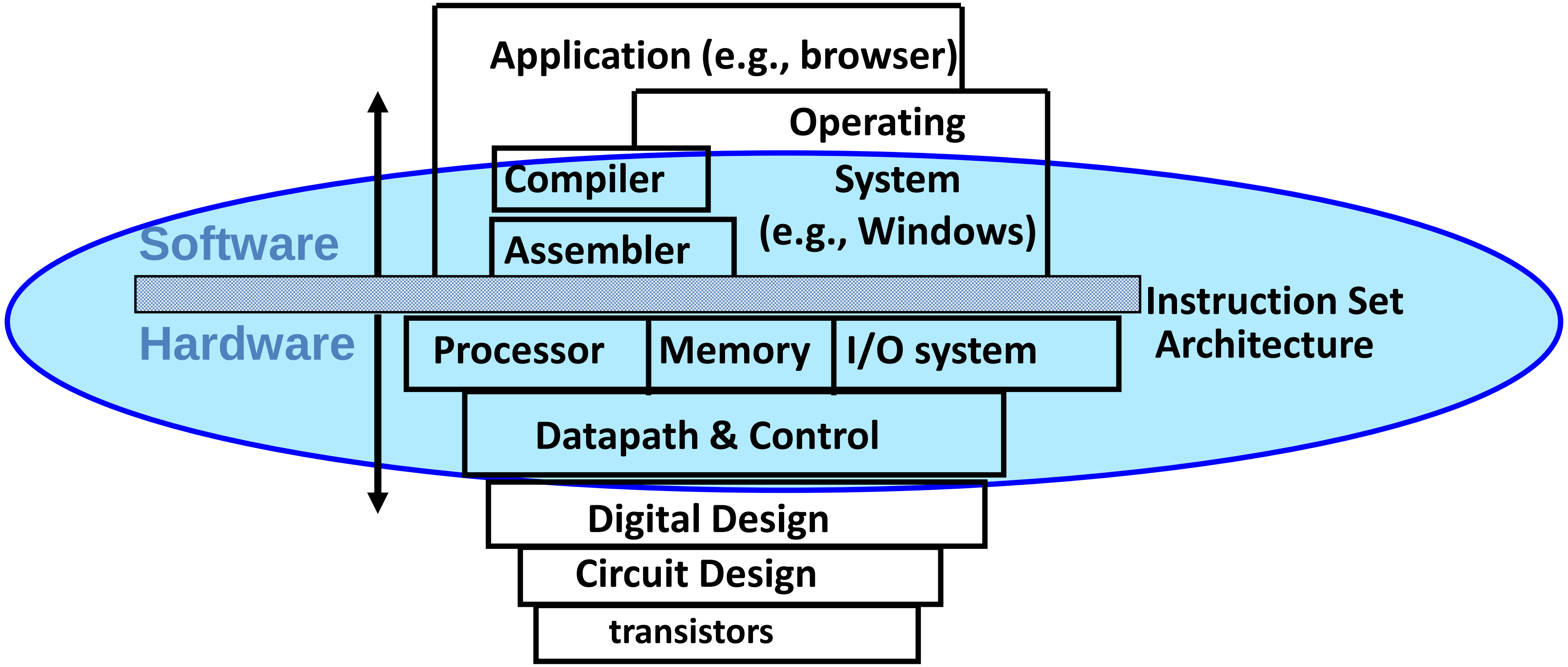
Registers and Memory

Agenda

- Registers
  - Data Register
  - Count registers
  - Shift registers
- Memory



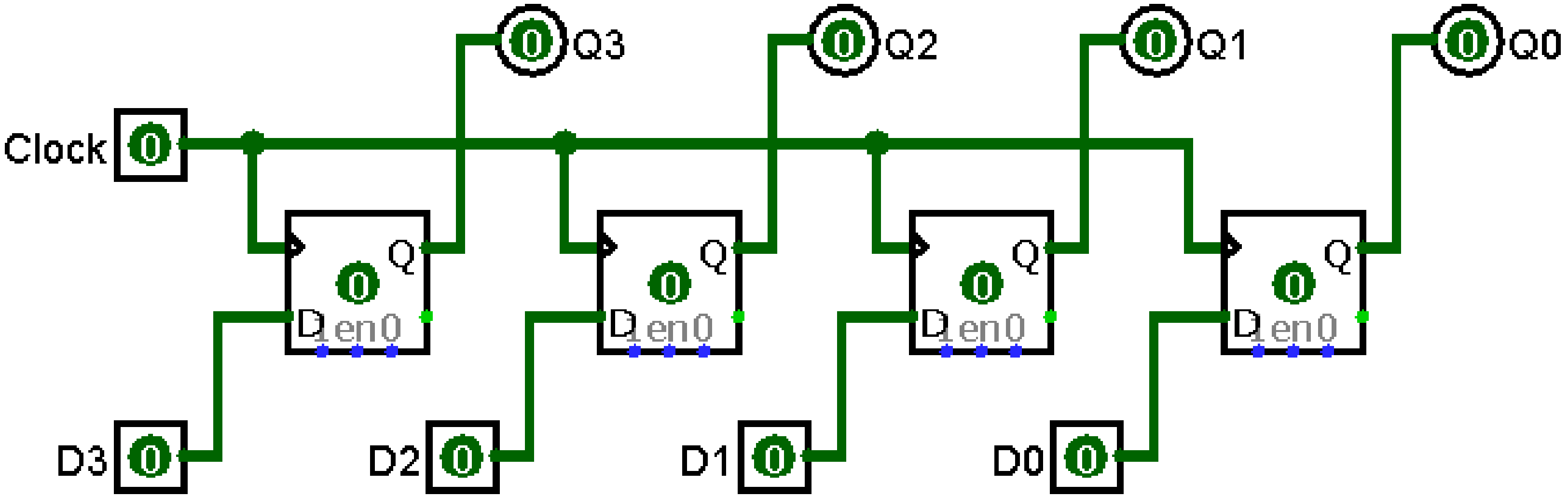
Registers

Register

- **Registers** are circuits inside of the processor
- Registers used extensively throughout the datapath
  - e.g. adding two numbers. The numbers are temporary stored in the registers and becomes the inputs of ALU

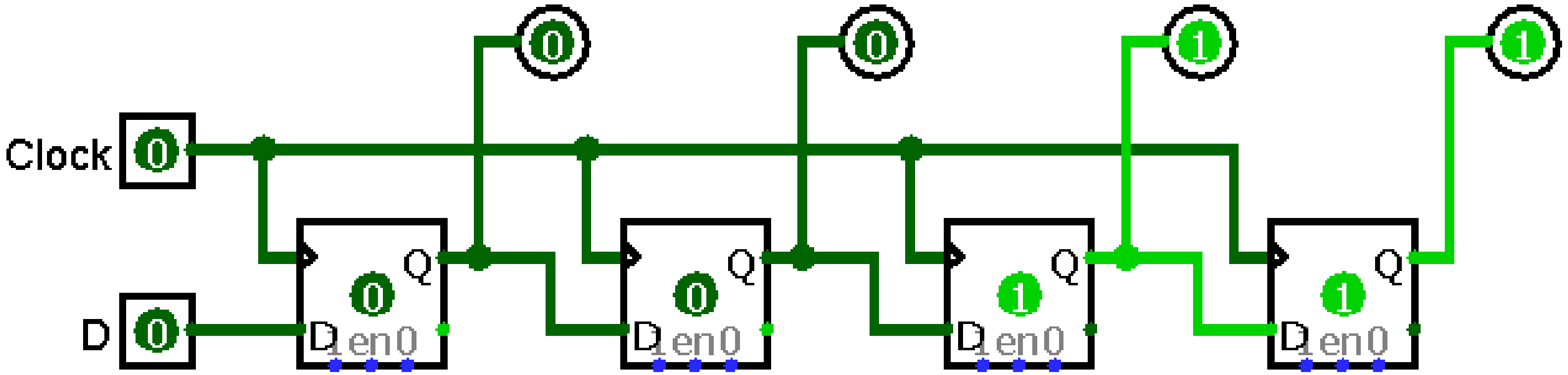
(General) Register

- General purpose: Store a multibit datum, e.g., byte or word
- Can implement with **an array of D flip-flops**



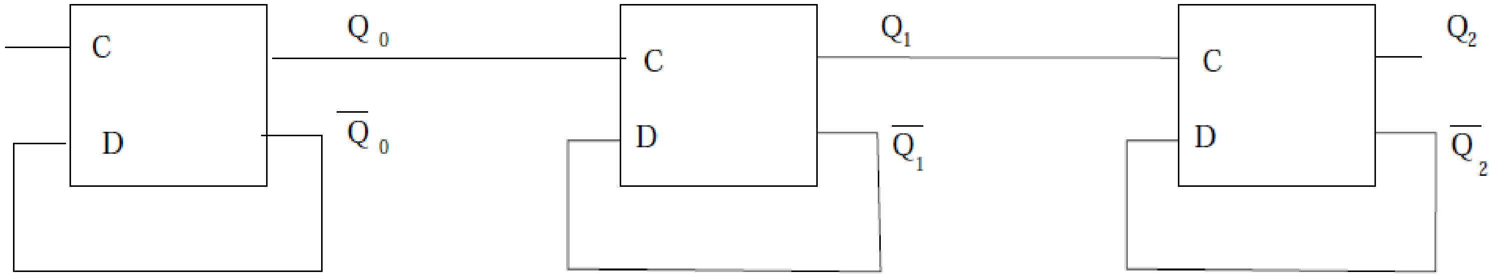
4-bit General Register

Shift Register

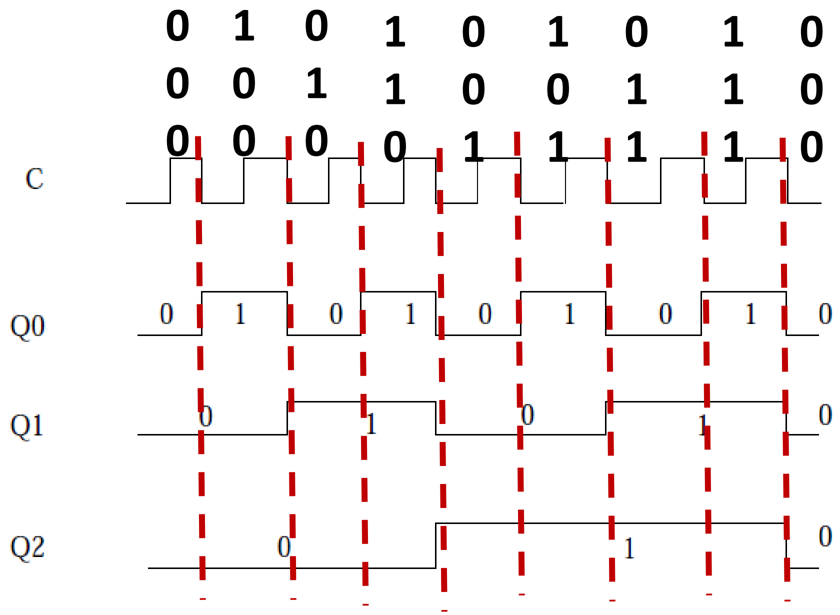


- Shifts the stored bits on each clock
- Can be designed to shift left or right
- What can this be useful for?
  - Multiplication or division by two
  - Conversion between serial and parallel interfaces

Count Register



- The counter for the processor
- A set of T flip flop
- Count up or count down, but when?
  - Count up toggled on falling-edge
  - Count down toggled on rising-edge
  - Alternatively, send Q' to next clock instead of Q

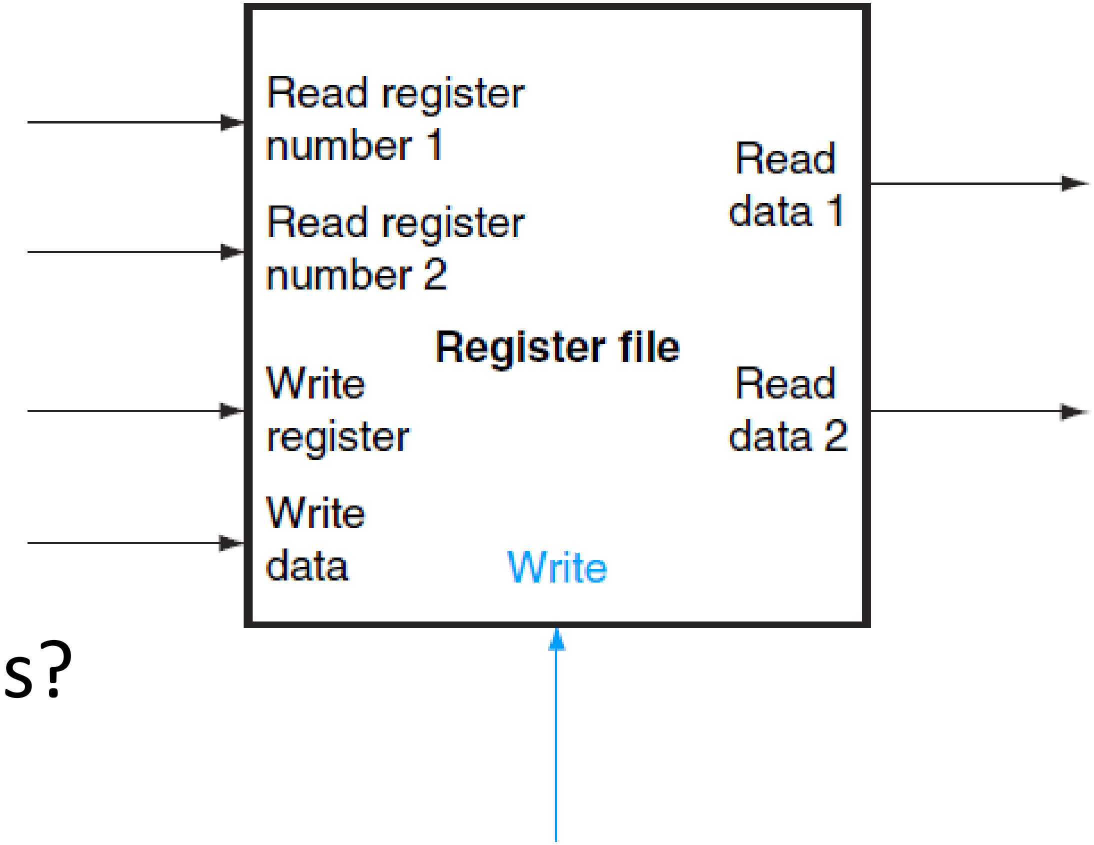


Register Files

Register File

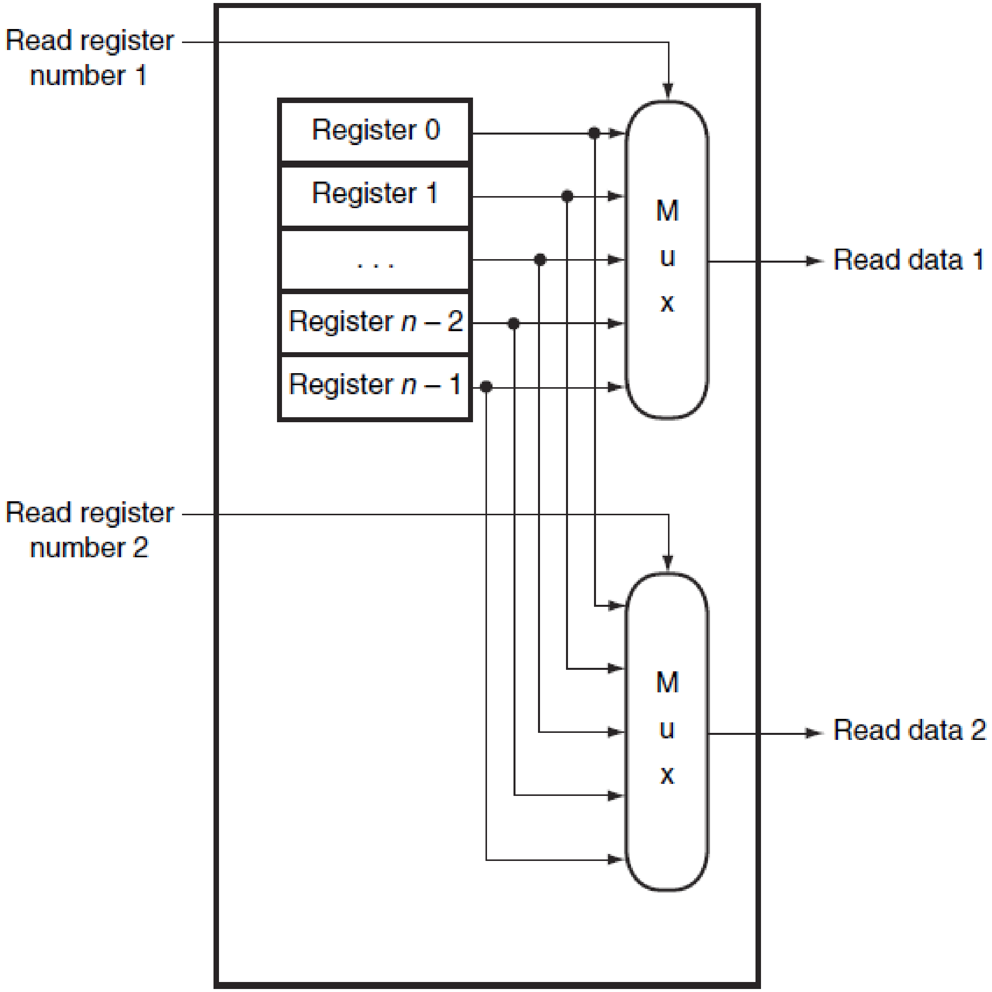
- A collection of registers
- Can read two registers
- Can write one register
- Write triggered by clock (blue)

- How many bits on different inputs?



Register File - Reading

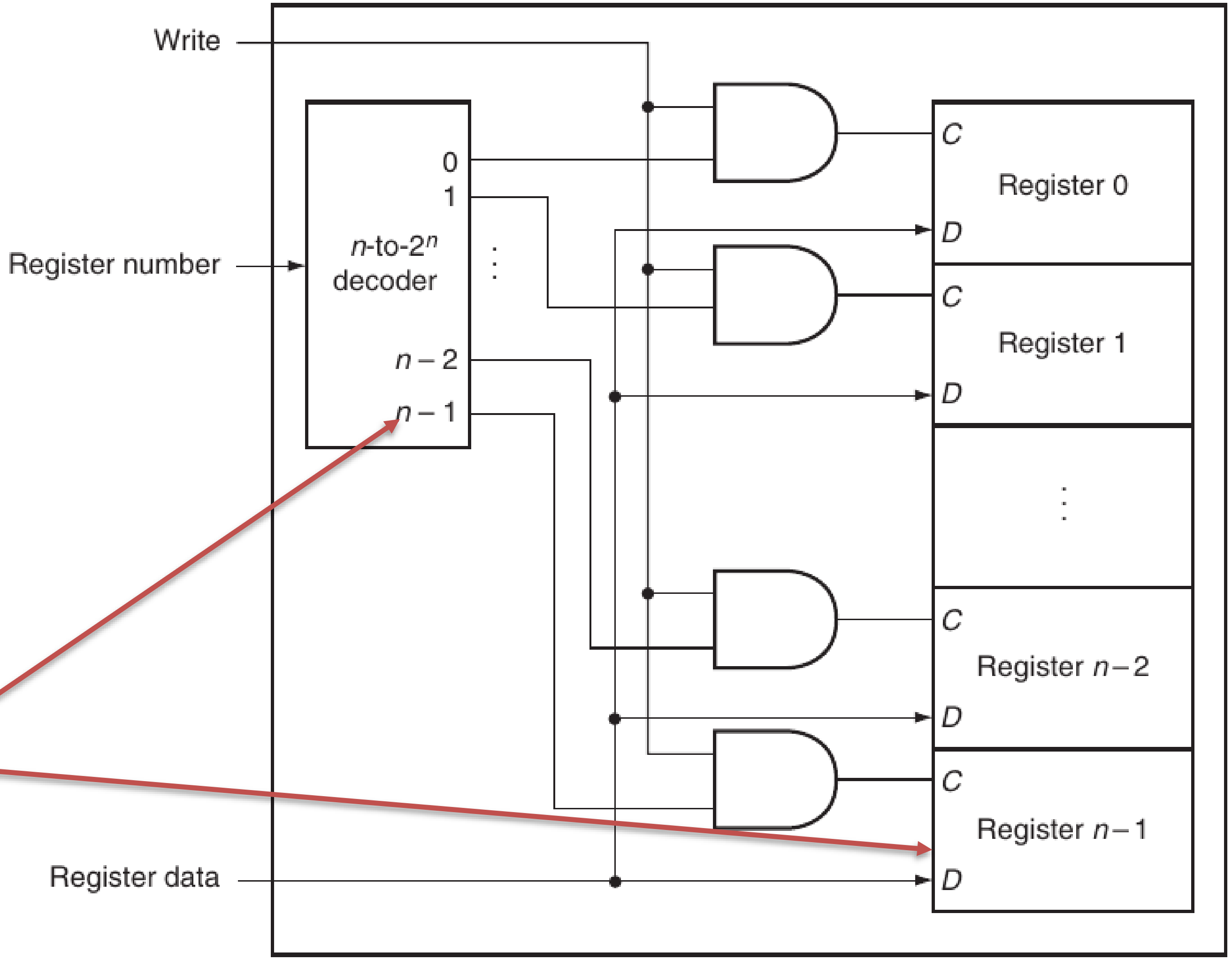
- Use **multiplexor** to select which registers to read



Register File - Writing

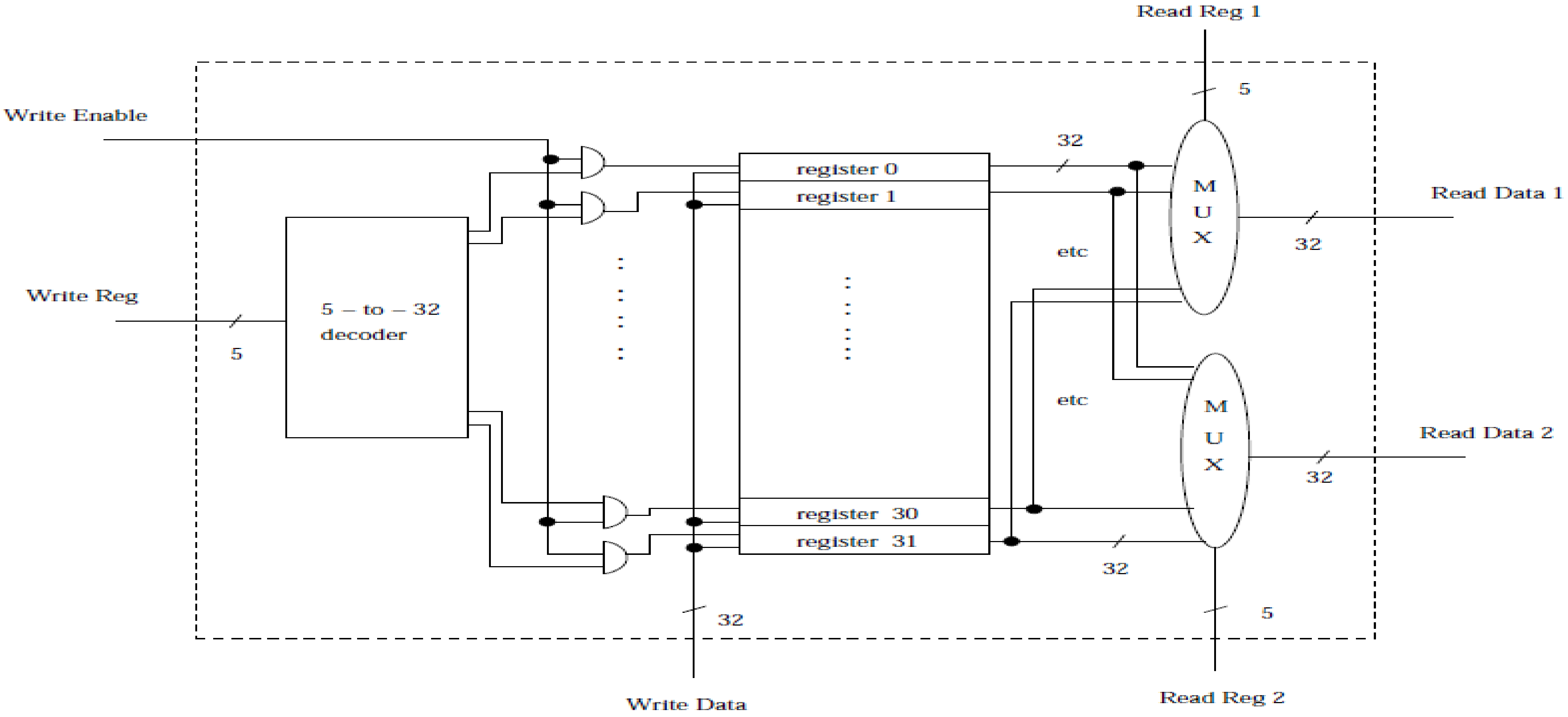
- Combine decoder with clock using AND to store the write data

Recall that the write operation of the D flip flops always needs to be triggered with the clock (i.e., to have the input D written to the output Q of the D flip flop). That's why we have a write enable for the write operation to be used for enabling the clocks of the D latches/flipflops

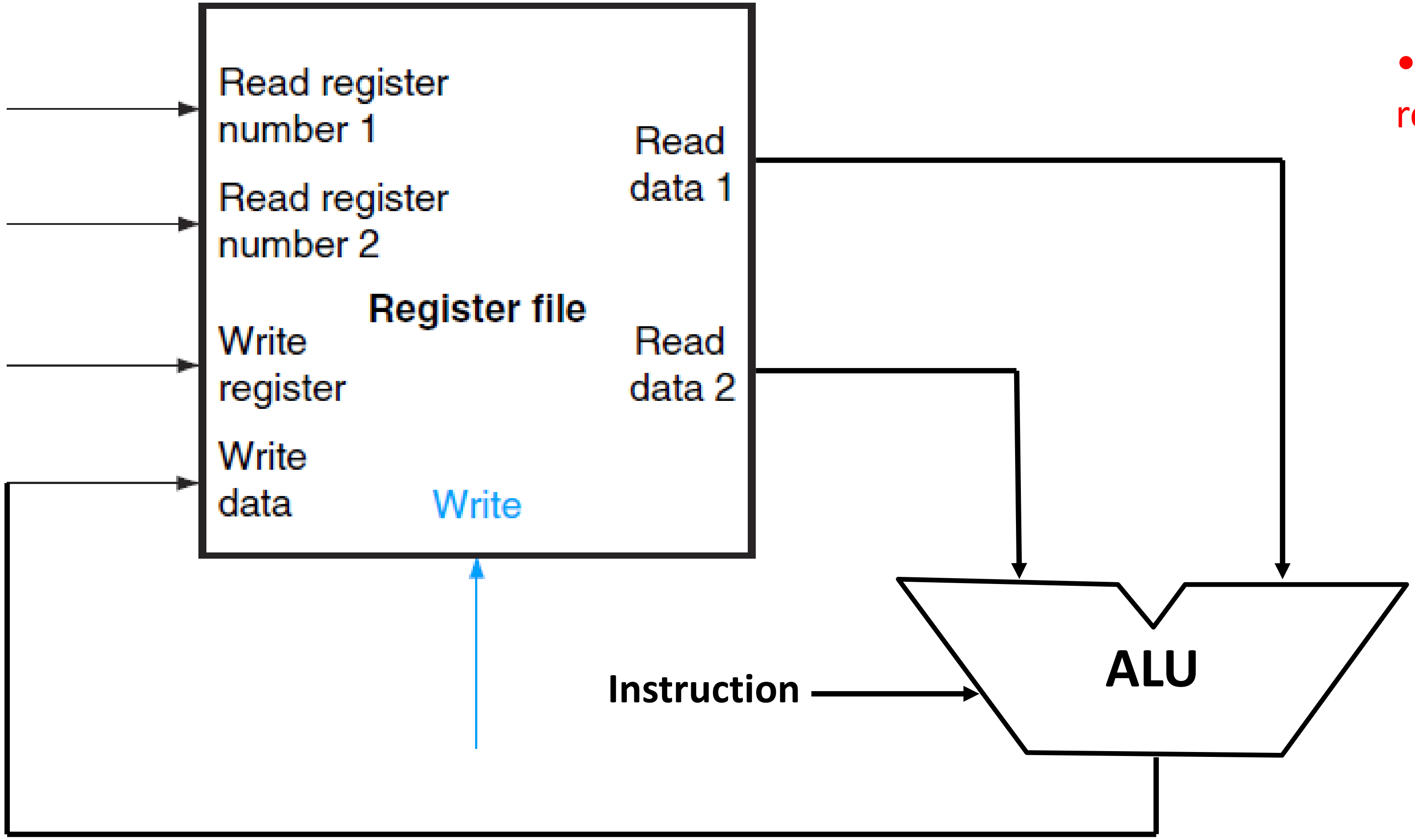


Please note that this figure shows a subset of the register file , which contains only Register 0 to Register n-1, Register n to Register 2^(n) –1 are omitted from the figure for simplicity

MIPS register file with 32 word registers



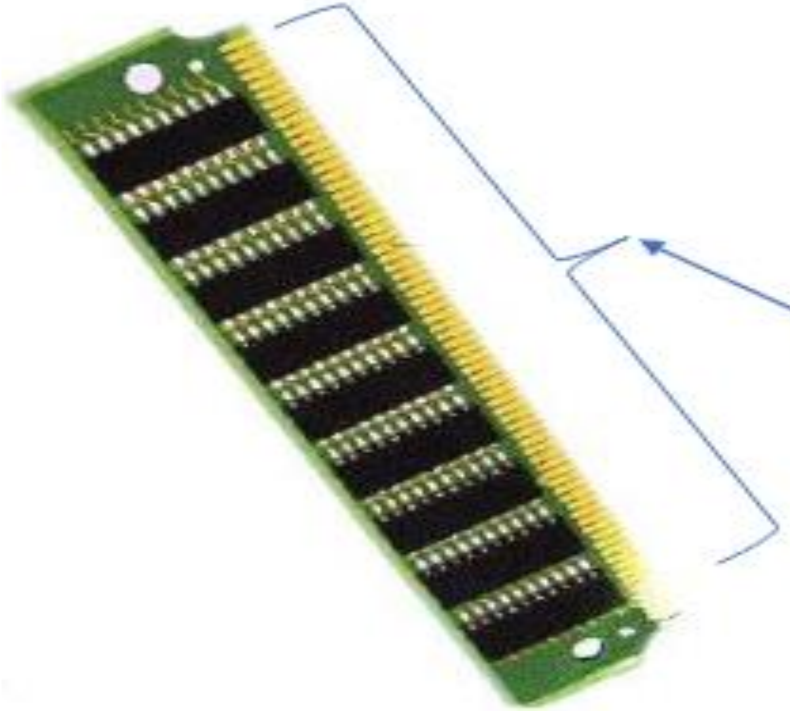
Register File with ALU



- Classify input/output lines of a register file into three types:
  - ***Addresses:*** Read register number 1, Read register number 2 and Write register.
  - ***Data:*** Read data 1, Read data 2 and Write data.
  - ***Control:*** Write

Memory

RAM



Random Access Memory

Construction:

- An array of bytes
- Read & Write control circuitry
  - **Address Register** = integer
  - **Mode Register** = flag
  - **Data Register** = byte

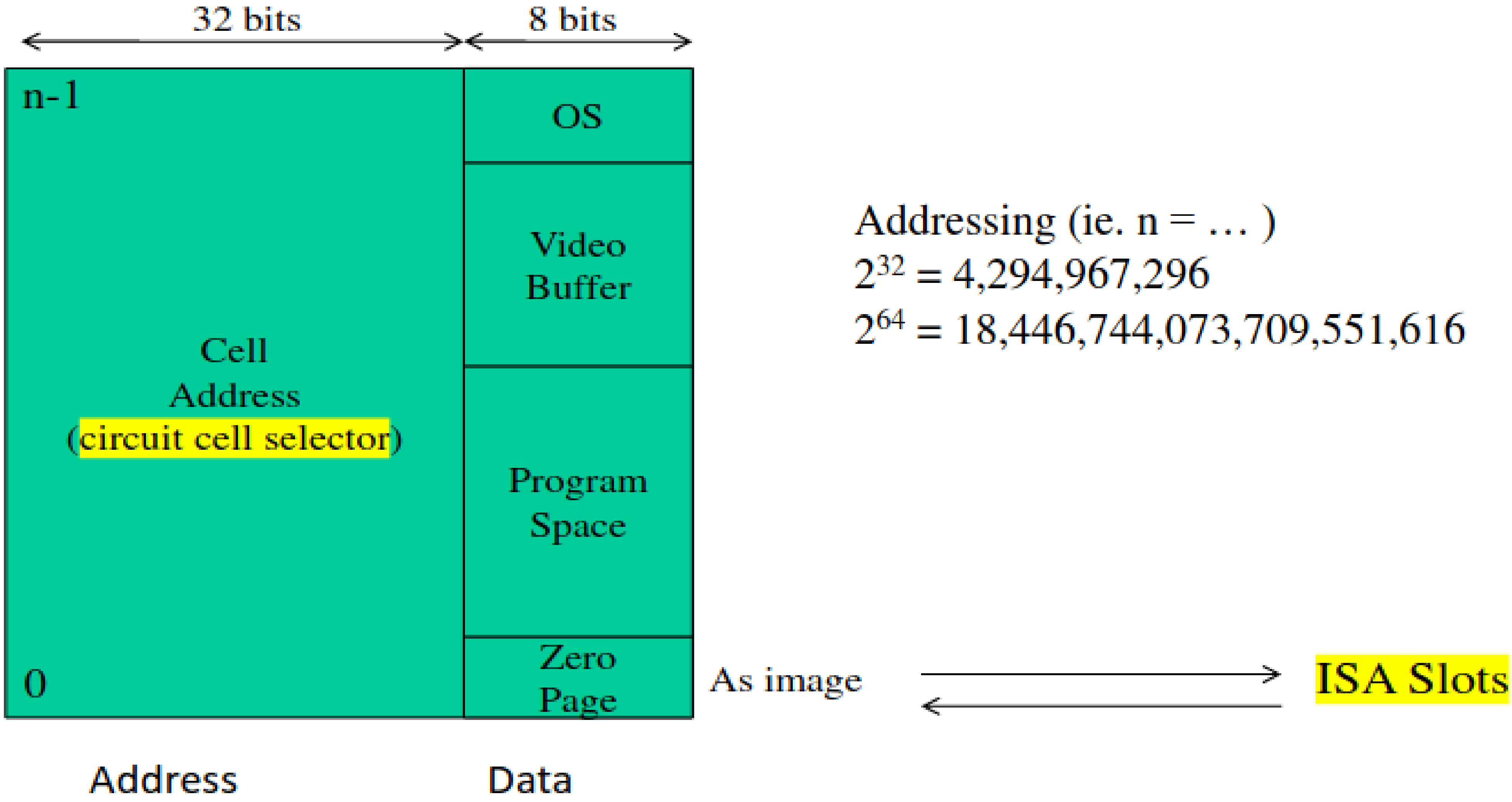
RAM

- To understand the architecture of the RAM, you should understand what are the tasks that are executed within the RAM that leads to this architecture.

- Imagine you are a delivery guy who has two perform one of the following two tasks:
  - Deliver a package to a certain address (Write certain memory location).
  - Receive a package from a certain address (Read certain memory location).

- The delivery man needs three piece of information to perform those tasks:
  - The address where he should deliver the package to, or receive the package from (Address register, and for RAM, the address should be encoded in binary format).
  - The type of the task, i.e., deliver a package or receive a package (Mode register for write or read)
  - The package itself to deliver or receive (data register).

Basic RAM Architecture

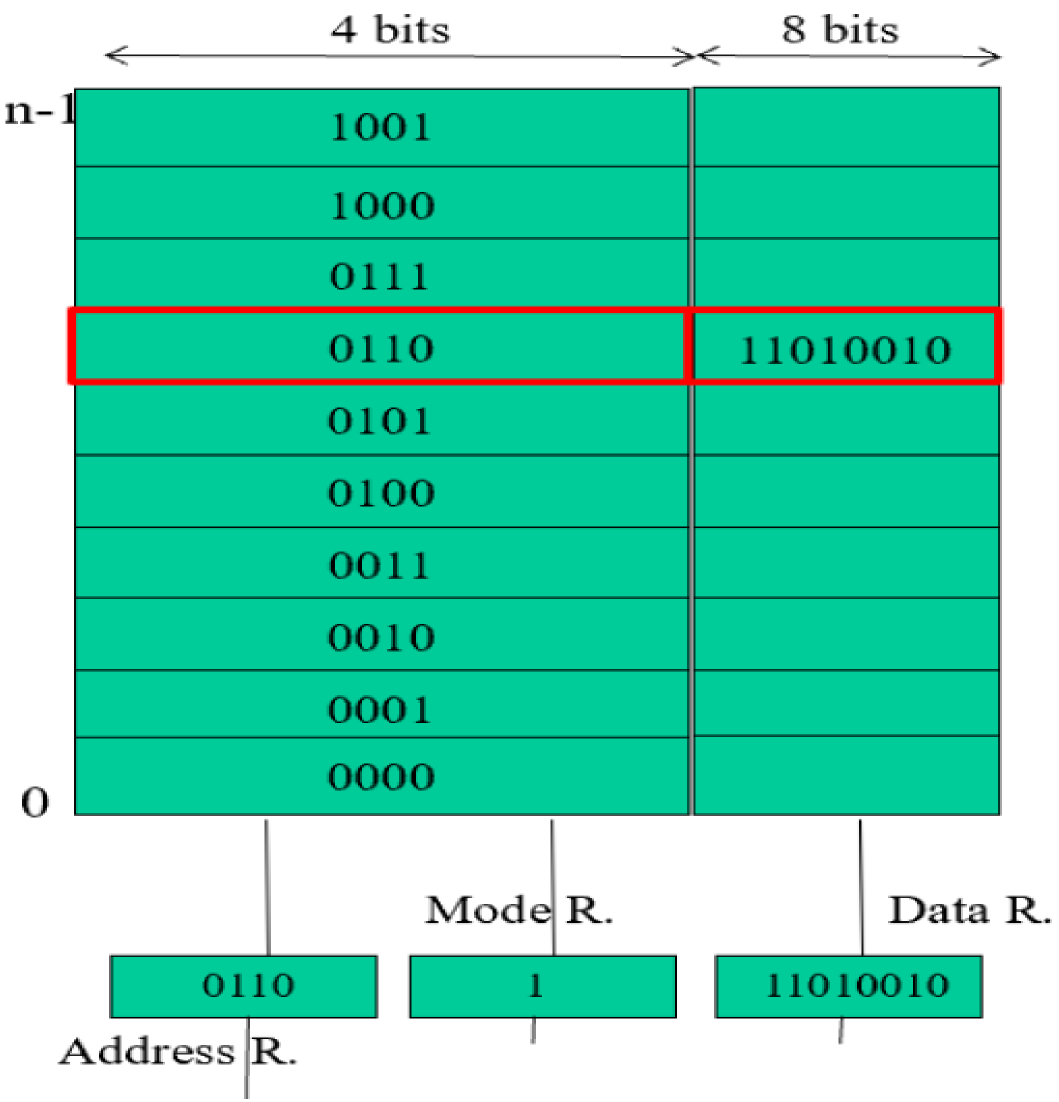


Addressing (ie. n = …)

$2^{32} = 4,294,967,296$

$2^{64} = 18,446,744,073,709,551,616$

Basic RAM Architecture



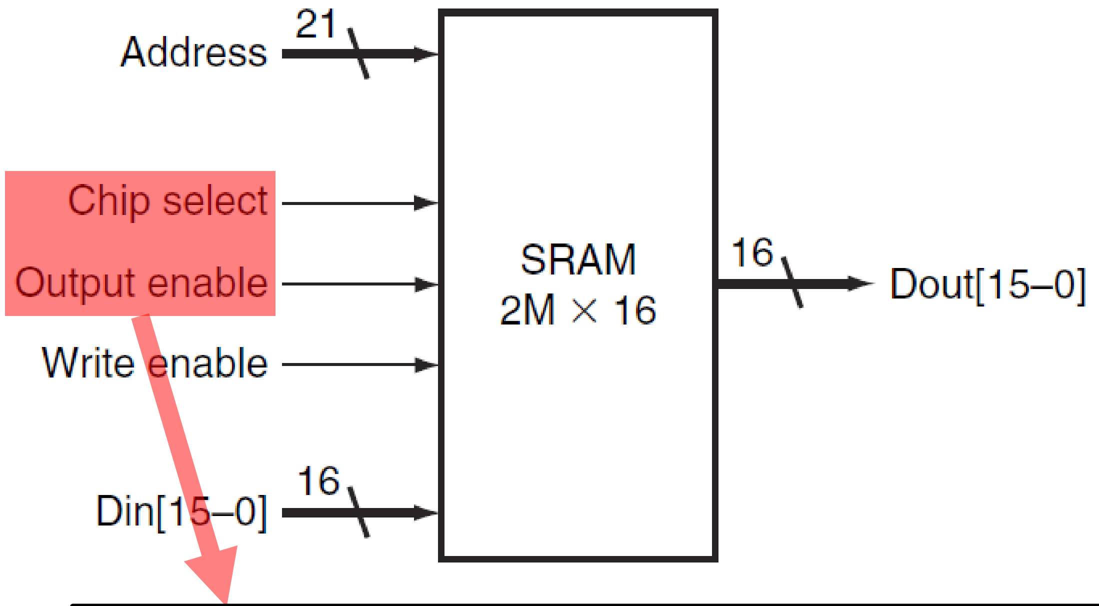
	4 bits	8 bits
n-1	1001	
	1000	
	0111	
	0110	11010010
	0101	
	0100	
	0011	
	0010	
	0001	
0	0000	

Operation 1:

Read contents of cell 0110

I/O of SRAM

Example: 2Mx16 configuration
(2M entries of 16-bit output lines)

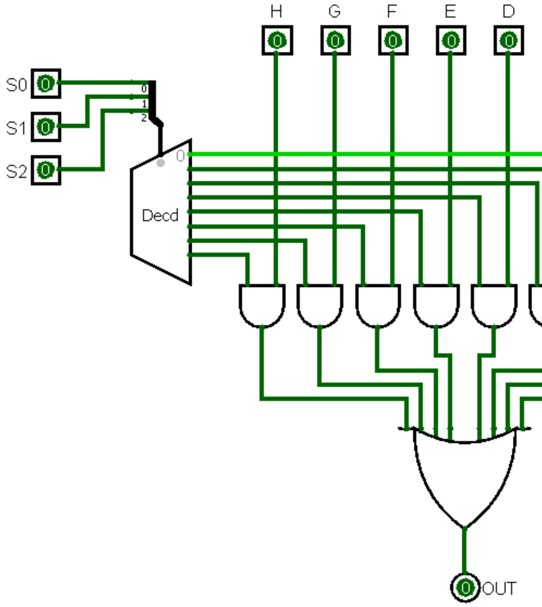


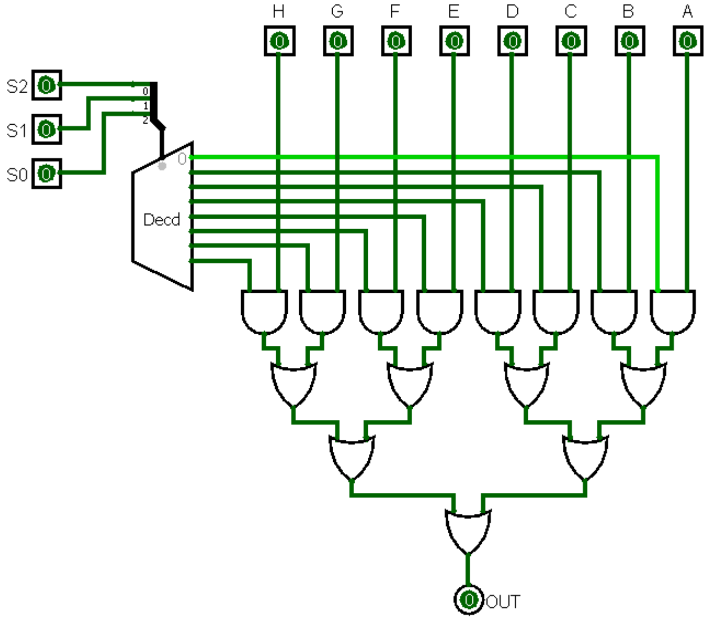
FINE PRINT: useful for both putting multiple memories on the same bus, and because output lines may be shared with input!

- **A set of memory locations**
- **Address** selects location
- **Chip select** must be active to read or write
- **Output enable** determines if output lines are driven
- **Write enable** determines if memory should be written
- **Din** is the Data

SRAM Multiplexor

- **SRAM is HUGE and cannot be built like register files**
  - The multiplexor grows with the number of memory locations
  - Giant OR is more realistically implemented with a tree of OR gates

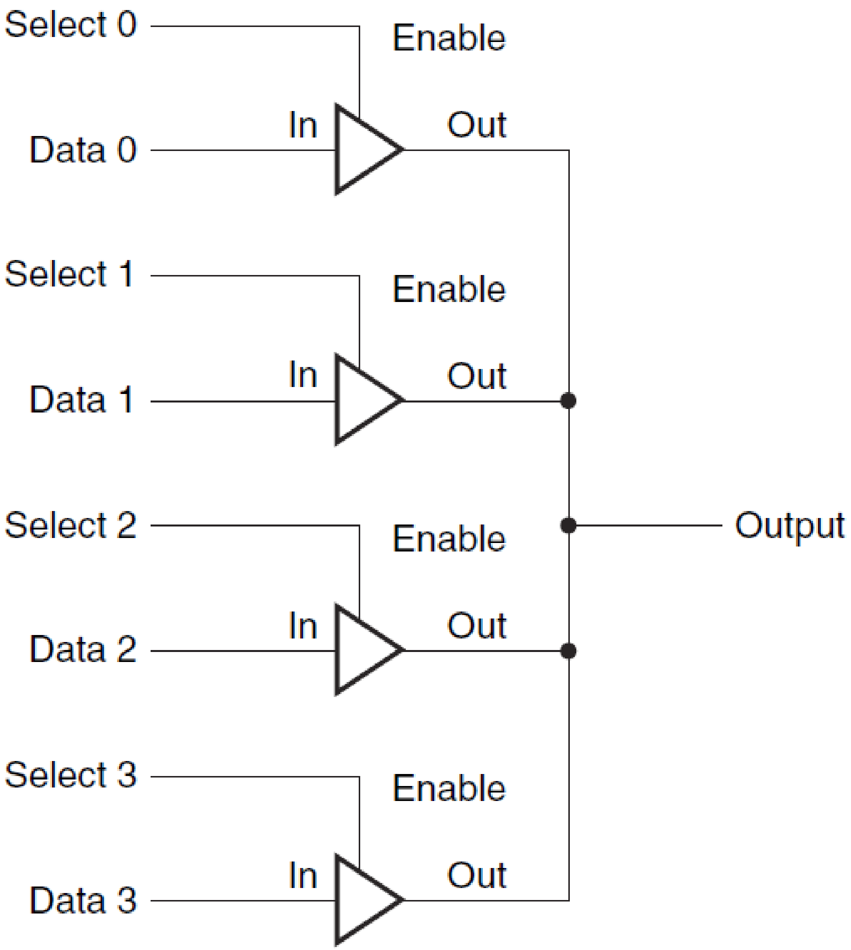




SRAM Multiplexor

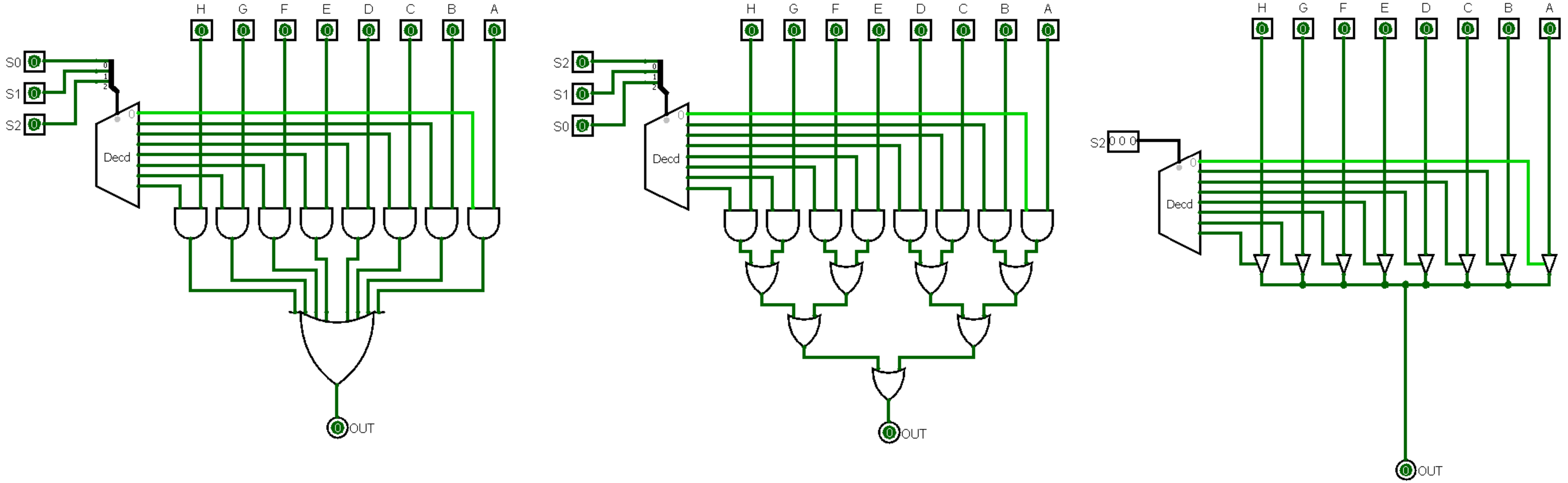
Data	Enable	Out
0	1	0
1	1	1
x	0	Z

- Instead, the multiplexor for SRAM is implemented with **tri-state buffer**
- Tri-state buffer
  - Select = 1, output = data
  - Select = 0, output = **high impedance state**
    - Effectively disconnected
    - Denoted Z in truth table
- Only one out is active (or **short circuit**)



SRAM Multiplexor

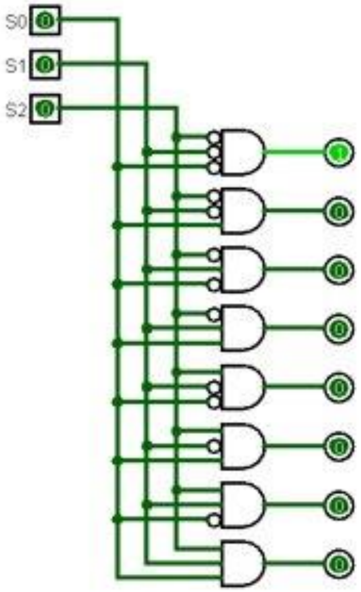
Avoid AND array and OR tree with an array of Tri-State buffers



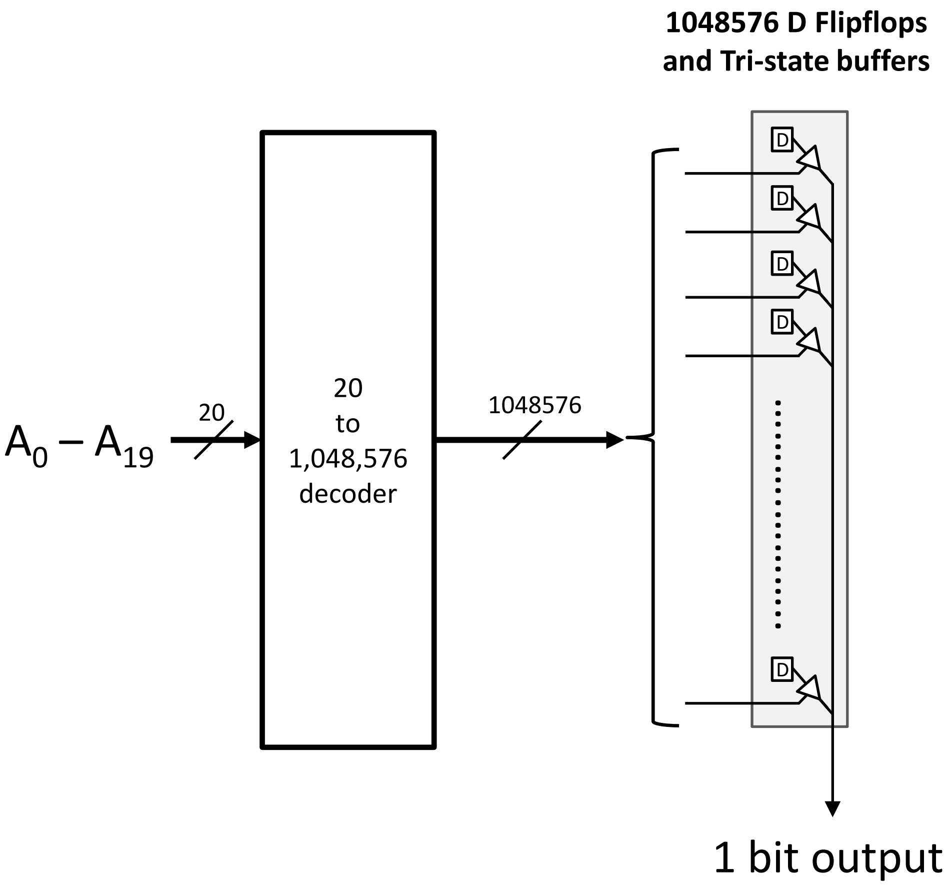
Need carefully design the select signal to avoid short circuits

SRAM Decoder

- May still need a large decoder
- 1 megabit memory would need a 20 to 1,048,576 decoder!

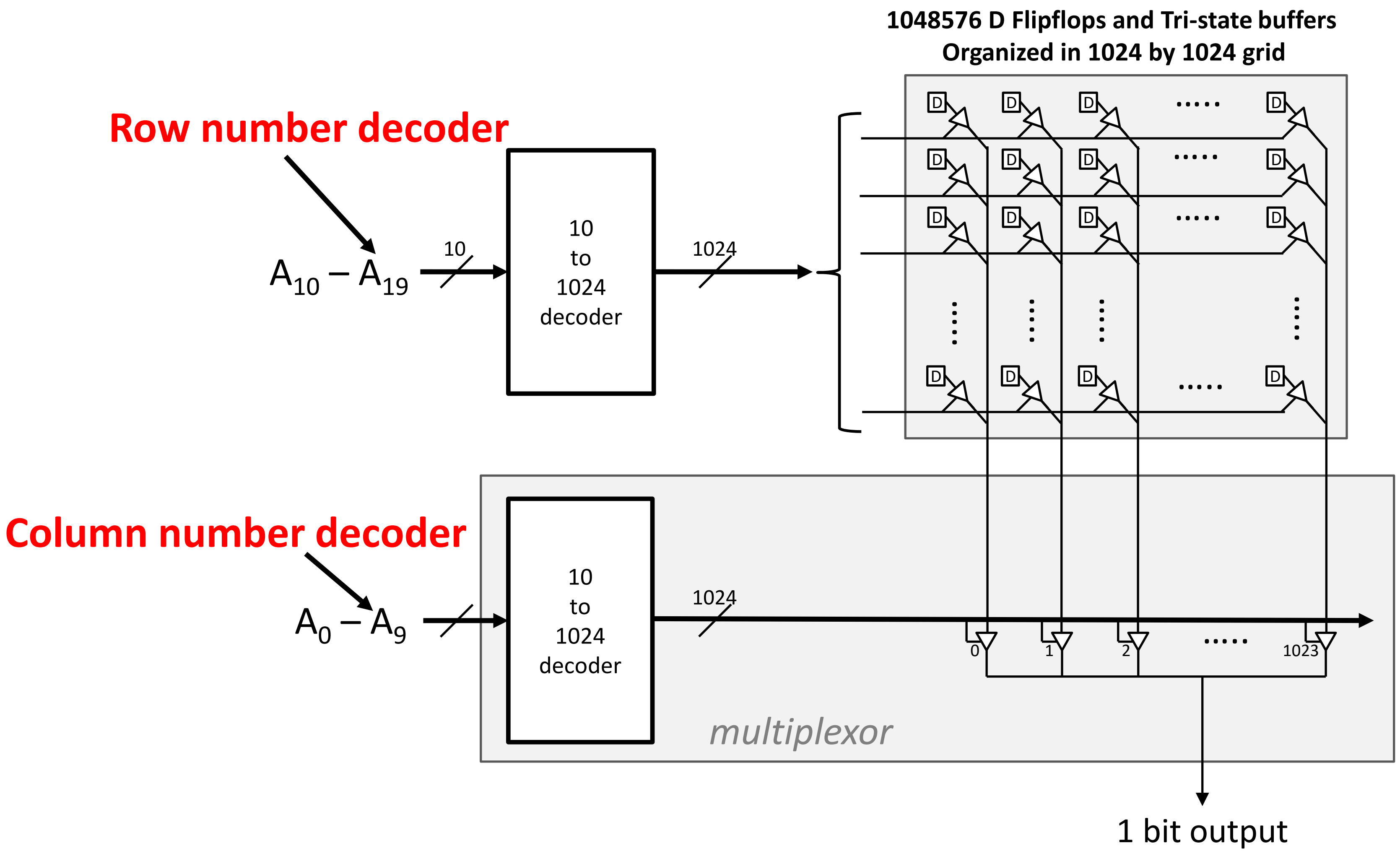


3-to-8 decoder



HUGE decoder with over one million outputs!

SRAM Decoder

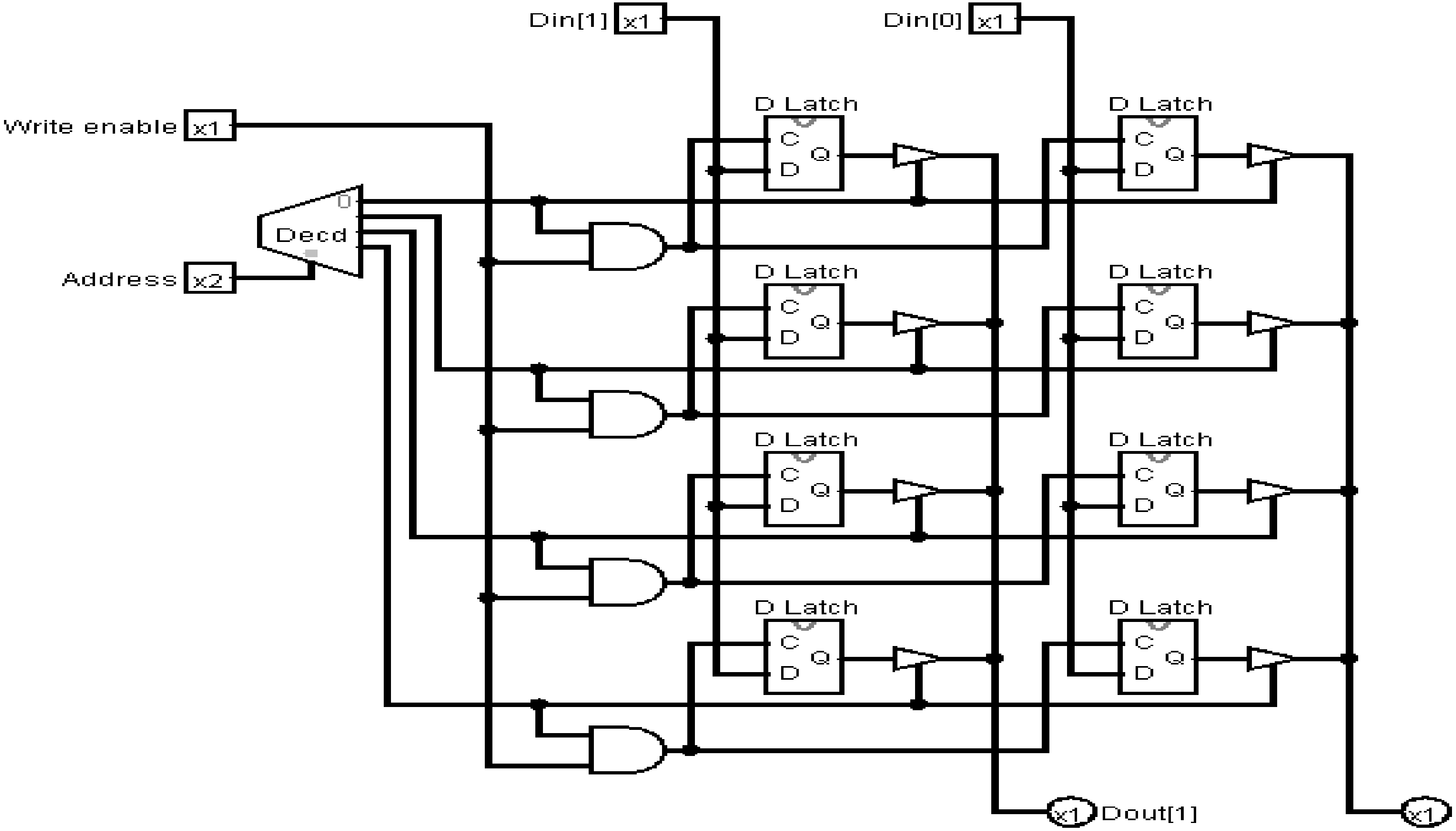


Organizing memory in rectangular arrays and use two decoders

SRAM Decoder

- How much can we save from 1MB memory
- A linear array
  - One 20 to 1048576 decoder for all address lines
- A rectangular array
  - One 10 to 1024 decoder for the upper address lines
  - One 10 to 1024 decoder for the lower address lines

Multi-bit Access



Basic 4x2 SRAM

Review

- Registers
  - Data registers
  - Count registers
  - Shift registers
- Memory
  - SRAM multiplexer
  - SRAM decoder
  - Multi-Bit Access
- Textbook
  - B8, B9 of 5th and 6th edition, C7-C10 of 4th edition