The build-process in C

Abdelrahman Aboghanima

February 23, 2023

What is the build process

The process of converting the

`*.c, *.h` files, the human readable code, to machine code.

Preprocessor

The preprocessor provides the ability for the inclusion of header files, macro expansions, conditional compilation, and line control.

Example

```c
#ifndef _TYPES_H_
#define _TYPES_H_

typedef unsigned char u8;

#endif /* _TYPES_H_ */
```

```c
#ifndef _FUNC1_H_
#define _FUNC1_H_
#include "types.h"

u8 func1(u8 num);

#endif /* _FUNC1_H_ */
```

```c
/*func1.c*/
#include "func1.h"
#include "types.h"

#define X 55

u8   initialized_var  =55;
const u8  const_global_var = 70;
u8   uninitialized_var  ;

u8 func1(u8 num)
{
  u8  local_var =10;
  return  num+local_var+X;
}
```

```
$ gcc -E -P func1.c
```

Preprocessor Output

```
1  typedef unsigned char u8;
2  u8 func1(u8 num);
3  u8 initialized_var=55;
4  const u8 const_global_var= 70;
5  u8 uninitialized_var;
6  u8 func1(u8 num)
7  {
8    u8 local_var=10;
9    return num+local_var+55;
10 }
```

Compiler

The process of converting the portable C code into the architecture specific assembly code The compiler takes a translation unit (A preporecessed source code) and converts it to assembly code.

*.i → **Compiler** → *.asm

```
1   $ gcc -S func1.c
```

```
1   $ avr-gcc -S func1.c
```

```
1   $ arm-none-eabi-gcc -S func1.c
```

```
    .file   "func1.c"
    .text
    .globl  func1
    .type   func1, @function
func1:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    %edi, %eax
    movb    %al, -4(%rbp)
    movzbl  -4(%rbp), %eax
    addl    $55, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   func1, .-func1
    .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"
    .section .note.GNU-stack,"",@progbits
```

```
    .cpu arm7tdmi
    .arch armv4t
    .fpu softvfp
    .eabi_attribute 20, 1
    .eabi_attribute 21, 1
    .eabi_attribute 23, 3
    .eabi_attribute 24, 1
    .eabi_attribute 25, 1
    .eabi_attribute 26, 1
    .eabi_attribute 30, 6
    .eabi_attribute 34, 0
    .eabi_attribute 18, 4
    .file   "func1.c"
    .text
    .align  2
    .global func1
    .syntax unified
    .arm
    .type   func1, %function
func1:
    @ Function supports interworking.
    @ args = 0, pretend = 0, frame = 8
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    str     fp, [sp, #-4]!
    add     fp, sp, #0
    sub     sp, sp, #12
    mov     r3, r0
    strb    r3, [fp, #-5]
    ldrb    r3, [fp, #-5]
    add     r3, r3, #55
    and     r3, r3, #255
    mov     r0, r3
    add     sp, fp, #0
    @ sp needed
    ldr     fp, [sp], #4
    bx      lr
    .size   func1, .-func1
    .ident  "GCC: (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824 (release)"
```

```
    .file   "func1.c"
__SP_H__ = 0x3e
__SP_L__ = 0x3d
__SREG__ = 0x3f
__tmp_reg__ = 0
__zero_reg__ = 1
    .text
.global  func1
    .type   func1, @function
func1:
    push r28
    push r29
    push __zero_reg__
    in  r28,__SP_L__
    in  r29,__SP_H__
/* prologue: function */
/* frame size = 1 */
/* stack size = 3 */
.L__stack_usage = 3
    std Y+1,r24
    ldd r24,Y+1
    subi r24,lo8(-(55))
/* epilogue start */
    pop __tmp_reg__
    pop r29
    pop r28
    ret
    .size   func1, .-func1
    .ident  "GCC: (GNU) 5.4.0"
```

Assembler

The process of transforming the architecture assembly code to its assocaiated machine code. The assembler takes the assembly code and converts it to a machine code in an *object file*

*.asm → **Assembler** → *.o

```
1  $ gcc -c func1.s -o func1.x68.o
```

```
1  $ avr-gcc -c func1.s -o func1.avr.o
```

```
1  $ arm-none-eabi-gcc -c func1.s -o func1.arm.o
```

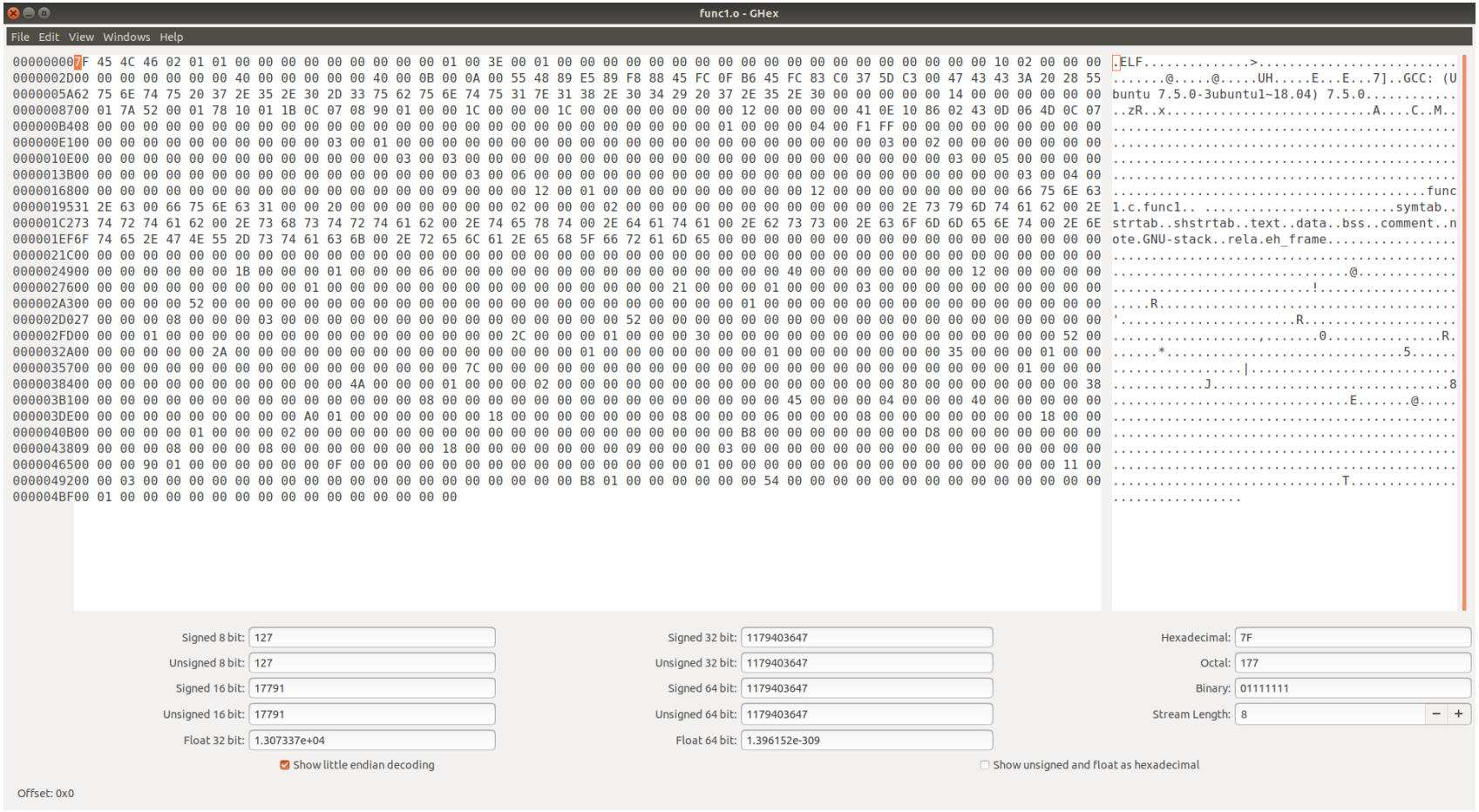


Figure: The machine code for x86 architecture.

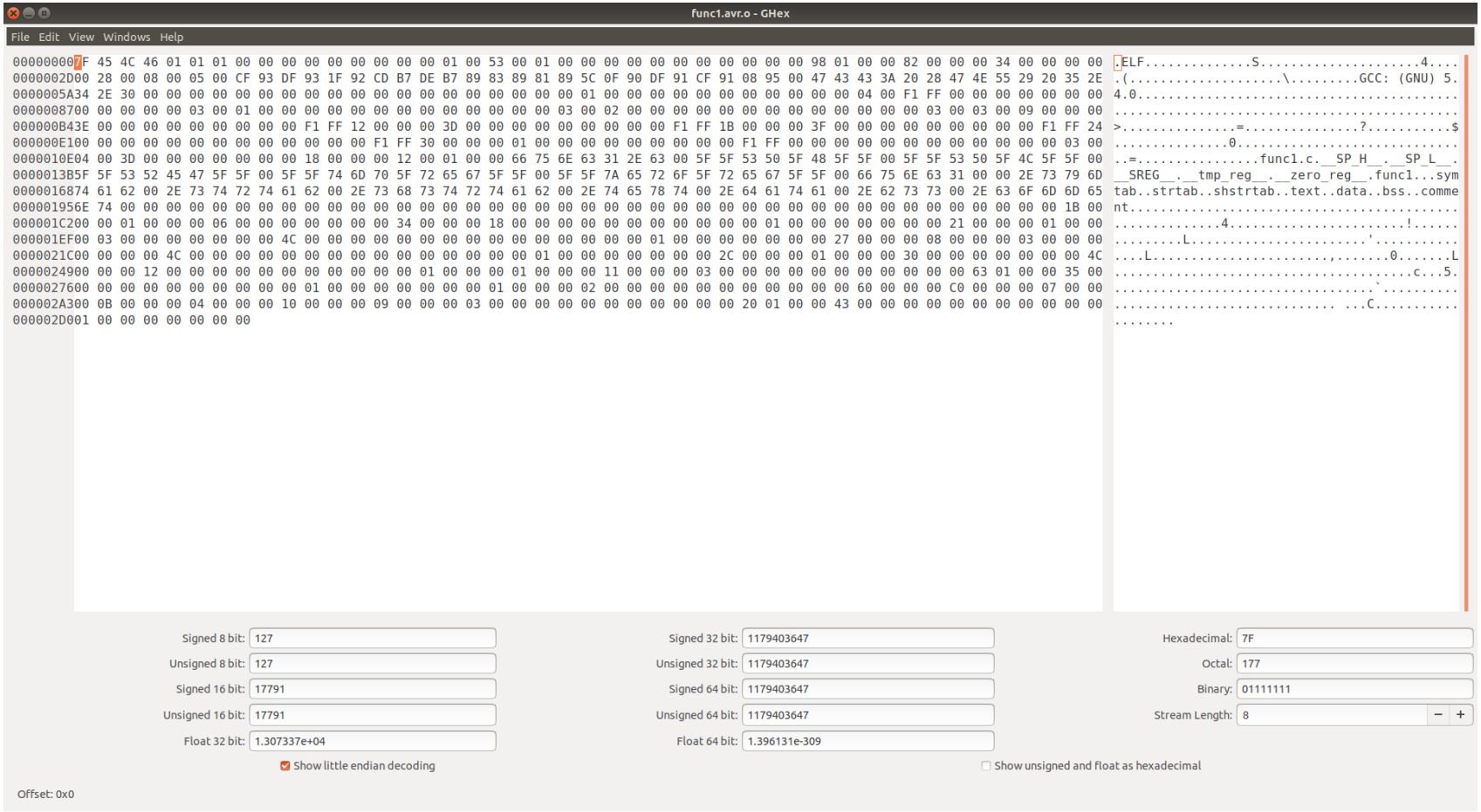


Figure: The machine code for avr architecture.

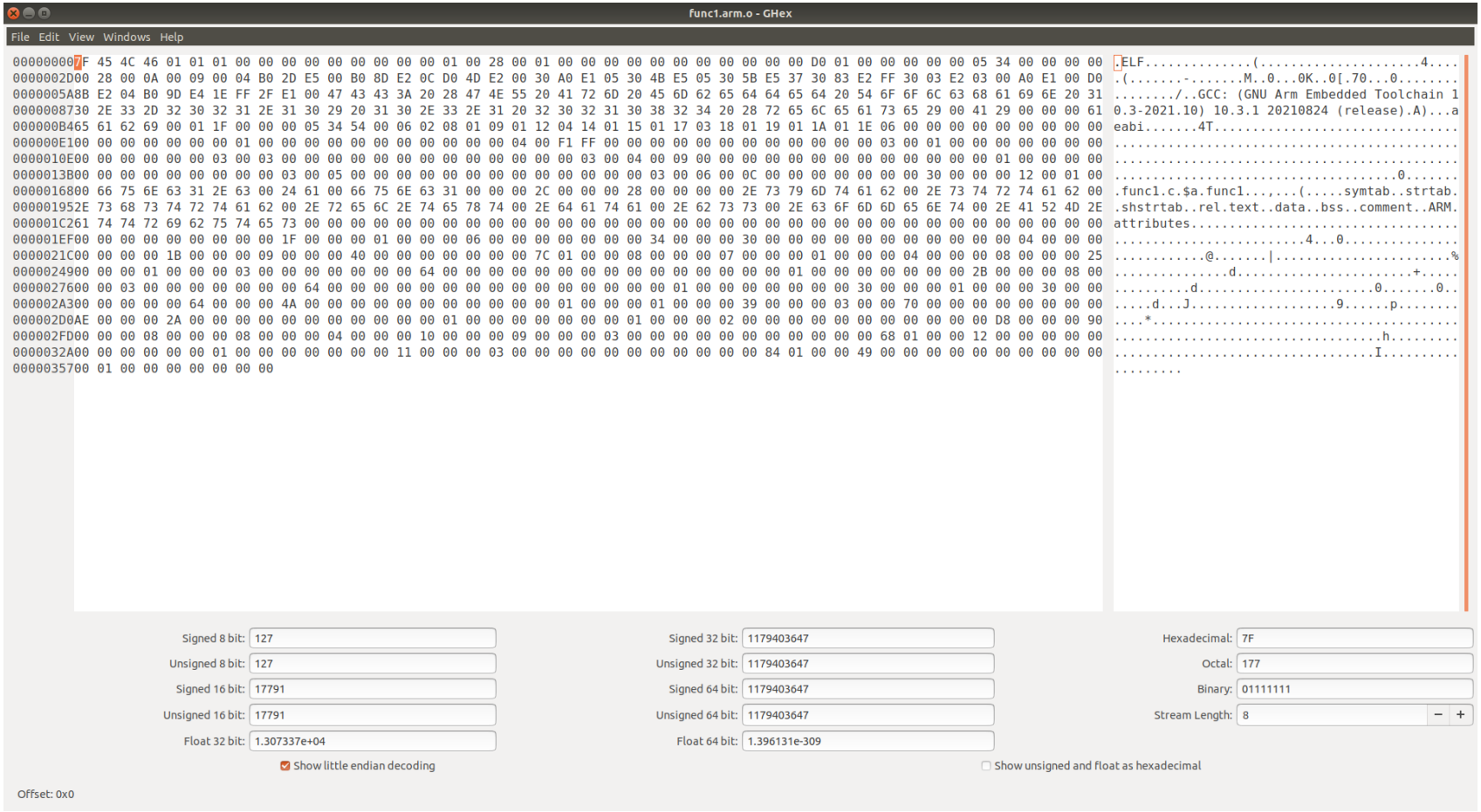


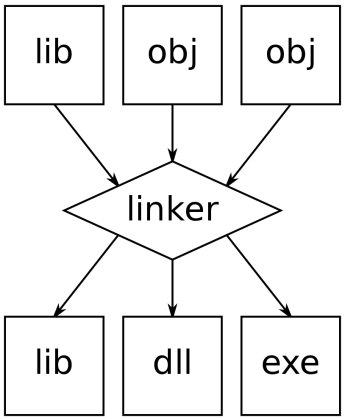
Figure: The machine code for arm architecture.

Linker

Definition

Linker: is a computer system program that takes one or more object files (generated by a compiler or an assembler) and combines them into a single executable file, library file, or another "object" file.

*.o, *.a or *.so or *.dll → **Linker** → relocatable file



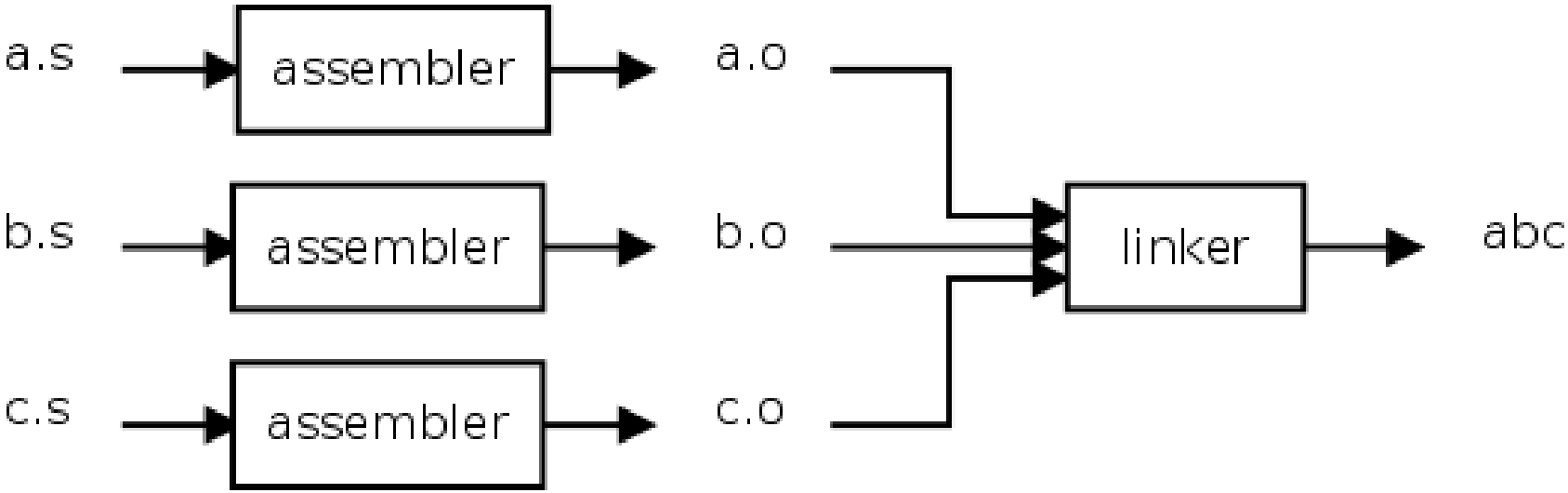


Figure: The linker inputs and outputs.

Types of linking

- Static Linking

Details

The code for all routines called by your program becomes part of the executable file.

Types of linking

- Static Linking
- Dynamic Linking

Details

Every dynamically linked program contains a small, statically linked function that is called when the program starts. This static function only maps the link library into memory and runs the code that the function contains. The link library determines what are all the dynamic libraries which the program requires along with the names of the variables and functions needed from those libraries by reading the information contained in sections of the library.

Linking our program

```
1   $ avr-ld main.o func1.o && avr-objdump -d a.out
```

```
Disassembly of section .text:
00000000 <main>:
   0: cf 93        push r28
   2: df 93        push r29
   4: cd b7        in r28, 0x3d  ; 61
   6: de b7        in r29, 0x3e  ; 62
   8: 80 e0        ldi  r24, 0x00  ; 0
   a: 90 e0        ldi  r25, 0x00  ; 0
   c: df 91        pop  r29
   e: cf 91        pop  r28
  10: 08 95        ret
00000012 <func1>:
  12: cf 93        push r28
  14: df 93        push r29
  16: 00 d0        rcall .+0       ; 0x18 <func1+0x6>
  18: cd b7        in r28, 0x3d  ; 61
  1a: de b7        in r29, 0x3e  ; 62
  1c: 8a 83        std  Y+2, r24 ; 0x02
  1e: 8a e0        ldi  r24, 0x0A  ; 10
  20: 89 83        std  Y+1, r24 ; 0x01
  22: 9a 81        ldd  r25, Y+2 ; 0x02
  24: 89 81        ldd  r24, Y+1 ; 0x01
  26: 89 0f        add  r24, r25
  28: 89 5c        subi r24, 0xC9  ; 201
  2a: 0f 90        pop  r0
  2c: 0f 90        pop  r0
  2e: df 91        pop  r29
  30: cf 91        pop  r28
  32: 08 95        ret
```

Locator

The locator main process is mapping the memory virtual addrsses of the sections to the physical addresses with the aid of the *linker script file.*


```
1  SECTIONS
2  {
3    . = 0x10000;
4    .text : { *(.text) }
5    . = 0x8000000;
6    .data : { *(.data) }
7    .bss : { *(.bss) }
8  }
```

Listing 1: A simple linker script file

```
abdo@abdo:~/buid-process/ex$ avr-gcc -mmcu=atmega32 main.o func1.o
abdo@abdo:~/buid-process/ex$ avr-objdump -d a.out

a.out:     file format elf32-avr


Disassembly of section .text:

00000000 <__vectors>:
   0:   0c 94 2a 00     jmp     0x54    ; 0x54 <__ctors_end>
   4:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
   8:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
   c:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  10:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  14:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  18:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  1c:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  20:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  24:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  28:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  2c:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  30:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  34:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  38:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  3c:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  40:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  44:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  48:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  4c:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>
  50:   0c 94 47 00     jmp     0x8e    ; 0x8e <__bad_interrupt>

00000054 <__ctors_end>:
  54:   11 24           eor     r1, r1
  56:   1f be           out     0x3f, r1        ; 63
  58:   cf e5           ldi     r28, 0x5F       ; 95
  5a:   d8 e0           ldi     r29, 0x08       ; 8
  5c:   de bf           out     0x3e, r29       ; 62
  5e:   cd bf           out     0x3d, r28       ; 61

00000060 <__do_copy_data>:
  60:   10 e0           ldi     r17, 0x00       ; 0
  62:   a0 e6           ldi     r26, 0x60       ; 96
  64:   b0 e0           ldi     r27, 0x00       ; 0
  66:   ea ec           ldi     r30, 0xCA       ; 202
  68:   f0 e0           ldi     r31, 0x00       ; 0
  6a:   02 c0           rjmp    .+4             ; 0x70 <__do_copy_data+0x10>
  6c:   05 90           lpm     r0, Z+
  6e:   0d 92           st      X+, r0
  70:   a4 36           cpi     r26, 0x64       ; 100
  72:   b1 07           cpc     r27, r17
  74:   d9 f7           brne    .-10            ; 0x6c <__do_copy_data+0xc>
```

```
00000076 <__do_clear_bss>:
  76:   20 e0           ldi     r18, 0x00       ; 0
  78:   a4 e6           ldi     r26, 0x64       ; 100
  7a:   b0 e0           ldi     r27, 0x00       ; 0
  7c:   01 c0           rjmp    .+2             ; 0x80 <.do_clear_bss_start>

0000007e <.do_clear_bss_loop>:
  7e:   1d 92           st      X+, r1

00000080 <.do_clear_bss_start>:
  80:   a5 36           cpi     r26, 0x65       ; 101
  82:   b2 07           cpc     r27, r18
  84:   e1 f7           brne    .-8             ; 0x7e <.do_clear_bss_loop>
  86:   0e 94 49 00     call    0x92    ; 0x92 <main>
  8a:   0c 94 63 00     jmp     0xc6    ; 0xc6 <_exit>

0000008e <__bad_interrupt>:
  8e:   0c 94 00 00     jmp     0       ; 0x0 <__vectors>

00000092 <main>:
  92:   cf 93           push    r28
  94:   df 93           push    r29
  96:   cd b7           in      r28, 0x3d       ; 61
  98:   de b7           in      r29, 0x3e       ; 62
  9a:   80 e0           ldi     r24, 0x00       ; 0
  9c:   90 e0           ldi     r25, 0x00       ; 0
  9e:   df 91           pop     r29
  a0:   cf 91           pop     r28
  a2:   08 95           ret

000000a4 <func1>:
  a4:   cf 93           push    r28
  a6:   df 93           push    r29
  a8:   00 d0           rcall   .+0             ; 0xaa <func1+0x6>
  aa:   cd b7           in      r28, 0x3d       ; 61
  ac:   de b7           in      r29, 0x3e       ; 62
  ae:   8a 83           std     Y+2, r24        ; 0x02
  b0:   8a e0           ldi     r24, 0x0A       ; 10
  b2:   89 83           std     Y+1, r24        ; 0x01
  b4:   9a 81           ldd     r25, Y+2        ; 0x02
  b6:   89 81           ldd     r24, Y+1        ; 0x01
  b8:   89 0f           add     r24, r25
  ba:   89 5c           subi    r24, 0xC9       ; 201
  bc:   0f 90           pop     r0
  be:   0f 90           pop     r0
  c0:   df 91           pop     r29
  c2:   cf 91           pop     r28
  c4:   08 95           ret

000000c6 <_exit>:
  c6:   f8 94           cli

000000c8 <__stop_program>:
  c8:   ff cf           rjmp    .-2             ; 0xc8 <__stop_program>
```

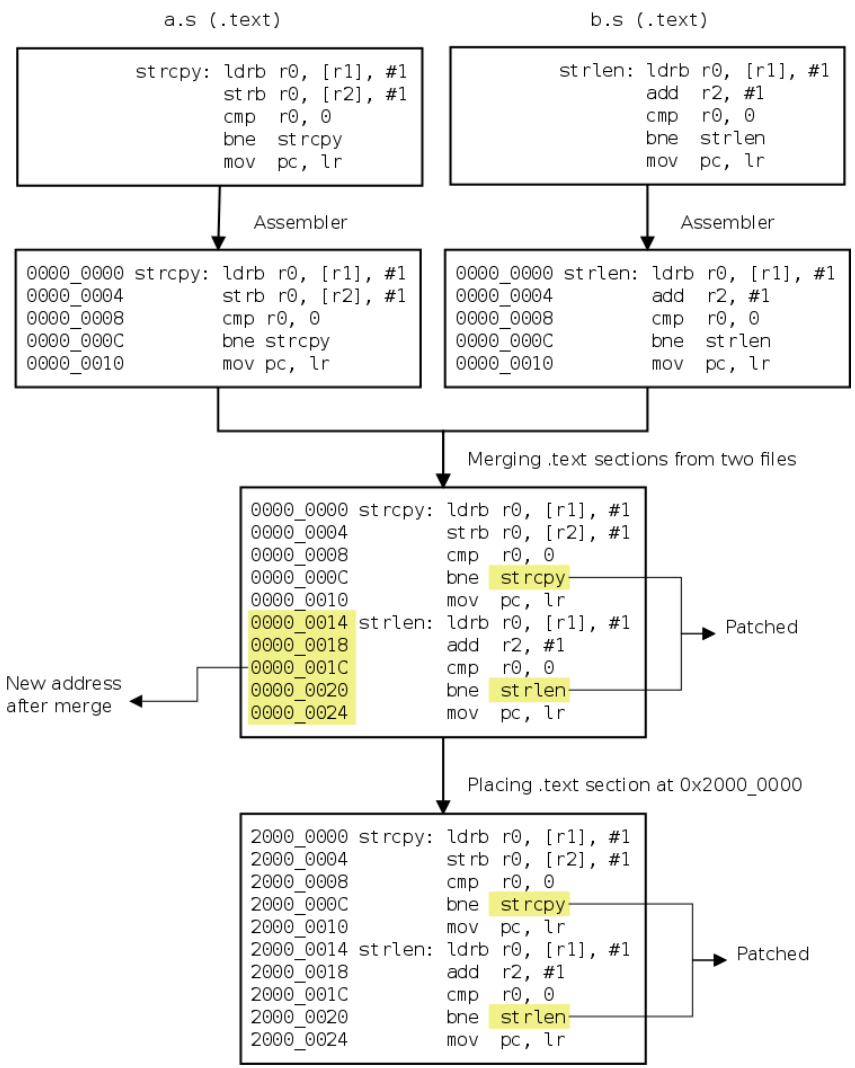


Figure: Merging and placement done by linker and the locator.

Refrences


Relocation (computing)


Object File


Linker


Static and Dynamic Linking


Dynamic and static linking, IBM


Linker, Bravegnu