# A Single Clock Cycle MIPS RISC Processor Design using VHDL

Mamun Bin Ibne Reaz, *MIEEE*, Md. Shabiul Islam, *MIEEE*, Mohd. S. Sulaiman, *MIEEE*
Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia

***Abstract*** **This paper describes a design methodology of a single clock cycle MIPS RISC Processor using VHDL to ease the description, verification, simulation and hardware realization. The RISC processor has fixed-length of 32-bit instructions based on three different format R-format, I-format and J-format, and 32-bit general-purpose registers with memory word of 32-bit. The MIPS processor is separated into five stages: instruction fetch, instruction decode, execution, data memory and write back. The control unit controls the operations performed in these stages. All the modules in the design are coded in VHDL, as it is very useful tool with its concept of concurrency to cope with the parallelism of digital hardware. The top-level module connects all the stages into a higher level. Once detecting the particular approaches for input, output, main block and different modules, the VHDL descriptions are run through a VHDL simulator, followed by the timing analysis for the validation, functionality and performance of the designated design that demonstrate the effectiveness of the design.**

## I. INTRODUCTION

Increasing performance and gate capacity of recent FPGA devices permits complex logic systems to be implemented on a single programmable device. Such a growing complexity demands design approaches, which can cope with designs containing hundreds of thousands of logic gates, memories, high-speed interfaces, and other high-performance components. One category of such design approaches are design methodologies based on language VHDL. It allow designers to develop hardware systems at high level [1].

RISC or Reduced Instruction Set Computer is a design philosophy that become mainstream in the last few years, as the quest for raw speed has dominated the highly competitive computer industry. There is a desire to enhance the processor's speed and simplify the hardware for reasons of cost. RISC design resulted in computers that execute instructions faster than other computers built of the same technology [2].

In a RISC machine, the instruction set is based upon a load/store approach. Only load and store instructions access memory. No arithmetic, logic or I/O instruction operates directly on memory contents. This is the key to single-cycle execution of instructions. The simplification results in an instruction decoder that is small, fast and relatively easy to design [2]. Due to the robust performance, simple instruction formats, breadth of products and depth of support, the MIPS RISC processor architecture becomes the market leader in high performance embedded 32-bit and 64-bit RISC processors [3].

In this paper we present a design study of a single clock cycle MIPS RISC processor using VHDL. The goal of this work was to evaluate the feasibility of using VHDL for rapid design and prototyping of microprocessors. The use of VHDL for modeling is especially appealing since it provides a formal description of the system and allows the use of specific description styles to cover the different abstraction levels (architectural, register transfer and logic level) employed in the design [4].

## II. MATERIALS AND METHODS

The architecture of the MIPS RISC processor is designed based on three MIPS 32-bit instruction formats R-format, I-format and J-format illustrated in Tab. 1. The design of this project consists of 32-bit instructions and 32-bit datapath.

| Field Size | 6-bits | 5-bits | 5-bits | 5-bits | 5-bits | 6-bits |
|---|---|---|---|---|---|---|
| R-Format | Opcode | Rs | Rt | Rd | Shift | Function |
| I-Format | Opcode | Rs | Rt | Address/immediate value | | |
| J-format | Opcode | Branch target address | | | | |

Tab. 1 MIPS 32-bit Instruction Formats.

The implementation of RISC performs fetch, decode, and execute in one clock cycle. The single clock cycle MIPS RISC architecture is separated into five stages: instruction fetch, instruction decode, execute, data memory and write back.

0-7803-7578-5/02/$17.00 ©2002 IEEE

### Instruction Fetch Stage

The first stage of a MIPS RISC is instruction fetch (IF) stage. The IF stage obtain the requested instruction from memory. The operation of the IF stage starts when the program counter (PC) a 32-bit register is sent out to fetch the instruction from memory into the instruction register (IR) and the PC is incremented by 4 through an adder to address the next sequential instruction. The IR is used to hold the instruction needed on subsequent clock cycles. A block diagram of IF stage is shown in Fig. 1.

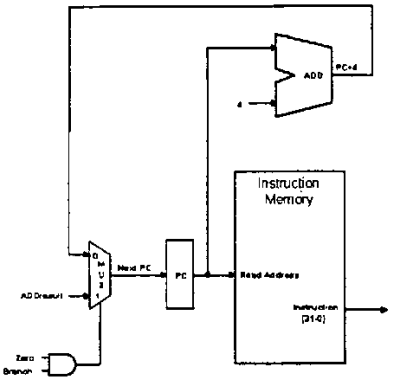



Fig. 1 Instruction Fetch Stage

The branch instruction has three operands, two registers that compare the equality and a 16-bit offset to compute the branch target address relative to the branch instructon address. The branch target address is computed by adding the instruction's sign-extended offset field to the PC.

The AND gate in the instruction fetch cycle is set to 1 when branch and zero are both true and the input Addresult from the ALU unit fed into the PC as the next target address else the normal incremented PC replace the current.

### Instruction Decode Stage

When the instruction is fetched from the IF stage, the instruction's opcode is sent to the control unit and the function code is sent to the ALU. control unit. The instruction's register address fields are used to address the two-port register file. The two-port register performs two independent reads and one write in one clock cycle. Hence implements the decode operation.

In this stage, the instructions are decoded and the register file is accessed to read the registers. The outputs of the general-purpose registers are read into two temporary registers, register 1 and register 2, for use in later clock cycles. The lower 16 bits of the IR are sign-extended and stored into the temporary register IMM, for use in the next cycle. The processor's 32 registers are stored in 'register file' that contains the register state of the machine.

For R-format instructions, there are three register operands. Two data words are read from the register file and one data word is written into the register file for each instruction. Four inputs: three for register numbers and one for data and two outputs (both for data) are used. The register number inputs are 5 bites wide to specify one of the 32 registers, whereas the data input and two data output buses are each 32 bits wide. The block diagram of Instruction decode is shown in Fig. 2 and the format of the three instruction classes of MIPS: the R-type, branch and load/store instructions are tabulated in Tab. 2.

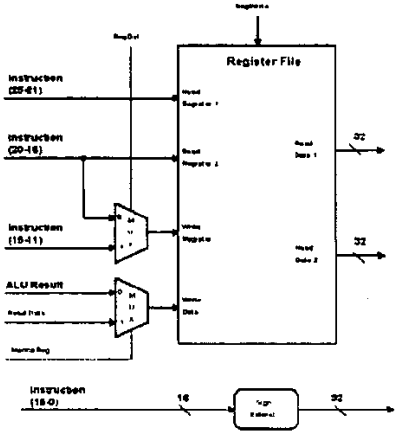



Fig. 2 Architecture of instruction decode stage

| a. R-type instruction | | | | | | |
|---|---|---|---|---|---|---|
| Field | 0 | rs | rt | rd | shamt | funct |
| Bit positions | 31-26 | 25-21 | 20-16 | 15-11 | 10-6 | 5-0 |
| b. Load or Store Instruction | | | | | | |
| Field | 35 or 43 | rs | rt | address | | |
| Bit positions | 31-26 | 25-21 | 20-16 | 15-0 | | |
| c. Branch Instruction | | | | | | |
| Field | 4 | rs | Rt | address | | |
| Bit positions | 31-26 | 25-21 | 20-16 | 15-0 | | |

Tab. 2 The three instruction formats

In the VHDL coding, D flip-flop with enable is used to solve the problem of unstable transients so that it is sensitive to its excitation inputs only during rising or falling of transitions of the clock. In this design, positive edge triggered is used.

### Execution Stage

After the instruction decode stage, the function code is sent to the execution stage. The execution stage performs calculations. The components in this stage are a data ALU and a branch address adder. The ALU is made up of arithmetic, logic and shifting capabilities and the branch address adder is used for PC-relative branch instructions.

To compute the branch target address, the branch datapath in the execution stage includes a sign extension unit and an adder. To perform the comparison, two register operands are fed into the ALU unit with a control set to do a subtraction operation. A block diagram of execution stage shown in Fig. 3.

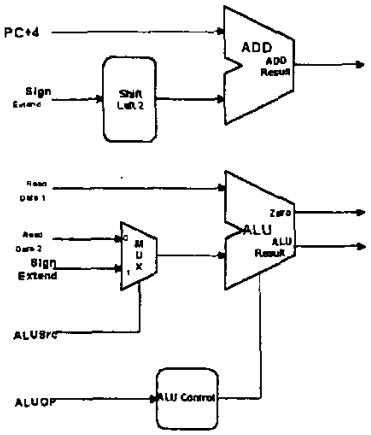



Fig. 3 Architecture of Execution Stage

*The ALU Control*

The ALU control unit has 5-bit instruction function field and a 2-bit control field as input ALUOp. The output of the ALU control unit is a 3-bit signal that directly controls the ALU by generating one of the five 3-bit combinations as shown in Tab. 3.

| ALU control inputs | Function |
|---|---|
| 000 | AND |
| 001 | OR |
| 010 | add |
| 110 | subtract |
| 111 | set on less than |

Tab. 3 The values of three ALU control lines and corresponding ALU operations

Tab. 4 shows the setup of the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code for the R-type instruction and Tab. 5 shows the partial truth table for the three ALU control bits.

| Instruction opcode | ALUOp | Instruction operation | Funct field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|
| LW | 00 | load word | XXXXXX | add | 010 |
| SW | 00 | store word | XXXXXX | add | 010 |
| Branch equal | 01 | branch equal | XXXXXX | subtract | 110 |
| R-type | 10 | add | 100000 | add | 010 |
| R-type | 10 | subtract | 100010 | subtract | 110 |
| R-type | 10 | and | 100100 | and | 000 |
| R-type | 10 | or | 100101 | or | 001 |
| R-type | 10 | set on less than | 101010 | set on less than | 111 |

Tab. 4 Setup of ALU control inputs.

| ALUOp | | Funct Field | | | | | | Operation |
|---|---|---|---|---|---|---|---|---|
| ALUOp1 | ALUOp0 | F5 | F4 | F3 | F2 | F1 | F0 | |
| 0 | 0 | X | X | X | X | X | X | 010 |
| X | 1 | X | X | X | X | X | X | 110 |
| 1 | X | X | X | 0 | 0 | 0 | 0 | 010 |
| 1 | X | X | X | 0 | 0 | 1 | 0 | 110 |
| 1 | X | X | X | 0 | 1 | 0 | 0 | 000 |
| 1 | X | X | X | 0 | 1 | 0 | 1 | 001 |
| 1 | X | X | X | 1 | 0 | 1 | 0 | 111 |

Tab. 5 Truth table of ALU control bits as function of ALUOp and function code field.

The Karnaugh maps (K-maps) are used to further simply the truth table in designing the logic. A partial VHDL code that generate the ALU control bits are shown below:

```
ALUctl(0) <= (Func_op(0) OR Func_op(3)) AND ALUOp1;
ALUctl(1) <= (NOT Func_op(2)) OR (NOT ALUOp1);
ALUctl(2) <= (Func_op(1) AND ALUOp1) OR ALUOp0;
```

*ALU Operations*

To compute the branch target address, the sign extend bits are shifted left two bits. This process is done by concatenating the lower 6 bits of the sign extend bits with $00_2$. Then, these shifted bits are added with the PC+4 from the Instruction Fetch stage.

## Data Memory Stage

The data memory stage stores and loads values to and from memory. This stage is active during the branch instruction. Data Memory takes two inputs for the address and the write data, and one output for the read. Two separate read and write controls either MemWrite or MemRead asserted on any given clock.

*Memory Read*

To load an instruction the read memory control MemRead is asserted to enable the data to be read from the data memory. The data memory takes an address from one of the inputs. The data is then sent out from the data memory to register file through its output Read Data.

While developing the VHDL codes for Data Memory stage, the memory is initialized to some value to ease the simulation process and error checking. A partial code is shown below.

```
address <= readadd(2 downto 0);

mem_out <= mem0 WHEN address=To_stdlogicvector(B"000") ELSE
     mem1 WHEN address=To_stdlogicvector(B"001") ELSE
          mem2 WHEN address=To_stdlogicvector(B"010") ELSE
          To_stdlogicvector(X"FF");
```

*Memory Write*

To store an instruction the write memory control MemWrite is asserted to enable the data to be written into the data memory. The data memory takes an address from one of the inputs.

The data to be written into the data memory is sent into it through the input Write Data.

In VHDL coding, the write control signal of the corresponding memory asserted in order to begin the writing process into the memory. This is done by inserting an AND gate between the memWrite control signal and the address that indicates the memory is to be written. Then, the data on the write data bus is written into the corresponding memory location.

### Write Back Stage

The Write Back stage writes the result of a calculation, memory access or input into the register file. The component multiplexor determines whether the data from data memory or the ALU result from the ALU to be written back into the register file. The operation of this multiplexor is controlled by signal MemtoReg.

The operations involved in this stage are handled by instructions Register-Register ALU instruction and Load instruction. For Register-Register ALU instruction, the MemtoReg is deasserted. For Load instruction, the MemtoReg is asserted. When it is asserted, the value, which comes out from the Read Data of data memory is selected that becomes the input to the write data of the register file. The block diagram of write back stage is shown in Fig. 5.

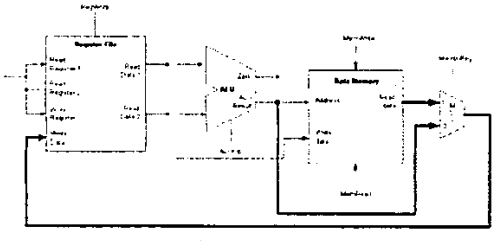


Fig. 5 Architecture of the Write Back Stage.

### Control Unit

In every stage of MIPS RISC, there are some control signals that controls the operations of each of the stages that illustrated in Tab. 6. Tab. 7 shows the effect of each of the seven control signals. The control unit is able to take inputs and generate a write signal for each state element, the selector control for each multiplexor, and the ALU control.

| MIPS RISC Stage | Control Signals |
|---|---|
| Instruction Decode | RegDst |
| | RegWrite |
| Execution | ALUSrc |
| Data Memory | MemWrite |
| | MemRead |
| Write Back | MemtoReg |

Tab. 6 Control Signals in each RISC stage

| Signal Name | Effect when deasserted | Effect when asserted |
|---|---|---|
| RegDst | The register destination number for the Write register comes from the rt field (bits 20-16). | The register destination number for the Write register comes from the rd field (bits 15-11). |
| RegWrite | None | The register on the Write register input is written with the value on the Write data input. |
| ALUSrc | The second ALU operand comes from the second register file output (Read data 2). | The second ALU operand is the sign-extended, lower 16 bits of the instruction. |
| PCSrc | The PC is replaced by the output of the adder that computes the value of PC+4. | The PC is replaced by the output of the adder that computes the branch target. |
| MemRead | None | Data memory contents designated by the address input are put on the Read data output. |
| MemWrite | None | Data memory contents designated by the address input are replaced by the value on the Write data input. |
| MemtoReg | The value fed to the register Write data input comes from the ALU. | The value fed to the register Write data input comes from the data memory. |

Tab. 7 Effect of each of the seven control signals

The different stages of MIPS RISC are combined to make a datapath for MIPS architecture. Fig. 6 shows the datapath obtained by composing the separate stages.

Since the setting of the control lines depends only on the opcode, each control signal is either 0, 1 or don't care (X), for each of the opcode values. Tab. 8 defines the control signals that set for each opcode.

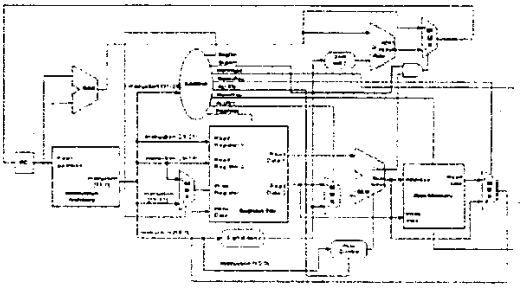


Fig. 6 The Simple Datapath with the Control Unit

| Instruction | Reg Dst | ALUSrc | Memto -Reg | Reg Write | Mem Read | Mem Write | Branch | ALU Op1 | ALU Op0 |
|---|---|---|---|---|---|---|---|---|---|
| R-format | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |

Tab. 8 Setting of the control lines

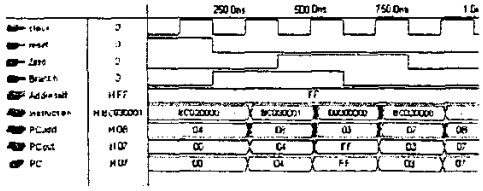
While developing the codes for control unit, the LCELL function is used to isolate logic blocks in the MIPS design. The LCELL component is a buffer that allocates a logic cell for all the files that are associated with the RISC design. The LCELL buffer produces the true and complement of a logic function.

## III. Simulation And Discussion

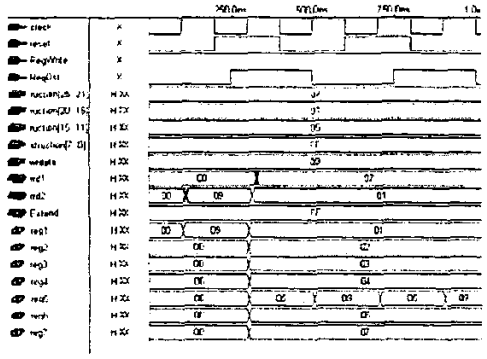
In this project the compilation and simulation of the model is run using MAX+PLUS II version 9.23. Simulation of design is done in a bottom-up fashion to verify the correctness of design. Small

modules are simulated in separate testbenches before they are integrated. There is one testbench for each small module and a testbench for the complete device. Simulated waveforms for each stages are shown below. The simulated results of the top-level module are tabulated in Tab. 9 that shows the correctness of the model.

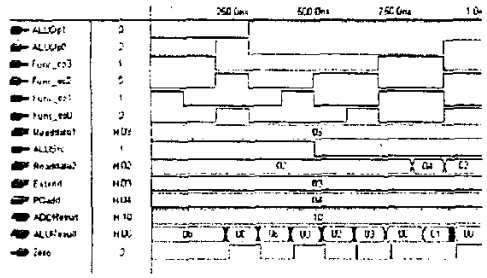
**Simulation of instruction fetch stage:**



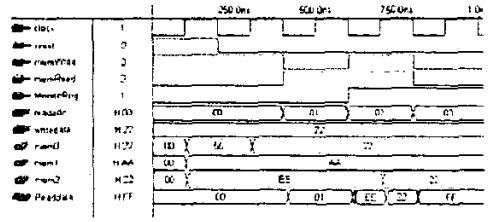
**Simulation of instruction decode stage:**



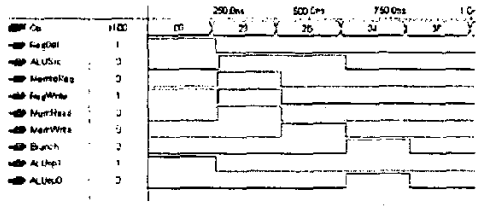
**Simulation waveform of execution stage:**



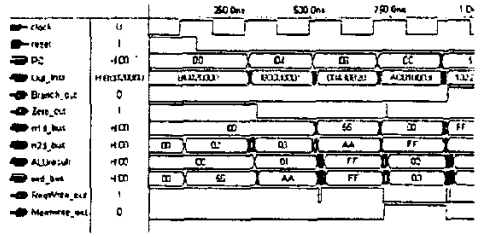
**Simulation of data memory and write back stage:**



**Simulation waveform of control unit:**



**Simulation waveform of top-level module:**



| PC | Inst_bus | Op | Inst | Rsrc1 | RegWrite | MemtoReg | Rs | rs1d | Rt | rs2d | Rd | ALU | Zero | write_back |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 8C020000 | 35 | Lw | 0 | 1 | 1 | Reg0 | 00H | Reg2 | 01H | - | 00H | 1 | 55H |
| 04 | 8C030001 | 35 | Lw | 0 | 1 | 1 | Reg0 | 00H | Reg3 | 01H | - | 01H | 0 | AAH |
| 08 | 00430820 | 00 | R | 0 | 1 | 0 | Reg2 | 55H | Reg3 | AAH | Reg1 | FFH | 0 | FFH |
| 0C | AC010003 | 43 | Sw | 0 | 0 | 1 | Reg0 | 00H | Reg1 | FFH | - | 03H | 0 | FFH |
| 10 | 1022FFF7 | 4 | Beq | 1 | 0 | 0 | Reg1 | FFH | Reg2 | 55H | - | AAH | 0 | AAH |
| 14 | 1001FFFA | 4 | beq | 1 | 0 | 0 | Reg1 | FFH | Reg1 | FFH | - | 00H | 1 | 00H |

Tab. 9 Results of the Top-Level Module simulation.

## IV. CONCLUSION

By simulating with various test vectors a single clock cycle 32-bit MIPS RISC processor using VHDL is successfully designed, implemented and tested. An 8-bit datapath is used with the intention of hardware realization onto Altera FLEX 10K FPGA chip. Currently, we are conducting further research that considers further reductions in the hardware complexity in terms of synthesis and finally download the code into Altera FLEX10K: EPF10K10LC84 FPGA chip on LC84 package for hardware realization.

## REFERENCES

[1]D. Sulik, M. Vasilko, D. Durackova, P. Fuchs, "Design of a RISC Microcontroller Core in 48 Hours", http://www.itpapers.com/cgi/PSummaryIT.pl ?paperid=12747&scid=108 (current Sep. 30, 2002).

[2]Charles E. Gimarc, Veljko M. Milutinovic, "RISC Principles, Architecture, and Design", Computer Science Press Inc., 1989.

[3]White paper, "Wide range of comprehensive tools speed, Development of high-performance embedded system", MIPS Technologies Inc., http://www.mips.com/whitepapers/030399WP1.html (current Sep. 30, 2002).

[4]S. Chen, B. Mulgrew, and P. M. Grant, "A clustering technique for digital communications channel equalization using radial basis function networks," *IEEE Trans. Neural Networks*, vol. 4, pp. 570–578, July 1993.