# Week 6 Lab A: Addressable Memory

## Objectives

Develop understanding and experience of:

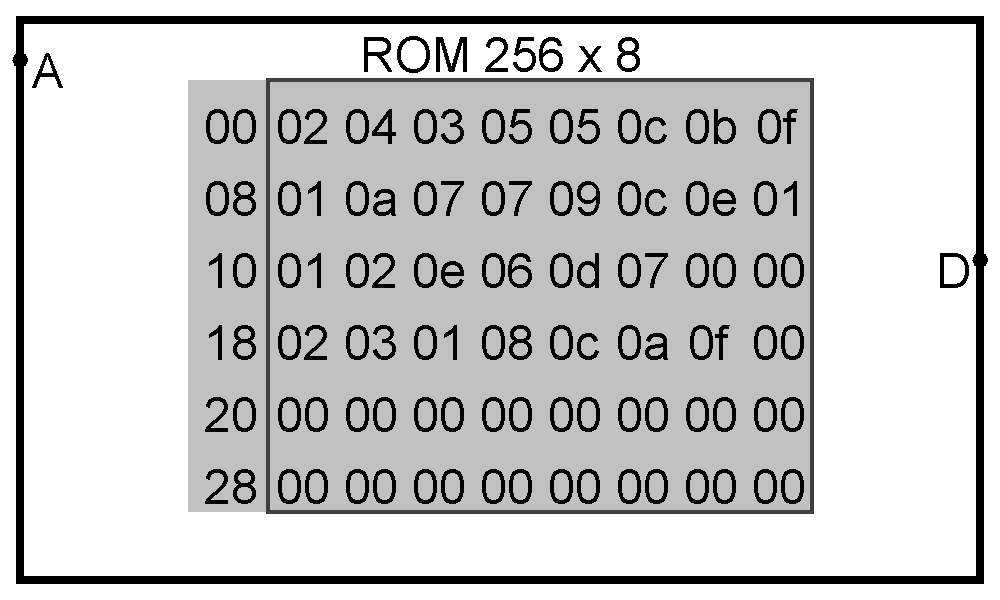
1. Addressable Memory
2. Using memory components in Logisim Evolution
3. Create a simulated automated accumulating adding machine

## Addressable memory

Working with memory addresses is common in many areas of computing.

* What is meant by a memory address?

reference to a specific memory location used at various levels by software and hardware



* What hexadecimal values are **stored** at the following hex addresses in the image above?

Address 05: Value stored = 0c

Address 0f: Value stored = 01

Address 15: Value stored = 07

Address 1a: Value stored = 01

* What addresses (in hex) are the following values stored at?

Value 04: Address = 01

Value 09: Address = 0c

Value 0d: Address = 14

* Why do memory addresses have to be unique?

So, the CPU can locate the correct data.

* Do the values stored have to be unique?

No multiple can have the same value.

* Why does the number of unique memory addresses depend on the number of bits in the address?

The number of bits give many values we can create.

## Building an Accumulating Adding Machine

A screenshot of a computer

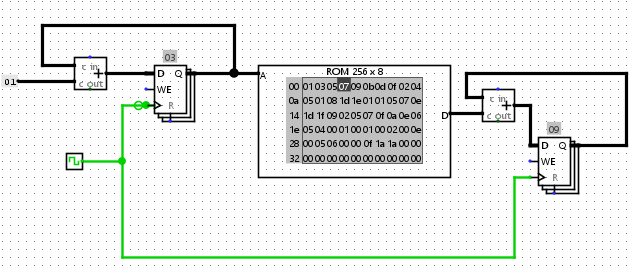
Description automatically generated

1. You should build the circuit above which was also explained in last week’s lecture. All the components are working with 8-bit data. The register on the right-hand side is now the accumulator and is keeping the total of the values in the memory. The register on the left-hand side is a counter (like you built in last week’s lab) and is going through each memory address in turn starting from 00. Note that the memory is a ROM (from the Memory tools in Logisim Evolution) meaning the values stored can’t be changed by other parts of the circuit and so it doesn’t need a clock signal. Set the Appearance property of the ROM to “Classic Logisim” Note that the same clock signal is connected to both registers.
2. To test the circuit, you could use the hand-icon and overtype any of the values you want to change. An alternative way is to load the data from a file. On Moodle in the W6 Lab Starter Code, you will find a file W6\_LA\_data.dat. Download that file and open it in Notepad++ (or another plain text editor). You will see the odd numbers up to f as shown in the memory in the image above.

Work out what you expect the total to be when adding those up and convert to hexadecimal. You may use any suitable online tool to convert to hexadecimal.

Total in decimal 01

Total in hexadecimal 03

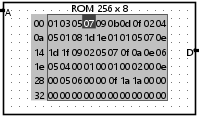


To load data into memory in Logisim Evolution, with the hand icon highlighted, right-click on the memory, click Load Image and navigate to where you have stored the data file.

Run your circuit and add an image showing the final result in the accumulator and check against your expected results. To reset your registers (but not the ROM) use Ctrl+R.

1. Make a copy of W6\_LA\_data.dat and this time add **all** the whole numbers from 1 to 20 (when given in decimal). Keep the first line of the file (v2.0 raw) Each number will have to be separated from the next by a space. Remember that the numbers will have to be in hexadecimal.

Paste your datafile here



Work out the total of the numbers (look online or use a formula if you know one).

Total in decimal

Total in hexadecimal

Run your circuit and add an image showing the final result in the accumulator and check against your expected results.

## Extension exercises

To complete the aim of creating all required components for a simulated computer from logic gates, you should look at building memory components from Registers and other components in Logisim Evolution.

1. Using built-in registers that are available in the Memory tools in Logisim Evolution, create addressable ROM that is made of two 4-bit registers and 1-bit address (use a 1-bit input pin). For this investigation, the registers will not need input or a clock signal as you will type in values manually for testing. Remember that the address will be used to *select* which of the registers to pass the value to the output.

Use a 4-bit output pin to receive the output from the ROM, you will need to change the data bits property of the pin and may want to change the radix to hexadecimal rather than binary. To change the values in your registers, make sure you are in simulation mode (the hand icon) and highlight the number just above the register and change it to any valid hex digit. Test your ROM by making sure that one value is selected for address 0 and the other for address 1.

1. Create a new version of the circuit to have four registers and a 2-bit address, still with 4-bit data. Based on the testing you did above, think about how to test the circuit systematically to check it is working correctly.
2. Compare your ROM to the ROM component built-in to Logisim Evolution as follows.

Create a new circuit in Logisim Evolution and add a ROM component from the memory tools. To match task b above, set the Address Bit Width to 2 and the Data Bit Width to 4. I prefer to set the appearance to “Classic Logisim” as it is less cluttered. Connect a 2-bit input pin to the Address (A) input on the ROM and a 4-bit output pin to the Data (D) output. You might want to change the radix for the output pin.

Test the circuit as follows. When you are in simulation mode (with the hand highlighted) you can click on data at individual addresses to change its value (as you could with the registers). Note that the zero on the left is the *address* of the first item on that line, not the value stored.

1. This task is a little more complex, but now try to create a RAM component in a similar way to the ROM created above, so that you have addressable memory that can be written to and read from.

Start by taking a copy of the ROM that you created that had a 2-bit address, that is it had four registers. You will need a 4-bit input pin to represent the data to be stored and a 1-bit pin as a store signal to indicate that the RAM should be updated. You will now need a clock signal. The ROM that couldn’t be updated didn’t need a clock.

The RAM should be able to store the input from the input pin in the register corresponding to the address input when the store signal is 1. This means that the address will need to be decoded to determine which register to update. The store signal can connect to the Enable input of a decoder and the outputs of the decoder to the write enable (WE) inputs of the relevant registers.

Have a go at creating a RAM and testing it. It should still pass the appropriate output to the output pin according to the address signal as did the ROM.

1. Create a circuit that uses a Logisim Evolution built-in RAM component to achieve the same result. The RAM should have a 2-bit address, a store signal, a clock signal, a 4-bit data input and a 4-bit data output. To adjust the RAM to match the one you created in task b and the use of the RAM component in the coursework, set the appearance to be “Classic Logisim”, the Data Interface attribute of the RAM to be “Separate load and store ports” and set “Asynchronous read” to “Yes”. Test the circuit to compare to that created above.