ELEC2204 Computer Emulation in C

Harry Beadle
27770834
hb11g15

February 10, 2017

Abstract

The design and implementation of a processor architecture as an emulator written in C. Including a definition of a simple assembler language and a compiler program to allow the emulated CPU to run generic programs. Additionally a `DEBUG` mode that shows the internal state of the emulated machine.

1 Architecture

The processor is designed on a 16-bit architecture. Instructions contain an opcode and one 12-bit or two 6-bit operands.

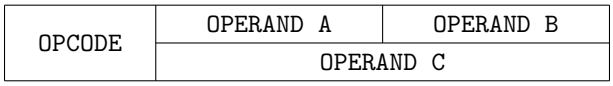

Figure 1: Graphic showing the two possible make ups of an instruction.

1.1 Opcodes

Table 1: Opcodes and their meanings.

Code	Abbreviation	Description
Maths		
0x0	ADD	Add Operand A or B
0x1	SUB	Subtract Operand A from B
Logical Operations		
0x2	AND	Bitwise AND of A and B
0x3	OR	Bitwise OR of A and B
0x4	NOT	Bitwise NOT of A
Control		
0x5	JMP	Jump to C
Movement		
0x6	STO	Store the value in the accumulator in C

2 Memory

The memory design is simple, generic, 16-bit memory. It takes three control signals, and if it's given an unexpected signal it returns 1, causing the emulation to stop.

It is connected directly to the Control Unit via an `address` bus and to the rest of the processor by the `data` bus.

The size of allocated memory is determined by a `#define` called `MEMORY_SIZE`.

Table 2: Control signal names and descriptions for the ALU.

Signal	Description
`MEM_HIZ`	Don't drive the `data` bus.
`MEM_SET`	Set the memory at the `address` to the value of the `data` bus
`MEM_ENB`	Drive the `data` bus with the data stored at the `address`.

2.1 Implementation

The memory is implemented in the `memory.c` and `memory.h` files. The function `updateMemory()` is called on each clock tick. It consists of a single `switch case` statement that handles the behavior defined above.

2.2 Testing

The memory is tested by setting a reading each possible value to each cell in memory and then reading it back. This test can be found in `test/memory_test.c`.

If the operation fails then some debugging data is output to `stdout`: the address, data and state of `memory_control`.

3 Arithmetic and Logic Unit

The ALU takes two inputs, and outputs them onto the data bus when a control input other then `ALU_HIZ` is given.

In order to simplify the control unit the ALU takes the same control input as the opcode. This means the only input left to define is `ALU_HIZ` which is defined as `0xFF`.

3.1 Implementation

The ALU is implemented in the `alu.c` and `alu.h` files. The function `updateALU()` is called on each clock tick. It consists of a single switch case that switches on the `alu_control` input. The data bus is then driven with the output based on the inputs. Input buffering is handled by the control unit.

3.2 Testing

ALU testing is handled by `test/alu_test.c`. It cycles though each of the operations with all possible inputs and asserts that they are correct. If the inputs are not correct then debugging data is printed to `stdout`: the type of operation, the two inputs and the output of the ALU as it drives the data bus.

4 Control Unit

5 Wrapper

6 Assembler