## Concepts Introduced

- truth tables, logic equations, and gates
- combinational logic
- sequential logic



## Digital Electronics

- Digital electronics operate at either high or low voltage.
- Computers use a binary representation since it matches the underlying electronic states.
- Rather than referring to voltage levels, designers refer to signals as being asserted (logically true) or deasserted (logically false).



## Logic Blocks

- A logic block is a group of logic elements that are connected in some way.
- Logic blocks are categorized into two types.
  - combinational logic - A logic system whose blocks do not contain memory and compute the same outputs given the same inputs.
  - sequential logic - A logic system that contains memory (state) whose value depends on the inputs as well as the current contents of the memory.



## Truth Tables

- A truth table is a table used in logic to specify the outputs for each possible set of inputs, which is useful for specifying combinational logic since combinational logic output only depends on its inputs.
- For a logic block with $n$ inputs, there are $2^n$ entries in the truth table.

## Truth Table Example

- Assume D is true if at least one input is true, E is true if exactly two inputs are true, and F is true only if all three inputs are true.

| Inputs | | | Outputs | | |
|---|---|---|---|---|---|
| A | B | C | D | E | F |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |



## Logic Equations

- Logic equations provide a more concise way to specify logic.
- Logical `OR` operator for operands A and B is written as $A + B$. The result of the operation is called a *logical sum*.
- Logical `AND` operator for operands A and B is written as $A \cdot B$. The result of the operation is called a *logical product*.
- Logical `not` operator for operand A is written as $\overline{A}$ or $A'$.
- A logic equation performs logical operations on one or more inputs and produces a single output.



## Logic Equations Example

- What are the logic equations for D, E, and F in the previous truth table, where D is true if at least one input is true, E is true if exactly two inputs are true, and F is true only if all three inputs are true?

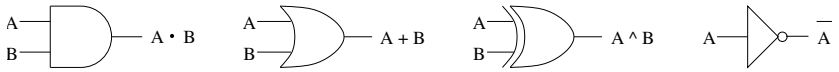
$$D = A + B + C$$
$$E = (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot C)$$
$$F = A \cdot B \cdot C$$



## Gates

- A gate is a electronic device that implements basic logic functions.

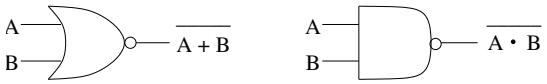
| A | B | A and B | A or B | A xor B | not A |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |



## NOR and NAND Gates

- Any logical function can be constructed using `AND` gates, `OR` gates, and inversion (`NOT` gates).
- All logical functions can be constructed with a single *universal* gate type, if that gate is inverting. Two common inverting gates are the `NOR` and `NAND` gates.

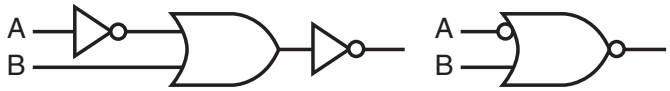
| A | B | A nor B | A nand B |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |






## Logic Blocks

- Logic blocks are built from gates that implement basic logic functions.
- The example below shows the logic gate implementation of $\overline{\overline{A} + B}$ using explicit inverts on the left and bubbled inputs and outputs on the right.





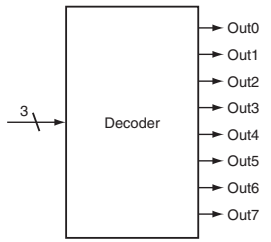

## Common Logic Blocks

- Gates connected together with no state elements are called combinational logic as its outputs depend only on its inputs.
- There are several combinational logic blocks that are commonly used.
  - decoders
  - multiplexors
  - programmable logic arrays (PLAs)



## Decoders

- A decoder is a logic block that has an $n$-bit input and $2^n$ outputs, where only the $i$th output signal is asserted when the input combination represents the binary value $i$.
- The following example depicts 3-bit decoder and a truth table. This decoder is also called a 3-to-8 decoder since there are 3 inputs and 8 ($2^3$) outputs.




a. A 3-bit decoder

| Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| I2 | I1 | I0 | Out7 | Out6 | Out5 | Out4 | Out3 | Out2 | Out1 | Out0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

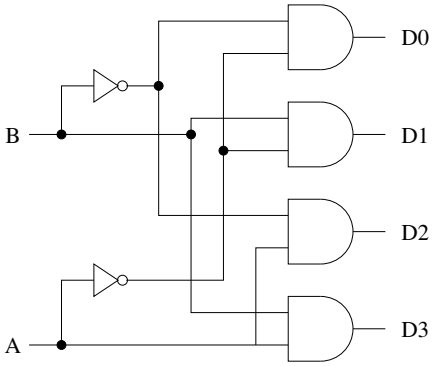
b. The truth table for a 3-bit decoder

## Implementing a 2-to-4 Decoder

- Below is the truth table and logic block for a 2-to-4 decoder.
- The logic equations for the truth table are:

$D0 = \overline{A} \cdot \overline{B}$ $D1 = \overline{A} \cdot B$ $D2 = A \cdot \overline{B}$ $D3 = A \cdot B$

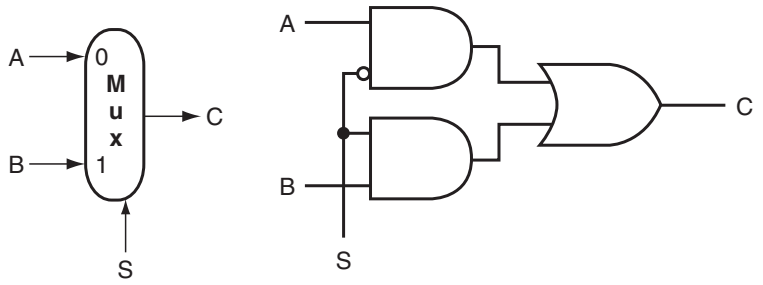
| Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| A | B | D3 | D2 | D1 | D0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |






## Multiplexors

- A multiplexor is a device that selects from different input values based on the setting of control lines.
- The figure below shows the symbol for a two-input multiplexor and its implementation with gates on the right.






## Multiplexors in General

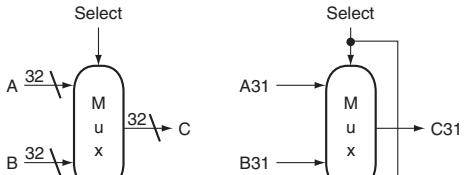
- In general, if there are $n$ data inputs, then there has to be $\lceil log_2 n \rceil$ selector inputs.
- A multiplexor consists of three parts.
  - A decoder that generates $n$ signals, each indicating a different input value.
  - An array of $n$ AND gates, each combining one of the inputs with a signal from the decoder.
  - A single OR gate that takes as input the outputs of the AND gates.



## Array of Logic Elements

- Many combinational operations are performed on data that is 32 bits wide.
- We can build an array of similar logic elements and show that a given operation will occur on a collection of inputs.
- A *bus* is a collection of data lines treated as one logical signal.

## Array of Logic Elements Example




a. A 32-bit wide 2-to-1 multiplexor




b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors



## Two Level Logic

- Any logic function can be written in a canonical form, where every input is a true or complemented variable and there are only two levels of gates, consisting of one level being AND gates and the other level being OR gates.
- A *sum of products* is a canonical form that employs a logical sum (OR) of products (terms joined using the AND operator).
- A *product of sums* is a canonical form that employs a logical product (AND) of sums (terms joined using the OR operator).



## Producing Sum of Products from a Truth Table

- A sum of products representation can be produced from a truth table. A logical product is all the inputs or their complements when an entry is a 1 or 0, respectively. The sum of products is the logical sum of the products where the output is true.
- Sum of products for the outputs in the truth table are:

$$D = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot\overline{C} + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$

$$E = \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C}$$

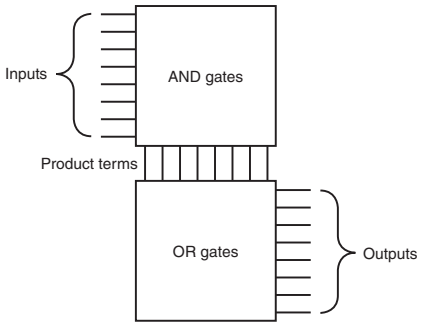
$$F = A\cdot B\cdot C$$

| Inputs | | | Outputs | | |
|---|---|---|---|---|---|
| A | B | C | D | E | F |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |



## Programmable Logic Array (PLA)

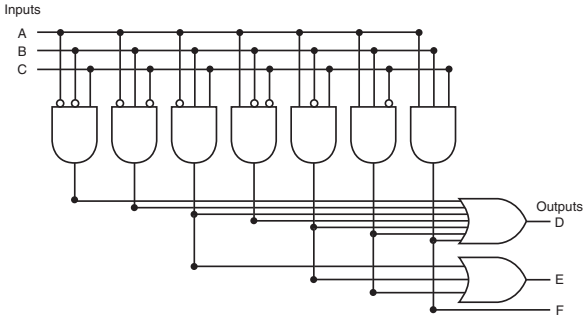
- Programmable logic arrays (PLAs) are logic implementations that build on the sum of products representation.
- Has two stages of logic:
  - First stage is an array of AND gates to form a set of product terms.
  - Second stage is an array of OR gates to form a logical sum of the product terms.



## PLA Example

- Below is an example PLA implementing the logic function represented by the truth table.
- When drawing a logic block, dots are used to indicate the connection between signal, input, and output lines.

| Inputs | | | Outputs | | |
|---|---|---|---|---|---|
| A | B | C | D | E | F |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |





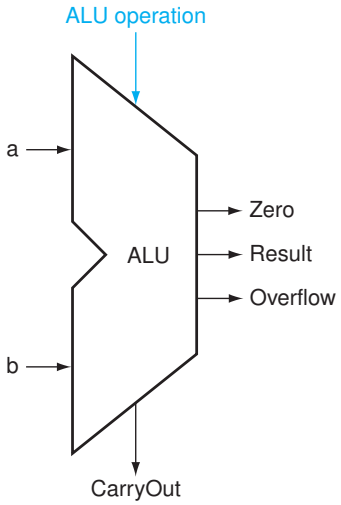

## Read-Only Memory (ROM)

- Read-only memory (ROM) is fixed at the time the ROM is manufactured.
- A ROM has a set of input address lines and a set of outputs.
- A ROM with $2^m$ addressable entries (height) has $m$ input lines.
- The number of bits in each addressable entry is equal to the number of output bits (width).
- A ROM can encode the logic functions associated with a truth table. The entries in the input portion of the truth table represents the addresses of the entries of the ROM. The contents of the output portion of the truth table comprise the contents of the ROM.
- A ROM is similar to PLA, except that the ROM produces a full output word for every possible input combination (fully decoded).



## Universal ALU Symbol

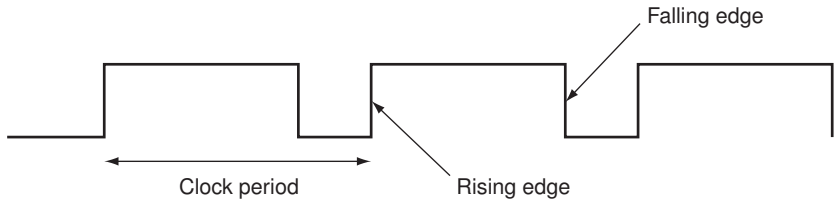
- Below is the universal symbol that is used to represent a complete ALU.






## Clocks

- A *clock* is a signal that ocsillates between low and high voltages in a fixed period of time.
- The *clock cycle time* or *clock period* is the time between two transitions from a low voltage to high voltage (rising edges) or the time between two transitions from a high voltage to a low voltage (falling edges).
- *Edge-triggered clocking* means all state changes occur on the active (rising or falling) clock edge.



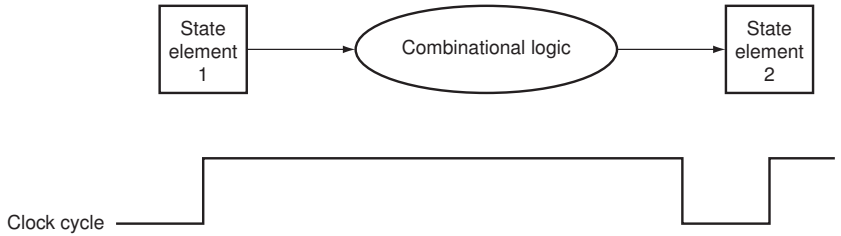
## State Elements

- A *state element* has some type of internal storage and at least two inputs and one output.
- The required inputs to a state element are the data value to be written and the clock signal, which indicates when the data value is to be written.
- The output from a state element is the value that was written on a previous cycle.
- Some state elements are only written when there is an explicit write signal is active.



## Synchronous System

- A signal is valid when it is stable (not changing) and the value will not change again until the inputs change.
- A *synchronous system* is a logic system that employs clocks and data signals are read only when the clock indicates that the signal values are stable (not changing).
- In a synchonous system, state elements provide inputs to a combinational logic block and the outputs of the combinational logic are stored in state elements, which occurs only on the active clock edge.





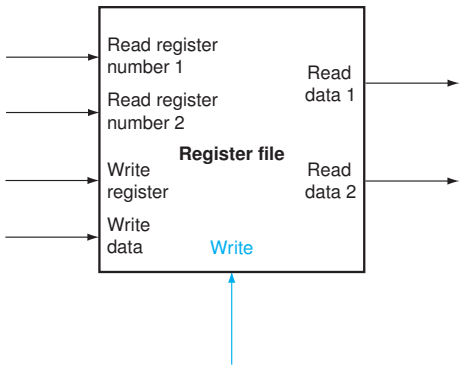

## Latches and Flip-Flops

- *Latches* and *flip-flops* are the simplest state elements.
- The difference is the point at which the clock causes the state to change. In a clocked latch, the state can change when the clock signal is asserted and in a flip-flop, the state is changed only on an active clock edge.



## Register Files

- A register file of 32-bit values can be implemented as an array of registers built from flip-flops, where each register requires 32 flip-flops.
- Typically a register file has at least two read ports and one write port.



## SRAM

- SRAM - Static Random Access Memory
- Used in caches.
- Usually has a single access port that can provide either a read or write access.
- Requires 6 transistors per bit to prevent data from being corrupted when read and requires 4 times the amount of space as compared to DRAM for each stored bit.
- Each bit value is stored in a cell by using a pair of inverting gates and the value can be kept indefinitely as long as power is applied, which is why SRAM is called static.
- SRAM assess time is about 5 to 10 times faster than DRAM.
- SRAM is perhaps 20 times more expensive than DRAM.
- Synchronous SRAM (SSRAM) has a synchronous interface to allow burst transfers, where a clock is used to transfer successive words given only a starting address and length.



## DRAM

- DRAM - Dynamic Random Access Memory
- Used for main memory.
- Requires a single transistor per bit, which is lost after being read, so each read requires that the data be written back.
- The value representing a bit is kept in a cell that is stored as a charge in a capacitor that is accessed by the single transistor.
- DRAM requires that the data be refreshed periodically, about 1% to 2% of the cycles, which is why DRAM is called dynamic and is accomplished by reading the data and writing it back.



## Finite State Machines

- The behavior of sequential systems depends on both the inputs and the contents of internal memory and *finite state machines* (FSMs) are used to describe their behavior.
- A finite state machine consists of:
  - set of states
  - A next-state function that given the current state and current inputs determines the next state.
  - An output function that produces a set of outputs from:
    - the current state only (Moore machine)
    - the current state and the current inputs (Mealy machine)
- The finite state machines we will examine are synchronous meaning that a new state is computed once per clock cycle.



## Implementing a Finite State Machine

- A FSM is implemented with a state register that holds the current state and a combinational logic block to compute the next state and the outputs.