Integers

n	2^n	Hex	Binary	n	2^n
0	1	0x00	00000	17	131072
1	2	0x01	00001	18	262144
2	4	0x02	00010	19	524288
3	8	0x03	00011	20	1048576
4	16	0x04	00100	21	2097152
5	32	0x05	00101	22	4194304
6	64	0x06	00110	23	8388608
7	128	0x07	00111	24	16777216
8	256	0x08	01000	25	33554432
9	512	0x09	01001	26	67108864
10	1024	0x0A	01010	27	134217728
11	2048	0x0B	01011	28	268435456
12	4096	0x0C	01100	29	536870912
13	8192	0x0D	01101	30	1073741824
14	16384	0x0E	01110	31	2147483648
15	32768	0x0F	01111	32	4294967296
16	65536	0x10	10000	33	8589934592

- **Word size** – nominal size of pointer data
- Addresses go up to $2^w - 1$ for w-bit word size
- **Little Endian** – least significant byte comes first
- **Big Endian** – most significant byte comes first
- Example of storing `0x1234567` at `0x100`

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

- Arithmetic between signed and unsigned values automatically casts all signed values to unsigned
- $-TMin_w = TMin_w$
- $\lceil x/2^k \rceil$ is given by `(x + (1 << k) - 1) >> k`
- $x/2^k$ is given by `(x<0 ?  x+(1<<k)-1 :  x) >> k`

Value	Word size w		
	8	16	32
$UMax_w$	0xFF 255	0xFFFF 65,535	0xFFFFFFFF 4,294,967,295
$TMin_w$	0x80 -128	0x8000 $-32,768$	0x80000000 $-2,147,483,648$
$TMax_w$	0x7F 127	0x7FFF 32,767	0x7FFFFFFF 2,147,483,647
-1	0xFF	0xFFFF	0xFFFFFFFF

Bitwise Operations

- **Logical shift** – Fills left end with zeros
- **Arithmetic shift** – Sign-extends left end

~	
0	1
1	0

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

Floating Point

- Floating point lacks associativity
- $V = (-1)^s \times M \times 2^E$
- Sign bit s – whether the number is positive or negative, represented by 1-bit field
- Exponent E weights the value by a possibly negative power of 2, represented by k-bit `exp` field
- Significand (mantissa) M – fractional binary number between 1 and $2 - \varepsilon$ or between 0 and $1 - \varepsilon$, represented by n-bit `frac` field $(f_{n-1} \cdots f_1 f_0)$
- Normalized values
  - Most common case
  - `exp` is neither all zeros nor all ones
  - Exponent field represents biased signed integer
  - $E = e - \text{Bias}$ where e is the unsigned number in `exp` and $\text{Bias} = 2^{k-1} - 1$
  - `frac` represents $0 \leq f < 1$ with $0.f_{n-1} \cdots f_1 f_0$ and $M = 1 + f$ – implied leading 1
- Denormalized values
  - Exponent field all zeros
  - Exponent value is $E = 1 - \text{Bias}$, significand value is $M = f$ (no leading 1)
  - Numbers close to zero (inclusive), evenly spaced near 0.0
- Special values
  - Exponent field is all ones
  - Fraction field all zeros can represent $\pm\infty$, depending on sign bit
  - Nonzero fraction field is NaN
- Rounding
  - Rounds to the nearest even
  - `BBGRXXXX`
  - `G` – Guard bit; least significant bit of result
  - `R` – Round bit; first bit removed
  - `XXXX` – Sticky bit; OR of remaining bits
  - Round up conditions:
    - Round = 1, Sticky = 1 $\rightarrow >0.5$
    - Guard = 1, Round = 1, Sticky = 0 $\rightarrow$ Round to even
- 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 $M_1 \times M_2 = M \geq 2$ shift M right, increment $E = E_1 + E_2$
  - If E out of range, overflow
  - Round M to fit `frac` precision

x86-64 Data Alignment

- Internal padding – added between struct elements
- External padding – added after struct elements
- The entire struct is externally padded to align to its largest element

K	Types
1	`char`
2	`short`
4	`int, float`
8	`long, double, char *`

Caches

- $M = 2^m$ unique addresses of m bits
- $S = 2^s$ cache sets
- Each set consists of E cache lines
- Each line consists of a data block of $B = 2^b$ bytes, a valid bit and $t = m - (b + s)$ tag bits
- Capacity of a cache is $C = S \times E \times B$
- Address

t bits	s bits	b bits
$\leftarrow m - 1$		$0 \rightarrow$
Tag	Set index	Block offset

- Direct-mapped cache has one line per set ($E = 1$)
- Non-direct caches sometimes referred to as E-way set associative cache
- Fully-associative cache has one set ($E = C/B$).

Conditional Control

- Carry flag (`CF`) – most recent op generated carry of most significant bit, detects overflow for unsigned
- Zero flag (`ZF`) – most recent op yielded zero
- Sign flag (`SF`) – most recent op yielded negative value
- Overflow flag (`OF`) – most recent op caused two's complement overflow
- `test` instruction behaves like `and` instructions but sets condition codes without altering source or destination often see `testq %rax,%rax` to check if return val is neg, zero, or pos

`set D` and `jmp` suffixes

Instruction	Syn.	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 $<=$

Assembly Basics

- "word" refers to 16-bit data type, with "double word" referring to 32-bit (int) and 64-bit quantities referred to as "quad words"
- On 64-bit machines pointers are 8-byte quad words
- 16 general purpose registers storing 64-bit values (register file)
- In operands, scaling factor s must be either 1, 2, 4, or 8
- `mov S, D` has the effect of $S \rightarrow D$
- `movzbq` moves from byte to quad with zero-extended whereas `movsbq` does the same but sign-extended
- Stack grows down if increasing addresses grow up – "top" of the stack at the bottom
- `leaq S, D` has the effect of $\&S \rightarrow D$

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

Type	**64-bits**	**32-bits**	**16-bits**	**8-bits**
Return val	`%rax`	`%eax`	`%ax`	`%al`
Callee	`%rbx`	`%ebx`	`%bx`	`%bl`
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`
Callee	`%rbp`	`%ebp`	`%bp`	`%bpl`
Stack ptr	`%rsp`	`%esp`	`%sp`	`%spl`
Caller	`%r10`	`%r10d`	`%r10w`	`%r10b`
Caller	`%r11`	`%r11d`	`%r11w`	`%r11b`
Callee	`%r12`	`%r12d`	`%r12w`	`%r12b`
Callee	`%r13`	`%r13d`	`%r13w`	`%r13b`
Callee	`%r14`	`%r14d`	`%r14w`	`%r14b`
Callee	`%r15`	`%r15d`	`%r15w`	`%r15b`

Linking

- Relocatable object files – combine with other relocatables at compile time to create executables, made by compiler and assembler
- Executable object files – contain binary data that can be directly copied to memory and executed, made by linker
- Shared object files – special relocatable objects that can be linked dynamically at load or run time
- Static symbols – defined locally to an object file (module)
- Global symbols – defined locally and referred to elsewhere
- Externals – Global symbols referenced locally but defined elsewhere
- Local linker symbols are different from local program variables
- Functions and initialized global variables are exported to the assembler as strong by the compiler
- Uninitialized global variables are weak
- Linkers error on multiple same-name strong symbols, pick strong over weak, and randomly choose from weak symbols

Processes

- `getpid(void)` – `pid_t` of the process
- `getppid(void)` – `pid_t` of the parent process
- Processes are running, stopped, or terminated
- `exit(int status)` – called once, never returns
- `fork(void)` – `pid_t` of child in parent or 0 if in child; called once, returns twice
- Open file table and vnode table are managed by OS and shared across processes, file descriptor table is process-specific
- `waitpid(pid_t pid, int *statusp, int options)` – returns `pid_t` of child, 0 if not waiting (`WNOHANG`) or error
  - `WNOHANG` – return (0) immediately and don't wait for child
  - `WUNTRACED` – check for terminated and stop children
  - `WCONTINUED` – check for waiting (child) process to be continued
  - OR (`|`) flags together to form a bit vector
  - `WIFEXITED(status)` – whether child exited normally
  - `WEXITSTATUS(status)` – returns exit status of terminated child if `WIFEXITED` is true
  - `WIFSIGNALED & WTERMSIG` – as above but for signals
  - `WIFSTOPPED & WSTOPSIG & WIFCONTINUED` – as above for stopped/continued processes
- `wait(int *statusp)` $\cong$ `waitpid(-1, &status, 0)`
- `sleep(unsigned int secs)` – returns short counts
- `pause(void)` – sleeps until a signal is received
- `execve(char *filename, char *argv[], char *envp[])` – does not return unless error; last arg in each array is `NULL`

Virtual Memory

- $N = 2^n$ addresses in n-bit virtual address space
- $M = 2^m$ addresses in m-bit physical address space (not necessarily power of 2)
- $P = 2^p$ bytes per virtual/physical page
- Virtual pages are either unallocated, cached (allocated in PM), or uncached (allocated, not in PM)
- DRAM caches are often fully associative
- Page table maps virtual pages to physical pages
- Valid bit in PTE set indicates cached in PM, rest indicates virtual or physical address (depending on valid bit)
- Page fault is DRAM cache miss; triggers exception
- Translation lookaside buffer (TLB) is a cache of PTEs; each line holds a block with one PTE; highly associative
- TLB has $T = 2^t$ sets
- Virtual address

Virtual Page Number		
$\leftarrow n-1$	$\leftarrow p+t \mid p \rightarrow$	$0 \rightarrow$
TLB Tag	TLB index	Virtual page offset

Dynamic Memory Allocation

- `sbrk(intptr_t incr)` – extend the heap by `incr` (basically just an `int`) and return the old break pointer

Signals

- Default action of `SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP` is to ignore and stop (respectively per pair)
- Signals can be sent to process groups; child inherits parent process group by default; `setpgid(0, 0)` sets the process group ID to the current process id
- `kill(pid_t pid, int sig)` – send a `sig` to `pid`, unless `pid` is 0 then send it to every process in the process group of the calling process; if `pid` < 0, send to every process in process group $|$`pid`$|$
- `signal(int sig, void *hndlr_t (int) handler)` – handle `sig` with `handler` function pointer; `handler` can be `SIG_IGN` to ignore or `SIG_DFL` for default
- `sigprocmask(int how, sigset_t *set, sigset_t *oldset)`
  - `SIG_BLOCK` – `blocked = blocked | set`
  - `SIG_UNBLOCK` – `blocked = blocked & set`
  - `SIG_SETMASK` – `blocked = set`
  - Old bit vector stored in `oldset` (a.k.a. `prev`)
- `sigemptyset(sigset_t *set)`
- `sigfillset(sigset_t *set)`
- `sigaddset(sigset_t *set, int sig)` – add `sig`
- `sigdelset(sigset_t *set, int sig)` – delete `sig`
- Rules for signal handlers
  1. Keep them simple
  2. Only call async-signal-safe functions (reentrant or uninterruptible)
  3. Save and restore `errno`
  4. Block all signals
  5. Declare global variables with `volatile` – force memory read each time (no storage in registers)
  6. Declare flags with `sig_atomic_t` – atomic r/w
- Signals are **not** queued
- `sigsuspend(sigset_t *mask)` – atomically replace `blocked` with `mask` and suspend until handler returns after receipt of a signal, then restore `blocked`

Concurrency with Threads

- `pthread_create(pthread_t *tid, NULL, void *func(void *), void *arg)` – run `func` with `arg` in a new thread, joinable by default
- `pthread_self(void)` – return current thread id
- `pthread_exit(void *return)` – exit the current thread
- `pthread_cancel(pthread_t tid)` – terminate another thread without waiting
- `pthread_join(pthread_t tid, NULL)` – block and wait for the thread with `tid` to terminate
- `pthread_detach(pthread_t tid)` – make thread `tid` detached (not joinable), often called on self
- Threads share everything in memory except for registers and stack, though they can access addresses in other thread stacks

System I/O

- `open(char *filename, int flags, mode_t mode)` – returns file descriptor
  - `O_RDONLY`, `O_WRDONLY`, `O_RDWR` flags
  - `O_CREAT` – create a new file if it doesn't exist
  - `O_TRUNC` – truncate the file if it exists
  - `O_APPEND` – before write, set file pos. to end of file
  - Mode given by OR (|) combination of `S_I{R, W, X}{USR, GRP, OTH}`
  - Each call creates new open file table entry
- `read(int fd, void *buf, size_t n)` – read up to `n` bytes into `buf` and return the number of bytes actually read, update file descriptor table position by return value
- `write(int fd, void *buf, size_t n)` – write up to `n` bytes from `buf`, return the number of bytes actually written, update file descriptor table position by return value
- Each process has unique descriptor table pointing to entries in global file table
- OS maintains open file table shared by all processes, each entry has file position, ref count, and pointer to v-node table
- OS maintains v-node table with information about each file
- Parent and child process must both close file descriptors for kernel to remove file table entry
- `dup2(int oldfd, int newfd)` – copies descriptor entry `oldfd` to `newfd`, overwriting `newfd` (closes `newfd` if open); *i.e.*, `newfd` entry points to `oldfd` entry

Thread Synchronization

- `sem_init(sem_t *sem, 0, int val)` – initialize lock
- `sem_wait(sem_t *sem)` – `P(sem)`; block until get lock
- `sem_post(sem_t *sem)` – `V(sem)`; release lock
- In producer-consumer solution, producer produces whenever there is space in buffer, and consumer consumes whatever is there as fast as it can
- First readers-writers problem favors readers
- Second readers-writers problem favors writers
- Readers-writers solutions can result in starvation
- Deadlock – threads are waiting for condition that will never be true

Network Programming

- `socket(int domain, int type, int protocol)` – usually `AF_INET, SOCK_STREAM, 0`, respectively; create endpoint of connection
- `connect(int clientfd, struct sockadd *addr, socklen_t addrlen)` – establish connection with a server at `addr` (client)
- `bind(int sockfd, struct sockaddr *adrr, socklen_t addrlen)` – associate server socket address with given socket descriptor (server)
- `listen(int sockfd, int backlog)` – set `sockfd` to actively listen (server)
- `accept(int listenfd, struct sockaddr *addr, int *addrlen)` – block until a connection is made then return file descriptor for connection

Scratch work:

Created for 15-213 at Carnegie Mellon University
in Spring 2019 by Jacob Strieb.
jstrieb@alumni.cmu.edu
https://git.io/JcZ29