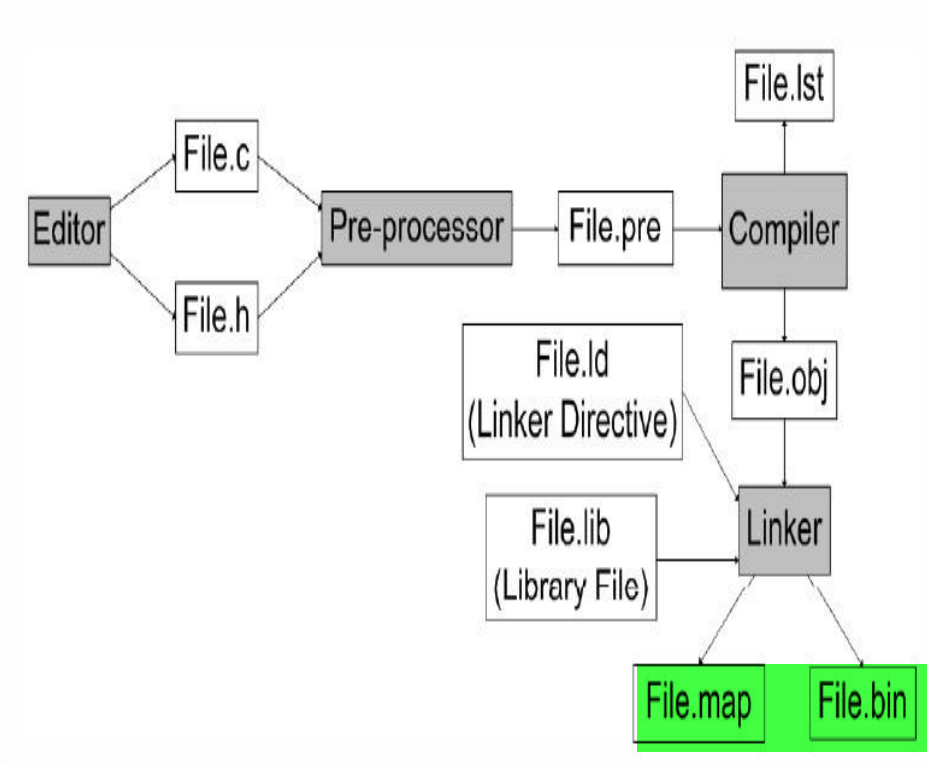
Building Process

Outline

- The Course consists of the following topics:
  - Building Overview
  - Compilation stage
  - Object File
  - Linker File
  - Linking Stage
  - ELF , BIN, MAP Files
  - Building Scenario
  - Microcontroller Memory Segments
  - Memory Allocation Scenario

Building Overview



Compilation Stage

This stage divided to three steps :

- Front End (source code parsing)
- Middle End (optimization)
- Back End(Code generation)

Compilation Stage

Front End (source code parsing):

1- Preprocessing :

- The input to this phase is the .c File and .h Files
- The preprocess process the preprocessor keywords like #define,
- #ifdef, #include, etc. and generate a new .pre file or .i file after the
- It is a text replacement process.
- The output of this phase is a C Code without any preprocessor keyword.

Preprocessing

My_defs.h

```
#define JAN
#define FEB
#define MAR
#define PI
1
2
3
3.1416
double my_global;
```

My_prog.c

```
#include
“my_defs.h”
Int main()
{double
angle=2*PI;

printf(“%s”,month[F

EB]);

}
```

My_prog.i

```
#define JAN
#define FEB
#define MAR
#define PI
1
2
3
3.1416
double my_global;
Int main()
{
double
angle=2*3.1416;
printf(“%s”,month[2]);
}
```

Compilation Stage

2- Tokenising:

Identify tokens like: keywords, operators, identifier and comments

3- Syntax Analysis:

The input to this phase C Files without any „#" statement.

The compiler parse the code, and check the syntax correctness of the file.

4- Intermediate Representation:

Generates a file.i

Compilation Stage

Middle End (optimization):

1- Semantic Analysis:

The compiler tries to understand the logic of the code if there is a problem it generates a warning: “conflicting”, “missing”

The output of this step is

a-“Program Symbol Table” which includes:

- Global Symbols (every non static global variable or function)
- Private Symbols(every static global variable or function)

b-”Debug Info” : which is the connection between the debugger and the machine code

Compilation Stage

2- Optimization :

Convert the C Code into faster in execution and smaller size code with the same functionality.

Compilation Stage

Back End(Code generation)

- This stage contains the “Assembler” that he Assembler coverts the assembly code to the corresponding machine language code.
- The output of this phase is object file .o file or .obj file.

Object File

The object file is the output of the Compilation stage it contains many sections :

1- “.bss” section : which contains the un initialized global and un initialize static variables

2-”.data” section: which contains the initialize global and initialized static variables

3-”.text” section”: which contains the code that executes

4-”.rodata” section: which contains the constant variables.

5-”symbol table” section: which have the global symbols and the private symbols

6-”exports” section: which contains each global symbol

7-”imports” section: which contains the needed symbols to be used in this file from other files

8-”debug info” section : which is used by the Debugger

Linker File

- The input file to the linker and it contains the memory mapping information of the segments in the microcontroller memory.
- So using the linker file you could place any segment in any place in the memory.
- #pragma is used to change the default segment for specific variable or code

Linker

- The input to this phase multiple .obj Files.
- The Linker merges different object files and library files.
- Linker is used to be sure that every .obj files or .o files have all the external data and functions that is defined.
- And allocate target memory (RAM,ROM and Stack) to give addresses to the variables and function according to .ld file(linker file) and generate .map file and .elf File.
- The output of this phase is the .elf file and the .map file.

What Does a Linker Do?

- Merges object files
- Resolves external references
- Relocates symbols

Executable and Linkable Format (ELF)

- Standard binary format for object files
- Derives from AT&T System V Unix
  - Later adopted by BSD Unix variants and Linux
- One unified format for
  - Relocatable object files (.o),
  - Executable object files
  - Shared object files (.so)
  - Generic name: ELF binaries
- Better support for shared libraries than old a.out formats.
- The .elf is target-independent format so it should be converted to a native format like

.bin or .hex which burn in target

MAP FILE

- The map file is an additional output file that contains information about the location of sections and symbols. You can customize the type of information that should be included in the map file.

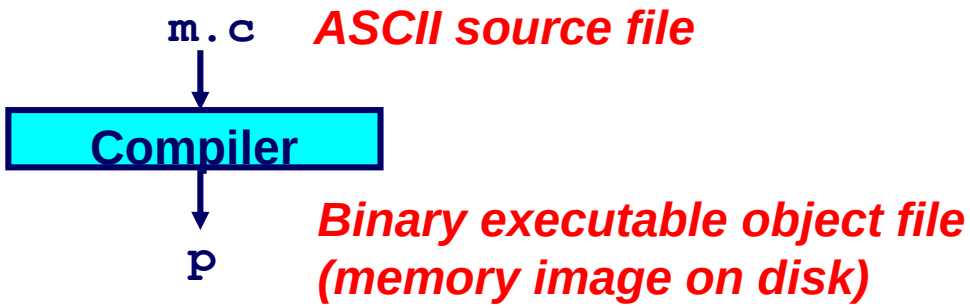
MAP FILE

```
MEMORY MAP OF MODULE:  Test (?C_STARTUP)
```

START	STOP	LENGTH	TYPE	RTYP	ALIGN	TGR	GRP	COMB	CLASS	SECTION NAME
000000H	000003H	000004H	---	---	---	---	---	---	* INTVECTOR TABLE *	
000004H	00000DH	00000AH	XDATA	REL	WORD	---	---	GLOB	---	?C_INITSEC
00000EH	00001DH	000010H	CONST	ABS	WORD	---	---	PRIV	---	?C_CLRMEMSEC
00001EH	00005EH	000041H	DATA	REL	BYTE	---	---	PUBL	FCONST	?FC?TEST
000060H	00009FH	000040H	DATA	REL	WORD	---	---	PUBL	FCONST	?FC??PRNFMT
0000A0H	0001CDH	00012EH	CODE	REL	WORD	---	---	PRIV	ICODE	?C_STARTUP_CODE
0001CEH	00065BH	00048EH	CODE	REL	WORD	---	2	PRIV	NCODE	?PR?SCANF
00065CH	00098FH	000334H	CODE	REL	WORD	---	2	PUBL	NCODE	?C_LIB_CODE
000990H	000A25H	000096H	CODE	REL	WORD	---	2	PUBL	NCODE	?PR?TEST
000A26H	000A57H	000032H	CODE	REL	WORD	---	2	PRIV	NCODE	?PR?PUTCHAR
000A58H	000A85H	00002EH	CODE	REL	WORD	---	2	PUBL	NCODE	?PR?GETCHAR
000A86H	000A9FH	00001AH	CODE	REL	WORD	---	2	PUBL	NCODE	?PR?ISSPACE
000AA0H	000AABH	00000CH	CODE	REL	WORD	---	2	PUBL	NCODE	?PR?GETKEY
000AACH	000AB3H	000008H	CODE	REL	WORD	---	2	PUBL	NCODE	?PR?UNGET
008000H	008FFFH	001000H	DATA	REL	WORD	---	1	PUBL	NDATA	?C_USERSTACK
009000H	009003H	000004H	DATA	REL	WORD	---	1	PUBL	NDATA0	?ND0?TEST
009004H	009004H	000001H	DATA	REL	BYTE	---	1	PUBL	NDATA0	?ND0?GETCHAR
009006H	009055H	000050H	DATA	REL	WORD	---	---	PUBL	FDATA0	?FD0?TEST
00FA00H	00FBFFH	000200H	---	---	---	---	---	---	* SYSTEM STACK *	
00FC00H	00FC1FH	000020H	DATA	---	BYTE	---	---	---	*REG*	?C_MAINREGISTERS

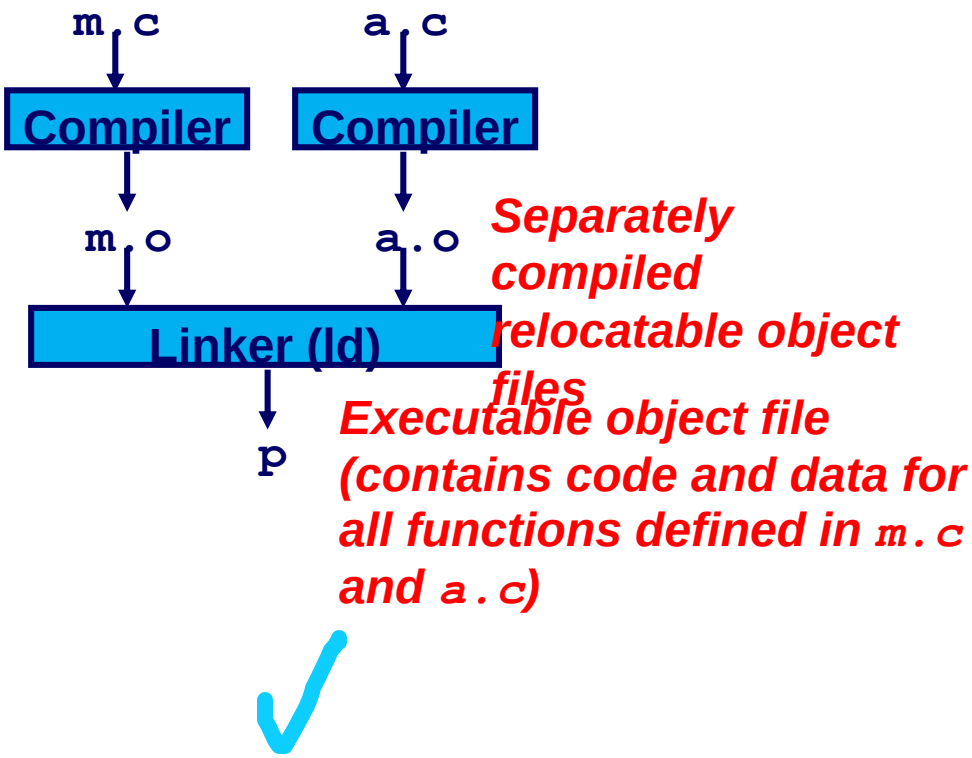
Building Scenario

- A Simplistic Program Translation Scheme



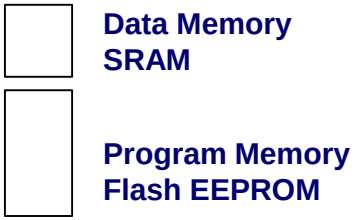
Building Scenario

- A Better Scheme Using a Linker



Microcontrollers Memory Segments

- The computer program memory is organized into the following:
  - Data Segment (.data + .bss + Heap + Stack)
  - Constant Segment (.rodata)
  - Code segment (.text)



Microcontrollers Memory Segments

- .data Segment
  - Holds initialized variables
- .bss Segment
  - Holds uninitialized variables
- .stack Segment
  - Holds the stack
- .rodata Segment
  - Holds the constant data
- .text Segment
  - Holds the program code

Example C Program

m.c

```
int e=7;

int main() {
  int r = a();
  exit(0);
}
```

a.c

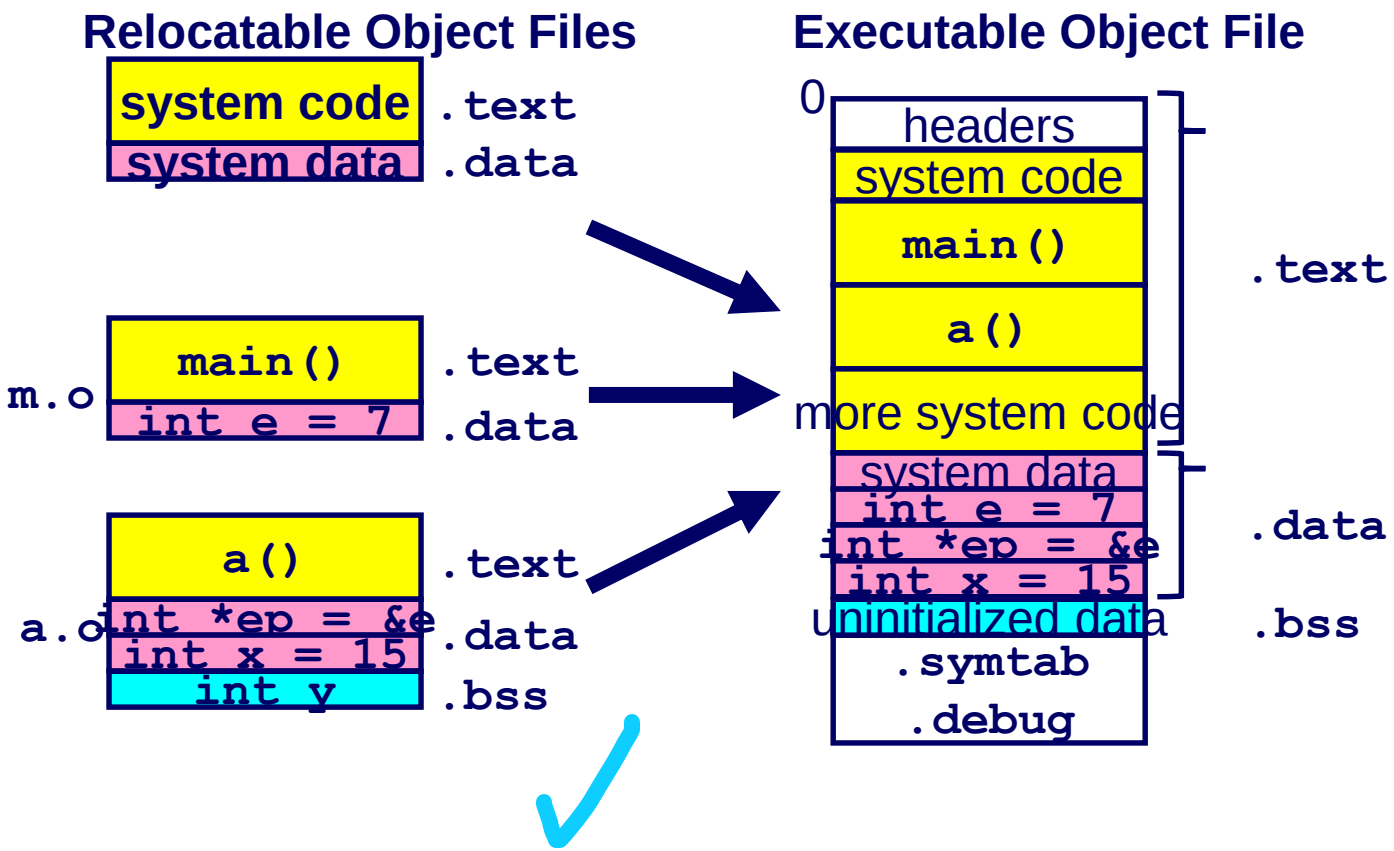
```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
  return *ep+x+y;
}
```

Merging Object Files

- Merging Re-locatable Object Files into an Executable Object File



Relocation

- Relocating Symbols and Resolving External References
  - Symbols are lexical entities that name functions and variables.
  - Each symbol has a value (typically a memory address).
  - Code consists of symbol definitions and references.
  - References can be either local or external.

m.c

```
int e=7;

int main() {
  int r = a();
  exit(0);
}
```

a.c

```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
  return *ep+x+y;
}
```

Def of local symbol e

Ref to external symbol exit (defined in libc.so)

Ref to external symbol a

Def of local symbol ep

Def of local symbol a

Refs of local symbols ep,x,y

Ref to external symbol e

Defs of local symbols x and y

m.o Relocation Info

m.c

```
int e=7;

int main() {
  int r = a();
  exit(0);
}
```

```
Disassembly of section .text:

00000000 <main>: 00000000 <main>:
   0:   55                  pushl  %ebp
   1:   89 e5               movl   %esp,%ebp
   3:   e8 fc ff ff ff      call   4 <main+0x4>
                            4: R_386_PC32   a
   8:   6a 00               pushl  $0x0
   a:   e8 fc ff ff ff      call   b <main+0xb>
                            b: R_386_PC32   exit
   f:   90                  nop
```

```
Disassembly of section .data:

00000000 <e>:
   0:   07 00 00 00
```

a.o Relocation Info (.text)

a.c

```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
   return *ep+x+y;
}
```

```
Disassembly of section .text:

00000000 <a>:
   0:   55                  pushl  %ebp
   1:   8b 15 00 00 00      movl   0x0,%edx
   6:   00
                            3: R_386_32     ep
   7:   a1 00 00 00 00      movl   0x0,%eax
                            8: R_386_32     x
   c:   89 e5               movl   %esp,%ebp
   e:   03 02               addl   (%edx),%eax
  10:   89 ec               movl   %ebp,%esp
  12:   03 05 00 00 00      addl   0x0,%eax
  17:   00
                            14: R_386_32    y
  18:   5d                  popl   %ebp
  19:   c3                  ret
```

a.o Relocation Info (.data)

a.c

```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
   return *ep+x+y;
}
```

```
Disassembly of section .data:

00000000 <ep>:
   0:   00 00 00 00
                        0: R_386_32
e
 00000004 <x>:
   4:   0f 00 00 00
```

Executable (.text)

- Executable After Relocation and External Reference Resolution (.text)

```
08048530 <main>:
 8048530:       55                      pushl  %ebp
 8048531:       89 e5                   movl   %esp,%ebp
 8048533:       e8 08 00 00 00          call   8048540 <a>
 8048538:       6a 00                   pushl  $0x0
 804853a:       e8 35 ff ff ff          call   8048474 <_init+0x94>
 804853f:       90                      nop

08048540 <a>:
 8048540:       55                      pushl  %ebp
 8048541:       8b 15 1c a0 04          movl   0x804a01c,%edx
 8048546:       08
 8048547:       a1 20 a0 04 08          movl   0x804a020,%eax
 804854c:       89 e5                   movl   %esp,%ebp
 804854e:       03 02                   addl   (%edx),%eax
 8048550:       89 ec                   movl   %ebp,%esp
 8048552:       03 05 d0 a3 04          addl   0x804a3d0,%eax
 8048557:       08
 8048558:       5d                      popl   %ebp
 8048559:       c3                      ret
```

Executable (.data)

- Executable After Relocation and External Reference Resolution(.data)

m.c

```
int e=7;

int main() {
  int r = a();
  exit(0);
}
```

a.c

```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
  return *ep+x+y;
}
```

```
Disassembly of section .data:

0804a018 <e>:
 804a018:       07 00 00 00

0804a01c <ep>:
 804a01c:       18 a0 04 08

0804a020 <x>:
 804a020:       0f 00 00 00
```