



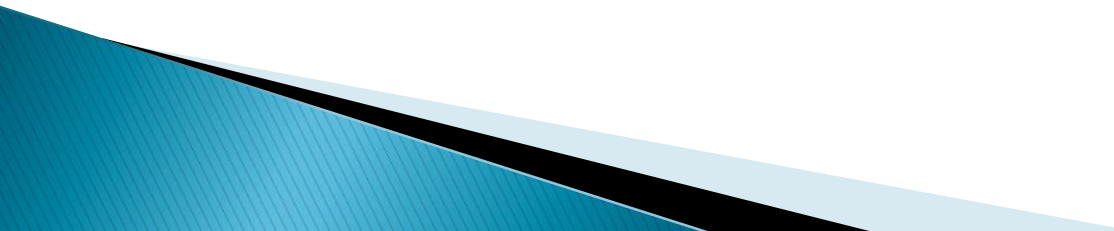
# Parallel Programming Platforms

- ▶ In traditional view of a computer, processor and memory are connected by data path.
  - ▶ Processor, memory and datapath present an bottlenecks to the overall processing rate of a computer system.
  - ▶ To overcome, multiplicity is implemented in processing unit, memory and datapath, which is shown either directly in implicit architecture and indirectly to the programmer in different forms.
  - ▶ The main objective is to provide sufficient details to programmer to be able to write efficient code on variety of platform.
  - ▶ Performance of various parallel algorithm.
- 

# Implicit Parallelism: Trends in Microprocessor Architectures\*

- ▶ Microprocessor clock speeds have posted impressive gains over the past two decades (two to three orders of magnitude).
  - ▶ Higher levels of device integration have made available a large number of transistors.
  - ▶ The question of how best to utilize these resources is an important one.
  - ▶ Current processors use these resources in multiple functional units and execute multiple instructions in the same cycle.
  - ▶ The precise manner in which these instructions are selected and executed provides impressive diversity in architectures.
- 

# Pipelining and Superscalar Execution

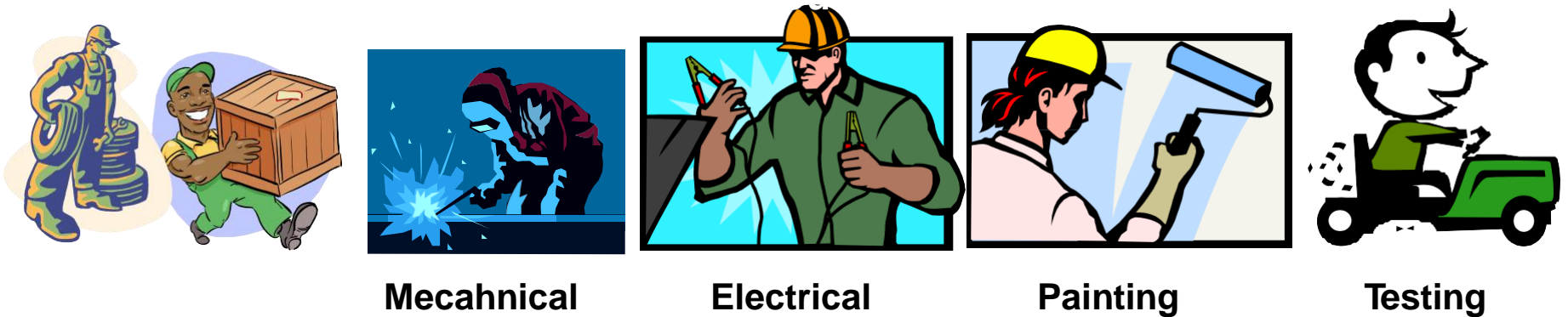
- ▶ Pipelining overlaps various stages of instruction execution to achieve performance.
  - ▶ At a high level of abstraction, an instruction can be executed while the next one is being decoded and the next one is being fetched.
  - ▶ This is akin to an assembly line for manufacture of cars.
- 

# Automobile Team Assembly



car assembled every four hours  
6 cars per day  
180 cars per month  
2,040 cars per year

# Automobile Assembly Line



**First car assembled in 4 hours (pipeline latency)**  
**thereafter, 1 car completed per hour**  
**21 cars on first day, thereafter 24 cars per day**  
**717 cars per month**  
**8,637 cars per year**  
***What gives 4X increase?***

# Throughput: Team Assembly

Red car  
started

Red car  
completed

Mechanical Electrical Painting Testing

Mechanical Electrical Painting Testing

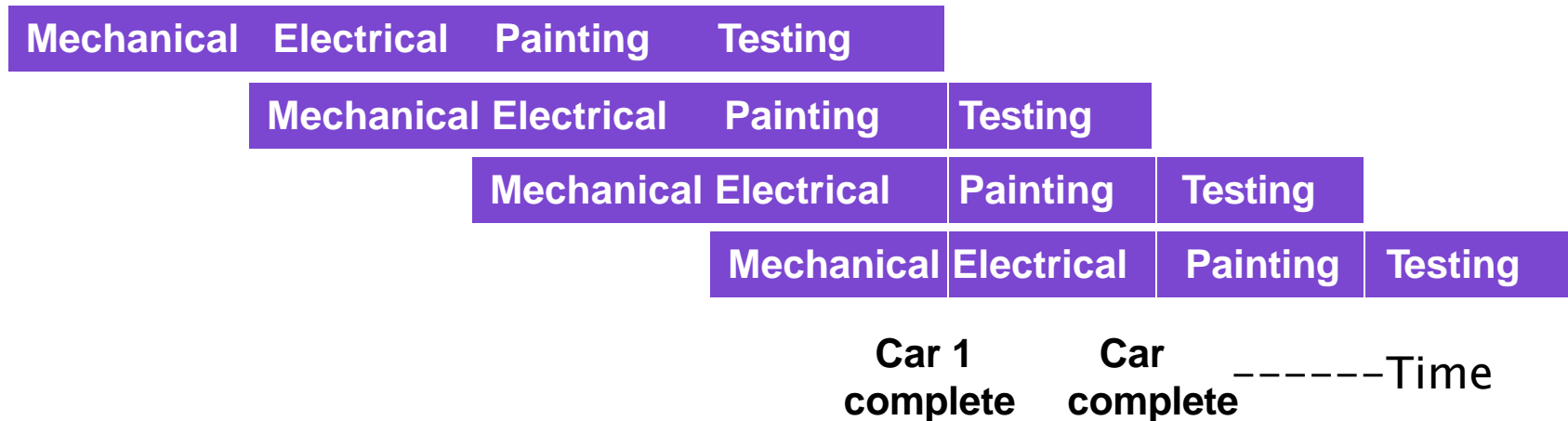
Blue car  
started

Blue car  
completed

Time of assembling one car =  $n$  hours where  $n$  is the number of nearly equal subtasks, each requiring 1 unit of time

Throughput =  $1/n$  cars per unit time

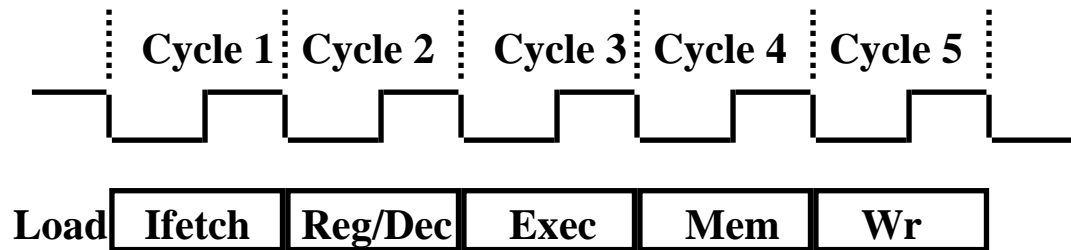
# Throughput: Assembly Line



Time to complete first car =  $n$  time units (latency)  
 Cars completed in time  $T = T - n + 1$   
 Throughput =  $1 - (n - 1) / T$  cars per unit time

$$\frac{\text{Throughput (assembly line)}}{\text{Throughput (team assembly)}} = \frac{1 - (n - 1) / T}{1/n} = n - \frac{n(n - 1)}{T} \rightarrow n \text{ as } T \rightarrow \infty$$

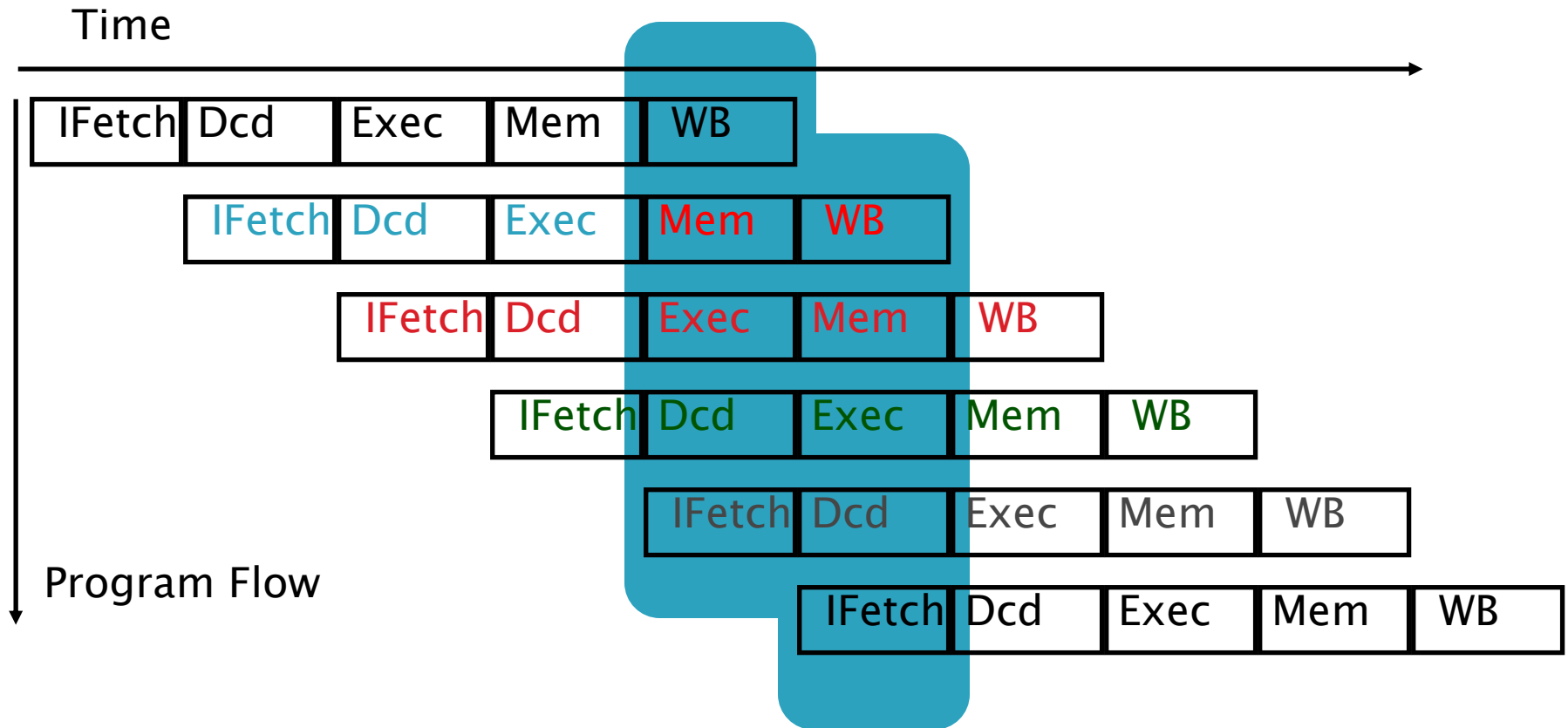
# Five Stages of an Instruction



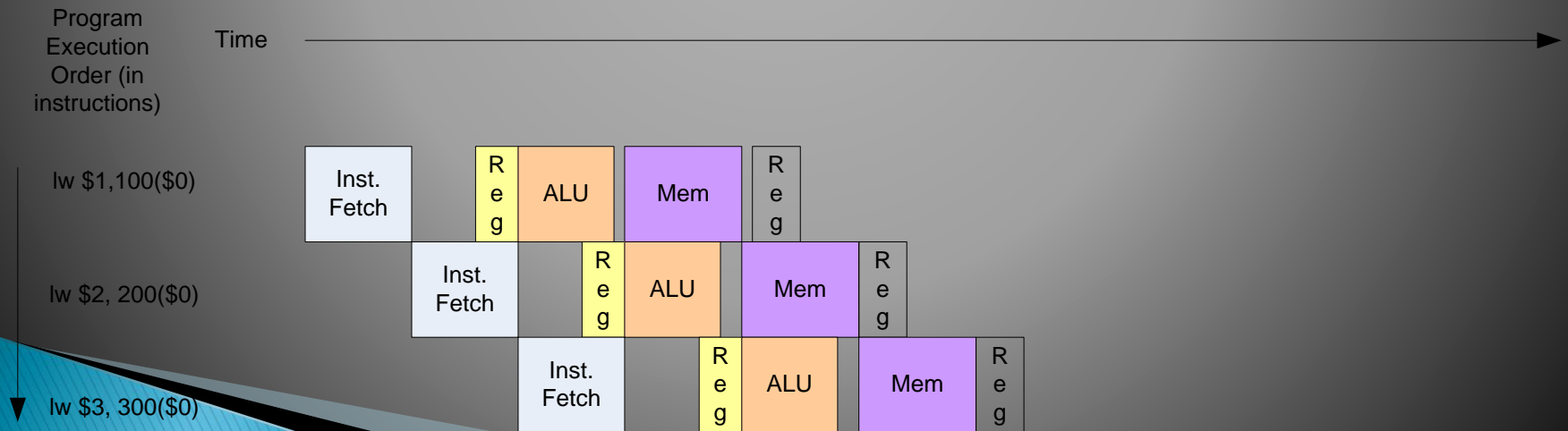
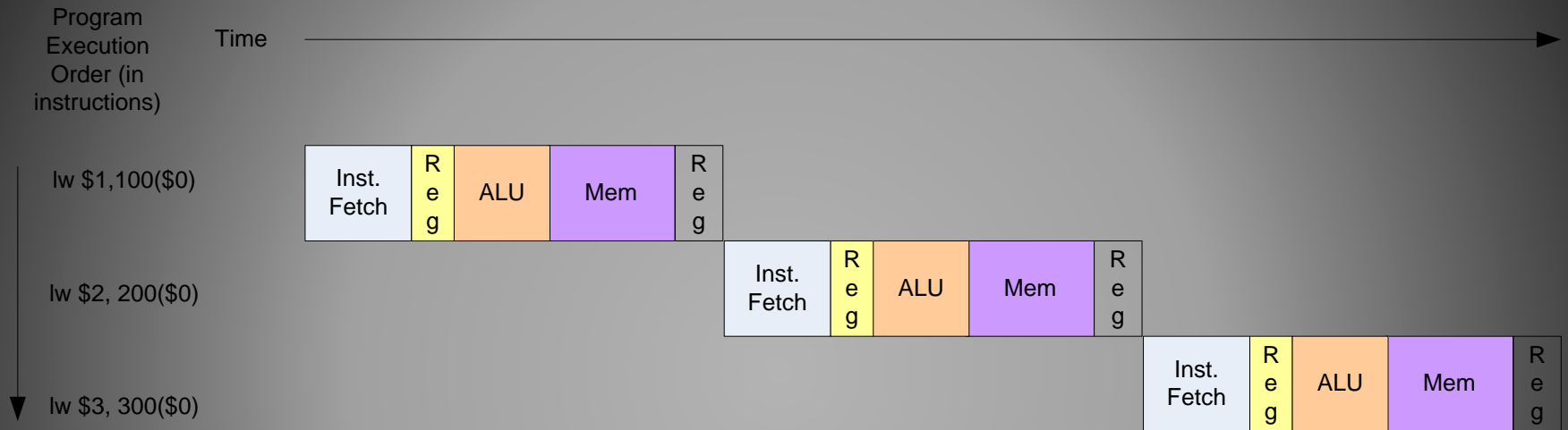
- ▶ **Ifetch: Instruction Fetch**
  - Fetch the instruction from the Instruction Memory
- ▶ **Reg/Dec: Registers Fetch and Instruction Decode**
- ▶ **Exec: Calculate the memory address**
- ▶ **Mem: Read the data from the Data Memory**
- ▶ **Wr: Write the data back to the register file**



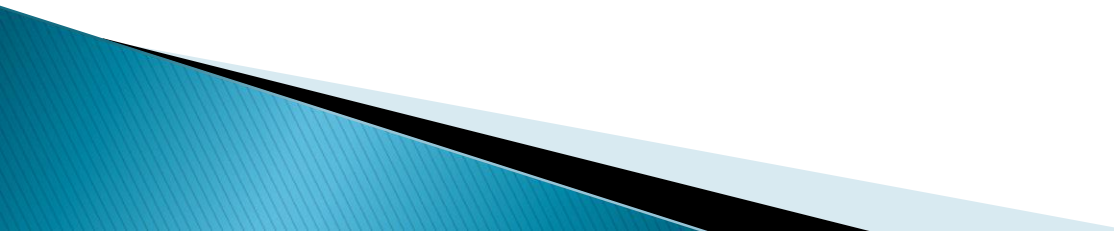
# Conventional Pipelined Execution Representation



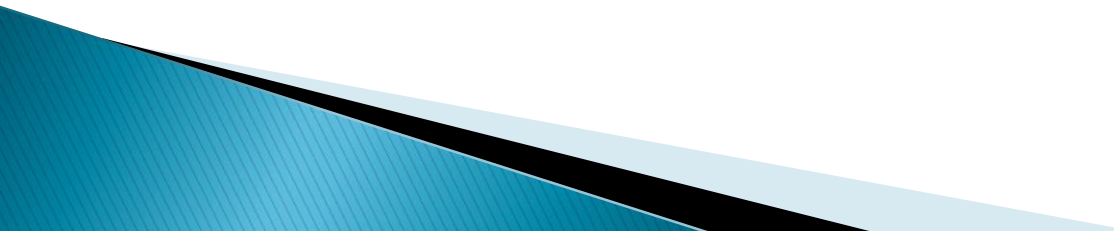
# Example



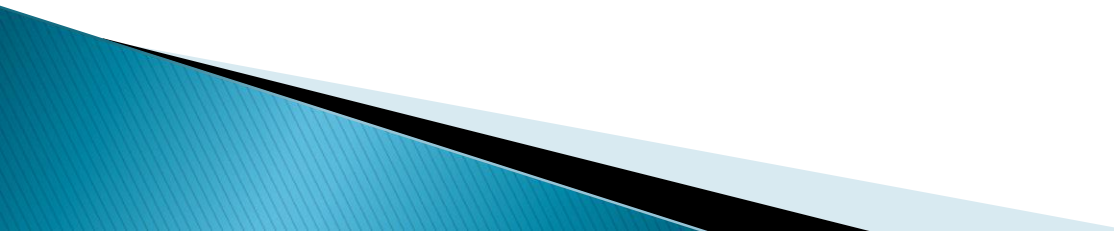
# Pipelining and Superscalar Execution

- ▶ Processor have relied on pipelines for improving execution rate.
  - ▶ To increase speed of a single pipeline, one would break down the tasks into smaller units.
  - ▶ To improve execution rate is to use multiple pipelines.
- 

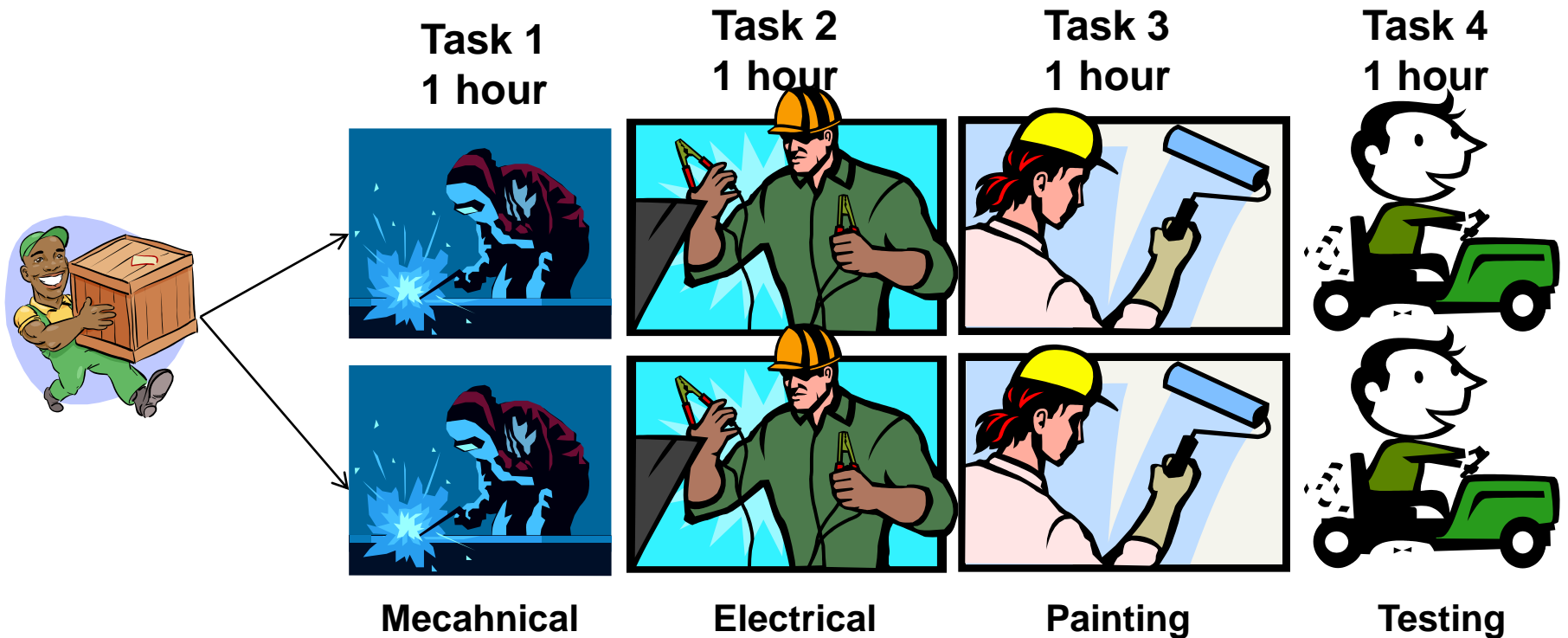
# Pipelining and Superscalar Execution

- ▶ Pipelining, however, has several limitations.
  - ▶ The speed of a pipeline is eventually limited by the slowest stage.
  - ▶ For this reason, conventional processors rely on very deep pipelines (20 stage pipelines in state-of-the-art Pentium processors).
  - ▶ However, in typical program traces, every 5- 6th instruction is a conditional jump! This requires very accurate branch prediction.
  - ▶ The penalty of a misprediction grows with the depth of the pipeline, since a larger number of instructions will have to be flushed.
- 

# Pipelining and Superscalar Execution

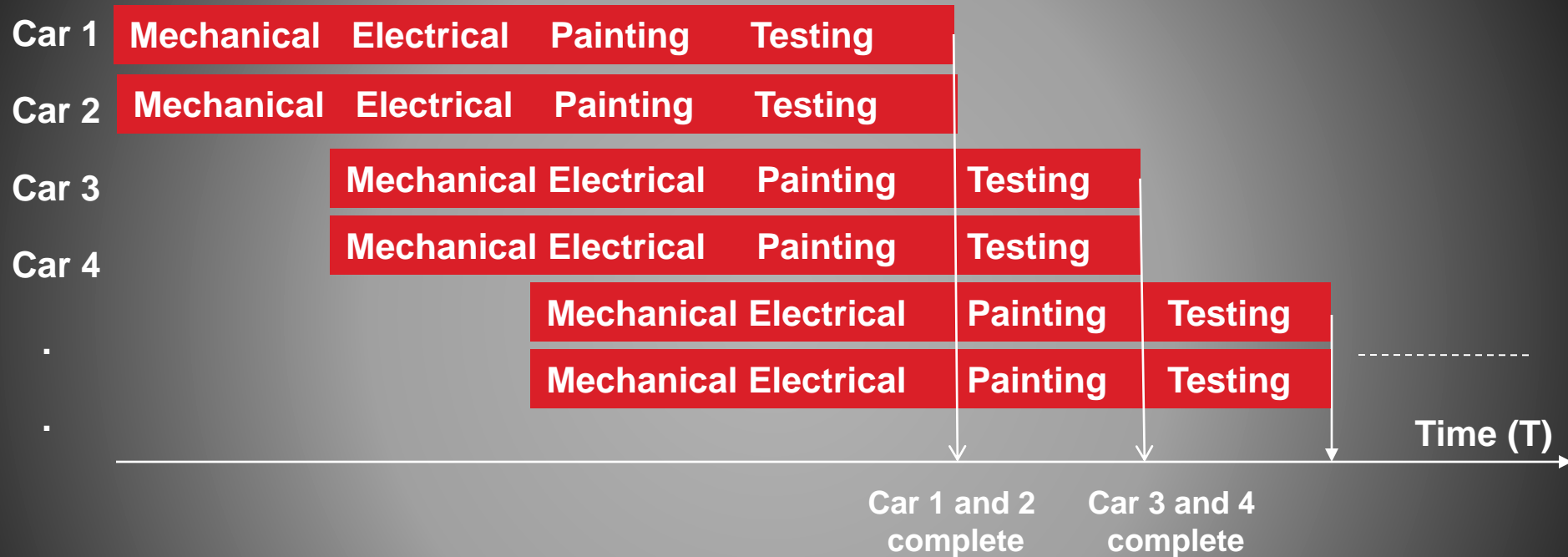
- ▶ One simple way of alleviating these bottlenecks is to use multiple pipelines. The question then becomes one of selecting these instructions.
  - ▶ Ability of processor to issue multiple instruction in the same cycle is – superscalar execution.
- 

# Multiple Assembly Line



Two cars are assembled in 4 hours (pipeline latency)  
thereafter 2 cars per hour  
42 cars on the first day and thereafter 48 cars per day  
1,432 cars per month  
17,272 cars per year

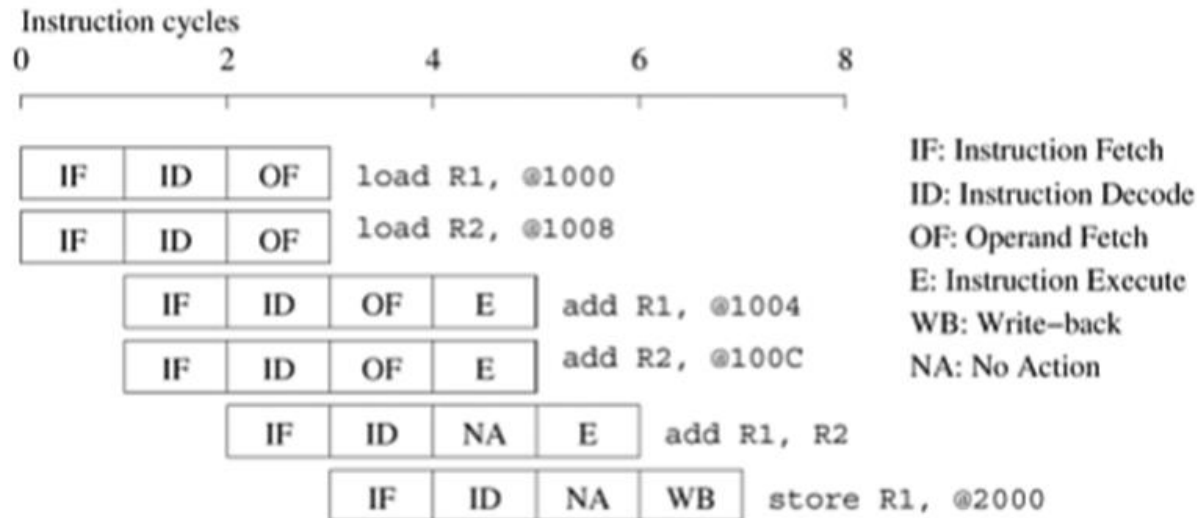
# Throughput: Multiple Assembly Line



**Figure 2.1. Example of a two-way superscalar execution of instructions.**

1. load R1, @1000	1. load R1, @1000	1. load R1, @1000
2. load R2, @1008	2. add R1, @1004	2. add R1, @1004
3. add R1, @1004	3. add R1, @1008	3. load R2, @1008
4. add R2, @100C	4. add R1, @100C	4. add R2, @100C
5. add R1, R2	5. store R1, @2000	5. add R1, R2
6. store R1, @2000		6. store R1, @2000
(i)	(ii)	(iii)

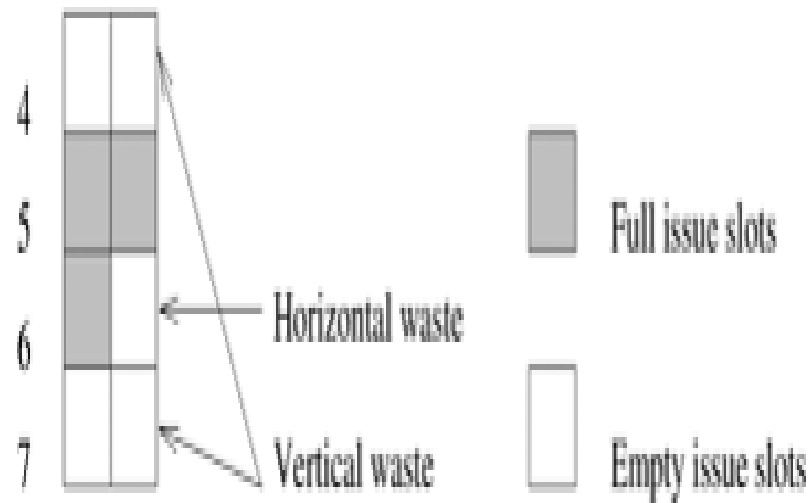
(a) Three different code fragments for adding a list of four numbers.



(b) Execution schedule for code fragment (i) above.



Clock cycle

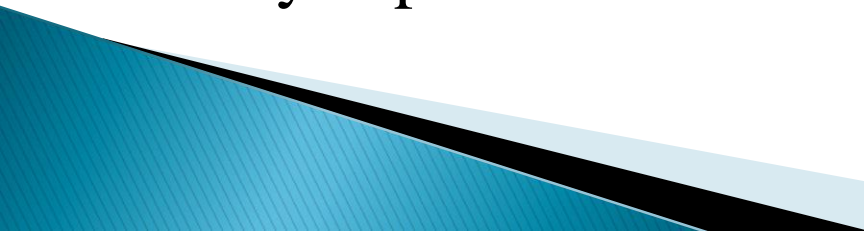


Adder Utilization

(c) Hardware utilization trace for schedule in (b).

# Superscalar Execution

## Issue mechanism:

- ▶ Not all functional units can be kept busy at all times.
  - ▶ If during a cycle, no functional units are utilized, this is referred to as vertical waste.
  - ▶ If during a cycle, only some of the functional units are utilized, this is referred to as horizontal waste.
  - ▶ Due to limited parallelism in typical instruction traces, dependencies, or the inability of the scheduler to extract parallelism, the performance of superscalar processors is eventually limited.
  - ▶ Conventional microprocessors typically support four-way superscalar execution.
- 


# Superscalar Execution

## Issue mechanism:

- ▶ In the above example, there is some wastage of resources due to data dependencies.
- ▶ The example also illustrates that different instruction mixes with identical semantics can take significantly different execution time.

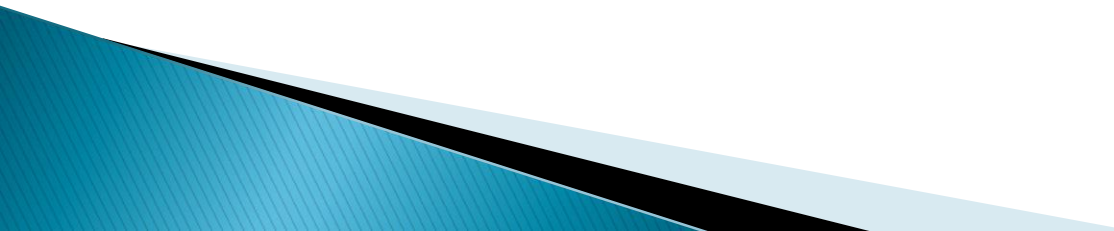
# Superscalar Execution

## Issue mechanism:

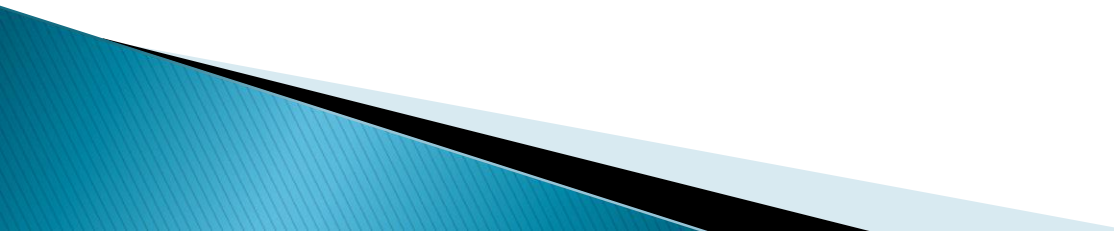
- ▶ Scheduling of instructions is determined by a number of factors:
    - True Data Dependency: The result of one operation is an input to the next.
    - Resource Dependency: Two operations require the same resource.
    - Branch Dependency: For conditional branch instruction destination is known only at the point of execution, scheduling instruction prior may lead to error.
    - The scheduler, a piece of hardware looks at a large number of instructions in an instruction queue and selects appropriate number of instructions to execute concurrently based on these factors.
    - The complexity of this hardware is an important constraint on superscalar processors.
- 

# Superscalar Execution

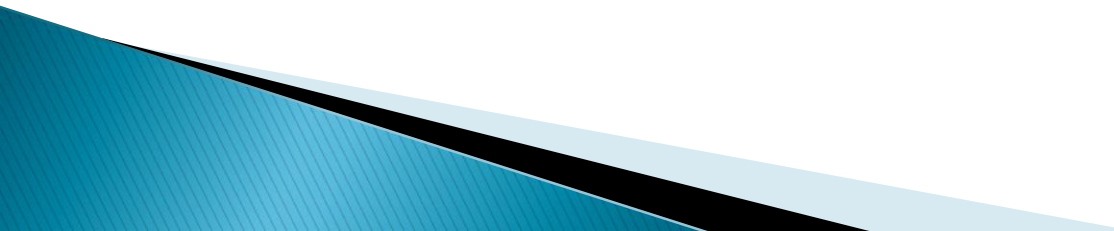
## Issue mechanism:

- ▶ In the simpler model, instructions can be issued only in the order in which they are encountered. That is, if the second instruction cannot be issued because it has a data dependency with the first, only one instruction is issued in the cycle. This is called in-order issue.
  - ▶ In a more aggressive model, instructions can be issued out of order. In this case, if the second instruction has data dependencies with the first, but the third instruction does not, the first and third instructions can be co-scheduled. This is also called dynamic issue.
  - ▶ Performance of in-order issue is generally limited.
- 

# Very Long Instruction Word (VLIW)Processor

- ▶ The hardware cost and complexity of the superscalar scheduler is a major consideration in processor design.
  - ▶ To address this issues, VLIW processors rely on compile time analysis to identify and bundle together instructions that can be executed concurrently.
  - ▶ These instructions are packed and dispatched together as single instruction long word, and thus the name very long instruction word.
- 

# Very Long Instruction Word (VLIW)Processor:Advantages

- ▶ To resolve dependency and resource availability at compile time
    - Scheduling is done in software
    - Decoding is easy
    - Additional parallel instruction made available to control parallel execution
- 

# Very Long Instruction Word (VLIW)Processor: Limitations

- ▶ Issue hardware is simpler.
  - ▶ Compiler has a bigger context from which to select co-scheduled instructions.
  - ▶ Compilers, however, do not have runtime information such as cache misses. Scheduling is, therefore, inherently conservative.
  - ▶ Branch and memory prediction is more difficult.
  - ▶ VLIW performance is highly dependent on the compiler. A number of techniques such as loop unrolling, speculative execution, branch prediction are critical.
  - ▶ Typical VLIW processors are limited to 4-way to 8-way parallelism.
- 