Cedar Assembler Manual

Petar Peychev
27 April 2019

Project manual, submitted as part of the Computer Technology and Mathematics ITEC10261 module, School of Science and Technology, Nottingham Trent University

Overview

My project consists of an advanced assembler, which takes as input an instruction set and a valid assembly language program, then outputs a Cedar Logic memory file (.cdm). It's written in C++ in an object-oriented style.

1. Features

The assembler implements the following features:

From the Project Suggestions document:

1. Read the instruction definitions from a file.
2. Use a class instead of a struct for the instructions – in other words adopt a proper "Object Oriented" approach to the program design and syntax.
3. Implement the directives "ORG", "DC" and "DS" in addition to the instructions.
4. Implement the "Label" concept within the assembly language.
5. Implement the "EQU" directive.
6. Compute the offsets for relative branches – allowing labels to be used to specify the jumps.*

*** Note:** The current method for differentiating between relatively addressed instructions and absolutely addressed instructions is by the address field size. 12 bits correspond to absolute addressing, while 8 and 4 bits to relative. Although this is not the suggested way, it works fine for the Cedar Logic processors built during this module and the change to a specification in the instruction set definition file would be trivial to implement.

Extra features:

1. Implementation of the "END" directive, which checks for insufficient address space to assemble the program.
2. The file paths to the Instruction Set, Input Program and Output File are all taken as command-line arguments to the assembler.
3. Resolution of nested "EQU" directives during assembly.

Ex:

a	EQU	b		a	EQU	FF0
b	EQU	c	**is resolved to**	b	EQU	FF0
c	EQU	FF0		c	EQU	FF0
	JMP	a			JMP	FF0

2. Usage Guide

Since the executable program requires command-line arguments, one must run it through either a CMD or PowerShell instance on Windows. It takes 3 arguments:

- Path to the instruction set.
- Path to the input program.

-Path to the output .cdm file. (If this file doesn't exist yet, it will be created.)

Example:

```
PS C:\Users\lemonade-win\Desktop\test> .\cedar-assembler.exe iset.txt prog.txt out.cdm
Assembly successful. File located at out.cdm
PS C:\Users\lemonade-win\Desktop\test> _
```

The instruction set is a list of colon-delimited pairs of instruction mnemonics on the left and opcodes on the right. Whitespace between the sections is ignored.

Example:

```
1  STA+    :  FFFE
2  STA     :  FFFF
3  BNZ     :  FD
4  BZ      :  FE
5  LDD     :  9
6  STORE   :  A
7  MOVE    :  B
```

The assembly language syntax consists of up to 3 whitespace-delimited tokens per line. Each non-empty line forms a statement in the grammar. Labels, hexadecimal values and the directives "ORG", "EQU", "DC", "DS", "END" are supported by default, while instruction mnemonics are dynamically added.

The "EQU" directive requires a label and operand to work, while all other directives require only an operand.

The "DC" directive takes as input a literal value and assigns it to a memory location.

The "DS" directive takes as input a decimal number and allocates the same number of words in memory for storage purposes.

3. Demonstration

In order to demonstrate the features of the assembler, I will show a program, which makes use of many of these features. The instruction set I will be using is the one from the Lab 7 processor:

```
1   STA+    :   FFFE
2   STA     :   FFFF
3   BNZ     :   FD
4   BZ      :   FE
5   LDD     :   9
6   STORE   :   A
7   MOVE    :   B
8   MOVEI   :   C
9   JMP     :   E
10  ADDI    :   D
```

The program, which I will be assembling is the following:

```
1   START   ORG     5
2   age     EQU     num
3   num     EQU     14
4   LABEL   MOVEI   4E2
5           ADDI    age
6           ORG     D
7           BNZ     label
8           DS      2
9           DC      AFD
10          END     FF
```

As you can see, it makes use of labels, the ORG, EQU, DS, DC and END directives, relative addressing, nested EQU's, as well as writing and reading from files.

We assemble the program over the given instruction set:

```
PS C:\Users\lemonade-win\Desktop\test> .\cedar-assembler.exe iset.txt prog.txt out.cdm
Assembly successful. File located at out.cdm
PS C:\Users\lemonade-win\Desktop\test>
```

And end up with the following .cdm file:

```
1   0005 : C4E2
2   0006 : D014
3   000D : FDF8
4   0010 : AFD
```

The beginning at address 0x5 and sudden jump to 0xD shows the function of ORG, the unassigned memory locations between 0xD and 0x10 demonstrate DS, the literal value at 0x10 shows DC, the address of BNZ on line 3 evaluating to F8 (-8) and pointing back to the beginning shows relative address labeling, the constant 0x14 operand on the second line shows nested EQU declarations and the succesfull assembly proves that the address counter hasn't exceeded the value specified by the END directive.