Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
30/09/2011

SURNAME:...
NAME:.. Student ID:............
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other:.....
Instructor: ○ Prof. Breveglieri ○ Prof. Crespi ○ Prof Morzenti

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

Modify the specification of the lexical analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle the di gestire the code execution driven by a the state of a bit of the program status word (PSW) via the `on <flag> run` construct, where `<flag>` $\in$ {`neg`, `zero`, `pos`, `notzero`}.

```
int a,b,c;
on zero run a=0;
c= 5 * a + 7;
on neg run {
    while(c>0){
        c = c-a;
    }
}
```

(a) Sample Code Snippet

Flag	PSW Bit
`nonzero`	Z clear
`zero`	Z set
`pos`	N clear
`neg`	N set

(b) Program Status Word States Reference

Figura 1: Example of a `on <flag> run` construct

The `on <flag> run` enables the execution of the code block following the keyword `run` only if the correspondent bit in the MACE CPU program status word is in the correct state. In particular, the correspondence between the value of `<flag>` and the program status word bit which allows its execution is reported in Table 1(b). `on <flag> run` constructs can be arbitrarily nested.

Explicit any further assumption which you assume necessary in order to complete the given specification.

[1]Time 45'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

1. Define the tokens (and the related declarations in **Acse.lex** e **Acse.y**). (3 points)

   The solution is in the attached patch.

2. Define the syntactic rules or the modifications required to the existing ones. (4 points)

   The solution is in the attached patch.

3. Define the semantic actions needed to implement the required functionality. (18 points)

   The solution is in the attached patch.

4. Given the code snippet in Figure 2:

```
c & a * b == d
```

Figura 2: Mixed Expression

Write down the syntactic tree generated during the parsing with the Bison grammar described in Acse.y *starting from the* `exp` *nonterminal.* (5 points)

The solution is in Figure 3.

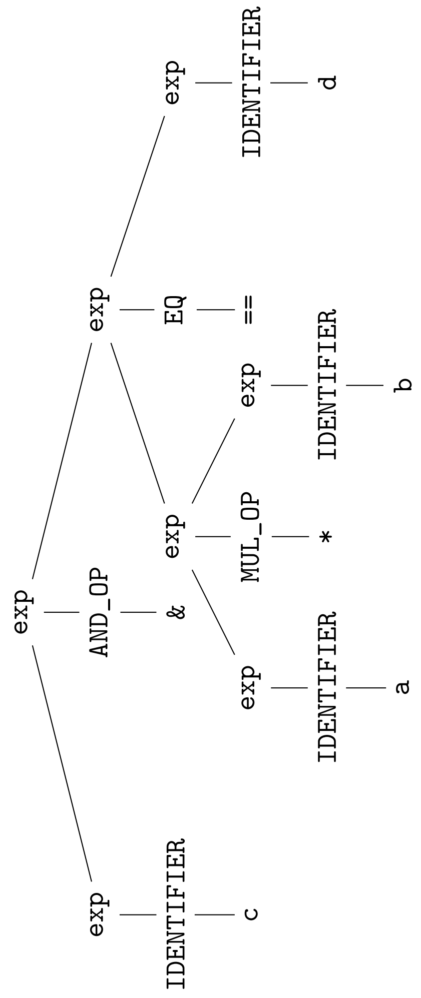

Figura 3: Syntactic tree of the statement in Figure 2

5. (Bonus) Assume to be willing to support an extension of the `on <flag> run` construct into `on (<flag1>,<flag2>) run`. This extended construct allow the execution of the code only if *both* the condition specified by the two `<flag>` tokens are true.
   Which modifications to the `Acse` compiler are due to support the aforementioned extended construct?
   How is it possible to produce optimized code in case there is an instance of `on (pos,neg) run` in the source code?.

Applicare una patch

Sul sito del corso è disponibile una patch contenente la soluzione del tema d'esame per quanto riguarda la modifica della macchina `Acse`.

Per applicare la patch:

1. scaricare la macchina `Acse` versione 1.1.0
2. scaricare la patch `soluzione-20-06-11.diff`
3. scompattare l'archivio contenente la macchina `Acse`
4. usando il terminale, portarsi nella directory in cui è stata estratta la macchina `Acse`
5. copiare in tale cartella la patch
6. applicare la patch tramite il comando

```
patch -p1 < soluzione-20-06-11.diff
```

La patch è un normalissimo file di testo, contenente le differenze tra la versione di `Acse` con implementata la soluzione dell'esame e la versione 1.1.0.

Le righe che iniziano con il carattere `+` sono state aggiunte, mentre quelle con il carattere `-` sono state rimosse.