School of Mathematical and Computational Sciences

Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial III (21 January, 2026)

M. Sc Semester IV: 2025-2026 *Instructor:* Goutam Biswas

Following are two examples of GNU x86-64 inline assembly language code.

```
#include <stdio.h>
int main(){     // asm1.c
int m, n=20;

__asm__ __volatile__ (
"sall $2, %0     # 4*n \n\t"
"movl %0, %%eax  # m = 4*n \n\t"
  : "=a" (m)
  : "r" (n)
);
printf("m: %d\n", m);
return 0;
}
```

```
#include <stdio.h>
int main(){     // asm2.c
int m=20;

__asm__ __volatile__ (
"movl %%eax, %%ebx # ebx <-- m \n\t"
"sall $1, %%eax    # 2*m \n\t"
"addl %%ebx, %%eax  # m = 3*n \n\t"
 : "+a" (m)
 :
);
printf("m: %d\n", m);
return 0;
}
```

Corresponding outputs are:

```
$ cc -Wall asm1.c
$ ./a.out
m: 80
$
```

```
$ cc -Wall asm2.c
$ ./a.out
m: 60
$
```

Following two functions use `read()` *and* `write()` *system calls to read and write one character from the* `stdin` *and* `stdout`*.*

```
#include <unistd.h>
char readChar(){
     char c;

     read(STDIN_FILENO, &c, 1);
     return c;
}
```

```
#include <unistd.h>
void writeChar(char c){
     write(STDOUT_FILENO, &c, 1);
}
```

These call to `read()` *(similarly* `write()`*) can be replaced by embedded inline assembly language code of x86-84 architecture with a software interrupt.*

```
char readChar(){
     char c=0;

      __asm__ __volatile__ (
     "movq $0, %%rax # 0 for read \n\t"
     "movq $0, %%rdi # 0 for stdin \n\t"
     "movq $1, %%rdx # 1 byte \n\t"
     "syscall \n\t"
     :          // write "=r" or read/write "+r"
     :"S" (&c)  // read-only "r"
     ) ;
```

```
    return c;
}
```

Note the following ABI specification:

(a) `rax` ← *system call code.*

(b) `rdi` ← *1st parameter.*

(c) `rsi` ← *2nd parameter.*

(d) `rdx` ← *3rd parameter.*

(e) `eax` → *return value.*

Link:
`https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md`

Also note the following inline assembly language codes for different CPU registers.

(i) `rdi`*: 'D'*

(ii) `rsi`*: 'S'*

(iii) `rdx`*: 'd'*

(iv) `rax`*: 'a'*

Link: `https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html`

Exercise 1. **[3+3+4]**

(a) Write a function `int myRead(int fd, char *bP, int cC)` *using inline assembly code and* `syscall` *(no other function call) to read characters from the open file in the buffer.* `fd` *is the open file descriptor,* `bP` *is the starting address of the buffer, and* `cC` *is the number of characters to read.*

(b) Write a function `int myWrite(int fd, char *bP, int cC)` *using inline assembly code and* `syscall` *(no other function call) to write characters from the buffer to the open file. The open file descriptor is* `fd`*,* `bP` *is the starting address of the buffer,* `cC` *is the number of characters to write.*

(c) Write a function `int myReadWrite(int rw, int fd, char *bP, int cC)` *that will behave like* `myRead()` *or* `myWrite()` *depending on the first parameter* `rw` *(0: read, 1: write). This also will use inline assembly code and* `syscall` *(no other function call).*

(d) The return value is the actual number of characters read/written. This is available from the register `%eax`*.*

Your functions will be tested as follows:

```
int main(){
    char buff[100]={'\0'};
    int len;
    len = myRead(0, buff, 99);
    myWrite(1, buff, len);
```

```
    myReadWrite(1, 1, buff, len);
    len = myReadWrite(0, 1, buff, 99);
    myReadWrite(1, 1, buff, len);

    return 0;
}
$ ./a.out
I can read
I can read
I can read
I can write
I can write
```

Send the three functions in a file `<roll-no>.3.c` *to* `goutamamartya@gmail.com`. **Do not include** `main()` *in the file.*

Exercise 2.

(a) *Construct the DFA corresponding to the regular expression* $(a|b)^*a(a|b)(a|b)$ *using the* $'\bullet'$*-items.*

(b) *Augment the regular expression* $(a+b)^*ba(a+b)^*$ *with* $'\#'$.

(i) *Construct the syntax tree of the augmented regular expression and attach position numbers to different leaf nodes with symbols of the alphabet.*

(ii) *Label the nodes with firstpos and lastpos.*

(iii) *Identify the nullable nodes and compute sets of followpos of different positions.*

(iv) *Construct the NFA and then the DFA equivalent to the regular expression.*