Content

Eight Great Ideas

ملاحظة تبين إنه كل سنة أو سنة ونص بتضاعف عدد الTransistors اللي بنقدر نحطهم على 1 integrated circuit

§1.2 Eight Great Ideas in Computer Architecture

← مؤسس شركة intel

- Design for ***Moore's Law***

← الاشي المعقد تخليه أبسط (تخلي الأسهل هو المكشوف ليسهل التعامل).

- Use ***abstraction*** to simplify design

← الحالة اللي بنستخدمها كثير نخليها أسرع . (ما بسرع الكل عشان التكلفة والغلبة)

- Make the ***common case fast***

← تقسيم الشغل على أكثر من شخص ، توزيع الشغل عأكثر من طريق)

- Performance *via* ***parallelism***

← الشغل على أكثر من مهمة عالتوالي

- Performance *via* ***pipelining***

← الشغل بناء عالتنبؤ المستقبلي

- Performance *via* ***prediction***

← لما أبني ال memory ما بخليها شيئ واحد ، (بقسمها)

- ***Hierarchy*** of memories

← قنيش الحسابات اللي بعملوا بنثق فيها و قديه بنعتمد و بنوثق فيه

- ***Dependability*** *via* redundancy

← وجود بديل لو خرب اشي في اله

بديل مثل وجود مروحتين تبريد

بعطونا high → Perf.. بنقدر ننجز أكثر من اشي

→ cash ما بنحط كل شي بال

1: كل ما كبر الشيئ بصير أبطئ

2: التكلفة عالية جدًا

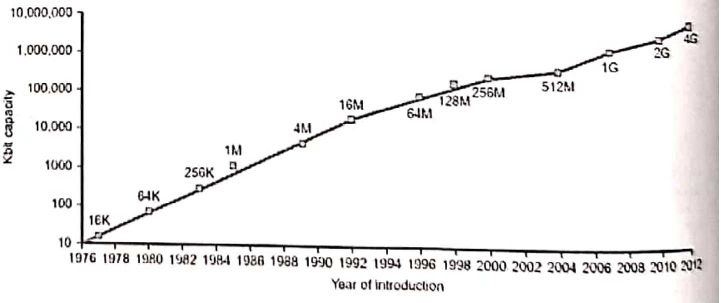
Content

بحتاج لتحقيق higher Performance : وجود تكلفة و دوائر لعمل المهمات
وبعالم الحواسيب هاي الدوائر لازم تكون أكبر وبتكون أغلى وأسرع .

Technology Trends

* Vaccum tobe فيها transistor واحد بينما ال chip فيها أكثر من transistor *

- Electronics technology continues to evolve تطور
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

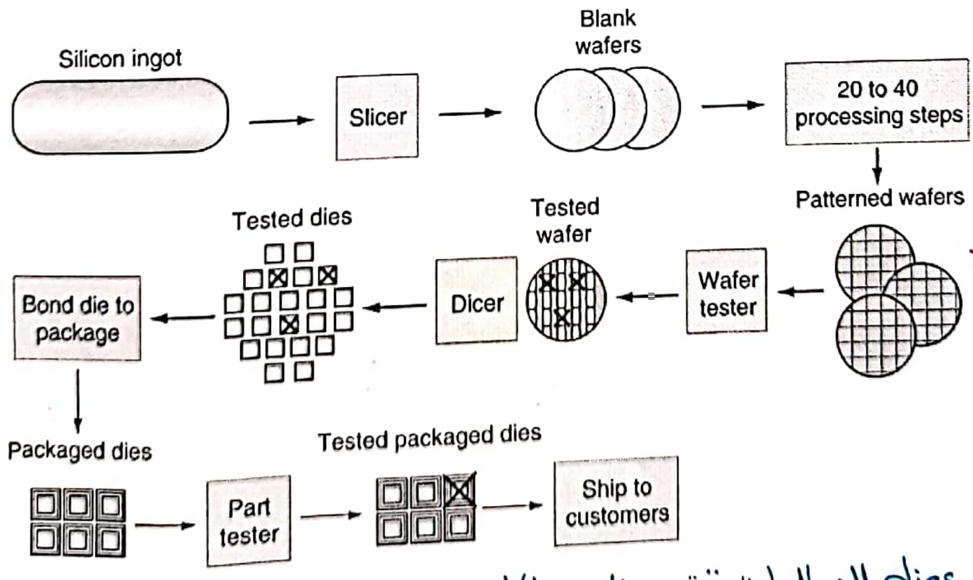
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor ← المادة الرئيسية
- Add materials to transform properties: ووظيفتها
  - ↳ Conductors موصل
  - ↳ Insulators عازل
  - Switch مفتاح

تصنيع Manufacturing ICs → "كيف بنعمل"

" مكلفة عشان هيك حاليا شركات معينة اللي بتعملها بس "

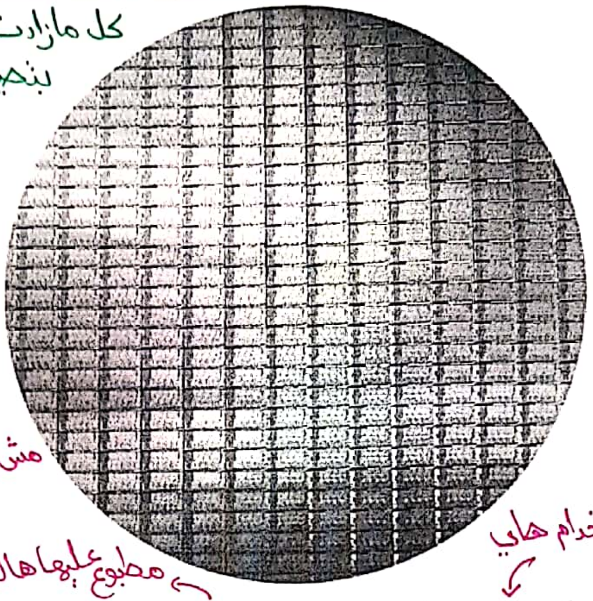


* كل ما يكون عندي خبرة عدد القطع الخربانة بتقل فال Yield بصير أعلى فالتكلفة بتقل و ما الك ذلك .

- Yield: proportion of working dies per wafer
  نسبة
  ← عدد ال dies الصالحات تقسيم العدد الكلي
  لأنه بال wafer ممكن يكون فيه dies معطلة .

Intel Core i7 Wafer

كل ما زادت مساحة ال die
بنصير تكلفتها أعلى



مش كلهم ممكن يكونوا صالحين
↑
القطر ← مطبوع عليها هالقد ←

باستخدام هاي ↙

بنحدد فيه ال transistor
صغار أو كبار .

- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

↓
أبعادها

كلماكان الرقم أصغر
ال transistor بكونوا أقل .

(ممكن يكون تربيعي ، linear ولكن مش ← cost $\overset{\text{يعتمد}}{\propto}$ area)

Integrated Circuit Cost

ال cost ← العلاقة بين

$$\overset{\text{لكل}}{\text{Cost per die}} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

عدد ال Chips ← Dies per wafer ≈ Wafer area/Die area
بال wafer

مساحة الدائرة ← (Wafer area) ; مساحة مستطيل ← (Die area)

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

↓
بتطلع 280 تقريبًا
Chips

$$\text{Yield} = \frac{1}{(1+(\underline{\text{Defects}}\text{ per area}\times\text{Die area}/2))^2}$$

← منطقة فيها خربان transistor

* كلما كانت الكثافة أقل كلماكان ال Yield أفضل

له بالأصل أكبر من 280
بس لأنه اللي
جايين عأطراف
الدائرة مش كاملين
فبنلغيهم

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed ثابت
  - Defect rate (معدل الخلل) determined (تحدد) by manufacturing process
  - Die area determined by architecture and circuit desi

$$* \text{ Defect Per area} = \frac{\#\text{ of defects}}{\text{wafer area}}$$

Content

1.2 Eight Great Ideas in Computer Architecture (*Review*)

1.5 Technologies for Building Processors and Memory

1.6 Performance (*Review*)

1.7 The Power Wall

1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors

1.9 Real Stuff: Benchmarking the Intel Core i7

1.10 Fallacies and Pitfalls

1.11 Concluding Remarks

Response Time and Throughput

- Response time
  - How long it takes to do a task → قديه وقت لأنفذ برنامج أو instruction
- Throughput
  - Total work done per unit time → قديه بنقدر نعمل شغل بوحدة الزمن
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

استجابة

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y" « Performance = $\frac{1}{T}$ »

لـ يهمنا يكون أكبر ما يمكن

$$\text{Performance}_X / \text{Performance}_Y$$

$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

يهمنا يكون أقل ما يمكن

لـ لما اشتغل عال Pipelining أو Parallelism مش دقيقة فبحس ←

عدد الـ Parallel units

ومش واحد لا P ↑

Throughput = $\frac{P}{T}$

Time للعملية الواحدة

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

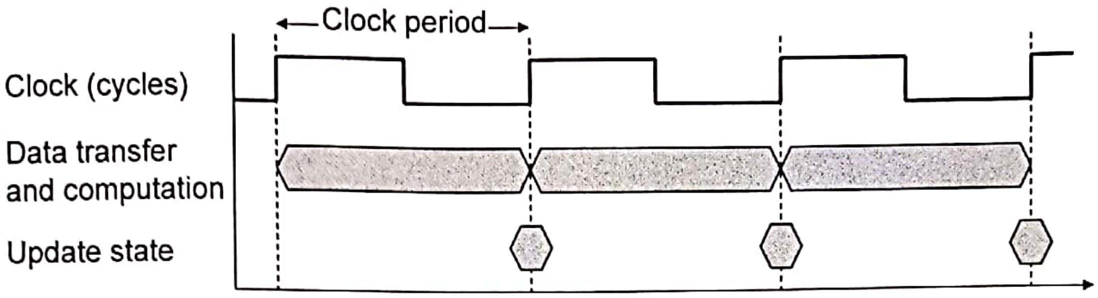
Measuring Execution Time

- منقضي Elapsed time
  - Total response time, including all aspects النواحي
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time الوقت المستغرق في معالجة وظيفة معينة
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

يتحكم

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instructio n Count} \times \text{Cycles per Instructio n}$$

instruction قديه فيه

↳ زمن ال cycle الوحدة

أهم معادلة

$$\text{CPU Time} = \text{Instructio n Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

↳ clock per instruction

$$= \frac{\text{Instructio n Count} \times \text{CPI}}{\text{Clock Rate}\ (f)}$$

$f = \frac{1}{T}$ ←

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

↳ وقت لتخلص task

Ex : Time = 0.1 seconds / CPI = 0.5 / IC = 400 Million Instructions

↳ بقدر ينفذ 2 instru.. في ال cycle الوحدة

(f = ??)

Sol : $0.1 = 400 \times 10^6 * 0.5 * T$

$\rightarrow T = 5 \times 10^{-10}$

$f = \frac{1}{T} \rightarrow f = 2 * 10^9$

$= 2$ GH

CPI in More Detail

- If different instruction classes take differen numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

Performance Summary

*كل ما قل ال time زاد ال performance

The BIG Picture

IC كم instruc.. في البرنامج

$$\text{CPU Time} = \frac{\overset{\text{IC}}{\cancel{\text{Instructions}}}}{\text{Program}} \times \frac{\overset{\text{CPI}}{\cancel{\text{Clock cycles}}}}{\cancel{\text{Instruction}}} \times \frac{\overset{\text{T}}{\text{Seconds}}}{\cancel{\text{Clock cycle}}}$$

→ الوحدة اللي بتطلع cpu time

= قديه محتاج Seconds لينفذ program معين

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

كيفية تقليل : 1- $\frac{\text{Instructions}}{\text{Program}}$: بعتمد عالخوارزميات و كيف استخدموها للتقليل من ال Instructions بالعملية الوحدة.

2- $\frac{\text{Clock cycles}}{\text{Instructions}}$: تخلص أكثر من instruc.. بنفس ال cycle يعني نعمل Pipelining و هكذا.

3- $\frac{\text{Seconds}}{\text{clock cycle}}$: لو بدي أقلل ال Period لازم اشتغل على high frequency عن طريق استخدام Smaller transistors لانهم بصيروا أسرع

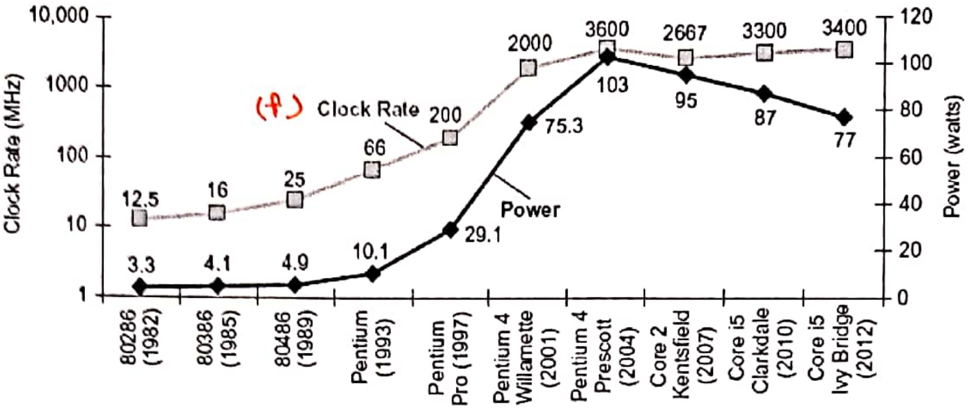
Content

- 1.2 Eight Great Ideas in Computer Architecture (*Review*)
- 1.5 Technologies for Building Processors and Memory
- 1.6 Performance (*Review*)
- 1.7 The Power Wall
- 1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors
- 1.9 Real Stuff: Benchmarking the Intel Core i7
- 1.10 Fallacies and Pitfalls
- 1.11 Concluding Remarks

Power Trends

حنحكي عن استهلاك الطاقة الكهربائية

* ال Processors بستهلكوا طاقة كهربائية وهاي الطاقة مع الزمن بتزداد *



بشحن و بفرغ

↑

- In CMOS IC technology

capacitors بال transistors ما في تيار بمر فيهم فبنتظر الى إلهم ال

ال Power تتناسب طرديا معها

↑

اللي بسبها ← $\text{Power} = \frac{1}{2}\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$

Dynamic Power

(بعتمد على: حجم العدد ل transistors)

×30 حجم ال transistor

كلما قل بقل ال Power

يعني كل ما زاد ال transistor ← عددهم بقل حجمهم بقل ال Power

5V → 1V

له من 5 ل 1 وهلا أقل من 1 بشوي وما بنزل أكثر من هيك

×1000

له بتزداد ألف مرة

* ال static Power دايما ال chip بتستهلكها * = supply voltage * current leakage

* بطلنا نزيد ال frequency عشان نقلل ال time لأنه كل ما زدنا ال frequency بتزداد ال Power

فصرنا ندور حلول أخرى لتقليل ال time. *

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further

ما بنقدر ننزله أكثر من هيك

  - We can't remove more heat

ما بنقدر نعمل remove لأكثر من 100 أو 150 watt.

- How else can we improve performance?

له بزيدوا ال Performance بالوقت الحاضر عن طريق ال Multiprocessor

Content

- 1.2 Eight Great Ideas in Computer Architecture (*Review*)
- 1.5 Technologies for Building Processors and Memory
- 1.6 Performance (*Review*)
- 1.7 The Power Wall
- 1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors → multiple cpu's
- 1.9 Real Stuff: Benchmarking the Intel Core i7
- 1.10 Fallacies and Pitfalls
- 1.11 Concluding Remarks

التغيير الكبير

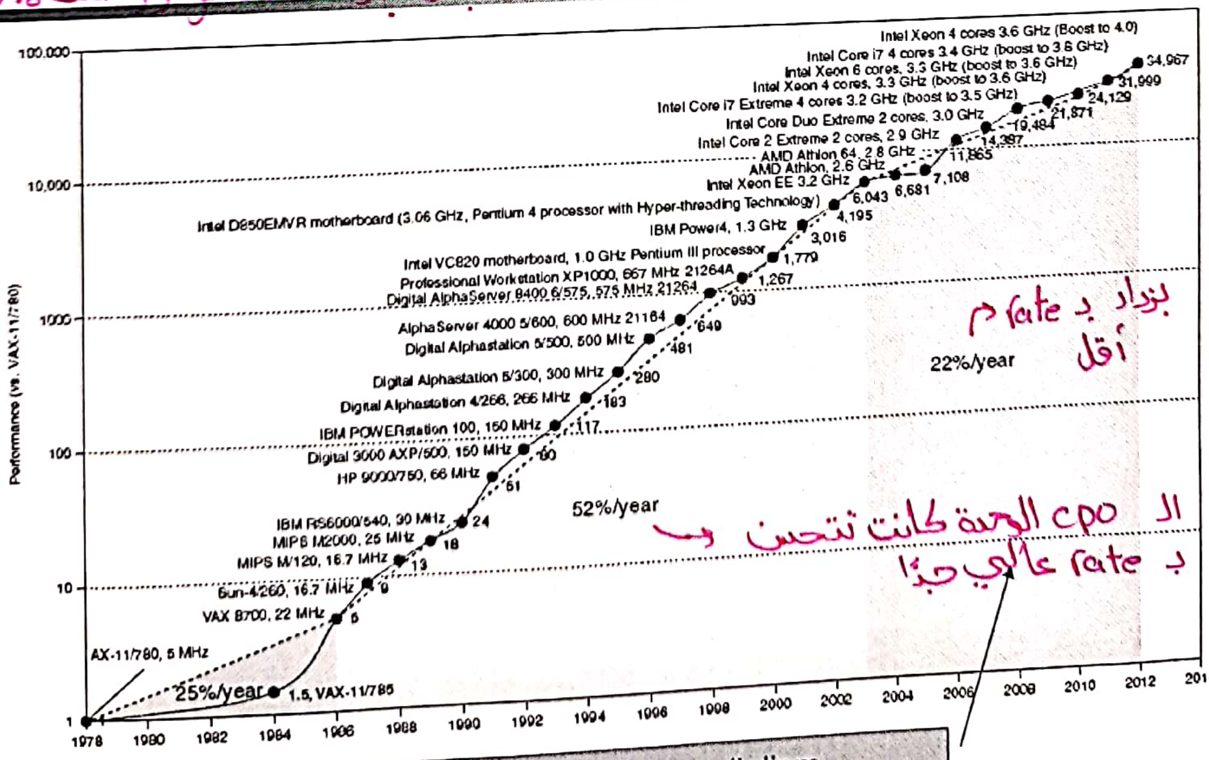
بال chip الوحدة

لأنه بطلنا نقدر نقلل ال voltage أقل من هيك ، بس الحل اجى بزيادة ال # of logical cores

Uniprocessor Performance

قدرتنا على تحسين ال Performance ل one Prosessor بتقل بالوقت الحاضر ووصلت 3.5%

§1.8 The Sea Change: The Switch to Multiprocessors



بزداد بـ rate أقل

الـ cpo الوحدة كانت تتحسن بـ rate عالي جدًا

Constrained by power, instruction-level parallelism, memory latency

* الفرق بين : Processor : الوحدة الحاسبة بال Computer

Multiprocessors

Microprocessor : برضه الوحدة الحاسبة بال Computer لكن شرط انها تكون على one chip

- Multicore microprocessors
  - More than one processor per chip

← لازم نلجأ اله لنستفيد من Multicore

- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware ~~executes~~ بنفذ multiple instructions at once
    - مخفي Hidden from the programmer

← حل في عبء عالمبرمجين ، إنهم يعيدوا كتابة برامجهم حتى تصير Parallel program حتى تستفيد من ال Multiple cores.

  - Hard to do
    - Programming for performance

← لإنه برامجك لازم تكون صح ولازم تعطيك Performance أعلى

    - Load balancing
    - Optimizing communication and synchronization

← نخليه قليل وسريع

← لانهم بحتاجوا وقت ليتفاهموا مع بعض

بدك تقسم العمل على أجزاء متساوية بحيث ال Multiple cores كل core ياخد شغل قد التاني لإنه لو واحد أخذ شغل أكثر من غيره رح ياخذ كل البرنامج ولازم يستنوه لحتى يخلص

Content

- 1.2 Eight Great Ideas in Computer Architecture (*Review*)
- 1.5 Technologies for Building Processors and Memory
- 1.6 Performance (*Review*)
- 1.7 The Power Wall
- 1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors
- 1.9 Real Stuff: Benchmarking the Intel Core i7
- 1.10 Fallacies and Pitfalls
- 1.11 Concluding Remarks

→ Benchmark نوع من أنواع ال

SPEC CPU Benchmark

→ "طرق قياسية تستخدم للمقارنة بين جهازين أو تصميمين وهكذا."

- Programs used to measure performance
  - Supposedly typical of actual workload → يمثلوا ال Workload الحقيقيات ← Benchmark ال
- Standard Performance Evaluation Corp (SPEC) → منظمة
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006 → فعالة وفي قبلها كان وفي بعدها 2017
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance → "SPEC بتركز cpu"
  - Normalize relative to reference machine → $\frac{T_{R_i}}{T_i}$
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

→ Summerising Perf.- هو اللي بنستخدمه بـ

→ Selection of program
43 يعملوا الك
تم اختيارهم لأنهم
Typical of actual workload

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i} \rightarrow \frac{T_R}{T_i}$$

مضروب

T_R → وقت ال older reference machine

T_i → الوقت تاعك

يعني → cpu integer

CINT2006 for Intel Core i7 920

→ T_i كلهم شغلوهم ليحسبوا

T_i ↓ ↓ T_R

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

↓ Integer app

↓ برمجيات مختلفة.

→ نفس عددهم

الناتج ←

بضربهم ببعض كلهم بعدين بياخذ الجذر ال 12 لالهم ←

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

ال Power مهمة لإنها *بتستهلك مال وكل ما زاد ال Power بتزيد فاتورة الكهرباء
* وللحفاظ على الطبيعة
* وكشان البطارية تعيش

Power benchmark

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

مجموع
مجموع

* The SPEC cpu 2017 benchmark package contains 43 benchmarks, organised into four suites:

1- SPEC speed integer and SPEC speed floating] are used for comparing time for a computer to complete single tasks

يعني لو بدي أشوف الجهاز قديه بياخد وقت لمهمة وحدة

2- SPEC rate integer and SPEC rate floating] measure the throughput or work per unit of time

يعني لما بدي أشغل أكثر من مهمة بنفس الوقت

تستخدم بـ general app مثل compiler أو Sorting أو Exel وهكذا. يستخدم للبرمجيات اللي بتعملنا أمور مختلفة

غالبية الحسابات العلمية والهندسية تستخدمها.

SPECpower_ssj2008 for Xeon X5650

operation per second ← وحدته

← بتشغل ال Processor على different target load

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

← كل ما كان أكبر كل ما كان أفضل

كل ما زاد ال target load بزيد ال average Power لأنه : بصير في شغل وكل ما زاد بزيد الاستهلاك للطاقة .

Content

- 1.2 Eight Great Ideas in Computer Architecture (*Review*)
- 1.5 Technologies for Building Processors and Memory
- 1.6 Performance (*Review*)
- 1.7 The Power Wall
- 1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors
- 1.9 Real Stuff: Benchmarking the Intel Core i7
- 1.10 Fallacies and Pitfalls
- 1.11 Concluding Remarks

←عالم computer eng قديم

Pitfall: Amdahl's Law

§1.10 Fallacies and Pitfalls

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

تقسم الـ exec. time

بالـ Taffected

و Tunaffected

الزمن الجديد ←

بنعدل عليه ←

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvemen t factor}} + T_{\text{unaffected}}$$

↳ مانعدل عليه

- Instructions ← Example: multiply accounts for 80s/100s → بياخدوا وقت 80 من أصل 100
  - How much improvement in multiply performance to get 5× overall?

↳ $5 = \frac{T_{old}}{T_{improved}}$

$$20 = \frac{80}{n} + 20$$

- Can't be done! speedup ←

$5 = \frac{100}{T_i}$

$\rightarrow T_i = 20$

- Corollary: make the common case fast

Scanned with CamScanner

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

Content

1.2 Eight Great Ideas in Computer Architecture (*Review*)

1.5 Technologies for Building Processors and Memory

1.6 Performance (*Review*)

1.7 The Power Wall

1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors

1.9 Real Stuff: Benchmarking the Intel Core i7

1.10 Fallacies and Pitfalls

1.11 Concluding Remarks

Concluding Remarks

§1.11 Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

Chapter 4

The Processor

Adapted by Prof. Gheith Abandah

Contents

- 4.6 Pipelined Datapath and Control (Review)
- 4.7 Data Hazards: Forwarding versus Stalling
- 4.8 Control Hazards
- 4.9 Exceptions
- 4.10 Parallelism via Instructions
- 4.11 Real Stuff: The ARM Cortex-A53 and Inte
  Core i7 Pipelines
- 4.14 Fallacies and Pitfalls
- 4.15 Concluding Remarks

Contents

4.6 Pipelined Datapath and Control (Review)

- Five-Stage Pipeline
- Pipeline Control
- Pipeline Hazards

Five-Stage Pipeline

F: Fetch instruction from the instruction memory → بنجيب ال inst من inst. memory

D: Decode instruction and read operands → تحليل ال inst وقرائتها

E: Execute operation or calculate address → الحساب

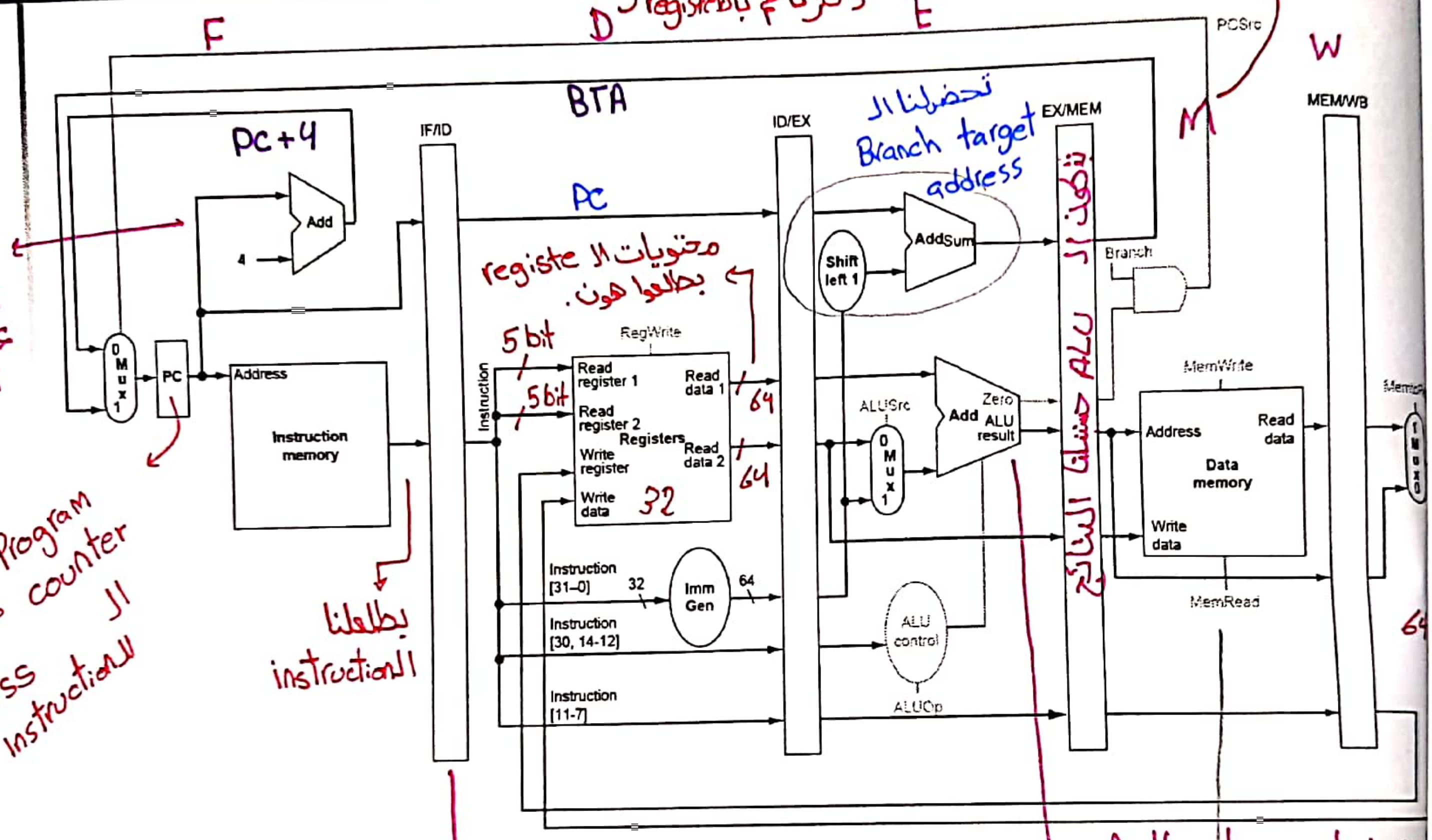
M: Memory access → اذا كانت Load / Store

W: Write result to the register

Five-Stage Pipeline

بنهاية بنكون قرينا ال operands وخزناهم بال register

بس ال load/store بنستخدم هاي المرحلة.



بتحضرلنا ال Branch target address

تكون ال ALU حسبتلنا النتائج

محتويات ال register بطلعوا هون.

adder بودينا عال instruction الي بعدها

Program counter ال address الي فيه ال instruction

بطلعلنا ال instruction

Register → بحمل النواتج من مرحلة وبستخدمهم ك input للمرحلة التي تليها.

Arithmetic and logical unit

كل instruction فيه 32 bit يعني 4 byte

Chapter 4 — The Processor —

لوكانت العملية load بنقرأ

لوكانت العملية store بنكتب

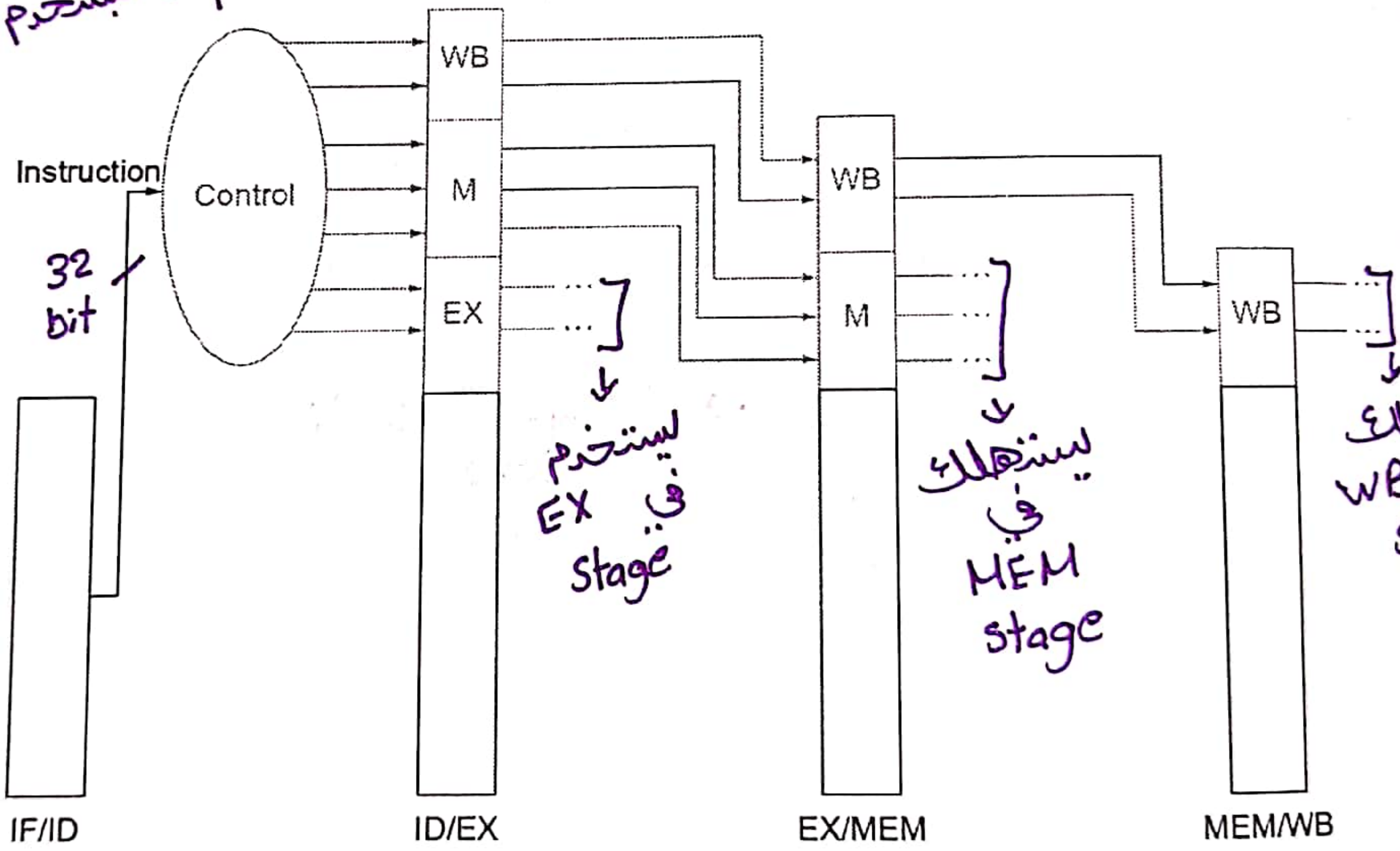
وهاي ال data بتيجينا من Read data 2

Pipelined Control

- Control signals derived from instruction

- Control signals derived from instruction
  - As in single-cycle implementation

opcode الي هو 7-bit بستخدم ←
لنطلع ال control
signals



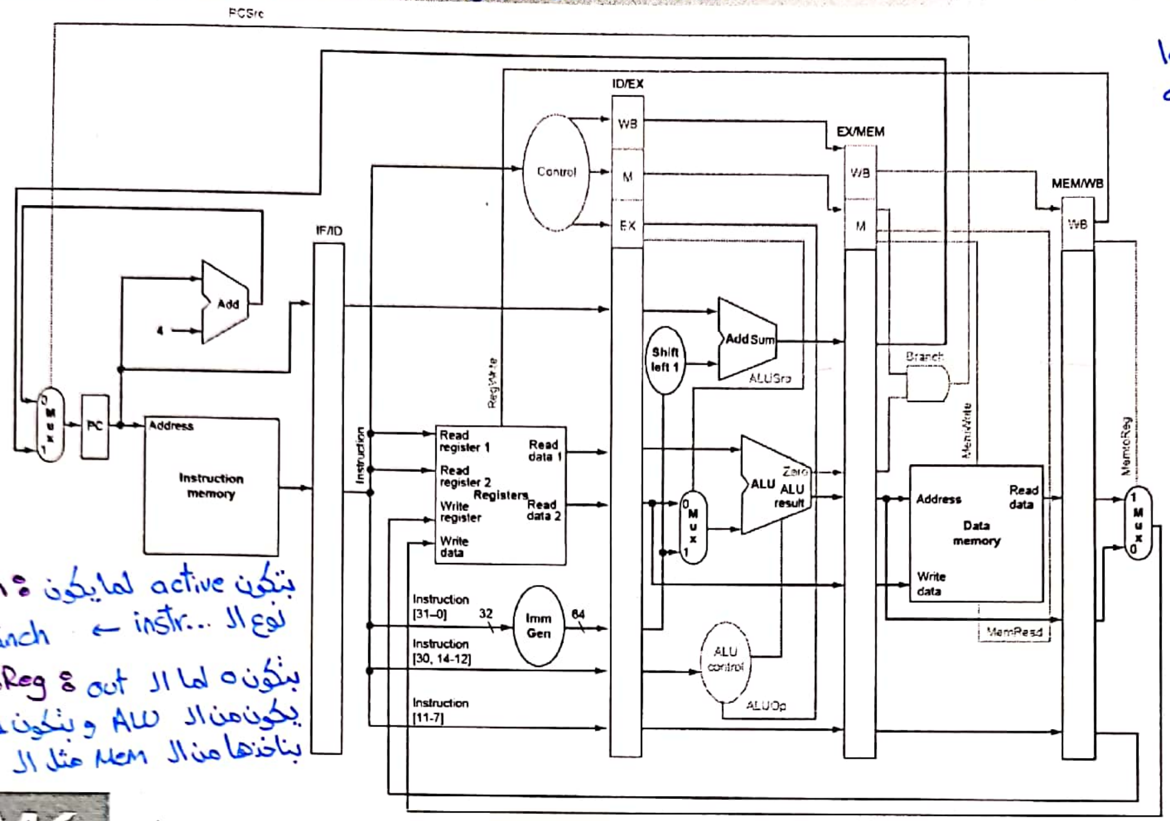
يستخدم في EX stage

يستهلك في MEM stage

يستهلك في WB stage

Pipelined Control

*Regwrite : بتكون 1 عشان بدنا نكتب عالregister و 0 لما مابدنا نكتب مثل : الـ Branch مابنكتبوا عالreg ومثل الstore كمان
↓
مثل الload
و add/sub



*Branch : بتكون active لمايكون نوع الـ instr... ← branch

*MemtoReg : بتكون 0 لما الـ out يكون من الـ ALU و بتكون 1 لما الداتا بناخذها من الـ Mem مثل الـ load

*ALUsrc : لو كانت 0 فال ALU بجيبوا الداتا من الـ register
و لو كانت 1 بدخل الـ Imm instruction

*ALUop : عبارة عن 2bits بتختارلي العملية اللي تعملها الـ ALU

Hazards

*Mem Write : بتكون 1 لما بدنا نكتب عالMem مثل الinstruction : store
*Mem read : بتكون 1 لما بدنا نقرأ من الـ Mem مثل الinstruction : load

- Situations that prevent starting the next instruction in the next cycle

*الفائدة الأساسية من الـ Pipelining : لتحسين الPerformance

1 Structure hazards → في مشكلة بالـ organisation ← توقف خط العمل
  - A required resource is busy → بنزيد عتصميمنا التحسينات

2 Data hazard
تأخر اللي بتقرأ الـ data عبين ماتخلص اللي بتكتب
  - Need to wait for previous instruction to complete its data read/write

3 Control hazard → الهم علاقة بالـ jump والـ Branch
  - Deciding on control action depends on previous instruction

Contents

- 4.6 Pipelined Datapath and Control (Review)
- 4.7 Data Hazards: Forwarding versus Stalling
- 4.8 Control Hazards
- 4.9 Exceptions
- 4.10 Parallelism via Instructions
- 4.11 Real Stuff: The ARM Cortex-A53 and Inte Core i7 Pipelines
- 4.14 Fallacies and Pitfalls
- 4.15 Concluding Remarks

Contents

4.7 Data Hazards: Forwarding versus Stalling

Hardware Solutions [
- Data Hazards in ALU Instructions
- Load-Use Data Hazard

Software Solutions [
- Code Scheduling

Data Hazards in ALU Instructions

- Consider this sequence:

register instructions كلهم ماعدا ال store

```
sub   x2, x1,x3
and   x12,x2,x5
or    x13,x6,x2
add   x14,x2,x2
sd    x15,100(x2)
```

* store : read from register and write to memory

* load : read from memory and write to register

offset ↲ ↳ base register

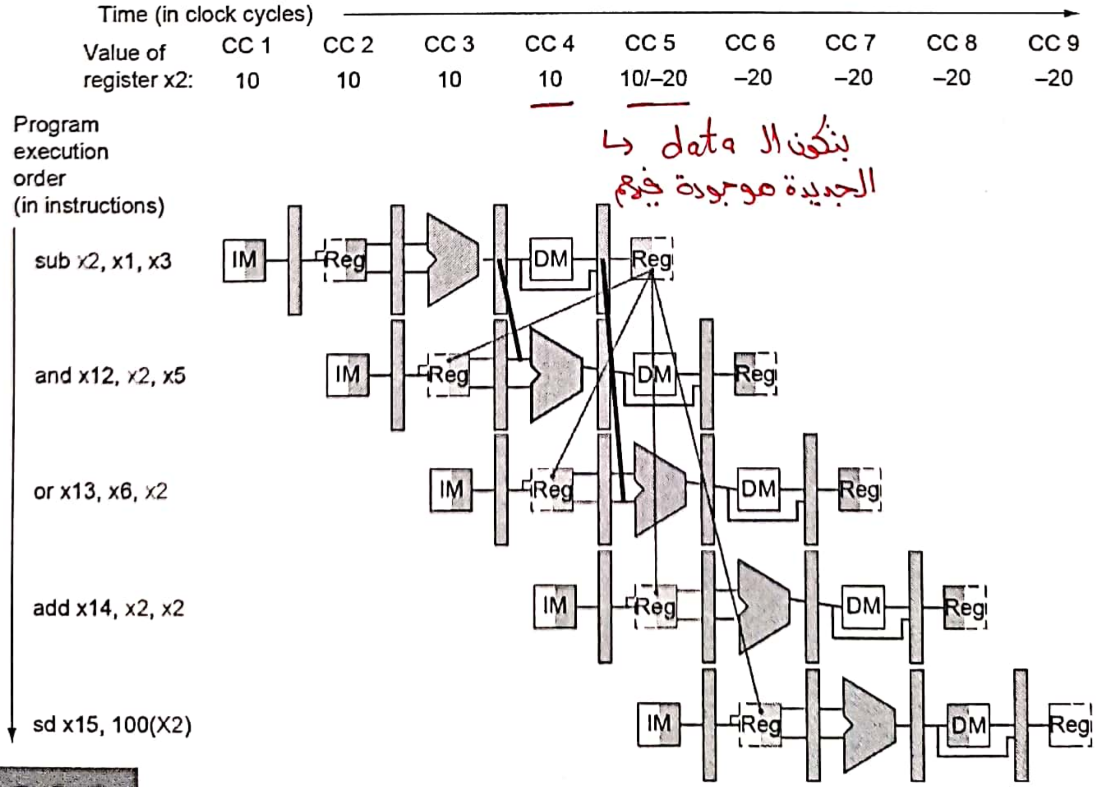
كـ نبقرا من ال reg محتويات x15 وبدنا نخزنها بال Mem وال Mem مكانه = محتويات x2 + 100

- There are multiple true data dependencies read-after-write (RAW), on register x2.
- We can resolve hazards with stalls or forwarding.

*حل ال Hazards بطريقتين : 1- Stalls 2- forword

والحل الأفضل هو ال Forword لإنه بزيد ال Performence ويقلل ال Stalls ⤷ ولكن في زيادة hardware بس دنا كتير transistors فمش مشكلة

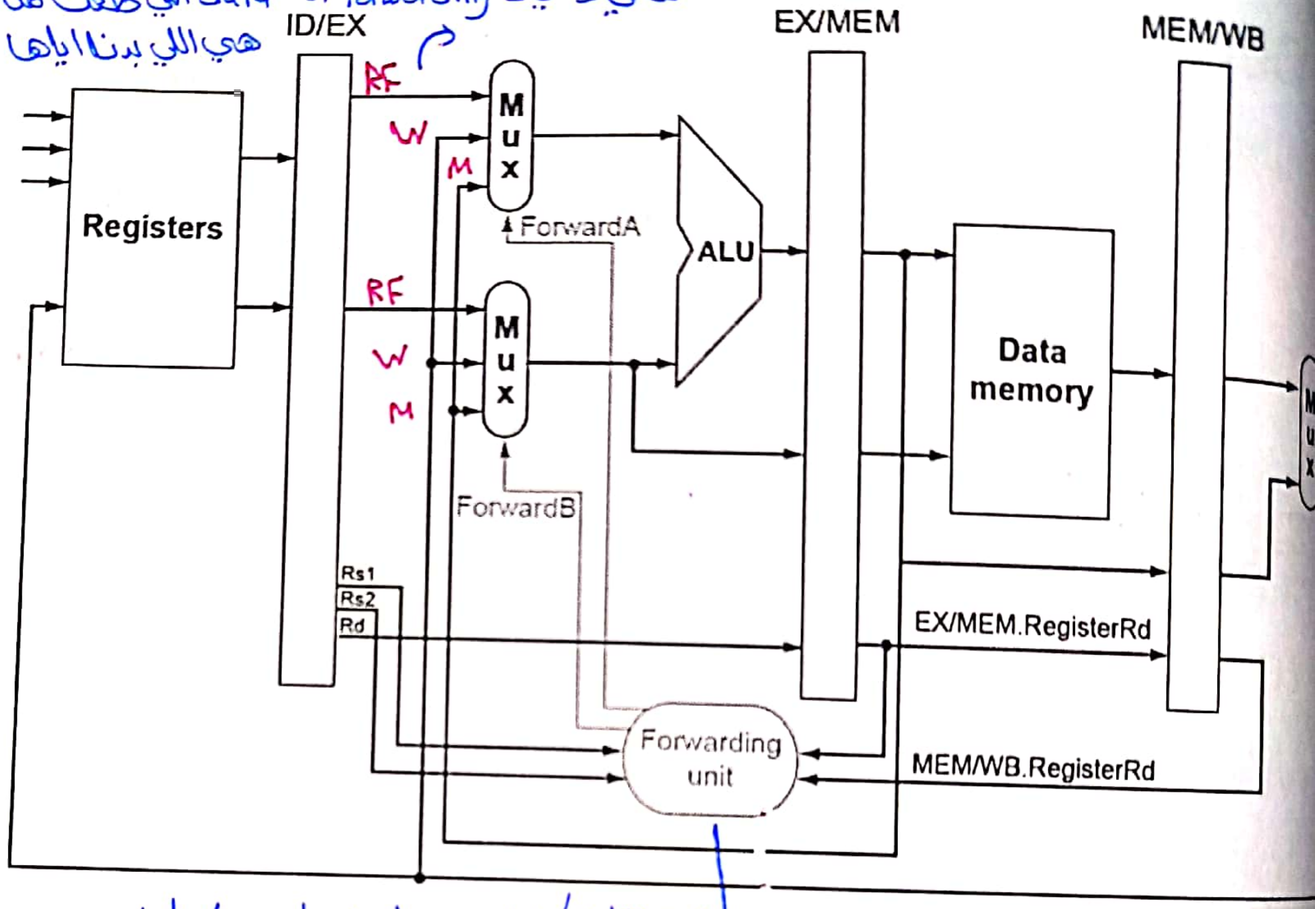
Dependencies & Forwarding



↳ بنكون ال data الجديدة موجودة فيهم

Forwarding Paths

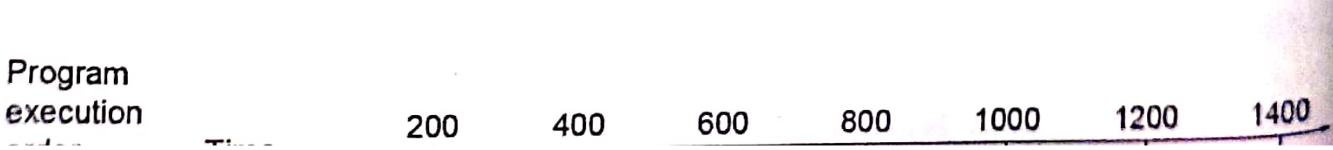
ما في داعي لل forwarding ال data اللي طلعت من ال reg file
هي اللي بدنا اياها



بيها تكتشف هل في data hazards
وهل في حاجة لل forwarding ولا
في حاجة بيها تعطي control signal
ملائمة لل MUX لحتى ناخد ال data من المكان الصحيح

Load-Use Data Hazard

- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



Slide 8 :

← عدد ال cycles

	1	2	3	4	5	6	7	8	9	10
I1	F	D	E	M	W					
I2		F	D	E	M	W				
I3			F	D	E	M	W			
I4				F	D	E	M	W		
I5					F	D	E	M	W	

*تسمى :
"Multicycles Diagram"
لإنه احنا بنتابع بأكثر
من cycle شو عم
يصير لل instructions

↳ 5 Instructions

*لومافي Stalls بنحتاج إلك 9 cycles لتنفيذ ال instructions = 5 + 4 = 9

↳ مجموع عدد ال cycles الزايدة بكل instruction

↳ عدد ال cycles اللي احتجهم لتنفيذ I1

*لوماكان عندي pipeline كان احتجت 5x5 = 25 cycles لتنفيذ ال instructions

Slide 11 : Assume no forwarding (except through the Register file) and hazards are solved by stalls

	1	2	3	4	5	6	7	8	9	10	
Sub X_2, X_1, X_3	F	D	E	M	W						
and X_{12}, X_2, X_5		F	D	D	D	E	M	W			
or X_{13}, X_6, X_2			F	F	F	D	E	M	W		
add X_{14}, X_2, X_7						F	D	E	M	W	
sd $X_{15}, 100(X_2)$							F	D	E	M	W

عملنا stall لـ 2 cycles
← بين ماحصلنا القيمة
الجديدة
↳ بنضل بال F بين مانخلص
مرحلة ال D اللي قبلها

*بال Stalls احتجت 11 cycles = 4 + IC + stalls

↳ كل instruction بدها 1 cycle لتنفذ ال W / عدد ال cycles الزيادة

↳ (ثابت) 4 cycles لحتى نعبي ال pipeline

*هون في 2 stalls بس لإنه ال stall اللي عند ال F لسببه نفس سبب ال Stall اللي صار عند ال D

*هاد الحل مو كويس لإنه نزّل من ال performence

Slide 12 : forwording solution :

بنهاية ال E رح يكون الناتج الجديد موجود

	1	2	3	4	5	6	7	8	9	10
sub X_2, X_1, X_3	F	D	E	M	W					
and X_{12}, X_2, X_5		F	D	E	M	W				
or X_{13}, X_6, X_2			F	D	E	M	W			
add X_{14}, X_2, X_2				F	D	E	M	W		
sd X_{15}, 100(X_2)					F	D	E	M	W	

*خلصناهم ب 9 cycles *

Homework 1

A multicore process has a clock rate of 3.0 GHz and voltage of 1.0 V. Assume that, on average, it consumes 75 w of static power and 30 watt of dynami Power when all its six cores are active. it executes a program consisting of 10^{10} instructions on a single core with the following instruction mix.

Instruction type	CPI	Frequency
Arithmetic	1.0	50%
Load / Store → أبطأ اشي	2.0	40%
Branch instructions	1.5	10%

(a) Find the total execution time for this Program on one core?

$\text{Cpu time} = IC \times \Sigma(CPI \times \text{frequency}) \div \text{clock rate} = 10^{10} \times (1.0 \times 0.5 + 2 \times 0.4 + 1.5 \times 0.1) \div 3 \times 10^{9} = 4.833$ Second

(b) when this program is parallized and run on the six cores, the number of Arithmetic and Load/Store instructions per core is devided by 4, but the number of branches remains the same? what is the execution time of the parallel program. $= IC \times \Sigma(CPI \times \text{frequency}) \div \text{clock rate}$

$10^{10} \times (1.0 \times 0.5 \div 4 + 2.0 \times 0.4 \div 4 + 1.5 \times 0.1) \div 3 \times 10^{9} = 1.583$ Sec

(c) What is the speedup of the parallel program? Speedup $= 4.833 \div 1.583 = 3.05$ ← ممكن يوصل أحسن من هيك يعني 6 مثلاً

(d) Assumes that the Power consumed is proportional to the number of active cores. what are the energies consumeb by the serial program on the single core and the parallel program on the six cores? Energy of the serial program = exec.time x Power = $4.833 \times ((75+30) \div 6) = 85$ Joules

Perfor... لو كان همي الـ بختاره ← Energy of the parallel program = exec.time x Power = $1.583 \times (75+30) = 166$ Joules

(e) what is the average capacitive load of this processor when the six cores are active?

$\text{Power} = \frac{1}{2} \times \text{capacitive load} \times \text{voltage}^2 \times \text{frequency}$

$30 = \frac{1}{2} \times \text{capacitive load} \times 1^2 \times 3 \times 10^{9} \rightarrow$ capacitive load = 20 nano frades

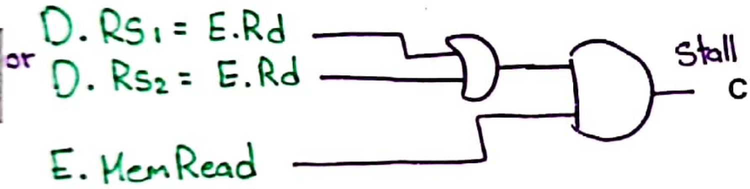
(f) what is the average current drawn by this processor when the six cores are active?

power = current x voltage $\Rightarrow 75+30 = \text{current} \times 1 \rightarrow \text{current} = 105$ Ampere. → كبير يعتبر ← بدخلوها عن طريق عشرات الـ pins in parallel

* لو همي higher performence رح استهلاك طاقة أكبر * (c)

Load-Use Hazard Detection

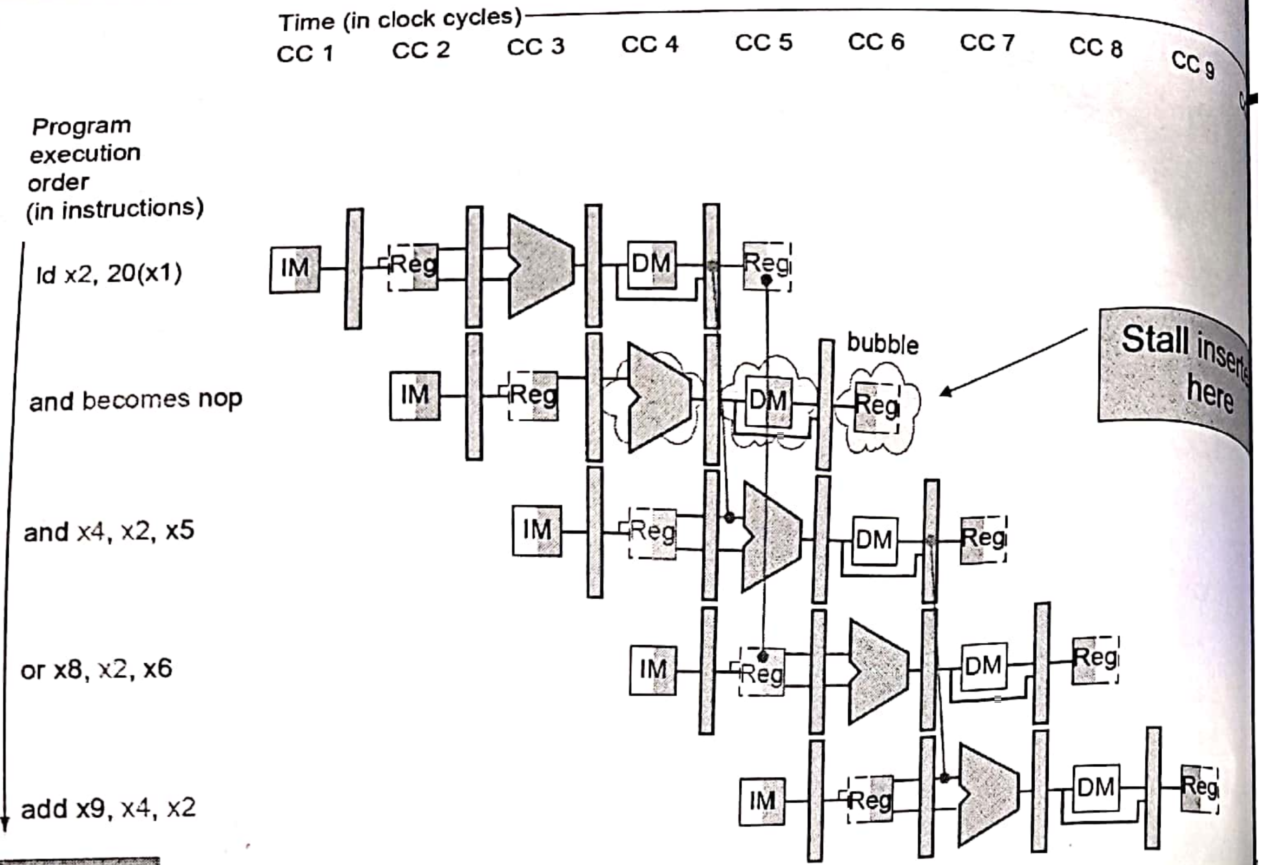
- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- Load-use hazard when Load-use شروط استخدام
  - ID/EX.MemRead and
    5 bits ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or 5 bits
    (ID/EX.RegisterRd = IF/ID.RegisterRs1))
- If detected, stall and insert bubble



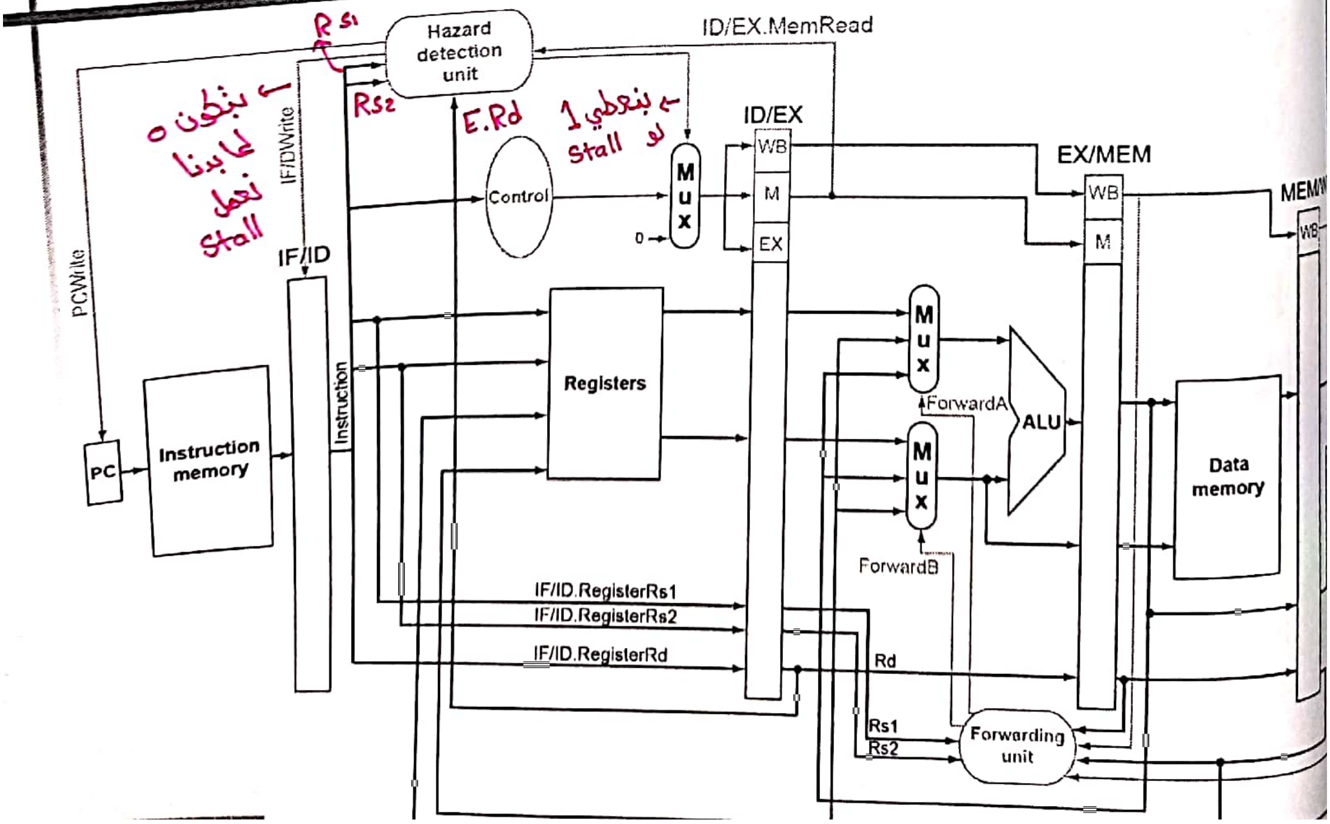
How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `ld`
    - Can subsequently forward to EX stage

Load-Use Data Hazard



Datapath with Hazard Detection



Stalls and Performance

The BIG Picture

- Stalls reduce performance
  - « بنحتاجه بحالات » But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

* شو بنحتاج نعمل بال Stall cycle : → slide 16

1- Signal تروح لل PC تعمل disable
2- IR → Rigister after PC disable
3- No-operation (0) to Execution

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for a = b + e; c = b + f;

↓ RISC-V

```
         ld    x1, 0(x0)      b
         ld    x2, 8(x0)      e
stall →  add   x3, x1, x2
         sd    x3, 24(x0)     a
Statement 1
         ld    x4, 16(x0)     f
stall →  add   x5, x1, x4
         sd    x5, 32(x0)     c
Statment 2
```

13 cycles

4+7+2=13

```
ld    x1, 0(x0)
ld    x2, 8(x0)
ld    x4, 16(x0)
add   x3, x1, x2
sd    x3, 24(x0)
add   x5, x1, x4
sd    x5, 32(x0)
```

11 cycles

4+7=11

Scanned with CamScanner

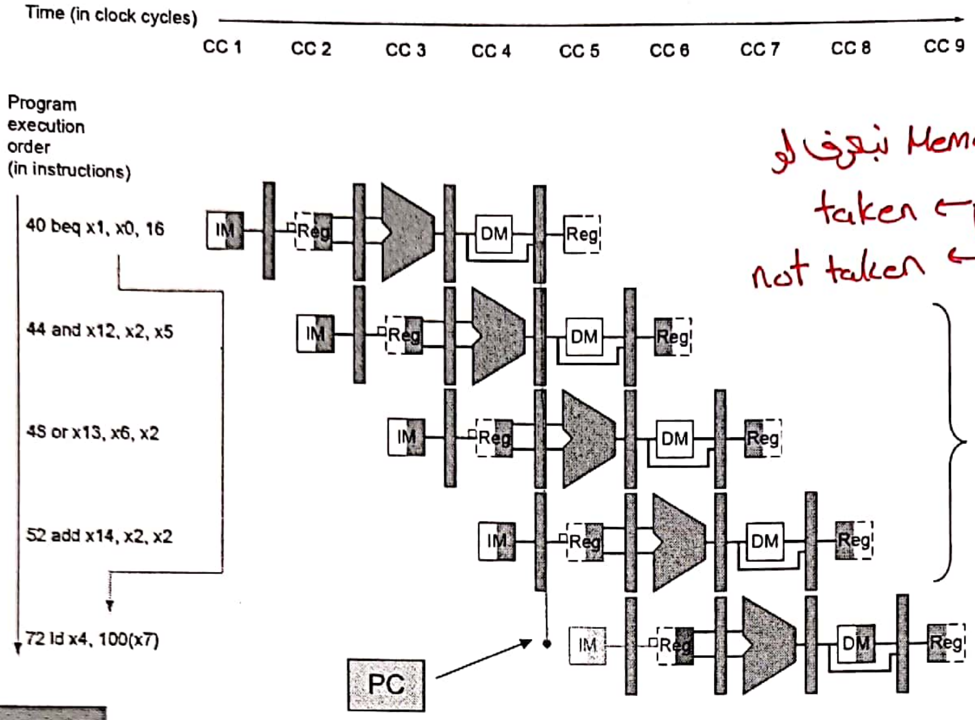
Contents

Contents

بعملوا delay
و stalls فكيف
نقللهم
عن طريق

Branch Hazards

- If branch outcome determined in MEM



* بال Memory بنعرف لو
ال Branch taken ←
not taken ←

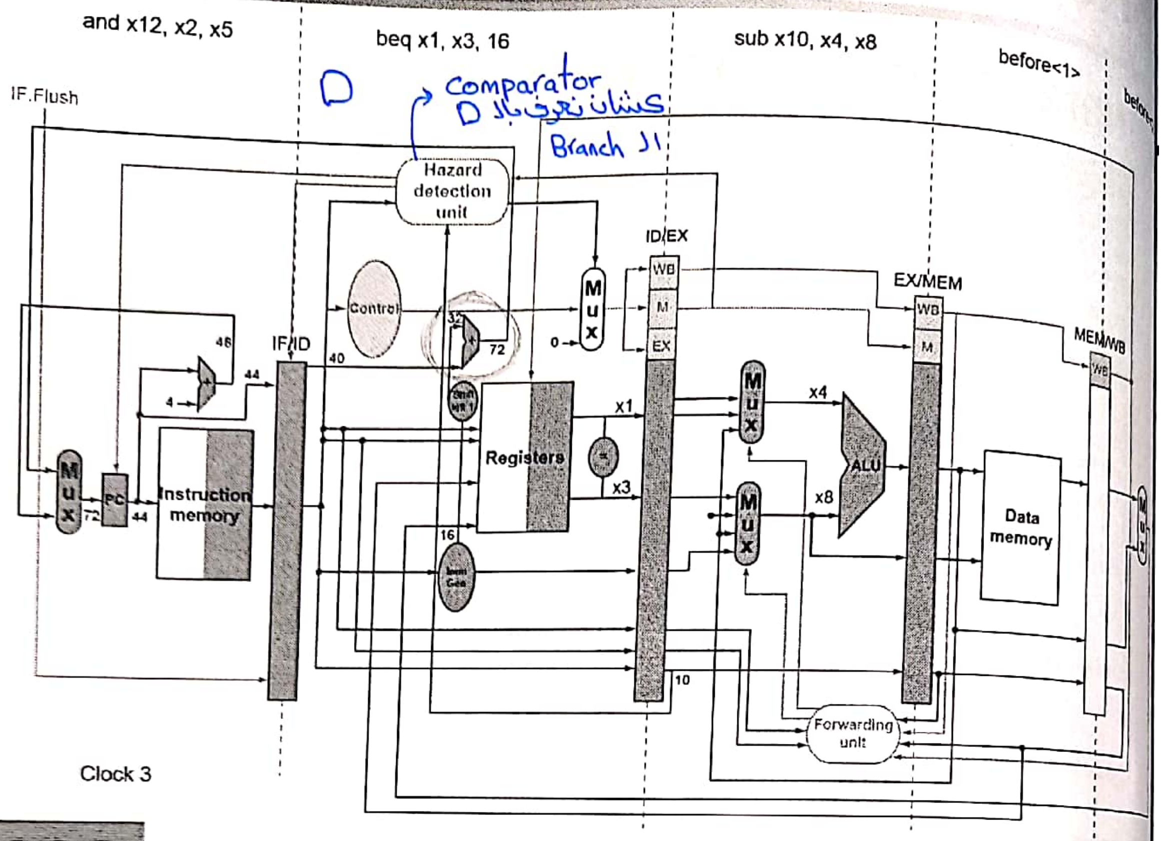
Flush these instructions (Set control values to 0)

Reducing Branch Delay

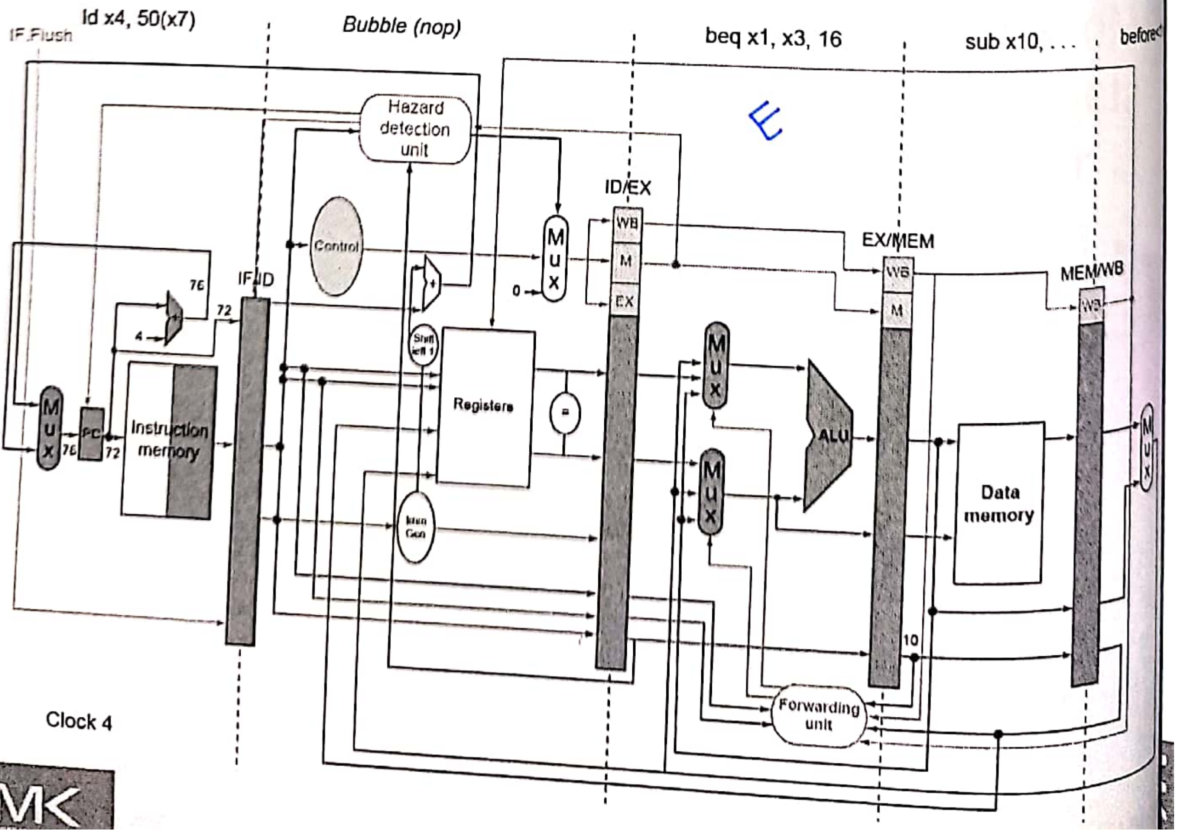
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub  x10, x4, x8
40:  beq  x1,  x3, 16  // PC-relative branch
                       // to 40+16*2=72
44:  and  x12, x2, x5
48:  orr  x13, x2, x6
52:  add  x14, x4, x2
56:  sub  x15, x6, x7
     ...
72:  ld   x4, 50(x7)
```

Example: Branch Taken



Example: Branch Taken



Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In RISC-V pipeline
  - Can predict branches not taken → أسهل الطرق افترض branches not taken
  - Fetch instruction after branch, with no delay

لإنه طريقة Predict not taken لو كان ال Pipline طويل فحيطلع ال Branch Delay كبير

More-Realistic Branch Prediction

1) Static branch prediction → Static بتتصرف يعني ال Branch prediction unit مش بناء على الحالة الأصلية اللي مفروض تتعلم منها.
   - Based on typical branch behavior
   - Example: loop and if-statement branches
     - Predict backward branches taken
     - Predict forward branches not taken

بتحسنة لتقريباً 90% [illegible]

2) Dynamic branch prediction
   - Hardware measures actual branch behavior → عكس ال Static، بتتعلم من ال actual اللي صار
     - e.g., record recent history of each branch
   - Assume future behavior will continue the trend
     - When wrong, stall while re-fetching, and update history

بنستفيد من الماضي عشان تتنبأ في الحاضر

Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

أبسط
طريقة بال dynamic Branch prediction ←

Branch History Table (BHT)

NT : 0
T : 1

← Branch prediction bits هدفه يعطيني

One-Level Branch Predictor

Memory

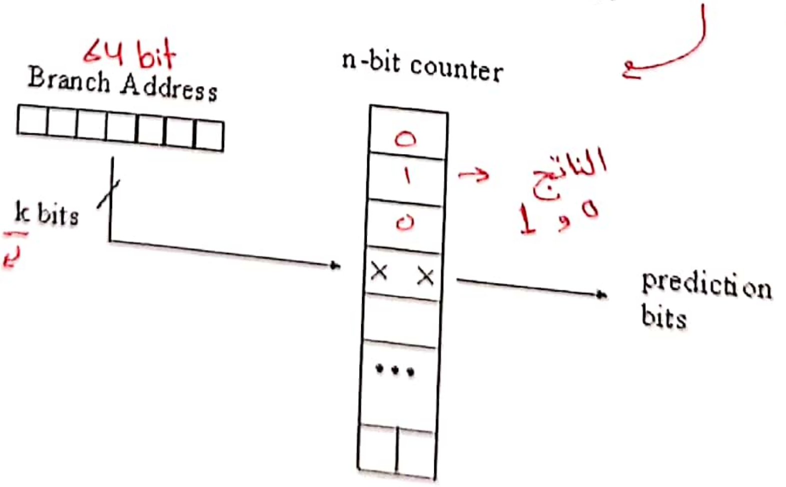
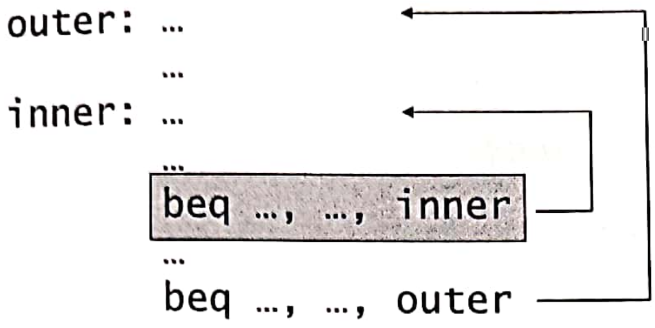


Table size = n × 2^k bits

$2 \times 2^{10} = 2k$ bits ← كمثال.

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

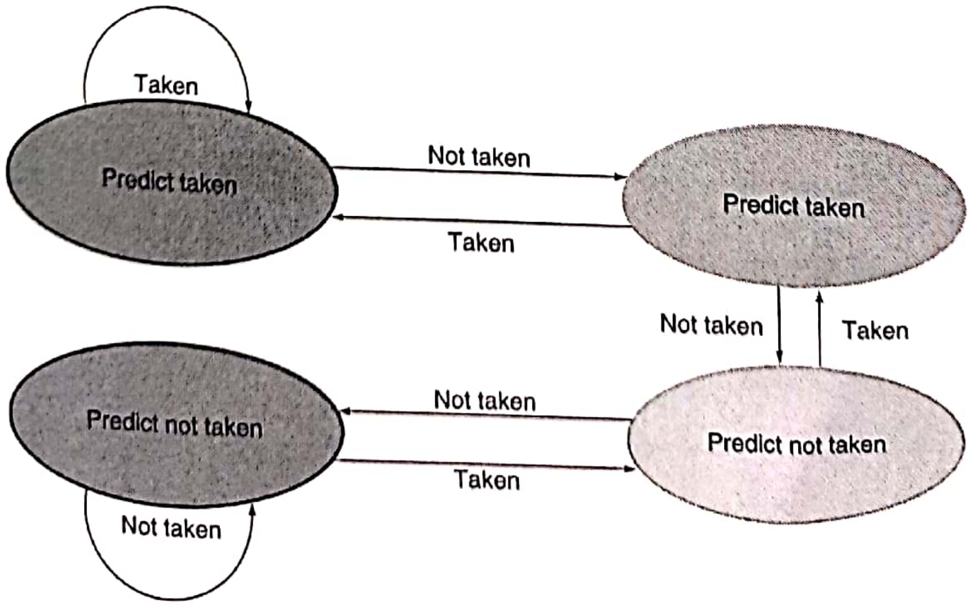


```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

  - Mispredict as taken on last iteration of inner loop
  - Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

- Only change prediction on two successive mispredictions



Calculating the Branch Target

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- Branch target buffer
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately

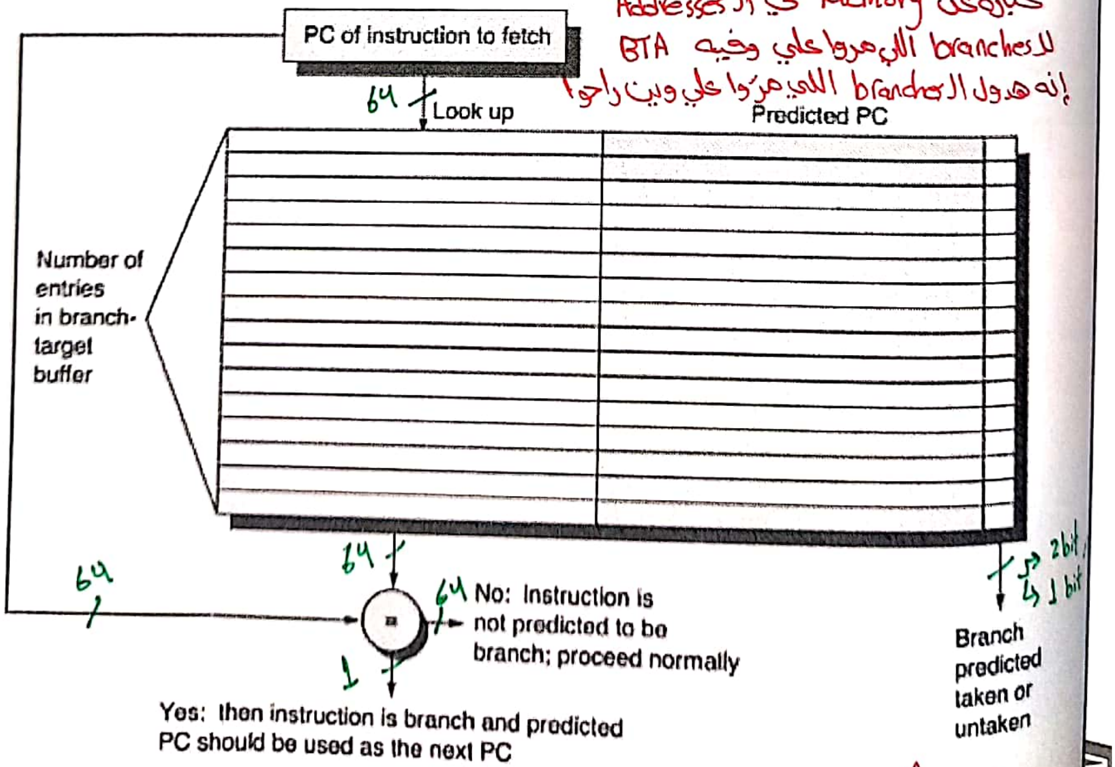
كشان بالمستقبل لو مرت علي نفس ال branch وكان ال address تاعوا نفس الي مخزنه عندي فبوديني ال BTA عن طريق ال MUX

كلما يمر علي Branch بخزن ال address تبعها ولوين بتروح

من ال PC في Address هاد ال Address بستخدمه لنوجد BTB

Branch Target Buffer (BTB)

عبارة عن Memory في ال Addresses للbranches اللي مروا علي وفيه BTA إنه هدول ال branches اللي مرّوا علي وين راحوا


PC of instruction to fetch
64
Look up
Predicted PC
Number of entries in branch-target buffer
64
64
64
=
1
No: Instruction is not predicted to be branch; proceed normally
2 bit
1 bit
Branch predicted taken or untaken
Yes: then instruction is branch and predicted PC should be used as the next PC

ممكن يخزنوا معاه ال BHT بس بعلي الأوقات ما بكون موصولين سوا غالبا مفصولين

Contents

Contents

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently
- ١) Exception
  - Arises within the CPU → يعني داخل ال Pipline وهو بتنفذ ممكن يصير error بحتاج معالجة
    - e.g., undefined opcode, syscall, … و division by Zero
- Interrupt
  - From an external I/O controller → بيجونا من الخارج
- Dealing with them without sacrificing performance is hard

Handling Exceptions

- Save PC of offending (or interrupted) instructio → ال instruction اللي عملت Excep.
  - In RISC-V: Supervisor Exception Program Counter (SEPC)
- Save indication of the problem
  - In RISC-V: Supervisor Exception Cause Register (SCAUSE)
  - 64 bits, but most bits unused ← مش كلهم مستخدمات
    - Exception code field: 2 for undefined opcode, 12 for hardw malfunction, …
- Jump to handler
  - Assume at 0000 0000 1C09 0000_{hex}

An Alternate Mechanism

- Vectored Interrupts
  - Handler address determined by the cause
- Exception vector address to be added to a vector table base register:
  - Undefined opcode 00 0100 0000_{two}
  - Hardware malfunction: 01 1000 0000_{two}
  - ...: …
- Instructions either
  - Deal with the interrupt, or
  - Jump to real handler

Handler Actions

- Read cause, and transfer to relevant handler
- Determine action required
- If restartable
  - Take corrective action
  - use SEPC to return to program
- Otherwise
  - Terminate program
  - Report error using SEPC, SCAUSE, …

Slide 14 : Load-use data hazard

	1	2	3	4	5	6	7	8	9	10
ld X1 , 0(X2)	F	D	E	M	W					
sub X4, X1 , X5		F	D	D	E	M	W			

→ $\overset{\text{ثابت}}{4} + \overset{\text{Instruction}}{2} + \overset{\text{Stalls}}{1}$

= 7 cycles

* ال sub وال add بتجهز ال data بال Execute stage

* ال load بتجهز ال data بال Memory بنهايتها

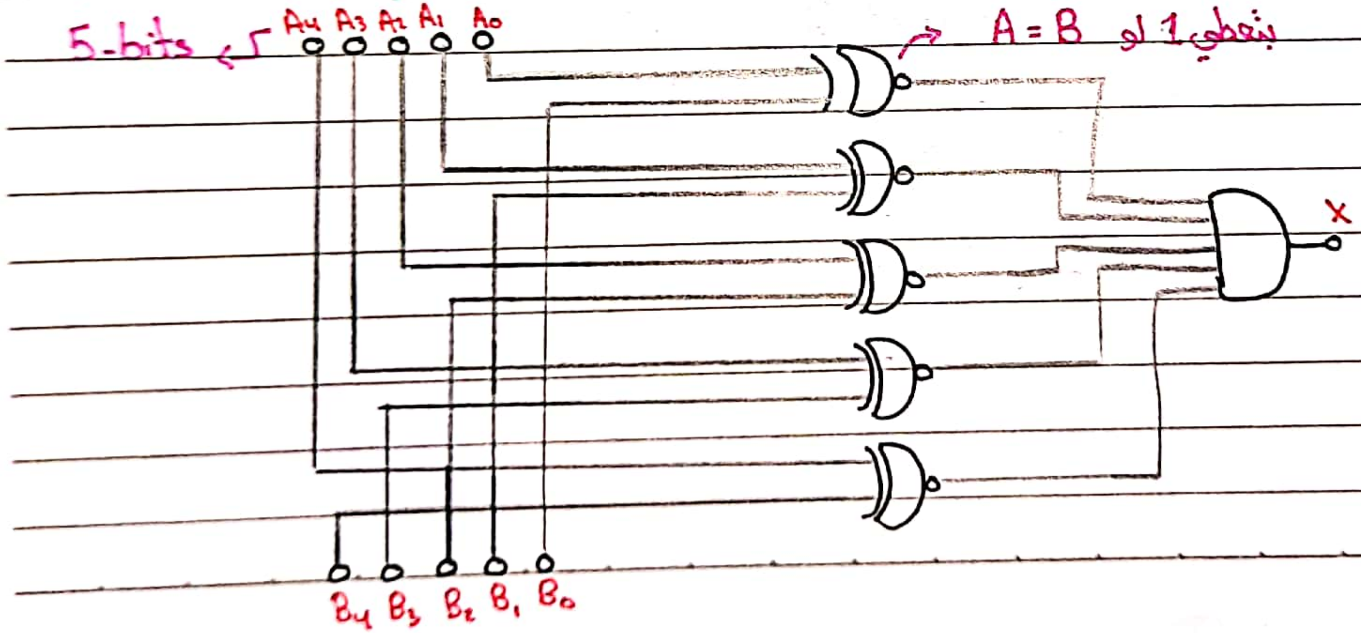
لـ عملنا 1 stall عشان ال data بال load بتجهز نهاية ال Memory ، بعدين عملنا Forwording عشان نحط القيم الجديدة .

* لحل ال Load-use بنحتاج نعمل 1 stall و Forwording .

* الدائرة اللي بنحتاجها لتنفيذ : D.RS1 = E.Rd أو D.RS2 = E.Rd :

5 bit ↑ ↑ 5 bit

بنعطي 1 لو A = B



Slide 20: Code Scheduling to Avoid Stalls:

EX: ld X1, 0(X2)
add X3, X1, X4
ld X5, 0(X6)
↳ independent

$\xrightarrow[\text{ترتيب}]{\text{اعادة}}$

ld X1, 0(X2) F D E M W
ld X5, 0(X6) F D E M W
add X3, X1, X4 F D E M W

↓
هون بهيك ترتيب مابحتاج أعمل Stall بس Forwarding

Slide 23: Branch Hazards

Solving branches in the Memory stage

* بنحولهم لـ null instruction لأنو مابدنا ينفذوا

* بال Memory بنعرف إذا ال Branch taken أو not taken بنعاملها

* افترضنا أنه ال Branch is taken

		1	2	3	4	5	6	7	8	9	10
40	beq X1, X0, 16	F	D	E	M	W					
44	and X12, X2, X5		F	D	E	n	n				
48	or X13, X6, X2			F	D	n	n	n			
52	add X14, X2, X2				F	n	n	n	n		
72	ld X4, 100(X7)	–				F	D	E	M	W	

↳ target address = 40 + 2 × 16 = 72
ل لأنه half bytes لازم نضرب في 2

↳ لأنو بنهاية ال cycle4 وبداية cycle5 عرف انه ال Branch is taken فبعمل F

* هون ال Performance بيقل لأنه كان عندي 3 stall cycles

* Branch delay = 3

* و لتقليل ال Branch delay ← هنالك حل بأنو بدل ما ال Branch يجوز بال Memory stage بخلوا يجوز بال Decode stage

Slide 24 :

Solving branches in the Decode stage → صرنا هون نعرف نتيجة الـ Branch

		1	2	3	4	5	6	7	8	9	10
40	beq $x_1, x_0, 16$	F	D	E	M	W					
44	and x_{12}, x_2, x_5		F	n	n	n	n				
48	or x_{13}, x_6, x_2										
52	add x_{14}, x_2, x_2										
72	ld x_4, 100(x_7)			F	D	E	M	W			

*افترضنا إنه Branch is taken

Branch delay = 1 *

* التحسين الجديد هو أن أقوم بعمل ← Branch prediction ↓
بخلي الـ Hardware يتنبأ بالنتيجة.

Slide 28 : Predict Not taken

Solving branches in the Decode stage

① Assume branch is not taken

	1	2	3	4	5	6	7	8	9	10
beq x_1, x_0, L	F	D	E	M	W					
I_2		F	D	E	M	W				
L I_T										

بهيك توقعنا →
وكان توقعنا صح
وما ضيعنا أي
cycle
→ Branch Delay
= 0

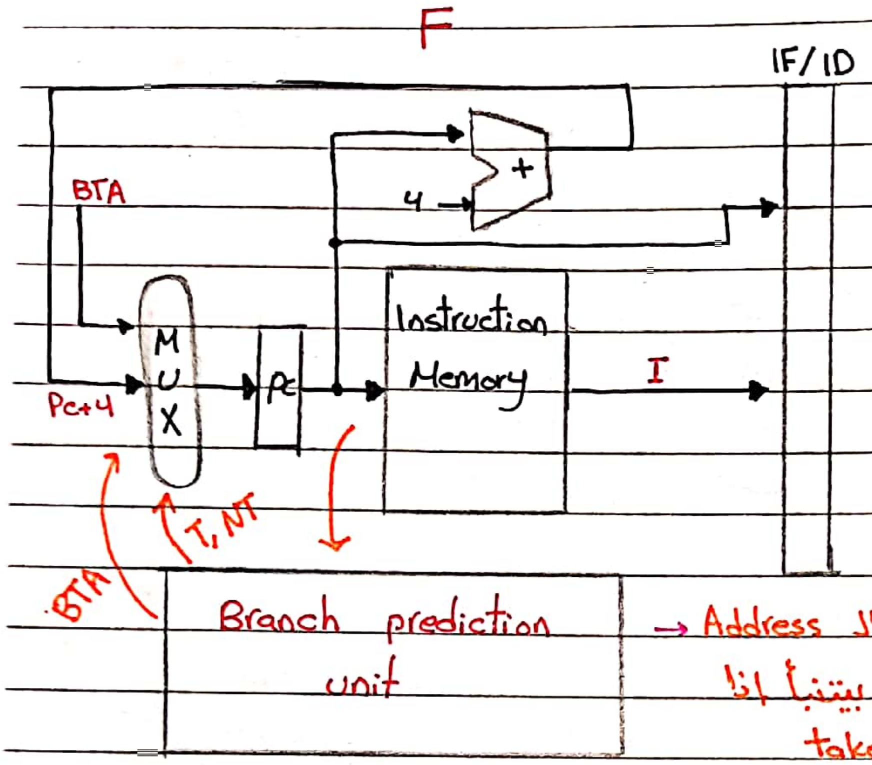
② Assume branch is taken

avg Branch Delay = $\frac{1}{2}$ cycle ←

	1	2	3	4	5	6	7	8	9	10
beq x_1, x_0, L	F	D	E	M	W					
I_2		F	n	n	n	n				
L I_T			F	D	E	M	W			

→ Branch
Delay = 1

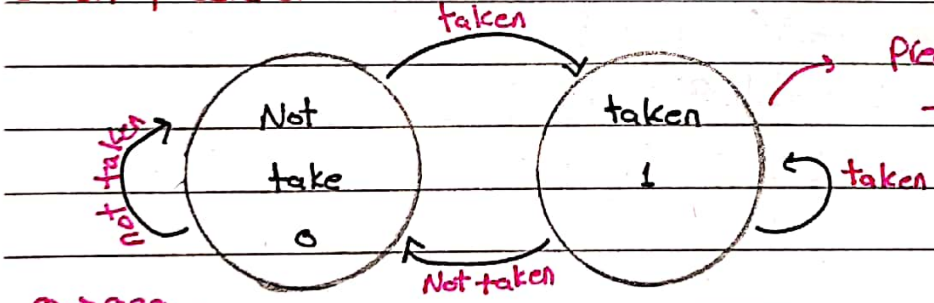
Slide 28 : Branch predection Unit



→ Address إضافته للتصميم وشغلته ← من الـ
لـ instruction اللي قاعدين بنجيبها بيتنبأ إذا
الـ Branch ← taken or not taken
وأيضا مسؤولة عن إعطائي الـ BTA

Slide 30 : BHT → n=1 / n=2

1-bit predictor ↲ ↓ n=1



لو افترضنا الـ Prediction taken
وطلعت فعلاً taken بنخليها taken
وزي هيك عالحالات الباقية

0→999

Assume 1000 iterations for the inner loop and for the outer loop

```
outer :...  → 1000 مرة
   ...      فحتتنفذ 1000000 مرة
inner : ...  1000 = مرة
             x1000
   ...
   beq,..,..,inner
   ----
   beq,..,..,outer
```

لأنه المفروض يطلع ما ينفذها

Iteration	997	998	999	0	1
Prediction Result	T/C	T/C	T/I	NT/I	T/C

→ أخطأ مرتين

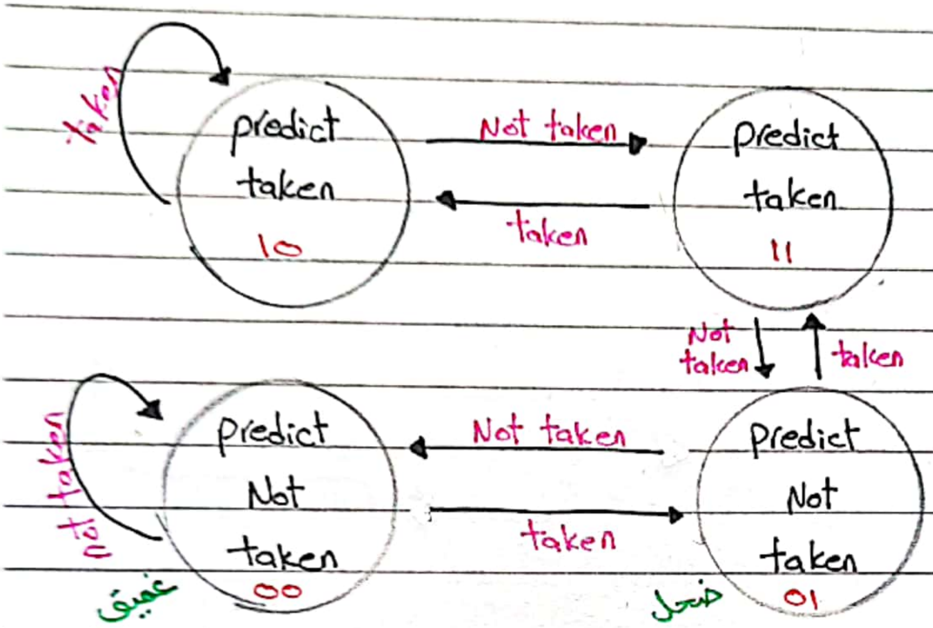
* T: taken / NT: Not taken

C: correct prediction / I: incorrect prediction

* ال one bit بغير رأيه بسرعة فاتجهنا لل 2 bit بغير رأيه بطريقة أبطأ.

Slide 32 :

2-bit predictor



T 1 → most sig bit

NT 0 → most sig bit

```
outer: .....
       ....
inner: ...
       ....
    beq,..,..,inner
       .....
beq .., ..., outer
```

Iteration	997	998	999	0	1
Prediction result	T/C	T/C	T/I	T/C	T/C

1 miss Prediction

DATE

Slide 38 : Handling Exceptions

*ميزة هاي الطريقة إنها Simple hardware
بس الـ Software أصعب

11 *لنفرض صار Exception في 13
12
13 * → المفروض مانكمل 14 و 15
14 بل نروح عال Address اللي
15 موجود عنده الـ Exception handling
وبالـ RISC-V هاد الـ Address المخصص

0000 0000 1C09 0000 hex

Exception Handling Routine

برنامج جزء من الـ operating system بقدر يتعامل مع الـ Exceptions

SEPC, SCAUSE → داخل الـ Prossesor لازم نضيف Additional Register اللي هم SEPC , SCAUSE

هدول بس يصير في Exception لازم نخزن الـ Address الـ Instruction اللي صار عليها الـ Exception
وبنفس الوقت بدي أخزن سبب ونوع الـ Exception

* Exception Handling Routine : رح يستفيد من سبب الـ Exception اللي خزناه بالـ SCAUSE ويعالجه للـ Exception وبعين من الـ SEPC بيقدر يرجع للـ Instruction اللي توقفنا عندها ويتابع تنفيذ البرنامج الأصلي

* الـ Exception Handling Routine مش دايما بقدر يرجع للـ Instruction اللي توقفنا عندها ، إذا قدر يحل المشكلة وبنقدر نكمل البرنامج بيرجع ، وبعض المرات بروح ع مكان تاني بالـ operating system لو كان ما بقدر يحل المشكلة مثل مشكلة : illgal code وساعتها الـ OS بطلع للـ user رسالة بتقله إنه عندك مشكلة ما بنقدر نكمل برنامجك

Slide 40 :

11 undefined opcode 00 0100 0000 two
12 Hardware malfunction: 01 1000 0000 two
13 * هاي الطريقة حسب نوع الـ Exception بنروح لـ Address مختلف.
14 * ميزة هاي الطريقة بتخليه أسرع .
15 * الـ Software أسهل .

SEPC, SCAUSE

Exceptions in a Pipeline

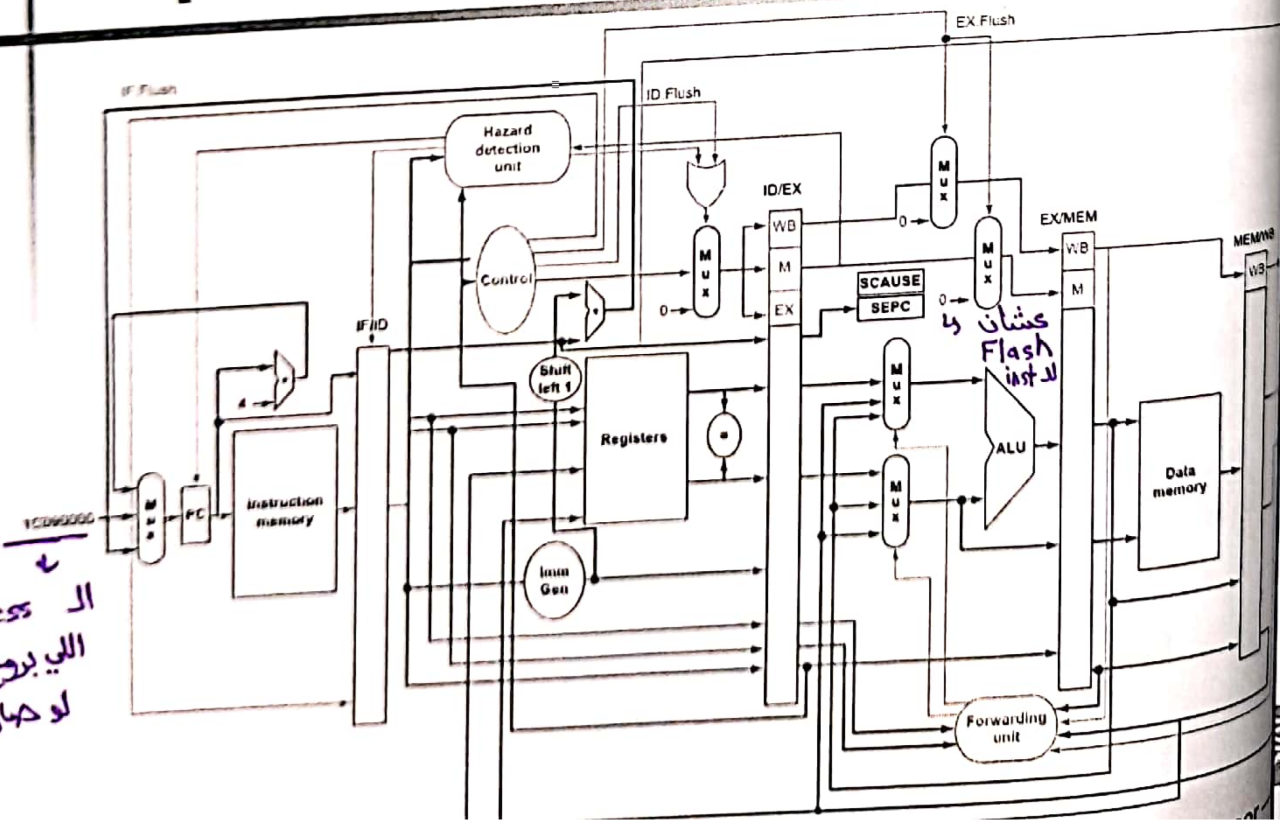
التعامل صعب معهم

- Another form of control hazard
- Consider malfunction on add in EX stage

```
add x1, x2, x1
```

  - Prevent x1 from being clobbered
  - Complete previous instructions
  - Flush add and subsequent instructions
  - Set SEPC and SCAUSE register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

Pipeline with Exceptions



الـ address اللي بروح عليه لو صارلها exception

Exception Properties

- Restartable exceptions →
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch
- PC saved in SEPC register
  - Identifies causing instruction

يعني بنقدر نرجع للinstruction
اللي صار عليها
Exception
بعدما نحله
↓
عكسهم ال
unrestartable

↓
ال instruction
اللي صار عندها exception

Exception Example

- Exception on add in

```
40    sub  x11, x2, x4
44    and  x12, x2, x5
48    orr  x13, x2, x6
4c    add  x1,  x2, x1
50    sub  x15, x6, x7
54    ld   x16, 100(x7)
…
```

هدول لازم
→ ينفذوا ويخلصوا

← "هون صار فيه
exception"

→ هدول بنعملهم
Flash

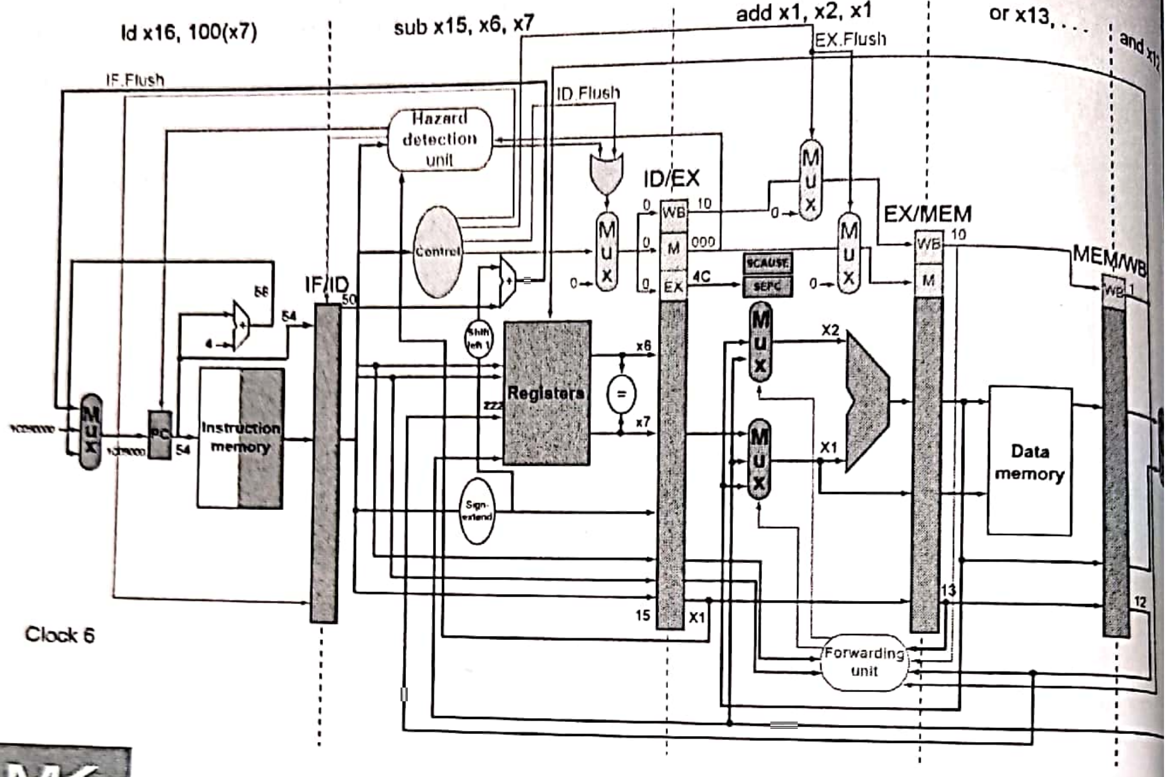
- Handler

```
1C090000   sd  x26, 1000(x10)
1c090004   sd  x27, 1008(x10)
…
```

بالنهاية رح يعمل
ld x27
ld x26

بالعادة ال Exception handling subroutine بيلشوا ب sd : لإنه حتى يشتغل بده reg
يقرأ ويكتب منهم وبنفس الوقت مابده يخرب ال data اللي بالبرنامج الأصلي وما يضيعها
فبعملهم save

Exception Example



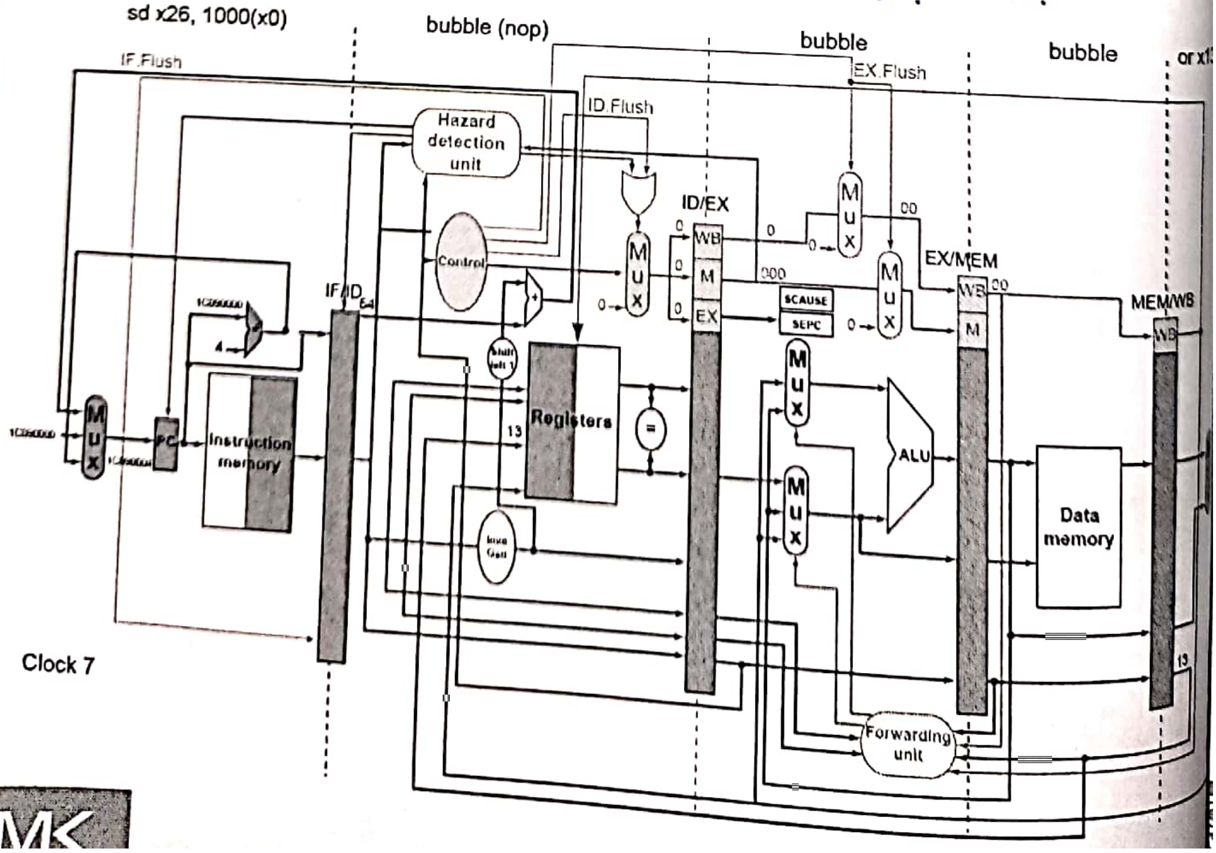
Chapter 4 — The Processo

* الدكتور بعمل Flash عند W مش عند الـ M لإنه هيك أرق وأصح ليعطي مجال نخلص

الـinstruction اللي قبل

مش زي السلايد
بال M بعملوا

Exception Example



Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions
  - "Precise" exceptions → زمان ماكانوا ابقدروا يعملوا هيك
- In complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

كانوا يعملوا هيك

Imprecise Exceptions → Processors كانت لما يصير exception ما نعرف مين خلص ومين ما خلص وهو عملية معقدة و بعض المرات لو صار Exception ما بنقدر نرجع للبرنامج الأصلي لأنه ما بنعرف لو كل ال instruction اللي قبل ال exception خلصوا ولا لا و لا اللي بعدها

- Just stop pipeline and save state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require "manual" completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines

Contents

4.6 Pipelined Datapath and Control (Review)

4.7 Data Hazards: Forwarding versus Stalling

4.8 Control Hazards

4.9 Exceptions

4.10 Parallelism via Instructions

← بنفذ Multiple instructions every cycle وهذا موجود بال modern Processors

4.11 Real Stuff: The ARM Cortex-A53 and Int
Core i7 Pipelines

4.14 Fallacies and Pitfalls

4.15 Concluding Remarks

Contents

4.10 Parallelism via Instructions

- Instruction-Level Parallelism (ILP)
- Multiple Issue
- Static Multiple Issue
- VLIW
- Scheduling Static Multiple Issue
- Loop Unrolling
- Dynamic Multiple Issue
- Register Renaming
- Speculation
- Why Do Dynamic Scheduling

أحدث طريقة : الجاهز بتنفذ

Instruction-Level Parallelism (ILP)

§4.10 Parallelism via Instructions

- Pipelining: executing multiple instructions in parallel → Time = IC × CPI × Tinst

الي أخذناه قبل ال Multiple instruction
بشغلوا ب cycle وحدة بس
بمراحل مختلفة

- To increase ILP

طريقتين

1. Deeper pipeline T↓
   - Less work per stage ⇒ shorter clock cycle
2. Multiple issue
   - Replicate pipeline stages ⇒ multiple pipelines
   - Start multiple instructions per clock cycle
   - CPI < 1, so use Instructions Per Cycle (IPC)
   - E.g., 4GHz 4-way multiple-issue → instruction per cycle
     - 16 BIPS, peak CPI = 0.25, peak IPC = 4
   - But dependencies reduce this in practice

مستخدم هاي في المحاضرات السابقة

→ مثلا يكون

F D E
F D E
F D
F D
⋮

F تحت بعض عادي

f × IPC

لما بنزل ال CPI فال Time بقل

بقدر ل 4 inst:

F D E
F D E
F D E
F D E

$* \ CPI = \frac{1}{IPC} = \frac{1}{4} = 0.25$

Multiple Issue

1. Static multiple issue → العبئ بيجي على ال compiler أو المبرمج

   أبسط

   والمبرمج بحط ال instruction اللي ما بعتمدوا عبعض، اللي ممكن تنفيذهم مع بعض بمجموعات

   - Compiler groups instructions to be issued together

   وال Hardware بياخذ هاد ال issue slots وما بيحتاج

   - Packages them into "issue slots"
   - Compiler detects and avoids hazards

   بفحصه لإنه ال compiler detects and avoids hazards

2. Dynamic multiple issue

   أقوى و higher performence

   - CPU examines instruction stream and chooses instructions to issue each cycle
   - Compiler can help by reordering instructions
   - CPU resolves hazards using advanced techniques at runtime

ال CPU نفسها بتجيب ال instruction وبتعملها Decode وبعرف ايش ال dependences
بنظم وال Hardware بنفسه chooses instructions to issue each cycle

Static Multiple Issue

- Compiler groups instructions into "issue packets" → issue Packets ب independent instructions بجمع
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations
  - ⇒ Very Long Instruction Word (VLIW)

مثال Itanium ← له بشبه ال Static Multiple issue

MK

Chapter 4 — The Processor

Ex : add x1, x2, x3
sub x4, x1, x5
add x6, x7, x8

dependent ← مابنقدر ننفذهم مع بعض

independent بنقدر أنفذهم مع بعض وأحطهم ب issue Packet

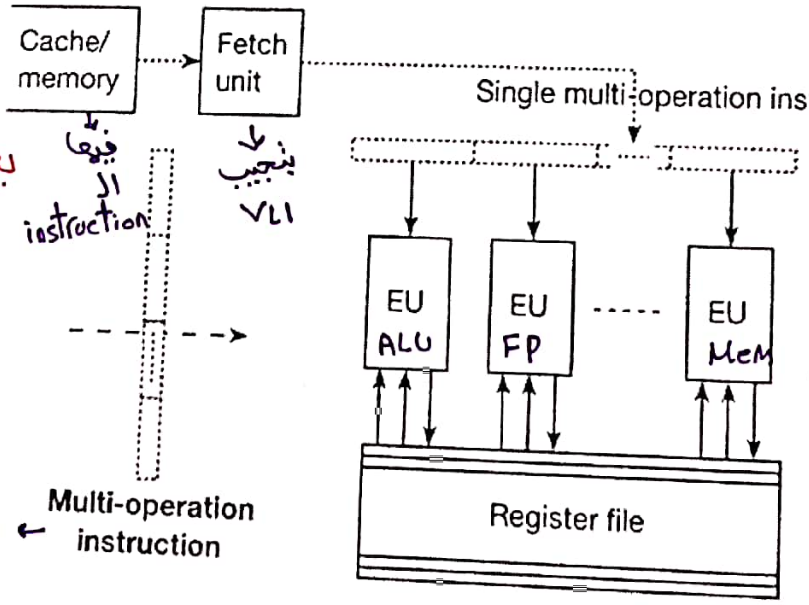
→ add x1, x2, x3
add x6, x7, x8
→ issue Packets

VILW

ممكن نعتبره نوع من ال Static Multiple issue ← VLIW ال instruction الوحدة أكبر بكثير من 32 bit

(very long instruction word, 1024 bits!)

وهاي ال Instruction Specify multiple concarent operations يعني بنقول لل Processor اعمل أكثر من one operation بنفس الوقت (concarent)



VLIW approach

يعني instruction فيها Multiple opertion بكون 128 or 256 bit

ال Compiler حد ايش ال operations الي ممكن ننفذهم مع بعض وحطهم ب VLI وكل operation تشغل على Execution unit خاصة

MK

ممكن نلاقيه لحد الآن

Scheduling Static Multiple Issue

ما بيحمل كثير و بوفر كهربا و بيعطي good performence مش Excellent performence

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with in a packet
  - Possibly some dependencies between packets → instruction يكون فيه Packet ممكن بين ال Packet وال بتعتمد على بعضها
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary → إذا ما قدر يلاقي independent instruction كافية يحطهم بـ Packet واحد بعبي باقي ال packet بـ no operations

يعني يعبي المحلات الفاضية

RISC-V with Static Dual Issue

يعني ال issue packet بكون فيه 2 instructions

- Two-issue packets
  - One ALU/branch instruction وحدة بتكون ALU/branch ووحدة load/store
  - One load/store instruction ولو ما قدر يلاقي بضيف (nop)
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

مش 4 زي أول PC+8

مفترض انه ما في stalls ولا في hazards

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

issue Packet

issue Packet

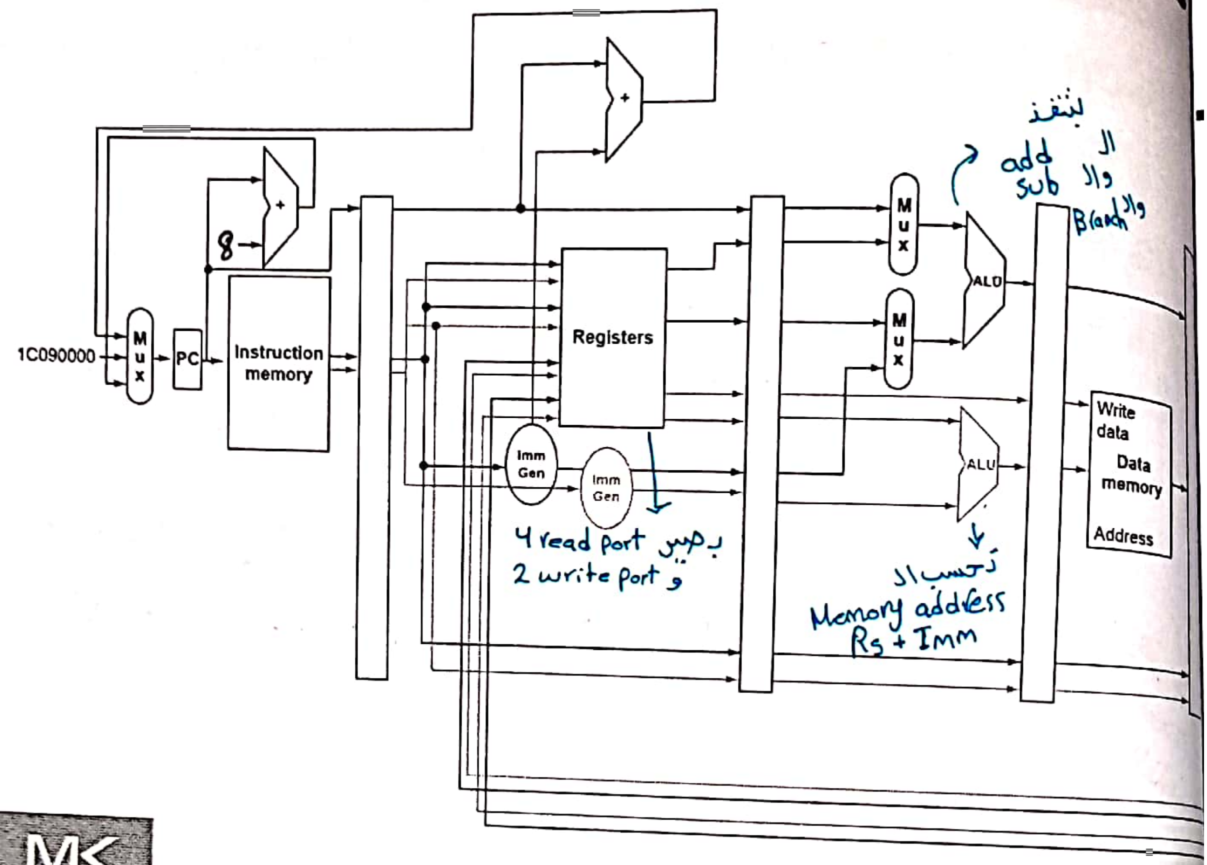
issue Packet

بصير ال throupot أفضل لإنه بنفذ 2 في cycle

2 IPC → instruction per cycle

$\frac{1}{2}$ CPI → clock per Instruction

RISC-V with Static Dual Issue



Hazards in the Dual-Issue RISC-

- More instructions executing in parallel
- EX data hazard → بين ال issue وال issue اللي بعديه ممكن يكون في dependent ونحتاج Forwarding
  - Forwarding avoided stalls with single-issue
  - Now can't use ALU result in load/store in same packe

```
add  x10, x0, x1
ld   x2, 0(x10)
```

← ما بنقدر نحطهم مع بعض لأنه يعتمدوا على بعض

    - Split into two packets, effectively a stall
- Load-use hazard
  - Still one cycle use latency, but now two instructions
- More aggressive scheduling required

Forwarding in Dual-Issue RISC-V

- In addition to forwarding from M and W to E, there are additional forwarding paths among the two pipelines, e.g.:
  - From W in memory pipeline to E in ALU pipeline

```
ld    x31, 0(x20)
add   x31, x31, x21
```

أخذناه قبل

  - From M in ALU pipeline to M in memory pipeline

```
add   x31, x31, x21
sd    x31, 0(x20)
```

Scheduling Example

* حصلنا على IPC = 1.25 عشان ال nop
فلو حطينا بدل ال nop ← instructions
بنزيد ال IPC فلذلك بنستخدم حل أحسن والي هو loop unrolling

- Schedule this for dual-issue RISC-V

```
Loop: ld    x31,0(x20)      // x31=array element
      add   x31,x31,x21     // add scalar in x21
      sd    x31,0(x20)      // store result
      addi  x20,x20,-8      // decrement pointer
      blt   x22,x20,Loop    // branch if x22 < x20
```

ممكن نحط ب 1 برضه

حطيناها أول وحدة لأنها أول عمليه

2 issue Packets

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

ما قدرنا نحط بالثانية لأنه load use hazard

لأنها بيجي بعد ال add
بس ممكن أحطها ب 3 برضه

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

دايماً بنحط آخر اشي

instruction ع issue Packet

ال addi قبله صار
فلازم أخليها 8 مو 0 عشان
نعكس أثر ال addi

Loop Unrolling

IPC فيها بنزيد عدد ال instructions الموجودة هنا عشان نزيد ال

- Replicate loop body to expose more parallelism
  - فوائد Reduces loop-control overhead
- Use different registers per replication
  - Called "register renaming" → بخلينا نتغلب على أنواع إضافية من ال dependence
  - Avoid loop-carried "anti-dependencies"
    - Store followed by a load of the same register
    - Aka "name dependence", write-after-read
    - Or "output dependence", write-after-write
      - Reuse of a register name

loop unrolling → حنحتاج لنعمل

مثال :

```
if ( - - —, )   i+=1
  a[i] = a[i]+3;
```

نعمله بطريقة أحسن وبكبر ال loop وبزيد instruction

```
if ( - - — i )   i+=2
{
  a[i] = a[i]+3;
  a[i+1] = a[i+1]+3;
}
```

Unrolling Steps

بعملها المبرمج و هاي الأيام بعملها ال compiler

1. Replicate the loop instructions n times ← أنت بتختاره
2. Remove unneeded loop overhead
3. Modify instructions
4. Rename registers
5. Schedule instructions

Loop Unrolling Example

	ALU/branch	Load/store	cycle
Loop:	addi x20,x20,-32	ld x28, 0(x20)	1
	nop	ld x29, 24(x20)	2
	add x28,x28,x21	ld x30, 16(x20)	3
	add x29,x29,x21	ld x31, 8(x20)	4
	add x30,x30,x21	sd x28, 32(x20)	5
	add x31,x31,x21	sd x29, 24(x20)	6
	nop	sd x30, 16(x20)	7
	blt x22,x20,Loop	sd x31, 8(x20)	8

32 general

32 Floating

بس عادي لإنه كنا كثير register

ما وصلنا 2 عشان لسا في nop

بس هاي طريقة أحسن

IPC = 14/8 = 1.75

لإنه قبل كنا نستخدم بس ..X31, X20, بس هلا صرنا نستخدم X28, X29, X30 بالإضافة للسابق

Closer to 2, but at cost of registers and code size

فلو بهمني الـ Performence ما بهمني لو حجم البرنامج كبر شوي

هو بحسن الـ Performence بس على حساب الـ registers والـ code size

يعني استغلكنا Memory usage ولكن هاي المشاكل مقبولة والـ Mem هاي الأيام رخيصة وكبيرة

كبر حجمه أكثر من 3 أضعاف

* الـ code الجديد : 8 × 2 × 4 = 64 bytes ← issue Packets

2 → instruction per issue packets

4 → 32 bit

* الـ code اللي قبل : 5 × 4 = 20 byte

5 → instructions

4 → 32 bit

Slide 41 : Exceptions in a pipeline :

بدي أفترض : إنه بس أعملها E في مشكلة Exception إذا جمعت int + int طلع كبير ما بقدر أخزنه بـ X_1

		1	2	3	4	5	6	7	8	9	10	11	12	13
	I1	F	D	E	M	W								
	add X_1 , X_2,X_1		F	D	E*	M	n							
x	I3			F	D	E	n							
x	I4				F	D	n							
x	I5					F	n							
IHS							F	D	E	M	W			

بتكون من Many Instructions لـ ما

ممكن نشتغل الـ Mem بس لما توصل الـ W

ما بصير نعملها و بنتحول لـ null و بنروح للـ IHS

وهيك ضمنا إنه الـ instruction اللي قبلها خلصوا لإنه ما كان عندهم مشكلة

عملنالهم Flash لإنه ما بصير يكملوا

بعد ما حلينا الـ exception باستخدام SEPC register بقدر إنه يرجع للـ I3 مش للـ add بهذا المثال خصيصًا لإنه عملية الـ add ما بتزبط لإنه الناتج كبير ما بنخزن.

* فإحنا ممكن نرجع لنفس الـ instruction أو للي بعدها وهذا بعتمد على الحالة.

Slide 47 : Multiple exceptions :

حفترض إنه illegal opcode ما بنقدر نعملها ← E

		1	2	3	4	5	6	7	8	9	10	11	12	13
	I1	F	D	E	M	W								
	add X1,X2,X1		F	D	E*	M*	n							
	I3(bad)			F	D*	E*	n			F	D*	E*	M*	n
	I4				F	D	n							
	I5					F	n			رجع لـ I3			[illegible] I3	
IHS							F	D	E	M	W			
								F	D	E	M	W		

← بالأول بنحل مشكلة الـ add قبل الـ I3 وبنرجع وبيجي دور I3 بعدين

Slide 58 : Hazards and forwording in Daul issue static processor :

Forwoding

Dependent load ↴	1	2	3	4	5	6	7	8	9	10
add X_{10}, X_0, X_1	F	D	E	M	W					
ld $X_2, 0(X_{10})$		F	D	E	M	W				

ما حطيناهم ←
بـ issue Packet
وحدة لإنه
بيعتمدوا على بعض
فلازم بما إنه
الـ ld بيعتمد عال add نكون الـ ld متأخرة في issue packet لاحقة

Slide 58 : Load use hazard :

	1	2	3	4	5	6	7	8	9	10
ld $X_1, 0(X_2)$	F	D	E	M	W					
sub X_4, X_1, X_5			F	D	E	M	W			

مابنقدر نحطهم ←
بنفس الـ issue Packet
فلازم نحط الـ sub
بـ issue packet
ثانية بس يكون
فيه بعد الـ ld كمان issue packet عشان يزبط الحل

Slide 59 : From W in Memory Pipeline to E in ALU Pipeline :

	1	2	3	4	5	6	7	8	9	10
ld $X_{31}, 0(X_{20})$	F	D	E	M	W					
sub X_{31}, X_{31}, X_{21}			F	D	E	M	W			

PAGE
DATE

Slide 59 : From M in ALU Pipeline to M in Memory pipeline :

	1	2	3	4	5	6	7	8	9	10
add X31 , X31 , X21	F	D	E	M	W					
sd X31 , 0 (X20)	F	D	E	M	W					

* أسهل من → load use hazard
لإنه الـ address
للـ sd بنحسب من
X20 وهو ما بعتمد

عالـinstruction الي فوقها والـ X31 بينحسب بنهاية الـ cycle 3 والـ sd بدها تاخد الـ X31 وتبعته
عالـ Memory وبتبعته عالـ M بالـ cycle4 بالـ M
فبقدر أحطهم بـ issue Packet 1 بشرط نعمل Forwarding .

Slide 62 : Example of loop unrolling :

```
Loop :
    ld    X31, 0 (X20)
    add   X31, X31, X21
    sd    X31, 0 (X20)
    addi  X20, X20, -8
    blt   X22, X20, Loop
```

① Replicate the loop instructions n times → 4

```
Loop :
    ld    X31, 0 (X20)
    add   X31, X31, X21
    sd    X31, 0 (X20)
    addi  X20, X20, -8
x   blt   X22, X20, Loop
```

```
    ld    X31, 0 (X20)
    add   X31, X31, X21
    sd    X31, 0 (X20)
    addi  X20, X20, -8
x   blt   X22, X20, loop
```

```
    ld    X31, 0(X20)
    add   X31, X31, X21
    sd    X31, 0 (X20)
    addi  X20, X20, -8
x   blt   X22, X20, Loop
```

```
    ld    X31, 0 (X20)
    add   X31, X31, X21
    sd    X31, 0 (X20)
    addi  X20, X20, -8
    blt   X22, X20, Loop
```

② Remove unneeded loop overhead

Loop :

ld X31, 0 (X20)
add X31, X31, X21
sd X31, 0 (X20)

ld X31, $\overset{-16}{\underline{0}}$(X20)
add X31, X31, X21
sd X31, $\overset{-16}{\underline{0}}$(X20)

ld X31, $\overset{-8}{\underline{0}}$ (X20)
add X31, X31, X21
sd X31, $\overset{-8}{\underline{0}}$(X20)

ld X31, $\overset{-24}{\underline{0}}$(X20)
add X31, X31, X21
sd X31, $\overset{-24}{\underline{0}}$ (X20)
addi X20, X20, $\overset{-32}{\underline{-8}}$
blt X22, X20, Loop

③ Modify instructions

Loop:

ld X31, 0 (X20)
add X31, X31, X21
sd X31, 0 (X20)

ld X31, -16 (X20)
add X31, X31, X21
sd X31, -16 (X20)

ld X31, -8 (X20)
add X31, X31, X21
sd X31, -8 (X20)

ld X31, -24 (X20)
add X31, X31, X21
sd X31, -24 (X20)
addi X20, X20, -32
blt X22, X20, Loop

④ Rename registers

Loop:

```
ld   X28, 0 (X20)          ld   X30, -16 (X20)
add  X28, X28, X21         add  X30, X30, X21
sd   X28, 0 (X20)          sd   X30, -16 (X20)

ld   X29, -8 (X20)         ld   X31, -24 (X20)
add  X29, X29, X21         add  X31, X31, X21
sd   X29, -8 (X20)         sd   X31, -24 (X20)
                           addi X20, X20, -32
                           blt  X22, X20, Loop
```

*ال Renaming رح يساعدني إني أداخل بين ال instructions وأنا بعمل Schedule حتى أقلل من المشاكل الي ممكن تطلعلي (anti-dependence, output dependence, WAW, WAR)

⑤ Schedule instructions

ممكن أحطها ? خطوة كمان

	ALU / branch	Load / store	cycle		
					ld X28, 0 (X20)
Loop :	addi X20, X20, -32	ld X28, 0 (X20)	1		add X28, X28, X21 sd X28, 0 (X20)
	nop	ld X29, 24 (X20)	2		ld X29, -8 (X20) add X29, X29, X21
	add X28, X28, X21	ld X30, 16 (X20)	3		sd X29, -8 (X20)
	add X29, X29, X21	ld X31, 8 (X20)	4		ld X30, -16 (X20) add X30, X30, X21
	add X30, X30, X21	sd X28, 32 (X20) بدل ٥ دم	5		sd X30, -16 (X20) ld X31, -24 (X20)
	add X31, X31, X21	sd X29, 24 (X20)	6		add X31, X31, X21
	nop	sd X30, 16 (X20)	7		sd X31, - 24 (X20) addi X20, X20, -32
	blt X22, X20, Loop	sd X31, 8 (X20)	8		blt X22, X20, Loop

* نخلص تعبية ال loads أول لإنه يفضل تنفيذهم أسرع ما يمكن لإنو ال instructions ال data الي بحتاجوها ال load بتجيبها

* بغير ال offset عشان يزبط مع ال addi :

-8 = X-32 / -16 = X-32 / -24 = X-32

Dynamic Multiple Issue

→ أحدث وأقوى
ويصرفوا كهرباء أكثر

- "Superscalar" processors →dynamic Multiple issue الاسم الثاني لل
- CPU decides whether to issue 0, 1, 2, … each cycle

  → مثلاً ال ALU
  ما بصير ابعتلها
  2 instruction مع بعض

  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling

  ↳ ما بنحتاجه
  ولكن لو عمل مثلاً
  Loop unrolling
  يحسن أكثر

  - Though it may still help
  - Code semantics ensured by the CPU

↓

* صحة ال Code مسؤولية ال CPU مش مسؤولية ال Compiler

Dynamic Pipeline Scheduling

أهم اشي بعمله

- Allow the CPU to execute instructions out of order to avoid stalls

إنه إذا فيه instructions قاعدين بستنوا بعض بطلهم يستنوا لكن لو في بعدهم الـ instructions آخرين بقدر آخدهم فبخدمهم

  - But commit result to registers in order

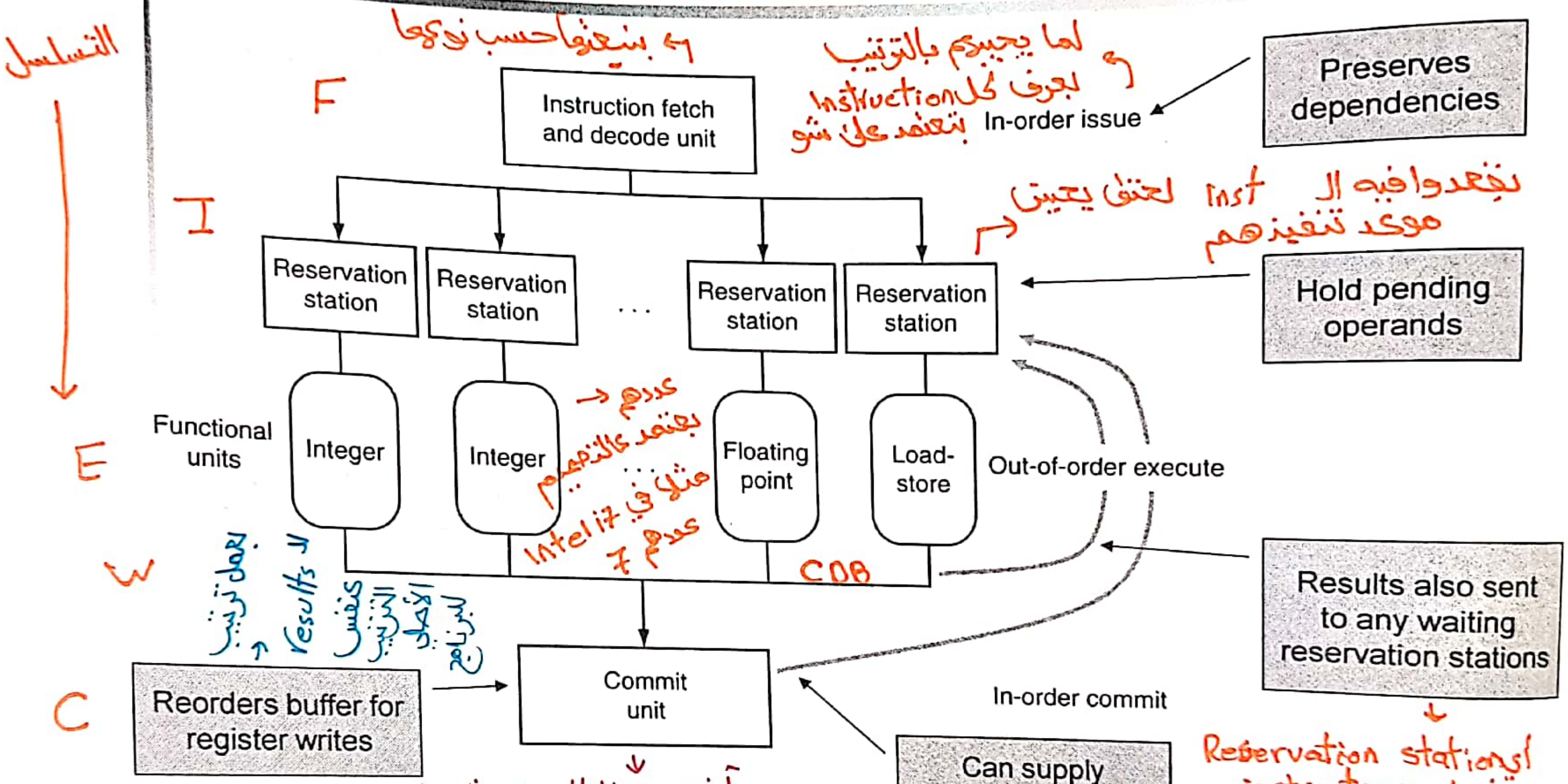
حتى نحافظ على ترتيب البرنامج ببعت النتيجة النهائية لل register على الترتيب الأصلي للبرنامج.

- Example

```
ld    x31,20(x21)
add   x1,x31,x2
sub   x23,x23,x3
andi  x5,x23,20
```

  - Can start sub while add is waiting for ld

Dynamically Scheduled CPU



التسلسل

F

I

E

W

C

← بنيعفوا حسب نوعها

Instruction fetch and decode unit

لما يجيبهم بالترتيب ← بعرف لكل Instruction بتعتمد على شو

In-order issue

Preserves dependencies

بيقعدوا فيه ال Inst لحتى يجيش موعد تنفيذهم

Reservation station

Reservation station

. . .

Reservation station

Reservation station

Hold pending operands

Functional units

Integer

Integer

. . .

Floating point

Load-store

عددهم ← بيعتمد عالتصميم مثلاً في Intel i7 عددهم 7

Out-of-order execute

CDB

Results also sent to any waiting reservation stations

بعمل ترتيب ← ال Results كنفس الترتيب الأصلي للبرنامج

Reorders buffer for register writes

Commit unit

In-order commit

Can supply operands for issued instructions

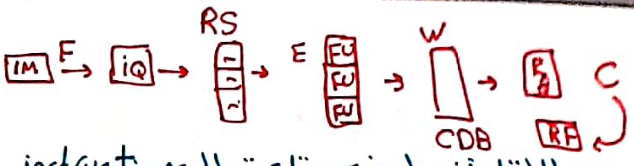
آخر مرحلة ال result بشرحل من Commit unit ال Register file

Reservation stations أي instruction فيها بتستنى ال result لإنها بتعتمد عاشي قبلها بوديها عليه (result)

3 مصادر لل operands & RF / Commit unit / CDB

وهاد بحل مشكلة RAW

Pipeline Stages



الداتا بنحط فيه تاعت الـ instructions

F: Fetch from instr. memory (IM) to instr. queue (IQ).

I: Issue from IQ to reservation stations (RS), reading ready operands from register file (RF).

E: Execute when functional unit (FU) is free and instr. In RS has ready operands. → متغيراتها جاهزين

W: Write result from FU through common data bus (CDB) to reorder buffer (ROB) and RS.

C: Commit results in order from ROB to RF and memory → بال Store بنكتب فيها

- Loads have **FIAMWC**, stores have **FIAC**. **A**: Address calculation

بدل ال E نسمي Address calculation

بال C بنكتب بال Memory

Register Renaming

RS | op | op₁ | op₂ |

- Reservation stations (RS) and reorder buffer effectively provide register renaming
- On instruction issue to reservation station
  - If operand is available in register file or reorder buffer
    - Copied to reservation station
    - No longer required in the register; can be overwritten
  - If operand is not yet available
    - It will be provided to the reservation station by a function unit
    - Register update may not be required

Examples

كتير الـ RAW في أمور ثانية

- Assume superscalar processor of deg
- Name dependence (WAR)

```
mul   x1,x2,x3
add   x4,x1,x5
ld    x5,16(x21)
```

بالـ 5 stage Pipeline لما كنا ننفذهم عالترتيب ماكان في مشكلة بنوم

ممكن نحله بتغير اسم الـRegister

الخوف لما يصير عدم ترتيب في التنفيذ وتعمل الـ ld قبل الـ add وهيك بصير غلط

- Output dependence (WAW)

```
mul   x1,x2,x3
add   x4,x1,x5
ld    x1,16(x21)
```

الخوف مرة ثانية إنه هاي الـinstruction اللي هي ld تسبق الـ mul وهيك خطأ.

ممكن نحلها بإنه نغير اسم الـ Vegister

Speculation

مبني على الـ dynamic Exec.. ونزيد عليه الـ Speculation

الهاردوير بيعمل تنبؤ زي الـ Branch مثلاً بتوقع لو NT/T وبناء عالتوقع بلش ينفذ الـ instruction اللي بعديها as soon as Possible وبس يوصل لل C تبع الـ Branch بيعرف لو تنبأ صح أو غلط

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

Compiler/Hardware Speculation

- Compiler can reorder instructions
  - e.g., move load before branch
  - Can include "fix-up" instructions to recover from incorrect guess
- Hardware can look ahead for instructions to execute
  - Buffer results until it determines they are actually needed
  - Flush buffers on incorrect speculation

جدا مفيد لإنه التنبؤ حاليا بكون صحيح بنسبة 95% تقريبا

Branch Speculation

- Predict branch and continue issuing
  - Don't commit until branch outcome determined → ما بنعمل commit إلا بس
    نعمل لل Branch عشان نتأكد من التنبؤ
- **Example**: Assume a superscalar processor of degree 2 and the branch prediction is not taken. → بقدر يجيب 2 instructions
  Every cycle

```
ld    x1,0(x20)
beq   x1,x2,Skip
I3
I4
```

Load Speculation

- Avoid load and cache miss delay
  - Load before completing outstanding stores
  - Predict the effective address or loaded value
  - Bypass stored values to load unit
- Don't commit load until speculation cleared
- **Example**: Superscalar of degree 3.

```
ld    x1,0(x20)
sd    x2,0(x1)
ld    x3,0(x21)
```

RAW

independent
ما بعتمدوا عبعض
عالأغلب

```
F I A M W C
F I       A C
F I       A M W C
```

ما بتعتمد عاللي فوق .

في حالات نادرة بكونوا dependent لو مثلاً ال Address اللي حطيناه بـ X1 بالصدفة كان بيساوي ال Address اللي موجود بـ X21 وهاي حالة نادرة

لو بدون Speculation حنضطر نتأخر لنعرف بس بال Speculation ✗

Speculation and Exceptions

- What if exception occurs on a speculatively executed instruction?
  - e.g., speculative load before null-pointer check
- Static speculation
  - Can add ISA support for deferring exceptions
- Dynamic speculation
  - Can buffer exceptions until instruction completion (which may not occur)

Exceptions Examples

- Assume superscalar processor of degree 3 with 2 address calculation units
- E1: Predict branch as not take, but resolve to taken. The `ld` has exception in M.

```
beq   x1,x2,L1
ld    x5,16(x21)
```

- E2: Assume first `sd` has exemption in C.

```
ld    x1,0(x20)
sd    x1,0(x21)
sd    x2,16(x21)
```

التصاميم الحديثة فيها commit unit وهي بتساعد كثير
منها إنها بتحل الـ WAW والنتائج بتتخزن حسب الترتيب الأصلي تاعهم.
وجود ROB بيمكنا إنه نعمل speculation وأيضا بيمكنا نتعامل مع الـ Exception بطريقة سليمة

Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?
- Not all stalls are predicable
  - e.g., cache misses
- Can't always schedule around branches
  - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

Does Multiple Issue Work?

The BIG Picture

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

Power Efficiency

much more complex than Single Pipeline

- Complexity of dynamic scheduling and speculations requires power

more transistors more circuites

- Multiple simpler cores may be better

مختلف عن اللي درسناه لانفذ بـ Simple Pipeline زي اللي درسناه

ماكان بحاجة نحط مروحة

تغير خلال 40 سنة تقريبا

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Po
Intel 486 Cisc	1989	25 MHz	5	1	No	1	5
Intel Pentium	1993	66 MHz	5	2	No	1	10
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103
Intel Core	2006	2930 MHz	14	4	Yes	2	75
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	2-4	87
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77

لـ لما ال Pipeline stages بنزيد الاشي اللي بدي اعمله بـ stage وحدة بحير أقل والدائرة بـ stage وحدة بتصير أصغر وبالتالي بتقدر تشتغل على shorter clock Period يعني أعلى Perform...

وهاد كان يزيد ال Power كثير بس سيطروا عليه آخر السنوات وقللوا عدد ال Piepline stages

Contents

4.6 Pipelined Datapath and Control (Review)

4.7 Data Hazards: Forwarding versus Stalling

4.8 Control Hazards

4.9 Exceptions

4.10 Parallelism via Instructions

4.11 Real Stuff: The ARM Cortex-A53 and Intel Core i7 Pipelines

مثالين مهمات
ل modern Process

4.14 Fallacies and Pitfalls

4.15 Concluding Remarks

استخدم من شركة Apple في بعض هواتفها والسبب, انه مصمم يصرف كهرباء أقل

Cortex A53 and Intel i7

برضه يستخدم في
ال high end Processor
زي ال computer PC

§4.11 Real Stuff: The ARM Cortex-A53 and Intel Core i7 Pipelines

Processor	ARM A53	Intel Core i7 920
Market بشو مستخدم	Personal Mobile Device	Server, cloud
Thermal design power	100 milliWatts (1 core @ 1 GHz)	130 Watts
Clock rate	1.5 GHz	2.66 GHz
Cores/Chip	4 (configurable)	4 → ممكن هلانلاقي 6 و 8
Floating point?	Yes	Yes
Multiple issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2 → 2 instruction every cycle	4 → 4 instruction every cycle
Pipeline stages	أصغر لأنه بيشتغل Frequency أقل ← 8	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	طرق متطورة أكثر من ال BHT Hybrid	2-level
1st level caches/core	16-64 KiB I, 16-64 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-2048 KiB	256 KiB (per core)
3rd level caches (shared)	(platform dependent)	2-8 MB

الجيل الأول هاد

بعدين بنشرح

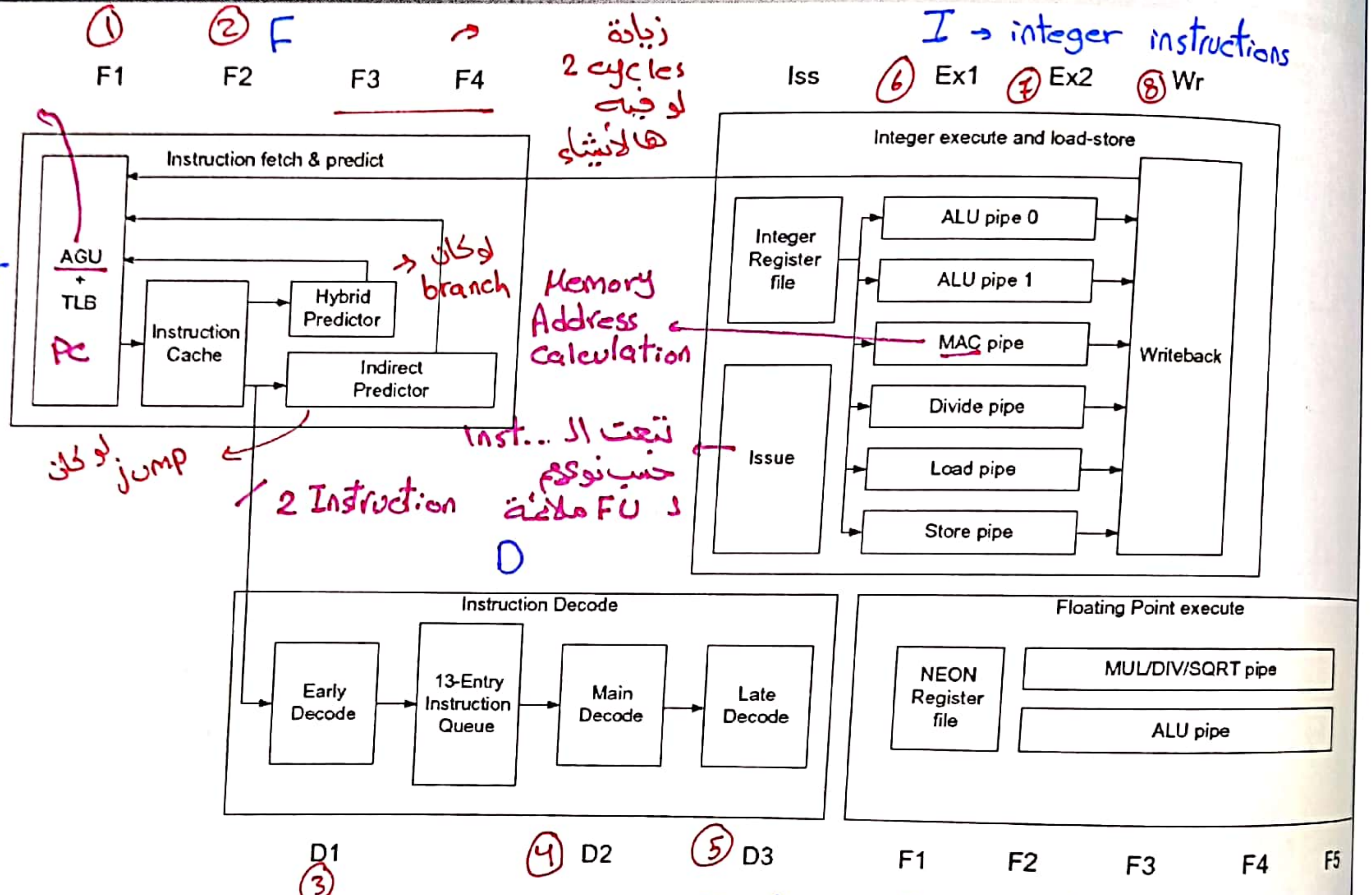
ـه سلسلة ناجحة جدا من ال microprocessors

ARM Cortex-A53 Pipeline

في 8 stages بأحسن الأحوال

Address generation unit

أهم اشي فيه هو ال Instruction cach الي فيه ال instructions الي نستخدم

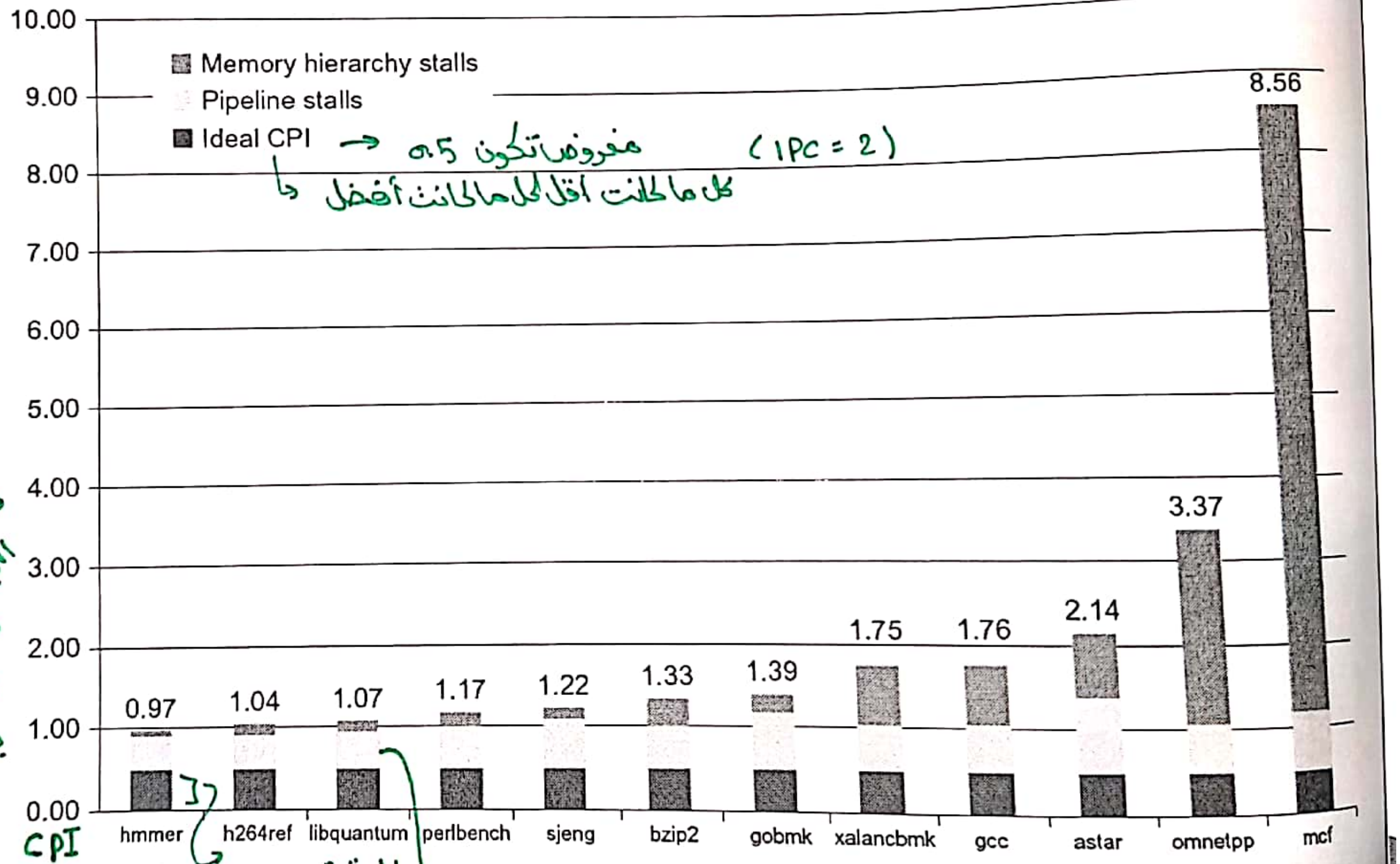


I → Floating point instructions لـ

لو كان Floating في 10 cycles or stages

ARM Cortex-A53 Performance

← بالدراسة هاي قاسوا قديه بياخذ time بال sec وقديه بتنفذ instructions



→ مفروض تكون 0.5 (IPC = 2)

كل ما كانت اقل كل ما كانت أفضل

∴ sec × f → cycle

∴ cycle / IC

فقسموا عدد ال cycles الي بنحتاجهم لتنفيذ كل ال instructions لهذا البرنامج بطلعوا معي هذا ال ratio

CPI

المتوقع

الهاسبين: يا إنه ال instructions في بينها اعتماد كبعض

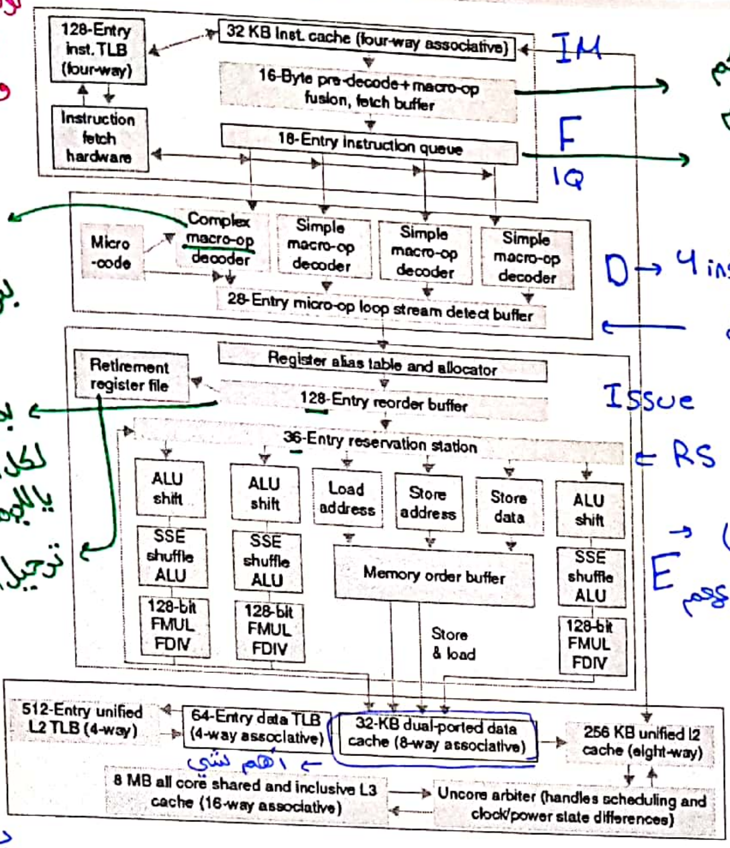
أو إنه ال Processor بحاول يطلب ال data من الcache وهيك بصير → بحتاج فترة طويلة لتيجي ال data من ال Mem

Memory hiraray Stalls

Core i7 Pipeline

→ Processor much more complex
Superscalar Processor with degree 4

كـ ماتغير من ماسبق لهلأ الّي بختلف من وقت لوقت عدد وأحجام ال IQ و RoB و RS وحتى عدد ال FU



128-Entry inst. TLB (four-way)
32 KB Inst. cache (four-way associative)
16-Byte pre-decode + macro-op fusion, fetch buffer
Instruction fetch hardware
18-Entry instruction queue
Micro-code
Complex macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
28-Entry micro-op loop stream detect buffer
Retirement register file
Register alias table and allocator
128-Entry reorder buffer
36-Entry reservation station
ALU shift
SSE shuffle ALU
128-bit FMUL FDIV
ALU shift
SSE shuffle ALU
128-bit FMUL FDIV
Load address
Store address
Store data
Memory order buffer
Store & load
ALU shift
SSE shuffle ALU
128-bit FMUL FDIV
512-Entry unified L2 TLB (4-way)
64-Entry data TLB (4-way associative)
32-KB dual-ported data cache (8-way associative)
256 KB unified l2 cache (eight-way)
8 MB all core shared and inclusive L3 cache (16-way associative)
Uncore arbiter (handles scheduling and clock/power state differences)

IM

ما بنعرف كم Instructions فيهم

F
IQ

بسع 18 مجموعة
→ 18 * 16 byte

complex instructions بترجموها ل simple

D → 4 instructions

ال decode instructions بينحطوا فيه

Issue

← RS
I

بمرحلة ال Issue بنحجز لكل Instruction بال RoB

بالمنصبيص ال commit ترحيل النتائج ال حسب ترتيبها الأصلي

→ (FU) تنفيذ الInstructions بـ Functional unit حسب نوعهم
E
* بقدروا يعملوا ل Floating Point

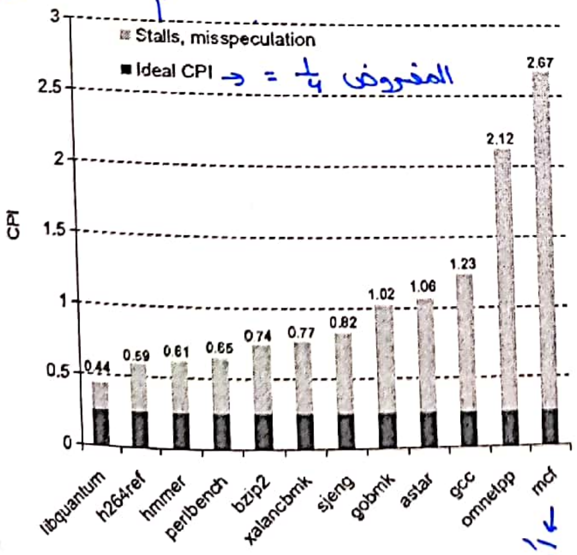
← أهم شي

M

← مش نحكي كثير بالوقت الحاضر

Core i7 Performance

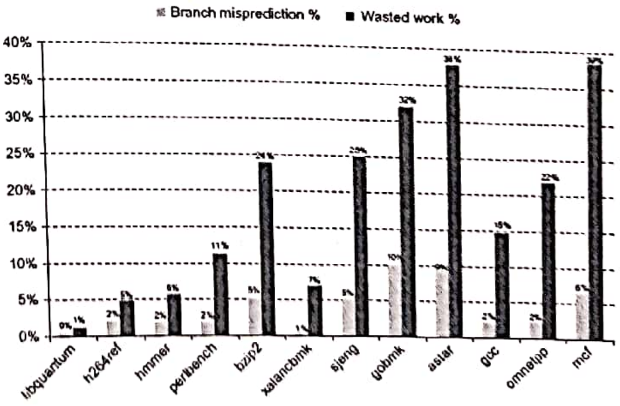
بقلل من الPerformance ↑



Ideal CPI → = $\frac{1}{4}$ المفروض

← أسوأ واحد

أسوأ مايمكن 10%



∴ ٣ مرات أسرع من ال ARM
لأنه بشتغل على high frequency
في dynamic exc... وكلها مع بعضها بتعطي نتائج أفضل
لكن على حساب ال Power .

Contents

- 4.6 Pipelined Datapath and Control (Review)
- 4.7 Data Hazards: Forwarding versus Stalling
- 4.8 Control Hazards
- 4.9 Exceptions
- 4.10 Parallelism via Instructions
- 4.11 Real Stuff: The ARM Cortex-A53 and Inte Core i7 Pipelines
- 4.14 Fallacies and Pitfalls
- 4.15 Concluding Remarks

Fallacies

- Pipelining is easy (!)
  - The basic idea is easy
  - The devil is in the details
    - e.g., detecting data hazards
- Pipelining is independent of technology
  - So why haven't we always done pipelining?
  - More transistors make more advanced techniques feasible
  - Pipeline-related ISA design needs to take account of technology trends
    - e.g., predicated instructions

Pitfalls

- Poor ISA design can make pipelining harder
  - e.g., complex instruction sets (VAX, IA-32)
    - Significant overhead to make pipelining work
    - IA-32 micro-op approach
  - e.g., complex addressing modes
    - Register update side effects, memory indirection
  - e.g., delayed branches
    - Advanced pipelines have long delay slots

Contents

4.6 Pipelined Datapath and Control (Review)

4.7 Data Hazards: Forwarding versus Stalling

4.8 Control Hazards

4.9 Exceptions

4.10 Parallelism via Instructions

4.11 Real Stuff: The ARM Cortex-A53 and Intel Core i7 Pipelines

4.14 Fallacies and Pitfalls

4.15 Concluding Remarks

Concluding Remarks

- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

PAGE
DATE

Slide67 : Dynamic Scheduling :

Single issue → 1 instruction يعني بجيب بس

	1	2	3	4	5	6	7	8	9	10
ld X31 , 20 (X21)	F	I	A	M	W*	C				
add X1 , X31 , X2		F	I	–	–	E	W	C		
sub X23 , X23, X3			F	I	E	W*	–	–	C	
andi X5, X23 , 20				F	I	–	E	W	–	C

* بنعملهم I بالترتيب الأصلي تبعهم وكذلك ال C ولكن E مش عالترتيب الجاهز بنفذ

Slide 69 :

3 things يعني أي stage بتقدر تعمل

* Triple Issue : Name dependence (WAR)

نفترض إنو ال mul تحتاج 3cycles بال Ex

	1	2	3	4	5	6	7	8	9	10
mul X1 , X2, X3	F	I	E	E	E	W	C			
add X4, X1 , X5	F	I	–	–	–	–	E	W	C	
ld X5 , 16(X21)	F	I	A	M	W	–	–	–	C	

نجمع محتويات X21 مع 16

* الكتابة على ال RF بتتم بال C مش ال W

* Triple Issue : output dependence (WAW)

نفترض إنه ال mul تحتاج 3 cycles بال Ex

كتبت ب ROB مش بال RF

	1	2	3	4	5	6	7	8	9	10
mul X1 , X2, X3	F	I	E	E	E	W	C →			
add X4, X1 , X5	F	I	–	–	–	–	E	W	C	
ld X1 , 16 (X21)	F	I	A	M	W	–	–	–	C →	

هون بكتب ع X1

هون بكتب X1

لـ هاي كتبت بال reorder buffer مش بال RF

وهون X1 بتخزن بال RF لإنو نتيجة X1 النهائية

Slide 72 :

Example: Assume a superscalar processor of degree 2 and the branch predection is not taken. (Correct prediction)

	1	2	3	4	5	6	7	8	9	10
ld X_1 , 0 (X_{20})	F	I	A	M	W	C				
beq X_1 , X_2, Skip	F	I				E	W	C		
I_3		F	I	E	W			C		
I_4		F	I	E		W			C	
...										
Skip:										

فحنعمل C لـ I3
بنفس الـ cycle عادي
لما عملنا C اكتشفنا
انه التنبؤ صح.

بنعملهم F بالـ →
2 cycle لإنه توقعنا
انه not taken
الـ Branch وتوقعنا
صحيح كان.

أخرنا الـ W لإنه مابقدر أعمل 3 write
بنفس الـ cycle لإنه البرنامج of degree 2
فغالبا بقدر أعمل 2 من C / W / M / I / F
بس الـ E بعتمد قديه عندي Functional unit

E A

Slid 72 :

Example : Assume a superscalar processor of degree 2 and the Branch Prediction is Not taken. (Incorrect prediction)

	1	2	3	4	5	6	7	8	9	10
ld X_1 , 0 (X_{20})	F	I	A	M	W	C				
beq X_1, X_2, Skip	F	I				E	W	C		
I_3		F	I	E	W			n		
I_4		F	I	E		W		n		

Skip :									F	

بس عملنا E
اكتشفنا إنه التنبؤ
خاطئ

ما بنعملهم C إلا
بعد ما الـ Branch →
ينعملوا C

Scanned with CamScanner

Slide 73 :

Example : Load speculation. Assume a superscalar processor of degree 3. Predict the second load does not depend on the store.
(Correct prediction)

	1	2	3	4	5	6	7	8	9	10
ld X1 , 0(X20)	F	I	A	M	W	C				
sd X2, 0 (X1)	F	I				A	C			
ld X3, 0 (X21)	F	I	A	M	W		C			

C → بنكتب عال Mem
بال Commit

Slide 73 :

Example : Load speculation. Assume a superscalar processor of degree 3. Predict the second load does not depend on the store. (Incorrect prediction).

→ اكتشفنا إنه $X_1 = X21$

	1	2	3	4	5	6	7	8	9	10
ld X1 , 0 (X20)	F	I	A	M	W	C				
Sd X2, 0 (X1)	F	I				A	C			
ld X3, 0 (X21)	F	I	A	M	W		n	F		

] → RAW

على فرض إنه $X_1 \neq X21$

لإنها أخذت قيمة خاطئة
بنعيدها مرة ثانية لإنها كانت خطأ
بالبداية وهيك بنمشي صح.

Slide 75 : Speculation and Exceptions

Example 1: Assume superscalar processor of degree 3 with 2 address calculations units.

Predict branch as not taken, but resolve to taken. The ld has exception in M.

هون بنكتشف إنها Taken

	1	2	3	4	5	6	7	8	9	10
beq X1, X2, L1	F	I	E	W	C					
ld X5, 16 (X21)	F	I	A	M	n					

جبناها بناء عالافتراض تاعنا إنه ال beq

← Not taken

هون اكتشفنا إنه فيه Exception

معناها لازم نروح لل EHS ← Exception Handling Subroutine

بس بالتصاميم الحديثة إذا فيه Exception إحنا مابنعالجه إلا بال Commit بس كمان

احنا بننفذ ال ld عالفاضي لإنه ال beq ← taken فببساطة بنعملها null

وال Exception ولا كأنه صار

Example 2 : Assume superscalar processor of degree 3 with 2 address calculation units.

Assume First sd has exception in C.

	1	2	3	4	5	6	7	8	9	10
ld X1 , 0 (X20)	F	I	A	M	W	C				
sd X1 , 0 (X21)	F	I	A	-	-	n				
sd X2, 16 (X21)	F	I	-	A	-	n				
EHS :							F			

عرفنا إنه فيه exception فبنحول من C إلى null

وبروح لل EHS وبس أخلص والأمور تكون ملائمة برجع بعمل F لل instruction اللي عملتلهم null وبرجع أنفذهم مرة ثانية

Chapter 5

Large and Fast: Exploiting Memory Hierarchy

التركيز على
ال Memory ، الهدف، اني أصمم Memory حديثة تكون كبيرة وبنفس الوقت سريعة وهذا مش موضوع سه

Adapted by Prof. Gheith Abandah

Contents

→ حندرسهم من ناحية
ال Memory

صفة من صفات البرامج لما نتعامل مع ال Memory
وبنفس الوقت موجودة بحياتنا.

مهمة

Principle of Locality

- Programs access a small proportion of their address space at any time → مثلا برنامج معين ما بنشتغل عكل إمكانياته بنشتغل عشغلات معينة فيه بس.
- Temporal locality
  ← من كلمة time
  - Items accessed recently are likely to be accessed again soon → إذا اشي انت طلبته هسا في احتمال كبير انك تطلبه بعد شوي.
  - e.g., instructions in a loop, induction variables
    ↳ Ex: for (i=0 . . .)
- Spatial locality
  ← جاية من كلمة space
  - Items near those accessed recently are likely to be accessed soon → إذا في اشي عملته access احتمال اعمل access لل locations اللي قريبين منه
  - E.g., sequential instruction access, array data

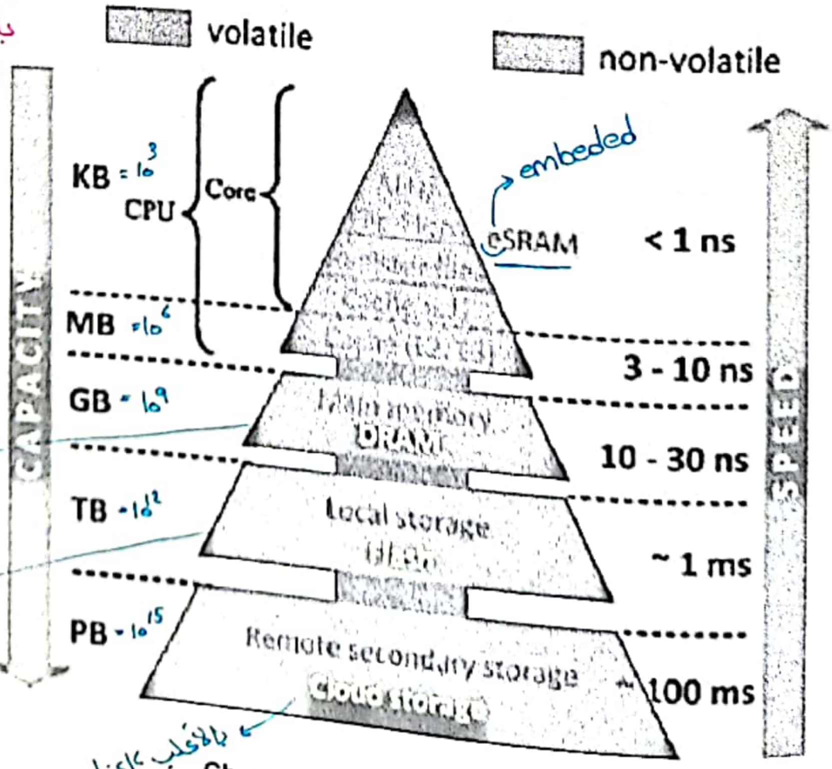
Taking Advantage of Locality

المومن ال disk ولما نسكر الجهاز محتوياتها ما يروح

- Memory hierarchy ← إنه الـ Mem تكون أكثر من نوع وأكثر من طبقة
- Store everything on disk or Flash ← مثل البرامج اللي مخزنة بـ Smart Phones
  كبير وبطيئ
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory ← مبنية مش سريعة كثير ورخيصة وهذا المستوى الثاني بعد الـ Flash / disk
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory ← الشغلات اللي شغالين عليها والقريبة منها و هاي سريعة لكن صغيرة
  - Cache memory attached to CPU

Memory Hierarchy

* تصميم مثل هذا التصميم بالنسبة لل Processor مارح يحصل مع بعض Fast and large



هي اللي بنستخدمها أكثر ← Fast →

دائماً DRAM ←

Large → بالأغلب disk / Flash ←

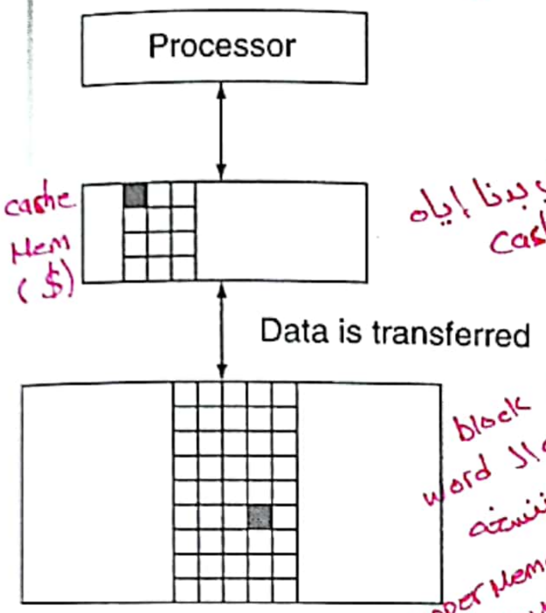
disk بالأغلب ←

Memory Hierarchy Levels

اسم ثاني إله

م ال Memory بتتقسم لـ blocks
يعني لما نجيب data من تحت لفوق
بنجيبها blocks مش متفردة

- Block (aka line): unit of copying
  - May be multiple words 64 bytes → لل block الواحد
- If accessed data is present in upper level → اذا الProcessor طلب شغلة من ال Cashe بنقول عن نوع هاي ال access إنها Hit
  - Hit: access satisfied by upper level ← لقينا اللي بدنا إياه بال cashe
    - Hit ratio: hits/accesses → $= \frac{\text{hits}}{\text{hits + misses}}$
- If accessed data is absent → الداتا اللي بدنا ياها مش موجودة بال cashe موجودة بال lower Memory level
  - Miss: block copied from lower level ← بناخد ال block اللي فيه ال word المطلوبة وبننسخه للupper Memory level
    - Time taken: miss penalty → الزمن الإضافي اللي بحتاجه بين ما أجيب ال data
    - Miss ratio: misses/accesses
= 1 – hit ratio
  - Then accessed data supplied from upper level



Ex: hit ratio = 95 % → دايمًا عالي
miss ratio = 100% - 95% = 5%

Contents

5.1 Introduction

5.2 Memory Technologies → أنواع ال Memory Technologies

Introduction

SRAM

DRAM

Flash

Disk Storage

Memory Technology (2012)

- Static RAM (SRAM) → الأسرع ولكن الأغلى
  - 0.5ns – 2.5ns, \$2000 – \$1000 per GB

لل gega ما
byte
الواحدة

- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$10 – \$20 per GB
- Flash memory
  - 5,000ns – 50,000ns, \$0.75 – \$1.00 per GB
- Magnetic disk
  - 5ms – 20ms, \$0.05 – \$0.10 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

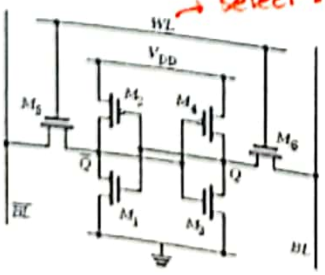
1970 منذ ← → Static Random Access Memory

SRAM Technology

"SRAMs lose data content on power loss"

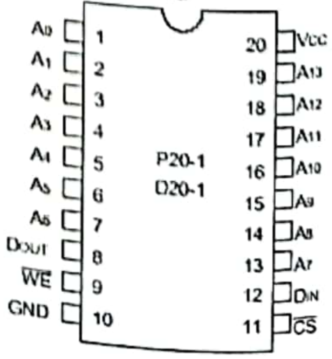
- Static RAM
- 6-8 transistors per bit
- Fast but not dense → كثيف
- Often has standby mode → وضعية الاستعداد

→ select line



IDT6167SA/LA
CMOS Static RAM 16K (16K x 1-Bit)

Pin Configurations



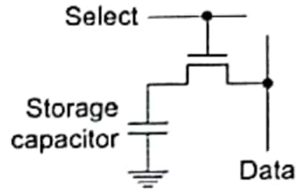
$2^{14} = 2^4 \times 2^{10} = 16$ k bits ←

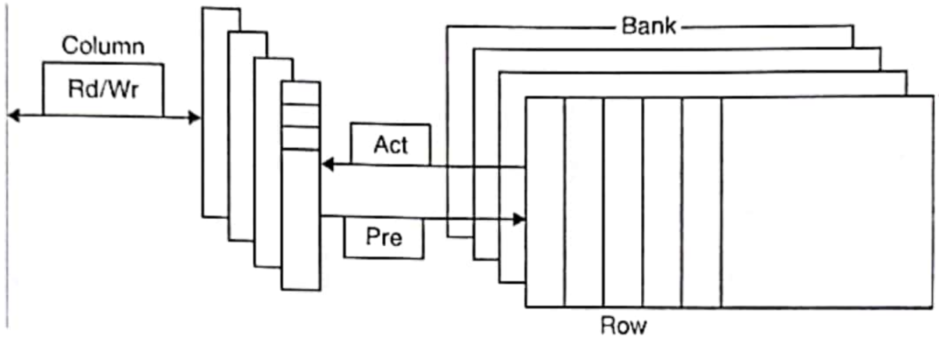
1969 ← → Dynamic Random Access Memory

DRAM Technology →

بتعتمد إنه تخزن المعلومات
ب Capacitor وبتحتاج
one transistor

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM "row"



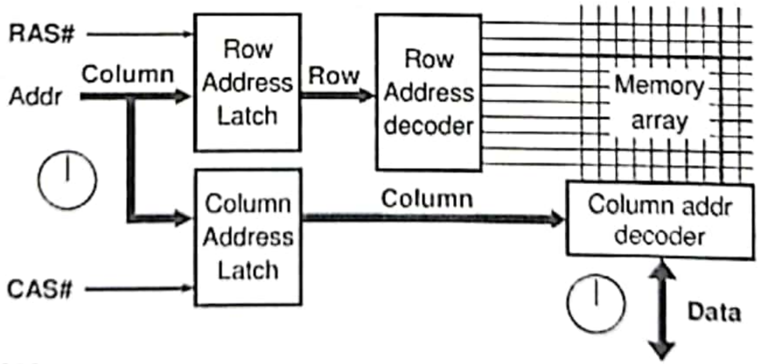


* بمساحة 1 transistor
بتخزن 1bit

* DRAM : Refresh Each memory cell 16 times Per Second (or more) or risk losing content.

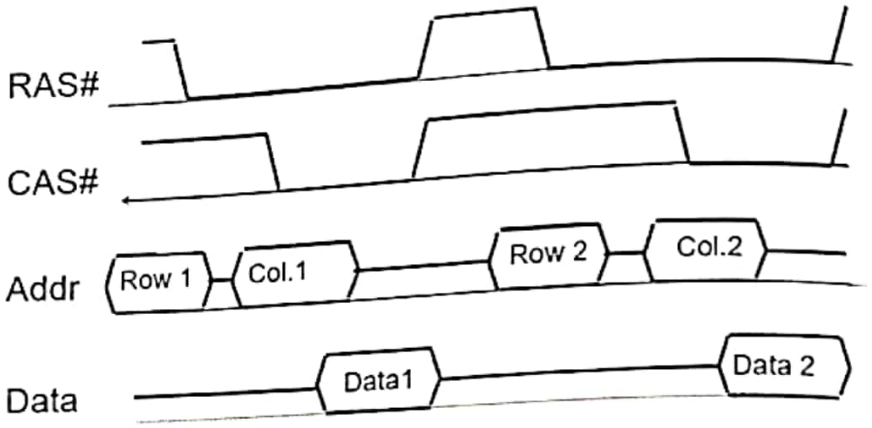
Classic DRAM

Basic DRAM chip



- DRAM access sequence
  - Put Row on addr. bus
  - Assert RAS# (Row Addr. Strobe) to latch Row
  - Put Column on addr. bus
  - Wait RAS# to CAS# delay and assert CAS# (Column Addr. Strobe) to latch Col
  - Get data on address bus after CL (CAS latency)

Classic DRAM



Every access - individual

2009-2013 ©S.Maciulevičius

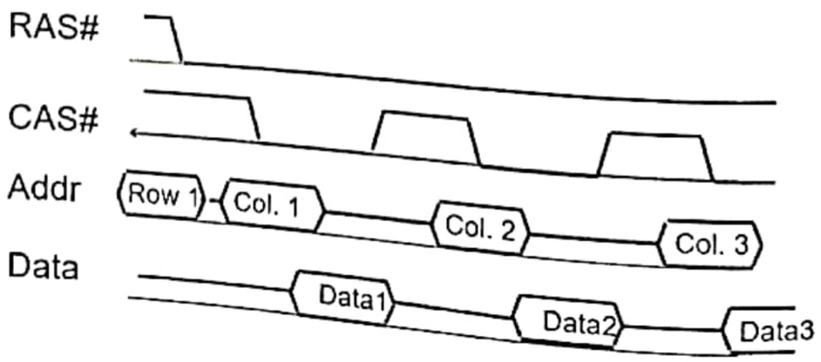
- Low bandwidth

Advanced DRAM Organizatio

التطورات

- Access an entire row and save it in a **row buffer**.
- **Fast page mode**: supply successive words from the row buffer with reduced latency

← وقت الاستجابة

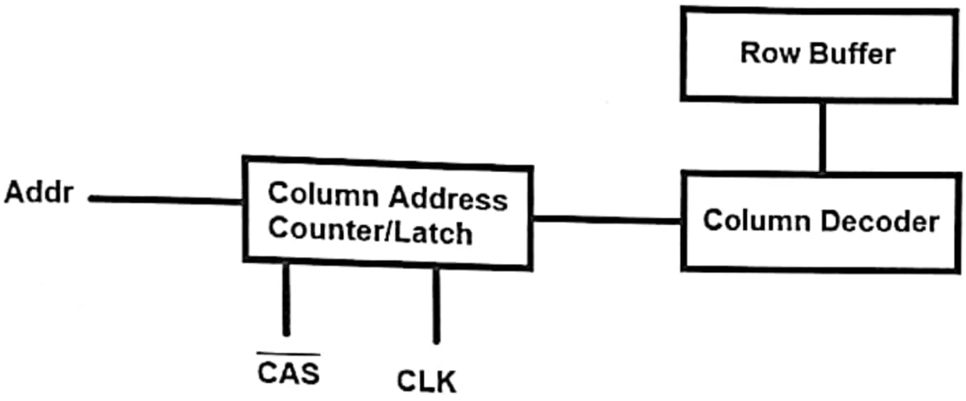


Scanned with CamScanner

Advanced DRAM Organization

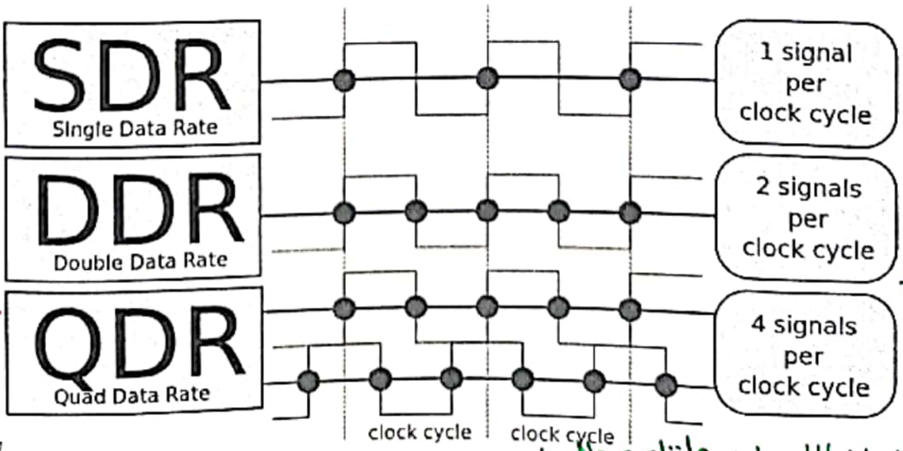
متزامن

- **Synchronous DRAM** (SDRAM) has a counter that increments the column address using a clock signal.



Advanced DRAM Organization

- **Double data rate** (DDR) SDRAM
  - Transfer on rising and falling clock edges
- **Quad data rate** (QDR) SDRAM
  - Separate DDR inputs and outputs



معظم الأجهزة بنستخدمه →

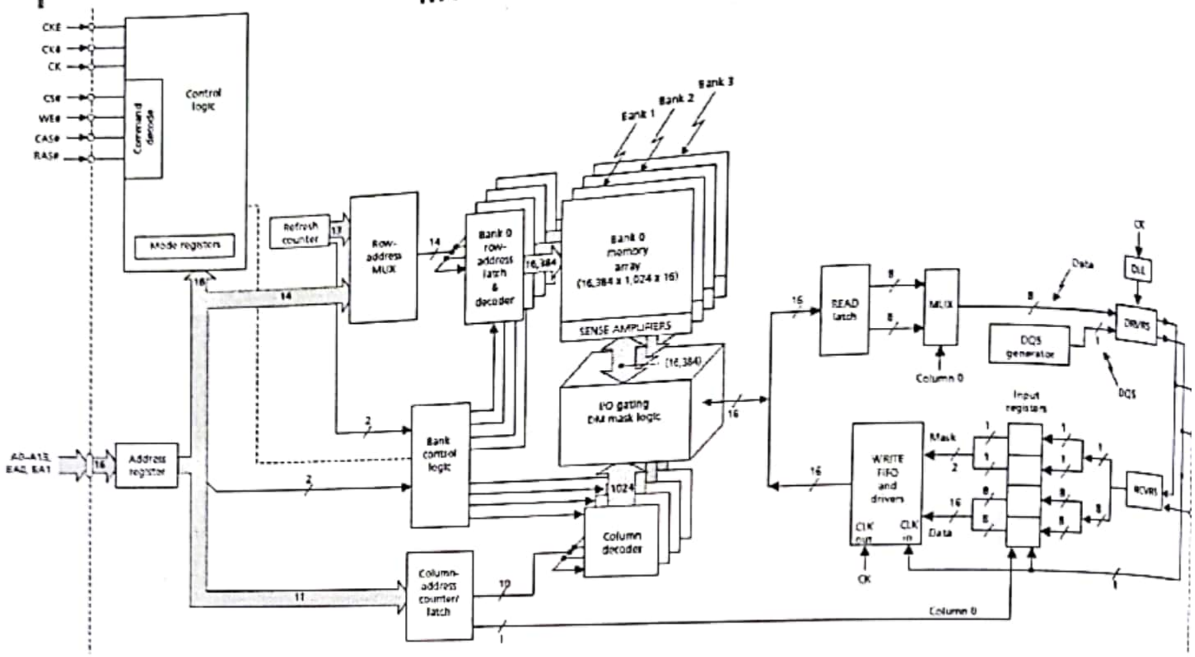
بدل 1 port ←
فيه 2 ports

فكرته, انه في clock
والداتا بتنتقل
مرتين كل cycle
عند الـ Raising Edge
وعند الـ falling Edge
فبال cycle الواحدة
بيتم انتقال 2 data
transfers
وهذا النوع من الـ Mem الـ bandwidth تاعها عالي

Micron 1Gb DDR-SDRAM

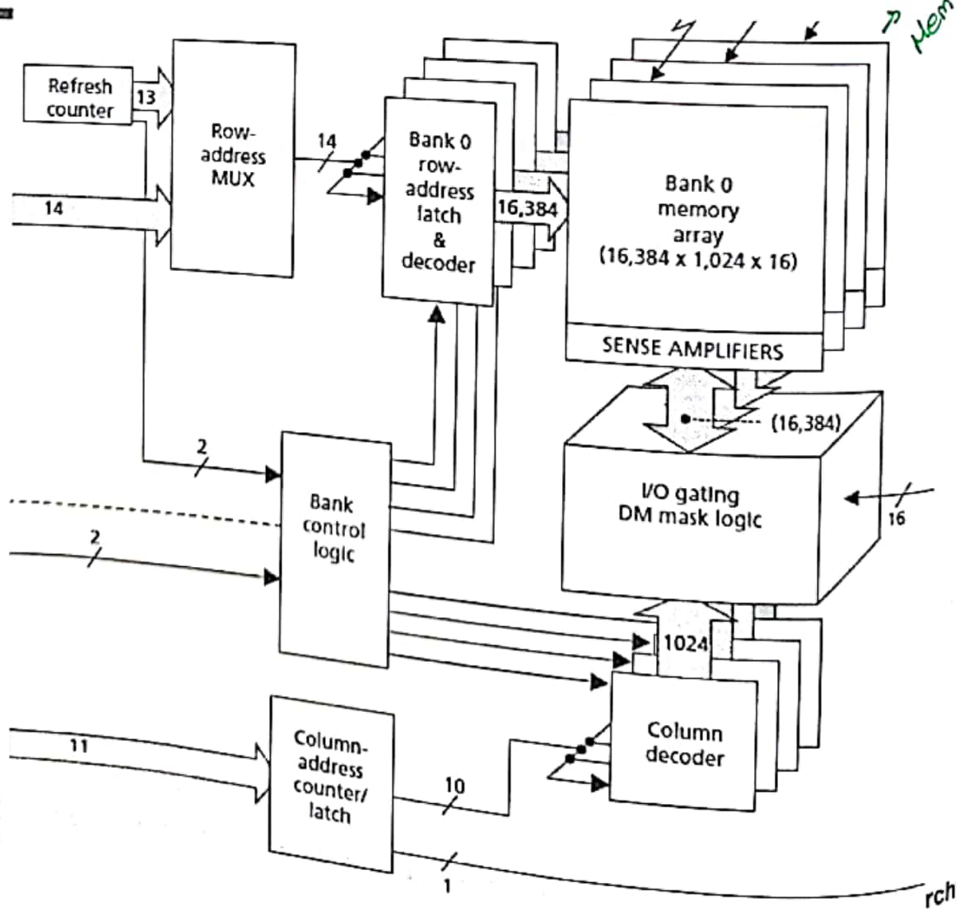
شركة ←

MT46V128M8 – 32 Meg X 8 X 4 Banks, Datasheet



Micron 1Gb DDR-SDRAM

داخل أكتر شي → Mem array



DRAM Generations

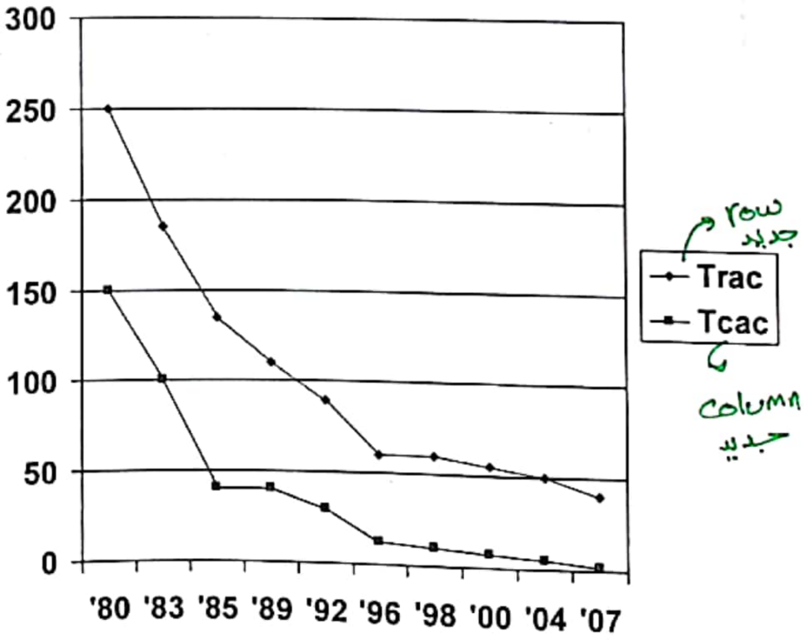
Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	$1,500,000	250 ns	150 ns
1983	256 Kibibit	$500,000	185 ns	100 ns
1985	1 Mebibit	$200,000	135 ns	40 ns
1989	4 Mebibit	$50,000	110 ns	40 ns
1992	16 Mebibit	$15,000	90 ns	30 ns
1996	64 Mebibit	$10,000	60 ns	12 ns
1998	128 Mebibit	$4,000	60 ns	10 ns
2000	256 Mebibit	$1,000	55 ns	7 ns
2004	512 Mebibit	$250	50 ns	5 ns
2007	1 Gibibit	$50	45 ns	1.25 ns
2010	2 Gibibit	$30	40 ns	1 ns
2012	4 Gibibit	$1	35 ns	0.8 ns

بيقل السعر مع الزمن

بتتحسن مع الزمن

DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



row جديد

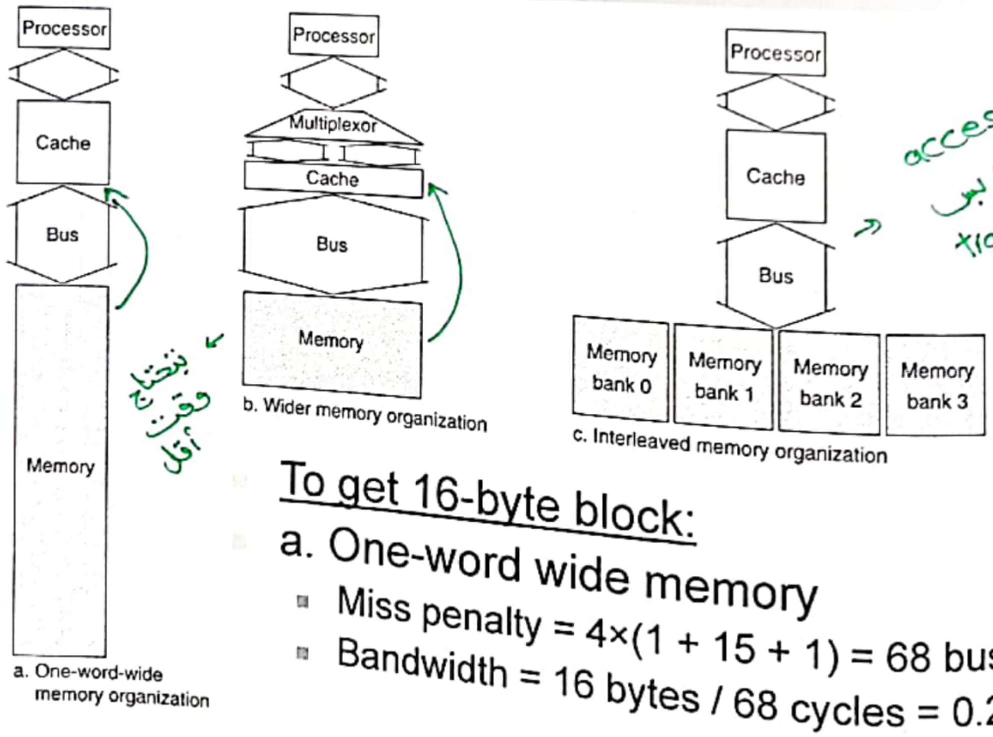
column جديد

DRAM Performance Factors

- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- DRAM banking
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth

Increasing Memory Bandwidth

تصاميم مختلفة لل Memory



a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

بتحتاج وقت كبير

بتحتاج وقت أقل

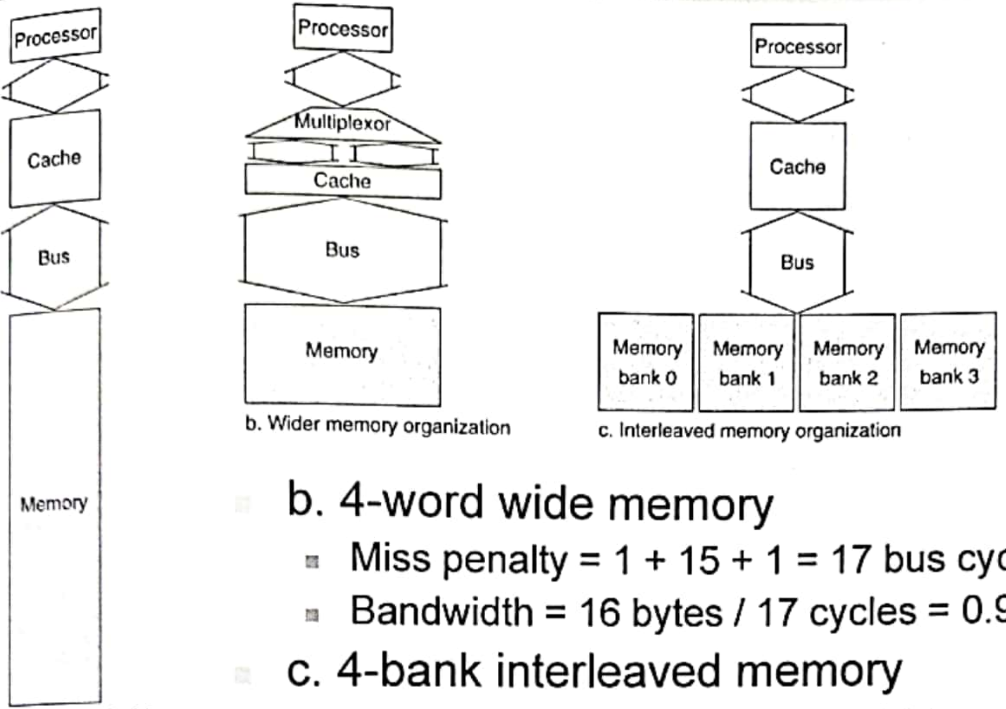
ال access بياخذ وقت بس ال transfer سريع

To get 16-byte block:

a. One-word wide memory

- Miss penalty = 4×(1 + 15 + 1) = 68 bus cycles
- Bandwidth = 16 bytes / 68 cycles = 0.24 B/cycle

Increasing Memory Bandwidth



a. One-word-wide memory organization

b. Wider memory organization

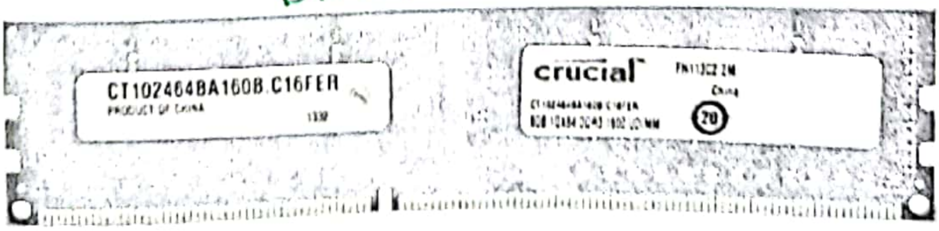
c. Interleaved memory organization

- b. 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- c. 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Increasing Memory Bandwidth

- d. DDR-SDRAM
  - Miss penalty = 1 + 15 + 4×0.5 = 18 bus cycles
  - Bandwidth = 16 bytes / 18 cycles = 0.89 B/cycle

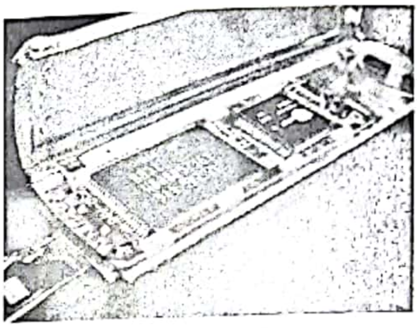
DIMM



Flash Storage

سريع
بس بخرب
مع الوقت

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

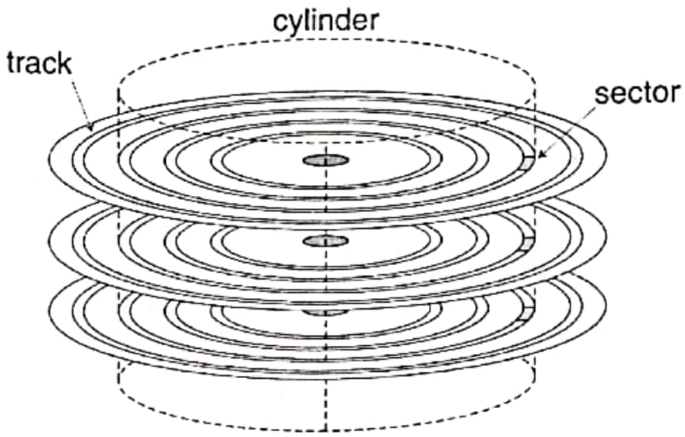


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- If actual average seek time is 1ms
  - Average read time = 3.2ms

Disk Access Example 2

- Given
  - 15,000rpm, 2MB/cylinder
- Sustainable peak transfer rate?

Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Contents

5.1 Introduction

5.2 Memory Technologies

5.3 The Basics of Caches → مبدأ مستخدم بشكل واسع بحياتنا
وبسهل علينا.

- Direct Mapped Cache
- Cache Example

] Reading

- Larger Block Sizes
- Writing to the Cache
- Example: Intrinsity FastMATH

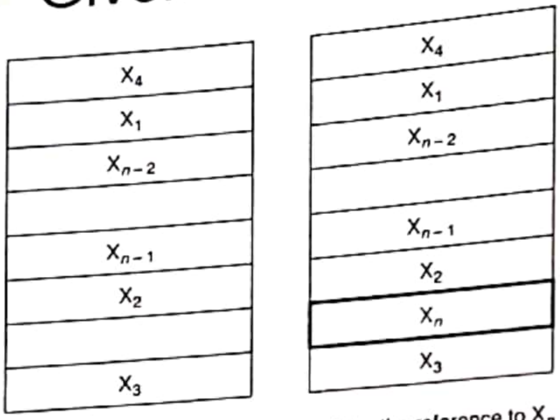
Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU

ال Mem تتكون من طبقات قريبة من ال CPU

- Given accesses $X_1, \ldots, X_{n-1}, X_n$

أبسط حالة Cache واحد



a. Before the reference to X_n

b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

أبسط أنواع ال cache

- Location determined by address

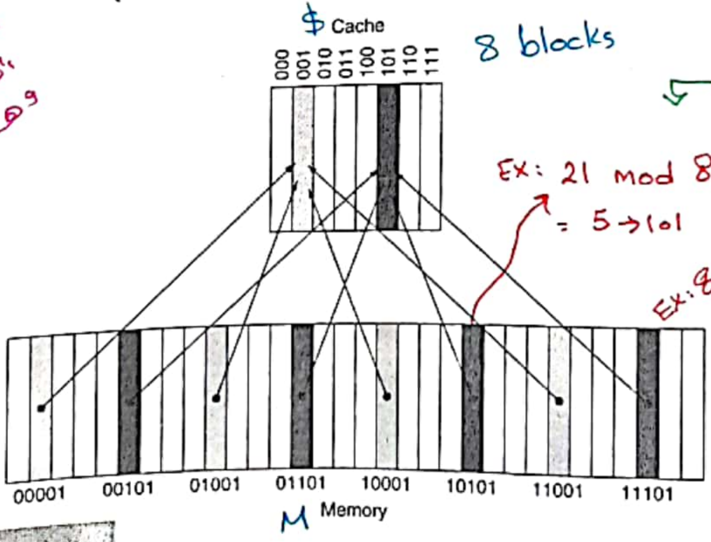
هو بحدد مكان ال block في ال Cache

- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)

أي اشي بدنا نجيبه من ال Memory بالمكان واحد فقط وهذا بعتمد عال address نفسه

بناخد ال block Address وبنقسم على عدد ال blocks بال cache وبناخد الباقي

8 blocks



Ex: 21 mod 8 = 5→101

- #Blocks is a power of 2

Ex: $8 = 2^3$

- Use low-order address bits

32 blocks

least significant bits

Tags and Valid Bits

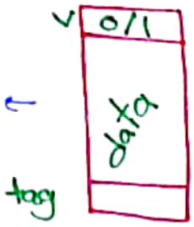
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag اول

لما نجيب block من
الـ Memory مش
بس بنخزن الـ data
تاعته منخزن كمان
الـ Address
تاعته

بنحتاج بس الـ high Address bit
يعني بنحتاج أجيب الـ bits الباقيات بس لأنه آخر 3 أنا أخذتهم أصلاً

- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

لما نخزن tag و لما الـ Processor يطلع address للـ Cache بنفحص
الـ Address اللي طالبه مع الـ tag الموجود بالـ cache إذا تساووا
معناته الـ data اللي بدنا ياها موجودة بالكاش، إذا ما تساووا معناته
في عنا Miss.



Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

< 32 bit > or < 1 w >

	Index	V	Tag	Data
0	000	N = 0		
1	001	N = 0		
2	010	N = 0		
3	011	N = 0		
4	100	N = 0		
5	101	N = 0		
6	110	N = 0		
7	111	N = 0		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

بالعشري

لإنه مافي بال cache
block اشي .

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

هاي ال data وحطيناها بال data
Part

وبعدين ال Most significant bit بنحطهم بال tag بال cache عشان بالمستقبل نعرف شو
في هون

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y=1	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

نفس ال
data اللي
بدنا ياها

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

ال tag اللي فيه
نفس اللي أنا طالبه

Cache Example

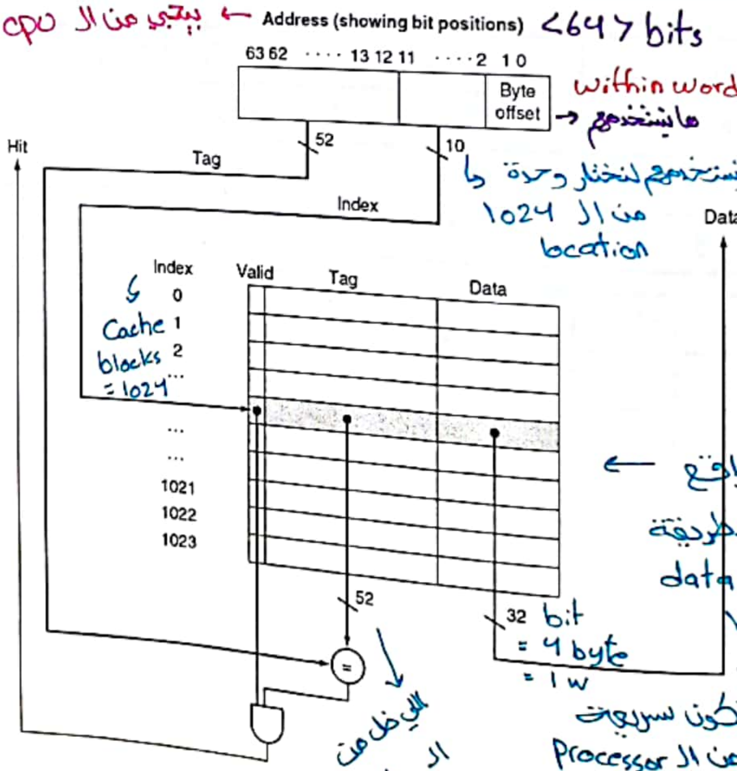
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

ك لإنطال وها tag اللي موجود فيه مش نفس اللي أنا طالبه

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision

ك تصميم لـ cache منطقي أكثر



بيجي من الـ cpu ←

<64> bits

within word

→ مابنستخدمه

ينستخدمه لنختار وحدة ما من الـ 1024 location

$2^{⑩} = 1024$

Data → بتروح للـ cpu

ك Cache blocks = 1024

4 bytes

لو بدنا نختار منهم بنحتاج 2 bits ← 00

ك 01

ك 10

ك 11

اللي ظل من الـ 64

32 bit = 4 byte = 1 w

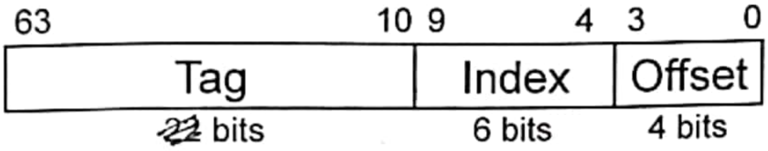
هذا في الواقع ← Memory مرتبة بطريقة انه نحط فيها data و tag و Valid وغالبا تصنع من SRAM حتى تكون سريعة ولتكون قريبة من الـ Processor

Example: Larger Block Size → أكبر من 1w

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?

← هذا ال address كامل 1200 فبدي الاقي ال block Address فبقسم على 16

- Block address = ⌊1200/16⌋ = 75
- Block number = 75 modulo 64 = 11 → Location 11 بال cache



54

<64 bit> ↓

$2^{⑥}=64$

↳ بيمكنونا نختار 1 من 16

$2^{④}=16=\lg_2 16=\lg_2 2^4=4$

Block Size Considerations

- Larger blocks should reduce miss rate → لإنه لما يصير عندك Miss لـ 1w بنجيب block كامل فالبتالي
  - Due to spatial locality

بنستفيد من ال spatial locality وإنه ال Processor عالغالب حيطلب ال near by words الي قبلها أو بعدها ، فلما أجيب ال block كامل بوفر عحالي Some misses بالمستقبل القريب.

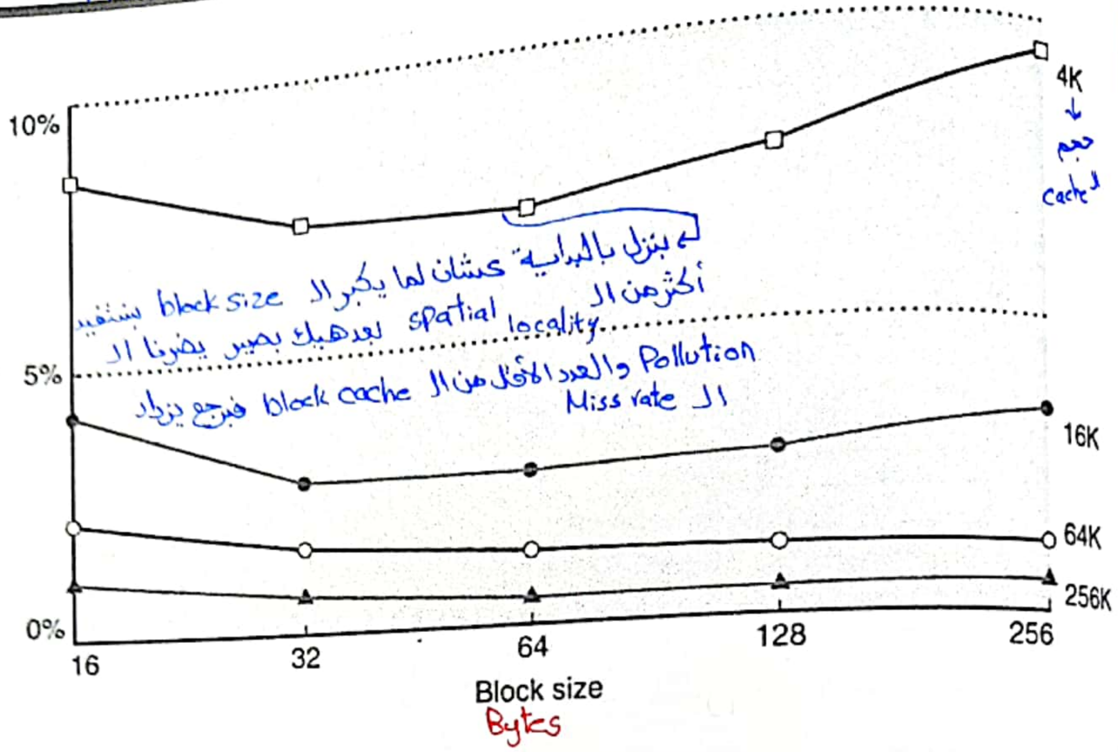
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate → كل ما أكبر ال block الواحد بصير عندي عدد blocks أقل
  - Larger blocks ⇒ pollution → يعني بنزيد عال cache بـ data ما بنحتاجوا
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

↓ بمجرد ما توصل ال critical word فما تستنى ال block كامل ← بدل ما تيجي ال critical word بحيلها لل forwording Processor

حسب ترتيبها بالمثال بنبعت ال address كاملها أول اشي بعدين بنطلب باقي ال block

Block Size Considerations

لـ العلاقة بين حجم ال cache وال block size وال Miss rate



4K ← حجم ال cache

لـ بنزل بالبداية عشان لما يكبر ال block size بنستفيد أكثر من ال spatial locality بعدهيك بصير يضرنا ال Pollution والعدد الأقل من ال block cache فيرجع يزداد ال Miss rate

Miss rate ← كل ما كان أقل أحسن

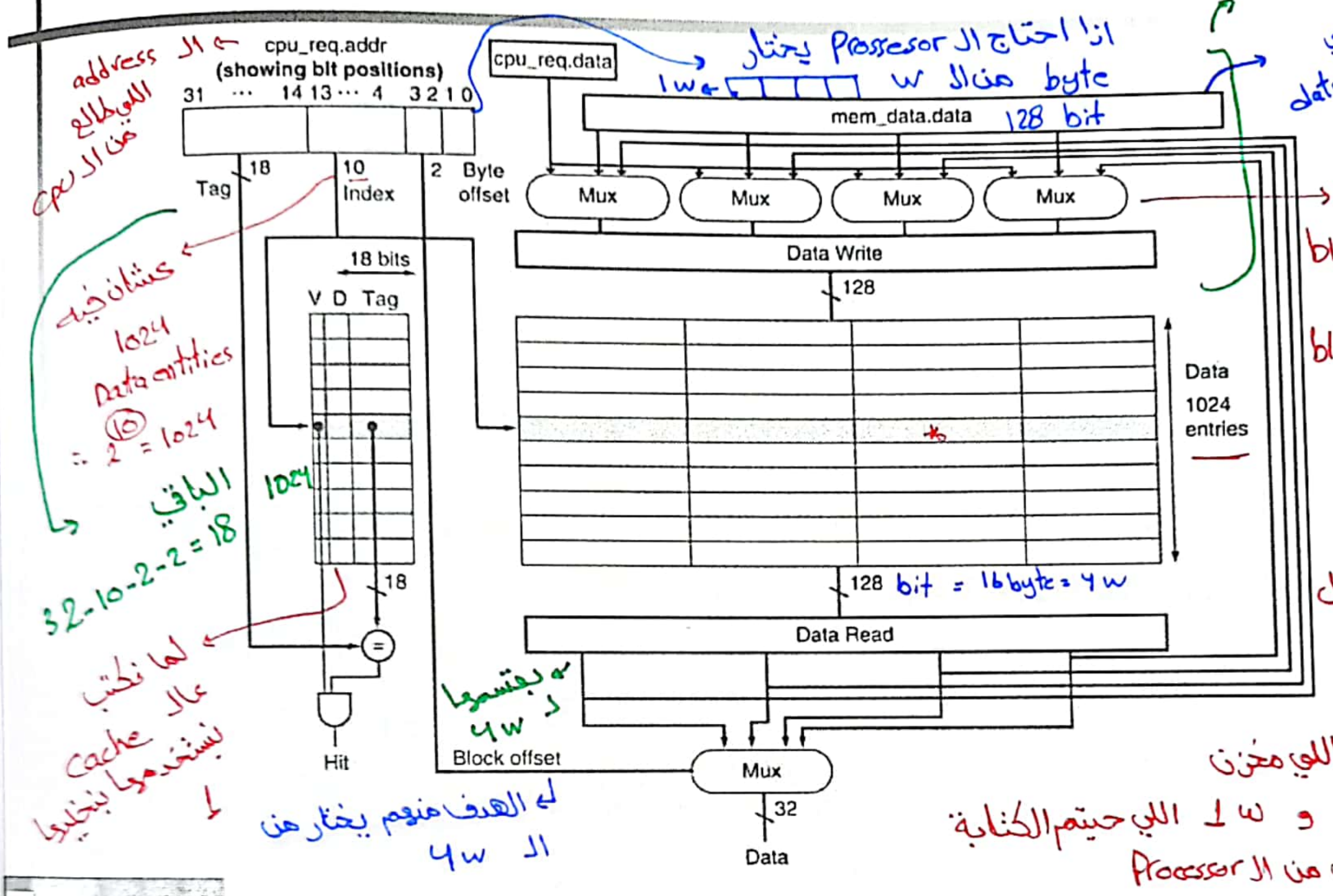
Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss → لما يكون في cache miss ال data ما بتكون موجودة بال cache
  - Stall the CPU pipeline ← وبدنا زمن أطول لنجيبوا فيعملوا
  - ← اللي صار عليها ال Miss
  - Fetch block from next level of hierarchy لـ بطلب من اللي تحته ما لقيت بطلب من اللي تحت تحته وهيك
  - Instruction cache miss
    - Restart instruction fetch → بطلب ال instruction مرة ثانية
  - Data cache miss
    - Complete data access

← بده يعمل Instruction load / store أو data و ما يلاقيها بتكون data miss

تصميم ل cache بنقدر نكتب منه ونقرأ عليه.

Writing to the Cache



← ال address المطالع من ال cpu

اذا احتاج ال Prossesor يختار byte من ال w

1w ←

128 bit

عشان فيه 1024 Data entities
∴ $2^{10} = 1024$

1024

الباقي
32-10-2-2 = 18

لما نكتب عال cache بنستخدموا بنخليوا 1

128 bit = 16 byte = 4w

← بتقسموا ل 4w

له الهدف منهم يختار من ال 4w

to write to the cache
بنستخدمه لما يجير في cache Miss فبنجيب ال data من ال Memory

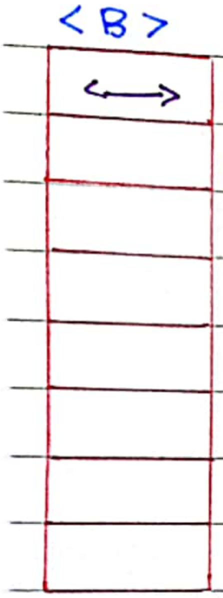
كشان لما بدنا نكتب حنكتب جزء من ال block مش كله فبداية بنقرأ ال block كامل و بدخله عال MUX واللي ما تغيروا بدخلوا زي ما هم وال MUX المسؤول عن ال word اللي بدها تتغير ف بوباخدهم من اللي مخزن بال cache و 1 w اللي حيتم الكتابة كليوا بتيجي من ال Processor

* الكتابة بال cache الها 3 مصادر : 1- من ال Processor
2- من ال Memory
3- من ال cache للشي اللي مابه يتغير

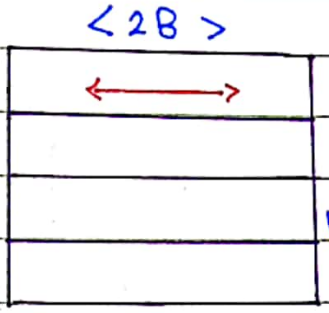
Write-Through

(في أكثر من طريقة للتعامل مع ال write)

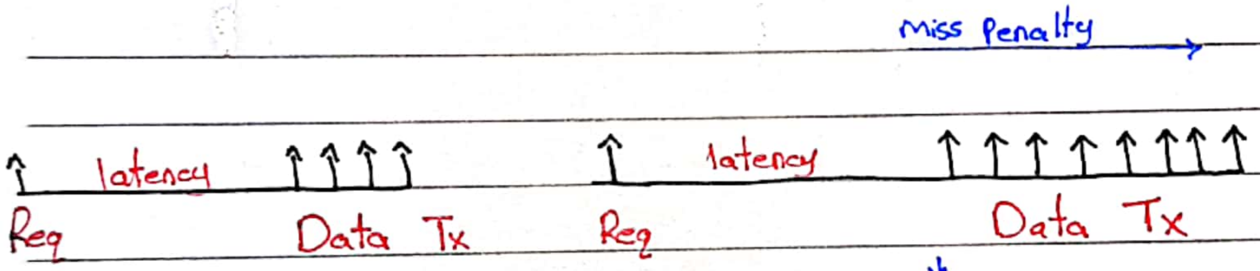
Slide 43 : Block Size Considerations



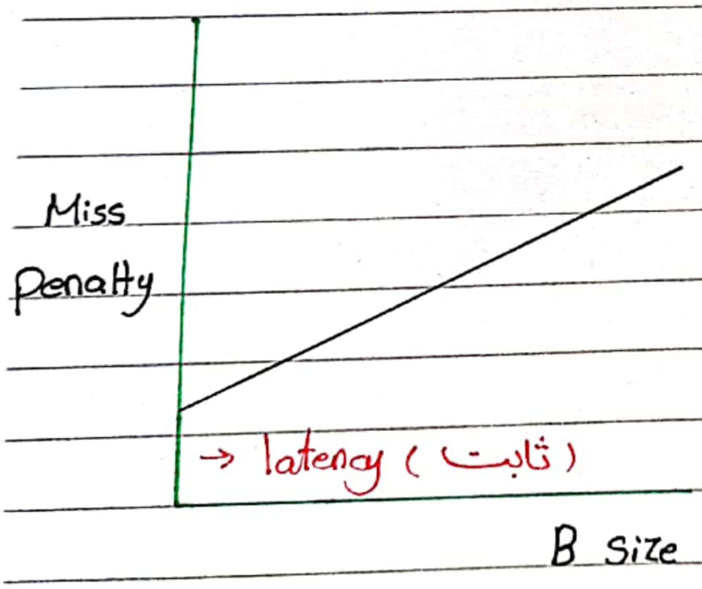
↳ ال block size
صغير فبكون عدد ال blocks
كبير
(هذه أفضل)



→ اذا ضاعفت ال block size,
وبنفس حجم ال Memory
اللي عندك بصير عدد ال blocks
أقل



رح يزيد الزمن



Write-Through (في أكثر من طريقة للتعامل مع ال write)

← هون بنقالك بكتب عال cache بس . بس ما بياخذ بعين الاعتبار إنه ممكن نفقد ال data وتصير معلومات ال $ مختلفة عن معلومات ال M

← ال processor بده يكتب شيء وال address اللي بده يكتب عليه موجود بال cache

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent ← بتصير المعلومات بال cache مختلفة عن اللي بال Memory

↳ مكان مؤقت لحفظ ال data

- أول حل → Write through: also update memory → عشان يحلوا المشكلة اللي فوق.
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer → عبارة عن Memory سريعة
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full → بخليه يكتب عال cache وال write buffer لحد ما يتعطل إذا ما لحق ال write buffer على ال cpu وتبقى فيها لما يرحلوهم لل Memory

Write-Back

→ مستخدمة أكثر الآن
وهو التصميم اللي بريح
ال Memory.

write through ← لا

- Alternative: On data-write hit, just update the block in cache → فقط بنكتب ال block اللي بال Cache
  - Keep track of whether each block is dirty 0=1
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing bloc to be read first ← عشان أحل مشكلة البطء،

منيح بس
المساوئ

Write Allocation

ال data مش موجودة بال Cache

- What should happen on a write miss?
- Alternatives for write-through
  - ① Allocate on miss: fetch the block
  - ② Write around: don't fetch the block (No Allocate)
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

← بس عموما بنستخدم
write
Allocate

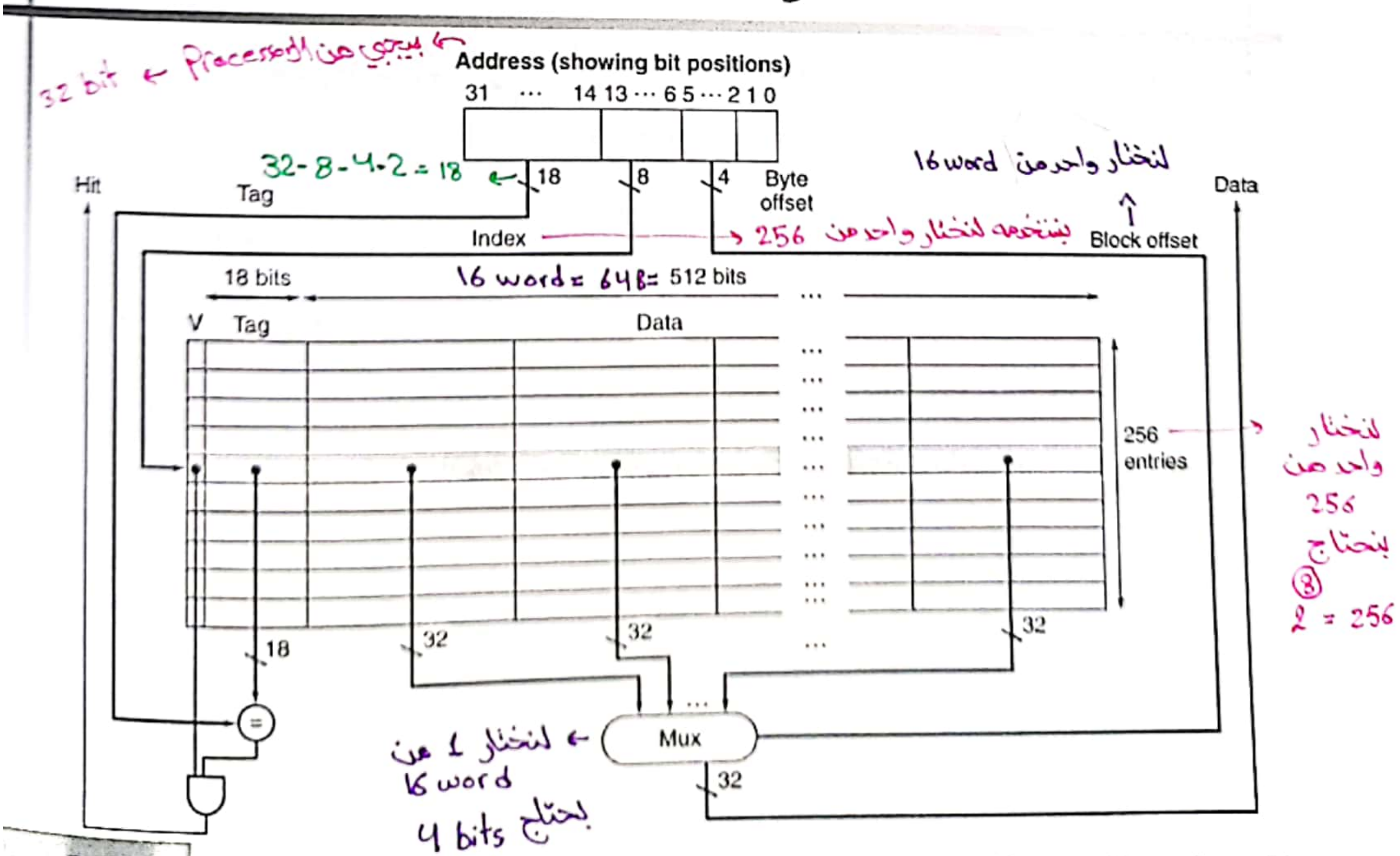
Example: Intrinsity FastMATH

مثال: ال microprocessor الي بالثلاجة أو طابعة أو هيك

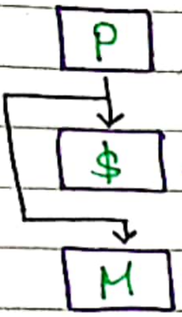
- Embedded MIPS processor
  - 12-stage pipeline في ال F و load/store ال 2 caches لل I و D فبنفس ال cycle بعمل
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache اسم التصميم
  - Each 16KB: 256 blocks × 16 words/block كل واحد فيهم
  - D-cache: write-through or write-back → بختاروه حسب التصميم
- SPEC2000 miss rates
  - I-cache: 0.4% → أكثر hits Instruction cache
  - D-cache: 11.4% Data cache عليه فال data الي بتعامل معها بالعادة أكثر من ال Instruction الي بنفذها ال Processor
  - Weighted average: 3.2% ← بوخذ بعين الاعتبار التكرار

* ال I-cache ال Processor بس بقرأ منه فهو لا write-back ولا write-through لإنه ماني write بالمرة بس read

Example: Intrinsity FastMATH


32 bit ← Processor بيجي من ←
Address (showing bit positions)
31 ··· 14 13 ··· 6 5 ··· 2 1 0
32-8-4-2 = 18
18
8
4
Byte offset
Hit
Tag
Index → 256 بنخدمه لنختار واحد من
لنختار واحد من 16 word
Block offset
Data
18 bits
16 word = 64B = 512 bits
V
Tag
Data
256 entries → لنختار واحد من 256 بنحتاج $2^{8} = 256$
18
32
32
32
=
Mux
32
← لنختار 1 من 16 word بحتاج 4 bits

Slide 47 : Write through :

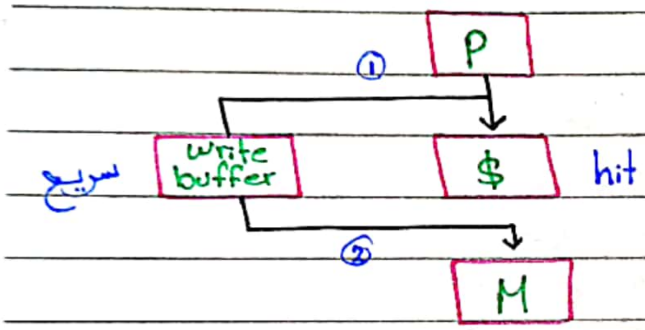


→ لما ال Processor يكتب بيكتب على ال cache وبكتب عال Memory فهيك ما في inconsistent

→ سريع

→ بطيئة

* Write buffer :



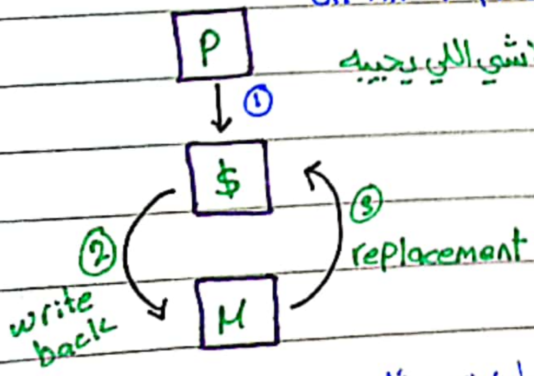
*لما بكتب عال M مباشرة ، أنا بدي أوصل ال data لل Memory وبديش ياها تضيع علي فال data اللي بدي ياها تروح ال Memory ببعت نسخة من اللي حكتبه بال cache لل write buffer وهي عبارة عن Memory سريعة فبعد هيك ال Processor بكمل شغله و ال write buffer على راحته بكتب بال Memory وهاد بحللنا مشكلة ال Performence .

* ال write buffer لازم يكون كبير كفاية إذا ال Processor بعت Many store instructions ودا بعض يتخزنوا بال write buffer ومع الوقت يترحلوا لل Memory .

*بيرهق ال Memory لأنه ال Processor بشتغل كثير وبنفذ Many instructions وكل ال store instructions يااللي به بينفذهم رح يوصلوا لل Memory فحرهق ال Memory بال accesses ، فبنروح ل Write-Back

Slide 48 : write-Back :

on hit → Processor write just on cache (سريع)

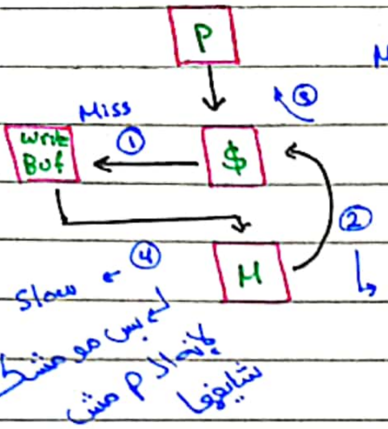


*لما يصير في Miss وبه يعمل replacement : يعني به الاشي اللي يجيبه من ال Mem ليتبدل ل block موجود بال cache وفيه data فعشان ما تضيع ال Data قبل ما أعمل replacment أنقل ال block اللي بال cache وعليه data جديدة للMemory

*بال Miss حتكون بطيئة لإنها بتمر بمرحلتين وال M بطيئة جداً

Slide 48 :

* لأحل مشكلة البطء في ال Write-Back :

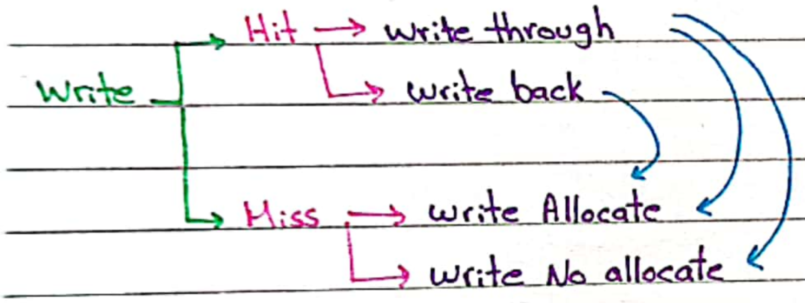


① لو كله hit بضل يقرأ ويكتب من ال cache بسرعة وما بروح لل M

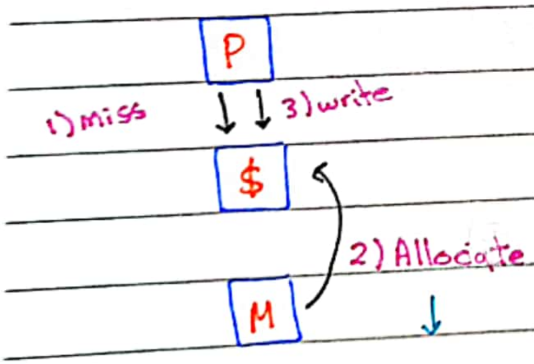
② إذا صار Miss كخطوة أولى بنقل ال block اللي بدنا نعمله replacement لل write buffer من ال cache وهاي عملية سريعة بعدين بنعمل replacement بعدها ال P خلص بياخذ اللي بده ياه . صح بطيئة لكن مو مشكلة بعدين على راحتنا بنعمل write back

Slide 49 : write Allocate:

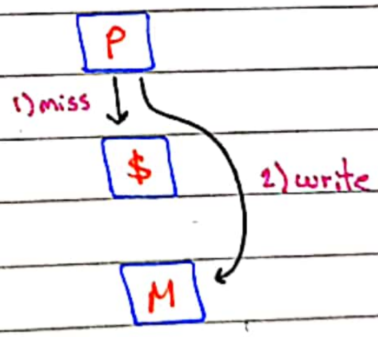
read-miss-Allocate


Write
Hit → write through
↳ write back
Miss → write Allocate
↳ write No allocate

Allocate



No Allocate



لما يصير في Miss بنجيب ال data من ال Mem بعدين بنكتب عليها

لما يصير في Miss ال data مو موجودة بال M والمقر النهائي إلها هو ال Memory فبكتب مباشرة بال Memory

* الأفضل ال Allocate لأنه بخليني أستفيد من ال spatial locality

* للوهلة الأولى بنفكر ال No Allocate لأنه أسرع بالبداية . بس ال Allocate عنه بعد نظر

Contents

$CPI_{total} = CPI_{base} + \frac{Stalls}{Instructions}$

Measuring Cache Performance

- Components of CPU time → يتكون من جزئين
  - ① Program execution cycles → الزمن اللي بيقضوه الـinstructions داخل الـPipeline (F I E W C)
    - Includes cache hit time ← F I A M W C سواء كانت miss أو hit
  - ② Memory stall cycles → بسبب الـMissis
    - Mainly from cache misses → الـMissis
- With simplifying assumptions:

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} \quad ①$$

كل ما كان أعلى الـStall أعلى

Miss rate: $\frac{Missis}{hits + Missis}$ → كل ما كان أعلى كلها الـStall أعلى

Miss penalty: كل ما كانت أعلى كل ما الـStall أعلى

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \quad ②$$

حسب المعطيات بنختار أي معادلة استخدم

ده من بين كل اللي نفذناه قديه في Missis

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles → إذا كان في Miss بأخذ 100 cycle
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

- Hit → Fast time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
    ↓ Slow
- Example
  - $f = \frac{1}{T} = 1$ GHz
  - miss time = hit time + miss penalty = 1 + 20 = 21
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate → الProcessor أسرع
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

← مش بس الProcessor يكون سريع لازم يكون الMemory hierarchy تكون مصممة بشكل صحيح.

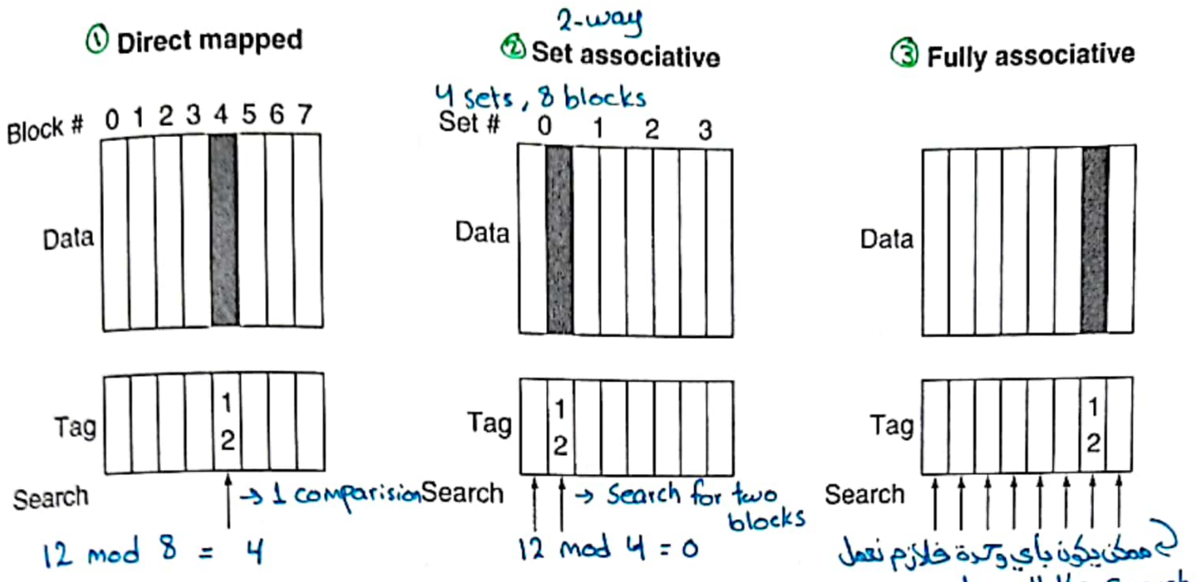
← لتحسين الcache بدل ما يكون Direct Mapped cache

Associative Caches

→ نوع من الcaches متطورات وأقدم من الDirect Mapped cache ولكن بيعطونا high performence

1. Fully associative ← عكس الDirect Mapped cache اللي هو لكل Memory block في مكان واحد إله بالcache بينما الFully Associative cache أي Memory block ممكن نحطه بأي مكان بالcache
   - Allow a given block to go in any cache entry
   - Requires all entries to be searched at once يتطلب
   - Comparator per entry (expensive) → تكلفته أعلى
2. *n*-way set associative ← 2, 4, 8, 16
   - Each set contains *n* entries
   - Block number determines which set
     - (Block number) modulo (#Sets in cache)
   - Search all entries in a given set at once
   - *n* comparators (less expensive) → أقل بسعر من الFully بس أعلى من الDirect

Associative Cache Example

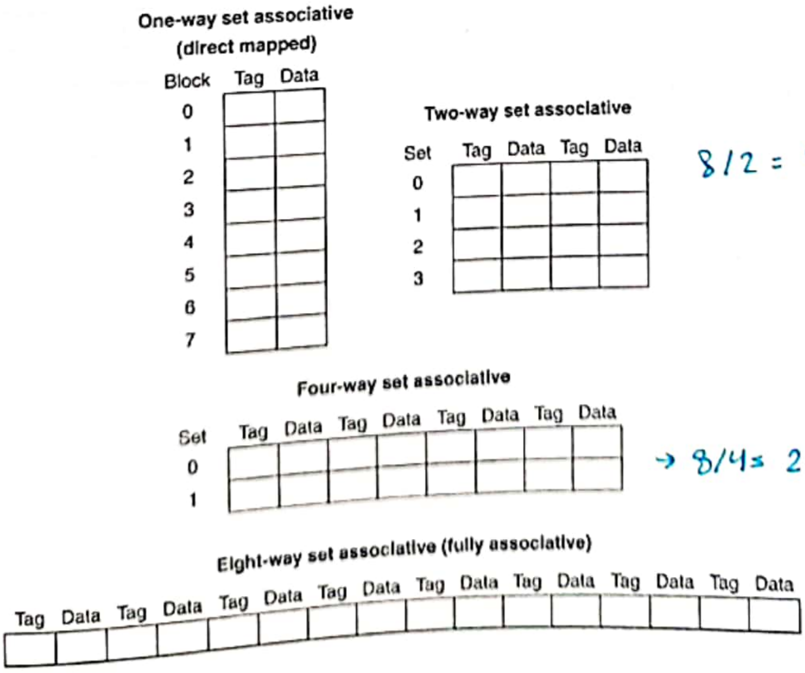


block address Number 12 → وين ندور عليه بعد ولال 3 أنواع من ال Caches .

→ Associativity يعني قديه ممكن نزيد ال

Spectrum of Associativity

- For a cache with 8 entries = 8 blocks



8/2 = 4 sets

→ 8/4 = 2 sets

8/8 = 1 sets = Fully Associative

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8 ↲

cpu ← طلعنهم جاهزات

- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

↓ بطلعه من :

Block Address mod 4

↳ 5 accesses وضعه سيئ لإنه في

5 missis ؛

لتحسين ال Performance

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

↓

Block Address mod 2 بطلعه من

↳ 4 missis

- Fully associative

one set

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

لع كل ال cache
مفتوح إلي فبقدر
استخدم أي
index

↳ 3 missis

How Much Associativity

- Increased associativity decreases miss rate → Performance بالتالي التحسن بصير لل
  - But with diminishing returns
- Simulation of a system with 64KB → حجم ال cache D-cache, 16-word blocks, SPEC2000 → benchmark

↳ 16x4 = 64 bytes

cache blocks = $\frac{64\ KB}{64}$ = 1K = 1024 block

Direct Mapped cache ↤
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

1.7 %

0.3 %

0.2 %

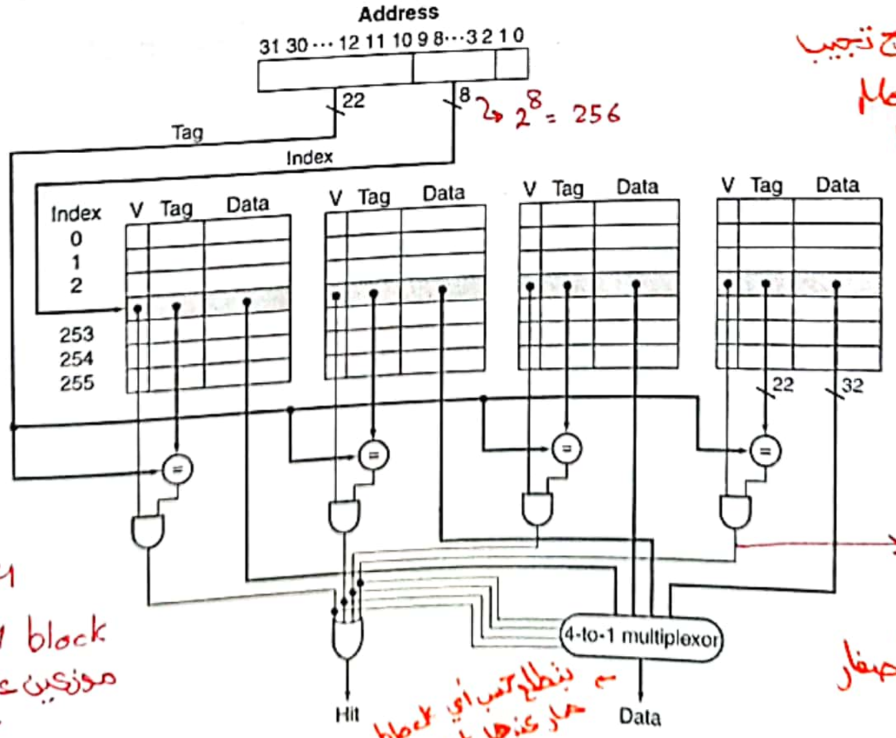
لـ أحسن واحد

Miss Rate

* مش دائمًا بنستفيد لما نروح لـ higher Associativity

1S = 4 blocks → 256 sets / (4-way) → 4 comparators لأنه في

Set Associative Cache Organization



8 ↳ 2^8 = 256

* لو صار Miss فبروح تجيب ال data من الMemory وبتحطها بال block

مستحيل يجبنا 1 على الأربعة بس بكون 1 على واحد منهم أو كلهم أصفار

256 x 4 = 1024 block

256 موزعين على set

← بنطلع حسب أي block صار عنده hit

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- ② Least-recently used (LRU) الأشي اللي من زمان ما استخدمناه واللي بنعملها Replacement
  - Choose the one unused for the longest time
    - hardware Simple for 2-way, manageable for 4-way, too hard beyond that لـ أو أكثر بصير ال hardware أصعب
- ① Random
  - غير مكلفة Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Primary cache attached to CPU = L1
  - Small, but fast
- Level-2 cache services misses from primary cache = L2 = Secondary
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Secondary لـ برضه اسمه

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz → $\frac{1}{f} = 0.25$ ns
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns = 400 cycles
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles → 400 بدل
- CPI = 1 + 0.02 × 20 + 0.005 × 500 = 3.9
- Performance ratio = 9/3.9 = 2.3

1 level of cache ↳ 2 level of cache

Multilevel Cache Considerations

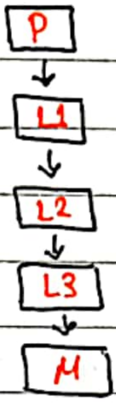
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Miss Rate يكون
قليل حتى مانحتاج ال
Memory

Slide 55 : AMAT :

Find the average Memory Access time of a Memory hierarchy with the following specifications.

	Memory level	Hit Time	Miss Rate
	L1 cache	1 cycle	6.0%
	L2 cache	10 cycles	4.0%
	L3 cache	20 cycles	2.0%
	Main Memory	250 cycles	0 %



AMAT = Hit time + Miss rate x Miss Penalty

مو معطاه بشكل صحيح بس ال Main Memory لـ
ما خليها Miss فال average = 250 فلو
بدنا نروح من L3 لل Memory رح نحتاج 250
فال Miss penalty لـ L3 = 250 ، فحل السؤال حيكون من الأسفل إلى الأعلى .

$AMAT_{L3} = 20 + 0.02 \times 250 = 25$ cycles

$AMAT_{L2} = 10 + 0.04 \times 25 = 11$ cycles

$AMAT_{L1} = 1 + 0.06 \times 11 = 1.66$ cycles

$\therefore$ AMAT = 1.66 → processor اللي بشوفه ال

Slide 64 : LRU Replacement Policy :

4-way associative cache, initially empty

hit hit

Address	1	2	3	4	1	5	3	6
MRU	1	2	3	4	1	5	3	6
		1	2	3	4	1	5	3
			1	2	3	4	1	5
LRU				1	2	3	4	1

← Information maintained for LRU replacement for each set

لـ يعني بحتاج 4 register احتفظ فيهم مين
ال MRU ومين ال LRU

* MRU = Most recently Used
* LRU = Least Recently Used

$

Interactions with Advanced CPUs

dynamic execution

- Out-of-order CPUs can execute instructions during cache miss →

  ld
  add] هدول صار فيهم stall
  sub → هاي ما بتعتمد عليهم فبنفذوها ما بستنى

  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow → يعني ممكن ال miss تعطل ال instructions اللي بعتمدوا عليها
  - Much harder to analyse
  - Use system simulation → الاعتماد عليه بالوقت الحاضر ليعرف سريع ولا لأ لأنه بالمعادلات صعب نطلع السرعة

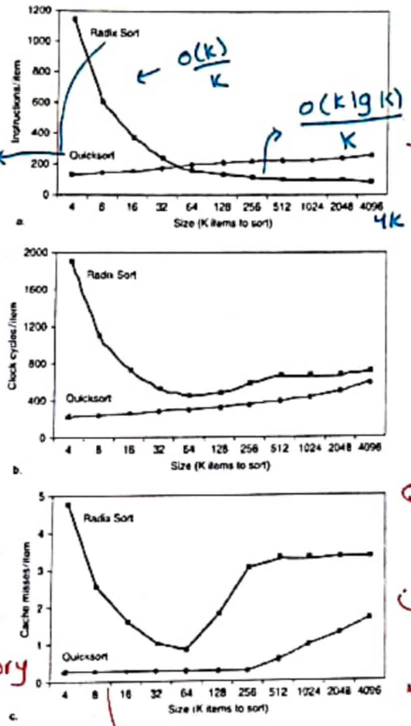
نتيجته

$

Interactions with Software

2 Algorthims مختلفة

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



برتبوا الداتا

4K → $\lg_2 4K = 12$

For large data ال Radix تبين أفضل ولكن لما نحسب ال Clockcycle بطلع ال Quick Sort أحسن وفعلاً هو الأحسن رغم إنه بحتاج عدد Instructions أكثر

← ال Radix إله كثير cache missis فيستخدم large Memory عشان هيك بطيء

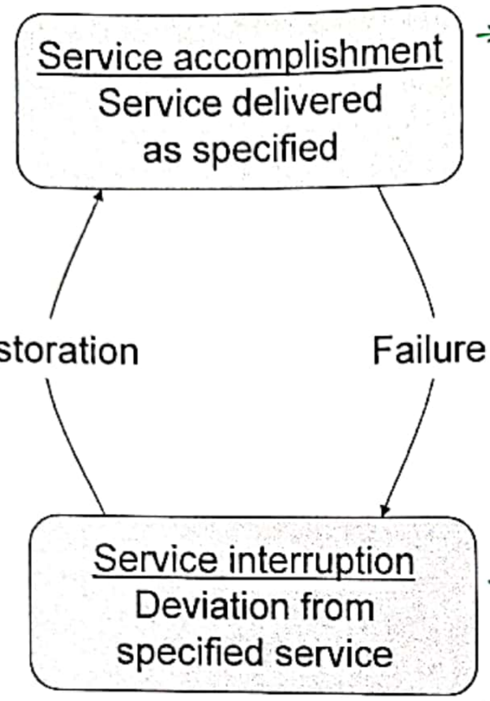
← بستخدم حجم أقل من ال Mem واله cache missis أقل

ال Radix بده Memory large وبعبي فيه ال data اللي بنيجي بعدين لما يجي بده يقراها بقراها بشكل متعب من مكان لمكان ولأنه ال Cache ما بوسع كل هالداتا بصير عنده alot of cache Missis بينما ال Quick فيه شغل على منطقة منطقة at a time ففيه عنده bette Memory access

Contents

يعني اللي بنحطه بال MeM نرد نقدر نقراه

Dependability → نفكر هل السستم شغال ولا خربان


Service accomplishment
Service delivered as specified
Restoration
Failure
Service interruption
Deviation from specified service

✓ OK

→ يعني بيشغل المطلوب بشكل صحيح

لما نصلح العطل اللي كان ←

X

→ في مشكلة

- Fault: failure of a component → hardware → Software
  - May or may not lead to system failure

→ يعني هذا السستم اللي بيخدمنا يصير فيه
عطل معين سواء بال Software
أو ال hard ware

ممكن لو صار fault مايؤدي ل Failure
مثل لو صار تعطل بالكيبورد فهو مادخله أو لو تعطل harddisck ما بنستخدمه
أو لو كان في redundanay في بديل لاشي لو عطل فما بصير لا system
Failure

Dependability Measures

- Reliability: mean time to failure (MTTF) ← معتمد عليها
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR) ← متاح

أحسن availability
تكون واحد ، 100% ←
بس أحسن سستم
بكون إله
99.999...%

- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

The Hamming SEC Code → تصحيح الأخطاء

- Hamming distance → بقرأمثل الأرقام
  - Number of bits that are different between two bit patterns ← نقل شفرات معينة

0 0 0 0
1 0 0 1] → different = 1 bit
2 0 1 0] → // = 2 bit

بس الـ Minimum = 1

- Minimum distance = 2 provides single bit error detection → بقدر أمثل الأرقام وأكتشف الخطأ
  - E.g. parity code
- Minimum distance = 3 provides single error correction, 2 bit error detection

بقدر أمثل الأرقام واكتشف الخطأ وأحله

Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power 2 are parity bits
  - Each parity bit checks certain data bits:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
odd الـ →	p1	X		X		X		X		X		X	
Parity bit coverate →	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

بيعدوا 2 [illegible]

Decoding SEC

→ Single Error correction

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - E.g.
    - Parity bits = 0000 indicates no error
    - Parity bits = 1010 indicates bit 10 was flipped
- <u>Example</u>:
  - Parity What will be stored for $\underbrace{100\overset{9}{1}}\ \underbrace{1\overset{10}{0}10}$?
  - If you read 0111 0010 1110, is there error? Correct it.

SEC/DED Code

→ Single Error correction/ Double Error detection

- Add an additional parity bit for the whole word (p_n) → اضافة إلى ال Parity Bit
- Make Hamming distance = 4
- Decoding:
  - Let H = SEC parity bits ← $P_1P_2P_4P_8$
    - H = 0, p_n even, no error
    - H ≠ 0, p_n odd, correctable single bit error
    - H = 0, p_n odd, error in p_n bit
    - H ≠ 0, p_n even, double error occurred
- <u>ECC DRAM</u> uses SEC/DED with 8 bits protecting each 64 bits

↳ Error correction code

بفيد انه خالية

ال Errors بكونوا Single وبقدر أصلحهم بس في مرات بكونوا double error

بقدر اكتشفهم بس ما بقدر أصلحه

فبعطي رسالة للمستخدم إنه في error

correctable بس 1 error ←
لإنه H باشر لمكانه

صار ال error عال P ما بهمنا ←
بهمنا ال data تكون صحيحة

ما بنقدر نصلح
ال double
بس بنكتشفهم

8 Parity Bit
7 = H
1 = P_n

Contents

RAID

- Redundant Array of Inexpensive (Independent) Disks رخيص
  - Use multiple smaller disks (c.f. one large disk)
  - Parallelism improves performance عشان ال data تكون موجودة بأكثر من disck
  - Plus extra disk(s) for redundant data storage
- Provides fault tolerant storage system → تتحمل الأخطاء وما بتأثر فيها
  - Especially if failed disks can be "hot swapped"

وهو ال Server شغال بتقدر إنك تشيل ال component المعطلة وتحط واحد جديد

- ① RAID 0
  - No redundancy ("AID"?)
    - Just stripe data over multiple disks
  - But it does improve performance

②　③

RAID 1 & 2

- RAID 1: Mirroring
  - N + N disks, replicate data
    - Write data to both data disk and mirror disk
    - On disk failure, read from mirror
- RAID 2: Error correcting code (ECC)
  - N + E disks (e.g., 10 + 4)
  - Split data at bit level across N disks
  - Generate E-bit ECC → $P_1\ P_2\ P_4\ P_8$
  - Too complex, not used in practice

④

RAID 3: Bit-Interleaved Parity

- N + 1 disks
  - Data striped across N disks at byte level
  - Redundant disk stores parity
  - Read access → الداتا مجزئة فلازم
    - Read all disks　نقرا من كل ال disks
  - Write access
    - Generate new parity and update all disks
  - On failure → لو تعطل واحد بجيبه من ال Parity
    - Use parity to reconstruct missing data　لأنه الأرقام
- Not widely used

5

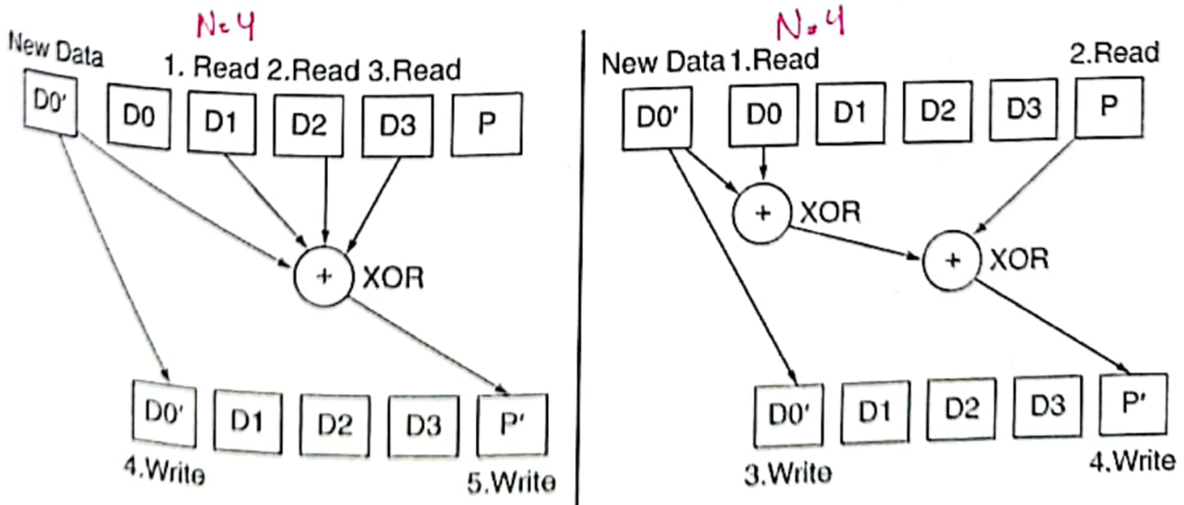
RAID 4: Block-Interleaved Parity

- N + 1 disks
  - 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
  - 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

* الأوفر RAID 3 / RAID4 من ال RAID2 لإنهم أرخص

RAID 3 vs RAID 4

→ مستخدم أكثر من RAID3

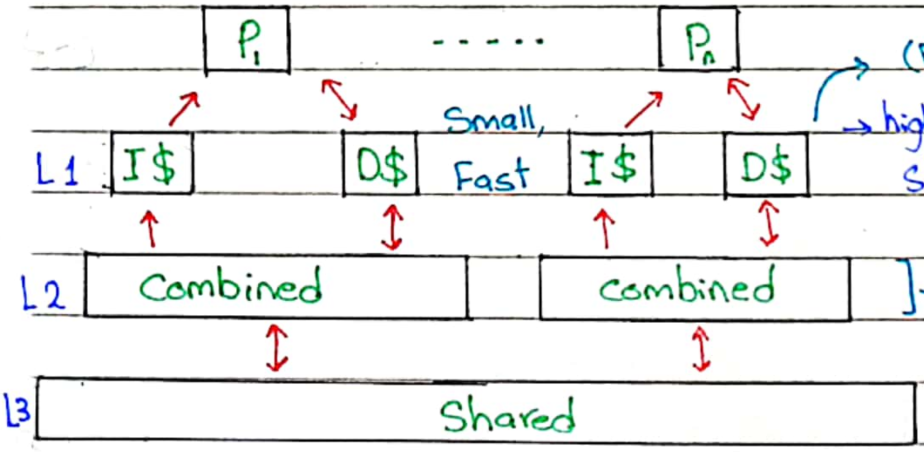


N-1 reads
2 writes

2 reads
2 writes

Slide 68 : Multilevel Caches

* Microprocessor with multiple cores.



(Private) → high miss rate لإنهم سريعين بكون ال

فبكونوا مدعمين ب Secondry cache

(private) → large , slower

بخدم ال I$ وال D$

سوا لهيك هو combined

بفيدنا لإنهم أكبر إنه بقلل ال Miss rate فبزيد ال Performance

أكبر وأبطأ من L2 و Shared تعني كل ال Processors

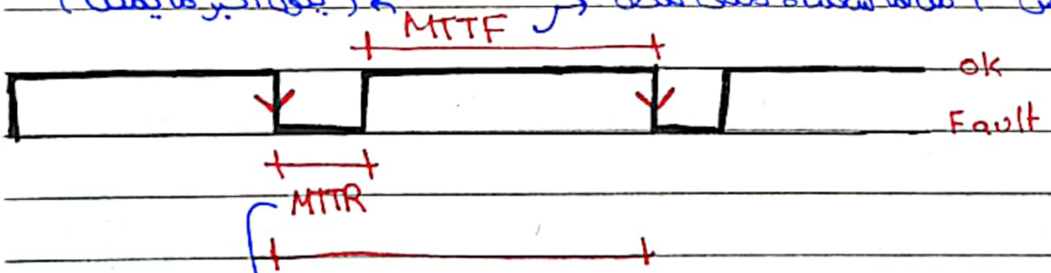
بستخدموه مش Processor واحد (Shared) وهاد اشي مفيد لما ال Processor

يتعاملوا مع نفس ال data وال code فهيك بقلل ال "Miss Rate" وبزيد ال "hit Rate"

SRAM

Slide 73 : Dependability Measures :

(بالساعات أو الأيام يقاس) من لما شغلناه لحتى تعطل ← MTTF → (بكون أكبر ما يمكن)



MTTR → زمن التصليح (كل ما كان أصغر كل ما كان أفضل)

MTBF → مجموع الزمنين

PAGE

DATE

Slide 73 : Improving Availabilty.

① Increase MTTF:

- Fault Avoidance: من البداية أتجنب الخطأ بإنه الـ software و الـ hardware يكونوا أفضل ما يمكن ومن الصعب أحصل عليها لإنها غالية جدًا بتكون.

- Fault tolerance: يعني لو تعطل عندي اشي يكون في إله بديل عشان ما نوقف الشغل مثل إنه يكون فيه مروحتين بتبريد مثلاً.

- Fault Forecasting: أتنبأ بالأعطال اللي ممكن تصير ونعمل على تبديل القطعة لما أعرف إنه حيخلص عمرهم أو حيخرب.

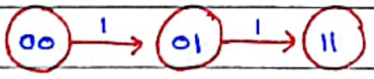
② Reduce MTTR: أحسن الـ tools عشان تنبهني بالأخطاء وتشخصها وبساعدني أكتشف العطل و الآن لنقلل زمن التصليح: 1- يكون عندي مكان أدخل عليه العطل أحله بدون ما أأثر عالباقي 2- نزيد أعداد المصلحين 3- نستعمل harddisck ثاني فلازم يكون عندي قطع غيار من أول احتياطًا.

Slide 74: Hamming Distance :

Distance = 1

Example Binary

0	00
1	01
2	10
3	11



Parity

Ex: 011 → 0 ✓

010 → 1 X

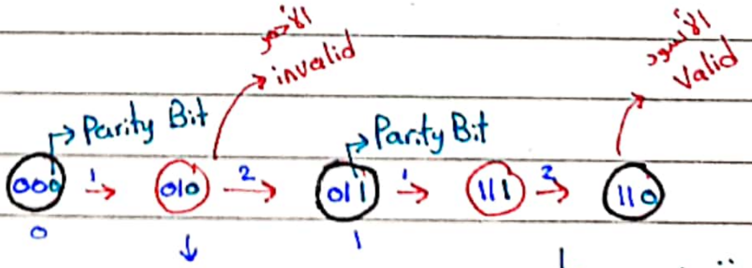
Distance = 2

Example Binary with Parity

0	00	0
1	01	1
2	10	1
3	11	0

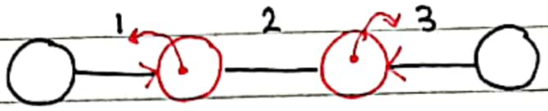
even ↲

لو عدد الـ 1 زوجي بنحط 0 ولو فردي بنحط 1



وهون نفس المبدأ ← من 0 لـ 1 بنحتاج نعمل تغييرين هي أول تغير وثاني تغير تغير الـ Parity من 0 ← 1

Distance = 3



ب distance = 2 ← لو اختار وحدة من الحمر حيتحير لأي سودة يروح بس

ب distance = 3 ← لو اختار وحدة من الحمر بختار السودة الأقرب إله

Slide 76 : Encoding SEC :

Example

Data

1001 1010

Bit position		1	2	3	4	5	6	7	8	9	10	11	11
Encoded data bits		P1	P2	d1	P4	d2	d3	d4	P8	d5	d6	d7	d8
Parity bit Coverate	P1	X		X		X		X		X		X	
	P2		X	X			X	X			X	X	
	P4				X	X	X	X					X
	P8								X	X	X	X	X

0 1 1 1 0 0 1 0 1 0 1 0

From cpu to Memory عدد الـ 1 زوجي دا ← هذا اللي بنخزن هنا

From Memory to cpu 0111 0010 1110 P8 P4 P2 P1

Single Error فعندي في اختلاف بين اللي حليناه ← 1 0 1 0

بين الأزرق والأحمر

← لما حسبت الـ parity بين إنه في error لأنه مو 0 والرقم اللي طلعه = 10

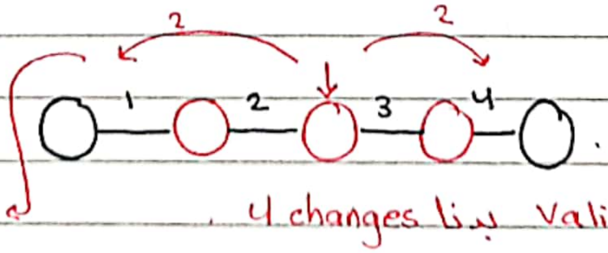
وهو المكان اللي حصل فيه الخطأ والحل إني أقلب قيمته للقيمة الصحيحة.

⇒ 0111 0010 1110

0111 0110 1010

Slide 77 :

Distance = 4



← ما بنعرف وين نروح المسافة متساوية فبسمح نعمل SEC / DED

من Valid code لـ Valid code بدنا 4 changes .

PAGE
DATE

Slide 79 8 RAID

RAID 0

Data

Disck

A1	A2
A3	A4
A5	A6
A7	A8

لوهاد تعطل فبتكون
ضاعت كل ال Data
اللي عليه

RAID 1

A1	A1
A2	A2
A3	A3
A4	A4

* لازم يكون عدد زوجي من ال disck
* لو تعطل واحد بقرأ من ال Mirror تاعه
* اللي تعطل بستبدله و بنقل عليه ال data

RAID 2

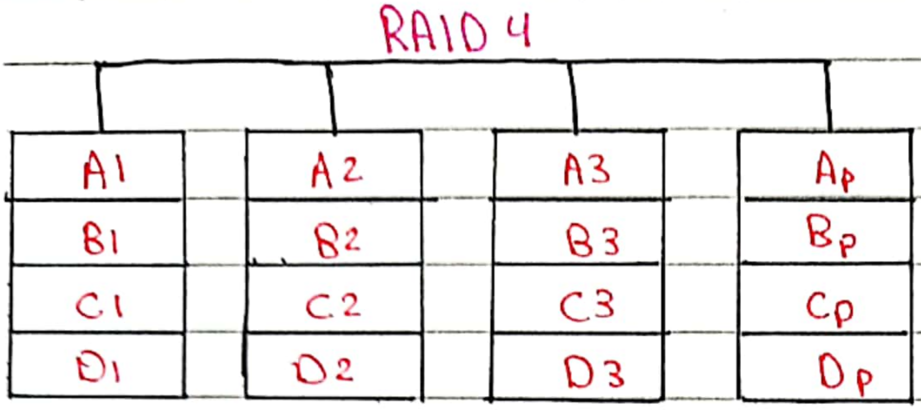
A1	A2	A3	A4	AP_1	AP_2	AP_3
B1	B2	B3	B4	BP_1	BP_2	BP_3
C1	C2	C3	C4	CP_1	CP_2	CP_3
D1	D2	D3	D4	DP_1	DP_2	DP_3
Disck0	Disck1	Disck2	Disck3	Disck4	Disck5	Disck6

* غالي وصعب ال hardware

RAID 3

A1	A2	A3	Ap(1-3)
A4	A5	A6	Ap(4-6)
B1	B2	B3	Bp(1-3)
B4	B5	B6	Bp(4-6)

N = 3



A1	A2	A3	Ap
B1	B2	B3	Bp
C1	C2	C3	Cp
D1	D2	D3	Dp

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

إذا في تغير رح نعكس الParity

و بنستخدم ناتج ال Xor مع الParity القديمة ونعملهم Xor بطلعوا الParity الجديدة.

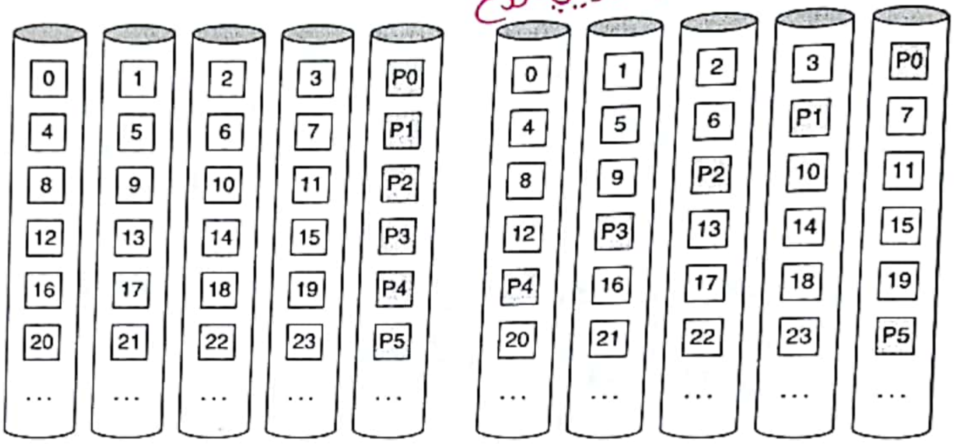
6

(أفضل واحد في الوقت الحاضر)

RAID 5: Distributed Parity

- N + 1 disks
  - Like RAID 4, but parity blocks distributed across disks
    - Avoids parity disk being a bottleneck
- Widely used

صاروا كلهم زي بعض والباريتي موزع



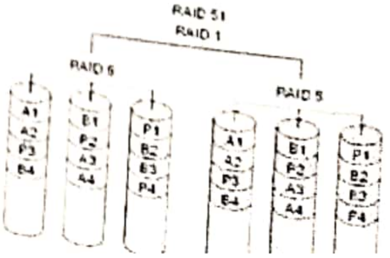
RAID 4 RAID 5

7

RAID 6: P + Q Redundancy

عشان لو بالصدفة تعطل 2 discks ماتروح كلينا وبكون قادر يرجع الداتا

- N + 2 disks
  - Like RAID 5, but two lots of parity
  - Greater fault tolerance through more redundancy
- Multiple RAID → للمؤسسات الكبيرة اللي إلها أكثر من أفرع
  - More advanced systems give similar fault tolerance with better performance
  - Example RAID 51



RAID Summary

- RAID can improve performance and availability لأنه بدال ما يكون عندك Disck واحد بكون أكثر من Disck
  - High availability requires hot swapping → إذا تعطل اشي ما بوقف الجهاز وبيبدل المعطل بجديد
- Assumes independent disk failures → إذا تعطل واحد البقية ما بتعطلوا
  - Too bad if the building burns down!

→ إذا احترقت البناية راح كل ال RAID والحل تروح لل Multiple RAID مثل RAID 51

Contents

5.1 Introduction
5.2 Memory Technologies
5.3 The Basics of Caches
5.4 Measuring and Improving Cache Performance
5.5 Dependable Memory Hierarchy
5.11 Redundant Arrays of Inexpensive Disks
5.6 Virtual Machines → إلها علاقة بال Cloud
5.7 Virtual Memory
5.8 A Common Framework for Memory Hierarchy
5.9 Using a Finite-State Machine to Control a Simple Cache
5.10 Cache Coherence
5.13 The ARM Cortex-A53 and Intel Core i7 Memory Hierarchies
5.16 Fallacies and Pitfalls
5.17 Concluding Remarks

بستخدم ← مثلاً لو بدي أشغل برنامج الـ operating system المختلف عاللي عندي

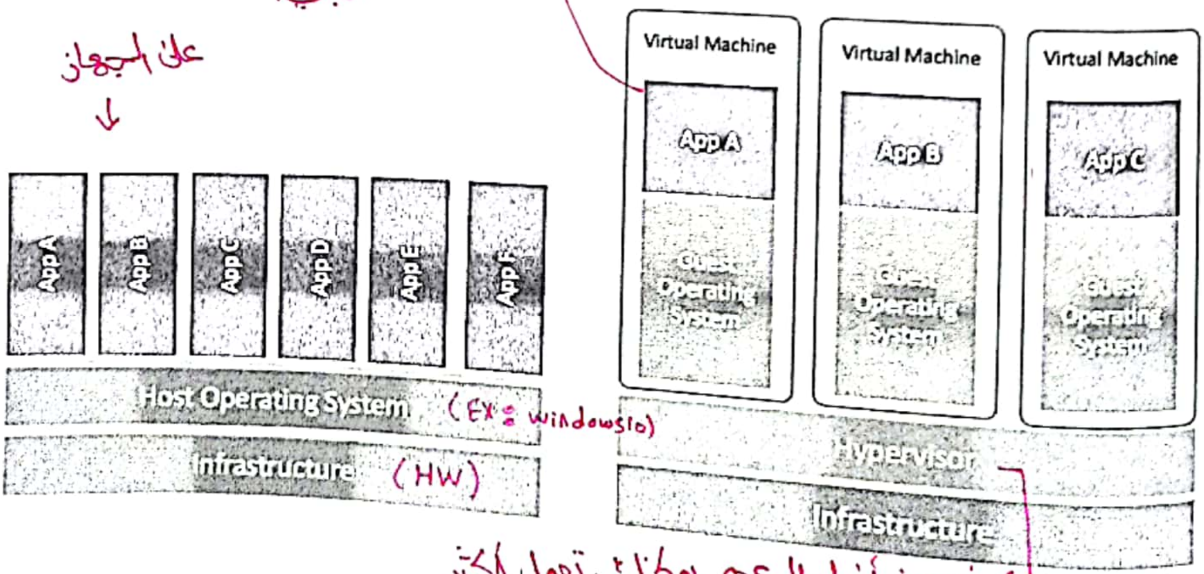
Virtual Machines

- Host computer emulates guest operating system and machine resources بيقلد أو بحاكي
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources → بنظم كل App قديه ياخذ من الـ resources
- Virtualization has some performance impact
  - ممكن Feasible with modern high-performance comptuers
- Examples
  - ① IBM VM/370 (1970s technology!)
  - ② VMWare من شركة oracle
  - ③ Microsoft Virtual PC

ما بنشغل كل الـ computer بس الـ cores اللي حطت عليهم

كل VM ← ما بتشوف إلا ملفاتها ولو تعطلت ما بتخرب غيرها

Virtual Machines



cloud ↓

لو خرب واحد من الـ App ما بتأثروا الباقي.

على الجهاز ↓

(Ex: windows10)

(HW)

نوع من أنواع الـ OS بيمكنك تعمل أكثر من VM

Virtual Machine Monitor

- Maps virtual resources to physical resources Guest
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode → بيشتغل بال 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

Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode الي يتعاملوا مع الهاردوير ويعملو دا input/output
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers

Contents

…

5.5 Dependable Memory Hierarchy

5.11 Redundant Arrays of Inexpensive Disks

5.6 Virtual Machines

5.7 Virtual Memory

Introduction

باستخدامها بصير ← Page Tables

الـ access بطيئ فبنلجأ لـ → Fast Translation Using a TLB

Memory Protection → ما في Process بيقدر تتدخل بـ code أو data لـ Process تاني وهذا فائدة الـ Virtual Memory

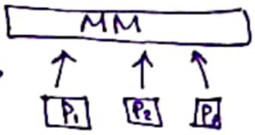
Virtual Memory

زاكرة افتراضية

→ (موجودة بكل أنظمة الحواسيب) يعني البرامج بتفترض اشي والحقيقة شي آخر

- Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

    ← يعني الـhardware للتالي بنقل الـ Data من الـ Memory للـ Cache لكن نقل الـ Data بين الـ Memory والـ disk بتم بتعاون software + hardware
- Programs share main memory

  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page
  - VM translation "miss" is called a page fault

Sharing the Physical Memory

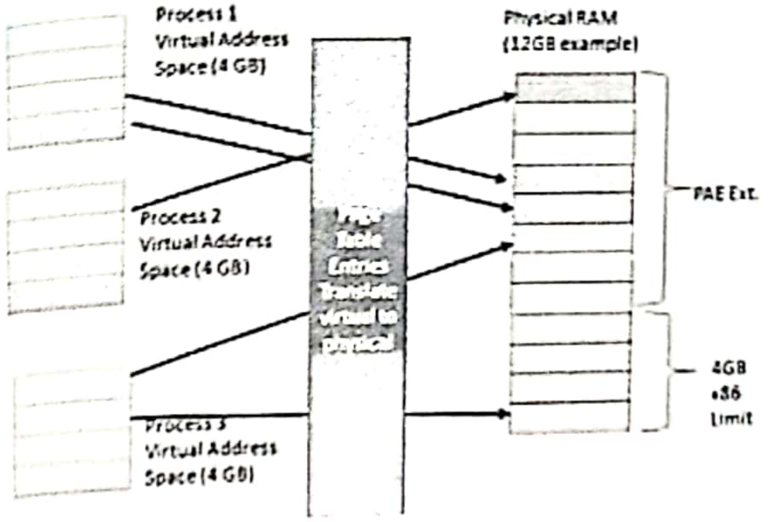
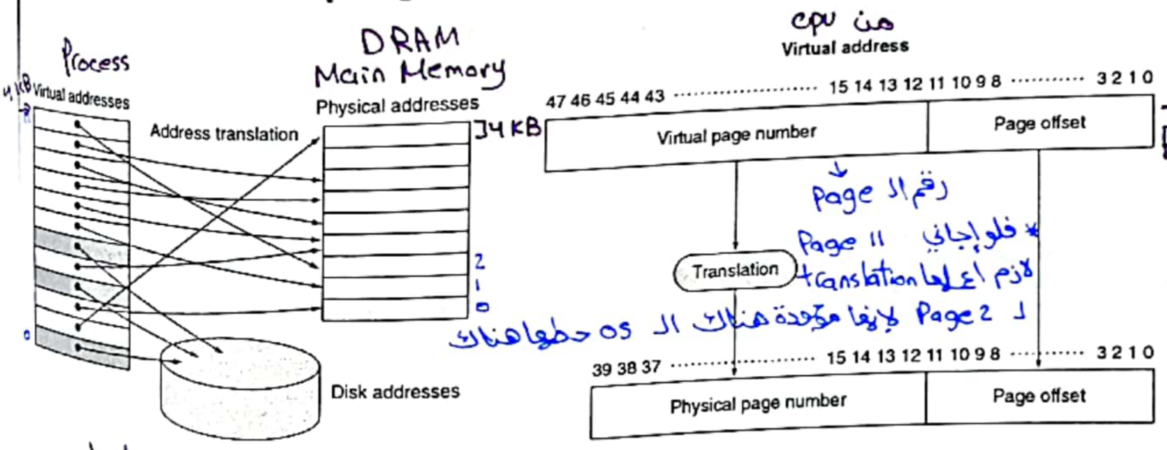


Figure 3

Address Translation

- Fixed-size pages (e.g., 4K)



رقم ال Page

* فلوإجاني ال Page لازم اعملها translation
لـ Page 2 لإنها موجودة هناك ال OS حطها هناك

بنمكني →
اختار ال Byte
اللي بدي ياها
من ال 4KB
اللي موجودين
بال Page
ولإنهم نفس
الترتيب
فال Page
offset مافي داعي هعملها
translation

* ال Virtual Addresses ممكن يكونوا
أكبر من ال Physical Addresses
فمش كلهم إلهم مكان بال MM
فبكون في أشياء بال Disk

هو اللي بروح لل Mem
وبنستخدمه لل access
عال Mem ولو ماكانت
ال Page عال Mem فبصير Page fault

Page Fault Penalty

Main Memory Access time = 30 n sec
Disk Access time = 5 m sec

← أبطأ منها بعشرة أضعاف من الـ M فلو الشغلة ما لقيناها بالـ Memory

وصار عنا Page fault واضطرينا نروح لـ Disk فجيبكون عنا Penalty

- On page fault, the page must be fetched from disk → بديل الـ Disk هالي الأيام ؛ معمول من الـ Flash → SSD
  - Takes millions of clock cycles → لإنه يقاس بالـ M sec
  - Handled by OS code → عليه معقدة الـ hardware ما بقدر يعملها الـ OS هو اللي بعملها لما يصير في Page fault بيدخل وبوف المشكلة وبجيب الـ Data اللي مش موجودة بالـ MM
- Try to minimize page fault rate → الـ OS مكتوبة بإنها تقلل قد ما تقدر الـ Page fault
  1. Fully associative placement
  2. Smart replacement algorithms

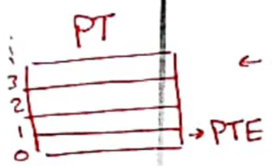
يعني إذا جبت اشي من الـ disk بحطه بأي مكان بدي يله يكون فاضي

مثلاً Least Recently used / Random
↓
أحسن بس أعقد
↳ معقد عالـ hardware بس عالـ Software بنقدر نكتب .

Page Tables

→ الـ OS حتى يقدر يعمل translation بكون عنده Page tables

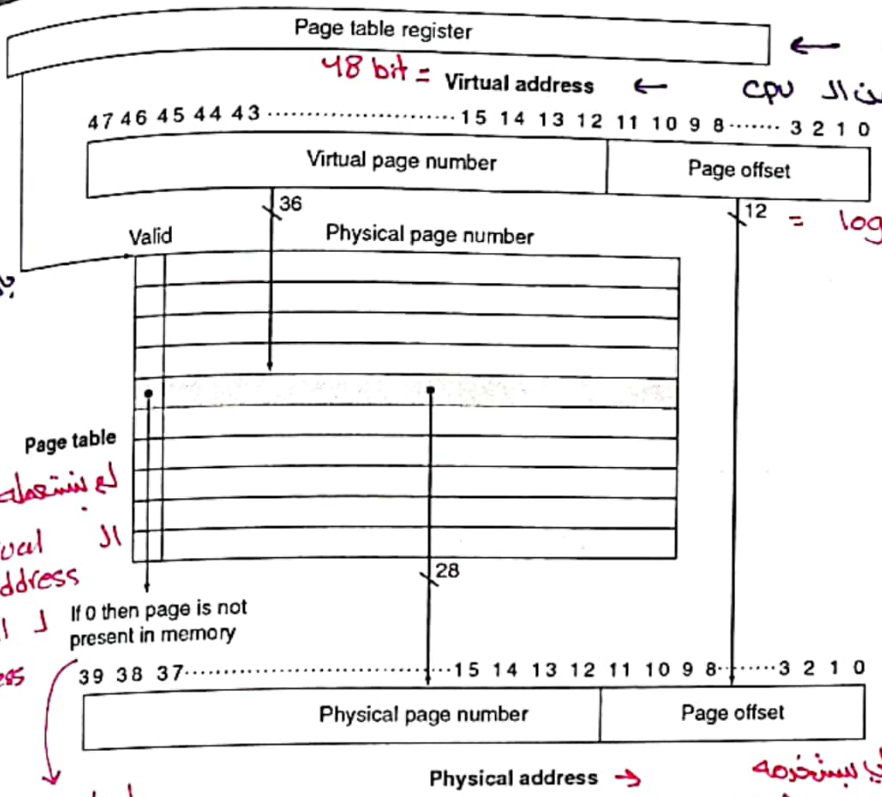
- Stores placement information → يعني بخزن كل Virtual page وين موجودة بالـ Main M
  - Array of page table entries, indexed by virtual page number → عن طريق الـ VPN بنوصل لـ Array و بنطلع لـ PPN (physical Page Number)
  - Page table register in CPU points to page table in physical memory ← لكل process



الـ OS يعملها PT ولما 1 process تكون شغالة بنستخدم الـ PT تاعها فالـ PTR بأشر كبداية الـ PT لـ Process اللي شغال

- If page is present in memory (Page hit)
  - PTE stores the physical page number PPN
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

Translation Using a Page Table



← cpu

48 bit = Virtual address ← ببيجي من ال cpu

= log 4 KB = 12

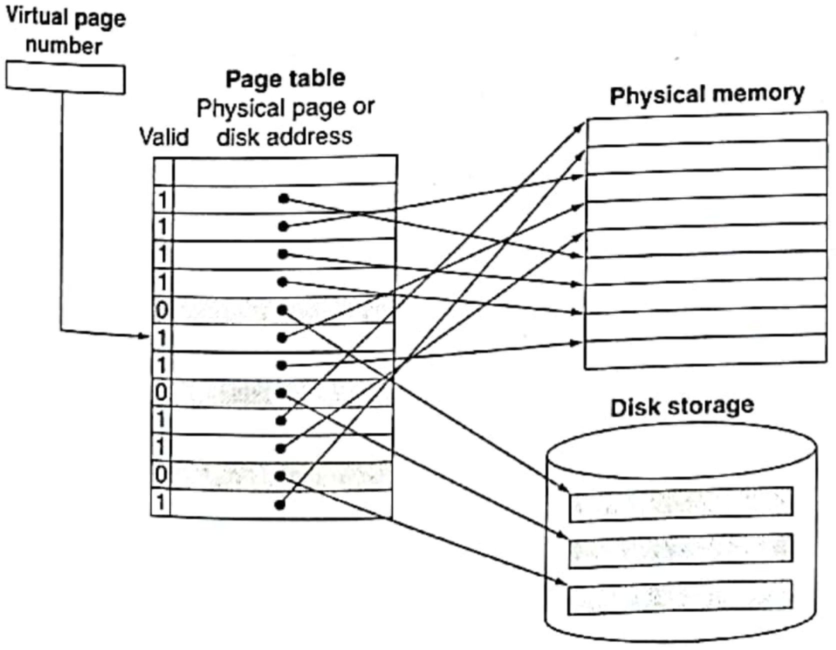
بأشر كبداية ال رام

لم بنستعمله لنحول ال virtual Address ل physical address

لو كانت ١ بكون Page fault وبكون موجود بال disk

Physical address → الي بستخدمه ل Access ال Mem

Mapping Pages to Storage



Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS ← كل كم ثانية الـ OS بصفر الـ use bit
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations ← بنكتب بالـ Mem ما بنحطها عالـ disk
  - Write through is impractical ← مش عملي
  - Use write-back ← هو العملي
  - Dirty bit in PTE set when page is written

← طرق تسرع الـ translation

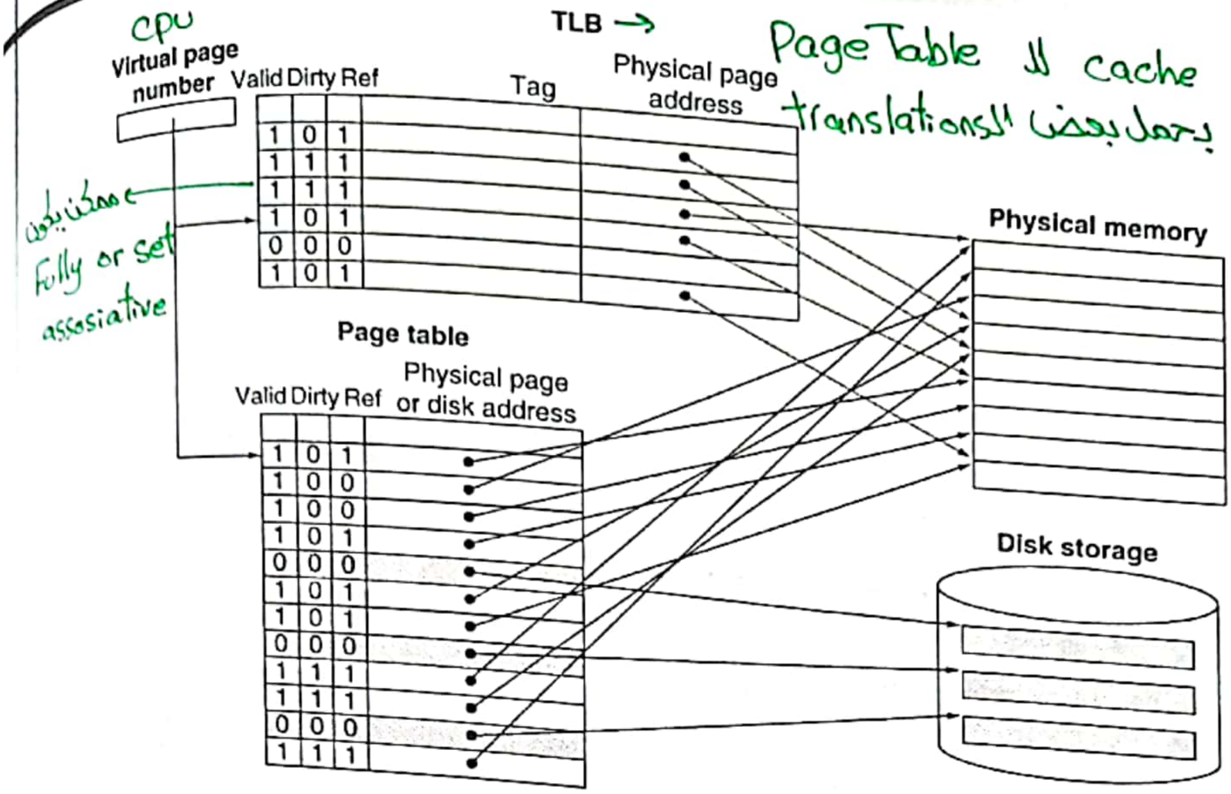
Fast Translation Using a TLB

- Address translation would appear to require extra memory references VA → PT → PA → M
  - One to access the PTE
  - Then the actual memory access → أبطأ جدًا من الـ Process
- But access to page tables has good locality ← الـ Page الوحدة بنستخدمها مرات عديدة
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB) ← داخل الـ TLB
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate ← عشان في good locality
  - Misses could be handled by hardware or software

← الـ cpu بتروح لـ TLB وهو مافيه كل شي في جزء منا الـ translations ولو ما لقيت جزء بلك تروح لـ PT وحتى تروح PT وتجيب المطلوب وتحطه بالـ TLB

بعض الـ System بنستخدم hardware أو Software

Fast Translation Using a TLB



Valid	Dirty	Ref
1	0	1
1	1	1
1	1	1
1	0	1
0	0	0
1	0	1

Valid	Dirty	Ref
1	0	1
1	0	0
1	0	0
1	0	1
0	0	0
1	0	1
1	0	1
0	0	0
1	1	1
1	1	1
0	0	0
1	1	1

TLB Misses

- ① If page is in memory → Memory الPage موجودة بال
  - Load the PTE from memory and retry PT TLB بس مش موجودة بال
  - Could be handled in hardware → يعتمد فيه الPT Simple فنستخدم الhardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler ← عدد الinstructions محسوبة بدقة
- ② If page is not in memory (page fault) → بكلم الsystem بتصرف الOS
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

TLB Miss Handler

- TLB miss indicates
  - ① Page present, but PTE not in TLB هاي أبسط
  - ② Page not preset (Page fault)
- Must recognize TLB miss before destination register overwritten → مثلاً لو عندي ld x1, لو عملت TLB Miss مش لازم نخليها تكمل ولا تكتب ع x1 لإنه ال translation ما صار لسا.
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

لما ال Software يتصرف

Page Fault Handler → جزء من ال OS بشتغل لما يصير عنا page fault

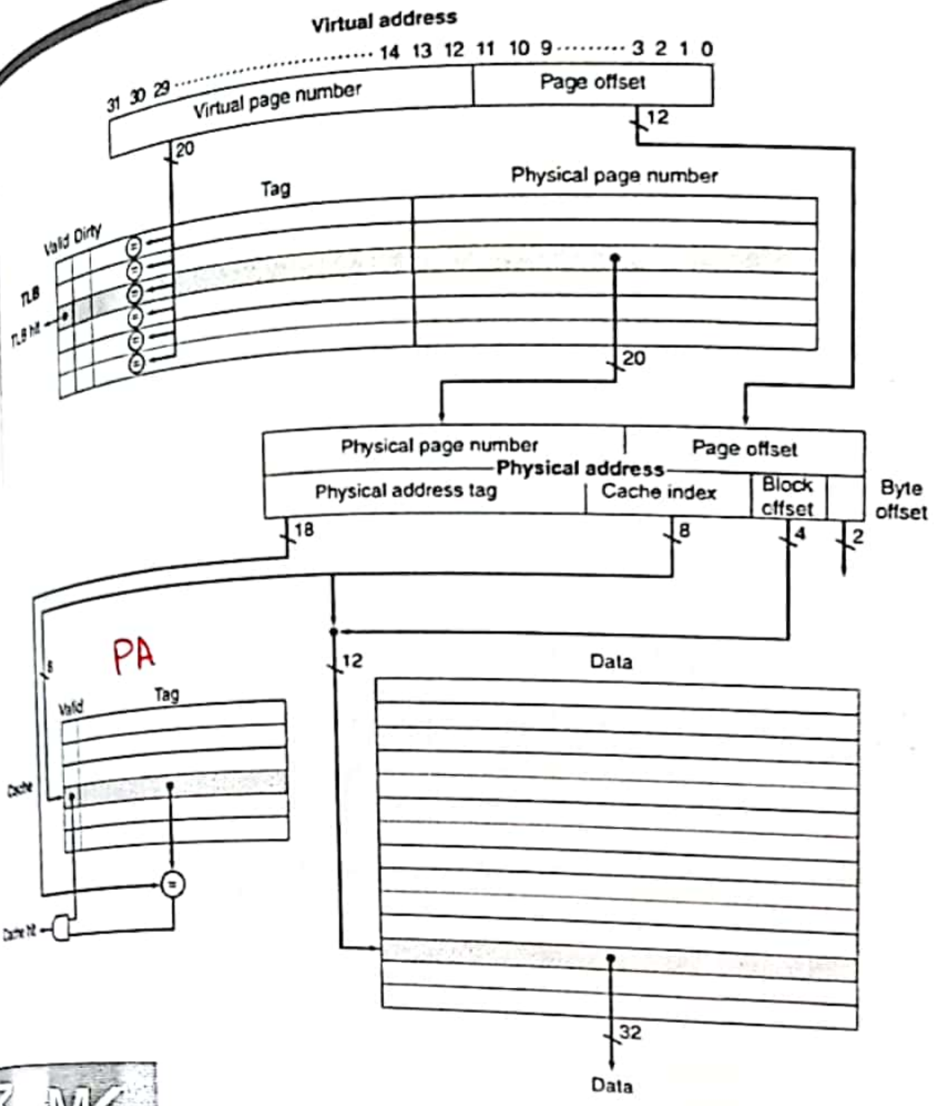
- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

Ex: ld TLB hit ✓

Miss PT Valid ✓

Page fault

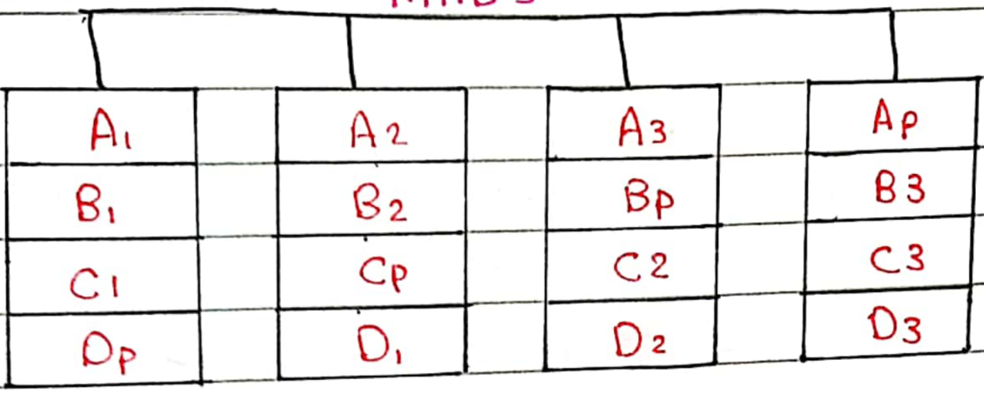
TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag أسرع
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

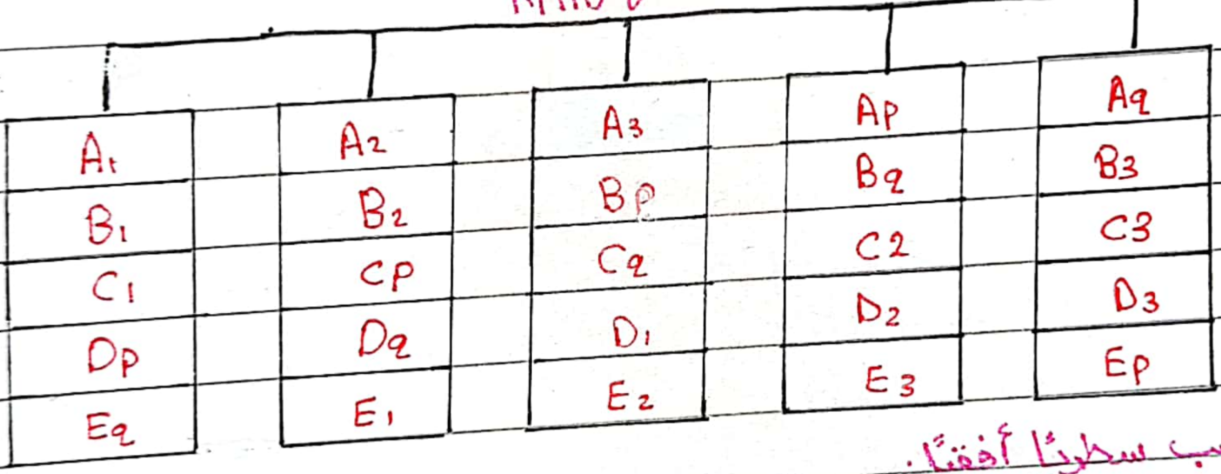
لـ يعني ممكن يكون عندك
2P لـ Shared وهي one page
يطلعولها different Addresses

RAID5



A_1	A_2	A_3	A_p
B_1	B_2	B_p	B_3
C_1	C_p	C_2	C_3
D_p	D_1	D_2	D_3

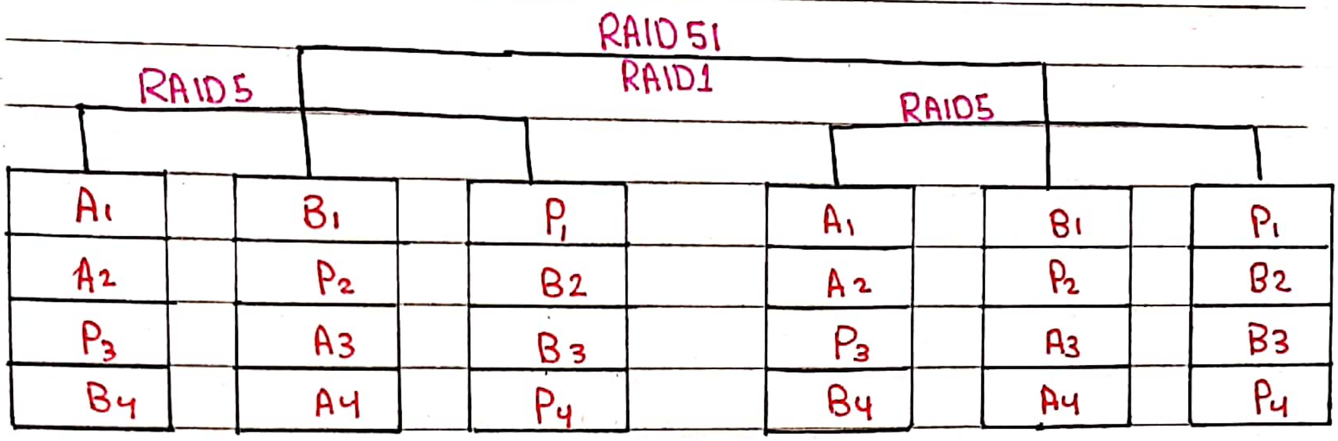
RAID 6



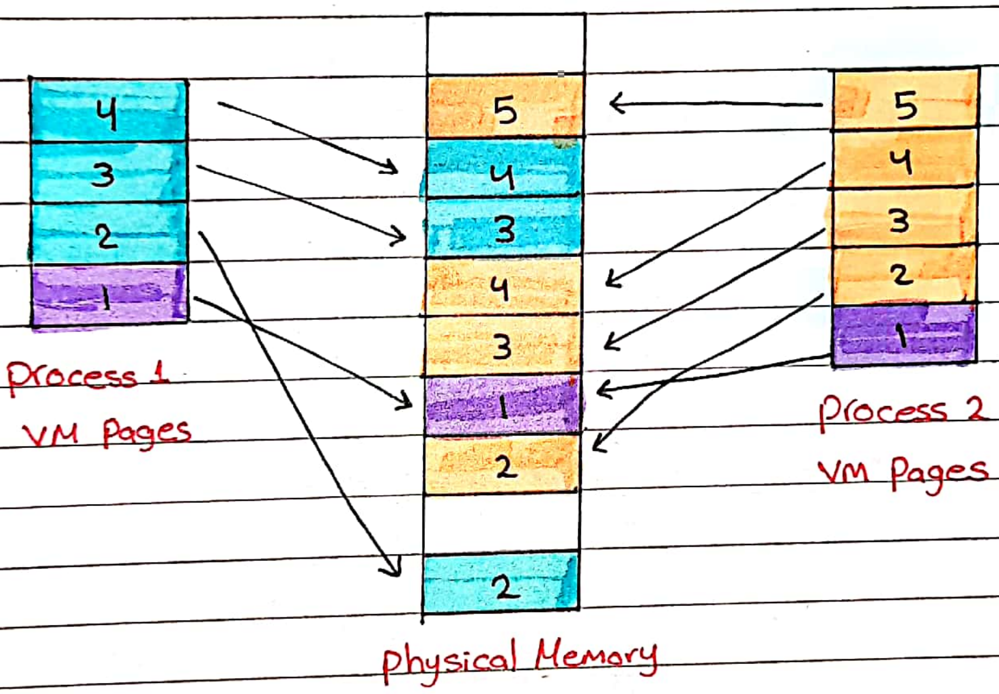
A_1	A_2	A_3	A_p	A_q
B_1	B_2	B_p	B_q	B_3
C_1	C_p	C_q	C_2	C_3
D_p	D_q	D_1	D_2	D_3
E_q	E_1	E_2	E_3	E_p

* ال p يحسب سطرياً أفقياً.

* ال q تحسب بطريقة أخرى كسطر مائل مش أفقي ، قطري diagonal (



Slide 94 : Sharing the physical Memory :



*مقسمة لـ Pages والـ page الوحدة أكثر شي = 4KB .

* بالأصل كل Process الها Pages خاصة فيها، الا إذا بدنا نعمل Sharing زي رقم 1 .

* الأبيض بيعبر عن Pages فاضية .

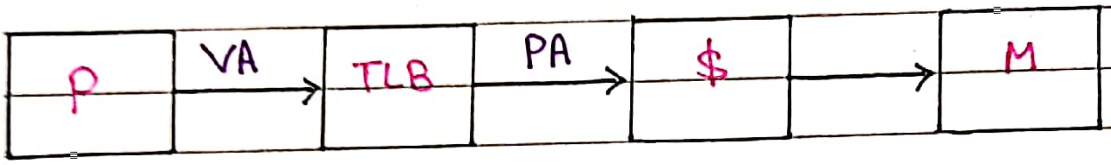
* المبرمج بفترض إنه بيقدر يستخدم أي Address من أكبر رقم لأقل رقم .

* مسؤولية الـ cpu hardware + OS بيوزعوا الـ Virtual spaces بالـ Physical Memory (Pages)

Slide 106 : TLB- Cache Interaction :

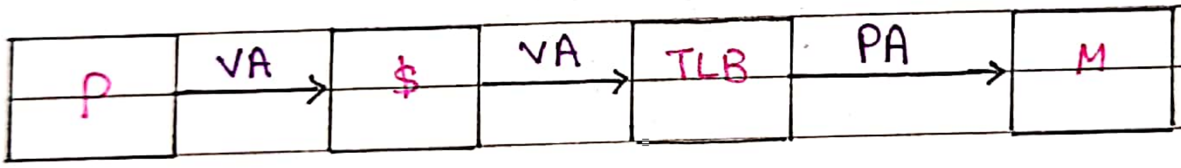
* Physical Address Tag :

* يعني ال cache يكون فيه physical Address مثله مثل ال Memory
فلازم أول يكون ال TLB وهيك بكون الوصول لل cache أبطأ.

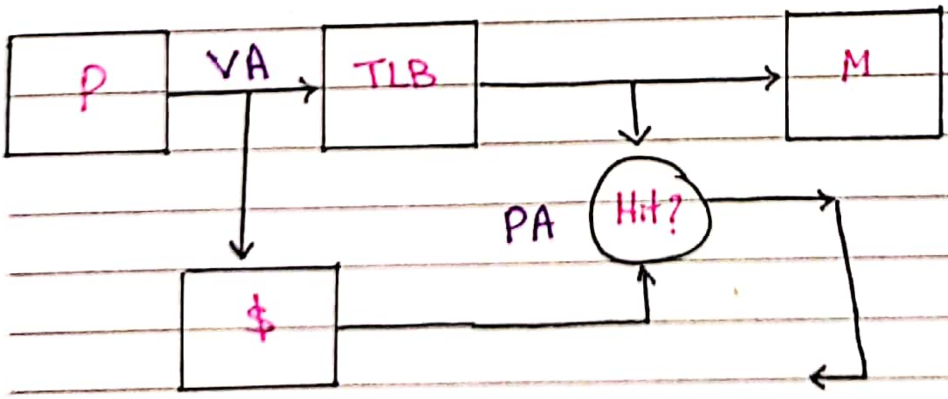


* Virtual Address Tag :

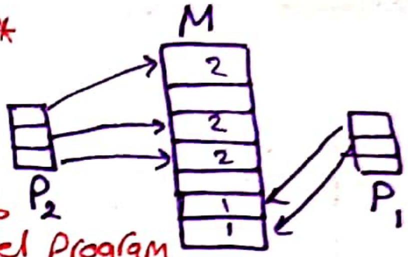
* بنخلي ال tags اللي في ال cache عبارة عن Virtual Address معناه ما في حاجة
نعمل Translation الا لو صار Miss .



* Physical Address Tag : cache Size = Page Size * associativity
] بهاي الحالة الخاصة .



*مافي Page في Process بنقدر تشوف Page ثانية
في Process ثاني وبالتالي هنا Protection. إلا إذا كانوا
الـ Process
عبارة عن Parallel program وطلبوا من الـ system إنه يعملوا shared فبقدروا
يشوفوا بعض.



Memory Protection

- Different tasks can share parts of their virtual address spaces

(read) (write) (Fetch
r w F

  - But need to protect against errant access

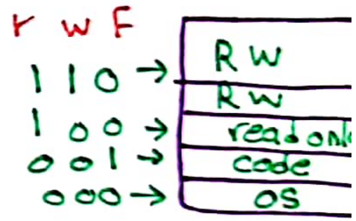
access غير صحيحة

  - Requires OS assistance

هو اللي بيكتشف الأخطاء

هو اللي بنسق وبحدد

- Hardware support for OS protection



  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode

بتودينا للـ operating system

  - System call exception (e.g., ecall in RISC-V)

Contents

The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy * كل اشي هو cache للي تحته موجود فيه جزء من معلومات اللي تحته .
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement → إذا جبنا اشي من تحت لفوق وين نحطه .
  - Finding a block → وين هذا الـ address إذا موجود بالـ level اللي احنا فيه .
  - Replacement on a miss → إذا كل الأماكن مليانة وين نحط .
  - Write policy → لما بدنا نكتب كيف نكتب

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement كل Mem block إله مكان واحد بنحدد بال index.
  - n-way set associative (2/4/8/16) مش أكثر من هيك
    - n choices within a set
  - Fully associative
    - Any location ماشا index كل الأماكن بال Cache متاحين.
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

مشكلات ال higher associativity

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index → بنحدد ال block بال Cache	1 ⊜
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries $	#entries
	Full lookup table VM	0

↳ فيها Page table

- Hardware caches
  - Reduce comparisons to reduce cost $VA \rightarrow \boxed{PT} \rightarrow PA$
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

Replacement

بال direct map مافي choices
بال high/fully associative فيه

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

يقسم ال Page لجزئين : recently used
و not recently used و بيتم ال Replacement
من ال not recently used
وال bit اللي بال VM اللي بستخدمها ال os
هي (٢ bit) ← يعني ال os كل
فترة وفترة بحط

فيها 0 بلف عكل ال Pages وبحط فيها 0 بعدين
ال Page اللي بصيرلها access بنحول ل 1 وهيك بنكون
أشرنا عال Pages اللي recently used واللي ضلوا صفر Not recently used

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Sources of Misses

تصنيف الـ Misses cache حسب سببهم.

- إجباري Compulsory misses (aka cold start misses)
  - First access to a block

لما أول مرة بدنا نعمل و نشغل البرنامج الـ Caches بكونوا لسا فاضيين فأول access, اجباري تكون Miss لأنها مش موجودة.

- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again

بحجم ما بكون واسع في الـ cache فلو عملنا → access عملومة ما لقيتها لأنه ادخطمكانها اشي ثاني فبصير في Miss

إذا بدي أقلل الـ capacity Miss بزيد حجم الـ cache.

- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

يعني بصرف بالـ cache مكان معين, اله index معين, انه الـ P بطلب من الـ M كذا blocks أكثر من واحد إلهم نفس الـ index بالـ cache فلو طلب الأول بيجي عادي لو طلب الثاني بده يجيبه للـ Cache بس يطرد الأول فبصير الأول والثاني يضلهم يطردوا بعض.

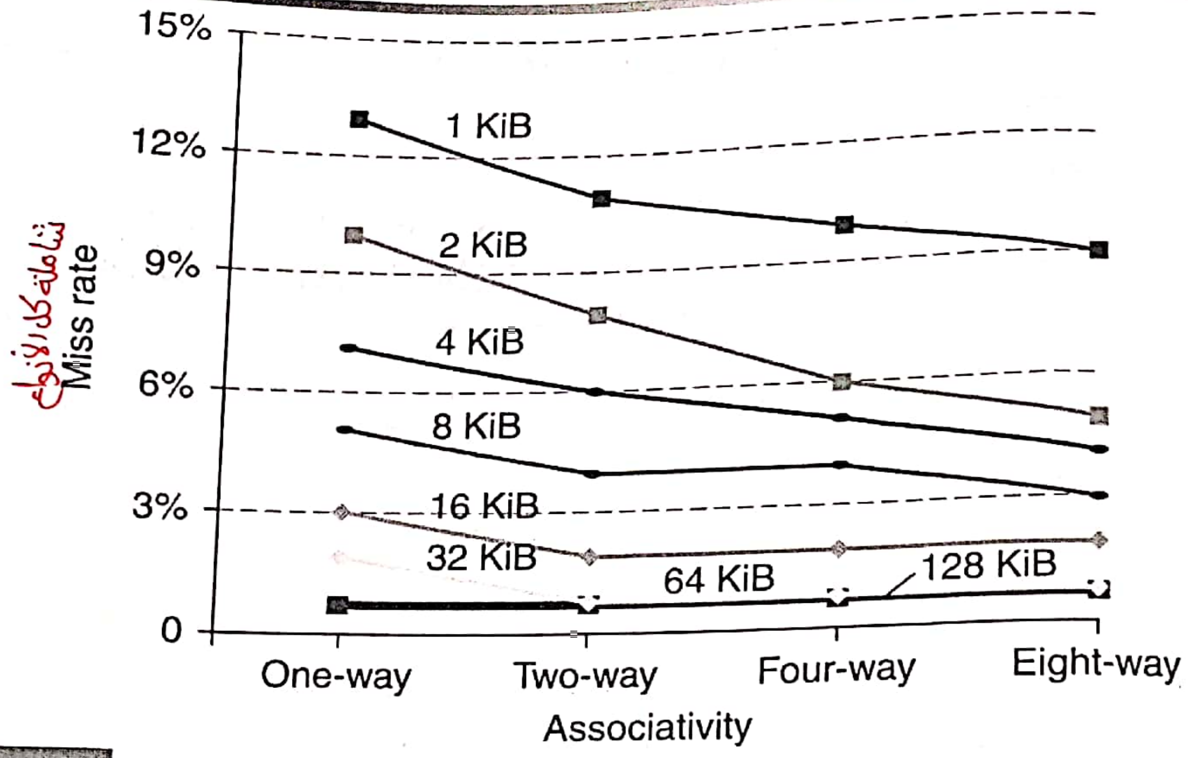
Cache Design Trade-offs

مع إنه الـ cache في أماكن فاضية بس أشروا عنفس الـ index

* AMAT = HT + Miss rate * Miss Penalty

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses لما الـ block size يكبر بجيب أجيب بـ Miss وحدة more data فلو بحتاجوا بكون وفرت عحالي Misses بالمستقبل	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Data Cache Miss Rate



* العلاقة بين ال Miss rate و ال Associativity : بتقل ال Miss rate كل مايزيد الAssociativity
السبب إنه ال conflict Missis بتقل

Contents

* لما يكبر ال cache بتقل ال Miss rate لأنه الMissis capacity بتقل

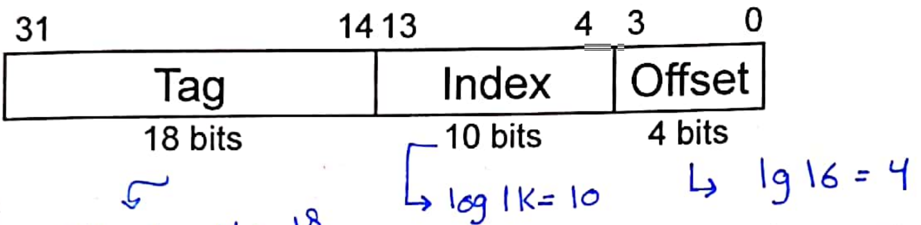
* لما نوصل ل 128 KiB ولو زدنا الAssociativity بضل ثابت ال Miss rate والسبب وجود ال compulsory Missis فلا يمكن ينقص أكثر من هيك

Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate

    dirty bit فيه ←
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block

    ↳ write_back لأنه
  - Blocking cache
    - CPU waits until access is complete



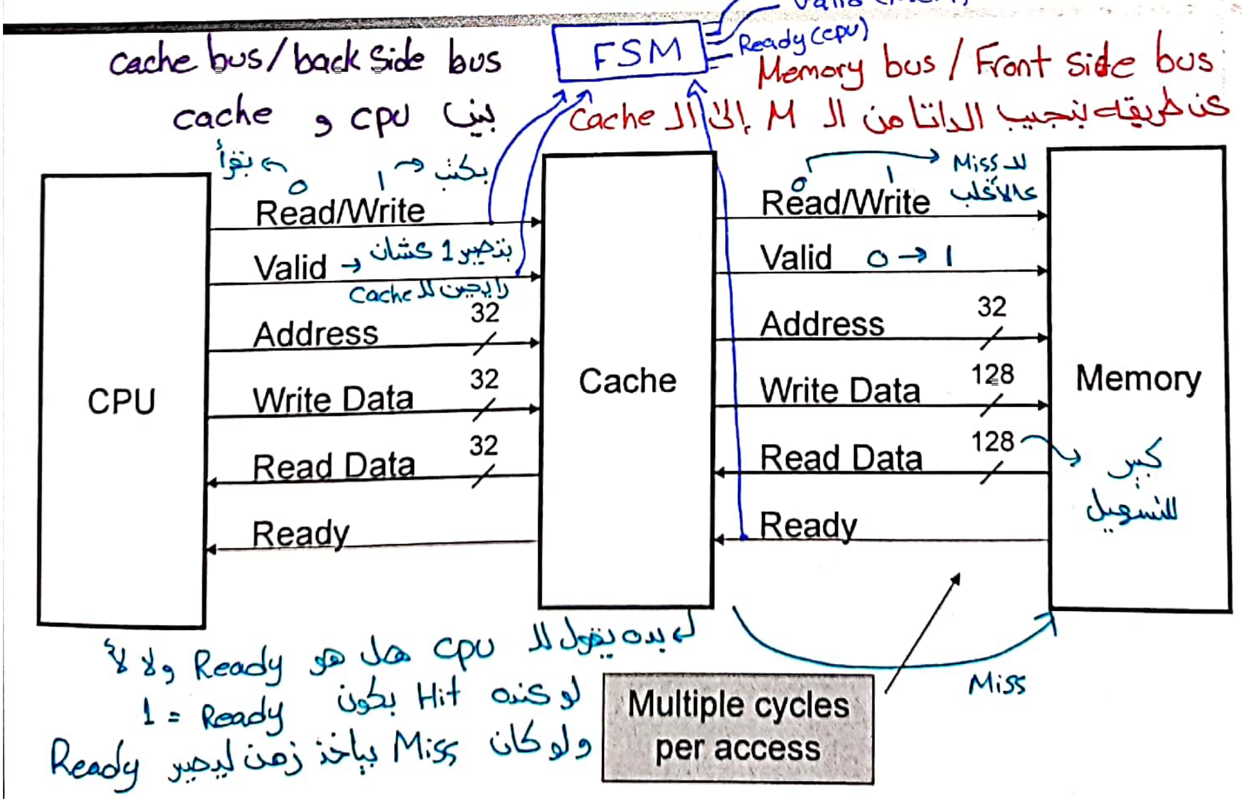
32 - 10 - 4 = 18

Interface Signals

Read/write (Mem)

Valid (Mem)

Ready (cpu)



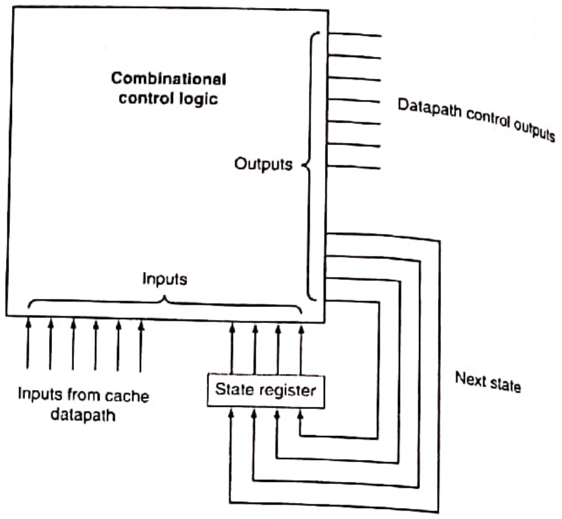
لـ بدو يقول للcpu هل هو Ready ولا لأ

لو كنه Hit بكون Ready = 1

ولو كان Miss بياخذ زمن ليصير Ready

Finite State Machines (Sequential)

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state = f_n (current state, current inputs)
- Control output signals = f_o (current state)



Sn	I/P	State	o/P
:	:	:	:

* خطوات تصميم Sequential circuits

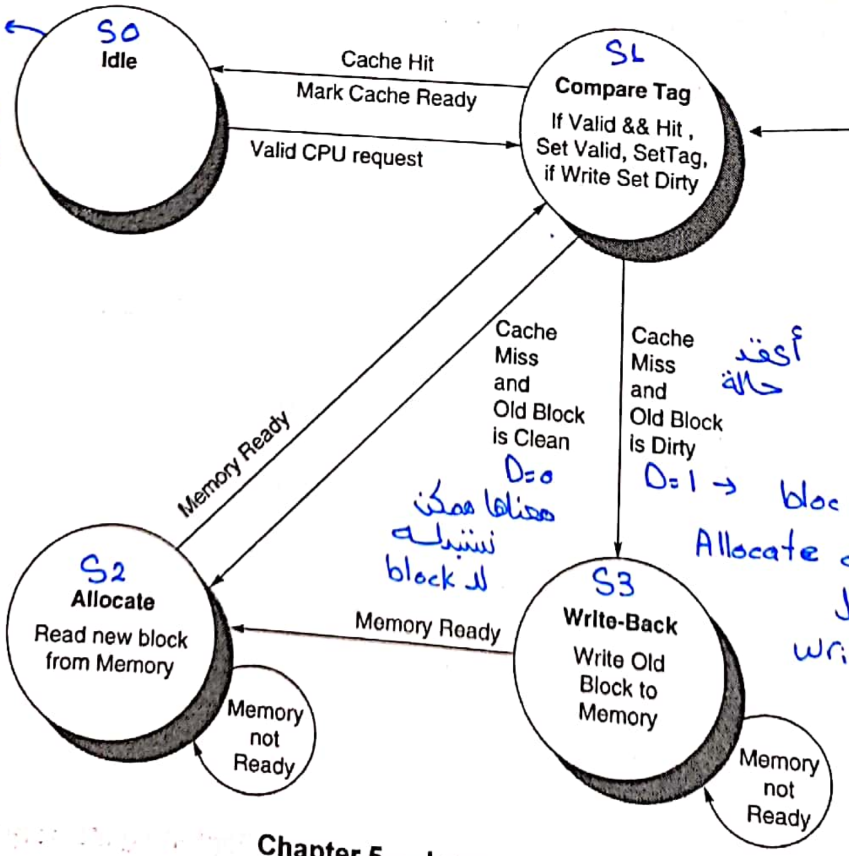
Sequential circuits → input فيها ذاكرة و ال out بعتمد ع ال
والـ state

Combinational circuits → ما فيها ذاكرة
والـ out بعتمد عال input بس

Cache Controller FSM

State البدائية

ال cache بكون بال Idle state



S0
Idle
Cache Hit
Mark Cache Ready
Valid CPU request
S1
Compare Tag
If Valid && Hit, Set Valid, SetTag, if Write Set Dirty
Memory Ready
Cache Miss and Old Block is Clean
D=0
معناها ممكن نستبدله لل block
Cache Miss and Old Block is Dirty
أعقد حالة
D=1 → يعني ال block قبل ما أعمله Allocate لازم أعمل Write-Back
S2
Allocate
Read new block from Memory
Memory not Ready
Memory Ready
S3
Write-Back
Write Old Block to Memory
Memory not Ready

Could partition into separate states to reduce clock cycle time

Contents

مترابطة ومتناضرة ومتراصة

Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches → هاي المشكلة بتصير بال Write-through
وال write-back

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				X= 0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

رح يكتب عال cache وعال Memory = 1

وهون في مشكلة: لأنه ال caches صار فيا عندهم نسختين من X وواحد عدل والثاني لا

فلو CPU B قرأ X حيقرأها 0 مش 1 وهاي مشكلة والحل في الخلف

Coherence Defined

تعريف الـ cache coherence

- Informally: Reads return most recently written value → يعني لما الـ cpu تعمل read لازم تقرأ أجدد قيمة
- Formally: التعريف المطول
  - ① P writes X; P reads X (no intervening writes)
    ⇒ read returns written value
  - ② P$_1$ writes X; P$_2$ reads X (sufficiently later) بعد فترة زمنية معقولة
    ⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - ③ P$_1$ writes X, P$_2$ writes X
    ⇒ all processors see writes in the same order
    - End up with the same final value for X ← نشوف بأي ترتيب كتبوا و بناخذ آخر كتابة كتبوها أو انكتبت

Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access

أنواع الـ cache coherence

- ① Snooping protocols
  - Each cache monitors bus reads/writes
- ② Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory ← دليل

Modified ← Shared → Invalid
(M S I) وحناخذ نوع من أكثر في عشان ←

Invalidating Snooping Protocols

حصريا

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
		I	I	X = 0
CPU A reads X	Cache miss for X	0	I	0
CPU B reads X	Cache miss for X	S 0	S 0	0
CPU A writes 1 to X	Invalidate for X	M 1	I	0
CPU B read X	Cache miss for X	S 1	S 1	1

Memory Consistency

لما P واحد بيكتب الثانيين
متى بشوفوا الكتابة.

- When are writes seen by other processors
  - "Seen" means a read returns the written value
  - Can't be instantaneously → بسبب إنه في تباعد بين ال Processors
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses ← مافي ← sequential consistency
    هاي الأيام لإنه بخلي البرامج بطيئة
    وإذا موجود لازم يكون معه ←
- Consequence
  - P writes X then writes Y
    ⇒ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

Contents

Static Dynamic

Multilevel On-Chip Caches

1 GHz 3-4 GHz

P
I$ D$
32 k 32 K

← الصفة

Characteristic	ARM Cortex-A53	Intel Core i7
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	Configurable 16 to 64 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	Two-way (I), four-way (D) set associative الضغط عليه أكبر	Four-way (I), eight-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes → Hit → Miss	64 bytes 64/4=16 word = 8 Dw each block
L1 write policy	Write-back, variable allocation policies (default is Write-allocate)	Write-back, No-write-allocate Hit Miss
L1 hit time (load-use)	Two clock cycles	Four clock cycles, pipelined → أسرع
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 2 MiB	256 KiB (0.25 MiB)
L2 cache associativity	16-way set associative	8-way set associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	12 clock cycles → أكبر وأوفر	10 clock cycles → أصغر وأبسط
L3 cache organization	-	Unified (instruction and data)
L3 cache size	-	8 MiB, shared
L3 cache associativity	-	16-way set associative
L3 replacement	-	Approximated LRU
L3 block size	-	64 bytes
L3 write policy	-	Write-back, Write-allocate
L3 hit time	-	35 clock cycles

2-Level TLB Organization

Characteristic	ARM Cortex-A53	Intel Core i7
Virtual address	48 bits	48 bits
Physical address	40 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 2 MiB, 1 GiB	Variable: 4 KiB, 2/4 MiB
TLB organization ↓ Cache PTE ل VA يحول PA الى	1 TLB for instructions and 1 TLB for data per core Both micro TLBs are fully associative, with 10 entries, round robin replacement 64-entry, four-way set-associative TLBs TLB misses handled in hardware The micro TLBs are L1 TLBs that are backed by two 64-entry l2 TLBs .	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, seven per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

→ 2^{44} : 16 TB

128+7 64+32

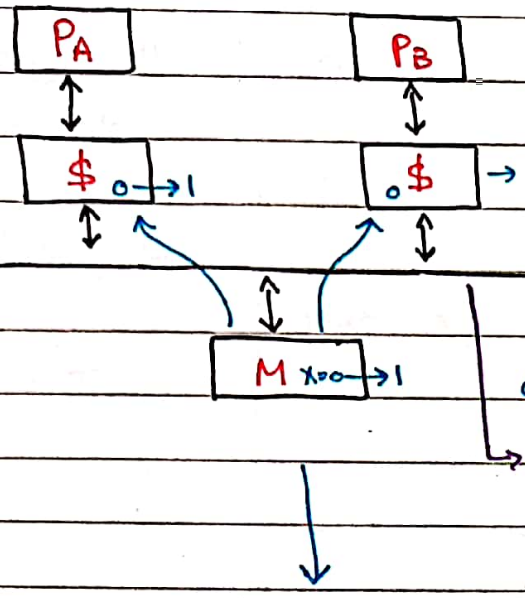
→ L1 TLB I TLB D

L2 TLB 512

Supporting Multiple Issue

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts
- Other optimizations
  - Return requested word first
  - Non-blocking cache
    - Hit under miss
    - Miss under miss

Slide 123 : Cache Coherence.



→ بضل فيه القيمة القديمة لـ X اللي هي صفر.

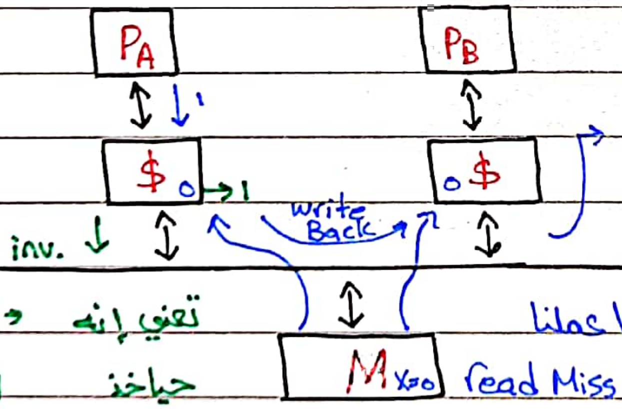
فهيك بطل في تعاون بين A و B

وهذا هو cache coherence problem

أي شي بروح عالـ M ببين عالـ Bus ↲

Time step	Event	CPU A's cache	CPU B's Cache	Memory
0				0
1	cpu A reads X	0		0
2	cpu B reads X	0	0	0
3	cpu A writes 1 to X	1	0	1

Slide 126 : Snooping Protocols.



Signal → تعني إنه

Exec. copy حياخذ

→ لما يراقب الـ cache الـ Bus ويشوف

صار فيه Exec. copy بخليه

invalid يعني بشيل الصفر

والـ M ضل فيها القيمة القديمة ولما رجعنا عملنا

read لـ B بكون Miss فبطلع الـ Bus ← read Miss

و بياخذ الـ 1 من الـ A

CPU activity	Bus Activity	CPU A's cache	CPU B's cache	Memory
				0
cpu A reads X	Cache Miss for X	0		0
cpu B reads X	Cache Miss for X	0	0	0
cpu A writes 1 to X	invalidate for X	1		0
cpu B reads X	Cache Miss for X	1	1	1

Slide 127 : Memory Consistency.

Processor A	Processor B
Flag = False ;	
	while (flag == false);
X = Compute();	
	Print (X) ;
Flag = True ;	

2 Processors بتعاملوا مع بعض : A بيعمل حسبة معينة وبيكتب قيمة جديدة لـ x
وبصير flag= true يعني إنه خلّص .
B بده يطبع القيمة الجديدة لـ x بس قبل ما يقراها ويطبعها بكون براقب الـ flag لو لسّا
flase بضلها بالـ while لو صارت الـ Flag=true بطلع من الـ Loop وبقدر إنه ينفذ
الـ Print . ولو فيه Sequential Consistency فلمّا تطلع الـ flag=t معناها كتب
قيمة X الجديدة فبشتغل البرنامج صح .

هاي الأيام ما في Sequential Consistency والـ Processors بتكتب وفي out of order
execution فممكن إنه قيمة الـ Flag تظهر لـ B قبل ما تظهرله القيمة الجديدة لـ X
وبالتالي البرنامج ما بشتغل صح فالحل إنه المبرمجين هالأيام لما يكتبوا Parallel Programs
يعتمدوا على تقنيات خاصة (For synchronisation) حتى تضمنلهم إنه الشغل
صح .

Supporting Multiple Issue

(بس احنا حنحكي بشكل عام)

→ ARM and Intel

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts (الدفتر بشرحها)
- Other optimizations
  - ① Return requested word first
  - ② Non-blocking cache
    - Hit under miss
    - Miss under miss
  - ③ Data prefetching → بساعدنا نقلل مثلا capacity Missis

Contents

- 5.1 Introduction
- 5.2 Memory Technologies
- 5.3 The Basics of Caches
- 5.4 Measuring and Improving Cache Performance
- 5.5 Dependable Memory Hierarchy
- 5.11 Redundant Arrays of Inexpensive Disks
- 5.6 Virtual Machines
- 5.7 Virtual Memory
- 5.8 A Common Framework for Memory Hierarchy
- 5.9 Using a Finite-State Machine to Control a Simple Cache
- 5.10 Cache Coherence
- 5.13 The ARM Cortex-A53 and Intel Core i7 Memory Hierarchies
- **5.16 Fallacies and Pitfalls**
- **5.17 Concluding Remarks**

Pitfalls

§5.16 Fallacies and Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

Pitfalls

- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores ⇒ need to increase associativity
- Using AMAT to evaluate performance of out-of-order processors
  - Ignores effect of non-blocked accesses
  - Instead, evaluate performance by simulation

Pitfalls

- Extending address range using segments
  - E.g., Intel 80286
  - But a segment is not always big enough
  - Makes address arithmetic complicated
- Implementing a VMM on an ISA not designed for virtualization
  - E.g., non-privileged instructions accessing hardware resources
  - Either extend ISA, or require guest OS not to use problematic instructions

Concluding Remarks

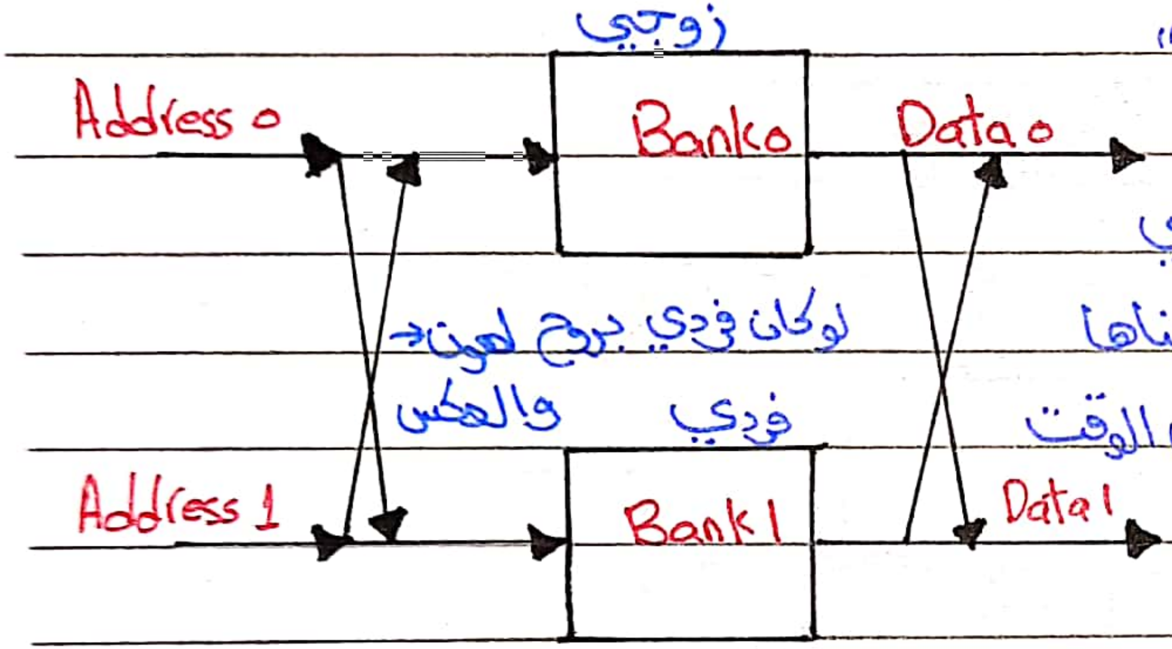
- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors

COMPUTER ORGANIZATION AND DESIGN

The Hardware/Software Interface

RISC-V Edition

Slide 131 : Supporting Multiple Issue.



" هذه الرسمة توضح cache مقسوم ل 2 Banks "
فبنحط ال Set اللي ال Address تاعهم زوجي
ب Bank وال Set اللي الAddress تاعهم فردي
بال Bank الآخر. ولما يكون في 2 Bank معناها
فيه 2 Ports فممكن ال P يعمل 2 access بنفس الوقت
وهيك بنكون حسنا الPerformance

Slide 131 : Return requested word first.

- Early restart

1	2	3	4

1 → 2 → 3 → 4

*Requested word : word 3

* 🟦 processing performed background

- Critical word first

1	2	3	4

3 → 4 → 1 → 2

* الـ Block هون بتكون من 4 word في التصاميم القديمة كنا لازم نجيب w1, w2, w3, w4 بعد ما نجيبهم كلهم نحطهم بالـ Cache والـ P بياخد الـ Data من الـ cache بعد ما يوصل الـ Block كامل.

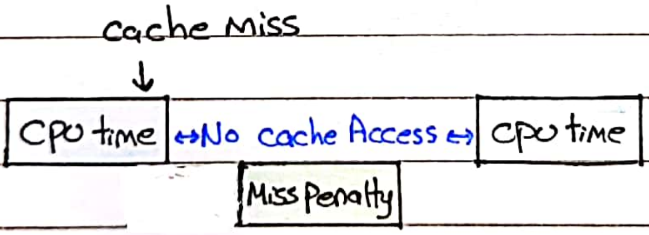
* بالـ Early restart لما توصل w3 ما بنستنى للتخزن بالـ Cache ونعطيها للـ P اللي بدو يه من الـ cache لا بنمررها للـ P .

* بالـ Critical word first بدل ما ينجابوا w1, w2, w3, w4 بنطلبها من الـ Memory من : Address 3 فبيجوا w3, w4, w1, w2 فبنقلل الـ Miss penalty أكثر .

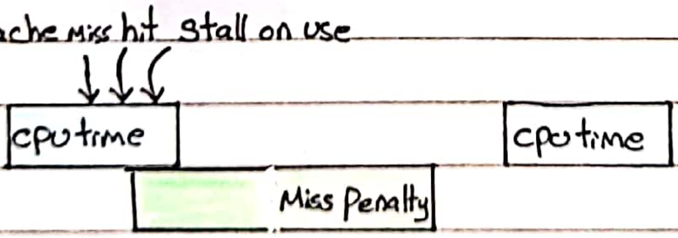
* في الأجهزة الحديثة موجودة الطريقتين .

Slide 131 : Non-blocking cache.

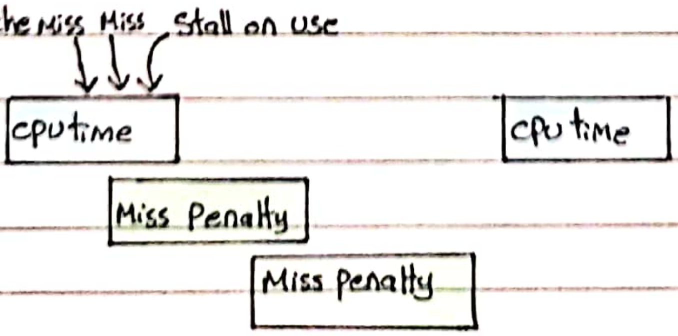
Blocking Cache :



Hit Under Miss :



Miss under Miss :



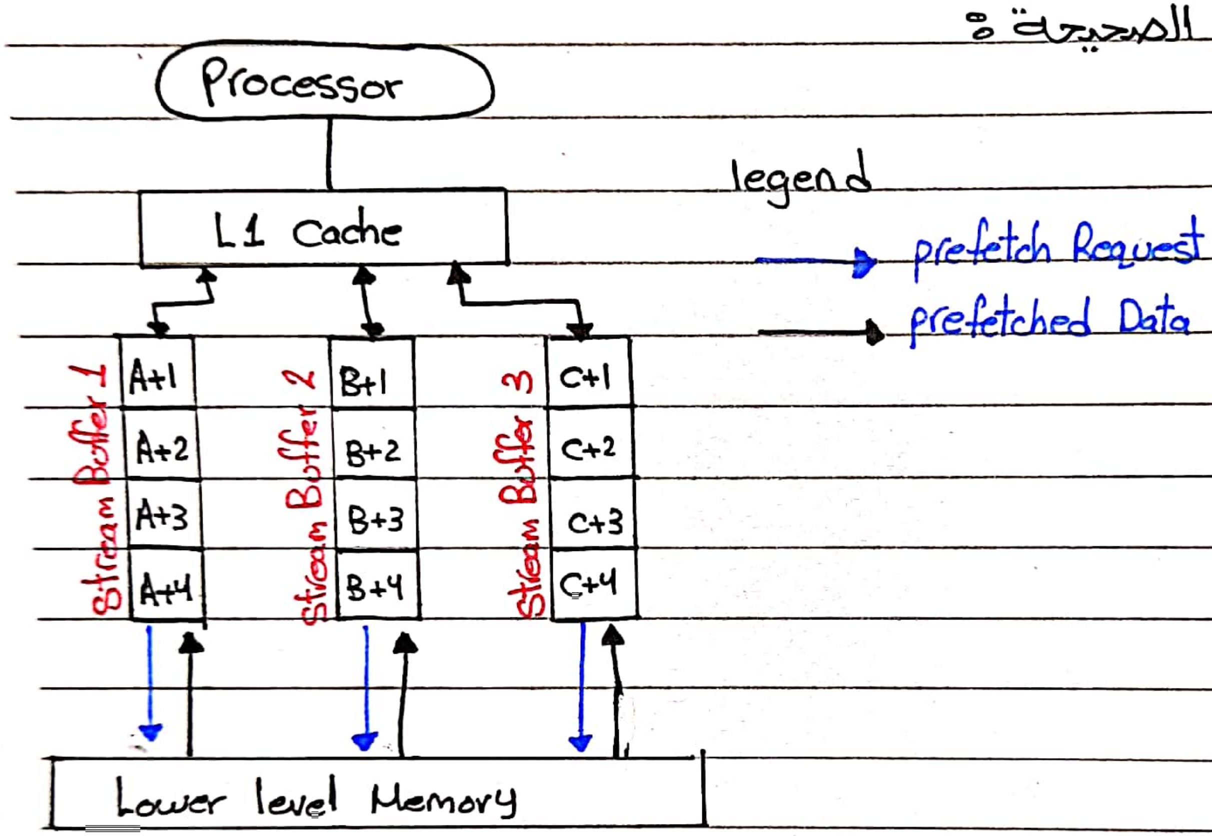
Slide 131 : Data prefetching

Access Block A

لما الـ Processor يطلب A الـ prefetch circuit بتستبق الأحداث →
وبتعمل prefetching الـ Next sequential Block وهنا اشي
ممتاز وبنوفر على حالنا Miss .

Prefetch Block A+1

← الطريقة المعقدة الصحيحة :



Chapter 6

Parallel Processors from Client to Cloud

مثل الهواتف
والنوت بوك

Adapted by Prof. Gheith Abandah

Contents

Introduction

- Goal: connecting multiple computers to get higher performance
  - Multiprocessors (بصير هيك اسم الجهاز)
  - Scalability, availability, power efficiency

① قدرات ال Multip أكبر من ال Single P ② ③ بصرف طاقة أقل → multiple processors are more power efficiency than Single Processors

- Task-level (process-level) parallelism
  - High throughput for independent jobs

→ ال multiple jobs يشتغلوا على Multiple cores

- Parallel processing program
  - Single program run on multiple processors Ex. SPMD → Data
- Multicore microprocessors
  - Chips with multiple processors (cores)

Hardware and Software

- Hardware
  - ① Serial: e.g., Pentium 4 → كان موجود قبل 20 سنة
  - ② Parallel: e.g., quad-core Xeon e5345

    ↳ بنفذ 4 program بنفس الوقت
- Software
  - ← لو نفذهون بياخذ core واحد وبترك الباقي
  - ① Sequential: e.g., matrix multiplication
  - ② Concurrent: e.g., operating system → معقد قسموه لأجزاء كل جزء اله برمجة خاصة فيه وبتعاونوا مع بعض
- Sequential/concurrent software can run on serial/parallel hardware
  - Challenge: making effective use of parallel hardware

ممكن نفذه

← لو نفذ هون رح ينفذ بزمن أقل. وبنفذوا وحدة ورا الثانية

What We've Already Covered

- §2.11: Parallelism and Instructions ← بالاوع
  - Synchronization
- §3.6: Parallelism and Computer Arithmetic
  - Subword Parallelism → بعطينا Performence أعلى.
- §4.10: Parallelism and Advanced Instruction-Level Parallelism
- §5.10: Parallelism and Memory Hierarchies
  - Cache Coherence

Contents

6.1 Introduction

6.2 The Difficulty of Creating Parallel Programs → البرمجة صعبة

6.3 SISD, MIMD, SIMD, SPMD, and Vector

6.4 Hardware Multithreading

6.5 Shared Memory Multiprocessors

6.6 Introduction to Graphics Processing Units

6.7 Clusters and Message-Passing Multiprocessors

6.8 Introduction to Multiprocessor Network Topologies

6.10 Multiprocessor Benchmarks and Performance Models

6.11 Benchmarking Intel Core i7 960 and NVIDIA Tesla GPU

6.12 Multiple Processors and Matrix Multiply

6.13 Fallacies and Pitfalls

6.14 Concluding Remarks

Contents

6.2 The Difficulty of Creating Parallel Programs

- Parallel Programming
- Amdahl's Law
- Scaling
- Strong and Weak Scaling

Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement → البرنامج مش بس يشتغل صح لا يشتغل صح
  ويكون الPerformence عالي و التحسين جدًا عالي يكون .
  - Otherwise, just use a faster uniprocessor, since it's easier! ← لو كان صعب عليك تعمل
    البرمجة بتحسين عالي فبنستخدم ال Fster uniprocessor.
- Difficulties
  - Partitioning → تجزء الشغل اللي بدك تعمله لأجزاء
  - Coordination → خلال التنفيذ لازم ينسقوا ال cores كيف يبلشوا
    ويخلصوا مع بعض و يشتغلوا مزبوط
  - Communications overhead
    ← لما يصيروا ال Processors يتعاونوا مع بعض يصيروا يحتاجوا
    Data من بعض فبصير في زمن كبير غيرعن لما ال Processor يشتغل
    لحاله بدون ما يشارك
    Processors آخرين.

← لما نكتب Parallel Program لازم ننتبه لهاي الصعوبات

Amdahl's Law

بحكيلنا عن ال Speed up اللي
ممكن نحصله .

← وأحسن Speed up ممكن نحصله يعتمد ع Sequential part (1-f)

- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
  - $T_{new} = T_{parallelizable}/100 + T_{sequential}$
  - $$\text{Speedup} = \frac{1}{(1-F_{paralleliable})+F_{paralleliable}/\underset{p}{100}} = 90$$
  - Solving: $F_{parallelizable} = 0.999$
- Need sequential part to be 0.1% of original time

وهنا فعلًا صعب فهذا واحد من الصعوبات اللي بنواجهها

Scaling Example

أمثلة بتوضحلنا العلاقة بين حجم المسألة وعدد ال Processors

نعتبرها Serial

- Workload: sum of 10 scalars, and 10 × 10 matrix sum

  بقدر أجزئه

  - Speed up from 10 to 100 processors → مقارنة لنسبة ل 1 processor
- Single processor: Time = (10 + 100) × t_{add} رح يحتاج ١١٠ عملية جمع
- 10 processors

  عملية جمع ← processor →

  - Time = 10 × t_{add} + 100/10 × t_{add} = 20 × t_{add} → بدل ما يكون ١١٠ بكون 20
  - Speedup = 110/20 = 5.5 (55% of potential) → $Eff = \frac{S}{P} * 100\%$

    speedup → S; عدد ال Processors → P
- 100 processors

  عملية جمع ← Processor →

  - Time = 10 × t_{add} + 100/100 × t_{add} = 11 × t_{add}
  - Speedup = 110/11 = 10 (10% of potential) → Eff ↓ بضل سيء
- Assumes load can be balanced across processors

Scaling Example (cont)

- What if matrix size is 100 × 100?
- Single processor: Time = (10 + 10000) × t_{add}
- 10 processors
  - Time = 10 × t_{add} + 10000/10 × t_{add} = 1010 × t_{add}
  - Speedup = 10010/1010 = 9.9 (99% of potential) ↳ Eff (وهذا جيد)
- 100 processors
  - Time = 10 × t_{add} + 10000/100 × t_{add} = 110 × t_{add}
  - Speedup = 10010/110 = 91 (91% of potential) ↳ Eff
- Assuming load balanced

Strong vs Weak Scaling

- Strong scaling: problem size fixed
  - As in example
- Weak scaling: problem size proportional t number of processors
  - 10 processors, 10 × 10 matrix
    - Time = 20 × t_{add}
  - 100 processors, 32 × 32 matrix
    - Time = 10 × t_{add} + 1000/100 × t_{add} = 20 × t_{add}
  - Constant performance in this example

لو كبرنا المسألة
لما نستخدم عدد
أكبر من ال
Processors
بصير weak
Scaling

Contents

Contents

6.3 SISD, MIMD, SIMD, SPMD, and Vector
- Flynn's Classification
- Vector Processors
- SIMD Instruction Extensions

Instruction and Data Streams

- An alternate classification

← بجيب Instruction
I → P
↕
D → بشتغل ع Data

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: → أبسط اشي Intel Pentium 4	**SIMD**: SSE (vector) منها instructions of x86
	Multiple	**MISD**: Data وحدة بشتغل عليها بأكثر من Instruction No examples today	**MIMD**: مثال intel core i7 Intel Xeon e5345

← برنامج واحد وبيحوي if statement عشان كل core ينفذ شغلته بس

- SPMD: Single Program Multiple Data
  - A parallel program on a MIMD computer
  - Conditional code for different processors

← كل Processor بشتغل شغلة مختلفة عن ال Processor الآخر.

Vector Processors

- Highly pipelined function units
- Stream data from/to vector registers to units
  - Data collected from memory into registers
  - Results stored from registers to memory
- Example: Vector extension to RISC-V
  - v0 to v31: 32 × 64-element registers, (64-bit elements)
  - Vector instructions
    - `fld.v`, `fsd.v`: load/store vector (Floating Point)
    - `fadd.d.v`: add vectors of double
    - `fadd.d.vs`: add scalar to each element of vector of double
- Significantly reduces instruction-fetch bandwidth

بتطلع ال Data Streams مش 1 value

اضافات

ممكن أضرب Vector بـ Scalar مثلاً

بـ 1 instruction بنفذ (64 element)

Example: DAXPY (Y = a × X + Y)

```
for(i=0, i<64, i++)
  y[i]= a* x[i]+y[i];
```

① Conventional RISC-V code:

```
       fld     f0,a(x3)       // load scalar a
       addi    x5,x19,512     // end of array X
loop:  fld     f1,0(x19)      // load x[i]
       fmul.d  f1,f1,f0       // a * x[i]
       fld     f2,0(x20)      // load y[i]
       fadd.d  f2,f2,f1       // a * x[i] + y[i]
       fsd     f2,0(x20)      // store y[i]
       addi    x19,x19,8      // increment index to x
       addi    x20,x20,8      // increment index to y
       bltu    x19,x5,loop    // repeat if not done
```

بأشر ع بداية ال Array

64*8

بأشر عالنهاية

لـ بنفذوا 64 مرة

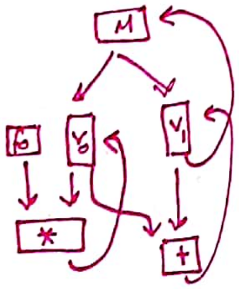
بأشر ع Y

2 + 8 * 64 = 514 instruction

② Vector RISC-V code:

```
fld         f0,a(x3)      // load scalar a
fld.v       v0,0(x19)     // load vector x
fmul.d.vs   v0,v0,f0      // vector-scalar multiply
fld.v       v1,0(x20)     // load vector y
fadd.d.v    v1,v1,v0      // vector-vector add
fsd.v       v1,0(x20)     // store vector y
```

بأشر ع X



هنا البرنامج أفضل بس 6 instruction

Vector vs. Scalar

- Vector architectures and compilers
  - Simplify data-parallel programming
  - Explicit statement of absence of loop-carried dependences
    - Reduced checking in hardware
  - Regular access patterns benefit from interleaved and burst memory
  - Avoid control hazards by avoiding loops
- More general than ad-hoc media extensions (such as MMX, SSE)
  - Better match with compiler technology

SIMD (Single Instruction Multiple Data)

- Operate elementwise on vectors of data
  - E.g., MMX and SSE instructions in x86
    - Multiple data elements in 128-bit wide registers
- All processors execute the same instruction at the same time
  - Each with different data address, etc.
- Simplifies synchronization
- Reduced instruction control hardware
- Works best for highly data-parallel applications

طلع بعدهم

AVx 256 bit
AVx 512 bit

بنقدر نقسمه

double ← 8DP ل

Single ← 16SP

Vector vs. Multimedia Extensions

مثل ال SIMD

- Vector instructions have a variable vector width, multimedia extensions have a fixed width

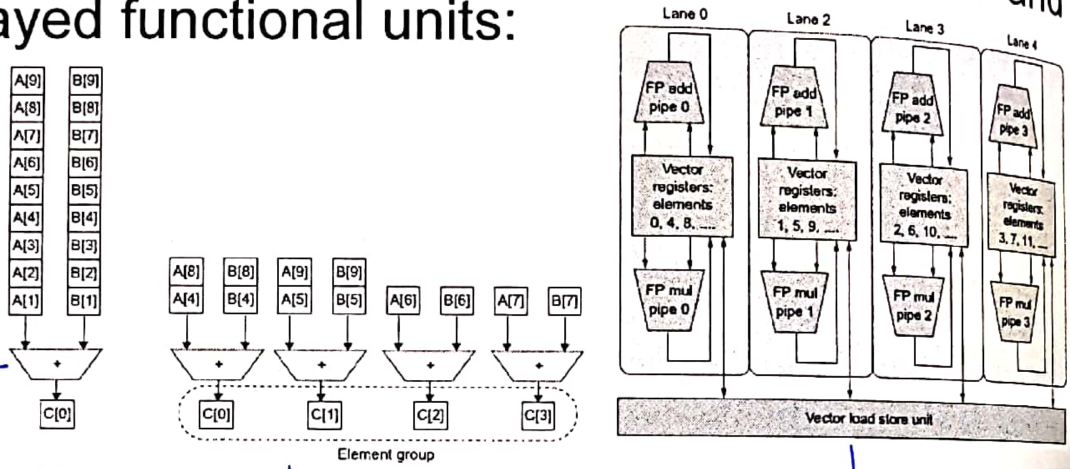
إنت بتحدد طول ال Vector

- Vector instructions support strided access, multimedia extensions do not

يكون في قفزة بين ال Data والثانية

لازم تشتغل ع Data جنب بعضها

- Vector units can be combination of pipelined and arrayed functional units:



A[9] A[8] A[7] A[6] A[5] A[4] A[3] A[2] A[1]
B[9] B[8] B[7] B[6] B[5] B[4] B[3] B[2] B[1]
+
C[0]
A[8] A[4] B[8] B[4] A[9] A[5] B[9] B[5] A[6] B[6] A[7] B[7]
C[0] C[1] C[2] C[3]
Element group
Lane 0
Lane 2
Lane 3
Lane 4
FP add pipe 0
FP add pipe 1
FP add pipe 2
FP add pipe 3
Vector registers: elements 0, 4, 8, ...
Vector registers: elements 1, 5, 9, ...
Vector registers: elements 2, 6, 10, ...
Vector registers: elements 3, 7, 11, ...
FP mul pipe 0
FP mul pipe 1
FP mul pipe 2
FP mul pipe 3
Vector load store unit

FU Pipelined

بدل ما كل الشغل ينعمل ع FU لحال صار ع 4 FU

نقسموا ال register ل 4 أجزاء

بسهل وبقلل عدد ال input/output لل register الواحد

Contents

Multithreading

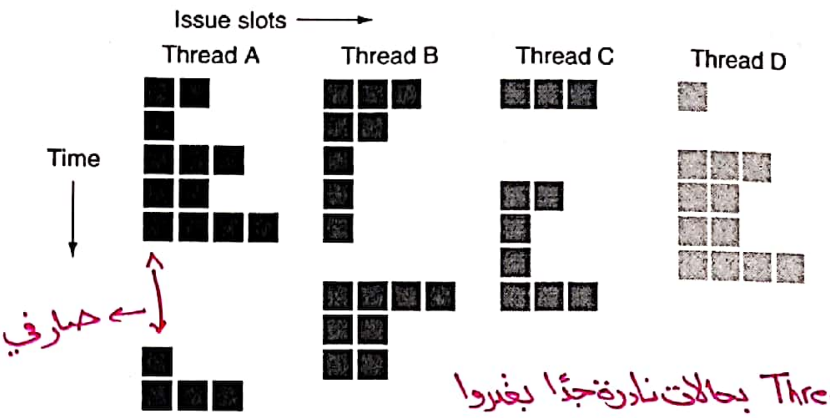
§6.4 Hardware Multithreading

- Performing multiple threads of execution in parallel → Multiple process Core واحد بنفذ
  - Replicate registers, PC, etc.
  - Fast switching between threads
- ① Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- ② Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

Simultaneous Multithreading

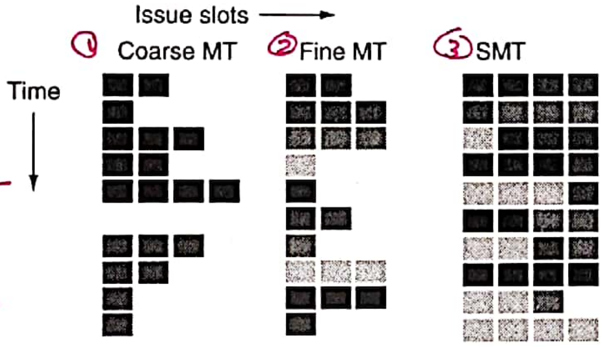
- ③ In multiple-issue dynamically scheduled processor (SMT)
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT → Hyper Threading
  - Two threads: duplicated registers, shared function units and caches

Multithreading Example



حارفي انقطاع لأي سبب كان

* يبين إنه ال Thread بحالات نادرة جدًا بقدروا
ينفذوا 4



لما يشعر إنه في توقف
بينقل لل Thread اللي بعده ،
قادر على إخفاء long stalls

بخلطوا ال Threads
مع بعض بس كل وحدة
معها Flag هي لأي
Thread تابعة
وبخلط تنفيذهم بس مش على
شكل مجموعات . أعقد

لـ كل cycle بجيب من
Thread

وهاد كويس لإنه بباعد ال Instruction بـ thread واحد عن بعض
وهاد يحسن لو كان في dependence ، قادر على إخفاء
short /small
. Stalls
أبسط

Future of Multithreading

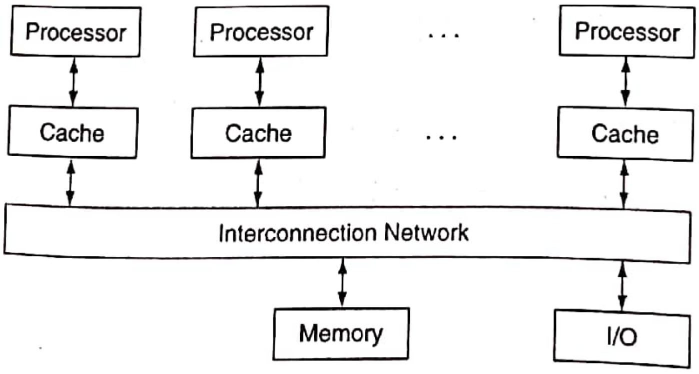
- Will it survive? In what form?
- Power considerations ⇒ simplified microarchitectures
  - Simpler forms of multithreading
- Tolerating cache-miss latency
  - Thread switch may be most effective
- Multiple simple cores might share resources more effectively

Contents

Shared Memory

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - ① UMA (uniform) vs. ② NUMA (nonuniform) → shared أنواع ال Memory

كلهم بعيدين نفس البعد
عن ال Mem



→ 1 Memory نتشاركوا مع

Example: Sum Reduction

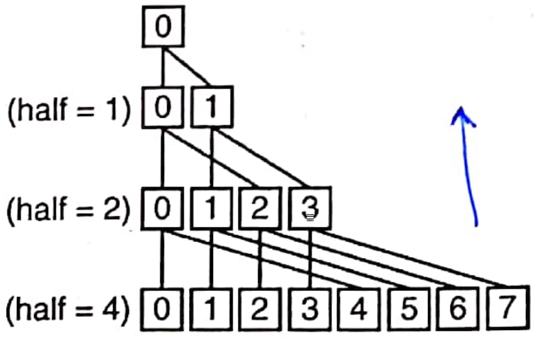
- Sum 64,000 numbers on 64 processor UMA
  - Each processor has ID: 0 ≤ Pn ≤ 63 → عشان كنا 64 processor
  - Partition 1000 numbers per processor
  - Initial summation on each processor

```
sum[Pn] = 0;
  for (i = 1000*Pn;
         i < 1000*(Pn+1); i += 1)
    sum[Pn] += A[i];
```

- Now need to add these partial sums
  - Reduction: divide and conquer
  - Half the processors add pairs, then quarter, …
  - Need to synchronize between reduction steps

Example: Sum Reduction

(1 التنصيف وبكون مسؤولية النص التحتاني
انه ينقل قيمة الفوقاني)



```
half = 64;
do
  synch();
  if (half%2 != 0 && Pn == 0)
    sum[0] += sum[half-1];
    /* Conditional sum needed when half is odd;
       Processor0 gets missing element */
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] += sum[Pn+half];
while (half > 1);
```

← الهدف منه
إذا كان الرقم
فردي عشان نعالج
المشكلة

Contents

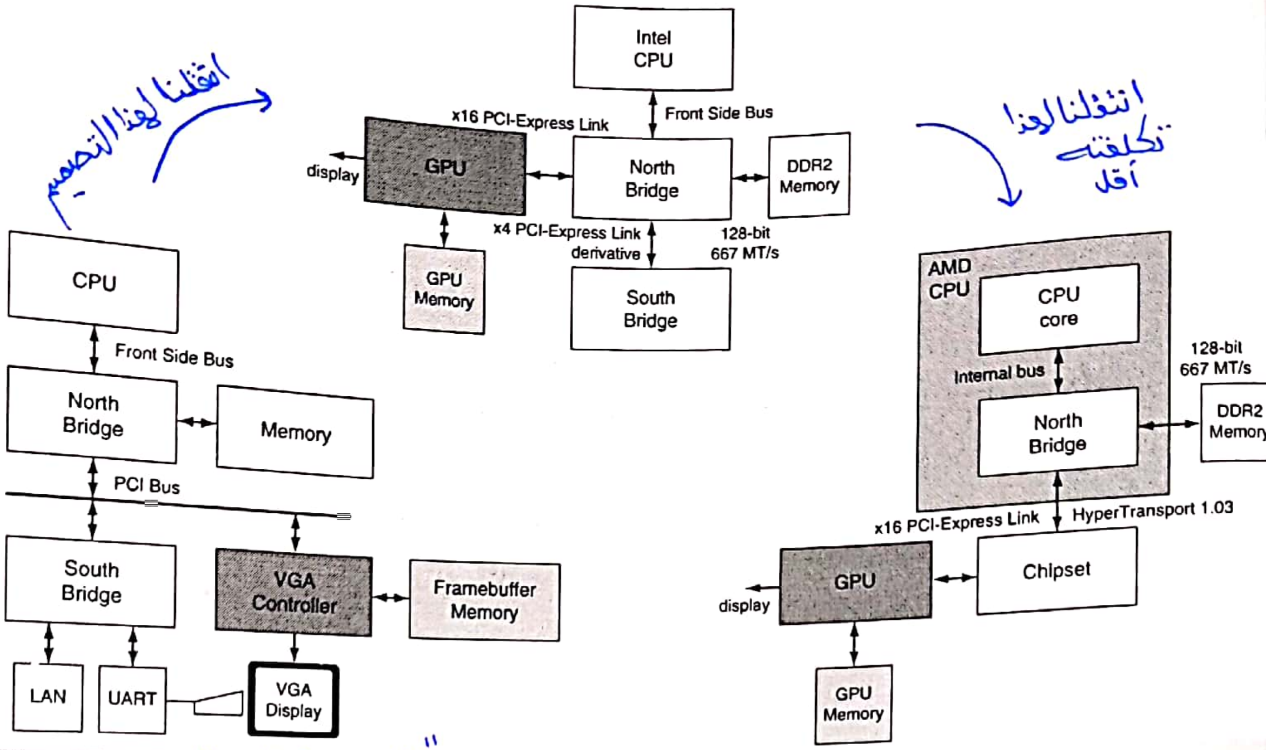
History of GPUs

- Early video cards → بالثمانينات
  - Frame buffer memory with address generation for video output
- 3D graphics processing → بنهاية الثمانينات
  - Originally high-end computers (e.g., SGI)
  - Moore's Law ⇒ lower cost, higher density
  - 3D graphics cards for PCs and game consoles → مثل البلايستيشن
- Graphics Processing Units GPUs
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

تحدد الأماكن المضاءة

تحويل الفيديو لـ Signals

Graphics in the (Computer) System



انتقلنا لهذا التصميم

انتقلنا لهذا تكلفته أقل

"تصميم قديم"

GPU Architectures

- Processing is highly data-parallel → Alot of data بنشغل عليهم نفس الحسابات
  - GPUs are highly multithreaded
  - Use thread switching to hide memory latency
    - Less reliance on multi-level caches
  - Graphics memory is wide and high-bandwidth
- Trend toward general purpose GPUs متنوعة التطبيقات
  - Heterogeneous CPU/GPU systems → Performance بتعطيني أفضل من ال CPU
  - CPU for sequential code, GPU for parallel code
- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)

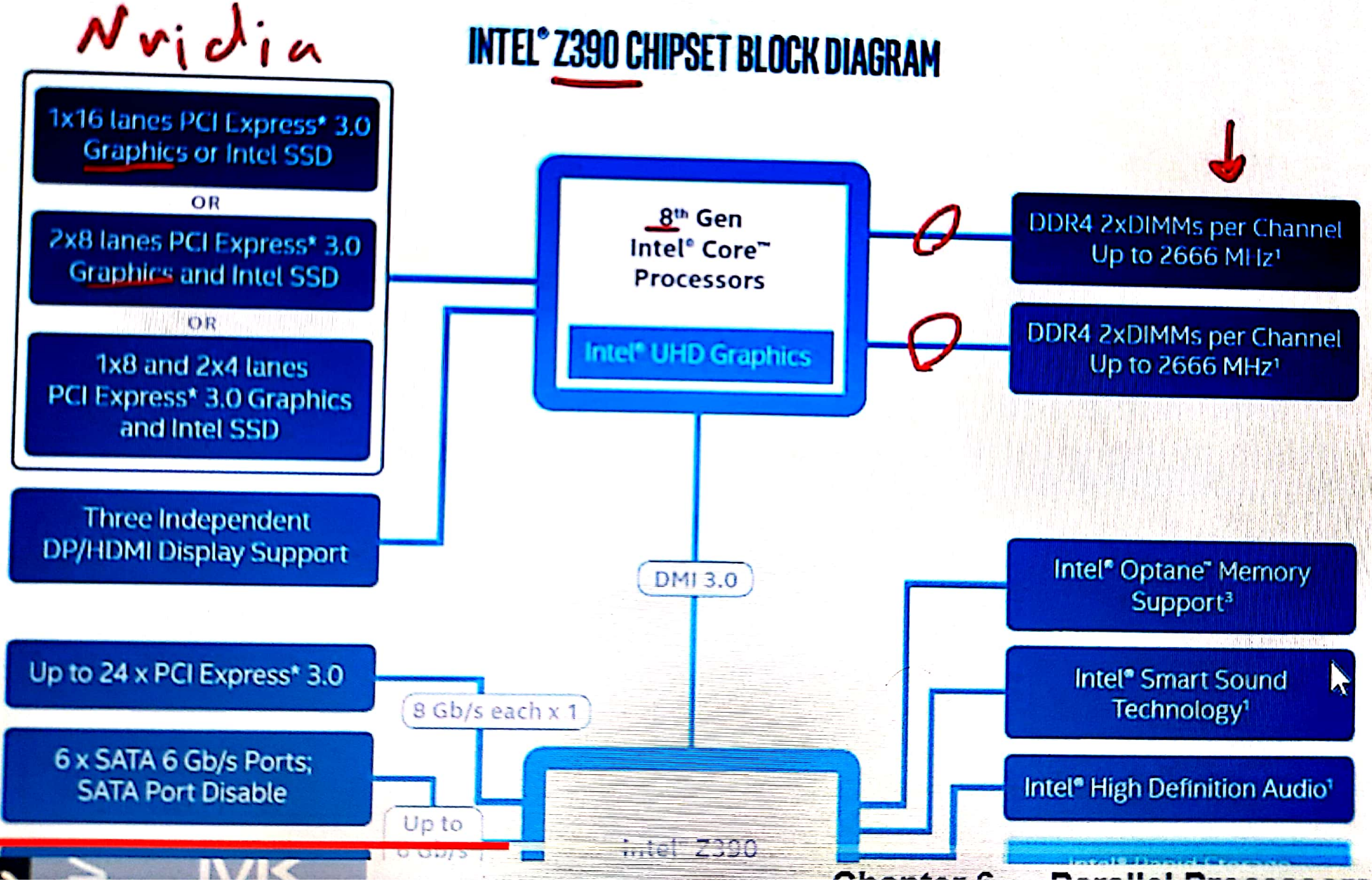
لعمن NVidia

Modern Computer

Thursday, April 23, 2020 9:45 AM

i3 i7

Nvidia



INTEL® Z390 CHIPSET BLOCK DIAGRAM

1x16 lanes PCI Express* 3.0 Graphics or Intel SSD

OR

2x8 lanes PCI Express* 3.0 Graphics and Intel SSD

OR

1x8 and 2x4 lanes PCI Express* 3.0 Graphics and Intel SSD

Three Independent DP/HDMI Display Support

8th Gen Intel® Core™ Processors

Intel® UHD Graphics

DDR4 2xDIMMs per Channel Up to 2666 MHz[1]

DDR4 2xDIMMs per Channel Up to 2666 MHz[1]

DMI 3.0

Up to 24 x PCI Express* 3.0

8 Gb/s each x 1

6 x SATA 6 Gb/s Ports; SATA Port Disable

Up to 6 Gb/s

Intel® Z390

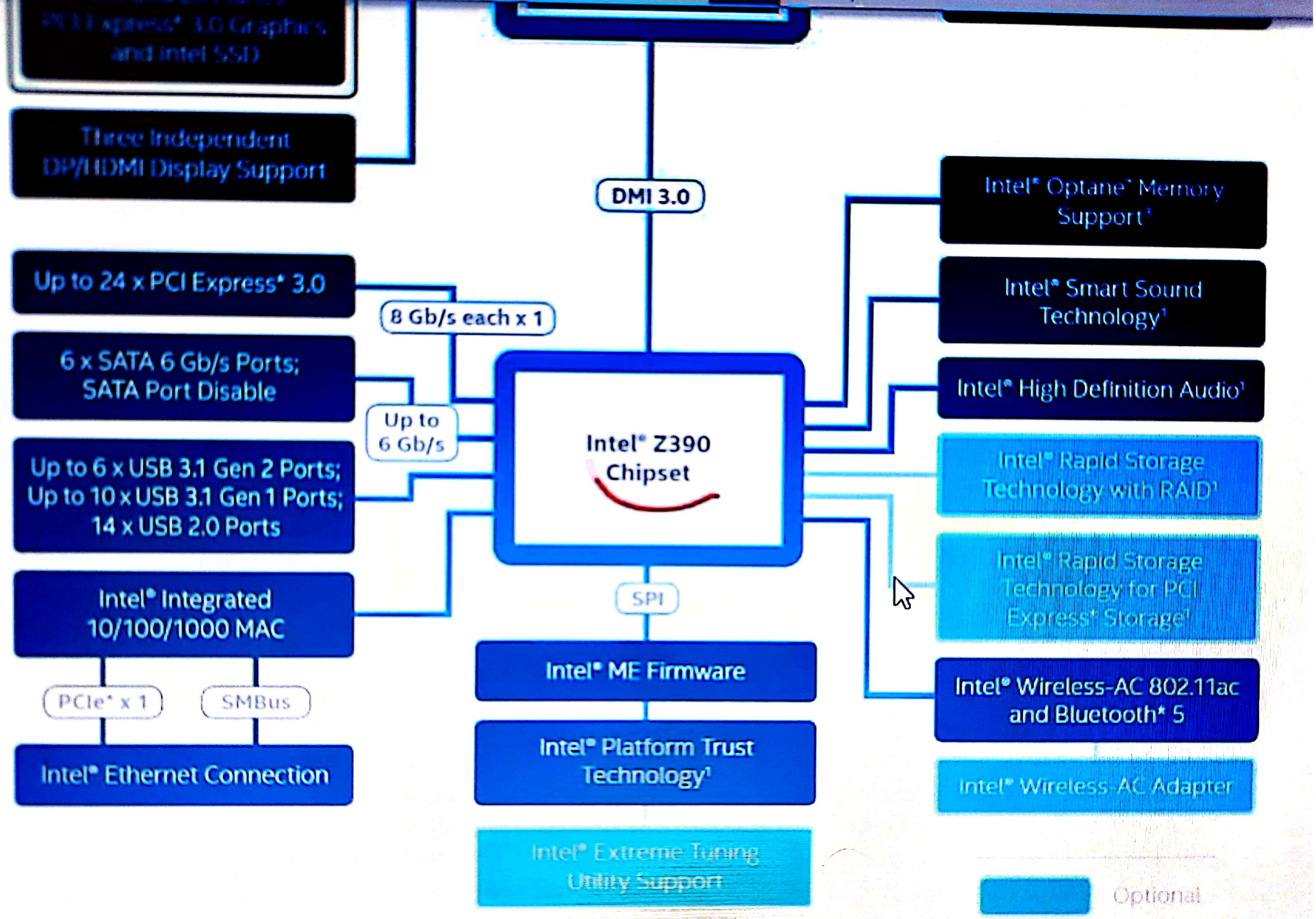
Intel® Optane™ Memory Support[3]

Intel® Smart Sound Technology[1]

Intel® High Definition Audio[1]

Intel® Rapid Storage

Chapter 6 — Parallel Processors



PCI Express* 3.0 Graphics and Intel SSD

Three Independent DP/HDMI Display Support

DMI 3.0

Up to 24 x PCI Express* 3.0

8 Gb/s each x 1

6 x SATA 6 Gb/s Ports; SATA Port Disable

Up to 6 Gb/s

Up to 6 x USB 3.1 Gen 2 Ports; Up to 10 x USB 3.1 Gen 1 Ports; 14 x USB 2.0 Ports

Intel® Integrated 10/100/1000 MAC

PCIe* x 1

SMBus

Intel® Ethernet Connection

Intel® Z390 Chipset

SPI

Intel® ME Firmware

Intel® Platform Trust Technology[1]

Intel® Extreme Tuning Utility Support

Intel® Optane* Memory Support[1]

Intel® Smart Sound Technology[1]

Intel® High Definition Audio[1]

Intel® Rapid Storage Technology with RAID[1]

Intel® Rapid Storage Technology for PCI Express* Storage[1]

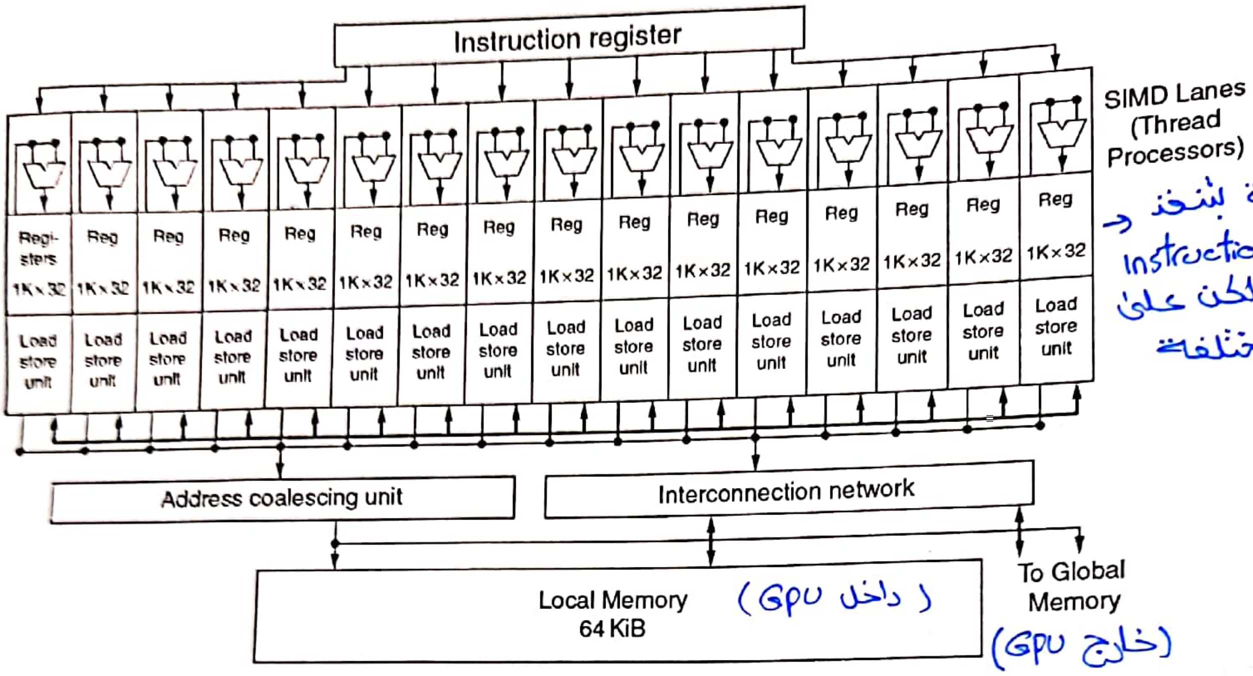
Intel® Wireless-AC 802.11ac and Bluetooth* 5

Intel® Wireless-AC Adapter

▨ Optional

Example: NVIDIA Fermi

- Multiple SIMD processors, each as shown:



→ كل وحدة بتنفذ منهم instruction نفسها لكن على Data مختلفة

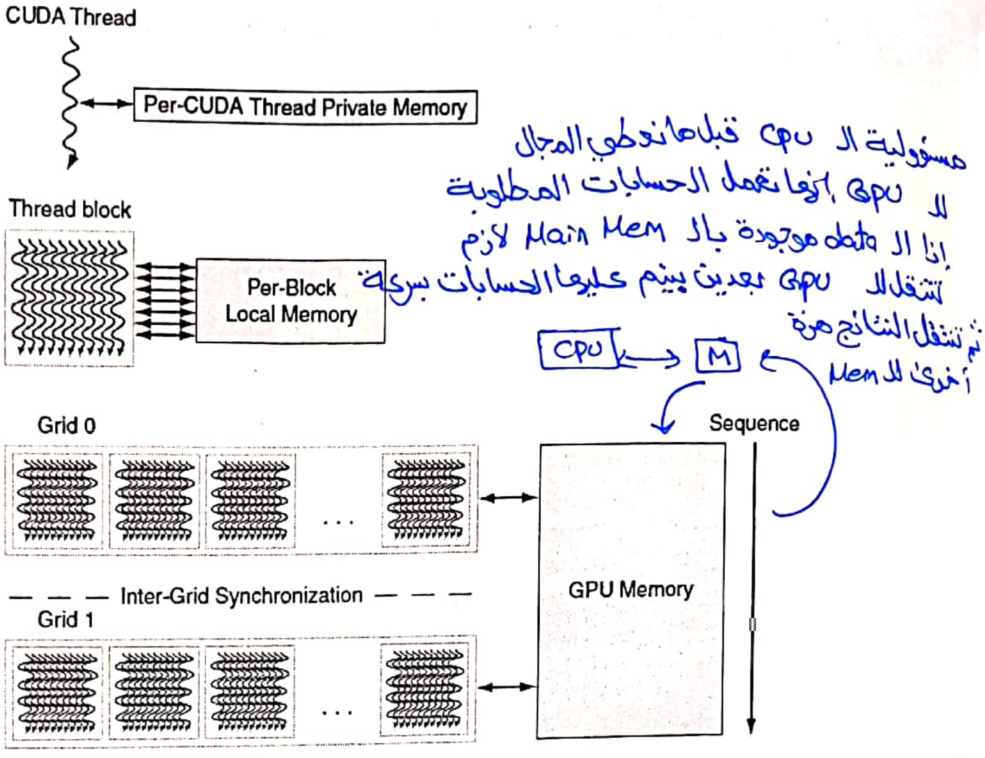
(داخل GPU)

(خارج GPU)

Example: NVIDIA Fermi

- SIMD Processor: 16 SIMD lanes
- SIMD instruction
  - Operates on 32 element wide threads
  - Dynamically scheduled on 16-wide processor over 2 cycles
- 32K x 32-bit registers spread across lanes
  - 64 registers per thread context

GPU Memory Structures

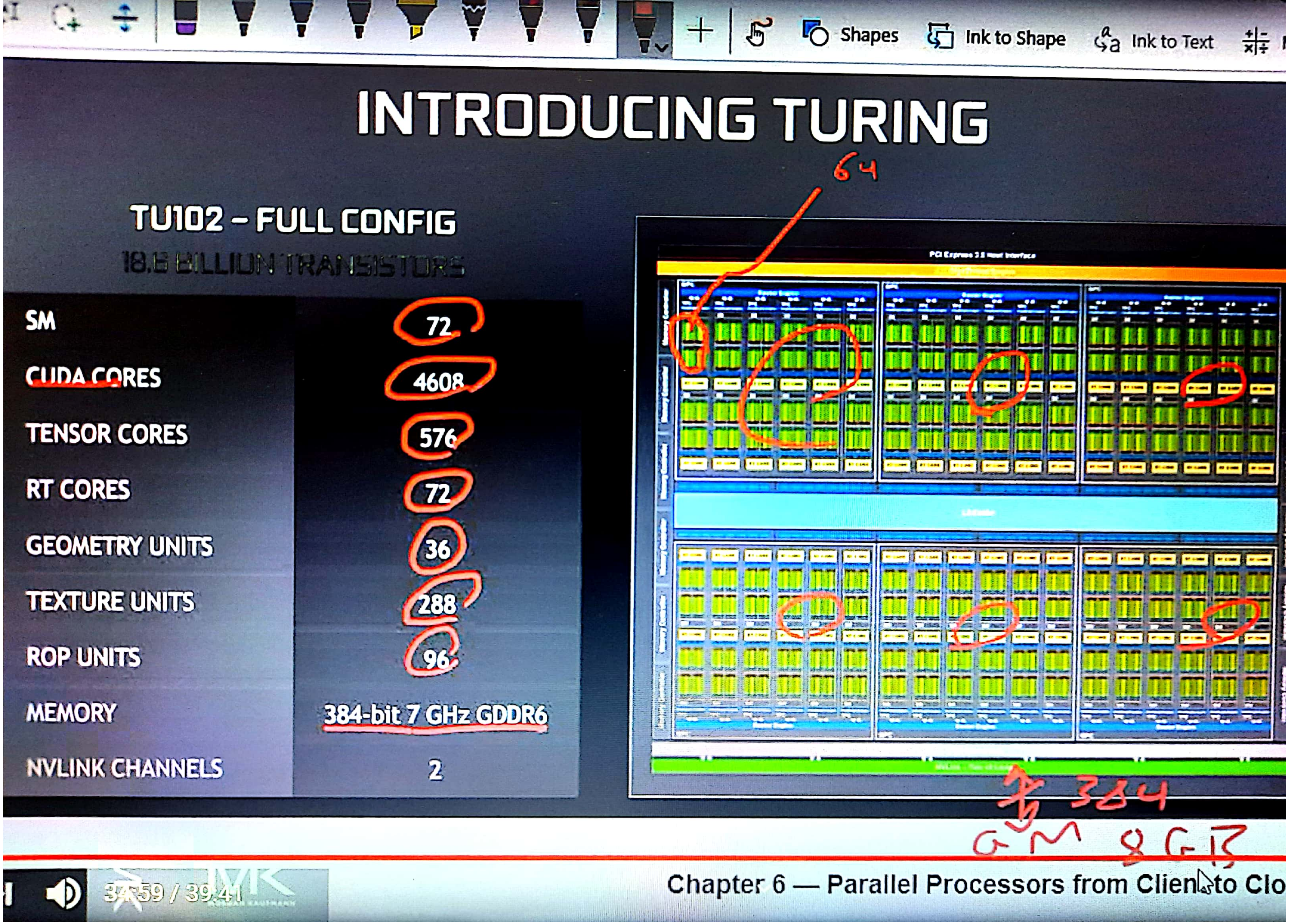


مسؤولية ال cpu قبل ما نعطي المجال
لل GPU إنها تعمل الحسابات المطلوبة
إنا ال data موجودة بال Main Mem لازم
تنقل لل GPU بعدين يتم عليها الحسابات بسرعة
ثم تنقل النتائج مرة
أخرى لل Mem

Classifying GPUs

- Don't fit nicely into SIMD/MIMD model
  - Conditional execution in a thread allows an illusion of MIMD
    - But with performance degredation
    - Need to write general purpose code with care

	Static: Discovered at Compile Time	Dynamic: Discovered at Runtime
Instruction-Level Parallelism	VLIW	Superscalar
Data-Level Parallelism	SIMD or Vector	Tesla Multiprocessor GPUs



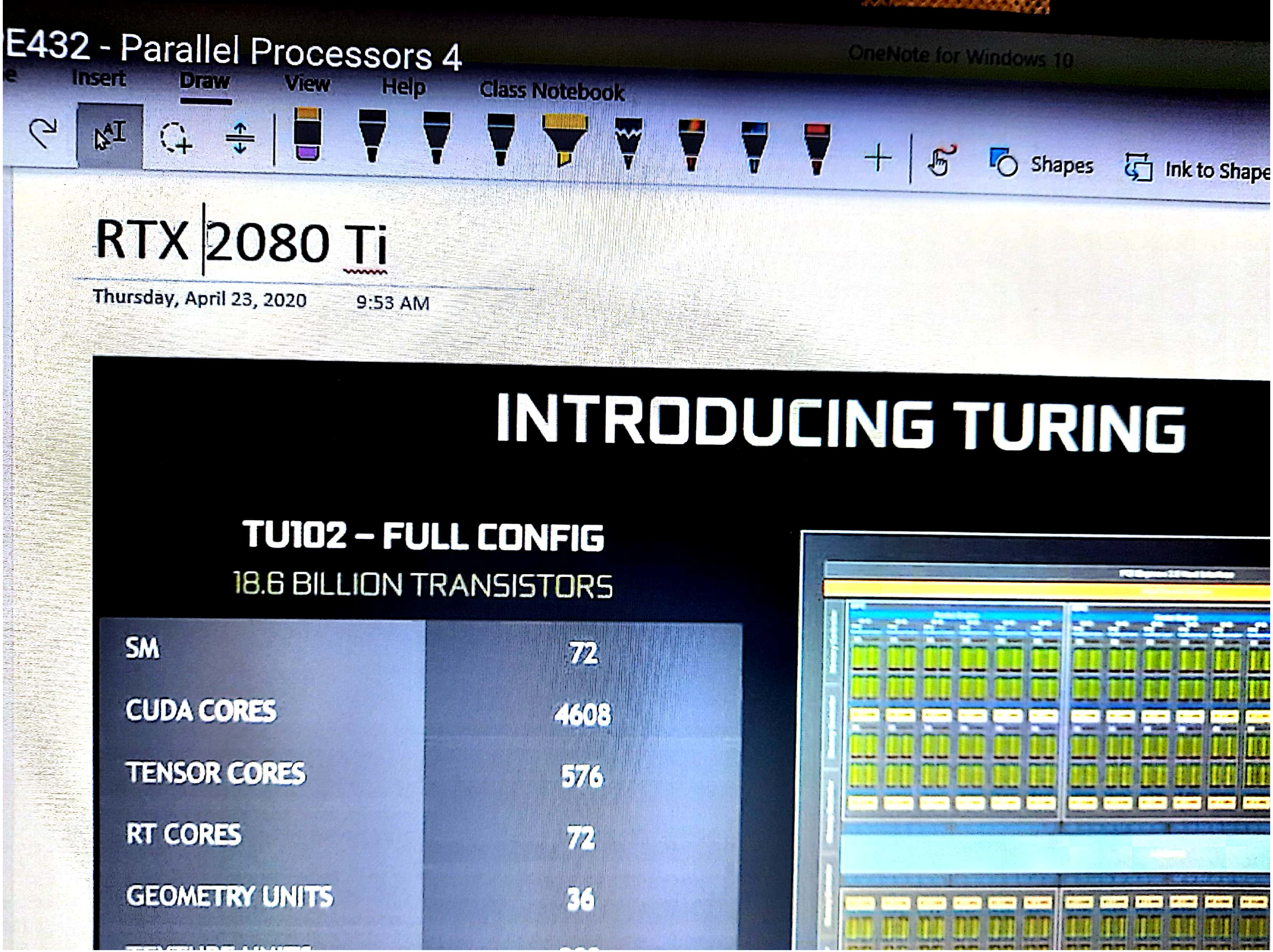
INTRODUCING TURING

TU102 – FULL CONFIG

18.6 BILLION TRANSISTORS

SM	72
CUDA CORES	4608
TENSOR CORES	576
RT CORES	72
GEOMETRY UNITS	36
TEXTURE UNITS	288
ROP UNITS	96
MEMORY	384-bit 7 GHz GDDR6
NVLINK CHANNELS	2

Chapter 6 — Parallel Processors from Clien to Clo



RTX 2080 Ti

Thursday, April 23, 2020 9:53 AM

INTRODUCING TURING

TU102 – FULL CONFIG

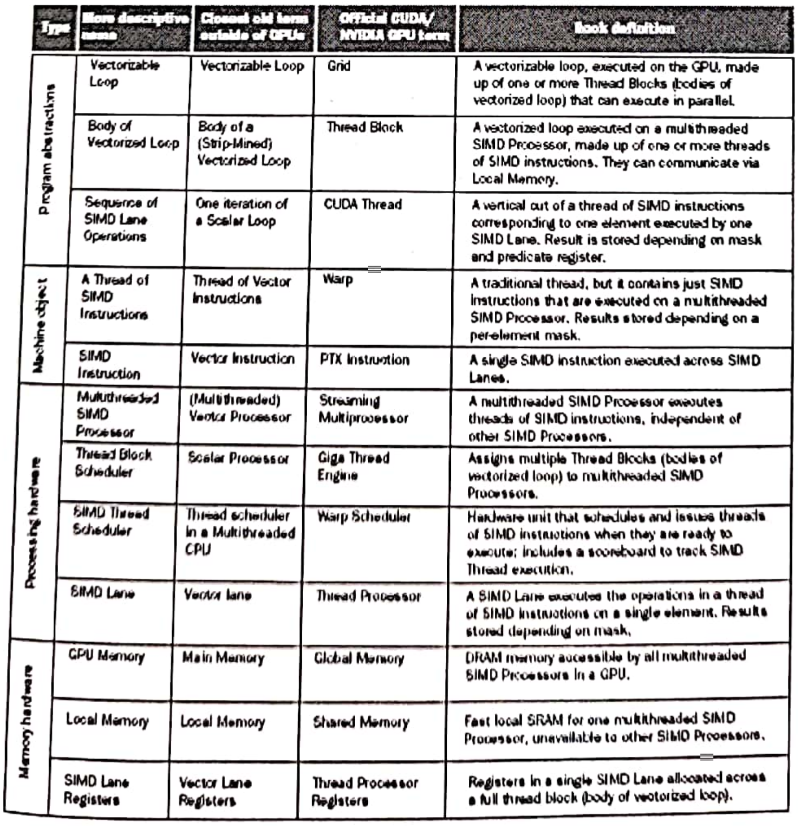
18.6 BILLION TRANSISTORS

SM	72
CUDA CORES	4608
TENSOR CORES	576
RT CORES	72
GEOMETRY UNITS	36
TEXTURE UNITS	[illegible]

Putting GPUs into Perspective

Feature	CPU Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16 or 32 or 64
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Typical ratio of single precision to double-precision performance	2:1	2:1
Largest cache size	8 MB	0.75 MB
Size of memory address	64-bit	64-bit
Size of main memory	8 GB to 256 GB	4 GB to 6 GB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Integrated scalar processor/SIMD processor	Yes	No
Cache coherent	Yes	No

Guide to GPU Terms

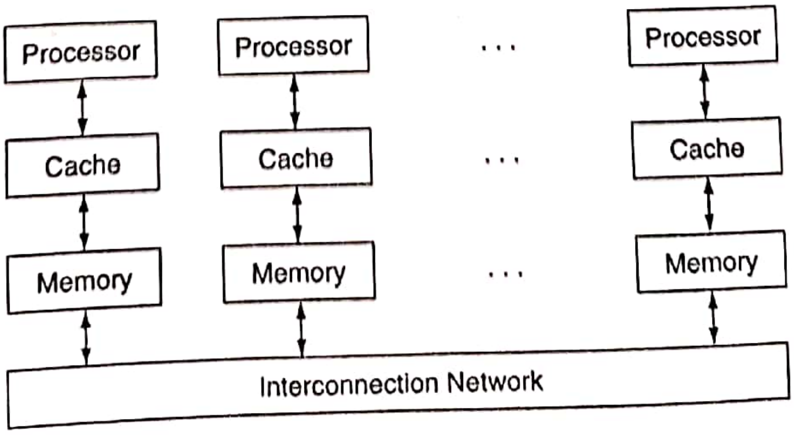


Type	More descriptive name	Closest old term outside of GPUs	Official CUDA/NVIDIA GPU term	Book definition
Program abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel.
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via Local Memory.
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register.
Machine object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it contains just SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask.
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes.
Processing hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors.
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors.
	SIMD Thread Scheduler	Thread scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution.
	SIMD Lane	Vector lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask.
Memory hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU.
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors.
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full thread block (body of vectorized loop).

Contents

Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors



ما بتعاونوا عن طريق ال Memory →

Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet → بطيء.
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
  ↪ اذا واحد تعطل البقية بكملوا.
- Problems
  - Administration cost (prefer virtual machines) → يستخدموه معها عشان يسهل
  - Low interconnect bandwidth
    - عكس ↓ c.f. processor/memory bandwidth on an SMP ↩ هدول أفضل

Sum Reduction (Again)

- Sum 64,000 on 64 processors → موزعة 64000
  64P كل P بياخذ 1000
- First distribute 1000 numbers to each
  - The do partial sums

```
sum = 0;
for (i = 0; i<1000; i += 1)
  sum += AN[i];
```

- Reduction
  - Half the processors send, other half receive and add
  - The quarter send, quarter receive and add, …

Sum Reduction (Again)

- Given send() and receive() operations

```
limit = 64; half = 64;/* 64 processors */
do
  half = (half+1)/2; /* send vs. receive
                        dividing line */
  if (Pn >= half && Pn < limit)
    send(Pn - half, sum);
  if (Pn < (limit/2))
    sum += receive();
  limit = half; /* upper limit of senders */
while (half > 1); /* exit with final sum */
```

هاذ الجديد المهم

- Send/receive also provide synchronization
- Assumes send/receive take similar time to addition

Grid Computing

- Separate computers interconnected by long-haul networks
  - E.g., Internet connections
  - Work units farmed out, results sent back
- Can make use of idle time on PCs
  - E.g., SETI@home, World Community Grid

* يعني لو الجهاز مو بستخدمه بقدر أتيحه عالانترنت لحد بره بيستخدمه *

Contents

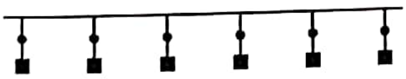
تشبيك ال Processors أو ال computers مع بعض

Interconnection Networks

*فائدة ال Torus 2D عن ال Mesh اللي بالصورة
انه بسرع عملية التواصل لانه تشبك الأطراف مع بعض
و تقلل ال latency .

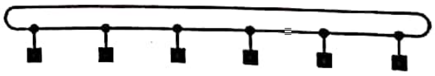
- Network topologies
  - Arrangements of processors, switches, and links

1



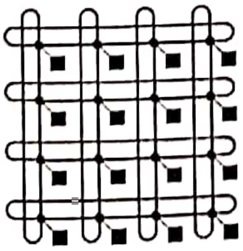
Bus

2



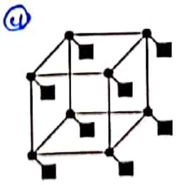
Ring

3



2D Mesh
(Torus)

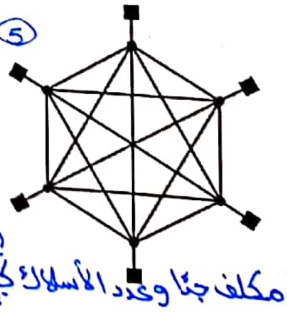
4



N-cube (N = 3)

dimension = log N
latency = diminsion

5



$\frac{N(N-1)}{2}$ مشكلته مكلف جدا وعدد الأسلاك كبير

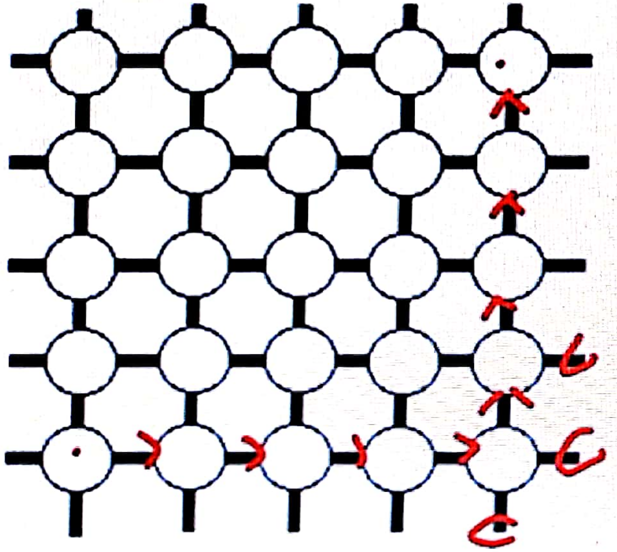
Fully connected

latency = 1

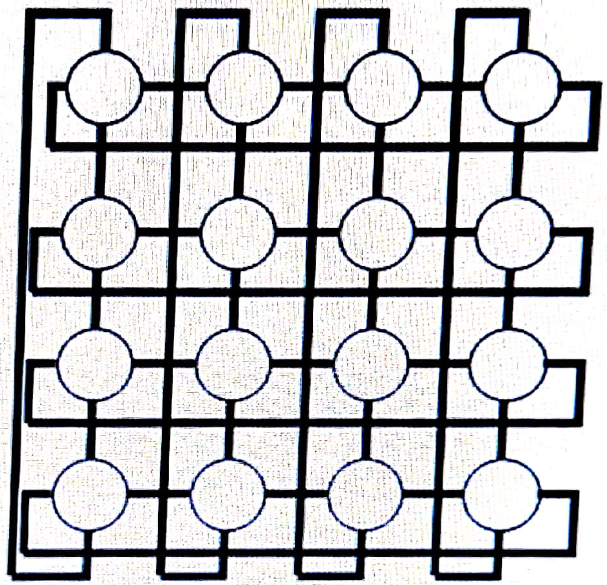
كل نقطة مشبوكة مع الكل

Interconnection Networks

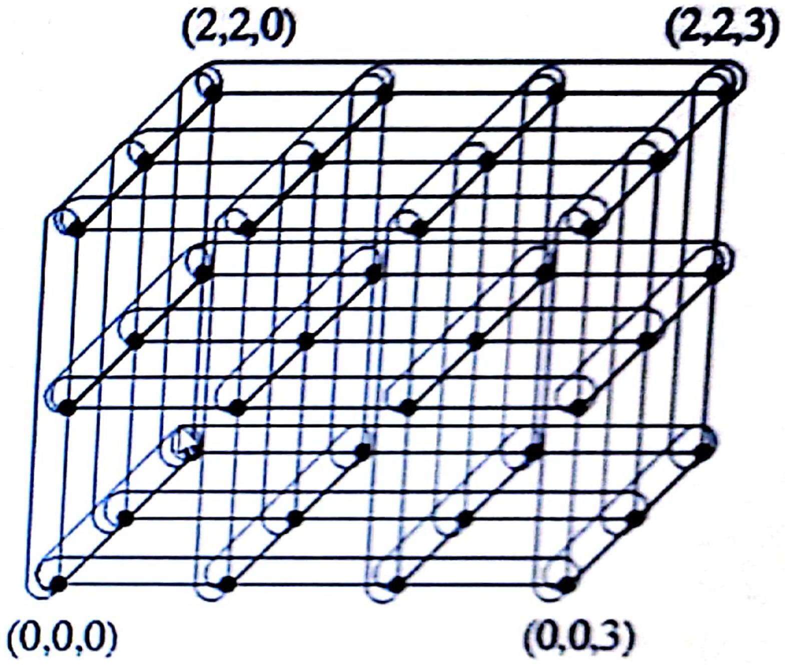
Sunday, April 26, 2020 10:02 AM



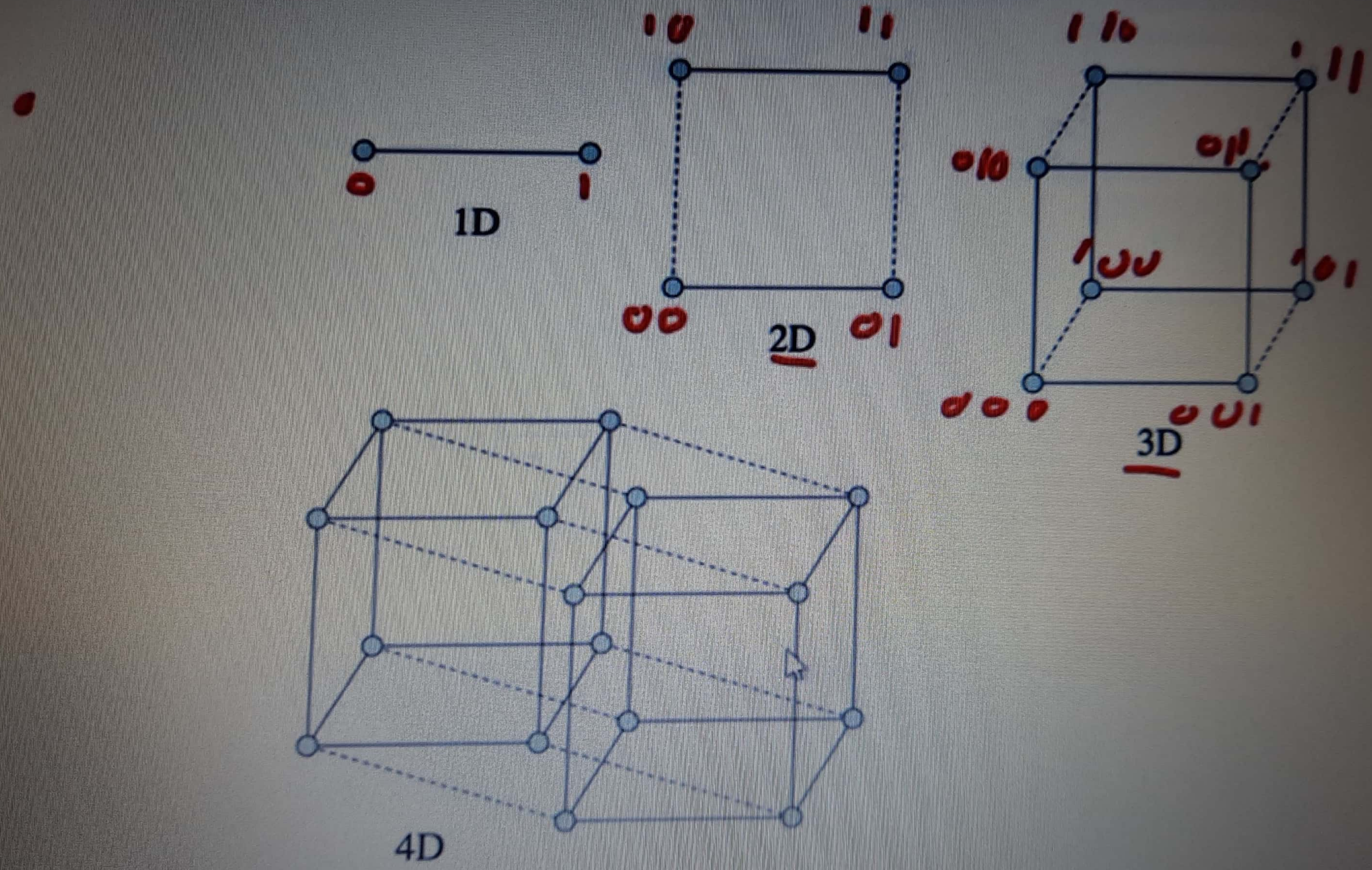
Mesh Topology



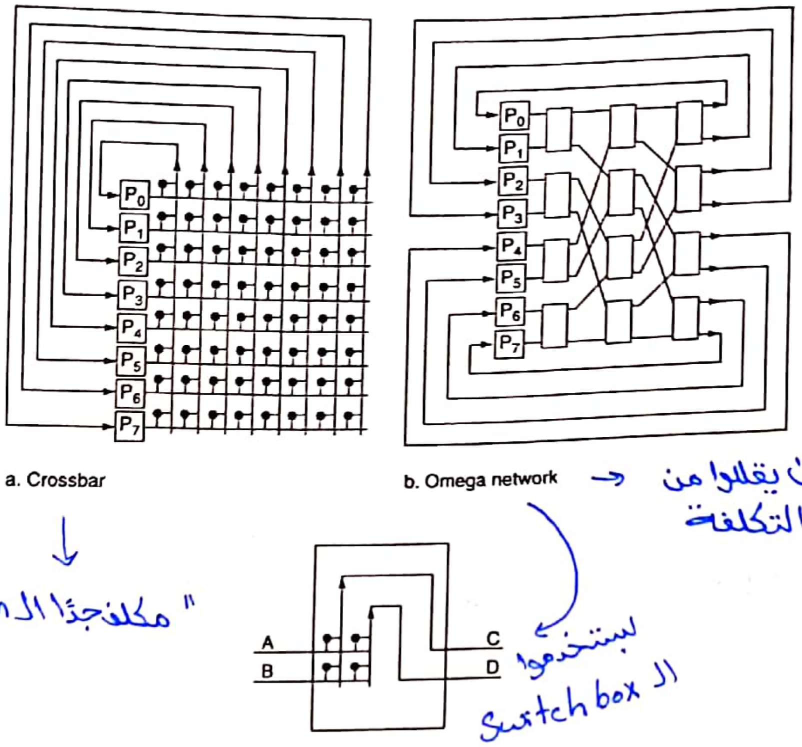
2D Torus



3D Torus


0
0
1
1D
10
11
00
01
2D
110
111
010
011
100
101
000
001
3D
4D

Multistage Networks



a. Crossbar

" مكلفة جدًا ال Switch غالي "

b. Omega network → لحتى يقللوا من التكلفة

بستخدموا ال Switch box

c. Omega network switch box

Network Characteristics

- Performance
  - Latency per message (unloaded network) (أقل ما يمكن : بس مش ع حساب ال cost
  - Throughput
    - Link bandwidth → بعتمد على عدد الأسلاك وال Frequency
    - Total network bandwidth
    - Bisection bandwidth → لما أقسمهم لنصين كم link بضل عندي

    بوخذ على G

    أعلى ما يمكن بس مش ع حساب ال cost
  - Congestion delays (depending on traffic)
- Cost (معقولة)
- Power
- Routability in silicon

8 8

$d = \lg_2 64 = 6$

$\frac{64 \times 63}{2}$

?

Evaluation category	Bus	Ring	2D mesh	2D torus	Hypercube	Fat tree	Fully connected
Performance							
BW$_{\text{Bisection}}$ in # links	1	2	8	16	32	32	1024
Max (ave.) hop count	1 (1)	32 (16)	14 (7)	8 (4)	6 (3)	11 (9)	1 (1)
Cost							
I/O ports per switch	NA	3	5	5	7	4	64
Number of switches	NA	64	64	64	64	192	64
Number of net. links	1	64	112	128	192	320	2016
Total number of links	1	128	176	192	256	384	2080

Figure F.15 Performance and cost of several network topologies for 64 nodes. The bus is the standard reference at unit network link cost and bisection bandwidth. Values are given in terms of bidirectional links and ports. Hop count includes a switch and its output link, but not the injection link at end nodes. Except for the bus, values are given for the number of network links and total number of links, including injection/reception links between end node devices and the network.

PAGE
DATE

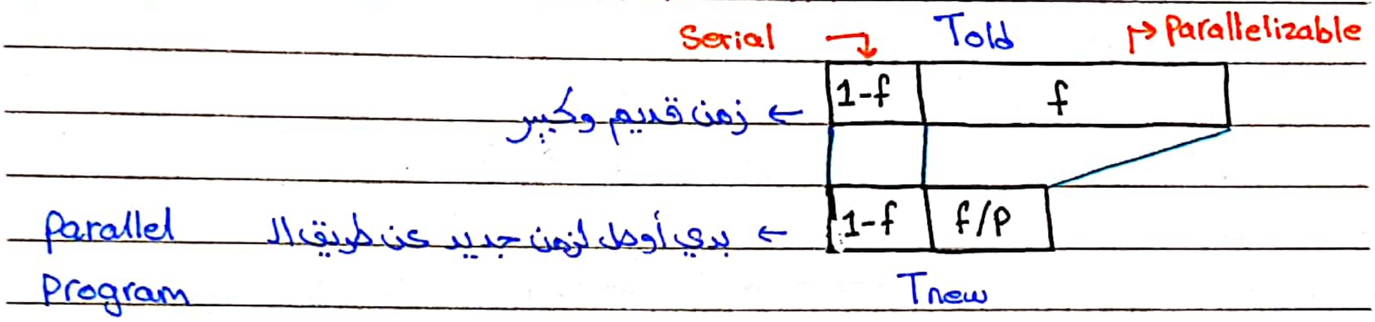
CH6 :

Slide 9 : Amdahl's law :

$T_{old} = T_{old} * (1-f+f)$

$T_{new} = T_{old} * (1-f+f/p)$

Speed up $= T_{old} / T_{new} = 1/(1-f+f/p)$ → عدد الprocessors



* Serial : لابد انك تنفذه مابتقدر تعدله

* parallelizable : ممكن انو نحوله لانه بينفذه أكثر من P

$\therefore \lim_{p\to\infty}$ speed up $= \frac{1}{1-f}$ → أعلى Speedup ممكن تحصله

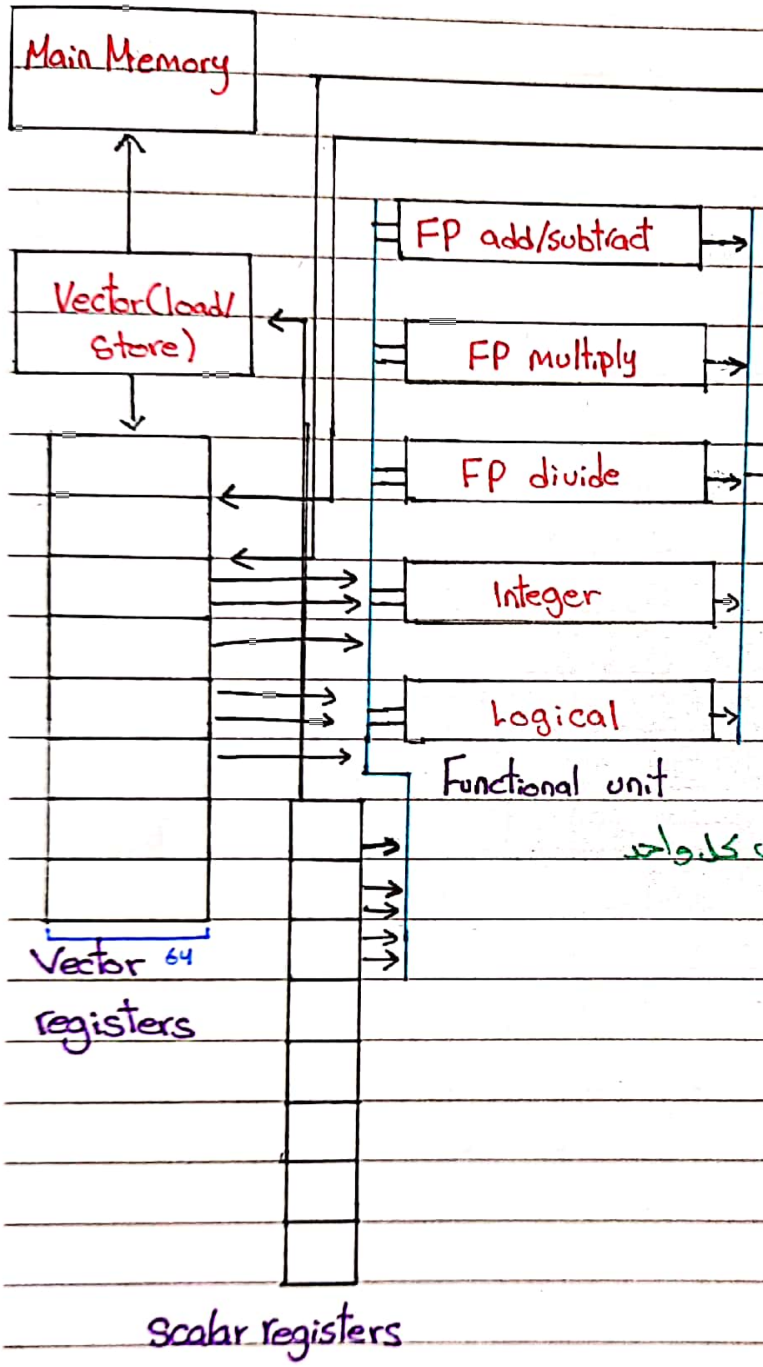
Slide 11 : StrongScalling (تخلي المسألة ثابتة وتزيد عدد ال Processors .)

Size			P=1		P=10		P=100	
10+ 100	Speedup	Efficiency	1	100%	5.5	55%	10	10%
10+ 1000	Speedup	Efficiency	1	100 %	9.2	92 %	51	51%
10+ 10000	Speedup	Efficiency	1	100%	9.9	99%	91	91%

→ اضافة من الدكتور,

* بنلاحظ كل ما زاد عدد ال Processors ال Efficiency بتقل لانو المسألة ثابتة.

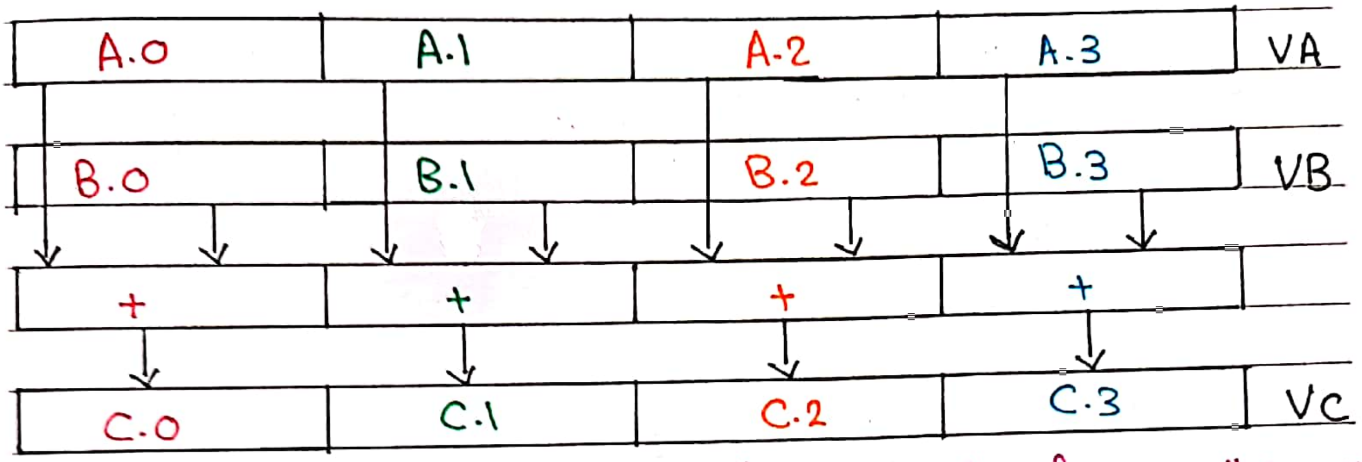
Slide 16 : Vector processor (رسمة توضيحية)



* المهم هنا إنه داخل
الـ Vector computer
في Vector registers
وهدول الـ Vectors ممكن
يكون فيه منهم 32 لكن كل
واحد منهم
Vector register مش بيتسع
لـ 1 value بيتسع لـ
Vector كامل فكل واحد
من الـ Vector register ممكن يكون
بيتكون من 64 (64 register)
=> 32 VR x 64 = 2K reg
(مشان هيك هو غالي)

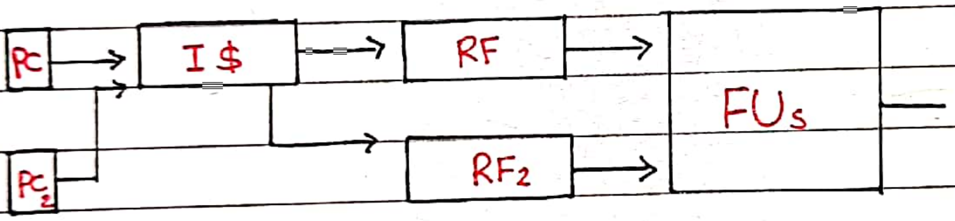
* Scalar ممكن يكون 32 واحد بس كل واحد
Registe : فيه 1 value

Slide 19 : SIMD (add VC, VA, VB)

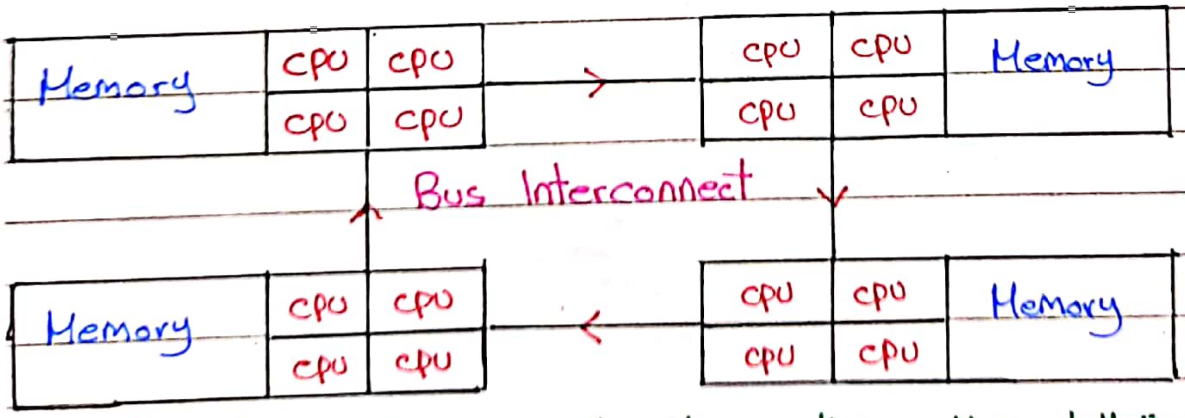


* يحسن ال performence لأنه بـ 1 instruction بيعمل 4 additions .

Slide 22 : Multithreading



Slide 27 : NUMA



* كل مجموعة إلها جزء خاص من ال Memory و ال 4 Memories متشاركين وأي cpu بشوف الباقي لكن ال Memory اللي في 4 cpus قريبة أكثر من باقي ال Memory وبوصلولها بشكل أسرع .

PAGE

DATE

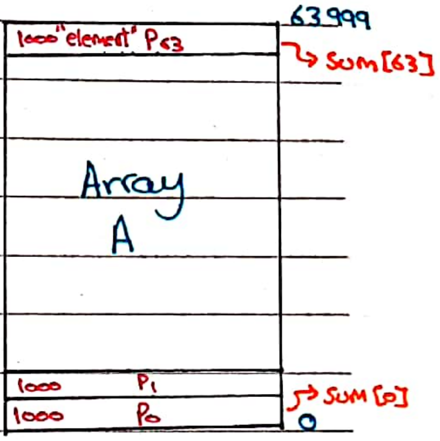
Slide 28 : Sum Reduction

Serial :

```
Sum = 0;
  for ( i=0; i<64000; i+=1)
        Sum += A[i];
```

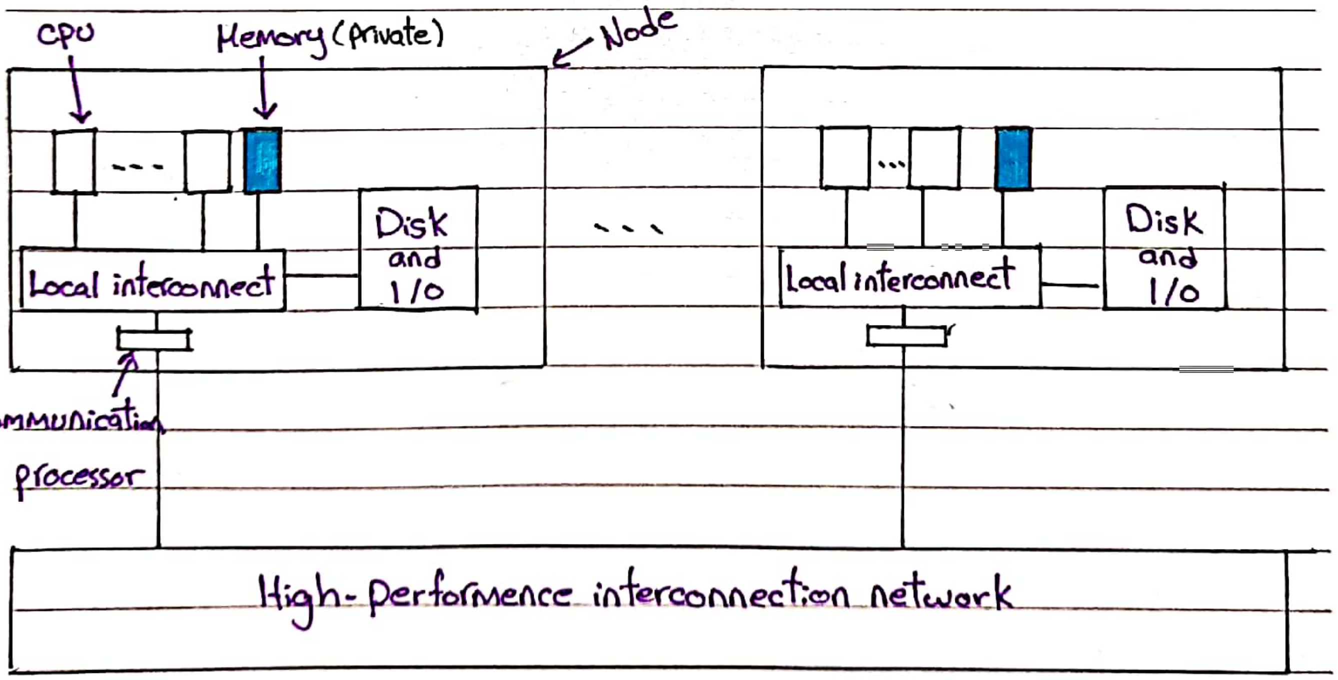
Parallel :

```
Sum [Pn] = 0;
  for (i=1000*Pn; i<1000*(Pn+1); i+=1)
      Sum[Pn] += A[i];
```



عشان نختصر الوقت ، بنقسم كل الشغل ـها
على (Processor 64) . حنخلص أقل بكثير من الـ Serial
بطلعلي 64 جواب و بعد بجمعهم كلهم برقم واحد
وبدل ما أعمل loop عشان أجمع الـ 64 نعمل اللي بسلايد 29

Slide 41 : Message-Passing Multicomputers



Scanned with CamScanner

Parallel Benchmarks

نشغلهم عتصاميم مختلفة تقلنا
مين الأحسن

- *Linpack*: matrix linear algebra
- *SPECrate*: parallel run of SPEC CPU programs
  - Job-level parallelism
- SPLASH: Stanford Parallel Applications for Shared Memory
  - Mix of kernels and applications, strong scaling
- NAS (NASA Advanced Supercomputing) suite
  - computational fluid dynamics kernels
- PARSEC (Princeton Application Repository for Shared Memory Computers) suite
  - Multithreaded applications using Pthreads and OpenMP

لغات
برمجة.

Code or Applications?

- Traditional benchmarks
  - Fixed code and data sets
- Parallel programming is evolving
  - Should algorithms, programming languages, and tools be part of the system?
  - Compare systems, provided they implement a given application
  - E.g., Linpack, Berkeley Design Patterns
- Would foster innovation in approaches to parallelism

بقوي الابتكار بالطرق اللي إلها علاقة بال Parallelism

الهدف نقيم الPerformance ونعرف هل هو كويس ولا محتاج اكتب برنامج أحسن

Modeling Performance

- Assume performance metric of interest is achievable GFLOPs/sec

قديه الcomputer بقدر يعمل Floating point operations ومش دايما هو المعيار الوحيد بس هو الأهم هون

  - Measured using computational kernels from Berkeley Design Patterns
- Arithmetic intensity of a kernel
  - FLOPs per byte of memory accessed

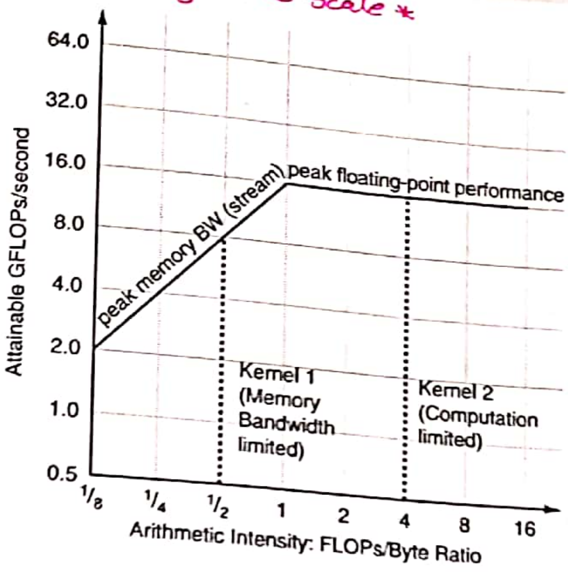
مهمة للتمييز بين ال Kernel المختلفات

- For a given computer, determine
  - Peak GFLOPS (from data sheet)
  - Peak memory bytes/sec (using Stream benchmark)

لو شغلنا كل ال ALUs بهذا ال computer قديش ممكن نعمل GFlops

Roofline Diagram

* logarithmic Scale *



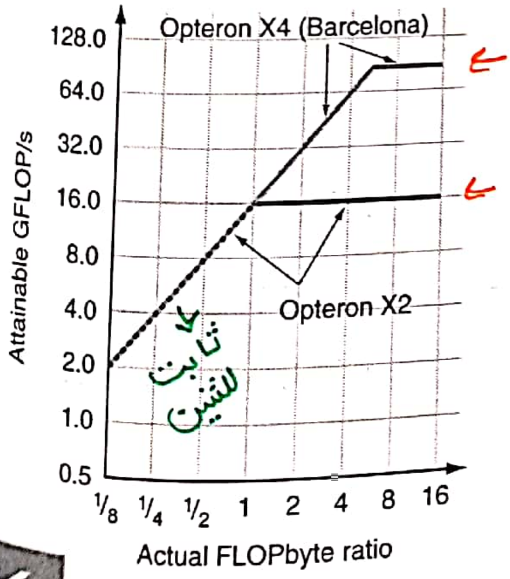
Attainable GPLOPs/sec
= Max (Peak Memory BW × Arithmetic Intensity, Peak FP Performance)

Comparing Systems

- Example: Opteron X2 vs. Opteron X4
  - 2-core vs. 4-core, 2× FP performance/core, 2.2GHz vs. 2.3GHz, 1 × 2 SIMD vs. 2 × 2 SIMD

لـ كل cycle بنفذ فيها 1 SIMD بعمليتين ↳ بعمل 2 SIMD op كل cycle
كل وحدة فيها عمليتين

  - Same memory system



- To get higher performance on X4 than X2
  - Need high arithmetic intensity
  - Or working set must fit in X4's 2MB L-3 cache

opteron X2 : Peak FP Performance = 2 cores/chip * 2 SIMD/core * 2 FP/SIMD * 2.2 Gcycle/s = 2*2*2*2.2 = 17.6 GFlops/s

opteron X4 , Peak FP Performance = 4 cores/chip * 4 SIMD/core * 2FP/SIMD * 2.3 Gcycles/s = 4*4*2*2.3 = 73.6 GFlops/s

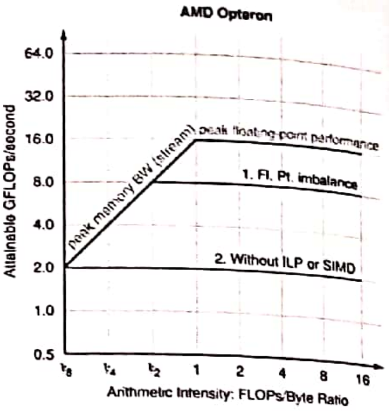
Optimizing Performance

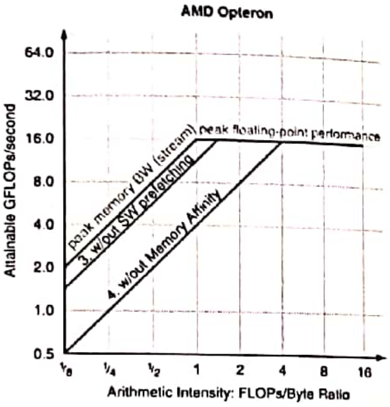
- Optimize FP performance
  - Balance adds & multiplies
  - Improve superscalar ILP and use of SIMD instructions
- Optimize memory usage
  - Software prefetch
    - Avoid load stalls
  - Memory affinity
    - Avoid non-local data accesses

حلول لتوصل
Performanceل
الأفضل

Instruction
level
Parallelism

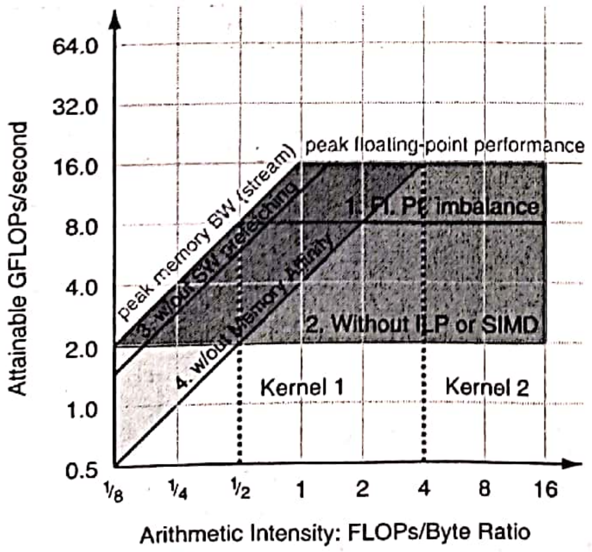
الحسبة تاعتك
تمشي عالسطر كامل
قبل ما تشغل
للعامود
الي بعده





Optimizing Performance

- Choice of optimization depends on arithmetic intensity of code



- Arithmetic intensity is not always fixed
  - May scale with problem size
  - Caching reduces memory accesses
    - Increases arithmetic intensity

Contents

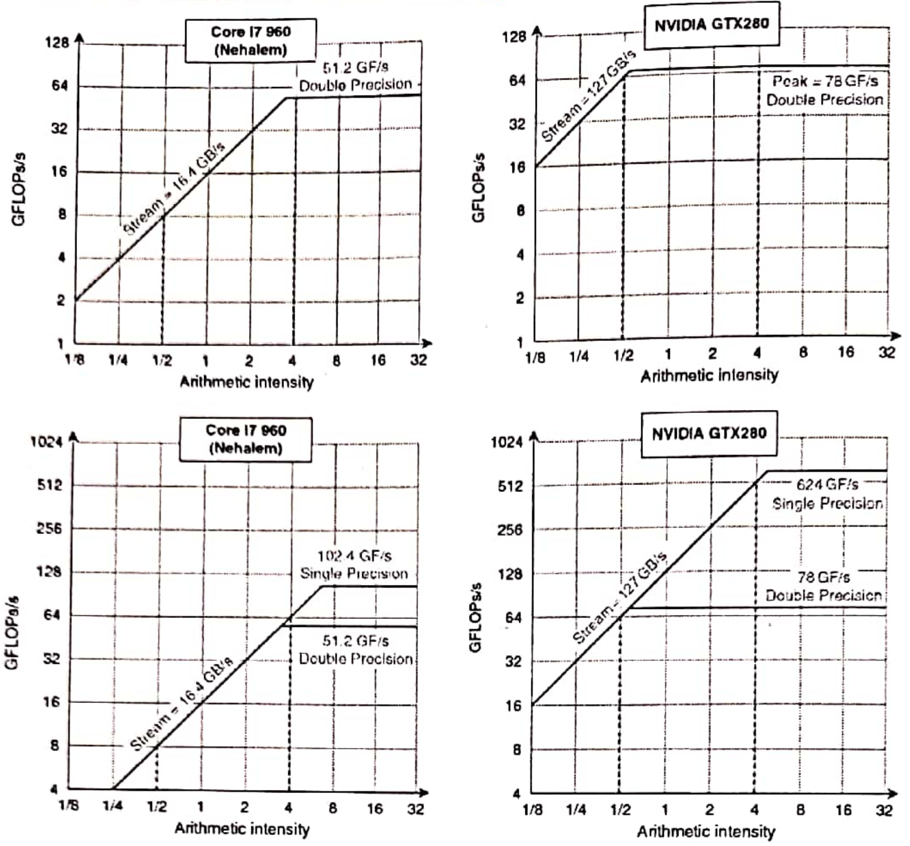
i7-960 vs. NVIDIA Tesla 280/480

§6.11 Real Stuff: Benchmarking and Rooflines i7 vs. Tesla

	CPU Core i7-960	GPU GTX 280	GPU GTX 480	Ratio 280/i7	Ratio 480/i7
Number of processing elements (cores or SMs)	4	30	15	7.5	3.8
Clock frequency (GHz)	3.2	1.3	1.4	0.41	0.44
Die size	263	576	520	2.2	2.0
Technology	Intel 45 nm	TCMS 65 nm	TCMS 40 nm	1.6	1.0
Power (chip, not module)	130	130	167	1.0	1.3
Transistors	700 M	1400 M	3100 M	2.0	4.4
Memory brandwith (GBytes/sec)	32	أسرع 141	177	4.4	5.5
Single frecision SIMD width	4	8	32	2.0	8.0
Dobule precision SIMD with	أحسن 2	أسوأ 1	16	0.5	8.0
Peak Single frecision scalar FLOPS (GFLOP/sec)	26	117	63	4.6	2.5
Peak Single frecision s SIMD FLOPS (GFLOP/Sec)	102	311 to 933	515 to 1344	3.0-9.1	6.6-13.1
(SP 1 add or multiply)	N.A.	(311)	(515)	(3.0)	(6.6)
(SP 1 instruction fused)	N.A	(622)	(1344)	(6.1)	(13.1)
(face SP dual issue fused)	N.A	(933)	N.A	(9.1)	-
Peal double frecision SIMD FLOPS (GFLOP/sec)	51	78	515	1.5	10.1

Rooflines

high performence أسهل تحصل



Benchmarks

Kernel	Units	Core I7-960	GTX 280	GTX 280/ I7-960
SGEMM	GFLOP/sec	94	364	3.9
MC	Billion paths/sec	0.8	1.4	1.8
Conv	Million pixels/sec	1250	3500	2.8
FFT	GFLOP/sec	71.4	213	3.0
SAXPY	GBytes/sec	16.8	88.8	5.3
LBM	Million lookups/sec	85	426	5.0
Solv	Frames/sec	103	52	0.5
SpMV	GFLOP/sec	4.9	9.1	1.9
GJK	Frames/sec	67	1020	15.2
Sort	Million elements/sec	250	198	0.8
RC	Frames/sec	5	8.1	1.6
Search	Million queries/sec	50	90	1.8
Hist	Million pixels/sec	1517	2583	1.7
Bilat	Million pixels/sec	83	475	5.7

Performance Summary

- GPU (480) has 4.4 X the memory bandwidth
  - Benefits memory bound kernels
- GPU has 13.1 X the single precision throughout, 2.5 X the double precision throughput
  - Benefits FP compute bound kernels
- CPU cache prevents some kernels from becoming memory bound when they otherwise would on GPU
- GPUs offer scatter-gather, which assists with kernels with strided data
- Lack of synchronization and memory consistency support on GPU limits performance for some kernels

Contents

- 6.1 Introduction
- 6.2 The Difficulty of Creating Parallel Programs
- 6.3 SISD, MIMD, SIMD, SPMD, and Vector
- 6.4 Hardware Multithreading
- 6.5 Shared Memory Multiprocessors
- 6.6 Introduction to Graphics Processing Units
- 6.7 Clusters and Message-Passing Multiprocessors
- 6.8 Introduction to Multiprocessor Network Topologies
- 6.10 Multiprocessor Benchmarks and Performance Models
- 6.11 Benchmarking Intel Core i7 960 and NVIDIA Tesla GPU
- X 6.12 Multiple Processors and Matrix Multiply مش مطلوب
- 6.13 Fallacies and Pitfalls
- 6.14 Concluding Remarks

Fallacies

- Amdahl's Law doesn't apply to parallel computers
  - Since we can achieve linear speedup
  - But only on applications with weak scaling
- Peak performance tracks observed performance
  - Marketers like this approach!
  - But compare Xeon with others in example
  - Need to be aware of bottlenecks

§6.13 Fallacies and Pitfalls

Pitfalls

- Not developing the software to take account of a multiprocessor architecture
  - Example: using a single lock for a shared composite resource
    - Serializes accesses, even if they could be done in parallel
    - Use finer-granularity locking

Contents

6.1 Introduction
6.2 The Difficulty of Creating Parallel Programs
6.3 SISD, MIMD, SIMD, SPMD, and Vector
6.4 Hardware Multithreading
6.5 Shared Memory Multiprocessors
6.6 Introduction to Graphics Processing Units
6.7 Clusters and Message-Passing Multiprocessors
6.8 Introduction to Multiprocessor Network Topologies
6.10 Multiprocessor Benchmarks and Performance Models
6.11 Benchmarking Intel Core i7 960 and NVIDIA Tesla GPU
6.12 Multiple Processors and Matrix Multiply
6.13 Fallacies and Pitfalls
6.14 Concluding Remarks

Concluding Remarks

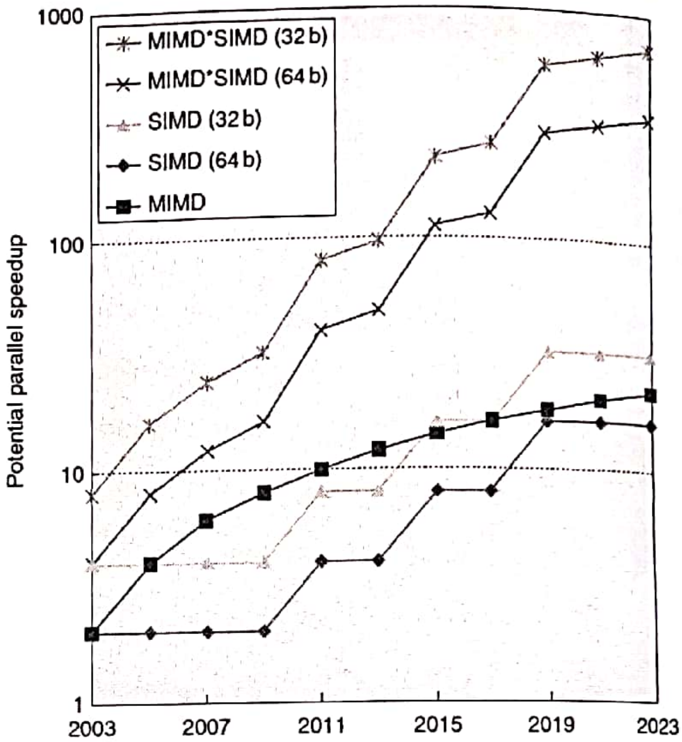
§6.14 Concluding Remarks

- Goal: higher performance by using multiple processors
- Difficulties
  - Developing parallel software
  - Devising appropriate architectures
- SaaS importance is growing and clusters are a good match
- Performance per dollar and performance per Joule drive both mobile and WSC

Concluding Remarks (con't)

- SIMD and vector operations match multimedia applications and are easy to program

- Adding 2 cores/chip every 2 years.
- Doubling SIMD operations every 4 years.



Slide 53 : Roofline Model

DAXPY :

```
for (i=0; i<N; i+=1)
    Y[i] += S* X[i]+Y[i];
      ↓       ↑   ↑    ↑   ↑
```

∴ Arithmetic Intensity = 2 Flops / (2 loads + 1 store)

= 2 / (3*8) = 0.0833 Flop / byte

8 byte كل واحد

(Memory intensive قليل فوائا)

Scanned with CamScanner