Embedded Software Essentials

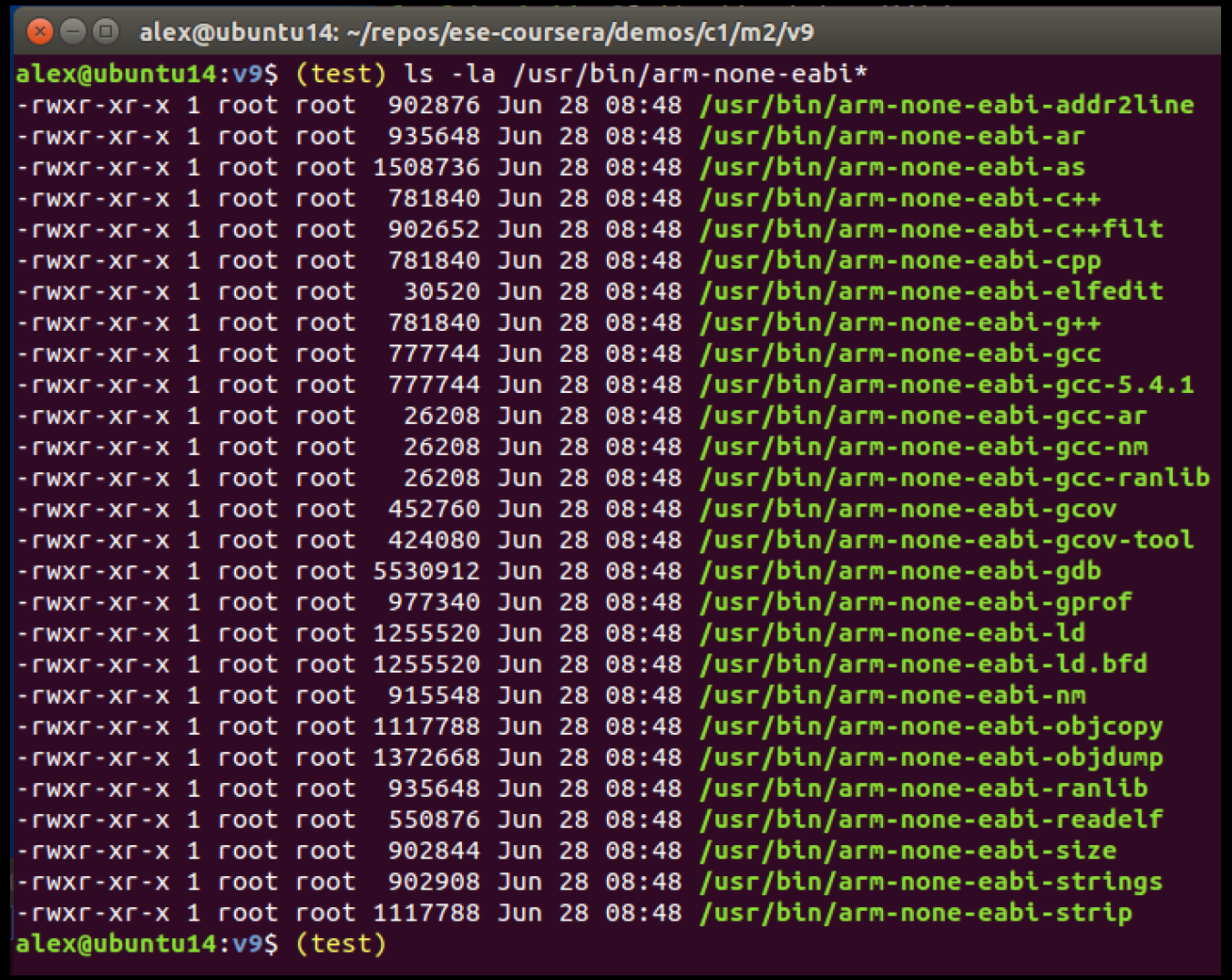
Other Useful GNU Tools

C1 M2 V9

Copyright

GNU Binary Utilities

- Extra programs to help with the process of building
- These include
  - The Assembler (ar)
  - The Linker (ld)
  - Conversion of executables
  - Sizing compiled images
  - Library/Archive creation
  - Symbol Listing
  - Debugging
  - Many more!



Useful GNU Tools

Name	Purpose	ARM Executable
size	Lists the section sizes for object and executable files	arm-none-eabi-size
nm	Lists the symbols from object files	arm-none-eabi-nm
objcopy	Copies and translates object files	arm-none-eabi-objcopy
objdump	Displays information from object files	arm-none-eabi-objdump
readelf	Displays information from elf files	arm-none-eabi-readelf
gdb	GNU Project Debugger	gdb

GNU Size Utility

- Use GCC's size to display the sizes of the compiled sections inside your object files and executable file outputs

- Gives you an idea of your memory footprint is for you executable
  - Code Memory
  - Data Memory

```
alex@ubuntu14:v9$ (test) arm-none-eabi-size -Atd demo.out
demo.out  :
section              size      addr
.init                  12     32768
.text               34600     32780
.fini                  12     67380
.rodata              1312     67392
.ARM.exidx              8     68704
.eh_frame               4     68712
.init_array             8    134252
.fini_array             4    134260
.jcr                    4    134264
.data                2244    134272
.bss                  112    136516
.stab                 156         0
.stabstr              333         0
.comment              110         0
.debug_frame         4700         0
.ARM.attributes        40         0
Total               43659
```

All Memory sections compiled in output executable

```
alex@ubuntu14:v9$ (test) arm-none-eabi-size -Bx demo.out
   text    data     bss     dec     hex filename
 0x8c6c    0x8d4    0x70   38320    95b0 demo.out
alex@ubuntu14:v9$ (test) arm-none-eabi-size -Bd demo.out
   text    data     bss     dec     hex filename
  35948    2260     112   38320    95b0 demo.out
```

HEX

Decimal

```
alex@ubuntu14:v9$ (test) arm-none-eabi-size -Btd demo.out main.o my_file.o
   text    data     bss     dec     hex filename
  35948    2260     112   38320    95b0 demo.out
     30       0       0      30      1e main.o
    192       0       0     192      c0 my_file.o
  36170    2260     112   38542    968e (TOTALS)
```

Each File Section size

NM Utility

- The symbol utility allows us to investigate the size of all the possible symbols that are defined in a given executable or object file

- Symbols are identifiers in your source code that can be referenced
  - Variables
  - Functions
  - Debug

Symbol Descriptions
- **T**: Code
- **R**: Read Only
- **D**: Initialized Data
- **B**: Uninitialized Data (BSS)

```
alex@ubuntu14:v9$ (test) arm-none-eabi-nm -S --defined --size-sort -s demo.out
000084f8 00000002 T _exit
0000850c 00000004 R _global_impure_ptr
00018960 00000004 D _impure_ptr
00018980 0000000a B memory
000083e0 00000010 T atexit
0000802c 00000018 t register_fini
00008194 0000001a T get_value
000081f0 0000001e T clear_all
00008174 0000001e T clear_value
0000800c 00000020 T exit
00008150 00000022 T set_value
000083f0 00000034 T __libc_fini_array
0000811c 00000034 T main
000081b0 0000003e T set_all
00008210 00000048 T __libc_init_array
00008258 0000008c T memset
00008424 000000d4 T __register_exitproc
000082e4 000000fc T __call_exitprocs
00018538 00000428 d impure_data
```

All defined symbols (variables and code) in the output executable

NM example

- ***my_file.c*** contains 5 function definitions, no global variables
- Ending Code Address: 0xBE
  - ~190 Bytes of code Memory

- **Biggest Function**: *set_all*
- **Smallest Function**: *clear_value, set_value*

my_file.c Symbol Table

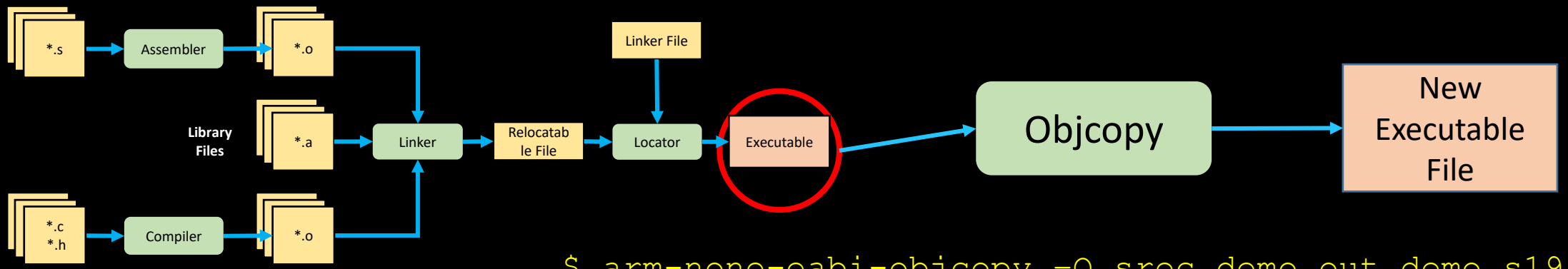
Name	Starting Address	Code Size [Bytes]
clear_all	0x00	0x22
clear_value	0x24	0z1E
get_value	0x44	0x1A
set_all	0x60	0x3E
set_value	0xA0	0x1E

```
alex@ubuntu14:v9$ (test) arm-none-eabi-nm -S --defined -s my_file.o
000000a0 0000001e T clear_all
00000024 0000001e T clear_value
00000044 0000001a T get_value
00000060 0000003e T set_all
00000000 00000022 T set_value
```

Objcopy Utility

- The object copy utility is used to:
  - Convert objects files from one format to another
  - Make a copy of an object file
- Executables are a type of object file
  - Many different object file formats

Object Format Types:
- Binary
- srec (Motorola S-record)
- intel Hex Record (ihex)
- elf32-littlearm
- elf32-bigarm




```
$ arm-none-eabi-objcopy -O srec demo.out demo.s19
```

Objdump Utility

- Dumps information about an object file
  - Section Headers
  - Symbols
  - Debugging
- Can take object files and dump assembly from the machine code
  - Regular object files
  - Output Executables

```
demo:     file format elf32-littlearm


Disassembly of section .init:

00008000 <_init>:
    8000:       b5f8            push    {r3, r4, r5, r6, r7, lr}
    8002:       46c0            nop                     ; (mov r8, r8)
    8004:       bcf8            pop     {r3, r4, r5, r6, r7}
    8006:       bc08            pop     {r3}
    8008:       469e            mov     lr, r3
    800a:       4770            bx      lr

Disassembly of section .text:

0000800c <exit>:
    800c:       b510            push    {r4, lr}
    800e:       2100            movs    r1, #0
    8010:       0004            movs    r4, r0
    8012:       f000 f967       bl      82e4 <__call_exitprocs>
    8016:       4b04            ldr     r3, [pc, #16]   ; (8028 <exit+0x1c>)
    8018:       6818            ldr     r0, [r3, #0]
    801a:       6bc3            ldr     r3, [r0, #60]   ; 0x3c
    801c:       2b00            cmp     r3, #0
    801e:       d000            beq.n   8022 <exit+0x16>
    8020:       4798            blx     r3
    8022:       0020            movs    r0, r4
    8024:       f000 fa68       bl      84f8 <_exit>
    8028:       0000850c        .word   0x0000850c
```

Address

Machine code

Assembly Instructions

Objdump Utility

- Debug symbols allow your C-program and assembly to intermix
- Debugger uses this debug software

C-programming Statements

```
main.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
#include "my_memory.h"

extern char memory[MAX_LENGTH];

/* A pretty boring main file */
int main(void){
   0:   b580            push    {r7, lr}
   2:   af00            add     r7, sp, #0

  clear_all(memory, MAX_LENGTH);
   4:   4b0a            ldr     r3, [pc, #40]   ; (30 <main+0x30>)
   6:   210a            movs    r1, #10
   8:   0018            movs    r0, r3
   a:   f7ff fffe       bl      0 <clear_all>
  set_value(memory, 0xAA, 0);
   e:   4b08            ldr     r3, [pc, #32]   ; (30 <main+0x30>)
  10:   2200            movs    r2, #0
  12:   21aa            movs    r1, #170        ; 0xaa
  14:   0018            movs    r0, r3
  16:   f7ff fffe       bl      0 <set_value>
  set_value(memory, 0xFF, 1);
  1a:   4b05            ldr     r3, [pc, #20]   ; (30 <main+0x30>)
  1c:   2201            movs    r2, #1
  1e:   21ff            movs    r1, #255        ; 0xff
  20:   0018            movs    r0, r3
  22:   f7ff fffe       bl      0 <set_value>

  return 0;
  26:   2300            movs    r3, #0
}
```

Address

Machine code

Assembly Instructions

Readelf Utility

- Displays information about a ELF formatted file
  - Compiled Sections
  - Memory Sections
  - Symbol Tables
  - Architecture Specifics

ELF Files are not human readable they contain lots of hidden information in binary data

```
alex@ubuntu14:v9$ (test) arm-none-eabi-readelf demo.out --all
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x80a5
  Start of program headers:          52 (bytes into file)
  Start of section headers:          39140 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         18
  Section header string table index: 15

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .init             PROGBITS        00008000 008000 00000c 00  AX  0   0  4
  [ 2] .text             PROGBITS        0000800c 00800c 0004f0 00  AX  0   0  4
  [ 3] .fini             PROGBITS        000084fc 0084fc 00000c 00  AX  0   0  4
  [ 4] .rodata           PROGBITS        00008508 008508 000008 00   A  0   0  4
  [ 5] .ARM.exidx        ARM_EXIDX       00008510 008510 000008 00  AL  2   0  4
  [ 6] .eh_frame         PROGBITS        00008518 008518 000004 00   A  0   0  4
  [ 7] .init_array       INIT_ARRAY      0001851c 00851c 000008 00  WA  0   0  4
  [ 8] .fini_array       FINI_ARRAY      00018524 008524 000004 00  WA  0   0  4
  [ 9] .jcr              PROGBITS        00018528 008528 000004 00  WA  0   0  4
  [10] .data             PROGBITS        00018530 008530 000434 00  WA  0   0  8
  [11] .bss              NOBITS          00018964 008964 000028 00  WA  0   0  4
  [12] .comment          PROGBITS        00000000 008964 00006e 01  MS  0   0  1
  [13] .debug_frame      PROGBITS        00000000 0089d4 000184 00      0   0  4
  [14] .ARM.attributes   ARM_ATTRIBUTES  00000000 008b58 000028 00      0   0  1
  [15] .shstrtab         STRTAB          00000000 009849 000098 00      0   0  1
  [16] .symtab           SYMTAB          00000000 008b80 0008b0 10     17  99  4
  [17] .strtab           STRTAB          00000000 009430 000419 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```