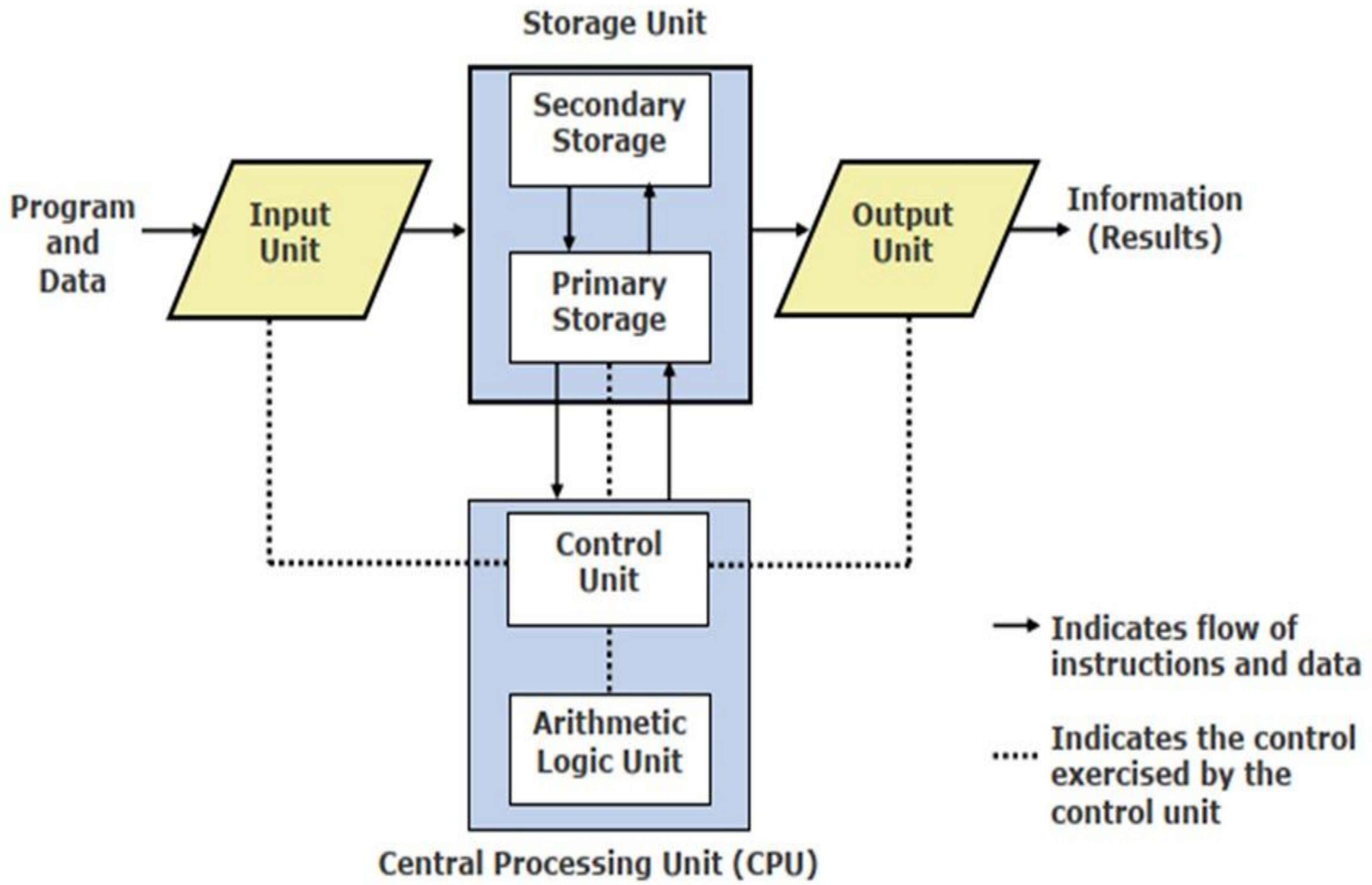
Introduction

1. Digital circuits are frequently used for arithmetic operations
2. Fundamental arithmetic operations on binary numbers and digital circuits which perform arithmetic operations will be examined.
3. HDL will be used to describe arithmetic circuits.
4. An arithmetic/logic unit (ALU) accepts data stored in memory and executes arithmetic and logic operations as instructed by the control unit.

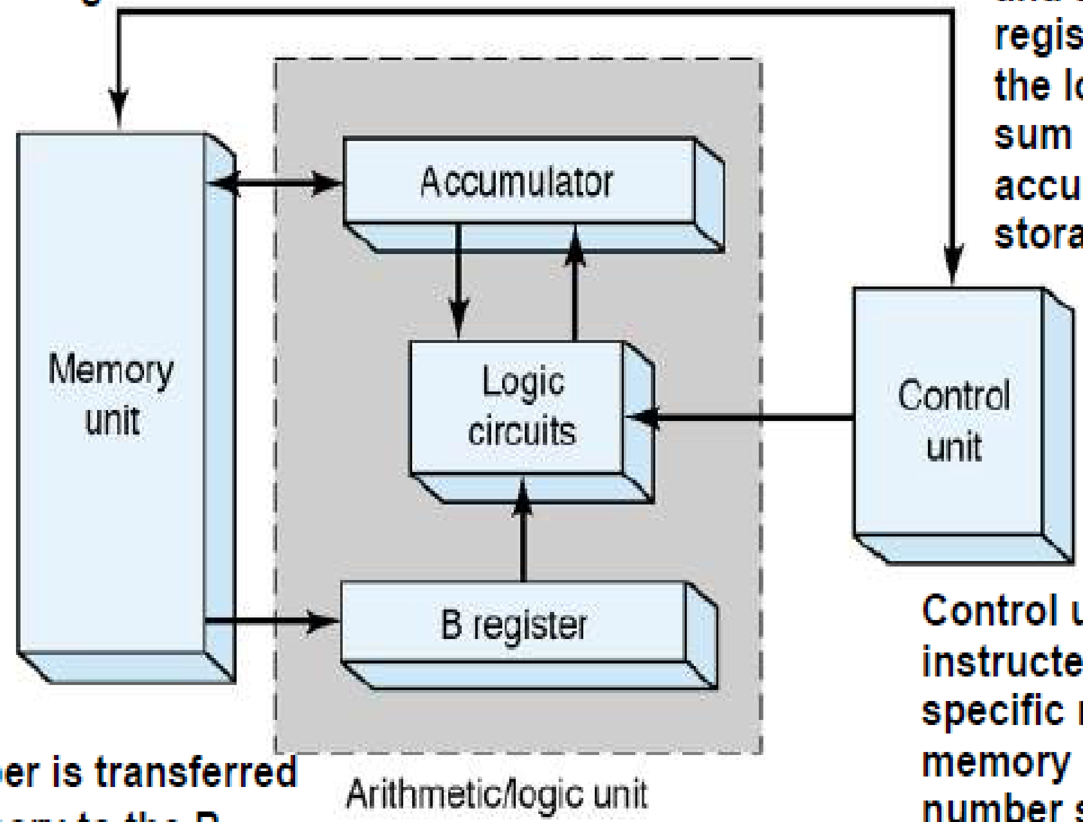
Basic Organization of a Computer System



Arithmetic Circuits

4 The new number remains in the accumulator for further operations or can be transferred to memory for storage

3 Number in B register and accumulator register are added in the logic circuit, with sum sent to accumulator for storage.



2 The number is transferred from memory to the B register.

1 Control unit is instructed to add a specific number from a memory location to a number stored in the accumulator register.

Datapaths

- We'll focus on computer architecture: how to assemble the combinational and sequential components we've studied so far into a complete computer.
- The datapath is the part of the central processing unit (CPU) that does the actual computations.

Keeping it simple!

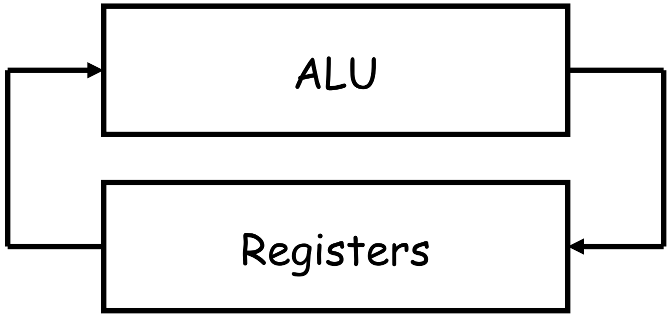
- *Abstraction* is very helpful in understanding processors.
  - Although we studied how devices like registers and muxes are built, we don't need that level of detail here.
  - You should focus more on *what* these component devices are doing, and less on *how* they work.
- Otherwise, it's easy to get bogged down in the details, and datapath and control units can be a little intimidating.

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.

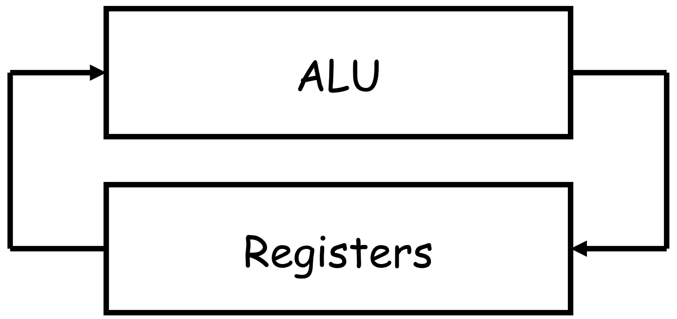
- We'll look in detail at a processor's datapath, which is responsible for doing all of the dirty work.
  - An ALU does computations, as we've seen before.
  - A limited set of registers serve as fast temporary storage.
  - A larger, but slower, random-access memory is also available.

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.

Register transfers



- Fundamentally, the processor is just moving data between registers, possibly with some ALU computations.
- To describe this data movement more precisely, we'll use a register transfer language.
  - The objects in the language are *registers*.
  - The basic operations are *transfers*, where data is copied from one register to another.
- We can also use the ALU to perform arithmetic operations on the data while we're transferring it.

Register transfer language review (from Chapter 8)

- Two-character names denote registers, such as R0, R1, DR, or SA.
- Arrows indicate data transfers. To copy the contents of the source register R2 into the destination register R1 in one clock cycle:

$$R1 \leftarrow R2$$

- A conditional transfer is performed only if the Boolean condition in front of the colon is true. To transfer R3 to R2 when K = 1:

$$K\text{: } R2 \leftarrow R3$$

- Multiple transfers on the *same* clock cycle are separated by commas.

$$R1 \leftarrow R2,\ \ K\text{: } R2 \leftarrow R3$$

- Don't confuse this register transfer language with assembly language, which we'll discuss later.

Register transfer operations (cont'd)

- We can apply arithmetic operations to registers.

$$R1 \leftarrow R2 + R3$$
$$R3 \leftarrow R1 - 1$$

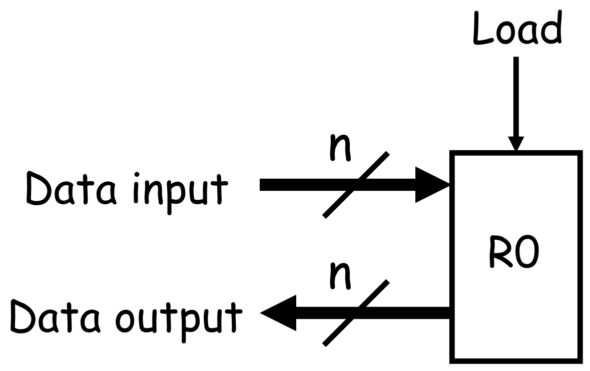
- Logical operations are applied bitwise. AND and OR are denoted with special symbols, to prevent confusion with arithmetic operations.

$R2 \leftarrow R1 \wedge R2$	bitwise AND
$R3 \leftarrow R0 \vee R1$	bitwise OR

- Lastly, we can shift registers. Here, the source register R1 is not modified, and we assume that the shift input is just 0.

$R2 \leftarrow \text{sl } R1$	left shift
$R2 \leftarrow \text{sr } R1$	right shift

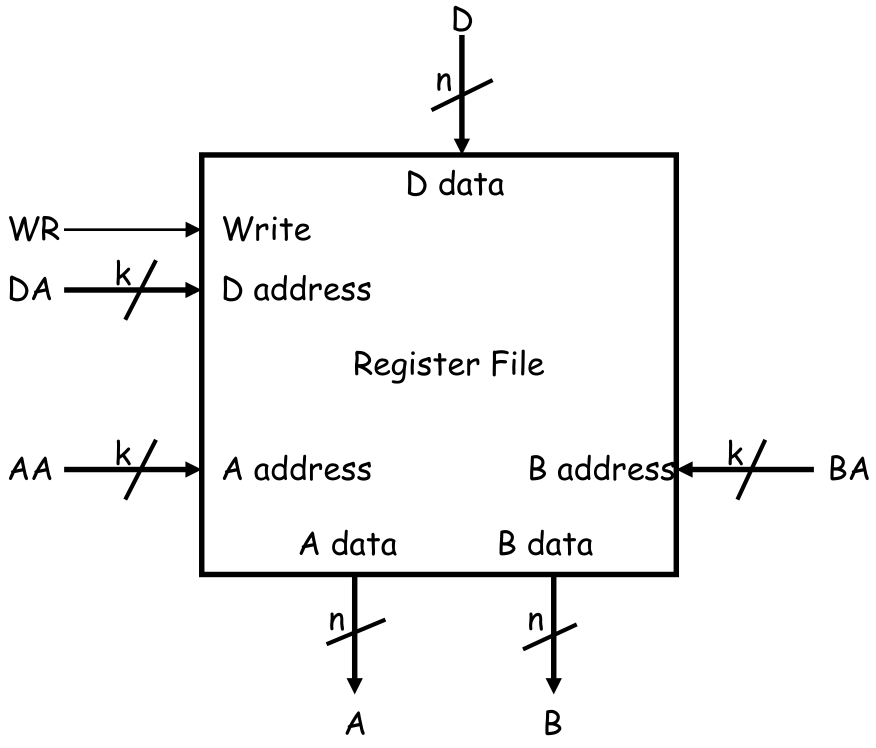
Block symbols for registers



- We'll use this block diagram to represent an n-bit register.
- There is a data input and a load input.
  - When Load = 1, the data input is stored into the register.
  - When Load = 0, the register will keep its current value.
- The register's contents are always available on the output lines, regardless of the Load input.
- The clock signal is not shown because it would make the diagram messy.
- Remember that the input and output lines are actually n bits wide!

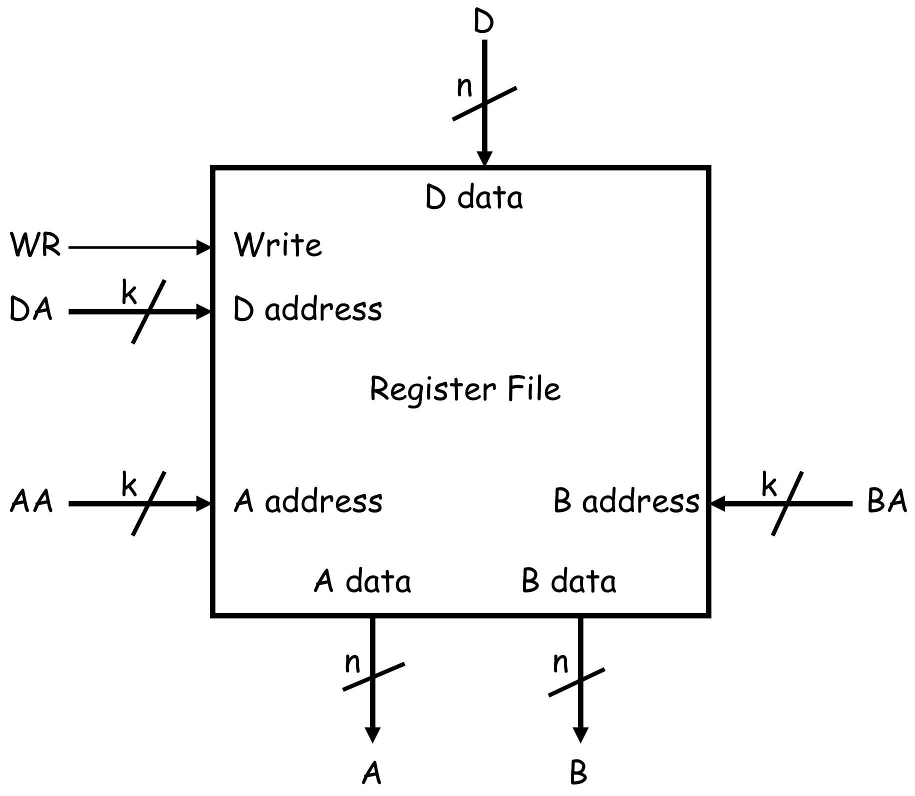
Register file

- Modern processors have 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 $2^k \times n$ 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.



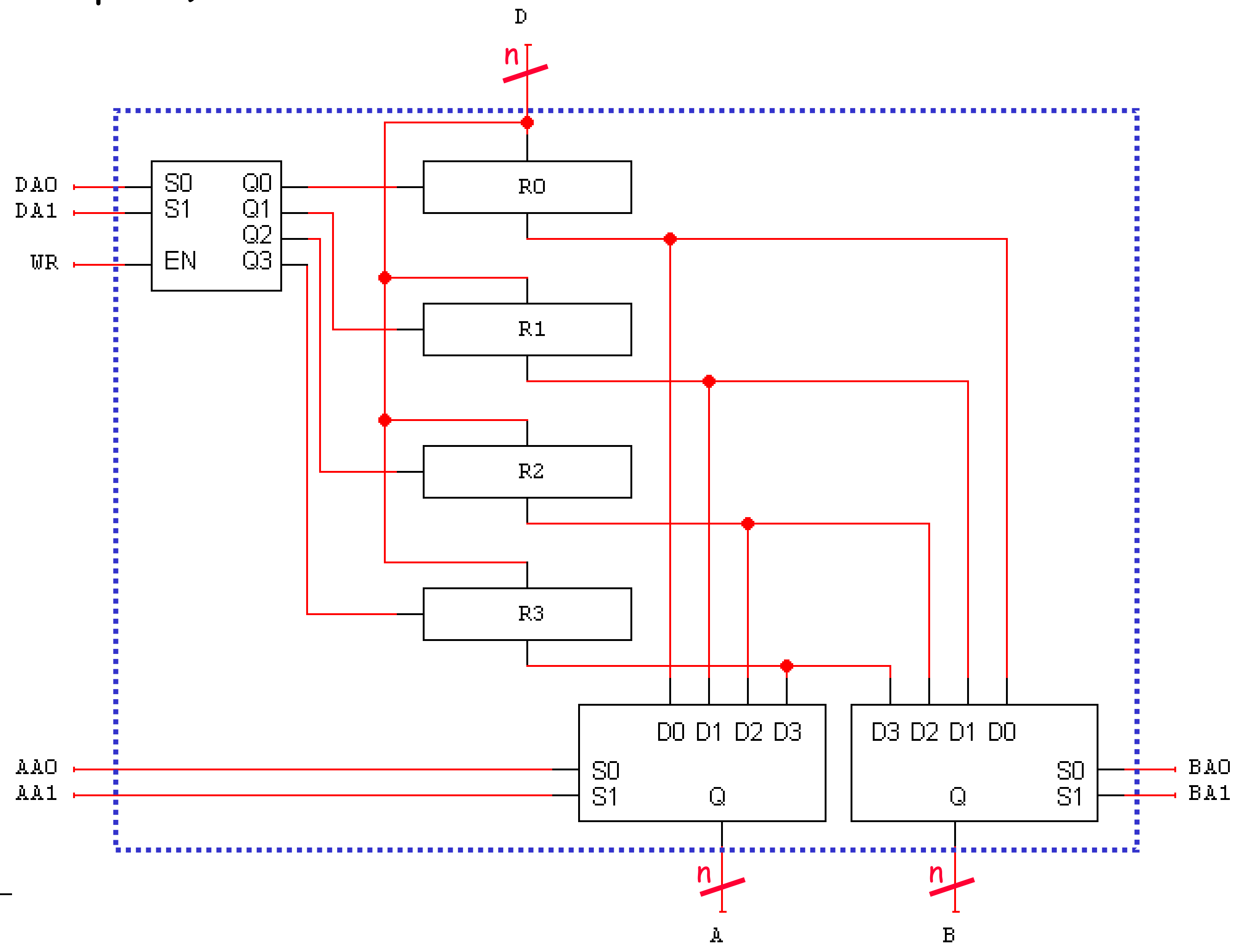
Accessing the register file

- 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 4 x n register file for all our examples.)

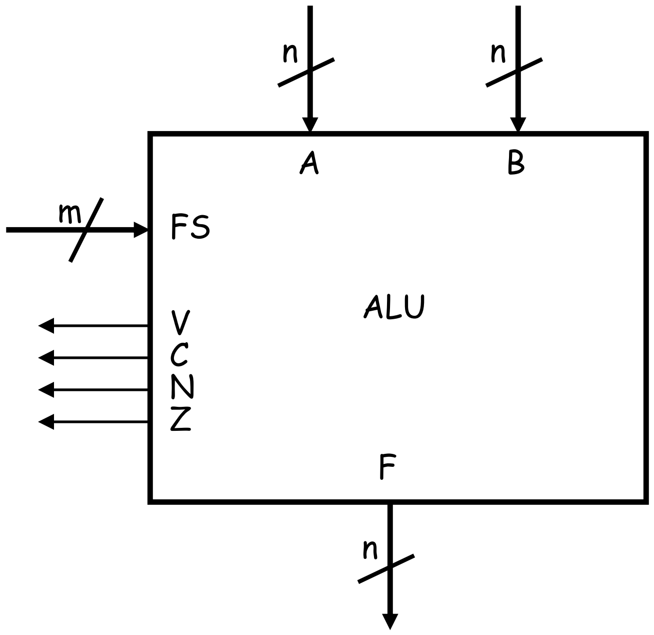


Explaining the register file

- 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.



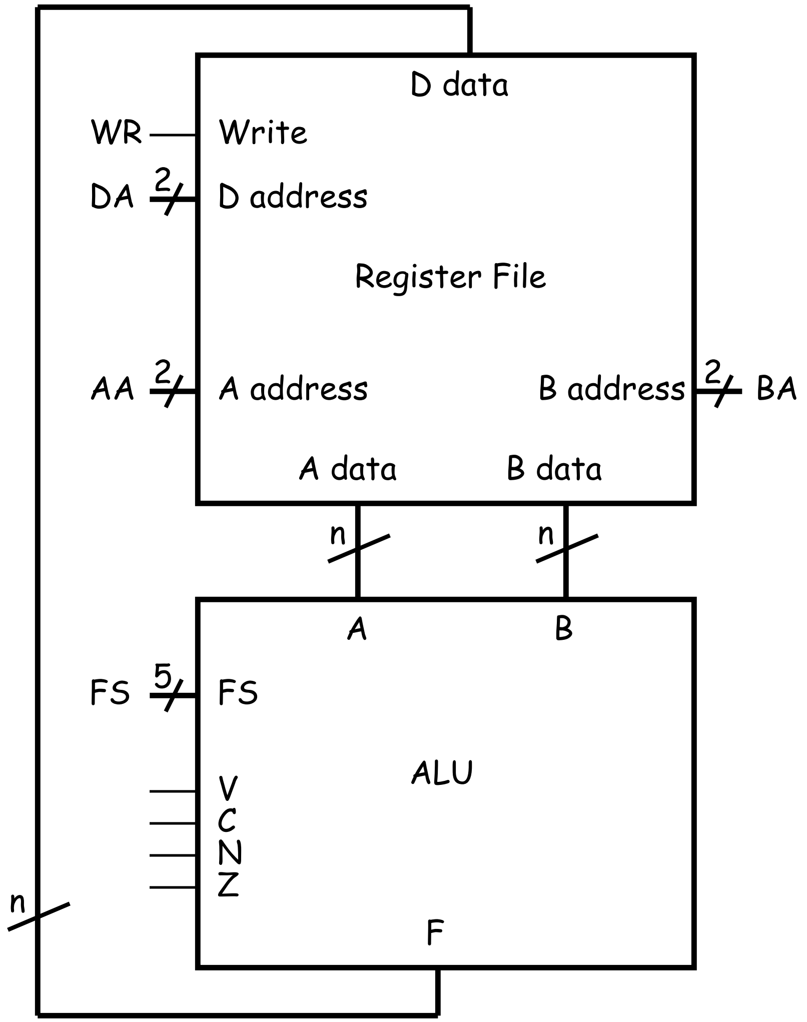
ALU functions

- For concrete examples, we'll use the ALU as it's presented in the textbook.
- The table of operations on the right is taken from the book.
- The function select code FS is 5 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
00000	F = A
00001	F = A + 1
00010	F = A + B
00011	F = A + B + 1
00100	F = A + B'
00101	F = A + B' + 1
00110	F = A – 1
00111	F = A
01000	F = A $\wedge$ B (AND)
01010	F = A $\vee$ B (OR)
01100	F = A $\oplus$ B
01110	F = A'
10000	F = B
10100	F = sr B (shift right)
11000	F = sl B (shift left)

Our 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.
- Remember that many of the signals here are actually multi-bit values.

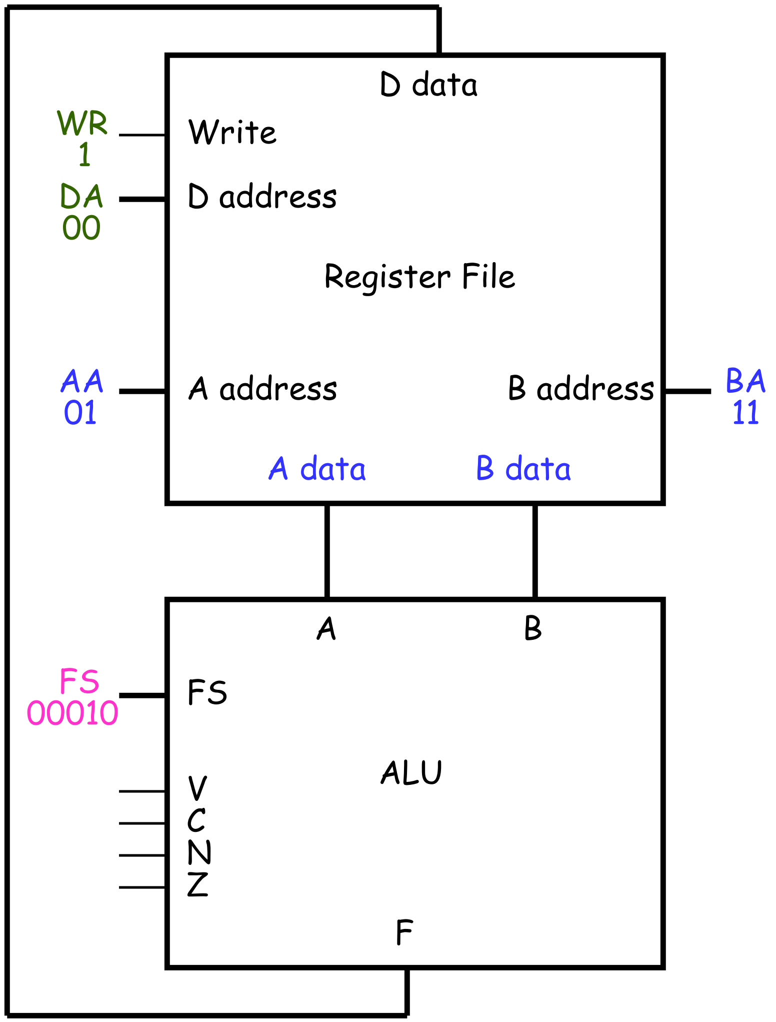


An example computation

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

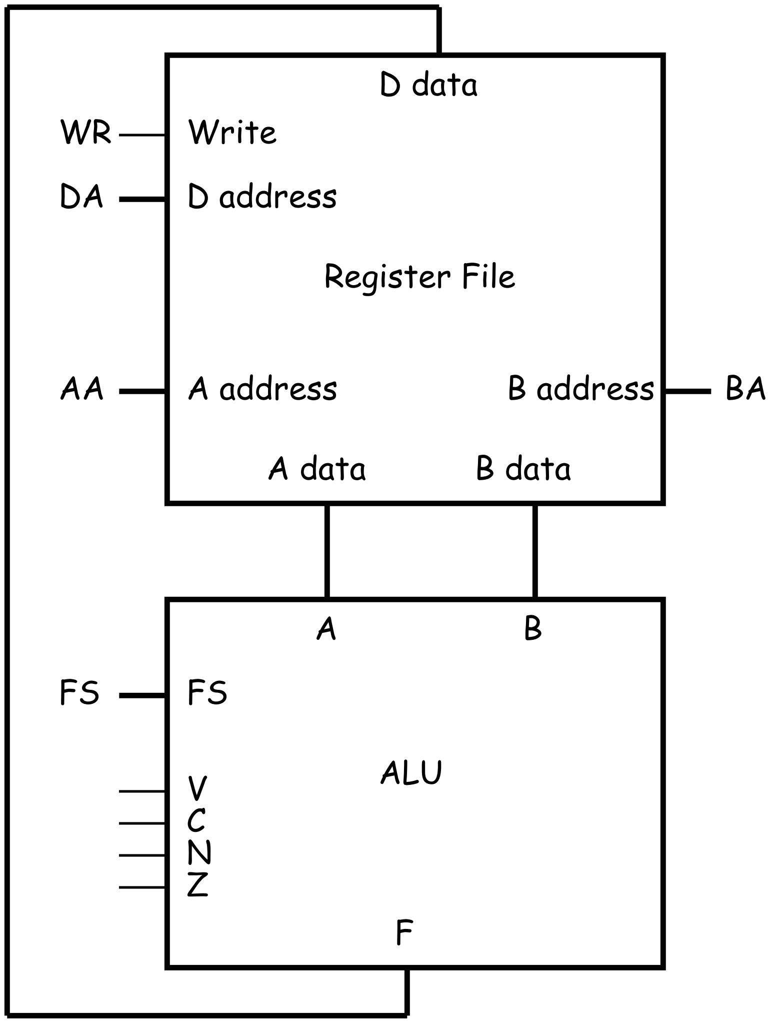
  $$R0 \leftarrow R1 + R3$$

- 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 = 00010 (A + B).
- Set DA = 00 and WR = 1. On the next positive clock edge, the ALU result (R1 + R3) will be stored in R0.



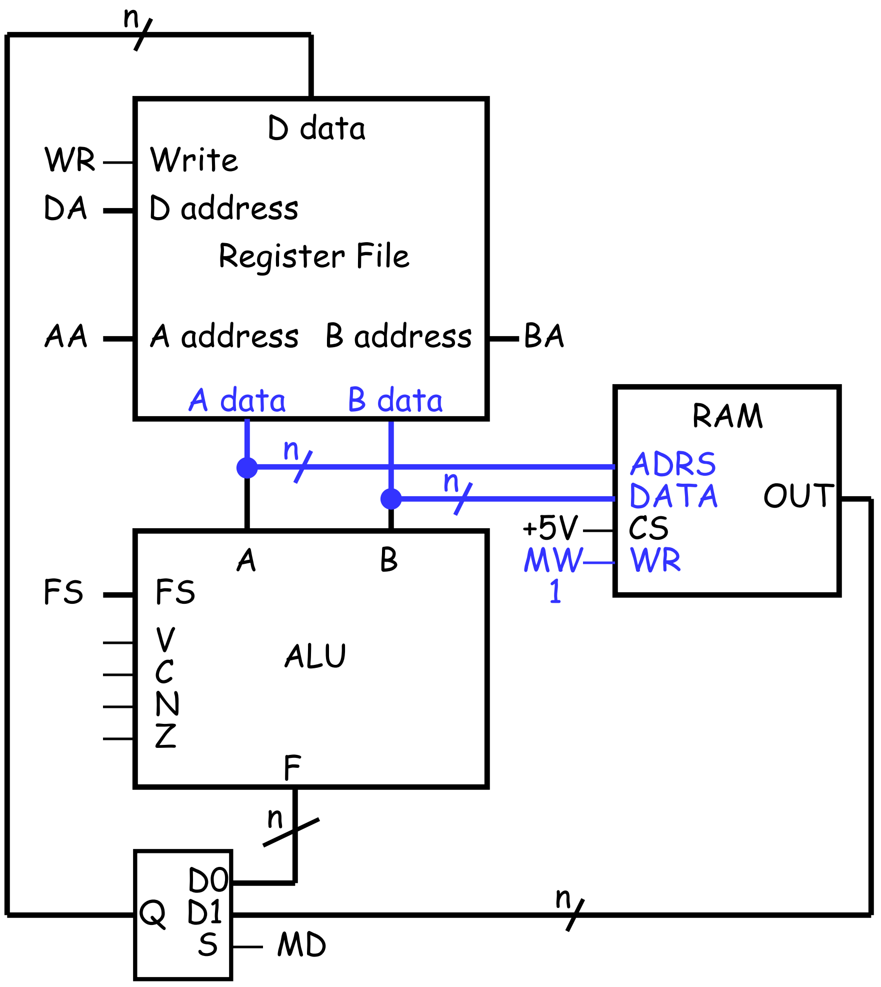
Two questions

- 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?



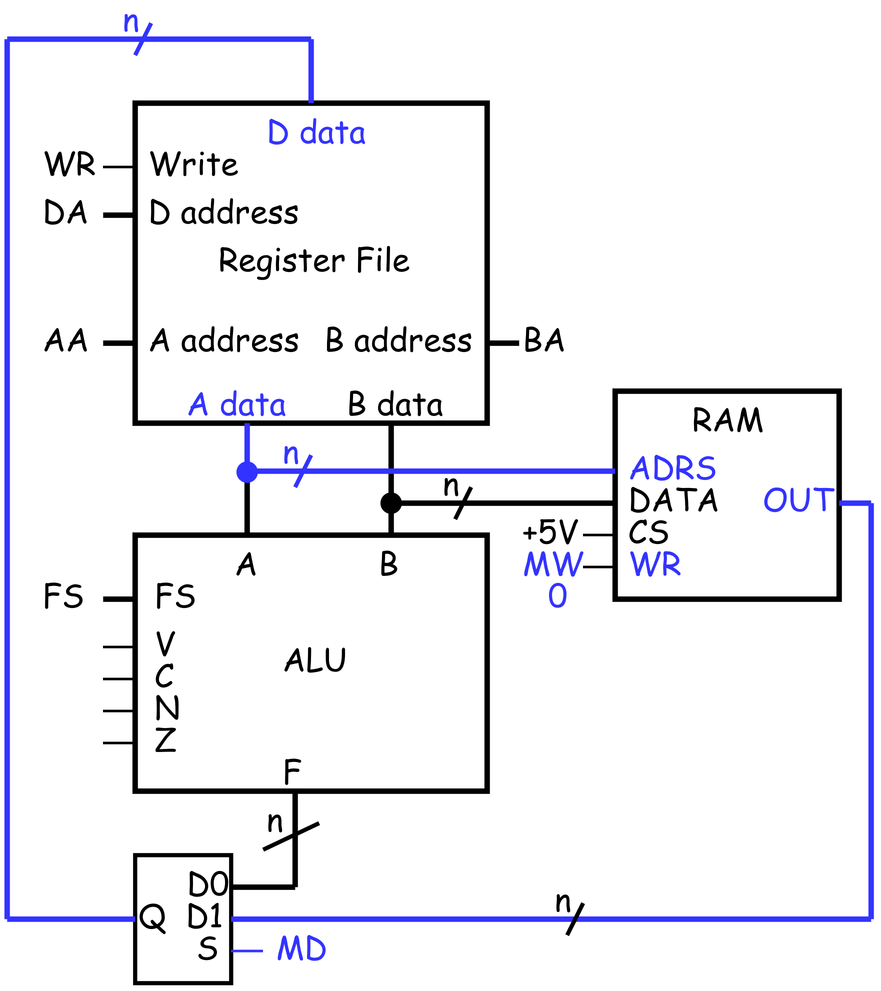
We can access RAM also

- 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.)



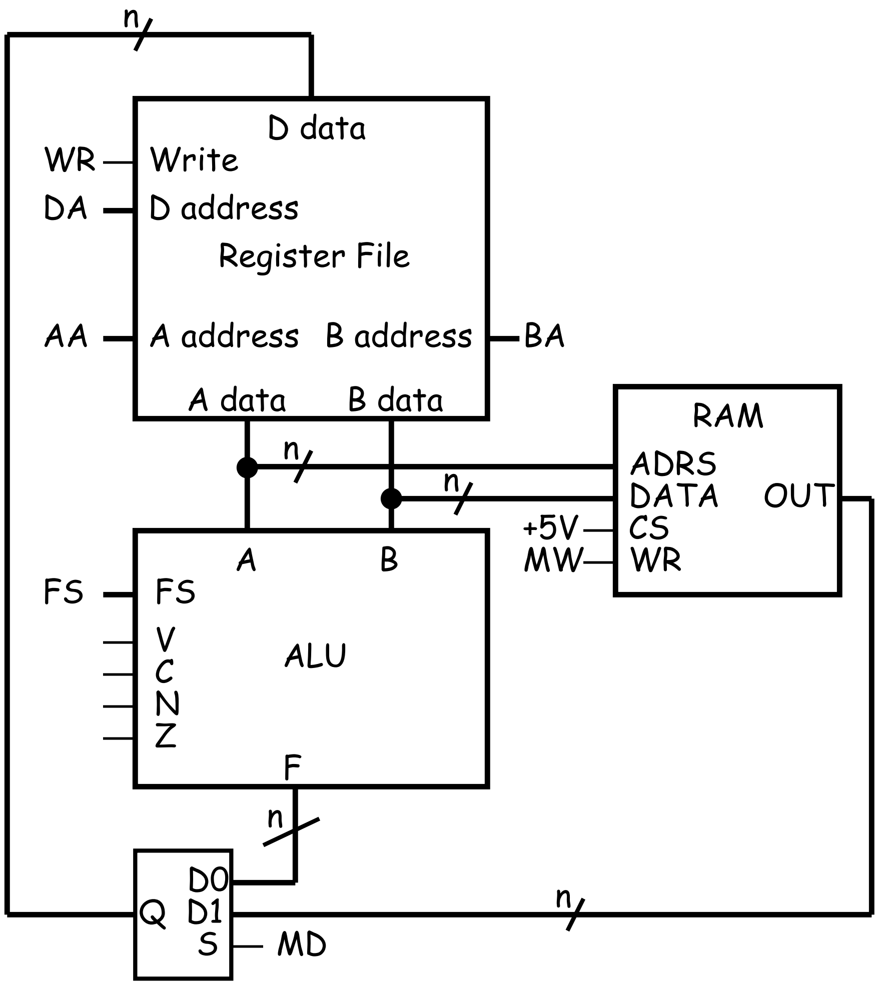
Reading from RAM

- 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.



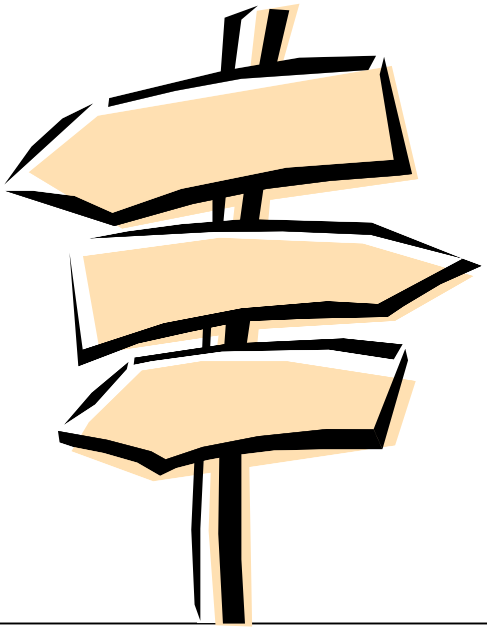
Notes about this setup

- 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 $2^n \times n$ RAM.
- For simplicity we'll assume the RAM is at least as fast as the CPU clock. (This is definitely *not* the case in real processors these days.)



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

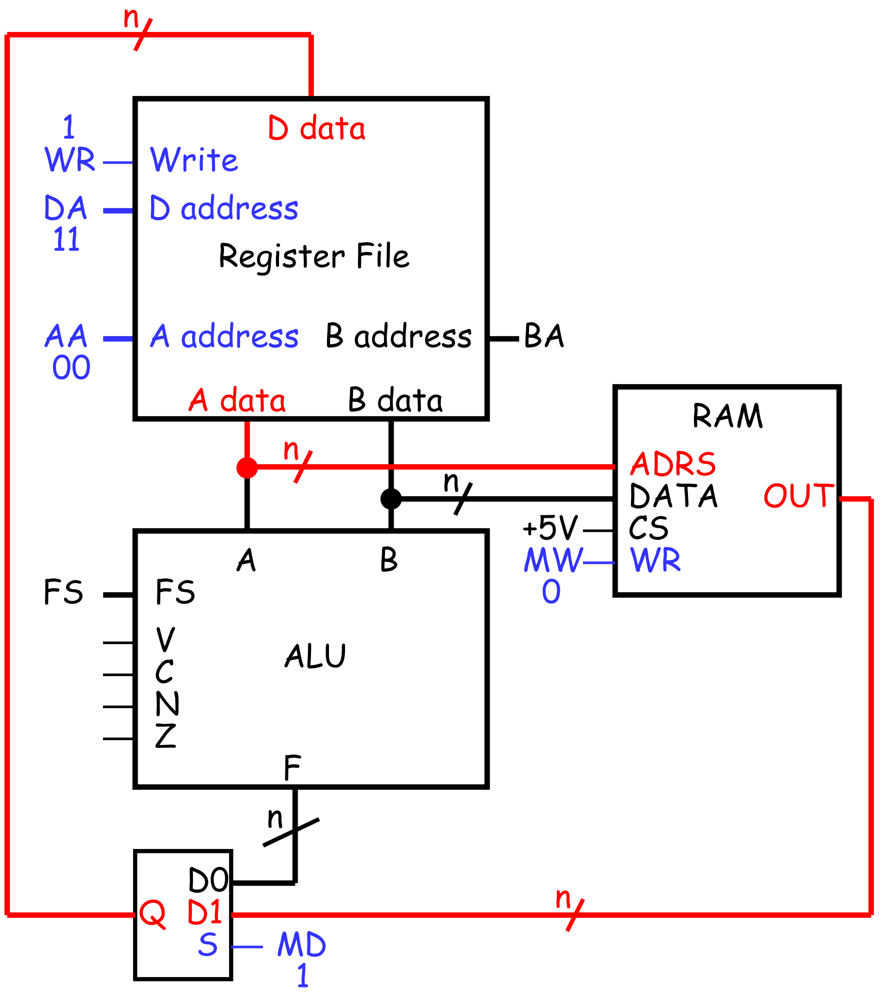
- Here is a simple series of register transfer instructions:

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

- 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?

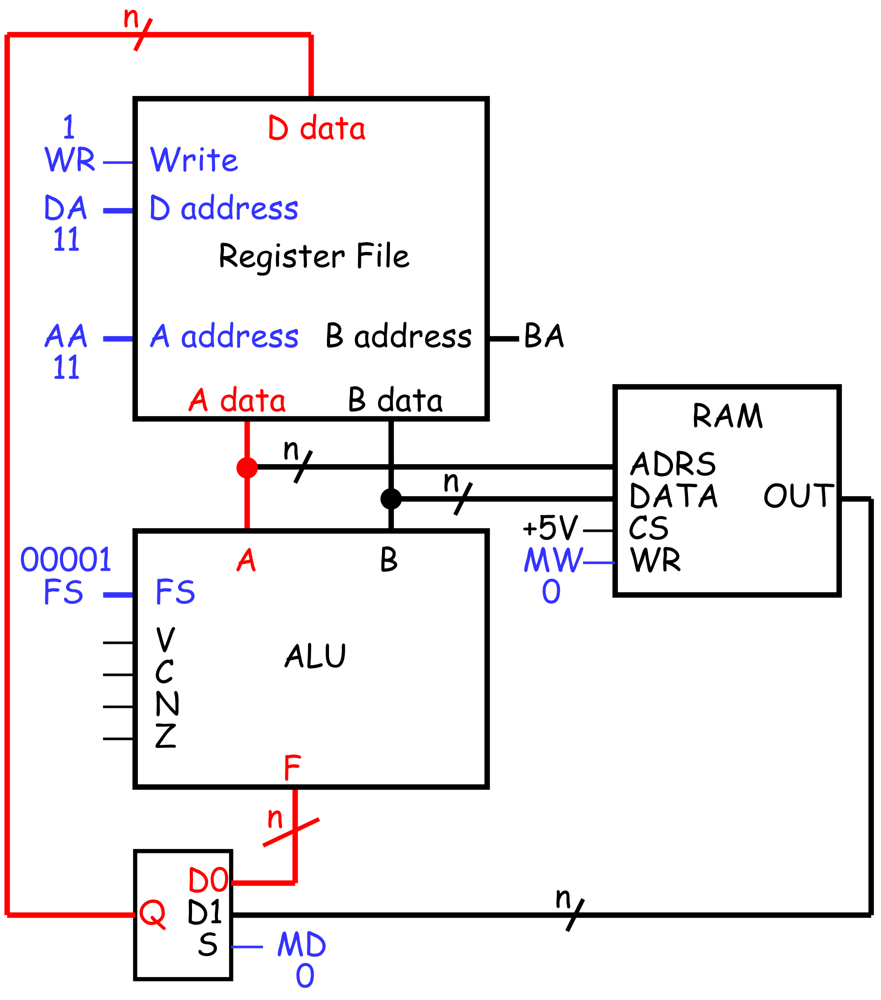
R3 ← M[R0]

- 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).



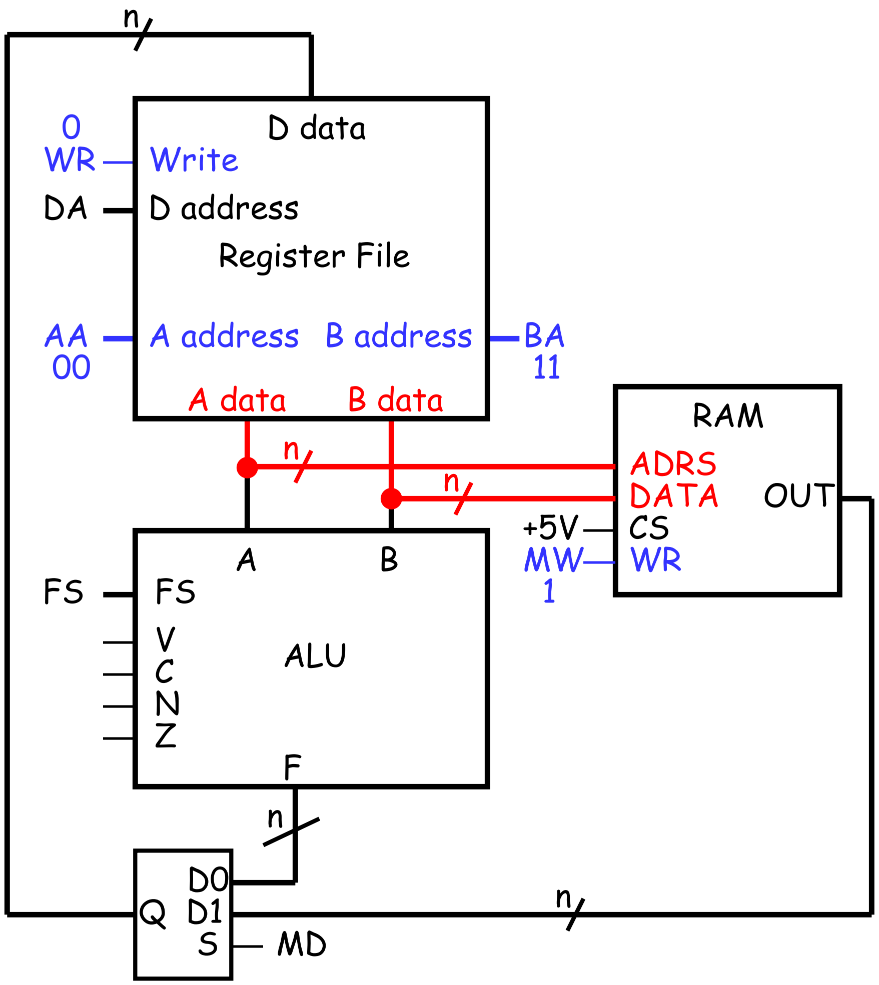
R3 ← R3 + 1

- AA = 11, so R3 is read from the register file and sent to the ALU's A input.
- FS needs to be 00001 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.



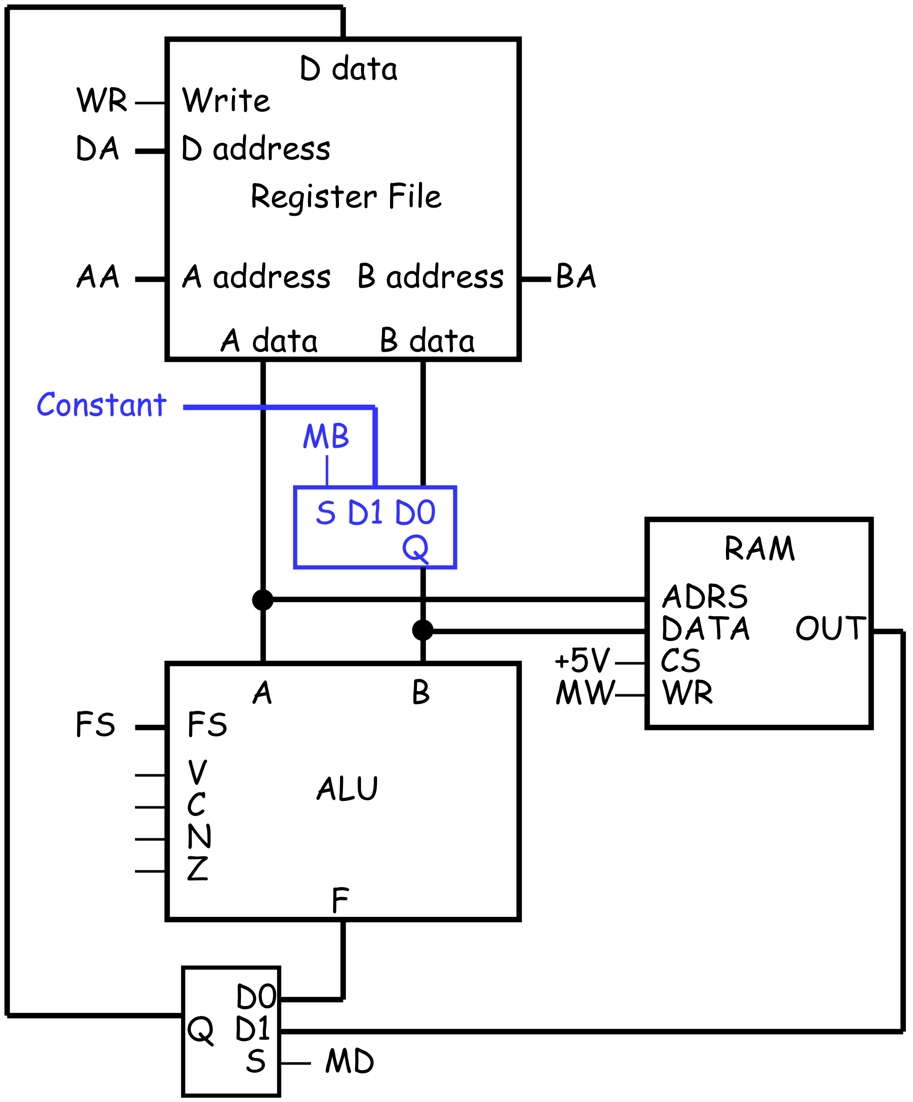
M[R0] ← R3

- 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.
- We'll see how this is used later. Intuitively, it provides an easy way to initialize a register or memory location with some arbitrary number.



Control units

- From these examples, you can see that different actions are performed when we provide different inputs for the datapath control signals.
- The second question we had was "Who exactly decides which registers are read and written and which ALU function is executed?"
  - In real computers, the datapath actions are determined by the program that's loaded and running.
  - A control unit is responsible for generating the correct control signals for a datapath, based on the program code.
- We'll talk about programs and control units later.

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, we'll see how programmers can give commands to the processor, and how those commands are translated in control signals for the datapath.

HW

1. Design a 4 bit arithmetic circuit, with two selection variables S1 and S0, that generates the arithmetic operations below. Draw the logic diagram for a single bit stage. (Q 10-4)
2. A computer has a 32 bit instruction word broken into fields as follows: opcode, 6 bits; two register fields, 6 bits each; and one immediate operand/register field, 14 bits.

- (a) What is the maximum number of operations that can be specified?
- (b) How many registers can be addressed?
- (c) What is the range of unsigned immediate operands that can be provided? (Q 10-13)

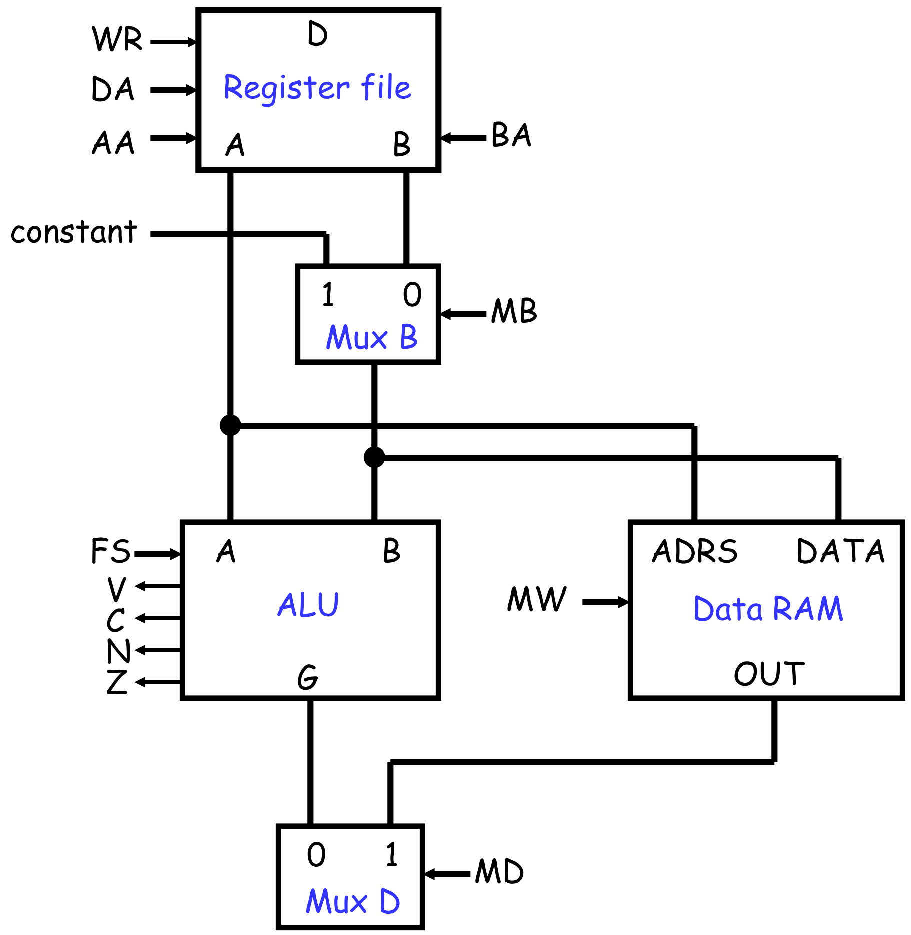
S1S0	Cin = 0	Cin = 1
00	F = A + B (add)	F = A + B + 1
01	F = A (transfer)	F = A + 1 (increment)
10	F = B′ (complement)	F = B′ + 1 (negate)
11	F = A + B′	F = A + B′ + 1 (subtract)

Control units

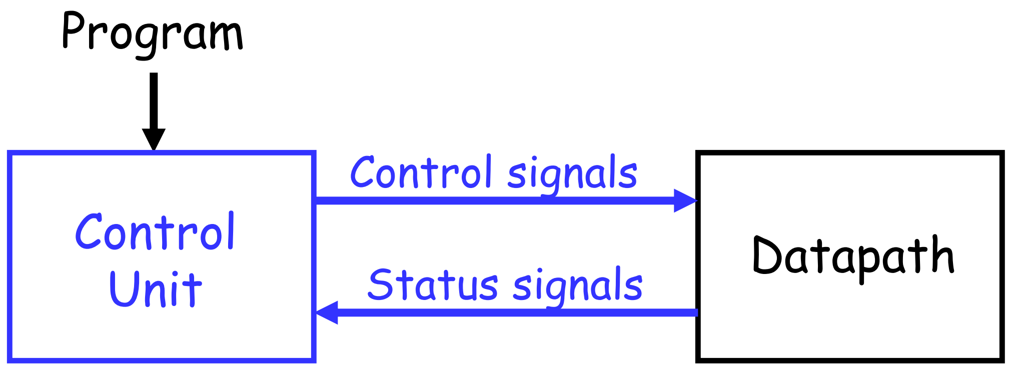
- We introduced the basic structure of a control unit, and translated assembly instructions into a binary representation.
- The last piece of the processor is a control unit to convert these binary instructions into datapath signals.
- At the end we'll have a complete example processor!

Datapath review

- Set WR = 1 to write one of the registers.
- DA is the register to save to.
- AA and BA select the source registers.
- MB chooses a register or a constant operand.
- FS selects an ALU operation.
- MW = 1 to write to memory.
- MD selects between the ALU result and the RAM output.
- V, C, N and Z are status bits.



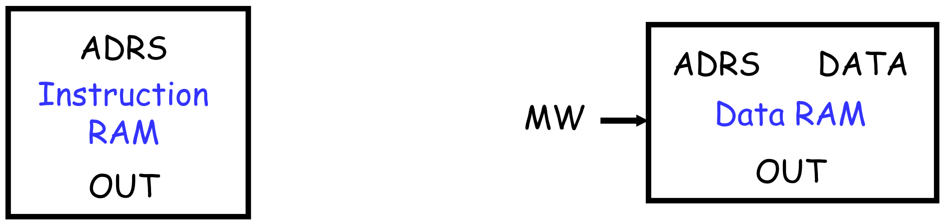
Block diagram of a processor



- The control unit connects programs with the datapath.
  - It converts program instructions into control words for the datapath, including signals WR, DA, AA, BA, MB, FS, MW, MD.
  - It executes program instructions in the correct sequence.
  - It generates the "constant" input for the datapath.
- The datapath also sends information back to the control unit. For instance, the ALU status bits V, C, N, Z can be inspected by branch instructions to alter a program's control flow.

Where does the program go?

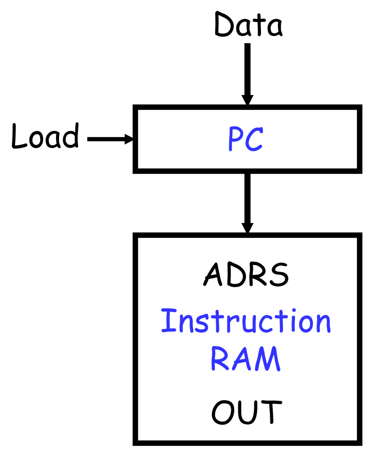
- We'll use a Harvard architecture, which includes two memory units.
  - An instruction memory holds the program.
  - A separate data memory is used for computations.
  - The advantage is that we can read an instruction *and* load or store data in the same clock cycle.
- For simplicity, our diagrams do not show any WR or DATA inputs to the instruction memory.



- Caches in modern CPUs often feature a Harvard architecture like this.
- However, there is usually a single main memory that holds both program instructions and data, in a Von Neumann architecture.

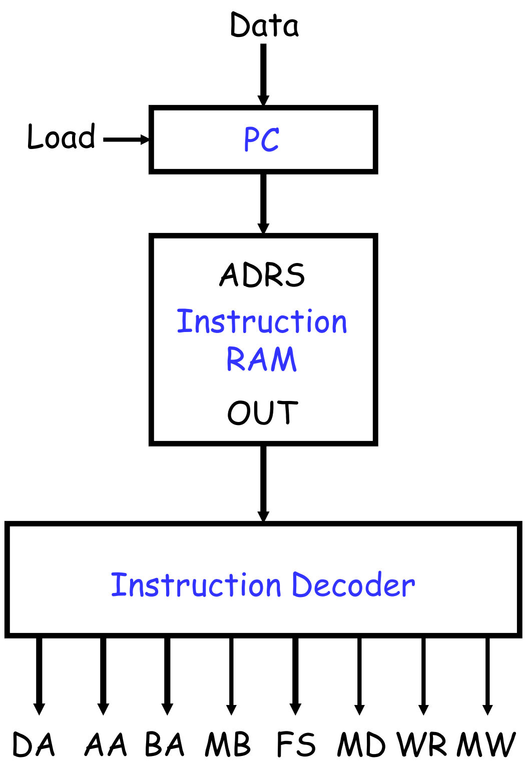
Program counter

- A program counter or PC addresses the instruction memory, to keep track of the instruction currently being executed.
- On each clock cycle, the counter does one of two things.
  - If Load = 0, the PC increments, so the next instruction in memory will be executed.
  - If Load = 1, the PC is updated with Data, which represents some address specified in a jump or branch instruction.



Instruction decoder

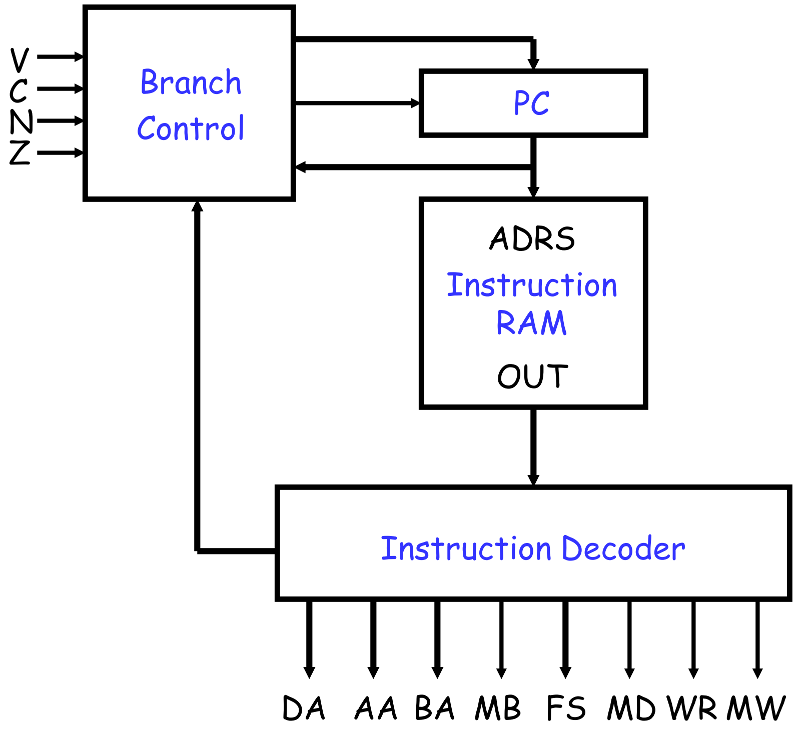
- The instruction decoder is a combinational circuit that takes a machine language instruction and produces the matching control signals for the datapath.
- These signals tell the datapath which registers or memory locations to access, and what ALU operations to perform.



(to the datapath)

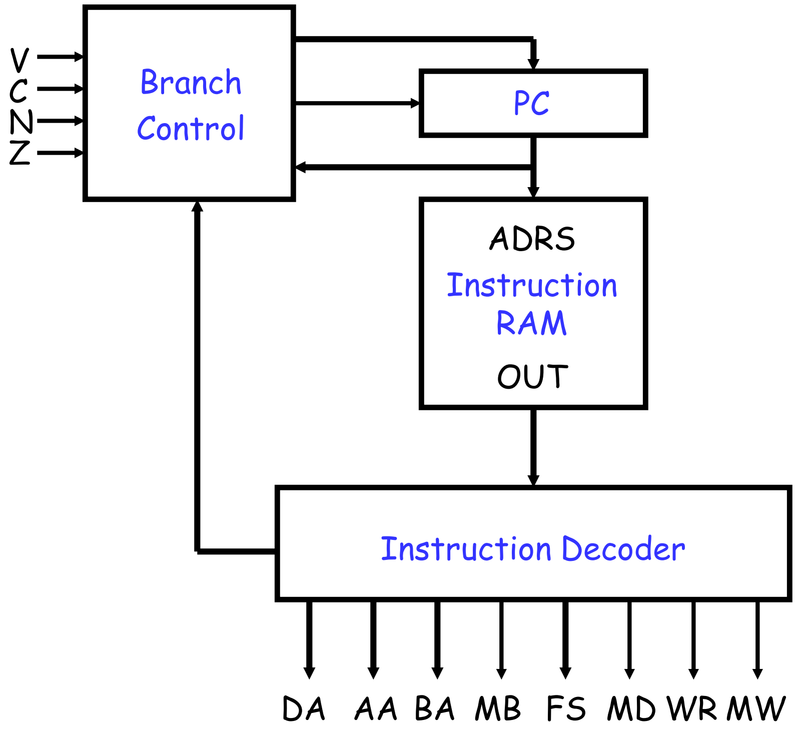
Jumps and branches

- Finally, the branch control unit decides what the PC's next value should be.
  - For jumps, the PC should be loaded with the target address specified in the instruction.
  - For branch instructions, the PC should be loaded with the target address only if the corresponding status bit is true.
  - For all other instructions, the PC should just increment.

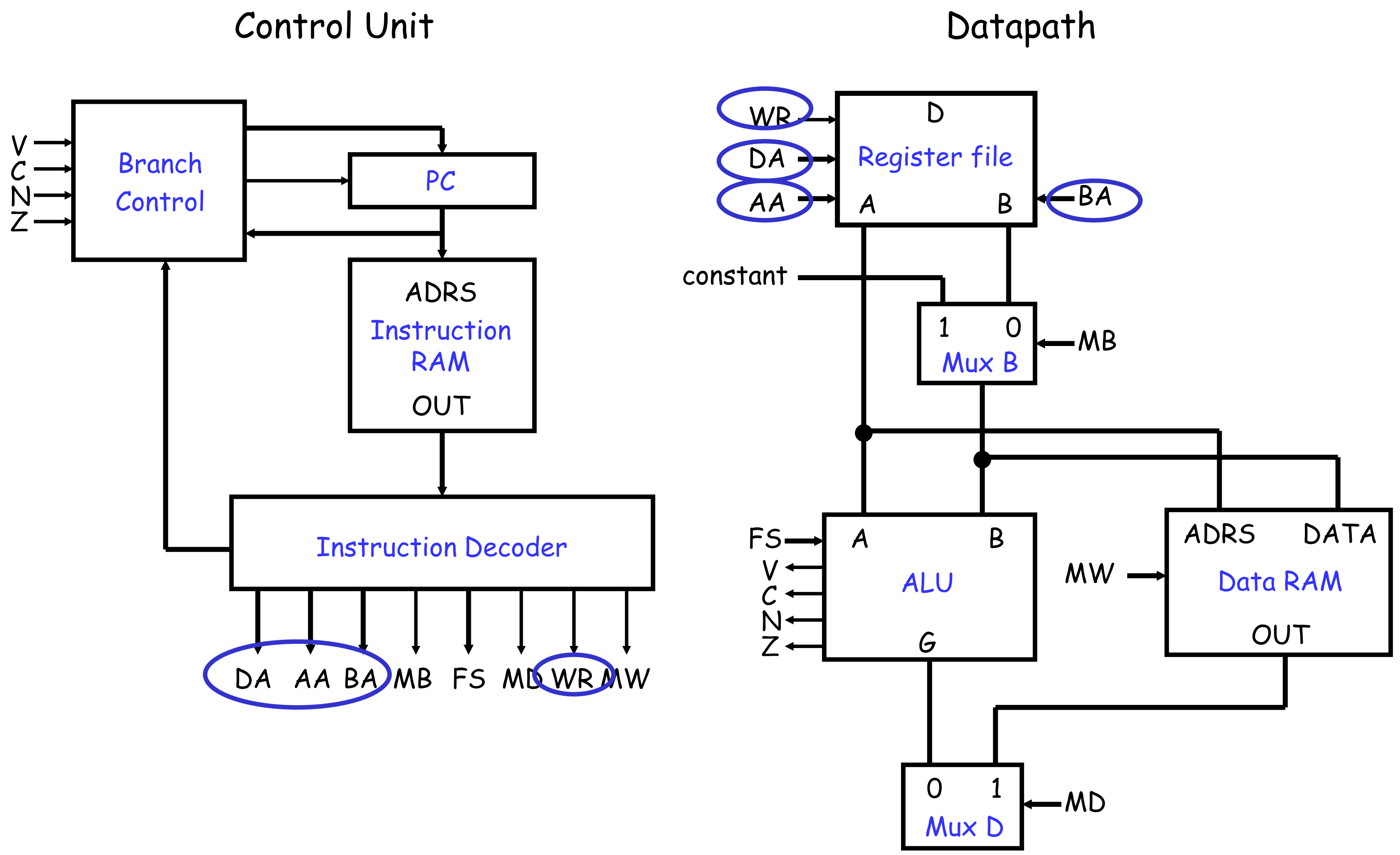


That’s it!

- This is the basic control unit. On each clock cycle:
  1. An instruction is read from the instruction memory.
  2. The instruction decoder generates the matching datapath control word.
  3. Datapath registers are read and sent to the ALU or the data memory.
  4. ALU or RAM outputs are written back to the register file.
  5. The PC is incremented, or reloaded for branches and jumps.



The whole processor

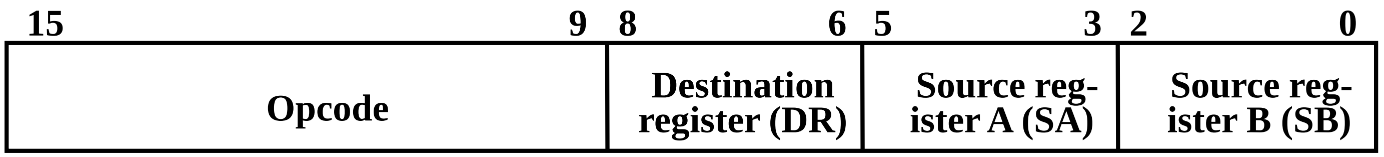


Instruction format

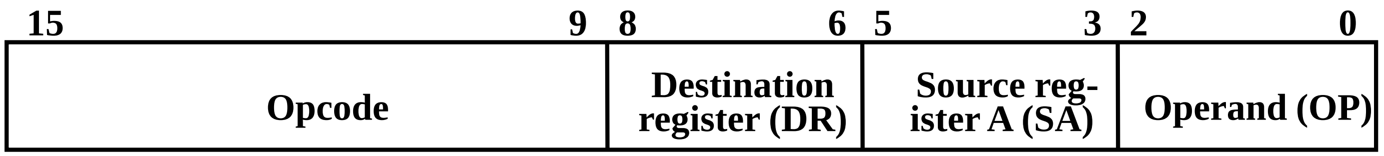
- We have three different instruction formats, each 16 bits long with a seven-bit opcode and nine bits for source registers or constants.
- The first three bits of the opcode determine the instruction category, while the other four bits indicate the exact instruction.
  - For ALU/shift instructions, the four bits choose an ALU operation.
  - For branches, the bits select one of eight branch conditions.
  - We only support one load, one store, and one jump instruction.

15 9	8 6	5 3	2 0
Opcode	Destination (DR) or Address 5-3 (AD)	Register A (SA)	Register B (SB), Operand (OP), or Address 2-0 (AD)

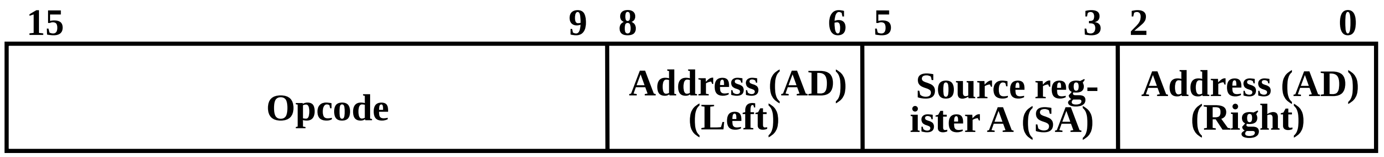
Instruction Formats



(a) Register



(b) Immediate



(c) Jump and Branch

- The three formats are: Register, Immediate, and Jump and Branch
- All formats contain an Opcode field in bits 9 through 15.
- The Opcode specifies the operation to be performed

Register format

15 9	8 6	5 3	2 0
Opcode	Destination Register (DR)	Source Register A (SA)	Source Register B (SB)

- An example register-format instruction:

```
ADD R1, R2, R3
```

- Our binary representation for these instructions will include:
  - A 7-bit opcode field, specifying the operation (e.g., ADD).
  - A 3-bit destination register, DR.
  - Two 3-bit source registers, SA and SB.

Immediate format

15 9	8 6	5 3	2 0
Opcode	Destination Register (DR)	Source Register A (SA)	Operand (OP)

- An example immediate-format instruction:

  ADD R1, R2, #3

- Immediate-format instructions will consist of:
  - A 7-bit instruction opcode.
  - A 3-bit destination register, DR.
  - A 3-bit source register, SA.
  - A 3-bit constant operand, OP.

Jump and branch format

15 9	8 6	5 3	2 0
Opcode	Address Bits 5-3 (AD)	Source Register A (SA)	Address Bits 2-0 (AD)

- Two example jump and branch instructions:

```
BZ   R3, -24
JMP  18
```

- Jump and branch format instructions include:
  - A 7-bit instruction opcode.
  - A 3-bit source register SA for branch conditions.
  - A 6-bit address field, AD, for storing jump or branch offsets.
- Our branch instructions support only one source register. Other types of branches can be simulated from these basic ones.

Assembly → machine language

- we defined a machine language, or a binary representation of the assembly instructions that our processor supports.
- Our CPU includes three types of instructions, which have different operands and will need different representations.
  - Register format instructions require two source registers.
  - Immediate format instructions have one source register and one constant operand.
  - Jump and branch format instructions need one source register and one constant address.
- Even though there are three different instruction formats, it is best to make their binary representations as similar as possible.
  - This will make the control unit hardware simpler.
  - For simplicity, all of our instructions are 16 bits long.

Table 10-8

Instruction Specifications for the Simple Computer - Part 1

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD,RA	$R[DR] \leftarrow R[SA]$	N, Z
Increment	0000001	INC	RD,RA	$R[DR] \leftarrow R[SA] + 1$	N, Z
Add	0000010	ADD	RD,RA,RB	$R[DR] \leftarrow R[SA] + R[SB]$	N, Z
Subtract	0000101	SUB	RD,RA,RB	$R[DR] \leftarrow R[SA] - R[SB]$	N, Z
Decrement	0000110	DEC	RD,RA	$R[DR] \leftarrow R[SA] - 1$	N, Z
AND	0001000	AND	RD,RA,RB	$R[DR] \leftarrow R[SA] \wedge R[SB]$	N, Z
OR	0001001	OR	RD,RA,RB	$R[DR] \leftarrow R[SA] \vee R[SB]$	N, Z
Exclusive OR	0001010	XOR	RD,RA,RB	$R[DR] \leftarrow R[SA] \oplus R[SB]$	N, Z
NOT	0001011	NOT	RD,RA	$R[DR] \leftarrow \overline{R[SA]}$	N, Z

Summary

- We saw an outline of the control unit hardware.
  - The program counter points into a special instruction memory, which contains a machine language program.
  - An instruction decoder looks at each instruction and generates the correct control signals for the datapath and a branching unit.
  - The branch control unit handles instruction sequencing.
- The control unit implementation depends on both the instruction set architecture and the datapath.
  - Careful selection of opcodes and instruction formats can make the control unit simpler.
- We now have a whole processor! This is the culmination of everything we did this semester, starting from primitive gates.