吴昊- Haoyi Wu.

L1. 6 Great Ideas:

① Abstraction (Layers of Representation / Interpretation)
② Moore's Law (Designing through trends).
③ Principle of Locality (Memory Hierarchy)
④ Parallelism
⑤ Performance Measurement & Improvement.
⑥ Dependability via Redundancy.

L2. 1. C Pre-Processor (CPP)

* Macro
# transform parameter into strings
## connect 2 tokens into 1 token

foo.c $\xrightarrow{CPP}$ foo.i → compiler (expr ##_num)

2. enum color { RED, GREEN, BLUE }; enum color c; c = RED; // c=1;

3. FALSE: ① 0 ② NULL ③ No explicit Boolean type.

4. one element past end of array must be a valid addr.

L4. 1. ISA, Instruction Set Architecture.
RISC, Reduced Instruction Set Computing.

2. no-op: add x0 x0 x0.
standard no-op improves disassembler and.
potentially improves the processor

3. call func jal label, jal rd offset,
jal offset = jal x1 offset

j offset = jal x0 offset.

jalr rd rs offset. sw: M[R[rs1]+imm] = R[rs2]

ret; jr ra jr rs = jalr x0, rs, 0

L5. 1. 2 storage classes. automatic and static.
local exist across exits from and entries to procedures

procedure frame / activation record: segment of stack with saved registers and local variables.

L6. 1. keep sp the lowest addr. used: Interrupts may use the stack; arguments are in the frame of the caller.

2. Stored-program computer, consequence
#1: Everything Addressed
#2: Binary Compatibility.
PC: Program Counter - Instruction Counter.

3. rs: source register
rd: destination register.

4. branch: reach $\pm 2^{10} \times 32$ bit instructions. on either side of PC.

5. lui edge case, if last 12 bit is negative, add 1 to lui imm, then addi.

6. J format: $\pm 2^{18}$ 32-bit instructions.

* C优先级:

→	后缀	()[]->, ++ --	→	位与	&
←	一元	+ - ! ~ ++ -- (type)* & sizeof	→	位异或	^ 01->1 11->0
→	乘除	* / %	→	位或	|
→	加减	+ -	→	逻辑与	&&
→	移位	<< >>	→	逻辑或	||
→	关系	< <= > >=	←	条件	?:
→	相等	== !=	←	赋值	= ...

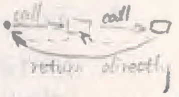
L7 1. All I type do sign extension include sltiu.

2. Interpreter: easier to write, better error message, slower, code smaller, instruction set independence (portable).
Translation, more efficient, higher performance, hide source from user.

3. CALL: C program: foo.c → Compiler
→ Assembly program: foo.s → Assembler
→ Object (mach lang module) foo.o
lib.o → Linker → Executable (mach lang pgm): a.out
→ Loader → Memory.

4. Compiler: output may have pseudo-instructions
Lexer, Parser, Semantic Analysis & Optimization
Code generation.

5. Assembler: reads and uses directives.
1° inst & labels have addr. .text; .data; .globl sym: declare sym global & can be referenced from other files;
2° Create 2 tables .asciiz str, store str in memory &
3° Gen obj file. null-terminate it; .word w1 ... wn

Tail Call Optimization 

Compress code.
Forward Reference Problem: 2 pass over the pg.
Symbol Table: list of labels and static data that can be referenced.
Relocation Table: identifies lines of code that need to be fixed up later. (deal relocate)

6. Linker. 4-type of addr. + absolute addr.
PC-Relative Adressing PIC. position independent code
External Function Reference } always relocate
Static Data Reference
knows: length and ordering of text and data segment
calculates, absolute addr. of each label to be jumped to and each piece of data being referenced
Output: executable with text and data + header.

7. Loader. OS.
read header → create space → cp. inst. & data
→ cp. arg. on stack → init reg. → start up routine & set PC.

8. Areas of memory
static data, stack, heap.

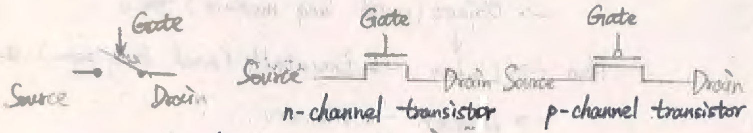
* Assembler generates machine language code.
only allows generation of TAL (True Assembly Language) no pseudo inst
Linker only allows generation of binary machine code

Static
DLL: Dynamically linked libraries .dll, .so

L8. 1. Synchronous Digital System (SDS)
central clock High voltage true 1
Low false 0

2. Transistors.
High voltage (Vdd) means 1 modern processor Vdd ~ 1.0 volt
midpoint to decide 0 or 1. Higher Vdd, higher clk frequency

3. CMOS Transistors.
MOS: Metal-Oxide on Semiconductor
C: complementary
use pairs of normally-on and normally-off switches ✓



Gate is low 0 off on
high 1 on off.

MOSFET: metal-oxide-semiconductor field-effect transistors. FET: CMOS circuits use a combination of p-type and n-type.

4. Moore's Law: 2x Transistors/chip every 2 years.

5. N-type (negative) pass weak 1's (Vdd-Vth)
strong 0's (Ground) ✓
P-type (positive) pass weak 0's (Vth)
strong 1's (Vdd) ✓

6. $\bar{B}C + B\bar{C} = B \oplus C$. Truth table ⇸ Gate diagram

$X + YZ = (X+Y)(X+Z)$

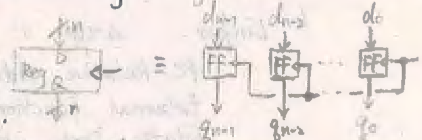
$XY + X = X$, $(X+Y)X = X$

$\bar{X}Y + X = X + Y$, $(\bar{X}+Y)X = XY$

$\overline{XY} = \bar{X} + \bar{Y}$, $\overline{X+Y} = \bar{X}\,\bar{Y}$ DeMorgan's Law

7. SDS { CL: Combinational Logic ALU
SL: Sequential Logic Register (state elements)

8. Register,
n istances of
"D-type flip-flop".

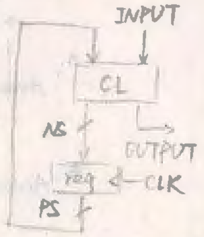


(write before read in 1 clk)
* propagation delay. 传输延迟
on LOAD = sample on rising edge of CLK
(positive edge triggered).

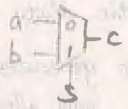
L9. 1. Hold Time violations
Clk → Q + best case combinational delay < Hold Time
Sol. Add delay (2 inverters)

FSM — I/O
input / output

2. PS, present state;
NS, next state.



Finite State Machines
3. mux.
$c = \bar{s}a + sb$



4. adder.

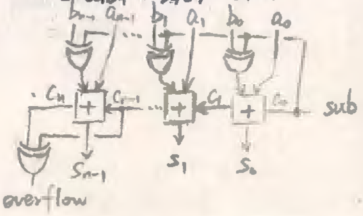
$$\begin{array}{ccccc} & c_3 & c_2 & c_1 & \\ & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & S_3 & S_2 & S_1 & S_0 \end{array}$$

xor serves as conditional inverter.

$S_i = XOR(a_i, b_i, c_i)$ 1 or 3 ones output 1

$C_{i+1} = MAJ(a_i, b_i, c_i)$

$= a_ib_i + b_ic_i + c_ia_i$



L10. 1. break into stages:
- smaller stages are easier to design.
- easy to optimize (change) one stage without touching others (modularity).

2. 5 stages:
fetch inst, PC+4 IF: Instruction Fetch
opcode, read reg, immgen ID: Instruction Decode
ALU EX: Execute (ALU: Arithmetic-Logic Unit)
MEM: Memory Access
WB: Register Write.

3. register, write enable: Negated 0, Asserted 1

4. Implementing:
Imm: Itype: -2048 ~ 2047
Btype: -4096 ~ 4094
in 2-byte increments
UJtype (jal): $-2^{19} \sim 2^{19}-1$
2-bytes apart
± 2^{18} 32-bit instructions.
Utype: $0 \sim 2^{20}-1$ (upper bit)

L11. 1. ROM: Read-Only Memory.
Combinatorial Logic truth table → gates.

2. Control Block Design.
11-bit addr. inputs: Inst[30, 14:12, 6:2], BrEq, BrLT
AND Logic. (with xor, unused) - Address Decoder
OR Logic. for output.

3. RTL: Register Transfer Level.
RISC-V Green Card, Verilog
HDL, Hardware Description Languages.
e.g. ABEL, VERILOG, VHDL
Advantage: Documentation; Flexibility; Portability; one language for modeling simulation, synthesis; productivity.
However, different way engineers not used to.
Use HDL to create
VLSI (Very Large Scale Integration).
ASIC (Application-specific integrated circuit).
a program for FPGA (Field-programmable gate array).

4. A higher clock frequency will improve throughput ↑ and latency ↓
Pipelining: Higher throughput ↑ good, Higher latency ↑ bad

* Adder overflow, carry into MSB ≠ Carry out MSB.

*

type	m32	m64	
size_t	4	8	NULL, long int, void*.

Padding is based on the largest element.

* ASCII standard character from 0 to 127.

```
* .data
    str: .string "Hello world!"
  .text
    la a1, str
 ecall     a0
 print_int     1    int in a1
 print_string  4    null-terminated str in a1
 exit          10
 print_char    11   ASCII char in a1
```

*	
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

吴昊一 Haoyi Wu.

L12. 1. Pipelining Hazards

1) Structural hazard

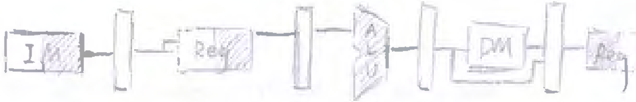
a required resource is busy, needed in multiple stages.

Sol. add more

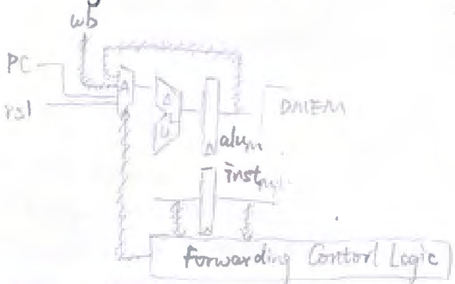
- Regfile, 2 independent read ports and 1 independent write ports.
- Memory Access, Data M & Inst M. Cache.

RISC ISAs designed to avoid structural hazards. at most 1 memory access / instruction,

2) Data hazard: Register Access.



① ALU res Sol. Forwarding. = Bypassing.



② Load Data Hazard.

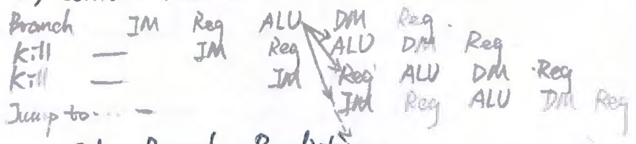
Slot after a load, load delay slot.

If next inst use load res, stall.

Sol. put unrelated inst into load delay slot.

3) Control hazard.

cost 2



Sol. Branch Prediction.

L13. 1. ILP: Instruction-Level Parallelism.

to issue 发出

- Multiple issue "superscalar". CPI (Cycle per Instruction) < 1. use IPC.
- "Out-of-Order" execution.
- Hyper-threading.

2. Hyper-threading

Duplicate all registers, same CL blocks,

3. Superscalar: higher throughput.

- Iron Law of Processor Performance.

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \underbrace{\frac{\text{Cycles}}{\text{Instruction}}}_{\text{CPI}} \times \frac{\text{Time}}{\text{Cycle}}.$$

Or calculate CPI:

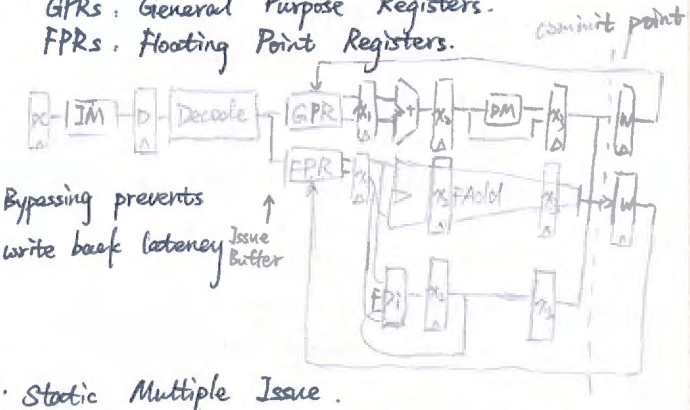
$$\text{CPI} = \sum_{\text{inst}} \text{freq}(\text{inst}) \times \text{CPI of}(\text{inst}).$$

4. Complex pipelines

Issue: Assign instruction to functional unit.

GPRs: General Purpose Registers.

FPRs: Floating Point Registers.



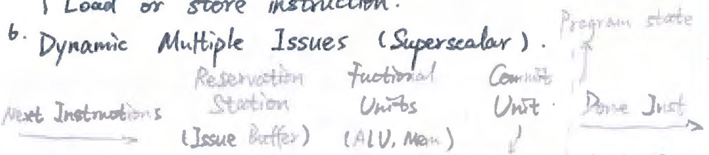
Bypassing prevents write back latency

5. Static Multiple Issue.

= Very Long Instruction World (VLIW).

{ ALU or branch instruction
Load or store instruction.

6. Dynamic Multiple Issues (Superscalar).



Commit unit (Reorder Buffer) (ROB) = result buffer

OoO Fetch - Decode/Rename - Execute - Commit (graduction)

OoO does NOT need speculation

OoO: Out-of-Order; speculative 推测的 → Instruction scheduling

InO: In-Order.

Same: Dispatch (Issue) (put inst into machine) always in-order

Inst fetch / decode / rename always in-order.

7. Data-in-ROB Design.

v. i | Opcode | p. Tag. Src1 | ~ | p. Reg. Res. Exp?

p = 1 use data from architectural reg.

Tag: index in ROB

v: valid bit, set on dispatch; issue bit i. set on issue

complete → set p bit.

8. Renaming.

resolve fake reg. dependencies.

CPU can have more physical reg.s than ISA.

When write of same reg. commits we can reuse it.

L14. 1. Locality:

{ Temporal Locality
Spatial Locality

Principle of Locality: Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality).

Inst: PC+4; loop iteration.

Stack: subroutine call / return.

Data: vector / scalar (向量、常数)

2. Cache gives illusion of speed of fastest memory with the size of largest memory. (cheapest)

principle locality + memory hierarchy

3. Block size (offset) = $\log_2$ (# of bytes per block)

# of cache lines (blocks) = cache size / block size in bytes

# of sets = # of cache lines / # of ways

Size of index = $\log_2$ (# of sets)

Size of tag = addr. size - size of index - size of block.

Tag | Set Index | Block offset

4. valid bit - program start, all invalid (0).

5. Fully Associative (no index) more associativity (ways) → ×2/bit

Direct Mapped   Tag | Index | Block offset

N-way Set Associative. fixed-size cache

6. Total cache capacity =

Associativity × # of sets × block-size

Bytes = blocks/set × sets × Bytes/block

C = N × S × B

L15. 1. Handling store with write.

1) Write-Through Policy.

write → cache $\xrightarrow{\text{write to}}$ memory

Include Write Buffer, update mem parallel to processor. If store miss, load cache line.

1. Simpler control logic
2. Easier to make reliable (Redundancy)

2) Write-Back Policy

Need "Dirty" Bit.

Write when evict block from cache.

1. complex control logic
2. variable timing (0,1,2)
3. reduce write traffic
4. harder to make reliable

Write-allocate (No write allocate, write without fetch)

Store, cache miss: read data from memory, then update and set dirty bit.

2. AMAT: Average Memory Access Time.

AMAT = Time for a hit + Miss rate × Miss penalty.

3. LRU: Least Recently Used

2-way SA $, add a bit, set to 1 if referenced. reset the other (last used)

Random Replacement, First-in First-Out (FIFO), Not-most-Recently Used (NMRU) (use replacement ptr)

4. Cache Misses (3Cs)

- Compulsory (cold start or process migration).
  Calculate: set the cache size to inf. and fully associative. count # of misses.
  Sol: increase block size (increase miss panalty / miss rate).
- Capacity
  Calculate: change cache size from inf. to current, usually in power of 2. count misses.
  Sol.: increase cache size (increase access time).
- Conflict (Collision)
  Multiple memory locations mapped to the same cache block.
  Calculate: change from fully associative to n-way set associative while counting misses
  Sol. increate cache size
  increate associativity (increase assess time)

5. Improve Cache Performance (AMAT)

Hit time (Smaller cache) decrease block size? associativity?

Miss rate (Bigger cache, Better program)

Miss penalty (Multiple cache level).

Simplicity often wins.

6. Cache flush: invalidate all entries.

Cache capacity: total # of bytes in the cache

Cache line = cache block.

Cache block size: # of bytes in each cache line.

L16. 1. Victim cache. (full-associative, 16-64 entry).

collect evicted cache lines.

2\.

	Hit time	Miss rate	Miss penalty
associativity ↑	↑	↓	–
# entries ↑ (capacity ↑)	↑	↓	–
block size ↑ (same capacity & ways.)	–? ↑? slightly ↓?	↑ ↓at first	–? ↑?

3. Branch Predictor.

e.g. N entry, direct-mapped 1024×1 bit mem.

PC[11:2] as index, if bit set, predict jump.

Branch comp in ALU (EX)

Predict Place	Predict Action	Correct	Stalls
IF	–	✓	0
IF	–	×	2
ID	Taken	✓	1
ID	Not Taken	✓	0
ID	–	×	2

Improve:

Not take 00 - 01 | 10 - 11 take

diminishing returns 收益递减

4. return target location: stack stores ra.

Branch target buffer.

5. Local miss rate of L2$ = L2$ Misses / L1$ Misses

Global miss rate of L2$ = L2$ Misses / Total accesses.

= Local miss rate L2$ × Local miss rate L1$

L17. 1. Performance (Power is not a good measure)

Latency (response time / execution time)

Bandwidth (throughput)

Performance = 1 / Program Execution Time.

$$\text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Instruction Count   CPI   1/Clock rate

ISA affects all 3 things, other affects former 2.

2. Workload: Set of programs run on a computer

programs, inputs, relative frequencies

Benchmark: Program selected for use in comparing computer performance

基准 forms a workload

SPEC: System Performance Evaluation Cooperative

$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$ (than ref computer).

3. m bits × n bits = m+n bit product.

unsigned: 竖式运算

recommend: mulh + mul, div + rem, can fuse 2 ops into 1

4. (尾数) mantissa → $1.01_{two} \times 2^{-1}$ ← exponent

binary point ↑   ↑ radix (base)

Overflow, $> 2.0 \times 2^{126}$, $< -2.0 \times 2^{126}$

Underflow: negative exp larger than field

Significand: sign-magnitude.

Exponent: Biased Notation

Exponent	Significand	Object	
0	0	0	sign for ±0
0	nonzero	Denorm	$\times 2^{-126}$, no leading
1-254	anything	+/- fl. pt. #	
255	0	+/- ∞	
255	nonzero	NaN	Significand to debug

吴昊一 Haoyi Wu

L18. Amdahl's Law, Data-level Parallelism (DLP)

1. Single - Instruction / Single - Data Stream (SISD)

Superscalar is SISD. — Intel Pentium 4
RISCV CPU

Flynn Taxonomy
SIMD, MIMD, MISD(rare).
SSE, MMX, AVX \ multicore CPUs
SSE inst of x86 Intel Xeon e5345 (Clovertown)

2. Amdahl's (Heartbreaking) Law

$$\text{Speedup w/ } E = \frac{1}{(1-F)+\frac{F}{S}}$$

E: enhancement.
F: fraction can accleracte.

3. Strong scaling: speedup without increase size of problem.

Weak scaling: ↗ problem size proportionally to ↗ # of processors

Load balancing (schedule (dynamic))

4. Intel SSE Intrinsics ← These are C instructions that are translated directly to SSE assembler instructions enabling us to put SSE instructions in C code

Loop unrolling. Take a for loop in C and explicitly write a certain number of those loop iterations in the body of loop.

Advantages: less loop overhead; it can avoid data hazards; SIMD instructions can be used.

```
mulpd %xmm0 %xmm1
=> xmm1[0]=xmm0[0]*xmm1[0];
   xmm1[1]=xmm0[1]*xmm1[1];
```

L19. Thread-Level Parallelism (TLP) & OpenMP Intro.

1. Thread: a sequential flow of instructions that performs some task.

Operacting System Threads: give the illusion of many active threads by time-multiplexing software threads onto hardware threads.

Hardware Multithreading (Hyperthreading).

2. OpenMP is a language extension used for multi-threaded, shared-memory parallelism.

```
gcc -fopenmp name.c
# pragma omp parallel private (x)
{                          reduction (+ : sum)
}
```

3.① Read / Write Pairs (Solution 1 for Synchronization).

load reserved: lr rd, rs
store conditional: sc rd, rs1, rs2

Test-and-Set. at s1

② Atomic Memory Operations (AMOs)

```
amoadd.w  rd, rs2, (rs1):
   t = M[x[rs1]];
   x[rd] = t;
   M[x[rs1]] = t + x[rs2];
```

```
       li  t0, 1
Try:   amoswap.w.aq  t1, t0, (a0)
       bnez  t1, Try
       # critical section here
       amoswap.w.rl  x0, x0, (a0)
```

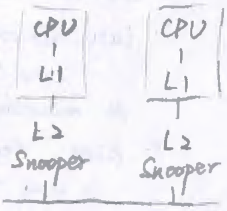
```
        li   t2, 1
Try:    lr   t1, s1
        bne  t1, x0, Try
        sc   t0, s1, t2
        bne  t0, x0, Try
Locked: ...
Unlock: sw  x0, 0(s1)
```

L20 Cache Coherence.

1. When cache miss/writes, notify other processors and invalidate any other copies.

Snoopy Cache

Inclusion property
entries in L1 must in L2.



false sharing: Block ping-pongs between two caches even though processors are accessing disjoint variables

2. Coherence Misses (communication misses) ↑

L21 Intro to Operacting System.

1. core of OS

Provide interaction with the outside world.
Provide isolation between running programs

2. switch on computer.

BIOS: Find a storage device and load first sector
Bootloader: Load the OS kernel from disk into a location in memory and jump to it.
OS Boot: Initialize services, drivers, etc.
Init: Launch an application that waits for input in loop; fork process

3. UEFI, Unified Extensible Firmware Interface
Successor of BIOS (Basic Input Output System)

4. Memory Mapped IO

Polling: Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg to say it's OK. Then loads from input or writes to output data register. → LSB.

cntrl reg.
data reg.

Interrupt-driven IO
Interrupt → CPU Interrupt Table → handler → perform jal to handler → ret

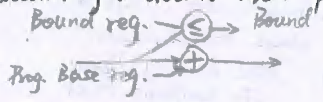
Trap: action of servicing interrupt or exception by hardware jump to "trap handler" code. → SEPC reg. hold inst addr.

5. Handle Traps.

earlier overrides later. M stage

If exception/interruption at commit: update Cause and SEPC reg., kill all stages, inject handler PC into fetch stage.

6. Syscalls, raise software interrupt
Context switch: switch between processes in OS
Scheduling: decide what process to run.



Base and Bound Machine

L22 Virtual Memory

1. Why VM? ① Adding Disks to Hierarchy.
② Simplifying Memory for Apps.
③ Protection Between Processes

Virtual Address Space; Physical Address Space.

Base and Bound → Memory Fragmentation. (lw/sw)

2. 16 KiB DRAM, 4 KiB Pages (VM), 128 B blocks ($), 4B words

3. Each user has a page table
Page Table (PT) in Physical Memory.

4. PTE (Page Table Entry) Access with PT Base Reg. + VPN (Virtual Page Num)
① 1 bit to indicate if page exists.
② PPN (physical page number) / DPN (disk page number)

5. Page fault: Instruction references a memory page that isn't in DRAM.

Page: Internal Fragmentation.

6. TLB: Translation Lookaside Buffers
Cache for the Virtual Memory Page Tables.
TLB reach = Page size × # of TLB entries.

7. Lazy allocation: not used -> not allocated.
read/write/write GBs -> works
No recently used pages -> Memory Compression.

8. KSM: Kernel Samepage Merging eg, 2 Win10 on Ubuntu.
a way to keep 1 copy of pages at host OS level

DMA
① Between L1 & CPU
✓ Free coherency
× Trash CPU working set with transfered data
② Between last-level cache and main memory
✓ not mess with cache
× need to explicitly manage coherency.

L23 Advanced Cache.

1. MRU is LRU (L1, L2)
- Inclusive: L2 contains L1 — evict L1 ✓, evict L2 -> evict L1
- Exclusive: No same elements, evict L1 will go to L2.
- ✓ Non-Inclusive: No relation. evict L1 ✓, evict L2 ✓

2. LLC is not monolithic. Last level cache
read speed differs in different places.

3. Prefetch to help cache.
can eliminate compulsory cache misses
done in cache block granularity.
Reduce miss rate / miss latency, done by hardware, compiler, programmer
Prefetch accuracy = used prefetches / sent prefetches.
stride = 2, N = 1 => 0%
stride = 1, N = 2 => 50%
Hardware Prefetching: Specialized hardware observes load/store access patterns and prefetches data based on past access behaviour.
Next-Line Prefetchers, Stride Prefetchers, Stream buffers

4. Scratchpad Memory (SPM) control the cache.

L25 FPGA (Field Programmable Gate Array)

1. Embedded System Design
any computing system with ① Specifically-functioned ② Tightly constrained ③ Reactive and real-time ④ Hardware and software co-existence.
Timing Performance - Power consumption.
CAD: Computer-Aided Design
ASIC, Application-specific integrated circuits. (Permanent circuitry)

eg: Communicational devices (wired routers)
Automotive applications (airbag release system)
Aerospace applications (auto-pilots)
Defense systems
Radar systems

2. FPGA: Combine flexibility with performance
Shorter time-to-market and longer time-in-market
Functional Block, I/O Block, Routing
LUT: Look-Up Table; LE: logic elements; FF: flip flop

3. Routing, interconnecting.
+ Logic Optimization consume major part of design flow time

4. HDL, hardware description language.

L26. Warehouse-Scale Computing, MapReduce, and Spark.

1. WSC: Request / Data Level Parallelism (RLP / DLP)
low per-unit cost, high # of failures

2. PUE: Power Usage Effectiveness power efficiency measure for WSC (servers)

$$PUE = \frac{\text{Total Building Power}}{\text{IT Equipment Power}}$$

Perfection: 1.0. Google: 1.2 Need cooling 25%

3. RLP, eg Web-Search, Independent, Redundant copies.

4. DLP, MapReduce
① Split inputs, start up programs on a cluster of machines
② Assign map & reduce tasks to idle workers
③ Perform a map task, generate intermediate key/value pairs
④ Write to the buffers.
⑤ Read intermediate key/value pairs, sort them by key
⑥ Perform a reduce task for each intermediate key, write the result to the output files

Map, Devide large data set into pieces for independent parallel processing
Reduce, Combine and process intermediate results to obtain final result.
Hadoop, Spark.

```
file = sc.textFile("hdfs ...")
file.flatMap(lambda line : line.split()) \
    .map(lambda word : (word, 1)) \
    .reduceByKey(lambda a,b : a+b)
```

L27 I/O, DMA, Disks, Networking.

1. DMA, Direct Memory Access
- Allows I/O devices to directly r/w main memory, requires hardware support: DMA engine.
- contains registers writen by CPU. memory addr. to place data, # of bytes, I/O device #, direction, unit of transfer, amount per burst
- Incoming Data: Receive interrupt from device; CPU takes interrupt, begins transfer; Device/DMA engine handle the transfer; Upon completion, Device/DMA engine interrupt the CPU again.
- Outgoing Data: CPU decides to initiate transfer, confirms that external device is ready; CPU begins transfer; Device/DMA engine handle the transfer; Device/DMA engine interrupts the CPU again to signal completion. -> CPU first
- Options: DMA first ← Burst Mode, Cycle Stealing Mode, Transparent Mode

2. HDD: Hard Disk Drives
Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead.
Rotation Time = $\frac{1}{2}$ time of a rotation (revolution)
Seek Time = (Number of tracks / 3) × time to move across 1 track.
- On disk cache, prefetch.

3. Flash, low power, no crashes.

4. Protocol: packet structure and control commands to manage communication.
TCP/IP, Transmission Control Protocol / Internet Protocol
WAN, wide area network.

L28. Dependability and RAID.

1. RAID: Redundant Arrays of Independent Disks.
ECC: Error Correcting Code. EDC: Error Detection Code
- Spatial Redundancy, Temporal Redundancy (Retry)

2. MTTF, Mean Time To Failure
MTTR: Mean Time To Repair
MTBF: Mean Time Between Failures = MTTF + MTTR

$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} \uparrow \Rightarrow MTTF \uparrow, MTTR \downarrow$$

3. AFR: Annualized Failure Rate. = Total time / MTTF

4. Even Parity.
Hamming ECC.

	1	2	3	4	5	6	7	8
p1	⊗		×		×		×	
p2		⊗	×		×	×		
p3				⊗	×	×	×	

Correction: add up wrong parity position (1, 2, 4, 8 ...)

5. RAID all improve performance ×
RAID 0: Striping, High performance. No fault tolerance or redundancy.
RAID 1: Disk Mirroring / Shadowing. Each disk fully duplicated. Most expensive. RAID 10 (striped mirrors), RAID 01 (mirrored stripes)
RAID 3: Parity Disk. Vertically store data, add even parity disk.
RAID 4: High I/O Rate Parity: Devide into small blocks. small reads ✓
RAID 5: High I/O Rate Interleaved Parity. Parity block distributed evenly in blocks small writes ✓ new data xor old data xor old parity = new parity.

L29. Security.

1. The lives of Others - Over the air Implementation flaw
Heartbleed, SSL/TLS: Secure Socket Layer (deprecated). Transport Layer Security.
message + length

2. The lives of Others - Very hard.
PES: Position Error Signal. Disk -> microphone.

3. Inception - Plant a value with a hammer.
DRAM is Prone to Disturbance Errors.
Adjacent rows - victim rows - flip bits
DRAM cells are too close to each other, not electrically isolated from each other. Physical Page # flips -> another page

4. Mission Impossible - when or where
Side Channel Timing / Access (Cache line) Based Attack

5. The water horse - Flush the Loch Ness
Flush and Reload. (build side channel)
inclusive cache - all levels will be flushed. Kernal Space ✓ TLB ×

6. Meltdown
Keywords: cache, timing, speculative execution, memory paging, OS pages, data of another process.
Aim, leak / dump memory.

```
raise_exception();
access(probe_array[secret * 4096]);
```

7. Spectre:

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

(k = array1[x])

Pre-requisities:
i. array1[x], with an out-of-bound x larger than array1_size, resolves to a secret byte k that is cached.
ii. array1_size and array2 uncached.
iii. Previous x values have been valid.

Regarding a misprediction with an illegal x, array2[k*4096] will not be used, but has been loaded into CPU cache. We can use Flush + Reload to guess k with array2. Aim, read out a victim's sensitive information.

CPU with Out of order speculative execution, CPU cache, DRAM, Disk, Network — Vulnerable Architecture