Advanced Computer Architecture

Performance

Clock “ticks” refer to clock edges (rising or falling)

Clock Cycles

Cycle time (period) = time between ticks = seconds per cycle
Clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)

A 2GHz clock has a cycle time of

$$\frac{1}{2000 \times 10^6 \text{ cycles/sec.}} \times \frac{10^9 \text{ nsec}}{\text{sec.}} = 0.5 \text{ nsec.}$$

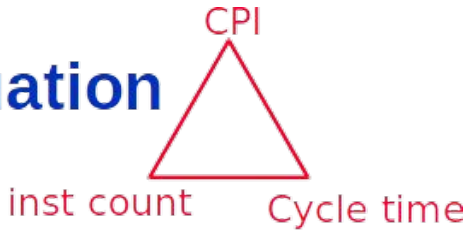
CPI

Cycles per instruction (CPI) – smaller is better

Instruction per cycle (IPC) bigger is better

If the cycles to execute one instruction vary depending on the instruction, then the average CPI or IPC of a program will depend on how many of each type of instruction is executed.

5) Processor performance equation



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

3) Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

4) Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$

Exercise

What is the overall speedup if you make 10% of a program 90 times faster?

What is the overall speedup if you make 90% of a program 10 times faster?

Problems

Q.1 Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Q.2 A common transformation required in graphics processors is square root. Implementations of floatingpoint (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

Amdahl's Law example

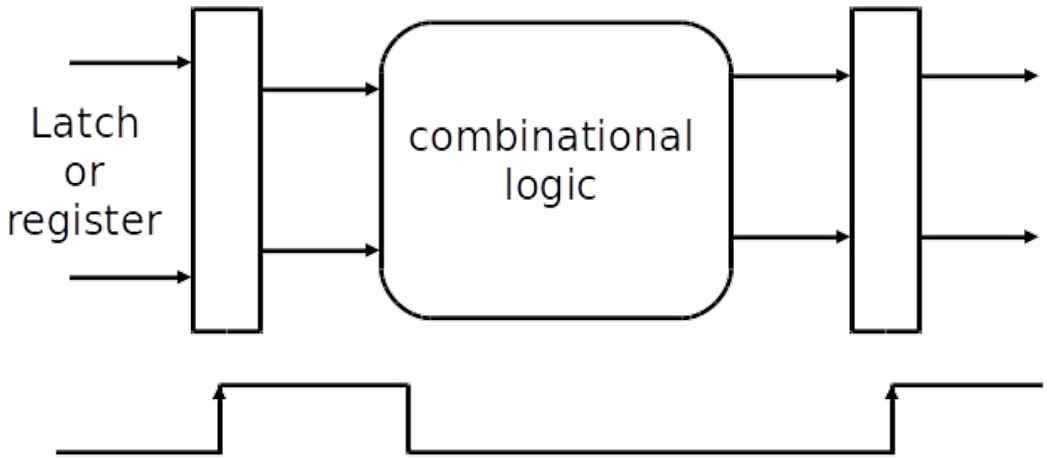
- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

What's a Clock Cycle?



- **Old days: 10 levels of gates**
- **Today: determined by numerous time-of-flight issues + gate delays**
  - **clock propagation, wire lengths, drivers**

Misleading Performance Measurement

Clock rate: 500MHZ

	Instruction class		
	A	B	C
CPI	1	2	3

Code from	Instruction counts (in billions) for each instruction class		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

-Execution time=$\Sigma(CPI_i * I_i)$/clock-rate

Execution time$_1$= $\{(1*5+2*1+3*1)*10^9\}/(500*10^6)=20s$

Execution time$_2$= $\{(1*10+2*1+3*1)*10^9\}/(500*10^6)=30s$

-MIPS=instruction count/(execution time$*10^6$)

$MIPS_1$= $\{(5+1+1)*10^9\}/(20*10^6)=350$

$MIPS_2$= $\{(10+1+1)*10^9\}/(30*10^6)=400$

Hardware-Oriented Metrics

Clock rate and IPC are often combined into various figures of merit:

- **MIPS** (Millions of Instructions Per Second) – pronounced “mips”
- **MOPS** (Millions of Operations Per Second) – pronounced “mops”
- **MFLOPS (**Millions of Floating-point Operations Per Second) – pronounced “megaflops” and sometimes written “megaFLOPS”

Replace first letter with **K** (kilo), **G** (giga), **T** (tera), **P** (peta), etc., as appropriate.

(or even **E** (exa), **Z** (zeta) ..)

Hardware-Oriented Metrics

Clock rate and IPC are often combined into various figures of merit:

- **MIPS** (Millions of Instructions Per Second) – pronounced “mips”
- **MOPS** (Millions of Operations Per Second) – pronounced “mops”
- **MFLOPS (**Millions of Floating-point Operations Per Second) – pronounced “megaflops” and sometimes written “megaFLOPS”

Replace first letter with **K** (kilo), **G** (giga), **T** (tera), **P** (peta), etc., as appropriate.

(or even **E** (exa), **Z** (zeta) ..)

Problems with Hardware-Oriented Metrics

- Processors with different ISAs may require a different number of instructions to perform the same task, so MIPS hard to compare

  - MOPS and MFLOPS are a somewhat better measure

  - How do you count floating-point divides?

- Vendors usually report “peak” rates

MIPS Calculation

One alternative to time as the metric is MIPS, or million instructions per second. For a given program, MIPS is simply

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} * 10^6} = \frac{\text{Clock rate}}{\text{CPI} * 10^6}$$

Limitations of MIPS

-Meaningful only for comparing machines with same ISA, same program, and same input

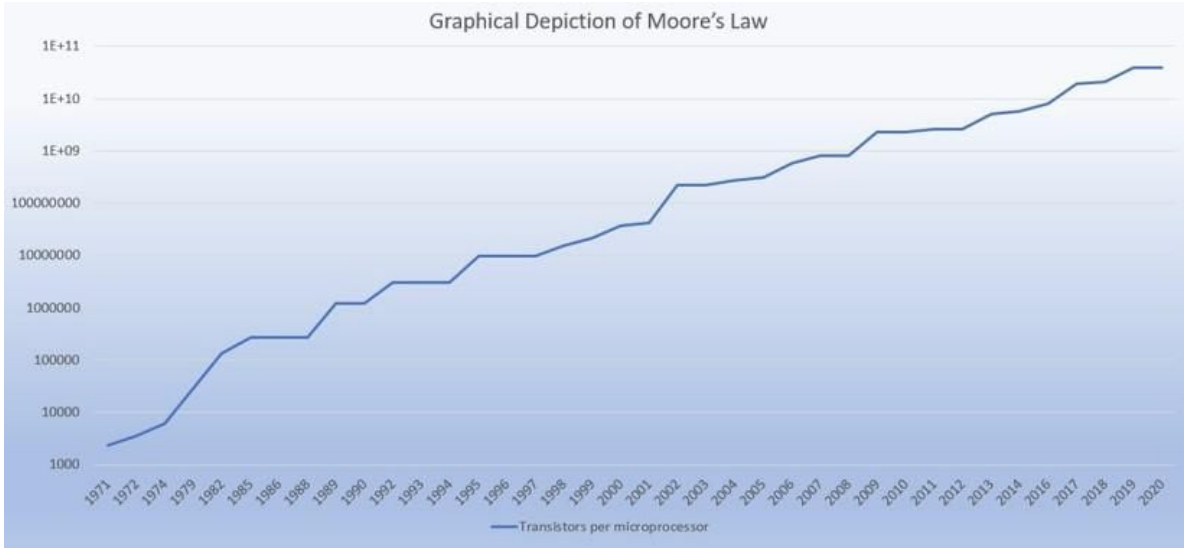
- Instruction capability not considered

-May vary inversely with performance!

- Instruction count is an absolute number without considering the frequency of each instruction class

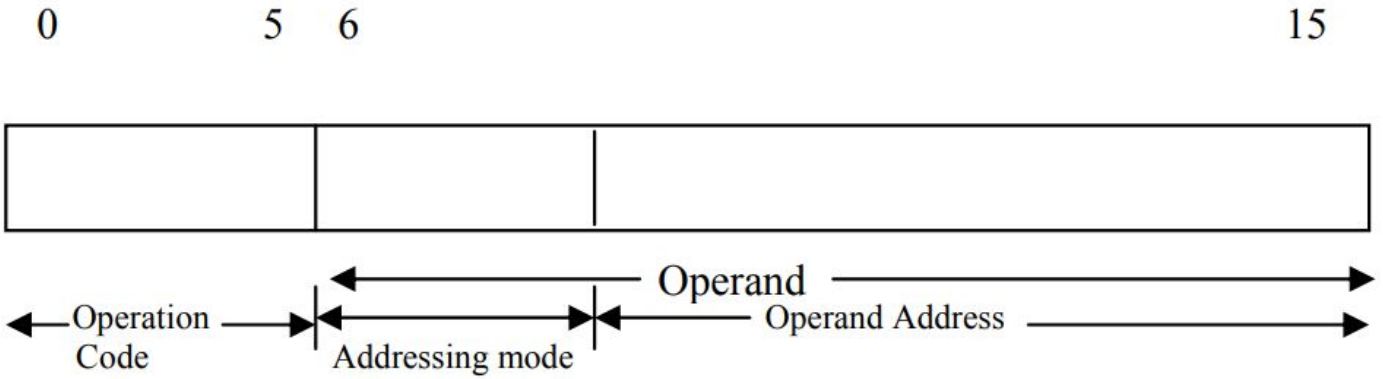
Moore's Law

Moore's Law is **the observation that the number of transistors on an integrated circuit will double every two years with minimal rise in cost**. Intel co-founder Gordon Moore predicted a doubling of transistors every year for the next 10 years.



Classification of parallel computers

A typical instruction in a program is composed of two parts: Opcode and Operand. The Operand part specifies the data on which the specified operation is to be done. (See Figure). The Operand part is divided into two parts: addressing mode and the Operand. The addressing mode specifies the method of determining the addresses of the actual data on which the operation is to be performed and the operand part is used as an argument by the method in determining the actual address.



Instruction Stream and Data Stream

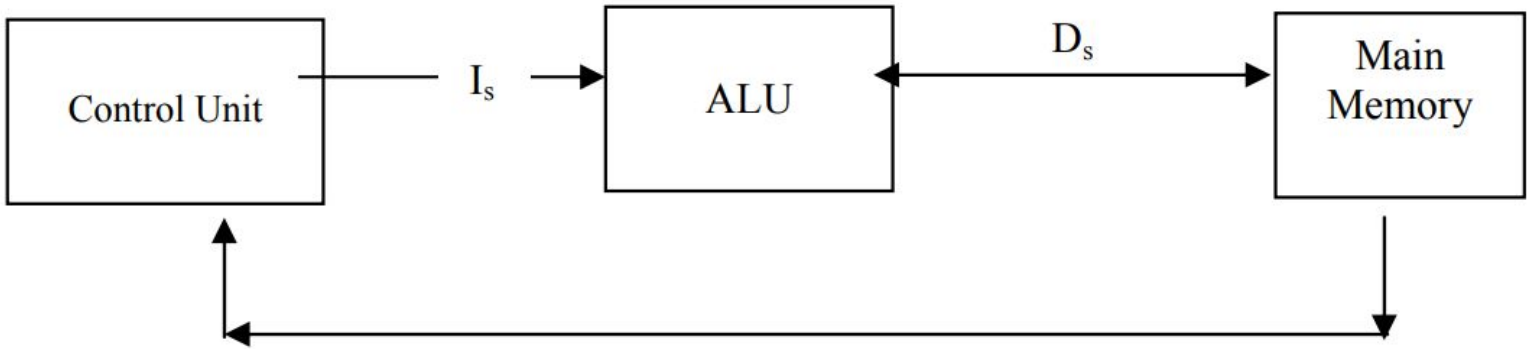
The term ‘stream’ refers to a sequence or flow of either instructions or data operated on by the computer. In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called instruction stream.

There is a flow of operands between processor and memory bi-directionally. This flow of operands is called data stream.

Flynn’s Classification

Based on multiplicity of instruction streams and data streams observed by the CPU during program execution.

I. Single Instruction and Single Data stream (SISD)

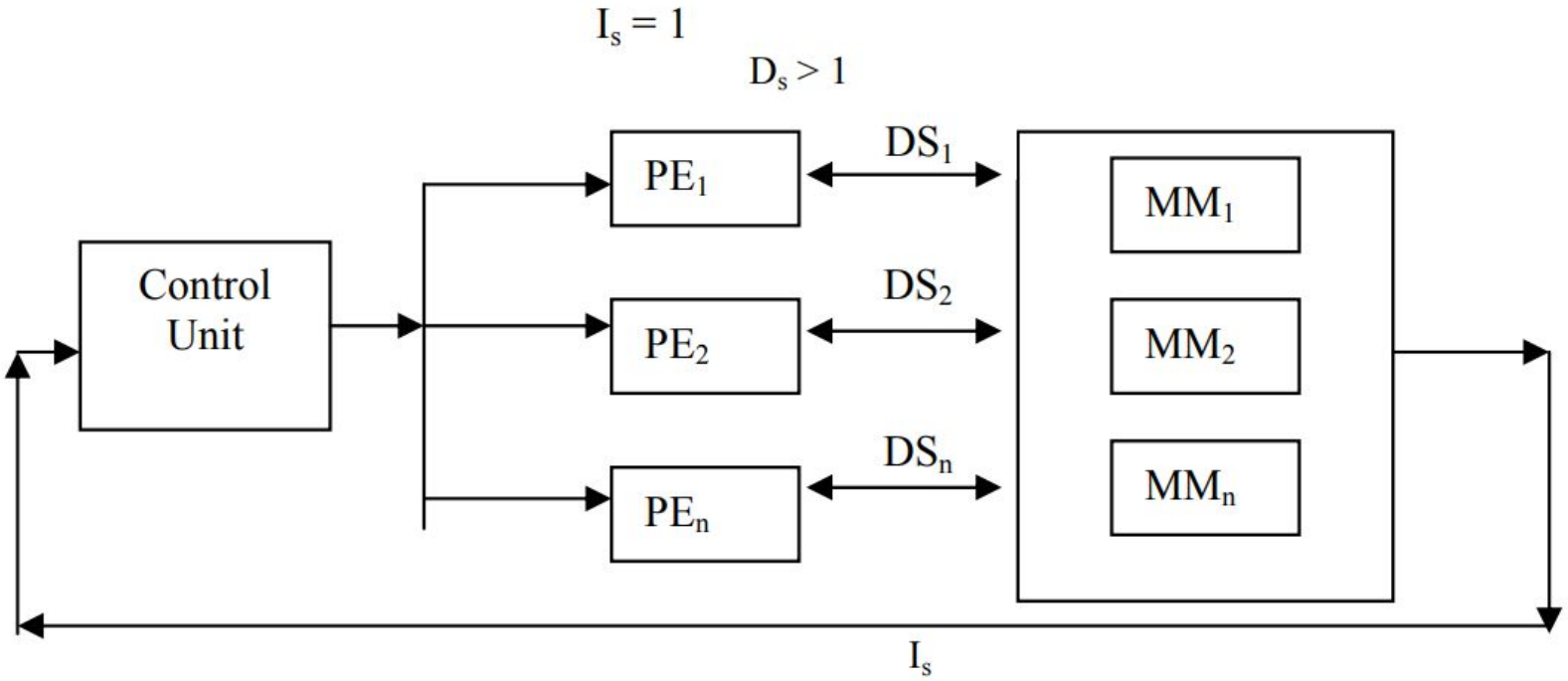


Examples of SISD machines

- CDC 6600 which is unpipelined but has multiple functional units.
- CDC 7600 which has a pipelined arithmetic unit.
- Amdhal 470/6 which has pipelined instruction processing.
- Cray-1 which supports vector processing.

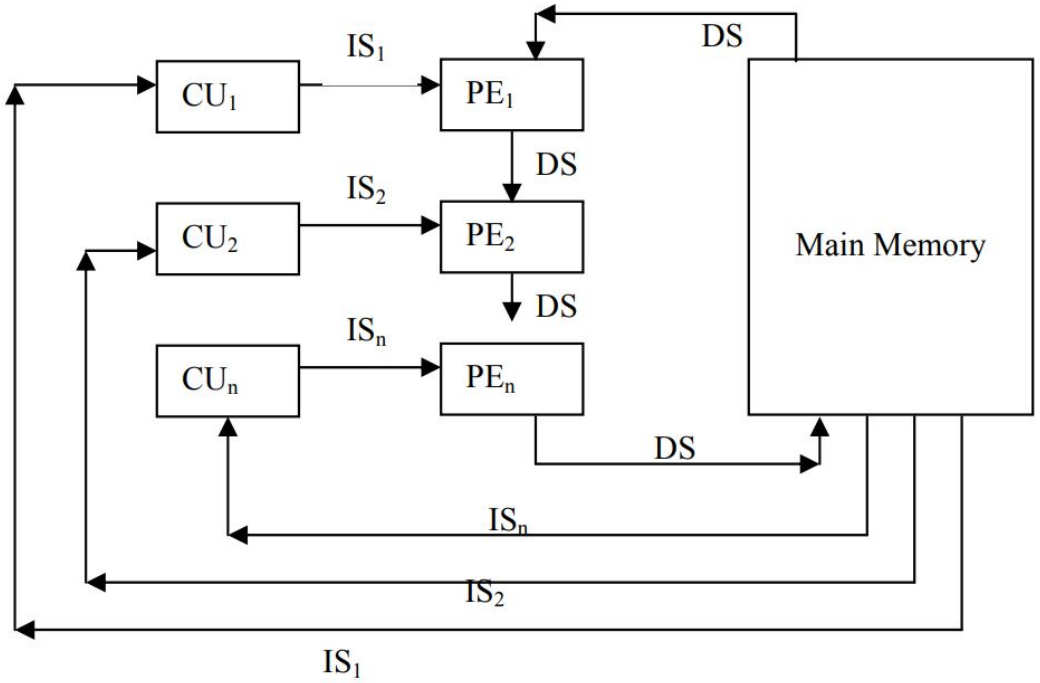
ii. Single Instruction and Multiple Data stream (SIMD)

ILLIAC-IV, PEPE, BSP, STARAN, MPP, DAP and the Connection Machine (CM-1).



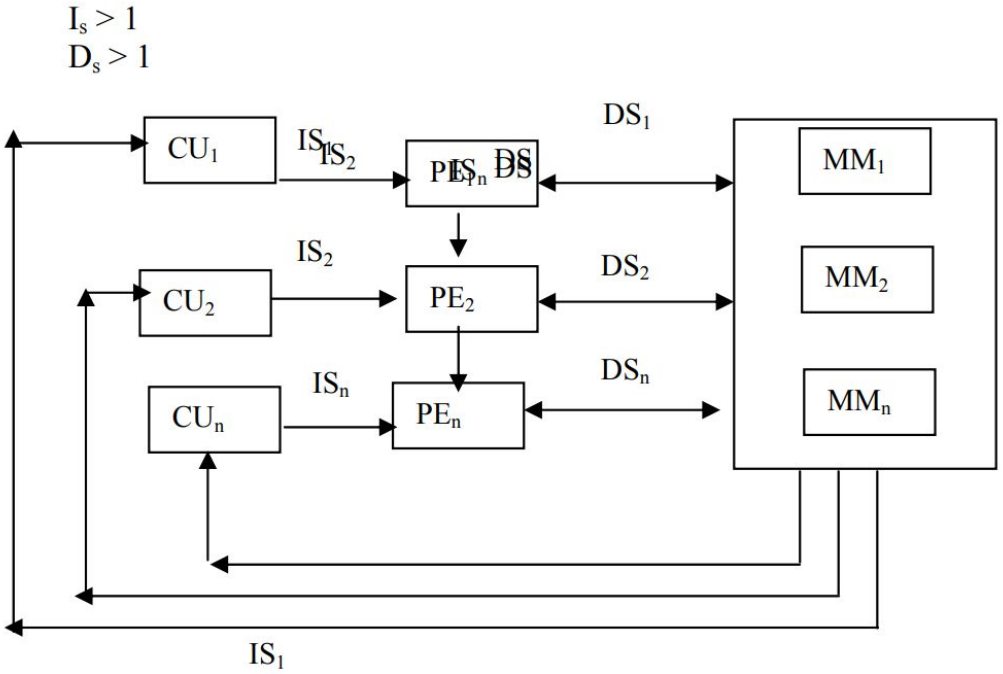
iii.Multiple Instruction and Single Data stream (MISD)

The only known example of a computer capable of MISD operation is the C.mmp built by Carnegie-Mellon University.



iv. Multiple Instruction and Multiple Data stream (MIMD)

All multiprocessor systems fall under this classification. Examples include; C.mmp, Burroughs D825, Cray-2, S1, Cray X-MP, HEP, Pluribus, IBM 370/168 MP, Univac 1100/80, Tandem/16, IBM 3081/3084, C.m*, BBN Butterfly, Meiko Computing Surface (CS-1), FPS T/40000, iPSC.



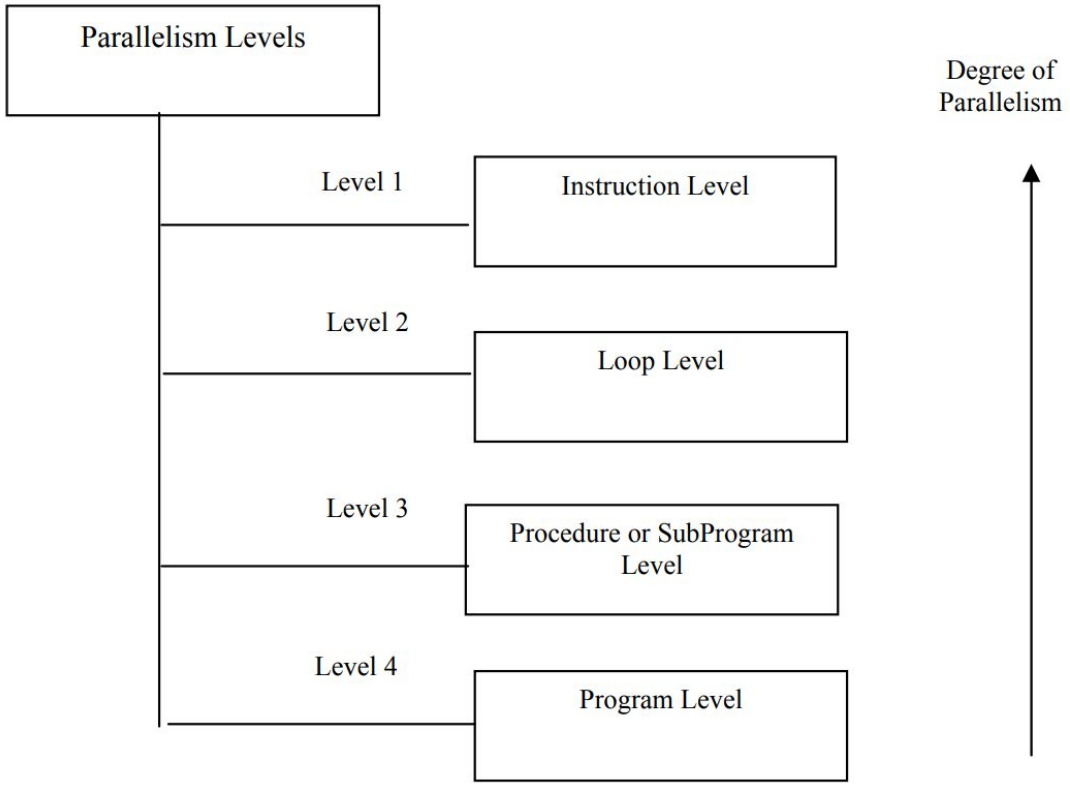
Parallelism based on Grain size

Grain size or Granularity is a measure which determines how much computation is involved in a process.

Fine Grain:

Medium Grain:

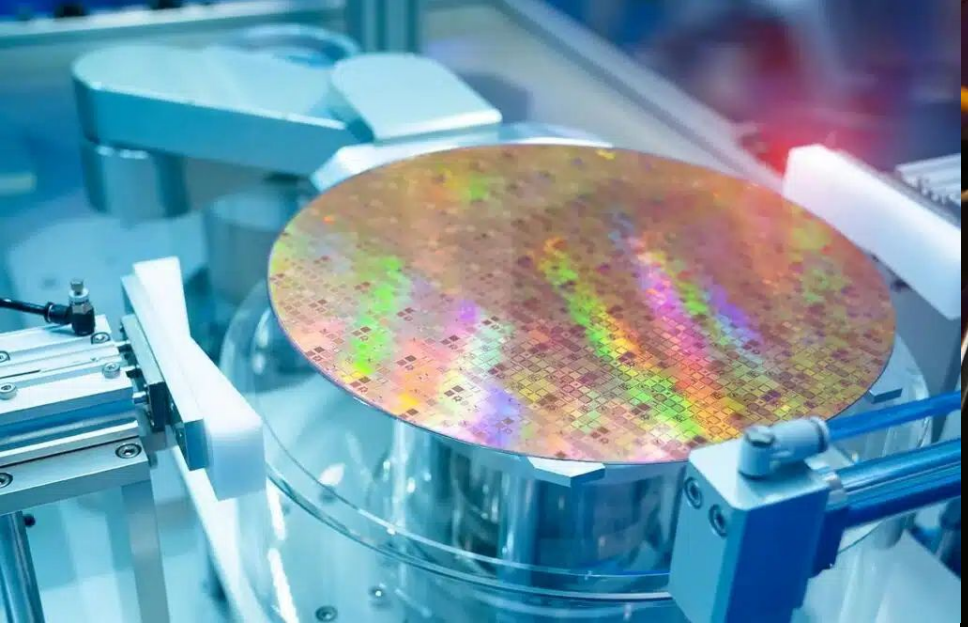
Coarse Grain:

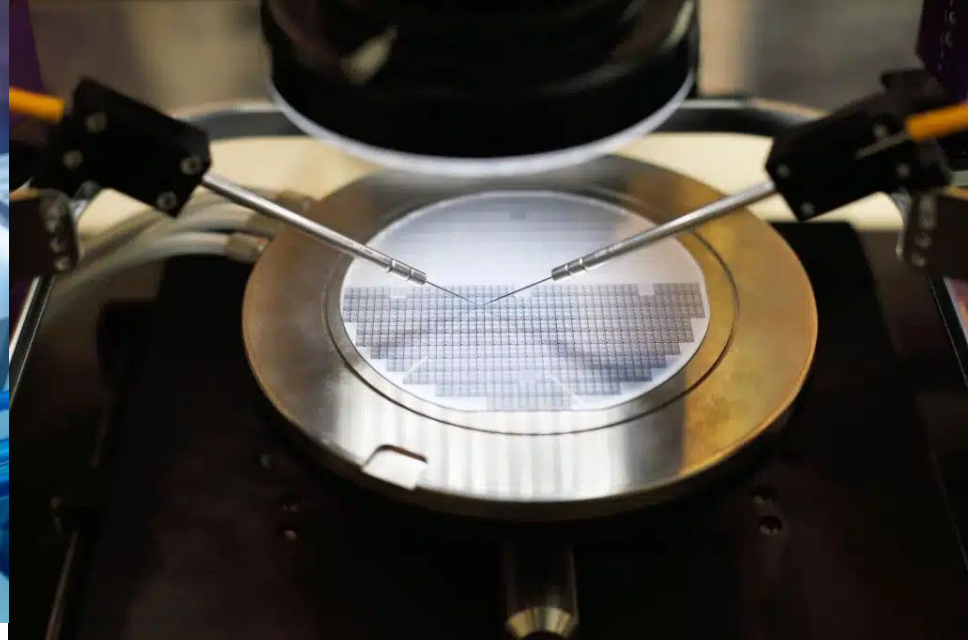


Trends in Cost

A wafer is need to be tested and chopped in to dies.

- First, a **wafer** is created. This is the shape that the silicon originally takes when it's grown from an ingot. There are many different circuits on one wafer.
- Then, the wafer is turned into various **dice**, each of which has a circuit on it. These dice are cut into individual pieces.
- After, you package it and turn it into an **integrated circuit**. This integrated circuit can then be put in a variety of devices, including phones, automobiles, and more.

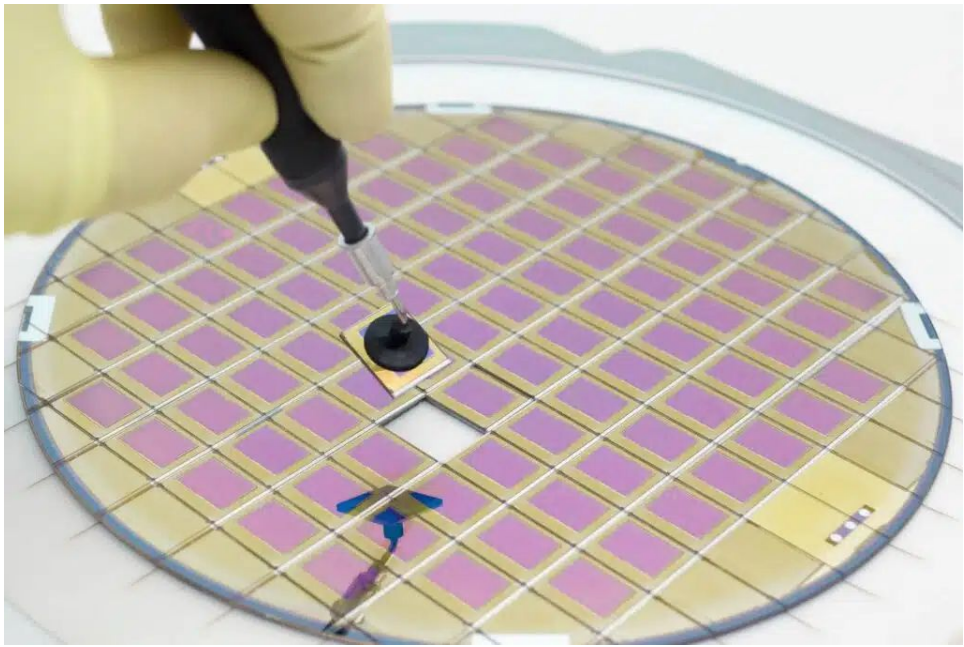




- Visual inspection is the first step, where each wafer is closely examined for visible flaws or damage. Technicians look for cracks, scratches, or any irregularities on the surface.
- Tests the electrical properties of the circuits on the wafer .

Based on the test results, individual dies on the wafer are sorted or “binned” according to their performance. Dies that meet the required specifications are classified for further processing, while those that don’t are either retested, if possible, or discarded.

Once the dies have been tested and sorted, the wafer is cut, or “diced,” into individual dies. This is typically done with a precision saw or laser cutting technique.



Integrated Circuit Cost

The cost of an integrated circuit can be expressed in three simple equations:

Cost per die = Cost per wafer /(Dies per wafer * yield)

Dies per wafer = Wafer area/ Die area

Yield = 1/ {1+ (Defects per area * Die area)/ 2}^2

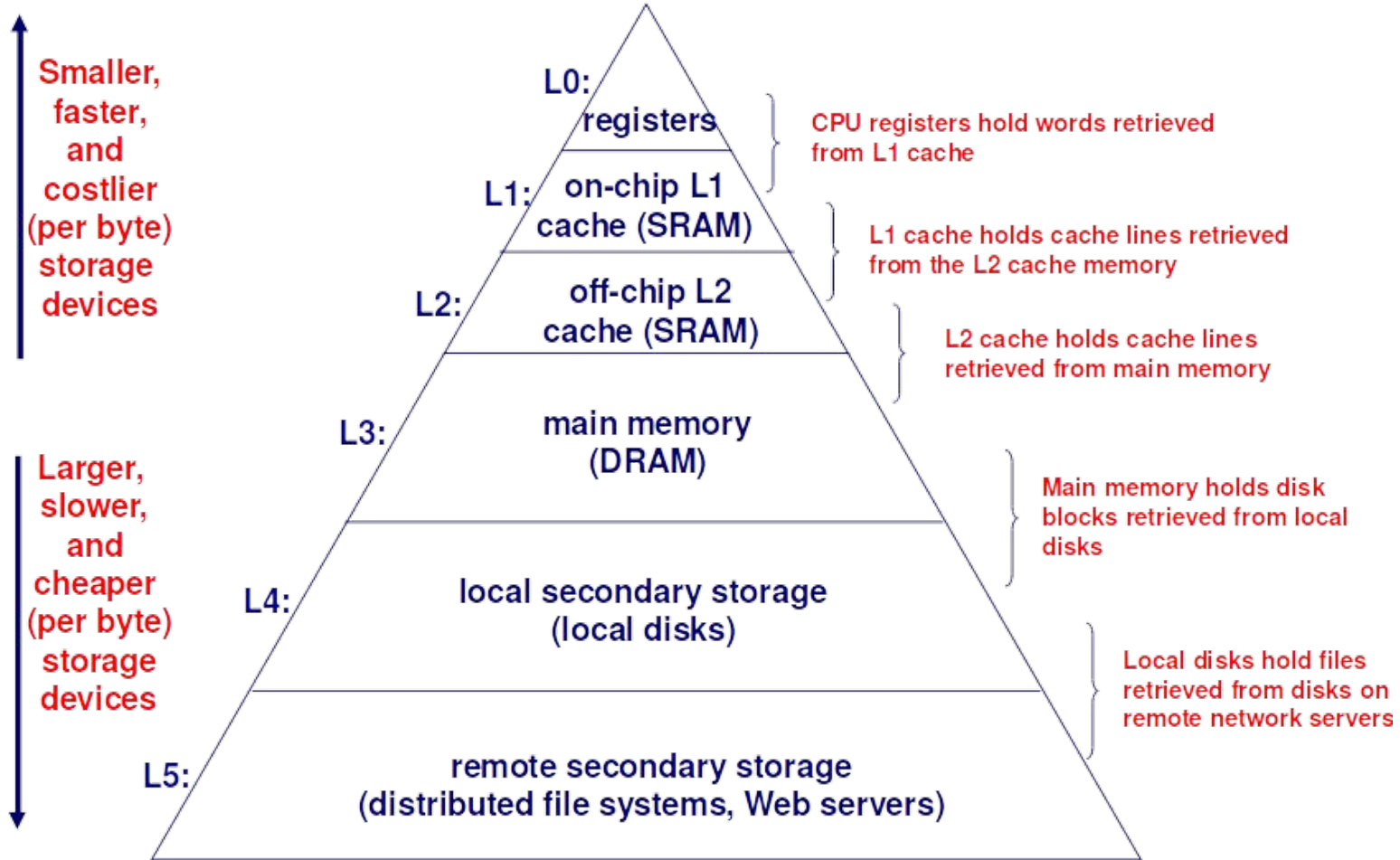
Cost of integrated circuit= (cost of die+cost of testing die+cost of packaging and final test)/Final test yield.

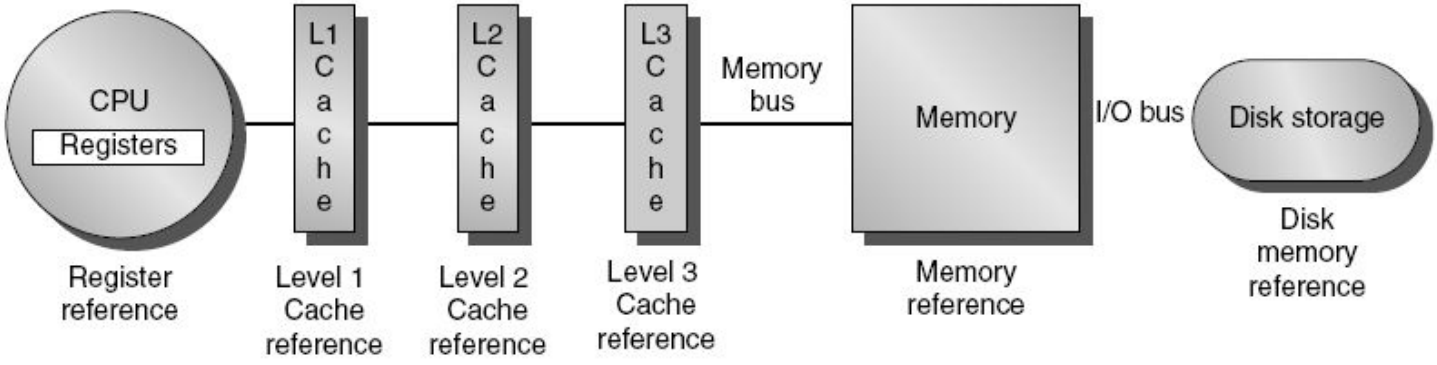
Dies/wafer=pie*(wafer diameter/2)^2/ Die area

Q.

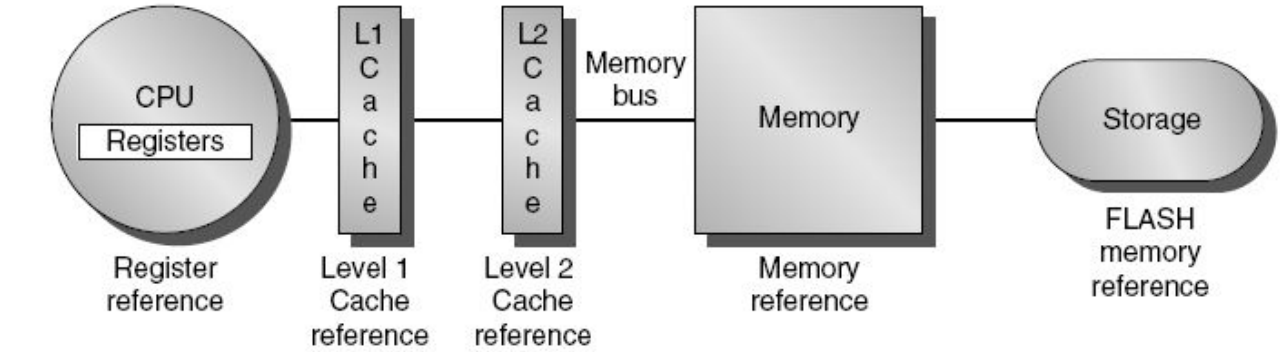
Find the number of dies per 300mm (30cm) wafer for a die that is 1.5 cm on a side.

An Example Memory Hierarchy





(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Cache Coherence Problem

In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy.

In a shared memory multiprocessor with a separate cache memory for each processor, it is possible to have many copies of any one instruction operand: one copy in the main memory and one in each cache memory.

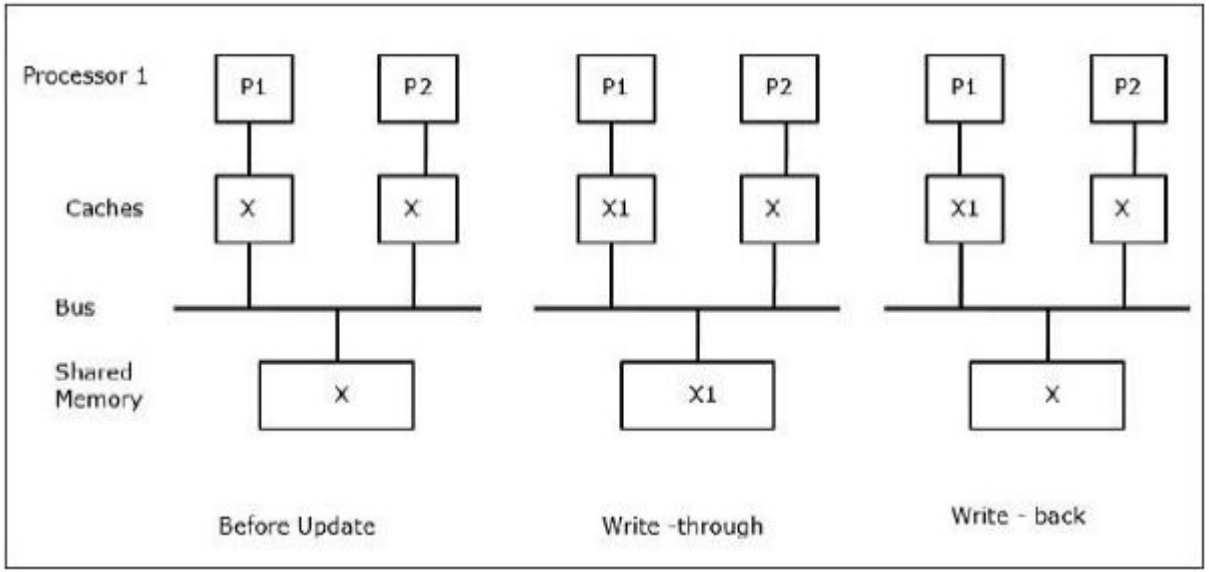
When one copy of an operand is changed, the other copies of the operand must be changed also.

Example : Cache and the main memory may have inconsistent copies of the same object.

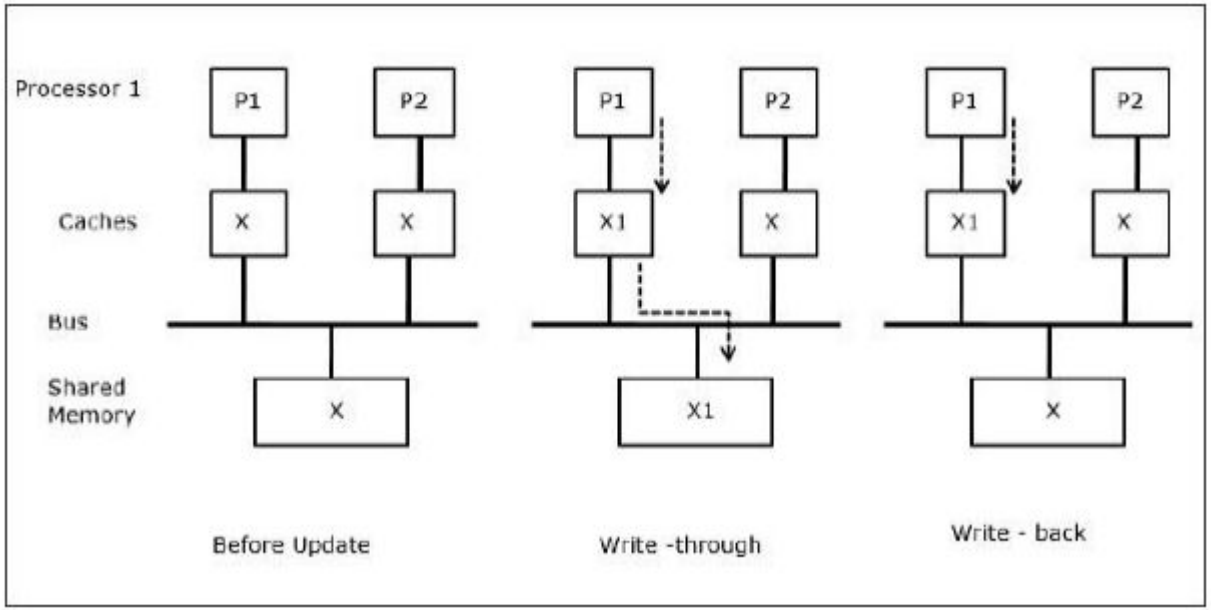
- In general, there are three sources of inconsistency problem – Sharing of writable data
- Process migration
- I/O activity

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

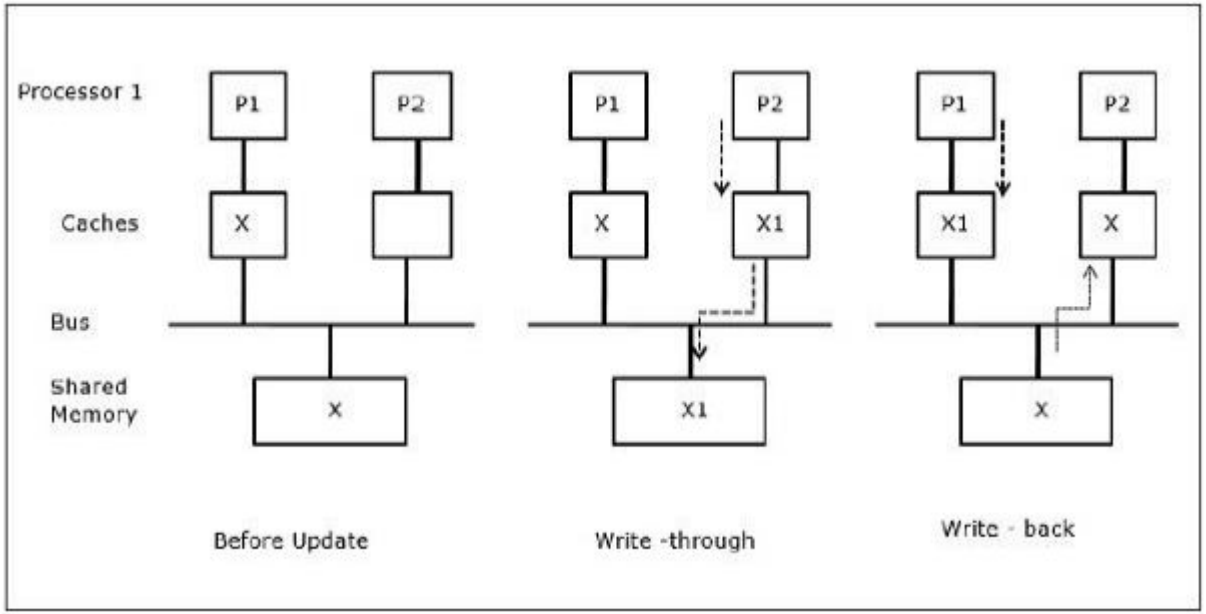
Figure 33.1


Processor 1
P1
P2
Caches
X
X
Bus
Shared Memory
X
Before Update
P1
P2
X1
X
X1
Write -through
P1
P2
X1
X
X
Write - back

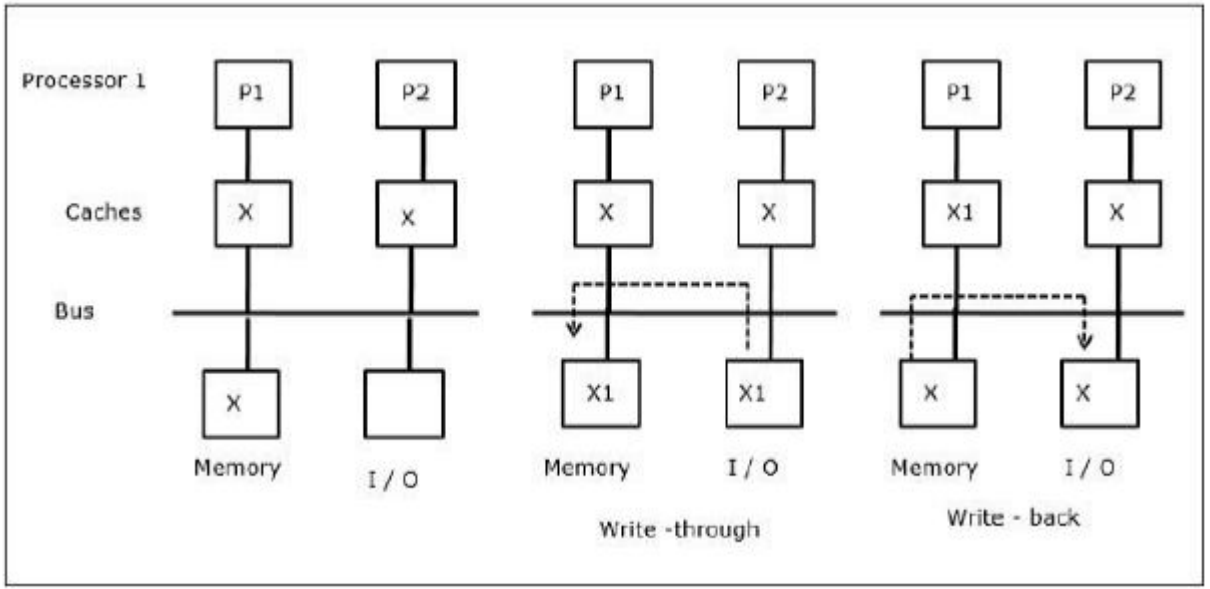
Sharing of writable data



Process Migration



I/O Activity


Processor 1
P1
P2
Caches
X
X
Bus
X
Memory
I / O
P1
P2
X
X
X1
X1
Memory
I / O
Write -through
P1
P2
X1
X
X
X
Memory
I / O
Write - back

Solution

Directory-based – In a directory-based system, the data being shared is placed in a common directory that maintains the coherence between caches.

The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed, the directory either updates or invalidates the other caches with that entry.

1. *Directory based*: The sharing status of a block of physical memory is kept in just one location, called the *directory*. The directory can also be distributed to improve scalability. Communication is established using point-to-point requests through the interconnection network.

Snoopy

Snoopy Cache Coherence Protocol: There are two ways to maintain the coherence requirement. One method is to ensure that a processor has exclusive access to a data item before it writes that item. This style of protocol is called a *write invalidate protocol* because it invalidates other copies on a write. It is the most common protocol, both for snooping and for directory schemes. Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs: All other cached copies of the item are invalidated.

The alternative to write invalidate is the *write broadcast* or *write update*mechanism. Here, all the cached copies are updated simultaneously. This requires more bandwidth. Also, when multiple updates happen to the same location, unnecessary updates are done. However, there is lower latency between the write and the read. We shall assume a write invalidate approach for the rest of the discussion.

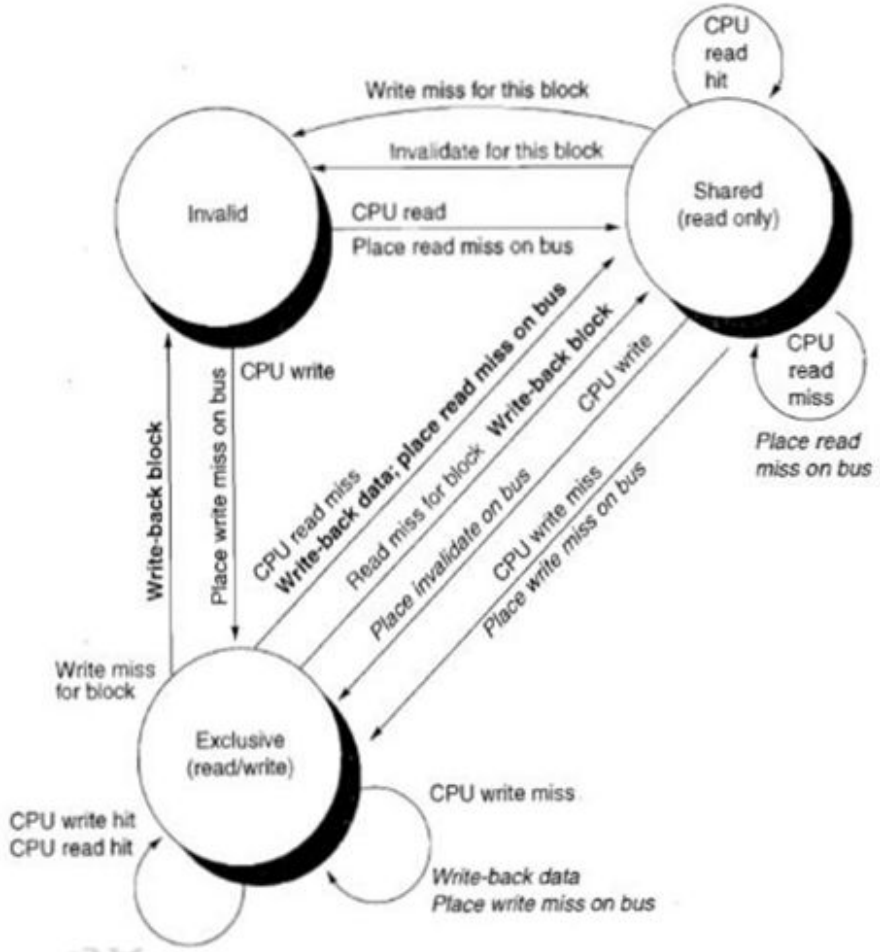

CPU read hit
Write miss for this block
Invalidate for this block
Invalid
Shared (read only)
CPU read
Place read miss on bus
CPU write
Write-back block
Place write miss on bus
CPU read miss
Write-back data; place read miss on bus
Read miss for block
Write-back block
Place invalidate on bus
CPU write
CPU write miss
Place write miss on bus
CPU read miss
Place read miss on bus
Write miss for block
Exclusive (read/write)
CPU write miss
CPU write hit
CPU read hit
Write-back data
Place write miss on bus

Figure 33.4