF_6H&_WRTD_OFF_6H
__CONFIG _CONFIG7L,_EBTR0_OFF_7L&_EBTR1_OFF_7L&_EBTR2_OFF_7L&_EBTR3_OFF_
__CONFIG _CONFIG7H,_EBTRB_OFF_7H
END
```

C Programmer's Interface

Accessing Peripheral Control and Status Bits

- All peripheral control bits set up in <processor>.h file as:

<peripheral register name>bits.<bit name>

- Example:
  - GIEH bit of INTCON can be accessed by:

INTCONbits.GIEH

Reset Vector

- Located at 0x00000, compiler automatically initializes variables
- Calls main() after variable initialization
- Loops back and calls main() again if main exits
- Generally, main() should stay in loop and not exit:

```
void main(void){
     // Place your initialization code here

  while(1){
          // Place your main loop here
          }
  }
```

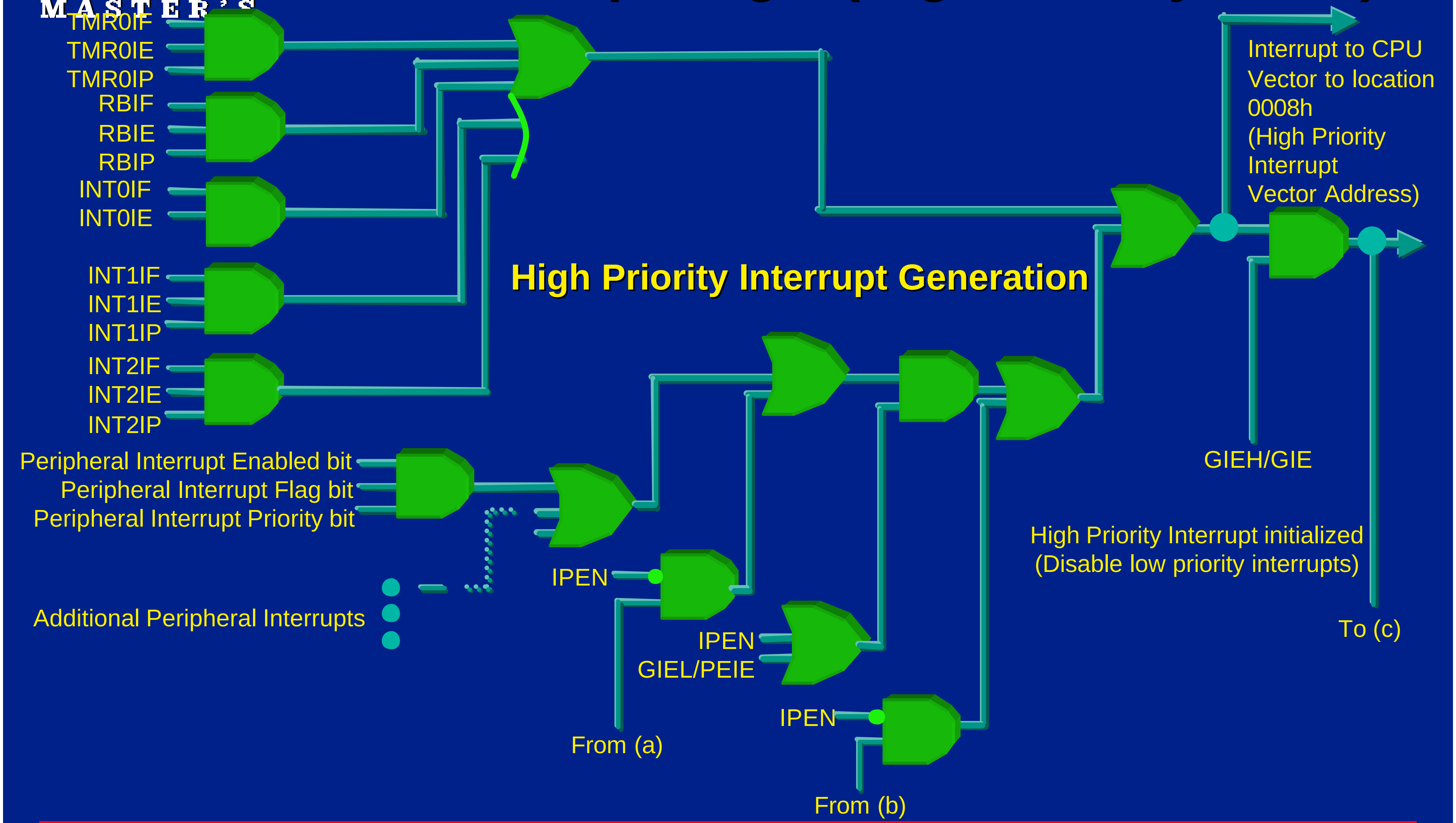
PIC18 Architecture

Interrupt Overview

- Interrupt Sources can individually
  - Assigned to high or low priority vector
    - High Priority Vector at **000008h** (Default)
    - Low Priority Vector at **000018h**
  - Polled or interrupt driven
- Automatic context save WREG, STATUS and BSR on High Priority Interrupt
- Most interrupts wake processor from sleep
- Fixed interrupt latency is three instruction cycles

PIC18 Architecture

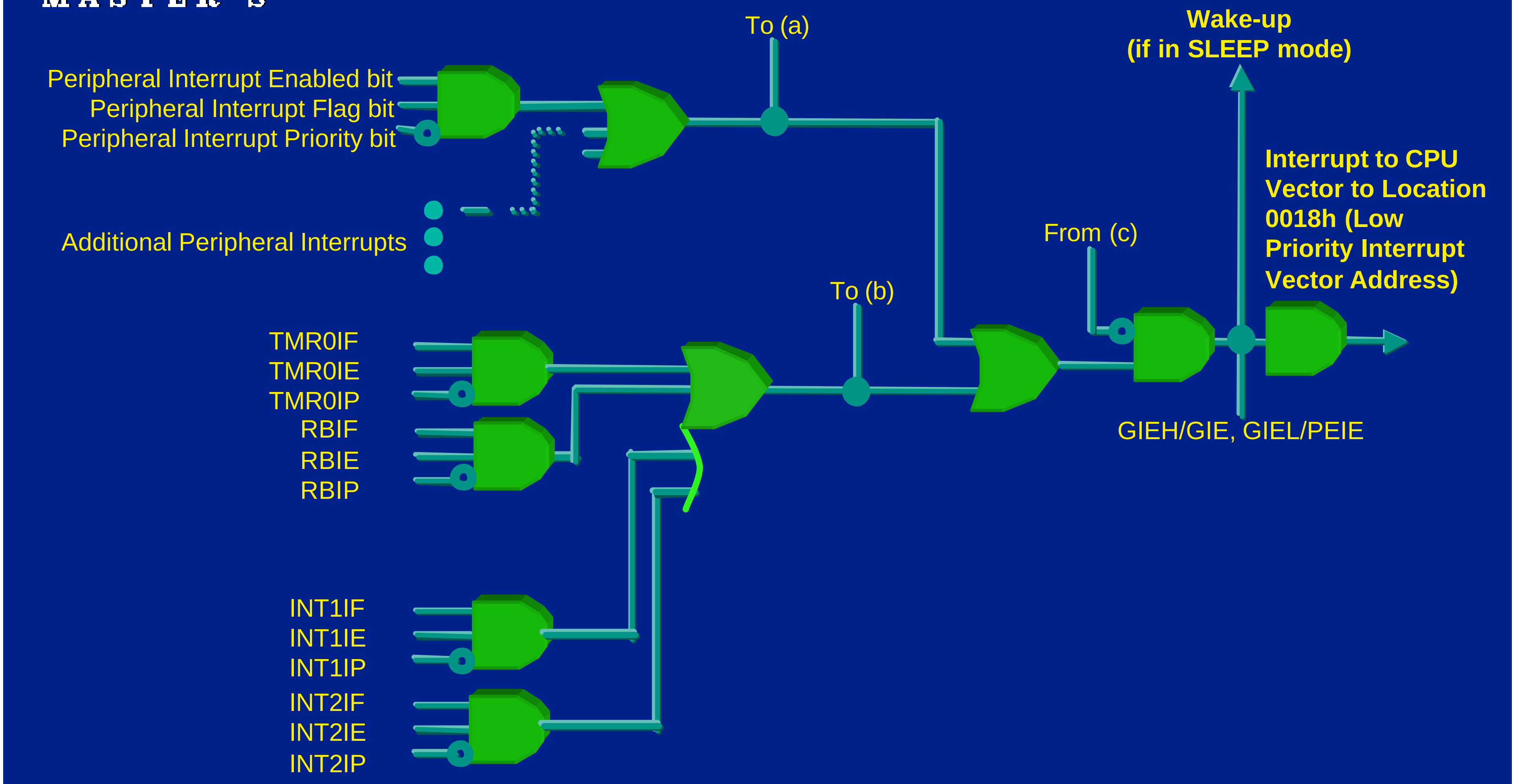
Interrupt Logic (High Priority Level)



TMR0IF
TMR0IE
TMR0IP
RBIF
RBIE
RBIP
INT0IF
INT0IE
INT1IF
INT1IE
INT1IP
INT2IF
INT2IE
INT2IP
Peripheral Interrupt Enabled bit
Peripheral Interrupt Flag bit
Peripheral Interrupt Priority bit
Additional Peripheral Interrupts
High Priority Interrupt Generation
Interrupt to CPU
Vector to location
0008h
(High Priority
Interrupt
Vector Address)
GIEH/GIE
High Priority Interrupt initialized
(Disable low priority interrupts)
To (c)
IPEN
IPEN
GIEL/PEIE
IPEN
From (a)
From (b)

PIC18 Architecture

Interrupt Logic (Low Priority Level)



Interrupt Priority Enable

- New bit added to the RCON register - IPEN

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	-	RI	TO	PD	POR	BOR
bit7	6	5	4	3	2	1	0

- Enables / Disables Interrupt Priority and 16C Compatibility
  - If IPEN=0, priority is disabled and the interrupts are compatible with 16C (default)
  - If IPEN=1, priority is enabled and the interrupts are NOT compatible with 16C
- Registers have been added to set priority for each interrupt source, except INT0.

Peripheral Interrupt Control Registers

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
	bit7	6	5	4	3	2	1	0

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
	bit7	6	5	4	3	2	1	0

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
	bit7	6	5	4	3	2	1	0

	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
PIR2	-	-	-	-	BCLIF	LVDIF	TMR3IF	CCP2IF
	bit7	6	5	4	3	2	1	0

	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
PIE2	-	-	-	-	BCLIE	LVDIE	TMR3IE	CCP2IE
	bit7	6	5	4	3	2	1	0

	U-0	U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1
IPR2	-	-	-	-	BCLIP	LVDIP	TMR3IP	CCP2IP
	bit7	6	5	4	3	2	1	0

GIE PEIE In Compatibility Mode

- When IPEN=0 Compatibility Mode
  - INTCON<7> is GIE
  - INTCON<6> is PEIE
  - Note: definition exactly same as 16C INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE/GIEH	**PEIE**/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF
bit7	6	5	4	3	2	1	

GIEH & GIEL In Priority Mode

- When IPEN=1 Priority Interrupt Mode

- INTCON<7> is GIEH
- INTCON<6> is GIEL

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE/**GIEH**	PEIE/**GIEL**	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF
bit7	6	5	4	3	2	1	

- High Priority Interrupt Enable GIEH replaces GIE
- Low Priority Interrupt Enable GIEL replaces PEIE

High Priority Interrupts

- High Priority Vector uses shadow registers for automatic context save / restore:

```
#pragma code HighVector=0x8
void HighVector (void)
   { _asm GOTO high_priority_interrupt _endasm}

#pragma code // return to default code section

#pragma interrupt high_priority_interrupt save=[symbol]
void high_priority_interrupt (void){
   // Place your high priority interrupt code here
}
```

Low Priority Interrupts

- Low Priority Vector - compiler saves context and restores it with “interruptlow” pragma

```
#pragma code lowVector=0x18
void LowVector (void)
{
_asm GOTO low_priority_interrupt _endasm
}
#pragma code

#pragma interruptlow low_priority_interrupt save=[symbol]
void low_priority_interrupt (void){
   // Place your low priority interrupt code here
}
```

Interrupt Context Save / Restore

- High priority interrupt uses Hardware shadow registers to save and restore WREG,BSR,STATUS.
- Low priority interrupt uses the software stack to manually save WREG,BSR,STATUS.
- You need to add save=[symbol or section] if your ISR is complicated by:
  - Accessing a calculated index within an array
  - Calls other user functions
  - Performs complex math (*,/,float)
  - Accesses a ROM qualified variable

Guidelines for ISR Save Context

ISR Code Behavior	Symbol or Section added to ISR Save List
Call functions that are also called within main code paths	`section(".tmpdata"), PROD`
Access values in Program Memory such as an array declared with the ROM keyword	`TABLPTR, TABLAT`
Performs Multiplication or accesses a calculated index of an array	`PROD`
Executes Division, 16 bit or greater Multiplication, Floating Point, Scientific functions	`section("MATH_DATA")`

Example: ISR accesses a calculated array index and executes a division within the ISR:

```
#pragma interrupt sample_adc save=PROD, section("MATH_DATA")
```

Large Arrays and Structures

- Linker attempts to fit each variable into a default 256 byte section
- Need to create a larger protected section for arrays and structures larger than 256 bytes:
- Modify <processor name>.lkr file as follows:

```
DATABANK NAME=gpr2       START=0x200 END=0x2FF
DATABANK NAME=big_array1 START=0x300 END=0x4FF  PROTECTED
DATABANK NAME=gpr5       START=0x500 END=0x5FF
SECTION  NAME=big_array  RAM=big_array1
```

Large Arrays and Structures (cont.)

- Add #pragma to use new section in source.c

```
#pragma udata big_array // Select large section
   unsigned char test[456];
#pragma udata  // Return to normal section
```

- Access these large (>256 byte) arrays and structures through pointers or a variable based index (array[index] or *array)
    - Avoid fixed element addressing on these large arrays and structures (ex: array[2])
- Pointers are more code efficient than array indexing

Peripherals

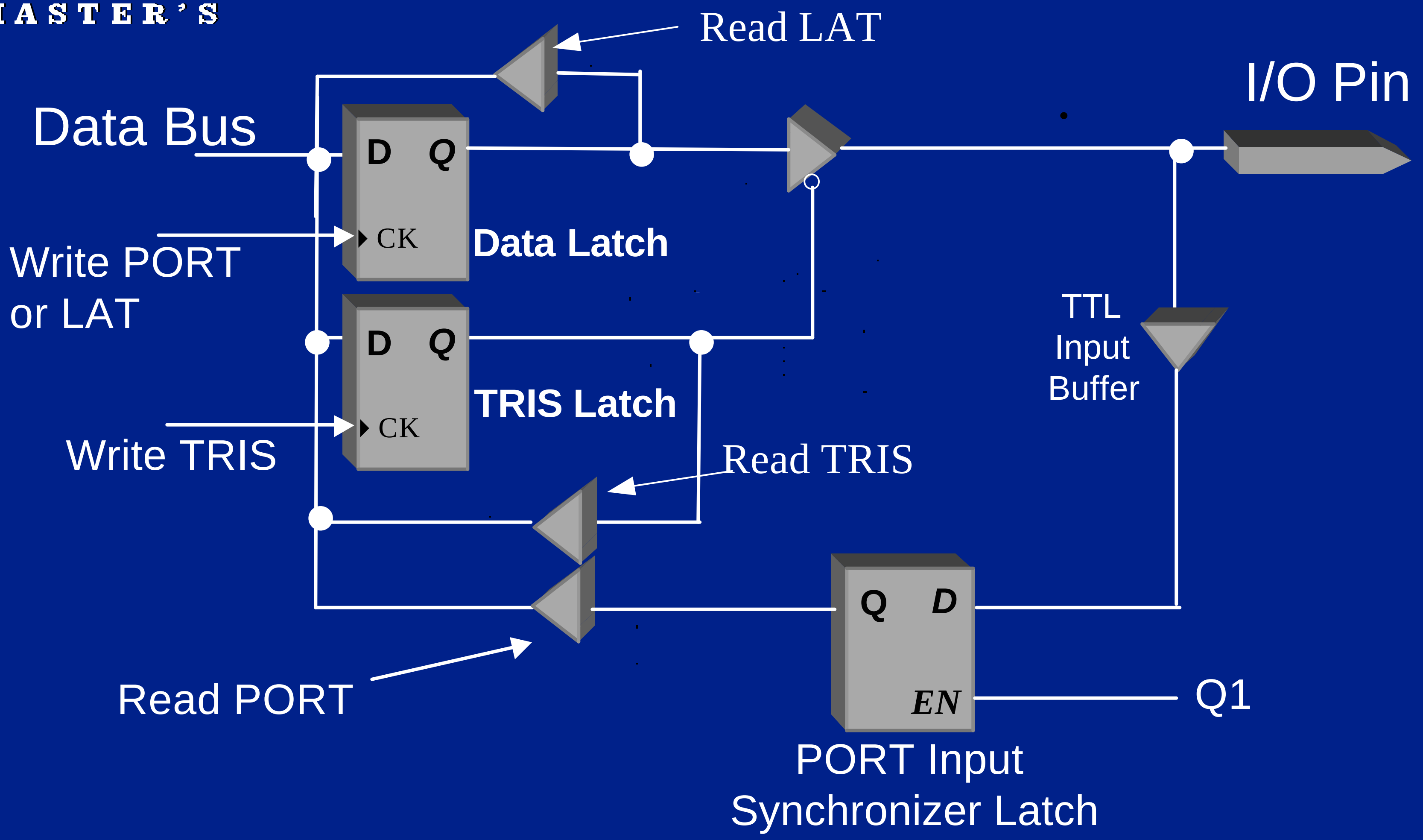
PIC18 Peripherals

- Digital I/O Ports
- Timer0, 1, 2, 3
- Compare/Capture/PWM (CCP)
- Analog-To-Digital Converter
- Analog Comparator
- Addressable USART (AUSART)
- Master Synchronous Serial Port (MSSP)
- External Memory Access (EMA)
- Controller Area Network (CAN)

PIC18 Peripherals
Digital I/O Ports

- Up to 68 bi-directional I/O pins
- High sink/source capability (up to 25mA)
- Direct bit (pin) manipulation (single-cycle)
- Each port pin has:
  - Individual direction control (TRISA~TRISJ)
  - Data Latch (LATA~LATJ - read-modify-writes)
  - Port Register (PORTA~PORTJ reads value on pins)
- All I/O pins have ESD protection

Port Latch Block Diagram



I/O pins have ESD protection diodes

I/O Pin Direction

- Direction of I/O pins controlled by individual TRIS bits
  - 1 = Input (default power on reset state)
  - 0 = Output
- Example

```
TRISAbits.TRISA5 = 0; // Make RA5 output
TRISB = 0b11110000;  // Make RB0:3 outputs,
                       // RB4:7 inputs
```

Reading / Writing I/O Ports

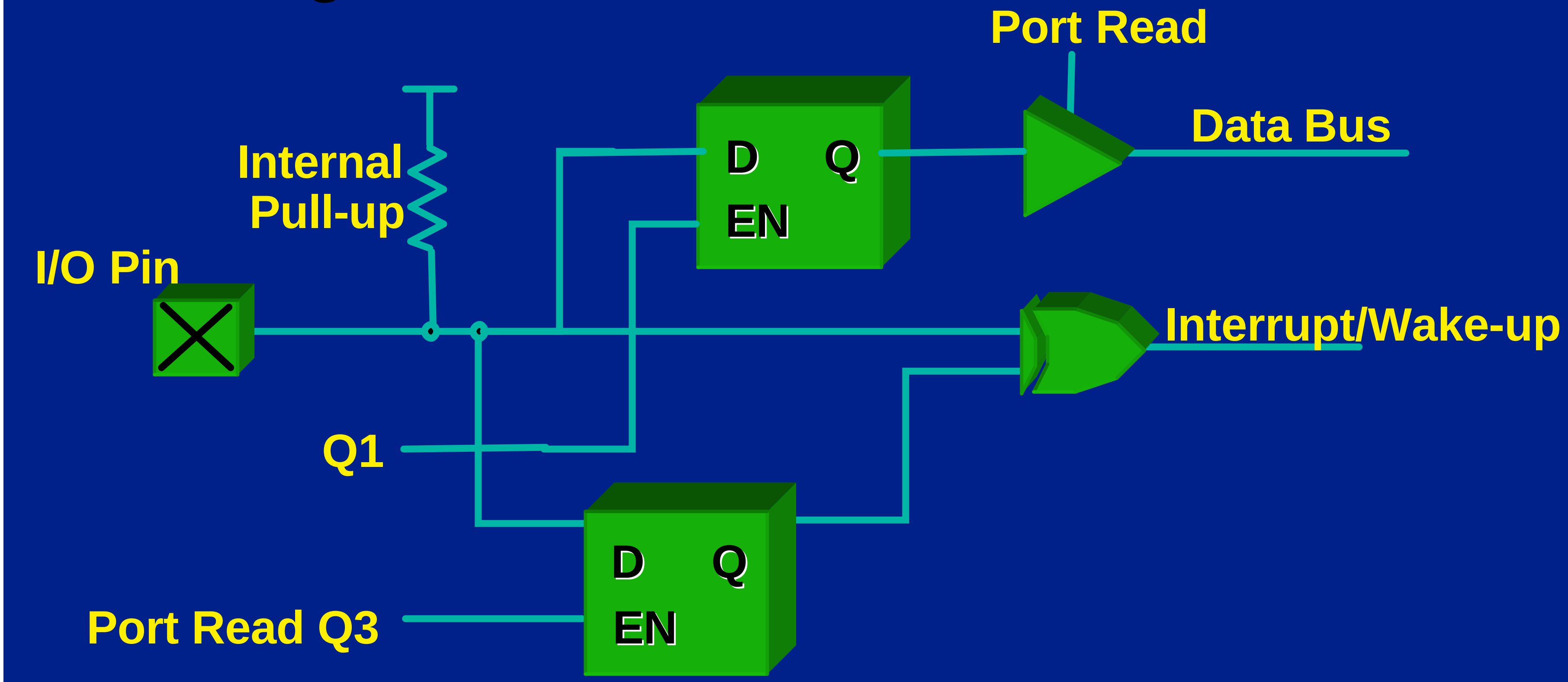
- Reading a I/O port or bit uses the PORT register
  - `if (PORTCbits.RC2) // Execute if RC2 = 1`
  - `if (PORTC == 0b11110000) // Check for F0`

- Writing to an I/O port or bit should use LAT register
  - `LATAbits.LATA0 = 1; // Set RA0`

  -
    ```
    LATB = 0xFF;   // Set all of PORTB output
                   // pins to a logic one
    ```

PIC18 Peripherals
PORTB : Interrupt on Change

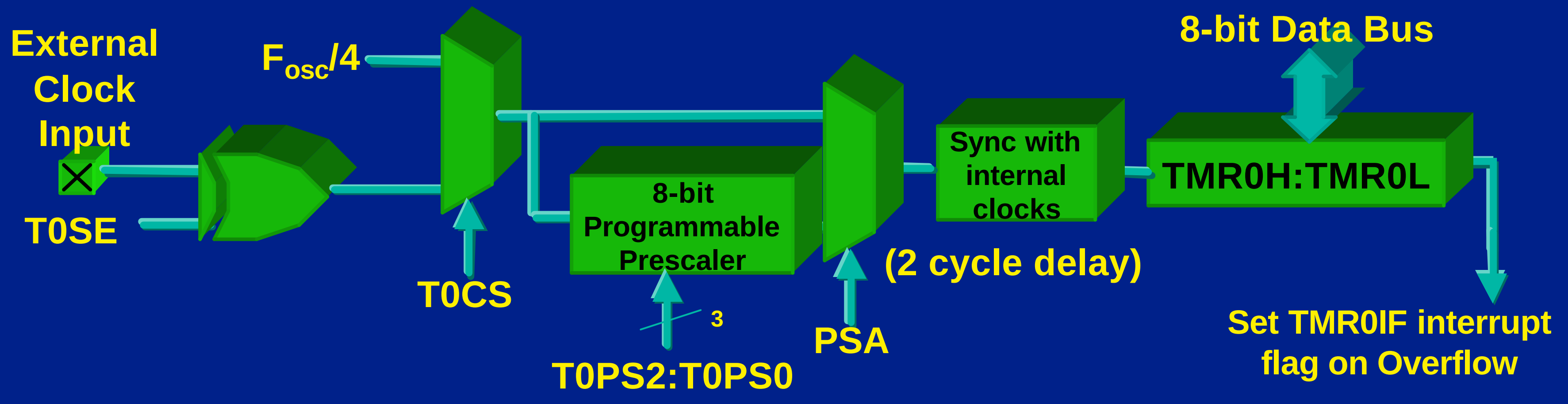
- Internal Pull-Ups and Wakeup/Interrupt On Change feature



PIC18 Peripherals

Timer0

- 8-bit/16-bit Timer/Counter
  - 16-bit Read and Writes
- 8-bit Software Programmable Prescaler
- Internal or External clock select
- Interrupt on overflow from **FFh/FFFFh** to **00h**



Timer 0 Setup

	bit 7							bit 0
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON	**Timer 0 On/Off Control** 1 = Enables Timer 0 0 = Stops Timer 0
T08BIT	**Timer 0 8-bit / 16-bit Select** 1 = Timer 0 configured for 8-bit mode 1 = Timer 0 configured for 16-bit mode
T0CS	**Timer 0 Clock Source Select** 1 = Transition on T0CKI pin (counter mode) 0 = Internal Instruction cycle (timer mode)
T0SE	**Timer 0 Source Edge Select** 1 = Increment on High -> Low T0CKI transition 0 = Increment on Low -> High T0CKI transition
PSA	**Timer 0 Prescaler Asignment** 1 = Timer 0 Prescaler is NOT assigned, prescaler bypassed 0 = Timer 0 Prescaler assigned and enabled
T0PS2:T0PS0	**Timer 0 Prescaler Selection** 111 = 1:256 011 = 1:16 110 = 1:128 010 = 1:8 101 = 1:64 001 = 1:4 100 = 1:32 000 = 1:2

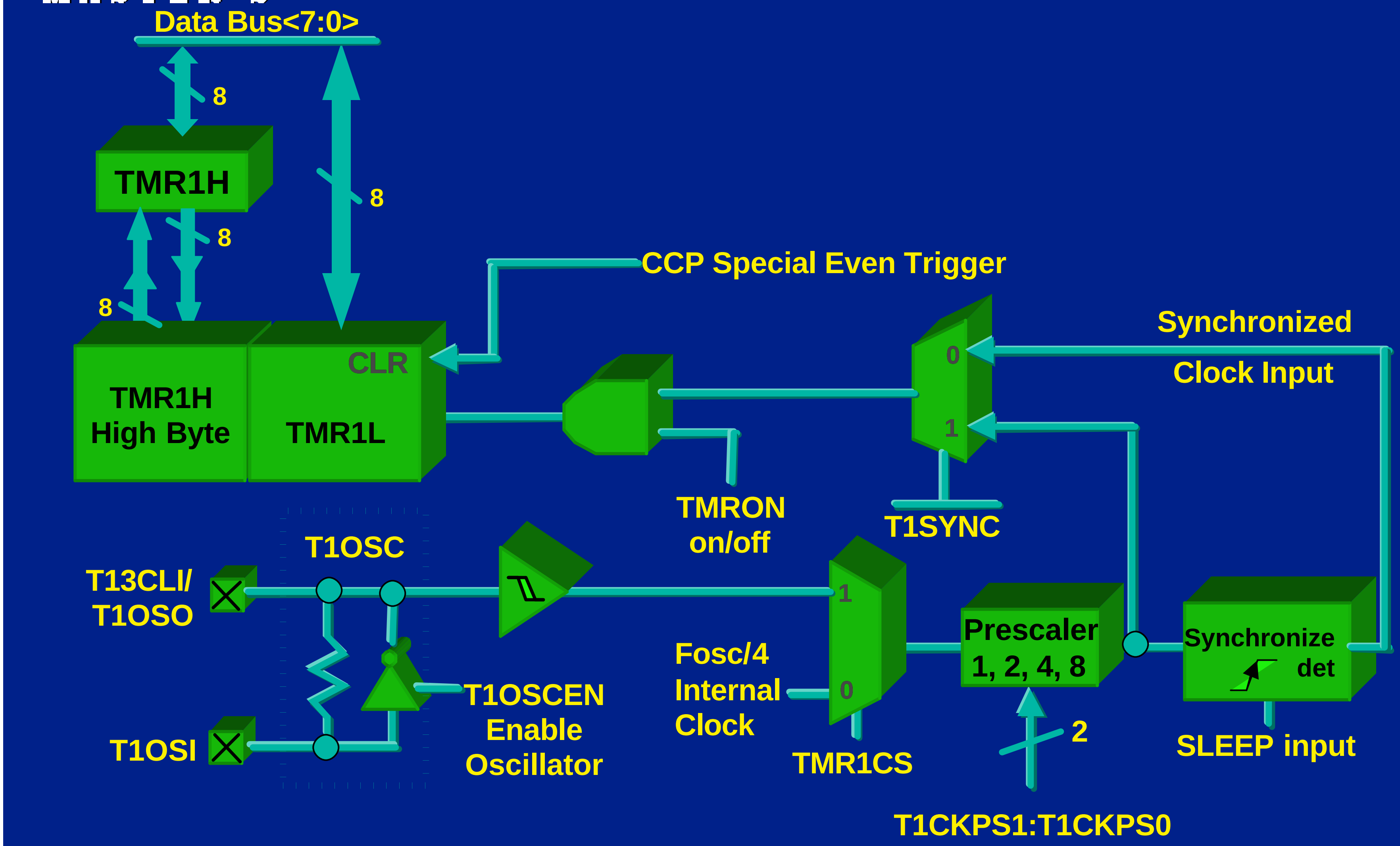
PIC18 Peripherals

Timer1 and Timer3

- 16-bit Timer / Counter
- Consists of two readable and writeable 8-bit registers
  - 16-bit Read / Write mode eliminates hazards
- ÷1, ÷2, ÷4, or ÷8 Prescaler
- Timer, Synchronous or Asynchronous Counter
- Timer1 can also operate from an external crystal with its built in oscillator feature.
- Interrupt on overflow from **FFFFh** to **`0000h`**

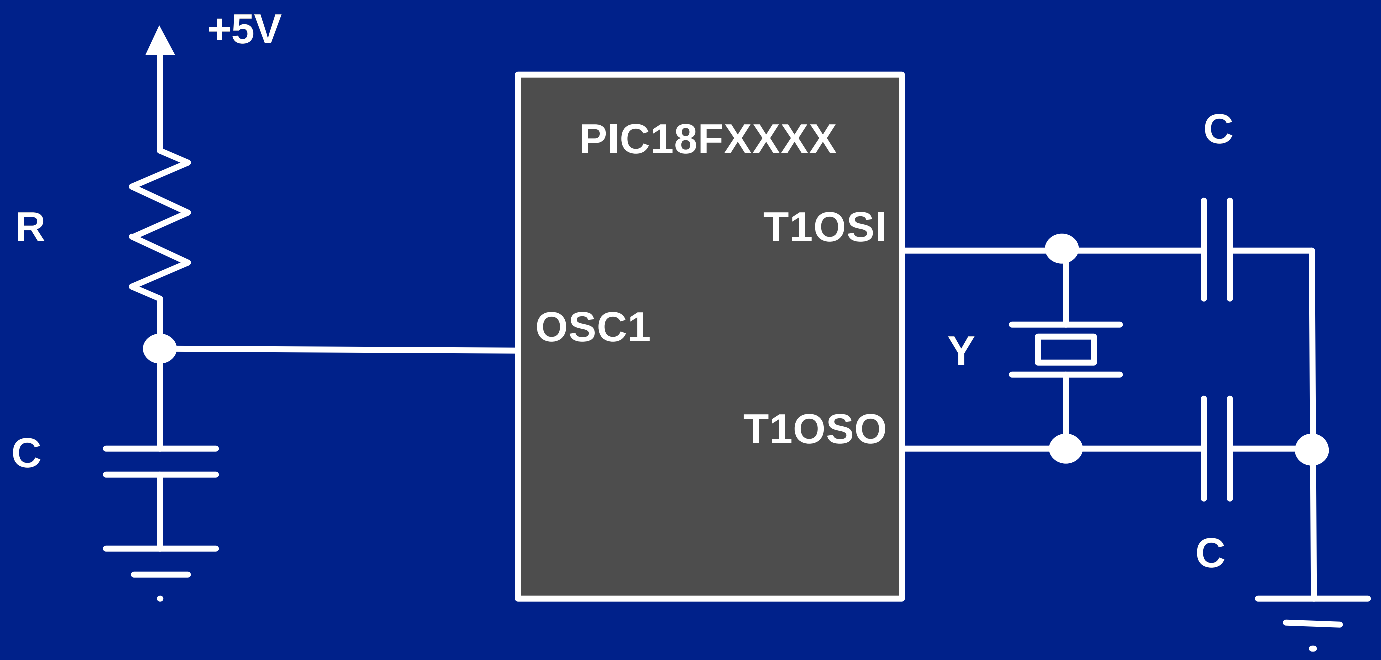
PIC18 Peripherals

Timer1 and Timer3 (Continued)



PIC18FXXX MCU Peripherals

TMR1 as a Real Time Clock



Preload TMR1H register for faster overflows:

TMR1H=80h → 1 second overflow
TMR1H=C0h →0.5 second overflow

See Application Note AN580 for more info.

Timer 1 Setup

T1CON

bit 7							bit 0
RD16	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNCH	TMR1CS	TMR1ON

RD16	**16-bit Read/Write Mode Enable** 1 = Enables Read/Write of Timer 1 in one 16-bit operation 0 = Enables Read/Write of Timer 1 in two 8-bit operations
T1CKPS1:T1CKPS0	**Timer 1 Input Clock Prescale Selection** 11 = 1:8 01 = 1:2 10 = 1:4 00 = 1:1
T1OSCEN	**Timer 1 Oscillator Enable** 1 = Timer 1 oscillator is enabled 0 = Timer 1 oscillator is disabled
T1SYNCH	**Timer 1 External Clock Synchronization Selection** 1 = Do NOT synchronize external clock 0 = Synchronize external clock input
TMR1CS	**Timer 1 Clock Source Selection** 1 = External clock from RC0/T1OSC0/T13CKI (counter) 0 = Internal Instruction Cycle
TMR1ON	**Timer 1 On / Off Selection** 1 = Enables Timer 1 0 = Disables Timer 1

Timer 3 Setup

T3CON

bit 7							bit 0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	T3SYNCH	TMR3CS	TMR3ON

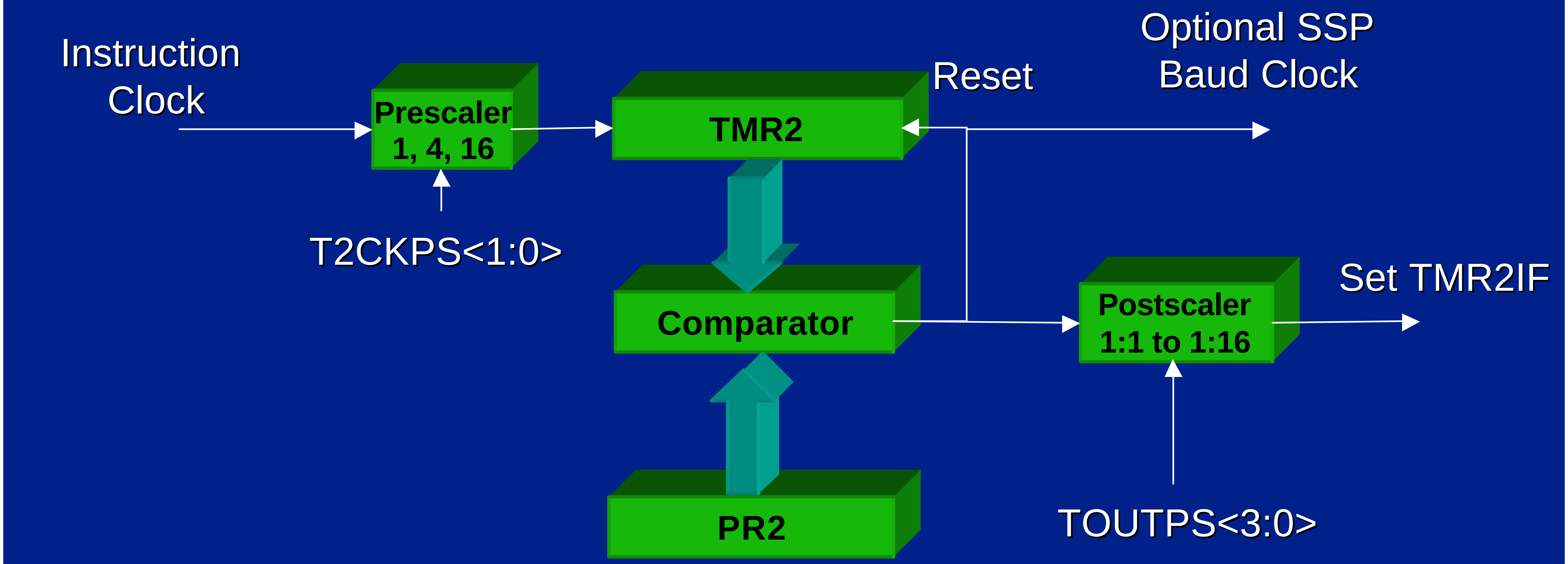
RD16	**16-bit Read/Write Mode Enable** 1 = Enables Read/Write of Timer 3 in one 16-bit operation 0 = Enables Read/Write of Timer 3 in two 8-bit operations
T3CCP2:T3CCP1	**Timer 3 and Timer 3 CCP Timebase Selection** 1X = Timer 3 is Capture/Compare clock source for all CCPs 10 = Timer 3 is Capture/Compare clock source for CCP2, Timer 1 is Capture/Compare clock source for CCP1 01 = Timer 1 is Capture/Compare clock source for all CCPs
T3CKPS1:T3CKPS0	**Timer 3 Input Clock Prescale Selection** 11 = 1:8 01 = 1:2 10 = 1:4 00 = 1:1
T3SYNCH	**Timer 3 External Clock Synchronization Selection** 1 = Do NOT synchronize external clock 0 = Synchronize external clock input
TMR3CS	**Timer 3 Clock Source Selection** 1 = External clock from RC0/T1OSC0/T13CKI (counter) 0 = Internal Instruction Cycle
TMR3ON	**Timer 3 On / Off Selection** 1 = Enables Timer 1 0 = Disables Timer 1

PIC18 Peripherals

Timer2 and Timer4

- 8-bit Timers with prescaler and postscaler
- TMR2 used as time base for PWM mode of CCP module
- TMR2/TMR4 are readable & writable
- TMR2/TMR4 increments until they match period PR2/PR4, then resets to 00h
- TMR2/TMR4 match with PR2/PR4 generates an interrupt through postscaler
- TMR2 can serve as baud clock for MSSP

PIC18 Peripherals
TMR2 Timer: Period Register



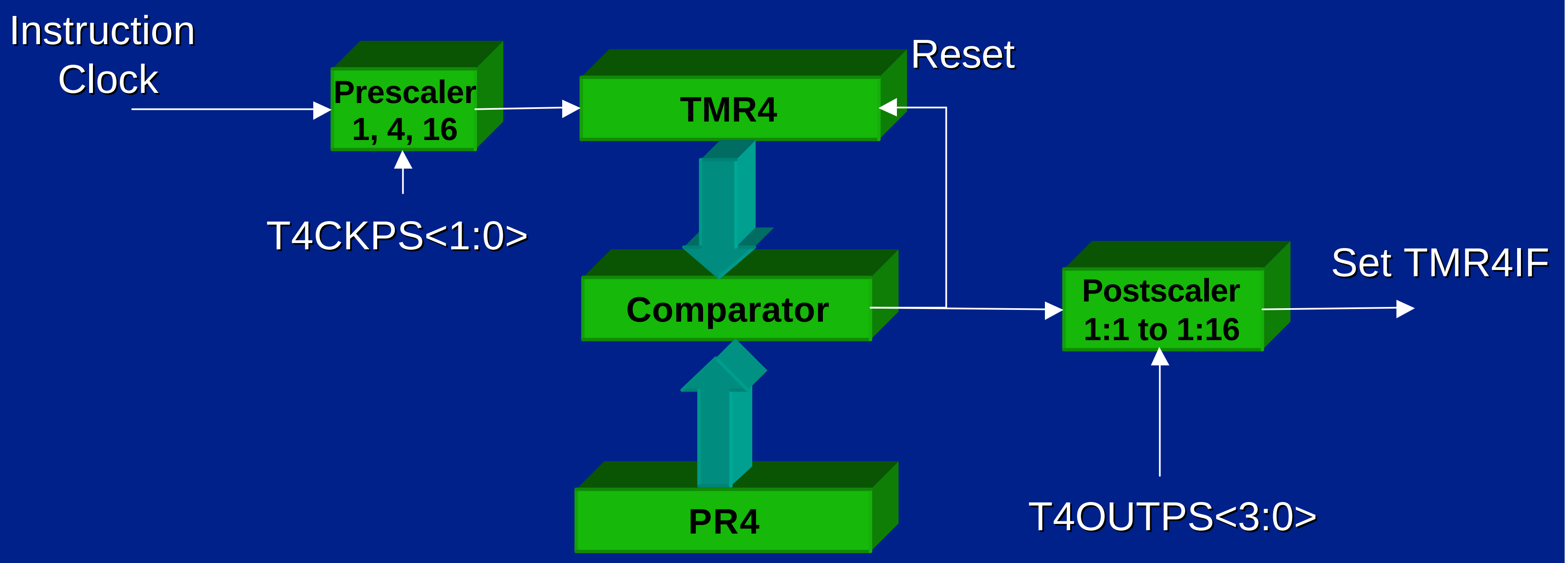
Timer 2 Setup

T2CON Register Format

bit 7							bit 0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

TOUTPS<3:0>	**Select Timer 2 Postscaler:** 0000 = 1:1 Postscale 0001 = 1:2 Postscale …. 1111 = 1:16 Postscale
TMR2ON	**Timer 2 On / Off Control:** 0 = Timer 2 is Off 1 = Timer 2 is On
T2CKPS1	**Select Timer 2 Prescaller:** 00 = Prescaller is 1 01 = Prescaller is 4 1X = Prescaller is 16

PIC18 Peripherals
TMR4 Timer: Period Register



Timer 4 Setup

T4CON Register Format

bit 7							bit 0
-	T4OUTPS3	T4OUTPS2	T4OUTPS1	T4OUTPS0	TMR4ON	T4CKPS1	T4CKPS0

T4OUTPS<3:0>	**Select Timer 4 Postscaler:** 0000 = 1:1 Postscale 0001 = 1:2 Postscale …. 1111 = 1:16 Postscale
TMR4ON	**Timer 4 On / Off Control:** 0 = Timer 4 is Off 1 = Timer 4 is On
T4CKPS1	**Select Timer 4 Prescaller:** 00 = Prescaller is 1 01 = Prescaller is 4 1X = Prescaller is 16

Timer 2 Interrupts

RCON Register

bit 7							bit 0
IPEN	-	-	~RI	~TO	~PD	~POR	~BOR

IPEN	**Interrupt Priority Level Enable:** 1 = Enable Interrupt Priority Levels 0 = Disable Interrupt Priority Levels

INTCON Register

bit 7							bit 0
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

GIE/GIEH	**Global Interrupt Enable** **IPEN=0** 1 = Enable Unmasked Interrupts 0 = Disable all interrupts	**IPEN=1** 1 = Enables High Priority Interrupts 0 = Disables High Priority Interrupts
PEIE/GIEL	**Peripheral Interrupt Enable** **IPEN = 0** 1 = Enables Unmasked Peripheral Interrupts 0 = Disables Peripheral Interrupts	**IPEN = 1** 1 = Enables Low Priority Interrupts 0 = Disables Low Priority Interrupts

Timer 2 Interrupts Continued

PIR1 (Peripheral Interrupt Request Flag) Register

bit 7							bit 0
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

TMR2IF	Timer 2 to PR2 Match Interrupt Flag 1 = TMR2 to PR2 Match Interrupt Occurred 0 = No TMR2 to PR2 Match Occurred

PIE1 (Peripheral Interrupt Enable) Register

bit 7							bit 0
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

TMR2IE	Timer 2 to PR2 Match Interrupt Enable 1 = Enable TMR2 to PR2 Match Interrupts 0 = Disable TMR2 to PR2 Match Interrupts

IPR1 (Peripheral Interrupt Priority) Register

bit 7							bit 0
PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

TMR2IP	Timer 2 to PR2 Match Interrupt Priority Selection 1 = TMR2 to PR2 Match Assigned to High Priority Interrupt 0 = TMR2 to PR2 Match Assigned to Low Priority Interrupt

TMR2 Initialization Example

- 200 uS / 5 Khz high priority interrupt, 40 Mhz clock / 10 Mhz instruction clock:

```
T2CON = 0b00001101;   // 4:1 pre 2:1 postscale
PR2 = 249;            // 250 count TMR2 period
RCON = 0b10000000;    // Enable Priority
PIE1 = 0b00000010;    // Enable TMR2 interrupt
IPR1 = 0b00000010;    // TMR2 high priority
PIR1bits.TMR2IF = 0;  // Optional to eliminate
TMR2 = 0;             // first interrupt
INTCON = 0b10000000;  // Turn on interrupts
```

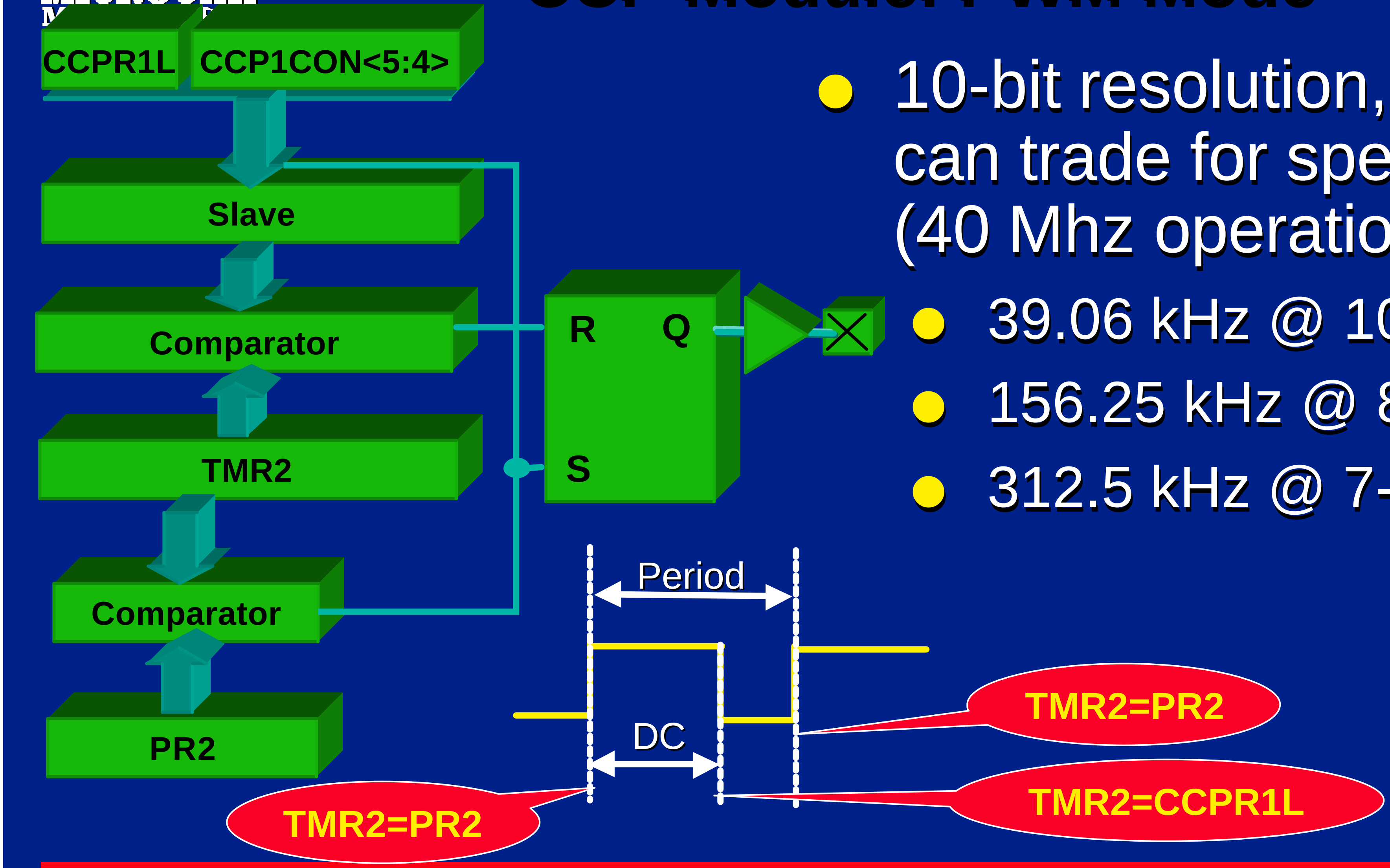
10,000,000 / (4 (prescale) * 2 (postscale) * 250 (period)) = 5,000 Khz or 200 uS period

Timer 2 ISR Example

- Test and Clear PIR1bits.TMR2IF:

```
void high_priority_interrupt(void){
   if (PIR1bits.TMR2IF){
      PIR1bits.TMR2IF = 0;
      // execute Timer 2 service code here
      }
   else if (<other high priority peripherals>){
      // Clear other peripheral bits
      // execute peripheral service code here
      }
   else Reset(); // Hit interrupt without valid
}          // flag - illegal condition so restart
```

PIC18 Peripherals
CCP Module: PWM Mode



- 10-bit resolution, can trade for speed (40 Mhz operation)
  - 39.06 kHz @ 10-bit
  - 156.25 kHz @ 8-bit
  - 312.5 kHz @ 7-bit

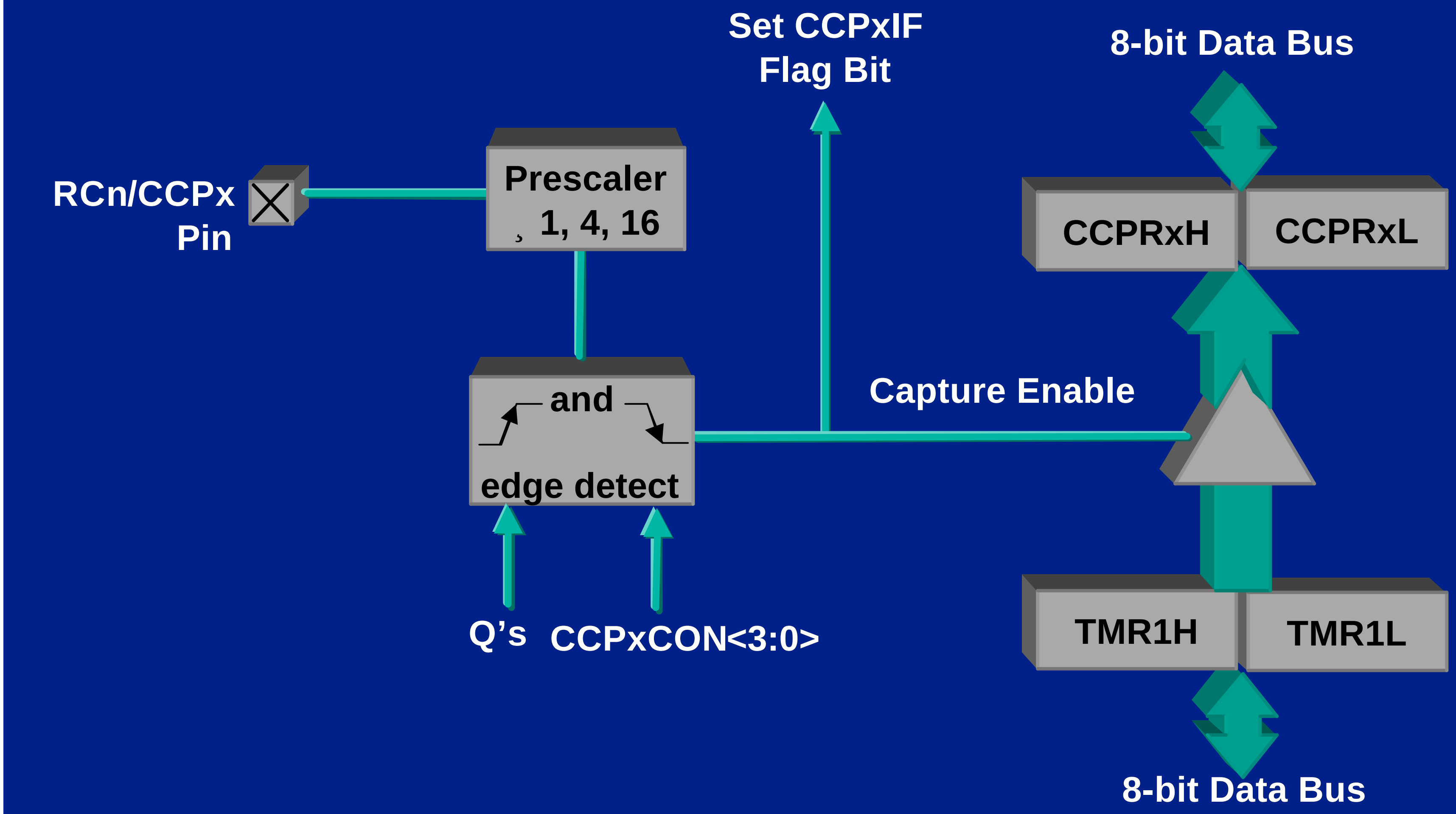
PIC18 Peripherals

CCP Module: Input Capture Mode

- Captures 16-bit TMR1 value when an event occurs on CCPx pin:
  - Every falling edge
  - Every rising edge
  - Every 4th rising edge
  - Every 16th rising edge
- Capture generates an interrupt

PIC18 Peripherals

CCP Module: Input Capture Mode *(continued)*



PIC18 Peripherals

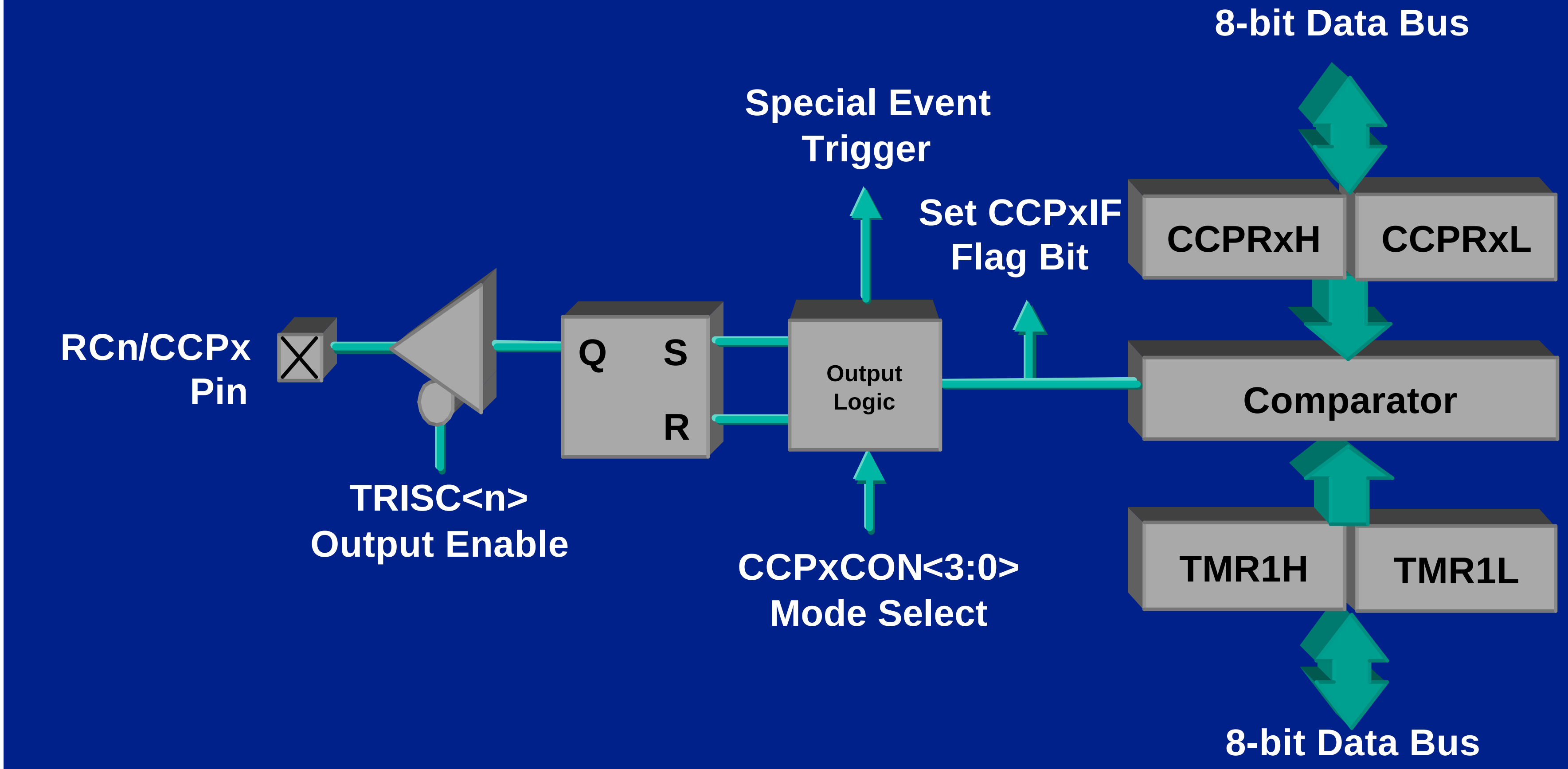
CCP Module: Output Compare Mode

- 16-bit CCPRx register value is compared to TMR1, and on match the CCPx pin is
  - Driven High/Low
  - Toggled
  - Unchanged
- Compare match generates interrupt
- Special event trigger clears TMR1 and can start A/D conversion

PIC18 Peripherals

CCP Module: Output Compare Mode

(continued)



CCP1 Setup

CCP1CON

bit 7							bit 0
-	-	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

DC1B1:DC1B0	**(2) LSBs of PWM Duty Cycle** **PWM Mode** -> (2) LSBs of a 10-bit Duty Cycle. The upper (8) bits (DC19:DC12) of the duty cycle are found in CCPR1L **Capture/Compare Modes** -> Unused
CCP1M3:CCP1M0	**CCP1 Mode Selection** 0000 = Capture/Compare/PWM 1 Disable (resets CCP1 module) 0001 = Reserved 0010 = Compare Mode, Toggle CCP1 output on match 0011 = Reserved 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, force CCP1 output High on match 1001 = Compare Mode, force CCP1 output Low on match 1010 = Compare Mode, CCP1 output unchanged 1011 = Compare Mode, Trigger Special Event 11XX = PWM Mode

Note: Pin defaults to ‘0’ when capture mode is engaged

CCP2 Setup

CCP2CON

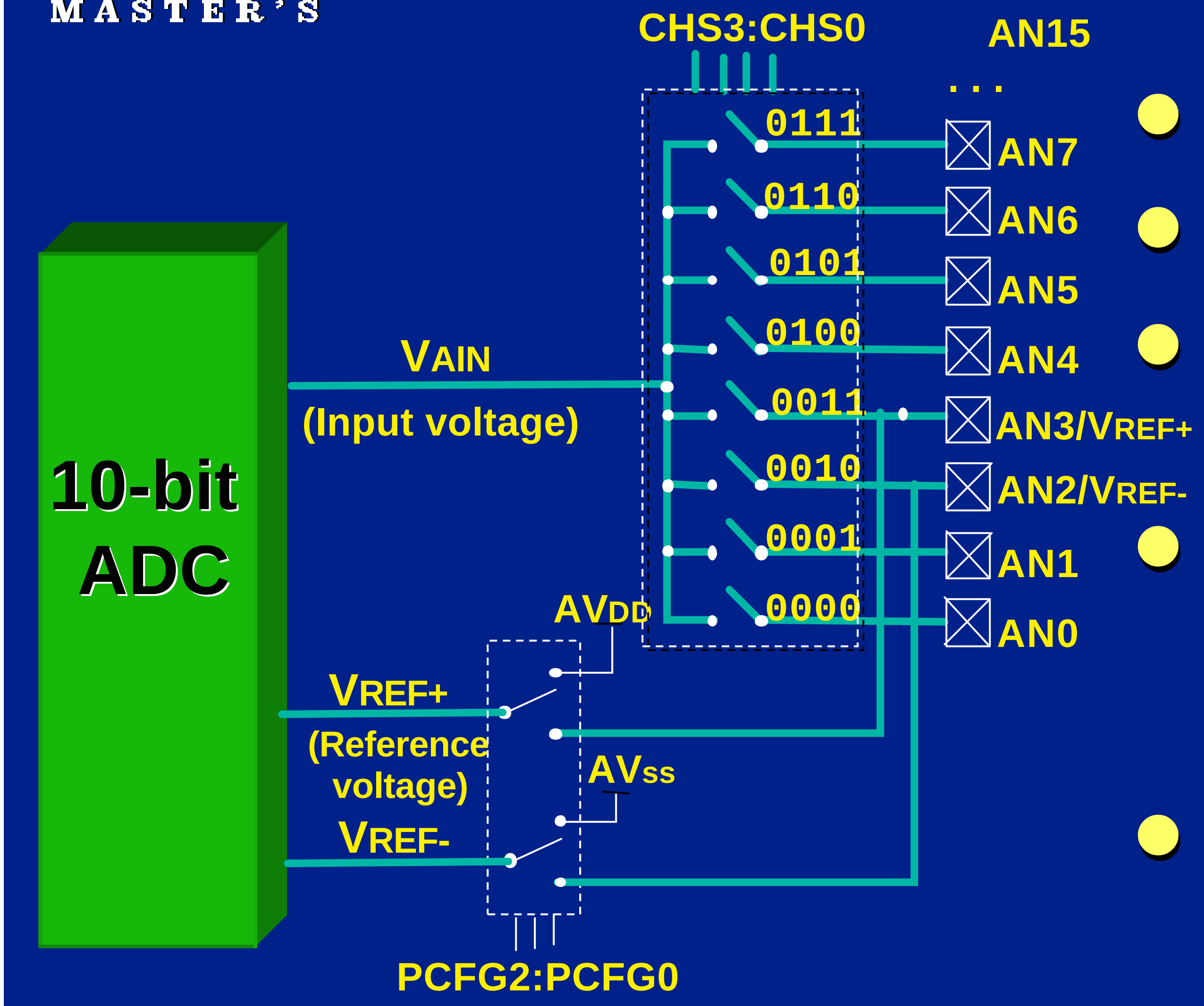
bit 7							bit 0
-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0

DC2B1:DC2B0	**(2) LSBs of PWM Duty Cycle** **PWM Mode** -> (2) LSBs of a 10-bit Duty Cycle. The upper (8) bits (DC29:DC22) of the duty cycle are found in CCPR2L **Capture/Compare Modes** -> Unused
CCP2M3:CCP2M0	**CCP2 Mode Selection** 0000 = Capture/Compare/PWM 1 Disable (resets CCP2 module) 0001 = Reserved 0010 = Compare Mode, Toggle CCP2 output on match 0011 = Reserved 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, force CCP2 output High on match 1001 = Compare Mode, force CCP2 output Low on match 1010 = Compare Mode, CCP2 output unchanged 1011 = Compare Mode, Trigger Special Event 11XX = PWM Mode

Note: Pin defaults to ‘0’ when capture mode is engaged

PIC18 Peripherals

10-bit ADC - Block Diagram



- Up to 16 ch.
- 10-bit ± 1 LSb
- Conversion during SLEEP
- Internal Or External Reference
- Up to 25ksps
  - 34 ksps without channel change

A/D Setup ADCON0

	bit 7							bit 0
ADCON0	ADCS1	ADCS0	CSH2	CHS1	CHS0	GO_DONE	-	ADON

ADCS1:ADCS0 **Also** **ADCON1 ADCS2**	**A/D Conversion Clock Select (ADCON1 contains ADCS2)** ADCON1.ADCS2 = 0 00 = FOSC/2 01 = FOSC/8 10 = FOSC/32 11 = Frc Internal RC Oscillator	 ADCON1.ADCS2 = 1 00 = FOSC/4 00 = FOSC/16 00 = FOSC/64 11 = Frc Internal RC Oscillator
CH2:CH0	**Analog Channel Select Bits** 000 = Channel 0, AN0 001 = Channel 1, AN1 010 = Channel 2, AN2 011 = Channel 3, AN3 100 = Channel 4, AN4 101 = Channel 5, AN5 110 = Channel 6, AN6 111 = Channel 7, AN7	
GO_DONE	**A/D Conversion Status and Conversion Start** 1 = Conversion in progress, set this bit to start a conversion 0 = Conversion complete, result in ADRES, cleared by A/D converter	
ADON	**A/D Converter On / Off Selection** 1 = Enables A/D Converter 0 = Disables A/D Converter	

A/D Setup ADCON1

ADCON1

bit 7							bit 0
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0

ADFM	**A/D Result Format Selection** 1 = Right Justified. (6) MSBs of ADRESH are ‘0’ 0 = Left Justified, (6) LSBs of ADRESL are ‘0’
ADCS2	**See ADCON0 for Conversion Clock Selection**
PCFG3:PCFG0	**Analog Port Configuration Control**

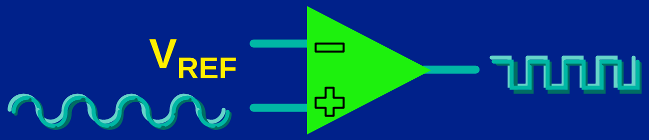
<3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C / R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7 / 1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5 / 0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4 / 1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3 / 0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2 / 1
011x	D	D	D	D	D	D	D	D	—	—	0 / 0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6 / 2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5 / 1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4 / 2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3 / 2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2 / 2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

Configuring Inputs as Digital or Analog

- Pins defined as digital enable the digital input buffer
  - Avoid voltages that reside below V_{IH} and above V_{IL} to prevent excessive current
  - PORT pin reads reflect the pin state
- Pins defined as analog disable the digital input buffer
  - Any voltage below Vdd and above Vss is fine
  - PORT pin reads will always be ‘0’
- All pins (D or A) can be digital outputs

PIC18 Peripherals

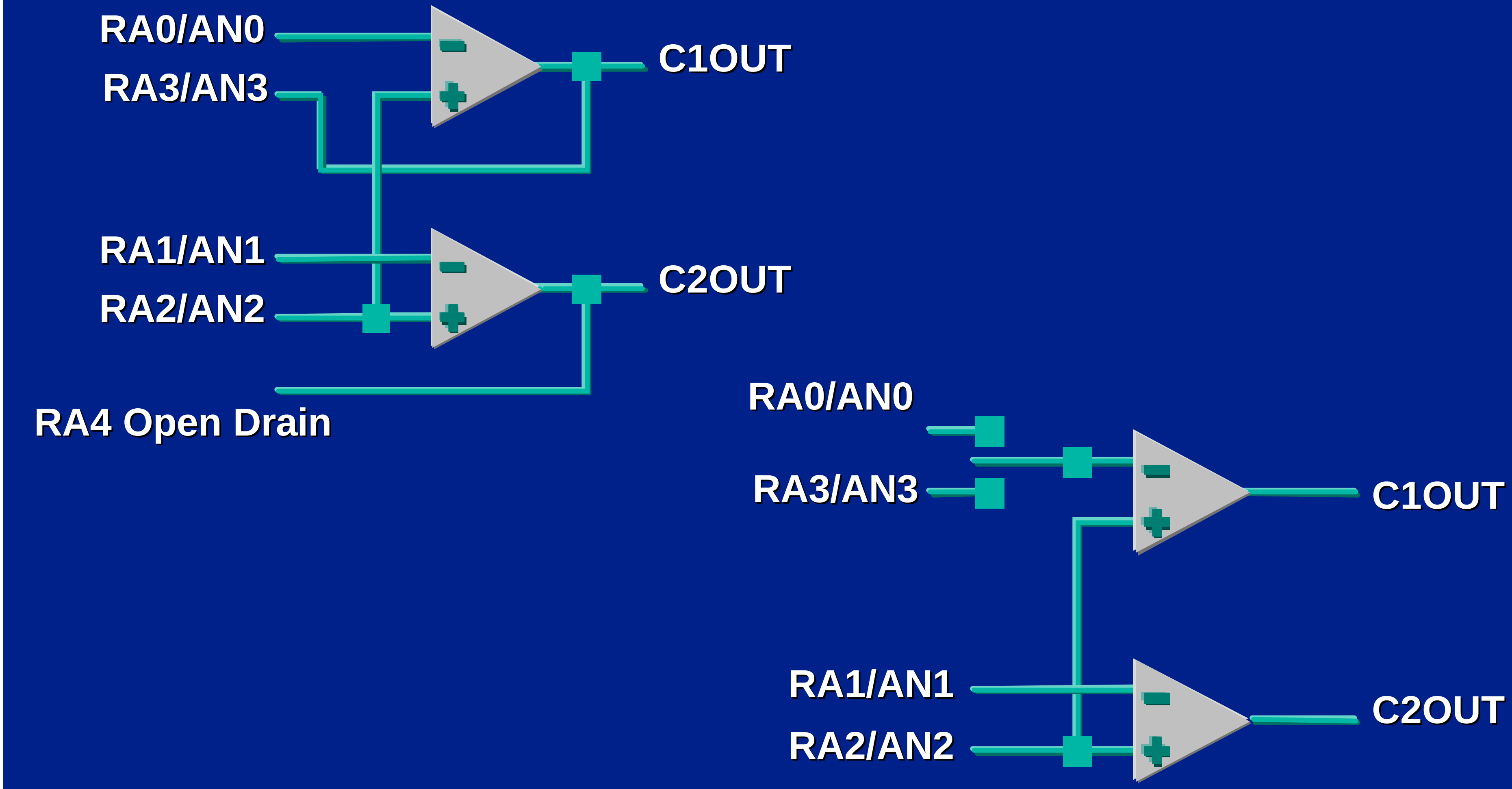
Analog Comparator Module



- Two Analog Comparators
- Programmable on-chip voltage reference
- Eight Programmable modes of operation
- Operates in SLEEP mode
- Generates interrupt / wake-up on output change
- Comparator output pin available

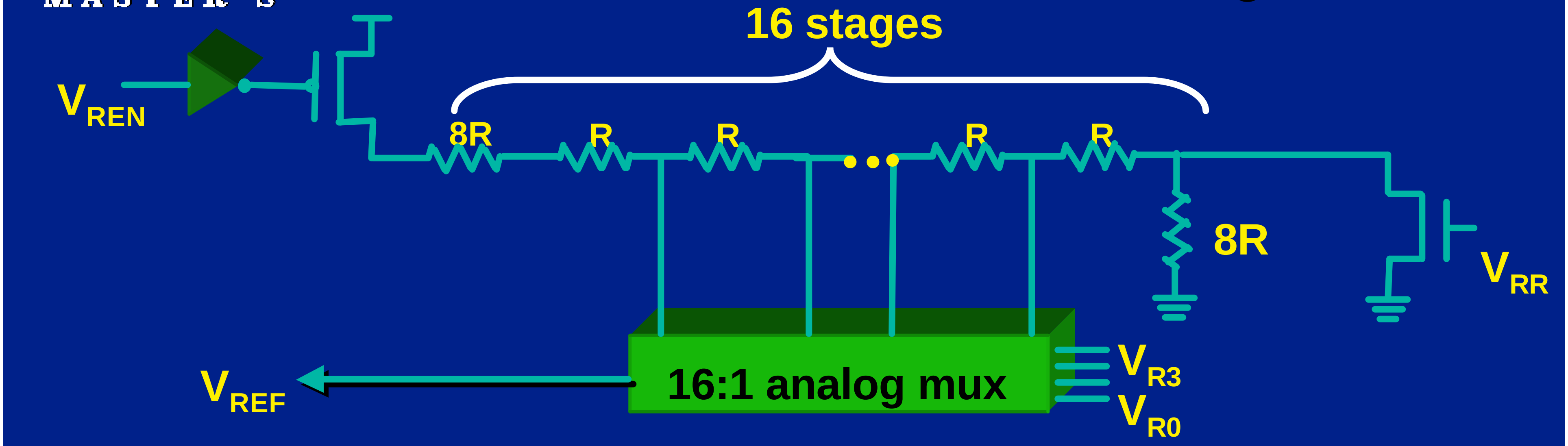
PICmicro MCU Peripherals

Analog Comparator Module *(continued)*



PIC18 Peripherals

Internal VREF: Block Diagram



- 24 or 32 step sizes
- Internal or External Voltage Reference
- Can be used as a D/A converter
- VREF can be directed to an output pin

Note: Check your device datasheet for availability

Comparator Setup

CMCON

bit 7							bit 0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0

C2OUT	**Comparator 2 Output Selection** **C2INV = 0:** **C2INV = 1:** 1 = C2 Vin+ > C2 Vin- 1 = C2 Vin+ < C2 Vin- 0 = C2 Vin+ < C2 Vin- 0 = C2 Vin+ > C2 Vin-
C1OUT	**Comparator 1 Output Selection** **C1INV = 0:** **C1INV = 1:** 1 = C1 Vin+ > C1 Vin- 1 = C1 Vin+ < C1 Vin- 0 = C1 Vin+ < C1 Vin- 0 = C1 Vin+ > C1 Vin-
C2INV	**Comparator 2 Output Inversion** 1 = C2 Output inverted 0 = C2 Output not inverted
C1INV	**Comparator 1 Output Inversion** 1 = C1 Output inverted 0 = C1 Output not inverted
CIS	**Comparator 1 Input Switch (when CM<2:0> = 110)** 1 = C1 Vin- connects to RF5/AN10, C2 Vin- connects to RF3/AN8 1 = C1 Vin- connects to RF6/AN11, C2 Vin- connects to RF4/AN9
CM<2:0>	**Comparator Mode Selection** See Comparator Mode Figure

Comparator Reference Setup

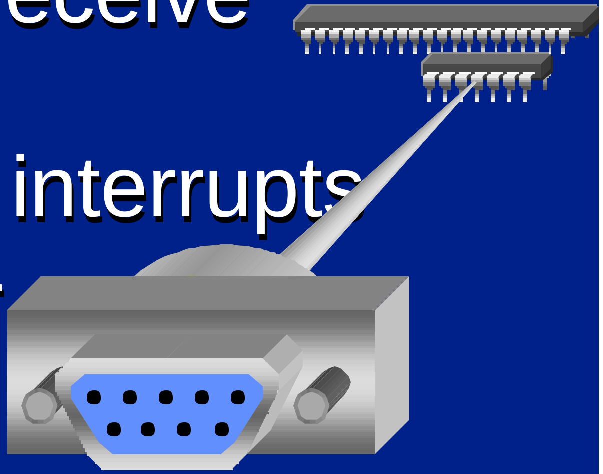
CVRCON

bit 7							bit 0
CVREN	CVROE	CVRR	CVRSS	CVR3	CVR2	CVR1	CVR0

CVREN	**Comparator Voltage Reference Enable** 1 = Enables CVREF Circuit, reference ON 0 = Disables CVREF Circuit, reference OFF
CVROE	**Comparator Output Enable** 1 = CVREF Voltage also driven onto RF5/CVREF pin 0 = CVREF disconnected from RF5/CVREF pin Note: TRISF<5> must be set to a ‘1’ (input)
CVRR	**Comparator VREF Source Selection** 1 = 0.00 CVRSRC to 0.75 CVRSRC with CVRSRC/24 step 1 = 0.25 CVRSRC to 0.75 CVRSRC with CVRSRC/32 step
CVR3:CVR0	**Comparator VREF Value Selection** When CVRR = 1 CVREF = (CVR<3:0>/24) * CVRSRC When CVRR = 0 CVREF = (0.25 + (CVR<3:0>/32))* CVRSRC

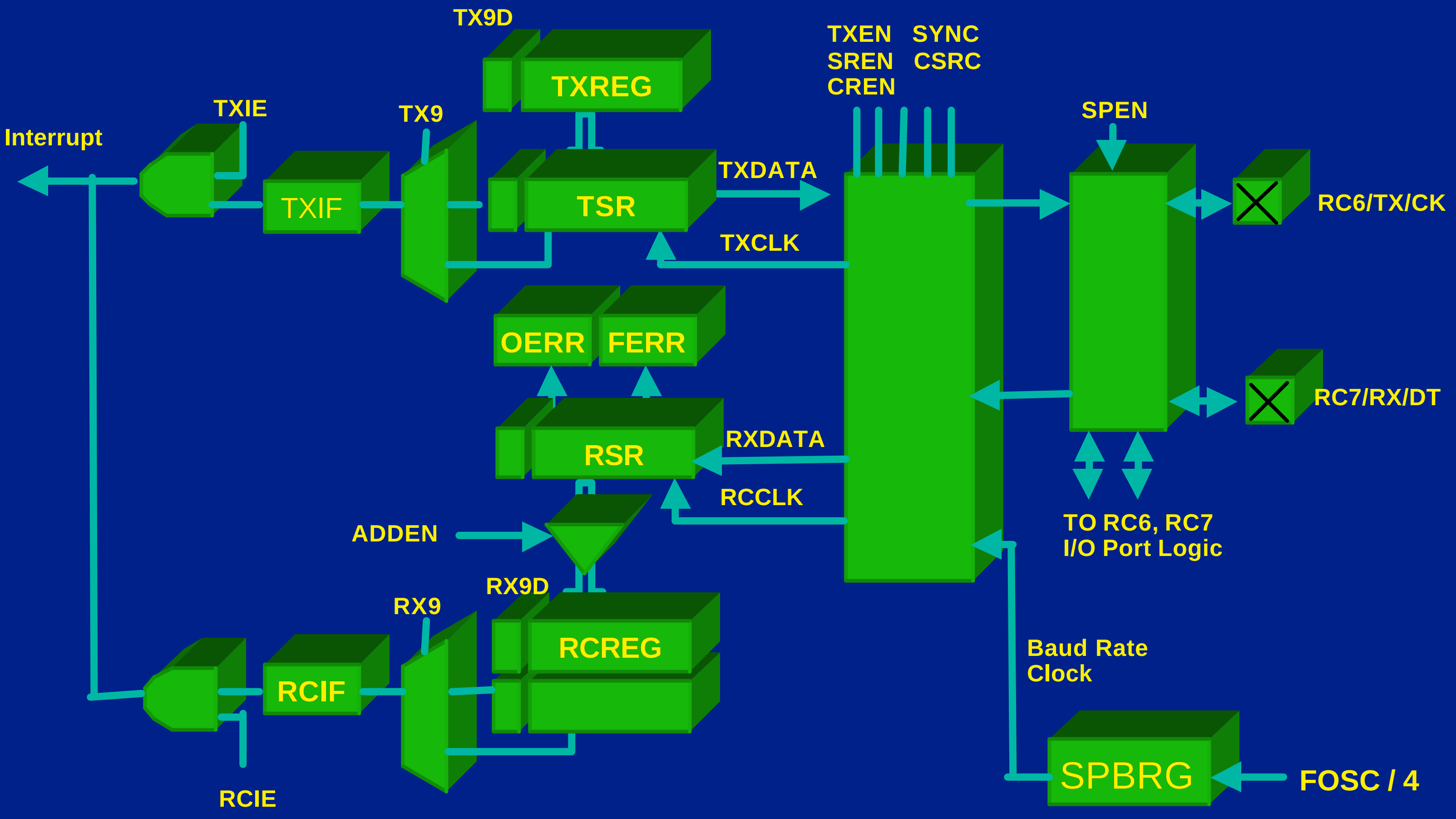
PIC18 Peripherals
Addressable USART (AUSART)

- Full-duplex Asynchronous Or Half-duplex Synchronous
- 9-bit Addressable mode
- Double-buffered transmit and receive buffers
- Separate transmit and receive interrupts
- Dedicated baud rate generator
- Max bit rates @ 40MHz
  - Asynchronous: 625 kbps / 2.5 Mbps
  - Synchronous: 10 Mbps



PIC18 Peripherals

USART Block Diagram



UART Tx Setup

TXSTA

bit 7							bit 0
CVREN	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D

CSRC	**Clock Source Selection (synch mode only)** 1 = Master mode, clock generated by internal BRG 0 = Slave mode, clock derived from external
TX9	**9-bit / 8-bit Mode Transmission Selection** 1 = 9-bit Transmission Format 0 = 8-bit Transmission Format
TXEN	**Transmit Enable (overridden by SREN/CREN in SYNC mode)** 1 = Transmitter Enabled 0 = Transmitter Disabled
SYNC	**Synchronous / Asynchronous Selection** 1 = Synchronous Mode 0 = Asynchronous Mode
BRGH	**High / Low Baud Rate Selection** 1 = High Speed Baud Rate, FOSC / 16 0 = Low Speed Baud Rate, FOSC / 64
TRMT	**Transmit Shift Register Status** 1 = Transmit Shift Register Empty 0 = Transmit Shift Register Full
TX9D	**9th Bit of Transmit Data (valid only in 9-bit mode)** Written before TXREG, used for parity or address/data

UART Rx Setup

RCSTA

bit 7							bit 0
SPEN	RXD	SREN	CREN	ADDEN	FERR	OERR	RX9D

SPEN	**Serial Port Enable** 1 = Serial Port Enabled, Uses RX and TX as serial port pins 0 = Serial Pore Disabled, RX and TX general purpose I/Os
RX9	**9-bit / 8-bit Mode Reception Selection** 1 = 9-bit Reception Format 0 = 8-bit Reception Format
SREN	**Single Receive Enable (Synchronous Mode Only)** 1 = Enable a Single Receive 0 = Disable Single Receive, cleared when reception completed
CREN	**Continuous Receive Enable** 1 = Enables Receiver; Continuous Reception in Synch mode, overriding SREN 0 = Disables Receiver in Asynchronous Mode, SREN controls Synch mode
ADDEN	**Address Detect Enable** 1 = Enables 9-bit Address Detection, Interrupt and load RCREG when bit 9 is ‘1’ 0 = Disables Address Detection, all bytes received
FERR	**Framing Error** 1 = Framing Error Occurred in this byte, clear by read RCREG + receive next byte 0 = No Framing Error
OERR	**Overrun Error** 1 = Overrun Error, cleared by clearing CREN 0 = No Overrun Error
RX9D	**9th Bit of Received Data (valid only in 9-bit mode)** Read before TXREG, used for parity or address/data

UART Baud Rate Generator

- Separate Resource does not use any timers
- Divides (FOSC / 16 or 64) by 1 to 256

$$\text{Baud Rate} = \frac{\text{FOSC}}{64 * (\text{SPBRG} + 1)}$$

Low Speed Mode
TXSTAbits.BRGH = 0

$$\text{Baud Rate} = \frac{\text{FOSC}}{16 * (\text{SPBRG} + 1)}$$

High Speed Mode
TXSTAbits.BRGH = 1

UART Buffers

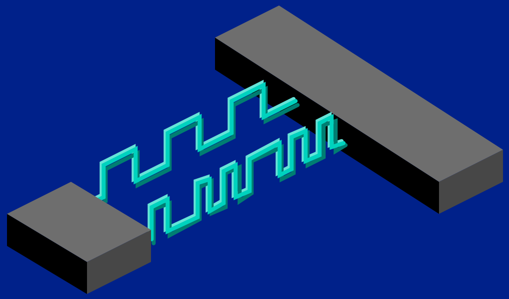
- Load TXREG with byte to be transmitted
  - Buffer empty ONLY when PIR1bits.TXIF is set
- Read received byte from RCREG
  - Received data ONLY when PIR1bits.RCIF is set

```
void putchar(value){
   while (PIR1bits.TXIF == 0);// Wait for empty FIFO
   TXREG = value;
   }
```

PIC18 Peripherals

Master Synchronous Serial Port

- Operates in either SPI™ or I²C™ mode
- SPI Mode
  - Programmable baud rate
  - Maximum baud rates (@ 40MHz)
    - Master: 10 Mbps
    - Slave: 2.5 Mbps Single Byte Tx
  - All four SPI modes supported (0,0;0,1;1,0;1,1)



- I²C Mode
  - Supports standard (100kHz), fast (400kHz), and Microchip's 1MHz I²C standards
  - Hardware Master/Slave implementation

SPI is a trademark of Motorola Semiconductor
I²C is a trademark of Philips Semiconductors

MSSP SPI Mode Setup

	bit 7							bit 0
SSPSTAT	SMP	CKE	D_A	P	S	R_W	UA	BF

SMP	**Input Sample Control** 1 = Input sampled at the end of data output time 0 = Input sampled at the middle of data output time
CKE	**Clock Edge Selection** If CKP = 0 If CKP = 0 1 = Data transmitted on SCLK rising edge 1 = Data transmitted on SCLK falling edge 0 = Data transmitted on SCLK falling edge 0 = Data transmitted on SCLK rising edge
D_A	**Data / Address bit used ONLY in I2C mode, unused in SPI mode**
P	**Stop bit used ONLY in I2C mode, unused in SPI mode**
S	**Start bit used ONLY in I2C mode, unused in SPI mode**
R_W	**Read / Write bit used ONLY in I2C mode, unused in SPI**
UA	**Update Address bit used ONLY in I2C mode, unused in SPI mode**
BF	**Buffer Full (Receive mode only)** 1 = Receive complete, SSBUF is full 0 = Receive not complete, SSBUF is empty

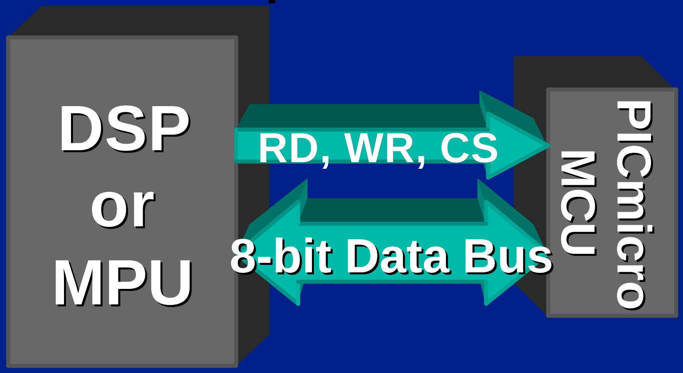
MSSP SPI Mode Setup Cont.

	bit 7							bit 0
SSPCON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

WCOL	**Write Collision Detection (Master Mode Only – Must be cleared in software)** 1 = The SSPBUF register was written while still transmitting a previous word 0 = No write collision
SSPOV	**Receive Overflow Indicator (Slave Mode Only – Must be cleared in software)** 1 = A new byte has been received from the master before the previous byte was read from SSPBUF. In case of overflow, the data is lost and SSPBUF must be read to clear overflow condition. Slave transmitter applications should also read SSBUF after each byte 0 = No Slave Receive Overflow
SSPEN	**Synchronous Serial Port Enable** 1 = Enables serial port and configures SCK, SDO, SDI and SS as serial port pins 0 = Disables serial port; allows SCK, SDO SDI and SS to be used as general purpose I/Os
SCP	**Clock Polarity Selection** 1 = Idle state for clock is a high level 0 = Idle state for clock is a low level
SSPM3:SSPM0	**Synchronous Serial Port Mode Selection** 0101 = SPI Slave Mode, Clock – SCLK, SS Control Disabled, SS is GPIO 0100 = SPI Slave Mode, Clock = SCLK, SS Control enabled 0011 = SPI Master Mode, Clock = Timer 2 Output / 2 0010 = SPI Master Mode, Clock = FOSC/64 0001 = SPI Master Mode, Clock = FOSC/16 0000 = SPI Master Mode, Clock = FOSC/4 **NOTE: Other combinations used in I2C mode or reserved**

PICmicro MCU Peripherals
Parallel Slave Port

- Provides an 8-bit interface such that the PICmicro MCU may be used as a peripheral to a microprocessor
- Three I/O on PORTE act as Chip Select, Read, and Write lines
- PORTD is the data bus
- Separate read and write interrupts available
- Currently available on most 40-pin, 14-bit core devices

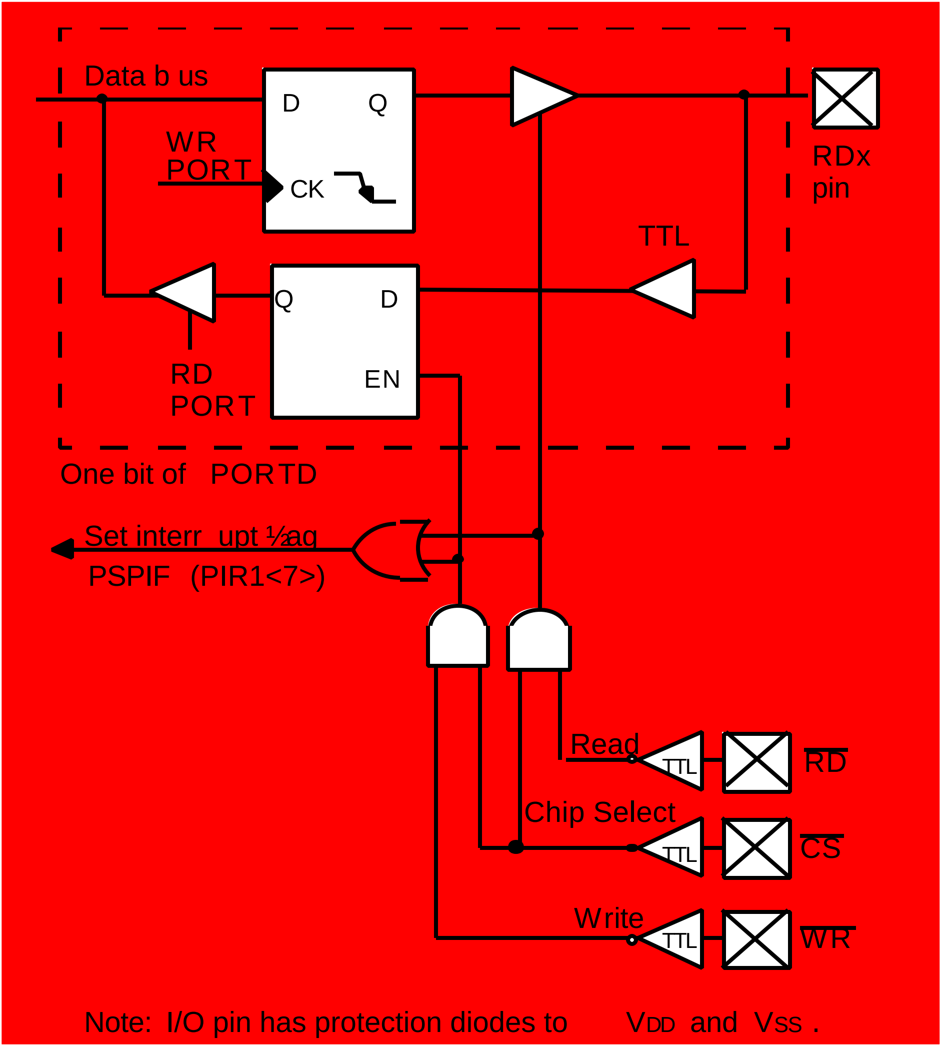


PICmicro MCU Peripherals
Parallel Slave Port: MCU Interface

- Direct interface to 8-bit microprocessor data bus
- Asynchronous operation (to external world)
- Interrupt generated on external read or write operation on parallel port
- Uses Port D and Port E
  - Port D: Data bus
  - Port E: Control signals (read, write, and chip select)

PICmicro MCU Peripherals

Parallel Slave Port: Block Diagram



Note: I/O pin has protection diodes to VDD and VSS .

Parallel Slave Port Setup

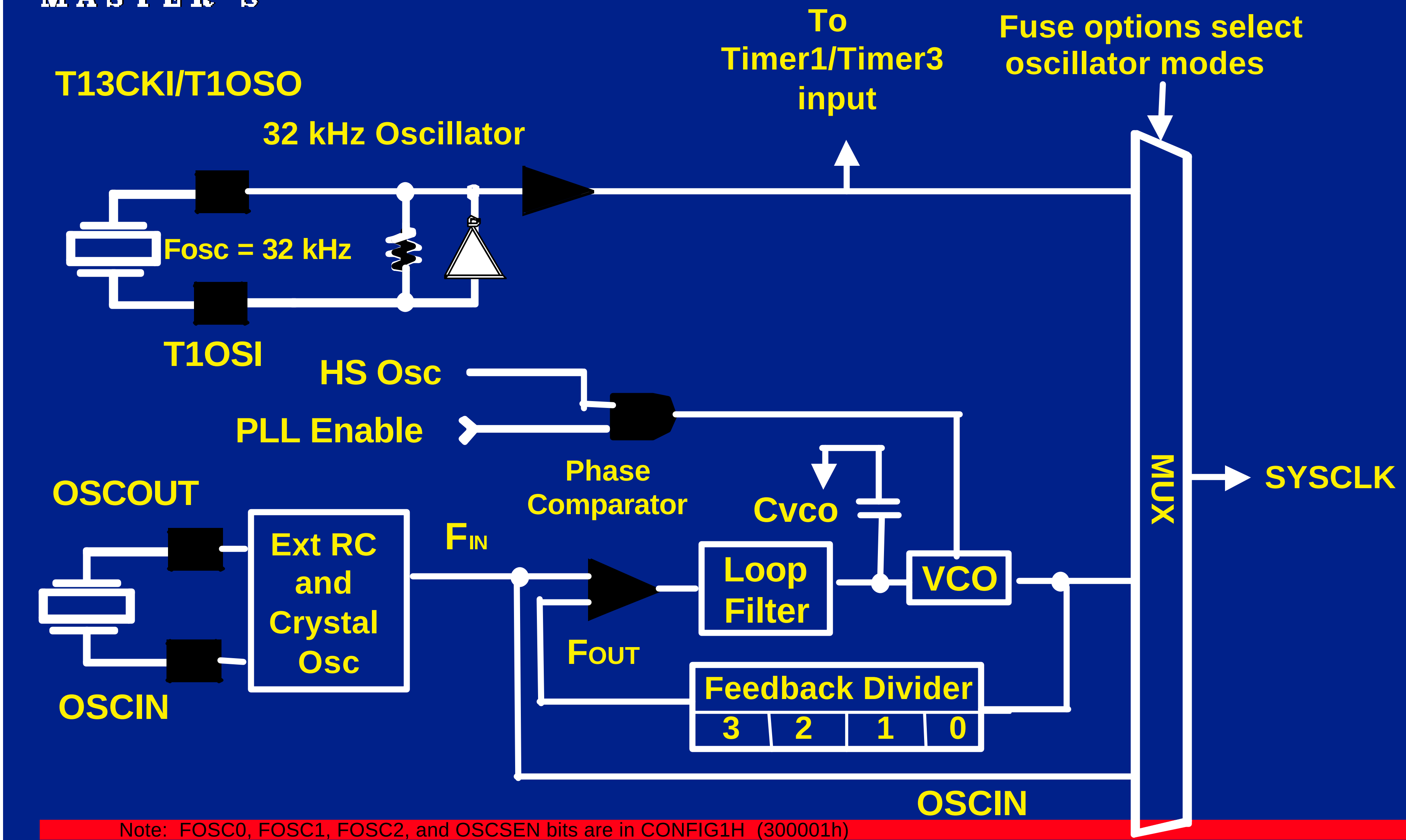
TRISE

bit 7							bit 0
IBF	OBF	IBOV	PSPMODE	-	TRISE2	TRISE1	TRISE0

IBF	**Input Buffer Full Status** 1 = A word has been received from the master into PORTD and is waiting to be read 0 = No word has been received from the master
OBF	**Output Buffer Full Status** 1 = The PORTD output buffer still holds a previously written word 0 = The PORTD output buffer has been read by the master and is now empty
IBOV	**Input buffer Overflow Detect Status (Must Be Cleared In Software)** 1 = The master wrote a byte before a previously written byte was read from PORTD 0 = No write overflow occurred
PSPMODE	**Parallel Slave Port Mode Selection** 1 = Enable Parallel Slave Port 0 = Disable Parallel Slave Port, PORTD and PORTE General Purpose I/Os
TRISE2:TRISE0	**PORTE, Pins RE2:RE0 Direction Control** 1 = RE x set to input 0 = RE x set to output

Special Features

New Oscillator Modes
PIC18F452 Oscillator Block Diagram



Note: FOSC0, FOSC1, FOSC2, and OSCSEN bits are in CONFIG1H (300001h)

PIC18 Special Features

Programmable Low Voltage Detect

- Provides “Early Warning”
- Programmable internal or external reference
  - Up to 14 internal reference voltages (2 - 4.77V)
- Operates during SLEEP
  - Low Voltage condition wakes-up/interrupts MCU
- Software Controlled enable/disable
  - Useful for low power applications

PIC18 Special Features

Programmable Brown-Out RESET

- Monitors operating voltage range
- Resets MCU when Vdd is below reference voltage
  - Deasserts RESET after Vdd is above reference voltage
  - Programmable internal reference
    - Up to 4 voltages (2.0, 2.7, 4.2, 4.5)
- Enabled via Configuration register

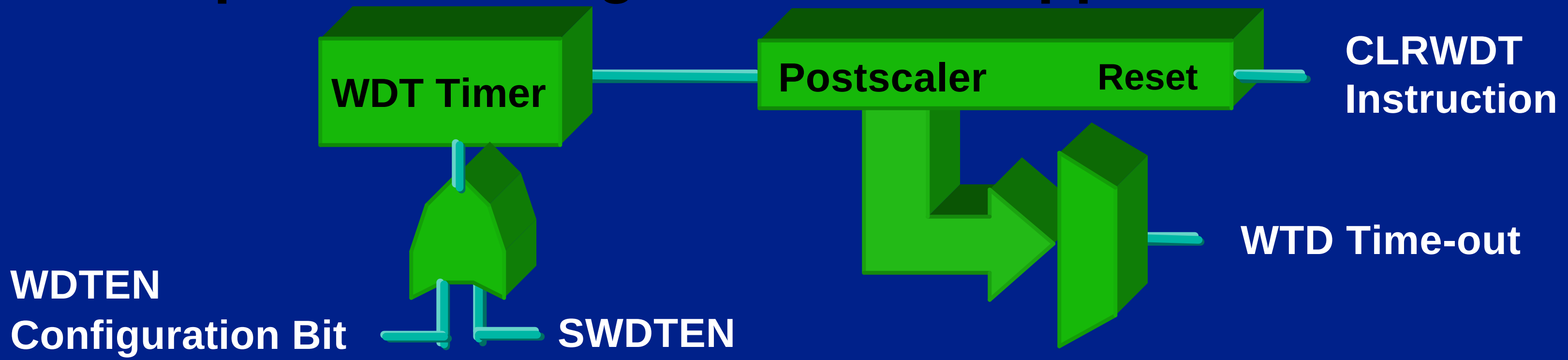
PIC18 Special Features

Watchdog Timer (WDT)

- Recovers from software malfunction
- Resets MCU if not attended on-time
  - Software must clear it periodically (**`CLRWDT`**)
- Programmable period
  - 18 ms to 3.0 s typical
- Configuration controlled postscaler
- Enabled via Configuration register or Software

Watchdog Enhancements Block Diagram

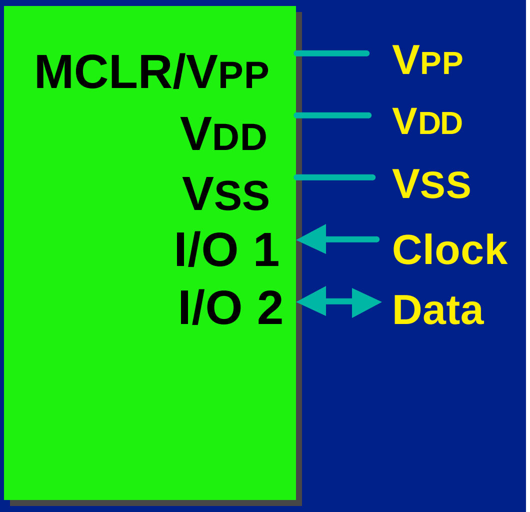
- The watchdog can be programmed on and off in software
  - **If Configuration bit WDTE = 1, the WDT cannot be turned off in software**
  - **If Configuration bit WDTE = 0, the software watchdog bit SWDTEN, can be used to enable/disable the WDOG timer**
  - **This is useful for applications that want to conserve power by turning off the WDT while in sleep or executing non-critical application code**



PIC18 Special Features

In-Circuit Serial Programming™

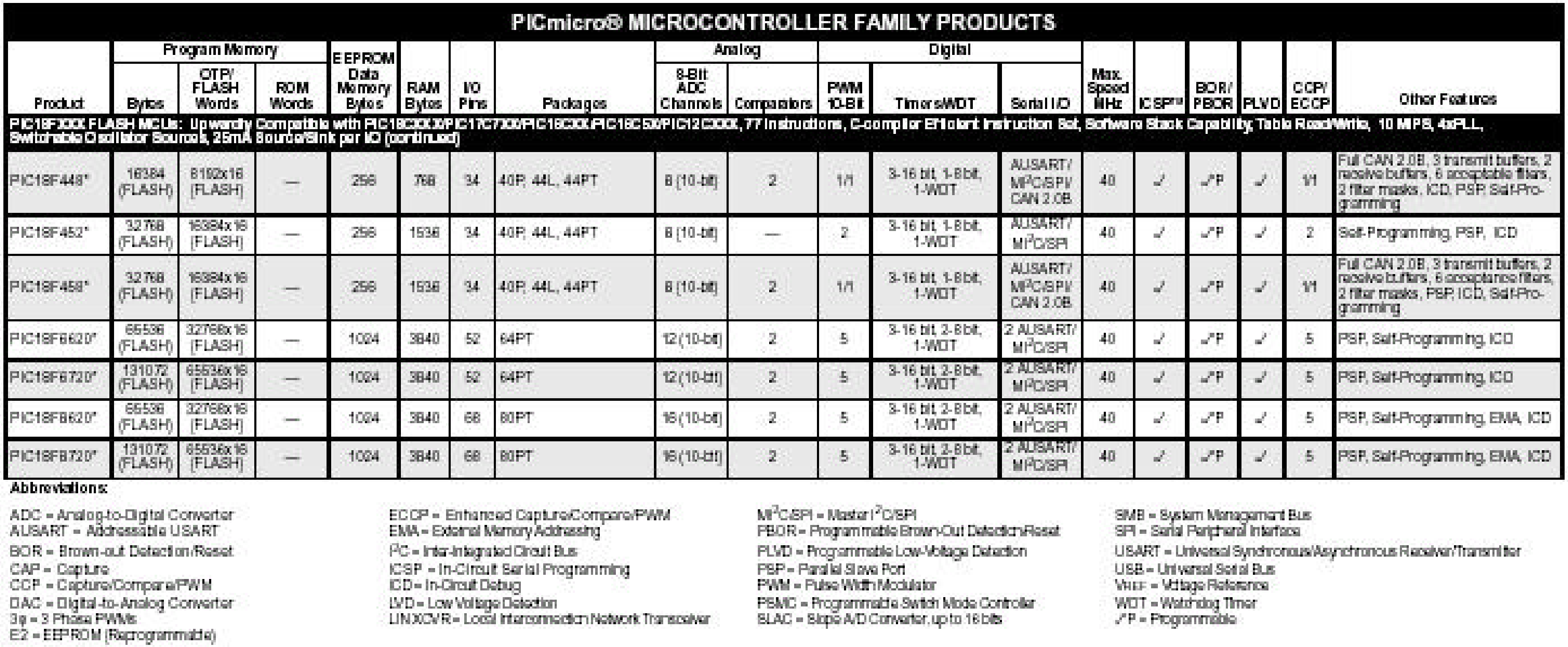
- Enhanced In-System Programming Method
- Uses only two pins to send/receive data
- Non-intrusive to normal operation
- Advantages of ICSP™ programming mode
  - Reduce cost of field upgrades
  - Calibrate and Serialize Systems during manufacturing
  - Reduce handling: Important for DIE and fine lead package



PICmicro Line Card

PIC18FXXX Product Migration

PIC18FXXX Product Line Card



PICmicro® MICROCONTROLLER FAMILY PRODUCTS

	Program Memory							Analog		Digital								
Product	Bytes	OTP/ FLASH Words	ROM Words	EEPROM Data Memory Bytes	RAM Bytes	I/O Pins	Packages	8-Bit ADC Channels	Comparators	PWM 10-Bit	Timers/WDT	Serial I/O	Max. Speed MHz	ICSP™	BOR/ PBOR	PLVD	CCP/ ECCP	Other Features
PIC18FXXX FLASH MCUs: Upwardly Compatible with PIC18CXXX/PIC17C7XX/PIC16CXXX/PIC16C5X/PIC12CXXX, 77 Instructions, C-compiler Efficient Instruction Set, Software Stack Capability, Table Read/Write, 10 MIPS, 4xPLL, Switchable Oscillator Sources, 25mA Source/Sink per I/O (continued)																		
PIC18F448*	16384 (FLASH)	8192x16 (FLASH)	—	256	768	34	40P, 44L, 44PT	8 (10-bit)	2	1/1	3-16 bit, 1-8bit, 1-WDT	AUSART/ MI²C/SPI/ CAN 2.0B	40	✓	✓P	✓	1/1	Full CAN 2.0B, 3 transmit buffers, 2 receive buffers, 6 acceptable filters, 2 filter masks, ICD, PSP, Self-Programming
PIC18F452*	32768 (FLASH)	16384x16 (FLASH)	—	256	1536	34	40P, 44L, 44PT	8 (10-bit)	—	2	3-16 bit, 1-8bit, 1-WDT	AUSART/ MI²C/SPI	40	✓	✓P	✓	2	Self-Programming, PSP, ICD
PIC18F458*	32768 (FLASH)	16384x16 (FLASH)	—	256	1536	34	40P, 44L, 44PT	8 (10-bit)	2	1/1	3-16 bit, 1-8bit, 1-WDT	AUSART/ MI²C/SPI/ CAN 2.0B	40	✓	✓P	✓	1/1	Full CAN 2.0B, 3 transmit buffers, 2 receive buffers, 6 acceptance filters, 2 filter masks, PSP, ICD, Self-Programming
PIC18F6620*	65536 (FLASH)	32768x16 (FLASH)	—	1024	3840	52	64PT	12 (10-bit)	2	5	3-16 bit, 2-8bit, 1-WDT	2 AUSART/ MI²C/SPI	40	✓	✓P	✓	5	PSP, Self-Programming, ICD
PIC18F6720*	131072 (FLASH)	65536x16 (FLASH)	—	1024	3840	52	64PT	12 (10-bit)	2	5	3-16 bit, 2-8bit, 1-WDT	2 AUSART/ MI²C/SPI	40	✓	✓P	✓	5	PSP, Self-Programming, ICD
PIC18F8620*	65536 (FLASH)	32768x16 (FLASH)	—	1024	3840	68	80PT	16 (10-bit)	2	5	3-16 bit, 2-8bit, 1-WDT	2 AUSART/ MI²C/SPI	40	✓	✓P	✓	5	PSP, Self-Programming, EMA, ICD
PIC18F8720*	131072 (FLASH)	65536x16 (FLASH)	—	1024	3840	68	80PT	16 (10-bit)	2	5	3-16 bit, 2-8bit, 1-WDT	2 AUSART/ MI²C/SPI	40	✓	✓P	✓	5	PSP, Self-Programming, EMA, ICD

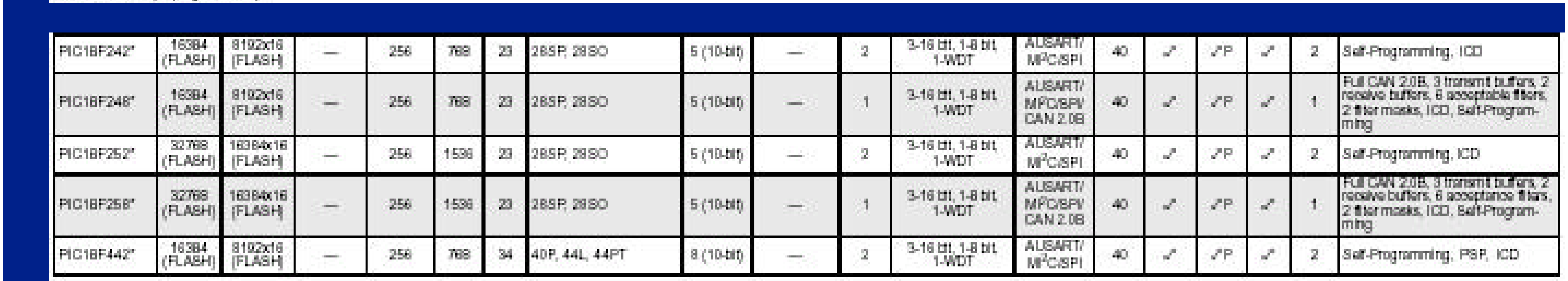
Abbreviations:

ADC = Analog-to-Digital Converter
AUSART = Addressable USART
BOR = Brown-out Detection/Reset
CAP = Capture
CCP = Capture/Compare/PWM
DAC = Digital-to-Analog Converter
3φ = 3 Phase PWMs
E2 = EEPROM (Reprogrammable)

ECCP = Enhanced Capture/Compare/PWM
EMA = External Memory Addressing
I²C = Inter-Integrated Circuit Bus
ICSP = In-Circuit Serial Programming
ICD = In-Circuit Debug
LVD = Low Voltage Detection
LIN XCVR = Local Interconnection Network Transceiver

MI²C/SPI = Master I²C/SPI
PBOR = Programmable Brown-Out Detection/Reset
PLVD = Programmable Low-Voltage Detection
PSP = Parallel Slave Port
PWM = Pulse Width Modulator
PSMC = Programmable Switch Mode Controller
SLAC = Slope A/D Converter, up to 16 bits

SMB = System Management Bus
SPI = Serial Peripheral Interface
USART = Universal Synchronous/Asynchronous Receiver/Transmitter
USB = Universal Serial Bus
VREF = Voltage Reference
WDT = Watchdog Timer
✓P = Programmable

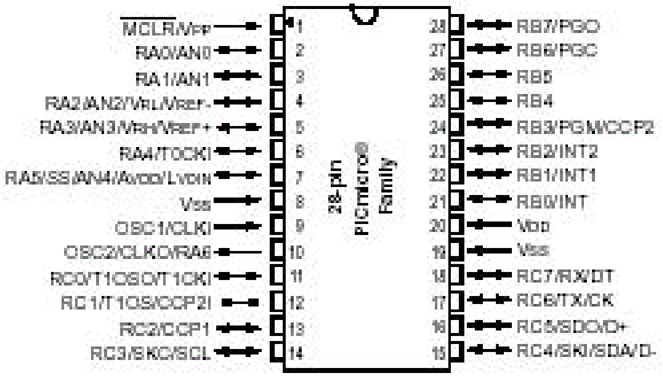


PIC18F242*	16384 (FLASH)	8192x16 (FLASH)	—	256	768	23	28SP, 28SO	5 (10-bit)	—	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI²C/SPI	40	✓	✓P	✓	2	Self-Programming, ICD
PIC18F248*	16384 (FLASH)	8192x16 (FLASH)	—	256	768	23	28SP, 28SO	5 (10-bit)	—	1	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI²C/SPI/ CAN 2.0B	40	✓	✓P	✓	1	Full CAN 2.0B, 3 transmit buffers, 2 receive buffers, 6 acceptable filters, 2 filter masks, ICD, Self-Programming
PIC18F252*	32768 (FLASH)	16384x16 (FLASH)	—	256	1536	23	28SP, 28SO	5 (10-bit)	—	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI²C/SPI	40	✓	✓P	✓	2	Self-Programming, ICD
PIC18F258*	32768 (FLASH)	16384x16 (FLASH)	—	256	1536	23	28SP, 28SO	5 (10-bit)	—	1	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI²C/SPI/ CAN 2.0B	40	✓	✓P	✓	1	Full CAN 2.0B, 3 transmit buffers, 2 receive buffers, 6 acceptance filters, 2 filter masks, ICD, Self-Programming
PIC18F442*	16384 (FLASH)	8192x16 (FLASH)	—	256	768	34	40P, 44L, 44PT	8 (10-bit)	—	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI²C/SPI	40	✓	✓P	✓	2	Self-Programming, PSP, ICD

PICmicro 28/40 Pin Device Compatibility

PIN AND CODE COMPATIBILITY CHART (CONTINUED)

28-pin PICmicro® MCU Family

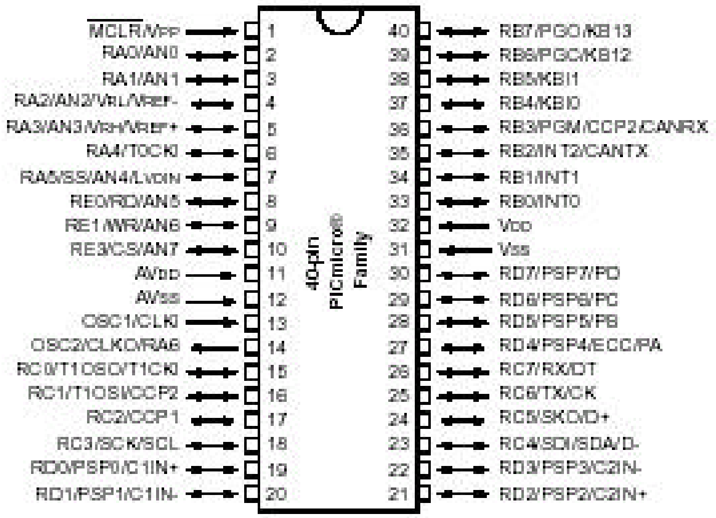


PIC16C62B
PIC16CR63
PIC16C63A
PIC16C66
PIC16CR72
PIC16C72A
PIC16F72
PIC16C73B
PIC16C76
PIC16F73

PIC16F76
PIC16C642
PIC16C745
PIC16C773
PIC16F870
PIC16F872
PIC16F873
PIC16F873A

PIC16F876
PIC16F876A
PIC18C242
PIC18C252
PIC18F2220
PIC18F2320
PIC18F242
PIC18F252
PIC18F248
PIC18F258

40-pin PICmicro® MCU Family

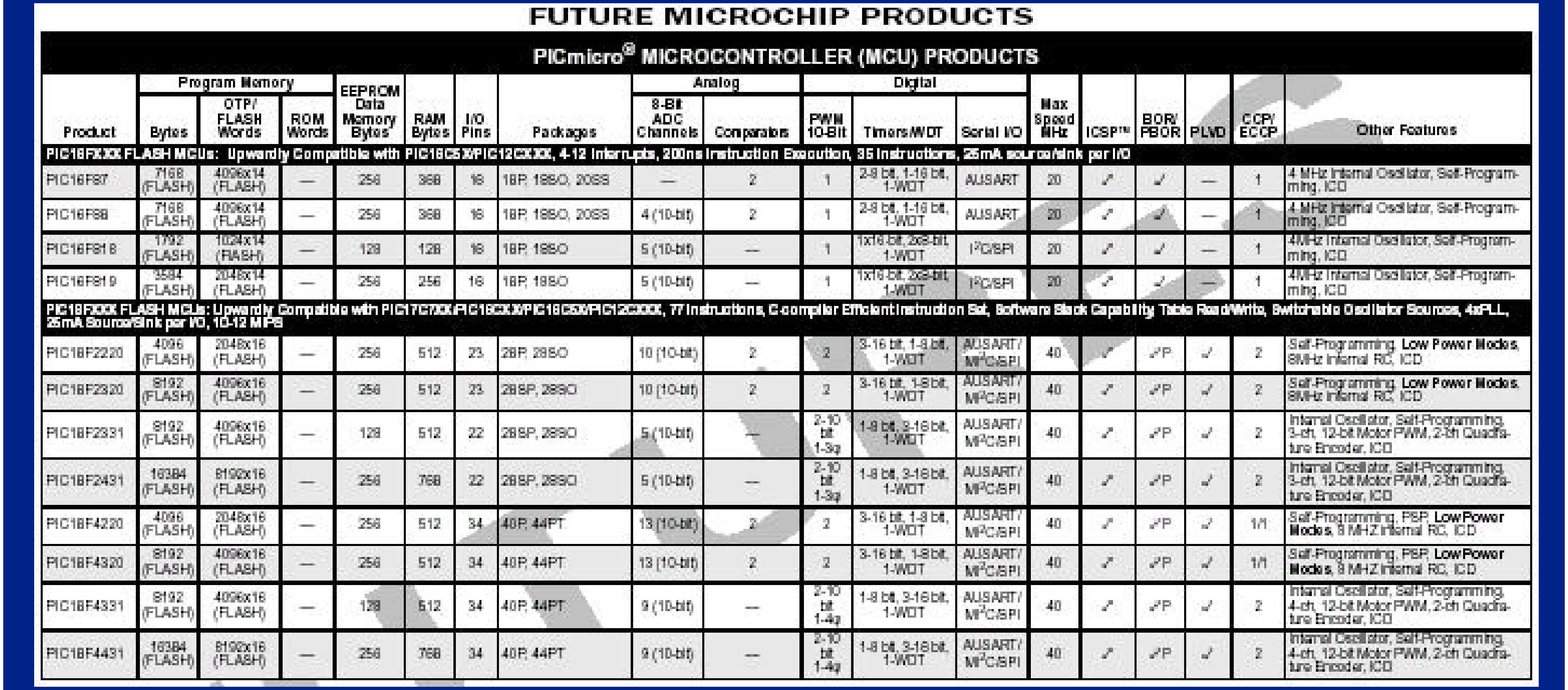


PIC16CR65
PIC16C65B
PIC16C67
PIC16C662
PIC16C74B
PIC16C77
PIC16F74
PIC16F77

PIC16C765
PIC16C774
PIC16F871
PIC16F874
PIC16F874A
PIC16F877
PIC16F877A

PIC18C442
PIC18C452
PIC18F4220
PIC18F4320
PIC18F442
PIC18F452
PIC18F448
PIC18F458

Future Products



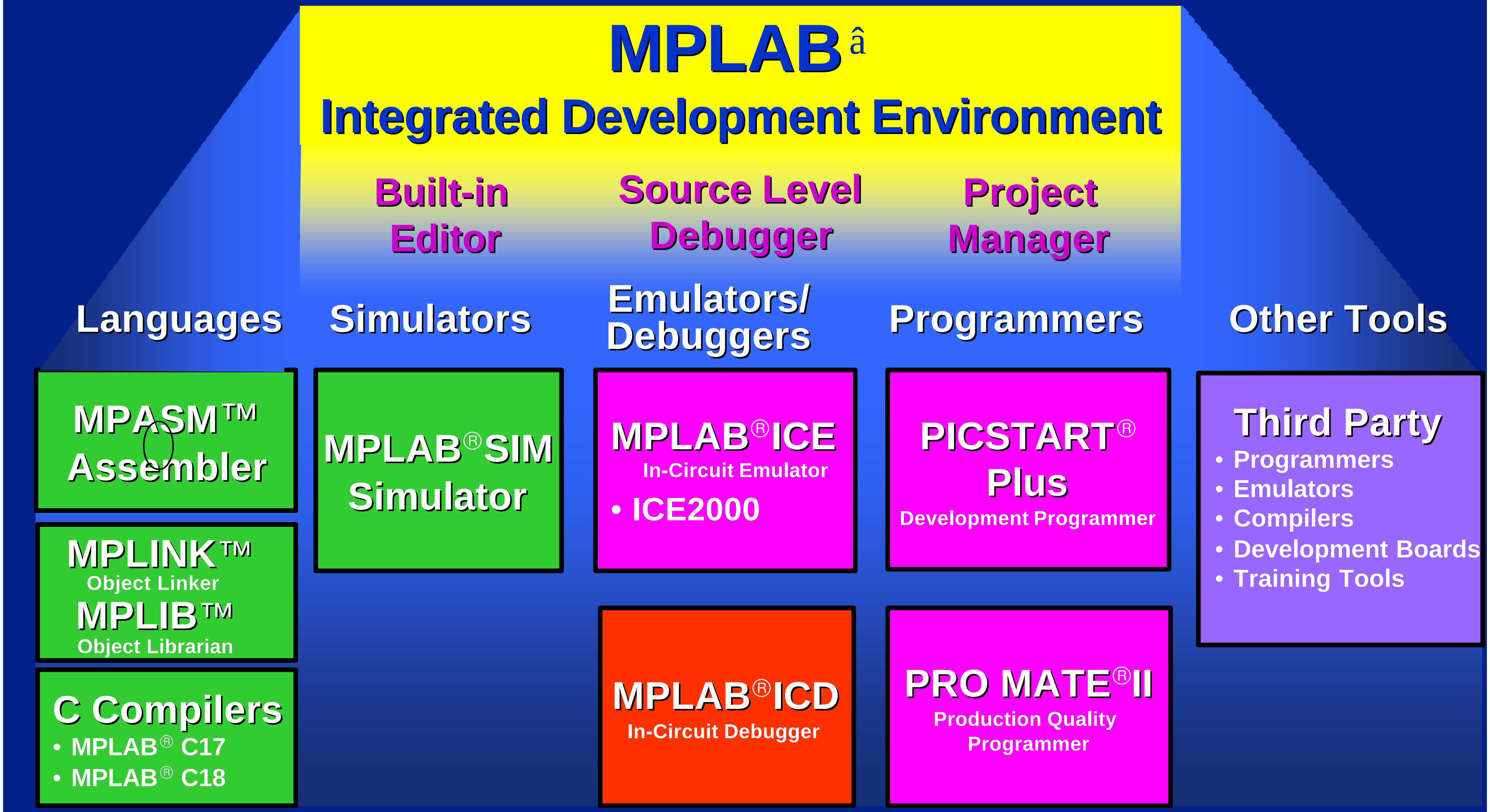
FUTURE MICROCHIP PRODUCTS

PICmicro® MICROCONTROLLER (MCU) PRODUCTS

Product	Program Memory			EEPROM Data Memory Bytes	RAM Bytes	I/O Pins	Packages	Analog		Digital			Max Speed MHz	ICSP™	BOR/ PBOR	PLVD	CCP/ ECCP	Other Features
	Bytes	OTP/ FLASH Words	ROM Words					8-Bit ADC Channels	Comparators	PWM 10-Bit	Timers/WDT	Serial I/O						
PIC16FXXX FLASH MCUs: Upwardly Compatible with PIC16C5X/PIC12CXXX, 4-12 Interrupts, 200ns Instruction Execution, 35 Instructions, 25mA source/sink per I/O																		
PIC16F87	7168 (FLASH)	4096x14 (FLASH)	—	256	368	16	18P, 18SO, 20SS	—	2	1	2-8 bit, 1-16 bit, 1-WDT	AUSART	20	✓	✓	—	1	4 MHz Internal Oscillator, Self-Programming, ICD
PIC16F88	7168 (FLASH)	4096x14 (FLASH)	—	256	368	16	18P, 18SO, 20SS	4 (10-bit)	2	1	2-8 bit, 1-16 bit, 1-WDT	AUSART	20	✓	✓	—	1	4 MHz Internal Oscillator, Self-Programming, ICD
PIC16F818	1792 (FLASH)	1024x14 (FLASH)	—	128	128	16	18P, 18SO	5 (10-bit)	—	1	1x16-bit, 2x8-bit, 1-WDT	I^{2}C/SPI	20	✓	✓	—	1	4MHz Internal Oscillator, Self-Programming, ICD
PIC16F819	3584 (FLASH)	2048x14 (FLASH)	—	256	256	16	18P, 18SO	5 (10-bit)	—	1	1x16-bit, 2x8-bit, 1-WDT	I^{2}C/SPI	20	✓	✓	—	1	4MHz Internal Oscillator, Self-Programming, ICD
PIC18FXXX FLASH MCUs: Upwardly Compatible with PIC17C7XX/PIC16CXXX/PIC18CXXX/PIC12CXXX, 77 Instructions, C-compiler Efficient Instruction Set, Software Stack Capability Table Read/Write, Switchable Oscillator Sources, 4xPLL, 25mA Source/Sink per I/O, 10-12 MIPS																		
PIC18F2220	4096 (FLASH)	2048x16 (FLASH)	—	256	512	23	28P, 28SO	10 (10-bit)	2	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Self-Programming, **Low Power Modes**, 8MHz Internal RC, ICD
PIC18F2320	8192 (FLASH)	4096x16 (FLASH)	—	256	512	23	28SP, 28SO	10 (10-bit)	2	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Self-Programming, **Low Power Modes**, 8MHz Internal RC, ICD
PIC18F2331	8192 (FLASH)	4096x16 (FLASH)	—	128	512	22	28SP, 28SO	5 (10-bit)	—	2-10 bit 1-3g	1-8 bit, 3-16 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Internal Oscillator, Self-Programming, 3-ch, 12-bit Motor PWM, 2-ch Quadrature Encoder, ICD
PIC18F2431	16384 (FLASH)	8192x16 (FLASH)	—	256	768	22	28SP, 28SO	5 (10-bit)	—	2-10 bit 1-3g	1-8 bit, 3-16 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Internal Oscillator, Self-Programming, 3-ch, 12-bit Motor PWM, 2-ch Quadrature Encoder, ICD
PIC18F4220	4096 (FLASH)	2048x16 (FLASH)	—	256	512	34	40P, 44PT	13 (10-bit)	2	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	1/1	Self-Programming, PSP, **Low Power Modes**, 8 MHZ Internal RC, ICD
PIC18F4320	8192 (FLASH)	4096x16 (FLASH)	—	256	512	34	40P, 44PT	13 (10-bit)	2	2	3-16 bit, 1-8 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	1/1	Self-Programming, PSP, **Low Power Modes**, 8 MHZ Internal RC, ICD
PIC18F4331	8192 (FLASH)	4096x16 (FLASH)	—	128	512	34	40P, 44PT	9 (10-bit)	—	2-10 bit 1-4g	1-8 bit, 3-16 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Internal Oscillator, Self-Programming, 4-ch, 12-bit Motor PWM, 2-ch Quadrature Encoder, ICD
PIC18F4431	16384 (FLASH)	8192x16 (FLASH)	—	256	768	34	40P, 44PT	9 (10-bit)	—	2-10 bit 1-4g	1-8 bit, 3-16 bit, 1-WDT	AUSART/ MI^{2}C/SPI	40	✓	✓P	✓	2	Internal Oscillator, Self-Programming, 4-ch, 12-bit Motor PWM, 2-ch Quadrature Encoder, ICD

Development Tools
MPLAB-IDE V6.0
MPLAB-ICD II
MPLAB C18

PICmicro® Microcontroller Development Tools



MPLABâ

Integrated Development Environment

Built-in Editor

Source Level Debugger

Project Manager

Languages

MPASM™ Assembler

MPLINK™
Object Linker

MPLIB™
Object Librarian

C Compilers
- MPLAB® C17
- MPLAB® C18

Simulators

MPLAB® SIM Simulator

Emulators/ Debuggers

MPLAB® ICE
In-Circuit Emulator
- ICE2000

MPLAB® ICD
In-Circuit Debugger

Programmers

PICSTART® Plus
Development Programmer

PRO MATE® II
Production Quality Programmer

Other Tools

Third Party
- Programmers
- Emulators
- Compilers
- Development Boards
- Training Tools

New MPLAB® V6.00

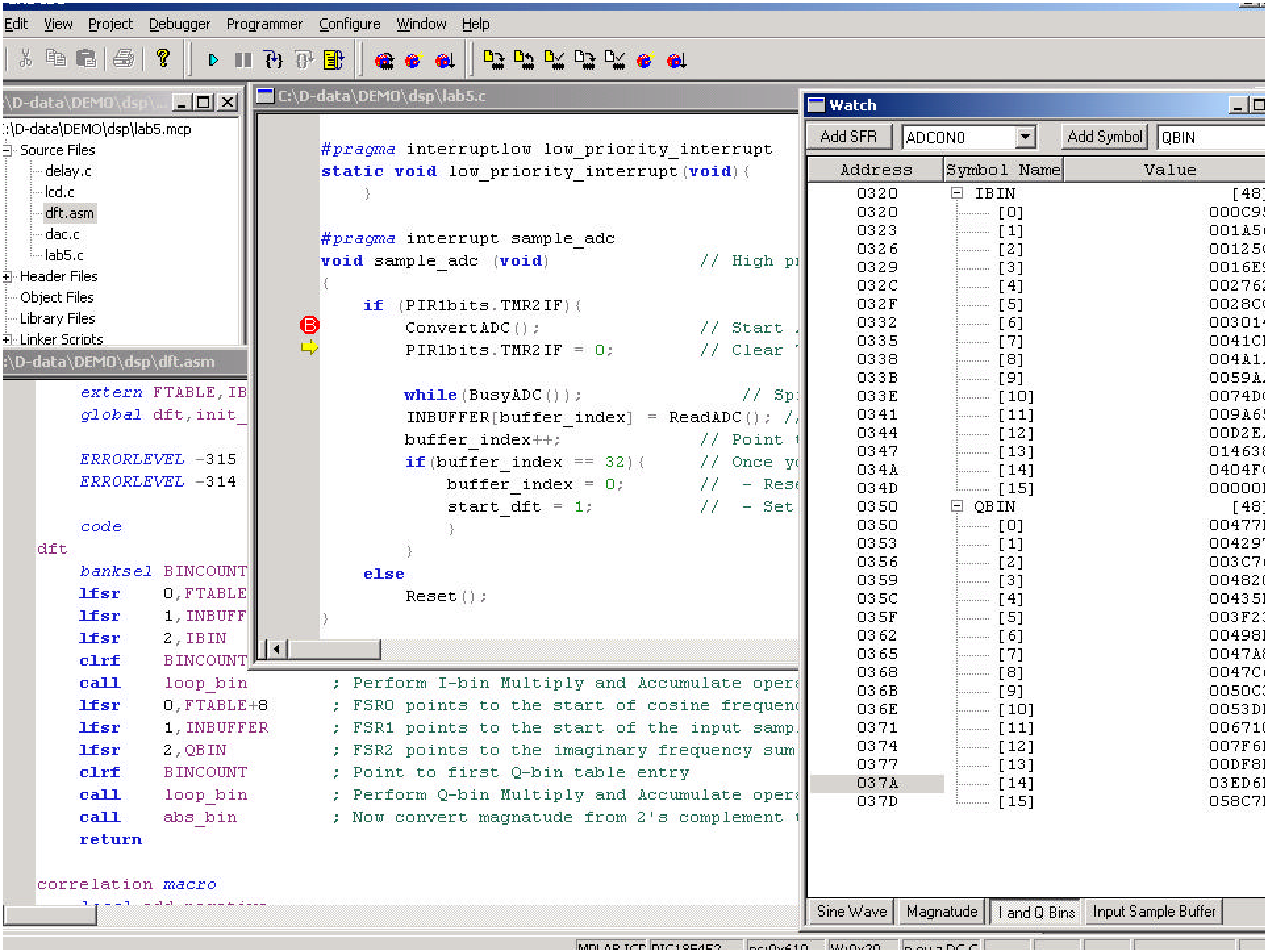
Native Windows / 32-Bit implementation

- Color Coded Context Sensitive Text Editor
- Relocatable projects with Win/32 long file names
- Automatic C variable sizing in watch windows
- Arrays and Structures views in watch windows
- Modify file registers within watch windows
- Breakpoint settings persistence
- Improved MPLAB-SIM Simulator speed
- Advanced project manager
- Full Speed USB interface for MPLAB-ICD2
- MPLAB-ICE2000 emulator PROMATE-II programmer and PICSTART PLUS programmer support

MPLAB® V6.10 Release

New features planned for this fall..

- Multi- language tools capability with Standardized 3rd party Compiler Interface (Hi-Tech, IAR, CCS)
- V6.00 supports PIC18C/FXXXX and dsPIC30F devices and MPLAB C18 compilers only
- V6.10 adds support for all PIC12/16C/FXXX and 3rd party compilers like Hi-Tech, IAR and CCS



```
#pragma interruptlow low_priority_interrupt
static void low_priority_interrupt(void){
    }

#pragma interrupt sample_adc
void sample_adc (void)                      // High pr
{
    if (PIR1bits.TMR2IF){
        ConvertADC();                       // Start .
        PIR1bits.TMR2IF = 0;                // Clear '

        while(BusyADC());                       // Sp:
        INBUFFER[buffer_index] = ReadADC(); //
        buffer_index++;                     // Point 1
        if(buffer_index == 32){             // Once yo
            buffer_index = 0;               //  - Rese
            start_dft = 1;                  //  - Set
            }
        }
    else
        Reset();
}
```

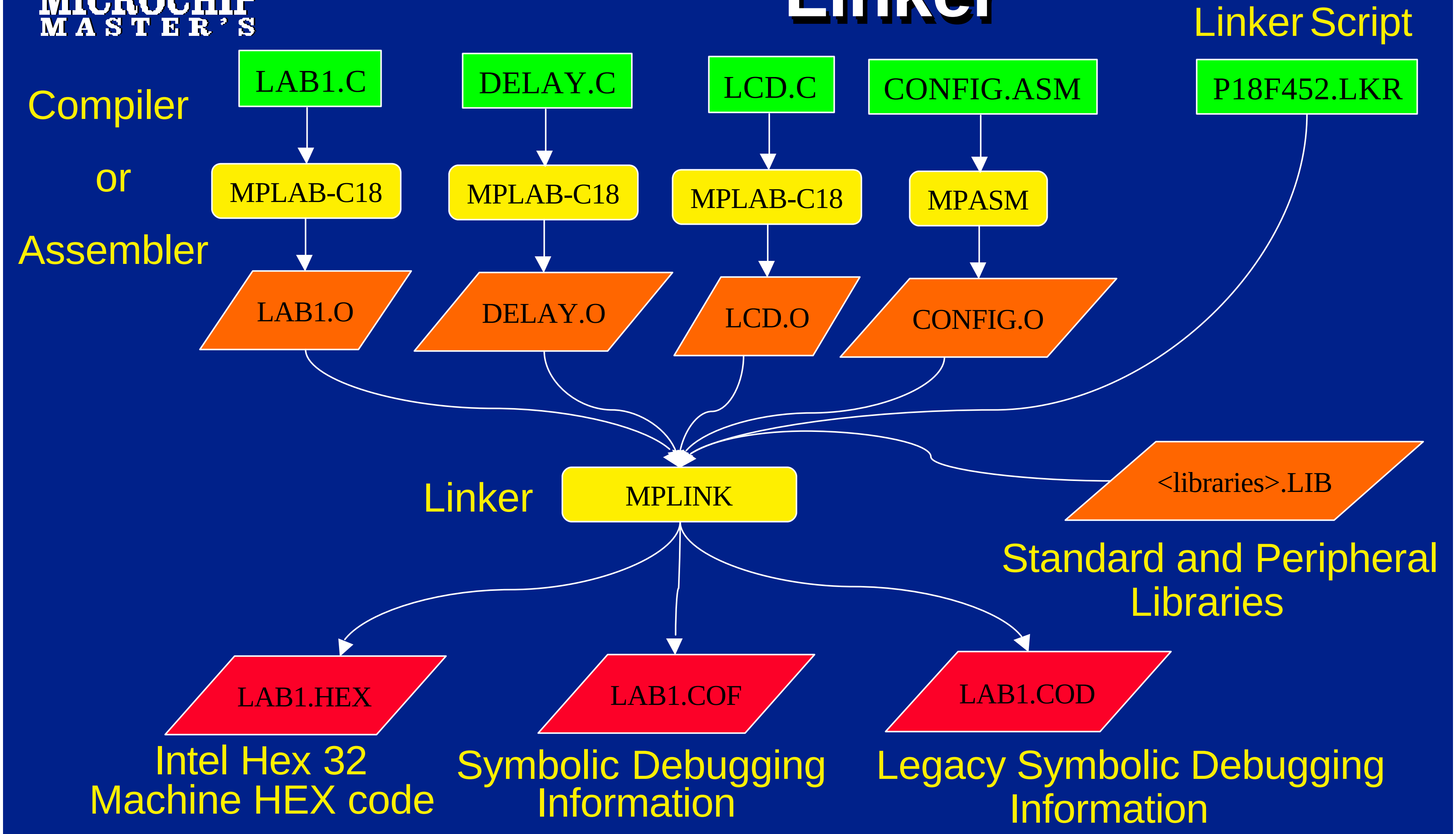
```
    extern FTABLE,IB
    global dft,init_

    ERRORLEVEL -315
    ERRORLEVEL -314

    code
dft
    banksel BINCOUNT
    lfsr    0,FTABLE
    lfsr    1,INBUFF
    lfsr    2,IBIN
    clrf    BINCOUNT
    call    loop_bin        ; Perform I-bin Multiply and Accumulate opera
    lfsr    0,FTABLE+8      ; FSR0 points to the start of cosine frequenc
    lfsr    1,INBUFFER      ; FSR1 points to the start of the input samp.
    lfsr    2,QBIN          ; FSR2 points to the imaginary frequency sum
    clrf    BINCOUNT        ; Point to first Q-bin table entry
    call    loop_bin        ; Perform Q-bin Multiply and Accumulate opera
    call    abs_bin         ; Now convert magnatude from 2's complement t
    return

correlation macro
    local add_negative
```

```
  Address   Symbol Name          Value
   0320     ⊟ IBIN                [48]
   0320       [0]                000C9
   0323       [1]                001A5
   0326       [2]                00125
   0329       [3]                0016E
   032C       [4]                00276
   032F       [5]                0028C
   0332       [6]                00301
   0335       [7]                0041C
   0338       [8]                004A1
   033B       [9]                0059A
   033E       [10]               0074D
   0341       [11]               009A6
   0344       [12]               00D2E
   0347       [13]               01463
   034A       [14]               0404F
   034D       [15]               00000
   0350     ⊟ QBIN                [48]
   0350       [0]                00477
   0353       [1]                00429
   0356       [2]                003C7
   0359       [3]                00482
   035C       [4]                00435
   035F       [5]                003F2
   0362       [6]                00498
   0365       [7]                0047A
   0368       [8]                0047C
   036B       [9]                0050C
   036E       [10]               0053D
   0371       [11]               00671
   0374       [12]               007F6
   0377       [13]               00DF8
   037A       [14]               03ED6
   037D       [15]               058C7
```

MPLAB-C18 C Compiler

- ANSI compatible Microchip developed compiler for PIC18FXXX devices
- Compatible with MPASM assembler, MPLINK linker and MPLIB librarian
  - Compatible at object level
  - Supports relocatable objects
  - Effortlessly mix C and Assembly source files
- Full Source Level Debugging using MPLAB V6.0
- Free 30 day copies available on the web **www.microchip.com**

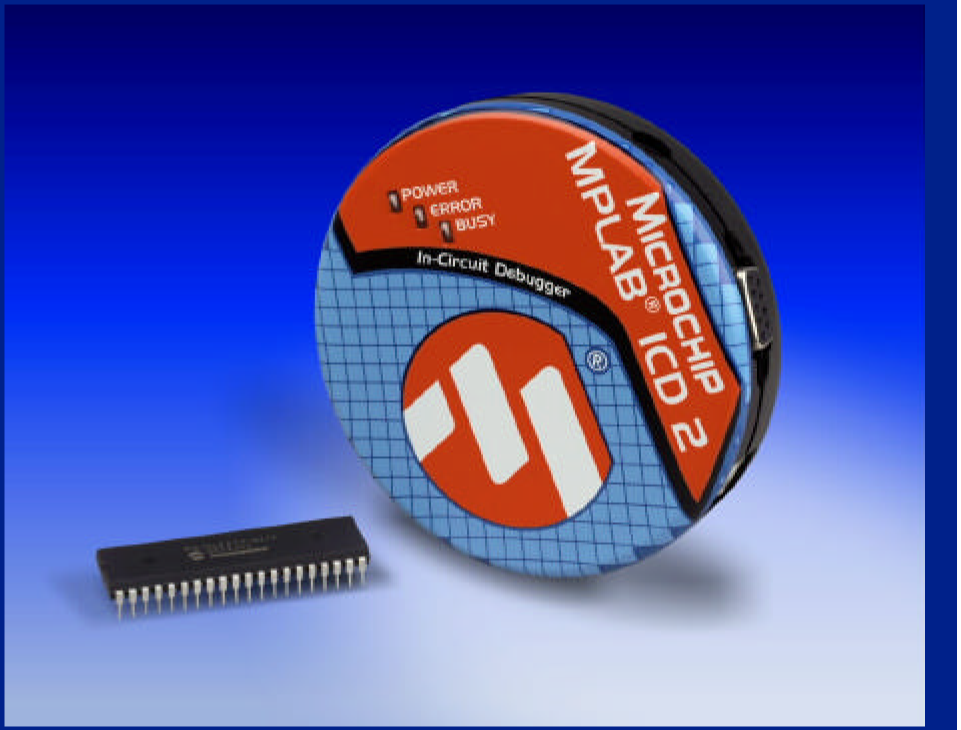
Compiler, Assembler and Linker



In Circuit Debugger MPLAB-ICD 2

DV164005	ICD 2 Module	$159 USD
DV164006	ICD 2 Module + PICDEM ä II+	$209 USD

- Full Speed 12 Mb/s USB PC interface
  - Powered supplied by USB port
- In System Serial Programmer
- In Circuit Real-time (C and Assembly) Source Code Debugger Supporting:
  - Single step through C and Assembly
  - Examine and Modify all internal RAM and Peripheral Registers
  - Program Memory Breakpoint
  - Full speed code execution with target clock and peripherals



MPLAB ICD 2 Options

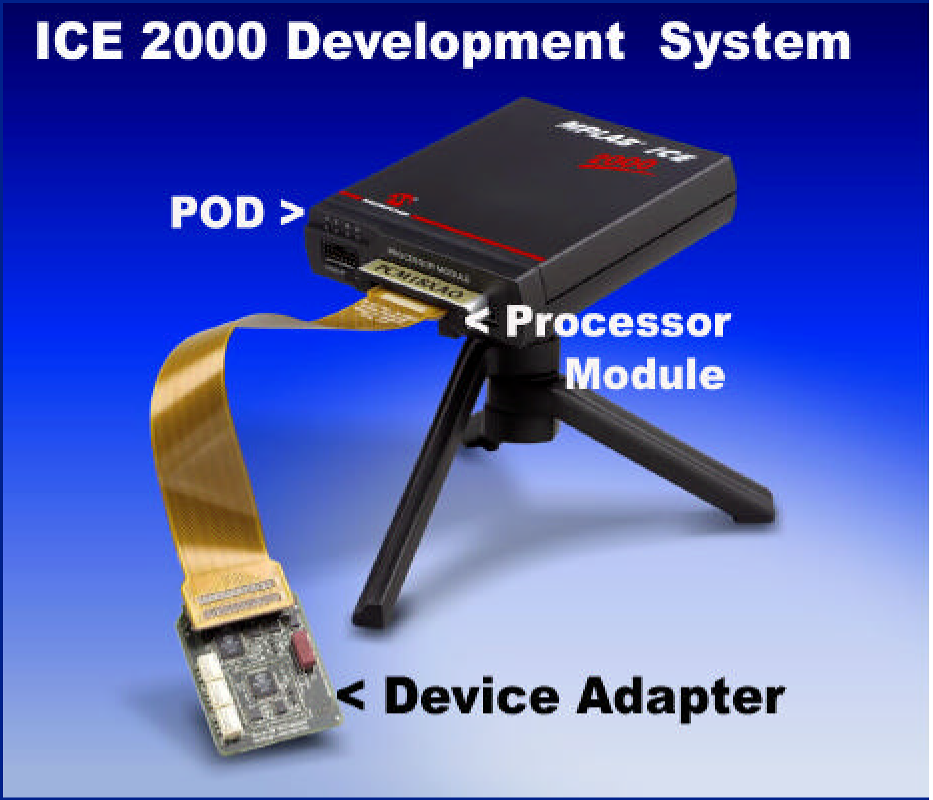
- Programmer board enables use as a universal PICmicro programmer
  - Can replace your PICSTART PLUS programmer
- RS-232 interface and power supply for legacy PCs without USB support

DV162049 ICD 2 Universal Programming Module	**$39 USD**
DV164007 ICD 2 Module + RS232 + Power Supply	**$188 USD**

MPLAB-ICE 2000

In Circuit Emulator

- Unlimited Program Breakpoints
- Trigger and Break on Data Memory Read / Write
- (4) Individual Trigger Events
- Programmable System Oscillator 32Khz ~ 25 Mhz
- Code Coverage
- Pass Counter
- 32K Trace Buffer traces program and data memory



MPLAB ICE 2000 Connectivity

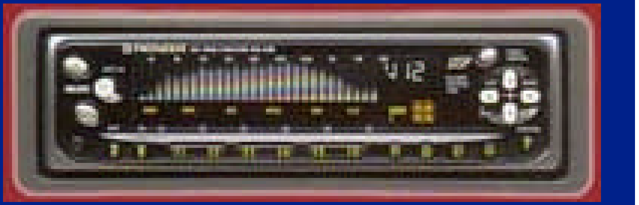
- Universal pod supports PIC12/16/18
- Processor module supports device family
- Device Adapter supports package type
- Transition Sockets support Surface Mount
- Parallel Port PC interface
- ~$2,000 for complete system

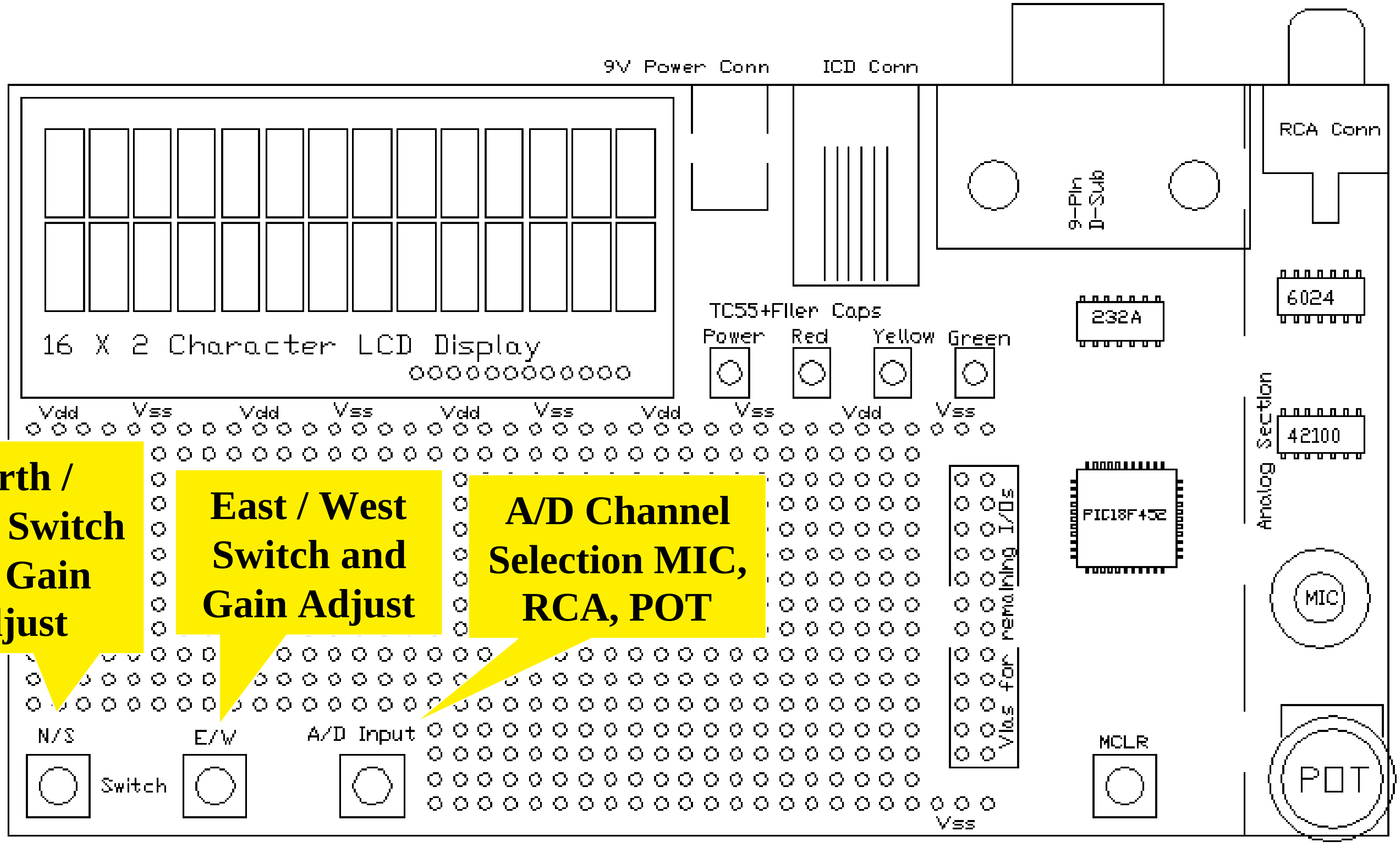
MPLAB V6.0, MPLAB-ICD-II PIC18FXXX Hands On Exercises

Hands On Exercises Agenda

- Lab 1 - Install MPLAB 6.0, MPLAB-ICD II, MPLAB-C18, Connect Demo board
  - Create Project, Compile, Download Code, Get First Demo Up and Running, MPLAB basics
- Lab 2 - Develop and Debug a traffic light
- Lab 3 - Develop and Debug A/D sampling ISR
- Lab 4 - Run DFT() algorithm on A/D Sampling Buffer results and pass array to display routine for graphing
- Lab 5 - Extra credit- Add Automatic Gain Control using SPI controlled Digital POT

Audio Spectrum Analyzer Board



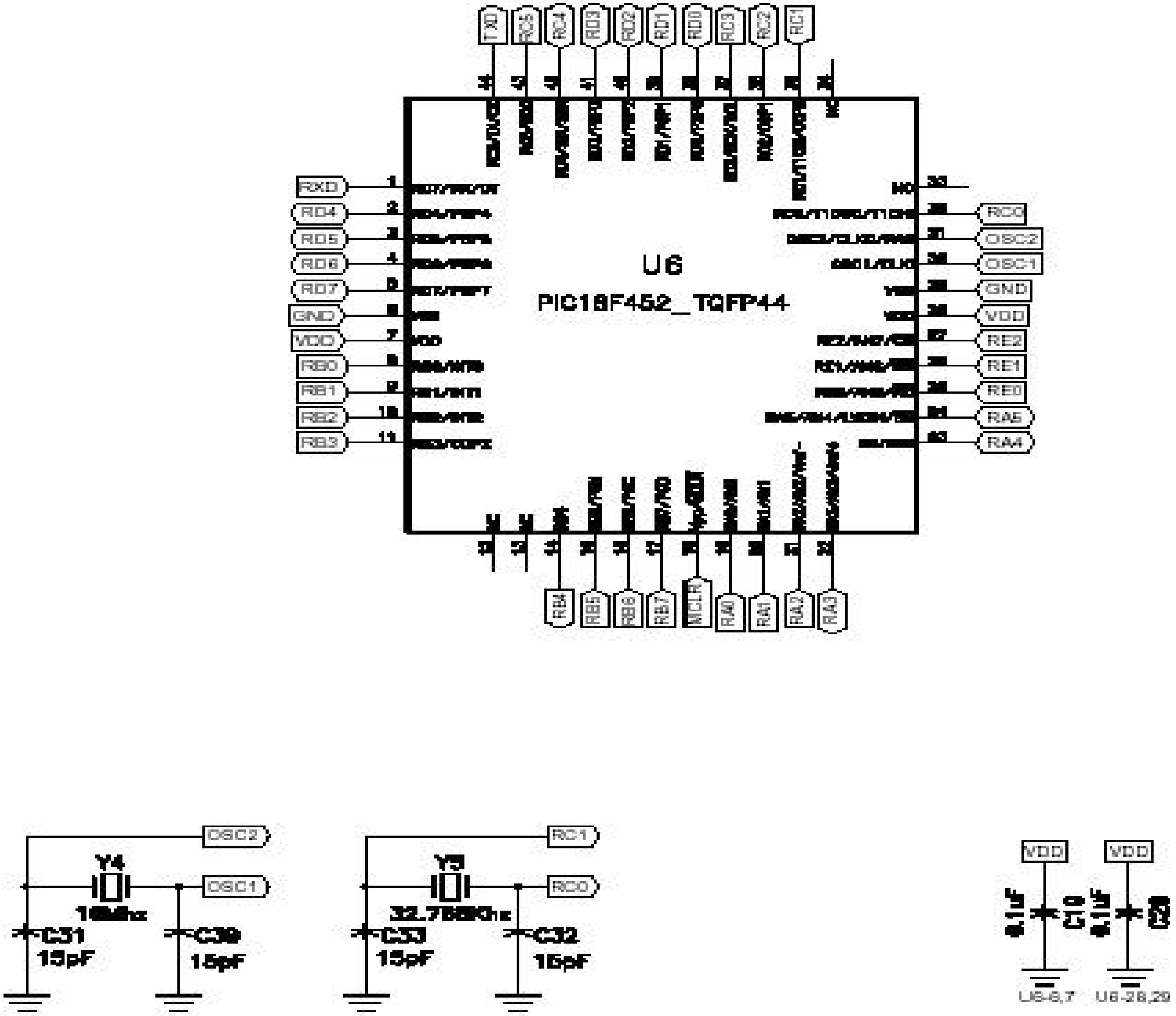


North / South Switch and Gain Adjust

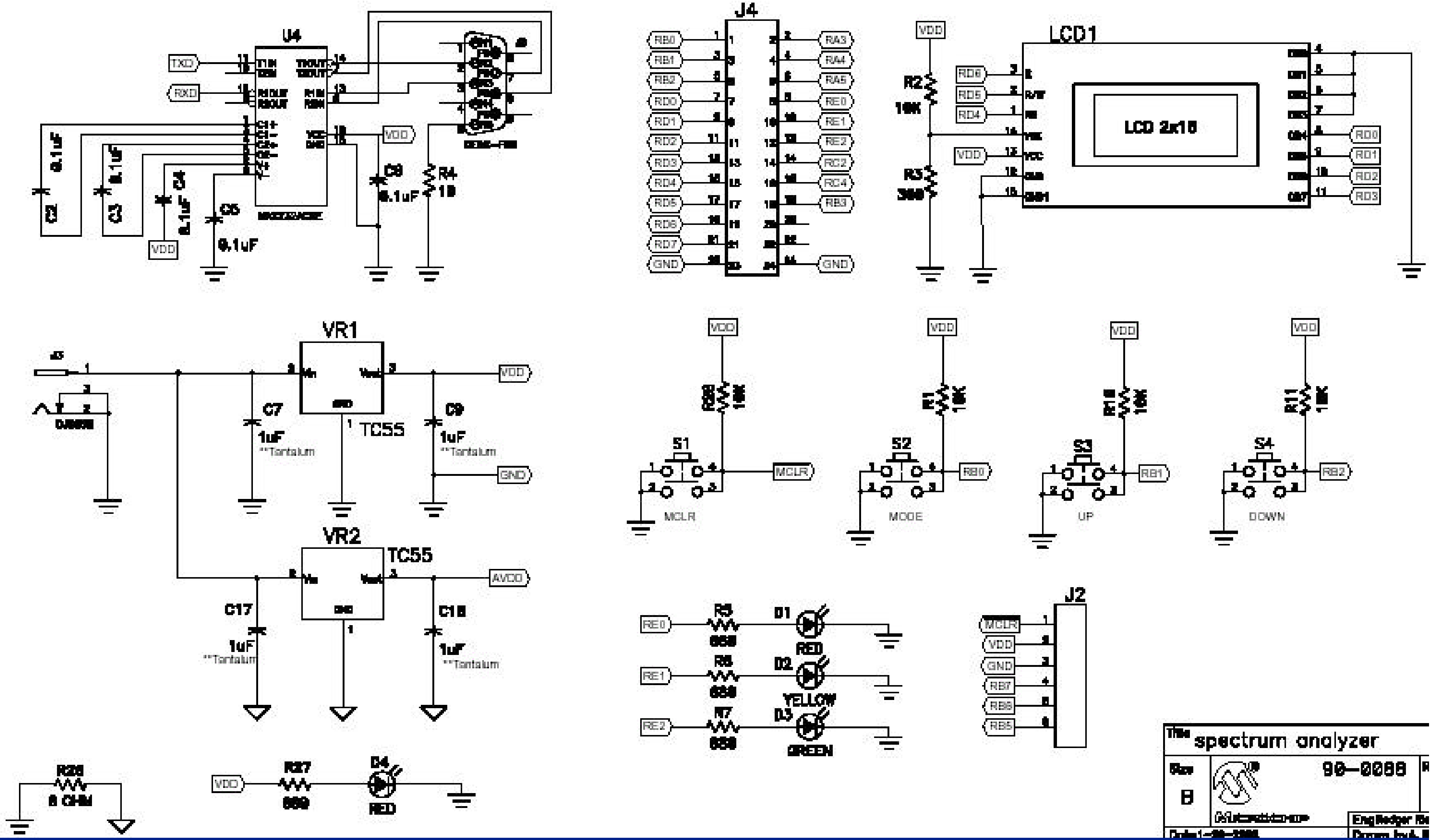
East / West Switch and Gain Adjust

A/D Channel Selection MIC, RCA, POT

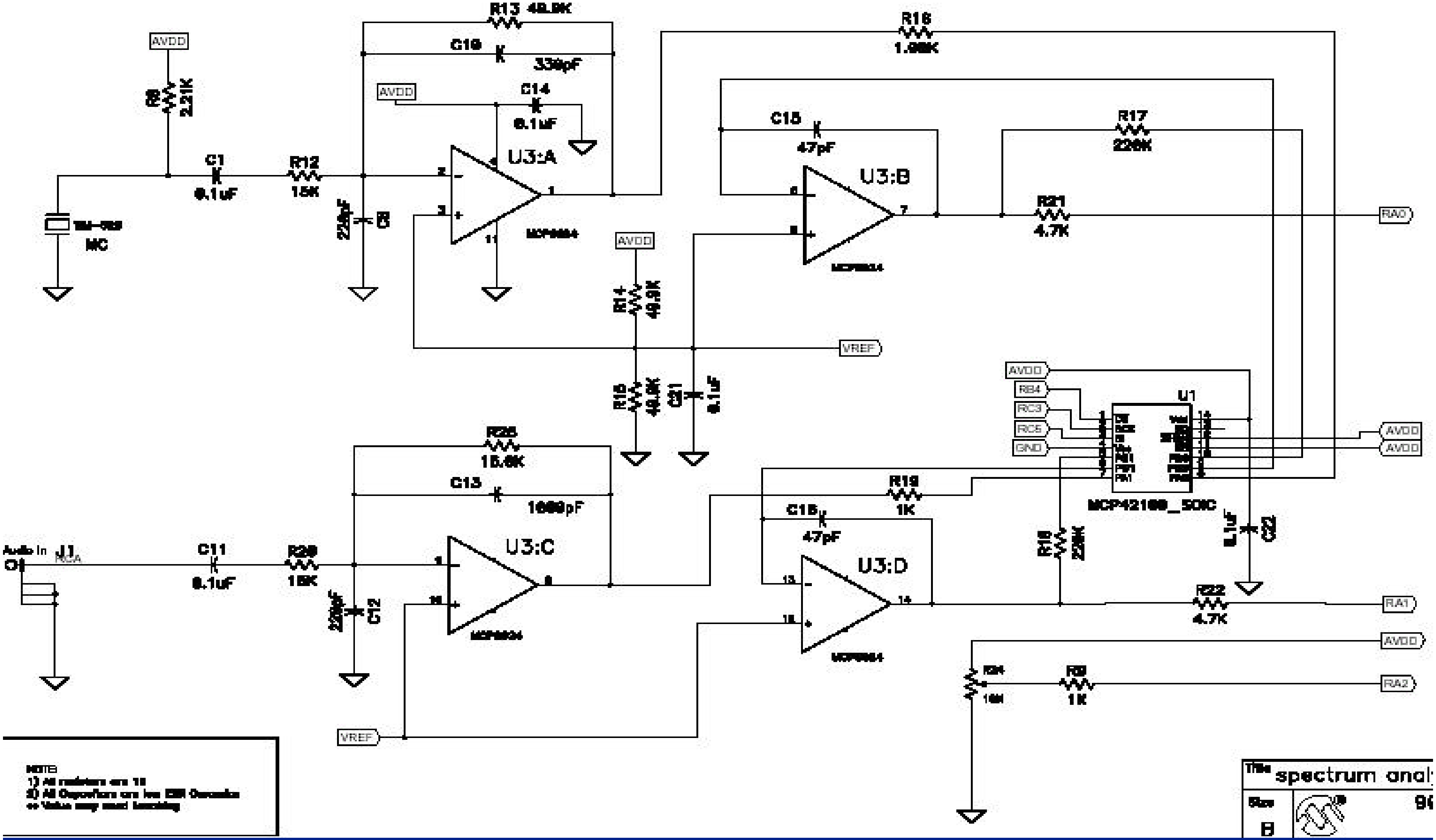
Schematics, Page 1



Schematics, Page 2



Schematics, Page 3



Install MPLAB V6.00, MPLAB-C18, MPLAB-ICD-II

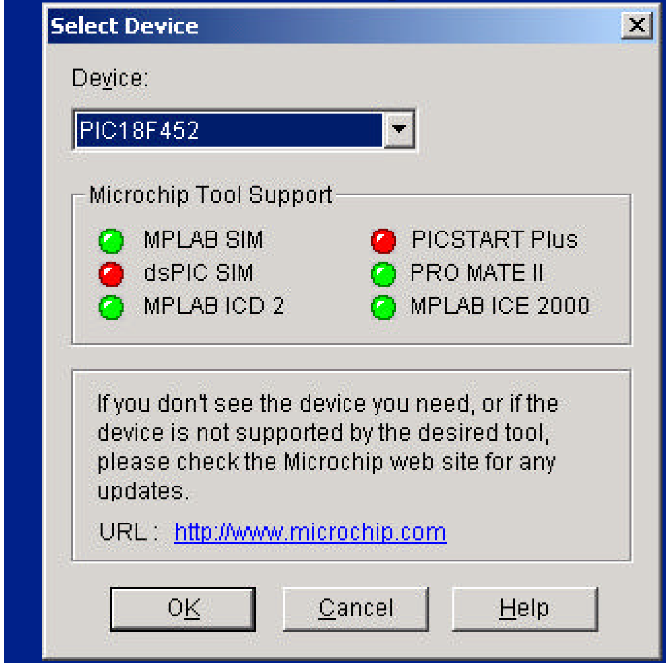
- Step 1: Connect USB cable of MPLAB ICD 2
- Step 2: Connect power supply to Workshop target board
- Step 3: Install MPLAB IDE, ICD 2, C18
  - • Run “MPLIDEV6.EXE” (MPLAB/32 IDE)
  - • Run “MPICD2.EXE” (MPLAB ICD 2)
  - • Re-boot after MPLAB IDE detect ICD 2
  - • Run “MCC18V20.EXE” (MPLAB C18)

Install Workshop Files

- **Step 4: Install workshop files**
  - • Run Workshop.bat, installing workshop files in default directory C:/workshop
- Step 5: Create your first Project
  - Verify MPLAB C18 Installation and Paths
  - Create Project, add source files
  - Build Project
  - Download HEX to target, Run
    - LCD display should show a message...

Setting Up MPLAB V6.0

- Configure -> Select Device -> PIC18F452



Device: PIC18F452

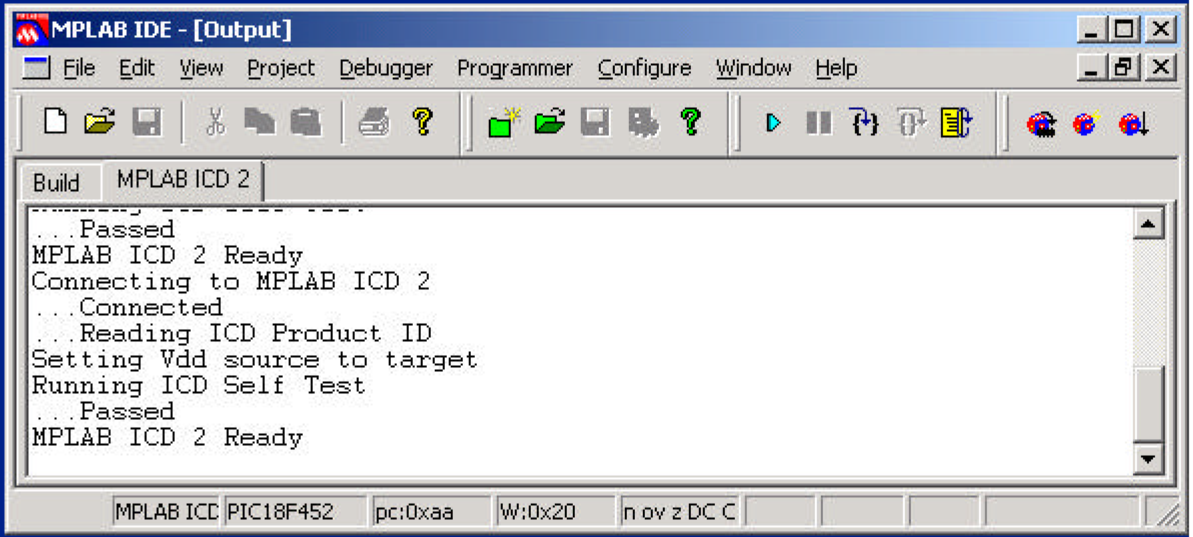
Microchip Tool Support

🟢 MPLAB SIM	🔴 PICSTART Plus
🔴 dsPIC SIM	🟢 PRO MATE II
🟢 MPLAB ICD 2	🟢 MPLAB ICE 2000

If you don't see the device you need, or if the device is not supported by the desired tool, please check the Microchip web site for any updates.

URL : http://www.microchip.com

- Debugger -> Select Tool -> MPLAB-ICD 2

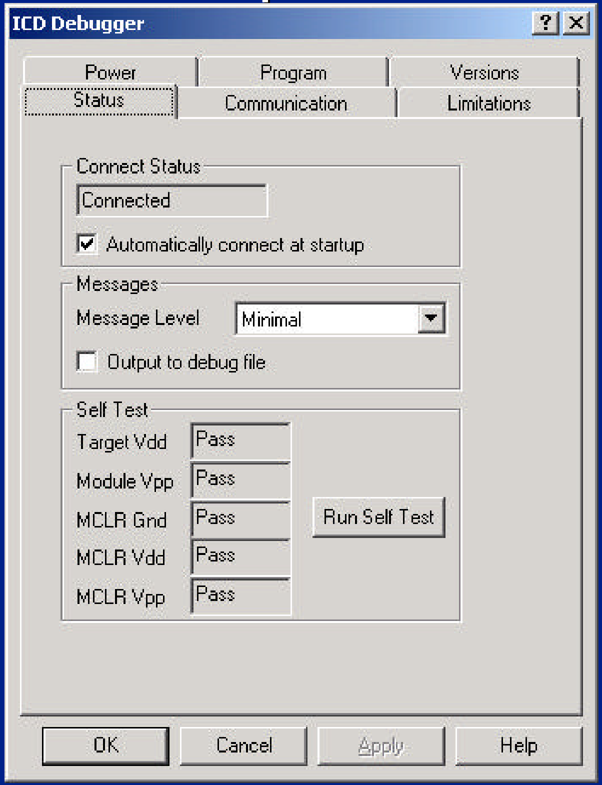


```
...Passed
MPLAB ICD 2 Ready
Connecting to MPLAB ICD 2
...Connected
...Reading ICD Product ID
Setting Vdd source to target
Running ICD Self Test
...Passed
MPLAB ICD 2 Ready
```

MPLAB ICD PIC18F452 pc:0xaa W:0x20 n ov z DC C

Configuring MPLAB ICD 2

- Debugger -> Settings
  - Status Tab - Check “Automatically connect at startup”
  - Program Tab - Press “Full Range” button and end address set to 0x7DBF



ICD Debugger

Tabs: Power | Program | Versions | Status | Communication | Limitations

Connect Status

Connected

☑ Automatically connect at startup

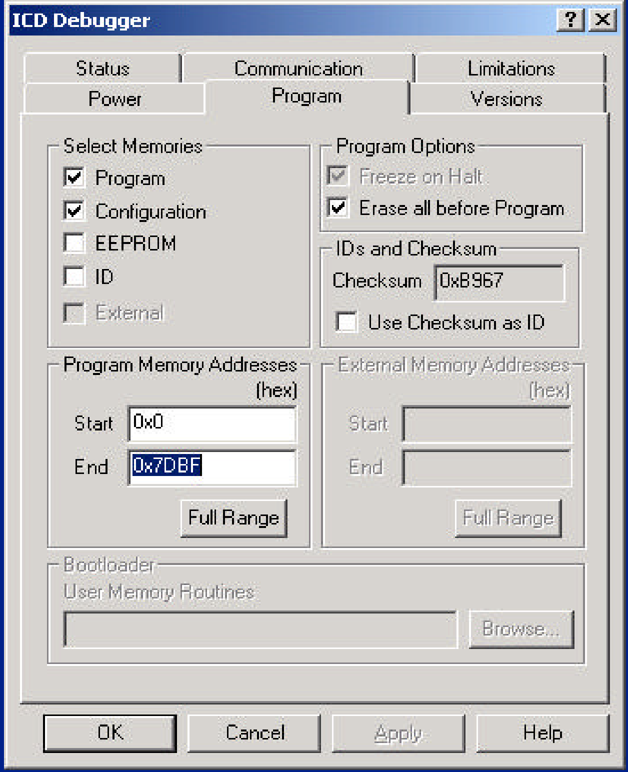
Messages

Message Level: Minimal

☐ Output to debug file

Self Test

Target Vdd	Pass
Module Vpp	Pass
MCLR Gnd	Pass
MCLR Vdd	Pass
MCLR Vpp	Pass



ICD Debugger

Tabs: Status | Communication | Limitations | Power | Program | Versions

Select Memories

☑ Program
☑ Configuration
☐ EEPROM
☐ ID
☐ External

Program Options

☑ Freeze on Halt
☑ Erase all before Program

IDs and Checksum

Checksum: 0xB967

☐ Use Checksum as ID

Program Memory Addresses (hex)

Start: 0x0

End: 0x7DBF

External Memory Addresses (hex)

Start:

End:

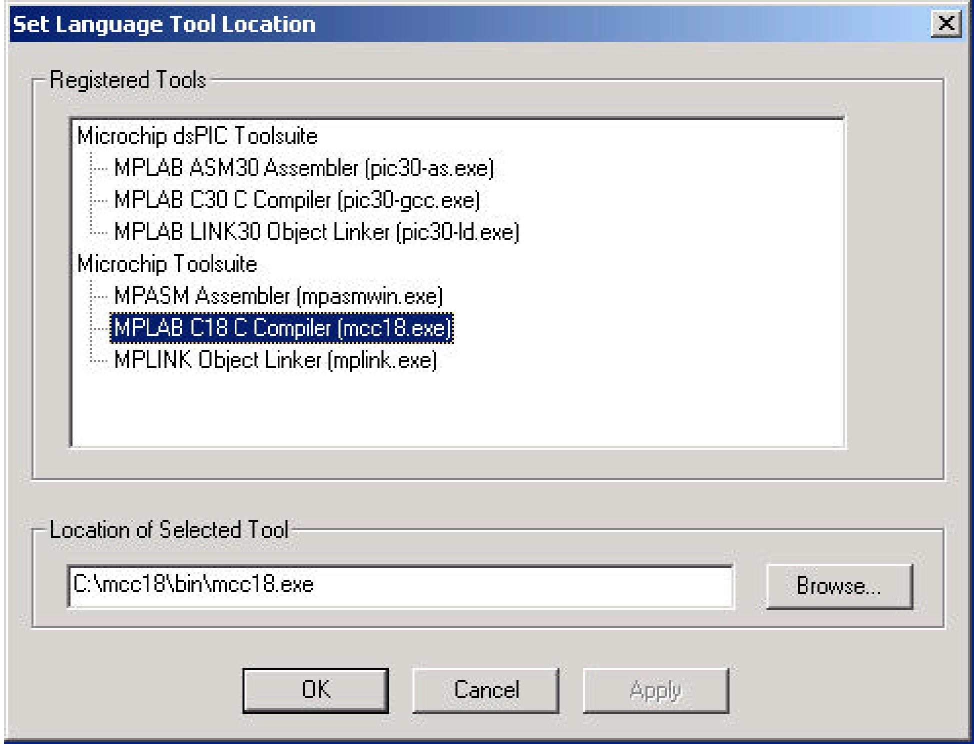
Bootloader

User Memory Routines

Verify Compiler Installation

- Project -> Set Language Tool Locations

MPLAB-C18 Compiler located in C:\mcc18\bin\mcc18.exe



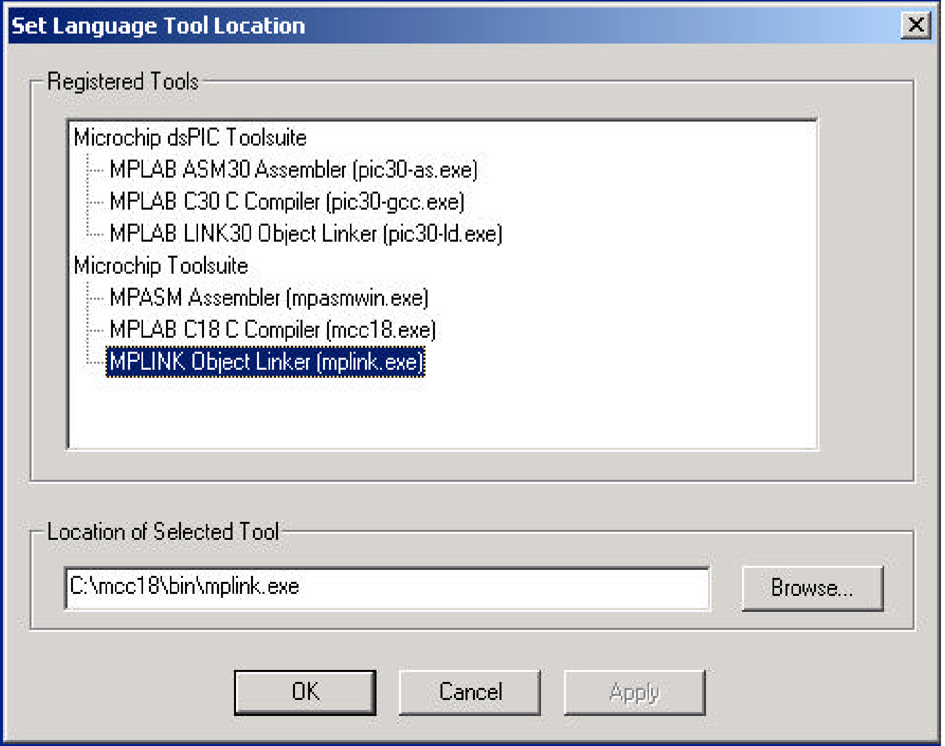
Set Language Tool Location

Registered Tools

- Microchip dsPIC Toolsuite
  - MPLAB ASM30 Assembler (pic30-as.exe)
  - MPLAB C30 C Compiler (pic30-gcc.exe)
  - MPLAB LINK30 Object Linker (pic30-ld.exe)
- Microchip Toolsuite
  - MPASM Assembler (mpasmwin.exe)
  - MPLAB C18 C Compiler (mcc18.exe)
  - MPLINK Object Linker (mplink.exe)

Location of Selected Tool: C:\mcc18\bin\mcc18.exe

MPLINK Linker located in C:\mcc18\bin\mplink.exe



Set Language Tool Location

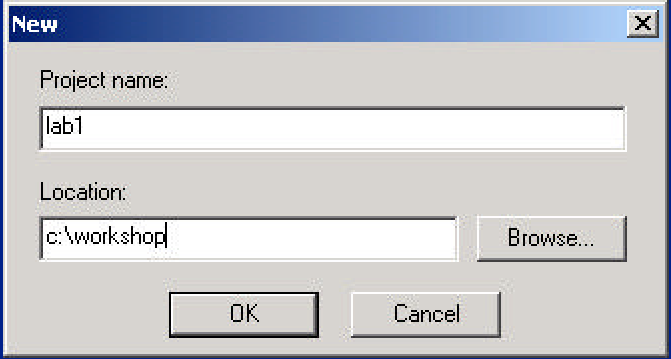
Registered Tools

- Microchip dsPIC Toolsuite
  - MPLAB ASM30 Assembler (pic30-as.exe)
  - MPLAB C30 C Compiler (pic30-gcc.exe)
  - MPLAB LINK30 Object Linker (pic30-ld.exe)
- Microchip Toolsuite
  - MPASM Assembler (mpasmwin.exe)
  - MPLAB C18 C Compiler (mcc18.exe)
  - MPLINK Object Linker (mplink.exe)

Location of Selected Tool: C:\mcc18\bin\mplink.exe

Project Creation

- Project -> New
  - Name project lab1 and place in c:\workshop

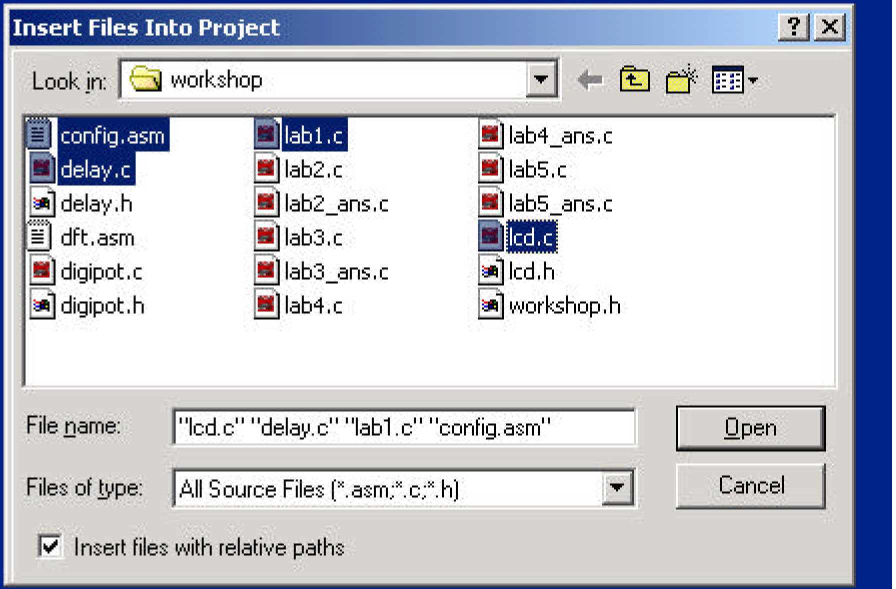


New

Project name: lab1

Location: c:\workshop

- Project -> Insert Files
  - Add **config.asm, delay.c, lab1.c, lcd.c** located in the c:\workshop directory
  - Hold CTL to add files



Insert Files Into Project

Look in: workshop

config.asm
delay.c
delay.h
dft.asm
digipot.c
digipot.h
lab1.c
lab2.c
lab2_ans.c
lab3.c
lab3_ans.c
lab4.c
lab4_ans.c
lab5.c
lab5_ans.c
lcd.c
lcd.h
workshop.h

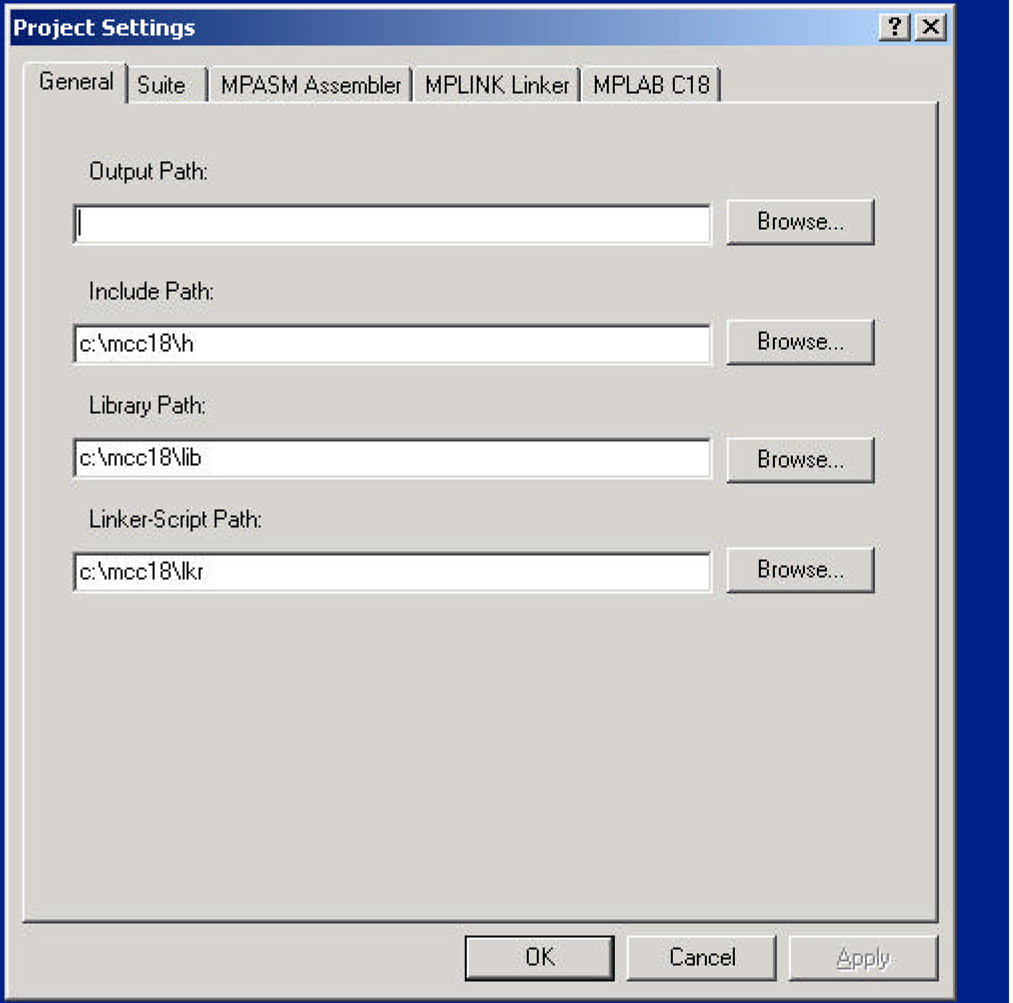
File name: "lcd.c" "delay.c" "lab1.c" "config.asm"

Files of type: All Source Files (*.asm;*.c;*.h)

☑ Insert files with relative paths

Add Compiler Path Information

- Project -> Settings Configure Paths
  - Include -> c:\mcc18\h
  - Library -> c:\mcc18\lib
  - Linker -> c:\mcc18\lkr
  - Output -> Blank…
- Expect this to be automated in future release



Output Path:

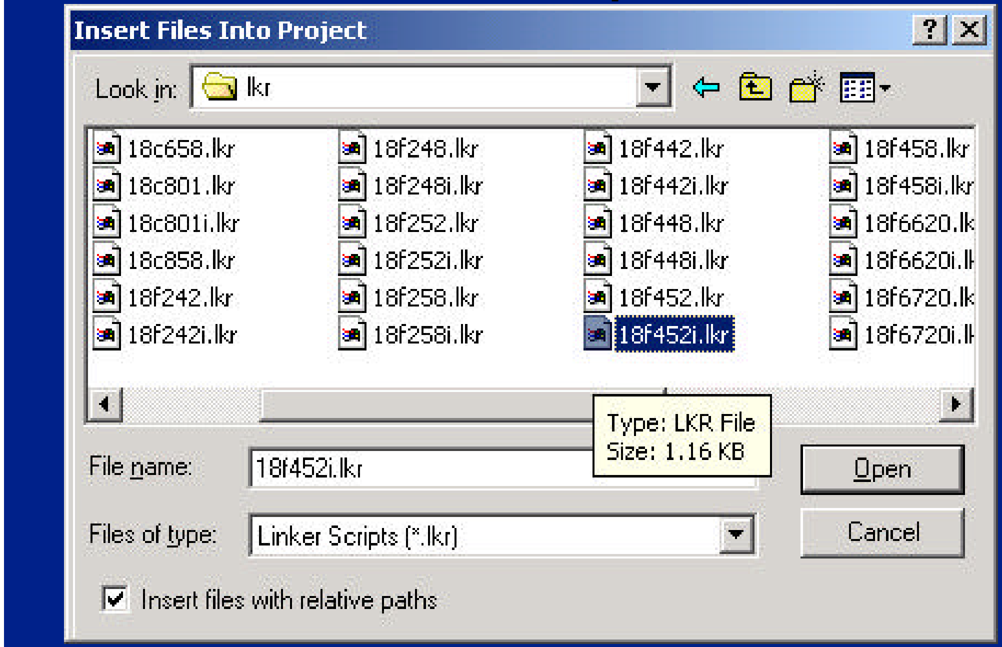
Include Path: c:\mcc18\h

Library Path: c:\mcc18\lib

Linker-Script Path: c:\mcc18\lkr

Add Your Linker Script

- Project -> Insert Files -> Select Linker Script C:\mcc18\lkr\p18F452i.lkr



Insert Files Into Project

Look in: lkr

18c658.lkr
18c801.lkr
18c801i.lkr
18c858.lkr
18f242.lkr
18f242i.lkr
18f248.lkr
18f248i.lkr
18f252.lkr
18f252i.lkr
18f258.lkr
18f258i.lkr
18f442.lkr
18f442i.lkr
18f448.lkr
18f448i.lkr
18f452.lkr
18f452i.lkr
18f458.lkr
18f458i.lkr
18f6620.lk
18f6620i.l
18f6720.lk
18f6720i.l

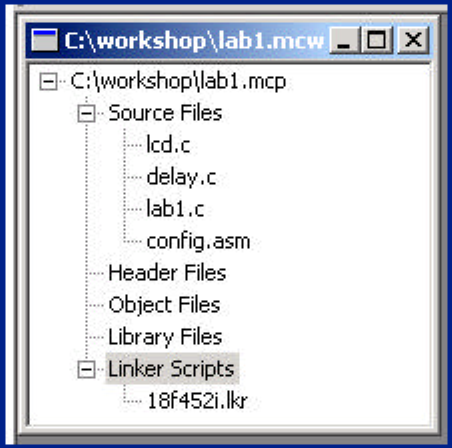
Type: LKR File
Size: 1.16 KB

File name: 18f452i.lkr

Files of type: Linker Scripts (*.lkr)

☑ Insert files with relative paths

- Your Project Should Look Like This...

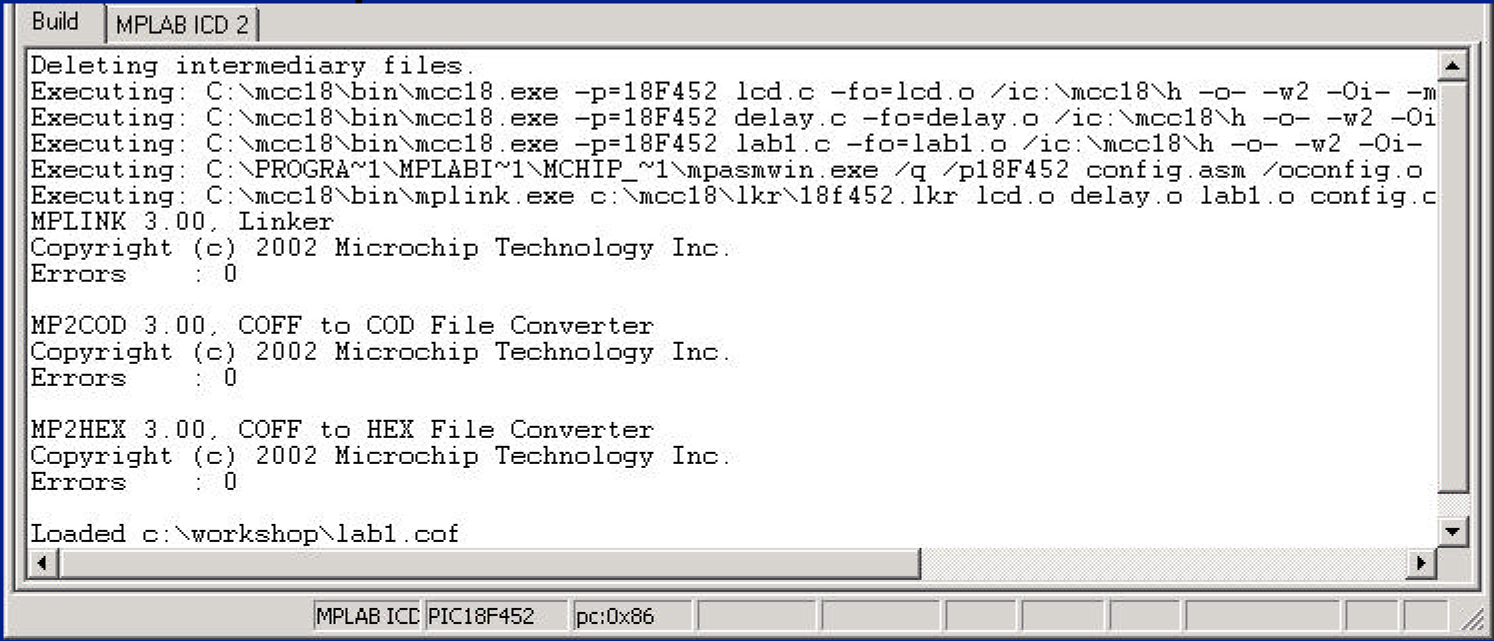


C:\workshop\lab1.mcw

```
C:\workshop\lab1.mcp
  Source Files
    lcd.c
    delay.c
    lab1.c
    config.asm
  Header Files
  Object Files
  Library Files
  Linker Scripts
    18f452i.lkr
```

Build Your Project

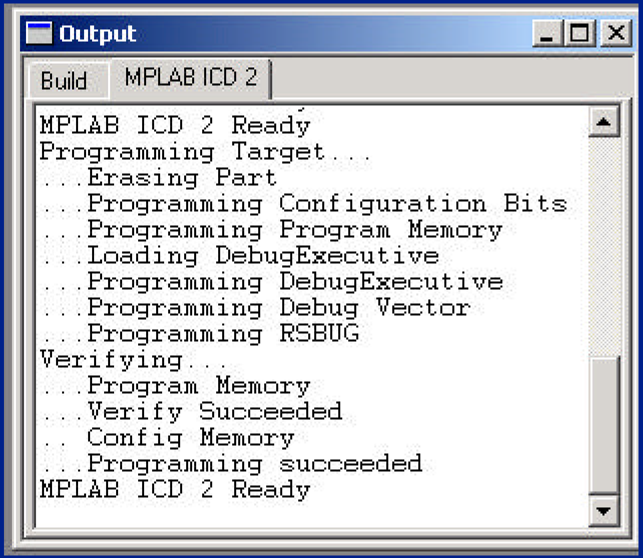
- Project->Build All
  - MPLAB-C18 runs on all C files in output window
  - MPASM assembler runs on config.asm file
  - MPLINK linker links all files together and creates HEX output



```
Deleting intermediary files.
Executing: C:\mcc18\bin\mcc18.exe -p=18F452 lcd.c -fo=lcd.o /ic:\mcc18\h -o- -w2 -Oi- -m
Executing: C:\mcc18\bin\mcc18.exe -p=18F452 delay.c -fo=delay.o /ic:\mcc18\h -o- -w2 -Oi
Executing: C:\mcc18\bin\mcc18.exe -p=18F452 lab1.c -fo=lab1.o /ic:\mcc18\h -o- -w2 -Oi-
Executing: C:\PROGRA~1\MPLABI~1\MCHIP_~1\mpasmwin.exe /q /p18F452 config.asm /oconfig.o
Executing: C:\mcc18\bin\mplink.exe c:\mcc18\lkr\18f452.lkr lcd.o delay.o lab1.o config.c
MPLINK 3.00, Linker
Copyright (c) 2002 Microchip Technology Inc.
Errors    : 0

MP2COD 3.00, COFF to COD File Converter
Copyright (c) 2002 Microchip Technology Inc.
Errors    : 0

MP2HEX 3.00, COFF to HEX File Converter
Copyright (c) 2002 Microchip Technology Inc.
Errors    : 0

Loaded c:\workshop\lab1.cof
```

Download Code to Target and Run…..

- Debugger -> Download to Target

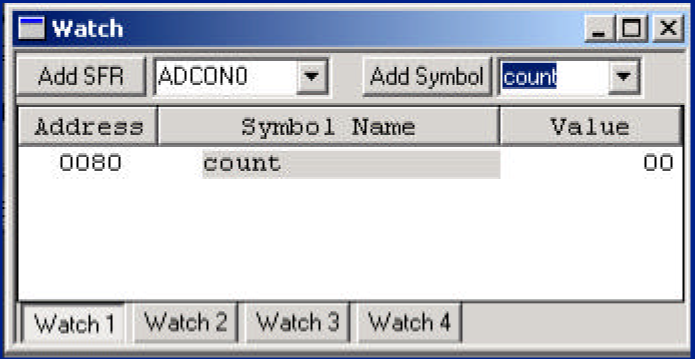


```
MPLAB ICD 2 Ready
Programming Target...
...Erasing Part
...Programming Configuration Bits
...Programming Program Memory
...Loading DebugExecutive
...Programming DebugExecutive
...Programming Debug Vector
...Programming RSBUG
Verifying...
...Program Memory
...Verify Succeeded
.. Config Memory
...Programming succeeded
MPLAB ICD 2 Ready
```

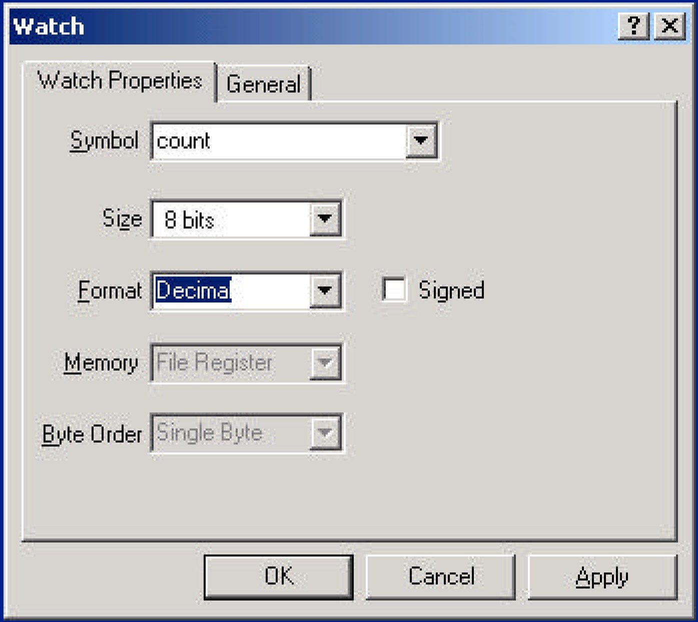
- Debugger -> Run
  - You should see a “Welcome” message on the LCD display

Creating Watch Windows

- Debugger -> Halt
- View->Watch, add count
- Right Click count and change Watch Properties Format to decimal and hit OK



Address	Symbol Name	Value
0080	count	00



Symbol: count

Size: 8 bits

Format: Decima

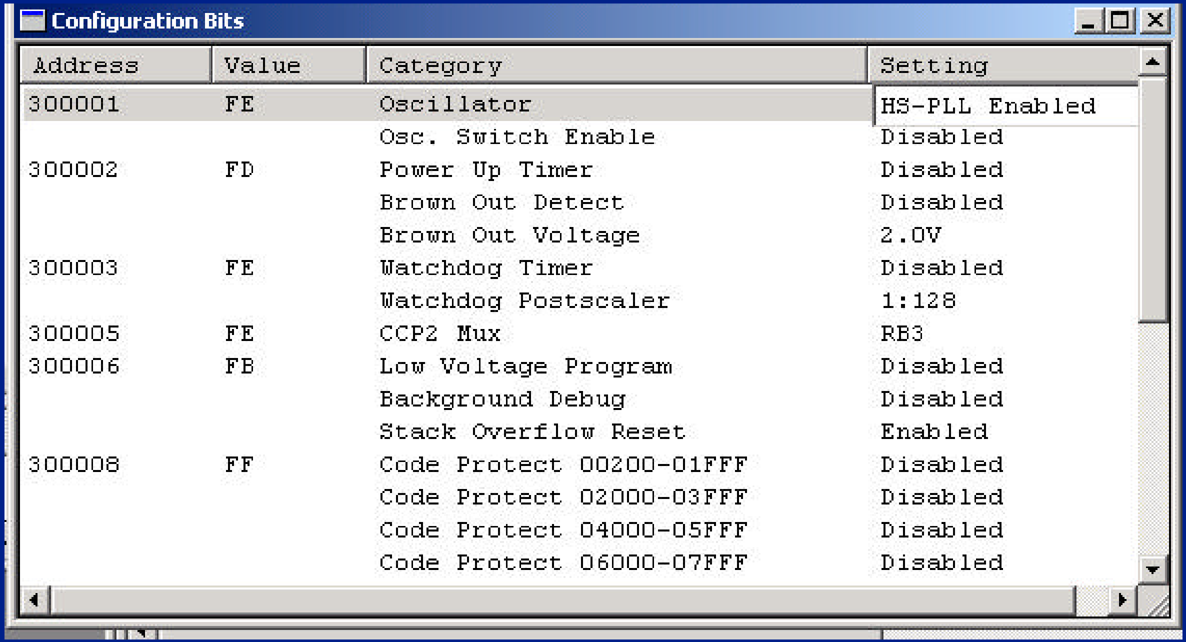
□ Signed

Memory: File Register

Byte Order: Single Byte

Configuration Bit Settings Window

- Configure -> Configuration Bits
  - Automatically Initialized by config.asm if included in your project

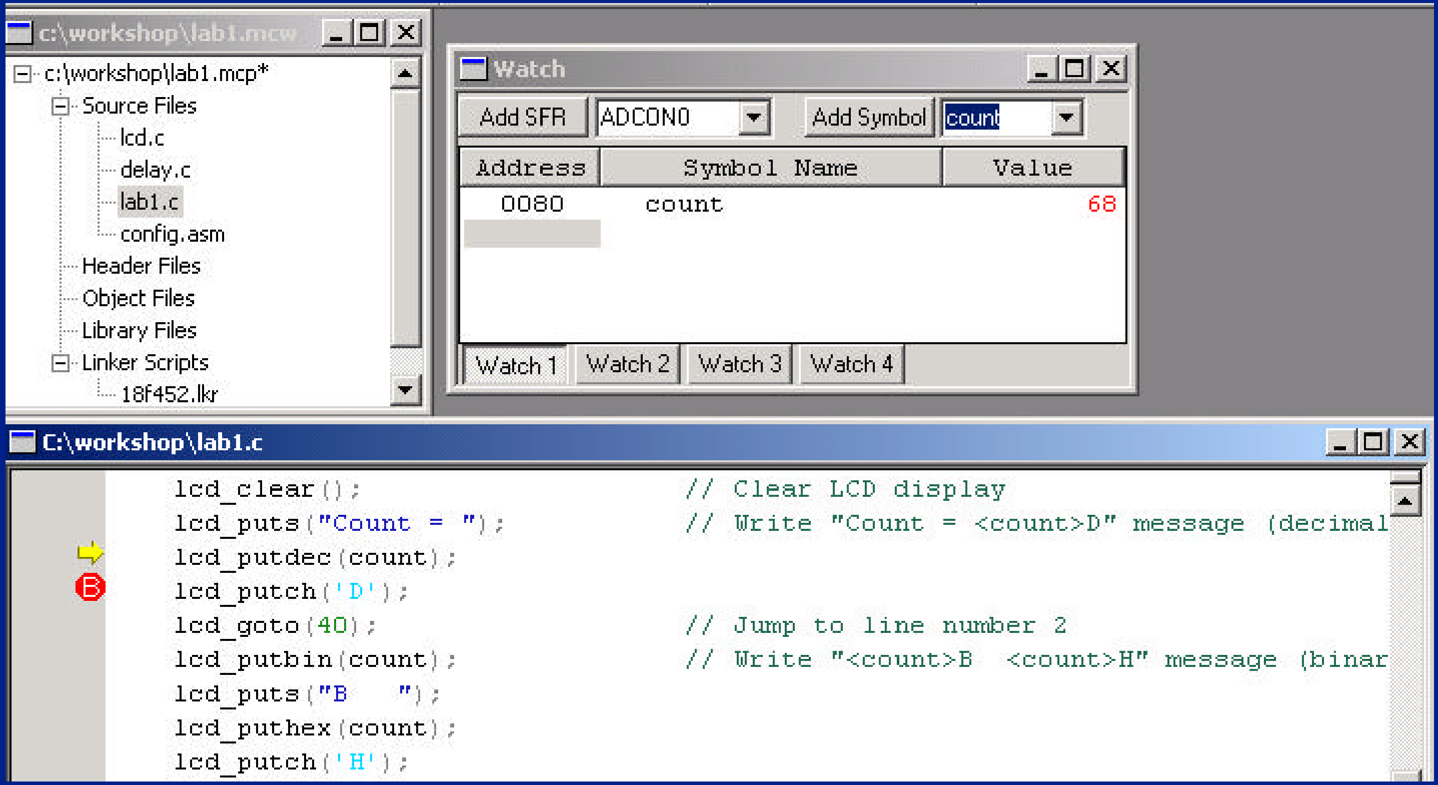


Configuration Bits

Address	Value	Category	Setting
300001	FE	Oscillator	HS-PLL Enabled
		Osc. Switch Enable	Disabled
300002	FD	Power Up Timer	Disabled
		Brown Out Detect	Disabled
		Brown Out Voltage	2.0V
300003	FE	Watchdog Timer	Disabled
		Watchdog Postscaler	1:128
300005	FE	CCP2 Mux	RB3
300006	FB	Low Voltage Program	Disabled
		Background Debug	Disabled
		Stack Overflow Reset	Enabled
300008	FF	Code Protect 00200-01FFF	Disabled
		Code Protect 02000-03FFF	Disabled
		Code Protect 04000-05FFF	Disabled
		Code Protect 06000-07FFF	Disabled

Setting Breakpoints

- Double Click on lab1.c in Project Window
- Find lcd_putch(‘D’); select and click this statement with right mouse button to set a breakpoint there



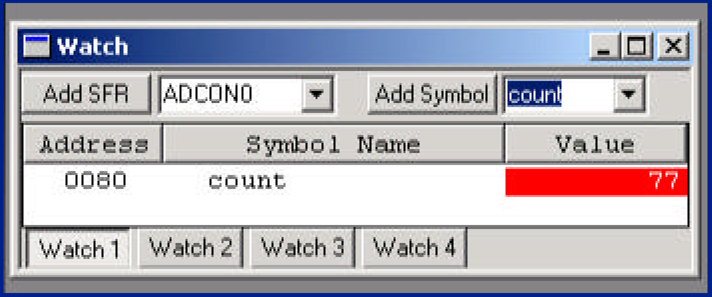
Address	Symbol Name	Value
0080	count	68

```
lcd_clear();                  // Clear LCD display
lcd_puts("Count = ");         // Write "Count = <count>D" message (decimal
lcd_putdec(count);
lcd_putch('D');
lcd_goto(40);                 // Jump to line number 2
lcd_putbin(count);            // Write "<count>B  <count>H" message (binar
lcd_puts("B    ");
lcd_puthex(count);
lcd_putch('H');
```

Modifying Watch Values

- Hit Debugger -> Run see debugger halt at lcd_putdec
- “Count = <value>”on the LCD should be the same as count in your watch window
- Place your cursor over the watch value, type a new number and re-run
- See new value

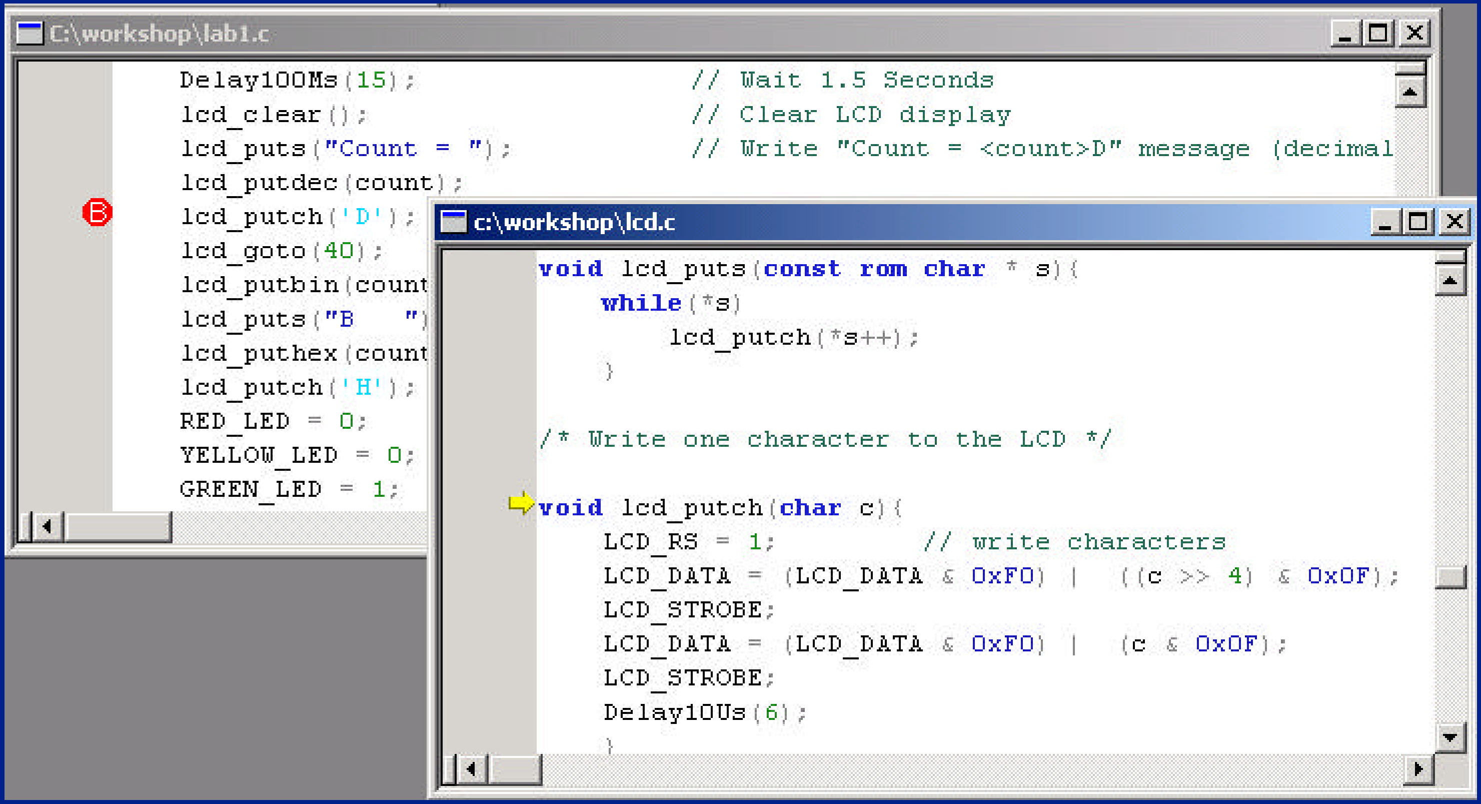
  on LCD display



Address	Symbol Name	Value
0080	count	77

Single Stepping

- Debugger->Step Into
  - MPLAB automatically opens lcd.c and steps into lcd_putch()….

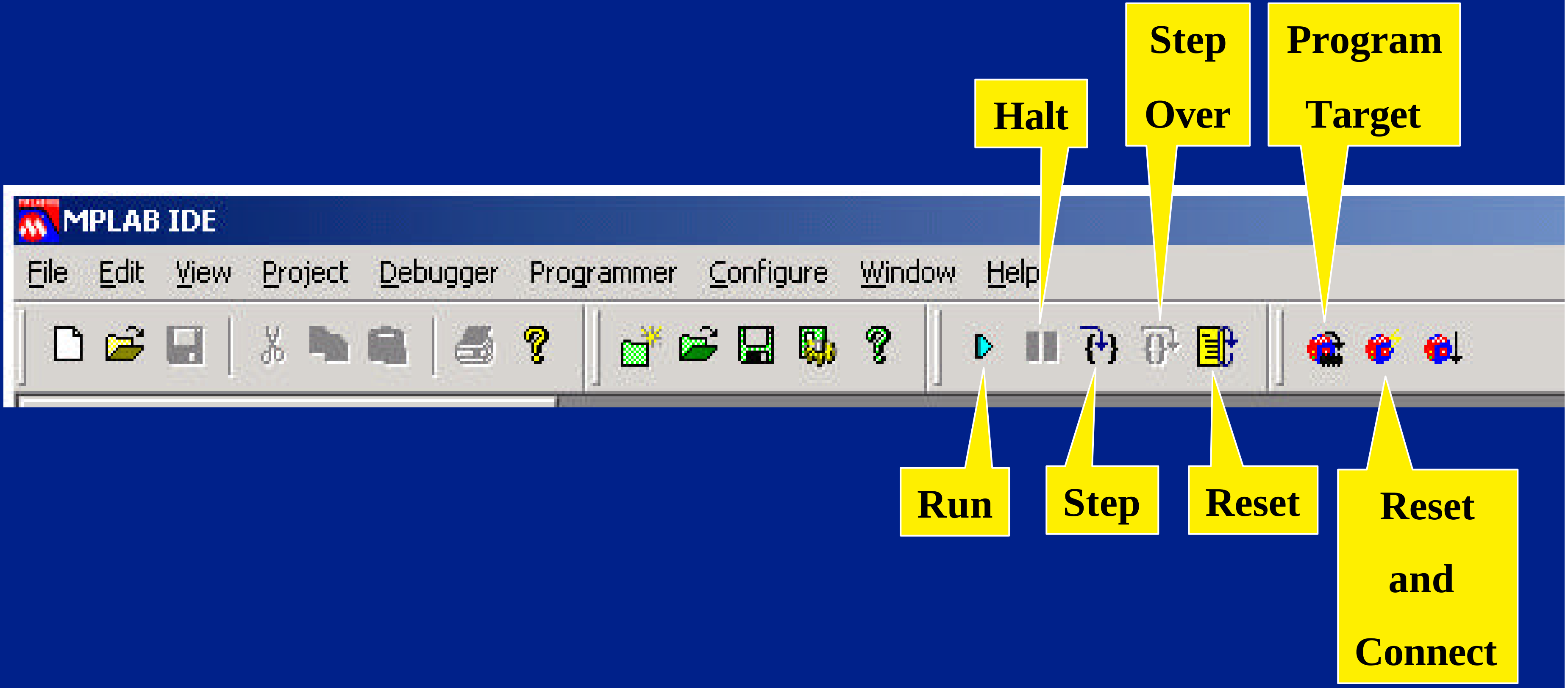


```
Delay100Ms(15);                   // Wait 1.5 Seconds
lcd_clear();                      // Clear LCD display
lcd_puts("Count = ");             // Write "Count = <count>D" message (decimal
lcd_putdec(count);
lcd_putch('D');
lcd_goto(40);
lcd_putbin(count
lcd_puts("B    ")
lcd_puthex(count
lcd_putch('H');
RED_LED = 0;
YELLOW_LED = 0;
GREEN_LED = 1;
```

```
void lcd_puts(const rom char * s){
    while(*s)
        lcd_putch(*s++);
    }

/* Write one character to the LCD */

void lcd_putch(char c){
    LCD_RS = 1;          // write characters
    LCD_DATA = (LCD_DATA & 0xF0) |  ((c >> 4)  & 0x0F);
    LCD_STROBE;
    LCD_DATA = (LCD_DATA & 0xF0) |  (c & 0x0F);
    LCD_STROBE;
    Delay10Us(6);
    }
```

Tool Bars

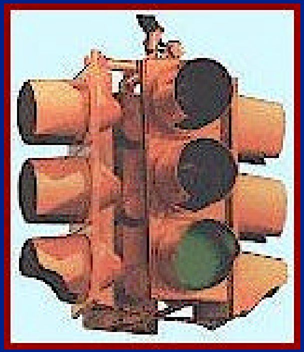
- Most common debugger functions are available on the toolbar

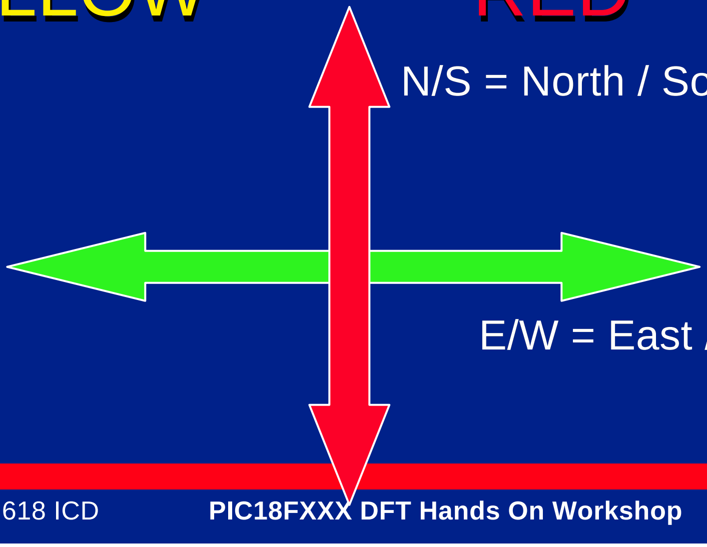


Lab 2 Traffic Light

- Traffic Light has (4) states:

State	North / South	East / West
EW_GREEN	RED	GREEN
EW_YELLOW	RED	YELLOW
NS_GREEN	GREEN	RED
NS_YELLOW	YELLOW	RED

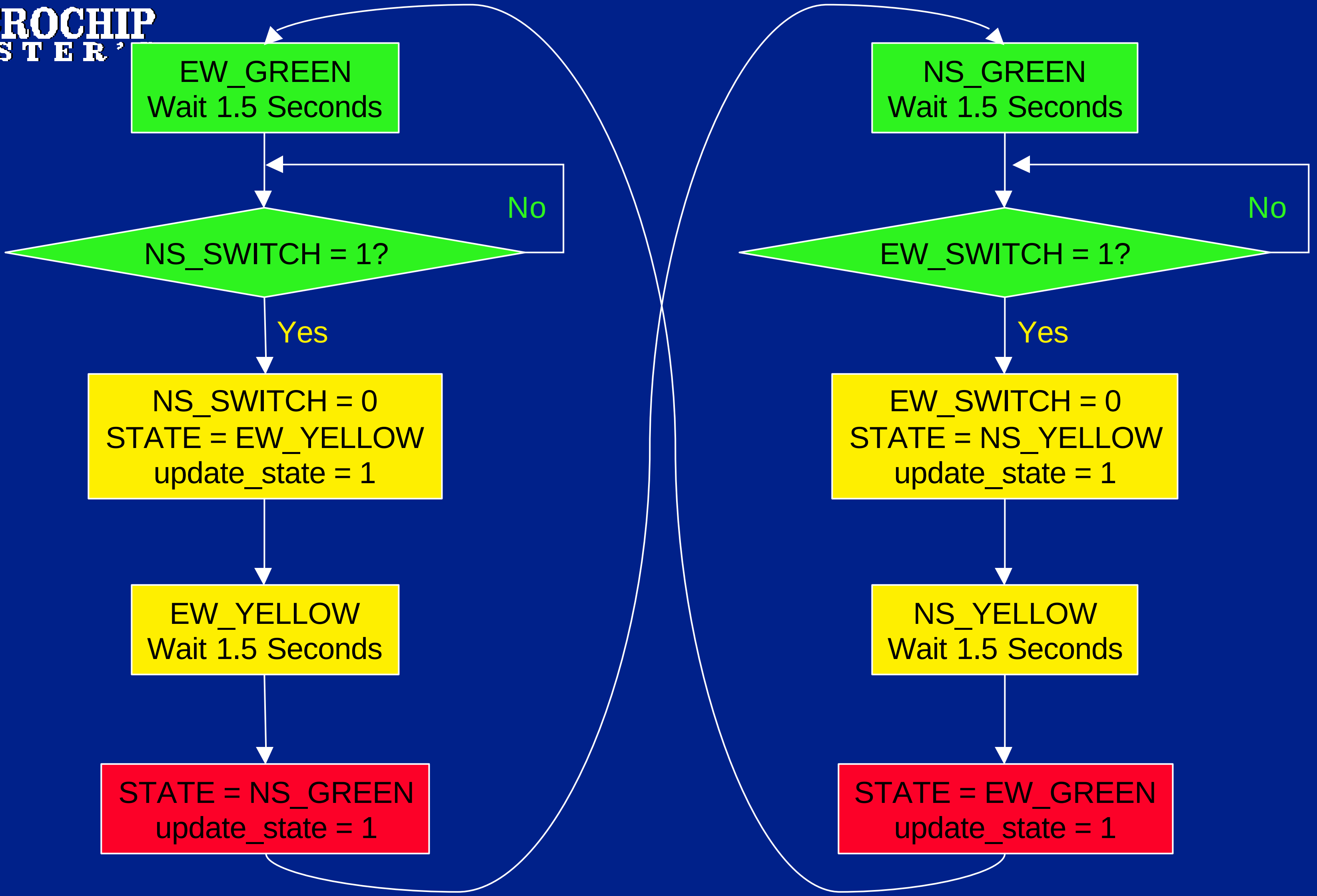




N/S = North / South

E/W = East / West

Lab 2: Traffic Light Logic



Lab 2:Traffic Light Implementation

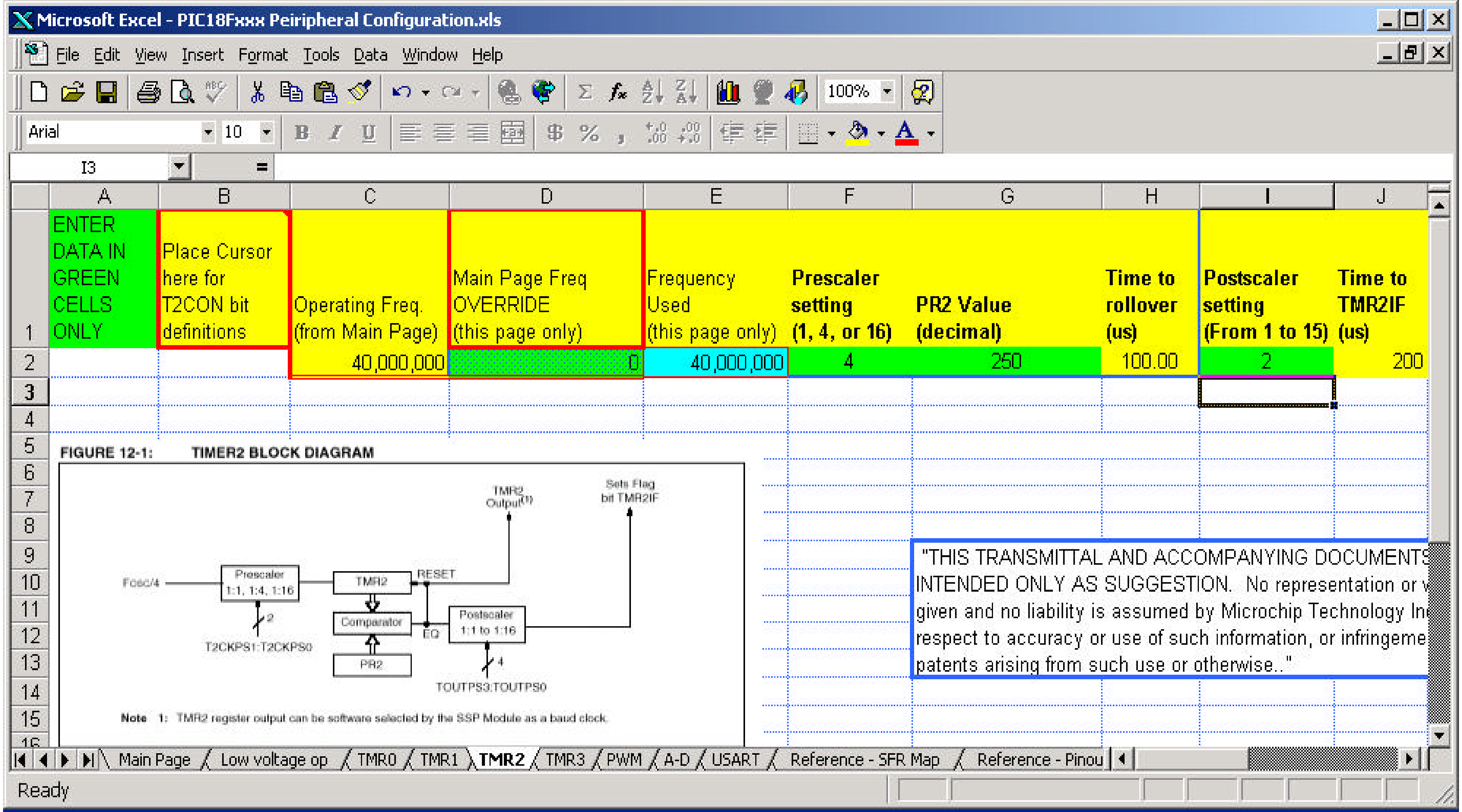
- Project -> Open -> lab2.mcp
- Edit lab2.c and add the state transition code as follows:

```
switch(state){
    case EW_GREEN: Delay100Ms(15);    // Wait 1.5 Seconds
               // Place your code for EW_GREEN here
               break;
    case EW_YELLOW: Delay100Ms(15);   // Wait 1.5 Seconds
               // Place your code for EW_YELLOW here
               break;
    case NS_GREEN: Delay100Ms(15);    // Wait 1.5 Seconds
               // Place your code for NS_GREEN here
               break;
    case NS_YELLOW: Delay100Ms(15);   // Wait 1.5 Seconds
               // Place your code for NS_YELLOW here
               break;
               }
```

Lab 3: Using Timer 2 for Sampling Interval

- Timer 2 and PR2 are used to create an automatic high priority periodic interrupt
- PICmicro running at 40 Mhz / 100 nS instruction cycle
- Desire 5 Khz sampling rate = 200 uS
- 200 uS / (100 nS instruction cycle = 2000 instruction cycles
- Assign Timer 2 to the high priority vector and enable high priority interrupts

Using PIC18FXXX Peripheral Calculations Spreadsheet



Microsoft Excel - PIC18Fxxx Peiripheral Configuration.xls
File Edit View Insert Format Tools Data Window Help
100%
Arial
10
I3
=
Main Page
Low voltage op
TMR0
TMR1
TMR2
TMR3
PWM
A-D
USART
Reference - SFR Map
Reference - Pinou
Ready

A	B	C	D	E	F	G	H	I	J
ENTER DATA IN GREEN CELLS ONLY	Place Cursor here for T2CON bit definitions	Operating Freq. (from Main Page)	Main Page Freq OVERRIDE (this page only)	Frequency Used (this page only)	**Prescaler setting (1, 4, or 16)**	**PR2 Value (decimal)**	**Time to rollover (us)**	**Postscaler setting (From 1 to 15)**	**Time to TMR2IF (us)**
		40,000,000	0	40,000,000	4	250	100.00	2	200

FIGURE 12-1: TIMER2 BLOCK DIAGRAM

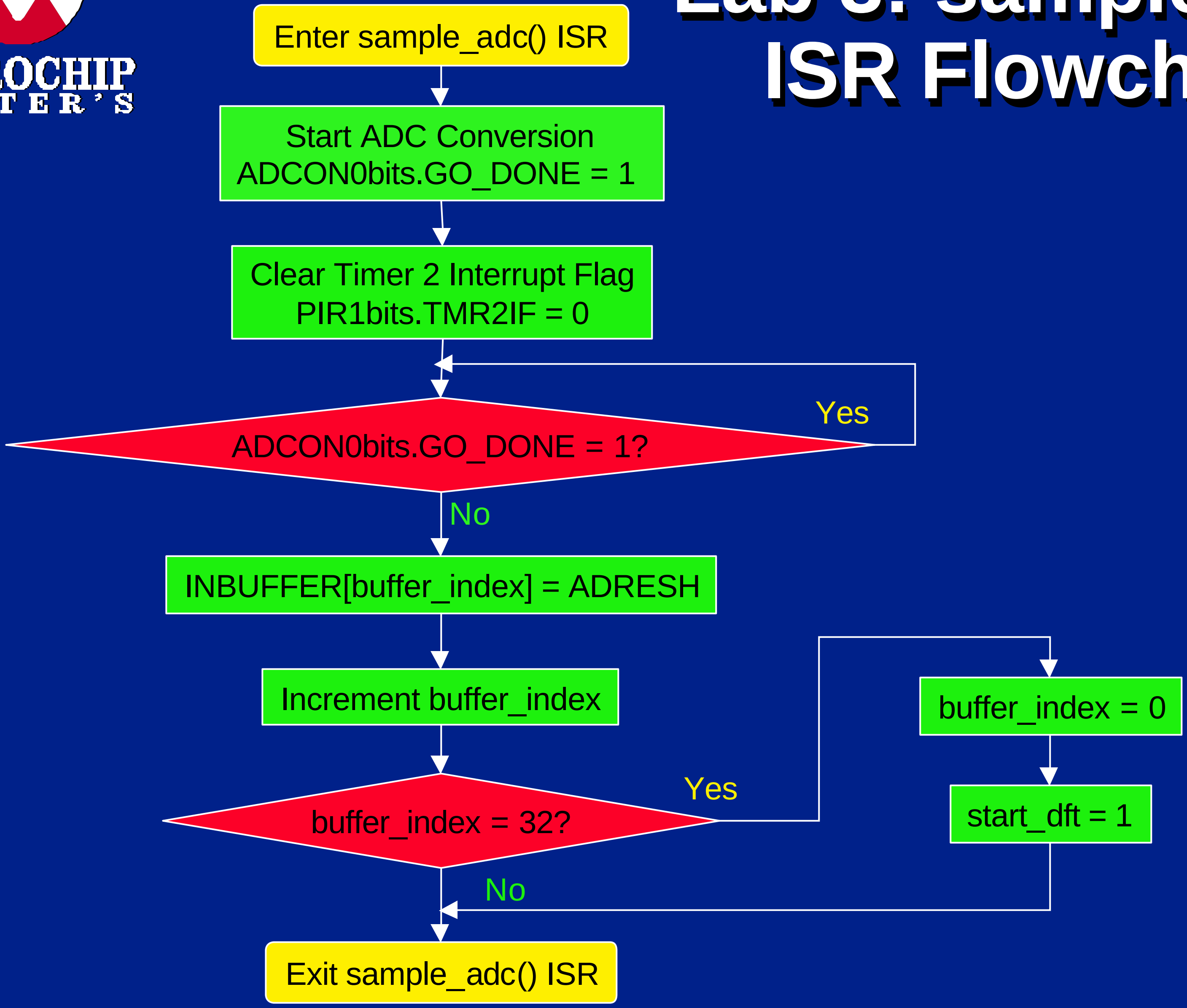
Note 1: TMR2 register output can be software selected by the SSP Module as a baud clock.

"THIS TRANSMITTAL AND ACCOMPANYING DOCUMENTS INTENDED ONLY AS SUGGESTION. No representation or w given and no liability is assumed by Microchip Technology In respect to accuracy or use of such information, or infringeme patents arising from such use or otherwise.."

Timer 2 and A/D Initialization and Interrupt Assignment

```
ADCON0 = 0b10000001;// A/D on, Channel 0, CLK/64 clock
ADCON1 = 0b01000010;// Left Justification, CLK/64
                    // clock, AN0~AN4 analog

T2CON = 0b00001101; // TMR2 On, 4:1 pre, 2:1 post scaller
PR2 = 249;          // Select 250 cycle period
PIE1bits.TMR2IE = 1;// Enable Timer 2 interrupts
IPR1 = 0b00000010;  // High priority for Timer 2
IPR2 = 0;           // Low priority for other peripherals
RCONbits.IPEN = 1;  // Enable high / low priority feature
INTCON = 0b11000000;// Enable Low and High priority interrupts
```

Lab 3: sample_adc ISR Flowchart



Lab 3: Write code for Interrupt Driven Sampling Routine

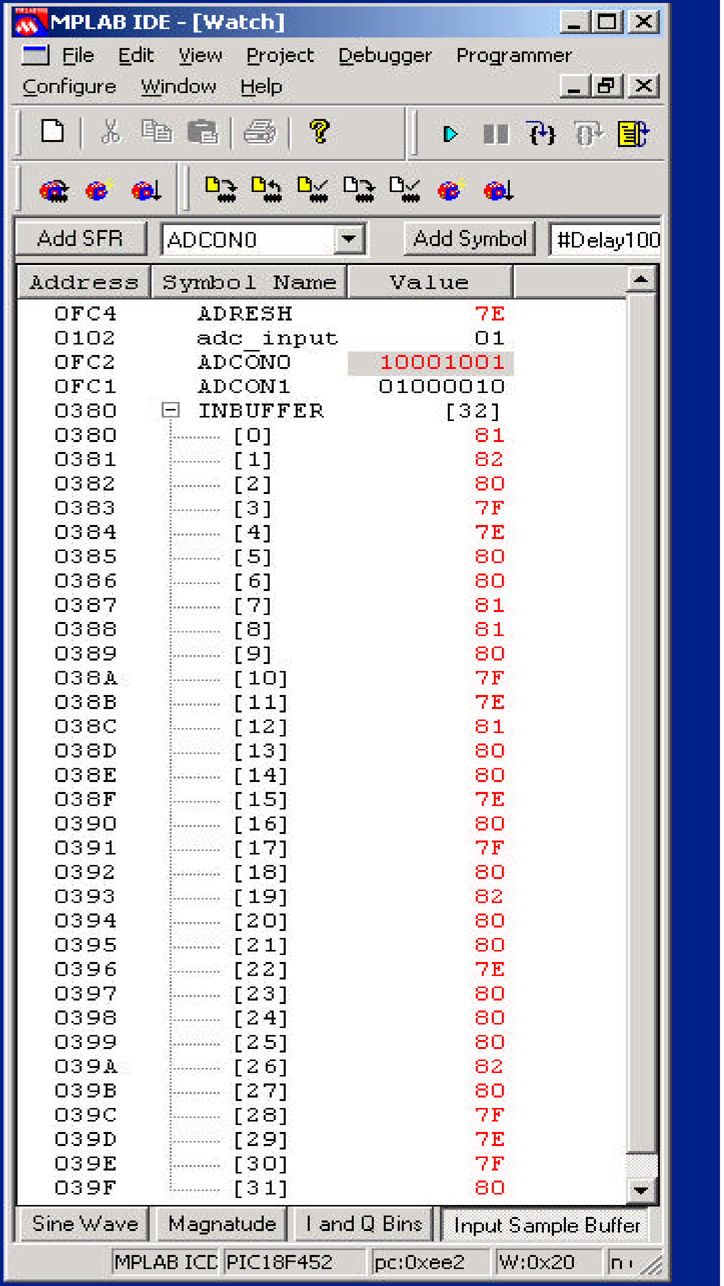
```
#pragma interrupt sample_adc // High priority interrupt
void sample_adc (void){ // TMR2 overflow every 2,000
                        // cycles, 200 uS / 5 Khz @ 40 Mhz
if (PIR1bits.TMR2IF){
   // Start A/D conversion
   // Clear TMR2 interrupt flag
   // Spin lock + wait for A/D conversion to complete
   // Store A/D result into next location in INBUFFER
   // Increment buffer_index and use next buffer location
   // Once you hit the end of the INBUFFER[32]:
      // - Reset the pointer to zero
      // - Set start_dft flag bit to run the DFT
   }
}
```

Lab 3: Instructions

- Fill in source code for sample_adc() using:
  - INBUFFER[32] stores results
  - buffer_index accesses each element
  - ADCON0bits.GO_DONE starts ADC conversion
  - ADCON0bits.GO_DONE is 1 when ADC is busy
  - ADRESH returns 8-bit ADC value
  - start_dft = 1 when buffer is full, clear buffer_index
- Build/Compile code, program target
- Set breakpoint where you set start_dft
- Run and examine INBUFFER[] for results ->>>

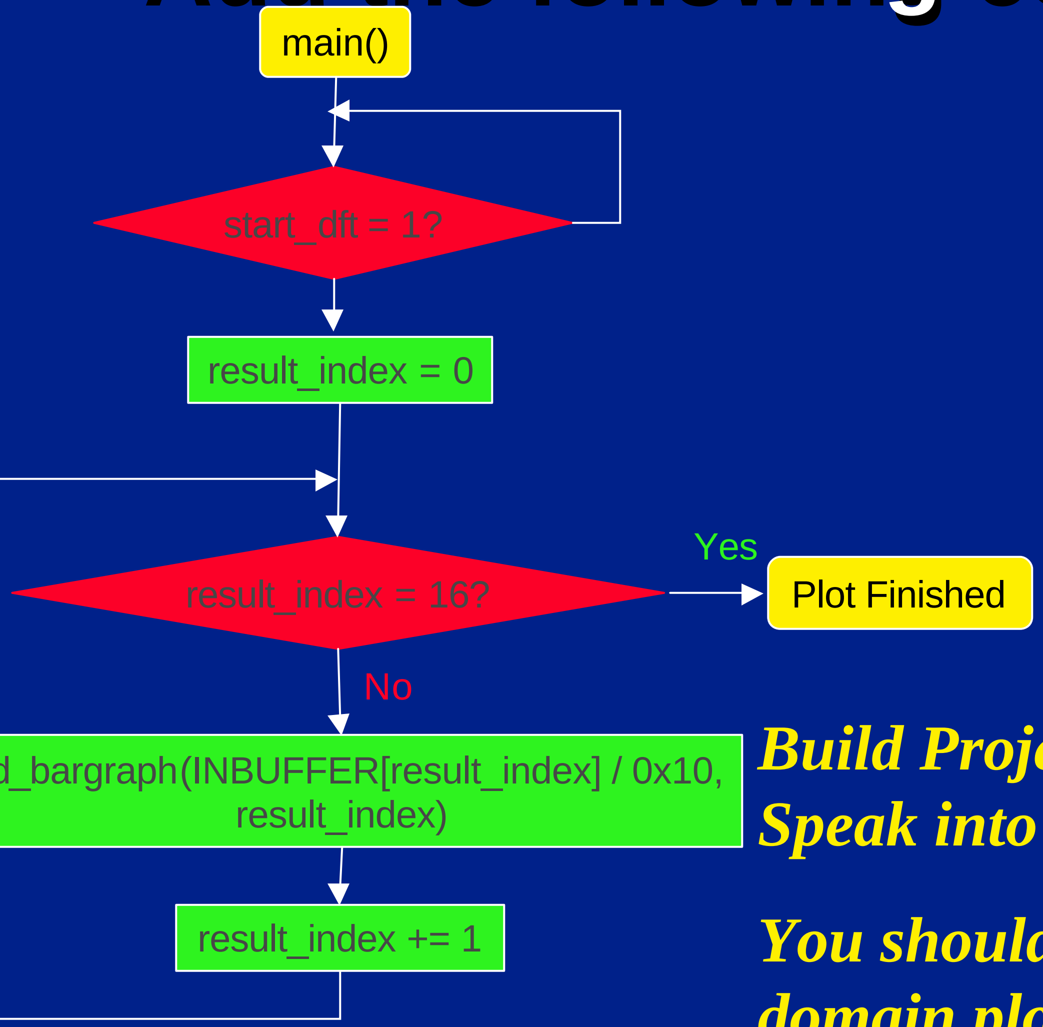
INBUFFER[32] Results

- View Input Sample Buffer in Watch Window:
  - INBUFFER[32] Shows A/D sampling Results
  - Values should be centered around 0x80
  - All 32 locations should be captured and stored



Address	Symbol Name	Value
0FC4	ADRESH	7E
0102	adc_input	01
0FC2	ADCON0	10001001
0FC1	ADCON1	01000010
0380	INBUFFER	[32]
0380	[0]	81
0381	[1]	82
0382	[2]	80
0383	[3]	7F
0384	[4]	7E
0385	[5]	80
0386	[6]	80
0387	[7]	81
0388	[8]	81
0389	[9]	80
038A	[10]	7F
038B	[11]	7E
038C	[12]	81
038D	[13]	80
038E	[14]	80
038F	[15]	7E
0390	[16]	80
0391	[17]	7F
0392	[18]	80
0393	[19]	82
0394	[20]	80
0395	[21]	80
0396	[22]	7E
0397	[23]	80
0398	[24]	80
0399	[25]	80
039A	[26]	82
039B	[27]	80
039C	[28]	7F
039D	[29]	7E
039E	[30]	7F
039F	[31]	80

Graphing INBUFFER[ ] Results
Add the following code…..



main()

start_dft = 1?

result_index = 0

result_index = 16?

Yes

Plot Finished

No

lcd_bargraph(INBUFFER[result_index] / 0x10, result_index)

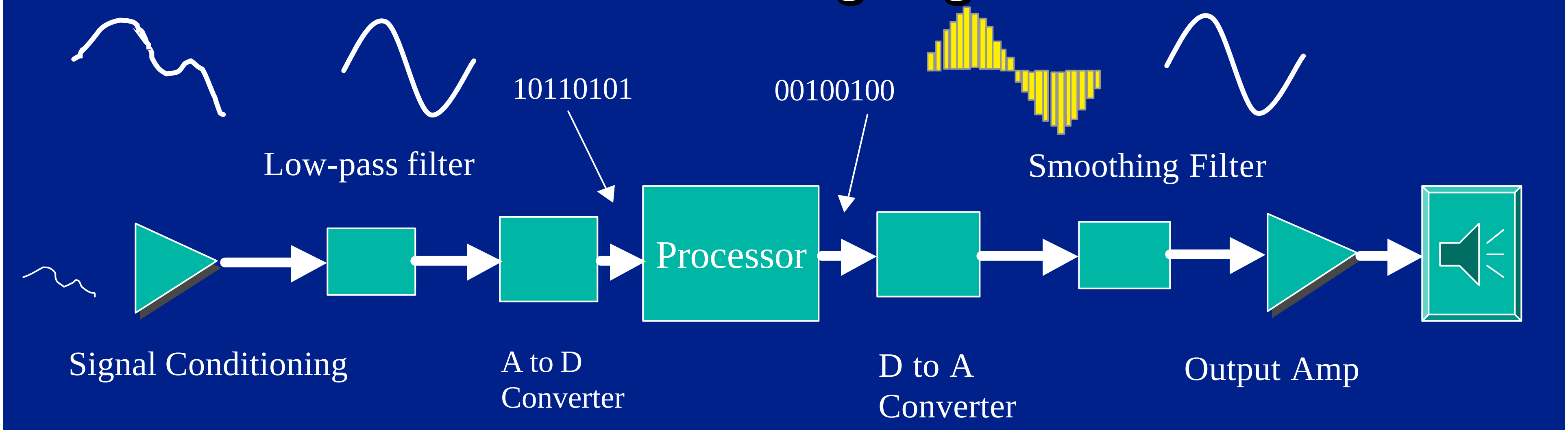
result_index += 1

Build Project, Run and Speak into the microphone.

You should see a time domain plot of your voice!

General DSP Model

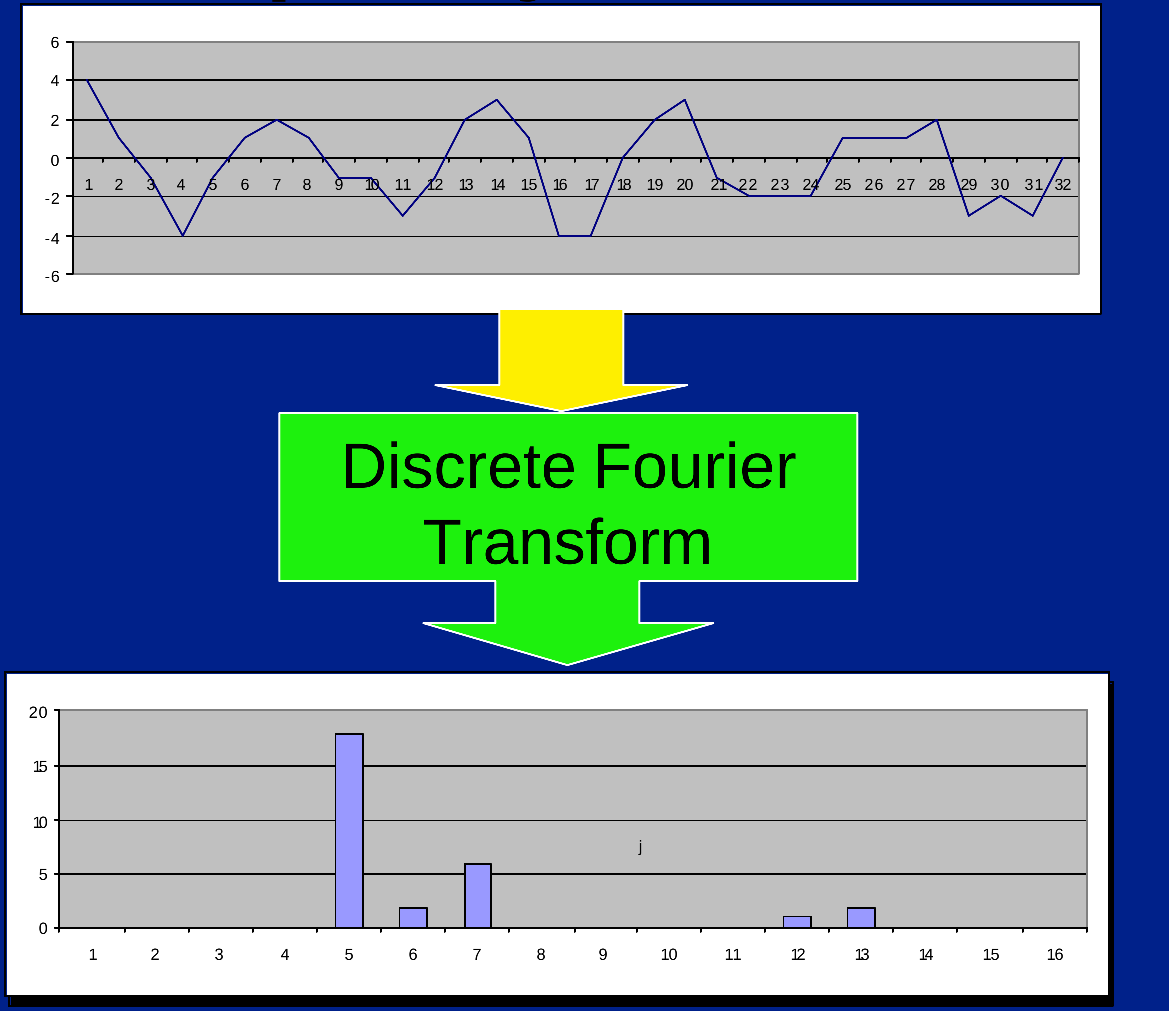
- Accepts an analog signal
- Converts this analog signal to digital domain
- Performs computations
- Displays results, makes decisions or converts results back into analog signal



Converting Time Domain to Frequency Domain

- Convert Time Domain Sampled Data to…

- Frequency Domain Data



Discrete Fourier Transform

Discrete Fourier Transform

Discrete Fourier Transform Formula:

$$X[F] \;=\; \sum_{n=1}^{n=N} \sqrt{X[n] \otimes \sin(2\pi nF/N)^{2\,+}\; X[n] \otimes \cos(2\pi nF/N)^{2}}$$

N = Number of Samples
F = Frequency Bin Number
F_{hz} = F * (Sampling Frequency / Number of Samples)

Example:
Sampling Frequency = 5 Khz
(32) Samples

f_{hz} = f * (5,000 / 32) = f * 156.25 Hz

F_1 = 156.25 Hz	F_5 = 781.25Hz	F_9 = 1406.25 Hz	F_{13} = 2031.25 Hz
F_2 = 312.5 Hz	F_6 = 937.5 Hz	F_{10} = 1562.5 Hz	F_{14} = 2187.5 Hz
F_3 = 468.75 Hz	F_7 = 1093.75 Hz	F_{11} = 1718.75 Hz	F_{15} = 2343.75 Hz
F_4 = 625 Hz	F_8 = 1250 Hz	F_{12} = 1875 Hz	F_{16} = 2500 Hz

DFT Frequency Bin Calculations

Imaginary Magnitude Calculation using Sine table

$$\mathrm{IBIN}\ [\mathrm{BINCOUNT}] = \sum_{\mathrm{BINCOUNT}=1 \quad N=1}^{\mathrm{BINCOUNT}=16 \quad N=32} \mathrm{INBUFFER}[N] \otimes \mathrm{FTABLE}\ [\mathrm{mod}_{32}(N \otimes \mathrm{BINCOUNT})]$$

(16) frequency bins * (32) samples = (512) 24-bit MAC operations

Real Magnitude Calculation using Cosine table

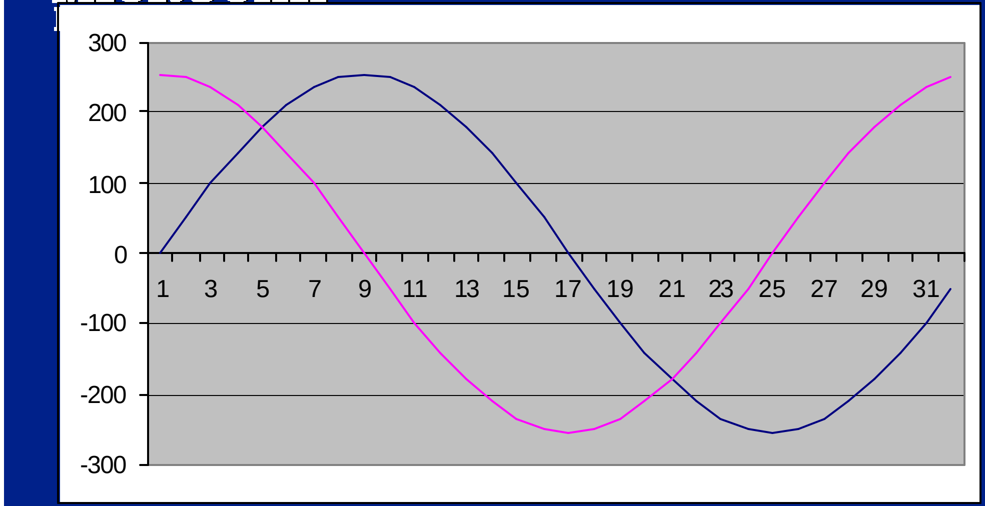
$$\mathrm{QBIN}\ [\mathrm{BINCOUNT}] = \sum_{\mathrm{BINCOUNT}=1 \quad N=1}^{\mathrm{BINCOUNT}=16 \quad N=32} \mathrm{INBUFFER}[N] \otimes \mathrm{FTABLE}\ [\mathrm{mod}_{32}(8 + N \otimes \mathrm{BINCOUNT})]$$

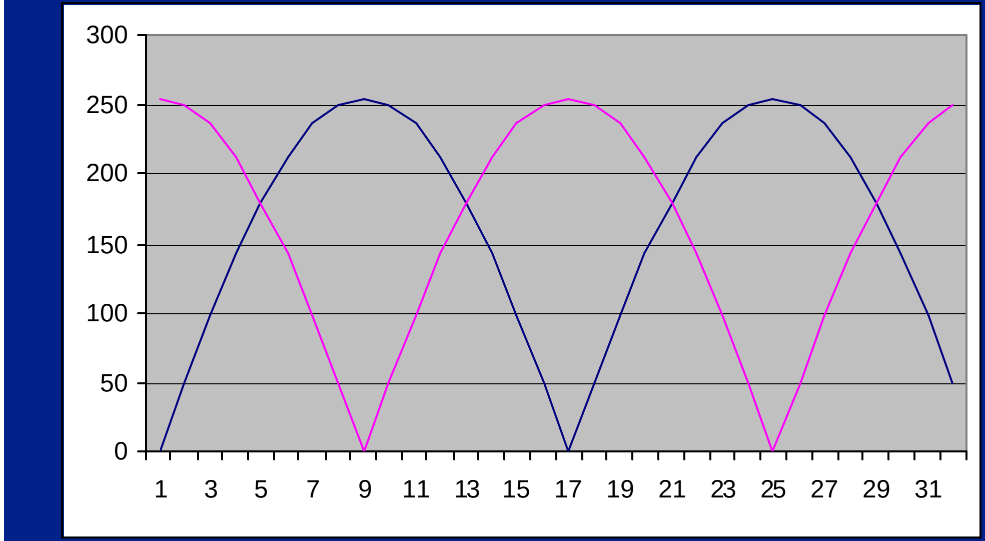
(16) frequency bins * (32) samples = (512) 24-bit MAC operations

DFT Data Structures

- **INBUFFER[32]** : 8-bit A/D samples buffer
- **FTABLE[32]** : 8-bit Signed sine wave
  - Cosine derived by phase shifting FTABLE[32] 90 degrees or (8) samples.
- **IBIN[16]** : 24-bit Imaginary result
- **QBIN[16]** : 24-bit real result
- **magnitude[16]** : 32-bit scaled $I^2 + Q^2$

Frequency Table FTABLE[32]





F_{sample} = 5 Khz / 200 uS
(32) Samples -> F[1] = 5 Khz / 32
F[1] 156.25 Hz, BINCOUNT = 1

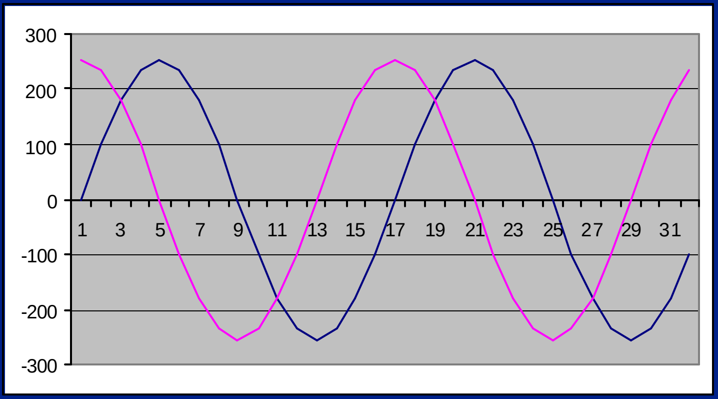
- 32-sample signed Sine and Cosine Wave, F[1]
- Single table can be used by phase shifting sine by 90 degrees or (8) sample points

- Absolute Value of 32-sample Sine and Cosine Wave, F[1]

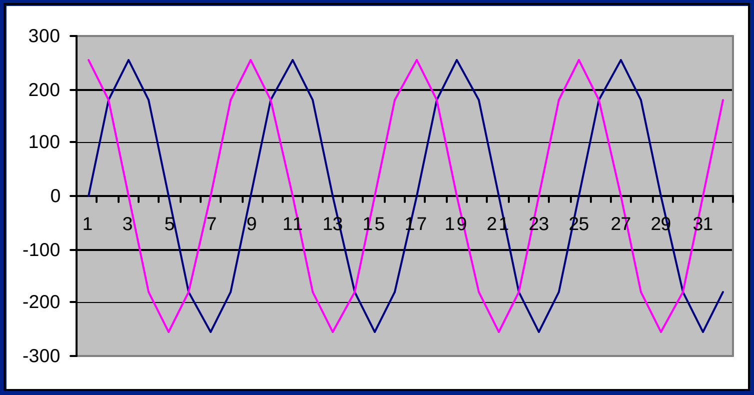
- Increases Resolution by one bit
- Simplifies signed accumulation math in DFT

F[N] Bin Generation

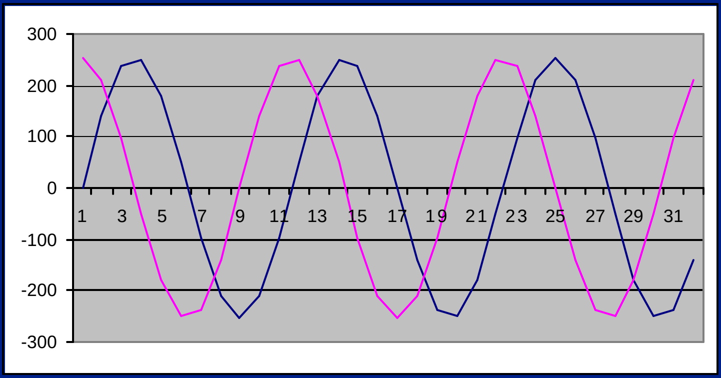
F[2] 312.5 Hz, BINCOUNT = 2



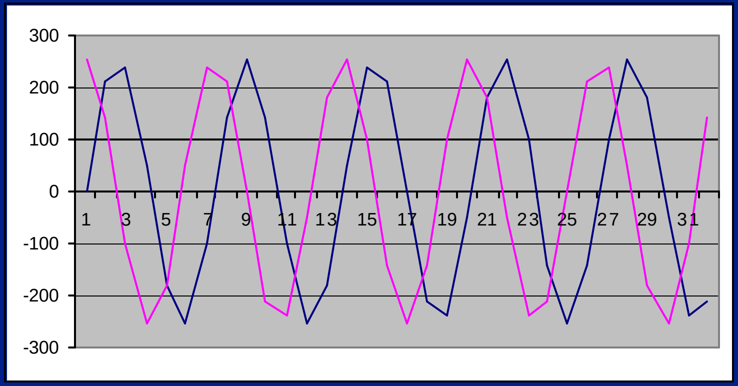
F[4] 625 Hz, BINCOUNT = 4



F[3] 468.75 Hz, BINCOUNT = 3



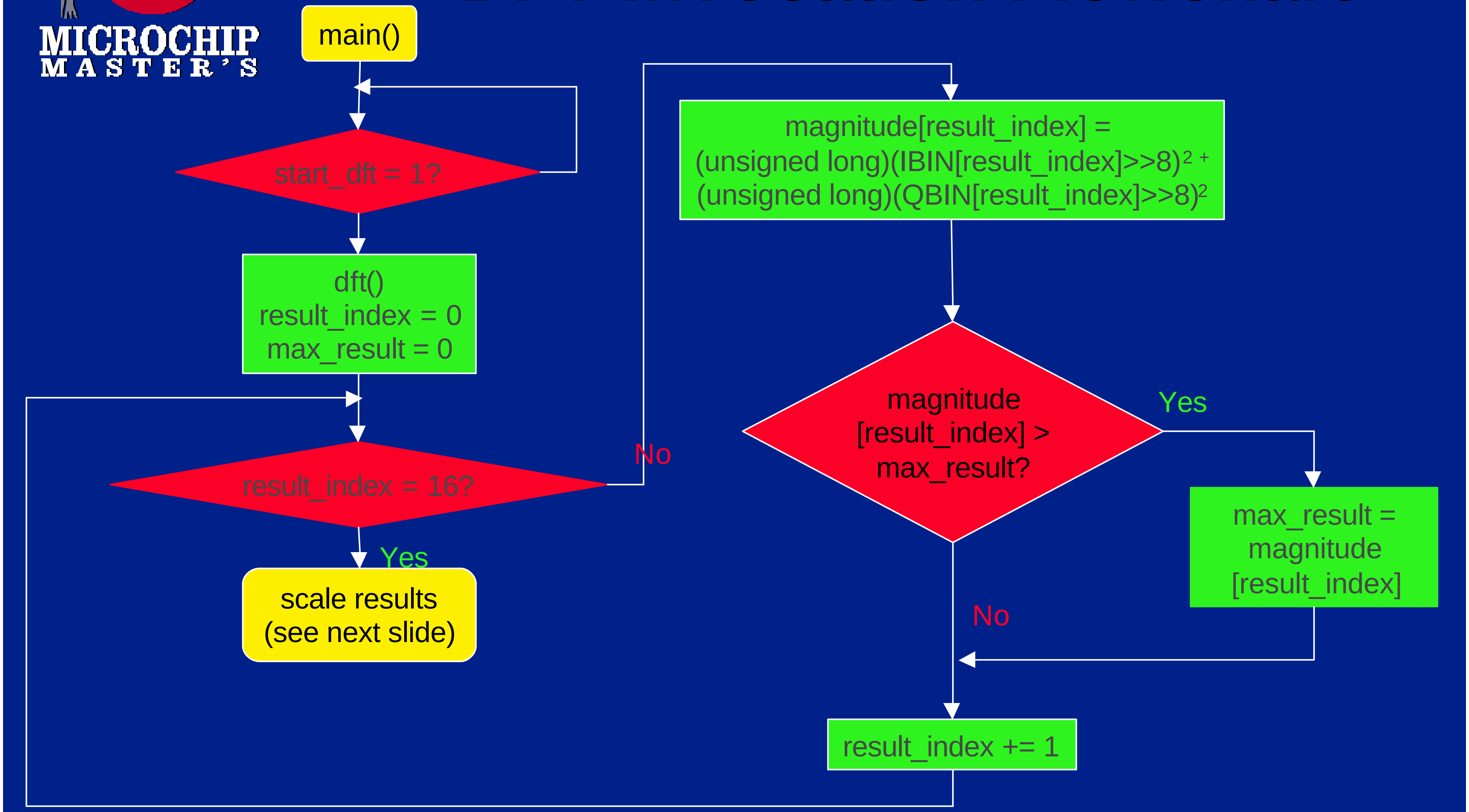
F[5] 781.25 Hz, BINCOUNT = 5



Lab 4: DFT Implementation

- Once 32 samples have been completed:
  - start_dft is set by ISR, indicating INBUFFER[32] is completed
  - Call dft();
    - dft() takes INBUFFER[32], convolves this with FTABLE[32] calculating IBIN[16] and QBIN[16]
  - magnitude[F] = (unsigned long)(IBIN[F]>>8)^2 + (unsigned long)(QBIN[F]>>8)^2
  - Scale magnitude result for 0~16 bar display
  - Use lcd_bargraph(magnitude,location) to plot all 16 frequency magnitude bins

DFT Invocation Flowchart



main()

start_dft = 1?

dft()
result_index = 0
max_result = 0

result_index = 16?

Yes

scale results
(see next slide)

No

magnitude[result_index] =
(unsigned long)(IBIN[result_index]>>8)^2 +
(unsigned long)(QBIN[result_index]>>8)^2

magnitude
[result_index] >
max_result?

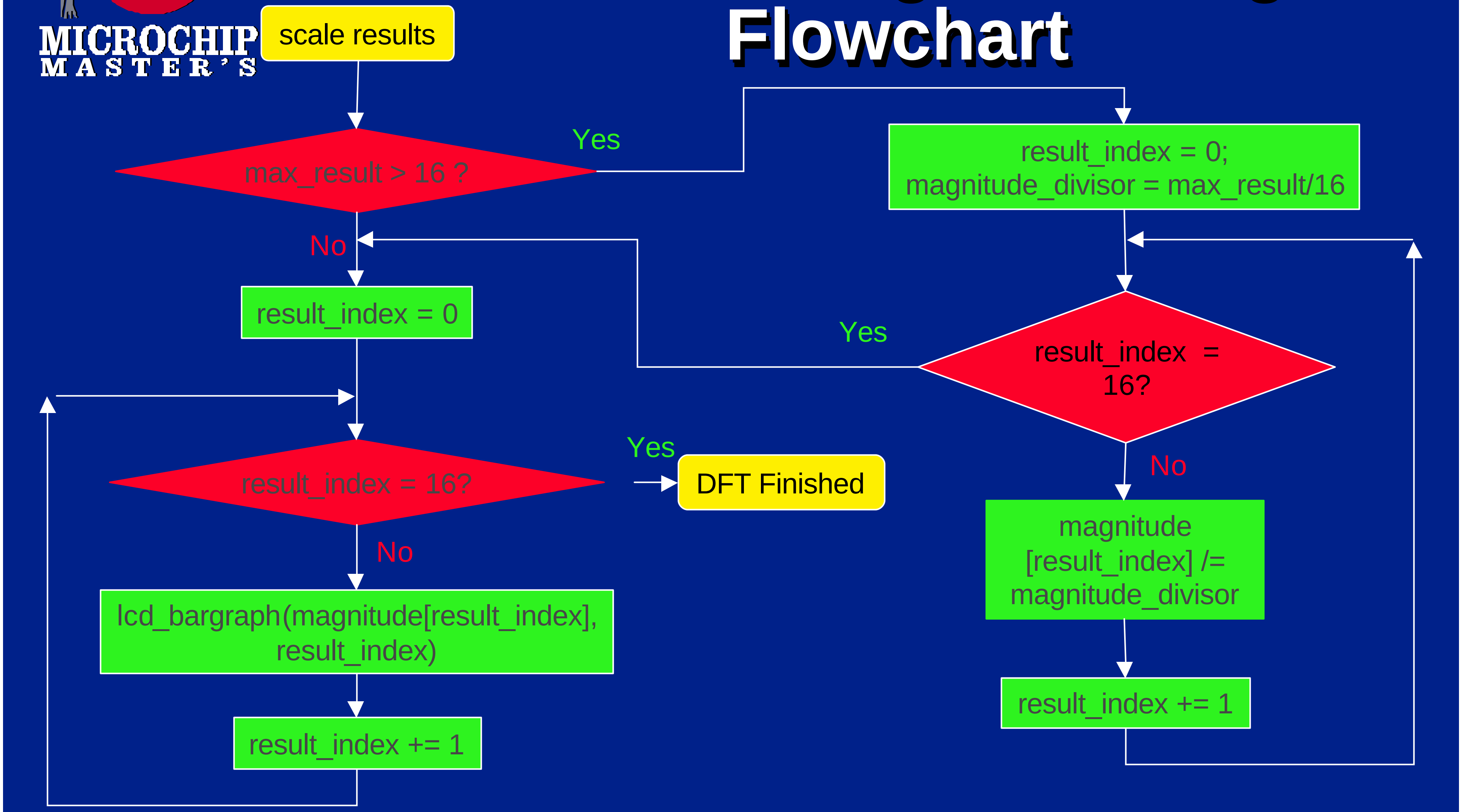
Yes

max_result =
magnitude
[result_index]

No

result_index += 1

DFT Scaling + Plotting Flowchart



scale results

max_result > 16 ?

Yes

result_index = 0;
magnitude_divisor = max_result/16

No

result_index = 0

Yes

result_index = 16?

result_index = 16?

Yes

DFT Finished

No

No

magnitude [result_index] /= magnitude_divisor

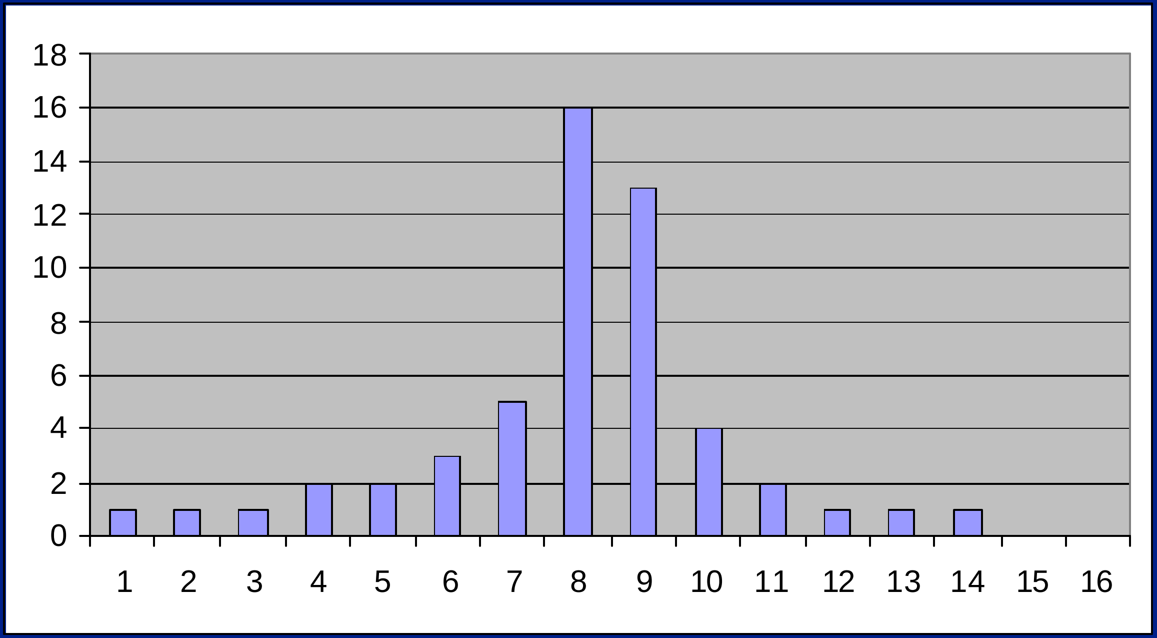
lcd_bargraph(magnitude[result_index], result_index)

result_index += 1

result_index += 1

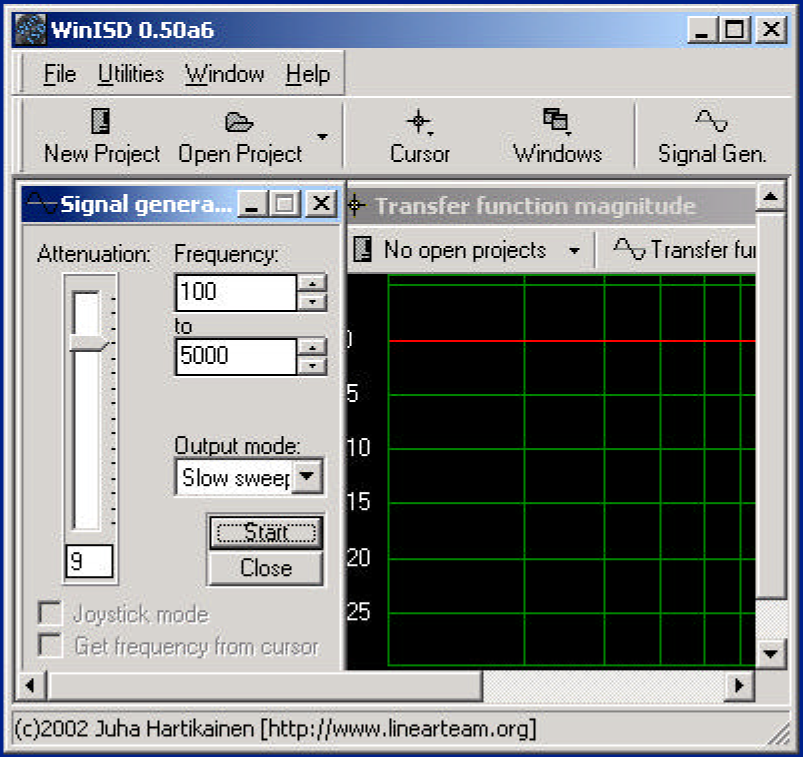
DFT Testing

- Build project, program target and run code
- LCD shows a real-time spectrum analyzer bargraph:



Audio Test Frequency Generator

- WinISD Audio Frequency Generator and Speaker Design Tool created by Juha Hartikainen www.linearteam.org



Frequency: 100 to 5000

Attenuation: 9

Output mode: Slow sweep

(c)2002 Juha Hartikainen [http://www.linearteam.org]

- **Sweeping Audio Tones**
- **Fixed Audio Tones**
- **Attenuation control**
- **Audio Speaker Design Tool**

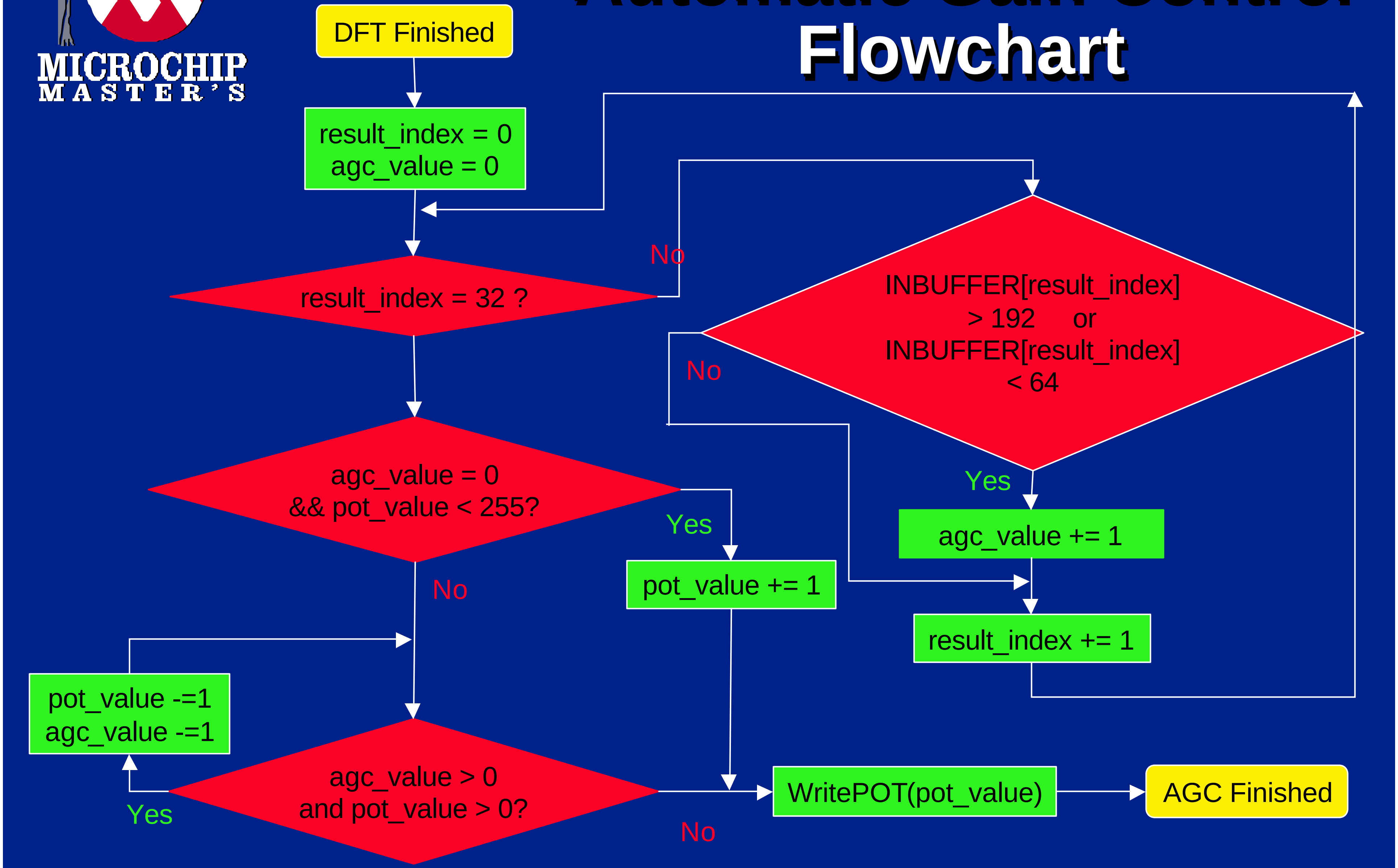
Internal DFT results

- Set breakpoint after lcd_bargraph invocation and View->Watch to look at the following values in watch window:
  - IBIN[16] - Real magnitude
  - QBIN[16] - Imaginary magnitude
  - magnitude[16] - Total magnitude
  - max_result - Maximum magnitude value
  - INBUFFER[32] - Input buffer samples
  - FTABLE[32] - Sine / Cosine waveforms used in DFT convolution.
- Select Decimal display format by right clicking on each variable -> Properties, Format = Decimal

Lab 5: Automatic Gain Control

- Digital POT selects microphone gain
- Gain stored in pot_value. Default = 0xF2
- Use WritePOT(pot_value) to change microphone gain
- Scan INBUFFER[32] for clipped values
  - Centered around 128
  - Clip when < 64 or > 192
- Increase pot_value gain by one until clipping
- Decrease pot_value gain by the number of clipping events

Automatic Gain Control Flowchart



Congratulations, PIC18FXXX Expert…….

- You now have the experience needed to design and complete an embedded systems application using the PIC18FXXX
- We hope you enjoyed this session!
- Please fill out the feedback forms
- Thanks for joining us!

Appendix A:

Improving Code Size With the MPLAB C18 Compiler

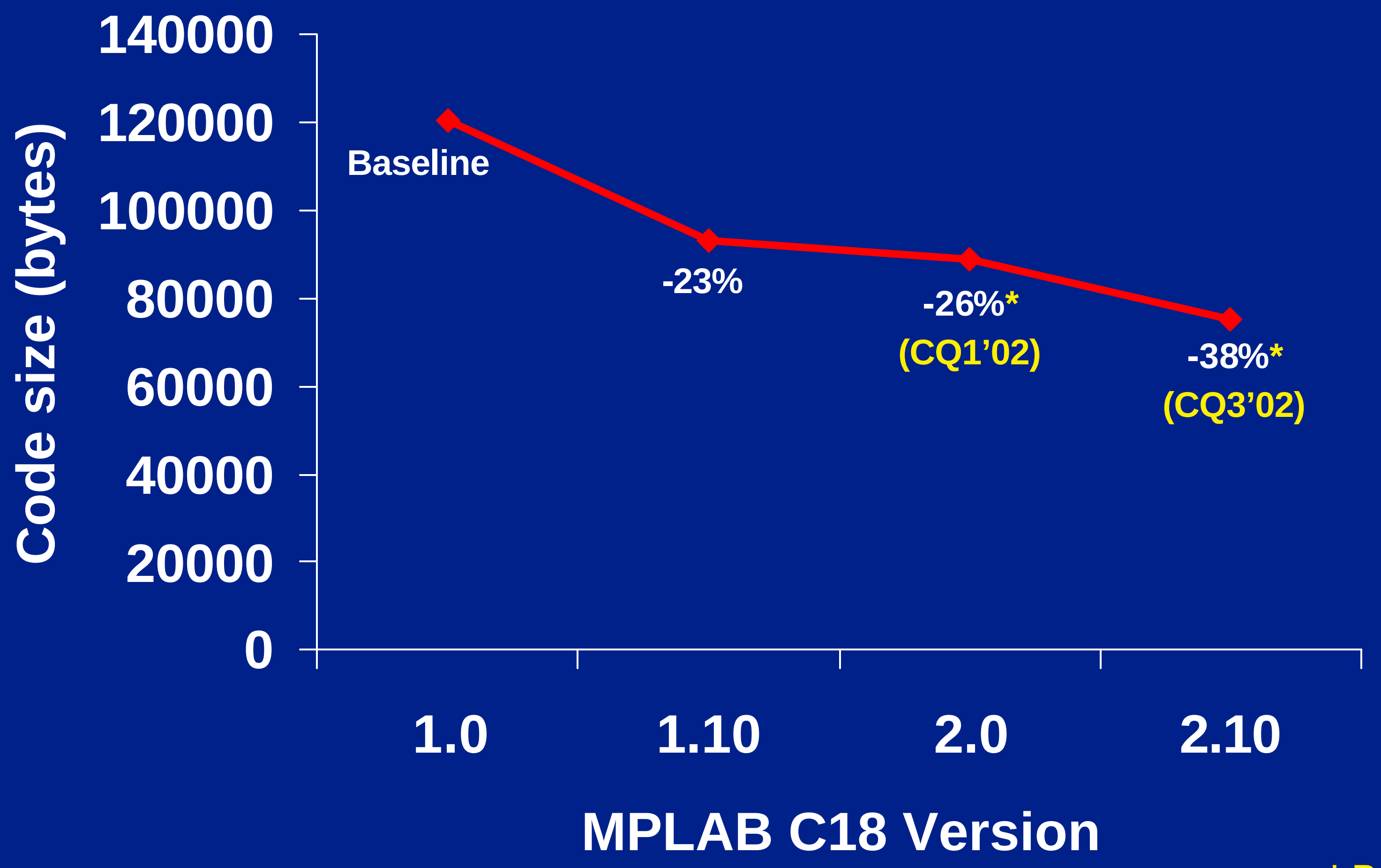
Our Goal: *To understand how to reduce C application code size on PIC18 MCUs through intelligent use of MPLAB C18 and careful structuring of C code.*

Suggestion #1

Use the latest version of MPLAB C18

Code Size Comparison

Default Options



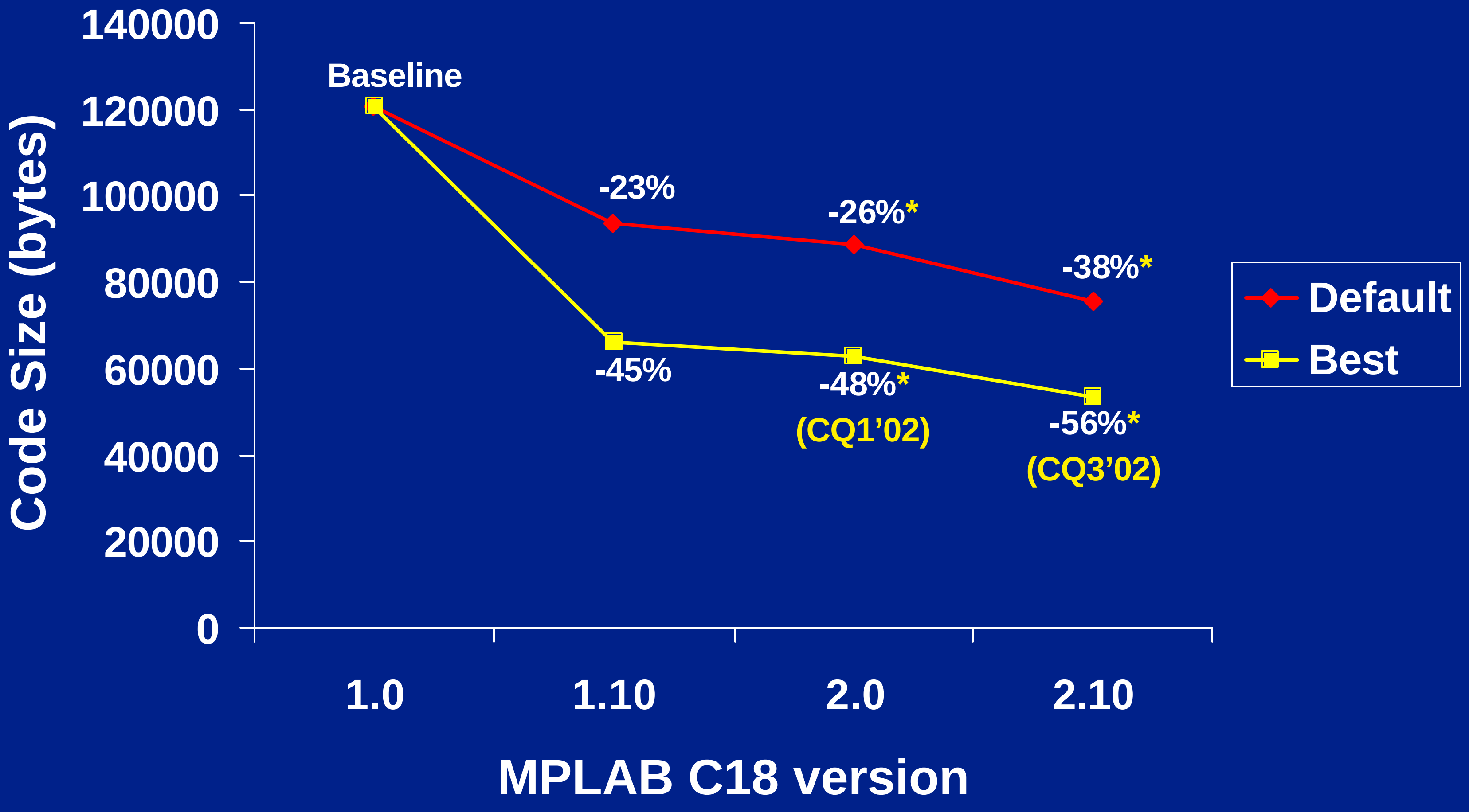
* Projected

Suggestion #2

Carefully select command-line options

Code Size Comparison

Choosing Command-Line Options



Command-Line Options

LFSR Use

- MPLAB-C18’s `-lfsr` switch enables use of the LFSR instruction
- Currently, MPLAB-C18 assumes that LFSR shouldn’t be used without the `-lfsr` switch given
- The switch should always be used when it is known that the LFSR errata doesn’t exist on the targeted part

Command-Line Options

Optimizations

- All of MPLAB-C18's optimizations currently target code size
- Optimizations should be enabled for smallest code size
- NOTE: Optimizations may interfere with MPLAB debugging

Command-Line Options

Memory Model

- MPLAB-C18 has two memory models:
  - `-ms`: small memory model (pointers to program memory are 16-bits wide)
  - `-ml`: large memory model (pointers to program memory are 24-bits wide)
- Use `-ms` whenever possible

Suggestion #3

Select appropriate storage class for data

Command-Line Options

Data Storage Class

- Default storage class for parameters and local variables is **`auto`**
  - ➢ Parameters are passed on the software stack
  - ➢ Locals are located on the software stack

Using auto Variables

Example - calculate the expression (a + b):

```
movlw       offset(a)
movff       PLUSW2, tmp
movlw       offset(b)
movf        PLUSW2
addwf       tmp
```

6 program words

(not counting prolog/epilog)

Command-Line Options

Data Storage Class

- C also provides for **`static`** local variables
- MPLAB-C18 extends C with **`static`** parameters (available in v1.10 and later)
- For example:

```
char add( static char a, static char b )
{
   static char result;
   result = a + b;
   return result;
}
```

Using static Variables

Example - calculate the expression (a + b):

```
movlb      b*
movf       b
addwf      a
```

*likely target for optimization

3 program words

(no prolog/epilog required)

`static` Gotchas

- Gotcha #1 - Reentrant code

  Variables may overwrite themselves

  - Recursion (function calls itself)
  - Function called (directly or indirectly) from main() and an ISR.

`static` Gotchas

- Gotcha #2 - Function pointers

  Address of parameters not known at compile time

- Function pointers may not be used with functions containing static parameters

`static` Gotchas

- Gotcha #3 - Matching declarations

  All declarations must use explicit storage class if not all files are compiled with the same default

- Example:

```
char add( char a, char b );
```

Will only work if the default storage class is identical in both the declaring and defining files.

`static` Gotchas

- What if one of the “static Gotchas” applies to your code?
  - Best case: use `-ol` on all files and explicit `auto` storage class as needed.
  - Intermediate case: Use `-ol` on as many files as possible and explicit storage classes as needed.
  - Worst case: Don’t use `-ol`, but use explicit `static` storage class as much as possible.

Command-Line Options

Data Storage Class

- MPLAB-C18 v2.0 and later extends C with the `overlay` storage class for local variables
  - Behaves identically to the `static` storage class, except:
  - RAM locations are overlaid by the linker when possible based on a call tree analysis
  - Default storage class can be set to `overlay` using the `-sco` option

Suggestion #4

Choose smallest data type possible

MPLAB-C18 Data Types

Type	Min Value	Max Value
`unsigned char`	0	255
`signed char`	-128	127
`unsigned int`	0	65,535
`signed int`	-32,768	32,767
`unsigned short long`	0	16,777,215
`signed short long`	-8,388,608	8,388,607
`unsigned long`	0	4,294,967,295
`signed long`	-2,147,483,648	2,147,483,647

Using Appropriate Data Types

c = a + b

char:

```
MOVLB     b
MOVF      b,0,1
ADDWF     a,0,1
MOVWF     c,1
```

(4 words)

int:

```
MOVLB     a
MOVF      b,0,1
ADDWF     a,0,1
MOVWF     c,1
MOVF      high(b),0,1
ADDWFC    high(a),0,1
MOVWF     high(c),1
```

(7 words)

Suggestion #5

Use access RAM for your variables

Variable Allocation

Using Access RAM

- MPLAB-C18 allows for efficient use of unbanked RAM with the **`near`** type specifier
- RAM variables will default to **`near`** by using the **`-oa`** option
- Compiler won’t emit **`movlb`** instructions for accessing these variables

Variable Allocation

Using Access RAM

- Use the **`near`** specifier for the most frequently accessed variables

- Gotcha: as with **`static`** and **`overlay`**, prototypes must match definitions

Suggestion #6

Keep definitions in same file with references

Variable Allocation

Defining Variables

- MPLAB-C18 can be more aggressive optimizing variables in the files where they are defined.

Source code:

```
char a, b, c;
void foo( void )
{
   c = a + b;
}
```

Machine code:

```
MOVLB     b
MOVF      b,0,1
ADDWF     a,0,1
MOVWF     c,1
```

(4 words)

Variable Allocation

Defining Variables

- MPLAB-C18 must be more conservative with externally-defined variables

Source code:

```
extern char a, b, c;
void foo( void )
{
   c = a + b;
}
```

Machine code:

```
MOVLB     b
MOVF      b,0,1
MOVLB     a
ADDWF     a,0,1
MOVLB     c
MOVWF     c,1
```

(6 words)

Suggestion #7

Use #pragma varlocate

Using #pragma varlocate

- Use #pragma varlocate to tell the compiler what bank a variable is located in

Source code:

```
extern char a, b, c;
void foo( void )
{
   c = a + b;
}
```

Machine code:

```
MOVLB     b
MOVF      b,0,1
MOVLB     a
ADDWF     a,0,1
MOVLB     c
MOVWF     c,1
```

(6 words)

Using #pragma varlocate

- Improves MPLAB-C18 banking optimizer

Source code:

```
#pragma varlocate 3 a, b, c
extern char a, b, c;
void foo( void )
{
   c = a + b;
}
```

Machine code:

```
MOVLB      b
MOVF       b,0,1
ADDWF      a,0,1
MOVWF      c,1
```

(4 words)

Using `#pragma varlocate`

Gotcha: *has no impact on how variables are actually allocated*

Suggestion #8

Replace Common Expressions With Variables

Common Sub-Expression Elimination

- Applies to all types of expressions

Source code:

```
MY_STRUCT s[10];
for(i=0; i<10; i++)
{
    s[i].a = i;
    s[i].b = 34;
}
```

Code size:

```
10 words to calculate s[i]
2 words to assign i
10 words to calculate s[i]
3 words to assign 34

    = 25 words total
```

Common Sub-Expression Elimination (Contd.)

Source code:

```
MY_STRUCT s[10];
MY_STRUCT *p = &(s[0]);

for(i=0; i<10; i++)
{
  p->a = i;
  p->b = 34;
  p++;
}
```

Code size:

```
0 words to calculate s[i]
6 words to assign i
7 words to assign 34
4 words to increment p

    = 17 words total
```

Suggestion #9

Don’t Use a Variable When a Constant Will Do

Constant Evaluations

- Pre-calculate all values that can be determined at compile-time.

Original source:

```
a = 2;
b = 17 + 52 * a;
c = b;
```

Transformed source:

```
c = 121;
```

Appendix B:

PIC18FXXXX Instruction Set and PIC16/17 Migration

PIC18 Architecture
ALU : Status Register

STATUS Register Format

bit 7							bit 0
-	-	-	N	OV	Z	DC	C

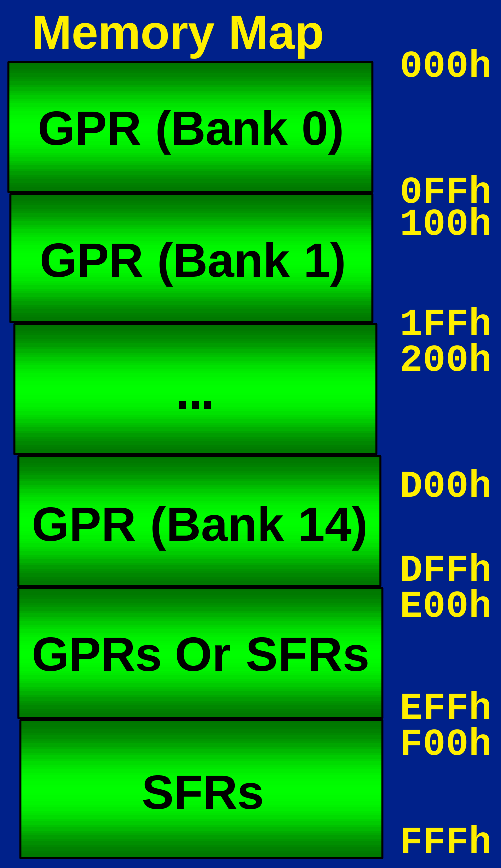
Bit definitions

N	**N**egative/Positive	*ALU result is negative*
OV	**OV**erflow	*2's Complement Overflow occurred*
Z	**Z**ero	*Result is zero*
DC	**D**igit **C**arry / !Borrow	*Carry/borrow from lower nibble*
C	**C**arry / !Borrow	*Carry/borrow from upper nibble*

PIC18 Architecture

Data Memory

- Up to 16 banks of 256 bytes of SRAM
  - Unused banks read '00h'
- Bank selected by BSR<3:0>
- Linear access
- SFR are located in Bank 14 and/or 15

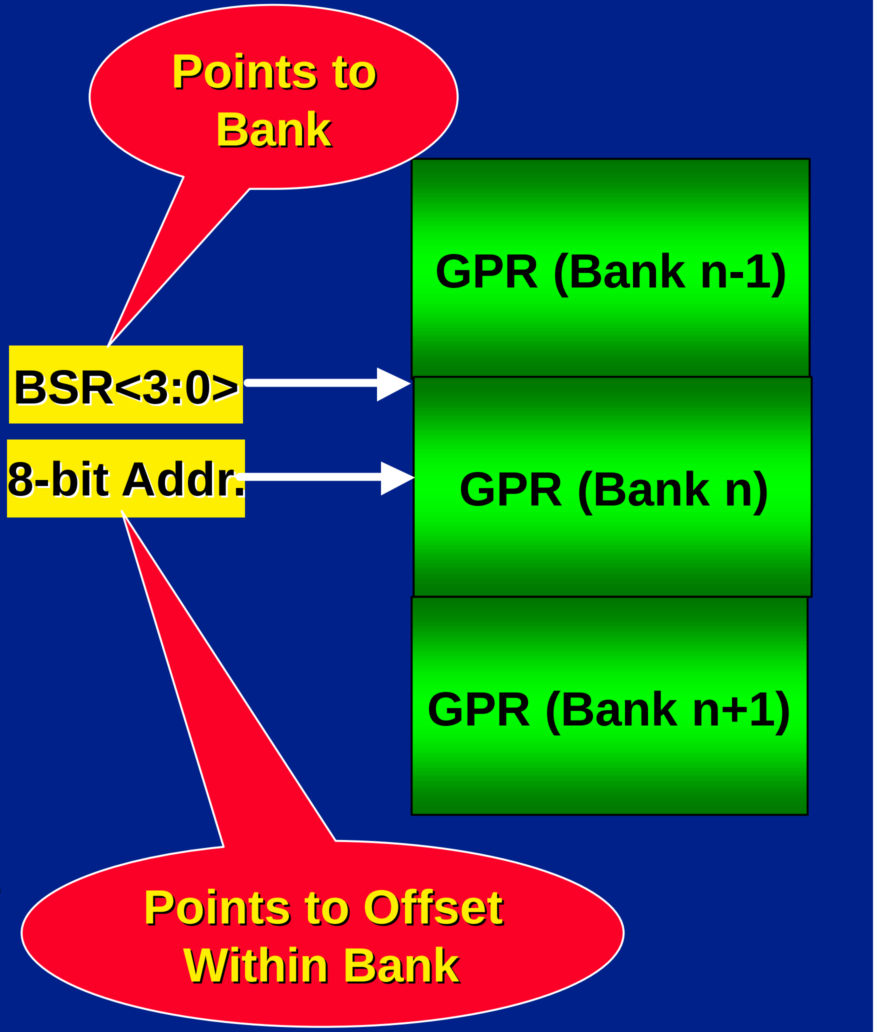


Memory Map

000h
GPR (Bank 0)
0FFh
100h
GPR (Bank 1)
1FFh
200h
...
D00h
GPR (Bank 14)
DFFh
E00h
GPRs Or SFRs
EFFh
F00h
SFRs
FFFh

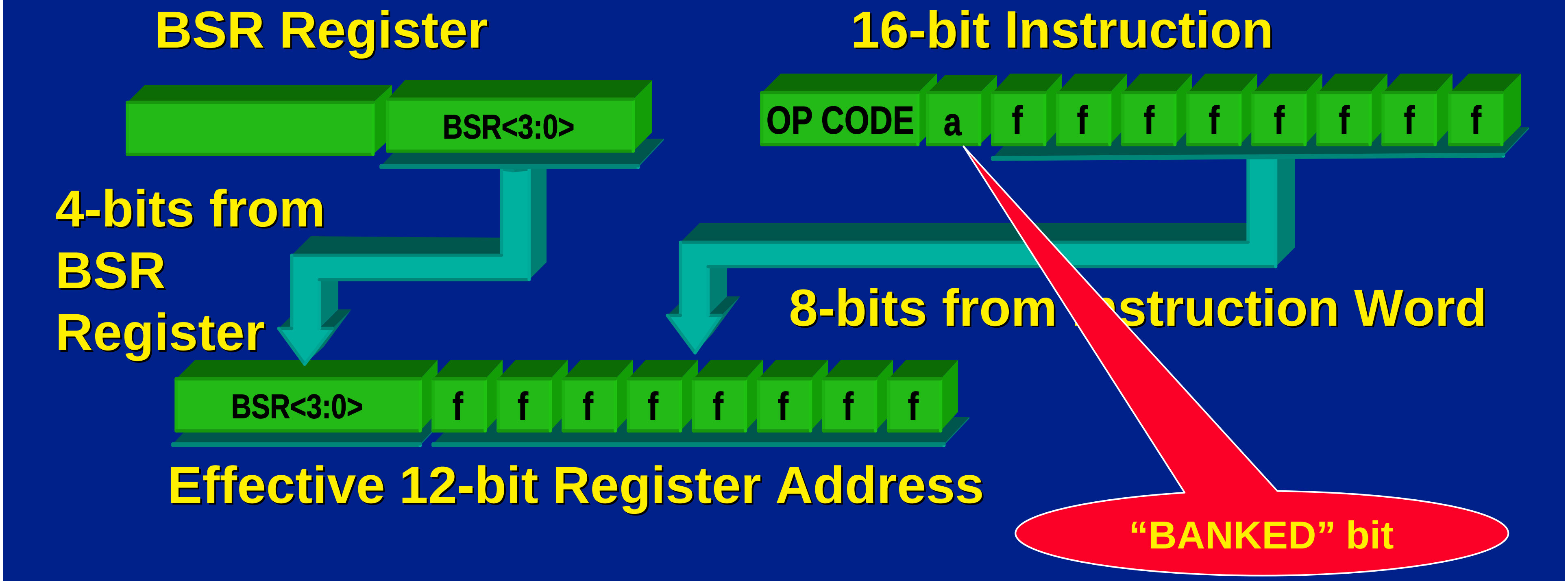
PIC18 Architecture
Accessing Data Memory

- Select a bank
  - BSR<3:0> contains bank
- Instruction with 8-bit address as operand
  - “**BANKED**” bit
- MPASM assembler tip
  - 12-bit Register address
  - Use BANKSEL directive
  - Let MPASM assembler set “**BANKED**” bit



PIC18 Architecture
Accessing Data Memory

- Instruction Format Example:



BSR Register

BSR<3:0>

4-bits from BSR Register

16-bit Instruction

OP CODE a f f f f f f f f

8-bits from Instruction Word

BSR<3:0> f f f f f f f f

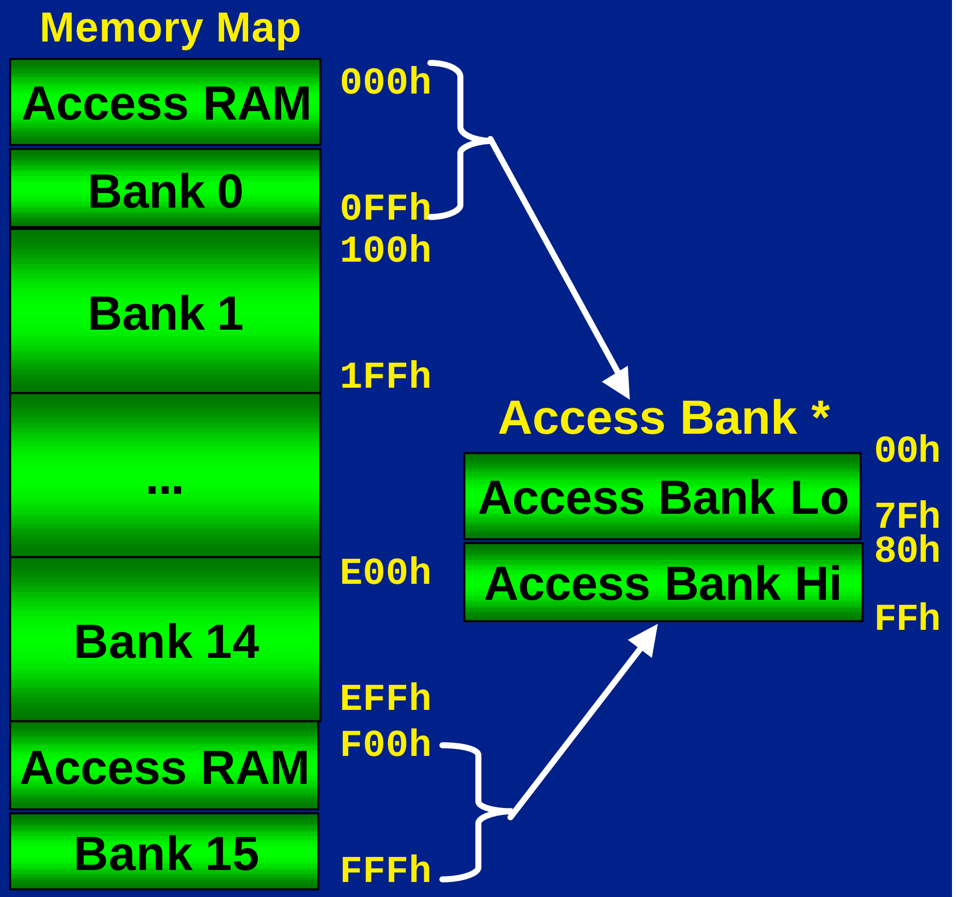
Effective 12-bit Register Address

“BANKED” bit

PIC18 Architecture

Access Bank

- 256 bytes of non-banked memory
- Fast access to frequently used registers (SFRs and GPRs)
- Size of Access Bank depends on device
  - e.g. PIC18FXX2: 128/128; PIC18FXX8: 96/160



Memory Map

Access RAM	000h
Bank 0	0FFh
Bank 1	100h 1FFh
...	
Bank 14	E00h EFFh
Access RAM	F00h
Bank 15	FFFh

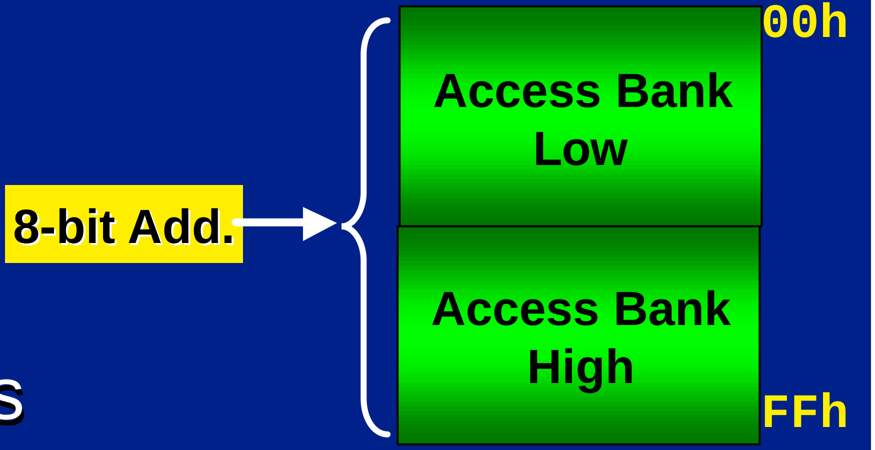
Access Bank *

Access Bank Lo	00h 7Fh
Access Bank Hi	80h FFh

* Note: Check your device datasheet

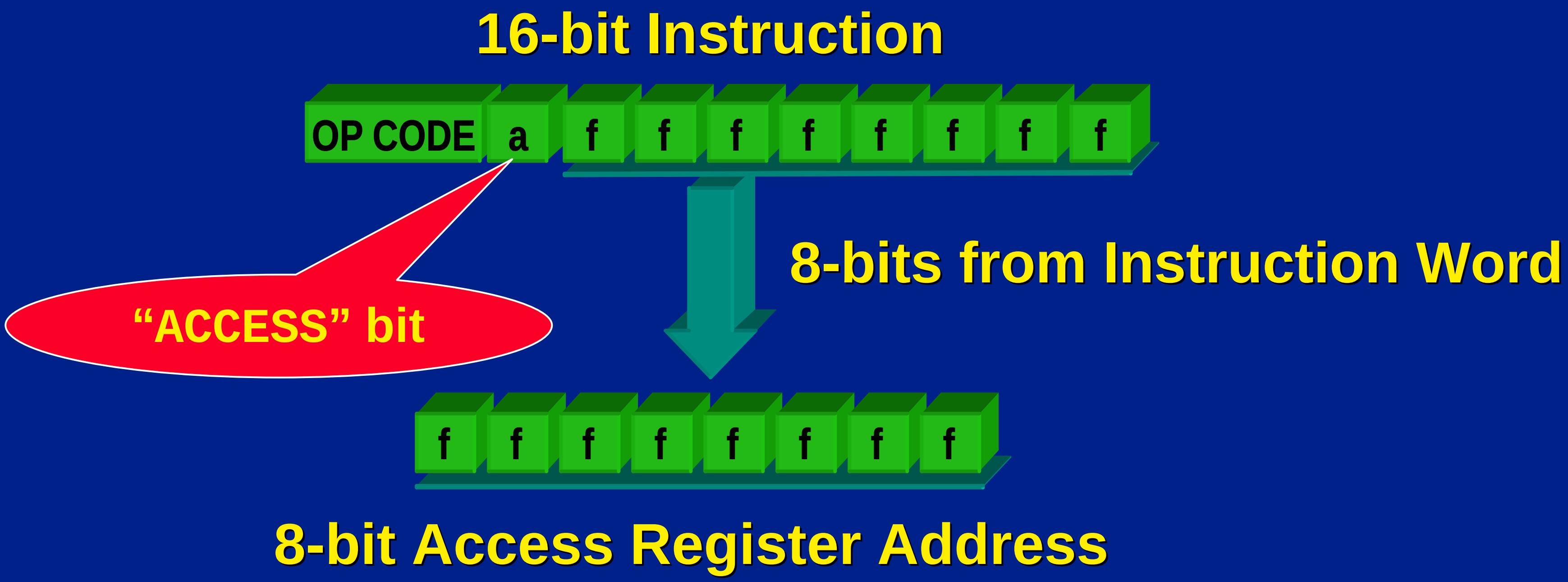
PIC18 Architecture
Accessing Access Bank

- Instruction with 8-bit address as operand
  - Special “**ACCESS**” bit
- MPASM assembler tip
  - 12-bit Register address
  - Let MPASM assembler set “**ACCESS**” bit



PIC18 Architecture
Accessing Access Bank

- Instruction Format Example:



PIC18 Architecture

Program Memory Storage Scheme

- Little-Endian Format

Instruction	Opcode	Memory	Address
...			00007h
MOVLW 55h	0E55h	55h	00008h
		0Eh	00009h
GOTO 06h	EF03h, F000h	03h	0000Ah
		EFh	0000Bh
		00h	0000Ch
		F0h	0000Dh

PIC18 Instructions

Instruction Features

- Upward compatible with PIC16, PIC17,

  16-bit Instruction width
- Instruction fetches are 16-bit wide
  - Fetch and Execution is overlapped
- Single Cycle 8 x 8 Multiply
- Generates compact code
- Most Instructions are Orthogonal

MICROCHIP MASTER'S

PIC18 Instructions
Instruction Features (Continued)

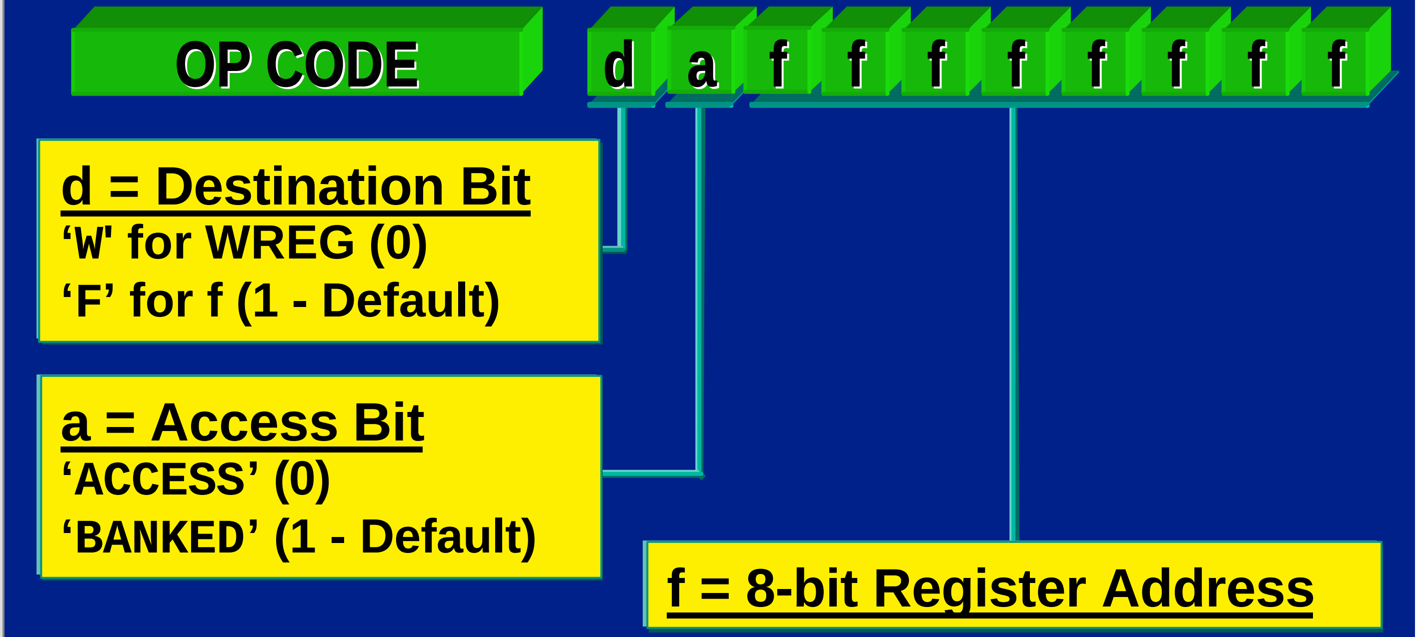
- Most Instructions are Single Word
  - 71 Single Word; 4 Double Word
- Most Instructions are Single Cycle
  - 17 are Double Cycle
  - 18 conditional branch/skips are 1, 2 (or 3)
- Register to Register transfer instruction
- Powerful bit manipulation
  - Available for entire data memory region

PIC18 Instruction

Byte-Oriented Operations

Byte-Oriented Operations	
ADDWF	f [,d [,a]]
ADDWFC	f [,d [,a]]
ANDWF	f [,d [,a]]
CLRF	f [,a]
COMF	f [,d [,a]]
CPFSEQ	f [,a]
CPFSGT	f [,a]
CPFSLT	f [,a]
DECF	f [,d [,a]]
DECFSZ	f [,d [,a]]
DCFSNZ	f [,d [,a]]
INCF	f [,d [,a]]
INCFSZ	f [,d [,a]]
INFSNZ	f [,d [,a]]
IORWF	f [,d [,a]]
MOVF	f [,d [,a]]
MOVFF	fs, fd
MOVWF	f [,a]

16-bit Instruction for Byte Oriented Operations



d = Destination Bit
'W' for WREG (0)
'F' for f (1 - Default)

a = Access Bit
'ACCESS' (0)
'BANKED' (1 - Default)

f = 8-bit Register Address

Example:

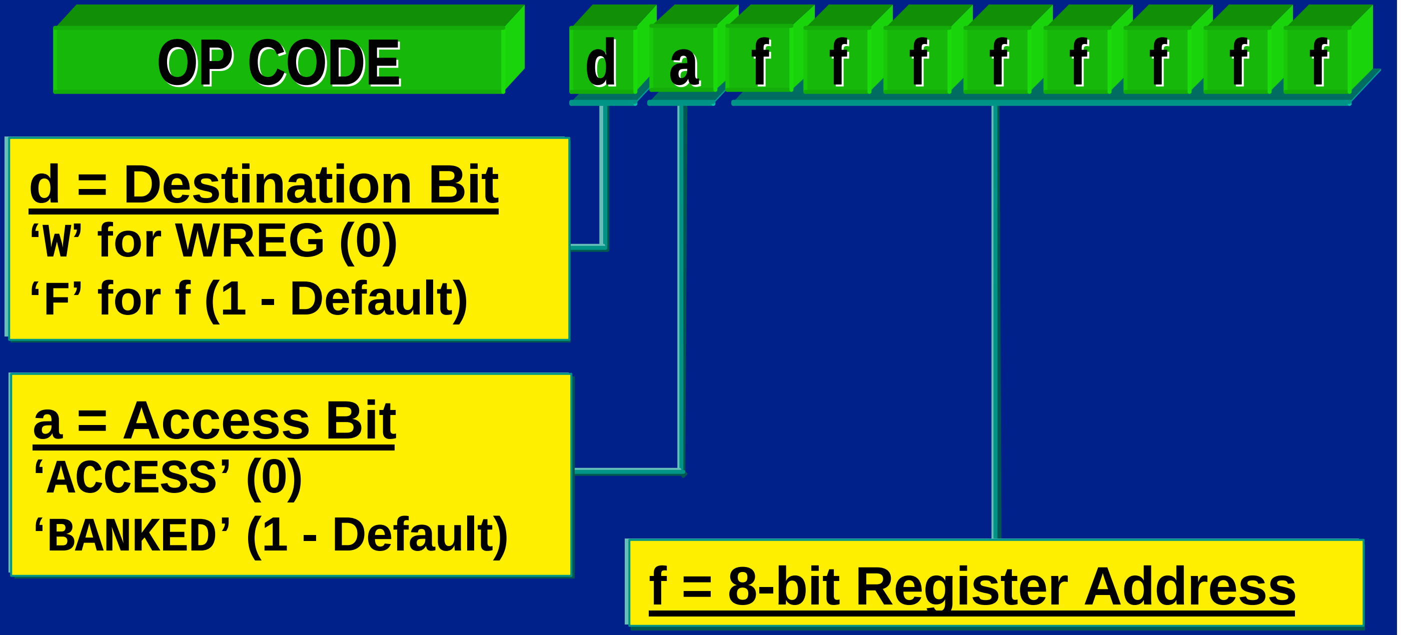
```
ADDWF     f [,d [,a]]      MOVFF fs, fd
ADDWF     Count            MOVFF Source, Dest
```

PIC18 Instructions

Byte-Oriented Operations (Continued)

Byte-Oriented Operations	
MULWF	f [,a]
NEGF	f [,a]
RLCF	f [,d [,a]]
RLNCF	f [,d [,a]]
RRCF	f [,d [,a]]
RRNCF	f [,d [,a]]
SETF	f [,a]
SUBFWB	f [,d [,a]]
SUBWF	f [,d [,a]]
SUBWFB	f [,d [,a]]
SWAPF	f [,d [,a]]
TSTFSZ	f [,a]
XORWF	f [,d [,a]]

16-bit Instruction for Byte Oriented Operations



d = Destination Bit
‘W’ for WREG (0)
‘F’ for f (1 - Default)

a = Access Bit
‘ACCESS’ (0)
‘BANKED’ (1 - Default)

f = 8-bit Register Address

Example:

```
SUBWF     f [,d [,a]]
SUBWF     Value, W
```

PIC18 Instructions

Byte-Oriented Operations - Example

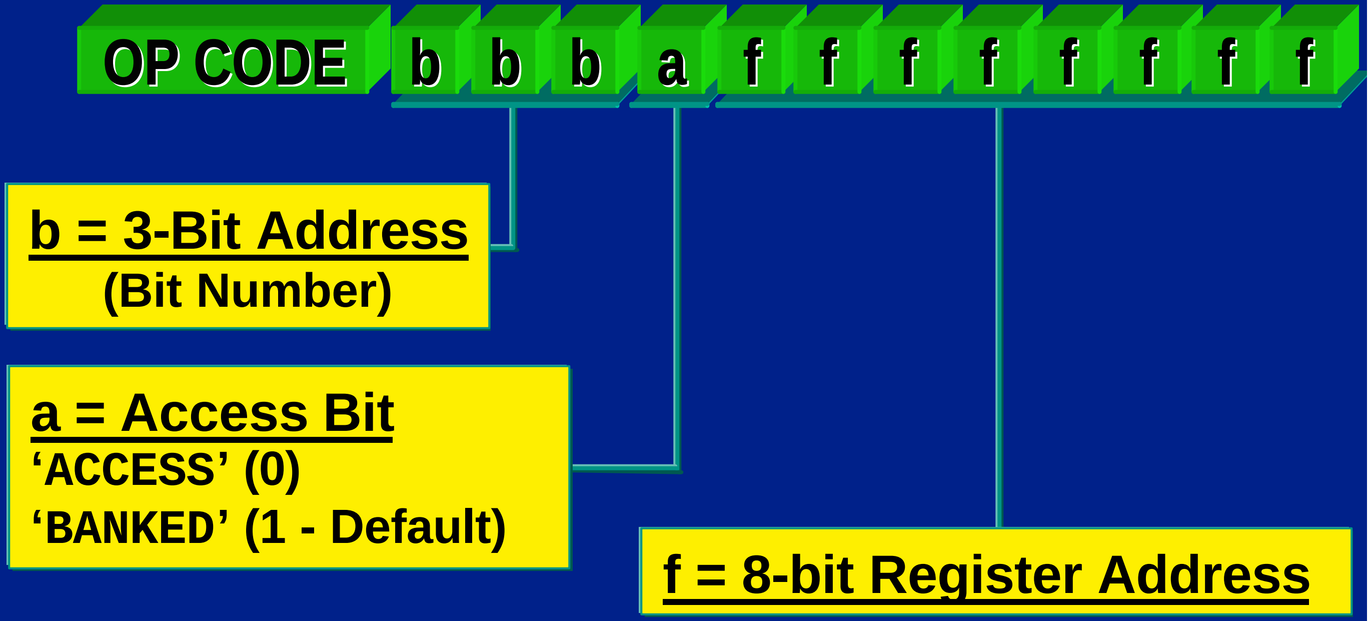
- Perform Multi-byte (4 byte) increment: “Count32++”

```
...
movlw      01h
addwf      Count32, F       ; Inc LSB by ‘1’
clrf       WREG             ; Pass the carry
addwfc     Count32+1, F     ; to LOW MSB
addwfc     Count32+2, F     ; to HIGH LSB
addwfc     Count32+3, F     ; to HIGH MSB
...
```

PIC18 Instructions
Bit-Oriented Operations

Bit-Oriented Operations	
BCF	f, b [,a]
BSF	f, b [,a]
BTG	f, b [,a]
BTFSC	f, b [,a]
BTFSS	f, b [,a]

16-bit Instruction for Bit Oriented Operations



OP CODE | b | b | b | a | f | f | f | f | f | f | f | f

b = 3-Bit Address
(Bit Number)

a = Access Bit
'ACCESS' (0)
'BANKED' (1 - Default)

f = 8-bit Register Address

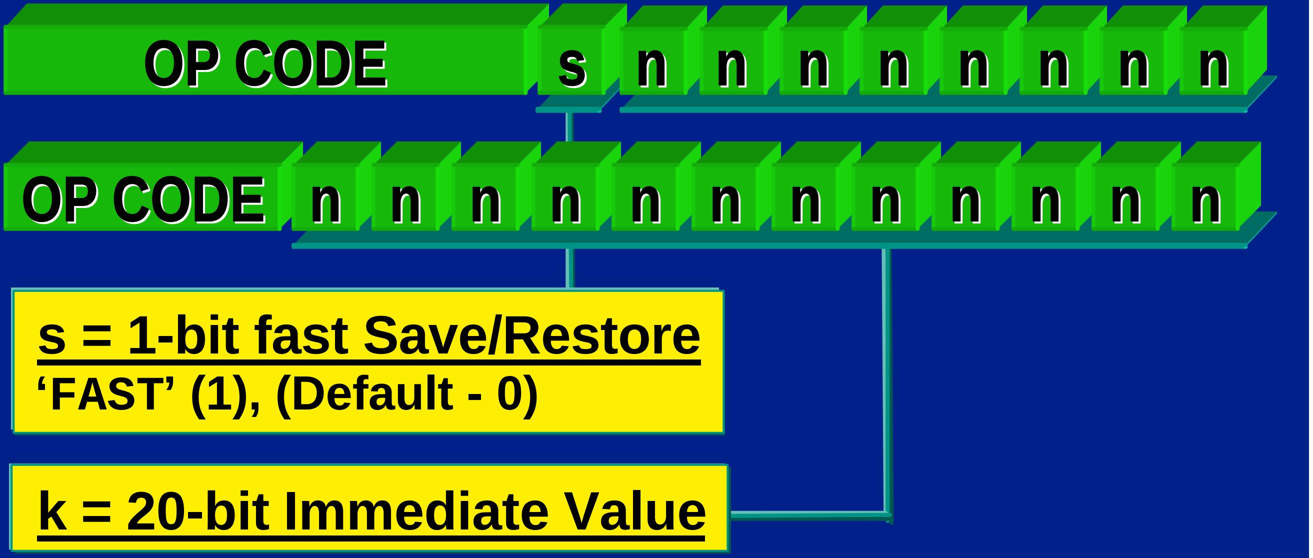
Example:

```
BTFSC     f, b [,a]
BTFSC     STATUS, C
```

PIC18 Instructions
Control Operations

Control Operations	
BC	n
BN	n
BNC	n
BNN	n
BNOV	n
BNZ	n
BOV	n
BRA	n
BZ	n
CALL	n [,s]
GOTO	n
RCALL	n
RETFIE	[s]
RETURN	[s]

16-bit Instruction for CALL and GOTO



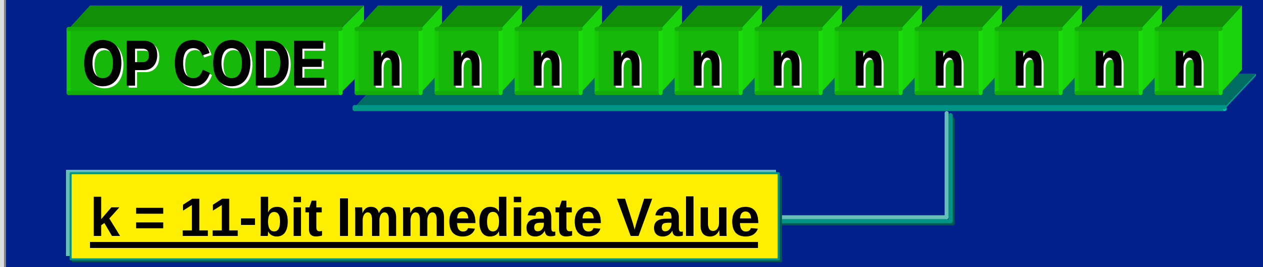
OP CODE | s | n | n | n | n | n | n | n | n

OP CODE | n | n | n | n | n | n | n | n | n | n | n | n

s = 1-bit fast Save/Restore
'FAST' (1), (Default - 0)

k = 20-bit Immediate Value

16-bit Instruction for RCALL and BRA



OP CODE | n | n | n | n | n | n | n | n | n | n | n

k = 11-bit Immediate Value

PIC18 Instructions

Control Operations (Continued)

Control Operations	
BC	n
BN	n
BNC	n
BNN	n
BNOV	n
BNZ	n
BOV	n
BRA	n
BZ	n
CALL	n [,s]
GOTO	n
RCALL	n
RETFIE	[s]
RETURN	[s]

- (Un)Conditional branches spans -128 through +127 Instructions
- **CALL** and **GOTO** contain full 21-bit address
  - Provides Linear access to 2MB
- RCALL spans -1024 through 1023 Instructions

PIC18 Instructions

Control Operations (Continued)

Control Operations

- CLRWDT
- DAW
- NOP
- POP
- PUSH
- RESET
- SLEEP

16-bit Instruction for **CLRWDT**

OP CODE

16-bit Instruction for **DAW**

OP CODE

PUSH and **POP** operate on Hardware Stack only

DAW operates on WREG only

PIC18 Instructions

Control Operations - Example #1

Utilize “Save Context”

Handling Interrupt

```
org         00008h
bra         HighISR
...
HighISR:
...
retfie      FAST
...
```

PIC18 Instructions

Control Operations - Example #2

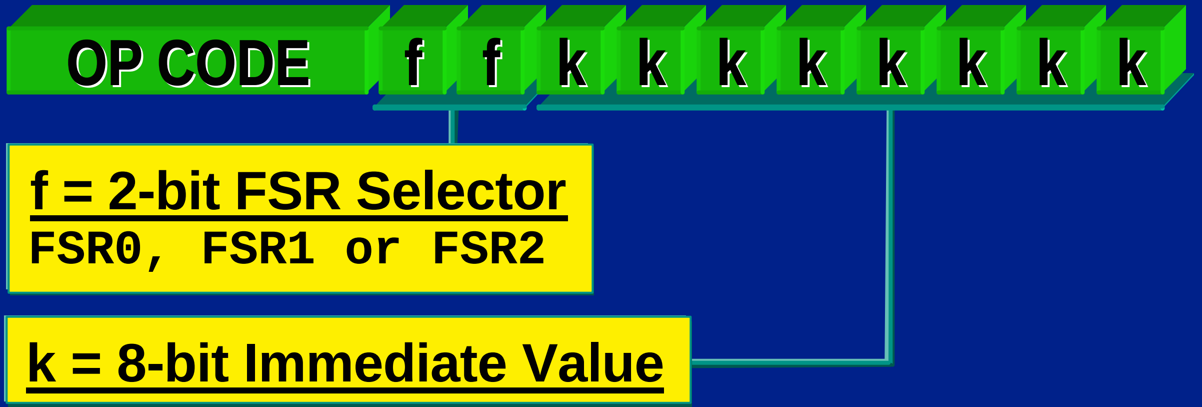
Wait for an input trigger on PORTB RB6 pin

```
...
btfsc     PORTB, RB6       ; Is RB6 low ?
bra       $-2              ; No.  Wait…
...                        ; Yes.
```

PIC18 Instructions
Literal Operations

Literal Operations	
ADDLW	k
ANDLW	k
IORLW	k
LFSR	f, k
MOVLB	k
MOVLW	k
MULLW	k
RETLW	k
SUBLW	k
XORLW	k

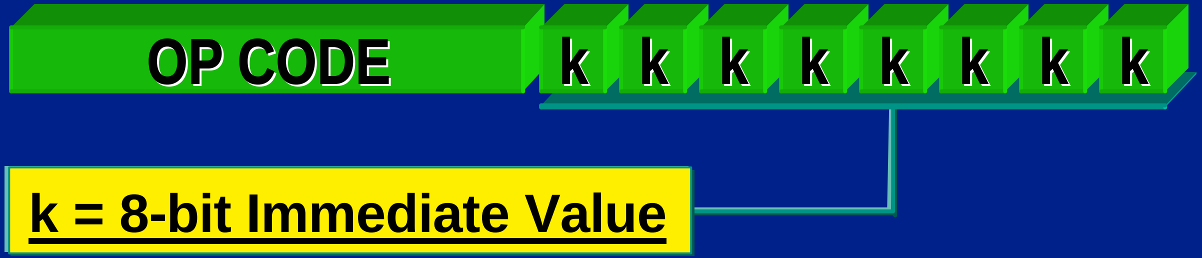
16-bit Instruction for LFSR



f = 2-bit FSR Selector
`FSR0, FSR1 or FSR2`

k = 8-bit Immediate Value

16-bit Instruction for Other Literal Operations



k = 8-bit Immediate Value

Example:

```
MOVLW     k            LFSR  f, k
MOVLW     5Ah          LFSR  FSR0, 400h
```

PIC18 Instructions

Literal Operations - Example

Immediate Operation

```
...
movlw      55h
movwf      PORTB
...
```

Indirect Operation

```
...
lfsr FSR0, 400h
movwf INDF0     ; *FSR0
movwf POSTINC0 ; *FSR0++
movwf POSTDEC0 ; *FSR0--
movwf PREINC0  ; *++FSR0
movwf PLUSW0   ; FSR0[WREG]
```

PIC18 Instructions

Data ↔ Program Operations

Control Operations

TBLRD*
TBLRD*+
TBLRD*-
TBLRD+*
TBLWT*
TBLWT*+
TBLWT*-
TBLWT+*

16-bit Instruction for TBLRD*/TBLWT*

OP CODE

TBLRD and **TBLWT** operate on TABLAT only

Example:

TBLRD*

TBLRD*+

PIC18 Instructions

Data ↔ Program Operations - Example

Read a lookup table entry:

```
movlw     upper(LookUpTable)   ;  Load look-up
movwf     TBLPTRU              ;  table
movlw     high(LookupTable)    ;  address
movwf     TBLPTRH
movlw     low(LookupTable)
movwf     TBLPTRL
tblrd*+                        ;  Read it.
```

PIC16C/FXXX to PIC18FXXXX Source Code Conversion Tips

PIC16F/CXXX to PIC18FXXXX Assembly Compatibility

- C source code on most PIC16C/FXXX platforms will directly port to PIC18FXXXX
- Compatible Assembly code source except:
  - Absolute constants used for program memory
  - Computed GOTO (addwf PCL,F)
  - RAM requirements above 256 bytes are selected by BSR not RP0 and RP1 bits
  - FSR is 12 bits wide, also includes auto increment
- Double check immediate constants when initializing peripherals

Code Conversion Tip

- Data Memory Access

  bsf STATUS,RP0 bcf STATUS,RP0
- These instructions can be ignored because bits 7,6,5 in STATUS register are unused
- For devices with less than 256 bytes of RAM, it is not necessary to be concerned with RAM locations. Why is this the case?
- Most memory accesses can be done in Access Bank
- Assembler will automatically select “a” bit when applicable
  - Address locations now use 12-bit values.
  - Set the BSR if you need to.

Code Conversion Tip

- PCLATU, PCLATH
  - CALL, GOTO instructions write directly to the program counter.
  - Operations to the PC latches before a CALL or GOTO will be ignored.
- Program addresses are now BYTE addresses
  - If labels are used, then any moves to PCLATH are still OK
  - If absolute values are used, then they must be modified
  - Example
    ```
    Goto $+1       ; PIC16CXXX
    Goto $+2       ; PIC18FXXX
    ```

Code Conversion Tip

- Conditional GOTO, Tables

```
Code    movlw   HIGH Table  ;(Table must be a label)
        movwf   PCLATH
        movlw   offset
        call    Table
.....
Table   addwf   PCL
        retlw   ‘A’
        retlw   ‘B’
.....
```

- What’s wrong with this code?

Code Conversion Tip

- Conditional GOTO, Tables

```
Code    movlw   HIGH Table
        movwf   PCLATH
        bcf     STATUS,C
        rlncf   offset,W     ; So, multiply offset by 2
        call    Table
.....
Table   addwf   PCL          ; On PIC18, this is a BYTE address
        retlw   'A'
        retlw   'B'
.....
```

Code Conversion Tip

- Be particularly careful about loading registers

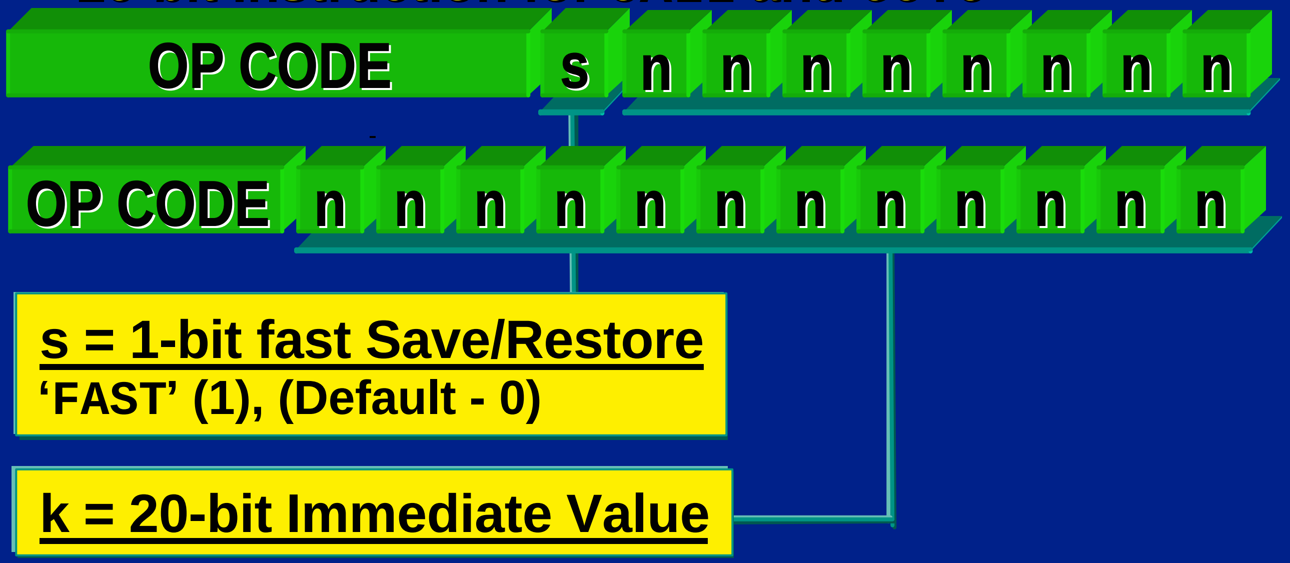
```
movlw    B’00100110’
movwf    register
```

- Most registers are compatible, but there are differences

- Use of symbolic bit names is safest

Code Conversion Tip

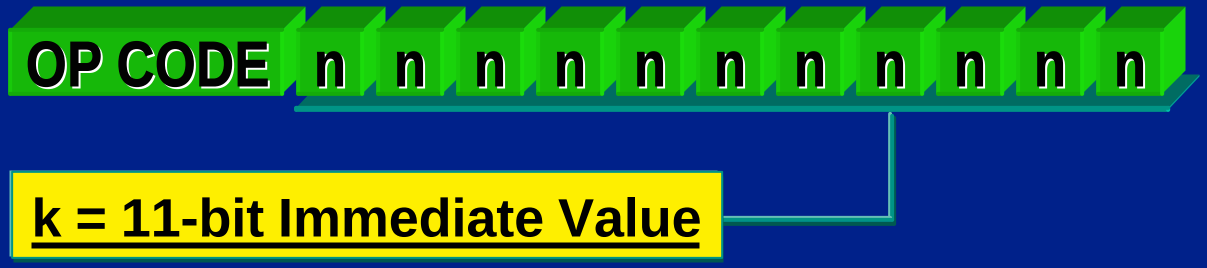
16-bit Instruction for CALL and GOTO



s = 1-bit fast Save/Restore
‘FAST’ (1), (Default - 0)

k = 20-bit Immediate Value

16-bit Instruction for RCALL and BRA



k = 11-bit Immediate Value

Appendix C:

PIC18FXXXX Flash Programming Tips

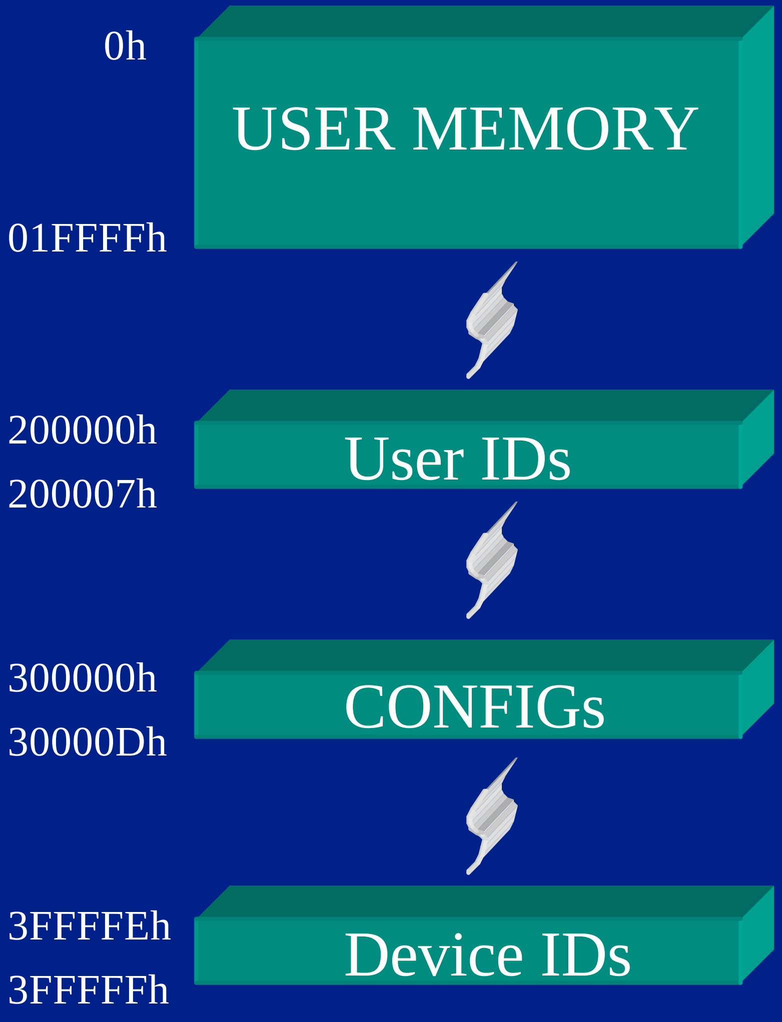
PIC18F FLASH

Program Memory Reads and Writes

- READs performed on bytes
- Can READ entire user Program Memory of up to 2M plus:
  - User ID locations 200000h-200007h
  - CONFIG registers 300000h-30000Dh
  - Device ID registers 3FFFFEh,3FFFFFh
- To READ Program Memory:
  - Load TBLPTRU,TBLPTRH,TBLPTRL
  - Execute one of the TBLRDs
    - TBLRD*, TBLRD*+, TBLRD+*, TBLRD*-
    - result in TABLAT

PIC18F ARCHITECTURE

- **18F Addressable Memory is divided into:**
  - USER MEMORY:
    - Up to 128 Kbytes internal
    - Up to 2 Mbytes external
  - USER IDs:
    - 8 modifiable bytes
  - CONFIGs:
    - Device settings, code protects, etc
  - DEVICE IDs:
    - Part and rev. signature



0h

USER MEMORY

01FFFFh

200000h

User IDs

200007h

300000h

CONFIGs

30000Dh

3FFFFEh

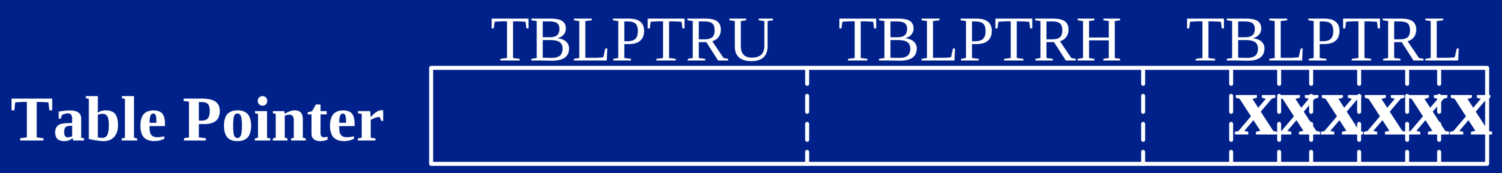
Device IDs

3FFFFFh

PIC18F FLASH

Program Memory Reads and Writes

- **ERASING User memory (USER MODE):**
  - Performed on 64 bytes (32 words)
  - Load TBLPTRU,TBLPTRH,TBLPTRL
    - TBLPTR 6 LSBs are don’t cares
  - Configure EECON1
  - Disable interrupts
  - Perform programming sequence
  - Start ERASE
    - Internally timed, NO CODE EXECUTION
  - Re-enable interrupts



TBLPTRU TBLPTRH TBLPTRL

Table Pointer

XXXXXX

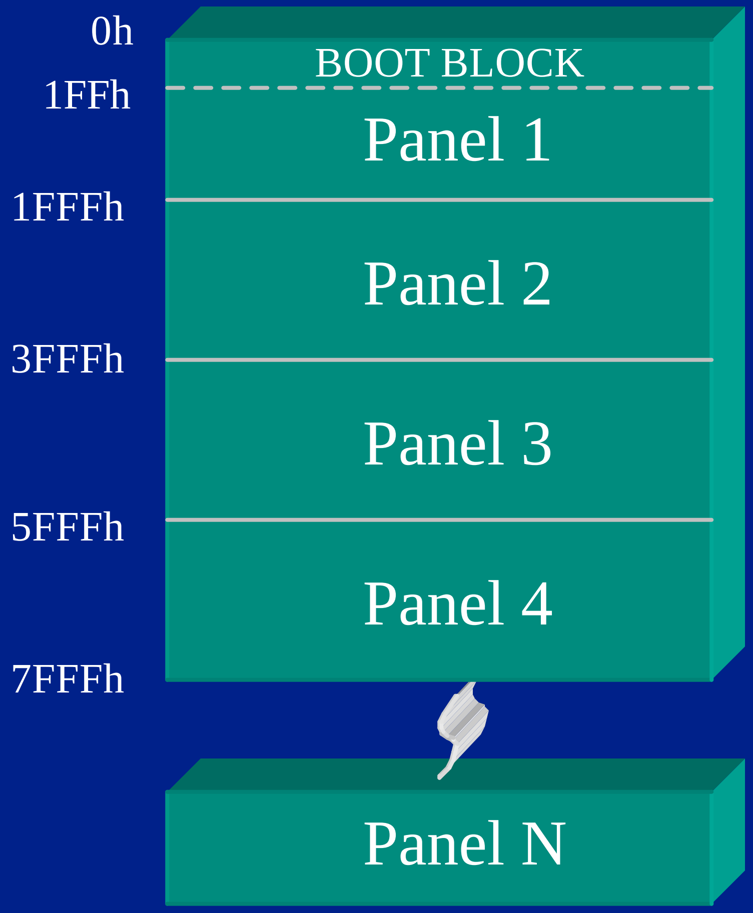
PIC18F FLASH

Program Memory Reads and Writes

- **WRITEs to User memory (USER MODE):**
  - Performed on 8 bytes (4 words)
  - Load TBLPTRU,TBLPTRH,TBLPTRL
  - Load 8 bytes into write buffers by 8 table write instructions
    - TBLWT*,TBLWT*+,TBLWT*-,TBLWT+*
  - Configure EECON1
  - Disable interrupts
  - Perform programming sequence
  - Start WRITE
    - Internally timed, NO CODE EXECUTION
  - Re-enable interrupts

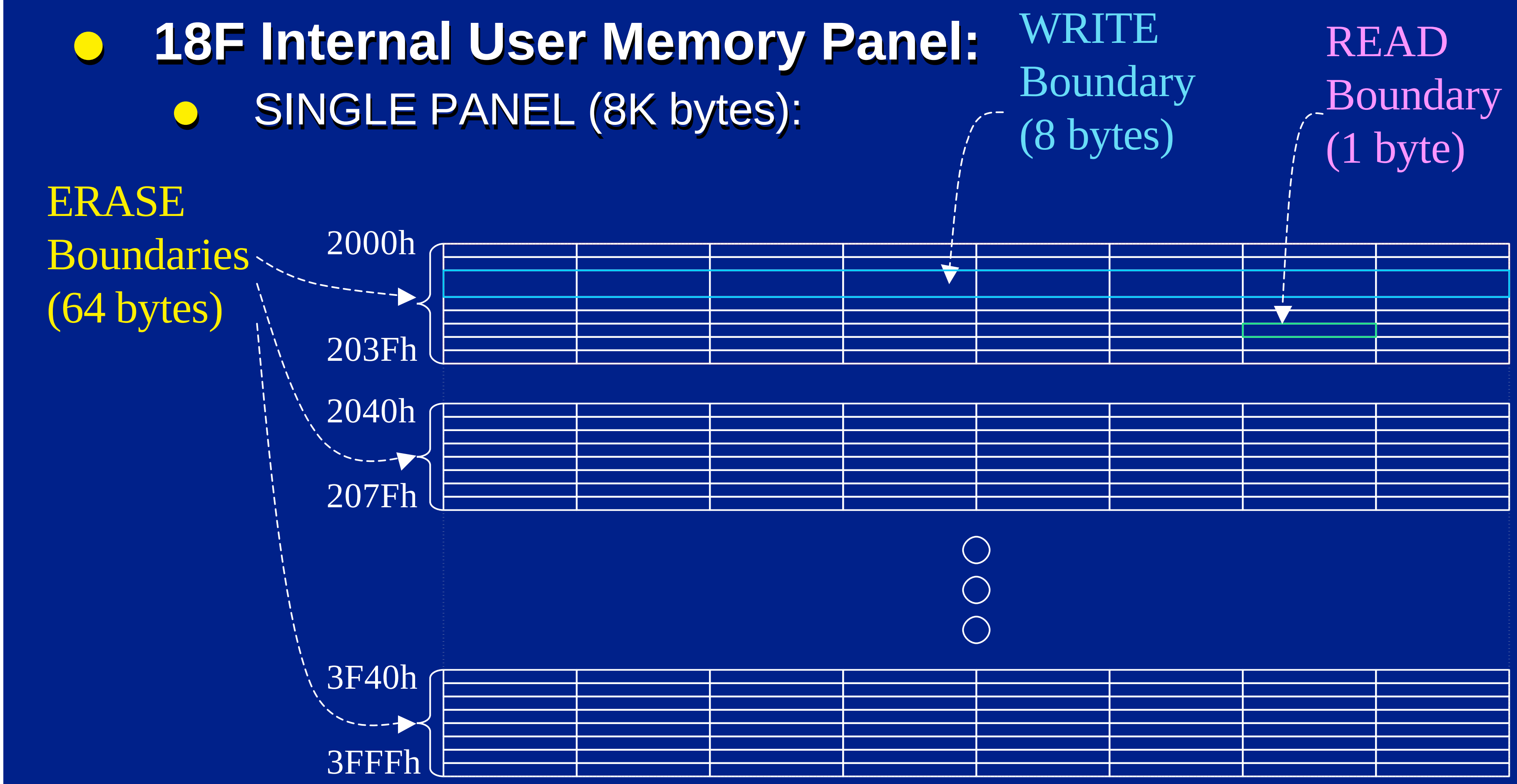
PIC18F ARCHITECTURE

- **18F Internal User Memory is separated by:**
  - PANELS:
    - Define internal cell grouping boundaries
    - Always 8 Kbytes (4 Kwords)
  - BLOCKS:
    - Define Code Protect boundaries
    - Minimum 512 bytes
    - Could be 16 Kbytes (18F8720)



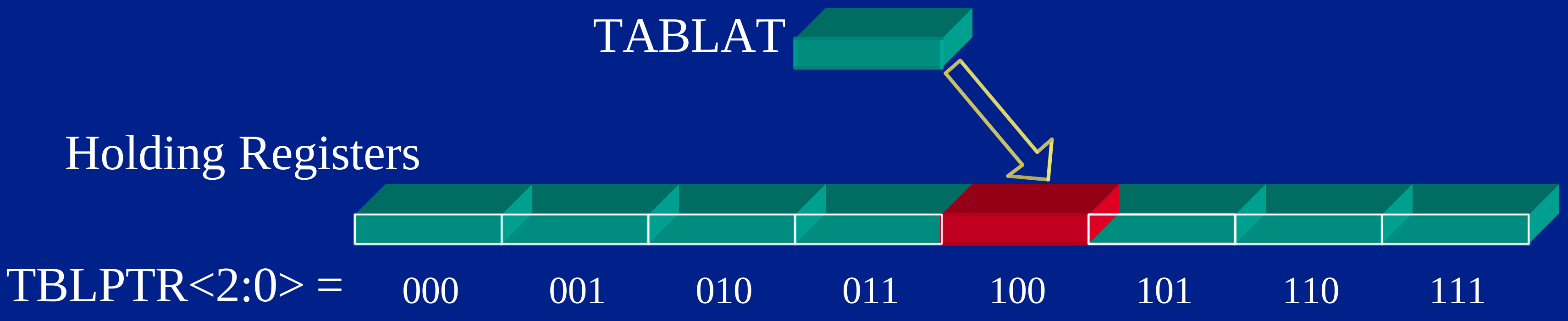
PIC18F ARCHITECTURE

- **18F Internal User Memory Panel:**
  - SINGLE PANEL (8K bytes):



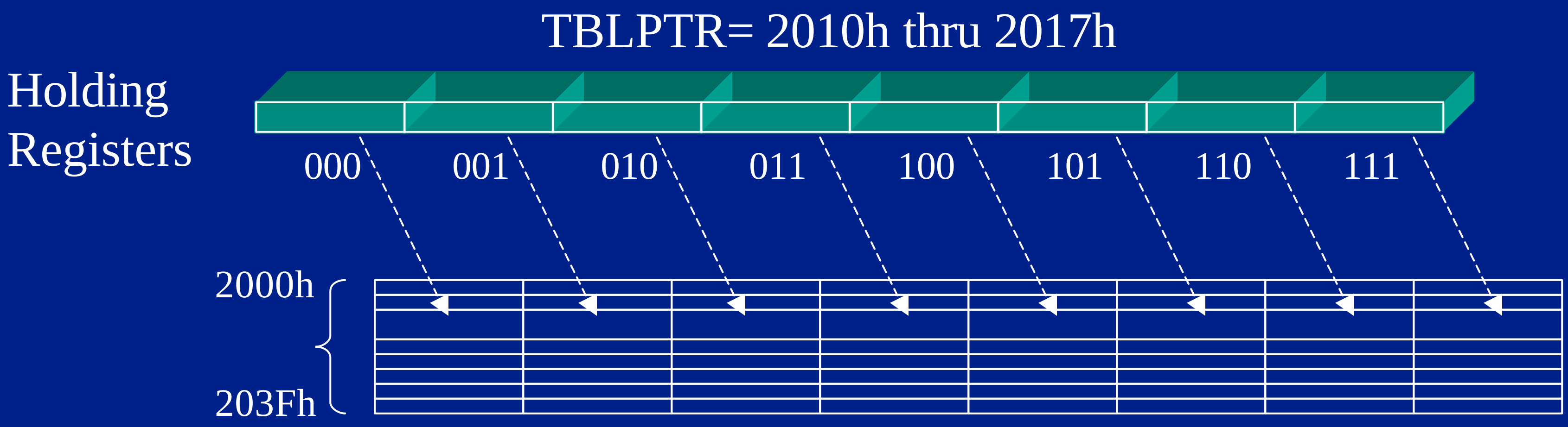
PIC18F ARCHITECTURE

- **18F Holding Registers:**
  - USER - 8 bytes for *entire code memory (single-panel programming)*
  - ICSP programming - 8 bytes for *each panel (multi-panel use)*
  - Loaded by TBLWT* instruction.
    - TBLWT* instruction moves contents of TABLAT to a holding register. The last three bits of TBLPTR determine which holding register. TBLPTR<20:3> are don't cares.



PIC18F ARCHITECTURE

- **18F Holding Registers (cont):**
  - Writes to Program Memory (details later)
    - TBLPTR <20:3> determines which 8 bytes of internal user memory Holding Registers will write to
    - In example, TBLPTR could be in range of 2010h - 2017h, and the holding registers will write same 8 bytes.



PIC18F EECON1

EECON1 REGISTER

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	-	FREE	WRERR	WREN	WR	RD
bit7	6	5	4	3	2	1	bit0

bit 7: **EEPGD:** FLASH Program or Data EEPROM Memory Select Bit

1 = Access Program Flash memory

0 = Access Data EEPROM memory

bit 6: **CFGS:** FLASH Program/Data EE or Configuration Select bit

1 = Access Configuration registers

0 = Access Program Flash or Data EEPROM memory

bit 5: **Unimplemented:** Read as ‘0’

bit 4: **FREE:** FLASH Row Erase Enable bit

1 = Erase the program memory row addressed by TBLPTR on next WR command (cleared on erase completion)

0 = Perform write only

PIC18F EECON1

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	-	FREE	WRERR	WREN	WR	RD
bit7	6	5	4	3	2	1	bit0

bit 3: **WRERR**: FLASH and EEPROM Error Flag Bit

1 = A write operation is prematurely terminated (RESET)

0 = The write operation completed

Note: When WRERR occurs, EEPGD and CFGS are not cleared.

bit 2: **WREN**: FLASH and EEPROM Write Enable Bit

1 = Allows write cycles

0 = Inhibits erases or writes to FLASH and EEPROM

bit 1: **WR**: Write Control Bit

1 = Initiates FLASH erase or write or EEPROM erase/write

0 = The write or erase operation is complete

bit 0: **RD**: Read Control Bit

1 = Initiates an EEPROM read

0 = Does not initiate an EEPROM read

PIC18F REQUIRED SEQUENCE

- WRITE and ERASE of internal user memory require six instructions as shown below:
  - Makes accidental writes and erases highly improbable.
  - First instruction following the WR bit set must be NOP. This instruction was pre-fetched and must be discarded.

```
movlw        55h
movwf        EECON2
movlw        AAh
movwf        EECON2
bsf          EECON1,WR
nop
```

PIC18F READ

- **READs performed on bytes**
- **Can READ entire user Program Memory of up to 2M plus:**
  - User ID locations 200000h-200007h
  - CONFIG registers 300000h-30000Dh
  - Device ID registers 3FFFFEh,3FFFFFh
- **To READ Program Memory:**
  - Load TBLPTRU,TBLPTRH,TBLPTRL
  - Execute one of the TBLRDs
    - TBLRD*, TBLRD*+, TBLRD+*, TBLRD*-
    - result in TABLAT *next instruction cycle*

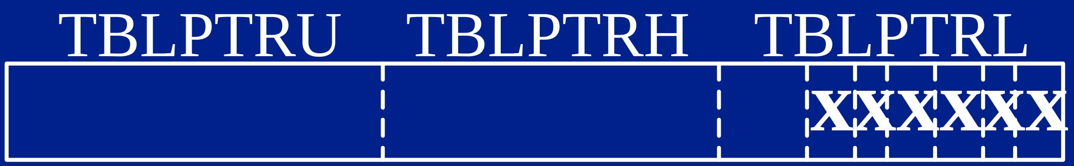
PIC18F READ

- READ Code example:

```
; Load Table Pointer
    movlw   UPPER(TBL_ADDR)
    movwf   TBLPTRU
    movlw   HIGH(TBL_ADDR)
    movwf   TBLPTRH
    movlw   LOW(TBL_ADDR)
    mowvf   TBLPTRL
    tblrd*
    movff   TABLAT,INDF0
```

PIC18F ERASE

- **ERASING User memory:**
  - Performed on 64 bytes (32 words)
  - Load TBLPTRU,TBLPTRH,TBLPTRL
  - Configure EECON1
  - Disable interrupts
  - Perform programming sequence
  - Start erase (Set WR bit)
    - *Internally timed 2 mS (typical)*
    - PROCESSOR ‘HALTS’, NO CODE EXECUTION
    - TBLPTR 6 LSBs are don’t cares



TBLPTRU TBLPTRH TBLPTRL

XXXXXX

  - Re-enable interrupts

PIC18F ERASE

- ERASE User Memory Code Example:

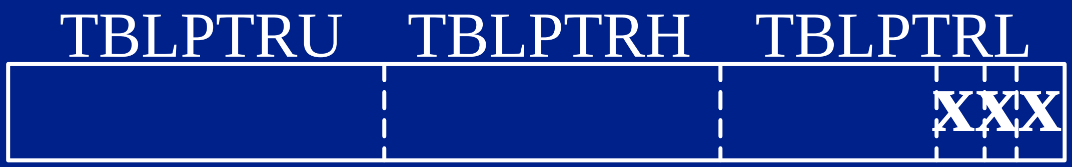
```
; Load Table Pointer
    bsf         EECON1,EEPGD
    bcf         EECON1,CFGS
    bsf         EECON1,WREN
    bsf         EECON1,FREE
    bcf         INTCON,GIE
    movlw       55h
    movwf       EECON2
    movlw       AAh
    movwf       EECON2
    bsf         EECON1,WR
    nop
    bsf         INTCON,GIE
    bcf         EECON1,WREN
```

PIC18F WRITE

- **WRITEs to User memory:**
  - Performed on 8 bytes (4 words)
  - Load TBLPTRU,TBLPTRH,TBLPTRL
  - Load 8 bytes into write buffers by 8 table write instructions
    - TBLWT*,TBLWT*+,TBLWT*-,TBLWT+*
  - Configure EECON1
  - Disable interrupts
  - Perform programming sequence

PIC18F WRITE

- **WRITEs to User memory (cont):**
  - Start write (set WR bit)
    - *Internally timed 2 mS*
    - PROCESSOR 'HALTS', NO CODE EXECUTION
    - TBLPTR 3 LSBs are don't cares



  - Re-enable interrupts

PIC18F WRITE

- What’s wrong with this?

```
; GIVEN:
; FSR0   ->   points to first of 8 bytes of a buffer
;             that will be written
; TBLPTR ->   points to first byte of 8 byte block in
;             internal user memory
; COUNTER =   8
; WRITEIT =   Correct programming macro

WRITE_TO_HREGS
      movff  POSTINC0,TABLAT          ; load Holding Regs
      TBLWT*+                         ;
      decfsz COUNTER                  ;
      bra    WRITE_TO_HREGS           ;
      WRITEIT                         ; Write Holding Regs
                                      ;  to user memory
```

- After the last TBLWT*+, TBLPTR would be pointing to the next 8 bytes block !

MICROCHIP
MASTER'S

Appendix D:

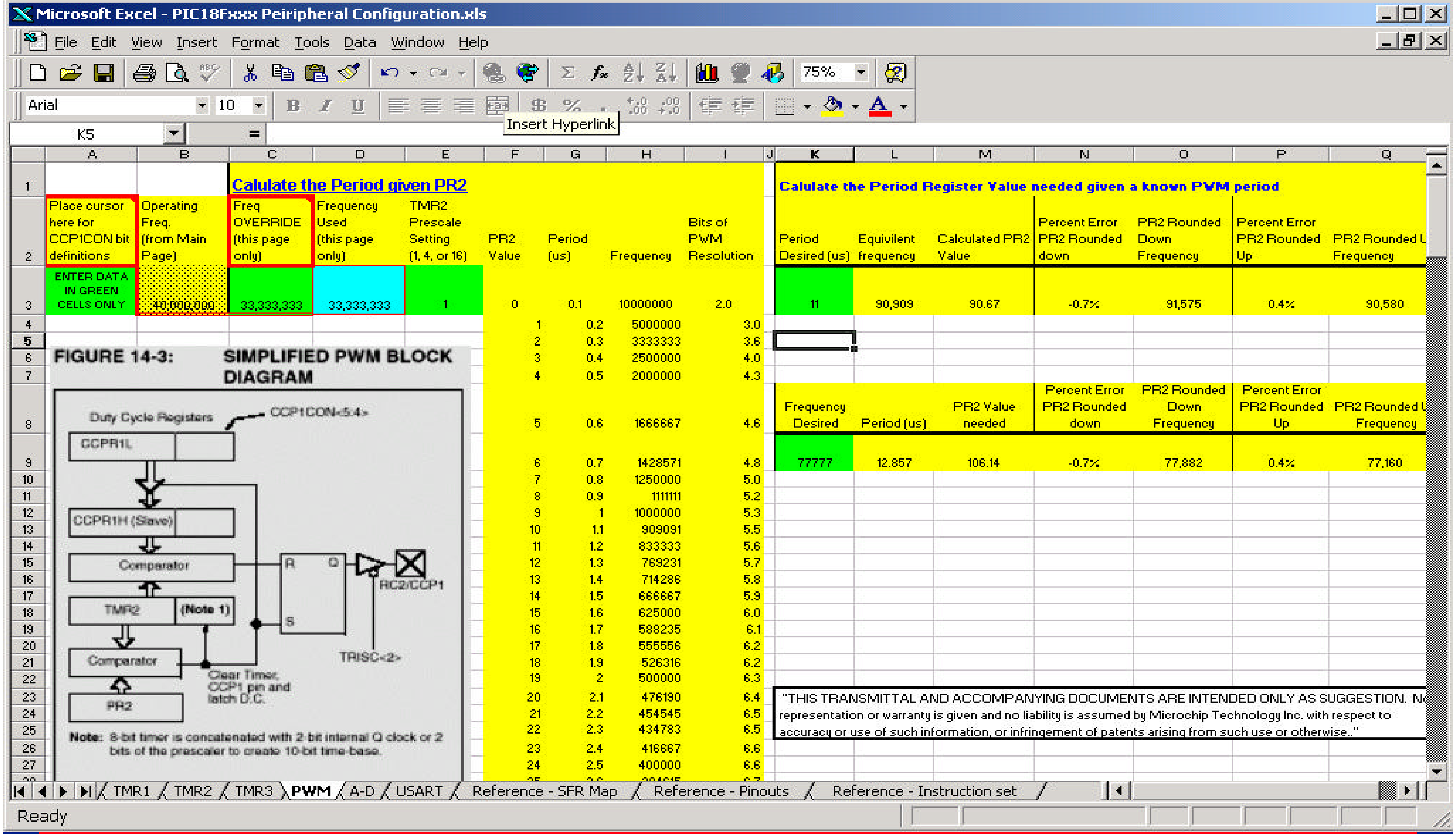
PIC18FXXXX Peripheral Configuration Spreadsheet

Spreadsheet Basics

PIC18Fxxx Peripheral Configuration.xls

- Spreadsheet based on Microsoft Excel
- Calculates period, baud rate, operating frequency for the following peripherals:
  - TMR0,TMR1,TMR2 and TMR3 period
  - PWM / CCP0 through PWM / CCP4 frequency
  - A/D conversion period
  - UART Baud Rate
- Contains reference map for Special Function Registers, Pinouts and Instruction Set

PWM Configuration Example



Calulate the Period given PR2

Place cursor here for CCP1CON bit definitions	Operating Freq. (from Main Page)	Freq OVERRIDE (this page only)	Frequency Used (this page only)	TMR2 Prescale Setting (1, 4, or 16)	PR2 Value	Period (us)	Frequency	Bits of PWM Resolution
ENTER DATA IN GREEN CELLS ONLY	40,000,000	33,333,333	33,333,333	1	0	0.1	10000000	2.0
					1	0.2	5000000	3.0
					2	0.3	3333333	3.6
					3	0.4	2500000	4.0
					4	0.5	2000000	4.3
					5	0.6	1666667	4.6
					6	0.7	1428571	4.8
					7	0.8	1250000	5.0
					8	0.9	1111111	5.2
					9	1	1000000	5.3
					10	1.1	909091	5.5
					11	1.2	833333	5.6
					12	1.3	769231	5.7
					13	1.4	714286	5.8
					14	1.5	666667	5.9
					15	1.6	625000	6.0
					16	1.7	588235	6.1
					17	1.8	555556	6.2
					18	1.9	526316	6.2
					19	2	500000	6.3
					20	2.1	476190	6.4
					21	2.2	454545	6.5
					22	2.3	434783	6.5
					23	2.4	416667	6.6
					24	2.5	400000	6.6

FIGURE 14-3: SIMPLIFIED PWM BLOCK DIAGRAM

Note: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

Calulate the Period Register Value needed given a known PWM period

Period Desired (us)	Equivilent frequency	Calculated PR2 Value	Percent Error PR2 Rounded down	PR2 Rounded Down Frequency	Percent Error PR2 Rounded Up	PR2 Rounded U Frequency
11	90,909	90.67	-0.7%	91,575	0.4%	90,580

Frequency Desired	Period (us)	PR2 Value needed	Percent Error PR2 Rounded down	PR2 Rounded Down Frequency	Percent Error PR2 Rounded Up	PR2 Rounded U Frequency
77777	12.857	106.14	-0.7%	77,882	0.4%	77,160

"THIS TRANSMITTAL AND ACCOMPANYING DOCUMENTS ARE INTENDED ONLY AS SUGGESTION. No representation or warranty is given and no liability is assumed by Microchip Technology Inc. with respect to accuracy or use of such information, or infringement of patents arising from such use or otherwise.."