• Parallel Processing
• Pipelining
• Arithmetic Pipeline
• Instruction Pipeline
• RISC Pipeline
• Vector Processing
• Array Processors
PARALLEL PROCESSING

Execution of *Concurrent Events* in the computing process to achieve faster *Computational Speed*

Levels of Parallel Processing

- Job or Program level
- Task or Procedure level
- Inter-Instruction level
- Intra-Instruction level
Architectural Classification

- Flynn's classification
  » Based on the multiplicity of *Instruction Streams* and *Data Streams*
  » Instruction Stream
    • Sequence of Instructions read from memory
  » Data Stream
    • Operations performed on the data in the processor

<table>
<thead>
<tr>
<th>Number of Instruction Streams</th>
<th>Number of Data Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>SISD</td>
</tr>
<tr>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td></td>
<td>SIMD</td>
</tr>
<tr>
<td></td>
<td>MISD</td>
</tr>
<tr>
<td></td>
<td>MIMD</td>
</tr>
</tbody>
</table>
COMPUTER ARCHITECTURES FOR PARALLEL PROCESSING

- Von-Neuman based
  - SISD
    - Superscalar processors
      - Superpipelined processors
        - VLIW
  - MISD
    - Nonexistence
  - SIMD
    - Array processors
      - Systolic arrays
      - Associative processors
  - MIMD
    - Shared-memory multiprocessors
      - Bus based
        - Crossbar switch based
        - Multistage IN based
      - Message-passing multicomputers
        - Hypercube
        - Mesh
        - Reconfigurable

- Dataflow

- Reduction
SISD COMPUTER SYSTEMS

Characteristics

- Standard von Neumann machine
- Instructions and data are stored in memory
- One operation at a time

Limitations

Von Neumann bottleneck

Maximum speed of the system is limited by the Memory Bandwidth (bits/sec or bytes/sec)

- Limitation on Memory Bandwidth
- Memory is shared by CPU and I/O
SISD PERFORMANCE IMPROVEMENTS

- Multiprogramming
- Spooling
- Multifunction processor
- Pipelining
- Exploiting instruction-level parallelism
  - Superscalar
  - Superpipelining
  - VLIW (Very Long Instruction Word)
Characteristics

- There is no computer at present that can be classified as MISD
Characteristics

- Only one copy of the program exists
- A single controller executes one instruction at a time
MIMD COMPUTER SYSTEMS

Characteristics

- Multiple processing units
- Execution of multiple instructions on multiple data

Types of MIMD computer systems

- Shared memory multiprocessors
- Message-passing multicomputers
**PIELING**

A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a partial dedicated segment that operates concurrently with all other segments.

\[ A_i \times B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7 \]

**Segment 1**
- \( A_i \)
- \( B_i \)
- Memory
- \( C_i \)

**Segment 2**
- Multiplier
- \( R3 \)
- \( R4 \)

**Segment 3**
- Adder
- \( R5 \)

**Instructions**
- \( R1 \leftarrow A_i \)
- \( R2 \leftarrow B_i \)
- Load \( A_i \) and \( B_i \)
- \( R3 \leftarrow R1 \times R2 \)
- \( R4 \leftarrow C_i \)
- Multiply and load \( C_i \)
- \( R5 \leftarrow R3 + R4 \)
- Add
### OPERATIONS IN EACH PIPELINE STAGE

<table>
<thead>
<tr>
<th>Clock Pulse Number</th>
<th>Segment 1</th>
<th>Segment 2</th>
<th>Segment 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
<tr>
<td>1</td>
<td>A1</td>
<td>B1</td>
<td>A1 * B1</td>
</tr>
<tr>
<td>3</td>
<td>A3</td>
<td>B3</td>
<td>A3 * B3</td>
</tr>
<tr>
<td>4</td>
<td>A4</td>
<td>B4</td>
<td>A4 * B4</td>
</tr>
<tr>
<td>5</td>
<td>A5</td>
<td>B5</td>
<td>A5 * B5</td>
</tr>
<tr>
<td>6</td>
<td>A6</td>
<td>B6</td>
<td>A6 * B6</td>
</tr>
<tr>
<td>7</td>
<td>A7</td>
<td>B7</td>
<td>A7 * B7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GENERAL PIPELINE

General Structure of a 4-Segment Pipeline

Clock

Input

S1 → R1 → S2 → R2 → S3 → R3 → S4 → R4

Space-Time Diagram

<table>
<thead>
<tr>
<th>Segment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clock cycles
Pipelining and Vector Processing

PIPEDLINE SPEEDUP

n: Number of tasks to be performed

Conventional Machine (Non-Pipelined)
- \( t_n \): Clock cycle
- \( \tau_1 \): Time required to complete the n tasks
- \( \tau_1 = n \times t_n \)

Pipelined Machine (k stages)
- \( t_p \): Clock cycle (time to complete each suboperation)
- \( \tau_k \): Time required to complete the n tasks
- \( \tau_k = (k + n - 1) \times t_p \)

Speedup
- \( S_k \): Speedup
- \( S_k = \frac{n \times t_n}{(k + n - 1) \times t_p} \)

\[
\lim_{{n \to \infty}} S_k = \frac{t_n}{t_p}
\]
Example
- 4-stage pipeline
- suboperation in each stage; \( t_p = 20\text{nS} \)
- 100 tasks to be executed
- 1 task in non-pipelined system; \( 20 \times 4 = 80\text{nS} \)

Pipelined System
\((k + n - 1)t_p = (4 + 99) \times 20 = 2060\text{nS} \)

Non-Pipelined System
\(n \times k \times t_p = 100 \times 80 = 8000\text{nS} \)

Speedup
\(S_k = \frac{8000}{2060} = 3.88 \)

4-Stage Pipeline is basically identical to the system with 4 identical function units

\[ \begin{array}{cccc}
  I_i & I_{i+1} & I_{i+2} & I_{i+3} \\
  \downarrow & \downarrow & \downarrow & \downarrow \\
  P_1 & P_2 & P_3 & P_4 \\
\end{array} \]
Floating-point adder

\[ X = A \times 2^a \]
\[ Y = B \times 2^b \]

1. Compare the exponents
2. Align the mantissa
3. Add/sub the mantissa
4. Normalize the result

**Segment 1:** Compare exponents by subtraction

**Segment 2:** Choose exponent

**Segment 3:** Add or subtract mantissas

**Segment 4:** Adjust exponent

**Mantissas**

A

B

**Exponents**

a

b

Difference

R

R

R

R

R

Align mantissa

Add or subtract mantissas

Normalize result

R

R

R
INSTRUCTION CYCLE

Six Phases* in an Instruction Cycle

[1] Fetch an instruction from memory
[2] Decode the instruction
[3] Calculate the effective address of the operand
[4] Fetch the operands from memory
[5] Execute the operation
[6] Store the result in the proper place

* Some instructions skip some phases
* Effective address calculation can be done in the part of the decoding phase
* Storage of the operation result into a register is done automatically in the execution phase

===> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory
[2] DA: Decode the instruction and calculate the effective address of the operand
[3] FO: Fetch the operand
[4] EX: Execute the operation
Execution of Three Instructions in a 4-Stage Pipeline

Conventional

\[
\begin{array}{cccc}
& & & \\
& & & \\
i & FI & DA & FO & EX \\
i+1 & FI & DA & FO & EX & \\
i+2 & FI & DA & FO & EX \\
\end{array}
\]

Pipelined

\[
\begin{array}{cccc}
& & & \\
& & & \\
i & FI & DA & FO & EX \\
i+1 & FI & DA & FO & EX & \\
i+2 & FI & DA & FO & EX \\
\end{array}
\]
INSTRUCTION EXECUTION IN A 4-STAGE PIPELINE

Segment 1: Fetch instruction from memory

Segment 2: Decode instruction and calculate effective address

Branch? yes

Segment 3: Fetch operand from memory

no

Segment 4: Execute instruction

Interrupt handling yes

Interrupt? yes

Update PC

Empty pipe

Step: 1 2 3 4 5 6 7 8 9 10 11 12 13

Instruction

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
MAJOR HAZARDS IN PIPELINED EXECUTION

**Structural hazards (Resource Conflicts)**

Hardware Resources required by the instructions in simultaneous overlapped execution cannot be met.

**Data hazards (Data Dependency Conflicts)**

An instruction scheduled to be executed in the pipeline requires the result of a previous instruction, which is not yet available.

R1 <- B + C
R1 <- R1 + 1

**Control hazards**

Branches and other instructions that change the PC make the fetch of the next instruction to be delayed.

Hazards in pipelines may make it necessary to **stall** the pipeline.

Pipeline Interlock: Detect Hazards Stall until it is cleared.
### RISC PIPELINE

**RISC**

- Machine with a very fast clock cycle that executes at the rate of one instruction per cycle
- Simple Instruction Set
  - Fixed Length Instruction Format
  - Register-to-Register Operations

### Instruction Cycles of Three-Stage Instruction Pipeline

**Data Manipulation Instructions**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>A</td>
<td>Decode, Read Registers, ALU Operations</td>
</tr>
<tr>
<td>E</td>
<td>Write a Register</td>
</tr>
</tbody>
</table>

**Load and Store Instructions**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>A</td>
<td>Decode, Evaluate Effective Address</td>
</tr>
<tr>
<td>E</td>
<td>Register-to-Memory or Memory-to-Register</td>
</tr>
</tbody>
</table>

**Program Control Instructions**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Instruction Fetch</td>
</tr>
<tr>
<td>A</td>
<td>Decode, Evaluate Branch Address</td>
</tr>
<tr>
<td>E</td>
<td>Write Register(PC)</td>
</tr>
</tbody>
</table>
### DELAYED LOAD

LOAD: \( R1 \leftarrow M[\text{address 1}] \)
LOAD: \( R2 \leftarrow M[\text{address 2}] \)
ADD: \( R3 \leftarrow R1 + R2 \)
STORE: \( M[\text{address 3}] \leftarrow R3 \)

#### Three-segment pipeline timing

Pipeline timing with data conflict

<table>
<thead>
<tr>
<th>clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load R1</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load R2</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add R1+R2</td>
<td>I</td>
<td>A</td>
<td><strong>E</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store R3</td>
<td>I</td>
<td>A</td>
<td><strong>E</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Pipeline timing with delayed load

<table>
<thead>
<tr>
<th>clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load R1</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load R2</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add R1+R2</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store R3</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data dependency is taken care by the compiler rather than the hardware.
**DELAYED BRANCH**

Compiler analyzes the instructions before and after the branch and rearranges the program sequence by inserting useful instructions in the delay steps.

### Using no-operation instructions

<table>
<thead>
<tr>
<th>Clock cycles:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Increment</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Add</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Subtract</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Branch to X</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. NOP</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. NOP</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Instr. in X</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Rearranging the instructions

<table>
<thead>
<tr>
<th>Clock cycles:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Increment</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Branch to X</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Add</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Subtract</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Instr. in X</td>
<td>I</td>
<td>A</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vector Processing Applications

- Problems that can be efficiently formulated in terms of vectors
  - 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 Processor (computer)

Ability to process vectors, and related data structures such as matrices and multi-dimensional arrays, much faster than conventional computers

Vector Processors may also be pipelined
VEVECTOR PROGRAMMING

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

Conventional computer

Initialize I = 0
20 Read A(I)
Read B(I)
Store C(I) = A(I) + B(I)
Increment I = I + 1
If I ≤ 100 goto 20

Vector computer

C(1:100) = A(1:100) + B(1:100)
### VECTOR INSTRUCTIONS

<table>
<thead>
<tr>
<th>Type</th>
<th>Mnemonic</th>
<th>Description (I = 1, ..., n)</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>VSQR</td>
<td>Vector square root</td>
<td>B(I) * SQR(A(I))</td>
</tr>
<tr>
<td></td>
<td>VSIN</td>
<td>Vector sine</td>
<td>B(I) * sin(A(I))</td>
</tr>
<tr>
<td></td>
<td>VCOM</td>
<td>Vector complement</td>
<td>A(I) * \overline{A(I)}</td>
</tr>
<tr>
<td>f2</td>
<td>VSUM</td>
<td>Vector summation</td>
<td>S * Σ A(I)</td>
</tr>
<tr>
<td></td>
<td>VMAX</td>
<td>Vector maximum</td>
<td>S * max{A(I)}</td>
</tr>
<tr>
<td>f3</td>
<td>VADD</td>
<td>Vector add</td>
<td>C(I) * A(I) + B(I)</td>
</tr>
<tr>
<td></td>
<td>VMPY</td>
<td>Vector multiply</td>
<td>C(I) * A(I) * B(I)</td>
</tr>
<tr>
<td></td>
<td>VAND</td>
<td>Vector AND</td>
<td>C(I) * A(I) \cdot B(I)</td>
</tr>
<tr>
<td></td>
<td>VLAR</td>
<td>Vector larger</td>
<td>C(I) * max(A(I),B(I))</td>
</tr>
<tr>
<td></td>
<td>VTGE</td>
<td>Vector test &gt;</td>
<td>C(I) * 0 if A(I) &lt; B(I)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C(I) * 1 if A(I) &gt; B(I)</td>
</tr>
<tr>
<td>f4</td>
<td>SADD</td>
<td>Vector-scalar add</td>
<td>B(I) * S + A(I)</td>
</tr>
<tr>
<td></td>
<td>SDIV</td>
<td>Vector-scalar divide</td>
<td>B(I) * A(I) / S</td>
</tr>
</tbody>
</table>
VECTOR INSTRUCTION FORMAT

Vector Instruction Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Base address source 1</th>
<th>Base address source 2</th>
<th>Base address destination</th>
<th>Vector length</th>
</tr>
</thead>
</table>

Pipeline for Inner Product

![Pipeline Diagram]

- Source A
- Source B
- Multiplier pipeline
- Adder pipeline
MULTIPLE MEMORY MODULE AND INTERLEAVING

Multiple Module Memory

Address Interleaving

Different sets of addresses are assigned to different memory modules.