Delphi.jr User Manual.

Introduction:

Thank you for going through my project. This guide is aimed to give a quick overview of how the delphi.jr (name of processor) works and an example code for simple addition.

To state technically, dephi is a very simple processor with no pipelining, no interrupt controller, no cache or cores, no MMU or memory protection and no peripheral interface. It is a simple von Neumann machine with four 8-bit registers and 8-bit address bus and 20-bit data bus. It runs on its own ISA which is also present in the same folder. The actual 20-bit control word is compressed to 8 bits using a ROM. The ISA sheet depicts the compressions.

Motivation for the ISA comes from the x86 architecture. With one control word and two/operands. The key difference is that this architecture relies on "Selection of operand type" ie , the type of operands must be specified before they are to be used using SEL instruction and three basic types called reg,mem and imm8 are recognized as "register",memory" and "immediate 8 bit" operands.

Looping and conditional jumps are implemented directly by modifying the program counter (PC).

You may need to install logisim simulator before you proceed. Simply double click the .circ file once logisim is installed.

Once opened, you will see wiring of different modules which in turn can be viewed from the right pane. I intend to just demonstrate the sample code of adding two numbers on this machine, though it can perform other tasks in the range of its ISA as well.

Firstly, delphi works in two modes.

* A real time mode like a calculator. Just put the operands and give the appropriate control word in "control\_signal" box, the result will be displayed on output probes. This mode is chosen when mode\_h/w button is in '0'.
* A programmable mode where you can write code. In this case, all the 20-bit control instructions must be written in "input\_to\_ram" probe while the "ready" is '0' and "mode\_h/w button" is '1'.

The instruction is of 20-bits long and,

First 8-bits [19:12] is the control instruction. Referred to from ISA sheet

Next 4-bits [11:8] is the ALU opcode. You can set it to zero if you don't want to use ALU. (in case of copying data in/out of reg, mem, etc.)

Last 8 –bits are [7:0] is for the operand.

Now, we shall write code to add 2 and 1. the result 3 will be seen on appropriate register. Set the clock speed to 1 Hz or lower for better visualization of the result.

Please program the first word as no-op (0x00000). It is just a convention.

Also, before you begin, set the 5th and 6th bits of "Control Signal" to '1'. They are essential since 5th bit is write enable(write\_enable) to RAM and 6th tells that the word stored is a control word. Not data. (Write\_type) . Refer to ISA for more info.

We shall do this in 3 steps.

* Copying operands to respective registers. In this case, copying '2' to register r0 gives the control instruction as,

0x20 -> machine instr for moving data to r0 reg (MOVR r0,imm8)

+ 0x0 -> we are not using ALU. So,

+ 0x02 -> immediate 8-bit operand. So, the total control word will be,

**0x20002**

* Similarly, for copying '1' in r1, **0x21001**
* Opcode for adding the arguments is 0x3.
* So, for adding, we need to select the operands. Here in this case, they are r0,r1. So,

0x10 -> SEL r0,r1 (note that sel r0,r1 and sel r1,r0 are different because, the result is written back to source register (same as x86).

+ 0x3 -> using ALU for addition

+ 0x00 -> we are not using immediate operands. So total control word will be,

**0x10300**

After programming, the result will be visible on output probes.

The processor can do other things too. Also, since the project was a casual one, it has been poorly documented and maintained. If you have any queries, please feel free to ask or request demos.

THANKS.!