



南开大学  
Nankai University

南开大学

计算机学院和网络空间安全学院

## 编译系统原理实验报告

---

### 了解编译器及 LLVM IR 编程

---

朱子轩 2110853 计算机科学与技术

刘修铭 2112492 信息安全

年级：2021 级

指导教师：王刚

2023 年 9 月 12 日

## 摘要

本次实验由朱子轩与刘修铭合作完成，主要通过改进给定的阶乘 C 语言程序代码，在 Ubuntu 虚拟机上对编译器的各个阶段及其功能进行了学习与探索。除了完成实验手册的要求内容外，我们还通过加入注释、引用头文件、宏定义、死代码等部分对编译的预处理部分功能进行验证性探索，加深对于词法分析与语法分析的过程的认识；同时，借助 VS code 的插件，完成了将 CFG 的可视化，进而实现对中间代码生成的多阶段的分析；代码优化部分，我们通过对 O0 O1 O2 等不同优化等级的对比，以程序的运行时间为参考，验证优化效果。除此之外，我们还对 X86、ARM 和 LLVM 的汇编和链接结果分别予以实现，并加以探索。最后，我们学习了 LLVM IR 的语法规则，并编写了一段 LLVM IR 程序，加深对 LLVM IR 中间语言的了解。

**关键字：**预处理器，编译器，汇编器，链接器，加载器，优化对比，交叉编译，性能测试，LLVM IR

## 目录

一、 分工	1
二、 预备工作及实验平台	2
三、 实验过程	4
(一) 预处理器 . . . . .	4
1. 预处理阶段功能 . . . . .	4
2. 实验验证 . . . . .	4
(二) 编译器 . . . . .	5
1. 词法分析 . . . . .	5
2. 语法分析 . . . . .	8
3. 语义分析和中间代码生成 . . . . .	9
4. 代码优化 . . . . .	13
5. 代码生成 . . . . .	14
(三) 汇编器 . . . . .	14
(四) 链接器加载器 . . . . .	16
(五) 执行 . . . . .	22
(六) LLVM IR 编程 . . . . .	22
四、 总结	25

## 一、 分工

小组二人均独立完成所给阶段 C 语言部分的编译复现工作，共同完成 LLVM IR 部分编写工作，并由刘修铭撰写实验报告，朱子轩予以修改补充。

实验所用代码及生成文件均打包附后。

NIKU

## 二、 预备工作及实验平台

按照实验指导，我们选用 Ubuntu 虚拟机配置了实验环境。实验平台如下：

设备名称	lxmliu2002-Ubuntu
系统名称	Ubuntu 22.04.3 LTS
操作系统类型	Linux 64 位
GNOME 版本	42.9
窗口系统	X11
虚拟化	VMware

表 1: 实验平台参数

同时，为了完成后续探索工作，我们修改了给定的阶乘程序。修改后的代码如下：

阶乘

```
1 // 引用头文件
2 #include <stdio.h>
3 #include <time.h>
4 // 宏定义
5 #define MAX 10000
6 int main()
7 {
8     int i, n, f;
9     scanf("%d", &n);
10    i = 2;
11    f = 1;
12    clock_t start, end;
13    double time;
14    start = clock();
15    // 死代码
16    if (i < 1)
17    {
18        printf("i 小于 1\n");
19    }
20    while (i <= n)
21    {
22        f = f * i;
23        i = i + 1;
24    }
25    if (f > MAX)
26    {
27        printf("输出超限\n");
28    }
29    else
30    {
31        printf("%d\n", f);
32    }
```

```
33     end = clock();  
34     time = (double)(end - start) / CLOCKS_PER_SEC;  
35     printf("程序运行时间为: %f\n", time);  
36     return 0;  
37 }
```

NIKU

## 三、 实验过程

借助 `clang -ccc-print-phases main.c` 指令，我们得到编译的整个流程如图1所示：

```
lxmliu2002@lxmliu2002-Ubuntu:~/桌面/lab1$ clang -ccc-print-phases main.c
+- 0: input, "main.c", c
+- 1: preprocessor, {0}, cpp-output
+- 2: compiler, {1}, ir
+- 3: backend, {2}, assembler
+- 4: assembler, {3}, object
5: linker, {4}, image
```

图 1: 编译各阶段

### (一) 预处理器

#### 1. 预处理阶段功能

编译器的预处理阶段是编译过程的第一个阶段，它在实际的编译之前执行，主要完成以下任务：

- 处理和预处理以 `#` 符号开头指令。
- 展开宏定义，将宏名称替换为其定义的文本。
- 根据条件编译指令（如 `#ifdef`、`#ifndef`、`#if`、`#elif` 等）来控制哪些部分的代码应该包含在编译中。
- 移除注释，删除多余的空格、制表符和换行符，以简化源代码。
- 将指定的头文件内容插入到源代码中，从而形成一个整体的源代码文件，以便后续编译阶段使用。

预处理阶段的主要目标是将源代码准备好，以便后续的编译阶段能够进行词法分析、语法分析和生成中间代码。完成预处理后，生成的预处理文件通常会保存为 `.i` 或 `.ii` 扩展名的文件，然后传递给编译器的下一个阶段进行编译。

#### 2. 实验验证

为了验证以上功能，我们将给定的阶乘代码予以改进，实现了对上述功能的验证。

首先，执行 `gcc main.c -E -o main.i` 指令，得到预处理后的 `main.i` 文件。然后观察生成的 `main.i` 文件并将其与 `main.c` 文件进行对比，可以发现：

- 代码的长度增加，由 37 行变为 1067 行，观察可知，引用的头文件已经插入其中。
- 一些无关内容被预处理删除。
- 宏名称已经被替换。

```

1027 # 4 "main.c" 2
1028
1029
1030
1031 # 6 "main.c"
1032 int main()
1033 {
1034     int i, n, f;
1035     scanf("%d", &n);
1036     i = 2;
1037     f = 1;
1038     clock_t start, end;
1039     double time;
1040     start = clock();
1041
1042     if (i < 1)
1043     {
1044         printf("i小于1\n");
1045     }
1046     while (i <= n)
1047     {
1048         f = f * i;
1049         i = i + 1;
1050     }
1051     if (f > 10000)
1052     {
1053         printf("输出超限\n");
1054     }
1055     else
1056     {
1057         printf("%d\n", f);
1058     }
1059     end = clock();
1060     time = (double)(end - start) /
1061 # 34 "main.c" 3 4
1062     ((__clock_t) 1000000);
1063 # 34 "main.c"
1064     printf("程序运行时间为: %f\n", time);
1065     return 0;
1066 }

```

图 2: 预处理结果

根据以上实验结果可知，预处理功能得以验证。

## (二) 编译器

### 1. 词法分析

词法分析阶段的主要目标是将源代码分割成词法单元，以便后续的语法分析阶段能够理解代码的结构和语法。

借助 `clang -E -Xclang -dump-tokens main.c` 命令，我们得到 token 序列：

词法分析结果

1	int 'int '	[StartOfLine]	Loc=<main.c:6:1>
2	identifier 'main'	[LeadingSpace]	Loc=<main.c:6:5>
3	l_paren '('		Loc=<main.c:6:9>
4	r_paren ')'		Loc=<main.c:6:10>
5	l_brace '{'	[StartOfLine]	Loc=<main.c:7:1>
6	int 'int '	[StartOfLine] [LeadingSpace]	Loc=<main.c:8:5>
7	identifier 'i'	[LeadingSpace]	Loc=<main.c:8:9>
8	comma ', '		Loc=<main.c:8:10>
9	identifier 'n'	[LeadingSpace]	Loc=<main.c:8:12>
10	comma ', '		Loc=<main.c:8:13>
11	identifier 'f'	[LeadingSpace]	Loc=<main.c:8:15>
12	semi ';'		Loc=<main.c:8:16>
13	identifier 'scanf'	[StartOfLine] [LeadingSpace]	Loc=<main.c:9:5>
14	l_paren '('		Loc=<main.c:9:10>
15	string_literal '""%d""'		Loc=<main.c:9:11>
16	comma ', '		Loc=<main.c:9:15>
17	amp '&'	[LeadingSpace]	Loc=<main.c:9:17>



```

18 identifier 'n'          Loc=<main.c:9:18>
19 r_paren ')'          Loc=<main.c:9:19>
20 semi ';'            Loc=<main.c:9:20>
21 identifier 'i'      [StartOfLine] [LeadingSpace]  Loc=<main.c:10:5>
22 equal '='          [LeadingSpace] Loc=<main.c:10:7>
23 numeric_constant '2' [LeadingSpace] Loc=<main.c:10:9>
24 semi ';'            Loc=<main.c:10:10>
25 identifier 'f'      [StartOfLine] [LeadingSpace]  Loc=<main.c:11:5>
26 equal '='          [LeadingSpace] Loc=<main.c:11:7>
27 numeric_constant '1' [LeadingSpace] Loc=<main.c:11:9>
28 semi ';'            Loc=<main.c:11:10>
29 identifier 'clock_t' [StartOfLine] [LeadingSpace]  Loc=<main.c:12:5>
30 identifier 'start'   [LeadingSpace] Loc=<main.c:12:13>
31 comma ','           Loc=<main.c:12:18>
32 identifier 'end'     [LeadingSpace] Loc=<main.c:12:20>
33 semi ';'            Loc=<main.c:12:23>
34 double 'double'     [StartOfLine] [LeadingSpace]  Loc=<main.c:13:5>
35 identifier 'time'    [LeadingSpace] Loc=<main.c:13:12>
36 semi ';'            Loc=<main.c:13:16>
37 identifier 'start'   [StartOfLine] [LeadingSpace]  Loc=<main.c:14:5>
38 equal '='          [LeadingSpace] Loc=<main.c:14:11>
39 identifier 'clock'   [LeadingSpace] Loc=<main.c:14:13>
40 l_paren '('         Loc=<main.c:14:18>
41 r_paren ')'         Loc=<main.c:14:19>
42 semi ';'            Loc=<main.c:14:20>
43 if 'if'             [StartOfLine] [LeadingSpace]  Loc=<main.c:16:5>
44 l_paren '('         [LeadingSpace] Loc=<main.c:16:8>
45 identifier 'i'      Loc=<main.c:16:9>
46 less '<'           [LeadingSpace] Loc=<main.c:16:11>
47 numeric_constant '1' [LeadingSpace] Loc=<main.c:16:13>
48 r_paren ')'         Loc=<main.c:16:14>
49 l_brace '{'         [StartOfLine] [LeadingSpace]  Loc=<main.c:17:5>
50 identifier 'printf'  [StartOfLine] [LeadingSpace]  Loc=<main.c:18:9>
51 l_paren '('         Loc=<main.c:18:15>
52 string_literal '"i 小于1\n"' Loc=<main.c:18:16>
53 r_paren ')'         Loc=<main.c:18:28>
54 semi ';'            Loc=<main.c:18:29>
55 r_brace '}'         [StartOfLine] [LeadingSpace]  Loc=<main.c:19:5>
56 while 'while'       [StartOfLine] [LeadingSpace]  Loc=<main.c:20:5>
57 l_paren '('         [LeadingSpace] Loc=<main.c:20:11>
58 identifier 'i'      Loc=<main.c:20:12>
59 lessequal '<='      [LeadingSpace] Loc=<main.c:20:14>
60 identifier 'n'      [LeadingSpace] Loc=<main.c:20:17>
61 r_paren ')'         Loc=<main.c:20:18>
62 l_brace '{'         [StartOfLine] [LeadingSpace]  Loc=<main.c:21:5>
63 identifier 'f'      [StartOfLine] [LeadingSpace]  Loc=<main.c:22:9>
64 equal '='          [LeadingSpace] Loc=<main.c:22:11>
65 identifier 'f'      [LeadingSpace] Loc=<main.c:22:13>

```

```

66 star '*' [LeadingSpace] Loc=<main.c:22:15>
67 identifier 'i' [LeadingSpace] Loc=<main.c:22:17>
68 semi ';' Loc=<main.c:22:18>
69 identifier 'i' [StartOfLine] [LeadingSpace] Loc=<main.c:23:9>
70 equal '=' [LeadingSpace] Loc=<main.c:23:11>
71 identifier 'i' [LeadingSpace] Loc=<main.c:23:13>
72 plus '+' [LeadingSpace] Loc=<main.c:23:15>
73 numeric_constant '1' [LeadingSpace] Loc=<main.c:23:17>
74 semi ';' Loc=<main.c:23:18>
75 r_brace '}' [StartOfLine] [LeadingSpace] Loc=<main.c:24:5>
76 if 'if' [StartOfLine] [LeadingSpace] Loc=<main.c:25:5>
77 l_paren '(' [LeadingSpace] Loc=<main.c:25:8>
78 identifier 'f' Loc=<main.c:25:9>
79 greater '>' [LeadingSpace] Loc=<main.c:25:11>
80 numeric_constant '1000' [LeadingSpace] Loc=<main.c:25:13 <Spelling=
    main.c:5:13>>
81 r_paren ')' Loc=<main.c:25:16>
82 l_brace '{' [StartOfLine] [LeadingSpace] Loc=<main.c:26:5>
83 identifier 'printf' [StartOfLine] [LeadingSpace] Loc=<main.c:27:9>
84 l_paren '(' Loc=<main.c:27:15>
85 string_literal ""输出超限\n"" Loc=<main.c:27:16>
86 r_paren ')' Loc=<main.c:27:32>
87 semi ';' Loc=<main.c:27:33>
88 r_brace '}' [StartOfLine] [LeadingSpace] Loc=<main.c:28:5>
89 else 'else' [StartOfLine] [LeadingSpace] Loc=<main.c:29:5>
90 l_brace '{' [StartOfLine] [LeadingSpace] Loc=<main.c:30:5>
91 identifier 'printf' [StartOfLine] [LeadingSpace] Loc=<main.c:31:9>
92 l_paren '(' Loc=<main.c:31:15>
93 string_literal ""%d\n"" Loc=<main.c:31:16>
94 comma ',' Loc=<main.c:31:22>
95 identifier 'f' [LeadingSpace] Loc=<main.c:31:24>
96 r_paren ')' Loc=<main.c:31:25>
97 semi ';' Loc=<main.c:31:26>
98 r_brace '}' [StartOfLine] [LeadingSpace] Loc=<main.c:32:5>
99 identifier 'end' [StartOfLine] [LeadingSpace] Loc=<main.c:33:5>
100 equal '=' [LeadingSpace] Loc=<main.c:33:9>
101 identifier 'clock' [LeadingSpace] Loc=<main.c:33:11>
102 l_paren '(' Loc=<main.c:33:16>
103 r_paren ')' Loc=<main.c:33:17>
104 semi ';' Loc=<main.c:33:18>
105 identifier 'time' [StartOfLine] [LeadingSpace] Loc=<main.c:34:5>
106 equal '=' [LeadingSpace] Loc=<main.c:34:10>
107 l_paren '(' [LeadingSpace] Loc=<main.c:34:12>
108 double 'double' Loc=<main.c:34:13>
109 r_paren ')' Loc=<main.c:34:19>
110 l_paren '(' Loc=<main.c:34:20>
111 identifier 'end' Loc=<main.c:34:21>
112 minus '-' [LeadingSpace] Loc=<main.c:34:25>

```

```

113 identifier 'start'      [LeadingSpace] Loc=<main.c:34:27>
114 r_paren ')'            Loc=<main.c:34:32>
115 slash '/'             [LeadingSpace] Loc=<main.c:34:34>
116 l_paren '('           [LeadingSpace] Loc=<main.c:34:36 <Spelling=/usr/include/
      x86_64-linux-gnu/bits/time.h:34:25>>
117 l_paren '('           Loc=<main.c:34:36 <Spelling=/usr/include/x86_64-linux
      -gnu/bits/time.h:34:26>>
118 identifier '__clock_t' Loc=<main.c:34:36 <Spelling=/usr/include/
      x86_64-linux-gnu/bits/time.h:34:27>>
119 r_paren ')'            Loc=<main.c:34:36 <Spelling=/usr/include/x86_64-linux
      -gnu/bits/time.h:34:36>>
120 numeric_constant '1000000' [LeadingSpace] Loc=<main.c:34:36 <Spelling=/
      usr/include/x86_64-linux-gnu/bits/time.h:34:38>>
121 r_paren ')'            Loc=<main.c:34:36 <Spelling=/usr/include/x86_64-linux
      -gnu/bits/time.h:34:45>>
122 semi ';'              Loc=<main.c:34:50>
123 identifier 'printf'    [StartOfLine] [LeadingSpace] Loc=<main.c:35:5>
124 l_paren '('           Loc=<main.c:35:11>
125 string_literal '"程序运行时间为: %f\n"' Loc=<main.c:35:12>
126 comma ','             Loc=<main.c:35:42>
127 identifier 'time'      [LeadingSpace] Loc=<main.c:35:44>
128 r_paren ')'            Loc=<main.c:35:48>
129 semi ';'              Loc=<main.c:35:49>
130 return 'return'        [StartOfLine] [LeadingSpace] Loc=<main.c:36:5>
131 numeric_constant '0'   [LeadingSpace] Loc=<main.c:36:12>
132 semi ';'              Loc=<main.c:36:13>
133 r_brace '}'            [StartOfLine] Loc=<main.c:37:1>
134 eof ''                 Loc=<main.c:37:2>

```

观察可知，在词法分析阶段，源程序中的字符串被扫描并分解，识别成为一个个单词，并被写明其类型，便于后续的语法分析。

## 2. 语法分析

将词法分析生成的词法单元来构建抽象语法树 (Abstract Syntax Tree, 即 AST)。通过 clang -E -Xclang -ast-dump main.c 命令获得相应的 AST，如图3所示：

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8  #include <sys/mman.h>
9  #include <sys/time.h>
10 #include <sys/resource.h>
11 #include <sys/wait.h>
12 #include <sys/queue.h>
13 #include <sys/uio.h>
14 #include <sys/xattr.h>
15 #include <sys/zfs.h>
16 #include <sys/zfs_ioctl.h>
17 #include <sys/zfs_mount.h>
18 #include <sys/zfs_name.h>
19 #include <sys/zfs_objset.h>
20 #include <sys/zfs_refcount.h>
21 #include <sys/zfs_replay.h>
22 #include <sys/zfs_spa.h>
23 #include <sys/zfs_vdev.h>
24 #include <sys/zfs_vfs.h>
25 #include <sys/zfs_znode.h>
26 #include <sys/zfs_zpool.h>
27 #include <sys/zfs_zprop.h>
28 #include <sys/zfs_zutil.h>
29 #include <sys/zfs_zutil_os.h>
30 #include <sys/zfs_zutil_unix.h>
31 #include <sys/zfs_zutil_windows.h>
32 #include <sys/zfs_zutil_linux.h>
33 #include <sys/zfs_zutil_darwin.h>
34 #include <sys/zfs_zutil_freebsd.h>
35 #include <sys/zfs_zutil_netbsd.h>
36 #include <sys/zfs_zutil_openbsd.h>
37 #include <sys/zfs_zutil_solaris.h>
38 #include <sys/zfs_zutil_hurd.h>
39 #include <sys/zfs_zutil_rtems.h>
40 #include <sys/zfs_zutil_tru64.h>
41 #include <sys/zfs_zutil_vxworks.h>
42 #include <sys/zfs_zutil_arm.h>
43 #include <sys/zfs_zutil_mips.h>
44 #include <sys/zfs_zutil_powerpc.h>
45 #include <sys/zfs_zutil_sh.h>
46 #include <sys/zfs_zutil_sparc.h>
47 #include <sys/zfs_zutil_tic6x.h>
48 #include <sys/zfs_zutil_x86.h>
49 #include <sys/zfs_zutil_x86_64.h>
50 #include <sys/zfs_zutil_x86_64_32.h>
51 #include <sys/zfs_zutil_x86_64_64.h>
52 #include <sys/zfs_zutil_x86_64_128.h>
53 #include <sys/zfs_zutil_x86_64_256.h>
54 #include <sys/zfs_zutil_x86_64_512.h>
55 #include <sys/zfs_zutil_x86_64_1024.h>
56 #include <sys/zfs_zutil_x86_64_2048.h>
57 #include <sys/zfs_zutil_x86_64_4096.h>
58 #include <sys/zfs_zutil_x86_64_8192.h>
59 #include <sys/zfs_zutil_x86_64_16384.h>
60 #include <sys/zfs_zutil_x86_64_32768.h>
61 #include <sys/zfs_zutil_x86_64_65536.h>
62 #include <sys/zfs_zutil_x86_64_131072.h>
63 #include <sys/zfs_zutil_x86_64_262144.h>
64 #include <sys/zfs_zutil_x86_64_524288.h>
65 #include <sys/zfs_zutil_x86_64_1048576.h>
66 #include <sys/zfs_zutil_x86_64_2097152.h>
67 #include <sys/zfs_zutil_x86_64_4194304.h>
68 #include <sys/zfs_zutil_x86_64_8388608.h>
69 #include <sys/zfs_zutil_x86_64_16777216.h>
70 #include <sys/zfs_zutil_x86_64_33554432.h>
71 #include <sys/zfs_zutil_x86_64_67108864.h>
72 #include <sys/zfs_zutil_x86_64_134217728.h>
73 #include <sys/zfs_zutil_x86_64_268435456.h>
74 #include <sys/zfs_zutil_x86_64_536870912.h>
75 #include <sys/zfs_zutil_x86_64_1073741824.h>
76 #include <sys/zfs_zutil_x86_64_2147483648.h>
77 #include <sys/zfs_zutil_x86_64_4294967296.h>
78 #include <sys/zfs_zutil_x86_64_8589934592.h>
79 #include <sys/zfs_zutil_x86_64_17179869184.h>
80 #include <sys/zfs_zutil_x86_64_34359738368.h>
81 #include <sys/zfs_zutil_x86_64_68719476736.h>
82 #include <sys/zfs_zutil_x86_64_137438953472.h>
83 #include <sys/zfs_zutil_x86_64_274877906944.h>
84 #include <sys/zfs_zutil_x86_64_549755813888.h>
85 #include <sys/zfs_zutil_x86_64_1099511627776.h>
86 #include <sys/zfs_zutil_x86_64_2199023255552.h>
87 #include <sys/zfs_zutil_x86_64_4398046511104.h>
88 #include <sys/zfs_zutil_x86_64_8796093022208.h>
89 #include <sys/zfs_zutil_x86_64_17592186044416.h>
90 #include <sys/zfs_zutil_x86_64_35184372088832.h>
91 #include <sys/zfs_zutil_x86_64_70368744177664.h>
92 #include <sys/zfs_zutil_x86_64_140737488355328.h>
93 #include <sys/zfs_zutil_x86_64_281474976710656.h>
94 #include <sys/zfs_zutil_x86_64_562949953421312.h>
95 #include <sys/zfs_zutil_x86_64_1125899906842624.h>
96 #include <sys/zfs_zutil_x86_64_2251799813685248.h>
97 #include <sys/zfs_zutil_x86_64_4503599627370496.h>
98 #include <sys/zfs_zutil_x86_64_9007199254740992.h>
99 #include <sys/zfs_zutil_x86_64_18014398509481984.h>
100 #include <sys/zfs_zutil_x86_64_36028797018963968.h>
101 #include <sys/zfs_zutil_x86_64_72057594037927936.h>
102 #include <sys/zfs_zutil_x86_64_144115188075855872.h>
103 #include <sys/zfs_zutil_x86_64_288230376151711744.h>
104 #include <sys/zfs_zutil_x86_64_576460752303423488.h>
105 #include <sys/zfs_zutil_x86_64_1152921504606846976.h>
106 #include <sys/zfs_zutil_x86_64_2305843009213693952.h>
107 #include <sys/zfs_zutil_x86_64_4611686018427387904.h>
108 #include <sys/zfs_zutil_x86_64_9223372036854775808.h>
109 #include <sys/zfs_zutil_x86_64_18446744073709551616.h>
110 #include <sys/zfs_zutil_x86_64_36893488147419103232.h>
111 #include <sys/zfs_zutil_x86_64_73786976294838206464.h>
112 #include <sys/zfs_zutil_x86_64_147573952589676412928.h>
113 #include <sys/zfs_zutil_x86_64_295147905179352825856.h>
114 #include <sys/zfs_zutil_x86_64_590295810358705651712.h>
115 #include <sys/zfs_zutil_x86_64_1180591620717411303424.h>
116 #include <sys/zfs_zutil_x86_64_2361183241434822606848.h>
117 #include <sys/zfs_zutil_x86_64_4722366482869645213696.h>
118 #include <sys/zfs_zutil_x86_64_9444732965739290427392.h>
119 #include <sys/zfs_zutil_x86_64_18889465931478580854784.h>
120 #include <sys/zfs_zutil_x86_64_37778931862957161709568.h>
121 #include <sys/zfs_zutil_x86_64_75557863725914323419136.h>
122 #include <sys/zfs_zutil_x86_64_151115727451828646838272.h>
123 #include <sys/zfs_zutil_x86_64_302231454903657293676544.h>
124 #include <sys/zfs_zutil_x86_64_604462909807314587353088.h>
125 #include <sys/zfs_zutil_x86_64_1208925819614629174706176.h>
126 #include <sys/zfs_zutil_x86_64_2417851639229258349412352.h>
127 #include <sys/zfs_zutil_x86_64_4835703278458516698824704.h>
128 #include <sys/zfs_zutil_x86_64_9671406556917033397649408.h>
129 #include <sys/zfs_zutil_x86_64_19342813113834066795298816.h>
130 #include <sys/zfs_zutil_x86_64_38685626227668133590597632.h>
131 #include <sys/zfs_zutil_x86_64_77371252455336267181195264.h>
132 #include <sys/zfs_zutil_x86_64_154742504910672534362390528.h>
133 #include <sys/zfs_zutil_x86_64_309485009821345068724781056.h>
134 #include <sys/zfs_zutil_x86_64_618970019642690137449562112.h>
135 #include <sys/zfs_zutil_x86_64_1237940039285380274899124224.h>
136 #include <sys/zfs_zutil_x86_64_2475880078570760549798248448.h>
137 #include <sys/zfs_zutil_x86_64_4951760157141521099596496896.h>
138 #include <sys/zfs_zutil_x86_64_9903520314283042199192993792.h>
139 #include <sys/zfs_zutil_x86_64_19807040628566084398385987584.h>
140 #include <sys/zfs_zutil_x86_64_39614081257132168796771975168.h>
141 #include <sys/zfs_zutil_x86_64_79228162514264337593543950336.h>
142 #include <sys/zfs_zutil_x86_64_158456325028528675187087900672.h>
143 #include <sys/zfs_zutil_x86_64_316912650057057350374175801344.h>
144 #include <sys/zfs_zutil_x86_64_633825300114114700748351602688.h>
145 #include <sys/zfs_zutil_x86_64_1267650600228229401496703205376.h>
146 #include <sys/zfs_zutil_x86_64_2535301200456458802993406410752.h>
147 #include <sys/zfs_zutil_x86_64_5070602400912917605986812821504.h>
148 #include <sys/zfs_zutil_x86_64_10141204801825835211973625643008.h>
149 #include <sys/zfs_zutil_x86_64_20282409603651670423947251286016.h>
150 #include <sys/zfs_zutil_x86_64_40564819207303340847894502572032.h>
151 #include <sys/zfs_zutil_x86_64_81129638414606681695789005144064.h>
152 #include <sys/zfs_zutil_x86_64_162259276829213363391578010288128.h>
153 #include <sys/zfs_zutil_x86_64_324518553658426726783156020576256.h>
154 #include <sys/zfs_zutil_x86_64_649037107316853453566312041152512.h>
155 #include <sys/zfs_zutil_x86_64_1298074214633707107132624082305024.h>
156 #include <sys/zfs_zutil_x86_64_2596148429267414214265248164610048.h>
157 #include <sys/zfs_zutil_x86_64_5192296858534828428530496329220096.h>
158 #include <sys/zfs_zutil_x86_64_10384593717069656857060992658440192.h>
159 #include <sys/zfs_zutil_x86_64_20769187434139313714121985316880384.h>
160 #include <sys/zfs_zutil_x86_64_41538374868278627428243970633760768.h>
161 #include <sys/zfs_zutil_x86_64_83076749736557254856487941267521536.h>
162 #include <sys/zfs_zutil_x86_64_166153499473114509712975882535043072.h>
163 #include <sys/zfs_zutil_x86_64_332306998946229019425951765070086144.h>
164 #include <sys/zfs_zutil_x86_64_664613997892458038851903530140172288.h>
165 #include <sys/zfs_zutil_x86_64_1329227995784916077703807060280344576.h>
166 #include <sys/zfs_zutil_x86_64_2658455991569832155407614120560689152.h>
167 #include <sys/zfs_zutil_x86_64_5316911983139664310815228241121378304.h>
168 #include <sys/zfs_zutil_x86_64_10633823966279328621630456482242756608.h>
169 #include <sys/zfs_zutil_x86_64_21267647932558657243260912964485513216.h>
170 #include <sys/zfs_zutil_x86_64_42535295865117314486521825928971026432.h>
171 #include <sys/zfs_zutil_x86_64_85070591730234628973043651857942052864.h>
172 #include <sys/zfs_zutil_x86_64_170141183460469257946087303715884105728.h>
173 #include <sys/zfs_zutil_x86_64_340282366920938515892174607431768211456.h>
174 #include <sys/zfs_zutil_x86_64_680564733841877031784349214863536422912.h>
175 #include <sys/zfs_zutil_x86_64_1361129467683754063568698429727072845824.h>
176 #include <sys/zfs_zutil_x86_64_2722258935367508127137396859454145691648.h>
177 #include <sys/zfs_zutil_x86_64_5444517870735016254274793718908291383296.h>
178 #include <sys/zfs_zutil_x86_64_10889035741470032508549587437816582766592.h>
179 #include <sys/zfs_zutil_x86_64_21778071482940065017099174875633165533184.h>
180 #include <sys/zfs_zutil_x86_64_43556142965880130034198349751266331066368.h>
181 #include <sys/zfs_zutil_x86_64_87112285931760260068396699502532662132736.h>
182 #include <sys/zfs_zutil_x86_64_174224571863520520136793399005065324265472.h>
183 #include <sys/zfs_zutil_x86_64_348449143727041040273586798010130648530944.h>
184 #include <sys/zfs_zutil_x86_64_696898287454082080547173596020261290861888.h>
185 #include <sys/zfs_zutil_x86_64_1393796574908164161094347192040522581723776.h>
186 #include <sys/zfs_zutil_x86_64_2787593149816328322188694384081045163447552.h>
187 #include <sys/zfs_zutil_x86_64_5575186299632656644377388768162090326895104.h>
188 #include <sys/zfs_zutil_x86_64_11150372599265313288754777536324180653790208.h>
189 #include <sys/zfs_zutil_x86_64_22300745198530626577509555072648361307580416.h>
190 #include <sys/zfs_zutil_x86_64_44601490397061253155019110145296722615160832.h>
191 #include <sys/zfs_zutil_x86_64_89202980794122506310038220290593445230321664.h>
192 #include <sys/zfs_zutil_x86_64_178405961588245012620076440581186890460643328.h>
193 #include <sys/zfs_zutil_x86_64_356811923176490025240152881162373780921286656.h>
194 #include <sys/zfs_zutil_x86_64_713623846352980050480305762324747561842573312.h>
195 #include <sys/zfs_zutil_x86_64_1427247692705960100960611524649495123685146624.h>
196 #include <sys/zfs_zutil_x86_64_2854495385411920201921223049298990247372893248.h>
197 #include <sys/zfs_zutil_x86_64_5708990770823840403842446098597980494745786496.h>
198 #include <sys/zfs_zutil_x86_64_11417981541647680807684892197195960989491572992.h>
199 #include <sys/zfs_zutil_x86_64_22835963083295361615369784394391921978983145984.h>
200 #include <sys/zfs_zutil_x86_64_45671926166590723230739568788783843957966291968.h>
201 #include <sys/zfs_zutil_x86_64_91343852333181446461479137577567687915932583936.h>
202 #include <sys/zfs_zutil_x86_64_182687704666362892922958275155135375831865167872.h>
203 #include <sys/zfs_zutil_x86_64_365375409332725785845916550310270751663730335744.h>
204 #include <sys/zfs_zutil_x86_64_730750818665451571691833100620541503327460671488.h>
205 #include <sys/zfs_zutil_x86_64_1461501637330903143383666201241083006654921342976.h>
206 #include <sys/zfs_zutil_x86_64_2923003274661806286767332402482166013309842685952.h>
207 #include <sys/zfs_zutil_x86_64_5846006549323612573534664804964332026619685371904.h>
208 #include <sys/zfs_zutil_x86_64_1169201309864722514706932960992866405323937067392.h>
209 #include <sys/zfs_zutil_x86_64_2338402619729445029413865921985732810647874134784.h>
210 #include <sys/zfs_zutil_x86_64_4676805239458890058827731843971465621295748269568.h>
211 #include <sys/zfs_zutil_x86_64_9353610478917780117655463687942931242591496539136.h>
212 #include <sys/zfs_zutil_x86_64_18707220957835560235310927375885862485182993078272.h>
213 #include <sys/zfs_zutil_x86_64_3741444191567112047062185475177172497036598615648.h>
214 #include <sys/zfs_zutil_x86_64_7482888383134224094124370950354344994073197231296.h>
215 #include <sys/zfs_zutil_x86_64_14965776766268448188248741900708689988146394462592.h>
216 #include <sys/zfs_zutil_x86_64_29931553532536896376497483801417379976292788925184.h>
217 #include <sys/zfs_zutil_x86_64_59863107065073792752994967602834759952585577850368.h>
218 #include <sys/zfs_zutil_x86_64_119726214130147585505989935205669519905171155700736.h>
219 #include <sys/zfs_zutil_x86_64_239452428260295171011979870411339039810342311401472.h>
220 #include <sys/zfs_zutil_x86_64_478904856520590342023959740822678079620684622802944.h>
221 #include <sys/zfs_zutil_x86_64_957809713041180684047919481645356159241369245605888.h>
222 #include <sys/zfs_zutil_x86_64_1915619426082361368095838963290712318482738491211776.h>
223 #include <sys/zfs_zutil_x86_64_3831238852164722736191677926581424636965476982423552.h>
224 #include <sys/zfs_zutil_x86_64_7662477704329445472383355853162849273930953964847104.h>
225 #include <sys/zfs_zutil_x86_64_15324955408658890944766711706325698547861907929694208.h>
226 #include <sys/zfs_zutil_x86_64_30649910817317781889533423412651397095723815859388416.h>
227 #include <sys/zfs_zutil_x86_64_61299821634635563779066846825302794191447631718776832.h>
228 #include <sys/zfs_zutil_x86_64_122599643269271127558133693650605588382895263437553664.h>
229 #include <sys/zfs_zutil_x86_64_245199286538542255116267387301211176765790526875107328.h>
230 #include <sys/zfs_zutil_x86_64_490398573077084510232534774602422353531581053750214656.h>
231 #include <sys/zfs_zutil_x86_64_980797146154169020465069549204844707063162107500429312.h>
232 #include <sys/zfs_zutil_x86_64_1961594292288338040930139098409689414126324215000858624.h>
233 #include <sys/zfs_zutil_x86_64_3923188584576676081860278196819378828252648430001717248.h>
234 #include <sys/zfs_zutil_x86_64_7846377169153352163720556393638757656505296860003434496.h>
235 #include <sys/zfs_zutil_x86_64_15692754338306704327441112787277515313010593720006868992.h>
236 #include <sys/zfs_zutil_x86_64_3138550867661340865488222557455503062602118744001373792.h>
237 #include <sys/zfs_zutil_x86_64_6277101735322681730976445114911006125324237488002747584.h>
238 #include <sys/zfs_zutil_x86_64_12554203470645363461952890229822012506448474976004495168.h>
239 #include <sys/zfs_zutil_x86_64_25108406941290726923905780459644025012896949952008990336.h>
240 #include <sys/zfs_zutil_x86_64_50216813882581453847811560919288050025793899904017980672.h>
241 #include <sys/zfs_zutil_x86_64_100433627765162907695623121838576100051587799808035961344.h>
242 #include <sys/zfs_zutil_x86_64_200867255530325815391246243677152200103175599616071922688.h>
243 #include <sys/zfs_zutil_x86_64_401734511060651630782492487354304400206351199232143845376.h>
244 #include <sys/zfs_zutil_x86_64_803469022121303261564984974708608800412702398464287690752.h>
245 #include <sys/zfs_zutil_x86_64_1606938044242606523129969949417217600825404796928575381504.h>
246 #include <sys/zfs_zutil_x86_64_3213876088485213046259939898834435201650809593857150763008.h>
247 #include <sys/zfs_zutil_x86_64_6427752176970426092519879797668870403301619187714301526016.h>
248 #include <sys/zfs_zutil_x86_64_12855504353940852185039759595337740806603238375428603052032.h>
249 #include <sys/zfs_zutil_x86_64_25711008707881704370079519190675481613206476750857206104064.h>
250 #include <sys/zfs_zutil_x86_64_51422017415763408740159038381350963226412953501714412208128.h>
251 #include <sys/zfs_zutil_x86_64_102844034831526817480318076762701826452825907003428824416256.h>
252 #include <sys/zfs_zutil_x86_64_205688069663053634960636153525403652905651814006857648832512.h>
253 #include <sys/zfs_zutil_x86_64_411376139326107269921272307050807305811303628013715297665024.h>
254 #include <sys/zfs_zutil_x86_64_822752278652214539842544614101614611622607256027430595330048.h>
255 #include <sys/zfs_zutil_x86_64_1645504557304
```

```

4 target triple = "x86_64-pc-linux-gnu"
5
6 @.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
7 @.str.1 = private unnamed_addr constant [10 x i8] c"i\E5\B0\8F\E4\BA\8E1\0A
   \00", align 1
8 @.str.2 = private unnamed_addr constant [14 x i8] c"\E8\BE\93\E5\87\BA\E8\B6
   \85\E9\99\90\0A\00", align 1
9 @.str.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
10 @.str.4 = private unnamed_addr constant [28 x i8] c"\E7\A8\8B\E5\BA\8F\E8\BF
   \90\E8\A1\8C\E6\97\B6\E9\97\B4\E4\B8\BA\EF\BC\9A%f\0A\00", align 1
11
12 ; Function Attrs: noinline nounwind optnone uwtable
13 define dso_local i32 @main() #0 {
14     %1 = alloca i32, align 4
15     %2 = alloca i32, align 4
16     %3 = alloca i32, align 4
17     %4 = alloca i32, align 4
18     %5 = alloca i64, align 8
19     %6 = alloca i64, align 8
20     %7 = alloca double, align 8
21     store i32 0, i32* %1, align 4
22     %8 = call i32 (@__isoc99_scanf(i8* noundef getelementptr inbounds
   ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32* noundef %3)
23     store i32 2, i32* %2, align 4
24     store i32 1, i32* %4, align 4
25     %9 = call i64 @clock() #3
26     store i64 %9, i64* %5, align 8
27     %10 = load i32, i32* %2, align 4
28     %11 = icmp slt i32 %10, 1
29     br i1 %11, label %12, label %14
30
31 12:                                     ; preds = %0
32     %13 = call i32 (@__printf(i8* noundef getelementptr inbounds ([10 x
   i8], [10 x i8]* @.str.1, i64 0, i64 0))
33     br label %14
34
35 14:                                     ; preds = %12, %0
36     br label %15
37
38 15:                                     ; preds = %19, %14
39     %16 = load i32, i32* %2, align 4
40     %17 = load i32, i32* %3, align 4
41     %18 = icmp sle i32 %16, %17
42     br i1 %18, label %19, label %25
43
44 19:                                     ; preds = %15
45     %20 = load i32, i32* %4, align 4
46     %21 = load i32, i32* %2, align 4

```

```

47 %22 = mul nsw i32 %20, %21
48 store i32 %22, i32* %4, align 4
49 %23 = load i32, i32* %2, align 4
50 %24 = add nsw i32 %23, 1
51 store i32 %24, i32* %2, align 4
52 br label %15, !llvm.loop !6
53
54 25:                                     ; preds = %15
55 %26 = load i32, i32* %4, align 4
56 %27 = icmp sgt i32 %26, 10000
57 br i1 %27, label %28, label %30
58
59 28:                                     ; preds = %25
60 %29 = call i32 @__isoc99_scanf(i8* noundef %getelementptr inbounds ([14 x
61 i8], [14 x i8]* @.str.2, i64 0, i64 0))
62 br label %33
63
64 30:                                     ; preds = %25
65 %31 = load i32, i32* %4, align 4
66 %32 = call i32 @__isoc99_scanf(i8* noundef %getelementptr inbounds ([4 x
67 i8], [4 x i8]* @.str.3, i64 0, i64 0), i32 noundef %31)
68 br label %33
69
70 33:                                     ; preds = %30, %28
71 %34 = call i64 @clock() #3
72 store i64 %34, i64* %6, align 8
73 %35 = load i64, i64* %6, align 8
74 %36 = load i64, i64* %5, align 8
75 %37 = sub nsw i64 %35, %36
76 %38 = sitofp i64 %37 to double
77 %39 = fdiv double %38, 1.000000e+06
78 store double %39, double* %7, align 8
79 %40 = load double, double* %7, align 8
80 %41 = call i32 @__isoc99_scanf(i8* noundef %getelementptr inbounds ([28 x
81 i8], [28 x i8]* @.str.4, i64 0, i64 0), double noundef %40)
82 ret i32 0
83 }
84
85 declare i32 @__isoc99_scanf(i8* noundef, ...) #1
86
87 ; Function Attrs: nounwind
88 declare i64 @clock() #2
89
90 declare i32 @printf(i8* noundef, ...) #1
91
92 attributes #0 = { noline nounwind optnone uwtable "frame-pointer"="all" "
min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-
buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx

```

```

    ,+sse,+sse2,+x87" "tune-cpu"="generic" }
90 attributes #1 = { "frame-pointer"="all" "no-trapping-math"="true" "stack-
    protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8
    ,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
91 attributes #2 = { nounwind "frame-pointer"="all" "no-trapping-math"="true" "
    stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"=
    "+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
92 attributes #3 = { nounwind }
93
94 !llvm.module.flags = !{!0, !1, !2, !3, !4}
95 !llvm.ident = !{!5}
96
97 !0 = !{i32 1, !"wchar_size", i32 4}
98 !1 = !{i32 7, !"PIC Level", i32 2}
99 !2 = !{i32 7, !"PIE Level", i32 2}
100 !3 = !{i32 7, !"uwtable", i32 1}
101 !4 = !{i32 7, !"frame-pointer", i32 2}
102 !5 = !{!"Ubuntu clang version 14.0.0-1ubuntu1.1"}
103 !6 = distinct !{!6, !7}
104 !7 = !{!"llvm.loop.mustprogress"}

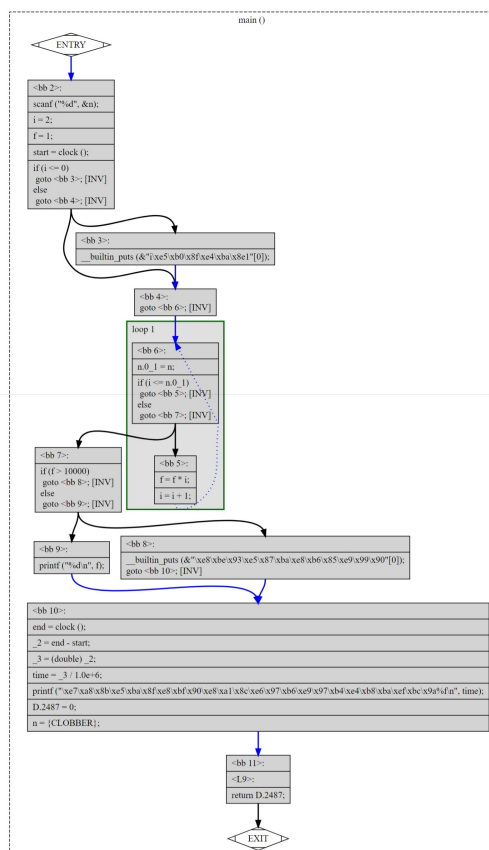
```

在 LLVM IR 程序的开始，给出了模块的 ID 并记录了源文件的程序名，接着记录了程序数据的字节顺序、类型长度以及程序运行环境等信息。接着是对出现在程序中所有的字符串按顺序定义为全局的常量（以 @ 开头），记录了每个字符串的长度和内容。

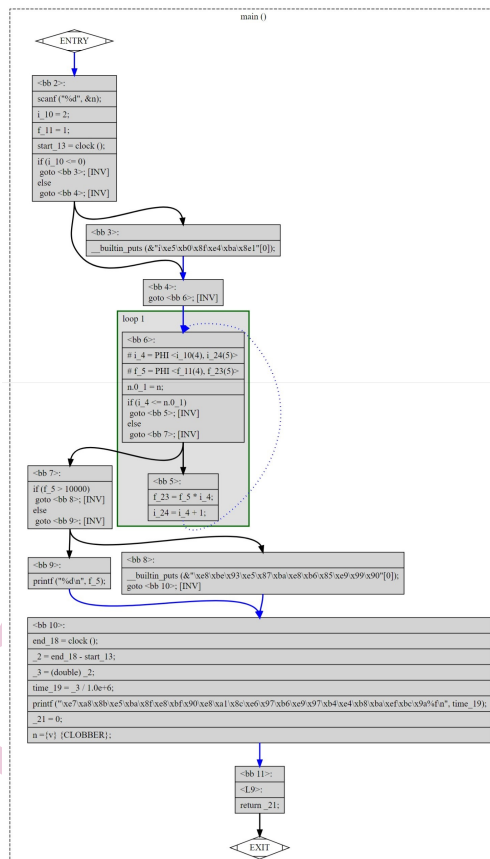
之后声明了返回类型为 i32 的 main 函数，#0 映射到一组函数属性，定义在文件的末尾。在函数内部，依次在栈帧中为函数中出现的所有变量分配了空间，按照源程序的内容初始化相应变量，调用相应的函数。对于每一个分支语句，如 if，while 等，会在可能需要跳转的地方生成标签，生成分支条件并控制跳转到相应的标签。在程序中部分指令后面会带有 nsw 的标记，这个标记指定该操作是不会溢出的，允许进行某些优化。函数的最后返回 0，结束调用。

在程序的最后，给出了所有调用的外部函数的声明和若干组函数属性，以及编译器的配置信息等。

利用 gcc -fdump-tree-all-graph main.c 命令可以得到中间代码生成的多阶段输出，此处借助 VS code 中的插件实现对 CFG 的可视化并加以分析。



(a) a-main.c.022t.fixup-cfg1.dot



(b) a-main.c.027t.fixup-cfg2.dot

图 4: CFG 可视化

CFG 图将代码转换为控制流图，可以清晰的反映出代码段之间的跳转关系。

观察生成图片可以发现，控制流、变量名等发生变化，推测其可能为代码生成不同阶段中的不同值。

#### 4. 代码优化

##### (1) 格式转换

借助 `llvm-as main.ll -o main.bc` 命令得到 LLVM IR 的二进制代码形式。

##### (2) 优化比较

借助 `opt -S -O0 main.bc -o main-O0.ll` 命令，实现对其 O0 优化。得到中间代码后继续处理，得到可执行文件，运行比较优化性能。

借助 `opt -S -O1 main.bc -o main-O1.ll` 命令，实现对其 O1 优化。得到中间代码后继续处理，得到可执行文件，运行比较优化性能。

借助 `opt -S -O2 main.bc -o main-O2.ll` 命令，实现对其 O2 优化。得到中间代码后继续处理，得到可执行文件，运行比较优化性能。

经过代码比较可知，O2 级优化与 O1 级优化代码相同，故而可知，O1 级优化已经优化完毕。运行三个可执行文件可知，O1 较 O0 有了一定程度优化，但 O2 级优化出现了不稳定情况。但是综合来看优化效果有限，推测可能是因为程序过于简单所导致。



```

lxmliu2002@lxmliu2002-Ubuntu:~/桌面/lab1$ ./main_00
10
输出超限
程序运行时间为: 0.000032
lxmliu2002@lxmliu2002-Ubuntu:~/桌面/lab1$ ./main_01
10
输出超限
程序运行时间为: 0.000016
lxmliu2002@lxmliu2002-Ubuntu:~/桌面/lab1$ ./main_02
10
输出超限
程序运行时间为: 0.000109

```

图 5: 优化结果比较

## 5. 代码生成

以中间表示形式作为输入，将其映射到目标语言。

利用 `llc main.ll -o main.S` 命令生成目标代码；利用 `gcc main.i -S -o main.S` 命令生成 x86 格式目标代码；利用 `arm-linux-gnueabi-gcc main.i -S -o main.S` 命令生成 arm 格式目标代码。由于篇幅有限，代码文件内容在此不做展示，源文件已放于附件之中。

### (三) 汇编器

汇编过程实际上把汇编语言程序代码翻译成目标机器指令的过程，其最终生成的是可重定位的机器代码。汇编器会将汇编程序中的伪指令翻译成等价的机器语言指令，将分支和数据传输指令中用到的标号都放入一个符号表中，通过查表可以生成对应指令的二进制表示。

分别使用 `gcc main.S -c -o main.o` 命令、`arm-linux-gnueabi-gcc main.S -o main.o` 命令以及 `llc main.bc -filetype=obj -o main.o` 命令实现对 x86 格式、arm 格式以及 LLVM 格式文件进行汇编，得到 main.o 文件。

使用 `objdump -d main_gcc.o` 命令、`arm-linux-gnueabi-objdump -d main_arm.o` 命令对上述文件进行反汇编。由于内容所限，此处仅对 x86 反汇编进行展示。

#### x86 反编译结果

```

1  main_gcc.o:      文件格式 elf64-x86-64
2
3  Disassembly of section .text:
4
5  0000000000000000 <main>:
6      0:  f3 0f 1e fa      endbr64
7      4:  55               push    %rbp
8      5:  48 89 e5         mov     %rsp,%rbp
9      8:  48 83 ec 30      sub     $0x30,%rsp
10     c:  64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
11    13:  00 00
12    15:  48 89 45 f8      mov     %rax,-0x8(%rbp)
13    19:  31 c0           xor     %eax,%eax
14    1b:  48 8d 45 d4      lea     -0x2c(%rbp),%rax
15    1f:  48 89 c6         mov     %rax,%rsi
16    22:  48 8d 05 00 00 00 00 lea     0x0(%rip),%rax      # 29 <main+0x29>
17    29:  48 89 c7         mov     %rax,%rdi

```

```

18  2c:  b8 00 00 00 00      mov     $0x0,%eax
19  31:  e8 00 00 00 00      call    36 <main+0x36>
20  36:  c7 45 d8 02 00 00 00  movl    $0x2,-0x28(%rbp)
21  3d:  c7 45 dc 01 00 00 00  movl    $0x1,-0x24(%rbp)
22  44:  e8 00 00 00 00      call    49 <main+0x49>
23  49:  48 89 45 e0          mov     %rax,-0x20(%rbp)
24  4d:  83 7d d8 00          cmpl    $0x0,-0x28(%rbp)
25  51:  7f 1f              jg      72 <main+0x72>
26  53:  48 8d 05 00 00 00 00  lea     0x0(%rip),%rax      # 5a <main+0x5a>
27  5a:  48 89 c7            mov     %rax,%rdi
28  5d:  e8 00 00 00 00      call    62 <main+0x62>
29  62:  eb 0e              jmp     72 <main+0x72>
30  64:  8b 45 dc            mov     -0x24(%rbp),%eax
31  67:  0f af 45 d8          imul    -0x28(%rbp),%eax
32  6b:  89 45 dc            mov     %eax,-0x24(%rbp)
33  6e:  83 45 d8 01          addl    $0x1,-0x28(%rbp)
34  72:  8b 45 d4            mov     -0x2c(%rbp),%eax
35  75:  39 45 d8            cmp     %eax,-0x28(%rbp)
36  78:  7e ea              jle     64 <main+0x64>
37  7a:  81 7d dc 10 27 00 00  cmpl    $0x2710,-0x24(%rbp)
38  81:  7e 11              jle     94 <main+0x94>
39  83:  48 8d 05 00 00 00 00  lea     0x0(%rip),%rax      # 8a <main+0x8a>
40  8a:  48 89 c7            mov     %rax,%rdi
41  8d:  e8 00 00 00 00      call    92 <main+0x92>
42  92:  eb 19              jmp     ad <main+0xad>
43  94:  8b 45 dc            mov     -0x24(%rbp),%eax
44  97:  89 c6              mov     %eax,%esi
45  99:  48 8d 05 00 00 00 00  lea     0x0(%rip),%rax      # a0 <main+0xa0>
46  a0:  48 89 c7            mov     %rax,%rdi
47  a3:  b8 00 00 00 00      mov     $0x0,%eax
48  a8:  e8 00 00 00 00      call    ad <main+0xad>
49  ad:  e8 00 00 00 00      call    b2 <main+0xb2>
50  b2:  48 89 45 e8          mov     %rax,-0x18(%rbp)
51  b6:  48 8b 45 e8          mov     -0x18(%rbp),%rax
52  ba:  48 2b 45 e0          sub     -0x20(%rbp),%rax
53  be:  66 0f ef c0          pxor    %xmm0,%xmm0
54  c2:  f2 48 0f 2a c0       cvtsi2sd %rax,%xmm0
55  c7:  f2 0f 10 0d 00 00 00  movsdb  0x0(%rip),%xmm1      # cf <main+0xcf>
>
56  ce:  00
57  cf:  f2 0f 5e c1          divsd   %xmm1,%xmm0
58  d3:  f2 0f 11 45 f0       movsdb  %xmm0,-0x10(%rbp)
59  d8:  48 8b 45 f0          mov     -0x10(%rbp),%rax
60  dc:  66 48 0f 6e c0       movq    %rax,%xmm0
61  e1:  48 8d 05 00 00 00 00  lea     0x0(%rip),%rax      # e8 <main+0xe8>
62  e8:  48 89 c7            mov     %rax,%rdi
63  eb:  b8 01 00 00 00      mov     $0x1,%eax
64  f0:  e8 00 00 00 00      call    f5 <main+0xf5>

```

```

65 f5: b8 00 00 00 00      mov     $0x0,%eax
66 fa: 48 8b 55 f8          mov     -0x8(%rbp),%rdx
67 fe: 64 48 2b 14 25 28 00 sub     %fs:0x28,%rdx
68 105: 00 00
69 107: 74 05                je      10e <main+0x10e>
70 109: e8 00 00 00 00        call   10e <main+0x10e>
71 10e: c9                    leave
72 10f: c3                    ret

```

将反编译的结果与汇编程序对比，可以发现这里所有涉及到变量、分支标签、调用外部函数对应位置的二进制代码全部为 0，反编译结果中的 call 指令全部指向了它的后一条指令，因为实际的地址是由下一条指令的地址加上 call 后面的地址之和，这里 call 后面的地址为 0，故它们目前均指向下一条指令。

#### (四) 链接器加载器

由汇编程序生成的目标文件不能够直接执行。大型程序经常被分成多个部分进行编译，因此，可重定位的机器代码有必要和其他可重定位的目标文件以及库文件链接到一起，最终形成真正在机器上运行的代码。进而链接器对该机器代码进行执行生成可执行文件。

通过 gcc main\_gcc.o -o main\_gcc 命令、gcc main\_arm.o -o main\_arm 命令和 clang main\_llvm.o -o main\_llvm 命令生成不同中间代码版本的 main 可执行文件。

我们可以修改一下参数实现对链接结果的调整。

- -o <output>: 指定输出文件的名称，例如 -o main 将生成一个名为 main 的可执行文件。
- -L<path>: 指定链接器搜索库文件的路径，例如 -L/usr/lib 将在 /usr/lib 目录下搜索库文件。
- -l<library>: 指定链接器需要链接的库文件，例如 -lm 将链接数学库。
- -I<path>: 指定编译器搜索头文件的路径，例如 -I/usr/include 将在 /usr/include 目录下搜索头文件。
- -static: 静态链接，将所有的库文件都静态链接到可执行文件中，而不是动态链接。
- -Wl,<option>: 将 <option> 传递给链接器，例如 -Wl,-rpath,/usr/local/lib 将设置运行时库搜索路径。
- -g: 生成调试信息，方便调试程序。
- -Wall: 启用所有警告信息。
- -O<level>: 指定优化级别，例如 -O2 进行中级优化。
- -std=<standard>: 指定使用的 C 语言标准，例如 -std=c11 指定使用 C11 标准。

使用 objdump -d main\_gcc 命令、arm-linux-gnueabi-objdump -d main\_arm 命令对上述文件进行反汇编。由于内容所限，此处仅对 x86 反汇编进行展示。

##### x86 反编译结果

```

1 main_gcc:      文件格式 elf64-x86-64
2

```

```

3
4 Disassembly of section .init:
5
6 0000000000001000 <_init>:
7     1000:      f3 0f 1e fa      endbr64
8     1004:      48 83 ec 08      sub    $0x8,%rsp
9     1008:      48 8b 05 d9 2f 00 00  mov    0x2fd9(%rip),%rax      # 3
        fe8 <__gmon_start__@Base>
10    100f:      48 85 c0          test   %rax,%rax
11    1012:      74 02            je     1016 <_init+0x16>
12    1014:      ff d0          call   *%rax
13    1016:      48 83 c4 08      add    $0x8,%rsp
14    101a:      c3              ret
15
16 Disassembly of section .plt:
17
18 0000000000001020 <.plt>:
19    1020:      ff 35 7a 2f 00 00      push   0x2f7a(%rip)      # 3fa0 <
        _GLOBAL_OFFSET_TABLE_+0x8>
20    1026:      f2 ff 25 7b 2f 00 00  bnd jmp *0x2f7b(%rip)      # 3fa8 <
        _GLOBAL_OFFSET_TABLE_+0x10>
21    102d:      0f 1f 00          nopl   (%rax)
22    1030:      f3 0f 1e fa      endbr64
23    1034:      68 00 00 00 00      push   $0x0
24    1039:      f2 e9 e1 ff ff ff  bnd jmp 1020 <_init+0x20>
25    103f:      90              nop
26    1040:      f3 0f 1e fa      endbr64
27    1044:      68 01 00 00 00      push   $0x1
28    1049:      f2 e9 d1 ff ff ff  bnd jmp 1020 <_init+0x20>
29    104f:      90              nop
30    1050:      f3 0f 1e fa      endbr64
31    1054:      68 02 00 00 00      push   $0x2
32    1059:      f2 e9 c1 ff ff ff  bnd jmp 1020 <_init+0x20>
33    105f:      90              nop
34    1060:      f3 0f 1e fa      endbr64
35    1064:      68 03 00 00 00      push   $0x3
36    1069:      f2 e9 b1 ff ff ff  bnd jmp 1020 <_init+0x20>
37    106f:      90              nop
38    1070:      f3 0f 1e fa      endbr64
39    1074:      68 04 00 00 00      push   $0x4
40    1079:      f2 e9 a1 ff ff ff  bnd jmp 1020 <_init+0x20>
41    107f:      90              nop
42
43 Disassembly of section .plt.got:
44
45 0000000000001080 <__cxa_finalize@plt>:
46    1080:      f3 0f 1e fa      endbr64
47    1084:      f2 ff 25 6d 2f 00 00  bnd jmp *0x2f6d(%rip)      # 3ff8 <

```

```

48      __cxa_finalize@GLIBC_2.2.5>
49      108b:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
50  Disassembly of section .plt.sec:
51
52  0000000000001090 <puts@plt>:
53      1090:      f3 0f 1e fa      endbr64
54      1094:      f2 ff 25 15 2f 00 00      bnd jmp *0x2f15(%rip)      # 3fb0 <
      puts@GLIBC_2.2.5>
55      109b:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
56
57  00000000000010a0 <clock@plt>:
58      10a0:      f3 0f 1e fa      endbr64
59      10a4:      f2 ff 25 0d 2f 00 00      bnd jmp *0x2f0d(%rip)      # 3fb8 <
      clock@GLIBC_2.2.5>
60      10ab:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
61
62  00000000000010b0 <__stack_chk_fail@plt>:
63      10b0:      f3 0f 1e fa      endbr64
64      10b4:      f2 ff 25 05 2f 00 00      bnd jmp *0x2f05(%rip)      # 3fc0 <
      __stack_chk_fail@GLIBC_2.4>
65      10bb:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
66
67  00000000000010c0 <printf@plt>:
68      10c0:      f3 0f 1e fa      endbr64
69      10c4:      f2 ff 25 fd 2e 00 00      bnd jmp *0x2efd(%rip)      # 3fc8 <
      printf@GLIBC_2.2.5>
70      10cb:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
71
72  00000000000010d0 <__isoc99_scanf@plt>:
73      10d0:      f3 0f 1e fa      endbr64
74      10d4:      f2 ff 25 f5 2e 00 00      bnd jmp *0x2ef5(%rip)      # 3fd0 <
      __isoc99_scanf@GLIBC_2.7>
75      10db:      0f 1f 44 00 00      nopl    0x0(%rax,%rax,1)
76
77  Disassembly of section .text:
78
79  00000000000010e0 <_start>:
80      10e0:      f3 0f 1e fa      endbr64
81      10e4:      31 ed      xor     %ebp,%ebp
82      10e6:      49 89 d1      mov     %rdx,%r9
83      10e9:      5e      pop     %rsi
84      10ea:      48 89 e2      mov     %rsp,%rdx
85      10ed:      48 83 e4 f0      and     $0xfffffffffffffff0,%rsp
86      10f1:      50      push    %rax
87      10f2:      54      push    %rsp
88      10f3:      45 31 c0      xor     %r8d,%r8d
89      10f6:      31 c9      xor     %ecx,%ecx

```

```

90 10f8:      48 8d 3d ca 00 00 00    lea    0xca(%rip),%rdi      # 11c9
    <main>
91 10ff:      ff 15 d3 2e 00 00    call   *0x2ed3(%rip)      # 3fd8 <
    __libc_start_main@GLIBC_2.34>
92 1105:      f4                      hlt
93 1106:      66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
94 110d:      00 00 00
95
96 0000000000001110 <deregister_tm_clones>:
97 1110:      48 8d 3d f9 2e 00 00    lea    0x2ef9(%rip),%rdi    #
    4010 <__TMC_END__>
98 1117:      48 8d 05 f2 2e 00 00    lea    0x2ef2(%rip),%rax    #
    4010 <__TMC_END__>
99 111e:      48 39 f8                  cmp     %rdi,%rax
100 1121:      74 15                     je      1138 <deregister_tm_clones+0
    x28>
101 1123:      48 8b 05 b6 2e 00 00    mov     0x2eb6(%rip),%rax    # 3
    fe0 <__ITM_deregisterTMCloneTable@Base>
102 112a:      48 85 c0                  test    %rax,%rax
103 112d:      74 09                     je      1138 <deregister_tm_clones+0
    x28>
104 112f:      ff e0                     jmp     *%rax
105 1131:      0f 1f 80 00 00 00 00    nopl    0x0(%rax)
106 1138:      c3                      ret
107 1139:      0f 1f 80 00 00 00 00    nopl    0x0(%rax)
108
109 0000000000001140 <register_tm_clones>:
110 1140:      48 8d 3d c9 2e 00 00    lea    0x2ec9(%rip),%rdi    #
    4010 <__TMC_END__>
111 1147:      48 8d 35 c2 2e 00 00    lea    0x2ec2(%rip),%rsi    #
    4010 <__TMC_END__>
112 114e:      48 29 fe                  sub     %rdi,%rsi
113 1151:      48 89 f0                  mov     %rsi,%rax
114 1154:      48 c1 ee 3f              shr     $0x3f,%rsi
115 1158:      48 c1 f8 03              sar     $0x3,%rax
116 115c:      48 01 c6                  add     %rax,%rsi
117 115f:      48 d1 fe                  sar     %rsi
118 1162:      74 14                     je      1178 <register_tm_clones+0x38>
119 1164:      48 8b 05 85 2e 00 00    mov     0x2e85(%rip),%rax    # 3
    ff0 <__ITM_registerTMCloneTable@Base>
120 116b:      48 85 c0                  test    %rax,%rax
121 116e:      74 08                     je      1178 <register_tm_clones+0x38>
122 1170:      ff e0                     jmp     *%rax
123 1172:      66 0f 1f 44 00 00        nopw    0x0(%rax,%rax,1)
124 1178:      c3                      ret
125 1179:      0f 1f 80 00 00 00 00    nopl    0x0(%rax)
126
127 0000000000001180 <__do_global_dtors_aux>:

```

```

128      1180:      f3 0f 1e fa      endbr64
129      1184:      80 3d 85 2e 00 00 00      cmpb    $0x0,0x2e85(%rip)      #
      4010 <__TMC_END__>
130      118b:      75 2b              jne     11b8 <__do_global_dtors_aux+0
      x38>
131      118d:      55                  push    %rbp
132      118e:      48 83 3d 62 2e 00 00      cmpq    $0x0,0x2e62(%rip)      # 3
      ff8 <__cxa_finalize@GLIBC_2.2.5>
133      1195:      00
134      1196:      48 89 e5              mov     %rsp,%rbp
135      1199:      74 0c              je      11a7 <__do_global_dtors_aux+0
      x27>
136      119b:      48 8b 3d 66 2e 00 00      mov     0x2e66(%rip),%rdi      #
      4008 <__dso_handle>
137      11a2:      e8 d9 fe ff ff      call    1080 <__cxa_finalize@plt>
138      11a7:      e8 64 ff ff ff      call    1110 <deregister_tm_clones>
139      11ac:      c6 05 5d 2e 00 00 01      movb    $0x1,0x2e5d(%rip)      #
      4010 <__TMC_END__>
140      11b3:      5d                  pop     %rbp
141      11b4:      c3                  ret
142      11b5:      0f 1f 00            nopl    (%rax)
143      11b8:      c3                  ret
144      11b9:      0f 1f 80 00 00 00 00      nopl    0x0(%rax)
145
146      00000000000011c0 <frame_dummy>:
147      11c0:      f3 0f 1e fa      endbr64
148      11c4:      e9 77 ff ff ff      jmp     1140 <register_tm_clones>
149
150      00000000000011c9 <main>:
151      11c9:      f3 0f 1e fa      endbr64
152      11cd:      55                  push    %rbp
153      11ce:      48 89 e5              mov     %rsp,%rbp
154      11d1:      48 83 ec 30          sub     $0x30,%rsp
155      11d5:      64 48 8b 04 25 28 00      mov     %fs:0x28,%rax
156      11dc:      00 00
157      11de:      48 89 45 f8          mov     %rax,-0x8(%rbp)
158      11e2:      31 c0                xor     %eax,%eax
159      11e4:      48 8d 45 d4          lea     -0x2c(%rbp),%rax
160      11e8:      48 89 c6              mov     %rax,%rsi
161      11eb:      48 8d 05 16 0e 00 00      lea     0xe16(%rip),%rax      # 2008
      <_IO_stdin_used+0x8>
162      11f2:      48 89 c7              mov     %rax,%rdi
163      11f5:      b8 00 00 00 00      mov     $0x0,%eax
164      11fa:      e8 d1 fe ff ff      call    10d0 <__isoc99_scanf@plt>
165      11ff:      c7 45 d8 02 00 00 00      movl    $0x2,-0x28(%rbp)
166      1206:      c7 45 dc 01 00 00 00      movl    $0x1,-0x24(%rbp)
167      120d:      e8 8e fe ff ff      call    10a0 <clock@plt>
168      1212:      48 89 45 e0          mov     %rax,-0x20(%rbp)

```

```

169 1216:      83 7d d8 00      cmpl    $0x0,-0x28(%rbp)
170 121a:      7f 1f                jg      123b <main+0x72>
171 121c:      48 8d 05 e8 0d 00 00    lea     0xde8(%rip),%rax      # 200b
    <_IO_stdin_used+0xb>
172 1223:      48 89 c7                mov     %rax,%rdi
173 1226:      e8 65 fe ff ff          call    1090 <puts@plt>
174 122b:      eb 0e                jmp     123b <main+0x72>
175 122d:      8b 45 dc                mov     -0x24(%rbp),%eax
176 1230:      0f af 45 d8              imul    -0x28(%rbp),%eax
177 1234:      89 45 dc                mov     %eax,-0x24(%rbp)
178 1237:      83 45 d8 01             addl    $0x1,-0x28(%rbp)
179 123b:      8b 45 d4                mov     -0x2c(%rbp),%eax
180 123e:      39 45 d8                cmp     %eax,-0x28(%rbp)
181 1241:      7e ea                jle     122d <main+0x64>
182 1243:      81 7d dc 10 27 00 00     cmpl    $0x2710,-0x24(%rbp)
183 124a:      7e 11                jle     125d <main+0x94>
184 124c:      48 8d 05 c1 0d 00 00     lea     0xdc1(%rip),%rax      # 2014
    <_IO_stdin_used+0x14>
185 1253:      48 89 c7                mov     %rax,%rdi
186 1256:      e8 35 fe ff ff          call    1090 <puts@plt>
187 125b:      eb 19                jmp     1276 <main+0xad>
188 125d:      8b 45 dc                mov     -0x24(%rbp),%eax
189 1260:      89 c6                mov     %eax,%esi
190 1262:      48 8d 05 b8 0d 00 00     lea     0xdb8(%rip),%rax      # 2021
    <_IO_stdin_used+0x21>
191 1269:      48 89 c7                mov     %rax,%rdi
192 126c:      b8 00 00 00 00          mov     $0x0,%eax
193 1271:      e8 4a fe ff ff          call    10c0 <printf@plt>
194 1276:      e8 25 fe ff ff          call    10a0 <clock@plt>
195 127b:      48 89 45 e8              mov     %rax,-0x18(%rbp)
196 127f:      48 8b 45 e8              mov     -0x18(%rbp),%rax
197 1283:      48 2b 45 e0              sub     -0x20(%rbp),%rax
198 1287:      66 0f ef c0              pxor    %xmm0,%xmm0
199 128b:      f2 48 0f 2a c0          cvtsi2sd %rax,%xmm0
200 1290:      f2 0f 10 0d b0 0d 00     movsd   0xdb0(%rip),%xmm1    #
    2048 <_IO_stdin_used+0x48>
201 1297:      00
202 1298:      f2 0f 5e c1              divsd   %xmm1,%xmm0
203 129c:      f2 0f 11 45 f0          movsd   %xmm0,-0x10(%rbp)
204 12a1:      48 8b 45 f0              mov     -0x10(%rbp),%rax
205 12a5:      66 48 0f 6e c0          movq    %rax,%xmm0
206 12aa:      48 8d 05 74 0d 00 00     lea     0xd74(%rip),%rax      # 2025
    <_IO_stdin_used+0x25>
207 12b1:      48 89 c7                mov     %rax,%rdi
208 12b4:      b8 01 00 00 00          mov     $0x1,%eax
209 12b9:      e8 02 fe ff ff          call    10c0 <printf@plt>
210 12be:      b8 00 00 00 00          mov     $0x0,%eax
211 12c3:      48 8b 55 f8              mov     -0x8(%rbp),%rdx

```



```

212 12c7:      64 48 2b 14 25 28 00    sub    %fs:0x28,%rdx
213 12ce:      00 00
214 12d0:      74 05                je     12d7 <main+0x10e>
215 12d2:      e8 d9 fd ff ff        call   10b0 <__stack_chk_fail@plt>
216 12d7:      c9                    leave
217 12d8:      c3                    ret
218
219 Disassembly of section .fini:
220
221 00000000000012dc <_fini>:
222 12dc:      f3 0f 1e fa                endbr64
223 12e0:      48 83 ec 08                sub    $0x8,%rsp
224 12e4:      48 83 c4 08                add    $0x8,%rsp
225 12e8:      c3                    ret

```

将其与上面的反编译代码进行比较，可以发现填补了之前一些地址的空白，还有其他库文件的反编译代码，从而验证了链接器的功能。

### (五) 执行

在终端中输入./main 即可执行该程序。

```

lxmliu2002@lxmliu2002-Ubuntu:~/桌面/lab1$ ./main
5
120
程序运行时间为：0.000039

```

图 6: main 程序运行结果

### (六) LLVM IR 编程

经过查询可知，LLVM IR 的语法规则如下：

- 模块 (Module)

```

1 ; ModuleID = 'example'
2 source_filename = "example"
3 @global_var = global i32 10
4 define i32 @main() {
5     ; function body
6     ret i32 0
7 }

```

- 全局变量 (Global Variable) : 以 @ 开头

```

1 @global_var = global i32 10
2 @global_array = global [10 x i32] zeroinitializer

```

- 函数 (Function) : 以 define 关键字开头

```

1  define i32 @add(i32 %a, i32 %b) {
2      ; function body
3      %sum = add i32 %a, %b
4      ret i32 %sum
5  }

```

- 基本块 (Basic Block) : 以 Label 开头, 以终止指令 Terminator Instruction 结束

```

1  entry:
2      ; instructions
3      %sum = add i32 %a, %b
4      br label %exit
5  exit:
6      ; instructions
7      ret i32 %sum

```

- 指令 (Instruction)

```

1  %sum = add i32 %a, %b
2  %result = mul i32 %sum, 2
3  %cmp = icmp eq i32 %result, 10
4  br i1 %cmp, label %true, label %false

```

- 标签 (Label): 以% 开头

```

1  entry:
2      ; instructions
3      br label %exit

```

基于以上语法规则, 我们小组尝试编写的 LLVM IR 程序如下:

#### LLVM IR 中间代码编写

```

1  ; ModuleID = 'test.c'
2  source_filename = "test.c"
3  target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8
4      :16:32:64-S128"
5  target triple = "x86_64-pc-linux-gnu"
6
7  @.str = private unnamed_addr constant [13 x i8] c"Hello World\0A\00", align 1
8
9  ; Function Attrs: noinline nounwind optnone uwtable
10 define dso_local i32 @main() #0 {
11     %1 = alloca i32, align 4
12     store i32 1, i32* %1, align 4
13     %2 = load i32, i32* %1, align 4
14     %3 = icmp ne i32 %2, 0
15     br i1 %3, label %4, label %6

```

```

16 4:
17   %5 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([13 x
      i8], [13 x i8]* @.str, i64 0, i64 0))
18   br label %6
19
20 6:
21   ret i32 0
22 }
23
24 declare i32 @printf(i8* noundef, ...) #1
25
26 attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="all" "
      min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-
      buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx
      ,+sse,+sse2,+x87" "tune-cpu"="generic" }
27 attributes #1 = { "frame-pointer"="all" "no-trapping-math"="true" "stack-
      protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8
      ,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
28
29 !llvm.module.flags = !{!0, !1, !2, !3, !4}
30 !llvm.ident = !{!5}
31
32 !0 = !{i32 1, !"wchar_size", i32 4}
33 !1 = !{i32 7, !"PIC Level", i32 2}
34 !2 = !{i32 7, !"PIE Level", i32 2}
35 !3 = !{i32 7, !"uwtable", i32 1}
36 !4 = !{i32 7, !"frame-pointer", i32 2}
37 !5 = !{!"Ubuntu clang version 14.0.0-1ubuntu1.1"}

```

编译成可执行文件后运行结果如下：



```

zzx@zzx:~/文档/编译原理$ clang test.c
zzx@zzx:~/文档/编译原理$ ./test
Hello World
zzx@zzx:~/文档/编译原理$

```

图 7: LLVM IR 程序运行结果

由上面运行结果可以看到，编写的 LLVM IR 程序的正确性得到了验证。

## 四、 总结

通过本次实验，我们依次完成了预处理、编译、汇编、链接、加载等过程，也对 LLVM IR 中间过程进行分析和编写，使用 gcc 生成控制流图，熟悉了每个中间步骤实现的操作以及产生的文件。也对不同优化情况下的代码对应的汇编结果进行比较，了解了基本的代码优化方向。同时也对 makefile 文件有了一定的了解，可以一定程度上免去重复工作带来的效率低下。对如何构建一个编译器有了初步的认识，这为我们后面实现一个编译器有很大的帮助。

NIJU