

## Area Optimized Implementation For Mips Processor

Dasari Kamala Kumari<sup>1</sup>, Popuri Venkateswarlu<sup>2</sup>

<sup>1</sup> PG Student (M. Tech), Dept. of ECE, Chirala Engineering College, Chirala., A.P, India.

<sup>2</sup> Assistant Professor, Dept. of ECE, Chirala Engineering College, Chirala., A.P, India.

---

**Abstract:** The implementation of 32 bit RISC processor with microprocessor without interlocked pipeline stages (MIPS) is presented. It was implemented in VHDL so as to reduce the instruction set present in the programmable memory. As the result the processor will contain the necessary logics for the implementation that requires fewer gates to be synthesized in the programmable matrix and has the capability to increase the speed of the target processor with reduced memory. In this paper we propose a novel technique of run-time loading of machine code for MIPS-32 soft-core processor. As we know, implementing fewer instructions on silicon reduces the complexity of the instruction decoder, the addressing logic, and the execution unit. This allows the machine to be clocked at a faster speed, since less work needs to be done each clock period. In this paper we used Xilinx-ISE tool for logical verification, and further synthesizing it on Xilinx-ISE tool using target technology and performing placing & routing operation for system verification.

**Keywords:** MIPS, Data Flow, Pipeline, area optimization

---

### I. Introduction

The idea of this project was to create a microprocessor without interlocked pipeline stages (MIPS) as a building block in which can be later included in larger design. This can be implemented to the system where a problem is easy to solve in software. A Finite state machine is designed in such a way to reduce the computation problems. However, at a high level of complexity it is easier to implement the function in software.

Typical processors incorporate arithmetic and logic functional units as well as the associated control logic, instruction processing circuitry, and a portion of the memory hierarchy. Portions of the interface logic for the input/output (I/O) and memory subsystems may also be infused, allowing cheaper overall systems. While many processors and single-chip designs, some high-performance designs rely on a few chips to provide multiple functional units and relatively large caches.

#### Operation of Processor:

The processor is an electronic circuit that operates at the speed of an internal clock thanks to a quartz crystal that, when subjected to an electrical current, send pulses, called "peaks". The clock speed (also called cycle), corresponds to the number of pulses per second, written in Hertz (Hz). Thus, a 200 MHz computer has a clock that sends 200,000,000 pulses per second. Clock frequency is generally a multiple of the system frequency, meaning a multiple of the motherboard frequency.

With each clock peak, the processor performs an action that corresponds to an instruction or a part thereof. A measure called CPI (Cycles per Instruction) gives a representation of the average number of clock cycles required for a processor to execute an instruction. A processor power can thus be characterized by the number of instructions per second that it is capable of processing. MIPS (millions of instructions per second) is the unit used and corresponds to the processor frequency divided by the CPI.

### II. MIPS FEATURES:

Processors are much faster than memories. For example, a processor clocked at 100 MHz would like to access memory in 10 nanoseconds, the period of its 100 MHz clock. Unfortunately, the memory interfaced to the processor might require 60 nanoseconds for an access. So, the processor ends up waiting during each memory access, wasting execution cycles.

To reduce the number of accesses to main memory, designers added instruction and data cache to the processors. A cache is a special type of high speed RAM where data and the address of the data are stored. Whenever the processor tries to read data from main memory, the cache is examined first. If one of the addresses stored in the cache matches the address being used for the memory read (called a hit), the cache will supply the data instead. Cache is commonly ten times faster than main memory, so you can see the advantage of getting data in 10 nanoseconds instead of 60 nanoseconds. Only when we miss (i.e., do not find the required data in the cache), does it take the full access time of 60 nanoseconds. But this can only happen once. Since a copy of the new data is written into the cache after a miss. The data will be there the next time we need it. Instruction cache is used to store frequently used instructions. Data cache is used to store frequently used data.

Implementing fewer instructions and addressing modes on silicon reduces the complexity of the instruction decoder, the addressing logic, and the execution unit. This allows the machine to be clocked at a faster speed, since less work needs to be done each clock period.

RISC typically has large set of registers. The number of registers available in a processor can affect performance the same way a memory access does. A complex calculation may require the use of several data values. If the data values all reside in memory during the calculations, many memory accesses must be used to utilize them. If the data values are stored in the internal registers of the processor instead, their access during calculations will be much faster. It is good then to have lot of internal registers.

### **Comparative Studies of CISC and RISC:**

#### **CISC:**

CISC is an acronym for complex instruction set computer. The reasons to have the large instruction set are:

- CISC systems are complex as the architecture and the instructions are modified as per the technology and hence there is a requirement for more number of instructions and therefore will have many addressing modes.
- The main function of any microprocessor is to fetch the data and process it. The chip as such doesn't have any provision for memory i.e., memory is external. As process needs to access data from a memory or from I/O peripherals there is a necessary for more instructions.

#### **Drawbacks behind having the large instruction set:**

- The large instruction set is responsible for a very complicated control logic that generates a host of control signals and interrupts that are important for the working of the processor.
- This large instruction set creates problems for the compiler as it has to write code for each individual instruction. It becomes difficult to debug if there is any mistake in the code of the compiler.
- The Large instruction set has a very complicated decoding logic which is responsible for a more number of stages or for an increased propagation delay of the control logic. This is because an increase in the number of stages leads to proportionate increase in the number of gates. Transistor has a rise time, a fall time contributing to the propagation delay of the signal. So at every gate the signal gets delayed and this cumulative delay in propagation adds up, to decrease the processing speed of the data.
- They have a variety of instruction formats like one byte, two byte, three byte. They have a large number of addressing modes.
- In general the pipelining is used to increase the speed of memory operations with separate control over the bus. They are difficult to handle if there is a considerable difference between instruction lengths and execution cycle of different instructions then the pipeline design will be much more complicated.

#### **RISC (Reduced Instruction Set Computer):**

The advantages of instruction set of RISC are as follows

- In the RISC based system the frequently used instructions are hardware realized and it is therefore minimizes the memory access time and the decoding time.
- It is easy to write code for fixed instruction set, so the compiler supports for efficient translation of high level language programs into machine language programs.
- As the architecture supports pipeline in different segments, it may be possible to execute number of instructions in a single instruction cycle.
- Memory access is limited to load accumulator and store accumulator instructions only.

### **III. The Mips Processor**

The MIPS instruction set architecture (ISA) is a RISC based microprocessor architecture that was developed by MIPS Computer Systems Inc. in the early 1980s. MIPS is now an industry standard and the performance leader within the embedded industry. Their designs can be found in Canon digital cameras, Windows CE devices, Cisco Routers, Sony Play Station 2 game consoles, and many more products used in our everyday lives. By the late 1990s it was estimated that one in three of all RISC chips produced was a MIPS-based design. MIPS RISC microprocessor architecture characteristics include: fix-length straightforward decoded instruction format, memory accesses limited to load and store instructions, hardwired control unit, a large general purpose register file, and all operations are done within the registers of the microprocessor. Due to these design characteristics, computer architecture courses in university and technical schools around the world often study the MIPS architecture. One of the most widely used tools that helps students understand MIPS is SPIM (MIPS spelled backwards) a software simulator that enables the user to read and write MIPS assembly language programs and execute them. SPIM is a great tool because it allows the user to execute programs one step or instruction at a time. This then allows the user to see exactly what is happening during their program

execution. SPIM also provides a window displaying all general purpose registers which can also be used during the debug of a program. This simulator is another impressive tool that gives the computer architecture students an opportunity to visually observe how the MIPS processor works.

**MIPS Instruction Set Architecture:**

As mentioned before MIPS is a RISC microprocessor architecture. The MIPS Architecture defines thirty-two, 32-bit general purpose registers (GPRs). Register \$r0 is hard-wired and always contains the value zero. The CPU uses byte addressing for word accesses and must be aligned on a byte boundary divisible by four (0, 4, 8, ...). MIPS only has three instruction types: I-type is used for the Load and Stores instructions, R-type is used for Arithmetic instructions, and J-type is used for the Jump instructions as shown in Figure1, Table1 provides a description of each of the fields used in the three different instruction types.

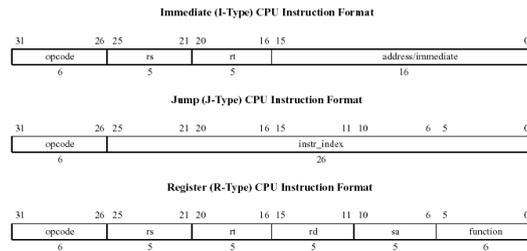
Field	Description
<i>opcode</i>	6-bit primary operation code
<i>rd</i>	5-bit specifier for the destination register
<i>rs</i>	5-bit specifier for the source register
<i>rt</i>	5-bit specifier for the target (source/destination) register or used to specify functions within the primary <i>opcode</i> REGIMM
<i>address/immediate</i>	16-bit signed <i>immediate</i> used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
<i>instr_index</i>	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
<i>sa</i>	5-bit shift amount
<i>function</i>	6-bit function field used to specify functions within the primary <i>opcode</i> SPECIAL

**Table 1 MIPS Instruction Fields**

MIPS is a load/store architecture, meaning that all operations are performed on operands held in the processor registers and the main memory can only be accessed through the load and store instructions (e.g lw, sw). A load instruction loads a value from memory into a register. A store instruction stores a value from a register to memory. The load and store instructions use the sum of the offset value in the address/immediate field and the base register in the \$rs field to address the memory. Arithmetic instructions or R-type include: ALU Immediate (e.g. addi), three-operand (e.g. add, and, slt), and shift instructions (e.g. sll, srl). The J-type instructions are used for jump instructions (e.g. j). Branch instructions (e.g. beq, bne) are I-type instructions which use the addition of an offset value from the current address in the address/immediate field along with the program counter (PC) to compute the branch target address; this is considered PC-relative addressing. Table 2 shows a summary of the core MIPS instructions.

Instruction	Symbol	Format	Example	Meaning	Comments
Add	add	R	add \$r1, \$r2, \$r3	$\$r1 = \$r2 + \$r3$	overflow detected
Add Immediate	addi	I	addi \$r1, \$r2, 100	$\$r1 = \$r2 + 100$	plus constant
Add Unsigned	addu	R	addu \$r1, \$r2, \$r3	$\$r1 = \$r2 + \$r3$	overflow undetected
Subtract	sub	R	sub \$r1, \$r2, \$r3	$\$r1 = \$r2 - \$r3$	overflow detected
Subtract Unsigned	subu	R	subu \$r1, \$r2, \$r3	$\$r1 = \$r2 - \$r3$	overflow undetected
And	and	R	and \$r1, \$r2, \$r3	$\$r1 = \$r2 \& \$r3$	bitwise logical and
Or	or	R	or \$r1, \$r2, \$r3	$\$r1 = \$r2   \$r3$	bitwise logical or
Shift Left Logical	sll	R	sll \$r1, \$r2, 10	$\$r1 = \$r2 \ll 10$	shift left by constant
Shift Right Logical	srl	R	srl \$r1, \$r2, 10	$\$r1 = \$r2 \gg 10$	shift right by constant
Set Less Than	slt	R	slt \$r1, \$r2, \$r3	if ( $\$r2 < \$r3$ ) $\$r1 = 1$ else 0	compare less than
Load Word	lw	I	lw \$r1, 100(\$r2)	$\$r1 = \text{mem}(\$r2 + 100)$	load word from mem to reg
Store Word	sw	I	sw \$r1, 100(\$r2)	$\text{mem}(\$r2 + 100) = \$r1$	store word from reg to mem
Branch on Equal	beq	I	beq \$r1, \$r2, 25	if ( $\$r1 = \$r2$ ) goto $\text{PC} + 4 + 100$	equal test
Branch on Not Equal	bne	I	bne \$r1, \$r2, 25	if ( $\$r1 \neq \$r2$ ) goto $\text{PC} + 4 + 100$	not equal test
Jump	j	J	j 100	goto 400	jump to target address

**Table 2 MIPS Core Instructions**



**Figure 1** MIPS Instruction Types

**MIPS SINGLE-CYCLE PROCESSOR**

The MIPS single-cycle processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write-back all in one clock cycle. First the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed. This instruction is then divided into the different fields shown in Table 1. The instructions opcode field bits [31-26] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address fields \$rs bits [25 - 21], \$rt bits [20 - 16], and \$rd bits [15-11] are used to address the register file. The register file supports two independent register reads and one register write in one clock cycle. The register file reads in the requested addresses and outputs the data values contained in these registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g. load or store), compute an arithmetic result (e.g. add, and or slt), or perform a compare (e.g. branch). If the instruction decoded is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

**MIPS PROCESSOR IMPLEMENTATION**

The processor is broken up into five stages: instruction fetch (IF), instructions decode (ID), execute (EX), data memory (M), and write-back (WB).

**Instruction Fetch**

The IF stage involves keeping track of the current/next instruction as well as retrieve the current instruction from memory. In this scenario, memory is split into separate instruction and data memories in order to avoid a structural hazard. That is, simultaneous access to memory, one for instructions and the other for data, is possible in the architecture shown in Figure 1.

**Instruction Decode**

On the next cycle, the fetched instruction moves into the ID stage. There, the instruction is broken up into several fields and inputs into the control logic and register file. Various control signals, register values, and intermediate values are handed to the EX stage where arithmetic operations are performed (in this case, integer add and subtract). In addition, the registers' addresses will be forwarded to the hazard detector in this stage. If there is potential hazard in the system, this stage will perform a stall.

**IV. Results & Conclusions**

In this research, we adopt top-down design method and use VHDL to describe system. At first, we design the system from the top, and do in-depth design gradually. The structure and hierarchical of design is very clear. It is easy to edit and debug. Design of instruction fetch (IF) stage simulates, integrate and routes on Quartus II 4.3. The result indicates IF stage completes prospective function. Figure 2 shows the simulation result of the designed processor and figure 3 & 4 shows the RTL schematic of the designed module. The architecture of the designed MIPS processor is shown in Figure 5. Table 3 shows The device utilization summary of MIPS Processor which suggests that this device is outperforming the earlier processors.

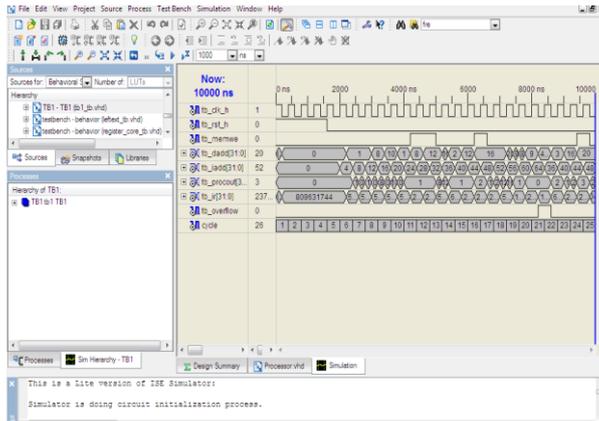


Figure 2 Simulation Results

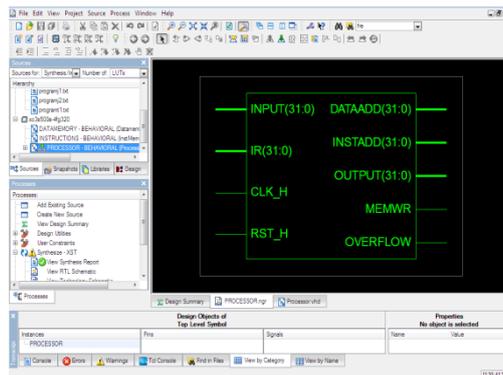


Figure 3 RTL Schematic

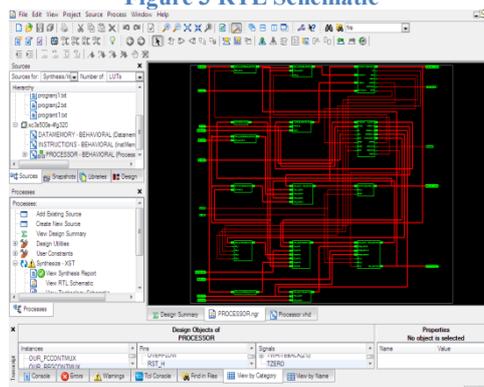


Figure 4 RTL Schematic Internal Section

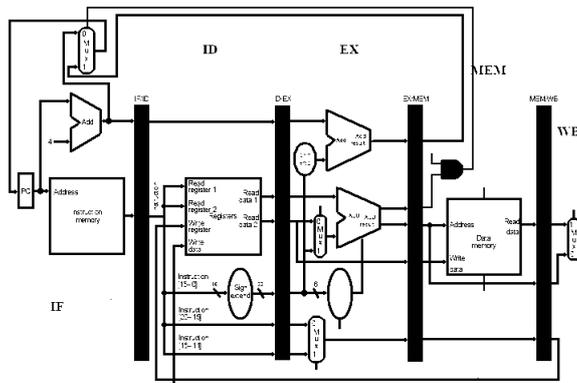


Figure 5 Architecture of the MIPS Processor

<b>Table-3</b>			
<b>Device Utilization Summary (estimated values)</b>			
<b>Logic Utilization</b>	<b>Used</b>	<b>Available</b>	<b>Utilization</b>
Number of Slices	1478	4656	31%
Number of Slice Flip Flops	896	9312	9%
Number of 4 input LUTs	2500	9312	26%
Number of bonded IOBs	164	232	70%
Number of GCLKs	1	24	4%

### **Acknowledgements**

The authors would like to thank the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of this paper.

### **References:**

- [1] Wang-AiYing, Organization and Structure of Computer, Tsinghua University Press, 2006.
- [2] Wang-Yuan Zhen, IBM-PC Macro Asm Program, Huazhong University of Science and Technology Press, 1996.9
- [3] Bai-ZhongYing, Computer Organization, Science Press, 2000.11.
- [4] MIPS Technologies, Inc. MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set June 9, 2003.
- [5] Zheng-WeiMin, Tang-ZhiZhong. Computer System Structure (The second edition), Tsinghua University Press, 2006.
- [6] Pan-Song, Huang-JiYe, SOPC Technology Utility Tutorial, Tsinghua University Press, 2006.
- [7] MIPS32 4KTMPProcessor Core Family Software User's Manual, MIPS Technologies Inc. [M]. BeiJing, Engine Industry Press. 2003.
- [8] "Rapid Prototyping of digital Systems, a tutorial approach "By – James O. Hamblen and Michael D. Furman.
- [9] Yi-Kui, Ding-YueHua, Application of AMCCS5933 Controller in PCI BUS, DCABES2007, 2007.7.759

### **Authors Profile:**



**Dasari Kamala Kumari** is Pursuing her M. Tech from Chirala Engineering College, Chirala in the department of Electronics & Communication Engineering (ECE) with specialization in VLSI & Embedded systems.



**Popuri Venkateswarlu** is working as an Associate Professor in the department of Electronics & Communication Engineering in Chirala Engineering College, Chirala. He has completed masters from JNTUK. He has over 6 years of teaching experience.