EELE 367 – Logic Design
Final Project Description

"8-Bit Microcomputer Design"

Project Description

For your final project, you are going to design, implement and document a complete 8-bit microcomputer. The computer system you design will be delivered as a fully functional product that a user that can program using pseudo-assembly language. The computer system will have an 8-bit processor, 128 bytes of program memory, 96 bytes of data memory (e.g., RAM), 16x 8-bit output ports, and 16x 8-bit inputs ports. The I/O ports of the computer will be mapped to a variety of peripherals on the DE0-CV FPGA board. The computer will be programmed by inserting opcodes and operands in the program ROM.

You are going to design a model of your computer that can be simulated in ModelSim (computer.vhd). A test bench is provided for these simulations (computer_TB.vhd), however to verify the operation of your computer you will be primarily changing the instructions in your program memory. The test bench simply provides the clock, reset, and initial values on the input ports. It is important that your computer model is functionally verified in ModelSim before moving it to the FPGA board or debugging it will be very difficult.

You are also going to implement your computer on the DE0-CV FPGA board. You will not use your computer.vhd as the top level for your FPGA implementation. Instead you will create a top.vhd that will instantiate your computer.vhd. This will allow the top level to contain the port naming conventions that match the FPGA board I/O in addition to instantiating the precision clock divider and character display decoders.

Outcomes

After completing this project you should be able to:

- Describe the basic components and operation of computer hardware.
- Describe the basic components and operation of computer software.
- Design a fully operational computer system using VHDL.

Deliverables

The deliverables for the microcomputer are spread out over a ~four week period and are listed below. Once you miss the deadline for a part, you lose all of the points associated with that part. There is no credit for late work, but you will still need to complete the part to move onto the next one. This schedule is designed to keep you making progress on the project and not waiting until the last day to try to get it working.

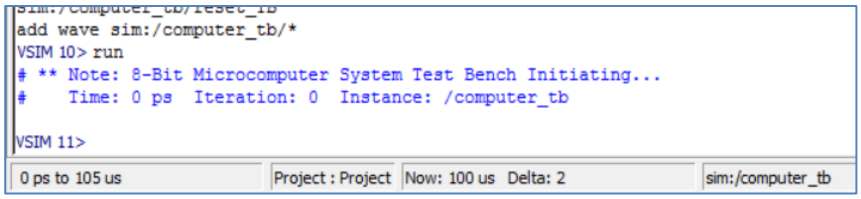
Part 1 – MC Quizzes & VHDL Shell **(Due Monday, 4/18/16 at 5pm, 10%)**

- Complete the **13.1 Multiple Choice Quiz** on Computer Hardware. (2.5%)
- Complete the **13.2 Multiple Choice Quiz** on Computer Software. (2.5%)
- Design the structural shell of the computer system based on the block diagrams provided below and perform a simulation with the test bench provided (computer_TB.vhd). You will need to create 8x VHDL files named as follows:

```
computer.vhd
└ cpu.vhd
    └ control_unit.vhd
    └ data_path.vhd
        └ alu.vhd
└ memory.vhd
    └ rom_128x8_sync.vhd
    └ rw_96x8_sync.vhd
```

In each of these files, you should enter the libraries, entity, and the architectural shell. Within the architecture shell, you should include all of your component declarations, associated signal declarations, a begin statement, and all component instantiations.

Recall that all that is needed for a VHDL model to compile and simulate is the correct syntax for the entity and architectural shell. Compiling your models will eliminate any syntax errors in your structure. Running a simulation will load all of your files and check that each component call has the appropriate port mappings. The simulation will not have any outputs, but it will verify the port mappings are accurate. This will allow you to get all of your connections correct before moving onto the behavioral modeling of the computer. The simulation will only run if all of the component port mappings are correct. Once the simulation runs successfully, you will get the following message in the transcript window. You will get a series of warnings, but that is OK for now.



Upload your 8 VHDL files (I don't need the test bench) and a screenshot of the transcript window (exactly the same as above) to the "**Project DropBox – Part 1**". (5%)

Part 2 – ModelSim Simulations of 4x Basic Instructions (Due Monday, 4/25/16 at 5pm, 40%)

- Implement the four instructions LDA_IMM, LDA_DIR, STA_DIR, and BRA. Perform a ModelSim simulation to verify the proper operation of your design. You will use the programs provided in exercise problems 13.3.1 and 13.3.2 from the text book.

- Upload your 8x VHDL files and screenshots of your simulations to the "**Project DropBox – Part 2**". Upload one screenshot demonstrating each instruction (4x total screenshots). Your screenshots should look nearly identical to examples 13.13, 13.15, 13.17, and 13.21 from the textbook.

Part 3 – Implementation on FPGA Board (Due Monday, 5/2/16 at 5pm, 20%)

- Implement your computer system on the DE0-CV FPGA board. You should continually read from SW(7:0) and write to LEDR(7:0), HEX(1:0), HEX(3:2), and HEX(5:4). You can accomplish this using the instructions you have already implemented in Part #2. Note that the FPGA block diagram uses your clock_div_prec.vhd with select lines coming from SW(9:8) and has character decoders driving each HEX display. You must demonstrate your program to the lab instructor and upload your VHDL files (including your top.vhd) to the "**Project DropBox - Part 3**".

Part 4 – Implementation of Additional Instructions (Due Thursday, 5/5/16 at 10am, 30%)

- Want to earn more than 70% on this project? Then just implement more instructions! Each instruction you implement from the list below is worth a certain number of points added to your grade. To get credit for the instruction, you **MUST** provide a **state diagram**, a **ModelSim waveform** showing proper operation of the instruction, and a **demonstration** of the instruction used in a program on the DE0-CV FPGA Board. You will upload your state diagrams, screenshots of simulations, and VHDL files to the "**Project DropBox – Part 4**".

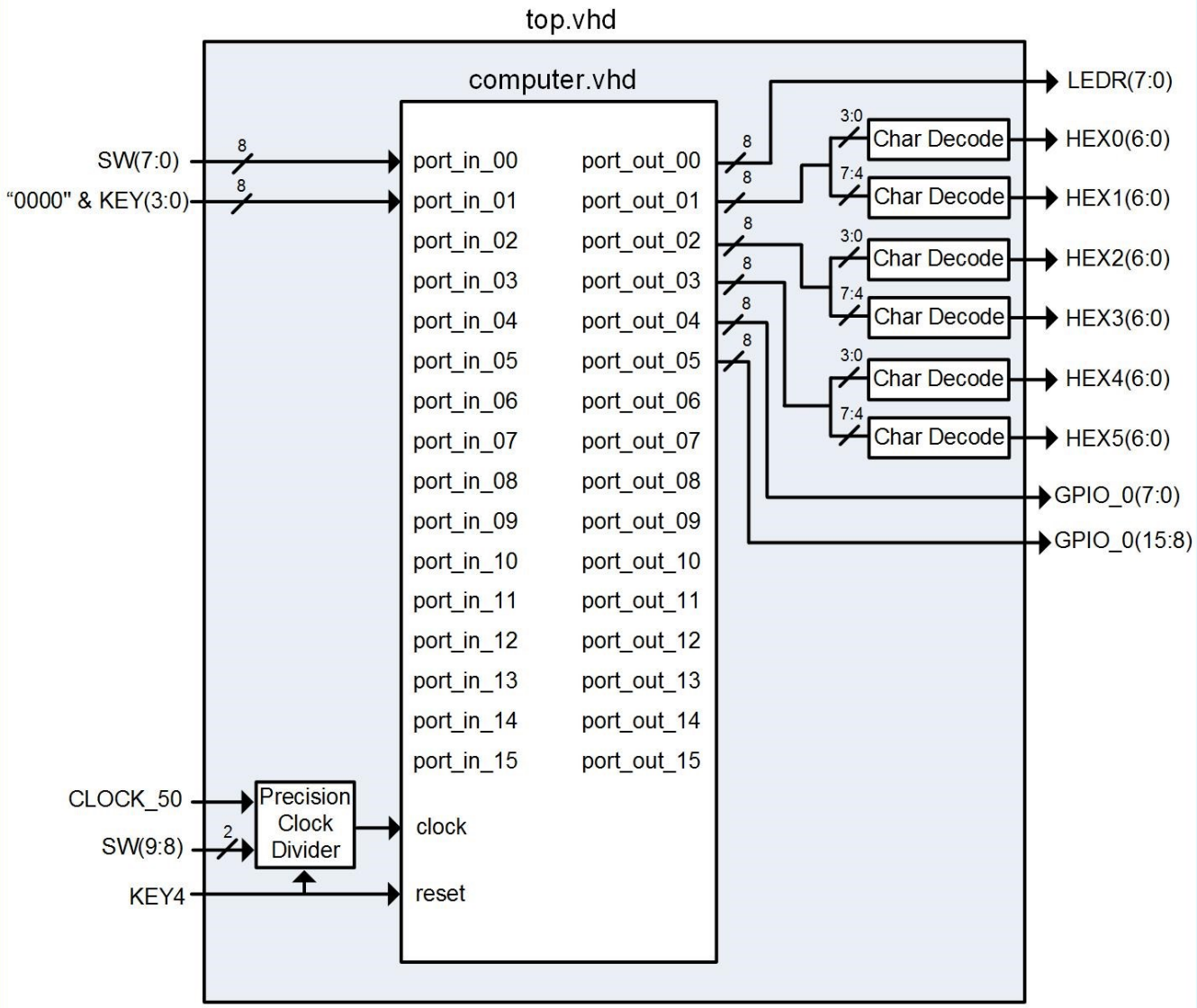
  Note that you can likely create a single program to test numerous instructions and only perform one simulation and one demonstration.

Instruction	Worth	Demo
LDB_IMM	2%	_____
LDB_DIR	2%	_____
STB_DIR	2%	_____
ADD_AB	2%	_____
SUB_AB	2%	_____
AND_AB	2%	_____
OR_AB	2%	_____
INCA	2%	_____
INCB	2%	_____
DECA	2%	_____
DECB	2%	_____
BEQ	2%	_____
BCS	2%	_____
BVS	2%	_____
BMI	2%	_____

System Architecture

DE0-CV Top-Level FPGA Port Mapping

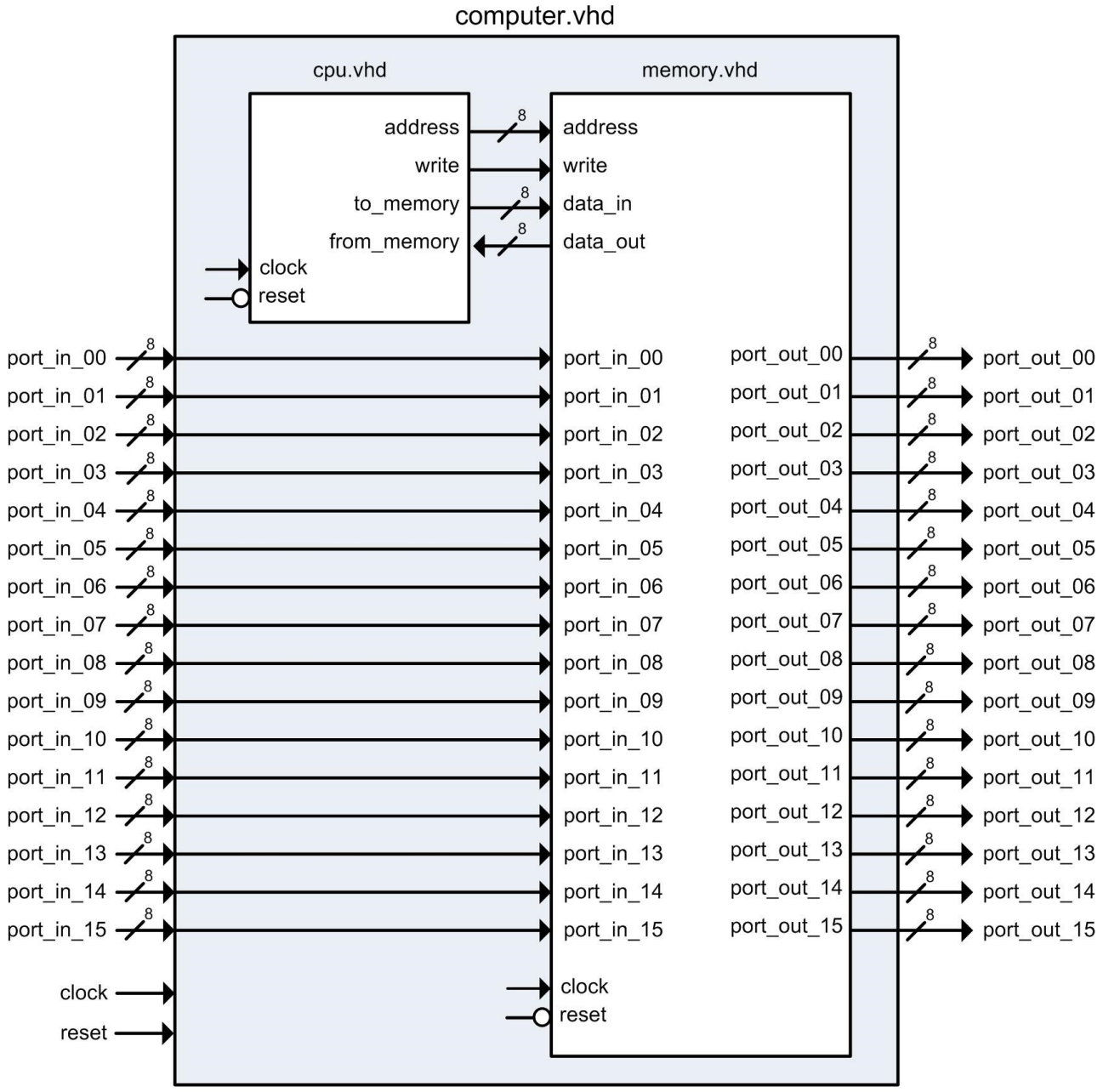
The following shows how the I/O of the microcomputer maps to the I/O on the FPGA.



NOTE: All of the following figures are straight out of the textbook. If you find a typo, please let me know!

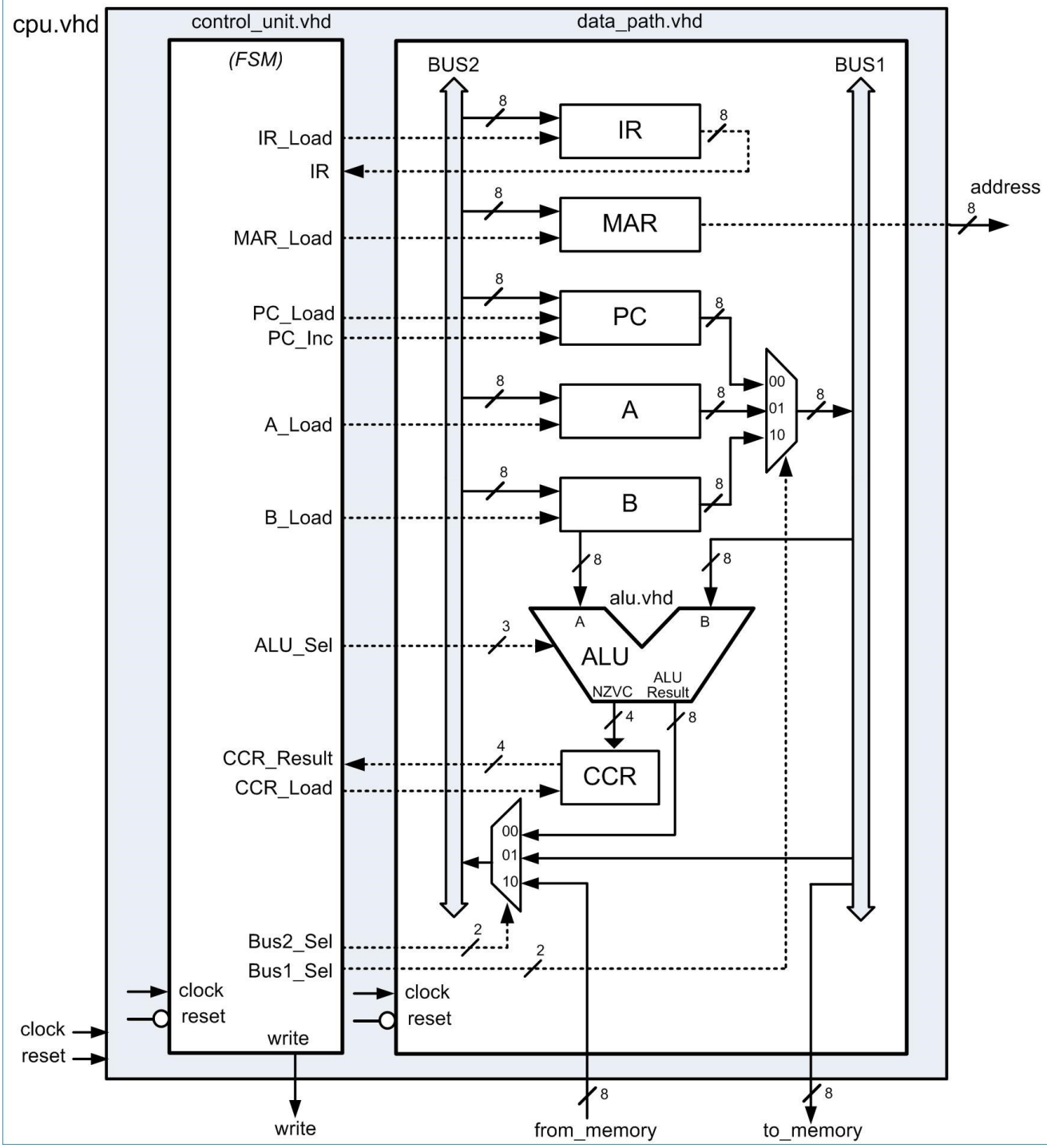
Example: Top Level Block Diagram for the 8-Bit Computer System

The following is the top level block diagram for our 8-bit computer system example.



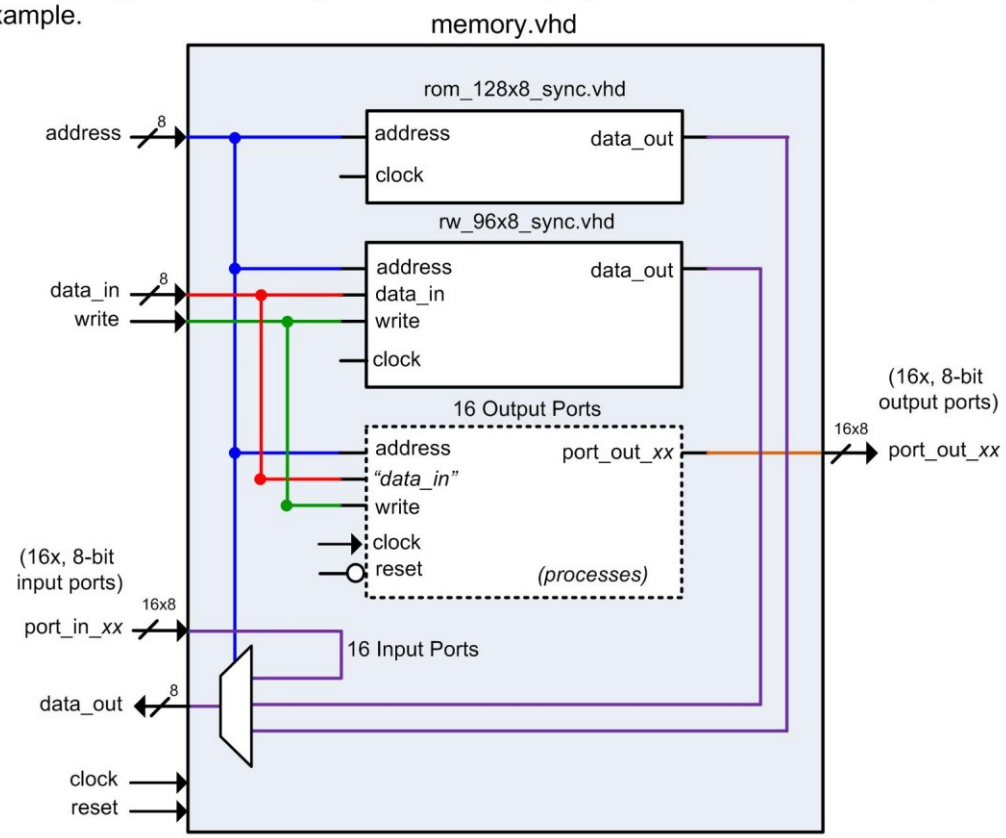
Example: CPU Block Diagram for the 8-Bit Computer System

The following is the block diagram for the CPU of our 8-bit computer system example.



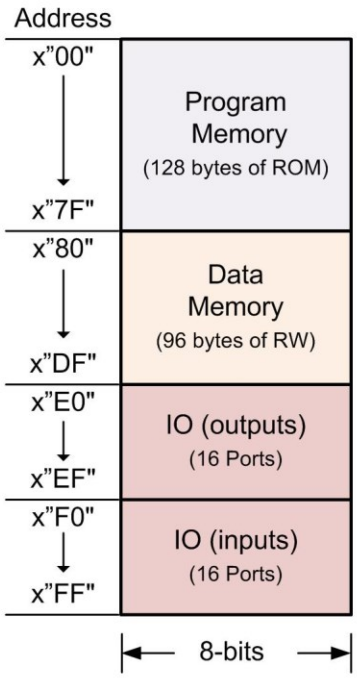
Example: Memory System Block Diagram for the 8-Bit Computer System

The following is the block diagram for the memory system of our 8-bit computer system example.



Example: Memory Map for a 256x8 Memory System

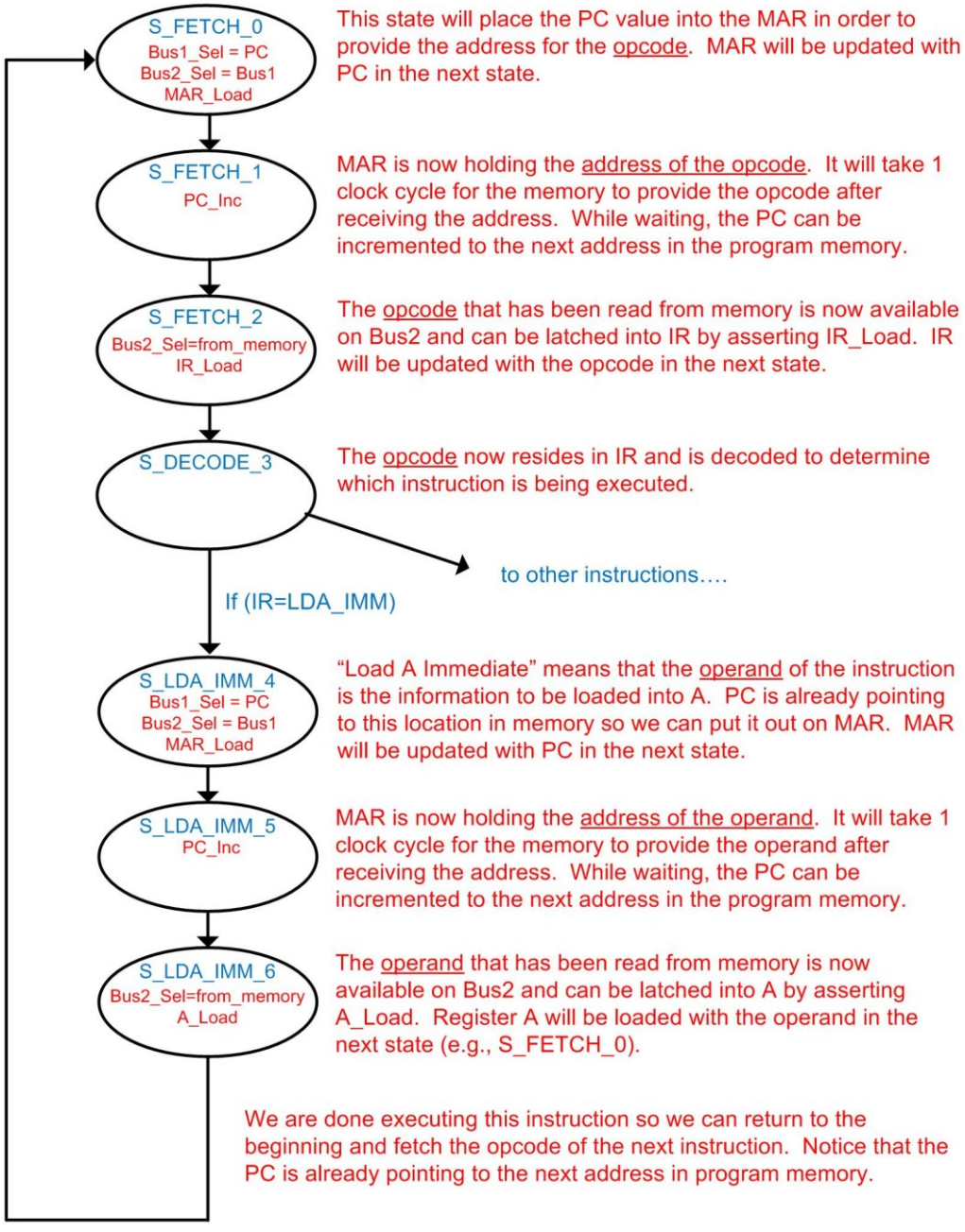
The following is a memory map for an example 8-bit computer system.



Instruction Execution

Example: State Diagram for LDA_IMM

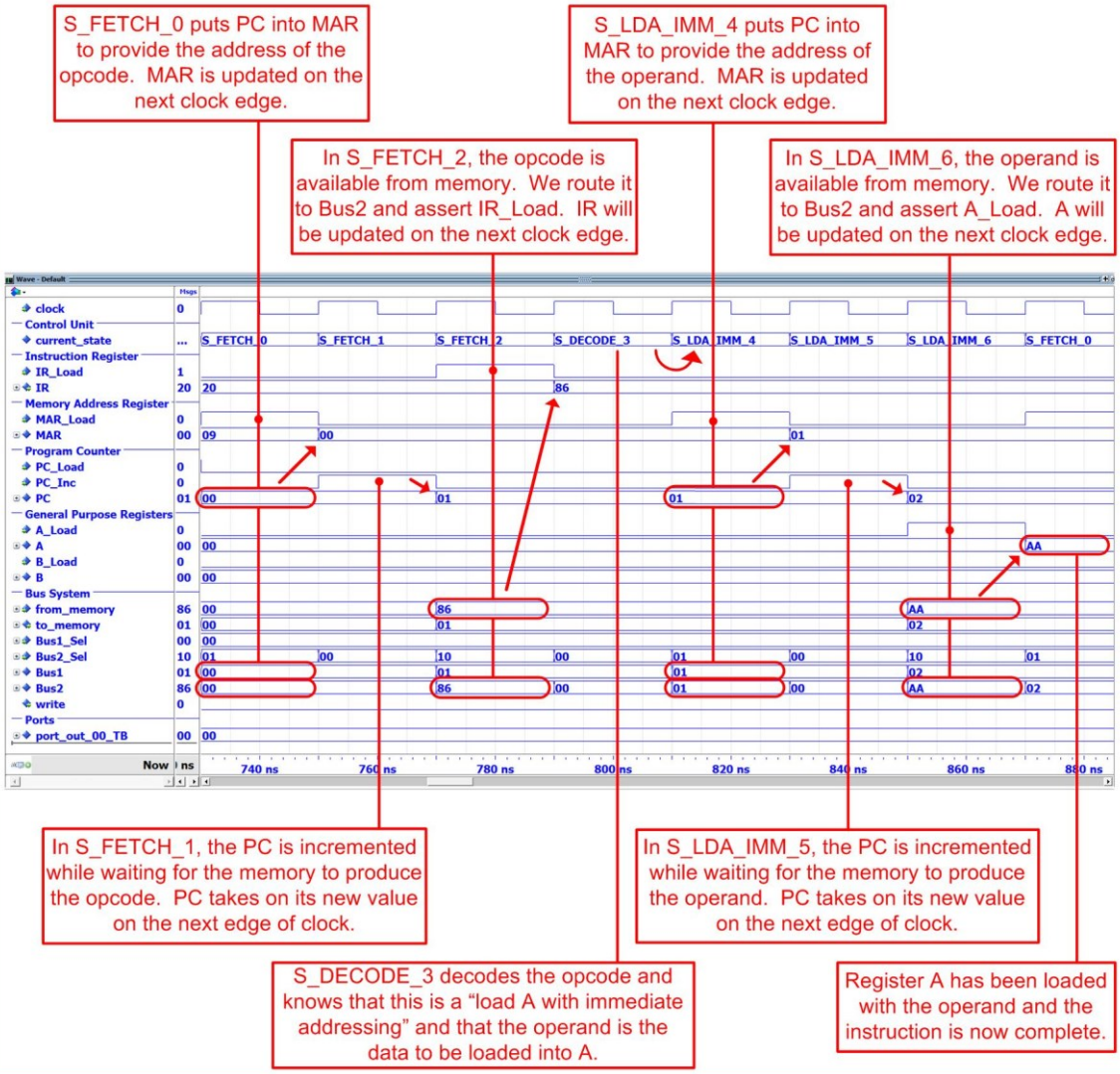
The following is the state diagram for LDA_IMM. This load instruction will move information from memory into register A. Immediate addressing implies that the information to be put into A is provided as the operand of the instruction.



Example: Simulation Waveform for LDA_IMM

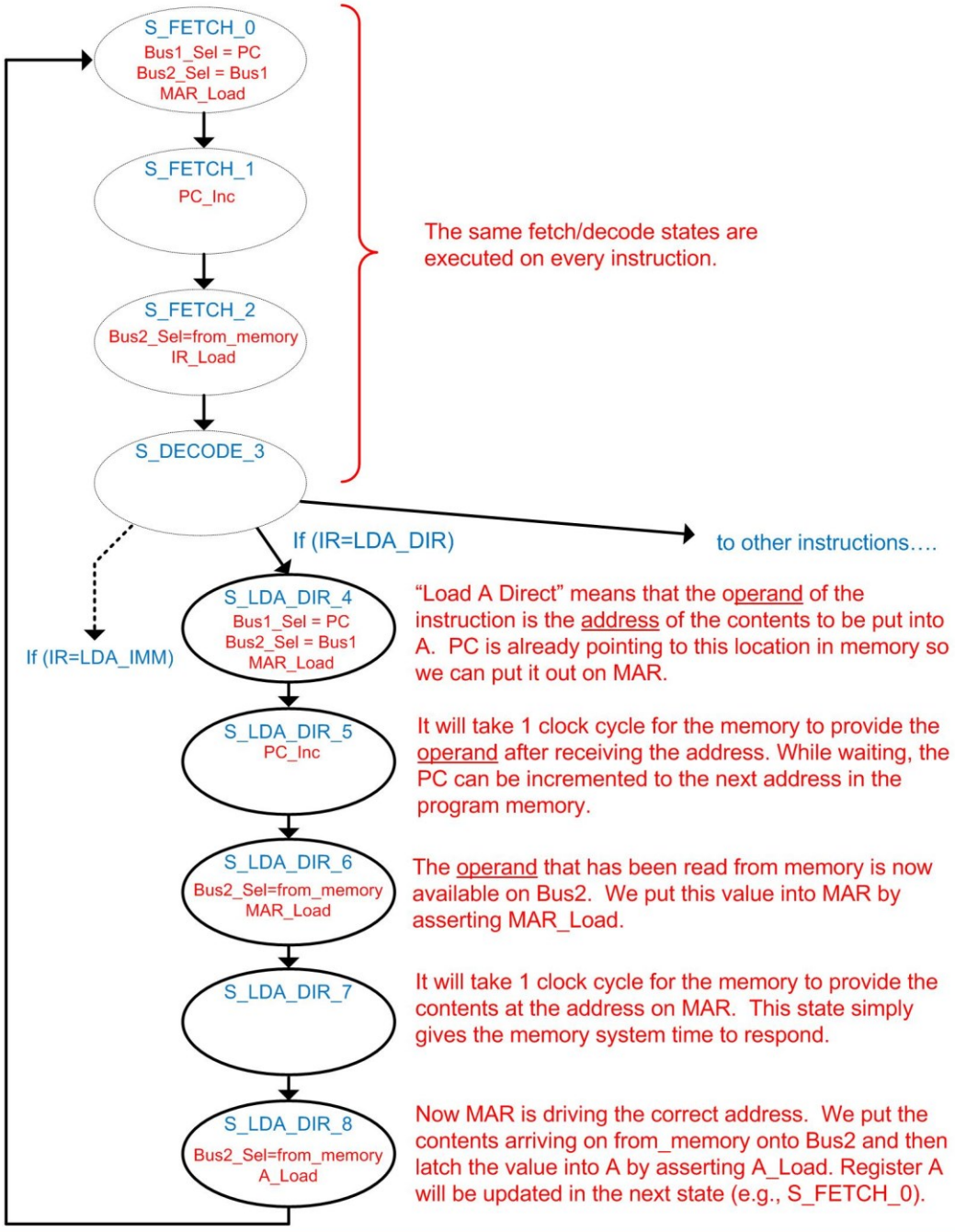
Let's look at the timing diagram when executing the following load instruction located at addresses x"00" and x"01" in program memory. The opcode for this instruction is x"86".

LDA_IMM x"AA"



Example: State Diagram LDA_DIR

The following is the state diagram for LDA_DIR. This load instruction will move information from memory into register A. Direct addressing implies that the information to be put into A is located at the address provided as the operand of the instruction.

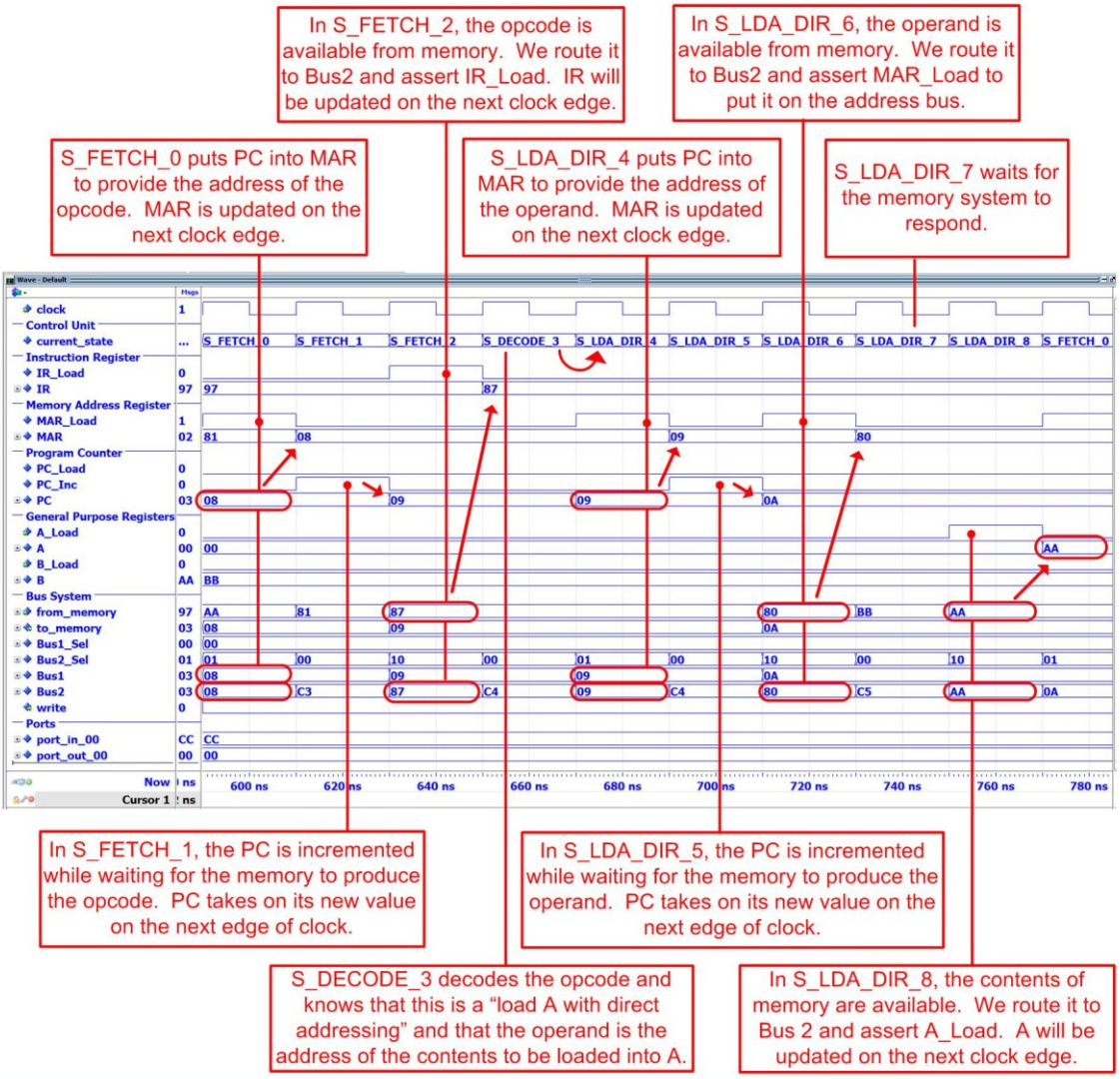


\

Example: Simulation Waveform for LDA_DIR

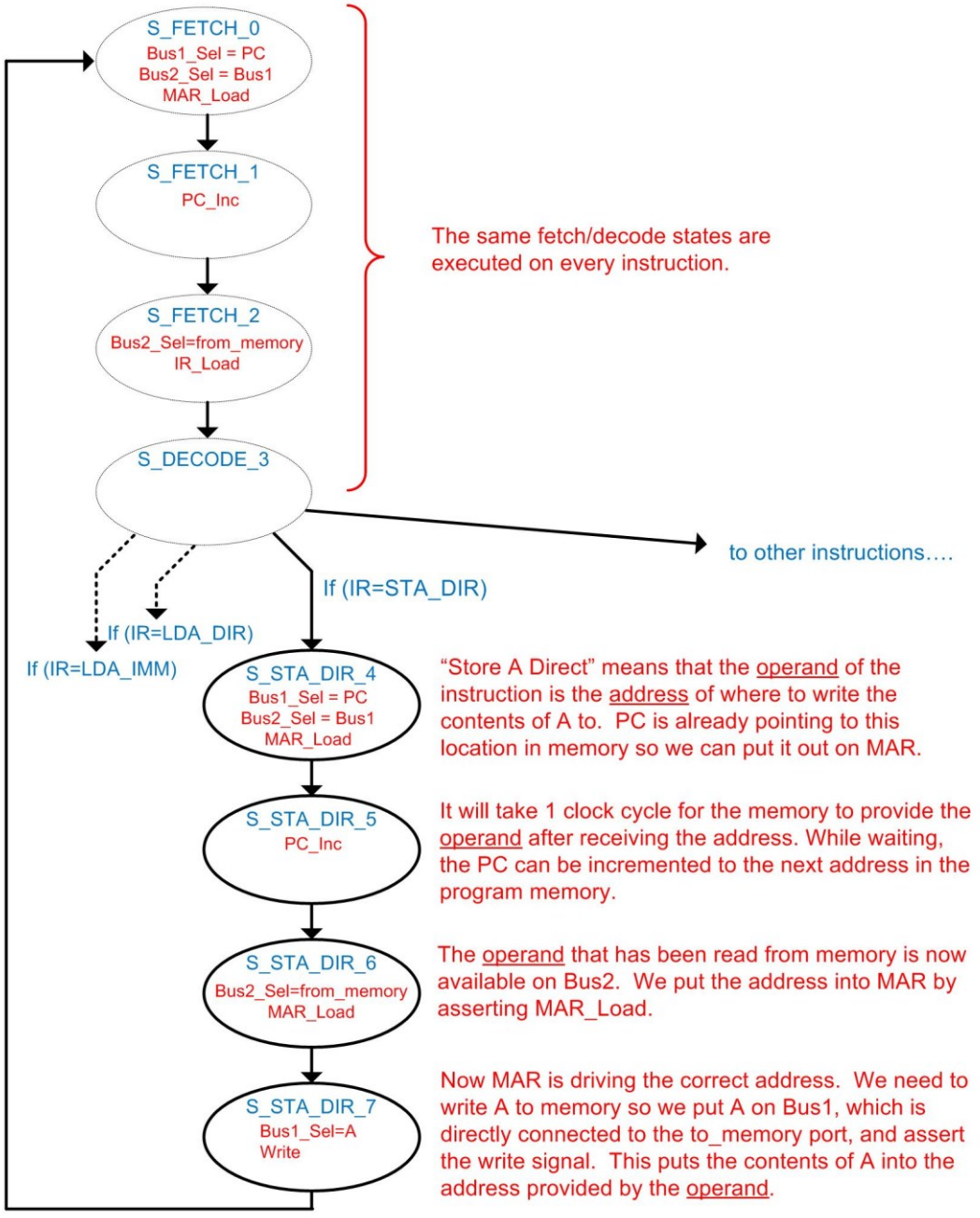
Let's look at the timing diagram when executing the following load instruction located at addresses x"08" and x"09" in program memory. The opcode for this instruction is x"87". The address x"80" is in data memory, which in this example is already holding x"AA" prior to this instruction.

LDA_DIR x"80"



Example: State Diagram for STA_DIR

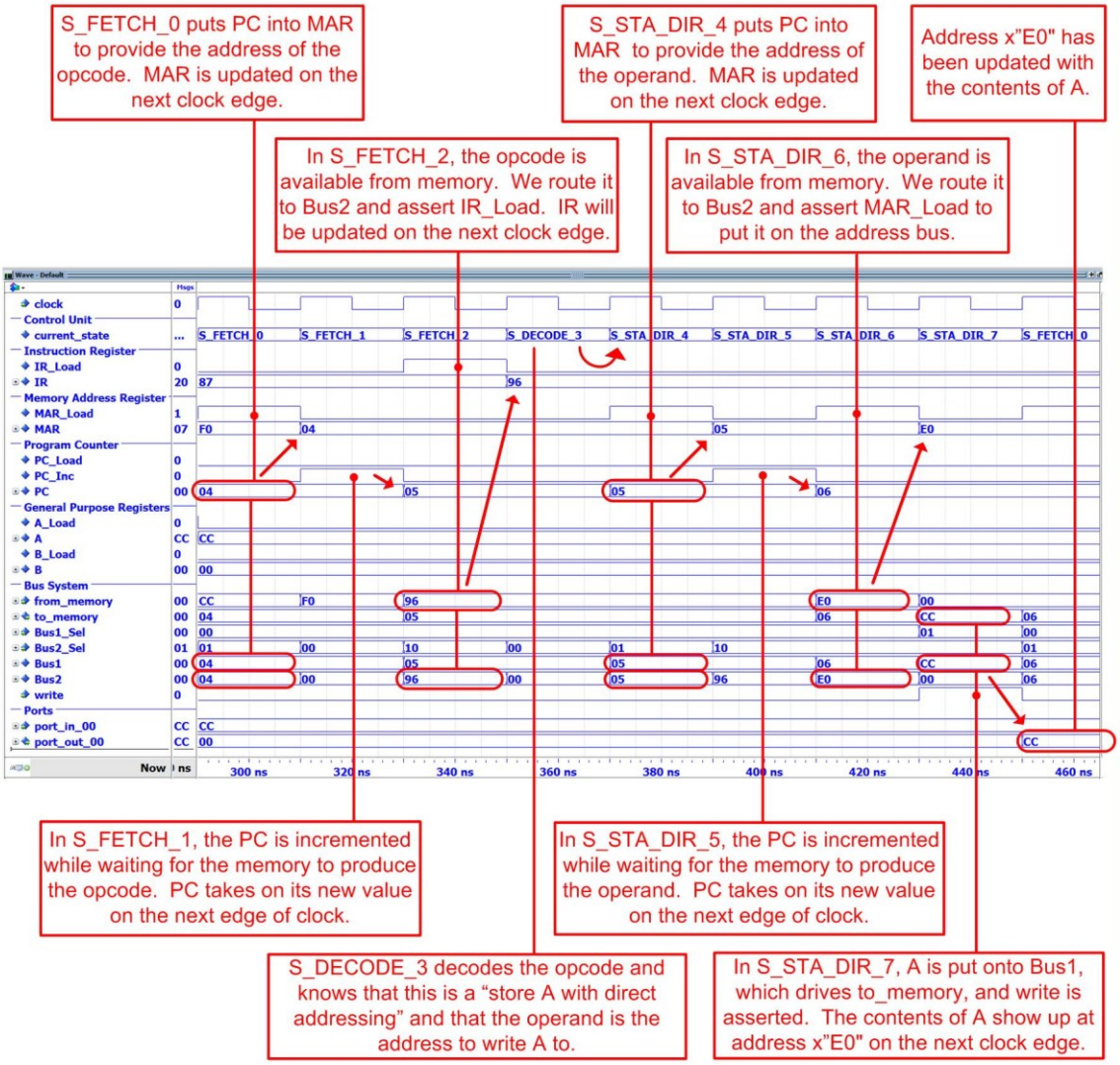
The following is the state diagram for STA_DIR. This store instruction will move information from register A into memory. Direct addressing implies that the operand provides the address of where to store A to.



Example: Simulation Waveform for STA_DIR

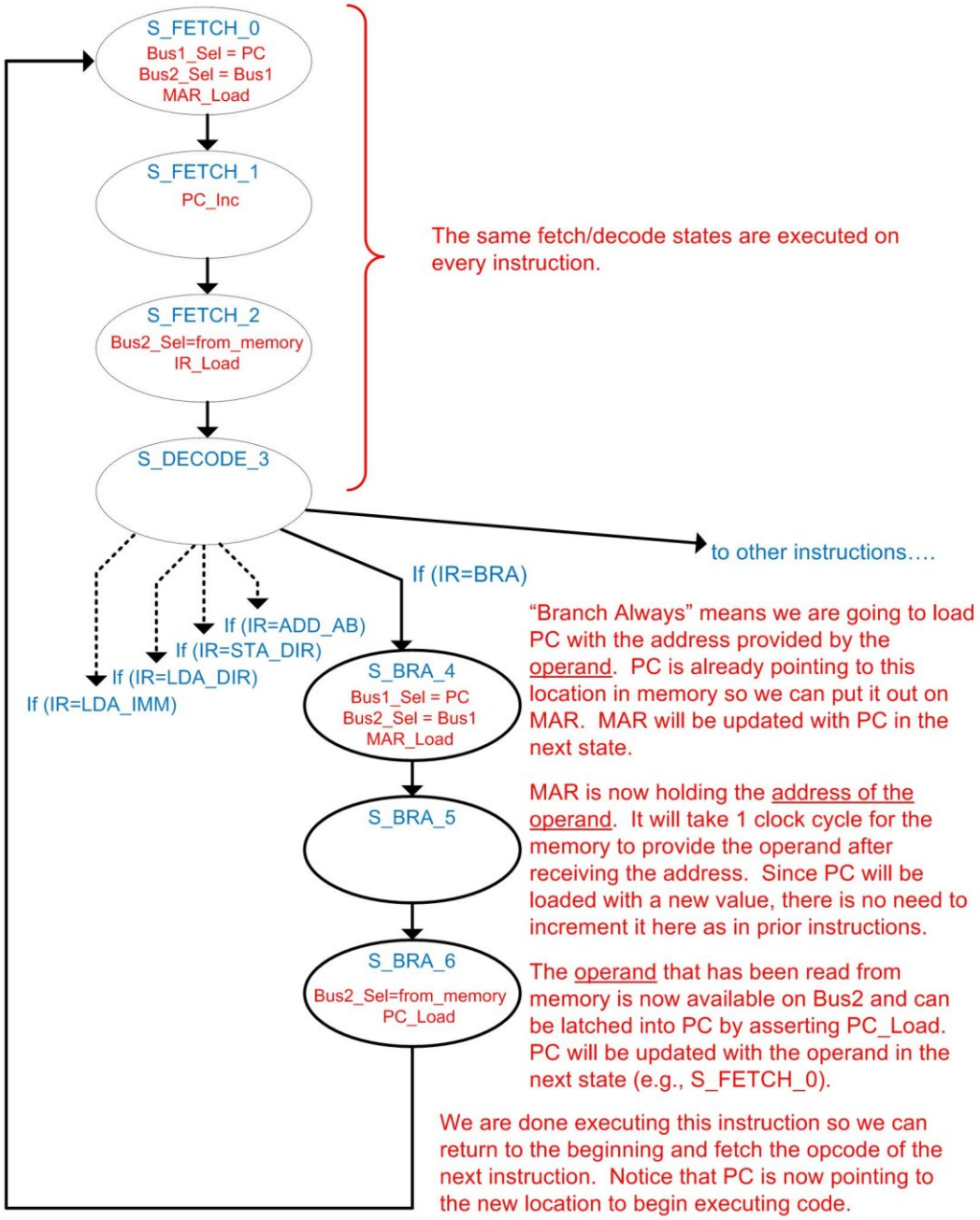
Let's look at the timing diagram when executing the following store instruction located at addresses x"04" and x"05" in program memory. The opcode for this instruction is x"96". The address x"E0" is for port_out_00. A already contains x"CC".

STA_DIR x"E0"



Example: State Diagram for BRA

The following is the state diagram for BRA. This instruction will load the program counter with the address supplied by the operand of the instruction. This has the effect of setting the address of the next instruction to be executed to a new location in program memory.



Example: Simulation Waveform for BRA

Let's look at the timing diagram when executing the following branch always instruction located at addresses x"06" and x"07" in program memory. The opcode for this instruction is x"20".

BRA x"00"