

15. Hardware and Virtual Machines

15.1 Processors and Parallel Processing

RISC and CISC

- **Complex Instruction Set Computers (CISC)**
 - To carry out a given task with **less assembly code**.
 - The processor converts a single, complex instruction into sub-instructions.
 - The sub-instructions are named as **microcodes**.
- **Reduced Instruction Set Computers (RISC)**
 - To carry out a given task with **higher processor performance**.
 - There is a smaller, *more optimized* set of instruction.
 - The assembly code instructions are simpler single-cycle instructions.
 - Uses a system where cache is split between data and instructions.

CISC	RISC
Many instruction formats	Fewer instruction formats/sets
More addressing modes	Fewer addressing modes
Multi-cycle instructions	Single-cycle instructions
Variable-length instructions	Fixed-length instructions
Longer execution time for instructions	Faster execution time for instructions
Decoding of instructions is more complex	Using general-purpose registers
More difficult to make pipelining work	Easier to make pipelining function
Microprogrammed control unit	Hard-wired control unit
Fewer Registers	Many registers
More use of cache	More use of RAM

Pipelining

- A strategy to **improve processing efficiency** by allowing **parallel instruction processings**.
1. Instructions are divided into **5 subtasks**.
 - Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Instruction Execute (IE), Write Back Result (WB)
 2. Each subtask is completed during one clock cycle.
 3. No two instructions execute their same stage at the same clock cycle.
 4. The second instruction begins in the second clock cycle, while the first instruction has moved on to its second subtask.
 5. The third instruction begins in the third clock cycle, while the first and second instructions move on to their second and third subtasks, respectively, and etc.

		Clock cycles									
Processor stages		1	2	3	4	5	6	7	8	9	10
	IF	A	B	C	D	E	F				
	ID		A	B	C	D	E	F			
	OF			A	B	C	D	E	F		
	IE				A	B	C	D	E	F	
	WB					A	B	C	D	E	F

Interrupt Handling

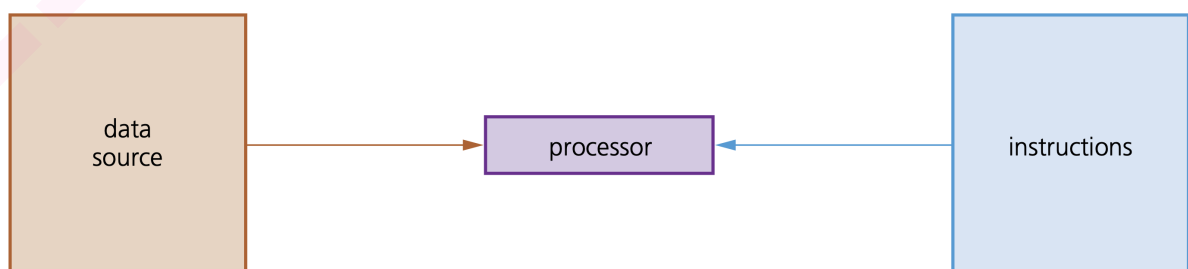
- Both RISC and CISC follow the same interrupt handling sequence:
 - Interrupt Detection:** Recognize an interrupt signal.
 - State Preservation:** Save the current context.
 - ISR Execution:** Uses an ISR via an interrupt vector table.
 - State Restoration:** Restore the saved context and resume execution.
- RISC:**
 - State saving is manual (often via explicit software instructions).
 - May require **pipeline flushing** (clearing the pipeline), increasing overhead.
 - Interrupt latency is more predictable due to fixed-length instructions and simpler pipelines.
- CISC:**
 - State saving is automated by hardware/microcode.
 - Complex pipelines may handle interrupts between micro-operations.
 - Interrupt latency is less predictable.

Parallel Processing

- Operation which allows a process to be split up and for **each part to be executed simultaneously**.

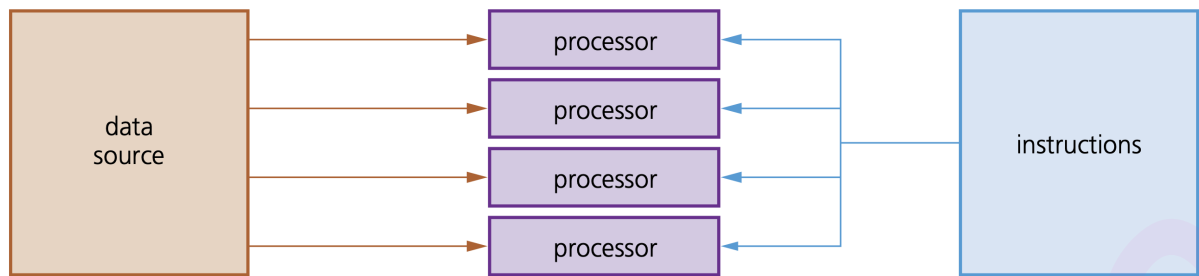
Single Instruction Single Data (SISD)

- One processor, handling one instruction, using one data source at a time.
- Each task is processed in a **sequential order**, disallowing parallel processing.
- There is no pipelining; found in early computers.



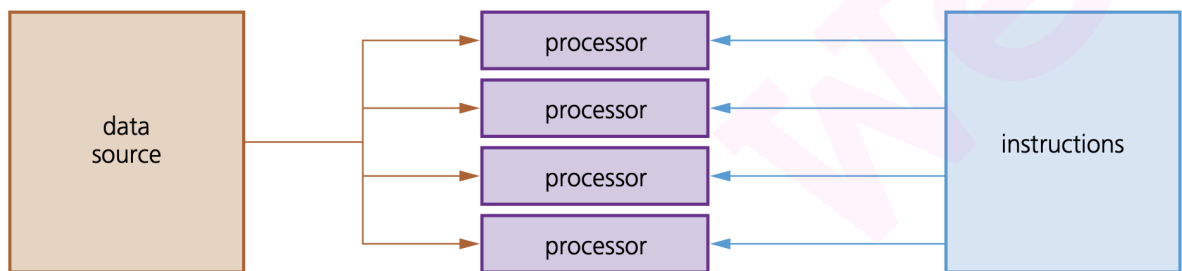
Single Instruction Multiple Data (SIMD)

- Many processors, handling one instruction, using different data sources.
 - Processors do the same calculations but on different data at a time.
- Referred as **array processors**; found in graphics cards.



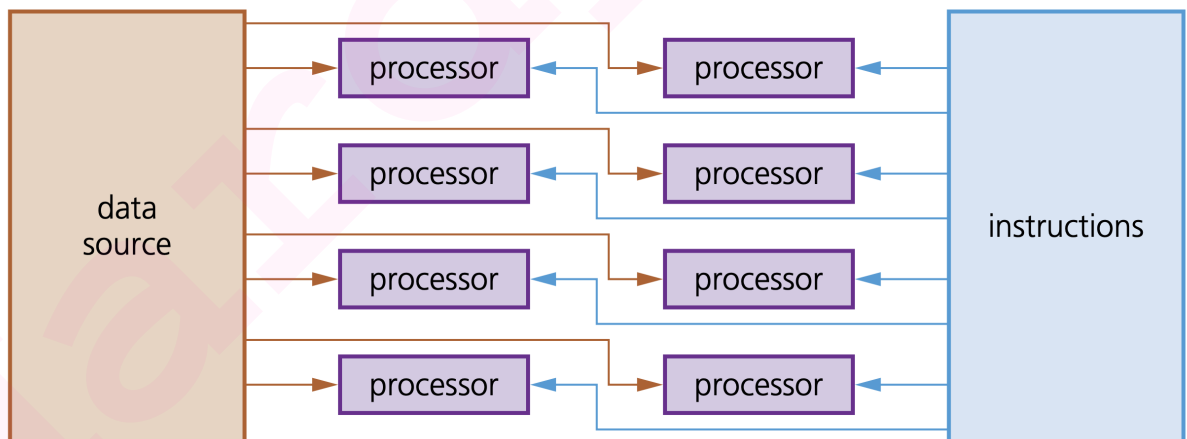
Multiple Instruction Single Data (MISD)

- Many processors, handling many instructions, using the same data source.



Multiple Instruction Multiple Data (MIMD)

- Many processors, handling many instructions, using many data sources.
- **Mostly used** in parallel processing.
- Found in multi-core systems (e.g., multicore chips).



Implementing parallel processing

- **Software-wise:**
 - Split codes into blocks that can be processed **simultaneously**,
 - ...instead of sequentially.
 - Each block is processed by a different processor.
 - Requires both **parallelism** and **coordination**.
- **Hardware-wise:**
 - Communication between different processors might be an issue,

- ...each processor needs a link to every other processor,
- ...which is a challenging topology.

Massively Parallel Computers

- A large number of processors / separate computers connected together,
 - ...simultaneously performing a set of **coordinated** computation.
- There needs to be a **network infrastructure**,
 - ...allowing processors to communicate using a **message interface**.

Virtual Machines

- The emulation of a computer system/hardware and/or software,
 - ...using a **host computer system**.
- Using **guest operating system(s)** for emulation.

Benefits

- Multiple guest operating systems on the same computer, which helps with:
 - Trying incompatible **legacy applications**;
 - Trying **suspicious applications**.
- A virtual machine can crash without affecting the host machine.
- Cost savings due to not needing to purchase extra hardware.
- Different instruction set architectures can be emulated on a single computer.

Drawbacks

- **Extra load on the host computer** makes the virtual machine **less efficient**.
- Complex to maintain, implement, and manage.
- Cannot emulate some hardware.
- A virtual machine may be affected by any weaknesses of the host machine.

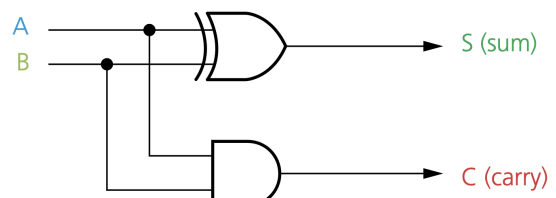
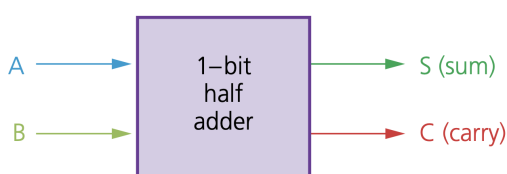
15.2 Boolean Algebra and Logic Circuits

Half Adder and Full Adder Circuits

- Two circuit structures that allow **binary additions**.

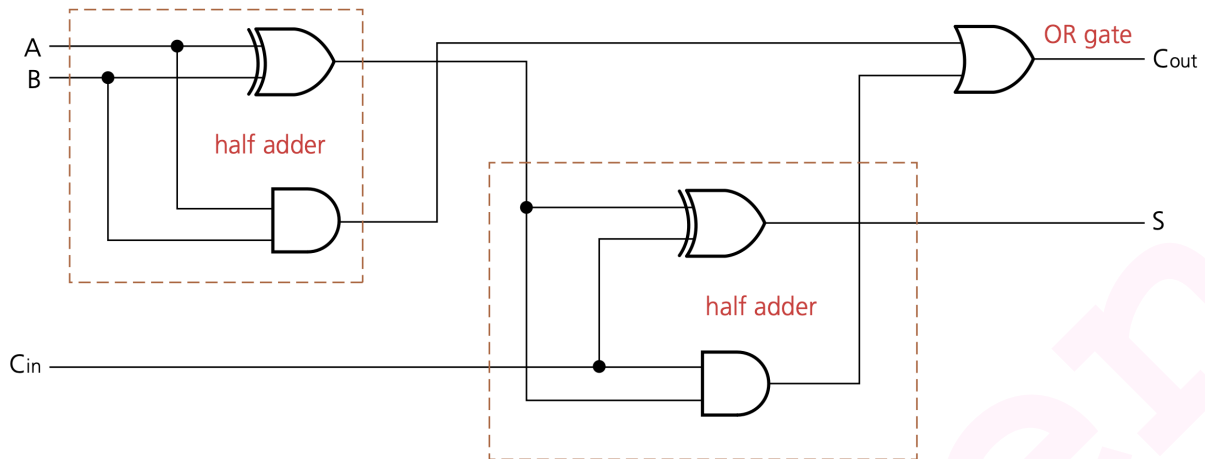
Half Adder Circuit

- A circuit structure that performs **binary addition** on **2 bits**.
 - It generates two output bits.
 - Considering $1 + 1 = 10$, the **carry bit** is "1," and the **sum bit** is "0."



Full Adder Circuit

- Consisting of several half-adders to deal with multiple single-digit inputs (e.g., $1 + 1 + 1$).

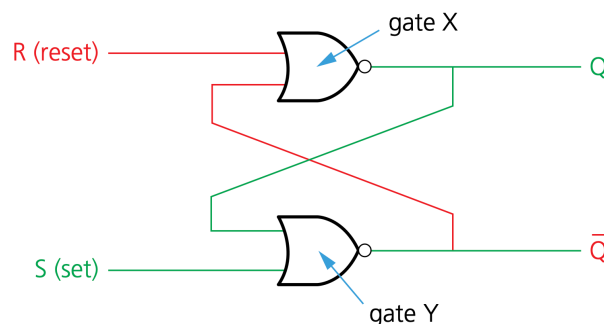


Flip-flop Circuits

- Circuits with two states that are used for data storage to store 1 bit of data.
- Flip-flop circuits are a kind of sequential circuits.
 - Combination circuits:** whose output depends solely on the input values.
 - Sequential circuits:** whose output depends on the previous output.

SR Flip-flops

- Consists of two cross-coupled NAND gates (or NOR gates equivalent).
 - A bit of value can be remembered since the gate goes in loops, and can also be changed through changing the inputs.
- There are four states to be remembered for the diagram below:
 - $S = 1, R = 0 \rightarrow$ The **set state**; sets Q to 1 and \bar{Q} to 0.
 - $S = 0, R = 1 \rightarrow$ The **reset state**; reverts the value of Q and \bar{Q} .
 - $S = 0, R = 0 \rightarrow$ The **hold state**; no effects on outputs.
 - $S = 1, R = 1 \rightarrow$ An **invalid case** because Q shall be a complement to \bar{Q} .
- By altering the position of R and S, one may create different set values, yet the invalid case may change to $S = 0, R = 0$.

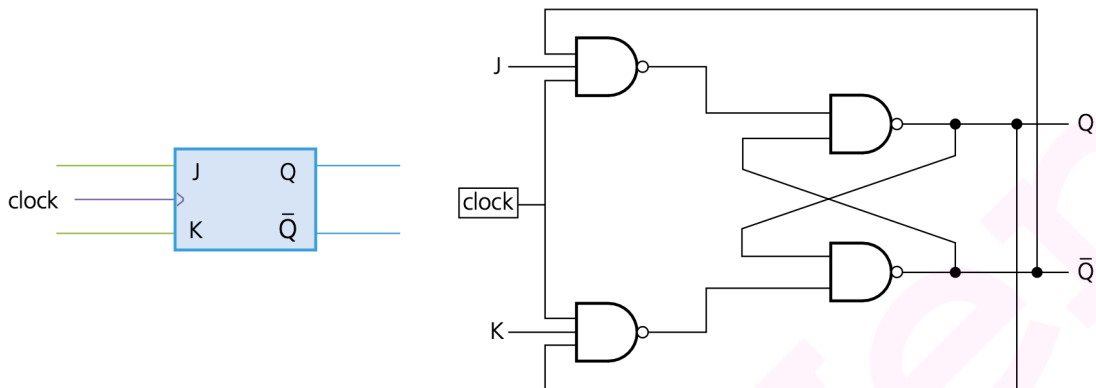


JK Flip-flops

- Designed to solve the problems of SR flip-flops:
 - The existence of invalid S & R conditions that lead to conflicting outputs.

2. **Instability** when the inputs don't arrive at the same time.

- A **clock** and **additional gates** are introduced.
 - The clock synchronizes the input.
 - The additional gates produce a **toggle state** → when an invalid output couple appears, the Q-value toggles after each clock pulse.



Boolean Algebra

Commutative Laws	$A + B = B + A$	$A.B = B.A$
Associative Laws	$A + (B + C) = (A + B) + C$	$A.(B.C) = (A.B).C$
Distributive Laws	$A.(B + C) = (A.B) + (A.C)$ $(A + B).(A + C) = A + B.C$	$A + (B.C) = (A + B).(A + C)$
Tautology/Idempotent Laws	$A.A = A$	$A + A = A$
Tautology/Identity Laws	$1.A = A$	$0 + A = A$
Tautology/Null Laws	$0.A = 0$	$1 + A = 1$
Tautology/Inverse Laws	$A.\bar{A} = 0$	$A + \bar{A} = 1$
Absorption Laws	$A.(A + B) = A$ $A + A.B = A$	$A + (A.B) = A$ $A + \bar{A}.B = A + B$
De Morgan's Laws	$(\bar{A}.B) = \bar{A} + \bar{B}$	$(\bar{A} + B) = \bar{A}.\bar{B}$

Sample Question

- **Simplify** $A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$.
 - Rewrite the expression as: $A.B.C + (\bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C})$
 - This becomes $(A.B.C + \bar{A}.B.C) + (A.B.C + A.\bar{B}.C) + (A.B.C + A.B.\bar{C})$
 - Which gives $B.C.(A + \bar{A}) + A.C.(B + \bar{B}) + A.B.(C + \bar{C})$
 - Hence, we have $B.C. + A.C. + A.B.$

- Two steps to simplify any Boolean equation:
 1. Add new terms to make up distributive terms.
 - Or make use of the **De Morgan's Laws**.
 2. Use the distributive laws to make up offset pairs (e.g., $A + \bar{A}$).

Karnaugh Maps (K-maps)

1. Produce a table that considers different combinations of X_i and \bar{X}_i , where X_i belongs to a set of variables of interest.
2. Group together **cells containing "1"s** in the table.

- Groups must (1) contain even number of 1s, (2) as large as possible, and (3) be a row, a column, or a rectangle.
- Groups may overlap.
- Cells of "1"s that locate at sides can be grouped.

AB \ CD	($\bar{A}.\bar{B}$) 00	($\bar{A}.B$) 01	($A.B$) 11	($A.\bar{B}$) 10
($\bar{C}.\bar{D}$) 00	1	0	0	1
($\bar{C}.D$) 01	0	0	0	0
($C.D$) 11	0	0	0	0
($C.\bar{D}$) 10	1	0	0	1

- Find the variable(s) that change despite all the cells being "1."
- The variable should be simplified in the final Boolean equation.

Sum of products gives:

$$A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$$

We can now produce the following Karnaugh map to represent this truth table (each 1 value in the K-map represents the above sum of products; so there will be four 1-values in the K-map, where A and BC intersect, where \bar{A} and BC intersect, where A and $\bar{B}C$ intersect, and where A and $B\bar{C}$ intersect):

BC \ A	($\bar{B}.\bar{C}$) 00	($\bar{B}.C$) 01	($B.C$) 11	($B.\bar{C}$) 10
(\bar{A}) 0	0	0	1	0
(A) 1	0	1	1	1

A.C B.C A.B

- Green ring: A remains 1, B changes from 0 to 1 and C remains 1 $\Rightarrow A.C$
- Purple ring: A changes from 0 to 1, B remains 1 and C remains 1 $\Rightarrow B.C$
- Red ring: A remains 1, B remains 1 and C changes from 1 to 0 $\Rightarrow A.B$

This gives the simplified Boolean expression: $A.C + B.C + A.B$

- Note that in the diagram above, a group of 2 is selected because there is only one pair of alternating variables (i.e., B and C) in each row or column.
- Hence, a group of 2 is enough for simplifying a variable out.