Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

CPU101 & CPU202 Datasheet

Author: Group 14

Cathy Ding (CID:01856648)

Charmaine Louie (CID:01711892)

James Ong (CID:01848230)

Christina Wang (CID:01866692)

Haoran Wu (CID:01954226)

Mengyuan Yin (CID:01842827)

December 2021

CPU101 & CPU202 Datasheet

1. Design and Architecture

CPU101: Non-Pipelined CPU

CPU101 is a non-pipelined, hardware-synthesizable MIPS CPU that utilizes the MIPS ISA with bus-based interface [1]. CPU101 features 5 states. Data flow across the states is achieved by using intermediate registers to store the value between blocks, ensuring the accessibility and segregation of correct values for each block, which paves the way to support pipelining.

CPU101 contains 7 modules, each executing distinct functionalities to ensure this CPU is hardware synthesizable. Furthermore, individual modules are replaceable if there is a more advanced version that can be easily integrated by connecting the inputs and outputs. With these two features, CPU101 can be manufactured for real-life implementations and provides high commercial value

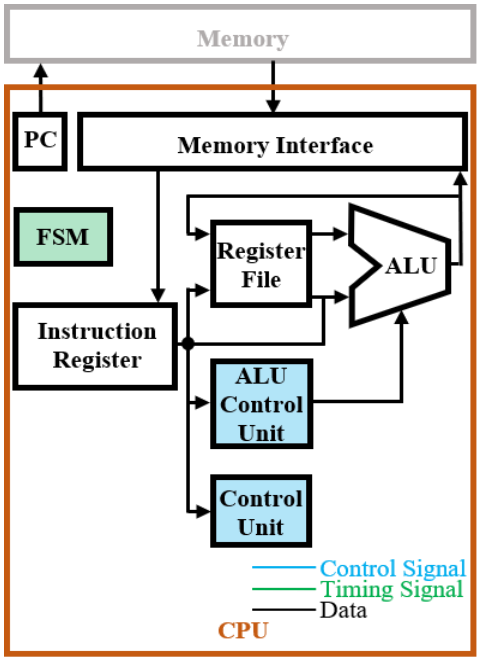


Figure 1: Overall Architecture of CPU101

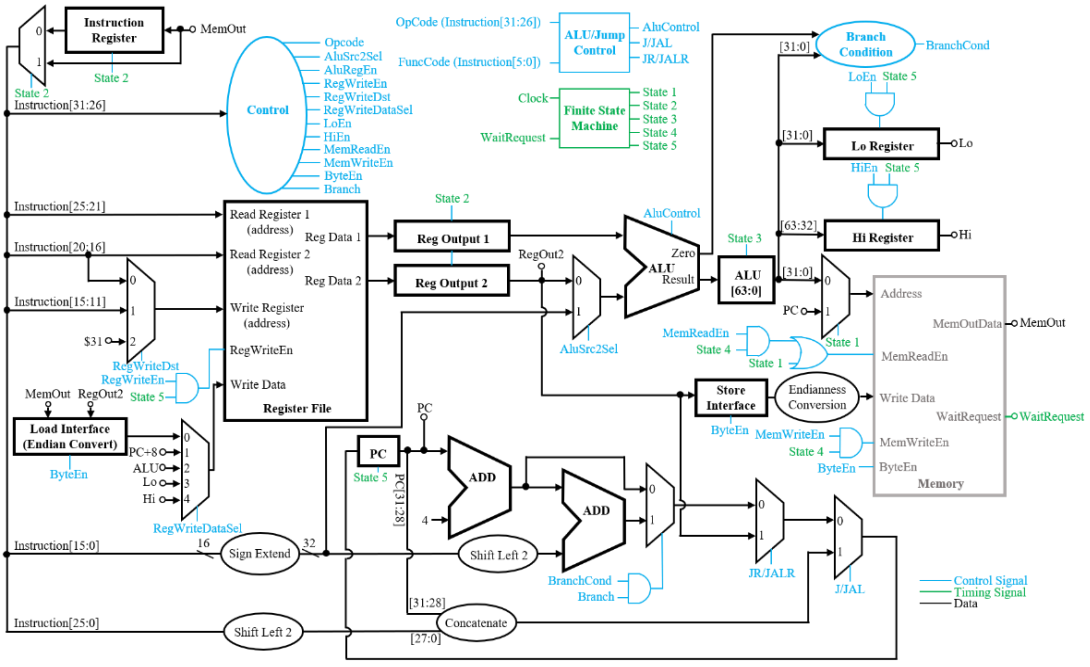


Figure 2: Detailed Architecture of CPU101

Details of the 7 modules:

- **Finite State Machine (FSM)**: CPU101 implements 5 cycles per instruction using a 5-state FSM. This is beneficial as it facilitates pipelining implementations and is the most common way to implement MIPS CPU, fitting the agreed-upon industry standard. Functionalities are clearly divided up into 5 sections in time, including reading from memory at State 1, reading from the two register file outputs at State 2, ALU calculations at State 3, loading from and storing to memory at State 4, and writing back to register file at State 5 (labeled green in Figure 2).
- **Program Counter (PC)**: The PC is responsible for determining the address of the next instruction. For the PC to be compatible with pipelining, a branch/jump delay slot is introduced to deal with branch/jump hazards. This is implemented by storing the destination of the branch/jump into a register. During the branch delay slot, the PC is updated with the next word address; one instruction later, the stored target address is fed into the PC. This enables a delay for the branch/jump instruction to take place.
- **Control Unit (CU)**: The CU serves as a driver for the other logical components of the CPU. It uses 32-bit input instructions and the current state to determine control signals necessary for other CPU blocks (labeled blue in Figure 2). This unit greatly simplifies the construction of each CPU block, making it far easier to implement on hardware.
- **ALU Control**: In order to minimize repetition in calculations, an ALU control module is added to the CPU. This module categorizes 50 listed instructions according to their arithmetic operations with a 4-bit ALU control signal, thus reducing the number of logic gates required and saving space.
- **ALU**: The ALU is at the center of all operations in the CPU. It performs different logical and arithmetic operations according to the 4-bit ALU control signal. The 64-bit ALU output is specifically chosen to accommodate multiplication and division instructions. This allows the result of multiplication and division to be stored in HI and LO registers in one cycle.

- **Memory Interface**: The memory interface serves two purposes. Firstly, it performs an endianness conversion to support the connection between a memory unit (with a little-endian interface) and a big-endian CPU. Secondly, the memory interface is used to prepare and interpret data used by load/store instructions. It utilizes a 4-bit byteenable signal to control which bytes of a word in memory should be fetched from or written into. Enabling instructions like LB, SB, LH, etc. to be achieved.
- **Register File**: The register file is adapted for multiplication and division instructions as it features the HI and LO registers in addition to 32 registers for storing values. It includes extra paths to assist the PC when executing branch and jump instructions with link functionality. During these executions, the current PC address is stored into a register to enable the return in a function call.

CPU202：Pipelined CPU

To enhance the performance, a pipelined processor CPU202 is built based on CPU101, with the same bus-based interface. Pipelining keeps all parts of the CPU occupied by breaking instructions across multiple cycles and using intermediate registers to store values between stages. This enables multiple instructions to be executed in parallel, hence increasing the throughput [2].

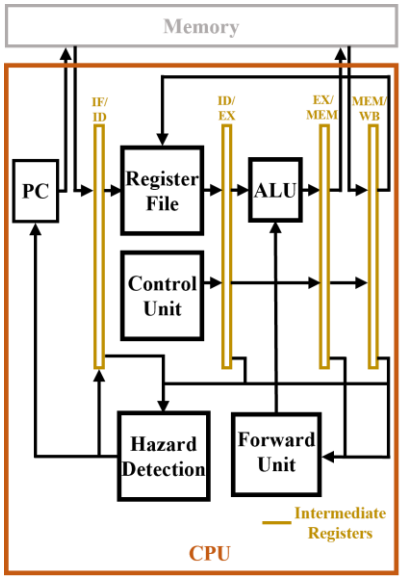


Figure 3: Overall Architecture of CPU202

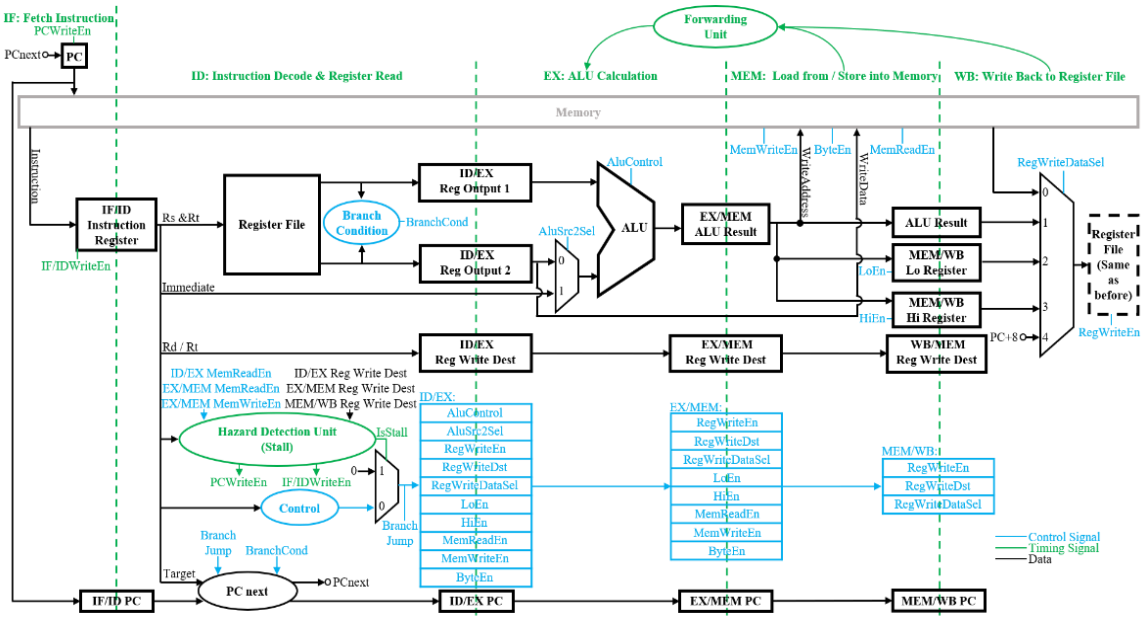


Figure 4: Detailed Architecture of CPU202

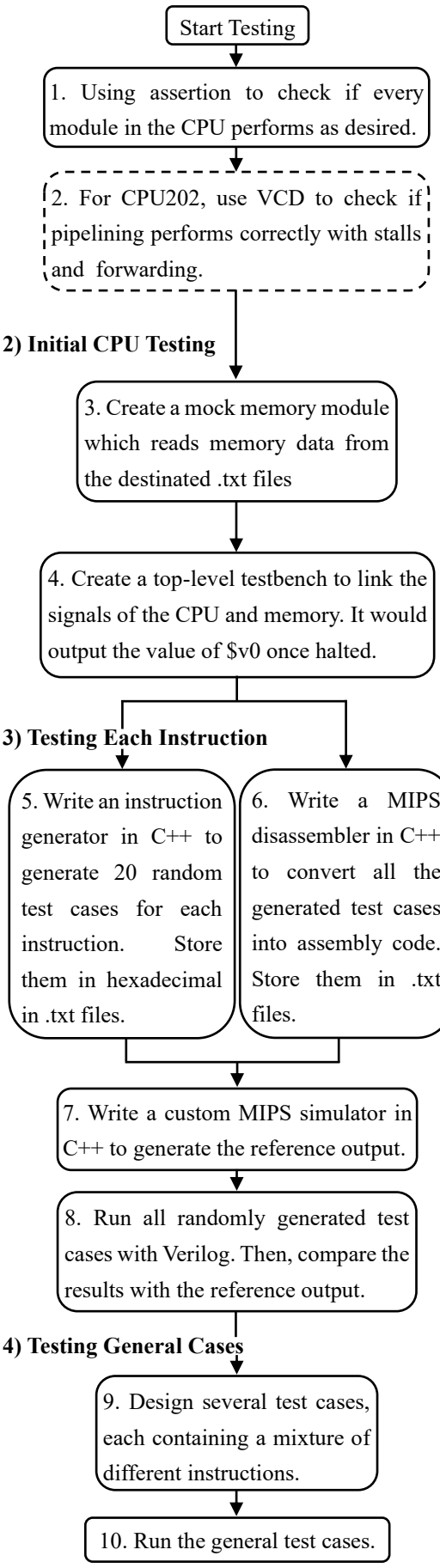
There are a few modules that need to be mentioned:

- **Forwarding Unit (FU):** The FU is introduced to resolve data hazards, which happen when the data of a register is required before it is updated. To access the renewed data earlier, data loaded from memory (outputted at WB state) and ALU result (obtained at MEM state) are directed to ALU input (required at EX state). This is prioritized for treating data hazards as it effectively prevents adding any data stall, which requires extra cycles.
- **Hazard Detection Unit (HDU):** The HDU is used to stall pipelining until the hazard has passed. It deals with data hazard situations that cannot be solved by FU. For example, an instruction is dependent on its previous instruction, which is a load/store. When the loaded data is retrieved from memory at WB state, the dependent instruction reaches MEM state, therefore, it misses the clock cycle and cannot be forwarded with the renewed data for ALU calculation. Moreover, instead of using a Harvard interface, CPU202 uses a bus-based interface, which is assumed to have only one address port. Consequently, during the MEM state of a load/store instruction, all other memory accesses are forbidden, hence fetching a new instruction will be delayed by one cycle. This effectively resolves the structural hazards brought by the bus-based interface.
- **Program Counter (PC):** The PC is designed to be updated at ID state, so that during a branch/jump instruction only one consecutive instruction will be fetched from the memory, before the branch/jump decision and destination is determined. Hence, only one branch delay slot will be required. Moreover, to achieve branch/jump with link instructions, the value of PC is stored in a sequence of registers until WB state when the value of PC is written back into the specified register.
- **HI & LO Registers:** Unlike being written back at WB state like general-purpose registers, HI & LO registers are renewed at MEM state. This is because data inputs for HI & LO registers only come from ALU calculation (EX state) and will not involve any memory access (MEM state). By writing into HI & LO registers one cycle earlier, possible stalls related to MTHI, MTLO, DIV, MULT can be prevented.

2. Testing Methodology

There are a total of 4 main stages of CPU testing, starting with modular testing and working towards integration testing. The following testing procedures are used on both CPU101 and CPU 202.

1) Modular Testing



1: Module testing is white-box oriented, which can identify potential errors at the early stage. The functionality of each module (i.e., PC, FSM, Register File, ALU, etc.) is tested with its own testbench. The testbenches consider edge situations, (e.g., overflow, boundary for branch decision etc.). Moreover, cases like negative offset for branch/jump, negative immediate, and sign extension are tested to guarantee that both CPUs can work under any condition.

2: The VCD waveforms should display indications of pipelining by showing that the instruction from memory is updated with each clock cycle. Instructions with the need for stalls and forwarding have been tested in detail.

3: To avoid overwhelming the .txt file with memory storage, the memory is split into two .txt files: the first file starts at address 0 and the second file has addresses that are all offset by the reset vector 0xBFC00000. Each of the two .txt files is mapped to a memory vector with 2^{16} elements, so that the accessible memory addresses are [0x0, 0x10000) and [0xBFC00000, 0xBFC10000). In this way, two smaller RAMs are used to emulate the behavior of a single large RAM. Despite the separation, data and instructions could be stored anywhere within the two files.

4: The wait request signal is set in this module using the $urandom_range() tool in Verilog, so that it would randomly go high when performing a memory read or write. Ensuring that the CPU can handle a series of stalls.

5&6: Random numbers are put into the replaceable fields (i.e., $Rs, $Rt, $Rd, Immediate, offset, etc.) of an instruction. This allows our testing to cover different situations with a simple algorithm. These test cases are designed to minimize the supporting instructions used with the instruction being tested. This makes the testing process less susceptible to errors [3].
The disassembler improves human readability to assist the debugging process.

7: A MIPS simulator is built using C++ for output verification. This process is efficient when handling a large number of test cases (as generated above) in contrast to calculating manually.

8: The approach is to use a mock memory module and a generic testbench to output the final value of $v0, which is then used for comparison with a reference output.

9&10: The instruction sets are designed in the way that they can perform the following complex functionalities:

1. For and While loop
2. Function call
3. Conditional statements

This is to test if the CPU can perform correctly in real-world scenarios.

During the testing process, the outputs from Verilog CPU and the C++ reference CPU are written into .txt files and then compared. This requires a considerable amount of file manipulation, hence consuming a large amount of time. Due to the limitation in total testing time, the number of test cases is restricted. Therefore, it is beneficial to develop a more efficient method for output inspection.

Overall, a systematic and detailed testing strategy is utilized. The automatic instruction generator ensures the extensiveness, randomness, and variety of our test cases. The reference CPU in C++ ensures the reliability and accuracy of testing results. In summary, more than 1000 test cases have been run (i.e., 20 test cases for each instruction and 6 general test cases with mixed instructions). Therefore, despite having room for improvement as mentioned above, the testing method still proves as a logical and credible way to validate CPU101.

3. Area and Timing Summary

Firstly, Cycles Per Instruction (CPI) is analyzed to evaluate the performance of the CPUs. For a pipelined CPU, the CPI value would ideally be 1, however, stalling would incur extra cycles for some instructions. For CPU202, an average CPI of 2.86 is calculated by running long lists of instructions. In the aspect of CPI, CPU202 has a lot of improvement space as a pipelined processor, however, it has outperformed CPU101(non-pipelined), which has a CPI of 5.

To better gauge the performance of the CPUs, Quartus was used to simulate an Intel Cyclone IV E FPGA. A fitter analysis is used to benchmark how effectively the CPU utilizes resources, while a timing analyzer is used to determine the efficiency.

	Logical elements	Max. clock rate (MHZ)[1]	Worst case slack (ns)
CPU101 (non-pipelined)	9594/39600 (24%)	104.45	-70.36
CPU202 (pipelined)	9778/39600 (25%)	52.61	-72.50

Table 1: Area and timing summary for Cyclone IV E (obtained from Quartus at 0°C and 1200mV)

Running a fitter analysis is used to determine how effectively each CPU performs in terms of area usage. As can be seen from Table 1, both CPUs require many logical elements on the FPGA, indicating how implementing the CPU has high area demands. The total number of logical elements used is marginally higher in CPU202 in comparison to CPU101. This result is sensible as the pipelined CPU contains more intermediate registers and introduces new blocks. Despite the slight difference, the number of logical elements used is still comparable for both CPUs.

Following this, timing analysis is run to determine the efficiency of both CPUs. Both CPUs have a negative worst-case slack value, indicating that the timing requirement is not met. This means that both CPUs would work poorly on actual hardware. Meanwhile, for the maximum clock rate, data connotes that CPU101 has almost double the maximum frequency of CPU202. This is expected as the pipelined CPU needs to propagate through more logical elements in each cycle (i.e., forwarding and hazard detection units).

While this suggests that the pipelined CPU may perform inferior in terms of maximum clock rate, this is untrue as pipelined architecture will theoretically be capable of performing instructions with a CPI close to 1 (about 2.86 in practice), whilst this would be capped at 5 on CPU101. Having a higher CPI with comparable area requirements negates the effect of having a slower clock rate and therefore indicates how a pipelined CPU is more efficient in general.

Overall, while the metrics of both CPUs indicate that the designs would be inefficient if synthesized. However, this was not the primary focus of this project. Both CPUs can execute instructions accurately in Verilog as prioritized, and they were not optimized for real-world performance. Additionally, being able to run a fitter and timing analysis on the E-FPGA shows that the design is ultimately hardware synthesizable as specified by project requirements.

4. Bibliography

[1] C. Price, *MIPS IV instruction set: Revision 3.2*. Mountain View, CA: MIPS Technologies, Inc., 1995.

[2] A. Gersnoviez, M. Brox, M. A. Montijano, J. A. Sujar, and C. D. Moreno, "UCOMIPSIM 2.0: Pipelined MIPS Architecture Simulator," *2018 XIII Technologies Applied to Electronics Teaching Conference (TAEE)*, 2018.

[3] I. Gribkov and A. Zakharov, *On one approach to random testing of MIPS microprocessors*, Moscow, Russia: Scientific Research Institute for System Studies, Russian Academy of Sciences, 2008.

[1]The measured max. clock rate acts as a best effort timing result based on client-provided information and can be tightened upon receiving detailed platform timing.