ICS 第 7 次小班课

2021-11-9

The place that I have never been to is termed the afar.
Yet what I keep longing for is still far beyond that.

Warm Reminders

- **Next week**
  - 本周三的课　　沈子楠
  - 下周三的课　　戚博闻
  - Archlab回顾　　黄埸瀚

- Attacklab 已放出　Due 11.28　23:59

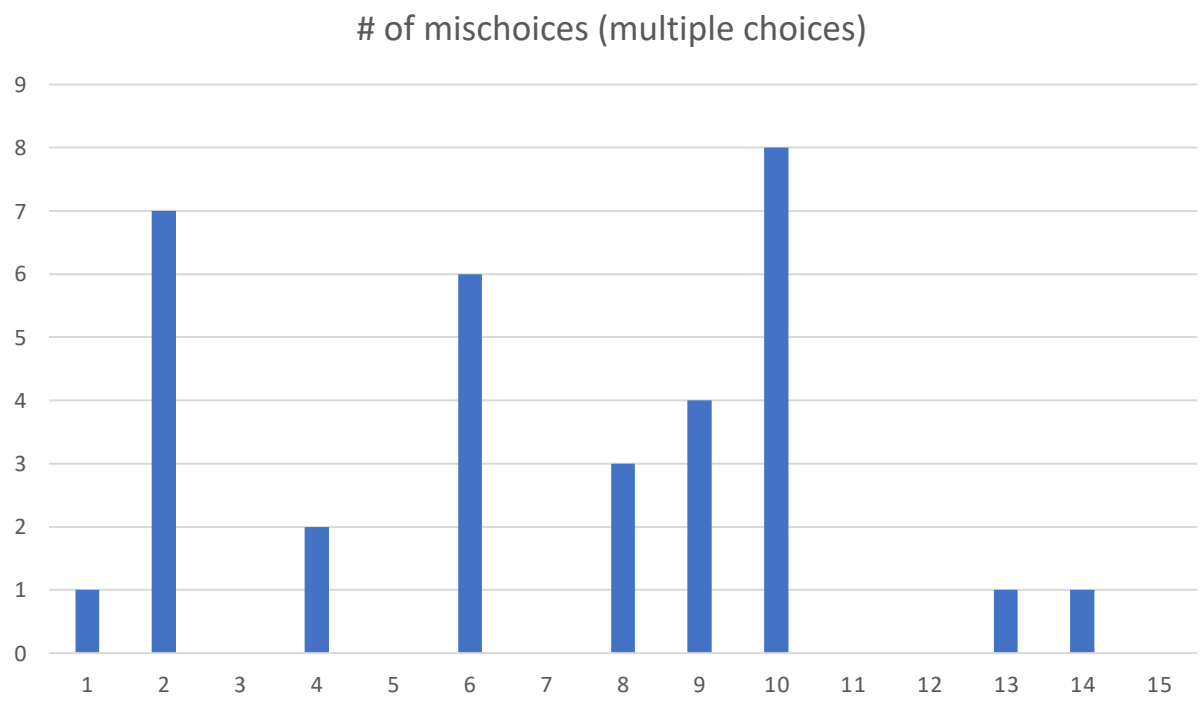
If you solution for Phase 5 has fewer gadgets, you can share it with me (though it may turn out to be serendipity).

Midterm: Overall

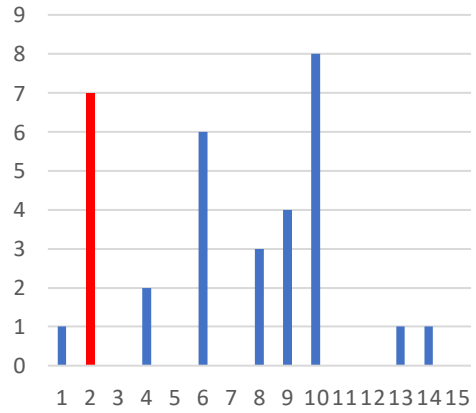
- Just bear in mind that you all really did a good job comparable to all students who registered for this class.
- ***Keep going!*** We are half-way there!

- Mild in terms of difficulty. The solution to a many questions features an inter-chapter connection.
- Wide coverage. Hardly there are tricky questions…

Midterm: Statistics



2nd Multiple Choices



2. 函数 g 定义如下：

```
int g(float f){
  union {
    float f;
    int i;
  } x;
  x.f = f;
  return x.i ^ ((1u << (x.i < 0 ? 31 : 0)) - 1);
}
```

Actually I dislike this code snippet…

Too artificial….

则对于两个 float 型变量 a,b，有 a<b 是 g(a)<g(b)的:B

A. 充分必要条件

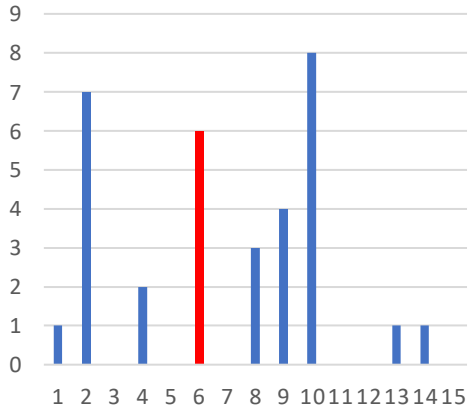
B. 充分不必要条件

C. 必要不充分条件

D. 不必要不充分条件

Think about +0 and -0 or special values

6th Multiple Choices



7.以下关于 x86-64 指令的描述，说法正确的是：B

A. 数据传送指令 movabsq $Imm, (%rax) 将以 64 位二进制补码表示的立即数 Imm 放到目的地址(%rax)中。

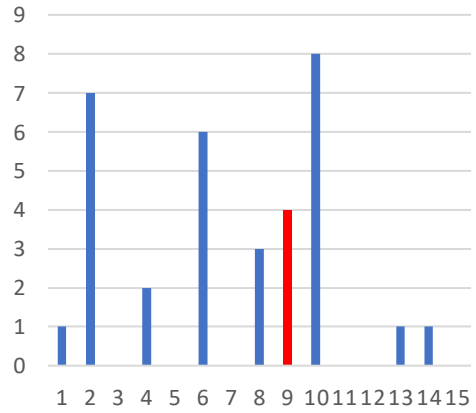
B. INC 和 DEC 指令会设置溢出标志 OF 和零标志 ZF，但不会改变进位标志 CF。

C. call *%rax 指令以%rax 中的值作为读地址，从内存中读出调用目标。

D. popq %rax 指令的行为等效于 movq %rsp, %rax; addq $8, %rsp。

A. movabsq always moves to a register.

9th Multiple Choices



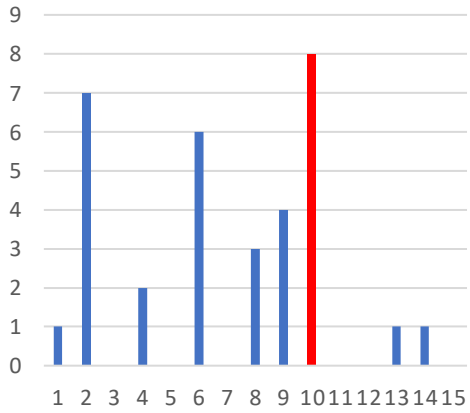
11. 在课本Y86-64的PIPE上执行以下的代码片段，一共使用到了(D)次数据转发。假设在该段代码执行前和执行后PIPE都执行了足够多的nop指令。

```
mrmovq 0(%rdx), %rax
addq   %rbx, %rax
mrmovq 8(%rdx), %rcx
addq   %rcx, %rax
irmovq $10, %rcx
addq   %rcx, %rax
rmmovq %rax, 16(%rdx)
```

A. 3　　B. 4　　C. 5　　D. 6

```
mrmovq 0(%rdx), %rax
addq   %rbx, %rax (1 stall)
mrmovq 8(%rdx), %rcx
addq   %rcx, %rax (1 stall) (Decode时第二条指令结果还未写回)
rmovq  $10, %rcx
addq   %rcx, %rax
rmmovq %rax, 16(%rdx)
```

10th Multiple Choices



12．在书中 Y86 的 SEQ 实现下，以下哪一条指令是现有信号通路能完成的：C

A． iaddq rA, V：将立即数 V 与 R[rA]相加，其中 rB 域设为 F，结果存入寄存器 rA

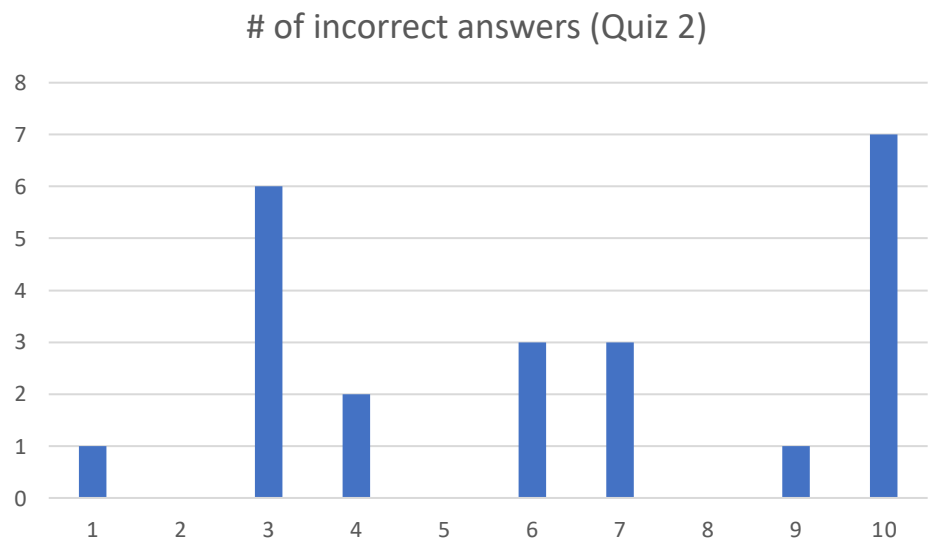
B．mmmovq rA, rB：将 R[rA]存的地址开始的 8 字节数据，移动到 R[rB]存的地址

C．leave：相当于先执行 rrmovq %rbp, %rsp，再执行 popq %rbp

D． enter：相当于先执行 pushq %rbp，再执行 rrmovq %rsp, %rbp

A. valA + valC ?
B. Violates the principle of no reading back
C. How?
D. R[%rsp] <- valE R[%rbp] <- valE

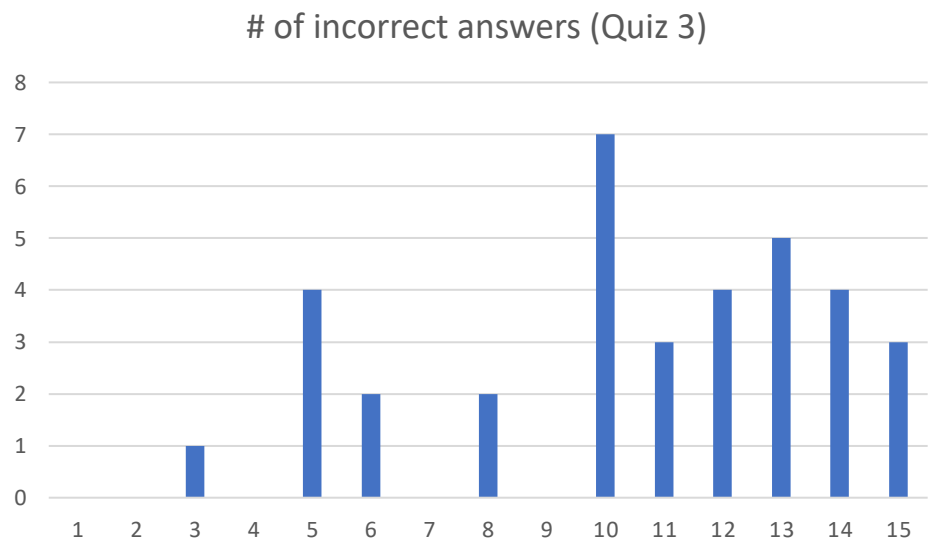
Midterm: Statistics



1．假设某 x86-64 机器在地址 0x100 和 0x101 处存储的数据用二进制表示分别为$[1010\ 1100]_2$和$[1111\ 1011]_2$.又假设 x 是一个 short 类型的变量,其地址为 0x100.则 x 的十六进制补码表示为 0x____(1)____,这是一个____(2)____(填“正”或“负”)数，其绝对值为____(3)____(用十进制数字表示).

```
int i, j, k;
for(i = 0; i <= 2147483647 - 2; i ++){
  j = i + 1;
  k = j + 1;
  float *x = (float *)(&i);
  float *y = (float *)(&j);
  float *z = (float *)(&k);
  if(*y - *x != *z - *y){
    printf("0x%08x\n", i);      //输出 8 位 16 进制数
    break;
  }
}
```

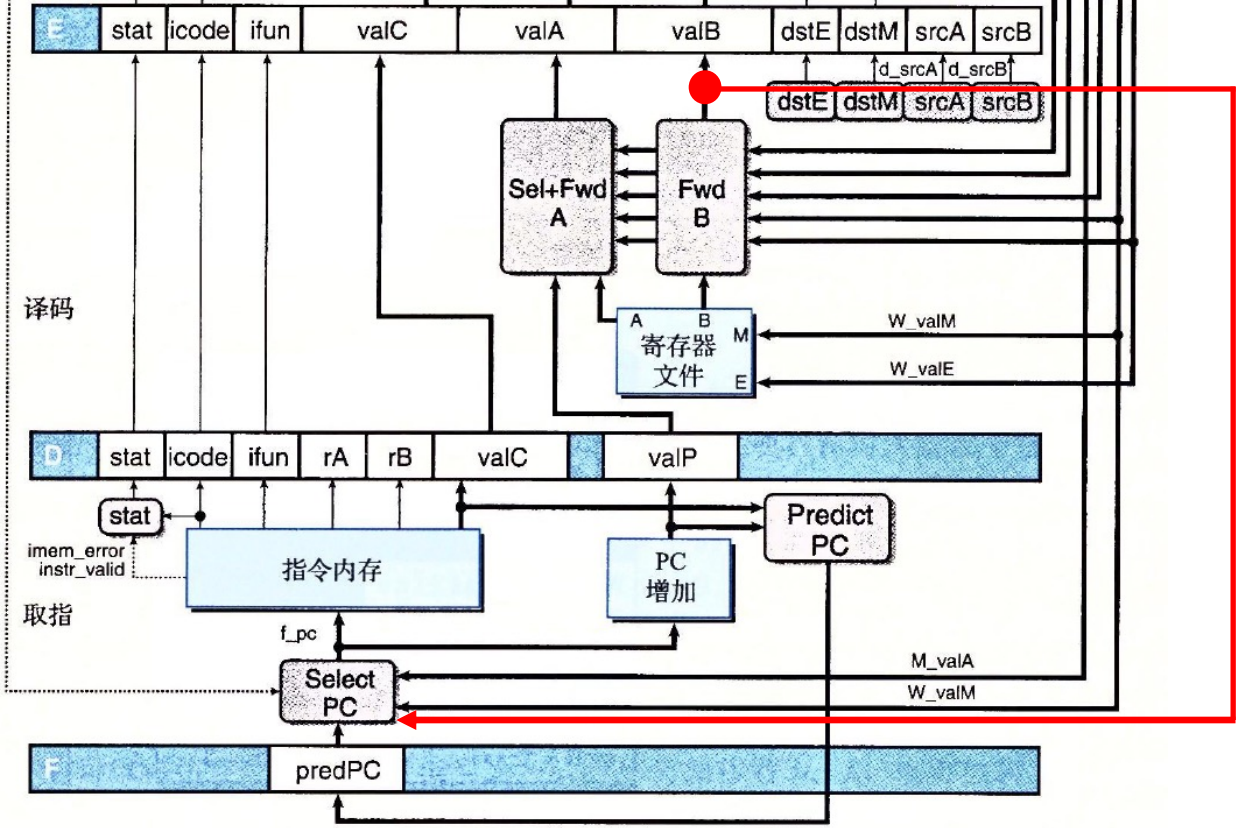
Midterm: Statistics



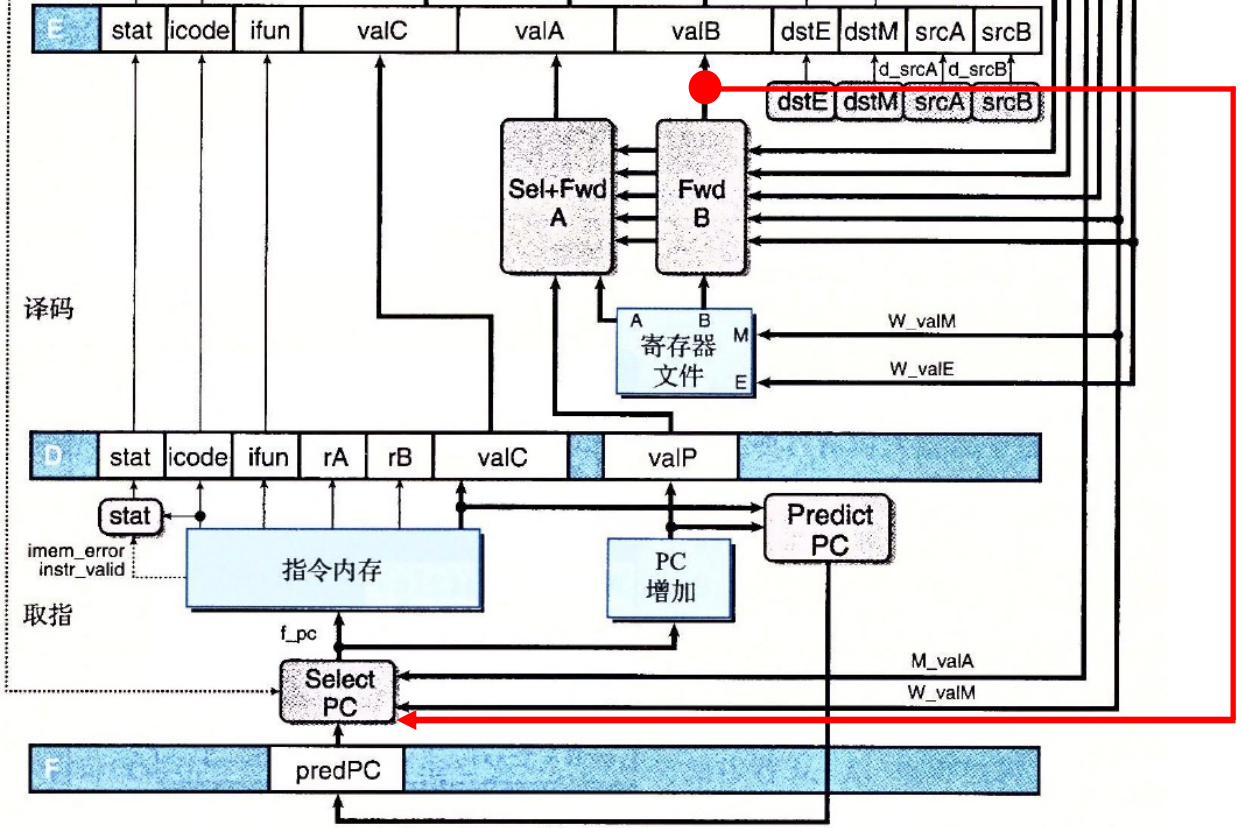
Midterm: Statistics

- Quiz 4: You may picture a diagram yourself haha.

Stage	jxx *rB
Fetch	icode : ifun ← M_1[PC]
Decode	
Execute	
Memory	
Write Back	
Update PC	PC <- Cnd ? valE: valP




```
word f_pc = [
    ① D_icode == IJREGXX: d_valB;
    (M_icode == IJXX || M_icode == IJREGXX) && !M_cnd : M_valA;
    ②
    W_icode == IRET : W_valM;
    ③
    1 : F_predPC;
    ④
]
```



触发条件：（如果有多种可能请任写一种）

_____ == IJREGXX && _____

M stage or E stage?

控制逻辑：（如果有多种可能请任写一种）

F	D	E	M	W

```
# Array of 3 elements
array:
    .quad return
    .quad L1
    .quad L2

# void foo(long n, long *arr)
# n in %rdi, arr in %rsi
foo:
    rrmovq  %rdi, %rdx      # line 1
    addq    %rdx, %rdx      # line 2
    addq    %rdx, %rdx      # line 3
    addq    %rdx, %rdx      # line 4
    irmovq  array, %rcx     # line 5
    addq    %rdx, %rcx      # line 6
    andq    %rdi, %rdi      # line 7
    jge     *%rcx           # line 8
return:                     #
    ret                     # line 9
L2:                         #
    mrmovq  16(%rsi), %rcx  # line 10
    rmmovq  %rcx, 8(%rsi)   # line 11
L1:                         #
    mrmovq  8(%rsi), %rcx   # line 12
    rmmovq  %rcx, (%rsi)    # line 13
    jmp     return          # line 14
```

在foo函数运行过程中，计算以下情况foo函数的执行周期数。周期数计算从执行foo第一条指令开始，直到其返回指令ret完全通过流水线为止。另外假设foo函数开始的若干条指令不会和foo函数体外的指令形成冒险。

n= -1 _____ **9+2+4=15**

n=0 _____ **9+4=13**

n=2 _____ **14+2+4=20**

Quiz 5

（3）在如下代码中，假设初始时 cache 为空，只考虑读写数据引起的 cache 访问，假定：

a. Cache 命中的延迟（hit time）为 10 周期，不命中处罚（miss penalty）为 190 周期；

b. 延迟与数据的长度无关；

c. 若数据对应到不同的缓存块，则需进行多次访问，且访问之间不能重叠（即访存不能并行）。

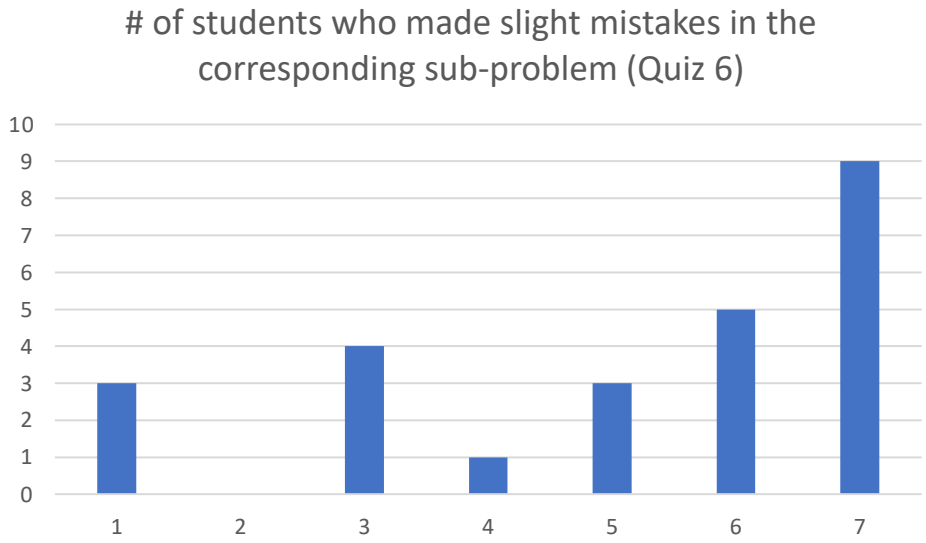
（共 4 分）

```
# define N 4
typedef struct {
    char x;
    short y;
} node;
node p[N];
int main() {
    for (int i = 0; i < N; i++) {
        p[i].x = i;
        p[i].y = 1 - i;
    }
    return 0;
}
```

① 若数组 p 的起始地址为 0x0000，cache block 大小为 8 Bytes，采用直接映射，则 cache 访问的总延迟为 460 周期（注意结构体数据对齐）。（2 分）

② 若结构体不采用数据对齐（即(void *)&p[i] - (void *)&p[0] == 3 * i），则上述代码在①中的条件下，则 cache 访问的总延迟为 470 周期。（2 分）

Midterm: Statistics



```
float func1(s_element* p){
    float ans1 = 0, ans2 = 0;          //line 1
    while( p && ____①____){            //line 2
        ans1 += p->u1.f;               //line 3
        ans2 += ____②____;             //line 4
        p = ____③____;                 //line 5
    }
    if(_____④_____)                    //line 6
        ans1 += p->u1.f;             //line 7
    return ans1 + ans2;              //line 8
}
```

①p->next

②p->next->u1.f

③p->next->next

④p

5．此时，同学 C 指出同学 I 的做法会导致新的错误，原因是_____。（2 分）

浮点数加法不满足结合律

6．同学 S 从书上读到，k * k 循环展开在 k 很大时反而可能获得较差的效果，这是因为 k 很大时会导致_____。（2 分）

寄存器溢出（过多的循环变量会分配到栈上）

7．同学 B 使用了同学 I 改进后的 func1 函数实现下面这段代码，并在大端法机器上运行，最终得到的输出是：_____。（2 分）

0x4098 -1

```
s_element x, y, z;
    x.u1.f = 1.2;
    y.u1.f = 3.55;
    x.next = &y;
    y.next = nullptr;
    z.u1.f = func1(&x);
    printf("0x%x  %d\n",  z.u1.s[0]-z.u1.s[1],  &z.u1.s[0]-
&z.u1.s[1]);
```

Coding Style

写好注释是一种调试的捷径。

Coding Style

Check the surrounding code and try to imitate it.

Several mainstream C coding styles exist. E.g. Visual Studio, LLVM, Google, Chromium, Mozilla, etc. They differ in

- Layout and Spacing
- Bracing
- Naming Conventions
  - Global/local variables
  - Functions
  - Members
- Commenting
  - Header Comments
  - Inline comments
  - Extraneous comments
- …

Some of these terms can be done via a reformat script.

What can be auto reformatted?

- **Layout and Spacing** (Suitable for C, since C is agnostic of spacing)
- Bracing (Except for re-positioning brackets, it will change the semantics)
- Naming Conventions (Dangerous!)
  - Global/local variables
  - Functions
  - Members
- Commenting (Breaking down a long comment into several lines is still about spacing)
  - Header Comments
  - Inline comments
  - Extraneous comments
- … (But who knows?)

Indeed, there is a widely-used tool named clang-format. Provided both as a command-line tool and a plugin supported by many IDEs, it may adjust the layout and spacing in your source files.

Clang-format Usage

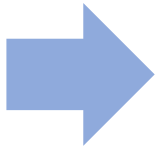
- In command line

  ```
  > clang-format –-style=LLVM file.c –i
  ```

  (-i stands for in-place reformatting)

```
#include <stdio.h>

int main() {
// print the first half
printf(
    "Survived ICS midterm! "
);
    // print the second half
printf("Yeah~\n"    );
    return 0;
}
```



```
#include <stdio.h>

int main() {
  // print the first half
  printf("Survived ICS midterm! ");
  // print the second half
  printf("Yeah~\n");
  return 0;
}
```

- It adjusts the layout and the spacing of your program without manipulating the original symbols you use. So this tool definitely *cannot* introduce new bugs into your code.

But a reformat tool has limited capacity…

- In order to write a better piece of code, you should pay heed to these:
  - No ACM style coding!
    - `int _,__,a,aa;`
    - `while(f(x)) {x+=1,y+=x;}`

  That is, all the symbols you defined for your variables, structures, members and functions should be distinguishable. It is even better if they are self-explanatory.
    - `int arr_index;`
    - `void binary_search(int *ordered_arr, int start, int end);`

  If the symbol comprises several words, it should stick to one particular naming conventions.
    - Camelcase (`countSomeElements`)
    - Underscores (`as_is_shown_above`)

    These two conventions are the most widely-adopted for C. For example, C std libs abide by the convention of underscores. You may recall `size_t` and `rand_r()`.

But a reformat tool has limited capacity…

- In order to write a better piece of code, you should pay heed to these:
  - Don't be a miser! Comment anywhere if you find it necessary.
  - Also, do not sprinkle comments all across your code!

  - Examples of when a comment is quite likely to be indispensable in C:
    - Tricks in your code. Comment how you implement it and justify its correctness if needed.

      E.g. You do the float inverse square root in this way:

```
float Q_rsqrt( float number ) {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;
    i = 0x5f3759df - ( i >> 1 );   // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );
//  y = y * ( threehalfs - ( x2 * y * y ) );
    return y;
}
```

It is a good habit that you highlight the trick even though you cannot understand it anyway.

When a comment is indispensable in C

- After critical lines

  For example, if the order of several consecutive lines cannot be changed, you had better drop a note around to inform the poor code maintainer (say, you yourself) later.
  As another example, if some lines are optional, you can point this fact using line comments.
  Continue with the previous example:

```
float Q_rsqrt( float number ) {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;            // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );   // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
//  y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed
    return y;
}
```

When a comment is indispensable in C

- Function definition of critical functions
  - Functions + Implementations. ***Be descriptive!***
  - (Optional) The meaning of each arguments.
  - (Optional) Resources that is allocated but not freed in this function.
  - On possible errors, how the function behaves or what value it returns.

  Continue with the previous example:

```
/* Q_rsqrt:
 * Compute the fast inverse square root
 * of a single precision floating point
 * using the trick in Quake III Arena.
 *
 * The argument is assumed to be strictly
 * positive.
 */
float Q_rsqrt( float number ) {
    ...
}
```

Good Examples from ICS Lab Submissions

- Codes that are tricky to get right (From Tshlab)

```
/*wait!显式地等待fg的进程终止,调用sigsuspend函数，在回收后解除阻挡*/
/*判断fg进程终止的方式：fg进程只有一个，而fgpid当且仅当没有fg进程时才会返回0*/
/*返回0就意味着fg进程的终止*/
 while ( fgpid(job_list) )
 {
        sigsuspend(&prev_mask) ;
 }
 sigprocmask(SIG_SETMASK,&prev_mask,NULL) ;
```

- Implementation details (From Archlab)

```
# There are mainly 4 tricks utilized in my approach.
#
# (1)    Loop unrolling
#    I manage to achieve 10x unrolling within the length limitation.
#    The idea of this trick is simply reducing instructions wasted
#    on mrmovq and rmmovq (and iaddq for loop update), by coding the bias
#    into the instructions (mrmovq 80(%rdi),%r10, etc.). The effect of this
#    trick fades quickly w.r.t the unrolling size.
#
# (2)    Trinary Tree
```

Linking: An Intro

What, Why and How?

The C compilation process

- Symbols in the program
- How to resolve the symbol (As they are collected from multiple modules)?
  - Each symbol is either a definition or a reference to the symbol already defined.
  - When it is a definition? Can we define a symbol multiple times? How to resolve the reference?

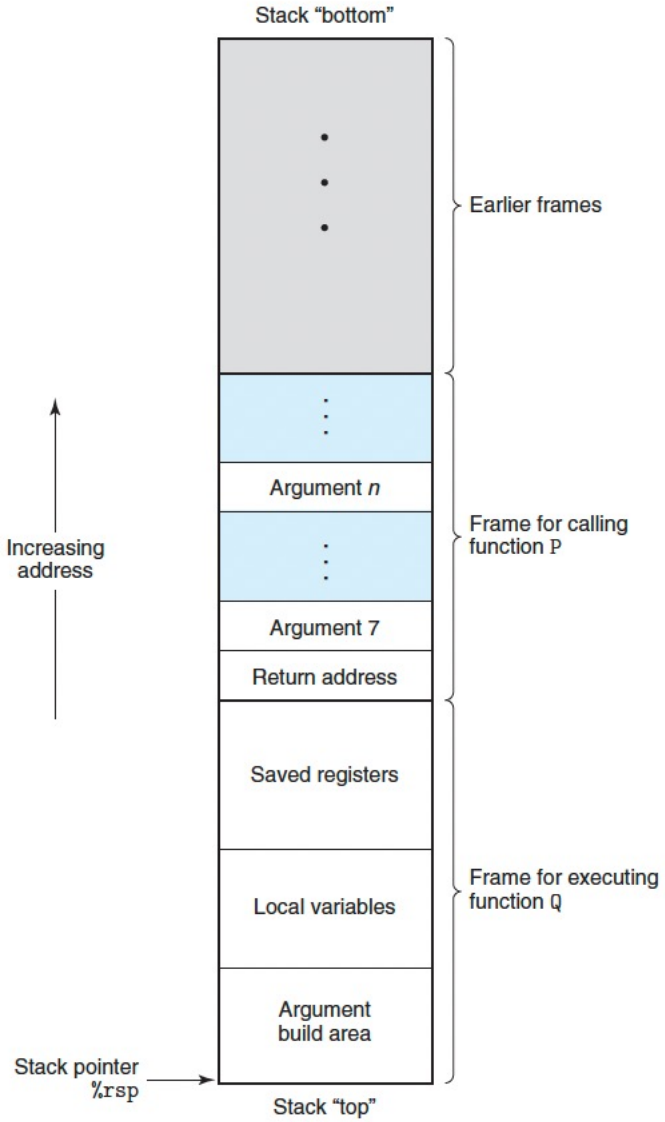
- How do symbols correspond with the original variables and functions in C source file?

(Static) Scope in C Grammar

- Scope: When a variable is valid (or visible).
- Global Scope & Global Variables
- Local Scope & Local Variables
- Static Variables / Functions
- Externed Variables / Functions
- Volatile Variables

Where are different kinds of variables stored?

- Global Variables
- Local Variables
- Static Variables
- Externed Variables
- Volatile Variables



How is this picture of storage be realized?

- Object files
  - Relocatable object files
  - Executable object files
  - Share object fieles

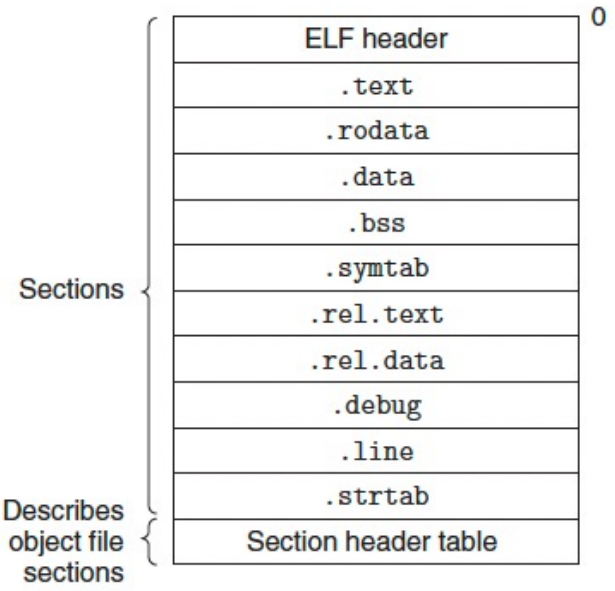


Figure 7.3
Typical ELF relocatable object file.