SYST26671
Computer Architecture

Sequential Logic Components

Contents

Sequential Logic Circuits

- Generates next output(s) from current input(s) and current output(s)
  - Attention: Current output is based on previous input
- Stores information
- Next output(s) typically triggered by an external event (e.g. a clock)

Sequential Logic Circuits (cont.)

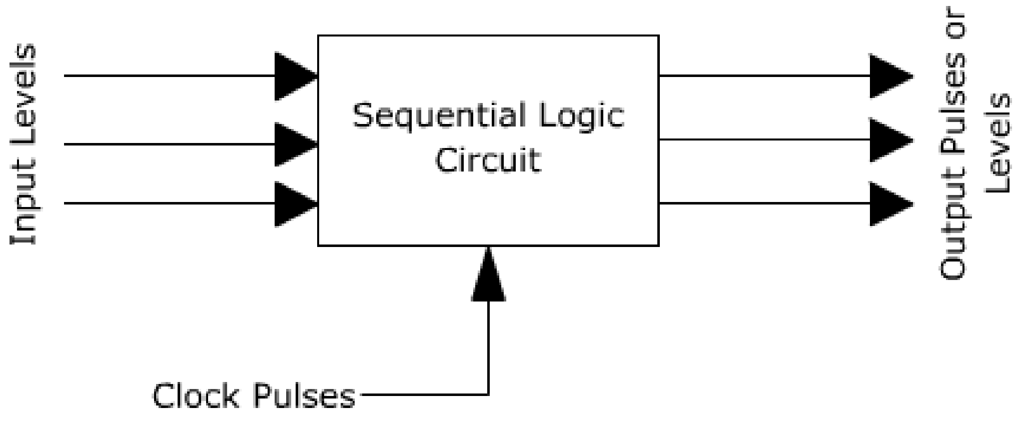
- Digital Logic Circuits with feedback
  - outputs = f(inputs, past inputs, past outputs)
  - Provides the base for building "memory" into logic circuits

Types of Sequential Circuits

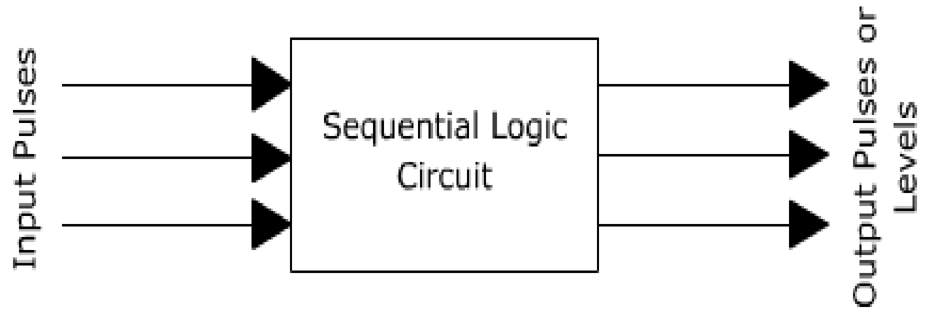
- Two main types
  1. Synchronous - state change triggered (activated) by a clock input
  2. Asynchronous - **<u>not triggered by a clock</u>** but by the pulse of the inputs

You do not need to know anymore about asynchronous circuits for the rest of this course

Synchronous



Asynchronous sequential circuits

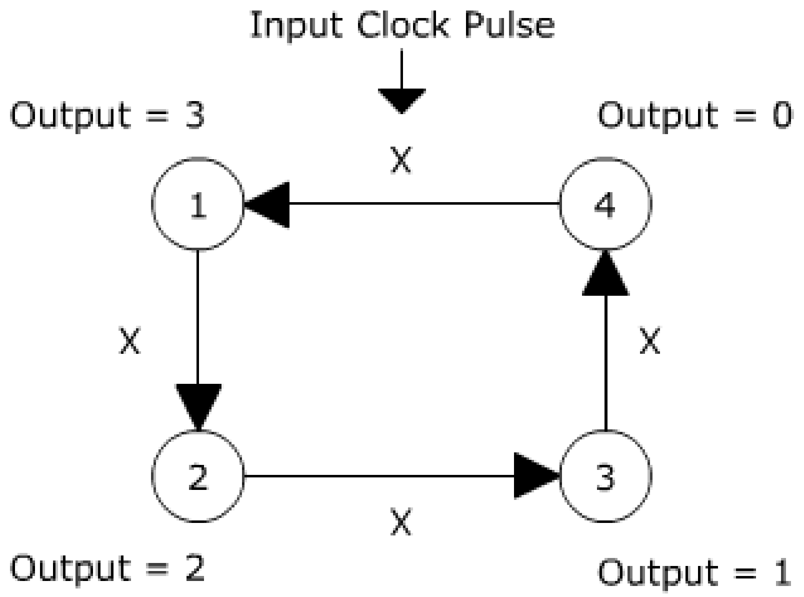


Understanding sequential circuits

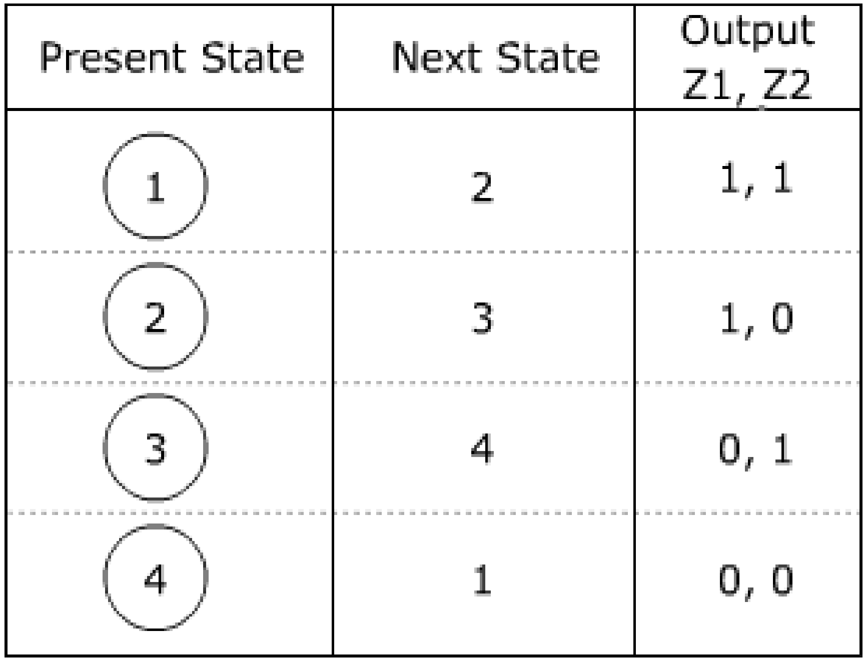
- Designing and understanding of sequential logic circuits requires a way of representing their internal states.
- State diagrams and state tables are a way of understanding this.
- See next slide for very simple example using a 2-bit Down Counter

A Simple State example

- Example: Design a logic system that counts down from logic 3 to logic 0 with every clock pulse then rolls over to 3 after 0.



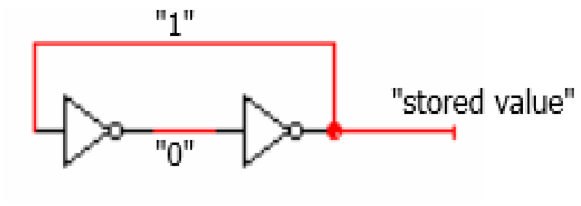
State Table



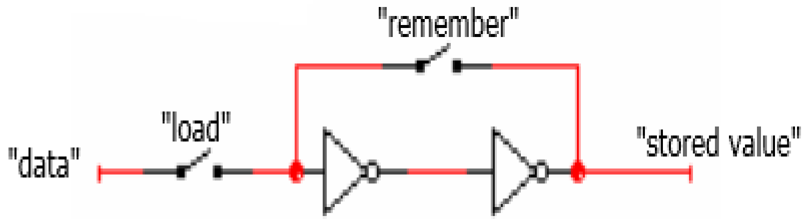
Present State	Next State	Output Z1, Z2
1	2	1, 1
2	3	1, 0
3	4	0, 1
4	1	0, 0

Simplest Circuits with feedback

- Two inverters form a static memory cell
  - will hold value as long as it has power applied

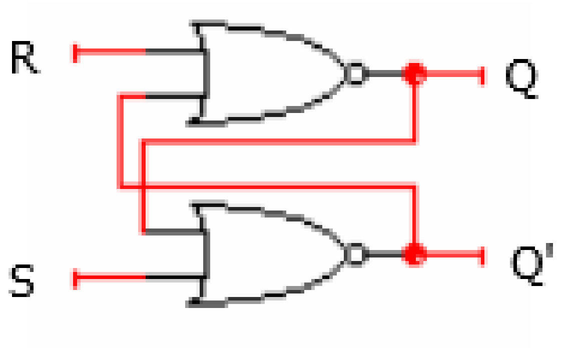


  - How to get a new value into the memory cell?
    - selectively break feedback path
    - load new value into cell



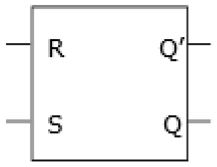
S-R Latch

- One of the simplest sequential circuits.
- Composed of two cross-coupled NOR gates
- R = Reset   S = Set
- Also called RS-Latch



Truth Table

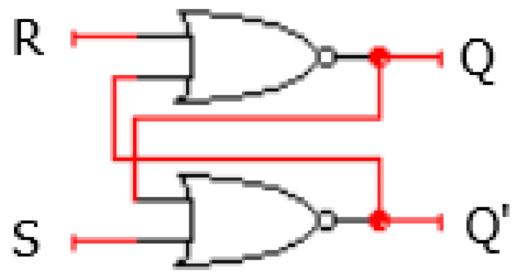
S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	unstable



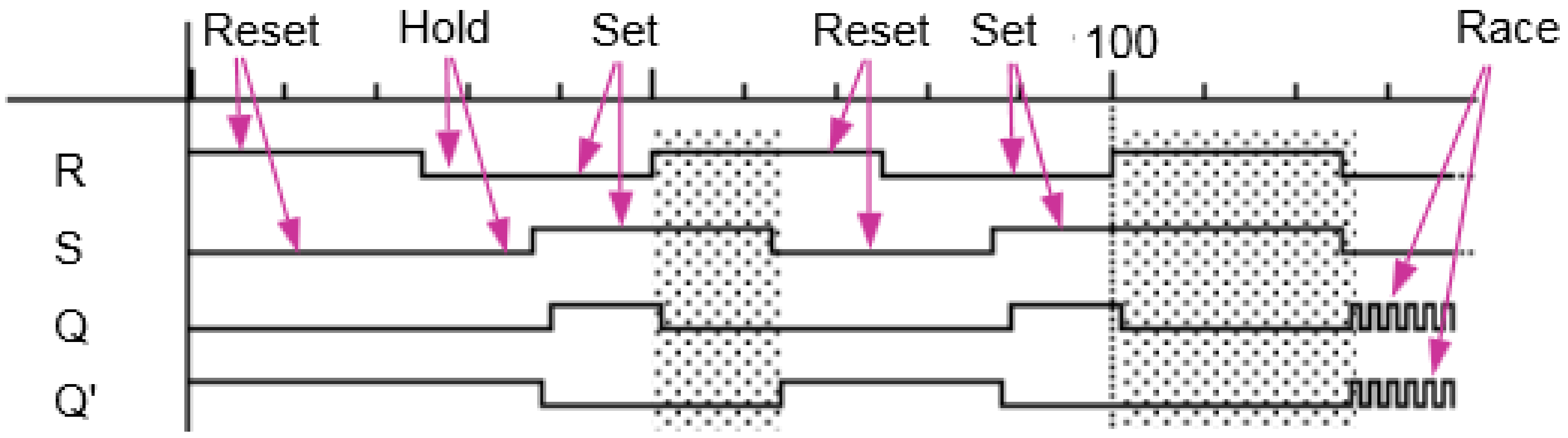
Symbol

S-R Latch

- Truth table and timing

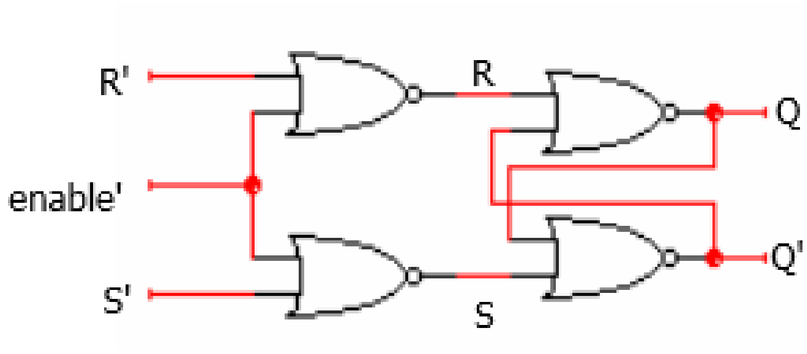


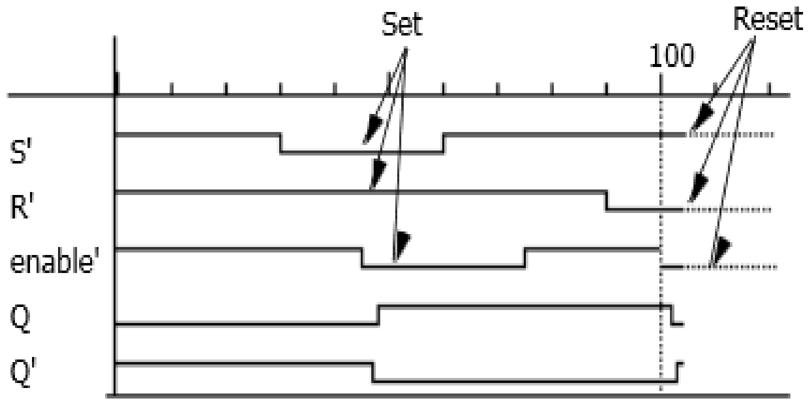
S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	disallow



Gated S-R latch

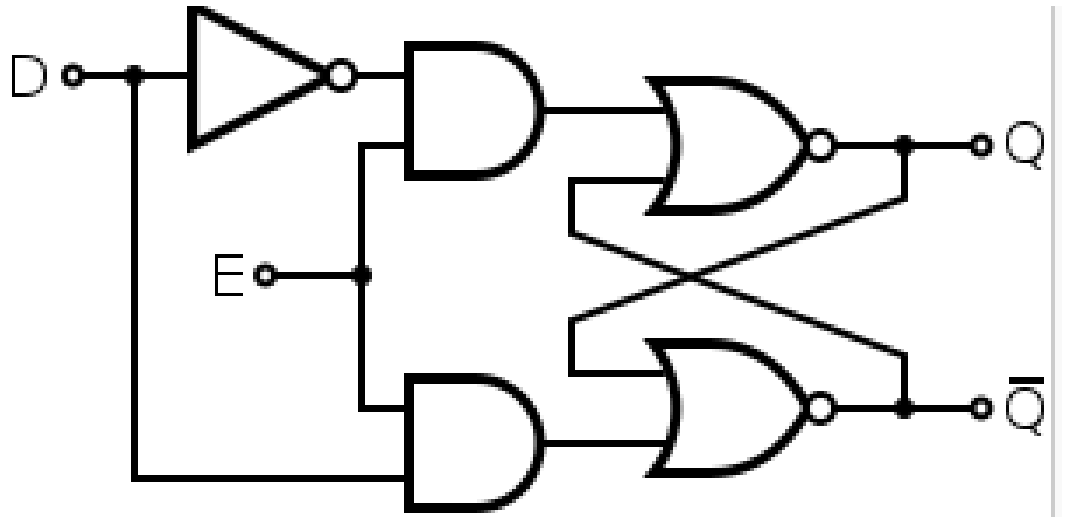
- Gates used to control when R and S inputs matters
  - – otherwise, the slightest glitch on R or S while in hold state could cause unwanted change

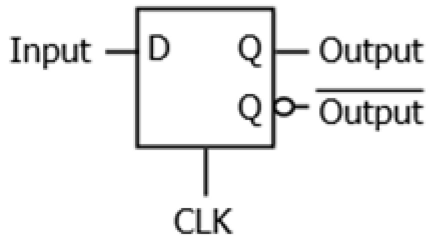


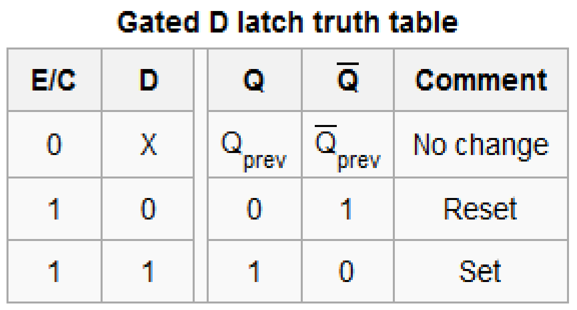


The D Latch

- Makes S and R complements of each other
- Ensures that S and R never change simultaneously







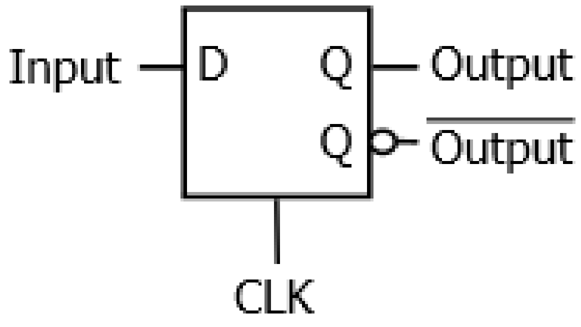
Gated D latch truth table

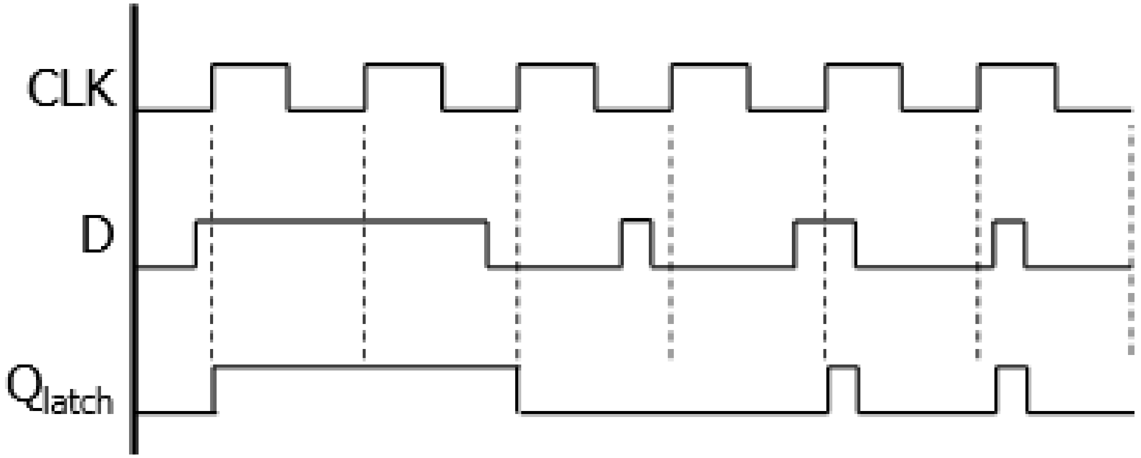
E/C	D	Q	$\overline{Q}$	Comment
0	X	Q_{prev}	$\overline{Q}_{prev}$	No change
1	0	0	1	Reset
1	1	1	0	Set

http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29

The D Latch cont.

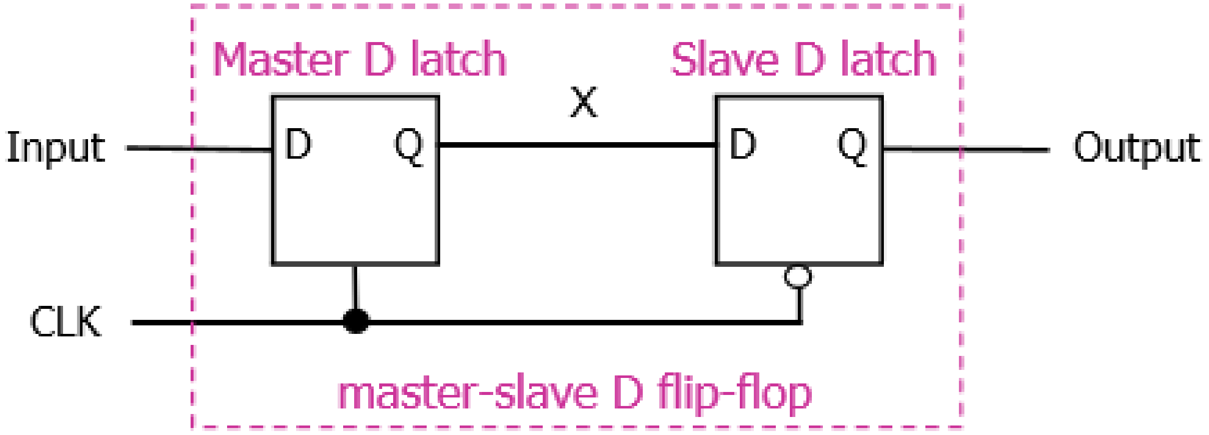
- Output depends on clock
  - Clock high: Input passes to output
  - Clock low: Latch holds its output
- Latch are level sensitive and transparent





D Flip-Flop (D FF)

- Created from two cascading D Latch



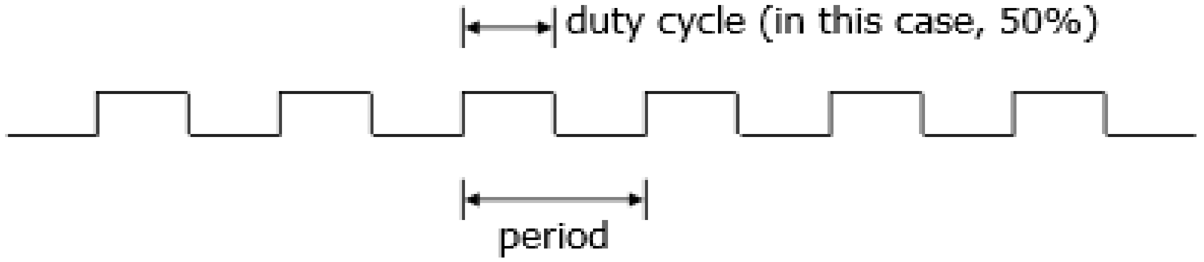
Clocks

Used to keep time

- wait long enough for inputs (R' and S') to settle
- then allow to have effect on value stored

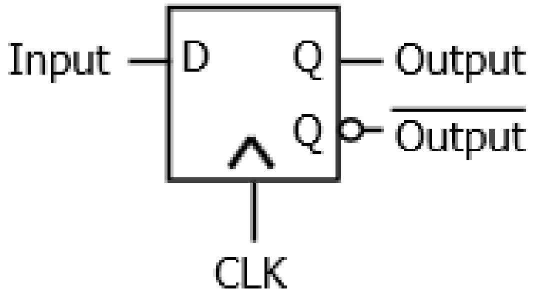
Clocks are regular periodic signals

- period (time between ticks)
- duty-cycle (time clock is high between ticks - expressed as % of period)

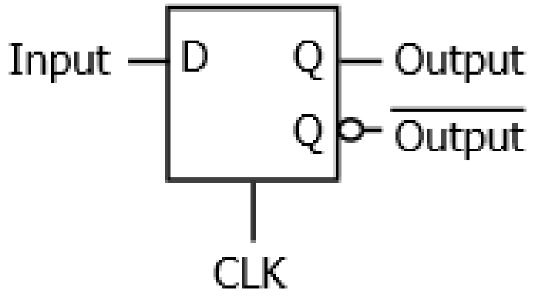


Terminology & Notations

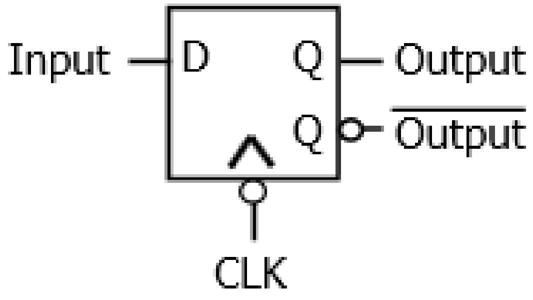
Rising-edge triggered D flip-flop



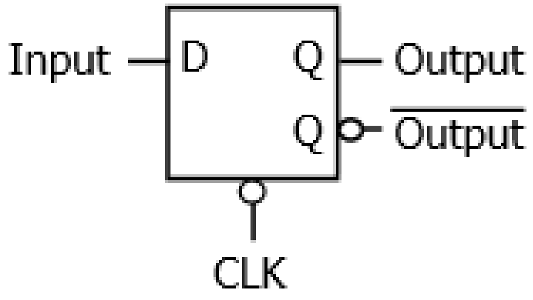
Positive D latch



Falling-edge triggered D flip-flop

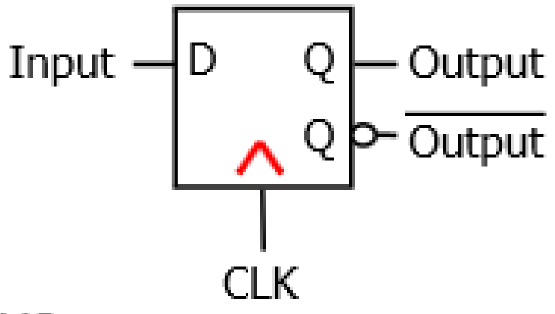


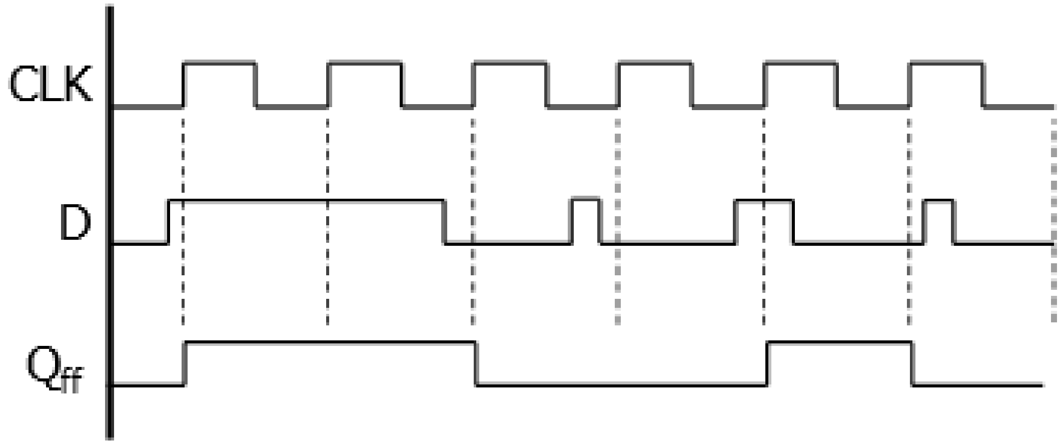
Negative D latch



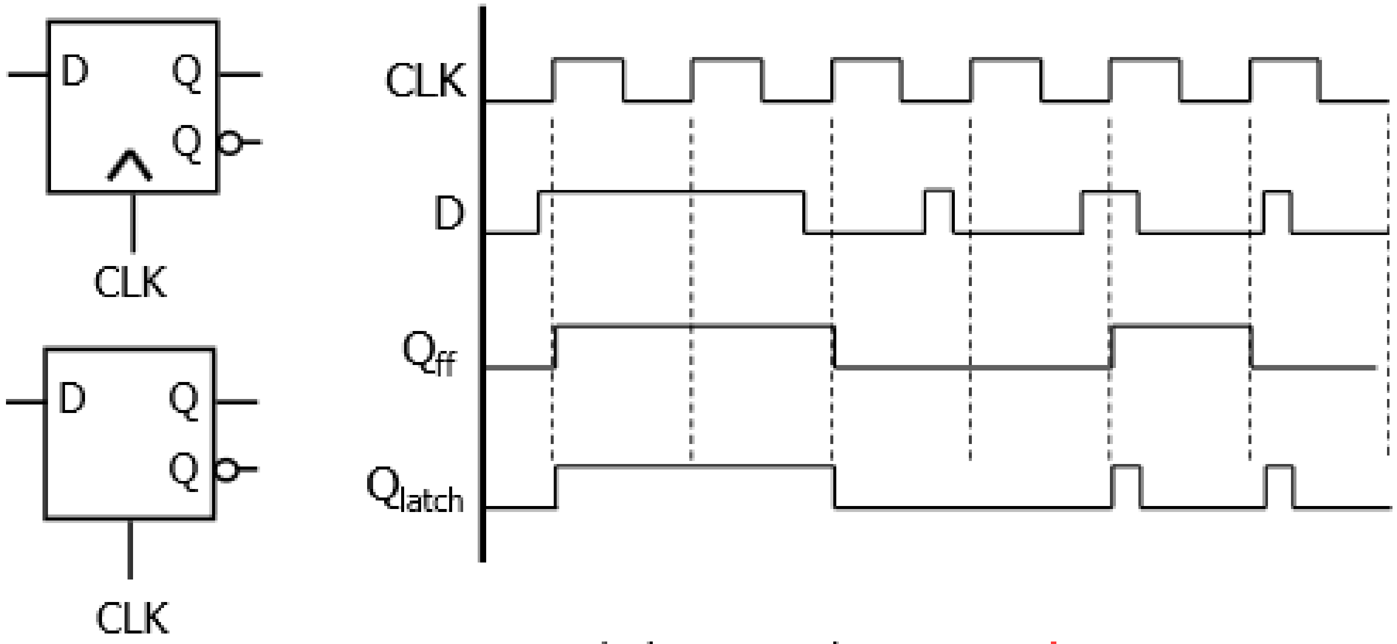
D FF

- Input sampled at clock edge
  - Rising edge: Input passes to output
  - Otherwise: Flip-flop holds its output
- Flip-flops are rising-edge triggered, falling-edge triggered, or master-slave





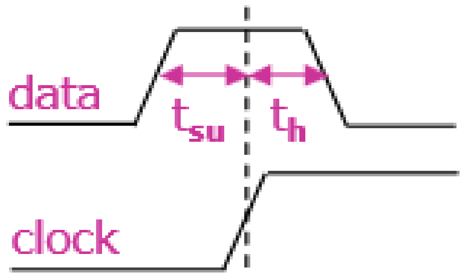
Latches vs Flip-flops

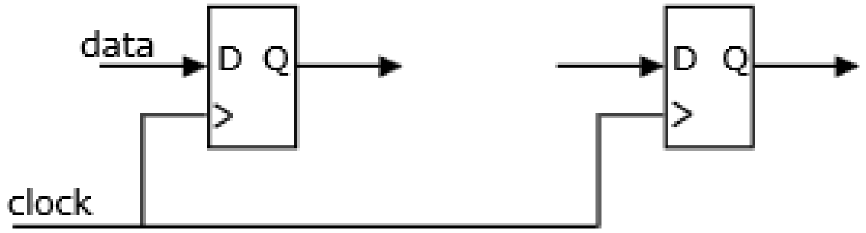


behavior is the same unless input changes while the clock is high

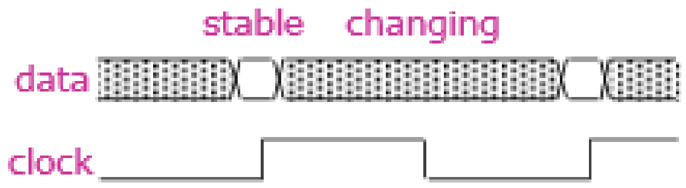
FF Timing

- Setup time t_{su}: Amount of time the input must be stable before the clock transitions high (or low for negative-edge triggered FF)
- Hold time t_h: Amount of time the input must be stable after the clock transitions high (or low for negative-edge triggered FF)



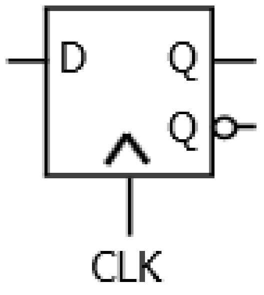


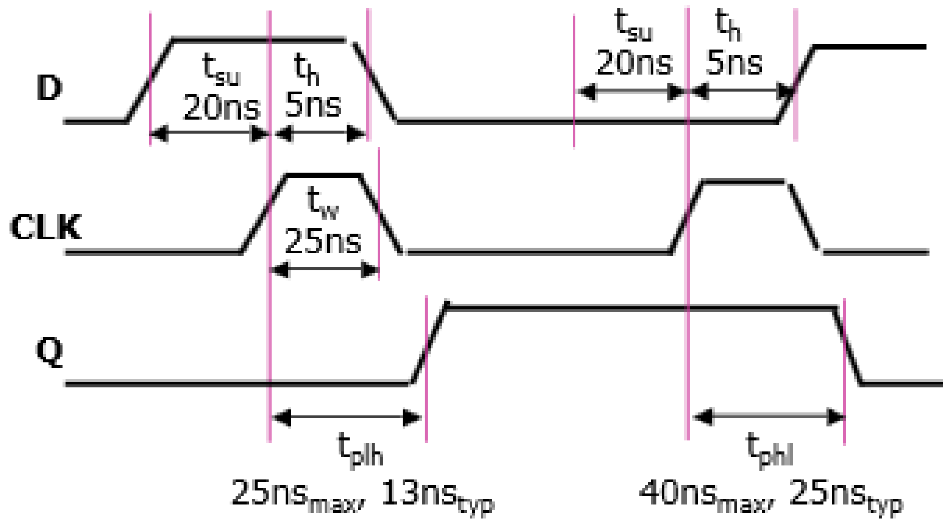
There is a timing "window" around the clock edge during which the input **must** remain stable



FF Timing cont.

- Timing constraints
  - Must meet setup and hold times
  - Must meet minimum clock width
  - Will have propagation delays (low to high & high to low)



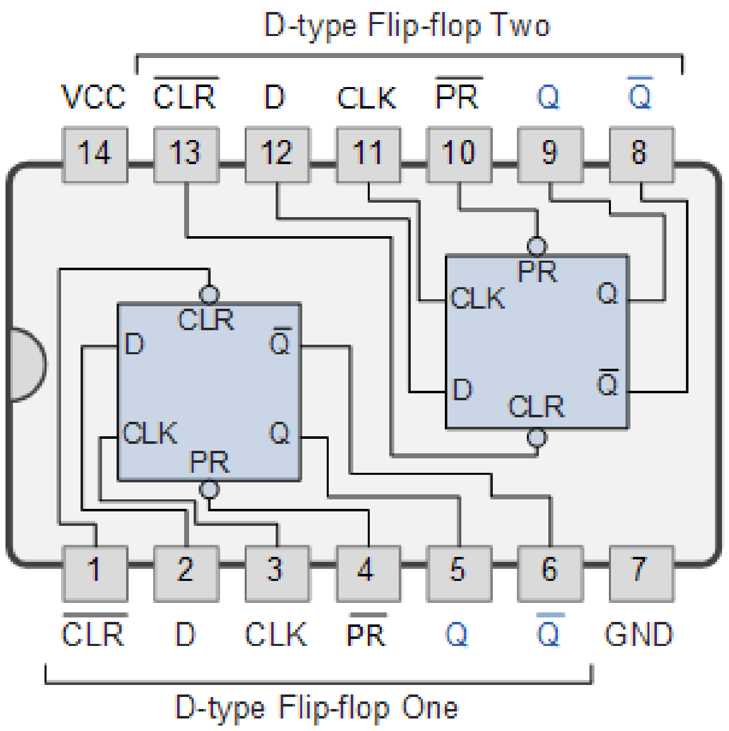


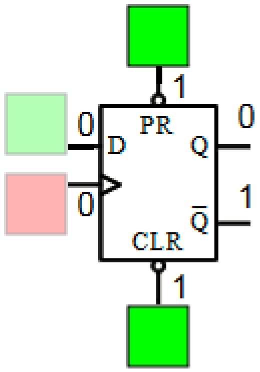
Clear and Preset in FFs

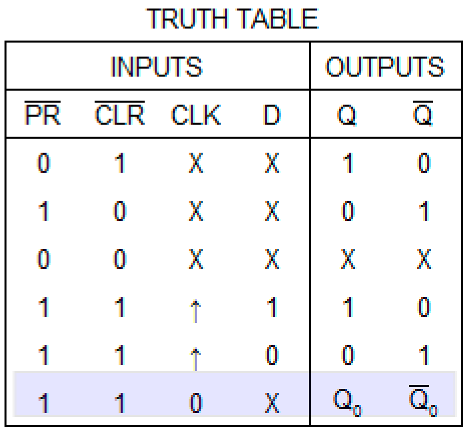
- Clear and Preset set flip-flop to a known state
  - Used at startup, reset
- Clear or Reset to a logic 0
  - Synchronous: Q=0 when next clock edge arrives
  - Asynchronous: Q=0 when reset is asserted
    - Doesn't wait for clock
    - Quick but dangerous
- Preset or Set the state to logic 1
  - Synchronous: Q=1 when next clock edge arrives
  - Asynchronous: Q=1 when reset is asserted
    - Doesn't wait for clock
    - Quick but dangerous

Example of chip containing D FF

74LS74 Dual D-type Flip Flop

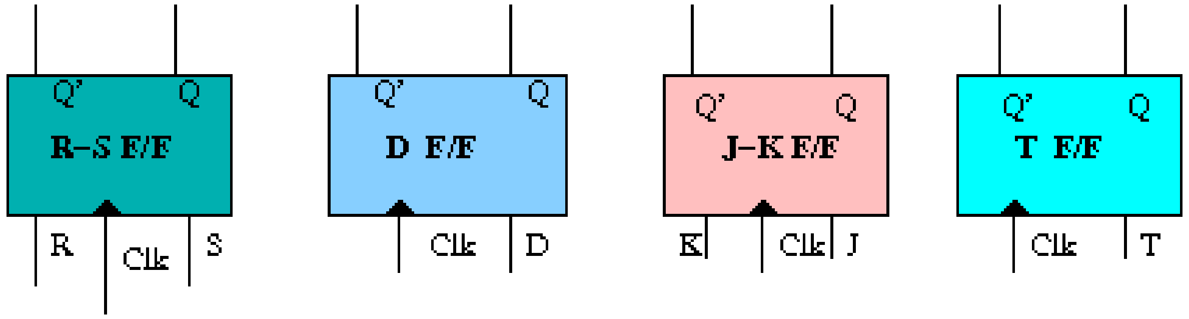





TRUTH TABLE

INPUTS				OUTPUTS	
$\overline{PR}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	X	X
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	0	X	Q_0	$\overline{Q}_0$

Other Types of Flip-Flops



1. RS flipflop If R is high then reset state occurs and when S=1 set state.the both cannot be high simultaneouly. this input combination is avoided.
2. JK flipflop If J and K are both low then no change occurs. If J and K are both high at the clock edge then the output will toggle from one state to the other
3. D flipflop The D flip-flop tracks the input, making transitions with match those of the input D.
4. Tflipflop The T or "toggle" flip-flop changes its output on each clock edge,

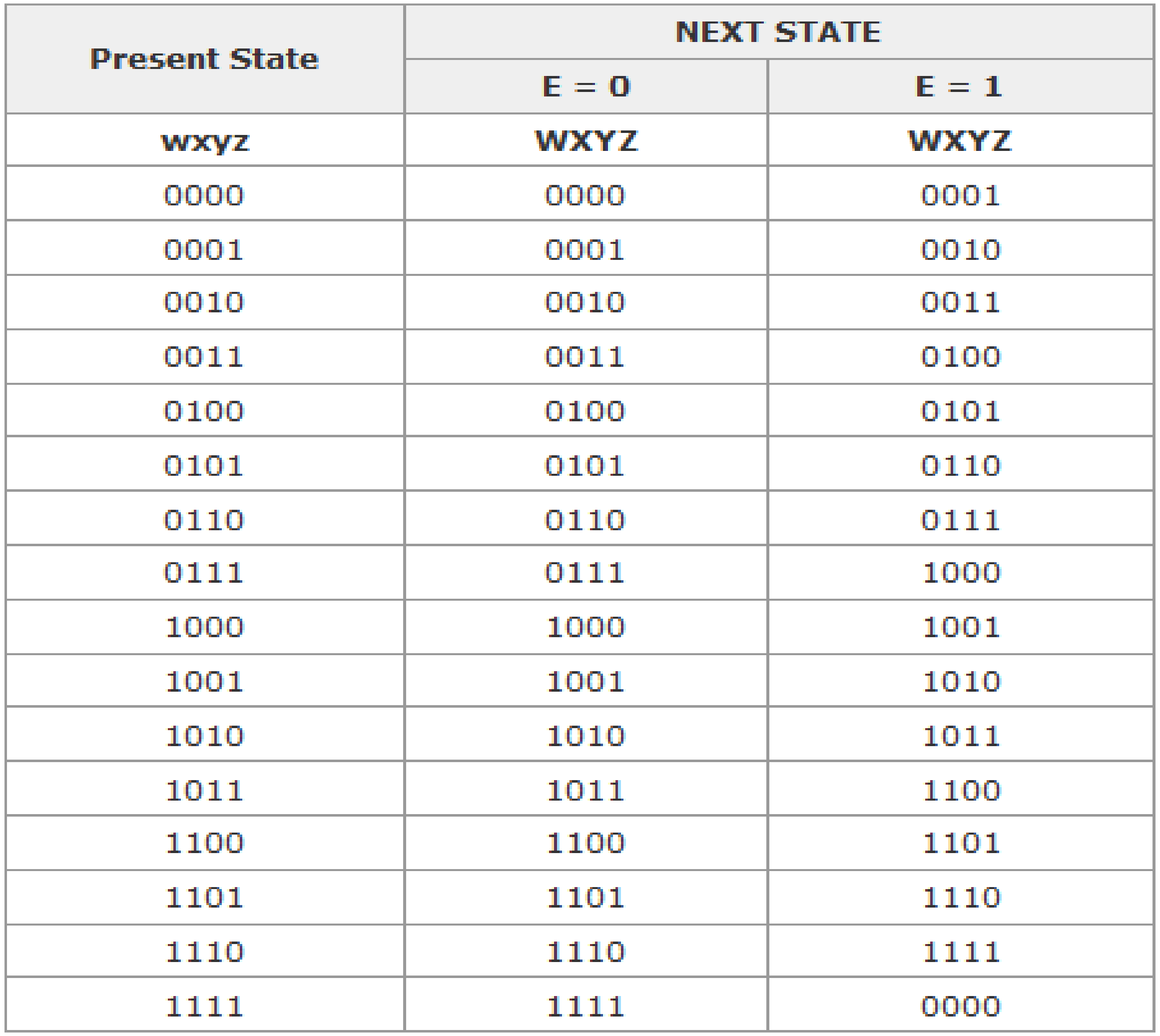
Sequential Logic Building Blocks

- Counters
- Shift registers
- Registers
- Memory Arrays

Counters

- A device which stores the number of times a particular event or process has occurred
- A sequential arithmetic circuit
- Has clock and reset input
- Has N-bit output Q
  - Gives the count
  - http://en.wikipedia.org/wiki/Counter
  - http://www.internetarchaeology.org/www.geocities.com/Templarser/counters.html

Example : Up Counter State Table



Present State	NEXT STATE	
	E = 0	E = 1
wxyz	WXYZ	WXYZ
0000	0000	0001
0001	0001	0010
0010	0010	0011
0011	0011	0100
0100	0100	0101
0101	0101	0110
0110	0110	0111
0111	0111	1000
1000	1000	1001
1001	1001	1010
1010	1010	1011
1011	1011	1100
1100	1100	1101
1101	1101	1110
1110	1110	1111
1111	1111	0000

Down counter example

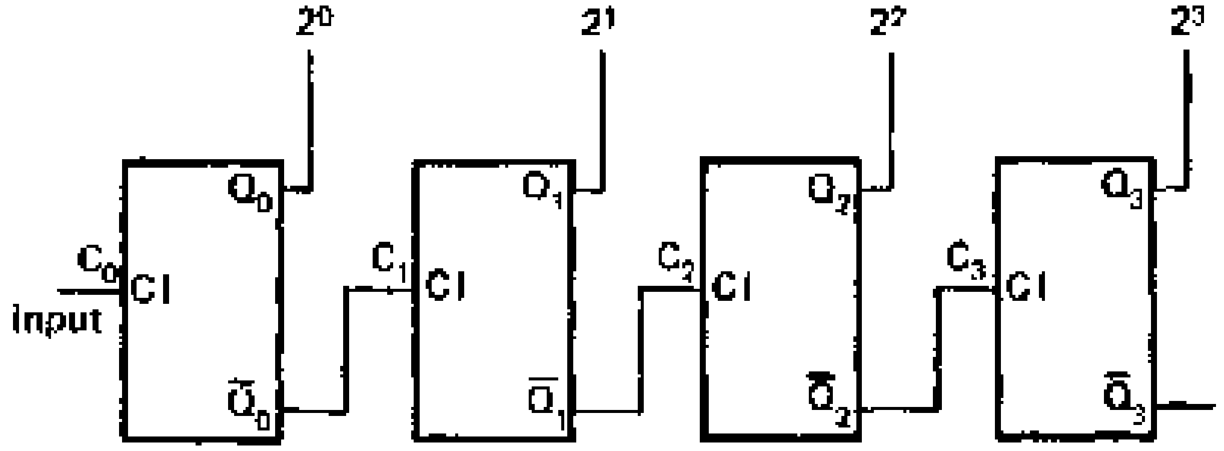
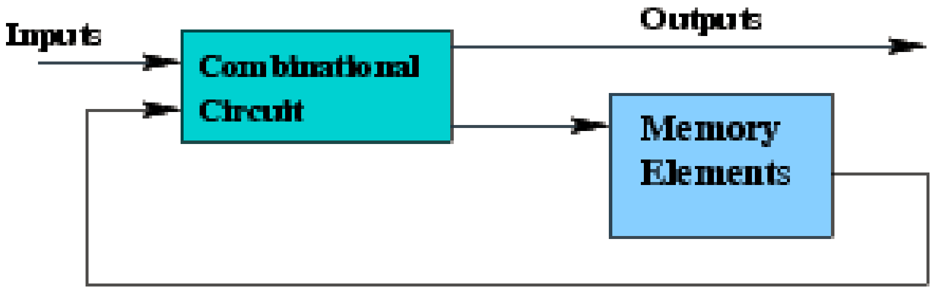


Figure 4.12. *A down-counter constructed by connecting each Q output to the next clock input. Only the Q-to-clock connections are shown here*

Registers

- A group of FF
- Used to hold information bits within a digital system
- Generally used as temporary storage during the computing process
- Constructed from a combination of combinational and sequential logic circuits



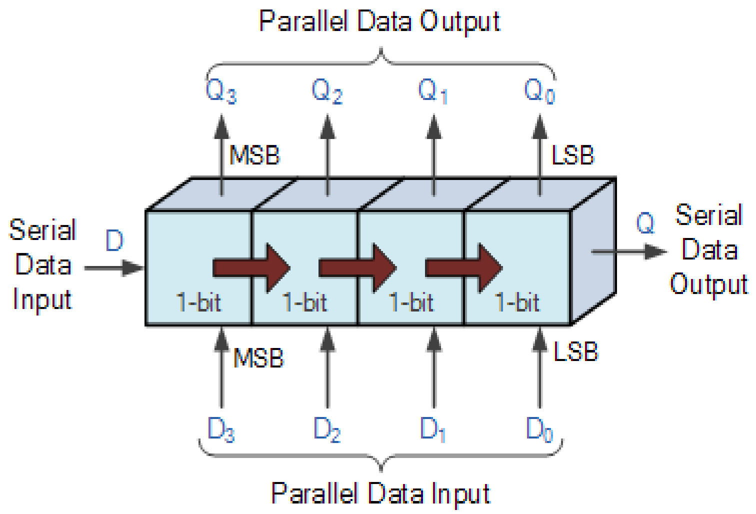
Types of Registers

- Shift registers
  - Serial-in Serial-out (SISO)
  - Serial-in Parallel-out(SIPO)
  - Parallel-in to Serial-out
  - Parallel-in Parallel-out

- Processor Registers

Shift Registers

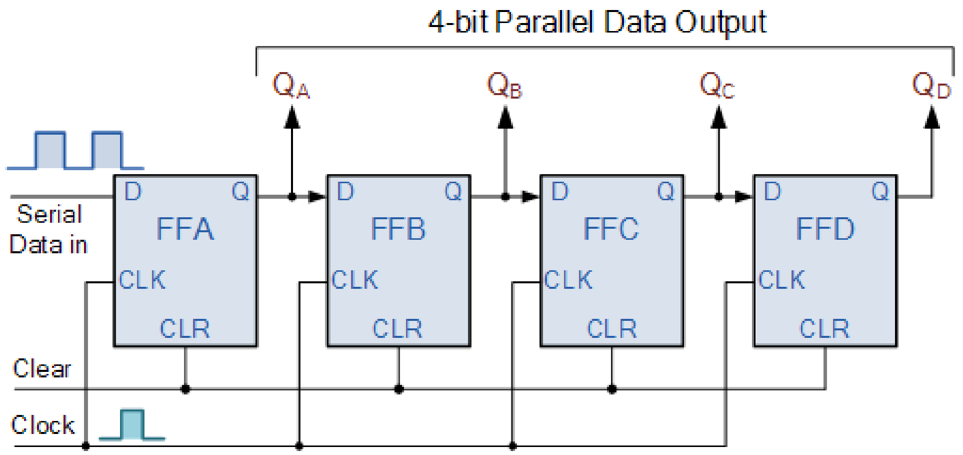
- a cascade of flip flops, sharing the same clock,
- can have both parallel and serial inputs and outputs
- Data is shifted in with each clock pulse and previous data shifted out

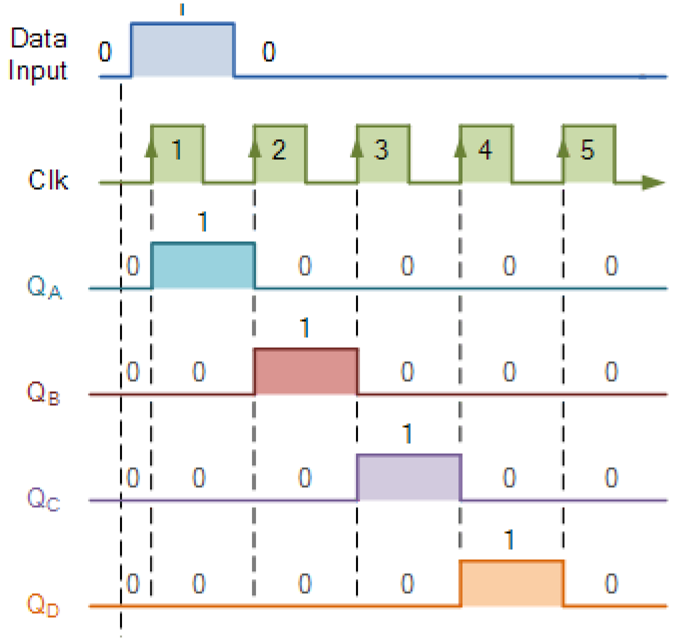


Serial-in to Parallel-out (SIPO) Shift Register

Serial-in to Parallel-out (SIPO) Shift Register

4-bit Serial-in to Parallel-out Shift Register



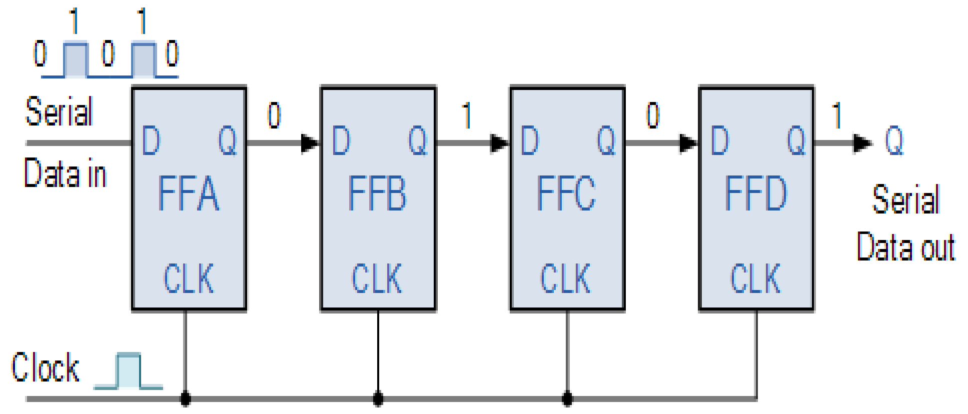


Basic Data Movement Through A Shift Register

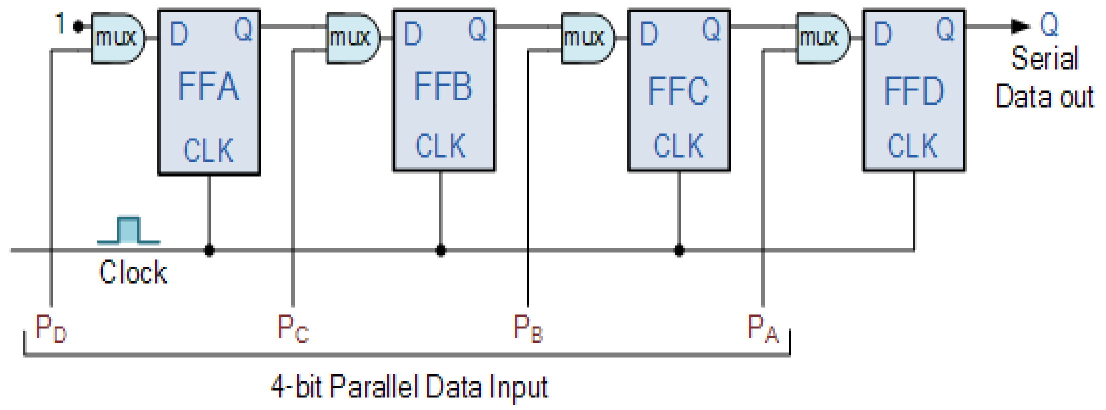
Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Shift Registers cont.

4-bit Serial-in to Serial-out Shift Register

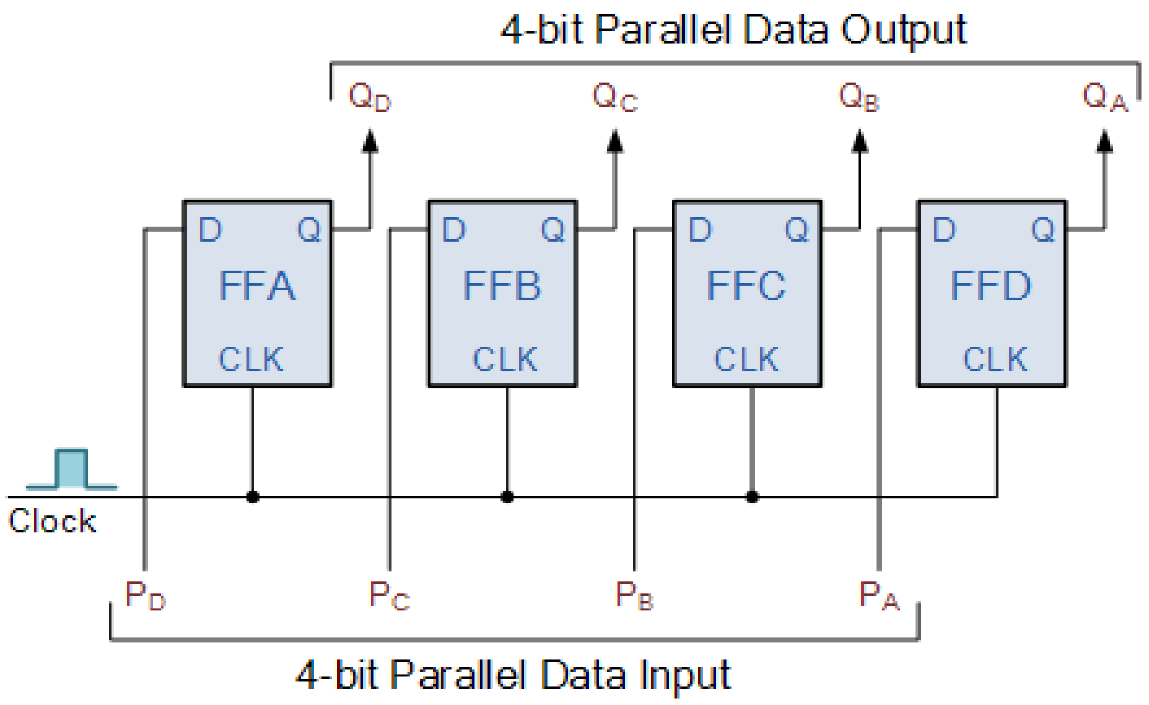


4-bit Parallel-in to Serial-out Shift Register



Parallel-in to Parallel-out

4-bit Parallel-in to Parallel-out Shift Register



This information was taken from the following link
http://www.electronics-tutorials.ws/sequential/seq_5.html

Processor Registers

- A small amount of storage available as part of the CPU
- Used to store data for the CPU
- Provide the fastest way to access data.
- Normally measured by the number of bits they can hold (e.g. 8-bit, 16-bit, 32bit,etc)
- Some accessible to the user and some are not

User Accessible Registers

- Data registers – *holds data, numbers, characters*
- Address registers – holds address info
- General Purpose – can be used for both data or address
- Conditional Registers – *hold truth values*
- Floating Point Registers – *hold floating point data*
- Constant Registers – *a read only register*

User Accessible registers

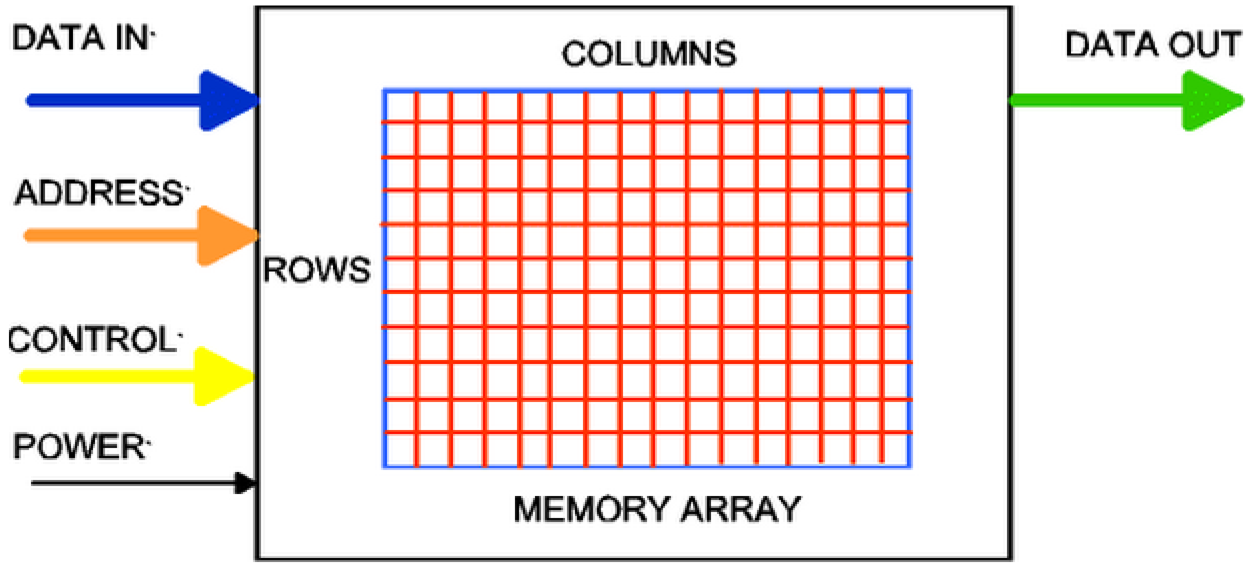
- Special Purpose registers
  - Program counter - *holds the address of the next instruction to be executed*
  - Status register(FLAGS) - *hold status information from ALU and other operations*

Internal registers

- Instruction Pointer(IP) *– holds the instruction that is currently being executed*
- Memory Data Register(MDR) *– holds data to be stored in memory*
- Memory Address Register(MAR) *– holds the address of the location in memory where data in the MDR is to be stored*
-

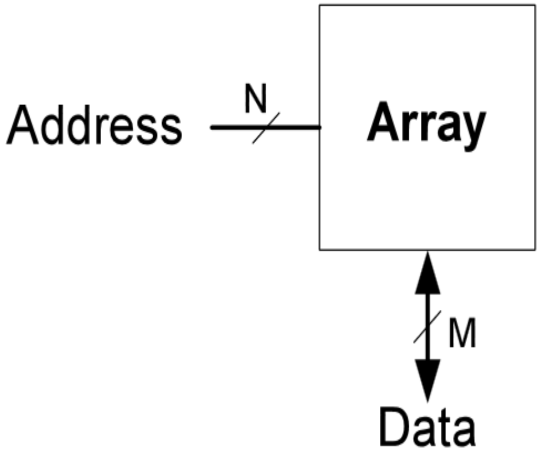
Memory Array

- Bit cells are arrange into rows and columns

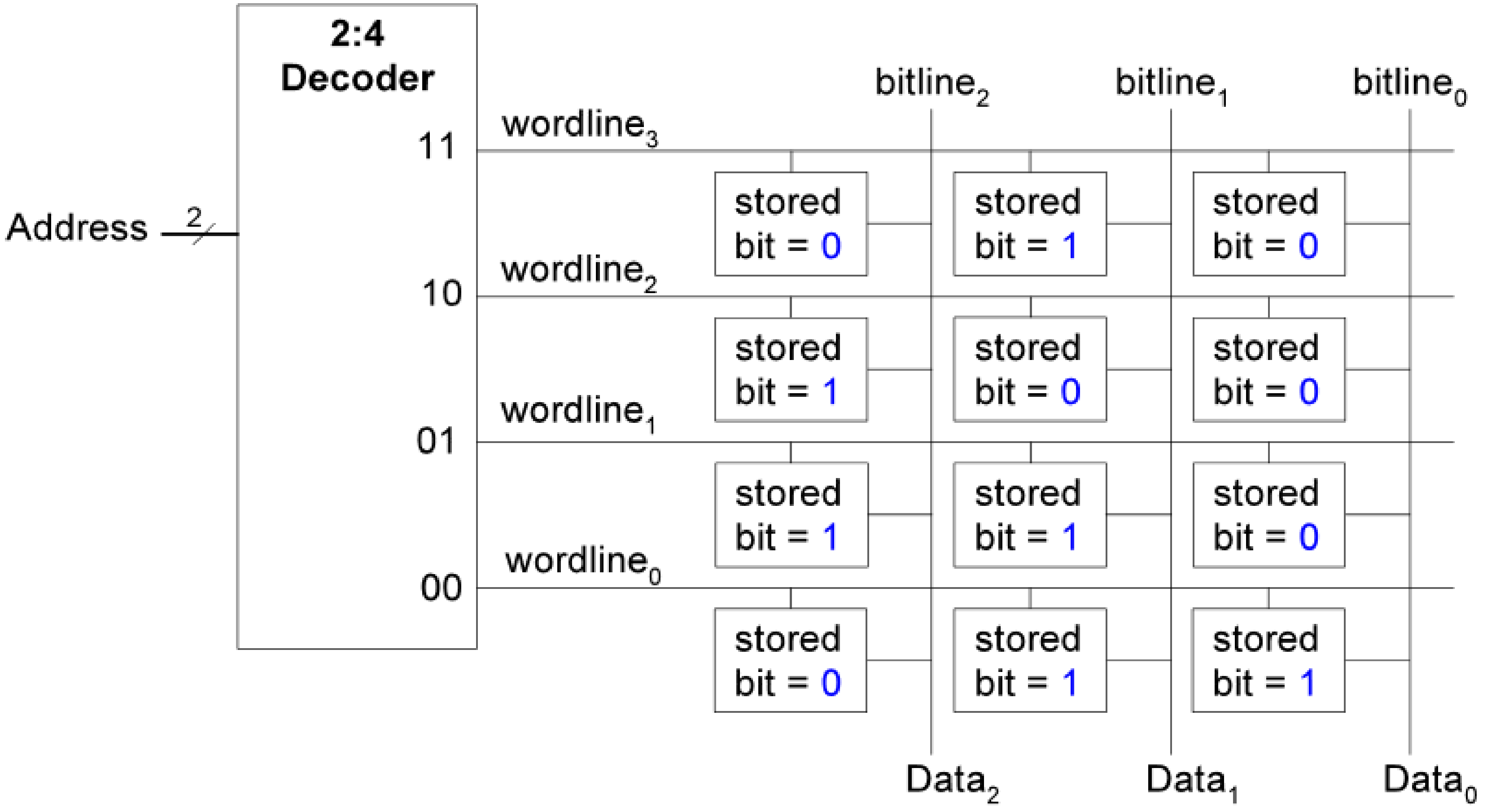


Memory Arrays(cont.)

- Efficiently store and access large amounts of data
- Three common types:
  - – Register Files
  - – Radom Access Memory(RAM)
    - Dynamic random access memory (DRAM)
    - Static random access memory (SRAM)
  - – Read only memory (ROM)
- An M-bit data value can be read or written at each unique N-bit address location.

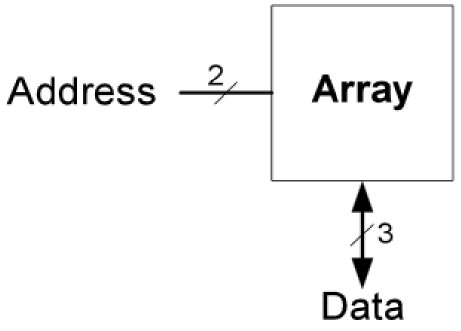


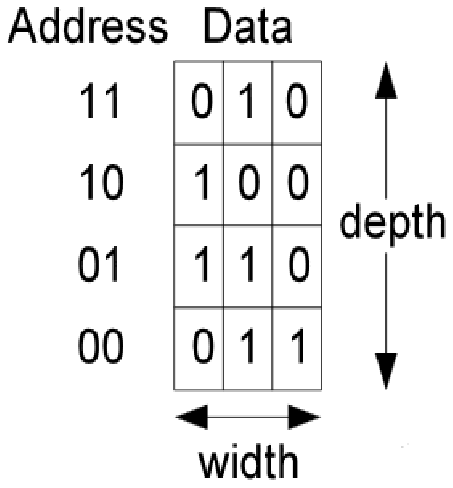
Memory Arrays(cont.)



Memory Array Dimensions

- Two-dimensional array of bit cells
- Each bit cell stores one bit
- An array with *N* address bits and *M* data bits:
  - 2^N rows and *M* columns
  - **Depth:** number of rows (number of words)
  - **Width:** number of columns (size of word)
  - **Array size:** depth × width = $2^N \times M$

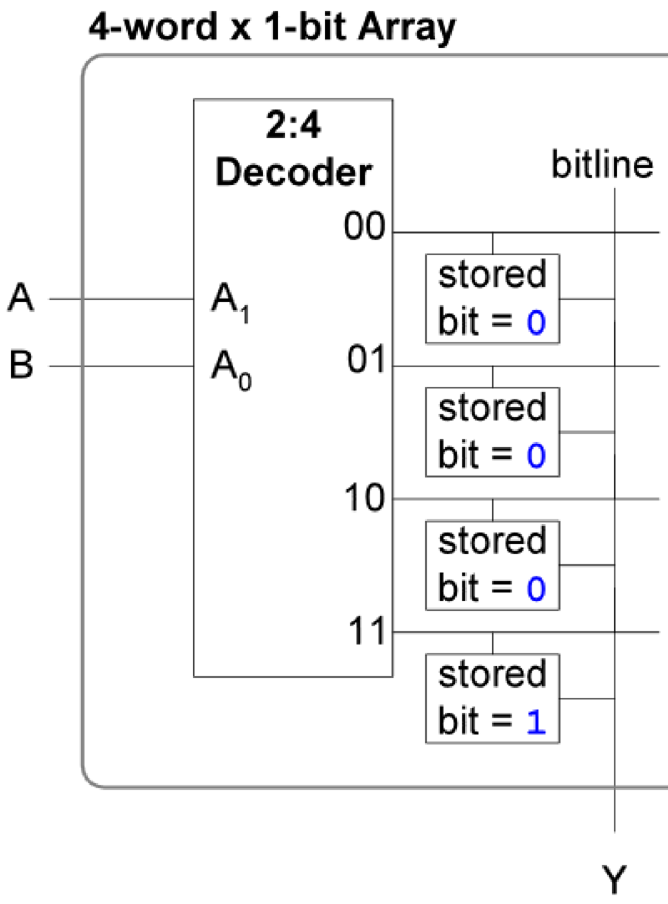




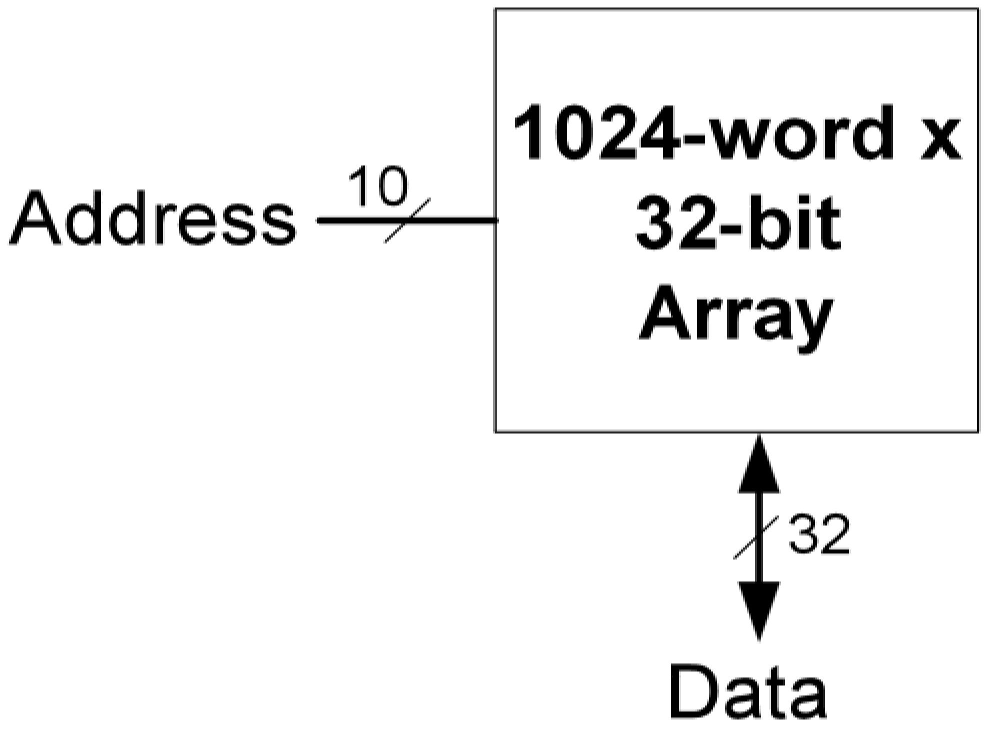
Example: 4 Row memory array

Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

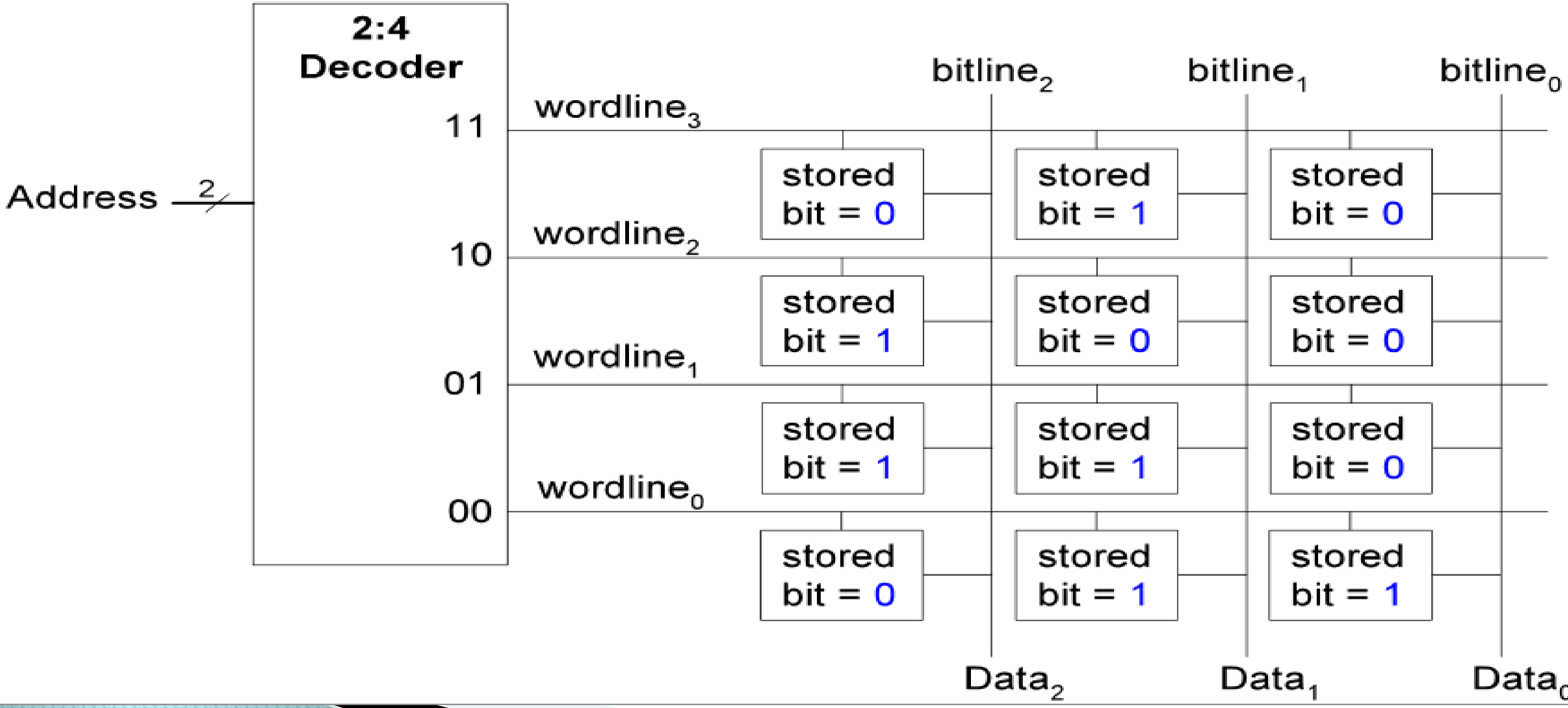


10-bit address with 32-bit words

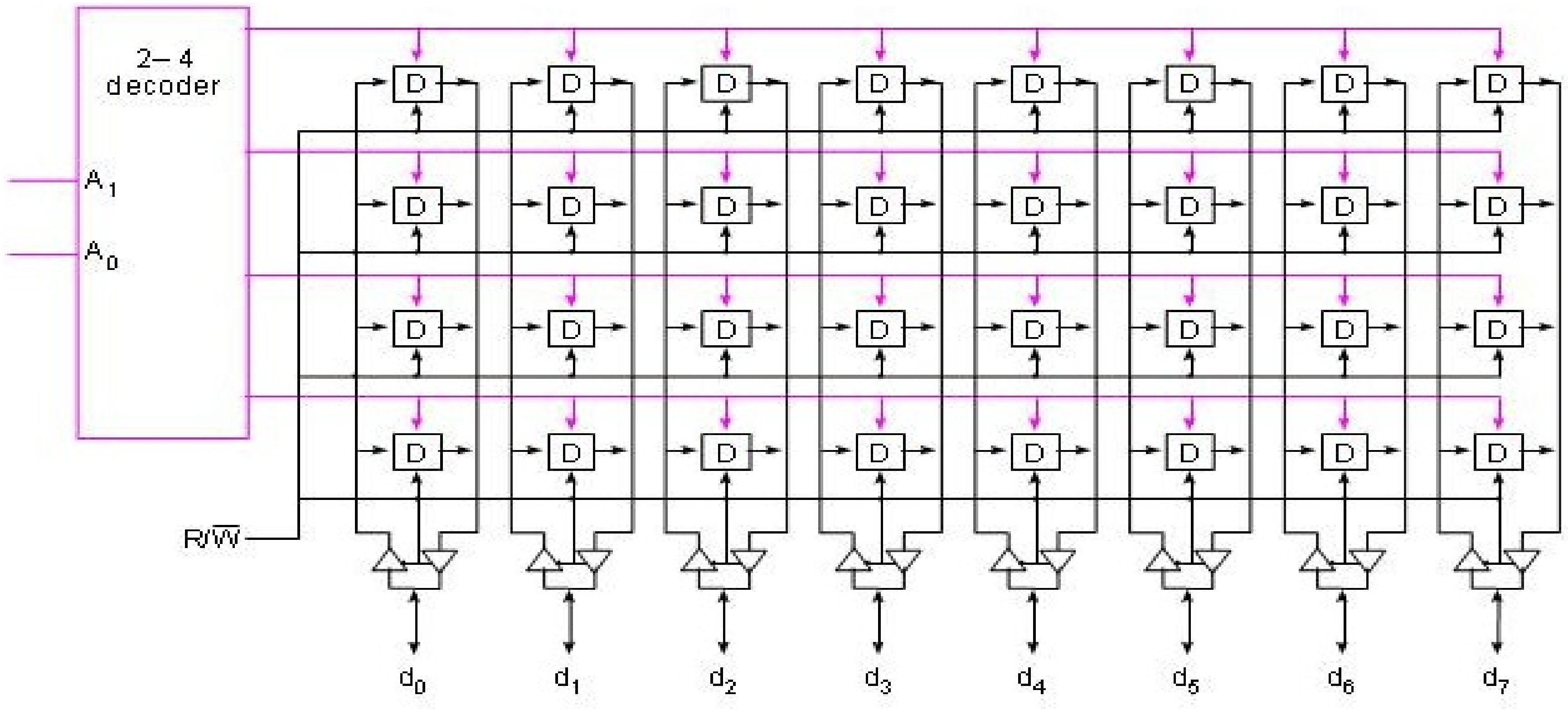


Accessing Rows in a Memory Array

- **Wordline:**
  - similar to an enable
  - allows a single row in the memory array to be read or written
  - corresponds to a unique address
  - only one wordline is HIGH at any given time

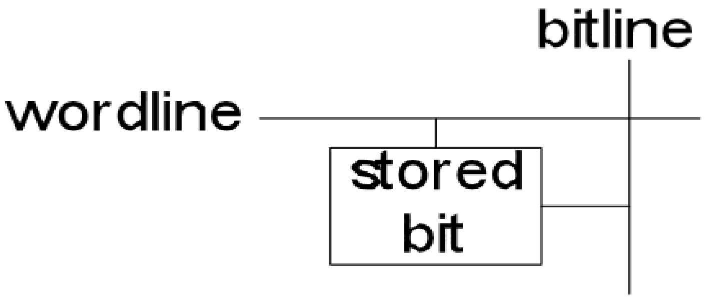


Example Memory Array using DFF

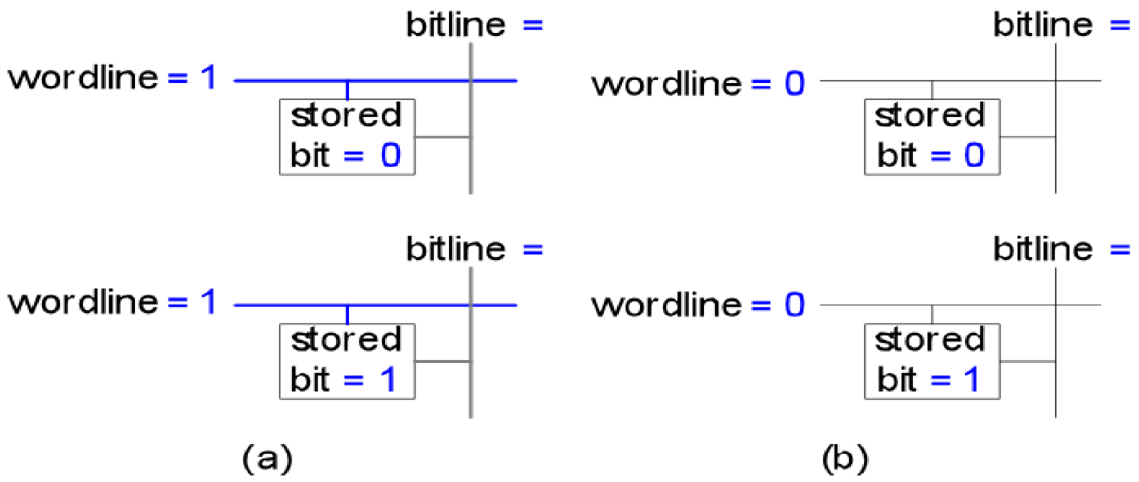


Source: Heuring – Jordan: Computer Systems Architecture and Design

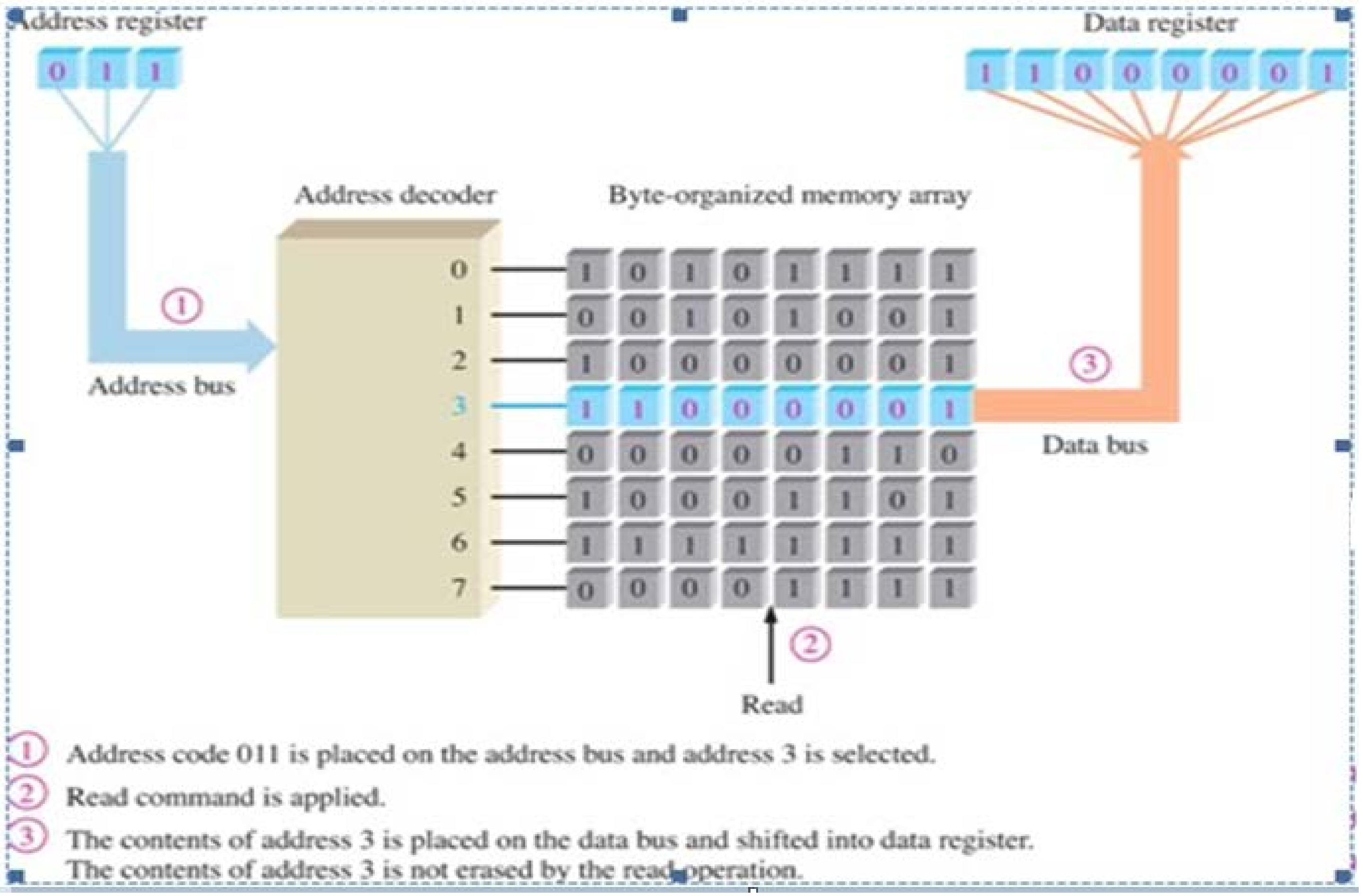
Memory array Bit cells



Example:



Memory READ



1. Address code 011 is placed on the address bus and address 3 is selected.
2. Read command is applied.
3. The contents of address 3 is placed on the data bus and shifted into data register. The contents of address 3 is not erased by the read operation.

Diagram taken from:
http://www.pradipyadav.com/2011_12_01_archive.html

Memory Write

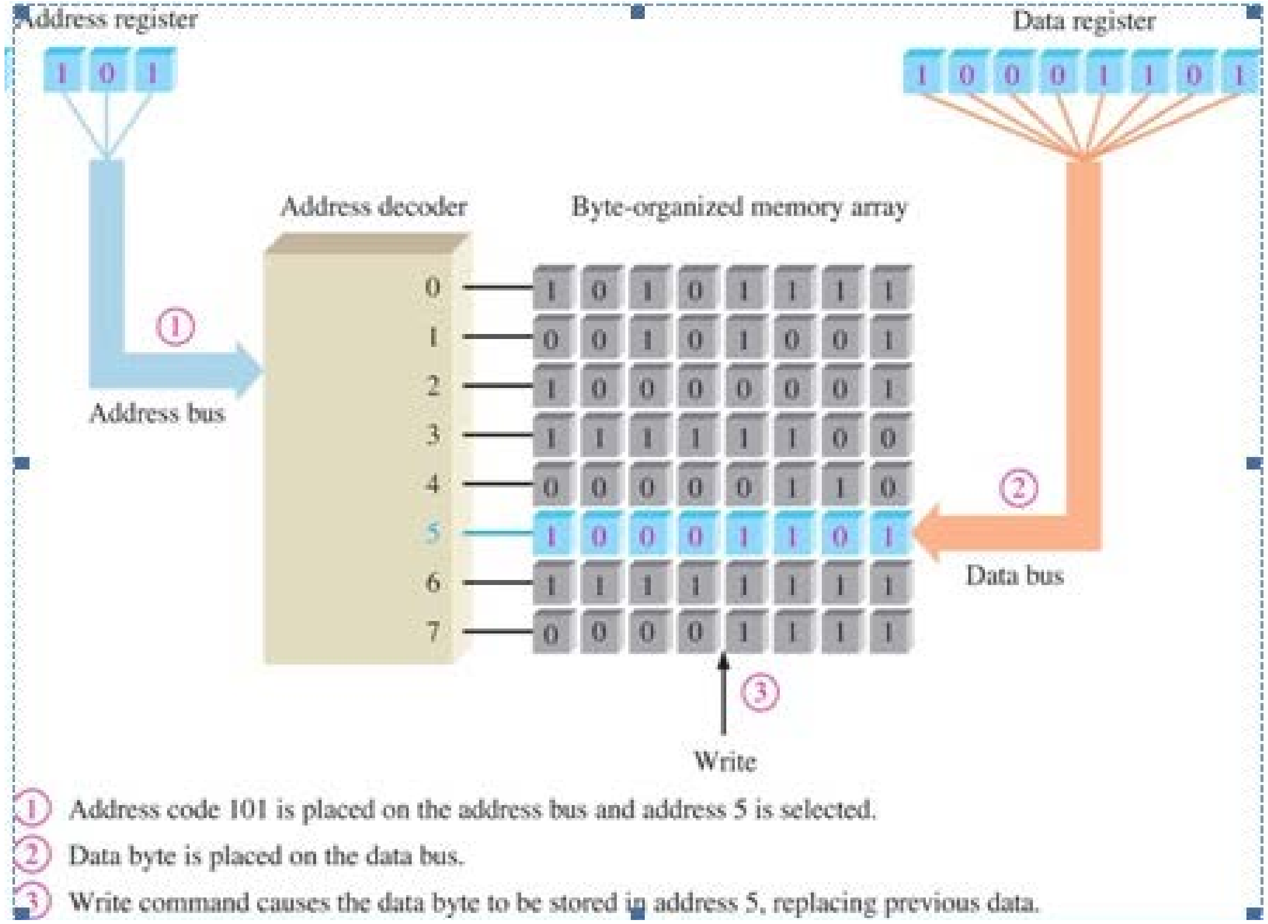


Diagram taken from:
http://www.pradipyadav.com/2011_12_01_archive.html

Memory Array Types

- Memory array used to define several memory devices.
  - RAM
  - ROM
  - Register files
  - etc

Radom Access Memory(RAM)

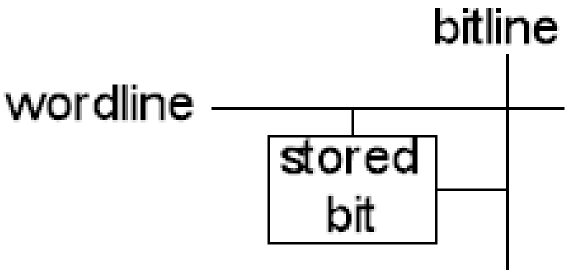
- Volatile: loses its data when powered off
- Read and written quickly
- Main memory in your computer is RAM(DRAM)

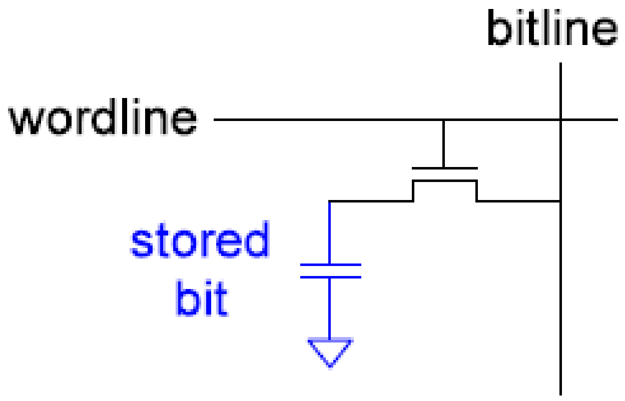
Types of RAM

- Two main types of RAM:
  - – Dynamic random access memory (DRAM)
  - – Static random access memory (SRAM)
- Differ in how they store data:
  - – DRAM uses a capacitor
  - – SRAM uses cross-coupled inverters
    - Or other types of gates, example shown with FF

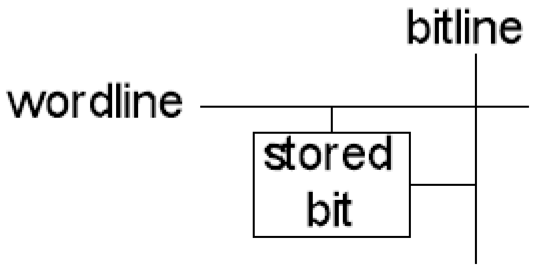
DRAM

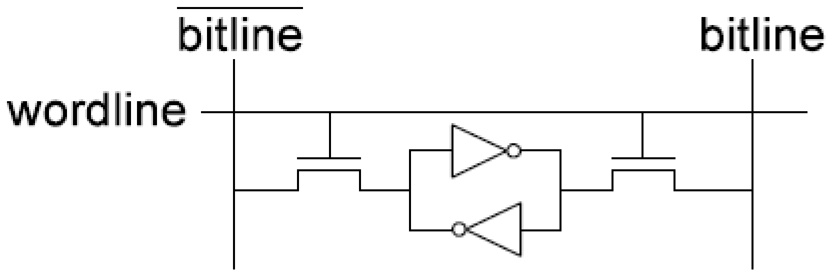
- Data bits stored on a capacitor
- Called *dynamic* because the value needs to be refreshed (rewritten) periodically and after being read:
  - Charge leakage from the capacitor degrades the value
  - Reading destroys the stored value





SRAM


bitline
wordline
stored bit

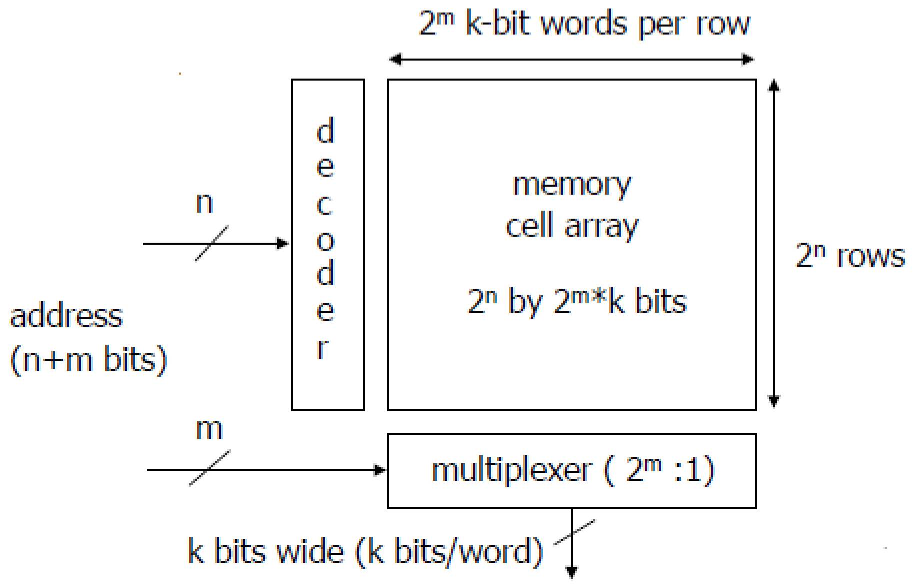

$\overline{\text{bitline}}$
bitline
wordline

Read Only Memory(ROM)

- Nonvolatile: retains data when power is turned off
- Read quickly, but writing is impossible or slow
- Flash memory in cameras, thumb drives, and digital cameras are all ROMs

Addressing Memory

- Want square memory array
- Want simple decoding logic
  - Problem: A 1Meg×1 RAM uses 1,048,576 20-input NANDs?
- Want short data & address lines
- Power of 2 addressing



Memory Addressing cont.

- Recall from lecture 1
  - 8-bits as a byte
  - 16 bits = ½ word
  - 32 bits = word
- Memory is addressed in bytes
  - i.e. each addressed location is 8-bits

Address Bus

- A set of wires used to connect the CPU and memory.
- Used by the CPU to assert the specific Memory Address location to be read or written
- The number of wires (one for each bit) in the address bus determines the maximum size of memory which the processor can access.

Data Bus

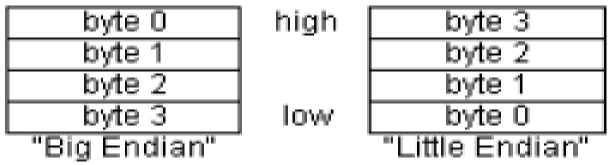
- A set of wires used to connect the CPU and memory.
- Used to transfer data to or from memory.
- The number of wires (one for each bit of data) in the Data bus determines the maximum size data that can be transferred during a read or a write.

How data is stored in Memory

Storage of Data Structure

Big Endian and Little Endian :

For multiple-byte data items stored in memory, need to specify which order:
(a) Most Significant 8 bits at lowest address ("Big Endian"), OR
(b) Least Significant 8 bits at lowest address ("Little Endian")



Little Endian: 80x86
Big Endian: mc680x0, SPARC, HP Precision (Vipers)
Difference in "endian-ness" can be a problem when transferring data between machines

Putting it all together

- A processor is constructed from abstract layers of:
  - Boolean logic
  - Combinational logic
  - Sequential logic
- Each component designed to carry out specific tasks
- Components fit together to implement processor

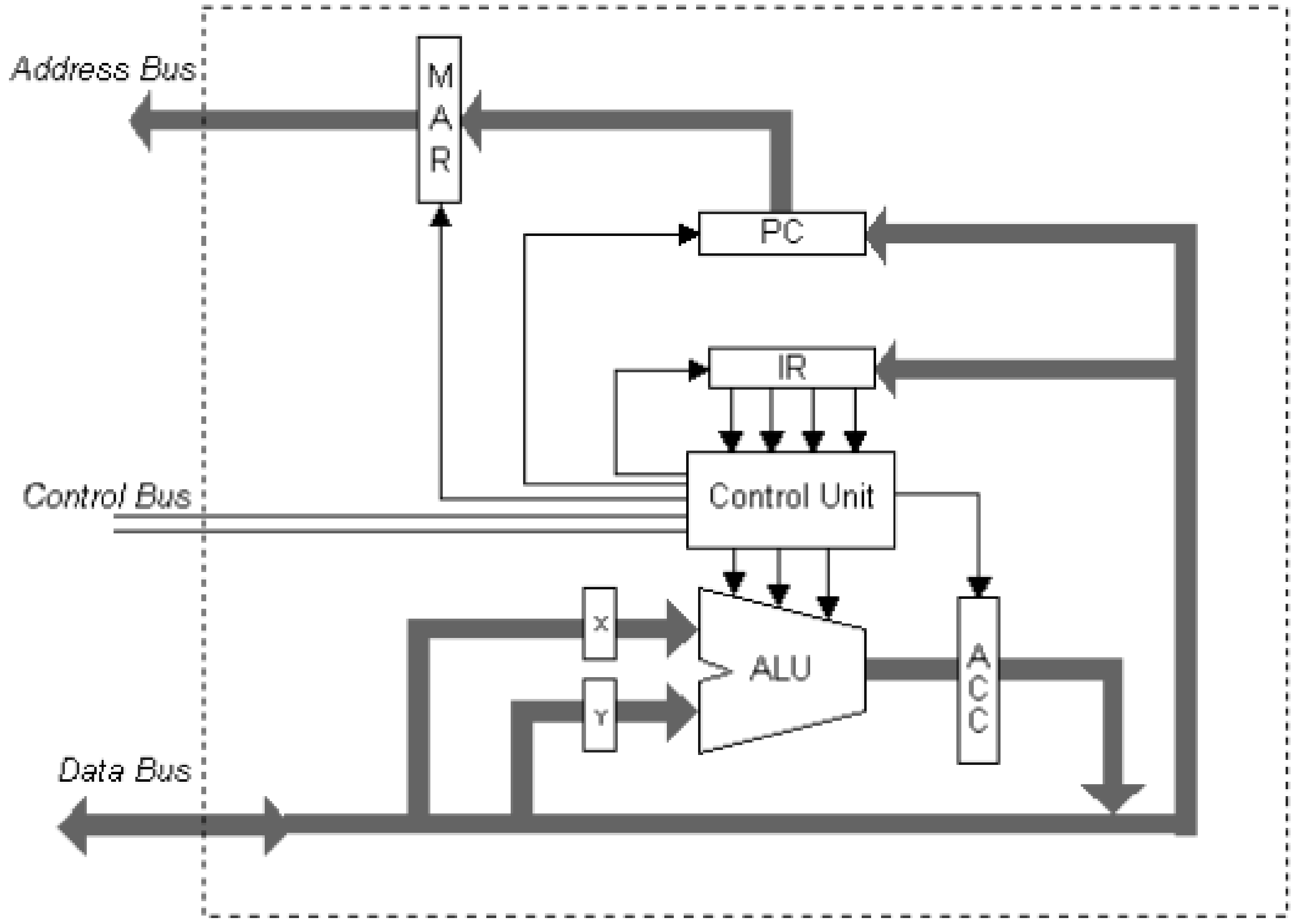
Primary CPU Components

1. Control unit
   1. Generates control/timing signals
   2. Controls decoding/execution of instructions
2. ALU
   1. Used during execution of instructions
   2. Mathematical operations: */+- etc.
   3. Logical operations: shift, rotate, AND, NOT, etc.
3. Registers:
   1. PC(a.k.a. Instruction Pointer, IP)
   2. IR
   3. Stack Pointer: address the top of the stack
   4. Base Pointer: address the bottom of the stack
   5. Accumulator: contains the result of the ALU operations
   6. General Purpose Registers: hold temporary results or addresses during execution

CPU Components cont.

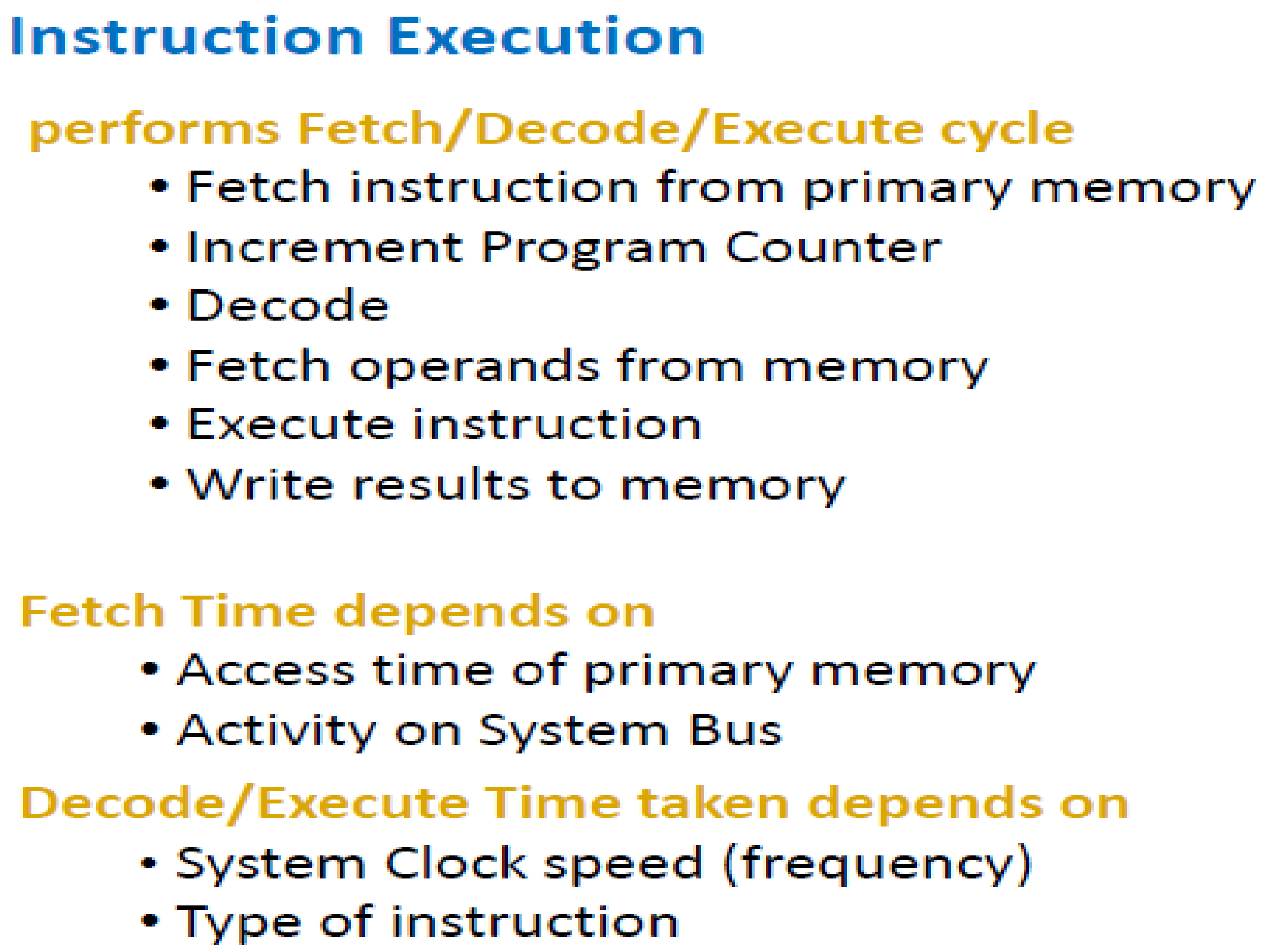
4. Address bus
5. Data Bus
6. Control bus

CPU Components



Simple CPU

Executing Instructions



Instruction Execution

performs Fetch/Decode/Execute cycle

- Fetch instruction from primary memory
- Increment Program Counter
- Decode
- Fetch operands from memory
- Execute instruction
- Write results to memory

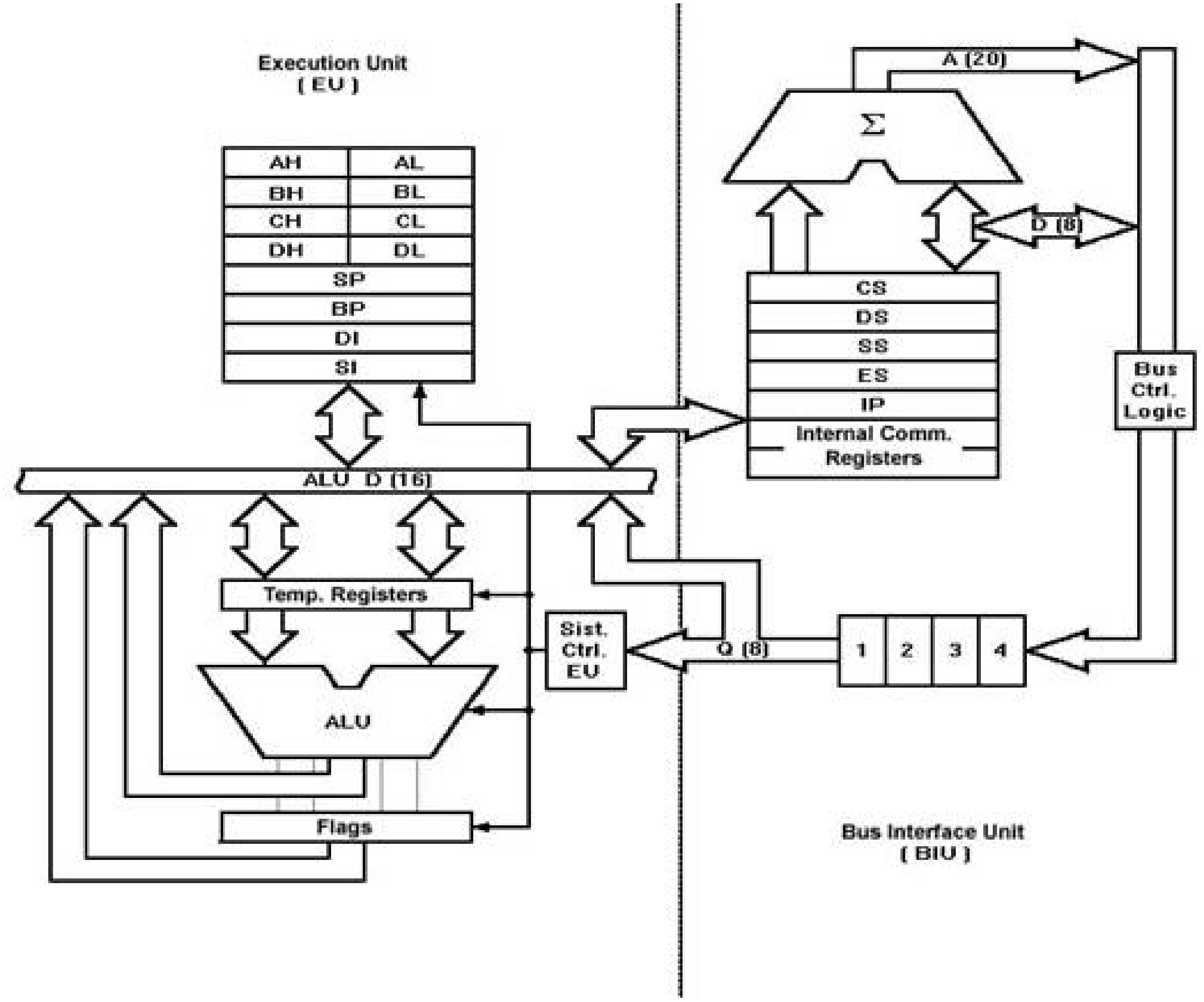
Fetch Time depends on

- Access time of primary memory
- Activity on System Bus

Decode/Execute Time taken depends on

- System Clock speed (frequency)
- Type of instruction

8086 Processor Architecture



Intel 8086 Architecture

- This is a 16-bit processor (because its register files are 16-bit)
- 8086 was introduced in late 1970’s; Intel’s modern x86 processors remain compatible with this architecture

Image courtesy of Wikimedia Commons

Instruction Set Architecture(ISA)

- Defines all the operations that the CPU can perform
- Defines how users interface with the CPU
- To understand ISA we will be looking at the ISA for the 8086 processor

Exercise

- Identify the different types of components in the 8086 processor
- What are the size of its Registers?
- What is the size of memory Addresses the CPU can handle?
- Take a look at the instruction set for the 8086 provided on Slate

Exercise cont.

- The specification of a memory array is given as 1024 x 4Byte.
    1. What is the size of this memory?
    2. Assuming this memory array can only be addressed as 32bit words how many bits make up the an Address?
    3. If this memory array can be addressed in minimum groups of 16 bits. Will there need to be a change in the number of address bits required? If so how by much?
    4. What component is used to select which group of 16-bits in the word is sent the memory output?
    5. Describe the specification of the component