2010-06-17

2010年 前学期 TOKYO TECH

計算機アーキテクチャ 第一 (E)

9. メモリ4:
キャッシュシステム, プロセッサシミュレータ

吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室 木曜日13:20 ー 14:50

レポート 問題

1. int add (int a, int b) { return a + b; }
   をクロスコンパイラにてMIPS命令セットにコンパイルし, コンパイルオプションによってどのように変化するかをまとめよ.
2. swap (int v[], int k)
   をクロスコンパイラにてMIPS命令セットにコンパイルし, コンパイルオプションによってどのように変化するかをまとめよ.
3. void max (int v[], int n)
   をクロスコンパイラにてMIPS命令セットにコンパイルし, コンパイルオプションによってどのように変化するかをまとめよ.
4. 同様に, サンプルアプリケーションを作成し, それをクロスコンパイラにてMIPS命令セットにコンパイルし, コンパイルオプションによってどのように変化するかをまとめよ.
5. この課題の感想をまとめること.
6. レポートはA4用紙2枚以内にまとめること. (必ずPDFとすること)
   (2段組, コードは小さい文字でもかまわない.)

レポート 提出方法

- 6月18日(午後7時)までに電子メールで提出
  - 人よりも先に提出している(先願性)と高得点
  - report10a_at_arch.cs.titech.ac.jp
- 電子メールのタイトル
  - Arch Report [学籍番号]
  - 例 : Arch Report [33_77777]
- 電子メールの内容
  - 氏名, 学籍番号
  - 回答
    - PDFファイルを添付 (必ずPDFとすること)
    - PDFファイルにも氏名, 学籍番号を記入すること.
    - A4用紙で2枚以内にまとめること.

レポート 問題

1. SimMipsにデータキャッシュのヒット率を測定する仕組みを追加し, ヒット率を測定せよ.
   1. ダイレクトマップ方式, ラインサイズは4ワードとする.
   2. セット数を8, 16, 32, 64, 128, 256, 512に変更した場合のヒット率を示せ.
   3. 以前作成した max (1000要素のランダムデータ) を含む3つのアプリケーションを作成し, そのヒット率を示すこと.
2. キャッシュのヒット率を改善する方式を実装し, その効果を示せ.
   1. 例えば, ラインサイズの変更
   2. 例えば, セットアソシアティブ方式
   3. 例えば, フルアソシアティブ方式
3. SimMipsに関する感想など
   1. この課題に要した時間, SimMipsへの感想, SimMipsに期待する改良.

講義用の計算機環境

- 講義用の計算機
  - 131.112.16.56
  - ssh arche@131.112.16.56
    - **ユーザ名: arche**
    - **パスワードは講義時に連絡**
  - cd myname (例: cd 06B77777)
  - cp –r /home/arche/v0.6.4 .
  - cd v0.6.4
  - memory.cc などを修正してコンパイル, 実行
- 注意点
  - 計算機演習室からは外部にsshで接続できないかもしれません.
  - Windowsからは Tera Term Pro などを利用してください.

レポート 提出方法

- 7月2日(午後7時)までに電子メールで提出
  - 人よりも先に提出している(先願性)と高得点
  - report10a_at_arch.cs.titech.ac.jp
- 電子メールのタイトル
  - Arch Report [学籍番号]
  - 例 : Arch Report [33_77777]
- 電子メールの内容
  - 氏名, 学籍番号
  - 回答
    - PDFファイルを添付 (必ずPDFとすること)
    - PDFファイルにも氏名, 学籍番号を記入すること.
    - A4用紙で4枚以内にまとめること.

SimMips …

- 藤枝直輝, 渡邉伸平, 吉瀬謙二: 教育・研究に有用なMIPSシステムシミュレータSimMips, 情報処理学会論文誌, Vol.50, No.11, pp. 2665-2676 (2009).

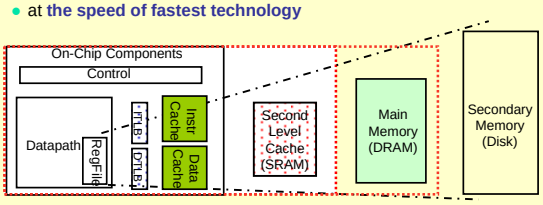
Acknowledgement

- Lecture slides for Computer Organization and Design, Third Edition, courtesy of **Professor Mary Jane Irwin**, Penn State University
- Lecture slides for Computer Organization and Design, third edition, Chapters 1-9, courtesy of **Professor Tod Amon,** Southern Utah University.

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

A Typical Memory Hierarchy

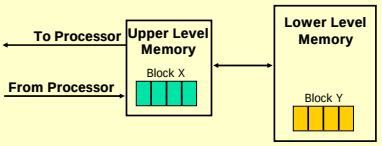
- By taking advantage of **the principle of locality** (局所性)
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

The Memory Hierarchy: Why Does it Work?

- **Temporal Locality** (時間的局所性, Locality in Time):
  - ⇒ Keep **most recently accessed** data items closer to the processor
- **Spatial Locality** (空間的局所性, Locality in Space):
  - ⇒ Move blocks consisting of **contiguous words** to the upper levels



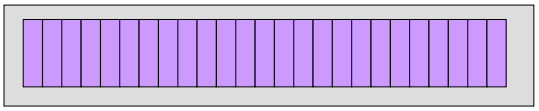
10

Cache

- Two questions to answer (in hardware):
  - Q1: **How do we know if a data item is in the cache?**
  - Q2: **If it is, how do we find it?**
- **Direct mapped**
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must **share** locations in the upper level
  - Address mapping: **(block address) modulo (# of blocks in the cache)**
  - First, consider block sizes of **one word**

11

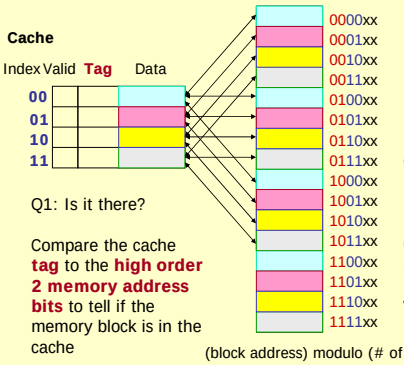
Cache



本棚

机

Caching: A Simple First Example



Q1: Is it there?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

Two low order bits define the byte in the word (32-b words)

Q2: How do we find it?

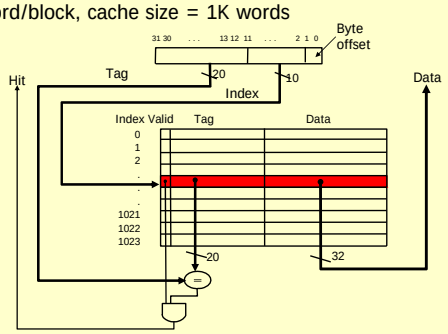
Use **next 2 low order memory address bits** – the **index** – to determine which cache block

(block address) modulo (# of blocks in the cache)

13

MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words

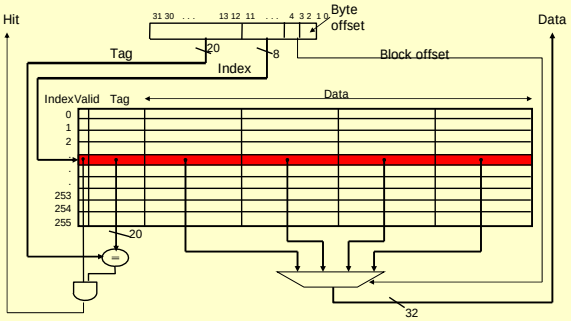


What kind of locality are we taking advantage of?

14

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

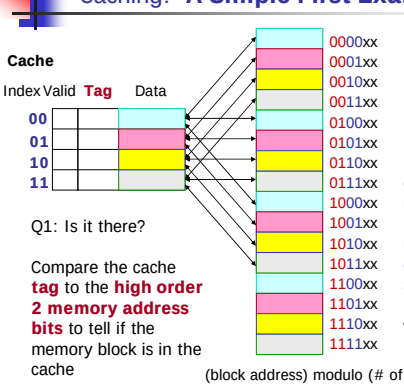
15

Reducing Cache Miss Rates, associativity

- **Allow more flexible block placement**
  - In a **direct mapped cache** a memory block maps to exactly **one cache block**
  - At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**
  - A compromise is to divide the cache into **sets** each of which consists of **n "ways"** (**n-way set associative**).
    A memory block maps to a unique set and can be placed in any way of that set (so there are **n choices**)

16

Caching: A Simple First Example



Q1: Is it there?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

Two low order bits define the byte in the word (32-b words)

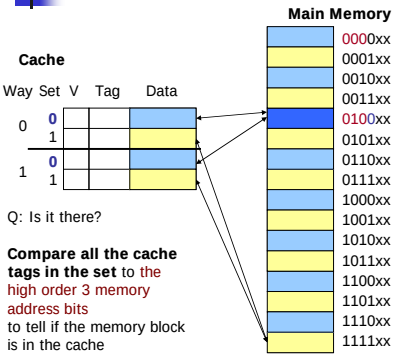
Q2: How do we find it?

Use **next 2 low order memory address bits** – the **index** – to determine which cache block

(block address) modulo (# of blocks in the cache)

17

Set Associative Cache Example



Q: Is it there?

Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Two low order bits define the byte in the word (32-b words)
One word blocks

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set

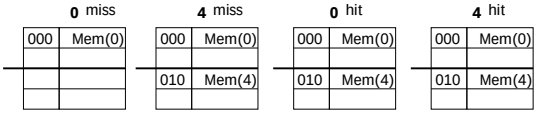
18

Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache – all blocks initially marked as not valid

0 4 0 4 0 4 0 4

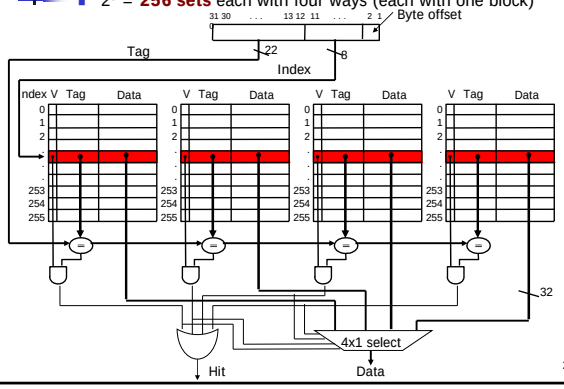


- 8 requests, 2 misses

- Solves the **ping pong effect** in a direct mapped cache due to conflict misses

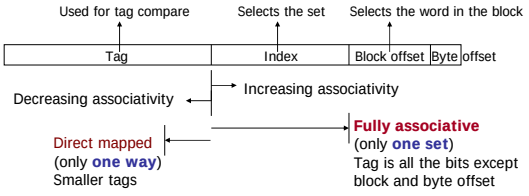
Four-Way Set Associative Cache

- 2^8 = **256 sets** each with four ways (each with one block)



Range of Set Associative Caches

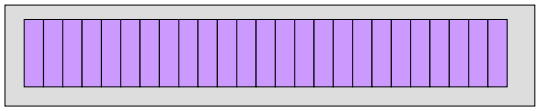
- For a fixed size cache



Costs of Set Associative Caches

- **N-way set associative** cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available **after** set selection and Hit/Miss decision.
- **When a miss occurs**, which way's block do we pick **for replacement** ?
  - **Least Recently Used (LRU):** the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used
    - For **2-way set associative**, takes **one bit per set** → set the bit when a block is referenced (and reset the other way's bit)
  - **Random**

Cache

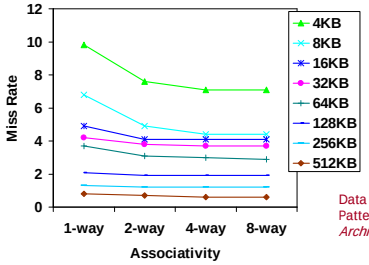


本棚

机

Benefits of Set Associative Caches

- The choice of direct mapped or set associative depends on **the cost of a miss** versus **the cost of implementation**



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Largest gains are in going from **direct mapped** to **2-way**

Reducing Cache Miss Rates by **multiple levels**

L1 cache | L2 cache | L3 cache

- Enough room on the die for **bigger L1 caches** *or* for a **second level of caches** – normally a **unified** L2 cache (i.e., it holds both instructions and data) and in some cases even a **unified L3 cache**
- For our example,
  - CPI_{ideal} of 2,
  - 100 cycle miss penalty (to main memory),
  - 36% load/stores,
  - a 2% (4%) L1I$ (D$) miss rate,
  - **add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate**

$$CPI_{stalls} = 2 + .02 \times 25 + .36 \times .04 \times 25 + \mathbf{.005 \times 100} + \mathbf{.36 \times .005 \times 100} = 3.54$$

(as compared to **5.44** with no L2$)

25

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
  - Secondary cache should focus on **reducing miss rate** to reduce the penalty of long main memory access times
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty

26

Key Cache Design Parameters

	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates	2% to 5%	0.1% to 2%

27

Two Machines' Cache Parameters

	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

28

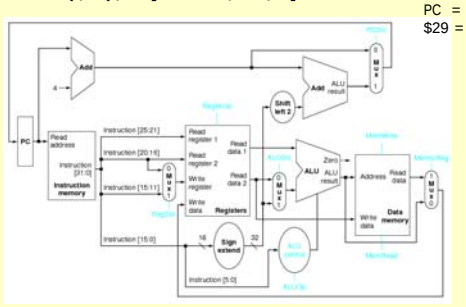
プロセッサの**データパス**(シングル・サイクル)

op	rs	rt	16 bit immediate

I format

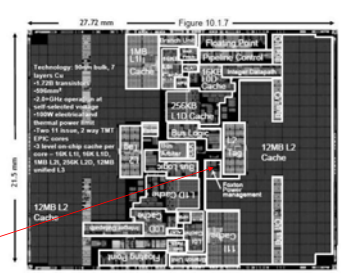
addi $sp, $sp, 4 [addi $29, $29, 4]

PC = 36
$29 = 8000



先端マイクロプロセッサ **Intel Montecito**

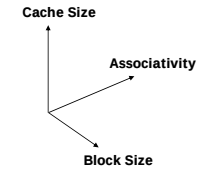
- 2個のEPICプロセッサコア
- 1MB L2, 12MB L3キャッシュ
- EPICコアは11 issue, 2way Temporal MT
- 初の10億超トランジスタ
  - 1.72BTrs
  - 21.5mm x 27.7mm
  - 90nm
  - 100W
- パワー制御用の専用チップFoxtonを搭載

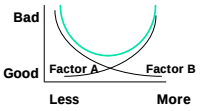


Source: ISSCC 2005 papers

Summary: The Cache Design Space

- **Several interacting dimensions**
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- **The optimal choice is a compromise**
  - depends on access characteristics
    - workload
    - I-cache, D-cache
  - depends on technology / cost
- **Simplicity** often wins





31

OPT: Optimal Replacement Policy

The Optimal Replacement Policy

- Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
- Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

- Lookahead Window : Window of accesses between miss causing access and the access to the least imminent replacement candidate. Single pass simulation of OPT make use of lookahead windows to identify replacement candidates and modify current cache state [Sugumar-SIGMETRICS1993]

OPT: あまり切迫していないものを置き換える.

Optimal Replacement Policy の例

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5	0		1		2	3	4					
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1, A_2, A_3, A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : $(A_1, A_2, A_3, A_4, A_5)$
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2
- A_6 self-replaces, lookahead window and imminence order in table