Class:	**CPE300L 1001**	Semester:	**Fall 2024**

Points		Document author:	**Alireza Bolourian**
		Author's email:	**bolouria@unlv.nevada.edu**
		Document topic:	**Postlab 07**

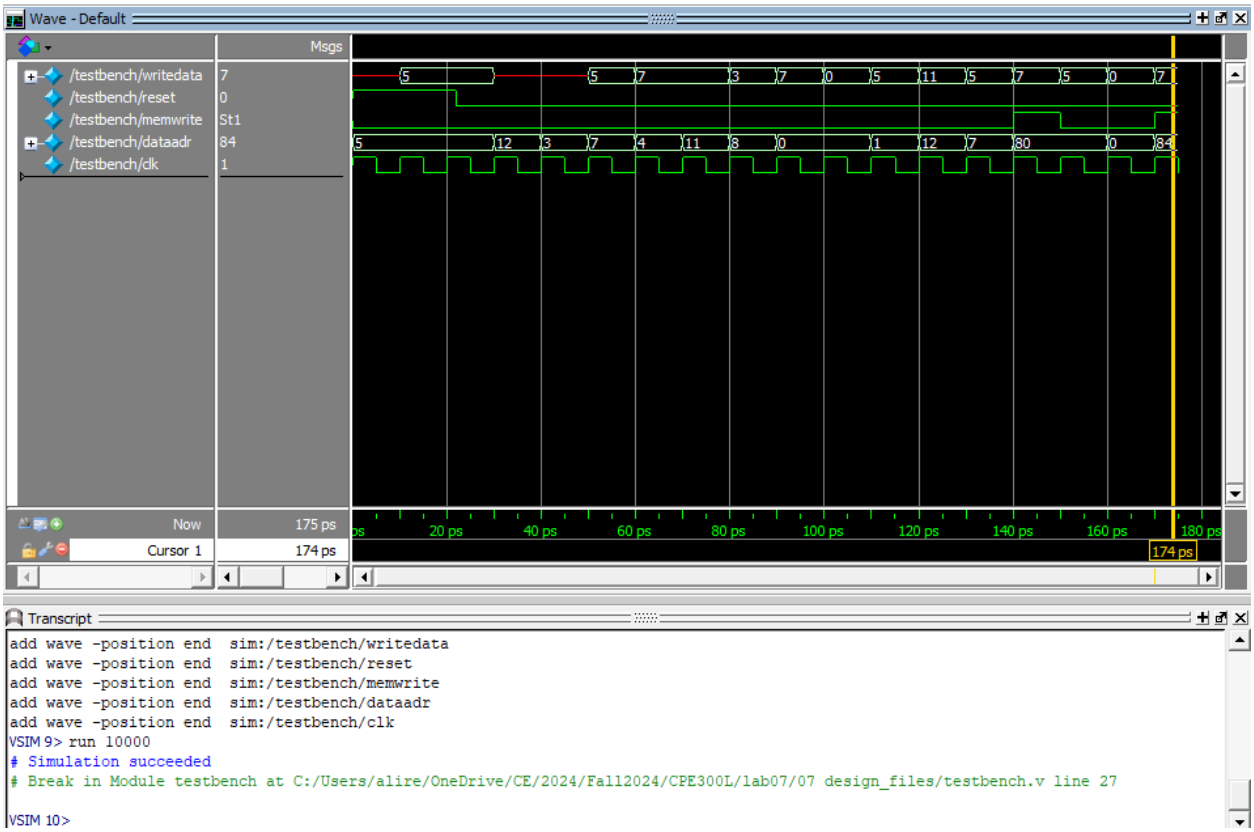
Instructor's comments:

1. Theory of Operation

A soft processor is a microprocessor that is implemented using programmable logic on existing hardware such as using a FPGA. It can be customized easily but has slower performance than hard processors because of the overhead of programmable logic. Hard processors are not reconfigurable once manufactured but are more efficient and cost effective for high volume production and applications.

In MIPS single cycle processor, each instruction is executed in a single cycle, so each instruction regardless of complexity takes the same time to complete. This means that simple instructions are slowed down to accommodate instructions that take longer time. In a MIPS multicycle processor, instructions are broken down into multiple stages that each are completed in one cycle, leading to a faster clock. This allows simple instructions to be completed in less cycles than complex cycles, optimizing time and resources. However, this requires additional logic and flip-flops for operation.

2. Prelab



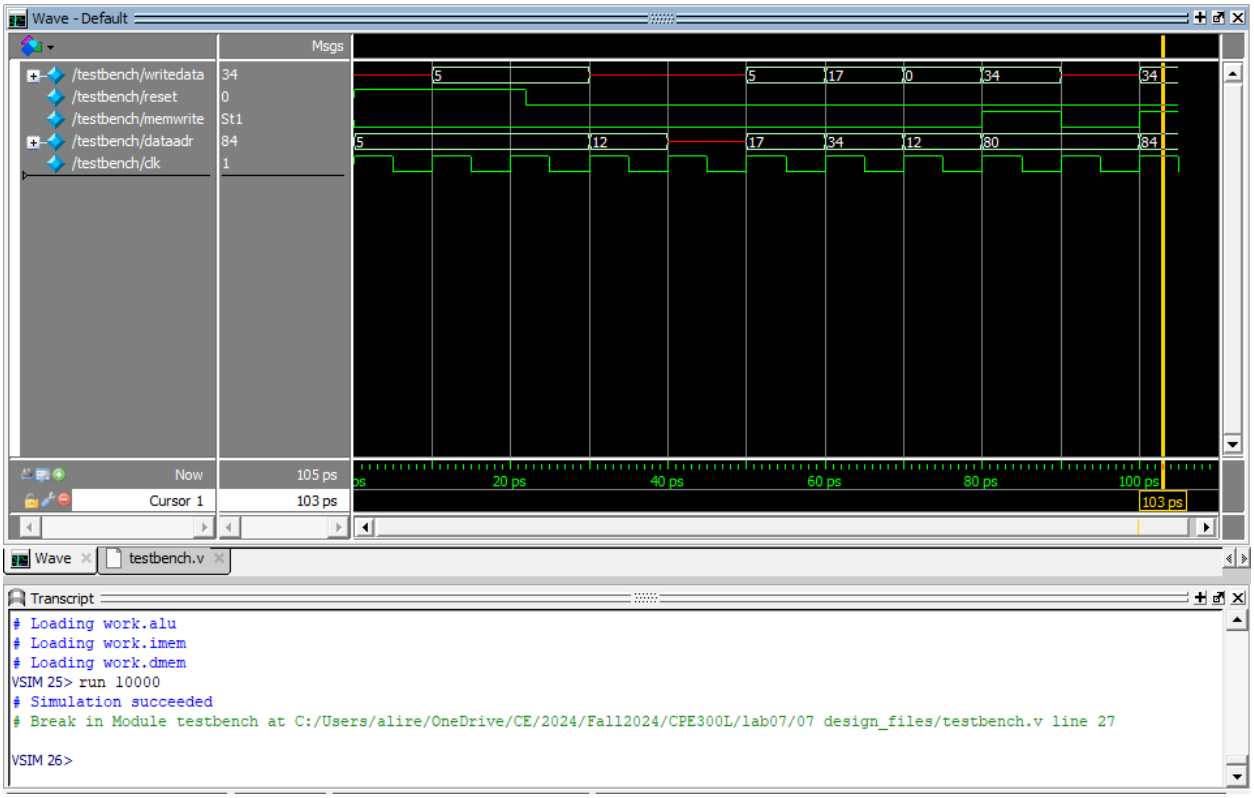
The testbench confirms that address 84 contains the value of 7 as expected from the MIPS instructions.

3. Results of the Experiments

The Verilog codes implementing JAL and JR instructions have been uploaded separately with comments indicating changes made.

The testbench and FPGA demonstration video have been uploaded separately (https://drive.google.com/file/d/1r9UwUYFsV419I7Dha8IATj270cHYMyJ2/view?usp=sharing). The following instructions were implemented on the Single-cycle MIPS to realize the function 2*(a+b) with a = 5 and b = 12. The machine code was generated using MARS. **In the video I mistakenly mention that the function squares the value of (a+b) instead of multiplying it by 2.**

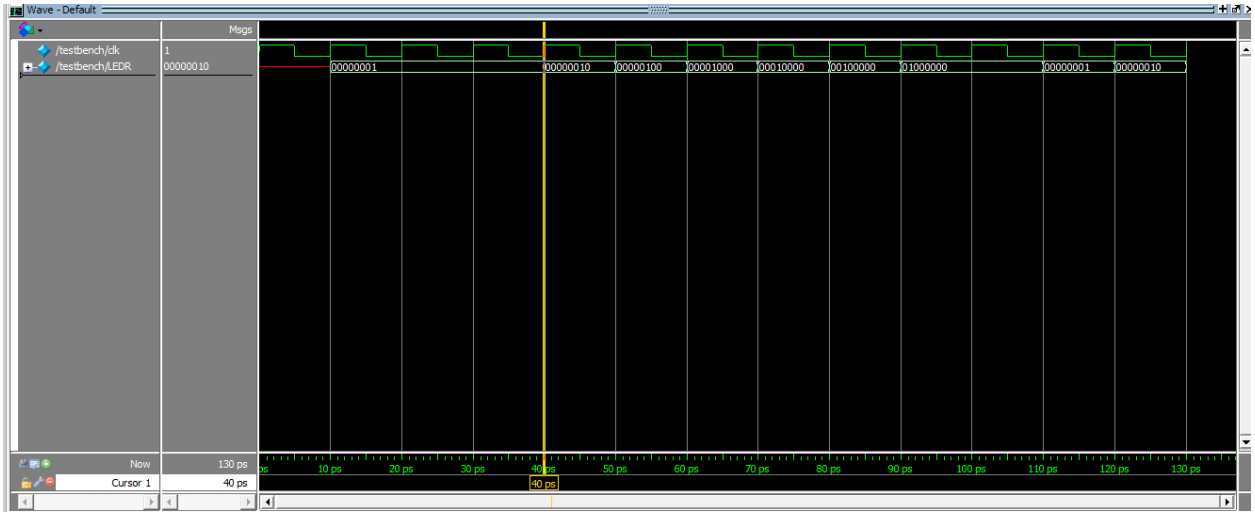
ADDRESS	Label	Instruction	MACHINE CODE	Comment (values of destination registers or memory locations)
0	main:	addi $4, $0, 5	0x20040005	Initialize $4 to 5
4		addi $5, $0, 12	0x2005000c	Initialize $5 to 12
8		jal func1	0x0c100006	Call func1
C		sw $4, 68($5)	0xaca40044	[80] = 34
10		lw $8, 80($0)	0x8c080050	$8 <= [80] = 34
14	end:	sw $8, 84($0)	0xac080054	[84] = 34
18	func1:	add $4, $5, $4	0x00a42020	$4 <= 12 + 5 = 17
1C		add $4, $4, $4	0x00842020	$4 <= 17 + 17 = 34
20		jr $31	0x03e00008	Go back to ra



The simulation waveform confirms that data address 84 contains the value of 34 which is the expected result.

The following instructions were implemented to realize the rotation of LEDs:

ADDRESS	Label	Instruction	MACHINE CODE	Comment (values of destination registers or memory locations)
0	main:	addi $4, $0, 1	0x20040001	Initialize $4 to 1
4		add $4, $4, $4	0x00842020	$4 <= 2
8		add $4, $4, $4	0x00842020	$4 <= 4
C		add $4, $4, $4	0x00842020	$4 <= 8
10		add $4, $4, $4	0x00842020	$4 <= 16
14		add $4, $4, $4	0x00842020	$4 <= 32
18		add $4, $4, $4	0x00842020	$4 <= 64
1C		j main	0x08100000	Jump to main



The simulation result shows that LEDR is shifting each clock cycle corresponding to the rotating operation of the LEDs on the FPGA.

Questions

1. How would you implement delay if needed in experiment 3? Say 10 sec between each led rotation.

To implement delay, we can change the count where the clock is asserted. For 10s delay we can increase the count to 5*50000000 = 250000000 with 5 seconds indicating half of the cycle. We must be careful, however, to make sure the new count has enough capacity to store this value.

2. What is the difference between jump vs jump and link instruction in MIPS?
Jump and link instruction is the same as the jump instruction with the additional process to save PC+4 into register ra ($31). It is commonly used to call functions at different memory locations in the instructions while saving the position to come back to when the function is completed.

Conclusions

In this lab I learned about single-cycle MIPS and how to modify the structure to implement new functionalities. The biggest problem I encountered was not knowing how to initialize instruction memory for FPGA MIPS implementation. I think one way is to manually initialize this memory using switches on the board to select memory locations and write into the memory. However, I decided to hard code the MIPS instructions into the imem file which provided me with the expected result from the MIPS instruction.