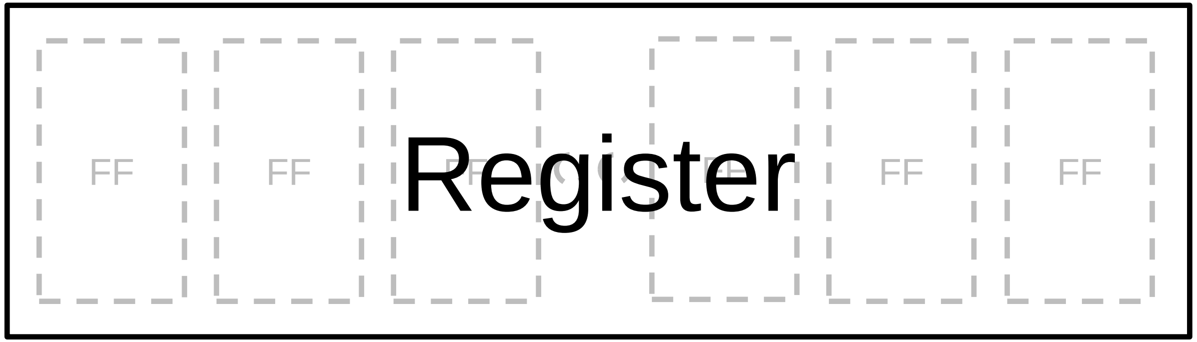
Registers

CMSC 313
Raphael Elspas

Register definition

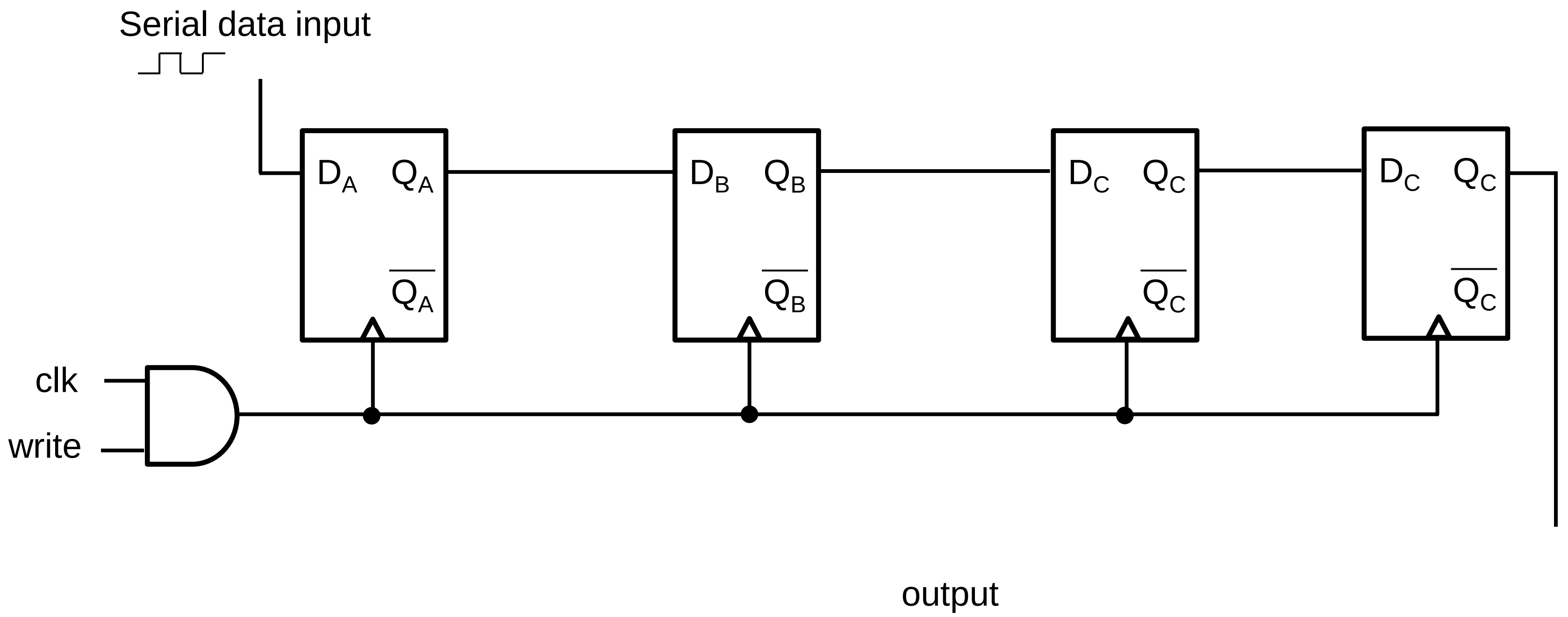
- A **register** is a device that can store binary information
- A register can be made with flip flops
- Each flip flop can store 1 bit of data, so an n-bit register needs n flip flops.



Loading and reading

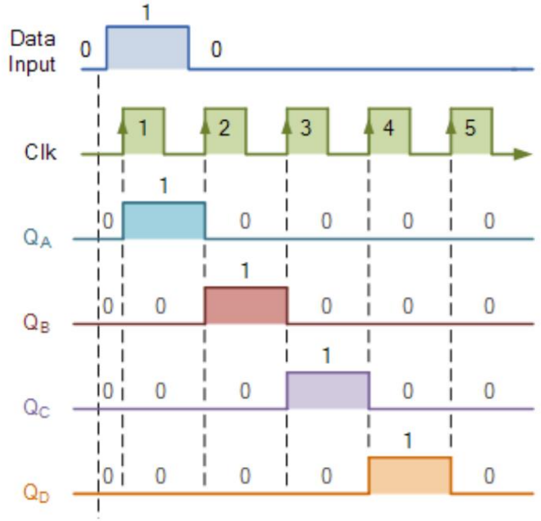
- Data can be loaded into the register in two ways: serially, or in parallel
- Data can be read from the register in two ways: serially, or in parallel
- This produces 4 different combinations in which registers can be made: Serial in, serial out (SISO); Serial in, parallel out (SIPO), parallel in, serial out (PISO), and parallel in, parallel out (PIPO).
- Thinking: What are the tradeoffs of parallel vs serial i/o?
- The last of these (PIPO) is the most powerful, lets see how to build it.

Serial in serial out (SISO)



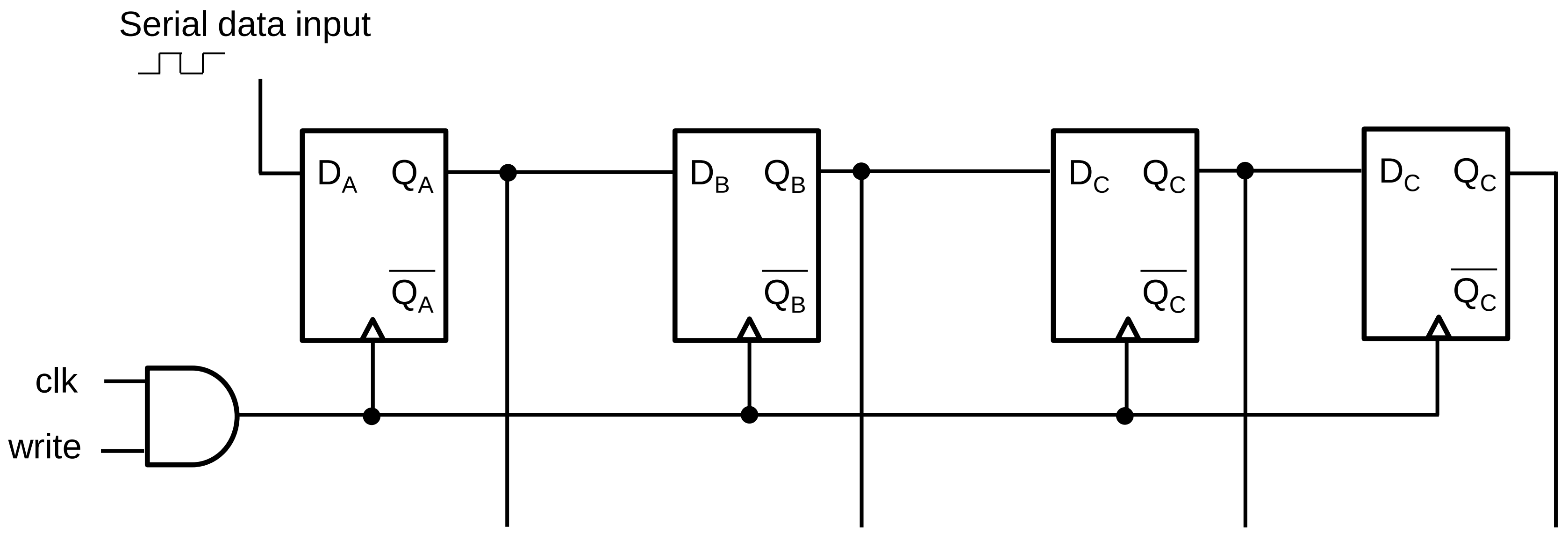
Example

- If I put a 1 in for one clock cycle, and 0s for 3 clock cycles, I will get this behavior



- A serial input shift register takes n clock cycles to write a value, where n is the number of bits in the register

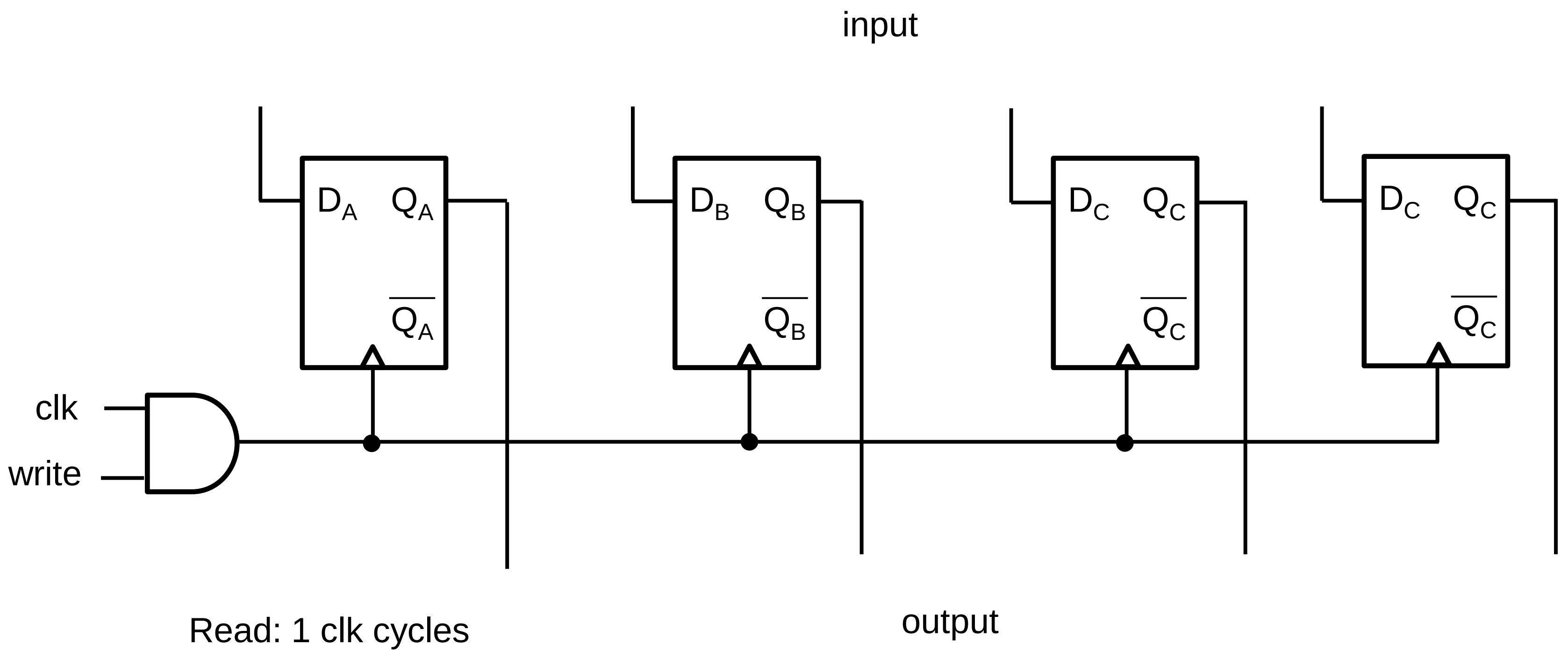
Serial in Parallel out (SIPO)



output

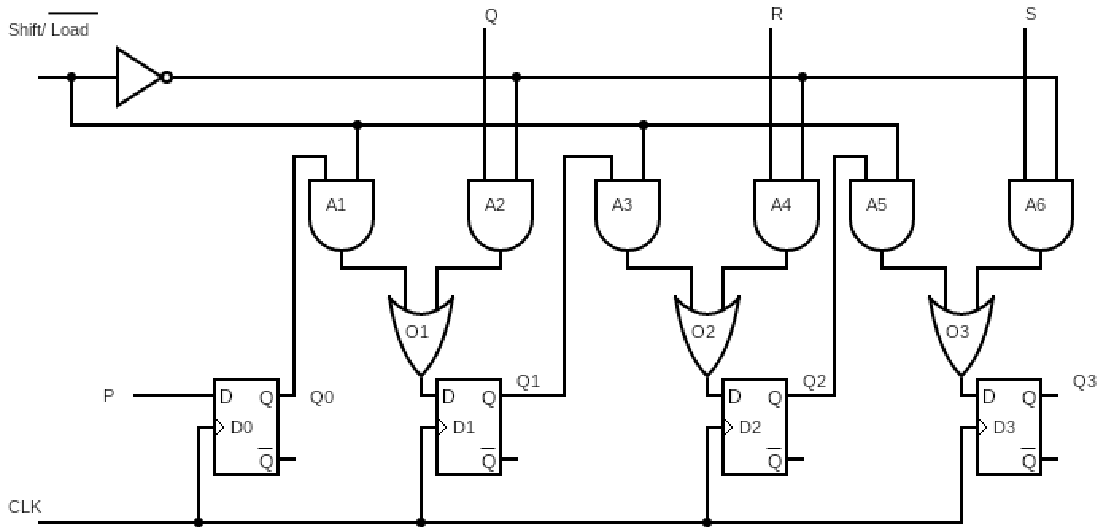
Read: 1 clk cycles
Write: n clk cycles

Parallel in, parallel out (PIPO) register



Read: 1 clk cycles
Write: 1 clk cycles

Parallel in Serial out (PISO)



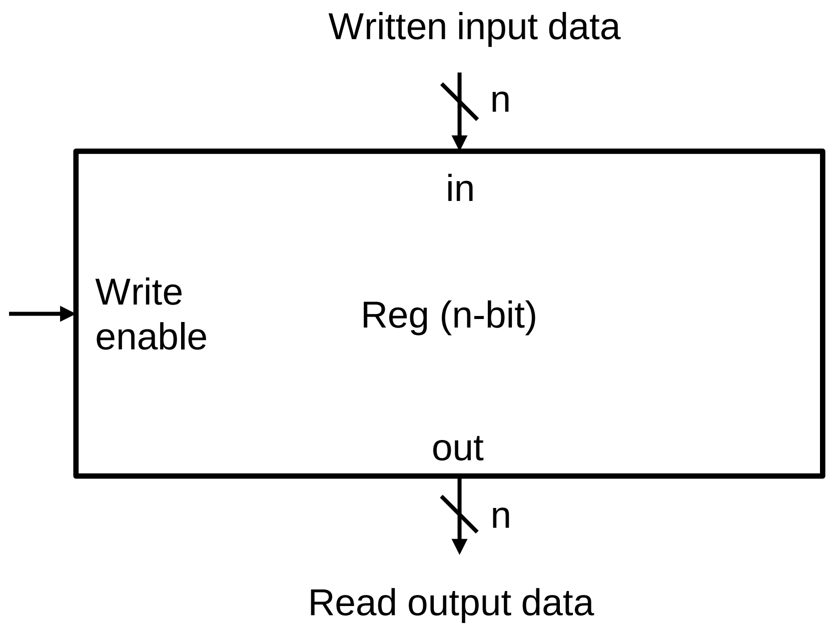
Read: n clk cycles
Write: 1 clk cycles

Has 2 functionalities:
1. Can store information
2. Can SHR (shift right) its contents

How is a register used?

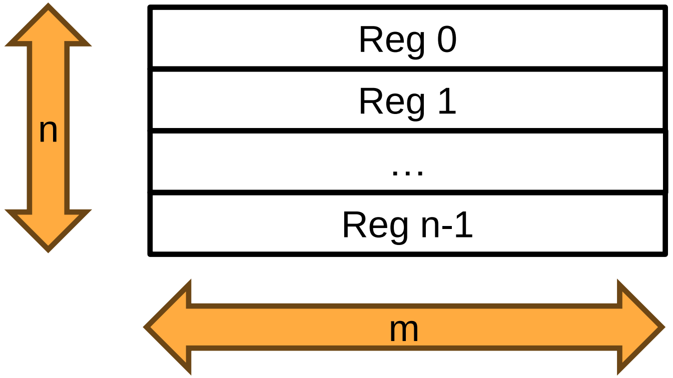
- A register is used like a variable to store a value
- We can write to store a value to use it for later
- We can read a value (multiple times if needed) in the future
- If the register is a PIPO, It takes one clock cycle to write to a register, and one clock cycle to read from a register.
- In a PIPO we can also write to a register and read its previous state in the same clock cycle (because of the master slave design)

Register block diagram for PIPO



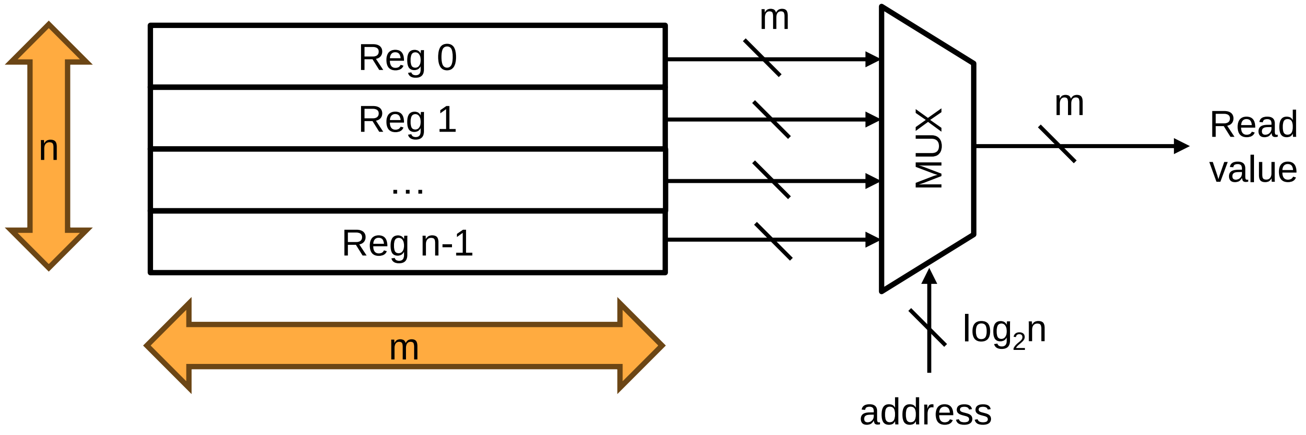
Register file

- It's useful to have a large block of registers both conceptually and from an implementation perspective
- A **register file** is a collection of registers where all the registers are the same length.
- An n x m-bit **register file** will have n registers within, each register being able to store m-bits each.



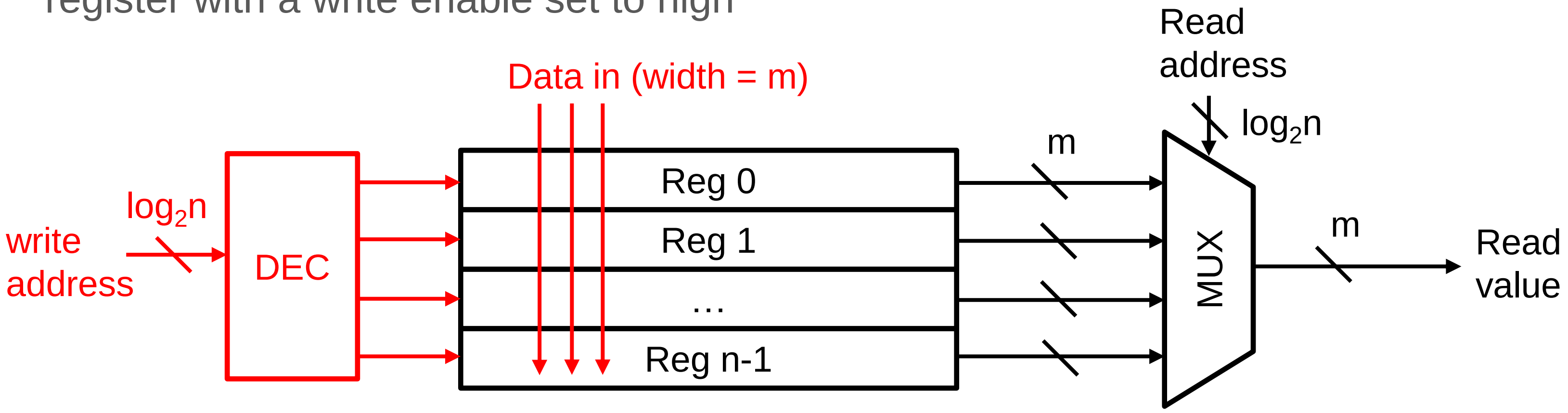
Register file indexing (read)

- In order to access a specific register from 0 through N-1, we can use a MUX to select which register to read from. The mux will have N select lines, each with a width of
- To identify which of the registers we're reading from, we'll use an **address** to index into the file



Register file indexing (write)

- Writing to a register requires 2 inputs: the data in and an enable to say that we actually want to overwrite the data. Not having this would mean every clock cycle, the data would be overwritten with whatever was on the input data lines
- To identify which of the registers we're writing to, we'll use an **address** to index into the file
- All write data inputs to each register are hooked up in parallel so data in will affect any register with a write enable set to high



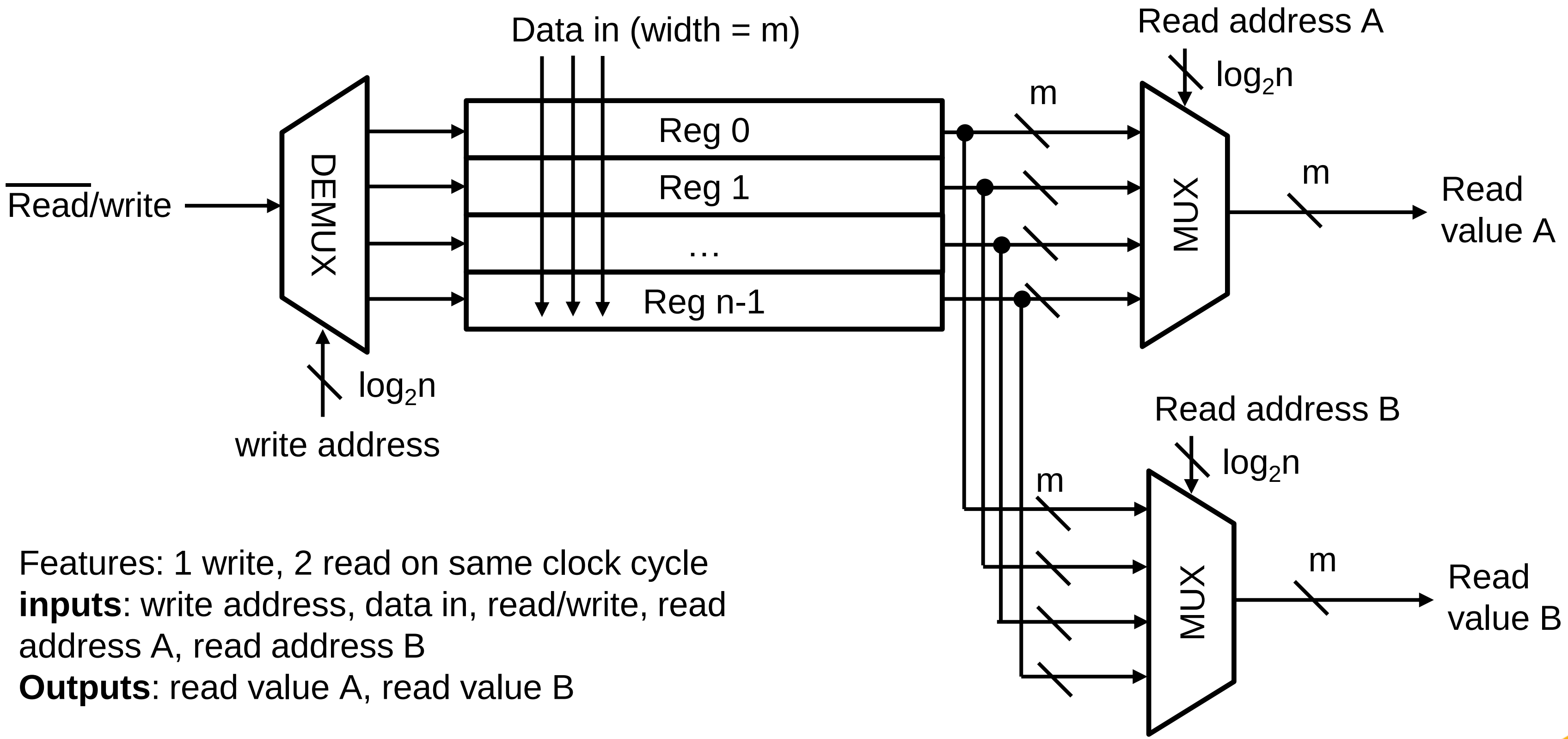
Register file (cont.)

- Let’s consider some design choices by looking at typical use cases in computer operations. For example:

  C = A + B

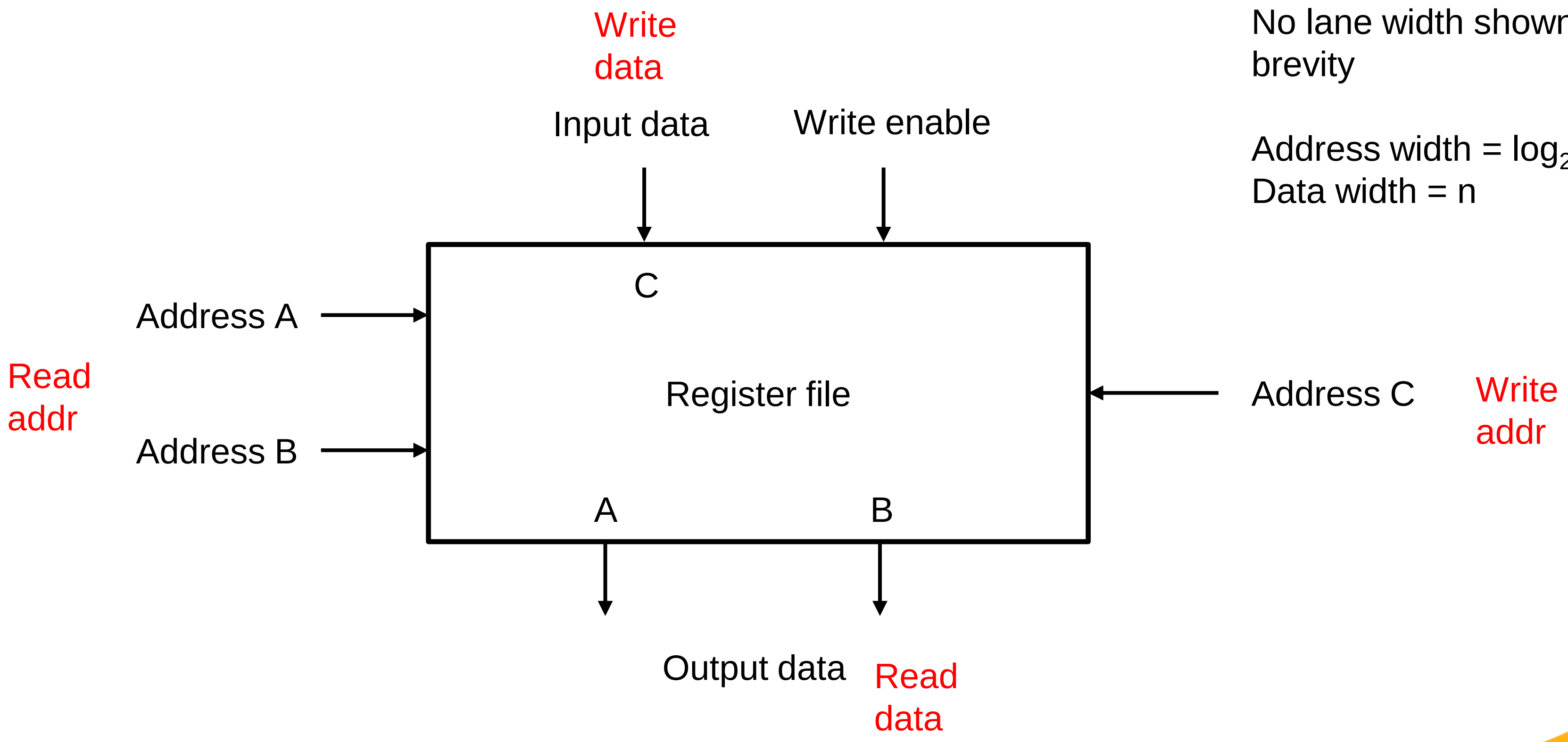
  Where A and B are values in 2 different registers and C is a value in a third register.
- We’d like to **read** 2 values at once and **write back** a value in one clock cycle to improve efficiency.
- We’d like to have independent read and write control, so we don’t write on every cycle

Register file (2 read ports)



Features: 1 write, 2 read on same clock cycle
inputs: write address, data in, read/write, read address A, read address B
Outputs: read value A, read value B

Register file block diagram

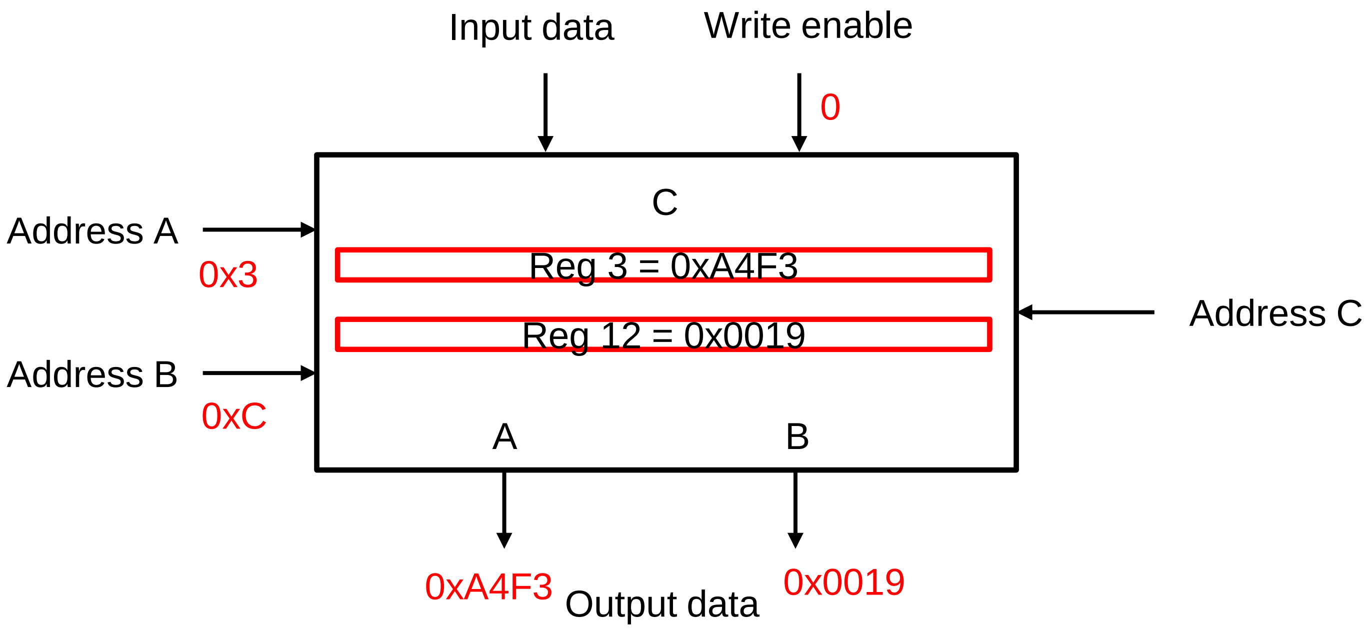


No lane width shown for brevity

Address width = $\log_2 n$
Data width = n

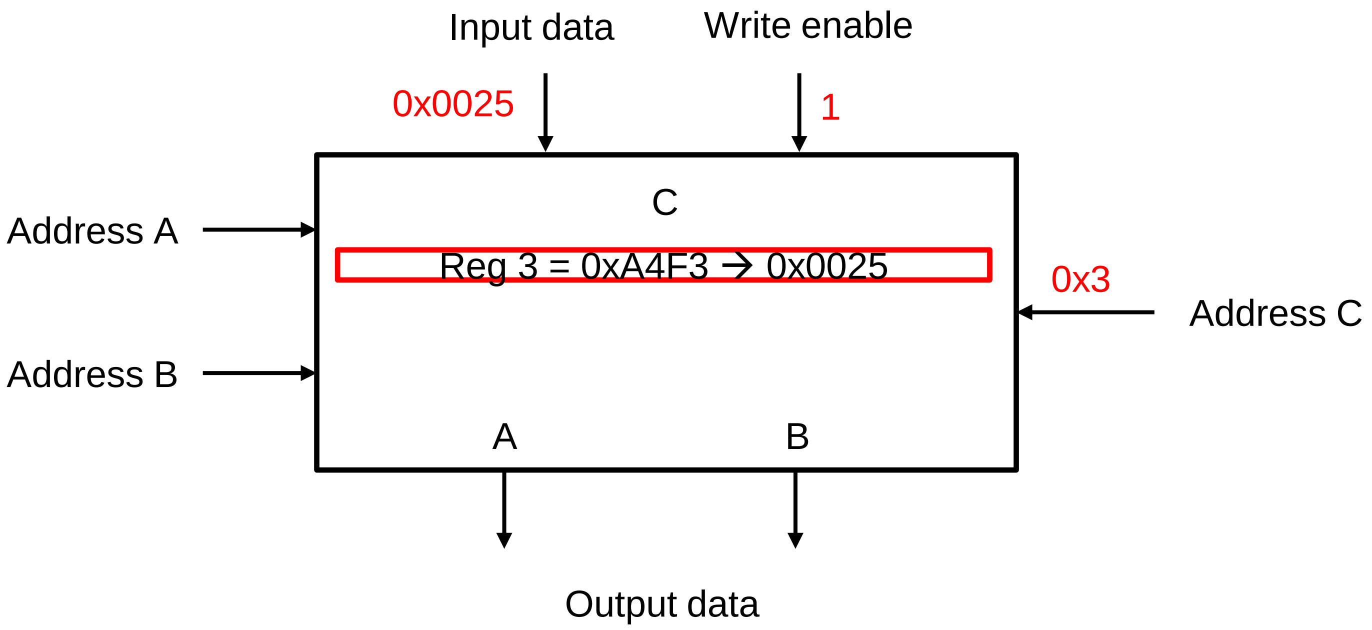
Register file block diagram example (read)

16x16-bit reg



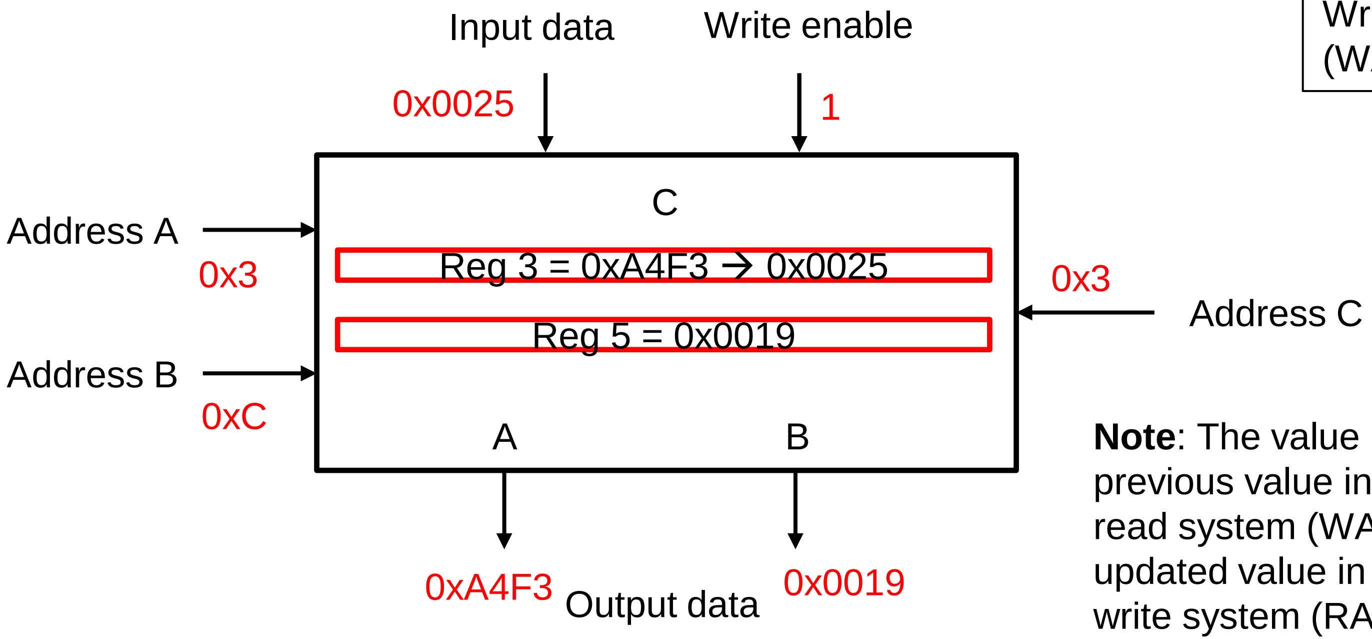
Register file block diagram example (write)

16x16-bit reg



Register file block diagram example (simultaneous r/w)

16x16-bit reg
Write after read
(WAR)



Note: The value read is the previous value in a Write after read system (WAR) and is the updated value in a read after write system (RAW).

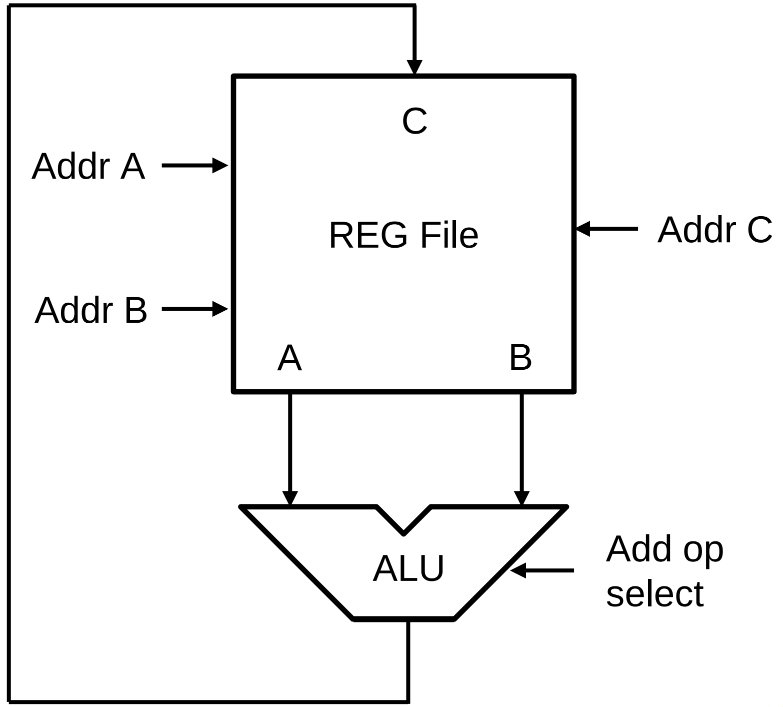
Regs and ALU

- We can combine registers with an ALU to make a rudimentary computer
- Data from a register (or 2) can be sent to an ALU in order to perform a computation
- The ALU will return a result. We can have this value stored to a new register or to one of the input registers.

Regs and ALU

- I want to add A + B and store it in C.
- The outputs of the read values of the reg file (A, B) are plugged into the ALU, where they get added (A+B)
- Within one clock cycle the values get output by the ALU and stored back in C

*In actual processors, the ALU can have different runtimes depending on operation. Therefore, in reality the “execute” step of the ALU is considered a separate clock cycle.



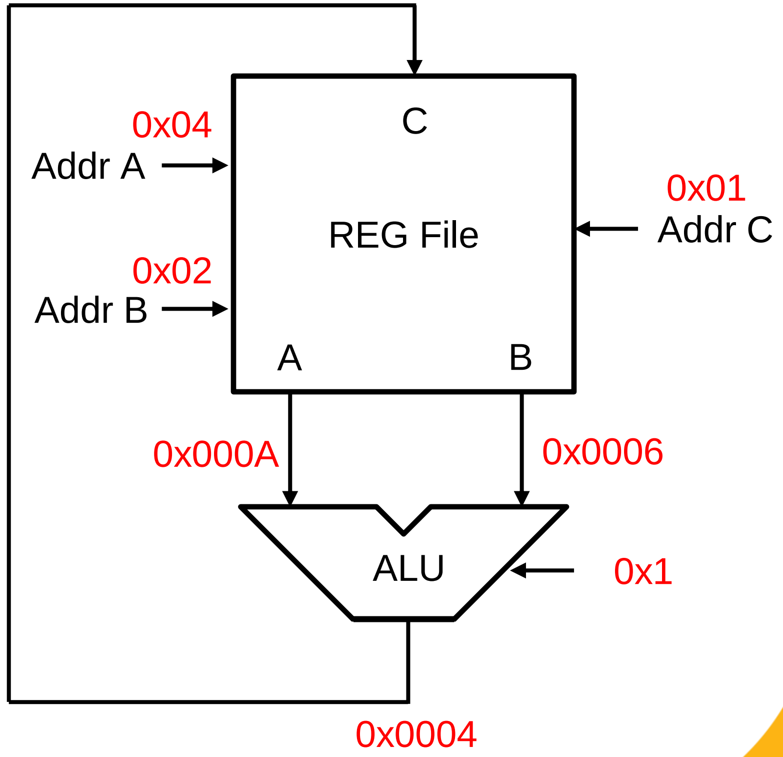
Regs and ALU example

- I want to subtract reg 2 (addr 2) from reg 4 (addr 4) and store the difference in reg 1 (addr 1)

  Reg 4 = 0x04, Reg 2 = 0x02,
  Reg 1 = 0x01
  value of reg 4 = 0x000A,
  value of reg 2= 0x0006,
  op = (subtract) 0x1 (given for this problem)

  reg1 = 0x0004

reg1 = reg4 – reg2



Summary

- Registers come in flavors relating to how the input and output are moved in and out (serial/parallel)
- Register file is a set of registers connected together in parallel
- Register file can be connected to ALU to make operations on specific variables

References

- https://electronics.stackexchange.com/questions/655621/help-with-register-file-implementation-on-logisim
- https://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html
- https://en.wikibooks.org/wiki/Microprocessor_Design/Register_File
- https://www.youtube.com/watch?v=7LmBcGiiYwk&list=PLmjEXDyU3L-mSz3eG4_JwVZt2fSon3tQX&index=56