Computer Architecture

Project 1

Building Assembler

Dr. Chris Martinez

Spring 2017

Computer architecture is made up of two main components the Instruction Set Architecture (ISA) and the RTL model for the CPU. In this class, we will have two software projects that will help in your understanding of the ISA and RTL model for the CPU. The projects can be done in any software language that you chose to use.

The first project is the creation of an assembler. The assembler will be use for you to write assembly language programs that will be executed on the CPU you will design in project #2. The CPU we will be using is based on the DLX design.

**What is an assembler?**

All computer software goes through a set of applications to create an executable file that can be used by the end user. The first program a programmer will used is a high level language program (C, C++, etc.). The programmer writes in a very high level abstract environment writing code such as:

For(x=0; x< 10; x++)

A[x] = x;

The high level language allows the programmer to write his program without knowing how the ISA and hardware architecture functions. The language hides all the hardware from the programmer. At some point, the source code needs to be transfer to a language that takes into account the ISA and hardware architecture. The complier is the next step in the development process.

The complier will take in high-level source code and translate the program into assembly language. Assembly language is the ISA for a computer. Assembly is the instructions that the computer can perform. The compiler will do the following:

C language: c = a + b;

Assembly: load r1, a

Load r2,b

Add r3,r1,r2

Store r3,c

The last part of the development process is taking the assembly code created by the complier and transferring into machine code that the computer will execute. A computer is not able to run written language such as ADD r1,r2,r3. The computer runs a program by reading data stored in memory. The memory does not the store the command ADD r1,r2,r3 but store machine code. Machine code is the binary representation of an assembly language instruction.

**The DLX Machine code**

All the information for the DLX machine code and assembly instructions can be found in the handouts given in class. Each of the DLX instruction is 32-bits in length. Each part of the instruction is assigned a set of bits in the instruction. DLX break down the instructions into three different types R,I, and J.

R-type instruction

An example of the R-type instruction is ADD rd, rs1, rs2

The instruction is 32-bits in length:

Bits 31-26 is the opcode. The opcode tell the CPU what instruction it is doing

Bits 25-21 is the rs1

Bits 20-16 is the rs2

Bits 15-11 is the rd

Bits 10-0 is the funct. Funct tells what type of ALU operation is being conducted.

I-type instruction

Ex ADDI rs1,rs2,immm

Bits 31-26 is the opcode

Bits 25-21 is rs1

Bits 20-16 is rs2

Bits 15-0 is immediate. The immediate value is the constant value to use in the instruction

J-type instruction

Ex J address

Bits 31-26 is the opcode

Bits 25-0 is the address to jump to

**What is my assignment?**

Getting Started (Items you have to review or learn to do for the project)

In order to get started on creating your assembler you must be able to have the program understand basic syntax of a text file containing a program. In this first part of the program you will deal with the basic functions of string processing and reading files. The following is a list of items to complete to help you do the project.

1) You will need to be able to read in arguments pass from the command line.

Ex: project1.exe inputfile.txt outputfile1.txt

2) You will need to be able to parse a string.

Ex: You will need to use the strtok() function in the string.h file.

3) You will need to convert all letters to lowercase letters.

4) You will need to compare a string to a list of know words.

(add, load, store, r0, r1, r2)

5) You will need to have a number in text 123 be converted to a number (integer) in the C program.

6) You will also need to be able to read and write files.

Project #1

Your assignment is to make a simple assembler for the DLX processor. We will simplify the assembler by only using the core instructions, we will not do the floating point instructions. The registers will be named as r0, r1, r2, … r29, r30, r31. We will also not allow any advance naming for saving constant or memory locations the assembly source code we write will hard code any constant and memory location to read and write to. The only advance item we will do is use a label for branching and jumping in code.

Items to turn in: a report that tells how you created your program, source code, test files and a flow chart.

**Tips**

It is common to run an assembler in a two pass structure. The first pass goes through the source files and finds all the labels and addresses for each instruction. You can start your program by assuming the first line of code is at memory location 0. You will make a table of label and address. On the second pass you will then convert the assembly instruction into machine code and fill in any labels using your table.