BCN 1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

BCN 1043

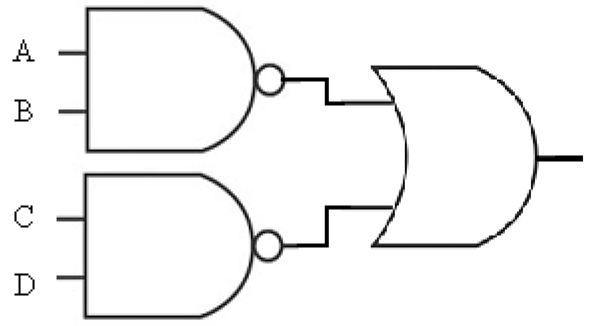
COMPUTER ARCHITECTURE & ORGANIZATION

Chapter 3 continues…

COMBINATIONAL CIRCUITS

Combinational Circuits

- We will combine logic gates together for calculations
  - Example: ~(A*B) and ~(C*D) with an OR gate



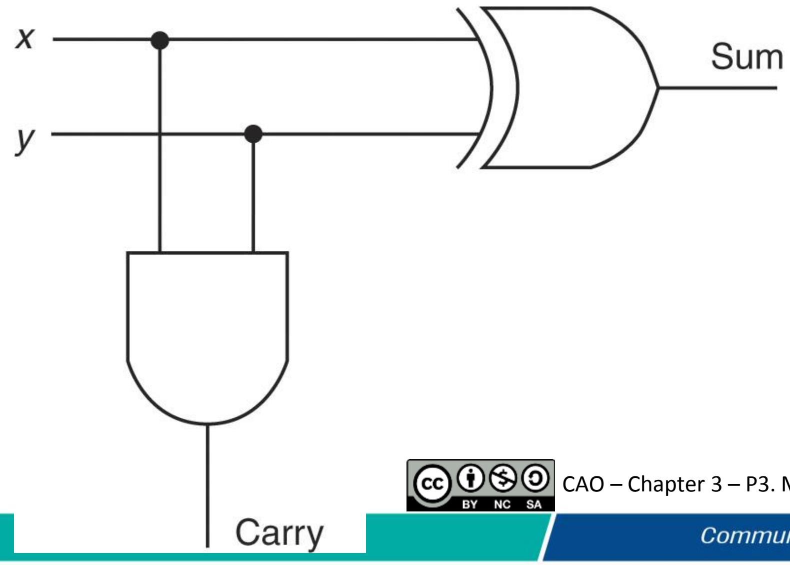
- The resulting circuit is a combinational circuit
  - Electrical current flows from one gate to the next
  - By combining gates, we can compute a boolean expression
  - What we want to do is:
    - Derive the boolean expression for some binary calculation (e.g., addition)
    - Then build the circuit using the various logic gates
  - This is how we will build the digital circuits that make up the ALU (arithmetic-logic unit) and other parts of the computer

An Example: Half Adder

- There are 4 possibilities when adding 2 bits together:
  - ➢ 0 + 0   0 + 1   1 + 0   1 + 1
    - In the first case, we have a sum of 0 and a carry of 0
    - In the second and third cases, we have a sum of 1 and a carry of 0
    - In the last case, we have a sum of 0 and a carry of 1
    - These patterns are demonstrated in the truth table above to the right
  - ➢ Note: sum computes same as XOR
  - ➢ carry computes the same as AND
  - ➢ Adder is built using just one XOR and one AND gate

Inputs		Outputs	
x	*y*	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

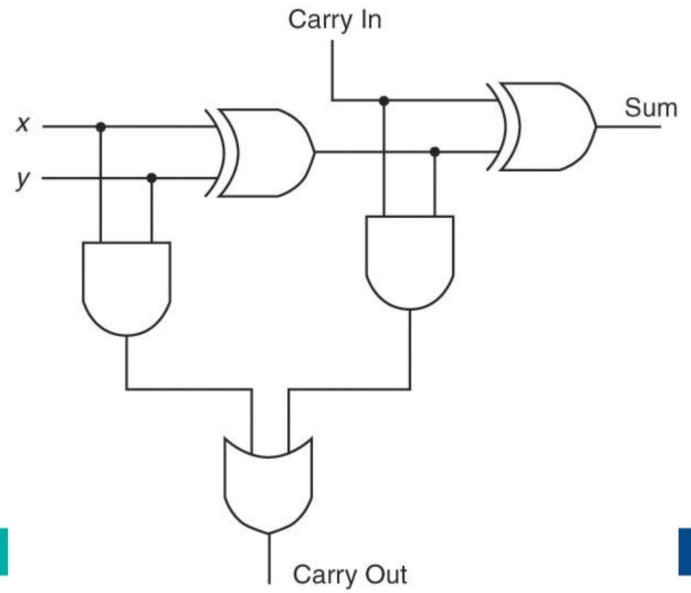
The truth table for Sum and Carry and a circuit to compute these



Full Adder

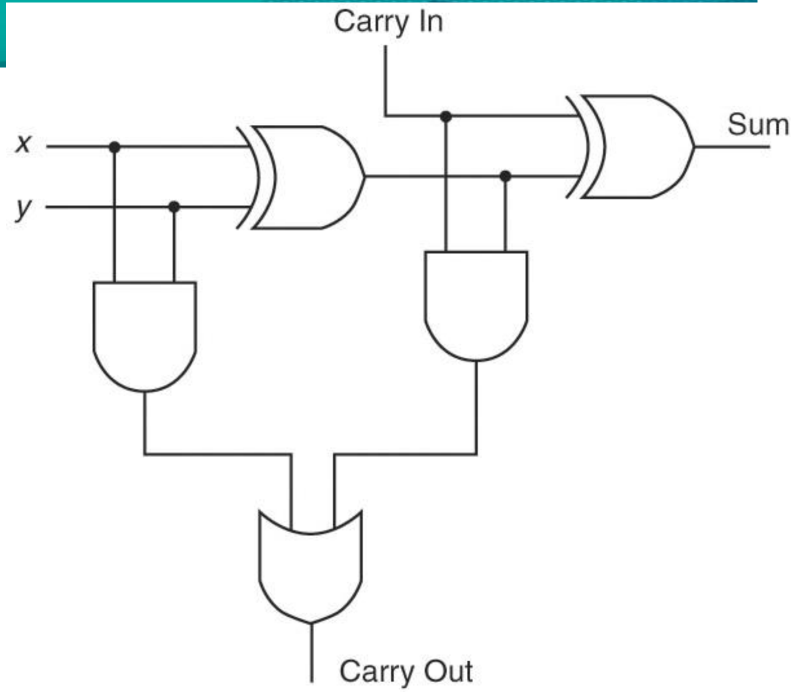
- The half adder really only does half the work
  - adds 2 bits, but only 2 bits
- If we want to add 2 n-bit numbers, we need to also include the carry in from the previous half adder
  - So, our circuit becomes more complicated
- In adding 3 bits (one bit from x, one bit from y, and the carry in from the previous addition), we have 8 possibilities
  - The sum will either be 0 or 1 and the carry out will either be 0 or 1

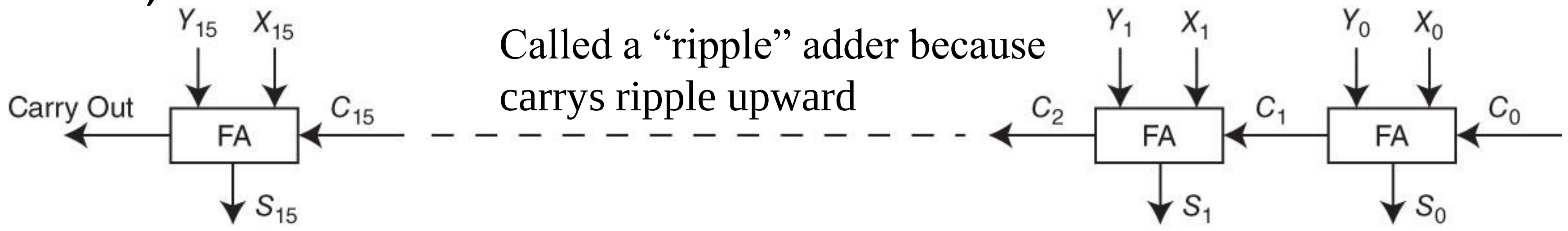
Inputs			Outputs	
x	*y*	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Building a Full Adder Circuit

- The sum is 1 only if one of x, y and carry in are 1, or if all three are 1, the sum is 0 otherwise
- The carry out is 1 if two or three of x, y and carry in were 1, 0 otherwise
  - The circuit to the right captures this by using 2 XOR gates for Sum and 2 AND gates and an OR gate for Carry Out
- We combine several full adders together to build an Adder, as shown below:



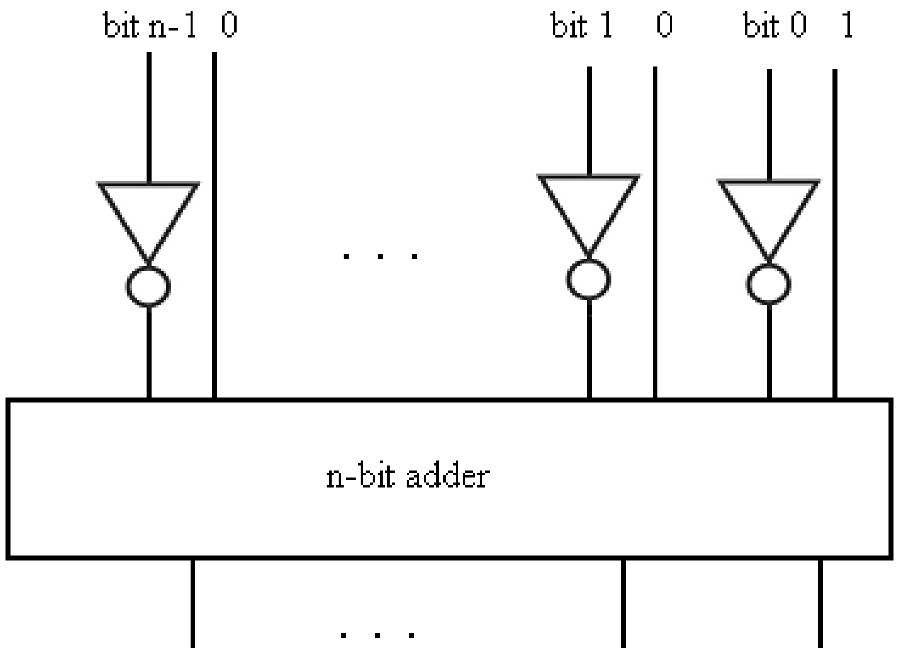


Called a "ripple" adder because carrys ripple upward

A 16-bit adder, comprised of 16 Full Adders connected so that each full adder's carry out becomes the next full adder's carry in

Complementor

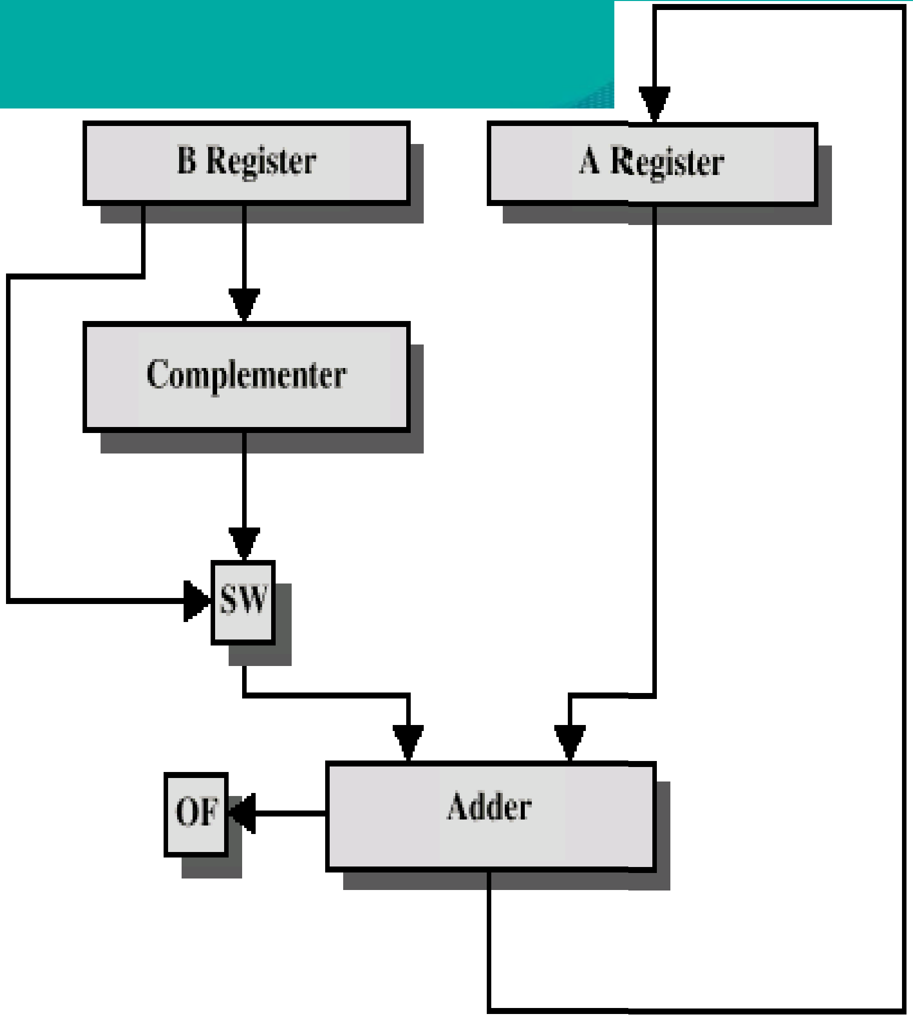
- Let’s design another circuit to take a two’s complement number and negate it (complement it)
  - Change a positive number to a negative number
  - Change a negative number to a positive number
- Recall to do this, you flip all of the bits and add 1
  - To flip the bits, we pass each bit through a NOT gate
  - To add one, send it to a full adder with the other number being 000…001



Adder/Subtractor

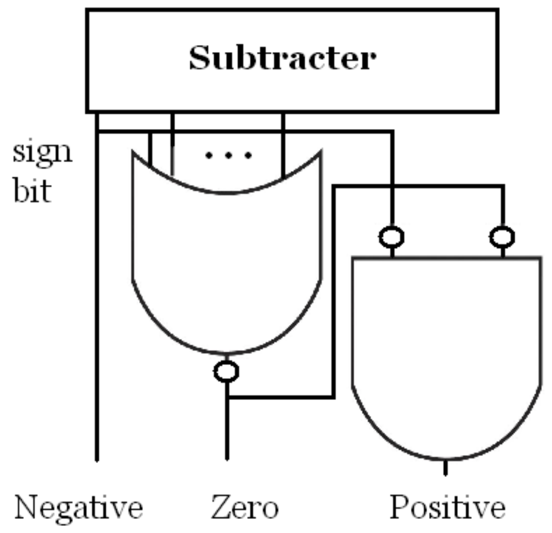
- Recall from chapter 2
  - two's complement subtraction can be performed by negating the second number and adding it to the first
    - We revise our adder as shown to the right
      - It can now perform addition (as normal)
      - Or subtraction by sending the second number through the complementor

The switch (SW) is a multiplexer, covered in a few slides



OF = overflow bit
SW = Switch (select addition or subtraction)

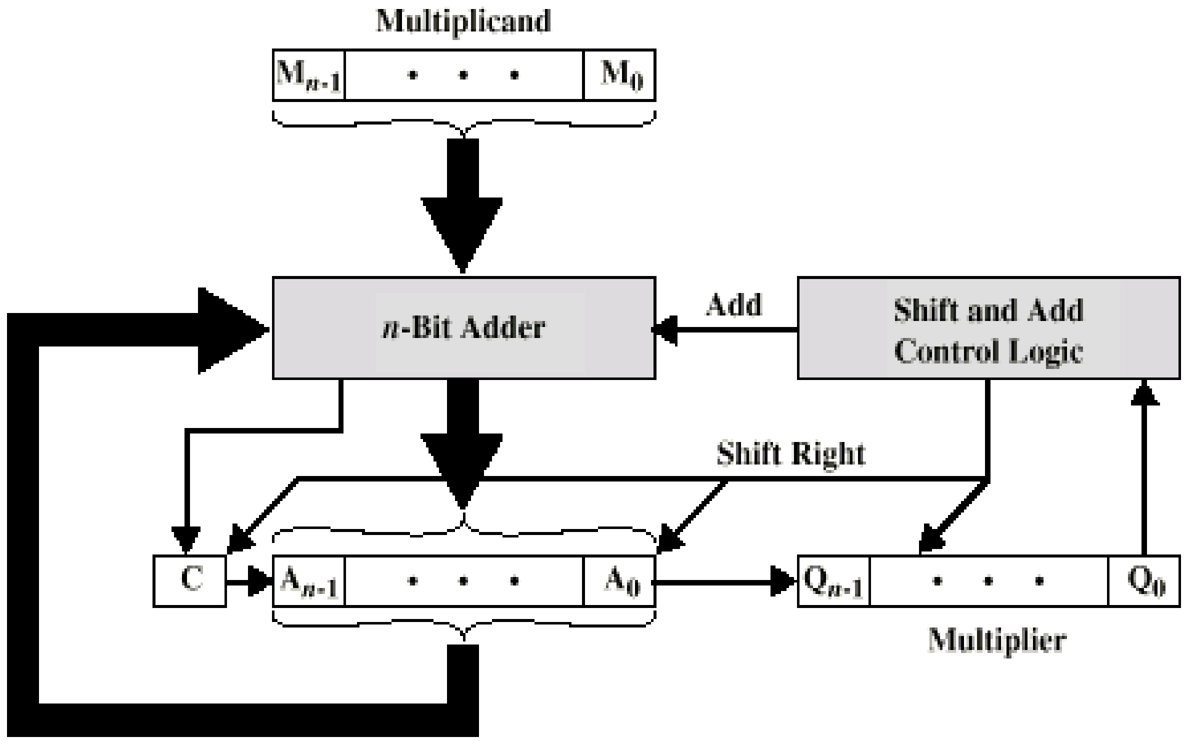
Comparator



- We have covered + and -, how about <, >, =
- To compare A to B, we use a simple tactic
  - Compute A – B and look at the result
    - if the result is -, then A < B
    - if the result is 0, then A = B
    - if the result is +, then A > B
    - if the result is not 0, then A != B
  - how do we determine if the result is -? look at the sign bit, if the sign bit is 1, then the result is negative and A < B
  - how do we determine if the result is 0? are all bits of the result 0? if so, then the result is 0 and A = B
    - we will build a zero tester which is simply going to NOR all of the bits together
  - how do we determine if the result is +? if the result of A – B is not negative and not 0, it must be positive, so we negate the results of the first two and pass them through an AND gate
- The comparator circuit is shown above (notice that the circuit outputs 3 values, only 1 of which will be a 1, the others must be 0)
  - NOTE: to compute !=, we can simply negate the Zero output

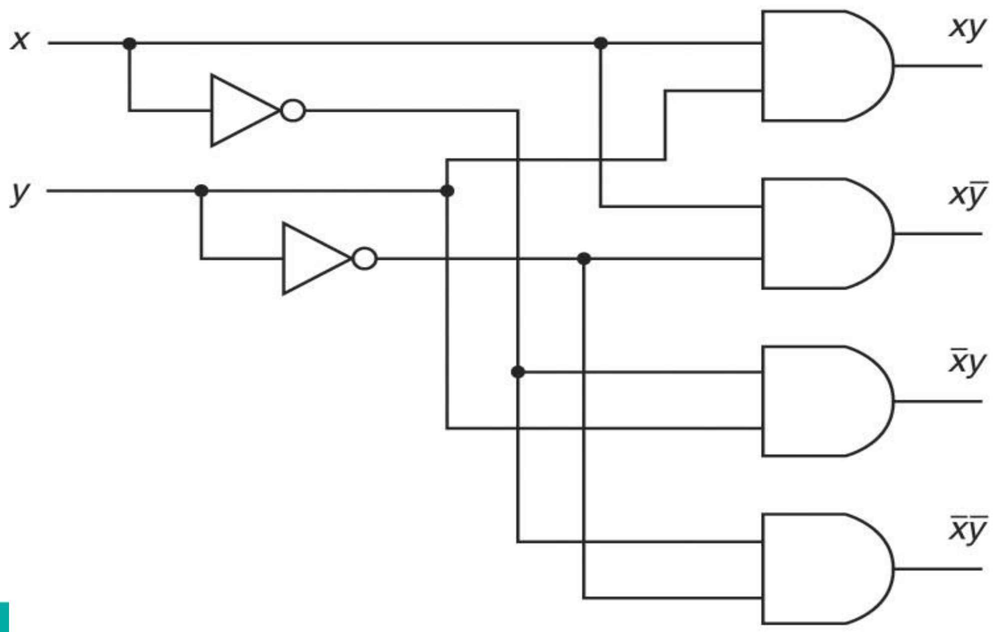
Multiplier

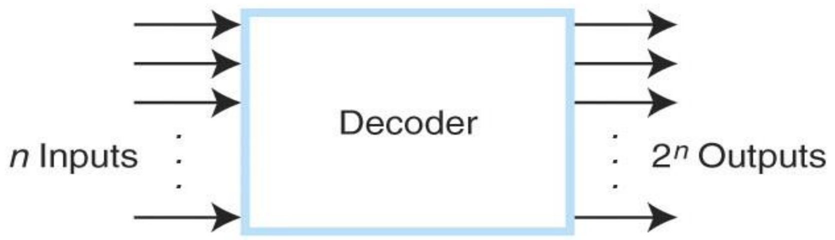
- The circuit below is a multiplication circuit
  - Given two values, the multiplicand and the multiplier, both stored in temporary registers
  - The addition takes place by checking the Q0 bit and deciding whether to add the multiplicand to the register A or not, followed by right shifting the carry bit, A and Q together
- shift/add control logic
  - set counter = n
  - compare Q0 to 1
    - if equal, signal adder to add
  - signal the shifter to shift
  - decrement counter
  - repeat until counter = 0



A Decoder

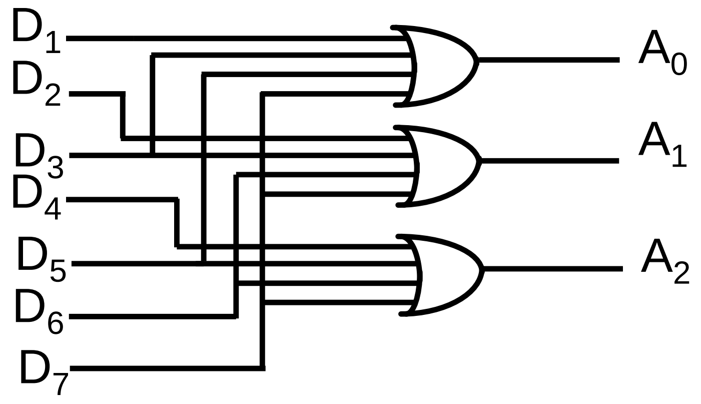
- The Decoder is a circuit that takes a binary pattern and translates it into a single output
  - This is often used to convert a binary value into a decimal value
    - For an n-bit input, there are 2^n outputs
    - Below is a 2 input – 4 output decoder
      - if input = 01, the second line (x*~y) on the right has current
      - the line 01 would be considered line 1, where we start counting at 0





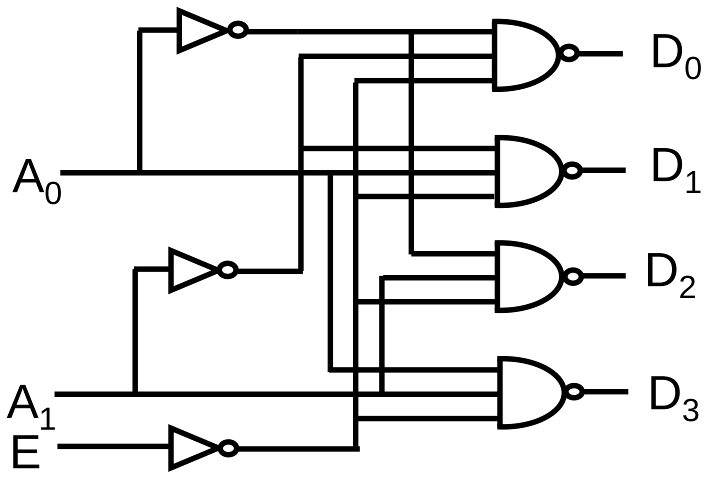
ENCODER/DECODER

Octal-to-Binary Encoder



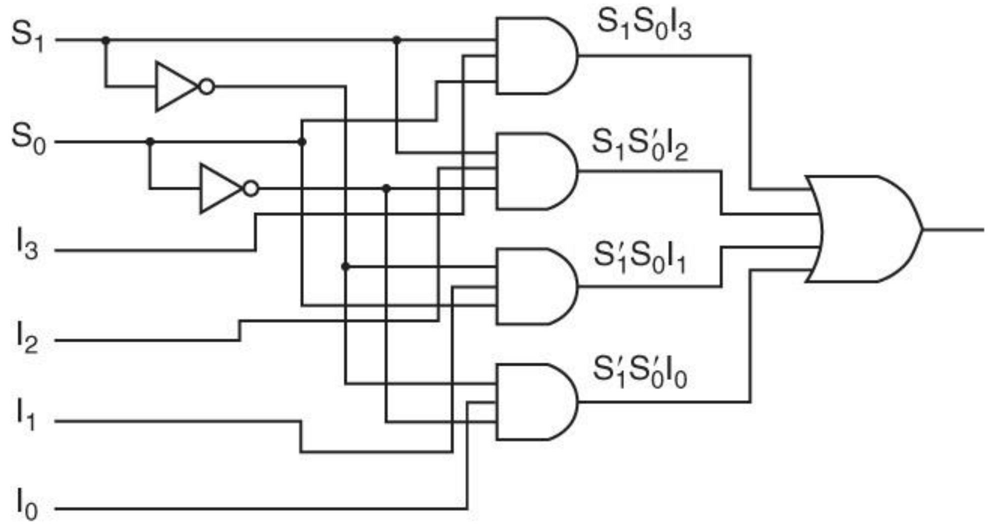
2-to-4 Decoder

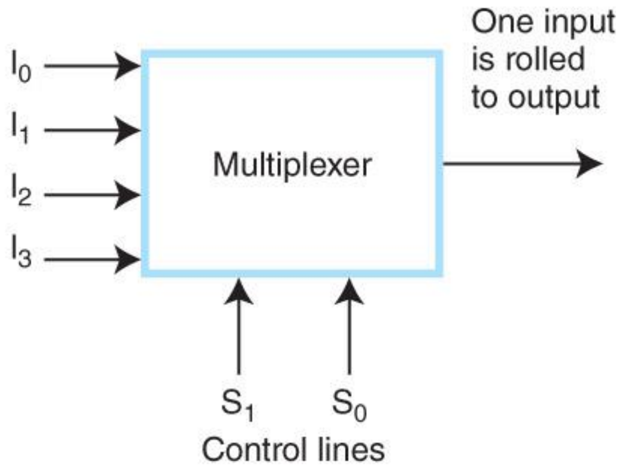
E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	d	d	1	1	1	1



A Multiplexer

- Multiplexer (abbreviated as MUX) is used to select from a group of inputs which one to pass on as output
  - Here, 1 of 4 single-bit inputs is passed on using a 2-bit selector (00 for input 0, 01 for input 1,10 for input 2, 11 for input 3)
    - While this circuit is more complex than previous ones, this is simplified for a MUX – imagine what it would look like if we wanted to pass on 16 bits from 1 of 4 inputs





A related circuit is the de-multiplexer (DEMUX) – it receives 1 input and a select and passes the input onto one of several outputs

A Simple 2-bit ALU

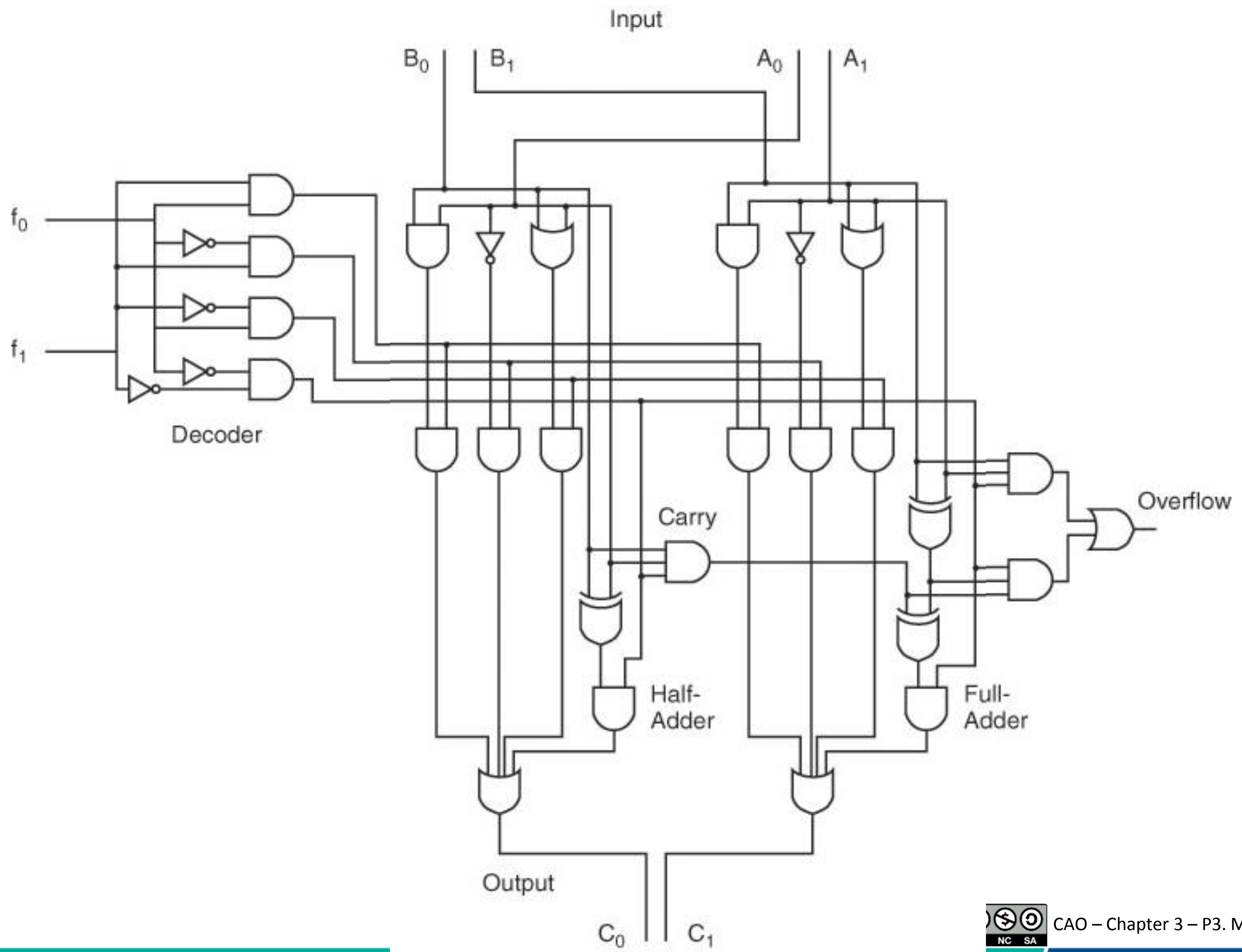
Putting all these ideas together

We have a 2-bit ALU

Given 2 values, A and B, each of which are 2 bits (A0, A1, B0, B1) and a selection from the control unit (f0, f1)

This circuit computes
A+B (if f0 f1 = 00)
NOT A (if f0 f1 = 01)
A OR B (if f0 f1 = 10)
A AND B (if f0 f1 = 11)

And passes the result out as C0 C1 along with overflow if the addition caused an overflow



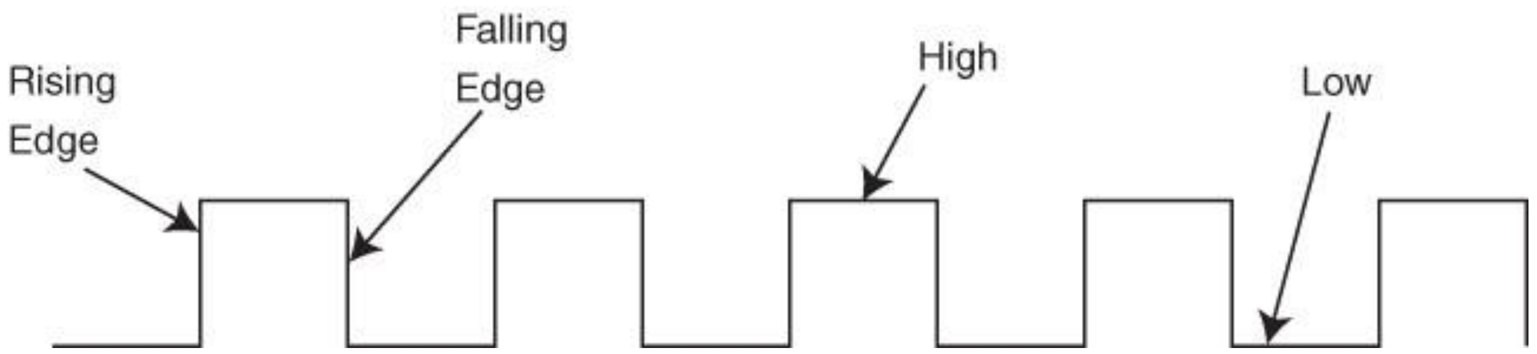
SEQUENTIAL CIRCUITS

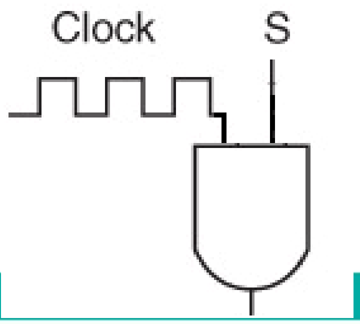
Sequential Circuits

- All of the previous circuits were combinational circuits
  - Current flowed in at one end and out the other
  - Combinational circuits cannot *retain* values
  - If we want to build a kind of memory, we need to use a sequential circuit
    - In a sequential circuit, current flows into the circuit and stays there
    - This is done by looping the output back into the input
  - Sequential circuits will be used to implement 1-bit storage
  - We can then combine 1-bit storage circuits into groups for n-bit storage (registers, cache)
    - These circuits will be known as flip-flops because they can flip from one state (storing 1) to another (storing 0) or vice versa

The Clock

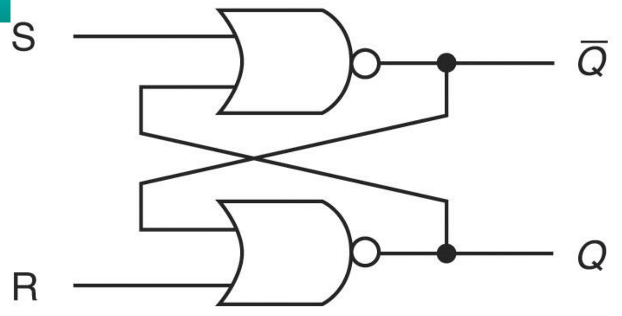
- The clock will control when certain actions should take place
- The clock simply generates a sequence of electrical current “pulses”
- In the figure below, when the line is high, it means current is flowing, when low it means current is not flowing
  - Thus, if we want to control when to shift, we connect the S input to an AND gate that includes the clock as another input





We will use the clock to control a number of things in the CPU, such as flip-flop changes of state, or when ALU components should perform their operation

An S-R Flip-Flop

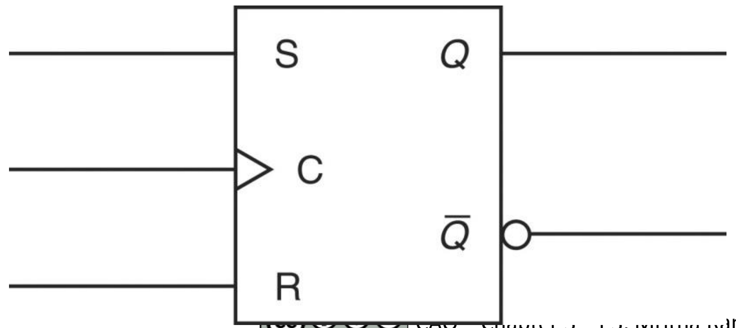


S	R	Q (t+1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	undefined

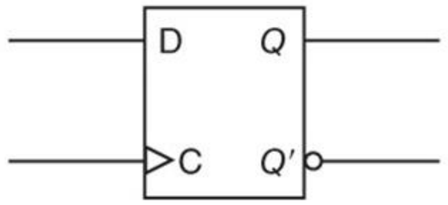
- The S-R flip-flop has 2 inputs and 2 outputs
  - The 2 inputs represent Set (storing a 1 in the flip-flop) and Reset (storing a 0 in the flip-flop)
  - It has two outputs although Q is the only one we will regularly use
  - To place a new value in the flip-flop, send a current over either S or R depending on the value we want, to get a value, just examine Q

Note that the S-R flip-flop is not controlled by the clock

the S-R flip-flop circuit diagram and truth table are given above, and can be represented abstractly by the figure to the right



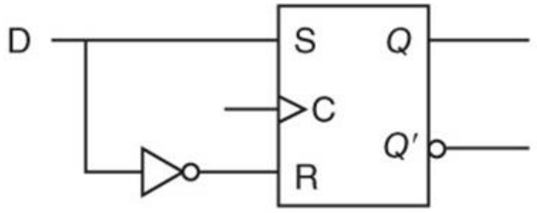
D and JK Flip-Flops



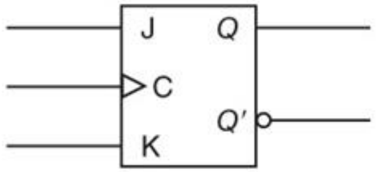
a)

D	Q(t + 1)
0	0
1	1

b)



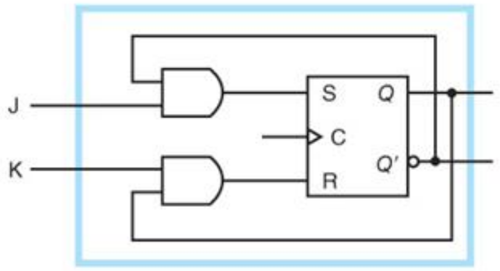
c)



a)

J	K	Q(t +1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	$Q'(t)$

b)

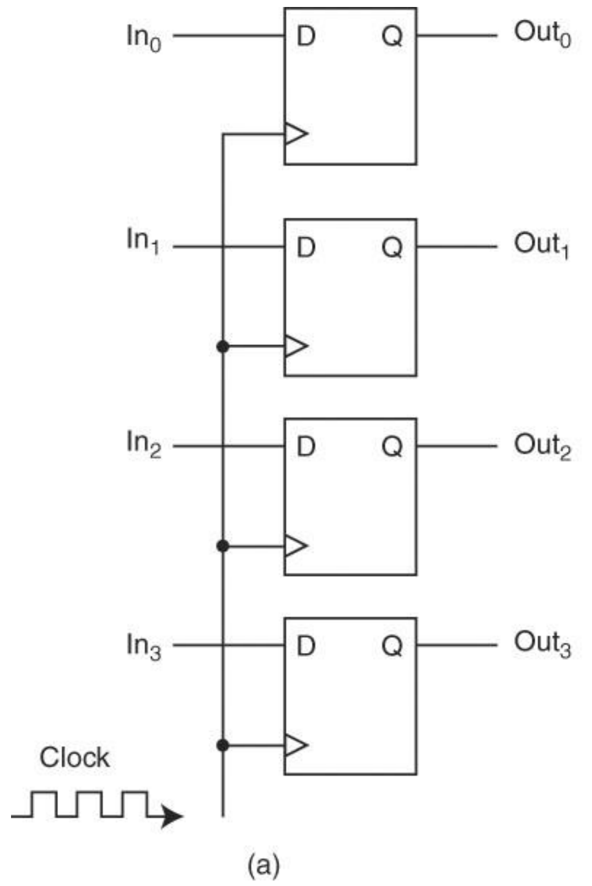


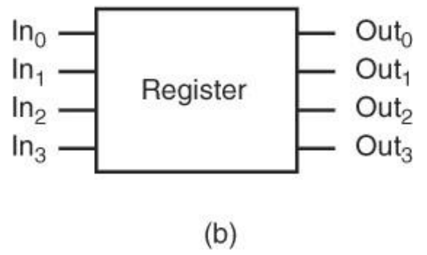
c)

Registers

- Since a single flip-flop stores a single bit, we combine n of them to create an n-bit register
  - However, the S-R flip-flop can be set or reset at any time, we instead want to use the system clock to determine when to change the value
- So, we will use a D flip-flop instead
  - In the D flip-flop, there are 2 input lines, but they represent different things than the S-R flip-flop
    - One input is the clock – the flip-flop can only change when the clock pulses
    - The other input, labeled as D – is the input
      - if 0, then the flip-flop will store 0,
      - if 1 then the flip-flop will store 1

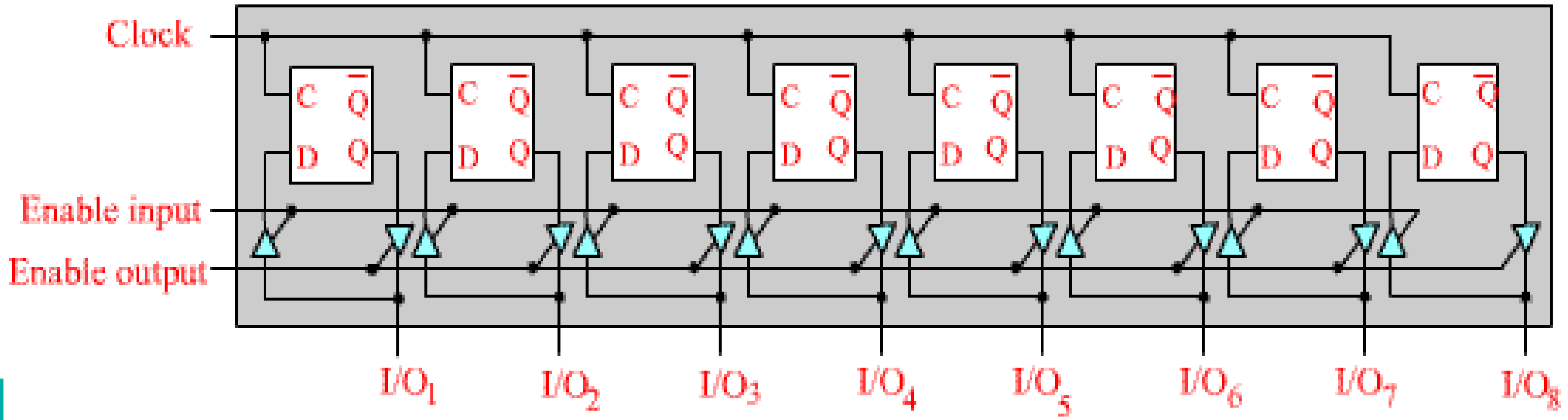
Registers From D Flip-Flops





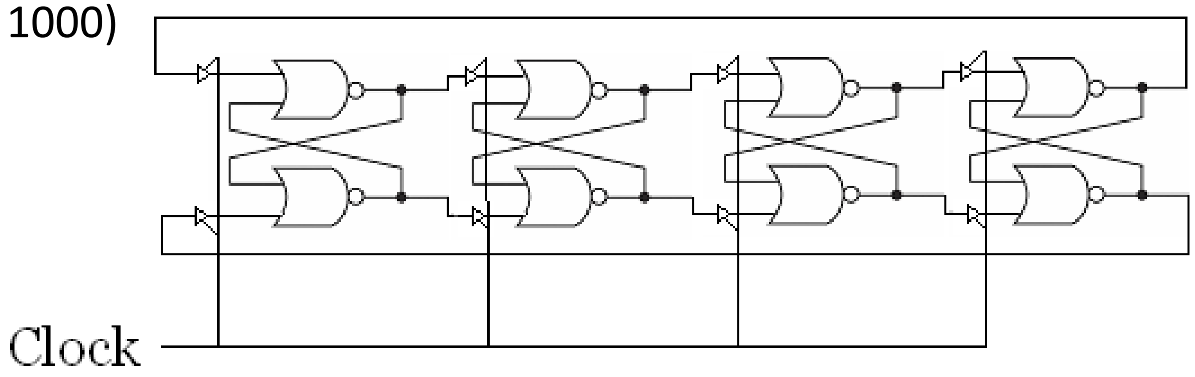
To the right is a 4-bit register
Triggered by the system clock
And connected to an input bus and
An output bus

Below is an 8-bit register with a single I/O bus



Shift and Rotate Registers

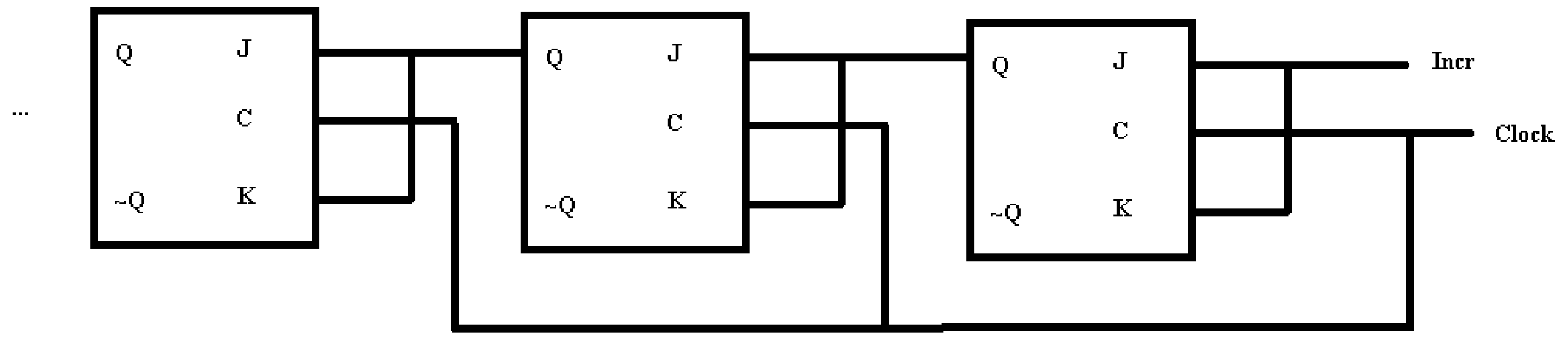
- The shift circuit we saw earlier is difficult to trace through although efficient in terms of hardware
  - we can also build a special kind of register called a shift register or a rotate register by connecting SR flip flops
  - this register will store a bit in each FF as any register, but the Q and ~Q outputs are connected to the SR inputs of a neighboring FF
    - below is a 4-bit right rotate (it rotates the rightmost bit to the leftmost FF, so 1001 becomes 1100 and 0001 becomes 1000)



Upon a clock pulse, each Q output is connected to the FF to the right's S input and each ~Q output is connected to the FF to the right's R input, so an output of Q = 1 causes the next FF to set (become 1) and an output of ~Q = 1 causes the next FF to reset (0)

Increment Register

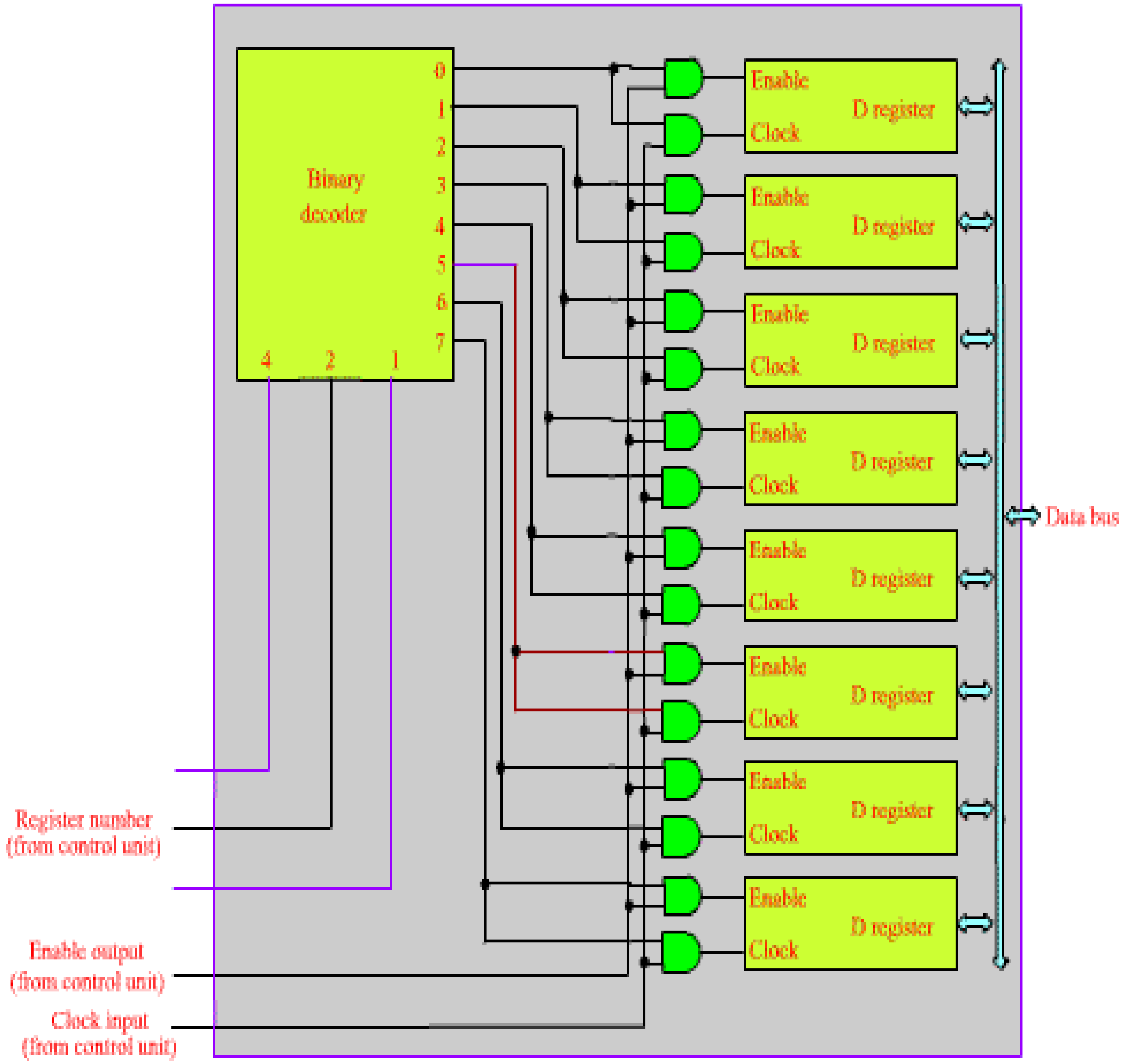
- The J-K flip flop is like the S-R flip flop except
  - J = 1 and K = 1 flips the bit
  - Flip flop only changes state on clock pulse
- Use J-K to implement an increment register – increments the value stored when it receives and Incr signal and a clock pulse



J = 0, K = 0 -- no change
J = 1, K = 1 -- flip bit
Otherwise, act like S-R flip flop (J = 1, set, K = 1, reset)

Upon Incr signal and clock pulse, J = 1 and K = 1, flip this bit
If this bit was 1, then next bit receives 1-1 to flip
If that bit was 1, then next bit receives 1-1 to flip
etc

A Register File

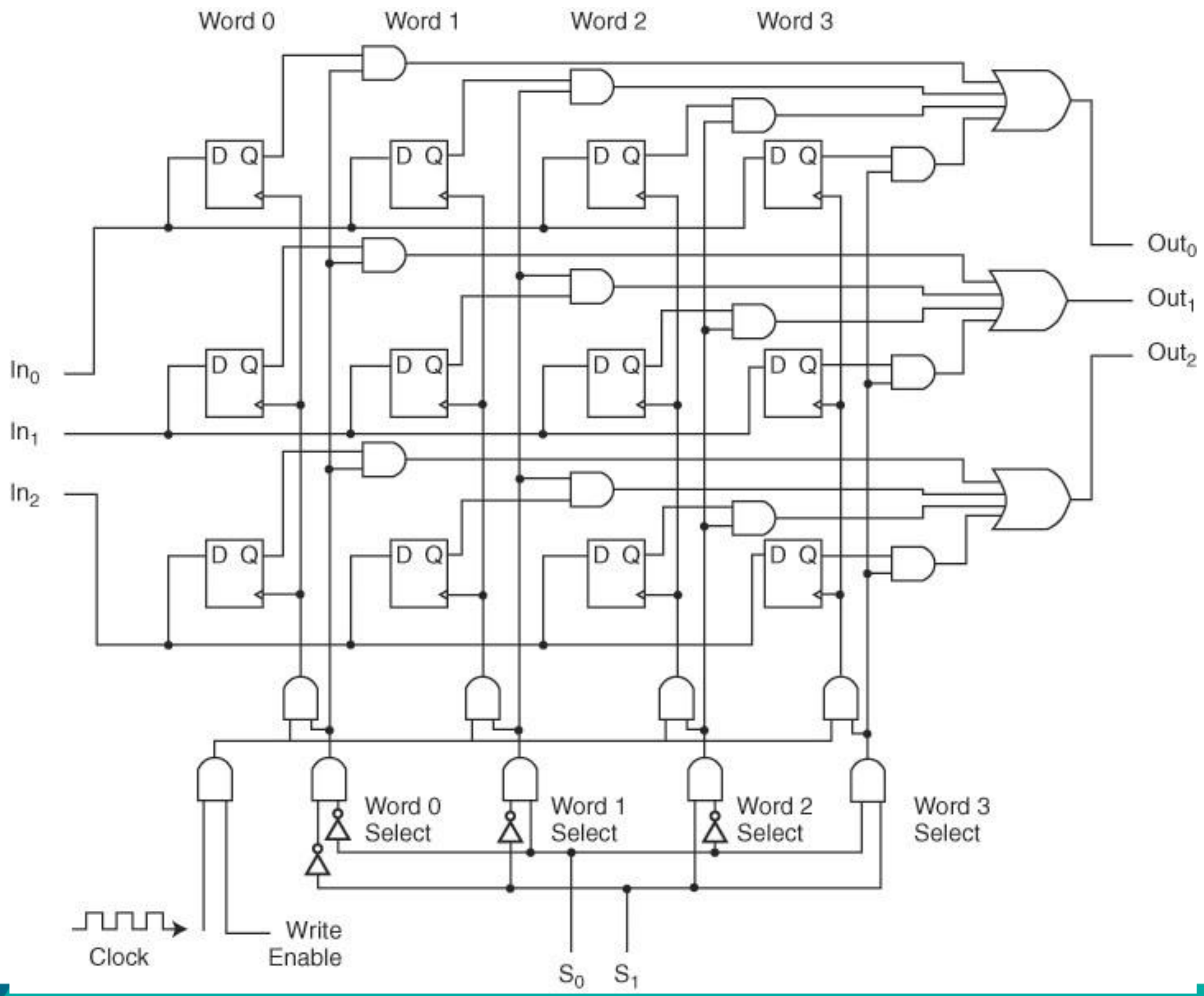

Binary decoder
0
1
2
3
4
5
6
7
4
2
1
Enable
Clock
D register
Data bus
Register number (from control unit)
Enable output (from control unit)
Clock input (from control unit)

The decoder accepts a 3-bit register number from the control unit

This along with the system clock selects the register

The data bus is used for both input and output to the selected register

A 4x3 Memory



This is a collection of flip-flops that can store 4 items (each consisting of 3 bits)

The two bit selector S0 S1 chooses which of the 4 items is desired

It should be noted that computer memory uses a different technology than flip-flops

Chapter 3 Review

A. Logic Gates

B. Boolean Algebra

C. Combinational Circuits

   A. Flip-Flops

D. Sequential Circuits

   A. Memory Components

Chapter 3 ends!