Use the command `$ lscpu`

## Machine

| | |
|---|---|
| Architecture: | x86_64 |
| CPU op-mode(s): | 32-bit, 64-bit |
| Byte Order: | Little Endian |
| CPU(s): | 4 |
| On-line CPU(s) list: | 0-3 |
| Thread(s) per core: | 1 |
| Core(s) per socket: | 4 |

| | |
|---|---|
| Socket(s): | 1 |
| NUMA node(s): | 1 |
| Vendor ID: | GenuineIntel |
| CPU family: | 6 |
| Model: | 158 |
| Model name: | Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz |
| Stepping: | 9 |

| | |
|---|---|
| CPU MHz: | 900.065 |
| CPU max MHz: | 3800.0000 |
| CPU min MHz: | 800.0000 |
| BogoMIPS: | 6799.81 |
| Virtualization: | VT-x |

| | |
|---|---|
| L1d cache: | 32K |
| L1i cache: | 32K |
| L2 cache: | 256K |
| L3 cache: | 6144K |
| NUMA node0 CPU(s): | 0-3 |

Use the command `$ grep MemTotal /proc/meminfo`

MemTotal: 8028156 kB = 8 GB

## Software

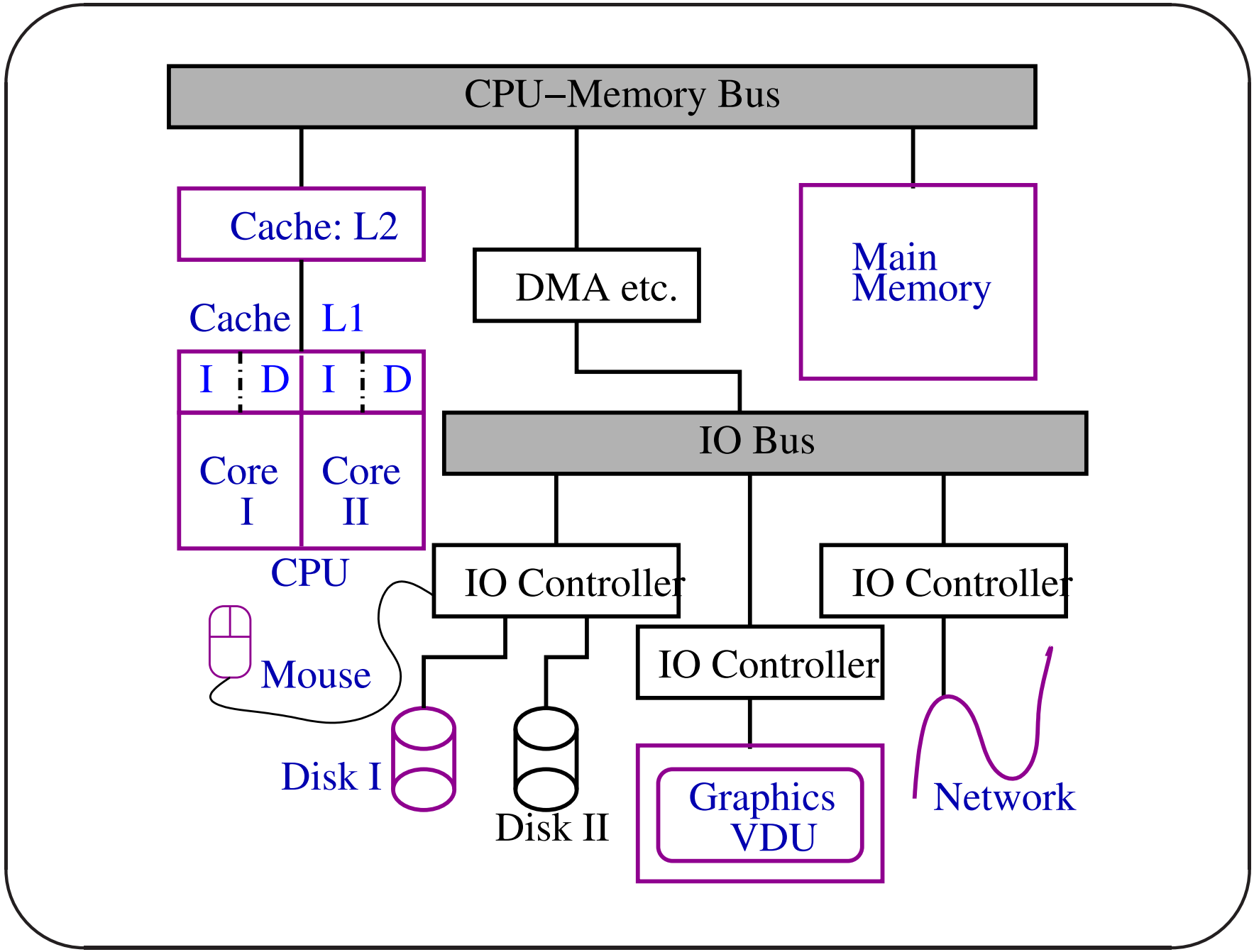
OS: GNU/Linux, 64-bit, x86_64

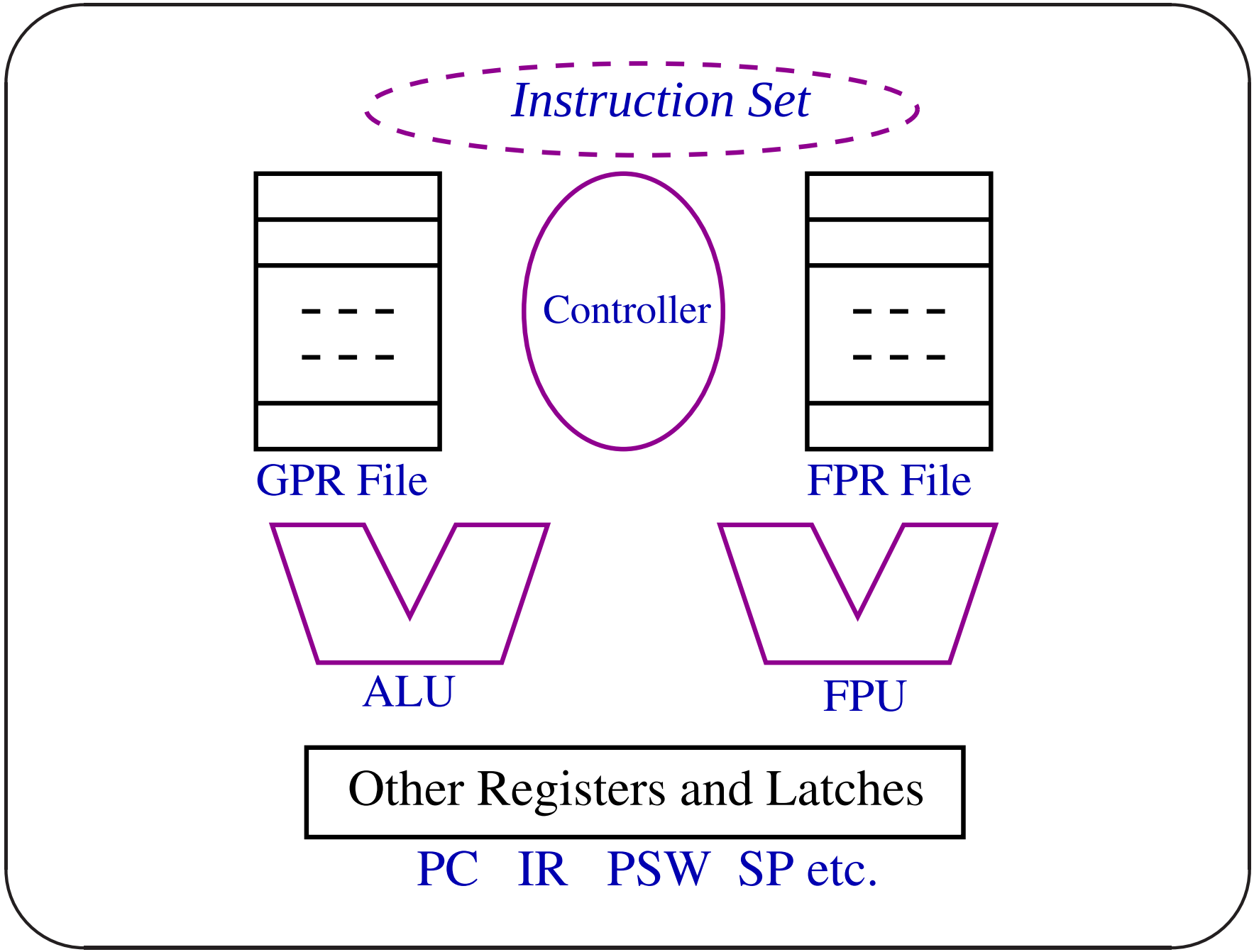
Software: GCC

Language: C

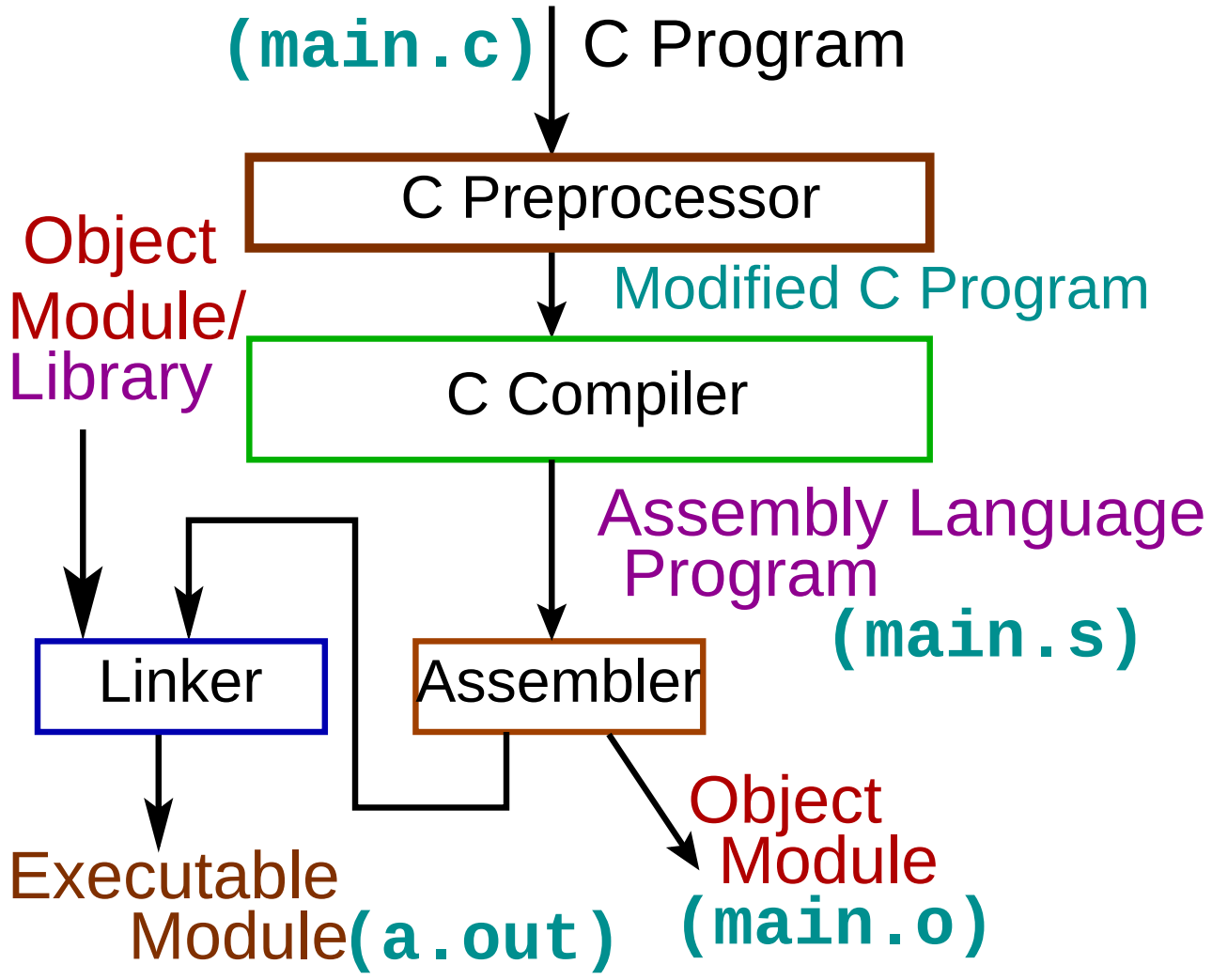
## Machines

You can use shell commands like `uname`, `lshw` to get information about the hardware system. You can also get information about the CPU from the file system - `$ cat /proc/cpuinfo`.

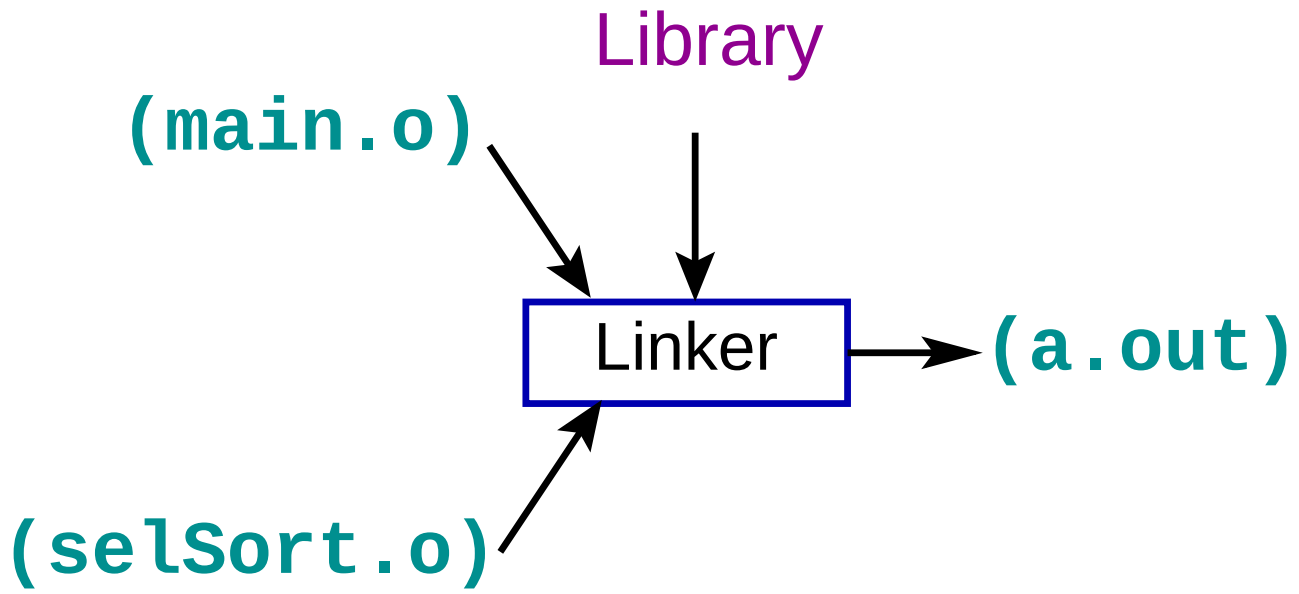

CPU–Memory Bus
Cache: L2
Cache L1
I D I D
Core I
Core II
CPU
DMA etc.
Main Memory
IO Bus
IO Controller
IO Controller
IO Controller
Mouse
Disk I
Disk II
Graphics VDU
Network


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

## CPP → Compiler → Assembler → Linker



## Separate Compilation and Linking



## Intel x86-64 Registers

GPRs: 64-bit integer registers (16) - rax, rbx, rcx, rdx, rsp, rbp, rsi, rdi, r8, $\cdots$, r15

FPRs: 80-bits floating point registers (8)- r0 $\cdots$ r7

MMXs: 64-bit SIMD registers (8) - mm0 $\cdots$ mm7

XMMs: 128-bit SSE registers (16) - xmm0 $\cdots$ xmm15

## Special Registers

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

## Main Memory Address

Address: 48 bits logical.

The width of any x86_64 address register is 64 bit. But the least significant 48 bits are taken as logical address.

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

## Register Usage Convention

| GPR(64) | 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 |

| 64-bit GPR | 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.

## Compiling a C Program

```
/*  main.c */
#include <stdio.h>
#define MAXNO 100
void function(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;
    function(data, no) ;
```

```
    printf("Data after processing: ") ;
    for(i = 0; i < no; ++i) printf("%d ", data[i]);
    putchar('\n') ;
    return 0 ;
}
```

## Compiling a C Program

```
/* function.c contains a function */
void function(int d[], int n){
     int i;

     for(i=0; i<n; ++i) d[i] = 6*d[i];
}
```

## Compilation

```
$ cc -Wall -S main.c ⇒ main.s
$ cc -Wall -c main.c ⇒ main.o
$ cc -Wall -S function.c ⇒ function.s
$ cc -Wall -c function.c ⇒ function.o
$ cc main.o function.o ⇒ a.out
```

C program files can be compiled separately and linked together.

## File Types

```
$ file main.o function.o
main.o:  ELF 64-bit LSB relocatable, x86-64,
version 1 (SYSV), not stripped
function.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), for GNU/Linux 2.6.24,
dynamically linked (uses shared libs), not
stripped
```

## Assembly Language Program: function.s

```
#/* function.c contains a function */
#void function(int d[], int n){
#     int i;
#
#     for(i=0; i<n; ++i) d[i] = 6*d[i];
#}

    .file   "function.c"     # Function name
    .globl   function        # 'function' is
                             # a global name
    .type   function, @function # function is a
```

```
                                # function
function:                       # Label function
.LFB0:
   .cfi_startproc
   pushq   %rbp                 # push rbp
   .cfi_def_cfa_offset 16
   .cfi_offset 6, -16
   movq    %rsp, %rbp           # rbp <-- rsp
   .cfi_def_cfa_register 6
   movq    %rdi, -24(%rbp)      # M[rbp-24] <-- rdi
                                # 1st parameter, d[],
                                # starting address of
                                #  the array
   movl    %esi, -28(%rbp)      # M[ebp-28] <-- esi
```

```
                                # 2nd parameter, n
   movl   $0, -4(%rbp)          # M[rbp-4] (i) <-- 0
   jmp    .L2                   # goto .L2
                                #  loop condition test
.L3:                            # .L3, loop starts
   movl   -4(%rbp), %eax        # eax <-- i
   cltq                         # rax <-- eax, 32-bit to
                                #    64-bit
   leaq   0(,%rax,4), %rdx      # rdx <-- 4*rax (4*i)
   movq   -24(%rbp), %rax       # rax <-- d[]
   addq   %rdx, %rax            # rax <-- d+4*i (&d[i])
   movl   (%rax), %edx          # edx <-- M[rax], d[i]
   movl   -4(%rbp), %eax        # eax <-- i
   cltq                         # rax <-- eax, 32-bit to
```

```
                                #    64-bit
    leaq    0(,%rax,4), %rcx    # rcx <-- 4*rax (4*i)
    movq    -24(%rbp), %rax     # rax <-- d[]
    addq    %rax, %rcx          # rcx <-- d+4*i (&d[i])
    movl    %edx, %eax          # eax <-- edx (d[i])
    addl    %eax, %eax          # eax <-- eax+eax (2*d[i])
    addl    %edx, %eax          # eax <-- eax+ edx (d[i]+
                                #           2*d[i]=3*d[i])
    addl    %eax, %eax          # eax <-- eax+eax (2*3*d[i]
                                #          = 6*d[i])
    movl    %eax, (%rcx)        # M[rcx] <-- eax
                                #   (d[i] = 6*d[i])
    addl    $1, -4(%rbp)        # i <-- i+1
.L2:                            # Loop condition test
```

```
    movl    -4(%rbp), %eax    # eax <-- i
    cmpl    -28(%rbp), %eax   # if i (eax) < n (M[rbp-4])
    jl    .L3                 #   goto .L3
    nop                       # no operation
    nop                       # no operation
    popq    %rbp              # pop rbp
    .cfi_def_cfa 7, 8
    ret                       # return
    .cfi_endproc
.LFE0:
    .size    function, .-function
    .ident    "GCC: (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0"
    .section    .note.GNU-stack,"",@progbits
```

## Assembly Language Program: main.s

```
#/*  main.c */
##include <stdio.h>
##define MAXNO 100
#void function(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;
#    function(data, no) ;
```

```
#    printf("Data after processing: ") ;
#    for(i = 0; i < no; ++i) printf("%d ", data[i]);
#    putchar('\n') ;
#    return 0 ;
#}

   .file   "main.c"          # source file name
   .section   .rodata        # read-only data section starts
   .align 8                  # align with 8-byte boundary
.LC0:                        # Label of string - 1st printf
   .string   "Enter the data, terminate with Ctrl+D"
.LC1:                        # Label of string scanf
   .string   "%d"
.LC2:                        # Label of string - 2nd printf
```

```
    .string   "Data after processing: "
.LC3:                          # Label of string - 3rd printf
    .string   "%d "
    .text                      # Code or text starts
    .globl    main             # main is a global name
    .type   main, @function    # main is function
main:                          # Label main:
.LFB0:
    .cfi_startproc
    pushq   %rbp               # Save old base pointer
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq   %rsp, %rbp          # rbp <-- rsp set new
    .cfi_def_cfa_register 6 #     base pointer
```

```
    subq    $432, %rsp          # Create space for local
                                # array and variables
    movq    %fs:40, %rax        # Save segment info
    movq    %rax, -8(%rbp)      # at M[rbp-8]
    xorl    %eax, %eax          # eax <-- 0
    movl    $0, -424(%rbp)      # no <-- 0
    leaq    .LC0(%rip), %rdi    # rdi <-- 1st parameter
                                #              of printf
    call    puts@PLT            # Calls puts for printf
    jmp    .L2                  # Goto the beginning of the
                                # while loop
.L3:
    addl    $1, -424(%rbp)      # no <-- no+1
.L2:                            # Body of the loop
```

```
leaq   -416(%rbp), %rax # rax <-- data
movl   -424(%rbp), %edx # edx <-- no
movslq   %edx, %rdx     # rdx <-- edx (no) (32-bits
                        #           to 64-bits)
salq   $2, %rdx         # rdx <-- 4*rdx (no)
addq   %rdx, %rax       # rax <-- data + 4*no
                        #           (&data[no])
movq   %rax, %rsi       # rsi <-- rax (&data[no])
                        #   2nd parameter
leaq   .LC1(%rip), %rdi # rdi <-- (addr. of format str)
                        #   1st parameter
movl   $0, %eax         # eax <-- 0
call   __isoc99_scanf@PLT  # call to scanf
                        # The return value is in eax
```

```
cmpl   $-1, %eax          # if return value
                          #          != -1 (EOF)
jne    .L3                # goto .L3 (loop)
movl   -424(%rbp), %edx   # edx <-- no
leaq   -416(%rbp), %rax   # rax <-- data
movl   %edx, %esi         # esi <-- edx (no) 2nd
                          #         parameter
movq   %rax, %rdi         # rdi <-- rax (data), 1st
                          #         parameter
call   function@PLT       # call to selection sort
leaq   .LC2(%rip), %rdi   # rdi <-- format str address
                          #    1st param
movl   $0, %eax           # eax <-- 0
call   printf@PLT         # call to printf
```

```
    movl    $0, -420(%rbp)    # i <-- 0
    jmp     .L4               # go to .L4
.L5:
    movl    -420(%rbp), %eax  # eax <-- i
    cltq                      # rax <-- eax (sign ext.)
    movl    -416(%rbp,%rax,4), %eax # eax <--
                              # Mem[(rbp - 416) + 4*rax]
                              # eax <-- data[i]
    movl    %eax, %esi        # esi <-- eax (data[i]),
                              #    2nd parameter
    leaq    .LC3(%rip), %rdi  # rdi <-- addr. format str,
                              #     1st parameter
    movl    $0, %eax          # eax <-- 0
    call    printf@PLT        # call to printf
```

```
   addl   $1, -420(%rbp)    # i <-- i + 1
.L4:                        # Label to test loop cond
   movl   -420(%rbp), %eax  # eax <-- i
   cmpl   -424(%rbp), %eax  # if i < no (jl is jump <)
   jl   .L5                 #    goto .L5
   movl   $10, %edi         # edi <-- 10 ('\n')
   call   putchar@PLT       # call putchar
   movl   $0, %eax          # eax <-- 0
   movq   -8(%rbp), %rcx    # Restore return info
   xorq   %fs:40, %rcx
   je   .L7                 # return
   call   __stack_chk_fail@PLT
.L7:
   leave                    # return
```

```
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident   "GCC: (Ubuntu 9.4.0-1ubuntu1~18.04) 9.4.0"
    .section   .note.GNU-stack,"",@progbits
```

## Assembly Language Program: sqrtNewton.s

```
#// 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;
#}
    .file   "sqrtNewton.c"
    .section    .rodata
.LC0:
    .string   "Enter a +ve number: "
.LC1:
    .string   "%lf"
.LC6:
```

```
    .string   "sqrt(%f) = %f\n"
    .text
    .globl    main
    .type     main, @function
main:
.LFB0:
    .cfi_startproc
    pushq     %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq      %rsp, %rbp
    .cfi_def_cfa_register 6
    subq      $32, %rsp
    movl      $.LC0, %eax
```

```
movq    %rax, %rdi
movl    $0, %eax
call    printf          # code up to this is
                        # similar to what we
                        # have already seen
movl    $.LC1, %eax     # eax <-- address of
                        #  the format string
                        # for scanf
leaq    -24(%rbp), %rdx # rdx <-- &k
movq    %rdx, %rsi       # rsi <-- rdx (&k)
                         #    2nd parameter
movq    %rax, %rdi       # rdi <-- rax, 1st param
movl    $0, %eax         # eax <-- 0
call    __isoc99_scanf   # call to scanf
```

```
    movsd   -24(%rbp), %xmm0   # xmm0 <-- k
    movsd   .LC2(%rip), %xmm1  # xmm1 <-- M[rip + .LC2]
                               # double word (64-bit)
    divsd   %xmm1, %xmm0       # xmm0 <-- xmm0/xmm1 (k/2)
    movsd   %xmm0, -16(%rbp)   # root <-- xmm0 (k/2)
.L2:
    movq    -16(%rbp), %rax    # rax <-- root
    movq    %rax, -8(%rbp)     # M[rbp - 8] <-- rax
                               # oldR <-- root
    movsd   -16(%rbp), %xmm0   # xmm0 <-- M[rbp-16] (root)
    mulsd   -16(%rbp), %xmm0   # xmm0 <-- xmm0*root
                               # xmm0 <-- root*root
    movsd   -24(%rbp), %xmm1   # xmm1 <-- k
    addsd   %xmm0, %xmm1       # xmm1 <-- xmm0 + xmm1
```

```
                                    # xmm1 <-- root*root + k
movsd   -16(%rbp), %xmm0            # xmm0 <-- root
addsd   %xmm0, %xmm0                # xmm0 <-- xmm0+xmm0
                                    # xmm0 <-- root + root
                                    # xmm0 <-- 2.0*root
                                    # strength reduction
movapd   %xmm1, %xmm2               # xmm2 <-- xmm1
                                    # xmm2 <-- root*root + k
divsd   %xmm0, %xmm2                # xmm2 <-- xmm2/xmm0
                                    # (root*root + k)/(2.0*root)
movapd   %xmm2, %xmm0               # xmm0 <-- xmm2
                                    # xmm0 <-- (root*root + k)/(2.
movsd   %xmm0, -16(%rbp)            # root <-- xmm0
movsd   -8(%rbp), %xmm0             # xmm0 <-- oldR
```

```
subsd   -16(%rbp), %xmm0  # xmm0 <-- oldR - root
divsd   -16(%rbp), %xmm0  # xmm0 <-- (oldR - root)/root
movsd   .LC3(%rip), %xmm1 # xmm1 <-- mask
andpd   %xmm1, %xmm0      # xmm0 <-- xmm0 & mask
                          # abs(oldR - root)/root)
movsd   .LC4(%rip), %xmm1 # xmm1 <-- 100
mulsd   %xmm1, %xmm0      # xmm0 <-- 100*abs(oldR - root
ucomisd   .LC5(%rip), %xmm0 #  Compare xmm0 > 0.01
seta   %al                #
testb   %al, %al
jne   .L2                 # Goto loop
movsd   -24(%rbp), %xmm0  # xmm0 <-- k
                          #  2nd param
movl   $.LC6, %eax        # eax <-- format
```

```
    movsd   -16(%rbp), %xmm1   # xmm1 <-- root
                               #  3rd param
    movq    %rax, %rdi         # rdi <-- 1st param
    movl    $2, %eax           # eax <-- 2 (?)
    call    printf             # call printf
    movl    $0, %eax           # eax <-- 0
                               # return value
    leave                      # purge activation record
    .cfi_def_cfa 7, 8
    ret                        # return
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .section    .rodata
```

```
    .align 8
.LC2:   # 2.0
    .long   0
        # 0000 0000 0000 0000 0000 0000 0000 0000
    .long   1073741824
        # 0 100 0000 0000 .0000 0000 0000 0000 0000
    .align 16
.LC3:   # Mask to take fabs()
    .long   4294967295
        # 1111 1111 1111 1111 1111 1111 1111 1111
    .long   2147483647
        # 0111 1111 1111 1111 1111 1111 1111 1111
    .long   0
    .long   0
```

```
    .align 8
.LC4:    # 100.0
    .long   0
         # 0000 0000 0000 0000 0000 0000 0000 0000
    .long   1079574528
         # 0 100 0000 0101 .1001 0000 0000 0000 0000
         # 100(D) = 1.100100 X 2^6, 6 is 6 + 1023
         # = 1029 = 1024 + 5
    .align 8
.LC5:    # 0.01
    .long   1202590843
         # 0100 0111 1010 1110 0001 0100 0111 1011
    .long   1065646817
         # 0 011 1111 1000 .0100 0111 1010 1110 0001
```

```
.ident   "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
.section   .note.GNU-stack,"",@progbits
```