ARMv8 – ENGR 4395

Version 1.0

Randy Cornell, James Hartmann, Austin Krogman,

Jose Lacavex, Michael LePere, Robert Stanton

**Overview**

The current iteration of this design follows a 5-stage pipeline dataflow made up of an: Instruction Fetch Stage, Decode Stage, Execute Stage, Memory Stage, and Write Back Stage. Each stage is controlled by one overall clock and can pass data of R-type commands with write back capabilities into a General-Purpose Register. This design is structured in a "Pipeline”, waveform outputs show data being moved from one stage to the next with each positive edge of a clock cycle, and has only been verified using a multistage approach using known instructions and values initialized in the General Purpose Register of the Decode stage. Following the waveform, it can be seen that any R-Type command from the ARMv8 Green Sheet[1] can pass through with the result being written back in the destination address of the General Purpose Register.

The simulation software used to verify the current design was iVerilog, using Powershell and GTKWave for overall verification of dataflow and checking each stage latching the data with each positive edge clock cycle. Verification that the General-Purpose Register was storing the correct result from the instruction was done using Vivado, by created virtual buses in the waveform to follow the data.

**Instruction Fetch Stage**

This is a 2 port module with a clock (**clk**) input and instruction (I**F\_ID[63:0]**) output (IF\_ID refers to the register bank between the instruction fetch and instruction decode stage). The module consists of a general memory unit that is 32-bits wide and can hold 64 individual instructions that can be initialized within the module with instructions for testing.

The program counter performs addition using the arithmetic operator plus (**+**) for a behavioral style, whose value is then passed along to the instruction memory as the address of the instruction to fetch. This coding style will simulate the necessary function, but may not be synthesizable depending on which software one uses to synthesize and implement the module.

The output of the module is a 64-bit long register called **IF\_ID[63:0]**, where IF[31:0] is reserved for the instruction to be queued for the next stage and IF\_IF[63:32] is reserved for the current state of the program counter.

**Decode Stage**

This is a 4 port module with 3 inputs - clock (**clk**), write back (**WB\_ID[37:0]**) and **IF\_ID[63:0]** - and one output, **ID\_EX[156:0]**, with IF\_ID being the instruction from the previous stage, WB\_ID being the input from the write back stage and ID\_EX being the decoded data from the instruction. This module consists of a control unit, general purpose register, and a sign extension unit.

The general purpose register is designed with 32 induvial registers that are 32-bit wide. The memory unit uses IF\_ID[20:16] and IF\_ID[9:5] for selecting the address for R[m] and R[n], respectively. Register 31 is hard coded and reserved for the value zero necessary for the architecture. This general-purpose register has a write enable capability for the return value from the write back stage. Data from address R[n] is placed into ID\_EX[116:85] and R[m] is placed into ID\_EX[84:53].

The sign extension unit is currently working, but not processing the correct data. This unit uses the Replicator Operation (**{{ }}**) to take IF\_ID[21:10] and repeat IF\_ID[21] to increase the IF\_ID[21:10] to 32-bits and place the value in ID\_EX[52:21].

The control unit uses case statements to take IF\_ID[31:21] and populates ID\_EX[156:149] with the necessary bits for the next stage correlating the desired instruction. ID\_EX[156:155] are reserved for commands using the General Purpose Register, ID\_EX[154:152] are reserved for any memory or branch instructions, and ID\_EX[151:149] are reserved for ALU control functions needed for the intended command.

Following the Green Sheet, the destination address IF\_ID[4:0] passes along to ID\_EX[4:0] and IF\_ID[9:5] (shift amount from R-Type command) is passed along to ID\_EX[9:5].

**Execute Stage**

This is a 3 port module with 2 inputs – clock (**clk**) and **ID\_EX[156:0]** - and one output – **EX\_MEM[106:0]**, with IF\_ID being the output from the previous stage and EX\_MEM being the output of the processed data. This module currently consists of an ALU and a control unit for determining which operation to perform.

The ALU is made up of five individual modules – bitwise XOR, AND, OR, a Ripple Carry Adder with Subtract, and a Barrel Shifter with Left and Right options. The ALU currently will take ID\_EX[116:85] and ID\_EX[84:53] for all R-Type commands, and will only use ID\_EX[84:53] and ID\_EX[9:5] for shifting. The output for the ALU is controlled by four 2-1 MUX's with the control signals below:

|  |  |
| --- | --- |
| Operation | Bits |
| ADD | 0000 |
| SUB | 0001 |
| OR | 0010 |
| XOR | 0011 |
| LSL | 0100 |
| LSR | 0101 |
| AND | 1000 |

This output is then stored in EX\_MEM[68:37] for use in next stage.

The ALU control unit has two inputs – ID\_EX[151:150] and ID\_EX[20:10] - using case statements, it enables the correct operation needed dependent on ID\_EX[151:150] and ID\_EX[20:10] for R-Type commands currently.

ID\_EX[4:0] is passed along to EX\_MEM[4:0], for destination address. ID\_EX[84:53] is passed to EX\_MEM[36:5] for memory address calculation and ID\_EX[156:152] is passed to EX\_MEM[106:102] for write back and memory access control.

**Memory Stage**

This is a 3 port module that has 2 inputs a clock (**clk**) and **EX\_MEM[106:0]** and one output **MEM\_WB[70:0].** EX\_MEM being the output from the previous stage and MEM\_WB will be the beginning input of the next stage. This module is currently made up of a 32-bit wide data memory unit that can store 1024 words (cache). This cache is currently a temporary placeholder memory structure to implement an instruction and hold memory value for test purposes.

EX\_MEM[106:105] passes through and is stored in MEM\_WB[70:69], these bits enable write to GPR and selects which data to pass through from either memory or ALU result in write back stage, respectively. The EX\_MEM[4:0] bits are passed through this stage and stored in MEM\_WB[4:0], for return data address destination. EX\_MEM[68:37] is the ALU result bits and they are passed through and stored in MEM\_WB[36:5] When EX\_MEM[103] is one (1), load word is enabled and the value stored in EX\_MEM[68:37] determines the address that data will be read from in memory and continue to the writeback stage. If EX\_MEM[102] is one (1) then store word is enabled, and EX\_MEM[68:37] determined the address that EX\_MEM[36:5] (ALU result) will be stored within the cache and no data will output to the writeback stage.

There currently is code for a direct mapped cache memory unit, but it has yet to be implemented into the Memory Stage. The direct mapping cache module itself has been verified, but when installed in the Memory Stage module, will need to be verified individually before being implemented into entire design.

**Write Back Stage**

This is a 3 port module that has 2 inputs – a clock (**clk**) and **MEM\_WB[70:0]** - and one output **WB\_ID[37:0]**, with MEM\_WB being the output from the previous stage and WB\_ID is feedback in to the Decoder Stage. This module is made up of a MUX that will pass either MEM\_WB[68:37] if MEM\_WB[69] is one (1), or MEM\_WB[36:5] if MEM\_WB[69] is zero (0) to WB\_ID[36:5].

MEM\_WB[4:0] is passed along to WB\_ID[4:0] for destination address and MEM\_WB[70] is passed along to WB\_ID[37] for write enable to the General Purpose Register in the Decode Stage.

**Testbench**

The overall design consists of five individual modules named after their respective function. The files for the ALU will need to be included with the complete design when compiled. A testbench has been included with all code and has been wired accordingly, with the output of one stage being the input to the next stage connected by wires with the Write Back Stage feeding back into the Decode Stage. The overall design has only one input, **clk**, and is toggled every 10ns currently. The General-Purpose Register has some values initialized directly in the module for testing purposes and the Instruction Memory unit has 32-bit instructions initialized as well. All files have been able to compile in iVerilog, ModelSIM, Vivado for simulation testing.

**Reference:**

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design ARM Edition: The Hardware Software Interface*, 1 edition. Amsterdam ; Boston: Morgan Kaufmann, 2016.