Lecture 10a - Single Processor Performance and Possible Parallel Programming Models

- **Performance of the single processor code has a big impact on the overall performances**.
- **CPU/memory speed gap:**
  - Processor performance doubles every 18 months: clock speed ~1ns
  - Memory performance doubles every 7 years: clock speed ~100ns
- **Memory hierarchy is used to solve this gap.**
- **Performance programming on single processor requires:**
  - understanding of memory system (levels, costs, sizes)
  - understanding of fine-grained parallelism in processor to produce good instruction mix

CISC (Old Technology)

- **<u>CISC</u> (Complex Instruction Set Computer) chip:**

  - Highly specialized machine instruction
  - Ideally suited for assembly programming
  - Non uniform instruction length (8,12,24,32 bits)

  Intel 80x86, Motorola 680x0, VAX

Instruction Pipeline (Speeding Up Processing)

- **Way to obtain one or more results per cycle**

- **Similar to vector-processing or streaming**

- **Basic idea: break-up the instruction into smaller parts**

1. Instruction fetch   op
2. Instruction decode   +
3. Fetch operands   x,y
4. Execute   x+y
5. Store results   z=x+y

1	2	3	4	5
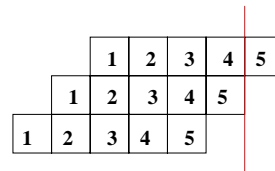
Variable-length instructions and variable execution times make implementing a pipeline difficult

RISC (Newer Technology)

- **<u>RISC</u> (Reduced Instruction Set Computer) chip:**

  - Reduced number of instructions
  - Uniform instructions length
  - Better pipelining
  - Advanced optimized compilers
  - Many registers

  Sparc, Alpha, MIPS, IBM Power

RISC Classes

- **Different classes of RISC processors:**

1. *Superscalars*: can do multiple instructions at the same time on the fly at runtime. Instructions must have the right mix (ex: one fp, one integer, and one memory op).

    IBM Power, Alpha

2. *Superpipelined*: refine the "granularity" of the pipeline even further.

    MIPS R4000

3. *Long instruction word processor* (LIW): processors get their performances from side by side operations. Scheduling cannot take place at runtime (hardware is simpler but compiler not):

    Intel i860

Memory Technology

- **DRAM (Dynamic Random Access Memory):**
  - Charge based devices: each bit is represented by an electrical charge stored in a capacitor. It needs to be continually refreshed:
    - The charge leaks away.
    - Reading the bit also discharge the capacitor
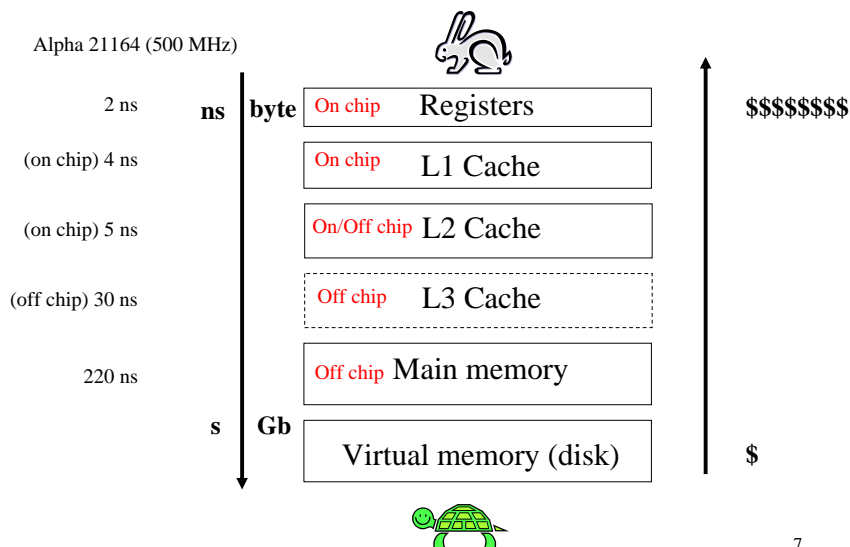   - Data cannot be accessed while the memory is refreshing

Best price/performance, high density: lower cost, less power, less heat

- **SRAM (Static Random Access Memory):**
  - Gates based. No refresh needed
    - Higher speed, higher cost
    - Cache, video memory
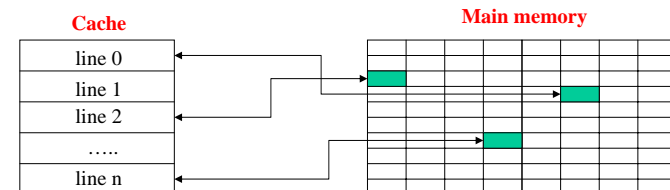
Memory Hierarchy

Alpha 21164 (500 MHz)

2 ns	**ns**	**byte**	On chip Registers	$$$$$$$$$
(on chip) 4 ns			On chip L1 Cache	
(on chip) 5 ns			On/Off chip L2 Cache	
(off chip) 30 ns			Off chip L3 Cache	
220 ns			Off chip Main memory	
	s	**Gb**	Virtual memory (disk)	$

Cache

- **The cache can be viewed as a linear memory divided into slots (cache lines):**

Cache		Main memory
line 0		
line 1		
line 2		
…..		
line n		

Cache size: total number of bytes available.
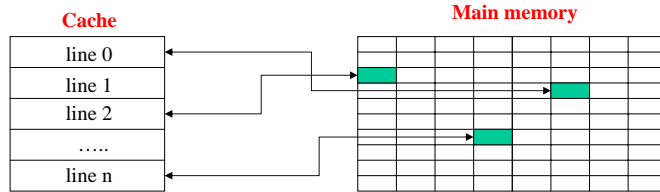Cache line size: number of bytes loaded together in one entry
Cache hit: the data is already in cache
Cache miss: the data is not in cache.

Cache Organization

- **The process of pairing memory locations with cache lines is called *mapping.***

- **Different cache organizations :**
  1) Direct mapped cache:
     A memory location is mapped always on the same line.

  2) Fully associative cache:
     Any memory location can be mapped into any cache line (better but expensive)

  3) Set associative cache:
     The cache is made up of different sets, each of which is direct mapped.

Cache Trashing

- **Alternating memory references to the same cache line:**
  Each reference causes a cache miss and replaces the entry just replaced

- **Direct mapped cache particularly affected by this problem**

- **Example: 4-KB (1K-word) direct mapped cache:**

```
real*4 A(1024),B(1024)
common/trash/A,B
do I=1,1024
  A(I)=A(I)*B(I)
end do
```

All the pairs A(I),B(I) want to use the same line!!!!

Cache Policy

- **Cache is optimized for reads since they dominate. Cache writes take longer than reads. There are two basic cache-write designs:**

- **<u>Write-back:</u>**
  - when a word is written only the cache is updated.
  - the modified cache line is marked *dirty*
  - when a dirty cache line is evicted, it must moved back to memory

- **<u>Write-through:</u>**
  - Memory is updated on every write

Example of Cache (IBM RS6000/590)

Cache size: 256KB

Cache line: 256 bytes (32 8-byte double words)

Mapping: 4-way set associative.

Write policy: write-back with LRU (Least Recently Used).

Cache hit: 1 cycle

Cache miss : 14-20 cycles

Cache line fill: 8 cycles (there is a quad load/store instruction)

Locality

• **The concept of Locality is exploited to make cache hits more efficient and computers faster**

A program generally spends 90% of its execution time in only 10% of the code. We can reasonably predict what instructions and data a program will use in the near future based on its accesses in the recent past.

•**Spatial locality:**

A good algorithm should use all the data located closed together in memory (they are on the same cache line)

• **Temporal locality:**

A good algorithm should reuse the data recently used. As long as a datum is used frequently enough, the cache line will not be evicted

Virtual Memory

• **Virtual memory (VM) decouples the addresses used by the program (virtual addresses) from the actual addresses where the data is stored in memory (physical addresses).**

• The program starts its address at 0.

• Virtual memory systems divide the program memory in *pages*.

• Page sizes vary from 512 bytes to several MB, depending on the machine.

• Pages don't have to be allocated contiguously (but the program still sees all the memory contiguously).

• The map containing translation from virtual to physical address is stored in *page tables*.

Translation Lookaside Buffer

• **Page faults (misses) of virtual memory are very time-consuming. To reduce the address translation time (between physical and virtual addresses), translation buffers are often used.**

•**TLB (translation lookaside buffer):** Special cache for virtual-to-physical memory address translation. It reduces the cost of memory references.

   • TLB lookups occur in parallel with instruction execution
   • TLB is limited in size. We can have TLB miss (as in the memory cache)

Compilers

• **Compilers and development tools are now very good.**

• **A good compiler will address most of the scheduling issues for you.**

• **Take your time to play with all the different options.**

• **Always use optimization for production runs!!!!!**

Software Pipelining

- **What is software pipelining?**
  - It allows the mixing of operations from different loop iterations in each iteration of the hardware loop.

- **Why is software pipelining used?**
  - It gets the maximum work done per clock cycle.

Alpha/EV68 Instruction Latency

- **Loads**
  - integer – 3 cycles D cache hit, 13+ miss
  - floating – 4 cycles D cache hit, 14+ miss
- **4 integer units**
  - add 1 cycle
  - multiply 7 cycle
- **2 floating point units, 1 adder, 1 multiplier**
  - floating add – 4 cycles, pipelined
  - floating multiply – 4 cycles, pipelined
  - floating divide (in adder) – 12 single, 15 double
  - square root (in adder) – 18 single, 33 double

do i=1,n
A(i) = A(i) + B(i)*C
end do

Example

Simple code	Unrolled 4x	Unrolled 4x, pipelined
1 ldt Ai; ldt Bi	1 ldt Bi; ldt Bi+1	
2	2 ldt Bi+2; ldt Bi+3	
3	3 ldt Ai; ldt Ai+1	
4	4 ldt Ai+2; ldt Ai+3	
5 mult Bi	5 mult Bi	
6	6 mult Bi+1	1 ldt Bi ;ldt Bi+1; mult Bi-4;addt Ai-8
7	7 mult Bi+2	2 ldt Bi+2 ;ldt Bi+3; mult Bi-3;addt Ai-7
8	8 mult Bi+3	3 ldt Ai ;ldt Ai+1; mult Bi-2;addt Ai-6
9 addt Ai	9 addt Ai	4 ldt Ai+2 ;ldt Ai+3; mult Bi-1;addt Ai-5
10	10 addt Ai+1	5 stt Ai-12;stt Ai-11
11	11 addt Ai+2	6 stt Ai-10;stt Ai-9;bne loop
12	12 addt Ai+3	
13 stt Ai; bne loop	13 stt Ai	
	14 stt Ai+1	
	15 stt Ai+2	
	16 stt Ai+3; bne loop	

13 cycles per iteration

4 iteration in 16 cycles = 4 cycles per iteration

4 iteration in 6 cycles = 1.5 cycles per iteration

Alpha EV68: 4 cycles for f.p. load from cache;
f.p. add and multiply 4 cycles (pipelined)