COMP40 Assignment: A Universal Virtual Machine

Contents

1 Purpose and overview

The purpose of this assignment is to understand virtual-machine code (and by extension machine code) by writing a software implementation of a simple virtual machine. You will also have an opportunity to demonstrate your ability to create a design with a clean modular structure, to document that design, to choose suitable data structures and implementation techniques for that program, and to document those choices.

You will also begin to learn how the structural choices you make affect the performance of your programs. The primary goal of your design and implementation is clean structure, but there are stated performance goals you must achieve to receive credit.

2 Getting Started

You will be building an executable named `um`. This Linux program takes a single argument which is the pathname for a file. That file, typically with a name like `some_program.um` contains the machine instructions that your emulator is to execute. *When a UM program is stored in a file, words are stored using big-endian byte order.* Thus the high order four bits of the first byte of the file will be the first operation code for that program. As described below, `stdin` and `stdout` are used for the implementations of the UM `Input` and `Output` instructions respectively. Thus, a typical invocation of your emulator might look like:

```
um some_program.um < testinput.txt >output.txt
```

Your program is to emulate a UM, so most of the specifications for your program consist of the specifications of the UM itself. You will find these outlined in the section below. These specifications describe the structure of the UM (how many registers, etc.) as well as the operation of each UM instruction. Start by reading and making sure you understand the UM specifications below.

Then, before you design or code anything, read the remaining sections of these instructions. They include additional specifications that apply to your emulator, e.g. how fast it must execute UM programs. You will also find instructions for writing your design documents for building unit tests, and for submitting your designs, unit tests and working UM emulator. You will also find hints relating to the trickier aspects of implementing a 32 bit UM on a 64 bit AMD 64 Linux machine.

3 Specification of the Universal Machine

3.1 Machine State

The UM has these components:

- Eight general-purpose registers holding one word each
- A very large address space that is divided into an ever-changing collection of *memory segments*. Each segment contains a sequence of words, and each is referred to by a distinct 32-bit identifier. The memory is *segmented* and *word-oriented*; you cannot load a byte
- An I/O device capable of displaying ASCII characters and performing input and output of unsigned 8-bit characters

- A 32-bit program counter

One distinguished segment is referred to by the 32-bit identifier 0 and stores the *program*. This segment is called the '0' segment.

3.2 Notation

To describe the locations on the machine, we use the following notation:

- Registers are designated $r[0] through $r[7]
- The segment identified by the 32-bit number i is designated $m[$i$]. The '0' segment is designated $m[0].
- A word at offset n within segment i is designated $m[$i$][$n$]. You might refer to i as the *segment number* and n as the *address within the segment*.

3.3 Initial state

The UM is initialized by providing it with a *program*, which is a sequence of 32-bit words. Initially

- The '0' segment $m[0] contains the words of the program.
- A segment may be *mapped* or *unmapped*. Initially, $m[0] is mapped and all other segments are unmapped.
- All registers are zero.
- The program counter points to $m[0][0], i.e., the first word in the '0' segment.

3.4 Execution cycle

At each time step, an instruction is retrieved from the word in the 0 segment whose address is the program counter. The program counter is advanced to the next word, if any, and the instruction is then executed.

3.5 Instructions' coding and semantics

The Universal Machine recognizes 14 instructions. The instruction is coded by the four most significant bits of the instruction word. These bits are called the *opcode*.