درس معماری کامپیوتر پیشرفته

دکتر غلامی

پنجشنبه - ۹۳/۱۱/۳۰

مرجع: معماری کامپیوتر پترسون

۶ نمره ارائه
۱۴ نمره امتحان پایانی
۲ نمره ارائه روش یا ایده جدید

eh.gholami@gmail.com

کارآیی: عبارتست از تعداد پردازش هایی که در واحد زمان انجام می‌شود.

$$performance = \frac{1}{exe\text{-}time}$$

کارآیی و سرعت دو پارامتر جدا از هم هستند که در انتخاب سیستم ها به عنوان اینکه کدامیک بهتر است به کار می روند.

$$Pm_1 = \frac{1}{x} \quad , \quad Pm_2 = \frac{1}{y} \qquad \frac{Pm_1}{Pm_2} = \frac{\frac{1}{x}}{\frac{1}{y}} = \frac{y}{x}$$

(Pm_1: کارآیی ماشین 1)

در بررسی کارآیی بین دو ماشین مشاهده می گردد که نسبت کارآیی آنها به یکدیگر با زمان نسبت عکس دارد.

زمان پاسخ: مدت زمان شروع تا خاتمه اجرای دستور را زمان پاسخ گویند.

واکش (وابسته به حافظه) - رمزگشایی - اجرا (وابسته به پردازنده)

زمان اجرا: فقط شامل مدت زمان اجرای دستور می باشد. به زمان اجرا cpu-time نیز گفته می‌شود.

زمان اجرا به موارد زیادی بستگی دارد مانند نوع مداراتی که برای اجرای آن دستور پیش بینی شده است.

cpu-time (در معماری های پیشرفته)
- زمان کاربری: user cpu-time (معمولاً به صورت نخت افزاری است)
- زمان سیستم: system cpu-time (به صورت نخت افزاری یا نرم افزاری می باشد)

* زمان سیستم برای مشخص کردن اینکه کدامیک از cpu ها وظیفه اجرای این دستور را به عهده بگیرد در نظر گرفته می‌شود.

کاهش هریک از دو زمان فوق (زمان کاربری و زمان سیستم) می تواند باعث افزایش کارآیی سیستم گردد.

معمولاً جهت بررسی کارآیی یک سیستم، یک برنامه خاص را روی آن انجام می دهند. لذا برای بررسی کارآیی یک سیستم مدت زمان اجرای آن برنامه خاص را در نظر می گیرند.

$$\text{CPU time for a program} = \begin{pmatrix}\text{Number of clock for a program}\end{pmatrix} \times \text{Time of each clock}$$

با توجه به آنکه زمان هر کلاک به فرکانس بستگی دارد و عبارتست از $\text{clock time} = \frac{1}{f}$ لذا رابطه فوق را می‌توان به صورت زیر نیز بیان نمود:

$$\text{CPU time for a program} = \begin{pmatrix}\text{Number of clock for a program}\end{pmatrix} \times \frac{1}{F}$$

Clock Rate
یا
clock cycle
یا
مدت زمان یک کلاک

۹۳/۱۲/۱۴

CMSC 611: Advanced Computer Architecture

Performance

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

Integrated Circuits: Fueling Innovation

- Technology innovations over time

Year	Technology used in computers	Relative performance/unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuits	900
1995	Very large-scale integrated circuit	2,400,000

Advances of the IC technology affect H/W and S/W design philosophy

کارآیی به معنای اجرای تراکنش های موفق است . یک تراکنش ممکن است بیش از یک دستور باشد. نکته ای که باید توجه شود آن است که ممکن است تعداد ۱۰۰ دستور مثلاً اجرا شود که مربوط به تراکنش های متفاوت می باشند . در سنجش کارآیی مهم آن است که چه مقدار از تراکنش ها به صورت صحیح انجام شده است .

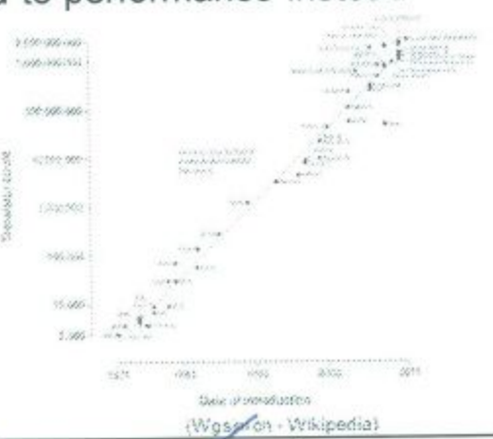
3

Moore's Law

- Transistors double every year
  - Revised to every two years
  - Sometimes revised to performance instead of transistors

Year	Processor	Transistors
1971	4004	2,300
1974	8080	4,500
1978	8086	29,000
1989	i486	1,200,000
1993	Pentium	3,100,000
2000	Pentium 4	42,000,000
2004	Itanium 2	592,000,000

(Source: Intel)



(Wgsimon - Wikipedia)

میزان توان مصرفی در بعضی جاها مانند یک web server هم سنیت و هم صورت نامحدود همواره در اختیار است در صورتی که مثلاً در یک سیستم حسگر محدودیت توان مصرفی وجود دارد .

4

Defining Performance

- Performance means different things to different people, therefore its assessment is subtle

Analogy from the airlines industry:

- How to measure performance for an airplane?
  - Cruising speed (How fast it gets to the destination)
  - Flight range (How far it can reach)
  - Passenger capacity (How many passengers it can carry)

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h)	Passenger throughput (Passenger × m.p.h)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Criteria of performance evaluation differs among users and designers

برای تعریف بازدهی یا throughput باید مشخص نمائیم ماشین ما قرار است چه کاری را انجام دهد . مثلاً برای یک web server ، انجام دستورات یا تراکنش ها به عنوان بازدهی آن تعریف نمی شود بلکه تعداد سرویس که به کاربران خود می دهد ، بازدهی سیستم را مشخص می نماید - یا مثلاً سرور مربوط به آموزش یک دانشگاه برای تعداد دانشجویانی که به صورت online می توانند سرویس دریافت نمایند به عنوان بازدهی تعریف می شود یعنی کارهاست که بازدهی یا throughput را تعریف می کنند .

performance یا کارآیی را به عنوان تعداد دستوراتی یا کارهایی که در واحد زمان انجام می‌شود و یا به صورت $\frac{1}{\text{Execution time}}$ تعریف می‌نمایند ولی در این رابطه تعیین شده است که زمان اجرای یک دستور، یا یک برنامه یا یک دستورالعمل در نظر گرفته می‌شود. زمانی که می‌خواهیم دو کامپیوتر را با یکدیگر مقایسه نماییم باید یک کار یکسان را برای مقایسه آنها در نظر بگیریم.

5

Performance Metrics

- Response (execution) time:
  - The time between the start and the completion of a task
  - Measures user perception of the system speed
  - Common in reactive and time critical systems
  - Single-user computer
- Throughput:
  - The total number of tasks done in a given time
  - Relevant to batch processing (billing, credit card processing)
  - Also many-user services (web servers)
- Power:
  - Power consumed or battery life
  - Especially relevant for mobile

در کامپیوترهایی که دستورات متفاوت انجام می‌دهند برای مقایسه کارآیی کدام زمان را در نظر بگیریم. مشخص است که کمترین زمان، بهترین کارآیی را به ما می‌دهد و دستورات طولانی که دارای زمان اجرای طولانی‌تری هستند بدترین کارآیی را مشخص می‌نمایند. در صورتیکه میانگین آنها را در نظر بگیریم باز هم مقدار دقیق کارآیی مشخص نخواهد شد.

6

Response-time Metric

- Maximizing performance means minimizing response (execution) time

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

پارامترهایی که در سنجش کارآیی یک سیستم مهم هستند:

۱- زمان پاسخ (Response Time): مدت زمان بین شروع و اتمام یک پردازش است.

۲- بازدهی (Throughput): تعداد کارها یا پردازش‌هایی که در یک مدت معین انجام می‌شود.

۳- توان مصرفی (power): مصرف توان یا طول عمر باطری

7

Response-time Metric

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

- Performance of Processor P1 is better than P2 if
  - **For a given work load L**
  - P1 takes less time to execute L than P2

$$\text{Performance}(P_1) > \text{Performance}(P_2) \text{ w.r.t L}$$
$$\Rightarrow \text{Execution time}(P_1, L) < \text{Execution time}(P_2, L)$$

8

Response-time Metric

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

- Relative performance captures the performance ratio
  - For the same work load

$$\text{Speedup} = \frac{\text{CPU Performance}(P_2)}{\text{CPU Performance}(P_1)} = \frac{\text{Total execution time}(P_1)}{\text{Total execution time}(P_2)}$$

برای یک کار مشخص مقدار speed up را می‌توان از رابطه زیر بدست آورد:

$$\text{speedup} = \frac{\text{کارآیی کامپیوتر } P_2}{\text{کارآیی کامپیوتر } P_1} = \frac{\text{زمان اجرای کامل } P_1}{\text{زمان اجرای کامل } P_2}$$

در عمل افزار معمولاً به جای زمان اجرا از مزدان کار استفاده می‌شود.

Designer's Performance Metrics

- Users and designers measure performance using different metrics
  - Users: quotable metrics (GHz)
  - Designers: program execution

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$= \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- Designer focuses on reducing the clock cycle time and the number of cycles per program
- Many techniques to decrease the number of clock cycles also increase the clock cycle time or the average number of cycles per instruction (CPI)

Example

A program runs in 10 seconds on a computer "A" with a 400 MHz clock. We desire a faster computer "B" that could run the program in 6 seconds. The designer has determined that a substantial increase in the clock speed is possible, however it would cause computer "B" to require 1.2 times as many clock cycles as computer "A". What should be the clock rate of computer "B"?

- Why would this happen?
  - Maybe one operation takes one full clock cycle at 400 MHz (1/400 MHz = 2.5 ns)
  - All the rest take less than one cycle
  - Split this operation across multiple cycles
  - Can now increase clock rate, but also increase total cycles

کارآیی یا performance را از دو دیدگاه می توان مورد بررسی قرار داد :

۱- از دید کاربرهایی که از این سیستم استفاده می کنند .

۲- از دید طراحان سیستم

* از دید کاربر هر سیستمی که دارای حداقل کار با بالاتری است دارای کارآیی بیشتری است در صورتیکه از دید یک طراح ، زمان اجرای یک برنامه مهم است که سریعتر انجام شود . ۷

مدت زمان هر کلاک × تعداد کلاک های cpu برای هر برنامه = زمان اجرای cpu برای یک برنامه خاص

طراح می تواند با کم کردن زمان هر کلاک و نیز تعداد کلاک ها برای اجرای یک برنامه باعث افزایش کارآیی سیستم شود.

11

Example

A program runs in 10 seconds on a computer "A" with a 400 MHz clock. We desire a faster computer "B" that could run the program in 6 seconds. The designer has determined that a substantial increase in the clock speed is possible, however it would cause computer "B" to require 1.2 times as many clock cycles as computer "A". What should be the clock rate of computer "B"?

$$\text{CPU time (A)} = \frac{\text{CPU clock cycles}}{\text{Clock rate (A)}} \qquad 10 \text{ seconds} = \frac{\text{CPU clock cycles of program}}{400 \times 10^6 \text{ cycles/second}}$$

$$\text{CPU clock cycles of program} = 10 \text{ seconds} \times 400 \times 10^6 \text{ cycles/second}$$
$$= 4000 \times 10^6 \text{ cycles}$$

To get the clock rate of the faster computer, we use the same formula

$$6 \text{ seconds} = \frac{1.2 \times \text{CPU clock cycles of program}}{\text{clock rate (B)}} = \frac{1.2 \times 4000 \times 10^6 \text{ cycles}}{\text{clock rate (B)}}$$

$$\text{clock rate (B)} = \frac{1.2 \times 4000 \times 10^6 \text{ cycles}}{6 \text{ second}} = 800 \times 10^6 \text{ cycles/second}$$

محاسبه زمان cpu:

یک برنامه از مجموعه‌ای از دستورات تشکیل شده است لذا زمان cpu را می‌توان از روابط زیر بدست آورد:

12

Calculation of CPU Time

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

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

Or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

تعداد دستورات → تعداد کلاک برای هر دستور

Component of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instructions (CPI)	Average number of clock cycles/instruction
Clock cycle time	Seconds per clock cycle

مثال: یک برنامه در مدت ۱۰ ثانیه بر روی ماشین A که دارای کلاک ۴۰۰ مگاهرتزاست اجرا می‌شود. ما تصمیم داریم یک کامپیوتر B که سریعتر است این برنامه را در مدت ۶ ثانیه اجرا نمائیم. در اینصورت کامپیوتر B باید دارای چه فرکانس کلاکی باشد که بتواند این کار را انجام دهد؟ اگر بخواهیم تعداد کلاک های ماشین B را به اندازه ۱/۲ برابر بیشتر باشد در آنصورت تعداد کلاک های ماشین B را بدست آورید.

A: $10\,S = \frac{x}{400\,MHz}$ → تعداد کلاک A $x = 4000 \times 10^6$ تعداد کلاک B $= 1.2 \times x$

B: $6\,S = \frac{4000 \times 10^6}{f_B}$ → $f_B = \frac{4000}{6} \times 10^6\,Hz = \frac{4000}{6}\,MHz$ △ $= \frac{1.2 \times 4000 \times 10^6}{6} = 800 \times 10^6$

با توجه اینکه دید طراحان به زمان اجرای برنامه است لذا برای بهبود سیستم طراح می تواند تعداد کلاک ها را برای اجرای برنامه کمتر نماید. البته زمان کلاک نیز اگر کمتر شود زمان اجرای برنامه کمتر می شود ولی نکته ای که وجود دارد تکنولوژی ساخت ترانزیستورها و گیت ها یک محدودیت را برای کلاک ایجاد می کند.

13

CPU Time (Cont.)

- CPU execution time can be measured by running the program
- The clock cycle is usually published by the manufacture
- Measuring the CPI and instruction count is not trivial
  - Instruction counts can be measured by: software profiling, using an architecture simulator, using hardware counters on some architecture
  - The CPI depends on many factors including: processor structure, memory system, the mix of instruction types and the implementation of these instructions

در مثال صفحه قبل مشاهده می شود که فرکانس کار ماشین B دو برابر ماشین A می باشد درصورتیکه زمان اجرای آن نصف نمی باشد بلکه نسبت $\frac{6}{10}$ را دارد.

14

CPU Time (Cont.)

- Designers sometimes uses the following formula:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Where: C_i is the count of number of instructions of class i executed
CPI_i is the average number of cycles per instruction for that instruction class
n is the number of different instruction classes

تغییر تعداد کلاک وابسته به سازندگان است ولی برای اجرای یک دستور می توان با استفاده از تکنیک های تعداد کلاک را کاهش داد مثلاً از پردازش های موازی استفاده نمود. همچنین حقدار حافظه نیز در تعداد کلاک های مورد نیاز برای اجرای یک دستور تاثیر دارد.

طراحان سیستم گاهی اوقات برای بدست آوردن زمان cpu از رابطه زیر استفاده می نمایند:

$$\text{cpu clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

مثال: فرض کنید برای دو مجموعه دستورالعمل های یک ان دو پیاده سازی متفاوت داشته باشیم. بردوی ماشین A مدت زمان هر کلاک برابر ۱ نانوثانیه و CPI برابر ۲ است. در ماشین B مدت زمان هر کلاک ۲ نانوثانیه و CPI برابر ۱٫۲ است. کدامیک از این ماشین ها برنامه را سریعتر اجرا می نماید و هم مقدار؟ هردو ماشین دستورات یکسانی را برای برنامه اجرا می کنند. در نظر بگیرید که تعداد دستورات برابر I باشد.

15

Example

Suppose we have two implementation of the same instruction set architecture. Machine "A" has a clock cycle time of 1 ns and a CPI of 2.0 for some program, and machine "B" has a clock cycle time of 2 ns and a CPI of 1.2 for the same program. Which machine is faster for this program and by how much?

Both machines execute the same instructions for the program. Assume the number of instructions is "I",

CPU clock cycles (A) = I × 2.0 CPU clock cycles (B) = I × 1.2

The CPU time required for each machine is as follows:

CPU time (A) = CPU clock cycles (A) × Clock cycle time (A)
= I × 2.0 × 1 ns = 2 × I ns

CPU time (B) = CPU clock cycles (B) × Clock cycle time (B)
= I × 1.2 × 2 ns = 2.4 × I ns

Therefore machine A will be faster by the following ratio:

$$\frac{\text{CPU Performance (A)}}{\text{CPU Performance (B)}} = \frac{\text{CPU time (B)}}{\text{CPU time (A)}} = \frac{2.4 \times I \text{ ns}}{2 \times I \text{ ns}} = 1.2$$

بنابراین ماشین A به مقدار ۱٫۲ برابر سریعتر از ماشین B خواهد بود.

16

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Answer:

Sequence 1: executes 2 + 1 + 2 = 5 instructions

Sequence 2: executes 4 + 1 + 1 = 6 instructions

مثال فوق برای برنامه یکسانی بود که مقدار CPI به صورت میانگین داده شده بود. حال اگر برنامه ای ماشین های A و B دستوراتی اجرا شود که برای اجرای هر دستور مقدار کلاک متفاوتی لازم باشد در آنصورت از رابطه زیر استفاده می شود:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

17

Comparing Code Segments

Using the formula: $\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$

Sequence 1: CPU clock cycles = (2 ×1) + (1 ×2) + (2 ×3) = 10 cycles
Sequence 2: CPU clock cycles = (4 ×1) + (1 ×2) + (1 ×3) = 9 cycles

- Therefore Sequence 2 is faster although it executes more instructions

Using the formula: $CPI = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$

Sequence 1: CPI = 10/5 = 2
Sequence 2: CPI = 9/6 = 1.5

- Since Sequence 2 takes fewer overall clock cycles but has more instructions it must have a lower CPI

پس بنابراین کارآیی یک ماشین را به روش های مختلفی می توان بررسی نمود.

18

The Role of Performance

- Hardware performance is a key to the effectiveness of the entire system
- Performance has to be measured and compared to evaluate designs
- To optimize the performance, major affecting factors have to be known
- For different types of applications
  - different performance metrics may be appropriate
  - different aspects of a computer system may be most significant
- Instructions use and implementation, memory hierarchy and I/O handling are among the factors that affect the performance

اولین روش به صورت کارآیی سخت افزاری است. Clock Rate به صورت سخت افزاری است CPI هم سخت افزاری و هم نرم افزاری است زیرا CPI نسبت مقدار کلاک های CPU به مقدار دستورات است که مقدار کلاک ها سخت افزاری و مقدار دستورات نرم افزاری است و می توان آنها را با استفاده از روش های خاص تغییر داد. سخت افزار در کارآیی بسیار موثر است و برای بهینه کردن کارآیی می توان تغییراتی را در سخت افزار ایجاد کرد.

19

Calculation of CPU Time

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

	Instr. Count	CPI	Clock Rate
Program	X		
Compiler	X	X	
Instruction Set	X	X	
Organization		X	X
Technology			X

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Where: C_i is the count of number of instructions of class *i* executed
CPI_i is the average number of cycles per instruction for that instruction class
n is the number of different instruction classes

20

Important Equations (so far)

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

$$\text{Speedup} = \frac{\text{Performance (B)}}{\text{Performance (A)}} = \frac{\text{Time (A)}}{\text{Time (B)}}$$

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{CPI}_i \times \text{Instructions}_i$$

21

Amdahl's Law

The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by the improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

- A common theme in Hardware design is to *make the common case fast*
- Increasing the clock rate would not affect memory access time
- Using a floating point processing unit does not speed integer ALU operations

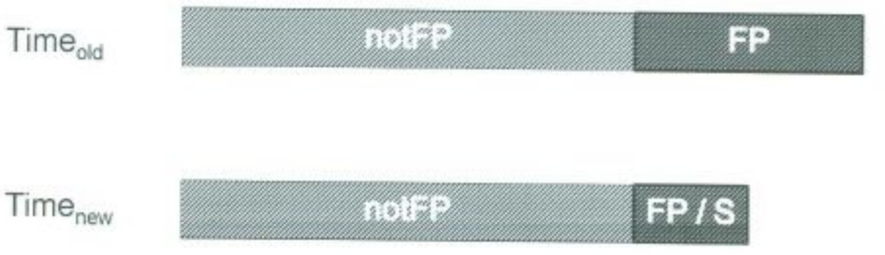
Example: Floating point instructions improved to run 2X; but only 34% of actual instructions are floating point

$Exec\text{-}Time_{new} = Exec\text{-}Time_{old} \times (0.66 + .34/2) = 0.83 \times Exec\text{-}Time_{old}$

$Speedup_{overall} = Exec\text{-}Time_{old} / Exec\text{-}Time_{new} = 1/0.83 = 1.205$

22

Ahmdal's Law Visually



Ahmdal's Law for Speedup

$$Time_{old} = Time_{old} * (Fraction_{unchanged} + Fraction_{enhanced})$$

$$Time_{new} = Time_{old} * \left(Fraction_{unchanged} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}\right)$$

$$Speedup_{overall} = \frac{Time_{old}}{Time_{new}} = \frac{Time_{old}}{Time_{old} * \left(Fraction_{unchanged} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}\right)}$$

$$= \frac{1}{Fraction_{unchanged} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

۹۳/۱۲/۲۱

pipeline یا خط لوله تکنیکی است که امکان اجرای سریع مجموعه دستورالعمل‌ها را فراهم می‌کند. اجرای دستورات می‌تواند به صورت ترتیبی یا pipeline صورت گیرد و از این در حالت خارج نیست.

CMSC 611: Advanced Computer Architecture

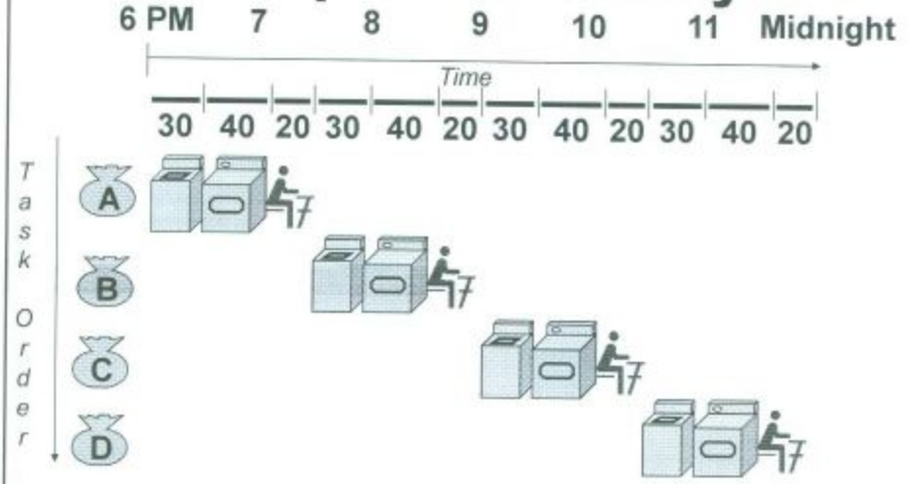
Pipelining

یکی از خصوصیات pipeline آن است که مدت اجرای یک دستور به تنهایی کاهش نمی‌یابد بلکه مجموع انجام کلیه کارها را کاهش می‌دهد یعنی سرعت اجرای مجموعه را کاهش می‌دهد. این روش برای سیستم‌های کامپیوتری نیز به همین صورت است.

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

مثال زیر کارها را به صورت مجموعه‌ای از شستن، خشک کردن و اطو کردن ترتیب کرده است و می‌خواهد چهار دست لباس را شسته، خشک کرده و اطو نماید. برای شستن ۳۰ دقیقه، خشک کردن ۴۰ دقیقه و اطو کردن ۲۰ دقیقه زمان نیاز است.

Sequential Laundry

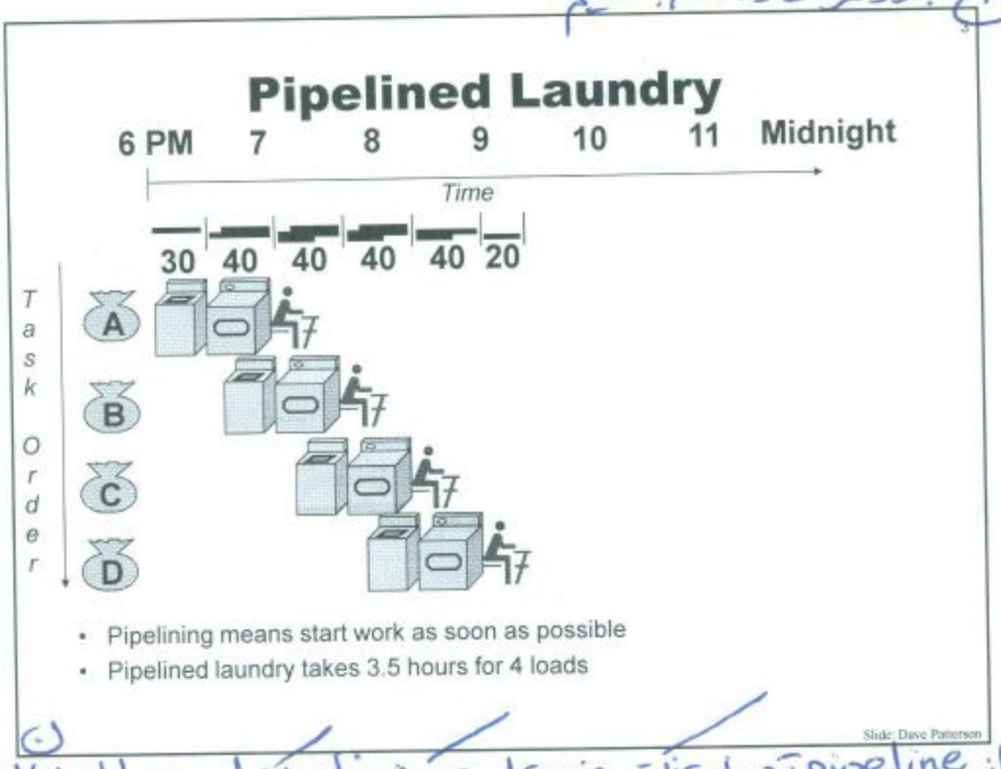


- Washer takes 30 min, Dryer takes 40 min, folding takes 20 min
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

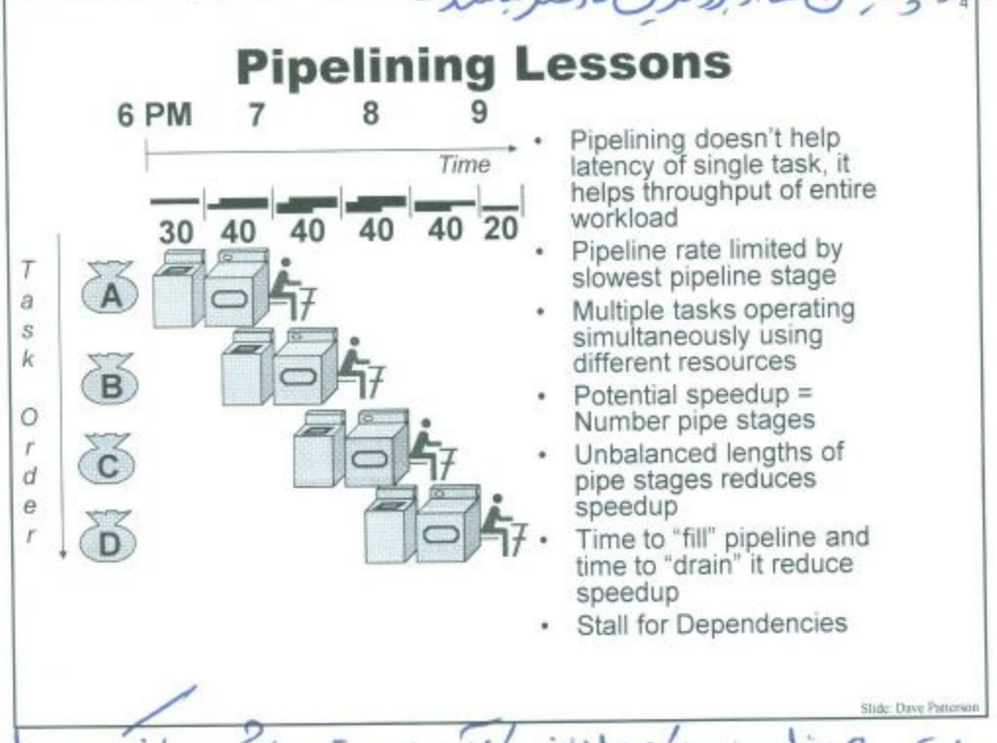
Slide: Dave Patterson

در مثال فوق اگر بخواهیم کارها را به صورت ترتیبی انجام دهیم عملاً برای هر دسته ۹۰ دقیقه زمان لازم است لذا برای چهار دسته به ۳۶۰ دقیقه زمان نیاز داریم در صورتیکه استفاده از pipeline این امکان را به ما می‌دهد که هم‌زمان از چند دستگاه استفاده نماییم که با استفاده از این روش مدت زمانی که طول می‌کشد تا ۴ دسته شسته شده، خشک شده و اطو شوند به سه ساعت و نیم یا ۲۱۰ دقیقه کاهش می‌یابد. البته باید توجه نمود برای آنکه بتوان از سیستم pipeline استفاده نمود باید امکان تفکیک منابع وجود داشته باشد.

اولین قدم در pipeline کردن یک کار، تفکیک درست منابع آن کار است. pipeline به زمان انجام یک کار به تنهایی کمک نمی‌کند بلکه زمان اجرای کل کارها را کاهش می‌دهد و باعث ارتقای کارآیی کل کار می‌شود. امکان آنکه بتوان در یک سیستم از pipeline استفاده نمود آن است که امکان استفاده از چند منبع بطور هم‌زمان را داشته باشیم.



در حالت استفاده از pipeline توجه به این نکته ضروری است که همه منابع ممکن است دارای زمان اجرای یکسان نباشند لذا ما باید کندترین زمان را برای اجرا در نظر بگیریم مانند زمانی که می‌خواهیم کار را در پارتیشن‌های مختلف قرار دهیم مجبوریم که اندازه پارتیشن‌ها را برزگترین کار تعیین کنند.

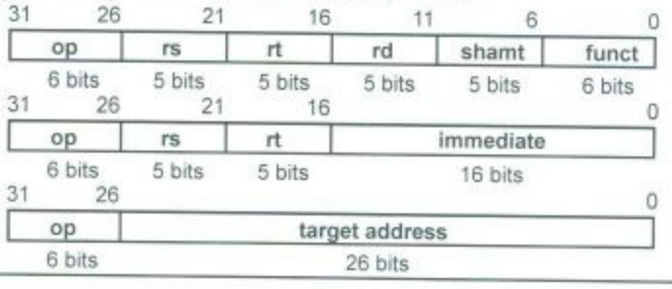


هدف از به کاربردن pipeline امکان افزایش کارآیی در سیستم می‌باشد. زمانی که pipeline استفاده می‌کنیم حداکثر میزان کارآیی به میزان تعداد مراحل pipeline نسبت به حالت ترتیبی افزایش می‌یابد یعنی مثلاً اگر تعداد مراحل pipeline سه مرحله باشد در آنصورت میزان کارآیی حداکثر در بهترین حالت سه برابر حالت ترتیبی خواهد شد یا به طور کلی اگر تعداد مراحل n مرحله باشد، کارآیی حداکثر n برابر حالت ترتیبی می‌شود.

یکی از عواملی که باعث می‌شود میزان سریع یا speed up به n نرسد، بالانس نبودن زمان اجرای مراحل pipeline خواهد بود لذا در pipeline کردن کارها، مراحل را به گونه‌ای تفکیک می‌کنیم که علاوه بر شرایط بالا از نظر مدت زمانی هم تقریباً یکسان باشند و تفاوتی با هم نداشته باشند.

MIPS Instruction Set

- RISC characterized by the following features that simplify implementation:
  - All ALU operations apply only on registers
  - Memory is affected only by load and store
  - Instructions follow very few formats and typically are of the same size



Single-cycle Execution

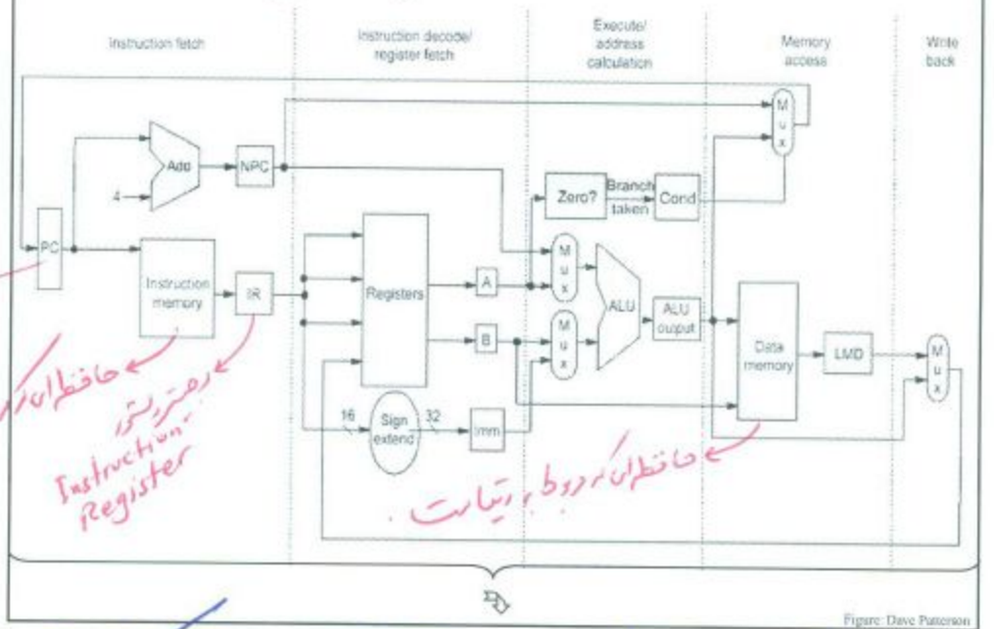
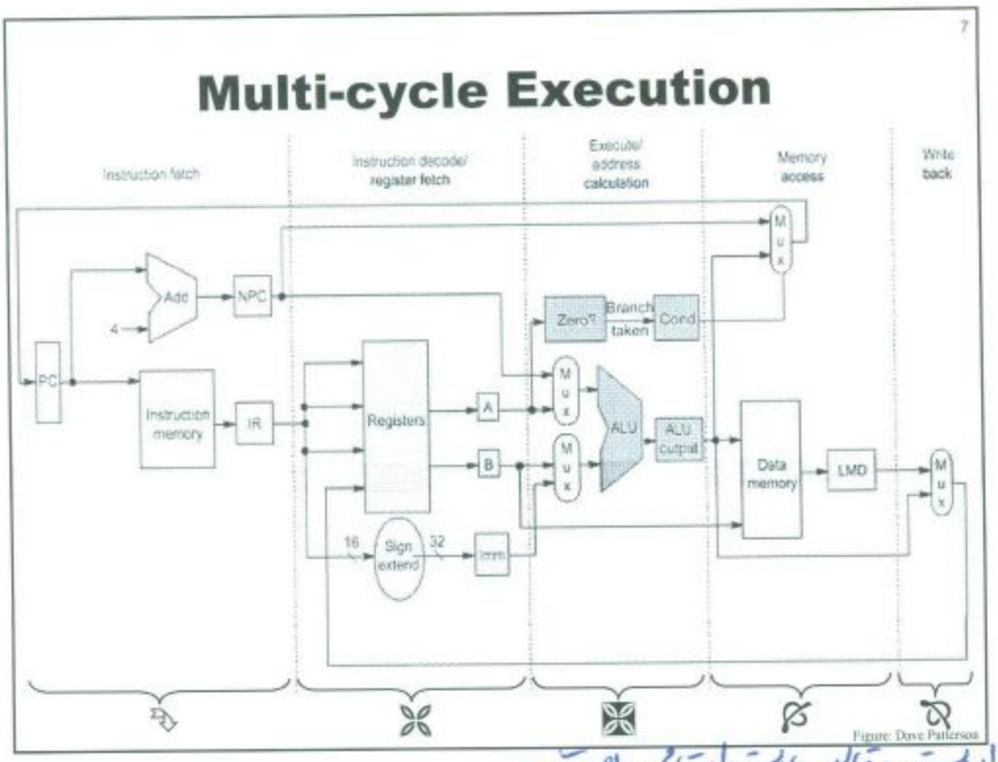


Figure: Dave Patterson

یکی دیگر از عوامل، شروع کار کردن با pipeline و پایان آن است. برای آنکه کاری شروع به انجام به صورت pipeline کند مدت زمانی طول می‌کشد که به آن زمان پر شدن یا Fill گفته می‌شود. همچنین زمانی طول می‌کشد تا کل کارها انجام شود که اصطلاحاً به آن زمان خالی شدن pipeline یا drain می‌گویند.

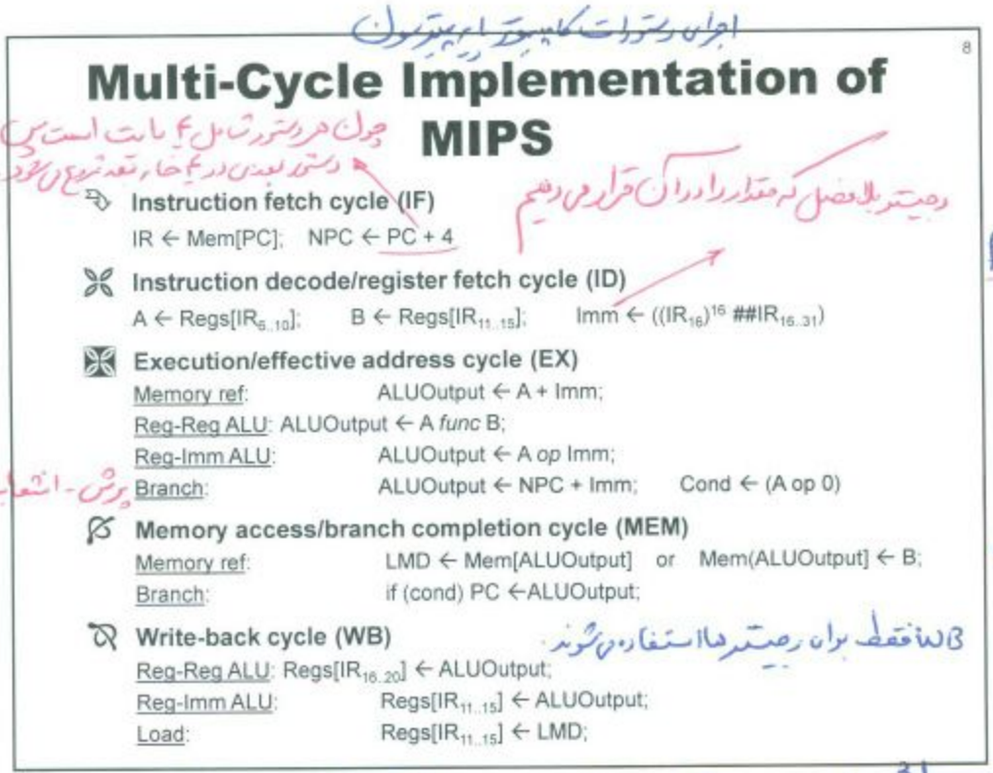
هرچه تعداد کارهایی که وارد pipeline می‌شود بیشتر باشد بهره‌برداری از pipeline بیشتر خواهد بود. در واقع هرچه تعداد کارها بیشتر باشد زمان پر شدن و خالی شدن بین کارهای بیشتری سرشکن می‌شود پس بنابراین زمان پر شدن و خالی شدن می‌تواند مقدار speed up را تغییر دهد.

گاهی اوقات لازم است بعضی از کارها در این کارهای دیگر به صورت تقلیدی درآیند. هرچه تعداد این کارهای تقلیدی بیشتر شود باعث کاهش تاخیر یا speedup می‌گردد.
توانایی speedup یا تاخیر سیستم در زمانی که از pipeline استفاده می‌کنیم حداکثر به میزان مراحل انجام pipeline است.



در کامپیوترهای RISC تعداد دستور و قالب دستورات محدود هستند.

اجرای دستورات کامپیوتر در پردازنده

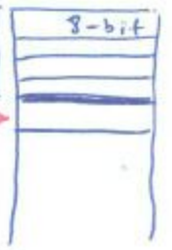


چون هر دستور شامل ۴ بایت است پس دستور بعدی در ۴ خانه بعدی قرار می‌گیرد.
رجیستر بالافصل که مقدار را در آن قرار می‌دهیم
کامپیوتر پایه پترسون
Fetch
واکشی
اجرا کردن
دیکد کردن
دستور آدرس حافظه
* در کامپیوتر پایه پترسون مراحل اجرای یک دستور چهار مرحله است.
برش - انشعاب
ALU فقط برای رجیسترها استفاده می‌شود.

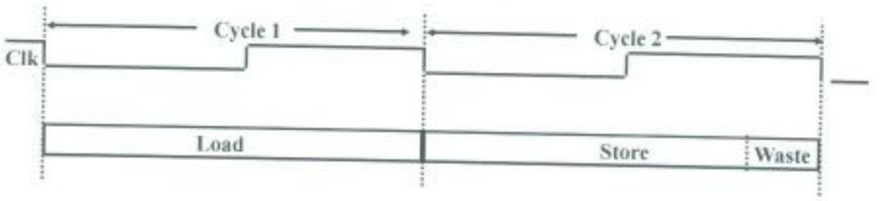
در کتاب مانو مراحل اجرای یک دستور سه مرحله بود
Fetch
dicode
exe
در کامپیوتر پایه پترسون دو حافظه مجزا وجود دارد که یکی جهت دستورات و دیگری جهت داده به کار می‌رود.
یکی از خصوصیات کامپیوتر پایه پترسون آن است که امکان ایجاد pipeline را به ما می‌دهد.
آدرس دستور همیشه در ثبات PC قرار دارد.

چون هر دستور به ۴ بایت حافظه در کامپیوتر با پترسون نیاز دارد پس دستور بعدی در آدرس چهار خانه بعد شروع خواهد شد.

در کامپیوتر با پایه مانند هر دستور در یک بایتی در زیر هم قرار می‌گیرد لذا فضایی این حافظه که برای هر دستور اشغال می‌شود با دستورات دو مرحله تفاوت دارد.



Single Cycle



- Cycle time long enough for longest instruction
- Shorter instructions waste time
- No overlap

Figure: Dave Patterson

کارهای یک دستور را می‌توانیم چند بخش تقسیم کنیم که برای اجرای هر بخش مجبوریم یک سری سیگنال‌های کنترلی را ایجاد نماییم:

- واکشی
- دی کد کردن
- اجرا
- دستری به حافظه
- write back

Multiple Cycle

دستور store نیازی به WB ندارد لذا پس از ۴ سیکل شروع به دستور بعدی می‌نماید.

- Cycle time long enough for longest stage
- Shorter stages waste time
- Shorter instructions can take fewer cycles
- No overlap

حالت مانع سیکل از تک سیکل بهتر است ولی با هم واقعی نیست

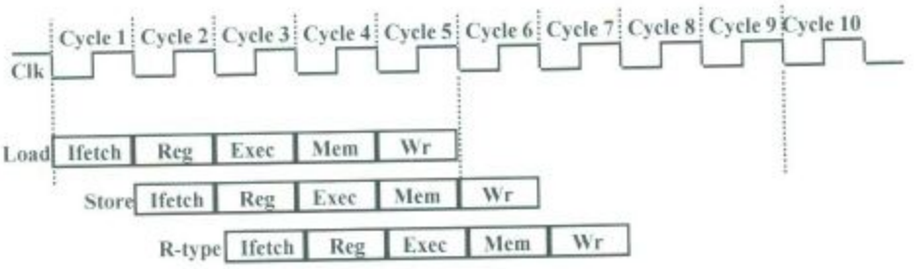
Figure: Dave Patterson

برخی از کامپیوترها به صورت تک سیکلی عمل می‌کنند بطوریکه برای اجرای هر دستور مجبور هستیم یک سیکل را فعال نماییم که این سیگنال باید به اندازه مدت زمان لازم برای اجرای آن دستور فعال باشد. در صورتیکه مدت زمان این سیکل به پایان رسید، کامپیوتر ما در صورتی موفق است که بتواند در مدت زمان آن سیکل دستورات را به پایان برساند. در کامپیوترها دستورات متفاوت با مدت زمان اجرای متفاوت وجود دارد لذا در کامپیوترهای تک سیکلی مدت زمان کلاک باید طوری باشد که کندترین دستورات بتوانند در یک سیکل اجرا شوند لذا برای بعضی از دستورات که سریعتر هستند مقداری اتلاف زمان خواهیم داشت. بنابراین پیاده‌سازی تک سیکلی باعث هدر رفتن زمان برای دستوراتی که به زمان کمتری نیاز دارند می‌شود.

اگر کامپیوتری به صورت تک سیکلی باشد نمی‌توان آن را به صورت pipeline استفاده نمود بلکه باید ابتدا آن را مرحله‌ای نموده و سپس به صورت pipeline استفاده نمائیم. مدت زمان سیکل‌ها را باید به گونه‌ای انتخاب نمائیم که کندترین دستورات قابل اجرا باشند لذا در اجرای دستورات به صورت pipeline، حداکثر دستور را مثلاً load که نیاز به پنج سیکل دارد را انتخاب می‌نمائیم.

11

Pipeline



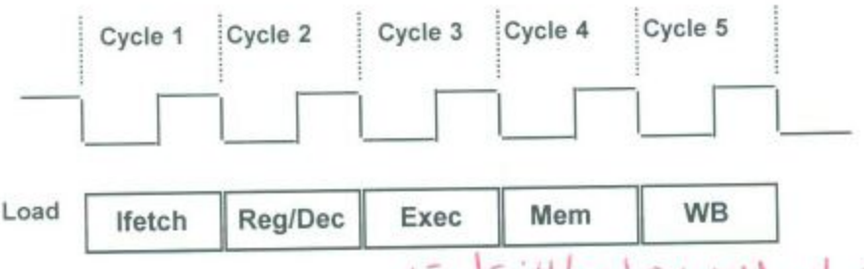
- Cycle time long enough for longest stage
- Shorter stages waste time
- No additional benefit from shorter instructions
- Overlap instruction execution

Figure: Dave Patterson

- مدت زمان اجرا
- مدت زمان بین اجرا

12

Stages of Instruction Execution



دستور load از همه دستورات طولانی‌تر است

- The load instruction is the longest
- All instructions follows at most the following five steps:
  - Ifetch: Instruction Fetch
    - Fetch the instruction from the Instruction Memory and update PC
  - Reg/Dec: Registers Fetch and Instruction Decode
  - Exec: Calculate the memory address
  - Mem: Read the data from the Data Memory
  - WB: Write the data back to the register file

Slide: Dave Patterson

مدت زمان اجرا:

مدت زمان شروع اولین دستور تا شروع آخرین دستور را گویند که هدف این مقدار کمتر باشد عملاً مثل این است که زمان پاسخ کوتاهتر است زیرا می‌دانیم زمان پاسخ برابر مجموع زمان انتظار و زمان اجرا می‌باشد. در pipeline زمان اجرا تغییر نمی‌کند در صورتی که زمان انتظار کاهش می‌یابد.

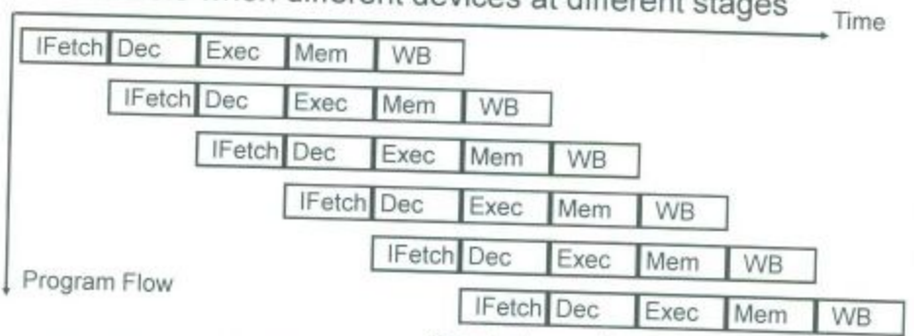
کارآیی یک pipeline به چند مورد بستگی دارد :

- هرچه تعداد دستورات بیشتر شود کارآیی بیشتر می‌شود ولی زمان اجرای یک دستور تغییر نمی‌کند.
- برای برخی از دستورات مستقل ممکن است زمان اجرا نسبت به حالت ترتیبی بیشتر شود.

13

Instruction Pipelining

- Start handling next instruction while the current instruction is in progress
- Feasible when different devices at different stages

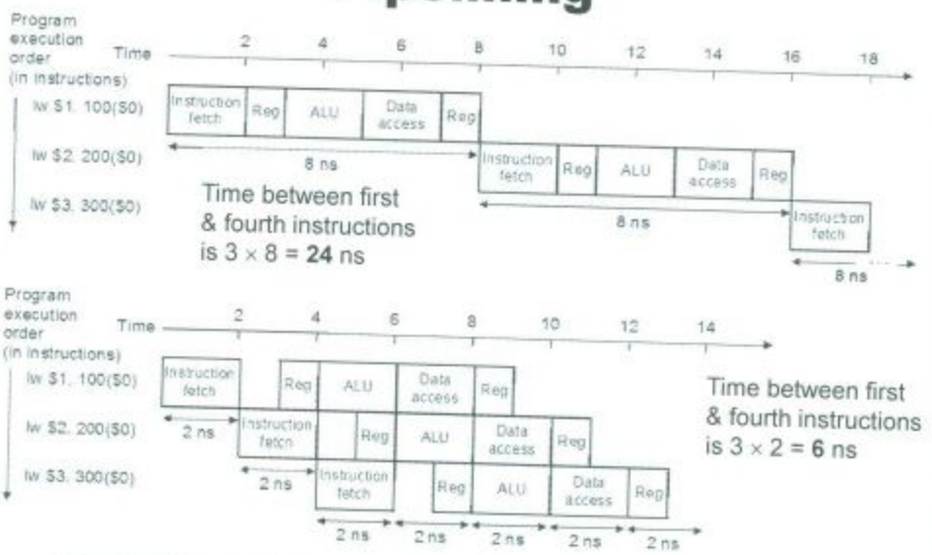


$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

Pipelining improves performance by increasing instruction throughput

14

Example of Instruction Pipelining



Ideal and upper bound for speedup is number of stages in the pipeline

در حالت استفاده از pipeline مشاهده می‌شود که زمان بین شروع اولین دستور و شروع دستور چهارم فقط ۶ نانوثانیه است.

15

Pipeline Performance

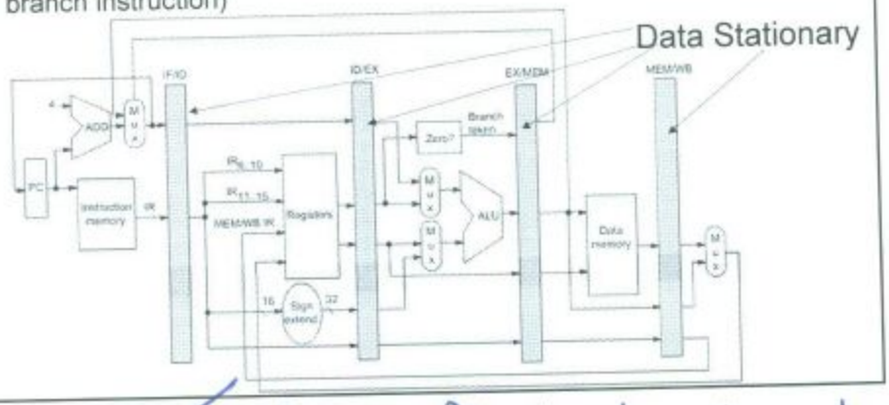
- Pipeline increases the instruction throughput
  - not execution time of an individual instruction
- An individual instruction can be **slower**:
  - Additional pipeline control
  - Imbalance among pipeline stages
- Suppose we execute 100 instructions:
  - Single Cycle Machine
    - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
  - Multi-cycle Machine
    - 10 ns/cycle x 4.2 CPI (due to inst mix) x 100 inst = 4200 ns
  - Ideal 5 stages pipelined machine
    - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns
- Lose performance due to fill and drain

در مثال فوق مقایسه ای بین زمان اجرا برای کامپیوترهای تک سیکلی، چند سیکلی و pipeline صورت گرفته است.

16

Pipeline Datapath

- Every stage must be completed in one clock cycle to avoid stalls
- Values must be latched to ensure correct execution of instructions
- The PC multiplexer has moved to the IF stage to prevent two instructions from updating the PC simultaneously (in case of branch instruction)



در پیاده سازی pipeline باید هر مرحله بدون داشتن وقفه در یک سیکل اجرا شود یعنی مراحل اجرای pipeline باید به صورت پشت سر هم انجام شود. همچنین دستورات باید پشت سر هم وارد شوند.

در اجرای pipeline باید از یک سری حافظه های موقتی بنام latch استفاده نمائیم تا از اجرای صحیح دستورات مطمئن شویم. تعداد latch ها باید برابر n-1 باشد که n تعداد مراحل اجرای pipeline می باشد. یعنی اگر ۱۰ مرحله برای اجرای pipeline داریم به تعداد ۹ عدد latch نیاز داریم.

علیرغم استفاده از حافظه‌های latch باز هم ممکن است مخاطره‌هایی وجود داشته باشد که نگذارند به کارآیی لازم دست یابی داشته باشیم که باید شناسایی شده و برطرف گردند. این مخاطرات را می‌توان به‌صورت زیر دسته‌بندی نمود: انواع Hazard :

- ساختاری structured hazard
- دیتا Data hazard
- کنترل Control hazard

17

Pipeline Stage Interface

Stage	Any Instruction		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if ((EX/MEM.opcode == branch) & EX/MEM.cond) {EX/MEM.ALUOutput} else {PC + 4});		
ID	ID/EX.A ← Regs[IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs[IF/ID.IR$_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← (IF/ID.IR$_{16}$)^{16} ## IF/ID.IR$_{16..31}$;		
	ALU	Load or Store	Branch
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A func ID/EX.B; Or EX/MEM.ALUOutput ← ID/EX.A op ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B ← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC + ID/EX.Imm; EX/MEM.cond ← (ID/EX.A op 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; Or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR$_{16..20}$] ← MEM/WB.ALUOutput; Or Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.ALUOutput;	For load only: Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.LMD;	

Hazard ها یا مخاطره‌ها هیچ با منتظر ماندن قابل رفع شدن هستند ولی این کار باعث کاهش کارآیی می‌گردد لذا برای افزایش تسریع باید با استفاده از روش‌های دیگری این مخاطره‌ها را برطرف نمائیم.

18

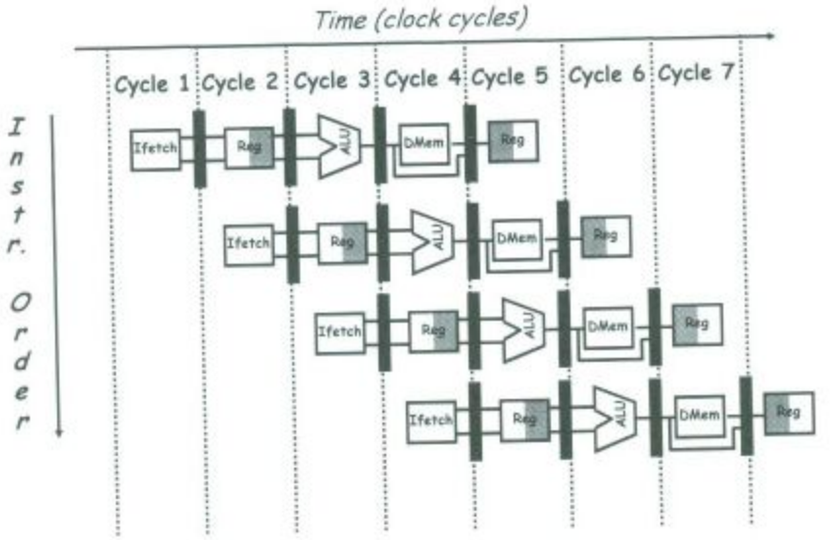
Pipeline Hazards

- Cases that affect instruction execution semantics and thus need to be detected and corrected
- Hazards types
  - Structural hazard: attempt to use a resource two different ways at same time
    - Single memory for instruction and data
  - Data hazard: attempt to use item before it is ready
    - Instruction depends on result of prior instruction still in the pipeline
  - Control hazard: attempt to make a decision before condition is evaluated
    - branch instructions
- Hazards can always be resolved by waiting

همانطور که گفته شد pipeline امکان اجرای پردازش دسته‌ای از دستورات را با هم سریعتر می‌کند. اولین کاری که بتوان کرد برنامه را به صورت pipeline اجرا نمود آن است که بتوان آن را مرحله‌بندی کرد. Hazard های ساختاری مربوط به سخت افزاری باشد. توسط سخت افزار باید بتوان مسیرهای مختلف دستورات را مرحله‌بندی نمود یعنی در واقع سخت افزار باید امکان تفکیک مراحل را فراهم نماید. زمانی که دستوراتی را به صورت pipeline اجرا می‌نمائیم در خیلی از مواقع ممکن است دستورات به هم وابسته باشند و نتوان آنها را تفکیک نمود.

19

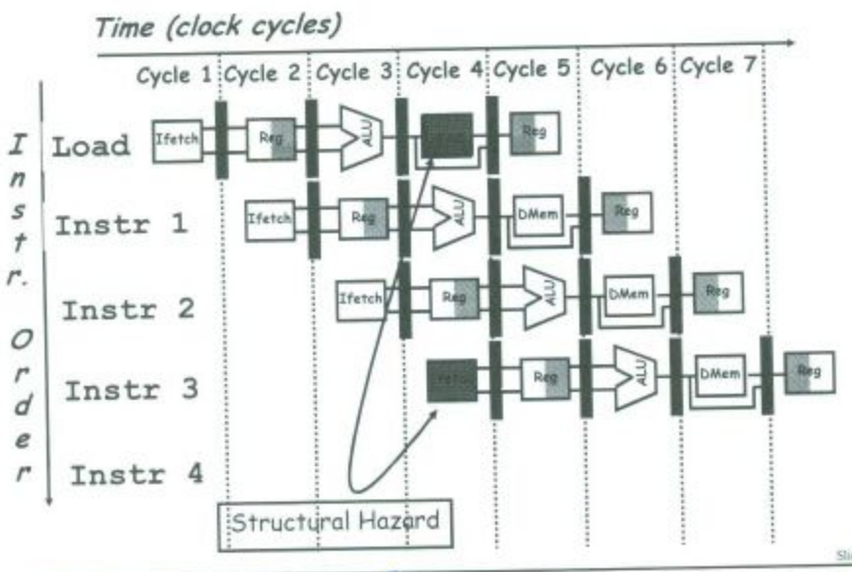
Visualizing Pipelining



Slide: David Culler

20

Example: One Memory Port/Structural Hazard



Slide: David Culler

در مثال فوق هم می خواهیم در حافظه بنویسیم و هم در دستور واکشی می خواهیم از حافظه بخوانیم که این امر باعث ایجاد خطا می شود. به این نوع فرآیند ساختاری یا سخت افزاری گفته می شود. برای رفع این اشکال یا باید با ایجاد یک حباب منتظر بمانیم یعنی یک شیفت در اجرای دستورات بعدی ایجاد نمائیم یا از سخت افزارهای دیگر استفاده کنیم.

if A < B
$r_1 = r_2 + r_3$
else
$r_1 = r_4 + r_5$

مخاطره های ساختاری را فقط به صورت سخت افزاری می توان برطرف نمود.
دو راهکرد مختلف برای مقابله با مخاطره های ساختاری وجود دارد:
۱- منتظر بمانیم یعنی عملاً آن مرحله را نمی توان تفکیک کرد یعنی سخت افزار اینگا تفکیک
مراحل را نداشته باشد پس بنابراین باید تا خیر را تحمل نمائیم.
۲- سخت افزار را تغییر دهیم.

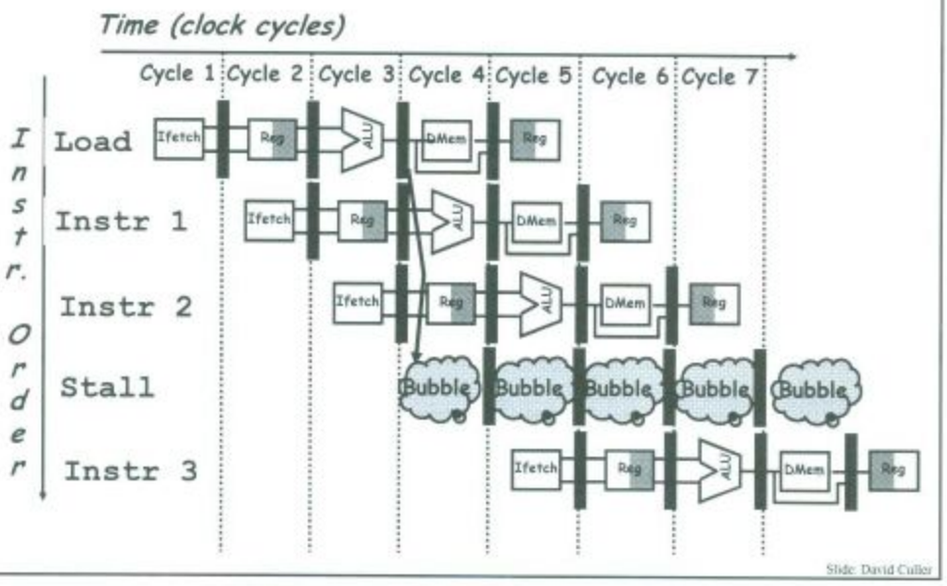
21

Resolving Structural Hazards

1. Wait
   - Must detect the hazard
     - Easier with uniform ISA
   - Must have mechanism to stall
     - Easier with uniform pipeline organization
2. Throw more hardware at the problem
   - Use instruction & data cache rather than direct access to memory

22

Detecting and Resolving Structural Hazard



Slide: David Culler

$r_1 = r_2 + r_3$
$r_2 = r_1 + r_4$ وابستگی داده

در این مثال عمل دی کد کردن دستور دوم و اجرای دستور اول همزمان می خواهد انجام شود در صورتیکه زمانی می توان دستور بعدی را اجرا نمود که دستور قبلی کاملا اجرا شده باشد. این نوع مخاطره را مخاطره دیتا می نامند.

23

Stalls & Pipeline Performance

$$\text{Pipelining Speedup} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

$$\text{Ideal CPI pipelined} = 1$$

$$\text{CPI pipelined} = \text{Ideal CPI} + \text{Pipeline stall cycles per instruction}$$
$$= 1 + \text{Pipeline stall cycles per instruction}$$

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

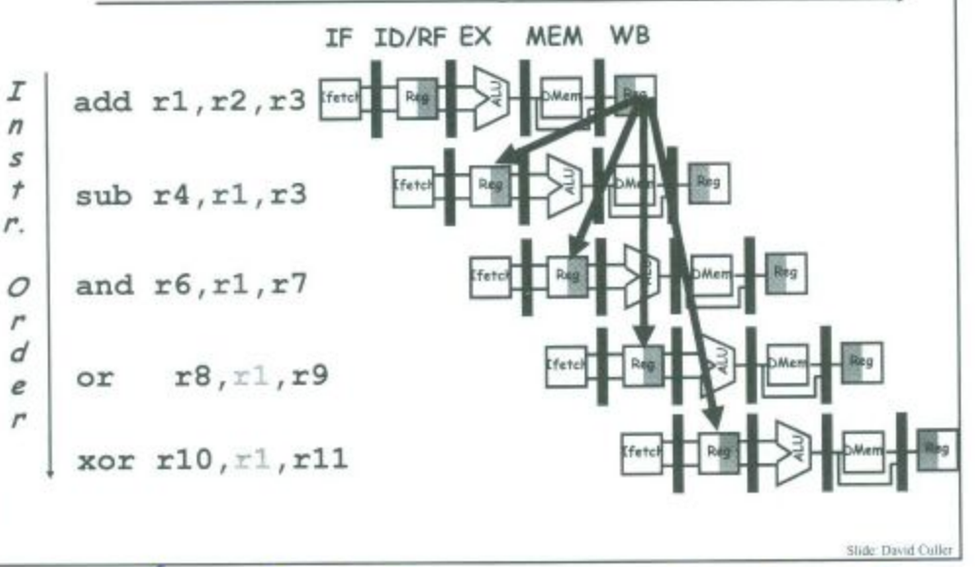
Assuming all pipeline stages are balanced

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

مخاطره های دیتا را هم می توان به صورت سخت افزاری و هم به صورت نرم افزاری برطرف نمود.
مخاطرات دیتا زمانی بوجود می آید که بخواهیم از داده ای استفاده نمائیم قبل از آنکه آن داده بدست آید.

24

Data Hazards



Slide: David Culler

در شکل فوق مشاهده می گردد ما می خواهیم از داده ای استفاده نمائیم که دو سیکل بعد در ثبات نوشته می شود پس عملا مخاطره دیتا به وجود می آید. در دستور سوم هم می خواهیم از داده ای استفاده نمائیم که در سیکل بعدی تهیه می شود. اما مشاهده می شود که از دستور چهارم به بعد مشکل بوجود نمی آید. همانگونه که مشاهده می شود در دستور اول در نیمه اول WB عمل ثبت داده انجام می شود لذا در دستور چهارم در نیمه دوم می توان از ثبات برای داده استفاده نمود و در اینحالت اشکالی بوجود نمی آید. توجه شود اگر WB کردن و دی کد کردن نیز در یک سیکل انجام می شد در آن صورت در اجرای دستور چهارم هم مخاطره دیتا پیش می آمد.

سه نوع مخاطره دیتا وجود دارد: ۱- خواندن بعد از نوشتن
۲- نوشتن بعد از خواندن
۳- نوشتن بعد از نوشتن

25

Three Generic Data Hazards

- Read After Write (RAW)
  $Instr_J$ tries to read operand before $Instr_I$ writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- Caused by a "Data Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Slide: David Culler

با استفاده از روش های نرم افزاری و سخت افزاری یا باید از وقوع مخاطرات دیتا جلوگیری نمائیم یا اگر به وقوع پیوست آنها را تشخیص داده و برطرف نمائیم.

26

Three Generic Data Hazards

- Write After Read (WAR)
  $Instr_J$ writes operand before $Instr_I$ reads it

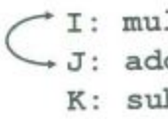
```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "anti-dependence" in compilers.
  - This results from reuse of the name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Slide: David Culler

Three Generic Data Hazards

- Write After Write (WAW)
  $Instr_J$ writes operand before $Instr_I$ writes it.



```
I: mul r1,r4,r3
J: add r1,r2,r3
K: sub r6,r1,r7
```

- Called an "output dependence" in compilers
  - This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Do see WAR and WAW in more complicated pipes

Slide: David Culler

۹۴/۱/۲۰

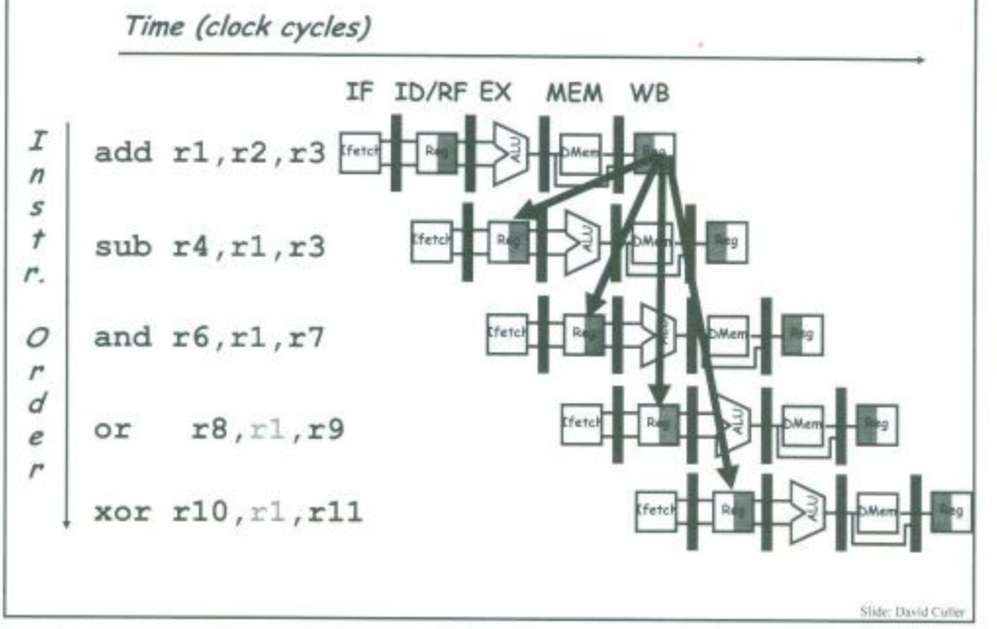
CMSC 611: Advanced Computer Architecture

Pipelining

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

2

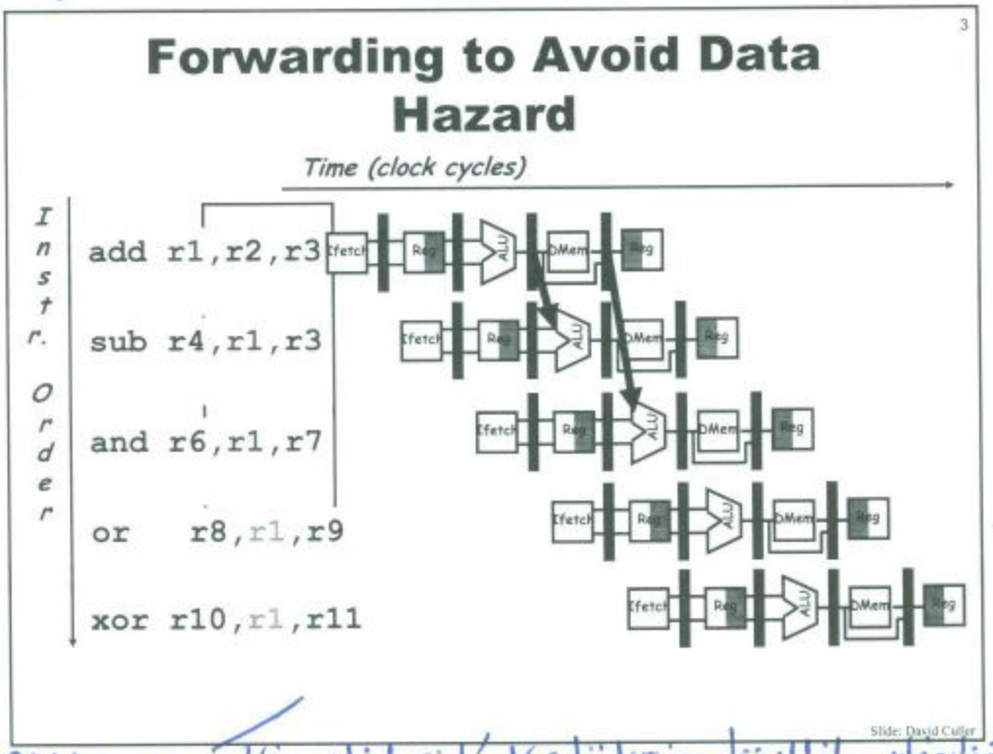
Data Hazards



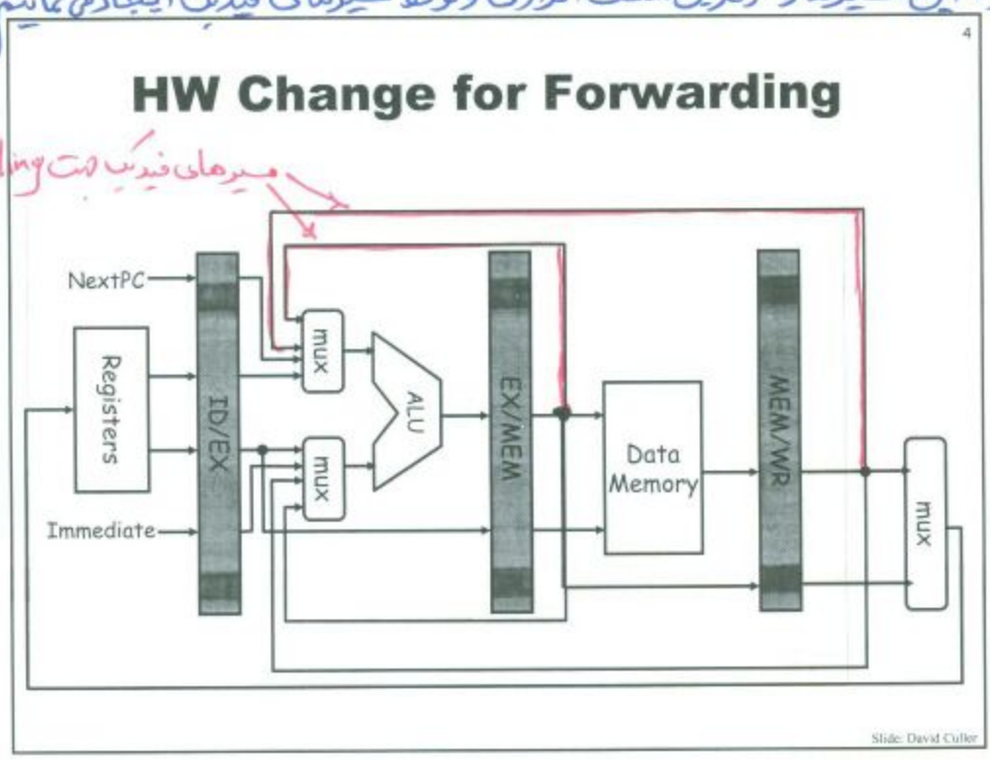
Slide: David Culler

همانطورکه گفته شد برای رفع مخاطره دیتا یکی از روش‌ها انتظار است که این کار باعث ایجاد تأخیر می‌گردد لذا باید سعی شود این شکل بودن ایجاد تأخیر برطرف شود. برای رفع اشکال مربوط به مخاطره دیتا یکی از راه‌ها استفاده از Forwarding است.

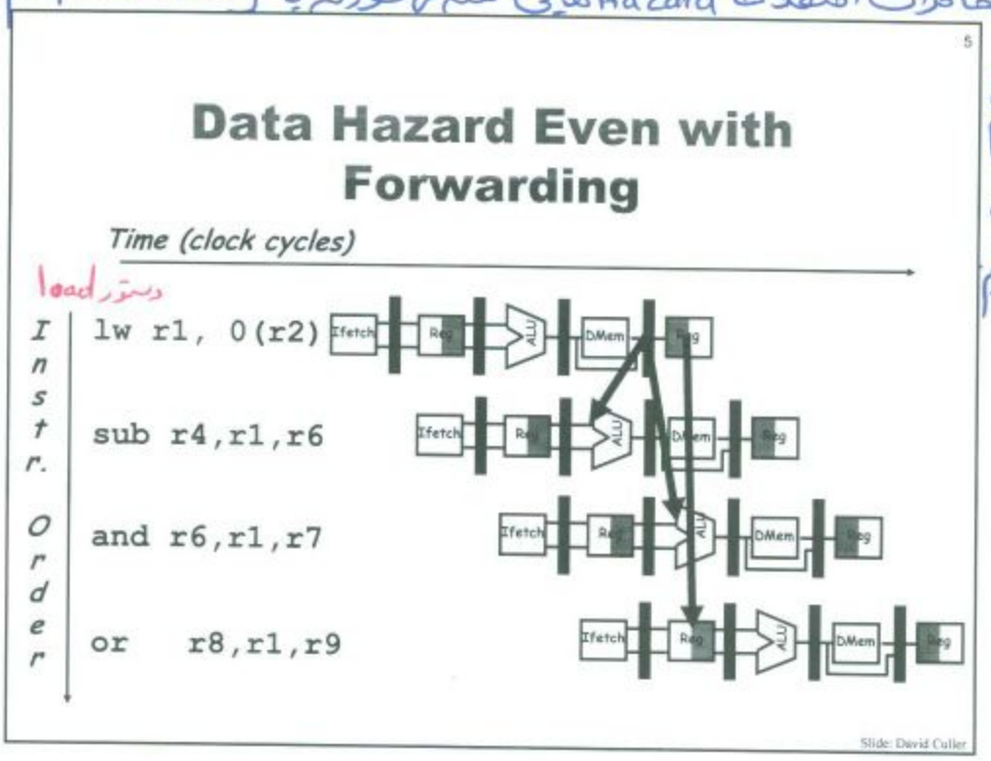
latch مانند چک نویسی است که به طور موقت اطلاعات هر مرحله را ذخیره می کند. latch این امکان را می دهد که اطلاعات دستور اول را ذخیره نموده و اطلاعات مربوط به دستور دوم را در ثبات های خودش بنویسد. در اینحالت ALU در دستور دوم قرار است از اطلاعات ثباتی استفاده نماید که هنوز اطلاعات در آن ثبت نگردیده است ولی به صورت چک نویس در latch مربوط به ALU قرار دارد و ALU می تواند از این اطلاعات مربوط

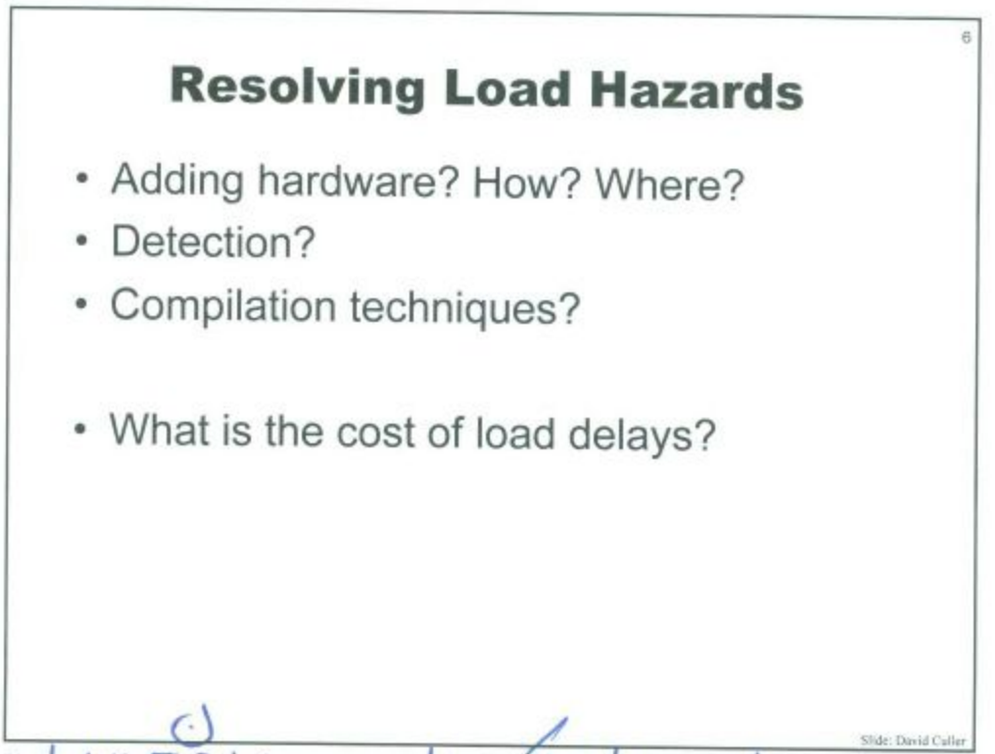


به latch خودش استفاده نماید. لذا از نظر سخت افزاری امکانی را فراهم می نماییم که در ورودی ALU به جای latch خودش منتقل گردد که به این عمل Forwarding می گویند. در دستور سوم نیز می توان از Forward مربوطه حافظه استفاده نمود. این مسیرها را از طریق سخت افزاری و توسط مسیرهای فیدبک ایجاد می نماییم.



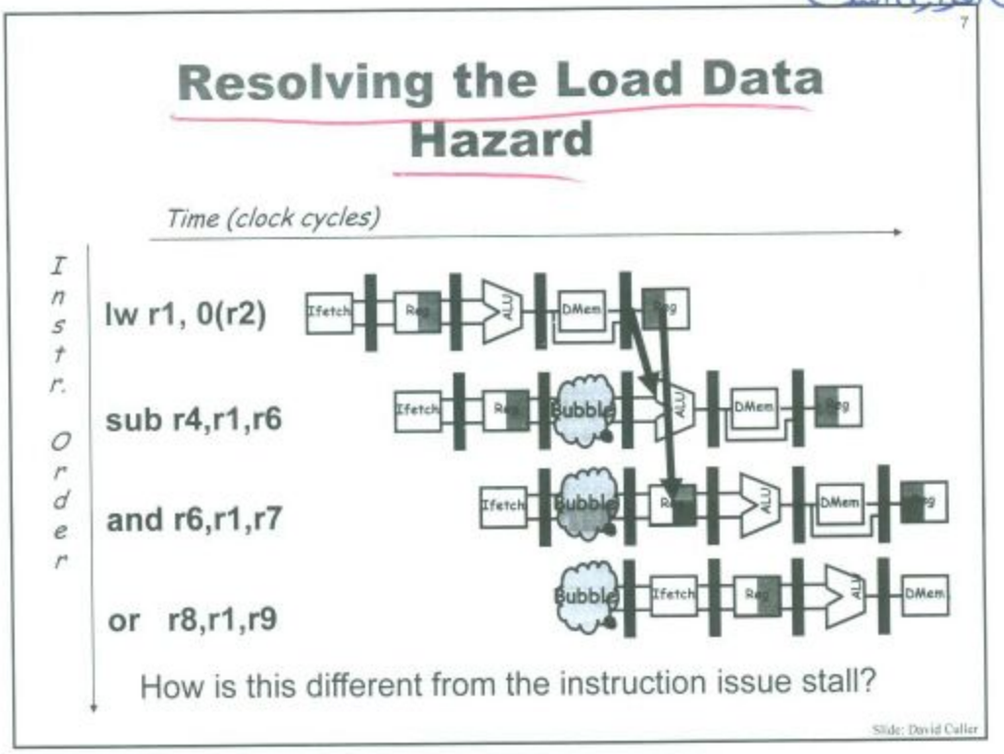
گاهی اوقات حتی با استفاده از Forwarding نیز Hazard دیتا رخ خواهد داد. اگر در دستورات، دستور load و store وجود داشته باشد علیرغم استفاده از Forwarding باز هم مخاطره داده بوجود می‌آید. در دستور load باید ابتدا داده از حافظه بیاید و آنرا وارد latch نمائیم و سپس در ثبات قرار دهیم. از نظر زمانی چون در اینحالت یک مرحله عقب تر هستیم پس در دستور دوم نمی‌توان از خروجی latch مربوط به حافظه استفاده نمود. به این مخاطرات اصطلاحاً Hazard هایی گفته می‌شود که با Forwarding هم نمی‌توان آنها را از بین برد. برای رفع این Hazard ها حتماً باید یک بازه زمانی منتظر بمانیم

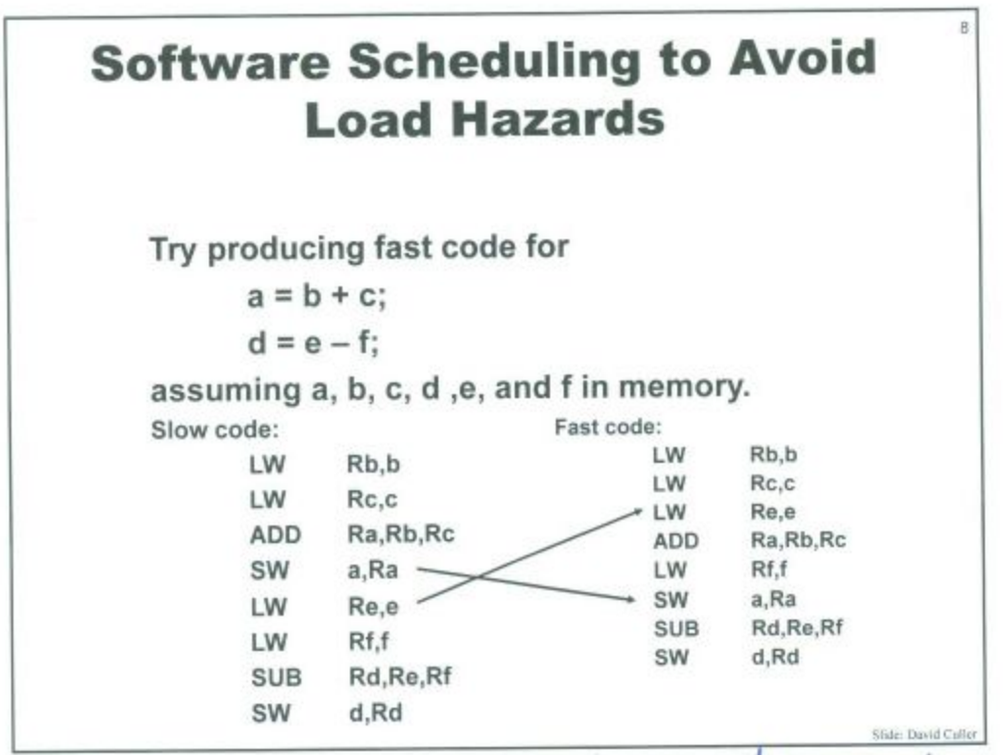




برای رفع Hazard های مربوط به دستور load که با Forwarding هم نمی‌توان آنها را برطرف نمود یا باید دست افزارهای دیگری اضافه نمائیم یا ترتیب اجرای دستورات را تغییر دهیم. در هر صورت باید توجه نمائیم که در اجرای هر یک از این روش ها باید بررسی نمائیم آیا روش به کار برده شده از نظر هزینه اجرای آن به صرفه است یا بهتر است که منتظر بمانیم. در هر صورت یک کامپایلر یا مربی باید وجود داشته باشد که ابتدا تشخیص دهد Hazard وجود دارد و سپس تشخیص دهد از کدام روش برای برطرف نمودن آن استفاده نماید.

تاکنون میگفتیم برای رفع Hazard میتوان یک Wait ایجاد نموده و اجازه ورود کل مراحل دستور را به pipeline ندهیم ولی برای دستوراتی مانند load یا store درصورتی که مخاطره‌ای رخ دهد میتوان به جای آنکه کل مراحل دستورات را با ایجاد حباب در حالت انتظار قرار دهیم فقط همان مرحله‌ای را که ایجاد مخاطره مینماید را منتظر نگه داریم. حسن این روش آن است که در ورود دستورات وقفه‌ای ایجاد نمی‌شود. این روش یک روش بهتر اقتصادی است





یکی دیگر از روش‌های برطرف نمودن Hazard های مربوط به دیتا، استفاده از روش جابجایی دستورات است. در مثال فوق با توجه به آنکه دستور Lw Re,e هیچ وابستگی به Rc ندارد لذا با جابجایی این دو دستور میتوان Hazard مربوط به دیتا را برطرف نمود. این روش در جاهایی استفاده می‌شود که از پردازنده‌های برای اجرای pipeline دستورات مشخص استفاده می‌شود که در آن موقع به صرفه است دستورات را به گونه‌ای جابجا نمود که کمترین Hazard مربوط به دیتا وجود داشته باشد. برطرف نمودن Hazard های دیتا به این روش اصلاً الگوریتم ثابتی ندارد و در خیلی از جاها امکان جابجایی دستورات وجود ندارد. ۳۲

9

Instruction Set Connection

- What is exposed about this organizational hazard in the instruction set?
- k cycle delay?
  - bad, CPI is not part of ISA
- k instruction slot delay
  - load should not be followed by use of the value in the next k instructions
- Nothing, but code can reduce run-time delays
- MIPS did the transformation in the assembler

Slide: David Culler

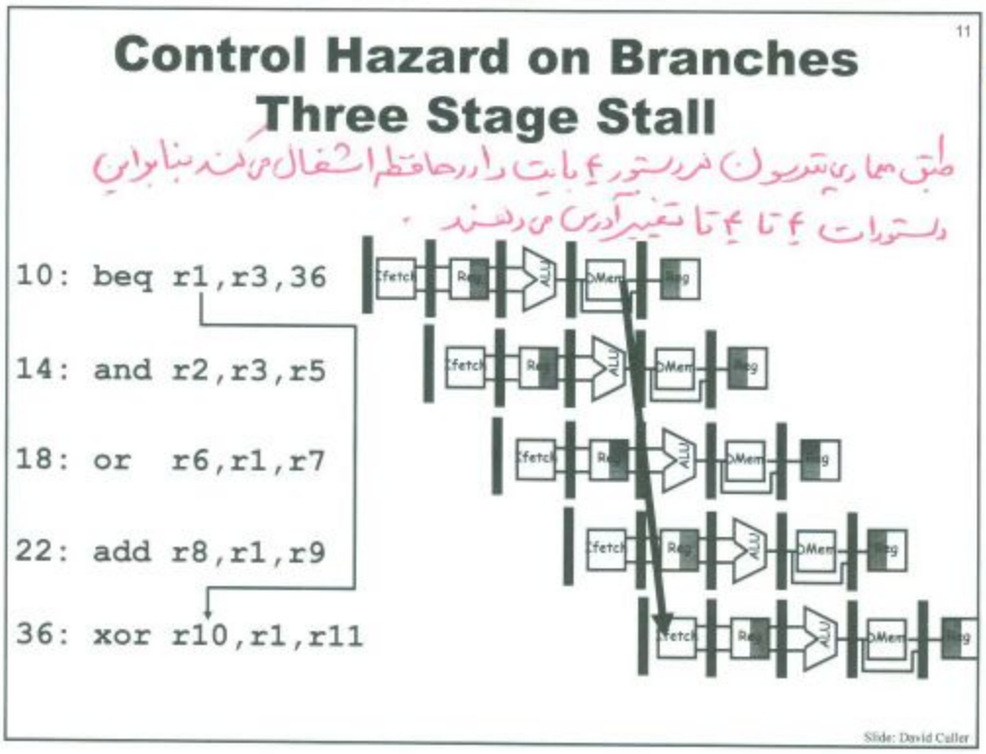
10

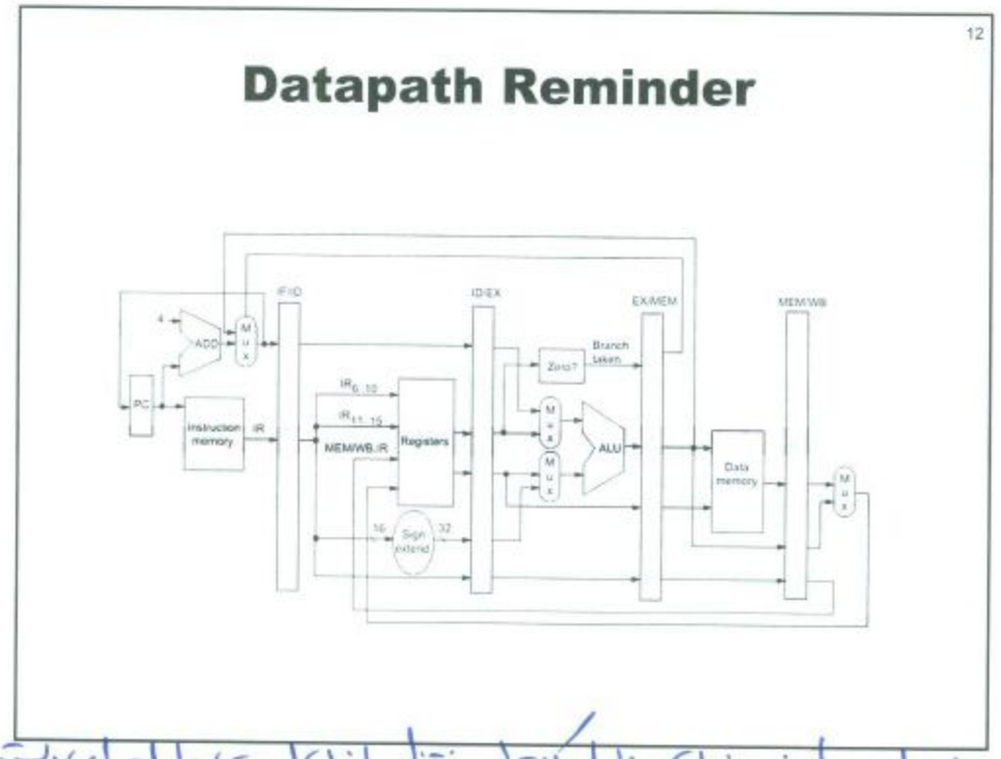
Pipeline Hazards

- Cases that affect instruction execution semantics and thus need to be detected and corrected
- Hazards types
  - Structural hazard: attempt to use a resource two different ways at same time
    - Single memory for instruction and data
  - Data hazard: attempt to use item before it is ready
    - Instruction depends on result of prior instruction still in the pipeline
  - Control hazard: attempt to make a decision before condition is evaluated
    - branch instructions
- Hazards can always be resolved by waiting

Hazard های کنترلی مخاطراتی هستند که در دستورات شرطی یا انشعاب (پرش) با آنها برخورد می‌کنیم. این نوع مخاطرات را می‌توان فقط با استفاده از روش‌های سخت‌افزاری و نرم‌افزاری بهبود بخشید و نمی‌توان آنها را از بین برد. 1

در مثال زیر بیان شده است اگر r1 برابر r3 باشد به خانه ۳۶ حافظه پرش نماید در غیر اینصورت دستور بعدی اجرا شود. این مخاطره مربوط به Hazard کنترلی خواهد بود. اگر شرط را اجرا نکنیم و دستورات را به ترتیب اجرا نمائیم در آنصورت اگر شرط برقرار بوده باشد سه دستور ۱۴، ۱۸ و ۲۲ نباید اجرا می شده اند و اگر تغییری ایجاد شده باشد باید دوباره آنرا به حالت اول برگردانیم و اگر پرش نمائیم و شرط برقرار نباشد در آنصورت اجرای سه دستور ۱۴، ۱۸ و ۲۲ را حذف کرده ایم.





ساده ترین راه حل برای برطرف نمودن مخاطره کنترلی منتظر ماندن است یا اجرای دستور شرطی که این روش بدترین روش است. روش دوم تخمین است یعنی پیش بینی کنیم که شرط درست است یا نادرست روش دیگر تخمین آماری یا پیش بینی آماری است یعنی براساس نمونه برداری های انجام شده تخمین بزنیم یعنی به صورت هوشمندانه و براساس تجربه و کارهای قبلی که انجام شده، پیش بینی را انجام دهیم. روش دیگر بستگی به نوع دستور دارد مثلاً برای دستور for بهتر آن است که پرش را انجام ندهیم در صورتیکه برای دستور if، else بهتر آن است که پرش نمائیم.

اگر دستورات شرط بیش از ۳ سیکل نیاز دارند. اگر ۳۰٪ دستورات به صورت شرط باشند و هر دستور ۳ سیکل نیاز داشته باشد، برای رفع مخاطره کنترلی دو کار باید انجام دار:

۱- تشخیص دهیم که شرط برقرار است یا نه که این کار با چک کردن پرچم ثبات انجام می‌شود.

۲- آدرس مقصد جایی که قرار است پرش به آنجا انجام شود را بدست آوریم.

13

Example: Branch Stall Impact

- If 30% branch, 3-cycle stall significant!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or ≠ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Slide: David Culler

14

Pipelined MIPS Datapath

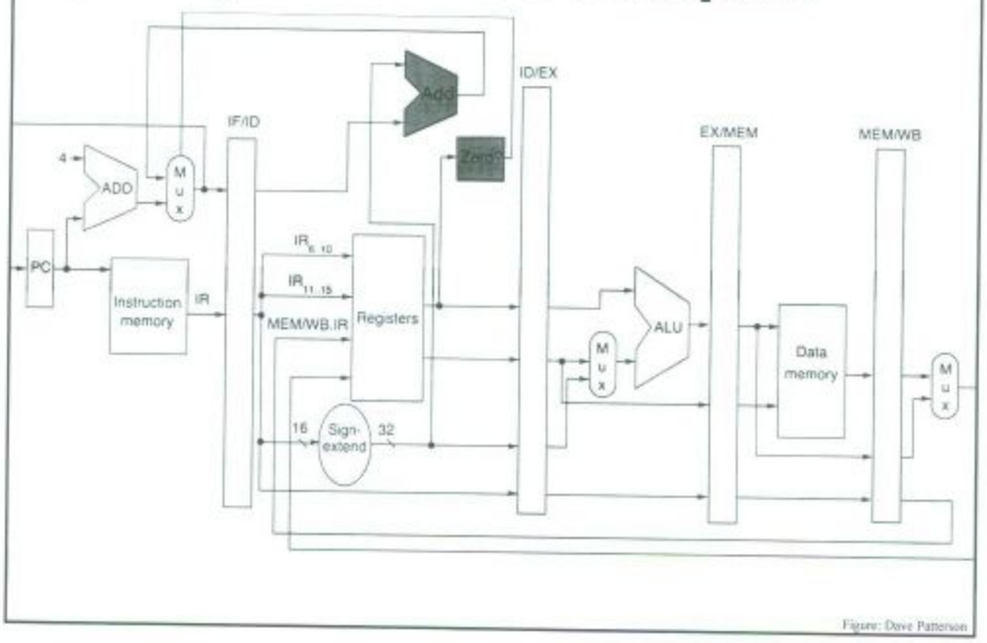


Figure: Dave Patterson

چون هر دستور در ۳ سیکل انجام می‌شود لذا در صورت اشتباه تشخیص دادن شرط مجبوریم دو سیکل به عقب برگردیم که به این کار جریمه شدن یا پنالتی گفته می‌شود. هر چه تعداد سیکل‌های مربوطه اجرای یک دستور بیشتر شود در صورت جریمه شدن تعداد سیکل بیشتری باید به عقب برگشت غالبا.

برای برخورد با Hazard های کنترلی چهار روش به صورت زیر ارائه گردیده است :

۱ - منتظر بمانیم تا ببینیم شرط درست است یا نه

۲ - با فرض برقرار نبودن شرط جلو رویم (معمولاً ۴۷٪ دستورات براساس معماری سیستم بیتربن شرط‌ها این برقرارند)

۳ - با فرض برقرار بودن شرط پیش رویم (معمولاً طبق آمار در ۵۳٪ دستورات، شرط‌ها برقرارهستند)

15

Four Branch Hazard Alternatives

1. Stall until branch direction is clear
2. Predict Branch Not Taken
   - Execute successor instructions in sequence
   - "Squash" instructions in pipeline if branch taken
   - Advantage of late pipeline state update
   - 47% MIPS branches not taken on average
   - PC+4 already calculated, so use it to get next instruction
3. Predict Branch Taken
   - 53% MIPS branches taken on average
   - But haven't calculated branch target address in MIPS
     - MIPS still incurs 1 cycle branch penalty
     - Other machines: branch target known before outcome

Slide: David Culler

۴ - براساس حالت های قبلی در دستورات قبلی از یک تخمین استفاده می کند، یعنی مثلاً اگر if بود از فرض برقرار نبودن شرط پیش می رود.

16

Four Branch Hazard Alternatives

4. Delayed Branch
   - Define branch to take place AFTER a following instruction

```
branch instruction
        sequential successor1
        sequential successor2
        ........                    } Branch delay of length n
        sequential successorn
.....
branch target if taken
```

   - 1 slot delay allows proper decision and branch target address in 5 stage pipeline
   - MIPS uses this

Slide: David Culler

رابطه‌ای که برای بدست آوردن speedup مربوط به pipeline استفاده می‌شود به صورت زیر است:

$$\text{pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{pipeline stall CPI}}$$

(طول پایپ لاین (چند مرحله است)

تعداد مراحل که باید منتظر بمانیم تا دستورات pipeline کامل شود.

17

Branch-Delay Scheduling Requirements

Scheduling Strategy	Requirements	Improves performance when?
(a) From before	Branch must not depend on the rescheduled instructions	Always
(b) From target	Must be OK to execute rescheduled instructions if branch is not taken. May need to duplicate instructions.	When branch is taken. May enlarge programs if instructions are duplicated.
(c) From fall through	Must be okay to execute instructions if branch is taken.	When branch is not taken.

- Limitation on delayed-branch scheduling arise from:
  - Restrictions on instructions scheduled into the delay slots
  - Ability to predict at compile-time whether a branch is likely to be taken
- May have to fill with a no-op instruction
  - Average 30% wasted
- Additional PC is needed to allow safe operation in case of interrupts (more on this later)

18

Example: Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}}$$

$$= \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Assume:
14% Conditional & Unconditional
65% Taken; 52% Delay slots not usefully filled

Scheduling Scheme	*Branch Penalty*	*CPI*	*Pipeline Speedup*	*Speedup vs stall*
Stall pipeline	3.00	1.42	3.52	1.00
Predict taken	1.00	1.14	4.39	1.25
Predict not taken	1.00	1.09	4.58	1.30
Delayed branch	0.52	1.07	4.66	1.32

Slide: David Culler

اگر بخواهیم براساس تخمین جلو برویم خواهیم داشت:

$$\text{pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

چند درصد دستورات به صورت انشعاب هستند

چند بار مجبور هستید بدلیل اینکه تخمین نادرست زده اید به عقب برگردید.

(تعداد سیکل هایی است که اگر دستورات pipeline را به اشتباه جلو رفته‌ایم باید برگردیم)

پنجشنبه - ۹۴/۱/۲۷

Multiprocessors

سیستم های چند پردازنده ای

اولین ایده برای استفاده از سیستم های چند پردازنده ای این بود که به جای استفاده از یک پردازنده قوی با حجم پردازش بالا، از چند پردازنده کوچکتر با در کنار هم قرار دادن آنها از توان پردازشی مجموع آنها برای رسیدن به هدف استفاده کنیم.

پردازنده قوی و بزرگ ≡ استفاده از چندین پردازنده کوچک در جهت رسیدن به توان پردازشی بالا

این پردازنده ها باید بتوانند دستورات را به صورت موازی انجام دهند. در pipeline نیز دستورات همزمان اجرا نمی شد بلکه روند اجرای دستورات سریعتر بود.

یکی از ویژگی هایی که سیستم ها را به صورت چند پردازنده ای استفاده می نمایند، قابلیت دسترس پذیری یا Availability آنهاست. در زمانیکه از چند پردازنده استفاده نمائیم در اثر از کار افتادن یکی از آنها سیستم هرچند با توان کمتر، ولی به کار خود ادامه خواهد داد در صورتیکه اگر از یک پردازنده قوی استفاده نمائیم، در صورت بروز اشکال در آن، سیستم Down یا متوقف خواهد شد. پس مهمترین خصوصیت استفاده از سیستم های چند پردازنده ای، خصوصیتی بنام قابلیت دسترس پذیری است. لازم به ذکر است مدیریت سیستم های چند پردازنده به نسبت تک پردازنده مشکل تر است.

لازمه استفاده از حداکثر کارآیی یک سیستم چند پردازنده ای، موازی سازی در سطح برنامه هایی می باشد که می باید توسط آن اجرا شوند. یعنی در واقع زمانی که می خواهیم از سیستم چند پردازنده ای استفاده نمائیم باید بتوانیم دستوراتی را که قرار است توسط این پردازنده اجرا شوند به صورت موازی اجرا نمائیم وگرنه نمی توانیم از حداکثر توان پردازنده ها استفاده کنیم. اگر بتوانیم دستورات خود را ۱۰۰٪ موازی سازی نمائیم آنگاه می توانیم از ۱۰۰٪ توان سیستم چند پردازنده ای خود استفاده نمائیم.

در سطح سخت افزار ما با برنامه سروکار نداریم بلکه با دستورات سروکار داریم. در یک لول بالاتر در سطح سیستم عامل با دو قسمت سروکار داریم پس موازی سازی می تواند در دو سطح انجام شود.

job (کاری که به سیستم عامل می دهیم تا سیستم عامل آن را مدیریت نموده و اجرا نماید.

process (هر job از تعدادی پروسس یا پردازش تشکیل شده است)

پس بنا براین هرچه بتوانیم job های بیشتری را به صورت موازی اجرا نمائیم، اجرای ما سریعتر خواهد بود.

مقیاس پذیری:

در صورتیکه روشی را که برای اجرای یک کار ارائه می کنیم با بزرگتر شدن کار باز هم قابل اجرا باشد گفته می شود که این روش مقیاس پذیر است. یکی از ویژگی های سیستم های چند پردازنده ای مقیاس پذیری است اما همیشه مقیاس پذیری انجام نمی شود.

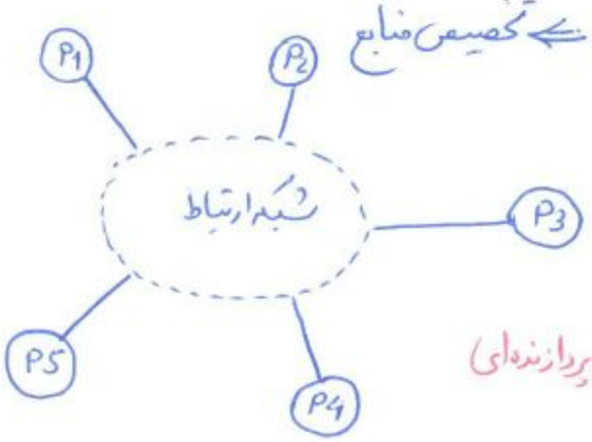
* مقیاس پذیری در صورت مقیاس پذیری موازی سازی نرم افزارها و برنامه های قابل اجرا به روی یک سیستم چند پردازنده، برای یک سیستم چند پردازنده امکان پذیر است. به عبارتی دیگر اگر برنامه بتواند خود را با هر تعداد از پردازنده ها تطبیق دهد، سیستم چند پردازنده ای دارای ویژگی مقیاس پذیری است.

اکثر سیستم های چند پردازنده ای موفق سیستم هایی هستند که به منظورهای خاصی طراحی شده اند زیرا سیستم های چند پردازنده ای کاملاً وابسته به نرم افزارهایی هستند که بر روی آنها اجرا می شوند یعنی کارآیی یک سیستم چند پردازنده کاملاً وابسته به موازی سازی برنامه هایی است که بر روی آن اجرا می شوند.

پس از مسئله موازی سازی مهمترین مسئله ای که در استفاده از سیستم های چند پردازنده ای مطرح است مسئله زمان بندی است یعنی اینکه اگر قرار است مثلاً دستورات توسط ۱۰۰ پردازنده به صورت موازی انجام شوند، کدام یک از دستورات جهت اجرا بر روی پردازنده ها قرار داده شوند یا زمانیکه از پردازنده هایی با توان های اجرای متفاوت استفاده می کنیم، کدام قسمت از برنامه را بر روی کدام یک از پردازنده ها اجرا نمائیم تا کارآیی سیستم بالاتر باشد.

پس از زمان بندی باید تخصیص منابع انجام دهیم یعنی ممکن است منابع متفاوتی وجود داشته باشد، حال اینکه کدام منبع را در هر زمانی در اختیار کدام پردازنده قرار دهیم که می تواند در کارآیی سیستم مؤثر باشد.

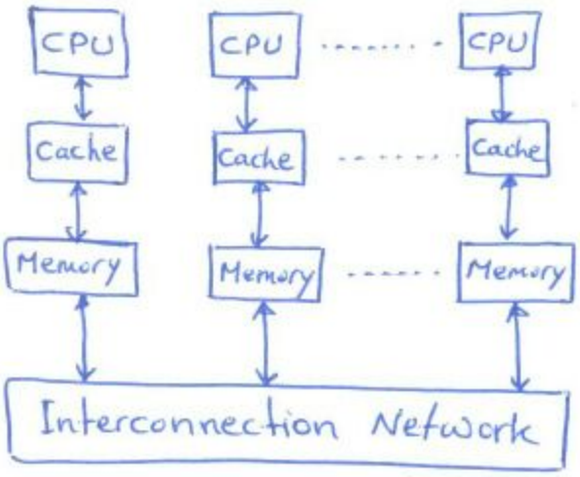
موازی سازی ⟸ زمان بندی ⟸ تخصیص منابع



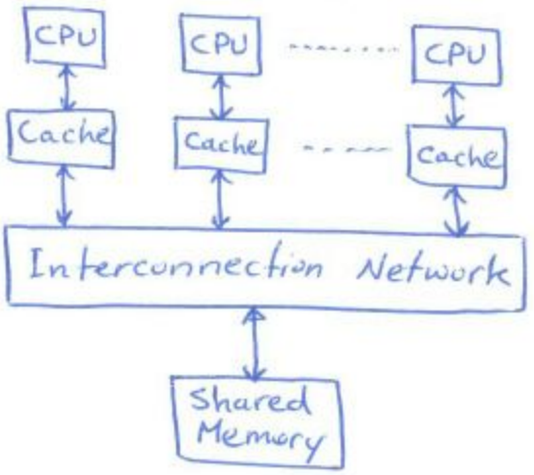
یک سیستم چند پردازنده ای

در سیستم های چند پردازنده ای، استفاده از حافظه می تواند به دو صورت زیر باشد. توجه شود که در هر دو روش، همه پردازنده ها دارای حافظه اختصاصی مربوط به خود بنام Cache می باشند.

Clusters Memory Multiprocessor (CMP)



Shared Memory Multiprocessor (SMP)



سنتی کلاسیک (مبتنی بر تبادل پیام)

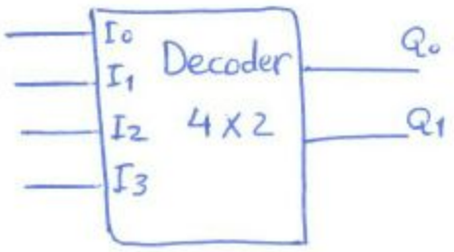
در حالت استفاده از حافظه مشترک، فضای حافظه به طور منطقی بین تمام پردازنده ها تقسیم می شود. با توجه به آنکه حافظه فقط دارای یک باس آدرس دهی می باشد لذا در یک زمان فقط یکی از پردازنده ها می تواند به آن دسترسی داشته باشد یعنی بر روی باس آدرس فقط می توان آدرس یک خانه را قرار داد که از آن خوانده یا نوشته شود. پس در هر زمان فقط یکی از پردازنده ها به حافظه دسترسی دارد.

در روش cluster که بر اساس سیستم مبتنی بر پیام می باشد، هر پردازنده علاوه بر حافظه Cache دارای یک حافظه اختصاصی بوده که دستورات را از آنجا می خواند و پس از اجرای دستورات، اطلاعات را مجدداً در آنجا ذخیره می کند. از طریق یک شبکه اتصال داخلی، پردازنده ها به حافظه های همدیگر دسترسی داشته و می توانند از اطلاعات ذخیره شده در آنها استفاده نمایند. این روش همانند استفاده از چند کامپیوتر مستقل برای رسیدن به یک توان پردازشی بالاتر می باشد که از طریق یک شبکه ارتباطی به یکدیگر متصل شده اند.

یکی از روش های دسترسی به حافظه مشترک یا منابع به صورت Random می باشد. در اینحالت اولویت بندی جهت دسترسی لحاظ نمی گردد. روش دیگر به صورت ترتیبی است که در اینحالت از یک دیکودر معمولی می توان استفاده نمود. روش دیگر استفاده از Encoder یا اولویت می باشد. مشخص است که روش اولویت بندی کارایی سیستم را بالاتر می برد ولی دسترسی به این روش پیچیده تر است.

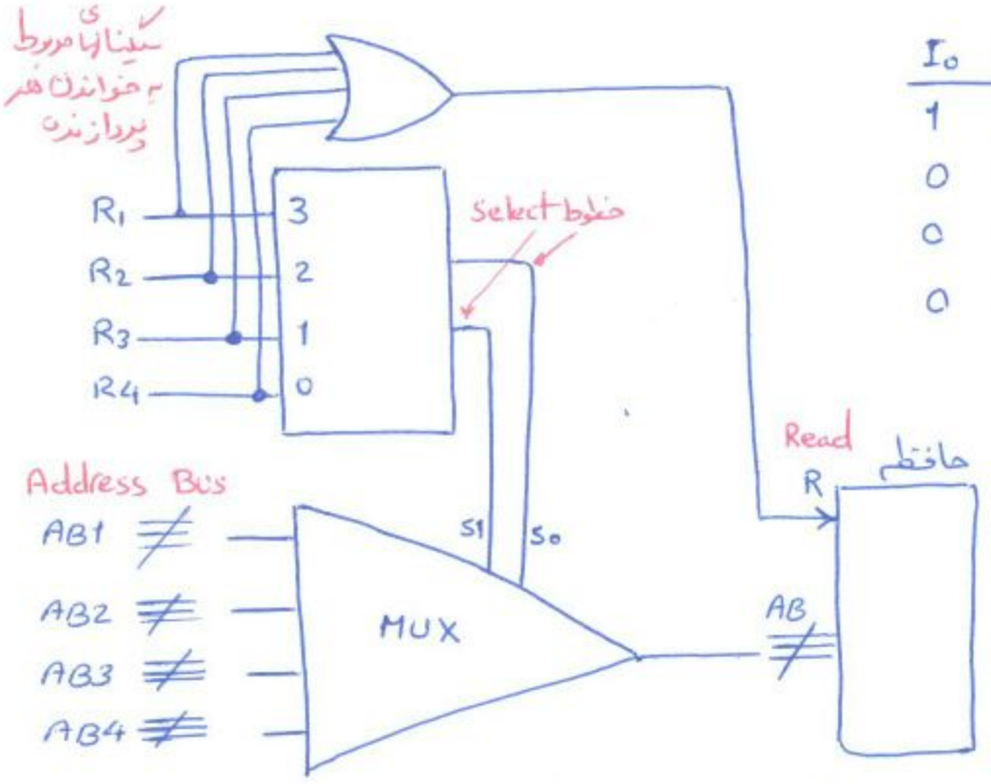
روش دیگر دسترسی به حافظه به این صورت است که اگر مثلاً تمام پردازنده ها همزمان نیاز به دسترسی به حافظه داشته باشند، اولویت با پردازنده شماره پائین تر باشد.

یکی از روش های سخت افزاری به صورت مدار زیر می باشد. در این روش می توان حافظه را به عنوان یک منبع ایجاد وقفه نیز در نظر گرفت.



I_0	I_1	I_2	I_3	Q_0	Q_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

اشکالی که Decoder معمولی دارد آن است که اگر همزمان دو ورودی فعال وجود داشته باشد جواب غلط خواهد



I_0	I_1	I_2	I_3	Q_0	Q_1
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	1	0	0

Encoder با اولویت

با استفاده از این روش سخت افزاری، می توان دسترسی به حافظه با اولویت ایجاد نمود.

دسترسی به حافظه در SMP

UMA: Uniform Memory Access

زمان دسترسی به حافظه در تمام پردازنده ها یکسان و یکنواخت است. برنامه نویسی آن به لحاظ زمان بندی ساده است. دسترسی به حافظه به صورت سخت افزاری غیر باشر بلکه به صورت Random است.

NUMA: Non-Uniform Memory Access

پردازنده ها در دسترسی به حافظه نسبت به یکدیگر دارای اولویت می باشند که زمان دسترسی به حافظه را برای آنها متفاوت می کند. برنامه نویسی آن سخت تر است.

هرچه دسترسی به حافظه به صورت یکسان و یکنواخت باشد امکان برخورد و خطا بیشتر است لذا امکان گسترش پذیری در NUMA بیشتر است.

این نوع ساختارها را ساختارهای مبتنی بر پیام می گویند زیرا پردازنده مستقیماً به حافظه دسترسی ندارد بلکه هر اطلاعاتی را که نیاز داشته باشد با ارسال یک پیام، حافظه یک کپی از آن را برای پردازنده ارسال می نماید.

نکته ای که باید توجه خود آن است که مدیریت سیستم هایی که از حافظه های مجزا استفاده می نمایند دشوارتر از حافظه های مشترک است.

در مالتی کرها (Multi cores) چندین پردازنده بر روی یک تراشه قرار گرفته اند و تکنولوژی آن از مالتی پروسسورها جدیدتر است ولی منطق همان منطق است.

NOC = Network on chip

شبکه ای از پردازنده ها که روی یک chip قرار گرفته اند.

تفاوتی که بین NOC ها با مالتی کرها وجود دارد آن است که در NOC ها از چندین پردازنده مختلف که جهت استفاده کارهای مختلف می باشند بر روی یک chip استفاده می گردد. NOC ها برای کاربردهای خاص استفاده می شوند مثلاً برای کاربردهای مخابراتی یا تصویری. در NOC موازی سازی را می توان ساده تر انجام داد.

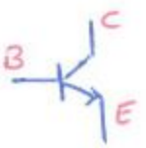
* مالتی کرها از یک حافظه مشترک استفاده می کنند.

کارت گرافیک نیز به صورت یک پردازنده است که جهت پردازش تصویر به صورت یک چند پردازنده عمل می کند.

لذا در کامپیوترها امکان ترکیب مالتی کرها با مالتی پروسسورها وجود دارد که در اینحالت نیز امکان استفاده از حافظه های مجزا یا مشترک وجود دارد.

پنجشنبه ۹۴/۲/۳

حافظه ها



بین ترانزیستورهایی که در نسل اول مورد استفاده قرار می گرفتند از ترانزیستورهای BJT بودند. این ترانزیستورها دارای سرعت بالایی بودند ولی مصرف توان بسیار زیادی نیز داشتند که به همین علت باعث شد تا از نوع دیگر از ترانزیستورها به نام Mos fet استفاده گردد.

ترانزیستورهای Mos fet در حالت استاتیک توان مصرفی صفر دارد و تنها توان مصرفی آن در زمان تغییر حالت قطع به وصل و برعکس می باشد.

در ترانزیستورهای BJT معیار ولتاژ بود و دارای دو حالت قطع و وصل بودند. در حالت وصل امیتر به کلکتور وصل شده و اختلاف ولتاژ بین آنها صفر می شود و در حالت قطع، اختلاف ولتاژ بین امیتر و کلکتور برابر ولتاژ منبع تغذیه می باشد. در Mos fet ها معیار جریان می باشد بدین صورت که در حالت وصل، یک جریان بین درین و سورس ایجاد می شود که در حالت قطع مقدار این جریان تقریباً صفر است.

در Mos fet ها در حالت وصل دو حالت داریم: تریود / اشباع

تریود: در این حالت جریان عبوری بین درین و سورس بر اساس جریان گیت کنترل می شود و هرچه جریان گیت بیشتر شود، جریان عبوری بین درین و سورس نیز بیشتر می شود.

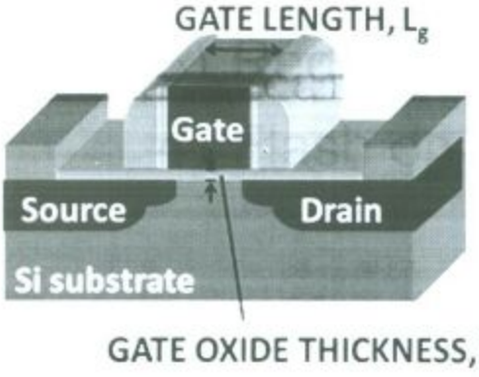
اشباع: در این حالت مقدار جریان عبوری بین درین و سورس مقدار ثابتی بوده و هرچه جریان گیت را بیشتر نماییم، این جریان تغییری نمی کند و در واقع به حالت اشباع رسیده است.

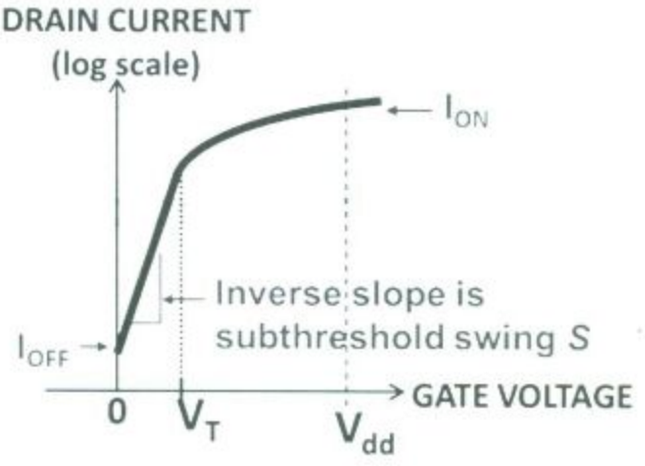
هرچه بتوانیم در یک فضای مشخص تعداد ترانزیستور بیشتری را بر روی یک تراشه ایجاد نماییم در واقع مقدار گیت بیشتری خواهیم داشت و هرچه تعداد گیت بیشتری داشته باشیم می توان محاسبات بیشتری را انجام دهیم و هرچه در یک فضای مشخص بتوانیم محاسبات بیشتری را انجام دهیم، توان پردازشی ما بیشتر خواهد بود. برای رسیدن به توان پردازشی بالاتر یکی از کارها، استفاده از قطعات ریزتر در سیستم های پردازشی به صورت موازی می باشد.

eetimes

1) IBM Fabrication
2) Reneseas: big/little
3) Reducing AMD power with gating
4) Flash on DDR4

Metal Oxide Semiconductor
Field-Effect Transistor:





DARPA UPSIDE

[August 17, 2012]

Digital Processors Limited by Power; What's The Upside?

(Targeted News Service Via Acquire Media NewsEdge) WASHINGTON, Aug. 14 -- The U.S. Department of Defense's Defense Advanced Research Projects Agency issued the following news release: Today's Defense missions rely on a massive amount of sensor data collected by intelligence, surveillance and reconnaissance (ISR) platforms. Not only has the volume of sensor data increased exponentially, there has also been a dramatic increase in the complexity of analysis required for applications such as target identification and tracking. The digital processors used for ISR data analysis are limited by power requirements, potentially limiting the speed and type of data analysis that can be done. A new, ultra-low power processing method may enable faster, mission critical analysis of ISR data.

fa

Chapter Outline

- Introduction
- Parallel Computers
- Shared-Memory Programming
- Synchronization
- Cache Coherence

Introduction

- Growth in data-intensive applications.
  - Data bases, file servers, …
- Growing interest in servers, server performance.
- Increasing desktop performance less important.
  - Outside of graphics
- Improved understanding in how to use multiprocessors effectively.
  - Especially servers where significant natural TLP
- Advantage of leveraging design investment by replication => CMPs or Multicores.
  - Rather than unique design

عواملی که باعث نیاز به پردازنده‌های سریع می‌باشند :

۱- اولین چیزی که باعث نیاز به پردازنده‌های سریع بود اپلیکیشن‌ها یا برنامه‌های مبتنی بر دیتابود مانند دیتا سنترها یا اپلیکیشن‌هایی که نیازمند پردازش و جستجوی سریع هستند .

۲- علاقه بیشتر به افزایش بهره برداری از سرورهای سریع

۳- در کامپیوترهای شخصی در زمانی که از گرافیک و پردازش تصویر استفاده می‌کنیم نیاز به پردازنده‌های سریعتر می‌باشد .

همه این عوامل باعث شده تا ما سیستم را بنام سیستم‌های مالتی پروسسور یا چند پردازنده معرفی نمائیم که مهمترین مزایای آن این است که الزامات ما را که همان انجام پردازش سریع می‌باشد را فراهم می‌نماید .

محدودیت‌هایی که برای کارآیی پردازنده وجود دارد :

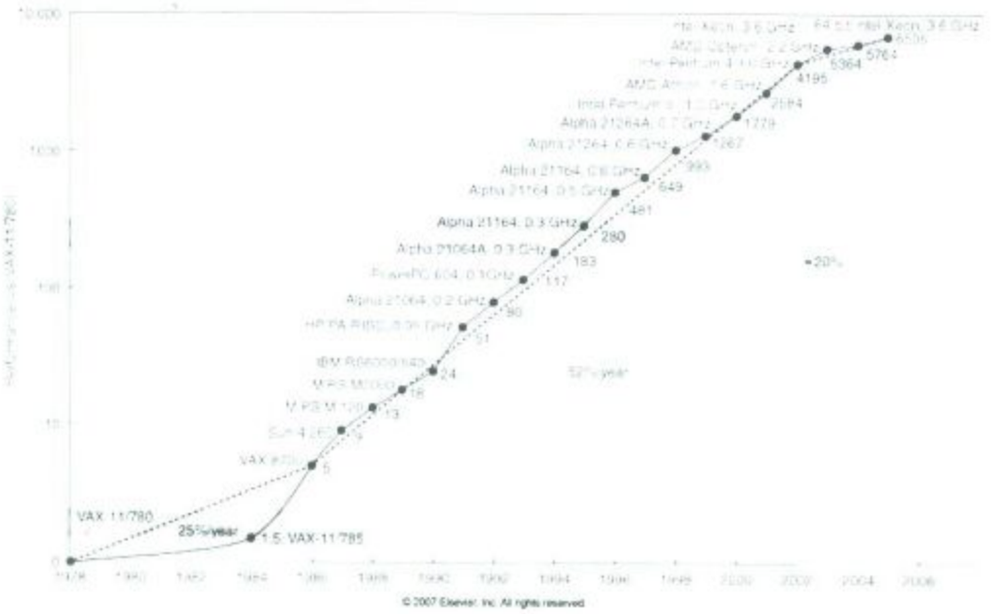
۱- پیچیدگی : هرچقدر ترانزیستورهای بیشتری ایجاد نمائیم به لحاظ کنترلی و مدارات ALU پیچیده‌تر خواهند بود و به همین دلیل پشتیبانی کردن از چنین ساختاری دشوار است .

۲- تعداد پین‌های ورودی به هر chip : تکنولوژی ساخت chip ها تا یک حدی به ما اجازه می‌دهد که تعداد پین‌های ورودی به chip را افزایش دهیم .

۳- خنک کردن و بسته‌بندی هر پردازنده که می‌تواند مشکلاتی را در ایجاد تبادل حرارت ایجاد کند . هرچه ترانزیستورهای بیشتری در chip به کار رفته باشد، پردازش‌های بیشتری را انجام می‌دهد و به همین نسبت حرارت بیشتری نیز ایجاد می‌شود که خنک کردن آن به صورت یک معضل می‌باشد .

۴- کارآیی پردازنده‌ها در استفاده از حافظه : زیرا حافظه‌ها کندترین عنصر در سیستم‌ها می‌باشند لذا حتی اگر پردازنده‌ای بسیار سریع ایجاد نمائیم بدلیل استفاده از دیتاهایی که در حافظه‌های کند وجود دارند، سرعت پردازنده نمی‌تواند از یک حد مشخص بیشتر شود و همانطور که گفته شد، رشد سرعت حافظه‌ها چندین برابر کندتر از رشد سرعت پردازنده‌ها بوده است لذا سرعت حافظه باعث ایجاد محدودیت می‌گردد .

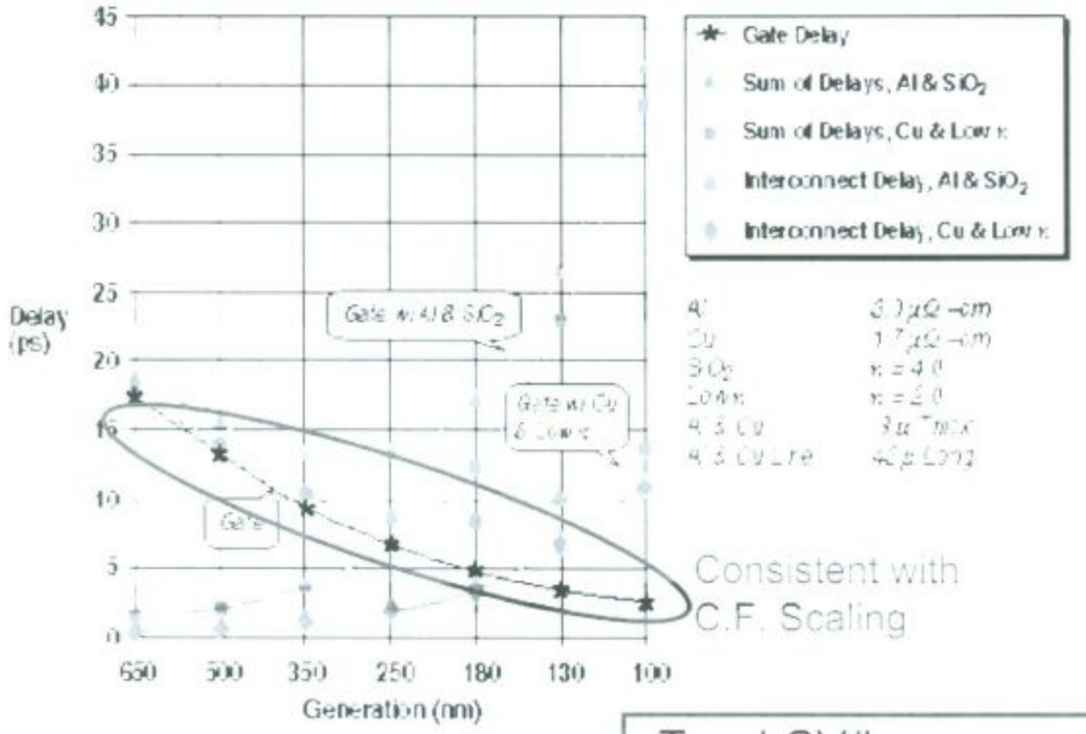
Another Reason



Limit to Processor Performance

- Complexity of exploiting ILP.
  - Difficult to support large instruction window and number of in-flight instructions.
- On-chip wires are becoming slower than logic gates.
  - Only a fraction of the die will be reachable within a single clock cycle.
- Cooling and packaging will be a real challenge due heat release.
- Memory and processor performance gap will continue to be a challenge.

Gate Delay Trends

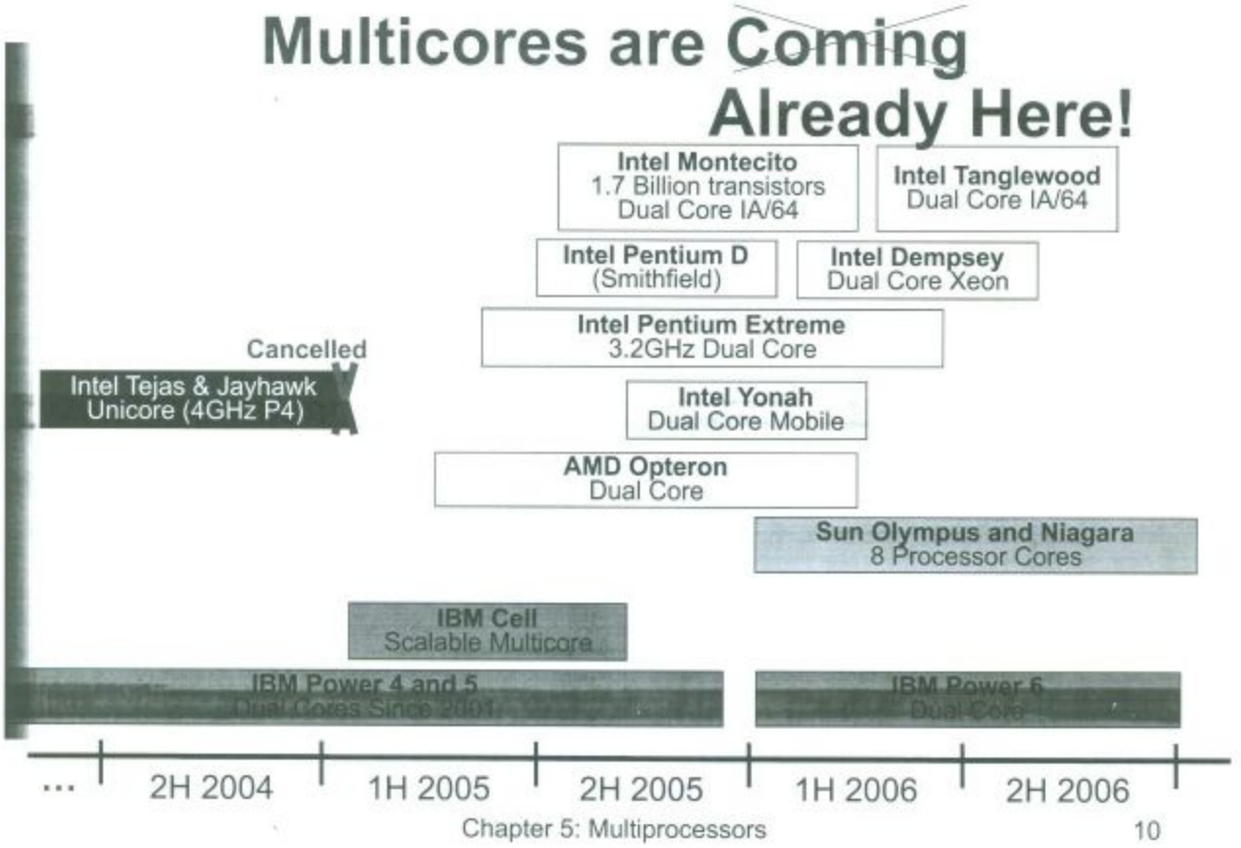


Consistent with C.F. Scaling

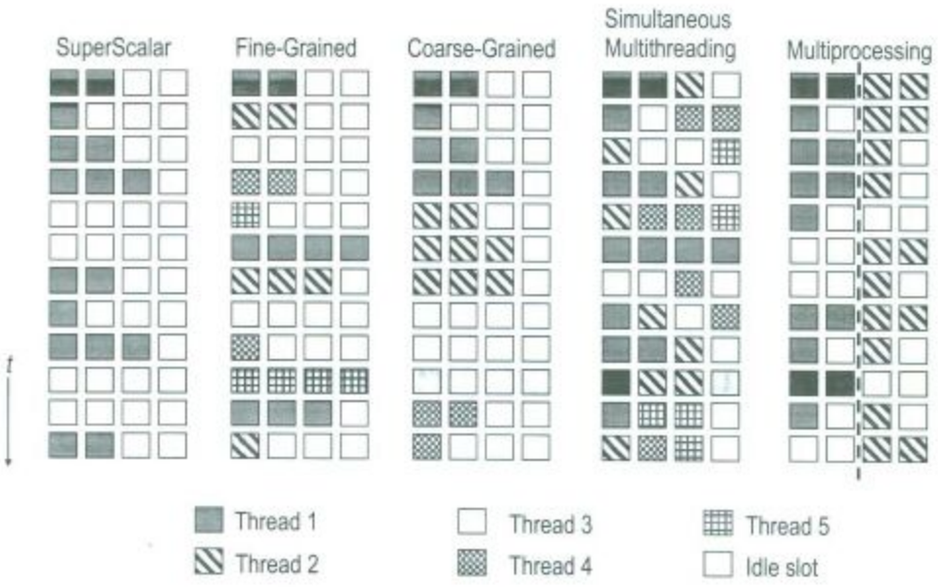
Each technology generation, gate delay reduced about 30% (src: ITRS '01)

$$T_d = kCV/I$$
$$= kCV/(V_{dd}-V_t)^{\alpha}$$

Multicores are ~~Coming~~ Already Here!



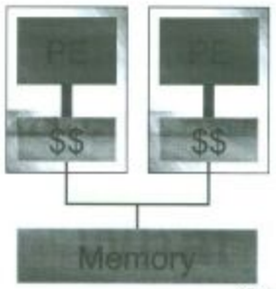
SS, MT, SMT, & MP



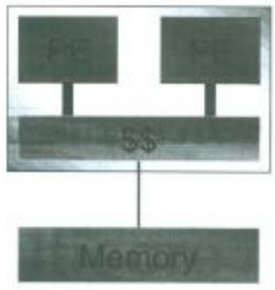
در مالتی پروسسور متنی هر پردازنده حافظه Cache مربوط به خودرا دارد و تمام اینها به حافظه اصلی به صورت اشتراکی در ارتباط هستند اما در مالتی کرها تمام هسته‌ها دارای یک Cache بوده و همگی آنها با یک حافظه اصلی (RAM) در ارتباط می‌باشند.

What is a Multicore?

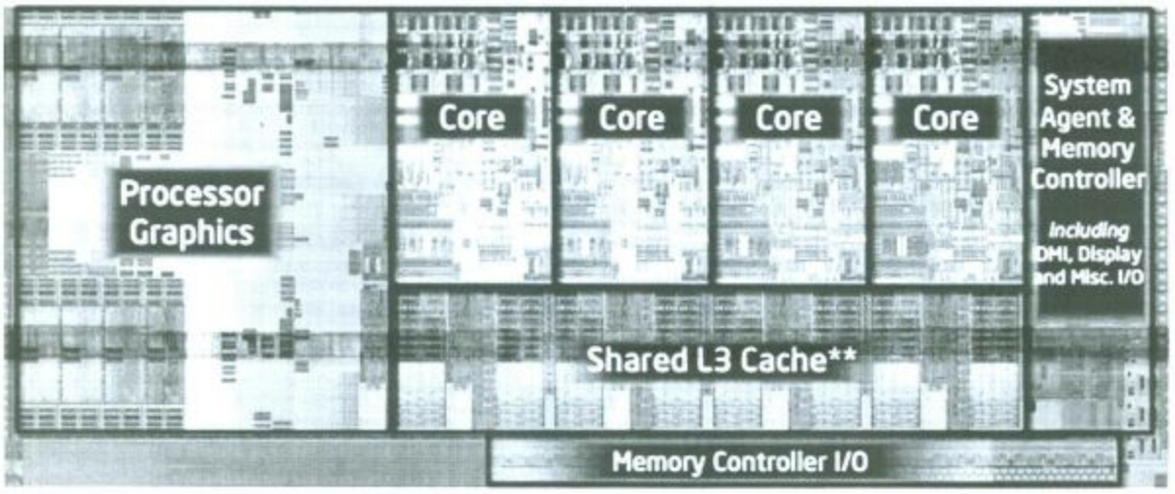
Traditional Multiprocessor



Basic Multicore IBM Power5

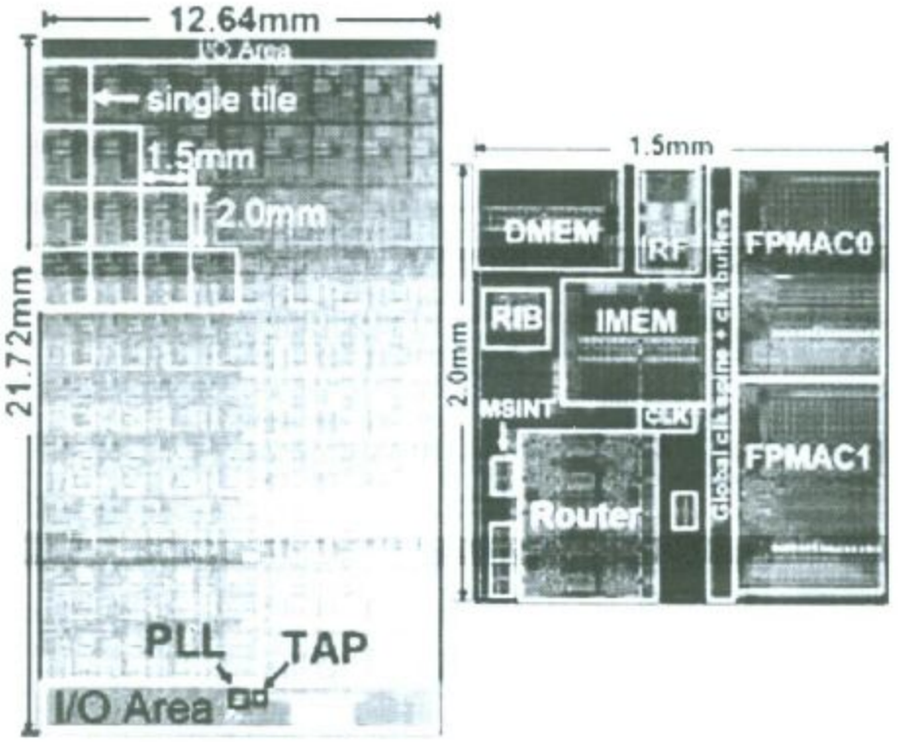


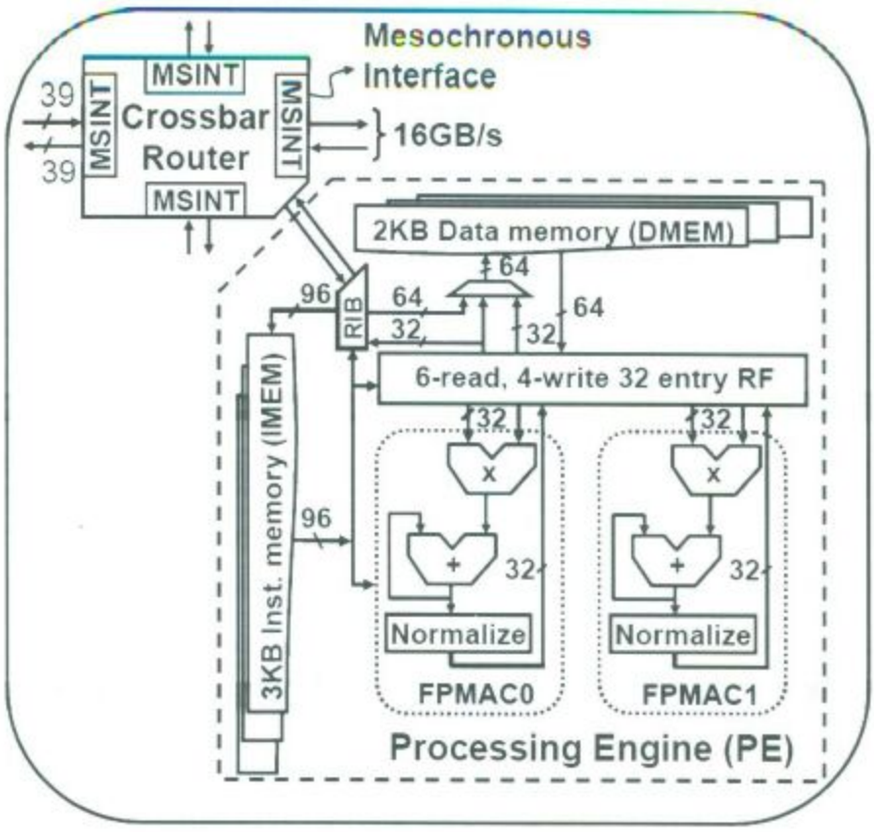
3rd Generation Intel® Core™ Processor: 22nm Process

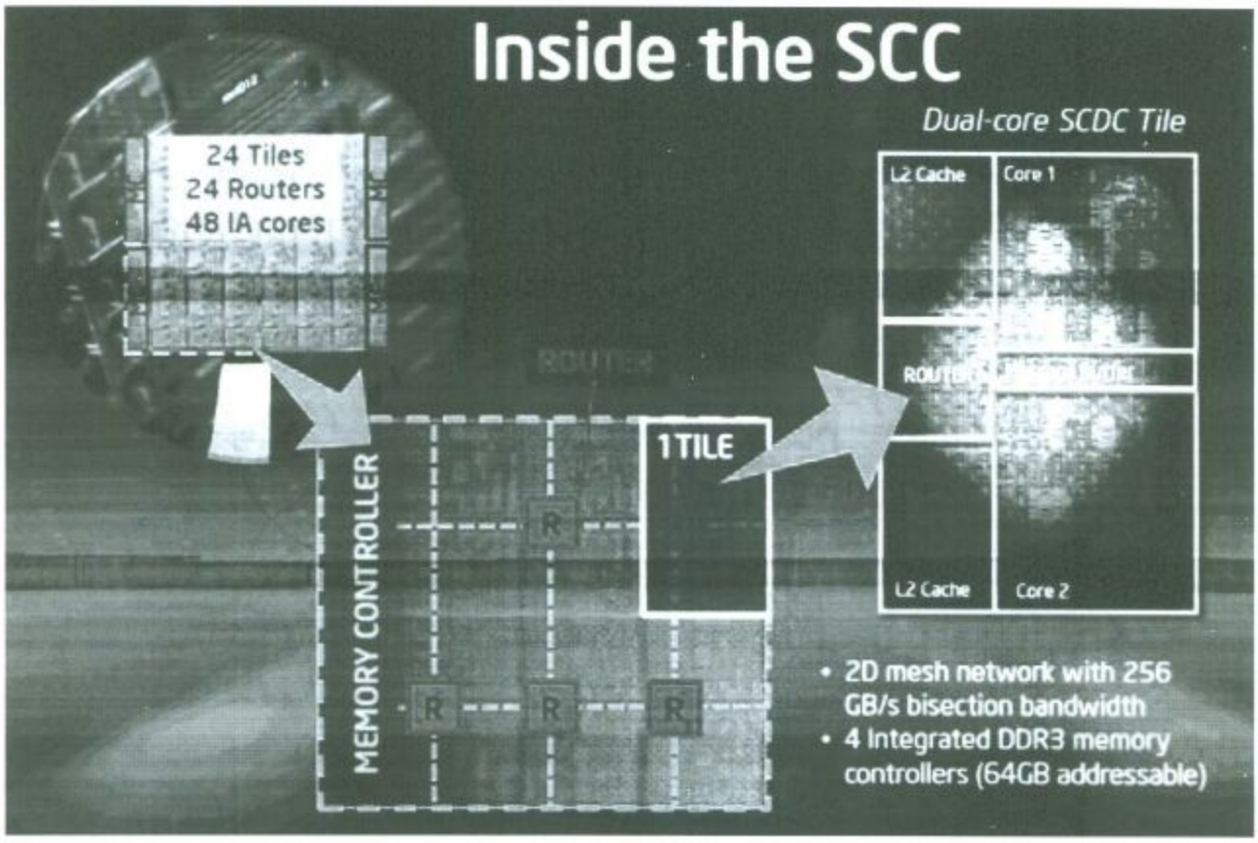


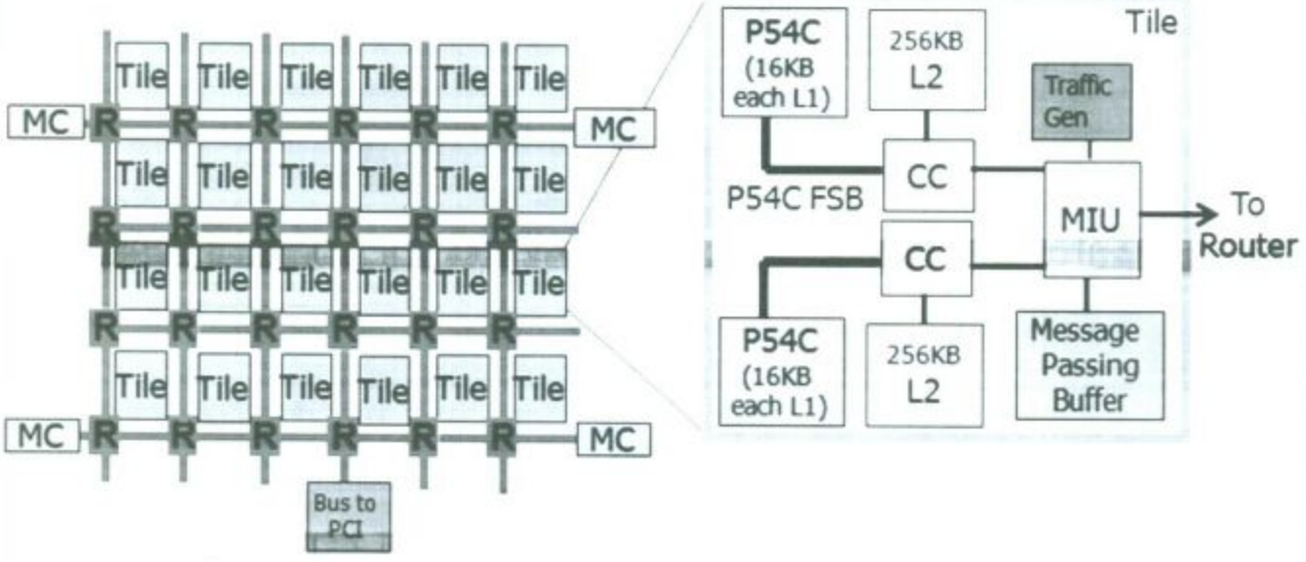
New architecture with shared cache delivering more performance and energy efficiency

Quad Core die with Intel® HD Graphics 4000 shown above
Transistor count: 1.4Billion Die size: 160mm²
** Cache is shared across all 4 cores and processor graphics




Mesochronous
Interface
MSINT
MSINT
MSINT
MSINT
Crossbar
Router
39
39
16GB/s
2KB Data memory (DMEM)
64
96
64
32
32
64
RIB
6-read, 4-write 32 entry RF
32
32
X
+
32
32
Normalize
Normalize
FPMAC0
FPMAC1
96
3KB Inst. memory (IMEM)
Processing Engine (PE)


Inside the SCC
Dual-core SCDC Tile
24 Tiles
24 Routers
48 IA cores
1TILE
MEMORY CONTROLLER
L2 Cache
Core 1
L2 Cache
Core 2
• 2D mesh network with 256 GB/s bisection bandwidth
• 4 Integrated DDR3 memory controllers (64GB addressable)



سیستم های مالتی پروسسور و مالتی کُر در عمل با هم تفاوتی ندارند اما در پیاده سازی کاملاً با یکدیگر متفاوت می باشند .

سؤال : چرا به جای مالتی پروسسورها از مالتی کُرها استفاده می نمائیم ؟

۱- انتشار اطلاعات بروزرسانی بُردها بین پردازنده های مالتی پروسسور باعث اتلاف توان می شود در صورتیکه بر روی chip ها بُردی وجود ندارد لذا مصرف توان نیز بسیار کم خواهد بود . وقتی از مالتی پروسسورها استفاده می کنیم عناصری مانند المانهای کنترلی ، op ، تقسیم کننده جریان ، نویزگیر و ... قرار دارد در صورتیکه در chip ها از اینها استفاده ای نمی گردد .

۲- تأثیر و کارآیی مالتی کُرها بهتر از مالتی پروسسورها می باشد زیرا بدلیل کاهش ارتباطات و هزینه در مالتی کُرها کارآیی بهتر است .

۳- مالتی پروسسورها دارای پیچیدگی بیشتری هستند و مدیریت آنها بدلیل استفاده از منابع ، حافظه و همزمان کردن پردازنده دشوارتر است .

۴- تأخیر مربوط به سیم ها و ارتباطات در مالتی پروسسورها بیشتر است در صورتیکه در مالتی کُرها تأخیری وجود ندارد .

۵- تمایل به بهینه کردن اجرای دستورات مجزا را در مالتی کُرها موازی سازی در سطح عالی تری صورت می گیرد و جریانی از دستورات را بهتر می توان به صورت موازی اجرا نمود .

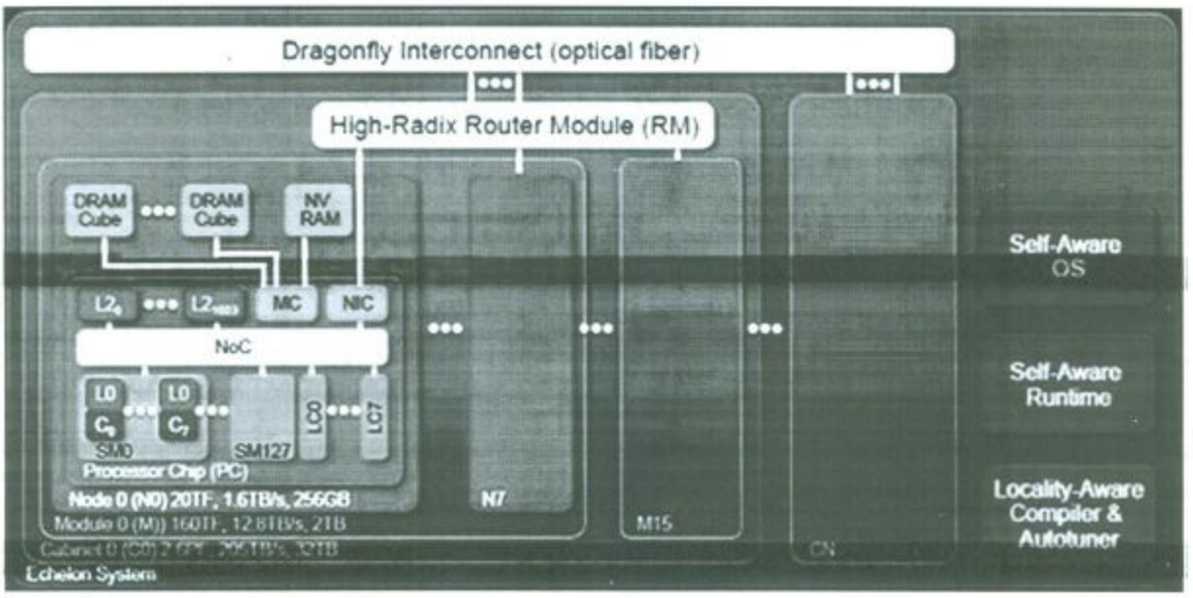


Figure 3: nVIDIA's Echelon Architecture

Why Move to Multicores

- Many issues with scaling a unicore
  - Power
  - Efficiency
  - Complexity
  - Wire Delay
  - Diminishing returns from optimizing a single instruction stream

همانطور که مشاهده می‌شود سرعت انتقال اطلاعات در حالتی که حدود ۱۰ هزار برابر حالتی پردازنده است. در حالتی که‌ها تعداد پین‌های ارتباطی درون chip بیشتر است لذا عملاً گلوگاه ارتباطی وجود ندارد و در واقع تراکم ارتباطات بیشتر بوده و به همین نسبت در یک سیکل، داده‌های بیشتری انتقال داده می‌شود.

همچنین در حالتی که‌ها مقدار clock Rate به نسبت حالتی پردازنده‌ها بیشتر است لذا در یک clock Rate می‌توان اطلاعات بیشتری را منتقل نمود.

رسانایی موادی که در حالتی که‌ها برای ارتباطات به کار می‌رود، دانسیته بالاتری دارد.

در حالتی پردازنده‌ها وقتی دیتا می‌خواهد از پردازنده خارج شده و وارد بافر یا بُرد گردد، این دو دارای دو استاندارد مختلف می‌باشند. همچنین وقتی اطلاعات می‌خواهد از بُرد واحد پردازنده گردد نیز دارای استانداردهای مختلفی می‌باشند. لذا در اینحالت باید عمل تبدیل یا مالتی پلکس انجام شود که این کار باعث ایجاد تأخیر می‌گردد در صورتیکه در حالتی که‌ها چنین چیزی وجود ندارد.

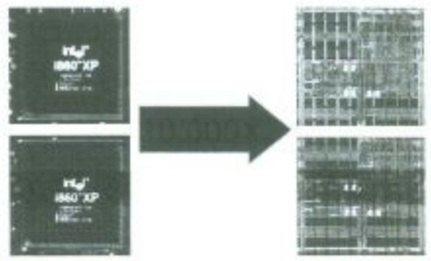
سؤال: با استفاده از حالتی که توانستیم سرعت انتقال داده داده پردازنده‌ها افزایش دهیم. حال این تغییر چه تأثیری بر روی برنامه نویسی ما دارد؟

با توجه به اینکه در حالتی که‌ها سرعت انتقال داده بین پردازنده‌ها بسیار زیاد می‌باشد لذا برای اینکه بتوان از حداکثر توان پردازنده‌ها استفاده نمود باید برنامه نویسی به گونه‌ای صورت گیرد که امکان اجرای موازی دستورات به طور کامل فراهم گردد در غیر اینصورت نمی‌توان از حداکثر توان پردازشی پردازنده‌ها استفاده نمود.

در حالتی که‌ها چون ارتباطات سریعتر است لذا نیاز به هماهنگ سازی و سنکرون کردن سریعتری نیز داریم. از طرفی برنامه‌ها باید به صورت real time (بی درنگ) اجرا شوند که اینها جزء بحث های کنترلی هستند که باید رعایت شوند.

Impact of Multicores

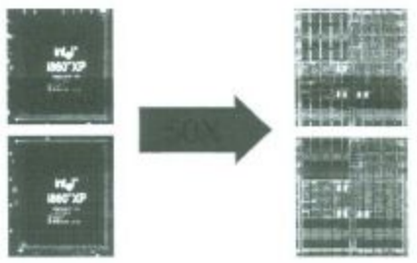
- How much data can be communicated between two cores?
- What changed?
  - Number of Wires
    - I/O is the true bottleneck
    - On-chip wire density is very high
  - Clock rate
    - I/O is slower than on-chip
  - Multiplexing
    - No sharing of pins
- Impact on programming model?
  - Massive data exchange is possible
  - Data movement is not the bottleneck
    - Locality is not that important



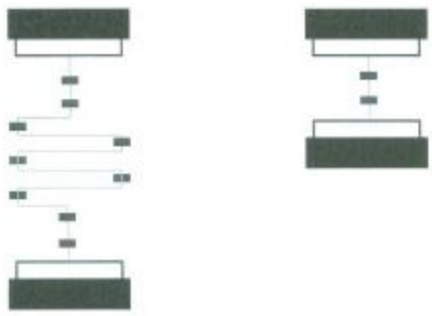
32 Giga bits/sec ~300 Tera bits/sec

Impact of Multicores

- How long does it take for a round trip communication?
- What changed?
  - Length of wire
    - Very short wires are faster
  - Pipeline stages
    - No multiplexing
    - On-chip is much closer
- Impact on programming model?
  - Ultra-fast synchronization
  - Can run real-time apps on multiple cores

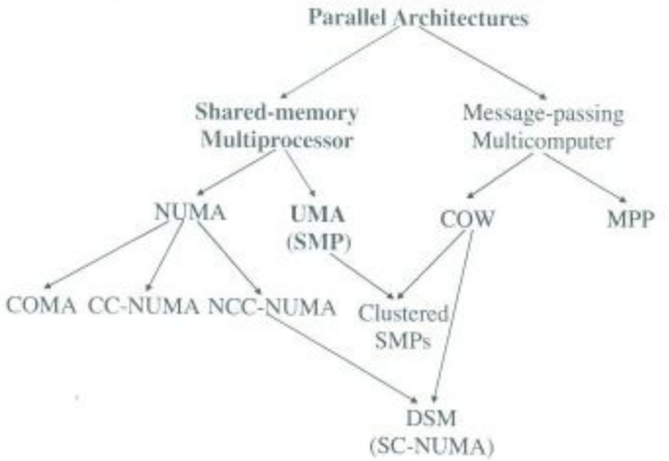


~200 Cycles ~4 cycles

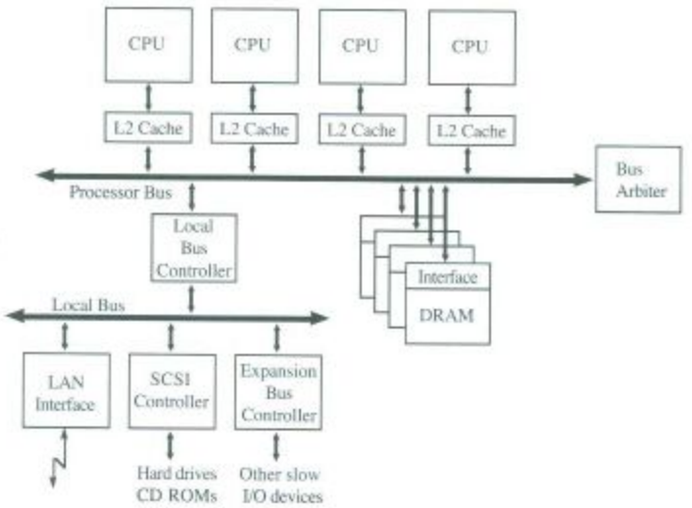


Parallel Computers

- "A parallel computer is a collection of processing elements that *cooperate* and *communicate* to solve large problems fast."
- Taxonomy of parallel computers:



Shared-Memory Multiprocessors



Issues in Multiprocessors

- Programming
  - Need to explicitly define parallelism
- Hardware
  - Interconnection
    - Bus
    - Network
  - Synchronization
    - Test-and-set
    - Barrier synchronization
    - Fetch and add
  - Cache Coherence
    - Snoopy protocols
    - Directory-based

در مالتی پروسسورها هم بصورت چند پردازنده‌ای و هم به صورت مالتی کُر، یک سری نیازمندی‌ها وجود دارد. در زبان‌نویسی حتی الامکان سعی می‌شود که اجرای دستورات به صورت موازی انجام شود. همچنین نیاز مند یک سری تجهیزات نرم‌افزاری اصلی می‌باشیم که مثلاً شامل ارتباطات داخلی می‌باشد. این ارتباطات را می‌توان به صورت BUS یا Network ایجاد نمود که مثلاً در مالتی کُرها به صورت Mesh پیاده‌سازی می‌شود در صورتیکه در مالتی پروسسورها معمولاً به صورت BUS می‌باشد.

یک سری نیازمندی‌های مربوط به سنکرون کردن و هماهنگ سازی نیاز می‌باشد که به چند روش می‌توان آن را انجام داد. همچنین وابستگی به حافظه Cache می‌باشد که باید آن را از بین برد که به دو طریق می‌توان این کار را انجام داد: ۱- استفاده از پروتکل‌های snoopy

۲- ارجاع با آدرس که به صورت Directory-based می‌باشد.

Shared-Memory Programming

- Many vendors have implemented their own proprietary versions of threads.
- A standardized C language threads programming interface, POSIX Threads or *Pthreads*
- Threaded applications offer potential performance gains and practical advantages:
  - Overlapping CPU work with I/O
  - Priority/real-time scheduling
  - Asynchronous event handling
  - Parallelization on SMPs
- Pthreads provide Over 60 routines for
  - Thread management - thread create, join, schedule, etc.
  - Mutexes - mutual exclusion
  - Conditional variables - provides communication between threads

برنامه نویسی حافظه مشترک

استفاده از زبانهای برنامه نویسی خاص مانند زبان C

(P. threads.h)
برنامه نویسی چندنخی

بعضی از این زبان های برنامه نویسی چندنخی کارهایی از قبیل overlap کردن کارهای cpu و I/O را انجام می دهند یا برروی اجرای دستورات real-time اولویت ایجاد می کنند یا موازی سازی برروی SMP ها را آسان تر می سازند .

پنجشنبه - ۹۴/۲/۱۰

5/9/2015 دنیا ما برای پردازنده ها مانند مواد اولیه عمل می نمایند و تا زمانیکه این مواد اولیه را در اختیار پردازنده ها قرار ندهیم، پردازنده هرچقدر هم که توان پردازشی بالایی داشته باشد نمی تواند کاری انجام دهد. حافظه ها مانند یک گلوگاه عمل می نمایند و سرعت رشد حافظه ها به نسبت پردازنده ها خیلی کندتر صورت می گیرد.

Advanced Computer Architecture
5MD00 / 5Z033

حافظه های سلسله مراتبی

Memory Hierarchy & Caches

Henk Corporaal
www.ics.ele.tue.nl/~heco/courses/aca
h.corporaal@tue.nl
TUEindhoven
2007

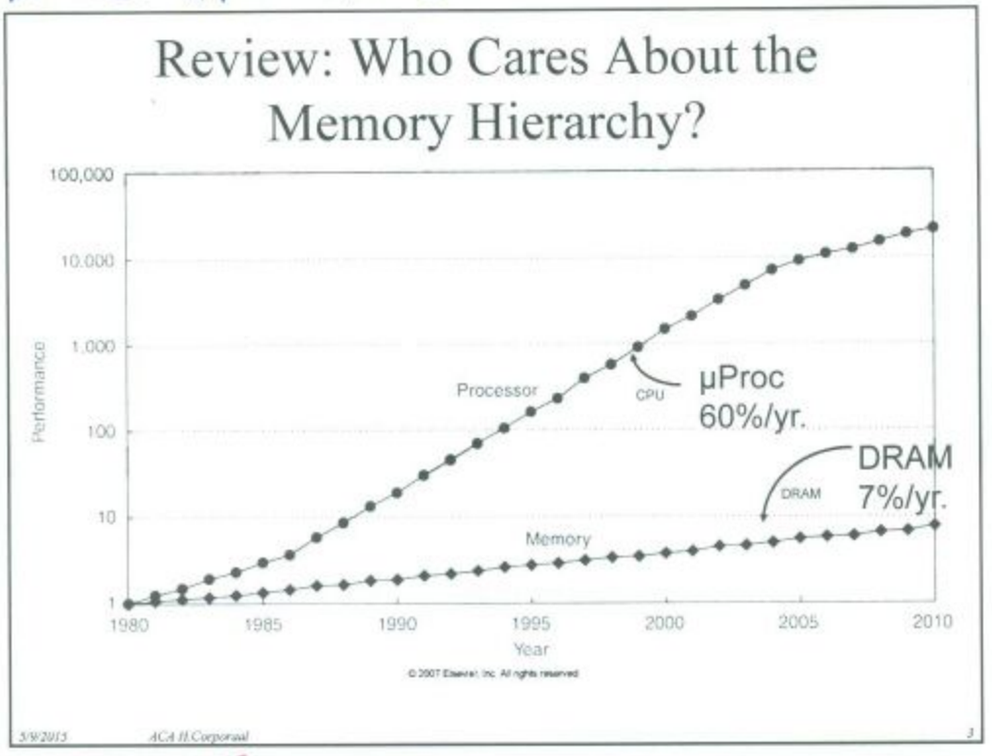
بخش از کندی حافظه ها را با استفاده از حافظه های سلسله مراتبی جبران می نماییم.

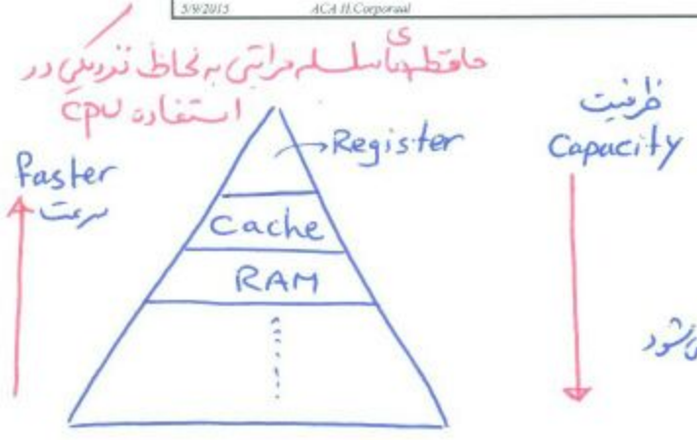
Topics

- Processor – Memory gap
- Recap of cache basics
- Basic cache optimizations
- Advanced cache optimizations
  - reduce miss penalty
  - reduce miss rate
  - reduce hit time

کارآیی حافظه: نسبت ظرفیت حافظه به سرعت انتقال داده را کارآیی حافظه گویند.
هرچه حافظه ای سرعت انتقال بیشتری داشته باشد و ظرفیت بیشتری را برای ما فراهم نماید عملاً دارای کارآیی بالاتری خواهد بود. در عمل این دو پارامتر یعنی سرعت و کارآیی در تضاد با یکدیگر هستند.
هرچه ظرفیت حافظه بزرگتر باشد زمان جستجوی در حافظه طولانی تر خواهد بود. همچنین تکنولوژی ساخت حافظه ها نیز در مدت زمان انتقال داده تاثیر دارد.

همانطور که در شکل زیر مشخص شده است رشد افزایش سرعت پردازنده ها از سال ۱۹۸۰ تا سال ۲۰۱۰ به طور متوسط سالانه ۶۰٪ افزایش داشته است در صورتیکه این رشد برای سرعت حافظه ها فقط حدود ۷٪ بوده است. پس عملاً حافظه ها به صورت یک گلوگاه محل می یابند لذا هرچقدر هم که سرعت پردازنده ها بالا باشد عملاً با وجود حافظه به کارآیی ۱۰۰٪ نخواهیم رسید. 5/9/2015





دسترسی CPU به حافظه از طریق آدرس دهی صورت می گیرد. آدرس دهی توسط پردازنده انجام می شود.

آدرس
- آدرس فیزیکی (واقعی) که در CPU استفاده می شود
- آدرس منطقی (مجازی)

* نسبت به ظرفیت حافظه به تعداد بیت جهت آدرس دهی نیاز داریم.

M: 2^x — x - بیت

مثلاً

M: 1 MByte = 2^{20} — 20 - بیت

Cache: 512KByte = 2^{19} — 19 - بیت

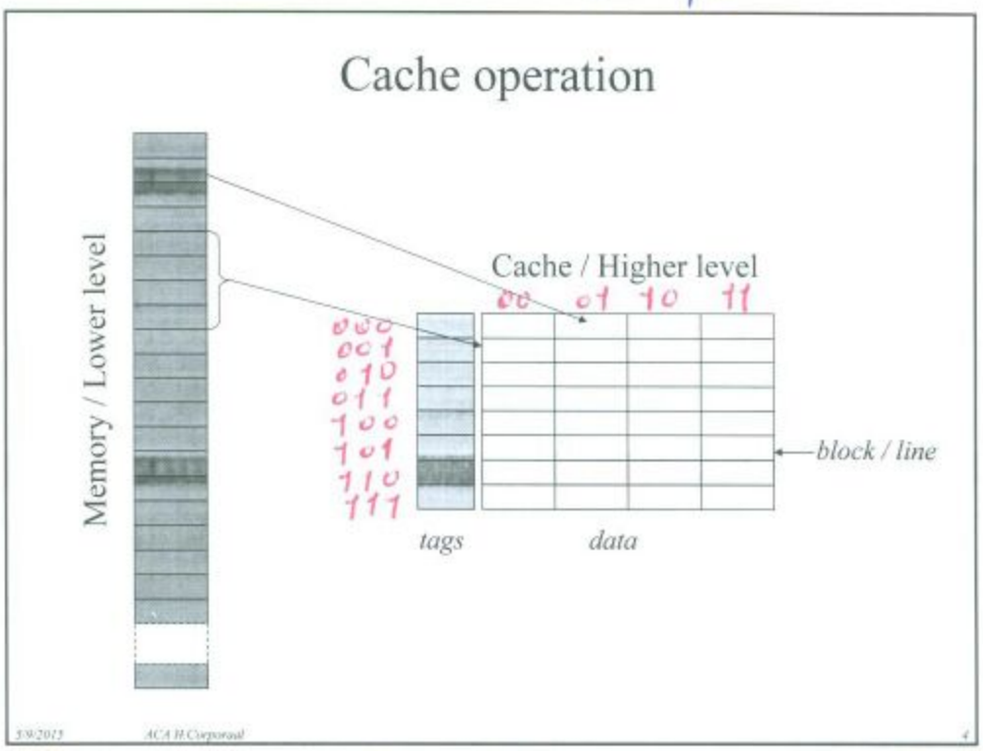
مثلاً در این مثال مشاهده می شود که آدرس های حافظه اصلی ۲۰ بیتی هستند در صورتیکه آدرس های Cache ۱۹ بیتی می باشند

پردازنده آدرس های حافظه های اصلی را تولید می نماید منتهی ما برای دسترسی به آنها باید آنها را به Cache انتقال دهیم پس روشی باید وجود داشته باشد تا آدرس های حافظه اصلی را به آدرس های حافظه Cache تبدیل نمائیم.[1]

نگاشت مستقیم : Direct Mapped

5/9/2015

در نگاشت مستقیم، قسمت کم ارزش آدرس را در خانه‌های حافظه Cache جستجو می‌کنیم و قسمت با ارزش آدرس را در خانه‌های Tag قرار می‌دهیم. پس از جستجوی آدرس اگر اطلاعات مورد نظر در Cache بود که از آن استفاده می‌کنیم و اگر نبود باید آن را از حافظه اصلی وارد Cache نماییم.



اگر حافظه Cache فضای مناسب خالی را برای انتقال داده از حافظه اصلی نداشته باشد باید مقداری از خانه‌های آن را خالی نموده تا فضا برای اطلاعات خواسته شده فراهم گردد.

تاری داده‌ها یا معتبر بودن آنها با استفاده از گیت‌های X-OR انجام می‌شود که در صورت مساوی بیت‌ها صفر و در صورت مخالف بودن آنها یک می‌دهد.

hit : اگر اطلاعات مورد نظر در حافظه Cache باشد اصطلاحاً گفته می‌شود که hit رخ داده است. یا به عبارتی اگر اطلاعات خواسته شده با اطلاعاتی که در Cache وجود دارد یکسان باشد در آن صورت یک سیگنال hit می‌آید. در اینحالت تلاش cpu برای دستیابی به اطلاعات موفقت آمیز بوده است.

$0 <$ hit Rate یا hit Ratio < 1

(hit Rate: نرخ برخوردهای موفق — hit Ratio: نسبت برخوردهای موفق)

miss : زمانیکه اطلاعات مورد نظر در حافظه Cache نباشد، miss رخ داده است.

$$\text{miss Rate یا miss Ratio} = 1 - \text{Hit Rate}$$

استفاده از حافظه‌های انجمنی : Set-Associative Cache

در اینحالت استفاده از حافظه‌ها براساس بلوک بندی است و مانند این است که از چند Cache موازی هم استفاده شده است. در حافظه‌های انجمنی نیز از یک ستون جهت مشخص نمودن شماره بلوک حافظه Cache جهت آدرس دهی استفاده می‌شود.

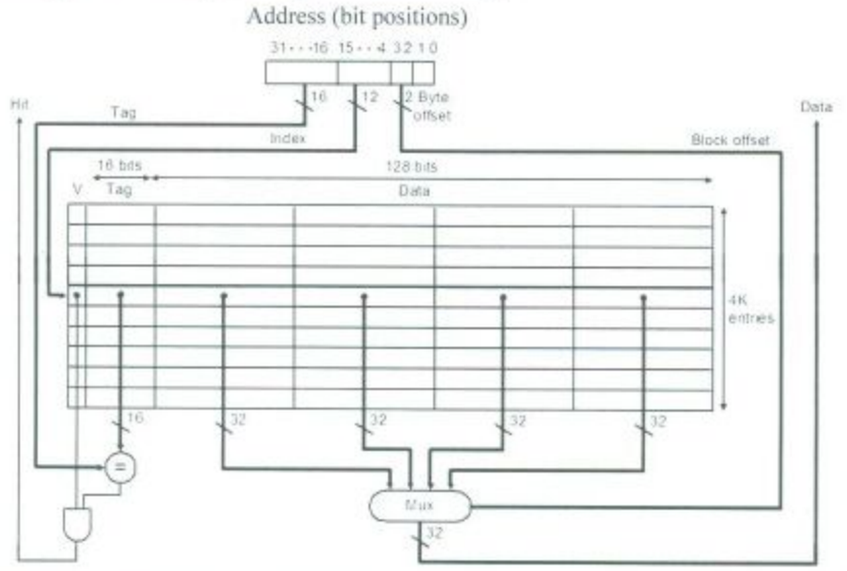
آدرس دهی انجمنی سریعتر از آدرس دهی به روش نگاشت مستقیم است .

5/9/2015

سخت افزار حافظه های انجمنی پیچیده تر است .

Direct Mapped Cache

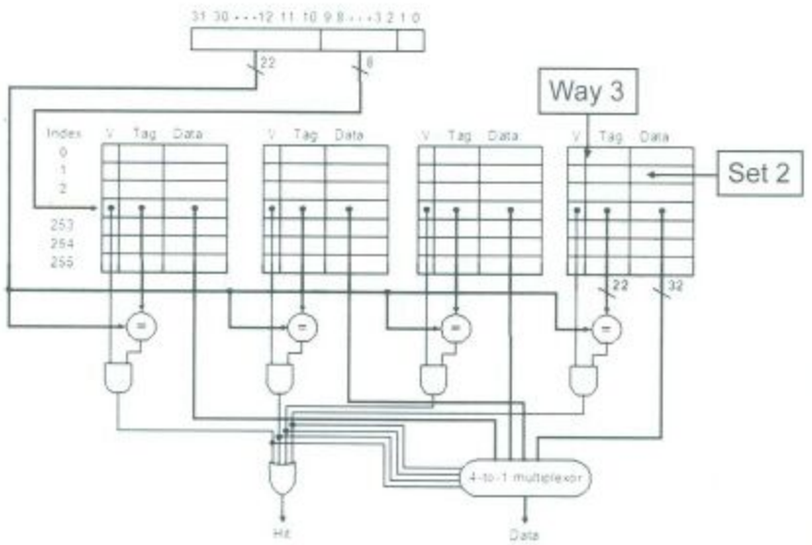
- Taking advantage of spatial locality:



5/9/2015 ACA H.Corporaal 5

کلاً Cache ها یا به صورت نگاشت مستقیم یا نگاشت انجمنی آدرس دهی می شوند .

A 4-Way Set-Associative Cache



5/9/2015 ACA H.Corporaal 6

* سرعت نگاشت مستقیم نسبت به حافظه های انجمنی بالاتر است .

در حافظه های با ظرفیت Cache کم ، نگاشت مستقیم بهتر است در صورتیکه در حافظه های Cache با ظرفیت بالاتر ، روش حافظه های انجمنی مناسب تر بوده و باعث افزایش کارآیی می گردد .

توان مصرفی در حافظه های انجمنی بیشتر است . حافظه های انجمنی اطلاعات را سریعتر جستجو می کنند .

- تمرکز موضعی زمانی (حذف اطلاعات از Cache در زمان حذف، اطلاعاتی را که قدیمی‌تر هستند و زمان طولانی‌تری از آخرین استفاده نشده است، حذف می‌شوند.)
- تمرکز موضعی مکانی (اضافه کردن اطلاعات به Cache)

5/9/2015

6 basic cache optimizations (App. C)

- Reduces miss rate
  - Larger block size
  - Bigger cache
  - Higher associativity
    - reduces conflict rate
- Reduce miss penalty
  - Multi-level caches
  - Give priority to read messes over write misses
- Reduce hit time
  - Avoid address translation during indexing of the cache

5/9/2015 ACA H.Corporaal 7

چگونه می‌توان کارآیی یک حافظه Cache را بهینه کرد؟

۱- کاهش miss rate یا درخواست‌های ناموفق

- بزرگتر کردن اندازه بلوک‌های حافظه که در اینحالت نرخ miss به همان میزان کاهش می‌یابد زیرا در اینحالت احتمال آنکه به خانه‌های مشغول شده نیاز باشد، زیاد است. البته باید توجه شود که اندازه این بلوک‌های حافظه را نمی‌توان خیلی بزرگ انتخاب نمود یعنی مثلاً نمی‌توان اندازه بلوک را برابر حافظه Cache یا نصف آن در نظر گرفت و این کار را معمولاً به روش‌های آماری انجام می‌دهند.
- بزرگتر کردن حافظه Cache زیرا مسلم است هرچه اندازه Cache بزرگتر باشد اطلاعات بیشتری را می‌توان در آن قرار داد و به همان نسبت مقدار miss rate نیز کاهش خواهد یافت.
- استفاده از حافظه‌های انجمنی بزرگتر

۲- کاهش جریمه یا هزینه مربوط به درخواست‌های ناموفق

- استفاده از حافظه‌های Cache چندسطحی با لول‌های مختلف
- ایجاد اولویت برخواندن اطلاعات در زمان نوشتن اطلاعات

۳- کاهش زمان درخواست‌های موفق یا hit time

یعنی جستجو در Cache سریعتر انجام شود که برای این منظور از روش‌هایی استفاده می‌شود که آدرس را در سریعتری را ایجاد نماید.

علاوه بر روش های گفته شده در صفحه قبل، با استفاده از روش های زیر می توان کارآیی حافظه Cache را بهبود بخشید. 5/9/2015

11 Advanced Cache Optimizations (&5.2)

- **Reducing hit time**

1. Small and simple caches
2. Way prediction
3. Trace caches

- **Increasing cache bandwidth**

4. Pipelined caches
5. Multibanked caches
6. Nonblocking caches

- **Reducing Miss Penalty**

7. Critical word first
8. Merging write buffers

- **Reducing Miss Rate**

9. Compiler optimizations

- **Reducing miss penalty or miss rate via parallelism**

10. Hardware prefetching
11. Compiler prefetching

5/9/2015 ACA H.Corporaal 8

کاهش زمان Hit time

۱- استفاده از Cache های کوچک و ساده البته باید توجه نمود که استفاده کردن از Cache های کوچکتر باعث آدرس دهی سریعتر شده ولی miss ها را نیز افزایش خواهد داد. در هر صورت همزمان نمی توان کلیه شرایط را ایده آل نمود.

۲- راههای پیش بینی کردن اطلاعات یعنی توان پیش بینی نمود که بلاکی را که به Cache منتقل می نمائیم در دستورات بعدی هم مورد استفاده قرار گیرد.

۳- Trace کردن Cache که باعث بالا رفتن سرعت جستجو می شود البته توان مصرفی را هم افزایش می دهد.

افزایش پهنای باند Cache

۴- ورود و خروج اطلاعات را به صورت pipeline انجام دهیم.

۵- استفاده از Cache های چند بانکی یعنی چندین فیلد Tag داشته باشند.

۶- از Cache های بلاک نشده استفاده شود زیرا در استفاده از Cache به صورت بلاک درست است که ممکن است اطلاعات منتقل شده مورد استفاده قرار گیرد ولی این احتمال نیز وجود دارد که اطلاعات اضافی منتقل شده مورد استفاده قرار نگیرد.

کاهش جریمه های مربوط به درخواست های ناموفق

۷- ابتدا داده های بحرانی یا Critical را به Cache منتقل نمائیم

۸- استفاده از یافرس Cache و حافظه اصلی زیرا یافر به صورت یک حافظه میانی عمل کرده و نیاز به جستجو هم ندارد.

کاهش miss Rate

۹- با بهینه سازی کامپایلر می توان دستورات را به صورت موازی اجرا نموده و miss rate را کاهش داد

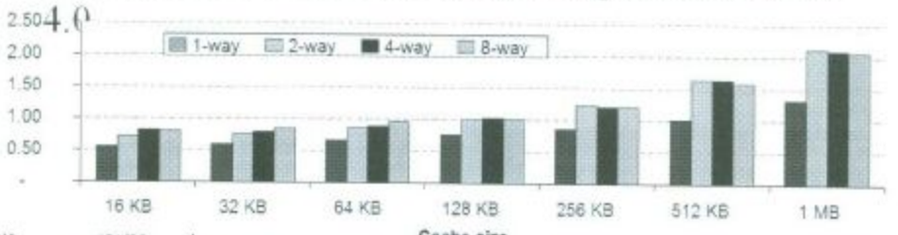
کاهش جریمه های مربوط به درخواست های ناموفق با استفاده از موازی سازی

۱۰- پیش واکش نخت افزاری مثلاً با استفاده از پردازنده های آرایه ای می توان برای دو عدد، چهار عمل جمع، تفریق، ضرب و تقسیم را انجام داد و سپس یکی از آنها را که مورد نیاز است استفاده نمود.

5/2/2015

1. Fast Hit via Small and Simple Caches

- Index tag memory and then compare takes time
- ⇒ Small cache is faster
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ direct mapping
  - Can overlap tag check with data transmission since no choice
- Access time estimate for 90 nm using CACTI model



5/2/2015 ACA H Corporaal 9

۱۱- با استفاده از کامپایلر می توان پیش واکش نرم افزاری داشته باشیم.

انجام نگاشت مستقیم، انجام جستجو و مقایسه اطلاعات را سریعتر می نماید.

2. Fast Hit via Way Prediction

- Make set-associative caches faster
- Keep extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed
  - Miss ⇒ 1st check other blocks for matches in next clock cycle
- Accuracy ≈ 85%
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Used for instruction caches vs. L1 data caches
  - Also used on MIPS R10K for off-chip L2 unified cache, way-prediction table on-chip

5/2/2015 ACA H Corporaal 10

هرچه اندازه Cache کوچکتر باشد تعداد tag هایی که باید محتوای آنها با قسمت پر ارزش آدرس مقایسه شود، کمتر خواهد بود و لذا سریعتر این کار انجام می شود و نیز سریعتر مشخص می شود که اطلاعات مورد نظر در Cache قرار دارند یا خیر

استفاده از حافظه های انحنی و راههای پیش بینی می تواند باعث سریعتر شدن زمان Hit گردد.

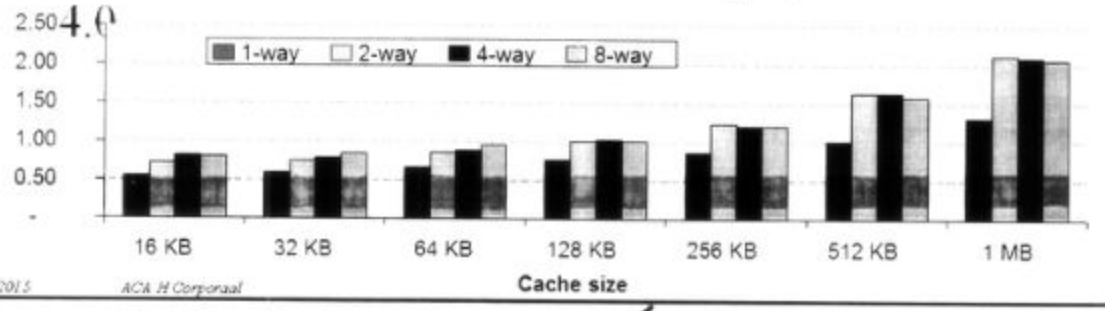
۹- با بهینه سازی کامپایلر می توان دستورات را به صورت موازی اجرا نموده و miss rate را کاهش داد

کاهش جریمه های مربوط به درخواست های ناموفق با استفاده از موازی سازی

۱۰- پیش واکشی سخت افزاری مثلاً با استفاده از پردازنده های آرایه ای می توان برای دو عدد، چهار عمل جمع، تفریق، ضرب و تقسیم را انجام داد و سپس کدی از آنها را که مورد نیاز است استفاده نمود.

1. Fast Hit via Small and Simple Caches

- Index tag memory and then compare takes time
- ⇒ Small cache is faster
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ direct mapping
  - Can overlap tag check with data transmission since no choice
- Access time estimate for 90 nm using CACTI model



۱۱- با استفاده از کامپایلر می توان پیش واکشی نرم افزاری داشته باشیم.

* انجام نگاشت مستقیم، انجام جستجو و مقایسه اطلاعات را سریعتر می نماید.

به طور کلی برای آنکه بتوان کلا آرایی یک Cache را بهبود بخشید می توان کارهای زیر را انجام داد:

۱- کاهش زمان Hit که زمان لازم برای جستجوی داده ای که در حافظه Cache قرار دارد می باشد که اگر بتوانیم این زمان را کاهش دهیم در واقع کارآیی Cache خود را می توانیم بالا ببریم زیرا یکی از راههای افزایش کارآیی، سرعت دسترسی به اطلاعات است.

۲- افزایش پهنای باند حافظه Cache یعنی بتوان داده های مفیدی را در حافظه Cache جای داد

۳- کاهش جریمه مربوط به درخواست های ناموفق که باعث پرداخت هزینه می شود یا مجبوریم زمانی را تحمل نمائیم تا داده از حافظه اصلی به حافظه Cache منتقل پیدا کرده و سپس در اختیار پردازنده قرار گیرد. لذا اگر بتوانیم این زمان را کاهش دهیم در واقع عملکرد Cache خود را افزایش داده ایم.

۴- کاهش نرخ miss یعنی داده هایی را که به احتمال زیاد پردازنده برای اجرای دستورات به آنها نیاز دارد را از قبل در حافظه Cache آماده کنیم.

۵- جریمه مربوط به درخواست های ناموفق یا نرخ miss را با استفاده از موازی سازی کاهش دهیم.

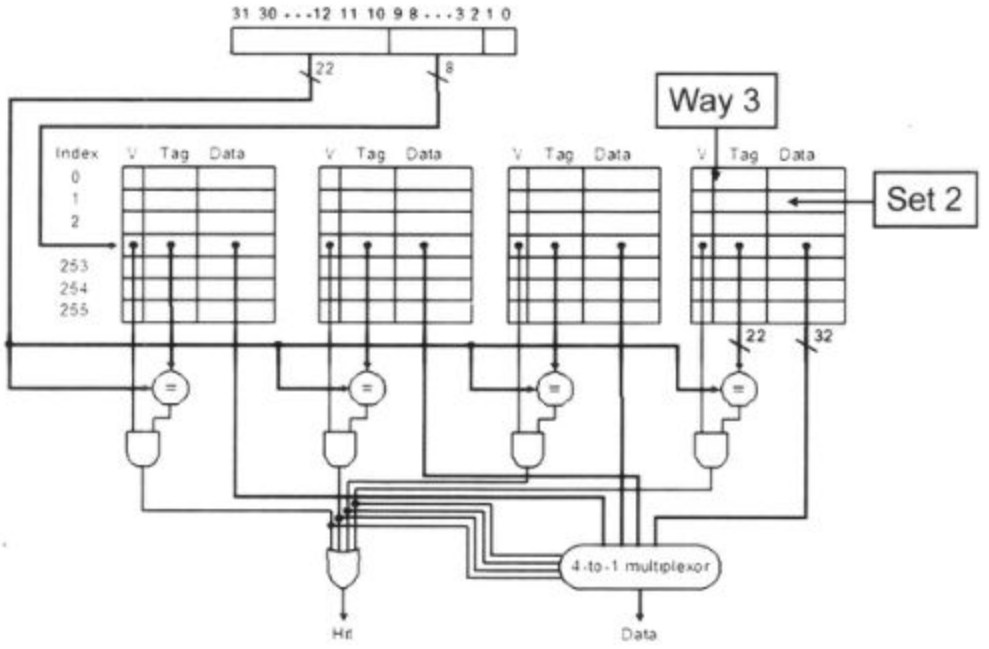
2. Fast Hit via Way Prediction

- Make set-associative caches faster
- Keep extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed
  - Miss ⇒ 1st check other blocks for matches in next clock cycle
- Accuracy ≈ 85%
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Used for instruction caches vs. L1 data caches
  - Also used on MIPS R10K for off-chip L2 unified cache, way-prediction table on-chip

هرچه اندازه Cache کوچکتر باشد تعداد Tagهایی که باید محتوای آنها با قسمت پرارزش آدرس مقایسه شود، کمتر خواهد بود ولذا سریعتر این کار انجام می شود و نیز سریعتر مشخص می شود که اطلاعات مورد نظر در Cache قرار دارند یا خیر

استفاده از حافظه های انحنی و راههای پیش بینی می تواند باعث سریعتر شدن زمان Hit گردد

A 4-Way Set-Associative Cache



5/20/2015 ACA H.Corporaal 11

همانگونه که گفته شد اگر بخواهیم Hit های خود را افزایش دهیم می توانیم از روش های پیش بینی

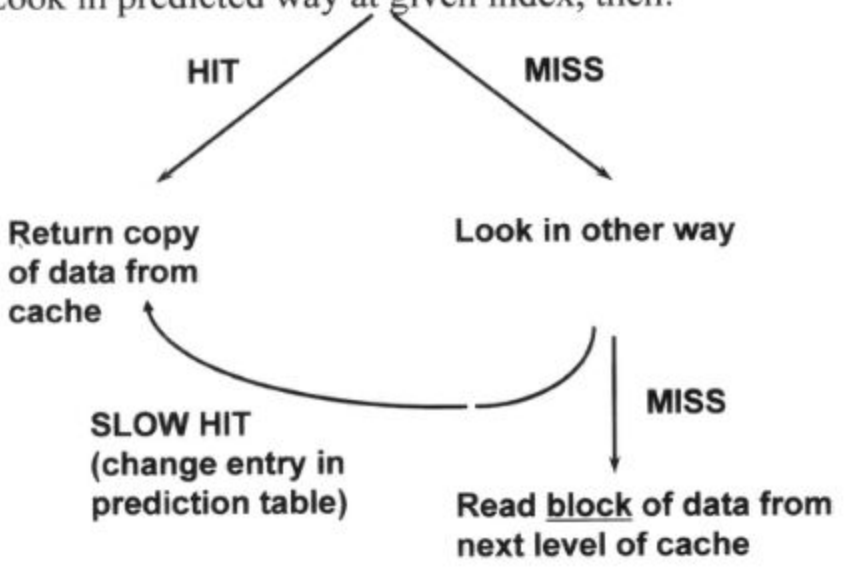
نیز روش های آدرس دهی استفاده کنیم که برای این منظور می توان از Cache های انجمنی استفاده

نمود که هر یک از آنها به صورت یک Cache نگاشت مستقیم عمل می کند که جستجوی داده در آنها

به صورت موازی انجام شده و در نتیجه سرعت Hit یا جستجوی داده در حافظه افزایش می یابد

Way Predicting Caches

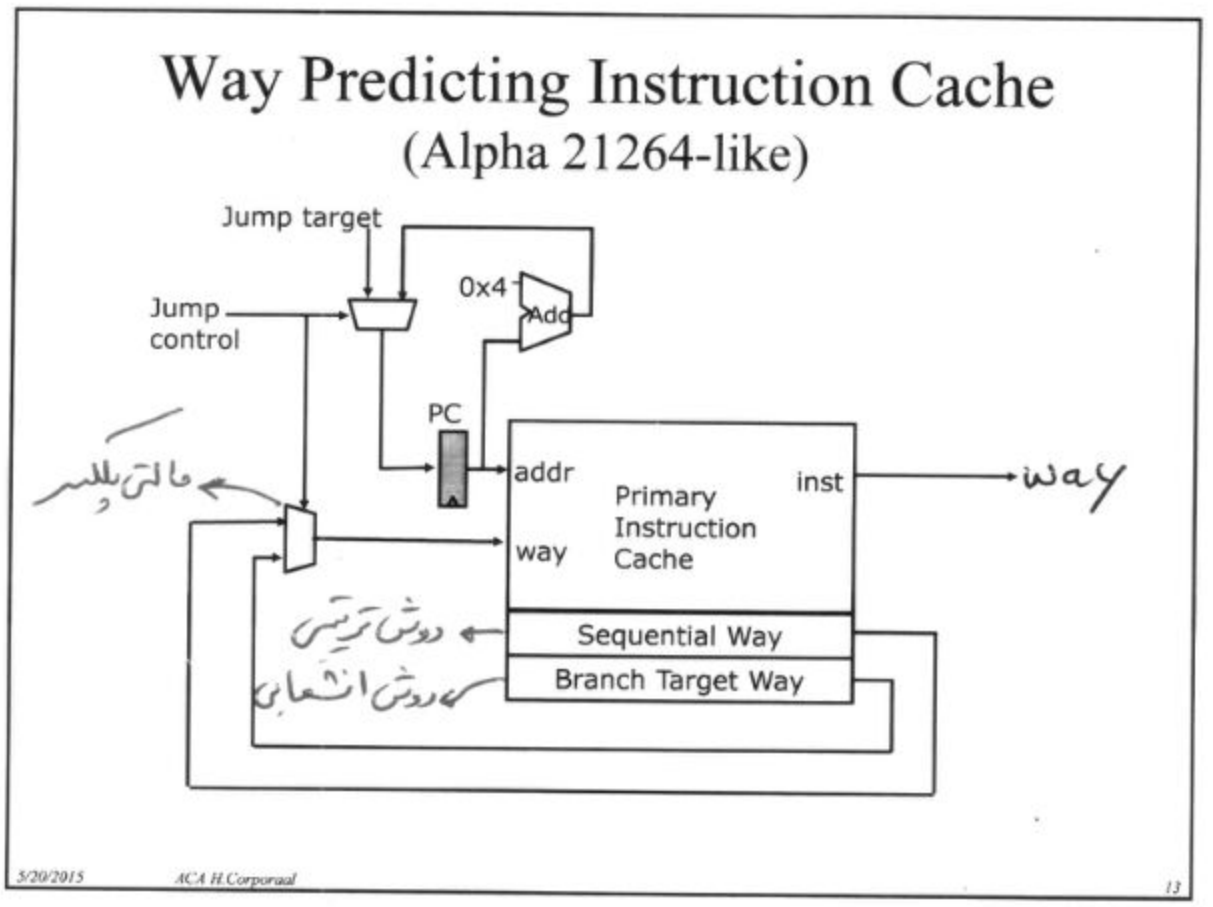
- Use processor address to index into way prediction table
- Look in predicted way at given index, then:



وقتی به دنبال داده‌ای در Cache می‌گردیم دو حالت اتفاق می‌افتد :

۱ ـ داده مورد نظر در Cache قرار دارد و به عبارتی Hit رخ می‌دهد که در اینصورت یک کپی از داده‌ها تهیه شده و به پردازنده ارسال می‌گردد .

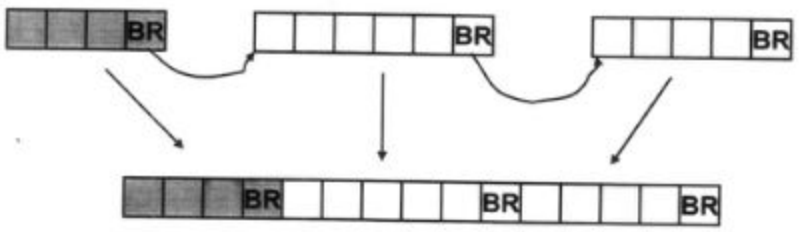
۲ ـ Miss رخ می‌دهد که در اینصورت باید برای یافتن داده مورد نظر ، در سطح‌های بعدی حافظه جستجو نماییم مثلاً اگر اطلاعات مورد نظر در Cache نبود باید به دنبال آن در RAM بگردیم و اگر آنجا نبود باید در حافظه جانبی جستجو نماییم و به همین ترتیب در لول‌های بعدی حافظه جستجو کنیم .



جستجو در حافظه به دو روش ترتیبی و انشعابی انجام می‌شود و در هر سطح حافظه باید به یکی از این دو روش عمل جستجو را انجام دهیم. در بلوک دیاگرام فوق روشی برای پیش بینی حافظه‌های Cache مربوط به دستورالعمل‌ها نشان داده شده است. توجه شود که حافظه‌های مربوط به داده و دستورالعمل‌ها با هم متفاوت می‌باشند و برای هریک از آنها حافظه‌های سلسله مراتبی جداگانه داریم.

3. Fast (Inst. Cache) Hit via Trace Cache

Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line



- **Single fetch brings in multiple basic blocks**
- **Trace cache indexed by start address *and* next *n* branch predictions**

روشی برای افزایش سرعت Hit یا کاهش زمان Hit ، استفاده از Trace کردن است .
Trace کردن یعنی جستجو کردن موردی یعنی جستجو را به صورت پرش از روی بعضی خانه های
حافظه انجام دهیم . در روش معمولی برای یافتن داده در حافظه ، جستجو در حافظه به صورت
خانه به خانه صورت می گیرد اما در روش Trace کردن با آدرسی که به هر قسمت حافظه مربوط به یک
دستورالعمل داده می شود می توان از روی بعضی از خانه های حافظه پرش نموده ولذا دستورالعمل
مورد نظر را سریعتر پیدا نمود زیرا در اکثر اوقات اطلاعات مربوط به یک دستورالعمل بدلیل حجم زیاد به
طور پشت سر هم قرار ندارند و هر قسمت آن ممکن است در یک قسمت از حافظه ذخیره شود . در اینحالت
در انتهای هر قسمت ، آدرس شروع قسمت بعد را قرار می دهیم . اگر بخواهیم از Cache بدین صورت
استفاده نمائیم می توانیم از اشاره گر استفاده کنیم بدین صورت که اگر اطلاعاتی را که نیاز داریم در یک
قسمت نبود ، در آنصورت با یک پروسه ما را به قسمت بعدی راهنمایی کند . در اینحالت لازم نیست
تمام بلاک های حافظه برای یافتن داده مورد نظر جستجو شود یعنی اگر آدرس قسمت بعدی در بلاک
بعدی نباشد از روی آن پرش نموده و جستجو را در بلاک مورد اشاره انجام می دهیم . لذا استفاده از
Trace کردن باعث افزایش سرعت جستجو می گردد .

3. Fast Hit times via Trace Cache

- Trace cache in Pentium 4
  1. Dynamic instr. traces cached (in level 1 cache)
  2. Cache the micro-ops vs. x86 instructions
     - Decode/translate from x86 to micro-ops on trace cache miss

\+ ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)

\- ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size

\- ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

استفاده از Trace کردن اولین بار در Pentium 4 مورد استفاده قرار گرفت .

نکات مثبت در Trace کردن :

هرچه بلوک های داده بزرگتر باشد بهره وری بیشتر خواهد بود . توجه شود که در روش Trace کردن وقتی در یک بلوک مشغول جستجو هستیم امکان اینکه از وسط بلوک عمل پرش را انجام دار امکان ندارد و لازم است جستجوی بلوک تا انتها انجام شود یعنی یک بلوک را کامل جستجو نموده و سپس براساس اطلاعات و آدرس که از این بلوک بدست می آید ، به بلوک دیگر پرش می نماید .

نکات منفی در Trace کردن :

— هرچه نگاشت آدرس روی پیچیده تر باشد ، Hit ها طولانی تر خواهد بود زیرا هرچه تعداد بخش های آدرس یا تعداد مولفه های آدرس کمتر باشد ، Hit سریعتری خواهیم داشت زیرا در آدرس دهی پیچیده نیاز به تعداد مراحل بیشتری داریم که این امر باعث تأخیر بیشتر شده و زمان جستجو طولانی تر خواهد بود .

— اگر Cache مورد استفاده برای ذخیره دستورالعمل باشد ، یک دستورالعمل مشخص می تواند در زمانهای متفاوت و در بخش های مختلف برنامه مورد استفاده قرار گیرد . در روش Trace کردن چون مجبور به استفاده از روش ترتیبی هستیم و چون معیار Trace کردن مخصوصاً در دستورهای انشعابی همیشه دارای معیار ثابت و مشخصی نمی باشد لذا برای دستورالعمل ها روش Trace کردن روش مناسبی برای کاهش Hit time نمی باشد در صورتیکه برای Cache هایی که مربوط به دیتا هستند مناسب می باشد .

4: Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Nr. of Instruction cache access pipeline stages:
- 1: Pentium
- 2: Pentium Pro through Pentium III
- 4: Pentium 4
- ⇒ greater penalty on mispredicted branches
- ⇒ more clock cycles between the issue of the load and the use of the data

یکی دیگر از روش های افزایش بهره وری Cache های پیشرفته، افزایش پهنای باند با استفاده از pipeline می باشد. همانطور که گفته شد می توان حافظه های چند سطحی را که به صورت پشت سر هم قرار می دهیم مانند ماشینی که به صورت pipeline عمل می کند، در نظر بگیریم.

استفاده از روش pipeline کردن در Cache هایی که برای حافظه های دیتا می باشد، روش مناسبی نیست و برعکس این روش برای Cache های مربوط به دستورالعمل ها مناسب می باشد.

برای هر انسان می‌توان یک کارآیی تعریف نمود. مثلاً برای پردازنده، کارآیی عبارت است از تعداد انجام دستورالعمل در واحد زمان

$$\text{cpu time} = \left(\begin{array}{c}\text{مدت زمان اجرای دستور} \\ \text{توسط پردازنده}\end{array} + \begin{array}{c}\text{مدت زمان واکشی دستور} \\ \text{از حافظه}\end{array} \right)$$

یا

$$\text{cpu time} = \left(\begin{array}{c}\text{تعداد کلاک های مورد} \\ \text{نیاز برای اجرای دستور}\end{array} + \begin{array}{c}\text{تعداد کلاک های مورد} \\ \text{نیاز برای واکشی دستور} \\ \text{از حافظه}\end{array} \right) * \begin{array}{c}\text{مدت زمان یک کلاک} \\ \text{clock cycle}\end{array}$$

خواندن از حافظه Read

نوشتن در حافظه Write

لازم به ذکر است در یک کامپیوتر تعداد کلاک های مورد نیاز برای خواندن و نوشتن با هم برابر نمی‌باشند و تعداد آنها کاملاً متفاوت می‌باشد. لذا در تعداد کلاک های مورد نیاز برای دسترسی به حافظه، باید هم کلاک های مربوط به خواندن و هم کلاک های مربوط به نوشتن را در نظر بگیریم که در واقع مجموع آنها در نظر گرفته می‌شود.

$$\begin{array}{c}\text{تعداد کلاک های مورد نیاز} \\ \text{برای دسترسی به حافظه}\end{array} = (\text{Read clocks} + \text{write clocks})$$

کارآیی یک حافظه باید در اجرای یک برنامه مورد بررسی قرار گیرد نه یک دستور لذا اگر دو حافظه تعداد کلاک خواندن و نوشتن یکسان داشته باشند، هر یک از آنها که دارای حجم حافظه بیشتری است دارای کارآیی بالاتری خواهد بود زیرا تعداد Miss هایی که رخ می‌دهد کمتر است زیرا تعداد دستورات بیشتری از برنامه که مورد نیاز است در حافظه قرار دارد.

برنامه $P_1 \Rightarrow n$ دستور m ارجاع به حافظه

تعداد دستورات ارجاع به حافظه خواندن از حافظه

$$\text{Read clock} = \frac{m_1}{n} \times (\text{Read hit Rate clock}) \times (\text{Read Miss Rate Penalty})$$

تعداد کلاک ها در زمان وقوع hit

تعداد کلاک های مورد نیاز زمانی که Miss رخ می‌دهد

تعداد دستورات ارجاع به حافظه نوشتن در حافظه

$$\text{write clock} = \frac{m_2}{n} \times (\text{write hit Rate clock}) \times (\text{write Miss Rate Penalty}) + \text{write Buffer clock}$$

باید توجه نمود که بین خواندن و نوشتن در حافظه تفاوتی وجود دارد و آن این است که وقتی پردازنده ای نیاز به خواندن از حافظه دارد ، آدرس آن تولید شده و جستجو می گردد و با توجه به کلاک های hit و یا در صورت ایجاد miss ، یک مقدار کلاکی ایجاد می شود و در نهایت داده مورد نظر پیدا شده و توسط پردازنده استفاده می گردد که در اینحالت پردازنده منتظر خواندن می ماند تا دراد ، اجرای دستورات خودش را کامل کند یعنی در زمان خواندن تا پردازنده چیزی دریافت نکند کاری انجام نمی دهد پس منتظر می ماند تا عمل خواندن انجام شود در صورتیکه در زمان نوشتن، پردازنده منتظر نمی ماند بلکه اطلاعاتی که قرار است در حافظه نوشته شود را در بافر قرار می دهد لذا مدت زمانی برای نوشتن داده ها در بافر نیاز می باشد و چون این عمل یک بار نیاز است لذا تاثیر آن در رابطه به صورت جمع قرار داده می شود .

AMT : Access Memory Time میانگین مدت زمان دسترسی به حافظه

در حافظه ای که دارای یک مدت زمان hit و یک نرخ miss می باشد ، میانگین مدت زمان دسترسی به حافظه (AMT) به صورت زیر بدست می آید . در اینحالت همیشه از کار آیی در میان نمی باشد .

$$AMT = \underset{\text{مدت زمان دسترسی موفق به حافظه}}{\text{hit time}} + \left((\text{miss rate}) \times \underset{\text{با وقوع miss}}{\text{زمان دسترسی به حافظه}} \right)$$

توجه شود برای هر miss حتماً یک hit رخ می دهد یعنی ابتدا باید حافظه جستجو شود تا مشخص گردد که داده مورد نظر در حافظه وجود ندارد و سپس به سطح مربوط به جستجوی آن در حافظه های سطح های پائین تر بررسی گردد .

حافظه های Cache حافظه های محدودی هستند و همین محدودیت باعث ایجاد دو رخداد hit و miss می شود . زمانیکه miss رخ می دهد یعنی داده های مورد نظر ما در حافظه قرار ندارند و باید به یک روش این اطلاعات از حافظه های سطح های بعدی به حافظه Cache منتقل شود . از طرفی در این انتقال ممکن است ظرفیت حافظه Cache تکمیل باشد پس مجبوریم ابتدا داده ای را از حافظه Cache حذف نموده و آن را به حافظه اصلی انتقال دهیم و به جای آن داده مورد نظر را از حافظه اصلی به حافظه Cache منتقل کنیم .

آنکه کدام یک از داده‌ها از حافظه Cache حذف شود تا فضای خالی برای انتقال داده مورد نظر از حافظه اصلی به حافظه Cache فراهم شود. در واقع تکنیکی است که برای جای‌گذاری یا Replacement استفاده می‌کنیم که این امر بسیار بر روی کارآیی Cache تاثیر دارد زیرا اگر داده‌ای را که بعداً مورد نیاز است از حافظه Cache به دلیل نبودن جای کافی خارج نمائیم عملاً باعث می‌شویم که در زمانهای بعد miss رخ دهد.

اگر توانیم اطلاعاتی را که جایگزینیم در جایی قرار دهیم که در زمانهای بعد تداخلی با داده‌های دیگر نداشته باشد می‌توان مطمئن بود که میزان miss Rate را کاهش داده‌ایم

جای‌گذاری

Cache Replacement Policies

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

ECE/CS 752 Spring 2012

علاوه بر Replacement چندپارامتر دیگر نیز وجود دارد که بر روی کارآیی Cache تاثیر دارند.

Cache Design: Four Key Issues

- These are: یعنی اینکه داده‌های جدیدی را که وارد حافظه می‌کنیم در کجا قرار دهیم
  - Placement – جای دهی
    - Where can a block of memory go?
  - Identification – شناسایی یا جستجو در حافظه
    - How do I find a block of memory?
  - **Replacement**
    - **How do I make space for new blocks?**
  - Write Policy – سیاست در نوشتن در حافظه
    - How do I propagate changes?
- Consider these for caches
  - Usually SRAM
- Also apply to main memory, disks

2

پارامترهای تاثیرگذار بر کارآیی حافظه Cache عبارتند از :

- placement یا جای دهی یعنی اینکه داده‌های جدیدی را که وارد حافظه Cache می‌کنیم در کجای حافظه قرار دهیم.
- Identification شناسایی یا جستجو در حافظه یعنی جستجوی در حافظه به چه صورت باشد. آیا به صورت ترتیبی یا ایندکس بندی شده باشد و یا به صورت موازی مانند حافظه‌های انجمنی صورت گیرد که این امر بسیار زیاد بر روی سرعت و بهره‌وری Cache تاثیر دارد.
- Replacement یا جای‌گذاری یعنی اینکه وقتی حافظه Cache پُر است کدام یک از خانه‌های حافظه را خالی کنیم که بعدها به آن نیاز نداشته باشیم.
- Write Policy یعنی سیاست در نوشتن

حافظه Cache یک حافظه کاملاً موقت است و زمانی که جریان الکتریکی قطع شود، اطلاعات حافظه Cache هم از بین می‌رود پس بنابراین زمانی که یک کپی از اطلاعات را درون Cache قرار می‌دهیم که برای اجرای دستورات، این اطلاعات باید تغییر پیدا نماید، حال چه سیاست و روشی را به کار ببریم که این تغییرات بر روی داده اصلی نیز اعمال شود که این امر مربوط به Write Policy می‌باشد.

در ثبات‌ها، کامپایلر و مدیریت برنامه تصمیم می‌گیرد که جای دهی در کجا انجام شود.
در حافظه Cache، جای دهی به صورت یک روش ثابت و از قبل تعیین شده می باشد مثلاً
به صورت نگاشت مستقیم یا استفاده از حافظه‌های انجمنی و یا تمام انجمنی انجام می‌گیرد.
در حافظه اصلی، عمل جای دهی توسط سیستم عامل مشخص می‌گردد.

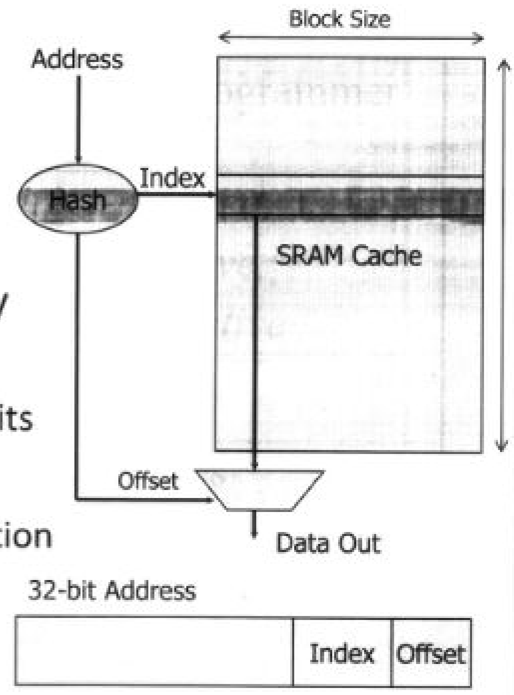
Placement

Memory Type	Placement	Comments
Registers	Anywhere; Int, FP, SPR	Compiler/programmer manages
Cache (SRAM)	Fixed in H/W	*Direct-mapped, set-associative, fully-associative*
DRAM	Anywhere	O/S manages
Disk	Anywhere	O/S manages

3

Placement

- Address Range
  - Exceeds cache capacity
- Map address to finite capacity
  - Called a *hash*
  - Usually just masks high-order bits
- *Direct-mapped*
  - Block can only exist in one location
  - Hash collisions cause problems



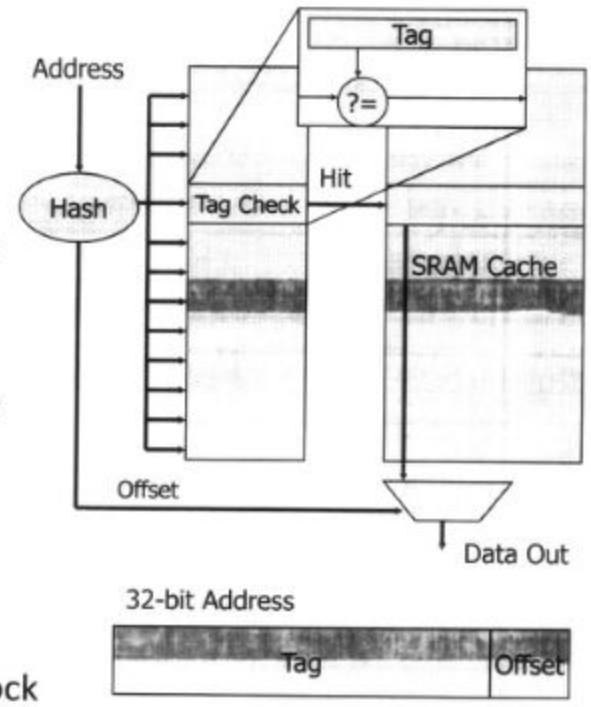
32-bit Address

	Index	Offset

4

Identification

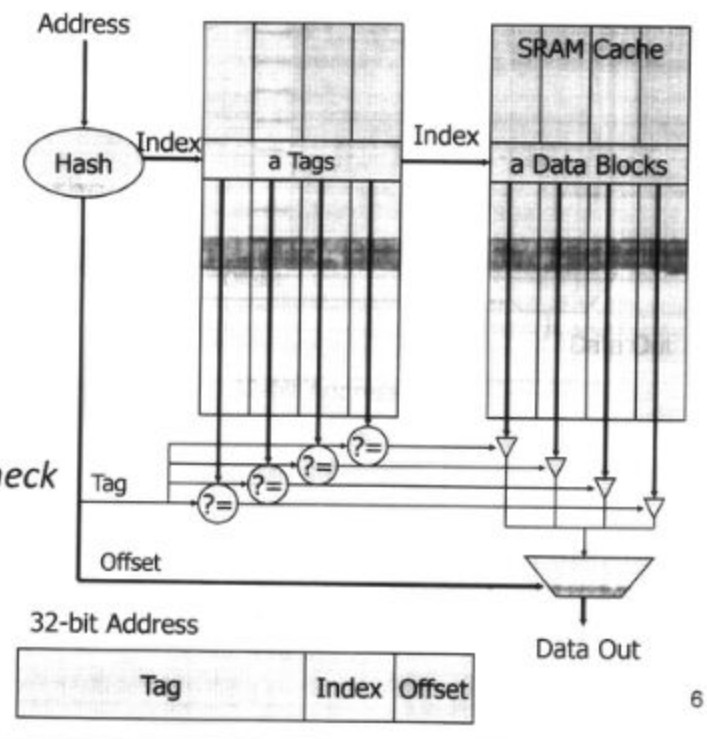
- *Fully-associative*
  - Block can exist anywhere
  - No more hash collisions
- *Identification*
  - How do I know I have the right block?
  - Called a *tag check*
    - Must store address tags
    - Compare against address
- Expensive!
  - Tag & comparator per block



5

Placement

- *Set-associative*
  - Block can be in *a* locations
  - Hash collisions:
    - *a* still OK
- *Identification*
  - Still perform *tag check*
  - However, only *a* in parallel



6

سایز حافظه Cache محدود است. وقتی یک حافظه Cache پر می‌شود و نیاز است داده‌ای را در آن قرار دهیم، چه استراتژی را برای آن به کار ببریم؟ کدام داده را باید خالی یا از Cache خارج نموده و داده جدید را به جای آن قرار دهیم. استراتژی Replacement بسیار زیاد بر روی hit و miss اثرگذار است.

Replacement

- Cache has finite size
  - What do we do when it is full?
- Analogy: desktop full?
  - Move books to bookshelf to make room
  - Bookshelf full? Move least-used to library
  - Etc.
- Same idea:
  - Move blocks to next level of cache

در زمانیکه Cache خالی است و پردازنده شروع به کار می‌کند، در ابتدا برای هر داده‌ای miss رخ می‌دهد یعنی برای اولین ارجاع حتماً miss رخ می‌دهد. این امر به هیچ عنوان قابل چشم پوشی و حذف نمی‌باشد.

Cache Miss Rates: 3 C's [Hill]

- Compulsory miss or Cold miss
  - First-ever reference to a given block of memory
  - Measure: number of misses in an infinite cache model
- Capacity
  - Working set exceeds cache capacity
  - Useful blocks (with future references) displaced
  - Good replacement policy is crucial!
  - Measure: additional misses in a fully-associative cache
- Conflict
  - Placement restrictions (not fully-associative) cause useful blocks to be displaced
  - Think of as *capacity within set*
  - Good replacement policy is crucial!
  - Measure: additional misses in cache of interest

استفاده از ظرفیت های بزرگ حافظه Cache اگرچه در ابتدا باعث افزایش miss rate می‌گردد اما در دراز مدت اگر از روش های مناسب جای دهی استفاده شود باعث کاهش نرخ miss می‌گردد.

اگرچه از حافظه های انجمنی استفاده شود می‌تواند miss Rate را کاهش دهد.

Replacement

- How do we choose *victim*?
  - Verbs: *Victimize, evict, replace, cast out*
- Many policies are possible
  - FIFO (first-in-first-out)
  - LRU (least recently used), pseudo-LRU
  - LFU (least frequently used)
  - NMRU (not most recently used)
  - NRU
  - Pseudo-random (yes, really!)
  - Optimal
  - Etc

Optimal Replacement Policy?

[Belady, IBM Systems Journal, 1966]

- Evict block with longest reuse distance
  - i.e. next reference to block is farthest in future
  - Requires knowledge of the future!
- Can't build it, but can model it with trace
  - Process trace in reverse
  - [Sugumar&Abraham] describe how to do this in one pass over the trace with some lookahead (Cheetah simulator)
- Useful, since it reveals ***opportunity***
  - (X,A,B,C,D,X): LRU 4-way SA $, 2nd X will miss

می‌خواهیم ببینیم جای‌گذاری به چه روشی انجام شود زیرا الگوریتم‌های مختلفی برای جای‌گذاری وجود دارد یعنی در زمانی که حافظه Cache پر است، کدام اطلاعات را حذف نمائیم که اطلاعات جدید جایگزین آن شود.

- FIFO ساده‌ترین روش آن است که هرکدام که اول آمده است زودتر نیز خارج شود. این روش برای استفاده از دستورالعمل‌ها بسیار مناسب می‌باشد زیرا دستورات به صورت ترتیبی و خط به خط و پشت سر هم اجرا می‌شوند در صورتیکه برای دیتا این چنین نیست و متغیرها به ترتیبی که ترتیب می‌شوند، استفاده نمی‌گردند و ممکن است یک متغیر در چندین جای برنامه استفاده شود و یا فقط در یک خط از برنامه استفاده گردد.

- LRU در این روش داده‌ای که در مدت زمان دورتری استفاده شده و یا مدت زمان زیادی گذشته و از آن استفاده نشده است جهت جای‌گذاری انتخاب می‌شود. این روش یک استراتژی است که بیان می‌کند داده‌ای که خیلی وقت پیش استفاده شده و مدت زیادی است که استفاده نشده است به احتمال زیاد در آینده نزدیک هم استفاده نمی‌شود.

- LFU روش دیگری که وجود دارد آن است که داده‌ای که اخیراً استفاده شده ممکن است تا مدت زیادی از آن استفاده نشود

- NMRU این روش به صورت الگوریتمی عمل می‌کند و چون استفاده از الگوریتم باعث صرف زمان از سوی پردازنده است لذا کارایی پردازنده را کاهش می‌دهد.

- NRU تقریباً شبیه NMRU است.

- Pseudo-Random در این روش یکی از داده‌ها را به صورت رندوم انتخاب می‌نمائیم که به صورت 50-50 می‌باشد. البته توابع رندوم باید عادلانه بوده و دارای توزیع یکسانی باشند.

- optimal در این روش می‌توان اولویت دهی نمود که کدامیک از داده‌ها خارج تا داده جدید جایگزین بهینه آن شود.

* نکته‌ای که باید توجه نمود آن است که هم این روش‌ها همیشه و در همه جا مناسب نیستند.