[redacted]

Performance

Metrics:

① Latency: $t_{stop} - t_{start}$ (seconds). A task on a CPU has latency.

- speed/performance = $\frac{1}{latency}$, speedup = $\frac{latency_{before}}{latency_{after}}$. { >1: faster; <1: slower.
- importance: waiting/human vision/due soon

② Bandwidth: work per time; bytes/second. instruction/seconds, Etc.

- latency: $\frac{work}{bandwidth}$. e.g. write 10MB at 10MB/s : 1s.

Key Equations

① Amdahl's Law: optimizations x (generally) uniformly affect the entire program

- $Speedup_{total} = \frac{1}{x_1/s_1 + x_2/s_2 + \cdots 1 - x_1 - x_2 \cdots - x_n}$ x_i: fraction of applications; s_i: speedup.
- corollaries: common $\xrightarrow{optimize}$ uncommon $\xrightarrow{time}$ new common ⇒ { global optimize; divide up dif
  * make common case fast until there is no common

② CPU Performance Equation

- Execution time = IC * CPI × CT = $\overset{CPI}{\frac{cycles}{instrn}}$ × $\overset{IC}{instructions}$ × $\overset{CT}{\frac{seconds}{cycles}}$
- CPI: average cycles, CT = $\frac{1}{frequency}$ (G: 10^9, M: 10^6), IC: dynamic instructions

A) Reduce Instruction Count: Algorithm/Compiler

{ O0: no optimization, variable on stacks. lots of loads/stores
O1: x loads/stores, fewer movs, nothing on stack

{ common sub elimination; constant propagation; function in-line

- inline → constant → loop unroll → partial evaluation (substitute computation)
- Impacts: program input, instruction set

B) Improve CPI: $\frac{total\ cycles}{total\ instructions}$ = average latency of an instruction

{ Program: float/memory, larger CPI
Inputs: exercise different parts of the program
Compile: which instruction. e.g. O0 Vs O1

c) Affect CT: processor design (Cycle $time_{min}$ = delay max along critical path)
Manufacturing variation (transistor chars). software policy

③ Power and Energy Consumption

A) Power = $\frac{Energy}{Time}$ = J/s = Watt; $P = V^2FaC + P_{idle}$ (voltage, activity factor, frequency, capacitance (area), idle power consumption)

- P increases with area(C) and switching factor (a)
- Pidle proportional to the area of the chip
- V and F are linearly related. $P \sim F^3aC + P_{idle}$: F↑, Power↑

B) Energy = Power × Time = $(V^2FaC + P_{idle}) \times (CPI \times IC \times CT)$

- Clock freq ↑ : E↑, P↑, ET↓

Benchmark well-defined computation used to quantify system characteristics

types { Micro: measure specific aspect; Macro: stand app using standard inputs; PoC: run a user-specific program

• characteristics { Realistic, portable, easy to use; own datasets, useful meas; capture compiler/architecture

• Spec2017: industry consortium (vetos), open-source, all in C/C++/Perl

x86 Assembly

ISA

Hardware - ISA { instruction format; instruction operation → software; interaction with memory

16	32	64	
ax	eax	rax	accumulator
bx	ebx	rbx	base
cx	ecx	rcx	counter

b	s	w	l	q	
8	16	16	32	64	bits

x86

- instruction src1 src2 dst;
- %<reg>: register, $imm: immediate, $label: label. m(%eax,%ebx,n) = Mem[m+(R[eax]+R[ebx]×n)]
- %eax: reg value; (%eax): Mem(R[eax]); n(%eax): Mem(R+n);
- compare: cmpl %eax %ebx : %eax-%ebx; e.g. cmp %eax, %ebx / jge label ⇒ %ebx >= %eax
  - ZF: 1 if s1 and s2 are equal; SF: 1 if s2 < s1.
- Function call: ret: return address; leave: restore call stack pointer variable { inst length

Types { CISC: complex to compiler, human readable, continue to add features [x86, VAX] (interface ✓, implementation ✗)
RISC: compiler outputable. [Alpha, MIPS, SPARC]. fixed: inst. length, behavior (interface ✗, implementation ✓)

- x86 → RISC-like instructions → execution

Efficient Execution

Instruction Execution: fetch → Decode → RF read → { Arithe; mem; branch } → next PC { single inst → 6 cycle; CPI↑, energy↑

Pipeline use all stages at a time cycles for an instrn remain constant

- limitation: IC [ISA, compiler], CPI=1 [Best case], CT [critical logic path in slowest stage]

① Control hazard: fetch stage x know next

- Problem: Perform ~ CPI=6 by stalling, Energy ~ a is small $\frac{CPI_n \times IC_n + CPI_m \times IC_m}{IC_n + IC_m}$

⇒ Speculative execution: predict on prior, detect misprediction

E.g. 1-bit local predictor $T[A\%2^N] == 1$ predict T else NT; $T[A\%2^N]$ = taken? 1:0

- Misspeculation: CPI↑, pipeline flushing
  - IC determined by non-speculative, compiler/ISA; CPI: cycles wasted Pidle
  - Energy: a=1 since all are busy. CPI↑ wasted energy; Non-speculative worse

② Data hazard: Forward, connections bypass the reg file

③ x86 terrible: micro-op, decouple instructions for flexibility. average 1.2 uops/x86
Decode decompose instruction into uops → decode queue

More performance

① Deeper pipeline: Fetch [|] → Decode [|], then CT↓ 50%, CPI↑ { cost of misprediction; Data x resolve by forwarding
x perfect pipeline adds overlay, ↑ space & energy

- Impact. ET↓ (CT↓50%), Energy (aC↑ extra area, switching). Power (F doubles) more than double

② Wider pipeline: two wide (rstr. data dependence)

- Impact: ET↓ some (CPI↓50%). Energy (aC↑ bigger reg file)↑ a bit
- wider: more hazards, idle units, overhead: long-latency complications

③ Out-of-Order Execution:

A) Data Dependency - instruction reads correct values/order

- { RAW; WAW, WAR } false dependency: no data flows but A must execute before B
- critical path: longest sequence of RAW, WAW, and/or WAR-dependent instr
- formulas: $CPI = \frac{CP}{\#instructions}$, average $ILP = \frac{1}{CPI}$, $IC = ILP * CP = \frac{1}{CPI} \times CP$

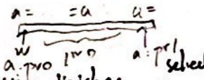
$$ET = IC \times CPI \times CT = IC \times \frac{CP}{IC} \times CT = CP \times CT$$

- effective latency: its contribution to CP only RAW constrain parallelism

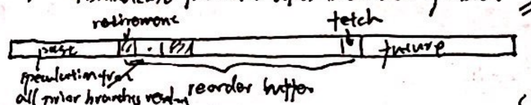
B) Register renaming: RAT ~ Architecture → Physical registers.

- ILP: examine larger region of the instruction stream ins = log(pr) × ar

C) Out of order issue [Tomasulo's Algo]

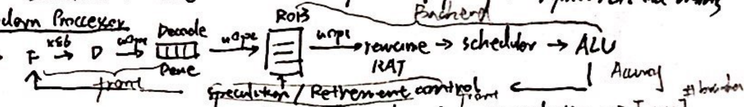


- Inputs { physical reg; broadcast from ALU after the instrn finishes scheduler ⇒ ALU



- Scheduler: issue instructions to ports [pipelined]. { reg value available; ✓ free, compatible port; prioritize old instrs

Modern Processor



- smaller uops retired each cycle → branch misprediction. = [acc] #branches
- Branch prediction ↑, ROB utilization ↑

Cache

MB = 1024 × 1024 = 2^{20}

formulas { # block offset bits = $\log_2$(block-size) bytes
# entries = $\frac{cache\ size\ (KB,\ 1024\ bytes)}{blocksizes \times n}$ tag = address − # index − # tag
index = $\log_2$(# entries) * block size: data transferred on miss

tag | index | offset , block size ↑ ⇒ miss↓, # blocks↓

localities: temporal ~ near in time, access same very soon; spatial ~ near in space, next access is close to last access.

cache misses:

{ compulsory: first access to the data
capacity: address requested AND fully-associative cache of same size exp a miss (cache isn't big enough)
conflict: indexing caused collisions. addr AND full-ass cache of sa exp a hit

Set Associative index → set [block1 / block2]

store policies notify lower cache the change when evicted

- miss: write allocate (bring cache into cache), write no-allocate (notify lower cache)
- hit: write through (tell lower level cache that was changed), write back: mark as dirty

Prefetching

H: Stream buffer { stride: difference; confidence • impact: { idle bandwidth delay other requests; energy: → lowe↑ energy

S: PREFETCH: insert your own preferences.

Locality

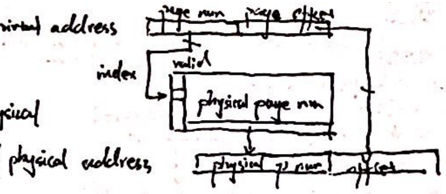
High locality: miss (write allocate), hit (write back)

low locality: miss (write no-allocate), hit (write through)

Levels

L1: write-through, write allocate; L2 write back and allocate (limited on store hit); L3/Memory ~ write through

Virtual Memory



Address translation

- map from virtual to physical
- cache virtual in memory

Translation look-aside

- cache for address translation
- Access : TLB miss , TLB exception [OS] , TLB fill . page fault [OS replace page table]
  TLB fill (valid) , L1 M , L2M , L3M , L3 fill , L2 fill , L1 fill .

Memory Level Parallelism

critical path : long latency instructions influence critial page ⇒ interdependency?

modern superscalar :

- Non-blocking cache   limit { finite load / finite MSHR / ROB , physical register
- Memory Status History Register : track outstanding misses

prefetcher : predictable access

{ finite resources ⇒ only a few accesses ahead
  idle bandwidth .

Multi-core

Unicore MV decreasing ; performance O(√n) , Power O(n) ; lack of ILP .

parallel architecture

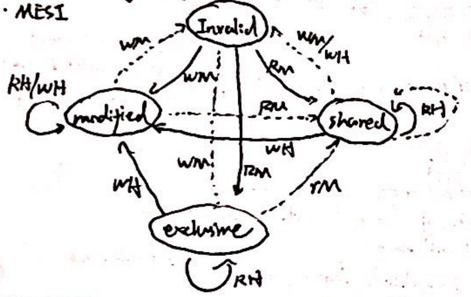
{ multiprocess : multiple CPUs tightly coupled enough to cooperate on single problem
  multithread : single CPU core that can execute multiple threads simultaneously
  multicore : CPU cores coexist on a single processor chip .
  Multi-socket : two or more share same system (RAM , Disk , Ethernet)

Policy

- Inclusive : everything in L1 must also be in lower level of caches
- LRU : different for different levels of caches ; L3 has few information
- LFU : # times sth is used ; LFUDA lowers counters periodically so old .

Cache Coherency

- shared memory : synchronization + mutex
- MESI



coherency miss : miss on sharing data

Floating Point

- Format

~~32-bit.~~

	sign	mantissa	exponent
32bit	~~32-b~~	24 bit	8 bit
64 bit		53 bit	11 bit

$(-1)^s \times (\frac{m}{2^{24}} + 1) \times 2^{(e-127)}$

- FP in Hardware is much faster .
- Dedicated FP physical register file , large ALU , [Silicon Area vs Instruction-fre]
  fully piped : CPI = 1 / # ports
- Methods : unrolling ⇒ ILP ; tiling ⇒ MLP

Vector

- Single Instruction Multiple Data
- Impact : static instructions ↑ dynamic ↓ , uops ↓ , Energy ↓ , Power ↑
- Compiler / language / Intrinstics

SMT

fetch instructions from different thread context [PC , reg file]

Mechanism

- scheduler / ALU process Uops ,
  two instruction : fetch , ROB , each thread RAT , load/store queue track
- small overhead

Security

- Meltdown : reveal any memory mapped to that address space
  - rely : aggressive speculation that causes exception
    aggressive memory permission check in the TLB .
    memory mapping optimizations , high resolution timer .
- Spectre : reveal any memory in the kernel
  aggressive speculation past branches .
  careful preparation of cache
  out of order execution
  high resolution timer .