Memory System Organization and Architecture

- Understanding the performance capabilities of a modern processor is the memory hierarchy.
- By using a hierarchy of memories, each with different access speeds and storage capacities, a computer system can exhibit performance above what would be possible without a combination of the various types. The base types that normally constitute the hierarchical memory system include registers, cache, main memory, and secondary memory.

- Terminology is used with memory hierarchy:

 Hit—The requested data resides in a given level of memory (typically, we are concerned with the hit rate only for upper levels of memory).

 Miss—The requested data is not found in the given level of memory.

 Hit rate—The percentage of memory accesses found in a given level of memory.

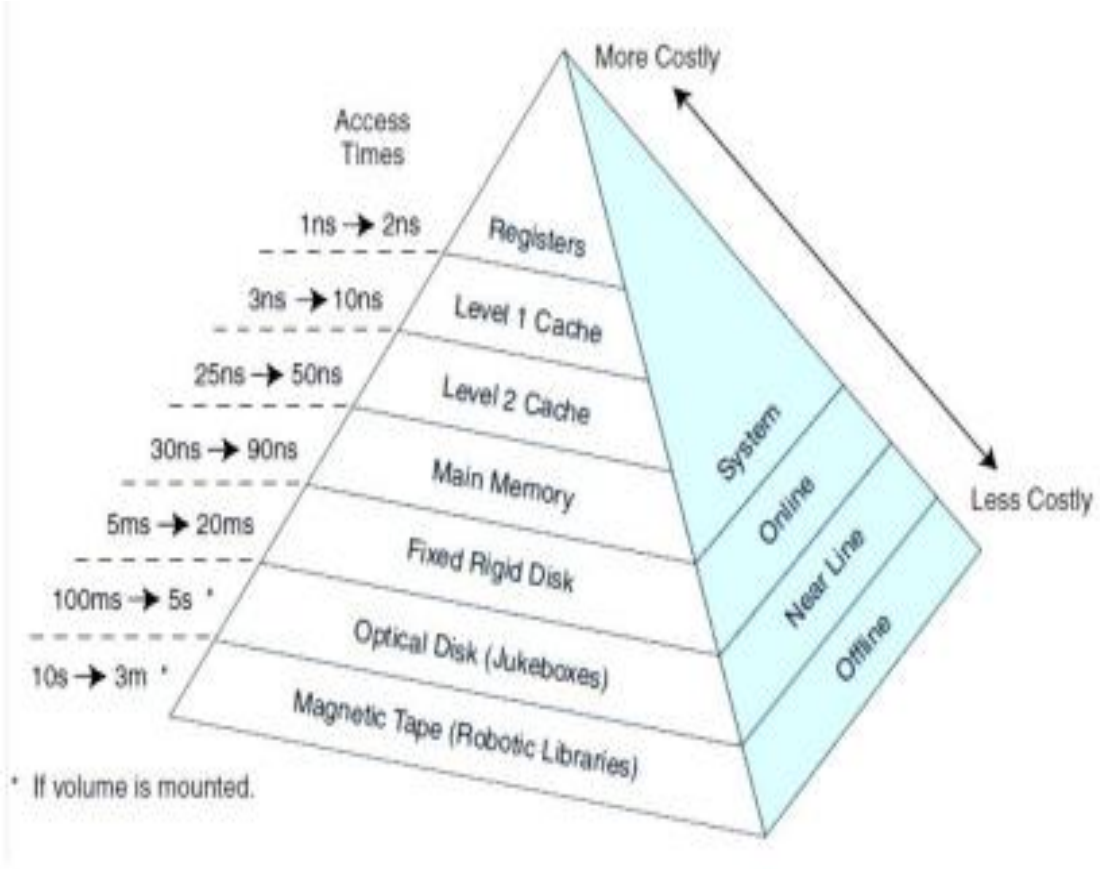
 Miss rate—The percentage of memory accesses not found in a given level of memory. Note: Miss Rate = 1 _ Hit Rate.

 Hit time—The time required to access the requested information in a given level of memory.

 Miss penalty—The time required to process a miss, which includes replacing a block in an upper level of memory, plus the additional time to deliver the requested data to the processor. (The time to process a miss is typically significantly larger than the time to process a hit.)

In the memory hierarchy, memories closer to the top tend to be smaller in size.

- For any given data, the processor sends its request to the fastest, smallest partition of memory (typically cache, because registers tend to be more special purpose). If the data is found in cache, it can be loaded quickly into the CPU. If it is not resident in cache, the request is forwarded to the next lower level of the hierarchy, and this search process begins again. If the data is found at this level, the whole block in which the data resides is transferred into cache. If the data is not found at this level, the request is forwarded to the next lower level, and so on.



Memory hierarchy

Hit and Miss

Occurrence of Hit— in a level when the address that an operation references is found in that level of the memory hierarchy

• Otherwise, a Miss

Hit and Miss Rates

• Hit rate of a level— the percentage of references that reach the level that result in hits
• Miss rate of a level— the percentage of references that reach the level that result in misses = 100%

Neither the hit rate nor the miss rate count the references that are handled by higher levels in the hierarchy
For example, requests that hit in the cache of our example memory hierarchy do not count in the hit rate or miss rate of the main memory

Computing the Average Access Time

• First know the hit rate and access time (time to complete a request that hits) for each level in the memory hierarchy
• Then compute the average access time of the memory hierarchy

Average access time T_{av} at a level

• $\mathbf{T_{av} = (T_{hit} \cdot P_{hit}) + (T_{miss} \cdot P_{miss})}$
• $\mathbf{T_{hit}}$ = The time taken to resolve requests that hit in the level,
• $\mathbf{P_{hit}}$ = The hit rate of the level (expressed as a probability)
• $\mathbf{T_{miss}}$ = The average access time of the levels below this one in the hierarchy, and
• $\mathbf{P_{miss}}$ = The miss rate of the level

Hit rate of the lowest Level

• Hit rate of the lowest level in the hierarchy = 100 percent (all requests that reach the bottom level are handled by the bottom level)
• Start at the bottom level and work up to compute the average access time of each level in the hierarchy

Example 1

• Assume— hit rate of 75 percent at a level of the memory hierarchy
• Assume— memory requests take 12 ns to complete if they hit in the level
• Assume— memory requests that miss in the level take 100 ns to complete

Find the Average access time of the level?
Solution
Using the formula, the average access time

$$T_{av} = (T_{hit} \cdot P_{hit}) + (T_{miss} \cdot P_{miss})$$

$$(12\ ns \cdot 0.75) + (100\ ns \cdot 0.25) = 34\ ns$$

Example 2

• Assume— a memory system contains the cache, main memory, and virtual memory
• Assume— the access time of the cache = 5 ns
• Cache hit rate = 80 percent
• The access time of the main memory = 100 ns
• Main memory hit rate = 99.5 percent
• The access time of the virtual memory = 10 milliseconds (ms)

Solution for the average access time of the hierarchy

• Start at the bottom of the hierarchy and work up
• Hit rate of the virtual memory =100 percent
• The average access time for requests that reach the main memory =

$$(100\ ns \cdot 0.995) + (10\ ms \cdot 0.005) = 50$$

Given this, the average access time for requests that reach the cache (which is all requests) =

$$(5\ ns \cdot 0.80) + (50{,}099.5\ ns \cdot 0.20) = 10{,}024ns$$

INSTRUCTION PIPELINING

The enhancements of processor organizational can improve performance. We have already seen some examples of this, such as the use of multiple registers rather than a single accumulator, and the use of a cache memory. Another organizational approach, which is quite common, is instruction pipelining.

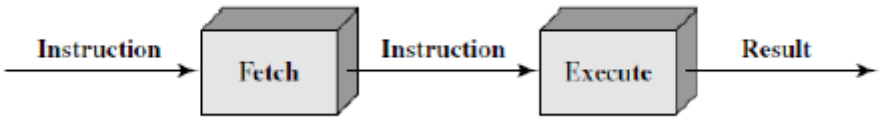
Pipelining Strategy

Instruction pipelining is similar to the use of an assembly line in a manufacturing

Plant To apply this concept to instruction execution; we must recognize that, in fact,

an instruction has a number of stages

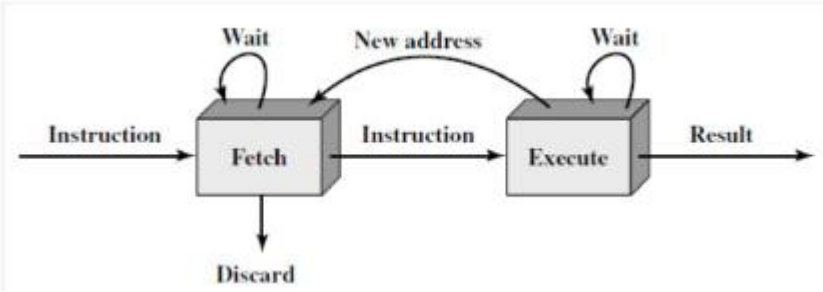
As a simple approach, consider subdividing instruction processing into two stages: fetch instruction and execute instruction. There are times during the execution of an instruction when main memory is not being accessed. This time could be used to fetch the next instruction in parallel with the execution of the current one. Figure below depicts this approach.



The pipeline has two independent stages. The first stage fetches an instruction and buffers it. When the second stage is free, the first stage passes it the buffered instruction. While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This is called instruction prefetch or fetches overlap.

Note that this approach, which involves instruction buffering, requires more registers. In general, pipelining requires registers to store data between stages. It should be clear that this process will speed up instruction execution. If the fetch and execute stages were of equal duration, the instruction cycle time would be halved.

However, if we look more closely at this pipeline (Figure below), we will see that this doubling of execution rate is unlikely for two reasons



1. The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.

2. A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

Guessing can reduce the time loss from the second reason. A simple rule is the following:

When a conditional branch instruction is passed on from the fetch to the execute stage, the fetch stage fetches the next instruction in memory after the branch instruction. Then, if the branch is not taken, no time is lost. If

the branch is taken, the fetched instruction must be discarded and a new instruction fetched.

To gain further speedup, the pipeline must have more stages. Let us consider the following decomposition of the instruction processing.

• **Fetch instruction (FI)**: Read the next expected instruction into a buffer.

• **Decode instruction (DI)**: Determine the opcode and the operand specifiers.

• **Calculate operands (CO)**: Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.

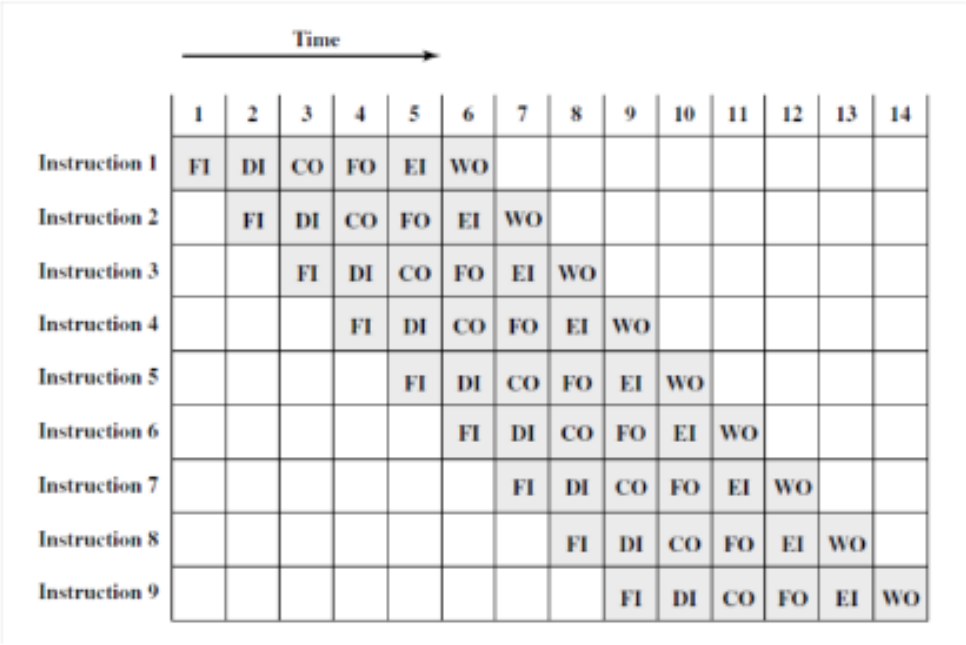
• **Fetch operands (FO)**: Fetch each operand from memory. Operands in registers need not be fetched.

• **Execute instruction (EI)**: Perform the indicated operation and store the result, if any, in the specified destination operand location.

• **Write operand (WO)**: Store the result in memory.

With this decomposition, the various stages will be of more nearly equal duration.

Figure below shows that a six-stage (equal duration) pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.



The diagram assumes that each instruction goes through all six stages of the pipeline. This will not always be the case. For example, a load instruction does not need the WO stage. However, to simplify the pipeline hardware, the timing is set up assuming that each instruction requires all six stages. Also, the diagram assumes that all of the stages can be performed in parallel.

In particular, it is assumed that there are no memory conflicts. Several other factors serve to limit the performance enhancement. If the six stages are not of equal duration, there will be some waiting involved at various pipeline stages, as discussed before for the two-stage pipeline. Another difficulty is the conditional branch instruction, which can invalidate several instruction fetches. A similar unpredictable event is an interrupt.

Other problems arise that did not appear in our simple two-stage organization.

The CO stage may depend on the contents of a register that could be altered by a previous instruction that is still in the pipeline. Other such register and memory conflicts could occur. The system must contain logic to account for this type of conflict

Pipeline Performance

The cycle time of an instruction pipeline is the time needed to advance a set of instructions one stage through the pipeline;

The cycle time can be determined as

$$T = \max[T_i] + d = T_m + d \qquad 1 \leq i \leq k$$

Where

T_i = Time delay of the circuity in the *i*th stage of the pipeline

T_m = Maximum stage delay (delay through stage which experiences the largest delay)

k = Number of stages in the instruction pipeline

d = Time delay of a latch, needed to advance signals and data from one stage to the next.

In general, the time delay d is equivalent to a clock pulse and Now suppose that n instructions are processed, with no branches. Let Tk,n be the total time required for a pipeline with k stages to execute n instructions. Then:

$$T_{k,n} = [k + (n-1)]_T$$

A total of k cycles are required to complete the execution of the first instruction, and the remaining instructions require cycles. The ninth instruction completes at time cycle 14:

$$14 = [6 + (9 - 1)]$$

Now consider a processor with equivalent functions but no pipeline, and assume that the instruction cycle time is the speedup factor for the instruction pipeline compared to execution without the pipeline is defined as:

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk_T}{[k + (n-1)]_T} = \frac{nk}{k + (n-1)}$$