

ELEC2041

Microprocessors and Interfacing

Lectures 31: Cache Memory - I

<http://webct.edtec.unsw.edu.au/>

May 2006

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec31-Cache-I.1 Some of the slides are adopted from David Patterson (UCB) Saeid Nooshabadi

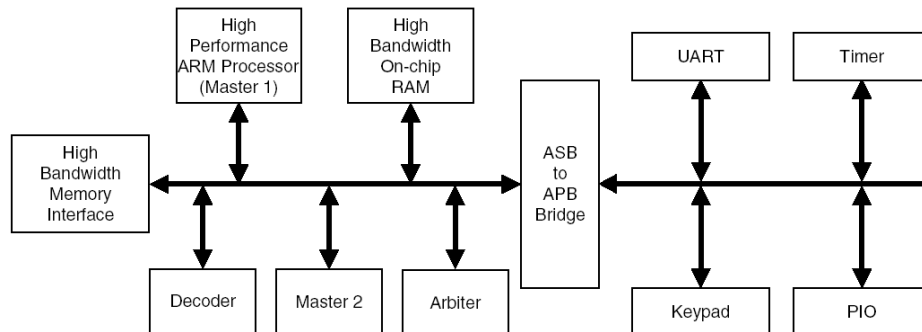
Outline

- Memory Hierarchy
- On-Chip SRAM
- Direct-Mapped Cache

ELEC2041 lec31-Cache-I.2

Saeid Nooshabadi

Review: ARM System Architecture



Fast on-Chip RAM
External Lower Speed SRAM, Slower DRAM,
Much Slower Flash-ROM

ELEC2041 lec31-Cache-I.3

Saeid Nooshabadi

Memory Hierarchy (#1/5)

- Processor
 - executes programs
 - runs on order of nanoseconds to picoseconds
 - needs to access code and data for programs: where are these?
- Disk
 - HUGE capacity (virtually limitless)
 - VERY slow: runs on order of milliseconds
 - so how do we account for this gap?

ELEC2041 lec31-Cache-I.4

Saeid Nooshabadi

Memory Hierarchy (#2/5)

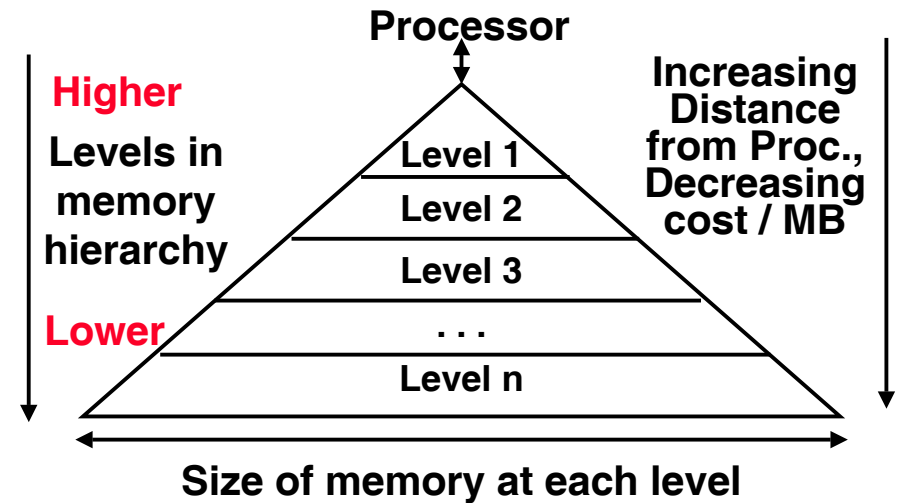
Memory (DRAM)

- smaller than disk (not limitless capacity)
- contains **subset of data** on disk: basically portions of programs that are currently being run
- much faster than disk: memory accesses don't slow down processor quite as much
- Problem: memory is **still too slow** (hundreds of nanoseconds)
- Solution: add more layers
 - On-chip **Memory**
 - On-chip **Caches**

ELEC2041 lec31-Cache-1.5

Saeid Nooshabadi

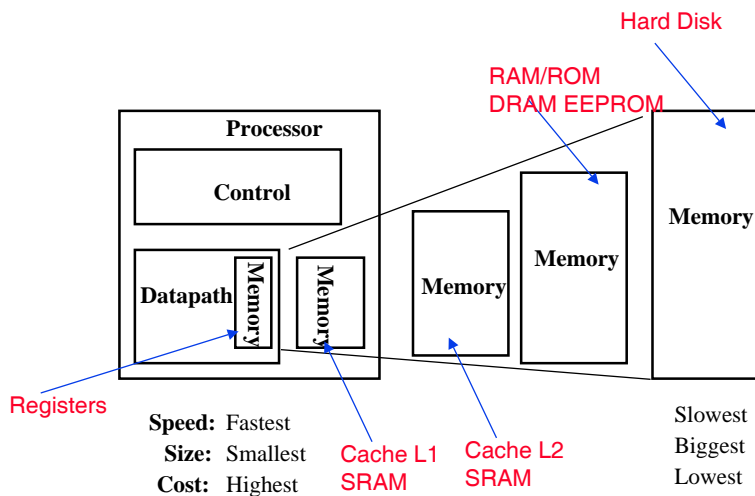
Memory Hierarchy (#3/5)



ELEC2041 lec31-Cache-1.6

Saeid Nooshabadi

Memory Hierarchy (#4/5)



ELEC2041 lec31-Cache-1.7

Saeid Nooshabadi

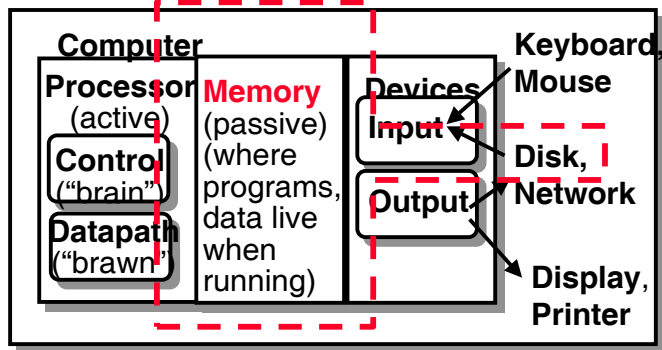
Memory Hierarchy (#5/5)

- If level is closer to Processor, it must be:
 - smaller
 - faster
 - **subset** of all lower levels (contains most recently used data)
 - **contain** at least all the data in all higher levels
- **Lowest Level (usually disk) contains all available data**

ELEC2041 lec31-Cache-1.8

Saeid Nooshabadi

Memory Hierarchy



◦ Purpose:

- Faster access to large memory from processor

Memory Hierarchy Analogy: Library (#1/2)

- You're writing an assignment paper (Processor) at a table in the Library
- Library is equivalent to **disk**
 - essentially limitless capacity
 - very slow to retrieve a book
- Table is **memory**
 - smaller capacity: means you must return book when table fills up
 - easier and faster to find a book there once you've already retrieved it

Memory Hierarchy Analogy: Library (#2/2)

- Open books on table are **on-chip memory/cache**
 - smaller capacity: can have very few open books fit on table; again, when table fills up, you must close a book
 - much, much faster to retrieve data
- Illusion created: whole library open on the tabletop
 - Keep as many recently used books open on table as possible since likely to use again
 - Also keep as many books on table as possible, since faster than going to library shelves

Memory Hierarchy Basis

- Disk contains everything.
- When Processor needs something, bring it into to all higher levels of memory.
- **On-chip Memory/Cache** contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Entire idea is based on **Temporal Locality**: if we use it now, we'll want to use it again soon (a Big Idea)

On Chip SRAM Memory

- Provides fast (zero wait state access to program and data)
- It occupies a portion of address space.
- Requires explicit management by the programmers.
 - Part of the program has to copy itself from slow external slow memory (eg flash-rom), into the internal on-chip ram and start executing from there
 - Works well for limited number of programs where, the program behaviour and space requirement is well defined.

ELEC2041 lec31-Cache-I.13

Saeid Nooshabadi

DSLMMU on-Chip RAM

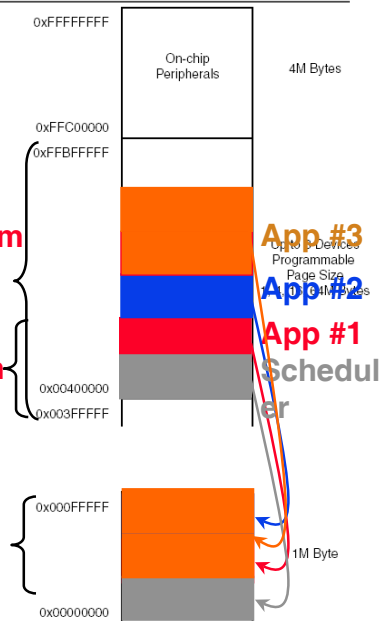
Applications are copied from the FLASH ROM to int. RAM to make them go faster

Works only if there are a few well behaved applications

External mem area

External Flash ROM

Int. SRAM



ELEC2041 lec31-Cache-I.14

A Case for Cache

- On-Chip SRAM requires explicit management by the programmer
 - Possible for an embedded system with small number of well defined programs
 - Not possible for a general purpose processor with many programs, where the application mix cannot be determined in advance
 - Explicit memory management become difficult
- We need a mechanism where the copying from the slow external RAM to Int. memory is automated by hardware (**Cache!**)

ELEC2041 lec31-Cache-I.15

Saeid Nooshabadi

Cache Design

- How do we organize cache?
- Where does each memory address map to? (Remember that cache is subset of memory, so multiple memory addresses map to the same cache location.) (Books from many shelves are on the same table)
- How do we know which elements are in cache?
- How do we quickly locate them?

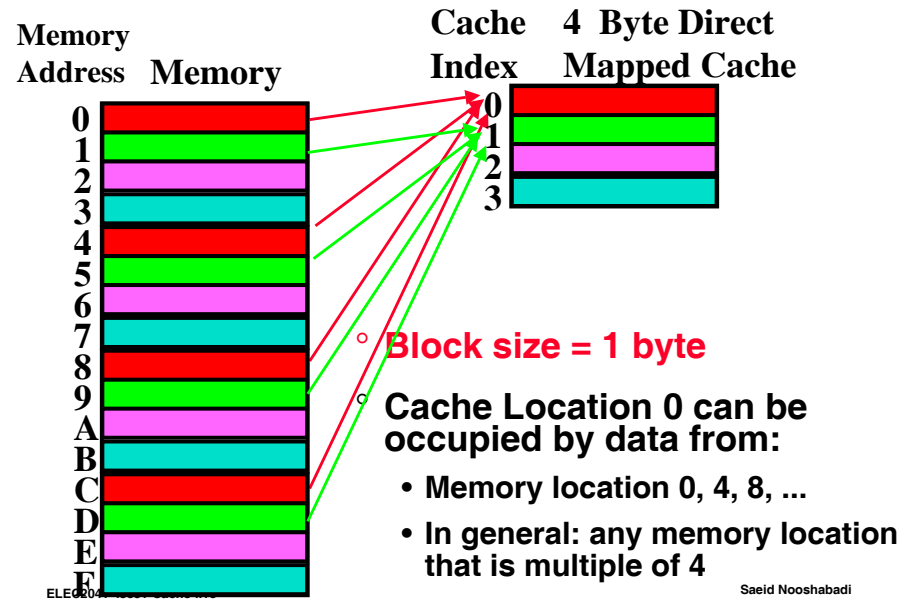
ELEC2041 lec31-Cache-I.16

Saeid Nooshabadi

Direct-Mapped Cache (#1/2)

- In a **direct-mapped cache**, each memory address is associated with one possible **block** within the cache
 - Therefore, we only need to look in a single location in the cache for the data to see if it exists in the cache
 - Block is the unit of transfer between cache and memory

Direct-Mapped Cache (#2/2)



Issues with Direct-Mapped

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- Store the address information along with the data in the cache

Address from the processor



address tag to check have correct block

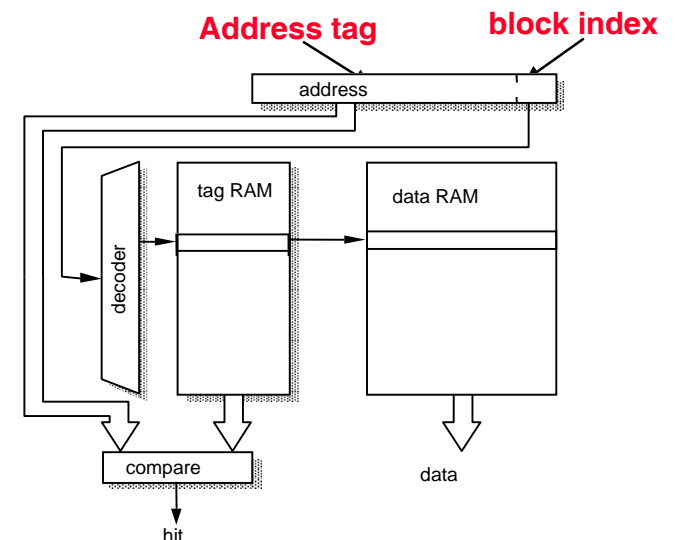
index to select block

Compare address tag with indexed value to check for match

Cache 4 Byte Direct Index Mapped Cache



Direct-Mapped with 1 Byte Blocks Example

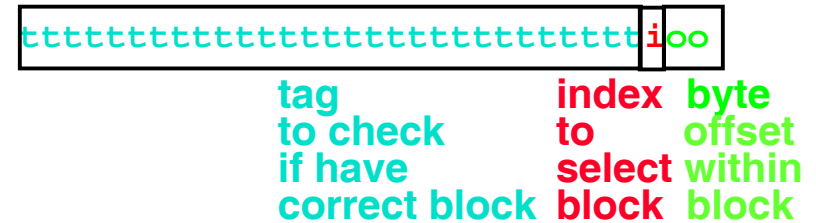


Reading Material

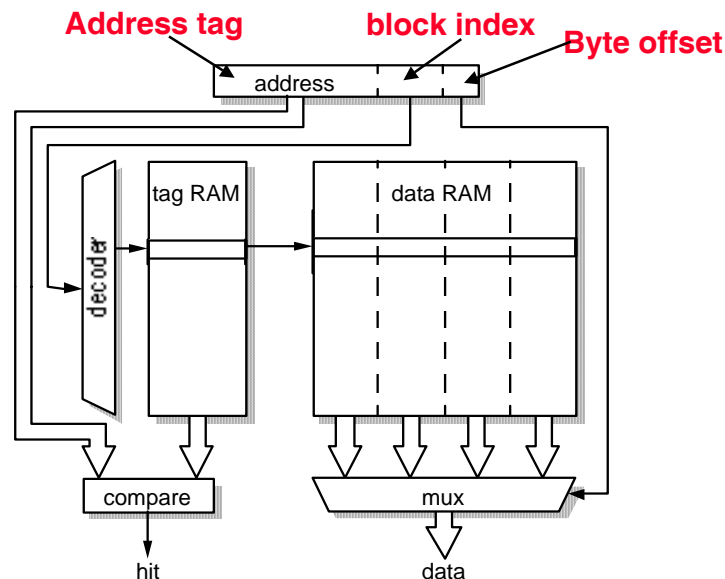
- Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. [Chapter 10](#).

Issues with Direct-Mapped with Larger Blocks

- Since multiple memory blocks map to same cache index, how do we tell which one is in there?
- How do we select the bytes in the block?
- Result: divide memory address into three fields



Direct-Mapped with Larger Blocks Example



Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- **Index**: specifies the cache index (which "row" or "line" of the cache we should look in)
- **Offset**: once we've found correct block, specifies which byte within the block we want
- **Tag**: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

Direct-Mapped Cache Example (#1/3)

- Suppose we have a 16KB of data in a direct-mapped cache with 4 word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture (ie. 32 address lines)
- Offset
 - need to specify correct byte within a block
 - block contains
 - 4 words
 - 16 bytes
 - 2^4 bytes
 - need **4 bits** to specify correct byte

Direct-Mapped Cache Example (#2/3)

- Index: (~index into an "array of blocks")
 - need to specify correct row in cache
 - cache contains 16 KB = 2^{14} bytes
 - block contains 2^4 bytes (4 words)
 - # rows/cache = # blocks/cache (since there's one block/row)
 - = $\frac{\text{bytes/cache}}{\text{bytes/row}}$
 - = $\frac{2^{14} \text{ bytes/cache}}{2^4 \text{ bytes/row}}$
 - = 2^{10} rows/cache
 - need **10 bits** to specify this many rows

Direct-Mapped Cache Example (#3/3)

- Tag: use remaining bits as tag
 - tag length = mem addr length
 - offset
 - index
 - = 32 - 4 - 10 bits
 - = 18 bits
 - so tag is leftmost **18 bits** of memory address
- Why not full 32 bit address as tag?
 - All bytes within block need same address (-4b)
 - Index must be same for every address within a block, so it is redundant in tag check, thus can leave off to save memory (- 10 bits in this example)

Things to Remember

- We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.
- So we create a memory hierarchy:
 - each successively lower level contains "most used" data from next lower level
 - exploits **temporal locality**
- Locality of reference is a Big Idea