认识编译器-GCC相关操作练习

相关代码

```
.
├── doc
│   └── answer.md
├── makefile
├── src
│   └── sample.c
└── target
```

```
#makefile
CC := gcc
NAME := sample
MACRO := NEG

pre:
	@$(CC) -E src/$(NAME).c -o target/$(NAME).i

pre-define-macro:
	@$(CC) -E -D $(MACRO) src/$(NAME).c -o target/$(NAME).i

asm:
	@$(CC) -S -m$(M) src/$(NAME).c -o target/$(NAME).s

obj: target/$(NAME).o

target/$(NAME).o:
	@$(CC) -c src/$(NAME).c -o target/$(NAME).o

disasm: target/$(NAME).o
	@objdump -dS target/$(NAME).o

print-symbol: target/$(NAME).o
	@nm target/$(NAME).o

build: target/$(NAME).o
	@ld /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtn.o target/$(NAME).o -lc -o
target/$(NAME)

clean:
	@rm target/*
```

```
//src/sample.c
#ifdef NEG
#define M -4
#else
#define M 4
#endif /* ifdef NEG */

int main(int argc, char *argv[]) {
  int a = M;
  if (a)
    a = a + 4;
  else
    a = a * 4;
  return 0;
}
```

编译过程

- 宏展开

```
❯ make pre
❯ cat target/sample.i
# 0 "src/sample.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "src/sample.c"






int main(int argc, char *argv[]) {
  int a = 4;
  if (a)
    a = a + 4;
  else
    a = a * 4;
  return 0;
}
```

- 问题1-1

gcc的`-D`选项用来定义宏

用法 `gcc [-D <macro-name>]`

```
❯ make pre-define-macro
❯ cat target/sample.i
# 0 "src/sample.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "src/sample.c"






int main(int argc, char *argv[]) {
  int a = -4;
  if (a)
    a = a + 4;
  else
    a = a * 4;
  return 0;
}
```

跟原来的.i文件相比a初始化为-4

- 编译成汇编形式

```
make asm M=<汇编的位数>
```

如

```
❯ make asm M=64
❯ cat target/sample.s
    .file   "sample.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    %edi, -20(%rbp)
    movq    %rsi, -32(%rbp)
    movl    $4, -4(%rbp)
    cmpl    $0, -4(%rbp)
    je  .L2
    addl    $4, -4(%rbp)
    jmp .L3
.L2:
    sall    $2, -4(%rbp)
.L3:
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (GNU) 13.2.1 20230801"
    .section    .note.GNU-stack,"",@progbits
```

或

```
❯ make asm M=32
❯ cat target/sample.s
    .file   "sample.c"
    .text
    .globl  main
```

```
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	subl	$16, %esp
	call	__x86.get_pc_thunk.ax
	addl	$_GLOBAL_OFFSET_TABLE_, %eax
	movl	$4, -4(%ebp)
	cmpl	$0, -4(%ebp)
	je	.L2
	addl	$4, -4(%ebp)
	jmp	.L3
.L2:
	sall	$2, -4(%ebp)
.L3:
	movl	$0, %eax
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.section
.text.__x86.get_pc_thunk.ax,"axG",@progbits,__x86.get_pc_thunk.ax,comdat
	.globl	__x86.get_pc_thunk.ax
	.hidden	__x86.get_pc_thunk.ax
	.type	__x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
.LFB1:
	.cfi_startproc
	movl	(%esp), %eax
	ret
	.cfi_endproc
.LFE1:
	.ident	"GCC: (GNU) 13.2.1 20230801"
	.section	.note.GNU-stack,"",@progbits
```

- 问题1-2
  - pushq和pushl区别

    b = byte

    s = 2bytes

    w = 2bytes

    l = 4bytes

    q = 8bytes

    都是压栈不过因为目标汇编的位数不一样一个压4字节一个压8字节
  - rsp和esp

    rsp8字节 esp4字节

- 生成目标文件

```
❯ make obj
❯ ls target
sample.i  sample.o  sample.s
```

- 用`nm`导出外部符号

```
❯ make print-symbol
0000000000000000 T main
```

- 用objdump生成反汇编代码

```
❯ make disasm

target/sample.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d ec                mov    %edi,-0x14(%rbp)
   7:   48 89 75 e0             mov    %rsi,-0x20(%rbp)
   b:   c7 45 fc 04 00 00 00    movl   $0x4,-0x4(%rbp)
  12:   83 7d fc 00             cmpl   $0x0,-0x4(%rbp)
  16:   74 06                   je     1e <main+0x1e>
  18:   83 45 fc 04             addl   $0x4,-0x4(%rbp)
  1c:   eb 04                   jmp    22 <main+0x22>
  1e:   c1 65 fc 02             shll   $0x2,-0x4(%rbp)
  22:   b8 00 00 00 00          mov    $0x0,%eax
  27:   5d                      pop    %rbp
  28:   c3                      ret
```

- 链接得到可执行文件

```
❯ make build
❯ ls target
sample  sample.i  sample.o  sample.s
```

不能直接ld因为sample还依赖了`_start`等外部符号