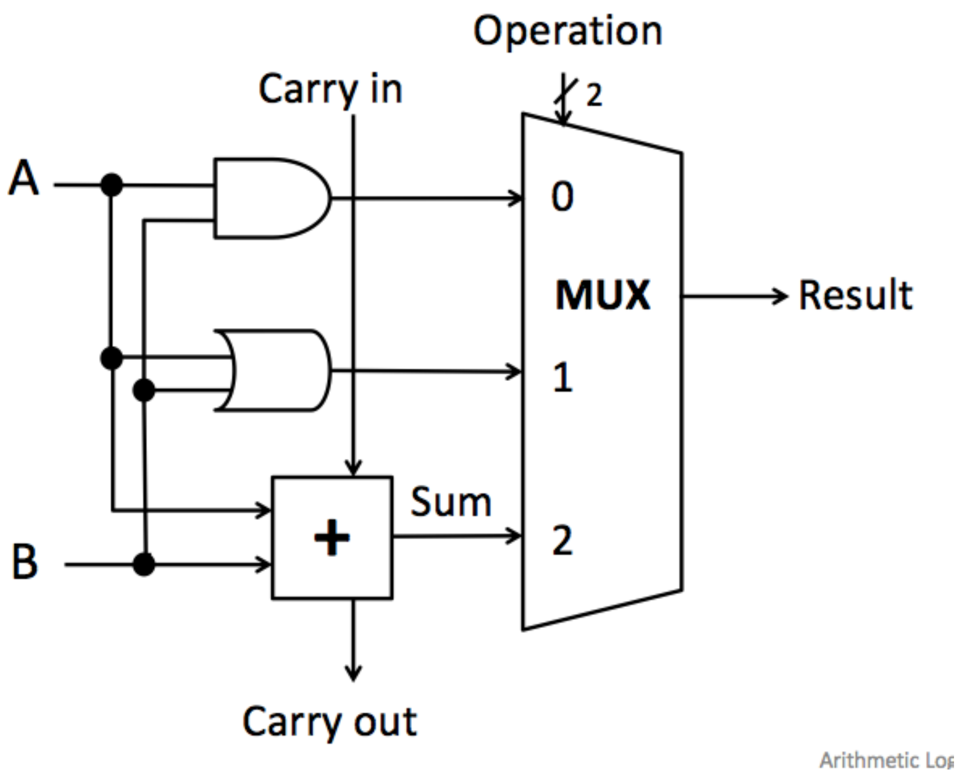
CS240 Laboratory 4
ALU and Sequential Logic

Arithmetic Logic Unit

An **ALU** is a combinational circuit used to perform all the arithmetic and logical operations for a computer processor.

A simple 1-bit ALU can be built using the basic components you have learned about: **AND** gate, **OR** gate, **1-bit adder**, and **multiplexer**:



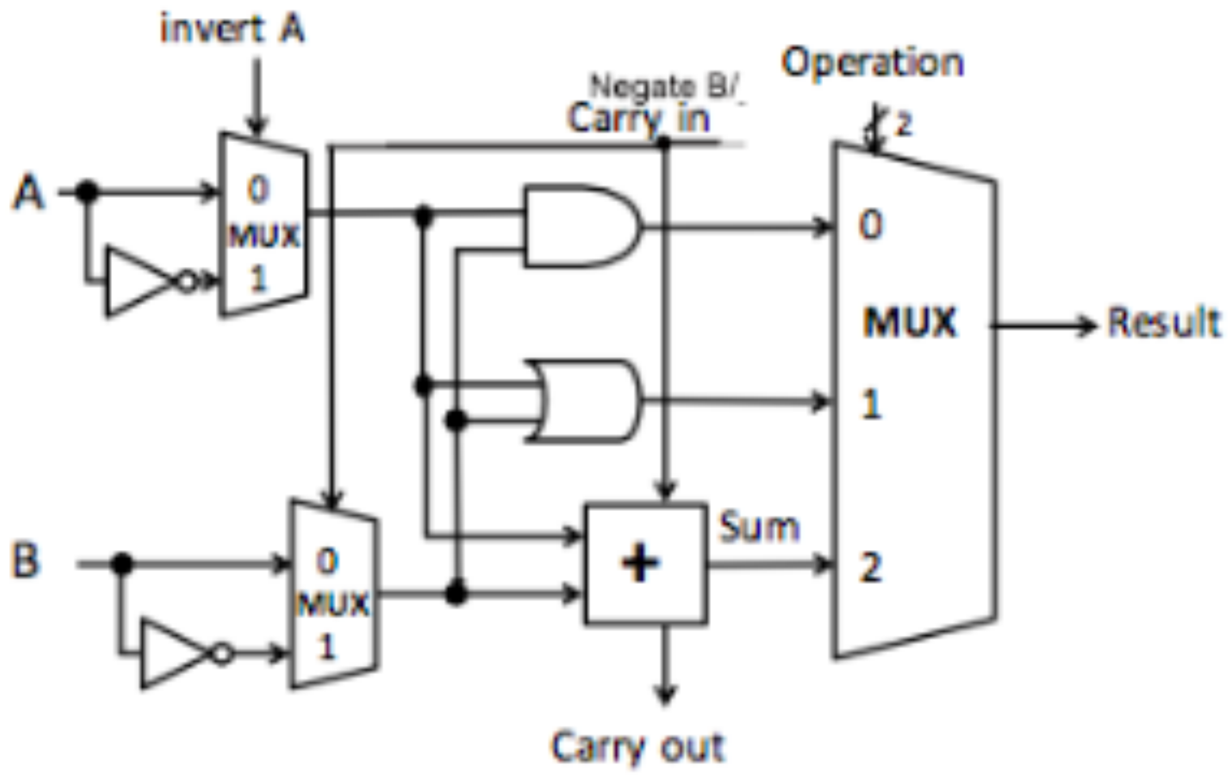
The **Operation (**a 2-bit value**)**, selects which operation should be produced as the **Result**:

Operation bits	Result
0 0	A AND B
0 1	A OR B
1 0	A + B

By adding some additional control inputs, it is possible to produce additional functions with the ALU.

Invert A is used to complement the input A.

Negate B/Carry in used to complement input B for logical operations, and as a carry-in when addition is performed.

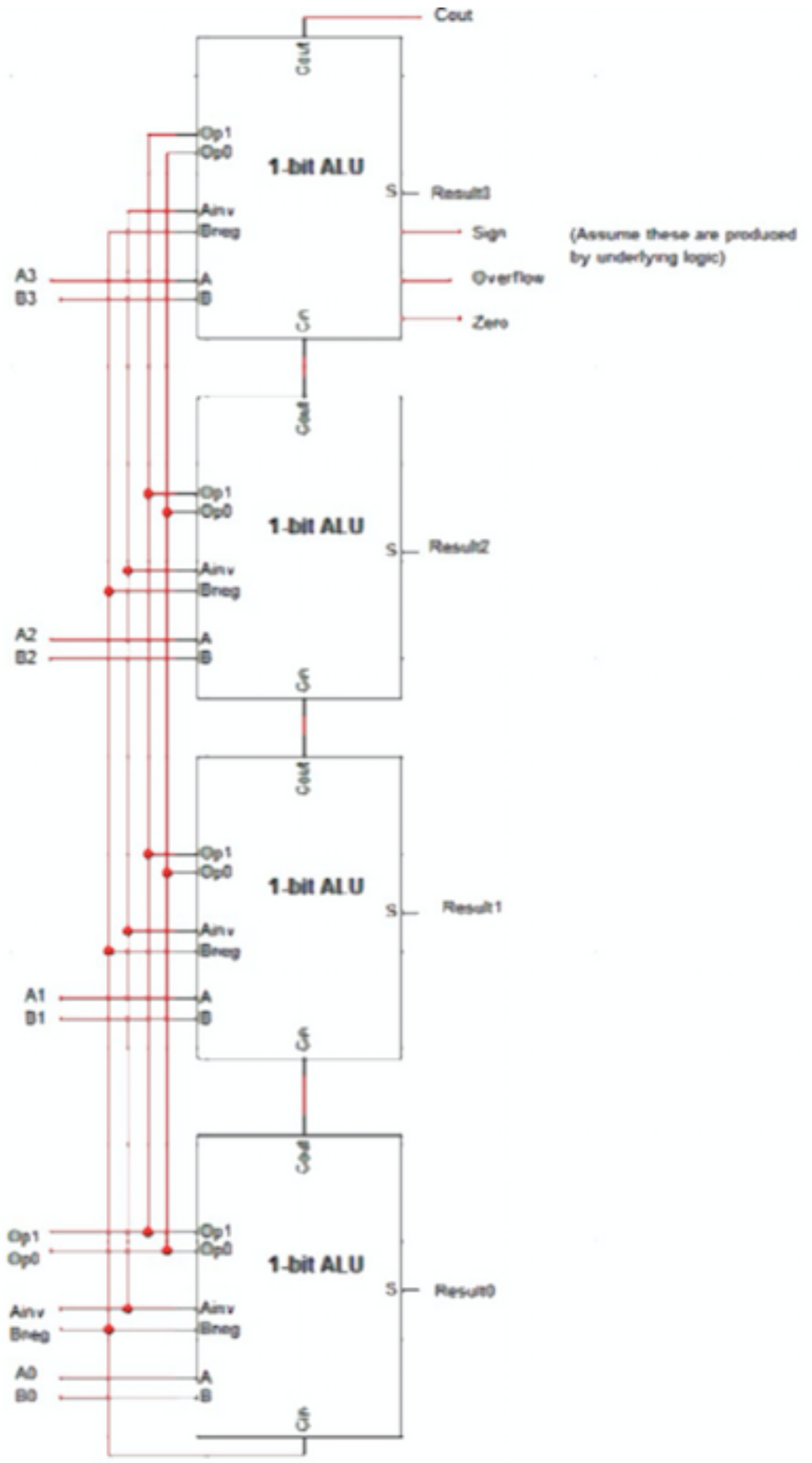


Basic Operations

- **add** (operation = add)
- **sub** (negate B/Carry in = 1, operation = ADD)
- **AND (**operation = AND**)**
- **OR (**operation = OR**)**
- **NOR (**invertA=1, negateB=1, operation = AND**)**

Control Inputs				**Result**
invertA	**negateB**	**Operation**		
0	0	0	0	a AND b
0	0	0	1	a OR b
0	0	1	0	a + b
0	1	1	0	a – b
1	1	0	0	a NOR b

A 4-bit ALU can be built from 4 1-bit ALUs in the same way that a 4-bit adder can be built from 1-bit adders:

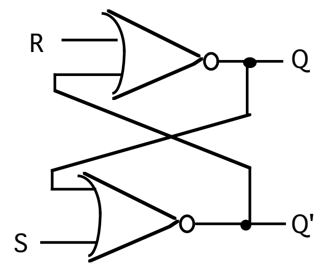


How do you produce the Sign, Overflow, and Zero bits?

Basic Memory Circuit

Latch Single-bit memory, level-triggered

SR (Set Reset) Latch



S	**R**	**Q**	**Q′**
0	0	Qp	Qp′ remember
0	1	0	reset (clear)
1	0	1	set
1	1	unpredictable	

What does **unpredictable** mean? Notice in a NOR gate, if either input = 1 to a gate, its output = 0 (**1** is a deterministic input):

A	**B**	**(A+B)′**
0	0	1
0	1	0
1	0	0
1	1	0

So, although you wouldn’t usually try to *set* and *reset* at the same time (it doesn’t make sense), if you did, Q and Q′ will both be 0 (which is not unpredictable).

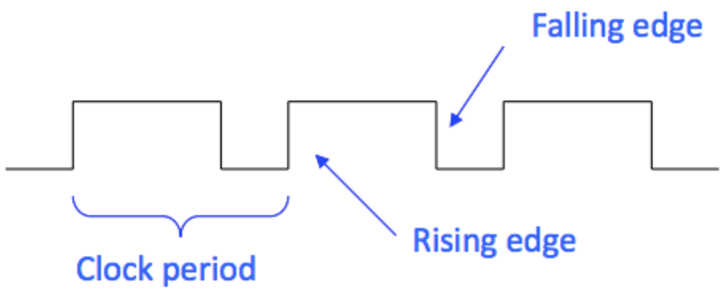
However, when you go back to the *remember* state (S=R=0), Q and Q′ will not stay at 0 0. The circuit passes through one of either the *set* or *reset* state on its way back to the *remember* state, and Q and Q' change to the complement of one another.

Since the final state depends on which transitional state was sensed on the way back to *remember*, you cannot predict whether the final state of Q will be 1 or 0.

Clocked SR Latch

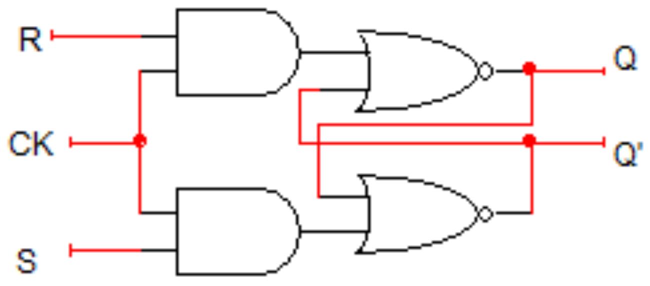
To synchronize when the latch changes state, add a **clock** input:

Clock: free-running signal
with fixed cycle time = clock period = T.
Clock frequency = 1 / clock period

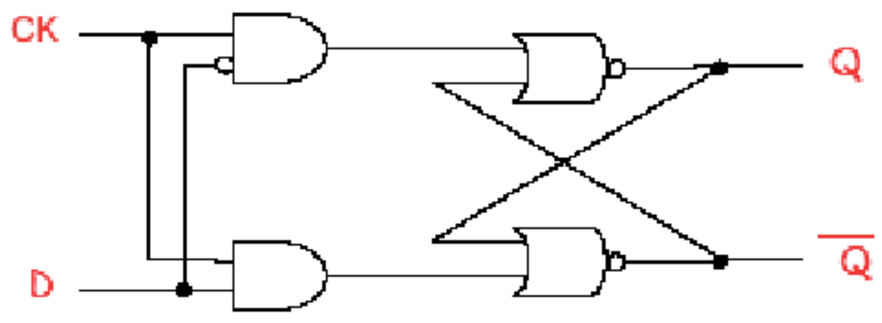


A clock controls when to update
a sequential logic element's state.

In a clocked latch, whenever the clock is high, the outputs/state of the latch can change.



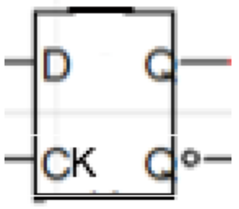
D Latch
The D latch is another 1-bit clocked memory device. It avoids the unpredictable state S=R=1 of the SR latch, because a single input D determines the next state of the circuit.



D	Q
0	0
1	1

Q gets the value of D when the CK is high.

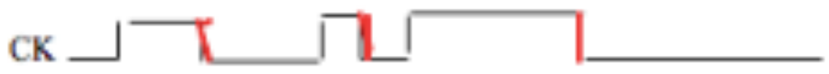
The D latch circuit can be abstracted to the following:



D Flip-Flop
Similar to a latch, a flip-flop is also a 1-bit memory.

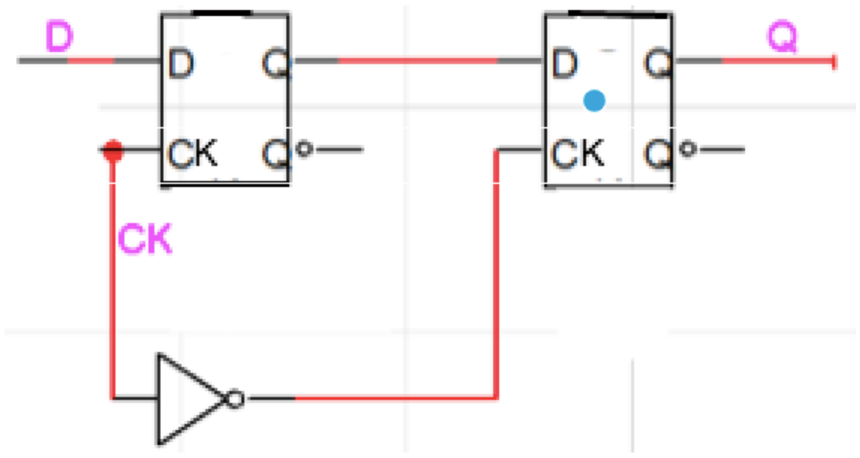
Rather than allowing change of state any time the clock is high (as for the latch), in the flip-flop, the change of state occurs on a clock *edge.*

The *falling edge* of the clock is the exact transition from high-to-low, rather than whenever the clock is high (the negative edges are marked in red below):



```
Internally, a flip-flop is actually made from 2 latches.

The first latch is controlled directly by the clock, but
the second latch is controlled by the inverse of the clock:
```



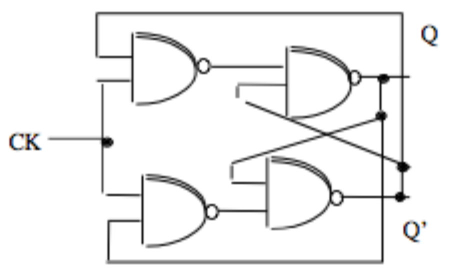
```
So, the input D will not be passed from the first latch to
the second latch until the clock goes low.

Once the clock is low, a new value on D will not store into
the first latch.  Overall, the flip-flop can change value
only exactly at the transition of the clock from high to
low.
```

T Flip-Flop

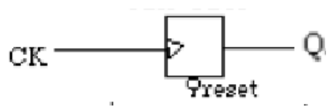
Another type of flip-flop which avoids the unpredictable state is a **T flip-flop.** It only has a clock input, and simply toggles to the opposite of the current state when it is clocked because the values of the current outputs are tied back into the inputs for the next state.

The T latch, upon which the flip-flop is based, looks like this:



Qprev	Qnext
0	1
1	0

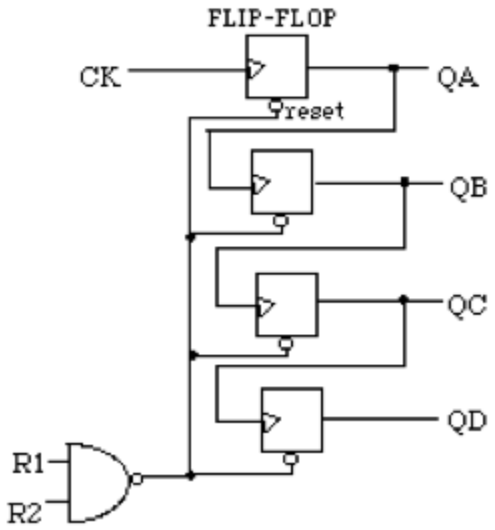
The T flip-flop can be abstracted to the following device (the **reset** input allows you to initialize the device to a value of 0):



In lab 2, we investigated the operation of a combinational circuit called a **binary counter**, which produces a 4-bit value which represents the sequence of binary numbers from 0 to 15.

At the time, we did not explain the details of the underlying circuit of the binary counter, because it uses flip-flops, which we had not yet discussed.

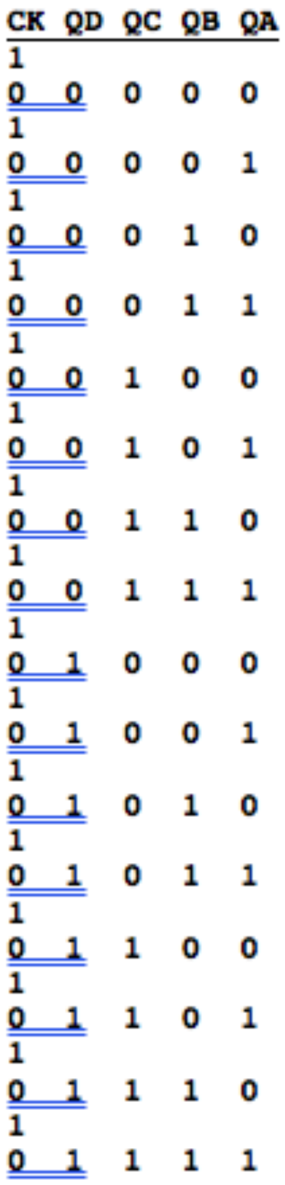
The **binary counter** uses 4 interconnected **T** flip-flops:



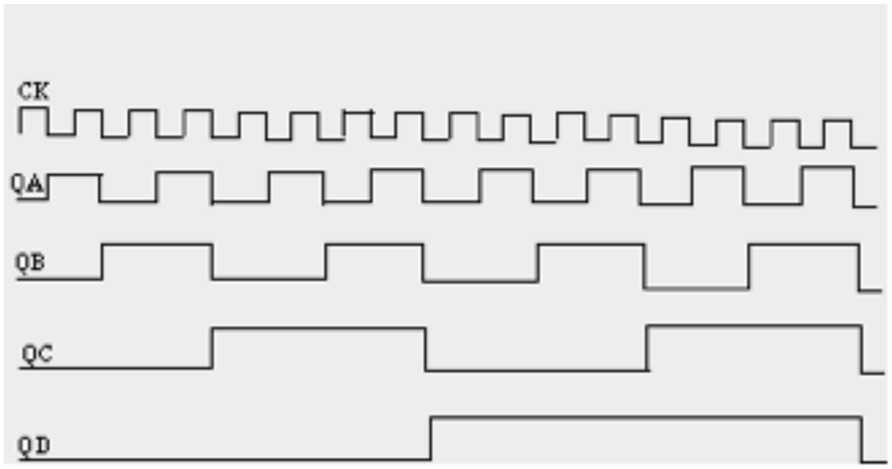
The output of the first T flip-flop, QA, serves as the clock to QB. So, QB only changes when QA falls from 1 to 0 (on the negative edge). QB therefore only changes half as frequently as QA.

A similar relationship exists for QB to QC, and QC to QD.

The pattern of outputs then represents the binary numbers, since that is exactly how the digits change as the numbers increment:

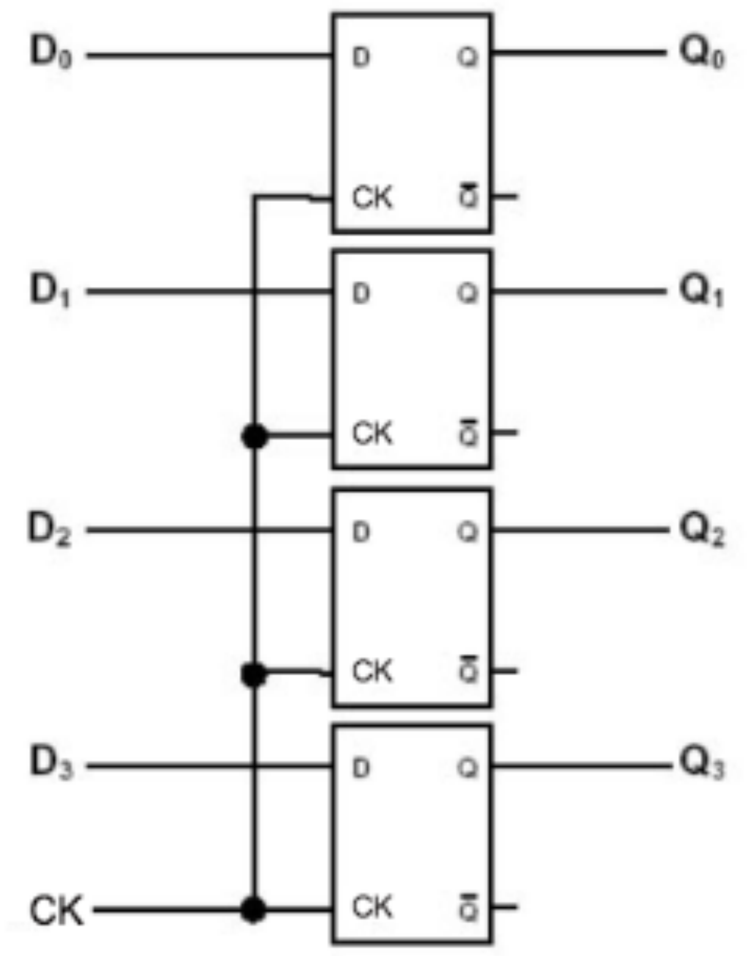


CK	QD	QC	QB	QA
1				
0	0	0	0	0
1				
0	0	0	0	1
1				
0	0	0	1	0
1				
0	0	0	1	1
1				
0	0	1	0	0
1				
0	0	1	0	1
1				
0	0	1	1	0
1				
0	0	1	1	1
1				
0	1	0	0	0
1				
0	1	0	0	1
1				
0	1	0	1	0
1				
0	1	0	1	1
1				
0	1	1	0	0
1				
0	1	1	0	1
1				
0	1	1	1	0
1				
0	1	1	1	1



Memory Devices using Flip-flops

Register - n-bit memory, uses *n* flip-flops, and a shared *clock* input. Shown below is a 4-bit register:



Registers are used to represent single n-bit value in a Computer Processing Unit (CPU).

Register File

The CPU contains a set of registers to hold values which are being used to execute an instruction.

This set of registers is called a **register file.**

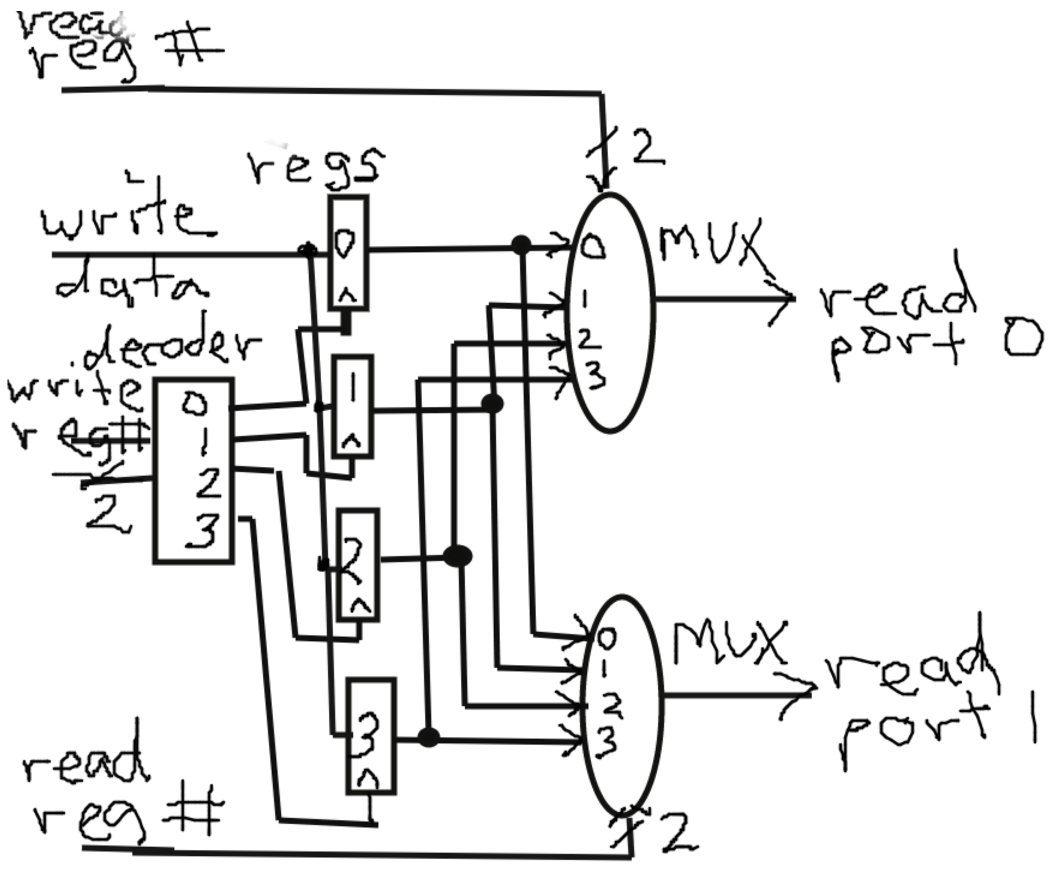
The register file contains multiple registers (although typically a limited number) and so is a larger memory than a single register.

However, in addition to the set of registers, the register file also contains circuits (multiplexers and decoders) which select which registers to read from or write to for a particular instruction.

A block diagram for a register file containing 4 registers is shown below.

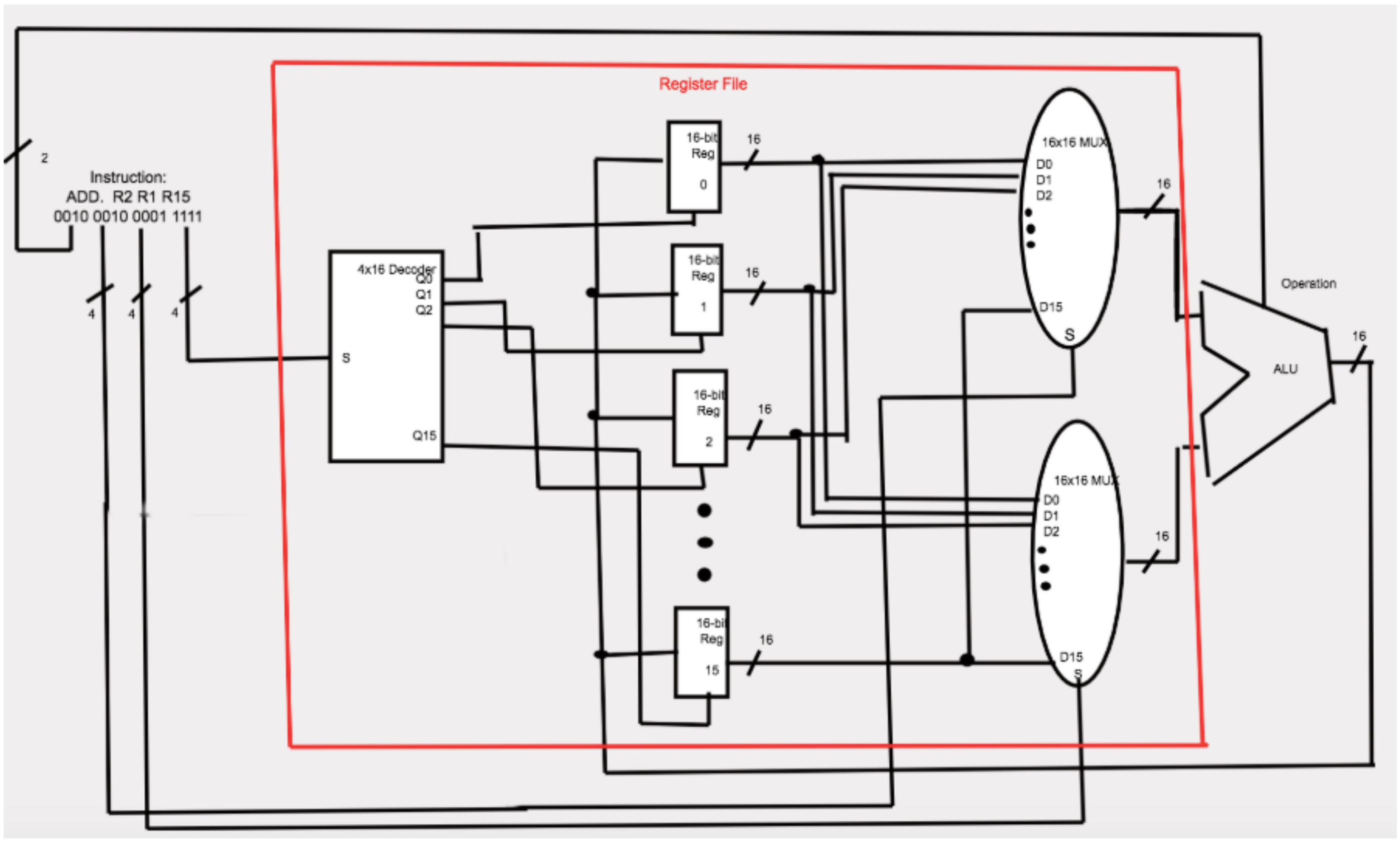
Notice the two 4x1 multiplexers on the right: these select two registers from the set of 4 to read from at any given time. The 2x4 decoder on the left selects a single register to be written to with a new value at any given time.

Note: a **bold** black line with **/2** through it would indicate that there are actually 2 wires/bits represented by the line



Although a register file contains multiple registers, it is still a fairly small memory, and is a special purpose circuit for the CPU.

Below is a circuit using a register file that represents the data path for the CPU we will be studying:

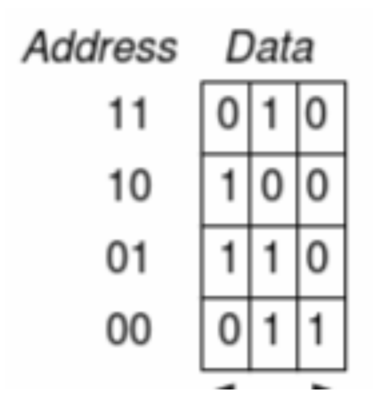

Register File
Instruction:
ADD. R2 R1 R15
0010 0010 0001 1111
2
4
4
4
4x16 Decoder
Q0
Q1
Q2
Q15
S
16-bit Reg 0
16-bit Reg 1
16-bit Reg 2
16-bit Reg 15
16
16x16 MUX
D0
D1
D2
D15
S
Operation
ALU

RAM memory

RAM (Random Access Memory) memories are general purpose, and can be used to store a large set of values (which can represent either instructions in a program or data).

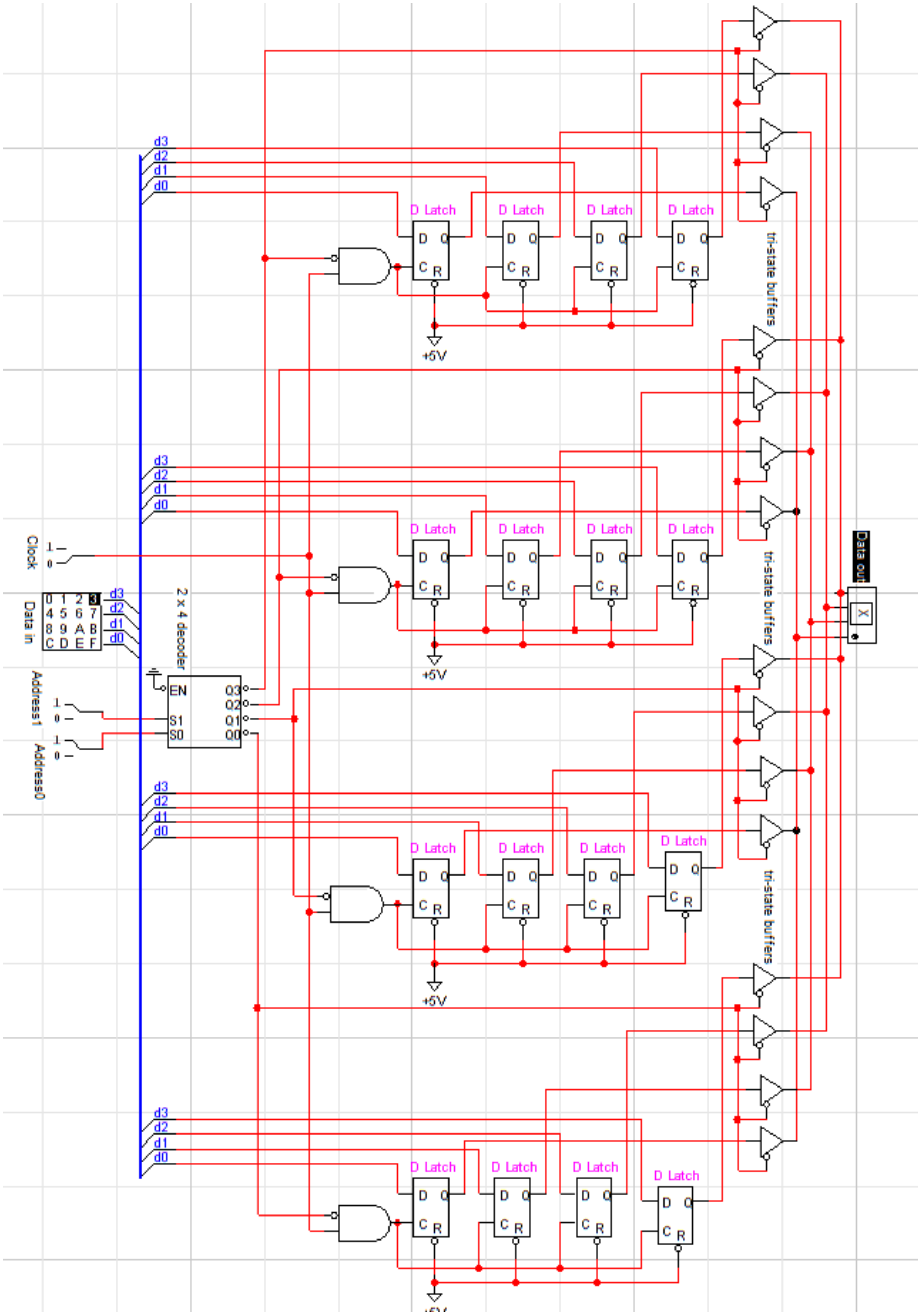
A RAM memory contains multiple flip-flops, organized into n-bit words, where each word can be accessed through use of an address. Another way to think of a memory is as an *array* of n-bit values. You access the value you want by specifying its index, which we refer to as an *address.*

In the diagram below, there are 4 **addresses** (specified as 00, 01, 10, and 11). Each address contains 3 bits of **data** (a flip-flop would be used to store each data bit of 1 or 0).

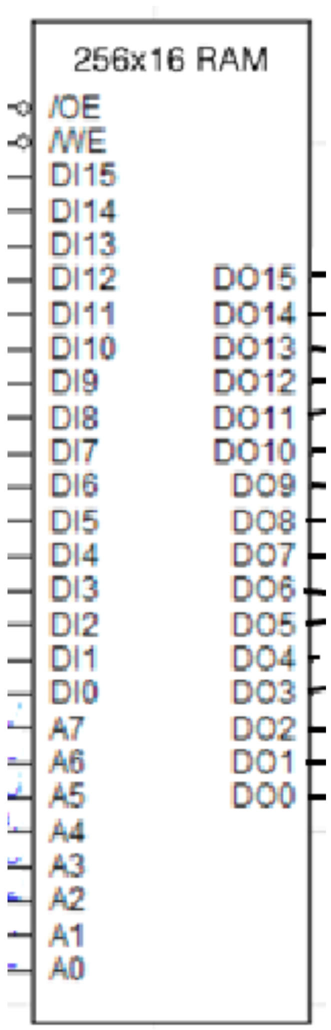


Address	Data		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

In addition to the flip-flops, a memory also contains multiplexer and decoder circuits to select which value is being accessed:



Clock
1
0
Data in
0 1 2 3
4 5 6 7
8 9 A B
C D E F
d3
d2
d1
d0
2 x 4 decoder
EN
S1
S0
Q3
Q2
Q1
Q0
Address1
Address0
D Latch
D
Q
C
R
+5V
tri-state buffers
Data out
X



A 256x16 RAM chip contain 256 16-bit values.

To **read** one of the stored values, you specify the address of the value on the A inputs, and the data at that address is read from the D0 outputs.

To **write** a new value to the memory:

- Specify the new value on the DI inputs.
- Specify the address where the new value will be stored on the A inputs.
- Activate the /WE control signal (which is basically the clock input to the flip-flops in the device).