CS5250 Assignment Report

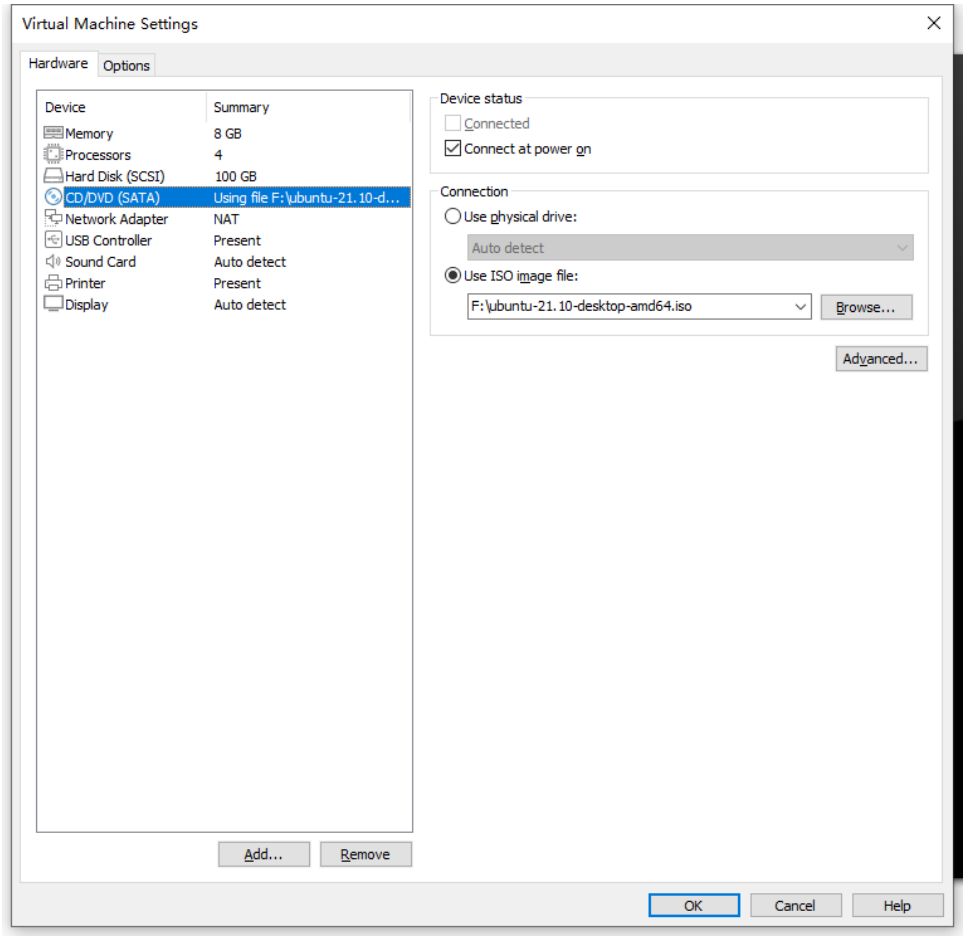
Part A: Linux Kernel Installation (20 marks)

1.1Install Virtualbox on your machine

- Instead of Virtualbox, I choose to use the VMware Workstation Pro 16.

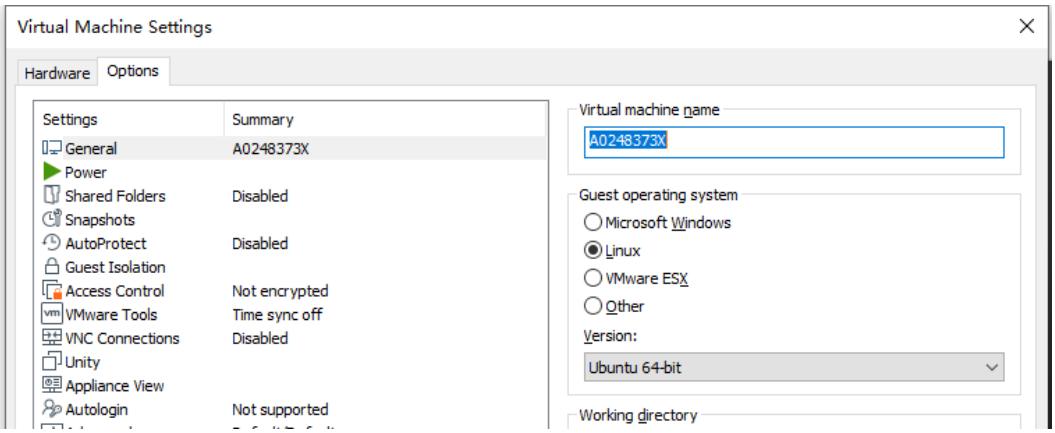
1.2Install a Linux guest machine

1. The version of Linux Ubuntu I use is 21.10 (64bit) version

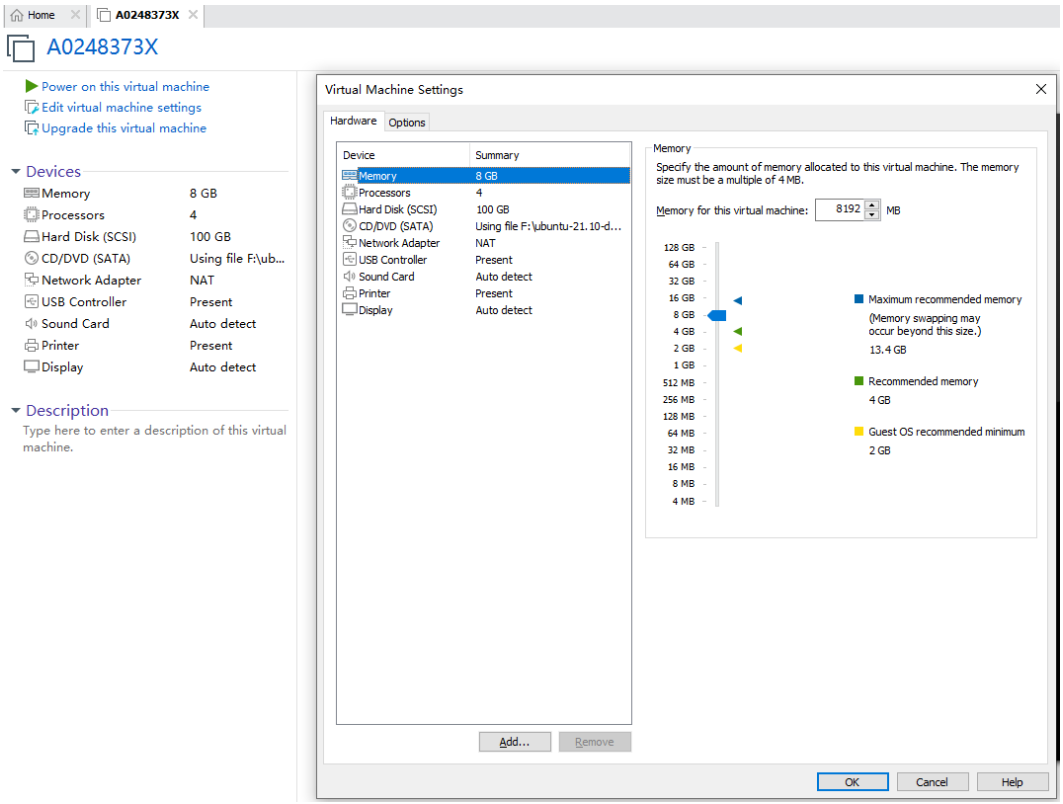


2. Set up a new virtual machine on VMware

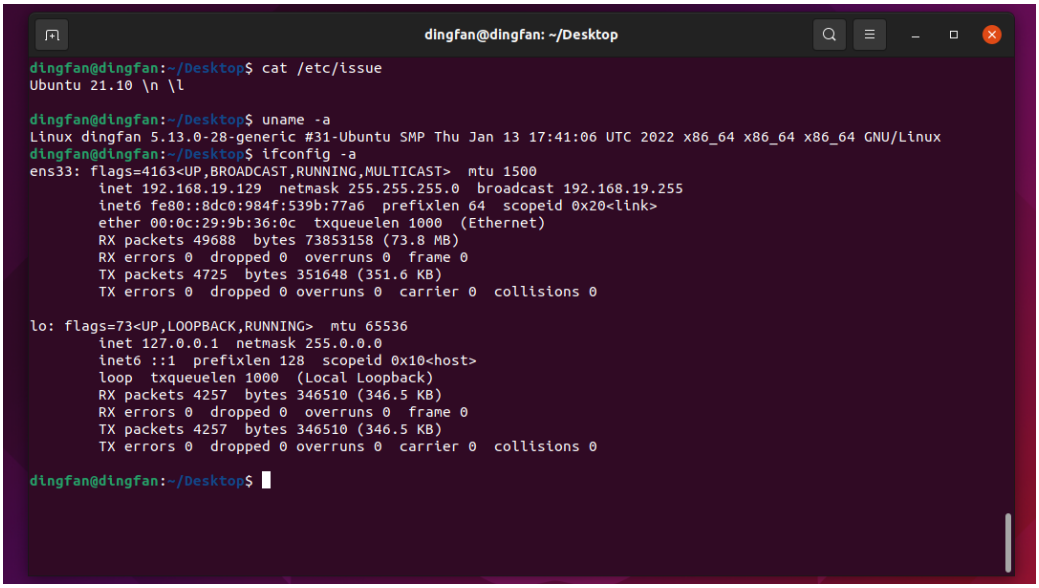
- VM Name and OS Type
  - Name: A0248373X
  - OS Type:
    a. Operating System: Linux
    b. Version: Ubuntu 64bit



- Memory：8096MB
- Virtual Hard Disk: 100GB

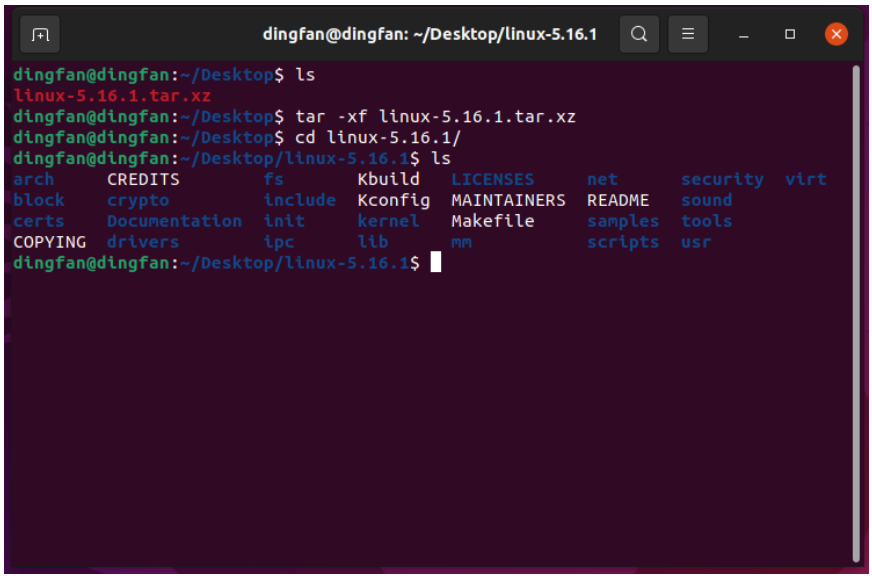


3. Start virtual machine, and install the Ubuntu as guided.
    a. Set username as my full name: dingfan
    b. *Open the terminal in your guest machine, output the OS, the kernel version and also your MAC address and give a screenshot including all the information.*
        - output OS: `cat /etc/issue` Ubuntu 21.10
        - output kernel version: `uname -a` 5.13.0
        - output MAC address: `ifconfig -a` 00:0c:29:9b:36:0c



1.3Build a specific kernel

1. Download linux-5.16.1.tar.xz from (https://cdn.kernel.org/pub/linux/kernel/v5.x/). Unzip the file using the `tar -xf` command, and enter the directory in terminal using `cd` command.



2. When try command `make menuconfig` I see some linux errors. After Google the problem, I figure it out that there are no specific header in my machine. The following is the problems and the related solutions.
    a. run `sudo apt-get install build-essential` to install `gcc` package

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ make menuconfig
Command 'make' not found, but can be installed with:
sudo apt install make        # version 4.3-4ubuntu1, or
sudo apt install make-guile  # version 4.3-4ubuntu1
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install build-essential
```

b. run `sudo apt-get install libncurses-dev` to install `ncurses` package

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ make menuconfig
  HOSTCC  scripts/basic/fixdep
*
* Unable to find the ncurses package.
* Install ncurses (ncurses-devel or libncurses-dev
* depending on your distribution).
*
* You may also need to install pkg-config to find the
* ncurses installed in a non-default location.
*
make[1]: *** [scripts/kconfig/Makefile:211: scripts/kconfig/mconf-cfg] Error 1
make: *** [Makefile:619: menuconfig] Error 2
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install libncurses-dev
```

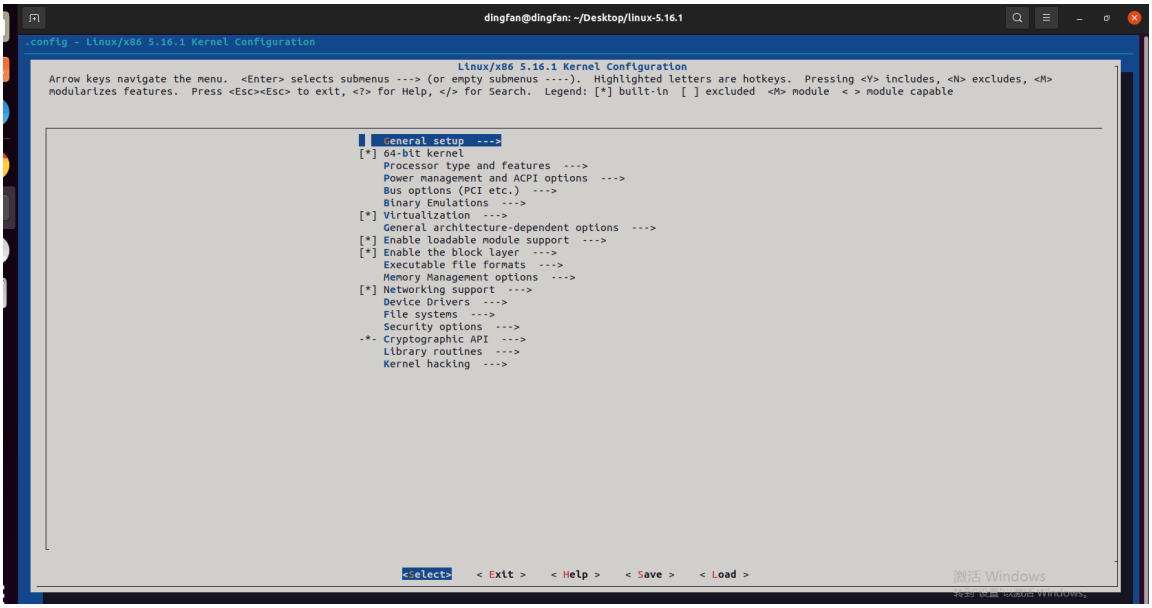
c. run `sudo apt-get install flex` to install `flex` package

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ make menuconfig
  UPD     scripts/kconfig/mconf-cfg
  HOSTCC  scripts/kconfig/mconf.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
  HOSTCC  scripts/kconfig/lxdialog/yesno.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
/bin/sh: 1: flex: not found
make[1]: *** [scripts/Makefile.host:9: scripts/kconfig/lexer.lex.c] Error 127
make: *** [Makefile:619: menuconfig] Error 2
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install flex
```

d. run `sudo apt-get install bison` to install `bison` package

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ make menuconfig
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
/bin/sh: 1: bison: not found
make[1]: *** [scripts/Makefile.host:17: scripts/kconfig/parser.tab.h] Error 127
make: *** [Makefile:619: menuconfig] Error 2
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install bison
Reading package lists... Done
```

3. When successfully run command `make menuconfig` , I can see the kernel configuration menu as follow:



> 💡 During the “make menuconfig” stage, there are three different choices, built-in, excluded
> and module. (i) What do they mean? (ii) Which are the ones that will appear in the kernel
> image?

i. (i) What do they mean?

- By google, I know that this `menuconfig` is used to generate a “.config ” file. This “.config ” file is file is a configuration file for compiling the linux kernel.
- the choice `[*] built in` means that, the feature will be compiled into the new linux kernel “vmlinuz”
- the choice `[ ] excluded` means that the feature will not be compiled
- the choice `<M> module` means that the feature will be compiled as a module, it can be dynamically loaded into the memory when needed

ii. Which are the ones that will appear in the kernel image?

- the features that with `[*] built in` choice will appear in the kernel image
- the features that with `<M> module` choice are not in kernel image, they are in the source directory of each module.

4. run `make` or `sudo make -j8` to compile the kernel

- run `sudo make -j8` meet some errors, and I resolve these problems by google. The problems and solutions are:

a. run `apt-get install libdw-dev` to install the `libelf-dev`

```
  YACC    scripts/genksyms/parse.tab.[ch]
  MKDIR   /home/dingfan/Desktop/linux-5.16.1/tools/bpf/resolve_btfids/libbpf/
<stdin>:1:10: fatal error: libelf.h: No such file or directory
compilation terminated.
  HOSTCC  /home/dingfan/Desktop/linux-5.16.1/tools/objtool/fixdep.o
```

```
                 from weak.c:10:
/home/dingfan/Desktop/linux-5.16.1/tools/objtool/include/objtool/elf.h:10:10: fatal error: gelf.h: No such file or directory
   10 | #include <gelf.h>
      |          ^~~~~~~~
compilation terminated.
make[3]: *** [/home/dingfan/Desktop/linux-5.16.1/tools/build/Makefile.build:97: /home/dingfan/Desktop/linux-5.16.1/tools/objtool/weak.o] Error 1
```

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install libdw-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

b. run `sudo apt-get install libssl-dev` to install the `openssl`

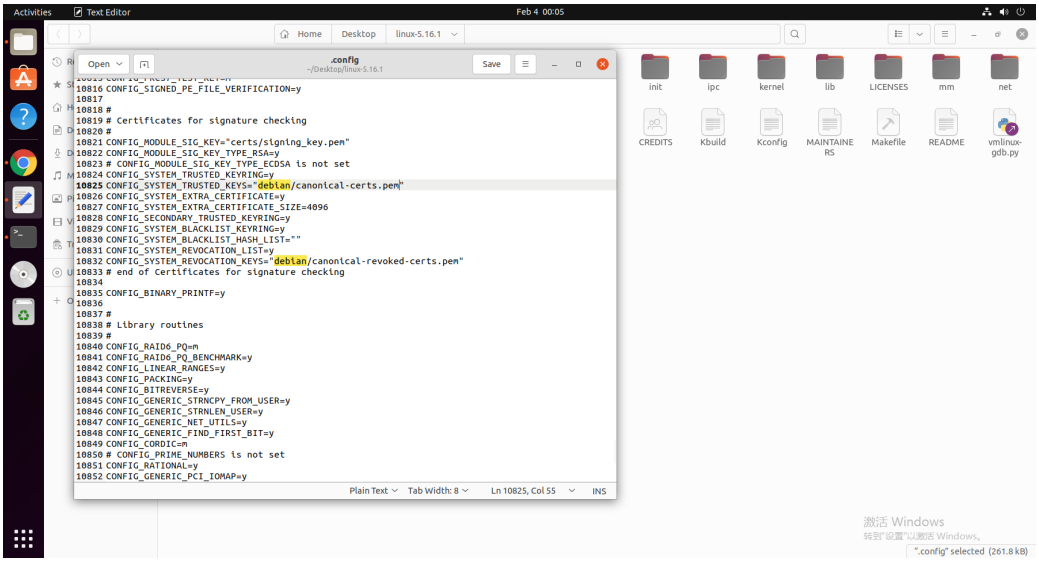
```
  HOSTCC  scripts/sign-file
scripts/sign-file.c:25:10: fatal error: openssl/opensslv.h: No such file or directory
   25 | #include <openssl/opensslv.h>
      |          ^~~~~~~~~~~~~~~~~~~~
compilation terminated.
make[1]: *** [scripts/Makefile.host:95: scripts/sign-file] Error 1
```

```
dingfan@dingfan:~/Desktop/linux-5.16.1$ sudo apt-get install libssl-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

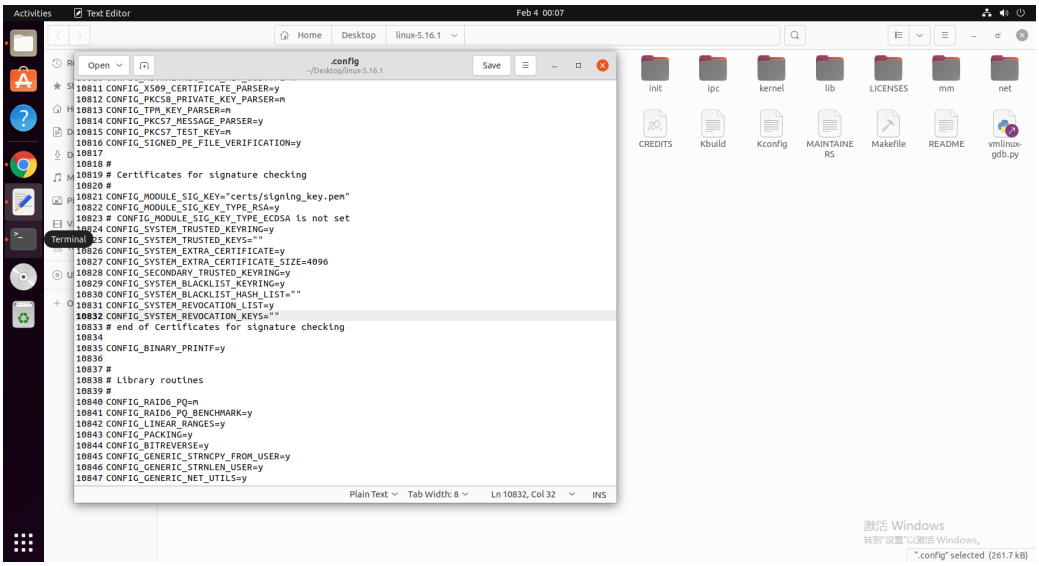
c. After google, I know the following bug can be resolved by modified the ".config" file:

```
  AS      arch/x86/entry/entry_64.o
  AR      init/built-in.a
make[1]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'.  Stop.
make[1]: *** Waiting for unfinished jobs....
  CC      certs/system_keyring.o
```

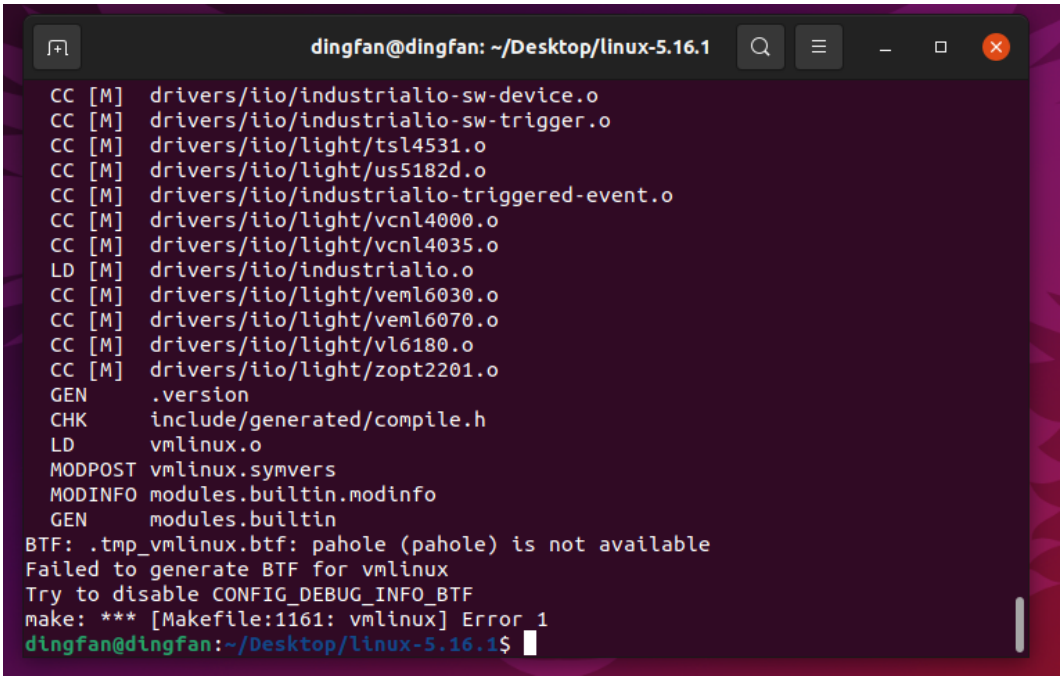
- set `CONFIG_SYSTEM_TRUSTED_KEYS = ""`
- set `CONFIG_SYSTEM_REVOCATION_KEYS = ""`
- before modify



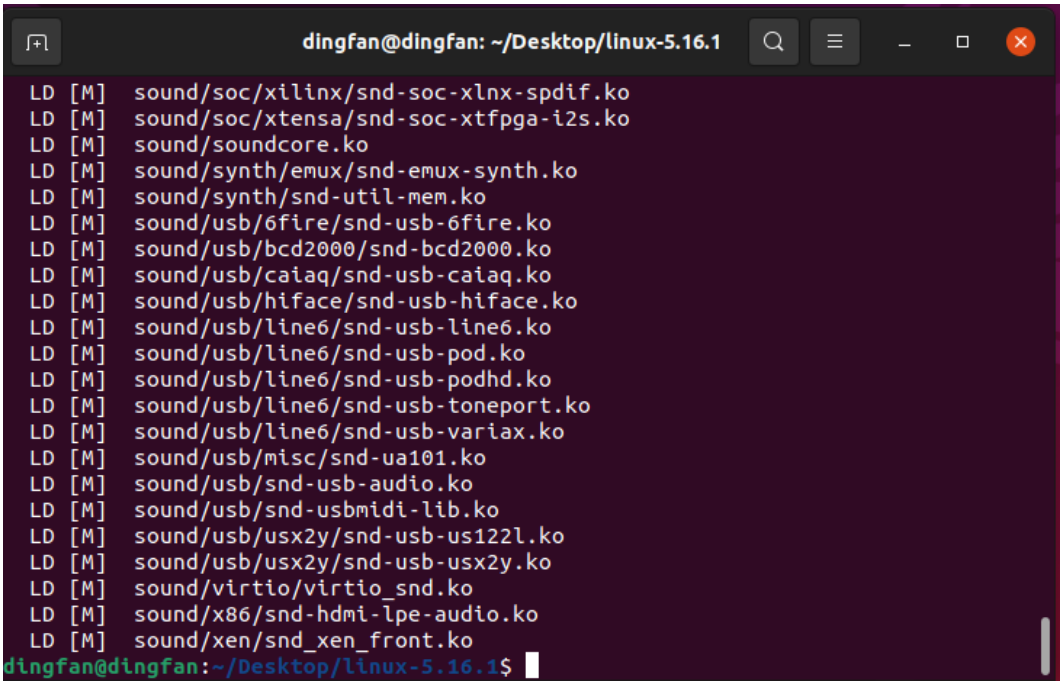
- after modify



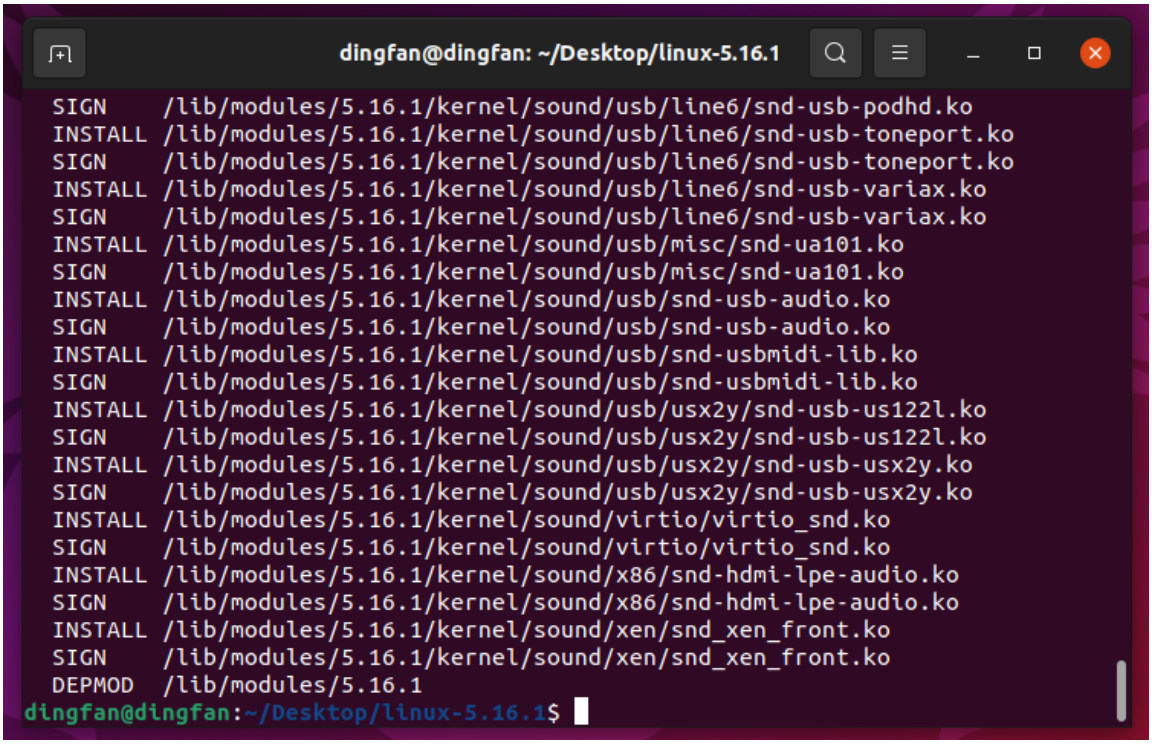
d. run `sudo apt-get install dwarves` to install `dwarves`



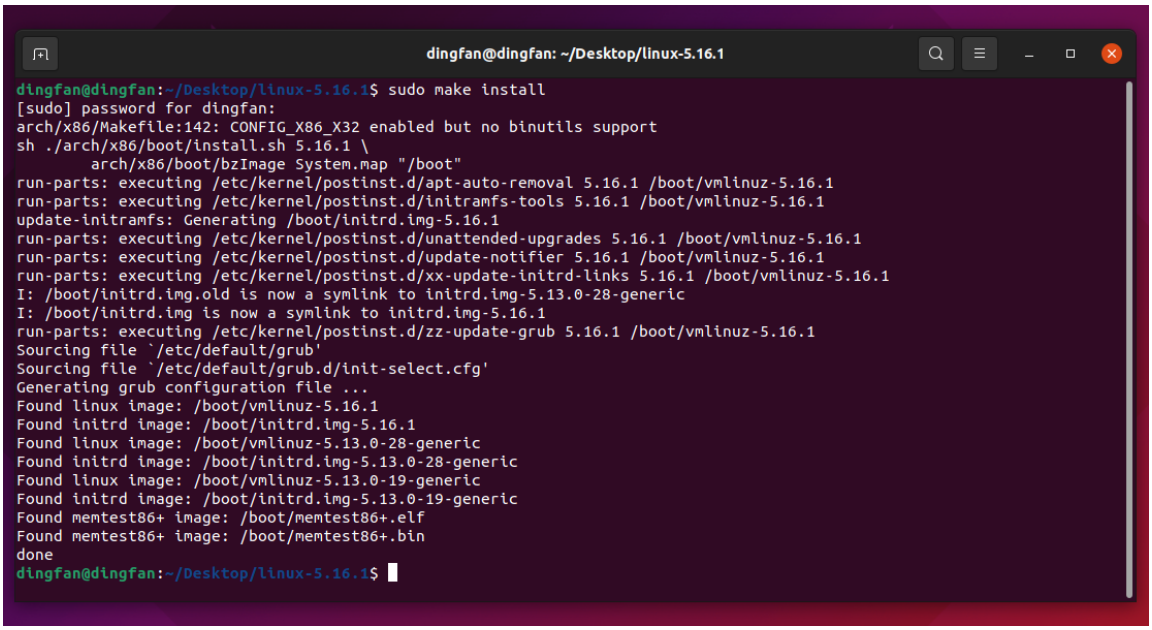
e. run `sudo make -j8` successfully



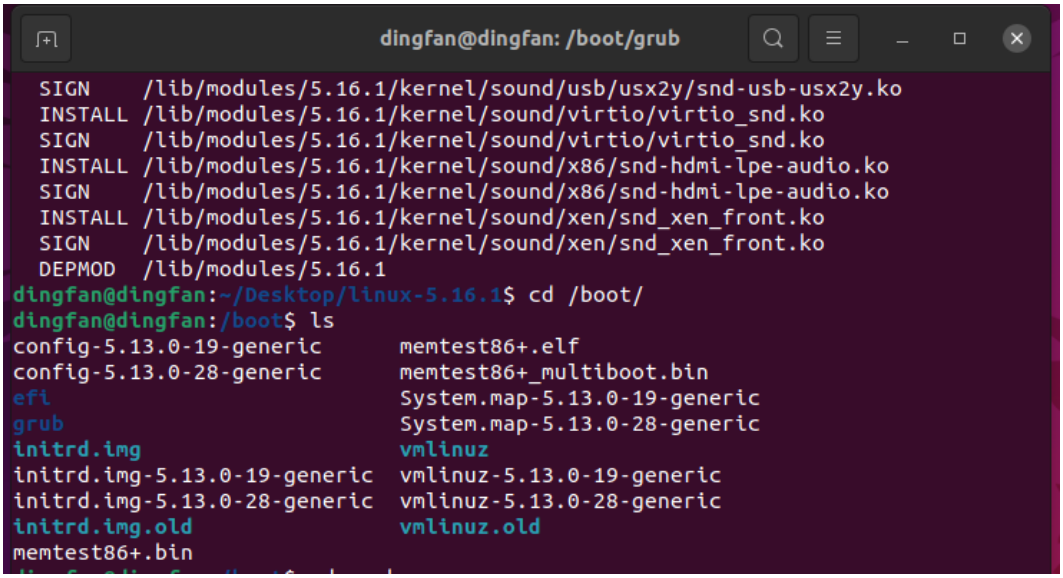
5. run `make modules_install`



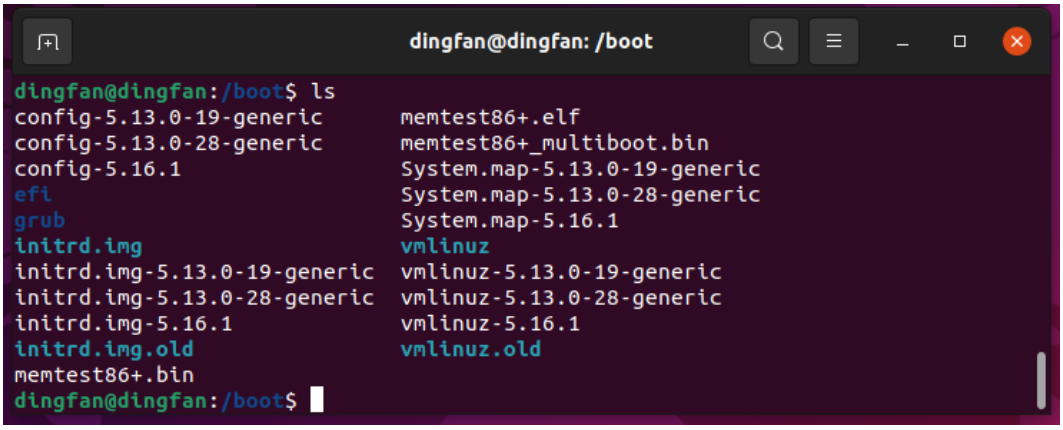
6. run `sudo make install`



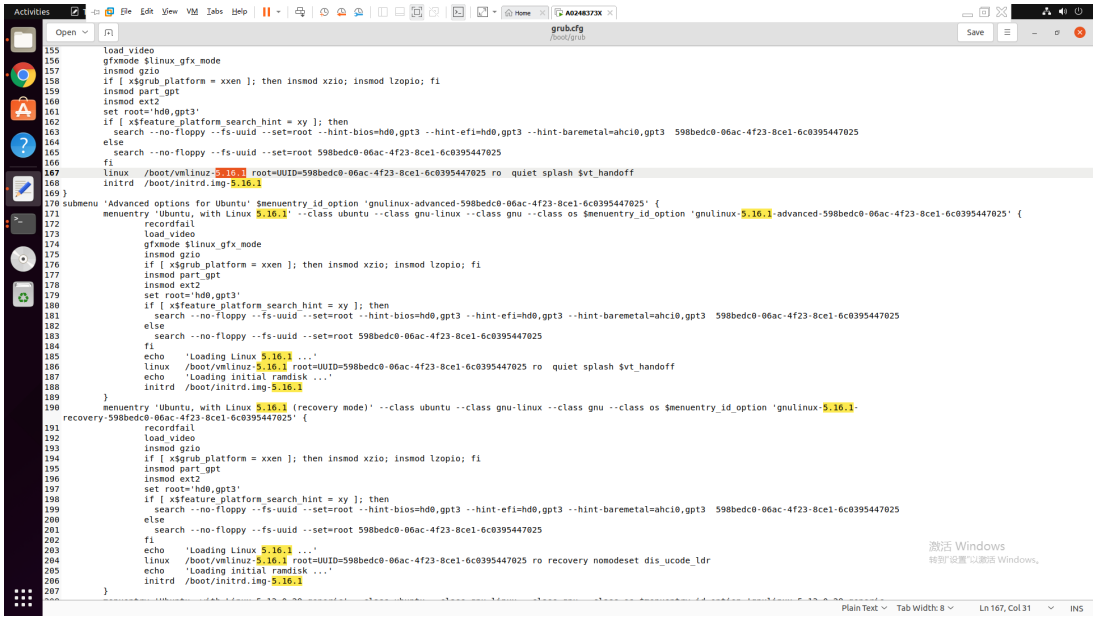
- we can `cd` the `/boot/` directory to check whether the new kernel is in the right place
- before run command `sudo make install`, I cannot see the new kernel



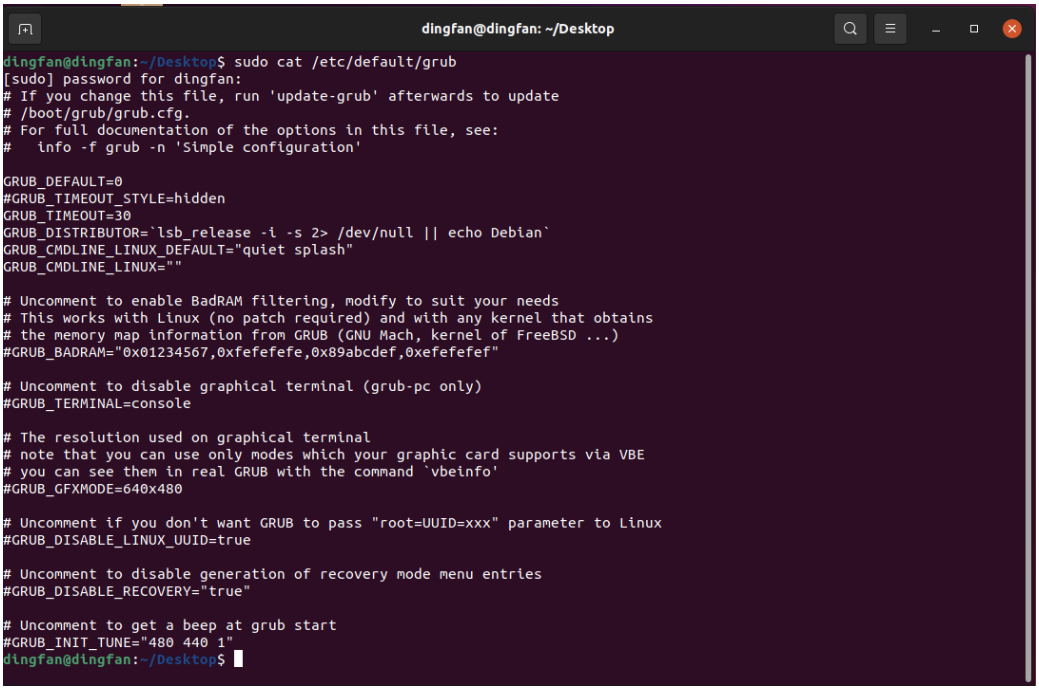
- after run command `sudo make install`, I can see the new kernel with version "5.16.1"



7. Set up bootloader
   - The order of OS booting is in `/boot/grub/grub.cfg`
   - open this file we can see, it seems that the `sudo make install` has helped me set the "5.16.1" kernel as the default option

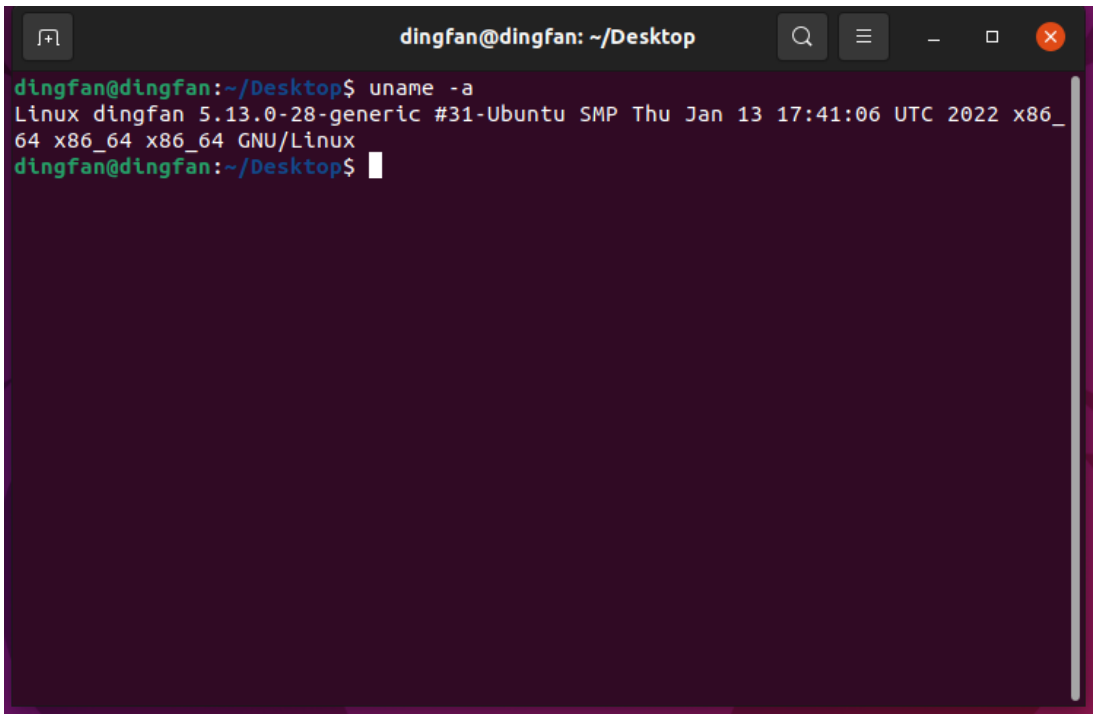


- I also modify the file `/etc/default/grub`
  - set the `GRUB_TIMEOUT_STYLE= hidden` to `#GRUB_TIMEOUT_STYLE= hidden` . Therefore, when the OS reboot, it will show the grub menu and I can choose the specific kernel manually to start up.
  - set `GRUB_TIMEOUT=30`, the machine will stay in the grub menu for 30 seconds.
  - the following picture is the modified version of `/etc/default/grub`

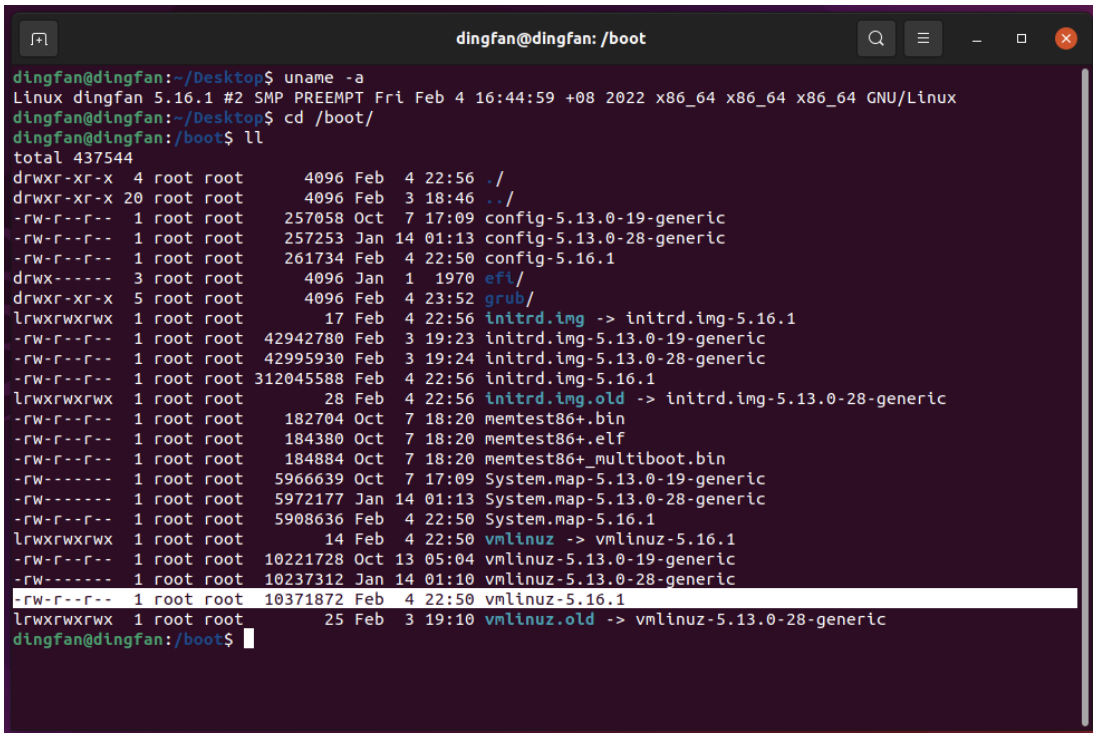


8. Reboot the virtual machine

a. before reboot my kernel version is 5.13.0

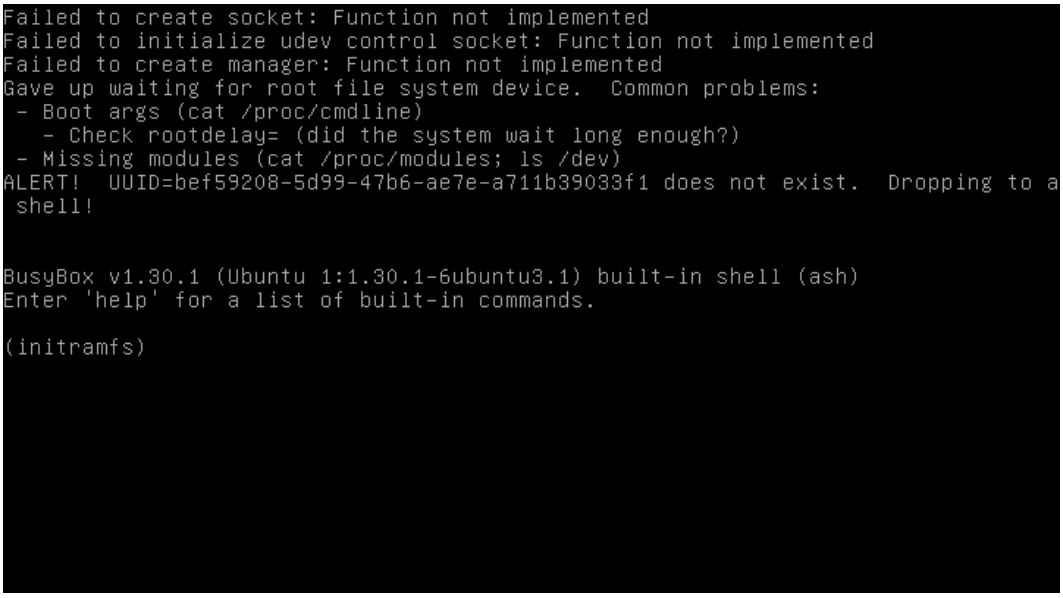


b. after reboot my kernel version is 5.16.1

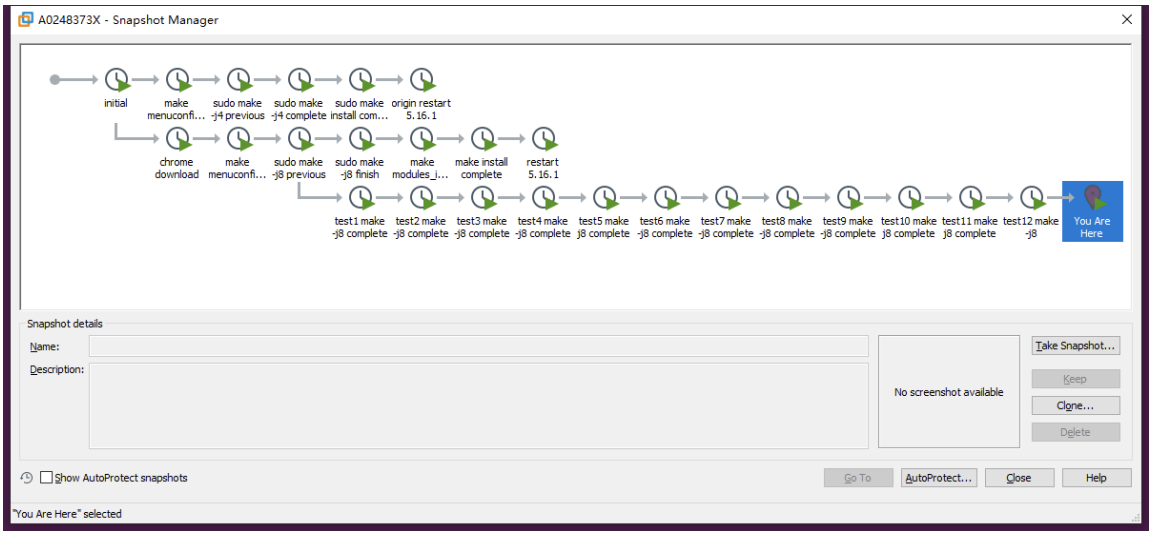


1.4 build smallest kernel

1. In this part, I try to excluded some features base on the default configuration of version “5.16.1” .
2. In this process, I have encountered failures several times. Because I excluded the important features sometimes, and the kernel cannot boot up successfully.
3. One of the failure is that I excluded many features in the “networking support”, that the system cannot reboot successfully as the following screenshot:

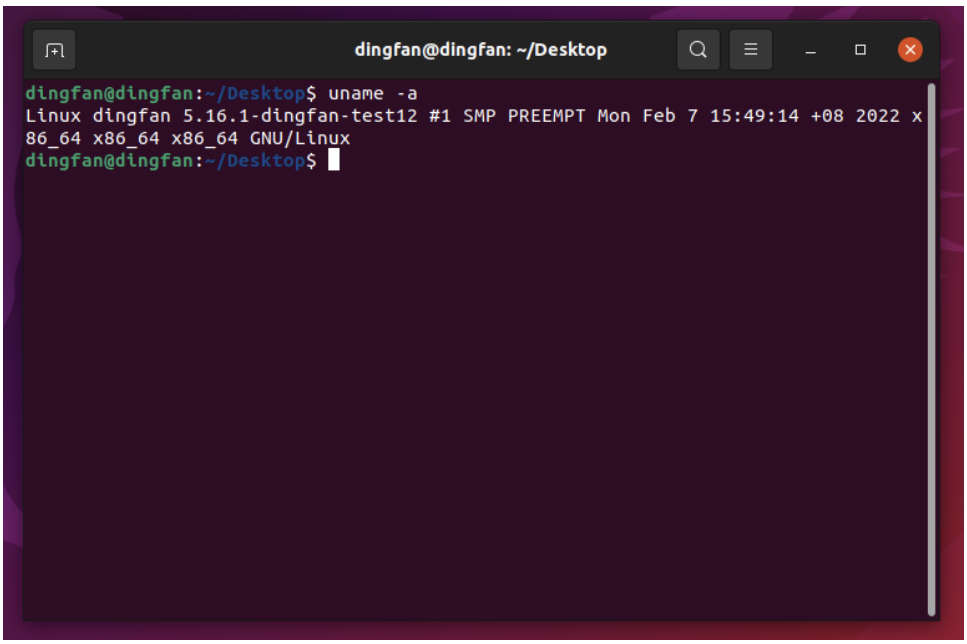


4. When I encounter failures, I need to goto the previous snapshot of my virtual machine and retry excluded features.

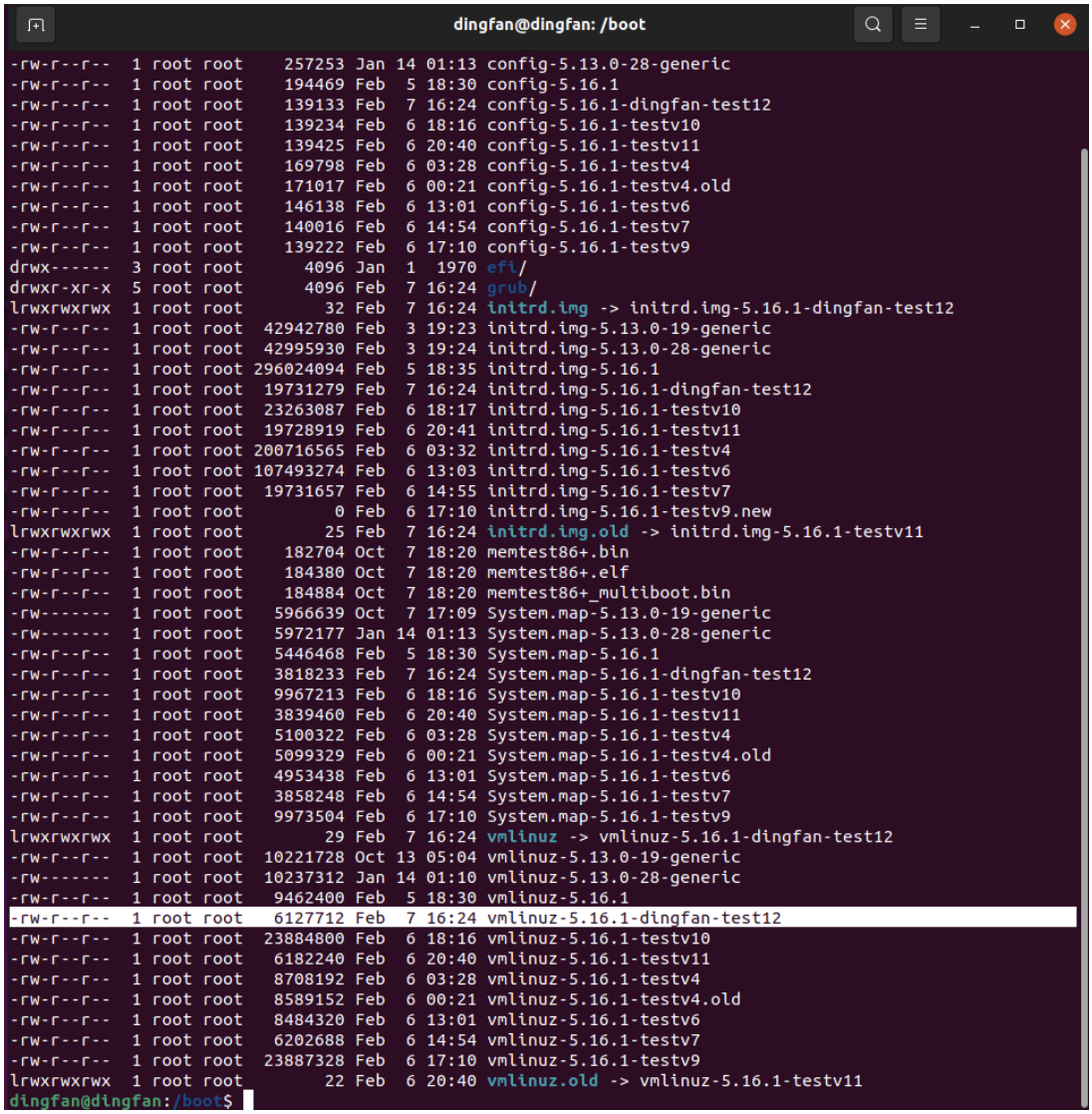


5. The smallest kernel:
    - The name of my smallest kernel version is 5.16.1-dingfan-test12

the screenshot when reboot successfully



- As a result, the size of the smallest kernel I built is `6.1MB`, which is highlight in the following picture



- compared to the original kernel(5.16.1) , the kernel size change from `10MB` to `6.1MB`

6. List the options that I disabled. (the feature I excluded):

Features	sublevel-1	sublevel-2	sublevel-3
Processor type and features	Enable MPS table		
Processor type and features	Support for extended (non-PC) x86 platforms		
Processor type and features	Enable Maximum number of SMP Processors and NUMA Nodes		
Processor type and features	Enable support for 16-bit segments		
Power Management and ACPI options	Garbage collector for user space wakeup sources		
Power Management and ACPI options	Power Management Debug Support		
Binary Emulation	x32 ABI for 64-bit mode		
Virtualization	all		
Executable file formats	Write ELF core dumps with partial segments		
Executable file formats	Enable core dump support		
Networking Support	networking options		
Networking Support	networking options	TCP/IP networking	
Networking Support	networking options	security marking	

Features	sublevel-1	sublevel-2	sublevel-3
Networking Support	networking options	timestamping in phy devices	
Networking Support	networking options	802.1d Ethernet Bridging	
Networking Support	networking options	802.10/802.1ad VLAN Support	
Networking Support	networking options	Phonet protocols family	
Networking Support	networking options	IEEE Std 802.15,4 Low-Rate Wireless Personal Area Networks support	
Networking Support	networking options	Qos and/or fair queueing	
Networking Support	networking options	Data Center Bridging support	
Networking Support	networking options	DNS Resolver support	
Networking Support	networking options	The IPV6 protocol	
Networking Support	networking options	MPTCP: Multipath TCP	
Networking Support	networking options	appletalk protocol support	
Networking Support	networking options	open vswitch	
Networking Support	networking options	network coding	
Networking Support	networking options	Multiprotocol Label Switching	
Networking Support	networking options	Use percpu variables to maintain network device refcount	
Networking Support	amateur radio support	all	
Networking Support	bluetooth subsystem support	all	

Features	sublevel-1	sublevel-2	sublevel-3
Networking Support	IPV6 support for rxrpc		
Networking Support	rxrpc kerberos security		
Networking Support	wireless	all	
Networking Support	RF switch subsystem support	all	
Networking Support	plan 9 resource sharing support(9p2000)	all	
Networking Support	CAIF support	all	
Networking Support	use in-kernel support for dns lookup		
Networking Support	NFC subsystem	all	
Networking Support	packet-sampling netlink channel		
Networking Support	Network light weight tunnels	all	
Networking Support	netlink interface for ethtool		
Device Drivers	PCI	PCI Express Hotplug driver	
Device Drivers	PCI	PCI Express ASPM control	
Device Drivers	PCI	Enable PCI resource re-allocation detection	
Device Drivers	PCI	Xen PCI Frontend	
Device Drivers	Generic Driver Options	Support for uevent helper	
Device Drivers	Generic Driver Options	Automount devtmpfs at /dev, after the kernel mounted the rootfs	
Device Drivers	Generic Driver Options	Allow device coredump	
Device Drivers	Macintosh device drivers		
Device Drivers	network device support	all	

Features	sublevel-1	sublevel-2	sublevel-3
Device Drivers	input device support		
Device Drivers	input device support	Joystick interface	
Device Drivers	input device support	Event debugging	
Device Drivers	input device support	touchscreens	
Device Drivers	input device support	joysticks/gamepads	
Device Drivers	input device support	Tablets	
Device Drivers	input device support	hardware I/O support	ct82c710 Aux port controller
Device Drivers	input device support	hardware I/O support	Parallel port keyboard adapter
Device Drivers	input device support	hardware I/O support	Raw access to serio ports
Device Drivers	input device support	hardware I/O support	Altera UP PS/2 controller
Device Drivers	input device support	hardware I/O support	Gameport support
Device Drivers	character devices	Legacy (BSD) PTY support	
Device Drivers	character devices	Non-standard serial port support	
Device Drivers	character devices	GSM MUX line discipline support	
Device Drivers	character devices	HSDPA Broadband Wireless Data Card - Globe Trotter	
Device Drivers	character devices	Parallel printer support	
Device Drivers	character devices	Support for user-space parallel port device drivers	
Device Drivers	character devices	IPMI top-level message handler	
Device Drivers	character devices	Applicom intelligent fieldbus card support	
Device Drivers	character devices	PCMCIA character devices	

Features	sublevel-1	sublevel-2	sublevel-3
Device Drivers	character devices	ACP Modem (Mwave) support	
Device Drivers	character devices	Hangcheck timer	
Device Drivers	character devices	Telecom clock driver for ATCA SBC	
Device Drivers	PPS support		
Device Drivers	Generic Thermal sysfs driver		
Device Drivers	remote controller support		
Device Drivers	multimedia support		
Device Drivers	sound card support		
Device Drivers	usb support		
Device Drivers	MMC/SD/SDIO card support		
Device Drivers	sony memorystick card support		
Device Drivers	accessibility support		
Device Drivers	Auxiliary Display support		
Device Drivers	VFIO Non-Privileged userspace driver framework		
Device Drivers	Microsoft Hyper-V guest support		
Device Drivers	Xen driver support		
Device Drivers	Generic Dynamic Voltage and Frequency Scaling (DVFS) support		
Device Drivers	External Connector Class (extcon) support		
Device Drivers	staging drivers		
Device Drivers	Platform support for Chrome hardware		
Device Drivers	Platform support for Mellanox hardware		
Device Drivers	Microsoft Surface platform-specific device drivers		

Features	sublevel-1	sublevel-2	sublevel-3
Device Drivers	Memory Controller drivers		
Device Drivers	Industrial I/O support		
Device Drivers	Non-Transparent Bridge support		
Device Drivers	VME bridge support		
Device Drivers	Pulse-Width Modulation (PWM) Support		
Device Drivers	IndustryPack bus support		
Device Drivers	Reset Controller Support		
Device Drivers	PHY Subsystem		
Device Drivers	Generic powercap sysfs driver		
Device Drivers	MCB support		
Device Drivers	Thunderbolt support for Apple devices		
Device Drivers	Reliability, Availability and Serviceability (RAS) features		
Device Drivers	android	android drivers	
Security Options	all		
Cryptographic API	all		
Library Routines	all		
Kernel Hacking	all		

Part B: x86 Assembly Programming (10 marks)

1.What is the IA32 (i.e., 32-bit Intel processor) instruction corresponding to these bytes, showing each step of your decoding process?

03 64 02 c5 (base 16)

solution:

1. (Byte1: `03H`), because this instruction is a IA32 instruction. So the first byte is a Opcode byte.
    - The Opcode of IA32 can be 1, 2 or 4 byte. Because the first opcode byte is not `0FH` or `0F 38H`, we can know that the opcode is 1 byte.
    - After looking the Table A-2 One-Byte Opcode Map, `03H` is `ADD (Gv, Ev)`

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	ADD Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH ES^{i64}	POP ES^{i64}
1	ADC Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH SS^{i64}	POP SS^{i64}
2	AND Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=ES (Prefix)	DAA^{i64}
3	XOR Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=SS (Prefix)	AAA^{i64}
4	INC^{i64} general register / REX^{o64} Prefixes eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	PUSH^{d64} general register rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA^{i64}/ PUSHAD^{i64}	POPA^{i64}/ POPAD^{i64}	BOUND^{i64} Gv, Ma	ARPL^{i64} Ew, Gw MOVSXD^{o64} Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	Jcc^{f64}, Jb - Short-displacement jump on condition O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Immediate Grp 1^{1A} Eb, Ib	Ev, Iz	Eb, Ib^{i64}	Ev, Ib	TEST Eb, Gb	Ev, Gv	XCHG Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	XCHG word, double-word or quad-word register with rAX rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	MOV AL, Ob	rAX, Ov	Ob, AL	Ov, rAX	MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv
B	MOV immediate byte into byte register AL/R8B, Ib	CL/R9B, Ib	DL/R10B, Ib	BL/R11B, Ib	AH/R12B, Ib	CH/R13B, Ib	DH/R14B, Ib	BH/R15B, Ib
C	Shift Grp 2^{1A} Eb, Ib	Ev, Ib	near RET^{f64} Iw	near RET^{f64}	LES^{i64} Gz, Mp VEX+2byte	LDS^{i64} Gz, Mp VEX+1byte	Grp 11^{1A} - MOV Eb, Ib	Ev, Iz
D	Shift Grp 2^{1A} Eb, 1	Ev, 1	Eb, CL	Ev, CL	AAM^{i64} Ib	AAD^{i64} Ib		XLAT/ XLATB
E	LOOPNE^{f64}/ LOOPNZ^{f64} Jb	LOOPE^{f64}/ LOOPZ^{f64} Jb	LOOP^{f64} Jb	JrCXZ^{f64}/ Jb	IN AL, Ib	eAX, Ib	OUT Ib, AL	Ib, eAX
F	LOCK (Prefix)	INT1	REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3^{1A} Eb	Ev

- `G` means the operand is REG, `E` means the operand is REG or MEM
- so the instruction is a ADD instruction

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8*.
REX + 80 /0 *ib*	ADD *r/m8*, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m8*.
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16*.
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64*.
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m16*.
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m32*.
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64*.
00 /r	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8*.
REX + 00 /r	ADD *r/m8*, r8**	MR	Valid	N.E.	Add *r8* to *r/m8*.
01 /r	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16*.
01 /r	ADD *r/m32, r32*	MR	Valid	Valid	Add *r32* to *r/m32*.
REX.W + 01 /r	ADD *r/m64, r64*	MR	Valid	N.E.	Add *r64* to *r/m64*.
02 /r	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8*.
REX + 02 /r	ADD *r8*, r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8*.
03 /r	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16*.
03 /r	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.
REX.W + 03 /r	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64*.

NOTES:
*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

2. (Byte2: `64H`) the second byte (after opcode byte) of the instruction is a ModR/M byte.
   - `64H` $\Rightarrow$ `01100100` (base 2)
   - the 3 field of ModR/M is
     - MOD: `01`
     - REG: `100`
     - R/M: `100`
   - after looking the table

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX] [ECX] [EDX] [EBX] [--][--][1] disp32[2] [ESI] [EDI]	00	000 001 010 011 100 101 110 111	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	10 11 12 13 14 15 16 17	18 19 1A 1B 1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B 2C 2D 2E 2F	30 31 32 33 34 35 36 37	38 39 3A 3B 3C 3D 3E 3F
[EAX]+disp8[3] [ECX]+disp8 [EDX]+disp8 [EBX]+disp8 [--][--]+disp8 [EBP]+disp8 [ESI]+disp8 [EDI]+disp8	01	000 001 010 011 100 101 110 111	40 41 42 43 44 45 46 47	48 49 4A 4B 4C 4D 4E 4F	50 51 52 53 54 55 56 57	58 59 5A 5B 5C 5D 5E 5F	60 61 62 63 64 65 66 67	68 69 6A 6B 6C 6D 6E 6F	70 71 72 73 74 75 76 77	78 79 7A 7B 7C 7D 7E 7F
[EAX]+disp32 [ECX]+disp32 [EDX]+disp32 [EBX]+disp32 [--][--]+disp32 [EBP]+disp32 [ESI]+disp32 [EDI]+disp32	10	000 001 010 011 100 101 110 111	80 81 82 83 84 85 86 87	88 89 8A 8B 8C 8D 8E 8F	90 91 92 93 94 95 96 97	98 99 9A 9B 9C 9D 9E 9F	A0 A1 A2 A3 A4 A5 A6 A7	A8 A9 AA AB AC AD AE AF	B0 B1 B2 B3 B4 B5 B6 B7	B8 B9 BA BB BC BD BE BF
EAX/AX/AL/MM0/XMM0 ECX/CX/CL/MM/XMM1 EDX/DX/DL/MM2/XMM2 EBX/BX/BL/MM3/XMM3 ESP/SP/AH/MM4/XMM4 EBP/BP/CH/MM5/XMM5 ESI/SI/DH/MM6/XMM6 EDI/DI/BH/MM7/XMM7	11	000 001 010 011 100 101 110 111	C0 C1 C2 C3 C4 C5 C6 C7	C8 C9 CA CB CC CD CE CF	D0 D1 D2 D3 D4 D5 D6 D7	D8 D9 DA DB DC DD DE DF	E0 E1 E2 E3 E4 E5 E6 E7	E8 E9 EA EB EC ED EE EF	F0 F1 F2 F3 F4 F5 F6 F7	F8 F9 FA FB FC FD FE FF

NOTES:

1. The [--][--] nomenclature means a SIB follows the ModR/M byte.
2. The disp32 nomenclature denotes a 32-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is sign-extended and added to the index.

- MOD: `01` means one-byte signed displacement follows addressing mode byte
- REG: `100` means the source or destination register is `ESP`
- R/M: `100` combined with MOD: `01` means that 8-bit displacement(1 byte) that follows the ModR/M byte and SIB Byte
    - so the third byte of the instruction is a SIB byte, the fourth byte of the instruction is a displacement byte

3. (Byte3: `02H`), according to the decoding of ModR/M byte, we know the third byte is a SIB byte
    - `02H` ⇒ `00000010` (base2)
        - scale : `00`
        - index: `000`
        - base: `010`

- after looking the following table

Table 2-3. 32-Bit Addressing Forms with the SIB Byte

r32 (In decimal) Base = (In binary) Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	**SS**	**Index**	**Value of SIB Byte (in Hexadecimal)**							
[EAX]	00	000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	99	9A	9B	9C	9D	9E	9F
none		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

NOTES:

1. The [*] nomenclature means a disp32 with no base if the MOD is 00B. Otherwise, [*] means disp8 or disp32 + [EBP]. This provides the following address modes:

MOD bits	Effective Address
00	[scaled index] + disp32
01	[scaled index] + disp8 + [EBP]
10	[scaled index] + disp32 + [EBP]

- scale : `00` means `Index*1`
- index: `000` means `EAX`
- base: `010` means `EDX`

4. (Byte4: `C5H`), according to the decoding of ModR/M byte, we know the fourth byte is signed one byte displacement.
   - `C5H` ⇒ `11000101` (base 2) ⇒ `-59` (decimal)
5. According to the previous 4 steps decoding process, we know the instruction is

```
ADDL -59(%edx, %eax), %esp
```

2. What is the 64-bit instruction (Intel64) encoding for the following instruction. You need to show each step of your

workings.

`addq -0xf00(%rax, %r12, 2), %rbp`

solution:

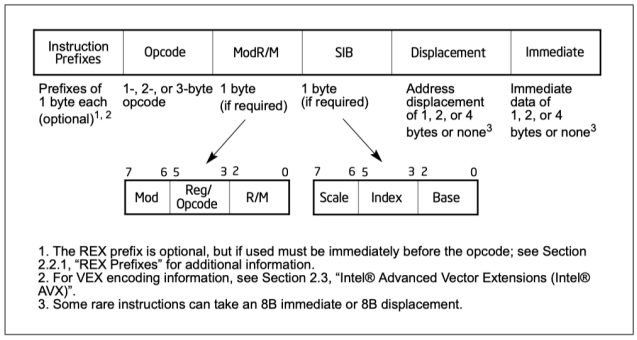


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

1. Encoding the opcode byte and part of the prefix byte:
   - from the instruction we know it is a add instruction that Add `m64` to `r64`
   - after looking the following table

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8*.
REX + 80 /0 *ib*	ADD *r/m8**, *imm8*	MI	Valid	N.E.	Add *sign-extended imm8* to *r/m8*.
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16*.
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64*.
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign-extended imm8* to *r/m16*.
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign-extended imm8* to *r/m32*.
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign-extended imm8* to *r/m64*.
00 /r	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8*.
REX + 00 /r	ADD *r/m8**, *r8**	MR	Valid	N.E.	Add *r8* to *r/m8*.
01 /r	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16*.
01 /r	ADD *r/m32, r32*	MR	Valid	Valid	Add *r32* to *r/m32*.
REX.W + 01 /r	ADD *r/m64, r64*	MR	Valid	N.E.	Add *r64* to *r/m64*.
02 /r	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8*.
REX + 02 /r	ADD *r8**, *r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8*.
03 /r	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16*.
03 /r	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.
REX.W + 03 /r	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64*.

NOTES:
*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

   - Because we know it is a add instruction that Add `m64` to `r64`, the Opcode is `REX.W+03/r`
   - So we can get the opcode byte is `03H`
   - Because this is a 64-bit instruction, the prefix byte is required. The `REX.W` is a part of instruction prefix byte.

- According to the following table, the first 5 bits of prefix byte is `01001` ;the last 3 bits of prefix byte is `RXB`, the value of `RXB` needs to be specify in the following encoding step.

Table 2-4. REX Prefix Fields [BITS: 0100WRXB]

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Operand size determined by CS.D
		1 = 64 Bit Operand Size
R	2	Extension of the ModR/M reg field
X	1	Extension of the SIB index field
B	0	Extension of the ModR/M r/m field, SIB base field, or Opcode reg field

2. Encoding the ModRM byte , SIB byte and displacement byte

   - According to the part of the instruction `-0xf00(%rax, %r12, 2), %rbp` , we know the 64-bit encoding instruction , must have a displacement byte (bytes) follows the ModRM byte and SIB byte, as the following table:

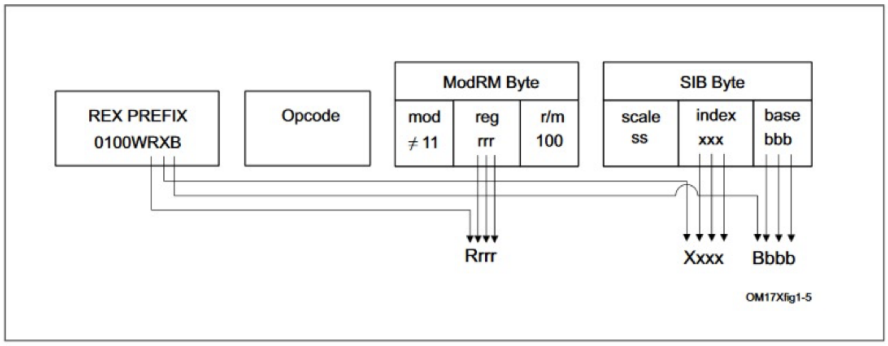


Figure 2-6. Memory Addressing With a SIB Byte

   a. Encoding the ModRM byte

      - `-0xf00` is the displacement which is longer than 1 byte (8 bit). After looking the following table, we know the displacement needs to be represent as a 4byte displacement. The effective address is `[SIB +dis32]` . So we can get the vale of MOD and B.R/M:

        - MOD: `10`
        - B.R/M: `1.100` ⇒ B: `0` , R/M: `100`

ModRM in 64 bit

32/64-bit	B.R/M															
Mod	0.000 AX	0.001 CX	0.010 DX	0.011 BX	0.100 SP	0.101 BP	0.110 SI	0.111 DI	1.000 R8	1.001 R9	1.010 R10	1.011 R11	1.100 R12	1.101 R13	1.110 R14	1.111 R15
00	[r/m]				[SIB]	[RIP/EIP][1,2] + disp32]	[r/m]						[SIB]	[RIP/EIP][1,2] + disp32]	[r/m]	
01	[r/m + disp8]				[SIB + disp8]	[r/m + disp8]							[SIB + disp8]	[r/m + disp8]		
10	[r/m + disp32]				[SIB + disp32]	[r/m + disp32]							[SIB + disp32]	[r/m + disp32]		
11	r/m															

- According to the following table with `%rbp`, we can get the value of R.Reg = `0.101`, the X.Reg in this table refer to the R.Reg in Intel Manual, R is the specific bit in prefix byte

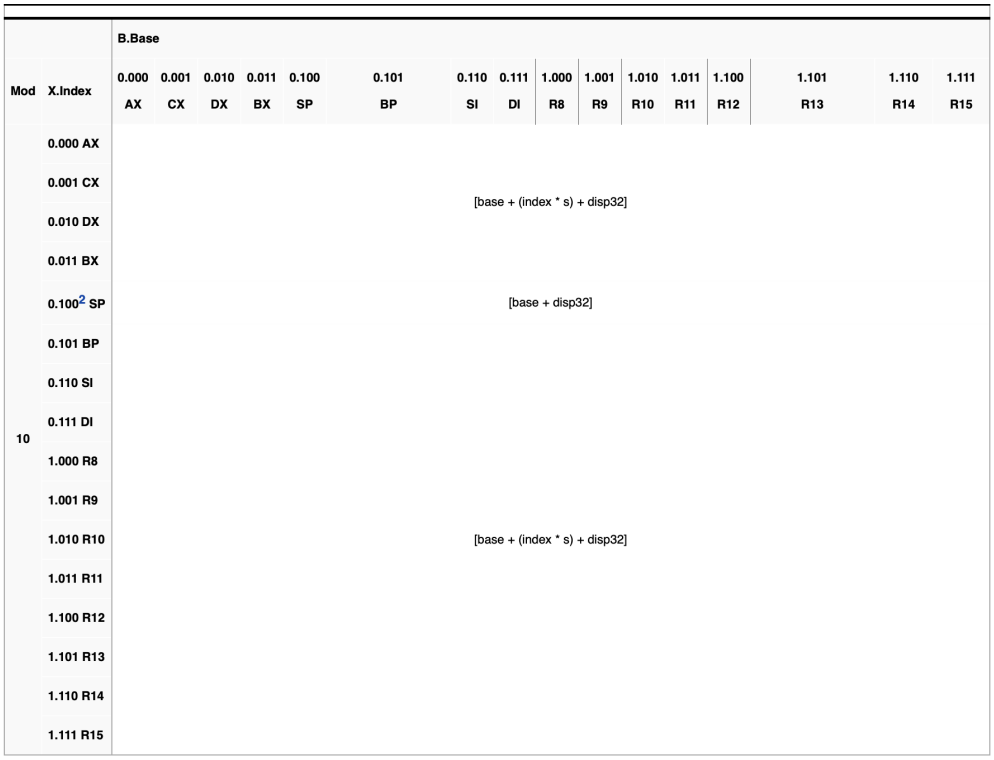
> ### Registers
>
> The registers are encoded using the 4-bit values in the X.Reg column of the following table. *X.Reg* is in binary.
>
> | X.Reg | 8-bit GP | 16-bit GP | 32-bit GP | 64-bit GP | 80-bit x87 | 64-bit MMX | 128-bit XMM | 256-bit YMM | 16-bit Segment | 32-bit Control | 32-bit Debug |
> |---|---|---|---|---|---|---|---|---|---|---|---|
> | **0.000 (0)** | AL | AX | EAX | RAX | ST0 | MMX0 | XMM0 | YMM0 | ES | CR0 | DR0 |
> | **0.001 (1)** | CL | CX | ECX | RCX | ST1 | MMX1 | XMM1 | YMM1 | CS | CR1 | DR1 |
> | **0.010 (2)** | DL | DX | EDX | RDX | ST2 | MMX2 | XMM2 | YMM2 | SS | CR2 | DR2 |
> | **0.011 (3)** | BL | BX | EBX | RBX | ST3 | MMX3 | XMM3 | YMM3 | DS | CR3 | DR3 |
> | **0.100 (4)** | AH, SPL[1] | SP | ESP | RSP | ST4 | MMX4 | XMM4 | YMM4 | FS | CR4 | DR4 |
> | **0.101 (5)** | CH, BPL[1] | BP | EBP | RBP | ST5 | MMX5 | XMM5 | YMM5 | GS | CR5 | DR5 |
> | **0.110 (6)** | DH, SIL[1] | SI | ESI | RSI | ST6 | MMX6 | XMM6 | YMM6 | - | CR6 | DR6 |
> | **0.111 (7)** | BH, DIL[1] | DI | EDI | RDI | ST7 | MMX7 | XMM7 | YMM7 | - | CR7 | DR7 |
> | **1.000 (8)** | R8L | R8W | R8D | R8 | - | MMX0 | XMM8 | YMM8 | ES | CR8 | DR8 |
> | **1.001 (9)** | R9L | R9W | R9D | R9 | - | MMX1 | XMM9 | YMM9 | CS | CR9 | DR9 |
> | **1.010 (10)** | R10L | R10W | R10D | R10 | - | MMX2 | XMM10 | YMM10 | SS | CR10 | DR10 |
> | **1.011 (11)** | R11L | R11W | R11D | R11 | - | MMX3 | XMM11 | YMM11 | DS | CR11 | DR11 |
> | **1.100 (12)** | R12L | R12W | R12D | R12 | - | MMX4 | XMM12 | YMM12 | FS | CR12 | DR12 |
> | **1.101 (13)** | R13L | R13W | R13D | R13 | - | MMX5 | XMM13 | YMM13 | GS | CR13 | DR13 |
> | **1.110 (14)** | R14L | R14W | R14D | R14 | - | MMX6 | XMM14 | YMM14 | - | CR14 | DR14 |
> | **1.111 (15)** | R15L | R15W | R15D | R15 | - | MMX7 | XMM15 | YMM15 | - | CR15 | DR15 |
>
> 1: When any REX prefix is used, SPL, BPL, SIL and DIL are used. Otherwise, without any REX prefix AH, CH, DH and BH are used.

  - R : `0`
  - Reg : `101`
- Because I already get : MOD: `10`, Reg : `101`, R/M: `100`,
- the ModRM byte is `10101100` (base2) ⇒ `ACH`

b. Encoding the SIB byte

- according to `(%rax, %r12, 2)`

- we can get
  - Index register: `%r12`
  - Base register: `%rax`
  - scale value: `2` ⇒ scale=`01`
- after looking the following table
- after looking the following table, we can get the value of X.index and B.base



Mod	X.Index	B.Base																	
		0.000 AX	0.001 CX	0.010 DX	0.011 BX	0.100 SP	0.101 BP	0.110 SI	0.111 DI	1.000 R8	1.001 R9	1.010 R10	1.011 R11	1.100 R12	1.101 R13	1.110 R14	1.111 R15
10	0.000 AX	[base + (index * s) + disp32]															
	0.001 CX																
	0.010 DX																
	0.011 BX																
	0.100[2] SP	[base + disp32]															
	0.101 BP																
	0.110 SI																
	0.111 DI																
	1.000 R8																
	1.001 R9																
	1.010 R10	[base + (index * s) + disp32]															
	1.011 R11																
	1.100 R12																
	1.101 R13																
	1.110 R14																
	1.111 R15																

  - X.index: `1100` ⇒ X: `1` , index: `100`
  - B.base: `0.000` ⇒ B: `0` , base : `000`
- because scale=`01` , index: `100` , base : `000`
- the SIB byte is `01100000` (base2) ⇒ `60H`

c. Encoding the prefix byte

- Because when encoding opcode byte we get the first 5 bits of prefix byte is `01001` , and get R : `0` , X: `1` , B: `0` in latest steps
- the prefix byte is `01001010` (base2) ⇒ `4AH`

d. Encoding the displacement bytes

- `-0xf00` ⇒ `0xfffff100`
- so the displacement bytes is `ff` `ff` `f1` `00`
- because the code must uses the GCC (i.e., AT&T) assembly syntax in this report, the displacement bytes need to be transfer into Little Endian: `00` `f1` `ff` `ff`

3. According to the previous steps, the 64-bit instruction encoding is

> 💡 4A 03 AC 60 00 F1 FF FF (base 16)

3. Disassemble the following IA32 assembly program by hand and recover the C function that performs the same task.

Disassemble the following IA32 assembly program by hand and recover the C function that performs the same task.

```
                .text
                .globl  foo
                .type   foo, @function
        foo:
                pushl   %ebp
                movl    %esp, %ebp
                pushl   %ebx
                subl    $4, %esp
                cmpl    $-1, 8(%ebp)
                jge     .L2
                movl    $-1, %eax
                jmp     .L3
        .L2:
                cmpl    $2, 8(%ebp)
                jg      .L4
                movl    $1, %eax
                jmp     .L3
        .L4:
                movl    8(%ebp), %eax
                subl    $2, %eax
                subl    $12, %esp
                pushl   %eax
                call    foo
                addl    $16, %esp
                movl    %eax, %ebx
                movl    8(%ebp), %eax
                subl    $3, %eax
                subl    $12, %esp
                pushl   %eax
                call    foo
                addl    $16, %esp
                addl    %ebx, %eax
        .L3:
                movl    -4(%ebp), %ebx
                leave
                ret
```

solution:

1. suppose the name of the function is `func`, the arguments pass to callee `func` is `num1,num2,num3.....`(if need). When the function is called, the assembly program will jump to the label `foo:`
2. the following code is the setup process:

```
pushl  %ebp
movl   %esp, %ebp
pushl  %ebx
subl   $4, %esp
```

   - push the old `%ebp` value to the stack to save it;
   - set the new `%ebp` value of callee `func` , it points to the bottom of the callee `func` on stack
   - save the `%ebx` value on stack prior to using, because the `%ebx` register is Callee-Save Temporaries
   - set the new `%esp` value of callee `func`, it points to the top of the callee `func` on stack

3. the following code is related to a “if statement”

```
cmpl $-1, 8(%ebp)
jge  .L2
movl  $-1, %eax
jmp  .L3
```

   - `8(%ebp)` get the first argument `num1`, which is passed from caller
   - `cmpl` is a compare instruction which compare `-1` and `num1`
   - `jge` is a jump instruction, which is used combine with the above `cmpl`, if `num1` greater or equal than `-1` (`num ≥ -1`), the assembly program jump to label `.L2`
     - otherwise(`num < -1`), set `%eax = -1`, then the assembly program jump to label `.L3`
   - the Pseudocode is following:

```
if (num1 >= -1){
  //jump to .L2;
}else if(num < -1){
  %eax = -1;//(return -1;)
```

```
  //jump to .L3
}
```

4. similarly, the following code is also related to a “if statement”

```
.L2:
      cmpl  $2, 8(%ebp)
      jg    .L4
      movl  $1, %eax
      jmp   .L3
```

- `8(%ebp)` get the first argument `num1`, which is passed from caller
- `cmpl` is a compare instruction which compare `2` and `num1`
- `jg` is a jump instruction, which is used combine with the above `cmpl`, if `num1` greater than `2` (`num1 > 2`), the assembly program jump to label `.L4`
  - otherwise(`num≤2`, also `num≥ -1` because the program jumped from step3), set `%eax = 1`, then the assembly program jump to label `.L3`
- the Pseudocode is following:

```
if(num1 > 2){
    //jump to .L4
}else if(num1 <=2 && num1 >=-1){
    %eax=1; //(return 1)
    //jump to .L3;
}
```

5. the label `.L4` is a complex part which, as caller, call the `func` ,as callee, twice time.
   a. the first call `func(num1 - 2)`

```
movl 8(%ebp), %eax
subl $2, %eax
subl $12, %esp
pushl %eax
call foo
addl $16, %esp
movl %eax, %ebx
```

- the 2 instructions: `movl 8(%ebp), %eax` and `subl $2, %eax` ⇒ set `%eax = num1 - 2`

- the 2 instructions: `subl $12, %esp`, `pushl %eax` is the setup process before call the `func`
- instruction : `call foo`, call the `func(num1-2)`, the return value is in `%eax`
- instruction : `addl $16, %esp`, is the finish process after call the `func(num1-2)`
- instruction: `movl %eax, %ebx` => set `%ebx` equal to the return value of `func(num1 -2)`

b. the second call `func(num1 - 3)`

```
movl 8(%ebp), %eax
subl $3, %eax
subl $12, %esp
pushl %eax
call foo
addl $16, %esp
addl %ebx, %eax
```

- the 2 instructions: `movl 8(%ebp), %eax` and `subl $3, %eax` ⇒set `%eax = num1 - 3`
- the 2 instructions: `subl $12, %esp`, `pushl %eax` is the setup process before call the `func`
- instruction : `call foo`, call the `func(num1-3)`, the return value is in `%eax`
- instruction : `addl $16, %esp`, is the finish process after call the `func(num1-3)`
- instruction: `movl %ebx, %eax`, set `%eax` = `%eax` + `%ebx`, which is `func(num1-3)` + `func(num1 -2)`

c. the Pseudocode is following:

```
%eax = func(num1-3) + func(num1 -2); // return func(num1-3) + func(num1 -2);
```

6. the following code is a finish process

```
.L3:
      movl -4(%ebp), %ebx
      leave
      ret
```

- because the return value of `func` is get from `%eax`

- the above code is equal to `return %eax`

7. According to the above analysis, the C function can be recovered as following:

```
int func(int num){
        if(num < -1){
                return -1;
        }else if(num >= -1 && num <=2 ){
                return 1;
        }else{
                return func(num - 2) + func(num-3);
        }
}
```