15-213/18-213 Final Exam Notes Sheet Spring 2013

Jumps

Jump	Condition
`jmp`	1
`je`	ZF
`jne`	~ZF
`js`	SF
`jns`	~SF
`jg`	~(SF^OF)&~ZF
`jge`	~(SF^OF)
`jl`	(SF^OF)
`jle`	(SF^OF)|ZF
`ja`	~CF&~ZF
`jb`	CF

Arithmetic Operations

Format		Computation
`addl`	*Src,Dest*	Dest = Dest + Src
`subl`	*Src,Dest*	Dest = Dest - Src
`imull`	*Src,Dest*	Dest = Dest * Src
`idivl`	*Src*	Divide signed contents of edx:eax by Src. Quotient goes into eax and remainder in edx
`sall`	*Src,Dest*	Dest = Dest << Src
`sarl`	*Src,Dest*	Dest = Dest >> Src
`shrl`	*Src,Dest*	Dest = Dest >> Src
`xorl`	*Src,Dest*	Dest = Dest ^ Src
`adnl`	*Src,Dest*	Dest = Dest & Src
`orl`	*Src,Dest*	Dest = Dest | Src

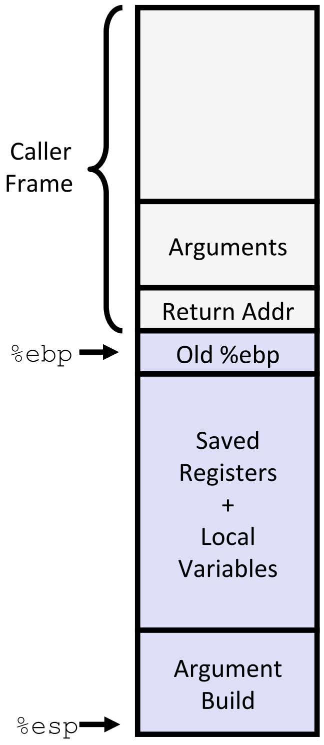
Memory Operations

Format	Computation
`(Rb, Ri)`	Mem[Reg[Rb]+Reg[Ri]]
`D(Rb,Ri)`	Mem[Reb[Rb]+Reg[Ri]+D]
`(Rb,Ri,S)`	Mem[Reg[Rb]+S*Reg[Ri]]

Registers

63	31	15 8	7 0	
%rax	%eax %ax	%ah	%al	Return value
%rbx	%ebx %bx	%bh	%bl	Callee saved
%rcx	%ecx %cx	%ch	%cl	Argument #4
%rdx	%edx %dx	%dh	%dl	Argument #3
%rsi	%esi %si		%sil	Argument #2
%rdi	%edi %di		%dil	Argument #1
%rbp	%ebp %bp		%bpl	Callee saved
%rsp	%esp %sp		%spl	Stack Pointer
%r8	%r8d %r8w		%r8b	Argument #5
%r9	%r9d %r9w		%r9b	Argument #6
%r10	%r10d %r10w		%r10b	Reserved
%r11	%r11d %r11w		%r11b	Used for linking
%r12	%r12d %r12w		%r12b	Callee saved
%r13	%r13d %r13w		%r13b	Callee saved
%r14	%r14d %r14w		%r14b	Callee saved
%r15	%r15d %r15w		%r15b	Callee saved

Linux Stack



Specific Cases of Alignment (IA32)

1 byte: char, …

no restrictions on address

2 bytes: short, …

lowest 1 bit of address must be 0_2

4 bytes: int, float, char *, …

lowest 2 bits of address must be 00_2

8 bytes: double, …

Windows (and most other OS’s & instruction sets):

lowest 3 bits of address must be 000_2

Linux:

lowest 2 bits of address must be 00_2

i.e., treated the same as a 4-byte primitive data type

12 bytes: long double

Windows, Linux:

lowest 2 bits of address must be 00_2

i.e., treated the same as a 4-byte primitive data type

C Data Type	Intel IA32	x86-64
char	1	1
short	2	2
int	4	4
long	4	8
long long	8	8
float	4	4
double	8	8
long double	10/12	10/16
pointer	4	8

Specific Cases of Alignment (x86-64)

1 byte: char, …

no restrictions on address

2 bytes: short, …

lowest 1 bit of address must be 0_2

4 bytes: int, float, …

lowest 2 bits of address must be 00_2

8 bytes: double, char *, …

Windows & Linux:

lowest 3 bits of address must be 000_2

16 bytes: long double

Linux:

lowest 3 bits of address must be 000_2

i.e., treated the same as a 8-byte primitive data type

Byte Ordering

4-byte variable 0x01234567 at 0x100

Big Endian

Least significant byte has highest address

0x100	0x101	0x102	0x103
01	23	45	67

Little Endian

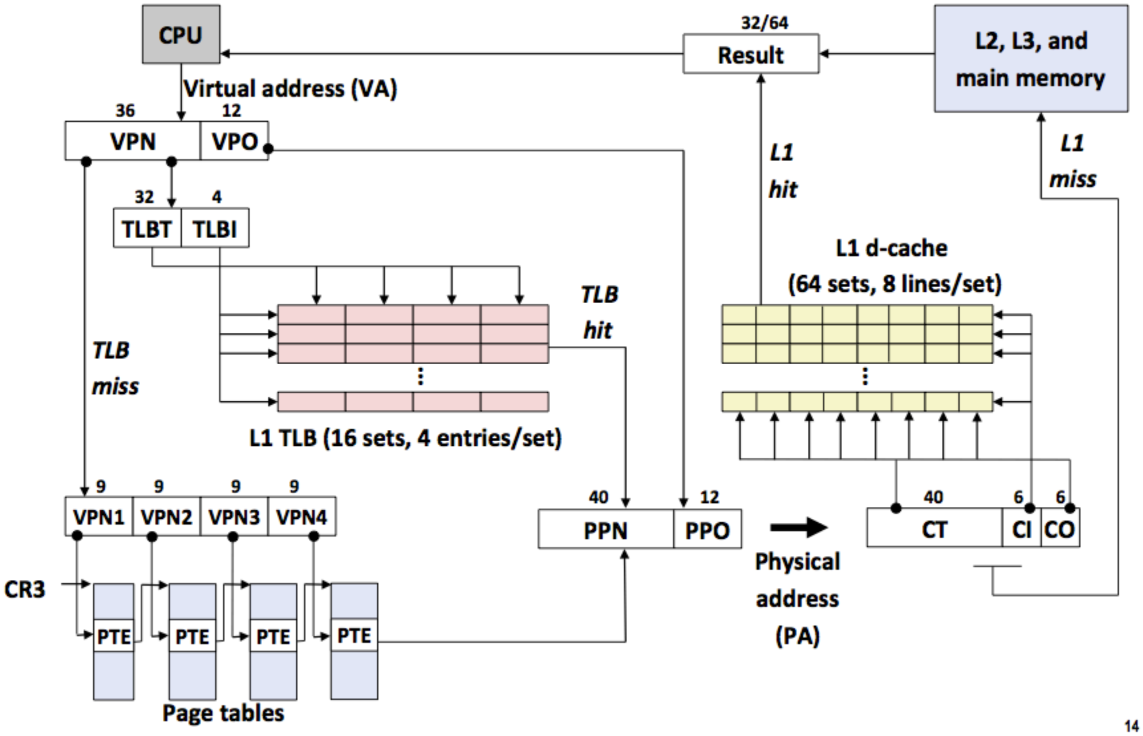
Least significant byte has lowest address

0x100	0x101	0x102	0x103
67	45	23	01

Floating Point

Bias = $2^{k-1} – 1$

End-to-end Core i7 Address Translation



VPN – Virtual Page Number
VPO – Virtual Page Offset
TLB – Translation Look-aside Buffer
PPN – Physical Page Number
PPO – Physical Page Offset
TLBT – TLB Tag
TLBI – TLB Index

NAME

execl, execlp, execle, execv, execvp - execute a file

SYNOPSIS

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg,
        ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

DESCRIPTION

The exec() family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for the function execve(2)

NAME

fork - create a child process

SYNOPSIS

pid_t fork(void);

DESCRIPTION

fork() creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0.

Under Linux, fork() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set appropriately.

NAME

dup, dup2 - duplicate a file descriptor

SYNOPSIS

int dup(int oldfd);
int dup2(int oldfd, int newfd);

DESCRIPTION

dup() and dup2() create a copy of the file descriptor oldfd.

After a successful return from dup() or dup2(), the old and new file descriptors may be used interchangeably. They refer to the same open file description (see open(2)) and thus share file offset and file status flags; for example, if the file offset is modified by using lseek(2) on one of the descriptors, the offset is also changed for the other.

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary.

NAME

wait, waitpid - wait for process to change state

SYNOPSIS

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

DESCRIPTION

All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal.

In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then terminated the child remains in a "zombie" state

If a child has already changed state, then these calls return immediately. Otherwise they block until either a child changes state or a signal handler interrupts the call.

NAME

read - read from a file descriptor

SYNOPSIS

```
ssize_t read(int fd, void *buf, size_t count);
```

DESCRIPTION

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

NAME

fflush - flush a stream

SYNOPSIS

```
int fflush(FILE *stream);
```

DESCRIPTION

The function fflush() forces a write of all user-space buffered data for the given output or update stream via the stream's underlying write function.
The open status of the stream is unaffected.

NAME

connect - initiate a connection on a socket

SYNOPSIS

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

DESCRIPTION

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. The addrlen argument specifies the size of addr. The format of the address in addr is determined by the address space of the socket sockfd; see socket(2) for further details.

If the socket sockfd is of type SOCK_DGRAM then addr is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type SOCK_STREAM or SOCK_SEQPACKET, this call attempts to make a connection to the socket that is bound to the address specified by addr.

Generally, connection-based protocol sockets may successfully connect() only once; connectionless protocol sockets may use connect() multiple times to change their association. Connectionless sockets may dissolve the association by connecting to an address with the sa_family member of sockaddr set to AF_UNSPEC (supported on Linux since kernel 2.2).

RETURN VALUE

If the connection or binding succeeds, zero is returned. On error, -1 is returned, and errno is set appropriately.

NAME

htonl, htons, ntohl, ntohs - convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

The htonl() function converts the unsigned integer hostlong from host byte order to network byte order.

The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.

The ntohl() function converts the unsigned integer netlong from network byte order to host byte order.

The ntohs() function converts the unsigned short integer netshort from network byte order to host byte order.

On the i386 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.