Imperial College London

Department of Electrical and Electronic Engineering

2nd Year Coursework

Instruction Architecture and Compilers

Synthesisable MIPS-Compatible CPU

Autumn 2021

TEAM 01

Overview

The CPU we have created implements a subset of the MIPS IV ISA revision 3.2 operating using 32, 32 bit registers. It follows the basic principles of a bus architecture, having instructions and data share the same interface to the CPU. This is a 32 bit, big endian system and there is no support for double word instructions or instructions involving floating point numbers. As in the given specification, the CPU uses the Intel Avalon memory mapping interface allowing for an overall more accessible solution.

The core components of the CPU are the ALU, controller, instruction register,program counter, register file:

ALU - Arithmetic Logic Unit

This block consists of an ALU controller, ALU, and a mult/div ALU. The majority of this block is combinational allowing for asynchronous operation (which we don't use). The mult/div section stops the whole ALU from being combinational as it contains the hi and lo registers which must be updated synchronously.

Controller

The controller serves to provide the other blocks and muxes with the information they need to carry out their respective instructions such as enables and op/funct codes using data from the 32 bit instruction

Instruction Register

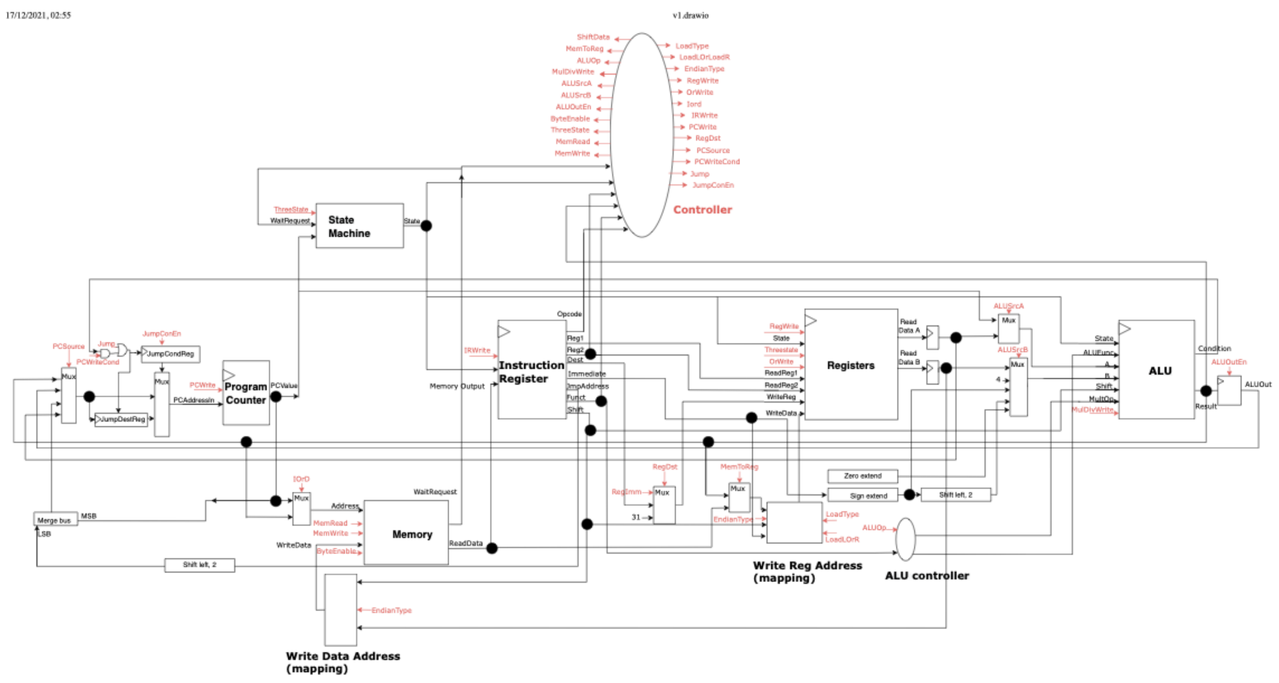
The instruction register is responsible for breaking up instructions into the register addresses, immediates, jump addresses and any other information required by the CPU in order to execute a given instruction.

Program Counter

The Program counter is used to indicate the address of the next instruction to be executed. This starts at 0xBFC00000 as this is the specified instruction start address. Address 0 is the halt address, if the program counter is sent to zero, the system halts.

Register File

This is where the 32 registers are. The register file is 2 read 1 write enabling 2/3 operand instructions to be executed within one cycle. For testing purposes, v0 is also an output as this is the main register used to confirm a test's correctness.



Top level diagram of CPU.

Implementation

The Solution drew inspiration from David A. Patterson and John L. Hennessy's "Computer Organization and Design - fifth edition". Within the text there is a simplified version of a MIPS CPU, which was adapted to fit our own specifications and contain the required instruction support.

Key features in our implementation include:

- Mult/Div integration into ALU - the mult/div block, also known as the MDU is built into the ALU block as this helps to focus all mathematical operations into one area. This decision was prompted by ease of design as the integration allowed for a single code to be sent to the ALU controller to then decipher what is required of its child module - reducing the number of (top level) controller outputs.
- Due to the limitations of SystemVerilog, we cannot simulate a 32-bit address RAM. To circumvent this, a 4096x32 RAM was used with a simulated address which directly maps to the corresponding address of a MIPS RAM. The CPU itself has a little endian interface and everything is calculated in little endian but must convert to and from big endian when outputting and inputting values to be stored correctly in memory.
- Using the Avalon interface meant that the waitrequest pin could go high at any time. It was important to ensure correct stalls in the correct cycles so as to not cause any unnecessary propagation delays.
- To implement the jump/branch delay slot, the cpu uses the signal identifying the jump/branch should take place to store the jump/branch destination at the end of the jump/branch instruction. This signal is then fed into a flip-flop which controls a MUX

that switches the program counter to take its next value from the stored destination rather than pc+4 during the fetch of the instruction in the delay slot.

Test Bench

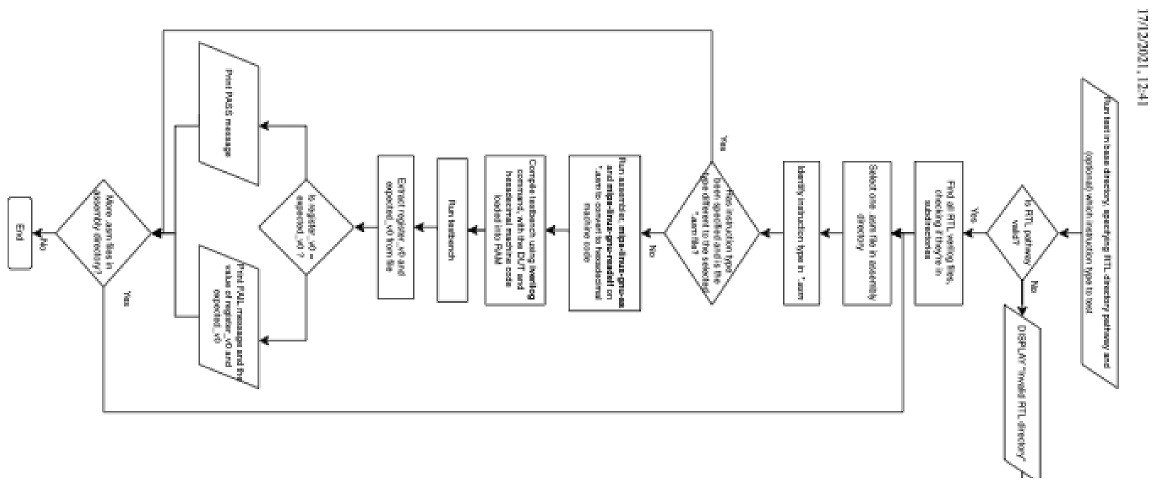
Firstly, in order to test our MIPS CPU, we designed a test RAM to load our instructions into.

Collectively, we wrote numerous programs to test particular instructions from the ISA, written in assembly code. The goal of these test programs was not only to test normal functionality, but edge cases as well ensuring a holistic correctness in a given design. Furthermore, various instructions depend on the success of other instruction tests. For example, a particular BLEZ test utilises a SUBU instruction to provide a negative input to the BLEZ instruction.

In order to load our test programs into RAM, the assembly code needed to be assembled into machine code (hexadecimal). Instead of converting each one manually (or writing a C++ conversion program), we used a linux package called *mips-linux-gnu-as*. This ensures that the conversion is accurate and more time can be spent managing other aspects of the testing process. We made sure that all our test programs store their result in register_v0 for ease of access.

For the jump and branch instructions, checking the next value in the program counter would be ideal, however, this would further complicate the test bench. To avoid this, extra instructions, such as ADDIU, are put before and after the branch on/jump target address , to deduce if the CPU executes these instructions properly. This means that these tests check whether instructions are skipped correctly as opposed to directly checking the program counter.

A script was made to automate the testing of instructions. The following diagram shows how the test script works:


17/12/2021, 12:41
Run test in base directory, specifying RTL directory pathway and (optional) which instruction type to test
Is RTL pathway valid?
No
DISPLAY "Invalid RTL directory"
Yes
Find all RTL verilog files, checking if they're in subdirectories
Select one .asm file in assembly directory
Identify instruction type in *.asm
Has instruction type been specified and is the type different to the selected *.asm file?
Yes
No
Run assembler, mips-linux-gnu-as and mips-linux-gnu-readelf on *.asm to convert to hexadecimal machine code
Compile testbench using iverilog command, with the DUT and hexadecimal machine code loaded into RAM
Run testbench
Extract register_v0 and expected_v0 from file
Is register_v0 = expected_v0 ?
Print PASS message
Print FAIL message and the value of register_v0 and expected_v0
More .asm files in assembly directory?
Yes
No
End

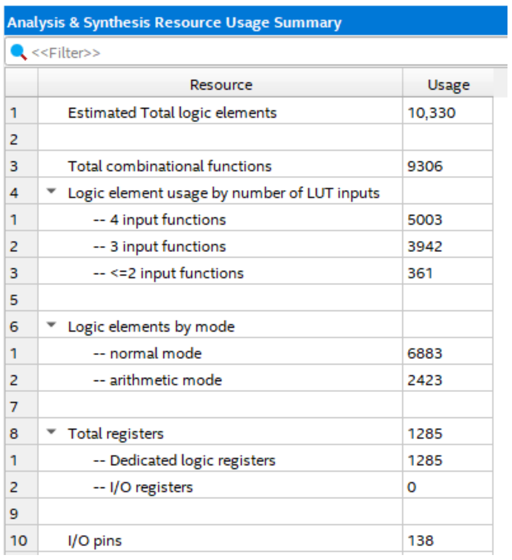
Flow Chart showing the test script steps.

Firstly, the script is called from the source directory:
test/test_mips_cpu_bus.sh [Directory] [Optional Instruction]
The directory refers to the *rtl* folder which may contain the verilog files or another subfolder called *mips_cpu*. The CPU files may be spread among these directories. The second argument allows the user to specify if only a single instruction type should be tested. If nothing is entered, by default, all instructions are tested.

The assembly programs are stored in *test/assembler/assembly*, where *mips-linux-gnu-as* is called to convert the *TESTNAME.asm* files into hexadecimal machine code. The machine code is then loaded into our *mips_cpu_avalon_RAM.v* and our test bench compiles and runs the MIPS CPU. After, the result of register_v0 is written to a text file and stored in a folder called *compiled_results*. A comparison is then made against a predetermined value stored in the *results* folder, if the text files contain the same value for register_v0, the test case passes. If the test case fails, the obtained and correct value for register_v0 is output for debugging purposes.

Evaluation

Using the timing and compilation tools in Intel's Quartus prime with the Cyclone IV E Device EP4CE30F23C6 setting, we were able to determine the total area that our solution would use:



Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	10,330
2		
3	Total combinational functions	9306
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	5003
2	-- 3 input functions	3942
3	-- <=2 input functions	361
5		
6	Logic elements by mode	
1	-- normal mode	6883
2	-- arithmetic mode	2423
7		
8	Total registers	1285
1	-- Dedicated logic registers	1285
2	-- I/O registers	0
9		
10	I/O pins	138

Simulating our design under the realistic conditions of 1200mV at 85 Degrees Celsius, we were able to determine a maximum clock speed of: **8.33MHz.**

Without timing optimisations, this is a strong start towards the approximate clock speed of 1GHz of the MIPS32 74k processor.