RAM and ROM tools

LogiSim File Format

For burning memory, LogiSim has its own file format. However, it is very similar to an ASCII text file expressing pure hex output. The first line contains the string "v2.0 raw"---this is the type of file format. It is the only format accepted by LogiSim. The rest of the file consists of hex values separated by whitespace, with one additional feature: you can write e.g. k* to denote k copies of the memory value So, for instance:

```
v2.0 raw
ff 12*00 ff
```

represents a byte of all ones, twelve bytes of zero, followed by a byte of all ones. You can place hex values on a line, separated by spaces. You can also just put one value per line.

Burning ROM

- First, save your LogiSim work. (The burning tool occasionally crashes.)
- Right-click on the ROM you want to burn, and select "Load Image..."

Loading RAM

Assuming you've already created your RAM device and placed it in your circuit

- Select the RAM device you want to load
- Right-click on the device and select "Load Image..."
- It will then prompt you for an input file. It must be in the format described above.

When Burning Code...

Keep in mind that a ROM or RAM device, by itself, does not know about the larger address space of the machine. That is: the zeroth word you load will live in the zeroth word in the device... but if you want this to live at 0x00003000 in the Y86 address space, you need to wire the device into the system that way.

Your File Formats

What you want to burn into a memory device may come into two file formats:

- **"Object," or "yo"** The output of the Y86 assembler: an ascii expression of the sequence of bytes living in the address space.
- **"FSM."** You might also want to create a spreadsheet to work out the FSM control ROM for you Y86. Each row would be an entry in the ROM; Each data cell has a 1, 0, or "x" (for "don't care").
  You might feel compelled to select the data cells in your spreadsheet, then cut-and-paste into a text file. We call the resulting file "FSM format."

Converting Between Formats

To make all our lives easier, we provide tools to convert between formats:

- **yo2hex.c**, to convert from "object" format to LogiSim's hex format.
- **fsm2hex.c**, to convert from "FSM" format to LogiSim's hex format. (If your rowsize is not a multiple of 4 bits, fsm2hex will silently pad it out on the right with zeros.)

These all live at **romtools.taz (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/y86/romtools.taz)**

We've standardized the invocation. Each program must have two arguments:

- the name of the input file---or just "-" if you want use stdin
- the name of the output file---or just "-" if you want use stdout

This lets you do things like pipe the output of one program into another.

yo2hex can take two additional optional arguments:

- "-b NNNN": begin with the byte at hex address NNNN
- "-e NNNN": end with the byte at hex address NNNN

(That is.... the object file specifies a sequence of bytes living in the address space; you can use -b and -e to generate a LogiSim hex file containing just those bytes in a specific region.)

That Darn Fake Harvard Architecture

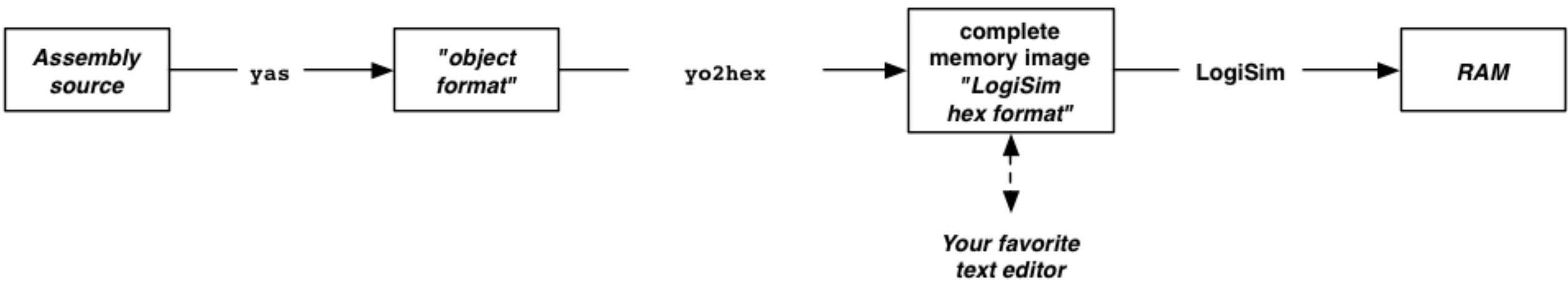
As I will lament in class... even though the Von Neumann architecture won out, the textbook develops a datapath where the instruction memory is distinct from the data memory.

If you do things that way, then nothing in your instruction memory can be referenced by a data memory instruction (e.g., mrmovl). Which is annoying!

If you're using this Fake Harvard approach, you might want to write your assembly code so that anything you want to use as data starts at some well-defined place after the instructions (e.g., with a .pos directive)---and then use the yo2hex -b and -e options to split the .yo file into the piece for the instruction memory and the piece for the data memory.

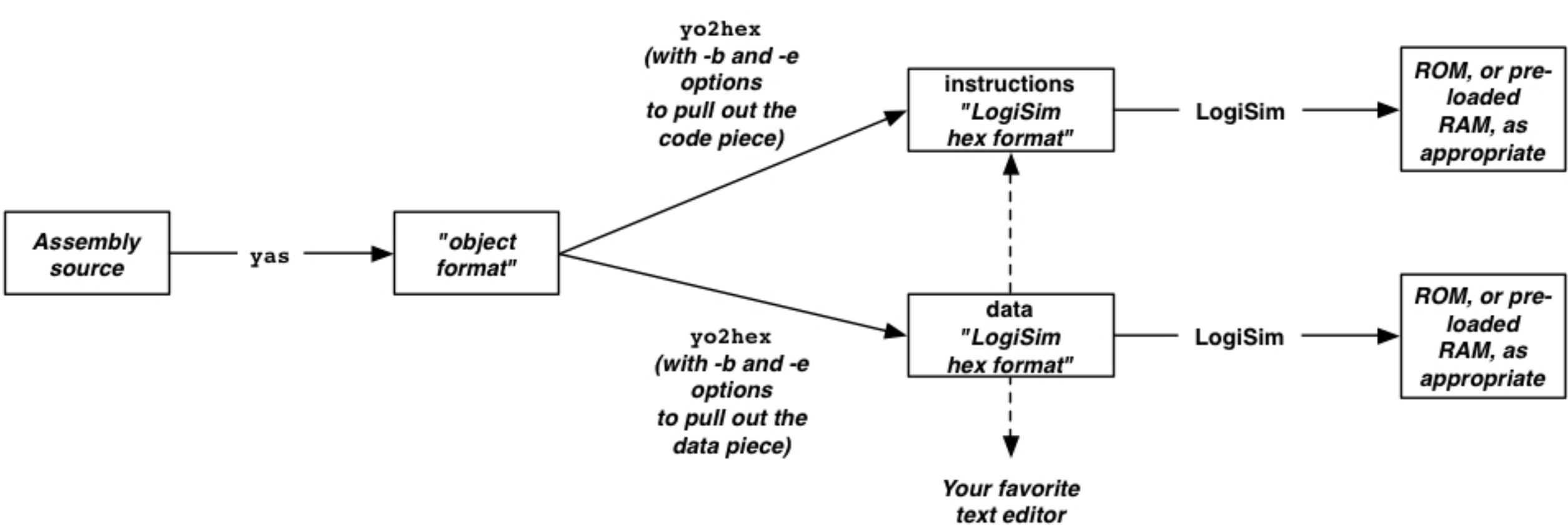
Putting it All Together

For a true Von Neumann Datapath



For a Fake Harvard Datapath

(Or if you just want code to be in a separate memory component)



For Microcode