International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 7, July - 2014

Multi Cycle Implementation Scheme for 8 bit Microprocessor by VHDL

Sharmin Abdullah, Nusrat Sharmin, Nafisha Alam
Department of Electrical & Electronic Engineering
Ahsanullah University of Science & Technology
Dhaka, Bangladesh

***Abstract*— Computers and computer systems play a significant role in the modern world. The central component of these computers and computer systems is the microprocessor. The applications of digital designs are present in designing microprocessors. This paper presents the implementation of the design of a Multi Cycle Central Processing Unit (CPU) in Very High Speed Integrated Circuits (VHSIC) Hardware Description Language or commonly known as VHDL. The implementation was carried out to understand the development of processor hardware as the design and customization of the processors which has become a mainstream task in the development of complex Systems-on-Chip.**

Keywords—ALU, Instruction fetch, Instruction decode, Execution, Control, ALU controller, Data memory, VHDL implementation.

I. INTRODUCTION

A microprocessor is an integrated circuit on a tiny silicon chip that contains thousands or millions of tiny on/off switches, known as transistors. It is designed to perform arithmetic and logic operations that make use of data on the chip and data in RAM (Random Access Memory) [1]. It computes instructions depending on how fast the processor is designed and if the processor's clock crystal (think of the clock crystal as a metronome). The faster (in MHz or GHz) the processor and its clock crystal are the better. Processors currently come in 8, 16, 32 and 64 bit versions. This paper presents the design of an eight bit multi cycle microprocessor by VHDL which can describe the behavior and structure of electronic systems, but is particularly suited as a language to describe the structure and behavior of digital electronic hardware designs, such as ASICs and FPGAs as well as conventional digital circuits. Processor´s speed can be improved by using multi cycle implementations.

In the first section starting from a simple implementation scheme of a MIPS (Million Instruction Per Second) subset the basic hardware of the microcontroller´s data path and its control is developed step by step and implemented in VHDL. Test benches will verify the correct implementation of the arithmetic logical instructions (*add*, *sub*, *and*, *or* and *slt*), the memory-reference instructions (*load word* and *store word*) and the branch instructions (*beq* and *jump*). Processor' speed can be improved by using multi cycle implementation. Then, instructions are allowed to take different numbers of clock cycles and functional units can be shared within the execution of single instructions. Multi cycle implementation also called multi clock cycle implementation [2].

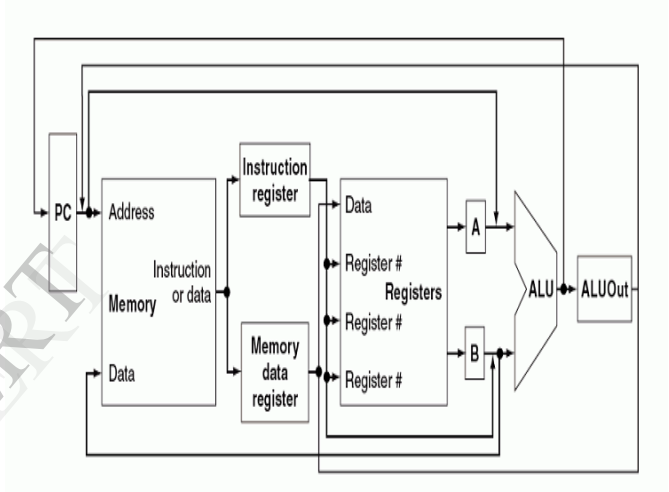

Fig. 1. High level view of multi cycle Data path [3].

In order to enhance the performance and to get very fast processors another implementation technique called pipelining is introduced. Multiple instructions are overlapped in execution so that some stages are working in parallel.

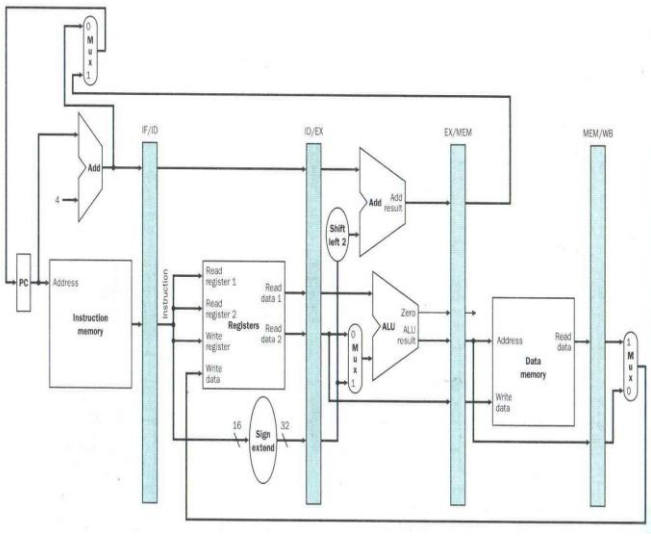

Fig. 2. Pipelined Version of the Data path [4].

IJERT

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

II. DESIGN TASK

A. Data path

A data path is a collection of functional units, such as arithmetic logic units or multipliers that perform data processing operations. It is a central part of many central processing units (CPUs) along with the control unit, which largely regulates interaction between the data path and the data itself, usually stored in registers or main memory.

B. Register transfer level models implemented in pure vhdl

The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). A VHDL project is portable. Being created once, a calculation block can be used in many other projects.

C. Synthesis tool: ALTERA QUARTUSII

The *Quartus II* development software provides a comprehensive environment for system-on-a-programmable-chip (SOPC) design. It provides a complete design environment for FPGA designs. Design entry uses schematics, block and Verylog HDL. Design analysis and synthesis, fitting, assembling, timing analysis, simulation.

D. Block diagram of first hierarchy levels

A Quartus II *Block Diagram* File can be used as part of a *hierarchical* design. That is, it can be represented as a component in a higher-*level* design.

Fig. 3. Block diagram of ALU.

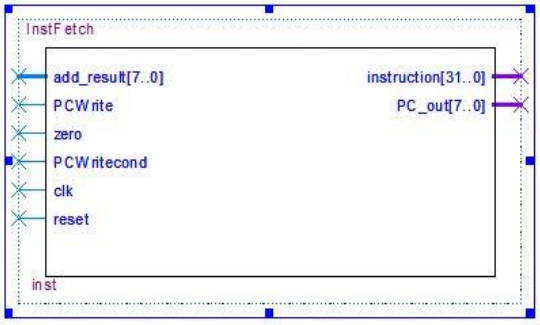

Fig. 4. Block diagram of INSTRUCTION FETCH.

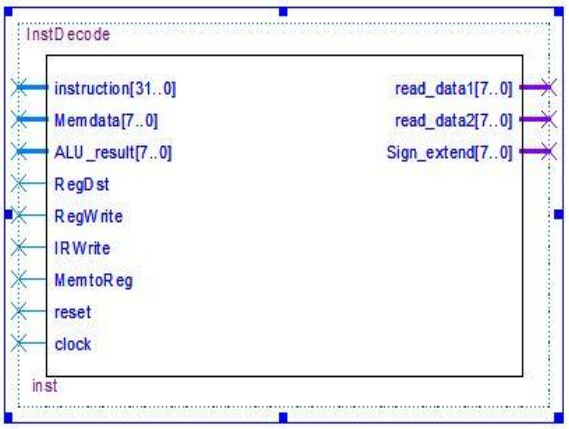

Fig. 5. Block Diagram of INSTRUCTION DECODE

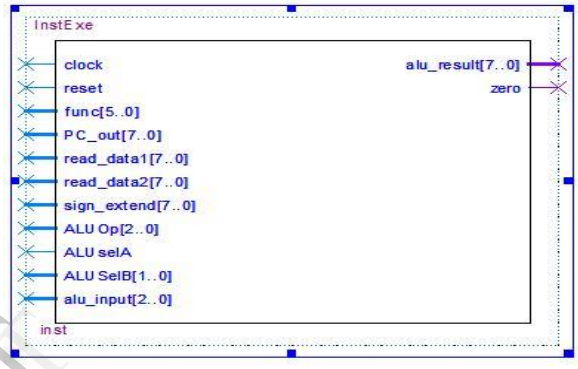

Fig. 6. Block diagram of EXECUTION UNIT

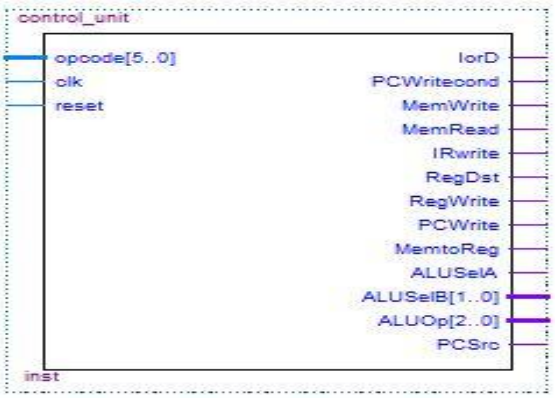

Fig. 7. Block diagram of CONTROL UNIT

Fig. 8. Block diagram of DATA MEMORY.

IJERT

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

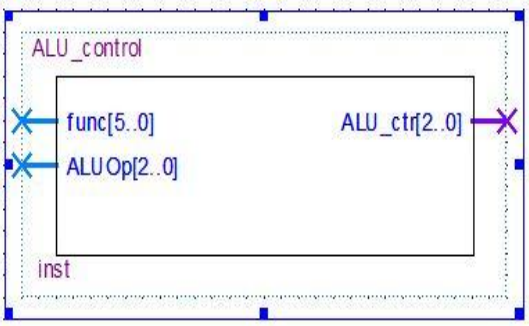

Fig. 9. Block diagram of ALU CONTROLLER

III. MULTI CYCLE IMPLEMENTATION

The Multi cycle implementation uses several registers to temporarily hold the output of the previous clock cycle. These include an Instruction register, Memory data register, ALU Output register, etc. The Multi cycle machine breaks simple instructions down into a series of steps. These steps are:

A. Instruction fetch step

During the instruction fetch step the Multi cycle processor fetches instructions from the memory and computes the address of the next instruction by incrementing the program counter (PC).

B. Instruction decode and Register fetch step

During the second step, the Instruction decode and register fetch step, we decode the instruction to figure out what type it is: memory access, R-type, I-type, branch.

C. Execution, memory address computation, or branch completion step

During the third step, the Execution, memory address computation, or branch completion step functions in different ways depending on what type of instruction the processor is executing. This third step is the last step for branch and jump instructions. It is the step where the next PC address is computed and stored.

D. Memory access or R-type instruction completion step

The fourth step only takes place in load word, store word, R-type, and I-type instructions. This step is when the load and store word instructions access the memory and use an arithmetic-logical instruction to write its result.

Values are either loaded from memory and stored into the memory data register, or loaded from a register and stored back into the memory. This fourth step is the last step for R-type and I-type instructions. For R and I type instructions this is the step where the result from the ALU computation is stored back into the destination register [5].

E. Memory read completion step

Only load instructions need the fifth step to finish up. This is the memory read completion step. In a load instruction the value of the memory data register is stored back into the register file.

TABLE I. How ALU Control bits are set and different function codes for the R type Instruction.

Instruction Opcode	ALU Op	Instruction operation	Function Field	Desired ALU action	ALU control Input
LW	000	Load word	XXXXXX	add	010
SW	000	Store word	XXXXXX	add	010
Branch equal	001	Branch equal	XXXXXX	subtract	110
R-type	010	Add	1000000	add	010
R-type	010	subtract	100010	subtract	110
R-type	010	AND	100100	and	000
R-type	010	OR	100101	or	001

These different steps are all controlled by the controller of the multi cycle CPU. The controller is a finite state machine that works with the Opcode to walk the rest of the components through all the different steps, or states. The controller controls when each register is allowed to write and controls which operation the ALU is performing.

The multi cycle data path requires some additional requirements to support branches and jumps. With the jump and branch instructions there are three possible sources for the value to be written into the PC.The output of ALU which has the value PC+4 during instruction fetch would be stored directly into the PC. The address of the branch target after computation is stored in the ALU Out register. The lower 26 bits of the IR shifted left by two and concatenated with the upper four bits of the incremented PC which is the source when the instruction is jump.

Two separate control signals are used in this implementation: PC Write which causes an unconditional write of the PC and PC Write Cond which causes a write of the PC if the branch condition is true.

The control lines are attached to the control unit and the control and data path elements needed to effect changes to the PC. To select the source of new PC value, multiplexer is used. Gates are used to combine the PC Write signals and control signals from PC Source, PC Write and PC Write Cond. The PC Write Cond signal is used to decide whether a conditional branch should be taken. This supports for jumps also. The complete multi cycle data path and control unit

IJERT

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

including the additional control signals and multiplexor for implementing the PC updating are shown.

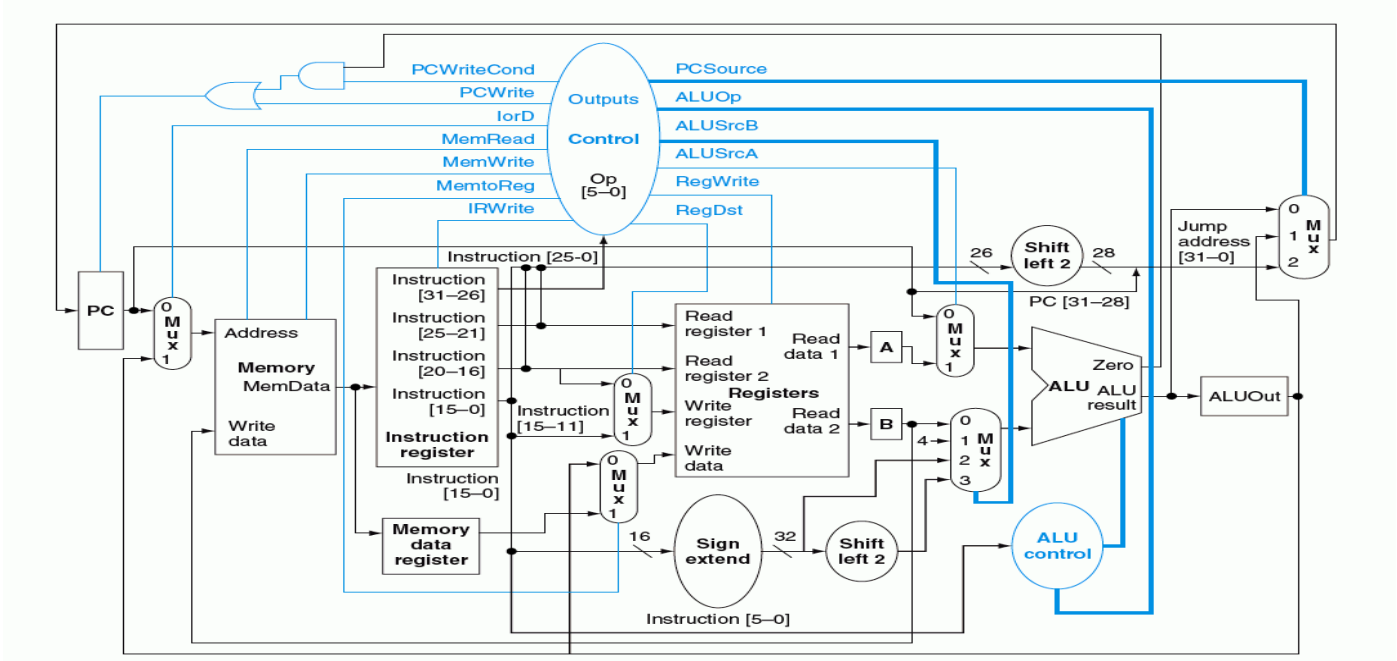

Fig. 10. The complete data path for the multi cycle implementation together with the necessary control lines [6].

The first method we use to specify the multi cycle control is a finite state machine. A finite state machine is a sequential logic function consisting of a set of inputs and outputs, a next state function that maps the current state and the inputs to a new state and an output function that maps the current state and possibly the inputs to a set of asserted outputs.

The figure which is shown below represents a complete specification of the control for MIPS subset with two types of exception.

The labels on the arcs are conditions that are tested to determine which state is the next state. When the next state is unconditional, no label is given Here states 10 and 11 are the states that generate the appropriate control for exceptions. The branch out of state is labeled 1 indicates the next state when the input does not match the output of any of lw, sw, zero(R-type), j or beq. The branch out of state labeled 7 over flow indicates the action to be taken when the ALU signals are overflow.

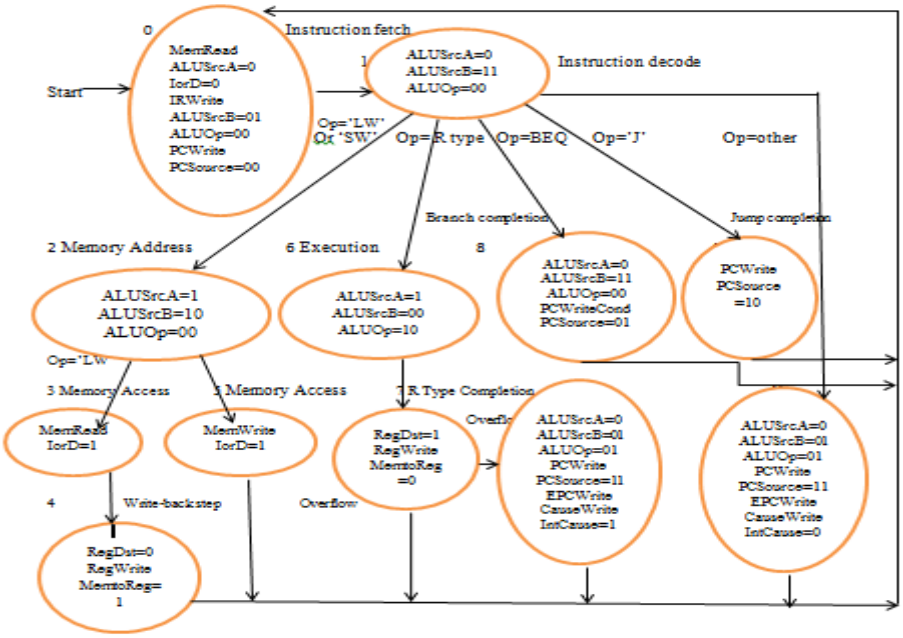

Fig. 11. The finite state machine with the addition to handle exception detection [7].

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

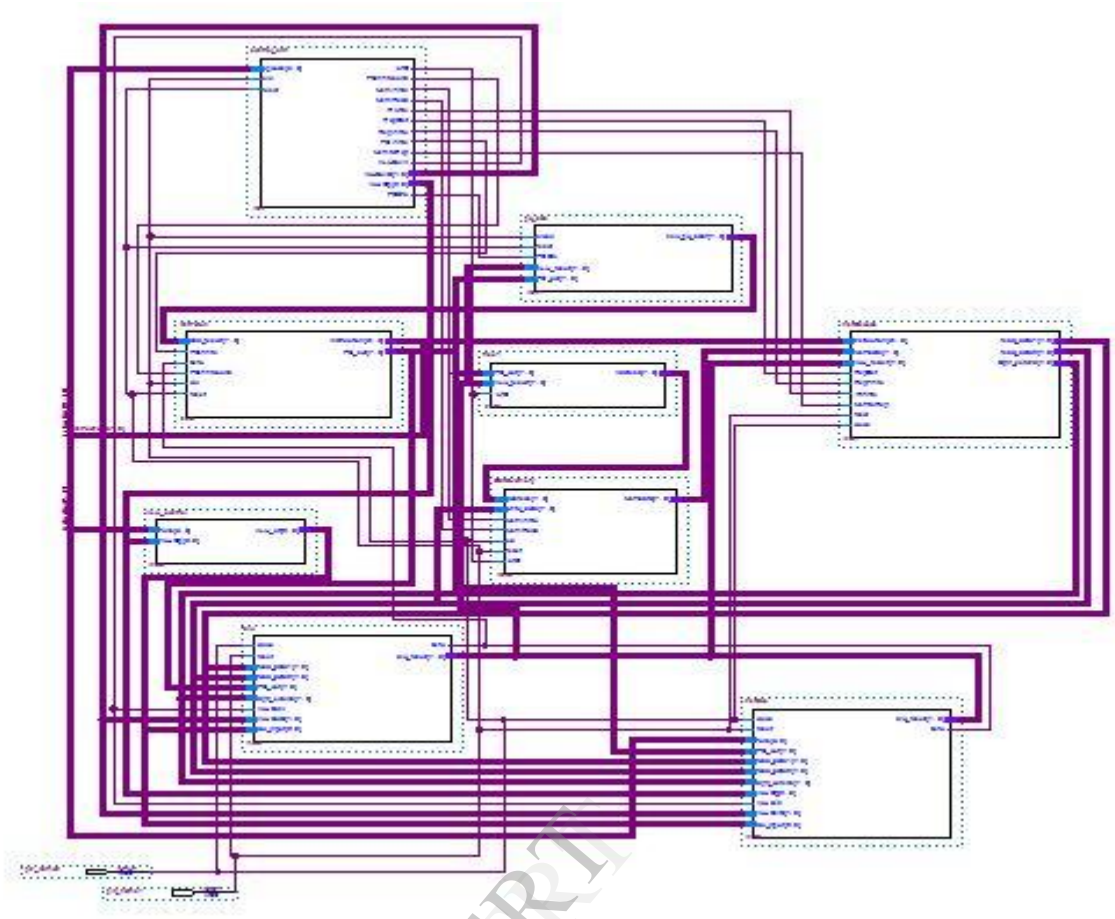

Fig. 12. Complete RISC Microprocessor using Blocks.

IJERT

IV. CONCLUSION

The design was tested and simulated, no error was encountered. The project is ready to implement further. The idea of this project was to create a microprocessor as a building block in VHDL that later easily can be included in a larger design. It will be useful in systems where a problem is easy to solve in software but hard to solve with control logic. A state machine dedicated to the function can of course replace the microprocessor and associated software. However, at a high level of complexity it is easier to implement the function in software. After going through all the hard work and facing problems, this project managed to complete its objectives that are to study different Multiplier and learn the Power and Time trade off among them so that we can design.

ACKNOWLEDGMENT

The authors thank the respective supervisor Prof. Dr. Satyen Biswas of Ahsanullah University of Science & Technology for the continuous support of the study and research and especially they dedicate their acknowledgment of gratitude toward their beloved and respective parents.

REFERENCES

[1] http://www.belarus.net/Intel/MUSEUM/micropr.htm
[2] https://www.cs.duke.edu/courses/spring01/cps104/lectures/2up-lecture15.pdf
[3] https://www.cs.drexel.edu/~jjohnson/2012-13/fall/cs281/lectures/pdf/cs281_lec16.pdf
[4] Computer Organization and Design - The Hardware/Software Interface, Third Edition, David A. Patterson, John L. Hennessy
[5] https://www.cs.duke.edu/courses/spring01/cps104/lectures/2up-lecture15.pdf
[6] Computer Organization and Design - The Hardware/Software Interface, Third Edition, David A. Patterson, John L. Hennessy
[7] http://faculty.washington.edu/lcrum/Archives/TCSS372AF08/MIPS_Arch2.ppt
[8] Altera Corporation www.altera.com
[9] The VHDL Cookbook-Peter J. Ashenden
[10] http://www.eecg.toronto.edu/~moshovos/ECE243-07/l20-multicycle.html
[11] http://www.pitt.edu/~kmram/CoE0147/lectures/multicycle1.pdf

(This work is licensed under a Creative Commons Attribution 4.0 International License.)