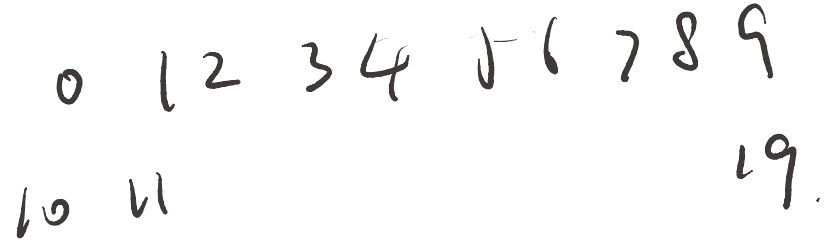
Homework7

Homework7

- 题目1
  - Consider the following program: #define LEN 10
    int a[LEN][LEN];
    void f(void) {
  - int i, j;
    for (i = 0; i < LEN; i++)
  - for (j = 0; j < LEN; j++) {
  - a[i][j] = i * LEN + j;
  - }
  - }
  - Suppose the address of a is 0x10000000. After the function f() finished, fill the following table (if you don't know the value, please write NONE):



Homework7

- 题目1

%eax	0x10000000
%ecx	22
$0x10000004	0X10000004
0x10000012	3
0xFFFFFF8	NONE
(%eax, %ecx, 8)	43

8x22+10000 —
44

Homework7

- 题目2
  - Fill the blanks of the C program:

```
int dw_loop(int x, int y, int n) {
    do{
        x += n;
        y *= n;
        n--;
    }while (n>0 && y<n);
    returen x;
}
```

- The assembly code is as follows:
- x@%ebp+8, y@%ebp+12, n@%ebp+16

```
  movl 8(%ebp), %eax      i = x
  movl 12(%ebp), %ecx     j = y
  movl 16(%ebp), %edx     k = n
.L2:
  addl %edx, %eax         i = i + k
  imull %edx, %ecx        j = j * k
  subl $1, %edx           k--
  testl %edx, %edx
  jle  .L5                k <= 0   k > 0
  cmpl %edx, %ecx
  jl .L2                  (k > 0 && j < k)
.L5:
```

Homework7

- 题目3
  - After ICS class, Barathrum has written a function like below: int cmov_complex(int x, int y) {
  - return x < y? x * y; (x + y) * y; }
  - (1). Please write down the corresponding assembly code by using conditional move operations.
  - (2). When Barathrum compiles it with gcc, he finds that there's no cmov at all in the assembly code! Please explain why gcc doesn't use conditional move operations in this case.

(1) x in %rdi, y in %rsi

```
cmov_complex:
    movq    %rsi, %rax
    mulq    %rdi, %rax
    movq    %rsi, %rdx
    addq    %rdi, %rdx
    mulq    %rsi, %rdx
    cmpq    %rsi, %rdi
    cmovge  %rdx, %rax
```

(2) 两个表达式不只是一条加法指令，较为复杂，需要考虑计算成本

因此 gcc 使用条件控制转移的方式。

Homework7

- 题目4
  - Translate the following switch statements into assembly using jump table.
  - int x = <some value>;
  - int result = 0;
  - switch (x) {
    - case 24:
      - result = x + x;
      - break;
    - case 27: case 28:
      - result = x + 10;
      - break;
    - case 26:
      - result = x * 2;
      - // Notice: there is no break here!
    - case 29: case 30:
      - result = result + 5;
      - break;
    - default:
      - result = 3;
      - break;
  - }

x in %rdi, result in %rdx

```
sub   $24, %rsi
cmpq  $6, %rsi
mov   $0, %rdx
ja
jmp   *.L4(,%rsi,8)
```

计算 n-24 = index

比较 index : 6

result = 0

如果 > goto loc_def

goto *jt[index]

```
.L3
  leaq  (%rdi,%rdi,1), %rdx
  jmp   .L2
.L5
  movq  %rdi, %rdx
  addq  $10, %rdx
  jmp   .L2
```

result = 2x

Goto done

24

result = x+10

Goto done

27.28

```
.L6
    leaq  (%rdi, %rdi, 1), %rdx
```

result = 2x

26

```
.L7
    movq  %rdi, %rdx
    addq  $5, %rdx
    jmp   .L2
```

result = x+5

29.30

```
.L8
    movq  $3, %rdx
```

result = 3

default

```
.L2  ret
```

done

跳转表：

```
.L4
    .quad   .L3     Case 24
    .quad   .L8     Case 25
    .quad   .L6     Case 26
    .quad   .L5     Case 27
    .quad   .L5     Case 28
    .quad   .L7     Case 29
    .quad   .L7     Case 30
```