# Comparative Architectures

## Source

- IB Architecture / Computer Design
- IB Compiler Construction
- II Advanced Computer Architecture / Comparative Architectures
- Computer architecture: a quantitative approach
    - Hennessy, J.L. & Patterson, D.A (2011)

## Analogue and digital

- What are the advantages and disadvantages of analogue computers over their digital counterparts?

| | **analogue (oscilloscope)** | **digital computer** |
|---|---|---|
| feature | continuous values / physical data | discrete values / binary system |
| speed | slow | fast |
| memory capacity | low or limited | large |
| reliable / accurate | no (checksum) | yes |
| usage, arch | complicated | easy |
| result | voltage signals | computer screen |
| energy | current -- power-hungry | lower power |
| reprogram | wirable | reconfigurable |
| communication | radio signal (speed) | bus, wire (1/10 speed of light) |

Relationship: Analog = Quantize [0,255] saturated by Max $\Longrightarrow$ Digital

Digital computer system comes from analog and the conversion has a cost.

Digital one has repeatable complex components, Inductor.

## Modern Compiler

Key takeaway from translator (interpreter) shown in Compiler Lecture,

- Divide from single into two stages
    - Compile (inspect)

    - Interpret (compute)
- Divide from single into two stacks (memories)
    - instruction stack / IM

```
PUSH, POP, MK_PAIR
```

    - data value stack / DM
- Separation of the two memories (Instruction and Data)
    - allows for simultaneous access
        - an instruction can be read while a data memory is read or written in the same cycle.
        - Motivation for pipeline and multi-issue *superscalar* (Instruction level parallelism)
        - more difficult with a unified cache/memory.
    - instruction memory is read-only and has less circuitry
        - has no dirty bits, no write back, etc
        - the IM and DM can have different associativity
    - Downside: von Neumann bottleneck
        - common bus (address, data, control)

Aside: Turing tax (universal computing machine) vs special purpose processor.

## Flynn's Taxonomy

Based on parallelism on instruction and data streams, the first four kinds listed below

SISD

- A simple processor

MISD

- Used for redundancy
    - Fligh control system, error-detection

SIMD

- Vector processing
    - Vector registers each hold several data items
        - hardware: Regs = Reg $\times n$
    - Vector operations (add, multiple)
        - hardware: ALU $\times n$
- Energy-efficient, data level parallelism

MIMD

- Multicore, standard general purpose CPUs

*Extra:* SIMT

- each thread has *separate state* (registers and memory)
    - e.g. stack pointer (sp)
- data level parallelism

| **Processor** | | | | | **Note** |
|---|---|---|---|---|---|
| instruction | ---S--- | Fetch | Decode | ---S--- | *Shared* |
| memory | ---S--- | Shared | Memory | ---S--- | *Shared* |
| processing | ALU0 | ... ... | ... ... | ALU31 | *single 32-value vector operation* |
| thread states | Regs 0<br>... ... | Regs 1<br>... ... | Regs 2<br>... ... | Regs 3<br>... ... | *each vector register contain 32 floats* |
| | ... ...<br>Regs 12 | ... ...<br>Regs 13 | ... ...<br>Regs 14 | ... ...<br>Regs 15 | *hide latency when stall* |
| thread context | T0 | T1 | T2 | T3 | only**one** run<br>*save context switch* |

## Architectures comparison

Source: Classifying Instruction Set Architectures (Textbook **A.2**)

| **Architecture** | **Accumulator** | **Stack** | **Register File** |
|---|---|---|---|
| operands:<br>from memory and | acc + 3 = 4 | top of the stack | rs1, rs2 (disjoint), rd<br>orthogonal needs less |
| instruction density | shortest<br>less mem space | concise (short instr) | longer |
| von Neumann bottleneck<br>(Mem bus) | worse for mem (RTT)<br>mem bus 2x CPU<br>⟺ 2x frequency | store imm in stack (near)<br>If stack is full, memory | store<br>in cache (nearer)<br>fast mem access<br>⟺ higher frequency |
| caching | hard to predict | predictable | in the middle |
| power consumption | less<br>few memory accesses | less for control<br>few memory accesses | most<br>multi-issue |
| multi-issue | 0 | 0 | Yes |
| performance | Calculator<br>ENIAC | razer printer,<br>compiler(JVM)<br>Hard for queue, list, swap | modern CPU<br>IC<br>best |

| | **superscalar** | **compiler VLIW** | **SIMD** | **multi-core** | **DSA** |
|---|---|---|---|---|---|
| parallelism | static or dynamic ILP | static ILP | DLP | TLP | custom |
| features | instruction fetch, dynamic prefetch, memory addr. alias, physical regs. rename | scheduling; sw. speculate; var. /fixed bundle | N ops., independent, same FU, disjoint regs., known mem access, | fine/coarse-grained vs. SMT | specialized |
| instr. count (IF/DE) | ↑ out-of-order | one VLI | ↓ | var. | custom |
| branch handling | dynamic branch pred. | limited | poor (predicted) | per-core | custom |
| limitations | fabrication, and below | tailor to a pipeline | data-level tasks | Amdahl's Law | inflexible |
| hardware cost/area | ↑ | ↓ | vector regs. | pipeline regs. | custom |
| interconnect | ↑ (in single core) | ↓ | wide data bus, lane | mesh/cache coherence | scratchpad |
| energy cost | ↑ | ↓ | ↓ | var. | ↓ ✰ |
| binary compatibility | ✓ | × | ▢ (✓ VLA) | ✓ | × |
| use cases | CPU, general-purpose | embedded, GPU | ML, graphics | CPU, server, SoC | TPU, DSP |

## VLIW

Variable-length bundles of independent instructions

| VLIW | fixed-width bundle | var-len bundle |
|---|---|---|
| code density,<br>i-cache size,<br>instr fetch | ↑<br>no-ops | ↓<br>only st., end |
| i-cache hit rate | ↓ | ↑ |
| fixed SLOT-to-FU | ✓ | × |
| hardware | simpler | + decoding;<br>check before issue;<br>+ interconnect,<br>muxes for op,<br>operands to FU. |

## Addressing and cache

Reference: Memory address calculation.