期末复习

王慧妍
why@nju.edu.cn

南京大学 计算机科学与技术系 计算机软件研究所

本讲概述

考试：占比不高，不必惊慌

- 本讲内容
  - 学期总结
  - 如何阅读汇编代码

学期总结

这学期讲了些什么？

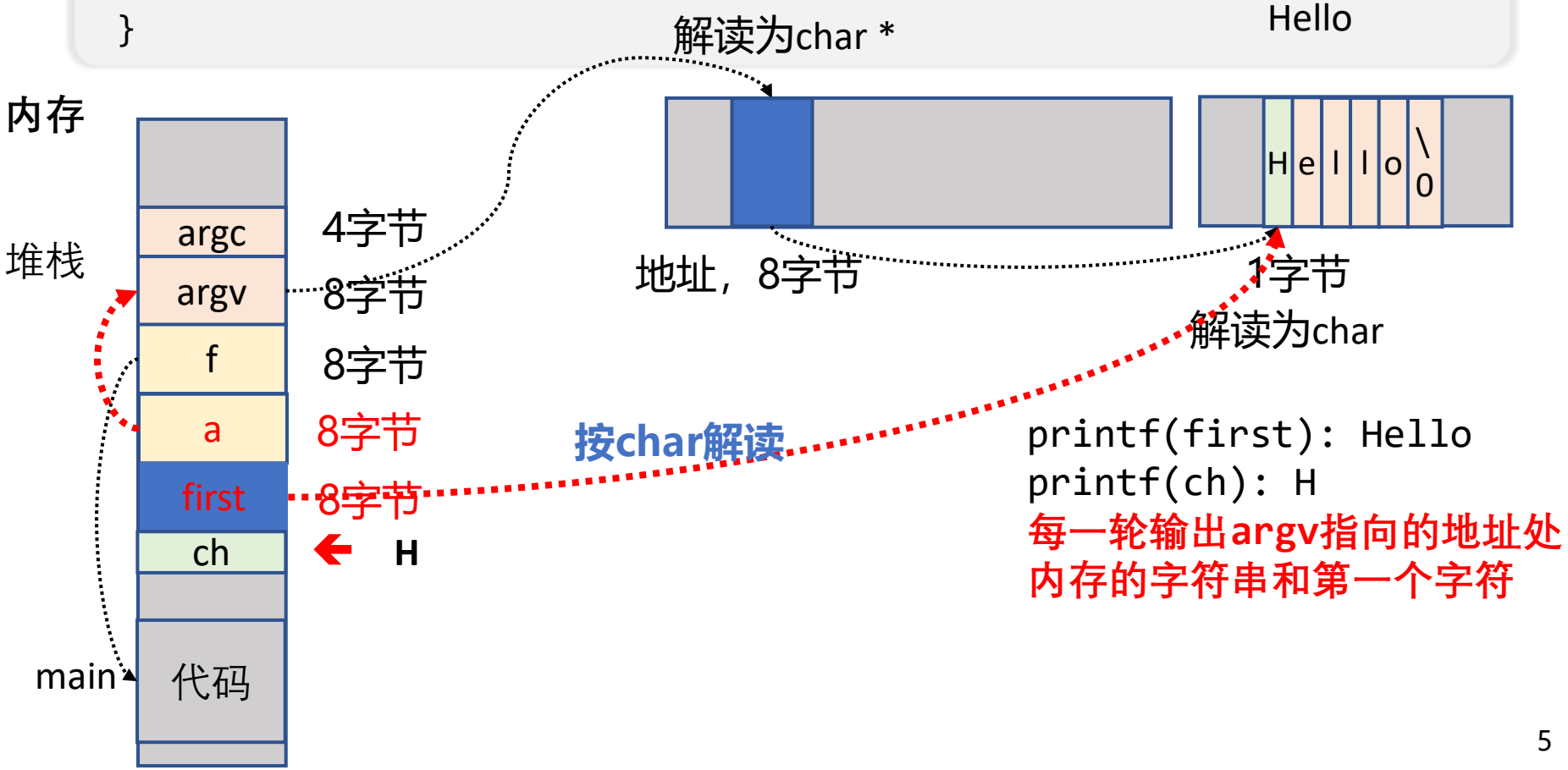
Week2: C语言拾遗1之机制

- 在IDE里，为什么按一个键，就能够编译运行?
  - 编译、链接
    - `.c` → 预编译 → `.i` → 编译 → `.s` → 汇编 → `.o` → 链接 → `a.out`
  - 加载执行
    - `./a.out`

- 背后是通过调用命令行工具完成的
  - RTFM: `man gcc; gcc –help; tldr gcc`
    - 控制行为的三个选项: `-E`, `-S`, `-c`

- 本次课程
  - 预热：编译、链接、加载到底做了什么?
  - RTFSC时需要关注的C语言特性

```c
int main(int argc, char *argv[]) {
    int (*f)(int, char *[]) = main;
    if (argc != 0) {
        char ***a = &argv, *first = argv[0], ch = argv[0][0];
        printf("arg = \"%s\"; ch = '%c'\n", first, ch);
        assert(***a == ch);
        f(argc - 1, argv + 1);
    }
}
```



Week3: C语言拾遗2之编写可读的代码

- IOCCC'11 best self documenting program
  - 不可读 = 不可维护

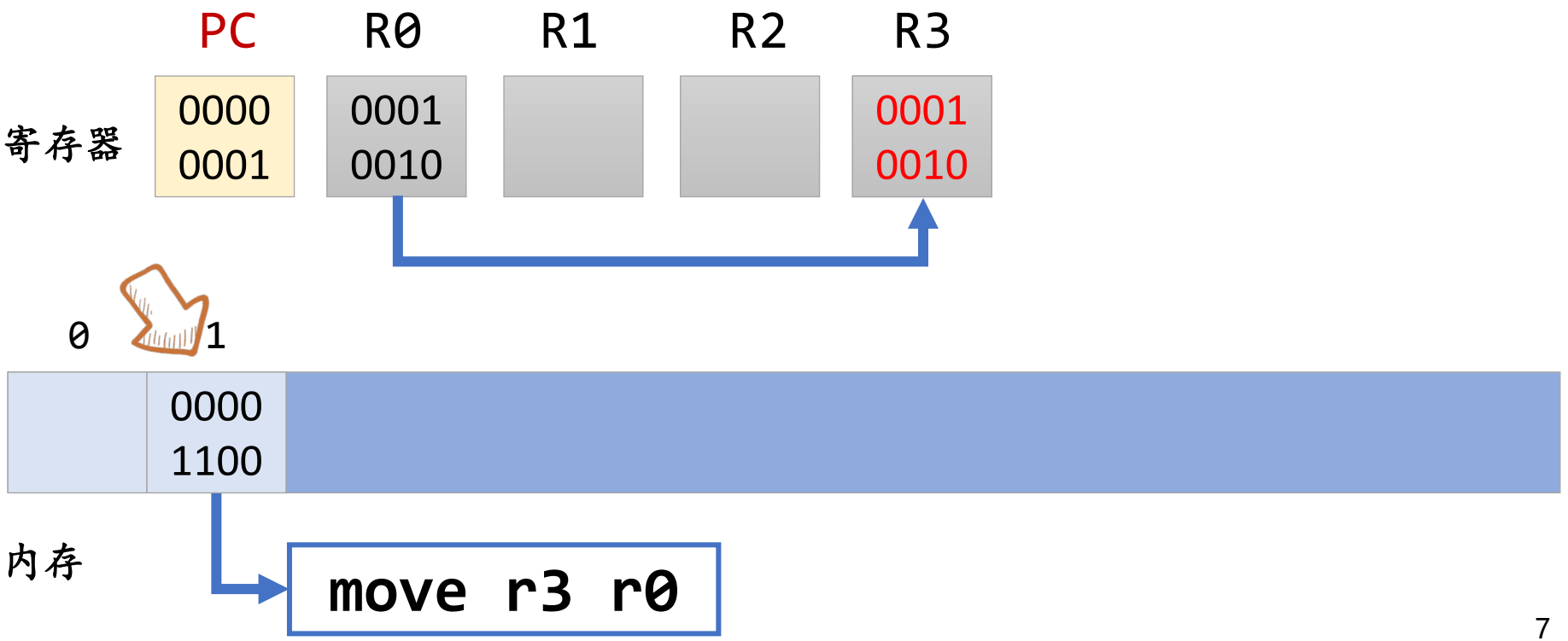
```
#define clear 1;
   if(c>=11){c=0;sscanf(_,"%lf%c",&r,&c);while(*++_-c);}\
   else if(argc>=4&&!main(4-(*_++=='('),argv))_++;g:c+=
#define puts(d,e) return 0;}{double a;int b;char
   c=(argc<4?d)&15;\ b=(*_%__LINE__+7)%9*(3*e>>c&1);c+=
#define I(d) (r);if(argc<4&&*#d==*_){a=r;r=usage?r*a:r+a;goto
   g;}c=c
#define return if(argc==2)printf("%f\n",r);return argc>=4+
#define usage main(4-__LINE__/26,argv)
#define calculator *_*(int)
#define l (r);r=--b?r:
#define _ argv[1]
```

教科书第一章上的 “计算机系统”

- 存储系统and指令集

寄存器：PC，RO (RA)，R1，R2，R3 (8-bit)
内存：16字节（按字节访问）

	7 6 5 4	3 2 1 0
mov	[0 0 0 0]	[rt][rs]
add	[0 0 0 1]	[rt][rs]
load	[1 1 1 0]	[addr]
store	[1 1 1 1]	[addr]

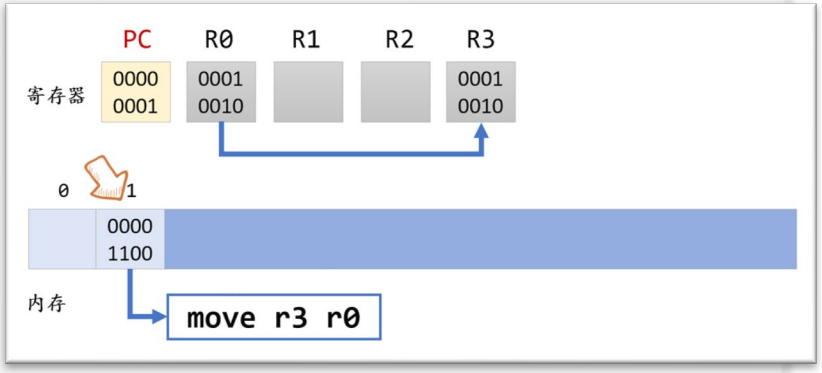


代码例子1

寄存器：PC，RO (RA)，R1，R2，R3 (8-bit)
内存：16字节（按字节访问）

	7 6 5 4 3 2 1 0
mov	[0 0 0 0] [rt][rs]
add	[0 0 0 1] [rt][rs]
load	[1 1 1 0] [addr]
store	[1 1 1 1] [addr]

```
void idex() {
    if ((M[pc] >> 4) == 0) {
        R[(M[pc] >> 2) & 3] = R[M[pc] & 3];
        pc++;
    } else if ((M[pc] >> 4) == 1) {
        R[(M[pc] >> 2) & 3] += R[M[pc] & 3];
        pc++;
    } else if ((M[pc] >> 4) == 14) {
        R[0] = M[M[pc] & 0xf];
        pc++;
    } else if ((M[pc] >> 4) == 15) {
        M[M[pc] & 0xf] = R[0];
        pc++;
    }
}
```

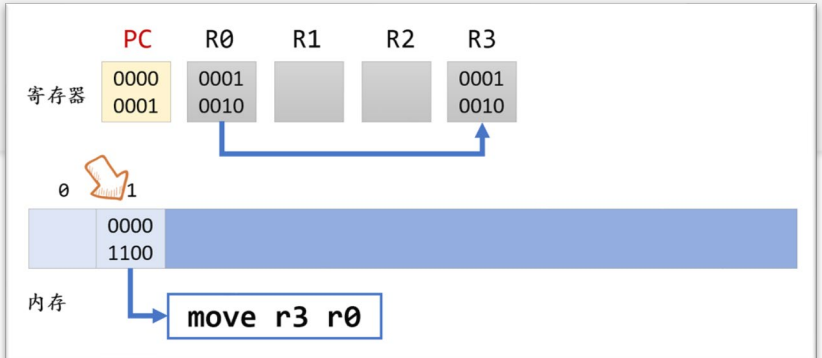


代码例子2

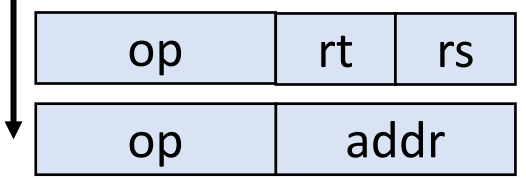
寄存器：PC，RO（RA），R1，R2，R3（8-bit）
内存：16字节（按字节访问）

	7 6 5 4 3 2 1 0
mov	[0 0 0 0] [rt][rs]
add	[0 0 0 1] [rt][rs]
load	[1 1 1 0] [addr]
store	[1 1 1 1] [addr]

```
void idex() {
    u8 inst = M[pc++];
    u8 op = inst >> 4;
    if (op == 0x0 || op == 0x1) {
        int rt = (inst >> 2) & 3, rs = (inst & 3);
        if (op == 0x0)          R[rt] = R[rs];
        else if (op == 0x1)     R[rt] += R[rs];
    }
    if (op == 0xe || op == 0xf) {
        int addr = inst & 0xf;
        if (op == 0xe) R[0] = M[addr];
        else if (op == 0xf) M[addr] = R[0];
    }
}
```



代码例子3（YEMU代码）




```
typedef union inst {
   struct { u8 rs : 2, rt: 2,  op: 4; } rtype;
   struct { u8 addr: 4,        op: 4; } mtype;
} inst_t;
#define RTYPE(i) u8 rt = (i)->rtype.rt, rs = (i)->rtype.rs;
#define MTYPE(i) u8 addr = (i)->mtype.addr;

void idex() {
   inst_t *cur = (inst_t *)&M[pc];
   switch (cur->rtype.op) {
       case 0b0000: { RTYPE(cur); R[rt] = R[rs];   pc++; break; }
       case 0b0001: { RTYPE(cur); R[rt] += R[rs];  pc++; break; }
       case 0b1110: { MTYPE(cur); R[RA] = M[addr]; pc++; break; }
       case 0b1111: { MTYPE(cur); M[addr] = R[RA]; pc++; break; }
       default: panic("invalid instruction at PC = %x", pc);
   }
}
```

有用的C语言特性

- Union / bit fields

```
typedef union inst {
   struct { u8 rs : 2, rt: 2,  op: 4; } rtype;
   struct { u8 addr: 4,        op: 4; } mtype;
} inst_t;
```

```
// decode
typedef struct {
  union {
    struct {
      uint32_t opcode1_0 : 2;
      uint32_t opcode6_2 : 5;
      uint32_t rd        : 5;
      uint32_t funct3    : 3;
      uint32_t rs1       : 5;
      int32_t  simm11_0  :12;
    } i;
    struct {
      uint32_t opcode1_0 : 2;
      uint32_t opcode6_2 : 5;
      uint32_t imm4_0    : 5;
      uint32_t funct3    : 3;
      uint32_t rs1       : 5;
      uint32_t rs2       : 5;
      int32_t  simm11_5  : 7;
    } s;
    struct {
      uint32_t opcode1_0 : 2;
      uint32_t opcode6_2 : 5;
      uint32_t rd        : 5;
      uint32_t imm31_12  :20;
    } u;
    uint32_t val;
  } instr;
} riscv32_ISADecodeInfo;
```

- 指针
  - 内存只是个字节序列
  - 无论何种类型的指针都只是地址 + 对指向内存的解读

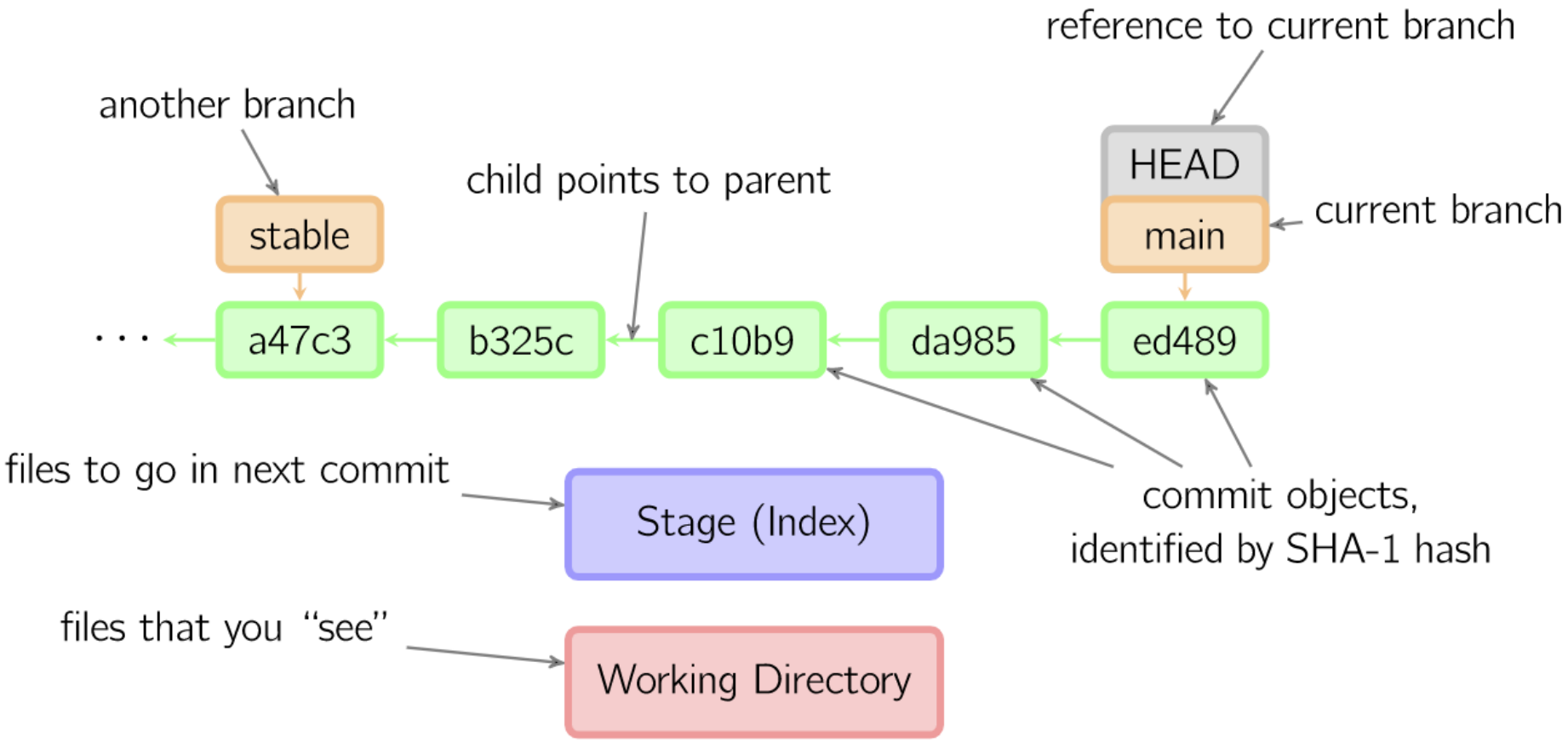
```
inst_t *cur = (inst_t *)&M[pc];
   // cur -> rtype.op
   // cur -> mtype.addr
```

Week4: 框架代码选讲1

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

- 无所不能的代码聚集地
  - 有整个计算机系统世界的代码
    - 硬件、操作系统、分布式系统、库函数、应用程序……
- 学习各种技术的最佳平台
  - 海量的文档、学习资料、博客（新世界的大门）
    - 提供友好的搜索（例子：**awesome C**）

A Visual Git Reference



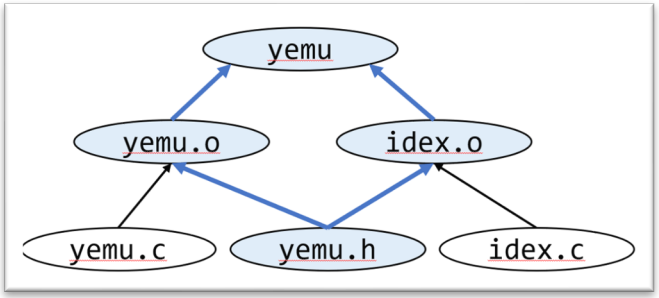

```
commit 8738b7b32e71a78f5b73376dc664780dd16800eb (HEAD -> pa1)
Author: tracer-ics2020 <tracer@njuics.org>
Date:   Thu Aug 19 18:27:15 2021 +0800
```

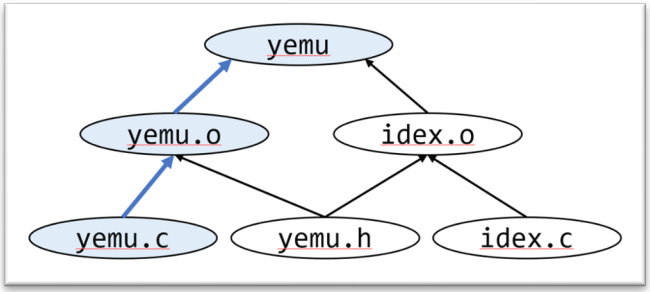
Make工具

- 回顾：YEMU 模拟器

- Makefile 是一段 “declarative” 的代码
  - 描述了构建目标之间的依赖关系和更新方法

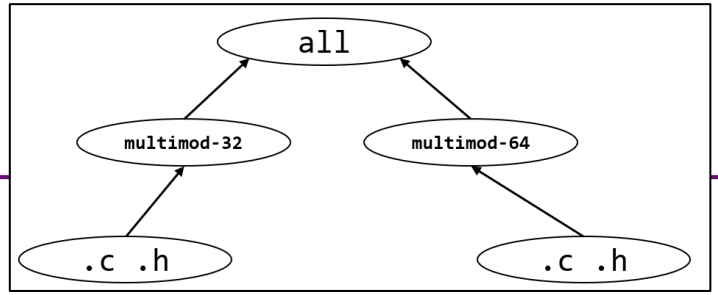
```
$ make
gcc -Wall -Werror -std=c11 -O2 -c -o yemu.o yemu.c
gcc -Wall -Werror -std=c11 -O2 -c -o idex.o idex.c
gcc  -o yemu yemu.o idex.o
$ make
make: 'yemu' is up to date.
```

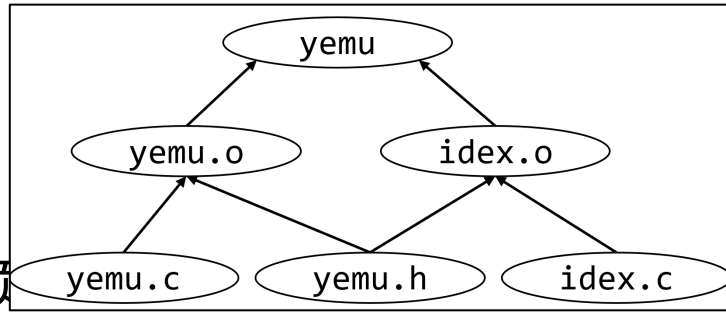




- `make -j8`

NEMU代码构建





- Makefile 真复杂
  - 放弃?
- 一个小诀窍
  - 先观察 make 命令实际执行了什么 (trace)
  - RTFM/STFW: make 提供的两个有用的选项
    - `-n` 只打印命令不运行
    - `-B` 强制 make 所有目标

```
make -nB \
  | grep -ve '^\(\#\|echo\|mkdir\)' \
  | vim -
```

- 其实没有那么复杂
  - 就是一堆gcc –c（编译）和一个gcc（链接）
    - 大部分Makefile都是编译选项

Week5: 框架代码选讲2

NEMU对大部分同学来说是一个“前所未有大”的项目。

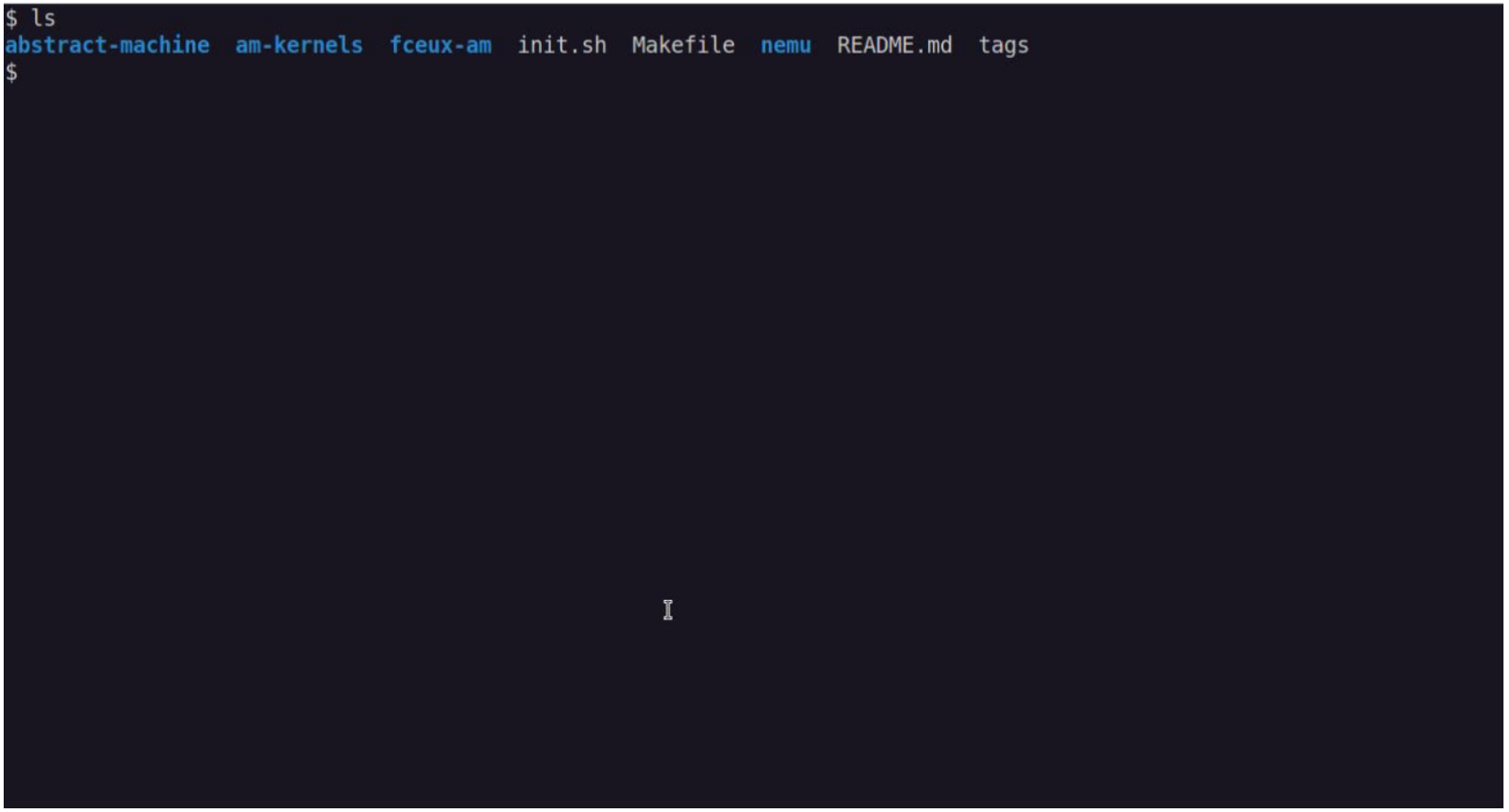
- 先大致了解一下
  - 项目总体组织
    - tree要翻好几个屏幕
    - `find . -name "*.c" -o -name "*.h"` (100+ 个文件)
  - 项目规模
    - `find ... | xargs cat | wc -l`

```
$ find . -name "*.c" -o -name "*.h" | xargs cat | wc -l
23069
```

```
$ find tools -name "*.c" -o -name ".h" | xargs cat | wc -l
17728
```

尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**



尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**

- The UNIX Way (无需启动任何程序，即可直接查看)

```
grep -n main $(find . -name "*.c") # RTFM: -n
```

grep -n main $(find . -name "*.c")

```
$ grep -n main $(find . -name "*.c")
./tools/gen-expr/gen-expr.c:13:"int main() { "
./tools/gen-expr/gen-expr.c:23:int main(int argc, char *argv[]) {
./tools/fixdep/fixdep.c:385:int main(int argc, char *argv[])
./tools/kconfig/mconf.c:1004:int main(int ac, char **av)
./tools/kconfig/symbol.c:1265:          /* for choice groups start the check with main
 choice symbol */
./tools/kconfig/menu.c:331:                     /* set the type of the remaining choic
e values */
./tools/kconfig/build/lexer.lex.c:144:/* The state buf must be large enough to hold on
e state per character in the main buffer.
./tools/kconfig/build/lexer.lex.c:2684:/** The main scanner function which does all th
e work.
./tools/kconfig/build/lexer.lex.c:4119:/* Defined in main.c */
./tools/kconfig/build/parser.tab.c:541:  "T_NOT", "$accept", "input", "mainmenu_stmt",
 "stmt_list",
./tools/kconfig/conf.c:500:int main(int ac, char **av)
./resource/sdcard/nemu.c:91:    unsigned int            blocks;         /* remaining P
IO blocks */
./src/engine/interpreter/init.c:3:void sdb_mainloop();
./src/engine/interpreter/init.c:10:  sdb_mainloop();
./src/isa/riscv64/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv64/instr/decode.c:65:  int idx = table_main(s);
./src/isa/riscv32/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv32/instr/decode.c:65:  int idx = table_main(s);
./src/nemu-main.c:8:int main(int argc, char *argv[]) {
./src/monitor/sdb/sdb.c:84:void sdb_mainloop() {
./src/monitor/sdb/sdb.c:97:    /* treat the remaining string as the arguments,
```

尝试阅读代码：从main开始

- C语言代码，都是从main()开始运行的。那么哪里才有main呢?
  - 浏览代码：发现main.c，估计在里面
  - 使用**IDE（vscode：Edit→Find in files)**

- The UNIX Way (无需启动任何程序，即可直接查看)

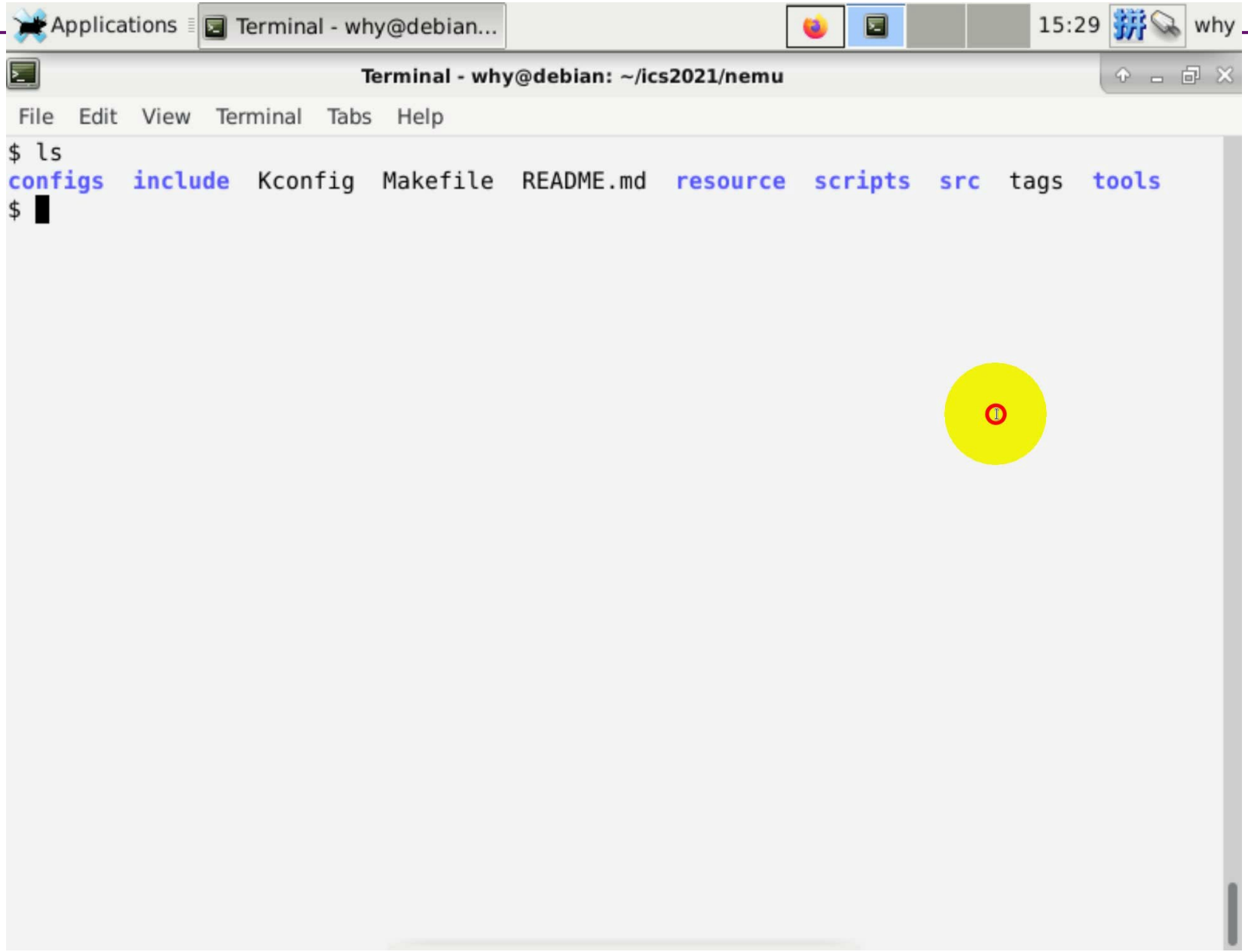
```
grep -n main $(find . -name "*.c") # RTFM: -n
```

```
find . | xargs grep --color -nse '\<main\>'
```

find . | xargs grep --color -nse '\<main\>'

```
$ find . | xargs grep --color -nse '\<main\>'
./README.md:7:The main features of NEMU include
./tools/gen-expr/gen-expr.c:13:"int main() { "
./tools/gen-expr/gen-expr.c:23:int main(int argc, char *argv[]) {
./tools/fixdep/fixdep.c:385:int main(int argc, char *argv[])
Binary file ./tools/fixdep/build/fixdep matches
Binary file ./tools/fixdep/build/obj-fixdep/fixdep.o matches
./tools/kconfig/mconf.c:1004:int main(int ac, char **av)
./tools/kconfig/symbol.c:1265:          /* for choice groups start the check with main
 choice symbol */
Binary file ./tools/kconfig/build/conf matches
Binary file ./tools/kconfig/build/obj-conf/conf.o matches
./tools/kconfig/build/lexer.lex.c:144:/* The state buf must be large enough to hold on
e state per character in the main buffer.
./tools/kconfig/build/lexer.lex.c:2684:/** The main scanner function which does all th
e work.
./tools/kconfig/build/lexer.lex.c:4119:/* Defined in main.c */
Binary file ./tools/kconfig/build/obj-mconf/mconf.o matches
Binary file ./tools/kconfig/build/mconf matches
./tools/kconfig/conf.c:500:int main(int ac, char **av)
./tools/kconfig/expr.h:70:      S_DEF_USER,             /* main user value */
./src/isa/riscv64/instr/decode.c:55:def_THelper(main) {
./src/isa/riscv32/instr/decode.c:55:def_THelper(main) {
./src/filelist.mk:1:SRCS-y += src/nemu-main.c
./src/nemu-main.c:8:int main(int argc, char *argv[]) {
Binary file ./build/riscv32-nemu-interpreter matches
./build/obj-riscv32-nemu-interpreter/src/nemu-main.d:1:cmd_/home/why/ics2021/nemu/buil
d/obj-riscv32-nemu-interpreter/src/nemu-main.o := unused
```

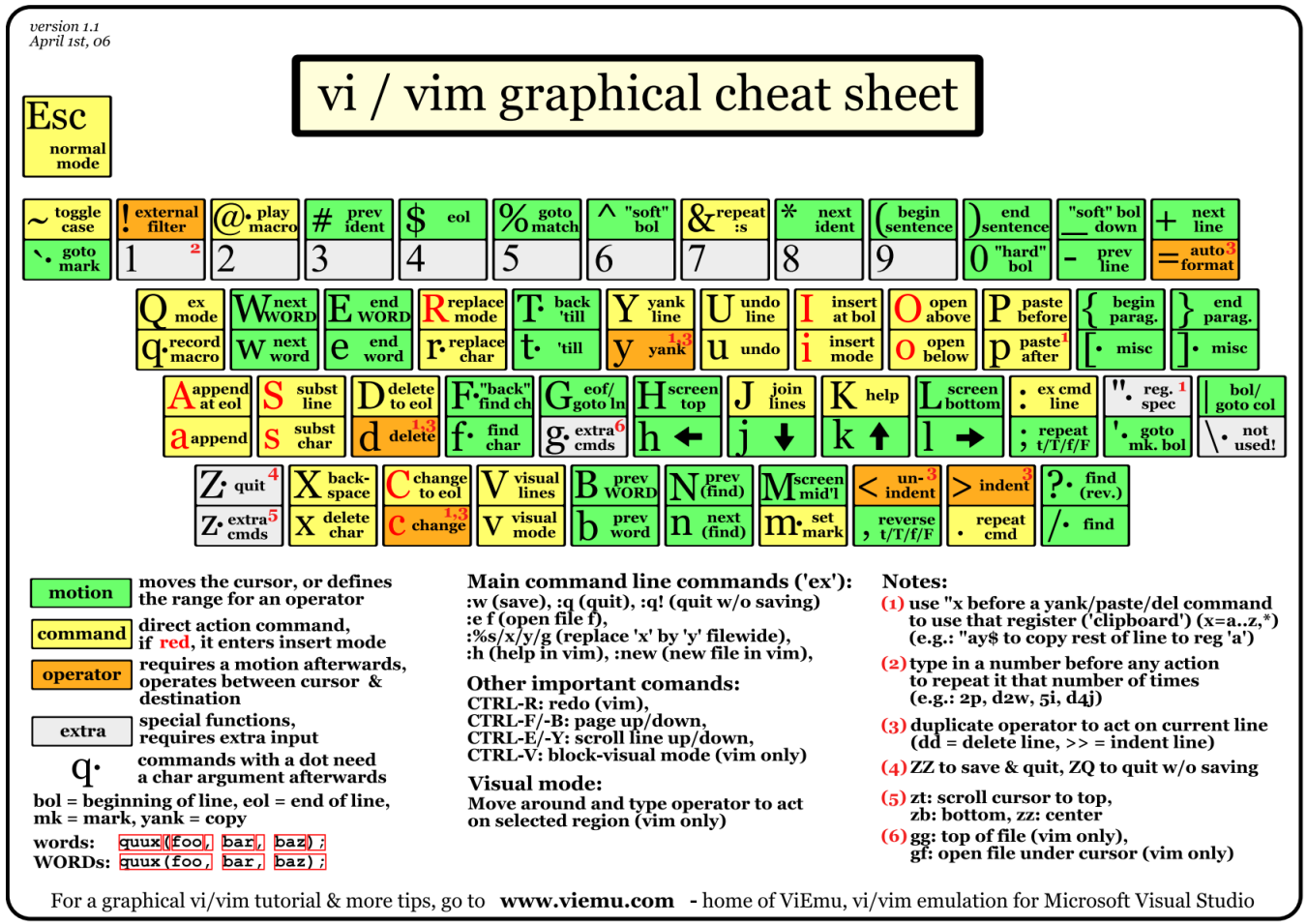
fzf

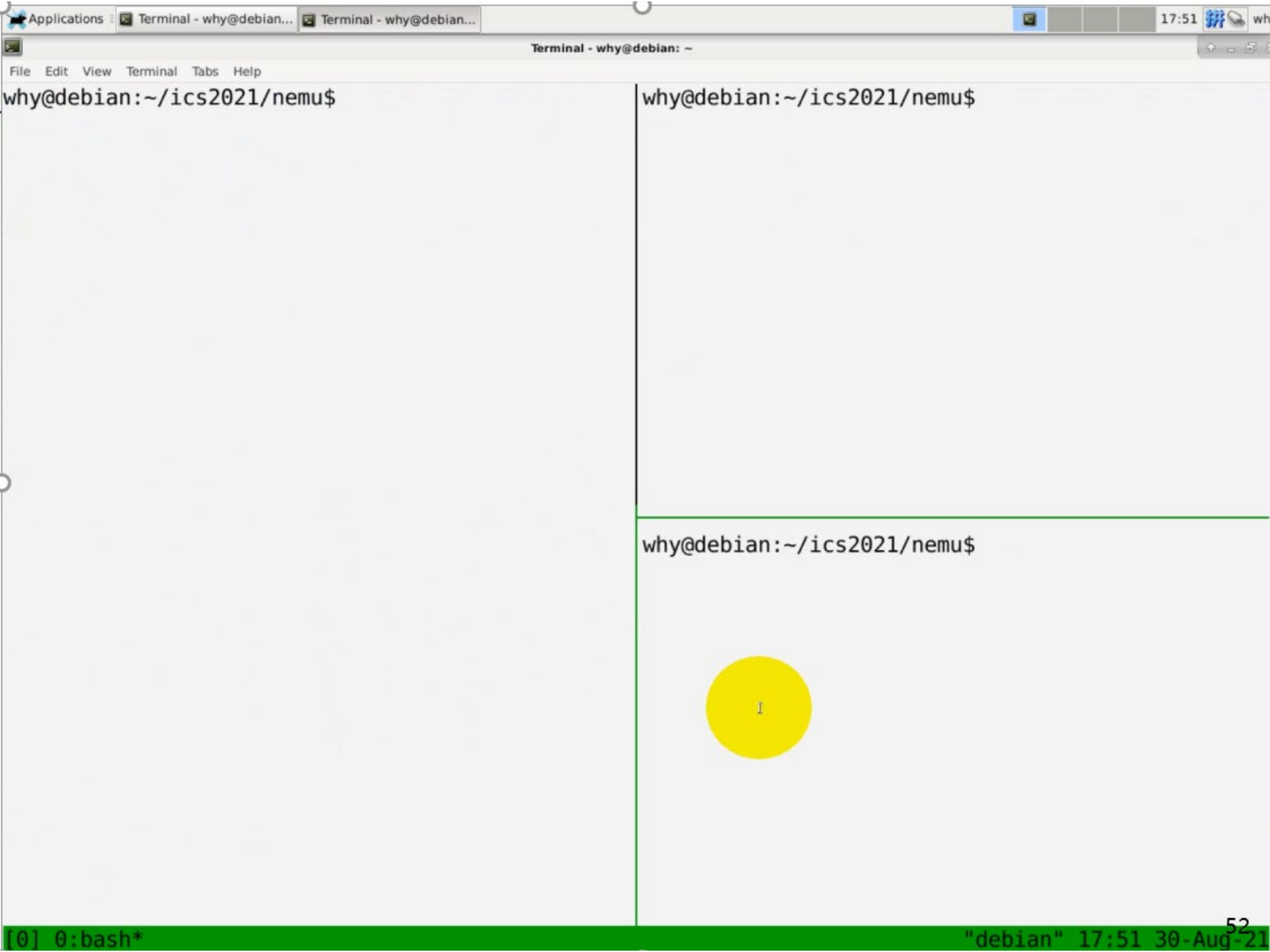

Applications
Terminal - why@debian...
15:29
why
Terminal - why@debian: ~/ics2021/nemu
File Edit View Terminal Tabs Help
$ ls
configs include Kconfig Makefile README.md resource scripts src tags tools
$

Vim：这都搞不定还引发什么编辑器圣战

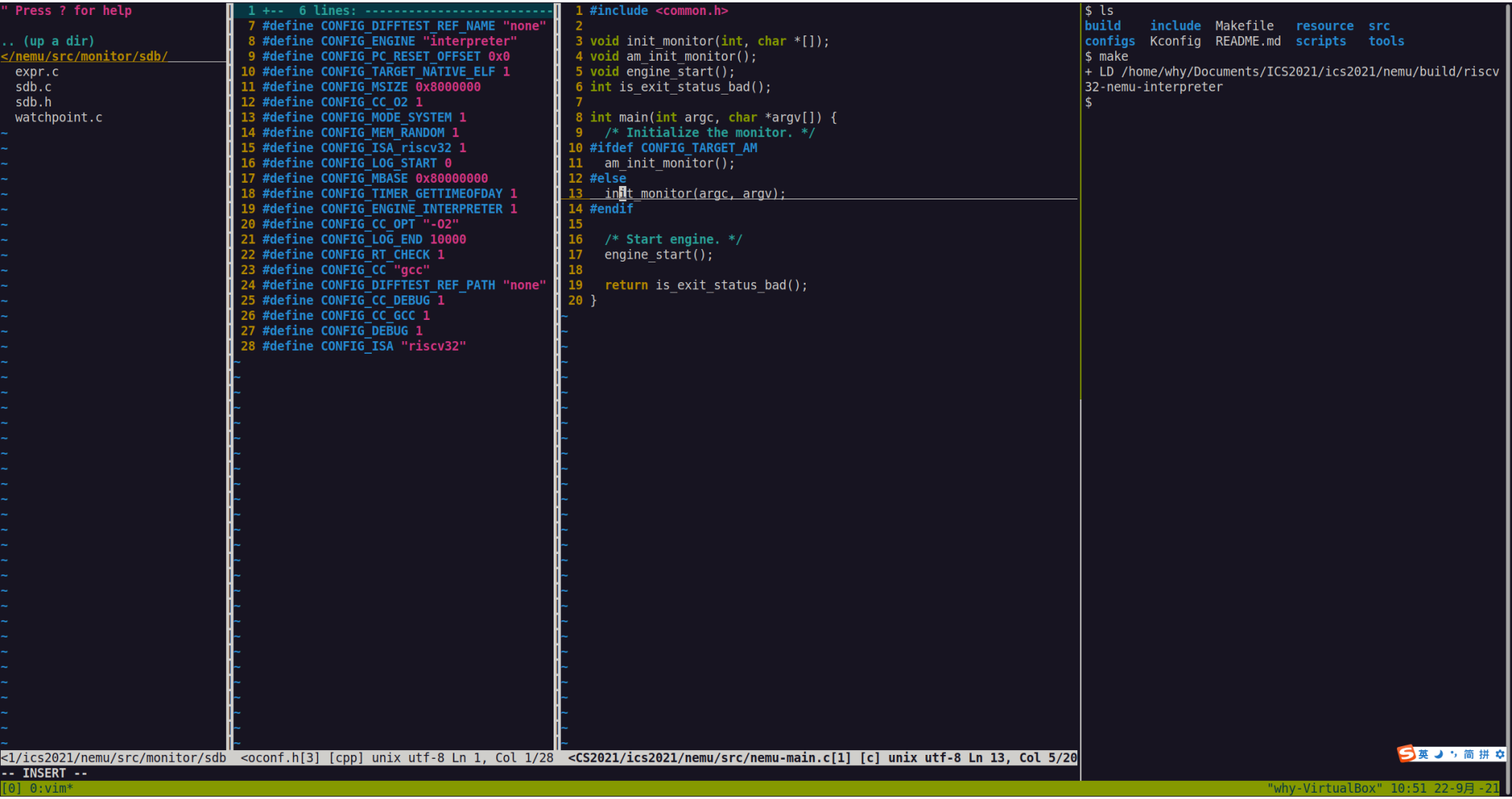
- Marks (文件内标记)
  - ma, 'a, mA, 'A, ...
- Tags (在位置之间跳转)
  - :jumps, C-], C-i, C-o, :tjump, ...
- Tabs/Windows (管理多文件)
  - :tabnew, gt, gT, ...
- Folding (浏览大代码)
  - zc, zo, zR, ...

- 更多的功能/插件：(RTFM, STFW)
  - vimtutor


version 1.1
April 1st, 06
vi / vim graphical cheat sheet
Esc
normal mode
~ toggle case
` goto mark
! external filter
1 [2]
@ play macro
2
prev ident
3
$ eol
4
% goto match
5
^ "soft" bol
6
& repeat :s
7
* next ident
8
(begin sentence
9
) end sentence
0 "hard" bol
_ "soft" bol down
- prev line
+ next line
= auto[3] format
Q ex mode
q record macro
W next WORD
w next word
E end WORD
e end word
R replace mode
r replace char
T back 'till
t 'till
Y yank line
y yank [1,3]
U undo line
u undo
I insert at bol
i insert mode
O open above
o open below
P paste before
p paste after [1]
{ begin parag.
[misc
} end parag.
] misc
A append at eol
a append
S subst line
s subst char
D delete to eol
d delete [1,3]
F "back" find ch
f find char
G eof/goto ln
g extra cmds [6]
H screen top
h
J join lines
j
K help
k
L screen bottom
l
: ex cmd line
; repeat t/T/f/F
" reg. [1] spec
' goto mk. bol
| bol/goto col
\ not used!
Z quit [4]
z extra cmds [5]
X back-space
x delete char
C change to eol
c change [1,3]
V visual lines
v visual mode
B prev WORD
b prev word
N prev (find)
n next (find)
M screen mid'l
m set mark
< un-indent [3]
, reverse t/T/f/F
> indent [3]
. repeat cmd
? find (rev.)
/ find
motion
moves the cursor, or defines the range for an operator
command
direct action command, if red, it enters insert mode
operator
requires a motion afterwards, operates between cursor & destination
extra
special functions, requires extra input
q·
commands with a dot need a char argument afterwards
bol = beginning of line, eol = end of line,
mk = mark, yank = copy
words: quux((foo, bar, baz);
WORDs: quux(foo, bar, baz);
Main command line commands ('ex'):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),
Other important comands:
CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)
Visual mode:
Move around and type operator to act on selected region (vim only)
Notes:
(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay$ to copy rest of line to reg 'a')
(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top, zb: bottom, zz: center
(6) gg: top of file (vim only), gf: open file under cursor (vim only)
For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

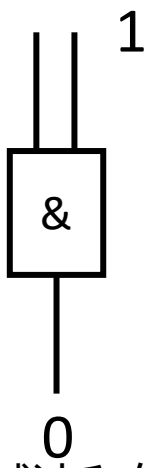

Applications
Terminal - why@debian...
Terminal - why@debian...
17:51
Terminal - why@debian: ~
File Edit View Terminal Tabs Help
why@debian:~/ics2021/nemu$
why@debian:~/ics2021/nemu$
why@debian:~/ics2021/nemu$
[0] 0:bash*
"debian" 17:51 30-Aug-21

如果你有块更大的屏幕


" Press ? for help
.. (up a dir)
</nemu/src/monitor/sdb/
expr.c
sdb.c
sdb.h
watchpoint.c
1 +-- 6 lines: ----------
7 #define CONFIG_DIFFTEST_REF_NAME "none"
8 #define CONFIG_ENGINE "interpreter"
9 #define CONFIG_PC_RESET_OFFSET 0x0
10 #define CONFIG_TARGET_NATIVE_ELF 1
11 #define CONFIG_MSIZE 0x8000000
12 #define CONFIG_CC_O2 1
13 #define CONFIG_MODE_SYSTEM 1
14 #define CONFIG_MEM_RANDOM 1
15 #define CONFIG_ISA_riscv32 1
16 #define CONFIG_LOG_START 0
17 #define CONFIG_MBASE 0x80000000
18 #define CONFIG_TIMER_GETTIMEOFDAY 1
19 #define CONFIG_ENGINE_INTERPRETER 1
20 #define CONFIG_CC_OPT "-O2"
21 #define CONFIG_LOG_END 10000
22 #define CONFIG_RT_CHECK 1
23 #define CONFIG_CC "gcc"
24 #define CONFIG_DIFFTEST_REF_PATH "none"
25 #define CONFIG_CC_DEBUG 1
26 #define CONFIG_CC_GCC 1
27 #define CONFIG_DEBUG 1
28 #define CONFIG_ISA "riscv32"
1 #include <common.h>
2
3 void init_monitor(int, char *[]);
4 void am_init_monitor();
5 void engine_start();
6 int is_exit_status_bad();
7
8 int main(int argc, char *argv[]) {
9 /* Initialize the monitor. */
10 #ifdef CONFIG_TARGET_AM
11 am_init_monitor();
12 #else
13 init_monitor(argc, argv);
14 #endif
15
16 /* Start engine. */
17 engine_start();
18
19 return is_exit_status_bad();
20 }
$ ls
build include Makefile resource src
configs Kconfig README.md scripts tools
$ make
+ LD /home/why/Documents/ICS2021/ics2021/nemu/build/riscv32-nemu-interpreter
$
<1/ics2021/nemu/src/monitor/sdb <oconf.h[3] [cpp] unix utf-8 Ln 1, Col 1/28 <CS2021/ics2021/nemu/src/nemu-main.c[1] [c] unix utf-8 Ln 13, Col 5/20
-- INSERT --
[0] 0:vim*
"why-VirtualBox" 10:51 22-9月 -21

Week6: 数据的机器级表示

- 逻辑门和导线是构成计算机 (组合逻辑电路) 的基本单元
  - 位运算是用电路最容易实现的运算
    - & (与), | (或), ~ (非)
    - ^ (异或)
    - << (左移位), >> (右移位)



  - 例子：一代传奇处理器 8-bit MOS 6502
    - 3510 晶体管；56 条指令，算数指令仅有加减法和位运算

Instructions by Name

ADC .... add with carry
AND .... and (with accumulator)
ASL .... arithmetic shift left
BCC .... branch on carry clear
BCS .... branch on carry set
BEQ .... branch on equal (zero set)
BIT .... bit test
BMI .... branch on minus (negative set)
BNE .... branch on not equal (zero clear)
BPL .... branch on plus (negative clear)
BRK .... break / interrupt

BVC .... branch on overflow clear
BVS .... branch on overflow set
CLC .... clear carry
CLD .... clear decimal
CLI .... clear interrupt disable
CLV .... clear overflow
CMP .... compare (with accumulator)
CPX .... compare with X
CPY .... compare with Y
DEC .... decrement
DEX .... decrement X
DEY .... decrement Y

Bit Set: 求 $|S|$ (S二进制表示有多少个1)

```
int bitset_size(uint32_t S) {
   int n;
   for (int i = 0; i < 32; i++) {
      n += bitset_contains(S, i);
   }
   return n;
}
```

```
int bitset_size1(uint32_t S) { // SIMD
   S = (S & 0x55555555) + ((S >> 1) & 0x55555555);
   S = (S & 0x33333333) + ((S >> 2) & 0x33333333);
   S = (S & 0x0F0F0F0F) + ((S >> 4) & 0x0F0F0F0F);
   S = (S & 0x00FF00FF) + ((S >> 8) & 0x00FF00FF);
   S = (S & 0x0000FFFF) + ((S >> 16) & 0x0000FFFF);
   return S;
}
```

Bit Set: 返回 $S \neq \emptyset$ 中的某个元素

- 有二进制数x = 0b+++++100，我们希望得到最后那个100
  - 想法：使用基本操作构造一些结果，能把+++++的部分给抵消掉

表达式	结果
x	0b+++++100
x-1	0b+++++011
~x	0b-----011
~x+1	0b-----100

- 一些有趣的式子：
  - x & (x-1) → 0b+++++000; x ^ (x-1) → 0b00000111;
  - x & (~x+1) → 0b00000100 (lowbit)
    - **x & -x**, (~x & (x-1))+ 1 都可以实现lowbit
    - 只遍历存在的元素可以加速求$|S|$

Bit Set：求$\lfloor \log_2(x) \rfloor$

- 等同于31−$\mathrm{clz}(x)$

```
int clz(uint32_t x) {
    int n = 0;
    if (x <= 0x0000ffff) n += 16, x <<= 16;
    if (x <= 0x00ffffff) n += 8, x <<= 8;
    if (x <= 0x0fffffff) n += 4, x <<= 4;
    if (x <= 0x3fffffff) n += 2, x <<= 2;
    if (x <= 0x7fffffff) n ++;
    return n;
}
```

x：0x00000001	16
x：0x00010000	24
x：0x01000000	28
x：0x03000000	30
$S = \{0\}$	31

- （奇怪的代码）假设x是lowbit的结果？

```
#define LOG2(x) \
    ("-01J2GK-3@HNL;-=47A-IFO?M:<6-E>95D8CB"[(x) % 37] - '0')
```

Undefined Behavior: 警惕整数溢出

表达式	值
`UINT_MAX+1`	0
`INT_MAX+1; LONG_MAX+1`	undefined
`char c = CHAR_MAX; c++;`	varies (???)
`1 << -1`	undefined
`1 << 0`	1
`1 << 31`	undefined
`1 << 32`	undefined
`1 / 0`	undefined
`INT_MAX % -1`	undefined

- W. Dietz, et al. Understanding integer overflow in C/C++. In *Proceedings of ICSE*, 2012.

整数溢出和编译优化

```
int f() { return 1 << -1; }
```

- 根据手册，这是个UB，于是clang这样处置

```
0000000000000000 <f>:
   0:   c3      retq
```

- 编译器把这个计算直接删除了

W. Xi, et al. Towards optimization-safe systems: Analyzing the impact of undefined behavior. In *Proceedings of SOSP*, 2013.

Bit Set：求$\lfloor \log_2(x) \rfloor$ (cont'd)

- 用一点点元编程（meta-programming）；试一试log2.c

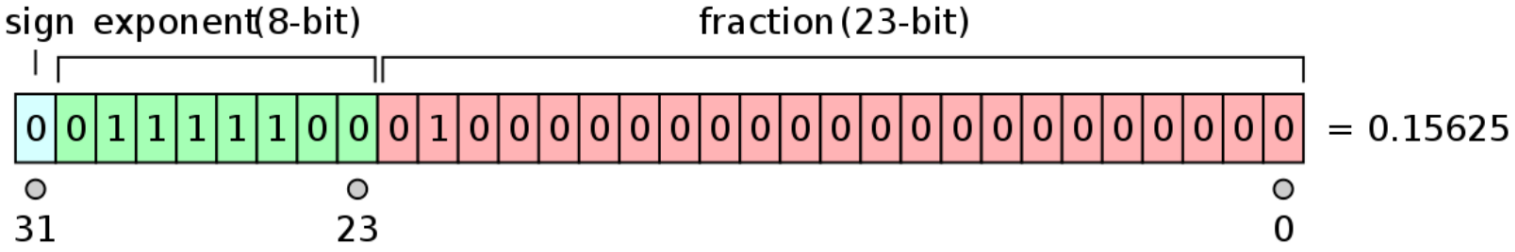
```
import json

n, base = 64, '0'
for m in range(n, 10000):
    if len({ (2**i) % m for i in range(n) }) == n:
        M = { j: chr(ord(base) + i)
              for j in range(0, m)
                  for i in range(0, n)
                      if (2**i) % m == j }
        break

magic = json.dumps(''.join(
    [ M.get(j, '-') for j in range(0, m) ]
    )).strip('"')

print(f'#define LOG2(x) ("{magic}"[(x) % {m}] - \'{base}\')')
```

实数的计算机表示

- 实数非常非常多
  - 只能用32/64-bit 01串来表述一小部分实数
    - 确定一种映射方法，把一个01串映射到一个实数
      - 运算起来不太麻烦
      - 计算误差不太可怕

- 于是有了IEEE754
  - 1bit S, 23/53bits Fraction (尾数), 8/11bits Exponent (阶码),

$$x = (-1)^S \times (1.F) \times 2^{E-B}$$



IEEE754: 你有可能不知道的事实

- 一个有关浮点数大小/密度的实验(float.c)
- 越大的数字，距离下一个实数的距离就越大
  - 可能会带来相当的绝对误差
  - 因此很多数学库都会频繁做归一化

- 例子：计算$1+\frac{1}{2}+\frac{1}{3}...+\frac{1}{n}$

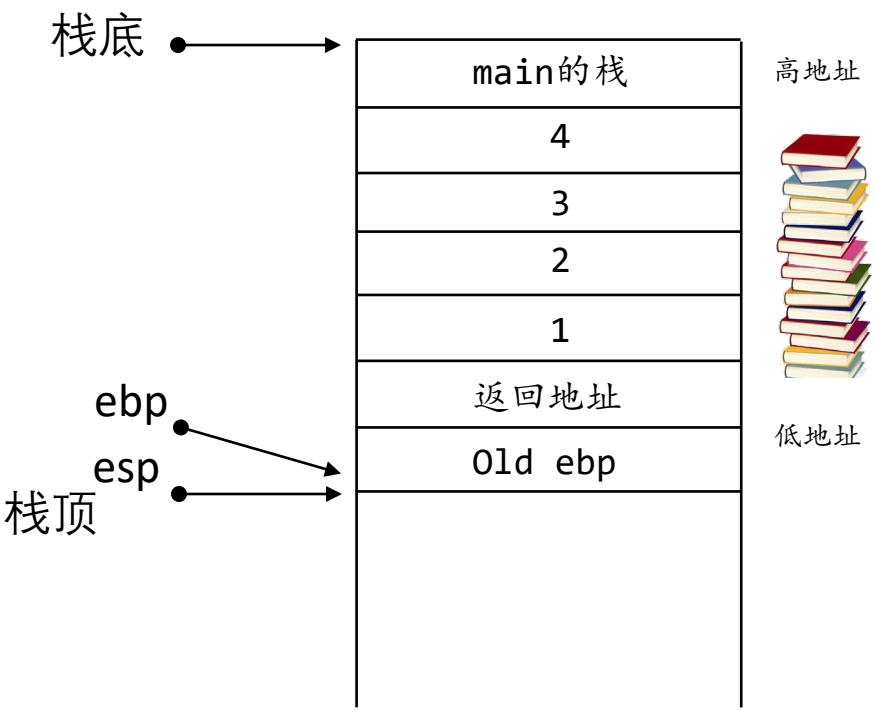
```
#define SUM(T, st, ed, d) ({ \
    T s = 0; \
    for (int i = st; i != ed + d; i += d) \
        s += (T)1 / i; \
    s; \
})
```

Week7: x86-64与内联汇编

```
int _cdecl foo(int m1, int m2, int m3, int m4);
```

```
int foo(int m1, int m2, int m3, int m4) {
    return m1 + m2 + m3 + m4;
}
```

```
int main() { foo(1,2,3,4);}
```



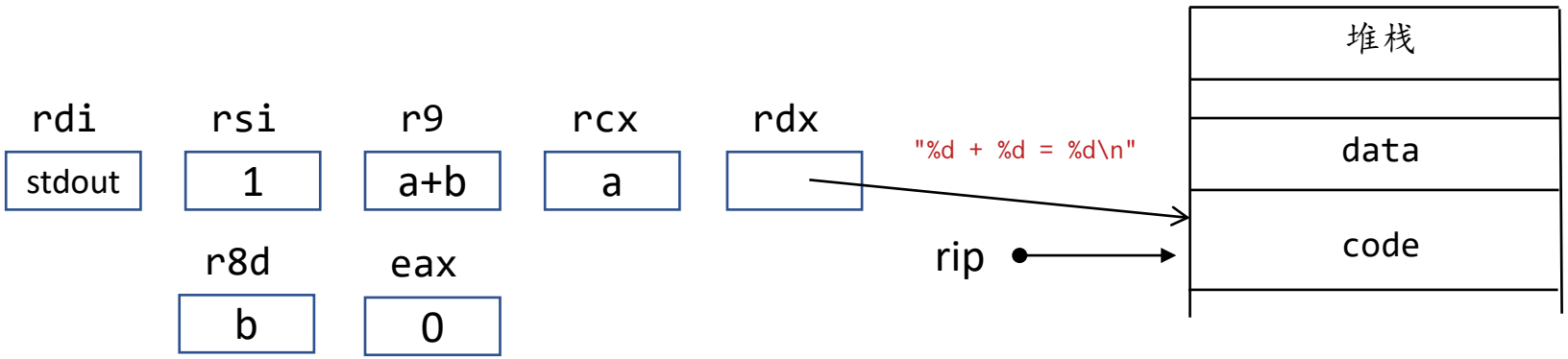

```
Disassembly of section .text:

00000000 <foo>:
   0:   f3 0f 1e fb             endbr32
   4:   55                      push   %ebp
   5:   89 e5                   mov    %esp,%ebp
   7:   e8 fc ff ff ff          call   8 <foo+0x8>
   c:   05 01 00 00 00          add    $0x1,%eax
  11:   8b 55 08                mov    0x8(%ebp),%edx
  14:   8b 45 0c                mov    0xc(%ebp),%eax
  17:   01 c2                   add    %eax,%edx
  19:   8b 45 10                mov    0x10(%ebp),%eax
  1c:   01 c2                   add    %eax,%edx
  1e:   8b 45 14                mov    0x14(%ebp),%eax
  21:   01 d0                   add    %edx,%eax
  23:   5d                      pop    %ebp
  24:   c3                      ret

00000025 <main>:
  25:   f3 0f 1e fb             endbr32
  29:   55                      push   %ebp
  2a:   89 e5                   mov    %esp,%ebp
  2c:   e8 fc ff ff ff          call   2d <main+0x8>
  31:   05 01 00 00 00          add    $0x1,%eax
  36:   6a 04                   push   $0x4
  38:   6a 03                   push   $0x3
  3a:   6a 02                   push   $0x2
  3c:   6a 01                   push   $0x1
  3e:   e8 fc ff ff ff          call   3f <main+0x1a>
  43:   83 c4 10                add    $0x10,%esp
  46:   b8 00 00 00 00          mov    $0x0,%eax
  4b:   c9                      leave
  4c:   c3                      ret
```

cdecl函数调用惯例

```c
void plus(int a, int b) {
   fprintf(stdout, "%d + %d = %d\n", a, b, a + b);
}
```

- 实际调用的是 __fprintf_chk@plt
  - 需要传递的参数：stdout, %d + %d = %d\n, a, b, a + b
  - calling convention: rdi, rsi, rdx, rcx, r8, r9

```
0000000000000700 <plus>:
 700: 44 8d 0c 37                lea (%rdi,%rsi,1),%r9d
 704: 89 f9                      mov %edi,%ecx
 706: 48 8b 3d 03 09 20 00       mov 0x200903(%rip),%rdi # 201010 <stdout@@GLIBC_2.2.5>
 70d: 48 8d 15 b0 00 00 00       lea 0xb0(%rip),%rdx # 7c4 <_IO_stdin_used+0x4>
 714: 41 89 f0                   mov %esi,%r8d
 717: 31 c0                      xor %eax,%eax
 719: be 01 00 00 00             mov $0x1,%esi
 71e: e9 5d fe ff ff             jmpq 580 <__fprintf_chk@plt>
```



在汇编中访问 C 世界的表达式

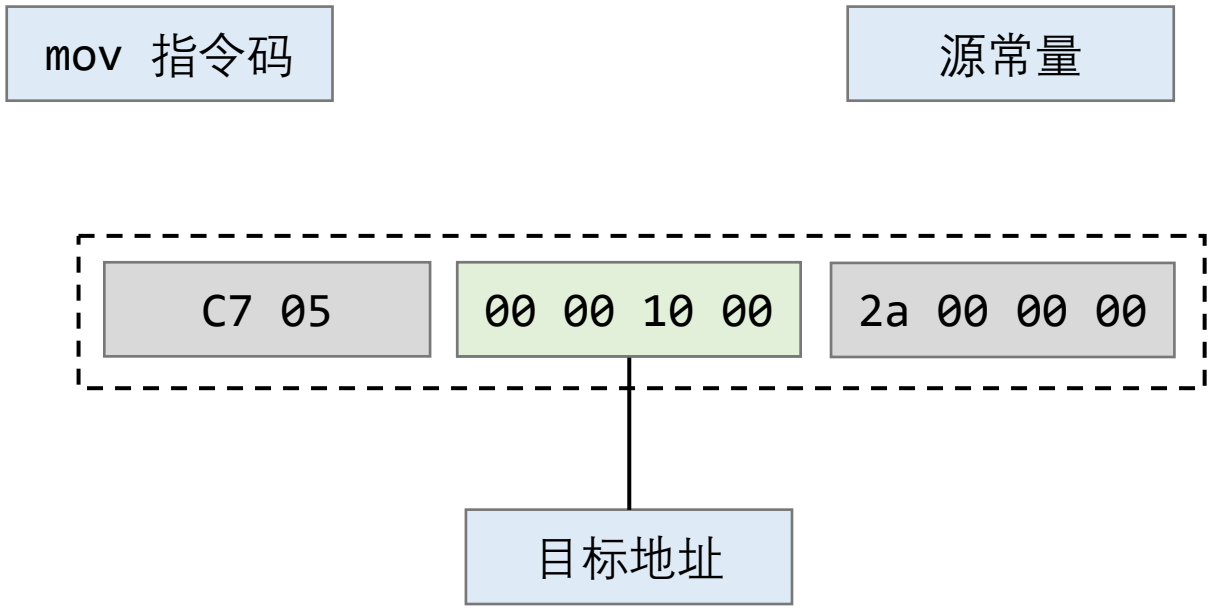
```c
int foo(int x, int y) {
    int z;
    x++; y++;
    asm (
        "addl %1, %2; "
        "movl %2, %0; "
        : "=r"(z) // output
        : "r"(x), "r"(y) // input
    );
    return z;
}
```

gcc -O0

```
0000000000000000 <foo>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   89 7d ec                mov    %edi,-0x14(%rbp)
   b:   89 75 e8                mov    %esi,-0x18(%rbp)
   e:   83 45 ec 01             addl   $0x1,-0x14(%rbp)
  12:   83 45 e8 01             addl   $0x1,-0x18(%rbp)
  16:   8b 45 ec                mov    -0x14(%rbp),%eax
  19:   8b 55 e8                mov    -0x18(%rbp),%edx
  1c:   01 c2                   add    %eax,%edx
  1e:   89 d0                   mov    %edx,%eax
  20:   89 45 fc                mov    %eax,-0x4(%rbp)
  23:   8b 45 fc                mov    -0x4(%rbp),%eax
  26:   5d                      pop    %rbp
  27:   c3                      ret
```

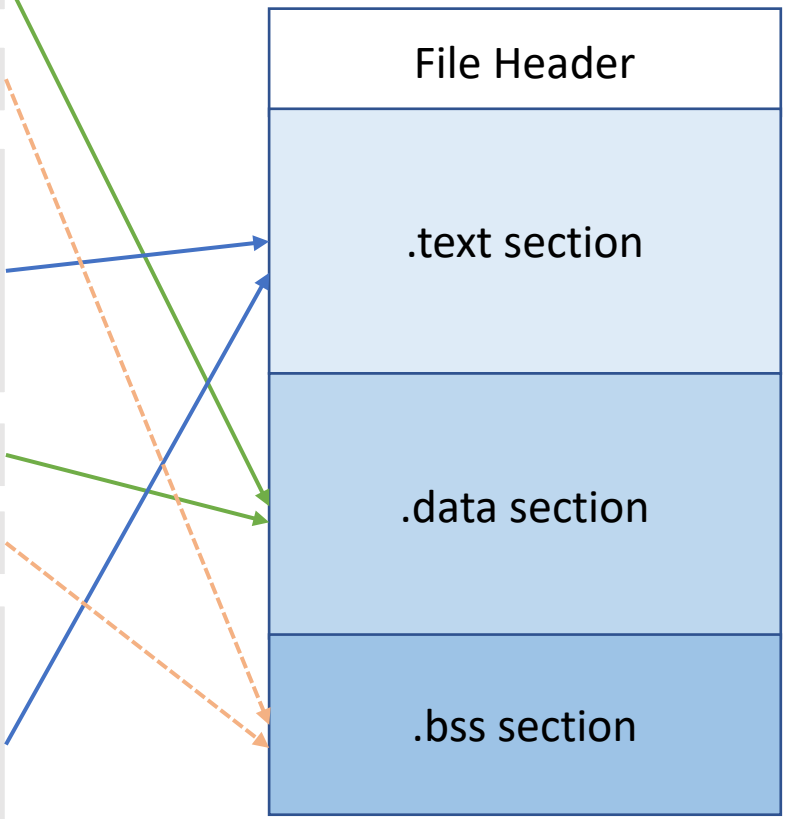
Week8: 链接与加载选讲

- A: 定义var (0x1000)
- B: movl $0x2a, var

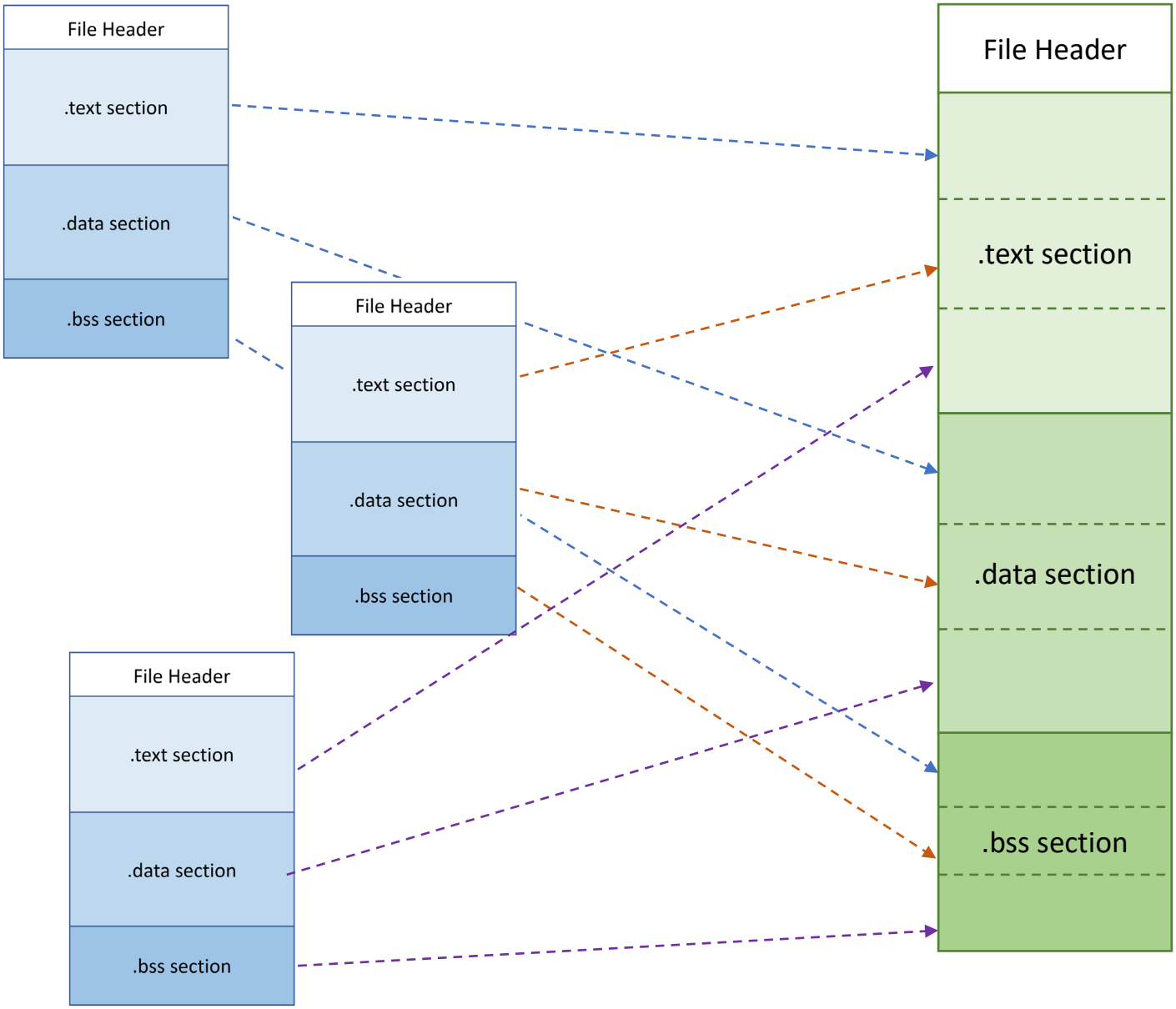


可执行文件格式之ELF

```
int global_int_var = 84;

int global_int_var2;

void func1(int i){
     printf("%d\n", i);
}
int main(void){

     static int static_var = 85;

     static int static_var2;

     int a = 1;
     int b;
     func1( static_var + static_var2
     + a + b);
     return 0;
}
```



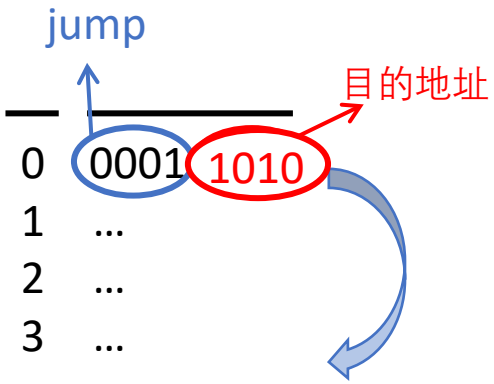
链接多个.o


File Header
.text section
.data section
.bss section
File Header
.text section
.data section
.bss section
File Header
.text section
.data section
.bss section
File Header
.text section
.data section
.bss section

链接 Two-pass linking

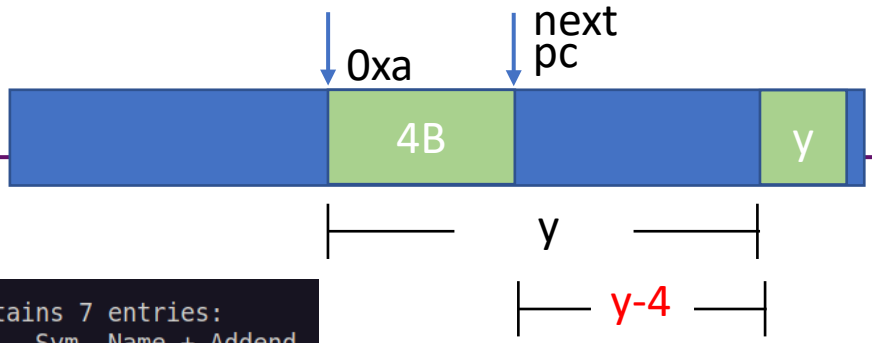
- 空间和地址的分配
  - 重新建立符号表，合并段，并计算段长度建立映射关系

- 符号解析和重定位
  - 如何填空?



10 1000 0111
11 …

填什么？为什么y - 4？

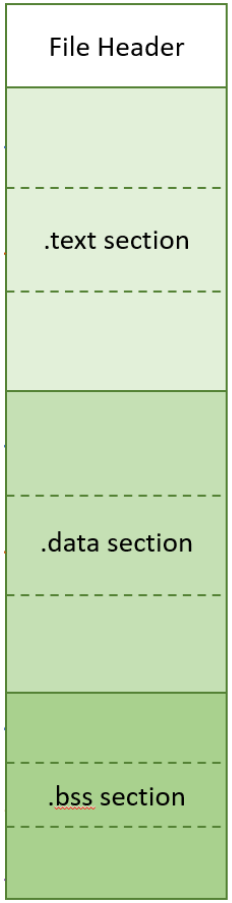


- readelf –a main.o

```
Relocation section '.rela.text.startup' at offset 0x208 contains 7 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000000a  000500000002 R_X86_64_PC32     0000000000000000 y - 4
000000000010  000600000002 R_X86_64_PC32     0000000000000000 x - 4
000000000015  000700000004 R_X86_64_PLT32    0000000000000000 foo - 4
00000000001b  000500000002 R_X86_64_PC32     0000000000000000 y - 4
000000000021  000600000002 R_X86_64_PC32     0000000000000000 x - 4
000000000026  00020000000a R_X86_64_32       0000000000000000 .rodata.str1.1 + 0
000000000035  000800000004 R_X86_64_PLT32    0000000000000000 __printf_chk - 4
```

- objdump –d main.o

```
Disassembly of section .text.startup:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   48 83 ec 08             sub    $0x8,%rsp
   8:   8b 35 00 00 00 00       mov    0x0(%rip),%esi        # e <main+0xe>
   e:   8b 3d 00 00 00 00       mov    0x0(%rip),%edi        # 14 <main+0x14>
  14:   e8 00 00 00 00          call   19 <main+0x19>
  19:   8b 0d 00 00 00 00       mov    0x0(%rip),%ecx        # 1f <main+0x1f>
  1f:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 25 <main+0x25>
  25:   be 00 00 00 00          mov    $0x0,%esi
  2a:   41 89 c0                mov    %eax,%r8d
  2d:   bf 01 00 00 00          mov    $0x1,%edi
  32:   31 c0                   xor    %eax,%eax
  34:   e8 00 00 00 00          call   39 <main+0x39>
  39:   31 c0                   xor    %eax,%eax
  3b:   48 83 c4 08             add    $0x8,%rsp
  3f:   c3                      ret
```



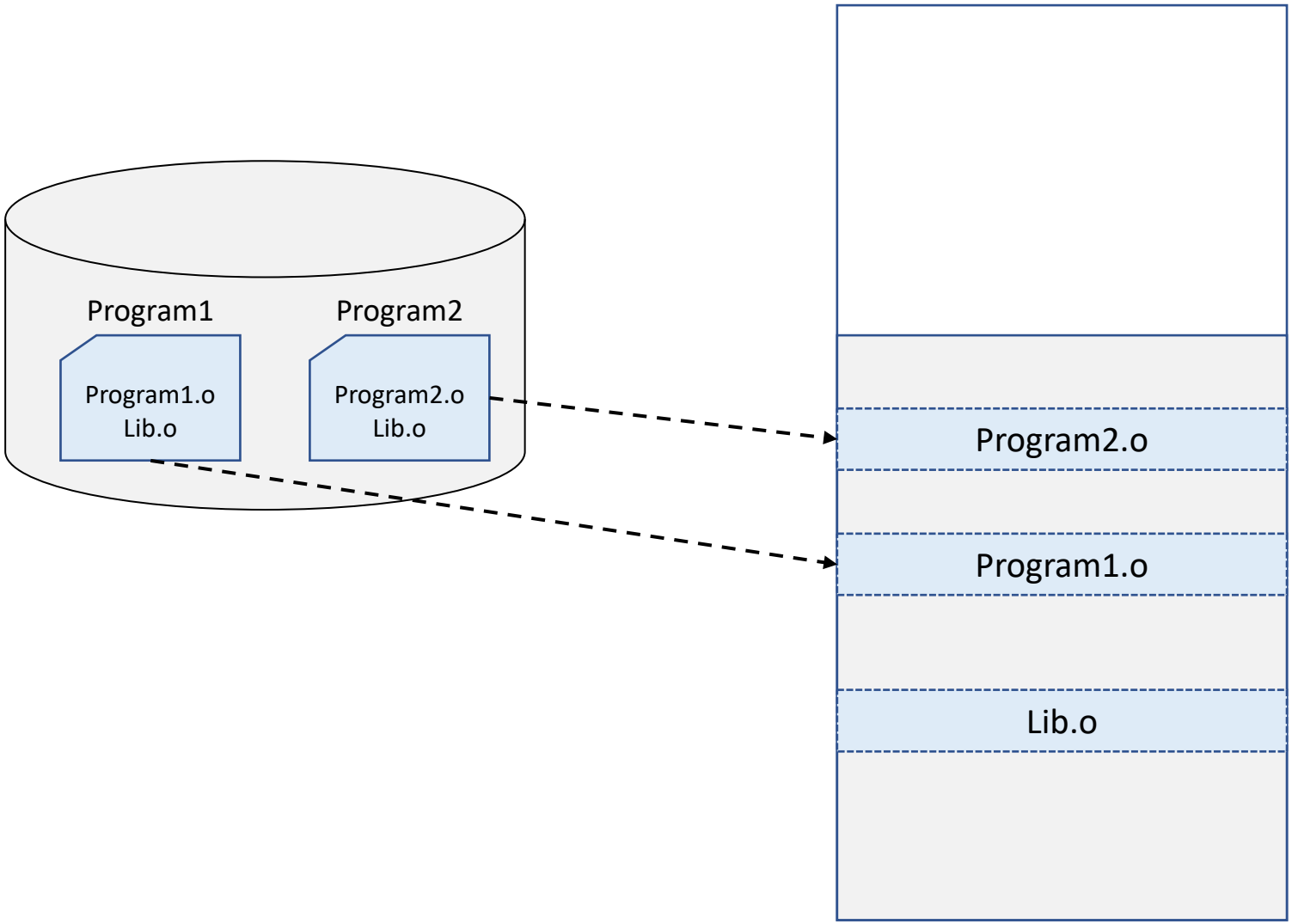
gcc和ld链接

不能用ld链接么？

- man gcc

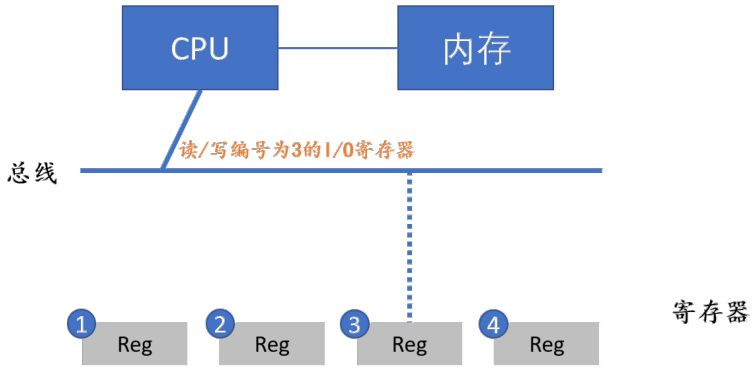
```
       to write -Xlinker "-assert definitions", because this passes the entire string as a
       single argument, which is not what the linker expects.

       When using the GNU linker, it is usually more convenient to pass arguments to linker
       options using the option=value syntax than as separate arguments.  For example, you
       can specify -Xlinker -Map=output.map rather than -Xlinker -Map -Xlinker output.map.
       Other linkers may not support this syntax for command-line options.

   -Wl,option
       Pass option as an option to the linker.  If option contains commas, it is split into
       multiple options at the commas.  You can use this syntax to pass an argument to the
       option.  For example, -Wl,-Map,output.map passes -Map output.map to the linker.  When
       using the GNU linker, you can also get the same effect with -Wl,-Map=output.map.

       NOTE: In Ubuntu 8.10 and later versions, for LDFLAGS, the option -Wl,-z,relro is used.
       To disable, use -Wl,-z,norelro.

   -u symbol
       Pretend the symbol symbol is undefined, to force linking of library modules to define
       it.  You can use -u multiple times with different symbols to force loading of
       additional library modules.

   -z keyword
       -z is passed directly on to the linker along with the keyword keyword. See the section
       in the documentation of your linker for permitted values and their meanings.

Options for Directory Search
   These options specify directories to search for header files, for libraries and for parts
   of the compiler:

   -I dir
   -iquote dir
Manual page gcc(1) line 11096/23056 53% (press h for help or q to quit)
```

为什么要动态链接?

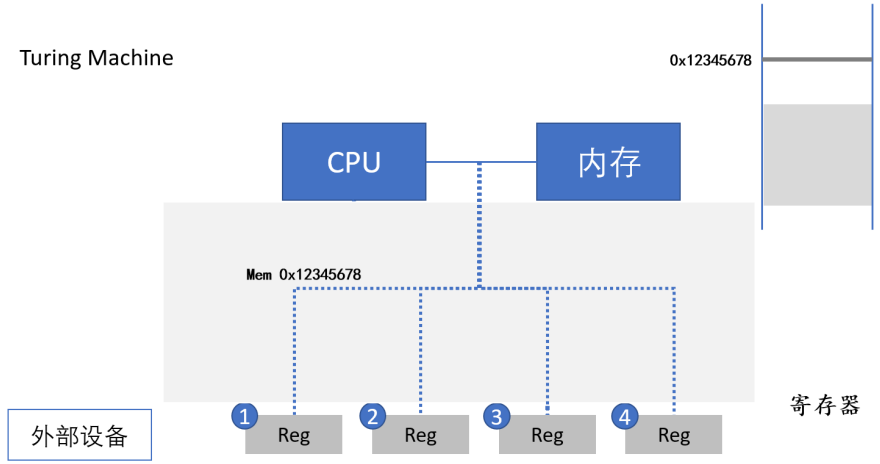

Program1
Program2
Program1.o
Lib.o
Program2.o
Lib.o
Program2.o
Program1.o
Lib.o

Week10: I/O设备选讲

- CPU可以直接通过指令读写这些寄存器
  - Port-mapped I/O (PMIO)
    - I/O地址空间（port）
    - CPU直连I/O总线



  - Memory-mapped I/O (MMIO)
    - 直观：使用普通内存读写指令就能访问
    - 带来了一些设计和实现的麻烦：编译器优化、缓存、乱序执行

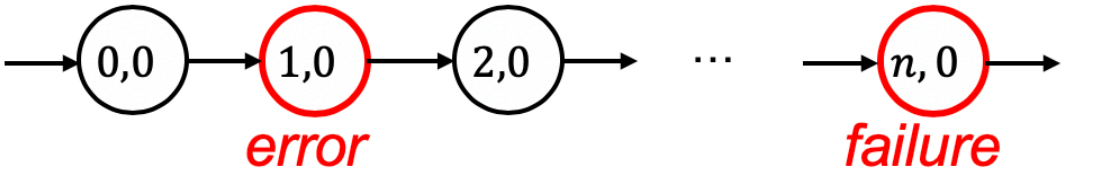


Week11: 调试理论

- 因为 bug 的触发经历了漫长的过程
  - 需求 → 设计 → 代码 → Fault (bug) → Error (程序状态错) → Failure
    - 我们只能观测到 failure (可观测的结果错)
    - 我们可以检查状态的正确性 (但非常费时)
    - 无法预知 bug 在哪里 (每一行 “看起来” 都挺对的)

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; i++)
    ...
```

fault 程序bug

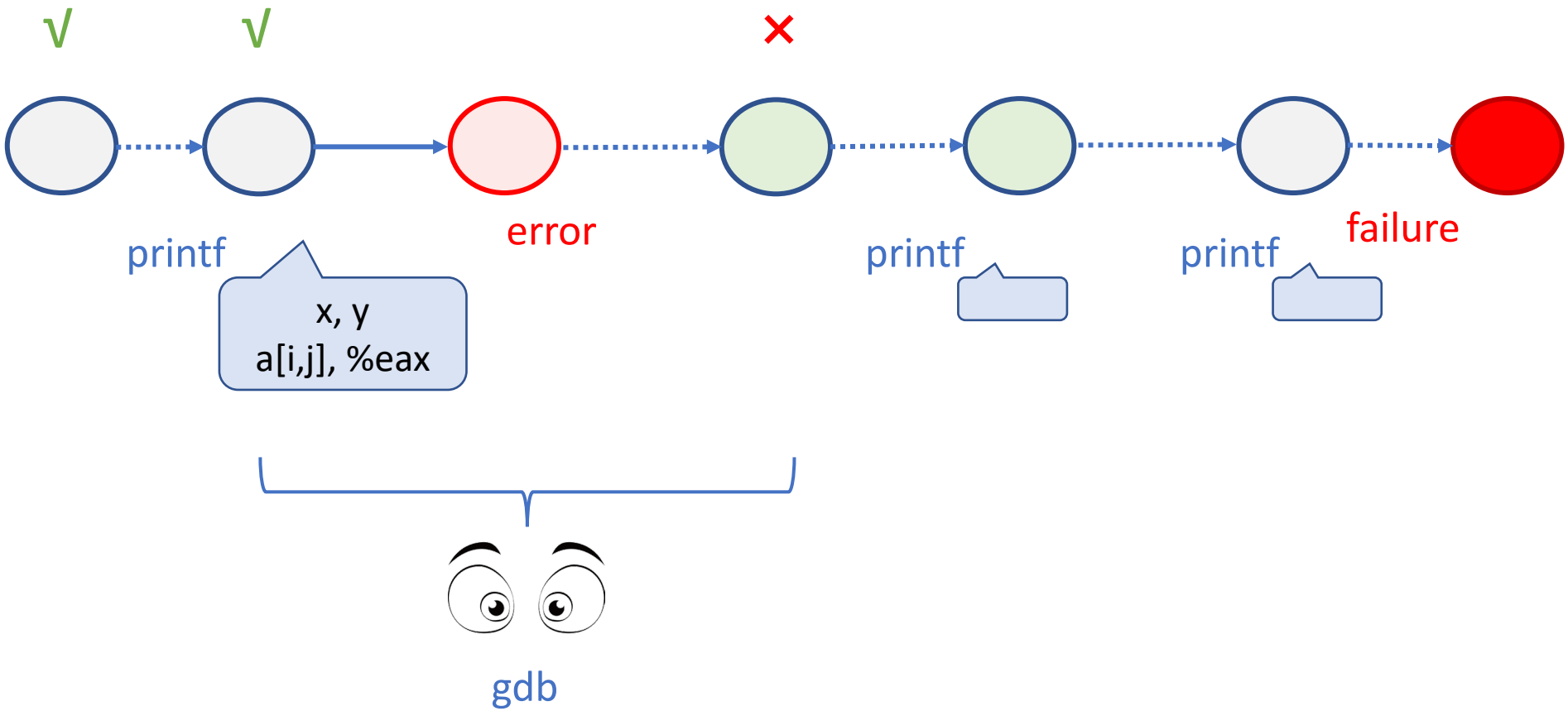


状态违反spec

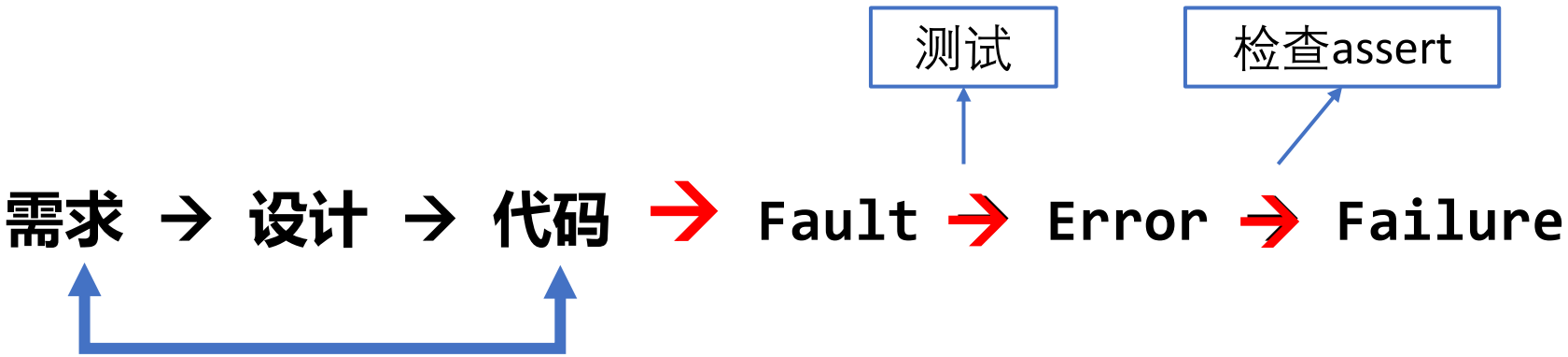
可观测的timeout

调试理论（cont'd）

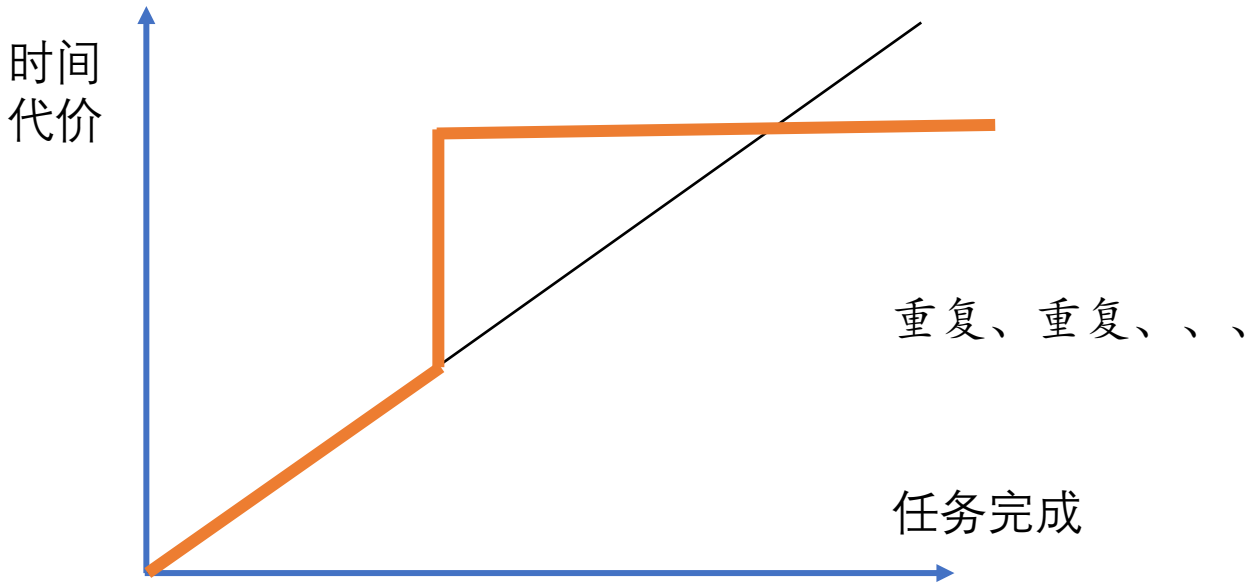
- 尝试接近状态的判定



调试理论：



Week12: 系统编程与基础设施

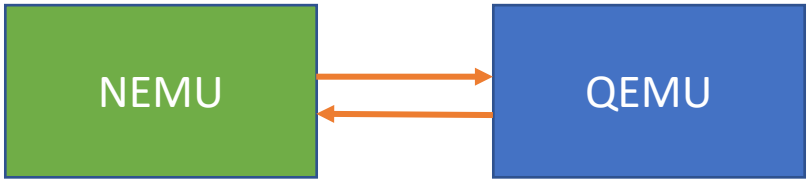


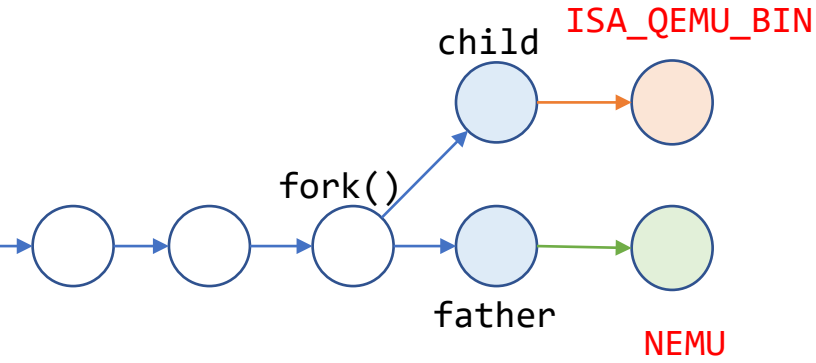
Differential Testing

> “同一套接口 (API) 的两个实现应当行为完全一致”

- 大腿同学 & 小腿同学：指令集的两套实现
  - 还有什么现实中软件系统的例子?
- 你能找到两份独立实现的东西，都可以测试
  - 浏览器 Mesbah and Prasad, ICSE'11
  - GCC (vs clang), Yang et al., PLDI'11
  - 文件系统, Min et al., SOSP'15
  - 数据库 Rigger and Su, OSDI'20
  - Gcov (vs llvm-cov)真的能在 gcc/llvm 里发现很多 bugs
  - DL library Pham et al., ICSE’19

NEMU和QEMU协作

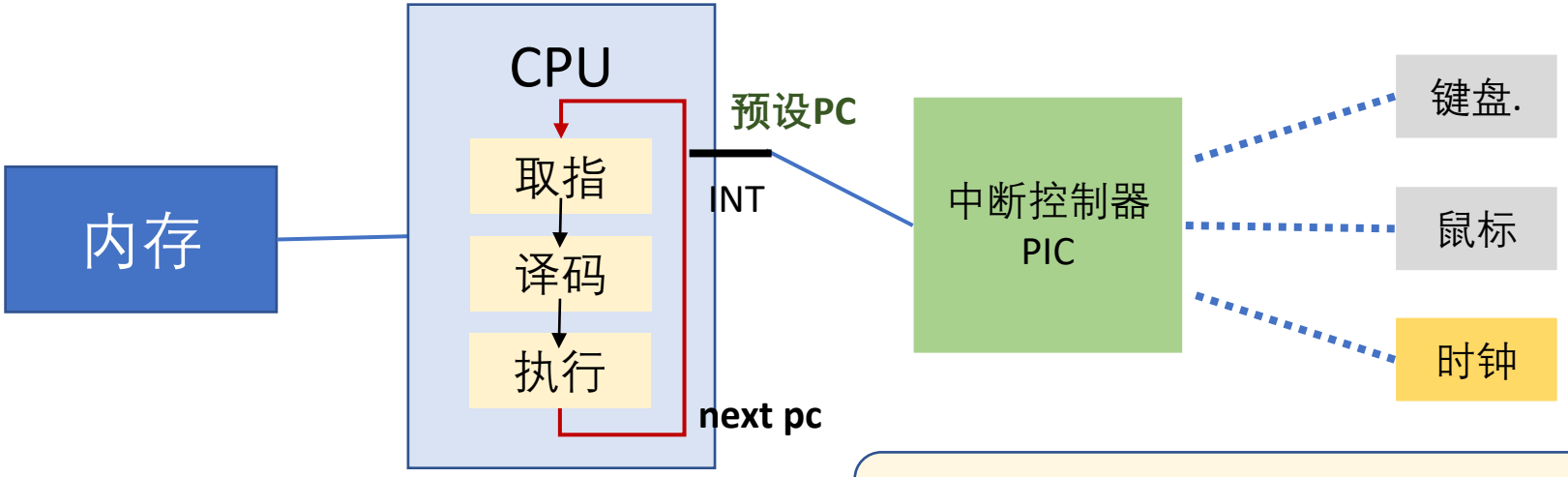





```
 9 #define ISA_QEMU_BIN "qemu-system-riscv32"
10 #define ISA_QEMU_ARGS "-bios", "none",
11 #elif defined(CONFIG_ISA_riscv64)
12 #define ISA_QEMU_BIN "qemu-system-riscv64"
```

```
38 void difftest_init(int port) {
39   char buf[32];
40   sprintf(buf, "tcp::%d", port);
41
42   int ppid_before_fork = getpid();
43   int pid = fork();
44   if (pid == -1) {
45     perror("fork");
46     assert(0);
47   }
48   else if (pid == 0) {
49     // child
50
51     // install a parent death signal in the chlid
52     int r = prctl(PR_SET_PDEATHSIG, SIGTERM);
53     if (r == -1) {
54       perror("prctl error");
55       assert(0);
56     }
57
58     if (getppid() != ppid_before_fork) {
59       printf("parent has died!\n");
60       assert(0);
61     }
62
63     close(STDIN_FILENO);
64     execlp(ISA_QEMU_BIN, ISA_QEMU_BIN, ISA_QEMU_ARGS "-S", "-gdb", buf,
65         "-serial", "none", "-monitor", "none", NULL);
66     perror("exec");
67     assert(0);
68   }
69   else {
70     // father
71
72     gdb_connect_qemu(port);
73     printf("Connect to QEMU with %s successfully\n", buf);
74
75     atexit(gdb_exit);
76
77     init_isa();
78   }
79 }
```

Week13: 中断与分时多任务



Hz

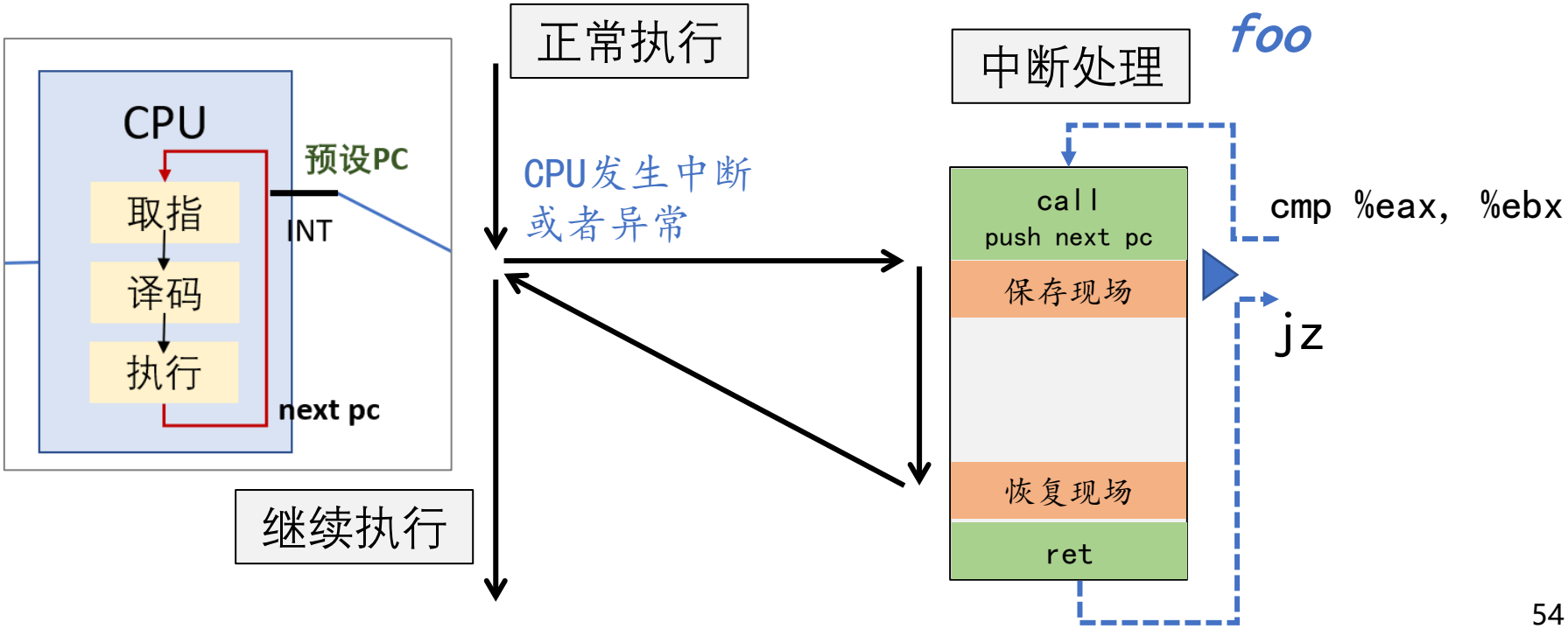
```
counter --;
if(counter == 0){
    counter = C;
    INT = ¬INT;
}
```

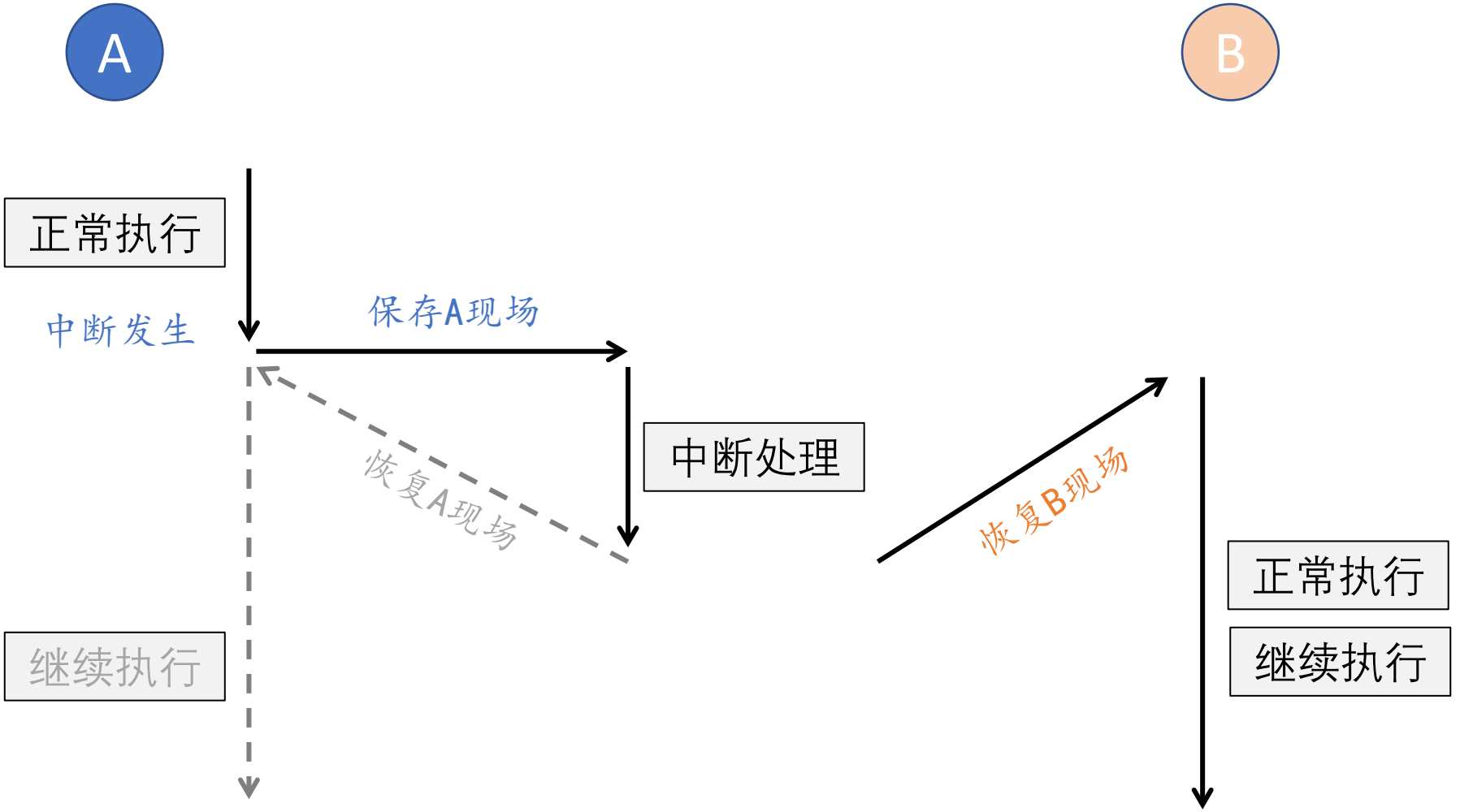
- 因为有中断
  - 相当于在指令之后插入一段代码

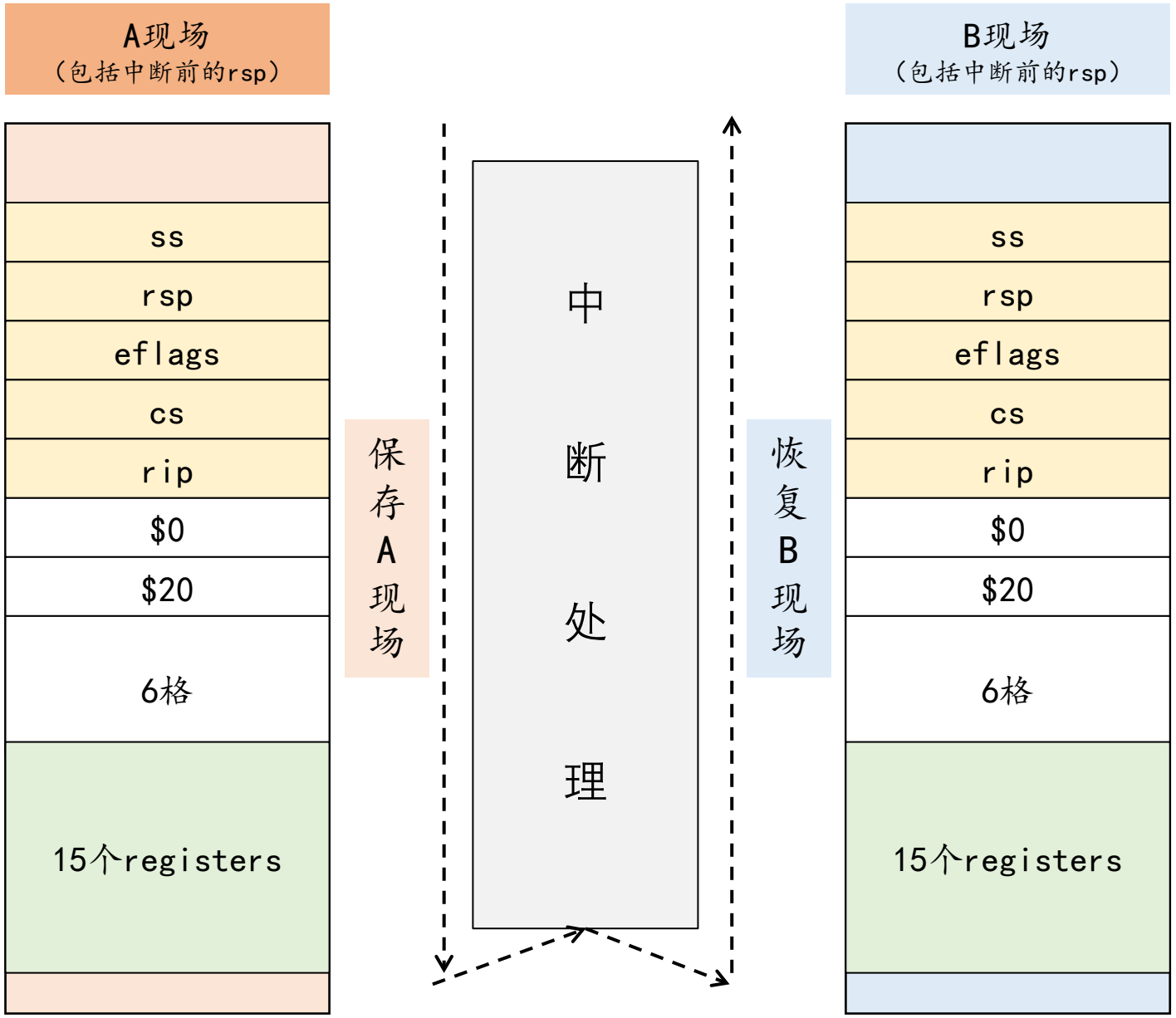
```
if (has_interrupt && int_enabled) {
   interrupt_handler();
}
```

中断的实现

- 比 “函数调用” 复杂一些
  - 函数调用需要保存 PC 到堆栈 (%rsp)
  - 但中断不仅要保存 PC (尤其是在有特权级切换的时候)
- 中断处理
  - 自动保存 RIP, CS, RFLAGS, RSP, SS, (Error Code)




A
B
正常执行
中断发生
保存A现场
中断处理
恢复A现场
恢复B现场
正常执行
继续执行
继续执行


A现场
（包括中断前的rsp）
ss
rsp
eflags
cs
rip
$0
$20
6格
15个registers
保存A现场
中
断
处
理
恢复B现场
B现场
（包括中断前的rsp）
ss
rsp
eflags
cs
rip
$0
$20
6格
15个registers

中断 + 更大的内存 = 分时多线程

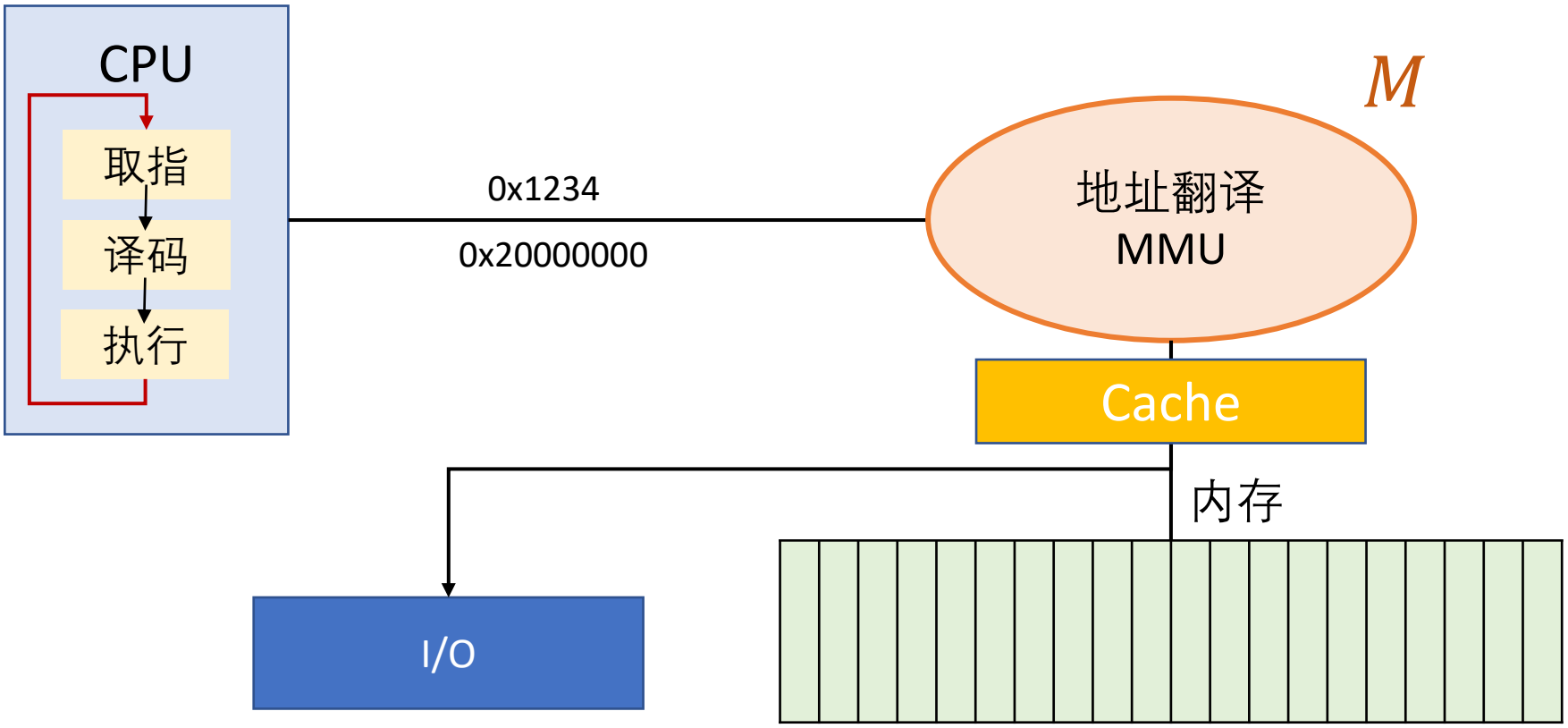
```
void foo() { while (1) printf("a"); }
void bar() { while (1) printf("b"); }
```

- 能够让foo()和bar()“同时”在处理器上执行?
  - 借助每条语句后被动插入的interrupt_handler()调用

```
void interrupt_handler() {
   dump_regs(current->regs);
   current = (current->func == foo) ? bar : foo;
   restore_regs(current->regs);
}
```

  - 操作系统背负了“调度”的职责

Week14: 虚拟存储选讲

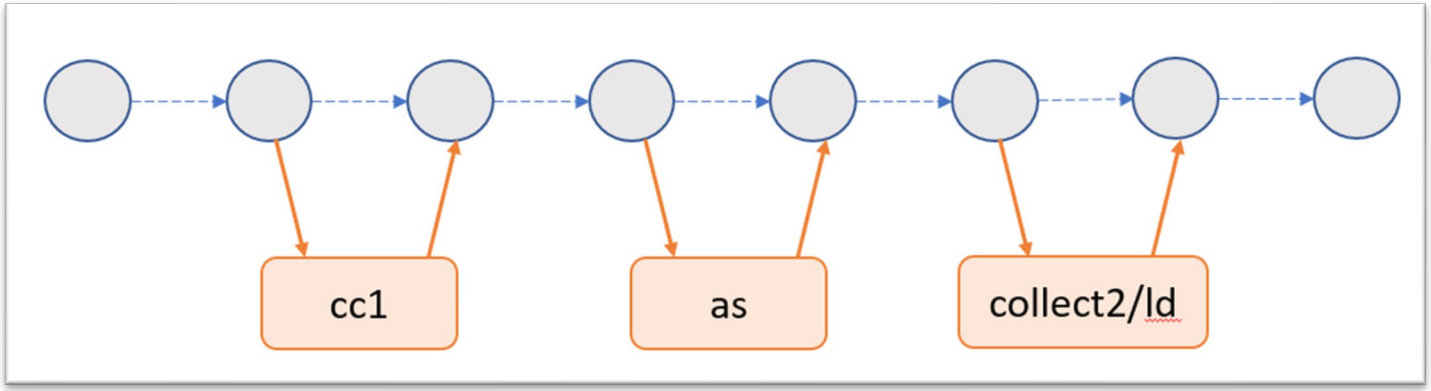


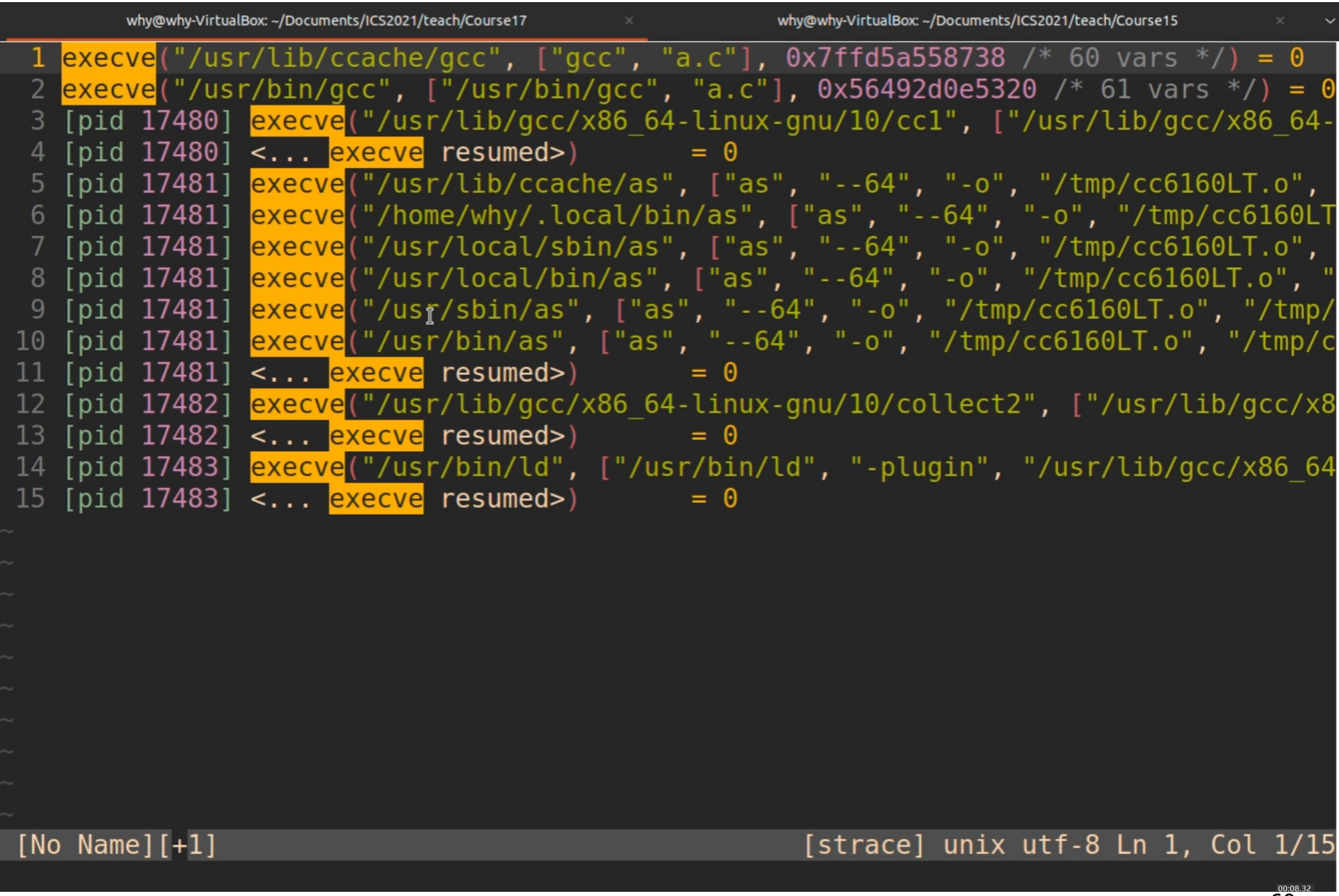

```
volatile int *p = ……;
for (int i = 1; i < 1000; i++)
   *p = 1;
```

W15：造轮子的方法和乐趣

- 在IDE里，为什么按一个键，就能够编译运行?
  - 编译、链接
    - `.c` → 预编译 → `.i` → 编译 → `.s` → 汇编 → `.o` → 链接 → `a.out`
  - 加载执行
    - `./a.out`

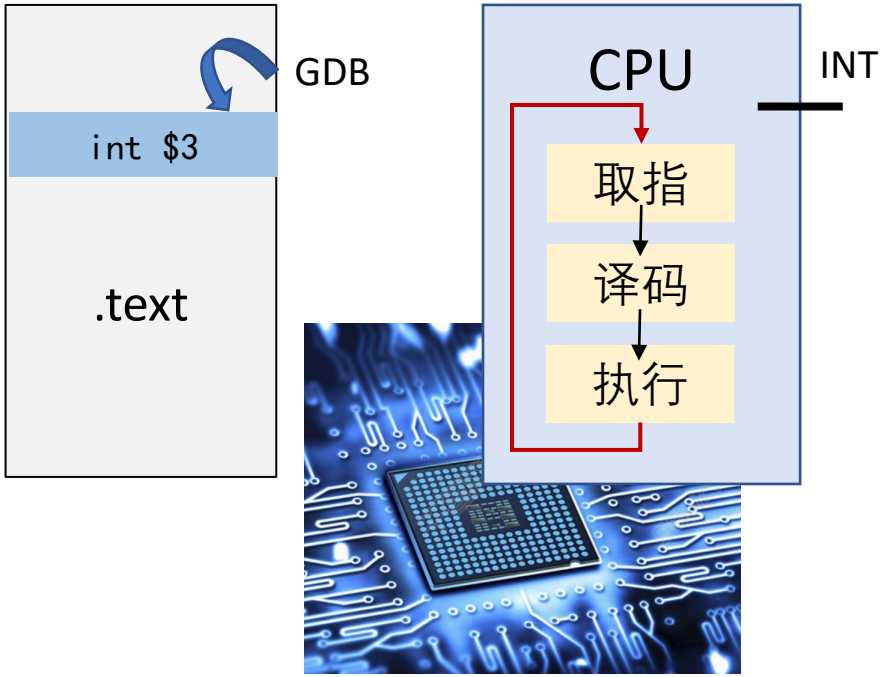
- 现在，一位同学对这个过程提出了质疑
  - 我不信！我就觉得是 gcc 一个程序直接搞定的
    - 道理上完全可以这么实现
  - 如何说服这位同学?

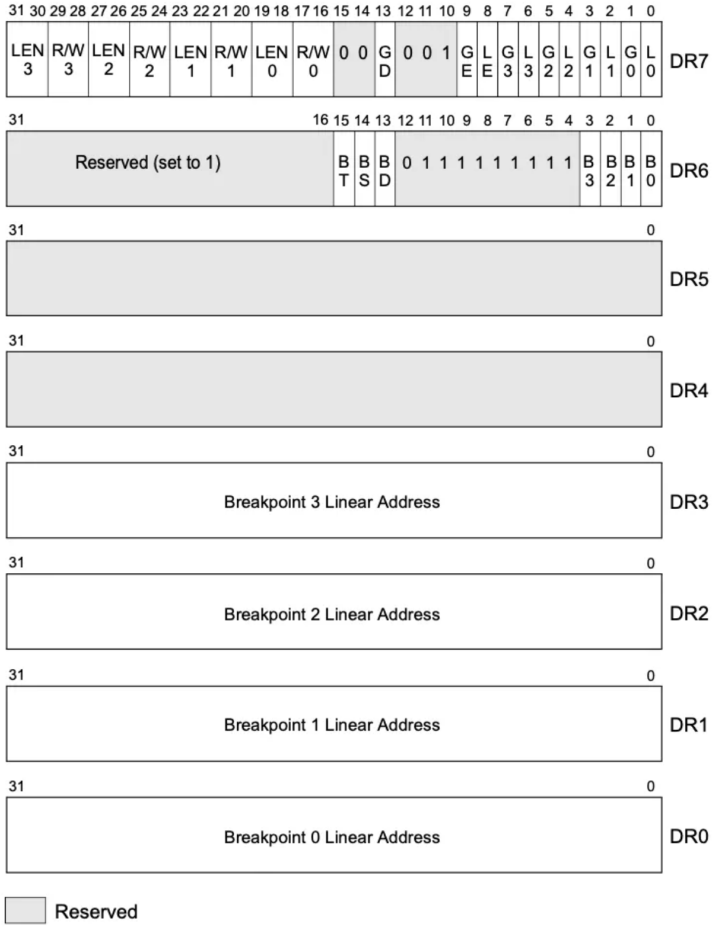



why@why-VirtualBox: ~/Documents/ICS2021/teach/Course17
why@why-VirtualBox: ~/Documents/ICS2021/teach/Course15
execve("/usr/lib/ccache/gcc", ["gcc", "a.c"], 0x7ffd5a558738 /* 60 vars */) = 0
execve("/usr/bin/gcc", ["/usr/bin/gcc", "a.c"], 0x56492d0e5320 /* 61 vars */) = 0
[pid 17480] execve("/usr/lib/gcc/x86_64-linux-gnu/10/cc1", ["/usr/lib/gcc/x86_64-
[pid 17480] <... execve resumed>) = 0
[pid 17481] execve("/usr/lib/ccache/as", ["as", "--64", "-o", "/tmp/cc6160LT.o",
[pid 17481] execve("/home/why/.local/bin/as", ["as", "--64", "-o", "/tmp/cc6160LT
[pid 17481] execve("/usr/local/sbin/as", ["as", "--64", "-o", "/tmp/cc6160LT.o",
[pid 17481] execve("/usr/local/bin/as", ["as", "--64", "-o", "/tmp/cc6160LT.o", "
[pid 17481] execve("/usr/sbin/as", ["as", "--64", "-o", "/tmp/cc6160LT.o", "/tmp/
[pid 17481] execve("/usr/bin/as", ["as", "--64", "-o", "/tmp/cc6160LT.o", "/tmp/c
[pid 17481] <... execve resumed>) = 0
[pid 17482] execve("/usr/lib/gcc/x86_64-linux-gnu/10/collect2", ["/usr/lib/gcc/x8
[pid 17482] <... execve resumed>) = 0
[pid 17483] execve("/usr/bin/ld", ["/usr/bin/ld", "-plugin", "/usr/lib/gcc/x86_64
[pid 17483] <... execve resumed>) = 0
[No Name][+1]
[strace] unix utf-8 Ln 1, Col 1/15

调试器GDB

- 好奇它是怎么实现的?
- 考虑核心功能：在任意 PC 的断点
- 其他功能都可以基于断点实现
  - 单步调试：在下一条指令打断点
  - watch point: 单步调试 + 检查条件





Week16：性能优化

- 理解程序如何编译+执行
- 理解现代处理器和存储系统
- 如何诊断性能瓶颈和提高性能
- 编译器的一些局限，理解编译器的优化痛点
  - 编译器优化前提是*safe*
    - Within procedure analyses
    - Static information analyses
    - “Conservative” to be “safe”

- Two optimization blocker
  - Memory aliasing
  - Function calls

示例程序1.2

```c
//a.c
#include <stdio.h>
void f1 (int *xp, int *yp){
        *xp += *yp;
        *xp += *yp;
}
```

```c
//b.c
#include <stdio.h>
void f1 (int *xp, int *yp){
        *xp += 2* *yp;
}
```

如果xp和yp指向同一块内存地址?

```c
        *xp += *xp;
        *xp += *xp;
```

```c
        *xp += 2* *xp;
```

```c
        *xp = 4 * *xp
```

```c
        *xp = 3 * *xp
```

Memory aliasing

```c
#include <stdio.h>

// Function to change the value of
// ptr1 and ptr2
int foo(int* ptr1, long* ptr2)
{
    *ptr1 = 10;
    *ptr2 = 11.0;
    return *ptr1;
}

// Driver Code
int main()
{
    long data = 100.0;

    // Function Call
    int result = foo((int *)&data, &data);

    // Print result
    printf("%d \n", result);
    return 0;
}
```

gcc -O2 a.c -fno-strict-aliasing

```
0000000000001149 <foo>:
    1149:       f3 0f 1e fa             endbr64
    114d:       c7 07 0a 00 00 00       movl   $0xa,(%rdi)
    1153:       48 c7 06 0b 00 00 00    movq   $0xb,(%rsi)
    115a:       8b 07                   mov    (%rdi),%eax
    115c:       c3                      ret
```

gcc -O1 a.c

```
0000000000001180 <foo>:
    1180:       f3 0f 1e fa             endbr64
    1184:       c7 07 0a 00 00 00       movl   $0xa,(%rdi)
    118a:       b8 0a 00 00 00          mov    $0xa,%eax
    118f:       48 c7 06 0b 00 00 00    movq   $0xb,(%rsi)
    1196:       c3                      ret
```

gcc -O2 a.c

```
  -fstack-reuse=[all|named_vars|none]   all
  -fstdarg-opt                          [enabled]
  -fstore-merging                       [enabled]
  -fstrict-aliasing                     [enabled]
  -fstrict-enums                        [available in C++, ObjC++]
  -fstrict-volatile-bitfields           [enabled]
  -fthread-jumps                        [enabled]
```

Inline substitution

```
long f();

long func1(){
    return f() + f() + f() + f();
}

long func2(){
    return 4 * f();
}
```

gcc -O0 a.c

gcc -O1 a.c

gcc -O2 a.c -fno-inline

```
0000000000000000 <func1>:
   0:
   4:
   5:   48 89 e5                mov    %rsp,%rbp
   8:   53                      push   %rbx
   9:   48 83 ec 08             sub    $0x8,%rsp
   d:   b8 00 00 00 00          mov    $0x0,%eax
  12:   e8 00 00 00 00          call   17 <func1+0x17>
  17:   48 89 c3                mov    %rax,%rbx
  1a:   b8 00 00 00 00          mov    $0x0,%eax
  1f:   e8 00 00 00 00          call   24 <func1+0x24>
  24:   48 01 c3                add    %rax,%rbx
  27:   b8 00 00 00 00          mov    $0x0,%eax
  2c:   e8 00 00 00 00          call   31 <func1+0x31>
  31:   48 01 c3                add    %rax,%rbx
  34:   b8 00 00 00 00          mov    $0x0,%eax
  39:   e8 00 00 00 00          call   3e <func1+0x3e>
  3e:   48 01 d8                add    %rbx,%rax
  41:   48 8b 5d f8             mov    -0x8(%rbp),%rbx
  45:   c9                      leave
  46:   c3                      ret
```

gcc -O2 a.c

```
0000000000000000 <func1>:
   0:   f3 0f 1e fa             endbr64
   4:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # b <func1+0xb>
   b:   48 8d 50 04             lea    0x4(%rax),%rdx
   f:   48 8d 04 85 06 00 00    lea    0x6(,%rax,4),%rax
  16:   00
  17:   48 89 15 00 00 00 00    mov    %rdx,0x0(%rip)        # 1e <func1+0x1e>
  1e:   c3                      ret
  1f:   90                      nop
```

超标量处理器

Superscalar Issue (Pentium)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$	Fetch	Decode	Wait			Execute	Write		
$Instr_3$		Fetch	Decode	Execute	Write				
$Instr_4$		Fetch	Decode	Wait			Execute	Write	
$Instr_5$			Fetch	Decode	Execute	Write			
$Instr_6$			Fetch	Decode	Execute	Write			
$Instr_7$				Fetch	Decode	Execute	Write		
$Instr_8$				Fetch	Decode	Execute	Write		

- Superscalar issue allows multiple instructions to be issued at the same time

分支预测

- 预测分支的跳转

```
404663: mov $0x0, %eax
404668: cmp (%rdi), %rsi
40466b: jge 404685
40466d: mov 0x8 (%rdi), %rax
……
404685: repz retq
```

Executing

How to continue?

  - 本质：猜测并预测分支的走向

  - java - Why is processing a sorted array faster than processing an unsorted array? - Stack Overflow

这学期学了什么?

`printf()` 在计算机软件/硬件系统上，各自发生了什么？

- CPU (NEMU) 始终在执行指令
  - (用户代码执行)
  - 库函数 (Navy, ...)
  - 系统调用 wrapper
  - `syscall / int $0x80`
  - (操作系统代码)
  - 中断/异常处理程序 (AbstractMachine)
  - 系统调用实现

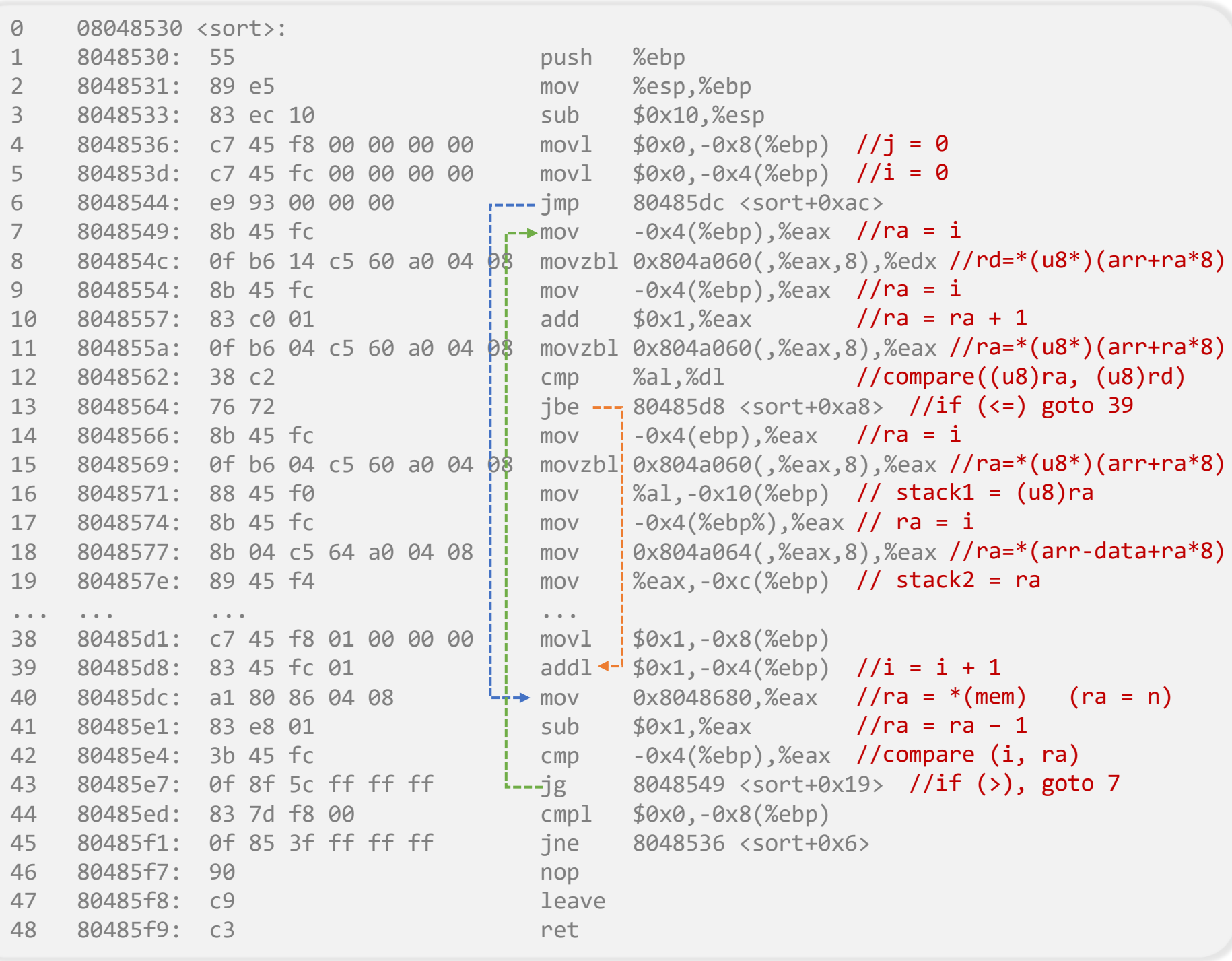
福利：如何阅读汇编代码

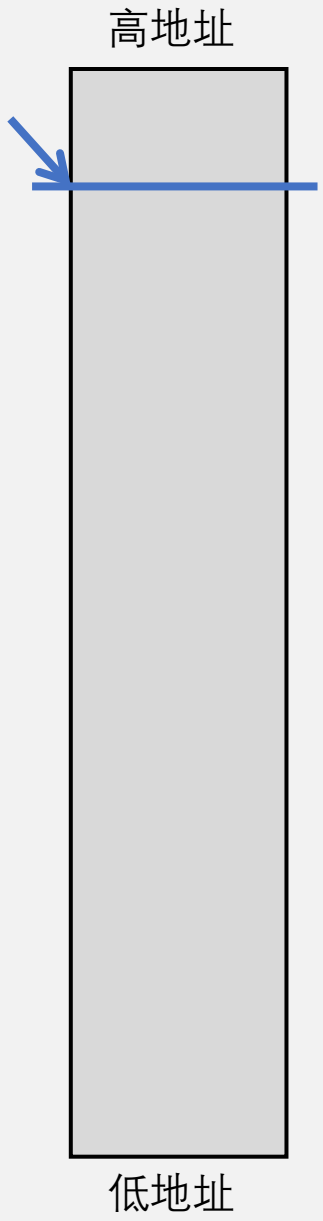
期末试题 (2018)

- 期末真题 dump.txt

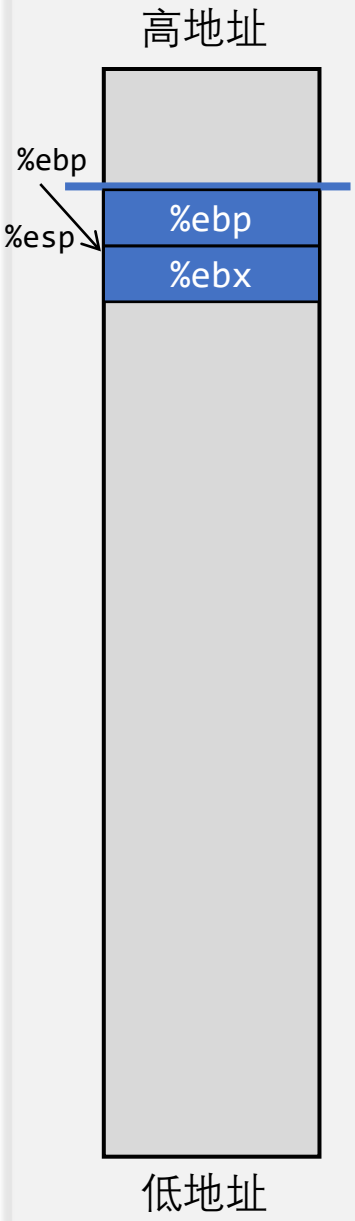
- 一些观察
  - 试卷同时提供 C 和对应的汇编代码
    - 并不严格需要 (可以用来检查你的理解是否准确)
  - 编译不带优化 (默认 -O0)
    - 大量冗余的视觉干扰
  - “7 ± 2 法则”
    - 除非刻意训练，否则人的短时记忆能力非常有限
      - George A. Miller, 1956
  - 抱怨做不完？先花 20 分钟把代码改写一下

```
0    08048530 <sort>:
1    8048530:  55                        push   %ebp
2    8048531:  89 e5                     mov    %esp,%ebp
3    8048533:  83 ec 10                  sub    $0x10,%esp
4    8048536:  c7 45 f8 00 00 00 00      movl   $0x0,-0x8(%ebp)
5    804853d:  c7 45 fc 00 00 00 00      movl   $0x0,-0x4(%ebp)
6    8048544:  e9 93 00 00 00            jmp    80485dc <sort+0xac>
7    8048549:  8b 45 fc                  mov    -0x4(%ebp),%eax
8    804854c:  0f b6 14 c5 60 a0 04 08   movzbl 0x804a060(,%eax,8),%edx
9    8048554:  8b 45 fc                  mov    -0x4(%ebp),%eax
10   8048557:  83 c0 01                  add    $0x1,%eax
11   804855a:  0f b6 04 c5 60 a0 04 08   movzbl 0x804a060(,%eax,8),%eax
12   8048562:  38 c2                     cmp    %al,%dl
13   8048564:  76 72                     jbe    80485d8 <sort+0xa8>
14   8048566:  8b 45 fc                  mov    -0x4(%ebp),%eax
15   8048569:  0f b6 04 c5 60 a0 04 08   movzbl 0x804a060(,%eax,8),%eax
16   8048571:  88 45 f0                  mov    %al,-0x10(%ebp)
17   8048574:  8b 45 fc                  mov    -0x4(%ebp),%eax
18   8048577:  8b 04 c5 64 a0 04 08      mov    0x804a064(,%eax,8),%eax
19   804857e:  89 45 f4                  mov    %eax,-0xc(%ebp)
...  ...       ...                       ...
38   80485d1:  c7 45 f8 01 00 00 00      movl   $0x1,-0x8(%ebp)
39   80485d8:  83 45 fc 01               addl   $0x1,-0x4(%ebp)
40   80485dc:  a1 80 86 04 08            mov    0x8048680,%eax
41   80485e1:  83 e8 01                  sub    $0x1,%eax
42   80485e4:  3b 45 fc                  cmp    -0x4(%ebp),%eax
43   80485e7:  0f 8f 5c ff ff ff         jg     8048549 <sort+0x19>
44   80485ed:  83 7d f8 00               cmpl   $0x0,-0x8(%ebp)
45   80485f1:  0f 85 3f ff ff ff         jne    8048536 <sort+0x6>
46   80485f7:  90                        nop
47   80485f8:  c9                        leave
48   80485f9:  c3                        ret
```

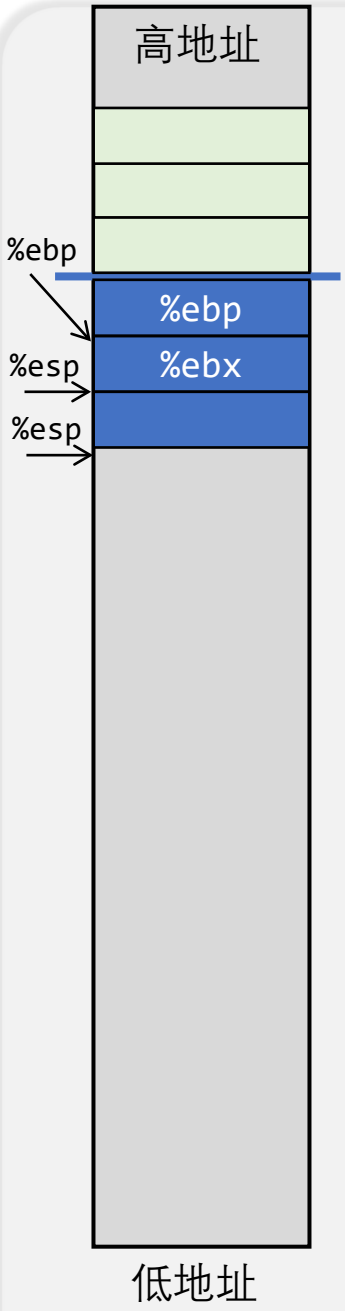

0 08048530 <sort>:
1 8048530: 55 push %ebp
2 8048531: 89 e5 mov %esp,%ebp
3 8048533: 83 ec 10 sub $0x10,%esp
4 8048536: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%ebp) //j = 0
5 804853d: c7 45 fc 00 00 00 00 movl $0x0,-0x4(%ebp) //i = 0
6 8048544: e9 93 00 00 00 jmp 80485dc <sort+0xac>
7 8048549: 8b 45 fc mov -0x4(%ebp),%eax //ra = i
8 804854c: 0f b6 14 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%edx //rd=*(u8*)(arr+ra*8)
9 8048554: 8b 45 fc mov -0x4(%ebp),%eax //ra = i
10 8048557: 83 c0 01 add $0x1,%eax //ra = ra + 1
11 804855a: 0f b6 04 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%eax //ra=*(u8*)(arr+ra*8)
12 8048562: 38 c2 cmp %al,%dl //compare((u8)ra, (u8)rd)
13 8048564: 76 72 jbe 80485d8 <sort+0xa8> //if (<=) goto 39
14 8048566: 8b 45 fc mov -0x4(ebp),%eax //ra = i
15 8048569: 0f b6 04 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%eax //ra=*(u8*)(arr+ra*8)
16 8048571: 88 45 f0 mov %al,-0x10(%ebp) // stack1 = (u8)ra
17 8048574: 8b 45 fc mov -0x4(%ebp%),%eax // ra = i
18 8048577: 8b 04 c5 64 a0 04 08 mov 0x804a064(,%eax,8),%eax //ra=*(arr-data+ra*8)
19 804857e: 89 45 f4 mov %eax,-0xc(%ebp) // stack2 = ra
... ...
38 80485d1: c7 45 f8 01 00 00 00 movl $0x1,-0x8(%ebp)
39 80485d8: 83 45 fc 01 addl $0x1,-0x4(%ebp) //i = i + 1
40 80485dc: a1 80 86 04 08 mov 0x8048680,%eax //ra = *(mem) (ra = n)
41 80485e1: 83 e8 01 sub $0x1,%eax //ra = ra – 1
42 80485e4: 3b 45 fc cmp -0x4(%ebp),%eax //compare (i, ra)
43 80485e7: 0f 8f 5c ff ff ff jg 8048549 <sort+0x19> //if (>), goto 7
44 80485ed: 83 7d f8 00 cmpl $0x0,-0x8(%ebp)
45 80485f1: 0f 85 3f ff ff ff jne 8048536 <sort+0x6>
46 80485f7: 90 nop
47 80485f8: c9 leave
48 80485f9: c3 ret



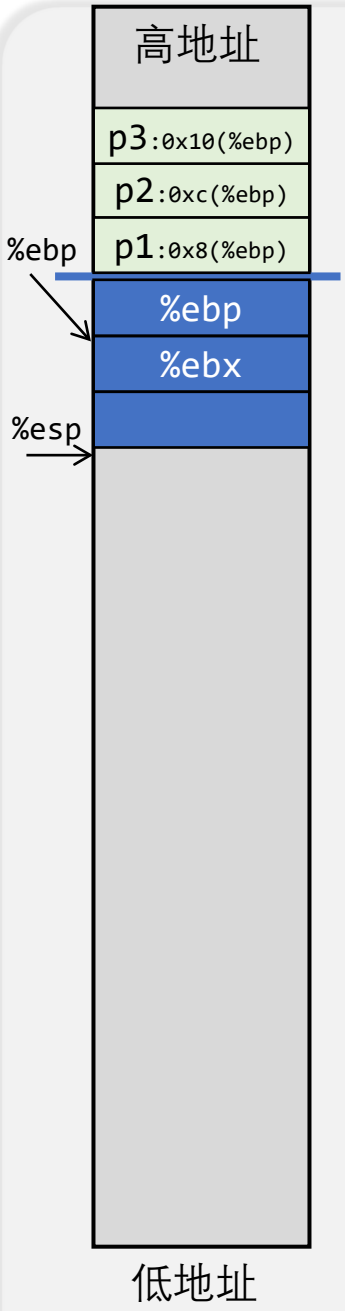

```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax
8  80491c7: 75 13                    jne    80491dc <func+0x26>
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax
12 80491d6: 01 d0                    add    %edx,%eax
13 80491d8: 8b 00                    mov    (%eax),%eax
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax
18 80491e9: 01 d0                    add    %edx,%eax
19 80491eb: 8b 18                    mov    (%eax),%ebx
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax
21 80491f0: 83 c0 01                 add    $0x1,%eax
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```



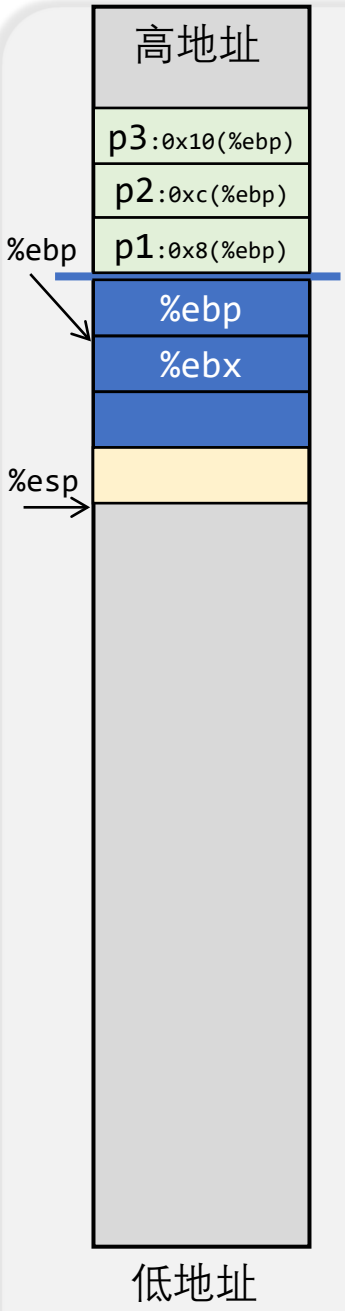

```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax
8  80491c7: 75 13                    jne    80491dc <func+0x26>
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax
12 80491d6: 01 d0                    add    %edx,%eax
13 80491d8: 8b 00                    mov    (%eax),%eax
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax
18 80491e9: 01 d0                    add    %edx,%eax
19 80491eb: 8b 18                    mov    (%eax),%ebx
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax
21 80491f0: 83 c0 01                 add    $0x1,%eax
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```



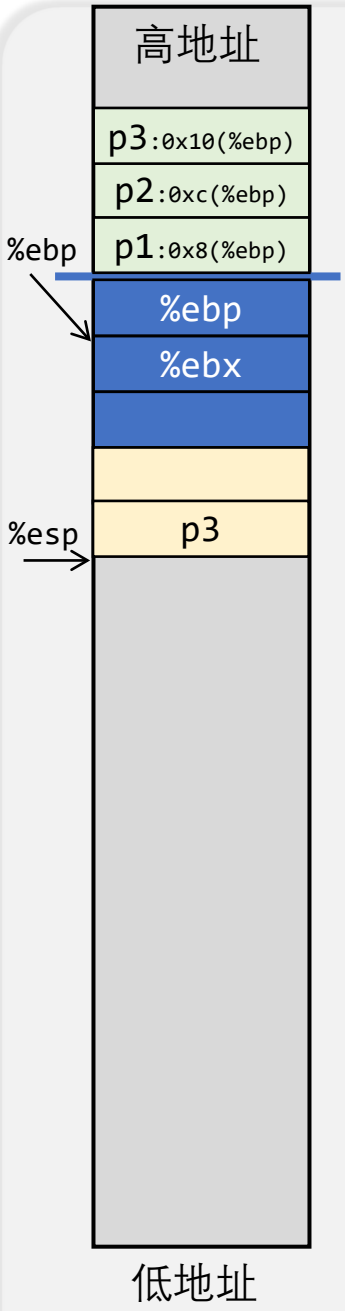

```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax
8  80491c7: 75 13                    jne    80491dc <func+0x26>
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax
12 80491d6: 01 d0                    add    %edx,%eax
13 80491d8: 8b 00                    mov    (%eax),%eax
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax
18 80491e9: 01 d0                    add    %edx,%eax
19 80491eb: 8b 18                    mov    (%eax),%ebx
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax
21 80491f0: 83 c0 01                 add    $0x1,%eax
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```



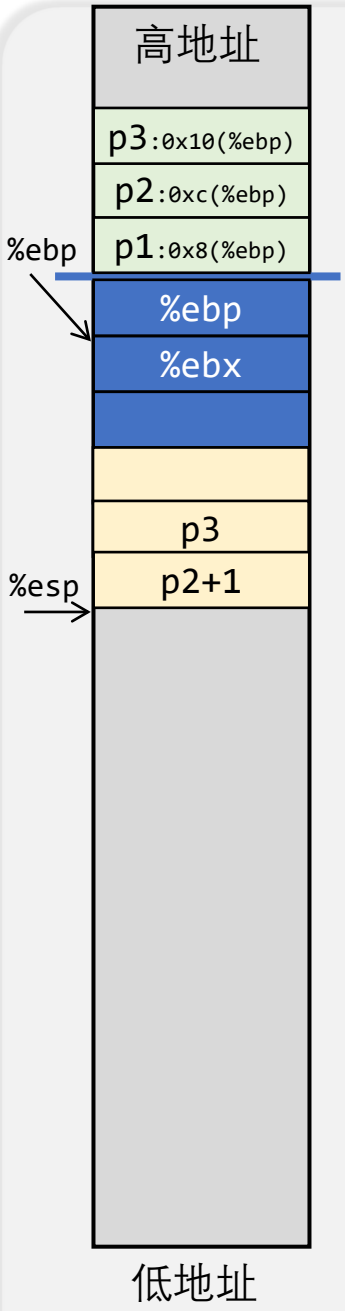

```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax //compare(p3,p2)
8  80491c7: 75 13                    jne    80491dc <func+0x26>//if(≠0)goto 15
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra*4
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
12 80491d6: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
13 80491d8: 8b 00                    mov    (%eax),%eax     //ra = *(p1+p2*4)
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra * 4
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
18 80491e9: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
19 80491eb: 8b 18                    mov    (%eax),%ebx     //rb = *(p1+p2*4)
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
21 80491f0: 83 c0 01                 add    $0x1,%eax       //ra = p2+1
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```



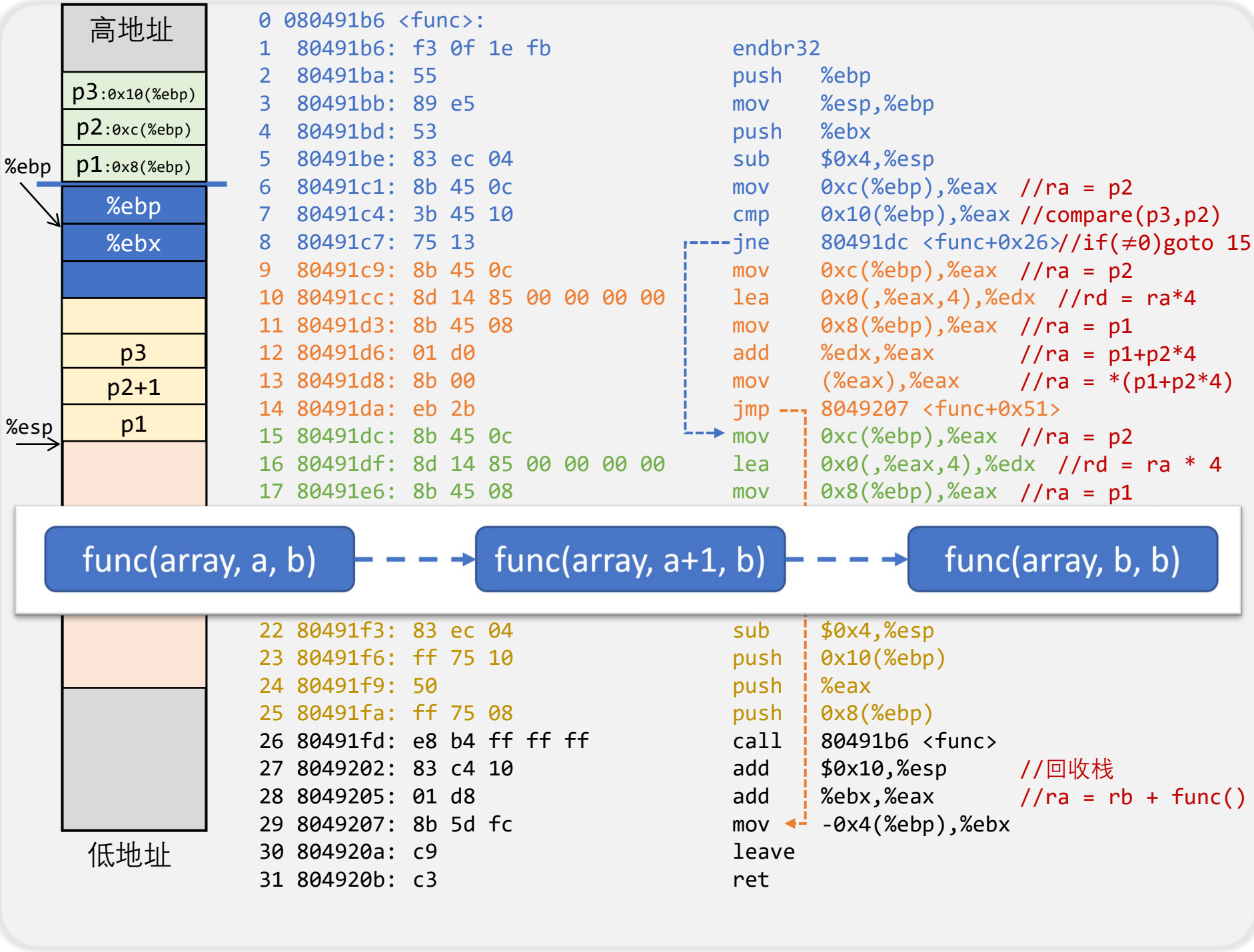

```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax //compare(p3,p2)
8  80491c7: 75 13                    jne    80491dc <func+0x26>//if(≠0)goto 15
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra*4
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
12 80491d6: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
13 80491d8: 8b 00                    mov    (%eax),%eax     //ra = *(p1+p2*4)
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra * 4
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
18 80491e9: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
19 80491eb: 8b 18                    mov    (%eax),%ebx     //rb = *(p1+p2*4)
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
21 80491f0: 83 c0 01                 add    $0x1,%eax       //ra = p2+1
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```




```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax //compare(p3,p2)
8  80491c7: 75 13                    jne    80491dc <func+0x26>//if(≠0)goto 15
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra*4
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
12 80491d6: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
13 80491d8: 8b 00                    mov    (%eax),%eax     //ra = *(p1+p2*4)
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra * 4
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
18 80491e9: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
19 80491eb: 8b 18                    mov    (%eax),%ebx     //rb = *(p1+p2*4)
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
21 80491f0: 83 c0 01                 add    $0x1,%eax       //ra = p2+1
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```




```
0 080491b6 <func>:
1  80491b6: f3 0f 1e fb              endbr32
2  80491ba: 55                       push   %ebp
3  80491bb: 89 e5                    mov    %esp,%ebp
4  80491bd: 53                       push   %ebx
5  80491be: 83 ec 04                 sub    $0x4,%esp
6  80491c1: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
7  80491c4: 3b 45 10                 cmp    0x10(%ebp),%eax //compare(p3,p2)
8  80491c7: 75 13                    jne    80491dc <func+0x26>//if(≠0)goto 15
9  80491c9: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
10 80491cc: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra*4
11 80491d3: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
12 80491d6: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
13 80491d8: 8b 00                    mov    (%eax),%eax     //ra = *(p1+p2*4)
14 80491da: eb 2b                    jmp    8049207 <func+0x51>
15 80491dc: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
16 80491df: 8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx  //rd = ra * 4
17 80491e6: 8b 45 08                 mov    0x8(%ebp),%eax  //ra = p1
18 80491e9: 01 d0                    add    %edx,%eax       //ra = p1+p2*4
19 80491eb: 8b 18                    mov    (%eax),%ebx     //rb = *(p1+p2*4)
20 80491ed: 8b 45 0c                 mov    0xc(%ebp),%eax  //ra = p2
21 80491f0: 83 c0 01                 add    $0x1,%eax       //ra = p2+1
22 80491f3: 83 ec 04                 sub    $0x4,%esp
23 80491f6: ff 75 10                 push   0x10(%ebp)
24 80491f9: 50                       push   %eax
25 80491fa: ff 75 08                 push   0x8(%ebp)
26 80491fd: e8 b4 ff ff ff           call   80491b6 <func>
27 8049202: 83 c4 10                 add    $0x10,%esp
28 8049205: 01 d8                    add    %ebx,%eax
29 8049207: 8b 5d fc                 mov    -0x4(%ebp),%ebx
30 804920a: c9                       leave
31 804920b: c3                       ret
```


高地址
p3:0x10(%ebp)
p2:0xc(%ebp)
p1:0x8(%ebp)
%ebp
%ebp
%ebx
p3
p2+1
%esp
p1
低地址
0 080491b6 <func>:
1 80491b6: f3 0f 1e fb endbr32
2 80491ba: 55 push %ebp
3 80491bb: 89 e5 mov %esp,%ebp
4 80491bd: 53 push %ebx
5 80491be: 83 ec 04 sub $0x4,%esp
6 80491c1: 8b 45 0c mov 0xc(%ebp),%eax //ra = p2
7 80491c4: 3b 45 10 cmp 0x10(%ebp),%eax //compare(p3,p2)
8 80491c7: 75 13 jne 80491dc <func+0x26>//if(≠0)goto 15
9 80491c9: 8b 45 0c mov 0xc(%ebp),%eax //ra = p2
10 80491cc: 8d 14 85 00 00 00 00 lea 0x0(,%eax,4),%edx //rd = ra*4
11 80491d3: 8b 45 08 mov 0x8(%ebp),%eax //ra = p1
12 80491d6: 01 d0 add %edx,%eax //ra = p1+p2*4
13 80491d8: 8b 00 mov (%eax),%eax //ra = *(p1+p2*4)
14 80491da: eb 2b jmp 8049207 <func+0x51>
15 80491dc: 8b 45 0c mov 0xc(%ebp),%eax //ra = p2
16 80491df: 8d 14 85 00 00 00 00 lea 0x0(,%eax,4),%edx //rd = ra * 4
17 80491e6: 8b 45 08 mov 0x8(%ebp),%eax //ra = p1
func(array, a, b)
func(array, a+1, b)
func(array, b, b)
22 80491f3: 83 ec 04 sub $0x4,%esp
23 80491f6: ff 75 10 push 0x10(%ebp)
24 80491f9: 50 push %eax
25 80491fa: ff 75 08 push 0x8(%ebp)
26 80491fd: e8 b4 ff ff ff call 80491b6 <func>
27 8049202: 83 c4 10 add $0x10,%esp //回收栈
28 8049205: 01 d8 add %ebx,%eax //ra = rb + func()
29 8049207: 8b 5d fc mov -0x4(%ebp),%ebx
30 804920a: c9 leave
31 804920b: c3 ret

阅读代码的建议

- “逆向工程”
  - 从汇编代码推导出程序的行为

- 几点要素
  1. 将不容易理解的符号用容易理解的方式代替
  2. 将一段汇编代码用容易理解的方式表达
  3. 假设内存/寄存器中的数值是变量，人肉模拟调单步调试
     - 每当出现分支时，世界都会复制两份

Happy New Year & 明年见 (?)

Take-aways

机器永远是对的

没什么是 RTFM/RTFSC 解决不了的

知道了计算机系统这个 “状态机” 是如何工作的