*ch.1:

$cpu\ time = IC_{total} \times CPI_{avg}$

① classes of computers:

[1] Personal → general purpose, variety of software
↓cost, ↑performance

[2] server → network based, range from small servers to building sized
↑capacity, ↑performance, reliability

[3] super → high-end sc and eng calculations.
↑est capability, but represent small fraction of the overall computer market.

[4] Embedded → hidden as a components of system, run one set of related apps that are normally integrated with hardware
↓power, ↑performance, ↓cost

② The postPC Era:

PMd → personal mobile device [Battery, internet, dollars] → [tablets, smartphones, glasses]

cloud computing → WSC ⇒ warehouse scale computer
→ Saas ⇒ software as a service. [Amazon, google]
→ cloud X & بتشتغل PMD X & run عليه

③ understanding perfs:

① algorithm (# operations) ② PL, compiler, Ar (# machine int) ③ P, memory (Fast insts) ④ I/O (Fast I/O op).

④ 8 great ideas:

① Moore's law (18-24) months
② abstraction (simplify)
③ make the common case fast
④ per via parallelism
⑤ per via pipelining
⑥ per via prediction
⑦ Hierarchy of memories
⑧ dependability via redundancy.
[60 years]

⑤ below my program:

① App software → HLL
② system software → compiler + OS → service c (I/O, memory & tasks, sharing resources)
③ Hardware → P, memory, I/O controllers. (HLL → mc)

⑥ touch screen:

tablets, smartph ⇒ capacitive [multi-touch], resistive [single ·]

⑦ Abstraction:

ISA → software/Hardware
ABI → ISA/system soft | Implementation → details underlying the interface.

	DRAM	MD	Flash
price	expensive	cheap	med
speed	fast	slow	med
volatile	بيفقد ✓	X	X
wearout	X	X	بيخرب ✓

⑧ networks:

① LAN → Ethernet
② WAN → internet [illegible]
③ wireless net → wiFi, bluetooth [illegible]

⑨

execution time → كم ثانية بالظبط الوحدة؟
throughput → كم task خلال فترة زمنية ثابتة؟

⑩

perf = 1/ET.
$cpu\ time = \frac{IC \times CPI}{c.rate} = IC \times CPI \times cc\ time$ (→ clock cycles)

⑪

same ISA, compiler → same IC_i, IC_{Total}
same cpu → same cc time, rate, CPI_i

⑫

$cc_{(total)} = \sum_{i=1}^{n} (CPI_i \times IC_i)$, $CPI_{Avg} = \sum_{i=1}^{n} \left(CPI_i \times \frac{IC_i}{IC_{tot}}\right)$ → r.F

⑬

Best perf → execution time.

⑭

PL, compiler, algorithm → affects IC, CPI.
- ISA → IC, CPI, CC time (cc period)
- microarchitecture design → CPI_{Avg}
- Hardware implementation → tech, clock period.

ch.2:

① RISC ⇒ reduced Instructions set computer.
Fixed length, Fast execution, simple functionally

② CISC ⇒ complex Instruction set computer.
variable length, slow, complex.

③ Design Principles:

[1] simplicity favors regularity — البساطة تفضل الانتظام.
[2] Smaller is faster — للـ registers
[3] good design demands good compromises — للـ Format

④

rs → read from it, rd → write on it

⑤

X_1 → return address, X_2 → SP
X_3 → global P, X_8 → Frame P, X_4 → thread P
X_5-X_7, $X_{28}-X_{31}$ → temp, $X_{10}-X_{11}$ → results/arg
$X_{12}-X_{17}$ → arg, X_9, $X_{18}-X_{27}$ → saved

⑥

ld rd, offset(rs1) → from m → r ⇒ compute, read
sd rs2, offset(rs1) → from r → m ⇒ compute, write

A[12] = h + A[8]; (A[8] → ld; A[12] → sd)
A: X22, h: X21

```
ld  X5, 64(X22)
add X5, X5, X21
sd  X5, 96(X22)
```

⑦

D = 3 - B; (D: X8, B: X9) ⇒

```
sub  X10, X0, X9
addi X8, X10, +3
```

make the common case fast (+3 → 12 Bit)

⑧ Unsigned Binary:

$X = X_0 2^0 + X_1 2^1 + \ldots + X_{n-1} 2^{n-1}$

range: $(0 \to 2^n - 1)$

⑨ signed magnitude:

$-(2^{n-1} - 1) \to (2^{n-1} - 1)$ (range)

⑩ 1's complement:

$-X = \bar{X}$, $X + (-X) = 2^n - 1$

$-X = 2^n - X - 1$

(11) 2's-complement: $(-2^{n-1}) \to (2^{n-1}-1)$

$X = X_0 2^0 + X_1 2^1 + \dots - X_{n-1} 2^{n-1}$

Most negative: 1000……0000

$-X = \bar{X} + 1 = 2^n - X$ $X + (-X) = 2^n$

(12) Formats:

(1) R-Format: add, sub   **add rd, rs1, rs2**

7	5	5	3	5	7
Funct7	rs2	rs1	Funct3	rd	opcode

(2) I-Format: addi, ld, and, or, xor,   **addi rd, rs1, imm**

12	5	3	5	7
imm[11:0]	rs1	Funct3	rd	op

(3) S-Format: sd rs2, offset(rs1)

7	5	5	3	5	7
imm[11:5]	rs2	rs1	Funct3	imm[4:0]	op

(4) shift imm-Format: slli, srli   **slli rd, rs1, imm**

6	6	5	3	5	7
Funct6	imm[5:0]	rs1	Funct3	rd	op

(5) SB-Format: Branch   rs1, rs2, imm   12 Bit

7		5	5	3	5		7
[12]	[10:5]	rs2	rs1	Funct3	[4:1]	[11]	op

Target address = PC + imm × 2 → half word

(6) UJ-Format: jal rd, imm → 20 Bit

[20]	[10:1]	[11]	[19:12]	rd	op

(7) U-Format: lui rd, 0x(5 digits)   → addi rd, rd, 0x(3 digits)

20	5	7
imm[31:12]	rd	op

32	20	12
Sign extension	imm	00…0

$\bar{X}$ ⇒ xori x9, x10, 0xFFF   invert

And → select some bits, clear others to 0

OR → set some bits to 1, leave others unchanged.

(bound) 0 ≤ x < y ⇒ bgeu x20, x11, outOfBound   (x20 ← , → x11)

Calling → jal x1, (proc) (↑ PC+4), return jalr x0, 0(x1)

push → store, pop → load   **beq x0, x0, Exit ≡ jal x0, Exit**

lbu rd, offset(rs1) → sign extension (2 digit, 1 memory)

lhu rd, offset(rs1) (4 digit, 2 memory)

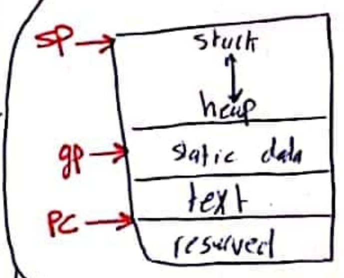
lwu rd, offset(rs1) (8 digit, 4 memory)

(rightmost)

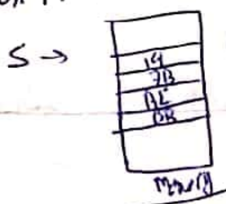
sb rs2, offset(rs1)

sh rs2, offset(rs1)

sw rs2, offset(rs1)



0x FF BC 23 45 14 78 AC DB



$(1000\ 000000)_{10}$

long long int in C → 64 bits

string in C → 16 char + null char ⇒ 128 Bits (1 quad word)

String in java → (2 quad word = 256)

*** Ranges: Max offset for branch: ±2K half word = ±4K Bytes = ±1K instructions.**

Max offset for jal: ±512K half word = 1M bytes = 256K instructions

* jalr: lui x20, 0x CABCD1   jalr x0, 0x 230(x20)   64

* bne x10, x0, L2, jal x0, L (20), L2: ……

lock(): ⇒ addi x12, x0, 1

again: lr.d x10, (x20)

bne x10, x0, again

sc.d x11, (x20), x12

bne x11, x0, again

unlock ⇒ sd x0, 0(x20)

lr.d rd, (rs1)   sc.d rd, (rs1), rs2

swap: again: lr.d x10, (x20)

sc.d x11, (x20), x23

bne x11, x0, again

addi x23, x10, 0