11034 MPL

Modular Coding Techniques using MPLINK™ Linker

Class Objective

- **When you finish this class you will:**
  - Understand what relocatable code is
  - Know advantages of relocatable code over absolutely located code
  - Combine code from 2 or more files to create a relocatable project
  - Create libraries using good coding practices and MPLIB™ object librarian

Agenda

- **Absolute and Relocatable Code**
  - Lab 1 - Migrating from absolute assembly
- **Creating Relocatable Assembly using MPLINK™ Linker**
- **Good Practices**
- **Common Errors**
  - Lab 2 - Creating a multi-file project
- **Creating and Using Libraries**
  - Lab 3 - Creating and Using Libraries

Absolute and Relocatable Code

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Absolute Code

- All code and data addresses must be explicitly defined.
- Projects consist of one "root" assembly source file
- All other assembly source files must be `#include`'d into root file
- Only root file can use END

Example of Absolute Code

modified 18F4620TEMP.ASM

```
        CBLOCK 0x080
        WREG_TEMP                   ; Context saving variable
        STATUS_TEMP                 ; Context saving variable
        BSR_TEMP                    ; Context saving variable
        ENDC

EXAMPLE EQU  0x000                  ; Define a variable

;*****************************************************************
        ORG    0x0018
        movff  STATUS,STATUS_TEMP           ; save STATUS
        movff  WREG,WREG_TEMP               ; Save WREG
        ...
        movff  WREG_TEMP,WREG               ; restore WREG
        movff  STATUS_TEMP,STATUS           ; restore STATUS
        retfie
```

Absolute Code

- Use **ORG** directives to specify starting location of program code in memory
- Use **EQU** statements to assign addresses

  Drawbacks

  - Must specify ***exact*** addresses for code and variables
  - Cannot be used with C18
  - Cannot be used with third-party libraries

Relocatable Code

- Organize program code and data into sections
- Reserve space with `RES`, `DB`, `DW` directives
- Memory allocation is handled by MPLINK™ Linker

Advantage

  - Flexible use of memory resources

Example of Relocatable Code

modified 18F4620TMPO.ASM

```
    UDATA
WREG_TEMP     RES 1          ;variable used for context saving
STATUS_TEMP   RES 1          ;variable used for context saving
BSR_TEMP      RES 1          ;variable used for context saving

    UDATA_ACS
EXAMPLE       RES 1          ;example of a variable in access RAM

;*****************************************************************
       CODE   0x0018
       goto   LowInt                  ;go to low priority ISR

PROGRAM_CODE  CODE
LowInt:
       movff  STATUS,STATUS_TEMP      ;save STATUS register
       movff  WREG,WREG_TEMP          ;save working register
       movff  BSR,BSR_TEMP            ;save BSR register
       …
```

Advantages of Relocatable Code

- Modularity
- Able to use third-party libraries
- Integrates with the MPLAB® C18 compiler
- Don’t need to specify addresses of variables or code
- For large, multi-file projects, Build Process can be significantly faster (USB, TCP/IP, MiWi™ protocol and Zigbee™ technology stacks)

Disadvantages

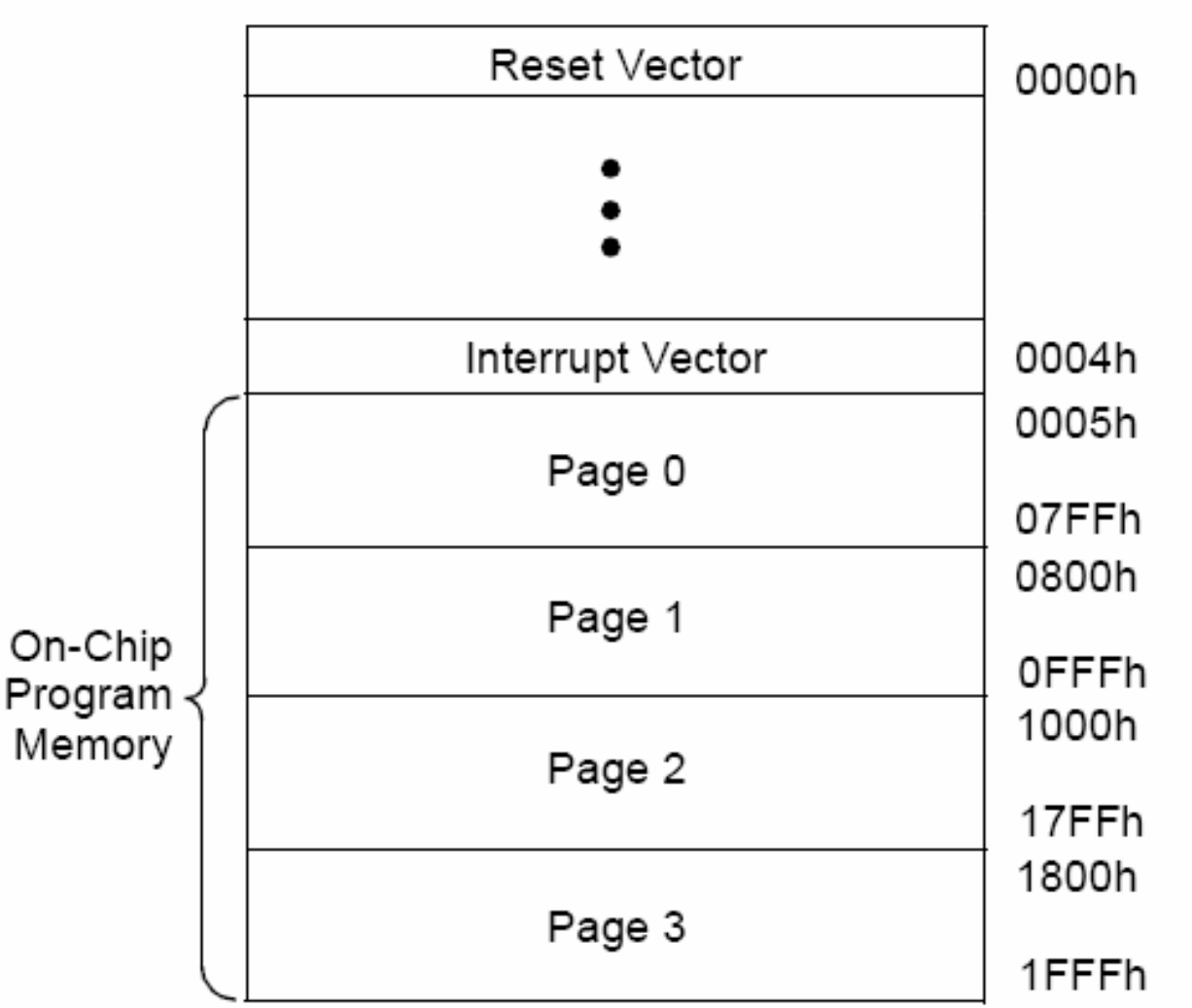
- **Migrating absolute code to relocatable code requires some effort**

Demo

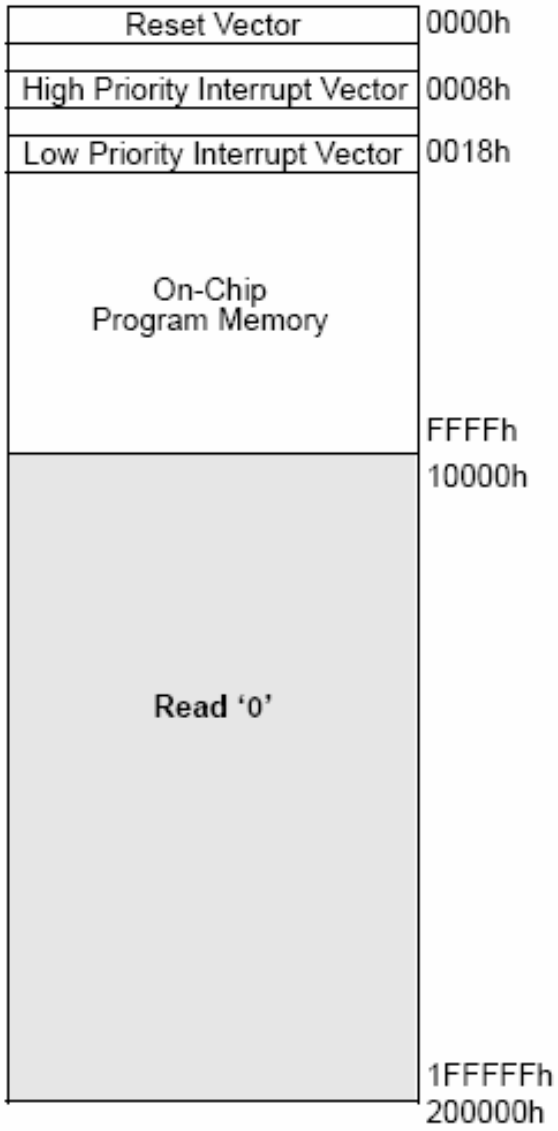
Building a Large Project

Memory Overview

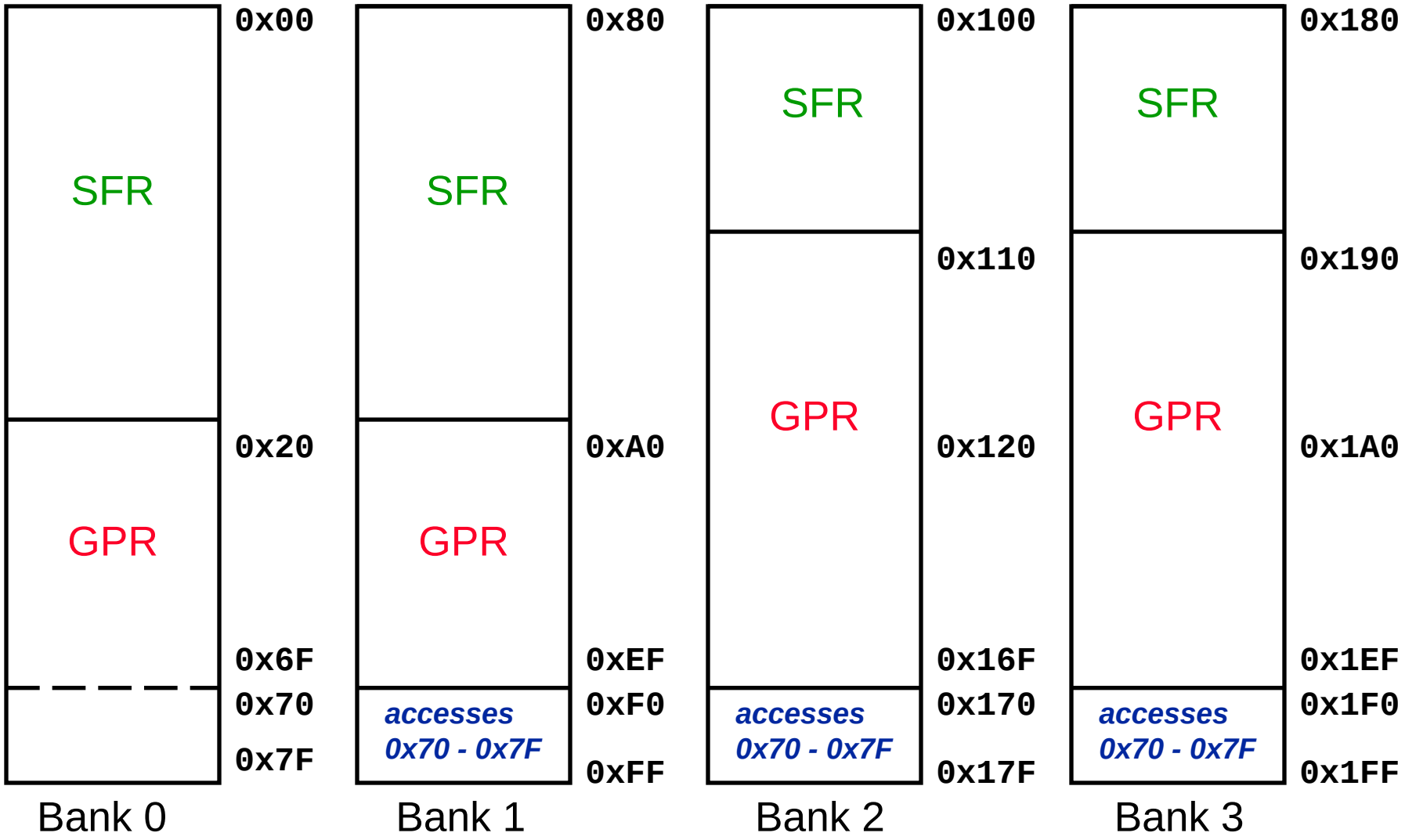
PIC16F887 Program Memory Map



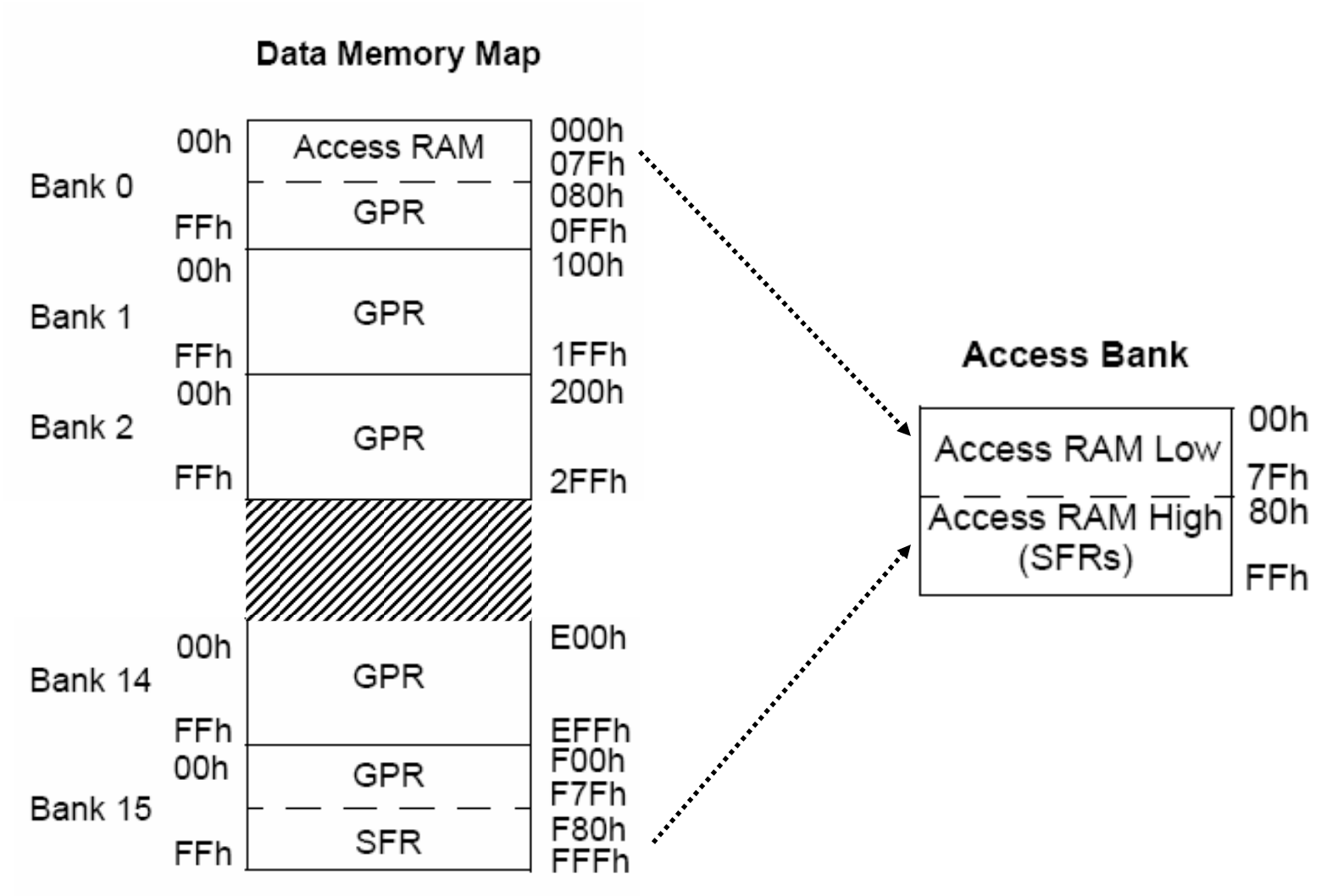
PIC18F4620 Program Memory Map



PIC16F887 Data Memory Map



PIC18F4620 Data Memory Map



Sections

Program Memory Sections

- **CODE**
  - Executable instructions
- **ROMDATA**
  - Constants in program memory

Data Memory Sections (Un-Initialized)

- **UDATA**
  - Uninitialized data in banked data memory
- **UDATA_OVR**
  - Uninitialized *overlay* data in banked data memory
- **UDATA_SHR**
  - Uninitialized data in unbanked data memory (non-PIC18 devices)
- **UDATA_ACS**
  - Uninitialized data in access RAM (PIC18 devices)
- **ACCESS_OVR**
  - Uninitialized *overlay* data in access RAM (PIC18 devices)

Data Memory Sections (Initialized)

- **IDATA**
  - Initialized data in banked data memory
- **IDATA_ACS**
  - Initialized data in access RAM (PIC18 devices)

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Declaring Sections

- **Syntax**

  `{Name}  <Section> {Address}`

  – name and address are *optional*

Note:	Two sections in the same source file are ***not*** permitted to have the same name.

Default Section Names

- For each assembly file, the assembler **creates a *default section*** of each type

Example: For file `foo.asm`

.<type>	.<type>_<filename>
Midrange Devices	**PIC18**
`.code`	`.code_foo.o`
`.idata`	`.idata_foo.o`
`.udata`	`.udata_foo.o`
`.romdata`	`.romdata_foo.o`

Declaring Variables

- **Syntax**

```
<name>   RES     <byte_size>
```

  - all fields are REQUIRED

```
DATA_TBL    UDATA
ONE_BYTE    RES     1
ONE_WORD    RES     2
```

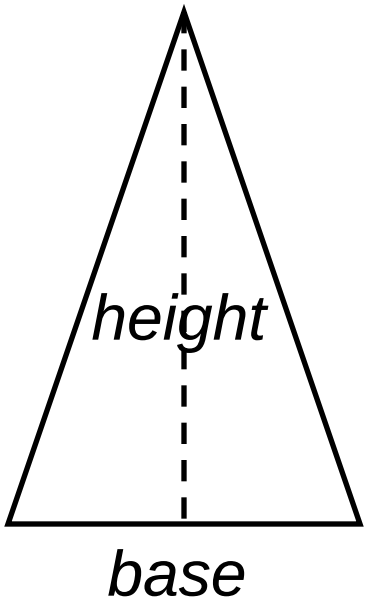
Declaring Variables (contd.)

- **Syntax**

```
<name>    DB     <init_value>
or
<name>    DW     <init_value>
or
<name>    RES    <byte_size>
```

  - **all fields are REQUIRED**

```
TRIANGLE    IDATA    0x130
BASE        DB       0x30
HEIGHT      DW       0x1000
AREA        RES      2
```



Section Examples

afile.asm

```
                udata_acs
Speed           res     1
Temperature     res     1

                idata
SetPoint        dw      0x1234

ISR_vector      code    0x0008
                goto    hi_ISR
```

IDATA Considerations

- **IDATA sections create a table of the initialization values in Program Memory**
- **Code is needed to copy this information into Data Memory**
- **This can be done using:**
  - C18 startup routines (C018i.o,C018iz.o)
  - IDASM16.ASM for Mid-range devices

Demo

Using IDATA

MPASM™ Assembler Linker Support Overview

Absolute Code

- **Use `ORG` directive to specify location of code in program memory**
- **Use `EQU` or `CBLOCK` directive to assign variable addresses**

Relocatable Code

- **Organize program code and data into sections**
  - Use `CODE` directive for Program Memory
  - Use `UDATA/IDATA` directives for Data Memory
- **Reserve space with `RES, DB, DW` directives**
- **Memory allocation is handled by MPLINK™ Linker**

Scope

Scope

- The part of a project where a symbol (variable or function) is "visible"
- Keeps symbols in different parts of the program distinct from one another
- In assembly, all symbols have file scope

Exporting Symbols

- Allows one module to access a subroutine or data from another module
- Export symbols from an MPASM™ assembler source file with the `GLOBAL` directive
- Format: `GLOBAL` *symName*

Importing Symbols

- Allows one module to access a subroutine or data from another module
- Import symbols into an MPASM™ assembler source file with the **EXTERN** directive
- Format: **EXTERN** *symName*

Importing/Exporting Symbols

declare.asm

```
   LIST   P=18F4620
   #INCLUDE "p18F1620.INC"

   GLOBAL mpy

mpy
   <subroutine code>
   return

   END
```

Exporting a symbol

reference.asm

```
   LIST   P=18F4620
   #INCLUDE "p18F1620.INC"

   EXTERN mpy

   call mpy

   END
```

Importing a symbol

Lab 1

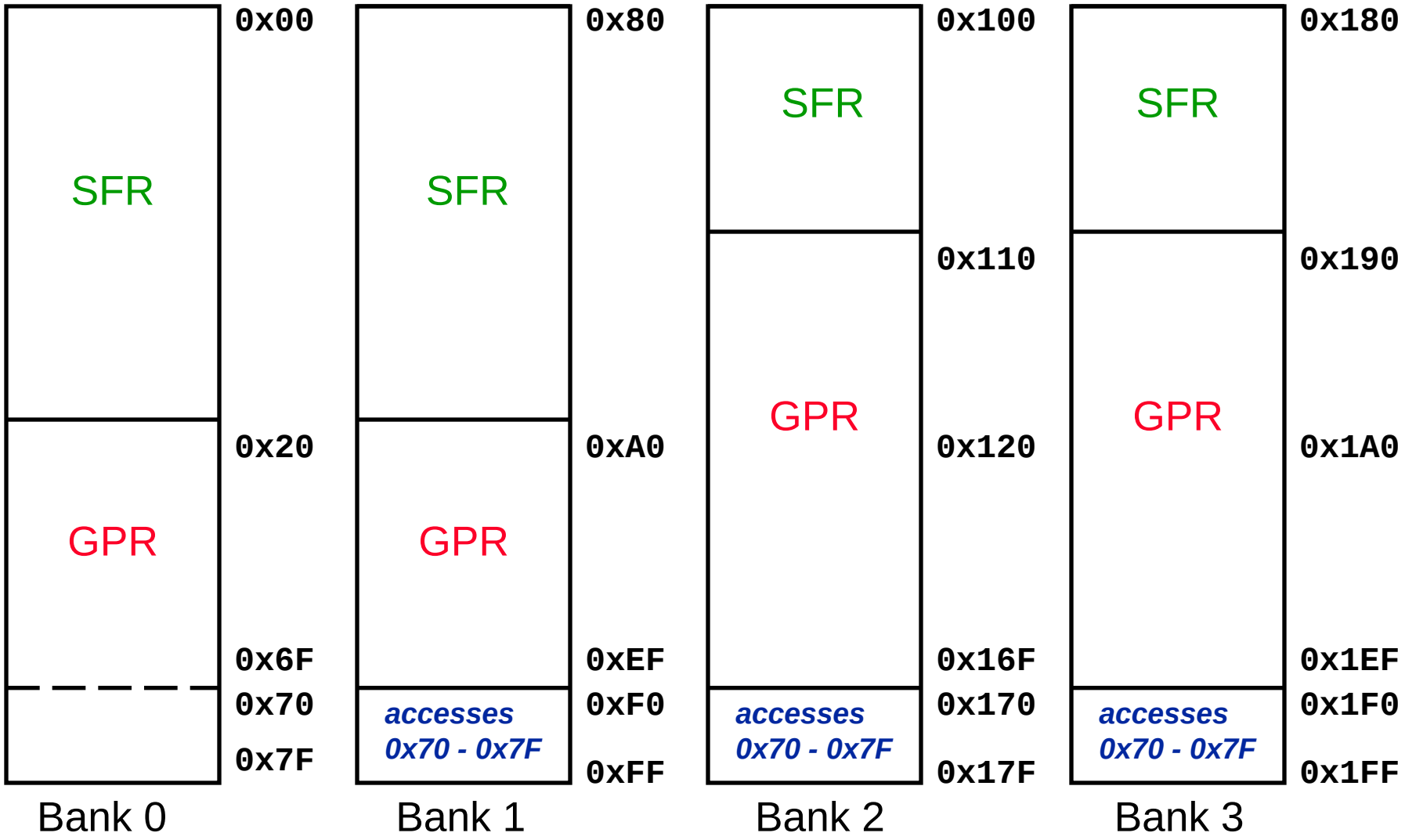
Migrating from Absolute Assembly Code

Lab 1
Migrating from Absolute Assembly

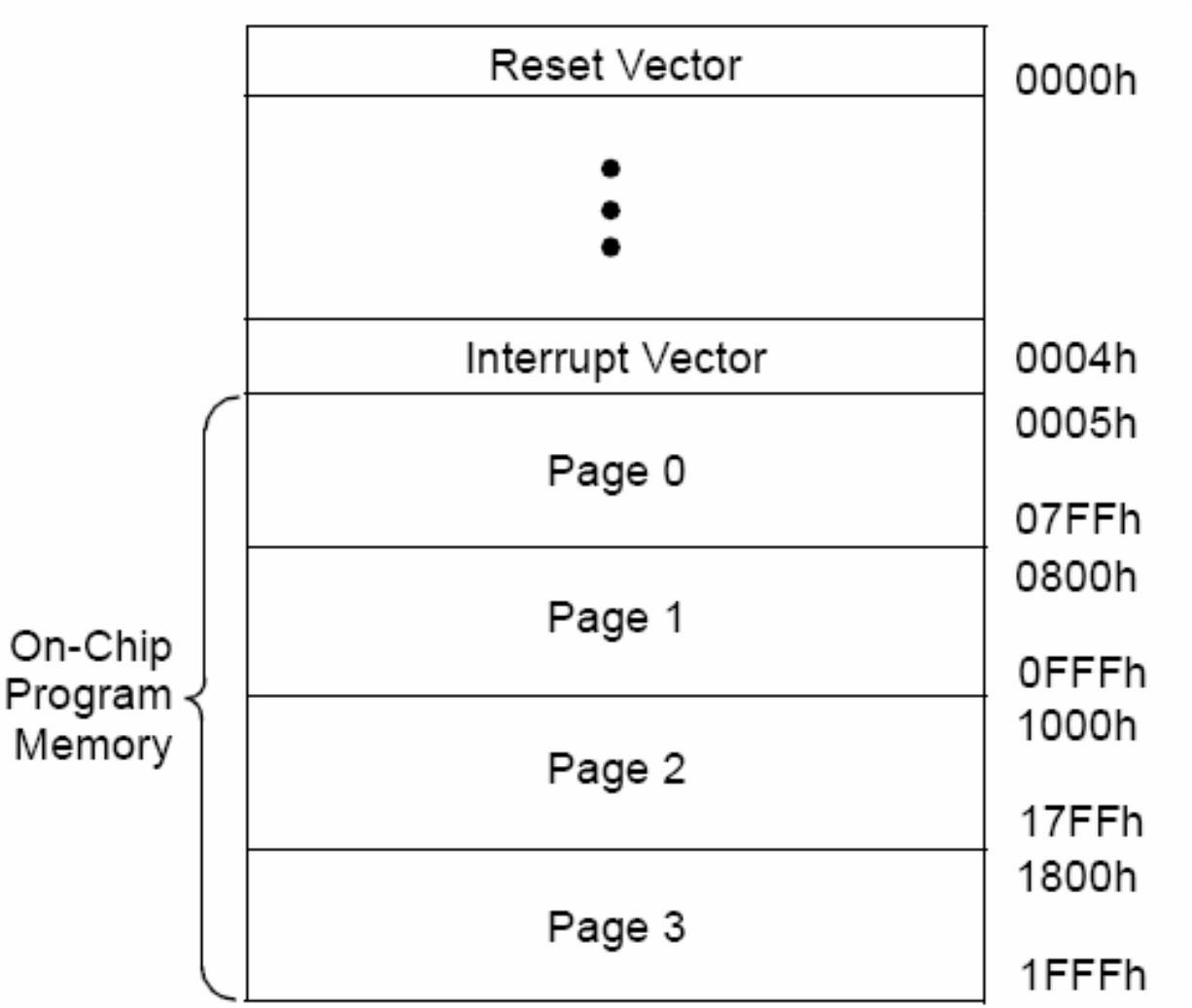
- **Reorganize program into sections**
  - CODE
    - **ORG → CODE sections**
  - DATA
    - **EQU → UDATA sections**
      RES statements
- **See handout for additional instructions.**

Banking and Paging

PIC16F887 Data Memory Map



PIC16F887 Program Memory Map



Banking and Paging Issues

- **Problem:**
  - Since sections are relocatable, we do not necessarily know where a section will reside
- **Question:**
  - How do we access these variables if we don't know what bank our variables are in or what page our code is in?

University of Microchip UofM MASTERs 2007

BANKSEL and PAGESEL

- We can use the **BANKSEL** and **PAGESEL** directives!
- These directives act like an internal macro

BANKSEL	PAGESEL
bank selection of Data Memory	page selection of Program Memory

BANKSEL

- **BANKSEL sets the bank selection bits to access the correct Data Memory location**

```
BANKSEL     MyVariable
clrf        MyVariable
```

might generate:

```
bcf         STATUS, RP0
bsf         STATUS, RP1
```

PAGESEL

- **PAGESEL sets the page (PCLATH or page select bits) to access the correct page of Program Memory**

```
PAGESEL     MyFunction
call        MyFunction
```

might generate:

```
bsf         PCLATH, 3
bcf         PCLATH, 4
```

BANKSEL and PAGESEL

```
; define processor
    list    p=16f887
; SFR and config bit definitions
    #include    <p16f887.inc>
    EXTERN      delay100

MAIN    CODE
start
    PAGESEL     delay100
    call        delay100
    ...
```

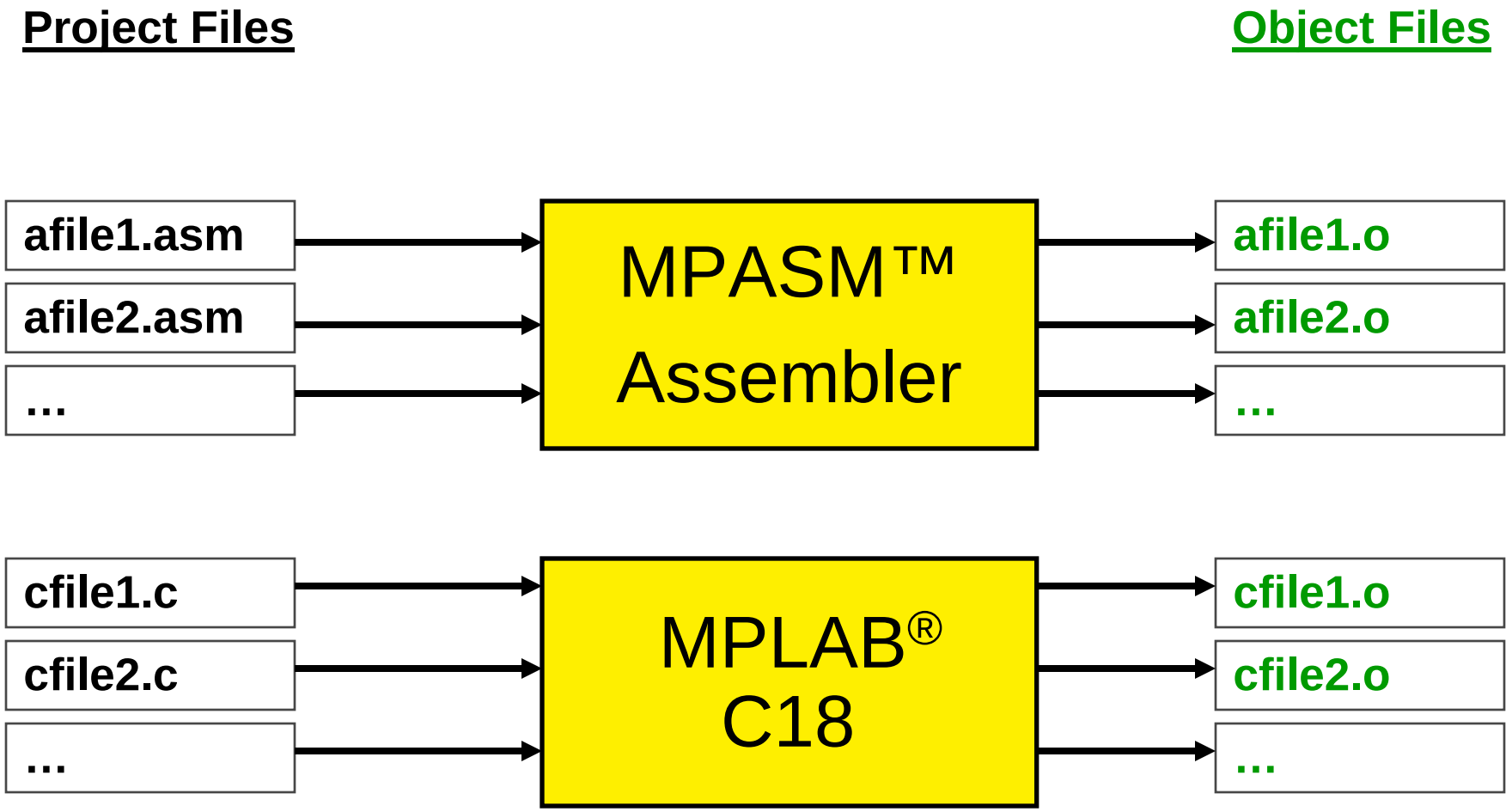
```
; define processor
    list    p=16f887
; SFR and config bit definitions
    #include    <p16f887.inc>
    GLOBAL      delay100

    UDATA
delay_count     RES 1

DELAY   CODE
delay100
    movlw       .101
    BANKSEL     delay_count
    movwf       delay_count
Lp: decfsz      delay_count, F
    goto        Lp
    return
```

MPLINK™ Linker

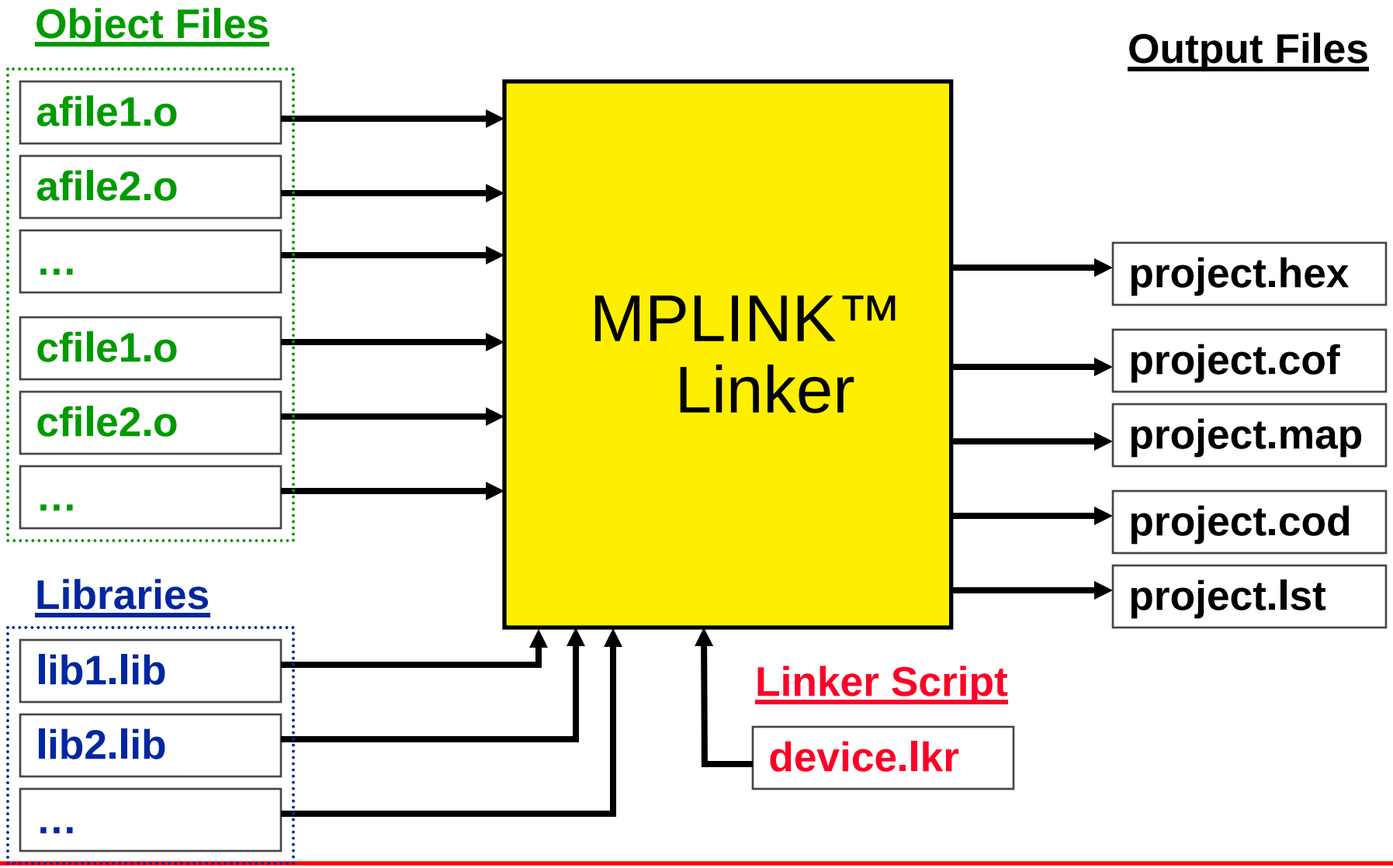
Object Files

- Object files are the relocatable code produced from source files
- If *any* section (subroutine or variable) within the object file is used, the *entire* object is placed into Program/Data Memory
- If *no* sections are used, the object is *not placed* into memory

Build Process - Compilation



Build Process - Linking



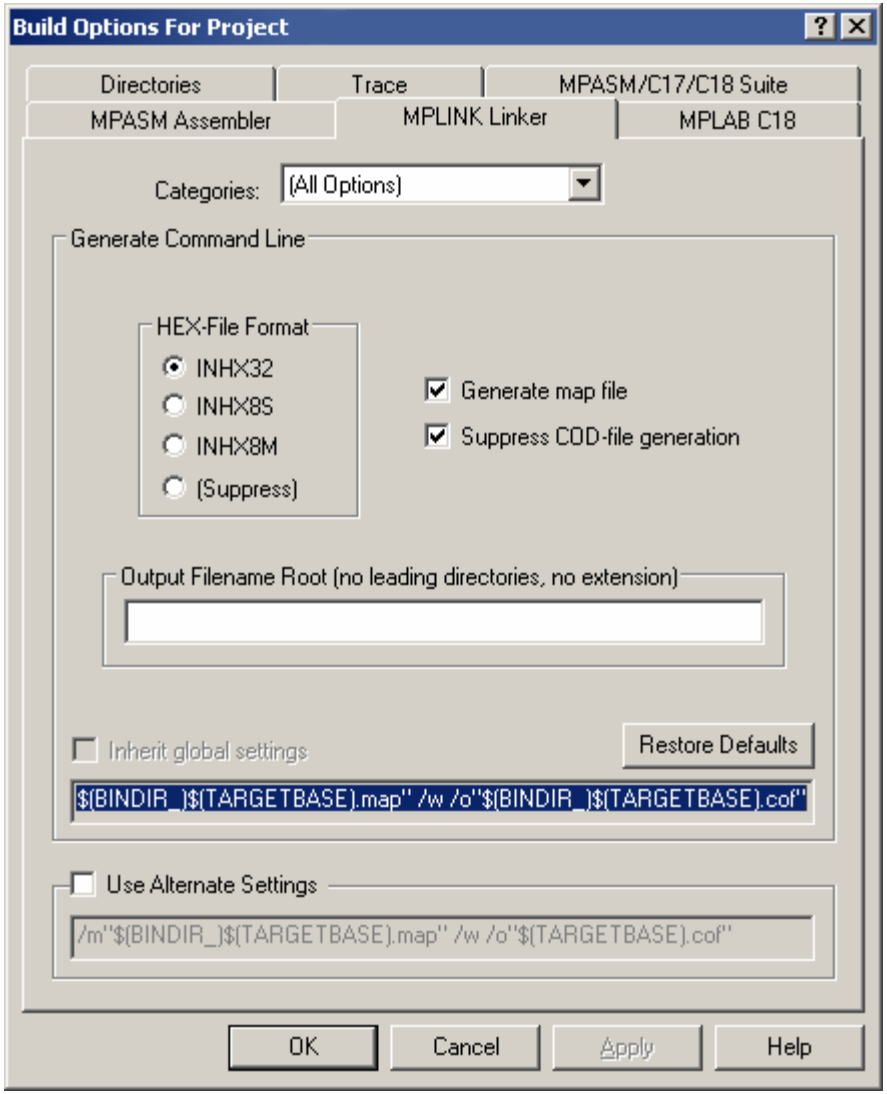
Linker Outputs

- **HEX file (.hex)**
  - Binary image with no debug information
- **COFF file (.out, .cof)**
  - Program code + debug information used by MPLAB® IDE v6.xx & later
- **Code file (.cod)**
  - Simplified version of COFF file used by MPLAB IDE v5.xx & earlier

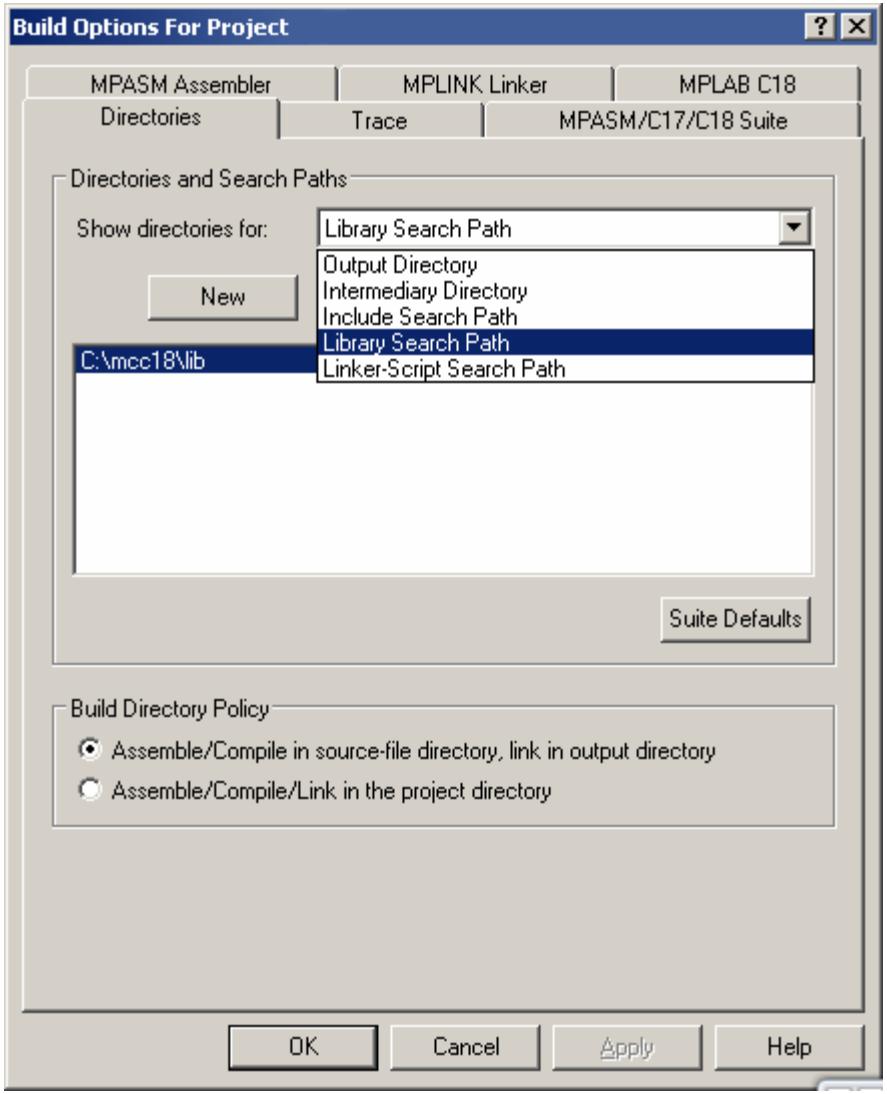
Linker Outputs (contd.)

- **Listing file (.lst)**
  - Original source code side-by-side with final assembly code
- **Map file (.map)**
  - Shows the memory layout after linking, indicates used and unused memory regions

MPLINK™ Linker Build Options



MPLINK™ Linker Build Options



MPLINK™ Linker Placement Algorithm

1. Place all absolute sections

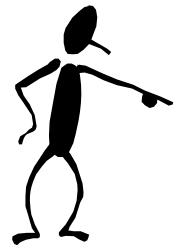
2. Place all assigned sections, using best-fit algorithm

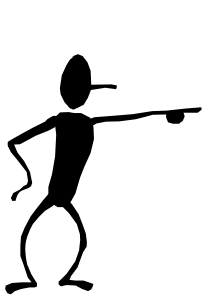
3. Place all unassigned sections, using best-fit algorithm

Linker Scripts

Linker Scripts

- **Linker scripts direct the linker in placement of code and variables**
- **They describe:**
  - Memory Regions
  - Logical Sections
  - Stack Size and Location





University of Microchip UofM MASTERs 2007

Example Linker Script

modified C18 18F4620i.LKR

Command Line Options

```
LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f4620.lib
```

Memory Region

```
CODEPAGE   NAME=vectors    START=0x0          END=0x29          PROTECTED
CODEPAGE   NAME=page       START=0x2A         END=0xFD7F
CODEPAGE   NAME=debug      START=0xFD80       END=0xFFFF        PROTECTED
CODEPAGE   NAME=idlocs     START=0x200000     END=0x200007      PROTECTED
CODEPAGE   NAME=config     START=0x300000     END=0x30000D      PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE     END=0x3FFFFF      PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000     END=0xF003FF      PROTECTED

ACCESSBANK NAME=accessram  START=0x0          END=0x7F
DATABANK   NAME=gpr0       START=0x80         END=0xFF
DATABANK   NAME=gpr1       START=0x100        END=0x1FF
DATABANK   NAME=gpr2       START=0x200        END=0x2FF
…
DATABANK   NAME=gpr14      START=0xE00        END=0xEF3
DATABANK   NAME=dbgspr     START=0xEF4        END=0xEFF         PROTECTED
DATABANK   NAME=gpr15      START=0xF00        END=0xF7F
ACCESSBANK `NAME=accesssfr  START=0xF80         END=0xFFF          PROTECTED
```

Logical Sections

```
SECTION    NAME=CONFIG     ROM=config
SECTION    NAME=DEEPROM    ROM=eedata
```

Stack

```
STACK      SIZE=0x100       RAM=gpr13
```

Command Line Options

modified C18 18F4620i.LKR

```
LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f4620.lib
```

Linker Script Directives
Command Line Options

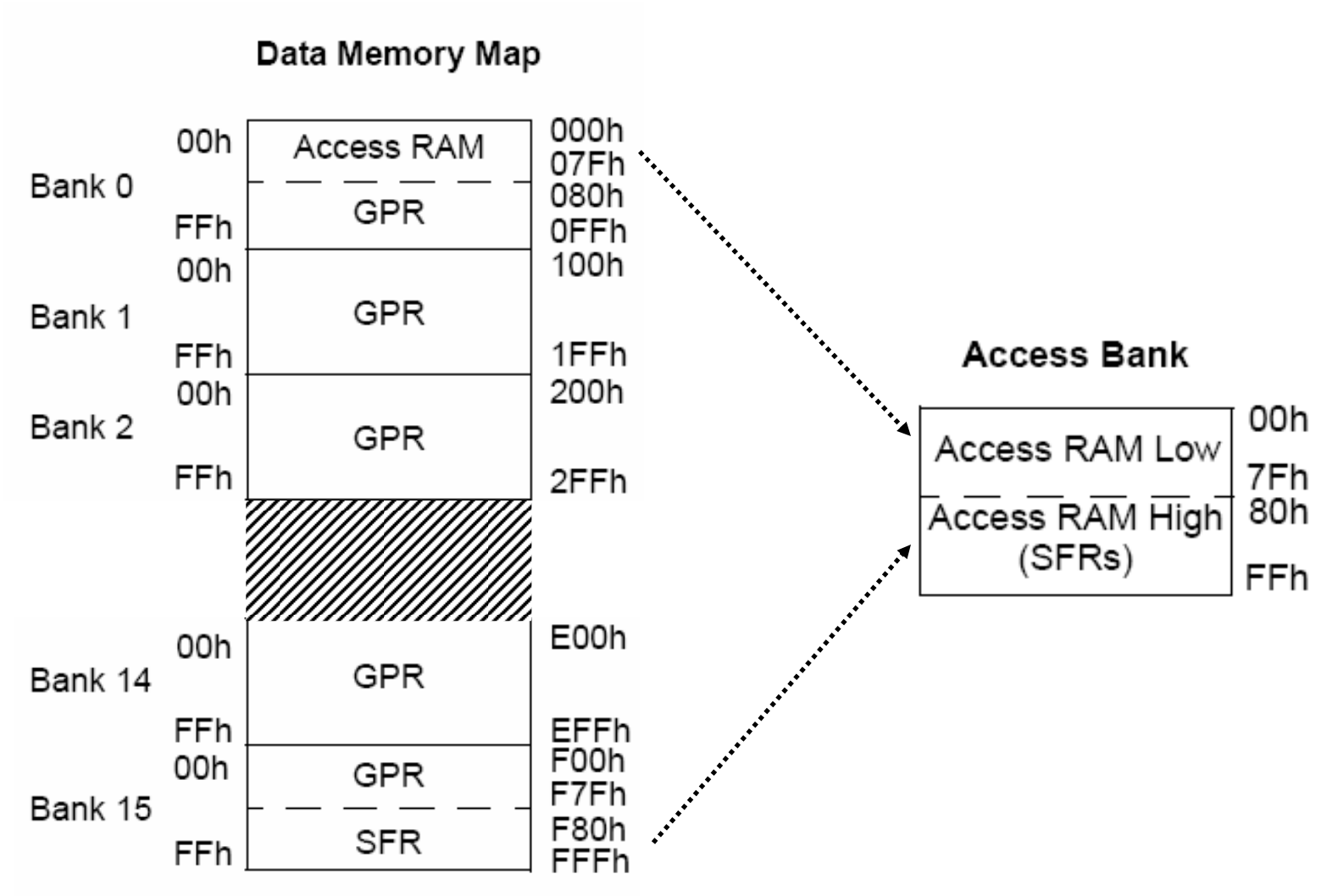
- **FILES**
  - specify source or library files to include in linkage
- **INCLUDE**
  - specify additional linker script files to use for this linkage
- **LIBPATH**
  - specify library search path
- **LKRPATH**
  - specify linker script search path

Memory Region

modified C18 18F4620i.LKR

```
CODEPAGE   NAME=vectors    START=0x0        END=0x29        PROTECTED
CODEPAGE   NAME=page       START=0x2A       END=0xFD7F
CODEPAGE   NAME=debug      START=0xFD80     END=0xFFFF      PROTECTED
CODEPAGE   NAME=idlocs     START=0x200000   END=0x200007    PROTECTED
CODEPAGE   NAME=config     START=0x300000   END=0x30000D    PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE   END=0x3FFFFF    PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000   END=0xF003FF    PROTECTED

ACCESSBANK NAME=accessram  START=0x0        END=0x7F
DATABANK   NAME=gpr0       START=0x80       END=0xFF
DATABANK   NAME=gpr1       START=0x100      END=0x1FF
DATABANK   NAME=gpr2       START=0x200      END=0x2FF
…
DATABANK   NAME=gpr14      START=0xE00      END=0xEF3
DATABANK   NAME=dbgspr     START=0xEF4      END=0xEFF       PROTECTED
DATABANK   NAME=gpr15      START=0xF00      END=0xF7F
ACCESSBANK NAME=accesssfr  START=0xF80      END=0xFFF       PROTECTED
```

PIC18F4620 Data Memory Map



Linker Script Directives
Memory Region Description

- **CODEPAGE**
  - Program Memory
- **DATABANK**
  - Banked Data Memory
- **SHAREBANK**
  - Unbanked Data Memory (non-PIC18 core)
- **ACCESSBANK**
  - Access RAM (PIC18 core)

PROTECTED Regions

- *The **PROTECTED** keyword restricts a region to assigned sections only*

```
DEEPROM     CODE
DB    “ABC”
```

```
CODE  0xF00000
DB    “ABC”
```

- **Examples**
  - Bootloaders
  - ICD2

Logical Sections

modified C18 18F4620i.LKR

```
SECTION    NAME=CONFIG      ROM=config
SECTION    NAME=DEEPROM     ROM=eedata
```

From Memory Region description

```
CODEPAGE   NAME=config      START=0x300000        END=0x30000D
…
CODEPAGE   NAME=eedata      START=0xF00000        END=0xF003FF
```

University of Microchip UofM MASTERs 2007

Linker Script Directives
Logical Section Description

- *Logical Section Syntax:*

```
SECTION NAME=<scn name> ROM=<region name>
SECTION NAME=<scn name> RAM=<region name>
```

Stack

modified C18 18F4620i.LKR

```
STACK       SIZE=0x100              RAM=gpr13
```

Linker Script Directives
Stack Description

- *Stack Definition Syntax:*

```
STACK SIZE=<stack size> {RAM=<region name>}
```

Where can I find my device's linker script?

- **MPASM™ Assembler**
  - C:\Program Files\Microchip\MPASM Suite\LKR
- **MPLAB® C18**
  - C:\MCC18\LKR

Note:

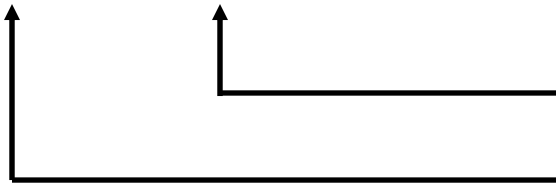
C18's linker scripts must be used in C18 projects.

They include the device's SFR definitions, standard C and peripheral libraries, and startup routines.

Linker Scripts

- **Make sure that you use the correct linker script:**

```
proc[i][_e].lkr
```



Extended Instruction Set

ICD2 Debugging

- **Example:**

	No ICD2	ICD2
Not extended	18f4620.lkr	18f4620i.lkr
Extended	18f4620_e.lkr	18f4620i_e.lkr

Map Files

Map Files

- **Map files provide information on the results of the link process**
- **They detail:**
  - Sections
  - Subroutines/Functions
  - Variables
  - Memory Use

Example Map File

demo0.map

```
                    Section Info
      Section       Type    Address   Location Size(Bytes)
    ---------  ---------  ---------  ---------  ---------
        .code       code   0x000000    program   0x000004
       .cinit    romdata   0x00002a    program   0x00000e
     .idata_i    romdata   0x000038    program   0x000007
       .idata      idata   0x000080       data   0x000007
       .udata      udata   0x000087       data   0x000004

                 Program Memory Usage
                  Start         End
                ---------   ---------
                0x000000    0x000003
                0x00002a    0x00003e
25 out of 66584 program addresses used, program memory utilization is 0%

                  Symbols - Sorted by Name
        Name    Address   Location    Storage File
    ---------  ---------  ---------  --------- ---------
  _.code_0000   0x000000    program     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
     ROMDATA   0x000083       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   idata_var0   0x000082       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   idata_var1   0x000080       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
 romdata_var0   0x000085       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
 romdata_var1   0x000083       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   udata_var0   0x000089       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   udata_var1   0x000087       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm

                  Symbols - Sorted by Address
        Name    Address   Location    Storage File
    ---------  ---------  ---------  --------- ---------
  _.code_0000   0x000000    program     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   idata_var1   0x000080       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   idata_var0   0x000082       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
     ROMDATA   0x000083       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
 romdata_var1   0x000083       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
 romdata_var0   0x000085       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   udata_var1   0x000087       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
   udata_var0   0x000089       data     static C:\MASTERs\11034 MPL\LABS\DEMO0\main.asm
```

University of Microchip UofM MASTERs 2007

Section Info

Section Info

Section	Type	Address	Location	Size(Bytes)
.code	code	0x000000	program	0x000004
.cinit	romdata	0x00002a	program	0x00000e
.idata_i	romdata	0x000038	program	0x000007
.idata	idata	0x000080	data	0x000007
.udata	udata	0x000087	data	0x000004

Program Memory Usage

```
Program Memory Usage
                    Start        End
                  ---------   ---------
                  0x000000    0x000003
                  0x00002a    0x00003e
25 out of 66584 program addresses used,
 program memory utilization is 0%
```

Symbols - Sorted by Name

Symbols - Sorted by Name

Name	Address	Location	Storage	File
_.code_0000	0x000000	program	static	main.asm
ROMDATA	0x000083	data	static	main.asm
idata_var0	0x000082	data	static	main.asm
idata_var1	0x000080	data	static	main.asm
romdata_var0	0x000085	data	static	main.asm
romdata_var1	0x000083	data	static	main.asm
udata_var0	0x000089	data	static	main.asm
udata_var1	0x000087	data	static	main.asm

Symbols - Sorted by Address

Symbols - Sorted by Address

Name	Address	Location	Storage	File
_.code_0000	0x000000	program	static	main.asm
idata_var1	0x000080	data	static	main.asm
idata_var0	0x000082	data	static	main.asm
ROMDATA	0x000083	data	static	main.asm
romdata_var1	0x000083	data	static	main.asm
romdata_var0	0x000085	data	static	main.asm
udata_var1	0x000087	data	static	main.asm
udata_var0	0x000089	data	static	main.asm

Good Practices

Using #define

#define Pinouts

- **Use #define statements to define pins used in one central location**

```
; LCD Pins
#define LCDEN      PORTA, 1
#define LCDRW      PORTA, 2
#define LCDREGSEL  PORTA, 3
```

#define Overview

- **A #define statement simply is a text substitution when used**

```
#define MYLABEL  PORTD, 4


bcf MYLABEL   ;
```

turns into:

```
bcf PORTD, 4   ;
```

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Using #define

- **Use:**

```
#define LCDEN    PORTA, 4
bcf   LCDEN      ; Set EN low
```

- **Use for C18:**

```
#define LCDEN    PORTAbits.RA4
LCDEN = 0;       // Set EN low
```

Benefits To #define

- **Changing a #define changes all uses of that definition**

  (easy update/migration of file)

- **More 'readable' code**

  (names not numbers)

Function Size

Split Up Large Functions

- **Split large functions into smaller functions which are reusable**
  - Allows for easier reuse by other parts of your project
  - Improves reuse in future projects

- **Downside**
  - Many nested calls may lead to stack overflow

UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007

Paging and Banking Tips

Banking

- **Use at the start of a call.**
  Assume prior bank is unknown.

```
LoadVariable:
    BANKSEL  MyVariable ;Before use
    movlw    0xFF
    movwf    MyVariable
    return
```

PAGESEL

- **Use PAGESEL before a call to an external label.**
  Assume page is unknown.

```
    .
    .
PAGESEL LoadVariable
call    LoadVariable
    .
    .
```

PAGESEL

- Each SECTION must fit in a page, and so calls within that SECTION are safe
- Calls to external labels are key points for potential error

Improper Use of BANKSEL & PAGESEL

- **Improper bank and page selection:**
  - Won't throw a build or link error
  - May cause erratic behavior
  - Are very frustrating to debug!

Banksel & Pagesel Summary

- **Use `BANKSEL` liberally to ensure proper banking at all times**

- **Projects larger than one page of Program Memory must:**
  - Use **`PAGESEL`** to ensure proper behavior

Common Linker Errors

University of Microchip UofM MASTERs 2007

Oversize Section

- Symptom:
  “section ‘.udata_test.o’ can not fit the section.

  Section ‘.udata_test.o’ length = 0x0000012c”

- Remedy:
  Use the error map file to see how memory was allocated when a failure occurred.

Oversize Section

Questions:

- **Does my data/code size exceed the parts resources?**
  - Enable optimizations
  - Move data between shared/GPR/access memories
- **Are my sections bigger than available memory regions?**
  - Break into smaller chunks

Undefined Symbol

- **Symptom:**

  “Error – could not find definition of symbol ‘FSR2L’ in file ‘./test.o’.”

Undefined Symbol

- **Questions:**
  - For MPLAB® C18, use compiler linker scripts, not the MPASM™ assembler linker scripts
  - Is your library search path correct?
  - Are all source files and libraries included?
  - Are all labels (symbols) properly defined? *(does other file use* **`global`***)*
  - Verify file containing symbol exists on linker command line *(make sure it is included in the link)*

Lab 2

Creating a Multi-file Project

Lab 2
Creating a Multi-file Project

- **See handout for instructions.**

Libraries

Using MPLIB™ Librarian

What is a Library?

- A collection of .o files placed into a .lib, library file
- Allows grouping of logical reusable subroutines

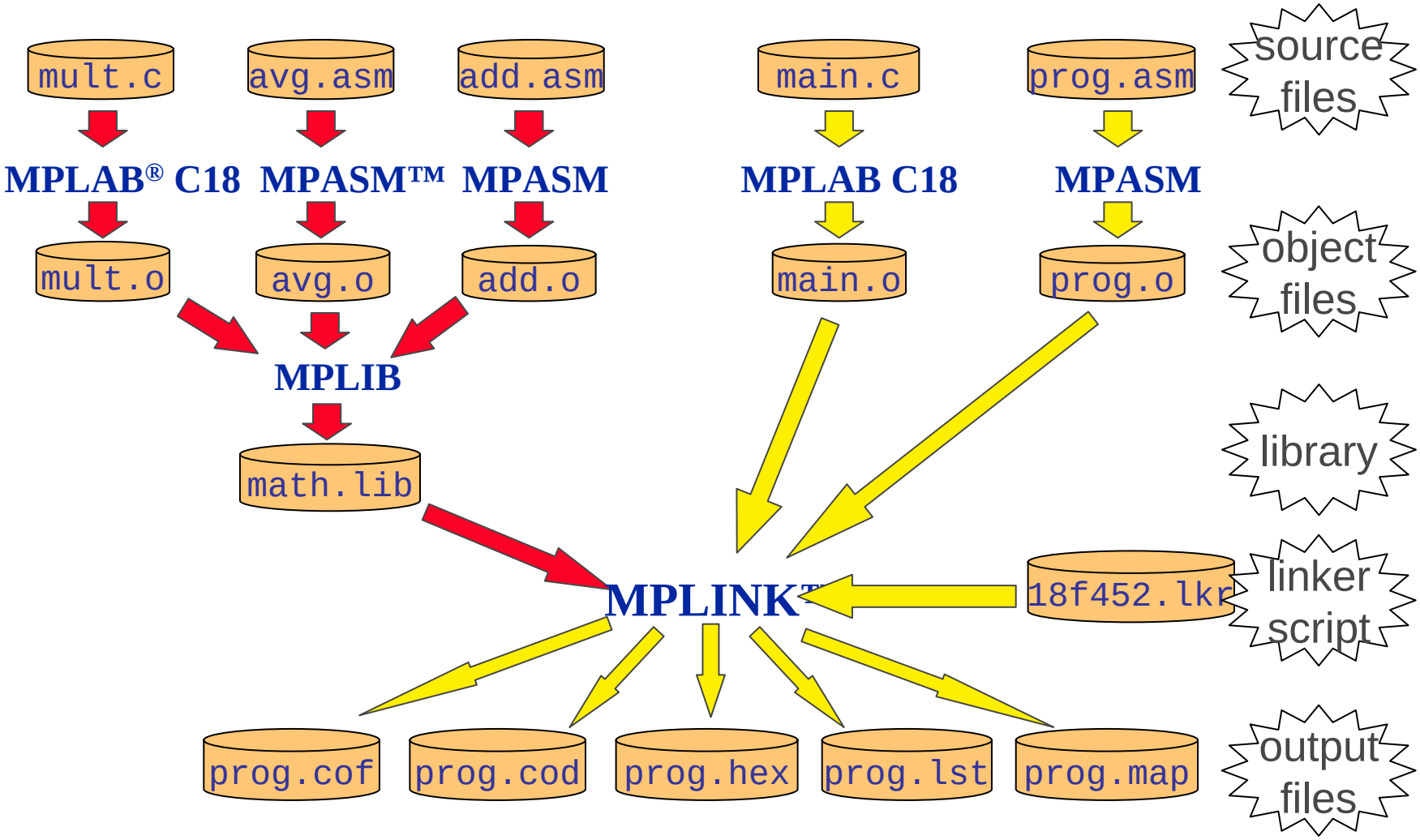
Other Benefits to a .lib

- **Must only link a single file**
  math.lib VS mult.o, div.o, add.o ...

- **Only .o files called within the project are used during linking and build.**
  (minimum space used)

- **Example:**
  Using multiply call from within math.lib does not use division calls from div.o

MPLINK™ Linker Build Process



Making Library Files

- **Two Options:**
  1. MPLAB® IDE Interface
  2. Command Line

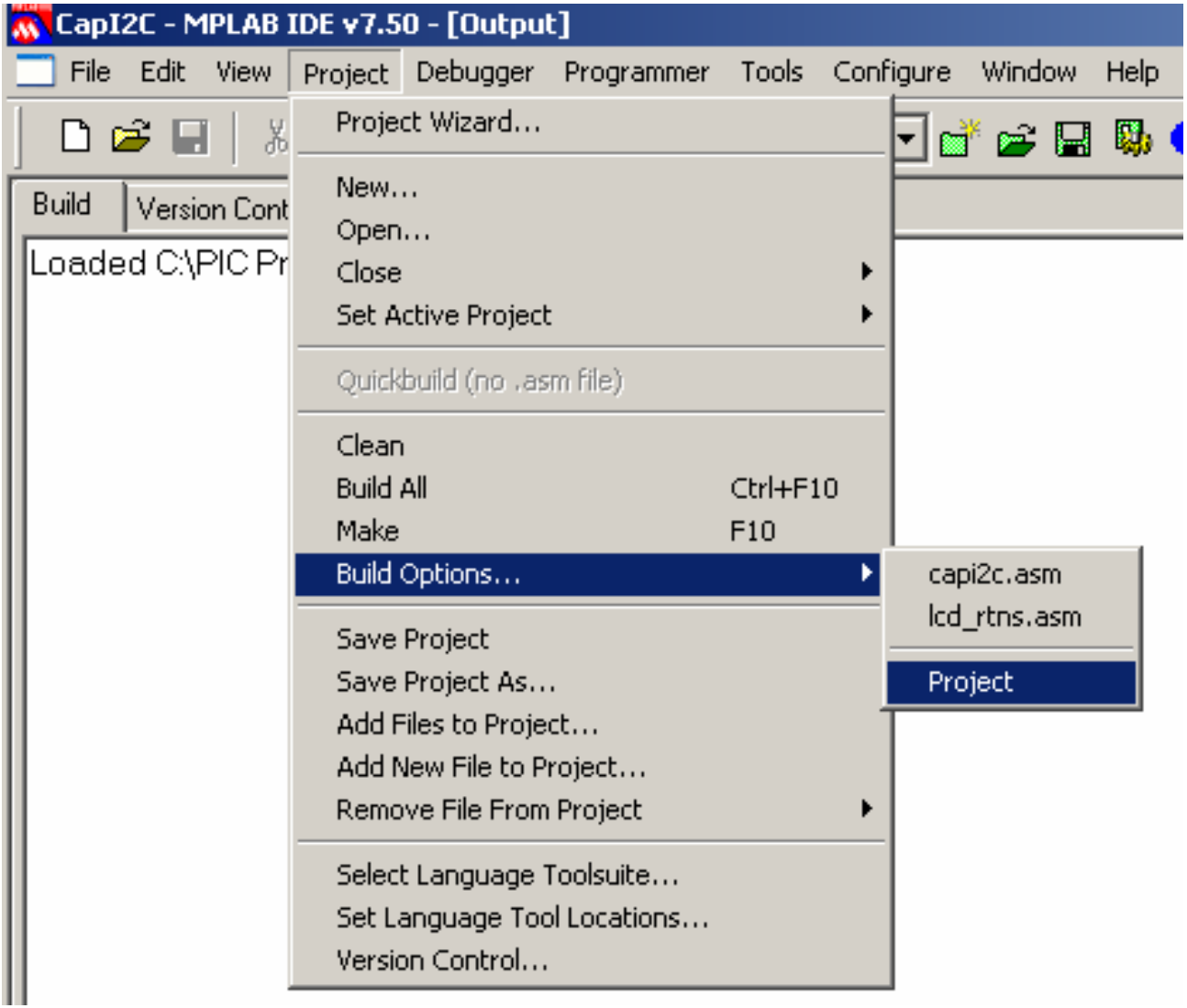
MPLAB® IDE Interface

MPLAB® IDE Interface

- **Easy GUI Interface**
- **Output .lib instead of .hex**
- **Change Project Build Options:**

  Project>Build Options>Project…
  (Opens new window)

MPLAB® IDE Interface



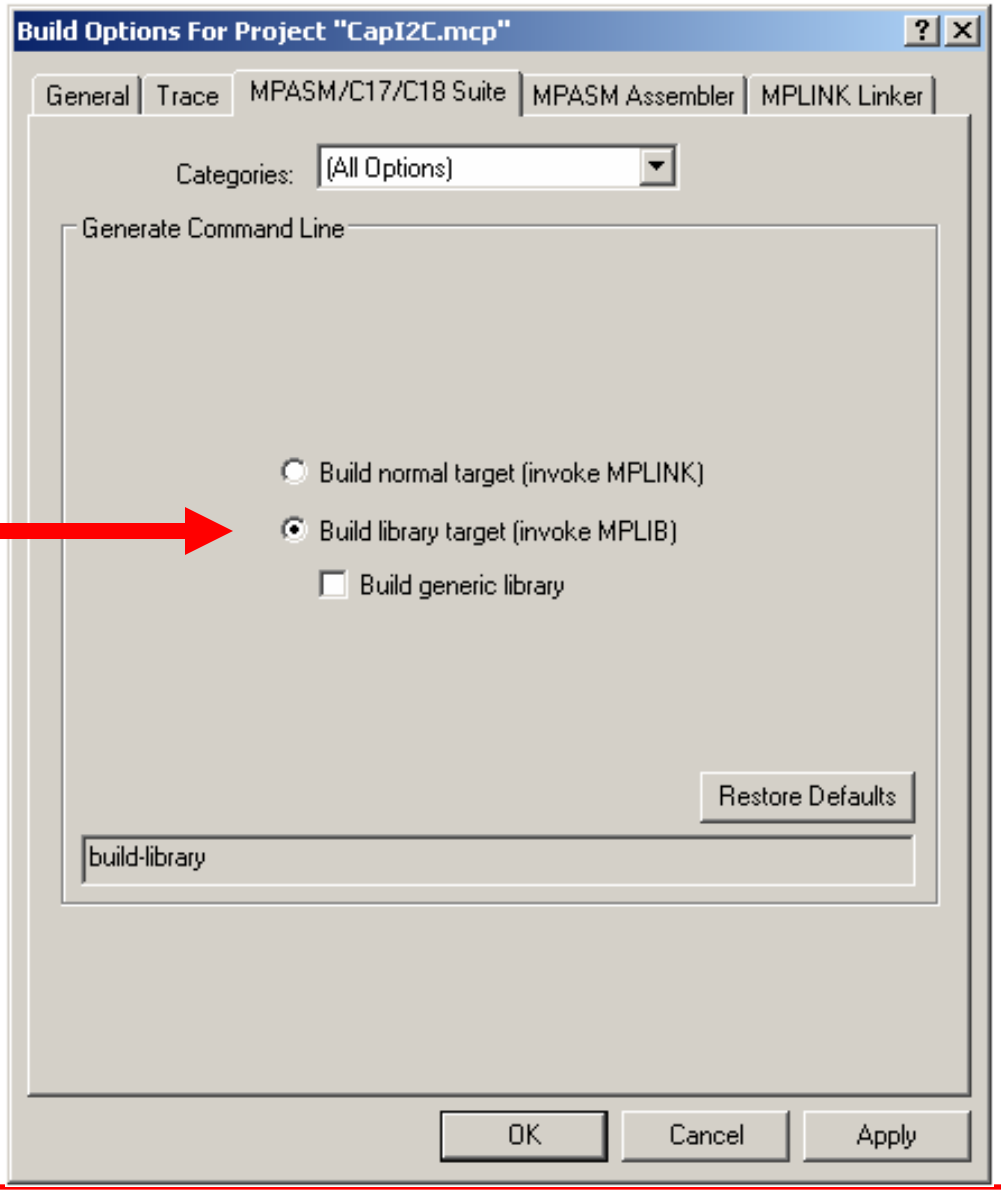
MPLAB® IDE Interface

- **With Project Window Open…**

- **Perform Two Steps:**
  1. Select MPASM/C17/C18 Suite Tab
  2. Check Radio Button
     "Build Library Target (invoke MPLIB)"

MPLAB® IDE Interface

Invoke MPLIB™ Librarian



Command Line Interface

Command Line Interface

- **Use:**

```
mplib [/q] /{ctdrx} LIBRARY [MEMBER...]
```

Option	Description
/c	Create Library
/d	Delete Member
/q	Quiet Mode
/r	Add/Replace Member
/t	List Members
/x	Extract Member

MPLAB® IDE Interface only uses this option.

Command Line Interface

- **Why use the command line?**
  - Manual and exact control for managing a library file's contents
  - Editing a library file built from a project
  - Automated Batch File Processing

Interface Summary Suggestion

- **Use MPLAB® IDE when creating a new library file**
  - Organize each desired .o as a file
  - Keep each .o as small as possible (Entire .o is linked for just 1 call)
  - Setup the project to output a .lib file
  - Build the project

Creating .lib Files

Creating Library Files

- **Elements of a good library:**
  - Subroutines/Functions which will be used often, but only written once
  - Clearly defined and documented interface for calls into the library

Creating Library Files

- **Exported Names**
  - Only 1 Instance of an exported name allowed via **`global`** directive
  - Applies across all .o files
  - If another .o file attempts to export the same name, an error occurs

University of Microchip UofM MASTERs 2007

Creating Library Files

8 and 16-bit addition: Sum = A + B

"add8.asm"

```
global A
global B
global Sum
```

"add16.asm"

```
global A
global B
global Sum
```

MPLIB™ Librarian Error!

University of Microchip UofM MASTERs 2007

Creating Library Files

8 and 16-bit addition: Sum = A + B

"add8.asm"

```
global A8
global B8
global Sum8
```

"add16.asm"

```
global A16
global B16
global Sum16
```

MPLIB™ Librarian Success

Creating Library Files

- **Suggested Naming Convention**
- **Use module descriptor at start of Label, Function, or Variable**

```
LCD_SendData
LCD_Clear
LCD_Goto
```

Creating Library Files

- When creating a library file, also create associated header file
- Useful to define all exported calls, variables, #defines located within the library
- Helps document the library file

University of Microchip UofM MASTERs 2007

Include File

- **Import Calls and Variables**

- **To use math.lib file:**

"math.inc"

```
extern A8          ; Import
extern B8          ; variables
extern Sum8        ; and call
extern Add8        ; from
                   ; add8.o
```

Using a Library File

- `#include` your .inc/.h file in your assembly or C code
- Link in your .lib file
- Use your library calls

Lab 3

Creating and Using Libraries

Lab 3
Creating and Using Libraries

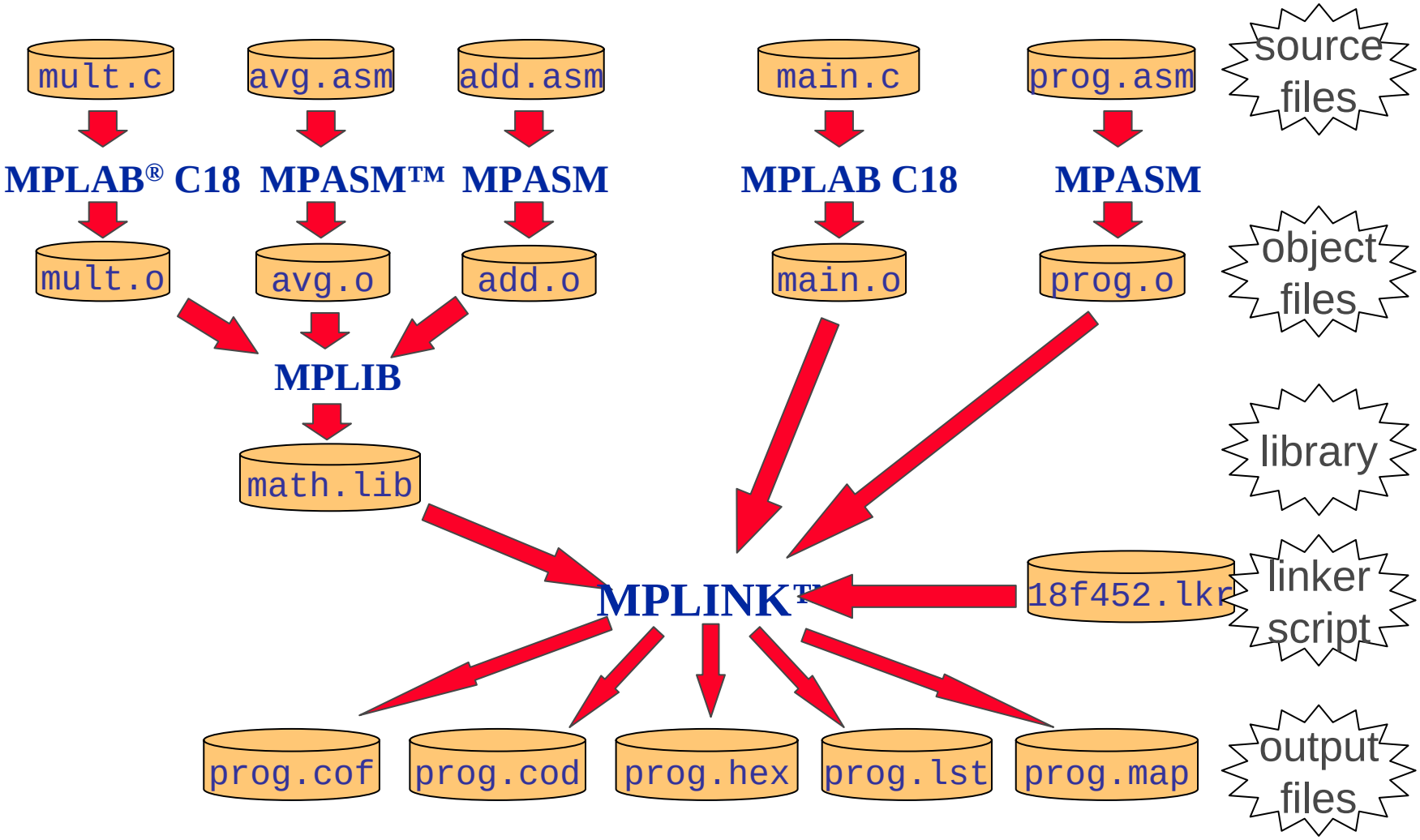
- **See handout for instructions.**

Summary

Summary

- **Relocatable code is:**
  - Modular
  - Flexible
  - Easy to use/reuse
  - Well suited for large projects

MPLINK™ Linker Build Process



MPLINK™ Linker Summary

- MPLINK Linker converts object files to an executable .hex file
- Source file inputs: .lib, .o, .lkr
- Linker script (.lkr) directs linker placement of variables and code from object (.o) and library (.lib) files

MPLIB™ Librarian Summary

- Creates a collection of object files from MPASM™ assembler or C18 .o files
- Use Build option to create .lib file
- Library files may be used in full or partially as needed
- Document your library!

Dev Tools used in this class

- **DV164006 – PICDEM™ 2 Plus Kit**
  - DM163022 - PICDEM 2 Plus board
  - DV164007 – MPLAB® ICD 2

Additional References

- DS33014 - MPASM™ Assembler/MPLINK™ Linker/MPLIB™ Librarian User's Guide
- MPLINK Linker and MPLIB Librarian documentation is available from within MPLAB® IDE

Other MASTERs Classes

- 11001 GS1 - Getting Started w/ Microchip Tools
- 11002 GS2 - Getting Started w/ Mid-Range Microcontroller Family
- 11003 GS3 Getting Started w/ PIC18

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.