

Compilers Laboratory (CS39003)

Autumn 2022

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 Jul 13 14:27:16 UTC 2022 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

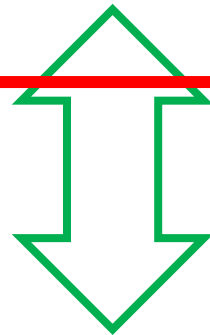
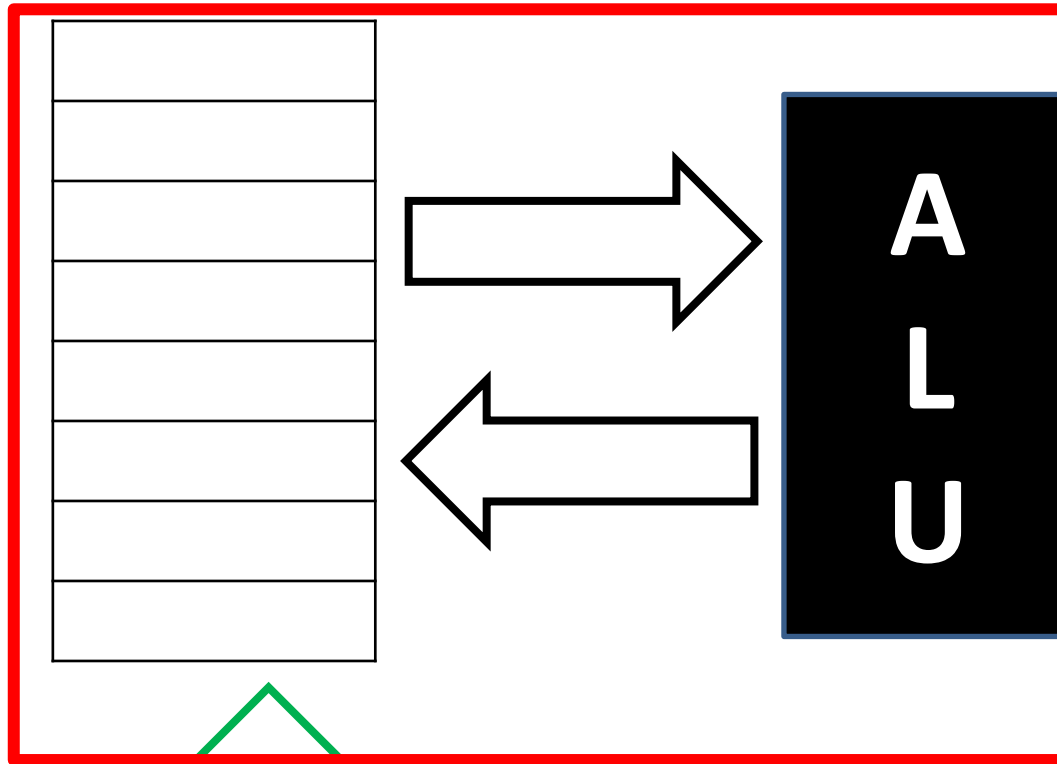
The width of any X86_64 address register is 64 bit. But the most significant 17 bits are either all 1's or all 0's. So the logical address space of any process is 48-bits.

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

CPU Chip

Single core

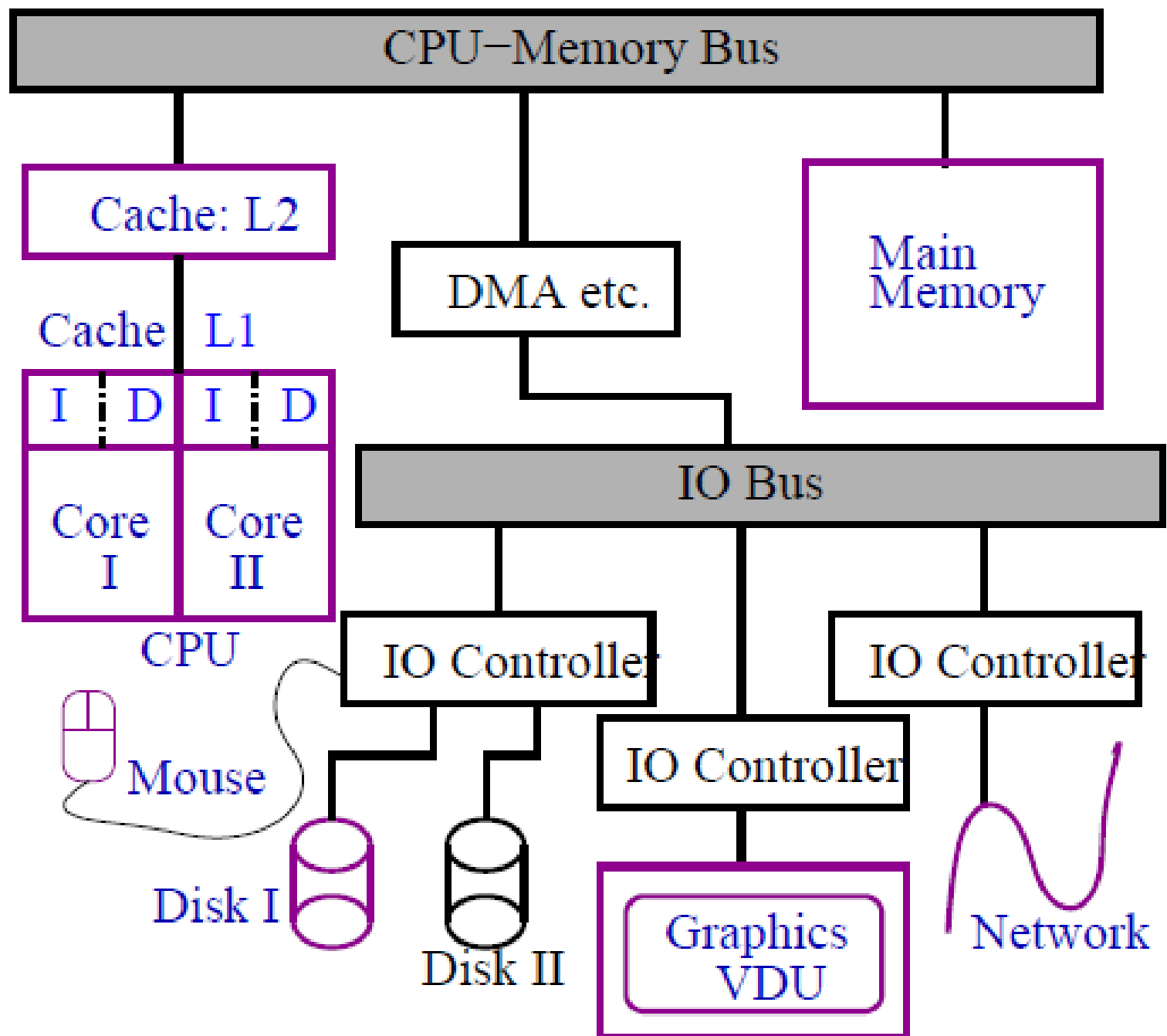
Register file

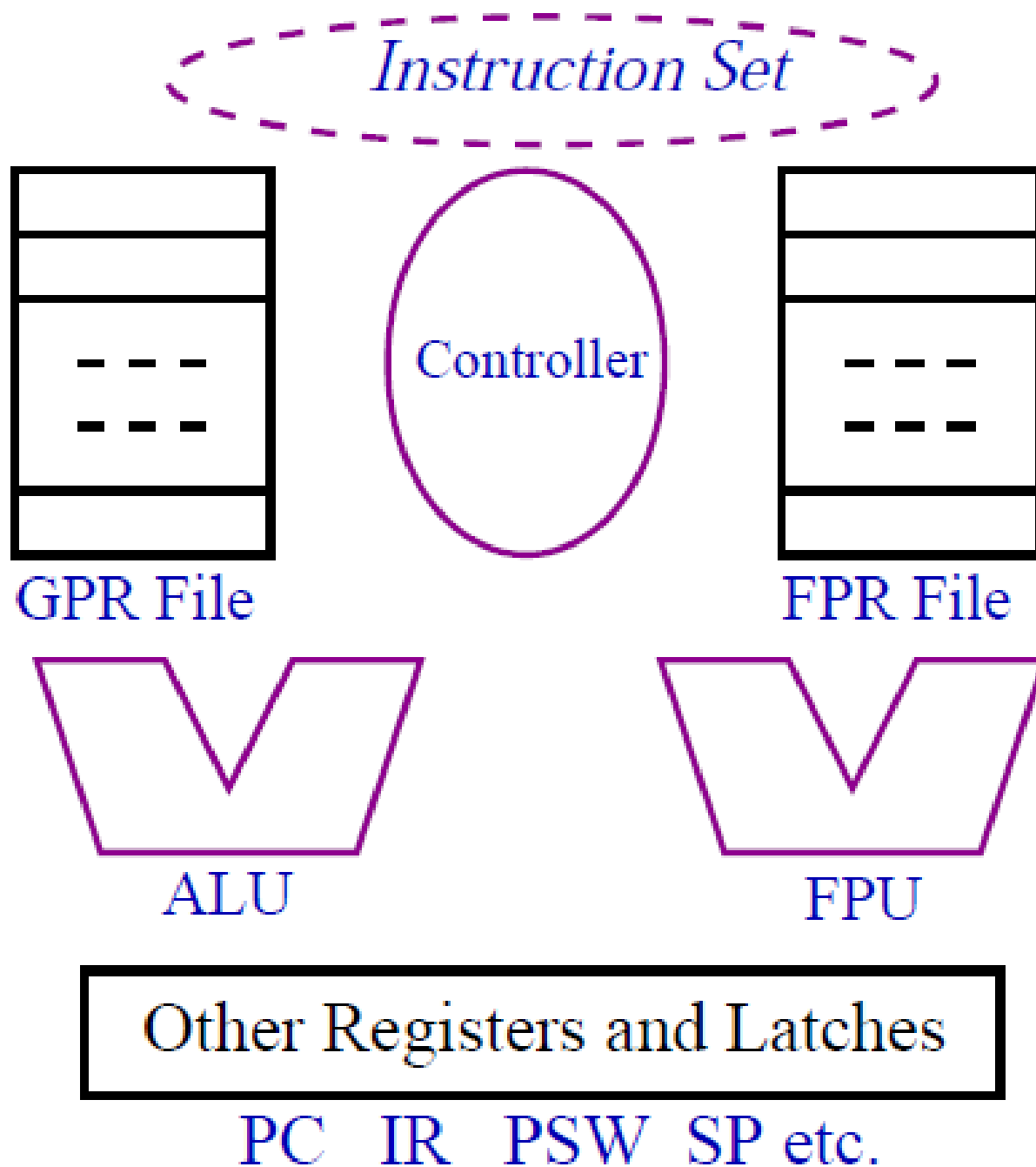


Bus interface

System bus







Intel 64-bit Registers

GPRs: 64-bit integer registers (16)

rax, rbx, rcx, rdx, rsp, rbp, rsi, rdi, r8, .. , r15

FPRs: 80-bit floating point registers (8)

r0, .. , r7

MMXs: 64-bit SIMD registers (8)


mm0, .. , mm7

XMMs: 128-bit SSE registers (16)

xmm0, .. , xmm15

**Streaming
SIMD**

Special Registers

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

Register Usage Convention

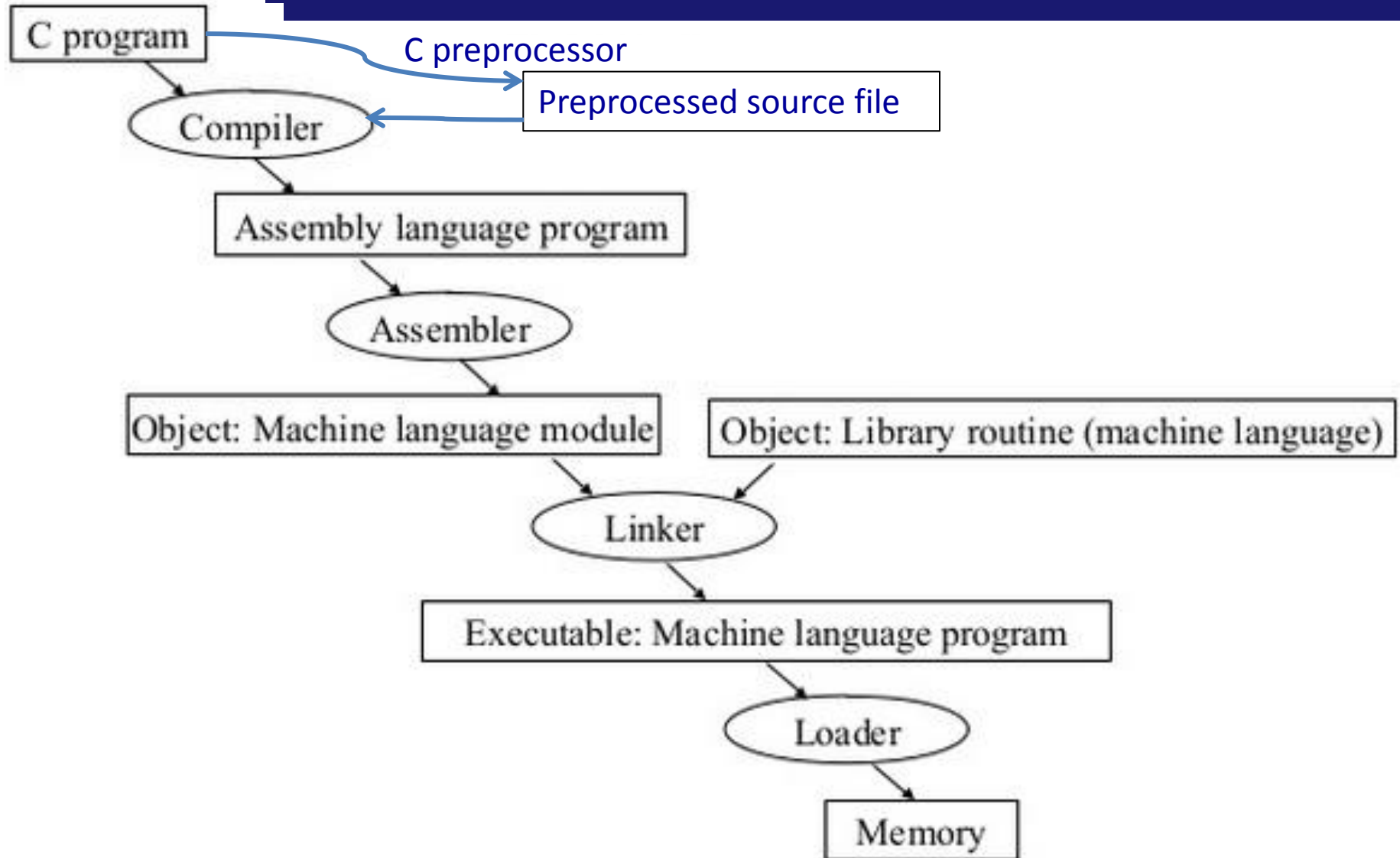
GPR (64-bit)	Usage Convention
<i>rax</i>	Return value from a function
<i>rbx</i>	Callee saved
<i>rcx</i>	4 th argument to a function
<i>rdx</i>	3 rd argument to a function Return value from a function
<i>rsi</i>	2 nd argument to a function
<i>rdi</i>	1 st argument to a function
<i>rbp</i>	Callee saved

Register Usage Convention

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

Function return address is at the top of the stack.

CPP → Compiler → Assembler → Linker



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 computePl.c → computePl.s

\$ cc -Wall -c computePl.c → computePl.o

\$ cc -Wall computePl.c → a.out

Conventions

Suffix

B

W

L

Q

Name

BYTE

WORD

LONG

QUADWORD

Size

1 byte (8 bits)

2 bytes (16 bits)

4 bytes (32 bits)

8 bytes (64 bits)

			%ah 8 bits	%al 8 bits
			%ax 16 bits	
	%eax 32 bits			
%rax 64 bits				

			%r8h 8 bits	%r8l 8 bits
			%r8w 16 bits	
	%r8d 32 bits			
%r8 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      "computePl.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
.....
.....
.....
```

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
```

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

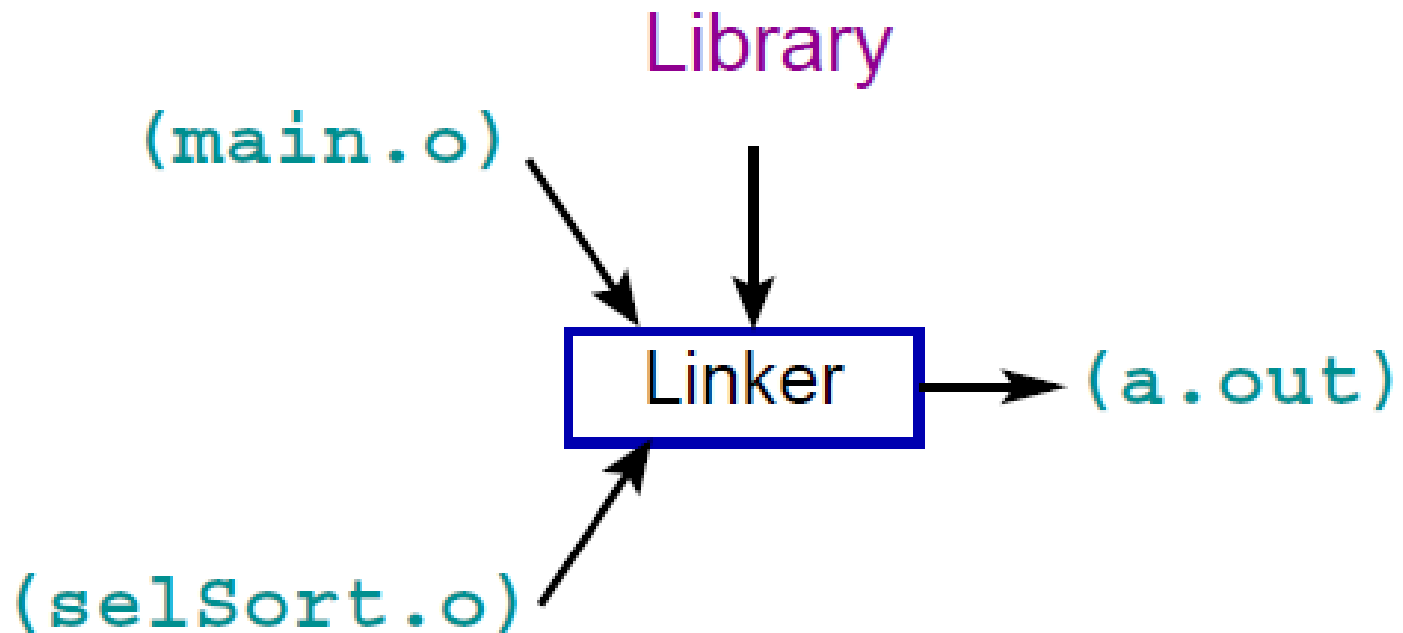
\$ cc -Wall -c main.c → main.o

\$ 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

Assembly Code

```
.file    "main.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 data, terminate with Ctrl+D"
.LC1:                # Label of f-string scanf
    .string "%d"
.LC2:                # Label of f-string - 2nd printf
    .string "Data in sorted Order are: "
.LC3:                # Label of f-string - 3rd printf
    .string "%d "
```

Assembly Code

```
.text                # Code starts
.globl main          # main is a global name
.type    main, @function # main is a function:
main:                # main: starts
    pushq    %rbp    # Save old base pointer
    movq     %rsp, %rbp # rbp <-- rsp set new
                        #   stack base pointer
    subq     $416, %rsp # Create space for local
                        # array and variables
#
    movl     $0, -8(%rbp) # no <-- 0
    movl     $.LC0, %edi  # edi <-- 1st parameter
                        #           of printf
    call     puts         # Calls puts for printf
```

Assembly Code

```
    jmp     .L2          # Goto the beginning of the
                          #   while loop
#
.L3:          # Increment code
    addl    $1, -8(%rbp) # M[rbp-8] <-- M[rbp-8] + 1
                          #   no <-- no + 1
.L2:          # label, body of the loop
    movl    -8(%rbp), %eax # eax <-- M[rbp-8] (no)
    cltq    # rax <-- eax (32-bits to
              #   sign ext. 64-bit)
    salq    $2, %rax      # rax <-- shift-arithmetic left
                          #   2-bit left (4*no)
    leaq    -416(%rbp), %rsi # rsi <-- (rbp - 416)
                          #   (&data)
```

Assembly Code

```
addq    %rax, %rsi    # rsi <-- rsi + rax
                        # (data+4*no = &data[nq])
                        # 2nd parameter
movl     $.LC1, %edi   # edi <-- starting of the
                        # format string,
                        # 1st parameter
movl     $0, %eax      # eax <-- 0 (?)
call     scanf          # call scanf, return
                        # value is in eax
cmpl     $-1, %eax     # if return value
                        # != -1 (EOF)
                        # (jne, jump not equal)
jne      .L3           # goto .L3 (loop)
                        # continue reading data ,
```


Assembly Code

```
movl    -8(%rbp), %esi # esi <-- no
                                # 2nd parameter
leaq    -416(%rbp), %rdi # rdi <-- data
                                # 1st parameter
call    selectionSort # call selectionSort
#
movl    $.LC2, %edi # edi <-- starting address
                                # of printf format string
                                # 1st parameter
movl    $0, %eax # eax <-- 0 (?)
call    printf # Call printf (2nd call)
movl    $0, -4(%rbp) # M[rbp-4] <-- 0,
                                # i <-- 0
```

Assembly Code

```
    jmp     .L5                # Goto loop test
#
.L6:
    movl    -4(%rbp), %eax     # eax <-- i
    cltq                    # rax <-- signExt(eax)
    movl    -416(%rbp,%rax,4), %esi # esi <--
                                #  Mem[(rbp - 416)+4*rax]
                                #  esi <-- data[i], 2nd par.
    movl    $.LC3, %edi        # edi <-- addr, of format str
                                #  1st parameter
    movl    $0, %eax           # eax <-- 0
    call    printf             # Call printf
    addl    $1, -4(%rbp)       # i <-- i+1
```

Assembly Code

```
.L5:                # Loop test
    movl    -4(%rbp), %eax # eax <-- i
    cmpl    -8(%rbp), %eax # if i < no
                                # (jl is jump less than)
    jl      .L6            # reEnter loop
#
    movl    $10, %edi      # edi <-- 10 (\n)
    call    putchar        # call putchar
    movl    $0, %eax       # eax <-- 0 (return 0)
    leave   # remove stack frame
    ret      # return
.LFE2:
    .size    main, .-main
    .section .eh_frame,"a",@progbits
```

Assembly Code

```
.file      "selSort.c" # file name
.text
.globl selectionSort    # selectionSort is global
.type      selectionSort, @function
selectionSort:
.LFB2:
    pushq    %rbp        # save old base pointer
.LCFI0:
    movq     %rsp, %rbp   # stack pointer is new
.LCFI1:                # base pointer
    movq     %rdi, -24(%rbp) # M[rbp - 24] <-- data
    movl     %esi, -28(%rbp) # M[rbp - 28] <-- nod
```

Assembly Code

```
    movl    $0, -16(%rbp)    # i <-- 0 (4-bytes)
                                #  init outer loop
    jmp     .L2              # goto .L2
                                #  test of outer loop
#
.L3:
    movl    -16(%rbp), %eax   # eax <-- i
    cltq                                # rax <-- eax
    salq    $2, %rax          # rax <-- 4*rax (4*i)
    addq    -24(%rbp), %rax   # rax <-- data + 4*i
    movl    (%rax), %eax      # eax <-- data[i]
    movl    %eax, -4(%rbp)    # temp <-- eax (data[i])
    movl    -16(%rbp), %eax   # eax <-- i
```

Assembly Code

```
    movl    %eax, -12(%rbp) # max <-- eax (i)
#
    movl    -16(%rbp), %eax # eax <-- i
    addl    $1, %eax        # eax <-- eax + 1 (i+1)
    movl    %eax, -8(%rbp)  # j <-- i+1
                                #  init inner loop
    jmp     .L4             # goto .L4
                                #  test of inner loop
#
.L5:
    movl    -8(%rbp), %eax  # eax <-- j
    cltq    # rax <-- eax
    salq    $2, %rax        # rax <-- 4*j
    addq    -24(%rbp), %rax # rax <-- data+4*j
```

Assembly Code

```
movl    (%rax), %eax    # eax <-- data[j]
cmpl    -4(%rbp), %eax  # if data[j] <= temp
jle     .L6             # goto .L6
                                #   inc. of inner loo
```

#

```
movl    -8(%rbp), %eax  # eax <-- j
cltq                                # rax <-- eax
salq    $2, %rax        # rax <-- 4*j
addq    -24(%rbp), %rax  # rax <-- data + 4*j
movl    (%rax), %eax    # eax <-- data[j]
movl    %eax, -4(%rbp)   # temp <-- data[j]
movl    -8(%rbp), %eax  # eax <-- j
movl    %eax, -12(%rbp)  # max <-- eax (j)
```

Assembly Code

```
.L6:                                # Inc. inner loop
    addl    $1, -8(%rbp)           # j <-- j+1
.L4:
    movl    -8(%rbp), %eax         # eax <-- j
    cmpl    -28(%rbp), %eax        # if j < nod
    jl      .L5                    # goto inner loop
#                                # Exchange starts
    movl    -16(%rbp), %eax        # eax <-- i
    cltq                                # rax <-- eax
    salq    $2, %rax               # rax <-- 4*i
    addq    -24(%rbp), %rax        # rax <-- data + 4*i
    movl    (%rax), %eax           # eax <-- data[i]
    movl    %eax, -4(%rbp)         # temp <-- data[i]
    movl    -16(%rbp), %eax        # eax <-- i
```


Assembly Code

```
cltq          # rax <-- eax
salq          $2, %rax      # rax <-- 4*i
movq          %rax, %rdx     # rdx <-- rax (4*i)
addq          -24(%rbp), %rdx # rdx <-- data + 4*i
movl          -12(%rbp), %eax # eax <-- max
cltq          # rax <-- eax
salq          $2, %rax      # rax <-- 4*max
addq          -24(%rbp), %rax # rax <-- data + 4*max
movl          (%rax), %eax   # eax <-- data[max]
movl          %eax, (%rdx)   # data[i] <-- data[max]
movl          -12(%rbp), %eax # eax <-- max
cltq          # rax <-- eax
salq          $2, %rax      # rax <-- 4*max
movq          %rax, %rdx     # rdx <-- rax (4*max)
```

Assembly Code

```
    addq    -24(%rbp), %rdx # rdx <-- data + 4*max
    movl    -4(%rbp), %eax  # eax <-- temp
    movl    %eax, (%rdx)    # data[max] <-- temp
#
    addl    $1, -16(%rbp)   # i <-- i+1
.L2:
    movl    -28(%rbp), %eax # eax <-- nod
    subl    $1, %eax        # eax <-- eax - 1
    cmpl    -16(%rbp), %eax # if (nod - 1) > i
    jg      .L3             # goto .L3
    leave                   # clear stack
    ret                    # return
.LFE2:
    .size   selectionSort, .-selectionSort
```

No Discussion on CFI Directives

`.cfi_startproc`

`.cfi_endproc`

`.cfi_def_cfa_offset offset`

`.cfi_offset 6, -16`

`.cfi_def_cfa_register`

CFI directives are used for the creation of `.eh_frame` to unwind stack frames for debugging and exception handling.