5. Internal Memory

(a) Categories of Internal and External Memory.

(b) Organization of Main Memory.

(c) Cache Memory.
(a) Categories of Internal and External Memory

Location

- Processor (Registers, some cache).
- Internal (Main memory).
- External (e.g. disk, tape).

Capacity

External and main memory measured in terms of

- bytes (1 byte = 8 bits),
- Kilobytes (= 1024 bytes) (Kbyte),
- Megabytes (= 1024 Kilobytes) (Mbyte),
- Gigabytes (= 1024 Megabytes) (Gbyte),
- Terabytes (= 1024 Gigabytes) (Tbyte).
**Remark**

Clock rates are measured in Hz (Hertz).

1 Hz = 1 (signal) per second,
2 Hz = 2 (signals) per second.

1 KHz = 1000 Hz. (**not** 1024 Hz!!)
1 MHz = 1000 kHz = 1 million (signals) per second.
1 GHz = 1000 MHz.

Note that 1kHz is **not** 1024 Hz!!
So the period between two impulses is in case of

<table>
<thead>
<tr>
<th>Hertz</th>
<th>duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Hz</td>
<td>1s (second)</td>
</tr>
<tr>
<td>2Hz</td>
<td>0.5s (seconds)</td>
</tr>
<tr>
<td>1KHz</td>
<td>$\frac{1}{1000}$s = 1ms (milisecond)</td>
</tr>
<tr>
<td>1MHz</td>
<td>$\frac{1}{1000\ 000}$s = 1µs (microsecond)</td>
</tr>
<tr>
<td>1GHz</td>
<td>$\frac{1}{1000\ 000\ 000}$s = 1ns (nanosecond)</td>
</tr>
</tbody>
</table>
Unit of Transfer (Granulation)

- **Word** = natural unit of organization of memory. Typically the number of bits used to represent a number and smallest length of an instruction (typically 2, 4, 8 bytes)

- **Addressable units** Usually word. Sometimes byte level addressable. If address has $A$ bits, the number of addressable units is $2^A$.

- **Unit of transfer**: Number of bits written out or from memory at a time. Not necessary a word. For external memory, often transfer in much larger units than words. Usually called blocks.
Method of Access.

- **Sequential Access:** Access of memory sequentially. Tape units have sequential access.
- **Direct Access:** Blocks have a direct address. Location within such a block is accessed sequentially. Example: disks.
- **Random Access:** Each addressable location of the memory can be addressed and accessed directly. Main memory some cache memory.
- **Associative:** Random-access type memory. One part of the address selects directly a sequence of addresses in the associative memory. The associative memory then verifies, if the rest of the address (tag) coincides with one of the tags of the addresses stored in it, and has, if yes, as result the value stored there. Will be discussed in the part about cache.
Implementation of Associative Memory

Requested Address

Compare

Tag of 1st line

Data of 1st line

(b = 1 bit)

Compare

Tag of 2nd line

Data of 2nd line

From other cache lines

From other cache lines

Hit/Miss

Read Data

From other cache lines
Performance Parameters

• **Access time:**
  - For *random-access memory*, time to perform read or write operation:
    Time from the instant that address is presented to the instant that data are stored or made available for use.
  - For *non-random-access memory*, time till read-write mechanism reaches the desired location.

• **Memory cycle time:** Applies to random-access memory.
  Access time plus additional time, till next access is possible.
Correction (Not Correct in Stallings)

- **Transfer rate**: Rate at which data can be transferred in or out (measured in bits per second, bps).
  - For random-access memory:
    
    \[
    \frac{K}{\text{memory cycle time}},
    \]
    
    where
    
    * \( K \) is the number of bits transferred in parallel, i.e. width of the bus.

  - For non-random access memory
    
    \[
    \frac{N}{T_A + \frac{N}{R}} = \frac{1}{\frac{T_A}{N} + \frac{1}{R}},
    \]
    
    where
    
    * \( T_A \) = average access time,
    * \( N \) = average number of bits per transfer,
    * \( R \) = transfer rate in bps.
Physical Types

- **Semi-conductor memory.**
- **Magnetic surface memory** (disk, tape).
- **Optical** memory.
- **Magneto-optical** memory.

Physical Characteristics

- **Volatile Memory**: information lost after power switched off.
- **Non-volatile**: information remains. (e.g. magnetic surface memory, ROM, PROM, EPROM, EEPROM, flash memory, old core memory; see below).
- **Erasable memory**: memory can be over-written.
- **Nonerasable memory**: Nonerasable memory is always nonvolatile.
Picture omitted for copyright reasons
(DVD = digital video disk; stores like a CD, but on a tape; enormous storage capacity: single layer: 4.7 GByte; double layer 8.5 GByte – CD has 774 MByte).

WORM = write once read many CD.

MO = magneto-optical disk, with magnetic storage but using a laser to store information more densely)

If one goes upwards in the memory hierarchy

- **Cost per bit** increase.
- **Capacity** decreases.
- **Access time** decreases.
- **Number of accesses** by processor increases.
Types of Random-Access Semiconductor Memory

- True Random-access memory (RAM). Electrically stored, erasure on byte level, volatile.
  Two sorts:
  - **Static RAM**: Uses flip-flops, holds data as long as power is supplied. More expensive per bit but faster than dynamic RAM. Used for small fast portions of memory.
  - **Dynamic RAM**: Based on capacitors, which discharge over time, needs periodic refresh (typical time: 4 ms). Smaller, therefore more dense. Used for large memory requirements.
• Read-only Memory (ROM).
  Created like a circuit chip with data stored using gates.
  Large cost in development, no room for error.
  Used for
  – Microprogramming of the CPU.
  – Function tables, frequently wanted functions, system programs.

• Programmable ROM (PROM).
  Nonvolatile, can be written only once.
  Writing process performed electrically, can be performed after the chip is produced.
  Special equipment for programming needed.
  Still attractive for high-volume productions.
• **Read-mostly Memory**: Write operation difficult, but can be performed several times.
  – Erasable read-only programmable memory (**EPROM**). Read and written electrically,
    One transistor per chip, therefore high density.
    Erasure possible using UV radiation. Takes 20 minutes.
  – Electrically erasable programmable read-only memory (**EEPROM**).
    Can be written to without erasing prior contents.
    Write operation takes approx several hundred microseconds per byte.
    More expensive and less dense than EPROM.
  – **Flash memory**. Electrical erasing technology like EPROM, EEPROM. Entire memory can be erased in one or a few seconds.
    Blocks of memory can be erased.
    No byte-level erasure.
    Density as for EPROM.
(b) Organization of Main Memory

Two extremes:

• Addressable up to word level. Cheaper, more dense, higher access time.
• Addressable up to bit level.
Example: 16-Mbit DRAM (next slide).

- Organized as four arrays of 2048x2048 bits each.
  - **Row decoder** used to access with 11 bits one of the $2^{11} = 2048$ rows of the memory block.
  - Similarly with **Column Decoder**.
- Output via 4 data lines (4 bits, D1 - D4).
- Address is **multiplexed**: First the 11 address bits A0 - A10 specify the row, in a next period the column.
- Timing done via Row address select (**RAS**) and column address select (**CAS**) signals.
- Refresh circuitry. **Refresh counter** disables DRAM chip.
  For each row, output lines from refresh counter supplied to the row-decoder, RAS line activated. Cell row is then refreshed.
Picture omitted for copyright reasons
Next slide: EPROM and DRAM chips.

- A0 - A19: Address lines.
- D0- D7: data lines.
- Vcc: Power supply.
- Vss: Ground pin.
- CE (chip enable): indicates, whether address is valid for the chip or not. (since there are more memory chips in use).
- Vpp: program voltage used during programming (write operation, for EPROM).
- WE: Write enable (for DRAM chip).
- OE: Output enable (for DRAM chip).
- RAS: row access select (for DRAM chip).
- CAS: column access select (for DRAM chip).
- NC: not connected (so that pins of the chip match pins of the socket).
Picture omitted for copyright reasons
Module Organization:

- Next slide: Organization of 256 Kbyte memory, using eight 256 kilobit chips. Each chip stores 1 bit of the word. (Word length = 8 bit = 1 byte).
- Second slide: Organization of 1 Mbyte memory, using 32 chips à 256 kilobit. The two MSB of the Memory address register select the group of chip, and give a corresponding chip-enable signal. The other bits are as before connected with the chips.
To be copied from the book
To be copied from the book
(c) Cache Memory

• Cache = buffer between CPU and main memory.
• Normally constructed from SRAM (static RAM).
• Stores **blocks** of memory.
  Typical length of a block: 2/4/8/16 bytes.
• Tag for each line in the cache, shows which memory cell the cache block is coming from.
Picture omitted for copyright reasons
Cache functions as follows:

- If CPU requires memory access, first it is checked whether this is in the cache.
- If a **hit** is found, cache returns the data.
- Otherwise a **miss** is declared. Cache reads block of memory containing memory location into a cache line, and passes data to the CPU.
Picture omitted for copyright reasons
Cache Size

*Tradeoff* between different cache sizes: The bigger the cache the slower it is.
Principle of *locality of reference*: Memory references tend to cluster.
Studies suggest that cache sizes between 1K and 512 K words are most effective.
Write Policy

When we store data in memory via cache, when do we replace data in main memory?

2 main policies:

- **Write through:**
  When writing data to the memory in cache, write it to main memory as well.

- **Write back:**
  - Associate with line \( l \) in cache a bit \( \text{modified}(l) \), which is 1, if the cache line has been changed without adapting the corresponding main memory location.
  - Whenever loading main memory into cache line \( l \), \( \text{modified}(l) := 0 \).
  - When writing data to cache memory in line \( l \), \( \text{modified}(l) := 1 \) (main memory remains unchanged).
  - When replacing cache line \( l \) then
    - If \( \text{modified}(l) = 1 \), write content of old cache line to main memory.
    - Otherwise just overwrite cache line \( l \).
Cache Coherence

Problem when several processors (or I/O with access to memory) share portions of main memory.

Cache coherency required. Approaches:

- **Bus watch with write through.**
  Write through policy for all caches. Cache of each processor watches address bus for writes to shared memory. If cache contains address written to it, invalidate cache line.

- **Non-cacheable memory.**
  Forbid cache of a processor to store memory shared by other processors.

- **Hardware transparency.**
  Extra hardware ensures that writes to a cache are mirrored in main memory and other copies held in other caches.

- Use special protocols like the **MESI-protocol**.
MESI Protocol

- One of 4 states associated with every cache line:
  - **Modified**: Line in cache has been modified (different from main memory), available only in this cache.
  - **Exclusive**: Line in cache is the same as that in main memory, and is not present in any other cache.
  - **Shared**: The line in cache is the same as that in main memory, and may be present in another cache.
  - **Invalid**: The line in cache does not contain valid data.

- Let in the following master be any unit which can request access to memory via the cache.
Coherence Conditions for Caches

We will consider invalid lines as non existent. MESI ensures that at any time either

- a memory address corresponds to no cache line, or
- it corresponds to exactly one cache line, which has state exclusive, and content there is identical with main memory, or
- it corresponds to exactly one cache line, which has state modified, or
- it corresponds to lines in several caches, all of which have state shared, all contents of those lines coincide with main memory.

Initial Case:

- All cache lines obtain state invalid.
Read Command

Assume master requests data from main memory.

- Case address corresponds to no cache line.
  Block corresponding to that address is loaded into cache and passed through to master. Cache line obtains state exclusive.
- Case address corresponds to one line in current cache.
  Data is read from cache. State of cache line remains unchanged.
- Case address corresponds to no line in current cache, but to exclusive or shared line in some other cache.
  Data in main memory is valid. Cache reads block from memory. All cache lines (including the current one) corresponding to that address obtain status shared.
• Case address **corresponds to line in other cache which is modified**.  
  That cache writes block to memory, modifies status of that line to **shared**.  
  Block is read into current cache, status of that line is now shared.  
  Data is passed on to master.
Write Command
Assume master writes data to memory.

- Case address **corresponds to no cache line**.
  Data is written to cache, other bytes needed are read into cache, line obtains state **modified**.
  (Alternatively write data only to main memory and not to any cache line).

- Case **address corresponds to line in current cache**.
  Data is written to current cache, line obtains state **modified**.
  Other corresponding cache lines (if line was shared) obtain state **invalid**.
• Case **address corresponds to no line in current cache, but to shared or exclusive line in other cache.**
  Corresponding lines in other caches become invalid.
  Block is written into current cache, which obtains status modified.
  Byte in that block is modified.

• Case address **corresponds to no line in cache, but to modified line in some other cache.**
  The other cache writes its block back to main memory,
  sets state of its cache line to **invalid.**
  Now current cache writes data to its cache, loads missing bytes from main memory,
  state of line is now **modified.**
Picture omitted for copyright reasons
Two level Cache

- On-chip cache is now technical standard (integrated in the processor).
- Additional not-on-chip cache of advantage, since it can store more memory (order of 256 KByte)
- L1 cache in the processor, L2 cache separated.
Split vs. Unified Cache

- **Unified cache** contains both data and instruction.
- **Split cache** provides separate caches for data and instructions.

**Advantage of split cache** in pipelining: Instruction cache can be accessed for fetching instructions while simultaneously data cache can be accessed for fetching data.
3 Mapping Functions

**Question:** Which memory blocks correspond to which cache lines?

We assume for the following 3 mapping functions as an example:

- Cache size: 64 Kbytes.
- Block size: 4 bytes.
- Therefore Cache has $2^{14}$ lines of 4 bytes each.
- Main memory: 16 Mbytes.
- Each byte addressable.
- Therefore address has 24 bits ($2^{24} = 16$M).

Further we apply a write through policy (Otherwise, the process of writing data to main memory via the cache and of replacing cache lines has to be changed accordingly).
Direct Mapping

- Each block of memory is associated with a unique cache line. Since cache size is smaller than memory, many blocks of memory are associated with the same cache line.
- Memory address divided as follows:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Line</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>14 bits</td>
<td>2 bits</td>
</tr>
</tbody>
</table>

- 2 bits for the byte within a block (4 bytes per block addressed by 2 bits).
- 14 bits for the line corresponding to the $2^{14}$ cache lines.
- 8 bits of tag, corresponding to the mapping of $2^8$ blocks of memory into the same cache line.
• Cache memory stores in each line a block of memory together with its tag.
• Using the tag, the number of the line and the number for a byte within the block, we can identify the address in memory for each byte of a block in a cache line.
• Assume master requests a byte from main memory.
  Divide address into tag $t$, line $l$, word $w$.
  – Compare $t$ with the tag $t'$ of cache line no. $l$.
  – If $t = t'$, send byte $w$ stored in cache line $l$ to master.
  – Otherwise load block with tag $t$, line $l$ into cache.
  Replace tag of cache line $l$ by $t$.
  Send word $w$ to master.
(If we had no write through policy and cache line was modified, we had to up-date main memory first).
• Assume master requests to store data in memory.
  Divide address into tag $t$, line $l$, word $w$.
  – Write byte to main memory (write through policy).
  – Compare $t$ with the tag $t'$ of cache line no. $l$.
  – If $t = t'$, store data at word-position $w$ in cache line $l$.
  – If $t \neq t'$, load corresponding block in main memory into cache, replace tag by $t$.
  Alternatively, if $t \neq t'$, only store byte $w$ in main memory.
Picture omitted for copyright reasons
Fully Associative Mapping

• Each block of memory can be associated with any cache line.
• Use of associative memory in order to find whether block is in cache.
• Memory address divided as follows:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  22 bits  2 bits

  – 2 bits for the byte within a block (4 bytes per block addressed by 2 bits).
  – 22 bits for the tag.
• Cache memory stores in each line a block of memory together with its tag.

• Assume master requests a byte from memory.
  Divide address into tag $t$ and word $w$.
  – Verify whether any cache line $l$ has same tag.
  – If yes, send byte $w$ stored in cache line $l$ to master.
  – Otherwise load block with tag $t$ into one cache line which gets this tag.
    (Which one see below).
    Send word $w$ to master.
• Assume master requests to store a byte in memory.
  Divide address into tag \( t \) and word \( w \).
  – Write byte to main memory (write through policy).
  – Compare \( t \) with the tags of all cache lines.
  – If \( t \) is equal to one of these, store byte \( w \) in that cache line \( l \).
  – Otherwise load corresponding block in main memory into one cache line, set tag to \( t \).
  (Modification of cache might be omitted).
Picture omitted for copyright reasons
Set-Associative Mapping

- Compromise between direct and associative mapping.
- Cache is divided into $v$ sets of $k$ lines each.
- Consider for instance $k = 2$. In our example we have therefore $\frac{2^{14}}{2} = 2^{13}$ sets.
- Memory address divided as follows:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Set</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 bits</td>
<td>13 bits</td>
<td>2 bits</td>
</tr>
</tbody>
</table>

  - 2 bits for the byte within a block (4 bytes per block addressed by 2 bits).
  - 13 bits for one of the cache sets.
  - 9 bits for the tag.
- A block of memory with tag $t$ and set $s$ can be mapped into any cache line of set $s$.
- Cache memory is associative memory.
• Cache memory stores in each line a block of memory together with its tag.
• Assume master requests a byte from memory.
  Divide address into tag $t$, set $s$ and word $w$.
  – Verify whether any cache line of set $s$ has tag $t$.
  – If yes, send byte $w$ stored in that cache line $l$ to master.
  – Otherwise load block with tag $t$ into one cache line of set $s$, which gets this tag. Send word $w$ to master.
• Assume master requests to store a byte in memory.

Divide address into tag $t$, set $s$ and word $w$.

– Write byte to main memory.
– Compare $t$ with the tags of cache lines of set $s$.
– If $t$ is equal to one of these, store byte $w$ in that cache line.
– Otherwise load corresponding block in main memory into one cache line of set $s$,
set tag to $t$.
(Modification of cache might be omitted).
Picture omitted for copyright reasons
Replacement Algorithm

If we want to load a new block into a cache which is full, we have to remove one block.

- In case of direct mapping there is only one block possible, replace that one.
- For associative and set-associative cache choose from the lines corresponding to that address one according to one of the following algorithms:
  - **Least recently used (LRU)**. Replace block which has been in cache longest without a read or write access.
  - **First in first out (FIFO)**. Replace block which has been in cache longest.
  - **Least frequently used (LFU)**. Replace block which had the least number of read or write accesses.
  - **Random**. Choose a block at random. Performance with random replacement algorithm is not much worse than with the other algorithms.