Software Compiler Assignment on:

Control Flow and i386

The Software Compiler course exam is composed by two parts. One is a written test, the other is an homework, to be terminated before course last class. The written test contributes with the 40% to the whole grade, while the homework contributes with the remaining 60%. To pass the whole exam, you must get a pass grade from both the test and the homework.

During the lab classes, we show you the ACSE compiler and the associated assembler [3]. They must be used as starting point for your homework. During the last class, you must present your work, showing your edits, giving a brief demo, and responding at some questions given by course lecturers.

Sources of ACSE can be found on the course site [5]. It is a tarball of the ACSE mercurial [1] repository. You are required to version your code with mercurial. A copy of your edits must be submitted to course lectures in the form of a patch with respect to the version of ACSE you have used as starting point.

Assignment

You are required to modify both the front-end of the compiler and the back-end.

Front-end

Figure 1 reports two missing control statement in LANCE.

```
int a, b;

a = b > 10 ? b - 1 : 4;
```

(a) Ternary operator

```
int a, b;

a = a + 1 @(b != 0);
```

(b) Predicated statement

Figure 1: Missing control statement in LANCE

The ternary operator is the usual ternary operator of C. In the example of Figure 1(a) `a` is set to `b - 1` if `b` is greater than 10, to 4 otherwise. Note that the ternary operator is not strictly related to assignments, and it can be used stand-alone.

A predicated statement is composed by two parts: a statement and a guard. If the guard is evaluated to 0, then the statement is not executed, and vice-versa. In the example of Figure 1(b), the statement `a = a + 1` is executed only if `b` is not equal to 0.

You are required to modify the ACSE compiler to add syntactic support for recognizing the two kind of control statements and to generate the proper assembly code.

Back-end

Currently the ACSE compiler targets a virtual machine, MACE. In order to executed code generated by ACSE on a real machine, it must be emulated. You are required to write an ACSE back-end for the i386 [2] architecture, translating ACSE assembly files into i386 assembly files compatible with the GNU asm [4] syntax, without using any tools other than the ones presented during the classes.

The back-end must be a new ACSE tool, like `asm`, that can be used as starting point.

Creating and Applying Patches

The ACSE distribution is already provided as a mercurial repository. See [6] for the basic mercurial commands.

Assuming that your work has been committed with revision X, to generate a patch use the following command:

```
hg diff -r 0 -r X
```

Assuming the patch has been saved in the file `patch.diff`, use the following command to apply it to a freshly unpacked ACSE source tree:

```
hg import patch.diff -m "Applied patch"
```

The submitted project must successfully pass the above process, i.e. before submitting your patch check that it can be applied to a freshly unpacked ACSE source tree including compilation and testing process where applicable.

References

[1] Mercurial. http://mercurial.selenic.com, 2011.

[2] x86 Assembly. http://en.wikibooks.org/wiki/X86_Assembly, 2011.

[3] A. Di Biagio and G. Agosta. Advanced Compiler System for Education. http://compilergroup.elet.polimi.it, 2008.

[4] GNU. GNU Assembler. http://www.gnu.org/software/binutils/, 2011.

[5] Formal Languages and Compilers Group. Software Compilers. http://compilergroup.elet.polimi.it, 2010.

[6] J. Spolsky. Hg Init: a Mercurial Tutorial. http://hginit.com, 2011.