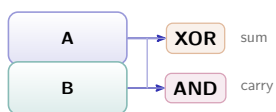


Hardware and Virtual Machines

A-Level Computer Science • Topic 15

A-Level Computer Science



Hardware and Virtual Machines

What we will cover

- 1 RISC vs CISC & pipelining
- 2 Flynn's taxonomy & parallelism
- 3 Virtual machines
- 4 Boolean algebra & K-maps
- 5 Adders & flip-flops
- 6 Recap

1 RISC vs CISC

Hardware and Virtual Machines

└ RISC vs CISC

Two styles of CPU design

CISC

复杂指令集

many, often **complex** instructions
variable length – de-
coding is intricate
many instructions touch memory
more work per instruction
e.g. Intel x86

RISC

精简指令集

few, simple instructions
fixed length – fast to decode
only **load** and **store**
touch memory
each instruction is
quick and predictable
e.g. ARM, RISC-V

Hardware and Virtual Machines

└ RISC vs CISC

The comparison that the exam wants

Feature	CISC	RISC
Instruction set	many	few
Instruction length	variable	fixed
Memory access	many instructions	only load/store
Pipeline-friendly	harder	naturally
Cycles per instruction	varies	usually 1

The trade-off is doing **more per instruction** (CISC) vs doing each one **faster and more predictably** (RISC). Modern Intel chips translate CISC into RISC-like micro-ops.

Hardware and Virtual Machines

└ RISC vs CISC

Pipelining – an assembly line for instructions

A **pipeline** 流水线 processes instructions in **overlapping stages**:

Fetch → Decode → Execute → Memory →
Write back

Once the pipeline is full, **one instruction completes per cycle**. RISC's fixed-length, simple instructions make every stage take the same time.

A pipeline **hazard** 冒险 stalls it: a **data hazard** (result not ready) or a **control hazard** (a branch makes the next address unknown).



2 Parallelism

Flynn's taxonomy

Flynn's taxonomy 弗林分类 sorts computers by instruction streams $\times$ data streams.

SISD

one instruction, one data stream – a traditional single core

SIMD

one instruction on *many* data items – GPUs, vector units

MISD

several operations on the same data – rare, mostly theoretical

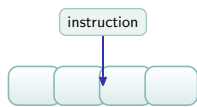
MIMD

many processors, different instructions on different data – multi-core, clusters

SIMD vs MIMD, in pictures

SIMD

one instruction, many data



a graphics card: thousands of cores, same op on many pixels

MIMD

many instructions, many data



multi-core CPUs and clusters

Massively parallel computers

A **massively parallel** 大规模并行 system uses **thousands of processors** on a fast network.

- each has its own memory – **distributed memory** 分布式内存
- they exchange data by messages
- it is MIMD
- needs specially-written software (MPI, CUDA)

Suits climate simulation, large **machine learning** 机器学习 training, astrophysics. The largest **supercomputers** 超级计算机 are massively parallel.

3 Virtual machines

Two kinds of virtual machine

A **virtual machine** 虚拟机 is a **software emulation** of a whole computer – the software inside sees a CPU, memory and disks that look real.

system VM

runs a complete OS
a **hypervisor** 虚拟机监控器 creates and manages VMs
uses: different OSes on one machine; server consolidation;
sandboxing 沙箱; snapshots

process VM

runs one program in portable **bytecode** 字节码
JVM, CLR, CPython
write once, run anywhere;
runtime safety; **JIT** 即时编译
cost: an extra layer

4 Boolean algebra

Laws you must be able to use

Boolean algebra 布尔代数 simplifies expressions. + is OR, · is AND, an overbar is

Law	Form
identity	$A + 0 = A, \quad A \cdot 1 = A$
null	$A + 1 = 1, \quad A \cdot 0 = 0$
idempotent	$A + A = A$
inverse	$A + \bar{A} = 1, \quad A \cdot \bar{A} = 0$
De Morgan 德摩根定律	$(A + B)' = A' \cdot B', \quad (A \cdot B)' = A' + B'$
absorption 吸收律	$A + AB = A$

Example: $Z = AB + A\bar{B} = A(B + \bar{B}) = A$. Fewer terms $\Rightarrow$ fewer gates.

Karnaugh maps – group the 1s

A **Karnaugh map** 卡诺图 simplifies by grouping adjacent 1s.

- rows and columns in **Gray code** 格雷码 (00, 01, 11, 10)
- groups are rectangles whose sides are 1, 2, 4, 8
- the map **wraps around** its edges
- a group of 2 drops one variable; of 4 drops two

1s in $\bar{A}B$ and AB

The two 1s share the $B = 1$ column. A changes, so it disappears:

$$X = B$$

Two gates fewer than $\bar{A}B + AB$.

5 Adders and flip-flops

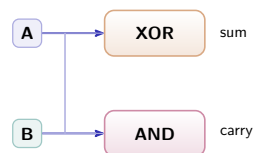
Half adder – two bits, no carry-in

A **half adder** 半加器 adds two bits A and B :

$$S = A \text{ XOR } B, \quad C = A \text{ AND } B$$

It **ignores any carry-in** – hence “half”.

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

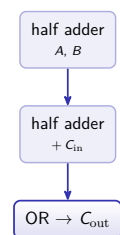


Full adder – three bits, including carry-in

A **full adder** 全加器 adds A , B and a carry-in:

$$S = A \text{ XOR } B \text{ XOR } C_{in}$$

Built from **two half adders plus an OR gate**. Chain them (each carry-out feeds the next carry-in) to make a multi-bit *ripple-carry* adder.



Flip-flops – one bit of memory

A **flip-flop** 触发器 is a **bistable** 双稳态 circuit: two stable states, so it **remembers** one bit. Building block of registers and SRAM.

SR

SR 触发器

S sets $Q = 1$; R resets $Q = 0$
 $S = 0, R = 0$ holds
 $S = 1, R = 1$ is *invalid*
two cross-coupled NOR gates

JK

JK 触发器

$J = 1, K = 1$ is a **toggle** 翻转
ideal for **counters** 计数器
usually *clocked* – inputs act on a clock edge

6

Recap

Topic 15 – key takeaways

1 RISC

simple, fixed-length, pipeline-friendly

2 Flynn

SISD, SIMD, MISD, MIMD

3 Logic

De Morgan, K-maps, half vs full adder

4 Memory

flip-flop stores one bit; VMs emulate a computer

Check yourself

- 1 Why does RISC suit pipelining better than CISC?
- 2 A GPU running one instruction on many pixels – SISD, SIMD or MIMD?
- 3 What extra input does a full adder have that a half adder does not?
- 4 What does $J = 1, K = 1$ do on a JK flip-flop?

Answers 1. fixed-length, simple stages of equal time 2. SIMD 3. carry-in 4. toggle