Compilers (CS31003)

Autumn 2019

Hardware

Processor: Intel(R) Core(TM) i5-4570 CPU
3.2 GHz (Max Turbo Freq: 3.6 GHz)
4 (# of cores)
4 (# of threads)

Memory: 6 MB Smart Cache
4 GB (main memory; max 32 GB)

Software

OS: GNU/Linux, 64-bit, x86_64

Software: GCC, Lex/Flex and Yacc/Bison

Language: C++

System

Hardware system information:

$ uname –a

Linux Pralay 2.6.32-504.el6.x86_64 #1 SMP Wed Oct 15 04:27:16 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux

CPU information:

$ cat /proc/cpuinfo

processor : 0
model name : Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
cache size : 6144 KB
core id : 0
cpu cores : 4
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual

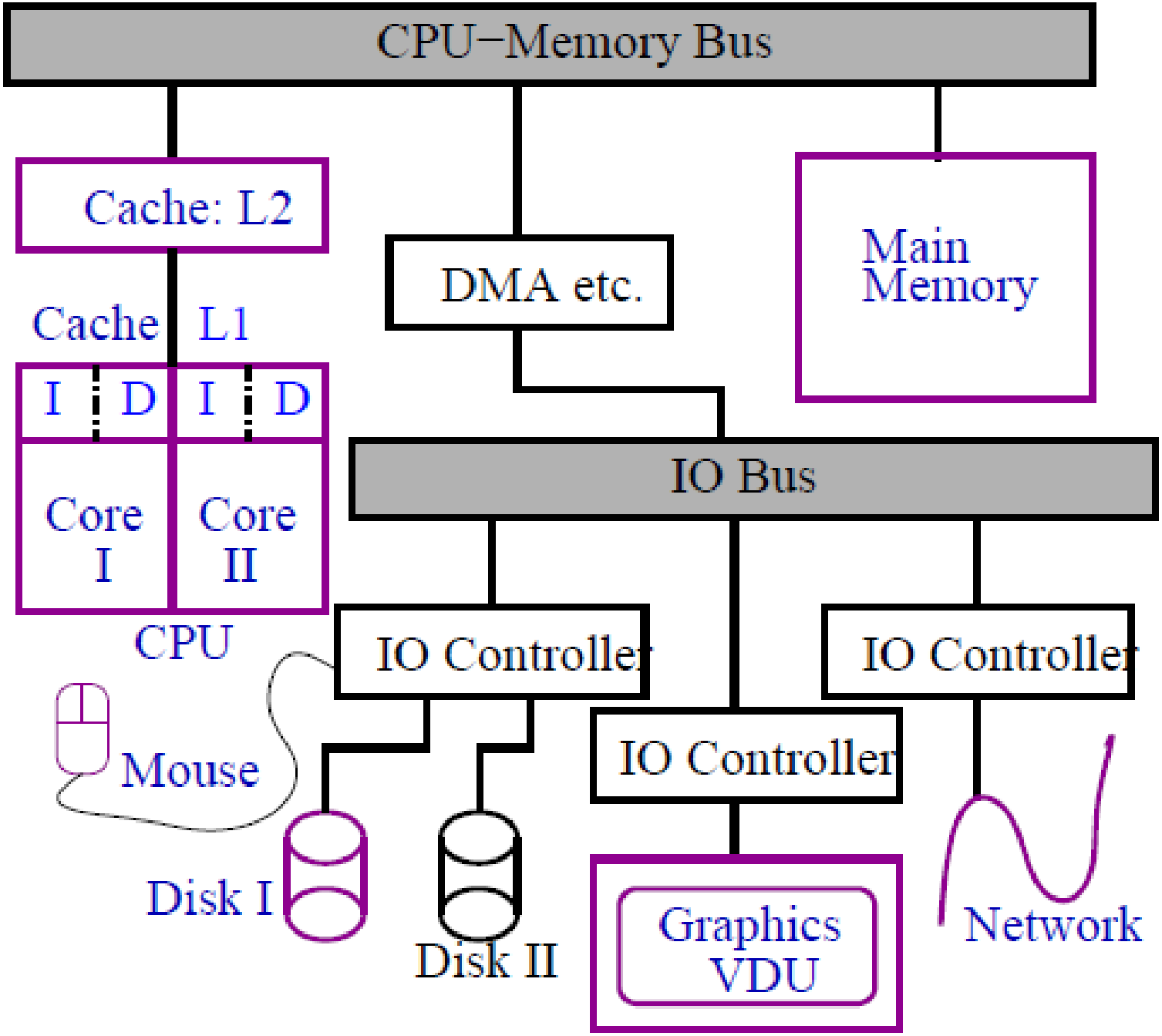
System

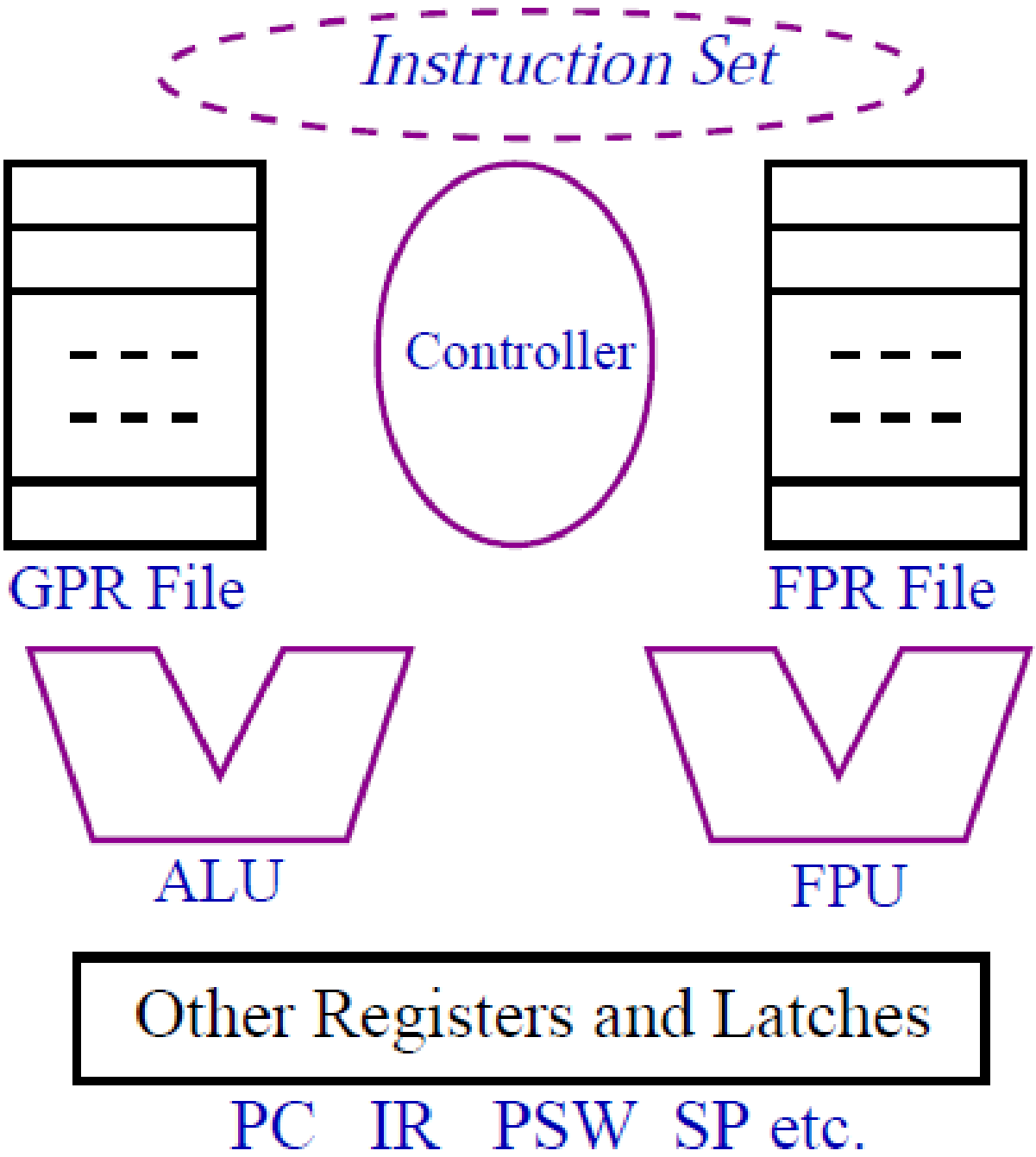
Main Memory Address

Address: 36 bits physical, 48 bits virtual/logical

The width of any X86_64 address register is 64 bit. But the most significant 17 bits are either all 1's or all 0's. So the logical address space of any process is 48-bits.

Depending on the model of the CPU, 48-bit logical address is translated to 36 to 40 bits of physical memory (main) address.


CPU–Memory Bus
Cache: L2
Cache L1
I D I D
Core I
Core II
CPU
DMA etc.
Main Memory
IO Bus
IO Controller
IO Controller
IO Controller
Mouse
Disk I
Disk II
Graphics VDU
Network


Instruction Set
Controller
GPR File
FPR File
ALU
FPU
Other Registers and Latches
PC IR PSW SP etc.

Intel 64-bit Registers

GPRs: 64-bit integer registers (16)

rax, *rbx*, *rcx*, *rdx*, *rsp*, *rbp*, *rsi*, *rdi*, *r*8, .. , *r*15

FPRs: 80-bit floating point registers (8)

*r*0, .. , *r*7

MMXs: 64-bit SIMD registers (8)

*mm*0, .. , *mm*7

XMMs: 128-bit SSE registers (16)

*xmm*0, .., *xmm*15

Streaming SIMD

Special Registers

64-bit *rflags*, 64-bit *rip* (PC), segment registers, control registers, debug registers, etc.

Register Usage Convention

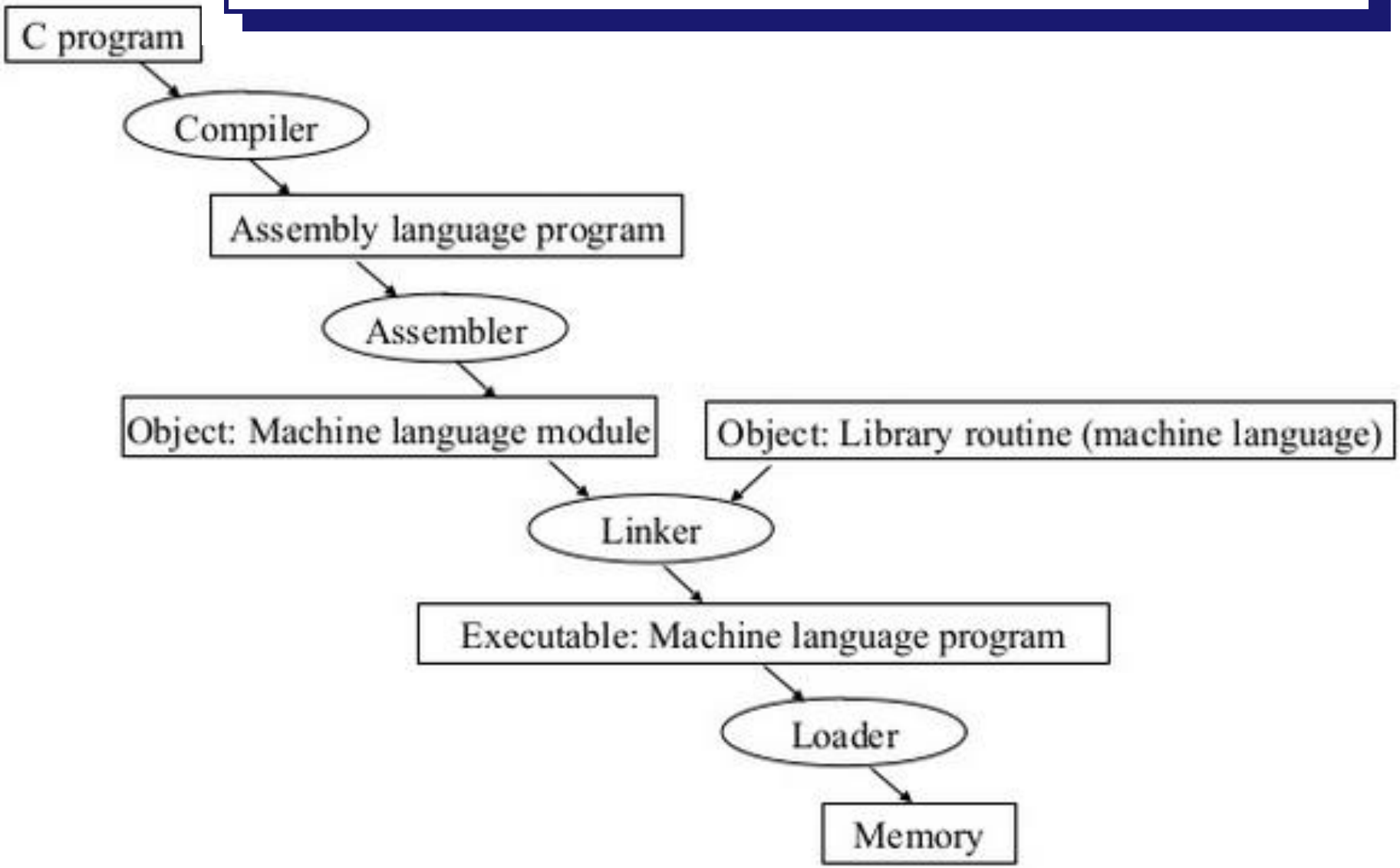
GPR (64-bit)	Usage Convention
rax	Return value from a function
rbx	Callee saved
rcx	4th argument to a function
rdx	3rd argument to a function Return value from a function
rsi	2nd argument to a function
rdi	1st argument to a function
rbp	Callee saved

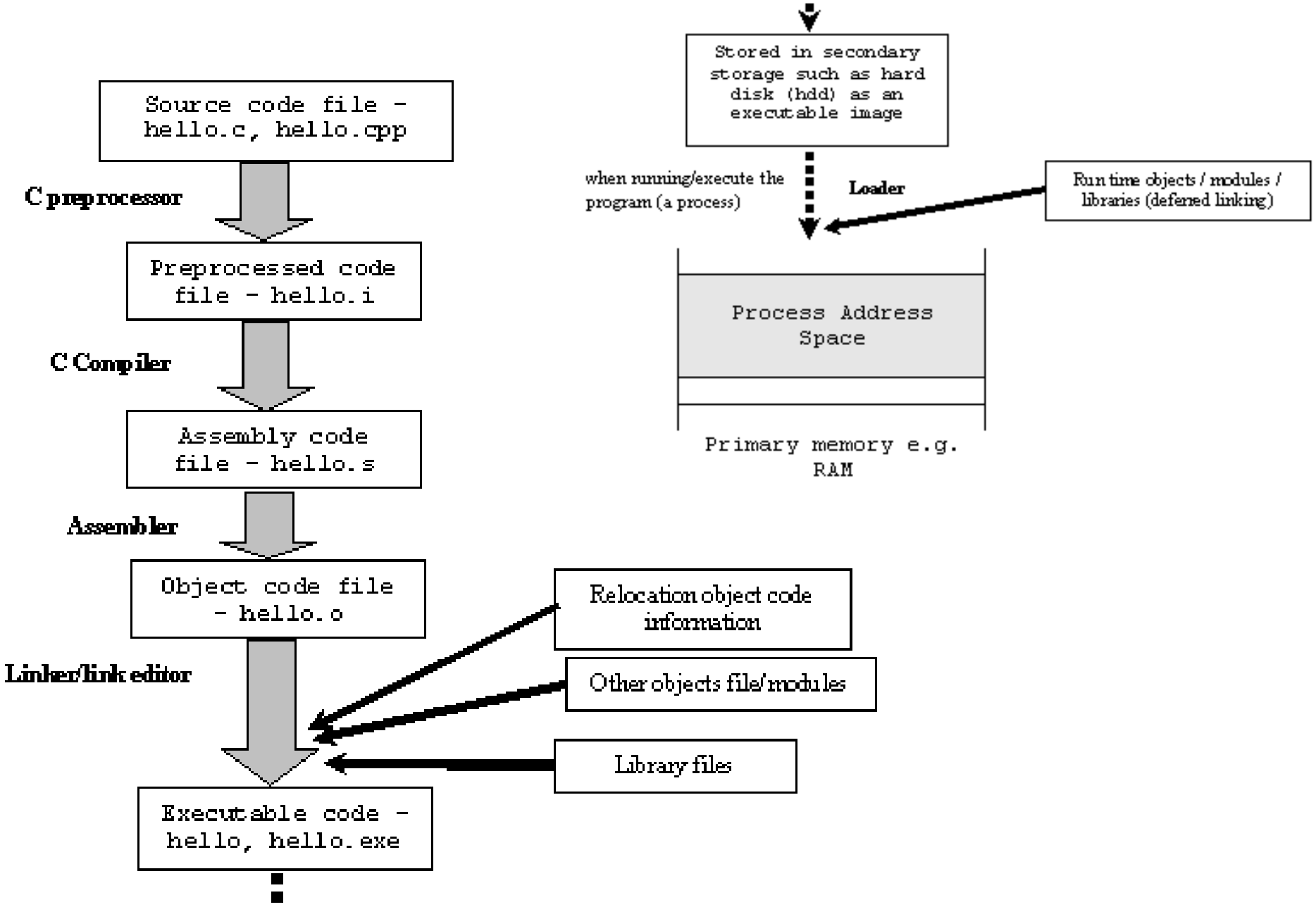
Register Usage Convention

GPR (64-bit)	Usage Convention
rsp	Hardware stack pointer
r8	5th argument to a function
r9	6th argument to a function
r10	Callee saved
r11	Reserved for linker
r12	Reserved for C
r13	Callee saved
r14	Callee saved
r15	Callee saved

Function return address is at the top of the stack.

CPP → Compiler → Assembler → Linker




Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Linker/link editor
Executable code - hello, hello.exe
Relocation object code information
Other objects file/modules
Library files
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

A simple Assembly Program

```
some_function:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $20, %esp
        movl    8(%ebp), %ebx
        movl    12(%ebp), %ecx
        movl    $0, %edx
        testl   %ecx, %ecx
        jle     .L152
        movl    $0, %eax
        movl    $0, %edx
.L153:
        addl    (%ebx,%eax,4), %edx
        addl    $1, %eax
        cmpl    %eax, %ecx
        jne     .L153
.L152:
        movl    %edx, 4(%esp)
        movl    $.LC14, (%esp)
        call    printf
        addl    $20, %esp
        popl    %ebx
        popl    %ebp
        ret
```

```
void some_function(int a[], int n) {
  int i, sum = 0;
  for (i = 0; i < n; i++) {
    sum += a[i];
  }
  printf("The sum is %d\n", sum);
}
```

Source Code

```
#include <stdio.h>

int main()
{
        int loop,terms;
        double pi,sign;

        printf("Enter the number of terms: ");
        scanf("%d",&terms);
        pi=3.0;
        sign=1.0;
        for(loop=1;loop<=terms;loop++) {
                pi+=sign*(4.0/((2.0*loop)*(2.0*loop+1)*(2.0*loop+2)));
                sign*=-1.0;
        }

        printf("\nValue of PI: %12.10lf\n",pi);
        return 0;
}
```

Compilation

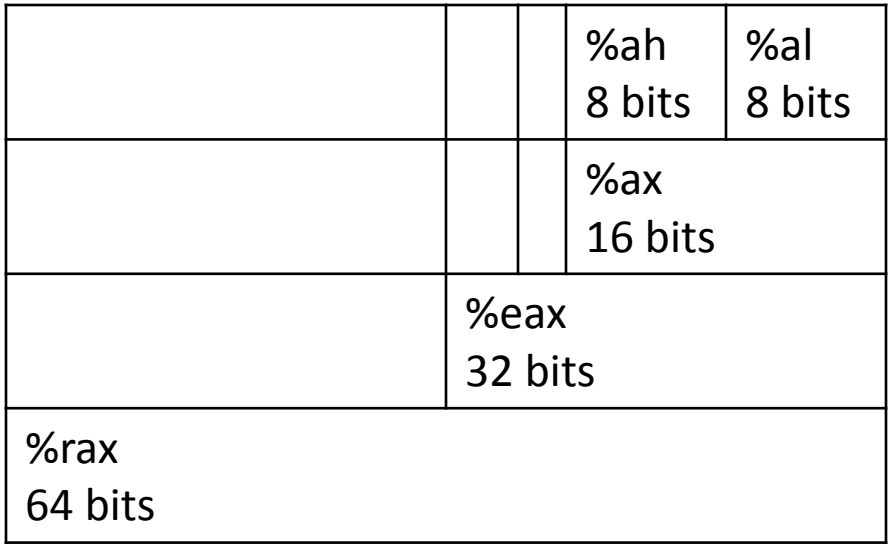
$ cc -Wall -S computePI.c → computePI.s

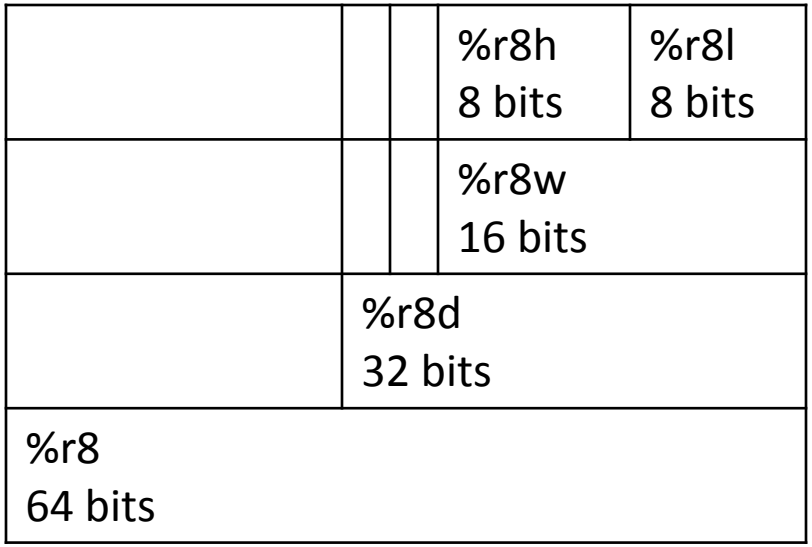
$ cc -Wall -c computePI.c → computePI.o

$ cc -Wall computePI.c → a.out

Conventions

Suffix	Name	Size
B	BYTE	1 byte (8 bits)
W	WORD	2 bytes (16 bits)
L	LONG	4 bytes (32 bits)
Q	QUADWORD	8 bytes (64 bits)





Conventions

Mode	Example
Global Symbol	MOVQ x, %rax
Immediate	MOVQ $56, %rax
Register	MOVQ %rbx, %rax
Indirect	MOVQ (%rsp), %rax
Base-Relative	MOVQ -8(%rbp), %rax
Offset-Scaled-Base-Relative	MOVQ -16(%rbx,%rcx,8), %rax

Assembly Code

```
        .file   "computePI.c“                   # source file name
        .section .rodata                        # read-only data section
        .align 8                                # align with 8-byte boundary
.LC0:                                           # Label of f-string-1st printf
        .string "Enter the number of terms: "
.LC1:                                           # Label of f-string scanf
        .string "%d"
.LC7:                                           # Label of f-string - 2nd printf
        .string "\nValue of PI: %12.10lf\n"
        .text                                   # Code starts
.globl main                             # main is a global name
        .type   main, @function                 # main is a function
main:                                           # main: starts
.LFB0:
        .cfi_startproc                          # Call Frame Information
        pushq   %rbp                            # Save old base pointer
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp              # rbp <-- rsp set new stack base pointer
        .cfi_def_cfa_register 6
```

Assembly Code

```
        subq    $32, %rsp               # Create space for local array and variables
        movl    $.LC0, %eax             # eax <-- starting of the format string, 1st param
        movq    %rax, %rdi              # rdi <-- rax
        movl    $0, %eax                # eax <-- 0 (?)
        call    printf                  # Call printf
        movl    $.LC1, %eax              # eax <-- starting of the format string
        leaq    -24(%rbp), %rdx         # rdx <-- (rbp - 24) (&terms)
        movq    %rdx, %rsi
        movq    %rax, %rdi
        movl    $0, %eax                # eax <-- 0 (?)
        call    __isoc99_scanf          # call scanf, return value is in eax
        movabsq $4613937818241073152, %rax # Move quad word
        movq    %rax, -16(%rbp)                         # put value rax to (rbp-16)
        movabsq $4607182418800017408, %rax # Move quad word
        movq    %rax, -8(%rbp)                          # put value rax to (rbp-8)
        movl    $1, -20(%rbp)                           # assign 1 to (rbp-20)
        jmp     .L2                                     # jump to L2
.L3:
        cvtsi2sd    -20(%rbp), %xmm0            # Convert Doubleword Integer to Scalar
                                                # Double- Precision Floating-Point Value
        movapd  %xmm0, %xmm1                    # Copy %xmm0 to %xmm1
        addsd   %xmm0, %xmm1                    # floating point add
```

Assembly Code

```
cvtsi2sd   -20(%rbp), %xmm0
addsd      %xmm0, %xmm0
movsd      .LC3(%rip), %xmm2
addsd      %xmm2, %xmm0
mulsd      %xmm0, %xmm1
cvtsi2sd   -20(%rbp), %xmm0
addsd      %xmm0, %xmm0
movsd      .LC4(%rip), %xmm2
addsd      %xmm2, %xmm0
mulsd      %xmm1, %xmm0
movsd      .LC5(%rip), %xmm1
movapd     %xmm1, %xmm2
divsd      %xmm0, %xmm2
movapd     %xmm2, %xmm0
mulsd      -8(%rbp), %xmm0
movsd      -16(%rbp), %xmm1
addsd      %xmm1, %xmm0
movsd      %xmm0, -16(%rbp)
movsd      -8(%rbp), %xmm1
movsd      .LC6(%rip), %xmm0
xorpd      %xmm1, %xmm0
```

Assembly Code

```
        movsd   %xmm0, -8(%rbp)
        addl    $1, -20(%rbp)
.L2:
        movl    -24(%rbp), %eax
        cmpl    %eax, -20(%rbp)
        jle     .L3
        movl    $.LC7, %eax
        movsd   -16(%rbp), %xmm0
        movq    %rax, %rdi
        movl    $1, %eax
        call    printf
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section    .rodata
        .align 8
```

Assembly Code

```
.LC3:
        .long   0
                # 0000 0000 0000 0000 0000 0000 0000 0000
        .long   1072693248
        .align 8
.LC4:
        .long   0
        .long   1073741824
        .align 8
.LC5:
        .long   0
        .long   1074790400
        .align 16
.LC6:
        .long   0
        .long   -2147483648
        .long   0
        .long   0
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-11)"
        .section .note.GNU-stack,"",@progbits
```

Source Code

```
// sqrtNewton.c
#include <stdio.h>
#include <math.h>
int main() // sqrtNewton.c
{
    double k, root, oldR ;

    printf("Enter a +ve number: ") ;
    scanf("%lf", &k) ;

    root = k/2;
    do {
        oldR = root ;
        root = (root*root + k)/(2.0*root) ;
    } while(fabs((oldR - root)/root)*100.0 > 0.01) ;
    printf("sqrt(%f) = %f\n", k, root) ;

    return 0;
}
```

Compiling a C program

```
#include <stdio.h>
#define MAXNO 100
void selectionSort(int [], int);
int main() // main.c
{
    int no = 0, i ;
    int data[MAXNO] ;

    printf("Enter the data, terminate with Ctrl+D\n") ;
    while(scanf("%d", &data[no]) != EOF) ++no;
    selectionSort(data, no) ;
    printf("Data in sorted Order are: ") ;
    for(i = 0; i < no; ++i) printf("%d ", data[i]);
    putchar('\n') ;
    return 0 ;
}
```

Compiling a C program

```
#define EXCH(X,Y,Z) ((Z)=(X), (X)=(Y), (Y)=(Z))
void selectionSort(int data[], int nod) {
     int i ;

     for(i = 0; i < nod - 1; ++i) {
          int max, j, temp;

          temp = data[i] ;
          max = i ;
          for(j = i+1; j < nod; ++j)
               if(data[j] > temp) {
                    temp = data[j] ;
                    max = j ;
               }
           EXCH(data[i], data[max], temp);
      }
 }
```

Compilation

$ cc -Wall -S main.c → main.s

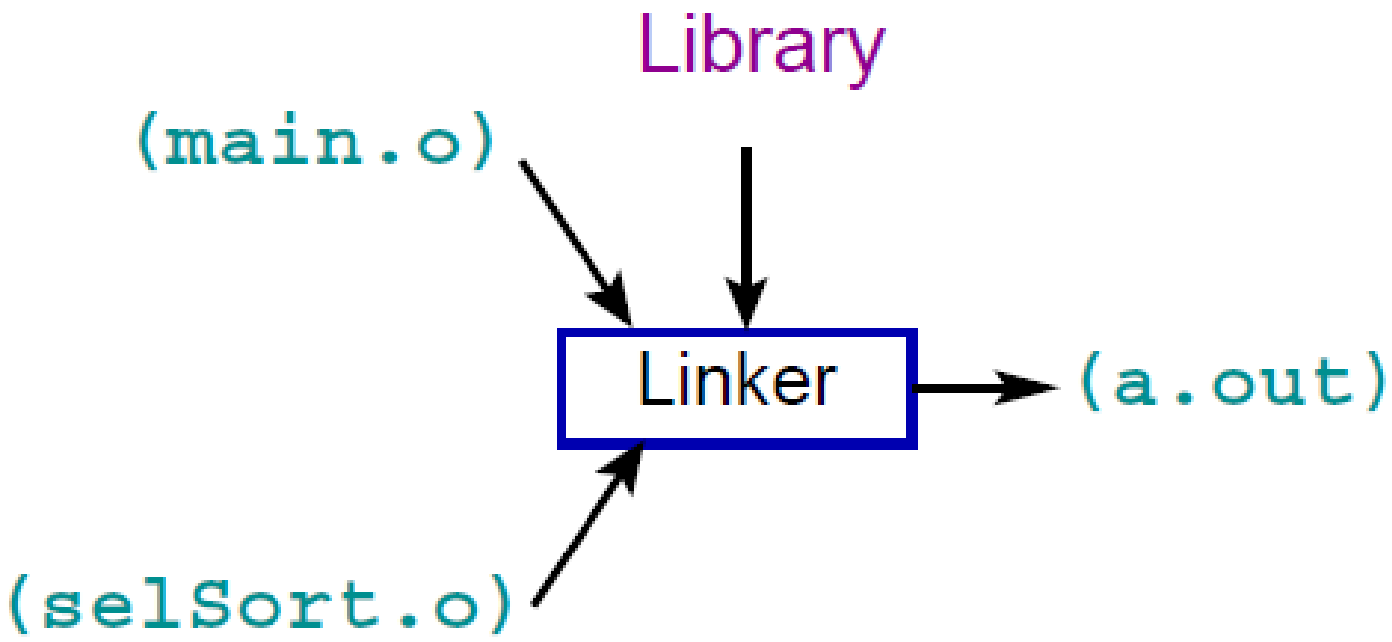
$ cc -Wall -c main.c → main.o

$ cc -Wall -S selSort.c → selSort.s

$ cc -Wall -c selSort.c → selSort.o

$ cc main.o selSort.o → a.out

Compilation and Linking



File Types

$ file main.c selSort.c
main.c: ASCII English text
selSort.c: ASCII text

$ file main.s selSort.s
main.s: ASCII English text
selSort.s: ASCII assembler program text

$ file main.o selSort.o
main.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
selSort.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped

$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, not stripped