MIPS CPU Architecture: Design

The overall design decisions made during the development of the CPU is based on the following requirements:

- Intuitive codebase structure
- Ability for parallel development
- Modularity to form compartmentalised CPU sections which minimise the severity of any possible bugs and any logic errors
- Scalability to ensure product is adaptable to any future standards and instructions to maintain a competitive advantage

Bus-interface system [*mips_cpu_bus.v*]

The bus interface is designed to function as a wrapper that hands all memory transactions and ensures the interface abides by the Intel Avalon standard. Any updates relating to memory transactions are handled in the wrapper function. This design decision allows for the parallel development of the CPU core and the memory interface. This parallel design minimises the possibility for any logic errors that arises due to only one team developing the system.

The wrapper design allows the CPU to potentially support other memory interface standards like Intel Avalon as long as the wrapper meets the required interface of the CPU core. These features build to the ability for the CPU to be easily and efficient scaled depending on the client's needs.

Control Unit placement [*mips_cpu_harvard_mod.v*]

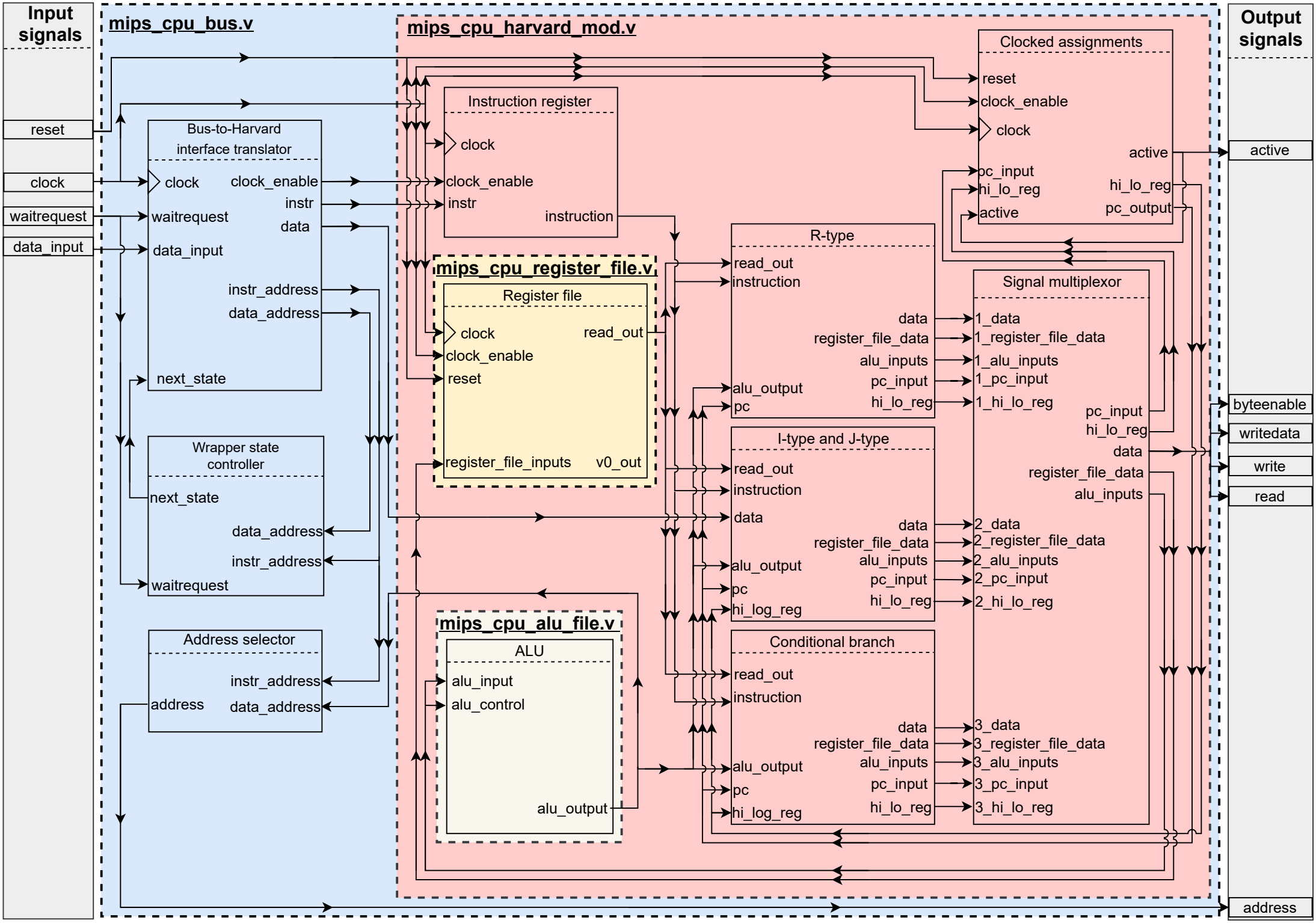
The main block, the Control Unit, is placed within the main file instead of moving it into its own file. This design decision made to aid in forming an intuitive codebase structure. The main structure of the codebase is a classification system that classifies every instruction into:

- *R-type*
- *I- and J- type*
- *Conditional branching*

Every control signal for a particular type is set within each of these blocks. This means all possible controls are well organised and allows new instructions to be easily added into the system.

This design leverages the similarities of instructions to create an easy and readable structure by classification and sub-classifications. This continues the theme of modularity and ensures any requests made by the client can be quickly incorporated.

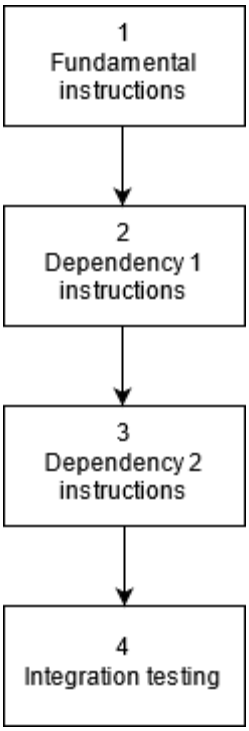
Standard commenting format is maintained throughout the codebase to aid any future programmer hired by the client to make the changes. With the control unit incorporated into the main classification structure and the standardised commenting, this creates an intuitive codebase that allows easy debugging and, due to the compartmentalised design, minimise any changes having unintended side effects.


Input signals
reset
clock
waitrequest
data_input
mips_cpu_bus.v
Bus-to-Harvard interface translator
clock
clock_enable
instr
waitrequest
data
data_input
instr_address
data_address
next_state
Wrapper state controller
next_state
data_address
instr_address
waitrequest
Address selector
instr_address
address
data_address
mips_cpu_harvard_mod.v
Instruction register
clock
clock_enable
instr
instruction
mips_cpu_register_file.v
Register file
clock
read_out
clock_enable
reset
register_file_inputs
v0_out
mips_cpu_alu_file.v
ALU
alu_input
alu_control
alu_output
Clocked assignments
reset
clock_enable
clock
active
pc_input
hi_lo_reg
active
hi_lo_reg
pc_output
R-type
read_out
instruction
data
register_file_data
alu_inputs
pc_input
alu_output
pc
hi_lo_reg
I-type and J-type
read_out
instruction
data
alu_output
pc
hi_log_reg
data
register_file_data
alu_inputs
pc_input
hi_lo_reg
Conditional branch
read_out
instruction
data
register_file_data
alu_inputs
pc_input
alu_output
pc
hi_log_reg
hi_lo_reg
Signal multiplexor
1_data
1_register_file_data
1_alu_inputs
1_pc_input
1_hi_lo_reg
2_data
2_register_file_data
2_alu_inputs
2_pc_input
2_hi_lo_reg
3_data
3_register_file_data
3_alu_inputs
3_pc_input
3_hi_lo_reg
pc_input
hi_lo_reg
data
register_file_data
alu_inputs
Output signals
active
byteenable
writedata
write
read
address

Testing and Verification: Testbench

The process of testing and verification of a given MIPS CPU is divided into 4 testing stages. The stages represent the classification of the testcases that the testbench uses. These stages are described in detail below. This classification system helps in narrowing and identifying any potential errors.

All test files and CPU outputs are located in the *"test"* folder of the main directory. In addition, a guide to the instructions tested and its classification in the testbench is included in the *"test"* folder.



1. Fundamental instructions:

   Fundamental instructions are instructions that can be tested without relying on other instructions to verify its result. These instructions include memory load instructions where data is directly loaded into the memory and add immediate (*addiu*).

   Any failure reported here will likely result in failures in the later stages as these stages rely on these fundamental instructions.

2. Dependency 1:

   "Dependency 1" instructions are instructions whose testcases include one other instruction during the testing process. The extra instructions are instructions from the "1: Fundamental instructions".

   Any failures in this stage can be identified by comparing the results of stage 1 and stage 2. If stage 1 reports no failure, any failures in stage 2 is solely due to the instruction tested.

3. Dependency 2:

   "Dependency 2" instructions are more complex instructions whose testcases include one or more other instruction during the testing process. These extra instructions can come from stage 1 or stage 2.

4. Integration testing:

   These testcases are complex sequences of instructions that attempts to imitate normal mathematical operations like calculating the Fibonacci series. These cases use a variety of instructions as a final test to verify the CPU can execute tasks expected of the CPU.

The testbench tests a given CPU by performing two checks for each testcase:

- Outputs of the CPU under test i.e. *register v0* output and *active* flag
- Memory transactions i.e. read and write operations of the RAM

The two outputs of the CPU can only verify certain aspects of a given instruction such as correct calculations and internal-CPU storage operations. All test cases ensure v0 output is always a non-zero value to ensure no false-positive occurs.

Other more general aspects of the CPU such as correct memory read/writes are verified by comparing the outputs of a modified RAM with the expected outputs in the reference files. The custom RAM imitates the full address 4GB range of memory through mapping and outputs every read and write operation performed. Any unauthorised memory access or writes immediately fails the given testcase. All RAM outputs and reference files are found in the respective folders *"test/4-output"* and *"test/5-reference"*.

Cyclone IV E 'Auto': Area summary

Flow Status:	Successful - Sat Dec 19 13:27:36 2020
Quartus Prime Version:	16.0.0 Build 211 04/27/2016 SJ Lite Edition
Revision Name:	mips_cpu_bus
Top-level Entity Name:	mips_cpu_bus
Family:	Cyclone IV E
Total logic elements:	8,659 / 15,408 (56 %)
Total combinational functions:	8,103 / 15,408 (53 %)
Dedicated logic registers:	1,252 / 15,408 (8 %)
Total registers:	1252
Total pins:	138 / 344 (40 %)
Total virtual pins:	0
Total memory bits:	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements:	16 / 112 (14 %)
Total PLLs:	0 / 4 (0 %)
Device:	EP4CE15F23C6
Timing Models:	Final

Cyclone IV E 'Auto': Timing summary

Fmax:	8.09 MHz
Restricted Fmax:	8.09 MHz
Clock Name:	clk