Compilers Laboratory: CS39003

Autumn Semester: 2014 - 2015

Instructor: P P Das & Goutam Biswas

Department: Computer Science and Engineering

Teaching Assistants

Name	Mobile No.	E-mail address
Tanwi Mallika	9674277774	tanwireachesu@...
Akash Anuj	7501465026	aakashanuj.iitkgp@...
Aakash Goenka	9831784511	AAKASH.GOENKA@...
Abhishek	7501465092	ABHISHEK2175@...
Rajan Buha	8927027373	RAJAN.BUHA@...

$\cdots \equiv$ gmail.com

Marks Distribution

- *Laboratory Quiz* - 30 marks
- *Laboratory Assignments* - 70 marks.

Class Timings

Class Room No. NC - 243

Day	*Time*
Tuesday	14:30 hours

Machines

Processor: Intel Core 2 Duo 32-bit,

OS: GNU/Linux, 32-bit

Software: GCC, Lex/Flex and Yacc/Bison

IA-32 Registers and Memory

GPRs: eight 32-bit registers - eax, ebx

ecx, edx, esp, ebp, esi, edi

FPRs: eight 80-bit floating point

registers - r0 $\cdots$ r7 (organized as stack)

Memory

Address: 32-bit byte address

32-bit eflags and 32-bit eip (PC)[a]

[a] There are many other registers e.g. MMX, XMM, control registers, segment registers etc.

x86-32 Integer Registers

32-bit GPR	GCC Usage Convention
eax	*return value*, *caller saved*
ebx	*callee saved*
ecx	*caller saved*
edx	*caller saved*
esi	*callee saved*
edi	*callee saved*
ebp	*callee saved*
esp	*hardware stack pointer*

x86-32 Integer Registers

After the Call	
32-bit GPR	GCC Usage Convention
eip	*First Instruction of the function*
esp+4	*First argument*
esp	*Return address*

x86-32 Integer Registers

After the Return	
32-bit GPR	GCC Usage Convention
eip	*Return address*
esp	*Arguments pushed by caller*
eax	*Return value*
ecx, edx	*Trash*
ebp, ebx	*Valid values*
esi, edi	*Valid values*

Intel-64 Registers and Memory

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

FPRs: eight 80-bits floating point registers - r0 $\cdots$ r7

MMXs: Eight 64-bit registers - mm0 $\cdots$ mm7

XMMs: Sixteen 128-bit registers - xmm0 $\cdots$ xmm15

Intel-64 Registers and Memory

Memory

Address: 64-bit byte address

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

x86-64 Integer Registers

64-bit GPR	Usage Convention
rax	*return value from a function*
rbx	*callee saved*
rcx	*4th argument to a function*
rdx	*3rd argument to a function*
rsi	*2nd argument to a function*
rdi	*1st argument to a function*
rbp	*callee saved*
rsp	*hardware stack pointer*

x86-64 Integer Registers

64-bit GPR	Usage Convention
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*

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 32-bit LSB relocatable, Intel
80386, ...
selSort.o:  ELF 32-bit LSB relocatable, Intel
80386, ...
$ file a.out
a.out:  ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.15, not
stripped
```

Assembly Language Program: main.s

```
   .file   "main.c"      # source file name
   .section   .rodata    # read-only data section
   .align 4              # align with 4-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:                       # Label main:
   pushl   %ebp             # Save old base pointer
   movl   %esp, %ebp        # ebp <-- esp, set new
                            # base pointer
   andl   $-16, %esp        # Adjust to 16-byte boundary
   subl   $432, %esp        # Create space for the
                            # local array
   movl   $0, 428(%esp)     # no <-- 0
   movl   $.LC0, (%esp)     # Push the 1st parameter
   call   puts              # Call puts() to print
   jmp   .L2                # goto Label label .L2
.L3:                        # while loop (label .L3)
```

```
   addl   $1, 428(%esp) # no <-- no+1
.L2:
   movl   428(%esp), %eax  # eax <-- M[esp + 428] (no)
   leal   0(,%eax,4), %edx # edx <-- 4*eax (4*no)
   leal   24(%esp), %eax   # eax <-- esp + 24 (data)
   addl   %edx, %eax       # eax <-- eax + edx
                           # eax <-- (data + 4*no)
   movl   $.LC1, %edx      # edx <-- address of format stri
   movl   %eax, 4(%esp)    # Push the second parameter
   movl   %edx, (%esp)     # Push the 1st parameter
   call   __isoc99_scanf   # Call scanf
   cmpl   $-1, %eax        # If the return value is
                           # not equal to EOF (-1)
   jne    .L3              # goto .L3 (loop)
```

```
    movl    428(%esp), %eax   # eax <-- no
    movl    %eax, 4(%esp)     # Push the second parameter
    leal    24(%esp), %eax    # eax <-- esp + 24 (data)
    movl    %eax, (%esp)      # Push the first parameter
    call    selectionSort     # call selectionSort()
    movl    $.LC2, %eax       # eax <-- address of printf stri
    movl    %eax, (%esp)      # Push the address
    call    printf            # Call printf
    movl    $0, 424(%esp)     # M[esp+424] (i) <-- 0
    jmp    .L4                # goto .L4
.L5:
    movl    424(%esp), %eax   # eax <-- i
    movl    24(%esp,%eax,4), %edx
                              # edx <-- M[esp+24+4*eax]
```

```
                                    # edx <-- M[data+4*i] (data[i])
   movl   $.LC3, %eax               # eax <-- address of format stri
   movl   %edx, 4(%esp)             # Push data[i] (second parameter
   movl   %eax, (%esp)              # Push format (1st parameter)
   call   printf                    # Call printf
   addl   $1, 424(%esp)             # i <-- i+1
.L4:
   movl   424(%esp), %eax           # eax <-- i
   cmpl   428(%esp), %eax           # if i < no
   jl   .L5                         # goto .L5 (loop)
   movl   $10, (%esp)               # Push 10 (\n)
   call   putchar                   # Call putchar
   movl   $0, %eax                  # eax <-- 0
   leave
```

```
ret
.size   main, .-main
.ident   "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
.section   .note.GNU-stack,"",@progbits
```

Assembly Language Program: selSort.s

```
   .file   "selSort.c" # source file
   .text               # code
.globl selectionSort   # selectionSort is global
   .type   selectionSort, @function
selectionSort:
   pushl   %ebp        # save old base pointer
   movl   %esp, %ebp   # ebp <-- esp, new
                       # base pointer
   subl   $16, %esp    # Create 16 byte stack
                       # frame
   movl   $0, -4(%ebp) # M[ebp - 4] (i) <-- 0
   jmp   .L2           # goto .L2
```

```
.L6:                        # outer loop
   movl   -4(%ebp), %eax # eax <-- i
   sall   $2, %eax         # eax <-- eax << 2
                           # eax <-- 4*i
   addl   8(%ebp), %eax  # eax <-- M[ebp + 8] + eax
                           # eax <-- data + 4*i
   movl   (%eax), %eax   # eax <-- M[data+4*i]
                           # eax <-- data[i]
   movl   %eax, -16(%ebp) # temp = data[i]
   movl   -4(%ebp), %eax #  eax <-- i
   movl   %eax, -8(%ebp) # max <-- eax (i)
   movl   -4(%ebp), %eax # eax <-- i
   addl   $1, %eax         # eax <-- eax + 1
                           # eax <-- i+1
```

```
    movl   %eax, -12(%ebp) # j <-- eax (i+1)
    jmp    .L3             # goto .L3
.L5:                       # inner loop
    movl   -12(%ebp), %eax # eax <-- j
    sall   $2, %eax        # eax <-- eax << 2
    addl   8(%ebp), %eax   # eax <-- data + 4*j
    movl   (%eax), %eax    # eax <-- data[j]
    cmpl   -16(%ebp), %eax # if data[j] <= temp
    jle    .L4             # goto .L4
    movl   -12(%ebp), %eax # eax < -- j
    sall   $2, %eax        # eax <-- 4*j
    addl   8(%ebp), %eax   # eax <-- data+4*j
    movl   (%eax), %eax    # eax <-- data[j]
    movl   %eax, -16(%ebp) # temp <-- data[j]
```

```
   movl   -12(%ebp), %eax # eax <-- j
   movl   %eax, -8(%ebp)  # max <-- j
.L4:
   addl   $1, -12(%ebp)   # j <-- j+1
.L3:
   movl   -12(%ebp), %eax # eax <-- j
   cmpl   12(%ebp), %eax  # if j < nod
   jl   .L5               # goto .L5
   movl   -4(%ebp), %eax  # eax <-- i
   sall   $2, %eax        # eax <-- 4*i
   addl   8(%ebp), %eax   # eax <-- data+4*i
   movl   (%eax), %eax    # eax <-- data[i]
   movl   %eax, -16(%ebp) # temp <-- data[i]
   movl   -4(%ebp), %eax  # eax <-- i
```

```
    sall    $2, %eax          # eax <-- 4*i
    addl    8(%ebp), %eax     # eax <-- data+4*i
    movl    -8(%ebp), %edx    # edx <-- j
    sall    $2, %edx          # edx <-- 4*j
    addl    8(%ebp), %edx     # edx <-- data + 4*j
    movl    (%edx), %edx      # edx <-- data[j]
    movl    %edx, (%eax)      # data[i] <-- data[j]
    movl    -8(%ebp), %eax    # eax <-- j
    sall    $2, %eax          # eax <-- 4*j
    addl    8(%ebp), %eax     # eax <-- data + 4*j
    movl    -16(%ebp), %edx   # edx <-- temp
    movl    %edx, (%eax)      # data[j] <-- temp
    addl    $1, -4(%ebp)      # i <-- i + 1
.L2:
```

```
movl   12(%ebp), %eax  # eax <-- nod
subl   $1, %eax        # eax <-- eax - 1 (nod - 1)
cmpl   -4(%ebp), %eax  # if nod - 1 > i
jg   .L6               # go to .L6 (loop)
leave
ret
.size   selectionSort, .-selectionSort
.ident   "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
.section   .note.GNU-stack,"",@progbits
```