

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

Modules

- Meaningful decomposition of the system into parts

Interfaces (*.h files) and implementations (*.c)

- Provide natural and easy-to-use abstractions for the users of the module
- Hide unnecessary details of the implementation

Modules tested independently with unit tests

Obi-wan, what is a good design for my modules?

"The Build"

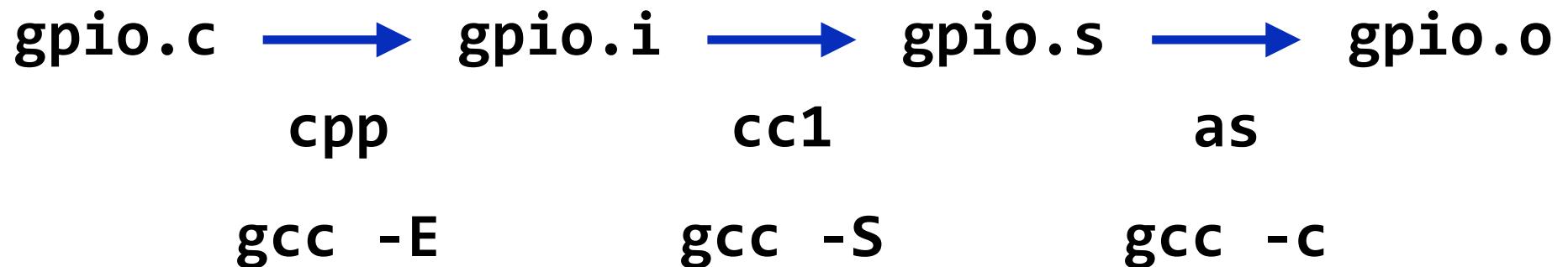


NASA Command Center during SpaceX Mission



cs107e Command Center: Makefile

gcc is all powerful



gcc –save-temp

`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)`

Common Errors

- 1. Symbol undefined**
- 2. Symbol multiplied defined**

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
```

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

```
$ 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
```

```
// _start must be global,  
// by default local.
```

```
// _start must be the first code  
// to execute
```

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

```
// memmap  
MEMORY  
{  
    ram : ORIGIN = 0x8000,  
              LENGTH = 0x8000000  
}  
.text : {  
    start.o (.text)  
    *(.text*)  
} > ram
```

// Why must start.o go first?

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

Symbol Resolution

Set of defined symbols D

Set of undefined symbols U

Moving left to right, for each .o file:

- $D' = D \cup D(\text{file})$**
- $U' = U - D(\text{file})$**

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

Libraries

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

// 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
```

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.

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 tricky.o
00000004 C i
00000000 b j
00000000 D k
00000004 B l
00000000 R n
00000000 T tricky
```

Const variables (n) are marked as read-only

External variables are upper-case, static lower-case

Zeroed global variables (j, l) in bss

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

NB The static const variable m optimized out

Guide to Symbols

T/t - text

D/d - read-write data

R/r - read-only data

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

C - common (instead of B)

lower-case letter means static

Sections in Memory

Instructions go in .text

Data goes in .data

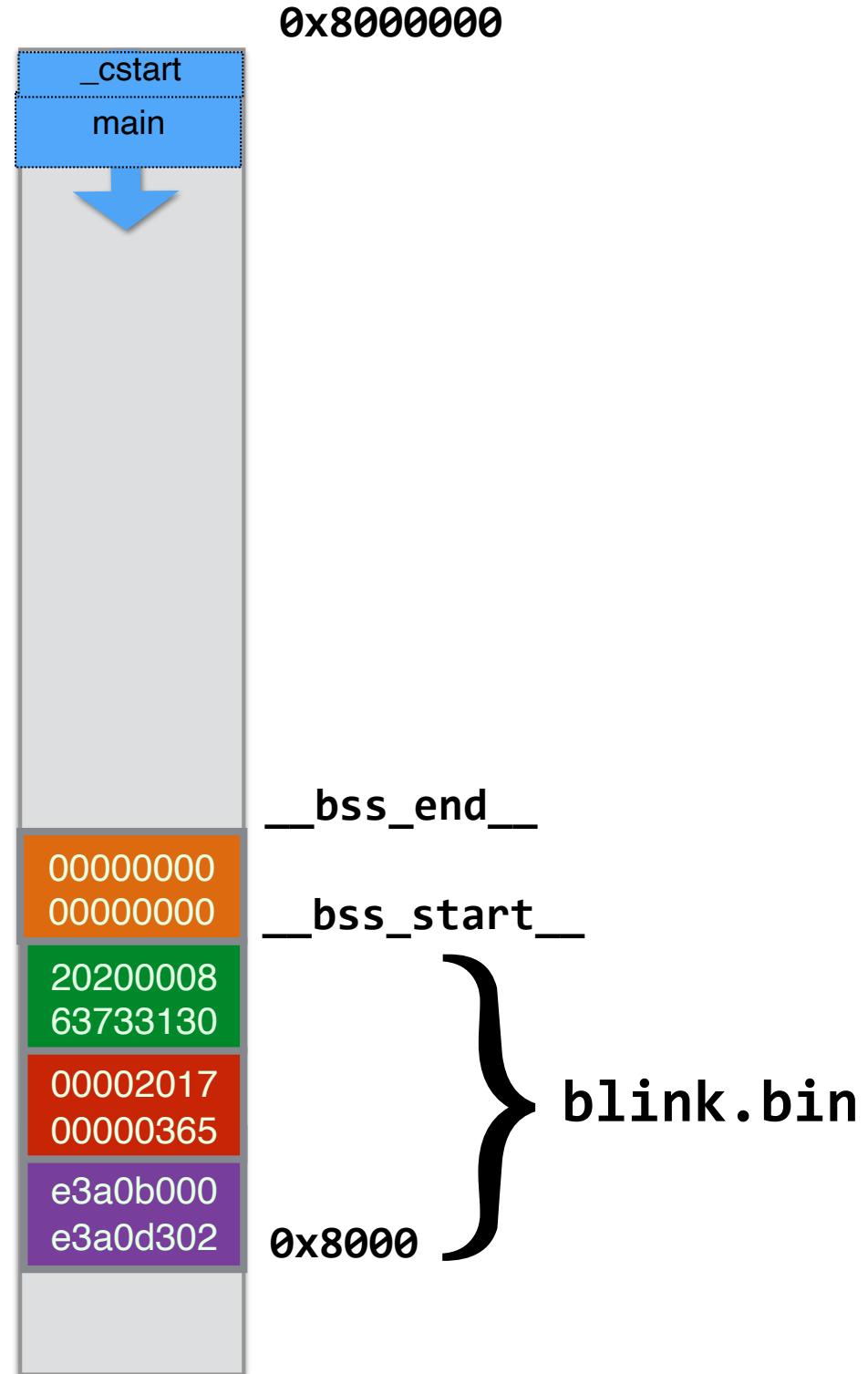
Read-only data (const) goes in .rodata

Zeroed data goes in .bss

SECTIONS

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

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



```
$ cd ../data
$ arm-none-eabi-nm -S -n main.elf
00008000 T _start
0000800c t hang
00008010 00000038 T main
00008048 00000040 T tricky
00008088 00000058 T _cstart
000080e0 00000004 D k
000080e4 00000004 R n
000080e8 R __bss_start__
000080e8 00000004 b j
000080ec 00000004 B l
000080f0 00000004 B i
000080f8 B __bss_end__
```

SECTIONS

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

Triggering a Rebuild

When to Rebuild?

Suppose you change gpio.c, what needs to be recompiled?

Suppose you change gpio.h, what needs to be recompiled?

Could you need to recompile if you changed the Makefile?

When to Rebuild?

Change to implementation (gpio.c)?

- Must recompile implementation (gpio.o)

Change to interface (gpio.h)?

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

Change to Makefile

- Adding a file to OBJECTS may require rebuilding executable main.elf
- Modify recipe for main.elf to depend on Makefile
- BEWARE: This is typical of a hidden dependency

depend/

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

Grading

See <https://cs107e.github.io/policies/#grading-policy>

A:

- **7 basic fully functional + 3 extensions + bonus**
- **ok style/tests**
- **outstanding final project**

B:

- **7 basic mostly functional + 1 extension or bonus**
- **ok style/tests**
- **good final project**

We will file bug reports on your code and give you one week to fix them and earn 1/2 the deducted points back

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: eaffffff      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: the address of _cstart is #80d8
// Now _start knows where _cstart is!