International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 8, October - 2012

32-Bit Risc Processor For Computer Architecture

Bighneswar Panda.
M.Tech student

B. Vijay Bhaskar
HOD, Dept. ECE

R. Surya Prakash
Asst. Prof., Dept. ECE

Avanti's St. Theressa Institute Engg & Tech, Chipurupally, Vizianagaram(Dist), AP

Abstract:

Now a days computers are used everywhere in our day to day lives ranging from electronics instruments to industrial process automation. Due to complexity of new applications computer engineers use embedded systems to develop high performance technological systems which can achieve high speed processing while used as hardware resources efficiently. To develop embedded systems, it is necessary to know the basic operation of a computer system, which is generally composed of memory, a peripheral controller and a microprocessor. This work presents the design and implementation of a 32 bit RISC processor intended for computer architecture. This RISC processor should support a specific instruction set and its own assembly code with proper instruction format. The instruction set should have simplicity and robustness features. It is considered to be an effective solution for computer compression. Final circuit can be used as a soft core for FPGA embedded designs to control and automation applications.

***Keywords**: Harvard architecture, PC, Subroutine, Digital Logic, Microprocessor, ALU, Computer Architecture, Programmable Logic, Opcode.*

I. INTRODUCTION

A digital system was developed by using a set of interconnected digital integrated circuits like counters, buffers, logic gates and memory. It requires lots of analysis, testing and to adapt the design metrics like speed, response time, power consumption etc. which resulted very difficult for development. Also every design change implied a whole analysis and most times expensive to upgrade.

Now-a-days, advanced technologies like programmable logic as Complex Programmable Logic Devices (CPLD) or Field Programmable Gate Arrays (FPGA) with more sophisticated simulation and design verification environments enable engineers to reach new levels of complexity and robustness, while greatly reducing the time between development and implementation. This way, one can develop a system which can be optimised for manufacturing on a single chip, with the capacity to add or remove modules according to the requirements in the future. Modern processor design sometimes reduce the implementation effort by acquiring some of these elements as Intellectual Property (IP) or through implementation techniques to build the other components, like VHDL or Verilog language and a proprietary synthesizer depending on each hardware vendor.

There exist different approaches to obtain high comprehension when explaining computer architecture. One approach uses FPGA devices and VHDL language to construct a simple computer, but there is disadvantage that by programming there is a lack of understanding in dataflow. Another approach uses MSI digital components such as TTL to construct a computer by interconnecting the components but sometimes using wires to build those connections is time-consuming.

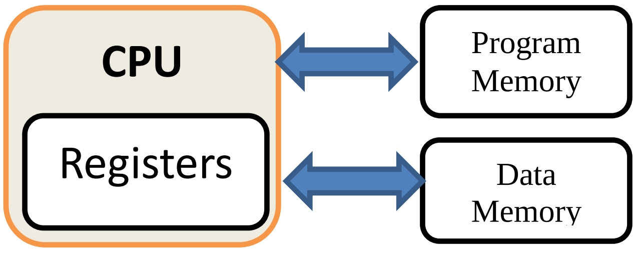
The design of a 32-bit data width Reduced Set Instruction Computer (RISC) processor was developed with simplicity and implementation efficiency. It has a complete instruction set, program and data memories, general purpose registers and a simple arithmetic logical unit (ALU) for basic operations. It operates following a multi-cycle execution nature and is implemented on a XILINX Spartan-3e FPGA.

II. CPU ARCHITECTURE

The processor is based on the Harvard architecture that any instruction occupies separated positions of program memory and data memory. Thus obtaining greater speed and a minor program length, also, the access time to the instructions can be superposed with one of the data, obtaining a greater speed in each operation.

The processor includes a RISC instruction set and uses a Single Instruction – Single Data (SISD) execution order. Its main characteristics are:

- Eight 32 – bit general purpose registers.
- 256 allocation of 32-bit wide ROM Program memory
- 256 allocations of 32-bit wide RAM data memory
- ALU with basic arithmetic and logical operations.



`Fig 1: The Harvard computer architecture composed of: Program and Data memories and the CPU interconnected by buses.

III. INSTRUCTION SET

To design a CPU, a specific RISC instruction set and its own assembly code with its proper instruction format is necessary. The instruction set should have two objectives: simplicity and robustness, i.e. using the simplest instructions that make the processor capable of executing complex operations. The instructions are classified into three groups:

- Operations(Arithmetic and Logical)
- Program Control(Jumps)
- Data Manipulation (Load and Storage)

The arithmetic operations are addition and subtraction, because it is possible to perform any other operation with just these two. These operations can be performed between two registers and between one register and an immediate data.

The logical operations are AND, OR, NOT, Shift Right and Shift Left. The first two operations are executed between two registers, while the other operations are applied to only one register called source register.

The branch instructions include

- An immediate jump, a jump to a value contained in a register and
- Conditional branches

The execution of conditional branches depends on the status of the proper flag in the status register.

The load and storage instructions required must address different data sources on an immediate, direct to data memory and indirect to data memory, addressing modes. Following table describes the complete instruction set, showing the mnemonic, description, syntax and micro-operation done for every instruction.

The instruction format was developed considering many factors like the CPU architecture, the number of instructions and operands involved. Each instruction has its own OPCODE (Operation Code), which is analogy to the assembly mnemonic in machine language.

TABLE 1: INSTRUCTION SET

Mnemonic	Description	Syntax	Micro-operation
ADD	Addition	ADD Rd, Rs	Rd←Rd + Rs
SUB	Subtraction	SUB Rd, Rs	Rd←Rd – Rs
ADDI	Immediate addition	ADDI Rd, k	Rd←Rd + k
SUBI	Immediate Subtraction	SUBI Rd, k	Rd←Rd – k
AND	Logic AND	AND Rd, Rs	Rd←Rd . Rs
OR	Logic OR	OR Rd, Rs	Rd←Rd OR Rs
NOT	Logic NOT	NOT Rd	Rd←NOT(Rd)
SHL	Shift register left	SHL Rd	Rd(n+1) ←Rd(n), Rd(0) ←0
SHR	Shift register right	SHR Rd	Rd(n) ←Rd(n+1), Rd(7) ←0
JMP	Immediate Jump		PC ← PC + k
JMR	Jump to register	JMR Rd	PC ← Rd
BRC	Branch if carry	BRC k	if (c=1) then PC ← PC + k
BRZ	Branch if zero	BRZ k	if (c=1) then PC ← PC + k
BRH	Branch if half-carry	BRH k	if (c=1) then PC ← PC + k
LDI	Load immediate	LDI Rd, k	Rd ← k
LDD	Direct load from memory	LDD Rd,[A]	Rd ← [A]
LDX	Indirect load from memory	LDX Rd,[Rs]	Rd ← [Rs]
STD	Direct storage to memory	STD [A],Rd	[A] ← Rs
STX	Indirect storage to memory	STX [Rd],Rd	[Rd] ← Rs
LDP	Load from PC	LDP Rd	Rd ← PC

Rd: Destination Register; Rs: Source Register; k: Constant; PC : Program Counter; A : address

Instructions are reclassified according to type of operands as follows:

- Type J (Jumps): only use the OPCODE and a constant value (k);
- Type I (Immediate data): use the OPCODE, a density (Rd) or source (Rs) register and a constant value(k);
- Type R (Register operation): require a single destination register or both: destination and source register.

Type J:

OPCODE	Not Used	K

Type I:

OPCODE	Rd	k/A

Type R:

OPCODE	Rd	Rs	Not Used

Fig 2. Instruction format per instruction type.

To specify a register (source or destination) 3 bits are used since the CPU has 8 general purpose registers. To specify a constant value, 8 bits are used. Instruction format is standardized to 32 bits. Fig. 2 shows the instruction format for each type.

IV. HARDWARE IMPLEMENTATION

- **Program Counter (PC)**

The program counter produces the address to read instruction from the program memory. It can load a random address if the program requires loops or branches.

- **ROM Program Memory**

The program memory stores the instructions to be executed. It is to be non-volatile and fast. It uses internal ROM as program memory, because it was the fastest option and eliminated the need for external storage. This memory was built with 256x1 bit ROM blocks, with a total of 32 blocks to store 256 32-bit instructions.

- **Instruction Register and Decoder**

Instruction registers stores the instructions read from the program memory and keep it as an output for the decoder, which separates the operation code and operands to execute the command. Two 32-bit registers (D-type flip-flops) were used, one for the higher word and one for the lower word (32-bit instruction). The instruction decoder deal with the raw data stored in the instruction registers and separates it in parts: Operation code (OPCODE), Rd, Rs and A/K, so that these values can be sent to the corresponding component, like the ALU, General Purpose register block etc. This is achieved simply by using a set of buffers inside a block to sort the signals to separate buses.

- **General Purpose Resisters (GPR)**

GPRs store and save operands and results during program execution. ALU and memories must be able to write/read those registers, so a set of eight 8-bit registers were used, along with multiplexers and a decoders to control which register is read or written. Two registers can be written at a time.

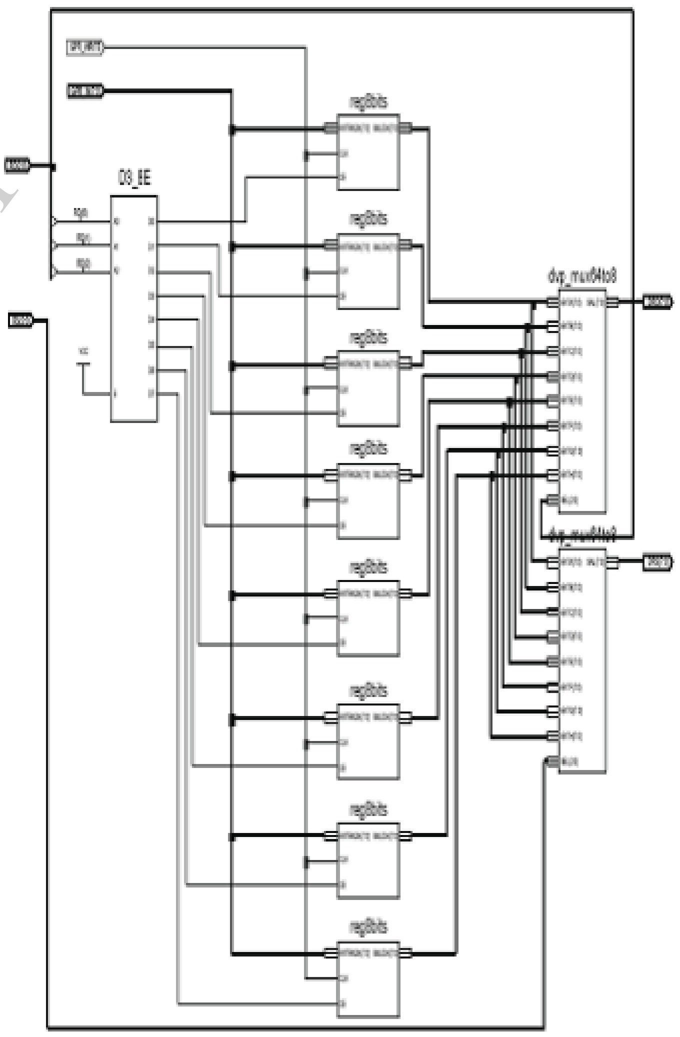


Fig 3. General Purpose Registers Block

- **Arithmetic Logic Unit (ALU)**

The ALU has 8 operations; each one of them was created and converted into a symbol, then a multiplexer was placed to a obtain a 3-bit selector. Also it has 3 Flags: carry (C), half carry (H) and zero (Z), which indicate whether there is a carry, a half carry or a zero after any ALU operation.

- **RAM Data Memory**

The RAM memory is a data storage block, there the stack is handled and other data are kept as variables. It is built with 32 memory blocks of 32 bits each one. The address input is divided in 2 parts. The first part has 3 bits select the memory block to read or write using a decoder. The second has 5 bits that select the memory location between 0 and 31.

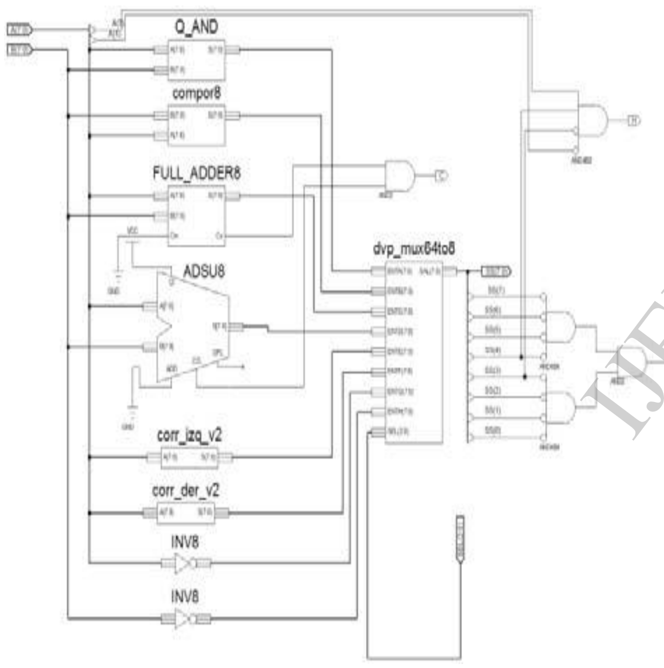


Fig 4. ALU block

- **Control Unit**

The control unit operates as a state machine. In general, it follows the state diagram shown in Fig 5. States 0, 1 and 3 remain equal , whereas the second state can change depending on the instruction read by the decoder. In this way: state 0 represents the fetch/write back stage and finally during state 3 the program counter is incremented.

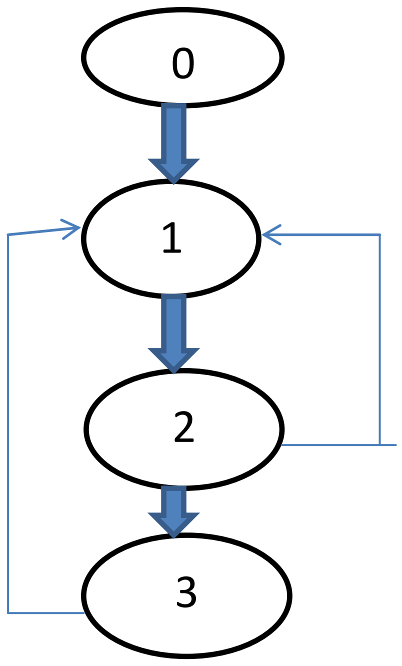


Fig. 5. Control Unit States Diagram

V. RESULTS

(a) Development Tools

For designing the CPU, Modelsim tool is used. Simulation is performed on Modelsim simulator, designed to work with files generated by Modelsim.

To simplify the conversion of CPU test programs, a specific translator is designed in Verilog HDL language. It is reads **.V** files which contains the program in assembly language, and translates it to machine code, giving the proper format to load the instructions to the program memory.

(b) Test Application

This program is used to evaluate the performance of the CPU by using all types of instructions to execute the subroutines, access RAM and manipulate the stack.

Table II : TRUTH TABLE FOR CONTROL UNIT STATE MACHINE

STATE	PC_LOAD	PC_CLOCK	PC SOURCE	IR_LOAD	IDR_LOAD	GPR_WRITE	GPR_SOURCE1	GPR_SOURCE0	ALU_SOURCE	RAM_WRITE	RAM SOURCE	INSTRUCTIONS
0	0	0	0	0	0	0	0	0	0	0	0	Reset
1	0	0	0	1	1	0	0	0	0	0	0	Fetch
2	1	0	0	0	0	0	0	0	0	0	0	JMP, BRC, BRZ, BRH
3	0	1	0	0	0	0	0	0	0	0	0	Inc. PC
2	0	0	0	0	0	1	0	0	0	0	0	LDI
2	0	0	0	0	0	1	0	1	0	0	0	LDD
2	0	0	0	0	0	0	0	0	0	1	0	STD
2	0	0	0	0	0	1	1	0	1	0	0	ADDI, SUBI
2	0	0	0	0	0	1	1	0	0	0	0	ADD, SUB, AND, OR, NOT, SHL, SHR
2	0	0	0	0	0	1	0	1	0	0	1	LDX
2	0	0	0	0	0	0	0	0	0	1	1	STX
2	0	0	0	0	0	1	1	1	0	0	0	LDP
2	1	0	1	0	0	0	0	0	0	0	0	JMR

Fig. 6 shows the jump to the subroutine which is executed when the PC changes from one value to another value.

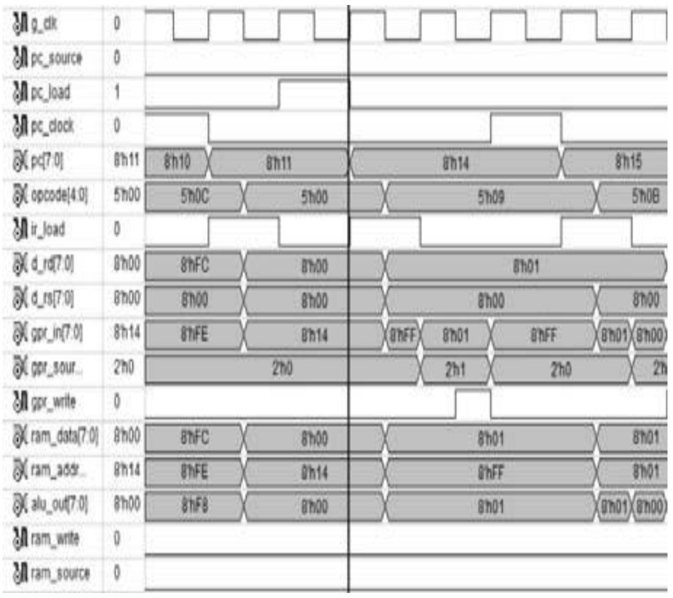


Fig 6. Simulation of an applicationI (Subroutine jump)

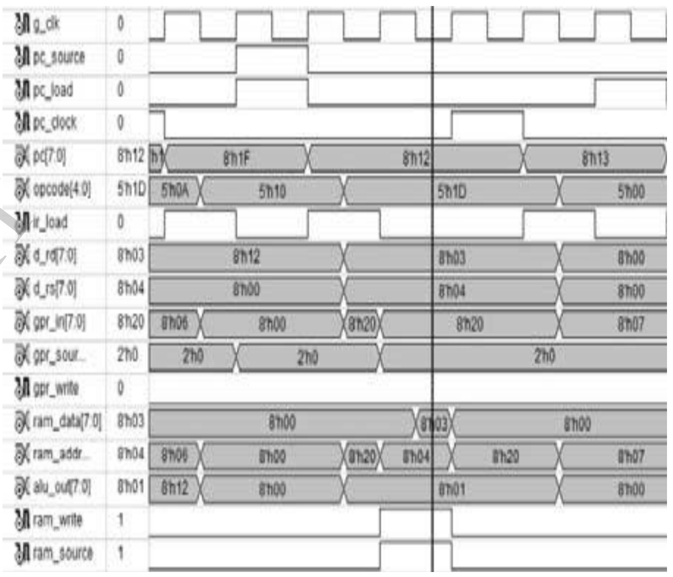


Fig 7. Simulation of a test application (Storage in RAM)

IV. CONCLUSION

From this task we can observe that flexibility of a design, which supports a great modification and improvement of the design. The simplicity of the design-based on functional blocks make understanding of each part of a modern computer works.

The circuit which is obtained can be used as soft core processor for FPGA designs that would like to add a CPU to handle other peripheral devices.

References:

[1] W. Stallings, “Organización y Arhitecture Computadoras,” 7ªEditión, Pearson, 2006.
[2] M. Mano, “Arquitectura de computadoras,” 3ª Editión, Pearson, 1994.

[3] A.K.Ray, “Advanced Microprocessor And Peripherals”, 2nd Edition, Tata McGraw-Hill,2008
[4] D.V.Hall,“Microprocessor And Interfacing”, 2nd Edition, Tata McGraw-Hill,2006
[5] D. Mandalidis, P. Kenterlis, J. Ellinas, “A computer architecture educational system based on a 32-bit RISC processor,” International Review on computers and Software, pp. 114-119, 2008.
[6] Tocci, Widmer, Moss, “Sistemas Digitales, Principios y aplicaciones,” 10ª Editión,Pearson, 2007.
[7] Antonio H. Zavala, Jorge Avante R.,”RISC-Based Architecture for computer Hardware Introduction”
[8] M. Jaumain, M. Osee, A. Richard, A. Vander Biest, P. Mathys, “Educational simulation of the RiSC processor,” ICEE International Conference on Engineering Education, 2007.
[9] P. Verplaetse, J. Campenhout, “ESCAPE: Environment for the Simulation of Computer Architecture for the Purpose of Education,” IEEE TCCA Newsletter, February, pp. 57-59, 1999.
[10] G. Wainer, S. Daicz, L. De Simoni, D. Wassermann, Using the Alfa-1 simulated processor for educational purposes,” Journal on Educational Resources in Computing, vol.1, Issue 4, pp.111–151, 2001
[11] J. Djordjevic, A. Milenkovic, N. Grbanovic, “An Integrated Environment for Teaching Computer Architecture,” IEEE Micro vol.20, Issue 3, pp. 66–74, 2000.
[12] M. Becvar, A. Pluhacek, J. Danecek, “DOP: a CPU core for teaching basics of computer architecture,” Workshop on Computer architecture education, Article No. 4, 2003.
[13] H. Elaarag, “A complete design of a RISC processor for pedagogical purposes,” Journal of Computing Sciences in Colleges, vol. 25, Issue 2, pp. 205-213, 2009.