Lab 9: Build a Computer

A conceptual block diagram of a simple computer is shown in Figure 1. In previous labs you have already designed the Addr_Mux, the ALU, the control unit (CCU) and the required registers. In this lab you will put all the components together to build a computer. The only missing block is the memory block which you can find here. You will also need the mem_init.v in which you are going to insert your program code. Use a graphical design to implement the computer as shown in Figure 1. Instructions on how to do that is provided.

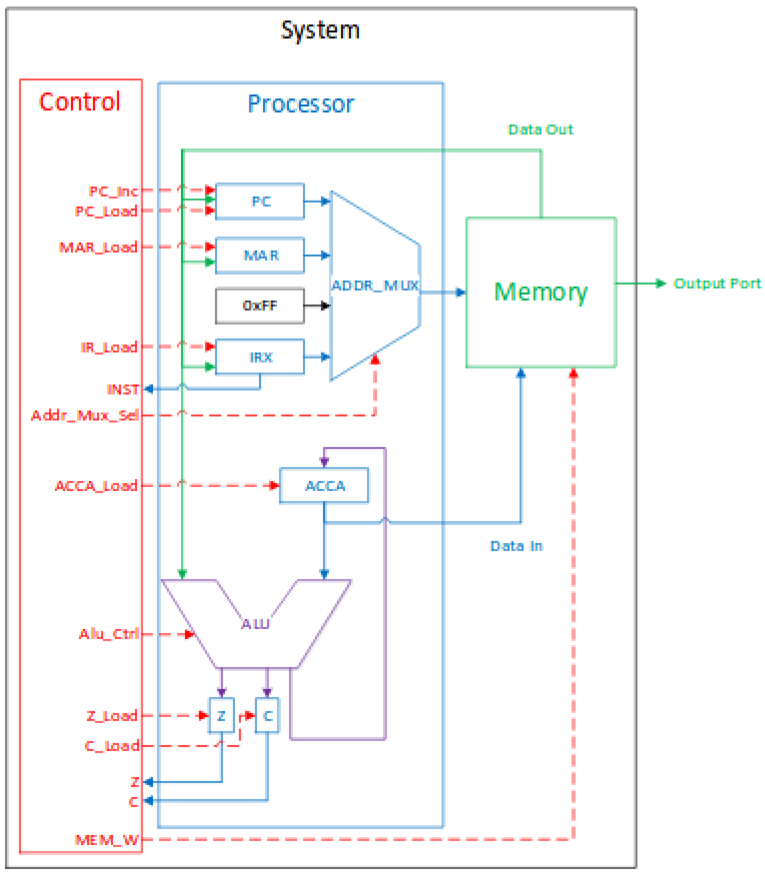

Figure 1. Processor Block Diagram

1 Prelab

1.1. Using the instruction set provided in Table 1, write a computer program to generate running lights. One way to accomplish this is to start with an 8-bit number 0000 0001 (where the one represents the LED that would be off). Then left shift that number once you have reached the end, jump back to the beginning of the program. Figure 2 shows the expected output at different time steps.

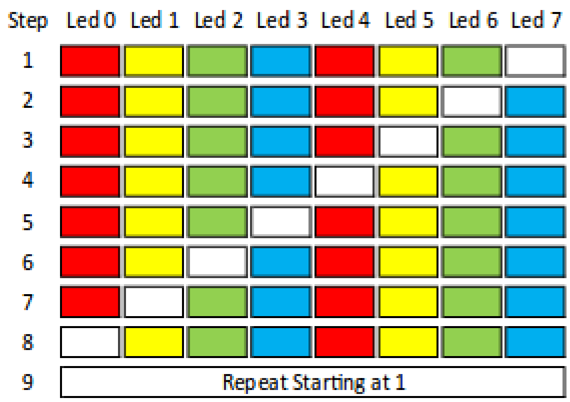

Figure 2: Running Lights LED Sequence

1.2. Using the graphical method as shown in Section 3, design and build the entire computer.

2 Lab

2.1. Enter your running light program into the mem_init.v file.

2.2. Simulate the computer you have created in the prelab.

2.3. Run your code on your board.

3 Supplement: Graphical Design of Processor

3.1. Start a new project and include the mem_block.v.

3.2. Include the other modules you need for this lab (e.g. the ALU.)

3.3. Create graphical symbols for each module

3.4. Open the mem_block.v file and select File > Create/Update > Create Symbol Files for Current File. This creates a memory block as shown below.

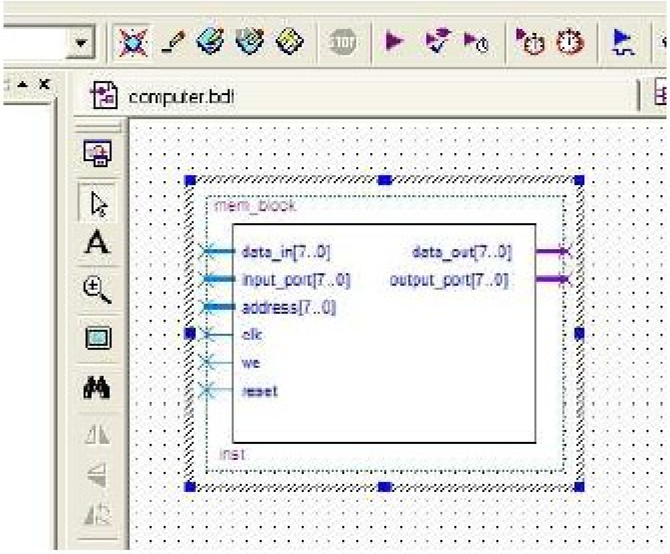

Figure 3. Memory Block Symbol

3.5. Open a block diagram file by File > New > Design Files > Block Diagram/Schematic file Input and output pins are located under primitives/pin. That creates a *.bdf file.

3.6. Right click in the main window and select Insert > Symbol.

3.7. Under you project directory select the mem_block you just created.

3.8. Keep adding the files that you have created in previous lab for remaining components of the computer. Each time create a Symbol file and added it to your main *.bdf file.

3.9. For the address mux and the reset constant you can use already existing block in Quartus. You can do that by Insert > Symbol, select ../quartus/libraries/, then select lpm_mux and/or lpm_constant which are located under megafunctions/gates. Input and output pins are located under primitives/pin.

3.1. Once you are done, start connecting the blocks as shown in Figure 1.

Appendix

Table 1: Computer Instructions

	Instruction	**Operation (Mnemonic)**
0	nop	Do nothing. (No Operation)
1	LDDA addr	Loads ACCA with the value in memory at address addr. C stays the same, Z changes. (Load ACCA from memory)
2	LDDA_IMM #num	Loads ACCA with num, the value in memory at the address immediately following the LDAA #num command. C stays the same, Z changes. (Load ACCA with an immediate)
3	STAA addr	Stores the value in ACCA at memory address addr. C stays the same, Z changes. (Store ACCA in memory)
4	ADDA addr	Adds the value in memory location addr to the value in ACCA and saves the result in ACCA. C and Z change. (Add ACCA and value in memory)
5	SUBA addr	Subtracts the value in memory location addr from the value in ACCA and saves the result in ACCA. C and Z change. (Subtract value in memory from ACCA)
6	ANDA addr	Perform a logical AND of the value in memory location addr with the value in ACCA. Save the result in ACCA. C stays the same, Z changes. (Logical AND of ACCA and value in memory)
7	ORAA addr	Perform a logical OR of the value in memory location addr with the value in ACCA. Save the result in ACCA. C stays the same, Z changes. (Logical OR of ACCA and value in memory)
8	CMPA addr	Compare ACCA to value in addr. This is done by subtracting the value in addr from ACCA. ACCA does not change. C and Z change. (Compares ACCA to the value in addr)
9	COMA	Replace the value in ACCA with its one's complement. C is set to 1 and Z changes. (Compliment ACCA)
A	INCA	Increment value in ACCA. C stays the same and Z changes. (INCA ACCA)
B	LSLA	Logical shift left of ACCA. C and Z change. (Logical shift left ACCA)
C	LSRA	Logical shift right of ACCA. C and Z change. (Logical shift right ACCA)
D	ASRA	Arithmetic shift right of ACCA. C and Z change. (Arithmetic shift right ACCA)
E	JMP addr	Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same. (Jump)
F	JCS addr	Jumps to the instruction stored in address addr if $C = 1$. If C is not set, continue with next instruction. C and Z stay the same. (Jump if carry set)
10	JCC addr	Jumps to the instruction stored in address addr if $C = 0$. If C is set, continue with next instruction. C and Z stay the same. (Jump if carry not set)
11	JEQ addr	Jumps to the instruction stored in address addr if $Z = 1$. If Z is not set, continue with next instruction. C and Z stay the same. (Jump if Z set)