YOU + MICROCHIP ENGINEERING THE FUTURE TOGETHER

1204 P16

Introducing the Enhanced PIC16

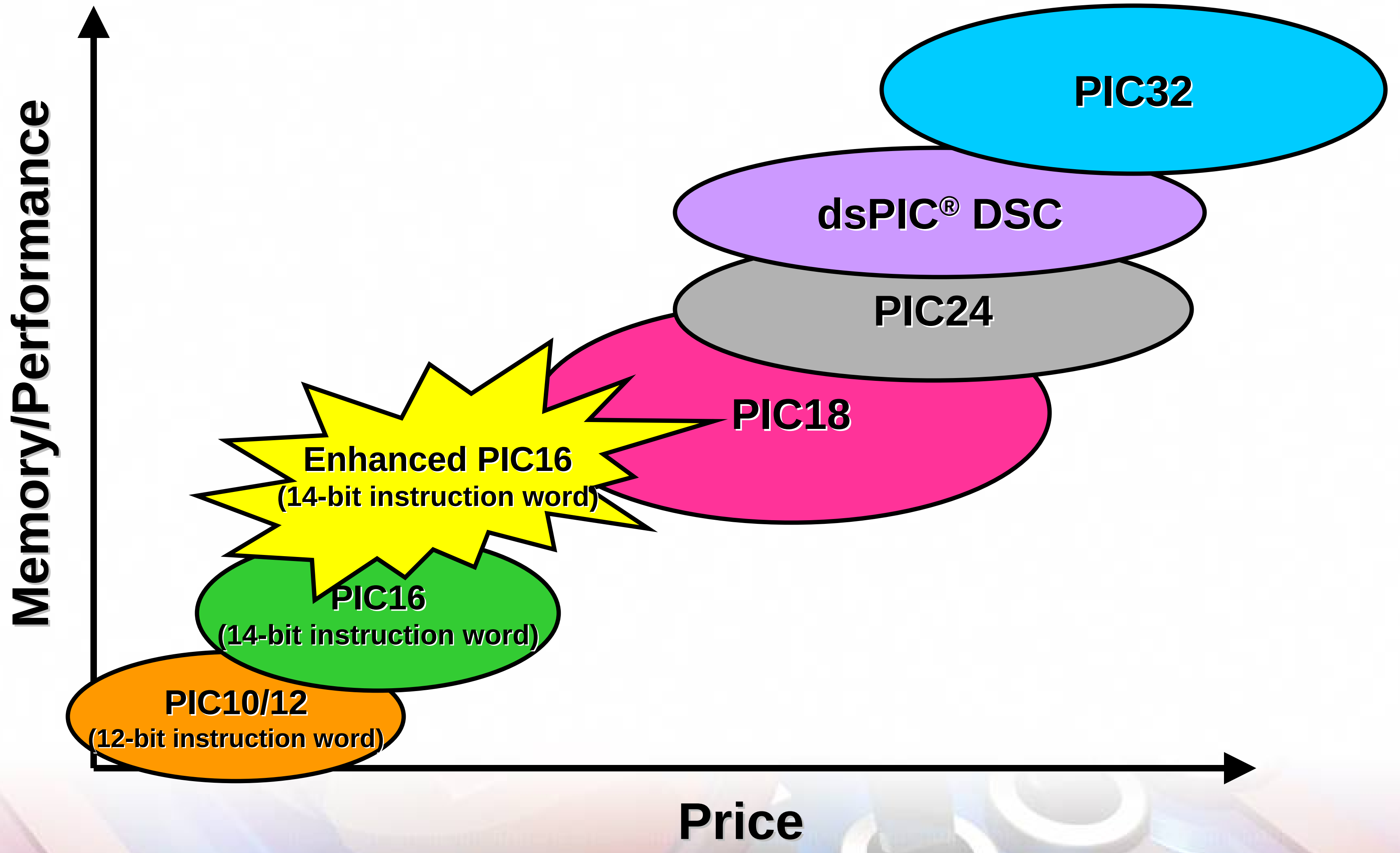
Agenda

- **Introducing the PIC16F1XXX**
- Migration to the PIC16F1XXX
- New Coding Tricks
- Advanced Capabilities

Introducing the PIC16F1XXX

- Overview
- Memory Map
- New Instructions
- Enhanced Indirect Memory Features

PIC® MCU Family Roadmap


Memory/Performance
PIC32
dsPIC® DSC
PIC24
PIC18
Enhanced PIC16
(14-bit instruction word)
PIC16
(14-bit instruction word)
PIC10/12
(12-bit instruction word)
Price

PIC16 Enhancement Goals

- Increase the maximum program memory
- Increase the space for peripherals
- Increase the maximum data memory
- Reduce the penalty for paging/banking
- Improve the ‘C’ efficiency
- Minimize the difficulty of migration

Front Page Comparison of PIC16XXX and PIC16F1XXX

OLD

- **High-Performance RISC CPU**
- **Only 35 instructions**
  - All single-cycle instructions except branches
- **Operating speed:**
  - DC – 20 MHz oscillator/clock input
  - DC – 200 ns instruction cycle
- **Interrupt capability**
- **8-level deep hardware stack**
- **Direct, Indirect and Relative Addressing modes**

NEW

- **High-Performance RISC CPU**
- **Only 48 instructions**
  - All single-cycle instructions except branches
- **Operating speed:**
  - DC – *32* MHz oscillator/clock input
  - DC – *125* ns instruction cycle
- **Interrupt capability with automatic context saving**
- **16-level deep hardware stack with overflow/underflow reset**
- **Direct, Indirect and Relative Addressing modes**
  - Two full 16-bit File Select Registers (FSRs)
  - FSRs read Program and Data memory.

Quick Comparison

	PIC16	Enhanced PIC16	PIC18
Max GPR/SFR	336/110	2496/316	4096/159+ More if the SFRs are outside of the access bank.
Max Program	8Kx14	32Kx14 16K is likely the largest device	1Mx16
FSRs	1	2 Can access Program Memory	3
Instruction Count	35	48	75 83 including the optional extended instructions
Stack	8	16 With over/underflow reset	31 With over/underflow reset
Interrupts	1	1 Hardware context save	2 Optional hardware context save
Program Memory Read	All devices via RETLW. Some devices via EEPROM interface	All devices via RETLW or FSR. All devices via EEPROM interface	All devices via TABLRD instructions

New Data Memory Map

- 32 banks of file registers
- 15 banks are reserved for the future

The memory map is simplified:

- Bottom 16 bytes of each bank are common
- First 12 bytes of each bank are for the CPU registers
- SFRs are located at address 12-31

New Features

- W is mapped to “w reg”
- Banks 16-30 are reserved for future functionality
- Bank 31 has advanced functions

Data Memory Map

	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	◄······►	Bank 31
0x000 0x00B	12 Common CORE SFRs							
0x00C 0x01F	SFRs 20	SFRs 20	SFRs 20	SFRs 20	SFRs 20	SFRs 20		Bank 31 Special Functions Stack Access and Debugging Registers
0x020 0x06F	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	GPR 80 Bytes	Banks 6-30	
0x070 0x07F	Common Memory (16 bytes)							

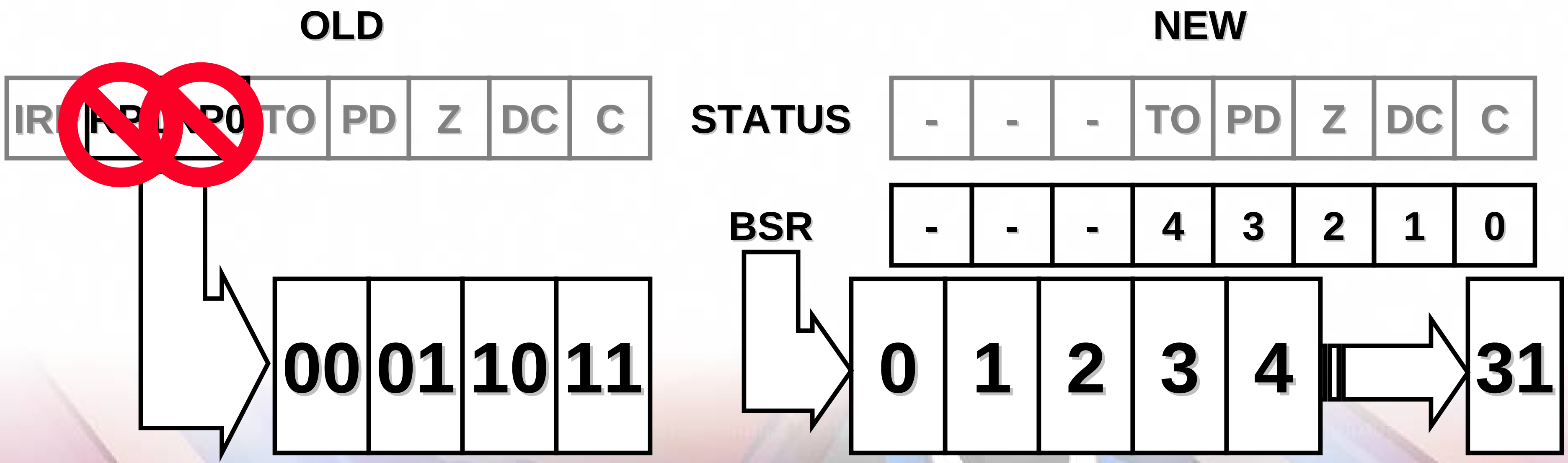
© 2008 Microchip Technology Incorporated. All Rights Reserved.

Common Core Registers

New	Saved	Address	Register	Function
		0x00	INDF0	Indirect Register 0
⚡		0x01	INDF1	Indirect Register 1
	●	0x02	PCL	Program Counter Low
	●	0x03	STATUS	Status Register
	●	0x04	FSR0 Low	File Select Register 0 Low Byte
⚡	●	0x05	FSR0 High	File Select Register 0 High Byte
⚡	●	0x06	FSR1 Low	File Select Register 1 Low Byte
⚡	●	0x07	FSR1 High	File Select Register 1 High Byte
⚡	●	0x08	BSR	Bank Select Register
⚡	●	0x09	WREG	Working Register
	●	0x0A	PCLATH	Program Counter Latch High
⚡		0x0B	INTCON	Interrupt Control Register

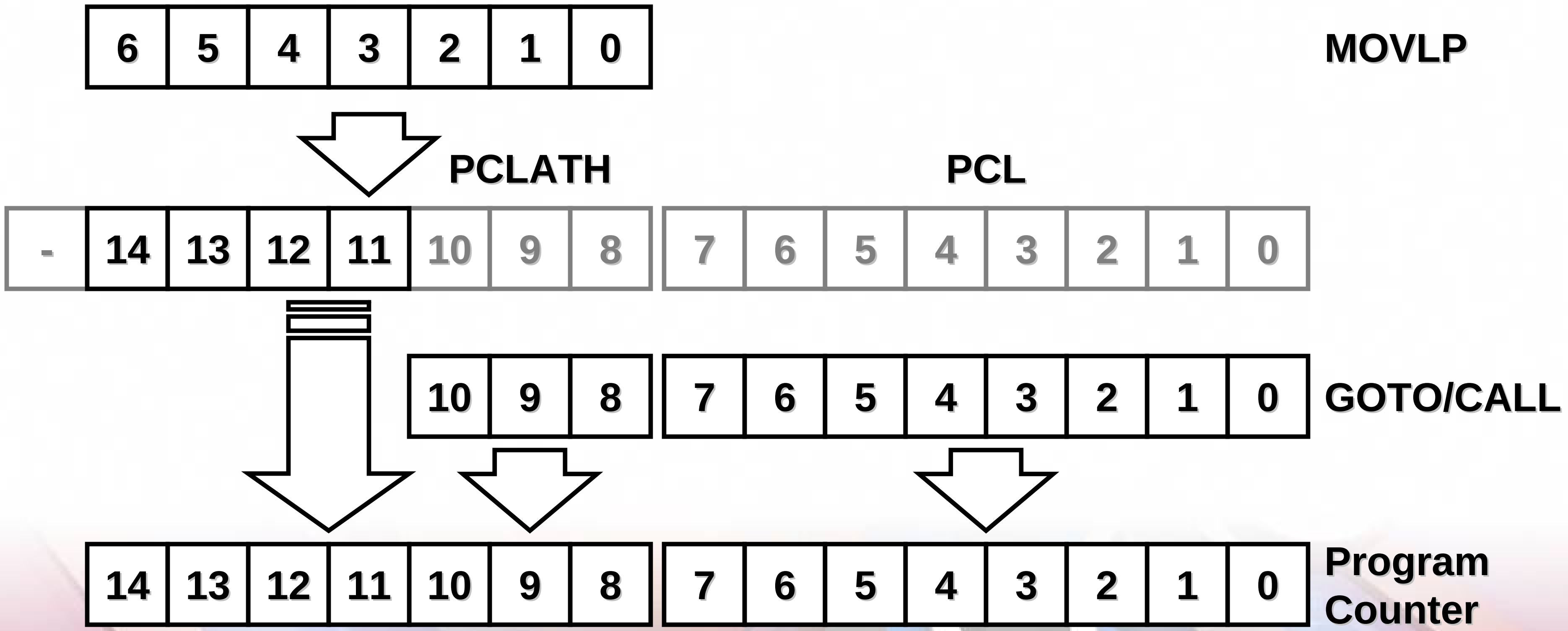
Data Memory Banking

- The old device required banking via RP0 and RP1 in the STATUS Register
- These bits NO LONGER EXIST
- Now the BSR register handles all banking
- The new MOVLB instruction selects the bank in one instruction



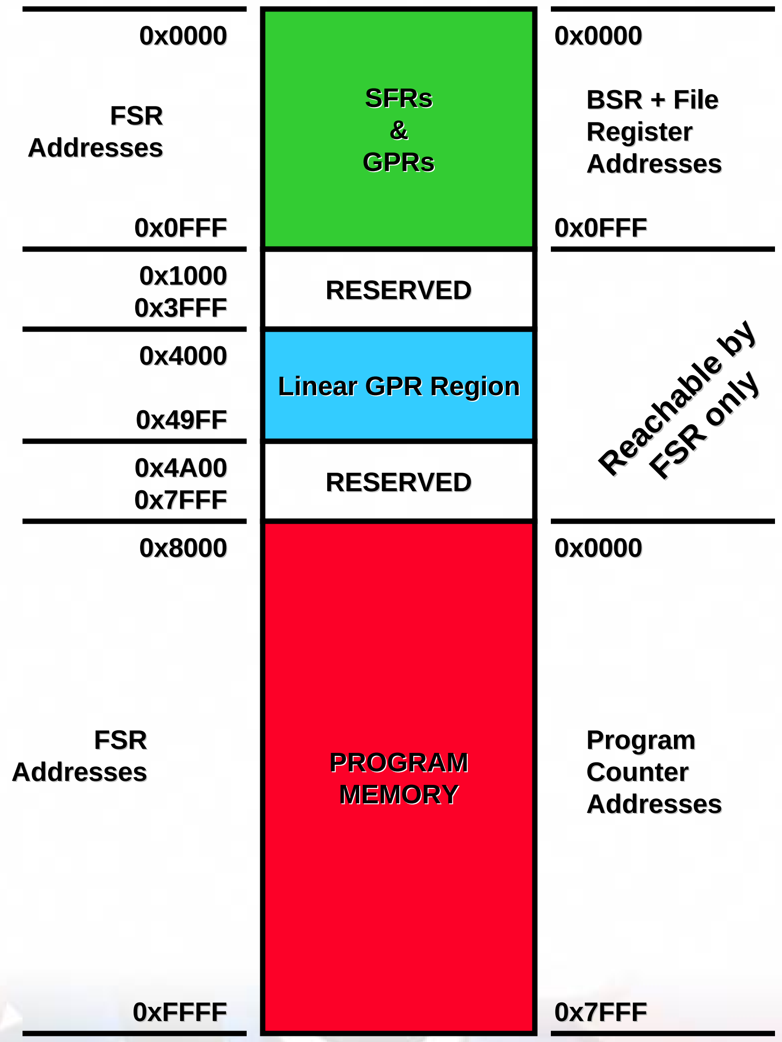
Program Memory

- Program Memory extended to 16 pages of 2 Kbytes
- Paging simplified with MOVLP instruction



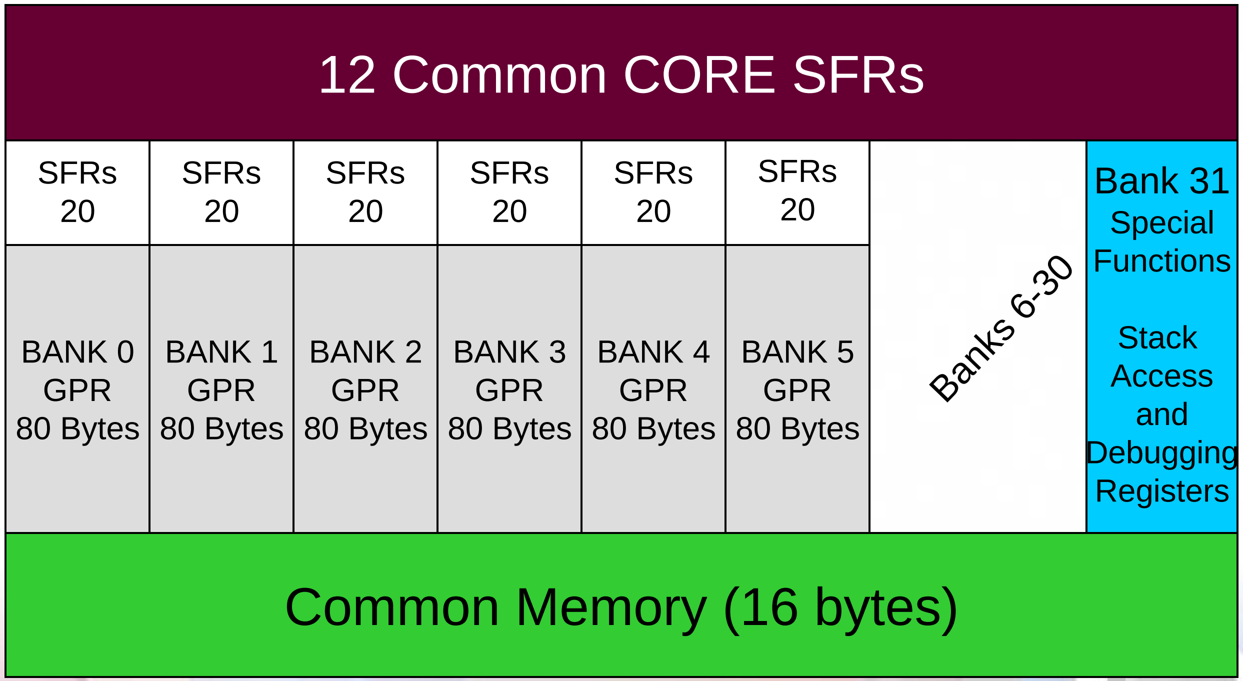
New FSR

- 2 16-bit FSRs
- FSR can access all file registers and all program memory
- New FSRs allow one data pointer for all memory
- FSRs are now supported by new instructions



Linear GPR Region

- Shadows 80 byte GPR blocks into linear array
- Keeps FSR operations inside GPR area
- Allows large stacks, arrays, buffers, etc.
- Access is via the FSR and a second address range

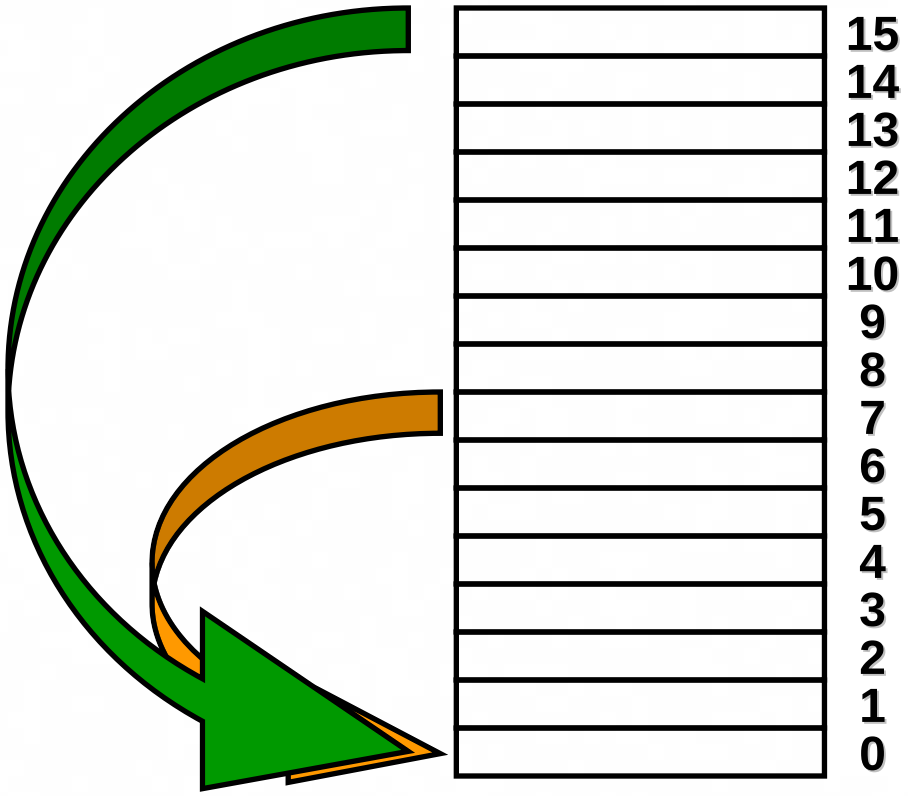


Fast Context Save

- **Interrupts automatically save the context**
  - W
  - STATUS
  - BSR
  - FSRs
  - PCLATH
- **RETFIE automatically restores the context**
- **You cannot disable fast context save**

Stack

- 16 Stack Entries
- Over/Underflow Reset (optional)
- User/ICD Stack access in Bank 31
  - Read/Write the stack in Bank 31
    - **Useful for RTOS or Safety Critical Debugging**



Stack Reset Mode

- **The STRVEN (Stack Reset Violation Enable) config bit enables the Stack Reset mode**
- **Stack Reset Mode Causes:**
  - Return when stack is empty causes a Reset
  - Call or interrupt when stack is full causes a Reset
  - Reading the top of stack when stack is empty returns 0

Normal Mode

- **The stack works exactly as the legacy device plus the following features:**
  - 16 entry stack
  - Stack access via SPTR and TOS

New Instructions

- ADDWFC – Add W + F with Carry
- SUBWFB – Subtract F – W with Borrow
- LSLF – Logical Shift Left
- LSRF – Logical Shift Right
- ASRF – Arithmetic Shift Right
- MOVLP – Move Literal to PCLATH
- MOVLB – Move Literal to BSR
- BRA – Branch Relative (signed)
- BRW – Branch PC + W (unsigned)
- CALLW – Call PCLATH:W
- ADDFSR – Add Literal to FSRn (signed)
- MOVIW – Move indirect to W
- MOVWI – Move W to Indirect
- RESET – Reset Hardware and Software

Arithmetic with Carry

- **ADDWFC**
  - Add with Carry
- **SUBWFB**
  - Subtract with Borrow
- **Literal operations with carry or borrow are not supported**

New Arithmetic Shifts

- **LSLF, LSRL, ASRF**
  - Logical Shift Left
    - **Shift Left, MSB goes to Carry, LSB is now Zero**
    - **This is the same as arithmetic shift left**
  - Logical Shift Right
    - **Shift Right, MSB is now Zero, LSB goes to Carry**
  - Arithmetic Shift Right
    - **Shift Right, Sign extend on MSB, LSB goes to Carry**

Paging/Banking

- **MOVLP**
  - Places 7-bit literal in PCLATH

    MOVLP HIGH LABEL
  - PAGESEL in 1 cycle without disturbing W
  - MOVLP + CALL/GOTO takes 3 cycles and 2 instructions but reaches ANYWHERE in memory
- **MOVLB**
  - Places 5-bit literal in BSR
  - BANKSEL in 1 cycle without disturbing W for ANY number of banks
  - *IRP, RP0, RP1 are now obsolete and have been removed*

Relative Branching

- **Relative Branching allows MOST code to eliminate the paging concerns**
- **BRA N**
  - Always branch to PC + N
  - Range is -256 <= N <= 255
  - PC + N is 15-bit math so no paging issues
- **BRW**
  - Always branch to PC + W (unsigned)
  - Fast lookup tables/State Machines
  - PC + W is 15-bit math so no paging issues
- **CALLW**
  - Call to PCLATH, W
  - Fast Lookup Tables/State Machines/Function Pointers
  - PCLATH:W direct address

FSR Support Instructions

- **ADDFSR Instruction**
  - Adds a signed literal to the selected FSR
  - Literal range is -32 to +31
- **MOVIW/MOVWI – Move Indirect to W and Move W to Indirect**
  - Special Modes
    - **Pre/Post Increment**
    - **Pre/Post Decrement**
    - **Relative Offset**
      - Same range as ADDFSR

MOVIW/MOVWI Syntax

- **Standard**

```
MOVWF INDF0
```

- **Pre Increment**

```
MOVIW ++INDF0
```

- **Post Increment**

```
MOVWI INDF0++
```

- **Pre Decrement**

```
MOVIW –INDF0
```

- **Post Decrement**

```
MOVWI INDF0--
```

- **Offset**

```
MOVWI k[INDF0]
```

-32 <= k <= 31

Miscellaneous Features

- **RESET instruction**
  - No more GOTO 0
  - All peripherals are reset
  - Software version of MCLR Reset
  - Another PCON bit is available to indicate a software Reset
- **Program Memory Read (PMR) is in every device**
- **Device ID, User ID and Config Word are now readable by the firmware**

Lab 1

- **Explore the new CPU via simulator**
- **Write the following functions:**
  - Clear all GPRs (RAM)
  - 16-bit Add

A prize is available for the
FASTEST RAM CLEAR IN THE WEST
The prize is awarded at the end of the class.

- **A working example is in the lab directory**

Agenda

- Introducing the PIC16F1XXX
- **Migration to the PIC16F1XXX**
- New Coding Tricks
- Advanced Capabilities

Migration

- Paging
- Banking
- Interrupts
- Indirect Memory

Paging

- **Use the PAGESEL macro, or**
  - Automatically uses MOVLP
- **Update all PCLATH code**
  - Assure 7-bit data in PCLATH
- **Convert to relative branches**
  - This will eliminate most paging issues

PAGESEL

My ASSEMBLY Code

```
My_Function
  movlw 0x04
  movwf  delay_cntr
My_function_loop
  decfsz  delay_cntr
  goto My_function_loop
  return

Main
    do lots of stuff
    PAGESEL My_Function
    call My_Function
    do lots of other stuff

    end
```

PAGESEL MACRO PIC16

```
My_Function
  movlw 0x04
  movwf  delay_cntr
My_function_loop
  decfsz  delay_cntr
  goto My_function_loop
  return

Main
    do lots of stuff
    movlw high My_Function
    movwf PCLATH
    call My_Function
    do lots of other stuff
    end
```

PAGESEL MACRO ENHANCED PIC16

```
My_Function
  movlw 0x04
  movwf  delay_cntr
My_function_loop
  decfsz  delay_cntr
  goto My_function_loop
  return

Main
    do lots of stuff
    movlp high My_Function
    call My_Function
    do lots of other stuff

    end
```

© 2008 Microchip Technology Incorporated. All Rights Reserved.

Banking

- **Use BANKSEL macro, or**
  - Automatically uses MOVLB
- **Replace writes to STATUS with writes to BSR**

BANKSEL

My ASSEMBLY Code

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

            ff
   B NKSEL Var1
   addwf  Var1
   do lots of other stuff


   end
```

Always works

BANKSEL MACRO PIC16

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

  code

Main
     do lots of stuff
     bsf STATUS,RP0
     bcf STATUS,RP1
     addwf  Var1
     do lots of other stuff


     end
```

BANKSEL MACRO ENHANCED PIC16

```
 data
Var1 res 1
Var2 res 1
Var3 res 1

  code

Main
     do lots of stuff
     movlb Var1 >> 7
     addwf  Var1
     do lots of other stuff


     end
```

Saves 1 instruction and accesses more banks of memory

Interrupts

- RETFIE works a little different
- Make sure interrupts do not pass parameters with W (bad practice)
- Remove core context save/restore software

Indirect Memory

- IRP bit is gone
- Accessing more than 256 bytes requires an update to FSR<x>H register
- Fastest method to update FSR<x>H register requires modifying W
- BANKISEL performs 8 bcf’s or bsf’s

Migration Summary

- Migrating to the PIC16F1XXX should be simple
- Migrating back from the PIC16F1XXX can be difficult
- Using the BANKSEL and PAGESEL macros will be a big help

Lab 2

- Migrate the supplied encryption program to the PIC16F1000
- Current software is in 1204/lab2/
- Do not optimize (that is the next lab)

Agenda

- Introducing the PIC16F1XXX
- Migration to the PIC16F1XXX
- **New Coding Tricks**
- Advanced Capabilities

New Code Tricks

- Relative Branching
- Table Reads
- 16-bit Arithmetic
- Robust Techniques

Relative Branches

ORIGINAL ASSEMBLY Code

```
Additional Code
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   goto My_function_loop
   return

Main
     do lots of stuff
     PAGESEL My_Function
     call My_Function
     do lots of other
stuff
```

Page Boundary at this point will force the need for a PAGESEL before My_function_loop

NEW ASSEMBLY ENHANCED PIC16

```
My_Function
   movlw 0x04
   movwf  delay_cntr
My_function_loop
   decfsz  delay_cntr
   bra My_function_loop
   return

Main
     do lots of stuff
     PAGESEL My_Function
     call My_Function
     do lots of other stuff
     end
```

Relative branch makes this code ALWAYS work with no paging issues.

No relative CALL support. CALLW is not relative.

Table Reads

- **The PIC16F1XXX has new ROM table access methods**
  - FSR
  - Relative Branch

Tables using FSR

- Long setup

```
The_CODE
        movlw  high Table_start
        movwf  FSR0H
        movlw  low Table_start
        movwf  FSR0L
        movlw  3
        addwf  FSR0L
        movf   INDF0,w

Table_start
        DT 3,4,5,6,7,8,9
```

Fast Call Tables

- **This code returns a constant from a table aligned on a 256 word boundary**

`Table_Function`	`The_CODE`
`    movlw  high Table_start`	`    movlp  high Table_start`
`    movwf  PCLATH`	`    movlw  3`
`    movlw  3`	`    callw`
`    movwf  PCL`	
`Table_start`	`Table_start`
`    DT 3,4,5,6,7,8,9`	`    DT 3,4,5,6,7,8,9`

© 2008 Microchip Technology Incorporated. All Rights Reserved.

More Fast Tables

- Returns a const from a table
- NO alignment issues
- Table start can occur ANYWHERE

```
The_CODE
        movlw  high Table_start
        movwf  PCLATH
        movlw  low Table_start
        addwf  3
        btfss  STATUS,C
        incf   PCLATH,f
        movwf  PCL
Table_start
        DT 3,4,5,6,7,8,9
```

```
The_CODE
        movlp  high Table_start
        movlw  3
        brw
Table_start
        DT 3,4,5,6,7,8,9
```

16-bit Arithmetic

- Carry support speeds 16-bit math

```
movf val_a_l            movf val_a_l
addwf val_b_l           addwf val_b_l
btfsc STATUS,C          movf val_a_h
incf val_b_h            addwfc val_b_h
movf val_a_h
addwf val_b_h
```

Robust Techniques

- RESET instruction
- Stack over/underflow reset

Lab 3

- **Modify lab 2 to take full advantage of the 16F1000**
- **Remember:**
  - FSR
  - Relative branching
  - New arithmetic instructions

Agenda

- Introducing the PIC16F1XXX
- Migration to the PIC16F1XXX
- New Coding Tricks
- **Advanced Topics**

Advanced Topics

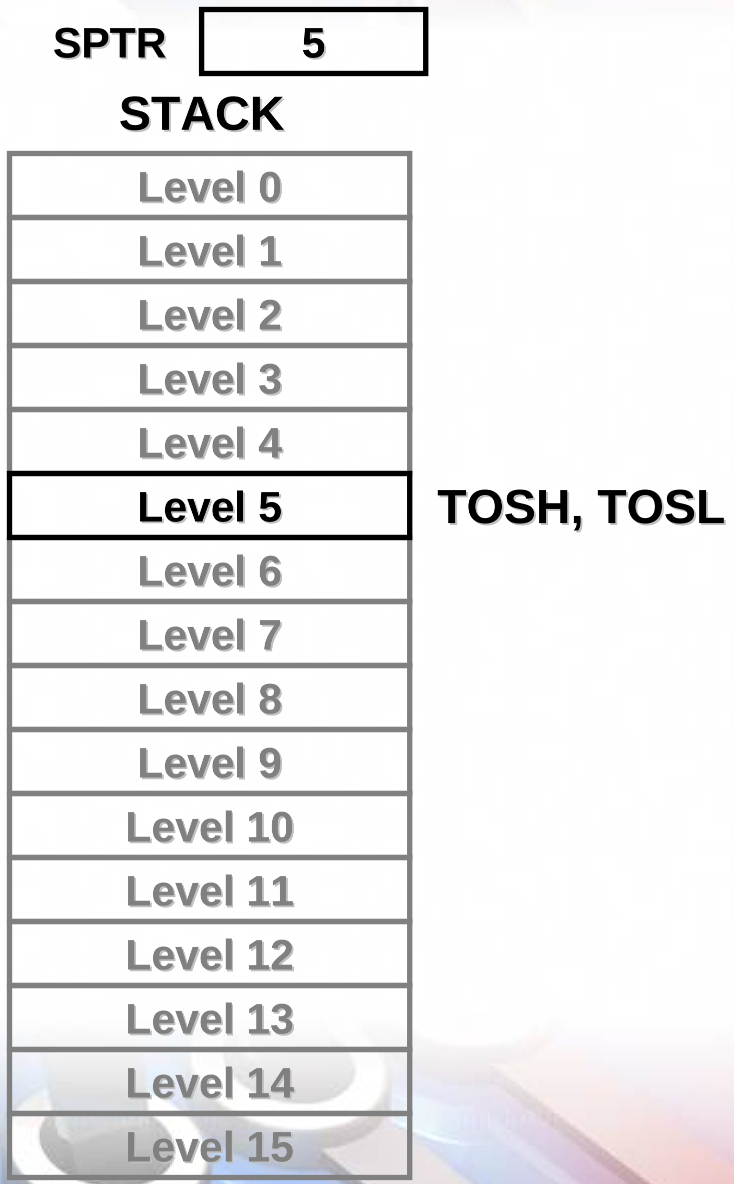
- Accessing the Stack
- Accessing the Context Shadows
- Reading the Device ID
- Preemptive Multitasking
- Error Diagnostics

Accessing the Stack

- The stack is available through the TOS and SPTR registers
- SPTR is the current value of the stack pointer
- TOS points to the TOP of the STACK
- Both registers are read/writeable
- TOS is split into TOSH and TOSL due to the 15-bit size of the PC

Stack Access

- The SPTR always indicates the current stack position
- TOSH,TOSL is the stack at the SPTR position
- Changing SPTR will move TOSH,TOSL
- SPTR is 5 bits



Accessing the Context Save Shadow

- The interrupt context save registers are read/writeable in bank 31

STATUS	STATUS_SHAD
FSR0 Low	FSR0L_SHAD
FSR0 High	FSR0H_SHAD
FSR1 Low	FSR1L_SHAD
FSR1 High	FSR1H_SHAD
BSR	BSR_SHAD
WREG	WREG_SHAD
PCLATH	PCLATH_SHAD

MASTERS 08 CONFERENCE

Device ID

- Select registers in the configuration memory are now accessible via the EE interface
- The User ID, Device ID and Configuration word can be read

Preemptive Multitasking

- Access to the stack and shadows allows an interrupt to replace the current task with a second task
- This allows easier RTOS programming for these devices

Error Diagnostics

- Access to the stack and context shadows also allows more extensive self check
- Stack verification is critical for some safety critical applications

Lab 4 (Advanced)

- Build a simple RTOS kernel
- A hint is in the 1204/lab4 directory

MASTERS 08 CONFERENCE

Trademarks

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

FilterLab, Linear Active Thermistor, 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, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC32 logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, 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.

© 2008, Microchip Technology Incorporated. All Rights Reserved.