CSCE 3953

Project Report 6

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Landon Reynolds

010975968

SEC001

12/7/23

**OBJECTIVE**

The objective of project six was to create a 16-bit microcontroller in the Verilog programming language. This microcontroller consists of an arithmetic logic unit, several D flip-flops some of which interact with the memory and decoder, several tri-state buffers, a program counter, a memory container, an instruction decoder, and six finite state machines that each execute their own type of instruction.

**DESIGN METHODOLOGY**

D Flip-Flop (DFF):

D Flip-Flops are used throughout the top-level design. They take form as an MAR (memory address register), MBR (memory buffer register), IR (instruction register), GPR’s (general purpose registers), I/O ports, and most frequently registers to latch data from the 16-bit bus.

Tri-state Buffer (TSB):

Tri-state Buffers are also used many times in the microcontroller design. They are exclusively used to drive the bus; anything that outputs to the bus should have a tri-state buffer attached to it.

Arithmetic Logic Unit (ALU):

The ALU takes in two values from a 16-bit bus and, depending on the instruction (add, addi, sub, subi, and, or, not, move, etc.), outputs the result back to the bus.

Program Counter (PC):

The program counter is simply a way for the microcontroller to sequentially execute instructions. It outputs a value that represents which instruction should be run next.

Memory:

The memory module contains the instructions to be executed by the microcontroller. These instructions are in the form of machine code (16-bit binary value) that is decoded and executed according to the encoded bits.

Decoder:

The decoder breaks the instruction into three or, depending on instruction, two parts. The four most significant bits are called the Operation Code or “opcode”, this tells the microcontroller what to do with the following twelve bits. The remaining twelve bits are split into two 6-bit binary values called registers and represent a number or register location. Based on the value of the opcode, the microcontroller will execute some task on the registers.

Finite State Machines (FSM’s):

In my microcontroller design there are six finite state machines. One for extracting the instruction from the memory, two for register based and immediate based ALU tasks, two for register based and immediate based data transaction, and one for writing to and reading from the memory. These FSM’s are initiated by the decoder based on the opcode of the given instruction.

**INSTRUCTION SET**

MOVI R0, #19   
MOV R3, R0   
ADDI R3, #3   
SUBI R0, #2   
XOR R0, R3   
NOT R3   
STORE R0, (R3)   
LOAD (R3), P0

**ASSUMPTIONS**

The important assumptions made relate to the GPR’s and decoder implementation. I chose to have four GPR’s to allow for more microcontroller storage and versatility. I arbitrarily chose which FSM executes based on the opcode.

**SYNTHESIS**

The logical synthesis process initially allowed the PC, DFF, TSB, and all FSM’s (besides alu and load/store one’s) to synthesize. The combination of synthesized and non-synthesized modules would have been inundated with glitches that may or may not have affected the execution of the microcontroller’s instructions.

**SIMULATION METHOD**

The microcontroller will be subjected to testing via a test bench. This test bench instantiates the memory module and microcontroller top-level module. To initiate the instruction execution, clock is set to 0, a place holder value is set in I/O port 1, and reset is kicked off. The execution is graphically represented by a waveform seen in the result section of this report.

Instruction code translated to machine code:

1011000000011001

1010000011000000

1000000011000011

1001000000000010

0110000000000011

0010000011000000

1101000000000011

1100000011000100

**RESULT**

Instruction 0:

A screen shot of a computer

Description automatically generated

Complete Instruction Execution:

A screen shot of a computer

Description automatically generated

**ANALYSIS AND CONCLUSION**

Overall, the microcontroller seems to have executed the first instruction correctly, however soon after the second instruction has been initiated, the program counter increments, and it is not able to fully execute its instructions. This pattern then continues for the subsequent instructions. Overall, I have put more than 100 hours into learning the many capabilities of programs like ModelSim and Design Vision. In the past I have created each of the components of this design individually, synthesized them, and simulated them. The culmination of my efforts leads to the construction of this microcontroller.