Experiment I: An Introduction

Machine

Processor: Intel Core 2 Duo 64-bit, E6550, 2.33GHz

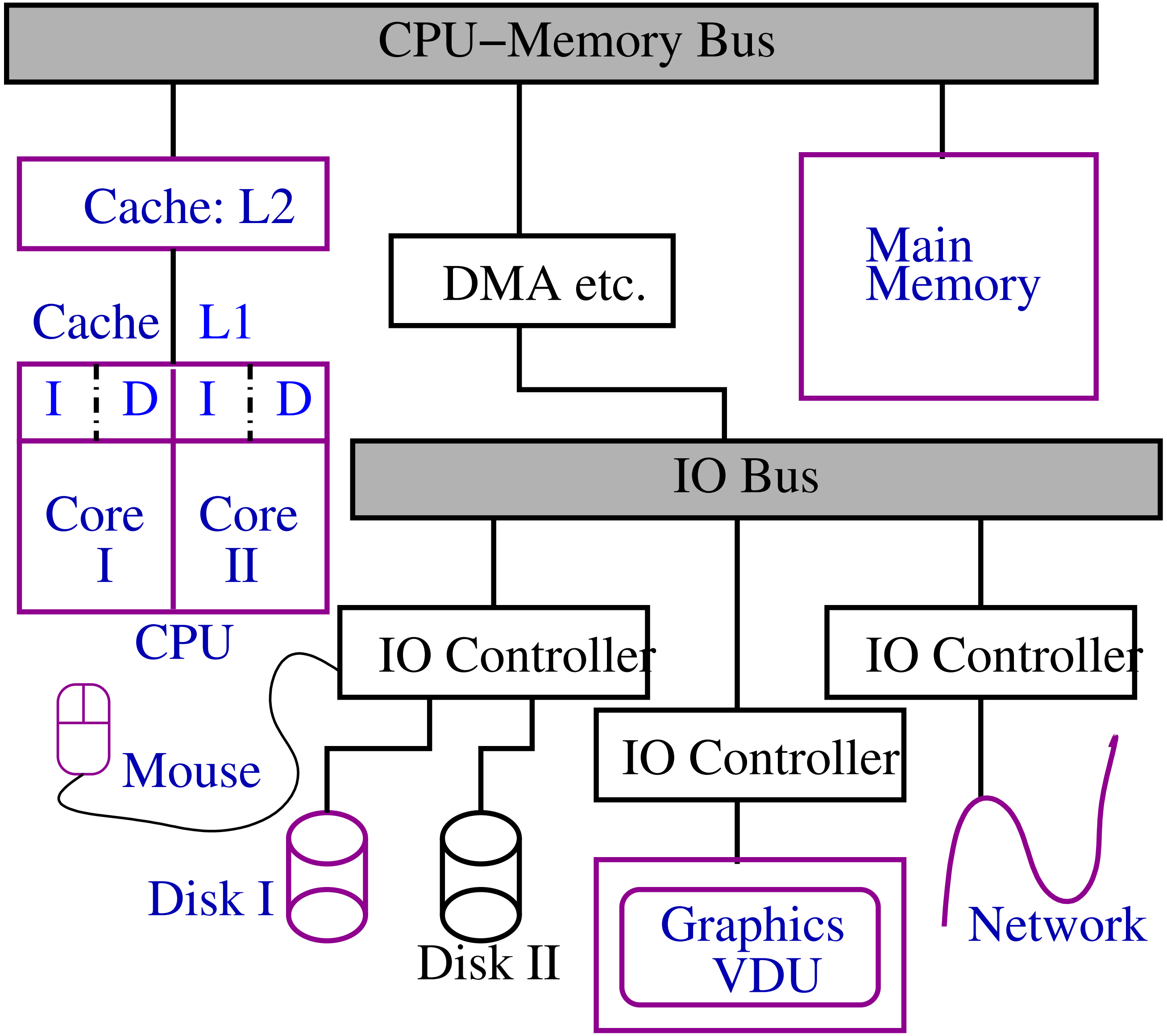
Memory: 2 × 32 KB (L1: 8-way set assoc. IC)
2 × 32 KB (L1: 8-way set assoc. DC)
4MB (L2: 16-way set assoc.),
4GB (main memory)

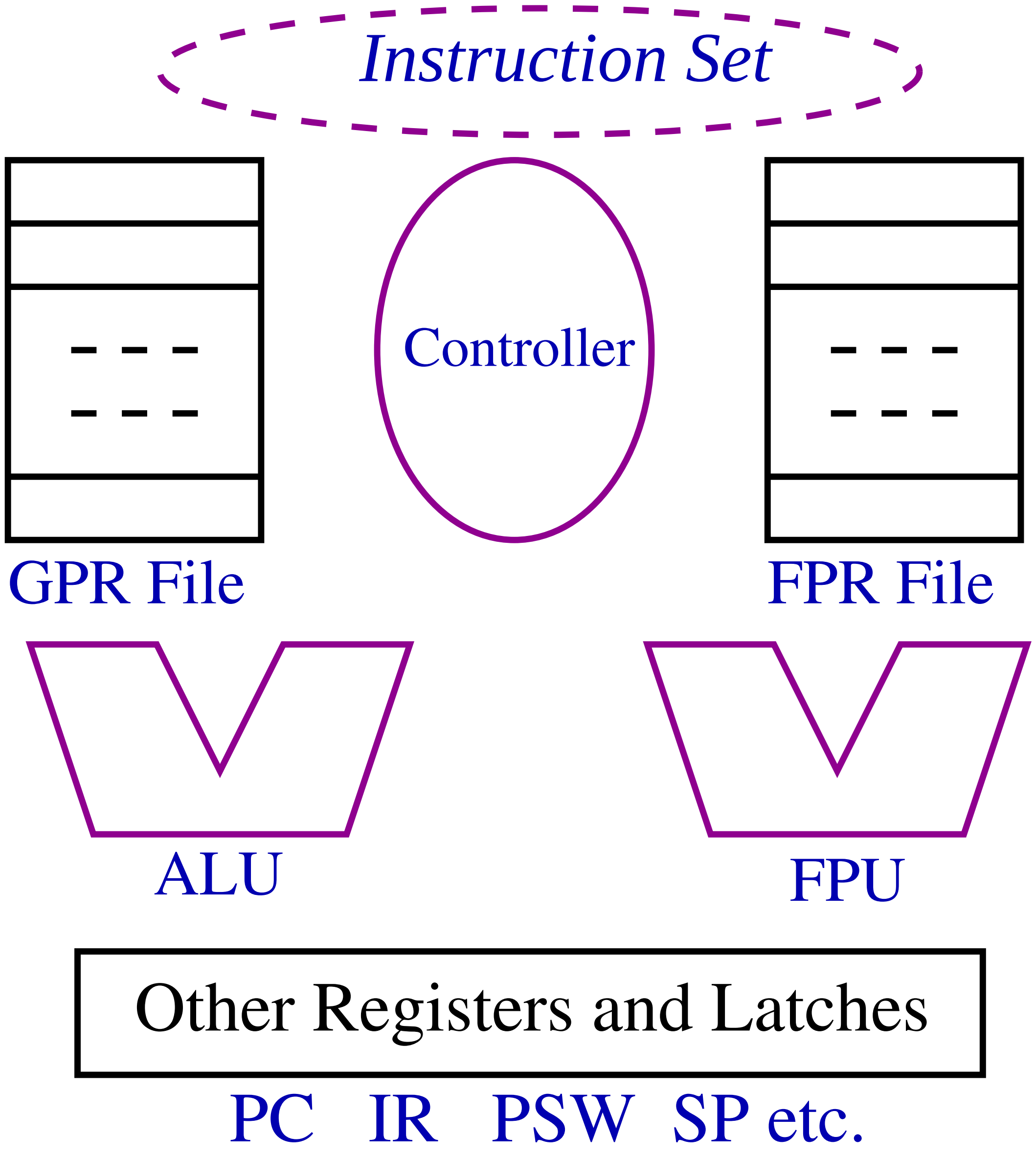
Software

OS:	GNU/Linux, 64-bit, x86_64
Software:	GCC
Language:	C++

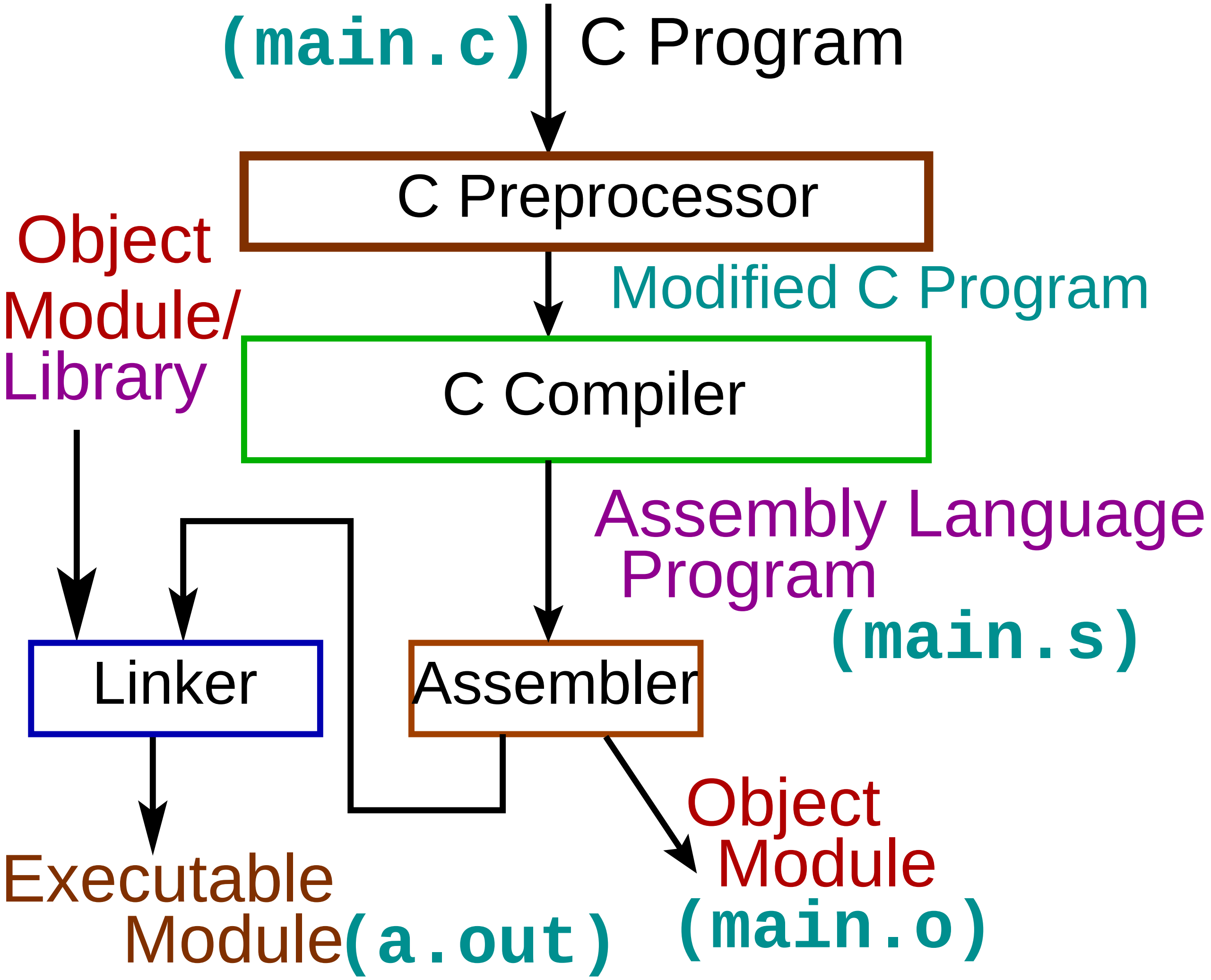
Machines

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`.

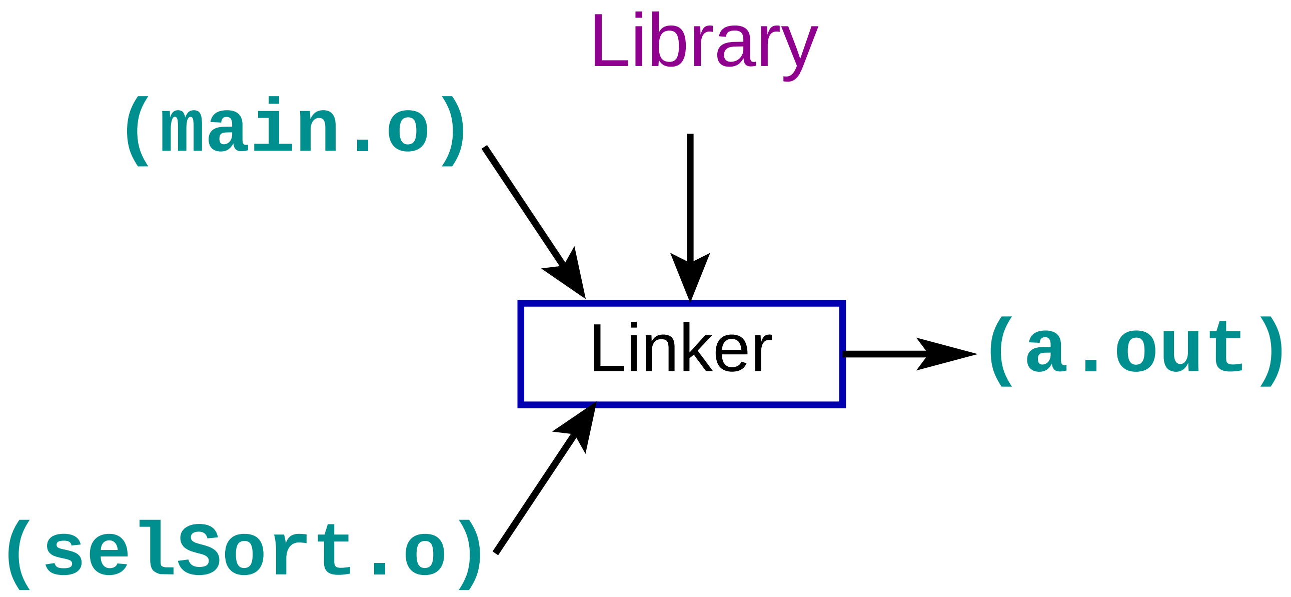

CPU–Memory Bus
Cache: L2
DMA etc.
Main
Memory
Cache L1
I
D
I
D
IO Bus
Core
I
Core
II
CPU
IO Controller
IO Controller
Mouse
IO Controller
Disk I
Disk II
Graphics
VDU
Network


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: 39 bits physical, 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

```
#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) { // selSort.c
     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);
     }
} // selSort.c
```

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
```

C program files can be compiled separately and linked together.

File Types

```
$ 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), for GNU/Linux 2.6.24,
dynamically linked (uses shared libs), not
stripped
```

Assembly Language Program: `main.s`

```
   .file   "main.c"        # source file name
   .section   .rodata      # read-only data section
   .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 in sorted Order are: "
.LC3:                      # Label of string - 3rd printf
   .string   "%d "
#
```

```
   .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
   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
                         #    2-bit left (4*no)
   leaq   -416(%rbp), %rsi  # rsi <-- (rbp - 416)
                         #       (&data)
```

```
addq    %rax, %rsi        # rsi <-- rsi + rax
                          #   (data+4*no = &data[no])
                          #   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
```

```
#
   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
   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
#
```

```
.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 Language Program: selSort.s

```
    .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
```

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

```
.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
```

```
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)
```

```
    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 .eh_frame

```
    .section    .eh_frame,"a",@progbits
.Lframe1:
    .long    .LECIE1-.LSCIE1
.LSCIE1:
    .long    0x0
    .byte    0x1
    .string    "zR"
    .............
    .align 8
.LEFDE1:
    .ident    "GCC: (GNU) 4.2.3 (4.2.3-6mnb1)"
    .section    .note.GNU-stack,"",@progbits
```

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.

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
```