Control Unit for MIPS Instruction Using Multicycle Implementation

Sai Yee Zame, Thwe Mu Han
Computer University, (Mandalay)
yeezame@gmail.com

Abstract

Architectural advances of modern systems have been added with control complexity, requiring significant effort in both design and verification. The control is the main part of the modern system. In modern system, it has three designs: single-cycle, multicycle and pipelining. In single-cycle, datapath and functional unit can't be used more than one per instruction because it takes one clock cycle for operation. In multicycle, it executes instruction into multiple steps and each step is executed in one clock cycle. It allows a functional unit to be used more than once per instruction. In pipelining, it is implementation technique in which multiple instructions are overlapped in execution. To have more performance and reduce amount of hardware components, this system uses multicycle. For multicycle operation, control unit generates control signals to datapath elements. So, datapath and control signals for each step of operation are implemented. This system is implemented by verilog language.

Keywords: multicycle datapath, main control, ALU control.

1. Introduction

Computer performance has increased rapidly for a number of years. Examining performance is the key to the effectiveness of an entire system of hardware and software. System complexity generally affects costs and reliability; more recently, it particularly affects power consumption and time. Time is the measure of the performance. The performance of a computer can be determined by three key factors: instruction count, clock cycle and clock cycle per instruction (CPI). Therefore, architecture innovations that enable efficient system designs to achieve higher performance at lower complexity have always been a primary target for computer architecture.

In the digital system, it has two parts: a datapath unit and a control unit. The datapath is a network of functional and storage units capable of performing certain operations on data words. The purpose of the control unit is to issue control signals to the datapath. These control signals enter the datapath at control points, where they select the functions to be performed at specific times and route the data through the appropriate parts of the datapath unit. In other words, the control unit logically reconfigures the datapath to implement some specified instruction or program. In this paper, control signals for each step of multicycle operation are implemented.

This paper is organized as follow. Section 2 discusses related work. Section 3 presents MIPS instruction set architecture (ISA). Section 4 describes control unit. Section 5 describes system overview design. Section 6 shows implementation of the system. Section 7 describes the conclusion of this paper.

2. Related Work

M. Linder, M. Schmid presented "Processor Implementation in VHDL". They designed and implemented processor structure using VHDL to study basic concepts and component, and also to study the technique for how to design and implement the processor datapath [2].

Victor P. Rubio, B.S. described "A FPGA Implementation of a MIPS RISC Processor for Computer Architecture Education". They emphasized on subsequent courses, which take the minimal implementation and guide the students [8].

Thant Zin Tun proposed **"**Implementation of Pipelined Datapath Design for Handling Branch Hazard". She implemented how to handling branch hazard for pipeline and the common pipeline datapath and control signals [6].

Thapyay proposed "A Dual Pipelines Datapath Design For Integer-Floating Point Instruction Pairs". She designed to produce the datapath for integer-floating point instruction pairs. She also implemented control signals in the dual pipelines datapath [7].

N. L. V. Calazans, F. G. Moraes and C. A. M. Marcon described "Teaching Computer Organization and Architecture with Hands-on Experience". They emphasized on subsequent

courses, which take the minimal implementation and guide the students through the necessary steps to add performance, such as memory management and basic I/O subsystems [4].

3. MIPS Instruction Set Architecture

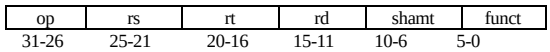
MIPS Instruction Set Architecture is a modern computer language. It interfaces between hardware and low level software. It has standardizes instructions and machine language bit pattern. The advantage of instruction is different implementation of the same architecture [9].

MIPS is an example of reduced instruction set computer (RISC) architecture. RISC architectures have a fewer number of simple instructions than complex instruction set computer (CISC) architecture. The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

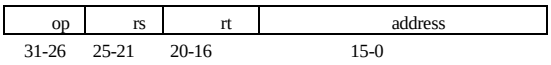
In RISC instruction set nature, there are three types of instruction set. They are:

- The memory reference instructions: load word (lw), store word (sw), branch on equal (beq).
- The arithmetic-logical instructions: addition (add), subtraction (sub), and-operation (and), or-operation (or), nor-operation (nor) and set on less than (slt).
- The jump instruction: jump (j).

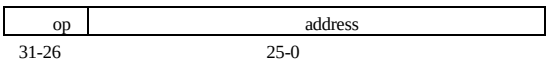
The three types of instruction set use 32 bits format. These format names are R-type (arithmetic or logic type), I-type (Immediate type) and J-type (Jump type). This format's operand is 32 bits register. The three formats are shown in figure 1.



(a) R-type instruction format



(b) I-type instruction format



(c) J-type instruction format

Figure 1. The three instruction classes

Each of these segments of an instruction format is called a field. There are several major observations about this instruction format. These instruction fields are following:

- The op field, also called the opcode, is always contained in bit 31:26 (6 bits).
- The two registers to be read are always specified by the rs (first source register) and rt (second source register) fields, at positions 25:21 and 20:16. This is true for R-type instructions, branch instructions and for store.
- The base register for load and store instructions is always in bit positions 25:21 (rs).
- The 16-bit offset for load and store instructions is always in bit positions 15:0.
- The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), while for an R-type instruction it is in bit positions 15:11 (rd). Thus, a multiplexer is needed to select which field of the instruction is used and to indicate the register number to be written.
- The 26-bit offset for jump instruction is always in bit position 25-0 [1].

Table 1. Types of instruction set based on RISC

Category	Instruction	Example	Meaning
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)
	set on less than	slt $1, $2, $3	if($2<$3) $1=1; else $ 1 = 0;
Data Transfer	load word	lw $s1,100($s2)	s1 = Memory [$s2 + 100]
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1
Conditional branch	Branch on equal	beq $s1,$s2,L	if($s1 == $s2) go to L
Unconditional branch	jump	j L;	go to L;

4. Control Unit

In this system, the execution of an instruction is divided into five steps. These different steps are all controlled and orchestrated by the brain of the system. This brain is the controller. The controller has two parts. They are main control and ALU control. The main control generates 13 signals to the desired components of the system. The ALU control generates 1 (4 bits) signal to only ALU.

4.1. Main Control

The instruction's opcode field is the input of the main control unit. When it received input, it will generate 10 (1bit) signals and 3 (2 bits) signals for each step. 10 (1bit) signals are RegDst, RegWrite, ALUSrcA, MemRead, MemWrite, MemtoReg, IorD, IRWrite, PCWrite and PCWriteCond. 3(2bits) signals are ALUOp, ALUSrcB and PCSource.

There are two different techniques to specify the main control in multicycle. The first technique is based on finite state machines that are usually represented graphically. The second technique is called microprogramming uses a programming representation for control.

4.1.1. Microprogrammed Control. An instruction is implemented by a sequence of one or more sets of concurrent micro operations. Each micro operation is associated with a group of control lines that must be activated in a prescribed sequence to trigger the microoperations. Microprogramming is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM or RAM called a control memory CM.

The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM in much the same way an instruction is fetched from main memory. Each microinstruction also explicitly or implicitly specifies the next microinstruction to be used, thereby providing the necessary information for microoperation sequencing. A set of related microinstructions forms a microprogram [1].

4.1.2. Finite State Machine. In the single step implementation the control was defined by simple truth tables that set the control signals depending on the instruction. This does not work for a multicycle datapath. The control is more complex, because it must specify both the signals to be set in any step and the next step in the sequence [1]. Therefore this system uses finite state machine. Figure 2 shows finite state machine for the system.

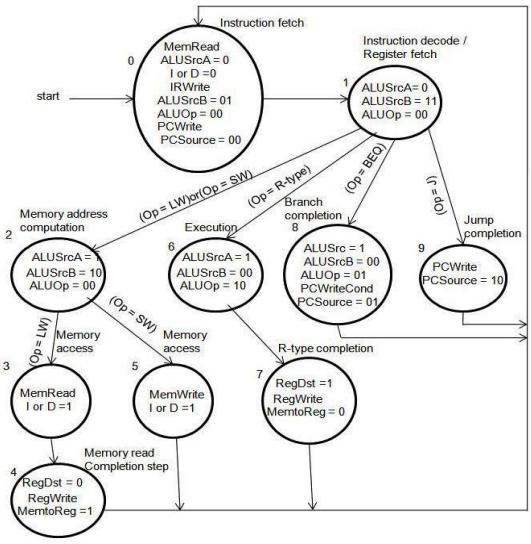


Figure 2. Finite state machine for the system

4.2. ALU Control

The ALU Control generates control signals for only ALU. The input of ALU Control is ALUOp and function field [6-0] of instruction. ALUOp signal is 2 bits signals that come from main control. For lw, sw, and beq instructions, it does not need function field. It needs ALUOp only. If ALUOp signal is 10, it needs function field to determine control signal for R-type.

Table 2. ALU control signal

Instruction Opcode	ALUOp	Instruction Code	Function Code	Desired ALU action	ALU control input
Lw	00	Load word	xxxxxx	Add	0010
Sw	00	Store word	xxxxxx	Add	0010
Branch	01	Beq	xxxxxx	Subtract	0110
R-type	10	Add	100000	Add	0010
R-type	10	Subtract	100010	Subtract	0110
R-type	10	AND	100100	And	0000
R-type	10	OR	100101	Or	0001
R-type	10	NOR	100111	Nor	1100
R-type	10	set-on-less-than	101010	set-on-less-than	0111

5. System Overview Design

In the system, it can capable of handling ten different instructions by using multicycle. These instructions are in a variety of categorized: R-type (add, sub, and, or, nor, slt), I-type (sw, lw, beq) and J-type (jump). Each of these categories has different instruction format. This system breaks this instruction into many smaller steps. Each step takes one clock cycle for operation. This system allows each functional block to be used more than once per instruction because instruction's step uses different clock cycle. It can also share modules, allowing the use of fewer hardware components. It uses only one ALU and memory for data and instruction. It uses several registers to temporarily hold the output of the previous clock cycle. These registers are Instruction register, Memory data register, ALUOut register, etc.

The multicycle machine breaks simple instructions down into a series of steps. These steps typically are:

1. Instruction fetch step
2. Instruction decode and Register fetch step
3. Execution, memory address computation, or branch completion step
4. Memory access or R-type instruction completion step
5. Memory read completion step

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

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

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. For a memory access instruction the ALU computes the memory address. An R-type instruction uses this third step to perform the actual arithmetic. 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.

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 and 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.

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. The multicycle datapath and control design of the system is shown in figure 3.

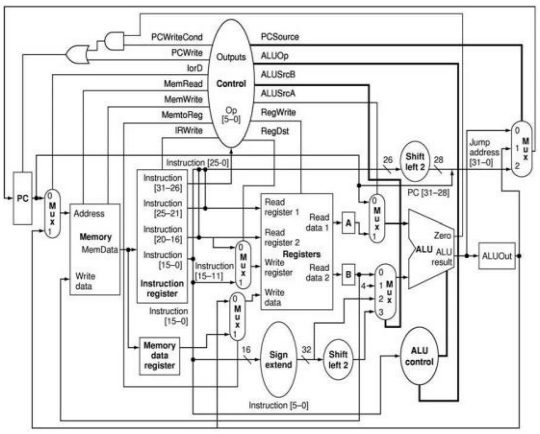


Figure 3. Multicycle datapath and control design of the system

6. Implementation of the System

In the system, it can execute core MIPS instructions. They are the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions. Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction. Each MIPS instruction needs from three to five steps. For every instruction, the first two steps are identical. The operation steps are following:

1. Instruction fetch step:

In this step, it sends the PC to memory as the address, perform a read, and write the instruction into the Instruction register (IR), where it is stored. Also increment the PC by 4.

2. Instruction decode and register read step

In this step, the rs and rt instruction fields indicates the two register for read. The values read from the register file may be needed in later stages. These values are stored into the temporary register A and B. The branch target address is computed with ALU that compute for branch instruction.

3. Execution, Memory Address computation or Branch completion step

In this step, it performs one of four functions, depending on the instruction class. For Memory reference, the ALU is adding the operands to form the memory address. For arithmetic-logical instruction, the ALU is performing the operation specified by the function code on the two values read from the register file in the previous cycle. For branch, the ALU is used to do the equal comparison between the two registers read in the previous step. The zero signal out of the ALU is used to determine whether or not to branch. For jump, ALU does not use. The PC is replaced by the jump address. The jump and beq instructions are completed at the step. Their control signals are shown in Table 3.

Table 3. Control signal for instruction (jump, beq)

bit	Control	first cycle	second cycle	third cycle	
				j	I(beq)
1 bit	PCWriteCond	0	0	0	1
	PCWrite	1	0	1	0
	IorD	0	-	-	-
	MemRead	1	0	0	0
	MemWrite	0	0	0	0
	MemtoReg	-	-	-	-
	IRWrite	1	0	0	0
	ALUSrcA	0	0	-	1
	RegWrite	0	0	0	0
	RegDst	-	-	-	-
2 bit	PCSource	00	-	10	01
	ALUOp	00	00	-	01
	ALUSrcB	01	11	-	00

4. Memory Access or R-type Completion Step

In the step, a load or store instruction accesses memory and an arithmetic logical instruction writes its result. If the instruction is a load, a data word is

retrieved from memory and is written into the MDR. If the instruction is a store, then the data is written into memory. In either case, the address used is the one computed during the previous step and stored in ALUOut. For a store, the source operand is saved in B. For arithmetic-logic instruction, it places the contents of ALUOut into the rd register.

5. Memory Read Completion Step

In this step, load instruction is completed by writing the load data into the register file.

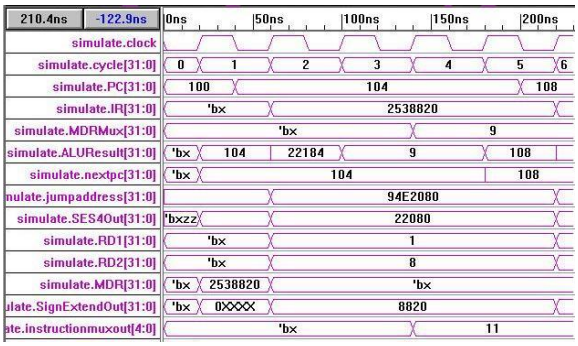
R-type and sw instructions are completed at step 4. Only lw instruction is completed at step 5. Their control signals are shown in Table 4.

Table 4. Control signals for R, I (lw, sw)

bit	Control signals	Third cycle		Fourth cycle			Fifth cycle
		R	I	R	I(sw)	I(lw)	I(lw)
1 bit	PCWriteCond	0	0	0	0	0	0
	PCWrite	0	0	0	0	0	0
	IorD	-	-	-	1	1	-
	MemRead	0	0	0	0	1	0
	MemWrite	0	0	0	1	0	0
	MemtoReg	-	-	0	-	-	1
	IRWrite	0	0	0	0	0	0
	ALUSrcA	1	1	-	-	-	-
	RegWrite	0	0	1	0	0	1
	RegDst	-	-	1	-	-	0
2 bit	PCSource	-	-	-	-	-	-
	ALUOp	10	00	-	-	-	-
	ALUSrcB	00	10	-	-	-	-

6.1. Output Timing Diagrams of R-type (add) Instruction, I-type (sw) and J-type (jump)

When the system starts, the control unit generates control signal. The functional units (PC, IR, ALU, etc.) operate and generate desired output when they receive the control signal. These outputs are implemented by timing diagram. 13 control signals are designed in this control unit. 10 different instructions are implemented by timing diagrams. These instructions are categorized into 3-types. Timing diagrams for R-type (add), I-type (sw) and J-type (jump) instructions are shown in figure 4, figure 5 and figure 6.



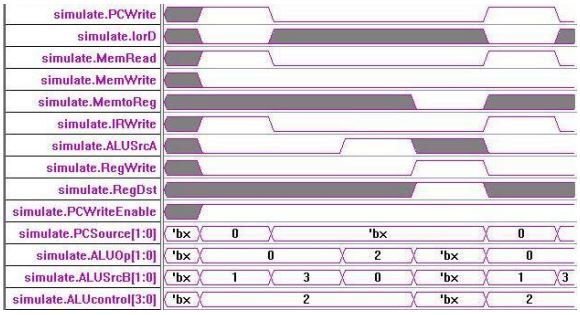


Figure 4. Timing diagram of multicycle for R-type (add) instruction

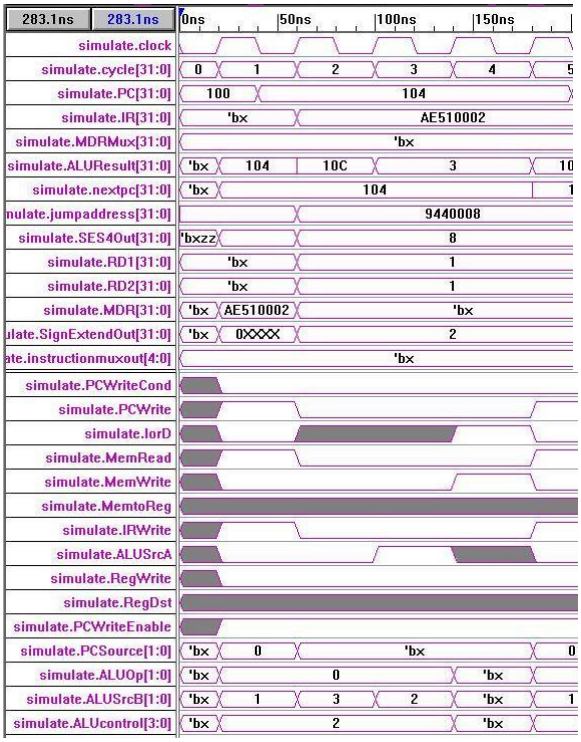


Figure 5. Timing diagram of multicycle for I-type (sw) instruction

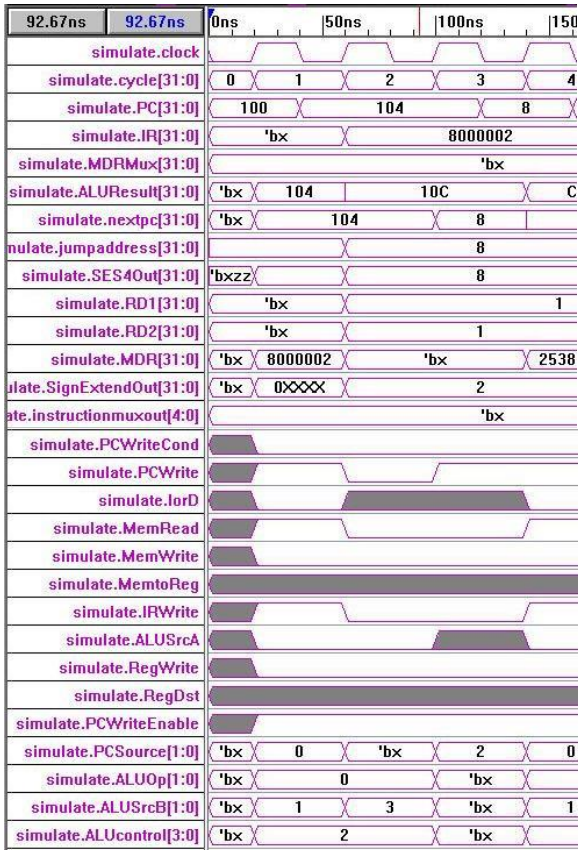


Figure 6. Timing diagram of multicycle for J-type (jump) instruction

7. Conclusion

The multicycle implementation of a CPU is a great improvement over a single-cycle implementation. In the multicycle design, the exact decomposition of the instruction execution into a number of cycles, which is based on the instruction set architecture, together with the datapath, defines the requirements on the control. Control is one of the most challenging aspects of computer design. A major reason is that designing the control requires an understanding of how all the components in the processor operate.

This system shows the implementation of a multicycle CPU capable of handling ten different instructions. The multicycle was implemented to break this instruction into many smaller steps. Each step needs control signal for operation. This paper implements the control unit generates the control signal for many steps of instructions.

These ten instructions are add, sub, and, or, nor, slt, lw, sw, beq and jump. The beq and jump need 3 clock cycle, add, sub, and, or, nor, slt and sw need 4 clock cycle and lw needs 5 clock cycle for operation. For all instruction, the control signal for first and second are equal. Other MIPS instructions (eg. addi, andi, bne, etc.) can't operate on this system. If these instructions are operation on this system, the required output does not have. To have the output of these instructions, this system needs to be reconstructed.

This system is designed with 32 bits processor for multicycle datapath. Thus, this design cannot apply for more than 32 bits applications. This system is not implemented to all MIPS instructions and floating point instruction.

8. References

[1] A. Patterson, David and L. Hennessy, John, *Computer Organization and Design, Hardware/Software Interface*, Third Edition, Morgan Kaufmann Publishers Inc San Francisco, California, 2005.

[2] M. Linder, M. Schmid *Processor Implementation in VHDL* University of Ulster at Jordanstown, 2007.

[3] Mark Holland *Harnessing FPGAs for Computer Architecture Education*, University of Washington, 2002.

[4] N. L. V. Calazans, F. G. Moraes and C. A. M. Marcon, "Teaching Computer Organization and Architecture With Hands-on Experience", 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, November 6-9, 2002.

[5] P. Hayes, John. *Computer Architecture and Organization* WCB/McGraw-Hill, Third Edition, 1998

[6] Thant Zin Tun *Implementation of Pipelined Datapath Design for Handling Branch Hazard* Computer University, Myeik, 2008

[7] Thapyay, "A Dual Pipelines Datapath Design for Integer-Floating Point Instruction Pairs", University of Computer Studies, Yangon, August, 2007.

[8] Victor P. Rubio, B.S. *A FPGA Implementation of a MIPS RISC Processor for Computer Architecture Education* New Mexico State University, 2004.

[9] *MIPS Instruction Set Architecture* Mount Holyoke College, 2007. Online document: http://cs.cmu.edu/afs/cs/academic/class/15740-97/public/doc/mips-isa.pdf