KMIPS Local User's Guide

This document describes how to compile, write ASM code for, and run my ultra-simple MIPS assembler.

This document pertains to version 0.2.0 and supersedes any previous document.

Compilation

KMIPS is written in standard C++. It was developed on a Mac using Apple's LLVM version 10.0.0, but should work fine with gcc 4.8.5. My guess is that it will work on any compiler that comes within spitting distance of the C++ standard.

Download the file and place it in an appropriate directory. (It will need to be in the same directory as the assembly language files you want to write.

To compile, use:

```
$ g++ -o kmips kmips.cpp
```

Usage

If you have compiled as directed above, invoke the program as:

```
$ ./kmips asmfile.asm
```

KMIPS doesn't care about filenames or extensions, so the input file can be called anything you want.

The assembler follows pretty common assembly language conventions:

- Comments start with a # character.
- Lines with executable code must have a space in the first column.
- Instruction mnemonics (e.g., `ADD`) can be entered in all uppercase or all lowercase, but not mixed case (e.g., `AdD`).
- Register names are of the form `$`*r*, where *r* is a number between 0-31.

The output filename is always called `a.out`. KMIPS will happily clobber any existing files with that name, so beware.

The Output

The output is formatted as follows:

```
// Original line of code
hexadecimal machine code
```

For example:

```
//  sw $6, -10($2)
ac46fff6
```

This file is intended to be read in by your processor using the `$readmemh()` Verilog task. Be sure that your instruction memory module has an `initial` block that reads it in appropriately.

For example:

```
module InstructionMemory(input [31:0] A,
                         output reg [31:0] RD);
    reg [31:0] instMemory [0:1023];
    initial
    begin
        $readmemh(“a.out”, instMemory);
    end

    // Insert behavioral code here
endmodule
```

Sample Program

This program does nothing but test out a variety of instructions mnemonics. It does not *do* anything meaningful.

```
# A simple MIPS assembly language file
 lw $1, 0($0)
 add $3, $2, $1
 sw $6, 10($2)
 beq $9, $8, 4
```

Limitations

There aren’t many, but they are quite significant.

- MIPS provides lots of special names for registers (e.g., `$t1`). These are not supported.
- There is no support for labels.
- There is no support for hexadecimal constants.
- Only a subset of the MIPS ISA is supported (particularly those needed by the processor project). The following instructions can be used:

Instruction Type	Supported Instructions
R-type	`add, and, or, slt, sub`
I-type	`lw, sw, addiu, andi, ori, siti`
Branch	`beq, bne`
J-type	`j`

Changes

0.2.0 Added in support for the final set of instructions and made the printouts a little prettier.

0.1.3 Since the last update, the `addi` and `j` instructions are supported, as well as negative constants.

Bugs

I’m sure there are many. The core of this program was written in about 4-5 hours with extremely rusty C++ skills. There is a 5-point bug bounty for the first reported instance of a

bug. Just email cmcischk@mtu.edu and I'll let you know if you get the bounty. A bounty will not be provided for explicitly listed limitations.