11026 C30

Advanced Features in MPLAB® C30

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Prerequisites/Goals

You should be familiar with Microchip's 16-bit architecture and have had some basic introduction to MPLAB® C30

We will provide some advanced information that will help you get what you need from the tools to use the advanced features of the architecture

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Objectives

- **Today you will learn about…**
  - Mixing C and assembly
  - Allocating many types of memory
  - Accessing data in Flash
  - CodeGuard™ Security Support

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Agenda

- **Mixing C & Assembly**
  - Overview
  - Calling assembly functions from C
  - Inline Assembly
- **Memory Models**
- **Program Space Visibility**
- **CodeGuard™ Security Support**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Mixing C and Assembly

Reasons to use assembly:

- **Architecture requirements:**
  - precise timing
  - to generate specific code sequences
  - to generate instructions not supported by compiler

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Mixing C and Assembly

- **Writing complete assembly function**
  - call an assembly routine from C

    or call a C routine from assembly
  - key topics:
    - **calling conventions and register usage**
    - **stack usage**
- **Writing inline assembly**
  - how to reference C variables

© 2007 Microchip Technology Incorporated. All Rights Reserved.

UNIVERSITY OF MICROCHIP U OF M MASTERs 2007

Mixing C and Assembly

Which kind should I use?

assembly function	inline asm
• long sequence	• short sequence
• call cost is minor	• call cost too great
• limited references to C data	• refers to C data
• control flow allowed	• no control flow

© 2007 Microchip Technology Incorporated. All Rights Reserved.

External Asm Functions

- **MPLAB® ASM30 reference: DS51317**
  - assembler syntax
- **dsPIC30F/33F Programmer's Reference Manual: DS70157**
  - assembly language instruction set
- **MPLAB C30 User's Guide: DS51284**
  - calling convention, chapter 4

© 2007 Microchip Technology Incorporated. All Rights Reserved.

University of Microchip UofM MASTERs 2007

External Asm Functions

- **Basic form of an assembly file:**

```
.section my_code, code
.global _myfunction
; myfunction is externally visible
; and starts here!
_myfunction:
 clr w0
 ; and so on
```

External Asm Functions

- How can I call it from C?
- First, declare the function `extern`

```
extern void myfunction(void);
```

- Then call it as a normal function!

```
void main(void) {
   myfunction();
   /* and so on */
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

UNIVERSITY OF MICROCHIP
U OF M
MASTERs 2007

Calling Convention

- **Parameters passed in W0 to W7**
  - parameters are placed in the first *properly aligned* register(s)
  - starting from the left-most parameter
  - if there are enough registers to hold the entire object
- **Additional parameters are pushed onto the stack**
  - right-most unallocated parameter is pushed *first*

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Calling Convention

- **Values returned in W0**
  - and W1 to W3 if required
- **A called function can use W0-W7 without preserving them**
  - Upon return to the calling function, these registers need not hold the same values
  - W8-W15 **must** be preserved
- **There are no unused registers**
- **ISRs – preserve all used registers!**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Register Alignments

- **What does *properly aligned* mean?**

Type	registers required	alignment
char	1	none
short	1	none
int	1	none
data pointer	1	none
long	2	even
float	2	even
long long	4	divisible by 4
long double	4	divisible by 4
structure	1 per 2 bytes	none

© 2007 Microchip Technology Incorporated. All Rights Reserved.

UNIVERSITY OF MICROCHIP
U OF M
MASTERs 2007

Inline Assembly

- **MPLAB® C30 User's Guide: DS51284**
  – syntax and guidelines, chapter 8

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **Two forms available**
- ***Simple:***

```
asm("assembly text");
```

- ***Complex:***

```
asm( "template" :
    "format"(variable),... :
    "format"(variable),... :
    "clobbers");
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **The complex version is, well, complex! Why use it?**
  - if the assembly instruction **uses any** register
  - if you need to access any C variable
- **Failure to understand this will cause unpredictable results**
  - programs will fail under changing circumstances

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **What is the format string for?**
  - identifies the kind of operand needed
  - constrains the variable to the correct format
  - Examples, "r" – a register or
    "m" – a memory address

```
asm( "template" :
   "format"(variable),... :
   "format"(variable),... :
   "clobbers");
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **A format string can include extra information, such as:**
  - read only operand
  - write only operand
- **Output (write) operands are listed 1st**

```
asm( "template" :
    "format"(variable),... :// outputs
    "format"(variable),... :// inputs
    "clobbers");
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **What are clobbers?**
  - Some instructions will implicitly modify the value stored in a register
    - **the compiler needs to know when this happens**

```
asm( "template" :
   "format"(variable),... :
   "format"(variable),... :
   "clobbers");
```

Inline Assembly

- **What is the template?**
  - mostly it is assembler text
  - special symbols **`%0,%1,...%n`** can refer to arguments
    - **these can be modified `%d1`**

```
asm( "template" :
   "format"(variable),... :
   "format"(variable),... :
   "clobbers");
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

- **Examples:**

```
asm ( “add %1, #%2, %0” :
        “=r”(result)       :
        “r”(input), “i”(CONSTANT) );

asm ( “mul.su %1, #%2, %0 :
        “=r”(result)          :
        “r”(input), “i”(CONSTANT) );
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Inline Assembly

```
#define CONSTANT 10
int add(int input) {
  int result;
  asm ( “add %1, #%2, %0” :
        “=r”(result)      :
        “r”(input), “i”(CONSTANT) );
  return result;
}
```

Inline Assembly

```
#define CONSTANT 10
long mulsu(int input) {
  long result;
  asm ( “mul.su %1, #%2, %0” :
        “=r”(result)          :
        “r”(input), “i”(CONSTANT) );
  return result;
}
```

Agenda

- **Mixing C & Assembly**
- **Memory Models**
  - 16-bit architecture review
  - Application use
- **Program Space Visibility**
- **CodeGuard™ Security Support**

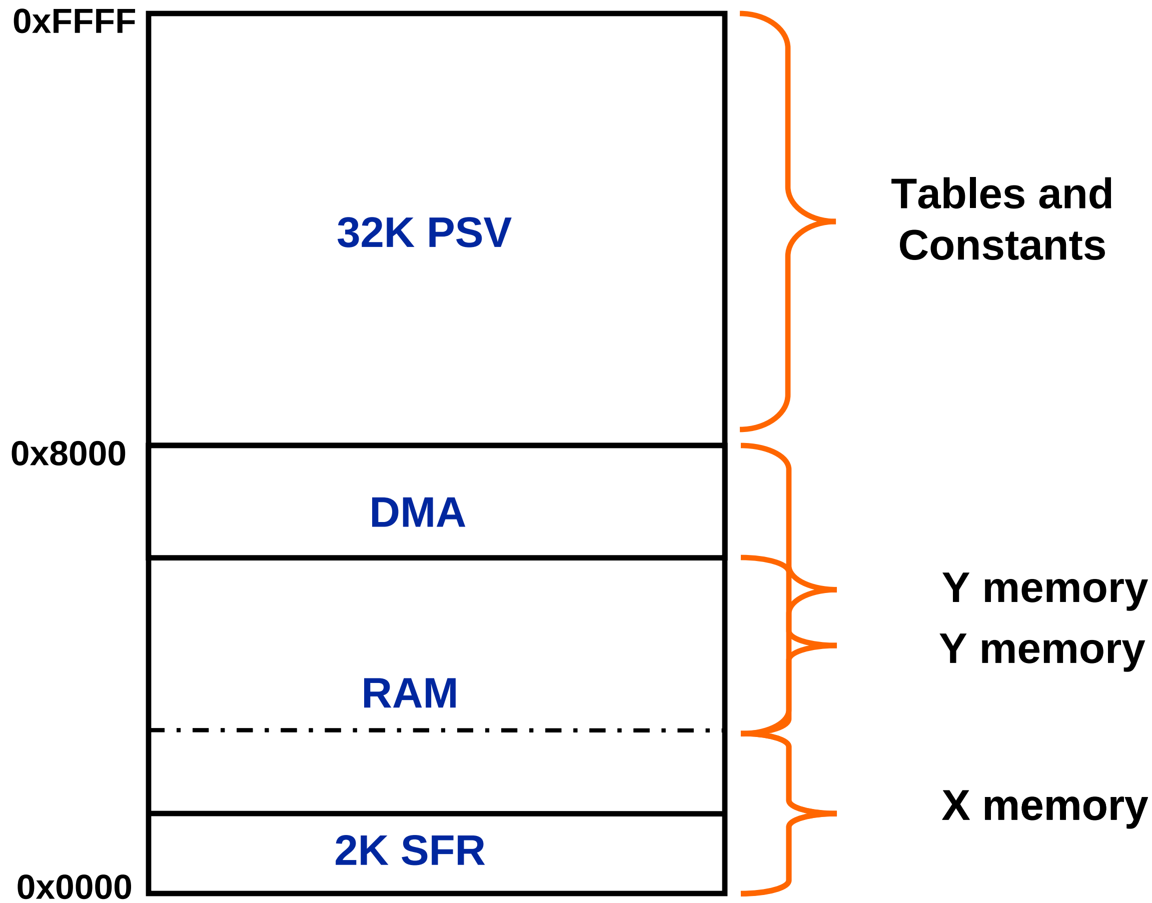
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Memory Architecture

- **Harvard Architecture**
  - DATA RAM
    - **16 bits wide**
    - **16 bits of address**
  - PROGRAM Flash
    - **24 bits wide**
    - **23 bits of address**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Data Space Memory Map



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Program Space Memory

- **Stores executable instructions**
- **Device configuration fuses**
- **EEPROM Data**
- **Can store DATA also**
  - accessed via Program Space Visibility (PSV) window
  - accessed via specialized read instructions
- **Reprogrammable during execution**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

UNIVERSITY OF MICROCHIP U OF M MASTERs 2007

Application Use

- **Why care about memory layout?**
  - Different data memory requirements:
    - **DSP uses X Memory or Y Memory**
    - **DMA uses the DMA memory**
  - Some instruction sequences are more efficient!
    - **Direct ALU access to near memory**
    - **Relative branches/calls faster**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Application Use

- **It's *easy* to constrain memory!**
- **C Global memory models**
  - via command-line options or IDE radio buttons
- **Individual memory settings**
  - via C or assembly attributes

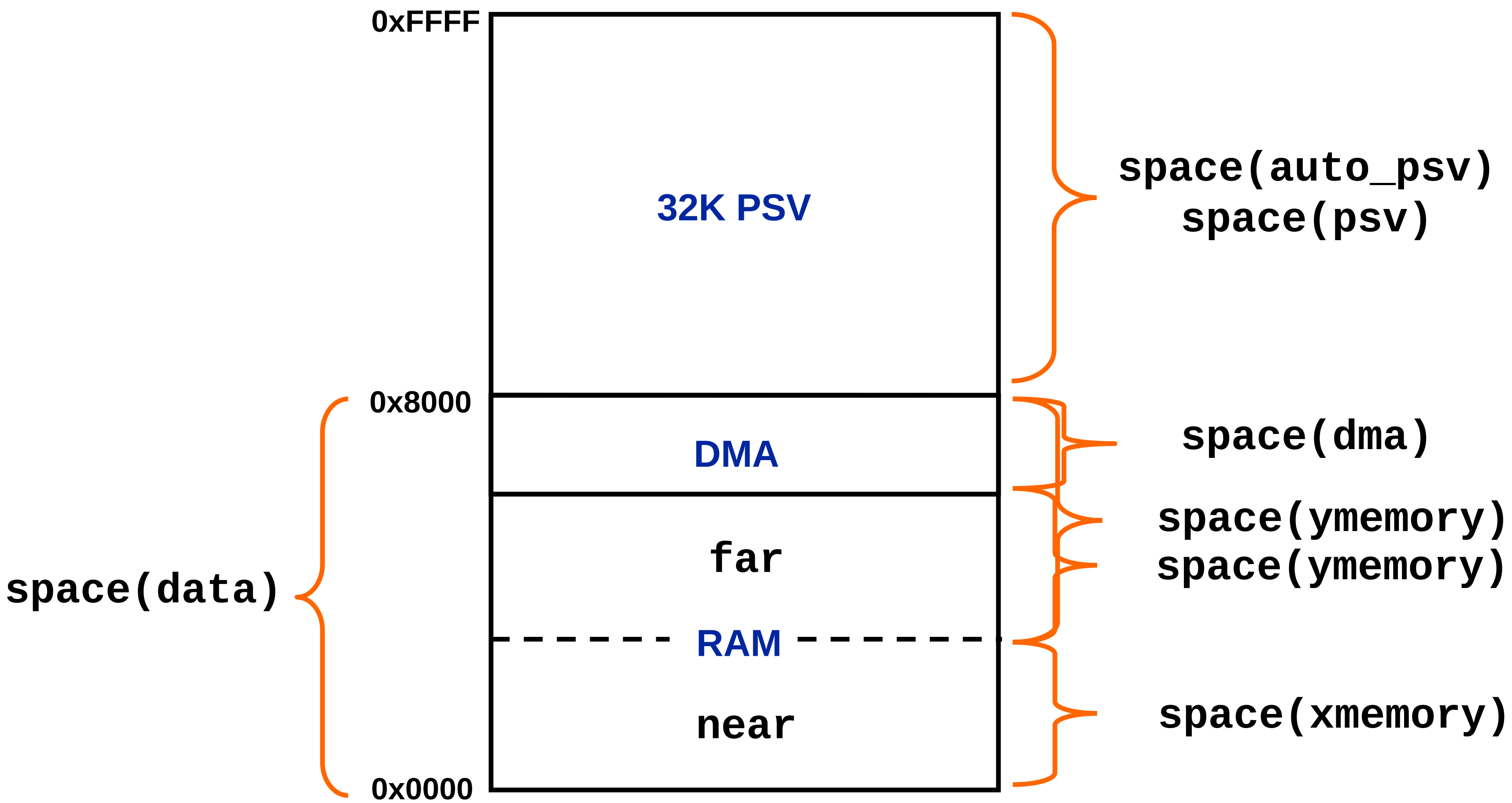
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Memory Models

- **MPLAB® ASM30 reference: DS51317**
  - assembler section directive, chapter 6.3
- **dsPIC30F Family Reference: DS70046**
  - hardware information
- **MPLAB C30 User's Guide: DS51284**
  - attributes, chapter 2.3
  - memory models, chapters 4.6,4.7,4.8

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Data Space Memory Map



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Default Memory Models

- **Data locations**

  Scalar variables: near data

  Aggregate variables: near data

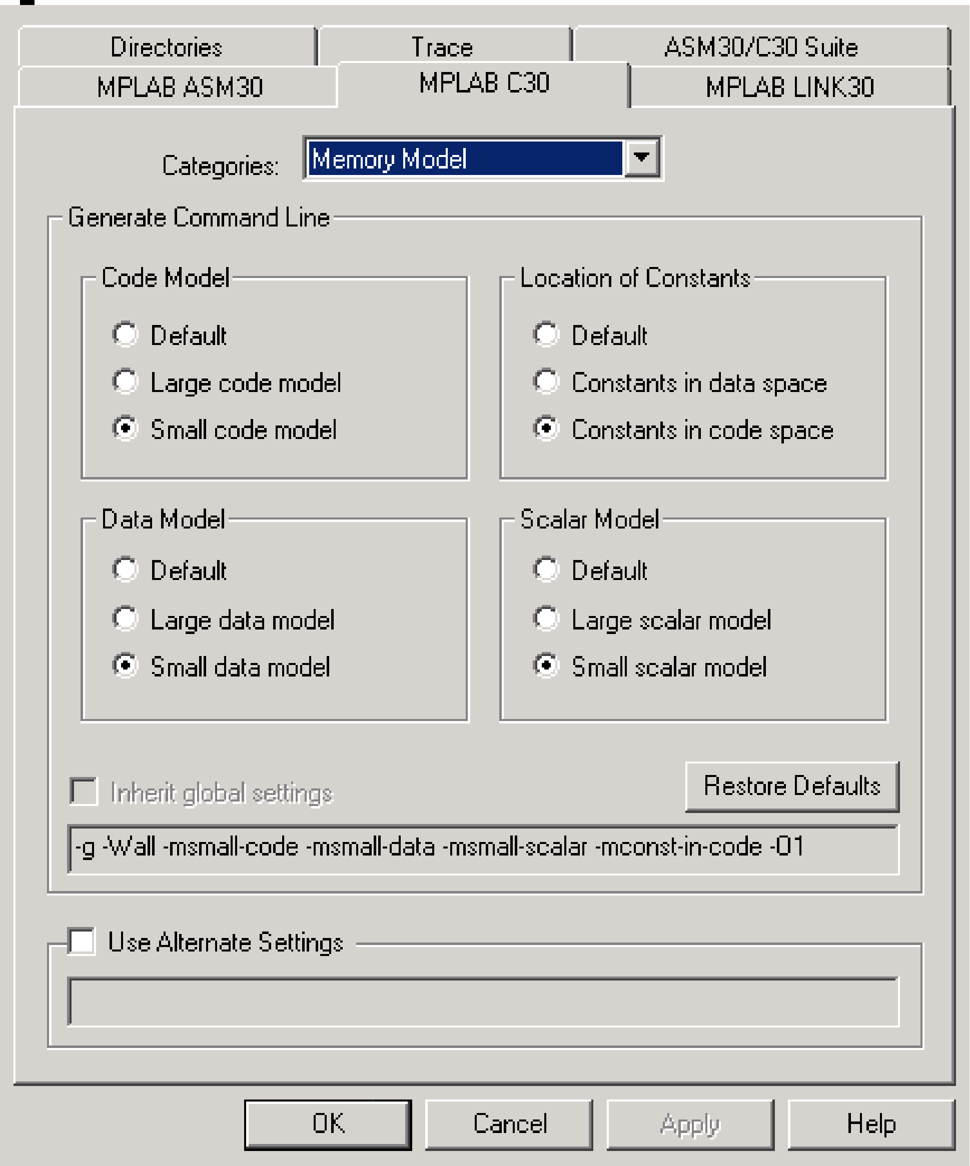
  Constants: automatic PSV

- **Functions**

  Small code model by default

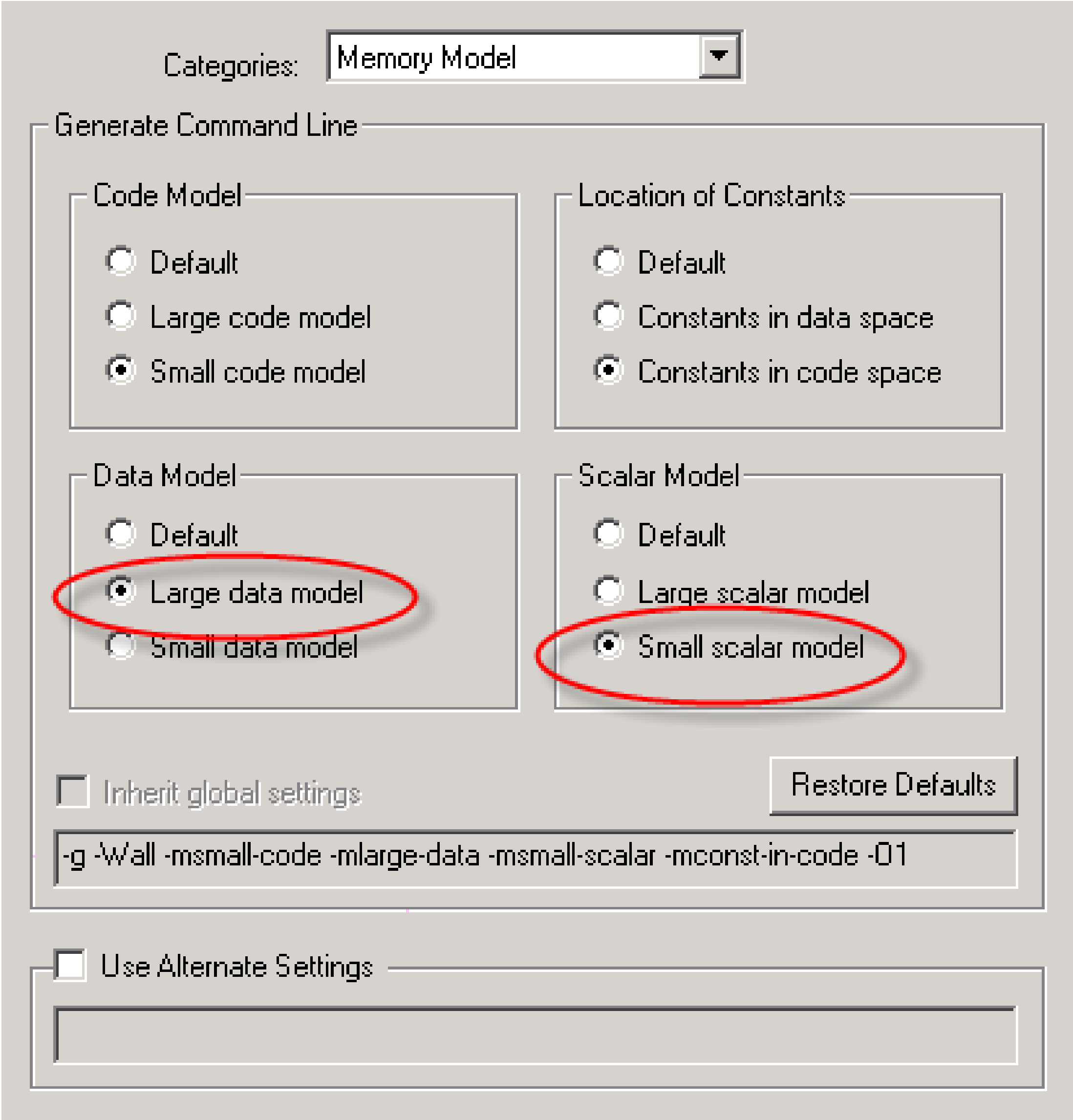
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Equivalent Model Settings



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Better Model Settings?



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Individual Memory Settings

- **Global settings cannot place variables into X or Y memory!**
- **Add attributes to declarations:**

```
int my_data[256]
    __attribute__((space(xmemory)));
int more_data[1024]
    __attribute__((space(dma)));
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

C Target Memory Attributes

- `__attribute__((space(area)));`

  where *area* is:
  - `data` - general data
  - `auto_psv` - compiler managed PSV
  - `psv` - user managed PSV
  - `xmemory` - data memory (X)
  - `ymemory` - data memory (Y)
  - `dma` - DMA memory
  - `eedata` - EEDATA memory
  - `prog` - program FLASH

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Asm Target Memory Attributes

- `.section` ***name, area***

  where *area* is:
  - **data** - initialized data memory
  - **bss** - zeroed data memory
  - **psv** - user managed PSV
  - **xmemory** - data memory (X)
  - **ymemory** - data memory (Y)
  - **dma** - DMA memory
  - **eedata** - EEDATA memory
  - **code** - program FLASH

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Individual Assembly Settings

```
 .section *,bss,xmemory
 .global _my_data
_my_data:
 .space 512

 .section *,bss,dma
 .global _more_data
_more_data:
 .space 2048
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Miscellaneous Attributes

- **`aligned()`** - start align boundary
- **`reverse()`** - end align boundary
- **`near`** - near data (1st 8K)
- **`far`** - far data (anywhere)
- **`address()`** - start address
- **`persistent`** - uninitialized on warm reset
- **`section`** - give a specific section name great for grouping (common PSV variables)

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Agenda

- **Mixing C & Assembly**
- **Memory Models**
- **Program Space Visibility**
  - Overview
  - Three modes of operation
- **CodeGuard™ Security Support**

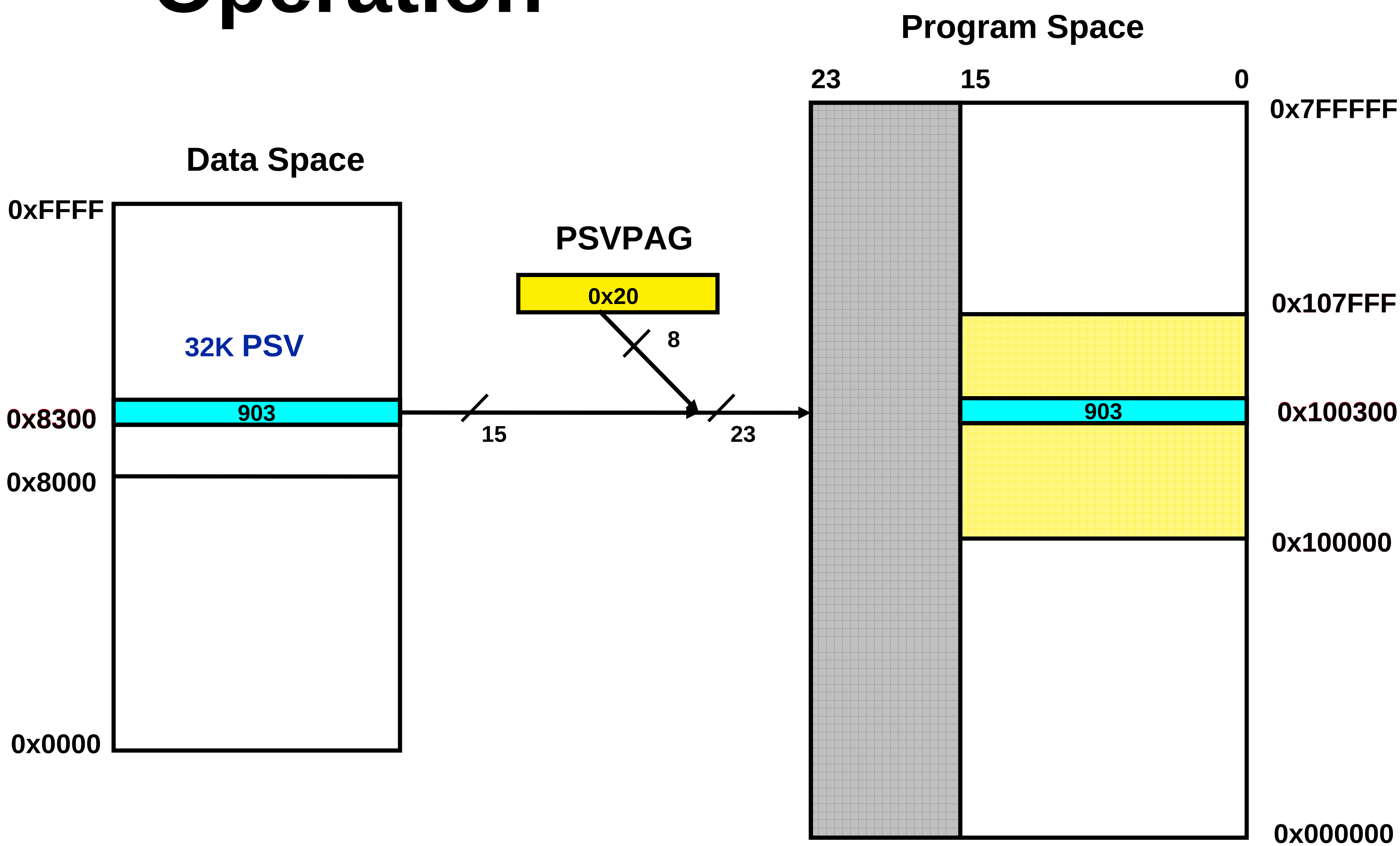
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Program Space Visibility

- **PSV offers a single 32K data window into Program Flash**
- **When enabled, it is mapped into data space from 0x8000 to 0xFFFF**
- **The compiler supports 3 modes of usage for the PSV window**
  - User managed PSV support
  - Auto PSV mode
  - Compiler managed PSV

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Program Space Visibility Operation



© 2007 Microchip Technology Incorporated. All Rights Reserved.

PSV Usage

- **MPLAB® ASM30 reference: DS51317**
  - special operators, chapter 4.5
- **dsPIC30F Family Reference: DS70046**
  - hardware information
- **MPLAB C30 User's Guide: DS51284**
  - PSV info, chapter 4.15
  - Built-in function info, chapter B.2

© 2007 Microchip Technology Incorporated. All Rights Reserved.

User Managed PSV

- **The tool chain does nothing for you!**
- **You must:**
  - Place data into Program Flash
    - `space(psv)`
  - Enable the PSV window
    - `CORCONbits.PSV = 1;`
  - Configure the PSV page
    - `PSVPAG = ???;`

© 2007 Microchip Technology Incorporated. All Rights Reserved.

User Managed PSV Example

```
int data[256]
    __attribute__((space(psv)));

main() {
  CORCONbits.PSV = 1; // enable PSV
  PSVPAG = __builtin_psvpage(&data);
  // now safe to access data[]
  if (data[26] == 3) {
  }
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Auto PSV

- **The tool chain does (almost) everything for you!**
  - One 32K PSV page is supported
- **You must:**
  - Place data into Program Flash
    - **`space(auto_psv)`** or
    - Apply **`const`** to declarations
  - Tool chain enables PSV and sets the page

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Auto PSV Example

```
int data[256]
    __attribute__((space(auto_psv)));

main() {
  // now safe to access data[]
  if (data[26] == 3) {
  }
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Managed PSV

- **The tool chain does (almost) everything for you!**
  - Many 32K PSV pages are supported
- **You must:**
  - Place data into Program Flash
    - `space(psv)` or
    - `space(prog)`
  - Identify declaration as managed
  - Tool chain enables PSV

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Managed PSV Example

```
__psv__ int data[256]
    __attribute__((space(psv)));

main() {
  // now safe to access data[]
  if (data[26] == 3) {
  }
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Managed PSV Detail

- **Two new type qualifiers added:**

  `__psv__` - object can't cross PSV page

  `__prog__` - object may cross PSV page

- **When applied to object, the compiler will set the PSV page before access**
- **In pointer declarations, can modify the pointed to object (just like `const`)**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Managed PSV Detail

- **Functions that take a pointer will not accept a managed PSV pointer!**
  - They are different
- **Our libraries do not currently accept managed PSV pointers**
  - `printf()` will not print a managed PSV string

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Managed PSV Examples

- **Managed PSV object**

```
__psv__ int foo
   __attribute__((space(psv)));
```

- **Pointer to object in managed PSV**

```
__psv__ int *foo_p = &foo;
```

  - pointer lives in data RAM

Managed PSV

- **Summary:**
  - Does not remove 32K data item limit
  - Pointers are larger to accommodate page information
  - Accesses are slower (page must be set)
  - Interrupt service routines may need modification
  - Beta support

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Agenda

- **Mixing C & Assembly**
- **Memory Models**
- **Program Space Visibility**
- **CodeGuard™ Security Support**
  - Boot, Secure Segments
  - Execution Control
  - Security Model

© 2007 Microchip Technology Incorporated. All Rights Reserved.

CodeGuard™ Security

- **What is CodeGuard Security?**
  - **A hardware feature that…**
    - Partitions memory into 2 or 3 segments
    - Controls visibility and execution between segments
- **How is it useful?**
  - **Allows multiple parties to share resources on a single chip**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

CodeGuard™ Security

- **What device families have CodeGuard Security?**
  - **dsPIC33F, PIC24H, and several dsPIC30F devices**
  - **But the language extensions are useful on any device!**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

CodeGuard™ Security

- **Documentation available at www.microchip.com/codeguard:**
  - Reference Manual
  - White Paper
  - Web Seminar

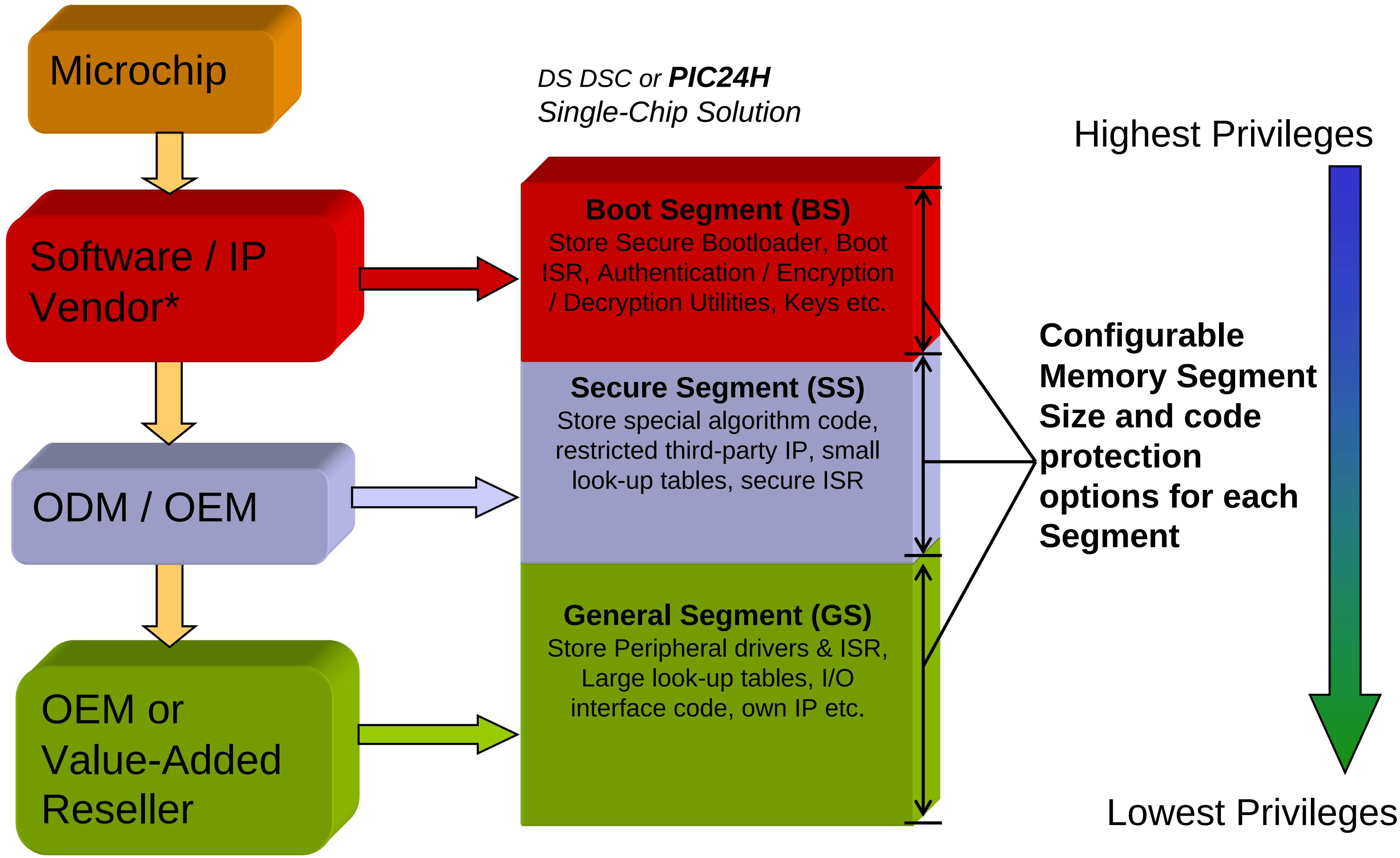
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Boot & Secure Segments

- **Memory can be partitioned**
  - into 1 or 2 special segments
  - plus the general segment
- **Segment sizes can be small, medium, or large (varies by device)**
- **Each segment can be linked separately**

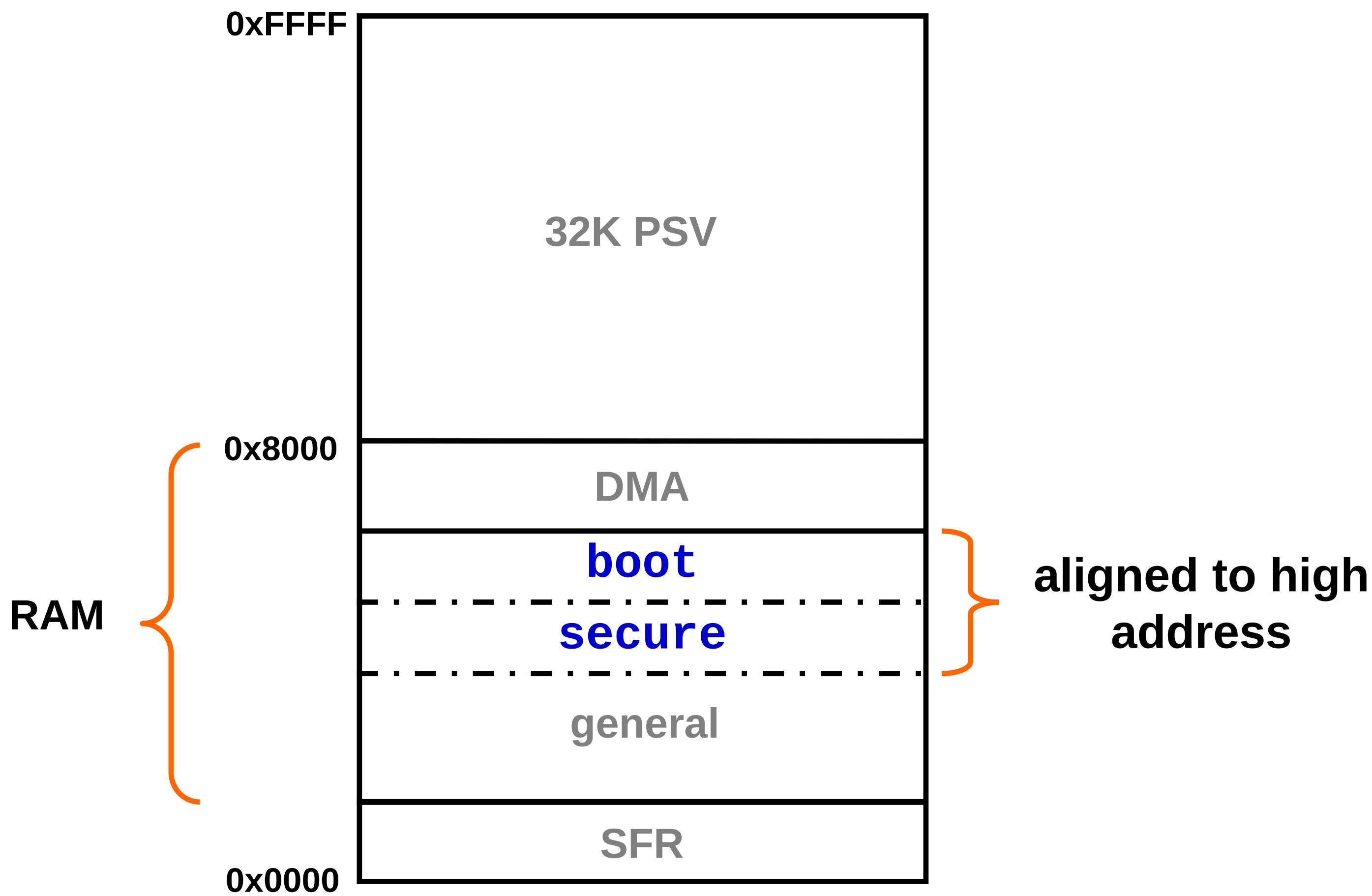
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Boot & Secure Segments



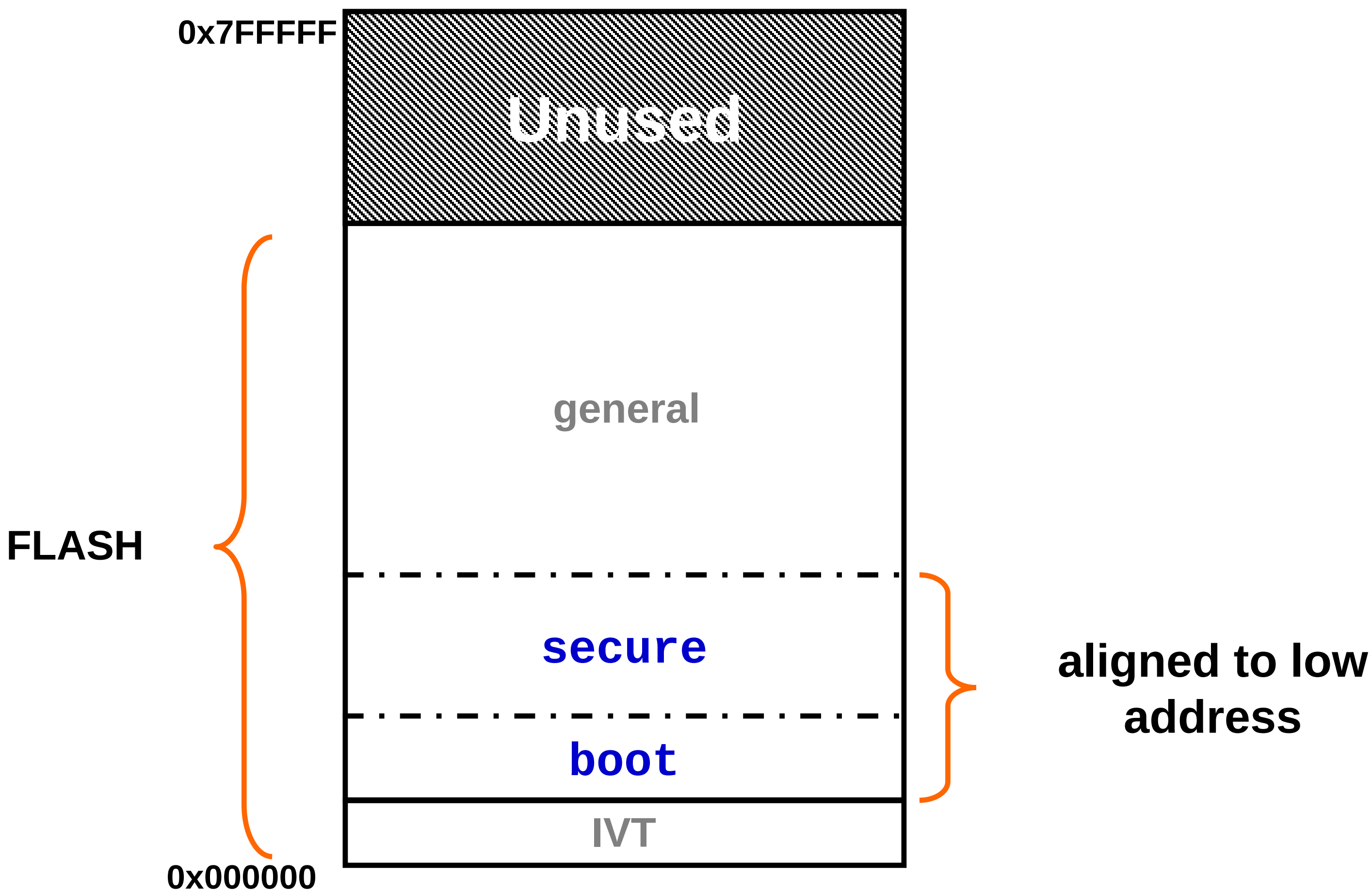
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Segments in Data Space



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Segments in Program Space



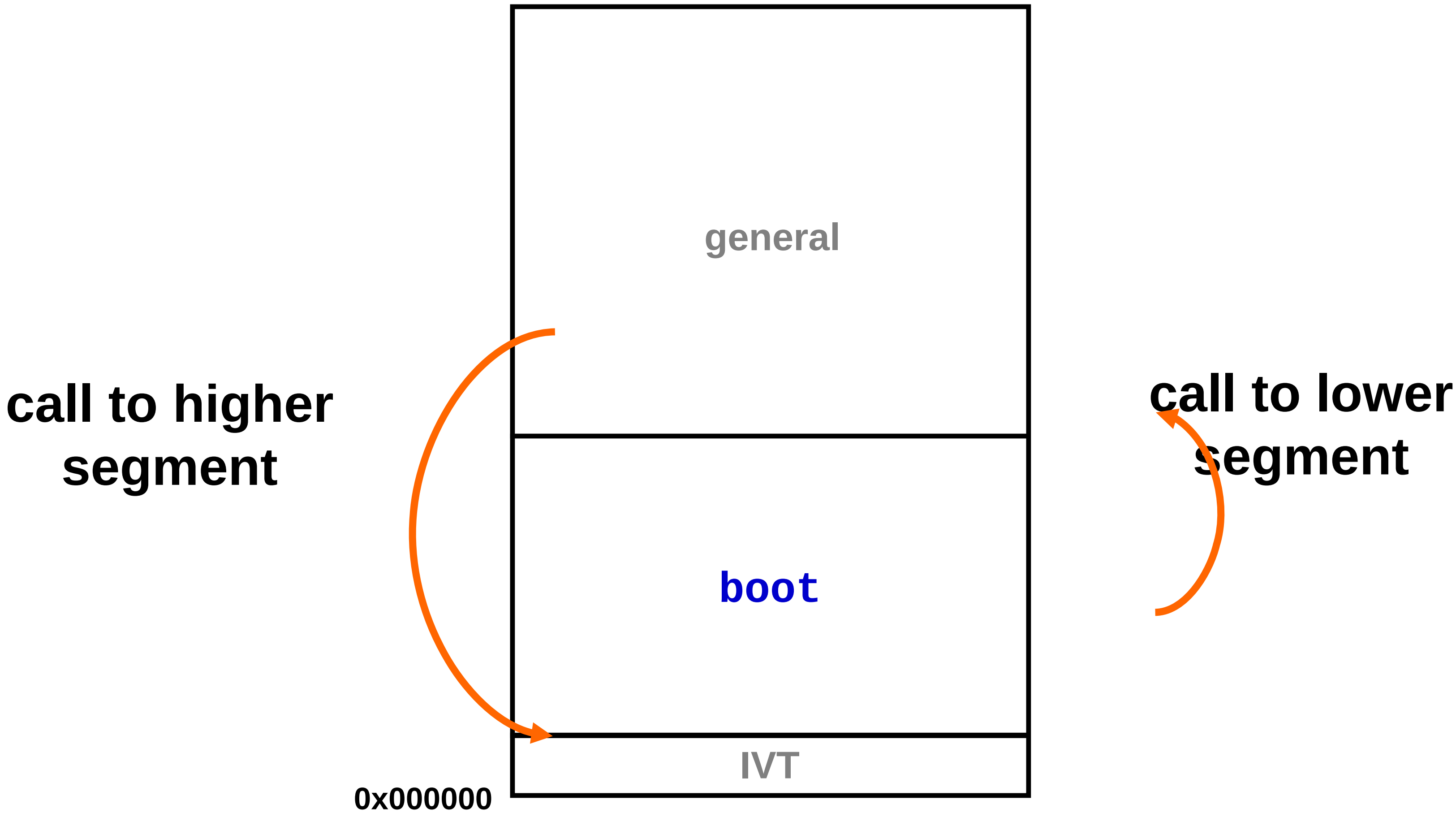
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Execution Control

- **Higher privilege segments can always call lower segments**
- **Standard Security**
  - Calls to higher segment are permitted
- **High Security**
  - Calls to higher segment must be vectored through access area

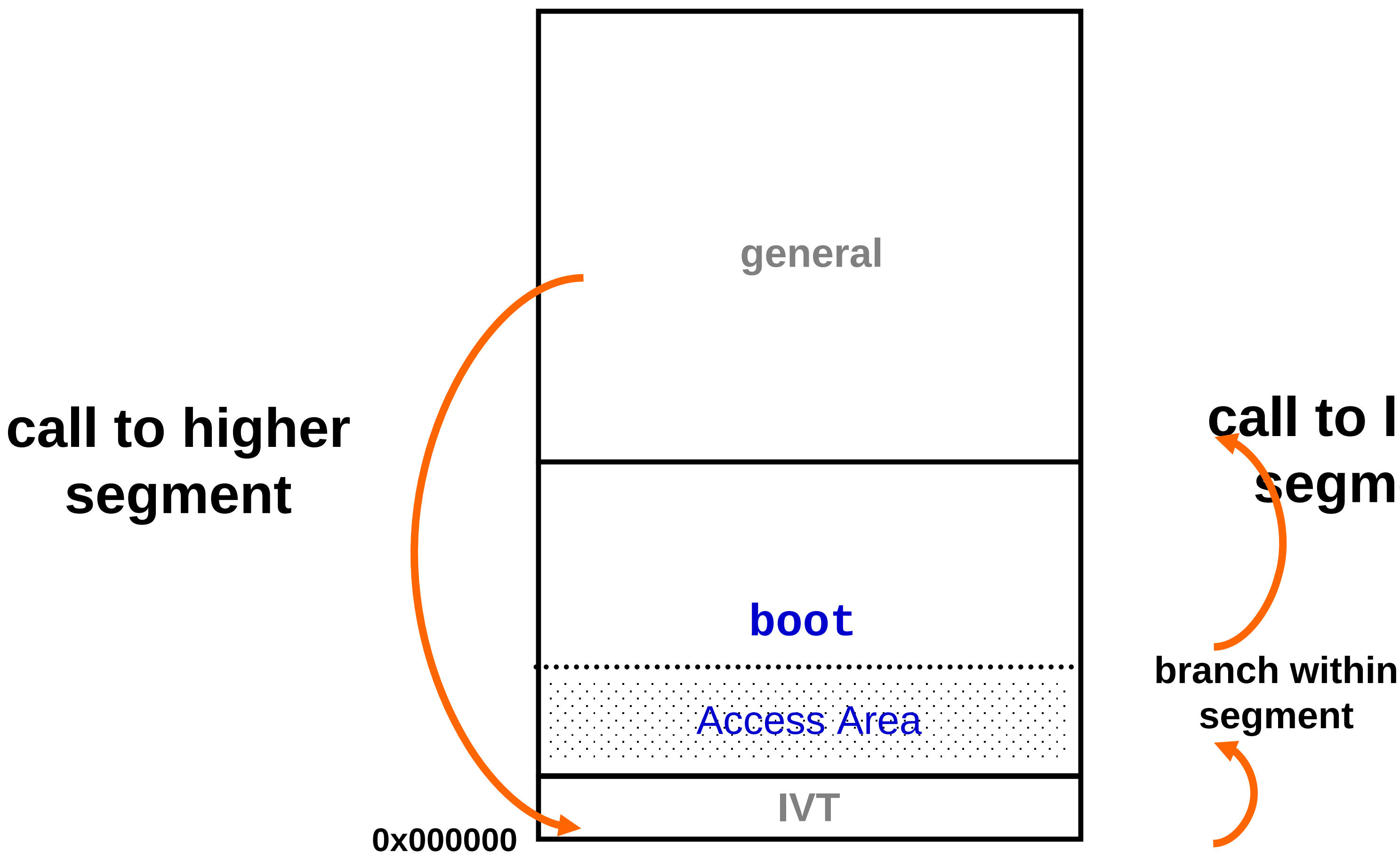
© 2007 Microchip Technology Incorporated. All Rights Reserved.

Execution Control: Standard Security



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Execution Control: High Security



© 2007 Microchip Technology Incorporated. All Rights Reserved.

Access Area

- **With high security:**
  - only the first 32 locations are accessible to a lower privilege segment
  - tools create access area and manage references automatically
  - implemented as a branch table

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Access Area

- **Transfer of control by slot number is the key to separately linked program segments**
- **Use these constructs on any device**
  - even without CodeGuard™ Security in hardware

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Access Area Example: C

- How can I call an access slot?
- First, declare the function

```
extern void __attribute__((boot(4)))
  myfunction(void);
```

- Then call it as a normal function!

```
void main(void) {
   myfunction();
   /* and so on */
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

C Example, cont.

- How can I define an access slot?
- Use the boot or secure attribute

```
void __attribute__((boot(4)))
  entry4(void)
 {
   /* insert code here */
 }
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Access Area Example: Asm

- How can I call an access slot?
- Use the boot or secure operator

```
call  boot(4)
rcall secure(2)
bra   cc,boot(8)

mov   #boot(5),w0 ; 16-bit address
call  w0
```

Asm Example, cont.

- How can I define an access slot?
- Use the boot or secure attribute

```
    .section *,code,boot(4)
    .global _entry4
_entry4:
    ; do something
    return
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Boot & Secure Interrupts

- While executing in a boot or secure segment, all interrupts vectored through access area
- All interrupt sources use a single access entry slot (16)

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Interrupt Example: C

- How can I define a boot interrupt handler?

```
void __attribute__((interrupt,boot))
  my_boot_isr(void) {
  /* insert code here */
}
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Interrupt Example: Asm

- **How can I define a boot interrupt handler?**

```
.section *,code,boot(isr)
    .global _my_boot_isr
_my_boot_isr:
    ; do something
    retfie
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Security Model

- **Segment sizes and options are encoded into 3 config words:**
  - FBS: boot segment
  - FSS: secure segment
  - FGS: general segment

- **Together, these settings comprise the 'Security Model'**

Security Model Example: C

- **Define in source code**

```
#include <p33Fxxxx.h>

_FBS(BSS_SMALL_FLASH_HIGH &
  BRWP_WRPROTECT_ON);
_FSS(SSS_MEDIUM_FLASH_STD);
_FGS(GWRP_OFF);
```

- **Or use the IDE**
  - Build Options:LINK30:Code Guard

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Security Model Example: ASM

- **Define in source code**

```
.include "p33Fxxxx.inc"

config _FBS, BSS_SMALL_FLASH_HIGH &
    BRWP_WRPROTECT_ON
config _FSS, SSS_MEDIUM_FLASH_STD
config _FGS, GWRP_OFF
```

- **Or use the IDE**
  – Build Options:LINK30:Code Guard

© 2007 Microchip Technology Incorporated. All Rights Reserved.

User-Defined Security Model

- For devices without CodeGuard™ Security in hardware
- Use linker command options

```
--boot flash_size=128
--boot ram_size=64

--secure flash_size=256
--secure ram_size=64:flash_size=256
```

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Class Summary

- **Mixing C and Assembly**
- **Memory Models**
- **Program Space Visibility**
- **CodeGuard™ Security**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Additional Resources

- Microchip Web Site
  - Student Edition version of tool suite
  - C30 README file
  - 16-Bit reference material
  - Development boards
  - Silicon

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Q & A

- **Questions?**
- **More questions?**
  – Go to “Ask the Experts”
  – Visit the online forum!

© 2007 Microchip Technology Incorporated. All Rights Reserved.

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.

© 2007 Microchip Technology Incorporated. All Rights Reserved.