COMP22111: Processor Microarchitecture

Exercise 3: Stump Datapath

In this exercise you will be required to produce a module description for the Stump datapath following on from the RTL description we developed in the lectures (see Figure 1). One approach could be to produce a schematic design of the Stump datapath, much like you would have done in COMP12111, this would be fine but with today's powerful CAD tools there's no need. A more attractive alternative is to represent the ***structure*** of the datapath in Verilog. Both approaches are valid and if designed correctly should enable a working, synthesizable design to be produced. In this exercise you are required to implement the Stump datapath using a *structural* Verilog representation.

A module, Stump_datapath.v, has been produced for you with the interface as defined in Figure 2; you should not make any changes to this interface. As the module is constructed using structural Verilog than all outputs and internal variables are defined as `wire` (the default type) – which I have chosen not to show here. Note: the signals `srC` and `regC` are not part of the Stump functional design; they are included in the register bank module so that Perentie can monitor the contents of the register bank for debugging purposes.

When completing your design use the net names illustrated in the RTL datapath in Figure 1 to maintain readability. A number of modules have been provided for you, an example is the Stump register bank, where a module Stump_registers.v has been provided for you with the interface defined in Figure 3. You can see the full list of modules in the Cadence NC window.

We can instantiate the Stump_registers module within our datapath as shown in Figure 4. Here we have called the instantiation `regbank` for clarity, but you can call your modules whatever you like, just make sure you use sensible, meaningful names. We have instantiated this for you in Stump_datapath.v, along with some of the internal variables. You will need to declare any other internal signals that you use.

Complete the structural Verilog representation of the Stump RTL datapath. Make sure you include all the functional elements of the datapath in your structural representation.

Testing

Proper testing of your Stump datapath design would require a complex testing suite comprised of a model control system and memory, which is beyond the scope of this course unit. Consequently, the only testing you can perform at this stage is to inspect your design to ensure you are confident that instantiated modules are connected correctly. TAs have access to a test script that will be used to verify your design during the demonstration.

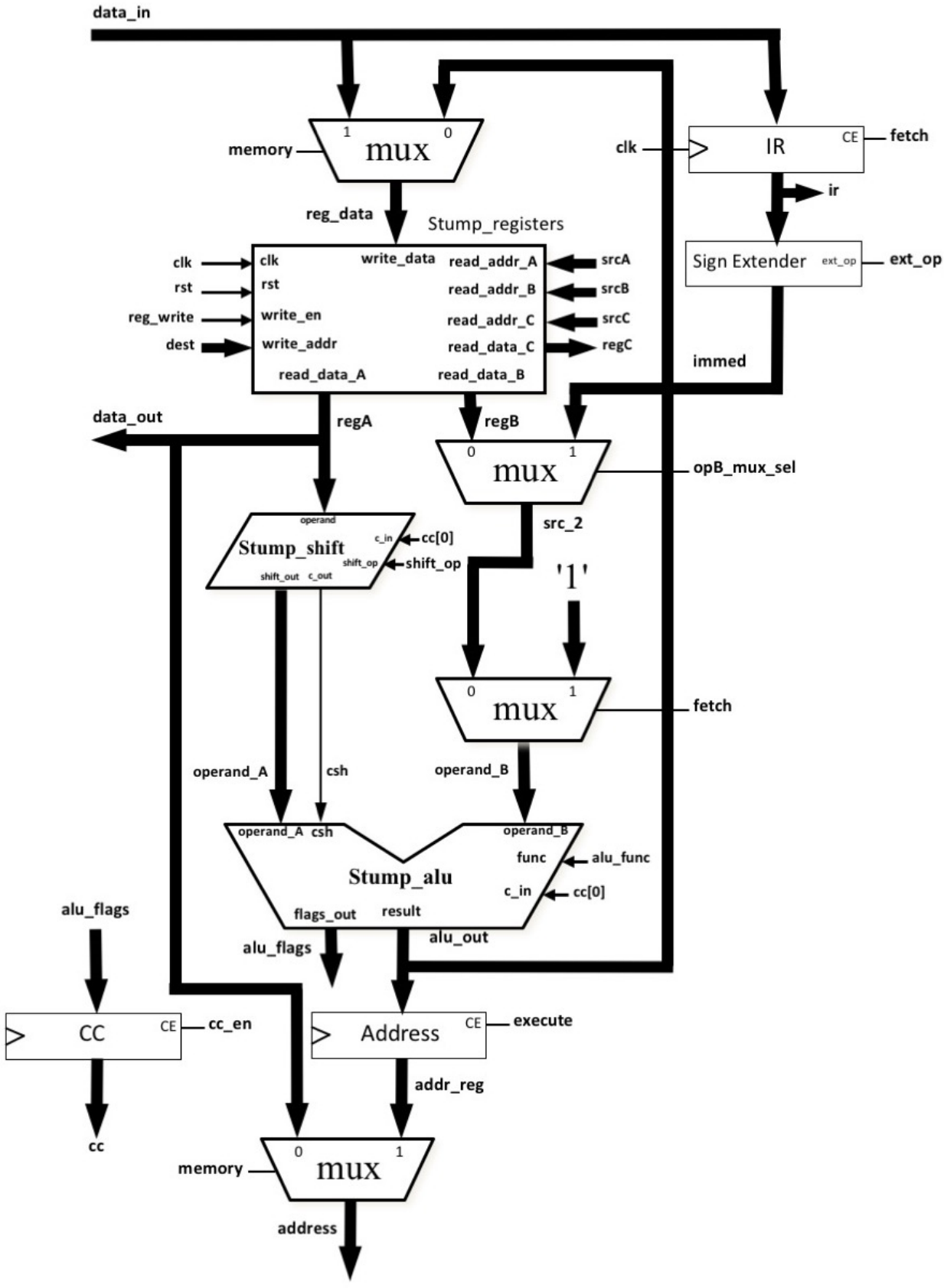

Figure 1: RTL datapath of the Stump Processor

```
module Stump_datapath (
    input           clk,            // System clock
    input           rst,            // Master reset
    input   [15:0] data_in,         // Data from memory
    input           fetch,          // State from control
    input           execute,        // State from control
    input           memory,         // State from control
    input           ext_op,         // sign extender control
    input           opb_mux_sel,    // src_B mux control
    input   [ 1:0] shift_op,        // shift operation
    input   [ 2:0] alu_func,        // ALU function
    input           cc_en,          // Status register enable
    input           reg_write,      // Register bank write
    input   [ 2:0] dest,            // Register bank dest reg
    input   [ 2:0] srcA,            // Source A from reg bank
    input   [ 2:0] srcB,            // Source B from reg bank
    input   [ 2:0] srcC,            // Used by Perentie
    output  [15:0] ir,              // IR contents for control
    output  [15:0] data_out,        // Data to memory
    output  [15:0] address,         // Address
    output  [15:0] regC,            // Used by Perentie
    output  [ 3:0] cc);             // Flags
```

Figure 2: Stump datapath module header

```
module Stump_registers (
    input           clk,            // System clock
    input           rst,            // Master reset
    input           write_en,       // Write enable
    input   [ 2:0]  write_addr,     // dest
    input   [15:0]  write_data,     // Data in
    input   [ 2:0]  read_addr_A,    // src_A
    output  [15:0]  read_data_A,    // Operand A
    input   [ 2:0]  read_addr_B,    // src_B
    output  [15:0]  read_data_B,    // Operand B
    input   [ 2:0]  read_addr_C,    // src_C
    output  [15:0]  read_data_C);   // Operand C
```

Figure 3: Definition of the Stump register bank

```
Stump_registers regbank   (.clk(clk),
                           .rst(rst),
                           .write_en(reg_write),
                           .write_addr(dest),
                           .write_data(reg_data),
                           .read_addr_A(srcA),
                           .read_data_A(regA),
                           .read_addr_B(srcB),
                           .read_data_B(regB),
                           .read_addr_C(srcC), //Debug – Perentie
                           .read_data_C(regC));
```

Figure 4: Instantiating Stump_resgisters.v

Submission: ex3 – Stump datapath

Once you are happy that your design is correct then you can submit your work using the submit script "22111submit" before the deadline for the exercise. Submit as ex3.

Print out a copy of the labprint for the exercise via the submit script and answer any questions before you demonstrate the exercise in your next scheduled lab..

You have now completed Exercise 3