/ /

Chapter 1 / Computer Abstractions and technology

History about this domain.

Eight Great Ideas:-

- Design for Moore's Law — مور هو مؤسس شركة intel
  - هاد ال observation بحق عال transistors
    ↓
    مثلاً دراهم لصغروا حجم
  - كل سنتين او سنة ونص عدد ال transistors اللي بنقدر نحطهم بتضاعف
  - حجم ال transistor صار صغير لنقيس ب nano meter

- abstraction → بندرس الاشياء levels مختلفة
  - we are not reinventing the wheel
  - abstracted levels of design

- Amdahl's law / make the common case fast

- Hierarchy of memory
  many types:
  - DRAM → بس تعدد الانواع وال hierarchy لانه بوفر علي
  - SRAM — قادرين انه نصنع capacity عالية بسعر رخيص
  وسرعة عالية توازي سرعة ال CPU، يعني لو احنا قادرين نصنع من hard disks
  تكنولوجيا ال SRAM، تكون مساحتها 1TeraB، كان كل ال hierarchy مش رح نحتاجها
  بس بدها ملايين!

- Redundancy → الاحتياط

/ /

Static power: the power consumed by the device when it is in quiescent mode.

Dynamic power: when the device in operation

Technology Trends

-Transistors

لحد هلأ قادرين نضاعف العدد حسب moore's law

...ت الحل لكبير.

- علىه احنا نضخم ضعف ال capacity بسهولة، بس مش قادرين نستخدم كلهم بنفس الوقت! ← | Dark Silicon |

↓ The amount of circuitry of an integrated circuit that cannot be powered on at the nominal operating voltage for a given Thermal Design Power (TDP) constraint

زي ال cluster الي على الموبايلات نفسها بحيث عنا اجزاء منه ليراني نشغل لبرامج معينة وبس بتكون طافية، وبالباقي → مطفية، او بتشتغل on/off باحد دوره لا اشي شايا

- single-thread performance

لحيث انتقلنا من ال single core لل multi core.

-frequency and Typical power

- كان توصل هلأ 5GH، ولكن بتشتغل core واحد.

بسبب الفيزياء ← ال transistors الهم نوعين power

← Dynamin Power: بتعتمد على مربع ال voltage وال frequency

فاذا زدنا ال f، ال power تزيد

Technology Trends آخر 40 سنة

- Transistors

- تقريبًا - linear relationship - كم هلأ قادرين نضاعف العدد حسب moore's law

- بس المضاعفة قاعدة هلأ بتاخذ وقت أطول لتكبر.

- ممكن احنا نحط ضعف ال capacity بسهولة، بس مش قادرين نستخدم كلهم بنفس الوقت! ← | Dark Silicon |

The amount of circuitry of an integrated circuit that cannot be powered on at the nominal operating voltage for a given Thermal Design Power (TDP) constraint

↓

زي ال cluster الي على الوبايلات نصها بحمي عنا اجزاء من البيرايني → مطفية، او تشتغل on/off

تشتغل لبرامج معينة والباقي بكون طافي، والباقي بإيصدروم لا اشي ساخن

- single-thread performance

لحيث انتقلنا من ال single core لل multi core

- frequency and Typical power

- خلاص توصل هلأ 5GH ولكن بتشغل core واحد بسبب الفيزياء ← ال transistors الموجودين power

← Dynamic Power: بتعتمد على مربع ال voltage وال frequency

فإذا زدنا ال f، ال power تزيد

- ال Frequency توقف بشكل عليها تقريبًا لأنها صارت مرتبطة بال power والتبريد

كانوا يلعبوا بال voltage مش بال frequency: 2000 ← 70's

خفضها ← حتى يقل ال P

لانها quadratic

مش liner (خطي)

ال P

بنأثر على ال TTL ← بطريقة الي صنع يكون منها

logic 1 أو logic 0

+ve voltage -ve voltage

الأيام هاي: ال voltage الي بنشتغل عليها صرلنا 20 سنة - 1 volt.

↳ range 0.9 → 1.1 (logic 1)

" 0 → 0.2 (logic 0)

بطلنا قادرين نخفض أكثر لأنه صار في عنا تداخل على ال ranges.

← وقتها بلشوا يلعبوا بال voltage بعدها ركزوا على ال frequency لانه بطلوا قادرين يخفضوا ال volt أكثر عشان ما يصير في noise ويولد مشاكل

فنا خلينا نخلص بشكل ثابت كامل ونرجع نزيد لل frequency

hierarchical

Quick overview :-

احنا وقعنا بمشكلة ، احنا اصطدمنا بال Thermal effects ،
نزلنا ال voltage قدمانقدر ، زدنا ال frequency قدمانقدر ، لكن
معادلة لـ زيادة ، الحرارة اللي بتعتمد على مربع ال voltage of power ، وال
frequency ، وصلنا مرحلة مش قادرين نعمل cooling

بلشنا ب ال 2004 يطلع عنا ال multi-core processors
لأنه بدنا performance ، وبدنا progress ويكمل الشغل ما نفلس ↓

- طلع ال intel dual processors / 2006
- " ال intel quad
- صرلنا 11 سنة شغالين عال i core ← i3 , i5 , i7 , i9

هاي التصاميم كانت موجودة قديماً ، لكن moore's law ما ساعدنا ننفذها
لهلأ ، قدرنا نحط ال cores على chip set وحدة ، فلسا نحطها ،
لأنها دلالة هاي ال transistors

زمان كانت تعمل بطريقة عن كل core كانه بيحط chipset وحدة .
كانوا تكون زي chip وحدة عليها ال core الرئيسي ، وتكون حوالها processors
فا كانه multi-core باعمل لحالي ، كانت arithemetic cores / SPU cores
لم built in هوا نفس ال chipset ، وصار ندعم اله مشاكله

number of logical cores

Notes:-

- الـ power كانت تزيد ، يعني لأنه كانت تزيد voltage و Frequency
كانت تخفض voltage ، وتزيد frequency ، كانت بتزيد بس تعتمد على
frequency ، مقدور عليه يعني

(التيار اللي بتسرب)
- بفترة معينة ، ثبتت الـ power بسبب مشاكل زي الـ leakage current

leakage current → كل ما نصغر الـ transistors ، الالكترونات بتتسرب
وبتهرب من الـ path تبعها ، فلما تهرب ، بتتجمع ، وبتعمل حرارة
أخذنا بعين الاعتبار انه سهل نتحكم بالـ dynamic power بس صعب نتحكم بالـ
leakage current ، ووصلنا مرحلة انه 50-60% من الـ power راحة
leakage

Mosfets / Intel 3D Transistors - فبالنتيجة صار عنا:
a field-effect transistor that has a thin layer of silicon oxide between the gate and the channel
بعملوا طبقات دعم
وهلا رجع الـ leakage شوي شوي قليل

Dennard Scaling :

احنا كنا دايماً بنصغر حجم ال transistor ، فكان بيصغر بالطول والعرض

↳ كل ما اقل halfing لحجم ال transistor ، بقدر اعمل 4 اضعاف بنفس المساحة

↳ كانت ال power density تقل للربع بردو

↳ على مدى سنوات طويلة كانت قيمة ال rate ثابتة

↳ بعد فترة بدأنا قدرنا ننزل ال voltage ، بردو كمان بنصغر بس

بقى قدرنا نخفض ال voltage

↳ اذا حللنا بقى ال trend ، راح نوصل لدرجة ال processors

حوالي 6000 درجة مئوية

فبالتالي الحل هنا لل: multi-core ، ونثبت ال frequencies

/ /

Slide (7)

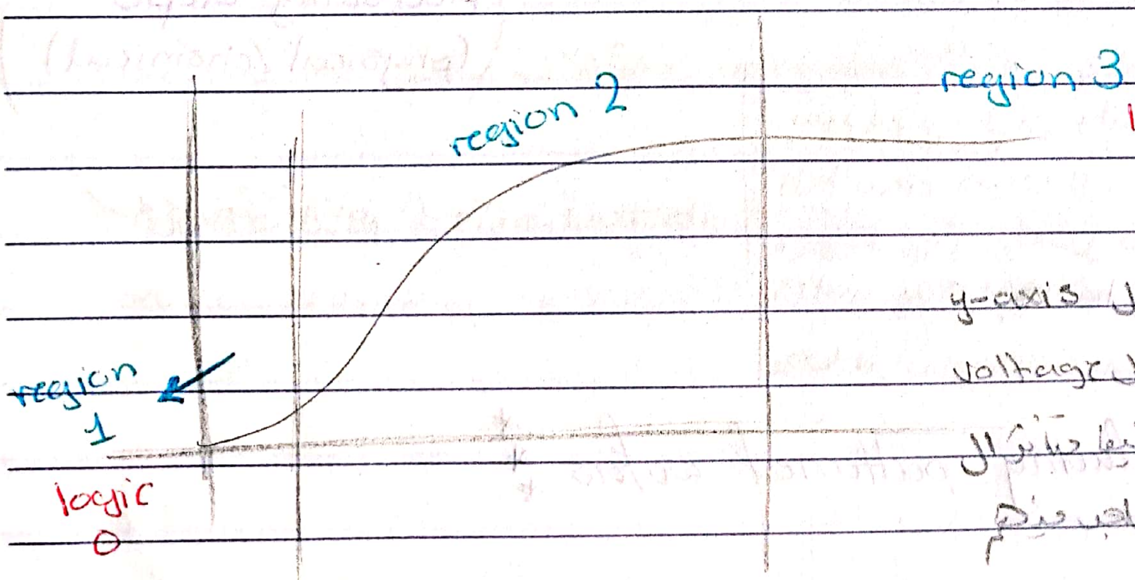
- Basic mechanical tools → بلشت من أيام الإغريق
- Mechanical computers → بلشت في ال 1870 تقريباً

computer systems ← كانوا كلهم mechanical، والفترات اللي كان عاملينها كان فيها analog computing، وكان مسيطر عليها الإشي اللي كان قبل ال transistors، اللي هو ال vacuum tubes

- صاروا يبنوا ال gates من ال transistors

- VLSI → very large scale Integration ← مثال ال gates

* Response of transistor



ال x-axis وال y-axis كانوا يعبروا عن ال voltage وال current، وكيف بتنزل ال response اللي بدي بياه

بيتعامل ال computer ما بيتعاملوا مع region 2، بس 1 و 3

/ /

[Process of manufacturing CPUs] Slide 18

- العملية بتبدأ بالرمل ، فيه شوائب وعناصر معينة ، الرمل بيكون فيه شوائب قليلة
↓
تجميع وتنقية وصهر
↓
silicon ingot ← صبغة ← بتتقطع لـ slices
slicing ← slicer

→ wafer بيوافر اللي تقطع سميها ← handling تتعامل بحذر
ال clean rooms ↓
Blank wafers ← ملساء جداً
↓
(processing steps (physical / chemical))

← بنطبق ال designs بال abstraction

بنفرد ال designs ، لو ال circuits تبعت ال circuits بنفردها لل counters وهما بنفردهم لل flip flops ، وهكذا بنفردهم لل gates ، وال gates بنفردهم لل transistors فبنطبعها ونحط فيها مليارات ال transistors

→ producing patterned wafers ***

from patterned wafers :-

↘ || wafer testing | → stage 1 of testing

بنزل عليها جهاز كله اسلاك مجهزة ناعمة

بينزل كل مجس فوق كل ببسليكة ، وبتلامس مخارج الـ CPU's

بتصير بتجرب signals ← inputs ، ولازم وصول الـ outputs

يطابقوا المواصفات المتوقعة outputs ← ليتأكد انه شغال صح

لو اكتشفنا بعد الـ testing انه الـ IC (integrated circuit) مزبوطة (CPU) الموجودة على wafer

اعلاه نقش لا نصنعه لو بانتقية بنزل الصنف مش شرط تكون نتيجة سلبية 100% ، فبتأثر على الـ IC's ، او بسبب خلل بالـ processing

الـ yield هو نسبة الـ IC's اللي نجحت على الـ wafer

↓

هلأ بيفصلوا الـ IC's الصحيحة عن الفاشلة وبيلشوا بالـ bonding

↳ Bond die to package (process for placing a chip on a package substrate) مشان نجمع الـ CPU's مع

Integrated circuit cost

كل ما زادت ال die area ، كل ما قلت ال yield ، كلما زادت التكلفة

Measuring Execution Time

System time ≡ Elapsed time *

- احنا ما بنقدر نشيل كل الوقت اللي بياخده ال OS على ال task من ال CPU time ، بيدخل في overhead.

لهيك قسمنا ال CPU time لجزئين :

① user CPU time
↓
الوقت اللي بتنفذ بالزبط على كود البرنامج بدون أي شيء آخر

② System CPU time
↓
الكود الأصلي + الجزء اللي ما قدرنا نشيله والمتعلق بال OS

← لأنه كل البرامج تنادي طلبات ال OS

↓
| this integrated |

- Slide 15 //

- Different programs are affected differently by CPU and system performance

two types of applications

- CPU bound → تطبيقات فيها ال processing عالية

- Memory bound → زي ال add وال shift
↓

برامج عليها ال load / store

فمعناته مع ال RAM و ال disk

/ /

CPU clocking

- Gates (And, OR, Not) .. they are only driven by their inputs.

- All our digital circuits are clock based

clock period / Clock cycle time = $\frac{1}{\text{Frequency / rate}}$

- clock is the main factor that affect the entire hardware
↓
كل ما زادت سرعة ال clk بزيد ال performance بافتراض ثبات ال design نفسه

→ in slide 17 : CPU time is the total time not the user CPU time or system CPU time.

- there is always trading off between the factors that can be used to improve the performance (slide 17)

IC and CPI

How the compiler affects IC?

instruction mix ; compiler كل ما كان ال ذكي بيقدر يختار
instructions ينفذ كود ال C وسطارته كل ما يوجد assembly
اقل ، يكون احسن

- The hardware affects IC depending on program, ISA, compiler, and the design of hardware itself.

→ The actual CPI is ICI

T_c : cycle period.

/ /

§ 1.7 // Power Trends

Summary of power trend from previous lectures:

- Dynamic power ✓
- Static power ✓
- problems of leakage current ✓

- بدأنا نثبت ال frequency عند مرحلة معينة ، او بدون تغيير كبير انزلت (فترة بعدين صارت بنفس الحدود تقريبًا)

- ثانيًا لنتحكم فيه انه يزيدوا عدد ال cores من هداك اليوم لهاد اليوم

- معادلة ال power بتعتمد على ① مربع ال voltage و ② frequency

- صرنا نخفض ال voltage من 5 volt TTL ← لـ 3.3 ← لحد 1.1
الايام هاي ، ومش قادرين انزلها اكثر بسبب ال noise ، بنبطل قادرين نميز ال 0s وال 1s من بعض اذا نزلنا تحت ال 1 volt

- قلنا دائمًا نلعب بال frequency ، كنا انزله مرحلة ، ونصغر ال transistor من هون ، وكانت ال power density ثابتة ← denard scaling

كنت بعتبر احط اربع اضعاف ال transistor بنفس المساحة مثلًا بس بنفس الوقت استهلاك ربع طاقة ، طاقتنا ($\frac{1}{4} \times 4$) بيعطي $\frac{1}{1}$
← ثابتة ال power

Scanned with CamScanner

- كنا ملاحظين قادرين ننزل الـ voltage، فقدنا هاي الميزة في الـ 2004 اللي هو Dennard scaling، وبالتالي بلشت الـ Dynamic power تزيد

- صرنا كل ما صغرنا الـ transistors، الـ leakage يزيد، فمن هناك اليوم لهاد اليوم واحنا بنحكي عن مشكلتين الـ Dynamic والـ static

Example:- (slide 23)

تم تصميم CPU جديد، في عنا أصلاً واحد قديم كـ base line أو reference، وصممت واحد جديد، فرق عن طريقه أقل الـ capacitive load بمقدار 15%، يعني هلا صار الـ

capactive load (new) = 85% of capacitive load (old)

ونقص، لو كنت نزلنا الـ voltage بمقدار 15% بدون مشاكل الـ noise

+ نزلنا الـ CPU frequency بمقدار 15%

→ ① نزلنا 15% من كلشي، فصار التسكين نزلنا الـ power للنص

$$\text{Dynamic Power} = \left(\frac{1}{2} \times C_{old} * 0.85\right) \times \left(V_{old} * 0.85\right)^2 \times \left(F_{old} * 0.85\right)$$

In comparison: $\frac{P_{new}}{P_{old}} = 0.85^4 = 0.52$

- The power wall
  - can't reduce voltage
  - still can (because of moore's law) many many transistors
  - These lead to increasing power density (because denard law has fallen)

but: still can't remove more heat caused by this power (no enough cooling capability)

→ Slide # 25:

the trend is not straight.

في دكسين، لكن في تباطؤ في التحسن

/ /

Multiprocessors

عند ال parallel programming

← مشكلة كانت إنه تفاصيل ال Hardware بتكون مخفية أحيانا عن المبرمج، ممكن يكتب كود parallel ولكن ال code تا بحدد كم حي core عند ال user، حال، actual performance ولتحسين شيء عن معروف إله وعن محسوبة بكفاءة.

← ال parallel programming صعبة لليوم، لأنه بيرضى تركيزه حل المشكلة قد ما في تركيز على ال performance نفسه

- load balancing: عشان ما يكبر ال load على ال core الواحد والباقيين مرتاحين ← توزع ال balance ع كل ال cores

- ال CPUs صممت مع أن لا تسخن عندها خاصية اسمها ال throttling ← لما تسخن، تنزل ال frequency للنصف، وهي تنزل استهلاك الطاقة

Dynamic Frequency scaling

لـ لذلك ال load balancing مهم، وهي اتفادى ال Hot spots، فبالتالي ال hardware ما ومنصف بسرعة وبالنتيجة أنا ما استفيد

تركيز الشغل على core واحد

مع بعض فرق بين separate programs شغالة على cores مختلفة.
مثلاً microsoft word على core، core على excel، وهكذا، هاد بيحكلك إنه كودي نفسه parallel، إنه الكود
بيقسم للبرنامج الواحد بحيث بيشتغل على كل ال cores مرة وحدة.
وهاد هو موضوع ال parallel processing

← فالمشكلة في ال parallel programming كيف أعمل عمليات communication + synchronization عشان كل ال cores بيشتغلوا مع بعض إلخ.

Benchmarking → المعايرة
Benchmark → المعيار

لما نحكي إنه جهاد
(جهاز أحسن من جهاز؟!

← كثير website بيعمل profiling لكل ال CPUs وبتعطي ال benchmarks

The standard benchmark in industry (The SPEC)
↓
measuring CPU time

- What are we benchmarking?!
  → CPU ?!
  → Entire System ?! (response time)

Benchmark

- اذا بدي اصمم Benchmark (تقييس المستمر)، لازم أنتقي من load البرامج تقييس الأداء للكامل
- اذا حبيت بدي اقييس اداء ال CPU جيد، فلازم اختار ال CPU bound Apps، اللي بتوصي وتحكيها تشتغل عال CPU، وتحتوي حجز عال memory وهيك

- programs used to measure performance:
- Supposedly typical of actual workload. ← ال tools اللي بتختارها مش بس تختارها بشكل صحيح، في شوية تفاصيل اكثر من هيك
→ لازم يكون نموذجي لحسب العمل الفعلي.

- يعني مثلاً فلازم اختار tools ال user عمره ما ح يستخدمها ← مثلاً لو بدنا نقيس machine for Gaming بدنا نستخدم tools الخاصة باختبارها.

← ال apps المستخدمة كـ tools ال SPEC مقسومة قسمين:
① مختصة بال integers ② مختصة بال floating point.

- we have to rely on something, (a reference for comparison)

Case Study / Slide 130 → الجدول بحكي عن البرامج اللي مستخدمة بالقياس

CINT: integer benchmark. , CFP: floating-point.

→ CORE i7 920
الأرقام اللي بتشكل (الوحش) بحيروا عند ال generation.
اول 3 ارقام ع اليمين بكونوا ال ~~generation~~ model number

* هل هذا ال bunch of programs بيعبر عن actual workload ؟
لا ناس بتحكي آه، ناس بتحكي لا.

- SPEC ratio بين ال → $\frac{\text{Reference time}}{\text{Execution time}}$

to find a number that describes the measurement for the user:-

- use geometric mean not the Arithmetic mean ← لأنه تتأثر بالقيم القصوى
slide 30
↓
بضربوا الارقام ببعض، وبياخذوا جذر عددها
يحتوي جذر بعدد ال ratios

→ in the study case you'll take $\sqrt[12]{\text{product of ratios}}$

tec

عشان نوفر طاقة وكلفة ونحمي البيئة.

-voltage
-capacitance
- frequency

- static power
has leakage current.
power distributed for useful work or unuseful work.

②

- on load zero %: → بستهلك طاقة بدون ما أشغل اشي.

- static power موجودة دائماً حتى عال 0% load. → بتشكل نسبة كبيرة من ال total power consumption. → حتى تقلل قيمتها وتأثيرها لازم نستخدم ال loads الأعلى

- كل ما اطلع من load لـ load أعلى، ما تتناسب بشكل طردي مع زيادة أو تغير ال Average power

- [low power at IDLE] is a fallacy.

- اجهزة google مثلاً دائماً مضغوطة بين ال 10% - 50%، حسب الـ من شغلها بكون ال load ← 100%

نسبة ال static power أعلى من actual useful work

- الهدف هنا، انه نصمم اجهزة فيها استهلاك طاقة يتناسب طردياً مع ال load حتى نخفف من هاي المشاكل

- desiring to design processors where power is directly proportional to load.

/ /

Pitfalls: Amdahl's Law → 1.10

3 laws controlled computer design last 60 years:
- moore's law is failing
- dinard scaling has fallen in 2004
- amdahl's law still affecting

→ "Make the common case fast"

- $T_{affected}$ → اللي بدك تشتغل عليه وتحسنه

to know how much you improved.

$$\frac{T_{old}\text{(long time)}}{T_{new}\text{(short time)}} = \text{improvement (speedup)}.$$

Pitfalls: MIPS as a metric

$$MIPS = \frac{\text{Clock rate}}{CPI \times 10^{6}}$$

↳ بفترضوا بالمعادلة هاي إنه CPI بتتأثر بال HW
وهذا غلط، الخوارزمية و الكومبايلر كمان بأثروا على ال CPI
ليس عاملاً يتم تحديد ال CPU حاله.

Scanned with CamScanner

Example for Amdahl's

program of 4 operations:

Divide	improving factor = 10	exec-time =	10
Multiply	" " = 5	" " =	20
ADD/Multiply	" " = 2	" " =	50
Other	no improvement	" " =	20

$$T_{new} = \frac{10}{10} + \frac{20}{5} + \frac{50}{2} + 20 = 1 + 4 + 25 + 20$$

$$= 50s$$

Chapter 4: The Processor

pipeline: opposed to single cycle

اول ما كان عندك single cycle ← كل ال hardware بكون بشتغل stages بس stage مش حاليا فما بيعمل instruction غير كل ال HW مشغول فيها، ما بقدر يعمل أي instruction ثانية

انه اصلاً ما عندك حد الـ registers او ال (pipes) تحطها بين كل stage والثانية

↓

propagation time for single cycle is big.

Max frequency $= \frac{1}{time}$ ← the min time to implement 1 inst

← وكل ما زاد تعقيد ال Processor بزيد ال time ويقل ال frequency فمش بنحكم جهاز بساعد ما بيصير.

Pipeline helped to solve these problems:

- parallelism
- pipes between each two stages, then the one stage becomes smaller, logic hardware of it is less, then propagation time of each stage is smaller → Higher frequency ← الي بحدد كم ال Max frequency هو مش ال path بأكمله، ولكن اطول stage بال path

for this reason, ISA + internal → design of computer, & they affect frequency.

Five-Stage Pipeline

- 4 pipes
- why pipes?
  - made from registers made from flip-flops
  - Sequential circuits (flip-flops), they are memory returning circuits, we save data in them, then by these pipes (registers) we are returning information from the previous instruction as it goes to the next stage. ← بنحفظ معلومات ال inst حتى ما تضيع

لـ مثل ال opcode وال operands وال control signals

to make this work اللي خاصة فيه

إضافة إلى أهم حبتين ال stages

Designs + Architectures

Harvard → for processors including embedded components or RISC-V

- physically RAMs ال مقسومة لقسمين مختلفين
- memory sticks: for codes
- another memory stick: data that codes work on

Von Neumann → Laptops + PC's

- one memory stick for every thing
- managing data flow is the job of OS + internal system

/ /

← Harvard بيحتاج ضعف كمية الأسلاك اللي في Von Neuman ، فيه
مكانين مش مكان واحد زي Von Neuman

→ Queueing problems appear for von Neuman, because one stick is doing the entire job, you have to wait.

↑ Harvard solves this

- the big amount of wires in Harvard causes, electrical interferance, causing a noise that affects the integrity of data

- in Harvard, each memory stick, can be made separately from different technology (SRAM, DRAM)

- Most Von Neuman is DRAM most times.

lecture 06 min 14

Five-stage Pipeline design

- ~~there is~~

- two distinct memories → Instruction memory / → Data memory] → Harvard design

Stage ① | Fetching |

- each memory location has an address in Inst. memory

- PC: program counter
  - → Instruction length: 32-bit or 64-bit
  - these days, most architectures are 64-bit designed
    - ↳ hense: inst. length = 64-bit

RISC-V → 32-bit design: adder after PC has operand | 4 | → inst. need 4 bytes in memory

64-bit design: " " " " " | 8 | → inst. need 8 bytes in memory

↓ constant

Stage ② | Decode |

register file: built-in: inside the core of CPU →
number for registers: 32 for integers. only.
↳ registers

ال core هو اللي بجيب منها الداتا اللي بتكون في الداتا ميموري اللي بتروح عال cache، واللي بنشتغل عليها في هاي اللحظة، هي اللي بتكون في ال SRAM Block اللي هي ال registers

↳ ↓ | length of one register is 64-bit |

| Floating-point عنا زيهم بس لل | ✓

| vectors وفي كمان لل | ✓

5-bits ← ؟ operand شو هو size اللي لحددنيه نوع ال ، → على اعتبار انه احنا عنا 32-bit RISC ← هو طول ال ~~pipe~~ register file لم

↓

$2^5 = 32$

→ بنحطوا من ال .inst وبروحوا على ال read ports

Opcode: is the part of instruction that identifies whether the second operand is going to be a constant or an operand of register file.

→ control unit is in this stage

Stage (3) : execute

- the design in slide 5 is 64-bit
- 32 registers : length of each is 8 bytes : 64 bits

ALU :

- first operand is always coming from the instruction
- Opcode controls behaviour of ALU control

Control signals

- RegWrite : decide to read from or write on register file stage (2)
- ALUSrc : ALU source: to choose one between imm, and inst. operand stage (3)

zero flag ?
- carry :
- half carry :
- overflow :

important flags for designs.

most important

- AddSum : address sum unit ✓ calculating target address
- when adding negative number it is mostly for (for-loop)
- positive added number (forwarding)

Stage (4)

- Branch control

↓

zero flag + branch line signal

0 ↩ 1 ↩

↓

Branch instruction

↓

you'll return PCSrc with value = 1 going to the Mux of PC to take next or previous instruction.

← بس هيك اذا وقت اي برنش اعمل Branching ، في ال stage الرابعة وقررنا فعلياً بالمستقبل ، وهيك كمان بكون رجعت instructions جداد عشان ال stages اللي قبل الرابعة بالترتيب ، وبنضفهم كلهم ← we flush every thing we took into stages by making all control signals equal zero just like we made empty instructions (bubble)

/ /

stage (5) Write Back

↳ to write on register file.

Question:-

assume in slide 5:-

stage 1: 10 ns
" 2: 15 ns
" 3: 50 ns
" 4: 15 ns
" 5: 5 ns

max frequency is $\frac{1}{50 ns}$

↳ اطول زمن لأطول stage فيهم

يمثل الزمن تبع ال cycle.

↓

job of designers:
1- to minimize the time elapsed by the longest stage
- another solution is to take the longest stage and divide it into many stages with less time, so that the longest elapsed time is less than the time elapsed by the original stage

مشكلة هاد حكي إنه لما نخلي هيك،
اصلا بنصير ال pipes اكثر، وهما ال
cost بتدفع علي، لانه هما عبارة
عن flip-flops، يعني sequential circuits

يعني بدهم clock، وفيه حرصنا
على ال interference، اصلا كهربا
- كل كمبيوتر، عدد ال stages اللي قبل
كثير بصير ال branch فتحتاج انا بكون مجهزين instructions
كثير الالاشي، وصرت كهربا عالفاضي وبدي flush
كل ال pipes

/ /

Pipelined Control

- we need to determine the control signals, as soon as possible, directly after determining the instruction

→ using Opcode to determine control signals

Hazards:-

- structural hazards happens for the five-stage pipeline when the hardware tries to load many instructions at one time, then there will be competition on parts of RF and ALU, like when you load 4 instructions and you have 2 ALUs
→ a lot of instructions + few resources ≡ delay and Queuing

- Data hazards
like Read after write.
to try to read from a register you haven't write on it yet.

- Control hazards
like late branch control.

The problem: ~~dependen~~ dependancy problem, sub inst / no problem, add inst needs to use the latest value of X2, X2 is found by sub inst, in AlU, and will be written on X2 reg in stage 5

يعني الخطوة اللي ال and بدها تقرأ X2 ، ال sub لساتو ما حسبها

Data Hazard in ALU instructions

Hardware level code analysis

and → راح تضلها واقفة بال decode stage

ولازم اسمح ال sub inst.

انو تضله يمشي بال pipeline

logical errors → لازم يكون عنا طريقة نوقف ال and لحد ما تظهر ال X2 من ال sub → stall

مش لازم احكي ال data تبعت ال and واقفة بال ID

مع ال clock cycles ترجع من ال pipe، بنانحكي complete hold

بالجزء الاول من ال pipeline، بس احكي ال sub تضلها ماشية

لباقي ال pipeline

and ح ال ID، وال sub بتمشي لا ال memory هدي

راح ترجع عل ال WB، بهاي اللحظات تكون stall

contents of execute + memory stages → should set all control signals to 0, nothing will respond to you (bubbles)

→ when add enters ID stage, sub will be in WB stage

نقرأ ونكتب ع X2 بنفس اللحظة عال RF، وهاد conflict

فشعنا هيك تصير الكتابة قبل القراءة

لعشان نوفر stalls وحل المشكلة بدون ولهم، بنحلها بال forwarding

/ /

Forwarding Solution for Dependencies

- ↳ only for structural hazards (وحدة واحدة بتحت بدون وقت)

how to solve :- Before that, we have this solution for instructions in the case of add to solve the conflict

- clock has two edges

always write values on RF in first edge

" read " from " " second "

to avoid the conflict

↓

ال edge الأول بكتب فيها ال latest update

مع الثانية انا بأكون ضامن انه القيمة انكتبت

This solves the conflict in between ← sub, add.

slide/12: how many true dependencies ?!

- and inst + or inst are true dependencies.
- add dependency is solved by hardware ~~forwarding~~ clocking
- sd has no dependencies.

total: 2 true dependencies.

↓

Most likely in 5-stage pipeline, we have the 2-subsequent have the risk of true data dependency.

Scanned with CamScanner

Forwarding solution / slide 14

instead of stall for and, or

- data is ready in EX stage of sub, why to wait until WB?
- do not read from RF for and, or, just take directly from where it is produced
- other instructions find it in reg file

Forwarding unit | comparator ✓

① when to forward?

in next inst use an operand need to be evaluated in previous inst (1st operand or 2nd operand = destination of previous inst)

inputs of FU:-

R1: 1st operand

R2: 2nd operand

EX/MEM.RegisterRd: destination of previous inst (Register) — first previous one

MEM/WB.RegisterRd: " " " " — second previous one.

outputs: slide 14

generates outputs it needs, forwarding them to the MUX (there must be control signals managing the mux: if forwarding is on, you take my forwarded data as an input for your ALU) otherwise you complete your path, with no need to any forwarded values, as there is no true dependency

solution of slide 15 in lecture 07/ min 22

with forwarding design slide 15

- X2 is ready in EX stage / cycle 3 / sub inst.
- ID of and won't read from RF the value of X2
- EX of and will read X2 by forwarding
- ID of or " " " " "

قيمة X2 ، بتمرّرها ال and وهي بال EX ، بالنسبة لل sub
بتكون بال MEM ، ما بدها تسأل مباشرة من ال EX بس sub ، وبشيك مع
ال Forwarding Unit عشان sub بس ال X2 لل MEM
اخطيها منهان لـ and وهي بال EX ، وبالتالي تمرّرها لل ALU بس
sub عشان اخطيها منهان لـ or وهي بال ID

← هاد فسر سلايد 14 لما ال FU تقرأ ال RD مرة من ال EX/M وال M/WB تجو لال sub

cycle 4 بتستخدم X2 and
cycle 5 بتستخدم X2 or ← ماسكلوا اسوا

3	4	5
EX*	MEM	WB
ID	EX	MEM
	ID	EX

→ we need X2 in EX, so we forward it to EX stage.

Forwarding works, and optmizing number of stalls if the instructions that are consecutive, are ~~arithmetic~~ arithmetic and logical, because their results, are prepared in EX stage Unlike load-use operations ~~and data hazards~~.

load-use → values are generated in MEM stage.

slide 16 ← load: load from memory and write to RF

min 26 lect 07

/ /

slide 17

assuming Forwarding

- value of load will be prepared in MEM stage of ld

4 5 6

MEM

EX MEM

لما تجهز القيمة لل next stage تبعت ال inst اللي بعدها بدخلشوا بنفس ال cycle بالحلوة بس تطلع من stage بتدخل اللي بعدها باستثناء ال WB دخل جاهز عادي بنقدر اموره ثم منها لا نفس ال cycle عشان ال clocking

cycle عال اللي بعدها

because of this, we are forced to stay on ID stage for sub inst for 2 cycles, until we can forward the loaded value from MEM of ld to EX of sub

There is Forwarding but with one bubble

/ /

Two ways to do load hazard detection.

① ID/EX|MemRead and Rd ↔ RS1, Rd ↔ RS2

→ control signal هي ال الوحيدة اللي تشتغل بال load لأنه اصلاً ال load ماستعمل اني حيز reading from ال memory.
طول ما قيمة ال (1) MemRead
يعني اصلا في ال Ld

Slide 18

② Opcode of the inst compared with Opcode of load and Rd ↔ RS1, Rd ↔ RS2

Hazard detection Unit is found in D stage of next instruction because if hazard detected we want to stall in D of dependent inst.

Scanned with CamScanner

Slide 19 // stall circuit → should have five NORs not 4, to be able to compare any possible register from 32 ones.

$2^5 = 32$ ←

↓

comparing registers RD and RS1, RS2

truth table of NOR

A	B	output
0	0	1
0	1	0
1	0	0
1	1	0

→ identical values = 1

otherwise = 0

A in this circuit → RS1 as example.

B " " " → Rd " "

← checking the first operand.

There will be another exactly the same circuit to compare RS2 with Rd (second operand)

The output of these two circuits will be both connected to an OR gate in order to say (yes there is dependency) if atleast one of these circuits has an output 1

you can't ~~[illegible]~~ stop combinational circuits → (مانع (الي زاد راح) , but you can stop sequential circuits
↓
Flip-Flops → let them keep previous values but not to update.
↓
How?
First solution : stop the clock and set it to 0
↓
نحط ال clk نفسه , يعني ال clk
اللي اعطيها input عن and gate .
ثاني ~~[illegible]~~ , فلو حطينا 1 بتضل
ال clk ماشية , ولو حطينا 0 بتبطل
ال clk موجودة.
↓
بس ما تحطلوا !!

→ The second way !!

Scanned with CamScanner

Way ② do not update!

→ PC counter should keep the previous value, and never update.

→ every thing stored in the pipe between IF and ID will remain the same

↑

حتى نمنع ال update عليهم ، وهما اصلاً كلهم register

نجعلهم ← disable

لأنه لما تمر ال clk عليهم تكون ماشية ما بتوقف
وال flip-flop بتكون disabled ، فهيك
؟ ← بروحوا تيجي العملية
هيك احنا بنعمل stall

Design of Register

D flip-flop   JK flip-flops

→ inputs (Q, $\bar{Q}$) or D

→ outputs

→ clk

→ set, reset

→ there will be enable/disable wire.

/ /

Datapath with Hazard detection

Hazard detection unit:

inputs: RS1, RS2, RD
+ ID/EX. MemRead control signal to detect load hazard

↓

اذا تحققت الشروط الي محققة منم بدها تعمل stall

بدنا نصفر ال control الاصفر و disable لل instructions الي عم تعمل

* first output: to the mux in ID stage that controls WB, Mem, EX control signals

→ control lines pass as usual
→ if hazard is detected, the selection lines forces the mux to pass the zero value and assign it to all control signals mentioned above

Two inputs of mux:
- 0
- control.

second output of Hazard detection unit:
- PCWrite: to disable if needed.

Third output:
- IF/ID Write: to disable pipe if needed.

Solution of slide 25 in lec 07 min 44

example slide 26 in lec 07 min 49

In General what is happening now making:

- registers of size 64-bit
- addresses of size 64-bit
- any thing loaded from the memory is of size 64-bit
- instructions depend on the architecture : 4 bytes or 8 bytes.

In slide 11:-

when add is in EX stage (will read from RF) and sub is in WB stage (will write on RF), and both are happening in the same time in the same cycle (This is a conflict)

This solved by clocking

- The job of rearranging codes is the job of an intelligent compiler, it does filling of such an instruction: that has no true dependency between two instruction, ~~that are depending on~~ where one of them depends on the other

↓To solve load-use hazards, because other data hazards can be solved by forwarding.

In slide 7

+ the operations and calculation in order to determine we have a branch is actually taken is done in EX stage, using zero flag an output + calculating the next branch pc

- But the real action is going to be done in MEM stage, (we don't take action on branch is taken) except in MEM stage.

لحد ما نعمل هيك بكون داخل 3 instructions طول ال pipeline
وهذا هو ال fall through يعني ال sequential instructions ،
المرتبين ورا بعض بعد ال branch ، [وهذا بضيف وقت] وقد يغير
النتائج

let assume that we have:-

beq x1, x2, 16 -> offset.

using design of slide 71-

- instruction is Fetched is IF stage
- registers are read in ID stage
- (x1 - x2) is done in the ALU of EX stage producing a zero flag, for which the zero value means, (x1=x2) then the branch is taken + next PC is found here by (AddSum circuit)
- producing zero flag as an input to the branch detector logic (And Gate), with contro signal of branch in MEM stage which is too late ✱

Observation #1 update

x1, x2 are read in stage ID, so we can work on them as soon as ID stage.

Observation #2

PC address which is used to find target PC is detected in IF stage and is passed through all stages until it reaches AddSum circuit in EX stage.

Observation #3

to know if the inst is branch, you can do that in ID stage by (branch) signal, you are not forced to wait until MEM stage comes and use this signal.

Solution #1

you can rearrange circuits in away you finish the job as soon as ID stage and protect 2 cycles from stalling.

Target address in branch inst. is calculated by adding (current address + shifted literal by 2)

in slide 30

40 + (16x2) = 40+32=72

you first sign-extend this literal and then you multiply by 2, by doing shifting in hardware

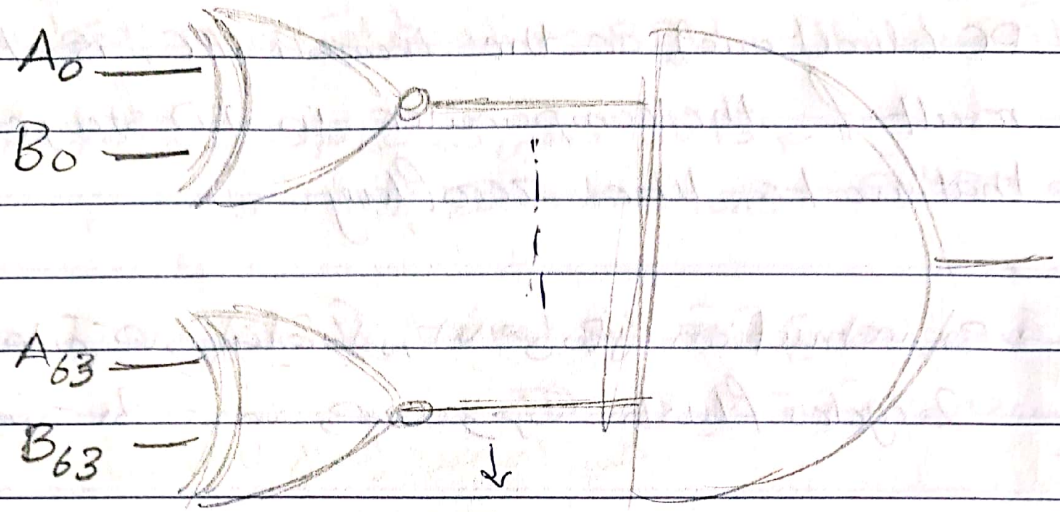
This solution is represented in slide 32

① - comparative circuits directly after RF to compare betwee RS1, RS2

The design of this comparative circuit it is made from ~~XNOR~~ XNOR Gates

if RS1 = A , RS2 = B

then :-



- you have 64 NORS here
- This comparator is like the stall circuit in slide 19.

But NOTE: AND gate can't have more than 8 inputs, then NORS are distributed to 8 ANDs then these ANDs are connected to one last AND

- This step detects if the branch is taken or not.

* ② AddSum circuit that will calculates the target PC, is put in ID stage not the EX
  - The shift circuit that multiplies the literal by 2 to find the offset is put with AddSum in ID after extending it in ImmGen

/ /

③ The selection line of the mux in stage IF that decides what to take, the next PC (direct one), or the branch PC, is the result of the comparator in ID stage that works like zeroflag.

طب ما تم هاي الحالة بكونه انا جبت بس instruction وحدة، ما صارت الا inst وحدة ومرتين 2 cycles flush

This solution is not optimal

because the designs of these days are deep pipeline designs, and this solution can't help in them.

BPU: Branch Prediction Unit

Solution #2
solution of RISC-V pipeline

ما بدنا تعقيدات ال BPU ، بدنا نحط افتراض واحد فقط ونمشي عليه
عملية احصائية ، يعني أنا تصميمي مبني على معرفة عالية بطبيعة ال instructions بلكي لأداء افضل

- طبعاً هون بإنه يتم توقع ال branch بال IF
- ويحسب قيمه حقيقة في ال ID

Giving not taken prediction ← هون اصلاً بفرض إنه ال BPU دائماً

مثال سلايد 36

- على افتراض إنه ال BPU دائماً بتعطي ← not taken
فضل ياخذ دائماً تاسيز بنحط ال fall through
↓
sequential instruction

- يعني بدنا نفترض انه ال comparator بال ID ونه عند اعطى انه ال RS1, RS2 مساويين مثلا يا branching

- إذا طلع ال comparator في ال ID اعطى انه Taken
خلافاً لل BPU ، هون اصلاً بدنا نعمل flush لل inst اللي كنا دخلناها ، ونخسر cycle

هذا الحل مش دايماً بزبط حسب الكود اللي ما فيش branches و احصائية الاداء

↑
This is called Predict Not Taken

The other design is Predict taken

improvement #1 on solution 2#

static branch prediction

↓

Backward

↓

Taken

↓

لانه بيروح عالبا

loops لاني

cost لذلك

wrong ال

prediction

صعب تكون اعلى

not taken من taken

لانه عند ال

عدد لفاته كتير

↓

forward

↓

Not Taken

↓

مع ال if

وال else

انا وحظي

improvement #2 on solution 2#

Dynamic branch prediction.

what is the history of this instruction behaviour?

- buffering: عملية تخزين.

Dynamic prediction buffer(table) :
all is put in IF stage
↓
لأنه بدنا نخفف ال stalls اللي بتصير هنا.

- We won't have any stall cycle only if the BPU in this case have given a wrong prediction.
- This wrong prediction will be ~~predicted~~ detected in ID by the logic of branching
- any wrong prediction will be added as new data to the buffer of BPU

The BHT (Branch History Table)

→ We know the branch inst from its address

← ال table بتحتوي ال addresses تبعون ال branch inst.

← المشكلة هون إنه ال processor هو 64-bit based ← ما نقدروا نبنوا هاردوير بتحمل 2^64 من ال addresses in total لـ

فصاروا ياخذوا عدد معين من ال bits. أول 6 مثلاً

مثلاً لو ياخذ 6-bits بصير طول ال table ← $64 = 2^6$

← كل ما أخذ أكبر وآخذ bits أكثر بتقل احتمالية تتشابه ال bits لأكثر من inst (conflict)

/ /

Big BHT : cost / time problems
small BHT : conflicts problems.

Table size = $n * 2^k$ bits
n : n-bit counter
k : bits you take from the addresses

→ A finute state machine

- The one bit predictor does the Job of updating data in BHT

مشكلته إنه عجول
لو update (على آخر
طلع مخالف بروح تحل
ال table بسرعة و
اذا التوقعات اختلفت
ويغيرت) بنفس هيك
يكون هيا ترها القد بال
comperator مع ال ID

→ هاي المشكلة بتبين لما يكون
في for loop داخل
for loop
↓
slide 41
assuming
→ inner loop will be executed 1000 times
→ outer loop will do so
↓
Million times you'll reach bey in the table

/ /

2-bit Predictor

Deep NT	Shallow NT	Deep taken	Shallow taken

بدل ما ننتقل مرة ثانية من T لـ NT دغري عشان نظل
ننتقل من Deep Taken ← Shallow Taken

← اذا صرلنا مرتين ورا بعض، كانه يا pattern، ووقتها
نقلب ال prediction

~~Improvement~~

Improvement on the whole design:-

- predict in IF and calculate target PC also how?

① first time you access beq inst as example, it will be calculated in ID by AddSum, and then save this target address to the corresponding inst address in BHT in the same raw with its 2-bit or 1-bit predictor value, to call this table BTB after adding the target PC ← هيك احنا مضطرين نعرف

على ال ID لنعرف ال target مرة وحدة بس، لتحسين دخير الـ predicted في ال IF من ال table

BHT: without target PC
BTB: with target PC ← Some hardwares divide the to 2 table
T1: inst address + bit-predictor
T2: " " + target PC

Another solution:
is to ~~built~~ build an ALU or adder that calculates the target PC in IF instead of dependency on the one in ID
هل اذا حطينا وحدة راح نستغني عن ال IF راح نستغني عن ال ID؟

Example:-
what is the size of BTB where
- k=10 (number of bits taken from addresses)
- R 64 VI Architecture.
- Instructions of 64 bits.

sizes $2^{10} * 64 * 64 * 2$
- 2^{10}: number of entries of BTB
- 64: lookup
- 64: Predicted
- 2: 2 bit-predictor

length of table = 2^k only.

Exceptions and Interrupts : Control flow Hazards

عشان اكون ماشي صح واحنا ضايفة اعمل flush control Hazard on flow

والشي اغلب الوقت بنتعامل بنفس الطريقة بالهاردوير

But Exceptions are → اشي مش كل بصير داخل ال CPU نفسه اثناء تنفيذ البرنامج

لو اغلب الاوقات بتكون مشكلة، واحيانا ال exception بكون اله علاقة بال OS

what kind of problems?

① problem in IF stage:

ال system صار في noise، ممكن لاعلية تقل ال inst من ال memory فلا هاد ال noise يعمل flip لـ bit فبخرب ال address يصير اشي مختلف

NPU : noise protection unit: اذا جبت على مكان ما حفظته بيعطيك انذار

يعني inst غلط مش الك، بسبب ال noise زي السبب الي حكيته

An exception in fetch stage ← it stops your code: "you are trying to access invalid memory location .... etc."

② problem in ID stage ← عشان يكون عندي ال opcode ال 7 bits الي بيستوعب ما يقدر 128 instruction

↓

undefined opcode

لكن احنا مصممين مثلاً arch فيها 80 inst, مثلاً

undefined

↓

ال assembler or compiler راح يعطينا ال opcode صحيح

↓

لكن لنفس الاسباب ال noise و حاجات فيزيائية عشان هذا ال opcode يصلك وينقلب لاشي ما يكون undefined ابدا

↓

The 80 is an example ← So in ID, this opcode is entering the control unit, that is designed with truth tables of logic to recognize only our 80 defined combinations, but now it's seeing something undefined, and it's not designed to recognize it.

↓

This rises an exception [ID exception]

/ /

③ problem in EX stage.

→ two operands
→ unsigned
] → not 2's complement representation → لما نجمع الاول والثاني بطلع الرقم اكبر من ال range (overflow)

we designed overflow circuit.

↓

take:

① the last carry of the addition operation
② the least sign bit of the result
] → inputs for XOR
1: overflow
0: no overflow

overflow: exception

These exceptions are disabled in many arch, we enable them manually. (Embedded controllers and processors)

④ Syscall: ... exception ...

When calling inst, commands, or functions of an operating system

In processors: two levels of execution

- privileged mode → OS, drivers
- Unprivileged mode → User mode (our codes and applications)

they can't communicate with HW components directly

we should have an intermediary component

which is OS

the instruction that tries to communicate with HW component, will give a Syscall → producing an exception that gives the handling of execution to OS to do the job and return results to you

And Interrupts are :→ dealing with input - output

flush circuit ← Re-fetch from new place

لما يصير interrupt شو بصير ؟!

بنا circuit ، حا مرة أول ما يصير input من اشي زي ال mouse مثلاً

هاي ال circuit بتبعت المسج لل processor اللي هو ال interrupt ، بتخليه يلجأ ال

inst اللي بين ايديه ، و بيادي ال insrutions تبعت كود ال mouse عشان

ينفذه

flush circuit ↔ Re-fetch from new place ←

/ /

Scenario | Handling Exceptions

offending inst: اللي سبب مشكلة

↓ causing an exception

interrupted inst:

causing an ~~ex~~ interrupt

↓

we keep their PC until we solve the interrupt or exception ← saved in EPC

↓ slide 48

The SCAUSE save problem indication

عشان بناءً عليه بدنا نشوف شو نسويها. هاي القيمة الي سببت المشكلة بحطها بال

register اط if-statement ، فاتنكرها وارجع انا لكود الماوس مثلاً،

او اروح احل المشكلة بالرجعة عنها لو كانت هي مشكلة.

↓

64-bits, not all are used; because we don't have 64 source of interrupts.

Way (1) # senario

Question: we want to get the mouse instruction, after we recieved the interrupt, can we go directly?

→ interrupt is recived,

→ we saved where to go back again (interrupted PC) in SEPC

→ saved the indication in SCAUSE

→ intending to go to the code of mouse, or to any code that can solve the problem if exists

how?

We don't know where are the addresses of instructions in memory (H/W does not), but the OS does so!!

So: ISR: interrupt service routine or
IHS: " handler subroutine or
IHF: " " function

This function where we are directed to, after finishing steps of SEPC, SCAUSE ← ال address ثابت

→ We go to handler, Automatically, this function is simply an if-statement code ← بتعمل تشيك عال bits اللي بتوضح ال SCAUSE ← وبناءً على ذلك بتروح تنادي ال solution instructions اللي ال OS سايبها بمكانه

/ /

* حليب بس لحل المشكلة بدنا نرجع لل PC اللي حزناه بال SEPC
لأنه تماماً ممكن ما يكون بالحل ال problem

بس هاي الحالة ما بيقدر يرجع ، بيعمل crash لل OS
و will throw an exception

could return? restartable Handler
could not return? termination happens.

Not All designs are alike

way (2) # scenario

بدل ما يروح address واحد عليه if-statement code ،
بتكون هاي المرة عاملة لكل واحد ال exceptions & interrupts
كل واحد اله address خاص فيه.

↓

no need to handler

faster solution ← vectored interrupts ← Hard coding ← Job of programmers

vectored interrupts → more complex hardware and higher cost

Hard coding → to handle exceptions.

Job of programmers → Not easy!

/ /

Exceptions in pipeline

$SEPC$ → بنخزن فيه حسب نوع ال instruction PC+4 أو PC لازم نخزن ال
↓ the next to the offending

مثلاً ① بيجيله invalid opcode ، فالمشكلة في ال instruction نفسه ، فبو ينادي ال handler ليحل المشكلة من هنا مثلا بس
يودي المشكلة لل OS وهو يدبر حاله ، بنهاي الحالة هو بخزن PC

② عند ال interrupts ، احنا بنكمل من ال next instruction لما نرجع ، فبنخزن ال PC+4

[Improvement on the Pipeline]

slide 55

① تحسين ال signals ←

- Signal IF flush on the mux before PC
- " ID " " " " in ID
- " EX " " " " muxes in EX
to decide for - WB signals
- MEM signals.

← بناءً على القرار اللي بطلع من ال control unit اول اذا flush

← مش كل مصدر ال control signals هو ال control unit ، مثلاً ال overflow

لما ادرس ال design ، بدي ارجع للمحاضرة 10 دقيقة 22 اسوي ال muxes
الجداد اللي انضافت و لهدف منها.

/ /

الاضافة الثانية :-

← لو صار عنا مشاكل بال stages اللي قبل ، راح نخزن ال wires ال SEPC SCAUSE
قيمهم لحد ما نوصل ال registers نستخدمهم با ال EX stage

الاضافة الثالثة :-

عنوان ال IHS بنضيفه input على ال mux اللي قبل ال PC

slide 55

Restartable exceptions — Go back to complete program

NonRestartable — termination

← با الديزاينات الحديثة ال SEPC SCAUSE بكونوا موجودين با كل stage
حتى ما نضيع cycles لنقدر نوصلهم زي ما رايا

Exceptions Example slide 57

Handler IHS
↓
one interrupt-vectored
↓
Handler code is if-statements

→ IHS code instructions
- sd .. why store?
↓ context switching
تغيير المحتوى
to save on stack
RF نحفظ القيم المخزنة با ال
با ال MEM عشان لما نرجع من محل ما وقف ما الا يا قيم ال inst القديمة اللي وقف
عندها صايرة

→ Handler

Store instructions

IF-statements

load instructions → to restore :-

First: ~~SEPC~~ old values

then: ~~SDESSEPC~~

SEPC لأنه مجرد ما خلصنا البدأنا بال ← هيشغل الكود القديم

← وطالما اكلت المشكلة، بدنا نرجع ال Flipped-bit بال SCAUSE من 1 إلى 0، عشان لو جاي exception جديد ويدور ال OS يفحص ال SCAUSE هيلاقي بيشوف قيمة مرجع، ما بيشوف لمشكلة قديمة وبرجع مكان غلط.

← بالخطوة اللي بعرف فيها مصدر مشكلتي - بعمل clear Flags لـ SCAUSE

* we'll lose three cycles in this example

Multiple instructions

Question: Could we have more than one exception?

yes, it could happen, but if so, multiple exceptions won't be in the same stage.

[slide 61] lecture 10 / min 32

- add instruction have exception in EX stage / cycle 3
- I3 (bad): invalid opcode. (will be detected in ID) — opcode الخاص بالـ I3 تم بطريقة غلط

↓

* 2 exceptions in the same time / same cycle

↓

to solve them, they are ordered depending on priority in modern designs

- exceptions have higher priority than interrupts

↓

لأنها بتضر بالـ CPU

دائماً الأولوية لذي شيء يضر بالـ CPU

← لسبب إنه exceptions الـ built-in أولى من الـ interrupts اضبط

أولويات بالـ software ← programable priority ← او بتضبط أولويات عن طريق الـ اسلاك وتشبيكهم

/ /

ـ في ديزاين ال RISC-V ، بتحل ال priorities في ال software

→ Solution is **precise**

they happened in the same time

but you solve the older in the sequence of code

(older have higher priority here)

the operation

→ fill SCAUSE with add bit

→ fill SEPC with PC or PC+4

→ IHS solves add exception

→ return to I3

→ I3 will get stuck again in ID

→ exception detected.

→ fill SCAUSE + SEPC

→ jump to IHS to solve exception

High Cost solution + Overhead

→ Solution Imprecise for imprecise exceptions

لأ مش ضروري يحافظ على الترتيب

علشان بنحل نتج ال opcode قبل ال add

→ they both Put SCAUSE and put 1 in it

→ Software (if statements) decide from where to start solving (software priority)

ـ وهاي الطريقة ما بتنفع للكومبيوتر العادي

Handler

Topic !!

Parallelism via Instructions

the design of pipelining and parallelism through stages is called [ILP]

The multi-cycle ←

5-stage pipeline

↓ instruction-level parallelism

↓

clocking depending on the time of least-time stage for all clocks → هيك الوقت قل الFrequency زاد

$time = \frac{1}{F}$ and vice versa

The new solution to increase performance:

The Multiple Issue.

for example:

→ The fetch stage is giving you more instructions [not only one instruction at a time]

↓

كأني عامل copies من ال pipeline بوديهم كل واحدة ع نسخة (نظرياً)

لكن حقيقةً احنا بنعمل replication لبعض ال components وبرضو شغلات ثابتة فقط يتم التعديل عليها واضافة اسلاك وماشابه لتحقيق هذا الغرض

/ /

→ * Ideally: in the normal situation, where no stalls, no problems (only sequence of instructions)

in the 5-stage pipe-line: each cycle you finish one inst

CPI = 1 ←

ومرة بتطلع من ال WB
ومرة بتدخل ال IF يعني بالدخلة

→ IPC = 1

in multiple issue: 2 instructions enters, 2 exits (for example)

$CPI = \frac{1}{2}$ → نطلع 2 وبندخل 2

جزء من ثرجة البرنامج

to avoid using fractions we use new unit called IPC: instruction per cycle, not cycle per instruction (المطلوب ✓)

IPC = 2

check the example in slide 65

peak = ideal

Problem in multiple issue: dependencies!!

two instructions have true dependencies in between, are going through stages together, while you can't forward, because they are the same place

Scanned with CamScanner

increasing the depth of pipeline (more stages/ sub stages) is not that best solution

↓

increase the width (multiple issue)

Multiple Issue

① Static multiple issue

for simpler computers (few power use)
+for cores of not high end computing ←
for things like videos applications

المسؤولية هون على الكومبايلر والمبرمج بانه ينسق الكود كاملة

↓

This is limiting factor, let it be more dynamic (by CPU)

② Dynamic Multiple issue

grouping and all these things, let them be the job of CPU

بخلي ال instructions يجو الكم، وانا كـ HW بصير أنفذ منهم، و باستخدم ال control signals بصير اتحكم بال dependencies وال hazards وانسق ال instructions بشاتهم وأحكم بال hardware

But compiler helps a little bit

Static Multiple Issue design

compile → instructions → packets

(Grouping)

the mix of instruction types in packets depends on the design, you can't put any instructions together with no restrictions

* → بداية ال static لما عملوها بال 70's سموها VLIW

↓

ما بتفرق كتير عن static بس حطوا ال concept ، بس سموها very long لانهم كانوا يجيبوا مجموعة inst ويعاملوهم كأنهم inst وحدة بتقوم بعدة عمليات بنفس الوقت

↓

multiple opcodes

① In VLIW designs, instructions are saved in external memory like DRAM or flash memory in embedded system, or in the CPU cache

② we have Execution Units (EU), multiple of them, ~~enog~~ enough to deal with the big number of instructions, and the absence of any one of them causes a structured hazard.

design of slide 67 is containing integer register file

③ we can't replicate register file. ← لأنه بصير بـ أكتر من X0 و فكان مختلف وكتر من X1 ، ، ، وهكذا لكل الـ X

↓

but we have to ~~[redacted]~~ modify it some way to be able to deal with the big number of instructions

because:

- we only have 2 read ports - 2 output ports
- port for the input coming from WB stage (write port)

But for 1024 bit (32 instruction) at one time, this is problem because the sequential access of instructions on RF

→ increase the number of ports ~~[redacted]~~ حيث تكون ثانية هي ال inst اذا احتاجت تقرأ من ال RF in parallel

/ /

→ we do padding when no operation can be added. → (giving all value 0)

↓

ضروري لأنه بالنهاية إلها دور

عشان الـ signals ضروري تكونوا

يا 0 يا 1 الـ bits ما فيش اشي

اسمه فاضي

We have Dual (2) issue, ~~trip~~ tripple (3) issue, Quad (4) issue ...

In slide 70

how we determine the address of first PC of the packet. (PC+X)

it depends on the arch:

- length of the instruction
- number of the instructions per packet

RISC-V with Static Issue

↓

The 5-stage pipeline

one of the most important targets:-
- to have enough resources in EX stage to deal with dual issue.

Slide 71:

First added unit to the design.

EX stage:

we leave the First ALU with its muxes to calculate for ALU instructions & branches.

But, we add additional ALU to calculate for LD/SD addresses.

No Structural Hazards

Second added unit

add new Imm Gen unit to decode stage

لأنه ال imm بكونه مختلف بال inst في bit 12 بس لازم نتعلم
LD/SD inst. وال ALU inst
extension ، وهاد كشي مختلف اصلاً ال

for example:-

ADD x1, x2, 45

LD x5, 8(x16)

Third Addition

Duplicate ports of RF in ID stage

لاحظنا بالسلايد 71، إنه ال PC بيضيف 4 على ALU لحساب ال next PC، بس هون عنا بال dual issue ال 32-bit inst
ال next packet بيلحق ب inst تبع 8 bytes، فلازم نحط 8 مكان 4، لأنه بنوصل تنين تنين اصلا.

Hazards with Dual-Issue RISC-V

① true data dependency, forward to next packet

② load-use dependency, forward to the packet after the next one

→ wires and units of forwarding should be replicated because we might be intending to forward to an instruction in the same packet, and for an instruction in the next packet.

Scheduling Example // Slide 78

- points to keep in mind
  ① compiler will take the role of spreading instructions to packets. (Static design)

Explanation of code:

X20 : the base address of our array is saved in this register
→ take offset 0 on this address, go to it, in memory, and load the element stored in that location

X31 : register containing array element.

X21 : any scalar

line 4 (decrementing pointer), to update the target address, so the next time you want to ld something from memory, you find address in X20 updated

→ we are updating the address using (-8) shift because we are dealing with long long values (8 bytes)

← سؤال : واحنا عم نحكيله دايماً نرجع address معين ، هو ليش لازم عشان اي byte بالعادي لازم نوقف بحيث يكون ال leading 8 bytes ميزه ؟

some analysis

- two instructions → load / store
- 3 instructions → Arithmetic / branch

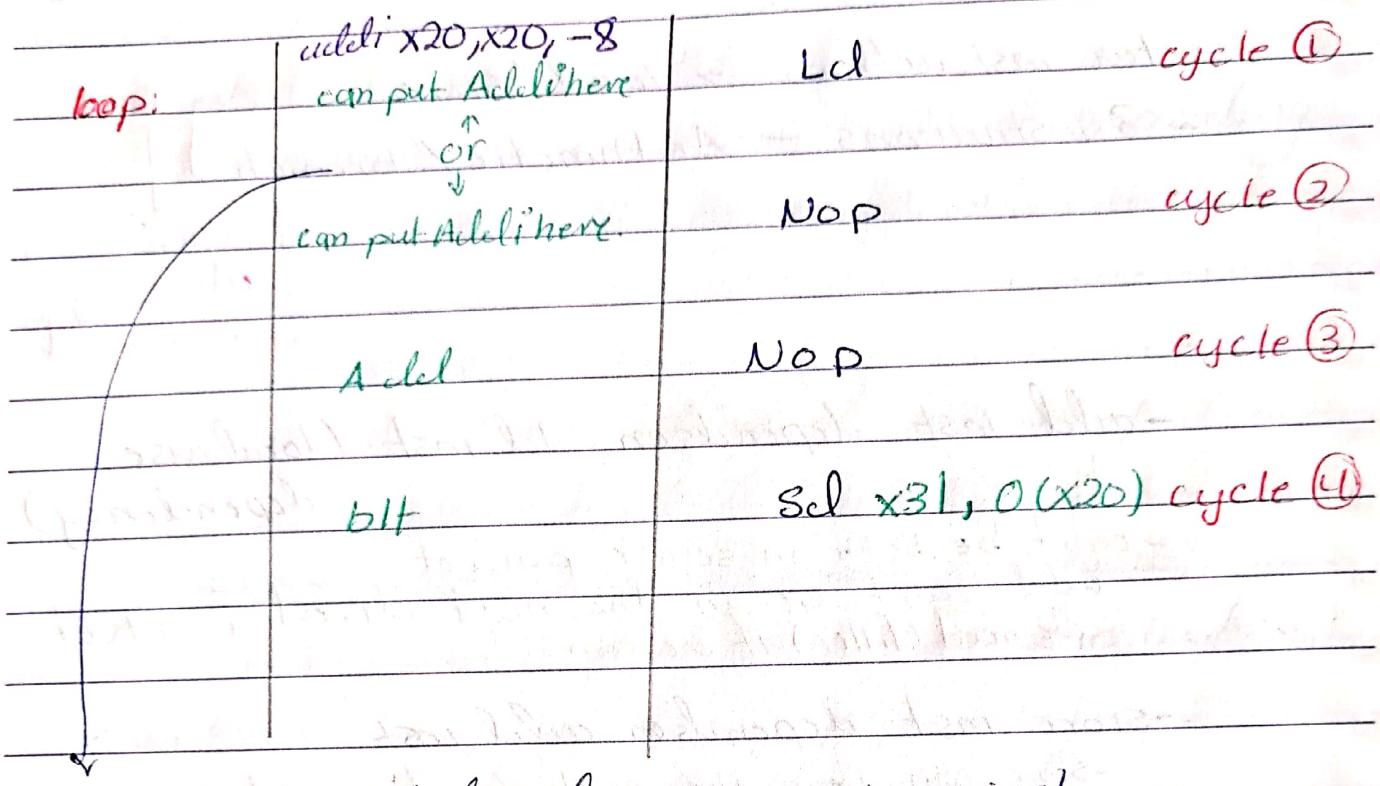
already imbalanced

- add inst depends on ld inst (load use dependency)
  - can't be both in same packet
  - add can't be in the next direct packet
    we need (filler) if we can.

- store inst depends on add inst
  - store can be in the next direct packet
    forwarding Unit can solve the problem

- blt depends on addi
  - each in separate packet.

First scheduling (naive)



addi does not depend on any previous inst, so we can put it on any packet before the packet of ~~and~~ add inst.

Mistake!! problem in solution

We've changed the sequence of code, the flow have been changed (logical error)

- original correct sequence!
- load element from address x20
- edit element value in x31
- save value x31 in the same current address saved in x20 (the old address)
- change the address in x20 to the next target one

/ /

- ~~the~~ new flow of sequence
- we put addi before sd inst, so that we update the address of x20 to let sd save x31 value in the next target address not the old address that we were working on through this operation.

الحل: بدنا نلغي تأثير التعديل المسبق على ال address اللي بدها تخزن عليه ال sd

بعد التعديل: new update on code:

وهيك ما بختل ← addi x20, x20, -8 → نقص من ال 8 address وخزنه بال x20

زي ما هو ولازم دخول طبيعي ← sd x31, 8(x20)

احنا بال sd ضفنا 8 عشان نلغي تأثير ال -8
بدون ما نرجع نغير القيمة اللي خزنتها بال x20

Hint: if you put the instruction of updating the address earlier, reflict the effect! because we basically update the address after we finish using the old address

IPC? $IPC = \frac{\text{number of actual instruction filled in packets}}{\text{cycles used for packets}}$

$= \frac{5}{4} = 1.25$ (peak = 2)

min 13

How to solve the problem of law IPC?

why this law IPC?
- filling with Nop
- the depecdency (but not as a direct reason)

لانه لما يكون طويل الكود، قادرين نستخدم inst تا بين نعبي فيه الفراغ الناجم عن ال depend → كيف ممكن نستخدم هاي الطريقة لحل المشكلة ؟

Solution: increase the number of instructions!

work on your code to expose more instructions on assembly!

↓

Loop Unrolling

compiler's job

Example:-

* the rolled loop

```
for (int i = 0; i<10 ; i++) {
    a[i] = a[i] + x;
}
```

* the unrolled loop

```
for (int i=0 ; i<10 ; i=i+4) {
        a[i] = a[i] + x;
        a[i+1] = a[i+1] + x;
        a[i+2] = a[i+2] + x;
        a[i+3] = a[i+3] + x;
}
```

→ This way: reduces loop-control overhead

loop-control: (i<10 ; i=i+4) → update i, → compare

* in rolled code: 10 times running code
11 times implementing
control (10 upgrades, 11 comparison)

* in unrolled code: 4 times upgrading i
5 " comparing i<4

← الكومبايلر بيعمل unrolling لكل الـ iterations، بس عند آخر ثنتين، بعالجهم كل وحدة لحال، عشان ما يوقع بال outOfBound ←

This operation is done in assembly not C!

Steps of Unrolling in Assembly! Example slide 82!

① Replicate 4 times! copy/paste

```
ld  x31 ,0(x20)
addl x31 ---
sd  x31 ---
```
→ a[i] = a[i] + x;

```
addi x20 ---
blt  x22 ---
```
i++, i<10 → this part is not needed in the first 3 replicas

we only want to change i value after 4th copy, and to compare there!

↓

② Remove unneeded loop overhead!

Still having some problems

4 copies are still identical instructions!

they all do a[i] = a[i] + x;

while we want them to work on 4 different elements of array [illegible]

that a[i] [illegible] precedes! play on offset!

/ /

③ Modify instructions
* modify offsets in this case!
-8, -16, -24, -32

problem!! we are only working on x31!!
- logically : no problem
- in case of schedulling : all your packets will depend on x31, which causes dependencies problems

← التوزيع عال packets مش حيكون sequential
مراح يسبب dependencies

④ Rename registers! (complete renaming)
- give each group different register name

شو حد اللي مخليني اعمل renaming ؟ هل بقدر اعمل 64 unrollings ؟!
واحنا عنا اصلاً registers محجوزين لأغراض خاصة، فعنا limit
عال renaming مش براحتي! عشان نعمل عادي جيبس الكومبايلر عشان يطور ب
dependencies ؛ بيعتمد عال data path عنا 8 inst مع بعض، ولا
2 inst مع بعض
← 2 inst ← بقدر يحط اعداد كبيرة، وكل ما يكون wider بصير أصعب

/ /

⑤ Scheduled instruction:

example continuing slide 90:

* take into account these things:

- we have load use dependency in each group between ld → add

filler packet. ← add لازم تفصل ال ld عن ال

- true data dependency between add, sd in each group (no need for a filler, just put each one in a packet)

dummy solution:

← ملاحظة: ال sd لازم تكون بعد ال add بس مش ضروري تكون directly in the next packet.

- we can change this to win some cycles

	ld x28	cycle ①
	ld x29	cycle ②
add x28	ld x30	" ③
add x29	sd x28	" ④
add x30	sd x29	" ⑤
	sd x30	" ⑥
	ld x31	" ⑦
addi x20		" ⑧
add x31		" ⑨
blt x22	sd x31	" ⑩

(load-use dep: ld x28 → add x28; true-data dep: add x28 → sd x28)

Scanned with CamScanner

Optimized solution :

cycle 1
" 2
" 3
" 4
solved in slide 91 ← " 5
" 6
But !! " 7
a problem " 8
← لاحظنا ال addi يا اول packet " 9
ما حسبنا حساب اينه هي بتعدل عل x20 " 10
الي كل ال ld و sd بتقرؤوا منها .

نلاحظ بال ← offset تبعت كل ld و sd
حينا مطرح اذا حطيت المعدل يخ ال addi
الي تبعته (-32) بنزيد ال offset بالماضي
packet مش ح تتأثر ✓

ملاحظنا نفس الشي لـ inst ↓ ld لاول x28
لنحطه ال packet
addi نز م نفس ال

$-32 + y = -8 \rightarrow y = 24$ inst : ld x24
$-32 + y = -16 \rightarrow y = 16$ inst : ld x30
$-32 + y = -24 \rightarrow y = 8$ " ld x31
$-32 + y = 0 \rightarrow y = 32$ " sd x28
$-32 + y = -8 \rightarrow y = 24$ " sd x29
$-32 + y = -16 \rightarrow y = 16$ " sd x30
$-32 + y = -24 \rightarrow y = 8$ " sd x31

Static scheduling ends here.

Scanned with CamScanner

✱ Back Deeper in Dynamic Multiple Issue

"superscalar" : more than one instruction entering at one time (this is done by scaling the number of instructions entering a stage / we are scaling it.

→ Avoids the need for compiler scheduling, but it may still help us in operations like: ~~toole~~ loop unrolling and exposing parallelism.

code semantics ensured by CPU

↓

الاشياء اللي اي
انه نتأكد ان
offsets
scheduling
اذا صح ولا لا
نعدل حتى الكود
يكون logically

this is the job of CPU now

To understand marketing sheet, Intel ARM, AMD, RISC-V, MIPS web sites (e.g. words like: (superscalar, out of order, in order) processors will be used

superscalar: Dynamic issue, multiple instructions being fetched, and handling + managing them is CPU's job

Out of order: our CPU, can do execution for all instructions coming from IF, ID stages, in orders that never relate to the original order these instructions came within

- Out of order execution
- in order fetch → we all return to something called (In order commit)

out of order expression only relates to execution stage ✓ Some thing have to be done to ensure logicality after this stage !!!

What we mean by Out of order ?!

سلايد 93 (مثال): بدي أسمح لل instructions بالصورة تشتغل instructions بينما تجهز نا لل

→ can start sub while add is waiting for lcl

حتى بصير احاوظ على logical results

معنى آخر، ستوتحل ؟!

- بالوقت ال processor بنادي ال sub وال add قبل ال lcl وال local-use dependency

كل هذول عالجهم dependency

- علشان ال sub خطها بين lcl وال add عشان يكل ال dependency بينهم

↓

كل هذول حقدة بسيطة على اشكال ال out of order

Dynamically Scheduled CPU (design slide 94)

→ As major stages, major view on design, all processors of this type have some thing similar to it, not forgetting that any stage in any design can be divided to many sub-stages.

→ (1) Stage (1): ID + IF stages are combined in one stage (Instruction fetch and decode unit) → control unit
→ loading instructions.
↓ depending on each inst type we'll spread instructions each to the suitable unit (available execution units)

(2) Stage (2): EX
functional units ↓
number of units and their types are different from design to the other
processor

Stage : Issue
Reservation Stations
- issuing instructions is done here
- like a buffer (they are registers) to let instructions wait until the units of executions become free to work on new inst, to avoid structured hazards of not enough resources.

- even if EX units are free, we may leave inst. in reservation units to wait until some problems of dependencies are solved.

reservation stations ← اصير بالحياز الأساسية ال operands اللي ع ال هوال RF اللي موجود ع ال IF/ID stage.

↓

if the inst. came and there is no dependency, it reads registers operands and puts them on the beginning of available units for this instruction is ready

→ بجهز شيئا هل بال كل ناصي ع ال EX units ولا لا

scenario: Reservation station

نفرض إنه دخلنا 4 inst عال RS الأولى، و 4 inst عال RS الثانية، والثانية خلصت ال EX تجويفها وهسا بقطعة ال integer unit المقابلة ال RS الأولى. سؤال: هل بقدر أنقل ال inst المعلقة بال RS الأولى، أوديها عالثانية اللي فضيت؟!

RS 1 + RS 2 : two RS in the left of slide 44, opposite to integer units.

اذا بدنا نغير فيه هاي المكانية، بنصفح RS وحدة بينهم الثنين، بس هاد الحكي مش موجود، فالديزاين تبعنا independent RS's وكل وحدة خاصة بال EX units المقابلين الها.

* the line that ~~connects~~ connects all execution units is called CDB (common data bus)
it is a shared bus!

يعني لو كان في 2 EX units طلعوا نواتج في نفس اللحظة ، اذا حطوا ، لنفترض نواتجهم عال CDB بنفس اللحظة بصير
ال bus وبصير short circuit

مقاومة التيار أكبر إشي

شو الحل ؟! بيكون في هاي الخطوط شي اسمه High impedance
لكل واحدة منهم ، ال controller تبع ال bus بلقط منها اذا
بدها تحط شي عال bus ، وغيرهم كلهم High impedance الا هي بتكون active
فتطلع نواتجها وتكمل ال path تبعها ، بيفحش يكونوا كلهم عبارة عن
0 و 1 ، فمش هيصير short circuit اصلا ، لازم يكونوا اشي High impedance
واشي active

← يا اللحظة اللي بكون فيها ناتج ال units ، والا باخد نتيجة
وحدة منها وبمررها لوحدة من ال inst اللي بتستنا يا وحدة من ال
RS ، هاد اللي اسمه forwarding

when is the soonest time to forward these results? ← باللحظة اللي بطلع فيها النتيجة عال CDB ، وهي عليه
بسحب منها حطا من CDB نفسه لل RS ، عشان ما اوفر stalls
واعمل forwarding

/ /

ما عادت هناك إشكالية إنه ال results على ال CDB ترتيبها حسب ترتيب ال inst لما دخلوا!

↓

all inst enter

↓

ready inst start to execute

↓

CPU is controlling and choosing

↓

no guarantees on maintaining the original order

↓

Because of this, at the beginning of commit unit there is a stage called ROB

*ROB stage: re-order buffer → بترجع الترتيب للنتائج لترتيب ال inst الأصلي

↓

How the ROB know the correct sequence?!

↓

هو بيقرأ نتائج، مش بده يفهم هاي النتائج شو مصدرها وكيف بده يعرف ترتيبها؟!

بعيداً عن ال ROB احنا بنعرف ترتيب ال inst من ال PC، وال PC بجيب ال inst من ال inst memory الي فيها مخزنة ال inst بالترتيب الأصلي، بنكون معها

→ Reservation stations + other circuits do the job of tracking the right sequence of inst. depending on their PC is

/ /

Single Issue Example slide 96

↓

بنا نفوت عليه خطوات ال
multiple issue.

Code Analysis :

- load- use dependency → ld → add.
- true data dependency → sub → andi

Single Issue ?! only one instruction

- EX stage sometimes limits the number of instructions

← يعني ممكن يكون نظام كامل مثلاً Quad ، لكن ال EX
ما بتحمل الا 2 instructions

min 41 ✓ lec 13

/ /

Words Recovery

Out of Order : describes EX stage

[ممكن يدخل أكثر من inst قبل ال inst الي لازم يخلص
مكلم دخلص لكن كله بالنهاية ع ترتيب Commited by order

↓

C stage is sequentialed → ال C لل inst التالية
ممنوع تكون قبل ال C لل inst
الي قبلها ، يا إما بنفس ال cycle
يا اما بالي بعدها

~~Renaming~~

Register renaming is replaced Slide 97

لو بال Dynamic ، المشكلة هاي انحلت بال
reservation stations وال ROB
[على سبيل المثال : هنا inst واقفة بال RS ، دخلو بيستنا
يجي قيمة ال operand تبعه سواء من ال RF او CDB او
ال Reorder Buffer (ROB)
← باللحظة الي بياخذ فيها القيمة من احدى هاي المصادر
ما عاد يفرق لو في inst ثانية تشتغل عال Registers
الي حاولة هاد القيم ، لأنه خلص صار اليه حق ، مش راح
يأثر على تعديل عليها من ال registers تبعتها

← هاردوير ال dynamic بسمحلنا نستخدم ال registers هيك
بدون ما نحتاج لل renaming
هو مش موجود بشكل explicit زي ال compiler ، بس ضمنيًا موجود بال HW

Recovery :-

RAW (true data dependency)
this dependency can exist for any type of design.

But the WAR (Name dependence)
WAW (Output dependence)
↓
they may only happen in Multiple issue out-of order pipelines

* Name dependence: WAR
should: Write After Read.

Code analysis Slide 98
- true data dependency → mul → addl.
waiting for x1 to get ready.

* ld does not depend on addl, but it will write on x5 before addl reads the correct value of x5 → in 5-stage pipeline this wasn't a problem

In dynamic processor (Superscalar) deg ≥ 3
① mul and ld are ready to move from RS to EX units.
② mul is a long instruction, it will take time to finish and find x1 which is the second operand of (addl), hence, addl will stay more time in RS
③ through this time ld will write on x5 before addl reads the correct value of it.

dependency

This problem will be solved automatically by the in order commit.

→ in order fetch
→ out of order EX
→ in order commit.

* Output dependence (WAW)

ال dependence فيها عـــوريّة.

فالوضع الطبيعي ما في مشكلة لأنهم بيجوا وراء بعض

* Because of multiple issue and out of order
① ld has finished execution earlier than mul, then the ~~latest~~ last value of mul will be from mul not from ld, and this is wrong logically

It will be solved by In order C

remember: W stage in Dynamic is the stage were results are written on CDB

The In order Commit + Reorder Buffer can solve these problems.

Scanned with CamScanner

Speculation (Guessing)

→ Branch ✓
→ load ✓

→ Remember: prediction implemetation is done in F stage

→ resolving prediction: depends on design
5-stage pipeline بال ← — it may be in M
ID حطيناها بال الموضوع ال — " " " " EX
— " " " " ID

checking if prediction is correct.

→ In superscalar: resolving branch is done in EX stage, because there we implement the subtraction between the two operands of branch insl.

→ In 5-stage pipeline, flushing instructions was easy, but in multip-issue, it is not.

What's the solution ?!

ما منح ل I3 و I4 ب تسليم ال ROS انو يكتبو نتائجهم عال RF او ال M خليهم تضل حاطه

let the control unit in EX stage of branch to give (nullify), to do not allow commit stage to work for fall-through, instead of C, → nullify
↓ signals of next stages set to 0

In other words:

- MEMRead control signal: 0
- RegWrite " " : 0

after Nullify:

- fetch new instructions directly after the cycle we determine that the prediction is correct or not, not after nullifying.

load speculation

load problem (Pointer Aliasing)

it happens most times in programs full of pointers. ✓

example:- slicle 106

- 3 inst are going through F, I...

(deg 3) → ld
→ sd
→ ld.

- first ld is done peacefully
- sd is waiting for x1 to get ready from first ld, but second ld is not waiting for anything.
- second sd waits in RS, while ld is about to finish execution

But ld will read from the same address the sd should write a new value in [ld should use the updated value, but this not possible in this situation]

Speculation and Exceptions

Exception Example slide 111

What is the exception of C?

Remember: we may have these exceptions
F stage: out of range address (when loading instructions)
D stage: invalid opcode
E stage: overflow

C ?? → it saves on RF and Memory
then exceptions are from type: out of range access.

after detecting the exception in cycle 7
- soonest time we can move to exception handling code is cycle 8 (perfected design)

- first couple of instructions to fetched ↑ exception handling code are gonna be of type (scl) because we should store our values of previous instructions to keep track on our program. (context switching procedure when moving from code to another, save previous registers)

Why Do Dynamic Scheduling? slide #12

→ Remember: we have a lot of ISAs
↓
with different latencies + Hazards.

ISA (1): X32 or X32-64
by Intel / AMD } Both can run windows + Both can run same programs
↓
because of same ISA

ISA (2): ARM V7 (ARM company)

ISA (3): RISC-V

ISA (4): MIPS

ISA (5): PIC

-Why not just let the compiler schedule code?

Answer:

-take into account that even one company can produce more than one generation and design based on one ISA.

Intel: -Pentium
- Core
- Core i3/i5/i7/i9
- XEON
- Silver/Bronze/Gold

→ To let the compiler schedule instructions into Issue packets, you should provide it with enough knowledge about all these designs. عشان هيك ال scheduling وال unrolling

كتابة كومبايلرز بهاي الصناعة هو شي صعب جداً

-Not all stalls (caused by hazards) are predicable by compiler (can be handled by compiler)

cache misses can't be handled by compiler because it does not know what exists in registers which is not helping to handle this situation.

- for modern CPUs (cache included in the design) : High-end designs.
  - instructions are in instruction cache
  - data ~~is~~ is in data cache

- Not modern CPU's
  - main memory (DRAM or Flash)

always the memory closer to CPU

- Can't always schedule around branches.
  - compiler does know if the branch is taken or not., this is detected at run time only.

- Delays, limited bandwidth of the memories (memory delays), can't fetch fully parallel instructions together, all functional units are waiting for ready instructions to be executed, it is hard to keep your pipelines full all the time.

We are better than before, but not as wanted to be.

Chapter 5: Memory

Computer System: - CPU and
- Buses
- Memory subsystem

- We assumed in previous lectures, M stage takes only one cycle, which is not realistic

Assume that:
our design is about: - CPU
- RAM only (no caches)

↓ M stage will take from tens to hundreds of cycles.

→ Our DRAM is a middle way to transport data to CPU instead of slow Hard disk where M could take millions of cycles.
→ CPU speed = 4 GHz (assumption)
→ hard disk at highest case: 7200 round per second (Mechanical)
Huge difference!!!

But while using DRAM, it is still bad with 100's of cycles of M → Solution: cache between CPU, DRAM
holds data

/ /

٣- وهون لاحظ انه ما اشتغلنا هو داتا بالكاش cache الا اللازم حتى نقلل روحات وجيات على ال RAM

أكيد الا ما رح نرجع نزيد ال Frequency حبس برضا نقلل ال DRAM

Principle of Locality ← من local (محلي)

Temporal → من كلمة time

Spacial → من space

← [time + space] الابعاد الي بنحكي عنها

Example :-

```
int x = 0;
for (i=0; i<1000; i++) {
    x++;
    a[i] = a[i] + 5;
}
```

سؤال: شو ال variables الي كل لفة قاعد ال loop نشتغل عليهم؟

i و x

انا قاعد بشتغل عليهم كل لفة ، وكل لفة ولفة بينهم فوارق زمنية قصيرة!
→ each one of them has temporal locality (adjacency in time)

رجعنا عن حكينا عونيه ، كل مرة بنشتغل ع نفس ال variables

/ /

But: a[i] ~~don't hav~~ does not have temporal locality [each iteration we work on different element of the array]

a[i]
a[i+1]
a[i+2]

This is adjacency in space [spacial locality]

each so it does not have a temporal locality, even though the time between two iterations is short, but it is not the same actual element But!

لفكرة من ال locality انه ندور عال variables اللي كل
لمه انا بشتغل عليها وحطها بال cache (هي اول خيار
لنحط لدا يا ال cache (Temporal locality

من جهة اخرى، اذا انا شغال على a[i]، منطقيا بدي اشتغل على
a[i+1] كمان، اغلب ال access لل array بالبرامج هيك، فال block
اللي بدنا نجيبها من ال memory تحتوي عال a[i] ومجموعة كاملة
من ال variables اللي جنبها، على أمل انه لبرنامج بده يعمل عليهم كلهم
~~Ep~~ access، وبدل ما اضل اجيب 1000 عنصر من ال array، فانا
بجيب chuncks مجزأة، فبدل ما اروح عال DRAM 1000 مرة،
بروح 30 مرة مثلاً Using spacial locality

We are putting the recently accessed and the nearby ~~ev~~ variables in cach

/ /

But: a[i] ~~don't have~~ does not have temporal locality [each iteration we work on different element of the array]

a[i]
a[i+1]
a[i+2]

This is adjacency in space [spacial locality]

each so it does not have a temporal locality, even though the time between two iterations is short, but it is not the same actual element But!

فكرة من ال locality انه بدور عال variables اللي كل
مرة انا بشتغل عليها ونحطها بال cache (هي اول خيار)
نحطها دايماً بال cache (Temporal locality

من جهة اخرى ، اذا انا شغال على a[i] ، منطقياً بدي اشتغل على
a[i+1] كمان ، اغلب ال array access بالبرامج هيك ، فال block
اللي بدنا نجيبها من ال memory بتحتوي عال a[i] ومجموعة كاملة
من ال variables اللي جنبها ، على أمل انه البرنامج بدو يعمل عليهم كلهم
access ، وبدل ما اجيب 1000 عنصر من ال array ، عشان
بجيب chunks صغيرة ، فبدل ما اروح عال DRAM 1000 مرة ،
بروح 30 مرة مثلاً Using spacial locality

we are putting the recently accessed and the nearby variables in cach

/ /

كيف بيترتب ال arrays جوا ال memory :

① ← raw major ابتخزن صف صف Java, C

② ← column major " عمود عمود matlab,

↓

بتفرق لغة البرمجة

- الهدف من مبدأ ال memory hierarchy انه to hide
البطئ الشديد لل harddisks

In slide 5:

local Storage (flash) → like USB sticks

↓

hard disk

↓

local Storage (Magnetic) HDD

But flash is SSD

Question: if we invented a technology that non-volatile memory, cheap, and can response to all load and store instructions as much as they need responses within one nano second

it will replace the HDD or SSD, and caches, DRAM and we remove them to put more cores.

But RF won't be replaced (it is the core of CPU)

Example

If 100 memory locations are loaded to cache, But the CPU could find 60 of them

Hit ratio (rate) = $\frac{60}{100}$

Miss ratio (rate) = $\frac{40}{100}$

what is the meaning of hit and miss?

we loaded a[i], because of spacial locality, a block of 16 elements of array a is loaded, when the CPU tried to access location 17 of this array, it could not hit it in cache (miss occured), so, it went to DRAM and loaded another block of 16 elements, where location 17 is the first element of it. and this takes tens of cycles.

/ /

Section 2.11 Memory technologies

SRAM technology

ال transistor هو عبارة عن switch يتحكم بمرور قيمة او يوقفها يا بمرر قيمة ، يا بوقفها

it passes the current or blocks it

مش بالتصميم الجديد بذكي ، همموها على اساس انه بس يفتح عليها current او يسكر ، بتضل محتفظة بال voltage تبعها طالما ثبتها ، بتحافظ على قيمة ال 1 وال 0 كما هي ... bloop

- Fast but not dense

$1*10^6$ transistors → 6 for each bit

$\frac{1*10^6}{6}$ = about $166*10^3$ bit → not dense

in embedded systems we put the instructions in flash memory, and data on SRAM memory → you can find chipsets of SRAM technology which you can interface them to CPU.

→ on the chipset of slide 4.

Example: Our SRAM size is 64K bit, how many chips you need? 4 chips.

64K / 16K = 4 chipsets

Scanned with CamScanner

/ /

the $\overline{CS}$ is connected to a decoder which decodes the address, and depending on that we choose which chip set to activate.

- output of decoder is a binary number where only 1 bit is 1, and the rest are zeros

In five-stage pipeline, in stage M: there was a control signal named Memwrite, this signal is connected to $\overline{WE}$ in SRAM

DRAM technology

it saves data on capacitors not transistors to save data permanently

← بشتغل ، بعطيه شحنة ، فبيشحن الـcapacitor لـ level معين

← اذا عطيته عليه 0 ، بفرغ

Problem (electrical)

the capacitor will discharge overtime, so it is not able to return our information

the solution is to read all saved data again (capacitors) periodicly, because this lets capacitor charge again so it keeps saved value. (refresh cycles)

- The refreshement makes the controller complicated, now we need more circuits to lead this operation of refreshment
now the controller reads (where to), writes (where to), & refreshes (How and when to)

A problem on refreshment:

conflict between refresh cycle and read instruction on DRAM. (when they are going to happen in the same moment.)

→ Designs have different points on giving the higher priority to one of them to be done first.

DRAM access sequence:

→ address of load or store instruction is analyzed by the controller inside the DRAM

→ after analyzing the address it is gonna be divided into two parts, a part to identify row, and one to identify column.

example:

lw x1, 8(x2)

address in memory = 8+x2 + (x2)

this address is put on bus to reach controller

→ address parts are saved in row address latch and column address latch, to save them from being lost because of the next coming addresses

Chipsets of DRAM are installed on PCB

printed circuit Board

هي لقطعة اللي تركب عليها لقطع تقريبا زي ال modem ال memory ال وغيرهم

silk screen اسمها ال PCB ← الكتابات اللي لونها ابيض على ال

on mother board.

PCB, chipsets on it, controller in the middle of chipsets

they have pins

bus on the mother board (north bridge / south bridge)

Black chipsets ICs ← mother board

they are the bridges
north
south

2 in usual, in intel

they manage bus access (buses between the components: CPU, memory ...)

/ /

Big image: bus, memory, CPU ✓

↓

← ال CPU بيجي بيعمل ld / sd

← بروح على ال cache، لو لقي ال address، بجيب hit

← ما لقي الداتا ← miss

↓

the controller starts working ← sends a request to memory to get the target address

↓

being put on bus

it reaches the controller → حسب التصميم ال adress lines are 64 lines or 32 lines

32 bit based design is not enough to install more than 2^{32} addresses / 4GB RAM on computer which is not enough, so we moved to 64 bits based design

بس فعلياً احنا لسا ما وصلنا انه نستخدم ال 64 كاملين، ف physically بيحطوا 48 lines حتى يخففوا signals ما في داعي.

Now, the controller divides the address into two parts (row, column).

/ /

Basically, why we divide the address?

Because the data is stored in memory as rows and columns (tables)

* Back to RAS and CAS

→ after the controller divides the address (row, column)

now we want to save them

→ give a signal named (RAS), to save (Row address in latch) then pass it to a decoder, to identify the row

(→ this operation takes time (TRAS: time to assert RAS (to latch and decode))

when this stops:

→ CAS starts (TCAS)

↓

بهاي الطريقة بنوفر، حاولوا (هي) لته
بما ظار منعدنا وحدة.

column addr يدخل عال decoder

↓

then the data is put on data bus to go to CPU

/ /

slide 12//

problem !!

احنا بكون عنا array وكلها موجودة في نفس ال raw ، لسى كل مرة دخل لحد

نختار ال raw من جديد ، هاد redundant work وهاد ده low Bandwidth

↓

big difference in time to put ready data on bus.

We want to solve that problem and improve!

↓

slide 13// advanced DRAM Organization.

← صنا نصير نجيب بدل ما نصير سطر

↳ نكتفي إنه نحدد بداية ال block (بداية) ال line الي بدي اجيب منه)

Or the row

ويعني بنصير نطلع عال bus بالترتيب ال data ، عند نقرأها

بنختار ال columns واحد واحد ، لأنه ال bus ضيق

ما بقدر نجيب ال row كامل مرة وحدة.

لهون الاشي بيعتمد عال special locality ، لأنه احنا اصلا بنجيب blocks

من ال memory

Note: the spacing between data 1, data 2 ... is now shorter than the previous design (higher bandwidth) → improvement

law bandwidth → classic DRAM

higher bandwidth [get arow, put in row buffer taking elements from it column by column]

→ fast page mode DRAM

↓

FPM DRAM ??

Fast page mode DRAM offers improved speed over standard DRAM. When memory access is performed within the same address (row) (page), a single precharge is required for only the first access. Subsequent access within the page can then be made without the time penalty of additional precharge cycles.

Although it has lost popularity in some areas to faster technologies such as doubled data rate SDRAM, But FPM DRAM is still a very cost-effective solution when the highest performance is not required

/ /

- A problem in FPM DRAM ?!

knowing that a row is entirely coming because of special locality

But! what is the thing that control the values of colu.1, col.2, col.3 ... (addresses of columns) in slide 13#

↓

بالتصميم العادي هون، كل مرة بدي ابعثله ال address جديد

يعمل ال decoding، وبطلعلي ال col.1، col.2

في manual work منطقي أكثر!!

بدنا نحسن هاد الموضوع، لأنهم جنب بعض، ليش كل مرة منستنى

يبعتلنا column address وهما جنب بعض!

بحل!!

لأنهم جنب بعض بدي بس أعرف أول عمود و نعطيه على counter، ونشتغل

عن هاد ال counter ← clock، والعداد بصير يعد automatically

وبصير يطلع ال elements، لنقدر نحط الصف كامل في row buffer

بدل ما أنا أعطيه رقم ال row بس، وأعطيه أول عمود ويكمل يأخذ

الحساب ال col. الباقيين

عبارة عملية التزويد أسرع!!

→ Synchronous DRAM : with clock

→ Asynchronous : without clock

Recovery for the name

→ RAM : you can load data from any where in it (Random Access Memory)

→ DRAM (Dynamic) : in the design we have 1 transistor with capacitor, and it has refresh cycles (a lot of dynamic work)

→ SDRAM (Synchronous Dynamic) : clocking for the counter that gives higher bandwidth and speed advantage.

DDR: depends on the concept of reading two columns in one cycle, one on the positive edge, and one on the negative

↳ while in SDRAM, we only were able to read one column in each cycle, either on the falling or rising edge.

Nowadays we have DDR1, DDR2, DDR3, DDR4...

↳ differences between them depends on the speed of clock, starting from 66 MH or 200 MH, to 4200 MH or 4660 MH speeds [solving the problems of electromagnetic interference and such problems, to be able to reach these velocities of RAM]

Higher speed → higher frequency → reading doubled data.

Higher Bandwidth ←

There is also GDR → GDR SRAM
→ GDR DRAM
} Not that popular

↳ Super computers application.

Main Difference between DDR and QDR :-

QDR: uses two clocks, with phase shift between them

→ 2 clocks (positive, negative) for each
→ two ports
→ then you can do 4 operations

But!!
- the first clk is dedicated for reading data
- the second clk is " " writing data

→ [in DDR, data had only one direction, you either write or read]

- Reading/writing can be done at the same time in QDR (dual ported)

read port ↓ write port ↓

→ Maximum usage of QDR only happens when you are reading and writing in the same time, throughout all your work.

Quick Recovery

There will be

[1] controller

to recieve the addresses of CPU and cache sub-system requests

↓

This address is transformed to be divided to two parts: Rows, Columns

in slide 17:

→ Row address latch and decoder

→ each decoder is 14→ 16384 decoder

↓

14 decoder

16384 outputs only one should equal (1) after decoding.

→ column decoder

10→ 1024 decoder (only 1 of these 1024 lines should equal 1 after decoding)

this means that each row has 1024 element

After putting data on buses (on parallel buses), then it reaches the north bridge, and the controler in the motherboard (in modern PCs and laptop's) from where it goes to the CPU and be put in the cache sub-system

-> DRAM Generations

- capacities are doubling, because DRAM are similar to CPUs, where the design is based on resistors, and we know that resistors are following the principle of Moore (shrinking the size of resistors, doubling the capacities each 1 year and half)

it is pretty easy to find 32GB (Byte) capacities these days on sticks

What is the important point now?!
- improving TRAS, and TCAS ✓

Reminder:
from the moment, Raw Address Strobe is asserted, and address is analyzed, latched then decoded, then them row is moved from row array to row buffer → this is TRAS

→ Asserting CAS, latching, and decoding column then putting data on bus from the row, this is TCAS.

Why the improvement on TCAS is better than TRAS?
↳ moving from classic DRAM
↳ to FPM DRAM
↳ to SDRAM
have improved the work on columns
↳ and this is improved more when we moved to DDR (reading two columns / cycle)

In slide 20#

talking about DRAM Banking

→from the beginning, Why Banking ?!
→ it increases the bandwidth, and this can be proved by numbers

But how Banks are used in design ?!
→Back to slide 17#
reading the procedure again :↓
① row address is found
② row address is now an input for the decoder
③ after getting the output from the decoder, you can read more than one row, each is from different region (Bank) in the same time (in parallel)
→this makes the control easier and helps to distribute data effeciently

But how to decide which row of those has to be read ?
the Bank Control logic (decoder) decides which row of these has to be read.

احنا بنكون موزعين ال rows على أكثر من Bank ، وكل ال banks جنب

البعض ، كلهم فيهم ← rows 16384 ، اذا احنا موزعينهم على bank

وبدنا row رقم 1000 ، بنروح ع Bank و نجيب ال row اللي

ترتيبه ألف

↓

So you have two controllers, decoder to identify row location, the other decoder is to identify from which bank you should read that location

احنا بكون مخزنين ال rows على اكتر من Bank وكل ال banks فيها
الليزان ، كلهم فيهم 16384 rows ، اذا احنا مخزنين على
bank وبدنا row رقم 1000 مثلا اي Bank و بدنا ال row اللي
جريبه انف

↓

So you have two controllers, decoder to identify row location, the other decoder is to identify from which bank you should read that location

studying slide 21

figure (4)

scenario on the communication:

Assume the processor is doing ld/sd, and is heading to cache, to miss or hit the target address (Hit: data is loaded from the cache to RF and the processor works on it)

But if Miss?!

↓

the cache sends a request to the higher level memory (DRAM in laptops/mobiles/computer systems), the wanted address is sent on the bus (requesting from memory)

↓

and the process of RAS, CAS, decoding that are done on the addresses in the memory are understood before.

→then the memory returns the requested data on the bus to the cache sub-system, to let the CPU work on it

So what should be found here?! (calculated)
Bandwidth and miss panelty

miss penalty: in the case of trying to access some address in the cache by the processor, but the wanted content is missed, you should go through the bus to memory (hence the time elapsed to pass the address from the cache, and doing all needed processes in the memory to return the data to cache)

↓

This is the **penalty**

Bandwidth: the amount of Data sent on bus / the time needed for this data to pass the bus.

The example in slide 21: miss needs to reload
- 16-bytes block from the memory
  - → Remembering (words vs bytes)
  - standard (most used) is word = 32 bit = 4 bytes
- intending to read 4 words from memory

But! what are the specifications of memory?

in figure (a)
memory is said to be : one-word-wide memory organization

↓ what is that ?!

locations are internally 4 bytes by 4 bytes

width of each location is one word (4 bytes)

↓

which means in this example that our 16-bytes are divided into 4 locations

*Bus has the width as same as the memory (one word wide)

↓

In summary:

- 4 bytes bus / 32-bits ; means that the bus can hold the entire word while transferring it (there is no any bottleneck here) !!

↓

the bus can transfer the entire word in parallel

- 4 bytes wide memory
- classic DRAM → needs dedicated request to load each word from memory

Example applied on figure (a):

- 1 cycle needed to request a word from memory
- 1 cycle to send data on bus from memory to cache
- 15 cycles needed to process data and get it from memory

miss panelty to get 4 words = 4(1+15+1) = 68 bus cycles

(we call it time delay)

4 is multiplied because it is a classic DRAM where dedicated complete work is needed for each word.

single request for each word

Bandwidth = $\frac{\text{number of bytes}}{\text{miss panelty}} = \frac{16 \text{ bytes}}{68 \text{ cycles}} = 0.24$ B/cycle

Target !! improve the previous design ?!

How ?!

suggested improvement: to increase the ~~wit~~ width of memory, to be as wide as the number of words you usually need to read.

↓

usually need to read 16 byte: let it be 16 byte wide (as an example)

→ bus should be widened also

→ even in classic DRAM, you still need one request for each 4 words (16 bytes)!

thats the point!! (one address needed)

↓

to load the block of 16 bytes

no redundancy ←

the operation is done once

one request to get the 16 bytes (4 words)

→ and then the cache (using a multiplexor) chooses the bytes that CPU needs [crossed out] specifically

اذا جبنا ال 16 byte, مين بدو يعطي ال CPU بدها؟ بنعرف، بنستخدم ال multiplexor

→ multiplexor slide 22 figure (b)

But! What is the problem of this design ?!

/ /

Quick Recovery

→ increased the width of bus

the bus is the electrical real lines that transfer data from the memory to CPU and cache subsystem

خطوط physical بنشوفها بالقاسيات بالرضية
عال ← mother board

→ because of this widening, now we use 128 physical and electrical line, instead of 32

↓ in routing (how you put them on mother board), will take a wide space

80 : ←

① we'll increase the size of motherboard which means higher cost.

② electro magnetic interference (EMI) which causes errors in the transferred data.

↑ this is the problem!!

what to do now ?!

to fix all these problems, it is said to be:

→ ① leave the bus ~~and~~ without increasing its width → 4 bytes wide only (one word)

→ ② instead of making the memory as one block (16 bytes next to each) / divide the memory for 4 banks
and now the data which was on one row, no we'll distribute them to banks

now remember, in banks we have 2 decoders
→ decoder to choose the row
→ " " " bank

شو اثر ترتيب ال bytes على نفس الصف بس موزعين على 4 banks
← في ديزاين figure (a) ، كانت ال bytes موزعينة على اسطر مو جنب بعض ، فاستخدمنا decoding للأول والثاني والثالث والرابع (الاسطر)

← لما عملنا banking ، اللي كانوا موجودين كلياتها بالصف الأول (صاروا ب bank 1) (صف أول - bank 2) ...
فلما نيجي نعمل row address decoding ، انا بجيب اعلمه مرة واحدة معظم لكل هاي ال words
لأنه كل هاي ال words راح تكون موجودة على نفس الصف لكن الأعمدة حسب ال bank

→ now, saving time in row decoding, that takes the longest time in the whole operation, only if takes time of decoding the row address once for all banks

Now:

① send the request in 1 cycle (send the address)

② in 15 cycles, you can get all the rows containing the target data from the same row from all banks

now the problem is (How to return data !!)

we can't put all the 16 bytes once on the bus (it is 1 word wide), and we should transfer 4 words from each bank.

So we use a multiplexor, and in each turn we choose one bank, as we are saying:

bank 0, put your data on bus and transfer (1 cycle)

" 1, " " " " " " " " (1 cycle)

" 2, " " " " " " " " (1 cycle)

till bank 3, hence we have 4 cyles

③ 4 cycles to transfer data from banks to cache

↓

20 cycles miss penalty!

→ wide, bandwidth close to the ideal without problems

↑ this is called bank interleaving

Can we do better? yes!

the previous banks system is not DDR, it is just SDRAM, when it is DDR (where you can send two words at a time / at a cycle)

now we multiply each bank by 0.5 not 1 because in one cycle we we transfer the data of two banks

positive edge negative edge positive edge negative edge

→ 0.5 of B0 + 0.5 of B1 + 0.5 of B2 + 0.5 of B3

cycle cycle cycle cycle

= 2 cycles only

slide 23 ↑

↓

✓ one on positive edge

✓ one on negative edge

→ if we used the QDR, we multiply by 0.25 not 0.5, because we transfer 4 words per cycle

But!! this calculation success when we are transferring data in both directions, (in and out of cache), because QDR transfer 4 words only in both directions. it can't transfer more than 2 in one cycle for one direction (read or write direction)

So it stays 0.5 not 0.25, because QDR is DDR for each port (write or read port)

Quick Recovery:

SRAM (of cache and registerfile), and DRAM
→ but these types are nonvolatile, but we need a permanent storage (so we have the Hard magnetic disks) and the flash memory

→ for flash storage; we don't need battery inside the SD card or the USB flash to retain frequently to keep the data inside it
→ even when applying it to system it does not consume huge amount of energy

→ the numbers written on chips of flash of memory are the speed grade
the speed grade of flash should be suitable for the application
(like HD and 4k cameras, that handle huge amount of data, they need high speed grade to be able to handle all video frames and save them)
→ the low speed grade can be used for cameras of images only

Flash types

→ the modern SSDs, there are algorithms written on the controller Hardware inside the SSD, which does something called [wear leveling] → مساواة الكتابة

↳ We can't keep writing on the same location and clear, write and clear, while we have empty capacity, which makes this location weared while other locations are still good.

if we don't use this algorithm we loose the capacity of half of our harddisk

wear leveling makes it live longer (SSD)

it remaps data to less used blocks

HHD : hybrid hard drive
HDD : Hard disk drive.

Disk Storage

The ports & interfaces

← زمان كان في اشي اسمه ال SCSI ، احد انواع التوصيلات الي بتوصل ال HDD بال mother board

وقبل مرة كان ال ATA هو الي عنا عبارة عن parallel port (حوالي 40 pins)

هاد هو الي دسا نه عنا حاليا.

← يعتبر هو ال bus أو ال interface ، إسلاك خارجي الها standard تحديدًا

في نقل الداتا ما بتوصل الها مثل ال mother board الي فيها ال internal buses

The embedde system

← بتحكم بكل الامور الميكانيكية

← في algorithms خاصة فيه بتحولها ال system بحول ال address الي جايه من ال computer لـ location على ال cylinder

(من address ارقام binary لموقع حقيقي او زوايا و levels)

cylinders

→ physically made from ferromagnetic material ← مواد داخلها الحديد وبتصير مغناطيسية

→ the iron inside this material controls the magnetic directions (the iron has magnetic characterstics), so if we directed iron attoms to north, then we have logic 1, if directed to south then it's logic 0 → such thing happen

* we have more than one layer of cylinder to increase density and capacity. (they could pack more and more)

→ so it is easy these days to find 8TB HDD

our data will be on the surfaces of these cylinders

sector sizes:

the sector does not have a constant size (capacity) it can change

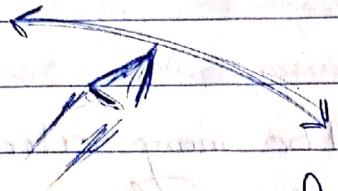
→ you change it from the laptop from the allocation unit size.

the default is usually 4096 bytes (4k per sector)

* ideally we hope that the sectors that holds the data of a video for example are existing directly next to each other, But this is not happening actually.

The parallel heads

→ each can move (right-left) on one level
two directions
from the smallest cycle to the biggest one



to reach the rest of sectors, cylinders(disks) will move around so that the head can read all seetors

The speeds of disks moving around are main 3:

| 5400 |

| 7200 |
standered
for all
PCs

| above
11000
or 15000 |
for special usage
(like servers)

* The thing that controls the rotation and the location of each head is the controller in the embedded system

Disk Sectors and Access

size of sectors (tricky)!

→ if it was so small, there will be many sectors which makes the management harder.
(عدد مهول كبير و الـ management أصعب)

→ if it was so large, there will be wasting of capacity some times, because in these times you have one file that is all about $\frac{1}{8}$ the size of the sector, you will waste the rest of available capacity this way
(بيضيع مساحات بسرعة أكبر)

Trade off
→ so then the default size is 4096 (4 kB), as the better size

Scanned with CamScanner

/ /

→like the EDC (the parity /error detection code)

ECC :

→errors may come from ~~electrical interference~~ (flipping bit-s)

→we have to solve these problems (detect them)

Because of the magnetic interference *

The problem about EDC (the purity; odd and even) is that it can detect odd number of bits only

→if 1 flipped bits it can detect it

→" 3 " " " " " "

→" 5 " " " " " "

it can't detect 2, 4, 6 ...

* But we want to detect all problems and errors and hide them, so we found ECC

* there are gaps between sectors to help the HDD recognize the beginning of a sector and its end.

Scenario:
We want to watch the movie, so we opened it, and the movie player loaded the frames of the movie to display.

→ at the beginning "miss" will occur
→ it goes to cache: miss also
→ " " " DRAM: miss also (called page fault)
↑ something like miss
→ it goes to HDD
→ it sends the address
(if the hard disk was idle, and not busy with another application requests)
Hopefully, the HDD is idle, // there is no Queuing delay
→ either we have Queuing delay (time or not) we will have these three thing when starting to response to request.
① seek time ② Rotational latency ③ Data Transfer
④ Controller overhead.

controller overhead: time needed to:
→ analyze the address
controller of embedded system. ← → determines its location
→ ,, the angle of location
→ ,, the track and sectors
↓
* to turn it from binary address to physical locations on harddisk

→ operation of moving the head (find track)
Seek: time needed to move untill you reach the target track
بالحسابات احنا بنتعامل مع ال average seek time، لانه الوقت بختلف حسب بعد ال track اللي بدي اروح عليه عن ال head الحالي

→ operation of rotating the disks (find sector)
Rotational latency:
Best case: accedently the head is pointing at the wanted sector
Worst case: we need a complete rotation

$\frac{\text{Best case} + \text{worst case}}{2}$ ← average time ← بنجيبو من مجموعهم بال
↓
هي النص

/ /

now reading the sectors depends on:

① data and sectors length

② data transfer rate → سرعة قراءة البيانات (قرص، طارد، رسل)

Disk Access Example

512 byte : sector size

15000 r/m → 250 r/second : rotational latency

$4*10^{-3}$ seconds : average seek time

$100 * 1024^2$ Byte/second : transfer rate

$0.2 * 10^{-3}$ seconds : controller overhead

no Queuing delay : idle disk

→ Average read:

Queuing (0 seconds) + controller ($0.2*10^{-3}$ seconds) + seek ($4*10^{-3}$ seconds) + rotational average ($\frac{1}{2}$ * 1/250 r/second) + transfer time = **6.2 ms**

عاملنا معكوسها لانه اعطانا second per round مش العكس تذكيرا بذكرة تكني

How to extract the transfer time for 512 Byte sector, depending on the previous information:

100 MB ⟶ 1 seconds to read

512 B ⟶ ? seconds (call it x)

$100MB * x = 512B$

$104857600B * x = 512$

$x = 4.8828 * 10^{-6}$ ⟵ تقريباً زي ال 0.005ms في السلايدات.

Note:

اذا كنا بدنا نقرأ اكثر من sector ، وصدفة كانت ال sectors كلها ورا بعض ، فأنا محتاج عشان احسب الوقت بس تتجزأ عني طول لما احسب ال data transfer بدل 512 اشي ثاني

لكن الباقي برضه بحسب مرة وحدة.

⟵ الآن لو افترضنا انه ال sectors مش ورا بعض او فيه بعضهم ما بكونو ورا بعض بسبب علة بحرف بالكرا ، فبكون في sectors خاصين بالبعض والـ controllers لازم يكون عنده وعي بوجود البرنامج ، ووين ال sectors الناقصة والبالية.

to keep track how the data is distributed.

بيحصل حالة ، بيصير ال head يتحرك يمين شمال فالفترة في seek time
اعلى ، في latency اكثر

to solve this problem, we have some tools on windows and linux, to determine the percentage of data scattering and then doing some permutations to put the sectors of the video or something directly next to each other to improve the performance (this is called fragmentation)

وما بصير يتحرك ال head من مكان لمكان وزيد سرعة القراءة

* do not do fragmentation for SSD

ما في اشي ميكانيكي بيصير كله كهرباء ، فوين ما احط الداتا بصير
راح اوصلها بنفس السرعة ، عشان ما دفرزة لانه اله
عدد مرات قراءة وكتابة محددة ، وهو مش محتاج اصلاً

~~[illegible]~~

Section 3 // Cache Memory

two questions! when reaching the cache to load:

① how to know if it really holds the data or not? (miss / hit)

② if it is (hit), how to find the data in the cache ?!

→ size of cache is much more smaller than DRAM

it is subsets from DRAM

cache (between K→M bytes) 32 kB 64 kB 256 kB

DRAM (talking about Gigas)

[1] Direct Mapped cache:-

There are no lines here / blocks, we just load the adjacent members (no usage of spacial locality principle) or temporal locality

cache:- (blocks structured)

each block capacity: 1 word / 4 bytes

→ DRAM is also blocks-structured (more than cache)

Here in this design:

→ each location is a block, either in cache or DRAM (4 bytes)

General rule to transfer the data from DRAM to cache:

[1] target address mod (number of Blocks in cache) = the location to put the data in, in the cache

↳ This may cause conflicts

The conflict is described as: (many locations (words) in DRAM, they will map to the same location in cache → Direct Mapped cache)

[2] Use $\lceil \log_2(N) \rceil$

[1] in cache, number of blocks is a multiple of 2

↓

[2] DRAM is the same

use $\log_2(N)$, N is the number of blocks in cache

so in slide 33, do the next:

① $\log_2(8) = 3$

↓

② take the first three bits (least significant bits) from the right-hand-side, these bits will determine the location in cache.

This is done upon the assumbtion of:

each 4 bytes are sharing the same block, and the same address in DRAM, so to load 1 word (4 bytes), you need one address.

نعيدًا ال DRAM حقيقة ماشية زي ما لكل 4 bytes عنوان واحد

→ This also can cause a conflict

The Question is

if one location in cache can hold the word of one of many locations in DRAM → [conflict]
so how to determine the source?
of loaded data.

→ we used the 3 least significant bits, but the rest of bits is called the [Tag of Address] → الجزء المتبقي من ال address

so we load the data to cache, and the Tag to determine the source [cache بنعرف منها ال داتا جت من فين بالظبط]

→ We will also add a bit in cache [0 or 1] to determine if the block in cache is empty or not. (Valid bit)

→ it is necessity to modify the content of some block in cach, to be basically empty block, the index may be correct, but the tag is wrong, so we have the data of wrong location in DRAM, then we go to request the right one from DRAM (because this is a miss), and modify the content of block + tag bit.

This is because many blocks can be mapped to the same location in cache

examples till slide 40:- are for the cache that is word-based.

4 bytes ← 32-bits

word 1	00000 → each address is for a word
word 2	00001
word 3	00010 → we use least significant 3 bits to identify the index the rest two are for the tags
word 4	00011

Things are different for 1 byte based caches

Slide 41 → 1024 Block

→ $\log_2(1024) = 10$, we need 10 bits from the the complete address to know where to save in cache [this can represent 1024 locations

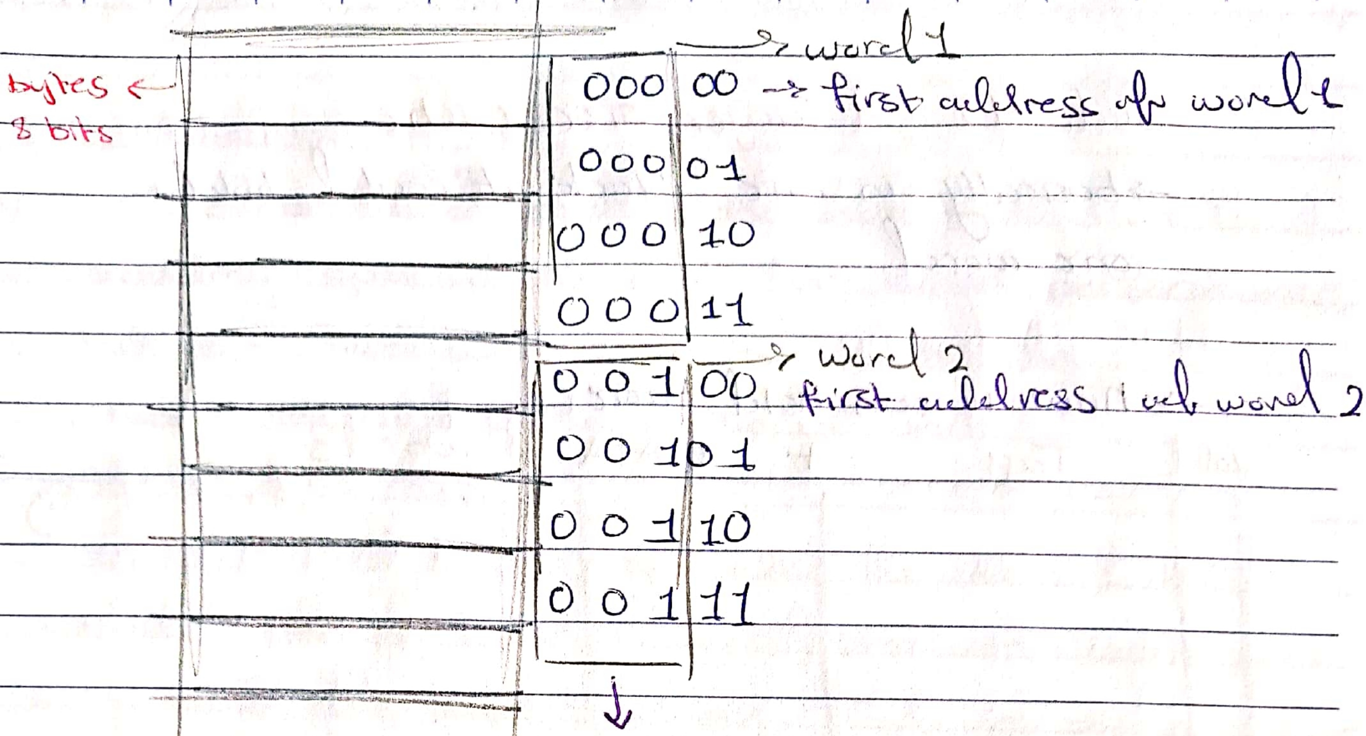
but from where to take these 10 bits from address ←

address →

rest is of tag (52 bits)	10 bits	2 bits shift

why to shift 2, then take 10 bits

لأنه بداية من ال bit التالتة لمدة 10 bits، بدءوا يتكرروا
معانا لكل 4 bytes، عشان نخلي ال word كلها نفس صول
ال 10 bits، و 2 bits الأولانيين، بيعملوا نمط متكرر لكل 4 bytes.



نحدد بعدد حقول
الي راح نختار عنه
جزئيهم loc الي v
ال cache عند آخر جزئية ← $\log_2(N)$

Slide 42: A larger size cache

→ basically in one block it could hold one word

→ now it can hold more.

valid	Tag	DATA: 4 words ①	②	③	④

هيك احنا صرنا نحط ال indices حسب ال address من كل مجموعة مكونة من 4 words، هيك ح نقرأ ال offset bits، عشان نعرف

من وين بدنا نطبق ناخد ال bits اللي حددوا ال index

Slide 42:

The more [big] line we get, the more words we load providing spacial locality for better performance, but this true with some constraints only:
the constraint is the cache is small

after knowing that, know when the block of data becomes wider than one word, we have two ways to identify where to save the data in the cache ✓

[1] Using Mod in slide 42

First divide the address on the number of bytes in the block to determine the group / the block that you'll load and your target data will be in) → كدي بجيب

then use mod
address mod number of blocks in cache after division

ال index اللي بيحط فيه بطريقة تشبه اني بجيب address اول address في ال block ده

[2] Using the offset bits

determine the number of offset bits to know from where to start taking bits of that block [like the ten bits in the design of slide 41]

then load the block

Example: you have 32k bytes, design it in two ways: length of 256, length of 1024

[1] length of 256

the block size = $\frac{32k}{256}$ = 128 byte [so wide]

↑ reduce miss rate

↳ This provides good spatial locality support, but 256 locations for a very big DRAM, will cause a lot of conflicts, a lot of addresses in DRAM are only competing on 256 locations in cache, and these conflicts will lead to modifying the content of these blocks many many times frequently, [128 byte will be modified at once when we load another address that compete on the same location in cache], these frequent modifications will conceal the advantage of spacial locality (why to load a lot of bytes while you know that you may modify them very much soon before you are able to access them in cache)

↳ Miss rate can increase here because of frequent changing of the content of blocks because of the competition on them

[2] length of 1024

size of block $= \frac{32 * 1024}{1024} = 32$ byte

→ this solves the problem of the previous design (alot of conflicts, few locations) we have 1024 locations

→ if you are working on an array, you will get a miss each (8 elements / where the element is a $1word), not good handling for spacial locality

↳ Miss rate can increase here because of small block sizes

→ This is tradeoff.

The question is on what to depend when designing these caches?

→ design, simulate, determine what is the best for you!

What if we want to store to data cache? [we can't write on instruction cache.

Design in Slide 16:

number of offset bits = $\log_2(16) = 4$

→ 10 bits for allocating words

→ 18 (the rest) is for tag

management size = 1024 rows [one for each Block] * (1+1+18)

- valid bit
- Dirty bit
- 18 bit of tag

1024 * 20 = ~~20k~~ bit

8 * 20 bits ← 2.5 kByte

* What is the percentage of mangement from the whole SRAM Block?!

$$\frac{2.5\ kB}{(2.5+16)\ kbyte} = 0.135$$

The D bit (Dirty bit)

→ we use it when we want to store (write)

The principle is, when you want to modify the value of some thing, modify it in the closest level to you (cache), not the RAM, because if we want to modify for more than one time, why to keep going to RAM in each time? just update the value on cache

↓

When you update the value of any bit in any location in cache, update its corresponding Dirty flag to 1, to announce that the value of this element is old in DRAM and the newest is here in the cache

↓

The same variable has 2 values → old in DRAM
→ new in cache.

اذا عملتي store ع اي مكان بال cache ، بيقلب ال D

عشان يحكي لل system انه The most recent and up-to-date data is here in the cache subsystem

/ /

in Design slide 46#

→ when we loaded a block from DRAM to cache, we loaded the target address + its adjacents

↓

but now how to transfer only the target data to CPU?

↓

الجزء اللي تحتاجه في الدرزاين check سلايد 26

بالضافة اذا تحققت منوجد ال management bits نبعث

as a decoder خط كامل enable

Question: on design (46) slide

if we raised the address from 32 bits to 64 bits, what is the size of management bits?

50 (tag) + 1 (D) + 1 (V) = 52 bits [high overhead]

→ This is considered when talking about the cost.

Design of slide 46

Assuming we want to write on cache!

SD X15, 8(X31)

Assumption: (of current scenario)

(1) the address in the existing scenario is existing in cache to update its content.

How cache works when it exists?

1) calculated 2*8+(X31) to know the address

2) identified the block that contains it and the word inside the block

The problem is you can't write on only (access only) one word [you either write on one block or read one.]

What to do after determining Block and word?

To store on cache you have two scenarios

[1] to load something from DRAM to cache
[2] to update some variable on cache from CPU coming data.

→

The overall scenario:

[1] data is read by [Data Read]
[2] all words read by it are taken to the muxes above.
[3] the muxes above have 3 inputs:
  [A] input from old data read by [Data read]
  [B] input from the data coming from DRAM [Mem_data.data]
  [C] input from the CPU to update the data on cache [CPU_req.data]

Three cases: The address you are trying to access to store on may be ↓

Miss
↓
enable
Mem_data.data
input.

Hit
↓
enable
CPU_req.data
or
Data Read (old values) input.

$: means cache

1. selection line on muxes identify which input to enable
- we have a dedicated mux for each word.

When updating data:
→ [Data Read] signals are sent on bus to muxes [each word to its corresponding mux]
→ CPU_req. data is sent to each mux
→ The right mux is enabled only to upload the data to its word and update it.
→ rest of mux only pass the old values to words without any update.

→ The selection line determines which value to pass to corresponding words

~~Remember~~
→ Remember when writing to cache and updating it, we update the value of D bit to 1's (value is updated)

D is flipped to 1! DRAM is now inconsistant and synchronized with the values in cache to announce the update

Question: What can we do instead of putting a dirty flag [?] to denote that there is an update

↓

① do write-through
update cache and DRAM
in the same time

Problem!! this way is slow!
trying to write on cache, 1 cycle or 2 are needed
" " " " RAM, 100 or 200 cycles are needed.

[!!] it is not acceptable to stop the program to update on cache and memory consuming all these cycles for a high delay [what is the advantage of cache then?]

[2] we may need to update the same value on cache multiple times after each other, it is not acceptable to go to RAM to ~~upda~~ update it also all these times [time consuming]

↳ it is better to wait until you finish all these updates on cache then take the final value to RAM

The only advantage of write-through is the consistency [the system has the most recent data every where]

*improvement on the design
(slide 48)

→ add the write buffer
→ the CPU can keep working
you update the data in cache
and write it to write buffer

→ let the system load the updated values from write buffer to Mem shortly to be up-to-date
[CPU is still working, cache is not blocked]

Two problems:
[1] the more cost
[2] the small size of write buf.

assume the case of fullfilling the write buffer, such that you can't write on it, here we stall the cache [stop updating on cache until we transfer data from the buf. to RAM, and be ready again to accept new updates]

The effective CPI, which means the penalty of store instructions, that works on a cache of write-through type, waits and writes on mem in the same time [the old write-through] without the buffer, is:

CPI = Old_CPI + percentage of store * cost of store (how much cycles it takes)

↓

The new number of cycles needed to finish 1 inst.

check shicle 48 r)

→ With write_buffer, we don't use the above equation except in the case that the buffer is full, such that we can't write on it any more.

Write through is slow, buffer provide s some improvements, but there is still some problems

Second way is to (Write Back)

→ note that both write-back and write through are being used, each is used for a level no one of them is better than the other.
dirty flag is not used in write through but used in write back

[2] Write Back:

General Rule: do not update from cache to RAM until there is local miss (read miss) in cache so we need to write from mem to cache to handle that miss

↓

→ store instruction on cache to update location A
→ load instruction from cache to read from location A
→ But location A may the data of different address than the target one
→ we want to load the right data from mem to location A
→ But we don't want to loose the updated values in location A
→ We store the updated value to mem, to ensure

we have the most recent data permanently, then we allow to load the missed data from mem to location A in cache to read it to CPU, while we are not worrying any more about our updated values

هسا ال load مشتغل كدا الكتبة ع الذاكرة
100 cycle ← كتابة ع ال mem
100 cycle ← كتابة ع ال cache
وال CPU قاعد بستنا ^^

To solve the problem of this delay, just change the sequence of execution:
instead of sending the update to mem, then loading data for load inst. from mem. to cache → (200 cycle)
we do the next:

[1] send the updated values from cache to temporary write buffer, and let the cache be ready to receive the data from mem to execute load
→ 100 cycles to transfer from mem. to cache, then sending them to the CPU, [CPU only waits 100 cycles instead of 200]

[2] letting the write buffer to send the update to mem in any time shortly without forcing CPU to wait for this to be done ✓

The previous two scenarios indicate that for store operation (on cache), we always get (hit)
→ in other words, the address that we want to update its value is always existing in cache!

But what if this address is not existing!? (miss)

what scenarios to be done // slide 51

- hit to store
  - ①
  - ②
- miss to store
  - ①
  - ②

① Write no Allocate:

you missed ?! then your address is not in cache just go directly to RAM and update it here.

you don't allocate, means you don't allocate a block and saves something on it

دحين من الآخر دا الموضوع يهوانه!!

② write Allocate

→ load the data (missed address) from memory and its adjacents

→ put it in Allocated block in cache

→ update their values of these loaded words

which is better? Allocate or no Allocate

→ no allocate is good for one variable, useless in spatial locality needs for arrays

→ allocate provides handling for spatial locality for things like arrays

it is true that we miss the first element in the array to update its value, but we are intending to update the values of the adjacent elements.

Scenario: we want to update first 4 elements of an array!

→ no allocate:

miss a[0] → 100 cycles to update in mem directly

" a[1] " " " " " " " "

" a[2] " " " " " " " "

" a[3] " " " " " " " "

/ /

allocate:

miss a[0]? load a[0] and its adjacents and update them in cache with less cycles required.

↓

and only send these values to RAM when some replacement is about to happen because of [local miss] in the location that holds the values of updated a[0], a[1], ---

حين الحسبة للبرامج تبعت ال loops ⟵ allocate

ايش تاني ⟵ no allocate

This question is answered by the statistical results using ~~[illegible]~~ Softwares, to determine which way is better for each type of software, if it is full of loops or something like that.

Last five minutes in lecture 23

Weighted average in slide 54

→ The normal average is to take = (I-cache miss rate + D-cache " ") ÷ 2

This way assumes that each one is accessed 50% times of all accesses

- But in fact, this does not happen, we may have 20% I-cache accesses, for 80% D-cache accesses, so the weighted average is considering this: we multiply 0.2 * I-cache accesses and 0.80 * D-cache accesses

(0.20 * I-cache miss rate) + (0.80 * D-cache miss rate)

2

Section 4 # Measuring & improving cache performance

Example to start : The program [ideal case assumed where the data we want is always ready in cache (hit) & M stage takes one cycle.

LD
LD
ADD
SUB
SD
MUL
MUL
LD
DIV
SD
SUBI
ADDI
ADD
SD
LD
MUL
SD

[1] determine the percentage of inst. that access the mem. to the number of all inst. [these are LD + SD]

$$\leftarrow \frac{\text{Memory Accesses}}{\text{Program}} = \frac{8}{17} \simeq 0.47 \rightarrow 47\%$$

تذكير :
عملية ترتيب ال CPI
مثال scheduling
بنجح عدد ال cycles
بنقصه ع عدد ال inst
وال IPC بنعكس

[2] determine if all of these LD and SD are having penalty of miss or not

What is the number of inst. that have a miss ?!

مش كل ال inst بالضرورة الكل ← miss
↓
Miss rate

But how the cache misses affect [the base or ideal CPI of ideal cases (always hit)]

[3] Multiply the miss penalty to miss rate to find the overall miss penalty

In English: from the entire program we find the instructions that are accessing memory [$\frac{\text{mem accesses}}{\text{program}}$], then we find the inst. that have misses from these instructions [miss rate], then we find the overall misspanelty to find memory stall cycles // slide 57

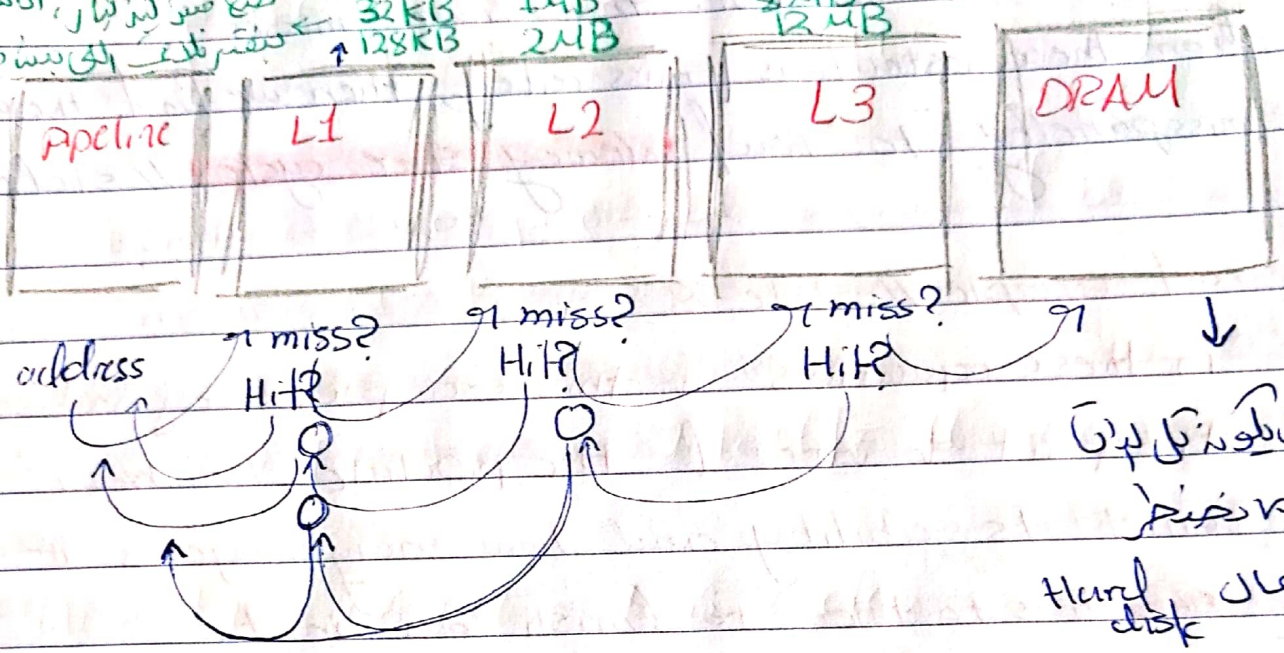
real example // slide 58

in these equations of this example, they indicate the point of view of the pipeline, for know what its ability and how many cycles it has to take to finish an inst.

↳ slide 59: use the point of view of the time [how much time we need to load our data from RAM to handle the miss]

More realistic examples on that with more levels of caches in Sheet 6.

Their sizes are not high



Hit in level 1: → pipeline

" " " 2: → L1 → pipe

" " " 3: → L2 → L1 → pipe

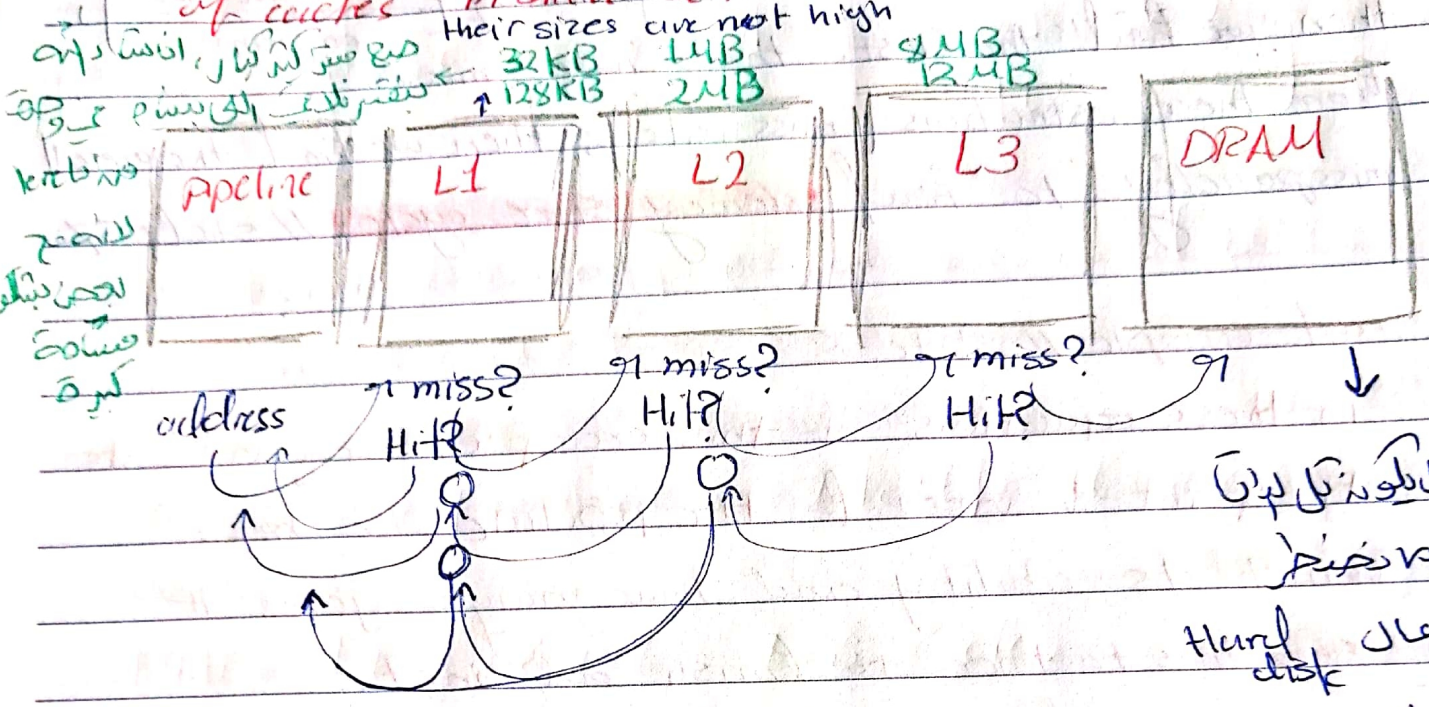
:

and so on

→ we pay the complete penelty when we should reach the DRAM level because of all misses

More realistic examples on that with more levels of caches in slide 60,

Their sizes are not high



Hit in level 1: → pipeline

" " " 2: → L1 → pipe

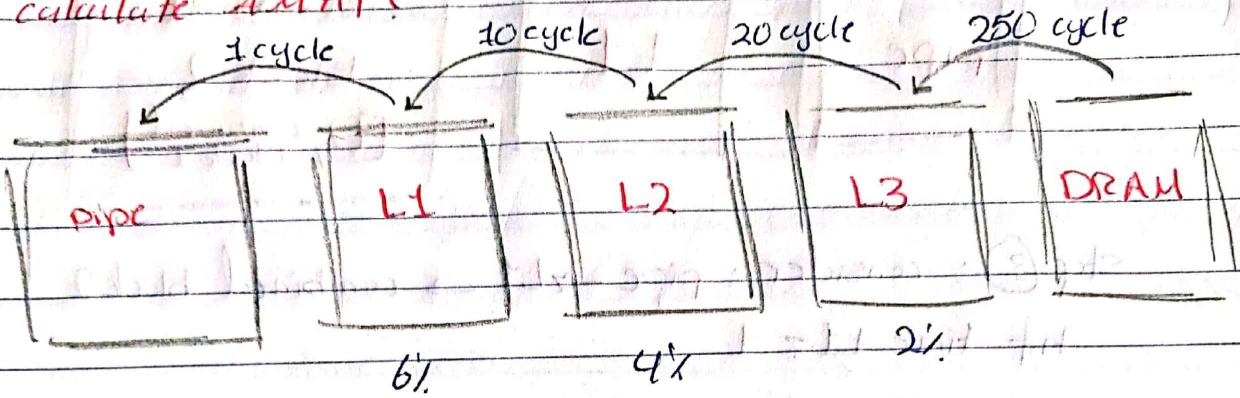
" " " 3: → L2 → L1 → pipe

:

and so on

→ we pay the complete penelty when we should reach the DRAM level because of all misses

In slide 60: how to analyze the system to calculate AMAT?



AMAT = hit time + miss rate * penalty

how to apply it to a hierarchical system?!

↓

operation always start from the slower level towards the pipeline

تأخذ L3 + DRAM وبصيروا/ بسيط L1, L2, pipe سوّيهم unit eg

ال level كل ما باخذ 2 levels بحولهم

step (1): consider: pipe → L3 → DRAM

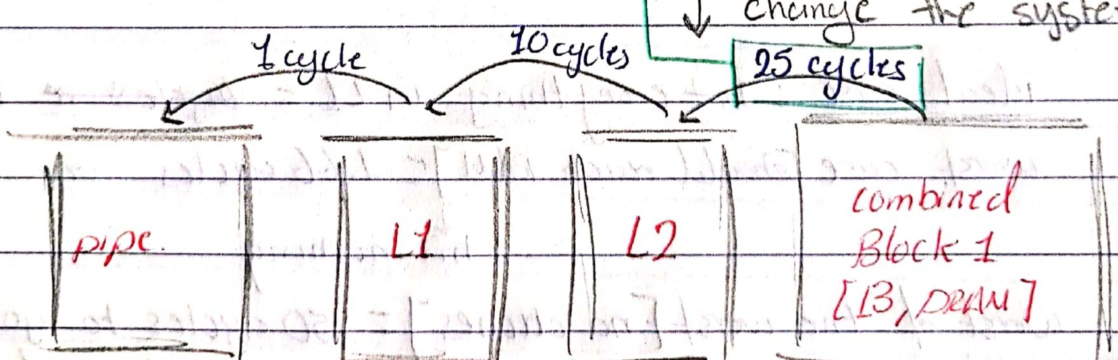
hit time L3 = 20 cycles

miss rate L3 = 2%

penalty to load from DRAM to L3 (Miss penalty) = 250 cycles

→ AMAT = 20 + 0.02 * 250 = 25 cycles → AMAT [L3, DRAM]

change the system



step (2): consider: pipe → L2 → combined block 1

hit time L2 = 10

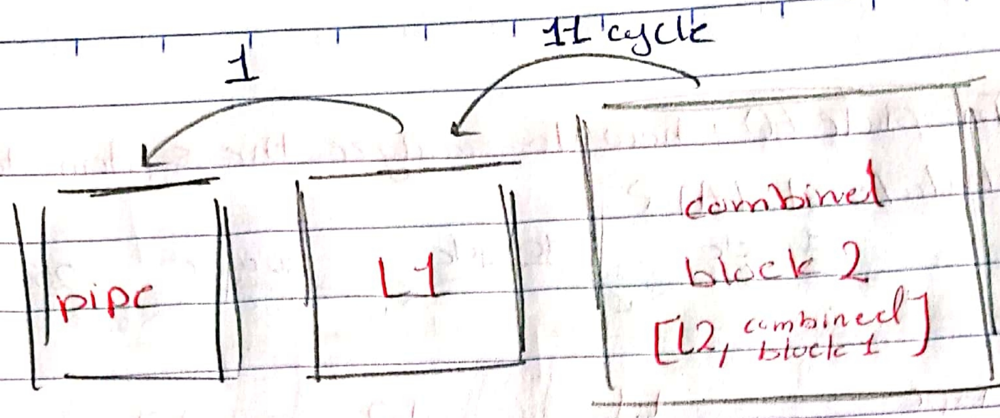
miss rate L2 = 4%

penalty (combined to L2) = 25

→ AMAT = 10 + 0.04 * 25 = 11 cycles

change the system

/ /



step ③ : consider : pipe → L1 → combined block 2

hit time L1 = 1

miss rate = 6%

panelty [combined block 2 to L1] = 11 cycles

AMAT = 1 + 0.06 * 11 = 1.66 cycles

So Average Memory Access time from CPU view for cycles and times ✓ = 1.66 cycles

analyze this number : 1.66 ✓

ideal case : hit everything in L1 = ~~1cycles~~ 1 cycle

worst case [should reach RAM] = 1.66 cycles on average
↓
hit nothing

Worst of the worst [no caches] = 250 cycles to go to RAM and load data from it

وجود ال cache حول الرقم من 250 ل 1.66 وقربها من ال ideal

in the mecusurment.

→ the more we added levels of caches the more we could reach ~~more~~ better AMAT, and closer to the ideal

This lead to the point of : Build large and fast memory hierarchy, but the two traditional designs we could have are:

Fast but small	large but slow
[pipe ↔ cache]	[pipe ↔ DRAM]

But, the hierarchical design of caches [levels of caches] could lead us to the point of large memory, but with fast performance, with some specifications like the example

slide 63

* Set associative

instead of dividing cache to blocks, divide to sets, and each set contains a group of blocks

Direct mapped

set associative

fully associative

Example:-

- 2-way set associative cache
- 8 locations only

How many sets? 4 sets

what if it is 4-way? 2 sets for 8 locations

slide 65, the cache:

V	Tag	word	
			0
			1
			2
			3

because only 4 blocks we have.

① first address: 0 mod 4 = 0

cycle #4

→ access cache index 0

→ cache is empty (V flag = 0)

go to RAM ← ↓ miss ✓

and load wanted data to location 0

② 2nd address: 8 mod 4 = 0

→ access cache index 0

→ V bit = 1

→ Tag of 8 ≠ Tag of 0

go to RAM ← ↓ miss ✓

and load data to location 0

③ address 0 again: 0 mod 4 = 0

→ access index 0

→ V bit = 1

→ TAG 0 ≠ TAG 8

↓

miss ✓

④ address 6 : 6 mod 4 = 2

→ access index 2

→ V bit = 0

load data from ← miss

RAM and put it to location 2

⑤ address 8 : 8 mod 4 = 0

→ index 0

→ V bit = 1

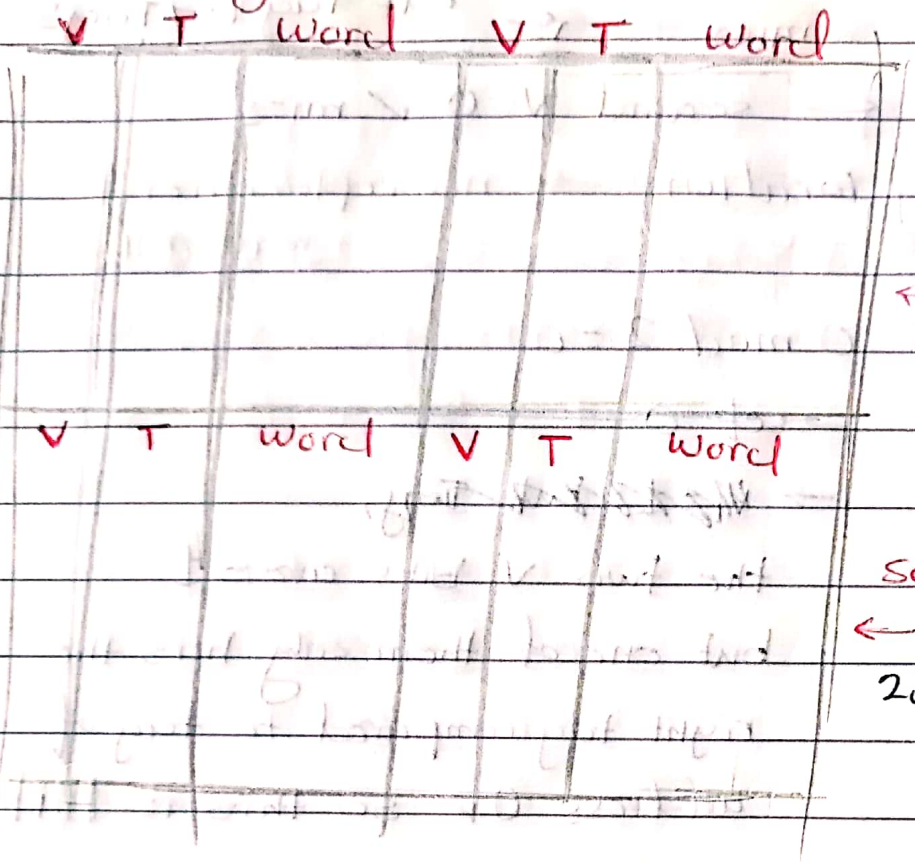
→ Tag 8 ≠ Tag 0

miss ✓

hit rate for this example for Direct mapped ? hit rate = 0%

Slide 68 :

- 2-way set associative



because we have 2 sets : we will divide on the number of sets in the equation, not the number of blocks [equation of mod]

① address 0 : 0 mod 2 = 0

→ access set 0

→ compare the two V bits in the set

↳ both are = 0 (cache empty)

← miss

go to RAM, and load the wanted data, to one of these words in the set (any one of them)

② address 8 : 8 mod 2 = 0

→ set 0

→ compare first V = 1 ✓ but Tag 0 ≠ Tag 8

miss ← second V = 0 ✓ miss

load from RAM ← to this empty location → no replacement required

③ address 0 : 0 mod 2 = 0

→ set 0

→ ~~Hit Bit Tag~~

the two V bits are = 1

but one of them only has the right tag compared to tag of address 0 ✓ so this is Hit

④ address 6 : 6 mod 2 = 0

→ set 0

→ V = 1 V = 1

load from RAM ← Tag 6 ≠ Tag 0 Tag 6 ≠ Tag 0

and we will assume miss ✓

to replace the least recently used

data (قديمة) → This is the location of Mem[8]

replace it!

⑤ address 8 : 8 mod 2 = 0

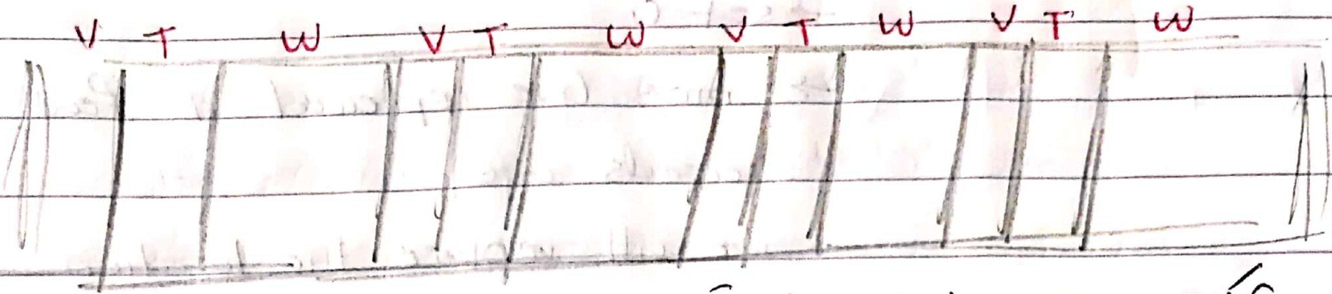
→ set 0

→ miss (we replaced it few seconds ago)

→ we will replace the location containing address 0 (the least recently used)

hit rate = $\frac{1}{5}$ = 0.2 = 20%

slide 66 : full associative



شكلها صاروا قطعة وحدة كلها : V, T, W بعدين V, T, W لأنه
ما راح نحتاج منها indexing

no need to use mod // no indexing or sets

① address 0 : → compare all V bits in cache (all are = 0)
miss ✓
→ load data from RAM to Randomly the first location ✓

← أول مكان فاضي لهاد

② address 8 : → first V = 1 , Tag 0 ≠ Tag 8
→ rest of V bits are = 0
→ load from RAM to randomly 2nd location.

← ثاني مكان فاضي لهاد

③ address 0 : → first V = 1 , Tag is correct ✓
Hit ✓

Scanned with CamScanner

(4) address 6 : → 1st V = 1 , Tag 0 ≠ Tag 6

→ 2nd V = 1 , Tag 8 ≠ Tag 6

miss ← → rest of V bits are = 0

↳ load from RAM to the first empty place (3rd location)

(5) address 8 : → 1st V = 1 , Tag 0 ≠ Tag 8

→ 2nd V = 1 ; Tag is correct ✓

hit ✓

hit rate = 40% = $\frac{2}{5} * 100\%$

law of diminishing returns

مثلاً: لو بدنا نطبقه على عمال بيشتغلوا هدفة ليخلصوا شغل معين، اذا حطينا نزيد عددهم لحد معين، بصير الشغل اسرع بدون تأثير سلبي، بس اذا زدنا عددهم عن هاد الحد بصير تأثير لزيادة سلبي وعلشان يكون ~~[illegible]~~ تخريبي

او بصير تحسين او سرعة انجاز الشغل مش كبير اله تأثير واضح (تحسين خفيف)، وهاد مقابل الكلفة المدفوعة على عدد العمال

design slide 68:

→ checking the bits of V and Tag of all locations in one set is done in parallel

→ because from each set we only load one word (one location), we don't need the block offset (we don't load the entire line)

→ 256 sets: needs 8 bits to identify the index of set

→ 22 bits are left for Tag

The Byte offset is related to the structure of DRAM, and how it is built.

→ when the structure was horizontally, (4 bytes next to each other, we didn't need to use the byte offset)

↳ word is 32 bits (4 bytes)

→ and when the DRAM is 32: byte offset = 2

" " " " = 64 : " " = 3

↳ word is 64 bits (8 bytes)

In the design of Direct Mapped:

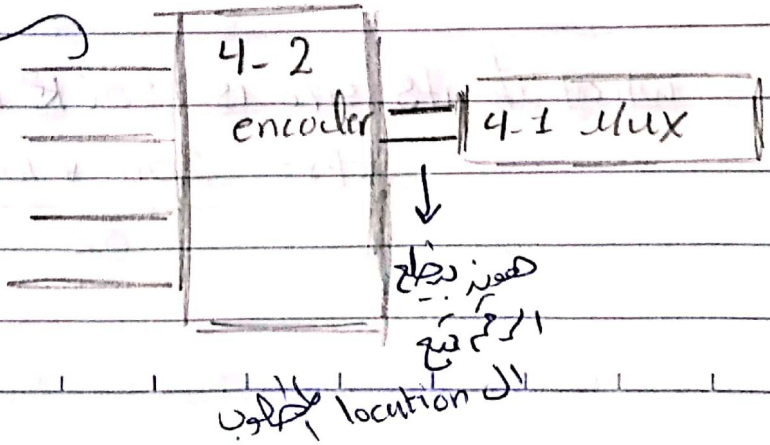
we used the block offset to supply it to the MUX that chooses the line (as a selector line)

→ But here in set Associative we don't have a block offset, now to identify the location needed?

→ We need 2 lines to select

But in design of slide 68: 4 lines are getting into the mux as the selection line, why?

it may be wrong or, we have an encoder



Assume the design is for fully associative, Find differences and cost!

→ there will only be one line // as we have one set
→ mux will be wider to handle all these locations at the same time

Example:-

cache → 8-way // each location is of 1 word
1 word = 4 byte
set has : 4 byte × 8 = 32 bytes

cache size : 32 k byte
how many sets

sets numbers $\frac{32 * 1024 \text{ byte}}{32 \text{ byte}}$

we have 1024 sets.

what if its size is 256 k byte

sets = $\frac{256 * 1024}{32}$ = 8k sets
= 8192 sets

what if its size 6MB and it is 12-way?

width is: 12 * 4 = 48 byte for each set

$$\text{sets} = \frac{6 * 1024^2}{48}$$

Why L3 is shared between all cores? slide 72

Two reasons!!

[1] Avoid Contention, what would happen if each one of them had its own L3 cache?!

→Basically the CPU is connected to the bus, to the DRAM, and because L3 cache is the last level then it is the one that CPU requests will pass to reach DRAM, then L3 is connected to the bus, then if each core had its own L3, then each one will have a connection point on the bus (shared bus), so contention will happen, because if a lot of cores tried to request something through the bus, L3 caches will compete to win the connection and pass the signal to DRAM

[2] parallel processing, such that alot of cores will work on one array for example in parallel, so it is better to keep it in a shared level like L3 cache, to keep up-to-date ~~with~~ with the updates coming from ~~the~~ other cores on this array.

Interactions with Software!! slide 77

elements and arrays are stored in two different ways in memory: -row major (bytes of elements are stored in row)
-column major (bytes of elements are stored in columns)

row like C++ and Java
column ,, MATLAB

→ Misses depend on memory access patterns (How the code affect the performance?)

Example: on C (row major)

```
for (int i=0; i<N; i++) {        // outer loop iterates over rows
    for (int j=0; j<M; j++) {    // inner loop iterates over columns
        arr[i][j] = rand() % 10; } }
```

access arr[0][0]: miss in cache
↳ load arr[0][0]: and a lot of its adjacents due to special locality, then a lot of elements will be hit
This code benifits from this special locality because it is row major (iterates over rows) → the code is written to do that.
and the language is row major itself.

How?
- hardware is row major // identical to language
- when cache loads data from DRAM, it loads the adjacents in rows
- language is row major
- code is written to iterate over rows
- cache has the adjacent elements of the row
- very low miss rate

if one modification happened on code to iterate over columns not rows: arr[j][i] = rand() % 100;
we won't benifit the special locality, because in cache we have adjacents of a row, not a column, all elements will be missed!

To test the cache effectively and the DRAM, and test how much hits and misses are affected and reacting with code??

we have to use kind of large array such that it does not fit completely in level 3 cache so we are forced to go in some cases to DRAM + make the number of iterations big to some level such that it satisfies the same idea

The experiment of testing the system, is repeated for thousands of times to get the average of all results, which is better.

Section. 11 // Redundant Arrays of Inexpensive Disks

اكثر الاجزاء معرضة للتلف هي الاجزاء الميكانيكية، هي حالياً ال Harddisks

The trend now is going to use cloud instead of RAID systems: - for less electricity usage
- " " cooling systems need
- " " cost of paying for RAID physical devices

* But, it is not the best choice for private data (security)

→ parallelism increases bandwidth, for which you read a part of your data from each disk, which increases performance [this is done in parallel instead of taking all chunks of data serially one after the other, which takes more time]

Hot swapping?

have made electrical designs & interfaces that allow the hot swapping, such that you can remove one hard disk from the system, and replace it with another, without needing to shut down the whole system for a long time, which makes it down and not responding for frequent accesses [This is a problem]

Types & raids due to different designs.
RAID 0, RAID 1, RAID 2 ....

Main difference between RAID 2 and RAID 3

[2]

- split at bit level
- parity is stored on multiple disks

[3]

- split at byte level
- parity is stored on one disk only

Main difference between RAID 3, and RAID 4

[B]
- splitted at byte level

[B]
- at block level
blocks have big sizes
which is satisfying

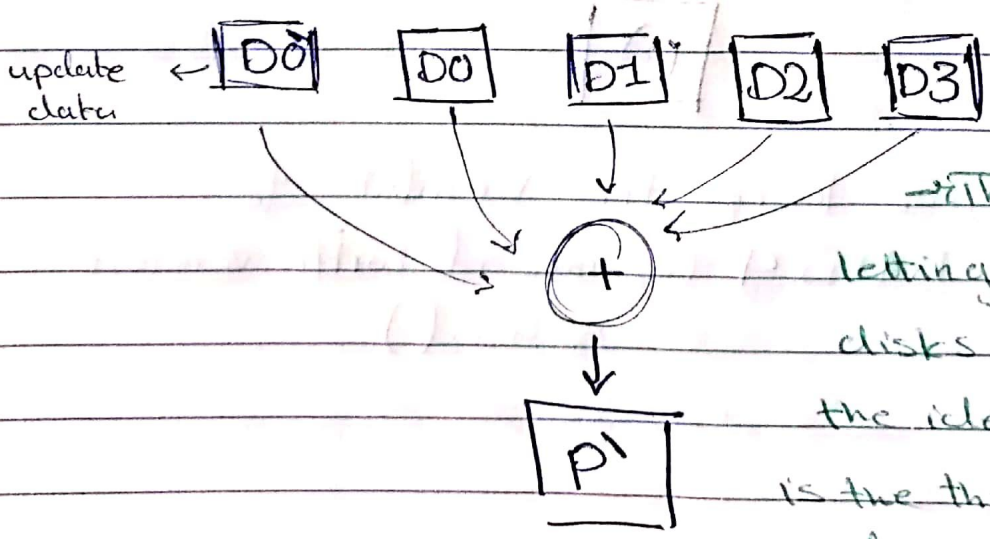
RAID 4 // Native

→ When to update data:

[1] D0' is put instead of D0
But parity disk should be updated too ✓

↓

Reuse all disks to recalculate P' through XOR circuit with the new data



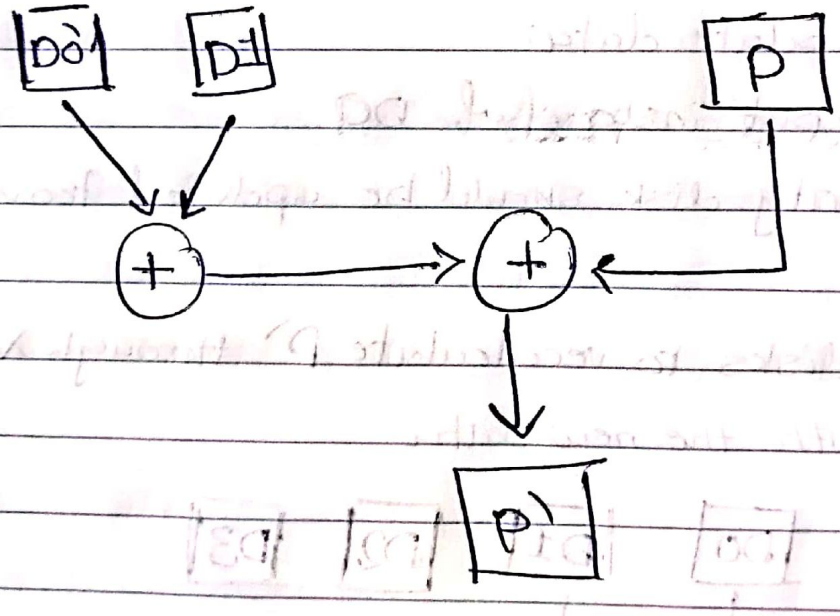
→ This requires letting all the disks work and leave the ideal mode, which is the thing that we tried to avoid by designing RAID 4 !!

This have a high cost on disks!!
all benifits are Gone!!!

RAID 4 // Optimize Write

→ no need to use the ~~completed~~ un-updated disks to recalculate the parity

you will have the same result by applying XOR between old data and new data, then apply XOR between the result of first XOR and old parity



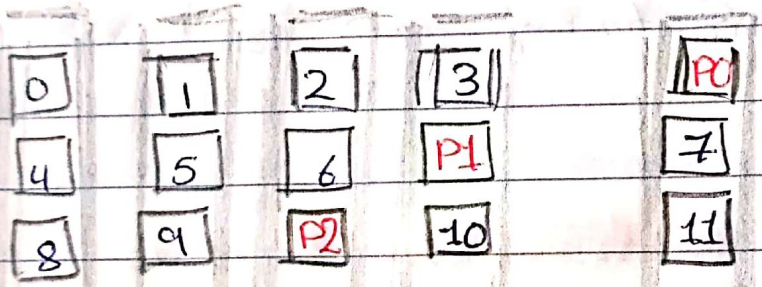
This way, we keep the benifit of designing RAID 4 (optimized write version is used)

Upon all the benefits of RAID 4 design, we have 2 new benefits come with RAID5 design.

1) if the parity disk failed, and got down for some reason

the diagonally distributed chunks of parity will find the lost data in parity disk

this disk is down!



P0 is lost
data 7 is lost
" " "
- you still can retrieve data 7 by P1, 6, 5, 4
- and retrieve data 11 by P2, 10, 9, 8 ✓

→ knowing that in some designs like RAID 3, if any one of data disk has fallen, we can retrieve all its data using the rest of data disks and parity disk
But the problem here, that if the fallen disk is the parity disk, the case when you can't retrieve any thing!

[2] we should not always use the parity disk to update data or retrieve it.

parity units are distributed to many disks, so whenever you need to use ECC for any reason you are not forced to access the parity disk necessarily which [if you are forced to do so] could decrease the life time of parity disk!

→ Problem of RAID is we store all the system in one place, so when some physical damage happen to that place, the entire system will be affected

to solve this problem, use Multiple RAID ✓

section 6#: Virtual Machines!!

advantages of virtual Machines:

[1] testing alot of programs on different operating systems only on one physical device (development advantages)

[2] Good for cloud computing, where you can use a shared hardware that provides the cloud and runs different VM for different OS of users [security + reliability]

↓

using a VM can isolate you safely and completely from other users or OSs

↳ when the OS stops responding suddenly, you can solve the problem by only shutting down the VM and restarting it again while other users & operating systems and VMs are still working

* These days VMs are supported by the hardware itself to accelarate it.

Virtual Machine Monitor

* The VMM does the job of mapping translation from the logical tasks of guest OS and real physical execution that should be done.

* User mode: Apps
* privilged mode: instructions dedicated for OS

* The problem is the Guest OS running on the device in user mode, while it is an OS and should be using the instructions of privilged mod

when it tries to do so, exception is thrown, and this exception is raised/traped to VMM as a request to be allowed to use the protected resources.

↑

This is another reason to dedicate parts of HW for virtual machines and guest operating system

* if one core is logically dedicated for the Guest OS, then the real physical execution can be entirely different after translation by VMM

Section 7: Virtual Memory

virtual ليه؟ machine

* Cache is managed by the physical gates of hardware

But when working with RAM, OS should be involved in the work to manage it with hardware support

Contents

RAID

Redundant Array of Inexpensive (Independent) Disks

- Use multiple smaller disks (c.f. one large disk) instead of
- Parallelism improves performance ✓ [Because same data has copies distributed to all these disks]
- Plus extra disk(s) for redundant data storage

Provides fault tolerant storage system →

- Especially if failed disks can be "hot swapped"

لو فشل واحد بنغير بديله على طول

↓

The modern standard.

if one of them fails, you need to change only that unit but not that whole entire system

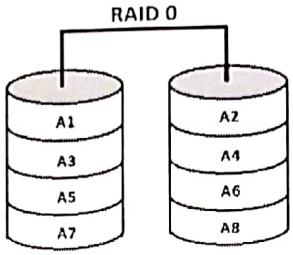
RAID 0 → Basically There is no RAID system

only a combination of ~~multiple~~ hard disk

internally: the data is distributed in away such that each chunck of data is stored in a different hard disk (There are no copies)

- No redundancy ("AID"?)
- Just stripe data over multiple disks → But it does improve performance

Data can be lost when there is failure.



advantages:
[1] reading data in parallel improves performance

disadvantage:
→ chunks of data can be lost with no Backup

advantage
[1] dependability: Backup for data
[2] less performance because you are forced to read serially always from one hard disk (data disk)

RAID 1 ↑

Have n number of hard disks anything written on one hard disk will be internally copied to other hard disks.

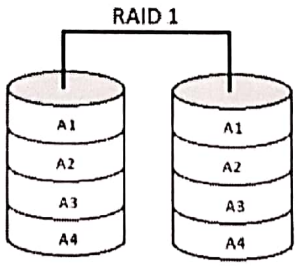
RAID 1: Mirroring

- N + N disks, replicate data

Write data to both data disk and mirror disk

- On disk failure, read from mirror → determined after RAID detection

* you only have access to data disk for writing or reading



* Gives you a time to replace the failing disk through hot swapping, and after replacing it internally, all the data updated on the mirror disk will be copied to the new data disk

So for example:
if a system is containing 2 of 6TB hard disks,
Then the system size will be 6TB, because you store 6TB on one of hard disks and the rest of them are Back up hard disks

you can't save ← more than 6TB

starting from the idea that fault tolerancy should include things more than dependability (Backups) and parallel access, things like flipping bits inside disk (which harms the data) should be solved

[Mechanically it is working, but the data is not correct] → logical tolerancy

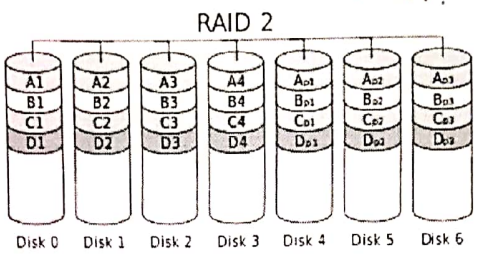
[store Error correction code with Data [ECC]

RAID 2

so RAID 2 has become

RAID 2: Error correcting code (ECC)

- N + E disks (e.g., 10 + 4) N for data disks, E for parity disks
- Split data at bit level across N disks
- Generate E-bit ECC → The parity
- Too complex, not used in practice ← !!!



when to read:
- read all disks together
- regenerate ECC
- compare it with stored ECC
- if something went wrong it corrects it
- the provides you with the data requested

Too complex because:
- it is complex to split at bit level
- bit level data requires you to let all the disks work to provide data.

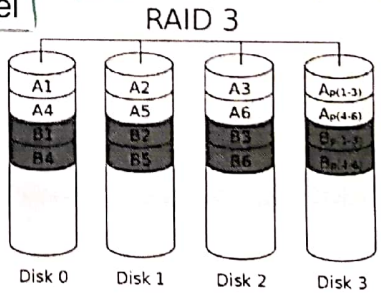
(high cost at all levels)
no one of them can stay idel for a short period of time

This increases the rate of disks fail to correct the wrong data.

RAID 3: Bit-Interleaved Parity

N + 1 disks → parity disk

- Data striped across N disks at **byte** level instead of bit level
- Redundant disk stores parity
- Read access
  - Read **all** disks
- Write access
  - Generate new parity and update **all** disks
- On failure
  - Use parity to reconstruct missing data



Not widely used (still alittle bit complex)

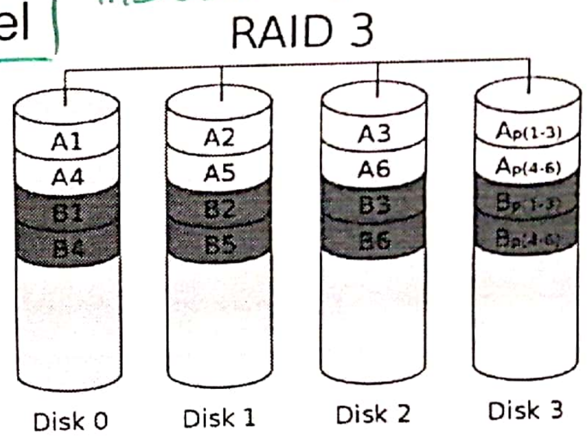
→ Have week points because of:
1. bytes splitting is still complex (not much better than bit level)
2. storing parity in one disk does not provide any dependability advantage so that parity can be lost at failure and

correct the wrong data.

RAID 3: Bit-Interleaved Parity

- N + 1 disks (1 → parity disk)
  - Data striped across N disks at **byte level** (instead of bit level)
  - Redundant disk stores parity
  - Read access
    - Read **all** disks
  - Write access
    - Generate new parity and update **all** disks
  - On failure
    - Use parity to reconstruct missing data

مش كتير اختلفت عن RAID2 هوه.



- Not widely used (still alittle bit complex)

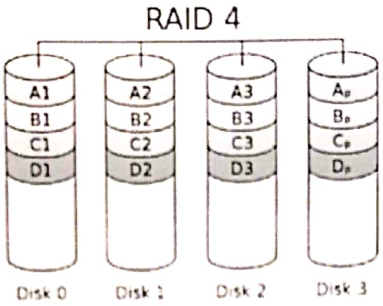
→ Have week points because of:

1. bytes splitting is still complex (not much better than bit level)
2. storing parity in one disk does not provide any dependability advantage, so that parity can be lost at failure and you lose the protection

RAID 4: Block-Interleaved Parity

N + 1 disks] like RAID3

- Data striped across N disks at **block** level
- Redundant disk stores parity for a group of blocks
- Read access
  - Read **only** the disk holding the required block



parity is calculated at block level.

mechanically better for hard disk to increase the time it will remain able to work

[1] Big sizes of blocks that are distributed to disks, allows that when you request some data, it all can be of size less than that block or equal to it, so you are not forced to let all the disks work to provide you the data, each time you are accessing some thing

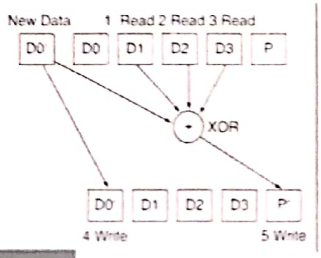
two designs !!

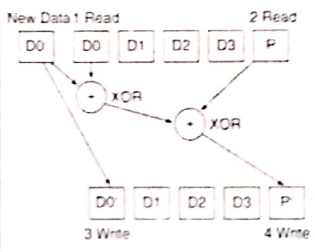
RAID 4: Naive vs. Optimized Write

[1] [2]

- Write access
  - Just read disk containing modified block, and parity disk
  - Calculate new parity, **update data disk and parity disk**
- On failure
  - Use parity to reconstruct missing data

Not widely used





A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

disadvantage:

data correcting that cares about fault tolerance is Gone

[1] having the parity entirely stored in one disk, what makes it hard to retrieve it when disk fails (data will be lost), so we lose the protection on data

RAID 5: Distributed Parity

improvement on RAID4 to solve the problem of storing the parity on one disk

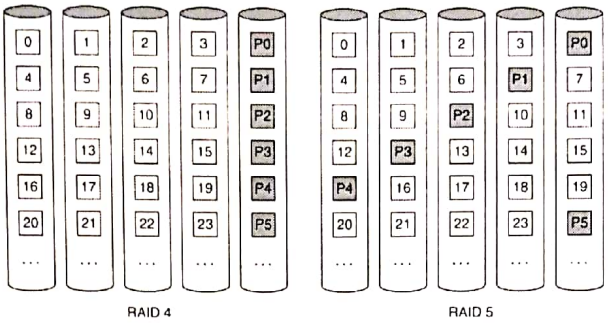
N + 1 disks

- Like RAID 4, but parity blocks distributed across disks
  - Avoids parity disk being a bottleneck

Widely used ✓

↓ Mixing

distributed in a diagonal way!



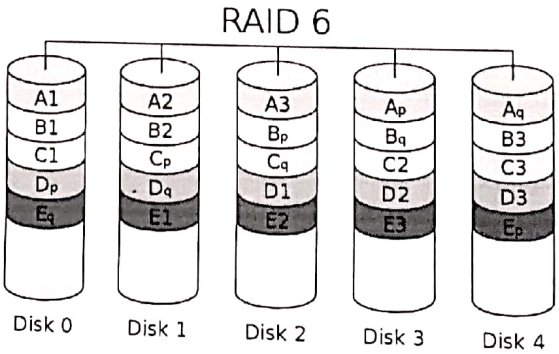
RAID 6: P + Q Redundancy

Same as RAID5 but with using more than one block for parity, which is also distributed diagonally.

N + 2 disks

- Like RAID 5, but two lots of parity
- Greater fault tolerance through more redundancy

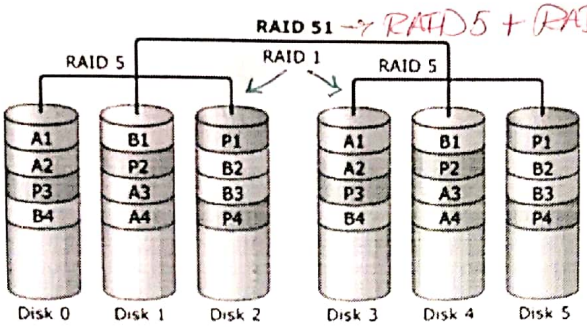
↓ stronger protection because of redundancy



Multiple RAID

Multiple RAID

- More advanced systems give similar fault tolerance with better performance
- Example RAID 51



RAID 1: to do replication with no parity generation

all the system here is copied to another system in different geographically place

locally: RAID5

Geographically: RAID1

RAID Summary

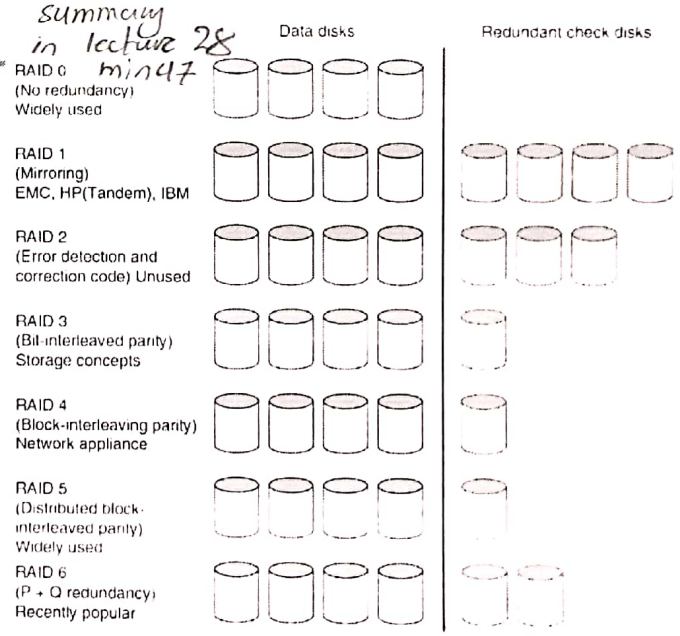
summary in lecture 28 min 47

RAID can improve performance and availability

- High availability requires hot swapping

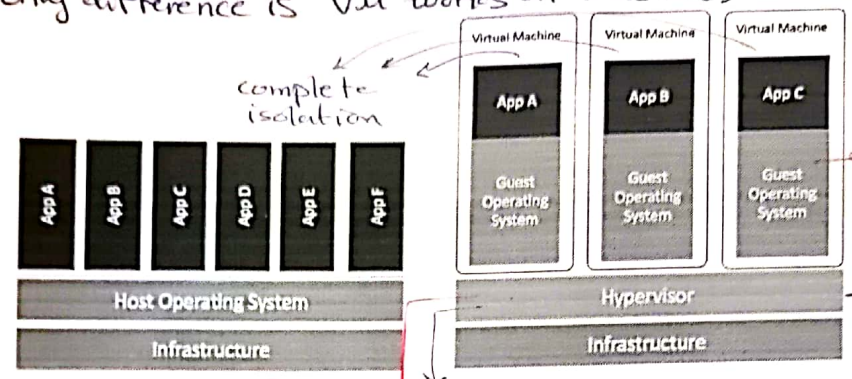
Assumes independent disk failures

- Too bad if the building burns down!



Virtual Machines

VM and Hypervisor are almost the same
the only difference is VM works on 1 device, while Hypervisor works on the cards



complete isolation

images of different OSs

The guest manager

run each App of a user on a Guest OS, inside a VM

The only problem is only when the server of Hypervisor gets down

* The single operating system (the Guest) does not have complete access on the host machine specifications completely, it just can access what you allowed it to access

Virtual Machine Monitor

→ dedicating resources for that Guest operating system

Maps virtual resources to physical resources
- Memory, I/O devices, CPUs

Guest code runs on native machine in user mode
- Traps to VMM on privileged instructions and access to protected resources

Guest OS may be different from host OS

VMM handles real I/O devices
- Emulates generic virtual I/O devices for guest