Integers

-

	0x100	0x101	0x102	0x103
Big endian	01	23	45	67
Little endian	67	45	23	01

- $\lceil x/2^k \rceil$ is given by `(x + (1 << k) - 1) >> k`
- $x/2^k$ is given by `(x<0 ? x+(1<<k)-1 : x) >> k`

Floating Point

- Lacks common +*rules `a*b*c != a*(b*c), a+b+c != a+(b+c), a*(b+c) != a*b + a*c`
- Contains 3 parts `sign,exp,frac`; $V = (-1)^s \times M \times 2^E$
- 3 parts of `float`: 1+8+23=32; `double`: 1+11+52=64
- Exponent E, k-bit `exp` field
- Significand (mantissa) M – $0 \le M < 1$ or $1 \le M < 2$, represented by n-bit `frac` field $(f_{n-1} \cdots f_1 f_0)$
- Normalized values
  - `exp` is neither all zeros nor all ones
  - $E = e-$Bias where e is the unsigned number in `exp` and Bias $= 2^{k-1} - 1$
  - `frac` represents $0 \le f < 1$ with $0.f_{n-1} \cdots f_1 f_0$ and $M = 1 + f$
  -

1	10000000	10101010101010101010101
$(-1) \times 2^1 \times 1.10101010101010101010101$		

- Denormalized values
  - `exp` all zeros
  - Exponent value is $E = 1 - Bias$, $M = f$ (no leading 1)
  -

0	00000000	10101010101010101010101
$1 \times 2^{1-01111111} \times 0.10101010101010101010101$		

- Special values
  - Exponent field is all ones
  - Fraction field all zeros can represent $\pm\infty$, depending on sign bit
  - Nonzero fraction field is NaN
  - $+\infty = 0, 11111111, 00000000000000000000000$
  - every conditional expression with `NaN` is 0
  - `NaN == NaN` is 0
  - every arithmetic expression with `NaN` is `NaN`
  - `NaN * 0` is `NaN`
  - forcibly trans `inf` or `NaN` to `int` gets int_{min}
- Rounding
  - 向更近的地方舍入，如果一样，就向偶舍入。比如下面保留四位：
  - `1.01110010011` → `1.0111`，末尾小于 0.5
  - `1.01111110011` → `1.1000`，末尾大于 0.5
  - `1.0101[100...]` → `1.0110`，末尾恰是 0.5，奇数向偶数
  - `1.0110[100...]` → `1.0110`，末尾恰是 0.5，偶数不变
- Multiplication
  - $(-1)^{s_1} M_1 2^{E_1} \times (-1)^{s_2} M_2 2^{E_2}$
    $= (-1)^{s_1 \hat{} s_2} (M_1 \times M_2) 2^{E_1 + E_2}$
  - If $M_1 \times M_2 \ge 2$, $M = M_1 \times M_2/2, E = E_1 + E_2 + 1$
  - If E out of range, overflow to `inf`
  - Round M to fit `frac` precision
  - 溢出优先于舍入

Assembly Basics

- "word"=16-bit; "double word"=32-bit (int); "quad words"=64-bit
- 64 位机器下，指针是 8-byte 的
- `mov S, D` has the effect of $S \to D$

`movq Src, Dest`	C Analog
`movq $0x4, %rax`	`tmp = 0x4`
`movq $-147, (%rax)`	`*p = -147`
`movq %rax, %rdx`	`tmp2 = tmp1`
`movq %rax, (%rdx)`	`*p = tmp`
`movq (%rax), %rdx`	`tmp = *p`

- `movzbq` 把 8 位 (b) 搬运到 64 位 (q) 寄存器，空出的高位补 0(z)
- `movsbq` 把 8 位 (b) 搬运到 64 位 (q) 寄存器，空出的高位根据符号位 (s)
- `leaq S, D` has the effect of $\&S \to D$
- `subq S, D` has the effect of $D - S \to D$
- `salq S, D` has the effect of $D \cdot 2^S \to D$

Opcodes of arithmetic operations

addq	+	xorq	⊕	salq	also `shlq <<`
subq	-	andq	&	sarq	Arithmetic >>
imulq	×	orq	|	shrq	Logical >>
incq	++	decq	−−	negq	− notq ~

- **Logical shift** – Fills left end with zeros (unsigned)
- **Arithmetic shift** – Sign-extends left end (signed)

Operand Types:

- **Immediate:** $\$Imm \to$ value is Imm
- **Register:** $r_a \to$ value is $R[r_a]$
- **Memory:** $Imm(r_b, r_i, s) \to \mathrm{M}[Imm + R[r_b] + R[r_i] \cdot s]$
- $Imm(r_b, r_i, s) \to Imm + R[r_b] + R[r_i] * s$

Operands (3 types)

Type	Form	Operand value
Immediate	$\$Imm$	Imm
Register	r_a	$R[r_a]$
Memory	$Imm(r_b, r_i, s)$	$\mathrm{M}[Imm + R[r_b] + R[r_i] \cdot s]$

16 general purpose registers storing 64-bit values

Type	64-bits	32-bits	16-bits	8-bits
Registers below are **caller saved**				
Return val	%rax	%eax	%ax	%al
1st arg	%rdi	%edi	%di	%dil
2nd arg	%rsi	%esi	%si	%sil
3rd arg	%rdx	%edx	%dx	%dl
4th arg	%rcx	%ecx	%cx	%cl
5th arg	%r8	%r8d	%r8w	%r8b
6th arg	%r9	%r9d	%r9w	%r9b
Caller	%r10	%r10d	%r10w	%r10b
Caller	%r11	%r11d	%r11w	%r11b

`%rsp` is the **top of the stack** and `%rbp` is the **bottom**

Conditional Control

`set D` and `jmp` suffixes

Instruction	Alias.	Cond.	Desc.
-e	-z	ZF	= /0
-ne	-nz	~ZF	! =/not zero
-s		SF	Neg
-ns		~SF	Nonneg
-g	-nle	~(SF^OF)&~ZF	signed >
-ge	-nl	~(SF^OF)	signed =>
-l	-nge	SF^OF	signed <
-le	-ng	(SF^OF)|ZF	signed <=
-a	-nbe	~CF&~ZF	unsigned >
-ae	-nb	~CF	unsigned >=
-b	-nae	CF	unsigned <
-be	-na	CF|ZF	unsigned <=

`cmpq B, A, jg L1` equals to `if (A > B) goto L1;`
`setne %al` sets `%al` to 1 if not equal, 0 otherwise

- Carry flag (`CF`) 最高位进位，检查无符号操作溢出。
- Zero flag (`ZF`) 最近的操作：结果为 0.
- Sign flag (`SF`) 最近的操作：结果为负数。
- Overflow flag (`OF`) 补码溢出，检查有符号操作溢出。
- `test` 模拟按位 `and`，不保留结果，只改变寄存器。
- `cmp B, A` 模拟减法 `A-B`，不保留结果，只改变寄存器。
- **Switch 语句**: 跳转表 jtab[索引 i] 对应 x=i 的地址。
- 编译器通常生成 $leaq + jmp * (\%reg)$

Machine Data

- K 字节数据需存放在地址为 K 的倍数位置 (Alignment)
- 基本类型大小：

Size (bytes)	Types
1	`char`
2	`short`
4	`int, float`
8	`long, double, char *`

- **Structs**:
- 每个字段的起始地址需满足自身类型对齐要求
- 字段间可能插入 padding
- 结构体总对齐要求 = 所有字段最大对齐
- 结构体总大小 = 最大对齐的整数倍
- Example: `struct {char a; int b; char c;}` →

12 bytes

0	1	2	3	4	5	6	7	8	9	10	11
a	pad			b				c	pad		

- Nested structs: 内层 struct 按自身对齐计算后整体视为一个字段
- Access field at offset d: `movq d(%rdi),%rax`
- **Unions**:
- 所有字段共享内存起始地址（偏移为 0）
- 联合体总大小 = 最大成员大小
- Example: `union {int i; float f; char c[4];}`

occupies 4 bytes total

0	1	2	3
i (4 bytes)			
f (4 bytes)			
c[0]	c[1]	c[2]	c[3]

- Access any member using base address: `movl (%rdi), %eax`
- Used for type punning 双关 or memory optimization
- `union{float f; int i;};` 是用来查看浮点数内部位模式.

SIMD

- `setzero`, `load` / `add`, `hadd` / `cast`, `extract` / `store`, `cvtss`
- `ps(float)`, `pd(double)`, `epi32(int)`

```
float arraySumVector(const float* values, int N) {
    const int W = 8; int i = 0;
    __m256 sum256 = _mm256_setzero_ps();
    for (; i <= N - W; i += W) {
        __m256 vec = _mm256_loadu_ps(values + i);
        sum256 = _mm256_add_ps(sum256, vec); }
    __m128 low = _mm256_castps256_ps128(sum256);
    __m128 high = _mm256_extractf128_ps(sum256, 1);
    __m128 sum = _mm_add_ps(low, high);
    sum = _mm_hadd_ps(sum, sum);
    sum = _mm_hadd_ps(sum, sum);
    float ans = _mm_cvtss_f32(sum);
    for(; i < N; i++) ans += values[i];
    return ans;
}
```

GNU Debugger

- `run kill step next stepi continue`
- `finish`: Continue until current function returns
- `break <function name> / <file>:<line number> / *<address>`
- `delete <num> clear enable / disable <num>`
- `info / breakpoints / registers / functions / locals / args / displays`
- `layout, layout asm, regs, src, update`
- `print [/FMT] <expr>`
- `x /[N][FMT][SIZE] <addr> x/4xw $rsp`
- see `help x` in GDB. hex(16),dec(10),oct(8),bin(2).

Linking

- **ELF**. 链接关注: Section, 执行关注: Segment.
- **Symbols** 不包括 Local non-static，与 linker 无关。
- **Global**: 函数和普通全局变量。
- Global 分为 **Strong**: 函数/初始化的; **Weak**: 未初始化。
- **External**: 定义在其他文件中。
- **Local**: 用 `static` 定义的函数和全局变量。
- `COMMON`: 未初始化的全局变量。
- `.bss`: 未初始化静态变量，为 0 的全局和静态变量。
- **Relocation.** PC32 是相对寻址，32 是绝对寻址。
- `offset` 是代码中的位置，`addend` 是跳转偏移量。

Processes

- `getpid()`, `getppid()`-parent, `exit(int)` 永不返回
- `fork(void)` pid-父, 0-子; 返回两次，先后顺序不定
- Processes are running, stopped, or terminated
- `wait(int *statusp)` ≅ `waitpid(-1, &status, 0)`
- `sleep`, tp̄ause(), `execve()`

Signals

- Signals 不是按顺序处理的。
- `setpgid(0, 0)` 会以当前 PID 作为 ID 创建新进程组
- `sigprocmask(int how, sigset_t *set, sigset_t *oldset)`
- `sigemptyset(sigset_t *set)`
- `sigfillset(sigset_t *set)`
- `sigaddset(sigset_t *set, int sig)` – add `sig`
- `sigdelset(sigset_t *set, int sig)` – delete `sig`
- `sigsuspend(sigset_t *mask)`

System I/O

- `open, open("foo.txt", O_RDWR | O_CREAT);`
- `read, read(fd, buf, 100);`
- `write, write(fd, msg, strlen(msg));`
- 每个进程都有唯一的描述符表，描述符是小整数。
- 操作系统维护所有进程共享的打开文件表
- 每个条目都有文件位置、引用计数和 v-node 表的指针
- 操作系统维护 v-node 表，其中包含有关每个文件的信息
- `fork()` 后子进程会继承父进程的文件描述符表
- 父进程和子进程必须关闭文件，以便内核删除文件表条目
- `dup(int oldfd), dup2(int oldfd, int newfd)`
- `0-stdin, 1-stdout, 2-stderr, 3... for dup`

Tiny Shell

```
void eval_external(struct cmdline_tokens tok, int bg, char *cmdline) {
    pid_t pid; sigset_t mask, mask_old;
    sigemptyset(&mask); sigaddset(&mask,SIGCHLD);
    sigaddset(&mask,SIGINT);sigaddset(&mask, SIGTSTP);
    sigprocmask(SIG_BLOCK, &mask, &mask_old);
    if((pid = fork()) == 0) {
        sigprocmask(SIG_SETMASK, &mask_old, NULL); setpgid(0, 0);
        execve(tok.argv[0], tok.argv, environ);
        sio_put("Command not found: %s\n", tok.argv[0]);exit(0);
    } else { addjob(job_list, pid, bg ? BG : FG, cmdline);
        sigprocmask(SIG_SETMASK, &mask_old, NULL);
        if(bg) {
            printf("[%d] (%d) %s\n",pid2jid(pid),pid,cmdline); }
        else { sigset_t mask_zero; sigemptyset(&mask_zero);
            while(fgpid(job_list)==pid)sigsuspend(&mask_zero);}}}
void eval_builtin_bg(struct cmdline_tokens tok) {
    int jid = atoi(tok.argv[1] + 1);
    struct job_t *job = getjobjid(job_list, jid);
    if(job==NULL){printf("%s: No such job\n",tok.argv[1]);return;}
    pid_t pid = job->pid; kill(pid, SIGCONT); job->state = BG;
    printf("[%d] (%d) %s\n", jid, pid, job->cmdline); }
void eval_builtin_fg(struct cmdline_tokens tok) {
    int jid = atoi(tok.argv[1] + 1);
    struct job_t *job = getjobjid(job_list, jid);
    if(job == NULL) {
        printf("%s: No such job\n", tok.argv[1]); return; }
    pid_t pid = job->pid; kill(pid, SIGCONT);
    job->state = FG; sigset_t mask_zero; sigemptyset(&mask_zero);
    while(pid == fgpid(job_list)) sigsuspend(&mask_zero); }
int get_kill_pid(char *arg, pid_t *kill_pid, int *is_job_id) {
    if(arg[0] == '%')  {
        *is_job_id = 1; long jid = strtol(arg + 1, NULL, 10);
        jid = jid < 0 ? -jid : jid;
        struct job_t *job = getjobjid(job_list, jid);
        if(!job) return -1; *kill_pid = -job->pid; }
    else {
        *is_job_id = 0; long pid = strtol(arg, NULL, 10);
        if(!pid) return -1; *kill_pid = (pid_t)pid; } return 0; }
void eval_builtin_kill(struct cmdline_tokens tok) {
    if(tok.argv[1] == NULL) {
       printf("kill command requires PID or %%jobid argument\n");
       return; }
    pid_t kill_pid; int is_job_id;
    if(get_kill_pid(tok.argv[1], &kill_pid, &is_job_id) < 0)  {
        if(is_job_id) printf("%s: No such job\n", tok.argv[1]);
        else printf("%s: No such process group\n", tok.argv[1]);
        return; }
    kill(kill_pid, SIGTERM);
}
void eval_builtin_nohup(char *cmdline) {
    sigset_t mask, mask_old;
    sigemptyset(&mask); sigaddset(&mask, SIGHUP);
    sigprocmask(SIG_BLOCK, &mask, &mask_old);
    eval(cmdline + 6); sigprocmask(SIG_SETMASK, &mask_old, NULL);
}
void eval(char *cmdline) { //...
int new_input_fd, new_output_fd, old_input_fd, old_output_fd;
if(tok.infile != NULL)  { old_input_fd = dup(STDIN_FILENO);
    new_input_fd = open(tok.infile, O_RDONLY);
    dup2(new_input_fd, STDIN_FILENO); close(new_input_fd); }
if(tok.outfile != NULL) { old_output_fd = dup(STDOUT_FILENO);
    new_output_fd=open(tok.outfile,O_WRONLY | O_TRUNC | O_CREAT);
    dup2(new_output_fd, STDOUT_FILENO); close(new_output_fd); }
switch(tok.builtins) {
    case BUILTIN_NONE: eval_external(tok, bg, cmdline); break;
    // ... }
if(tok.infile != NULL) {
    dup2(old_input_fd,STDIN_FILENO);close(old_input_fd); }
if(tok.outfile != NULL) {
    dup2(old_output_fd,STDOUT_FILENO);close(old_output_fd);}
return;
void sigchld_handler(int sig) {
    int errno_old = errno;
    pid_t pid; int status; struct job_t *job;
    sigset_t mask, mask_old;
    sigfillset(&mask);
    sigprocmask(SIG_BLOCK, &mask, &mask_old);
    while((pid = waitpid(-1, &status,
                  WNOHANG | WUNTRACED | WCONTINUED)) > 0) {
        job = getjobpid(job_list, pid);
        if (WIFEXITED(status)) deletejob(job_list, pid);
        else if (WIFSTOPPED(status)) {
            sio_put("Job [%d] (%d) stopped by signal %d\n",
                        pid2jid(pid), pid, WSTOPSIG(status));
            job->state = ST; }
        else if (WIFSIGNALED(status)) {
            sio_put("Job [%d] (%d) terminated by signal %d\n",
                        pid2jid(pid), pid, WTERMSIG(status));
            deletejob(job_list, pid); }
        else if (WIFCONTINUED(status)) job->state = BG;
    }
    sigprocmask(SIG_SETMASK, &mask_old, NULL);
    errno = errno_old; return; }
void sigint_handler(int sig)  {
    int errno_old = errno; pid_t pid = fgpid(job_list);
    if(pid > 0) kill(-pid, SIGINT);
    errno = errno_old; return; }
```

Optimiaztion

```
for (int bt = 0; bt < BT; bt += 8) {
    const float *in0 = inp + (bt + 0) * C; /*...*/
    for (int o = 0; o < OC; ++o) {
        float val0 = bias ? bias[o] : 0.0f;
        float val1 = val0, val2 = val0; /*...*/
        float *row_addr = &weight[o * C];
        for (int i = 0; i < C; ++i) {
            float w = row_addr[i];
            float t0 = in0[i]; float t1 = in1[i]; /*...*/
            val0 += t0 * w; val1 += t1 * w; /*...*/ }
        out[(bt + 0) * OC + o] = val0; /*...*/
    }
}
void advanced_memset(void *s, int c, size_t n) {
    unsigned char *schar = s;
    uintptr_t addr_val = (uintptr_t)schar;
    unsigned char c_byte = (unsigned char)c;
    unsigned long c_long = 0;
    for(size_t i = 0; i < WORD_SIZE; ++i)
        { c_long = (c_long << 8) | c_byte; }
    while(n > 0 && (addr_val % WORD_SIZE != 0)) {
        byte_write(schar, c_byte);
        --n; ++schar; addr_val = (uintptr_t)schar; }
    unsigned long *word_ptr = (unsigned long *)schar;
    size_t num_words = n / WORD_SIZE;
    while(num_words >= 4) {
        word_write(word_ptr + 0, c_long); /*...*/
        word_ptr += 4; num_words -= 4; }
    while(num_words > 0) {
        word_write(word_ptr, c_long);
        ++word_ptr; --num_words; }
    schar = (unsigned char *)word_ptr;
    size_t remaining_bytes = n % WORD_SIZE;
    while(remaining_bytes > 0) {
        byte_write(schar, c_byte);
        --remaining_bytes; ++schar; }
}
```

Created at Wuhan University in Spring 2025
by TonyYin & Pittow2
https://github.com/TonyYin0418/csapp-cheat-sheet
Based on https://git.io/JcZ29