思考题

Thinking 1.1

> 请阅读附录中的编译链接详解，尝试分别使用实验环境中的原生 x86 工具链（gcc、ld、readelf、objdump 等）和 MIPS 交叉编译工具链（带有 mips-linux-gnu-前缀），重复其中的编译和解析过程，观察相应的结果，并解释其中向 objdump 传入的参数的含义。

原生 x86 工具链

- main.c: 源文件

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

- main.i: `gcc -E main.c > main.i` 预处理结果（部分）

```
# 0 "main.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "main.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4

typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;


typedef signed char __int8_t;
typedef unsigned char __uint8_t;
typedef signed short int __int16_t;
typedef unsigned short int __uint16_t;
typedef signed int __int32_t;
typedef unsigned int __uint32_t;

typedef signed long int __int64_t;
typedef unsigned long int __uint64_t;

typedef __int8_t __int_least8_t;
typedef __uint8_t __uint_least8_t;
typedef __int16_t __int_least16_t;
typedef __uint16_t __uint_least16_t;
```

```
typedef __int32_t __int_least32_t;
typedef __uint32_t __uint_least32_t;
typedef __int64_t __int_least64_t;
typedef __uint64_t __uint_least64_t;


typedef long int __quad_t;
typedef unsigned long int __u_quad_t;

typedef long int __intmax_t;
typedef unsigned long int __uintmax_t;
# 141 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/typesizes.h" 1 3 4
# 142 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/time64.h" 1 3 4
# 143 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4


typedef unsigned long int __dev_t;
typedef unsigned int __uid_t;
typedef unsigned int __gid_t;
typedef unsigned long int __ino_t;
typedef unsigned long int __ino64_t;
typedef unsigned int __mode_t;
typedef unsigned long int __nlink_t;
typedef long int __off_t;
typedef long int __off64_t;
typedef int __pid_t;
typedef struct { int __val[2]; } __fsid_t;
typedef long int __clock_t;
typedef unsigned long int __rlim_t;
typedef unsigned long int __rlim64_t;
typedef unsigned int __id_t;
typedef long int __time_t;
typedef unsigned int __useconds_t;
typedef long int __suseconds_t;
typedef long int __suseconds64_t;

typedef int __daddr_t;
typedef int __key_t;


typedef int __clockid_t;


typedef void * __timer_t;

struct _IO_FILE;
typedef struct _IO_FILE __FILE;
# 42 "/usr/include/stdio.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/types/FILE.h" 1 3 4


# 2 "main.c" 2
```

```
# 3 "main.c"
int main()
{
    printf("Hello World!\n");
    return 0;
}
```

预处理将头文件内容加入源文件，但没有具体的函数实现。

- main.s: `gcc -S main.c` 预处理并编译结果

```
        .file   "main.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
```

```
        .align 8
4:
```

编译将C语言编译为汇编代码。

- main_obj: 预处理、编译并汇编后反汇编结果

```
main.o:     文件格式 elf32-tradbigmips


Disassembly of section .text:

00000000 <main>:
   0:   27bdffe0        addiu   sp,sp,-32
   4:   afbf001c        sw      ra,28(sp)
   8:   afbe0018        sw      s8,24(sp)
   c:   03a0f025        move    s8,sp
  10:   3c1c0000        lui     gp,0x0
  14:   279c0000        addiu   gp,gp,0
  18:   afbc0010        sw      gp,16(sp)
  1c:   3c020000        lui     v0,0x0
  20:   24440000        addiu   a0,v0,0
  24:   8f820000        lw      v0,0(gp)
  28:   0040c825        move    t9,v0
  2c:   0320f809        jalr    t9
  30:   00000000        nop
  34:   8fdc0010        lw      gp,16(s8)
  38:   00001025        move    v0,zero
  3c:   03c0e825        move    sp,s8
  40:   8fbf001c        lw      ra,28(sp)
  44:   8fbe0018        lw      s8,24(sp)
  48:   27bd0020        addiu   sp,sp,32
  4c:   03e00008        jr      ra
  50:   00000000        nop
        ...

Disassembly of section .reginfo:

00000000 <.reginfo>:
   0:   f2000014        0xf2000014
        ...

Disassembly of section .MIPS.abiflags:

00000000 <.MIPS.abiflags>:
   0:   00002002        srl     a0,zero,0x0
   4:   01010005        lsa     zero,t0,at,0x1
        ...

Disassembly of section .pdr:

00000000 <.pdr>:
   0:   00000000        nop
   4:   c0000000        ll      zero,0(zero)
   8:   fffffffc        0xfffffffc
```

```
        ...
  14:   00000020        add     zero,zero,zero
  18:   0000001e        0x1e
  1c:   0000001f        0x1f

Disassembly of section .rodata:

00000000 <.rodata>:
   0:   48656c6c        mfhc2   a1,0x6c6c
   4:   6f20576f        0x6f20576f
   8:   726c6421        0x726c6421
   c:   00000000        nop

Disassembly of section .comment:

00000000 <.comment>:
   0:   00474343        0x474343
   4:   3a202855        xori    zero,s1,0x2855
   8:   62756e74        0x62756e74
   c:   75203130        jalx    480c4c0 <main+0x480c4c0>
  10:   2e332e30        sltiu   s3,s1,11824
  14:   2d317562        sltiu   s1,t1,30050
  18:   756e7475        jalx    5b9d1d4 <main+0x5b9d1d4>
  1c:   31292031        andi    t1,t1,0x2031
  20:   302e332e        andi    t6,at,0x332e
  24:   地址 0x0000000000000024 越界。


Disassembly of section .gnu.attributes:

00000000 <.gnu.attributes>:
   0:   41000000        mftc0   zero,c0_index
   4:   0f676e75        jal     d9db9d4 <main+0xd9db9d4>
   8:   00010000        sll     zero,at,0x0
   c:   00070405        0x70405
```

注意此时和x86指令集不同，但函数调用部分的地址为0，因为此时还未链接。

- main: `gcc main.c -o main` 正常编译结果（部分）

```
main:     文件格式 elf64-x86-64


Disassembly of section .interp:

0000000000001140 <frame_dummy>:
    1140:       f3 0f 1e fa             endbr64
    1144:       e9 77 ff ff ff          jmp    10c0 <register_tm_clones>

0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 8d 05 ac 0e 00 00    lea    0xeac(%rip),%rax        # 2004
<_IO_stdin_used+0x4>
```

```
    1158:       48 89 c7                mov    %rax,%rdi
    115b:       e8 f0 fe ff ff          call   1050 <puts@plt>
    1160:       b8 00 00 00 00          mov    $0x0,%eax
    1165:       5d                      pop    %rbp
    1166:       c3                      ret

Disassembly of section .fini:

0000000000001168 <_fini>:
    1168:       f3 0f 1e fa             endbr64
    116c:       48 83 ec 08             sub    $0x8,%rsp
    1170:       48 83 c4 08             add    $0x8,%rsp
    1174:       c3                      ret
```

此时函数调用 `callq` 后被替换为一个地址，即实现了将 `printf` 库函数链接到目标文件。

MIPS 交叉编译工具链

- main.c: 源文件

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

- main.i: `mips-linux-gnu-gcc -E main.c > main.i` 预处理结果（部分）

```
extern char *ctermid (char *__s) __attribute__ ((__nothrow__ , __leaf__))
  __attribute__ ((__access__ (__write_only__, 1)));
# 867 "/usr/mips-linux-gnu/include/stdio.h" 3
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__));



extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__))
;


extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ ,
__leaf__));
# 885 "/usr/mips-linux-gnu/include/stdio.h" 3
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 902 "/usr/mips-linux-gnu/include/stdio.h" 3

# 2 "main.c" 2


# 3 "main.c"
int main()
{
    printf("Hello World!\n");
```

```
    return 0;
}
```

预处理将头文件内容加入源文件，但没有具体的函数实现。

- main.s: `mips-linux-gnu-gcc -S main.c` 预处理并编译结果

```
        .file   1 "main.c"
        .section .mdebug.abi32
        .previous
        .nan    legacy
        .module fp=xx
        .module nooddspreg
        .abicalls
        .text
        .rdata
        .align  2
$LC0:
        .ascii  "Hello World!\000"
        .text
        .align  2
        .globl  main
        .set    nomips16
        .set    nomicromips
        .ent    main
        .type   main, @function
main:
        .frame  $fp,32,$31              # vars= 0, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        addiu   $sp,$sp,-32
        sw      $31,28($sp)
        sw      $fp,24($sp)
        move    $fp,$sp
        lui     $28,%hi(__gnu_local_gp)
        addiu   $28,$28,%lo(__gnu_local_gp)
        .cprestore      16
        lui     $2,%hi($LC0)
        addiu   $4,$2,%lo($LC0)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $2,$0
        move    $sp,$fp
        lw      $31,28($sp)
        lw      $fp,24($sp)
        addiu   $sp,$sp,32
        jr      $31
        nop
```

```
        .set    macro
        .set    reorder
        .end    main
        .size   main, .-main
        .ident  "GCC: (Ubuntu 10.3.0-1ubuntu1) 10.3.0"
        .section        .note.GNU-stack,"",@progbits
```

编译将C语言编译为汇编代码。

- main_obj: 预处理、编译并汇编后反汇编结果

```
main.o:     文件格式 elf32-tradbigmips


Disassembly of section .text:

00000000 <main>:
   0:   27bdffe0        addiu   sp,sp,-32
   4:   afbf001c        sw      ra,28(sp)
   8:   afbe0018        sw      s8,24(sp)
   c:   03a0f025        move    s8,sp
  10:   3c1c0000        lui     gp,0x0
  14:   279c0000        addiu   gp,gp,0
  18:   afbc0010        sw      gp,16(sp)
  1c:   3c020000        lui     v0,0x0
  20:   24440000        addiu   a0,v0,0
  24:   8f820000        lw      v0,0(gp)
  28:   0040c825        move    t9,v0
  2c:   0320f809        jalr    t9
  30:   00000000        nop
  34:   8fdc0010        lw      gp,16(s8)
  38:   00001025        move    v0,zero
  3c:   03c0e825        move    sp,s8
  40:   8fbf001c        lw      ra,28(sp)
  44:   8fbe0018        lw      s8,24(sp)
  48:   27bd0020        addiu   sp,sp,32
  4c:   03e00008        jr      ra
  50:   00000000        nop
        ...

Disassembly of section .reginfo:

00000000 <.reginfo>:
   0:   f2000014        0xf2000014
        ...

Disassembly of section .MIPS.abiflags:

00000000 <.MIPS.abiflags>:
   0:   00002002        srl     a0,zero,0x0
   4:   01010005        lsa     zero,t0,at,0x1
        ...

Disassembly of section .pdr:
```

```
00000000 <.pdr>:
   0:   00000000        nop
   4:   c0000000        ll      zero,0(zero)
   8:   fffffffc        0xfffffffc
        ...
  14:   00000020        add     zero,zero,zero
  18:   0000001e        0x1e
  1c:   0000001f        0x1f

Disassembly of section .rodata:

00000000 <.rodata>:
   0:   48656c6c        mfhc2   a1,0x6c6c
   4:   6f20576f        0x6f20576f
   8:   726c6421        0x726c6421
   c:   00000000        nop

Disassembly of section .comment:

00000000 <.comment>:
   0:   00474343        0x474343
   4:   3a202855        xori    zero,s1,0x2855
   8:   62756e74        0x62756e74
   c:   75203130        jalx    480c4c0 <main+0x480c4c0>
  10:   2e332e30        sltiu   s3,s1,11824
  14:   2d317562        sltiu   s1,t1,30050
  18:   756e7475        jalx    5b9d1d4 <main+0x5b9d1d4>
  1c:   31292031        andi    t1,t1,0x2031
  20:   302e332e        andi    t6,at,0x332e
  24:   地址 0x0000000000000024 越界。


Disassembly of section .gnu.attributes:

00000000 <.gnu.attributes>:
   0:   41000000        mftc0   zero,c0_index
   4:   0f676e75        jal     d9db9d4 <main+0xd9db9d4>
   8:   00010000        sll     zero,at,0x0
   c:   00070405        0x70405
```

此时函数调用部分此时还未链接。

- main: `gcc main.c -o main` 正常编译结果（部分）

```
main:     文件格式 elf32-tradbigmips


Disassembly of section .interp:

00400194 <.interp>:
  400194:       2f6c6962        sltiu   t4,k1,26978
  400198:       2f6c642e        sltiu   t4,k1,25646
  40019c:       736f2e31        0x736f2e31
        ...
```

```
Disassembly of section .MIPS.abiflags:

004001a8 <.MIPS.abiflags>:
  4001a8:       00002002        srl     a0,zero,0x0
  4001ac:       01010005        lsa     zero,t0,at,0x1
        ...

Disassembly of section .reginfo:

004001c0 <.reginfo>:
  4001c0:       b20000f6        0xb20000f6
        ...
  4001d4:       00419010        0x419010

Disassembly of section .note.gnu.build-id:

004001d8 <.note.gnu.build-id>:
  4001d8:       00000004        sllv    zero,zero,zero
  4001dc:       00000014        0x14
  4001e0:       00000003        sra     zero,zero,0x0
  4001e4:       474e5500        bz.w    $w14,4155e8 <_end+0x4588>
  4001e8:       bf37a85b        cache   0x17,-22437(t9)
  4001ec:       31414459        andi    at,t2,0x4459
  4001f0:       526b78ce        beql    s3,t3,41e52c <_gp+0x551c>
  4001f4:       f89f658d        sdc2    $31,25997(a0)
  4001f8:       2122f1c6        addi    v0,t1,-3642

04006e0 <main>:
  4006e0:       27bdffe0        addiu   sp,sp,-32
  4006e4:       afbf001c        sw      ra,28(sp)
  4006e8:       afbe0018        sw      s8,24(sp)
  4006ec:       03a0f025        move    s8,sp
  4006f0:       3c1c0042        lui     gp,0x42
  4006f4:       279c9010        addiu   gp,gp,-28656
  4006f8:       afbc0010        sw      gp,16(sp)
  4006fc:       3c020040        lui     v0,0x40
  400700:       24440830        addiu   a0,v0,2096
  400704:       8f828030        lw      v0,-32720(gp)
  400708:       0040c825        move    t9,v0
  40070c:       0320f809        jalr    t9
  400710:       00000000        nop
  400714:       8fdc0010        lw      gp,16(s8)
  400718:       00001025        move    v0,zero
  40071c:       03c0e825        move    sp,s8
  400720:       8fbf001c        lw      ra,28(sp)
  400724:       8fbe0018        lw      s8,24(sp)
  400728:       27bd0020        addiu   sp,sp,32
  40072c:       03e00008        jr      ra
  400730:       00000000        nop
```

此时函数调用后被替换为一个地址，即实现了将 `printf` 库函数链接到目标文件。

objdump参数含义

```
用法：objdump <选项> <文件>
 显示来自目标 <文件> 的信息。
 至少必须给出以下选项之一：
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file header
  -p, --private-headers    Display object format specific file header contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
  -d, --disassemble        Display assembler contents of executable sections
  -D, --disassemble-all    Display assembler contents of all sections
      --disassemble=<sym>  Display assembler contents from <sym>
  -S, --source             Intermix source code with disassembly
      --source-comment[=<txt>] Prefix lines of source code with <txt>
  -s, --full-contents      Display the full contents of all sections requested
  -g, --debugging          Display debug information in object file
  -e, --debugging-tags     Display debug information using ctags style
  -G, --stabs              Display (in raw form) any STABS info in the file
  -W, --dwarf[a/=abbrev, A/=addr, r/=aranges, c/=cu_index, L/=decodedline,
              f/=frames, F/=frames-interp, g/=gdb_index, i/=info, o/=loc,
              m/=macro, p/=pubnames, t/=pubtypes, R/=Ranges, l/=rawline,
              s/=str, O/=str-offsets, u/=trace_abbrev, T/=trace_aranges,
              U/=trace_info]
                           Display the contents of DWARF debug sections
  -Wk,--dwarf=links        Display the contents of sections that link to
                            separate debuginfo files
  -WK,--dwarf=follow-links
                           Follow links to separate debug info files (default)
  -WN,--dwarf=no-follow-links
                           Do not follow links to separate debug info files
  -L, --process-links      Display the contents of non-debug sections in
                            separate debuginfo files.  (Implies -WK)
      --ctf[=SECTION]      Display CTF info from SECTION, (default `.ctf')
  -t, --syms               Display the contents of the symbol table(s)
  -T, --dynamic-syms       Display the contents of the dynamic symbol table
  -r, --reloc              Display the relocation entries in the file
  -R, --dynamic-reloc      Display the dynamic relocation entries in the file
 @<file>                   Read options from <file>
  -v, --version            Display this program's version number
  -i, --info               List object formats and architectures supported
  -H, --help               Display this information

 以下选项是可选的：
  -b, --target=BFDNAME           Specify the target object format as BFDNAME
  -m, --architecture=MACHINE     Specify the target architecture as MACHINE
  -j, --section=NAME             Only display information for section NAME
  -M, --disassembler-options=OPT Pass text OPT on to the disassembler
  -EB --endian=big               Assume big endian format when disassembling
  -EL --endian=little            Assume little endian format when disassembling
      --file-start-context       Include context from start of file (with -S)
  -I, --include=DIR              Add DIR to search list for source files
  -l, --line-numbers             Include line numbers and filenames in output
```

```
  -F, --file-offsets              Include file offsets when displaying
information
  -C, --demangle[=STYLE]          Decode mangled/processed symbol names
                                    STYLE can be "none", "auto", "gnu-v3",
                                    "java", "gnat", "dlang", "rust"
      --recurse-limit             Enable a limit on recursion whilst demangling
                                   (default)
      --no-recurse-limit          Disable a limit on recursion whilst demangling
  -w, --wide                      Format output for more than 80 columns
  -U[d|l|i|x|e|h]                 Controls the display of UTF-8 unicode
characters
  --unicode=[default|locale|invalid|hex|escape|highlight]
  -z, --disassemble-zeroes        Do not skip blocks of zeroes when disassembling
      --start-address=ADDR        Only process data whose address is >= ADDR
      --stop-address=ADDR         Only process data whose address is < ADDR
      --no-addresses              Do not print address alongside disassembly
      --prefix-addresses          Print complete address alongside disassembly
      --[no-]show-raw-insn        Display hex alongside symbolic disassembly
      --insn-width=WIDTH          Display WIDTH bytes on a single line for -d
      --adjust-vma=OFFSET         Add OFFSET to all displayed section addresses
      --special-syms              Include special symbols in symbol dumps
      --inlines                   Print all inlines for source line (with -l)
      --prefix=PREFIX             Add PREFIX to absolute paths for -S
      --prefix-strip=LEVEL        Strip initial directory names for -S
      --dwarf-depth=N             Do not display DIEs at depth N or greater
      --dwarf-start=N             Display DIEs starting at offset N
      --dwarf-check               Make additional dwarf consistency checks.
      --ctf-parent=NAME           Use CTF archive member NAME as the CTF parent
      --visualize-jumps           Visualize jumps by drawing ASCII art lines
      --visualize-jumps=color     Use colors in the ASCII art
      --visualize-jumps=extended-color
                                  Use extended 8-bit color codes
      --visualize-jumps=off       Disable jump visualization

objdump：支持的目标： elf64-x86-64 elf32-i386 elf32-iamcu elf32-x86-64 pei-i386 pe-
x86-64 pei-x86-64 elf64-l1om elf64-k1om elf64-little elf64-big elf32-little
elf32-big pe-bigobj-x86-64 pe-i386 srec symbolsrec verilog tekhex binary ihex
plugin
objdump：支持的体系结构： i386 i386:x86-64 i386:x64-32 i8086 i386:intel i386:x86-
64:intel i386:x64-32:intel iamcu iamcu:intel l1om l1om:intel k1om k1om:intel
```

其中`-D`表示显示所有节的汇编程序内容，`-S`表示显示混合源代码和反汇编。

Thinking 1.2

> 思考下述问题：
>
> • 尝试使用我们编写的 readelf 程序，解析之前在 target 目录下生成的内核 ELF 文件。
>
> • 也许你会发现我们编写的 readelf 程序是不能解析 readelf 文件本身的，而我们刚才介绍的系统工具 readelf 则可以解析，这是为什么呢？（提示：尝试使用 readelf -h，并阅读 tools/readelf 目录下的 Makefile，观察 readelf 与 hello 的不同）

- 解析内核ELF文件：

```
git@21371477:~/21371477/tools/readelf (lab1)$ ./readelf ../../target/mos
```

```
0:0x0
1:0x80010000
2:0x80011cd0
3:0x80011ce8
4:0x80011d00
5:0x0
6:0x0
7:0x0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
```

- 解析hello:

```
git@21371477:~/21371477/tools/readelf (lab1)$ readelf -h ./hello
ELF 头：
  Magic：   7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
  类别:                              ELF32
  数据:                              2 补码，小端序 (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI 版本:                          0
  类型:                              EXEC (可执行文件)
  系统架构:                          Intel 80386
  版本:                              0x1
  入口点地址：               0x8049600
  程序头起点：          52 (bytes into file)
  Start of section headers:          746252 (bytes into file)
  标志：             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         35
  Section header string table index: 34
```

- 解析readelf:

```
git@21371477:~/21371477/tools/readelf (lab1)$ readelf -h ./readelf
ELF 头：
  Magic：   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  类别:                              ELF64
  数据:                              2 补码，小端序 (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI 版本:                          0
  类型:                              DYN (Position-Independent Executable file)
```

```
系统架构:                          Advanced Micro Devices X86-64
版本:                               0x1
入口点地址:                0x1180
程序头起点:           64 (bytes into file)
Start of section headers:          14488 (bytes into file)
标志:              0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

可以发现由于自己编写的readelf只能处理32位ELF文件，而readelf在编译时未指定目标文件位数，默认生成与系统架构匹配的代码，即64位，因此无法解析。而系统自带的readelf工具有识别位数并使用相应方式处理的能力。

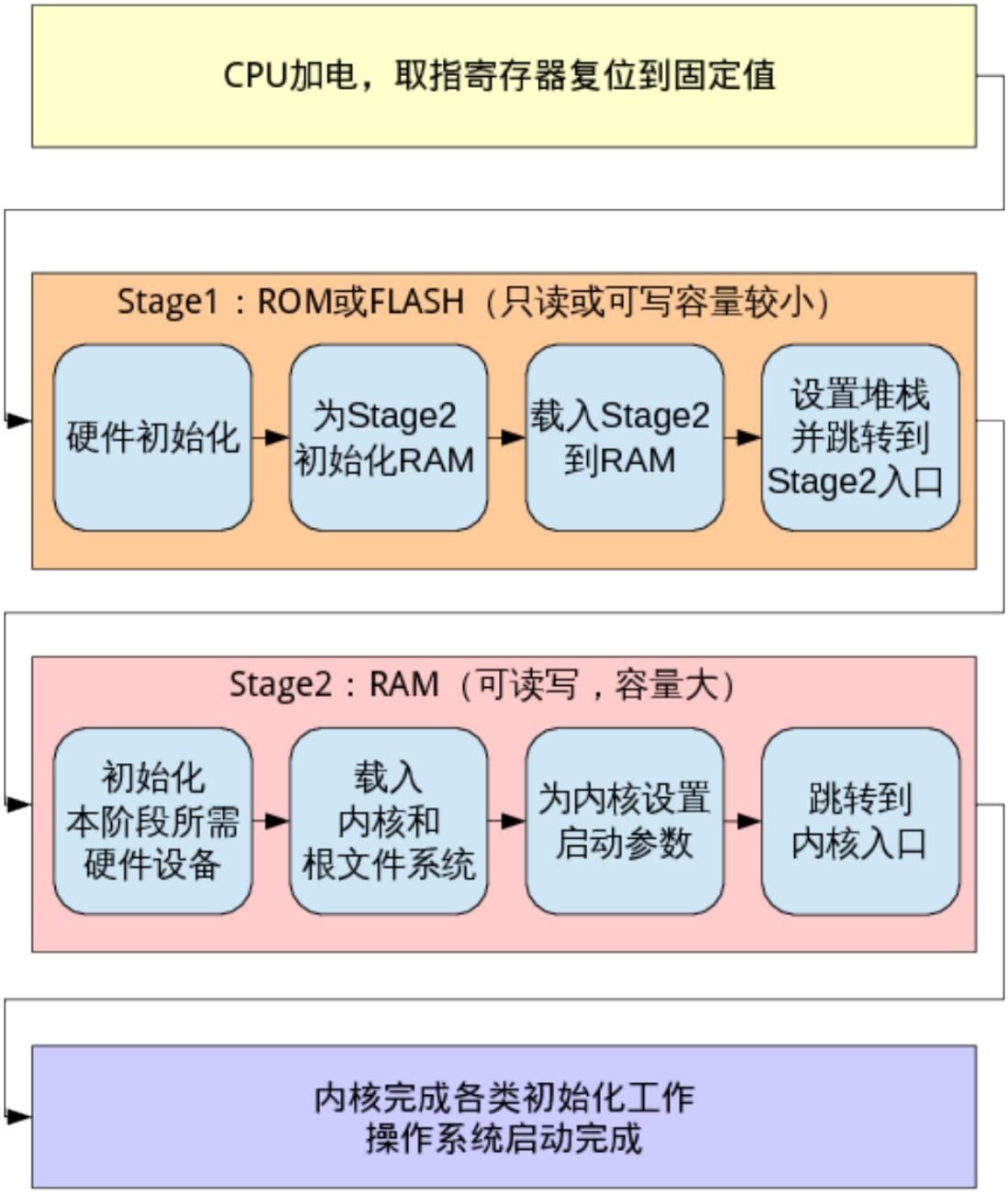
Thinking 1.3

> 在理论课上我们了解到，MIPS 体系结构上电时，启动入口地址为 0xBFC00000（其实启动入口地址是根据具体型号而定的，由硬件逻辑确定，也有可能不是这个地址，但一定是一个确定的地址），但实验操作系统的内核入口并没有放在上电启动地址，而是按照内存布局图放置。思考为什么这样放置内核还能保证内核入口被正确跳转到？（提示：思考实验中启动过程的两阶段分别由谁执行。）

- CPU上电后，首先会运行bootloader，大多数bootloader分为stage1和stage2两个部分。其中，stage1负责硬件的初始化，并且为stage2做好准备条件，载入stage2到RAM，设置堆栈并跳转到stage2入口。stage2首先初始化本阶段所需的硬件设备，载入内核和根文件系统到RAM，为内核设置启动参数，随后将跳转到内核入口。

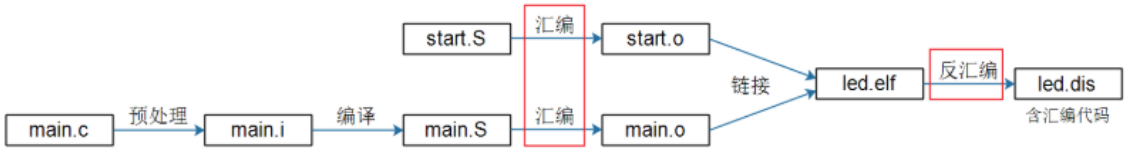
难点分析

- 操作系统启动



不同系统具体过程可能存在差异。

- GCC编译过程



头文件：只存储变量、函数或者类等这些功能模块的声明部分（无函数具体实现）（预处理环节）

> 指定头文件目录
>
> -I <头文件目录>

库函数：存储各模块具体的实现部分（一般以.o形式存储）

> 使用库函数首先需要通过引入包含该函数声明的头文件（即#include），这样可以隐藏具体的函数实现且不影响使用。

> 指定库函数（目录）
>
> -L <库文件目录>
>
> -l<库文件> // 中间无空格

链接：将**库文件**链接到目标文件，由**链接器**ld完成。

静态链接：可独立运行，移植性强。（当多次调用相同模块时，会导致代码冗余）可执行文件体积较大。静态链接库（.a）即多个简单目标文件（.o）的集合（使用`ar`命令打包）。

> .o文件间的链接也是静态链接。

动态链接：可执行文件体积小（只在可执行文件中记录功能模块的地址，然后通过跳转`call`到库函数文件）。无法独立运行，可移植性差。动态链接库（.so）也是由多个简单目标文件得到。

> 包含动态链接库的结果文件（.out）通常无法直接执行，通过执行`ldd main`指令，可以查看当前文件在执行时需要用到的所有动态链接库，以及各个库文件的存储位置。
>
> 解决方法：
>
> - 将链接库移动到标准库目录下。
> - 通过在终端或`~/.bashrc`中使用`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:xxx`命令。

PS：GCC默认使用动态链接，如果找不到才会使用静态链接（库）。

GCC参数：

参数	描述
-E	只进行预处理，需要重定向输出到文件
-S	只进行预处理和编译
-c	只进行预处理、编译和汇编
-o	指定输出文件名
-static	只使用静态链接
-Wall	生成所有警告信息

实验体会

- 对于操作系统的启动过程较难理解，需要结合计算机组成的相关知识。
- 编译链接的过程比较复杂，需要阅读一些课外资料。