College of Computer and Information Sciences
Department of Computer Science

CSC 220: Computer Organization

Unit 11
Basic Computer Organization and Design

- For the rest of the semester, we'll focus on computer architecture: how to assemble the combinational and sequential components we've studied so far into a complete computer.
- Today, we'll start with the datapath, the part of the central processing unit (CPU) that does the actual computations.

An overview of CPU design

- We can divide the design of our CPU into three parts:
  - The datapath does all of the actual data processing.
  - An instruction set is the programmer's interface to CPU.
  - A control unit uses the programmer's instructions to tell the datapath what to do.

- Today we'll look in detail at a processor's datapath.

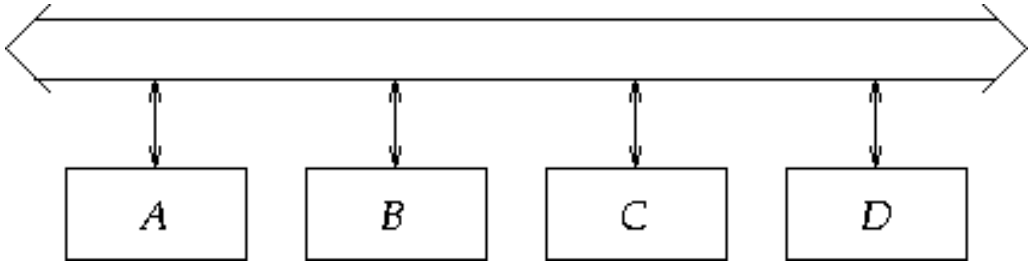
- An ALU does computations, as we've seen before.
  - A limited set of registers serves as fast temporary storage.
  - A larger, but slower, random-access memory is also available.

What's in a CPU?

- A processor is just one big sequential circuit.
  - Some registers are used to store values, which form the state.
  - An ALU performs various operations on the data stored in the registers.

Micro-Ops Transfer Bus

- A bus consists of a set of parallel data lines
- To transfer data using a bus: connect the output of the source register to the bus; connect the input of the target register to the bus; when the clock pulse arrives, the transfer occurs



Bus and Memory transfers

Figure 4-3 Bus system for four registers.

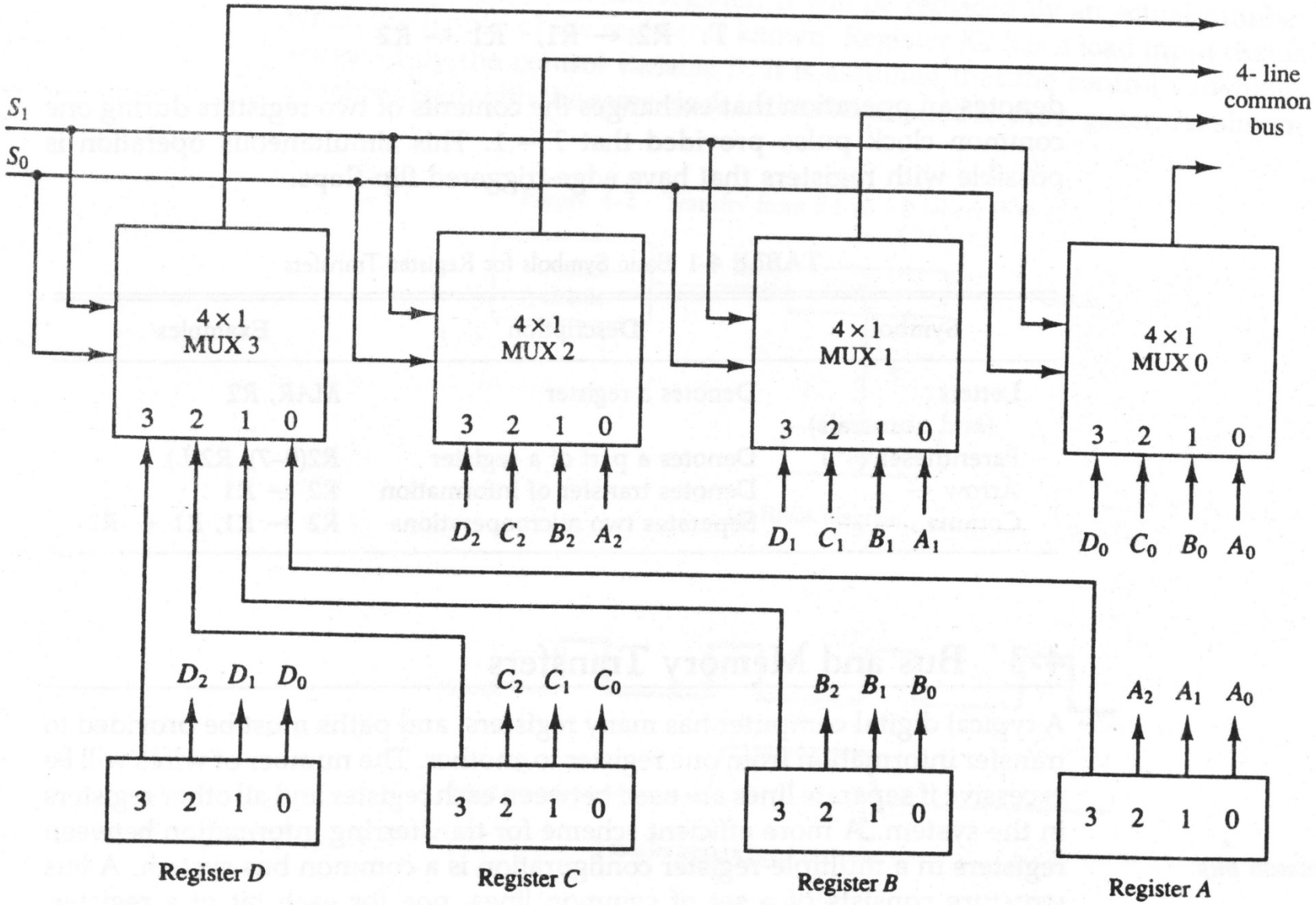


TABLE 4-2 Function Table for Bus of Fig. 4-3

S_1	S_0	Register selected
0	0	*A*
0	1	*B*
1	0	*C*
1	1	*D*

$BUS \leftarrow C, \qquad A \leftarrow BUS$

$A \leftarrow C$

The content of register C is placed on the bus, and the content of the bus is loaded into register A by activating its load control input.

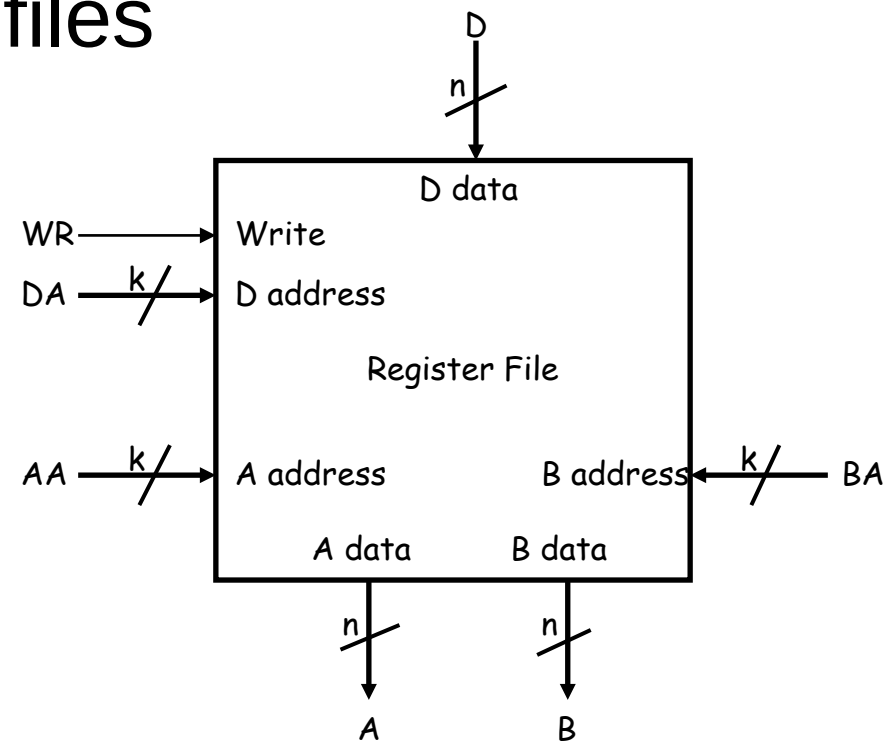
Question

A digital computer has a common bus system for 16 registers of 32 bit each. The bus is constructed with multiplexers.

- How many selection inputs are there is each multiplexer?
- What size of multiplexers is needed?
- How many multiplexers are there in the bus?

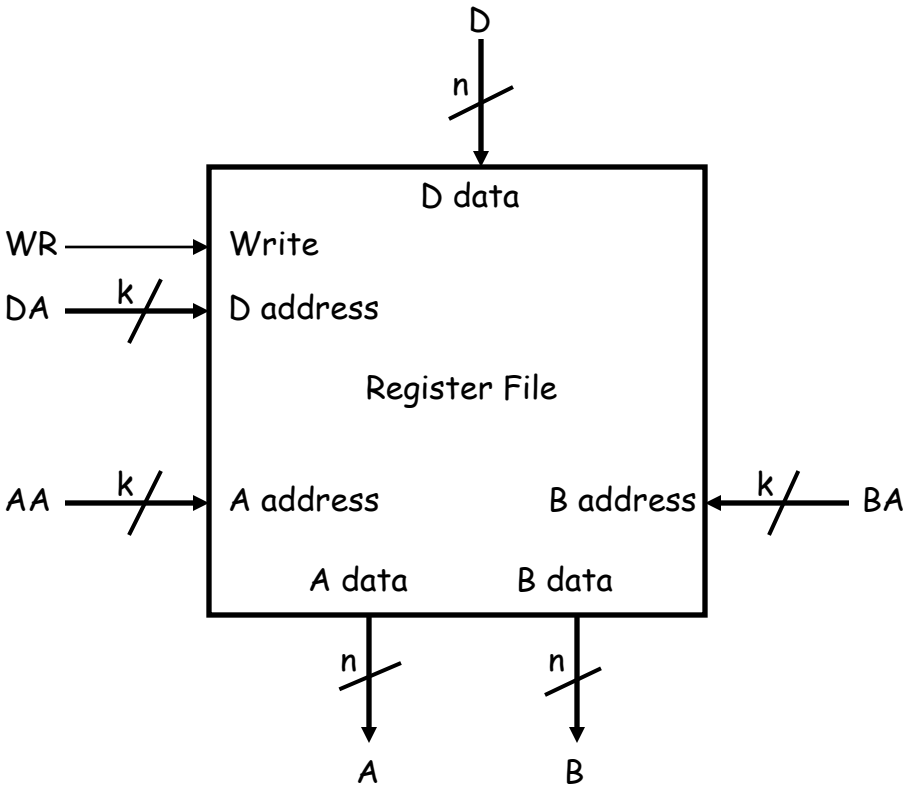
Register files

- Modern processors contain a number of registers grouped together in a register file.
- Much like words stored in a RAM, individual registers are identified by an address.
- Here is a block symbol for a 2^k x n register file.
  - There are 2^k registers, so register addresses are k bits long.
  - Each register holds an n-bit word, so the data inputs and outputs are n bits wide.



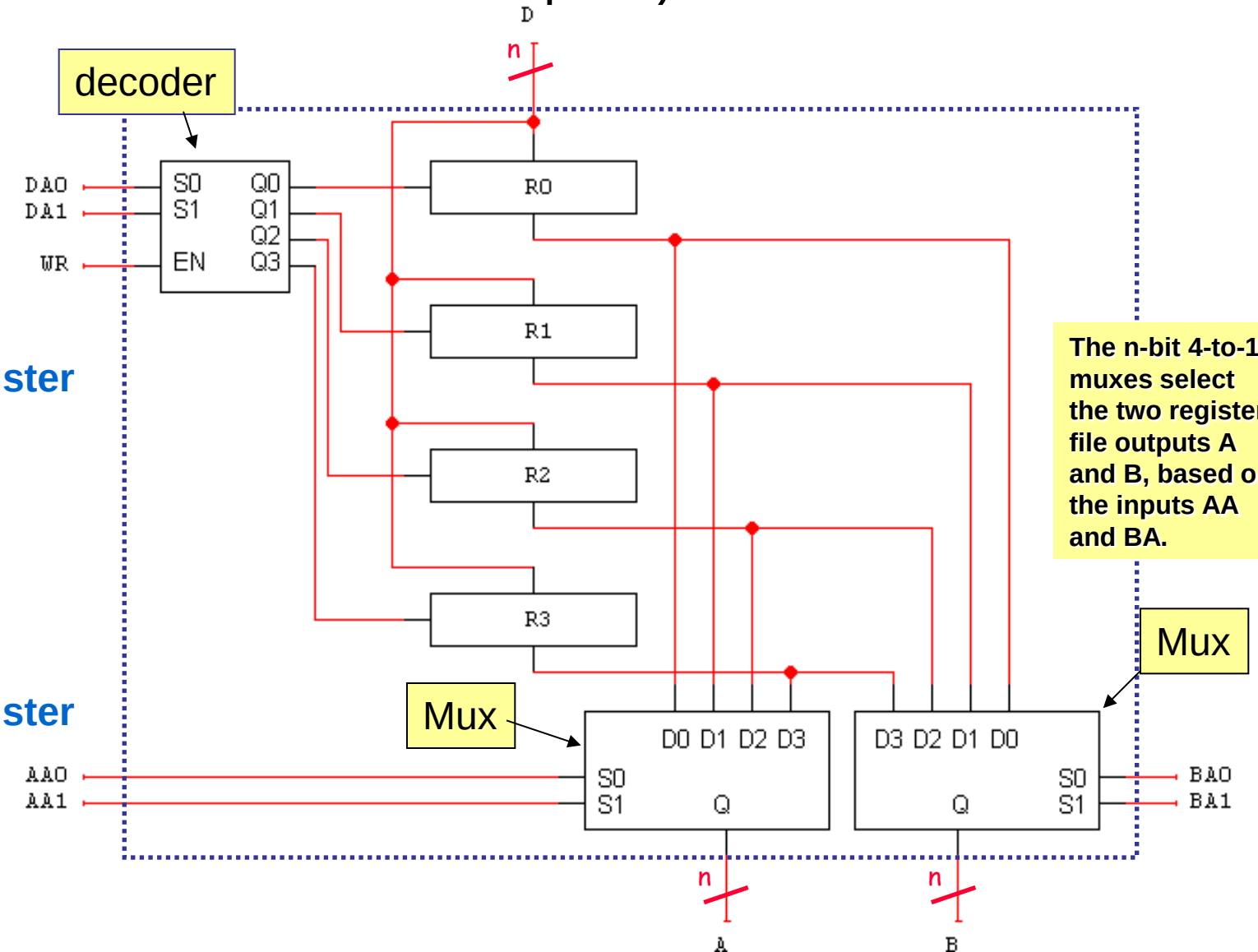
Accessing the register file

- You can read *two* registers at once by supplying the AA and BA inputs. The data appears on the A and B outputs.
- You can write to a register by using the DA and D inputs, and setting WR = 1.
- These are registers so there must be a clock signal, even though we usually don't show it in diagrams.
  - We can read from the register file at any time.
  - Data is written only on the positive edge of the clock.



What's inside the register file

- Here's a 4 x n register file. (We'll assume a 2^k x n = **4 x n** register file for all our examples.)



DA0 & DA1
Select the register to write in

AA0 & AA1
BA0 & BA1
Select the register to read from

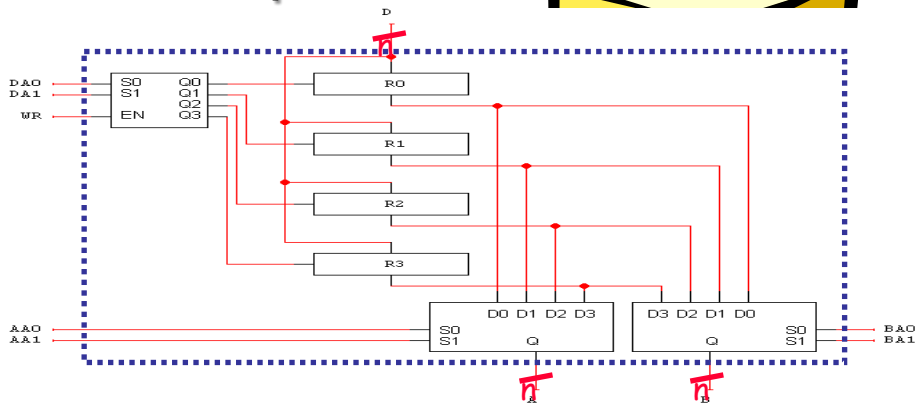
The n-bit 4-to-1 muxes select the two register file outputs A and B, based on the inputs AA and BA.

Explaining the register file

- The 2-to-4 decoder selects one of the four registers for writing. If WR = 1, the decoder will be enabled and one of the Load signals will be active.

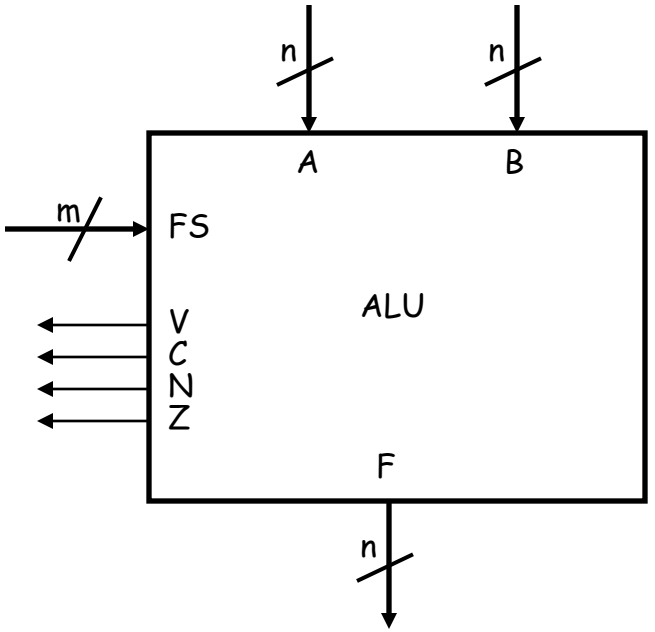
- The n-bit 4-to-1 muxes select the two register file outputs A and B, based on the inputs AA and BA.

- We need to be able to read two registers at once because most arithmetic operations require two operands.



The all-important ALU

- The main job of a central processing unit is to “process,” or to perform computations....remember the ALU from way back when?
- We’ll use the following general block symbol for the ALU.
  - A and B are two n-bit numeric inputs.
  - FS is an m-bit function select code, which picks one of 2^m functions.
  - The n-bit result is called F.
  - Several status bits provide more information about the output F:
    - V = 1 in case of signed overflow.
    - C is the carry out.
    - N = 1 if the result is negative.
    - Z = 1 if the result is 0.



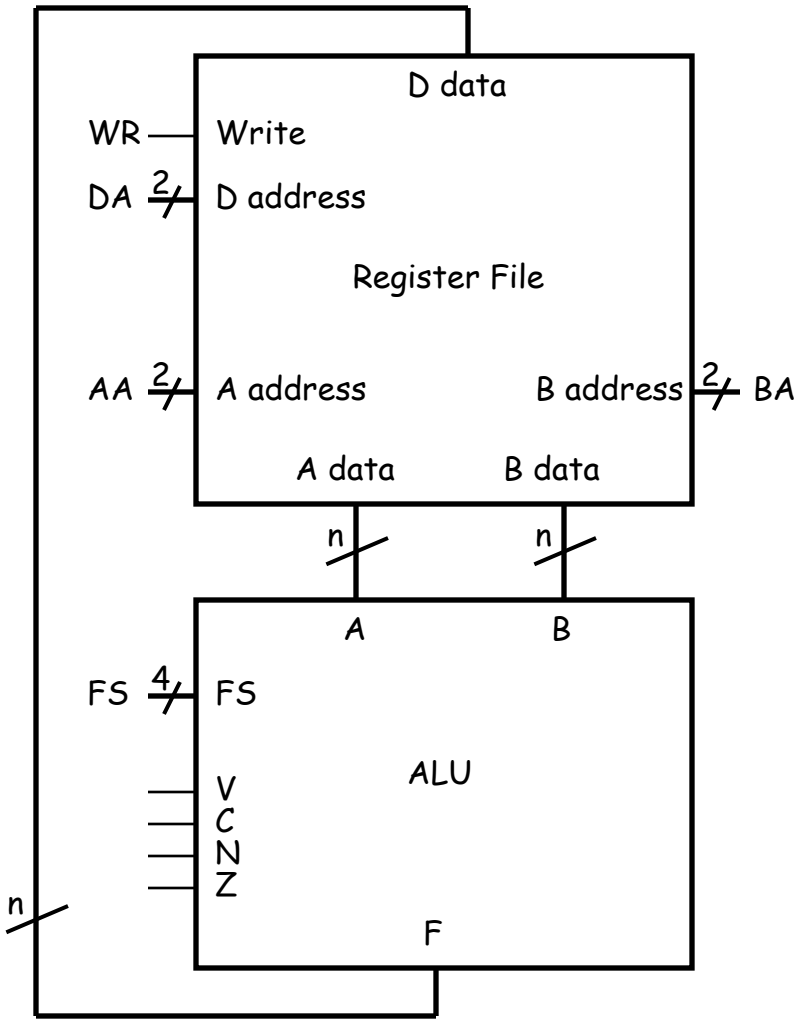
ALU functions

- For concrete examples, we'll use the ALU as it's presented in chapter 4.
- The function select code FS is 4 bits long, but there are only 15 different functions here.
- We use an alternative notation for AND and OR to avoid confusion with arithmetic operations.

FS	Operation
0000	F = A
0001	F = A + 1
0010	F = A + B
0011	F = A + B + 1
0100	F = A + B'
0101	F = A + B' + 1
0110	F = A - 1
0111	F = A
1000	F = A $\wedge$ B (AND)
1001	F = A $\vee$ B (OR)
1010	F = A $\oplus$ B (XOR)
1011	F = A'
1100	F = B
1101	F = sr B (shift right)
1110	F = sl B (shift left)

My first datapath

- Here is the most basic datapath.
  - The ALU's two data inputs come from the register file.
  - The ALU computes a result, which is saved back to the registers.



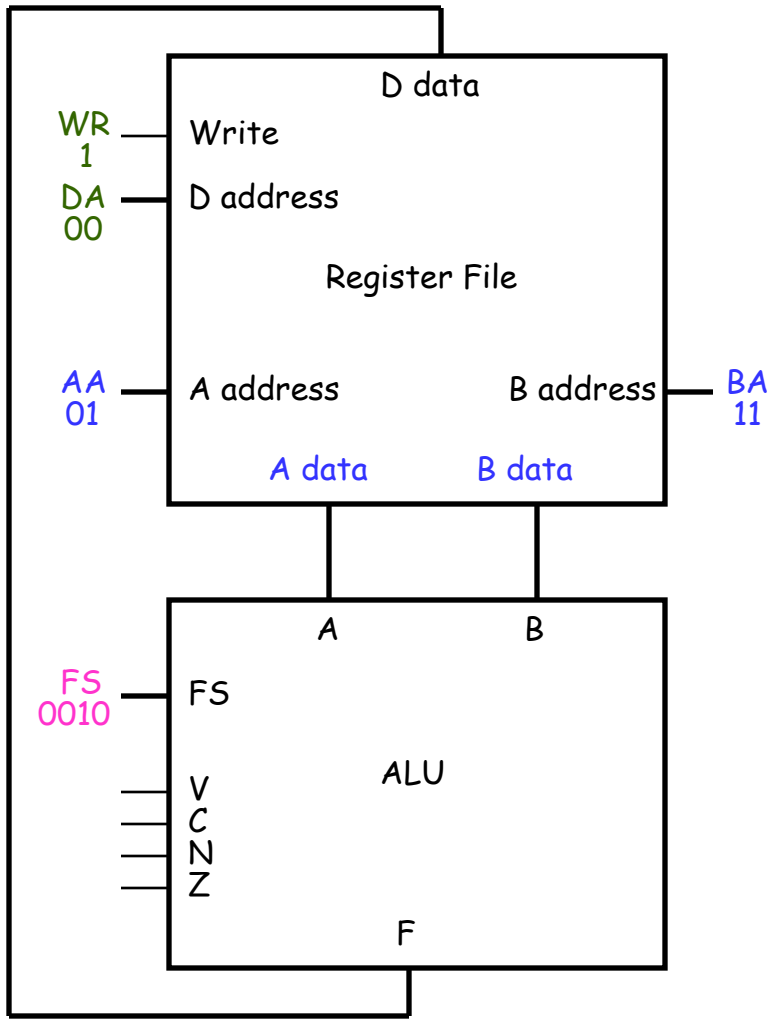
- WR, DA, AA, BA and FS are control signals. Their values determine the exact actions taken by the datapath— which registers are used and for what operation.

An example computation

- Let's look at the proper control signals for the operation below:

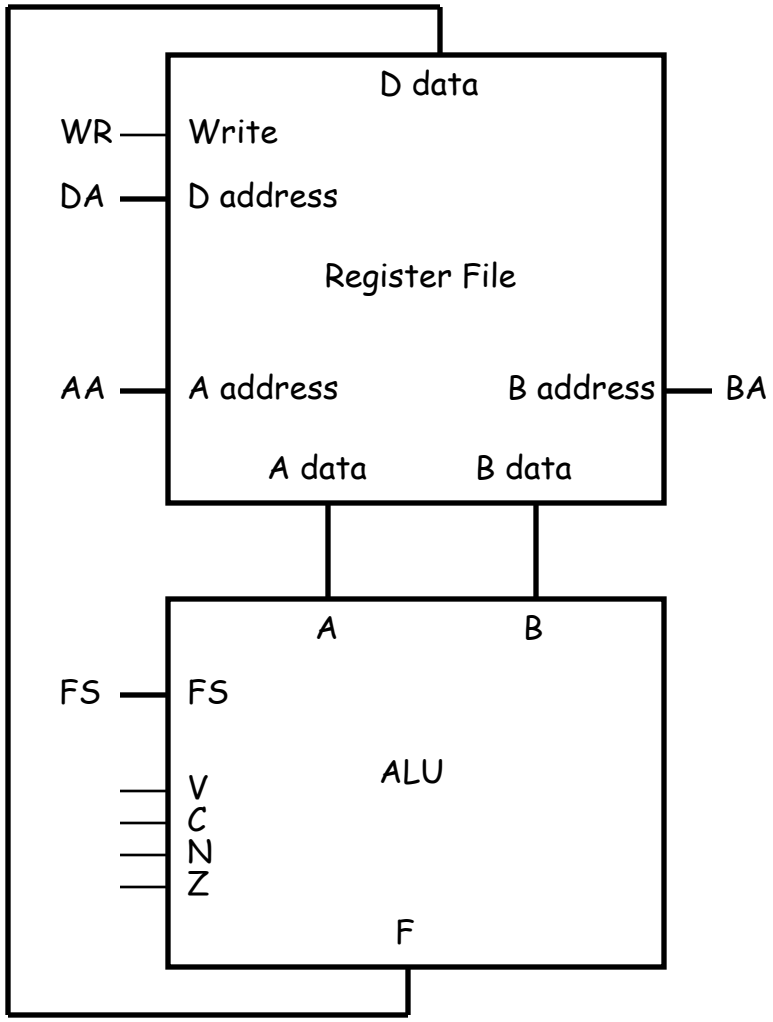
$$R0 \leftarrow R1 + R3$$

- Set AA = 01 and BA = 11. This causes the contents of R1 to appear at A data, and the contents of R3 to appear at B data.
- Set the ALU's function select input FS = 0010 (A + B).
- Set DA = 00 and WR = 1. On the next positive clock edge, the ALU result (R1 + R3) will be stored in R0.



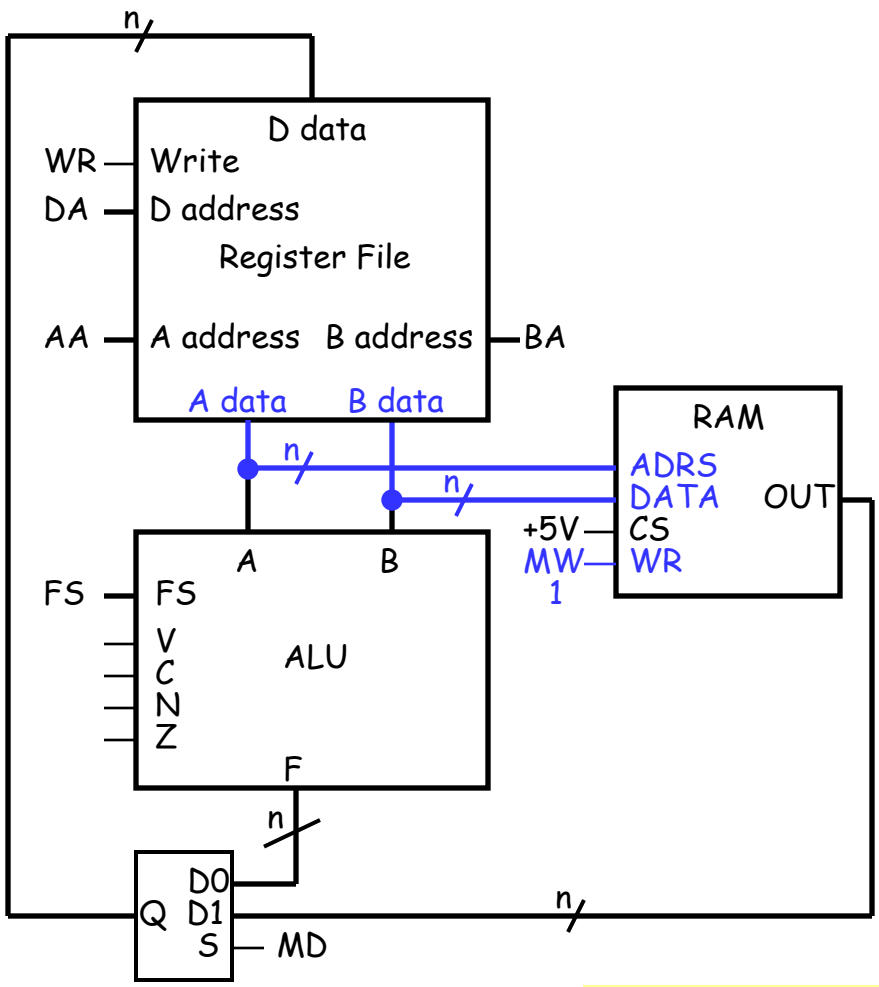
Two questions

- Four registers isn't a lot. What if we need more storage?
- Who exactly decides which registers are read and written and which ALU function is executed?



We can access RAM also

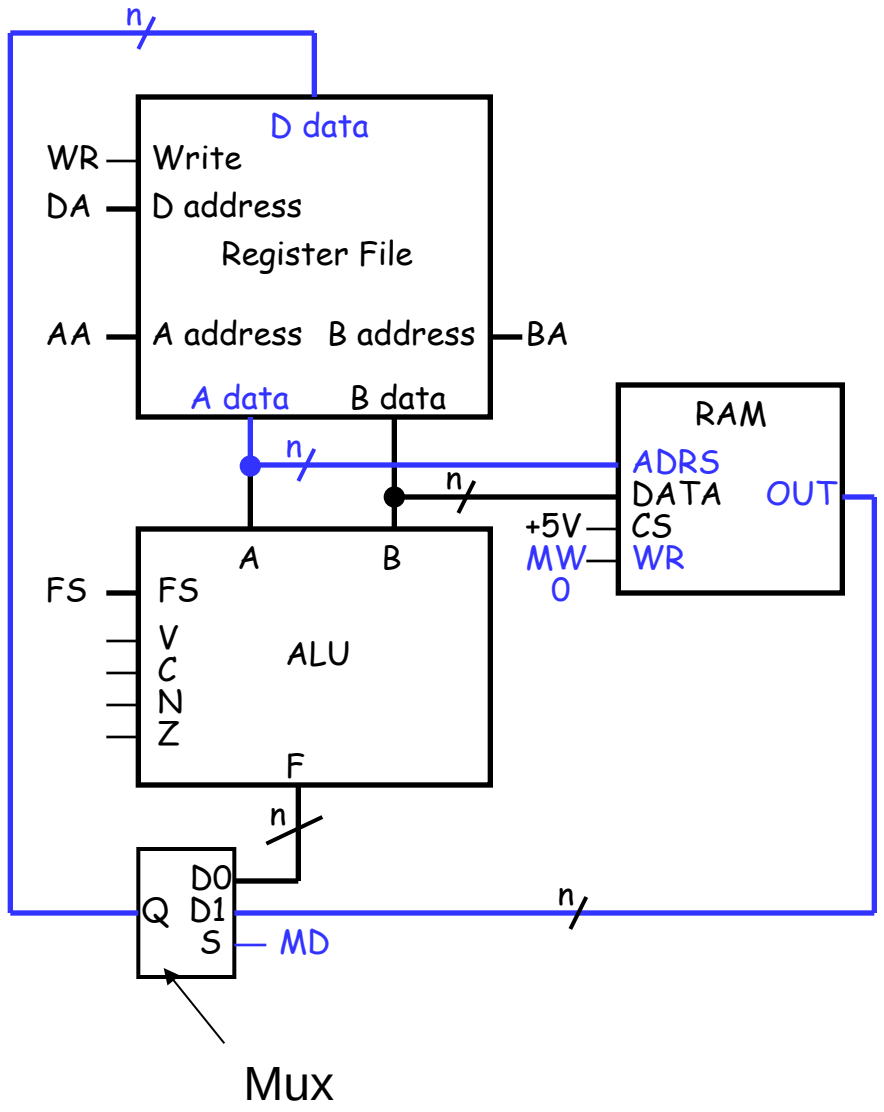
- Here's a way to connect RAM into our existing datapath.
- To *write* to RAM, we must give an address and a data value.
- These will come from the registers. We connect A data to the memory's ADRS input, and B data to the memory's DATA input.
- Set MW = 1 to write to the RAM. (It's called MW to distinguish it from the WR write signal on the register file.)



MW = 1
to write to the RAM

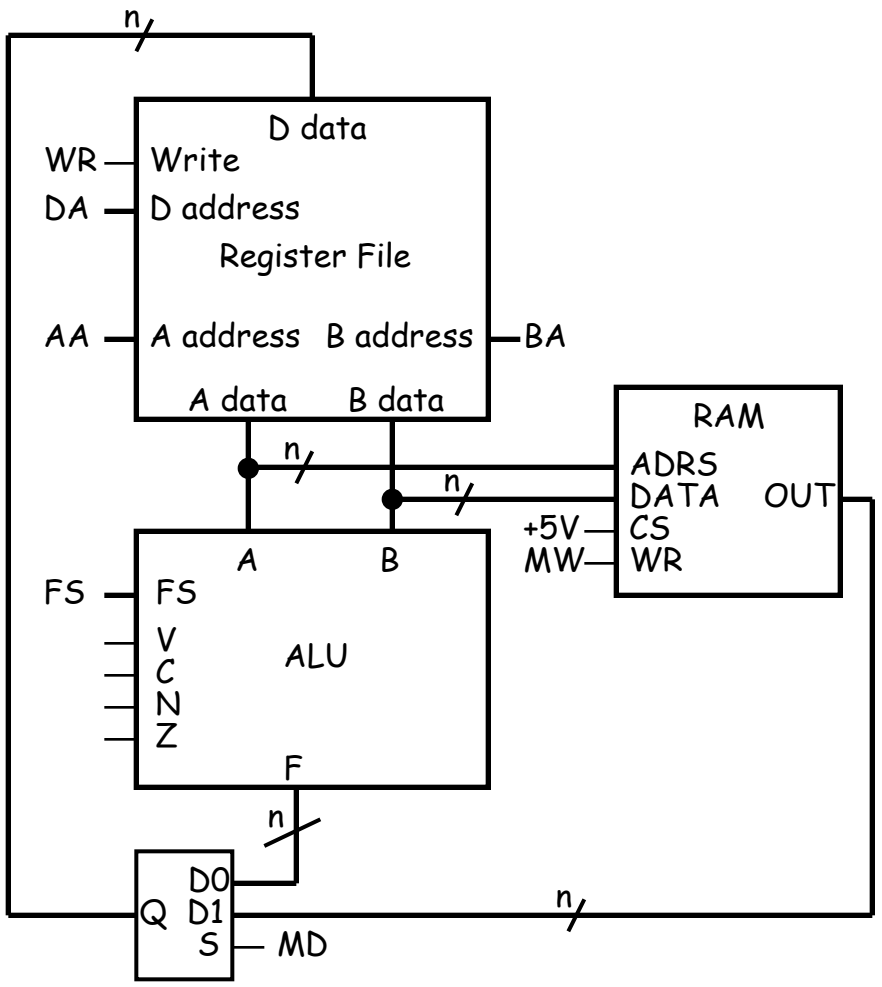
Reading from RAM

- To *read* from RAM, A data must supply the address.
- Set MW = 0 for reading.
- The incoming data will be sent to the register file for storage.
- This means that the register file's D data input could come from *either* the ALU output or the RAM.
- A mux MD selects the source for the register file.
  - When MD = 0, the ALU output can be stored in the register file.
  - When MD = 1, the RAM output is sent to the register file instead.



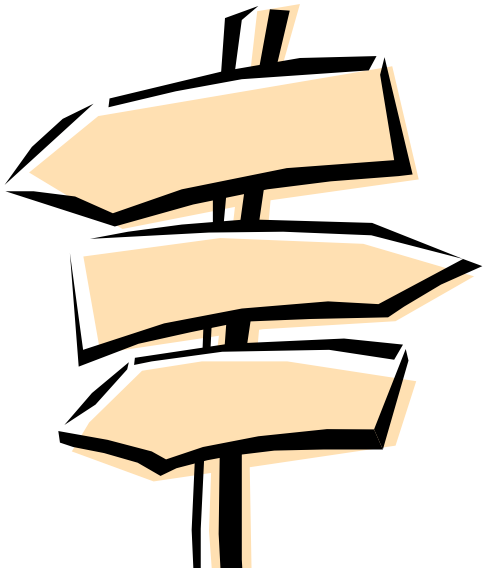
Notes about this setup

- We now have a way to copy data between our register file and the RAM.
- Notice that there's no way for the ALU to directly access the memory—RAM contents *must* go through the register file first.
- Here the size of the memory is limited by the size of the registers; with n-bit registers, we can only use a 2^n x n RAM.



Memory transfer notation

- In our transfer language, the contents at random access memory address X are denoted M[X]. For example:
  - The first word in RAM is M[0].
  - If register R1 contains an address, then M[R1] are the contents of that address.
- The M[ ] notation is like a pointer dereference operation in C or C++.



Example sequence of operations

- Here is a simple series of register transfer instructions:

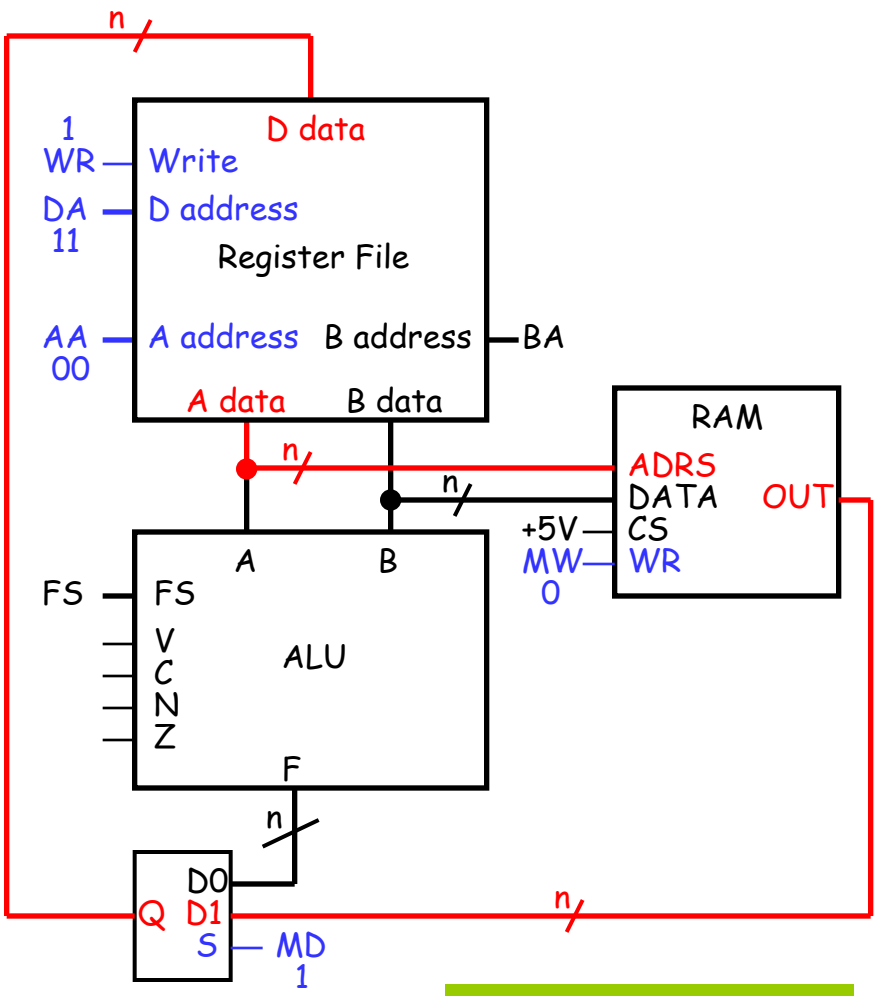
M[R0] ← M[R0] + 1 =

$$R3 \leftarrow M[R0]$$
$$R3 \leftarrow R3 + 1$$
$$M[R0] \leftarrow R3$$

- This just increments the contents at address R0 in RAM.
  - Again, our ALU only operates on registers, so the RAM contents must first be loaded into a register, and then saved back to RAM.
  - R0 is the first register in our register file. We'll assume it contains a valid memory address.
- How would these instructions execute in our datapath?

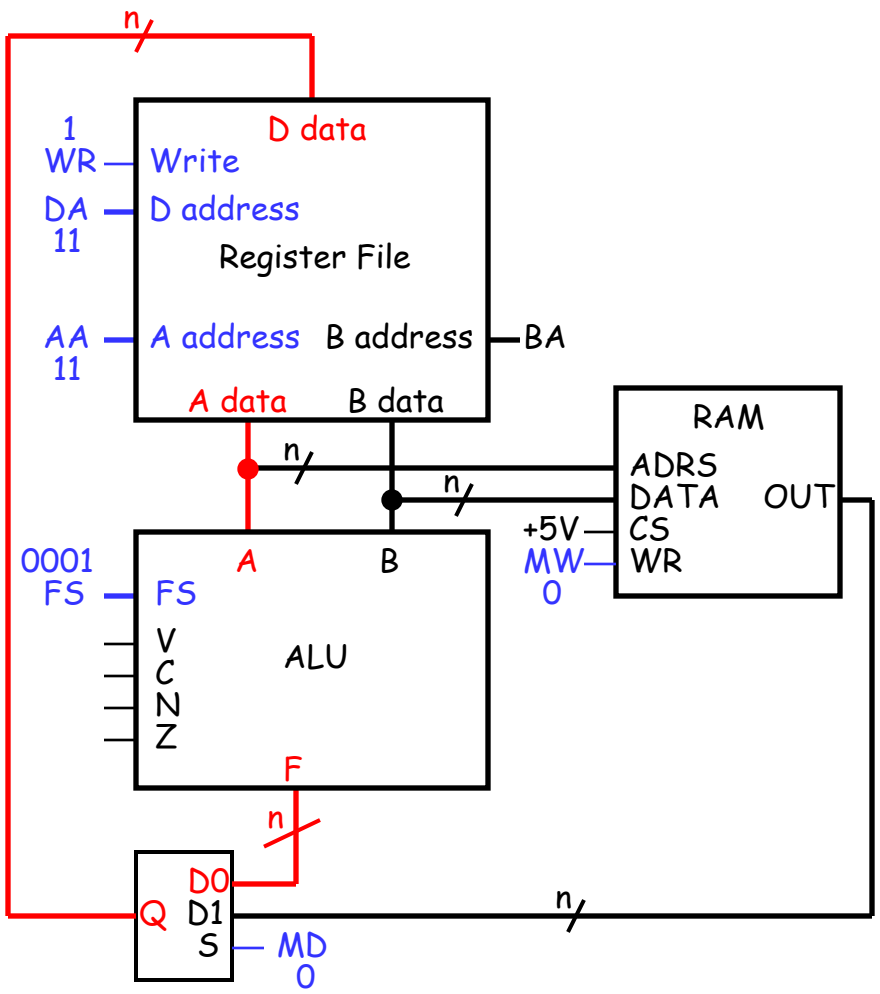
R3 ← M[R0]

- AA should be set to 00, to read register R0.
- The value in R0 will be sent to the RAM address input, so M[R0] appears as the RAM output OUT.
- MD must be 1, so the RAM output goes to the register file.
- To store something into R3, we'll need to set DA = 11 and WR = 1.
- MW should be 0, so nothing is accidentally changed in RAM.
- Here, we did not use the ALU (FS) or the second register file output (BA).



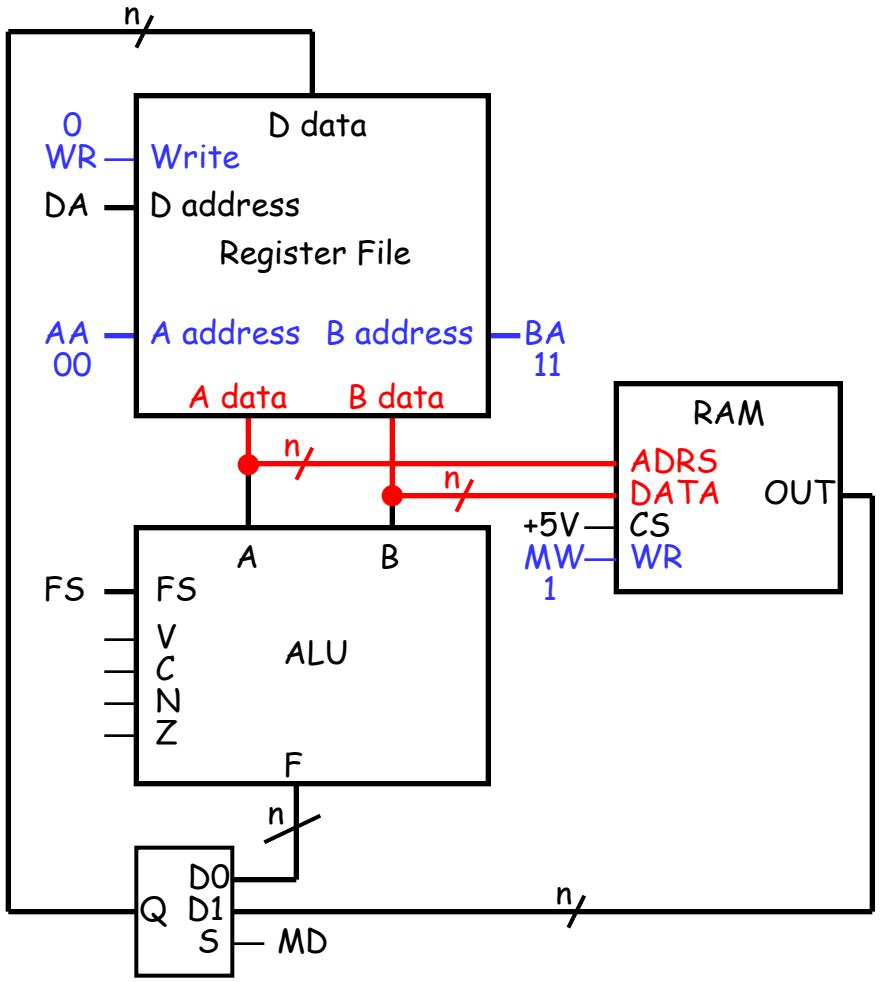
R3 ← R3 + 1

- AA = 11, so R3 is read from the register file and sent to the ALU's A input.
- FS needs to be 0001 for the operation A + 1. Then, R3 + 1 appears as the ALU output F.
- If MD is set to 0, this output will go back to the register file.
- To write to R3, we need to make DA = 11 and WR = 1.
- Again, MW should be 0 so the RAM isn't inadvertently changed.
- We didn't use BA.



M[R0] ← R3

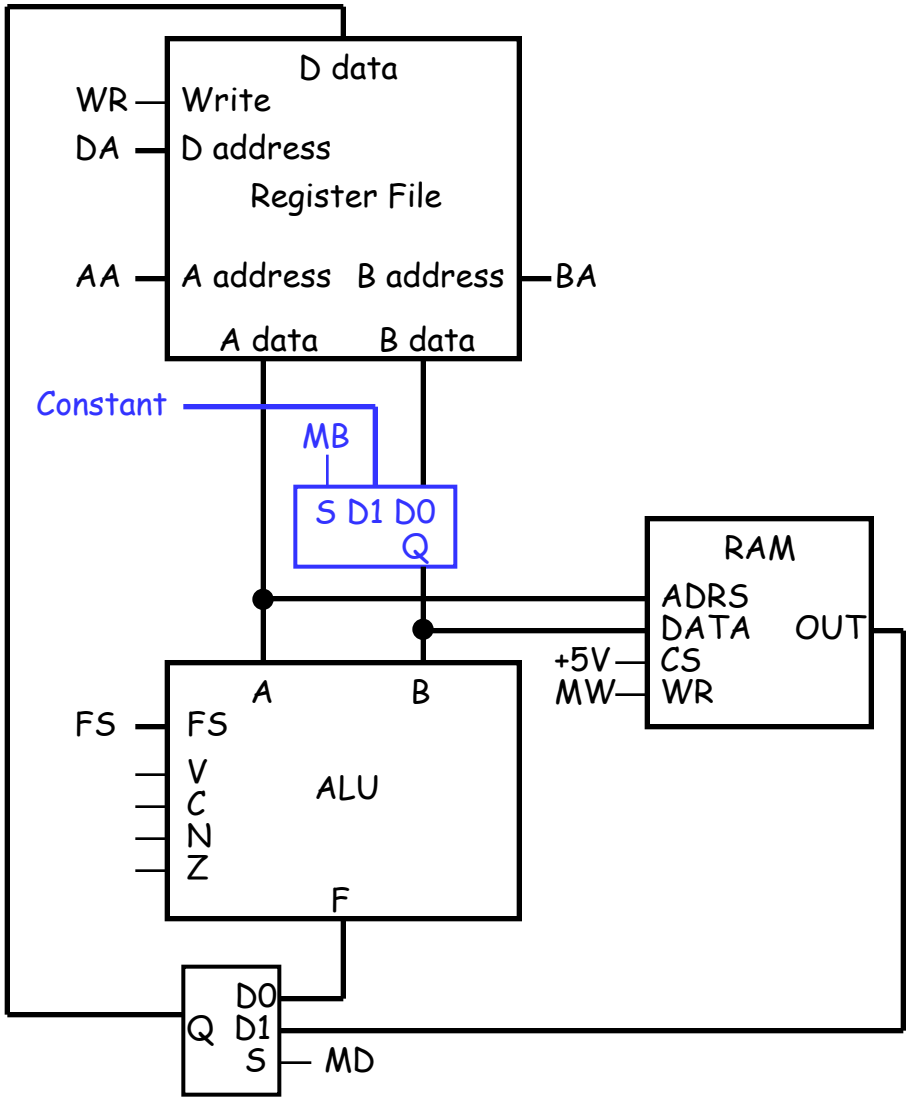
- Finally, we want to store the contents of R3 into RAM address R0.
- Remember the RAM address comes from “A data,” and the contents come from “B data.”
- So we have to set AA = 00 and BA = 11. This sends R0 to ADRS, and R3 to DATA.
- MW must be 1 to write to memory.
- No register updates are needed, so WR should be 0, and MD and DA are unused.
- We also didn’t use the ALU, so FS was ignored.



Constant in

- One last refinement is the addition of a Constant input.
- The modified datapath is shown on the right, with one extra control signal MB.

```
NOT R0, R1    R0 ← R1'
ADD R3, R3, #1  R3 ← R3 + 1
SUB R1, R2, #5 R1 ← R2 - 5
```



Summary

- The datapath is the part of a processor where computation is done.
  - The basic components are an ALU, a register file and some RAM.
  - The ALU does all of the computations, while the register file and RAM provide storage for the ALU's operands and results.
- Various control signals in the datapath govern its behavior.
- Next week, we'll see how programmers can give commands to the processor, and how those commands are translated in control signals for the datapath.