CMP 1203

LECTURE 5

Memory: Types

- Memory can become a bottleneck with increased CPU speed: needs to “keep up”
- ***Cache memory*** eliminates need to constantly “speed up” memory: buffer for frequently accessed data
- ***RAM (Read-write)***
- Given in computer specs
- Main memory
- Stores programs and data
- Loses information when power is turned off

Characteristics of Memory

Table 4.1 Key Characteristics of Computer Memory Systems

Location
- Internal (e.g., processor registers, cache, main memory)
- External (e.g., optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

RAM

- Built using two types of chips: Static RAM and Dynamic RAM
- Both used in combination due to relative advantages and disadvantages

SRAM	DRAM
•Keeps contents as long as power is available •Faster •More expensive •Types: synchronous SRAM, asynchronous SRAM, pipe line burst RAM etc	•Made from small capacitors •Needs to be recharged every few milliseconds to keep its data •Denser (stores more bits per chip) •Less power consumption •Generates less heat •Types: **M**ultibank DRAM; **S**ynchronous DRAM, **D**ouble **D**ata **R**ate SDRAM, **R**ambus DRAM etc.

Homework: what differentiates all these different types?

ROM

- Stores critical information needed to run the system e.g. boot programs
- Always retains its data
- Also used in embedded systems or systems where programming doesn't need to change e.g. appliances, toys, cars, peripheral devices etc.
- Five types:

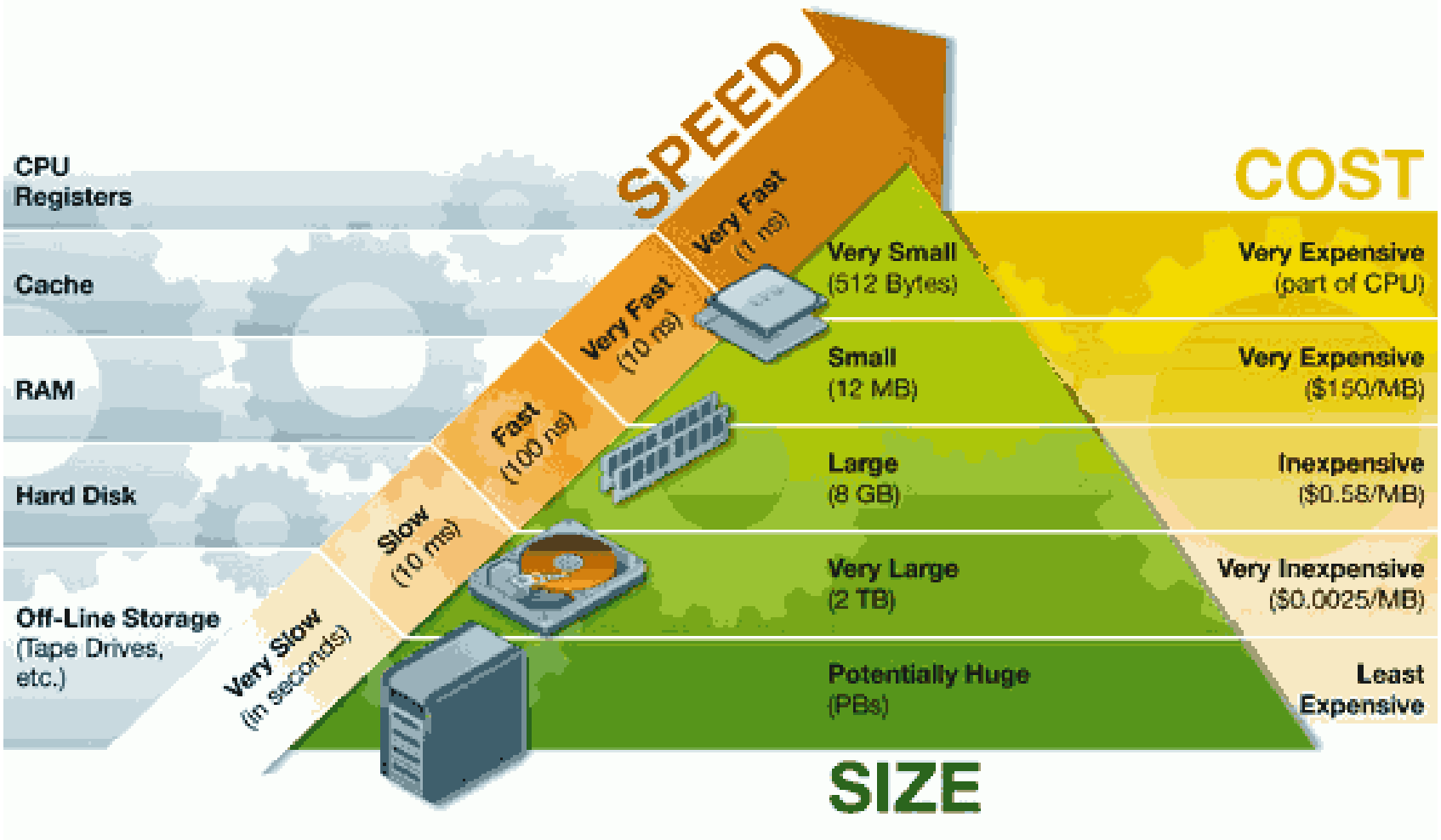
1. ***ROM***: hardwired
2. ***Programmable ROM (PROM)***: can be programmed by users with some special equipment. Once programmed, information cant be changed.
3. ***Erasable PROM***: reprogrammable but needs special UV light tools to erase
4. ***Electrically EPROM***: no special tools to erase chip and can be erased in portions – a byte at a time
5. ***Flash memory***: EEPROM that can be erased in blocks bigger than one byte hence its faster than EEPROM

Memory Hierarchy

- Different types of memory with varying cost and capabilities
- Use hierarchical memory to get best performance and best cost
- Five types are used in this hierarchical structure:

1. **Registers** – storage locations available on CPU itself
2. **Cache memory** – high speed used to temporarily store frequent accessed memory locations
3. **Main memory** – medium speed, larger than cache
4. **Secondary memory** – very large: made up of hard disk drives (magnetic or solid state) containing data that CPU doesn't directly access but contents transferred to main memory when needed.
5. **Off-line bulk memory** – Needs human/robotic intervention to access data. Includes tertiary and off-line storage. Data must be transferred first to secondary memory e.g. floppies, optical disks, removable drives which need to first be mounted

Memory Hierarchy



Trade-off: Cost Vs Size Vs Access Time

Image source:http://www.ruanyifeng.com/blog/2013/10/register.html

Memory Hierarchy

Terms

- **Hit-** requested data is found in a given level of memory
- **Miss** – requested data not found
- **Hit rate** – percentage of memory accesses found in a given level
- **Miss rate** – percentage of memory accesses not found
- **Hit time** – time required to access requested information in a level
- **Miss penalty** – time required to process a miss i.e. time to replace a block in an upper memory level + time to deliver requested data to processor

Memory Hierarchy

- CPU sends data request to smallest fastest memory i.e. cache
- If found in cache, data is loaded quickly , else forwarded to next lower level and so on
- Lower levels return requested data + data located around it i.e. entire blocks of memory
- Assume other data will be referenced in the near future
- Locality: since returns requested data located at X and at surrounding locations X+1, X+2, X-1 etc…. Processor can access this extra data
- If there was one miss for X, there will now be several hits for surrounding
- Locality of reference exists in 3 forms
- **Temporal locality** – recently accessed items tend to be accessed again in near future
- **Spatial locality** – accesses are clustered in the address space
- **Sequential locality** – instructions are usually accessed sequentially

Cache Memory

- Computer processors are very fast and constantly accessing information from memory
- Memory access times are smaller than processor speed
- Cache memory - small, temporary fast memory the processor uses for information it is likely to use again in the near future
- Computer has no way of knowing before hand what data is most likely to be used (unlike real world examples) so it uses locality principle ➔ transfer entire block from main memory to cache every time it has to make a main memory access
- Cache location for new block depends on: cache mapping policy and cache size

Cache memory

- Size varies
- Typically 512K for Level 2 cache and 8 – 64K for L1 cache
- L1 is on processor and L2 between CPU and main memory
- Many systems also have L3 cache
- Role is to speed up memory access by storing recently used data close to CPU
- Typically made of SRAM while main memory of DRAM
- Cache is accessed by content rather than by address.
- Alternatively called content addressable memory (CAM)

Cache Mapping

- CPU generates a memory address when accessing data or instructions
- How does it know where to find data in cache? Address of data in cache ≠ address in main memory ➔ cache mapping scheme: converts main memory address to a cache location

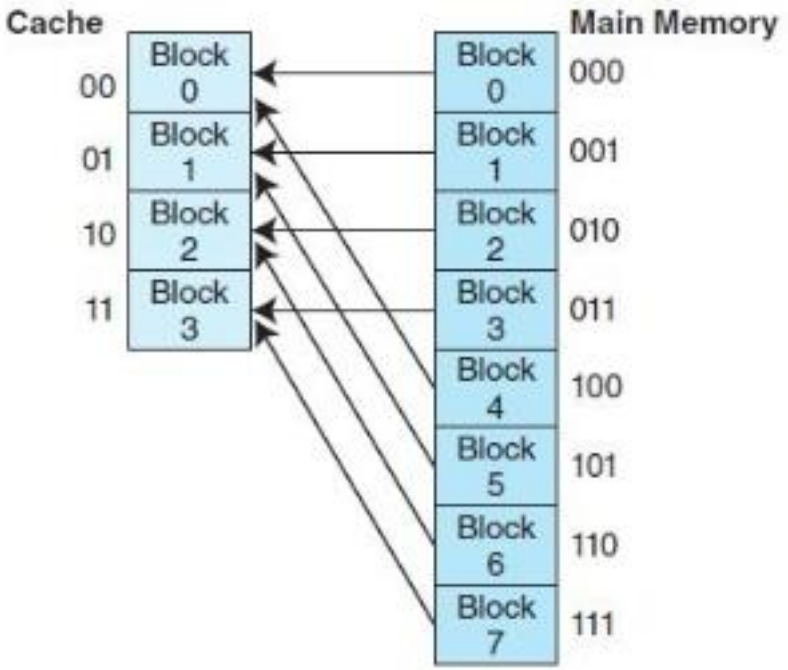
1. Divide bits in memory address into fields
2. 1 field of main memory address points to a location in cache where data can be found if it is there (cache hit) or where it is supposed to be put if its not there (cache miss)

Cache Mapping

3. Check referenced cache block to verify if its valid by using a valid bit: 0 ➔block not valid (cache miss) and main memory needs to be accessed, 1 ➔block valid (cache hit) but need to be sure
4. Compare tag in cache block to tag field in the address. Must be same. Confirms hit
5. Locate desired data in the block using offset field in memory address

Direct Mapped Cache

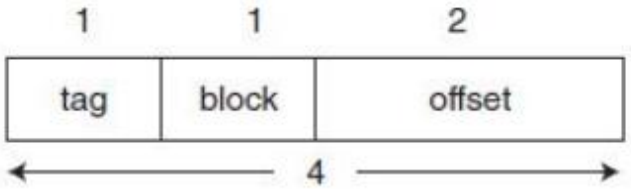
- Main memory blocks > cache memory blocks hence compete for space in cache
- Direct mapping: main memory block X mapped to cache memory block Y mod N ; N is total number of blocks in cache
- Each block has a tag stored with it so computer can i.d. which block is in cache



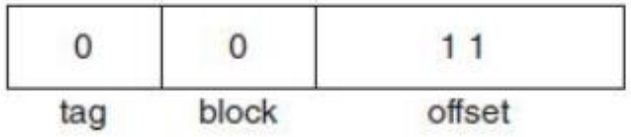
Direct Mapped Cache

- Format of main memory address using direct mapping
- Size of field determined by xtics of memory and cache
- Offset field: uniquely identifies address within a specific block – size determined by the number of bytes or words in block if it's byte or word addressable
- Block field: identifies unique block in cache – hence size determined by number of blocks in cache
- Tag field: identifies main memory block stored in cache- size is whatever is left over

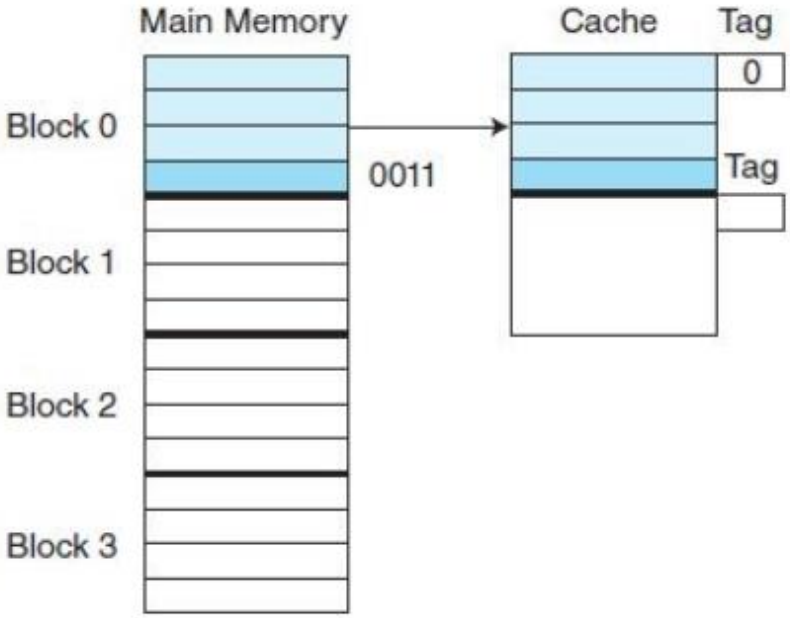
Example



a) Main Memory Format



b) The Address 0011 Partitioned into Fields



c) Mapping of Block Containing Address 0011 = 0x3

- Consider byte addressable memory with 4 blocks and cache with 2 blocks. Each block is 4 bytes.
- Memory is (4 blocks * 4 bytes) = 16 bytes hence memory address is 4 bits.
- Offset = 2 bits (block size), block field = 1 bit (2 cache block) , tag = 1 bit (remaining bits out of 4 in memory address)

Fully Associative Cache

- Direct mapping is cheap because no searching needed: each memory block has specific location it is mapped to in cache so CPU knows where to find each block
- Fully associative cache allows memory block to be placed anywhere
- Need to search for its tag on entire cache in parallel
- Needs special hardware (multiplexers and comparators) so very expensive
- Memory address has only tag and offset.
- E.g. memory with 2^{14} bytes, 16 block cache and 8 byte blocks: 3 bits for offset and 11 bits for tag

***offset i.d's addresses within cache block**

) The Main Memory Address Format for Associative Mapping

Set Associative Cache

- Direct mapping is restrictive if memory blocks using same cache block are accessed one after another.
- Fully associative mapping solves this but is expensive and makes cache larger because of larger tag
- N-way set associative mapping is in middle
- Cache divided into sets, each set has a number of cache blocks
- Like direct mapping since an address maps to a set of several cache blocks
- Each set is then treated as associative memory e.g. 2-way set associative mapping, byte-addressable, 16 block cache, 8 byte blocks and memory of 2^{14} bytes

FIGURE 6.12 Format for Set Associative Mapping for Example 6.5

- **2 way set implies 2 cache blocks per set**
- **Total blocks are 16 so in total there are 8 sets**

Replacement Policies

- Algorithms which determine victim block?
- Not needed for direct mapped because victim block is already pre-determined
- ***Optimal Algorithm***: "looks into future" to determine which block will not be used for longest period in future.
- Guarantees lowest miss rate
- Practically can't look into future but can apply it e.g. the 2nd time a program is run (because we know what was needed from first run) but cant predict for every single program
- So only used as metric to measure performance of other algorithms.
- The closer to optimal, the better
- ***Least Recently Used (LRU) Algorithm***: apply time stamp to block and throw out oldest
- ***First In First Out***
- ***Random Replacement***

Homework: read further!

Effective Access Time and Hit Ratio

- EAT or average time per access measures performance of hierarchical memory
- For a 2-level memory

$$EAT = H \times Access_C + (1 - H) \times Access_{MM}$$

H= hit rate, $Access_C$= Cache access time, $Access_{MM}$= main memory access time

- Formula can be applied to more level memories

Cache Write Policies

- When processor writes to memory, data may be written to cache assuming CPU will read it again soon.
- Write policy determines when actual main memory block is updated to match the cache block
- ***Write-through***
- Updates both cache and main memory at same time. Slow (requires MM access at every write) but ensures consistency
- ***Write-back***
- Updates MM only when cache block is selected as a victim and must be removed from cache. Faster but no consistency – data loss if a program crashes before write to MM is done

Further Reading

- Instruction and Data Caches
- Multilevel Caches
- Cache Sizing
- Locality

Handout

- Chapter 6 of Lobur and Null
- Chapter 5 and 6 of Stallings