CS2110 Fall 2011

Homework 7

Turn in this assignment by the date and time posted for it on T-Square

Rules and Regulations

Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course,** but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he turns it in as his own you will both be charged. We will be using automated code analysis and comparison tools to enforce these rules.
If you are caught you will receive a zero and will be reported to Dean of Students.

Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know ***IN ADVANCE*** of the due time supplying relevant documentation (i.e. note from the dean, doctor's note, etc.). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. No excuses, what you turn in is what we grade. In addition your assignment must be turned in on TSquare. When you submit the assignment you should get an email from T-Square telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over T-Square.

3. There is a random grace period added to all assignments and the TA who posts the assignment determines it. The grace period will last at least one hour and may be up to 6 hours and can end on a 5 minute interval; therefore, you are guaranteed to be able to submit your assignment before 12:55AM and you may have up to 5:55AM. As stated it

can end on a 5 minute interval so valid ending times are 1AM, 1:05AM, 1:10AM, etc. **Do not ask us what the grace period is we will not tell you**. So what you should take from this is not to start assignments on the last day and depend on this grace period past 12:55AM. There is also no late penalty for submitting within the grace period. If you can't submit your assignment on T-Square due to the grace period ending then you will receive a zero, no exceptions.

4. Although you may ask TAs for clarification but you are ultimately responsible for what you submit.

Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files.

2. In addition any code you write must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.

3. When preparing your submission you may either submit the files individually to T-Square (preferred) or you may submit an archive (zip or tar.gz only please) of the files.

4. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want.

5. Do not submit compiled files that is .obj files for assembly code and .o files for C code.

Overview

The Assignment

The overall objective of this assignment is for you to write a substantial assembly language program that includes writing recursive subroutines.

The particular program that you will write is what is known as a recursive descent parser. At the most basic level a parser is a program that can read some language and determine if the language read follows the rules of the grammar of the language. Parsers are at the heart of assemblers and compilers. In those applications parsers not only check that the language follows the rules of the grammar but can build a data structure that represents what the code is supposed to do. For this assignment the parser will be fairly simple and will simply tell you if the submitted language is legal per the grammar.

Note: The following information is supplied just to give you an idea of what is going on in this program. In reality we are supplying you with a working version of the program in C that you are to convert into assembly language.

Grammars and Languages

Before we can use a parser we must have some specification of the rules of the grammar we are parsing. For this we will use a notation known as Backus Naur Form. From Wikipedia:

BNF is an acronym for "Backus Naur Form". John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language (This was for the description of the ALGOL 60 programming language. To be precise, most of BNF was introduced by Backus in a report presented at an earlier UNESCO conference on ALGOL 58. Few read the report, but when Peter Naur read it he was surprised at some of the differences he found between his and Backus's interpretation of ALGOL 58. He decided that for the successor to ALGOL, all participants of the first design had come to recognize some weaknesses, should be given in a similar form so that all participants should be aware of what they were agreeing to. He made a few modifications that are almost universally used and drew up on his own the BNF for ALGOL 60 at the meeting where it was designed. Depending on how you attribute presenting it to the world, it was either by Backus in 59 or Naur in 60. (For more details on this period of programming languages history, see the introduction to Backus's Turing award article in Communications of the ACM, Vol. 21, No. 8, August 1978.)

Since then, almost every author of books on new programming languages used it to specify the syntax rules of the language.

Again, this assignment is not really about parsing but this is just to give you a peek into the box of material you will learn in later courses.

So, what does a grammar look like? Here is an example of a small subset of a grammar for English.

```
<sentence> ::= <noun-phrase><verb-phrase>
<noun-phrase> ::= < cmplx-noun> | < cmplx-noun><prep-phrase>
<verb-phrase> ::= < cmplx-verb> | < cmplx-verb><prep-phrase>
<prep-phrase> ::= <prep>< cmplx-noun>
<cmplx-noun> ::= <article><noun>
<cmplx-verb> ::= <verb> | <verb><noun-phrase>
<article> ::= a | the
<noun> ::= boy | girl | flower
<verb> ::= touches | likes | sees
<prep> ::= with
```

Symbols in blue are known as terminal symbols and are what you would expect to find in a sample of the actual language.

Important things to note: A grammar is not an algorithm. It is just a set of rules. You don't run it. The above grammar (used in conjunction with an appropriate parser) would recognize sentences like:

the boy likes the girl

The productions of the grammar would look like this:

```
<sentence>
<noun-phrase><verb-phrase>
<cmplx-noun><verb><noun-phrase>
<article><noun>likes<cmplx-noun>
the boy likes <article><noun>
the boy likes the girl
```

Our Grammar

The grammar we will be using is very simple. It looks like this:

```
<form> ::= <name> | N<form> | <BinaryOperator><form><form>
<BinaryOperator> ::= A | B | C | D
<name> ::= a | b | c | ... | x | y | z
```

Here are some examples of strings of symbols this grammar would recognize as valid well-formed formulas:

a
b
Nd
Bxy
CNdApq
DBpcCrNt

If this makes no sense at all imagine that the lower case letters represented Boolean variables and the upper case letters are logical operators like A-AND,B-OR, C-NAND, D-NOR and N-NOT so DBpcCrNt would be:

a	a
b	b
Nd	NOT d
Bxy	x OR y
CNdApq	(NOT d) NAND (p AND q)
DBpcCrNt	(p OR c) NOR (r NAND (NOT t))

Note: This example is just to give you an example that the grammar could represent. All we are concerned with is whether or not a given input string is valid or not per the grammar.

Language Description in English

In case you found all that symbolic manipulation too confusing!

The language you will be checking for has three parts:

1. The basic elements of the language are lowercase letters.

2. There are five more advanced elements in the language A, B, C, D, and N. Each of these is paired with one or more other elements, N is paired with one element, and the other four are paired with two.

3. The most advanced elements are sentences. A sentence is any one lower case letter, or A, B, C or D followed by two valid sentences, or N followed by one valid sentence. In the attached file “parser.c” you can see the exact program that you should implement. Convert parser.c to LC-3 assembly and put the assembly program in the file “parser.asm”.

4. On a Linux system you should be able to compile and run the parser by navigating to the directory where the parser.c file is located and typing:

```
gcc –o parser parser.c
```

then to run the program type:

```
./parser
```

A Few Requirements

1. Your code must assemble with NO WARNINGS

2. Comment your code! This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad left notes to let you know sections of code or certain instructions are contributing to the code. Comment things like what registers are being used for and what not so intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semi-colon (;), and the rest of that line will be a comment. Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

Good Comment
```
ADD R3, R3, -1 ;counter--
BRp LOOP ;if counter == 0 don't loop again
```

Bad Comment
```
ADD R3, R3, 1 ;Decrement R3
BRp LOOP ;Branch to LOOP if positive
```

3. Print your output to LC-3 console.

4. Get input from LC-3 console.

5. DO NOT assume that ANYTHING in the LC-3 is already zero. Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.

6. Test your assembly. Don't just assume it works and turn it in.

Deliverables

1. parser.asm