

Report

- **Debugging:**

-To debug the code we have to make sure that we have debug sections like that

```
ziade@LAPTOP-S0076FNK MINGW64 /d/Lab_1
$ readelf.exe learn-in-depth.elf -S
There are 16 section headers, starting at offset 0x85ac:

Section Headers:
[Nr] Name                Type           Addr          Off           Size      ES Flg Lk Inf Al
[ 0]                      NULL           00000000      000000      000000      00  0  0  0  0
[ 1] .startup                PROGBITS       00010000      008000      000010      00  AX  0  0  4
[ 2] .text                  PROGBITS       00010010      008010      000068      00  AX  0  0  4
[ 3] .data                  PROGBITS       00010078      008078      000064      00  WA  0  0  4
[ 4] .ARM.attributes        ARM_ATTRIBUTES 00000000      0080dc      00002e      00  0  0  0  1
[ 5] .comment               PROGBITS       00000000      00810a      000011      01  MS  0  0  1
[ 6] .debug_line            PROGBITS       00000000      00811b      0000ac      00  0  0  0  1
[ 7] .debug_info            PROGBITS       00000000      0081c7      000103      00  0  0  0  1
[ 8] .debug_abbrev          PROGBITS       00000000      0082ca      0000bf      00  0  0  0  1
[ 9] .debug_aranges         PROGBITS       00000000      008390      000060      00  0  0  0  8
[10] .debug_loc             PROGBITS       00000000      0083f0      000058      00  0  0  0  1
[11] .debug_str             PROGBITS       00000000      008448      000069      01  MS  0  0  1
[12] .debug_frame           PROGBITS       00000000      0084b4      000054      00  0  0  0  4
[13] .shstrtab              STRTAB         00000000      008508      0000a1      00  0  0  0  1
[14] .symtab                SYMTAB         00000000      00882c      000210      10  15 28  4
[15] .strtab                STRTAB         00000000      008a3c      000057      00  0  0  0  1
```

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
D (mbind), y (purecode), p (processor specific)

-To start debugging we will open gdb circuit in qemu tool for board that we debug on called versatilepb using this command

```
ziade@LAPTOP-S0076FNK MINGW64 /d/Lab_1
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
```

-Then we run gdb

```
GNU gdb (GDB) 7.5.1
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-mingw32 --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\Lab_1\learn-in-depth.elf...done.
(gdb) |
```

-To target the server

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:3
3      ldr sp,= stack_top
(gdb) |
```

-Some commands

```
(gdb) b Uart_send_string
Breakpoint 3 at 0x10038: file uart.c, line 7.
(gdb) c
Continuing.

Breakpoint 1, reset () at startup.s:4
4      bl main
(gdb) display/3i $pc
1: x/3i $pc
=> 0x10004 <reset+4>:  bl      0x10010 <main>
   0x10008 <stop>:    b       0x10008 <stop>
   0x1000c <stop+4>:  ldrdeq  r1, [r1], -r12
(gdb) l
1      .globl reset
2      reset:
3          ldr sp,= stack_top
4          bl main
5      stop: b stop(gdb) c
Continuing.

Breakpoint 2, main () at app.c:6
6      Uart_send_string(string_buffer);
1: x/3i $pc
=> 0x10018 <main+8>:  ldr      r0, [pc, #4] ; 0x10024 <main+20>
   0x1001c <main+12>:  bl      0x10028 <Uart_send_string>
   0x10020 <main+16>:  pop     {r11, pc}
(gdb) l
1      #include "uart.h"
2      unsigned char string_buffer[100]= "learn-in-depth:<Ziad>";
3
4      void main(void)
5      {
6          Uart_send_string(string_buffer);
7      }(gdb) watch string_buffer
Hardware watchpoint 4: string_buffer
(gdb) watch string_buffer
Hardware watchpoint 5: string_buffer
(gdb) print string_buffer[0]
$1 = 108 'l'
(gdb) print string_buffer
$2 = "learn-in-depth:<Ziad>", '\000' <repeats 78 times>
(gdb) backtrace
#0 main () at app.c:6
(gdb) c
Continuing.

Breakpoint 3, Uart_send_string (
P_tx_string=0x10078 <string_buffer> "learn-in-depth:<Ziad>") at uart.c:7
7      while(*P_tx_string != '\0')
1: x/3i $pc
=> 0x10038 <Uart_send_string+16>:  b       0x10058 <Uart_send_string+48>
   0x1003c <Uart_send_string+20>:  ldr r3, [pc, #48] ; 0x10074 <Uart_send_string+76>
   0x10040 <Uart_send_string+24>:  ldr     r2, [r11, #-8]
(gdb) l
```

-We will step in C until uart.c and we will find that the string will printed character by character on the qemu terminal :

The image shows two terminal windows side-by-side. The left window displays assembly code for a file named 'uart.c'. The right window shows GDB output for the same file, including disassembly and register values.

```

MINGW64/d/Lab_1
[ 2] .text          PROGBITS      00010010 008010 000068 00 AX 0 0 4
[ 3] .data          PROGBITS      00010078 008078 000064 00 WA 0 0 4
[ 4] .ARM.attributes ARM_ATTRIBUTES 00000000 0080dc 00002e 00 0 0 1
[ 5] .comment       PROGBITS      00000000 00810a 000011 01 MS 0 0 1
[ 6] .debug_line    PROGBITS      00000000 00811b 0000ac 00 0 0 1
[ 7] .debug_info    PROGBITS      00000000 0081c7 000103 00 0 0 1
[ 8] .debug_abbrev  PROGBITS      00000000 0082ca 0000bf 00 0 0 1
[ 9] .debug_aranges PROGBITS      00000000 008390 000060 00 0 0 8
[10] .debug_loc     PROGBITS      00000000 0083f0 000058 00 0 0 1
[11] .debug_str     PROGBITS      00000000 008448 000069 01 MS 0 0 1
[12] .debug_frame  PROGBITS      00000000 0084b4 000054 00 0 0 4
[13] .shstrtab     STRTAB        00000000 008508 0000a1 00 0 0 1
[14] .symtab       SYMTAB        00000000 00882c 000210 10 15 28 4
[15] .strtab       STRTAB        00000000 008a3c 000057 00 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
D (mbind), y (purecode), p (processor specific)
ziade@LAPTOP-S0076FNK MINGW64 /d/Lab_1
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
learn-in-depth:<Ziad>
1: x/3i $pc
=> 0x1004c <uart_send_string+36>:   ldr    r3, [r11, #-8]
0x10050 <uart_send_string+40>:   add    r3, r3, #1
0x10054 <uart_send_string+44>:   str    r3, [r11, #-8]
(gdb) s
while(*P_tx_string != '\0')
1: x/3i $pc
=> 0x10058 <uart_send_string+48>:   ldr    r3, [r11, #-8]
0x1005c <uart_send_string+52>:   ldrb   r3, [r3]
0x10060 <uart_send_string+56>:   cmp    r3, #0
(gdb) s
UARTODR = (unsigned int)(*P_tx_string);
1: x/3i $pc
=> 0x1003c <uart_send_string+20>:   ldr    r3, [pc, #48] ; 0x10074 <uart_send_string+76>
0x10040 <uart_send_string+24>:   ldr    r2, [r11, #-8]
0x10044 <uart_send_string+28>:   ldrb   r2, [r2]
(gdb) s
P_tx_string++;
1: x/3i $pc
=> 0x1004c <uart_send_string+36>:   ldr    r3, [r11, #-8]
0x10050 <uart_send_string+40>:   add    r3, r3, #1
0x10054 <uart_send_string+44>:   str    r3, [r11, #-8]
(gdb)

```

• Makefile for lab 1:

```

1  #@ copyright : Eng.Ziad
2
3  CC=arm-none-eabi-
4  CFLAGS= -mcpu=arm926ej-s -g
5  INCS=-I .
6  LIBS=
7  SRC = $(wildcard *.c)
8  OBJ = $(SRC:.c=.o)
9  AS = $(wildcard *.s)
10 ASOBJ = $(AS:.s=.o)
11
12 project_name=learn-in-depth
13
14 all: $(project_name).bin
15     @echo "-----all build is done-----"
16
17 %.o: %.c
18     $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
19
20 %.o: %.s
21     $(CC)as.exe $(CFLAGS) $< -o $@
22
23 $(project_name).elf: $(OBJ) $(ASOBJ)
24     $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@
25
26 $(project_name).bin: $(project_name).elf
27     $(CC)objcopy.exe -O binary $< $@
28
29
30 clean_all:
31     rm *.o *.elf *.bin
32 clean:
33     rm *.bin *.elf
34
35

```

• Lab2:

- Board name : STM32f103c8t6 arm-cortex-m3 based .
- Flash starts with 0x08000000
- Sram starts with 0x20000000

1.Main .c

```
typedef volatile unsigned int vuint32_t;

#include <stdint.h>

#define RCC_BASE      0x40021000
#define PORTA_BASE    0x40010800
#define RCC_IO        (1<<2)
#define RCC_APB2ENR   *((volatile uint32_t *) (RCC_BASE + 0x18))
#define GPIOA_CRH     *((volatile uint32_t *) (PORTA_BASE + 0x04))
#define GPIOA_ODR     *((volatile uint32_t *) (PORTA_BASE + 0x0c))

typedef union {
    vuint32_t  all_fields;
    struct {
        vuint32_t  reserved:13 ;
        vuint32_t  p_13:1 ;
    }pin;
}R_ODR_t;

volatile R_ODR_t* R_ODR =(volatile R_ODR_t*)(PORTA_BASE + 0x0c);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};

volatile int i;
int main(void)
{

    RCC_APB2ENR |= RCC_IO ;    //to set bit 2
    GPIOA_CRH &= 0xff0fffff; //to protect the bits
    GPIOA_CRH |= 0x00200000; //to make bits(20>>24)=2

    while(1)
    {
        R_ODR->pin.p_13= 1 ;           //set bit 13
        for( i=0 ; i<5000 ; i++);      //random delay
        R_ODR->pin.p_13= 0;             //to clear bit 13
        for( i=0 ; i<5000 ; i++);

    }

    return 0;
}
```

2.Startup.s

```
1  /* startup_cortexM3.s
2  Eng.Ziad
3  */
4  /*SRAM 0x20000000 */
5
6  .section .vectors
7  .word 0x20001000      /* stack top address */
8  .word _reset         /* 1 Reset */
9  .word Vector_handler /* 2 NMI */
10 .word Vector_handler /* 3 Hard Fault */
11 .word Vector_handler /* 4 MM Fault */
12 .word Vector_handler /* 5 Bus Fault */
13 .word Vector_handler /* 6 Usage Fault */
14 .word Vector_handler /* 7 RESERVED */
15 .word Vector_handler /* 8 RESERVED */
16 .word Vector_handler /* 9 RESERVED */
17 .word Vector_handler /* 10 RESERVED */
18 .word Vector_handler /* 11 SV call */
19 .word Vector_handler /* 12 Debug reserved */
20 .word Vector_handler /* 13 RESERVED */
21 .word Vector_handler /* 14 PendSV */
22 .word Vector_handler /* 15 SysTrick */
23 .word Vector_handler /* 16 IRQ0 */
24 .word Vector_handler /* 17 IRQ1 */
25 .word Vector_handler /* 18 IRQ2 */
26 .word Vector_handler /* 19 ... */
27                          /* 0n to IRQ67*/
28
29 .section .text
30
31 _reset:
32     bl main
33     b .
34
35 .thumb_func
36
37 Vector_handler:
38     b _reset
```

3.Linker script

```
1  /* linker script CortexM3
2  Eng.Ziad
3  */
4
5  MEMORY
6  {
7      flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
8      sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20k
9  }
10
11
12 SECTIONS
13 {
14     .text : {
15         *(.Vectors*)
16         *(.text*)
17         *(.rodata)
18     }
19     }> flash
20     .data : {
21         *(.data)
22     }> flash
23
24     .bss : {
25         *(.bss)
26     }> sram
27
28
29 }
```

4.Make file

```
1  #@ copyright : Eng.Ziad
2
3  CC=arm-none-eabi-
4  CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
5  INCS=-I .
6  LIBS=
7  SRC = $(wildcard *.c)
8  OBJ = $(SRC:.c=.o)
9  AS = $(wildcard *.s)
10 ASOBJ = $(AS:.s=.o)
11
12 project_name=learn-in-depth_cortex_m3
13
14 ▼ all: $(project_name).bin
15     @echo "-----all build is done-----"
16
17 ▼ %.o: %.c
18     $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
19
20 ▼ %.o: %.s
21     $(CC)as.exe $(CFLAGS) $< -o $@
22
23 ▼ $(project_name).elf: $(OBJ) $(ASOBJ)
24     $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@
25
26 ▼ $(project_name).bin: $(project_name).elf
27     $(CC)objcopy.exe -O binary $< $@
28
29
30 ▼ clean_all:
31     rm *.o *.elf *.bin
32 ▼ clean:
33     rm *.bin *.elf
34
35
```

• Lab 2 with Startup.c

1.main.c

```
18  */
19  typedef volatile unsigned int vuint32_t;
20
21  #include <stdint.h>
22
23  #define RCC_BASE      0x40021000
24  #define PORTA_BASE    0x40010000
25  #define RCC_IO        (1<<2)
26  #define RCC_APB2ENR    *((volatile uint32_t *) (RCC_BASE + 0x18))
27  #define GPIOA_CRH      *((volatile uint32_t *) (PORTA_BASE + 0x04))
28  #define GPIOA_ODR      *((volatile uint32_t *) (PORTA_BASE + 0x0c))
29
30  extern void NMI_Handler(void)
31  {
32
33  }
34  extern void Bus_Fault(void)
35  {
36
37  }
38  typedef union {
39      vuint32_t  all_fields;
40      struct {
41          vuint32_t  reserved:13 ;
42          vuint32_t  p_13:1 ;
43      }pin;
44  }R_ODR_t;
45
46  volatile R_ODR_t* R_ODR =(volatile R_ODR_t*)(PORTA_BASE + 0x0c);
47  unsigned char g_variables[3]={1,2,3};
48  unsigned char const const_variables[3]={1,2,3};
49
50
51  volatile int i;
52  int main(void)
53  {
54
55
56      RCC_APB2ENR |= RCC_IO;    //to set bit 2
57      GPIOA_CRH &= 0xff0ffff; //to protect the bits
58      GPIOA_CRH |= 0x00200000; //to make bits(20>>24)=2
59
60      while(1)
61      {
62          R_ODR->pin.p_13= 1 ;    //set bit 13
63          for( i=0 ; i<5000 ; i++); //random delay
64          R_ODR->pin.p_13= 0;    //to clear bit 13
65          for( i=0 ; i<5000 ; i++);
66      }
67
68
69
```

2.startup.c

```
1 //startup.c
2 //Eng.Ziad
3
4 #include <stdint.h>
5 extern int main(void);
6
7 void Reset_Handler(void);
8 void Default_Handler();
9
10
11 //to make it easy to override & all have the same handler to minimize the size
12
13 void NMI_Handler() __attribute__((weak,alias("Default_Handler")));
14 void H_Fault_Handler() __attribute__((weak,alias("Default_Handler")));
15 void MM_Fault_Handler() __attribute__((weak,alias("Default_Handler")));
16 void Bus_Fault() __attribute__((weak,alias("Default_Handler")));
17 void Usage_Fault_Handler() __attribute__((weak,alias("Default_Handler")));
18
19
20
21
22
23 extern unsigned int _stack_top;
24 uint32_t vectors[] __attribute__((section(".Vectors"))) = {
25     (uint32_t) &_stack_top,
26     (uint32_t) &Reset_Handler,
27     (uint32_t) &NMI_Handler,
28     (uint32_t) &H_Fault_Handler,
29     (uint32_t) &MM_Fault_Handler,
30     (uint32_t) &Bus_Fault,
31     (uint32_t) &Usage_Fault_Handler
32 };
33 extern unsigned int _S_DATA;
34 extern unsigned int _E_DATA;
35 extern unsigned int _S_BSS;
36 extern unsigned int _E_BSS;
37 extern unsigned int _E_Text;
38
39 void Reset_Handler (void)
40 {
41     volatile int i;
42     //copy data from ROM to RAM
43     unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
44     unsigned char* P_src = (unsigned char*)&_E_Text;
45     unsigned char* P_dst = (unsigned char*)&_S_DATA;
46
47     for(i=0;i<DATA_size; i++)
48     {
49         *((unsigned char*)P_dst++)= *((unsigned char*)P_src++);
50     }
51
52     // initialize the .bss with zero
53     unsigned int bss_size = (unsigned char*)&_E_BSS - (unsigned char*)&_S_BSS;
54     P_dst = (unsigned char*)&_S_BSS;
55
```

```
56     for(i=0 ; i<bss_size ; i++)
57     {
58         *((unsigned char*)P_dst++)=(unsigned char)0;
59     }
60
61     // jump to main
62
63     main();
64 }
65 void Default_Handler()
66 {
67     Reset_Handler();
68 }
```


3. linker script

```
1  /* linker script CortexM3
2  Eng.Ziad
3  */
4
5  MEMORY
6  {
7      flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
8      sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20k
9
10 }
11
12 SECTIONS
13 {
14     .text : {
15         *(.Vectors*)
16         *(.text*)
17         *(.rodata)
18         _E_text = . ;
19     }
20     > flash
21     .data : {
22         _S_DATA = . ;
23         *(.data)
24         _E_DATA = . ;
25     }
26     > sram AT> flash
27     .bss : {
28         _S_bss = . ;
29         *(.bss)
30         . = ALIGN(4);
31         _E_bss = . ;
32         . = ALIGN(4);
33         . = . + 0x1000 ;
34         _stack_top = . ;
35     }
36     > sram
37
38 }
```

4. make file

```
1  #@ copyright : Eng.Ziad
2
3  CC=arm-none-eabi-
4  CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
5  INCS=-I .
6  LIBS=
7  SRC = $(wildcard *.c)
8  OBJ = $(SRC:.c=.o)
9  AS = $(wildcard *.s)
10 ASOBJ = $(AS:.s=.o)
11
12 project_name=learn-in-depth_cortex_m3
13
14 all: $(project_name).bin
15     @echo "-----all build is done-----"
16
17 %.o: %.c
18     $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
19
20 %.o: %.s
21     $(CC)as.exe $(CFLAGS) $< -o $@
22
23 $(project_name).elf: $(OBJ) $(ASOBJ)
24     $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@
25
26 $(project_name).bin: $(project_name).elf
27     $(CC)objcopy.exe -O binary $< $@
28
29
30
31 clean_all:
32     rm *.o *.elf *.bin
33 clean:
34     rm *.bin *.elf
35
36
37
```

- .data section has LMA within flash range and it will be copied

to sram so it has VMA within start of sram as we want

- .bss section has VMA within sram range .

```
ziade@LAPTOP-S0076FNK MINGW64 /d/lab2.c
$ arm-none-eabi-objdump.exe -h learn-in-depth_cortex_m3.elf

learn-in-depth_cortex_m3.elf:      file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text          000001e0  08000000  08000000  00008000  2**2
    CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data          00000008  20000000  080001e0  00010000  2**2
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00001004  20000008  080001e8  00010008  2**2
    ALLOC
  3 .debug_info     00000301  00000000  00000000  00010008  2**0
    CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev   000001c6  00000000  00000000  00010309  2**0
    CONTENTS, READONLY, DEBUGGING
  5 .debug_loc      000000e8  00000000  00000000  000104cf  2**0
    CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges  00000040  00000000  00000000  000105b7  2**0
    CONTENTS, READONLY, DEBUGGING
  7 .debug_line     0000014d  00000000  00000000  000105f7  2**0
    CONTENTS, READONLY, DEBUGGING
  8 .debug_str      0000015e  00000000  00000000  00010744  2**0
    CONTENTS, READONLY, DEBUGGING
  9 .comment        00000011  00000000  00000000  000108a2  2**0
    CONTENTS, READONLY
10 .ARM.attributes 00000033  00000000  00000000  000108b3  2**0
    CONTENTS, READONLY
11 .debug_frame     000000a4  00000000  00000000  000108e8  2**2
    CONTENTS, READONLY, DEBUGGING
```

- As we Mention NMI_Fault handler & Bus_handler in main.c(override on weak symbol in main.c)

So they didn't take the same address of Default_handler and they take different addresses

```
ziade@LAPTOP-S0076FNK MINGW64 /d/lab2.c
$ arm-none-eabi-nm.exe learn-in-depth_cortex_m3.elf
20000008 B _E_bss
20000008 D _E_DATA
080001e0 T _E_text
20000008 B _S_bss
20000000 D _S_DATA
20001008 B _stack_top
08000028 T Bus_Fault
080001dc T const_variables
080001d0 T Default_handler
20000004 D g_variables
080001d0 W H_fault_Handler
20001008 B i
08000034 T main
080001d0 W MM_Fault_Handler
0800001c T NMI_Handler
20000000 D R_ODR
0800011c T Reset_Handler
080001d0 W Usage_Fault_Handler
08000000 T vectors
```

-Lets see the map file

```
32  *(.rodata)          0x080001dc      0x4 main.o
33  .rodata             0x080001dc      const_variables
34                      0x080001e0      _E_text = .
35
36
37  .glue_7             0x080001e0      0x0
38  .glue_7             0x00000000      0x0 linker stubs
39
40  .glue_7t            0x080001e0      0x0
41  .glue_7t            0x00000000      0x0 linker stubs
42
43  .vfp11_veneer       0x080001e0      0x0
44  .vfp11_veneer       0x00000000      0x0 linker stubs
45
46  .v4_bx              0x080001e0      0x0
47  .v4_bx              0x00000000      0x0 linker stubs
48
49  .iplt              0x080001e0      0x0
50  .iplt              0x00000000      0x0 main.o
51
52  .rel.dyn            0x080001e0      0x0
53  .rel.iplt           0x00000000      0x0 main.o
54
55  .data              0x20000000      0x8 load address 0x080001e0
56                      0x20000000      _S_DATA = .
57
58  *(.data)
59  .data              0x20000000      0x8 main.o
60                      0x20000000      R_ODR
61                      0x20000004      g_variables
62  .data              0x20000008      0x0 startup.o
63                      0x20000008      _E_DATA = .
64
65  .igot.plt           0x20000008      0x0 load address 0x080001e8
66  .igot.plt           0x00000000      0x0 main.o
67
68  .bss               0x20000008      0x1004 load address 0x080001e8
69                      0x20000008      _S_bss = .
```

ALGIN

```
62                      0x20000008      _E_DATA = .
63
64  .igot.plt           0x20000008      0x0 load address 0x080001e8
65  .igot.plt           0x00000000      0x0 main.o
66
67  .bss               0x20000008      0x1004 load address 0x080001e8
68                      0x20000008      _S_bss = .
69
70  *(.bss)
71  .bss               0x20000008      0x0 main.o
72  .bss               0x20000008      0x0 startup.o
73                      0x20000008      . = ALIGN (0x4)
74                      0x20000008      _E_bss = .
75                      0x20000008      . = ALIGN (0x4)
76                      0x20000100      . = (. + 0x1000)
77  *fill*             0x20000008      0x1000
78                      0x20001008      _stack_top = .
79  COMMON              0x20001008      0x4 main.o
80                      0x20001008      i
```