CS/COE 447 Computer Organization
Spring 2012

JrMIPS Assembler Reference Manual
April 1, 2012 (ISA version 0.4)

This document describes the rudimentary assembler for JrMIPS. The assembler is available on the CS/COE 447 web site.

The assembler uses Perl. You'll need access to Perl to use the assembler. You can get Perl from: http://www.perl.com/download.csp

1) Assembly Language Syntax

The JrMIPS assembler supports the instructions and assembly language syntax:

```
and   $rs,$rt
nor   $rs,$rt
add   $rs,$rt
addi  $rs,imm
addi  $rs,dlabel  (this is equivalent to la below)
sub   $rs,$rt
sll   $rs,shamt
srl   $rs,shamt
sllv  $rs,$rt
srlv  $rs,$rt
lw    $rs,$r1
sw    $rs,$r1
bp    $rs,label
bn    $rs,label
bz    $rs,label
bx    $rs,label
jal   $rt,label
jr    $rs
j     label
not   $rd,$rs  (pseudo instruction)
or    $rd,$rs,$rt  (pseudo instruction)
clr   $rs  (pseudo instruction: set $rs to 0)
li    $rs,imm  (load immediate into $rs)
la    $rs,dlabel  (load address of data labl dlabel into $rs)
put   $rs
halt
```

where, `$rs` and `$rt` are one of names `$r0`, `$r1`, `$r2`, `$r3`, `$r4`, `$r5`, `$r6`, `$r7`;
`shamt` is an unsigned value;
`label` is a symbolic label name declared in the text segment;

`dlabel` is a symbolic label name declared in the data segment; and,
`imm` is a signed constant decimal or hexadecimal value.

The assembler supports hexadecimal notation for values (immediates). To indicate hexadecimal, use "0x" before the value (e.g., 0xF00F is valid). A hex value may only be used in a location where an immediate value is expected, including instructions with an immediate (the last operand) or labels in the data segment.

Labels are allowed, but they must start with a letter and end with a colon. Labels are case insensitive. For example, the label `LABEL0` is the same as `Label0`.

The JrMIPS assembler supports a text and data segment. For the data segment, use the directive ".data" by itself on a line to begin the data segment. In this segment, you can declare initialized 16-bit word values. Each value may have a label. The syntax is:

label: value

where *label* is a symbolic name and *value* is the value to associate with the label. Unlike MIPS, you do not declare a type. All items in the data segment are 16-bit words. Every value must be declared on a line by itself with its own label. Hence, you cannot have a sequence of values separated by commas.

The data in the data segment must be processed and loaded separately from the text segment. The assembler option "-d" will output the data values in a form that can be loaded into a Logisim RAM component.

The text segment is the default. If you use a data segment, be sure to switch to the text segment with the ".text" directive.

Comments and blank lines are allowed. A comment is given with a semicolon (";").

There is almost **no error checking** by the assembler, so tread carefully.

2) Assembling a Program

To run the assembler, use the command:

```
perl jrmipsasm.pl progname.asm
```

This will cause `progname.asm` to be assembled. The assembler will output hex encoding for the instructions to the display. The output is in a format that can be loaded directly from within Logisim into a ROM component.

To save the output from the assembler to a file, use the command:

```
perl jrmipsasm.pl progname.asm > progname.dat
```

Then, load `progname.dat` into ROM from Logisim. To load a program into ROM, open your processor design in Logisim, poke the ROM, and click on the ROM's Load contents attribute. Select the filename with the encoded program.

To output the data segment, use:

```
perl jrmipsasm.pl –d progname.asm > progdata.dat
```

Then, load `progdata.dat` into RAM. Click on the RAM and press Control (control-click). This will bring up a menu option "Load Image", which will set the RAM to the contents in the file. You need to reload the RAM every time you reset the simulation.

The assembler supports some command line options.

An option ("-p") disables all pseudo-instructions.

A second option ("-v") causes the assembler to print verbose information about the assembled program. This option is useful to verify that the assembler generated the correct output.

A third option ("-l") will cause the assembler to dump the addresses it used for the labels in the input assembly language program.

A final option ("-d") causes the assembler to output data segment information.

Appendix: Example Programs

Here is an example program:

```
            clr   $r0
            # increment from 0 to 15
            # lowest hex digit will cycle from '0' to 'F'
loop0:      put   $r0,0       # output current value
            addi  $r0,1       # increment value by 1
            mov   $r1,$r0     # check for loop end
            addi  $r1,-16     # end of loop?
            bn    $r1,loop0
            halt
```

The output from the assembler (without any command line options) is:

```
v2.0 raw
# to load this file into Logisim:
# 1) save the output from the assembler to a file
# 2) use poke tool and control-click the ROM/RAM component
# 3) select Load Image menu option
# 4) load the saved file
2000
E000
1001
2240
0200
12F0
9201
F000
```

This is a second program with a data section:

```
      .data
a:    10
b:    0
      .text
      la $r0,a
      lw $r1,$r0
loop: add $r2,$r1
      addi $r1,-1
      bp $r1,loop
      la $r0,b
      sw $r2,$r0
      lw $r3,$r0
      put $r3,0  # answer should be 37h
      halt
```

The assembled instruction file is:

```
v2.0 raw
# to load this file into Logisim:
# 1) save the output from the assembler to a file
# 2) use the poke tool and control-click the ROM/RAM component
# 3) select Load Image menu option
# 4) load the saved file
2000
1000
B200
0440
12FF
8203
2000
1001
A400
B600
E600
F000
```

The data section file (loaded into the RAM) is:

```
v2.0 raw
# to load this file into Logisim:
# 1) save the output from the assembler to a file
# 2) use the poke tool and control-click the ROM/RAM component
# 3) select Load Image menu option
# 4) load the saved file
000A
0000
```