

CSE 141 – Computer Architecture Spring 2005

Lectures 15 Advanced Topics, Memory Hierarchy and Cache

Pramod V. Argade
May 18, 2005

Announcements

- **Reading Assignment**

Chapter 7: The Basics of Caches

– 7.1, 7.2

- **Homework 6:**

– 6.31, 6.32

– 7.2, 7.3, 7.4, 7.5, 7.9, 7.10, 7.12, 7.23

- **Quiz**

When: Mon., May 23rd, First 10 minutes of the class

Topic: Cache Chapter 7 **Need:** Paper, pen

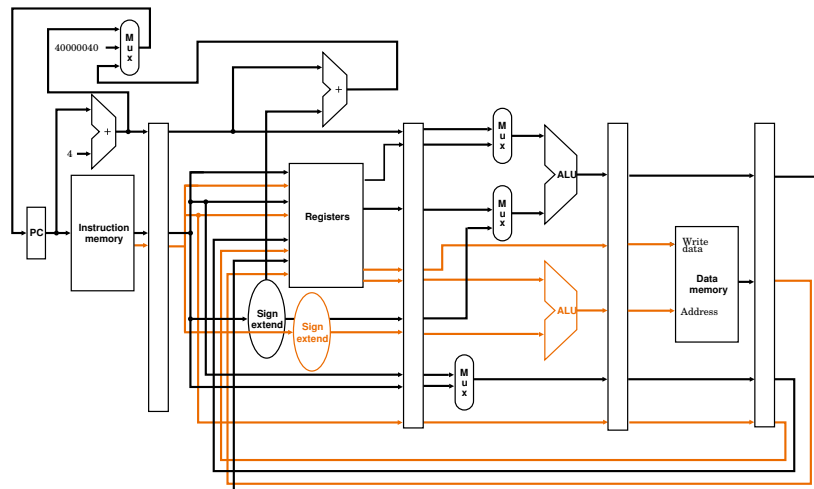
Advanced Techniques

Advanced Techniques

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{1}{\text{Clock Freq}}$$

- Superpipelining
 - More pipeline stages
 - Operand forwarding becomes complicated
 - Branch penalty is high
 - Must use branch prediction scheme
 - **Enables running the clock at higher frequency**
- Superscalar
 - Multiple pipelines executing in parallel
 - Each pipeline may be dedicated to a particular task (integer, float, mem)
 - Challenge is finding instructions in parallel
 - **Decreases CPI**

Superscalar MIPS Datapath



- Upto two instructions issued per clock: one integer ALU instruction and one LD/ST

Superscalar Issues

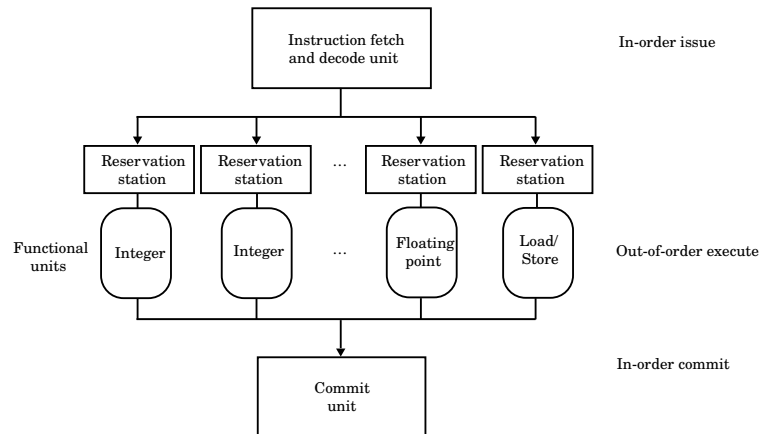
- Two instructions have to be fetched and decoded
 - 64-bits fetched at a given PC
- Additional ports are needed in the register file
 - Total 4 read ports, 2 write ports in our example
- Hardware resources have to be replicated
 - One ALU for arithmetic operation, another for MEM Address
 - Additional data forwarding paths, control logic, ...
- Problems
 - How to find multiple instructions to issue at run time?
 - Compiler technology needs to statically schedule instructions
 - Breaks binary compatibility
 - How to deal with a stall between LD and arithmetic instruction?
 - In this case, next two instructions cannot use load result w/o stalling

Advanced Techniques

Dynamic Pipeline Scheduling

- Dynamic pipelining
 - Execute instruction out-of-order to avoid pipeline hazards/stalls
 - A stalled instruction should not hold other instructions
 - Retire instructions in execution order (i.e. commit result)
 - **Decreases CPI**
- Three major sections
 - Instruction fetch and issue
 - Execute units
 - Each unit has reservation station to hold operands and operations
 - Instructions held in the reservation station until ready to execute
 - Commit unit
 - Common approach is in-order completion
 - Must discard instructions as a result of a mis-predicted branch

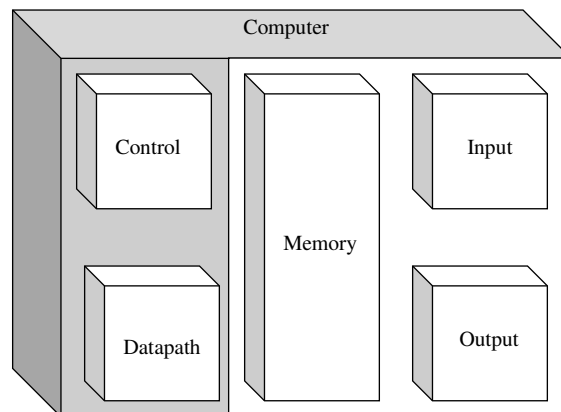
Dynamically Scheduled Pipeline



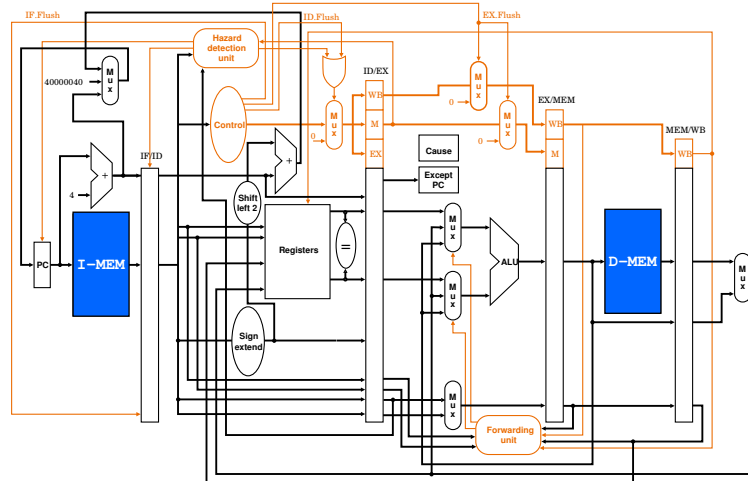
- **Very complex to design and verify**

Memory Hierarchy

Memory Systems

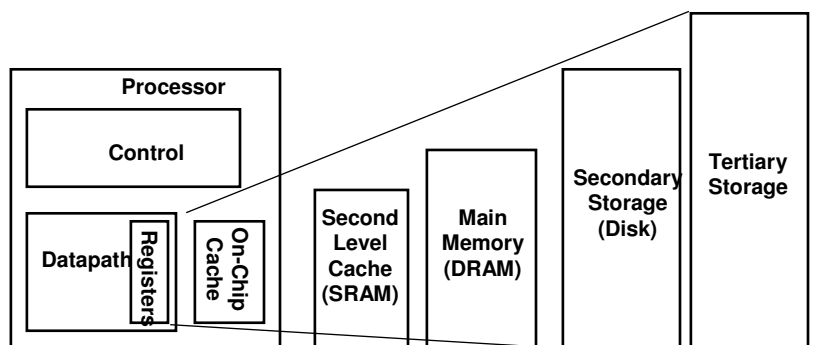


Pipelined Design: Datapath and Control



- Can arbitrarily large amount of I-MEM and D-MEM be accessed in a single cycle?

Memory Hierarchy in Computer Systems



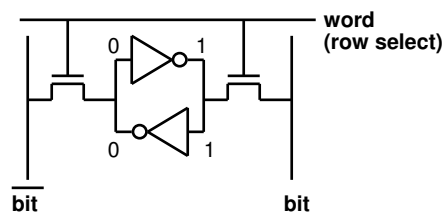
Speed:	1 ns	10's ns	100's ns	(10s ms)
Size (bytes):	100s	~ KBytes	~M Bytes	~G Bytes ~Tera Bytes
Cycles (3 GHz):	1	1- 10	100's	10's of Millions

Memory Subsystem Challenge

- Conflicting goals to provide:
 - Largest possible memory
 - At fastest access time
 - With lowest cost
- Processor speeds now exceed 3 Ghz (0.3 ns cycle time)
- DRAM access times are still ~10s of ns
- Serious Memory access gap
 - Every instruction has to be accessed from memory
 - ~15% of the instructions are load/store

Static RAM Cell and Data Access

6-Transistor SRAM Cell

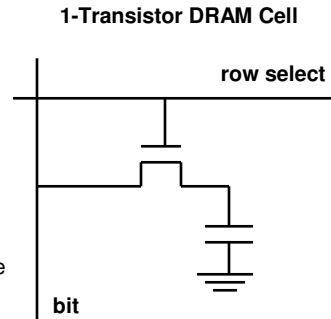


- **Write:**
 1. Drive bit lines to data
 2. Select row
- **Read:**
 1. Precharge bit and $\overline{\text{bit}}$ to Vdd
 2. Select row
 3. Cell pulls one line low
 4. Sense amp on column detects difference between bit and $\overline{\text{bit}}$

Fast access, large area (6 transistors per cell)

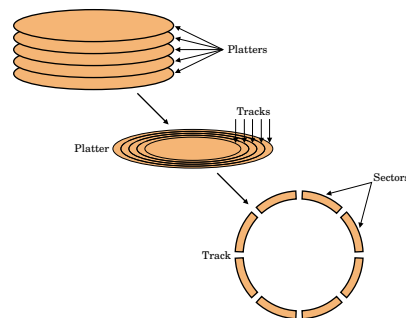
Dynamic RAM (DRAM) Cell and Data Access

- **Write:**
 - 1. Drive bit line to data
 - 2. Select row
- **Read:**
 - 1. Precharge bit line to V_{dd}
 - 2. Select row
 - 3. Cell and bit line share charges
 - Very small voltage changes on the bit line
 - 4. Sense voltage difference
 - Can detect changes of ~1 million electrons
 - 5. Write: restore the value
- **Refresh**
 - 1. Just do a dummy read to every cell



Slow access, small area (1 transistor per cell). Needs periodic refresh.

Magnetic Disk



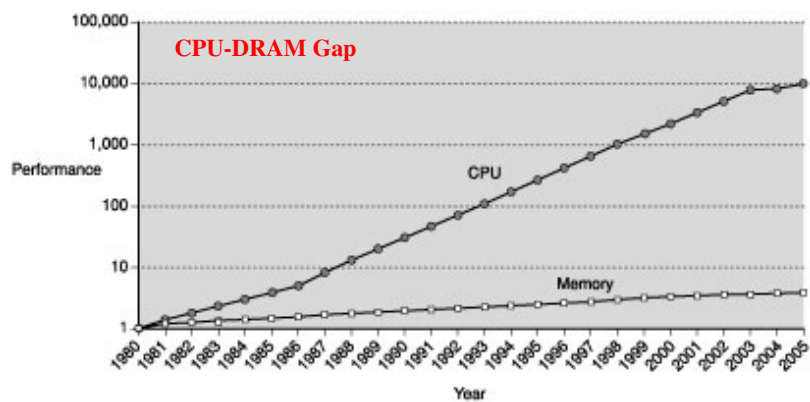
Average access time =
 Average seek time +
 Average rotational delay +
 Data transfer time +
 Disk controller overhead

Slow access (~ ms), very large capacity (100's GB)

Caches

Who Cares about Memory Hierarchy?

- Processor vs Memory Performance

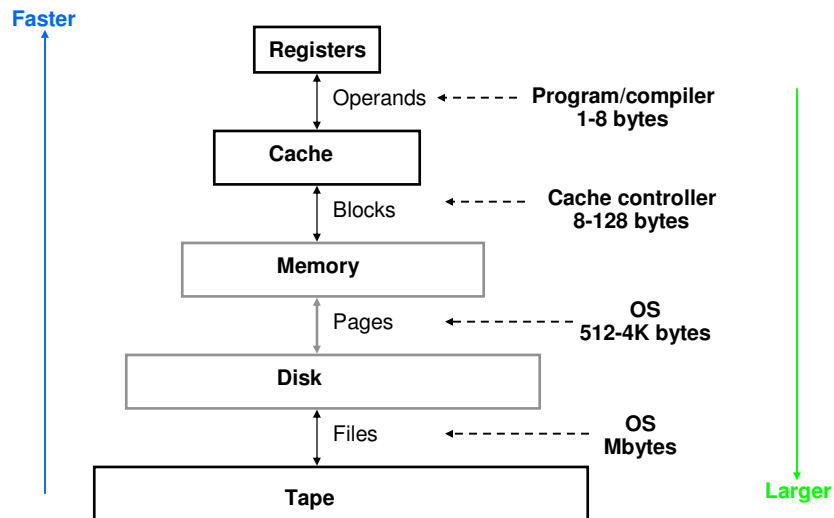


- Memory technology has not kept pace with Processor Performance
- Memory access time is the performance bottleneck

Memory Hierarchy and Locality

- **Memory locality** is the principle that future memory accesses are near past accesses
- There are two types of locality
 - **Temporal locality** -- near in time
 - > we will often access the same data again very soon
 - **Spatial locality** -- near in space/distance
 - > our next access is often very close in address to recent access
- Types(s) of locality in following address sequence?
1,2,3,4,7,8,9,10,8,8,4,8,9,8,10,8,8...
- Memory hierarchy is designed to take advantage of memory locality.
 - Cache is implemented with SRAM (fast, expensive)
 - Main memory is implemented with DRAM (cheap, slower)
 - Storage is disk and tape (very slow, cheap, vast)

Memory Hierarchy



What is a Cache?

- Dictionary meaning:
 - A hiding place used especially for storing provisions
- A cache is a small amount of fast memory
- Memory hierarchies exploit locality by *caching* (keeping close to the processor) data likely to be used again.
- It is impractical to build large, fast memories.
- Caches give an illusion of
 - Fast access time (of a SRAM)
 - With very large capacity (provided by a disk)

Locality and Caching

- A cache is a small amount of fast memory
- Memory hierarchies exploit locality by *caching* (keeping close to the processor) data likely to be used again.
- This is done because we can build large, slow memories and small, fast memories, but we can't build large, fast memories.
- If it works, we get the illusion of SRAM access time with disk capacity

SRAM (static RAM) -- 5-20 ns access time, very expensive

DRAM (dynamic RAM) -- 60-100 ns, cheaper

disk -- access time measured in milliseconds, very cheap

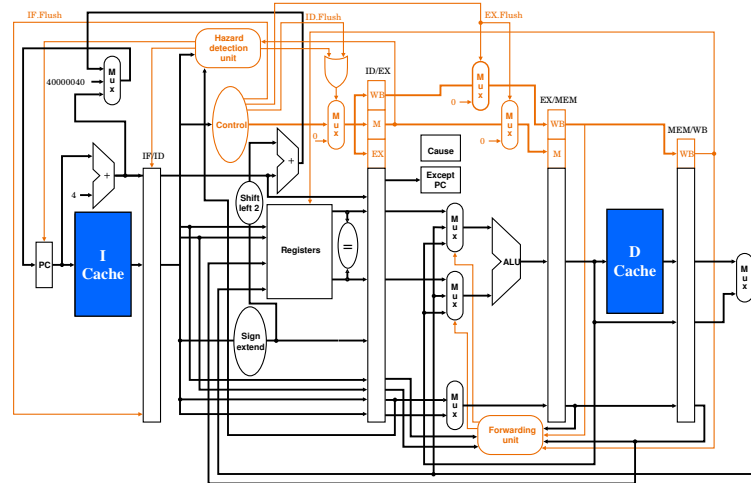
Cache Terminology

- **Instruction cache:** cache that only holds instructions
- **Data cache:** cache that only holds data
- **Split cache:** instruction and data cache are separate
 - Provides increased bandwidth from the cache
 - Hit rate is lower (than unified cache)
 - Wins over unified cache due to higher bandwidth
- **Unified cache:** cache that holds both instructions and data
 - Hit rate is higher
 - Bandwidth is lower (than that of of the split cache)

Cache Terminology

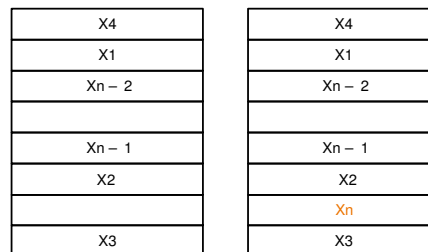
- **Cache hit:** an access where the data is found in the cache
- **Cache miss:** an access which is not found in the cache
- **Hit time:** time to access the cache
- **Miss penalty:** Time to process a cache miss
 - Move data from lower level memory to the cache and CPU
- **Hit ratio:** % of time the data is found in the cache
- **Miss ratio:** (1 - hit ratio)
- **Cache block size or cache line size:** the amount of data that gets transferred on a cache miss
- **Effective access time:**
 - $(\text{Hit Ratio} * \text{Hit Time}) + (\text{Miss Ratio} * \text{Miss Time})$

Pipelined Design: I-Cache & D-Cache



- How is the cache organized and managed?

How are Cache Entries Made?



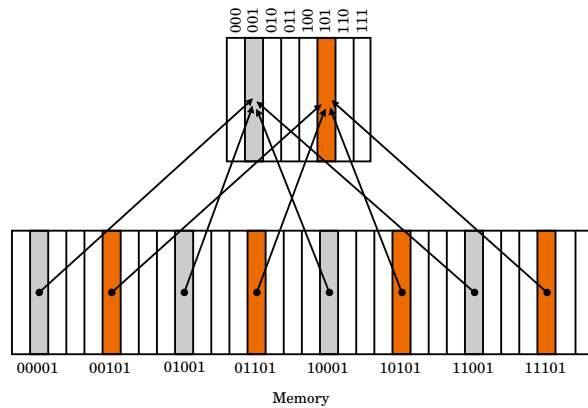
a. Before the reference to X_n

b. After the reference to X_n

- How is it determined whether the data for a given address is in the cache?
- In case of a miss, where is the data corresponding to the new address stored?

A Direct-mapped Cache

- If a data item is in the cache, how do we find it?
Cache location = (block address) modulo (Number of cache blocks in the cache)
- In the following case:
 - Number of cache blocks in the cache = 8



A Direct-mapped Cache, contd.

Address trace:

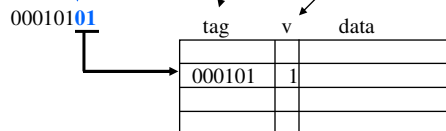
```

21 00010101
5  00000101
10 00001010
12 00001100
4  00000100
9  00001001
7  00000111
8  00001000
21 00010101
24 00011000
14 00001110
11 00001011
4  00000100
    
```

An index is used to determine which line an address might be found in the cache

The tag identifies a portion of address of the cached data

Valid bit indicates that entry is valid



4 entries, each block holds one word, each word in memory maps to exactly one cache location.

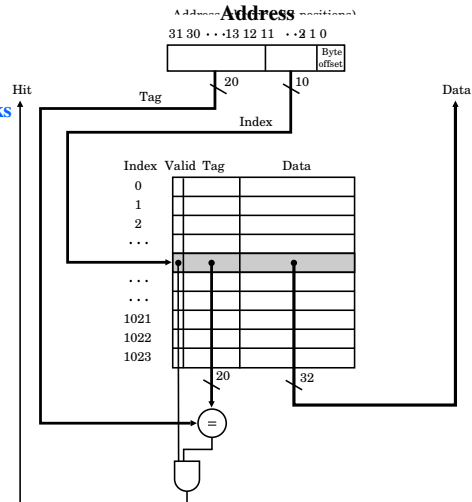
- A cache that can put a line of data in exactly one place is called a *direct-mapped cache*

How is a Block found in the Cache?

A 4 Kbyte Cache with a 1 word blocks
 Number of blocks
 = Cache Size/(Block Size)
 = 1 K

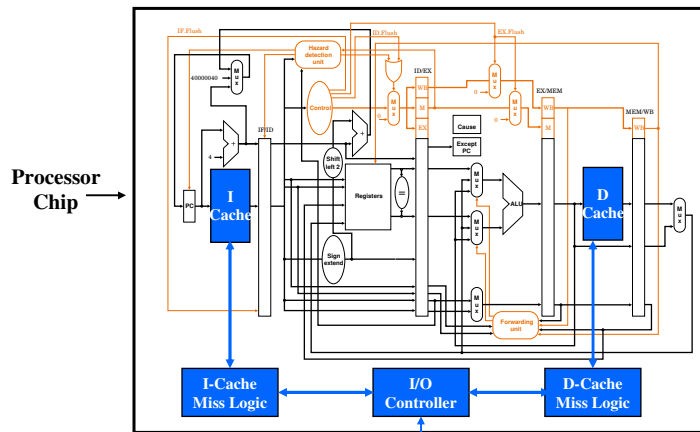
Index bits
 = $\log_2(\text{Number of blocks})$
 = 10 bits

Tag bits = (Total address
 - Index bits
 - Byte offset bits)
 = 32 - 10 - 2
 = 20



• 4 Kbyte Cache direct mapped cache with 1 word (4-byte) blocks

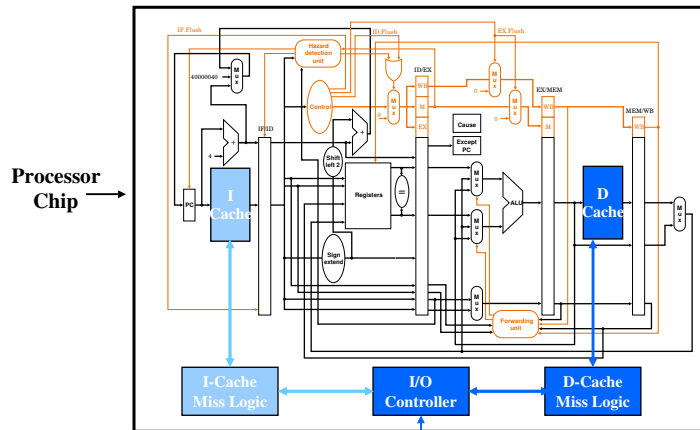
Handling a Cache Read Miss



A mis-match on tag and/or Valid bit indicates a miss. Stall CPU.

- Make read request to memory (via memory controller)
- When memory returns the data write it into the cache
- Return the data to the CPU

Handling a Cache Write Miss



A mis-match on tag and/or Valid bit indicates a miss.

- Write tag, valid bit and data into the cache
Works only if block size = word size
- Should the data be written to memory also?

Announcements

- **Reading Assignment**

Chapter 7: The Basics of Caches

- 7.1, 7.2

- **Homework 6:**

- 6.31, 6.32
- 7.2, 7.3, 7.4, 7.5, 7.9, 7.10, 7.12, 7.23

- **Quiz**

When: Mon., May 23rd, First 10 minutes of the class

Topic: Cache Chapter 7 **Need:** Paper, pen