

Computer Organization – Lab Exercises

Demonstrate your solutions and get them evaluated by one of your TAs in the respective lab session.

Completion of assignment does not guarantee marks, you need to convince the TA that you did own the code by answering TA’s questions!

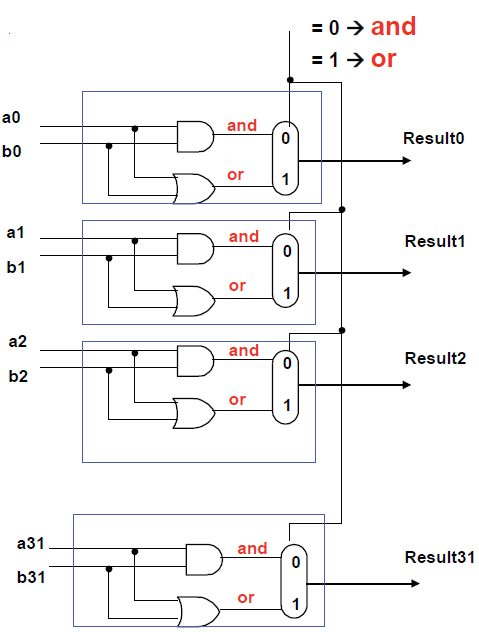
All these problems require installation of ARMSim# Emulator, Logisim, and few other tools

**ARMSIM Exercises:**

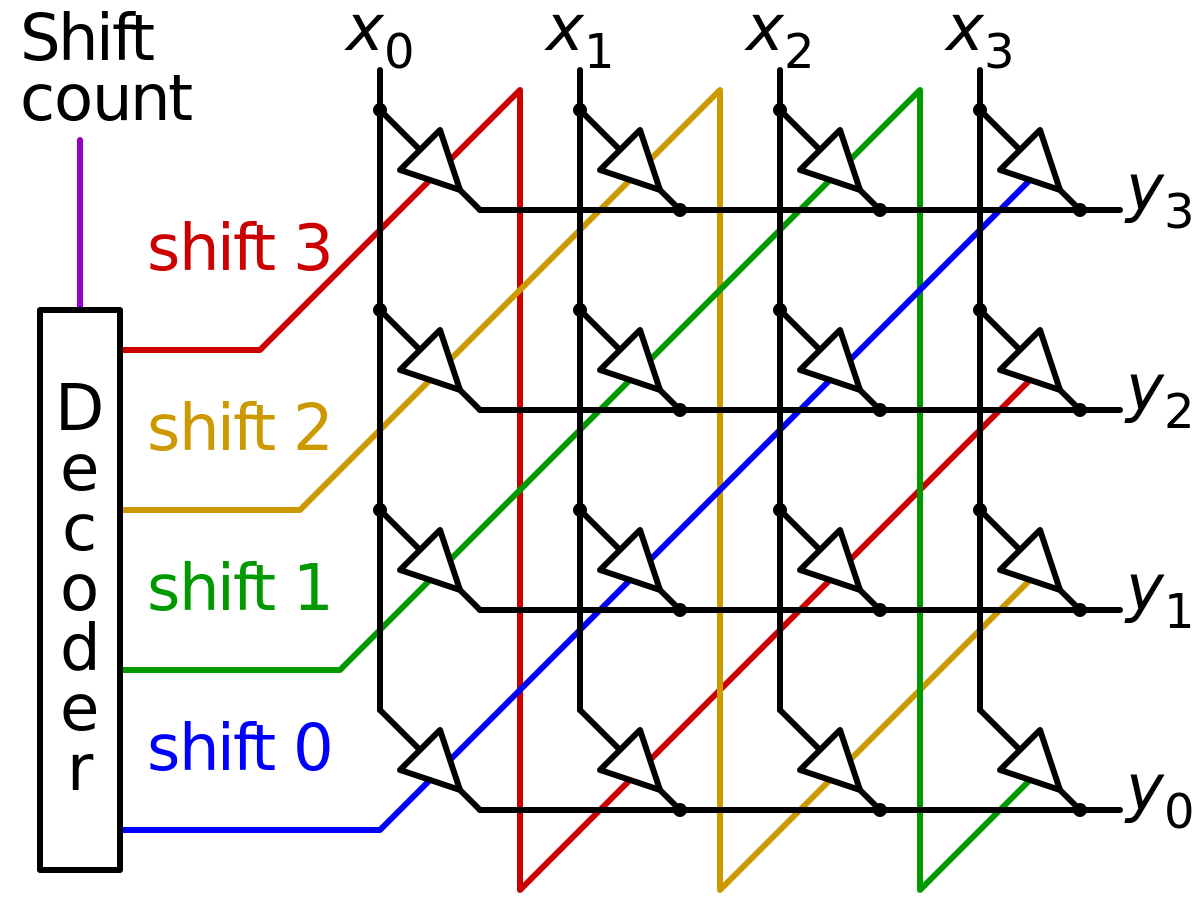
1. Write an ARM assembly program to WRITE to a file. Write 1 line of text to this file. Make sure you close the file after you write to it.
2. Write an ARM assembly program to read from the file that you created in Problem 1. Open this file in READ mode, READ and dump its contents to STDOUT.
3. Write an ARM assembly program to receive 3 different Integers (Comma separated) inputs from a file. After receiving all the three values, print them on STDOUT on separate lines**.**
4. In some single threaded embedded system applications, we use timers built using assembler sub-routine. This type of routines is sufficient and reasonably accurate for many applications. Develop a subroutine “*Stimer*” that can create 1000 (approximately) Clock cycles delay. Using this subroutine write another subroutine “*Ltimer*” that can create delay which are multiples of 10000 (approximately) cycles. The *Ltimer* is simple and no need to other timers. The *Ltimer* can be used to create periodic scan of I/O devices or any periodic activity.
5. Write ARM assembly code that can make the two red LEDs in *Embest Plugin of ARMSIM* to glow alternatively at observable rate. Your code should be user configurable for various rate. User timer developed in problem 4.
6. Write ARM assembly code that can make the 8-segment display to go from 0 to 9 at an observable rate. Your code should be user configurable for various rate.

**Logisim Exercises:**

1. Install Logisim. Complete **Beginners’** Tutorial and **Subcircuits** provided in the Help→User Guide
2. Using Logisim implement the following:
   1. 8-bit AND logic
   2. 8-bit OR logic
   3. 8-bit NOT logic
   4. 8-bit EXOR logic
3. Create simple 4-to-1 multiplexor. Use this for your next exercise.
4. Create a single selectable implementation of all the logic completed in previous exercise, similar to the AND-OR logic below.



1. Using Logisim implement the 4-bit barrel shifter with the corresponding 2-bit decoder as shown in the following figure.



1. Create a full adder then using this create a 4-bit Ripple Carry Adder using Logisim
2. Create 8-to-1 Multiplexor. Using this implement 3-input AND gate. Then using the same 8-to-1 Mux implement 3-input OR gate.

**ARMSIM Exercises (again):**

1. Write an ARM assembler program that increments the elements of a vector of size 100. After incrementing, copy the vector to a different part of the memory and add the two vectors and store the result where first vector was in. Repeat the above procedure three more times once with Multiplication (instead of ADD) by 5, once with divide by 4, and finally by adding 16384. Finally copy the vector to a file. Optimize code for execution time using appropriate cache configuration and choosing the appropriate assembly instruction. Solution with best execution time will be recognized suitably.
2. Plot your observations from problem 7 using Excel Spread Sheet.
3. Run the program that you wrote for Exercise 7 with ***unified cache with direct mapping and Write back****.* For the following combination of cache size and block size measure the *miss rate*. Plot your results and provide your observation. **Due on March 3rd or 4th.**

|  |  |  |
| --- | --- | --- |
| cache Size | Test 1 | Test 2 |
| Bock Size | Block Size |
| 128 | 16 | 32 |
| 256 | 16 | 32 |
| 512 | 16 | 32 |
| 1024 | 16 | 32 |
| 2K | 16 | 32 |
| 4K | 16 | 32 |
| 8K | 16 | 32 |

1. Run the program that you wrote for Exercise 7 ***split cache with direct mapping and Write back.***For the following combination of cache size and block size measure the hit and *miss rate*. Plot your results and provide your observation.

The next two exercises are optional, if you attempt and finish, get them evaluated from you TA; this could come handy later!

1. Run the program that you wrote for Exercise 7 with ***split cache with associative mapping and Write back****.* For the following combination of cache size and block size measure the hit and miss. Plot your results and provide your observation.

**Verilog Exercises:**

1. This exercise requires you to install Verilog development tools on your laptop and demonstrate Verilog “Hello World” Code. Use Icarus Verilog tools.
2. Demonstrate Verilog module with test bench using Icarus. This does not require you to write any code. Download code from Internet. Make sure you understand this code.
3. Develop Verilog code for behavioral design of ***Full Adder*** along with an appropriate Testbench. Demonstrate your code. Try all types of test benches: Simple testbench ♣ Self-checking testbench ♣ Self-checking testbench with test vectors
4. **GTKWave is a simple, lightweight application designed to read and display waveform files. The program is designed to help the IC designed view the signal transitions on the established timeline, as well as the interaction of signals from different designs. It can easily read and translate the hardware description language in waveforms.Follow the instructions in this web site to complete the Simulation with gtkwave tool. Follow the instructions in this web site to complete the Simulation with GTKWave tool**

<https://www.swarthmore.edu/NatSci/mzucker1/e15_f2014/iverilog.html>