How Computers Work

Processor and memory

Peripherals: GPIO, timers, UART

Assembly and machine language

C

Functions

Serial communication and strings

Linking, loading and starting

Questions for the Curious

What are object files (*.o) [ELF]? What's information is in an object file?

How are object files combined to form an executable?

What is the difference between an executable and a binary?

How do programs start? What is start.s?

How does the boot loader work?

Tools

`make`

`ld`

`nm`

`objdump`

`readelf`

guides/binutils.md **adafruit.com**

```
// nm crib sheet

% nm -h
…
-g // --extern-only
-u // --undefined-only, --defined-only
-S // --print-size -- print valu &size
```

```
// readelf crib sheet

% readelf -h
…
-f         // --file-header
-S         // --section-headers
-s         // --symbols
-x section // hexdump section
-r         // –relocation
```

```
// objdump crib sheet

% objdump -h
…
-d          // --disassemble code
-D          // --disassemble-all
-s          // display section contents
-j name     // --section name
```

code/single

Data

read-write vs read-only (const)

initialized vs uninitialized

global (extern) vs local (static)

Sections

Instructions go in .text

Data goes in .data

const data (read-only) goes in .ro_data

Uninitialized data goes in .bss

+ other information about the program

- symbols, relocation, debugging, ...

Symbols

T/t - text

D/d - read-write data

R/r - read-only data

B/b - bss

C - common

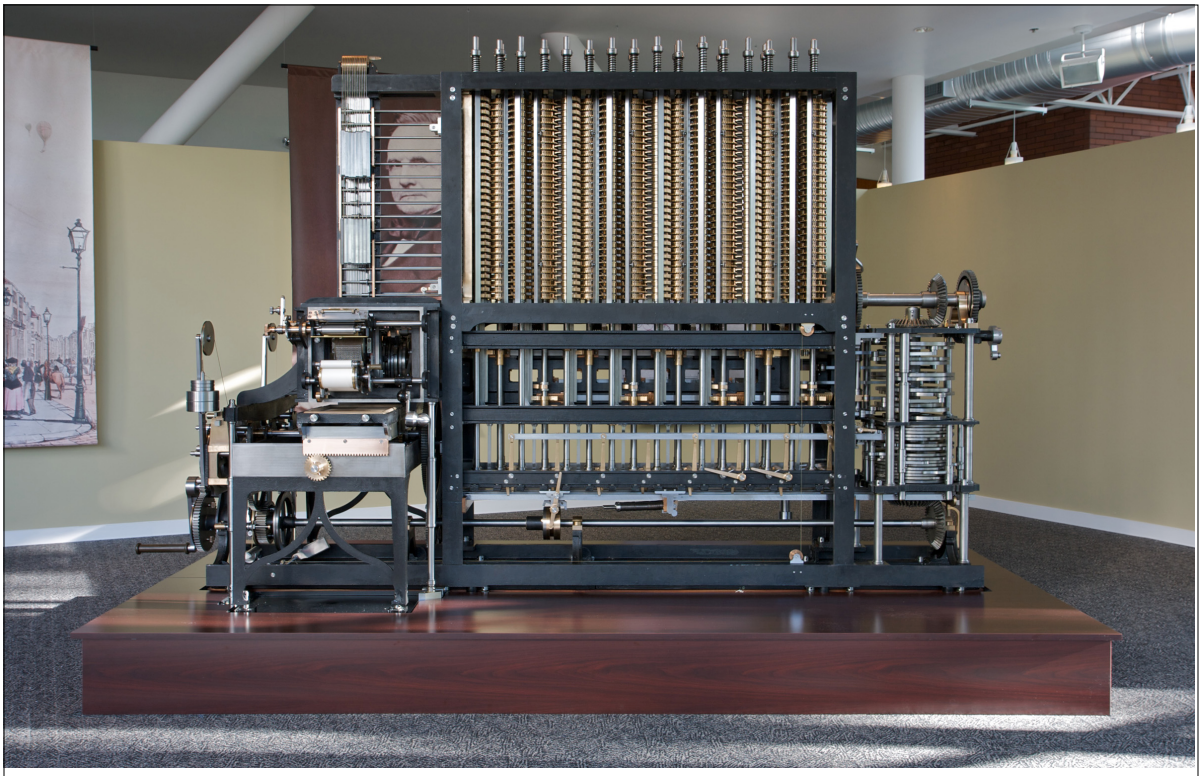
lower-case means static/local

LD: Linker Script

```
MEMORY
{
  ram : ORIGIN = 0x8000,
        LENGTH = 0x1000
}

SECTIONS
{
    .text : { *(.text*) } > ram
    .rodata : { *(.rodata*) } > ram
    .data : { *(.data*) } > ram
    .bss : { *(.bss*) } > ram
}
```

Quinn Dunki



Computer History Museum Tour
11 am Mon Feb 16 (Predident's Day)

Questions?

Loading and Starting

`memap`

- **Start memory at** `0x8000`
- **Place start at** `0x8000`

Loader / Bootloader

- **Program is loaded starting at** `0x8000`
- **Branch to** `0x8000`

`start.s`

- **Initialize stack pointer to** `0x8000`
- **Call** `main`

Initialization of BSS

`memmap`

- **Make space for bss**

`start.s`

- **Initialize stack pointer**
- **call** `cstart`

`cstart.c`

- **Zero bss memory**
- **Call** `main`

```
// Relocation

start:
mov sp, #0x8000
b main

Disassembly of start.o
0: e3a0d902   mov sp, #0x8000
4: ebfffffe   bl 0 <main>
```

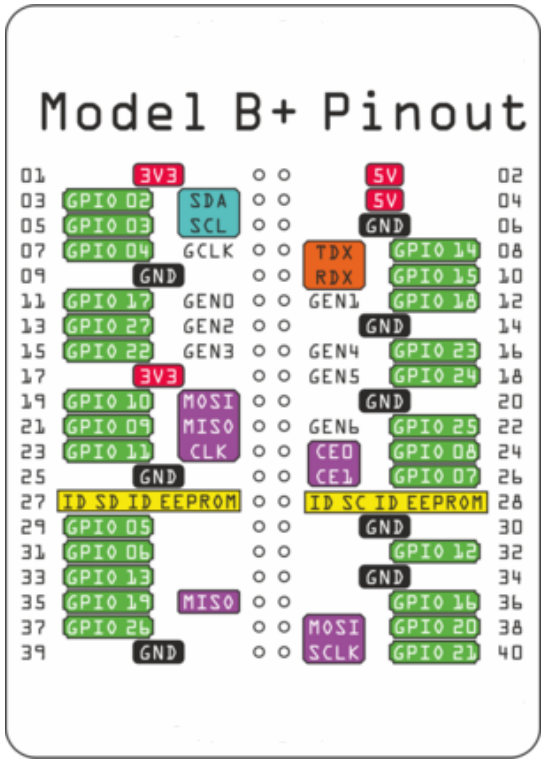
```
// Relocation

Disassembly of start.o
   0:e3a0d902   mov   sp, #0x8000
   4:ebfffffe   bl 0 <cstart>

Disassembly of main.exe
8000: e3a0d902   mov sp, #x8000
8004: eb000030   bl 80cc <cstart>
…
000080cc <cstart>:
80cc: e92d4800   push {fp, lr}
…
```

Assignment 3

stdarg.h



	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	<reserved>	BSCSL SDA / MOSI	SPI1_CE0_N	PWM0
GPIO19	Low	PCM_FS	SD11	<reserved>	BSCSL SCL / SCLK	SPI1_MISO	PWM1
GPIO20	Low	PCM_DIN	SD12	<reserved>	BSCSL / MISO	SPI1_MOSI	GPCLK0
GPIO21	Low	PCM_DOUT	SD13	<reserved>	BSCSL / CE_N	SPI1_SCLK	GPCLK1
GPIO22	Low	<reserved>	SD14	<reserved>	SD1_CLK	ARM_TRST	
GPIO23	Low	<reserved>	SD15	<reserved>	SD1_CMD	ARM_RTCK	
GPIO24	Low	<reserved>	SD16	<reserved>	SD1_DAT0	ARM_TDO	
GPIO25	Low	<reserved>	SD17	<reserved>	SD1_DAT1	ARM_TCK	
GPIO26	Low	<reserved>	<reserved>	<reserved>	SD1_DAT2	ARM_TDI	
GPIO27	Low	<reserved>	<reserved>	<reserved>	SD1_DAT3	ARM_TMS	
GPIO28	-	SDA0	SA5	PCM_CLK	<reserved>		
GPIO29	-	SCL0	SA4	PCM_FS	<reserved>		
GPIO30	Low	<reserved>	SA3	PCM_DIN	CTS0		CTS1
GPIO31	Low	<reserved>	SA2	PCM_DOUT	RTS0		RTS1
GPIO32	Low	GPCLK0	SA1	<reserved>	TXD0		TXD1
GPIO33	Low	<reserved>	SA0	<reserved>	RXD0		RXD1
GPIO34	High	GPCLK0	SOE_N / SE	<reserved>	<reserved>		
GPIO35	High	SPI0_CE1_N	SWE_N / SRW_N		<reserved>		
GPIO36	High	SPI0_CE0_N	SD0	TXD0	<reserved>		
GPIO37	Low	SPI0_MISO	SD1	RXD0	<reserved>		
GPIO38	Low	SPI0_MOSI	SD2	RTS0	<reserved>		
GPIO39	Low	SPI0_SCLK	SD3	CTS0	<reserved>		
GPIO40	Low	PWM0	SD4		<reserved>	SPI2_MISO	TXD1
	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5

Summary

Symbols

- **extern vs static**
- **initialized vs uninitialized**
- **const int vs int**

Sections

- **text, data, rodata, bas**

Summary

Startup

- **Start location**
- **Initialize memory: stack pointer and bss**

Relocation

- **Concatenating code**
- **Patching addresses**

Hadas64

JIM GOGARTY 2011