Chap. 9 Pipeline and Vector Processing



# Parallel Processing

## *Simultaneous* data processing tasks for the purpose of increasing the

**9-1**

## **=** computational speed

* + Perform *concurrent* data processing to achieve faster execution time
  + Multiple Functional Unit :

**Parallel Processing Example**

* + - ***Separate the execution unit into eight functional units operating in parallel***
  + Computer Architectural Classification

**To Memory**

**Floatint-point multiply**

**Processor registers**

**Adder-subtractor**

**Floatint-point divide**

**Floatint-point add-subtract**

**Incrementer**

**Shift unit**

**Logic unit**

**Integer multiply**

### Data-Instruction Stream : Flynn

* + - Serial versus Parallel Processing : Feng
    - Parallelism and Pipelining : Händler
  + Flynn’s Classification
    - 1) **SISD** (Single Instruction - Single Data stream)

» for practical purpose: only one processor is useful

» Example systems : Amdahl 470V/6, IBM 360/91



**MM**

**DS**

**IS**

**CU**

**IS**

**PU**

* 2) **SIMD**



**IS**

**MM n**

**DS n**

**IS**

**CU**

**DS 2**

**DS 1**

**Shared memmory**

**PU n**

**MM 2**

**PU 2**

**MM 1**

**PU 1**

(Single Instruction - Multiple Data stream)

» vector or array operations

* + one vector operation includes many operations on a data stream

» Example systems : CRAY -1, ILLIAC-IV

* 3) **MISD**

(Multiple Instruction - Single Data stream)

» Data Stream Bottle neck

**DS**

**PU n**

**IS n**

**CU n**

**IS n**

**MM 1**

**MM 2**

**MM n**

**IS 2**

**CU 2**

**IS 2**

**Shared memory**

**PU 1**

**IS 1**

**CU 1**

**IS 1**

**DS**

**PU 2**

* 4) **MIMD**

(Multiple Instruction - Multiple Data stream)

» Multiprocessor System



**MM n**

**IS n**

**CU n**

**IS n**

v

v

**MM 2**

**IS 2**

**CU 2**

**IS 2**

**MM 1**

**DS**

**IS 1**

**CU 1**

**IS 1**

**Shared memory**

**PU n**

**PU 2**

**PU 1**

## Main topics in this Chapter

### Pipeline processing :

» Arithmetic pipeline :

» Instruction pipeline :

Large vector, Matrices, Array Data

### Vector processing :adder/multiplier pipeline

* + Array processing : array processor

» Attached array processor :

» SIMD array processor :

# Pipelining

## Pipelining

### Decomposing a sequential process into suboperations

* + - Each subprocess is executed in a special dedicated segment concurrently
  + Pipelining Example
    - Multiply and add operation :
    - 3 Suboperation Segment

*Ai* \* *Bi*  *Ci*

( for i = 1, 2, …, 7 )

»1)

»2)

»3)

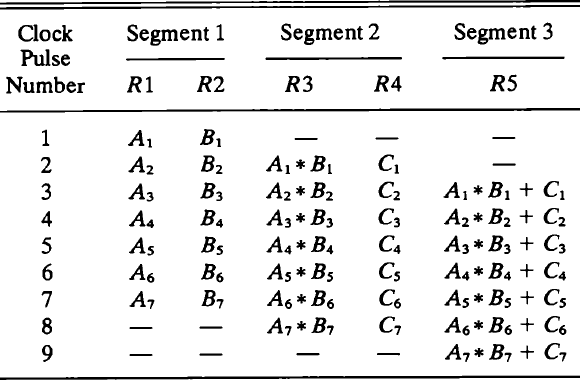
: Input Ai and Bi

: Multiply and input Ci

*R*1  *Ai*, *R*2  *Bi*

*R*3  *R*1\* *R*2, *R*4  *Ci R*5  *R*3  *R*4

: Add Ci



# Pipelining

## General considerations

### 4 segment pipeline :

» **S** : Combinational circuit for Suboperation

» **R** : Register(intermediate results between the segments)

### Space-time diagram :

» Show segment utilization as a function of time



**Clock cycles**

**1**

**2**

**3**

**4**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |  |
| **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |  |  |  |
|  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |  |
|  |  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |
|  |  |  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |

* + - Task : T1, T2, T3,…, T6

» Total operation performed going through all the segment

**Segment**

## Speedup S : Nonpipeline / Pipeline

### With pipeline: k-segment pipeline with a clock time tp to execute n tasks

* + Without pipeline: Each task takes tn
  + **S** = **n • tn / ( k + n - 1 ) • tp** = 6 • 6 tn / ( 4 + 6 - 1 ) • tp = 36 tn / 9 tn = 4

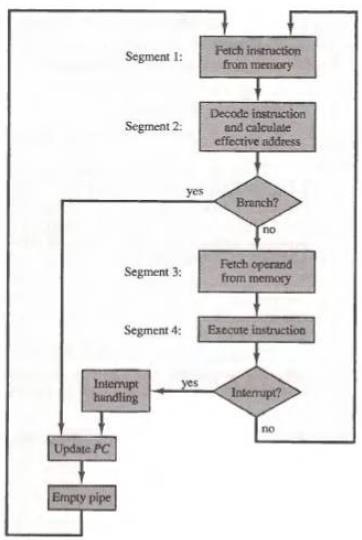
» n : *task number* ( 6 )

» tn : *time to complete each task in nonpipeline* ( 6 cycle times = 6 tp)



» tp : *clock cycle time* ( 1 clock cycle )

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| » | k : *segment number* ( 4 ) |  | | | | | | | | | | | |
|  |  | **Clock cycles** |  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |  |
|  |  |  | **1** | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |  |  |  |
|  |  | **ment** | **2** |  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |  |  |
|  |  | **Seg** | **3** |  |  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |  |
|  |  |  | **4** |  |  |  | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |



# Instruction Pipeline

## Instruction Cycle

### Fetch the instruction from memory

* + 1. Decode the instruction
    2. Calculate the effective address
    3. Fetch the operands from memory
    4. Execute the instruction
    5. Store the result in the proper place
* Instruction Pipeline
  + Example : Four-segment Instruction Pipeline
    - Four-segment CPU pipeline :

» 1) **FI** : Instruction Fetch

» 2) **DA** : Decode Instruction & calculate EA

» 3) **FO** : Operand Fetch

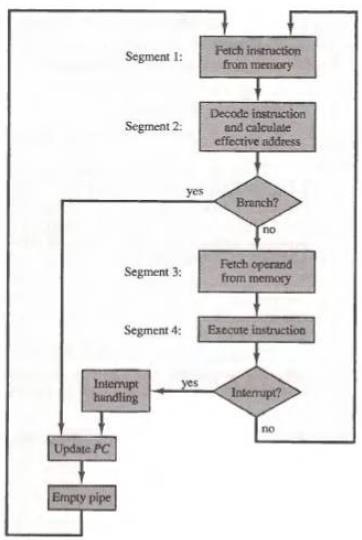
» 4) **EX** : Execution

### Timing of Instruction Pipeline :

» **Instruction 3 Branch**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step :** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| **Instruction :**  **(Branch)** | **1** | **FI** | **DA** | **FO** | **EX** |  |  |  |  |  |  |  |  |  |
| **2** |  | **FI** | **DA** | **FO** | **EX** |  |  |  |  |  |  |  |  |
| **3** |  |  | **FI** | **DA** | **FO** | **EX** |  |  |  |  |  |  |  |
| **4** |  |  |  | **FI** |  |  | **FI** | **DA** | **FO** | **EX** |  |  |  |
| **5** |  |  |  |  |  |  |  | **FI** | **DA** | **FO** | **EX** |  |  |
| **6** |  |  |  |  |  |  |  |  | **FI** | **DA** | **FO** | **EX** |  |
| **7** |  |  |  |  |  |  |  |  |  | **FI** | **DA** | **FO** | **EX** |

**No Branch**



**Branch**

* + Pipeline Conflicts : 3 major difficulties
    - 1) Resource conflicts

» memory access by two segments at the same time

### 2) Data dependency

» when an instruction depend on the result of a previous instruction, but this result is not yet available

### 3) Branch difficulties

» branch and other instruction (interrupt, ret, ..) that change the value of PC

## Data Dependency

### Hardware

» Hardware Interlock

* + - * previous instruction Hardware Delay

» Operand Forwarding

* + - * previous instruction

### Software

» Delayed Load

* + - * previous instruction No-operation instruction

## Delayed Branch

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Clock cycles :** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1. Load** | **I** | **A** | **E** |  |  |  |  |  |  |  |
| **2. Increment** |  | **I** | **A** | **E** |  |  |  |  |  |  |
| **3. Add** |  |  | **I** | **A** | **E** |  |  |  |  |  |
| **4. Subtract** |  |  |  | **I** | **A** | **E** |  |  |  |  |
| **5. Branch to X** |  |  |  |  | **I** | **A** | **E** |  |  |  |
| **6. No-operation** |  |  |  |  |  | **I** | **A** | **E** |  |  |
| **7. No-operation** |  |  |  |  |  |  | **I** | **A** | **E** |  |
| **8. Instruction in X** |  |  |  |  |  |  |  | **I** | **A** | **E** |
| 1. **Using no-operation instructions** 2. **Rearranging the instructions** | | | | | | | | | | |

» 1) No-operation instruction

» 2) Instruction Rearranging

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Clock cycles :** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **1. Load** | **I** | **A** | **E** |  |  |  |  |  |
| **2. Increment** |  | **I** | **A** | **E** |  |  |  |  |
| **3. Branch to X** |  |  | **I** | **A** | **E** |  |  |  |
| **4. Add** |  |  |  | **I** | **A** | **E** |  |  |
| **5. Subtract** |  |  |  |  | **I** | **A** | **E** |  |
| **6. Instruction in X** |  |  |  |  |  | **I** | **A** | **E** |

# 9-5 RISC Pipeline



**Conflict**

**Clock cycles :**

**1 2 3 4 5 6**

**1. Load R1**

**I A E**

**2. Load R2**

**I A E**

**3. Add R1+R2**

**I A E**

**4. Store R3**

**I A E**

**(a) Pipeline timing with data conflict**

**(b) Pipeline timing with delayed load**

## RISC CPU

### Instruction Pipeline

* + - Single-cycle instruction execution
    - Compiler support
  + Example : Three-segment Instruction Pipeline
    - 3 Suboperations Instruction Cycle

» 1) **I** : Instruction fetch

» 2) **A** : Instruction decoded and ALU operation

» 3) **E** : Transfer the output of ALU to a register, memory, or PC

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Clock cycles :** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1. Load R1** | **I** | **A** | **E** |  |  |  |  |
| **2. Load R2** |  | **I** | **A** | **E** |  |  |  |
| **3. No-operation** |  |  | **I** | **A** | **E** |  |  |
| **4. Add R1+R2** |  |  |  | **I** | **A** | **E** |  |
| **5. Store R3** |  |  |  |  | **I** | **A** | **E** |

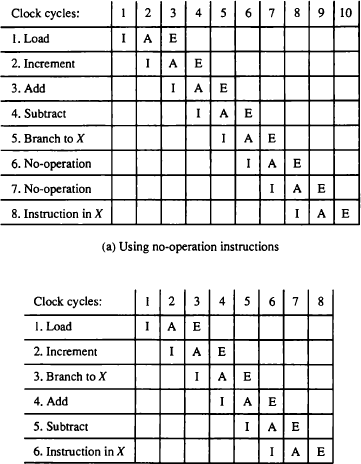
### Delayed Load :

» Instruction(**ADD R1 + R3**) Conflict

» Delayed Load

* + - * No-operation

### Delayed Branch :

* 9-5 RISC Pipeline
  + Example : Three-segment Instruction Pipeline
    - 3 Suboperations

Instruction Cycle

» 1) **I** : Instruction fetch

» 2) **A** : Instruction decoded and ALU operation

» 3) **E** : Transfer the output of ALU to a register,

memory, or PC

### Delayed Branch :

* 9-6 Vector Processing
  + Science and Engineering Applications
    - Long-range weather forecasting, Petroleum explorations, Seismic data analysis, Medical diagnosis, Aerodynamics and space flight simulations, Artificial intelligence and expert systems, Mapping the human genome, Image processing
  + Vector Operations
    - Arithmetic operations on large arrays of numbers
    - Conventional scalar processor

» Machine language

Initialize I = 0

20 Read A(I)

Read B(I)

Store C(I) = A(I) + B(I)

Increment I = I + 1 If I  100 go to 20 Continue

### Vector processor

» Single vector instruction

C(1:100) = A(1:100) + B(1:100)

» Fortran language

DO 20 I = 1, 100 20 C(I) = A(I) + B(I)

## Vector Instruction Format :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation code** | **Base address source 1** | **Base address source 2** | **Base address destination** | **Vector length** |

***ADD A B C 100***

* + Matrix Multiplication

### 3 x 3 matrices multiplication : **n2** = **9** inner product



*c*13 

23 

*c*33 

*c*

*c*12

22

*c*32

23   21

*b*33  *c*31

*c*

*b*13  *c*11

*b*   *c*

*b*12

22

*b*32

23   21

*a*33  *b*31

*a*31 *a*32

*b*

  *b*

*a*

22

*a*

21



*a*12 *a*13  *b*11

*a*

*a*11

» : inner product **9**

*c*11  *a*11 *b*11  *a*12 *b*21  *a*13 *b*31

### Cumulative multiply-add operation : **n3** = **27** multiply-add

*c*  *c*  *a*  *b*

» : multiply-add

**Initialize C11 = 0**

*c*11  *c*11  *a*11 *b*11  *a*12 *b*21  *a*13 *b*31

##      

9 X 3 multiply-add = **27**

## Pipeline for calculating an inner product :

### Floating point multiplier pipeline : 4 segment

* + Floating point adder pipeline : 4 segment

*C*  *A*1*B*1  *A*2 *B*2  *A*3 *B*3    *Ak Bk*



» after 1st clock input



**Multiplier**

**pipeline**

**Adder**

**pipeline**

**Source B**

**Source A**

**A1B1**

» after 8th clock input



**B5**

**B2 A1B1**

**Multiplier**

**pipeline**

**Adder**

**pipeline**

**Source B**

**Source A**

**6 A5**

**7 A6B**

**A7B**

**A8B8**

**3 A2**

**A3B**

**A4B4**



**B5**

**B2 A1B1**

**Multiplier**

**pipeline**

**Adder**

**pipeline**

Four section summation

**, , ,**  

**Source B**

**Source A**

**6 A5**

**7 A6B**

**A7B**

**A8B8**

**3 A2**

**A3B**

**A4B4**

*C*  *A*1*B*1  *A*5*B*5  *A*9 *B*9  *A*13 *B*13 

 *A*2 *B*2  *A*6 *B*6  *A*10 *B*10  *A*14 *B*14 

 *A*3*B*3  *A*7 *B*7  *A*11*B*11  *A*15 *B*15 

 *A*4 *B*4  *A*8 *B*8  *A*12 *B*12  *A*16 *B*16 

*A*2 *B*2  *A*6 *B*6

*A*1*B*1  *A*5 *B*5

»

» after 4th clock input

» after 9th, 10th, 11th ,...



**1**

**Multiplier**

**pipeline**

**Adder**

**pipeline**

**Source B**

**Source A**

**2 A1B**

**A2B**

**A3B3**

**A4B4**

## Memory Interleaving :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Address bus**  **AR** |  | **AR** |  | **AR** |  | **AR** |  |
|  |  |  |  |  |  |  |  |
| **Memory array** |  | **Memory array** |  | **Memory array** |  | **Memory array** |  |
|  |  |  |  |  |  |  |  |
| **DR** |  | **DR** |  | **DR** |  | **DR** |  |
|  |  |  |  |  |  |  |  |
| **Data bus** |  |  |  |  |  |  |  |

* + *Simultaneous* access to memory from two or more source using *one memory bus system*

### Even / Odd Address Memory Access

* Supercomputer
  + Supercomputer = Vector Instruction + Pipelined floating-point arithmetic
  + Performance Evaluation Index

» **MIPS** : Million Instruction Per Second

» **FLOPS** : Floating-point Operation Per Second

* + - megaflops : 106, gigaflops : 109

### Cray supercomputer : Cray Research

» Clay-1 : 80 megaflops, 4 million 64 bit words memory

» Clay-2 : 12 times more powerful than the clay-1

### VP supercomputer : Fujitsu

» VP-200 : 300 megaflops, 32 million memory, 83 vector instruction, 195 scalar instruction

» VP-2600 : 5 gigaflops