



SIMATS
ENGINEERING



SIMATS
Saveetha Institute of Medical And Technical Sciences
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

CSA 1220 – COMPUTER ARCHITECTURE FOR FUTURE ENGINEERS

EVALUATING THE PERFORMANCE OF DIVERSE COMPUTER ARCHITECTURE

Guided by

DR. T. KUMARAGURUBARAN

DR. S. SENTHILVADIVU

PRESENTED BY

CHETHRIKA SRI:(192511112)

JANANI J:(192512155)

SHAIK ARSHIYA:(192525332)



ABSTRACT

This study compares the performance of different computer architectures like RISC, CISC, VLIW, and multi-core systems. It analyzes speed, memory access, instruction handling, and power efficiency. Each architecture has strengths depending on the workload—RISC is fast and simple, CISC handles complex tasks well, VLIW depends on compiler efficiency, and multi-core excels in parallel processing. The conclusion shows that no single architecture is best for all scenarios; the choice depends on application needs and system design.



Classification of Computer Architecture

Computer architecture can be classified by multiple ways , based on

- Instruction set
- Data processing
- Memory organization
- Processing Technique
- Parallelism.

1. Based on Instruction Set Architecture

- **CISC (Complex Instruction Set Computer)**
 - Many instructions, including complex operations.
 - Variable-length instructions.
 - Examples: Intel x86, VAX.
- **RISC (Reduced Instruction Set Computer)**
 - Fewer instructions, optimized for speed.
 - Fixed-length instructions.
 - Examples: ARM, MIPS, SPARC.
- **VLIW (Very Long Instruction Word)**
 - Packs multiple operations into a single long instruction.
 - Exploits instruction-level parallelism.
 - Example: Intel Itanium.

2. Based on Number of Instructions Executed in Parallel

SISD (Single Instruction, Single Data)

- Classic sequential computer.
- Executes one instruction on one data at a time.
- Example: Basic uniprocessor.

SIMD (Single Instruction, Multiple Data)

- Executes same instruction on multiple data elements simultaneously.
- Example: Vector processors, GPUs.

MISD (Multiple Instruction, Single Data)

- Rare; multiple instructions operate on the same data.
- Mostly theoretical / fault-tolerant systems.

MIMD (Multiple Instruction, Multiple Data)

- Multiple processors execute different instructions on different data.
- Examples: Multicore CPUs, distributed systems.

3. Based on Memory Organization

- **Von Neumann Architecture**
 - Single memory for instructions and data.
 - Instructions and data share the same bus → Von Neumann bottleneck.
- **Harvard Architecture**
 - Separate memory for instructions and data.
 - Allows simultaneous instruction and data fetch.
 - Example: Microcontrollers like PIC, ARM Cortex-M series

4. Based on Processing Technique

Scalar Processor

- Executes one operation per clock cycle.

Vector Processor

- Executes operations on vectors (arrays) of data per instruction.

Superscalar Processor

- Executes multiple instructions per clock cycle using multiple pipelines.

Pipeline Architecture

- Overlaps instruction execution stages to improve throughput.
- Example: Classic 5-stage MIPS pipeline.

5. Based on Parallelism Level

- **Uniprocessor** – Single CPU core.
- **Multiprocessor** – Multiple CPU cores on same system.
- **Cluster / Distributed Systems** – Multiple computers working together.
- **GPU / Many-core Processor** – Hundreds/thousands of cores for parallel data processing.

How It Works

Asks user about:

- Number of instructions.
- Fixed or variable instruction length.
- Presence of complex instructions.
- Support for multiple operations per instruction

Uses simple rules to classify ISA:

- RISC: Small instruction set, fixed length, simple instructions.
- CISC: Large instruction set, variable length, complex instructions.
- VLIW: Multiple operations in a single instruction.

1. Role of Running Time in Architecture Classification

Running time (execution time) is the **actual performance** of a program on a processor. It depends on:

$\text{Execution Time} = \text{IC} \times \text{CPI} \times \text{Clock Cycle Time}$
 $\text{Execution Time} = \text{IC} \times \text{CPI} \times \text{Clock Cycle Time}$

Where:

- **IC** = Instruction Count (depends on ISA and compiler)
- **CPI** = Cycles per Instruction (depends on processor design)
- **Clock Cycle Time** = Hardware speed

By measuring running time for **same program or benchmark**, we can compare **different architectures**:

- RISC vs CISC
- Single-core vs Multi-core
- Superscalar vs Scalar

2. Examples of Architecture Classification Using Performance

RISC vs CISC

- RISC: Smaller IC but lower CPI → faster for simple instructions
- CISC: Larger IC, may have higher CPI → running time may vary depending on instruction mix

Superscalar / Pipelined vs Scalar

- Superscalar: Can execute multiple instructions per cycle → shorter running time
- Scalar: Executes one instruction per cycle → longer running time

Single-core vs Multi-core

- Multi-core can execute parallel threads → reduced running time for parallel programs

3. Limitations

- Running time alone cannot define architecture type.
Example: Two RISC processors may have different running times because of clock frequency or memory system.
- Running time is application-dependent. Different programs may favour different architectures.
- To classify architecture truly, you also need:
 - Instruction set type (CISC/RISC/VLIW)
 - Memory organization (Von Neumann)

DEMO EXECUTION

- However, we can write a C program to compare “architecture performance” using benchmarking, i.e., measuring the execution time of the same computation on different machines/architectures. This is commonly used to compare architectures in practice.
- Here’s an example C program that measures execution time for a CPU-intensive task:

```
#include <stdio.h>
```

```
#include <time.h>
```

```
#define SIZE 100000000 // Large number for benchmarking
```

```
int main() {
```

```
    clock_t start, end;
```

```
    double cpu_time_used;
```

```
    long long sum = 0;
```

```
printf("Benchmarking CPU...\n");
```

```
start = clock(); // Start time
```

```
// CPU-intensive task (simple sum loop)
```

```
for (long long i = 1; i <= SIZE; i++) {
```

```
    sum += i;
```

```
}
```

```
end = clock(); // End time
```

```
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

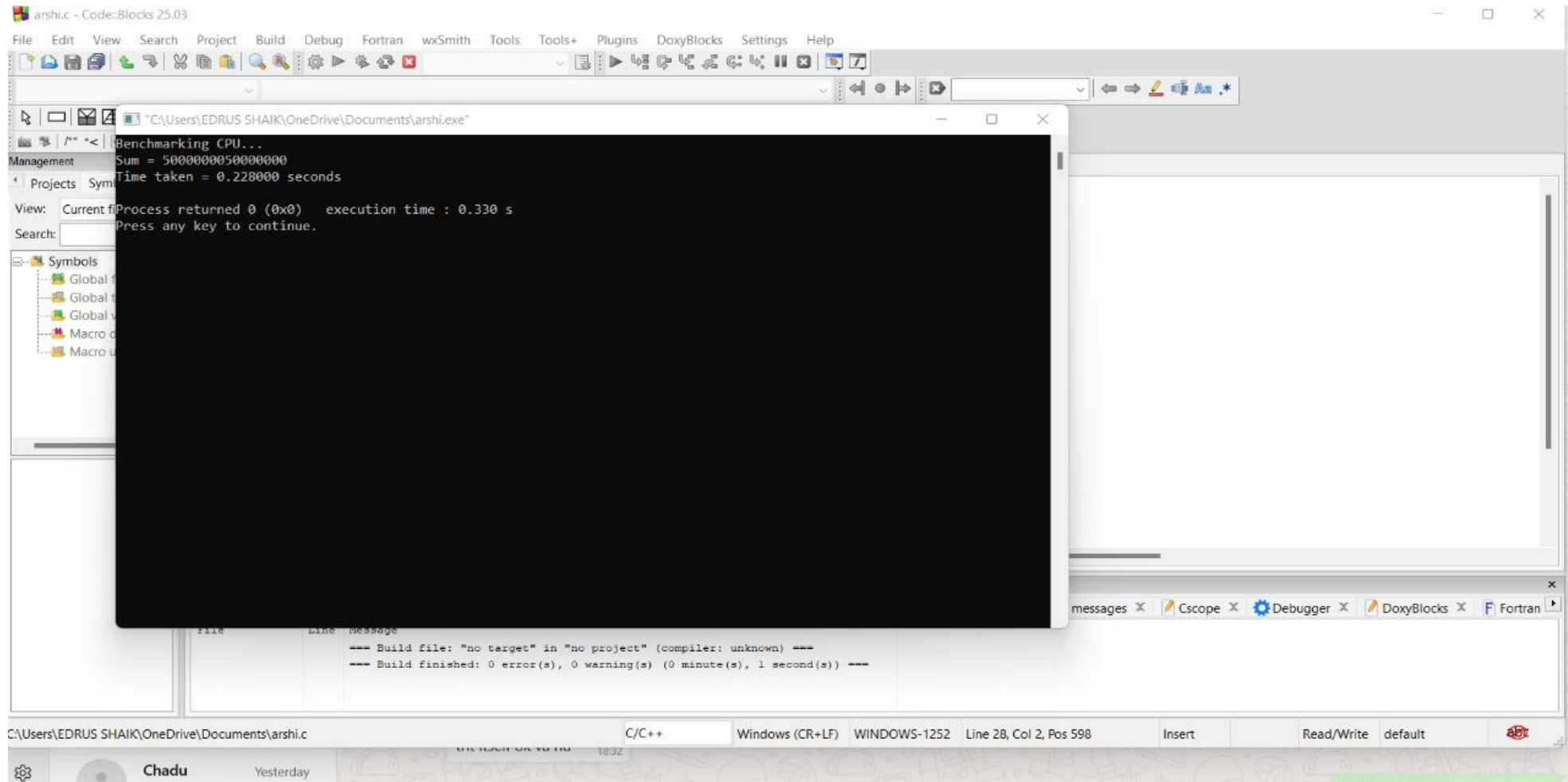
```
printf("Sum = %lld\n", sum);
```

```
printf("Time taken = %f seconds\n", cpu_time_used);
```

```
return 0;
```

```
}
```


OUTPUT



The screenshot displays the Code::Blocks IDE interface. A console window titled "C:\Users\EDRUS SHAIK\OneDrive\Documents\arshi.exe" is open, showing the following output:

```
Benchmarking CPU...  
Sum = 5000000050000000  
Time taken = 0.228000 seconds  
Process returned 0 (0x0)   execution time : 0.330 s  
Press any key to continue.
```

The background shows the IDE's main window with a menu bar (File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, Help) and a toolbar. The left sidebar contains a 'Management' pane with 'Projects' and 'Sym' tabs, and a 'Symbols' pane showing a tree view of global and macro symbols. The bottom status bar indicates the file path "C:\Users\EDRUS SHAIK\OneDrive\Documents\arshi.c", the language "C/C++", and the current line and column "Line 28, Col 2, Pos 598".

At the bottom of the IDE, a build messages pane shows the following output:

```
--- Build file: "no target" in "no project" (compiler: unknown) ---  
--- Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ---
```

CONCLUSION

In conclusion, the performance analysis of different computer architectures reveals how design choices—such as cache organization, pipeline depth, and instruction set architecture—directly impact system efficiency, speed, and responsiveness. Through simulation modules like Cache Memory and Instruction Pipeline, users can visualize and quantify these effects, gaining deeper insights into architectural trade-offs. These tools not only enhance understanding of core computing principles but also empower developers and learners to make informed decisions when designing or selecting systems for specific workloads. Ultimately, such analysis bridges theory and practice, fostering smarter, performance-driven computing solutions.

REFERENCES

- Standard Performance Evaluation Corporation (SPEC). (2023). *SPEC CPU benchmarks*. Retrieved from <https://www.spec.org/cpu/>
- SPEC CPU Benchmark Suite. (2022). Standard Performance Evaluation Corporation. Retrieved from <https://www.spec.org/cpu/>
- Intel Corporation. (2022). *Intel architecture instruction set extensions and performance benchmarks*. Retrieved from <https://www.intel.com>
- ARM Holdings. (2022). *ARM architecture reference manual*. Retrieved from <https://developer.arm.com>
- Hennessy, J. L., & Patterson, D. A. (2019). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann