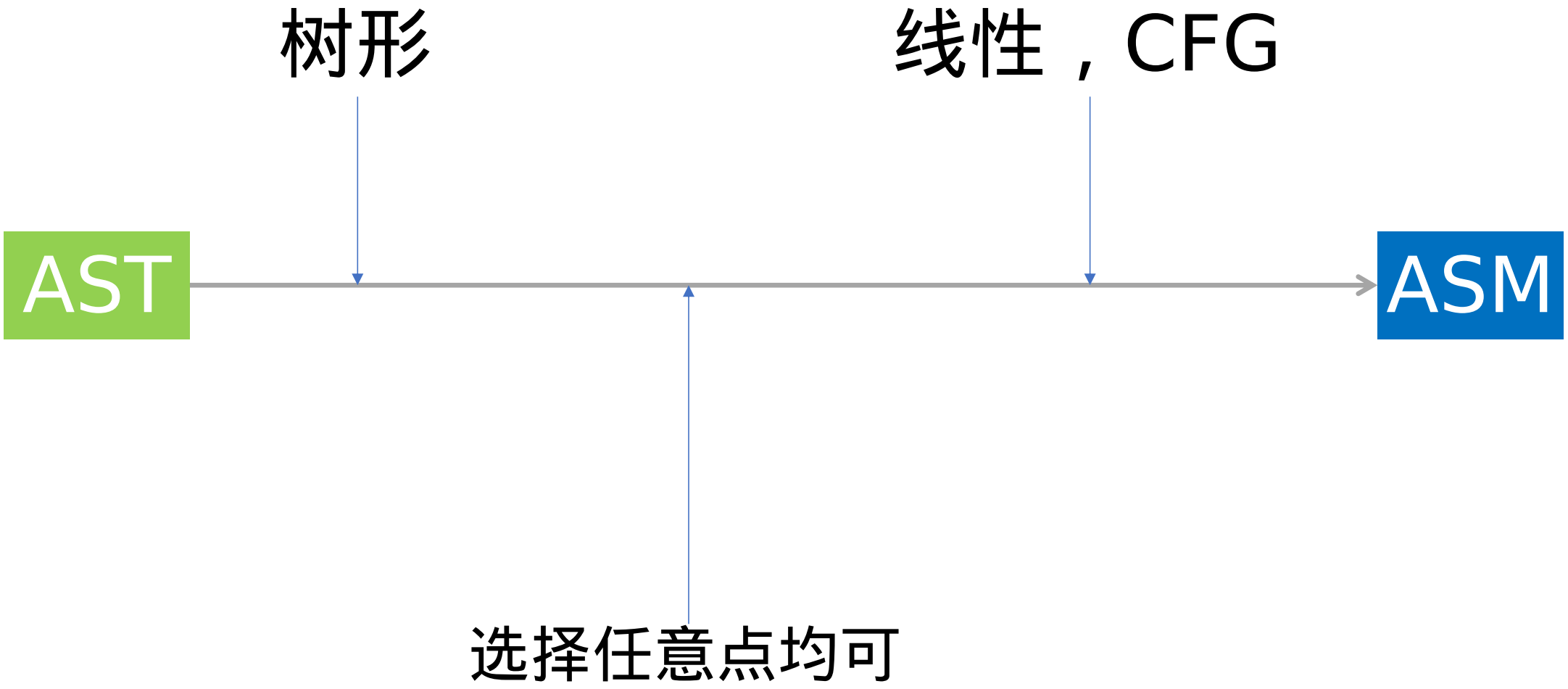
IR & NASM

Intermediate Representation

- 树形（虎书、自制编译器）
- 线性（龙书）
- CFG（Google）

线性和CFG均方便进行优化


树形
线性，CFG
AST
ASM
选择任意点均可

四元式

- 常用中间代码形式
- X=Y op Z
- （OP，OPERand1，OPERand2，Result）
- 接近汇编形式

NASM Tutorial

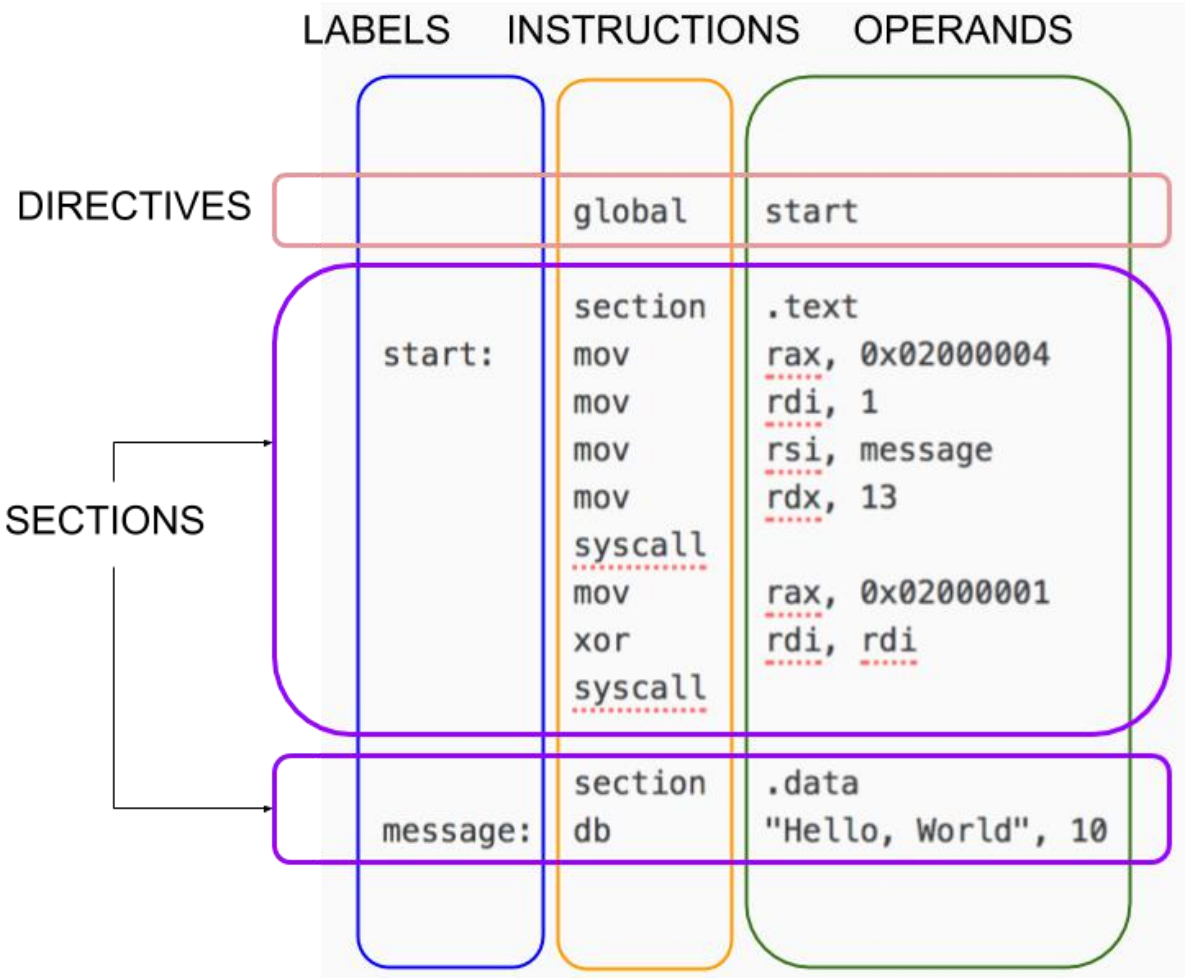
- Tutorial
  - http://cs.lmu.edu/~ray/notes/nasmtutorial/
- **x64 Cheat Sheet**
- Reference
  - 自制编译器：Good explanation on X86 instructions
  - CSAPP Chapter 3: Good explanation on X86-64
- **C2NASM**
  - A good tool to see the reference code by gcc

Hello.asm

hello.asm

```
; ------------------------------------------------------------------------------
; Writes "Hello, World" to the console using only system calls. Runs on 64-bit macOS only.
; To assemble and run:
;
;     nasm -fmacho64 hello.asm && ld hello.o && ./a.out
; ------------------------------------------------------------------------------

          global    start

          section   .text
start:    mov       rax, 0x02000004         ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to do the write
          mov       rax, 0x02000001         ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

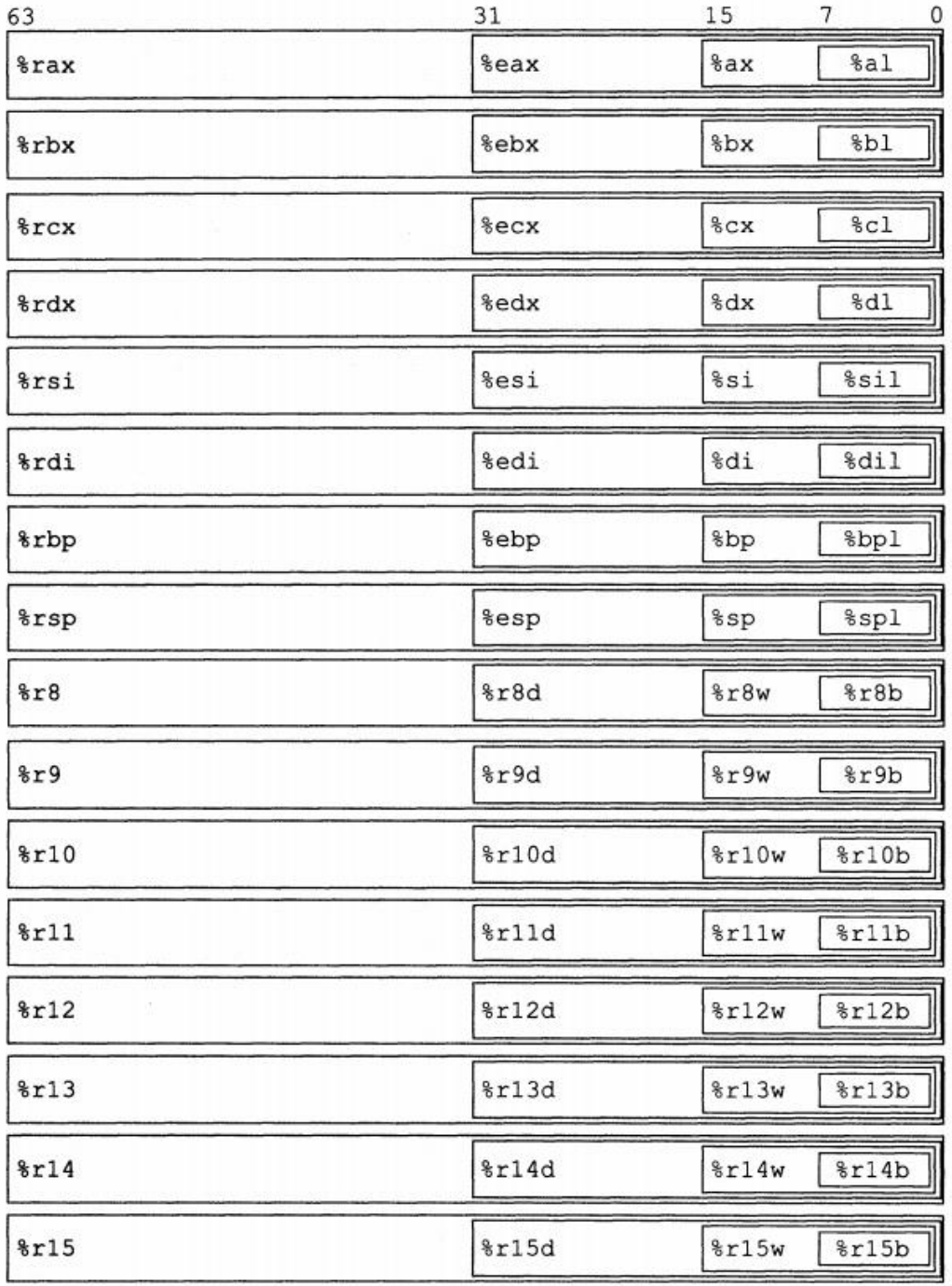
结构



伪指令，用于全局变量

Section3.2

寄存器


63
31
15
7
0
%rax
%eax
%ax
%al
%rbx
%ebx
%bx
%bl
%rcx
%ecx
%cx
%cl
%rdx
%edx
%dx
%dl
%rsi
%esi
%si
%sil
%rdi
%edi
%di
%dil
%rbp
%ebp
%bp
%bpl
%rsp
%esp
%sp
%spl
%r8
%r8d
%r8w
%r8b
%r9
%r9d
%r9w
%r9b
%r10
%r10d
%r10w
%r10b
%r11
%r11d
%r11w
%r11b
%r12
%r12d
%r12w
%r12b
%r13
%r13d
%r13w
%r13b
%r14
%r14d
%r14w
%r14b
%r15
%r15d
%r15w
%r15b

内存操作

- 格式

- `[ number ]`
- `[ reg ]`
- `[ reg + reg*scale ]` *scale is 1, 2, 4, or 8 only*
- `[ reg + number ]`
- `[ reg + reg*scale + number ]`

```
[750]                ; displacement only
[rbp]                ; base register only
[rcx + rsi*4]        ; base + index * scale
[rbp + rdx]          ; scale is 1
[rbx - 8]            ; displacement is -8
[rax + rdi*8 + 500]  ; all four components
[rbx + counter]      ; uses the address of the variable 'counter' as the displacement
```

byte word dword qword限定size

调用C函数（外部函数）

hola.asm

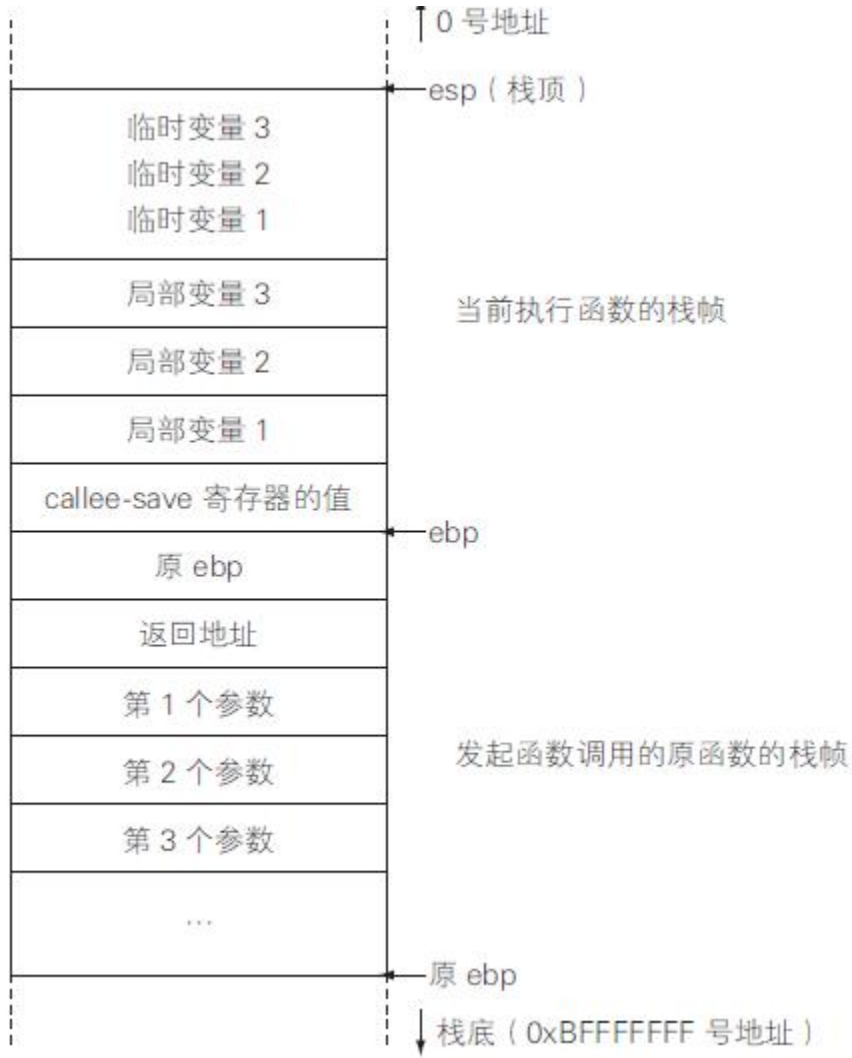
```
; ------------------------------------------------------------------------------
; Writes "Hola, mundo" to the console using a C library. Runs on Linux.
;
;     nasm -felf64 hola.asm && gcc hola.o && ./a.out
; ------------------------------------------------------------------------------

          global    main
          extern    puts

          section   .text
main:                                   ; This is called by the C library startup code
          mov       rdi, message        ; First integer (or pointer) argument in rdi
          call      puts                ; puts(message)
          ret                           ; Return from main back into C library wrapper
message:
          db        "Hola, mundo", 0    ; Note strings must be terminated with 0 in C
```

调用约定

- Calling convention
  - Integer Arguments :
    - rdi, rsi, rdx, rcx, r8, r9, then put to stack
  - Integer return value:
    - rax
  - Callee save register
    - RBX, RBP, R12, R13, R14, and R15.

函数调用流程

- 1、将caller save寄存器和参数压入栈中
- 2、rbp压入栈中，将rsp的值转给rbp，将callee save压入栈中
- 3、减少rsp，给函数分配栈帧
- 4、完成调用后恢复callee，将rsp置为rbp，rbp恢复为原rbp
- 5、清除栈中参数


0 号地址
esp（栈顶）
临时变量 3
临时变量 2
临时变量 1
局部变量 3
局部变量 2
局部变量 1
callee-save 寄存器的值
当前执行函数的栈帧
ebp
原 ebp
返回地址
第 1 个参数
第 2 个参数
第 3 个参数
…
发起函数调用的原函数的栈帧
原 ebp
栈底（0xBFFFFFFF 号地址）

递归调用

```
; ------------------------------------------------------------------------
; An implementation of the recursive function:
;
;   uint64_t factorial(uint64_t n) {
;       return (n <= 1) ? 1 : n * factorial(n-1);
;   }
; ------------------------------------------------------------------------

        global  factorial

        section .text
factorial:
        cmp     rdi, 1                  ; n <= 1?
        jnbe    L1                      ; if not, go do a recursive call
        mov     rax, 1                  ; otherwise return 1
        ret
L1:
        push    rdi                     ; save n on stack (also aligns %rsp!)
        dec     rdi                     ; n-1
        call    factorial               ; factorial(n-1), result goes in %rax
        pop     rdi                     ; restore n
        imul    rax, rdi                ; n * factorial(n-1), stored in %rax
        ret
```

内建函数

- 1、手写
- 2、用c写之后通过gcc -O3编译
- ~~3、抱大腿~~

```c
unsigned char *__lib_str_substring(unsigned char *a, long low, long high) {
    int l = high - low + 1;

    unsigned char *ret = (unsigned char *)malloc(l + 1 + sizeof(int));
    *((int *)ret) = l;
    ret = ret + sizeof(int);
    a += low;
    for (int i = 0; i < l; i++)
        ret[i] = a[i];
    ret[l] = 0;
    return ret;
}
```