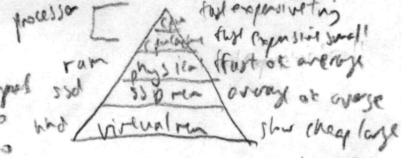
decimals into binary
.xx ×2 · <1 → 0
>1 → 1
=1 = 1 stop

registers
$v0, $v1 = return
$a0 … = arguments) caller
$ra = where are you
sw to memory, load into reg.

1 - registers - head
2 - on chip cache - room
10 - on-board cache - campus
100 - memory - Sacramento
10^6 - disk - pluto
10^9 - tape / optical robot - andromeda



LRU = least recently used

kibi / mibi

malloc → (int *) malloc(n × size of (x))

Victoria Lo
CS 61C

bitwise
& mask/and - all 0s unless both 1s
| set/or - all 1s unless both 0s
^ flip/xor - mismatch = 1, 0
~ flip - flip all

memory is unlimited, cache limited, so % by size of cache (rows)

Caches
- direct mapped - conflict check - unique in bits
- fully associative - garbage bag
- set associative - lost and found ; anywhere in its area

tag → Index → offset   tag bits + valid bits + dirty bits + (block size × $\frac{8 \text{ bits}}{\text{byte}}$)

3 types of misses: Compulsory - not there / not matching, Conflict - collision / cache miss w/ replacement, Capacity - space

shamt = shift, rs/rt/rd → # of bytes eg 2^{6}

& = address
* = points to value itself

1/8/23
floating pt - actual + bias → stored
zero: exp 0, fraction 0
denorm: exp 0, fraction != 0
∞: exp 1's fraction 0
NaN: exp 1 fraction != 0

one's complement = reverse
two's complement = reverse + 1

endotstring = \0
increment pointer → +4

R register - opcode = 0
I immediate - bne / beq
J - j / jal, opcode 2 or 3

Average memory Access time AMAT = Hit time + (Miss penalty × miss rate)

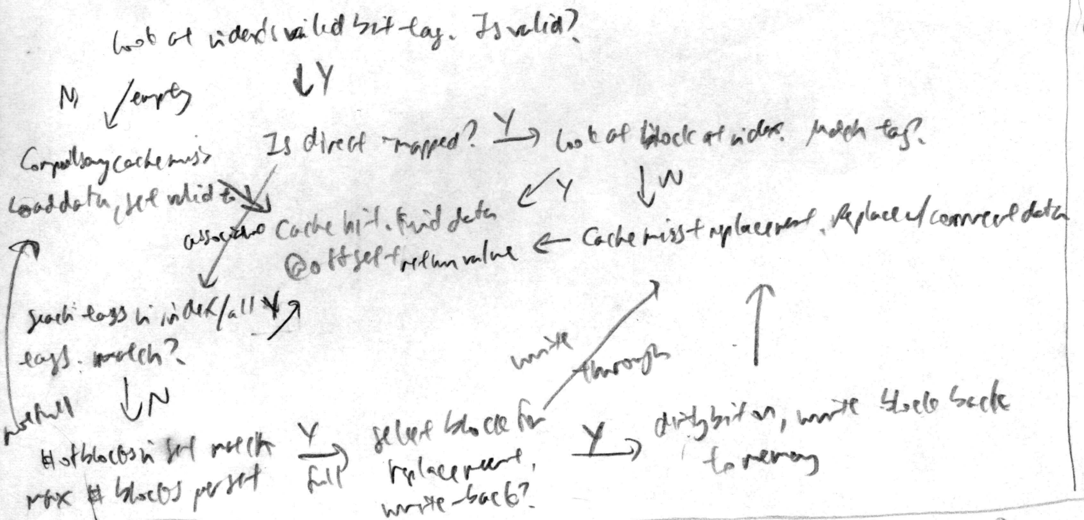
# of offset bits = $\log_2$ (block size)
# of of index bits = $\log_2$ (# of blocks)
cache size = $2^{\text{offset}} \times 2^{\text{index bits}}$
blocks = cache size ÷ block size
tag bits = total - offset - index
row bits = tag + data + dirty + valid

$Hz = \frac{1}{s}$ $1 MHz = \frac{1}{10^6 s}$; $1 GHz = \frac{1}{10^9} = \frac{1}{ns}$, $ps = 10^{-12}$

PUE
power usage effectiveness =
$\frac{\text{total building power}}{\text{IT equipment power}}$

Amdahl's law:
max speed from parallelism =
$(1 - P + (P/N))^{-1}$ where
P = proportion of program parallelizable
N = number of cores

and D F = AB
or D F = A+B
not D∘ F = $\overline{A}$
nand D∘ F = $\overline{(AB)}$
nor D∘ F = $\overline{(A+B)}$



Caches

Cache size: $2^{(\text{index bits} + \text{offset bits})}$

Temporal locality = use [illegible] element again

spatial locality - memory reads sequential / close

dirty bit for write back, valid bit for tag

direct mapped: each memory location associated w/ one location -

N-way set associative - [illegible] but sets have N blocks. 2-way set / performance, LRU - replace oldest

fully associative - blocks anywhere

offset = # of bytes in width (4 words → 2 5 bits)

index = # of bytes in # of rows (32 rows → 5 bits)

tag = 32 - 2 - 5 = 25

write through - write to cache / memory

write back - write to memory on flush, use dirty bit

bits: tag + valid + (dirty) + data (bits)

Misses:

Compulsory - cache empty

Conflict - 2 blocks to same spot, swap out

capacity - cache is too small

Powers of 2

prefixes: kibi (10), mebi (20), gibi (30), tebi (40), pebi (50), exbi (60), zebi (70), yobi (80)

$2^{23} = 2^3$ mebi = 8 mb

MIPS

32 registers, 32 bits for data / instructions

R instruction: 0 opcode, 2^6 funct codes, 61 regular, 3 for srl, sll, sra

I instruction: 61 opcodes, mirror of R, branch limits: PC 2^{17} + PC + 2^{17} - 4 ::

J instruction: 2 opcodes (j, jal), targets PC[32:29] target << 2, limits (2^{28} b/c 26 for [illegible])

beq: 2^{16} b/c 16 bits

declarations: text = machine code, data = binary rep of data, word = 32-bit quantities in order

memory - code at bottom, then static, then heap (grows up): stack at top, grows down

RTL = reg trans lang = "madd rd, rs, rt" - R[rd] = Mem[R[rs] + R[rt]]; PC ← PC + 4

- sll by 2 when [illegible] counters to pointers

prologue + epilogue - [illegible] to + all non-temp vars to stack, then restore

BNF has range 16 + 2, since last 2 bits are 0s. Also, 2s complement

Running a program (CALL)

Compile - high → low, C → MIPS

Assemble - outputs object code into tables, replaces pseudoinstr.

Link - input: object file header, text, data, reloc info, symbol table, [illegible]

Load - allocate space, copy code + vars, init regs, set pointer

Datapath

5 stages - IF = fetch, ID = decode, EX = execute (run ref = calc address, arithmetic, operations), run load + store data, WB = write data

3 hazards

1) structural - prevent [illegible] instruction (lw/sw)
2) data: need result for previous to complete
3) control - deciding dependent on previous (a lw) (jumps, branches)

bubble - [illegible] in nop.

flush - branch prediction checks [illegible].

pipelining - split a task → stages (latency = all stages, throughput = one step, single cycle stretch longest step)

Map reduce

map = take in data, emits intermediate (key, values)

Combiner - combines values of map into lists, w/o ss[illegible] in reduce

reduce - takes (key, value[ ]) + simplifies to (key, result), output

Bit Twiddling:

And: 0x0001 & = only last 4

Or: 0x0001 or = turn last bit on

GPU

throughput optimized, slow cores, arrays, load [illegible],

3D blocks + grid, checkerboards

Virtual memory

page size 4 KB

pages same size saved to disk in swap

TLB - N-ent entries - set / fully ass

offset = log page size

page table - holds all [illegible] of VAS - mapping / disk, depends VM

get addr, ignore offset, look at index, check TLB, page table, memory → page table, else page fault + disk, use offset to pull byte

VA = VPN + page offset. PA = PPN + page offset, same p.o. b/c page size

log(page size) = page offset bits, rest PPN / VPN

page table - VPN, valid, dirty, permissions, PPN, index = VPN

TLB = MRU VPN, 128 entries, set / fully ass

page table = # phys pages valid entries

I/O

polling - $\frac{\frac{\text{poll}}{s} \times \frac{\text{clocks}}{\text{poll}}}{\text{total clocks}}$

polling disk - $\frac{\text{mb}}{s} \rightarrow \frac{\text{poll}}{s}$

poll / s, clocks / poll

$\frac{\frac{\text{poll}}{s} \times \frac{\text{clocks}}{\text{poll}}}{\frac{\text{total clocks}}{s}}$

interrupts - proceed as normal, then interrupt

% active $\times \left(\frac{\text{mb}}{s}\right)$ / $\frac{\text{bytes}}{\text{interrupt}}$ = $\frac{\text{interrupts}}{\text{sec}} \times \frac{\text{clocks}}{\text{interrupt}}$ / $\frac{\text{total clocks}}{\text{sec}}$