A Classical CPU

Cache Me Outside

Allan Wang (260667681), Isaac Sultan (260680295) and Adam Edery (260691043)

Due: April 12, 2017

Professor: Joseph Vybihal

Course: Comp-273-001 Intro to Computer Systems

1. Image of the CPU

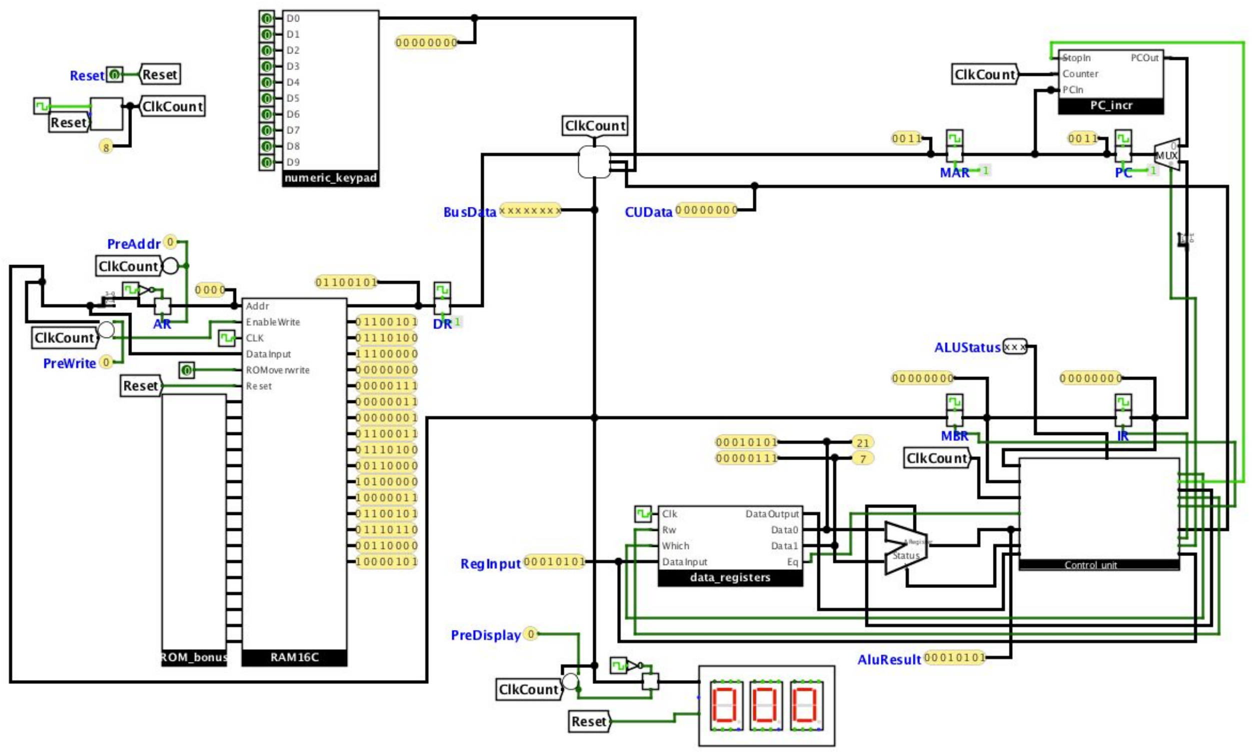

Figure 1.1: An image of our CPU, Cache Me Outside, as rendered by Logisim

2. Description of components

RAM: The RAM is composed of 16 single byte storage units. These single byte storage units are in turn composed of 8 JK flip-flops, each storing one bit of data. The RAM accesses the correct byte by decoding the address that it receives from the AR using a 4-bit decoder. It accepts a single bit for read/write bit that enables the correct single byte to be overwritten by the data input. The address is also passed to a multiplexer that allows the correct byte to be output by the RAM to the DR. The AR is a 4-bit register that stores the address in RAM being accessed, and the DR is a 8-bit register that stores the contents of this address.

PC: The PC is a simple 4-bit register that stores the address in RAM of the current instruction being executed. It is connected to another circuit that increments the value of its instruction when the CPU is ready to move on to the next instruction. The value stored in PC is used in a possible two ways. A stop instruction may be executed, preventing the incrementer from incrementing the PC. This forces the CPU to continue executing the stop instruction forever. The second option occurs when a jump instruction is executed. The PC is set to the address in the IR, making the PC obtain the instruction of the next address.

IR: The IR is an 8-bit register that stores the instruction. It gets its input from the MBR and passes it to the CU. The IR also sends the address portion of its data to the PC, allowing for jump instructions to be executed.

MBR: The MBR is also an 8-bit register that holds any byte passed from the system bus that may be needed. This includes the instructions it receives from RAM, as well as data from the RAM or keypad.

ALU: The ALU takes a mode signal that instructs whether addition, subtraction or multiplication will be performed. It then takes both data registers, adding them together in an 8-bit adder, subtracting register 2 from register 1 or multiplying the two registers. Multiplication was implemented with a sequence of 4 bit and 6 bit additions that simulate the shift and addition algorithm of multiplication. Since our maximum output is an 8 bit integer, multiplying numbers greater than 4 bits each causes an overflow The ALU outputs to a status register that says whether the inputs were too large (for multiplication), or the result was negative or zero or had overflow.

Data Registers: The two data registers are coupled together, and use the same read/write signal, the same output and the same input. In this circuit the two registers are compared bit-by-bit and the result is output to the CU. Each register is just a simple 8-bit register that can be used in the ALU or for the various operations of the CPU.

Display: The display uses a binary-to-decimal converter that turns the 8-bit input it receives into a decimal number and displays it in a 3-digit display. It uses an 8-bit register to store the value it displays.

Keypad: The numeric keypad uses a decimal-to-binary converter to turn the input from the presses of its buttons into an 8-bit binary number. This number is stored in a buffer composed of 4 D latches and fed to the bus when needed.

CU: The CU controls the read write access for most of the registers, as well as sending the appropriate address and operations to other components. Namely, it will tell the PC when to update and whether it should increment or take on a new address for jumping, it will tell the data registers and the ALU when to save or execute, and it will send various other data based on the input from the IR. The CU also holds a counter (top left in CPU.circ for easy viewing), which keeps track of which operation happens when. This allows for the system bus to pass the proper value. The details of each instruction cycle will be outlined below.

3. Instruction Cycle

Each instruction executes on a 16 tick cycle. Since Logisim implements a realistic delay for memory, we essentially use every other tick to ensure the data passes through, before continuing with the next operation. Since the counter starts at 0 to begin with and increments right away, we start our operations when the value is 1 rather than 0.

1. PC sends address to RAM

    tick

3. RAM reads data and sends it to MBR.

    tick

5. Keypad data gets sent to the MBR. Old MBR data sent to instruction.

    tick

7. CPU sends the address (last 4 bits of instruction). This will be used for load/store, but is otherwise useless. The first four bits in the data will hold the following:

    * Is the OP code store
    * Is the OP code print
    * Is the OP code stop
    * Is there an error in the ALU operation

    This data will be passed to the RAM, Display, and the Reset toggle

    tick

9. RAM outputs the data at the address it was previously given. Depending on our instruction, we may choose to save it.

    tick

11. CPU outputs data for RAM to write at its current address. If the bit at tick 7 was write (1), RAM will save it.

    tick

13. CPU outputs data to display. If the second bit at tick 7 was prt (1), the display will show it.

    tick

Concurrently, the following will happen during certain ticks:

9. The PC increments itself
11. We have all our data where we need it to be, so ALU operations may be executed.
13. We have both the instruction and the new data, so it may be saved into registers if we need it.
15. If we need to jump, the address will be overwritten on the PC

4. Language Syntax of instructions

LOAD: The first 3 bits of these instructions are 011. The 4th bit is the address of the register that we want to load the data to. These 4 bits make up the op code. The next 4 bits are the address in RAM to load the data from. The CU will then send that address back, receive the data at that address from the RAM, and store it in the specified register.

STORE: The first 3 bits of these instructions are 100. The 4th bit is the address of the register that we want to take the data from. These 4 bits make up the op code. The next 4 bits are the address in RAM to store the data in. The CU will then send the address to the RAM, as well as a bit priming it to enable write before sending the data from the specified register to write into RAM.

ADD: The first 4 bits of these instructions are 0011, this is its op-code. The next 4 bits do not matter as we are always adding the two registers and then storing the result in R1.

SUB: The first 4 bits of these instructions are 0010, this is its op-code. The next 4 bits do not matter as we are always subtracting R2 from R1 and then storing the result in R1.

BEQ: The first 4 bits of these instructions are 0100, this is its op-code. The next 4 bits are the address in RAM to jump to if the two registers are equal. Otherwise, move on to the next instruction.

BNQ: The first 4 bits of these instructions are 0101, this is its op-code. The next 4 bits are the address in RAM to jump to if the two registers are not equal. Otherwise, move on to the next instruction.

PRINT: The first 3 bits of these instructions are 101. The 4th bit is the address of the register that we want to print the data from. These 4 bits make up the op-code. The next 4 bits do not matter.

INPUT: The first 3 bits of these instructions are 110. The 4th bit is the address of the register that we want to move the keypad input to. These 4 bits make up the op-code. The next 4 bits do not matter.

STOP: The first 4 bits of these instructions are 0000. This is its op-code. The next 4 bits do not matter.

MULT: The first 4 bits of these instructions are 1110, this is its op-code. The next 4 bits do not matter as we are always multiplying the two registers and then storing the result in R1.

Multiplication program

Below is the pseudocode for a program which takes a number stored in memory, and a keyboard input and multiplies the two numbers, showing the output on the display.

```
1.  1101 0000    # load input -> r1
2.  0101 0111    # bne -> #8 (note that r0 = firstCycle ? 0 : input)
3.  0000 0000    # stop
4.  0000 0000    # [data] sum
5.  xxxx xxxx    # [data] val
6.  0000 0000    # [data] counter
7.  0000 0001    # [data] one
8.  0110 0011    # load sum -> r0
9.  0111 0100    # load val -> r1
10. 0011 0000    # add sum + val
11. 1010 0000    # prt sum (r0)
12. 1000 0011    # store sum (r0)
13. 0110 0101    # load counter -> r0
14. 0111 0110    # load one -> r1
15. 0011 0000    # add counter + one
16. 1000 0101    # store counter (r0)
```

The algorithm works by adding the number stored to an accumulator n times, where n is the number inputted using the keypad. For each addition, the counter variable is incremented by one, until it is equal to n when the program displays the final result.

The caveat is that the input may not be 0, otherwise the program will keep adding. We need more than 16 instructions or a jump to add that.