

Where are We Going?

Processor and memory architecture

Peripherals: GPIO, timers, UART

Assembly language and machine code

From C to assembly language

Function calls and *stack frames*

Serial communication and strings

Modules and libraries: Building and linking

Memory management: Memory map & heap

gpio
timer
uart
strings
printf
malloc
keyboard
fb
gl
console
shell



Good Modules

Decompose a system into smaller parts (modules)

- **Interface: what the module does**
- **Implementation: how it does it**

A good interface

- **An easy-to-understand and easy-to-use abstraction**
- **Can be implemented in different ways**

Designing good interfaces: THE 'art' of software engineering

Modules tested independently with unit tests

"The Build"

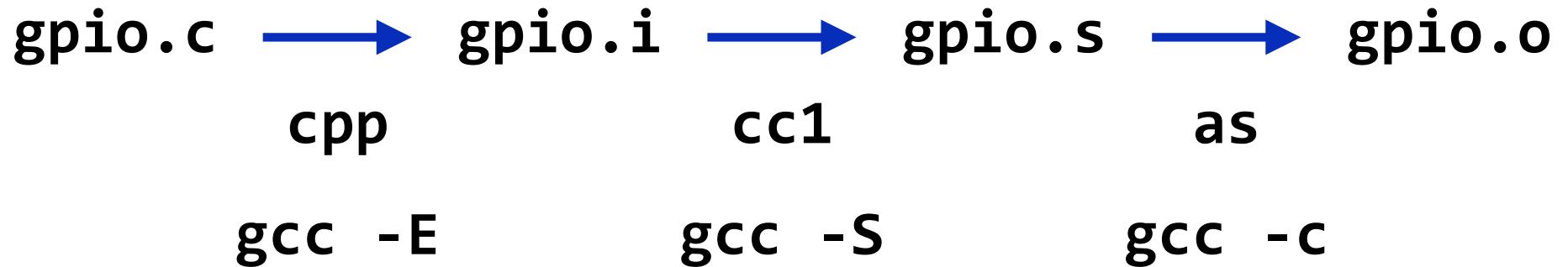


NASA Command Center during SpaceX Mission



cs107e Command Center: Makefile

gcc is all powerful



gcc --save-temp

Object Files (.o = ELF) and Symbols

```
// nm - display names (symbols)
$ arm-none-eabi-nm blink.o
    U gpio_init
    U gpio_set_function
    U gpio_write
    U timer_delay
    U timer_init
00000000 T main
```

```
// T - text symbol (function)
// U - undefined symbol
```

```
$ arm-none-eabi-nm -n gpio.o
00000000 T gpio_init
00000004 T gpio_set_function
00000008 T gpio_get_function
00000010 T gpio_set_input
00000014 T gpio_set_output
00000018 T gpio_write
0000001c T gpio_read
```

```
$ vi gpio.o.list
```

Symbols

Single global name space

- need `gpio_` prefix to distinguish names
- e.g. `gpio_init` versus `timer_init`

Defined vs. undefined symbols

Definitions: global vs local (static)

- by default symbols are local in .s files
- by default symbols are global in .c files

Local variables in functions are not symbols

Linking

**Combining Multiple Modules (.o)
into a
Single Executable (.elf)**

`blink.c` → `blink.o`

`gpio.c` → `gpio.o`

`timer.c` → `timer.o`

`cstart.c` → `cstart.o`

`start.s` → `start.o`

Linking

`blink.elf`

`ld (gcc)`

```
$ arm-none-eabi-nm -n blink.elf
00008000 T _start // start must be here!!
0000800c t hang
00008010 T main
00008060 T timer_init
00008064 T timer_get_ticks
0000806c T timer_delay_us
00008078 T timer_delay_ms
00008094 T timer_delay
000080b4 T gpio_init
000080b8 T gpio_set_function
000080bc T gpio_get_function
000080c4 T gpio_set_input
000080c8 T gpio_set_output
000080cc T gpio_write
000080d0 T gpio_read
000080d8 T __cstart
00008130 T __bss_end__
00008130 T __bss_start__
```

```
// size reports the size of the text
$ arm-none-eabi-size blink.elf
    text  data   bss  dec   hex filename
      304    0     0  304    130  blink.elf
$ arm-none-eabi-size *.o
    text  data   bss  dec   hex filename
      80    0     0   80    50  blink.o
      88    0     0   88    58  cstart.o
      36    0     0   36    24  gpio.o
      16    0     0   16    10  start.o
      84    0     0   84    54  timer.o
```

```
// The size of blink.elf is equal to
// sum of the sizes of the .o's
```

Symbol Resolution

Set of defined symbols D

Set of undefined symbols U

Moving left to right, for each .o file:

- $D' = D \text{ union } D(\text{file})$
- $U' = U \text{ difference } D(\text{file})$

Note: Adding a symbol from a file, causes all the symbols in that file to be added

Common Errors

- 1. "undefined reference"**
- 2. "multiple definitions"**

```
// _start must be declared global,  
// by default symbols in .s are local.
```

```
// Identify this section as the one  
// to go first in binary image  
.section ".text.start"
```

```
// initialize sp and fp  
.globl _start  
_start:  
    mov sp, #0x8000000  
    mov fp, #0  
    bl _cstart  
hang: b hang
```

**_start must be located
at #0x8000!!**

Magic constant that's part of Raspberry Pi boot sequence.

SECTIONS

```
{  
    .text 0x8000:{*(.text.start) *(.text*) }  
    .rodata: { *(.rodata*) }  
    .data:   { *(.data*) }  
    bss_start = .;  
    .bss : { *(.bss*) *(COMMON) }  
    bss_end = ALIGN(8);  
}
```

Libraries

**class library: blink-cs107e
your library: blink-libpi**

Libraries

The linker scans the library for any .o files that contain definitions of undefined symbols. If a file in the library contains an undefined symbol, the whole file and all its functions are linked in.

Adding the .o file from the library may result in more undefined symbols; the linker searches for the definition of these symbols in the library and includes the relevant files; this search is repeated until no more definitions of undefined symbols are found.

// An library is an unix archive file

// An archive is a set of .o files

```
% arm-none-eabi-ar cry libpi.a \
  gpio.o \
  timer.o \
  cstart.o \
  start.o
```

```
% arm-none-eabi-nm libpi.a
```

data/

```
// uninitialized global and static  
int i;  
static int j;
```

```
// initialized global and static  
int k = 1;  
int l = 0;  
static int m = 2;
```

```
// initialized global and static const  
const int n = 3;  
static const int o = 4;
```

```
% arm-none-eabi-nm -S tricky.o
00000004 00000004 C i
00000000 00000004 b j
00000000 00000004 D k
00000004 00000004 B l
00000004 00000004 d m
00000000 00000004 R n
00000004 00000004 r o
00000000 000000d8 T tricky
```

```
# The global uninitialized variable i
# is in common (C).
```

```
# If you compile with -Og, some variables
# are optimized out -- which?
```

Guide to Symbols

T/t - text

D/d - read-write data

R/r - read-only data

B/b - zeroed data - bss (*Block Started by Symbol*)

C - global uninitialized - common (instead of B)

lower-case letter means static (local)

Sections in Memory

Instructions go in .text

Data goes in .data

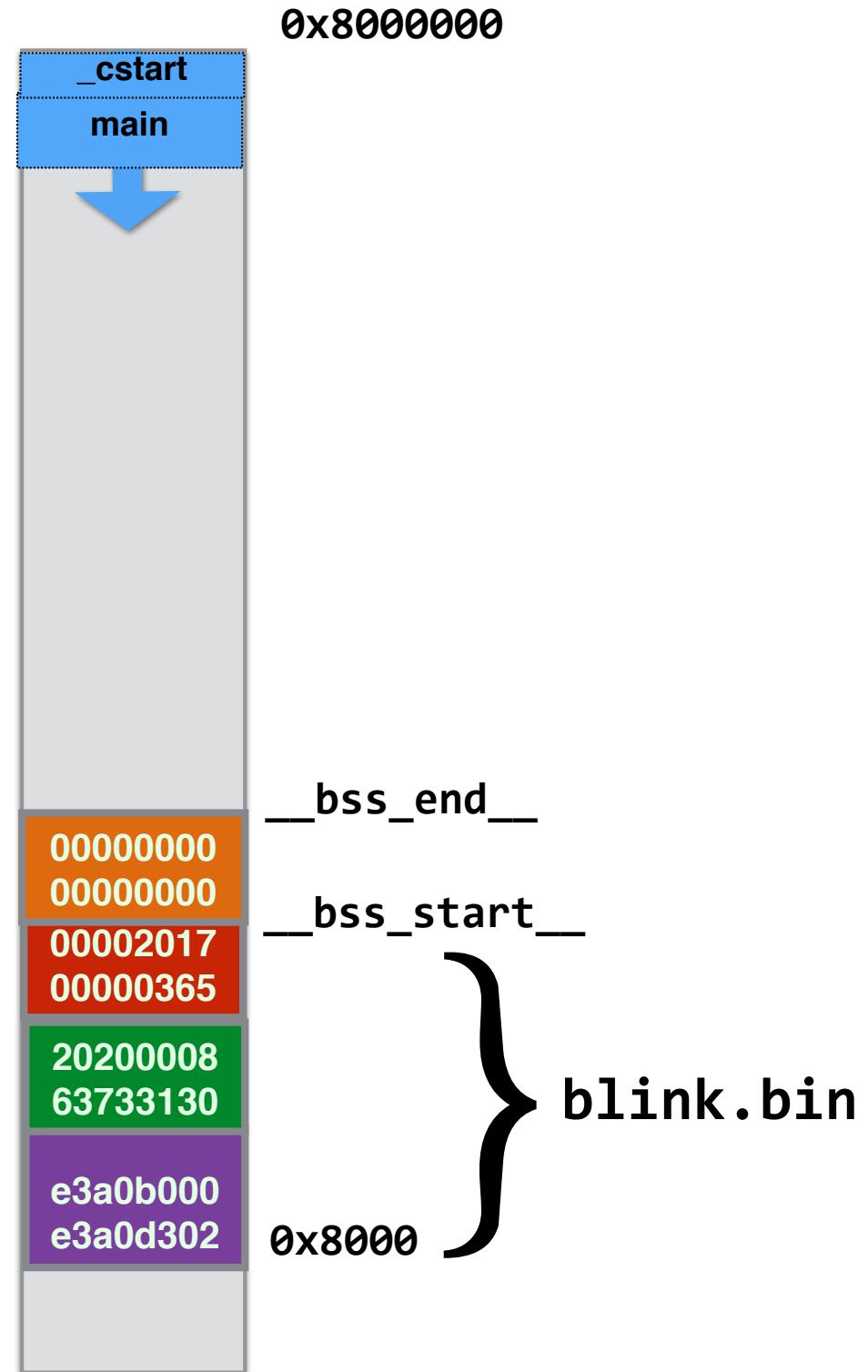
Read-only data (const) goes in .rodata

Zeroed data goes in .bss

SECTIONS

```
{  
    .text 0x8000 : { *(.text.start)  
                      *(.text*) }  
  
    .rodata : { *(.rodata*) }  
    .data : { *(.data*) }  
    __bss_start__ = .;  
    .bss : { *(.bss*)  
              *(COMMON) }  
    __bss_end__ = ALIGN(8);  
}
```

(zeroed data) .bss
(initialized data) .data
(read-only data) .rodata
 .text



```
$ arm-none-eabi-nm -S -n main.elf
00008000 T _start
0000800c t hang
00008010 00000048 T main
00008058 000000d0 T tricky
00008128 0000009c T _cstart
000081c4 00000004 R n
000081c8 00000004 r o
000081cc 00000004 D k
000081d0 00000004 d m
000081d4 D __bss_start__
000081d4 00000004 b j
000081d8 00000004 B l
000081dc 00000004 B i
000081e0 B __bss_end__
```

Triggering a Rebuild

When to Rebuild?

Change to implementation clock.c?

- Must recompile implementation `clock.c` to make `clock.o`

Change to implementation gpio.c?

- Must recompile implementation `gpio.c` to make `gpio.o`

Change to interface gpio.h?

- Should (must) recompile clients of the interface (`clock.c`)
- Add recipe that `clock.c` depends on `gpio.h`

Change to Makefile

- Adding a file to MODULES may require rebuilding `main.elf`
- BEWARE: This is a hidden dependency not tracked by make
- Modify recipe for `main.elf` to depend on Makefile

Builds

Automate the build! Manually typing in commands is error prone

Needs to be fast and reliable

- **Fast means compile modules only when necessary**
- **Reliably means keeping track of dependencies between files**

Separate system into small modules with minimal dependencies

Ensure Makefile contains all dependencies

Summary

Decomposing software into modules is a critical skill

- Like organizing an essay into sections, paragraphs, sentences

The linker combines modules into a larger binary

- Resolves undefined symbols to modules that define them
- Lays out data and code
- Relocation when needed

Makefiles designed to only compile modules that need recompilation

- If F changes, recompile everything that depends on F

Relocation

_start:

mov sp, #0x8000000

mov fp, #0

bl _cstart

hang: b hang

```
// Disassembly of start.o (start.o.list)
```

```
00000000 <_start>:
```

```
 0: e3a0d302      mov sp, #134217728 ;
```

```
0x8000000
```

```
 4: e3a0b000      mov fp, #0
```

```
 8: ebfffffe      bl  0 <_cstart>
```

```
0000000c <hang>:
```

```
c: eaaffffe      b    c <hang>
```

```
// Note: the address of _cstart is 0
```

```
// Why? _start doesn't know where c_start is!
```

```
// Note it does know the address of hang
```

```
// Disassembly of blink.elf.list
```

```
00008000 <_start>:
```

```
  8000: e3a0d302      mov sp, #134217728 ;
```

```
0x8000000
```

```
  8004: e3a0b000      mov fp, #0
```

```
  8008: eb000032      bl  80d8 <_cstart>
```

```
0000800c <hang>:
```

```
  800c: ea\xff\xfe      b   800c <hang>
```

```
000080d8 <_cstart>:
```

```
  80d8: e92d4008      push {r3, lr}
```

```
// Note: _cstart has been placed after hang
```

```
// Now the address of _cstart is #80d8
```

```
// Now _start knows where _cstart is!
```