Predication

Minimization of branches

- "guarded" or conditional instructions
  - the instruction executes, if a given condition is true.
    If the condition is not true the instruction does not execute
- the test is embedded in the instruction itself
  - no branch involved
- predicated instructions
  - predicates are precomputed values stored in registers
  - instruction contains a field to select a predicate from the predicate RF

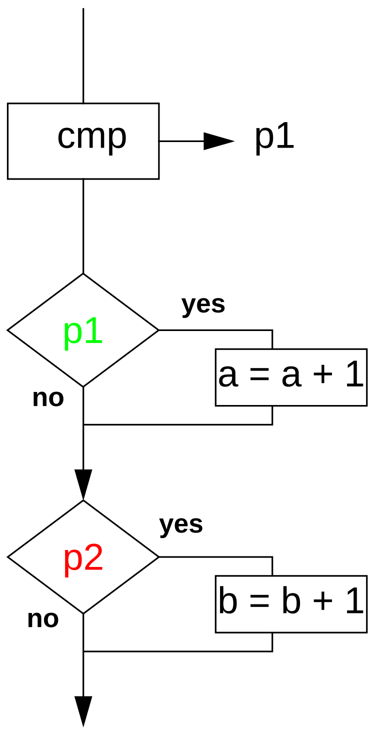
<u>'guarded'</u> or <u>conditional</u> instruction

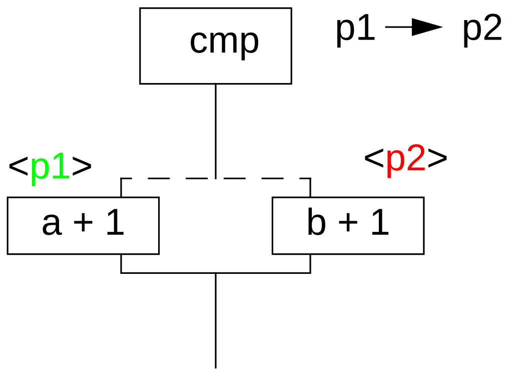
p1 = "true"   p2 = $\overline{\text{p1}}$ = "false"

<p1> a = a + 1   <p2> b = b + 1

$\uparrow$

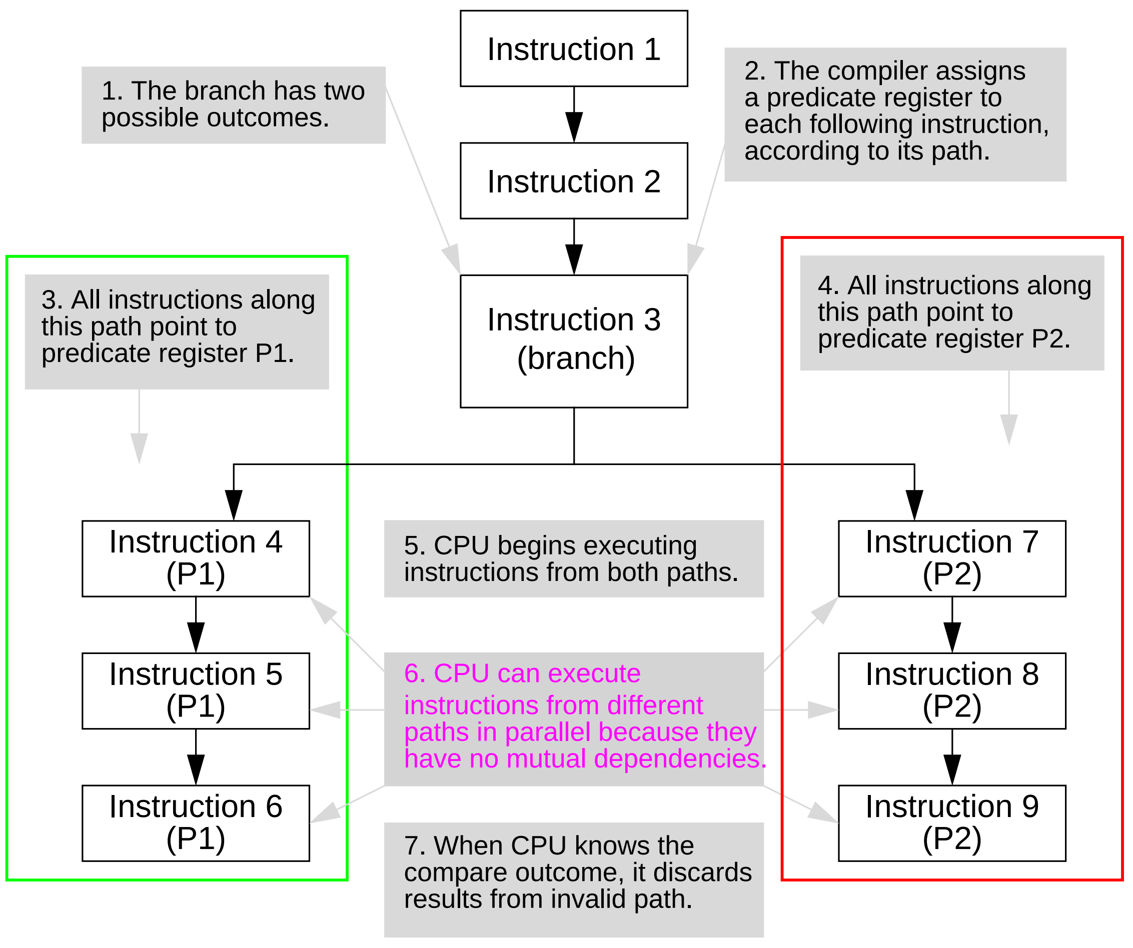
predicate



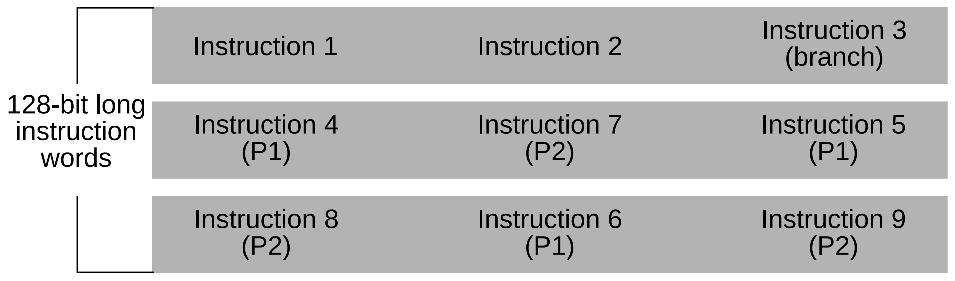


Predication

How Predication works


Instruction 1
Instruction 2
Instruction 3 (branch)
1. The branch has two possible outcomes.
2. The compiler assigns a predicate register to each following instruction, according to its path.
3. All instructions along this path point to predicate register P1.
4. All instructions along this path point to predicate register P2.
Instruction 4 (P1)
Instruction 5 (P1)
Instruction 6 (P1)
Instruction 7 (P2)
Instruction 8 (P2)
Instruction 9 (P2)
5. CPU begins executing instructions from both paths.
6. CPU can execute instructions from different paths in parallel because they have no mutual dependencies.
7. When CPU knows the compare outcome, it discards results from invalid path.

The compiler might rearrange instructions in this order, pairing instructions 4 and 7, 5 and 8, and 6 and 9 for parallel execution.



Predication replaces branch prediction by allowing the CPU to execute all possible branch paths in parallel.

Memory System Architecture

The two basic principles that are used to increase the performance of a processor are:

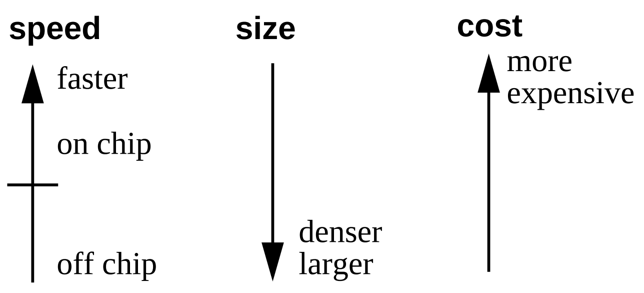
- **pipelining and parallel execution**
- **the optimization of the memory hierarchy.**

Applying a high-performance memory system to modern microprocessors is another step towards improving performance. Whenever off-chip accesses have to be performed, throughput is reduced because of the delay involved and the latency of external storage devices. The memory system typically consists of a hierarchy of memories.

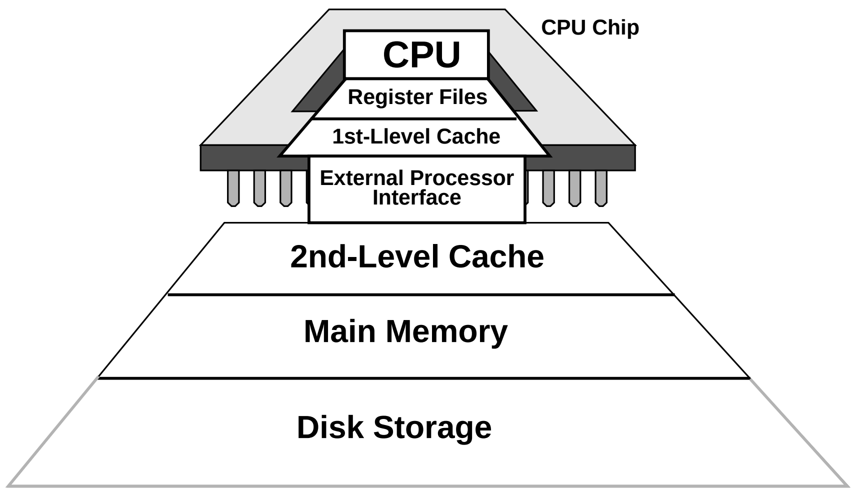
Definition : ***A memory hierarchy is the result of an optimization process with respect to technological and economic constrains. The implementation of the memory hierarchy consists of multiple levels, which are differing in size and speed. It is used for storing the working set of the 'process in execution' as near as possible to the execution unit.***

The memory hierarchy consists of the following levels:

- registers
- primary caches
- local memory
- secondary caches
- main memory



The mechanisms for the data movement between levels may be explicit (for registers, by means of load instructions) or implicit (for caches, by means of memory addressing).



Registers

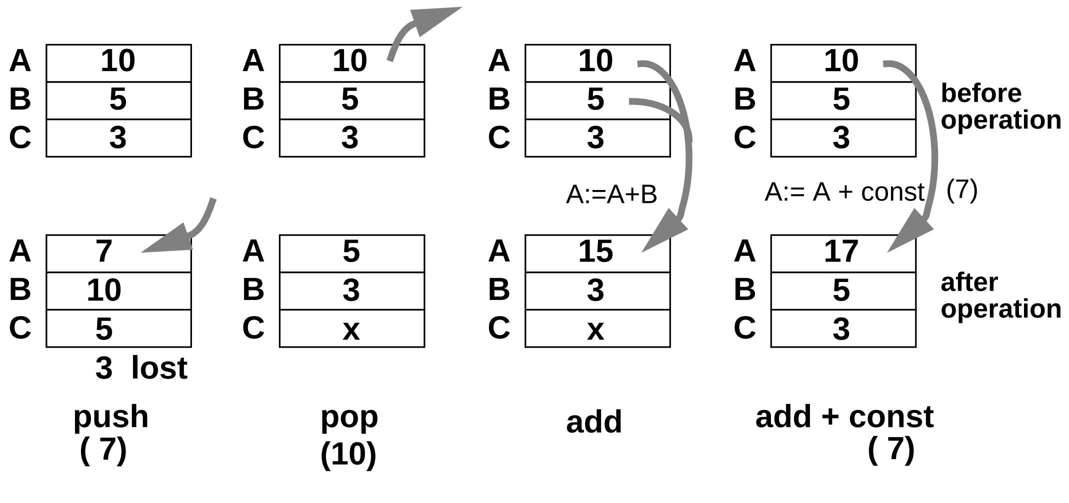
Registers are the fastest storage elements within a processor. Hence, they are used to keep values locally on-chip, and to supply operands to the execution unit(s) and store the results for further processing. The read-write cycle time of registers must be equal to the cycle time of the execution unit and the rest of the processor in order to allow pipelining of these units. Data is moved from and into the registers by explicit software control.

Registers can be grouped in a wide variety of ways:

- evaluation stack
- register file
- multiple register window
- register banks

Evaluation Stack

A very simple register structure is the evaluation stack. The addressing of values is done implicitly by most of the operations. Push and pop operations are used to enter new values on the stack, or to store the top of the stack back to the memory. Dyadic operations, like add, sub, mul, consume the top two registers as source operands, leaving the result on the top of the stack. Implicit addressing of the stack reduces the need for instructions to specify the location of their operands, which in turn reduces the size of instructions and results in a very compact code.



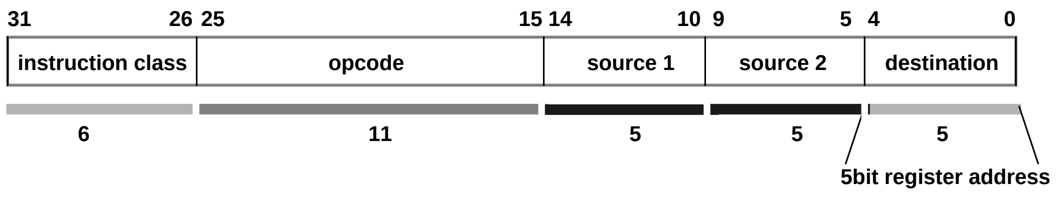
Register File

The collection of multiple registers into a register file, directly accessible from the instruction word, provides a faster access to data, that will have a high ‘hit rate’ within a short period of time. The compiler must identify those variables and allocate them into registers. This task is named register allocation. The reuse of these variables held in registers reduces main memory accesses and therefore speeds up execution.

The register file (RF) is the commonly used grouping of registers in a CPU. The number of registers depends on many constrains, as there are:

- the length of the instruction word containing the addresses of the registers combined in an operation
- the speed of the register file which is inverse proportional to the number of register cells and the number of ports.
- the number of registers needed for software optimization,
  parameter passing for function call and return (32 is more than enough)
- the penalty for saving and restoring the RF in case of a process switch
- the number of ports of the RF for parallel operations

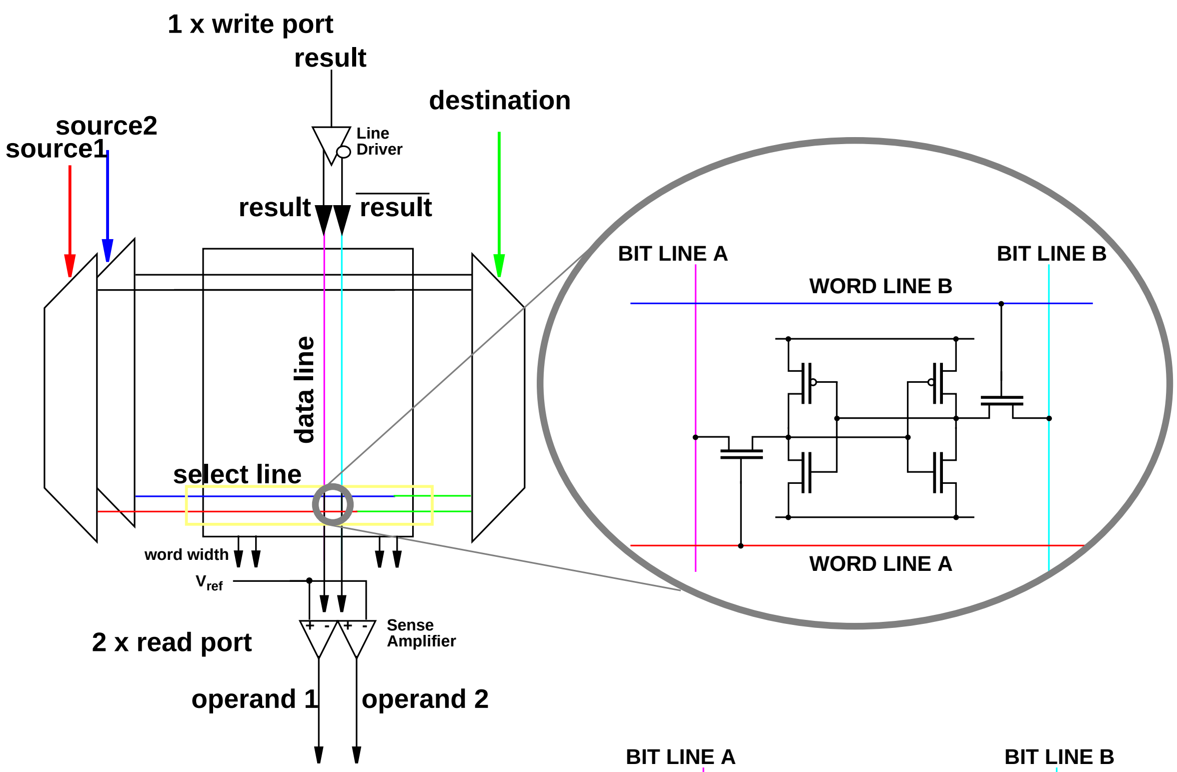
The addressing of the registers in the RF is performed by the register address of an instruction. The number of register address fields in the instruction field divide the processors into classes called two, 2 1/2, or three address machines. Because dyadic operations are very frequent, most modern processors are three address machines and have two source address fields and one destination address field. The following figure presents a typical 32-bit instruction format of a RISC processor.



The limitation of the instruction word width restricts the addressable number of registers to 32 caused by the 5-bit register address fields.

The typical VLSI realization of a RF is a static random access memory with a 6-transistor cell, 4 for the storage cell and two transistors for every read/write access port, which means two simultaneous reads (two read-ports) or one write (one write-port).

Register File



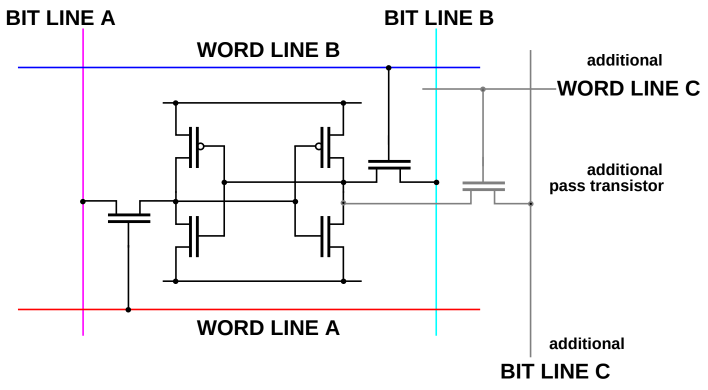


figure b

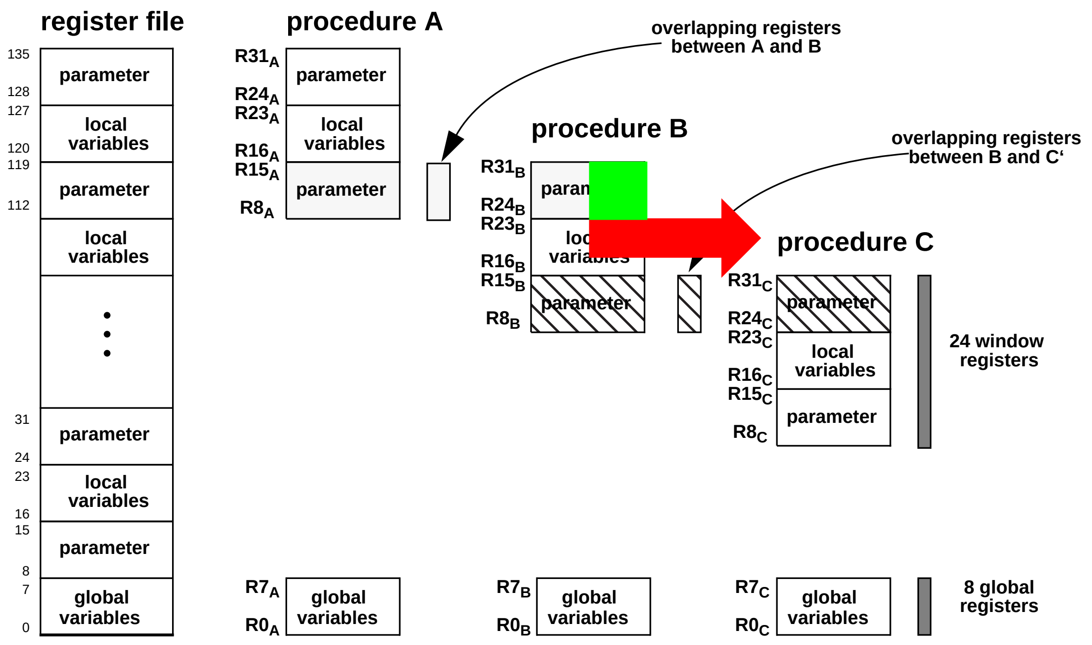
The register file features two read ports and one write port, which shares the two bit lines. The read and the write cycle are time-multiplexed.

The read operation activates the two source address decoders and they drive the correspon-ding word line A and B. Operand 1 and operand 2 can be read on the two bit lines. Special sense amplifiers and charge circuits control the data read-out. The write cycle activates the destination decoder which must drive both word lines. The result data is forced to the storage cell by applying data and negated data on both bit lines to one cell. The geometry of the tran-sistors is chosen so that the flip-flop is overruled by the external bit line information.

An additional read port (3read/1write port) requires one extra word line (from additional source 3 decoder) a pass transistor and an extra bit line as shown in figure b. Every extra port at the register file increases the area consumed by the RF and thus reduces the speed of the RF.

Register Windows

The desire to optimize the frequent function calls and returns led to the structure of overlapping register windows. The save and restore of local variables to and from the procedure stack is avoided, if the calling procedure can get an empty set of registers at the procedure entry point. The passing of parameter to the procedure can be performed by overlapping of the actual with the new window. For global variables a number of global registers can be reserved and accessed from all procedure levels. This structure optimize the procedure call mechanism of high level languages, and optimizing compiler can allocate many variables and parameters directly into registers.



Due to the fixed size of the instruction word the register select fields have the same width (5 bits) as in the register file structure. The large RF requires more address bits and therefore the addressing is done relatively to the window pointer. The window pointer is kept in a special register called current window pointer cwp and provides the base address of the actual window. An addition of the cwp with the content of the register select field provides the physical address to the large register file. The following figure presents the logical structure of the addressing scheme. The addition and the global register multiplexer can be incorporated in the register file decoder by using the addressing truth table as the basis for the decoder implementation. Nevertheless this address translation slows down the access of the RF.

The cwp is controlled directly by the instruction ‘call’, which decrements the pointer and by the ‘return’, which increments the cwp.

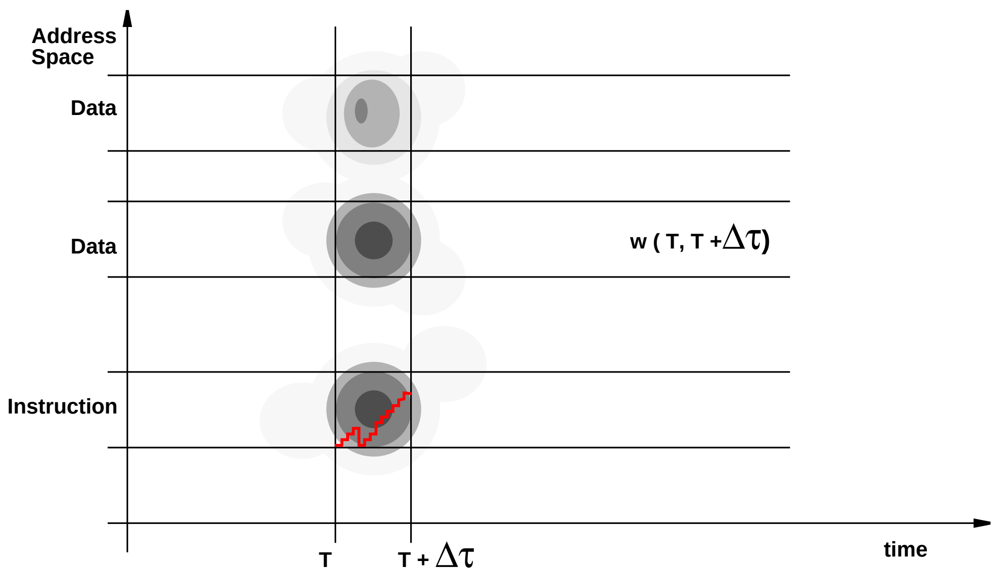
Caches

Caches are the next level of the memory hierarchy. They are small high speed memories employed to hold small blocks of main memory that are currently in use.

The principle of locality, which justifies the use of cache memories, has two aspects:

- locality in space or spatial locality
- locality in time or temporal locality

Most programs exhibit this locality in space in cases where subsequent instructions or data objects are referenced from near the current reference. Programs also exhibit locality in time where objects located in a small region will be referenced again within a short periode. Instructions are stored in sequence, and data objects normally being stored in the order of their use. The following figure is an idealized space/time diagram of address references, representing the actual working set w in the time interval $\Delta\tau$.



Caches are transparent to the software. Usually, no explicit control of cache entries is possible. Data is allocated automatically by cache control in the cache, when a load instruction references the main memory.

Some processors feature explicit control over the caching of data. Four types of user mode instructions can improve hit rate significantly (cache bypassing on stores, cache preloading, forced dirty-line flush, line allocation without line fill).

Caches

Cache memory design aims to make the slow, large main memory appear to the processor as a fast memory by optimizing the following aspects:

- maximizing the hit ratio
- minimizing the access time
- minimizing the miss penalty
- minimizing the overhead for maintaining cache consistency

The performance gain of a cache can be described by the following formula:

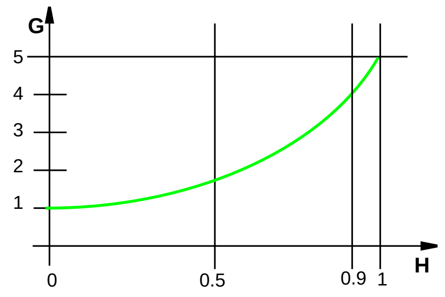
$$G_{cache} = \frac{T_m}{\underbrace{(1-H)}_{\text{miss ratio}} * T_m + \underset{\text{hit ratio}}{\underset{\uparrow}{H}} * T_c} = \frac{1}{(1-H) + H * \frac{T_c}{T_m}}$$

$$= \frac{1}{1 - H * \left(1 - \frac{T_c}{T_m}\right)}$$

T_m = t_{acc} of main memory

T_c = t_{acc} of cache memory

H = hit ratio [0, ...1]



example for $\frac{T_m}{T_c} = \frac{5}{1}$

The hit ratio of the cache (in %) is the ratio of accesses that find a valid entry (hit) to accesses that failed to find a valid entry (miss). The miss ratio is 100% minus hit ratio.

The access time to a cache should be significantly shorter than that to the main memory. On-chip caches (L1) normally need one clock tick to fetch an entry.

Access to off-chip caches (L3) is dependent on the chip-to-chip delay, the control signal protocol, and the access time of the external memory chips used.

Cache Terms

Cache block size:

Cache block size is the size of the addressable unit within a cache, which is a the same time the size of the data chunk to be fetched from main memory.

Cache line:

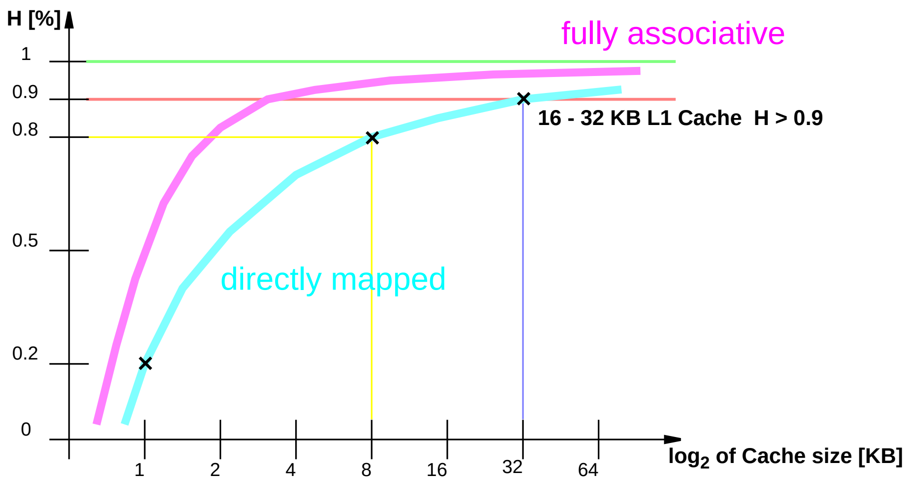
A cache line is the data of a cache block, typically a power of 2 number of words. The term is mainly used for the unit of transport between main memory and cache.

Cache entry:

A cache entry is the cache line together with all required management information, e.g. the tag field and the control field.

Cache frame:

The cache frame refers to the addressable unit within a cache which can be populated with data. It defines the empty cache memory space, where a cache line can be placed.



hit ratio versus cache size

Set:

A set is the number of cache blocks which can be reached by the same index value.

A set is only these pairs (2-way) or quadruples (4-way) of cache blocks, not the whole part of one way of the set-associative cache memory.

Bedauerlicherweise gibt es keinen Begriff für den jeweiligen Teil des set-associative cache Speichers. Somit wird ’set’ auch häufig als die Bezeichnung für diesen Teil des Speichers verwendet.

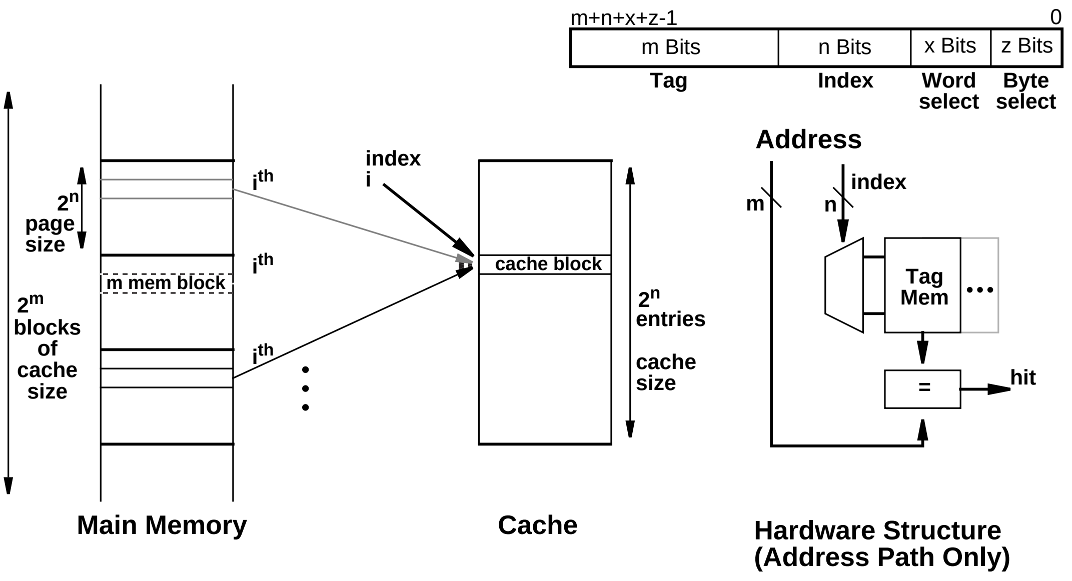
Cache Organizations

Five important features of the cache, with their various possible implementations are:

- mapping	direct, set associative, fully associative
- organization	cache block size, entry format, split cache, unified cache
- addressing	logically indexed, physically indexed, logically indexed/physically tagged
- management	consistency protocol, control bits, cache update policy
- placement	random, filo, least recently used

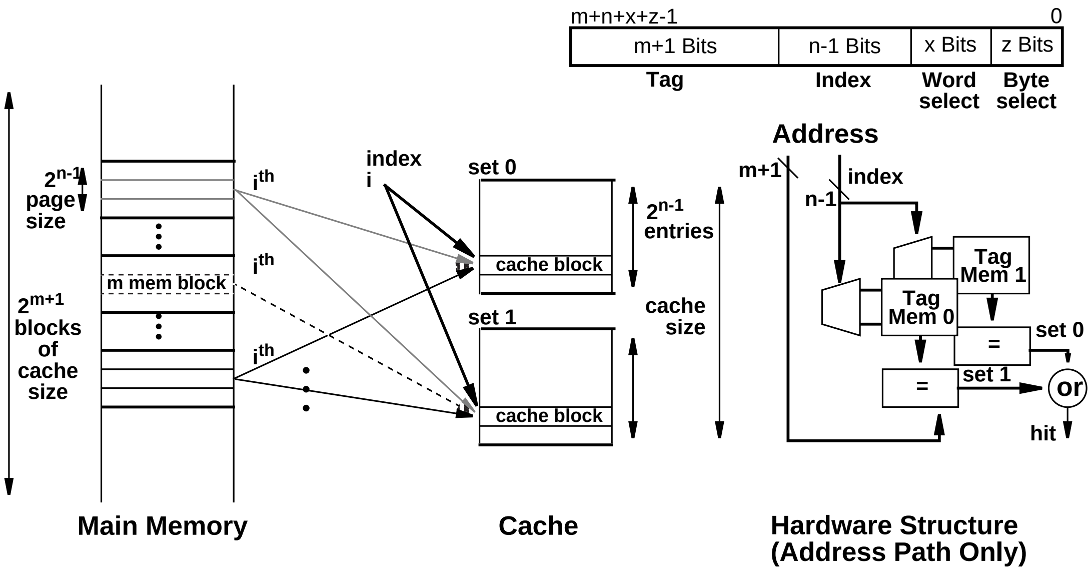
One of the most important features is the mapping principle. Three different strategies can be distinguished, but the range of caches - from directly mapped to set associative to fully associative - can be viewed as a continuum of levels of set associativity.

Cache Mapping

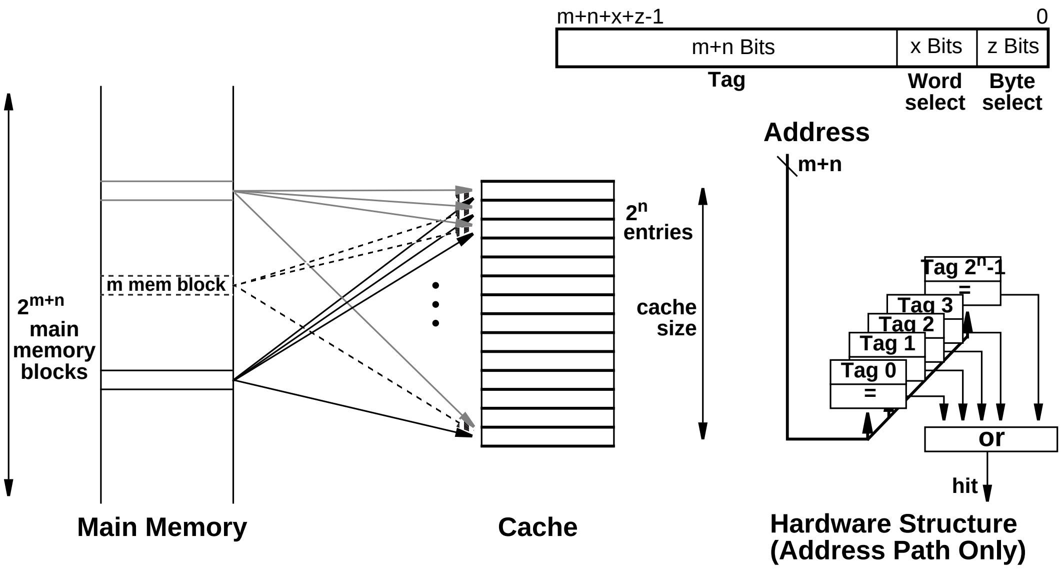


Directly Mapped Cache

Cache Mapping

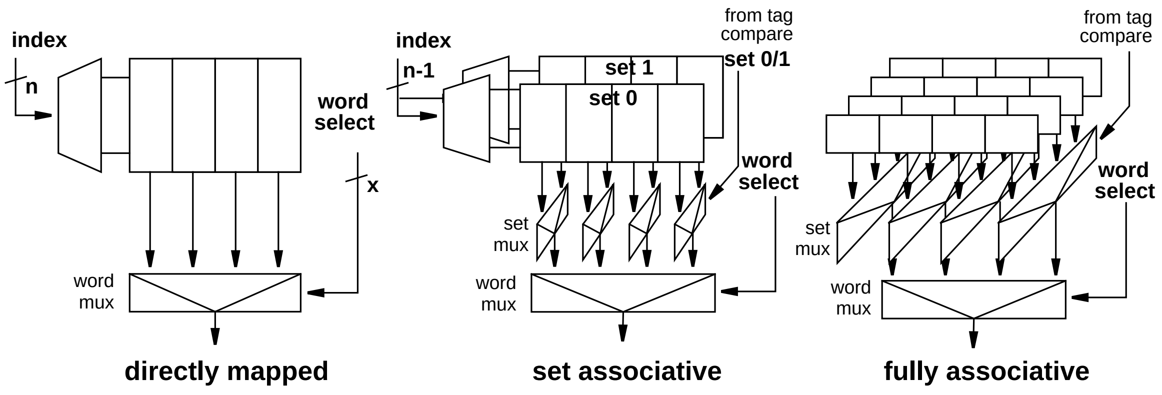


2-way Set Associative Cache



Fully Associative Cache

Cache Mapping



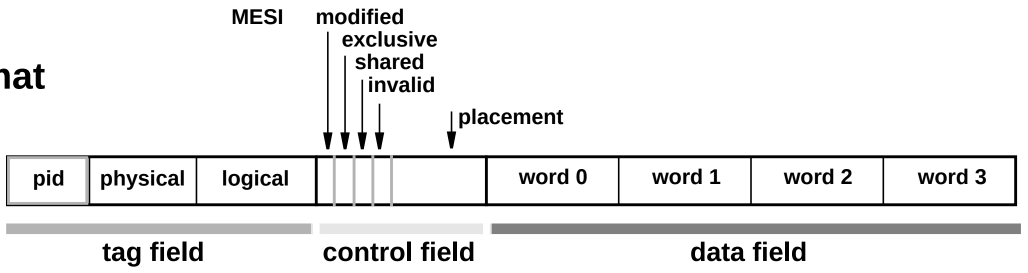
Data Paths of Differently Mapped Caches

Cache Organization

The basic elements of a cache organization are:

- the entry format
- the cache block size
- the kind of objects stored in cache
- special control bits

Entry format



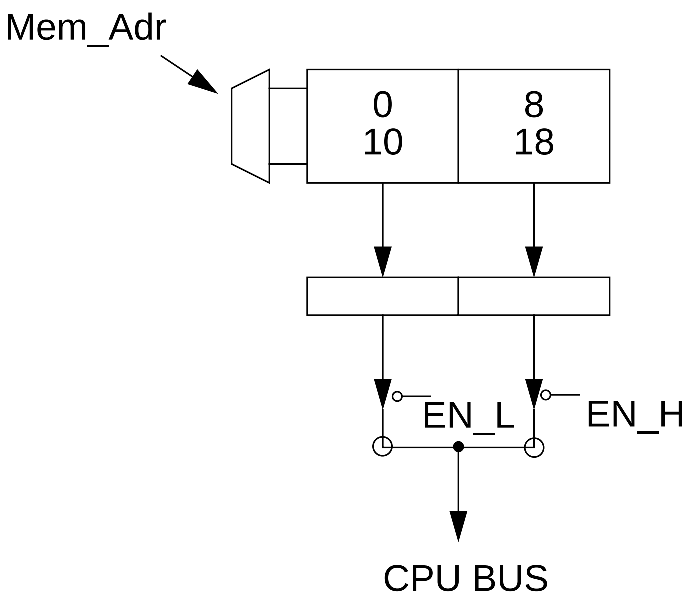
Cache Line Fetch Order

Always fetch required word first.

access sequence INTEL mode

0. Adr.	1. Adr.	2. Adr.	3. Adr.	
0	8	10	18	mod ++
8	0	18	10	mod - -
10	18	0	8	mod ++
18	10	8	0	mod - -

byte offsets



1. read of Mem

0,8
8,0

see also: DRAM Burst mode for further explanations

Cache Consistency

The use of caches in shared-memory multiprocessor systems gives rise to the problem of **cache consistency**. Inconsistent states may occur, when two processors keep a copy of the same memory cell in their caches, and one processor modifies the cache contents or the main memory by a write.

Two memory-update strategies can be distinguished:

- the write back (WB), sometimes also known as copy back,
- and the write through (WT).

The WT strategy is the simplest one. Whenever a processor starts a write cycle, the cache is updated and the same value is written to the main memory. The cache is said to be written-through. Nevertheless, this write must inform all other caches of the new value at this address. While the active bus master (CPU or DMA) is placing its write address on to the address bus, all other caches in the other CPUs must check this address against their cache entries so as to invalidate or update the cache line.

The WB strategy is more efficient, because the main memory is not updated for each store instruction. The modified data is stored in the cache data field only, the line being marked as modified in the cache control field. The write to the main memory is performed only on request, and then whole cache lines are written back (WB). This memory update strategy is called write back or copy back and allows the cache to hold newer values than the main memory. Information must be available in the cache line, which keeps track of the state of a cache entry. The MESI states and MESI consistency protocol are widely used and are therefore given here as an example of cache consistency protocols. Four possible states of a cache line are used by the MESI protocol:

- **M**odified: one or more data items of the cache line are written by a store operation, the modified or dirty bit being set
- **E**xclusive unmodified: the cache line belongs to this CPU only, the contents is not modified
- **S**hared unmodified: the cache line is stored in more than one cache and can be read by all CPUs. A store to this address must invalidate all other copies and update the main memory
- **I**nvalid: the cache entry is invalid; this is the initial state of a cache line that does not contain any valid data.

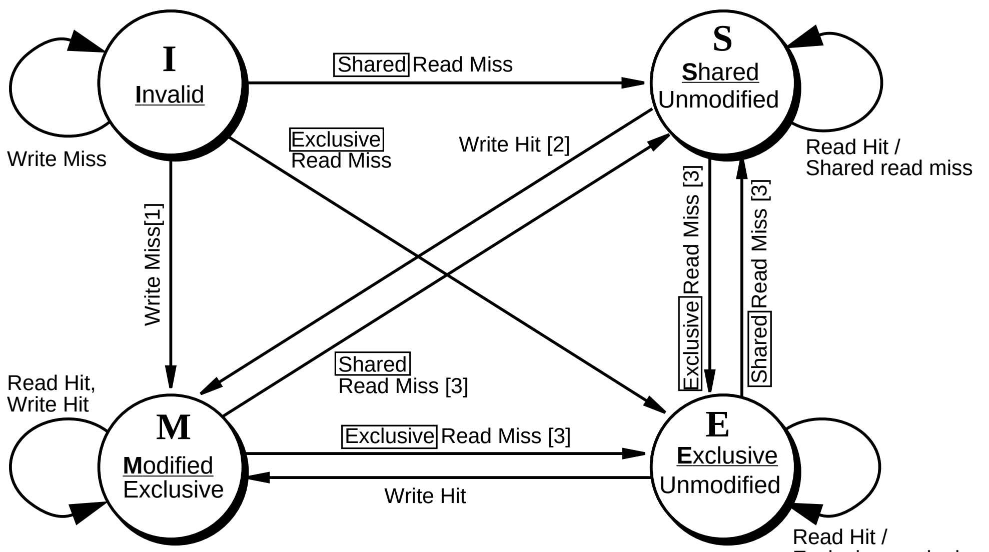
Cache Consistency

The states of the cache consistency control bits and the transitions between them are illustrated in two figures, the first one showing all transitions of the cache of a bus master CPU, and the second state diagram showing the transitions of a slave CPU cache (snooping mode).

Definition : ***A processor of a shared-memory multiprocessor system (bus-interconnected) is called bus master if it has gained bus mastership from the arbitration logic and is in the process of performing active bus transactions.***

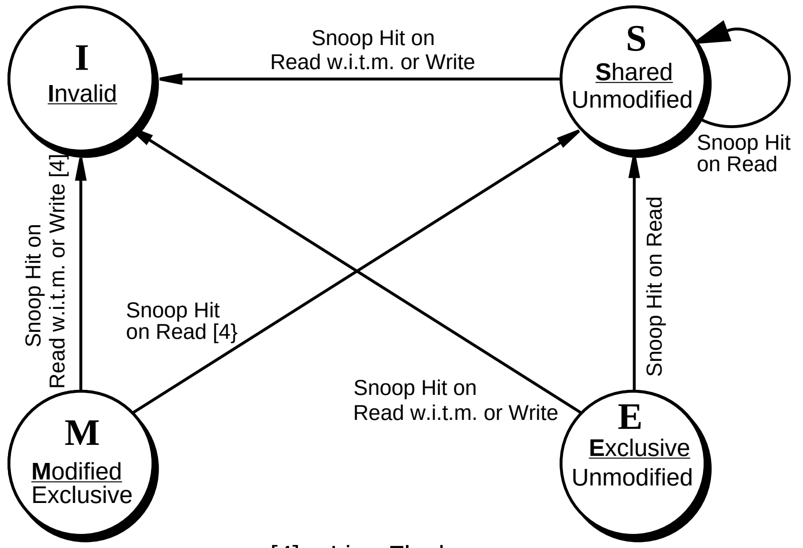
Processors of a shared-memory multiprocessor system are called bus slaves if these processors can not currently access the bus for active bus transactions. They have to listen passively (snooping) for the active transactions of the bus master.

Cache Consistency State Transitions for Bus Master CPU



[1] = Read with intent to modify
[2] = Invalidation Bus Transaction
[3] = Address tag miss
▭ = Snoop response

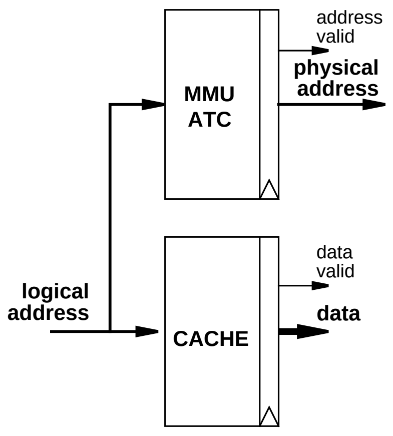
State Transitions for Snooping CPU (Slave)



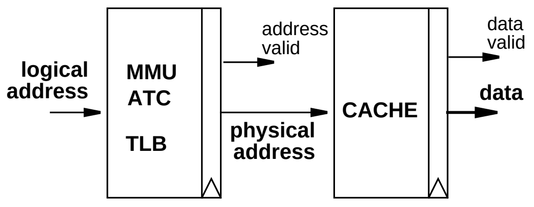
[4] = Line Flush

Cache Addressing Modes

The **logically addressed** cache is indexed by the logical (or virtual) address of the process currently running on the CPU. The address translation need not be performed for the cache access, and the cache can therefore be accessed very fast, normally within one clock cycle. Only the valid bit must be checked to see whether or not the data item is in the cache line. The difficulty with the logically addressed cache is that no snooping of external physical addresses is possible.

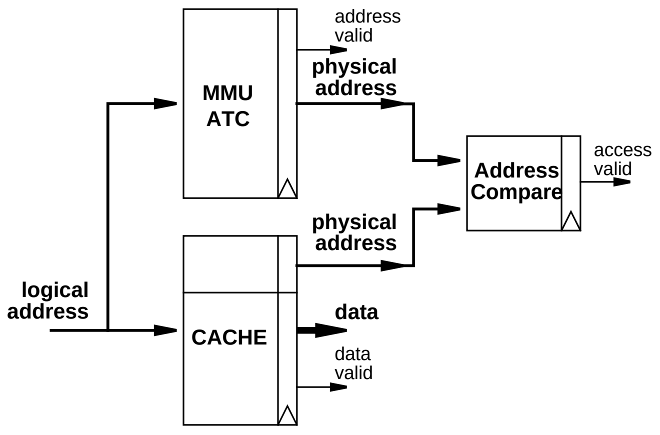


logically addressed



physically addressed

The **physically addressed** cache is able to snoop and need not be invalidated on a process switch, because it represents a storage with a one-to-one mapping of addresses to main memory cells. The address translation from logical to physical address must be performed in advance. This normally slows down the cache access to two clock cycles, the address translation and the cache access.

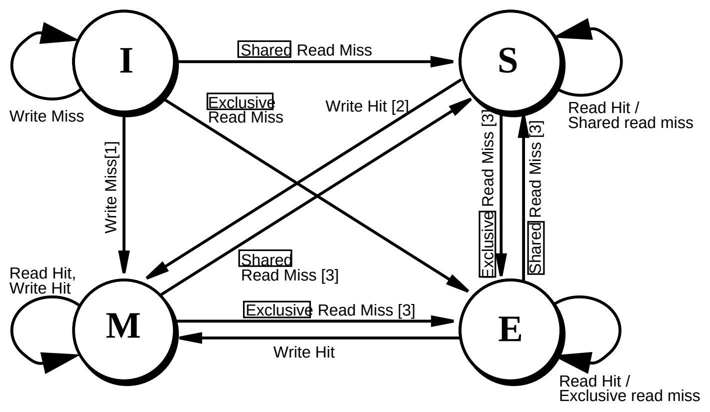


logically addressed/physically tagged

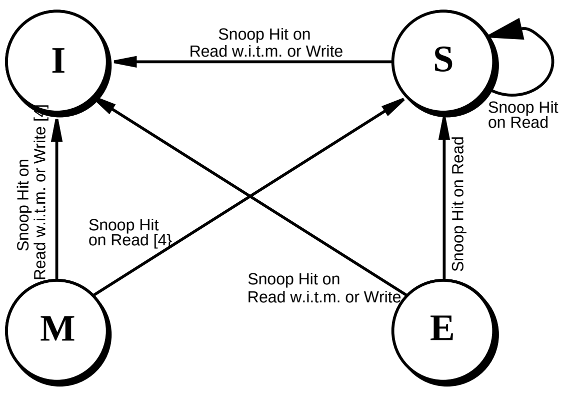
The **logically indexed/physically tagged** cache scheme avoids both disadvantages by storing the high-order part of the physical address in the cache line as additional tag information. If the size of the index part of the logical address is chosen to match the page size, then the indexing can be performed without MMU translation.

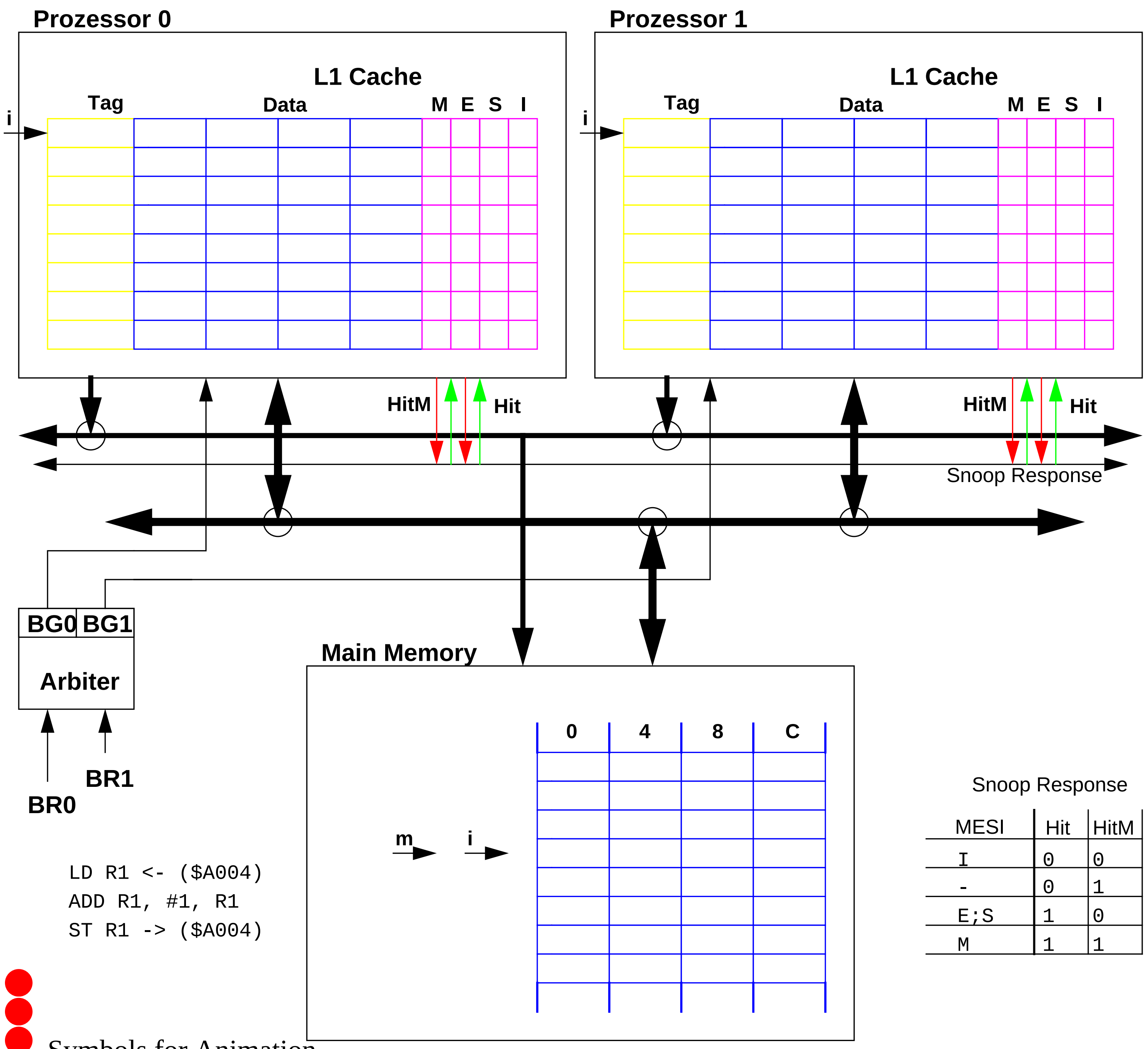
Cache Consistency

Cache Consistency State Transitions for Bus Master CPU



Cache Consistency State Transitions for BusSlave CPU (Snooping)


I
S
M
E
Snoop Hit on Read w.i.t.m. or Write
Snoop Hit on Read
Snoop Hit on Read w.i.t.m. or Write [4]
Snoop Hit on Read [4]
Snoop Hit on Read w.i.t.m. or Write
Snoop Hit on Read



```
LD R1 <- ($A004)
ADD R1, #1, R1
ST R1 -> ($A004)
```

Snoop Response

MESI	Hit	HitM
I	0	0
-	0	1
E;S	1	0
M	1	1

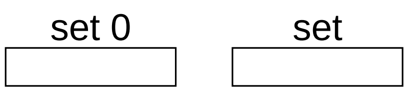
Symbols for Animation

Cache Placement

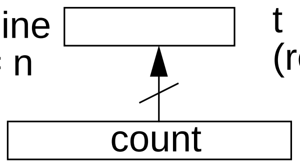
directly mapped cache

$\longrightarrow$ single entry
no choice

set/fully associative



- random replacement ——— random FF
- fifo replacement — first in - first out
- circular fifo — pointer/per index
  nod (number of sets) — per cache line
  set count = n



t
(read cache)

- least recently used LRU

am längsten zurückliegender Zugriff
eines Eintrags ... der Eintrag wird überschrieben

... Algorithmus