11010 SBC

16-bit Architecture, Standard Peripherals and Programming Using C30

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Objectives

- **When you walk out of this class, you will understand the 16-Bit MCU:**
  - Architecture
  - Interrupts
  - Basic Peripherals
  - Peripheral Configuration
  - Programming in C using MPLAB® C30

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Agenda

- **16-Bit Architecture Basics & MPLAB® C30**
  - MCU Data Path & Registers
  - Program & Data Memory
  - MPLAB C30 C-Language Extensions
    - **LAB 1: My First C30 Project**
  - Data Access From Program Memory
    - **LAB 2: PSV Initialization and its usage**
  - Stack and Frame pointer
  - Interrupt and Trap Handling
    - **LAB 3: Interrupt Service Routine and Interrupt Priority**

Agenda (*Continued..*)

- **16-Bit Peripherals**
  - I/O Ports
  - ADC
    - **LAB 4: ADC Configuration & usage**
  - Comparators & Voltage Reference generator
  - Timers
    - **LAB 5: Configuration of 32-bit Timers**
  - Capture
  - Compare & PWM
  - UART
    - **LAB 6: UART configuration and usage of FIFO**
  - I^2C™
  - SPI

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

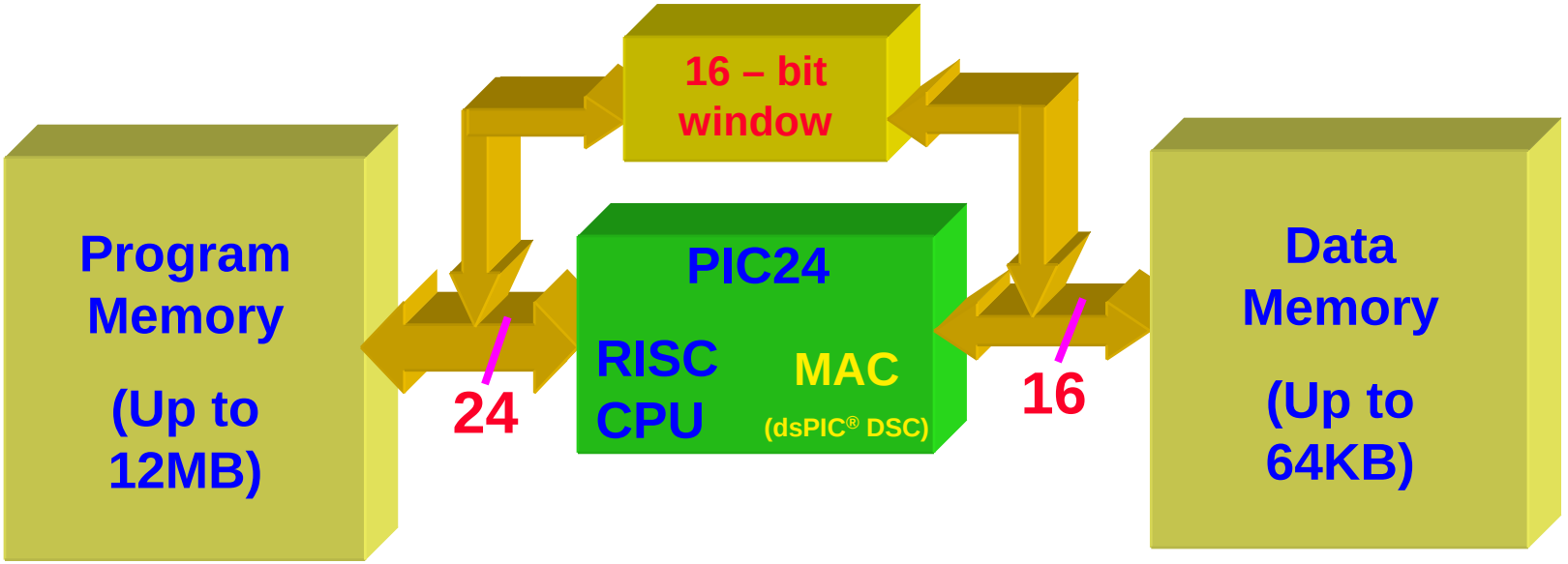
Agenda (*Continued..*)

- **Special features**
  - Oscillators and Power Saving modes
  - Resets
  - WDT
- **Summary**
- **References**

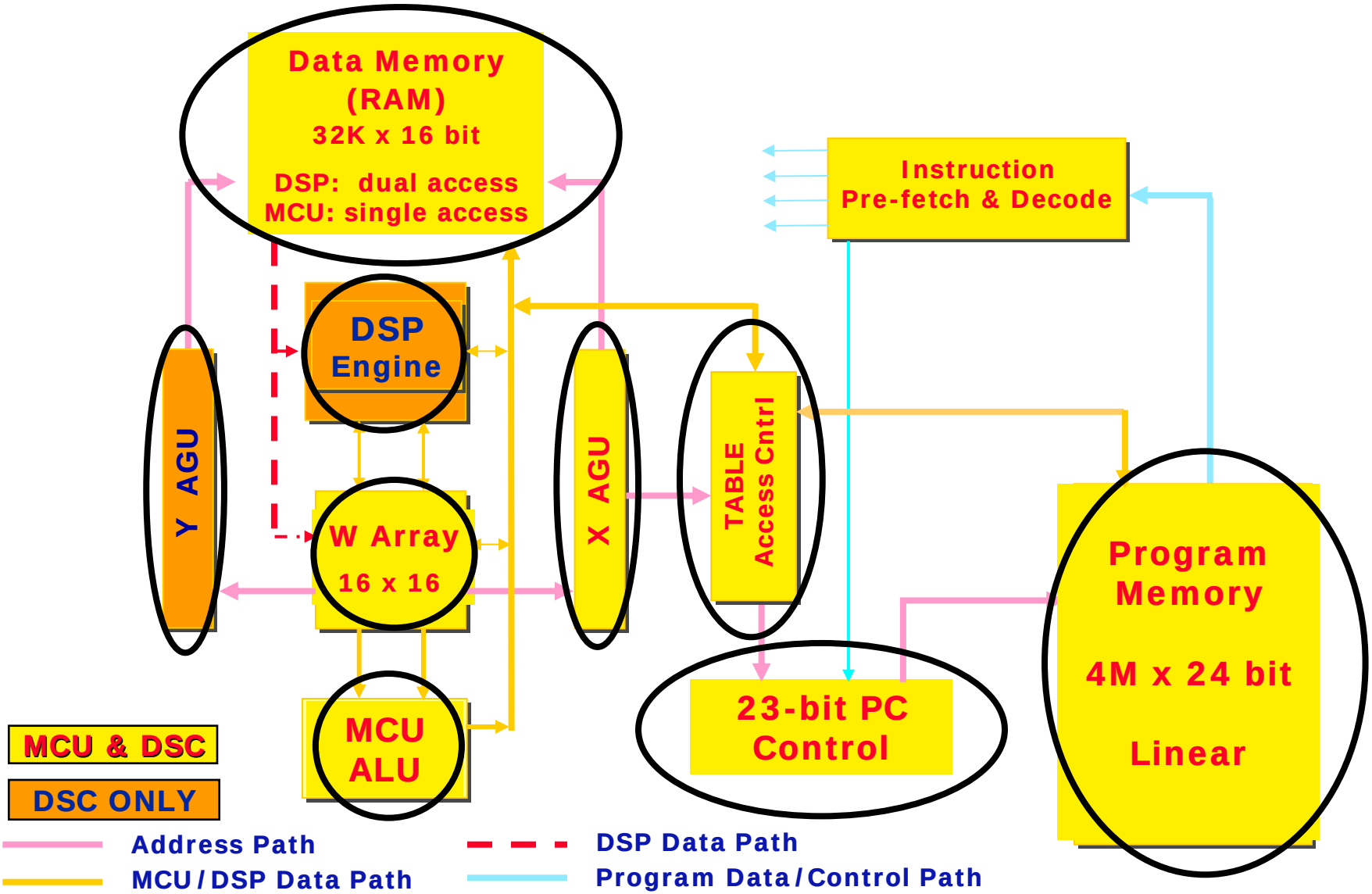
16-Bit Architecture Basics & MPLAB® C30

Harvard Architecture

- 16-bit microcontroller
- 24-bit Instruction width
- Data Transfer Mechanism between PM and DM

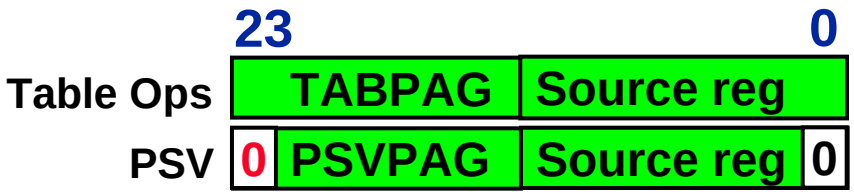


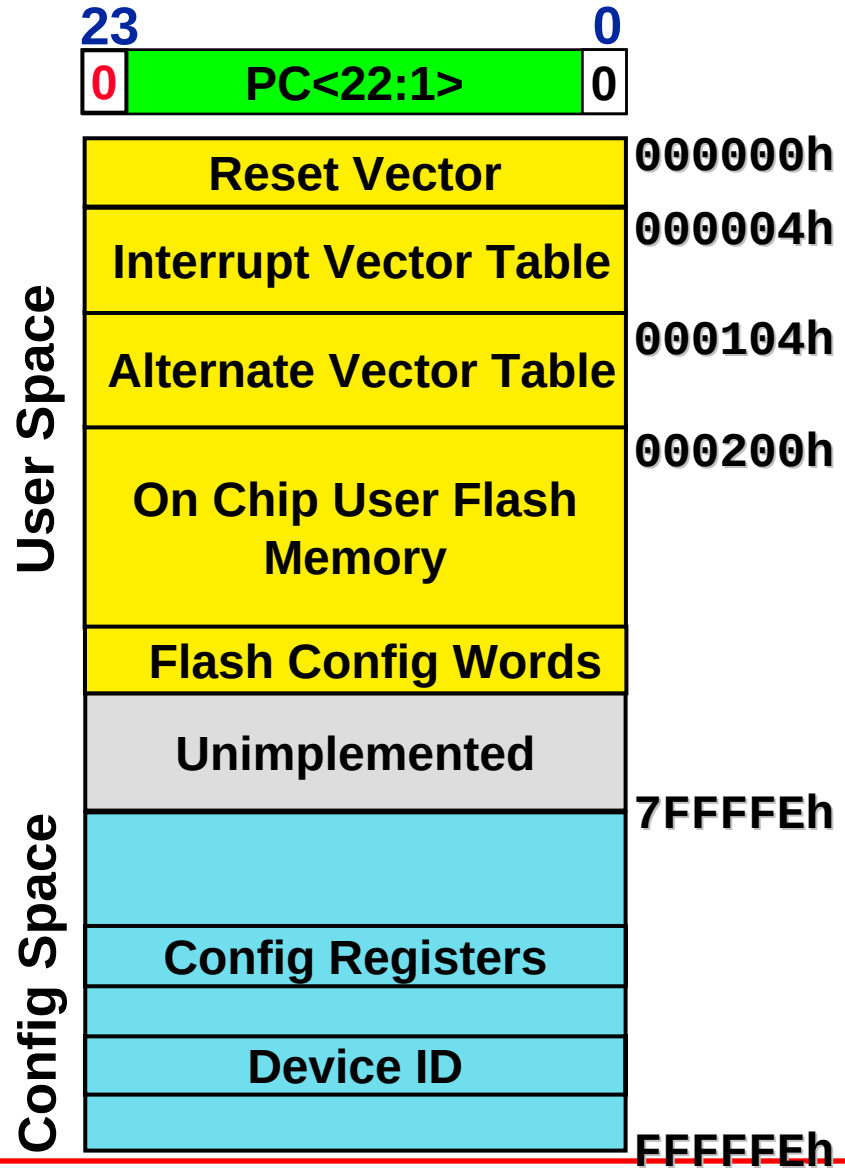
Architecture Block Diagram



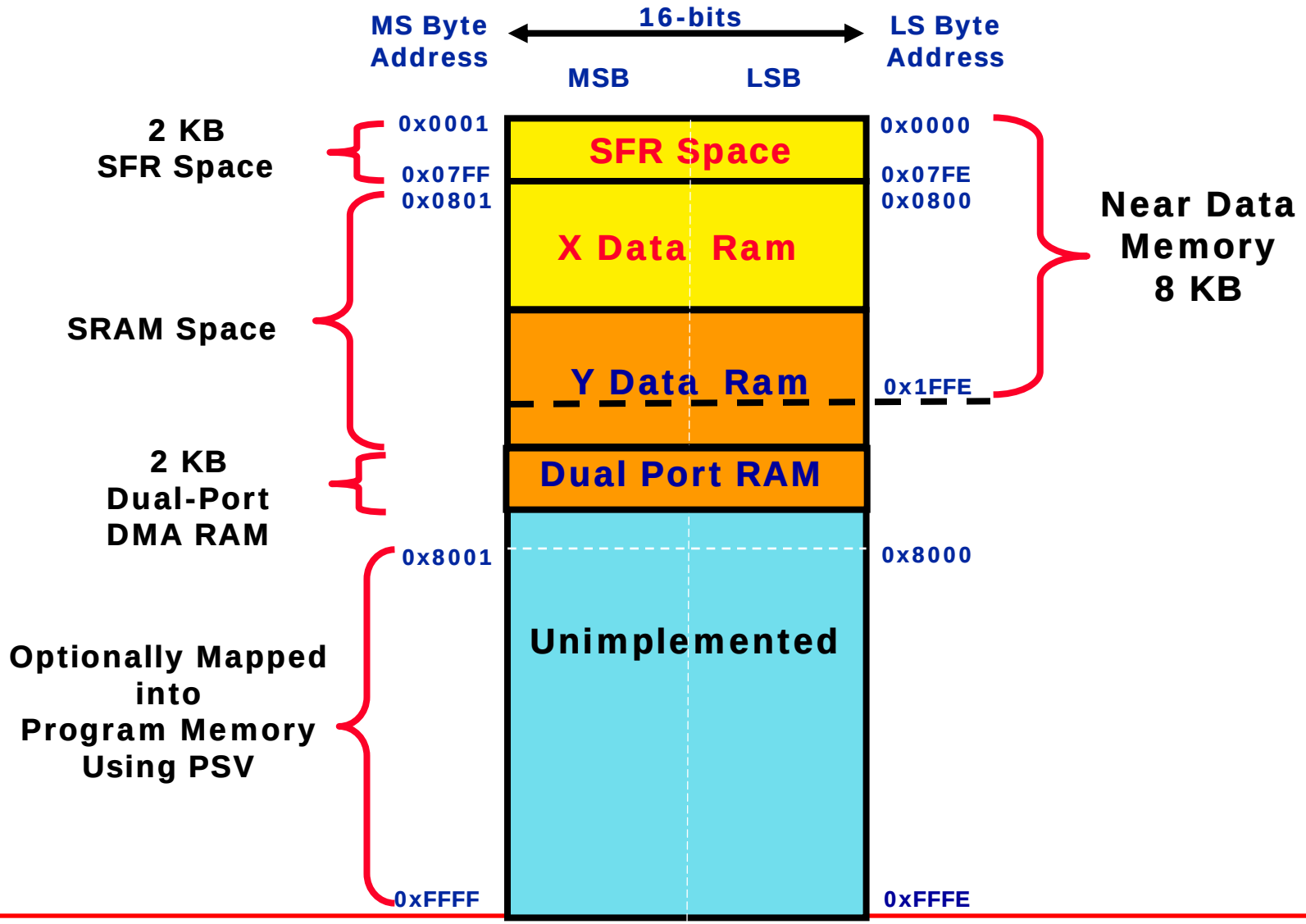
Program Memory Organization

- Maximum 12MB
  - 4MB x 24-bit
  - 23-bit PC (PCH & PCL)
- PC increments in words (LSB always '0')
- Reset Vector at 0
- Interrupt Vector Table from 4h to FEh
- User Code space from 200h to 7FFFFEh (what ever is implemented)



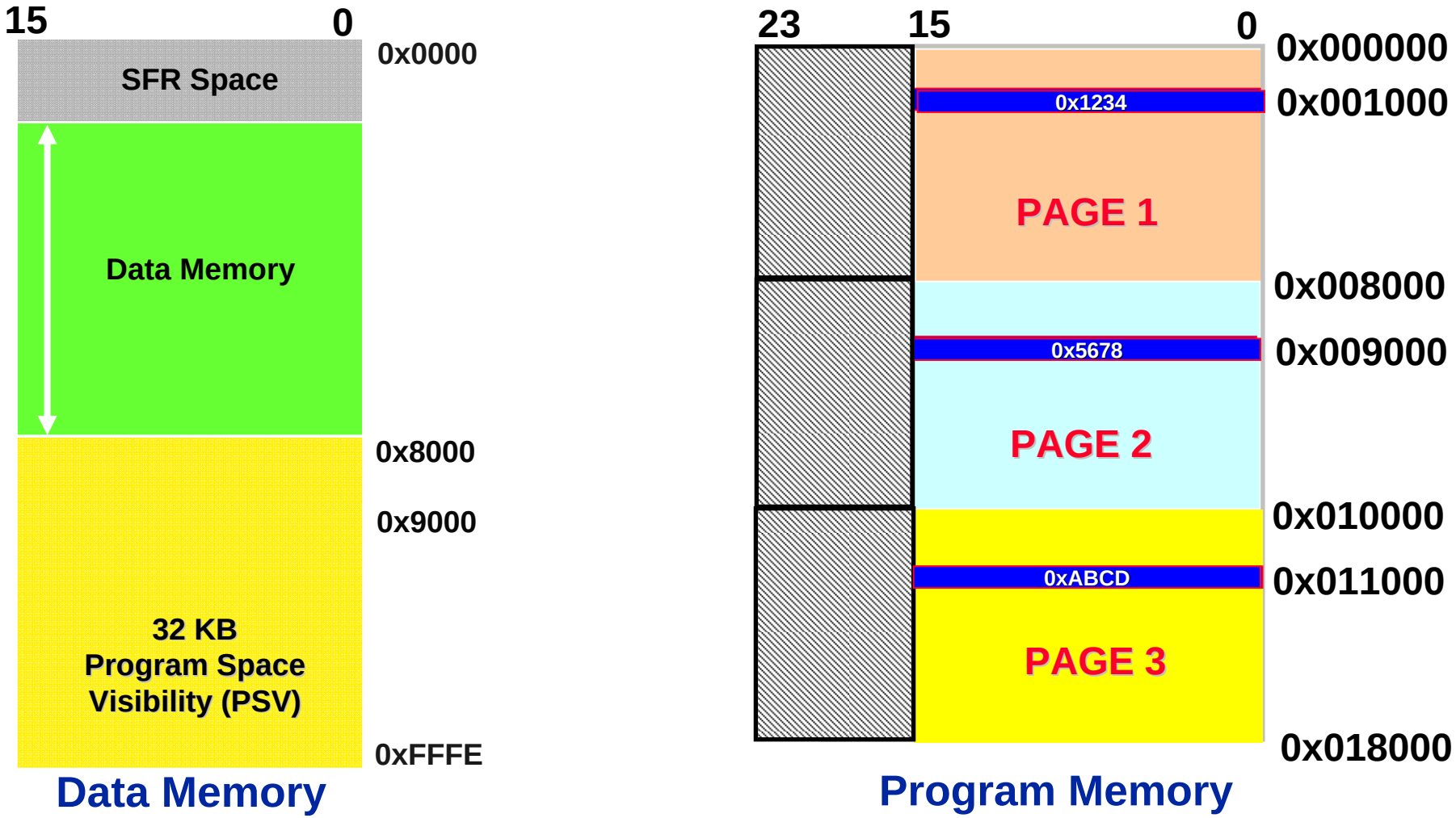


Data Memory Organization



PSV - Program Space Visibility From Data Space

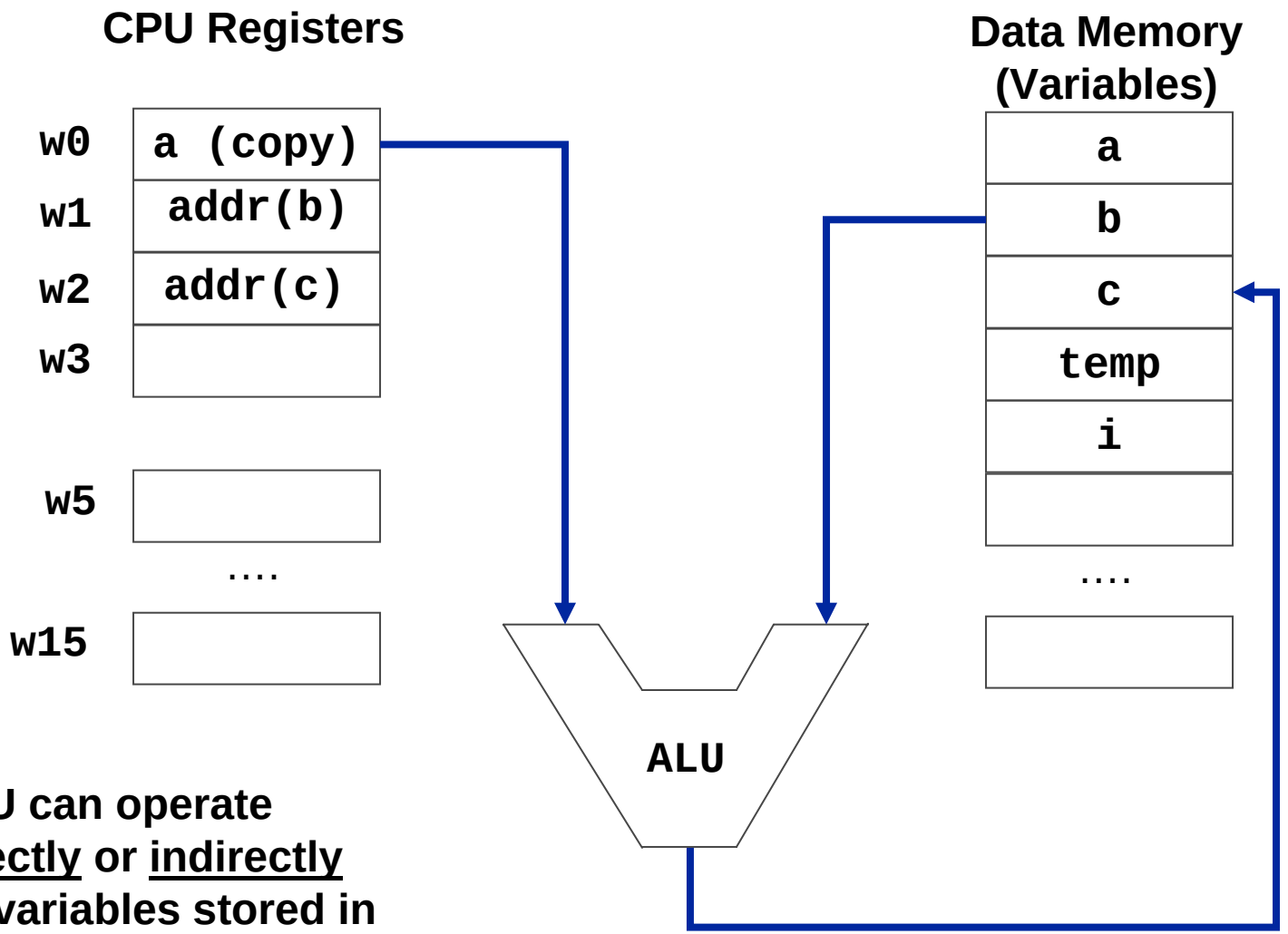
Ability to map 16KW (32KB) program memory segments into Data Memory



MCU ALU & Datapath
- Overview -

- **16-bit Family ALU is 16-bits Wide**
  - Operates on both 8 & 16-bit operands
- **Operations: 2's complement Add, Subtract, Single/Multi-bit shifts, Logical**
- **Operations can affect status flags:**
  - Carry/Borrow (C), Zero (Z), Negative (N), Overflow (OV), Half-Carry/Borrow (DC)

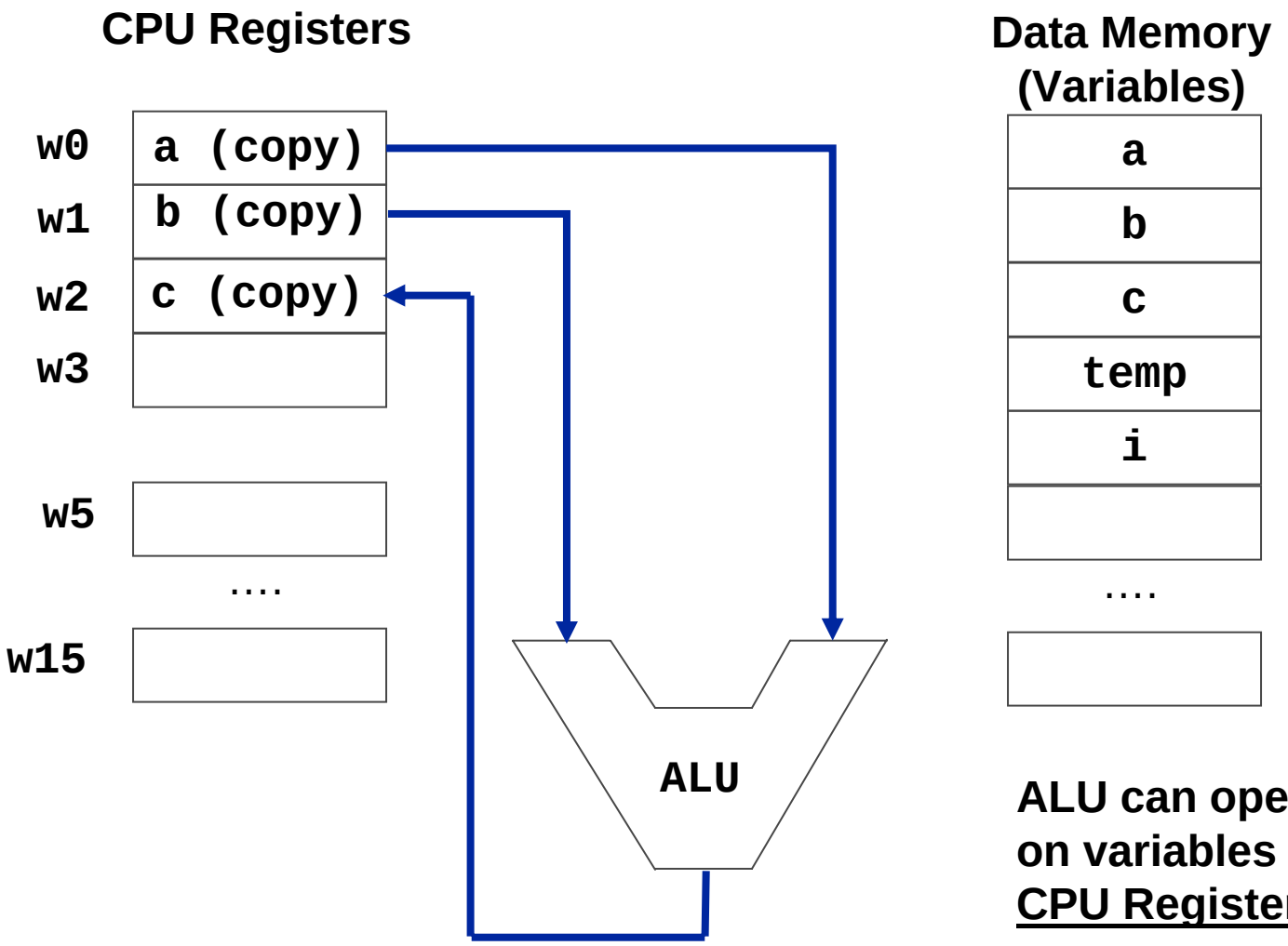
Datapath Example
(c = a + b;)



ALU can operate directly or indirectly on variables stored in Data Memory

```
add w0, [w1], [w2]
```

Datapath Example #2
(c = a + b;)



ALU can operate directly on variables stored in CPU Registers

```
add w0, w1, w2
```

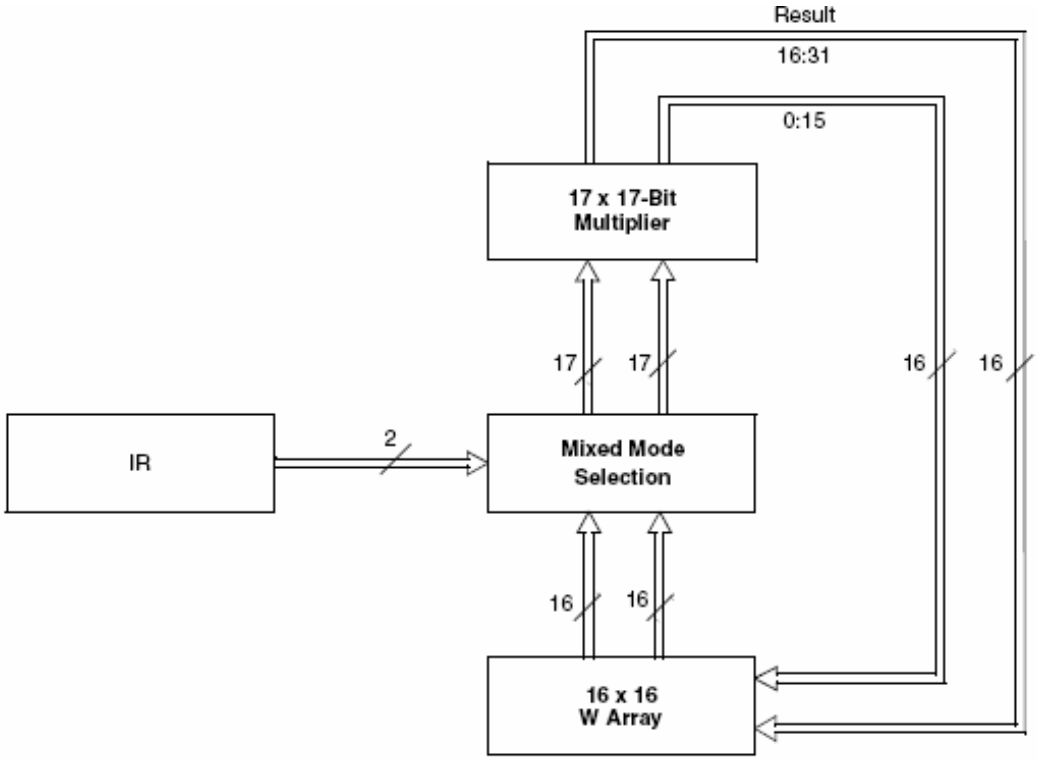
University of Microchip UofM MASTERs 2007

Multiply Support

- **Single-cycle, 17x17 multiplier supporting the following modes:**
  - 16-bit Signed x 16-bit Signed
  - 16-bit Unsigned x 16-bit Unsigned
  - 16-bit Unsigned x 16-bit Signed
  - 16-bit Signed x 5-bit (literal) Unsigned
  - 16-bit Unsigned x 5-bit (literal) Unsigned
  - 8-bit Unsigned x 8-bit Unsigned

Multiplier Block Diagram

- All operands zero/sign-extended to 17 bits
- Unsigned 16-bit multiplies produce a 32-bit, unsigned integer
- Signed 16-bit multiplies produce a 32-bit, 2's complement signed integer
- Mixed mode 16-bit multiplies produce a 32-bit, 2's complement signed integer



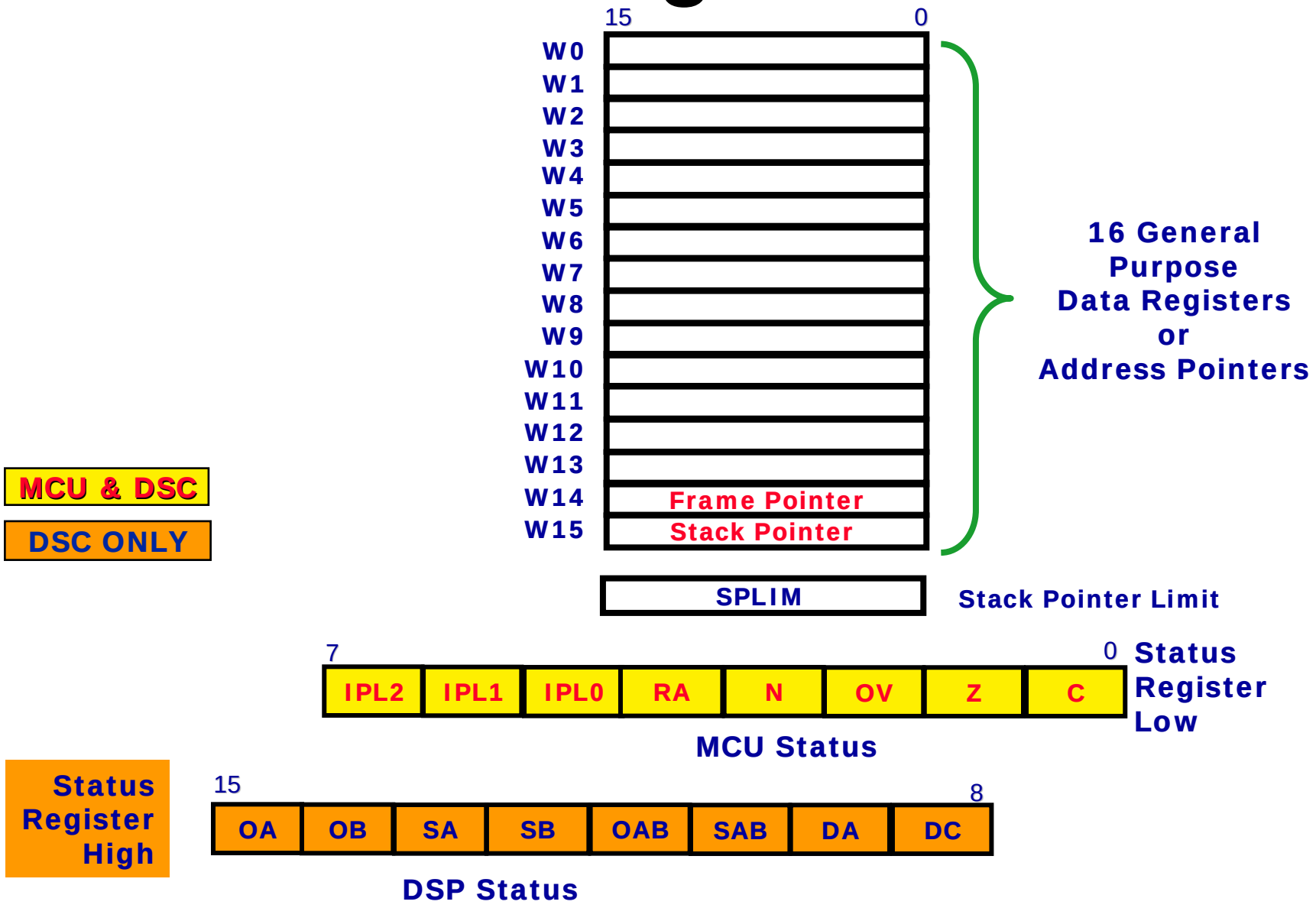
Divide Support

- 16-bit family features a 32/16 and 16/16 signed/unsigned integer divider
- Implemented as an *interruptible*, single instruction iterative loop, i.e.

```
repeat #17   ;must be executed within repeat loop
div.sd       ;signed division (16/16)
```

- Total execution time (incl. `repeat`)
  - 19*Tcy

Register Set



Register Set (contd.)

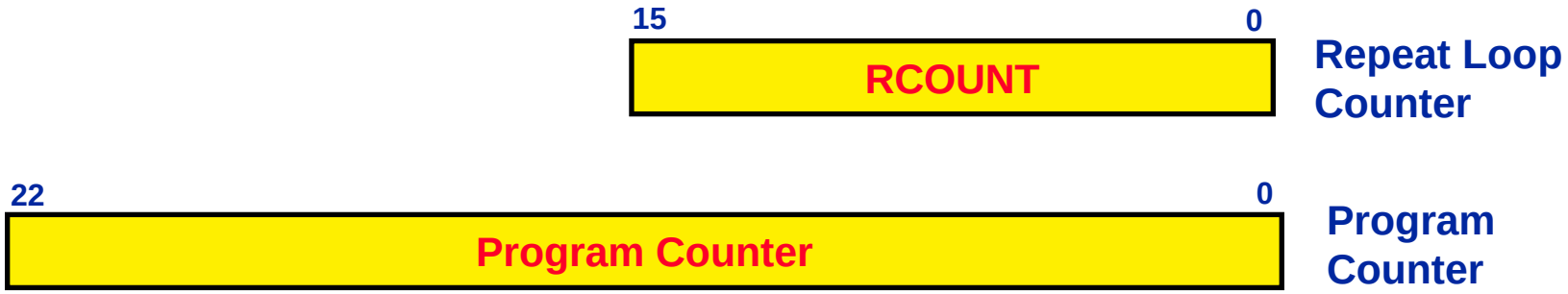
Core/DSP Control Register

Prg. Space Visibility Page Address

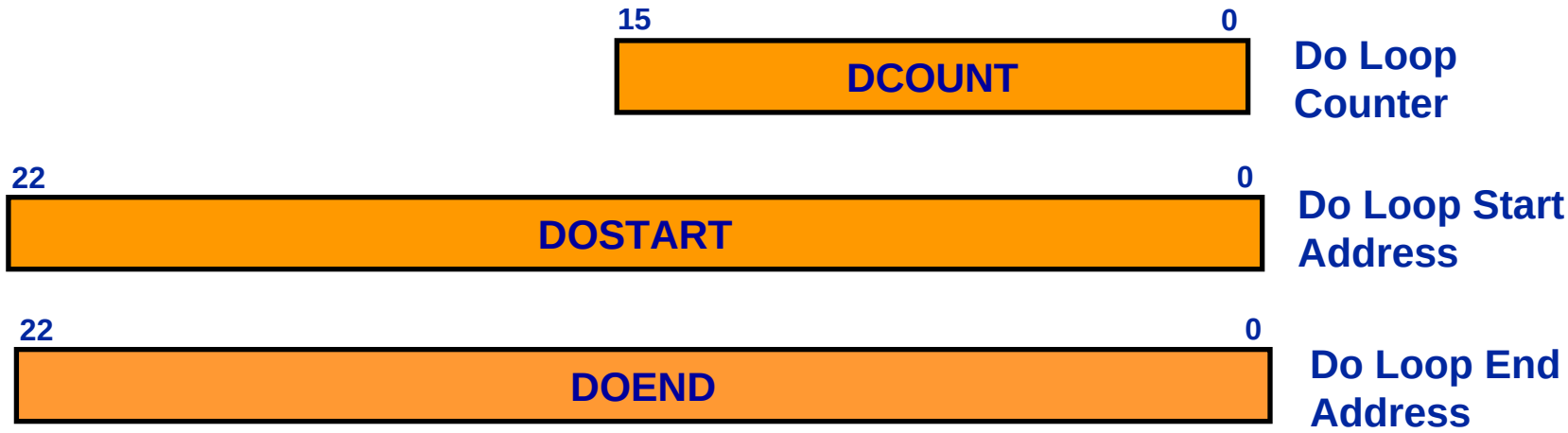
Data Table Page Address

Two DSP Accumulators

Register Set (contd.)

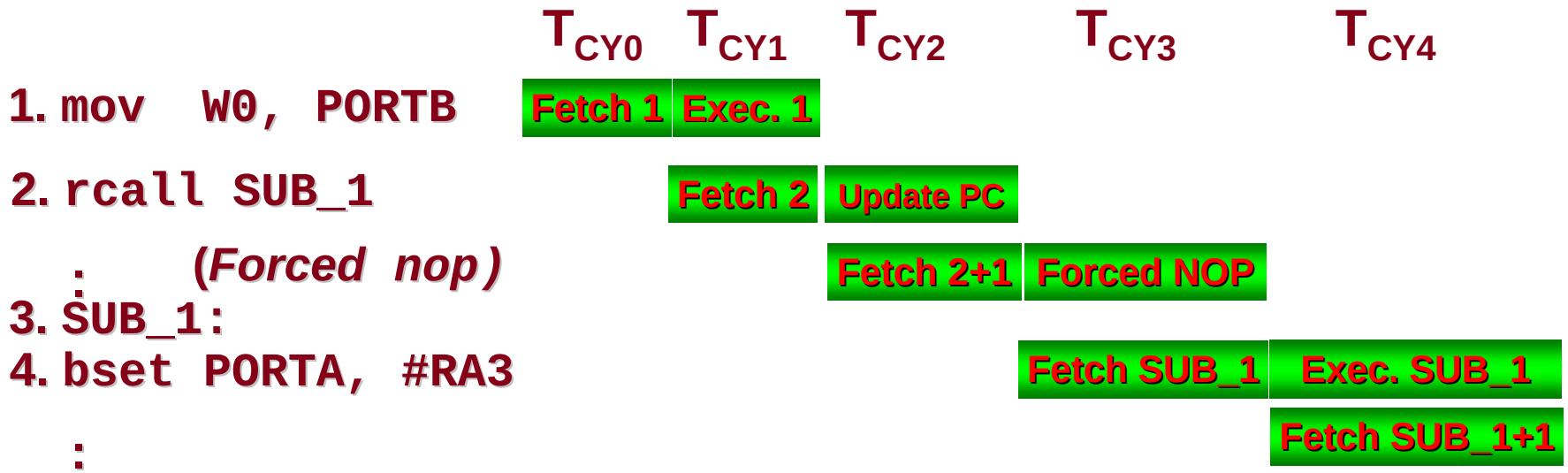


Do Loop Registers:



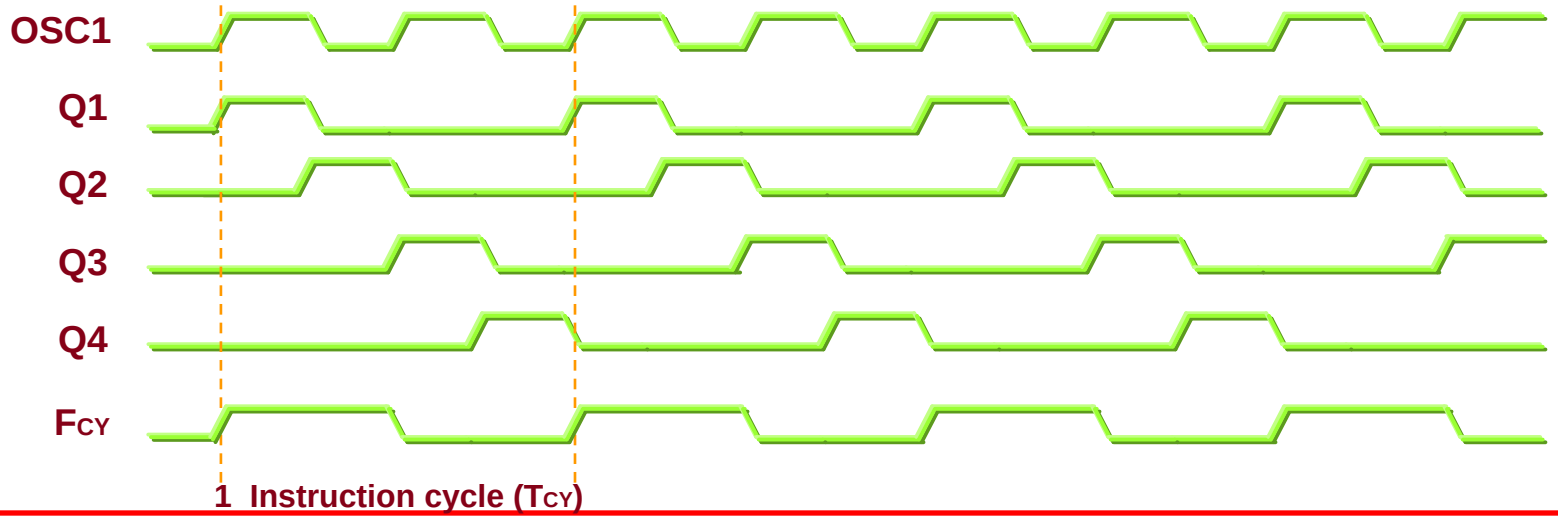
Instruction Pipeline

- Allows overlap of execution and fetch
- Makes single cycle execution
- Program branches (e.g. `goto`, `rcall`) take two cycles



Clocking Scheme

- **Instruction cycle = 2 Input Clocks**
- **PIC24F**
  - A 32Mhz clk = 16MIPS or Tcy = 62.5 nS
- **PIC24H/dsPIC33**
  - A 80Mhz clk = 40 MIPS or Tcy = 25 nS
- **dsPIC30F – Instruction cycle = 4 Input Clocks**
  - A 120Mhz clk = 30MIPS or Tcy = 33 nS



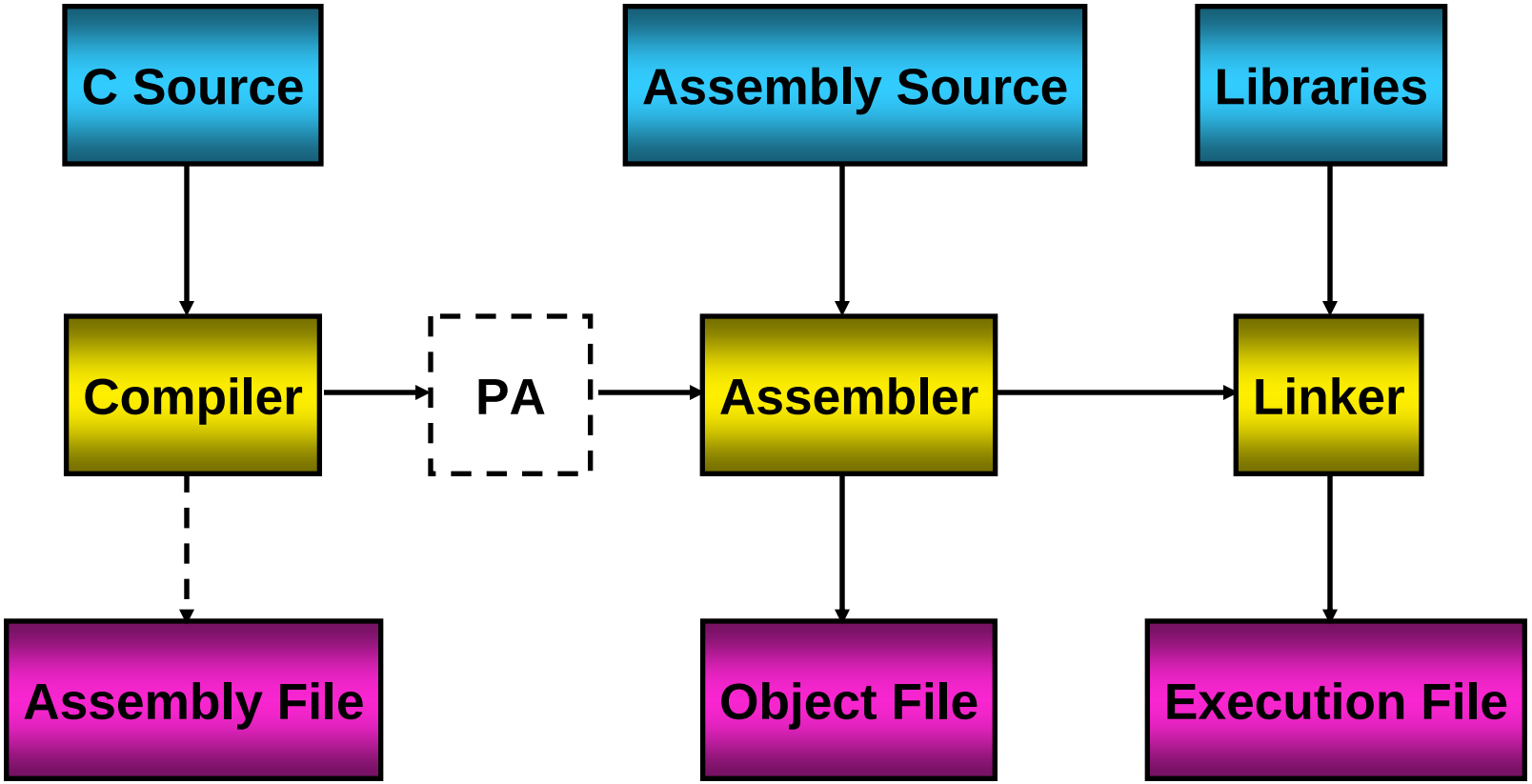
UNIVERSITY OF MICROCHIP
UofM
MASTERs 2007

MPLAB® C30 C-Language Extensions

MPLAB® C30 Compiler

- **C Compiler for PIC24 and dsPIC® DSC**
- **Targets ANSI: 1989 standard**
- **Built off GCC Compiler**
  - Standard C functions
- **Run time libraries**
- **One-time cost of $895US**
  - Student Version is free with Optimizations disabled after 60 days
  - www.microchip.com/c30

C30: Compilation Process



C30 C Language Extensions
- Agenda -

- **Program Memory Variables**
- **SFR Access**
- **Mixing C and Assembly**
- **Built-in Functions**
- **Specifying Attributes of Variables & Functions**
- **Interrupt Support**
- **Configuration Settings Support**

Variables in Program Memory

- **`const` - data type qualifier indicates that a variable is located in program memory space, and is to be accessed as an "Auto PSV" type (i.e. compiler-managed PSV)**
  - ex. `const char data[256];`
- **Can optionally place and access constants in data memory using the `constants-in-memory` memory model**
- **`const` data is limited to <32kB**
  - See PSV slides for more options (>32kB)

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

SFR Access

- **Special Function Registers can be accessed directly in C code:**
  - Word access: `PORTA = 0xFFEC;`
  - ***SFRNAME*`bits.`*BITNAME*** can be used to access bits and bit fields
    - **e.g.** `PORTAbits.RA1 = 1;` **or**
      `IPC0bits.INT0IP = 0;`
  - **_SFRNAME** and **_BITNAME** can also be used to reference an SFR or bit
    - **e.g.** `_PORTA` **or** `_RA1`
- **Must use standard header and linker script files**

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Mixing C and Assembly

- **Assembly files (.s) can be used in C30 projects and called from C**
  - Details available in C30 User's Guide
- **Inline assembly can be inserted into C source code**
  - Simple method:

```
asm("reset")  asm("clrwdt")
```

  - Complex method: a = b + c

```
asm volatile("add %1, %2, %0" : "=r"(a)
                              : "r"(b), "r"(c))
```

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

University of Microchip UofM MASTERs 2007

Built-in Functions

- **Enable use of special hardware features**
  - Multiply

```
__builtin_mulss(const signed int p0, const signed int p1)
```

  - Divide

```
__builtin_divsd(const long num, const int den)
```

  - Table pointers

```
__builtin_tblpage(const void *p)
__builtin_tbloffset(const void *p)
__builtin_tblwtl(unsigned int offset, unsigned int data)
__builtin_tblrdl(unsigned int offset)
```

  - PSV – retrieve and access PSV information

```
__builtin_psvpage(const void *p)
__builtin_psvoffset(const void *p)
```

University of Microchip UofM MASTERs 2007

Built-in Functions

- **More efficient code – Faster & Smaller**
  - Bit toggle

```
__builtin_btg(unsigned int *, unsigned int 0xn)
```

  - Locked register writes

```
__builtin_write_NVM(void);  //sets WR bit
__builtin_write_OSCCONL(unsigned char value)
__builtin_write_OSCCONH(unsigned char value)
```

  - All DSP Operations

```
__builtin_mac(…)
```

- **Other "built-ins"**
  - **`Nop(), Sleep() & Idle(), ClrWdt()`**

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Attributes of Variables

- **The keyword `__attribute__` allows you to specify special attributes of variables or structures including**
  - Placement of objects in certain locations in memory
  - Alignment of objects to memory boundaries

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Data Attributes: Examples

- **Place data at a specific address**

```
int MyVar __attribute__((address(0x860)))
int __attribute__((address(0x862))) MyVar2
```

  - Be careful about overlap and auto variables

- **Place into near memory (below 8kB)**

```
char __attribute__((near)) MyVar3, MyVar4
```

Data Attributes: Examples

- **Place in a specific section or space**

```
int foo __attribute__((section("ThePlace")))
int bar __attribute__((space(psv)))
```

  - New section will be created, if needed
  - Spaces: **`data, prog, psv, eedata`**

- **Align begin or end to a multi-byte boundary**

```
char __attribute__((aligned(32))) MyArray[18]
int __attribute__((reverse(64))) EndArray[25]
```

Attributes of Functions

- **The keyword `__attribute__` allows you to declare certain things about your function which help the compiler optimize function calls and check your code**
- **Examples include**
  - Absolute function placement in memory
  - Specification of ISR functions

Function Attributes: Examples

- **Placement of a function at a particular address**

```
void foo() __attribute__ (address(0x100))) {
...
}
```

- **Declaration of interrupt service routines…**

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Interrupt Support

- **Declaring interrupts with the compiler has two components:**
  - Declaring a function as an interrupt service routine (ISR)
  - Specifying additional attributes

Interrupt Support

- **Declare an interrupt function**

```
void __attribute__((interrupt))_ADCInterrupt(void)
```

- **Make function use shadow registers**

```
void
__attribute__((interrupt,shadow))_ADCInterrupt(void)
```

  - Saves W0-W3 and SR bits with `push.s` and `pop.s`

- **Optionally, save other variables on context save**

```
void
__attribute__((interrupt(save(var1,var2))))_ADCInterrupt(void)
```

C30 C Language Extensions

- Program Memory Variables
- SFR Access
- Mixing C and Assembly
- Built-in Functions
- Specifying Attributes of Variables & Functions
- Interrupt Support
- Configuration Settings Support

Configuration Settings

- **Device configuration can be set in MPLAB® IDE using "Configuration bits" window**
- **A better way is to use `_config` directives in C code**
  - **`_CONFIG1(<VALUE1> & <VALUE2>) ;`**
    `_CONFIG2(<VALUE>);`

`e.g. _CONFIG1(GCP_OFF & GWRP_OFF & FWDT_ON & ICS_PGx2)`

  - MCU header files contain valid config definitions and mnemonics

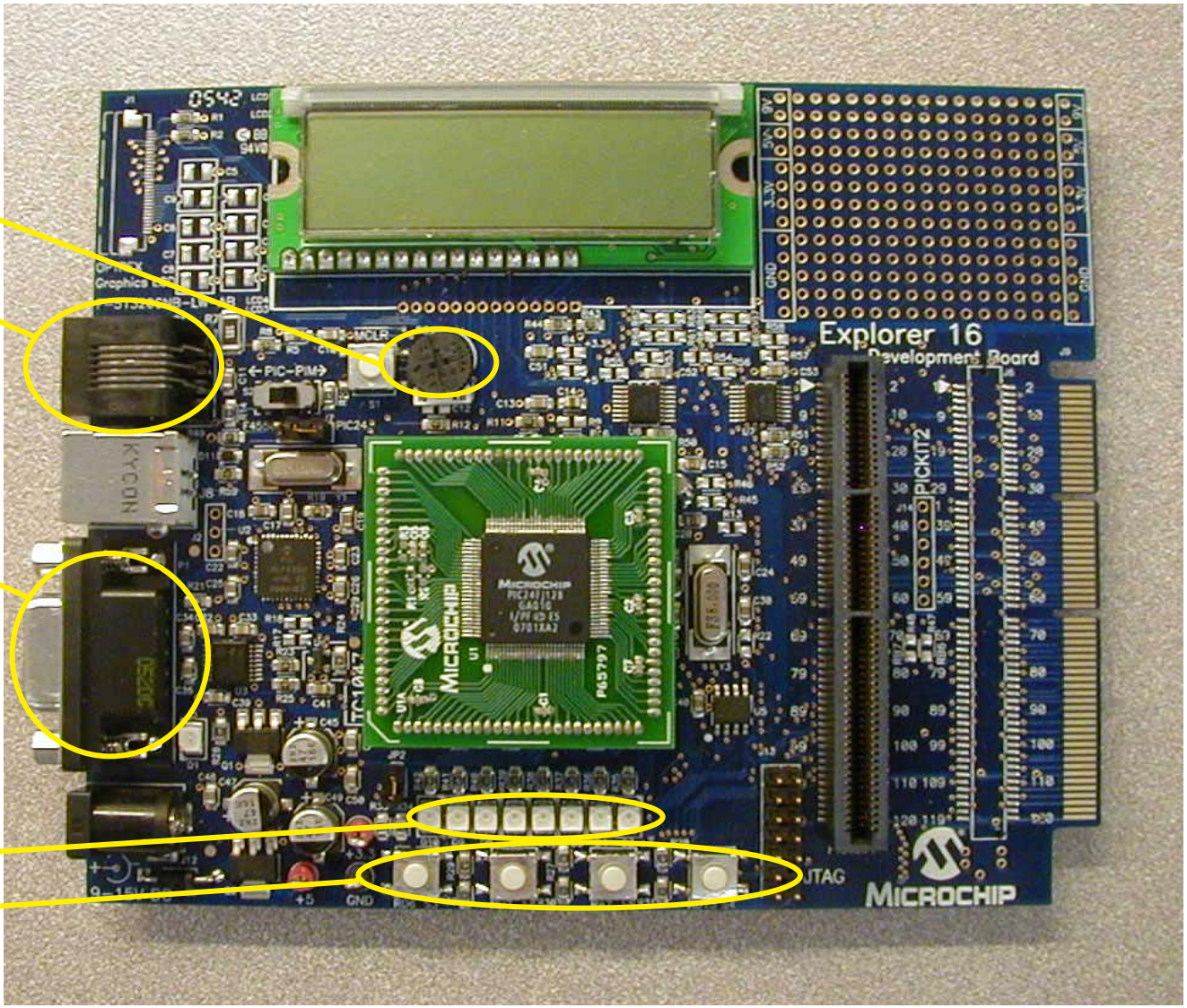
Lab 1:
My First C30 Project

University of Microchip UofM MASTERs 2007

Hands-on Exercises

- **All labs are in the following directory:**
  `C:\Masters\11010\Student\LabX`
  - where X is the number of the lab
- **Solution workspaces are supplied in the following directory:**
  `C:\Masters\11010\Student\LabX\Solution`
  - where X is the number of the lab
- **Directions for the labs are in the class handout which is also in the following directory:**
  `C:\Masters\11010\Student\Presentation & Handouts\11010_SBC_Handout.pdf`
- **Supplementary materials are in the following directory:**
  `C:\Masters\11010\Student\User's Guides and Data Sheets`

Development Tools
- Explorer 16 Board-



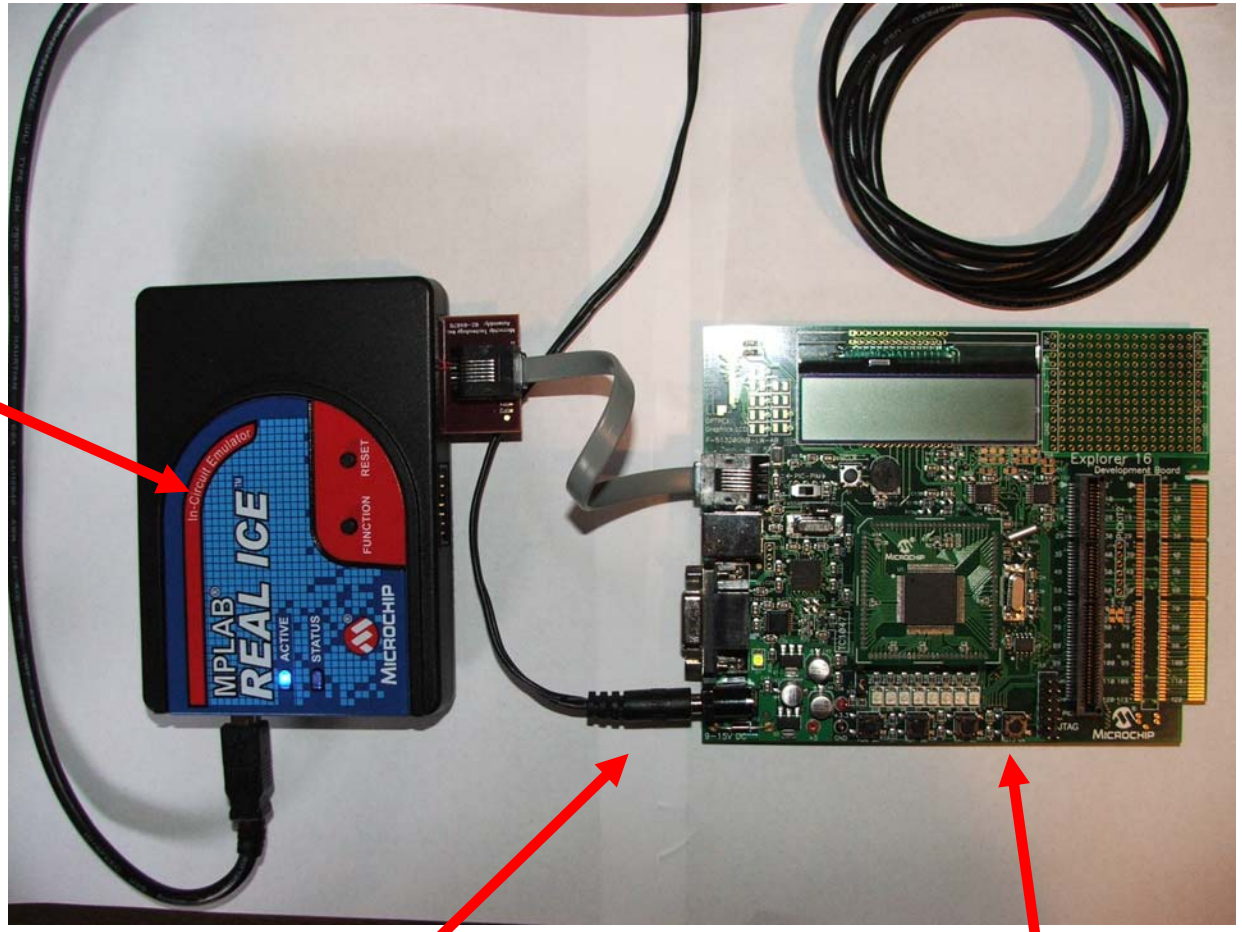
- Make sure to set the following:
  - S2→PIM
  - J7→PIC24
  - JP2→Shorted
  - PIM→PIC24F

Development Tools
- MPLAB® REAL ICE™ Emulator -

- Next-generation ICE platform
- Legacy (ICD2 RJ11) and high speed connectivity
- Full speed emulation
- High speed 480 Mb/S 2.0 USB workstation connection
- High speed target debug protocol
- Supports real-time watch, hardware stopwatch, and trace capture

Connecting the MPLAB® REAL ICE™ Debugger

MPLAB REAL ICE (w. "Standard" Probe)



9VDC Power Supply

Explorer 16 Demo Board

Lab 1: Working with C30 and MPLAB® IDE

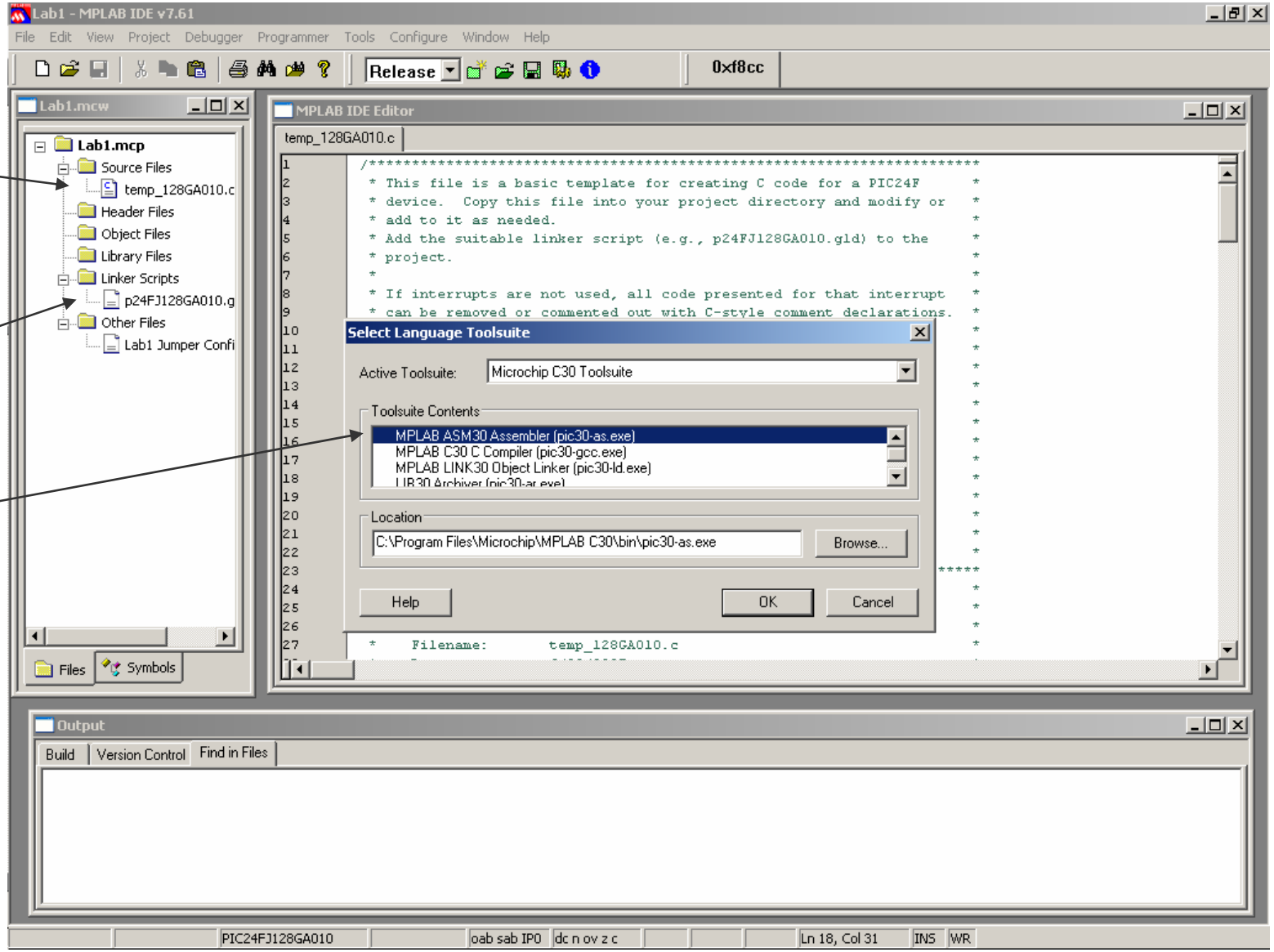
- **Goals:**
  - Learn the basics of C30 by working with a template project
- **To Do:**
  - Follow along with the presenter
- **Expected Result:**
  - Successfully build the project and program the device
  - LED D3 should blink

Lab 1: C30 Project

C Source File

Linker Script

C30 Assembler Compiler And Linker



Lab 1: Template File

```
//Include the appropriate header (.h) file, depending on device used
//Example (for PIC24FJ128GA010): #include <p24FJ128GA010.h>

#include <p24FJ128GA010.h>


//Configuration Bit Macros to define device config registers.
//Device header file contains usage details

_CONFIG1(JTAGEN_OFF & GCP_OFF & GWRP_OFF & BKBUG_OFF &
         ICS_PGx1 & FWDTEN_OFF & WINDIS_OFF)

_CONFIG2(IESO_OFF & FCKSM_CSDCMD & OSCIOFNC_ON &  FNOSC_FRC &
         POSCMOD_NONE)


// Define constants here

#define CONSTANT1 10
#define CONSTANT2 20
```

Include statements, Configuration macros, and Define statements

```
/************* START OF GLOBAL DEFINITIONS ********************

//Define arrays: array1[], array2[], etc.
//Use the entire attribute OR with macro from header file
int __attribute__((__space__(data), __aligned__(32)))
        ram_array[CONSTANT1];

int _BSS(32) ram_array2[CONSTANT1] ;

//Array without attributes
int ram_array3[CONSTANT2];   //array3 is NOT an aligned buffer

//Program memory array
int __attribute__ ((__space__(prog)))
        rom_array[] = {0x1, 0x2, 0x3, 0x4, 0x5};

//Program memory array using const
const int rom_array2[] = {0x6, 0x7, 0x8, 0x9, 0xA};


//Define global variables with attributes
int var1 __attribute__((__space__(data)));

//near pointer to program memory
int *rom_ptr __attribute__ ((near));

//Define global variables without attributes
int var2;
int *ram_ptr;
/************** END OF GLOBAL DEFINITIONS *********************
```

Global variable declarations with and without attributes

Lab 1: Template File Continued

```
/************* START OF MAIN FUNCTION *********

int main ( void )
{

    //Begin application code here

    while(1);
}
/************* END OF MAIN FUNCTION ***********
```

- Templates for main function and interrupt service routines
- Note the use of the various ISR attributes

```
/****** START OF INTERRUPT SERVICE ROUTINES ***************************/
/*
 * The names of various interrupt functions for
 * each device are defined in the linker script.
 */

//Interrupt Service Routine 1
//No fast context save, and no variables stacked
void __attribute__((__interrupt__,auto_psv)) _ADC1Interrupt(void)
{

// Interrupt Service Routine code goes here
}

//Interrupt Service Routine 2
//Fast context save (using push.s and pop.s)
void __attribute__((__interrupt__, __shadow__,auto_psv))
_T1Interrupt(void)
{

//Interrupt Service Routine code goes here
}

//Interrupt Service Routine 3: INT0Interrupt
//Save and restore variables var1, var2, etc.
void __attribute__((__interrupt__(__save__(var1,var2)),auto_psv))
_INT0Interrupt(void)
{

//Interrupt Service Routine code goes here
}

/********* END OF INTERRUPT SERVICE ROUTINES ************************/
```

Lab 1: Code

Add this code to main function

Create delay function

Now Compile! (F10)

MPLAB IDE Editor

Lab1_solution.c

```
136  /************** START OF MAIN FUNCTION ************
137
138  int main ( void )
139  {
140
141      TRISAbits.TRISA0 = 0; //setup for LED output
142
143      while(1){
144
145          __builtin_btg((unsigned int*)&LATA,0x0);
146
147          delay(); //wait for some time
148
149      }
150  }
151
152
153  void delay(void)
154  {
155      unsigned int i;
156
157      //delay for a while
158      for(i = 0; i < 0xFFFF; i++);
159
160
161  }
162  /************** END OF MAIN FUNCTION **************
```

Lab 1: Program Memory View

Reset and Interrupt Vectors

Initialization code inserted from `crt0.s`

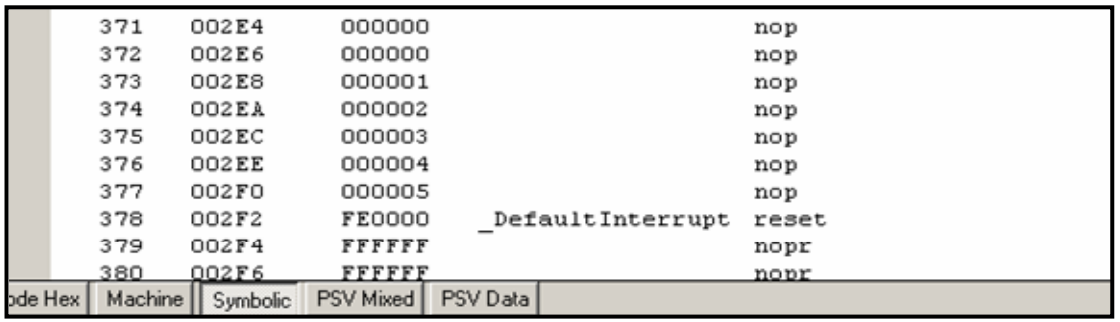
`main()` and `delay()` functions

Interrupt Service Routines

Line	Address	Opcode	Label	Disassembly
1	00000	040200		goto _reset
2	00002	000000		nop
3	00004	000312		_DefaultInterrupt
4	00006	000312		_DefaultInterrupt
257	00200	20880F	_reset	mov.w #0x880,w15
258	00202	227F80		mov.w #0x27f8,w0
259	00204	880100		mov.w w0,0x0020
260	00206	000000		nop
261	00208	070005		rcall _psv_init
262	0020A	07000C		rcall _data_init
263	0020C	020280		call main
264	0020E	000000		nop
265	00210	DA4000		ReservedBR
266	00212	FE0000		reset
321	00280	FA0000	main	lnk #i
322	00282	A902C0		bclr.b 0x02c0,#0
323	00284	202C40		mov.w #0x2c4,w0
324	00286	A20010		btg [w0],#0
325	00288	070001		rcall delay
326	0028A	37FFFC		bra 0x000284
327	0028C	FA0002	delay	lnk #0x2
328	0028E	EB0000		clr.w w0
329	00290	780F00		mov.w w0,[w14]
330	00292	370001		bra 0x000296
331	00294	E80F1E		inc.w [w14],[w14]
332	00296	EB8000		setm.w w0
333	00298	100F9E		subr.w w0,[w14],[w15]
334	0029A	3AFFFC		bra nz, 0x000294
335	0029C	FA8000		ulnk
336	0029E	060000		return
337	002A0	FA0000	_ADC1Interrupt	lnk #i
338	002A2	FA8000		ulnk
339	002A4	064000		retfie
340	002A6	FEA000	_T1Interrupt	push.s
341	002A8	FA0000		lnk #i
342	002AA	FA8000		ulnk
343	002AC	FE8000		pop.s
344	002AE	064000		retfie
345	002B0	781F80	_INT0Interrupt	mov.w w0,[w15++]
346	002B2	208280		mov.w #0x828,w0

Lab 1: Program Memory & Disassembly Listing

Program Memory Variables



```
371   002E4   000000                        nop
372   002E6   000000                        nop
373   002E8   000001                        nop
374   002EA   000002                        nop
375   002EC   000003                        nop
376   002EE   000004                        nop
377   002F0   000005                        nop
378   002F2   FE0000   _DefaultInterrupt    reset
379   002F4   FFFFFF                        nopr
380   002F6   FFFFFF                        nopr
```

ode Hex | Machine | Symbolic | PSV Mixed | PSV Data

Disassembly Listing

The assembly code that the compiler generates

```
135:                /************* START OF MAIN FUNCTION ********************************/
136:
137:                int main ( void )
138:                {
 00280  FA0000      lnk #0x0
139:
140:                 TRISAbits.TRISA0 = 0; //setup for LED output
 00282  A902C0      bclr.b 0x02c0,#0
141:
142:                 while(1){
143:
144:                      __builtin_btg(&LATA,0x0); //toggle the LED pin
 00284  202C40      mov.w #0x2c4,0x0000
 00286  A20010      btg [0x0000],#0
145:
146:                     delay(); //wait for some time
 00288  070001      rcall 0x00028c
147:
148:                 }
 0028A  37FFFC      bra 0x000284
149:                }
150:
```

Lab 1: Map File

- Locate memory sections in Map file

```
Program Memory Usage

section                   address   length (PC units)   length (bytes)  (dec)
-------                   -------   -----------------   --------------------
.reset                          0                 0x4              0x6  (6)
.ivt                          0x4                 0xfc           0x17a  (378)
.aivt                       0x104                 0xfc           0x17a  (378)
.text                       0x200                 0xca           0x12f  (303)
.const                      0x2ca                  0xa            0xf  (15)
.dinit                      0x2d4                 0x14           0x1e  (30)
.prog                       0x2e8                  0xa            0xf  (15)
.isr                        0x2f2                  0x2            0x3  (3)
__CONFIG2                 0x157fc                  0x2            0x3  (3)
__CONFIG1                 0x157fe                  0x2            0x3  (3)

                  Total program memory used (bytes):           0x46e  (1134) <1%

Data Memory Usage

section                   address      alignment gaps    total length  (dec)
-------                   -------      --------------    --------------------
.nbss                       0x800                   0            0x30  (48)
_01993c404685865b           0x840                   0            0x20  (32)
_01993d904685865b           0x860                   0            0x20  (32)

                       Total data memory used (bytes):           0x70  (112) 1%


Dynamic Memory Usage

region                    address                    maximum length  (dec)
------                    -------                    --------------------
heap                            0                                 0  (0)
stack                       0x880                            0x1f80  (8064)

                     Maximum dynamic memory (bytes):          0x1f80  (8064)
```

Memory usage data

```
External Symbols in Data Memory (by address):

                    0x0800                  _ram_array3
                    0x0828                  _var1
                    0x082a                  _rom_ptr
                    0x082c                  _var2
                    0x082e                  _ram_ptr
                    0x0840                  _ram_array
                    0x0860                  _ram_array2

External Symbols in Data Memory (by name):

                    0x0840                  _ram_array
                    0x0860                  _ram_array2
                    0x0800                  _ram_array3
                    0x082e                  _ram_ptr
                    0x082a                  _rom_ptr
                    0x0828                  _var1
                    0x082c                  _var2

External Symbols in Program Memory (by address):

                  0x000200                  __resetPRI
                  0x000280                  _main
                  0x00028c                  _delay
                  0x0002a0                  __ADC1Interrupt
                  0x0002a6                  __T1Interrupt
                  0x0002b0                  __INT0Interrupt
                  0x0002ca                  _rom_array2
                  0x0002e8                  _rom_array
                  0x0002f2                  __DefaultInterrupt

External Symbols in Program Memory (by name):

                  0x0002a0                  __ADC1Interrupt
                  0x0002f2                  __DefaultInterrupt
                  0x0002b0                  __INT0Interrupt
                  0x0002a6                  __T1Interrupt
                  0x000200                  __resetPRI
                  0x00028c                  _delay
                  0x000280                  _main
                  0x0002e8                  _rom_array
                  0x0002ca                  _rom_array2
```

Data Memory Variable Locations

Prog Mem Variable & Function Locations

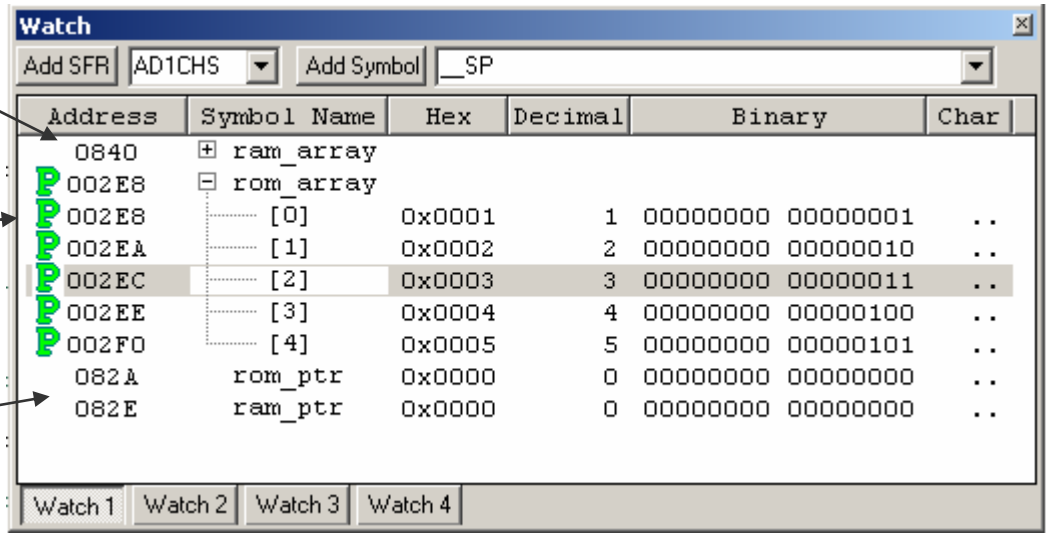
Lab 1: Watch Window

– **Compare addresses to expected values. Do they make sense?**

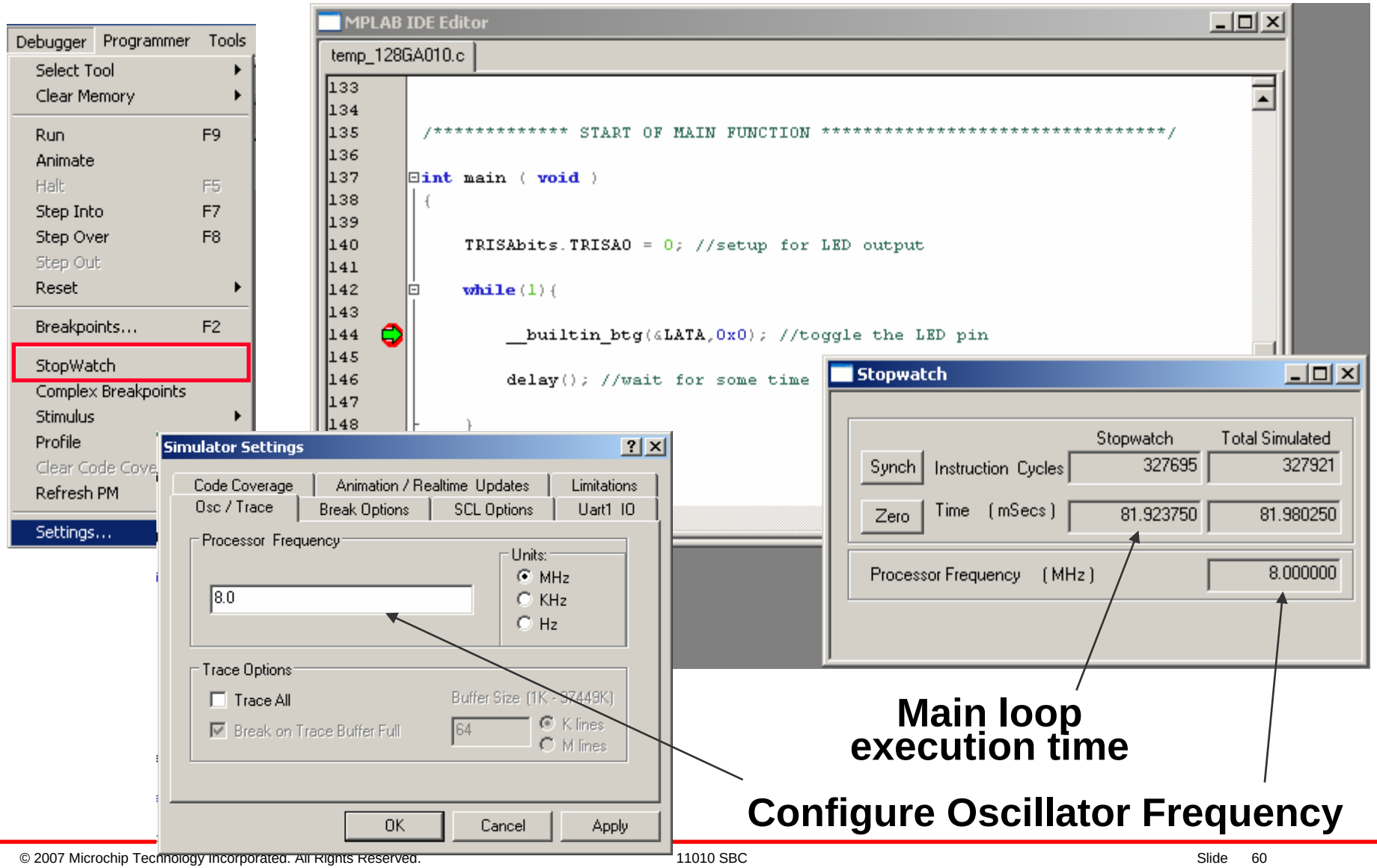
Data memory on 32-byte boundary (lower 5 bits of address 0)

Program Memory Variable – note longer address

Data memory variables



Lab 1: MPLAB® SIM

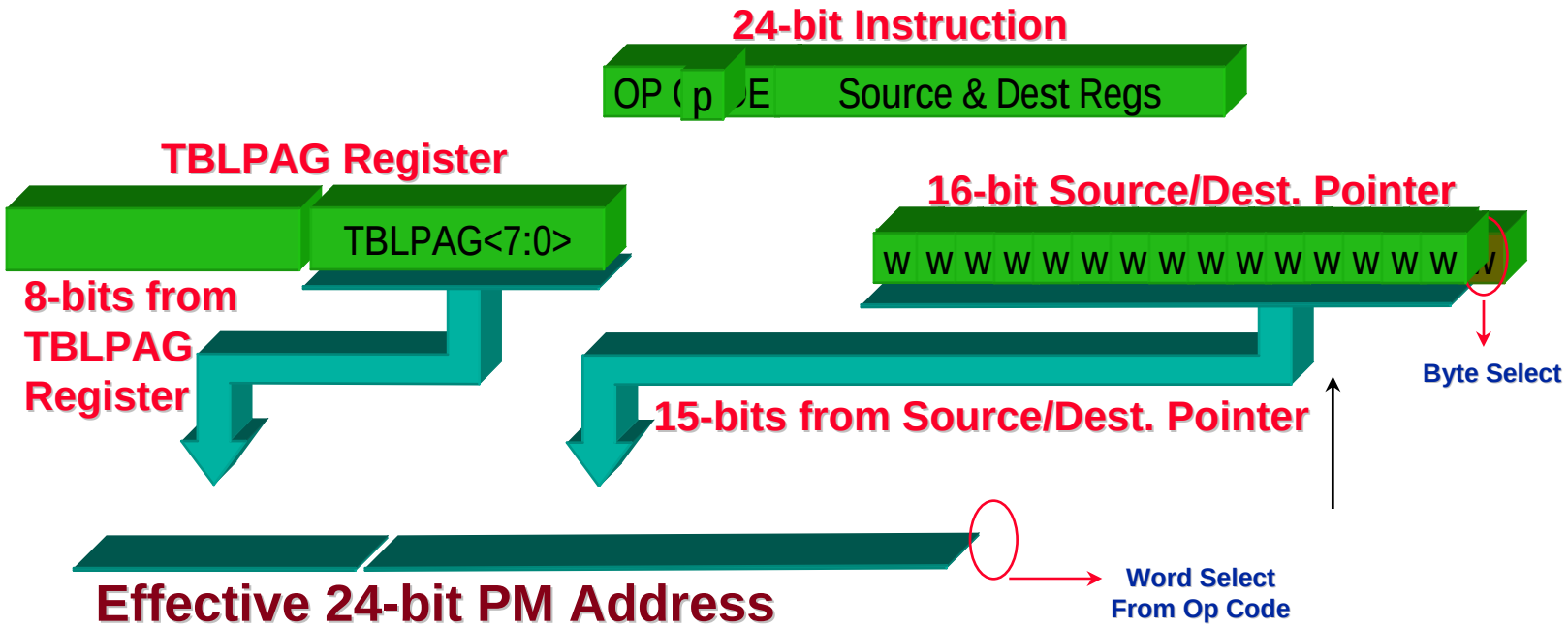


Data Access From Program Memory

Table Read/Write
Program Space Visibility (PSV)

Constant Data Access from PM Using Table Instructions

- **24-bit Effective Address(EA) formation**
  - Configuration space is accessed by setting TBLPAG<7> (i.e. EA<23>)
    - **Only means of accessing configuration space**



Constant Data Access from PM Using Table Instructions

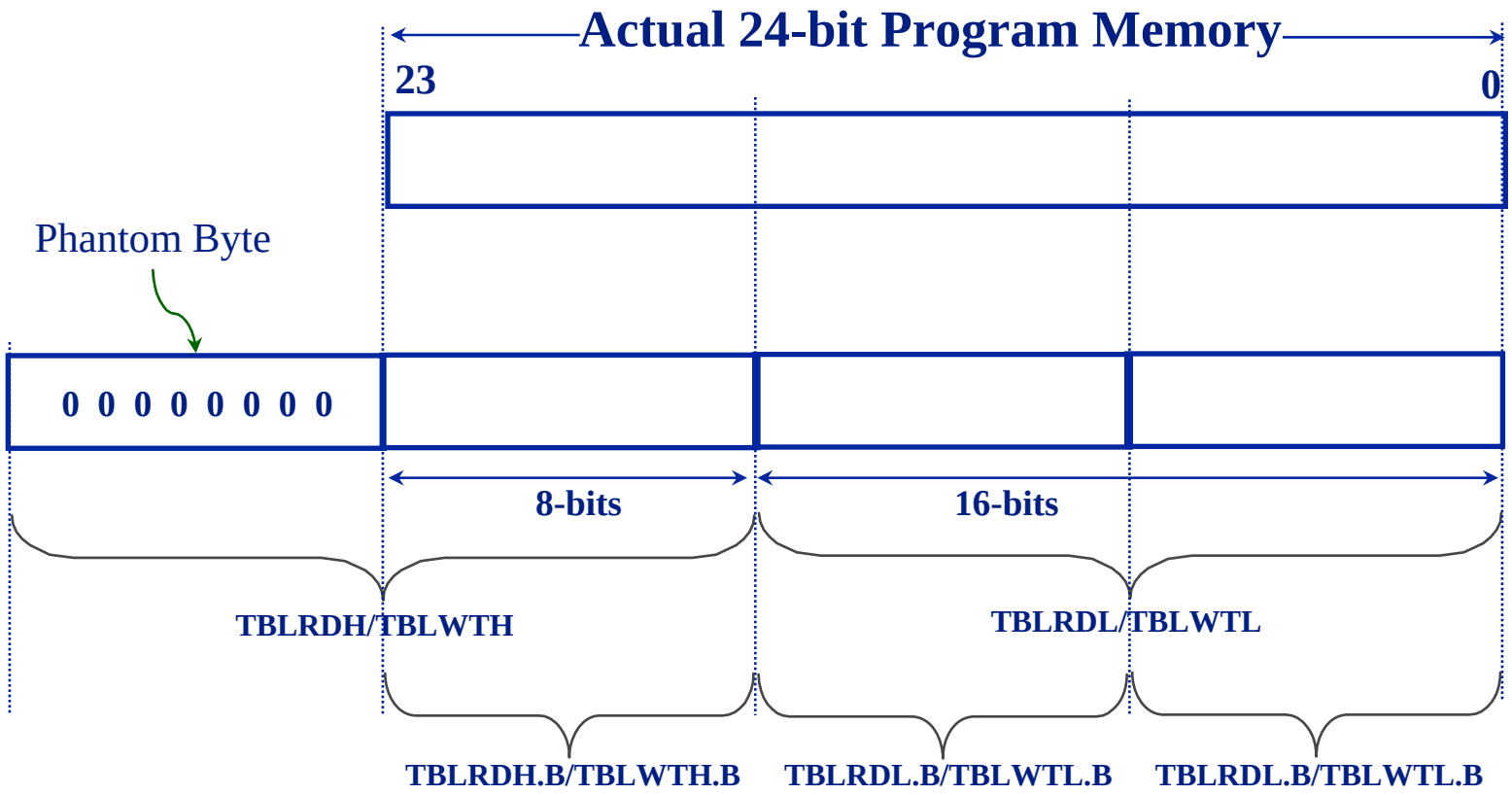


Table Instruction View of Program Memory

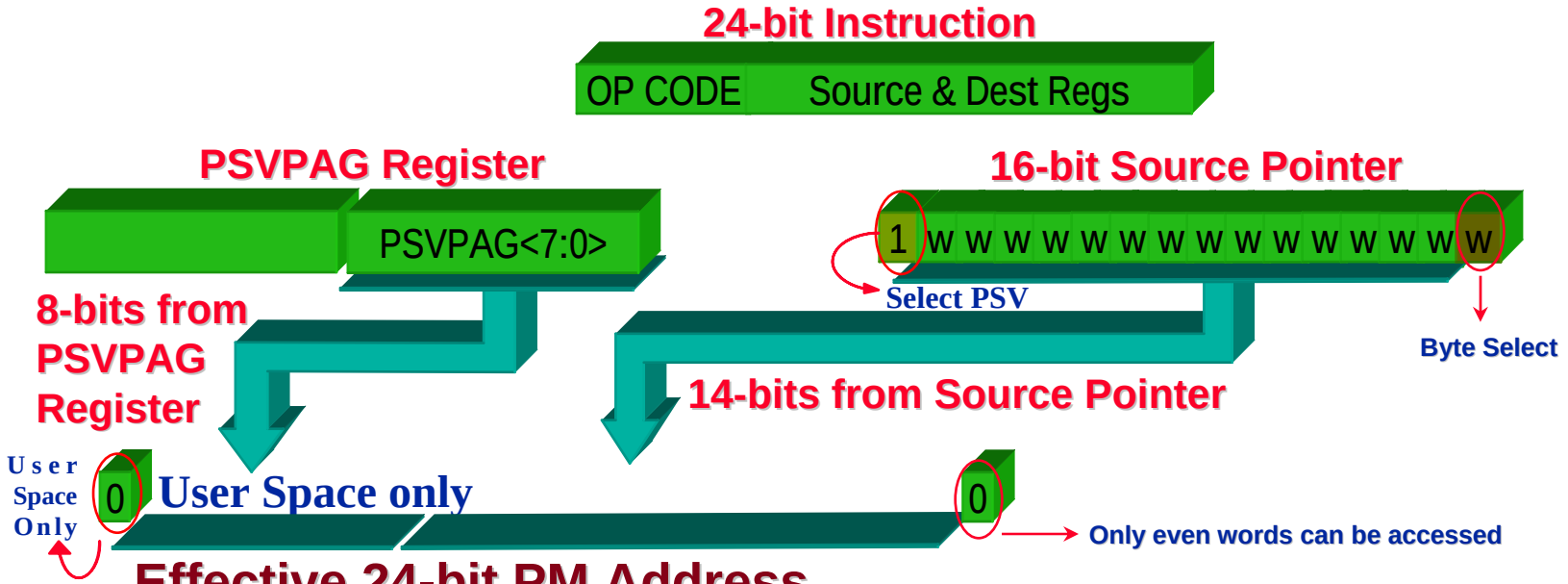
Using Table Read/Write in C30

- **Must use assembly for special instructions and exact timing**
  - Add assembly file to project, or,
  - Use C30 Built-ins:

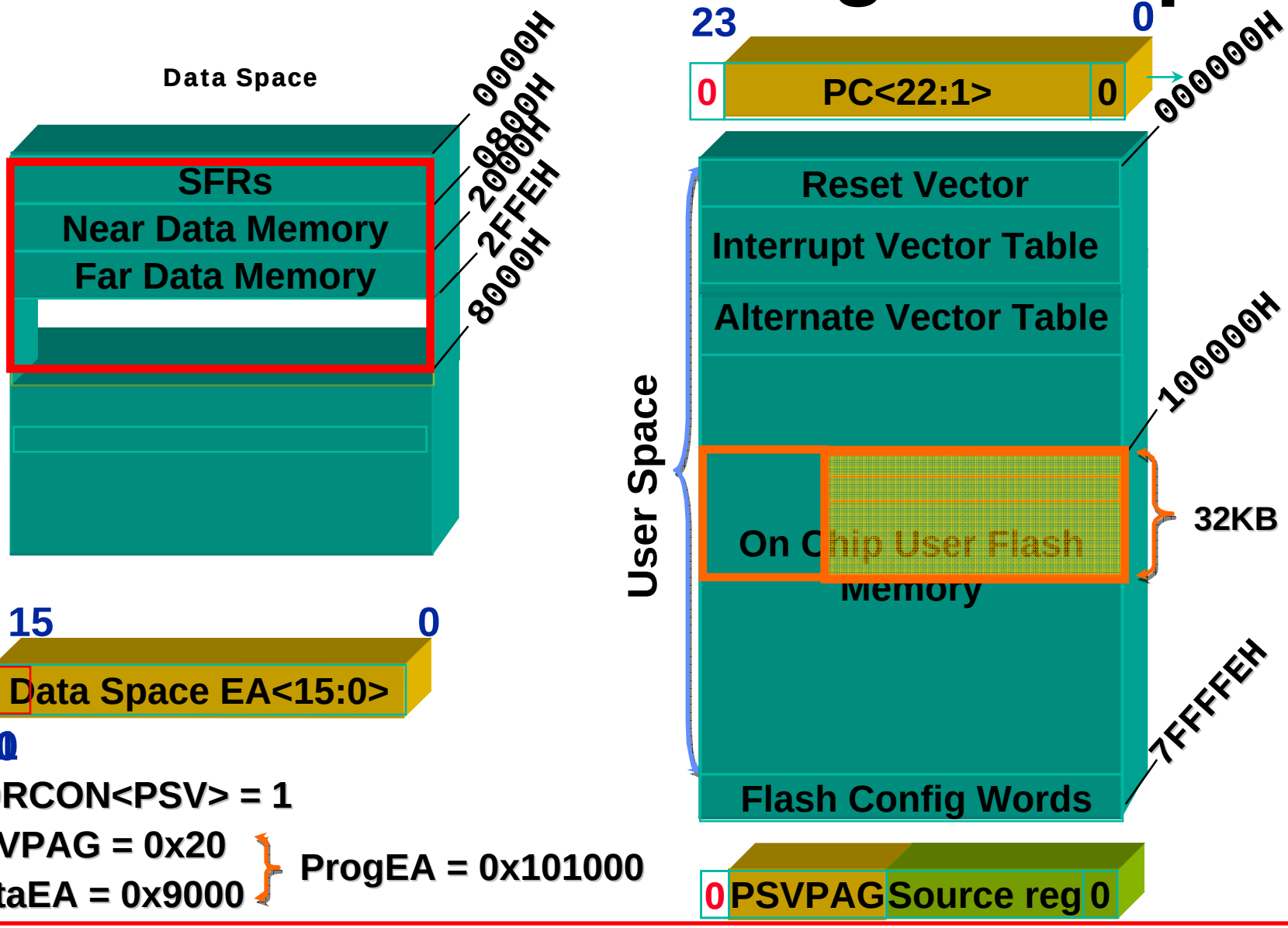
```
const unsigned int myVar = 0xFFFF;
unsigned int varAddr;

TBLPAG = __builtin_tblpage(&myVar);
varAddr = __builtin_tbloffset(&myVar);

__builtin_tblwtl(varAddr, 0xAAAA);  //load data to write:
                                    //0xAAAA
NVMCON = 0x4003;                 //NVM word write opcode
__builtin_write_NVM();           //unlock & set WR bit
```

Constant Data Access from PM Using Program Space Visibility

- **32 KB segment of PM may be mapped into the data memory address space**
  - If PSV bit (CORCON<2>) is set and if SrcPointer (DM EA)<15> is '1' then data is accessed from PM



PSV Addressing Example



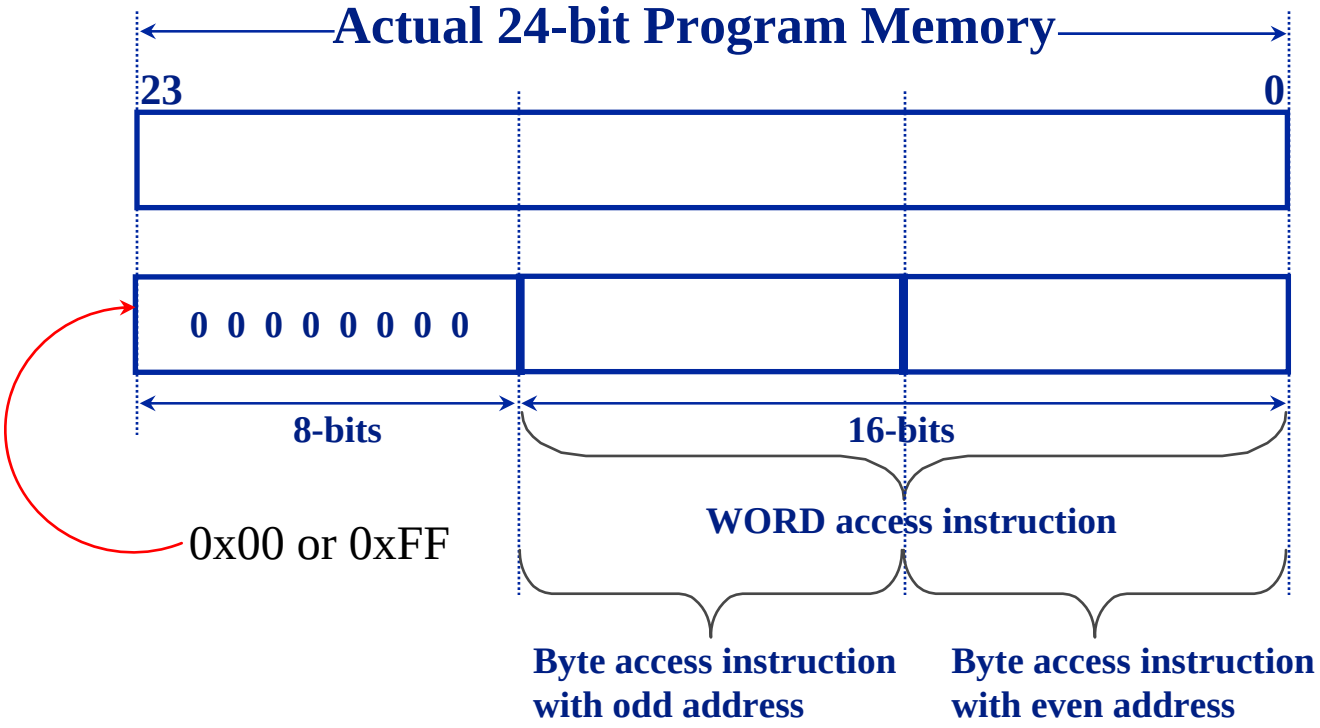
CORCON<PSV> = 1

PSVPAG = 0x20

DataEA = 0x9000

ProgEA = 0x101000

Data Access from PM Using PSV



- **Only lower 16 bits of PM location can be accessed via mapping**
  - Upper 8 bits should be programmed for a **`NOP`**

Using the PSV in C30

- **Three usage options**
  - Manual PSV
    - `char data[256] __attribute__ ((space(psv)))`
    - Must manually point to the correct PSV when accessing the variable
  - Auto PSV
    - `const char data[256];`
    - Maximum of 32 KB of constant data allowed
    - Linker does not allow PSV space to cross a PSVPAG boundary

University of Microchip UofM MASTERs 2007

Using the PSV in C30

- **Compiler Managed PSV**
  - Only in C30 v3.00 and later
  - Allows use of more than 32 KB of constants – compiler automatically sets PSVPAG
  - Must apply attribute space(psv) to variable & declare variable as managed with __psv__ qualifier
    - `__psv__ char data[256] __attribute__ ((space(psv)))`

Advantages of PSV

- **PSV allows very large tables of data to be stored and accessed quickly & efficiently**
- **PSV provides a bridge to a common data/program address space (Von Neumann) but only when needed**
- **Example:**
  - Large constant data for display
  - Sine Table

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Data Access Overhead Using PSV

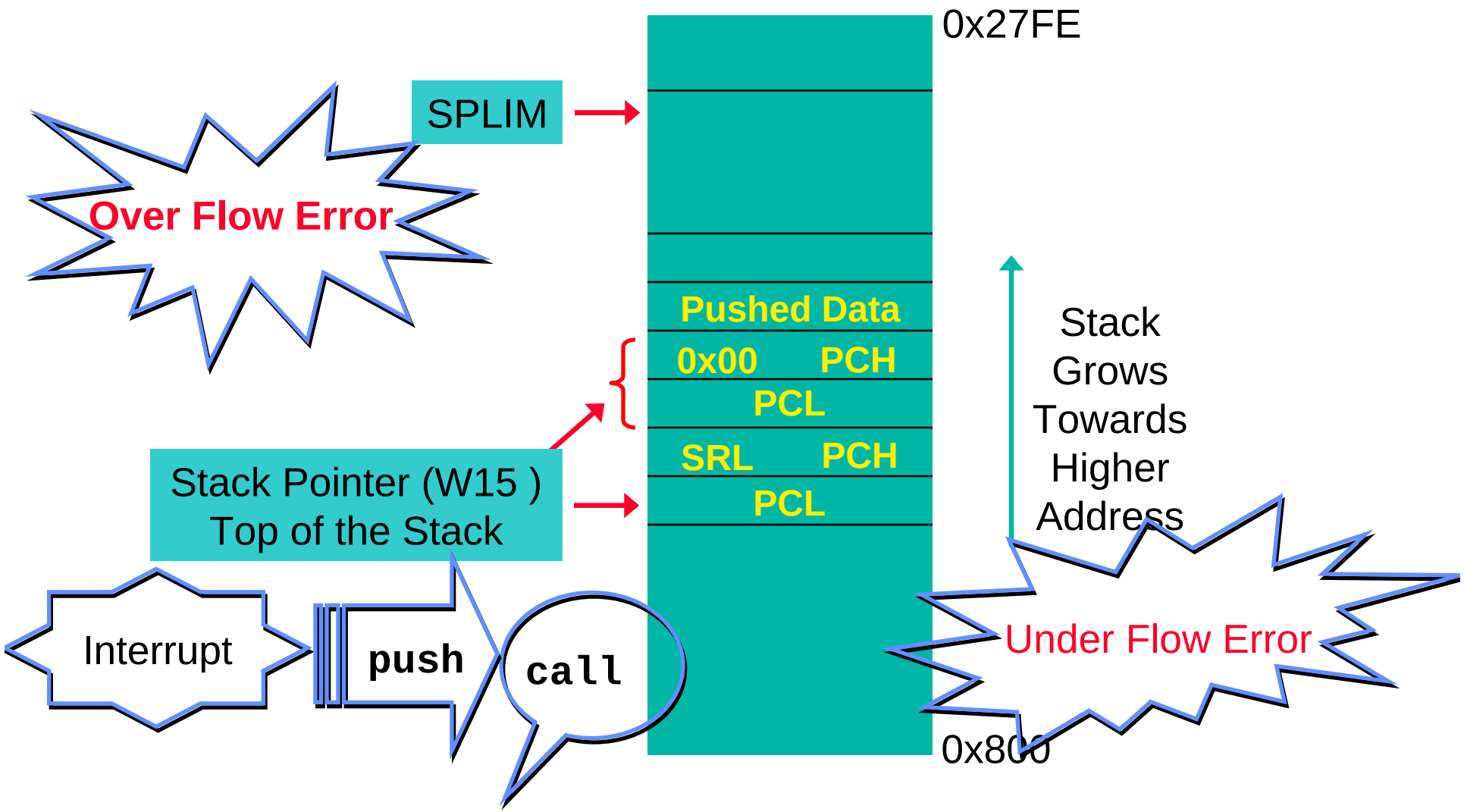
- **PSV data fetch overhead:**
  - Outside a REPEAT loop:
    - **Data move ops, overhead = 1 cycle**
    - **ALU based ops, overhead = 2 cycles**
  - Within a REPEAT loop:
    - **Data pipelined, so overhead for all ops = 0 cycles**
    - **First & last iteration - data pipeline fill/flush**
      - Data move ops, overhead = 1 cycle
      - ALU based ops, overhead = 2 cycles

Lab 2: Working with PSV

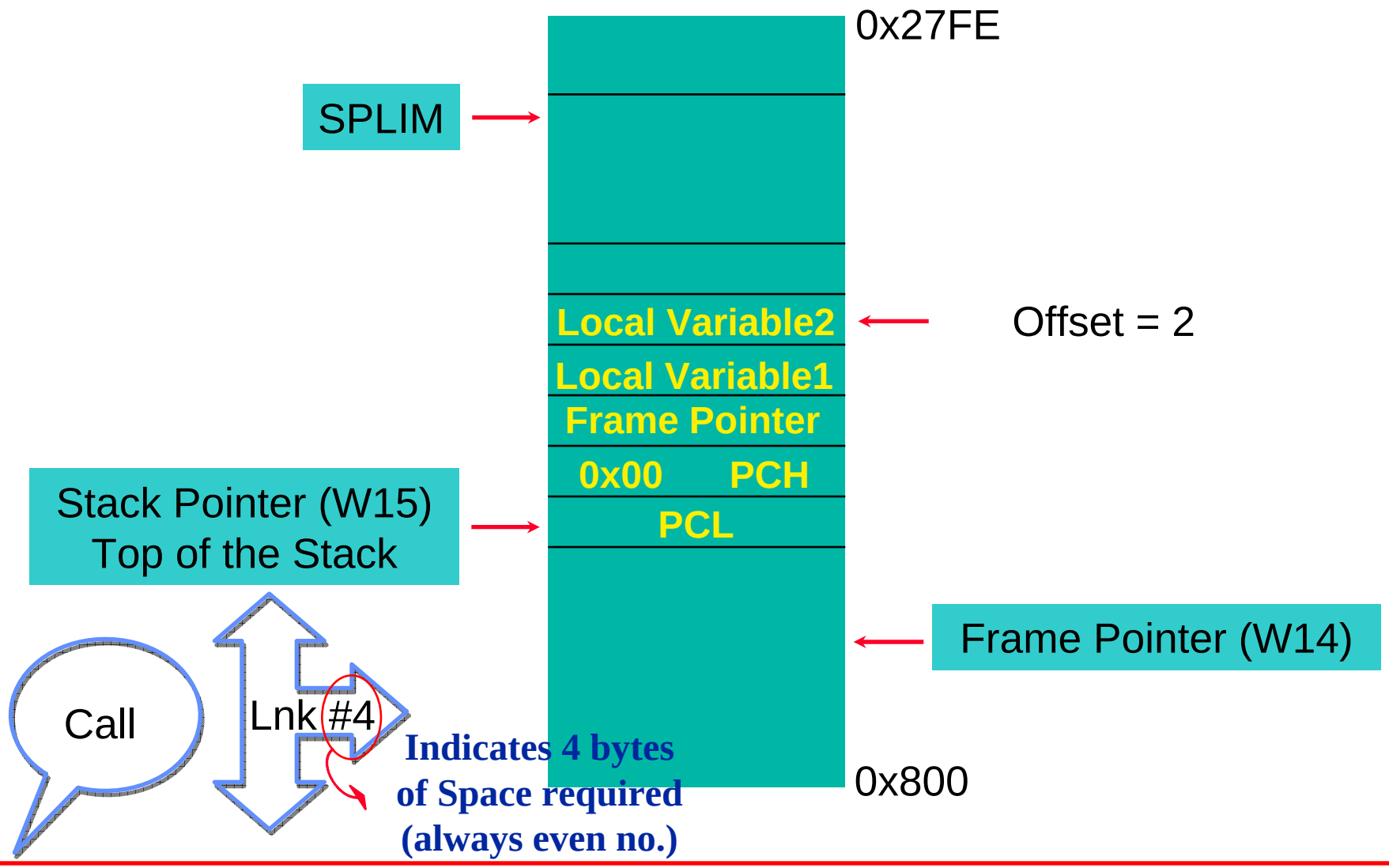
- **Goals:**
  - To initialize PSV
  - To store a Hello World string in PSV space
  - To read and transmit this string to LCD
- **To Do:**
  - Look into the Hand out provided
- **Expected Result:**
  - The text displayed on LCD should match the array defined

Software Stack

Software Stack in Data RAM

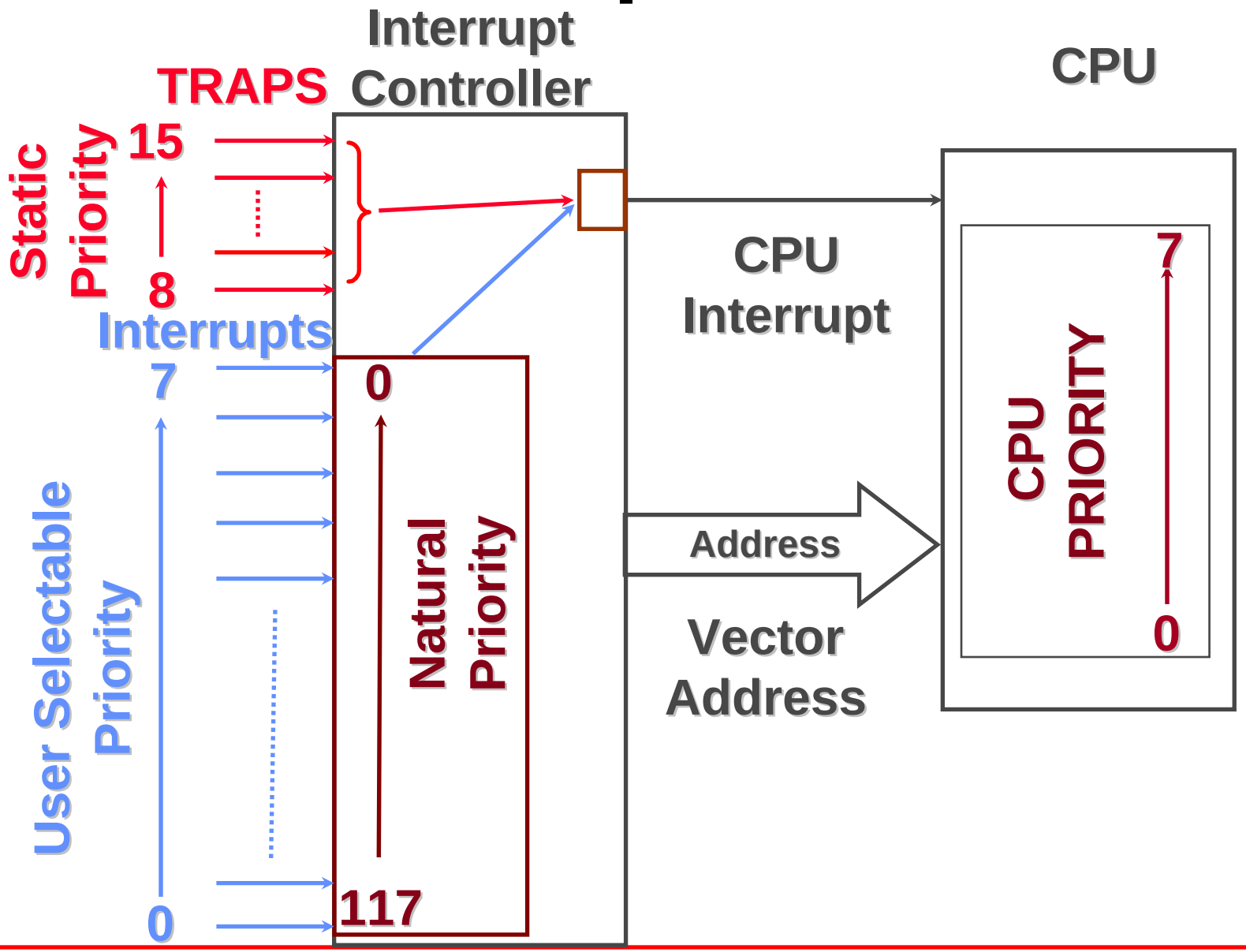


Frame Pointer

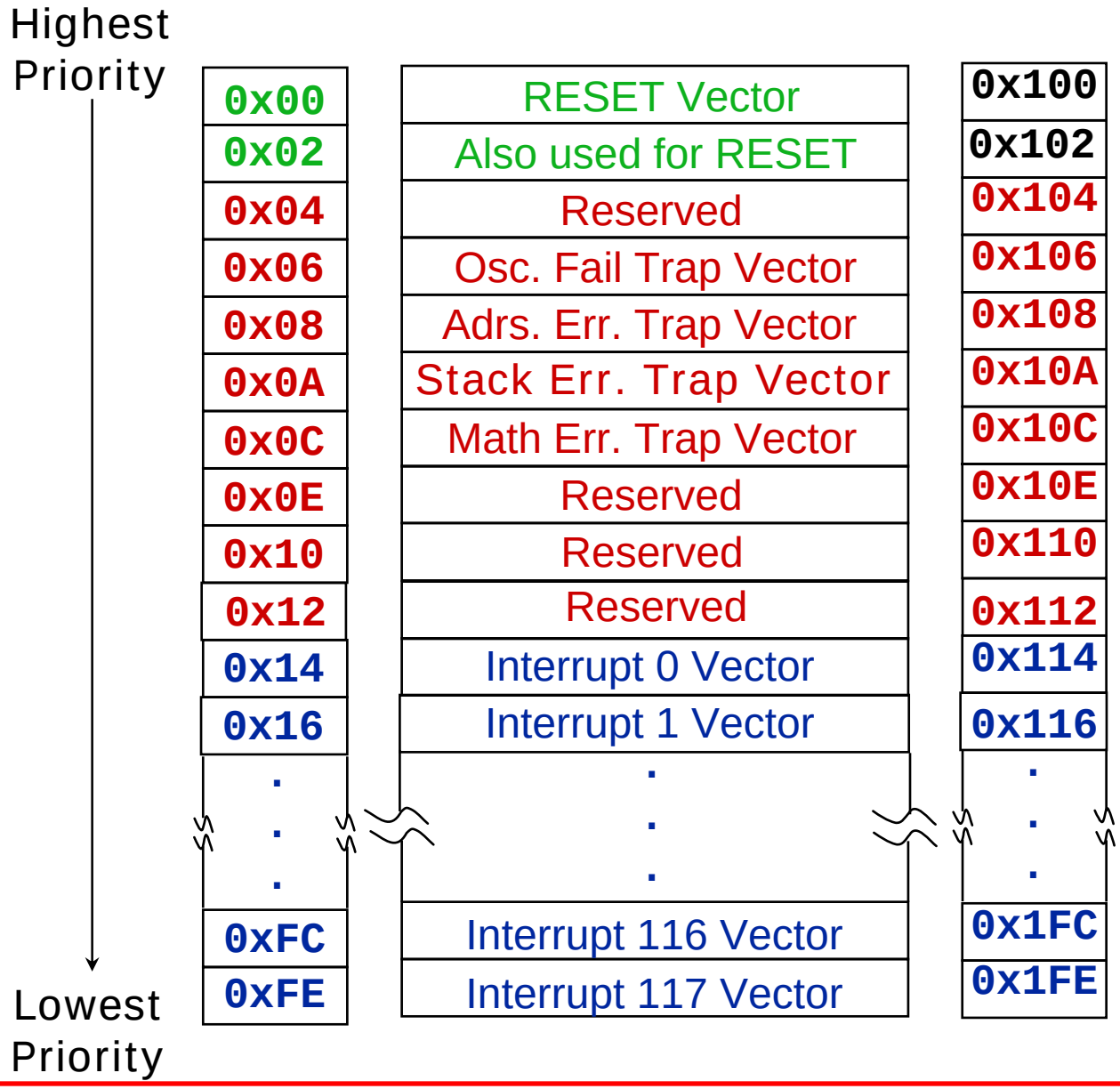


Traps and Interrupts

Interrupt Architecture



IVT / AIVT



University of Microchip UofM MASTERs 2007

AIVT Advantage

- IF “condition1” is true goto “application”
- Else goto “debug”

```
Application_main
        Use IVT
        ...
        .....
End
ISR1            ; add. 0x004
        Process data
```

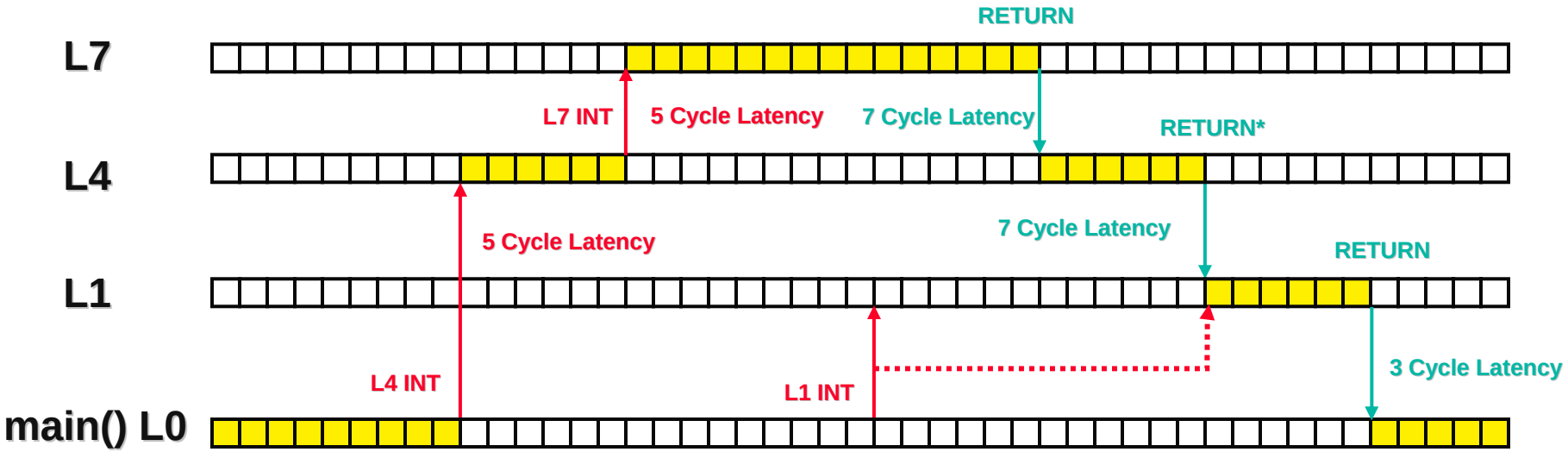
```
Debug_main
        Use AIVT
        ...
        .....
End
ISR1            ; add. 0x104
Glow LED; Visual Indication
```

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Interrupt Priority

- CPU has 16 priority levels
  - **Level 0 is the default CPU level (**main**)**
  - **Level 1 - 7 for user interrupts**
  - **Level 8 -15 reserved for traps (**NMI**)**
- Interrupt with priority level greater than current CPU level (**`IPL<3:0>`**) can interrupt the CPU
- Interrupts are nested by default, Nesting can be disabled by setting **`NSTDIS`** bit in **`INTCON1`** register
- IVT has natural priority to resolve conflicts
- User assigned priority overrides natural priority

Interrupt Nesting Example



CPU EXECUTION TRACE

Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								IPL2	IPL1	IPL0					

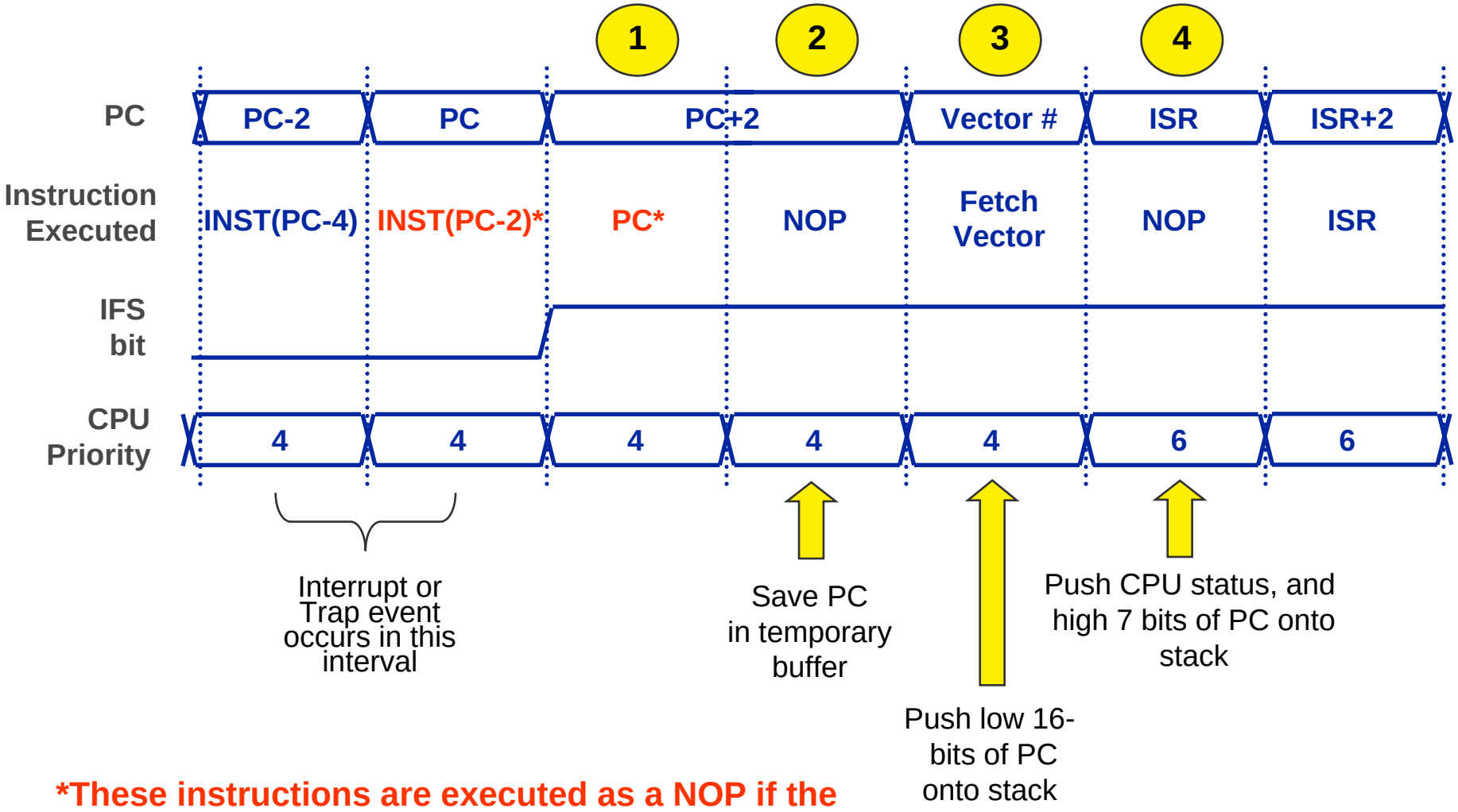
Core Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												IPL3			

IPL<3:0>

3	2	1	0
0	0	0	0

Interrupt Latency



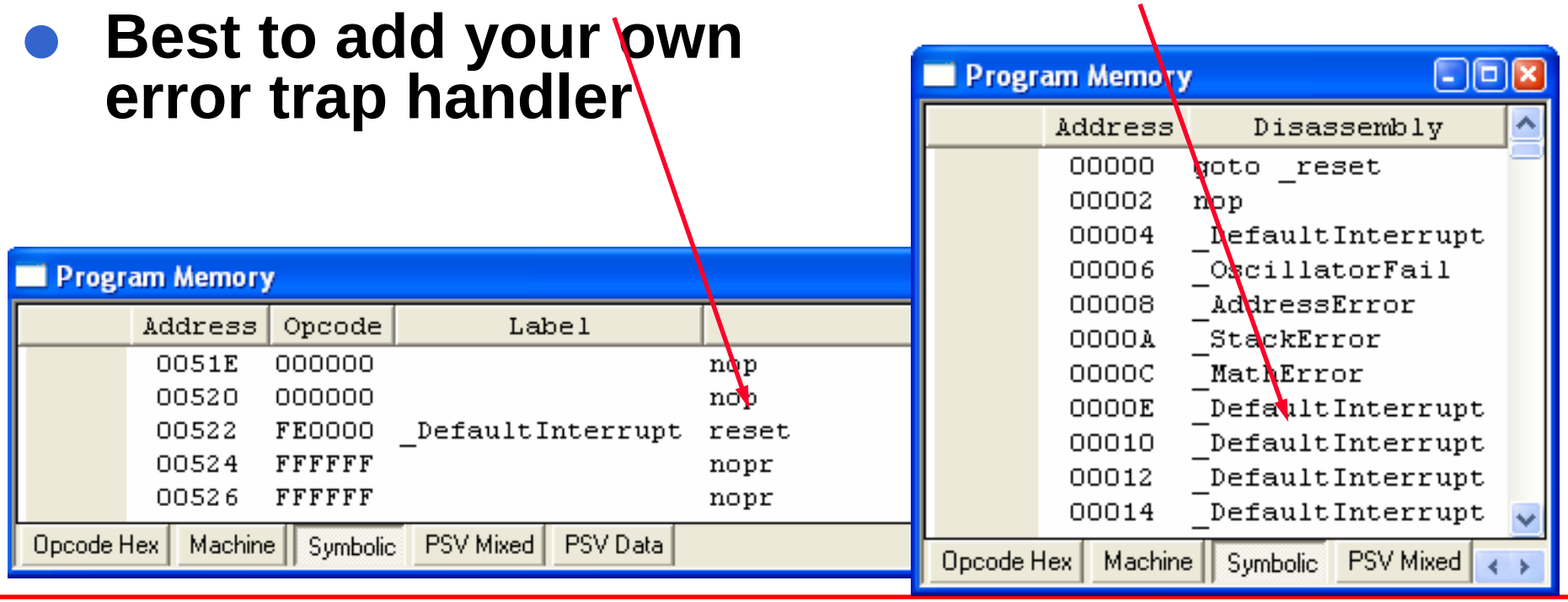
***These instructions are executed as a NOP if the previous instruction causes a Hard Trap**

Traps

- **Soft Traps**
  - Priority levels 8 through 12
  - Treated as an NMI with fixed priority
  - ‘Normal’ exception process is taken
  - Example:
    - **Arithmetic error trap (priority level 11)**
    - **Stack error trap (priority level 12)**
- **Hard Traps**
  - Priority levels 13 through 15
  - CPU execution flow is immediately suspended
  - Execution cannot resume until trap is acknowledged
  - Example:
    - **Address error trap (level 13)**
    - **Oscillator error trap (level 14)**

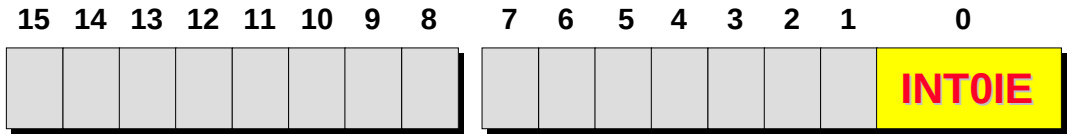
Error Traps

- **What happens when a trap occurs?**
- **Branches to address 0 by default**
- **Better to branch to a `reset` instruction**
  - Linker adds a **`reset`** instruction and vector
- **Best to add your own error trap handler**

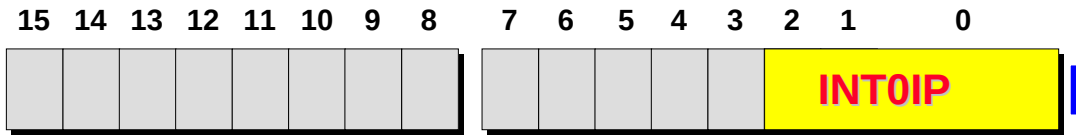


Configuring Interrupts

IEC0: Interrupt Enable Control Register 0 : Enable Interrupt

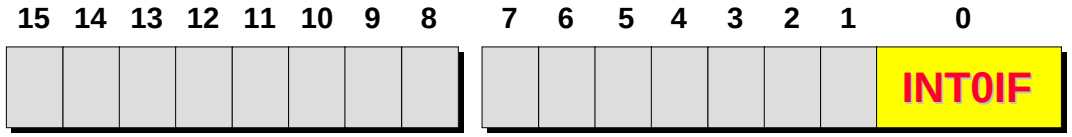


IPC0: Interrupt Priority Control Register 0 : Assign Priority



000: Disabled
001: Priority 1 Interrupt
010: Priority 2 Interrupt
011: Priority 3 Interrupt
100: Priority 4 Interrupt
101: Priority 5 Interrupt
110: Priority 6 Interrupt
111: Priority 7 Interrupt

IFS0: Interrupt Flag Control Register 0 : Clear Interrupt in the ISR



Interrupt Service Routines in C

- **Compiler saves / restores the ISR context**
  - Hardware saves return address and SR
- **Fast ISR saves W0-W3 in one cycle**
  - Uses `push.s` and `pop.s` (only 1 level deep)
- **Globals used by the ISR must be volatile**
  - `volatile unsigned global_flag;`
- **ISR must clear the Interrupt Flag**
  - `IFS0bits.T1IF = 0;`
- **Making function calls not recommended**
- **Linker will populate the Vector Table**
  - Must use designated interrupt names
  - The designated interrupt names can be found in the .gld file of that device, which can be found at `\\….\microchip\mplab c30\support\gld`

University of Microchip UofM MASTERs 2007

Function Attributes

- **Declare an interrupt function**

```
void __attribute__((interrupt)) _ADCInterrupt(void)
void _ISR _IC1Interrupt(void)
```

- **Make function use shadow registers**

```
void __attribute__((interrupt,shadow))
                    _ADCInterrupt(void)
void _ISRFAST _IC1Interrupt(void)
int __attribute__((shadow)) FastFunc(int Abc)
```

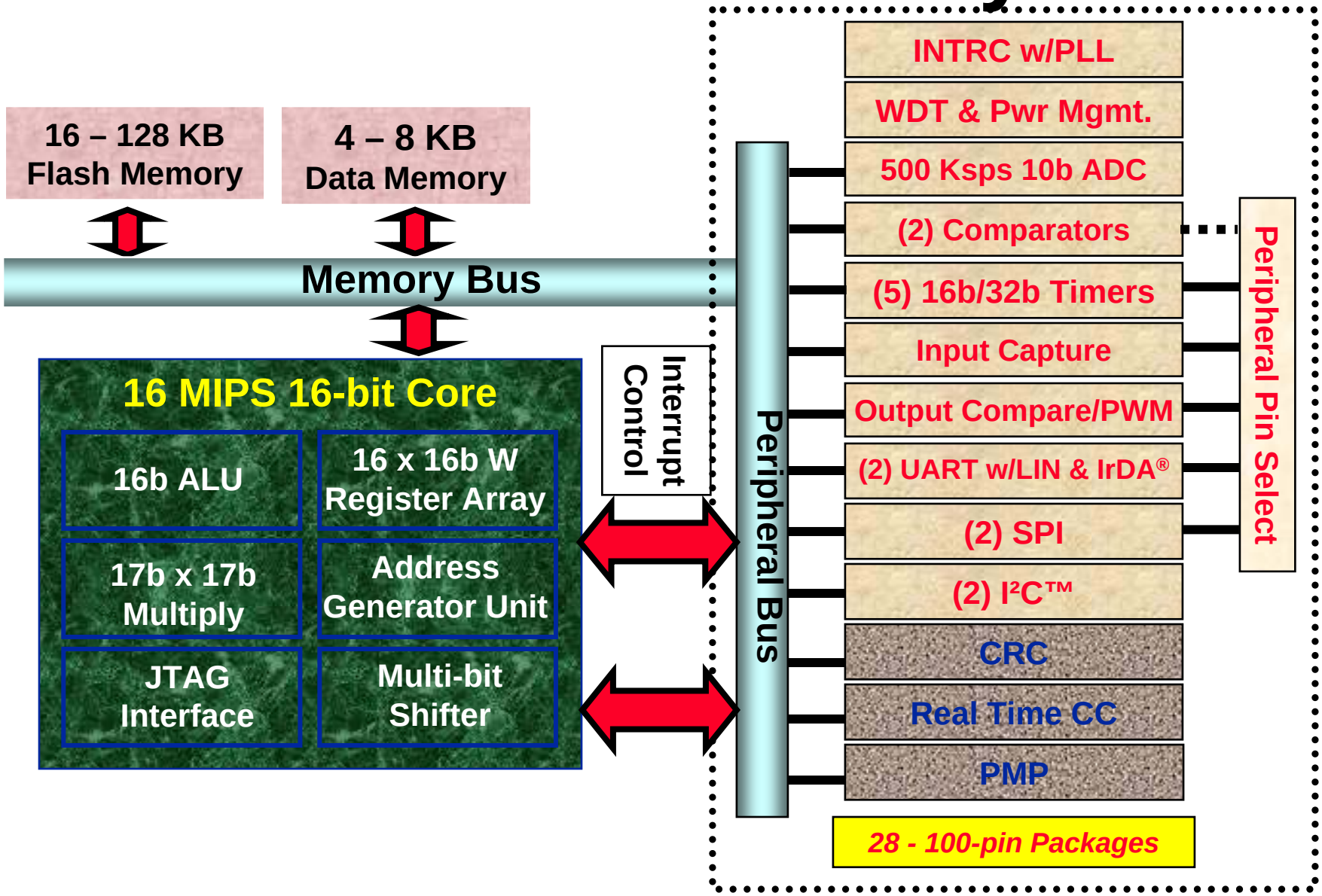
  - Saves W0-W3 and SR bits with **`push.s`** and **`pop.s`**

Lab 3: Working with Interrupts

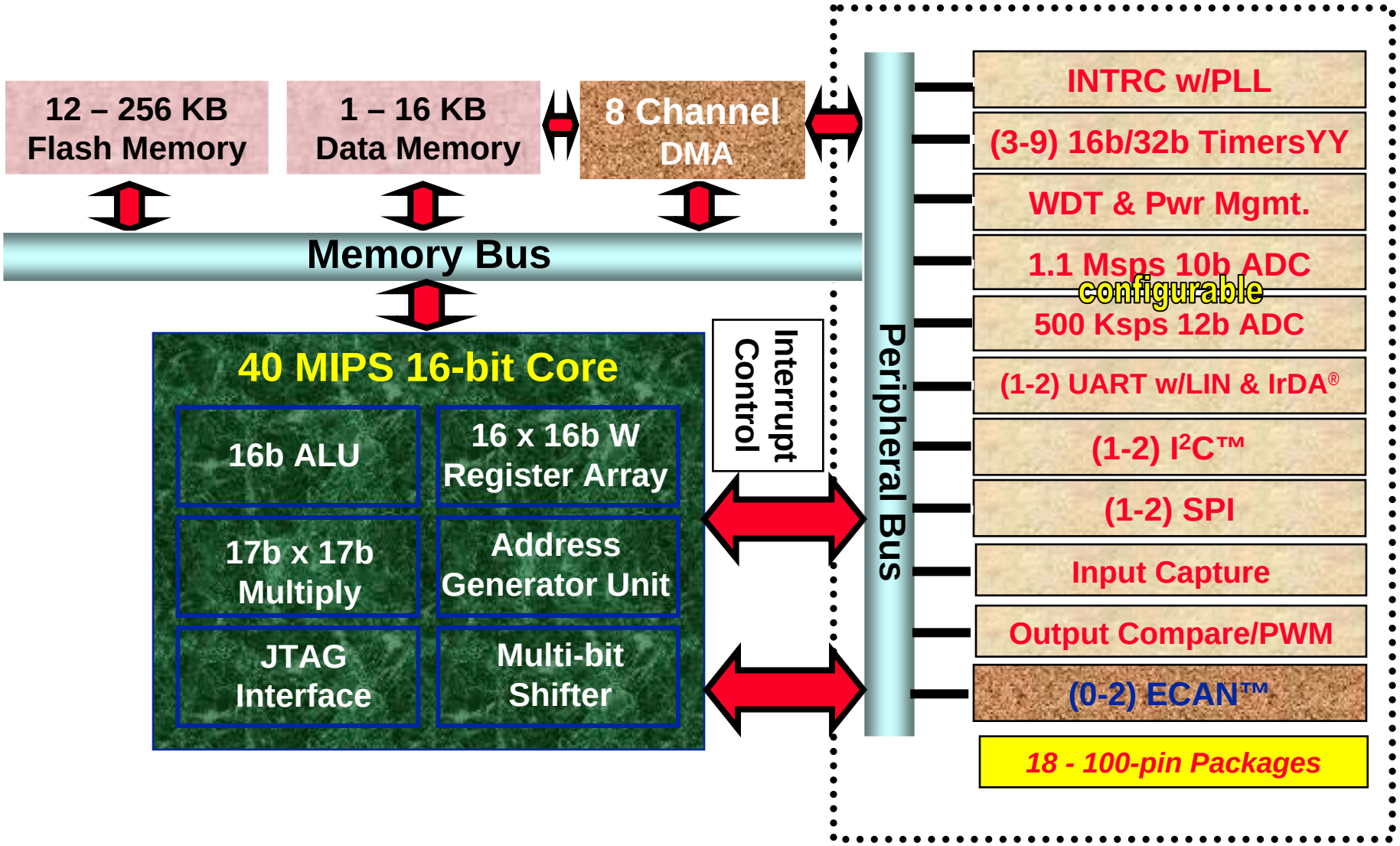
- **Goal:**
  - Understand Interrupt configuration
  - Understand Interrupt priority
  - Writing Interrupt handler for a given Interrupt vector
- **To Do:**
  - Look into the handout provided
- **Expected Result:**
  - Look into the handout provided

16-Bit Peripherals

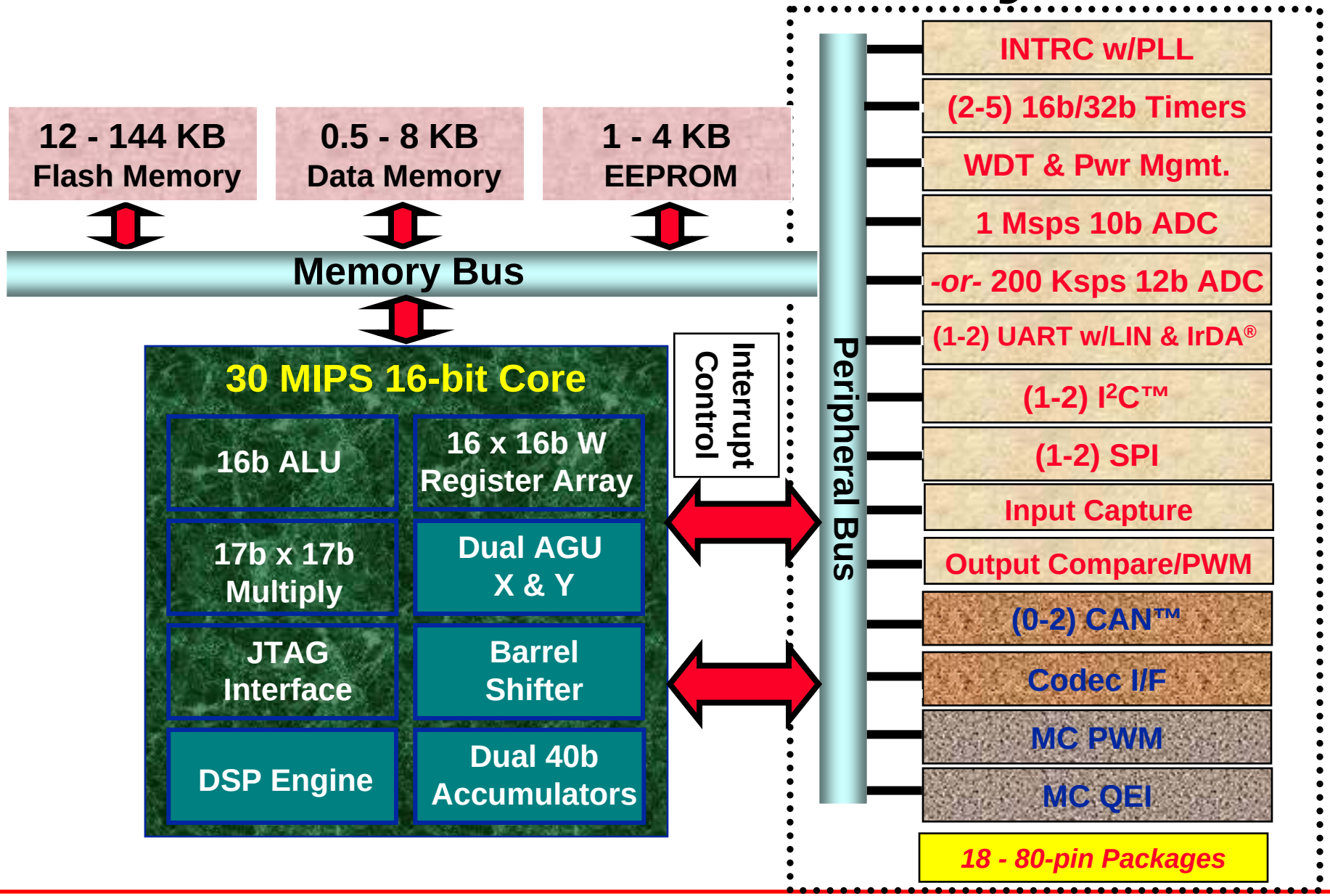
PIC24F Family



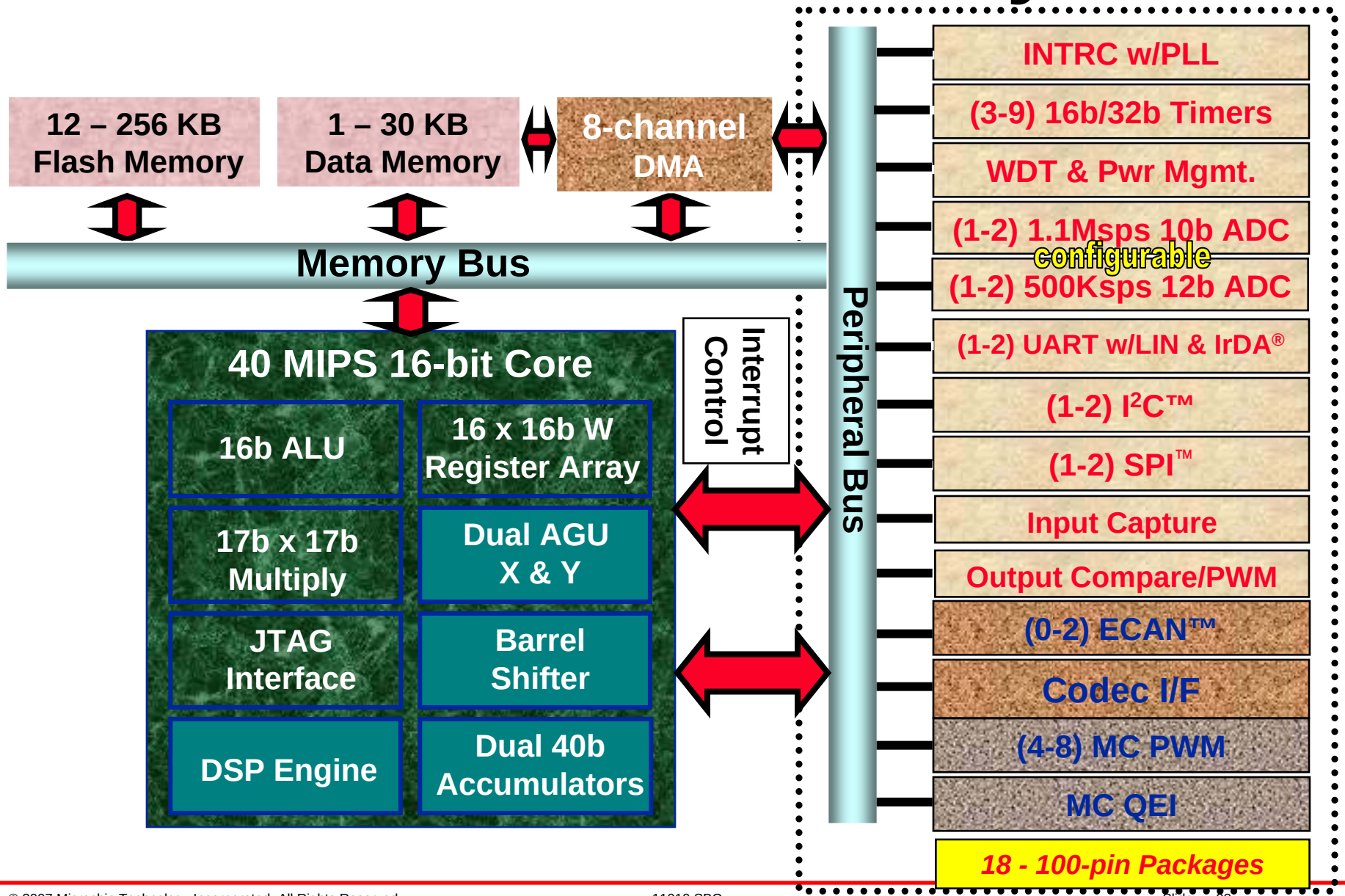
PIC24H Family



dsPIC30F Family

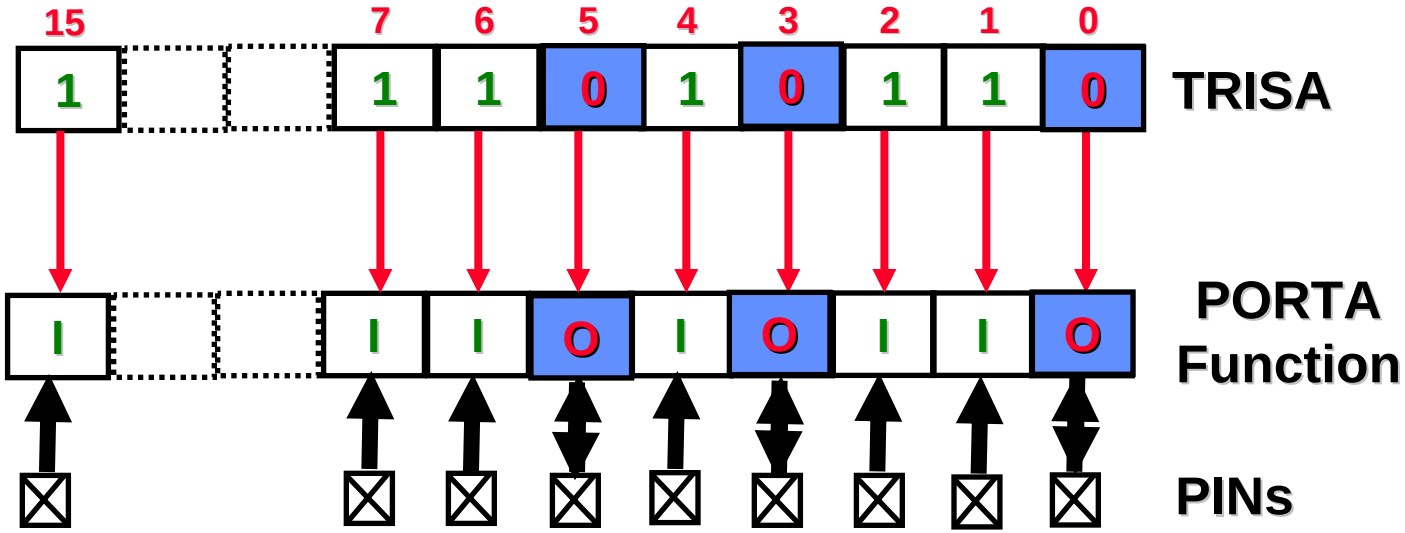


dsPIC33F Family

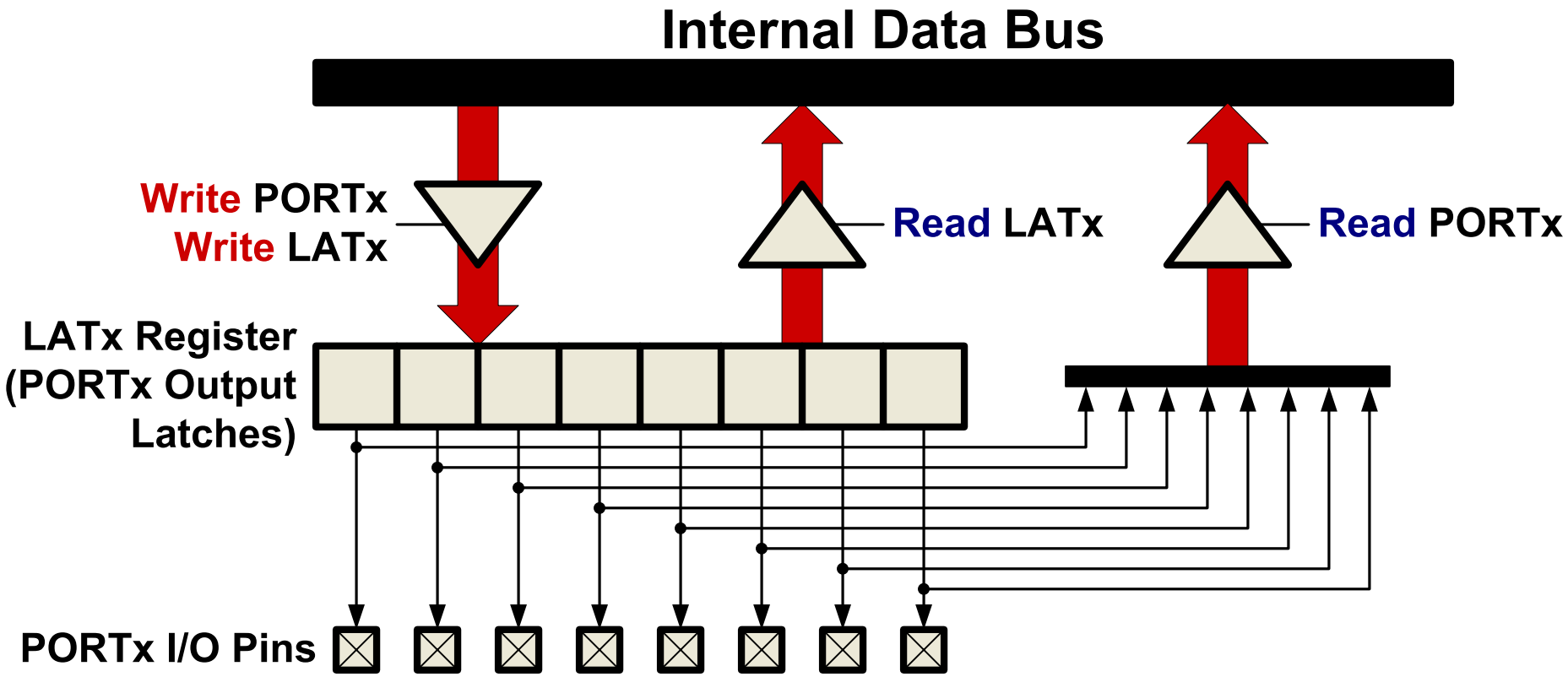


I/O Ports

I/O PORT

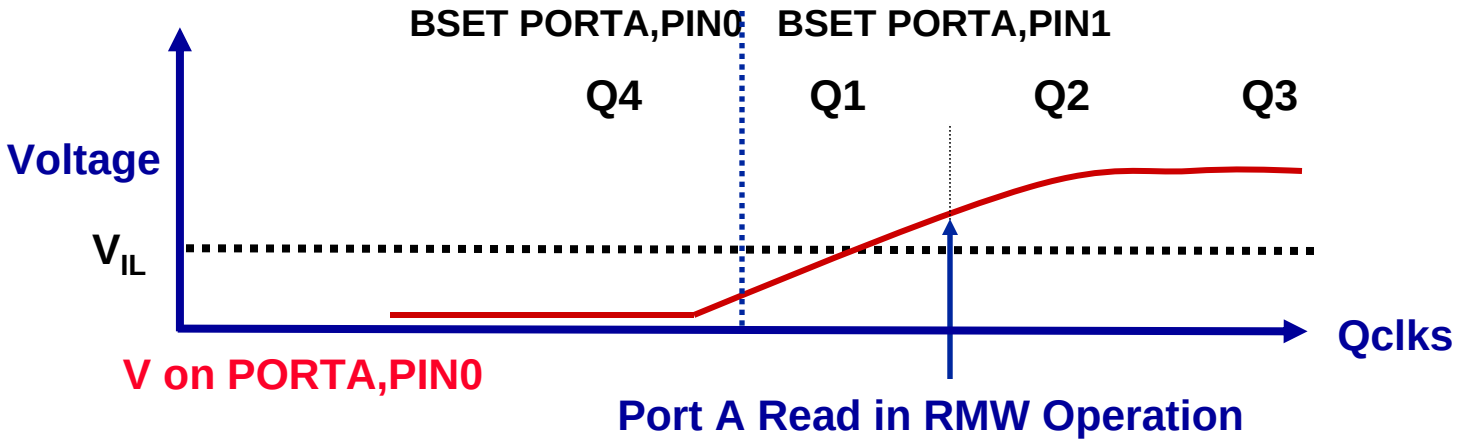


Digital I/O Ports

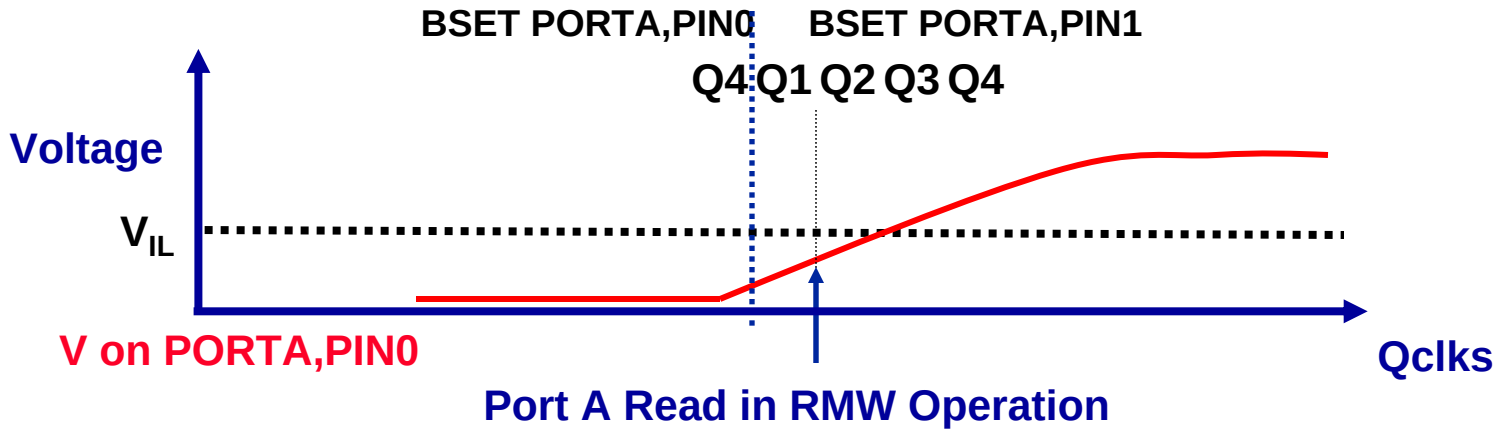


Why do we have LAT Reg?

At Low Frequency or Low Capacitive Loading



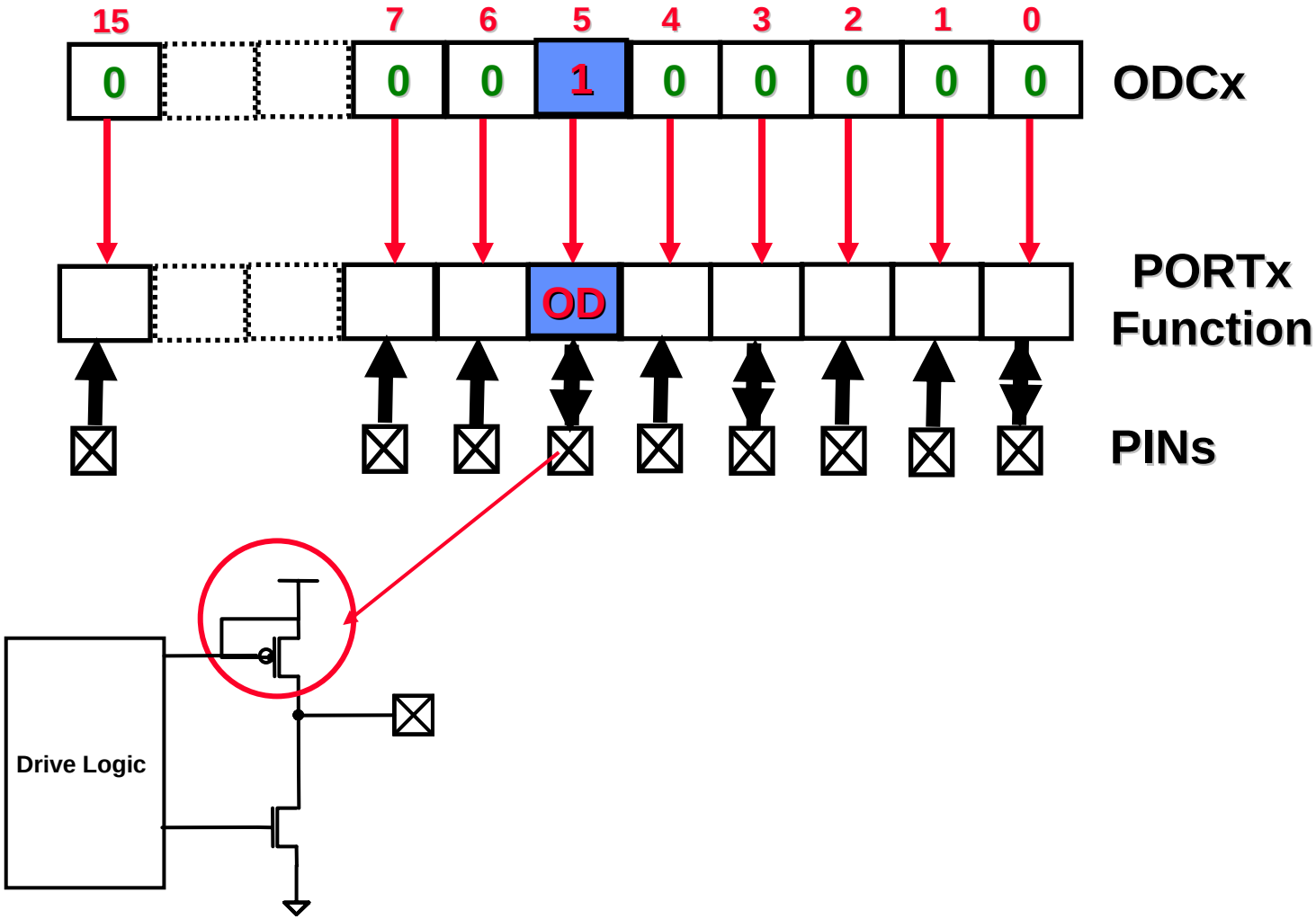
At High Frequency or High Capacitive Loading



Why do we have LAT Reg?

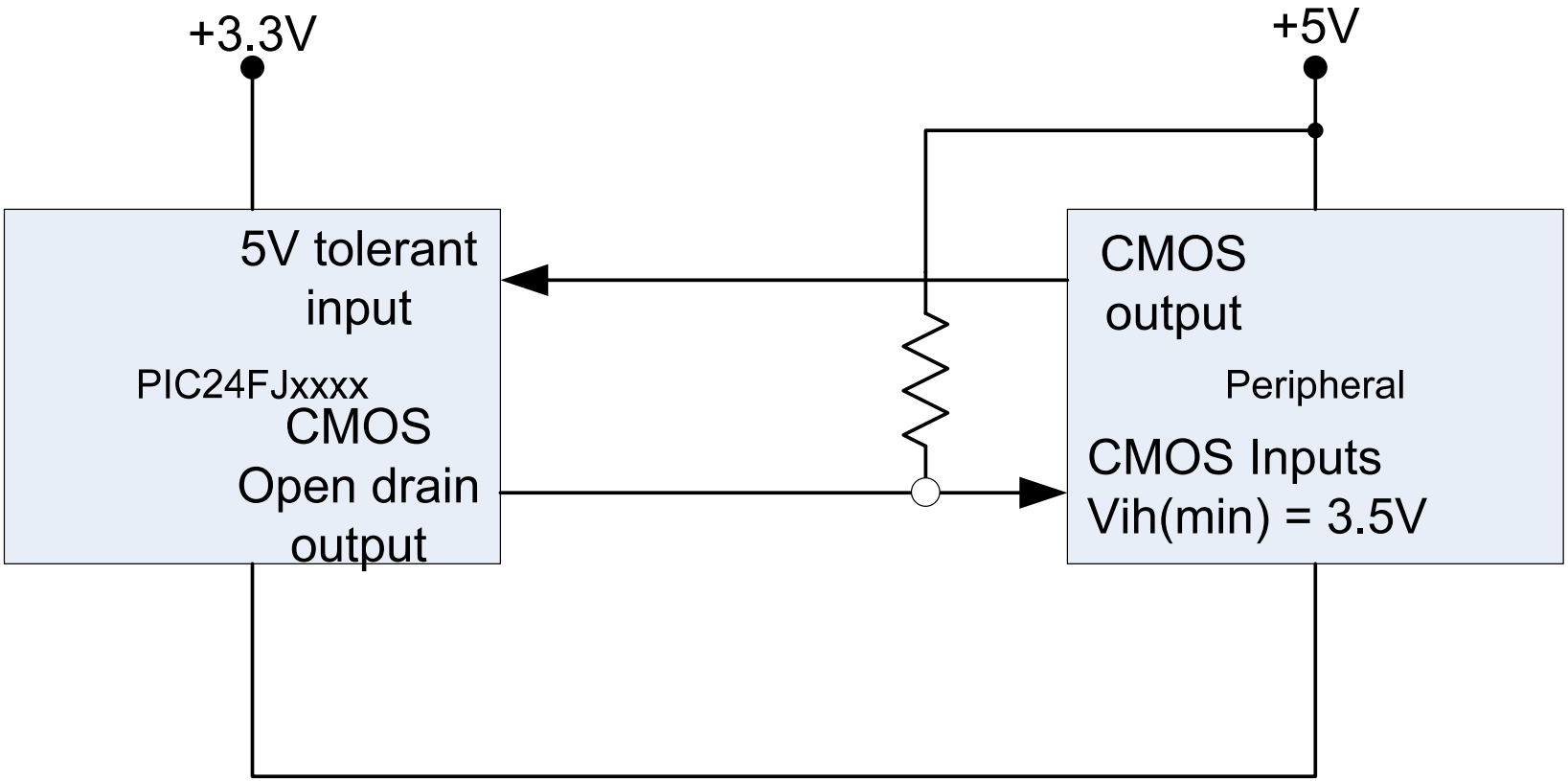
- Use RMW instruction on the LATch rather than PORTx:
  - `BSET LATA,#0`
  - `BSET LATA,#1`

I/O PORT
Open Drain Control



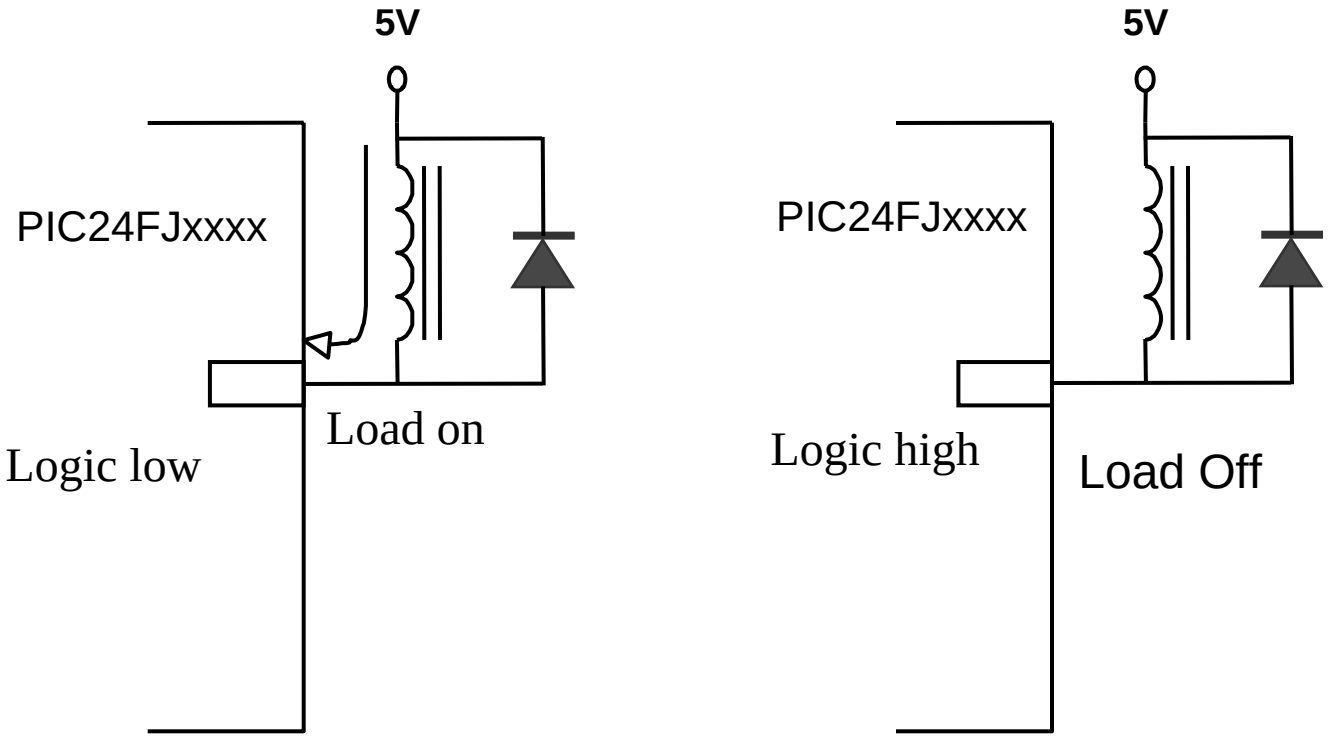
Interfacing to 5V Devices

- **5V tolerant input and Open drain configuration simplifies 5V interface**



Interfacing to 5V Devices

- **Interfacing to low impedence 5V load with open drain feature**



ADC

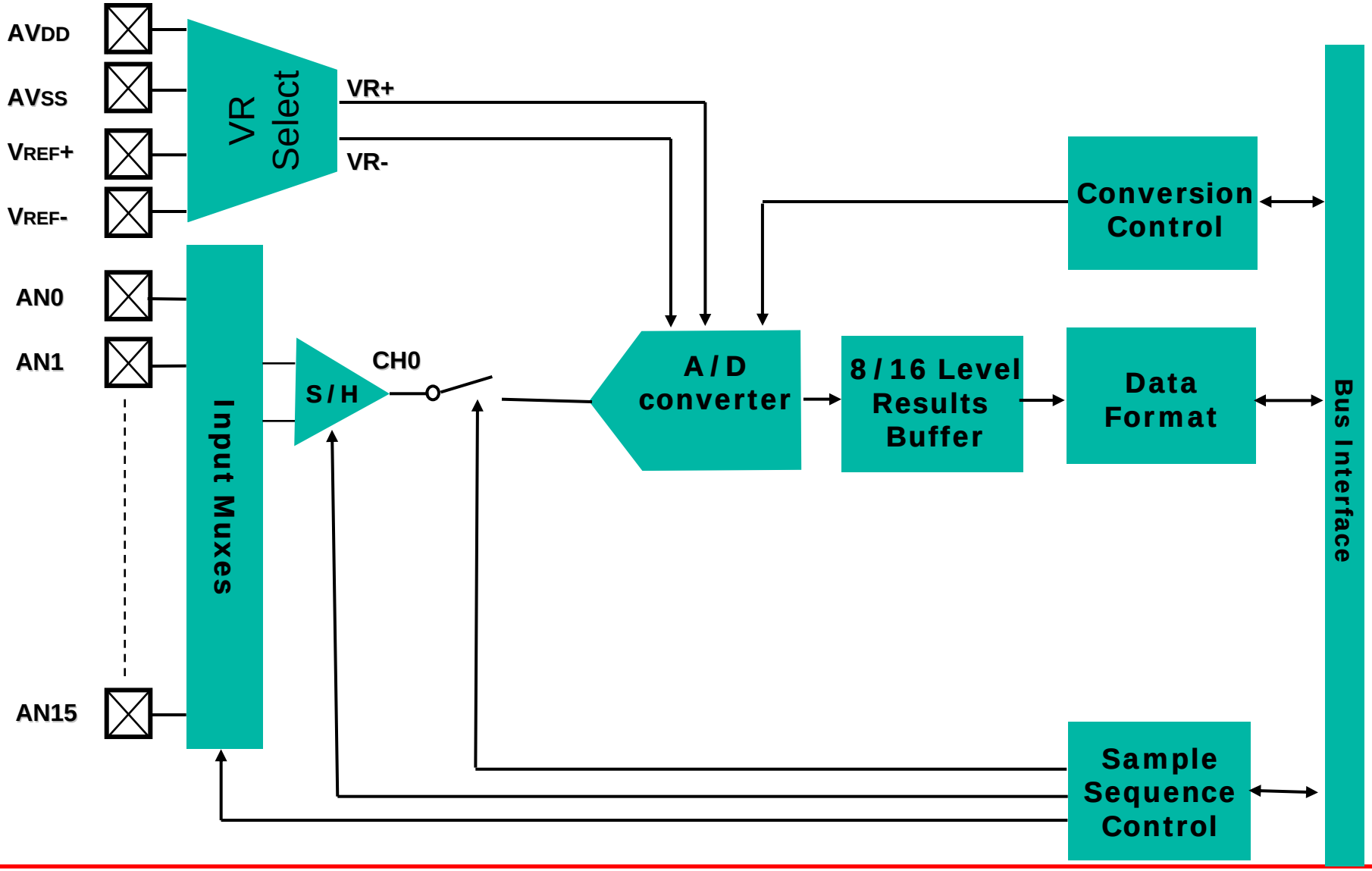
Analog to Digital Converter

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

10-bit ADC Features

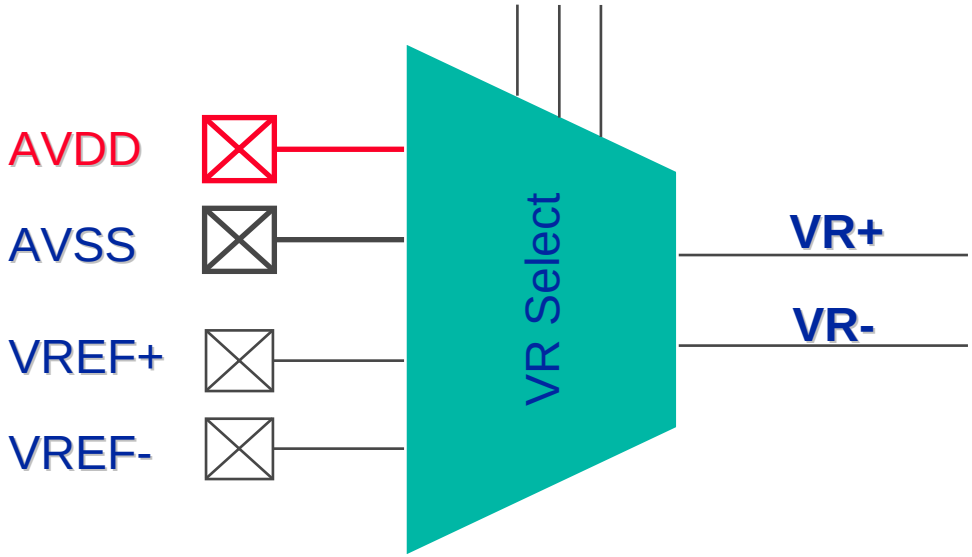
- Successive Approximation (SAR) conversion
- Conversion speeds of up to 500ksps
- Up to 16 analog input pins
- External Voltage reference pins
- Automatic Channel Scan mode
- Selectable conversion trigger source
- 16-word conversion result buffer
- Selectable Buffer Fill modes
- Four result alignment options
- Operation during CPU Sleep and Idle modes

PIC24F A/D



10-bit ADC - Block Diagram

AD1CON2<VCFG2:VCFG0>



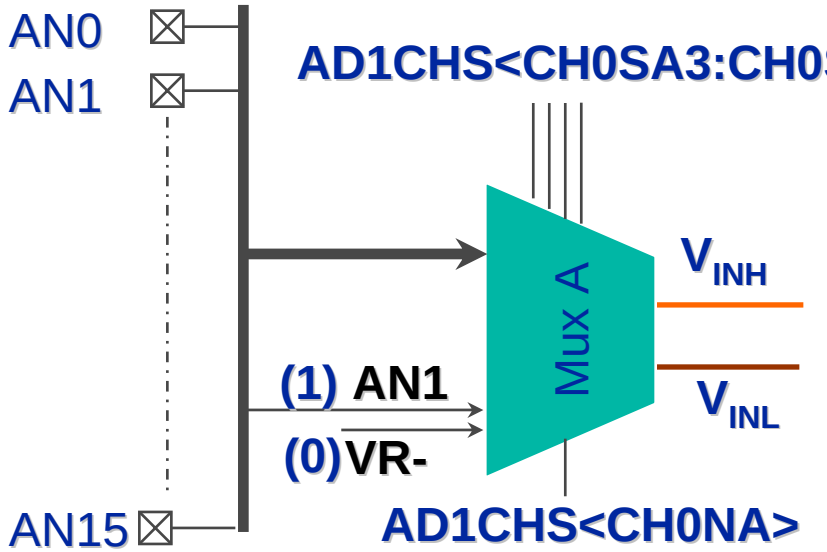
VCFG2:VCFG0	VR+	VR-
000	AVDD	AVSS
001	VREF+	AVSS
010	AVDD	VREF-
011	VREF+	VREF-
1xx	AVDD	AVSS

AD1CON2 Register

bit15							bit8
VCFG2	VCFG1	VCFG0	---	---	CSNA	---	---

bit7							bit0
BUFS	---	SMPI3	SMPI2	SMPI1	SMPI 0	BUFM	ALTS

10-bit ADC - Block Diagram



AD1CON2 Register

bit15							bit8
VCFG2	VCFG1	VCFG0	---	---	CSNA	---	---

bit7							bit0
BUFS	---	SMPI3	SMPI2	SMPI1	SMPI 0	BUFM	ALTS

AD1CSSL Register

bit15							bit0
CSSL15	CSSL14	CSSL13	.....		CSSL2	CSSL1	CSSL0

AD1CHS Register

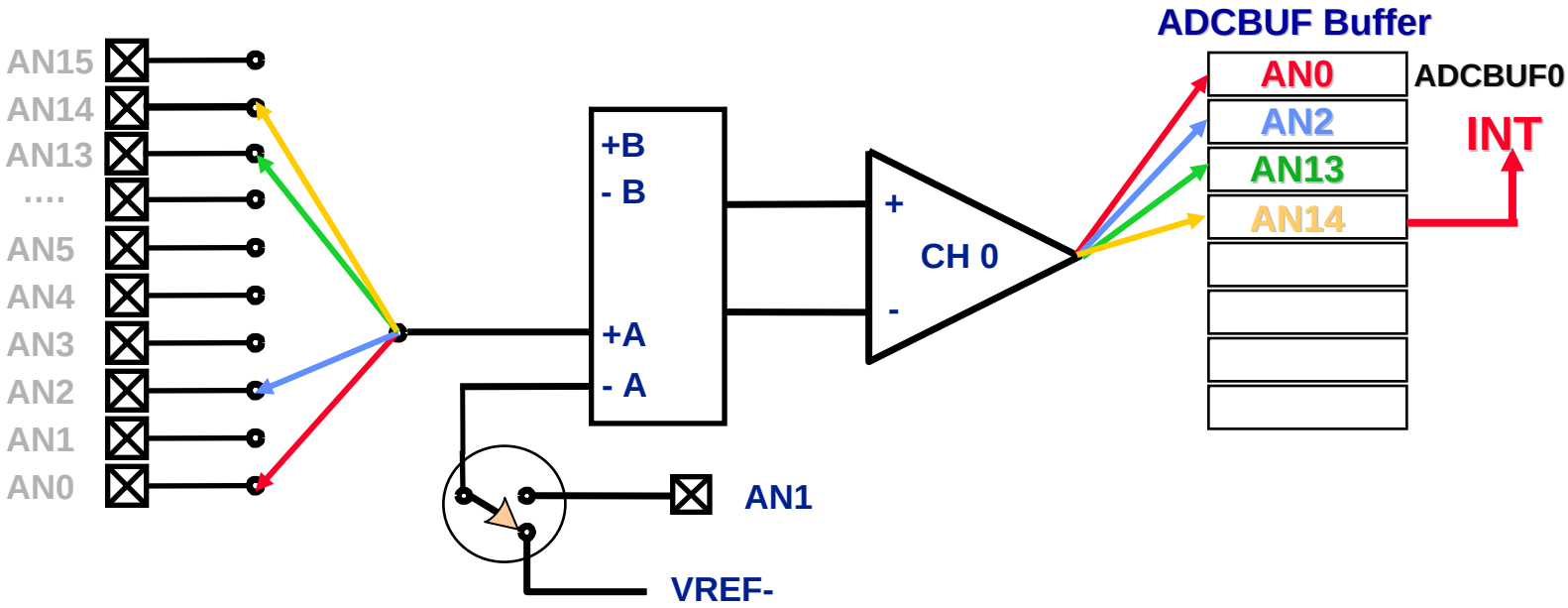
bit15							bit8
CH0NB	---	---	---	CH0SB3	CH0SB2	CH0SB1	CH0SB0

bit7							bit0
CH0NA	---	---	---	CH0SA3	CH0SA2	CH0SA1	CH0SA0

AD1PCFG Register

bit15							bit0
PCFG15	PCFG14	PCFG13	.....		PCFG2	PCFG1	PCFG0

Channel Scanning



AD1CSSL Register

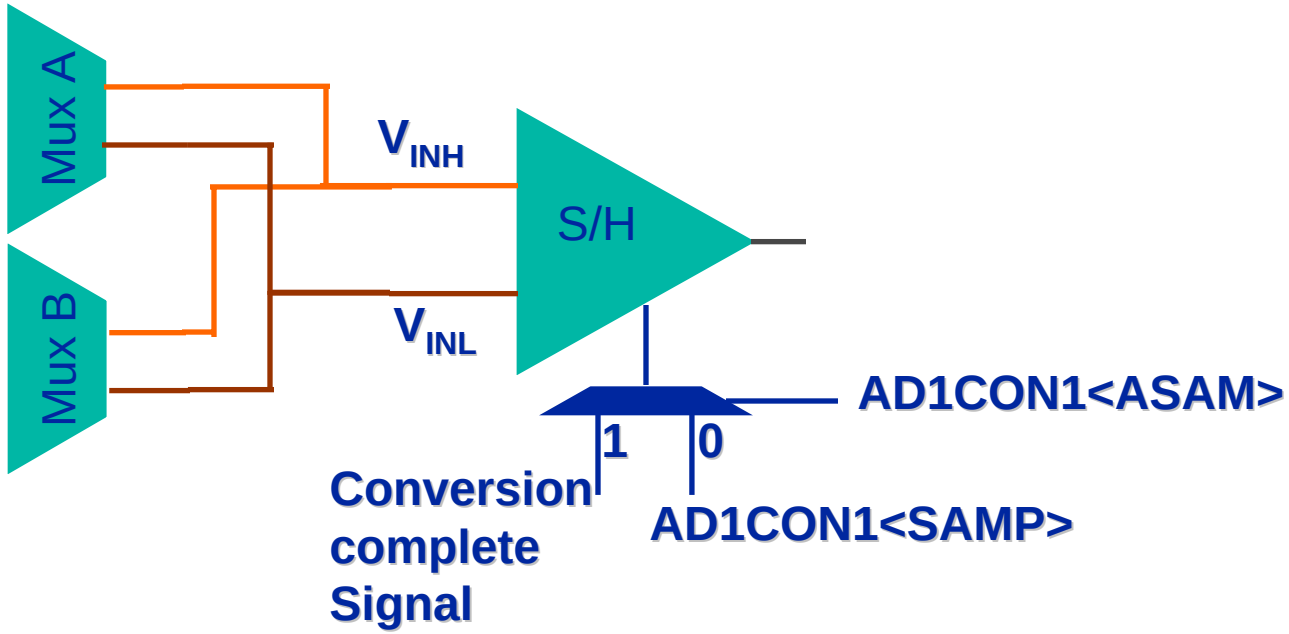
bit15							bit0
CSSL15	CSSL14	CSSL13	.....	CSSL2	CSSL1	CSSL0	

AD1CON2 Register

bit15							bit8
VCFG2	VCFG1	VCFG0	---	---	CSNA	---	---

bit7							bit0
BUFS	---	SMPI3	SMPI2	SMPI1	SMPI 0	BUFM	ALTS

10-bit ADC - Block Diagram

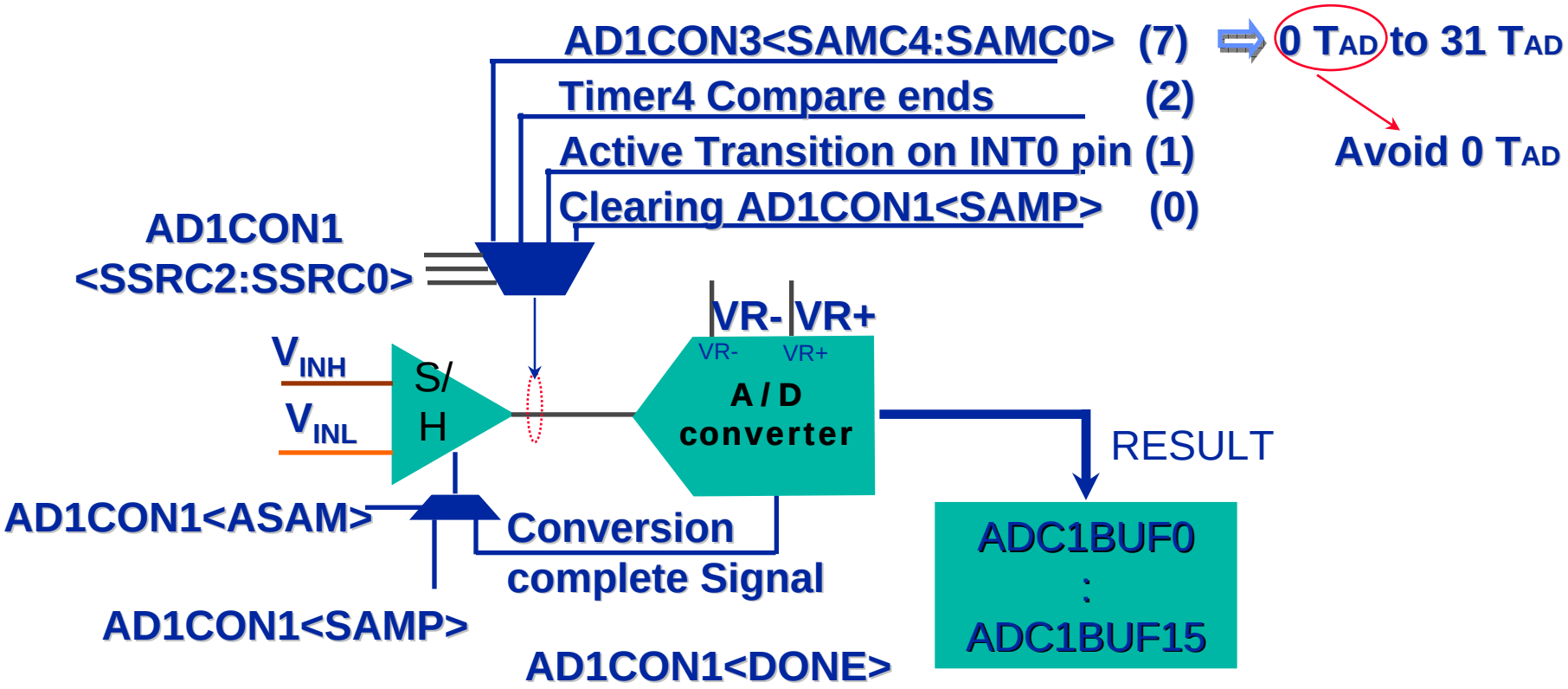


AD1CON2 Register

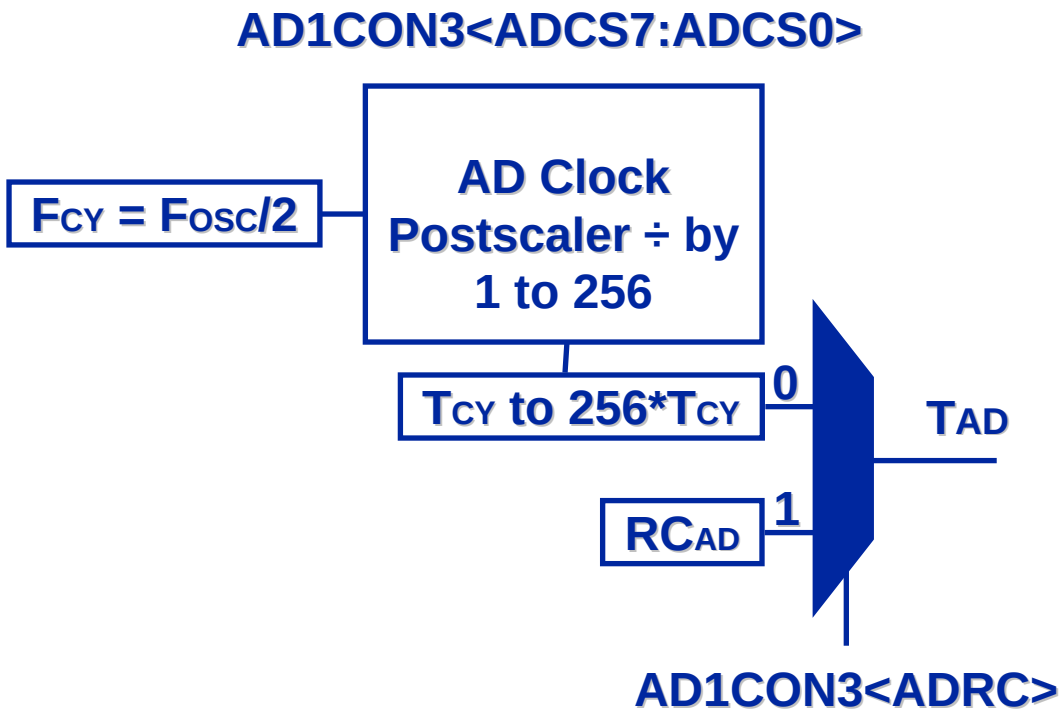
bit15							bit8
VCFG2	VCFG1	VCFG0	---	---	CSNA	---	---

bit7							bit0
BUFS	---	SMPI3	SMPI2	SMPI1	SMPI 0	BUFM	ALTS

10-bit ADC - Block Diagram



10-bit ADC - Block Diagram



10-bit ADC - Block Diagram

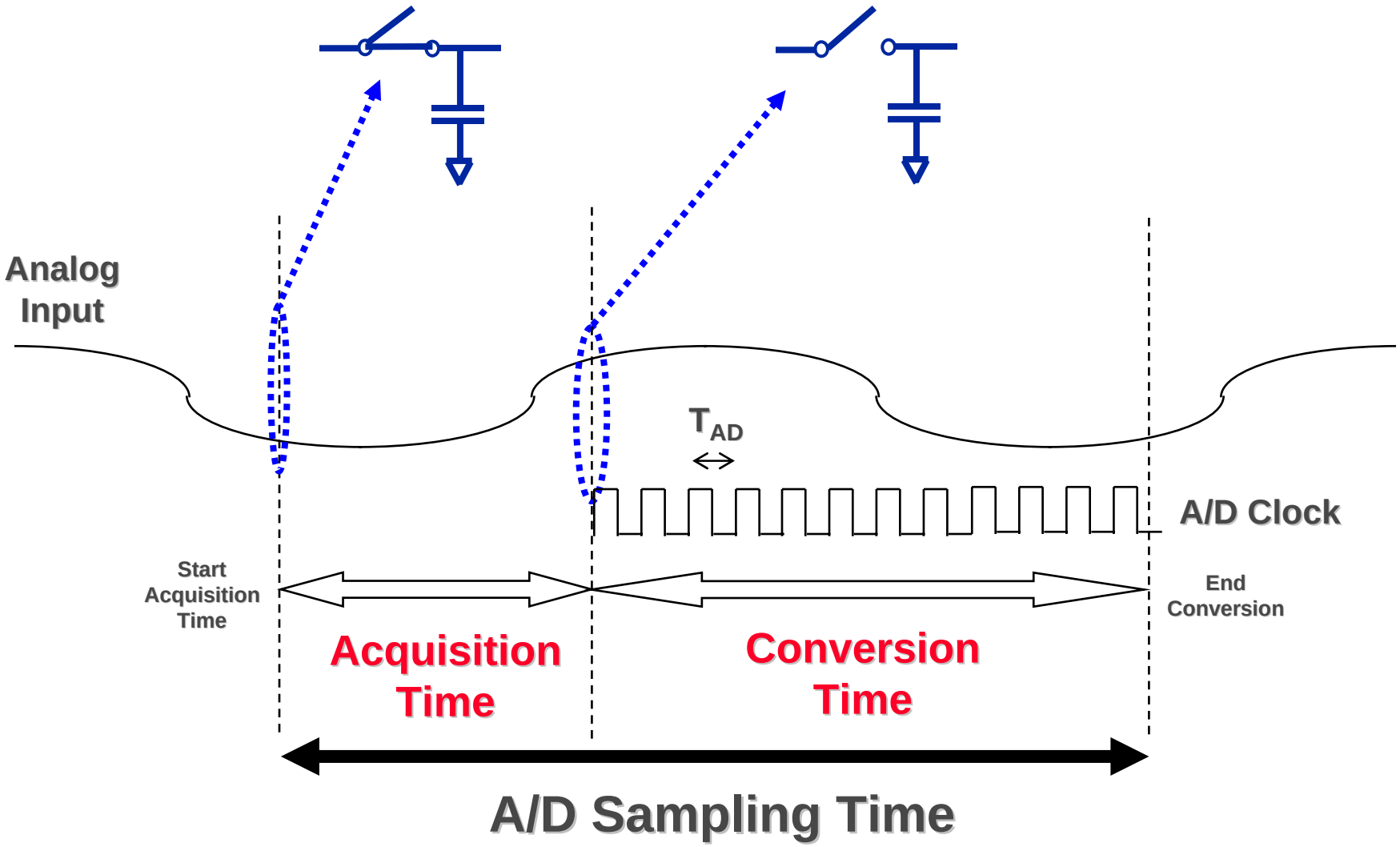
RESULT FORMAT

0000 00dd dddd dddd
ssss sssd dddd dddd
dddd dddd dd00 0000
sddd dddd dd00 0000

AD1CON1<FORM1:FORM0>

AD1CON2<SMPI3:SMPI0>

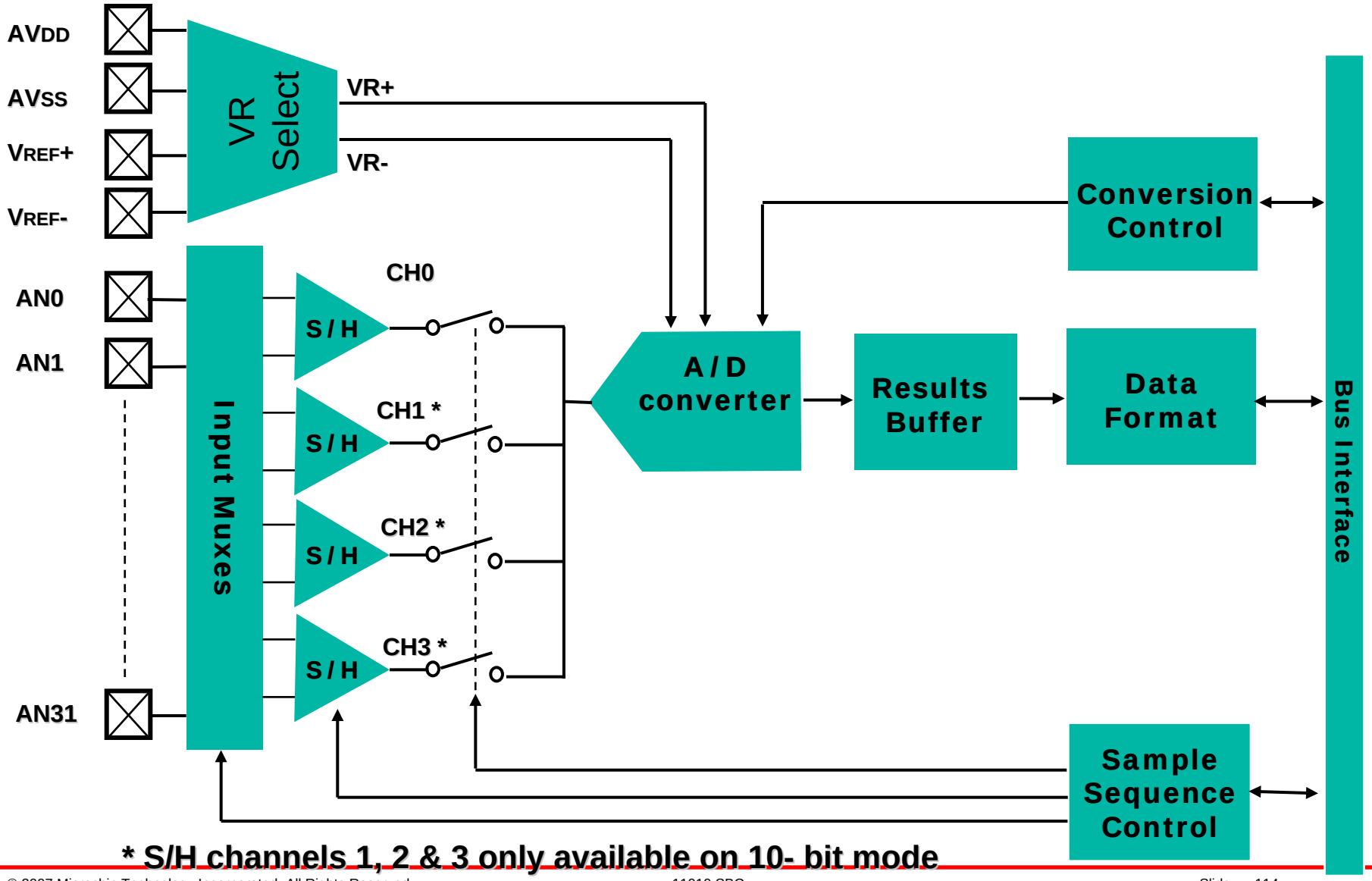
Acquisition/Conversion Timing



12-bit A/D Converter

- **200 K Samples / Sec Sampling Rate**
- **Up to 16 input channels**
- **16 ADC Result buffer to store converted data**
- **External VREF+ and VREF-**
- **Analog Input Range: 0 to 5V (VREF- to VREF+)**
- **Multiple Trigger sources for Start of Conversion**
  - Software Trigger
  - Motor Control PWM
  - Timer 3
  - External Interrupt
  - Automatic conversion on internal time out

PIC24H/30/33 A/D



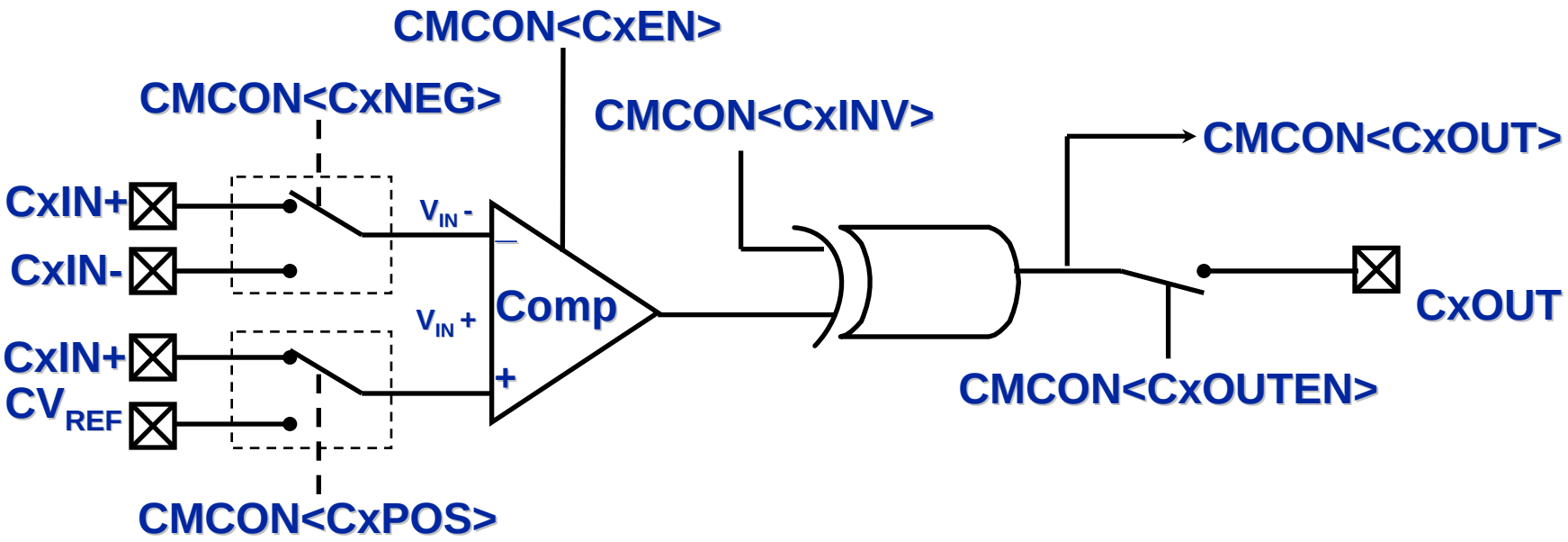
*** S/H channels 1, 2 & 3 only available on 10- bit mode**

Lab 4: Working with ADC

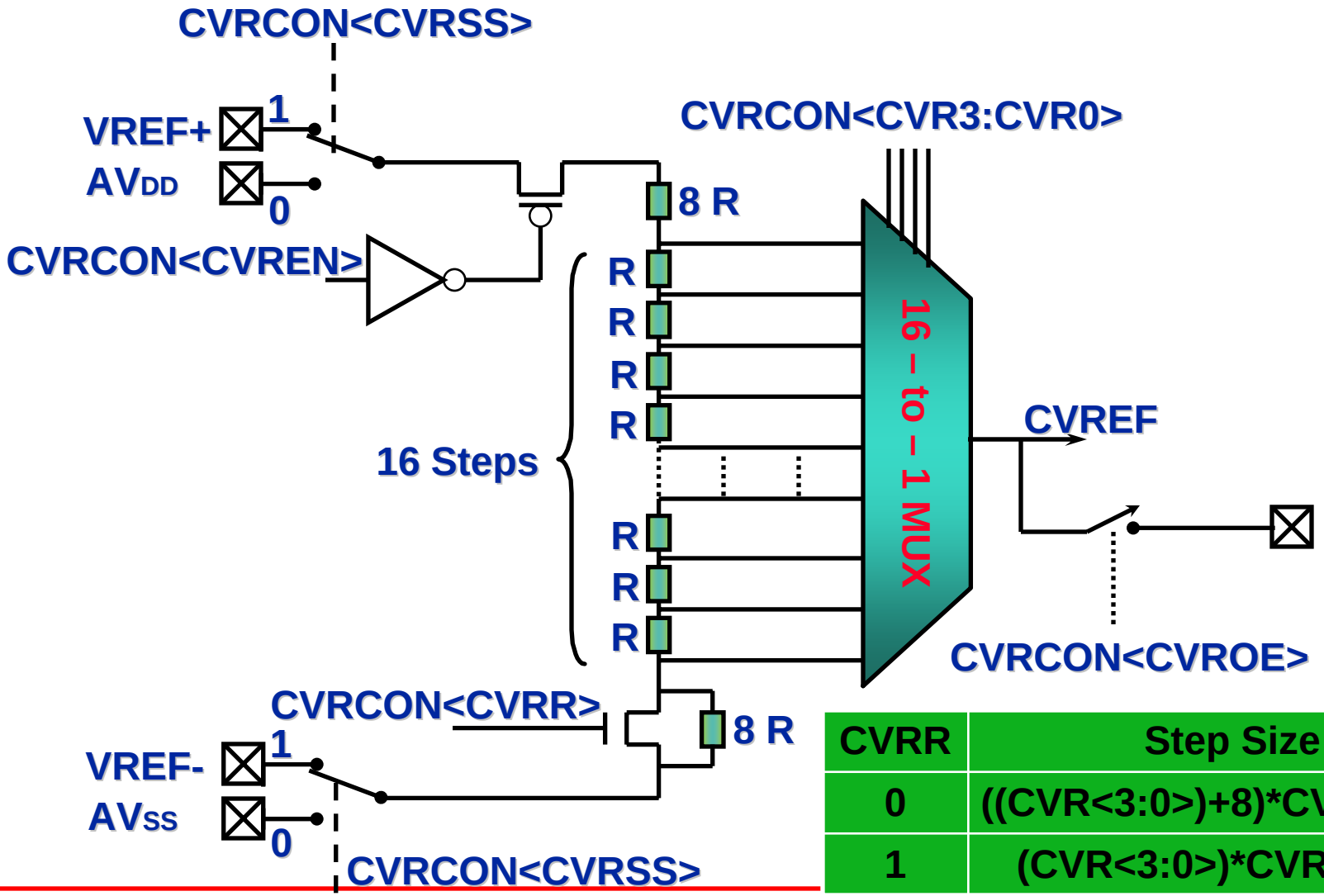
- **Goal**
  - To configure ADC
  - To configure I/O ports
  - To read ADC and o/p on to LEDs
- **To Do:**
  - Look into the handout provided
- **Expected Result:**
  - The average pot value is displayed on LEDs

Analog Comparator Module

Analog Comparator Block Diagram



Analog Comparator
Voltage Reference Generator



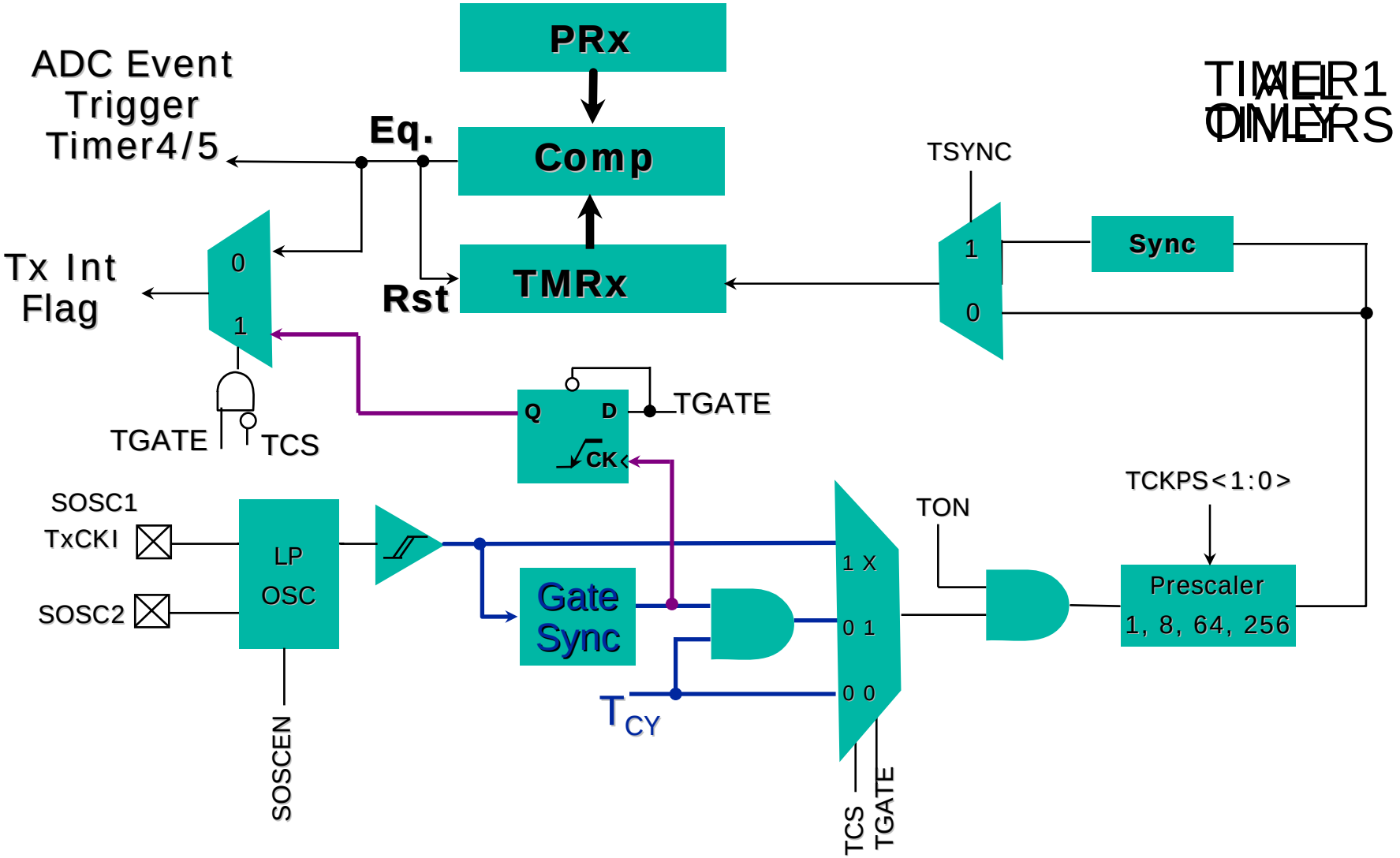
CVRR	Step Size
0	((CVR<3:0>)+8)*CVRSrc/32
1	(CVR<3:0>)*CVRSrc/24

Timers

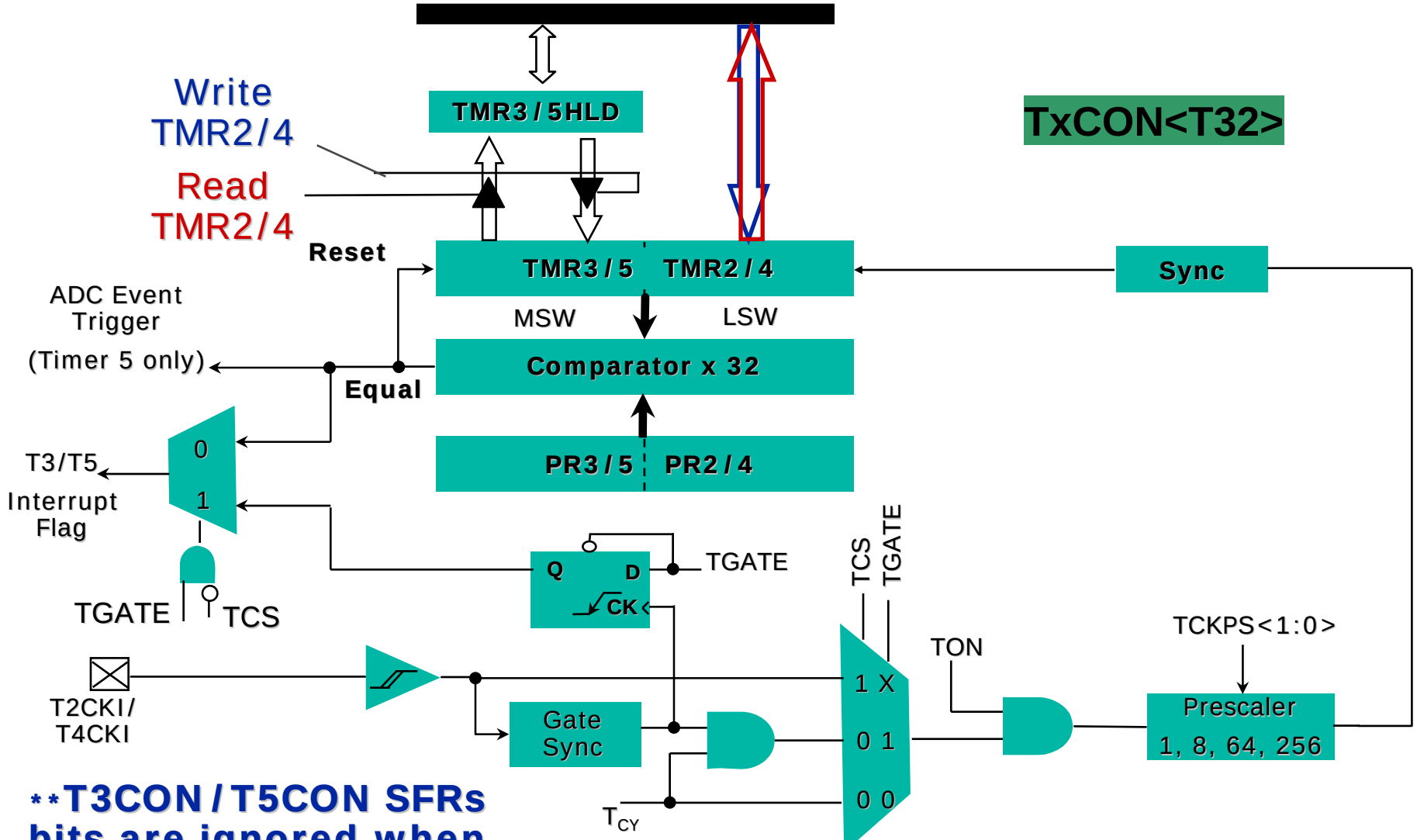
Timer Features

- **Five 16-bit General Purpose Timers / Counters**
  - Similar functionality between all 5 timers
    - **Asynchronous counter feature only in Timer1**
- **Period Registers for each**
  - Interrupt generation on match
  - Reset on match
- **Gated Timer operation on each**
  - Interrupt on falling edge of gate
- **Four of these timers (Timer 2+3 & 4+5) can make two 32-bit timers/counters**

Timer x



32-bit Timer (T2+T3/T4+T5)



**T3CON/T5CON SFRs bits are ignored when in 32-bit timer mode

Lab 5: Working with a 32-bit Timer

- **Goal**
  - Understand working of Timers in 32-bit mode
  - Configure the Timer 2/3 pair for 32-bit mode
  - Implement a stop watch
- **To Do:**
  - Look into the Hand out provided
- **Expected Result:**
  - Press the Switch S3 to start timer
  - Again press the Switch S3 to stop timer and LCD displays the Time elapsed between the start and stop

Input Capture

Why Do We Need IC Module

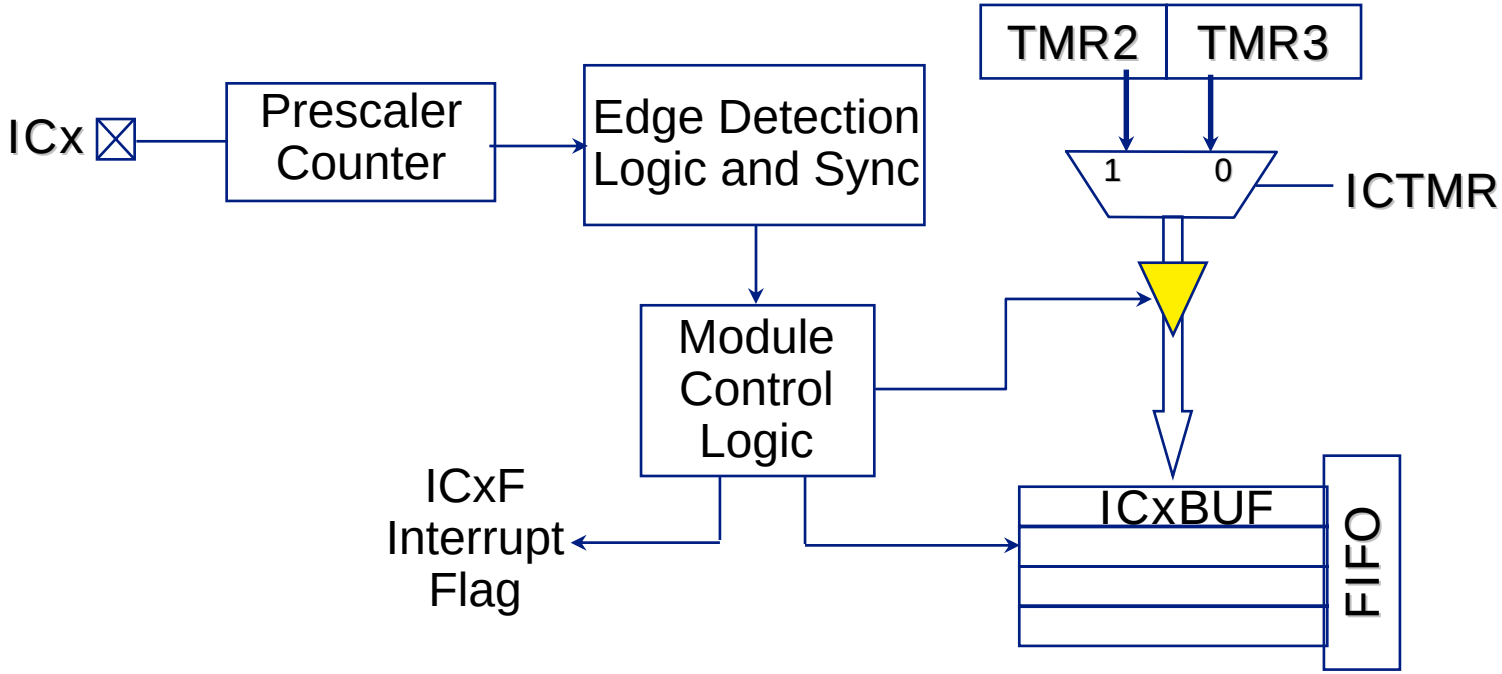
- Consider an Example to measure the Pulse width of the signal
- The IC Module should be configured to
  - Capture on Every Edge
  - Interrupt on every Second Event
- Then the Pulse width of the signal can be found by using the formula:

$$\text{Pulse Width} = \frac{\text{Buffer 1} - \text{Buffer 0}}{\text{Timer running Frequency}}$$

Input Capture

- **Input Capture**
  - Up to five Input Capture Channels
  - Captures 16-bit timer value
  - Tcy Resolution
  - Timer 2 or Timer 3 as time base

Input Capture



ICxCON<ICI1:ICI0>

11: Interrupt on every fourth capture event
10: Interrupt on every third capture event
01: Interrupt on every second capture event
00: Interrupt on every capture event

ICxCON<ICM2:ICM0>

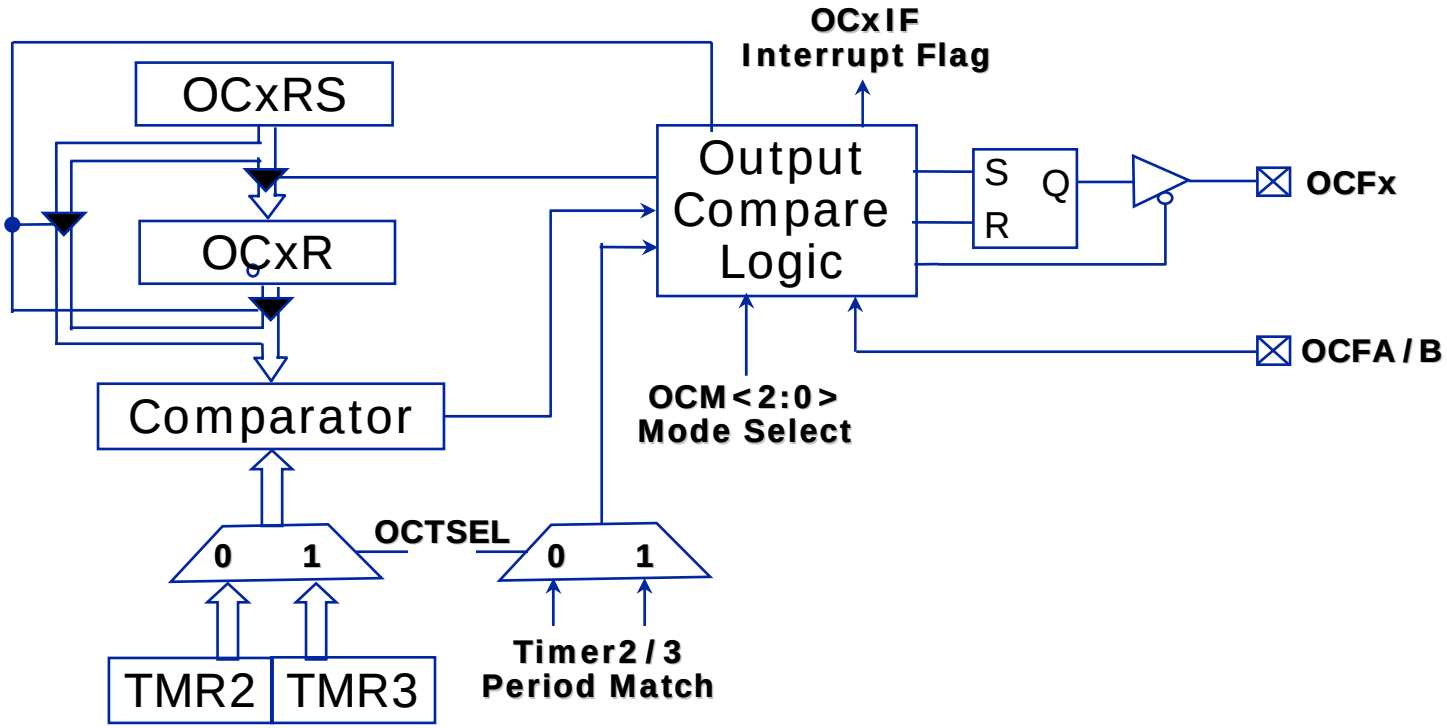
111: Input Capture functions just as an interrupt pin while in SLEEP or IDLE mode
101: Capture on every sixteenth rising edge
100: Capture on every forth rising edge
011: Capture on every rising edge
010: Capture on every falling edge
001: Capture on every edge (both rising and falling)
000: Capture module is turned off

Output Compare & PWM

Output Compare & PWM

- **Up to 8 Output Compare / PWM Channels**
- **Timer 2 or Timer 3 as time base**
- **16-bit Compare**
- **Minimum 1Tcy pulse width allowed**
- **Several Compare Modes:**
  - Set, Reset or Toggle Pin
  - Single Pulse, Continuous pulse
  - PWM

Output Compare



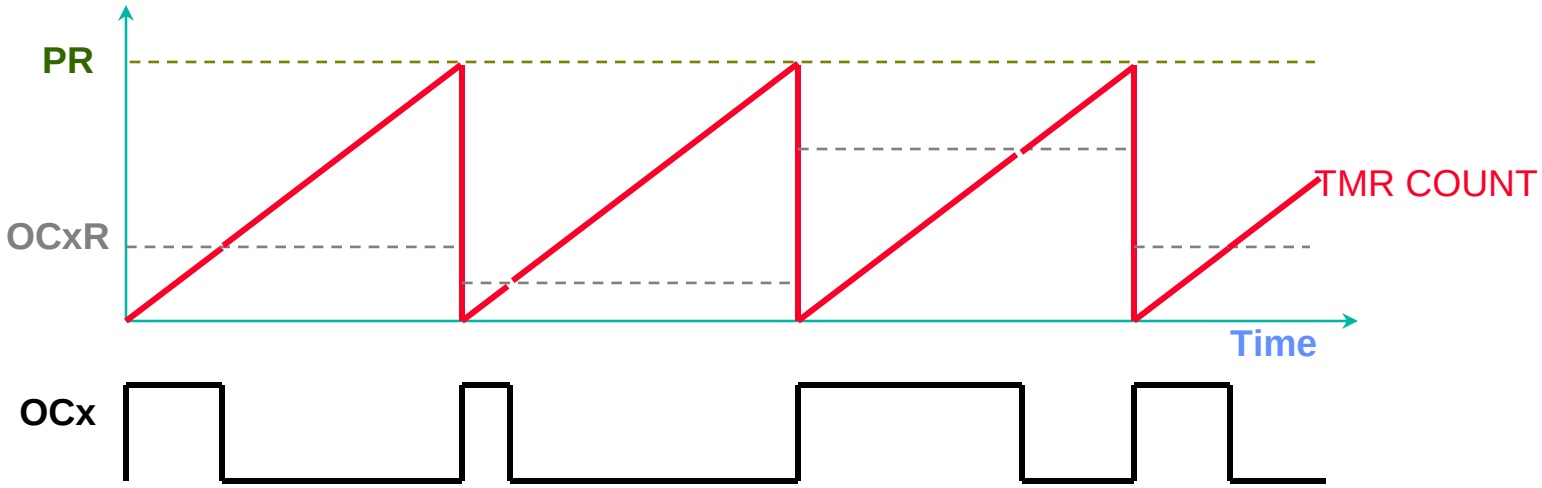
OCxCON<OCM2:OCM0>

111: PWM mode with Fault pin option
110: PWM mode without Fault pin option
101: Initialize OCx-low, generate continuous signal
100: Initialize OCx-low, generate single pulse
011: Toggle OCx on every compare match
010: Initialize OCx-High, pull OCx low on compare match
001: Initialize OCx-low, pull OCx high on compare match
000: Compare module is turned off

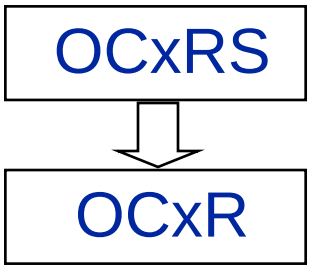
Output Compare Module

- **PWM mode**
  - 16-bit glitch-less (double buffered) PWM output
  - Full range of 0 to 100% duty cycle
  - Wide frequency range
  - Selectable PWM shutdown on fault detection
    - **This is an Asynchronous shutdown**

OC PWM Mode



- **On timer period match**
  - OCx set
  - OCxRS copied to OCxR
- **On OCxR match**
  - OCx clear



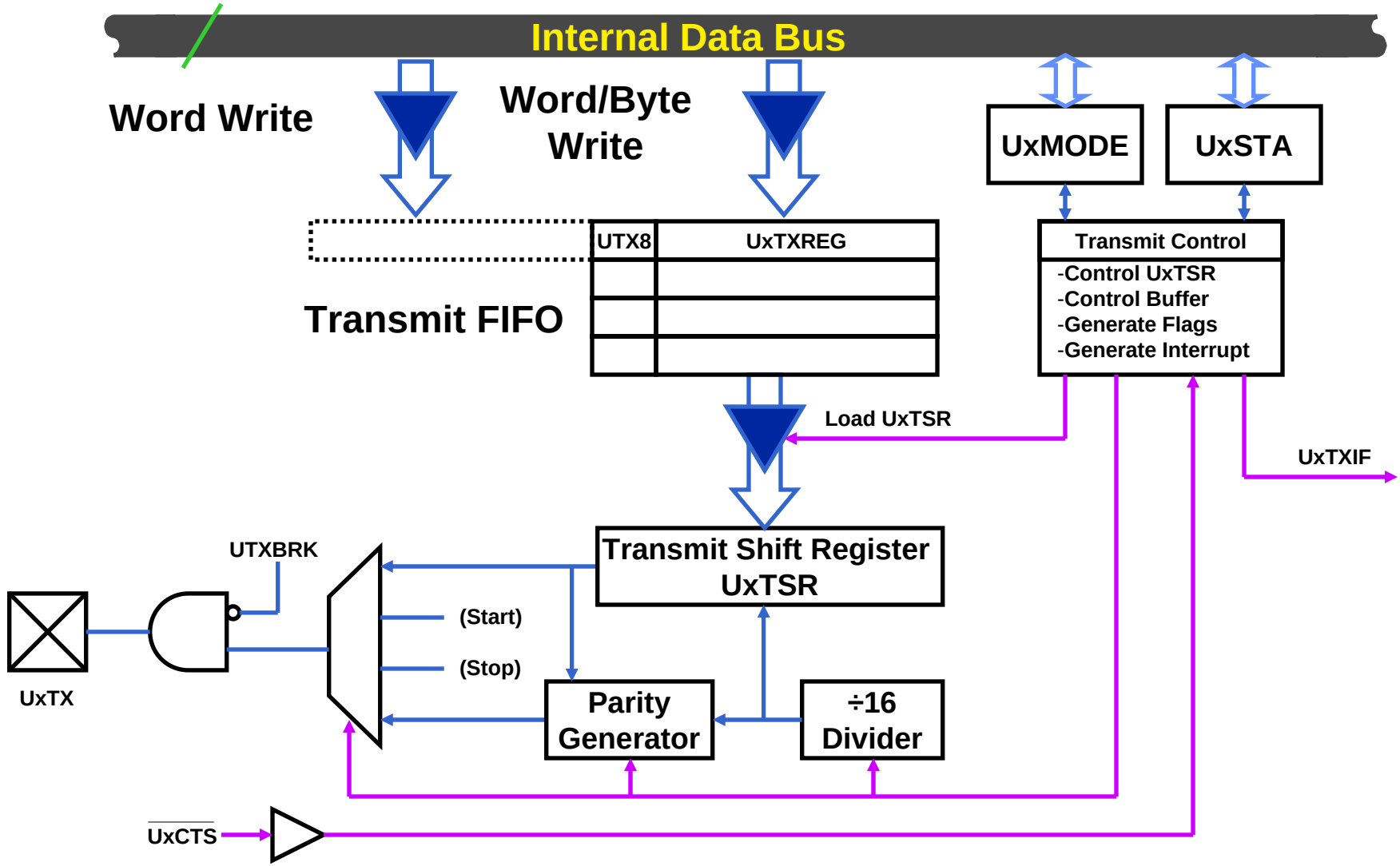
UART

University of Microchip UofM MASTERs 2007

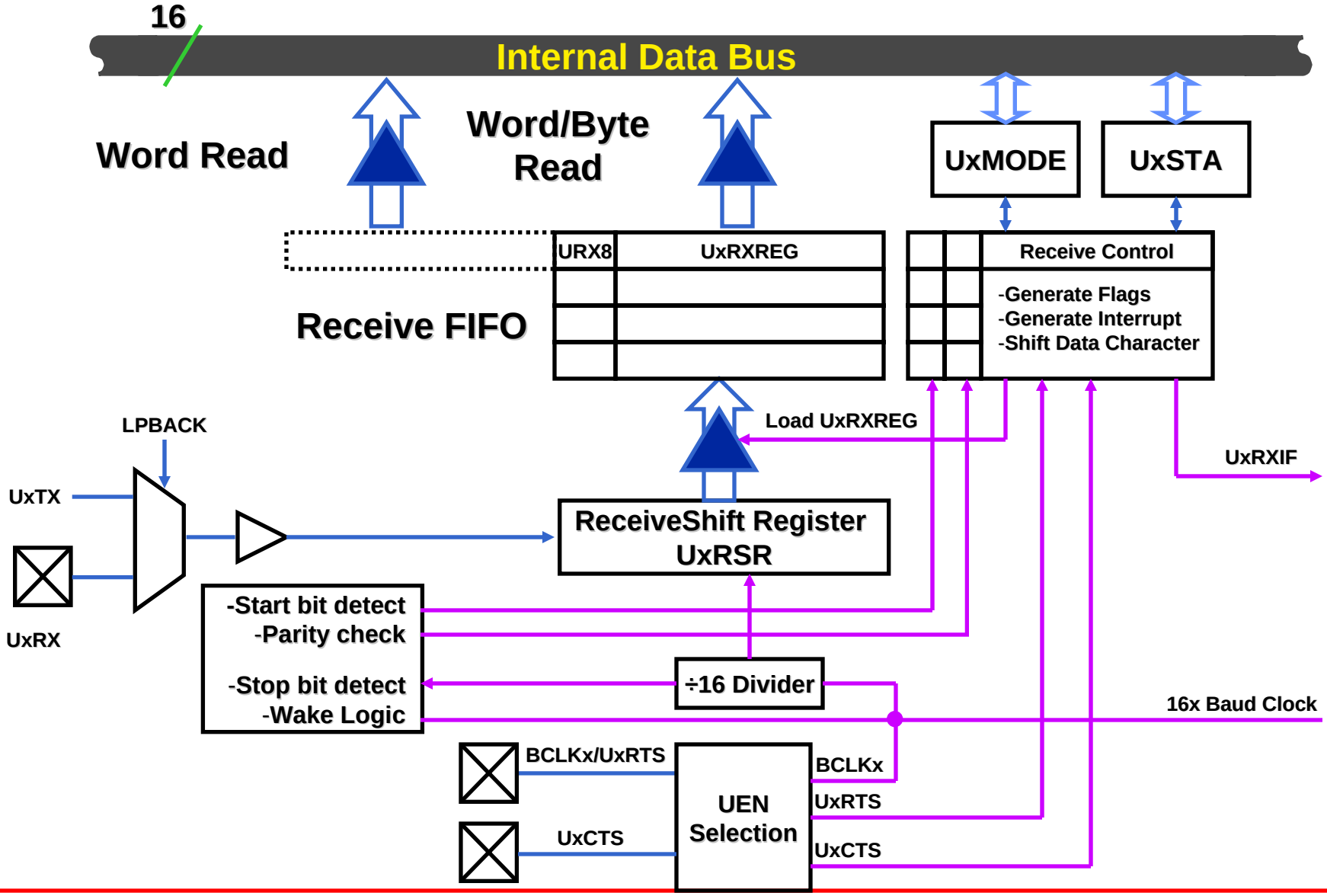
UART Features

- 4-deep FIFO Transmit Data Buffer
- 4-deep FIFO Receive Data Buffer
- Parity, Framing and Buffer Overrun Error Detection
- Support for 9-bit mode with Address Detect (9th bit = 1)
- Transmit and Receive Interrupts
- Loopback mode for Diagnostic Support
- LIN 1.2 Protocol Support
- IrDA® Support

UART Module – Transmit

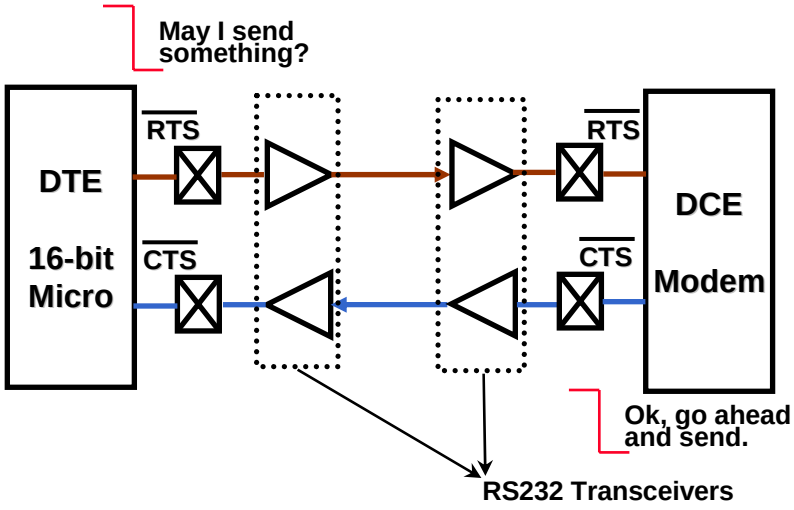


UART Module – Receive

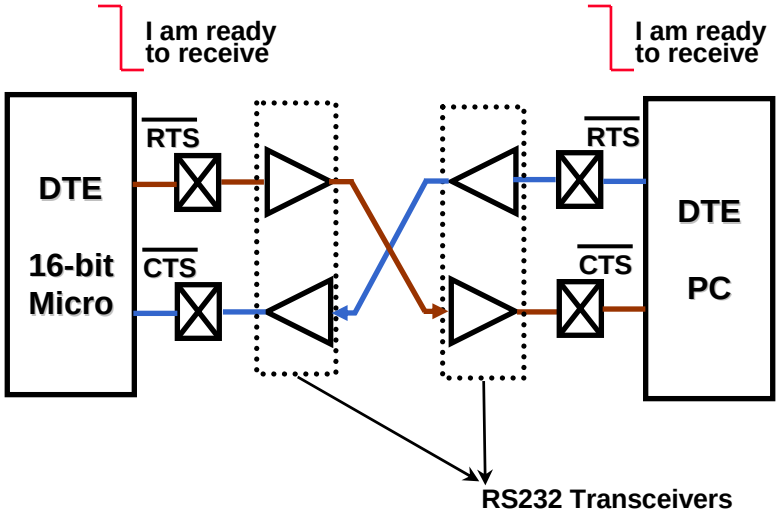


UART with CTS and RTS

- **Controlled Transmission and Reception**
  - Simplex Mode
  - Flow Control Mode



Simplex Mode



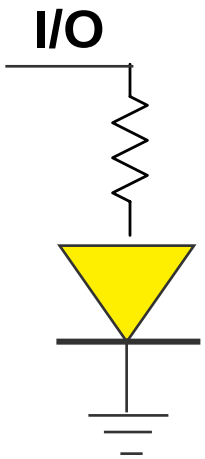
Flow Control Mode

UART - IrDA® Support

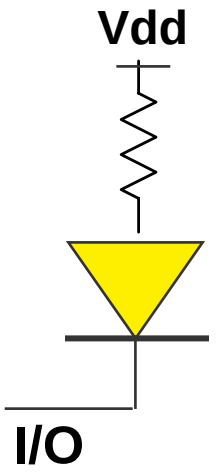
- **Built-in IrDA Encoder and Decoder**
  - The IrDA Encoder and Decoder can be enabled by setting **UxMODE<IREN>** bit
- **16x Baud Clock is generated to support external IrDA Encoder and Decoder**
  - 16x Baud Clock is generated by setting **UxMODE<UEN1:UEN0>** to '11'
  - The generated Baud Clock is available on the pin **BCLKx**

Optical Communication

- **Two ways to connect LED**

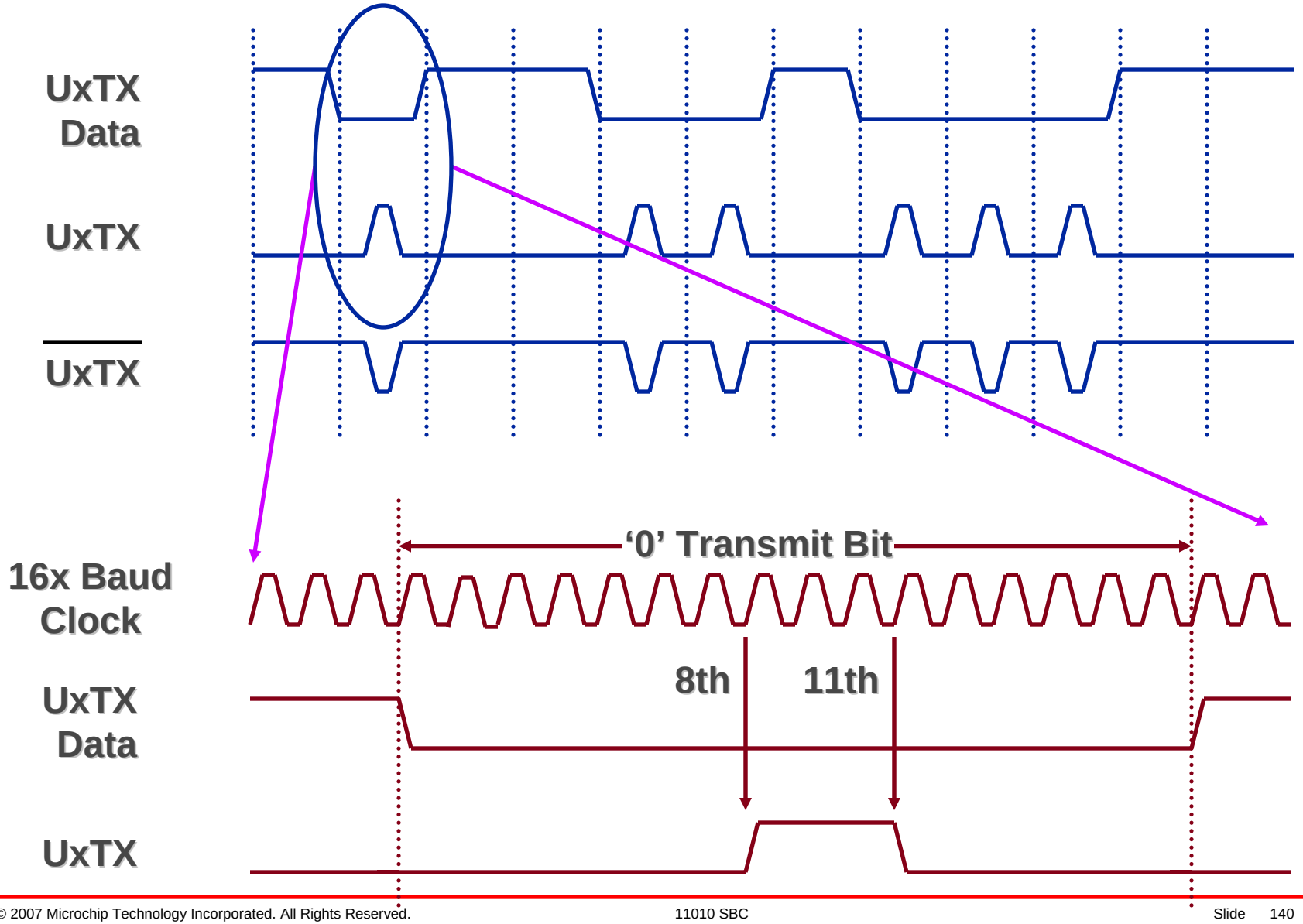


Logic High = LED On
Logic Low = LED off



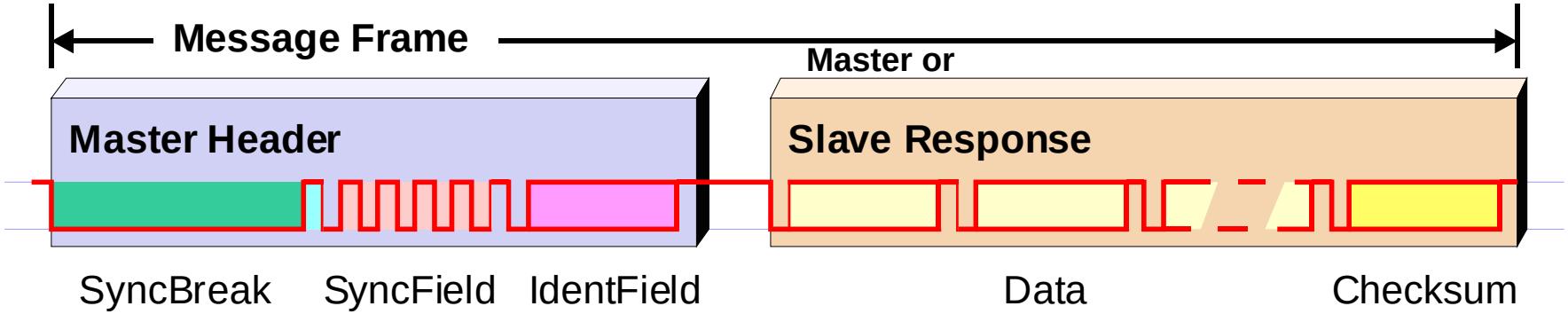
Logic Low = LED On
Logic High = LED off

IrDA® Mode Bit Encoding



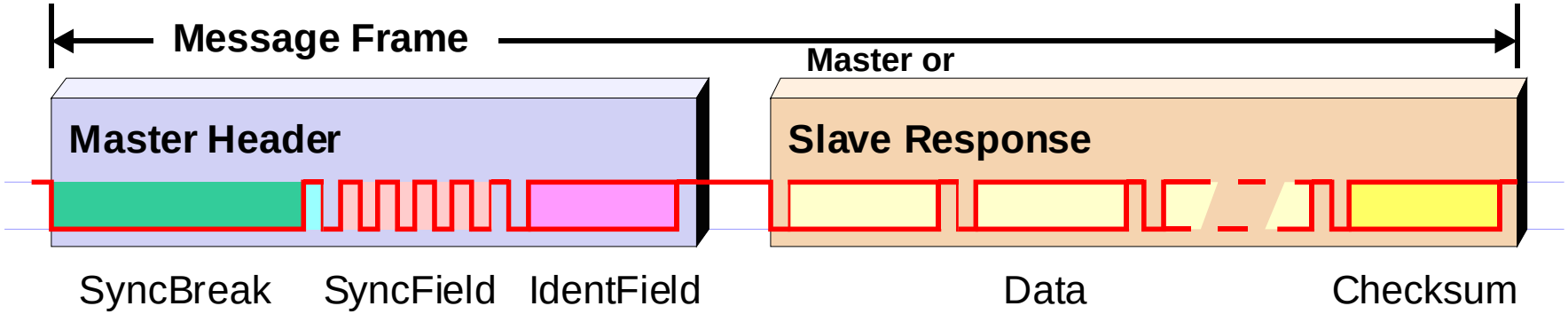
UART - LIN Support

- **Transmission of Break Characters**
  - A Break character can be transmitted by setting the **UxSTA<UTXBRK>** bit for at least 13 bit times
- **Autobaud Detection**



LIN - Protocol

- UxSTA<UTXBRK> **Enables Transmitting Break condition on the bus (Drive low TX pin for 13-Bit Time)**
- UxMODE<ABAUD> **Enables to measure the baud-rate using Input Capture unit during SYNCH Field (0x55)**



LAB 6: Working with UART

- **Goal:**
  - Understand Configuration of UART module
  - Understand Transmit and Receive interrupts
  - Write a software to Transmit and Receive data using UART
- **To Do:**
  - Look into the Hand out provided
- **Expected Result:**
  - LED D3 flashes at different rates to indicate lower CPU time spent handling UART data using FIFO buffer

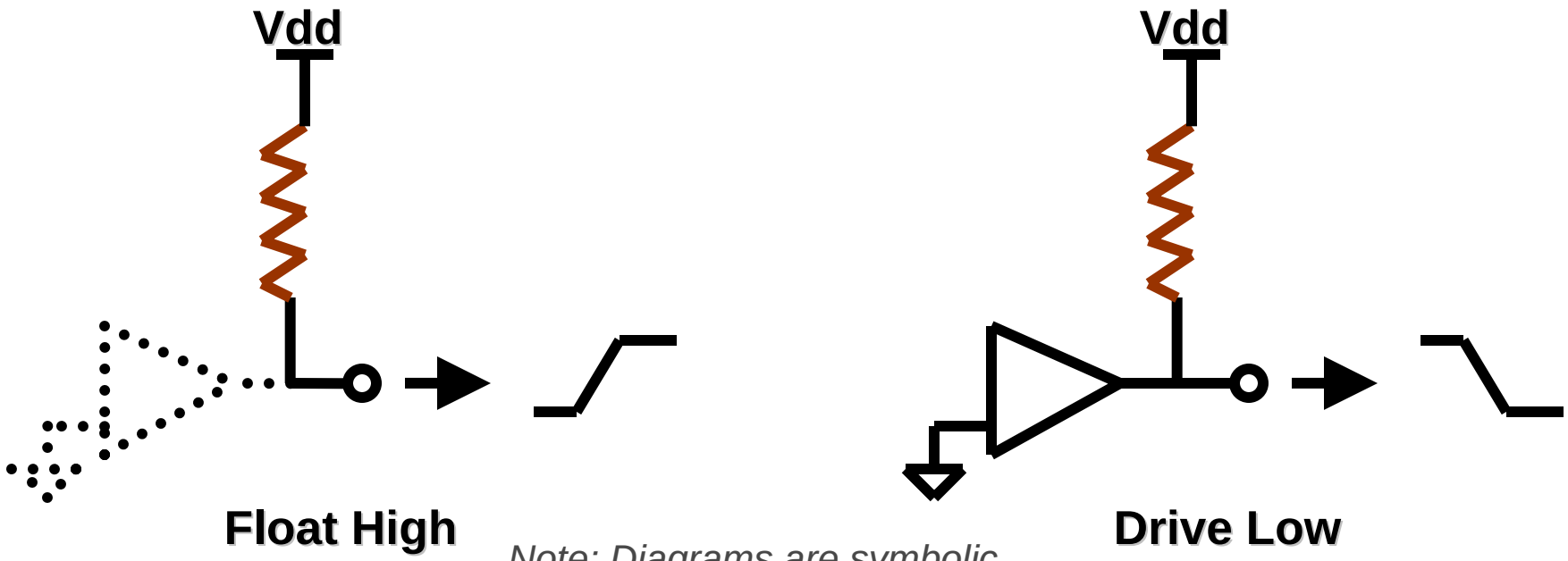
I²C™

I²C™ Overview

- **Synchronous 2-wire communication**
- **Master-Slave protocol**
- **Serial interface (Half-duplex):**
  - SDA: Data line to & from the master
  - SCL: Clock line, master controlled

Signal Connection

- **Serial interface (Half-duplex):**
  - SDA: Data line to & from the master
  - SCL: Clock line, master controlled
- **Wired-AND connection**
  - **Float High (logic 1)**
  - **Drive Low (logic 0)**

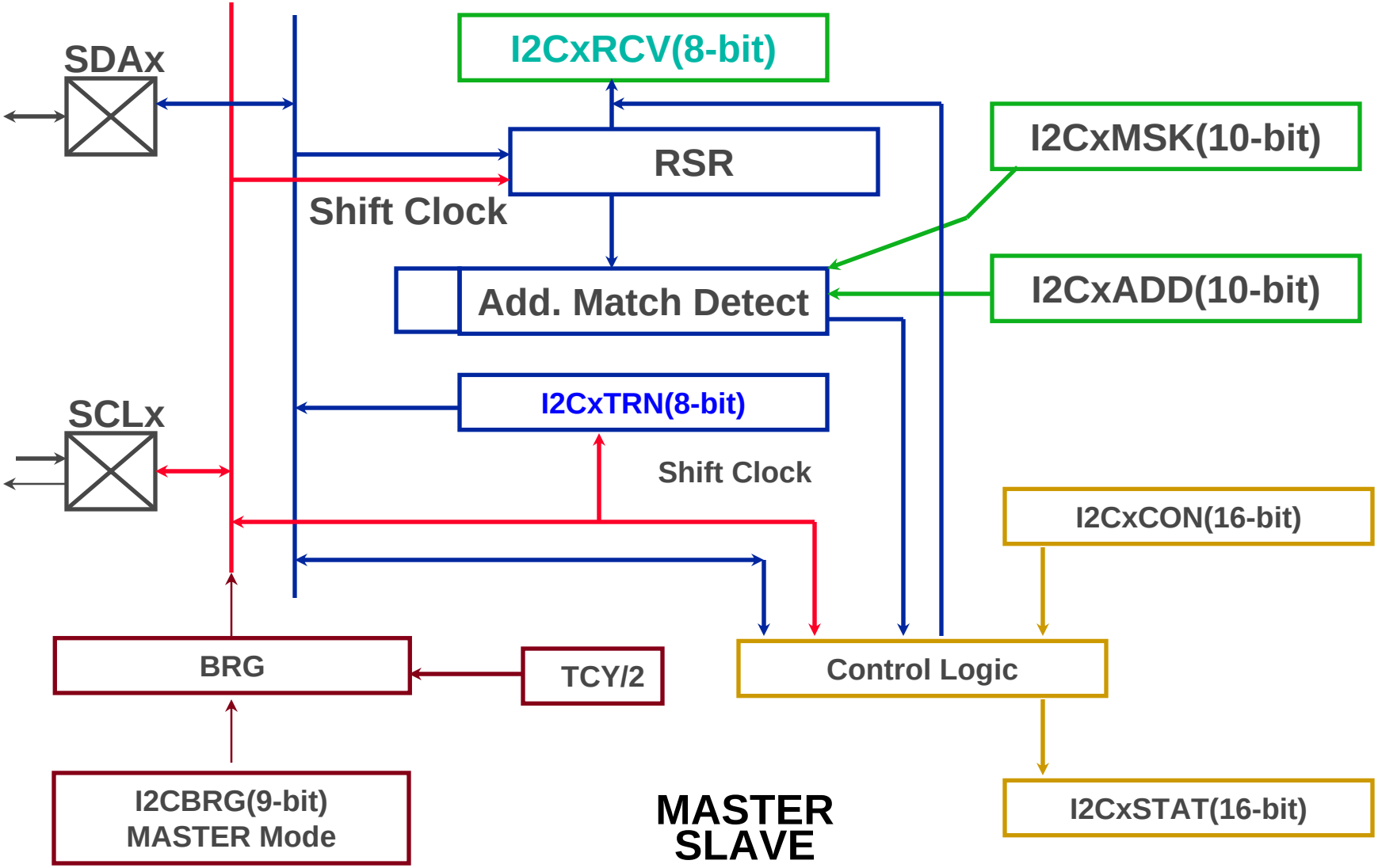


Note: Diagrams are symbolic

I²C™

- Master mode and Slave mode
- 7- and 10-bit address Slave mode
- Clock Stretching: Handshake Mechanism for suspend and resume operation
- Multi-Master operation: Detects bus collision and arbitrate accordingly
- General Call support
- Address Masking

I²C™



Address Masking

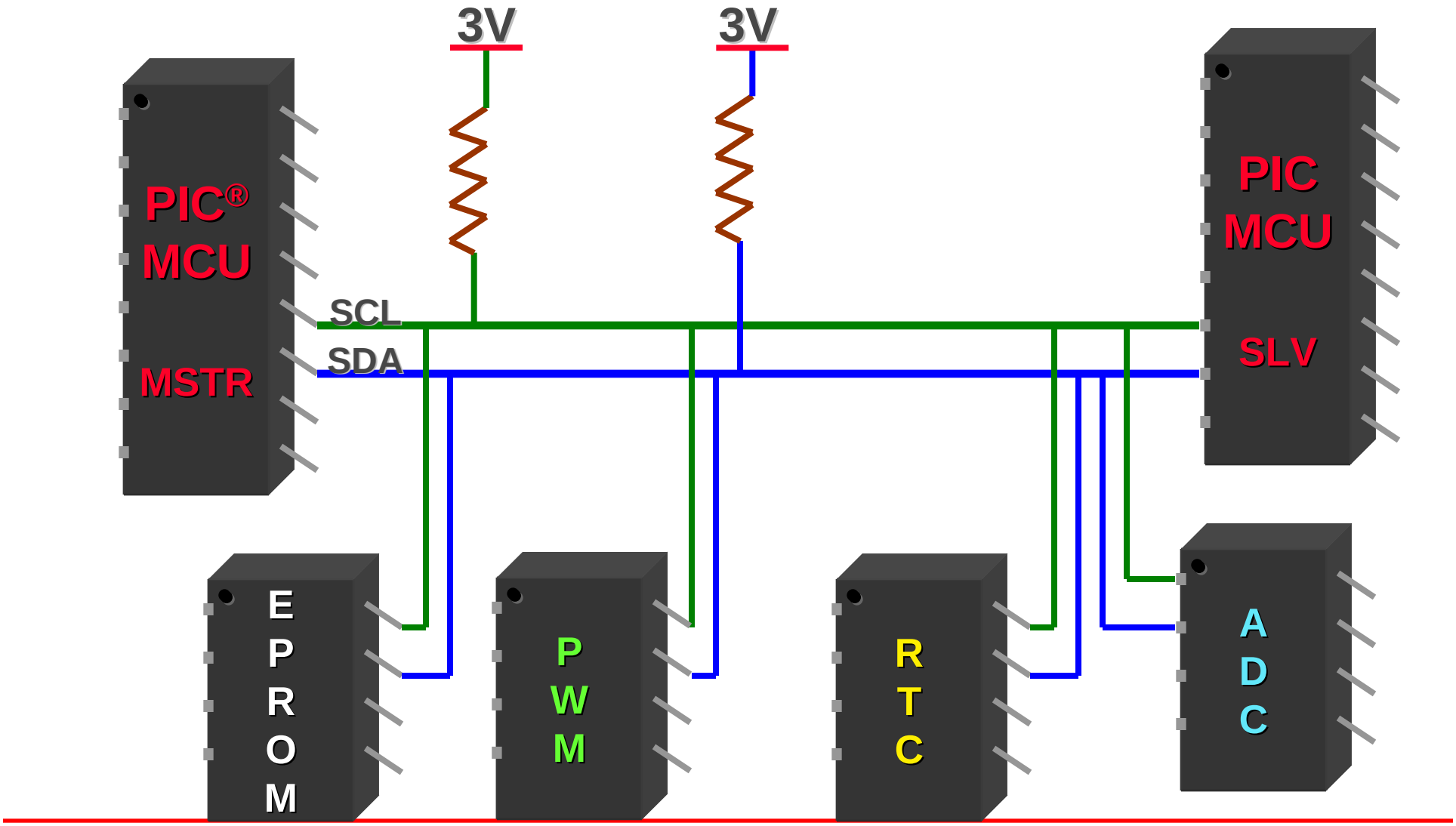
What is it?

- It allows a PIC® MCU to ACK more than one address.

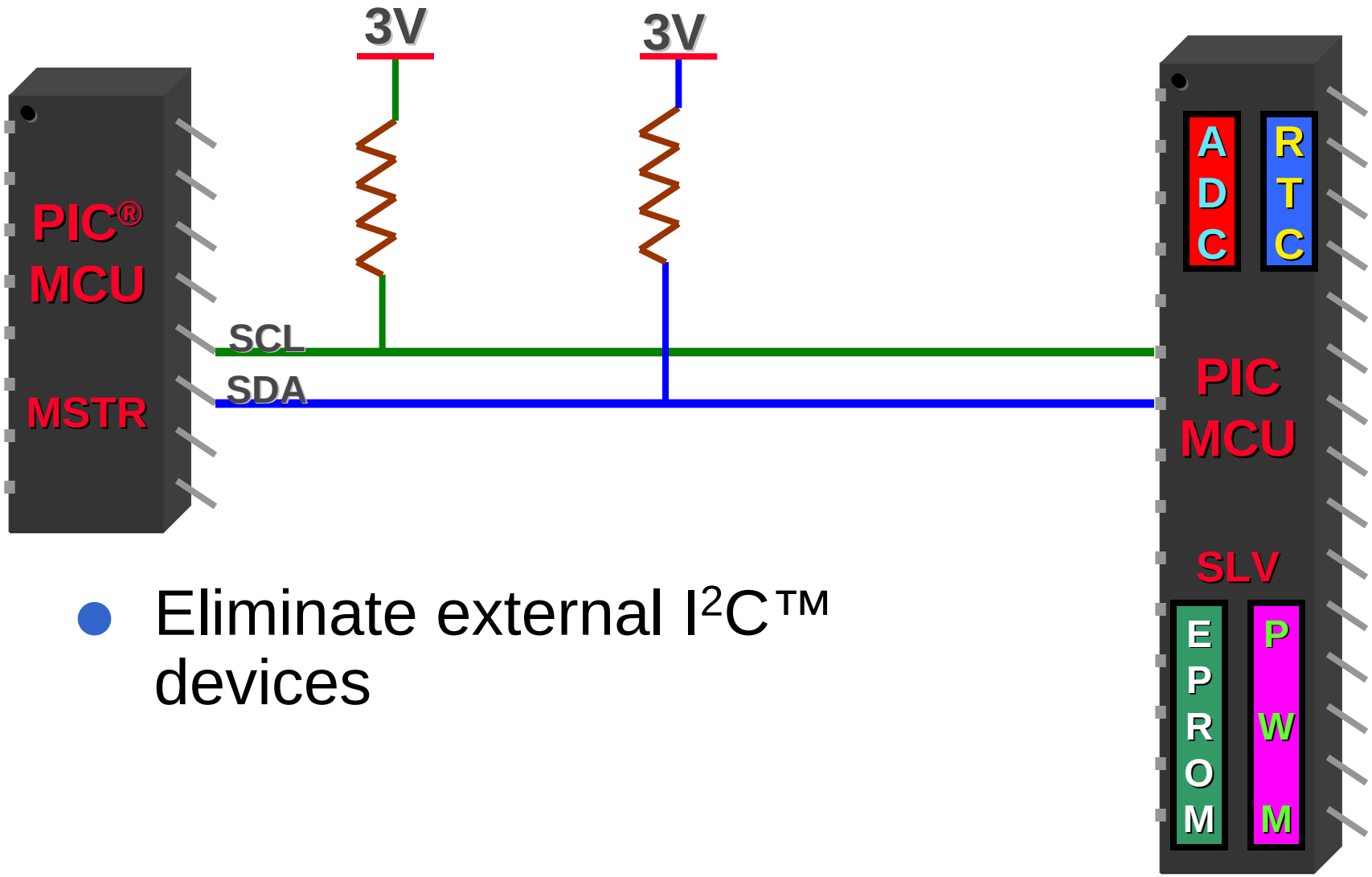
What is it good for?

- IPMI: Intelligent Platform Management Interface
- Embedded system device networking: using a PIC MCU as multiple I²C™ devices

Embedded System Device Networking



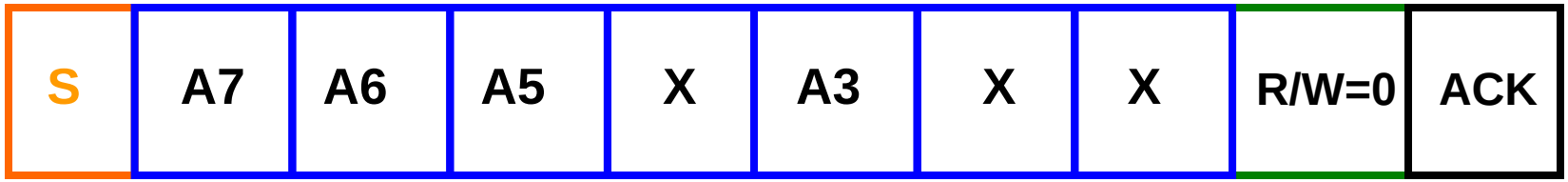
Embedded System Device Networking



- Eliminate external I²C™ devices

Address Masking Example

- By setting the ADMSK bits in SSPCON2, we mask the address bits.
- In this case ADMSK[7:1]=0001011



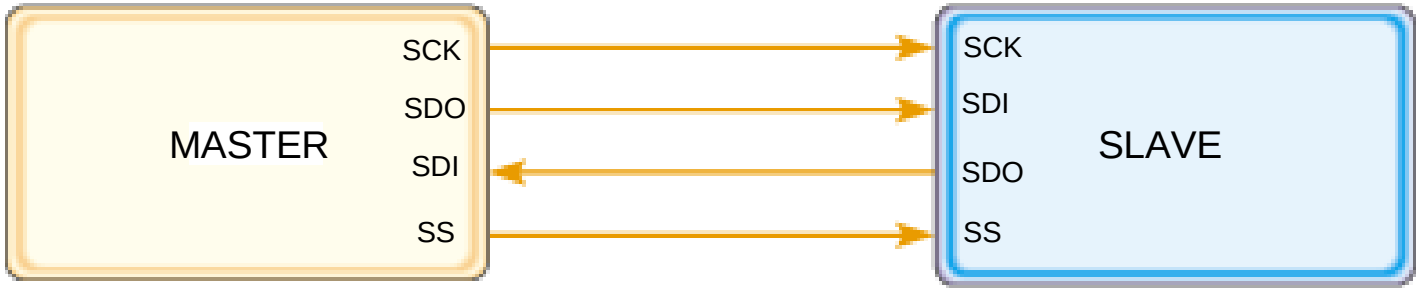
All addresses that will be ACKed:

- A7.A6.A5.0.A3.0.0
- A7.A6.A5.0.A3.0.1
- A7.A6.A5.0.A3.1.0
- A7.A6.A5.0.A3.1.1
- A7.A6.A5.1.A3.0.0
- A7.A6.A5.1.A3.0.1
- A7.A6.A5.1.A3.1.0
- A7.A6.A5.1.A3.1.1

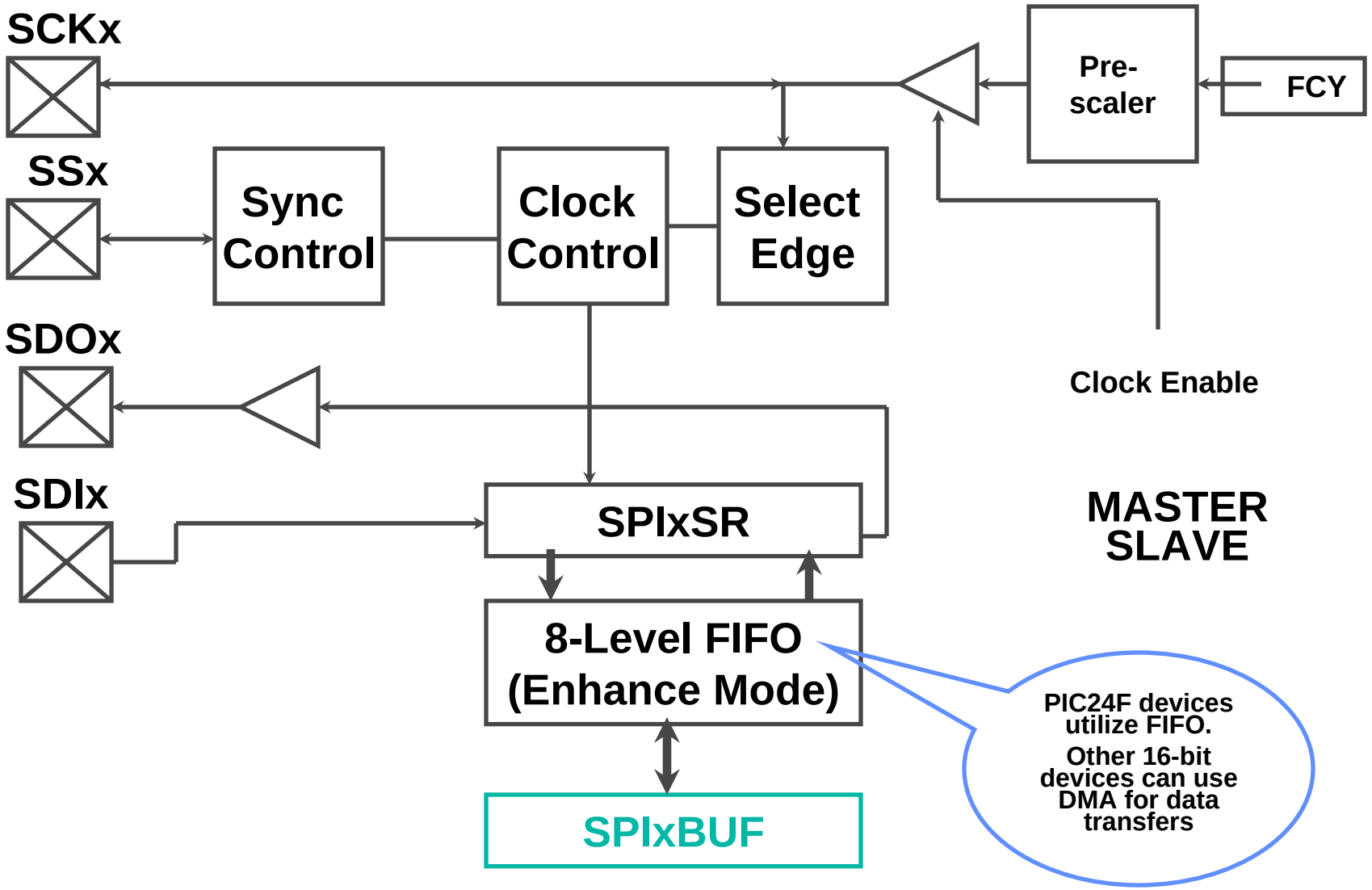
SPI

SPI - Overview

- Serial transmission and reception of 8/16-bit data
- Full-duplex, synchronous communication
- 4-wire interface
- Supports 4 different clock formats and serial clock speeds up to 10 Mbps
- Buffered Transmission and Reception

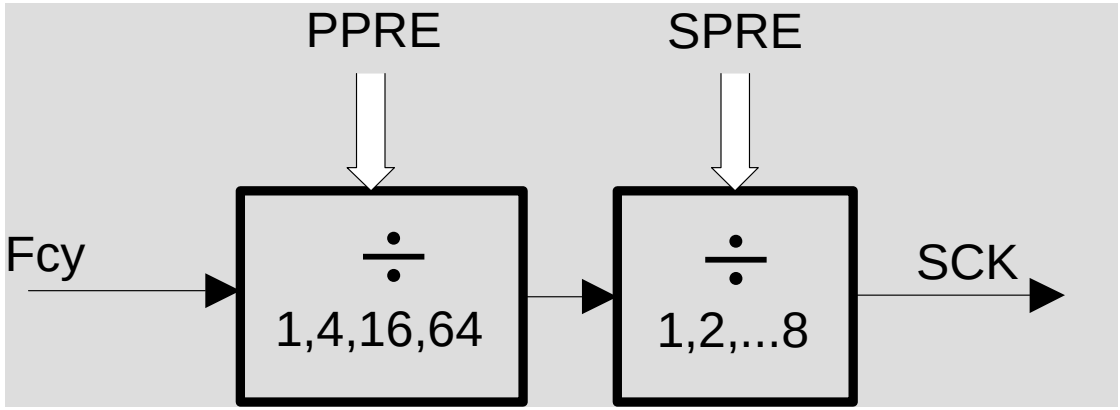


SPI

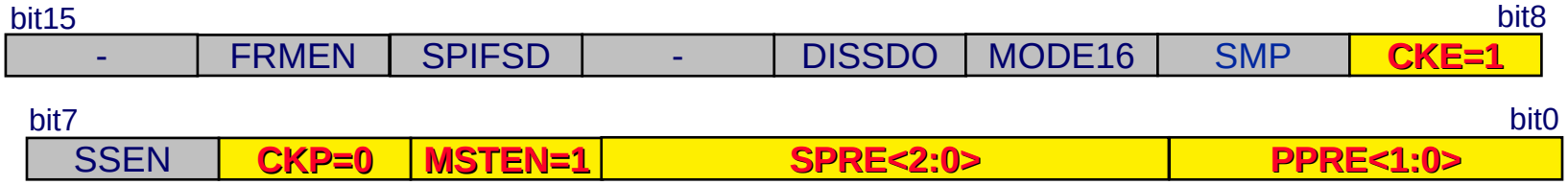


SPI Clock

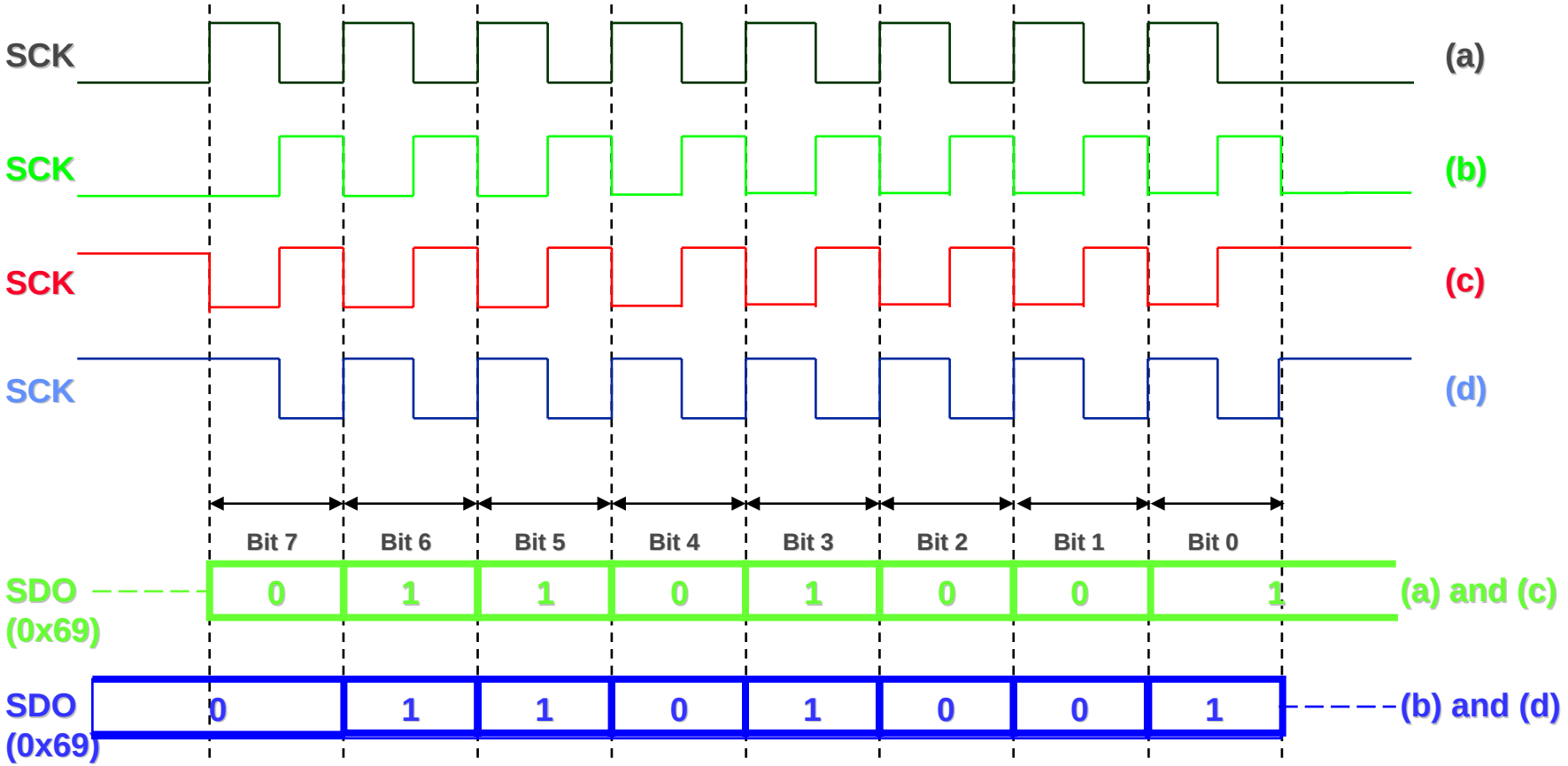
- **CKP: Clock Polarity Select**
  - **0:** Idle State of CLK is LOW
  - **1**: Idle State of CLK is HIGH
- **CKE: Clock Edge Select**
  - **0:** Transmit data on Idle to Active Edge
  - **1:** Transmit data on Active to Idle Edge



SPIxCON Register



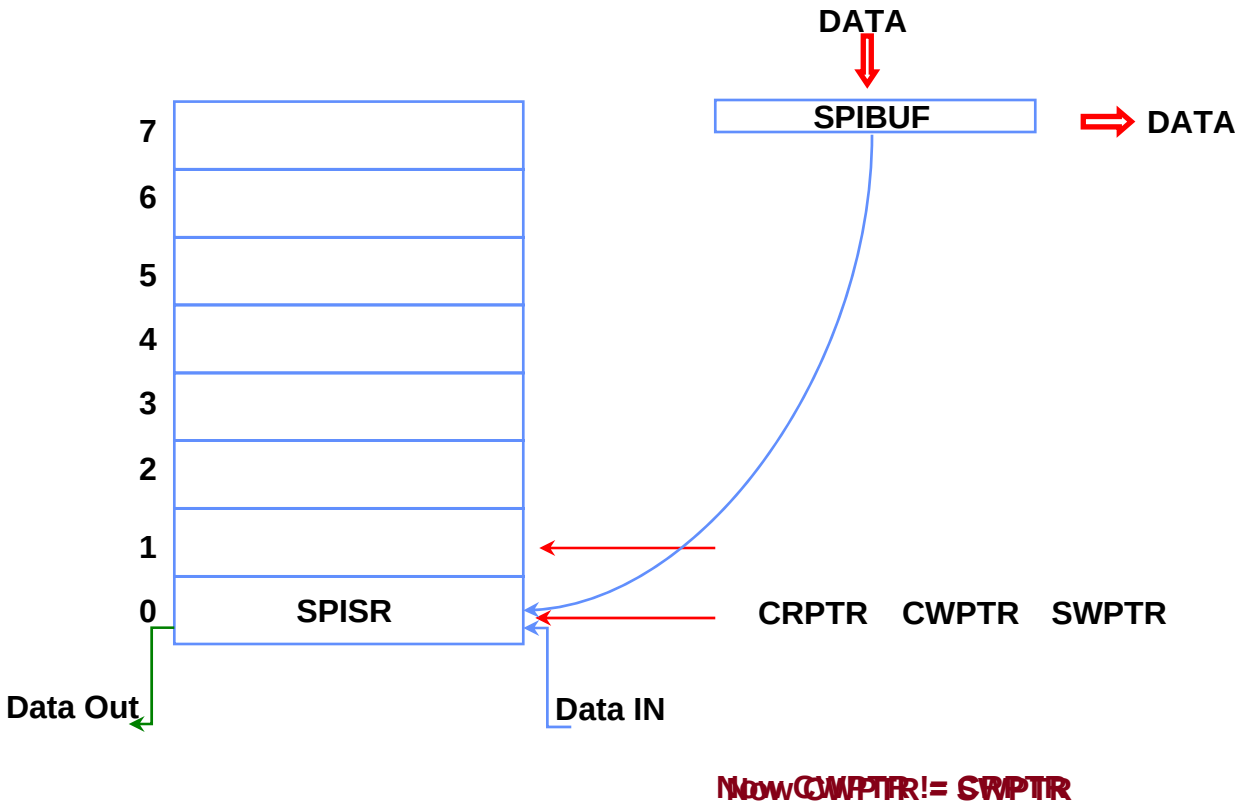
SPI - Serial Clock Formats



(a) CKP = 0, CKE = 0 (b) CKP = 0, CKE = 1 (c) CKP = 1, CKE = 0 (d) CKP = 1, CKE = 1

Buffer Operation in Enhanced Mode

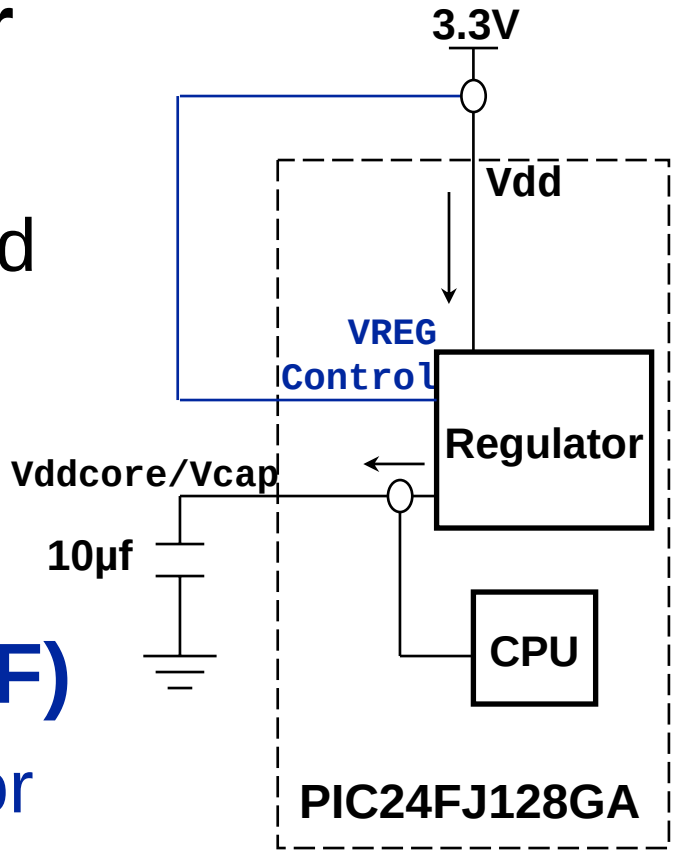
- The 8 level buffer can act as 8*8-bit array or 8*16-bit array, decided by the bit SPIxCON<MODE16>



Special Features

Powering CPU
PIC24FJ, PIC24HJ, dsPIC33FJ

- **Internal on-chip regulator**
  - 2.5V nominal for the CPU
  - 3.3V nominal for the I/O and peripheral voltage
- **Low ESR cap required**
  - 10µf recommended
- **VREG Control pin (PIC24F)**
  - Enable/disable the regulator
- **BOR**



UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

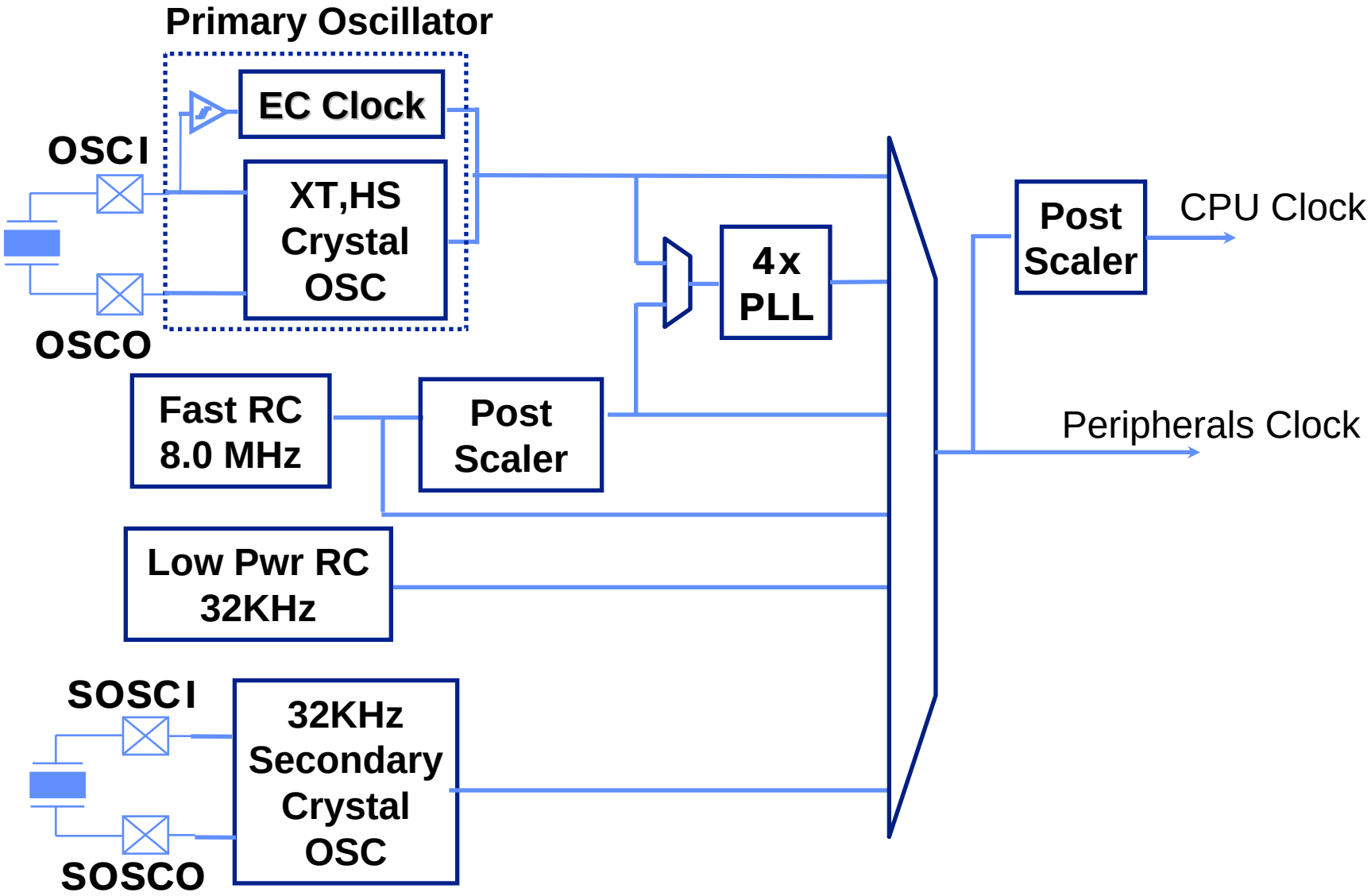
Power Management

- **Power management options:**
  - Lower the supply voltage
  - Lower the clock speed
    - **Postscaler**
    - **Clock switching**
  - Disable unused peripherals
    - **Minimal power saving**
  - Execute `pwrsav` instruction
    - **SLEEP**
    - **IDLE**
  - DOZE mode

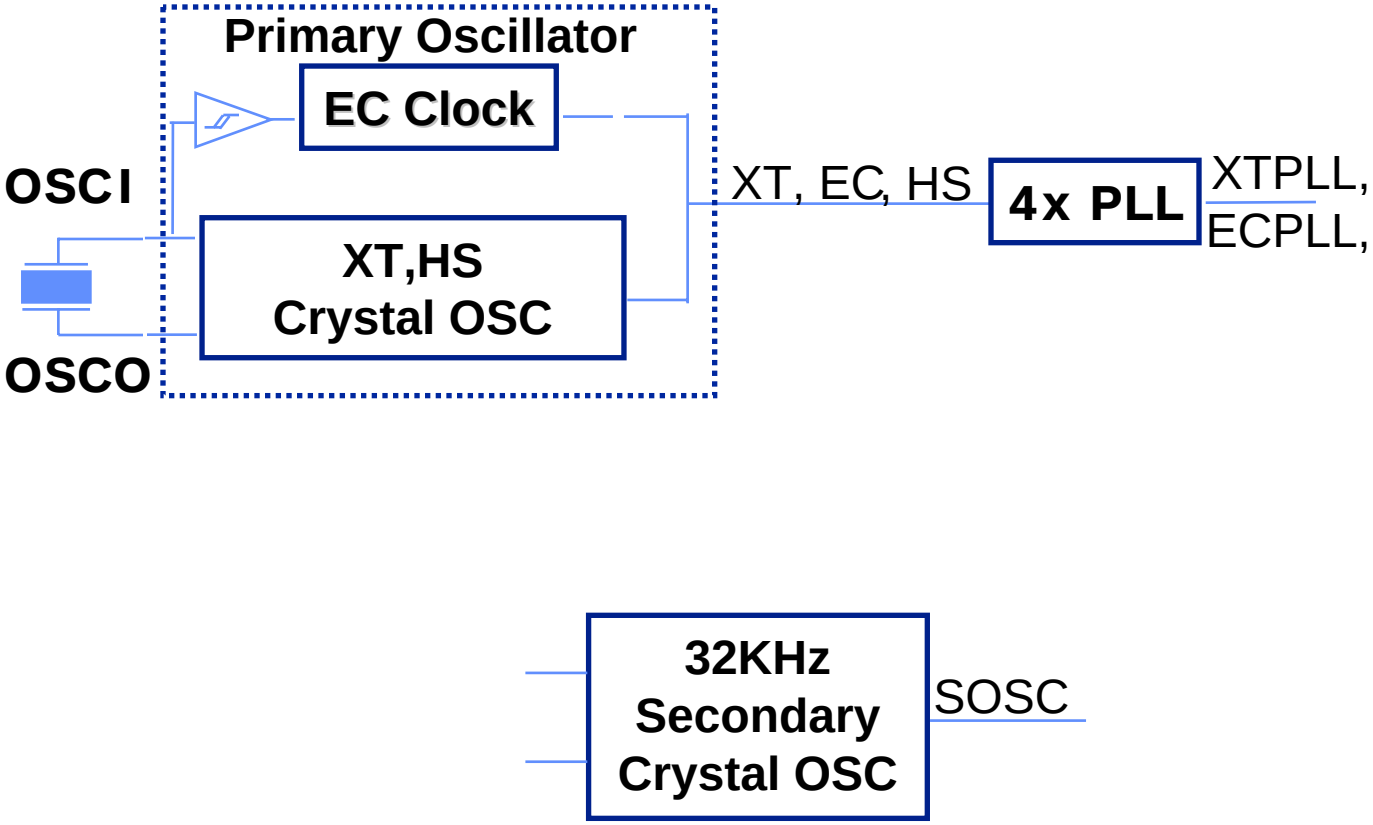
Power Management

- **Exiting Power Saving mode**
  - System Reset
  - Watch Dog Timer
  - Any enabled interrupt
    - **Interrupt Priority Level <= CPU IPL**
      - Execute the next instruction
    - **Interrupt Priority Level > CPU IPL**
      - Execute the interrupt handler
- **How to know it is a wake up**
  - Check **RCON<SLEEP>** and **RCON<IDLE>** bits

Clock Sources

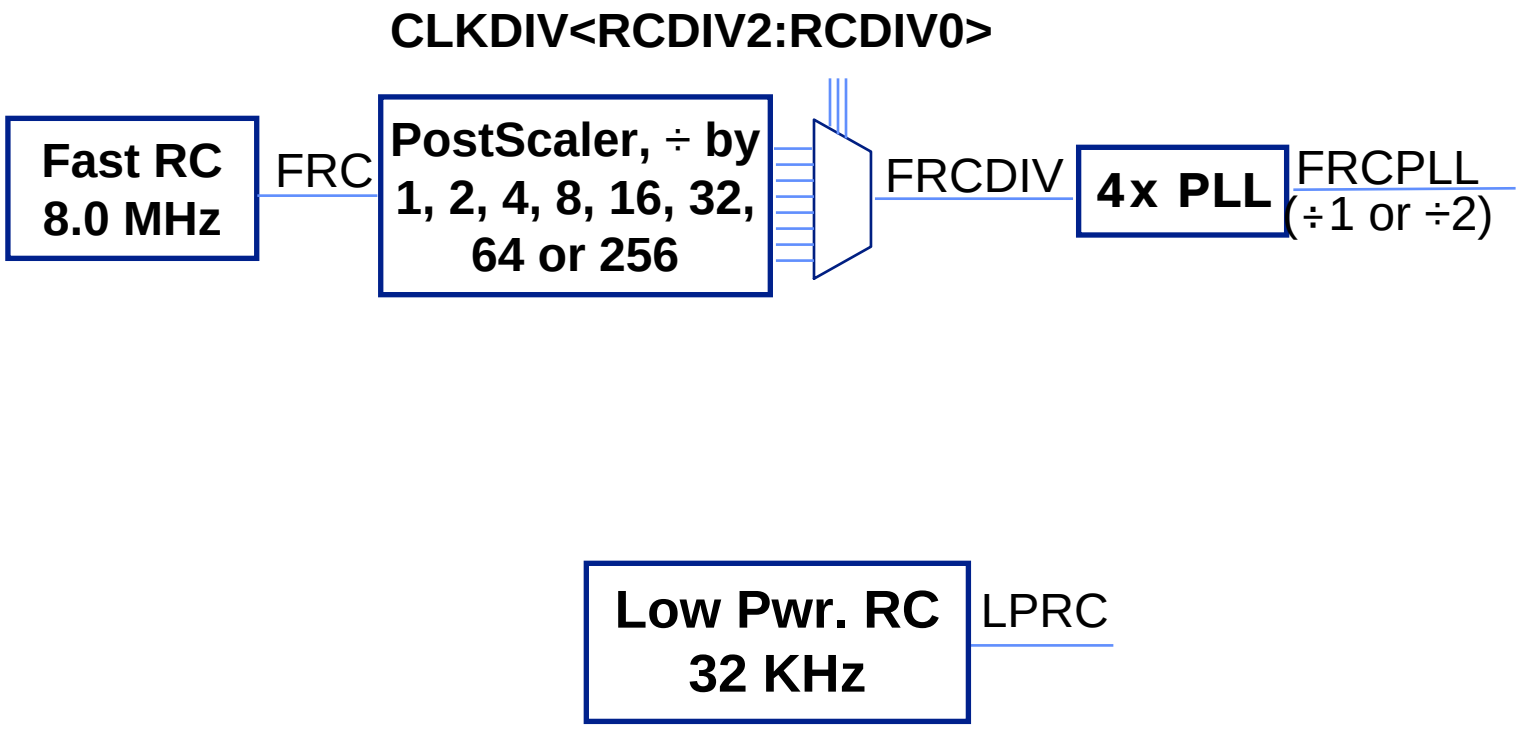


Clock Sources



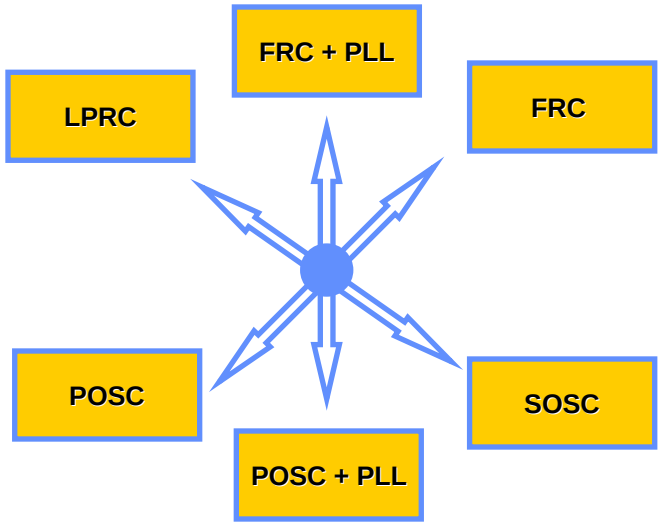
- **It is 4x PLL only in PIC24F. In other 16-bit controllers it can be configured into 4x, 8x or 16x PLL**

Clock Sources



Power Saving Technology

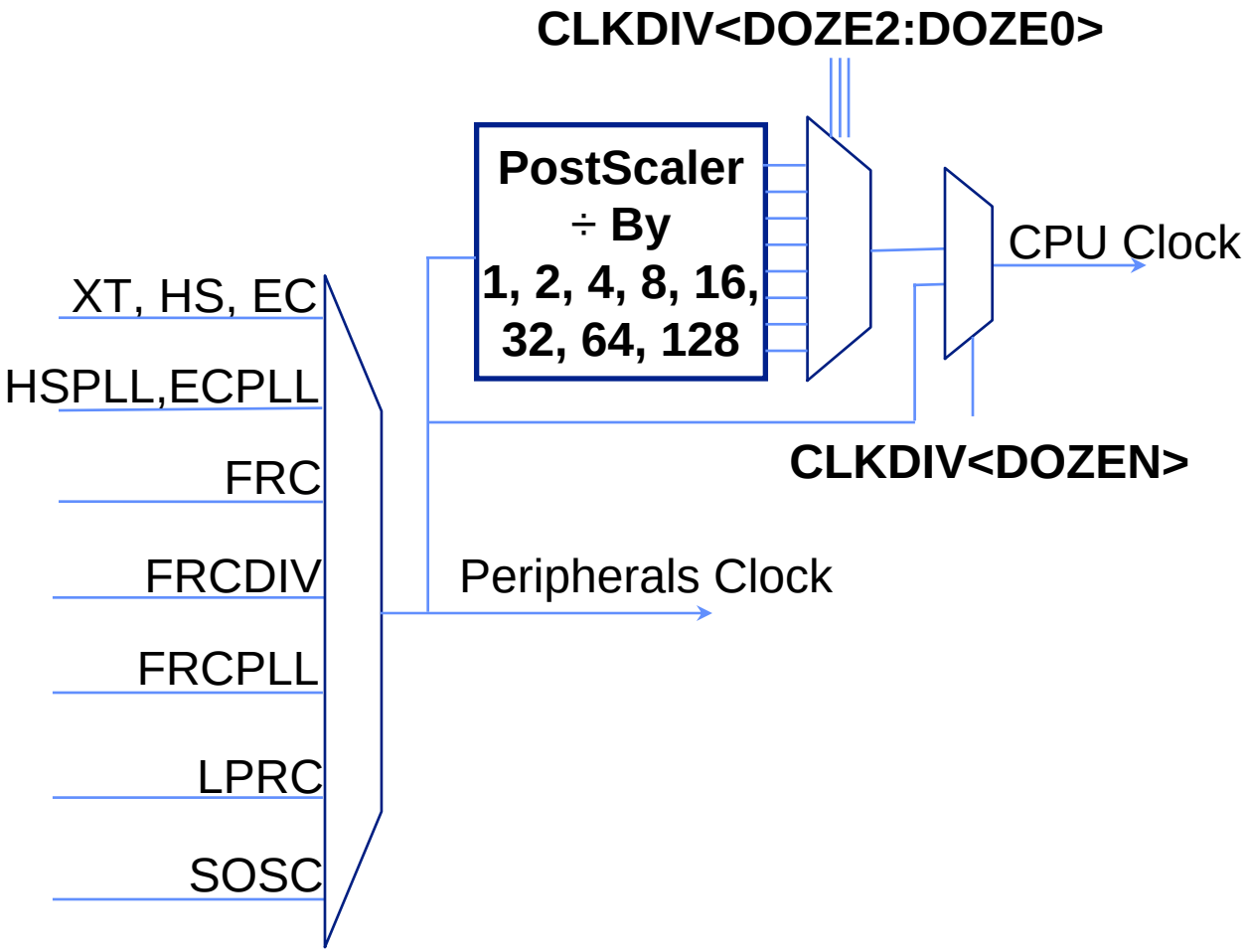
- **On-the-Fly Clock Switching**



  - **OSCCON<OSWEN>** bit to enable On Fly Clock Switching and **OSCCON<NOSC2:NOSC0>** bits to select the new clock source

- **Instruction-Based Power Saving Modes**
  - Sleep
  - Idle

Doze Mode



Fail-Safe Clock Monitor

- **What happens on Oscillator Failure?**
  - Device switches automatically to the FRC oscillator if FSCM is enabled
    - **FSCM controlled by the bit `FOSC<FCKSM>`**
  - Oscillator Failure Trap is generated
  - Code execution vectors to the Oscillator Fail trap handler, if one exists. Here the application should:
    - **Clear the Clock Fail bit, `OSCCON<CF>`**
    - **Clear the Osc FAIL trap flag, `INTCON1<OSCFAIL>`**
    - **Switch to required Clock Source form FRC**
    - **Execute a RETFIE**

Resets

- **PIC24 RESETS**
  - Power-on Reset (POR)
  - Pin Reset (MCLR)
  - RESET Instruction (SWR)
  - Watchdog Timer Reset (WDT)
  - Brown-out Reset (BOR)
  - Trap Conflict Reset (TRAPR)
  - Illegal Opcode Reset (IOPUWR)
  - Uninitialized W Register Reset (UWR)
  - Configuration word mismatch Reset (CM)
- **For all the resets PC vectors to address 0**

Watchdog Timer

- **Recovers from software malfunction**
- **Resets MCU if not attended on-time**
  - Software must clear it periodically (**`CLRWDT`**)
- **Programmable timeout period**
  - 1 ms to 131 s typical
- **Fuse time programmable prescaler and postscaler**
- **Both Fuse time and Run time Enable available**
- **Option to select Windowed WDT**
  - **`CLRWDT`** should be used only in last quadrant
  - If **`CLRWDT`** is used earlier, resets the device

University of Microchip UofM MASTERs 2007

Programming Support

- **In Circuit Serial Programming™ (ICSP™):**
  - Device can be programmed in circuit
  - Useful to Combine Programming and Final test
- **Enhanced In Circuit Serial Programming™ (Enhanced ICSP™) capability:**
  - Uses programming executive for faster programming
- **Run Time Self Programming (RTSP):**
  - Device can program its own Flash memory
  - Useful for Remote code updates
- **JTAG Interface**
  - Programming support through Serial Vector Format files
  - Provides for boundary scan

University of Microchip UofM MASTERs 2007

Summary

- **We reviewed the 16-Bit Family Architecture**
  - 16-bit Datapath & Key CPU registers
  - Program/Data Memory and Data Access from Program Memory
  - Interrupt Handling and latency
- **We reviewed the MPLAB® C30 C Language Extensions**
  - The basics of creating an embedded c-application for the PIC24F/dsPIC® DSC
- **We reviewed the 16-bit Family Basic Peripherals**
  - I/O Ports
  - ADC
  - Comparators, Voltage Reference
  - Timers
  - Input Capture
  - Output Compare & PWM
  - USART
  - I^2C™
  - SPI
- **Special features of 16-bit microcontrollers**

Summary

- Carried out the Labs on
  - Using MPLAB® and C30
  - PSV initialization and usage
  - Interrupt handling and its priority setting
  - Initialization and ADC conversion
  - 32-bit Timer configuration and its usage
  - UART configuration and use of FIFO

Development Tools Used

- **MPLAB® REAL ICE™ In-Circuit Emulator**
  - DV244005

- **Explorer 16 with PIC24FJ128GA010 PIM**
  - DM240001

References

- PIC24F, PIC24H, dsPIC33 F Datasheets
- PIC24F Family Reference Manual
- dsPIC30F Family Reference Manual
- Explorer 16 User's Guide
- C30 Compiler User's Guide
- Other 16-bit RTC Courses
  - **204 ADV: Advanced 16-bit peripherals**
  - **103 ASP: Intro. To 16-Bit Arch. & Assembly Language Programming**
  - **104 DSP: dsPIC® DSC DSP Engine**
  - **301 MCW: BLDC Motor Control Using dsPIC DSC**
  - **312 DPS: SMPS Design Using dsPIC DSC**

Please complete the review form

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.