University of Tromsø

INF-2200

Assignment 2 - MIPS

Vebjørn Haugland and Alexander Saaby Einshøj

Department of Computer Science

October 14, 2015

1 Introduction

This paper describes the design, implementation and understanding of a subset of the MIPS architecture with all of its instructions.

1.1 Requirements

Implementing a pipelined binary code simulator for a subset of the MIPS architecture.

2 Technical Background

In order to complete the assignment the most important thing was to read up on the MIPS architecture and its fields of application. To be able to implement the simulator you had to have the basic knowledge about: MIPS functional units, MIPS control elements, pipelining, data hazards and control hazards.

2.1 MIPS functional units

The MIPS functional units are the datapath elements that build up the processor. These units will be connected to other units within the processor through input, output, control signals and outputted control signals. In some way all of these elements will process or manipulate the input they receive in order to communicate and give the proper input to the next unit "in the line".

2.1.1 The program counter

The program counter is a register that holds the address of the next instruction. It has one input, which is the next address to be issued to the simulator. It also has a output which is the address that it holds, this output goes to the instruction memory.

2.1.2 The instruction memory

The instruction memory takes one input from the program counter, as mentioned, and this is the name of the address mapped in a memory. The instruction memory will retrieve the value stored on the address in the memory and this value is the instruction of the simulator.

2.1.3 The register file

The register file takes in four inputs, two to read registers, one to write to register and one to write data. Two of these inputs comes directly from the instruction memory, and two comes from multiplexors within the simulator. It also takes in a control signal from the control unit, this signal tells if the register file is to write to the register as well as reading from it. The register file has two outputs that reads data.

2.1.4 The ALU

The ALU is the datapath element that performs logical operations on its input. It has two inputs and the ALU will, based on its control input, perform different logical operations upon these two inputs. This logical operation will create the output from the unit, and in addition it will send a control signal.

2.1.5 The data memory

The data memory will take as input the result from the ALU which is the address and a read data value which will be taken in as a write data from the register file. It will in addition take in two control signals from the control unit. If the value the data memory fetches is both written and read, or just read is based upon the control signals. The value is then sent as output.

2.2 MIPS control elements

Some of the functional units listed above, needs control signals as input in order to "know" which instruction or manipulation to perform. This can for instance be the Arithmetic Logical Unit, hereby referred to as the ALU. The ALU needs a control input from a controller in order to know which logical operation to perform upon the given input, this can i.e. be the logical operations "add", "subtract" or "AND".
In a sentence; the control elements gives control input to other units within the simulator as a indication or an "advisement" to what to be done.

2.3 Pipelining

Pipelining[1] is an implementation technique in which multiple instructions are overlapped in execution. To explain the pipelining term we will use the laundry analogy from the book:
1. You place one dirty load of clothes in the washer.
2. When the washer is finished, place the wet load in the dryer.
3. When the dryer is finished, place the dry load on a table and fold it.
4. When folding is finished, you ask your roommate to put the clothes away.
When this is finished, start over with the next dirt load.
This requires a lot of time, and a pipelined approach takes a lot less time. If we look at each of these four parts of the analogy, we can interpret them as parts of the instruction set. As explained, the first part of the instruction set will not start until the last part of the instruction set is done. The pipelined approach is that each time one part of the instruction set is done, it will immediately start another one in the same part of the instruction set.
This means that the time to run a single instruction through the simulator will not take any shorter time, but running many instructions through the simulator will go faster as the instructions are working in parallel.

2.4 Data hazards

Data hazards[2] becomes a problem when different stages of the pipeline is data dependent. This means that a problem occurs when one part of the instruction needs data from another part of the pipeline. The three situations where data hazards can occur is:

2.4.1 Read after write data hazard

Read after write refers to the situation where an result has not yet been written by one instruction, but another instruction tries to read or refer to this result. This problem occurs because the previous instruction has not been processed through the pipeline when the next instruction requires some data that is not available.

2.4.2 Write after read data hazard

Write after read refers to the situation where one instruction tries to write a destination before it is read by the previous instruction in the pipeline. This can be a problem i.e if two registers execute two separate instructions simultaneously, but the second instruction needs data from the first instruction.

2.4.3 Write after write data hazard

Write after write data hazard is when an instruction tries to write a data before it is written by the first instruction. This is very similar to the write after read data hazard.

2.5 Control hazards

Control hazards, which occur when one decision is based upon the results of an instruction while others are executing. The processor will not be able to know the control signal because the previous instruction is not finished, and therefore it won't be able to insert a new instruction to the pipeline.

3 Design

In this assignment we were to design the MIPS instruction set and it had requirements that had to be followed, but the instruction set could be implemented in the design of the implementors choice as long as it fulfilled these standards

3.1 Connecting the elements

All of the datapath elements are of the type CPUElements. This means that they all have the same possibilities of recieving input and writing output. When two units are connected, some type of output, either a output or a control output, will be the some input to the unit it is connected to.
Let's take an example: If the program counter is connected to the instruction memory, and the program counter is the instruction just before the instruction memory instruction, the output from the program counter is the input to the instruction memory. This output/input coherence is the connection between these two units.This is the way all of the datapath elements are connected to each other, in either output/input coherence or control input/control output coherence. This also makes it possible to connect multiple elements to each other by having multiple output/input. For example can register file be connected to both the ALU and the data memory "output-wise" by having two independent outputs and the inputs to each of these elements will be the two independent outputs from the register file. Units can also be connected to multible units by using the same output on two or more datapath elements.

3.2 Writing the output

All the elements write some sort of output, either state output or control output. Based on what type of state element it is, the output it generates will be defined. For example the output the ALU generates will be defined by the control signal it recieves. So the output generated in this method will be the state output or control output that is connected to the next state element.

3.3 Summary of the datapath and controllers

3.3.1 First part

The simulator is not completed in full with data hazard handling, but every datapath element and control units have been implemented and checked to work through test code. The instruction set looks as follows:
The program counter takes in one input and this input is the address which the program counter holds, and sends to the instruction memory and the first adder.
The instruction memory will use this address in order to create six different outputs. These output will go into the IF/ID pipeline register.
The adder that takes input from the program counter will add this address with the constant 4, and this result will also be sent into the IF/ID pipeline register.
Now the IF/ID pipeline register gives just the same output as it got input. And we can start the second part of the instruction set.

3.3.2 Second part

Now the pipeline register IF/ID will give its output to the Control unit, the register file and the sign extension unit. A shift left 2 unit is also used in this part, and a jump unit.
The control unit will recieve input from the IF/ID pipeline register, this input, which actually is the operation field from the instruction memory, will be used to generate the proper control signals. These signals will be sent to the pipeline register ID/EX.
The register file also recieves four inputs, two of which comes directly from the IF/ID pipeline register. It generates two outputs that will be stored in the ID/EX pipeline register.
The sign extension unit takes in three inputs from the IF/ID pipeline register, puts the input together and extends the value.
The shift left 2 unit takes five inputs, puts them together and extends it by two bytes. And sends this result to the ID/EX.
The jump unit will take in the shift left 2 result and the result from the adder from the very beginning, this unit will jump to a new address with these inputs and stores this in the ID/EX pipeline register.

3.3.3 Third part

Now the pipeline register IF/ID will hold all the necessary information in order to continue the process. This part of the process holds these components: two multiplexor, the control unit of the ALU, the ALU, a shift left 2 unit, an adder, a branch not equal unit and a branch.

The first multiplexor recieves its input from the ID/EX pipeline register and recieves a control signal from the same register. The output will go into the EX/MEM pipeline register.

The control unit recieves its input from the ID/EX pipeline register and based on its input it will create a control signal that is sent to the ALU.

The next multiplexor will take its input from the ID/EX register, and its control signal from the same register. The result is given to the ALU.

The ALU recieves one input from the pipeline register, and one input from the multiplexor described above. The output is given to the EX/MEM pipeline register, and the Zero control signal is given to the branch not equal unit, and in essentiality also to the branch equal unit.

The shift left 2 unit takes its input from the ID/EX pipeline register. Its output is given to an adder associated with a multiplexor that is in on deciding the address of the program counter.

The branch not equal unit takes in one control signal from the ID/EX pipeline register, and one control signal from the ALU. The output control signal is given to a third multiplexor.

The branch unit takes one control input from the ID/EX and one control input directly from the ALU. The result is given to the multiplexor associated with the branch. The output from this multiplexor is given to the multiplexor associated with the jump unit.

3.3.4 Fourth part

For the fourth part of the process the EX/MEM pipeline register holds all the values, and will give them as inputs to the data memory.
The data memory takes two inputs from the EX/MEM pipeline register and two control signals from the EX/MEM. The output is given to the MEM/WB pipeline register.

3.3.5 Fifth part

A multiplexor recieves two inputs from the MEM/WB pipeline register, and it recieves a control signal from the MEM/WB pipeline register.

4 Implementation

All the code was written in the language "python" and all the datapath and control elements are implemented in separate files with complying names.

4.1 Similarities in each datapath element

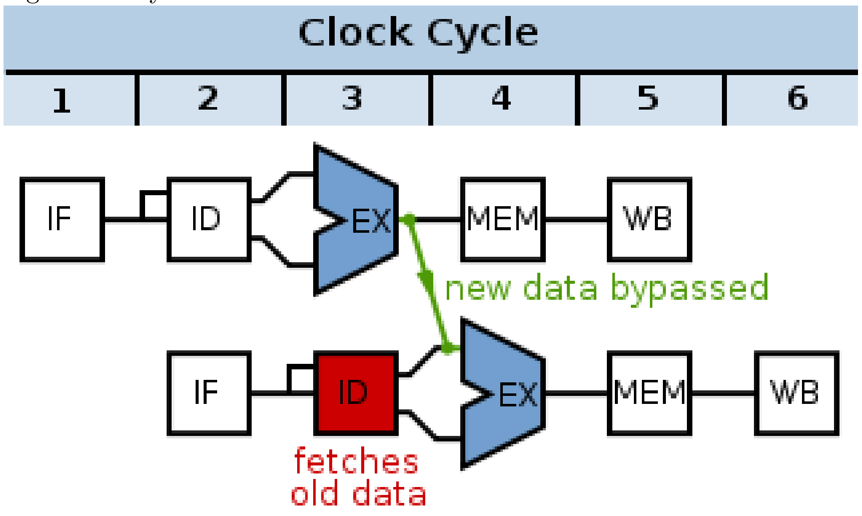
Each element has a connect method which connects it self to the unit(s) before and the unit(s) after. This method is similar in all of the elements, and the only difference is the number of state inputs, state outputs, control inputs and control outputs.
Every unit also has a method that defines the output to be written. This method generates output based on what the unit is supposed to do, what state input it takes in and what control input it recieves.

4.2 The tests

In addition every element has a test which tests the ability of the implemented unit to connect with a datapath element before and a data element after.
The basics of the tests are the same for each element: A test element is created to simulate the unit before the unit you are testing. Also, another test element is created to simulate the element after the unit one is testing. This means that you are giving some output from the test element which is first in line, the element you are testing recieves this as input, and should generate output which makes sence for the second test element to take as input. This is the test that was done for each datapath - and control element.

5 Discussion

The first part of the data hazard handling approach would be to implement a forwarding unit that acts as an internal buffer. This would resolve the problem of not having the needed information during a clock cycle.



In this figure, an example of forwarding to avoid data hazards by using internal buffers is shown.

5.1 What lacks in the implementation

All the datapath elements and control elements have been connected, and should work in a single cycle, but since the register file needs input from an instruction further on in the simulator, a single cycle won't be able to work. In addition pipelining has been implemented, but the pipelining does not seem to work properly. This may be because the forwarding has not been designed or implemented.

5.2 Limitations and known bugs

As far as the author know there are no known bugs. But there may be a amount of memory leakage.

6 Evaluation

6.1 Evaluation of the MIPS instruction set

The MIPS instruction set does not work, and therefore does not fulfill the requirements of the assignment. Accessed 14.10.2015

7 References

1: Patterson, D A and Hennessy, J L, 2014, Computer organization and design, pg. 272.
2: Wikipedia, 2015, "https://en.wikipedia.org/w/index.php?title=Hazard_(computer_architecture)&oldid=67772887

Accessed 14.10.2015

References

[?, ?, ?, ?]