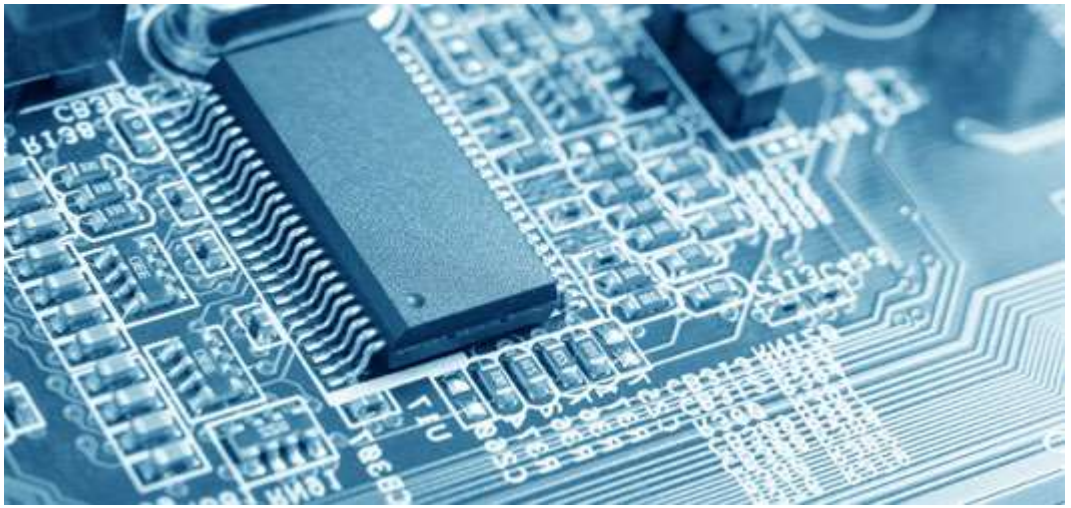
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT206 – Microcontrollers with Lab

Lectures 22-23 – Computer Design Basics: Datapath

12-17 April, 2018

Topics

Today's Topics

- Datapaths
  - Introduction
  - Datapath Example
  - Arithmetic Logic Unit (ALU)
  - Shifter
  - Datapath Representation and Control Word

Keep 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.

- Today we'll look in detail at a processor's datapath, which is responsible for doing all of the dirty work.
  - An ALU does computations,
  - 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.
  - Fundamentally, the processor is just moving data between registers, possibly with some ALU computations.
  - Some registers are used to store values, which form the state.
  - An ALU performs various operations on the data stored in the registers.

Computer Specification

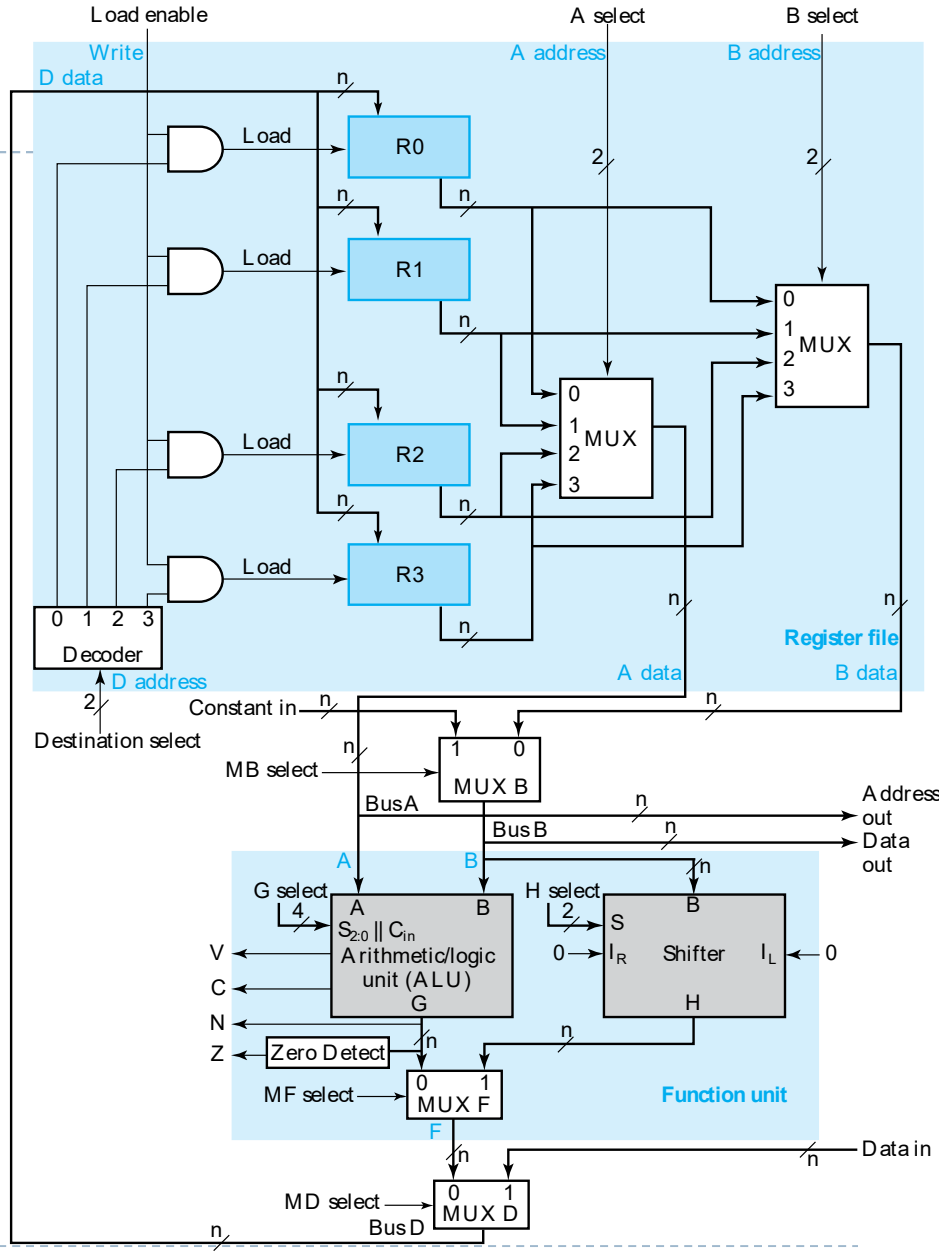
- Computer Specification
  - ***Instruction Set Architecture*** *(ISA)* - the specification of a computer's appearance to a programmer at its lowest level
  - ***Computer Architecture*** - a high-level description of the hardware implementing the computer derived from the ISA
  - The architecture usually includes additional specifications such as speed, cost, and reliability.

Datapaths

- Guiding principles for basic datapaths:
  - The set of registers
    - Collection of individual registers
    - A set of registers with common access resources called a ***register file***
    - A combination of the above
  - Microoperation implementation
    - One or more shared resources for implementing microoperations
    - Buses - shared transfer paths
    - ***Arithmetic-Logic Unit (ALU)*** - shared resource for implementing arithmetic and logic microoperations
    - **Shifter** - shared resource for implementing shift microoperations

Datapath Example

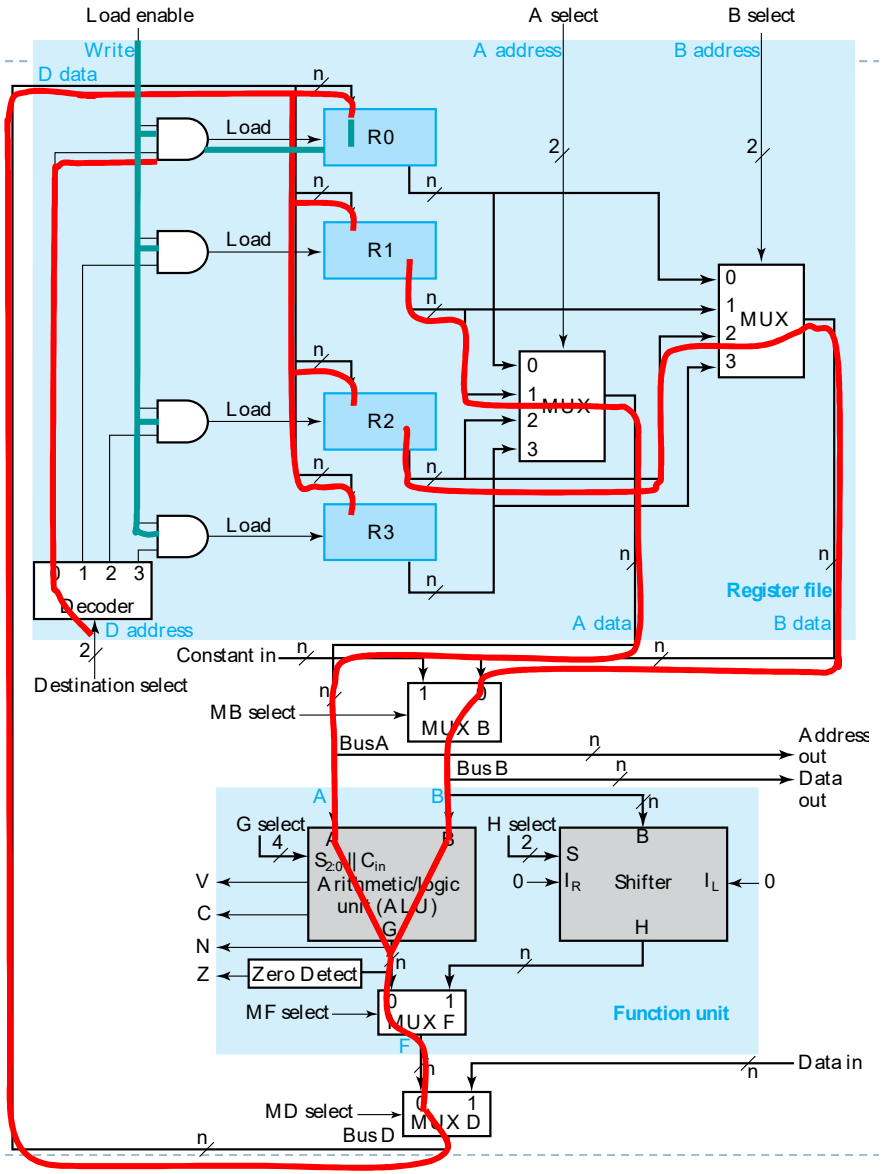
- Four parallel-load registers
- Two mux-based register selectors
- Register destination decoder
- Mux B for external constant input
- Buses A and B with external address and data outputs
- ALU and Shifter with Mux F for output select
- Mux D for external data input
- Logic for generating status bits V, C, N, Z



Datapath Example: Performing a Microoperation

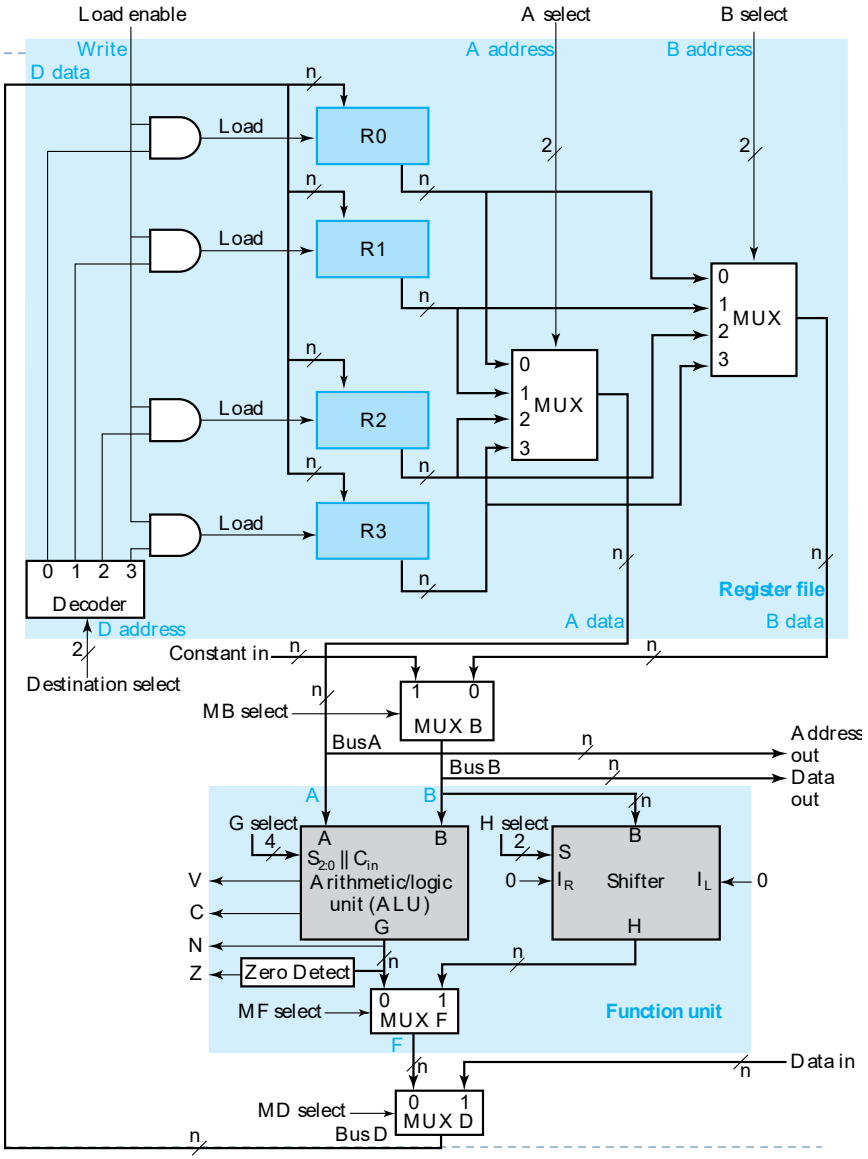
- **Microoperation: R0 ← R1 + R2**

  - Apply 01 to A select to place contents of R1 onto Bus A
  - Apply 10 to B select to place contents of R2 onto B data and apply 0 to MB select to place B data on Bus B
  - Apply 0010 to G select to perform addition G = Bus A + Bus B
  - Apply 0 to MF select and 0 to MD select to place the value of G onto BUS D
  - Apply 00 to Destination select to enable the Load input to R0
  - Apply 1 to Load Enable to force the Load input to R0 to 1 so that R0 is loaded on the clock pulse (not shown)
  - The overall microoperation requires 1 clock cycle



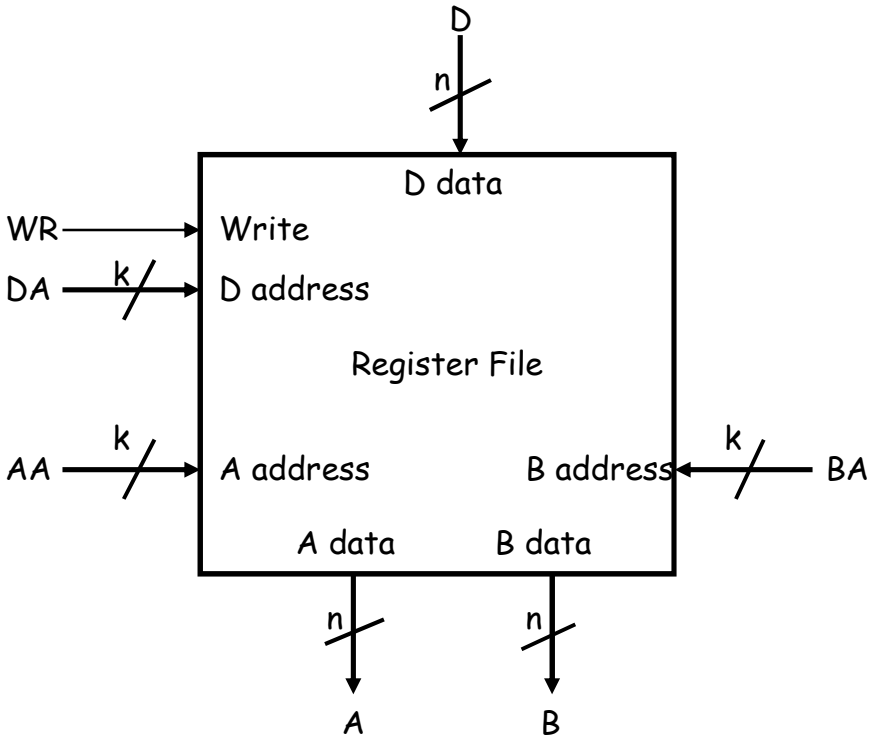
Datapath Example: Key Control Actions for Microoperation Alternatives

- Perform a shift microoperation – apply 1 to MF select
- Use a constant in a micro- operation using Bus B – apply 1 to MB select
- Provide an address and data for a memory or output write microoperation – apply 0 to Load enable to prevent register loading
- Provide an address and obtain data for a memory or output read microoperation – apply 1 to MD select
- For some of the above, other control signals become don't cares



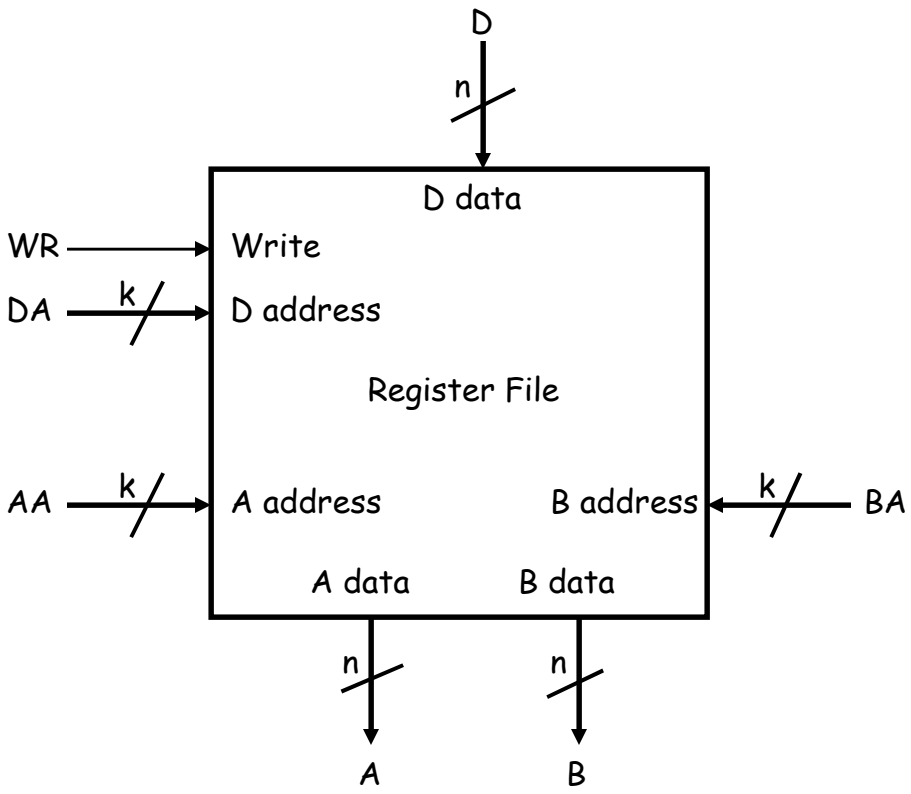
Register Files

- 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.



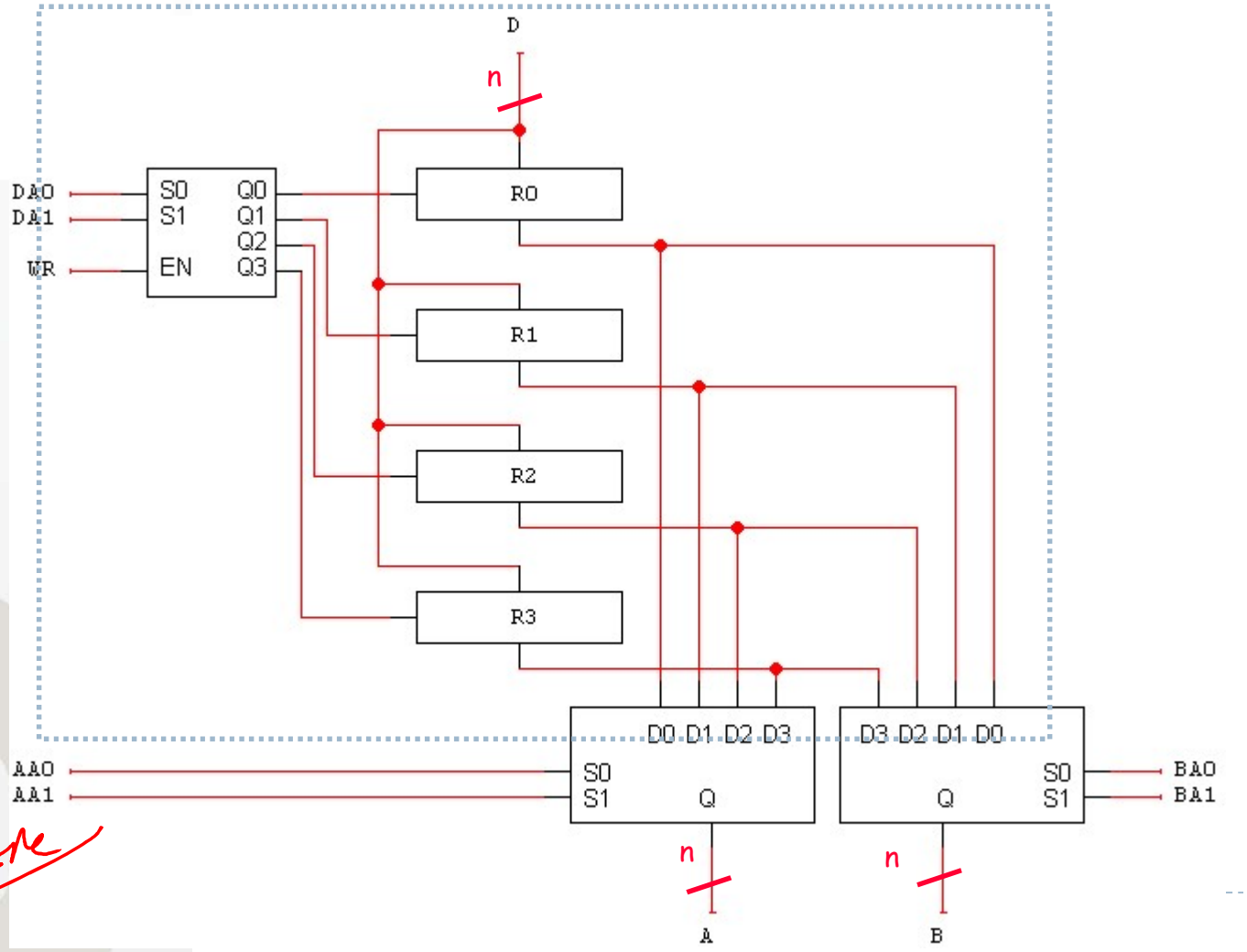
Accessing the Register File

- 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.



up to here

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.

Example

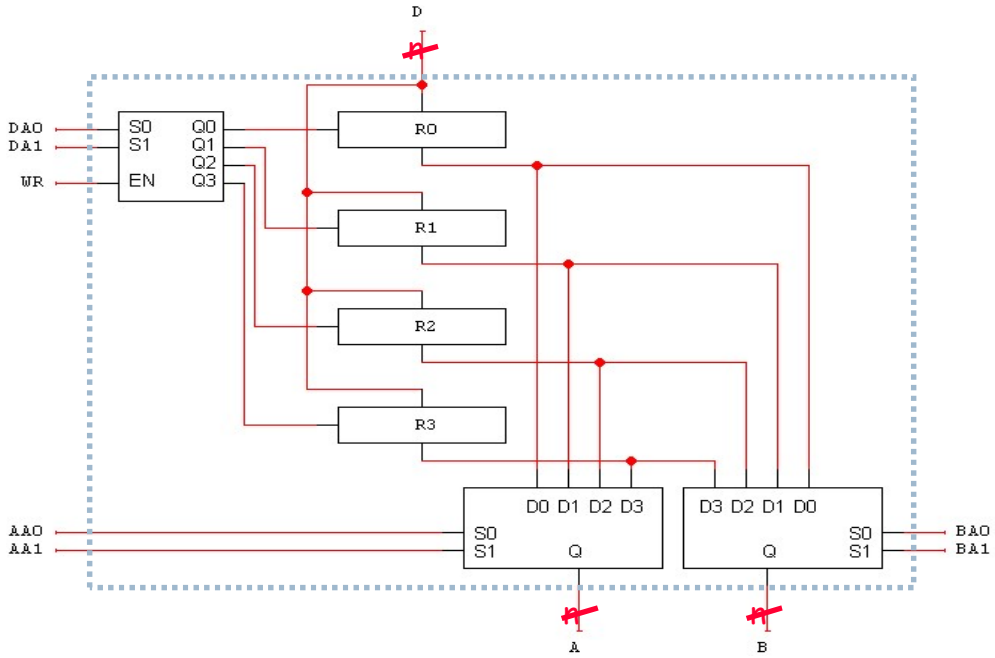
A register file has 16-bit words and 64 registers. We use a decoder for the DA input and multiplexers for the A and B output. The size of the decoder is:

A) 6 to 64

B) 4 to 64

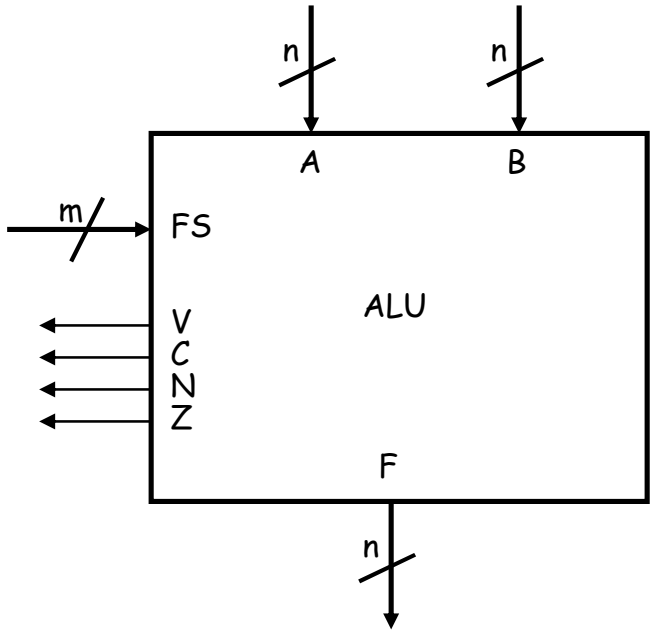
C) No decoder is used

D) 5 to 64



Arithmetic Logic Unit (ALU)

- The main job of a central processing unit is to "process," or to perform computations....
- 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.



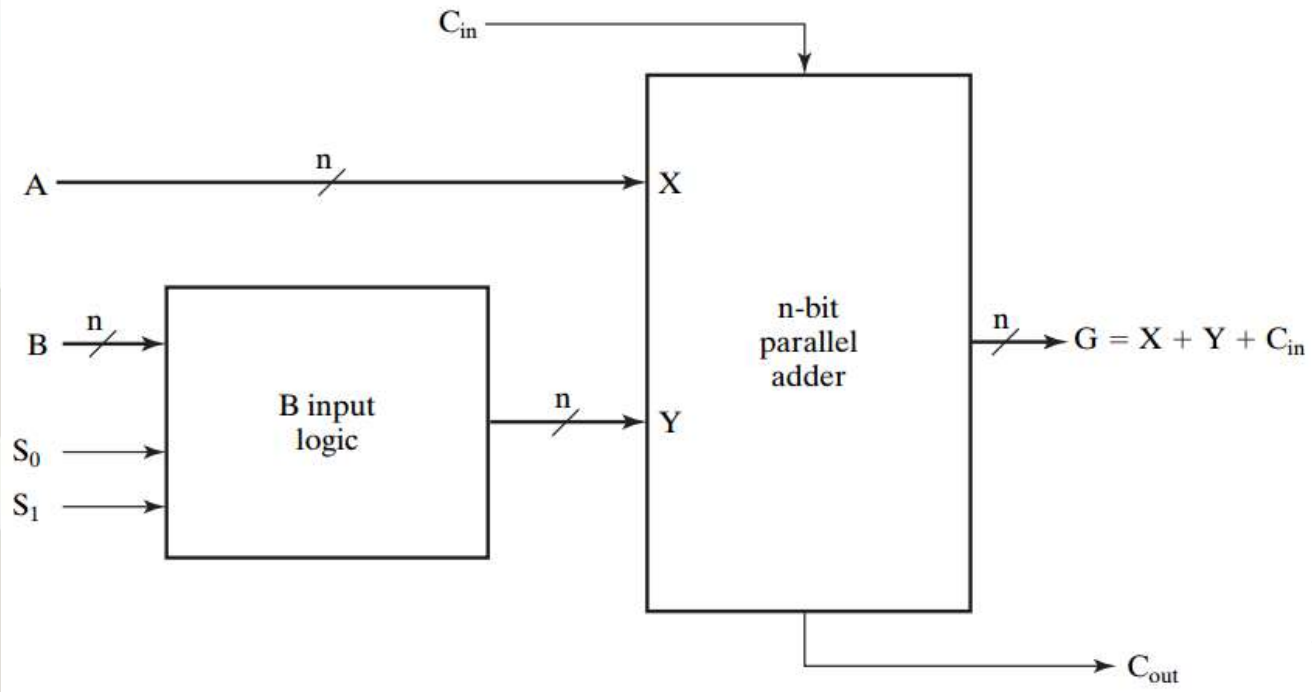
Arithmetic Logic Unit (ALU)

- Consider detailed design of typical ALUs and shifters
- Decompose the ALU into:
  - An arithmetic circuit
  - A logic circuit
  - A selector to pick between the two circuits
- Arithmetic circuit design
  - Decompose the arithmetic circuit into:
    - An n-bit parallel adder
    - A block of logic that selects four choices for the B input to the adder
    - See next slide for diagram

Arithmetic Circuit Design

- There are only four functions of B to select as Y in G = A + Y:

	$C_{in} = 0$	$C_{in} = 1$
0	G = A	G = A + 1
B	G = A + B	G = A + B + 1
$\overline{B}$	G = A + $\overline{B}$	G = A + $\overline{B}$ + 1
1	G = A – 1	G = A



Arithmetic Circuit Design

- Adding selection codes to the functions of B:

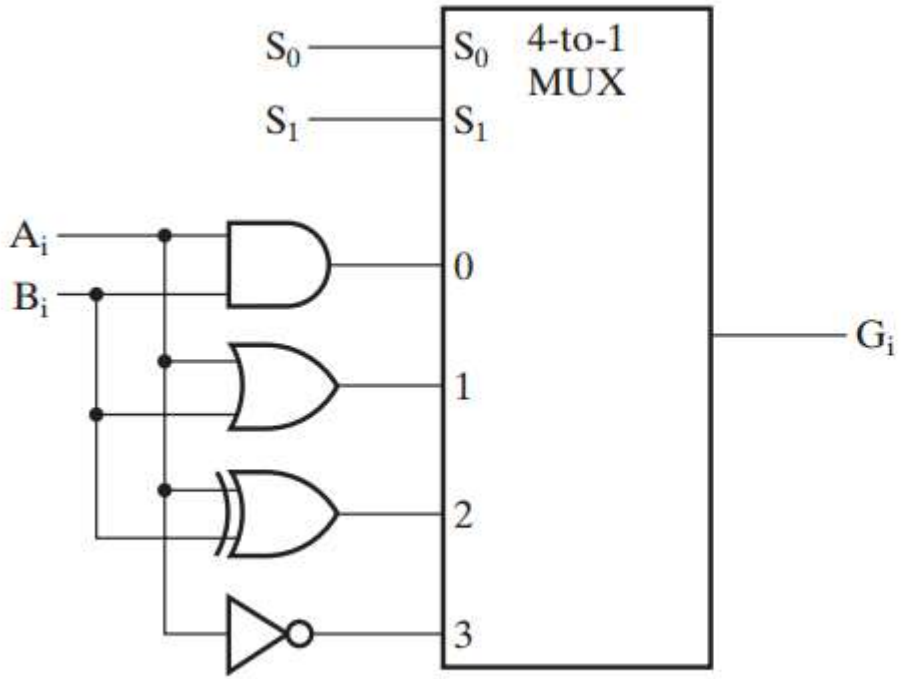
Function Table for Arithmetic Circuit

Select		Input	$G = (A \oplus Y \oplus C_{in})$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

- The useful arithmetic functions are labeled in the table
- Note that all four functions of B produce at least one useful function

Logic Circuit

- the logic circuit is implemented using a multiplexer plus gates implementing: AND, OR, XOR and NOT



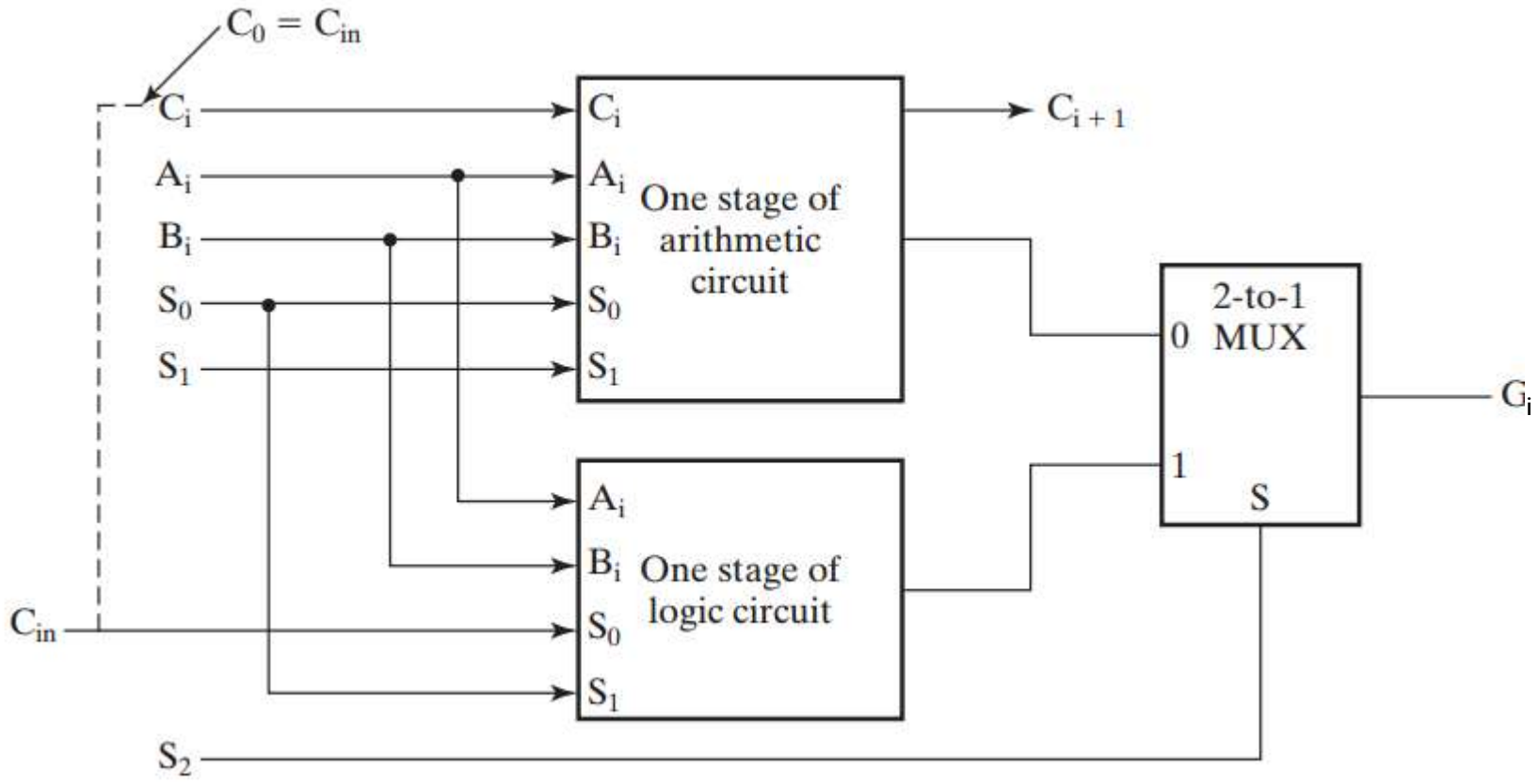
(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

(b) Function table

Arithmetic Logic Unit (ALU)

- The arithmetic circuit, the logic circuit, and a 2-way multiplexer are combined to form the ALU.



Arithmetic Logic Unit (ALU)

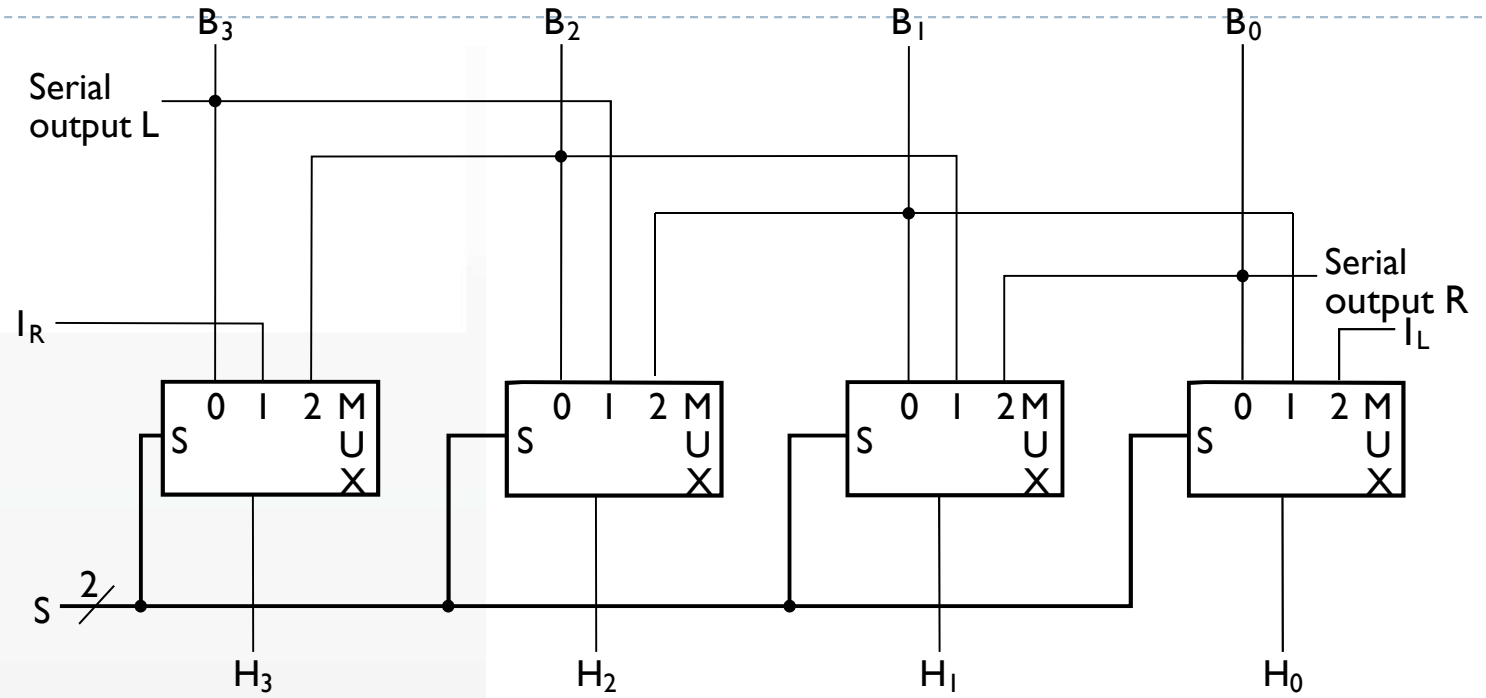
Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

Combinational Shifter Parameters

- Direction: Left, Right
- Number of positions with examples:
  - Single bit:
    - 1 position
    - 0 and 1 positions
  - Multiple bit:
    - 1 to n – 1 positions
    - 0 to n – 1 positions
- Filling of vacant positions
  - Many options depending on instruction set
  - Here, will provide input lines or zero fill

4-Bit Basic Left/Right Shifter

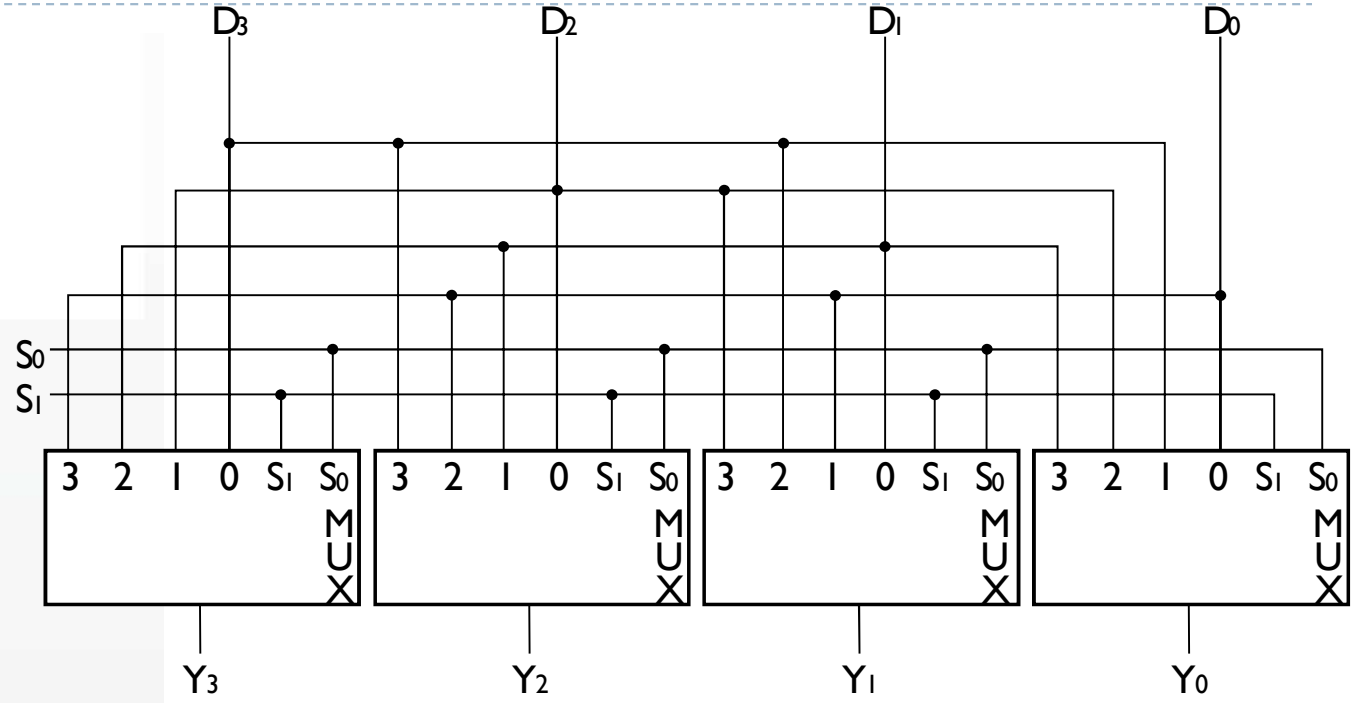


- Serial Inputs:
  - I_R for right shift
  - I_L for left shift
- Serial Outputs
  - R for right shift (Same as MSB input)
  - L for left shift (Same as LSB input)

- Shift Functions:
  (S_1, S_0) = 00 Pass B unchanged
  01 Right shift
  10 Left shift
  11 Unused

Barrel Shifter

- A rotate is a shift in which the bits shifted out are inserted into the positions vacated
- The circuit rotates its contents left from 0 to 3 positions depending on S:

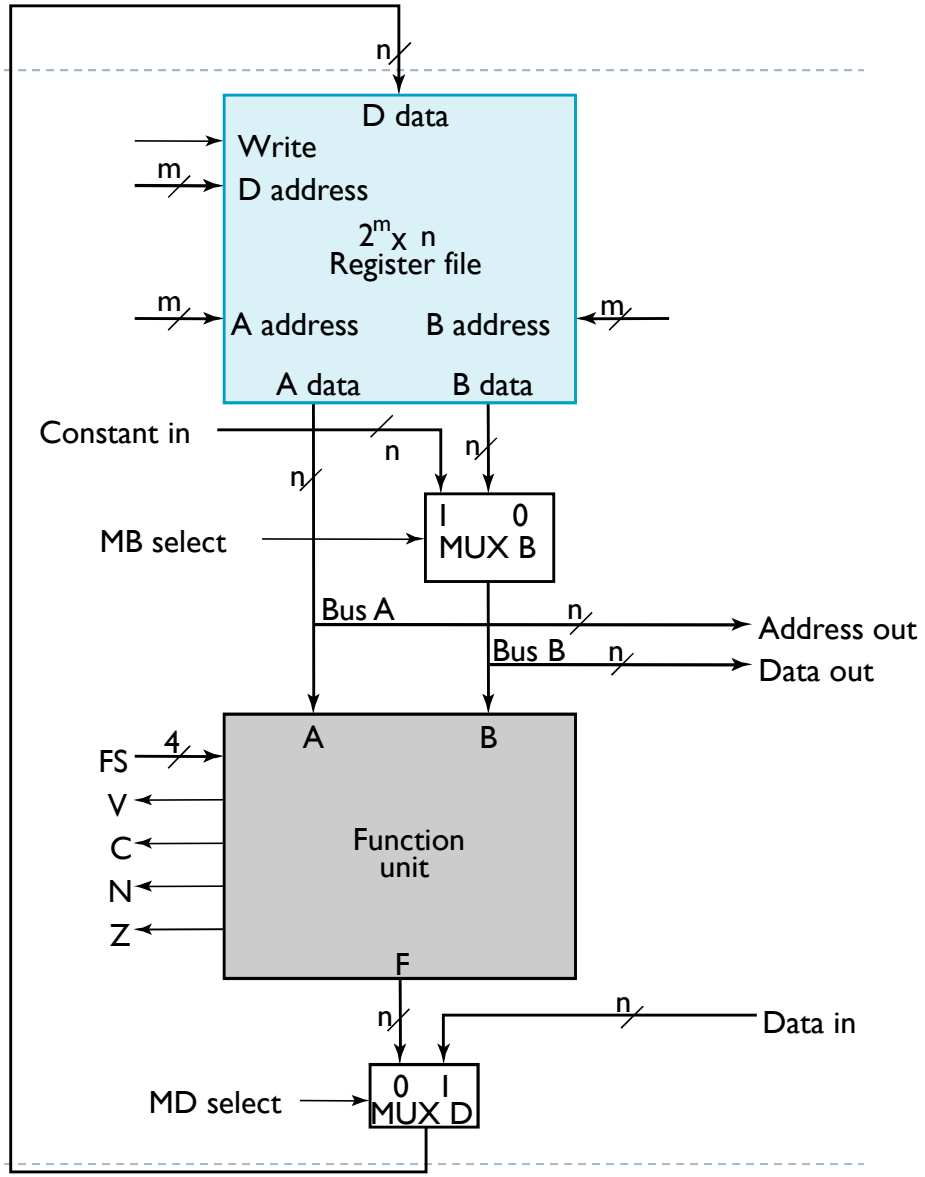


Function Table for 4-Bit Barrel Shifter

Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

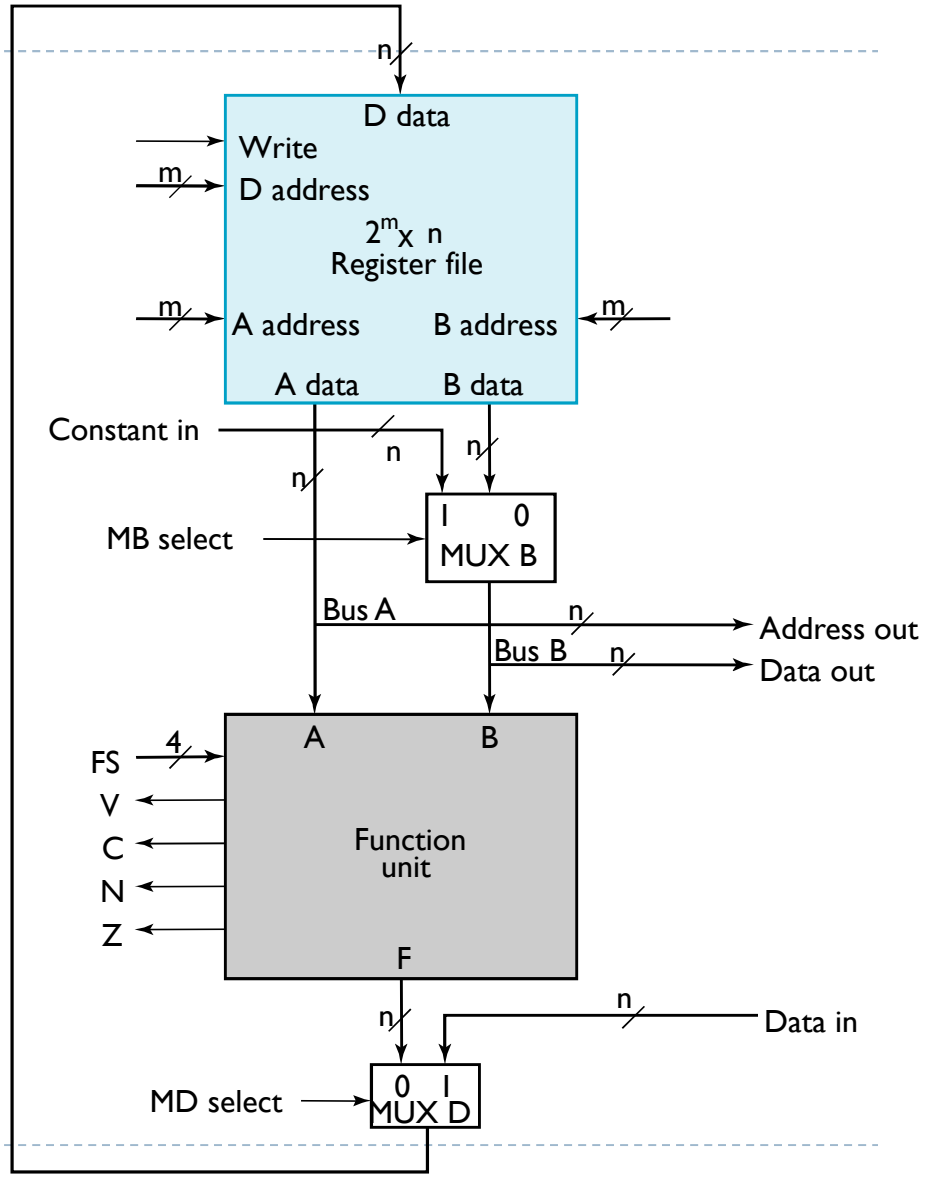
Datapath Representation

- Have looked at detailed design of ALU and shifter in the datapath
- Here we move up one level in the hierarchy from that datapath
- The registers, and the multiplexer, decoder, and enable hardware for accessing them become a *register file*
- The ALU, shifter, Mux F and status hardware become a *function unit*
- The remaining muxes and buses which handle data transfers are at the new level of the hierarchy



Datapath Representation

- In the register file:
  - Multiplexer select inputs become A address and B address
  - Decoder input becomes D address
  - Multiplexer outputs become A data and B data
  - Input data to the registers becomes D data
  - Load enable becomes write
- The register file now appears like a memory based on clocked flip-flops (the clock is not shown)
- The function unit labeling is quite straightforward except for FS

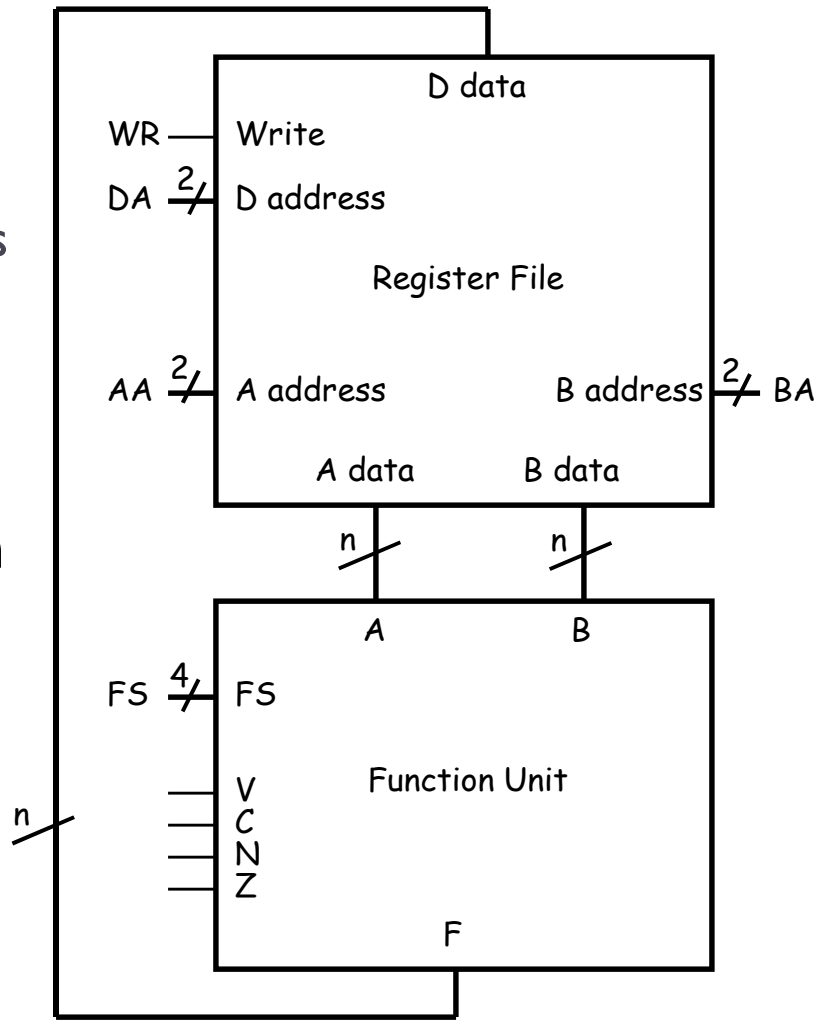


Definition of Function Unit Select (FS) Codes

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Micr ooperation
0000	0	0000	XX	$F \leftarrow A$
0001	0	0001	XX	$F \leftarrow A + 1$
0010	0	0010	XX	$F \leftarrow A + B$
0011	0	0011	XX	$F \leftarrow A + \overline{B} + 1$
0100	0	0100	XX	$F \leftarrow A + \overline{B}$
0101	0	0101	XX	$F \leftarrow A + \overline{B} + 1$
0110	0	0110	XX	$F \leftarrow A - 1$
0111	0	0111	XX	$F \leftarrow A$
1000	0	1 X00	XX	$F \leftarrow A \wedge B$
1001	0	1 X01	XX	$F \leftarrow A \vee B$
1010	0	1 X10	XX	$F \leftarrow A \oplus B$
1011	0	1 X11	XX	$F \leftarrow \overline{A}$
1100	1	XXXX	00	$F \leftarrow B$
1101	1	XXXX	01	$F \leftarrow \text{sr } B$
1110	1	XXXX	10	$F \leftarrow \text{sl } B$

An example computation

- 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.

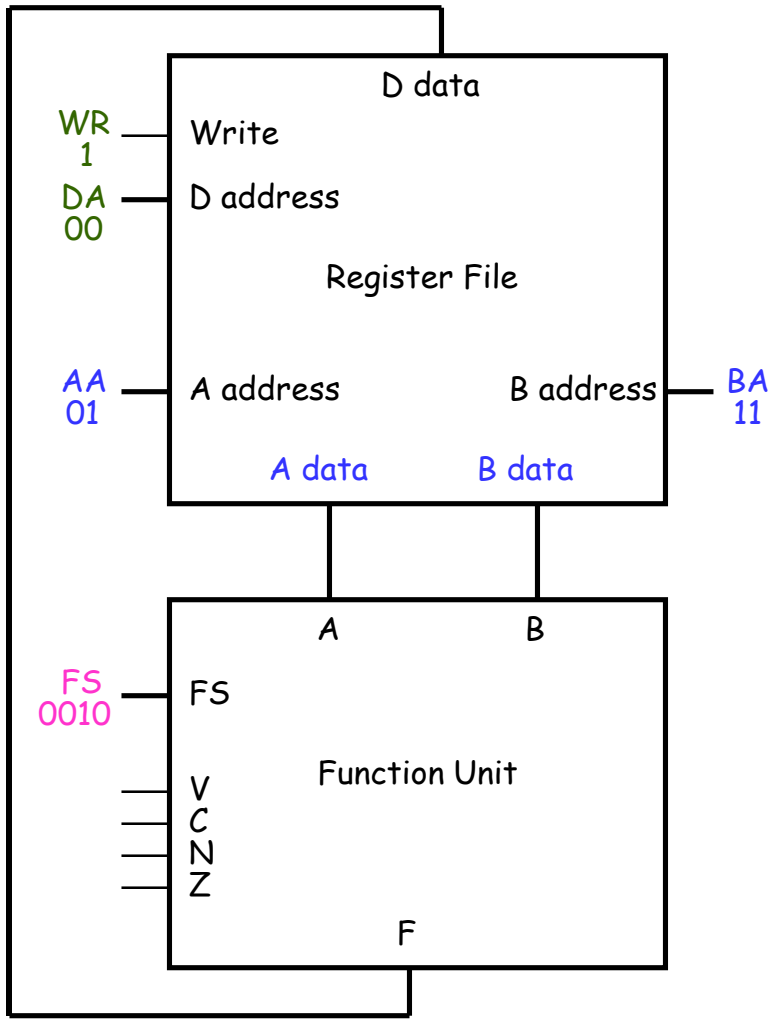


An Example Computation

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

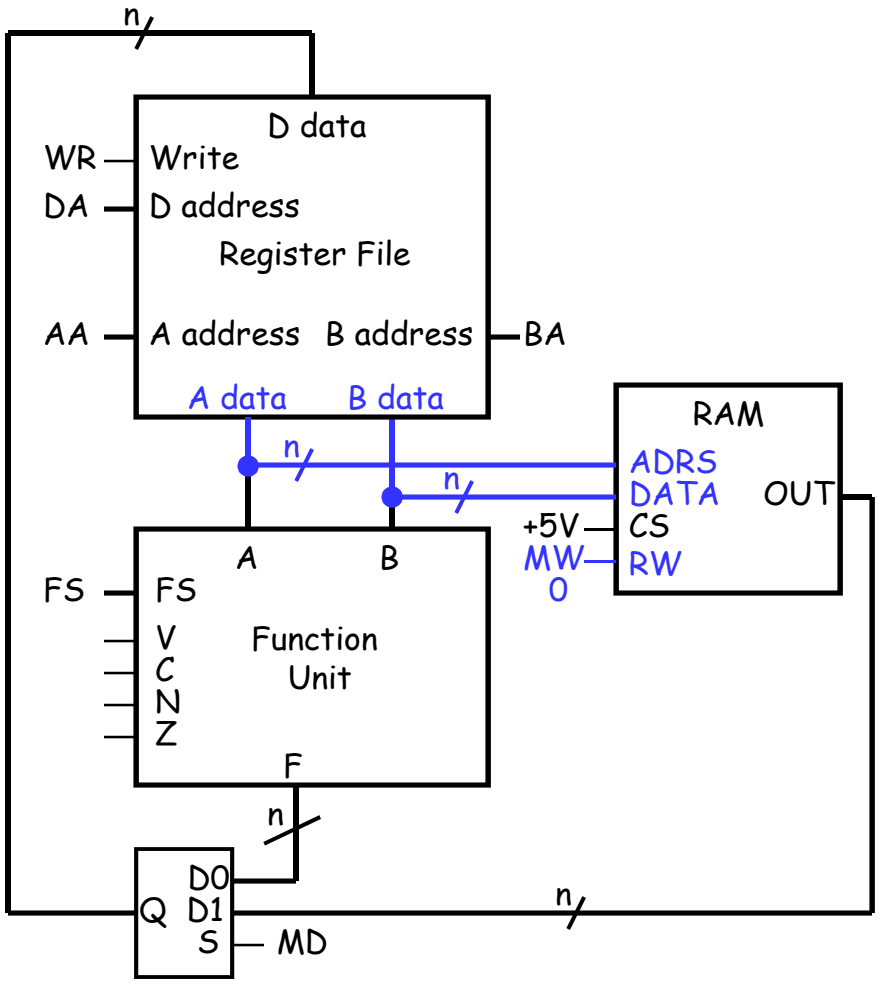
$$R0 \leftarrow R1 + R3$$

- 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 = 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.



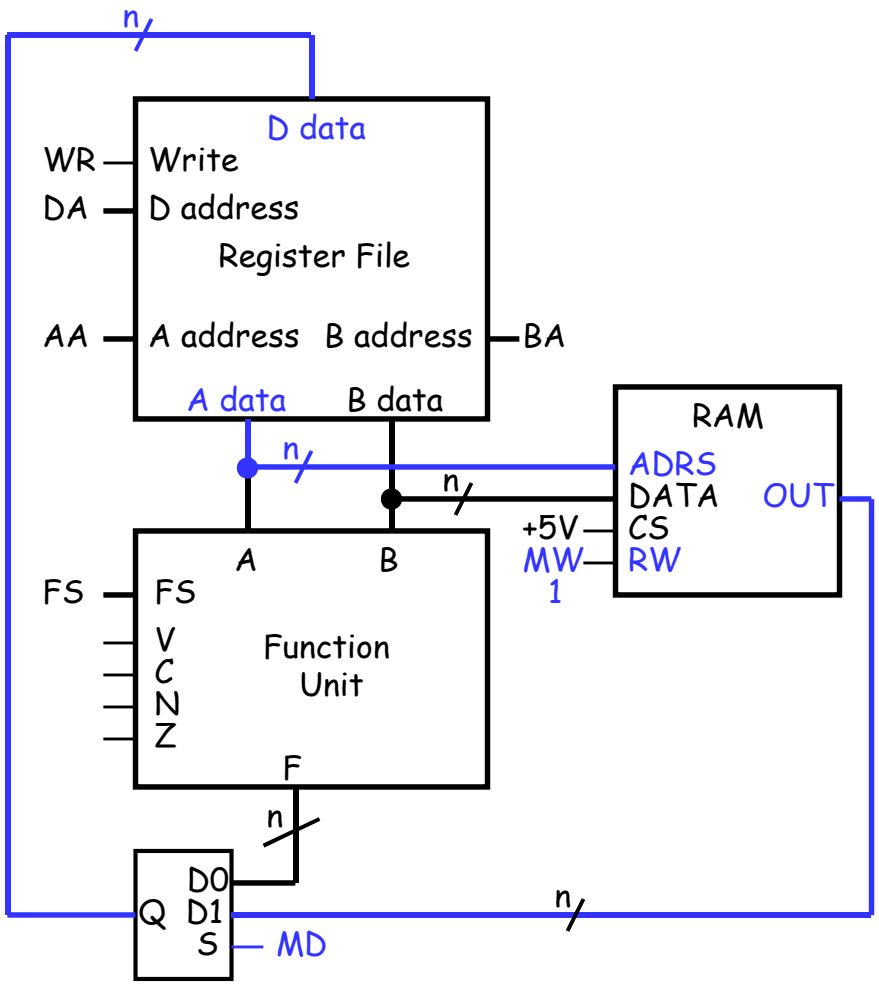
Connecting RAM

- 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.)



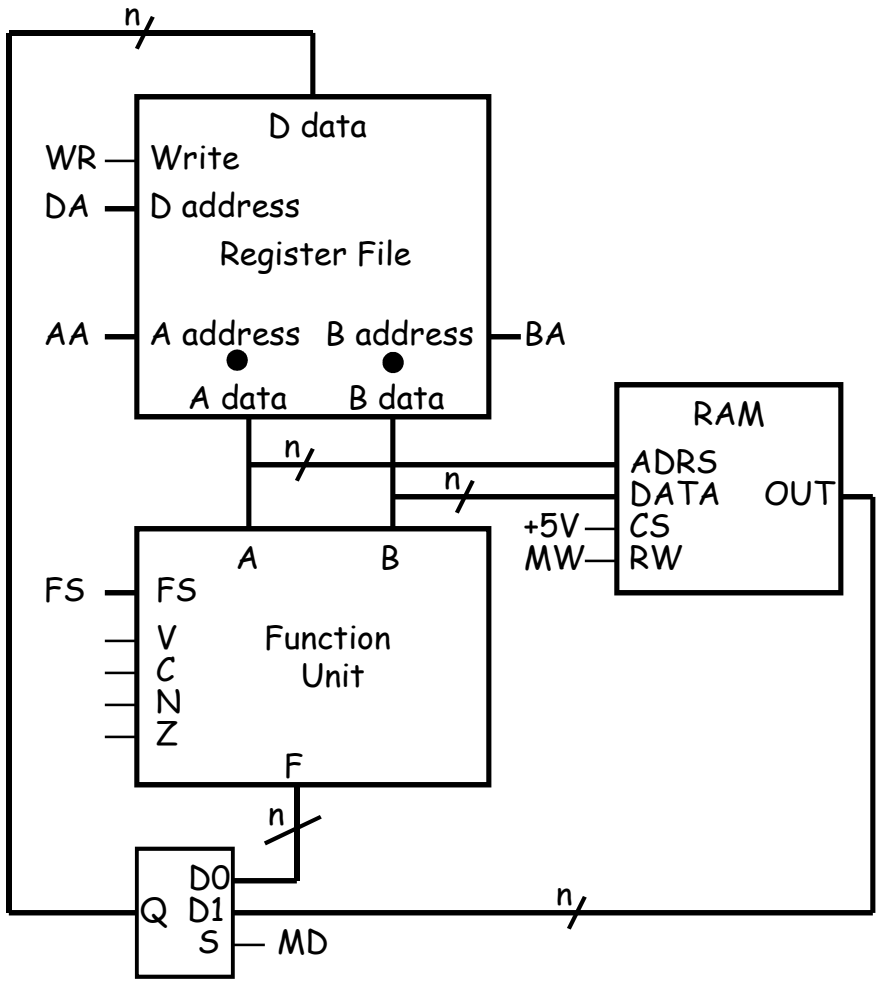
Reading from RAM

- 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 x 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

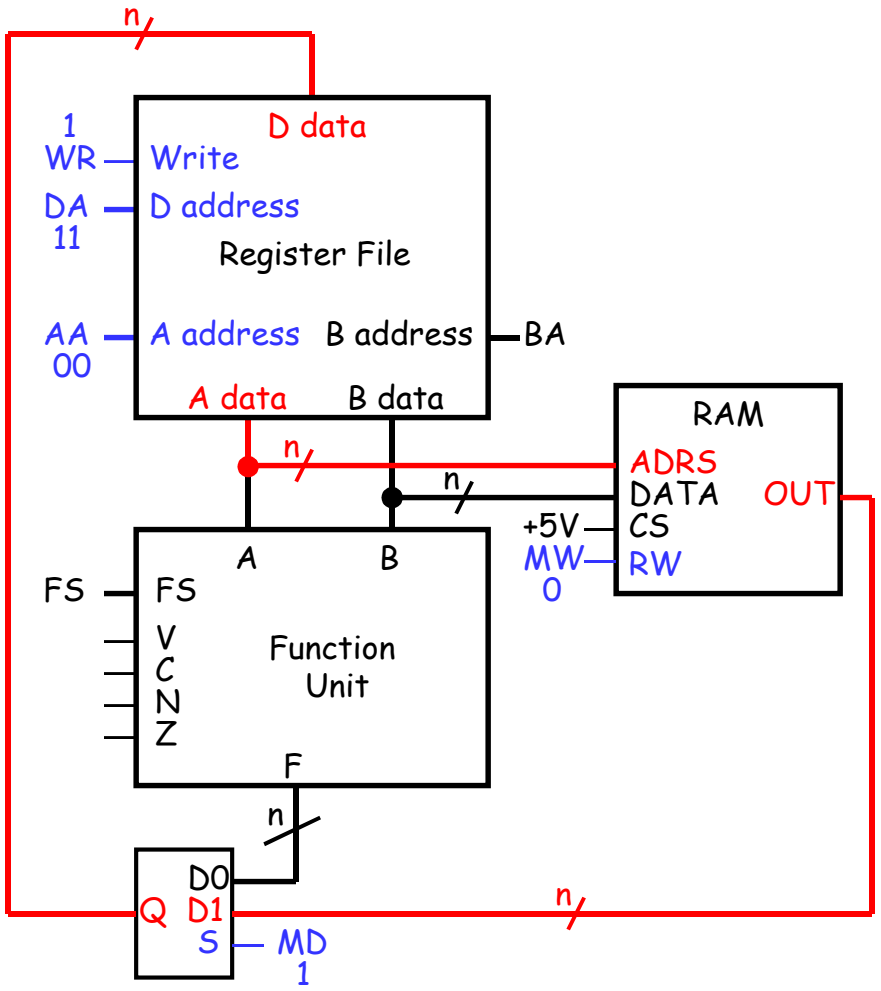
- Here is a simple series of register transfer instructions:

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

- 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?

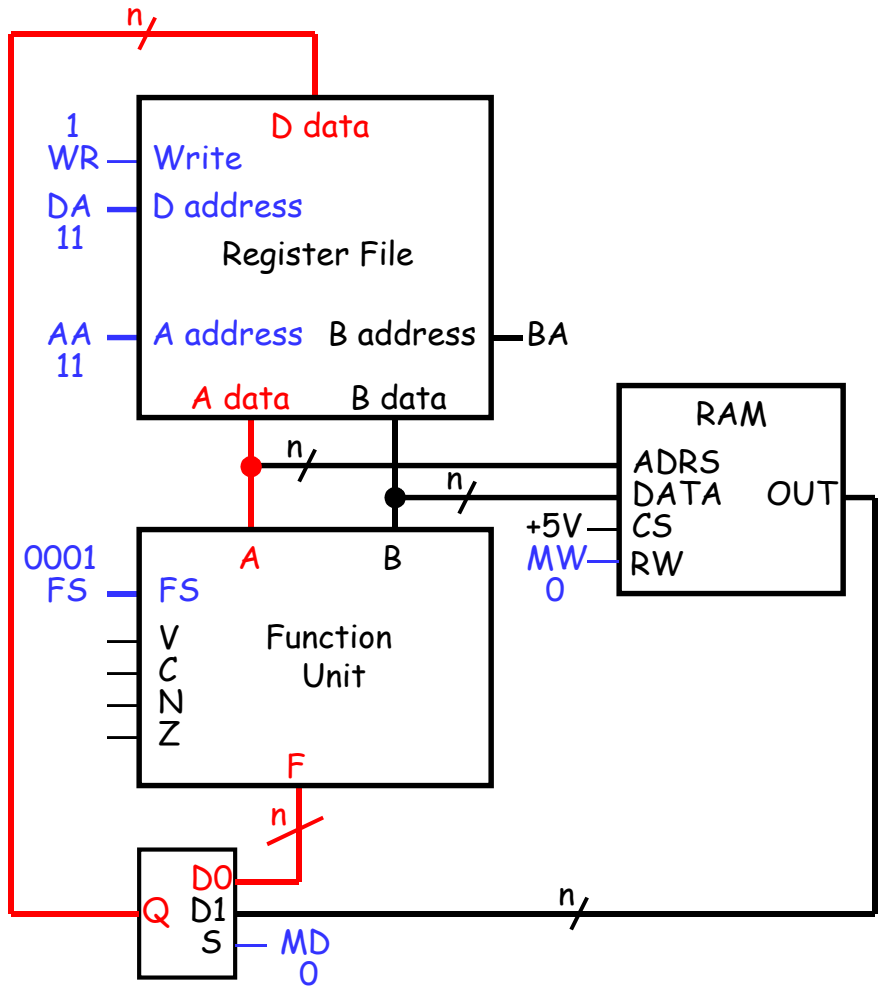
R3 ← M[R0]

- 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).



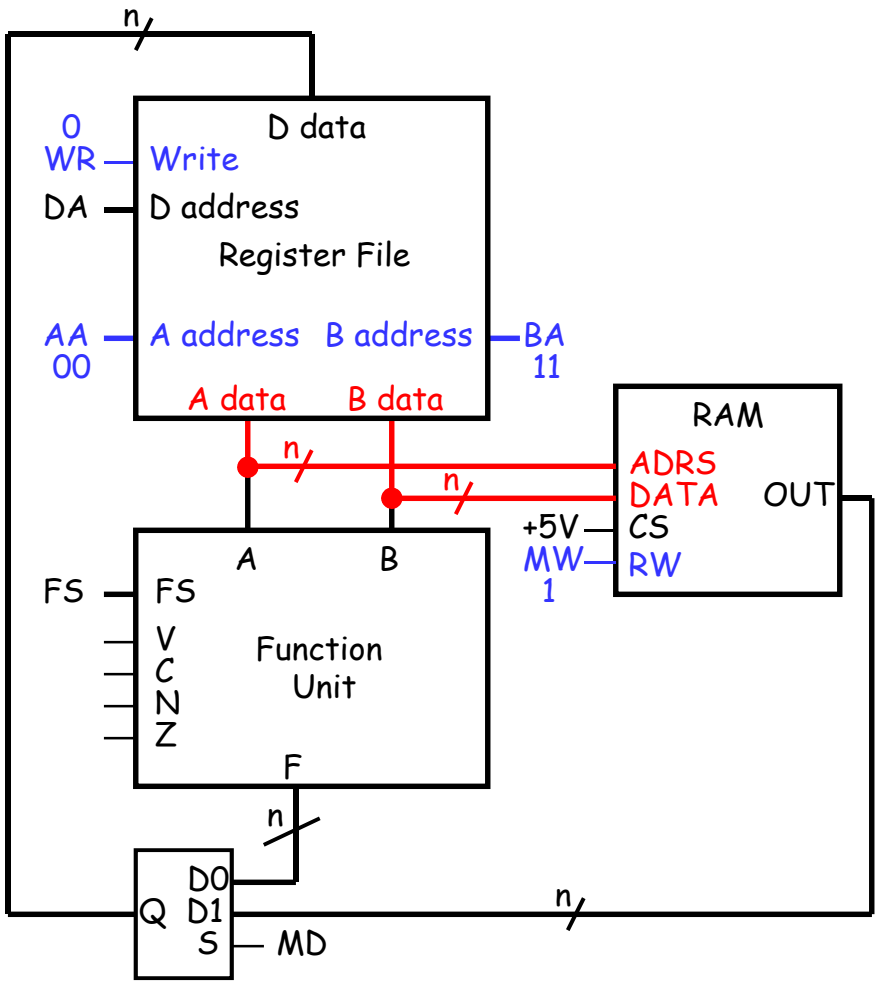
R3 ← R3 + 1

- AA = 11, so R3 is read from the register file and sent to the ALU's A input.
- FS needs to be 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 = 0.
- Again, MW should be 0 so the RAM isn't inadvertently changed.
- We didn't use BA.



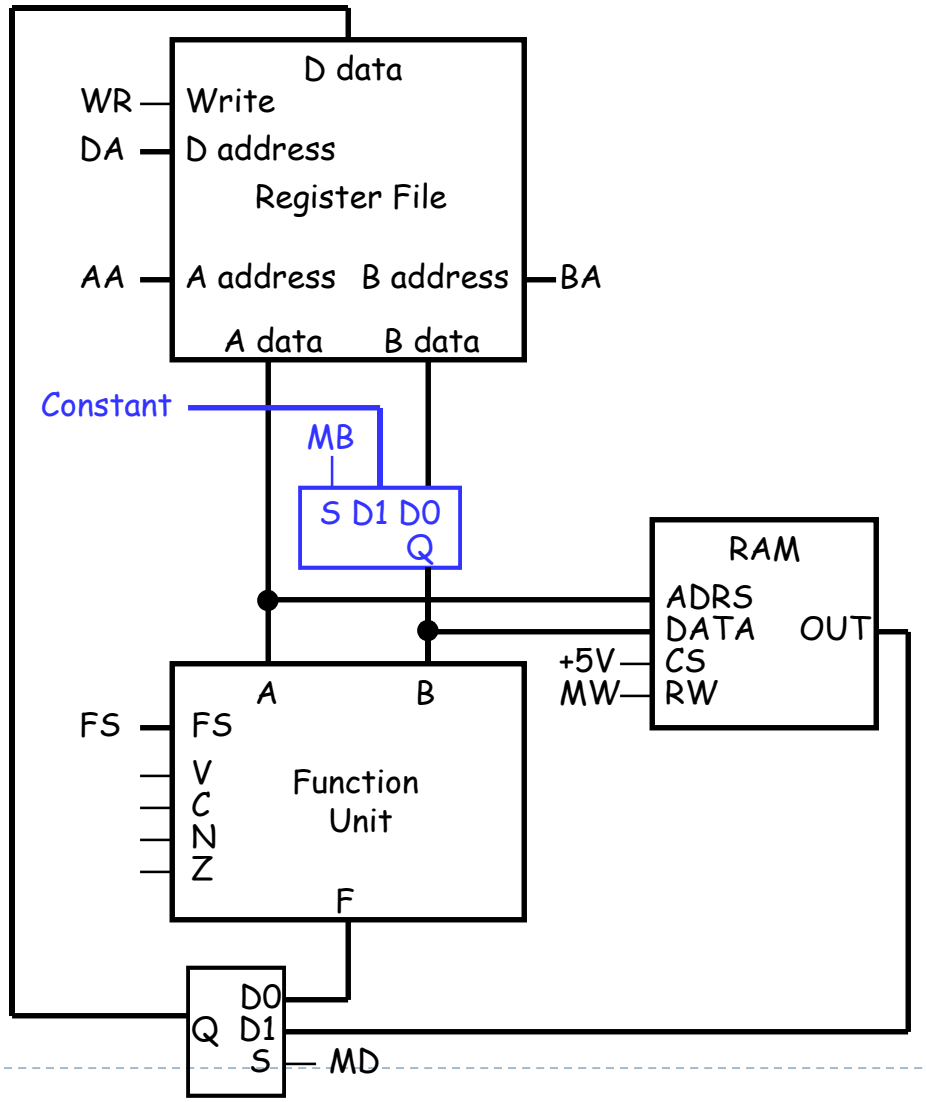
M[R0] ← R3

- 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

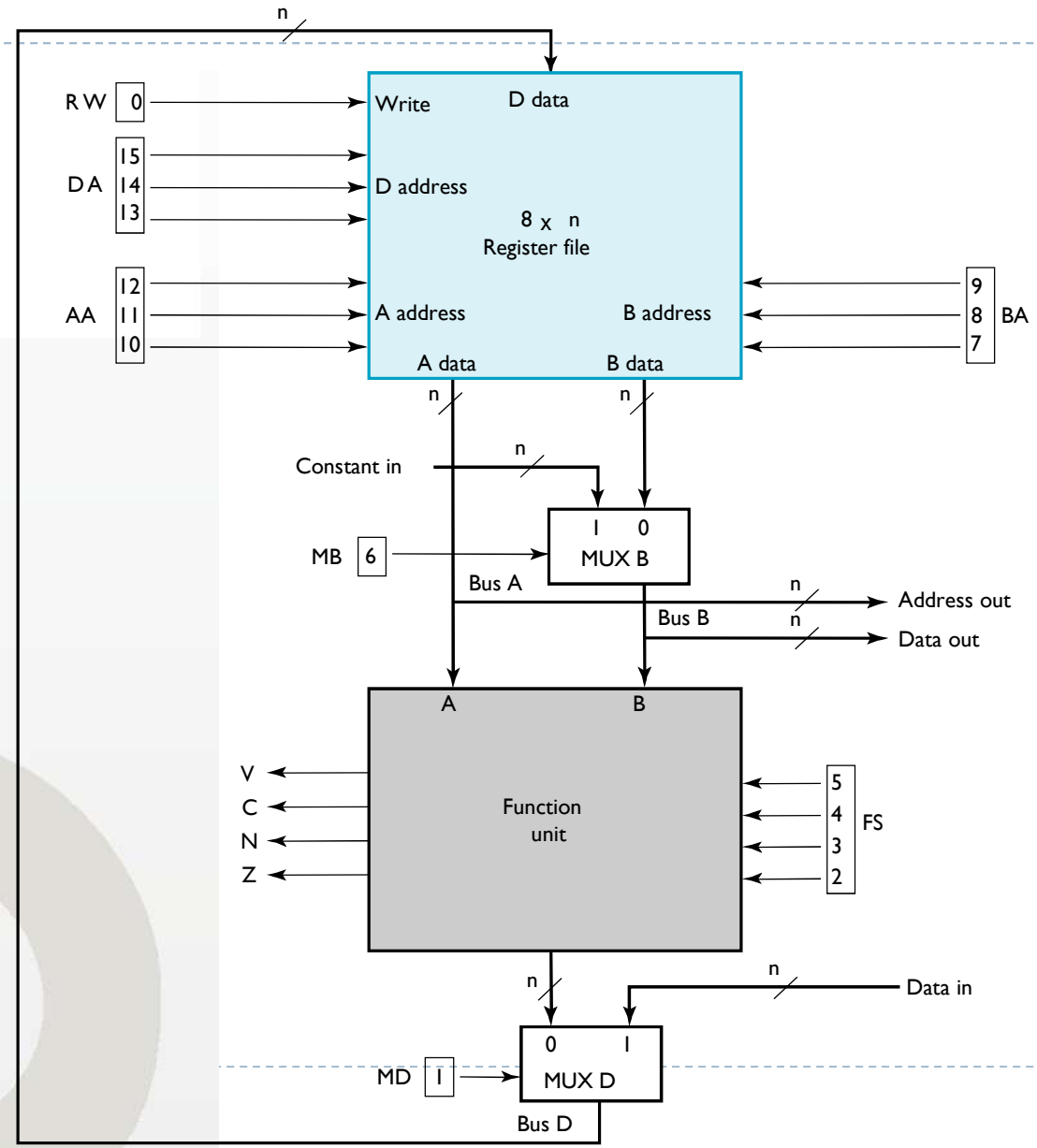
- One last refinement is the addition of a Constant input.
- The modified datapath is shown on the right, with one extra control signal MB.
- We'll see how this is used later. Intuitively, it provides an easy way to initialize a register or memory location with some arbitrary number.



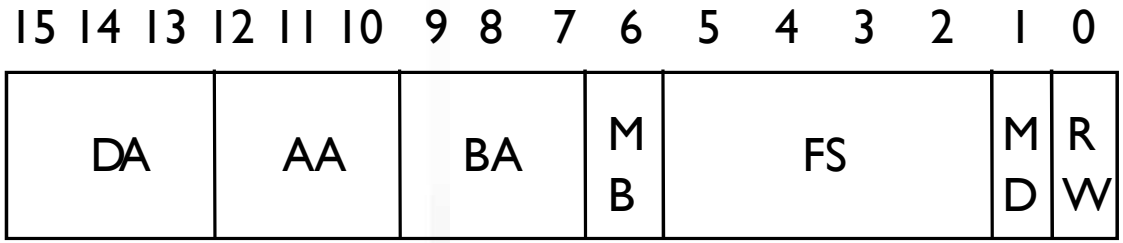
The Control Word

- The datapath has many control inputs
- The signals driving these inputs can be defined and organized into a *control word*
- To execute a microinstruction, we apply control word values for a clock cycle. For most microoperations, the positive edge of the clock cycle is needed to perform the register load
- The datapath control word format and the field definitions are shown on the next slide

Control Word Block Diagram



The Control Word Fields



Control word

- Fields
  - DA – D Address
  - AA – A Address
  - BA – B Address
  - MB – Mux B
  - FS – Function Select
  - MD – Mux D
  - RW – Register Write
- The connections to datapath are shown in the next slide

Control Word Encoding

Encoding of Control Word

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
$R0$	000	Register	0	$F \leftarrow A$	0000	Function	0	No write	0
$R1$	001	Constant	1	$F \leftarrow A + 1$	0001	Data In	1	Write	1
$R2$	010			$F \leftarrow A + B$	0010				
$R3$	011			$F \leftarrow A + B + 1$	0011				
$R4$	100			$F \leftarrow A + \overline{B}$	0100				
$R5$	101			$F \leftarrow A + \overline{B} + 1$	0101				
$R6$	110			$F \leftarrow A - 1$	0110				
$R7$	111			$F \leftarrow A$	0111				
				$F \leftarrow A \wedge B$	1000				
				$F \leftarrow A \vee B$	1001				
				$F \leftarrow A \oplus B$	1010				
				$F \leftarrow \overline{A}$	1011				
				$F \leftarrow B$	1100				
				$F \leftarrow \text{sr } B$	1101				
				$F \leftarrow \text{sl } B$	1110				

Microoperations for the Datapath - Symbolic Representation

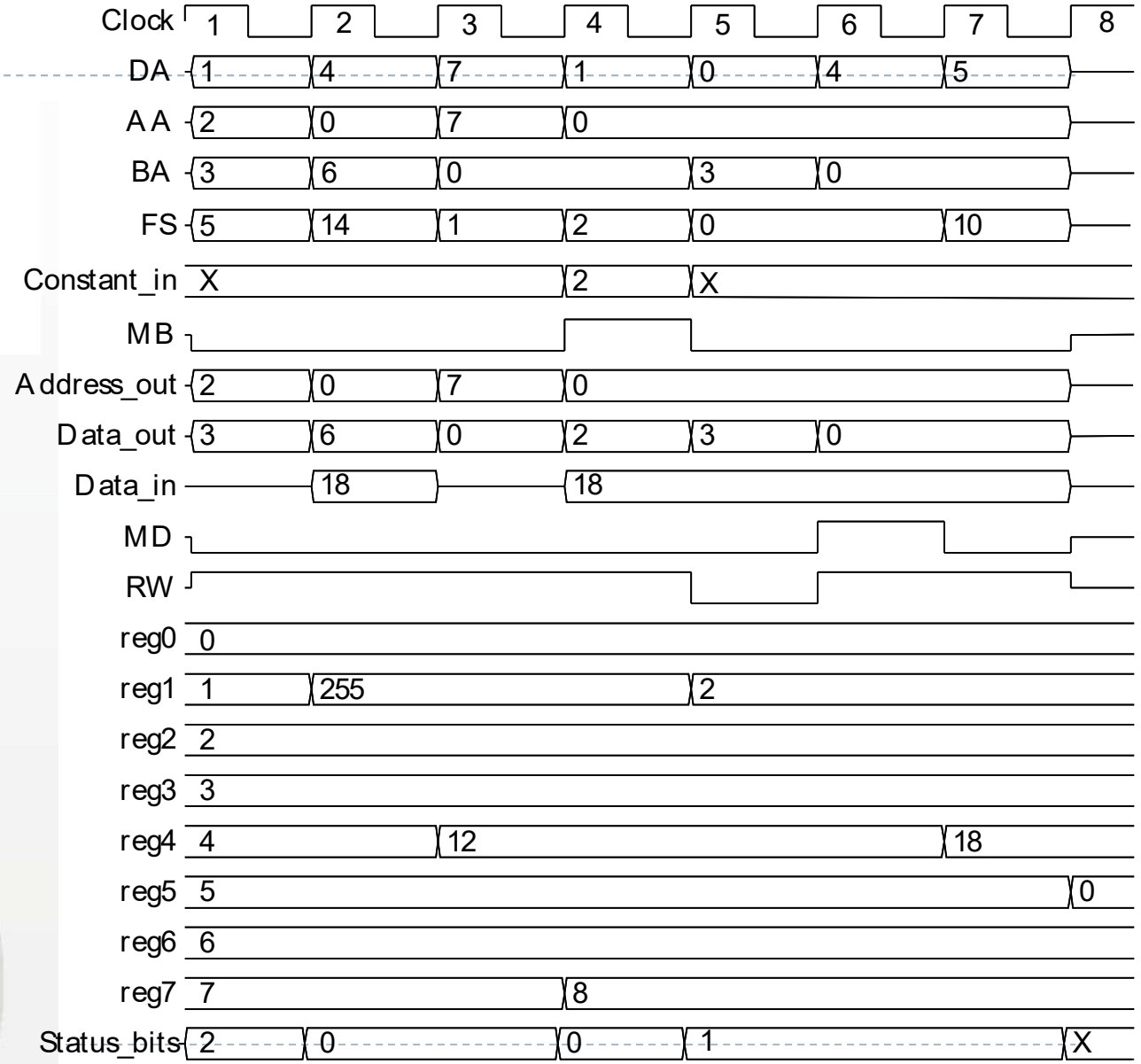
Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	—	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	—	Register	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	——		$R3$	Register	—	—	No Write
$R4 \leftarrow$ Data in	$R4$	——		—	—	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Microoperations for the Datapath - Binary Representation

Microoperation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	0	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

- Results of simulation of the above on the next slide

Datapath Simulation



Any Questions?