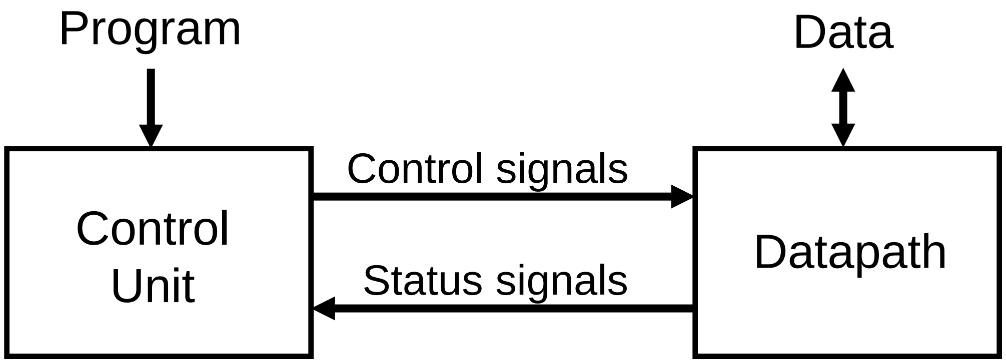
Processor Design Basics: Datapath

Overview

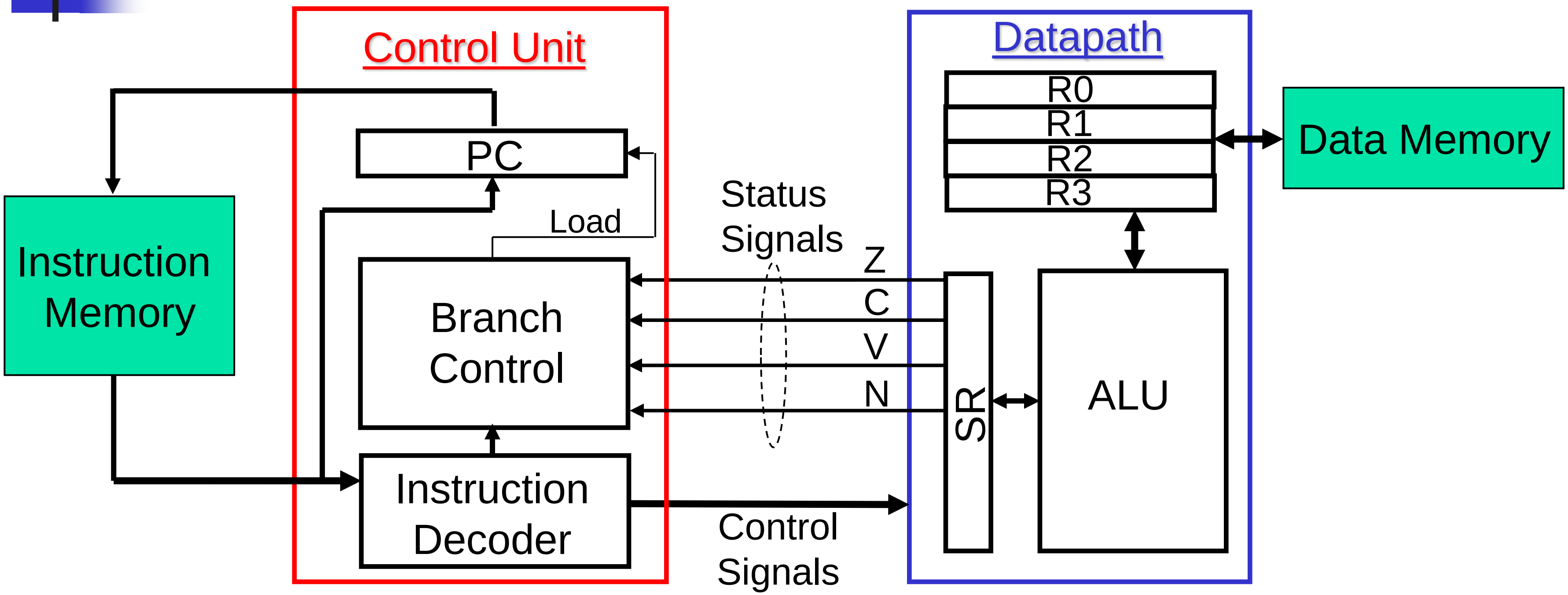
- Block Diagram of a Generic Processor
- Example of a Simple Processor
- Introduction to Datapath
- Register File
  - Accessing the Register File
  - Register File for Our Simple Processor
- Arithmetic and Logic Unit (ALU)
  - ALU for our Simple Processor
  - ALU Operation
- Initial Datapath for Our Processor
- Refinement of the Initial Datapath
  - To allow access to the RAM Data Memory
  - To allow load/store of Constant Values
- Summary

Block Diagram of a Generic Processor



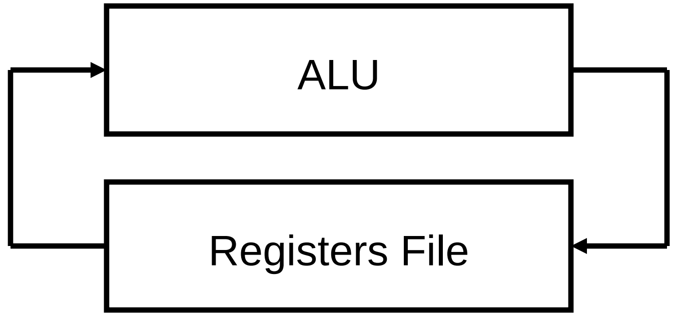
- We can divide the design of a processor into three parts:
  - An Instruction Set is the programmer's interface to the processor.
  - The Datapath does all of the actual data processing.
  - A Control unit uses the programmer's instructions to tell the datapath what to do.
- In this lecture we will discuss the design of a Datapath.

Example of a Simple Processor



- This processor and its Instruction Set Architecture have been discussed in Lecture 13.
- Here we will look in detail at the processor's datapath, which is responsible for doing all of the "dirty" work.
  - An ALU does arithmetic, logic, and shift operations.
  - A limited set of registers serves as fast temporary storage.
  - A larger, but slower, random-access memory is also available.

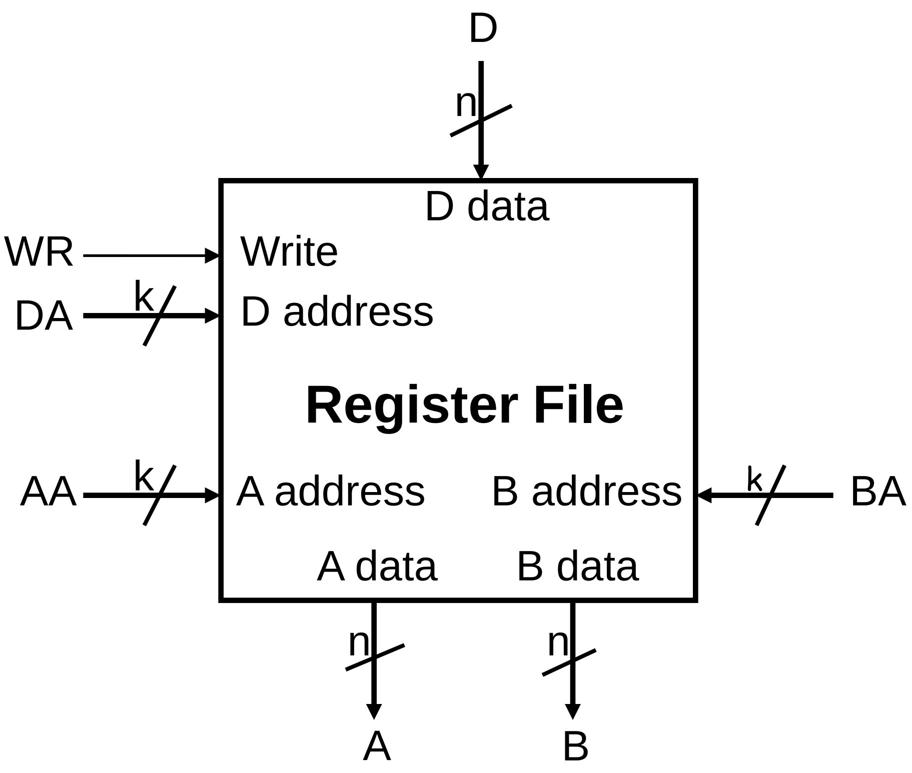
Datapath



- We can look at the datapath as a sequential circuit.
  - Registers are used to store values, which form the state.
  - ALU performs various operations on the data stored in the registers.
- Fundamentally, the processor is just transferring data between the registers using the datapath possibly with some ALU computations.
- ALU is used to perform arithmetic, logic, and shift operations on the data while the data is being transferred.

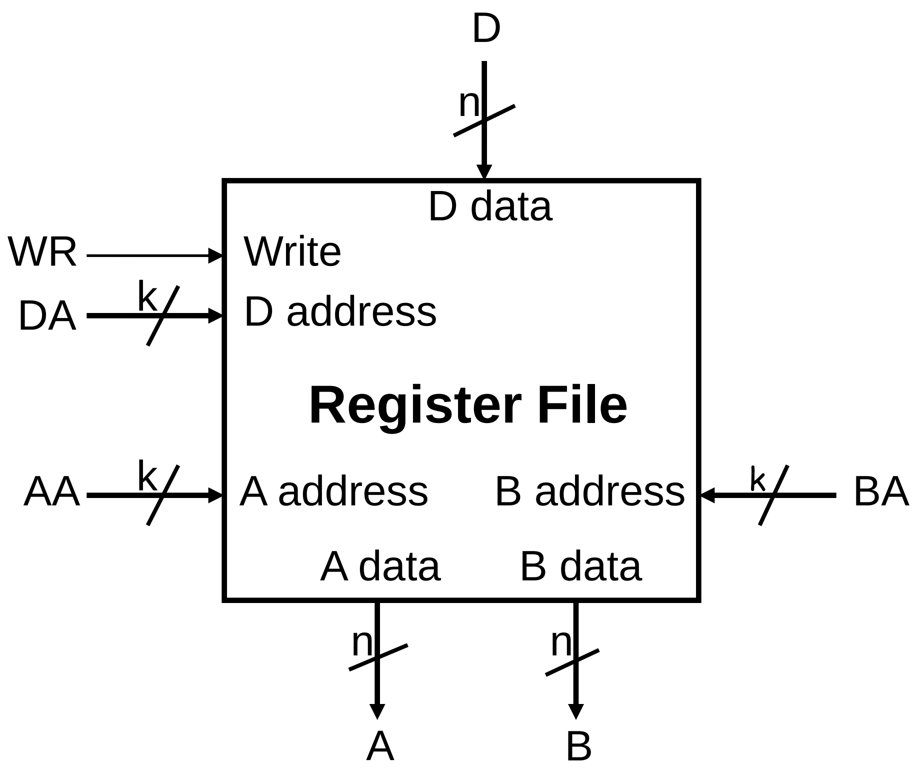
Register File

- Modern processors have a datapath with a number of registers grouped together in a register file.
- Individual registers are identified by an address
  - Much like words stored in a RAM
- 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.



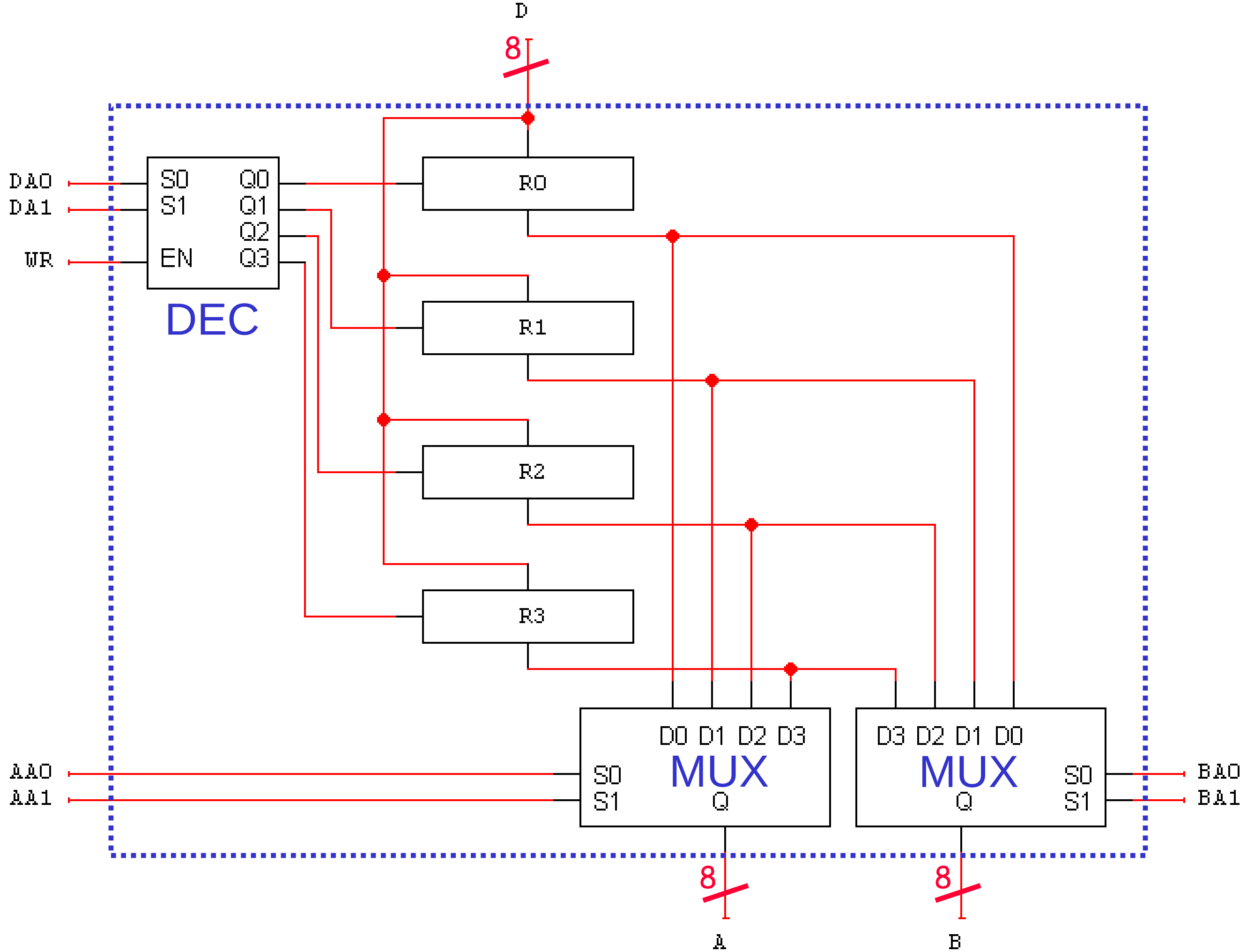
Accessing the Register File

- 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*** and ***reset*** signals, even though we usually do not 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.



Register File for our Processor

- We have to design a 4 x 8 register file because the Instruction Set Architecture of our processor specifies four 8-bit registers (R0 to R3).



Explaining Our Register File

- The 2-to-4 decoder DEC selects one of the four registers for writing using the inputs DA0 and DA1. If WR = 1, the decoder will be enabled and one of the Load signals will be active.
- The 8-bit 4-to-1 multiplexers MUXs select two registers from the file and connects them to outputs A and B, based on the inputs AA0,AA1 and BA0,BA1.
- We need to be able to read two registers at once because most of the instructions of our processor require two registers. See the next slide.

Recall the Instructions of Our Processor

Instruction Type	Operation	Mnemonic	Operation	Status Bits	Description
Data Manipulation Instructions	**Register-format Arithmetic & Logic Operations**	LDR Rj, Ri	Rj ← Ri	Z, N	
		INC Rj, Ri	Rj ← Ri + 1	Z, N	
		DEC Rj, Ri	Rj ← Ri - 1	Z, N	
		ADD Rj, Ri	Rj ← Rj + Ri	C, V, Z, N	
		ADDC Rj, Ri	Rj ← Rj + Ri + C	C, V, Z, N	
		SUB Rj, Ri	Rj ← Rj + Ri' + 1	C, V, Z, N	
		AND Rj, Ri	Rj ← Rj ∧ Ri	Z, N	
		OR Rj, Ri	Rj ← Rj ∨ Ri	Z, N	
		XOR Rj,Ri	Rj ← Rj ⊕ Ri	Z, N	
		NOT Rj, Ri	Rj ← Ri'	Z, N	
	Register-format Shift Operations	SHL Rj, Ri	Rj ← Ri **<<** 1	NO effect	
		SHR Rj, Ri	Rj ← Ri **>>** 1	NO effect	
Data Movement Instructions	**Memory write** (from registers)	ST (Rj), Ri	Mem[R0|Rj] ← Ri	NO effect	
	Memory read (to registers)	LD Rj, (Ri)	Rj← Mem[R0|Ri]	NO effect	
	Immediate transfer operations	LDI Rj, #const8	Rj ← const8	NO effect	
		STI (Rj), #const8	Mem[R0|Rj] ← const8	NO effect	
Control Flow Instructions	**Branches**	BZ #offset11	PC ← PC + offset11	NO effect	
		BNZ #offset11	PC ← PC + offset11	NO effect	
		BC #offset11	PC ← PC + offset11	NO effect	
		BNC #offset11	PC ← PC + offset11	NO effect	
		BV #offset11	PC ← PC + offset11	NO effect	
		BNV #offset11	PC ← PC + offset11	NO effect	
		BN #offset11	PC ← PC + offset11	NO effect	
		BNN #offset11	PC ← PC + offset11	NO effect	
	Jump	JMP Rj, Ri	PC ← Rj|Ri	NO effect	

Arithmetic & Logic Unit (ALU)

- The ALU has to perform all arithmetic, logic, and shift operations specified by the Instruction Set Architecture of a processor.
- To design the ALU of our simple processor we have to analyze the relevant part of the instruction set.

Instruction Type	Operation	Mnemonic	Operation	Status Bits	Description
Data Manipulation Instructions	Register-format Arithmetic & Logic Operations	LDR Rj, Ri	Rj ← Ri	Z, N	
		INC Rj, Ri	Rj ← Ri + 1	Z, N	
		DEC Rj, Ri	Rj ← Ri - 1	Z, N	
		ADD Rj, Ri	Rj ← Rj + Ri	C, V, Z, N	
		ADDC Rj, Ri	Rj ← Rj + Ri + C	C, V, Z, N	
		SUB Rj, Ri	Rj ← Rj + Ri' + 1	C, V, Z, N	
		AND Rj, Ri	Rj ← Rj ∧ Ri	Z, N	
		OR Rj, Ri	Rj ← Rj ∨ Ri	Z, N	
		XOR Rj,Ri	Rj ← Rj ⊕ Ri	Z, N	
		NOT Rj, Ri	Rj ← Ri'	Z, N	
	Register-format Shift Operations	SHL Rj, Ri	Rj ← Ri << 1	NO effect	
		SHR Rj, Ri	Rj ← Ri >> 1	NO effect	

- <u>Conclusion1:</u> The ALU must perform 12 operations therefore
  - we need at least 4 control inputs to select one of the 12 operations.

Arithmetic & Logic Unit (ALU)

Instruction Type	Operation	Mnemonic	Operation	Status Bits	Description
Data Manipulation Instructions	**Register-format Arithmetic & Logic Operations**	LDR Rj, Ri	Rj ← Ri	Z, N	
		INC Rj, Ri	Rj ← Ri + 1	Z, N	
		DEC Rj, Ri	Rj ← Ri - 1	Z, N	
		ADD Rj, Ri	Rj ← Rj + Ri	C, V, Z, N	
		ADDC Rj, Ri	Rj ← Rj + Ri + C	C, V, Z, N	
		SUB Rj, Ri	Rj ← Rj + Ri' + 1	C, V, Z, N	
		AND Rj, Ri	Rj ← Rj $\wedge$ Ri	Z, N	
		OR Rj, Ri	Rj ← Rj $\vee$ Ri	Z, N	
		XOR Rj,Ri	Rj ← Rj $\oplus$ Ri	Z, N	
		NOT Rj, Ri	Rj ← Ri'	Z, N	
	Register-format Shift Operations	SHL Rj, Ri	Rj ← Ri **<<** 1	NO effect	
		SHR Rj, Ri	Rj ← Ri **>>** 1	NO effect	

- Conclusion2: The operations require 1 or 2 operands therefore
  - The ALU must have 2 data inputs.
  - The operands are 8-bit binary numbers.

Arithmetic & Logic Unit (ALU)

Instruction Type	Operation	Mnemonic	Operation	Status Bits	Description
Data Manipulation Instructions	**Register-format Arithmetic & Logic Operations**	LDR Rj, Ri	Rj ← Ri	Z, N	
		INC Rj, Ri	Rj ← Ri + 1	Z, N	
		DEC Rj, Ri	Rj ← Ri - 1	Z, N	
		ADD Rj, Ri	Rj ← Rj + Ri	C, V, Z, N	
		ADDC Rj, Ri	Rj ← Rj + Ri + C	C, V, Z, N	
		SUB Rj, Ri	Rj ← Rj + Ri' + 1	C, V, Z, N	
		AND Rj, Ri	Rj ← Rj ∧ Ri	Z, N	
		OR Rj, Ri	Rj ← Rj ∨ Ri	Z, N	
		XOR Rj,Ri	Rj ← Rj ⊕ Ri	Z, N	
		NOT Rj, Ri	Rj ← Ri'	Z, N	
	Register-format Shift Operations	SHL Rj, Ri	Rj ← Ri **<<** 1	NO effect	
		SHR Rj, Ri	Rj ← Ri **>>** 1	NO effect	

- Conclusion3: Each operation returns 1 result, therefore
  - The ALU must have 1 data output.
  - The result is 8-bit binary number.

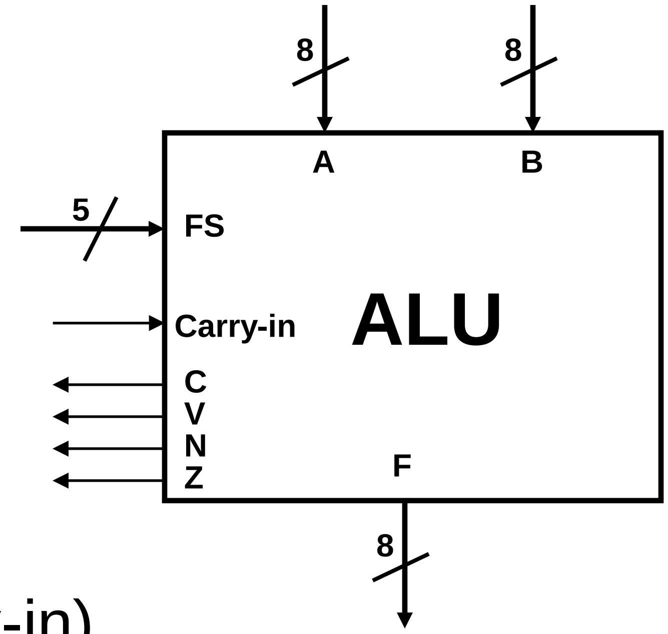
Arithmetic & Logic Unit (ALU)

Instruction Type	Operation	Mnemonic	Operation	Status Bits	Description
Data Manipulation Instructions	**Register-format Arithmetic & Logic Operations**	LDR Rj, Ri	Rj ← Ri	Z, N	
		INC Rj, Ri	Rj ← Ri + 1	Z, N	
		DEC Rj, Ri	Rj ← Ri - 1	Z, N	
		ADD Rj, Ri	Rj ← Rj + Ri	C, V, Z, N	
		ADDC Rj, Ri	Rj ← Rj + Ri + C	C, V, Z, N	
		SUB Rj, Ri	Rj ← Rj + Ri' + 1	C, V, Z, N	
		AND Rj, Ri	Rj ← Rj $\wedge$ Ri	Z, N	
		OR Rj, Ri	Rj ← Rj $\vee$ Ri	Z, N	
		XOR Rj,Ri	Rj ← Rj $\oplus$ Ri	Z, N	
		NOT Rj, Ri	Rj ← Ri'	Z, N	
	Register-format Shift Operations	SHL Rj, Ri	Rj ← Ri **<<** 1	NO effect	
		SHR Rj, Ri	Rj ← Ri **>>** 1	NO effect	

- Conclusion4: Some of the operations must modify status bits therefore
  - The ALU must have 4 status outputs to indicate
    - if Carry and/or oVerflow has occurred
    - if the result of an operation is Zero or Negative

The ALU for Our Processor

- We will use the following block symbol for the ALU.
  - A and B are two 8-bit data inputs for operands.
  - FS is a 5-bit control input to select an operation.
  - The 8-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.
  - Carry-in input is needed for instruction ADDC (ADD with carry-in).



- This block should look familiar to you from your design project!

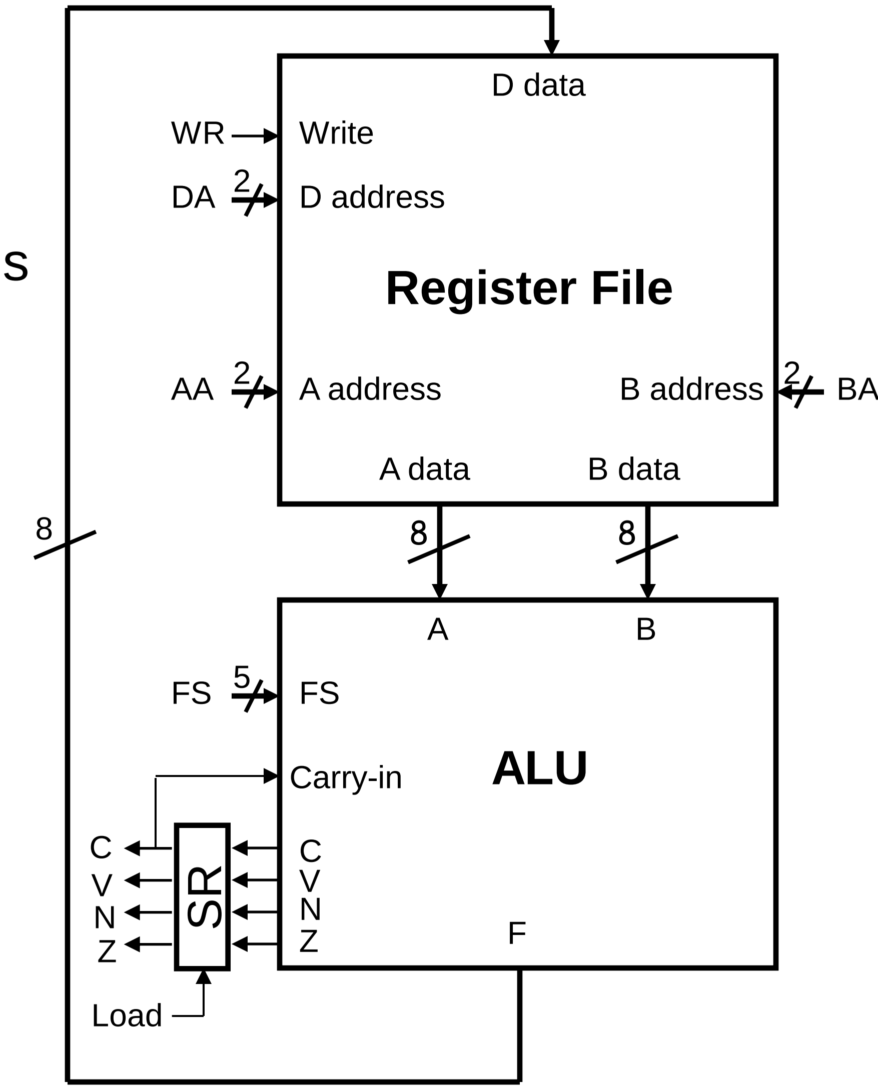
ALU Operations (Functions)

- Each ALU operation is **uniquely encoded** – see the **function selection code FS** in the table.
- The function select code FS is 5 bits long, but there are only 12 different operations here. Why?
- The FS code has a structure:
  - FS(5) = ‘**0**’ indicates data manipulation
  - FS(4:3) =
    - “**00**” indicates arithmetic operations
    - “**01**” indicates propagate or shift operations (except F = B - 1)
    - “**10**” indicates logic operations
- Structuring the FS code helps to design simpler decoder structure for the ALU.

INSTR	FS	Operation
INC	00000	F = B + 1
ADD	00001	F = A + B
ADDC	00010	F = A + B + Carry-in
SUB	00011	F = A + B’ + 1
DEC	00100	F = B - 1
LDR	00101	F = B
SHR	00110	F = sr B (shift right)
SHL	00111	F = sl B (shift left)
AND	01000	F = A $\wedge$ B (AND)
OR	01001	F = A $\vee$ B (OR)
XOR	01010	F = A $\oplus$ B
NOT	01011	F = B’

Initial Datapath for Our Processor

- 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.
  - The status bits are stored in the status register SR.
- WR, DA, AA, BA, FS and Load are control signals.
  - Their values determine the exact actions taken by the datapath,
  - That is, which registers are used and for what operation.

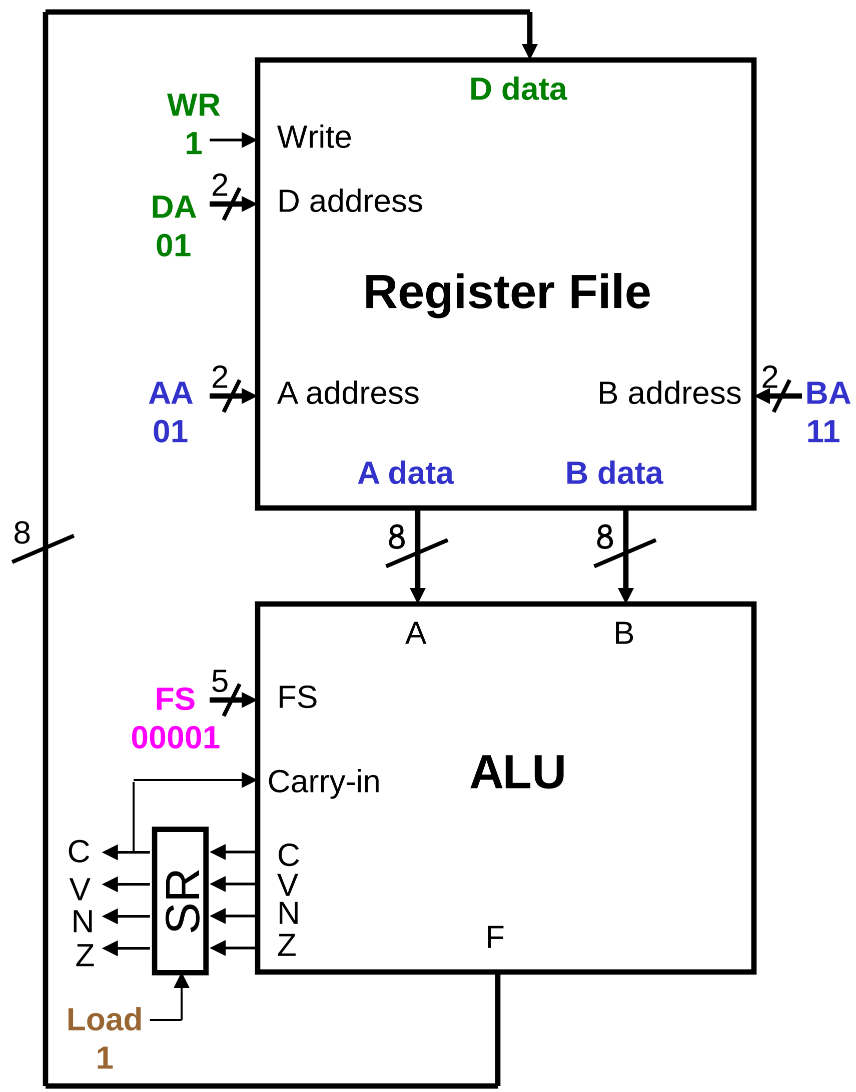


An Example Computation

- Let us look at the proper control signals for executing the processor instruction below:

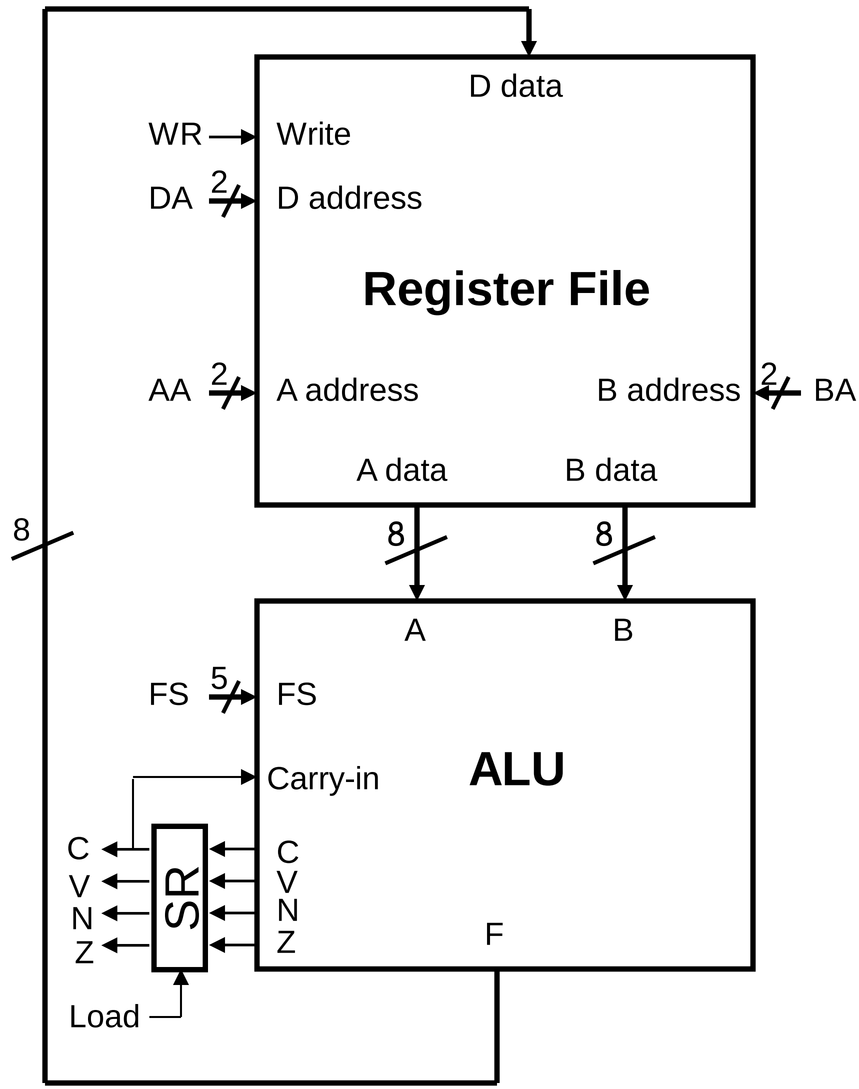
ADD R1, R3 R1 ← R1 + R3

- Set all control signals simultaneously as explained below.
- 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 = 00001 (A + B).
- Set DA = 01 and WR = 1. On the next positive clock edge, the ALU result (R1 + R3) will be stored in R1.
- Set Load = 1. On the next positive clock edge, the ALU status bits (C, V, N, Z) will be stored in SR.



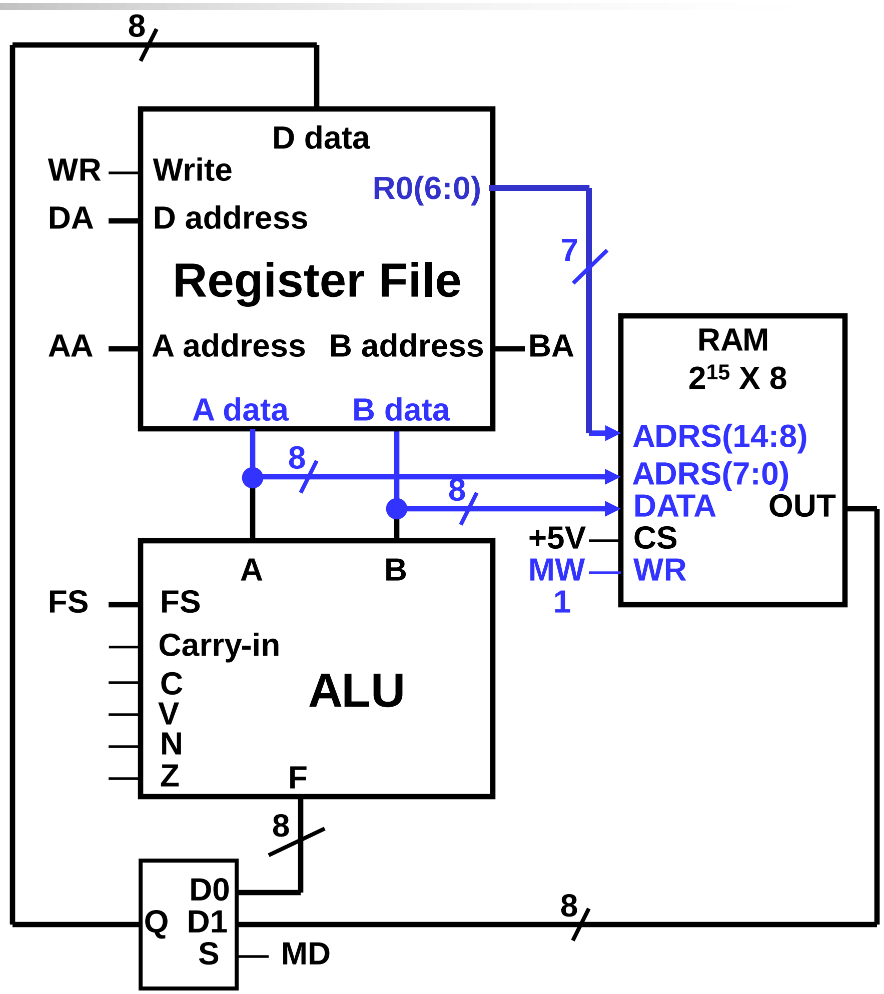
Two Issues with this Datapath

- **Q1:** Four registers is not a lot. What if we need more storage?
- **A1:** Our processor has a Data RAM Memory and supports data movement between RAM and registers with instructions **ST** and **LD**.
- **Q2:** What if we have to do operations with constants?
- **A2:** Our processor has two instructions: **LDI** (load an 8-bit constant in a register), **STI** (storing an 8-bit constant in a memory location)
- **Problem!** Our initial datapaht on the right does not support the answers A1 and A2. Why?
- **Solution:** We have to refine our initial datapath. See next slides!



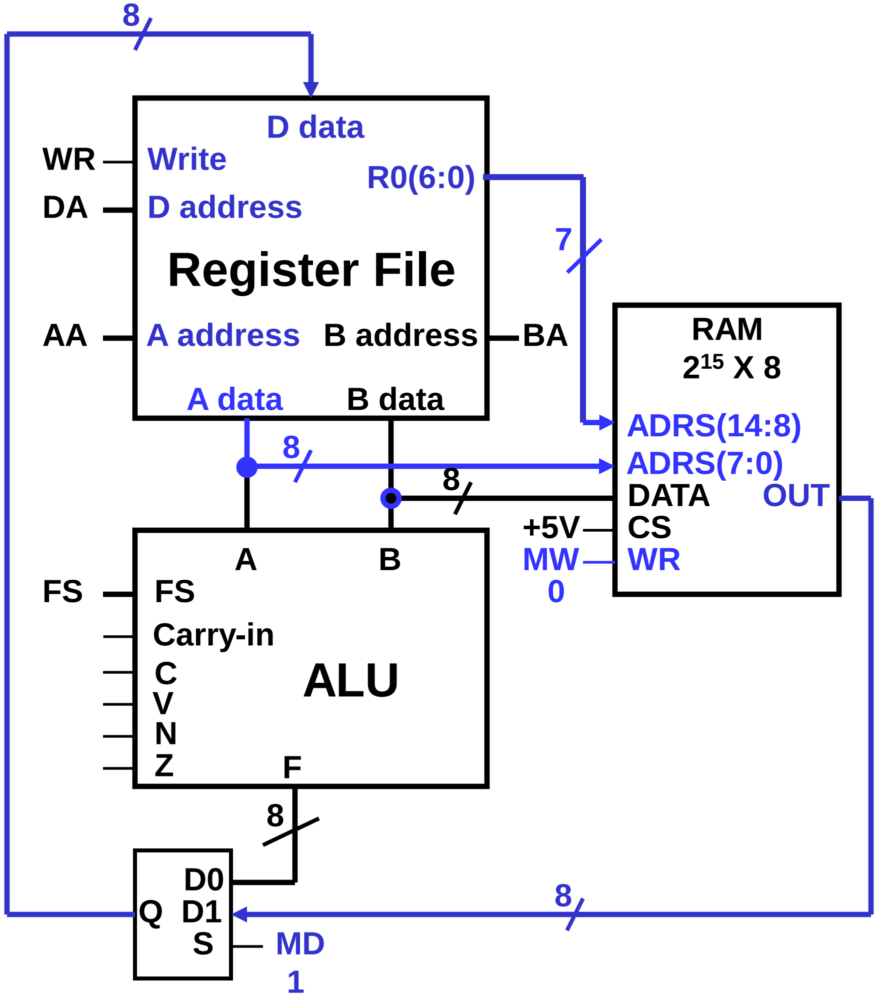
Writing to RAM

- Here is 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 and Register R0 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.)



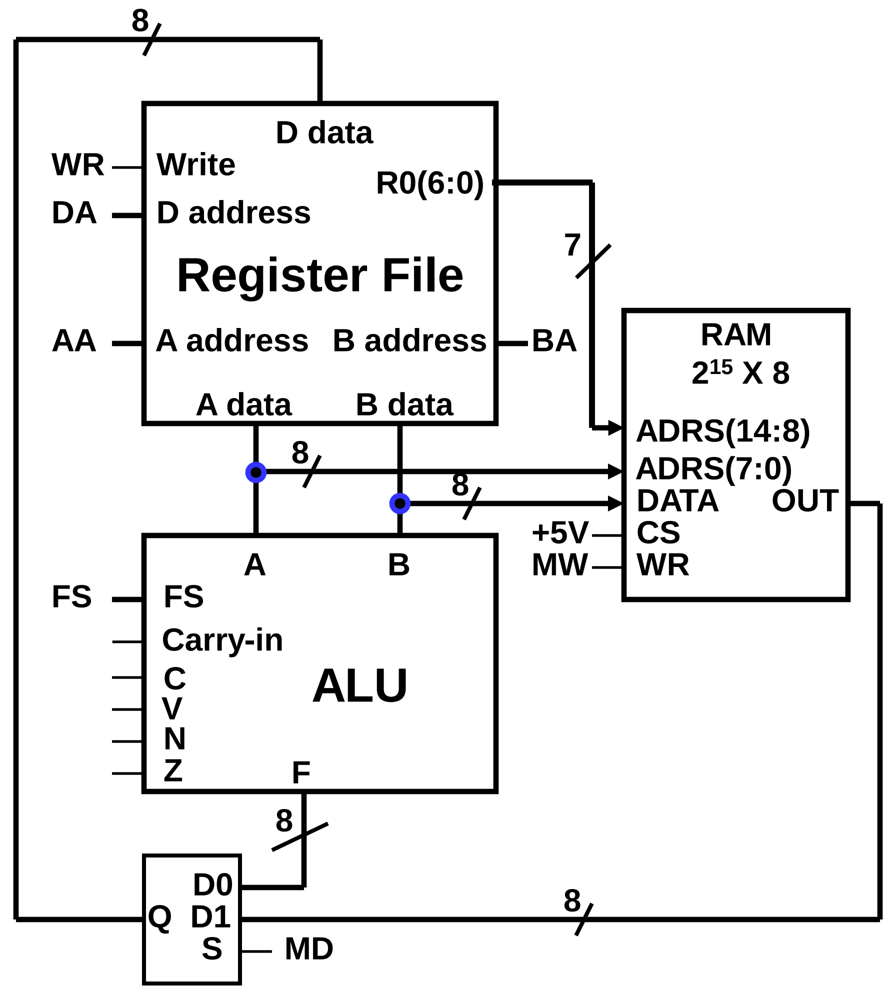
Reading from RAM

- To *read* from RAM, A data and register R0 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 is 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 8-bit registers, we use a 2^{15} x 8 RAM.
  - Address bits 14 down to 8 are always taken from register R0.
  - Address bits 7 down to 0 can be taken from any register.
- For simplicity we assume the RAM is at least as fast as the processor clock. (This is definitely *not* the case in real processors these days!)



Example Sequence of Instructions

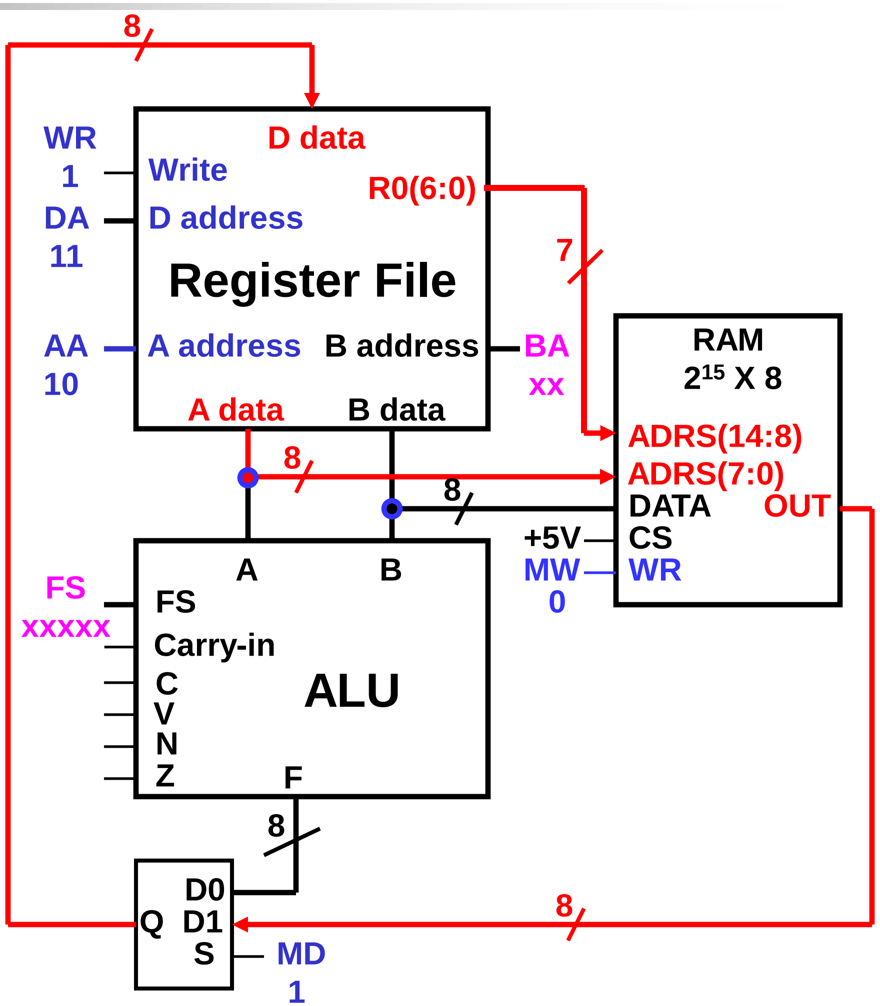
- The RAM memory access in our processor is supported by two instructions:
  - **LD Rj, (Ri)** -- load register Rj with the content of a RAM memory cell at address given by register Ri;
  - **ST (Rj), Ri** -- store the content of register Ri in a RAM memory cell at address given by register Rj;
- Here is a simple series of memory/register transfer instructions:

LD R3, (R2)	R3 ← Mem[R0|R2]
DEC R3, R3	R3 ← R3 - 1
ST (R2), R3	Mem[R0|R2] ← R3

- This just decrements the content of RAM memory cell at address R0|R2 .
  - Again, our ALU only operates on registers, so the RAM contents must first be loaded into a register, and then saved back to RAM.
  - We will assume that R0 and R2 contain a valid memory address.
- How would these instructions execute in our datapath?

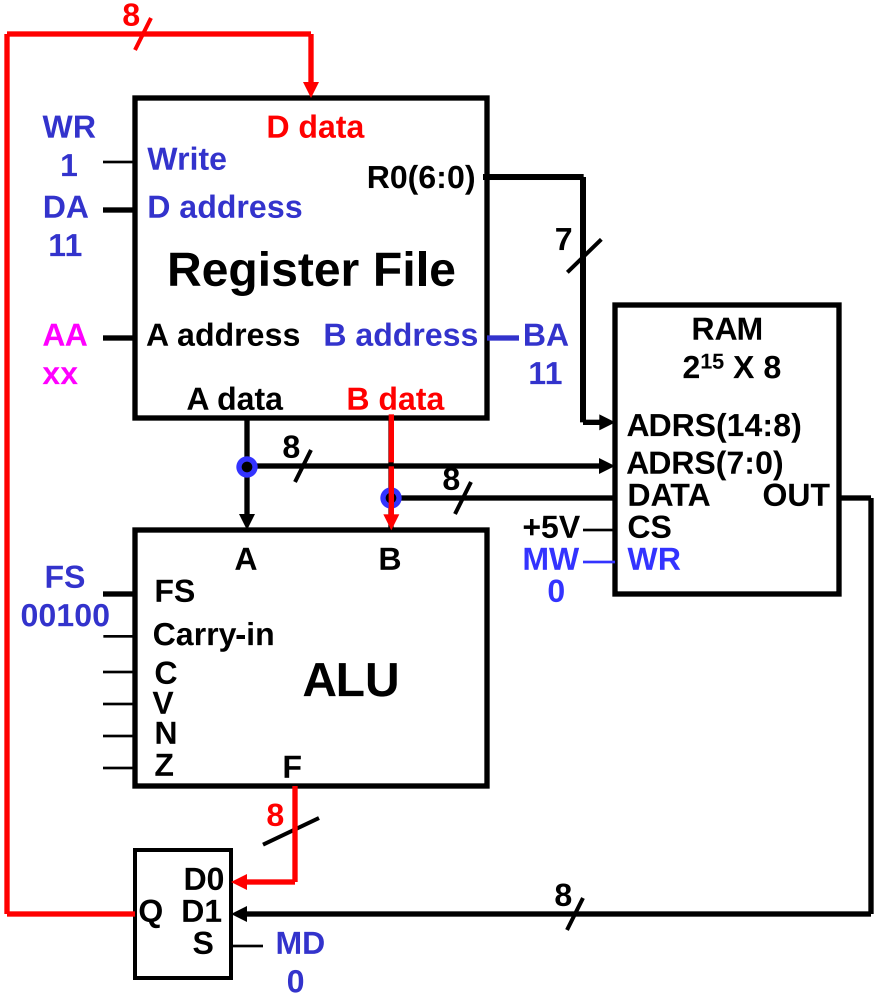
"LD R3, (R2)" is R3 ← Mem[R0|R2]

- AA should be set to 10, to read register R2.
- The value in R2 will be sent to the RAM address inputs, so Mem[R0|R2] 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 will need to set DA = 11 and WR = 1.
- MW should be 0, so nothing is accidentally changed in RAM.
- We do not use the ALU, thus FS value can be arbitrary)
- We do not use the second register file output, thus BA also can be arbitrary.



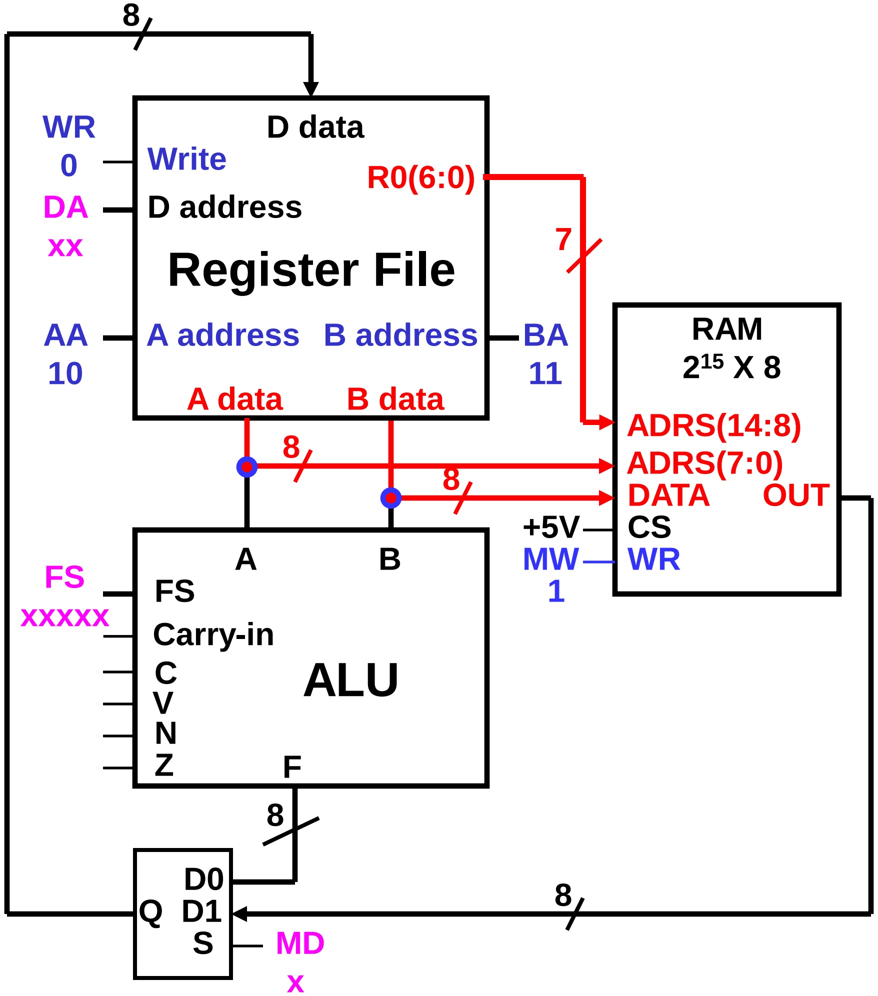
“DEC R3, R3” is R3 ← R3 - 1

- BA = 11, so R3 is read from the register file and sent to the ALU’s B input.
- FS needs to be 00100 for the operation B - 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 is not changed.
- We do not use AA.



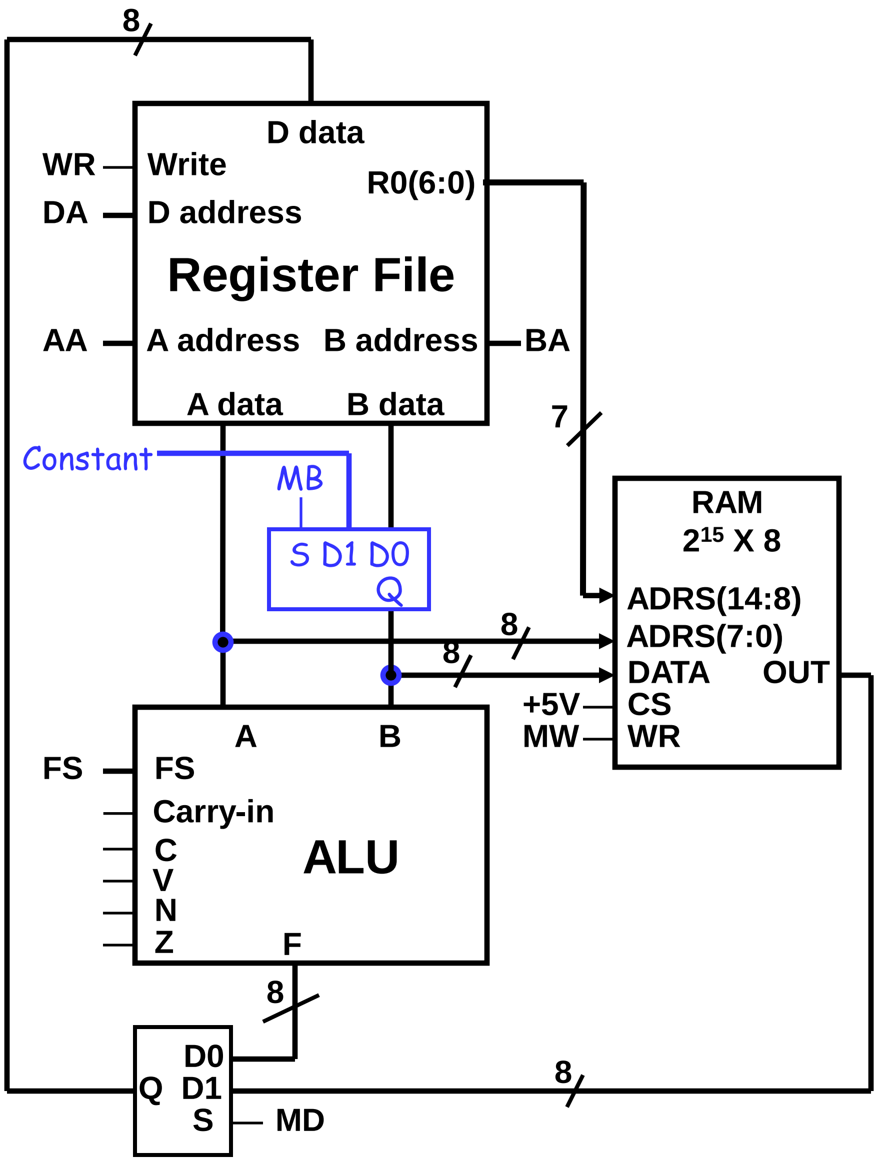
"ST (R2), R3" is Mem[R0|R2] ← R3

- Finally, we want to store the contents of R3 into RAM address R0|R2.
- Remember the RAM address comes from "A data," and the contents come from "B data."
- So, we have to set AA = 10 and BA = 11. This sends R2 to ADRS(7:0), 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 do not 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,
  - One extra MUX is added.
  - With one extra control signal MB.
- Intuitively, it provides an easy way to initialize a register or memory location with some arbitrary number (8-bit constant).
- The constant comes from the instructions **LDI** and **STI** (see instruction format 2!).
- At home try to set the control signals of the datapath on the right for the following instructions:
  - **LDI R2, #0xc8**
  - **STI (R1), #0x3f**



Who is Configuring the Datapath?

- The datapath on the previous slide is a complete datapath for our simple processor, i.e.,
  - the datapath supports all Data Manipulation instructions
  - the datapath supports all Data Movement instructions
- Different actions are performed when we provide different values for the datapath control signals
  - See the instruction examples on previous slides
- In processors, the datapath actions are determined by the program that is loaded and running
- The Control Unit is responsible for generating the correct control signals for a datapath, based on the program code
- We will talk about the control unit next week.

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
  - 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 will see
  - how programmers can give commands to the processor
  - how these commands are translated in control signals for the datapath.