ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Design and Simulation of 32-Bit Block Processor

Kunam Bhagya Lakshmi
M.Tech VLSI Design,
Vidya Jyothi Institute of Technology.

Sapati Upender, M.Tech, (Ph.D)
Associate Professor,
Vidya Jyothi Institute of Technology.

Dr.K.Manjunathachari
Professor & HOD,
Dept of ECE
GITAM University.

Abstract:

In this paper we propose a novel technique of run-time loading of machine code for 32-bit block processor. As we know, 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. Our proposed RISC block Processor technique sends the machine code to the instruction memory of the soft-core from the software tool through UART. The user should use that software tool to write assembly code, debug the code and generate the machine code. Also, the software tool is used for establishing UART connection.

Keywords:

MIPS, Data Flow, Data Path, Pipeline, RISC, CISC.

1. Introduction:

Processors are regarded as one of the most important devices in our everyday machines called computers. Before we start, we need to understand what exactly processors are and their appropriate implementations. Processor is an electronic circuit that functions as the central processing unit (CPU) of a computer, providing computational control. Processors are also used in other advanced electronic systems, such as computer printers, automobiles, and jet airliners, Calculators and etc. 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. Processors have been described in many different ways. They have been compared with the brain and the heart of humans. Their operation has been linked to a switched board, and to the nervous system in an animal. They have often been called microcomputers. The original purpose of the processor was to control memory. That is what they were originally designed to do, and that is what they do today. Specifically, a processor is “a component that implements memory.” 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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

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.

2. THE BLOCK PROCESSOR

The BLOCK instruction set architecture (ISA) is a RISC based microprocessor architecture that was developed by BLOCK Computer Systems Inc. in the early 1980s. BLOCK 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 BLOCK-based design. Architecture of BLOCK RISC microprocessor includes, 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 BLOCK architecture. One of the most widely used tools that helps students understand BLOCK is SPIM (BLOCK spelled backwards) a software simulator that enables the user to read and write BLOCK 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 BLOCK processor works.

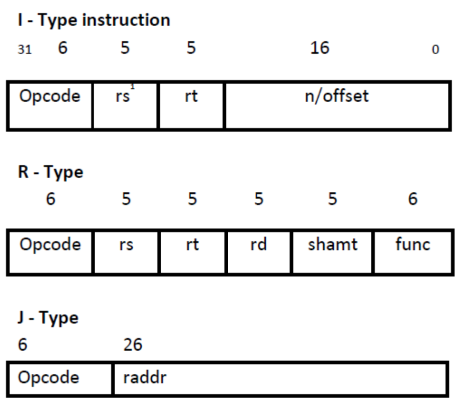


Figure 1 Instruction Formats

As mentioned before BLOCK is a RISC microprocessor architecture. The BLOCK Architecture defines 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, …). BLOCK 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 Figure 1 which provides a description of each of the fields used in the three different instruction types. BLOCK 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.

3. BLOCK PROCESSOR IMPLEMENTATION

The BLOCK 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 fig. 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.

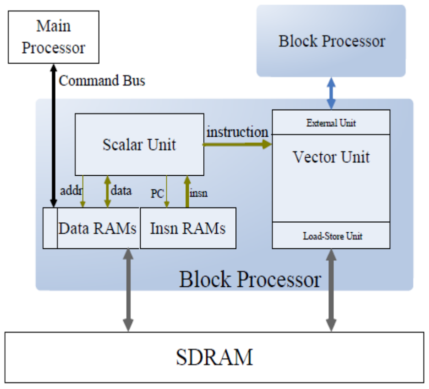


Figure 2.The BLOCK Processor

The initial task of this paper was to implement in VERILOG HDL the BLOCK single-cycle processor .A good VERILOG HDL reference and tutorial can be found in the appendices to the book Fundamentals of Digital Logic with VERILOG HDL Design by Stephen Brown and Zvonko Vranesic [2]. The IEEE Standard VERILOG HDL Language Reference Manual [3], also helped in the overall design of the VERILOG HDL implementation. The first part of the design was to analyze the single-cycle datapath and take note of the major function units and their respective connections. The BLOCK implementation as with all processors, consists of two main types of logic elements: combinational and sequential elements. Combinational elements are elements that operate on data values, meaning that their outputs depend on the current inputs. Such elements in the BLOCK implementation include the arithmetic logic unit (ALU) and adder. Sequential elements are elements that contain a hold state. Each state element has at least two inputs and one output. The two inputs are the data value to be written and a clock signal. The output signal provides the data values that were written in an earlier clock cycle. State elements in the BLOCK implementation include the Register File, Instruction Memory, and Data Memory as seen in Figure 2. While many of logic units are straightforward to design and implement in VERILOG HDL, considerable effort was needed to implement the state elements.

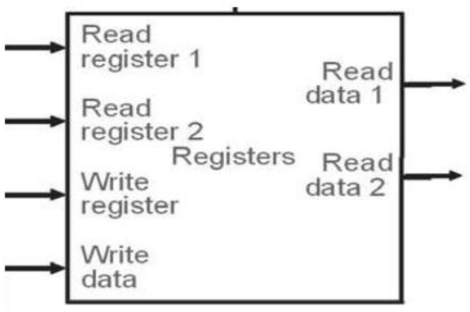


Figure 3 BLOCK Register File

INSTRUCTION FETCH UNIT

The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction as shown in Figure 4. Since this design uses an 8-bit data width we had to implement byte addressing to access the registers and word address to access the instruction memory. The instruction fetch component contains the following logic elements that are implemented in VERILOG HDL: 8-bit program counter (PC) register, an adder to increment the PC by four, the instruction memory, a multiplexer, and an AND gate used to select the value of the next PC.

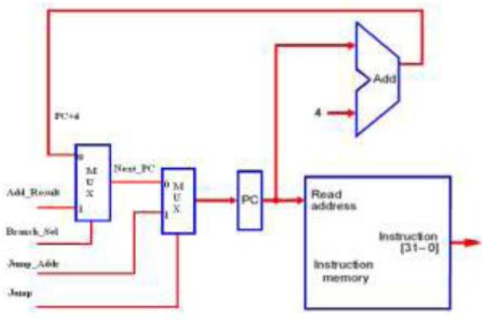
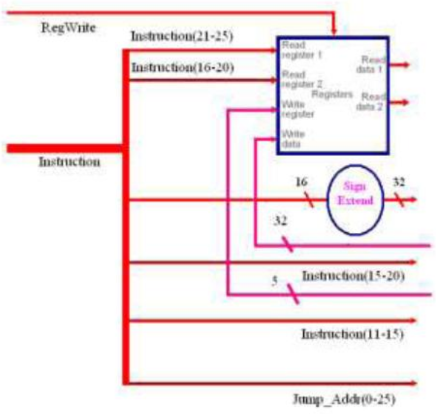


Figure 4 Instruction Fetch Unit

INSTRUCTION DECODE UNIT

The main function of the instruction decode unit is to use the 32-bit instruction provided from the previous instruction fetch unit to index the register file and obtain the register data values as seen in Figure 5. This unit also sign extends instruction bits [15 - 0] to 32-bit.

However with our design of 8-bit data width, our implementation uses the instruction bits [7 – 0] bits instead of sign extending the value. The logic elements to be implemented in VHDL include several multiplexers and the register file that was described earlier.



THE CONTROL UNIT

The control unit of the MIPS singlecycle processor examines the instruction opcode bits [31 – 6] and decodes the instruction to generate nine control signals to be used in the additional modules as shown in Figure 6. The RegDst control signal determines which register is written to the register file. The Jump control signal selects the jump address to be sent to the PC. The Branch control signal is used to select the branch address to be sent to the PC. The MemRead control signal is asserted during a load instruction when the data memory is read to load a register with its memory contents. The MemtoReg control signal determines if the ALU result or the data memory output is written to the register file. The ALUOp control signals determine the function the ALU performs. (e.g. and, or, add, sbu, slt) The MemWrite control signal is asserted when during a store instruction when a registers value is stored in the data memory. The ALUSrc control signal determines if the ALU second operand comes from the register file or the sign extend. The RegWrite control signal is asserted when the register file needs to be written.

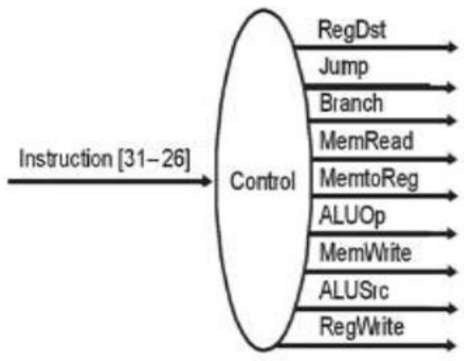
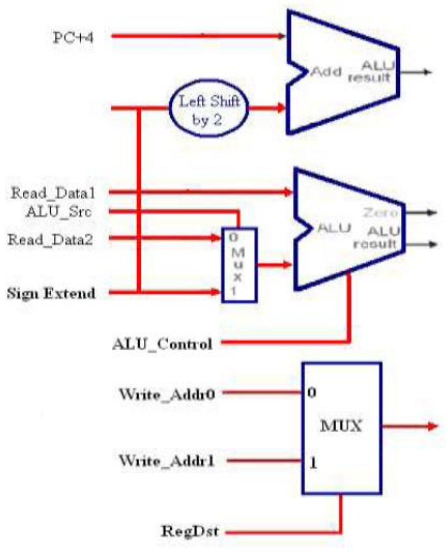


Figure 6 MIPS Control Unit

Execution Unit

The execution unit of the MIPS processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUop signal. The branch address is calculated by adding the PC+4 to the sign extended immediate field shifted left 2 bits by a separate adder. The logic elements to be implemented in VHDL include a multiplexer, an adder, the ALU and the ALU control as shown in Figure 2 & 7



DATA MEMORY UNIT

The data memory unit is only accessed by the load and store instructions. The load instruction asserts the MemRead signal and uses the ALU Result value as an address to index the data memory.

The read output data is then subsequently written into the register file. A store instruction asserts the MemWrite signal and writes the data value previously read from a register into the computed memory address. The VHDL implementation of the data memory was described earlier. Figure 8 shows the signals used by the memory unit to access the data memory.

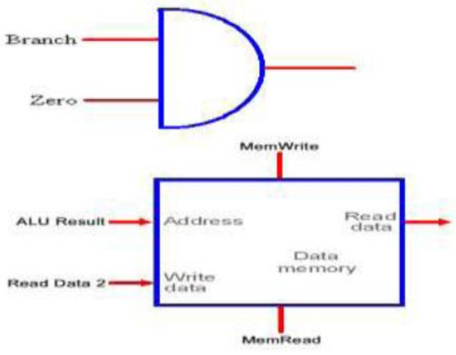


Figure 8 MIPS Data Memory Unit

4 . Simulation Results

The results obtained from the Xilinx ISE 12.2 which are implemented using Verilog HDL.

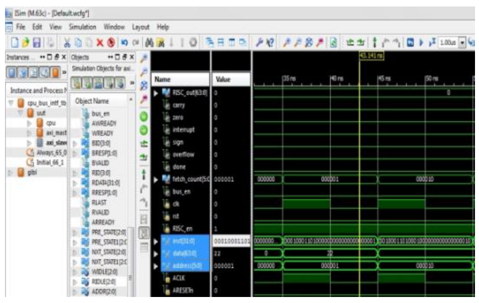


Figure 4.1 : Loading instructions

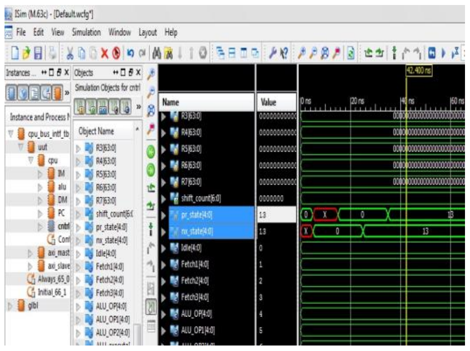


Figure 4.2: Loading instruction into Memory

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

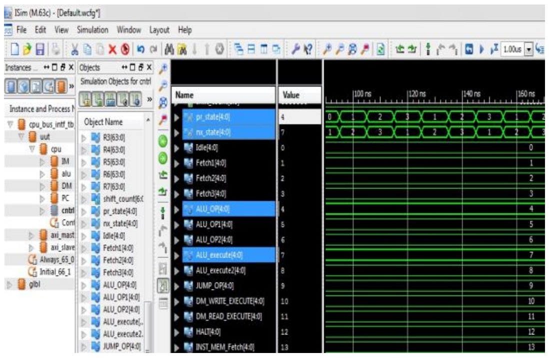


Figure 4.3: Executing ALU operation

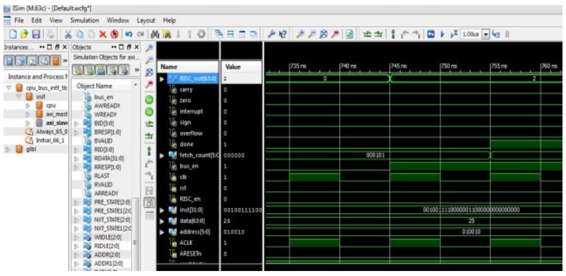


Figure 4.4: Block processor Output

5. Conclusion:

The work presented in this paper describes a functional implementation design of a Block Processor and pipelined processor designed using Verilog HDL. The results show first the instruction memory initialization, which is used to fill the instruction memory with the instructions to be executed, which are indexed by the program counter (PC). The second is the actual 32-bit instruction represented using hexadecimal numbers. The third is the PC value used to index the instruction memory to retrieve an instruction. From the results we have observed that the Block processor executes the instructions very fast.