2010-06-10

2010年 前学期 TOKYO TECH

計算機アーキテクチャ 第一 (E)

8. メモリ3：半導体メモリシステム

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 － 14：50

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**
  - mkdir myname (例: mkidr 06B77777)
  - cd myname (例: cd 06B77777)
- 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.

Sample program

コンパイラの最適化オプションを変更しながら，どのような命令列が出力されるか試してみる．

```
#include <stdio.h>
int main(){
  int i;
  int sum = 0;

  for(i=1; i<=100; i++)  sum += i;

  return sum;
}
```

mipsel-linux-gcc -O0 -S main.c -o main_opt0.s

/home/share/cad/mipsel/usr/bin/mipsel-linux-gcc

Sample program

```
#include <stdio.h>
int main(){
  int i;
  int sum = 0;

  for(i=1; i<=100; i++)
    sum += i;

  return sum;
}
```

mipsel-linux-gcc -O0 -S main.c -o main_opt0.s

```
main:
        .frame  $fp,24,$31
        .mask   0x40000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro

        addiu   $sp,$sp,-24
        sw      $fp,16($sp)
        move    $fp,$sp
        sw      $0,8($fp)
        li      $2,1
        sw      $2,12($fp)
        b       $L2
        nop

$L3:
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,8($fp)
        lw      $2,12($fp)
        nop
        addiu   $2,$2,1
        sw      $2,12($fp)
$L2:
        lw      $2,12($fp)
        nop
        slt     $2,$2,101
        bne     $2,$0,$L3
        nop

        lw      $2,8($fp)
        move    $sp,$fp
        lw      $fp,16($sp)
        addiu   $sp,$sp,24
        j       $31
        nop
```

Sample program

```
#include <stdio.h>

int main(){
  int i;
  int sum = 0;

  for(i=1; i<=100; i++)
    sum += i;

  return sum;
}
```

mipsel-linux-objdump -d ./a.out

```
004005c0 <main>:
  4005c0:   27bdffe8    addiu   sp,sp,-24
  4005c4:   afbe0010    sw      s8,16(sp)
  4005c8:   03a0f021    move    s8,sp
  4005cc:   afc00008    sw      zero,8(s8)
  4005d0:   24020001    li      v0,1
  4005d4:   afc2000c    sw      v0,12(s8)
  4005d8:   1000000a    b       400604 <main+0x44>
  4005dc:   00000000    nop
  4005e0:   8fc30008    lw      v1,8(s8)
  4005e4:   8fc2000c    lw      v0,12(s8)
  4005e8:   00000000    nop
  4005ec:   00621021    addu    v0,v1,v0
  4005f0:   afc20008    sw      v0,8(s8)
  4005f4:   8fc2000c    lw      v0,12(s8)
  4005f8:   00000000    nop
  4005fc:   24420001    addiu   v0,v0,1
  400600:   afc2000c    sw      v0,12(s8)
  400604:   8fc2000c    lw      v0,12(s8)
  400608:   00000000    nop
  40060c:   28420065    slti    v0,v0,101
  400610:   1440fff3    bnez    v0,4005e0 <main+0x20>
  400614:   00000000    nop
  400618:   8fc20008    lw      v0,8(s8)
  40061c:   03c0e821    move    sp,s8
  400620:   8fbe0010    lw      s8,16(sp)
  400624:   27bd0018    addiu   sp,sp,24
  400628:   03e00008    jr      ra
  40062c:   00000000    nop
```

Sample program

```
main:
        .frame  $sp,0,$31
        .mask   0x00000000,0
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro

        j       $31
        li      $2,5050
```

遅延分岐に注意

```
# Makefile
all:
    mipsel-linux-gcc -O0 -S main.c -o main_opt0.s
    mipsel-linux-gcc -O1 -S main.c -o main_opt1.s
    mipsel-linux-gcc -O2 -S main.c -o main_opt2.s
    mipsel-linux-gcc -O3 -S main.c -o main_opt3.s
```

Exercise 2

```
void swap (int v[], int k) {
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}

void max (int v[], int n){
  int i;
  for (i = 1; i < n; i +=1){
    if (v[i-1] > v[i]) swap(v, i-1);
  }
}
```


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

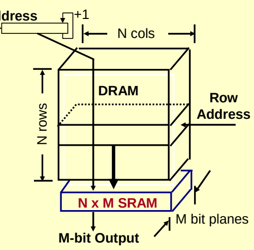
- 6月18日（午後7時）までに電子メールで提出
  - 人よりも先に提出している（先願性）と高得点
  - report10a_at_arch.cs.titech.ac.jp
- 電子メールのタイトル
  - Arch Report [学籍番号]
  - 例：Arch Report [33_77777]
- 電子メールの内容
  - 氏名，学籍番号
  - 回答
    - PDFファイルを添付（必ずPDFとすること）
    - PDFファイルにも氏名，学籍番号を記入すること．
    - A4用紙で2枚以内にまとめること．

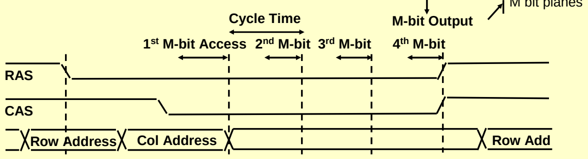
Acknowledgement

- 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.

Synchronous DRAM (**SDRAM**) Operation

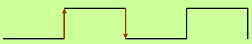
- After a **row** is read into the SRAM register
  - Inputs CAS as **the starting "burst" address** along with a burst length
  - Transfers a burst of data from a series of sequential addresses within that row



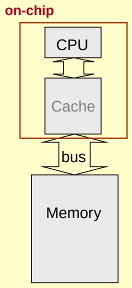


Other DRAM Architectures

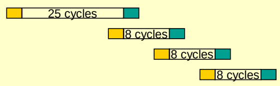
- Double Data Rate SDRAMs – **DDR-SDRAMs** (and DDR-SRAMs)
  - Double data rate because they transfer data on both the rising and falling edge of the clock
  - Are the most widely used form of SDRAMs
- **DDR2-SDRAMs**
  - **4 data prefetch**
- **DDR3-SDRAMs**
  - **8 data prefetch**



One Word Wide Memory Organization, con't

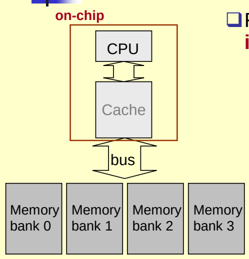


- What if the block size is **four words** and if a **page mode DRAM** is used?
  - **1** cycle to send 1st address
  - **25 + (3 * 8) = 49** cycles to read DRAM
  - **1** cycle to return last data word
  - **51 total clock cycles** miss penalty

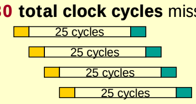


- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 51 = 0.314** bytes per clock

Interleaved（インターリーブ）Memory Organization



- For a block size of **four words with interleaved memory (4 banks)**
  - **1** cycle to send 1st address
  - **25 + 3 = 28** cycles to read DRAM
  - **1** cycle to return last data word
  - **30 total clock cycles** miss penalty



- Number of bytes transferred per clock cycle (**bandwidth**) for a single miss
  - **(4 x 4) / 30 = 0.533** bytes per clock

2010-06-10

2010年 前学期 TOKYO TECH

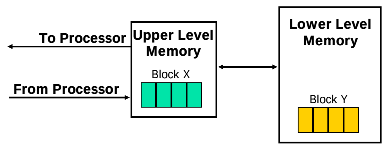
計算機アーキテクチャ 第一 (E)

キャッシュシステム

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 – 14:50

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



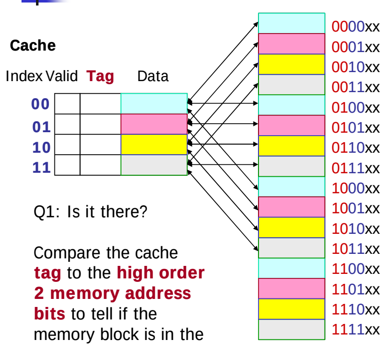
16

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**

17

Caching: A Simple First Example



Main Memory

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

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

Q2: How do we find it?

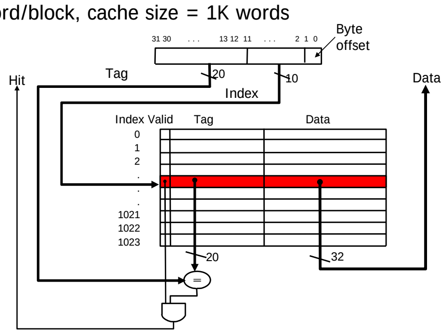
Use **next 2 low order memory address bits** – the **index** – to determine which cache block

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

18

MIPS Direct Mapped Cache Example

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



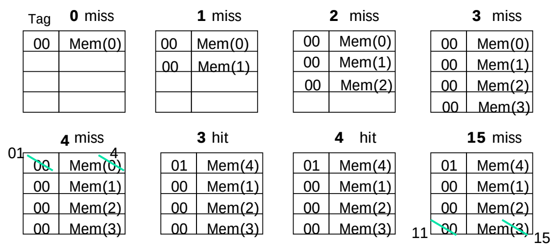
What kind of locality are we taking advantage of?

Direct Mapped Cache

- Consider the main memory word reference string

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

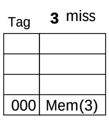
0 1 2 3 4 3 4 15



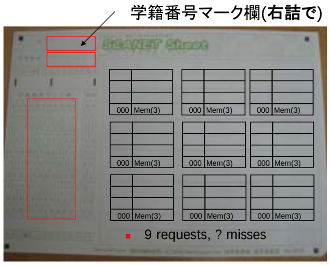
- 8 requests, 6 misses

Exercise

- Consider the main memory word reference string
  - 3, 2, 18, 3, 16, 2, 3, 18, 3



氏名，学籍番号，
学籍番号マーク欄(右詰で)

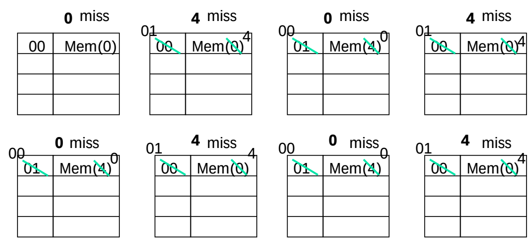


- 9 requests, ? misses

Another Reference String Mapping

- Consider the main memory word reference string

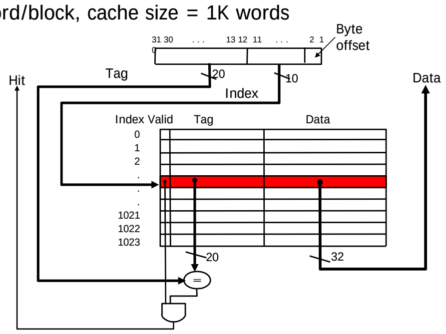
0 4 0 4 0 4 0 4



- 8 requests, 8 misses
- Ping pong effect due to **conflict** misses - two memory locations that map into the same cache block

MIPS Direct Mapped Cache Example

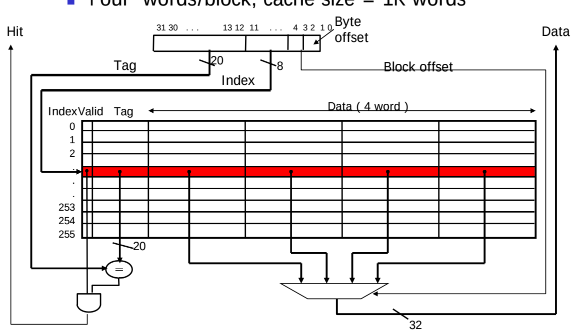
- One word/block, cache size = 1K words



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

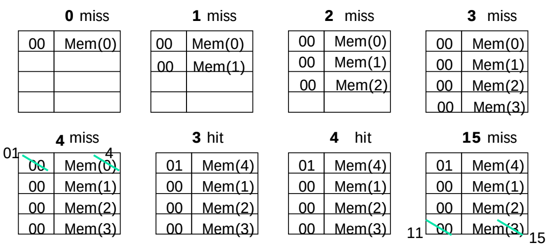
- Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

Direct Mapped Cache again!

- Consider the main memory word reference string
  0 1 2 3 4 3 4 15

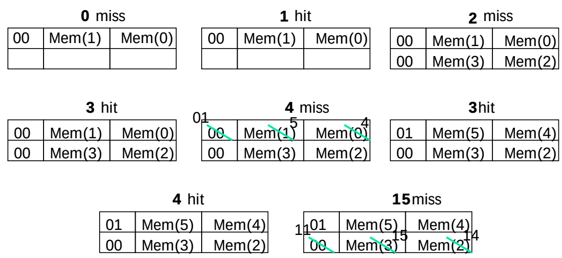


- 8 requests, 6 misses

25

Taking Advantage of Spatial Locality

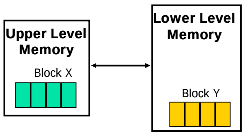
- Let cache block hold more than one word
  0 1 2 3 4 3 4 15



- 8 requests, 4 misses

26

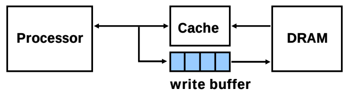
Handling Cache Hits (Miss is the next issue)



- **Read hits (I$ and D$)**
  - this is what we want!
- **Write hits (D$ only)**
  - allow cache and memory to be **inconsistent**
    - write the data only into the cache block (**write-back**)
    - need a **dirty** bit for each data cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be **consistent**
    - always write the data into both the cache block and the next level in the memory hierarchy (**write-through**) so don't need a dirty bit
    - writes run at the speed of the next level in the memory hierarchy – **so slow**! – or can use a **write buffer**, so only have to stall if the write buffer is full

27

Write Buffer for Write-Through Caching



- **Write buffer** between the cache and main memory
  - Processor: writes data into the cache and the write buffer
  - **Memory controller**: writes contents of the write buffer to memory
- The write buffer is just a **FIFO**
  - Typical number of entries: 4
  - Works fine if store frequency is low
- Memory system designer's nightmare, Write buffer **saturation**（飽和）
  - One solution is to use a write-back cache; another is to use an L2 cache

28

Sources of Cache Misses

- **Compulsory** (**初期参照ミス**, cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it
  - If you are going to run "millions" of instruction, compulsory misses are insignificant
- **Conflict** (**競合性ミス**, collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase **associativity**
- **Capacity** (**容量性ミス**)：
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

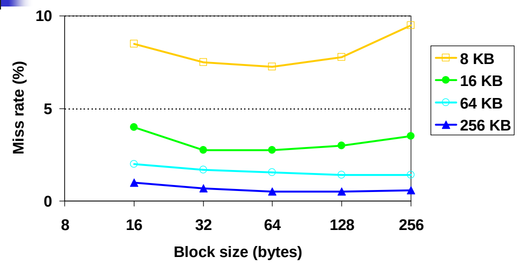
29

Handling Cache Misses

- **Read misses (I$ and D$)**
  - **stall**（ストール, 立ち往生させる）the entire pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- **Write misses (D$ only)**
  - **Write allocate** – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall
    - **stall** the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache, write the word from the processor to the cache, then let the pipeline resume
  - **No-write allocate** – skip the cache write and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full; must invalidate the cache block since it will be **inconsistent**

30

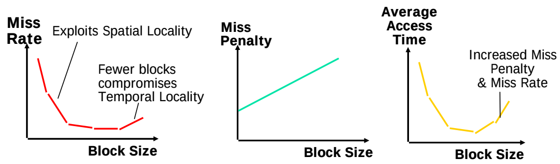
Miss Rate vs Block Size vs Cache Size



- Miss rate goes up if the block size becomes a significant fraction of the cache size
  because the number of blocks that can be held in the same size cache is smaller

Block Size **Tradeoff**

- Larger block sizes take advantage of spatial locality **but**
  - If the block size is too big relative to the cache size, the miss rate will go up
  - Larger block size means larger miss penalty
    - Latency to first word in block + transfer time for remaining words



❑ In general, **Average Memory Access Time**

= Hit Time + **Miss Penalty x Miss Rate**

Cache so far

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time
    - **Temporal Locality**: Locality in Time
    - **Spatial Locality**: Locality in Space
- Three major categories of cache misses:
  - **Compulsory misses**: sad facts of life. Example: cold start misses
  - **Conflict misses**: increase cache size and/or associativity
    Nightmare Scenario: ping pong effect!
  - **Capacity misses**: increase cache size
- **Cache design space**
  - total size, block size, **associativity** (replacement policy)
  - write-hit policy (write-through, write-back)
  - write-miss policy (write allocate, write buffers)