Logic Design (Part 4)
Latches – Storage devices
(Chapter 3 + Notes)

- Log in to a Windows laptop
- Download CedarLogic-Circuits-Latches.cdl file
  from webpage to your desktop and open in CedarLogic

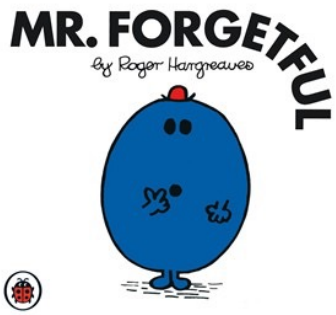
1

Recap: Combinational Logic & Devices

- Basic logic gates (AND, OR, NOT,….)
  - Built using transistors
- Combinational (more complex) logic devices
  - Multiplexers, Decoders, Adders, Sub, Multipliers, Comparators (x=y?)
  - Behavior modeled using a Boolean function
  - Built using logic gates
- Our current ”library”: basic logic gates, combinational logic devices
  - To build a circuit we can use anything from the library…..
  - Is this enough?

2

So Far: Combinational Logic



- Combinational Logic:
  - Always gives the same output for a given set of inputs
  - Aka “state-less” (i.e., no “state” or “memory”)
- Sequential Logic:
  - Its output depends on its inputs & its last output!
  - Forms the basis for “state” or “memory” for a computer

3

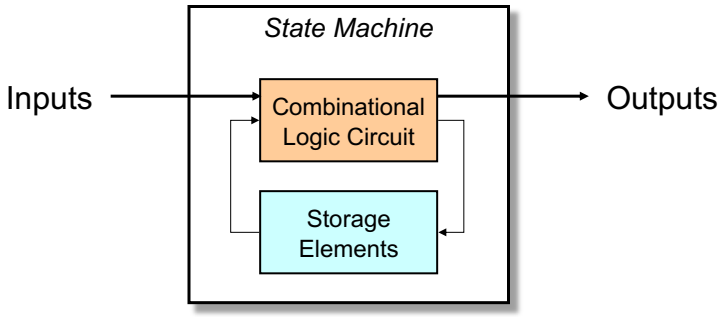
Next . . Circuits with “memory”

- First we need to build a device that can store a bit
  - Using our current ‘library’ of gates
  - Building memory follows
- How to model sequential circuits/machines
  - Methodology for designing these machines: Finite state machine
  - Model as a directed graph
- How do we “synchronize” and ”coordinate” the different pieces in the circuit….enter the CLOCK
- can we use a sequential circuit to “control” how computations take place in a processor ?
- Is a sequential circuit = Computer ?
  - Limitations of sequential machines..more in Foundations course

4

(Finite) State Machine

- type of sequential circuit
  - Combines combinational logic with storage
  - "Remembers" state, and changes output (and state) based on inputs and current state



5

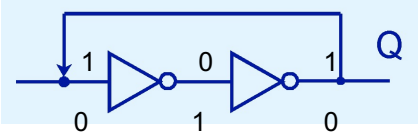
Sequential Logic: Starting point

- Where do we start: Build a device, using combinational logic devices, to store a value
  - RS Latch
    - Use this to build a more useful/easier to use latch – the D Latch
  - concept of memory
- Build it using the devices we have thus far
  - How ? Use "feedback" circuit

6

Feedback Circuits

- What happens if we feed the output of a combinational logic circuit to an input in the circuit ?
  - This is the key to circuits that can store values!
- Stable circuit
  - Output point of circuit retains value indefinitely
  - If Q=1 it will remain 1
  - If Q=0 it will remain 0



- Unstable circuit
  - State that remains constant only for a duration of a few gate delays
- key takeaway: build stable circuits that can “hold” their value

7

Latches and Flip-Flops

- Latch: basic circuit for storage
  - Operate on changes in Level (i.e., 1 or 0)
- Flip-flop:
  - Sequential circuits take input from output of storage
  - Latches that work on change of level can lead to unstable sequential circuits
    - As level changes the outputs change --- inputs change!
  - Flip-Flop circuits designed to operate properly when they are part of a sequential circuit
  - Designed to work in a synchronized circuit with a Clock

8

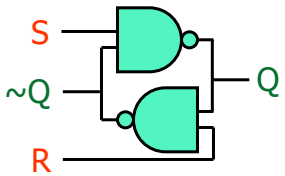
R-S Latch

- The R-S latch is a bi-stable circuit which means that it can happily exist in either of two stable states. Just like a see-saw.
- You can push the latch from one state to another by setting or resetting it with the S-R signals
- The logic levels are maintained because of the feedback paths from outputs to inputs.

9

Most Basic Sequential Logic Circuit: *R-S Latch*

- Most fundamental unit for static memory
  - Has the ability to "store" its last output
- R-S Latch – Cross-Coupled NAND gates
  - Output of each NAND gate serves as input to the other
  - Two inputs: S (*SET*) & R (*RESET*)
  - Two outputs: Q and NOT(Q) Recall: NOT(Q)= ~Q = Q' = $\overline{Q}$
  - Called a "Latch" because it can "Latch" onto data coming in



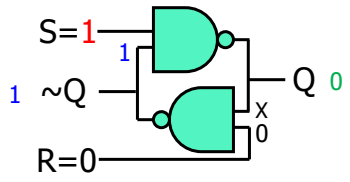
10

R-S Latch – Behavior/Truth Table

- First, recall truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:
  - Best place to start is S=1, R=0



Next, look at top NAND gate
→Its inputs are: 1 and 1
Blue 1, comes from lower NAND
→Produces a 0 at its output
Therefore, when S=1, R=0
The output of latch is: Q=0, ~Q=1

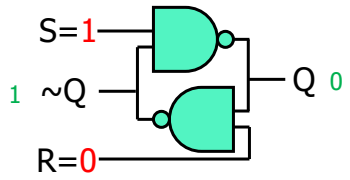
11

Most Basic Sequential Logic Circuit: *R-S Latch*

- First, recall truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:



Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
	0	1		
RESET	1	0	0	1
	1	1		

Called the "RESET" action, as Q is set to 0
Also, notice: Q and ~Q opposite

12

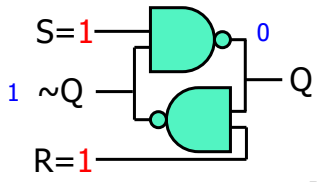
Most Basic Sequential Logic Circuit: *R-S Latch*

- Truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Next, set S=1 R=1
Check value of Q

- R-S Latch Operation:



Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
SET	0	1	?	?
RESET	1	0	0	1
			-	-
HOLD	1	1	0	1

HOLD's last value on its outputs!
OUTPUT depends on input and last output

13

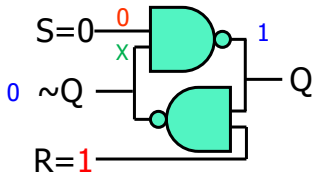
Most Basic Sequential Logic Circuit: *R-S Latch*

- Truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Next, set S=0 R=1
Check value of Q

- R-S Latch Operation:
  - Next input case is called the "SET", when inputs are: S=0, R=1



1st look at upper NAND gate
→Its inputs are: 0 and X (anything)
→Produces a 1 at its output
Lower NAND gate
→Inputs are: 1 and 1
→Produces a 0 at its output

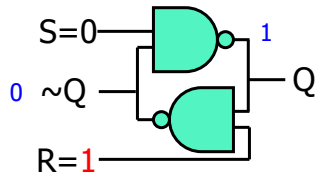
14

Most Basic Sequential Logic Circuit: *R-S Latch*

- Truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:



Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
SET	0	1	1	0
RESET	1	0	0	1
	1	1		

*SET*s LATCH to have a "1" at the output

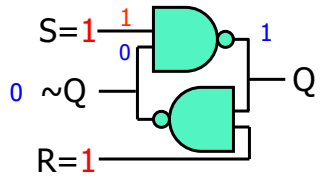
15

Most Basic Sequential Logic Circuit: *R-S Latch*

- Truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

- R-S Latch Operation:
  - Last valid input case is the "HOLD" S=1, R=1
    - If we have just "SET" Latch, we will have Q=1, ~Q=0, already on outputs



Upper NAND gate
→Has S=1 & former value of ~Q=0
→Produces a 1 at its output
(same ~Q as when it started)

Lower NAND gate
→Inputs are: 1 and 1
→Produces a 0 at its output *(same Q)*

16

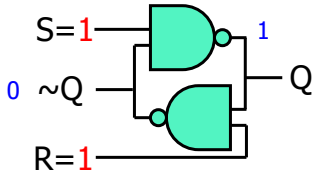
Most Basic Sequential Logic Circuit: *R-S Latch*

- Truth table for a NAND gate:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Next, set S=1 R=1
Check value of Q

- R-S Latch Operation:



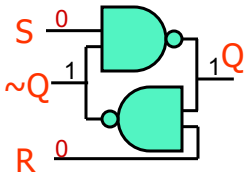
Truth Table for R-S Latch:

ACTION	S	R	Q	~Q
	0	0		
SET	0	1	1	0
RESET	1	0	0	1
HOLD	1	1	1	0

HOLD's value we "SET" last

17

Storage - Cross-Coupled NANDs (R-S Latch)



- Set value of Q by R=0 or S=0
- Store value of Q with R=1, S=1
- What happens with S=0 and R=0?
  - Short answer: confusion
  - Real circuits depend on both Q and ~Q
  - Strange things may happen if both are 1

ACTION	S	R	Q	~Q
ILLEGAL	0	0	1	1
SET	0	1	1	0
RESET	1	0	0	1
HOLD	1	1	1	0
HOLD	1	1	0	1

D-Latches shows a way to prevent
the RS Latch from ever getting S=R=0 as its input

18

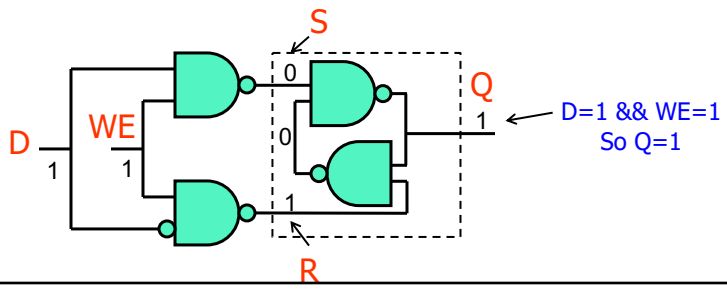
Circuits with Memory

- We now have a device (RS Latch) that is capable of storing a bit
  - It hold the previous value of Q
- but need to make sure the inputs are never R=S=0
- Modify the circuit to get a D latch

19

Gated D-Latch: Preventing "Illegal State" of RS Latch

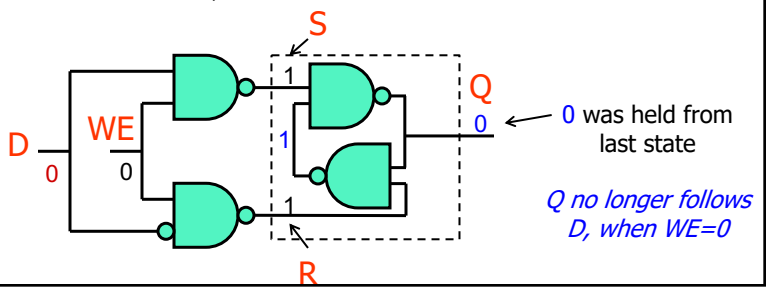
- Add logic to an R-S latch
  - Create a more convenient interface (1) prevent S=0 && R=0 and (2) control when you can "write" into storage
- Two inputs: D (data) and WE (write enable)
  - When WE = 1, latch is set to value of D
    - S = NOT(D), R = D



20

Gated D-Latch: Preventing "Illegal State" of RS Latch

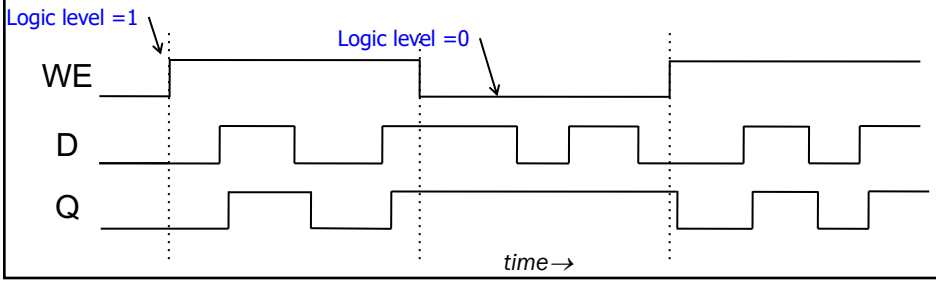
- Add logic to an R-S latch
  - Create a more convenient interface, prevent S=0 && R=0
- Two inputs: D (data) and WE (write enable)
  - When WE = 1, latch is set to value of D
    - S = NOT(D), R = D
  - When WE = 0, latch continues to hold previous value
    - S = R = 1 (hold condition for SR latch)
  - Extra logic does not allow S=0, R=0 case to occur



21

D-Latch Timing Diagram

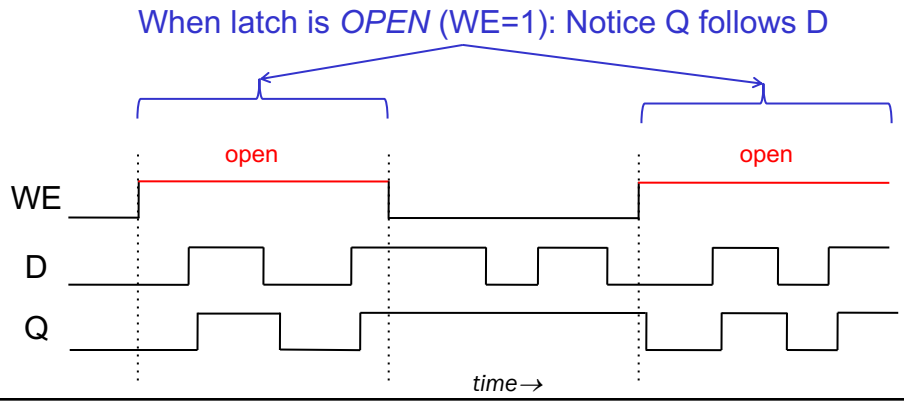
- The diagram below is called a "Timing" Diagram
  - Our D-Latch is previous-state dependent
    - We can think of this as a time dependency
    - Moving to the right on diagram, represents forward moving time
  - The inputs & outputs to our D-Latch are on left
    - Inputs/Outputs can be either "HIGH" (logic 1) or "LOW" (logic)
  - Think of this as a time-dependent truth table



22

D-Latch Timing Diagram

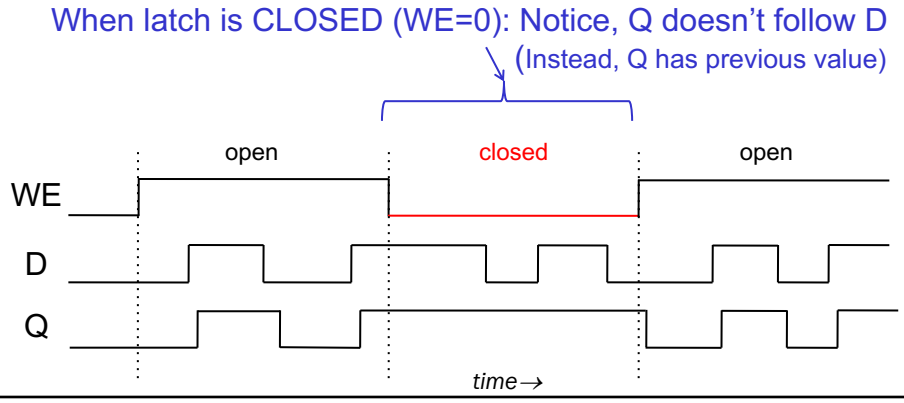
- When the WE signal is high the latch is said to be open and the output signal, Q, follows the input signal, D.
  - As in any combinational circuit there will be a small delay between the time that the input changes and the time that the output follows suit.



23

D-Latch Timing Diagram

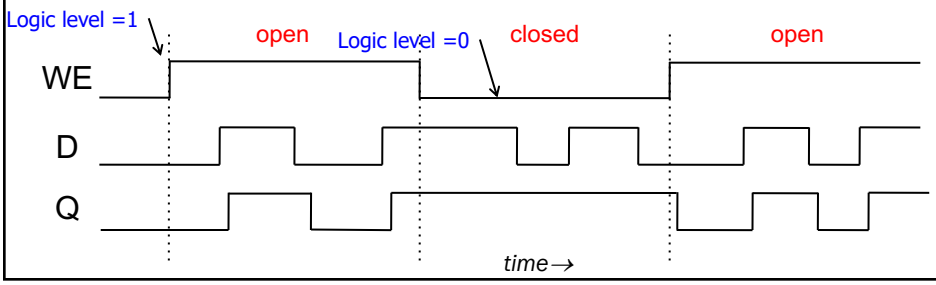
- When the WE signal is low the latch is closed and the output signal, Q retains its value.



24

D-Latch Timing Diagram

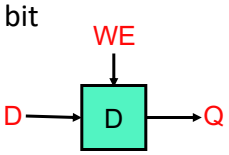
- Setup / Hold Times
  - The input signal should (D) be stable a certain amount of time before the WE signal is set to CLOSED (WE=0)
    - This is referred to as the SETUP time
  - In addition, the input signal (D) must be stable for a time after the WE is set to CLOSED (WE=0)
    - This is referred to as HOLD time
  - Why? Time must be given for inputs to propagate through NAND gates! Gates are not instantaneous!



25

Next… Storage Devices

- we now have a device (D-Latch) that can store a bit
  - Abstract the device: input D, WE; output/storage Q



- Use this to build 'real' storage devices….
- Temporary storage in a computer…*Register*
  - Where are variables stored before being sent to the arithmetic unit for operations on them?
  - Can we build an n-bit register using latches?
- What about "main" *memory*
  - Memory hierarchy ?

26

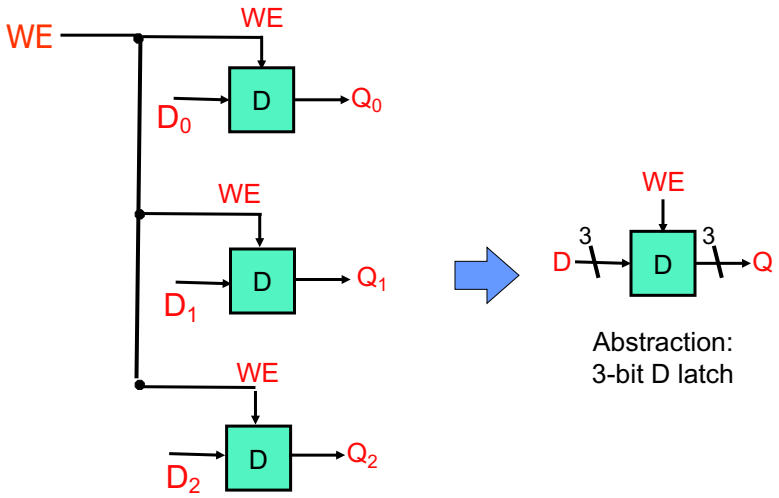
Question: Page 4 of Latches.cdl file

- Describe behavior

27

Multi–Bit D-Latch: Register ?

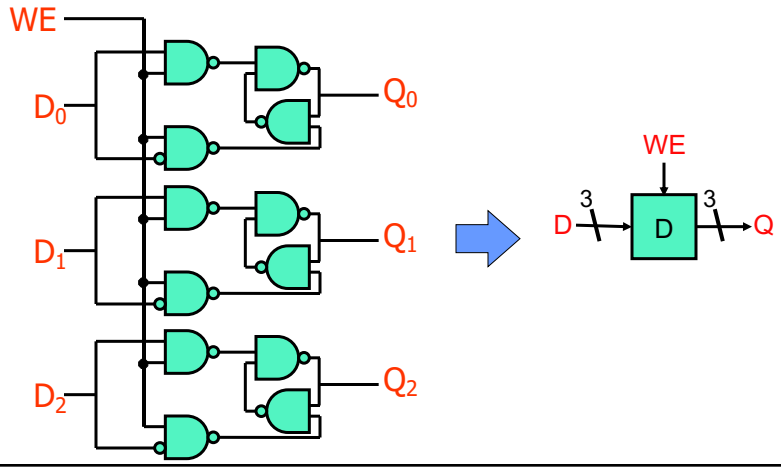
- A collection of D-latches, controlled by a common WE
- When WE=1, 3-bit value D is written to the outputs



28

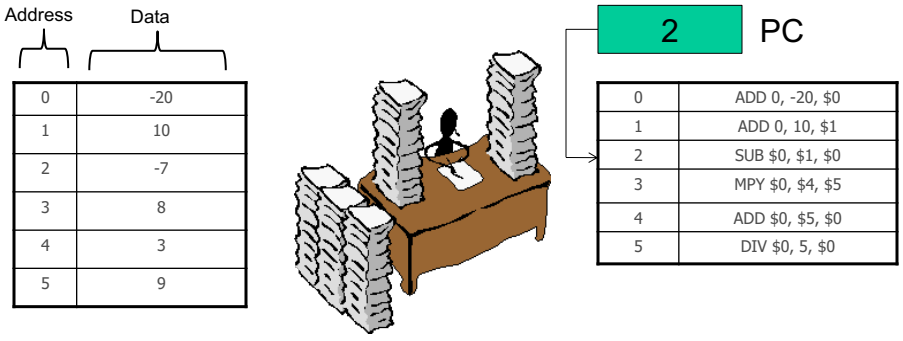
Multi–Bit D-Latch – Register: Inside the latches

- A collection of D-latches, controlled by a common WE
- When WE=1, n-bit value D is written to the outputs



29

A Basic Model of a Computer



Memory CPU Instructions

Essential Part of Computer!

Basic Components: Address: Looks up data
Note: both are in binary

30

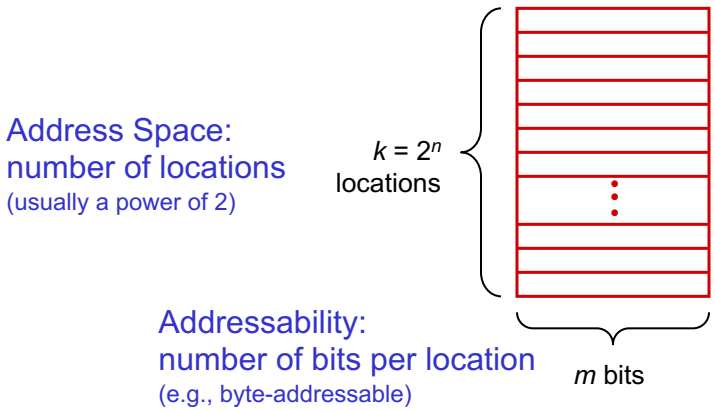
Memory

- We know how to store m-bit number in a register
- How about many m-bit numbers ?
  - Address space
- How to fetch a specific m-bit number?
  - addressing

31

Memory

- Now that we know how to store bits, we can build a memory – a logical *k by m* array of stored bits



32

Memory: Address space and total size

A large number of addressable fixed size locations

- Address Space

  n bits allow the addressing of 2^n memory locations.

  - Example: 24 bits can address 2^{24} = 16,777,216 locations (i.e. 16M locations).
  - If each location holds 1 byte (= 8 bits) then the memory is 16MB.
  - If each location holds one word (32 bits = 4 bytes) then it is 64 MB.

- Total size of memory is number of locations multiplied by number of bits *m* at each location = Address Space * Addressability = $2^n * m$

33

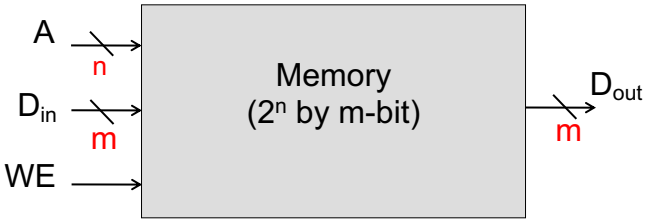
Memory - Addressability

- Computers are either byte or word addressable
  - - i.e. each memory location holds either 8 bits (1 byte), or a full standard word for that computer (16 bits for the LC-3, more typically 32 bits, though now many machines use 64 bit words).
- Normally, a whole word is written and read at a time:
  - If the computer is word addressable, this is simply a single address location.
  - If the computer is byte addressable, and uses a multi-byte word, then the word address is conventionally either that of its most significant byte (big endian machines) or of its least significant byte (little endian machines).

34

Memory Interface

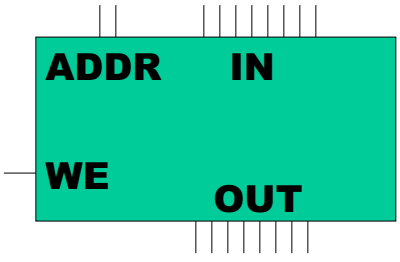
- There are two basic operations on a memory
  - Selecting one of the memory locations to read from
  - Selecting one of the memory locations to write to
- Interface signals
  - A: n-bit address lines to select/specify a location
  - D_{out} : Contents of selected location during read (m bits)
  - D_{in} : Value to be stored during write (m bits)
  - WE : If WE = 1 then write operation, WE = 0, read operation



35

Memory

- Looking from the outside, what do we need?
- READ operation: Given address A of N bits, fetch contents at that address
  - From 2^N locations we *select* one of them to be sent to the output
- WRITE: Given address A of N bits, write into exactly one of the 2^N locations.



36

Question: Page 5 of Latches.cdl file

- Describe behavior

37

Question: Devices to construct Memory ?

- 1. Do you have a device to store a m-bit number ?
- 2. For READ: do you have a device that can send one out of 2^N inputs (inputs are in the 2^N latches) ?
- 3. For WRITE: do you have a device that can enable exactly one Write Enable (WE) in the 2^N D-latches ?

38

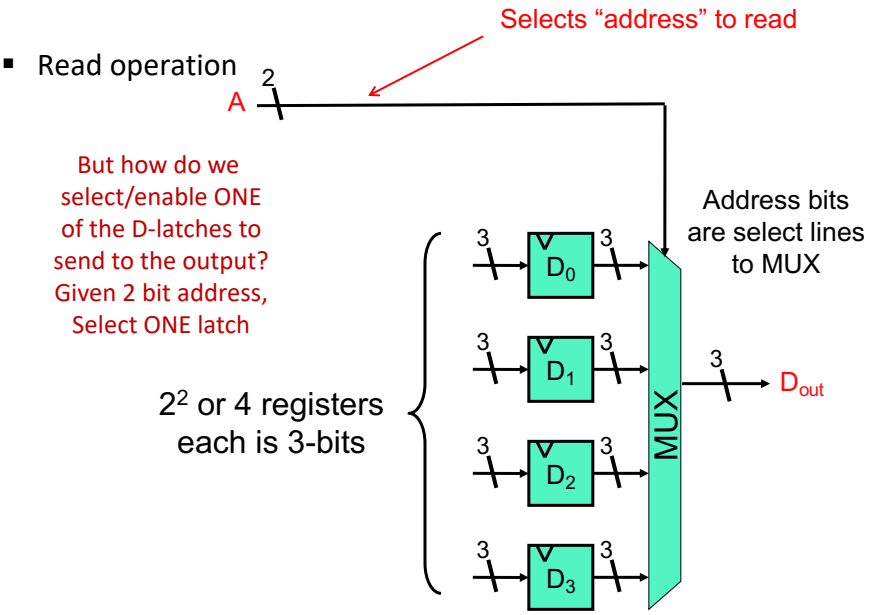
Question: Devices to construct Memory ?

- Memory is a 2^N by m array:
  - 2^N rows – each row/location address specified using N bits
  - Each row has/stores m-bits
- 1. Do you have a device to store a m-bit number ?
  - How many of them do you need for the entire 2^N by m memory ?
- 2. For READ: do you have a device that can send one out of 2^N inputs (inputs are in the 2^N latches/registers) ?
  - We have 2^N 'registers' each of m bits
  - Need to be able to send (read) contents from exactly one of these
- 3. For WRITE: do you have a device that can enable exactly one Write Enable (WE) in the 2^N D-latches ?

39

Example: 2^2 by 3-bit memory

- Read operation

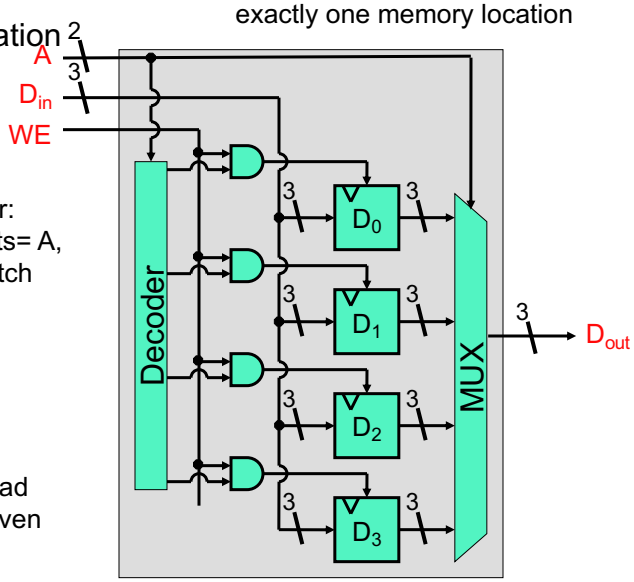


40

Example: 2^2 by 3-bit memory

How do we enable write into exactly one memory location

- Write operation



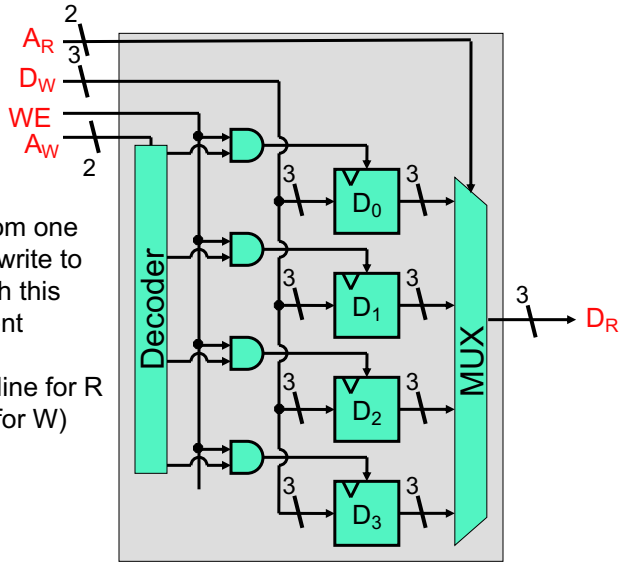
Use 2-4 Decoder: Input address bits= A, exactly one D-latch has WE=1

Limitation: You can only read or write at any given time

41

2^2 by 3-bit memory - Multiple "Ports"

- Independent Read/Write

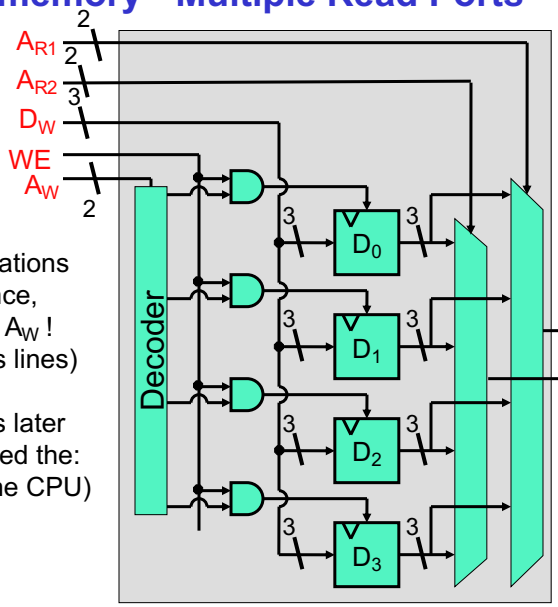


You can read from one address A_R and write to another A_W with this arrangement

(notice 1 address line for R 1 address line for W)

42

2^2 by 3-bit memory - Multiple Read Ports



Read from 2 locations A_{R1}, A_{R2} at once, Write to a third A_W ! (notice 3 address lines)

(We will use this later In something called the: "register file" for the CPU)

43

More Memory Details

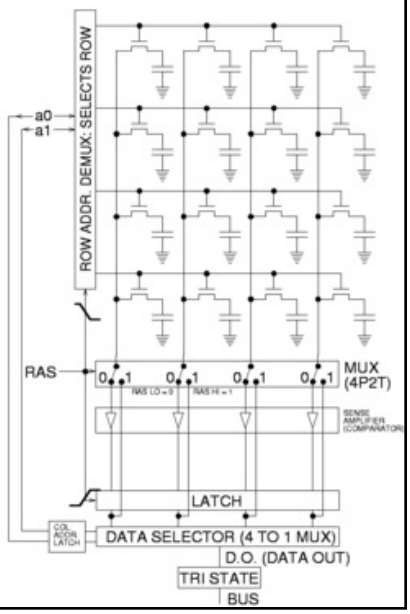
- This is still not the way actual memory is implemented
  - Real memory: fewer transistors, denser, relies on analog properties
- But the logical structure is similar
  - Address decoder
  - Word select line, word write enable
  - Bit line
- Two basic kinds of RAM (Random Access Memory)
- Static RAM (SRAM) - 6 transistors per bit
  - Fast, maintains data as long as power applied
- Dynamic RAM (DRAM) - 1 transistor per bit
  - Denser but slower, relies on "capacitance" to store data, needs constant "refreshing" of data to hold charge on capacitor

Also, non-volatile memories: ROM, PROM, flash, …

44

Dynamic RAM

- Information stored as charge on capacitors.
- Capacitors leak so values have to be 'refreshed' continually
- As memory chips get larger, access times tend to increase. The processor spends more time waiting for data.
  - ➢ This is a major issue limiting computer systems performance



45

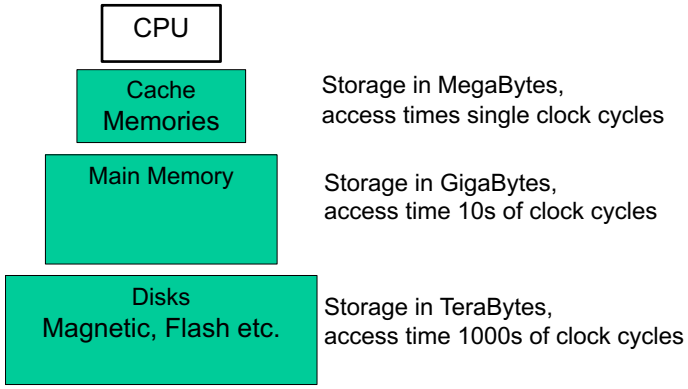
Speed mismatch: Example

- Intel Core i5 – Processor
  - Clock rates approx 2.5GHz, Clock period approx 0.4 ns
- DDR2-667 PC2-5300 SO-DIMM – 2 GB Memory
  - Can deliver at most 1 64-bit word every 1.5 ns
- Mismatch between processor speed and memory speed

46

Memory Hierarchy

- Modern computers try to mitigate memory delays by exploiting locality of reference through caches.
- Smaller, faster memory stores are placed closer to the CPU and bulk transfers from slower memory are used



47

Next: Design process for Sequential Circuits – Finite State Machines

- Recap: we now have a set of devices capable of storing bits
  - D-latch, Register, Memory (?),…
- Definition of sequential circuits
  - Components of a sequential circuit
- synchronization using a CLOCK
  - Modifying latches to work with a clock…Flip Flops
  - Flip Flops are the basic unit of storage in sequential circuits
- Designing sequential circuits – methodology ?
  - Finite state machine diagrams
  - Mapping to truth table…..build circuit

48

Next: Design process for Sequential Circuits – Finite State Machines

- Definition of sequential circuits
  - Components of a sequential circuit
- synchronization using a CLOCK
  - Modifying latches to work with a clock…Flip Flops
  - Flip Flops are the basic unit of storage in sequential circuits
- Designing sequential circuits – methodology ?
  - Finite state machine diagrams
  - Mapping to truth table…..build circuit

49