UNIVERSITY OF TROMSØ

INF-2200: Computer Architecture and Organization

ahe103

11.10.2016

1 Introduction

The assignment gave the task of programming all the components required for a CPU to perform a limited set of operations, then to pipeline the process.

1.1 Requirements

The following operations had to be implemented for the CPU to perform

J	Jump	**ADDI**	Add Immediate
BEQ	Branch Equal	**ADDIU**	Add Immediate Unsigned
BNE	Branch Not Equal	**SUB**	Subtract
LUI	Load Upper Immediate	**SUBU**	Subtract Unsigned
SLT	Set less than	**AND**	Binary AND
LW	Load Word	**OR**	Binary OR
SW	Store Word	**NOR**	Binary NOR
ADD	Add	**BREAK**	Break Execution
ADDU	Add Unsigned		

2 Technical Background

As this assignment revolves around a MIPS processor in its entirety one should know about the inner workings of a processor, its stages and elements as well as optimization that can be performed by "pipelining" the entire thing

2.1 Stages

A processor handles each operation it performs in stages, these stages are meant to streamline the process of performing operations. The stages themselves are called "Fetch" (IF), "Decode" (ID), "Execute" (EX), "Memory" (MEM) and "Write Back" (W/B)

2.1.1 Fetch

The task of the Fetch stage is to gather the necessary info on the instruction to perform next stage for that pass of the clock cycle. The stage refers to the Program Counter (PC) to find out what instruction the Instruction Memory (IM) should push out to the next stage. When that has been determined there is a separate loop that takes the PC info and increments it so that it knows where to "Fetch" the next instruction from. There are other elements that determines what that instruction should be mainly what the previous instruction was. Should it "Jump" to another part of the instruction set? Should it "Branch" back to repeat some instructions again? All that is determined in the next stage by the "Control" (CTRL).

2.1.2 Decode

Decode is the stage where all the instructions are determined and positioned in the correct position before the EX stage. It decodes the data that came from the IM into it's respective slots according to certain formats. The 32 bits in the data is structured so that the first 6 always resemble the "Operation Code" (OpCode). The next 26 bits of the instruction is determined by what the format was. There are 3 formats in a non float processor, the R-format, I-format and J-format.

The R-format refers to Register operations, from register A and register B to register C. All three of these registers are represented by 5 bits, so this format now has 10 bits left. Those go to a field called "Shift amount" (shamt, 5 bits)) used for logical shift operations, and a function for the Arithmetic Logic Unit (ALU) to perform (funct, 6 bits).

The I-format refers to Immediate (Imm) operations, from register A to register B with an immediate value of 16 bits

The J format refers to a Jump operation. The format uses the 26 bits left to represent an address to jump to, then the Jump Calculator (JmpCalc) mixes the jump address with the PC address in order to generate an address that the PC should jump to next

2.1.3 Execute

This stage of the processor does all the calculations, the control signals tells where to pull the data from and what operation to use, where in the register to store these calculations

2.1.4 Memory

The Memory stage is only really used if the instruction decoded from a few stages ago results in a read signal or write signal from the controller. That signal determines if the "Data Memory" should do anything or remain passive

2.1.5 Write Back

This is the final stage in a processors operation cycle. It's only purpose is to determine if the data to be returned to the register is coming from the Data Memory or if it was meant to be stored in the register directly

2.2 The Elements of a CPU

The innards of a CPU is vast array of different modules that when used in unison and connected in proper order works as a charm. Noteworthy elements are the ALU, Control, and ALU control.

2.2.1 Arithmetic Logic Unit

This unit is the main meat of the processor. Without it the processor would have severe problems performing any arithmetic. The purpose of this unit is in its name, to perform Arithmetic operations. This can be fleshed out to include addition, subtraction, setting states, logic operations like AND, OR and NOR.

2.2.2 Control unit

If the ALU is the main meat of the processor then the Control unit is the Brain of the processor. The control unit takes in the OpCode from the IM, and flips switches on flowgates called Multiplexers (MUX) determining what they should take in and let out. Make the register and Data Memory accept inputs or give outputs.

2.2.3 ALU control unit

As the Control unit the ALU control does a bit of thinking. It takes in a signal from the Control unit to determine what it should output, if it gets the command for an R-format instruction it will read the "funct" part of the instruction to determine what signals to send to the ALU to perform

2.3 Pipelining

What is in a water pipe? 99% of the time only water, no gaps, only water. This is in essence what pipelining in coding means too. In a processor you want every stage to be working on something, you don't want any gaps in the code execution. This means that once one instruction is sent the next should be able to be sent the next cycle of the processor, effectively removing all gaps in the processor. This is a very effective thing to accomplish, but there are several hazards when just pumping out code without looking for pitfalls such as read after write, or write after read

2.4 Hazards

There are several hazards when it comes to execute code in a processor. Some of them might seem innocuous but merely the act of reading before writing could introduce severe problems.

3 Design and Implementation

The design of the processor follows closely to the illustrations in the book, but some liberties had to be taken when it came to the amount of info that was sent to the ALU control and the ALU since some of the operands couldn't be covered in the amounts displayed in the book. One simplification used was to make the IM send out dedicated signals in order to reach the elements that required them. Instead of sending out just the 32 bin instruction,

Since the design follows very closely to the illustrations in the book there is only a few differences. Mainly the way the jump is handled, that got its own element called "Jump Calculator" now in order from the PC and following a single cycle the modules will be described in operation.

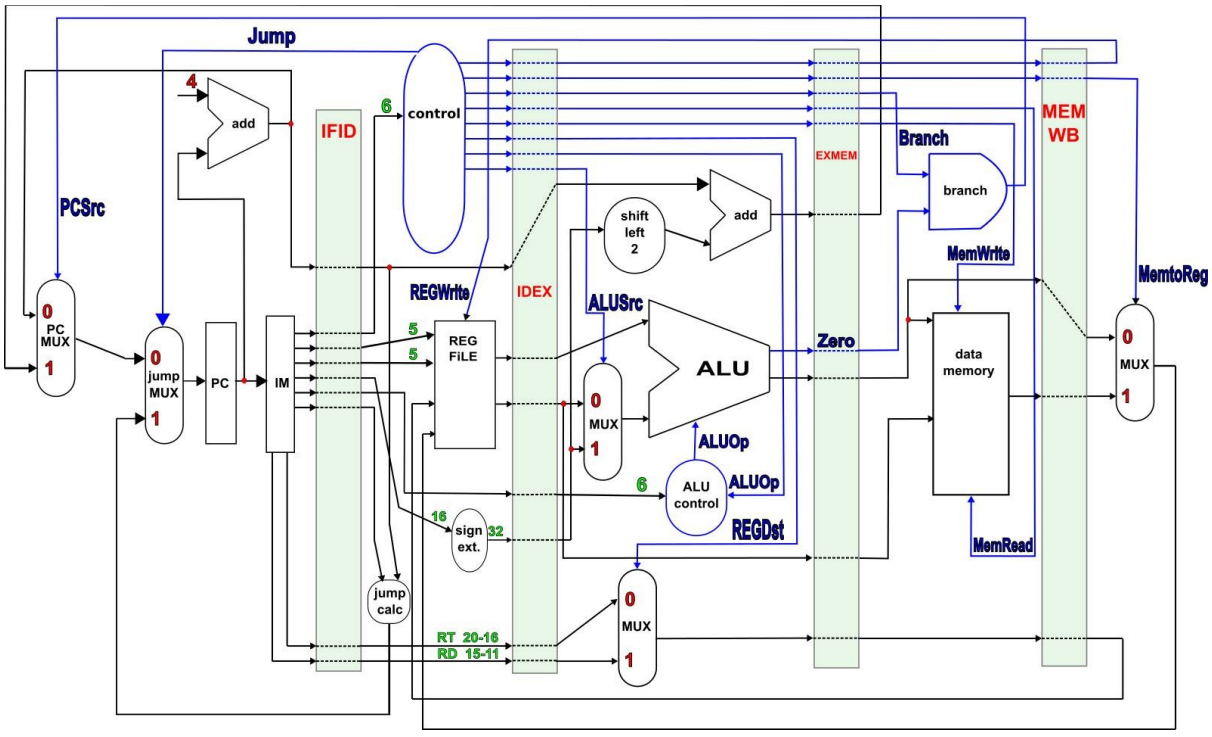

Figure 1: a diagram of the implementation and connections in the processor

Program Counter (PC): this element was not tampered with as such was not needed

Instruction Memory (IM): this element had to be initialized with a memory file in order to work. The memory file was read and put into a dictionary where the addresses was the keys and the instruction was the value. When the writeOutput method was called it put the instruction in all of the outputs with different bitmasks depending on what section of the instruction it was.

Add: this element was not altered

Mux: this element was not altered

Wall (IFID/IDEX/EXMEM/MEMWB): this element merely sent everything it got in, out the opposite side.

Register (reg file): since the Control unit gives the signal to read or write this element passes the register addresses' data out the other side. If the write signal is present it writes the data to the given register, but ignores the data if the register address corresponds with register Zero which must remain zero

Control: this unit takes a look at the OpCode that the IM sends out, then following an If test sets the correct settings for the given control signals before releasing them toward the IDEX wall

Sign extender: performs 16 left shifts on the input and pushes it out in order to make it 32 bit from the incoming 16

Shift Left 2: does as advertised, takes the input and shifts it left twice before sending it on

ALU control: takes in the "funct" part of an R-format instruction and the control signal from the Control unit. This signal can be any of the 2 bits available 0, 1, 2, or 3 this tells the ALU control what it should send out. At control code 2 it tells that the operation is an R-format operation it will If test the "funct" data to find out what op code to send to the ALU itself.

ALU: the ALU takes in the two inputs and performs one of the tasks the ALU control says it should. This encompasses AND, OR, Add, Sub, BNE, LUI.

Branch: this is basically an AND gate, if the control signal from Control and the ALU is both active the branch will take effect.

Data Memory: this is the main storage area for the processor. It will take any address and data, with one of the control signals for either writing to or reading from then write to memory or read from it according to the control signal

Jump Calculator: the JumpCalc handles the problem that the illustrations in the book explains poorly, how to jump. A jump is performed by sending the jump opcode and the address in 26 bits, how does that translate to a complete 32 bit address. It doesnt, that's why this unit was needed. This unit shifts the jump address by 2 bits left, to make a 28 bit address. Then the upper 4 bits is suffixed by OR-ing with the 4 upper bits of the Program Counter address, making a complete 32 bit address

4 Discussion

The processor handles the operands given by a program and it properly iterates the Program Counter, but it does not manage to write it to the register. This error was traced to some bug in the transition from the register to the ALU or from the ALU to the register. This was not fixed by the deadline.

The processor does not possess any form for hazard detection and handling, nor does it have any forwarding units. This would have helped with the failure to write to register.

5 Conclusion

The assignment gave the task of writing modules for a processor in order to assemble said modules into a working processor. All the modules are in place but some kinks makes it not run properly as it does not write to register. A key problem for any program trying to run on this platform. This assignment has given great insight into the hardware side of a processor and its data path. It's a truly remarkable piece of engineering.

6 References

[Patterson and Hennessy, 2014] Patterson, David A and John L, Hennessy
Computer Organization and Design
Elsevier Inc

Figure 1 provided by Raymon Hansen