**ASSIGNMENT – 3**

Id- 2018ucp1505

Name- Ashutosh Soni

**Question 1: Write Machine code translation for the following?**

**a)**

**If(cond1)**

**Statement 1;**

**Elseif(cond2)**

**Statement 2;**

**Else**

**Statement 3;**

**Endif**

**Statement 4;**

**Answer:**

Machine code translation,

cond1

Check cond1 == True

JNZ L1

Statement 1

JMP L3

L1: cond2

Check cond2 == True

JNZ L2

Statement 2

JMP L3

L2: Statement 3

L3: Statement 4

**b)**

**Switch(value) {**

**case val1: stmt1; break;**

**case val2: stmt2; break;**

**case val3: stmt3; break;**

**default stmt4;**

**}**

**Answer:**

The same program variant in if-elseif- else format

If (value == val1) {

stmt1;

}

Elseif (value == val2) {

stmt2;

}

Elseif (value == val3) {

stmt3;

}

Else {

stmt4;

}

Now, Machine code translation,

JMP Switch\_start

Case1:

stmt1

JMP End

Case2:

stmt2

JMP End

Case3:

stmt3

JMP End

Switch\_start:

Compare value, val1

JZ Case1

Compare value, val2

JZ Case2

Compare value, val3

JZ Case3

stmt4

End

**c) Repeat**

**Statement1**

**Until(cond)**

**Statement2**

**Answer:**

Machine code translation,

L1:

Statement1;

Check cond == True

JZ L1

L2:

Statement2

**d) How would the machine code translation change when break continue are used in the above program?**

**Answer:**

Let’s the code is

Repeat

Statement1;

if(cond1)

continue;

Statement2;

if(cond2)

break;

Statement3; // It contains something change the cond3

Until(cond3)

Statement4;

Machine code translation

L1:

Statement1

Check cond1 == True

JZ L1

Statement2

Check cond2 == True

JZ L2

Statement3

Check cond3 == True

JZ L1

L2:

Statement4

**Question 2: Study gcc command and list its options for**

**a) invoking linking to different libraries.**

**Answer:**

The program first compiles it to object file ‘. o' extension and up to this there is no linking of the library.

After that linking is performed when the input file is ‘. o' file rather than '.cpp' or '.c' file. GCC uses a separate linker program called ld.exe to perform the linking.

In case of linking some other libraries, which are not linked by default we simply use this type of format for linking

-L {path to file containing library} -l${library name}

For example, if I have a library libm.so in home/newall/lib/ the I had to do the following to link it into my program

$ gcc myprog myprog.c -L/home/newall/lib -lm

You may also need to specify and include path so the compiler can find the library header file: -I/home/newall/include

**b) Optimization related**

**Answer:**

There are many types of optimizations are there. They take sort of more time to compile the code and may result in large space and time efficient machine code. Some optimizations are also related with debugging.

Some of the optimizations listed below

-O0: This is the by default optimization which will be performed and results in Reducing compile time and make debugging as such to produce the expected result.

-O1: This optimization takes more time in compilation and lot more memory for a large function. This optimization turns on some flag. In this the compiler tries to reduce the code size as well as the execution time without performing the optimization that are great deal of compilation time.

-O2: This optimization optimizes even more than O1 optimization It covers up all the O1 optimization and add some more optimization by turning some more flags on. This option increases both compilation time and the performance of generated code.

-O3: This is even more optimize than O2 optimization. It turns on all the flags of O2 optimization and turns on some additional flags.

-Os: This is the optimization for size. This optimization covers all the O2 optimization except some which might increase size.

-Ofast: This optimization includes all the O3 optimization and some more optimizations that are not valid for all standard-complaint programs.

-Og: This optimization optimizes the debugging experience. This should be the choice for the standard edit-compile-debug cycle, offering a reasonable level of optimization while maintaining fast compilation and a good debugging experience.

They can be used as follow:

gcc -O {} filename.c -o filename

**c) Assembly output related**

**Answer:**

The gcc provides a feature to get the code before sending it to assembler. This will give the assembly language code. The command for the following is:

gcc -S programname.c

This command will convert the given high-level program to assembly level code and returns the same.

**d) Any other of interest**

**Answer:**

-Wall: This will check not only the error but also for all kinds warning like unused variable errors.

It can be break like

'all' and W stands for 'warning'.

**Question 3: What is ps command? What does it show about a process status?**

**Answer:**

In ps command ps stands for process status. It provides information about the currently running processes, including their process identification number (PIDs). This provides us the appropriate privileges to obtain information about the requested processes.

When a description says that ps lists all processes, it means all the processes on the system, provided that you have appropriate privileges

Some options it accepts are

-A: displays information on all accessible processes.

-a: displays information on all processes associated with terminals

-d: displays information for all processes except group leaders.

There are many more suffix which can be used with it to show desired information.