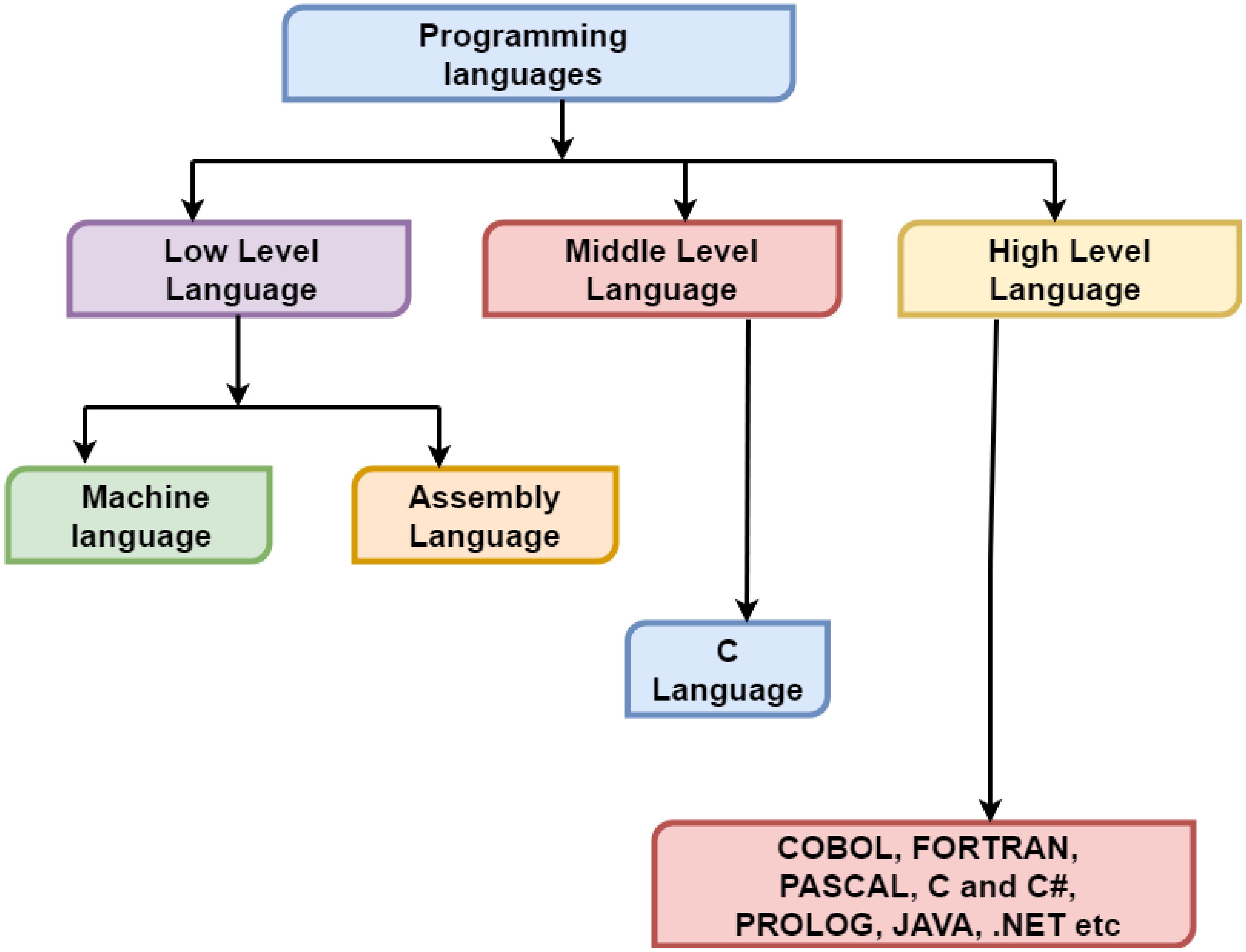
Advanced System Programming

Advanced System Programming

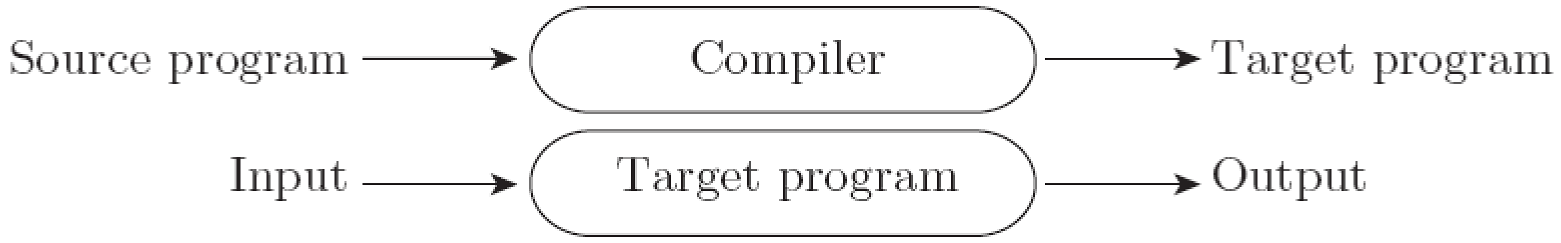
Agenda

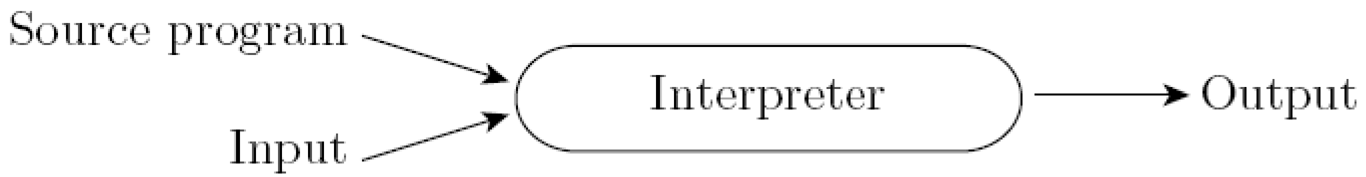
- Programming languages
- GCC compilation stages
- GCC compilation optimization techniques
- Bin utils
- Introduction to Linux
- Linux Filesystem Hierarchy
- X86 Assembly Language

Classification of Programming Languages

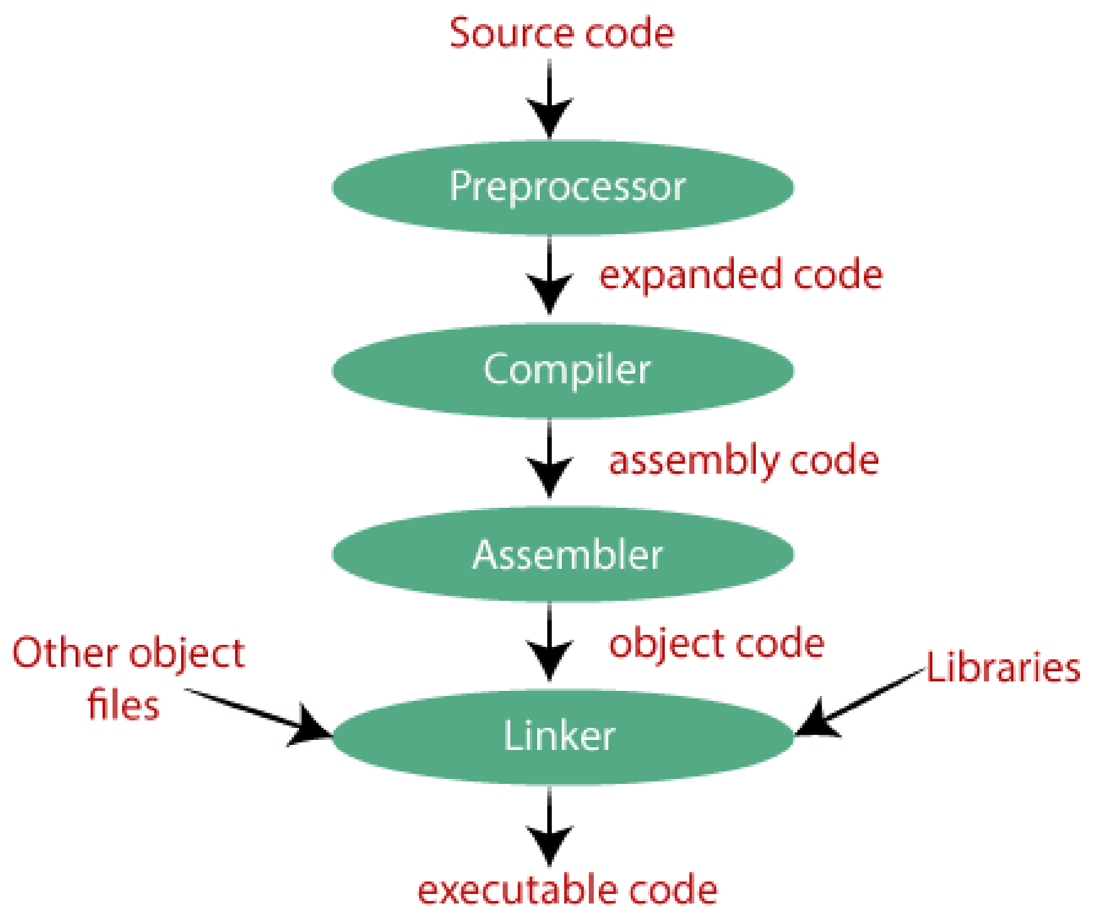


Compilation V/S Interpreration





Compilation Stages



Compilation Stages

- Normal compilation

  gcc –g –o hello hello.c

- Save all stage results

  gcc –g –o hello -save-temps hello.c

- Save desired stages

  -E – preprocessing

  -S – Assembler

  -c – Object code

- GCC optimization

  -O0

  -O1

  -O2

  -O3

  -Os

Compilation Stages – profile guided optimization

- Profile-guided optimization (PGO) also known as feedback-directed optimization (FDO) is a compiler optimization technique that provides even better performance gains than the well known -o3 optimization flag in gcc. It is however a bit more work than just setting a flag in the compilation.

- Google Chrome uses PGO since version 53 and they saw an up to 15% performance increase for Chrome on Windows. Broken down in different parts of Chrome they got a 14.8% faster new tab page load time, 5.9% faster page load and 16.8% faster startup time. All these performance gains not from optimizing their millions of lines of code, but from using a compiler optimization. Pretty impressive and this works for every program.

```
gcc -O3 -fprofile-generate=test -o sort_unopt sort.c
gcc -O3 sort.c  -o sort_opt -fprofile-use=test
```

Compiler utilites

- ldd
- objdump
- readelf
- ar
- ld

Introduction to Linux

- In the early 1990s, Torvalds became interested in a freeware product called Minix were written by Andrew S. Tanenbaum. Developed by Andrew S.Tanenbaum, Minix was a clone of the commercial UNIX operating system.

- Linux version 0.02, released on October 5, 1991, consisted of only the Linux kernel and three utilities:
  - bash : a command-line interface
  - update : a utility for flushing file system buffers
  - gcc : a C++ compiler

- Today used on 7-10 million computers with 1000's of programmers working to enhance it around the world.

Introduction to Linux

- GNU Project: Richard Stallman on September 27th 1983.
- The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system.
- GNU's kernel isn't finished, so GNU is used with the kernel Linux. The combination of GNU and Linux is the GNU/Linux operating system, now used by millions.

What is Linux

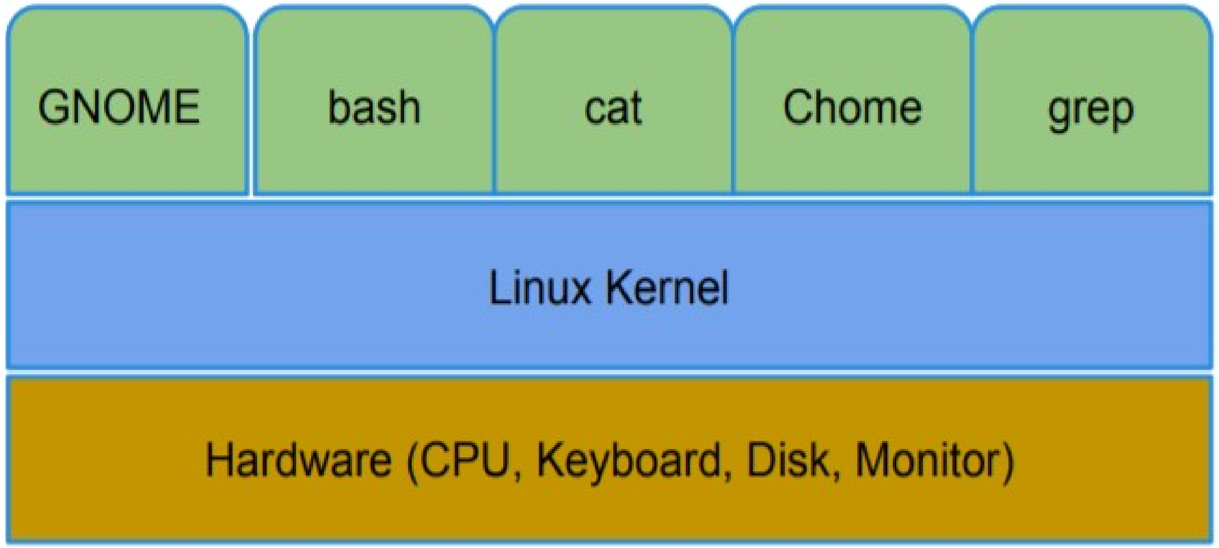
- A fully-networked 32/64-Bit Unix-like Operating System
  - Unix Tools Like sed, awk, and grep (explained later)
  - Compilers Like C, C++, Fortran, Smalltalk, Ada
  - Network Tools Like telnet, ftp, ping, traceroute
- Multi-user, Multitasking, Multiprocessor
- Has the X Windows GUI
- Coexists with other Operating Systems
- Runs on multiple platforms
- Includes the Source Code

Component of Linux

- ❑ The Linux Kernel
- ❑ Libraries
- ❑ Utilities
- ❑ User Interface

Linux Kernel

Very simple answer: it's a program that makes your hardware look and feel like an OS to other programs



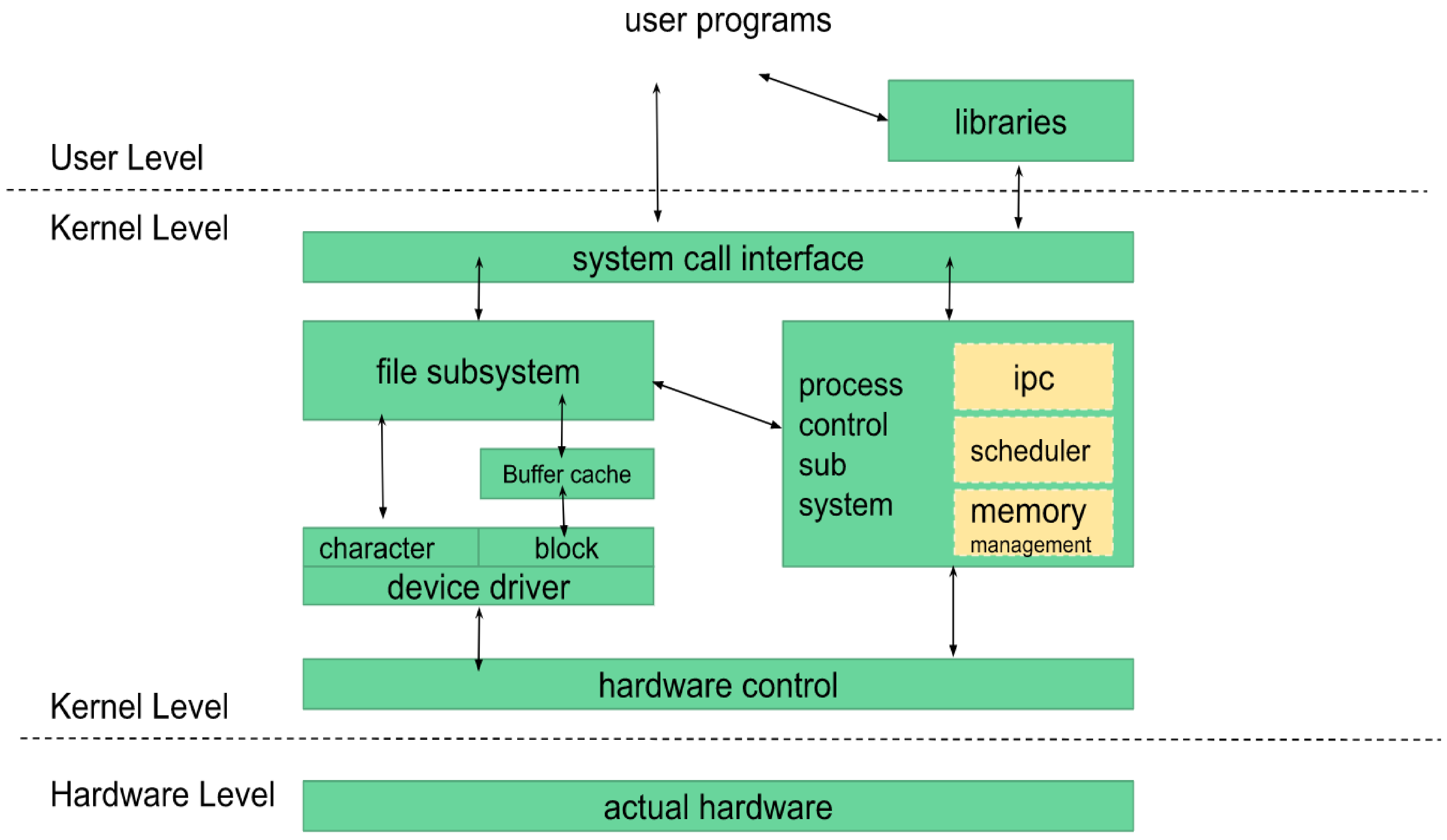
Linux Kernel

- **Libraries** are pre-written code “pieces” that application programmers use in their programs.
- **Utilities** maintaining the file system, editing text files, managing running processes, and installing new software packages.
- **User Interface** command-line interface (CLI) and a graphical user interface (GUI).

Kernel Space and User Space

- The Concept of kernel space and user space is all about memory and access rights.
- It is a feature of modern CPU, allowing it to operate either in privileged or unprivileged mode.
- One ,may consider kernel to be privileged and user apps are restricted.

Kernel Space and User Space

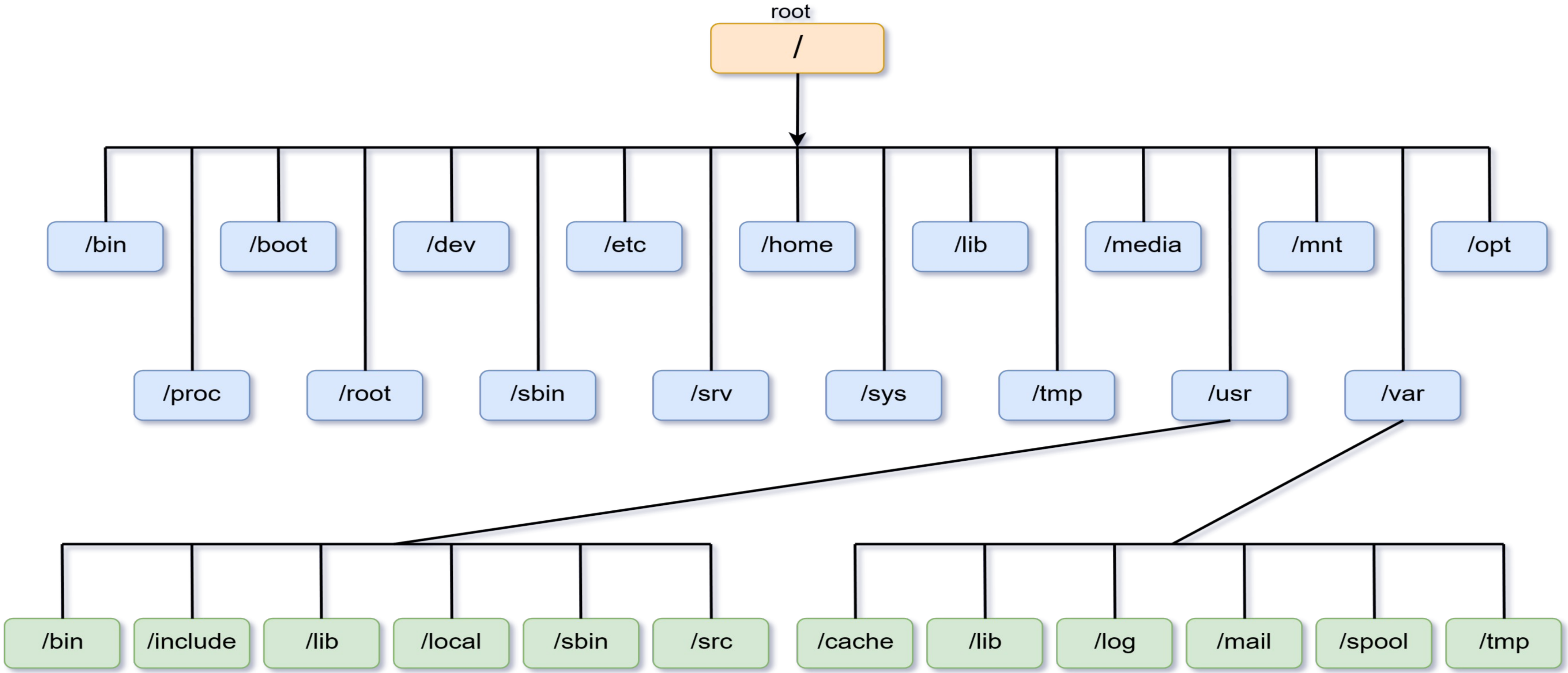


Block Diagram of the System Kernel

Linux Filesystem Hierarchy

- The Linux File Hierarchy Structure or the Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems. It is maintained by the Linux Foundation.
  - In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.
  - Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.
  - Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS and are not considered authoritative for platforms other than Linux.

Linux Filesystem Hierarchy



Linux Filesystem Hierarchy

- **/ (root)**
  - Primary hierarchy root and is the first directory and the root directory of the entire file system hierarchy.
  - It contains all other directories i.e. the sub directories.
  - Only the root user has the permissions to write here.
  - This is not the home directory for the root user
- **/bin (user binaries)**
  - It contains binary executables.
  - Binary executables of common linux commands used by all users in single-user mode are located in this directory.
  - Some files present in this directory are: ls, cp, grep, ping, cat, etc.

Linux Filesystem Hierarchy

- ❑ **/boot (boot loader files)**
  - It contains boot loader files.
  - kernel, initrd, grub, and other files and directories are located in this directory.
  - e.g. vmlinux-2.7.31.25-generic, initrd.img-2.3.32.24-generic, etc.
- ❑ **/dev (device files)**
  - It contains the essential files related to the devices attached to the system.
  - This includes terminal devices, USB, network devices and any other devices that are attached to the system.
  - e.g. /dev/usbmon0 , /dev/tty1 , /dev/null , etc.

Linux Filesystem Hierarchy

- ❑ **/etc (configuration files)**
- etc stands for ‘edit to config’
- This directory contains the configuration files that are required by the installed programs.
- The files are host-specific and system-wide configurations needed for the proper functioning of the system.
- This directory also contains shell scripts for system startup and system shutdown that are used to start or stop individual programs.
- The files in this directory should not be edited without proper knowledge of system configuration as improper configuration could brick the system.
- e.g. /etc/passwd , /etc/shadow , /etc/group , /etc/resolv.conf , etc.

Linux Filesystem Hierarchy

- **/home (home directories)**
  - This directory contains user's home directories, containing saved files and personal settings.
  - Each user will have an separate directory with their username under this directory except the root user because every time a new user is created, a directory is created in the name of the user within the home directory.
  - e.g. /home/user , /home/sage , /home/guest , etc.
- **/lib (system libraries)**
  - This directory contains libraries that are essential for the binaries in /bin and /sbin
  - Library filenames are either ld* or lib*.so.*
  - e.g. ld-2.11.1.so , etc.
- **/media (removable media devices)**
  - Temporary mount directory for removable media such as CD-ROM.
  - e.g. /media/cdrom for CD-ROM ; /media/floppy for floppy drives ; /media/cdrecorder for CD writer ; etc.

Linux Filesystem Hierarchy

- **/mnt (mount directory)**
  - Temporary mount directory where system administrator can mount file systems.
- **/opt (optional application software packages)**
  - This directory contains add-on applications from individual vendors.
- **/proc (process information)**
  - This is a virtual filesystem providing process and kernel information. This files in this directory are automatically generated, populated and deleted by the system. In Linux, corresponds to a procfs mount.
  - This directory contains information about the processes running in the system.
  - This directory also contains text information about running processes. e.g. /proc/uptime
  - e.g. /proc/{pid} directory contains information about the process with that particular pid that will be mentioned within the brackets.

Linux Filesystem Hierarchy

- ❑ **/root (root directory)**
  - This is the home directory for the root user.
- ❑ **/sbin (system binaries)**
  - This directory contains essential system binaries.
  - The linux commands that are located in this directory are used by system administrator, for system maintenance and configuration purpose.

  e.g. fsck , reboot , fdisk , ifconfig , init , etc.

Linux Filesystem Hierarchy

- ❑ **/root (root directory)**
  - This is the home directory for the root user.
- ❑ **/sbin (system binaries)**
  - This directory contains essential system binaries.
  - The linux commands that are located in this directory are used by system administrator, for system maintenance and configuration purpose.

  e.g. fsck , reboot , fdisk , ifconfig , init , etc.
- ❑ **/srv (service data)**
  - This directory contains site-specific data served by the system, such as data and scripts for web servers, data offered by FTP servers, and repositories for version control systems i.e. server specific services related data.

e.g. /srv/cvs contains CVS related data , etc.

Linux Filesystem Hierarchy

- ❑ **/sys (system)**
  - This directory contains information about devices, drivers, and some kernel features.
- ❑ **/tmp (temporary files)**
  - This directory contains temporary files created by system and the users that will be rebooted when the system is rebooted.
- ❑ **/usr (user programs)**
  - This directory contains read-only user data like binaries, libraries, documentation and source-code for second level programs like user utilities and applications.
  - /usr/bin → contains binary files for user programs. If you can’t find a user binary under /bin, then we should look under /usr/bin.
  - /usr/include → contains standard include files.

Linux Filesystem Hierarchy

- /usr/lib → contains libraries for the binaries in /usr/bin and /usr/sbin

- /usr/local → tertiary hierarchy for local data. contains users programs that you install from source. e.g., when you install apache, it goes under /usr/local/apache2

- /usr/sbin → /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, then you should look under /usr/sbin. It also contains non-essential system binaries. e.g. daemons for network-services.

❑ **/var (variable files)**

- This directory contains files whose content is expected to continually change during normal operation of the system—such as logs, spool files, and temporary e-mail file.
- /var/log → contains system log files.

X86 Register

- ❑ Little-Endian and Big-Endian Architecture

- ❑ There are sixteen, 64-bit General-Purpose Registers (GPRS) . The GPRs register can be accessed with all 64-bits or some portion or subset accessed.

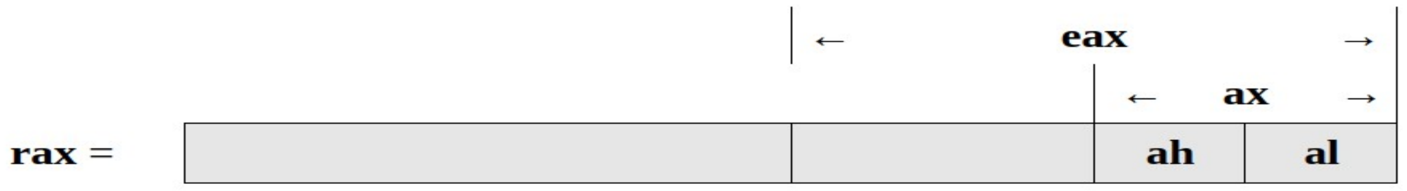
- ❑ Additionally , some of the GPR registers are used for dedicated purposes

- ❑ When using data element sizes less than 64-bits (i.e. 32-bit, 16-bit or 8-bit), the lower portion of the register can be accessed by using a different register.

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
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

X86 Register

- For example, when accessing the lower portions of the 64- bit rax register, the layout is as follows:



- The first four register rax, rbx, rcx and rdx also allow the bits 8-15 to be accessed with the ah, bh, ch and dh register names. With the exception of ah, these are provided for legacy support.

- The ability to access portions of the register means that, if the quad word rax register is set to 50,000,000,000 (fifty billion), the rax register would contain the following value in hex.

  rax = 0000 000B A43B 7400h

- If a subsequent operation sets the word ax register to 50,000decimal (fifty thousand, which is C350h), the rax register would contain the following value in hex.

  rax = 0000 000B A43B C350h

X86 Register

- ❑ When the lower 16-bit ax portion of the 64-bit rax register is set, the upper 48-bits are unaffected. Note the change in AX (from 7400h to C350h).

- ❑ If a subsequent operation sets the bytes sized al register to 50decimal (fifty, which is 32h), the rax register would contain the following values in hex.

  rax= 000 000B A43B C332h

- ❑ When the lower 8-bit al portion of the 64-bit rax register is set, the upper 56-bits are unaffected. Note the change in AL (from 50h to 32h).

- ❑ **Stack Pointer Register (RSP)**

One of the CPU registers, rsp, is used to point to the current top of the stack. The rsp register should not be used for data or other uses.

X86 Register

- **Base Pointer Register (RBP)**

One of the CPU registers, rbp, is used as a base pointer during function calls. The rbp register should not be used for data or other uses.

- **Instruction Pointer Register (RIP)**

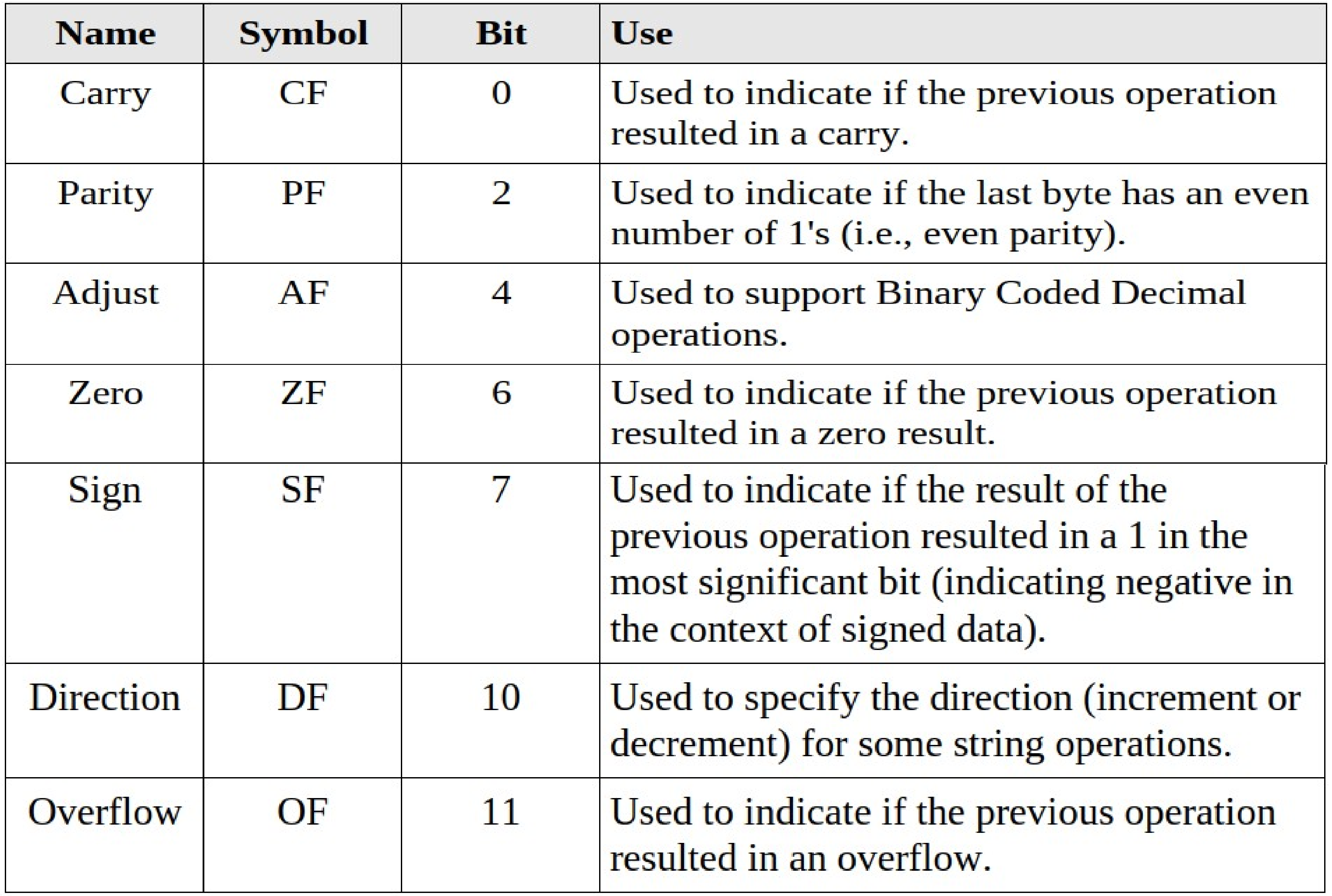
In addition to the GPRs, there is a special register, rip, which is used by the CPU to point to the next instruction to be executed.

- **Flag Register (rFlags)**

The flag register, rFlags, is used for status and CPU control information. The rFlag register is updated by the CPU after each instruction and not directly accessible by programs. This register stores status information about the instruction that was just executed. Of the 64-bits in the rFlag register, many are reserved for future use.

X86 Register

- The following table shows some of the status bits in the flag register.



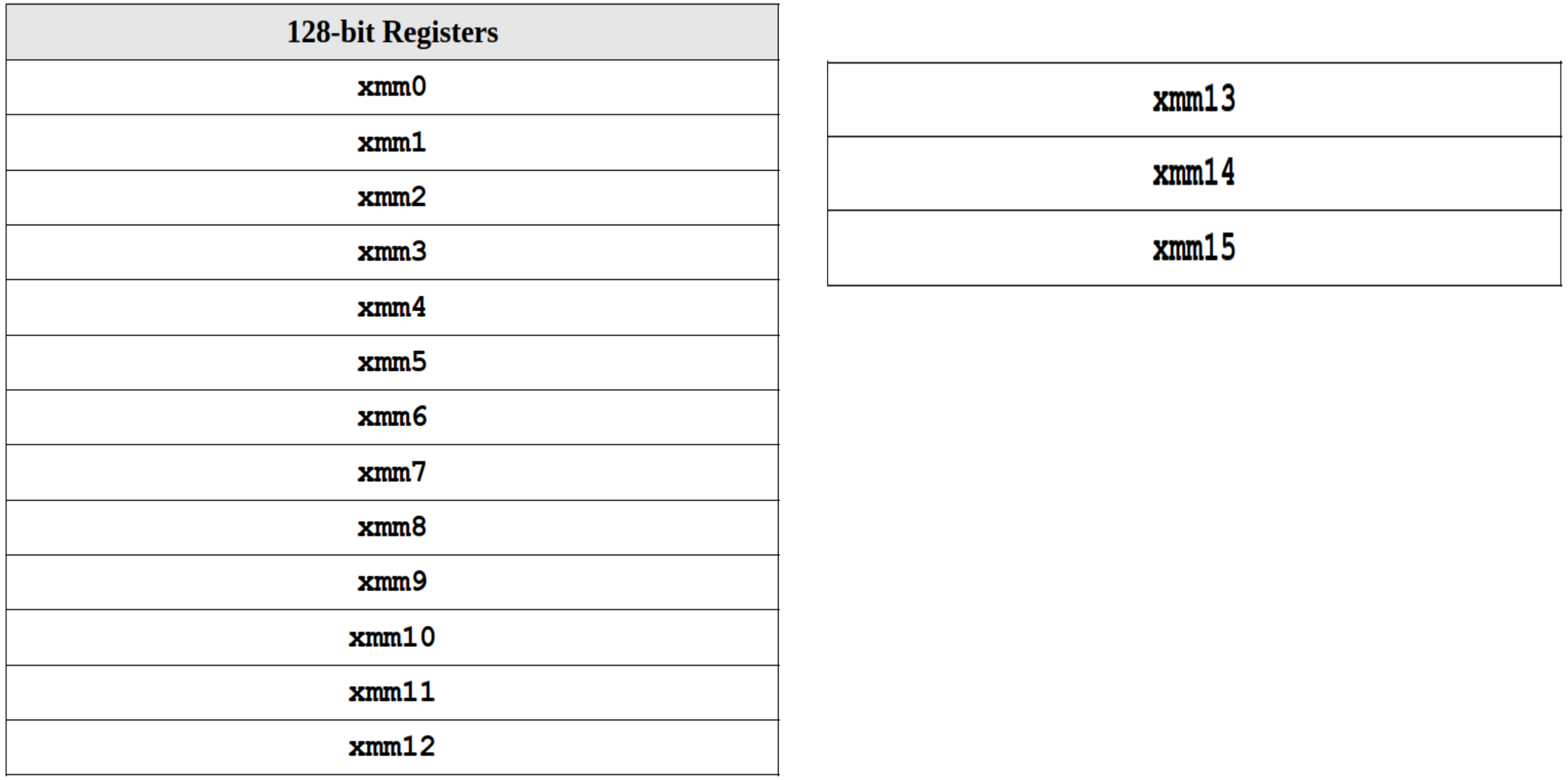
Name	Symbol	Bit	Use
Carry	CF	0	Used to indicate if the previous operation resulted in a carry.
Parity	PF	2	Used to indicate if the last byte has an even number of 1's (i.e., even parity).
Adjust	AF	4	Used to support Binary Coded Decimal operations.
Zero	ZF	6	Used to indicate if the previous operation resulted in a zero result.
Sign	SF	7	Used to indicate if the result of the previous operation resulted in a 1 in the most significant bit (indicating negative in the context of signed data).
Direction	DF	10	Used to specify the direction (increment or decrement) for some string operations.
Overflow	OF	11	Used to indicate if the previous operation resulted in an overflow.

X86 Register

❑ XMM Registers

There are a set of dedicated registers used to support 64-bit and 32-bit floating-point operations and Single Instruction Multiple Data (SIMD) instructions. The SIMD instructions allow a single instruction to be applied simultaneously to multiple data items. Used effectively, this can result in a significant performance increase. Typical applications include some graphics processing and digital signal

The XMM registers as follows:

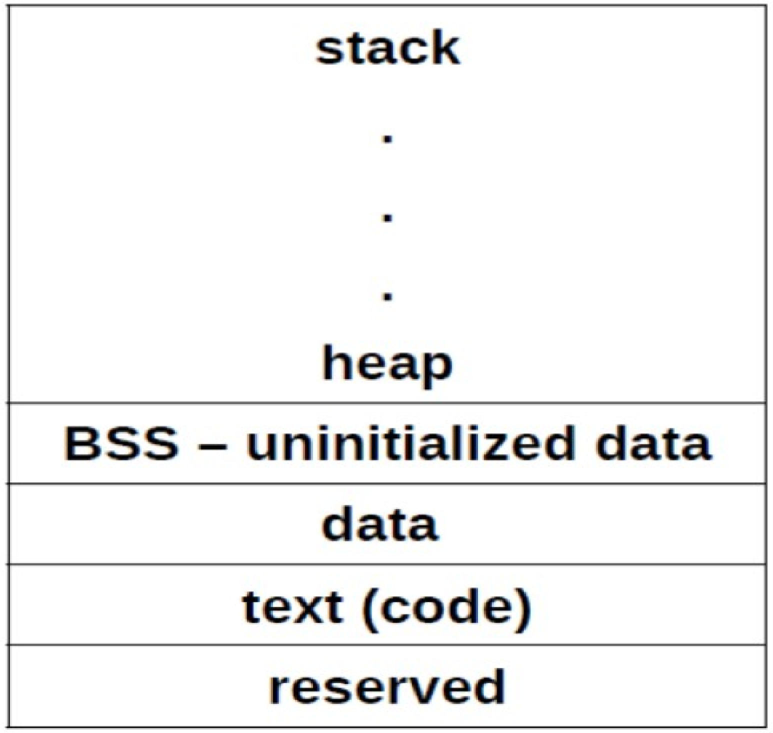


128-bit Registers
xmm0
xmm1
xmm2
xmm3
xmm4
xmm5
xmm6
xmm7
xmm8
xmm9
xmm10
xmm11
xmm12
xmm13
xmm14
xmm15

X86 Register

- Note, some of the more recent X86-64 processors support 256-bit XMM registers. This will not be an issue for the programs in this text.

- Additionally, the XMM registers are used to support the Streaming SIMD Extensions (SSE).



X86 Calling Convention

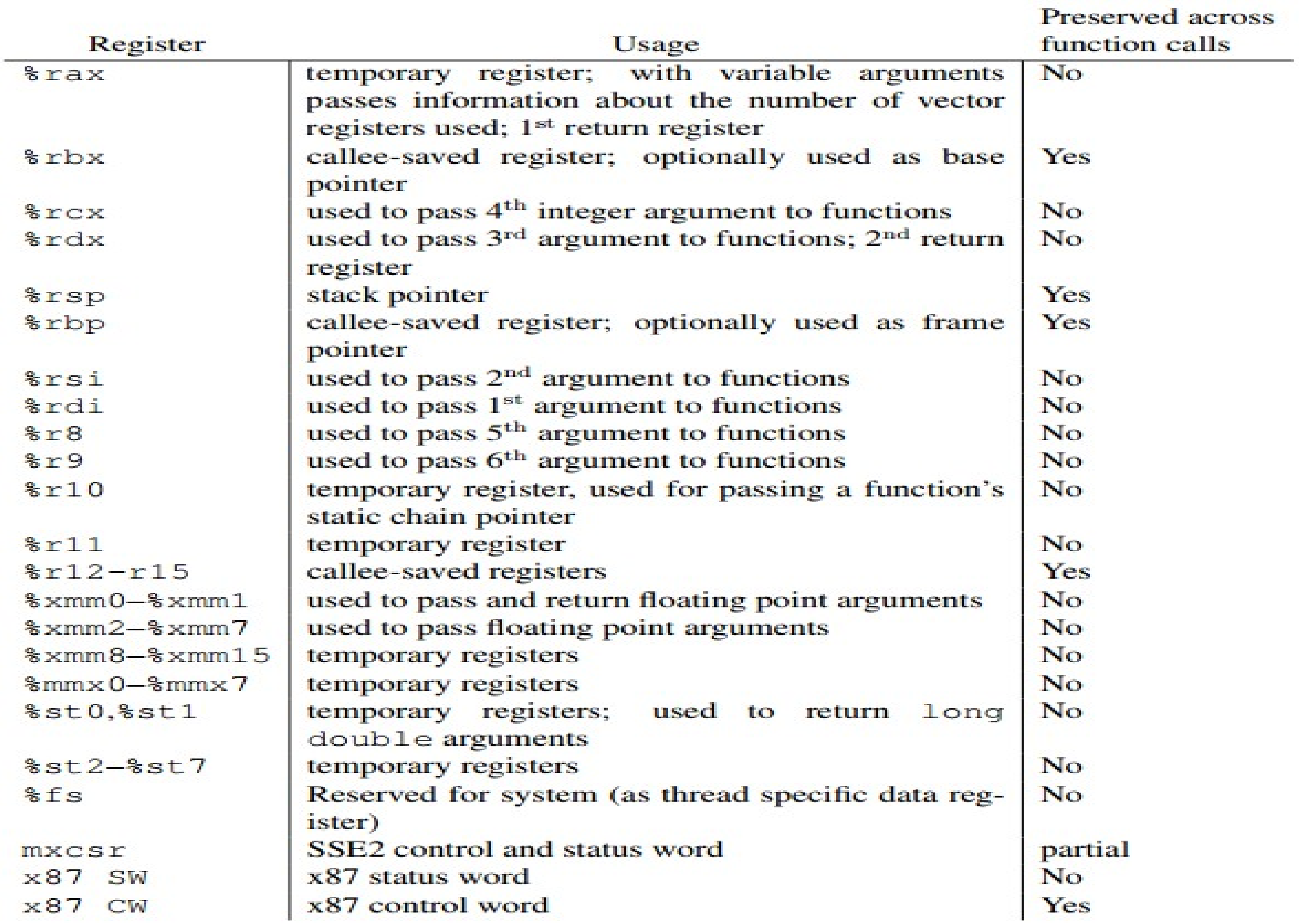
- The 64-bit calling conventions are a bit more detailed, and they are explained fully in the AMD64 ABI Reference:

- From left to right, pass as many parameters as will fit in registers. The order in which registers are allocated, are:
  - The first six integer arguments are passed in registers.
  - The seventh and any additional arguments are passed on the stack in reverse order.
  - For integers and pointers, rdi, rsi, rdx, rcx, r8, r9.
  - For floating-point (float, double), xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7
  - Additional parameters are pushed on the stack, right to left, and are removed by the caller after the call.

X86 Calling Convention

- The only registers that the called function is required to preserve (the calle-save registers) are: rbp, rbx, r12, r13, r14, r15. All others are free to be changed by the called function.
- Integers are returned in rax or rdx:rax, and floating point values are returned in xmm0 or xmm1:xmm0.

X86 Calling Convention



Register	Usage	Preserved across function calls
`%rax`	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
`%rbx`	callee-saved register; optionally used as base pointer	Yes
`%rcx`	used to pass 4th integer argument to functions	No
`%rdx`	used to pass 3rd argument to functions; 2nd return register	No
`%rsp`	stack pointer	Yes
`%rbp`	callee-saved register; optionally used as frame pointer	Yes
`%rsi`	used to pass 2nd argument to functions	No
`%rdi`	used to pass 1st argument to functions	No
`%r8`	used to pass 5th argument to functions	No
`%r9`	used to pass 6th argument to functions	No
`%r10`	temporary register, used for passing a function's static chain pointer	No
`%r11`	temporary register	No
`%r12–r15`	callee-saved registers	Yes
`%xmm0–%xmm1`	used to pass and return floating point arguments	No
`%xmm2–%xmm7`	used to pass floating point arguments	No
`%xmm8–%xmm15`	temporary registers	No
`%mmx0–%mmx7`	temporary registers	No
`%st0,%st1`	temporary registers; used to return `long double` arguments	No
`%st2–%st7`	temporary registers	No
`%fs`	Reserved for system (as thread specific data register)	No
`mxcsr`	SSE2 control and status word	partial
`x87 SW`	x87 status word	No
`x87 CW`	x87 control word	Yes

X86 Instruction Syntax

Intel syntax:

op dst, src (Intel manuals!)

AT&T (gcc/gas) syntax:

op src, dst (labs, xv6)

Example:

movl %eax, %edx	➔	edx = eax;	register mode
movl $0x123, %edx	➔	edx = 0x123;	immediate
movl 0x123, %edx	➔	edx = *(int32_t*)0x123;	direct
movl (%ebx), %edx	➔	edx = *(int32_t*)ebx;	indirect

X86 Addressing Modes

We will primarily use 3 simple addressing modes:

❑**Register mode:** the operand is the name of a register. For example

```
movq %rsp, %rax
```

which puts the stack pointer into register %rax.

❑**Indirect mode:** the operand is the value stored in a memory location, which is specified by an offset from the value in a register. For example

```
movl %eax, 8(%rsp)
```

This moves the 32 bits stored in %eax (the lower half of %rax) to the stack, 8 bytes below the top of the stack (the stack "grows" towards smaller addresses).

X86 Addressing Modes

- **Immediate mode**: The operand is a number, which is preceded by a $:

```
movl $23, %eax
```

puts the literal number 23 into %eax.

- **Direct mode:** The operand is specified by a symbol in the program. For example

```
call f
```

Here f must be a label at the start of a function.

For another example

```
jmp .L0
```

is an unconditional branch to label .L0

X86 Instruction Classes

data movement:

MOV, PUSH, POP, ...

arithmetic:

TEST, SHL, ADD, AND, ...

i/o:

IN, OUT, ...

control:

JMP, JZ, JNZ, CALL, RET

string:

REP MOVSB, ...

system:

IRET, INT

https://www.felixcloutier.com/x86/

X86 Instruction Classes

Data Instructions

MOVL, MOVQ: move 32 or 64 bits from the source to the destination. For example

movl $23, %eax puts 23 into the 32-bit register %eax

movq %rsp, %rax puts the stack pointer into %rax

LEAQ loads the "effective address" of the source into the destination. The data is an address so you want the Q-mode.For example

```
leaq 8(%rsp), %rax
```

puts the address 8 bytes below the top of the stack into %rax.You could just as easily do this as

```
movq %rsp, %rax
addq 8, %rax
```

X86 Instruction Classes

CLRL, CLRQ loads 0 into the 32-bit or 64-bit destination

```
clrl %eax puts 0 into %eax
clrq 0(%rsp) changes the top of the stack to 0
```

PUSH decrements $rsp by 8 bytes and puts the source data at this location.

```
push $23 pushes the number 23 onto the stack
```

Note that if you want to have 3 64-bit local variables allocated on the stack, you could do this with

```
push $0
push $0
push $0
```

which initializes them to 0, or with

```
subq $24, %rsp
```

which makes room for them on the stack but doesn't initialize them.

X86 Instruction Classes

POP puts the 8 bytes at the top of the stack into the destination and increments %rsp by 8.

pop %rax pops the stack into %rax

Before you leave a function call you will need to pop the local environment off the stack. This is usually more easily accomplished by incrementing %rsp than by issuing the right number of pop instructions.

X86 Instruction Classes

Calls

call F pushes the instruction pointer (the address of the next instruction; this is in register %RIP) and branches to label F.

return pops the stack into the instruction pointer register %rip.

Note that both call and return are minimalist instructions. Call does nothing with arguments or with local variables. We will have a set of conventions for calling that I call the "runtime environment". These will say where arguments go, who puts them there and who gets rid of them. Note also that return assumes that anything the called function has put on the stack has been popped off, so that the return address is at the top of the stack when you are ready to return.

X86 Instruction Classes

Arithmetic

ADDL, ADDQ adds 32-bit or 64-bit data

SUBL, SUBQ subtracts 32-bit or 64-bit data.

Both of these work by adding or subtracting the source and destination values, leaving the result in the destination:

addl $8, %eax increments %eax by 8

IMUL does 32-bit multiplication in ways you expect:

imul 0(%rsp), %eax multiplies the value in %eax by the value at the top of the stack.

X86 Instruction Classes

Branches

jmp label is an unconditional jump to the label There are 7 conditional branches: je, jne, jl, jle, jg, jge, jz. These all act on the value of the condition codes in the processor. Most arithmetic instructions set the condition codes, so if you are sure that the previous instruction was an arithmetic operation you might follow it immediately with a conditional branch. In most situations it is safer to do a comparison with CMP and then follow it with the conditional branch. The sequence

```
cmpl $8, %eax
jle L2
```

will branch to label L2 if the value in register %eax is less than or equal to 8. Note that this is backwards of the way most people would read this.

GCC Assembler Directives

All assembler directives have names that begin with a period (`.'). The rest of the name is letters, usually in lower case.

.ascii "string"...

.ascii expects zero or more string literals (see section Strings) separated by commas. It assembles each string (with no automatic trailing zero byte) into consecutive addresses.

.asciz "string"...

.asciz is just like .ascii, but each string is followed by a zero byte. The "z" in `.asciz' stands for "zero".

.data subsection

.data tells as to assemble the following statements onto the end of the data subsection numbered subsection (which is an absolute expression). If subsection is omitted, it defaults to zero.

GCC Assembler Directives

.global symbol, .globl symbol

.global makes the symbol visible to ld. If you define symbol in your partial program, its value is made available to other partial programs that are linked with it. Otherwise, symbol takes its attributes from a symbol of the same name from another file linked into the same program.

.text subsection

Tells as to assemble the following statements onto the end of the text subsection numbered subsection, which is an absolute expression. If subsection is omitted, subsection number zero is used.

GCC Assembler Directives

https://ftp.gnu.org/old-gnu/Manuals/gas-2.9.1/html_chapter/as_7.html

C program to assembly

- CFI: call frame information directives. These directives are used in the assembly to direct exception handling.

C program to assembly

```
if(a<100) {
        a=a+1;
}else{
       a=a+2;
}
return 0;
```

C program to assembly

```
ebx=a


    cmp $100,%ebx
    jge large
    add $1,%ebx
    jmp exit
large:
     add $2,%ebx
exit:
    mov $0,%eax
```

Inline Assembly

```
asm [ volatile ] ( /*asm statements*/
                   [: /* outputs - comma separated list of constraint name pairs */
                   [: /* inputs - comma separated list of constraint name pairs*/
                   [: /* clobbered space - registers, memory*/
                    ]]]);
```

asm statements - enclosed with quotes, at&t syntax, separated by new lines outputs & inputs - constraint-name pairs "constraint" (name), separated by commascregisters-modified - names separated by commas Constraints are

g - let the compiler decide which register to use for the variable

r - load into any avaliable register

a - load into the eax register

b - load into the ebx register

Inline Assembly

c - load into the ecx register

d - load into the edx register

f - load into the floating point register

D - load into the edi register

S - load into the esi register

The outputs and inputs are referenced by numbers beginning with %0 inside asm statements.

Inline Assembly

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

int f( int );

int main (void)
{
  int x;
  asm volatile("movl $3,%0" :"=g"(x): :"memory"); // x = 3;
  printf("%d  ->  %d\n",x,f(x));
}

/*END   Main    */
```

Inline Assembly

```
int f( int x )
{
  asm volatile("movl %0,%%eax
               imull $3, %%eax
               addl $4,%%eax"
         :
         :"a" (x)
         : "eax", "memory"
        ); //return (3*x + 4);
}
```

Thank you