ELEMENTS OF COMPUTING-II
HYBRID PROCESSOR

A PROJECT

Submitted by

GROUP 14

A P Devanampriya (CB.SC.U4AIE23001)

Mathivanan S (CB.SC.U4AIE23042)

Patel Srikari Shasi (CB.SC.U4AIE23053)

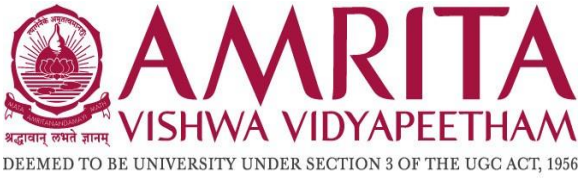
Rohith Balaji T (CB.SC.U4AIE23069)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE(AI)



Centre for Computational Engineering and Networking

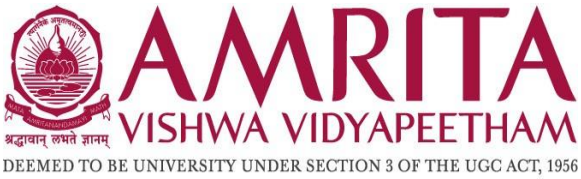
AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

JUNE - 2024

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112



BONAFIDE CERTIFICATE

This is to certify that the thesis entitled “Hybrid Processor” submitted by Group 14 (AIE-A), for the award of the Degree of Bachelor of Technology in the “CSE(AI) ” is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Ms. Sreelakshmi K

Project Guide

Dr. K.P.Soman

Professor and Head CEN

Submitted for the evaluation held on 04-06-2024

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

We, hereby declare that this thesis entitled “Hybrid Processor”, is the record of the original work done by me under the guidance of Ms. Sreelakshmi K, Assistant Professor, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

A P DEVANAMPRIYA **(CB.SC.U4AIE23001)**

MATHIVANAN S **(CB.SC.U4AIE23042)**

PATEL SRIKARI SHASI **(CB.SC.U4AIE23053)**

T ROHITH BALAJI **(CB.SC.U4AIE23069)**

Place: Coimbatore

Date:04-06-2024

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher (MS. SREELAKSHMI K ma'am), who gave us the golden opportunity to do this wonderful project on the topic ("Hybrid Processor"), which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given.

We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

ABSTRACT

This project explores the design and implementation of a hybrid processor that synergizes elements from the ARM (Advanced RISC Machine) architecture with the Harvard architecture. The ARM architecture, renowned for its efficiency and simplicity, is integrated with the Harvard architecture, which is distinguished by its separation of program and data memory. The objective is to leverage the benefits of both architectures to create a processor that offers enhanced performance, efficiency, and versatility for a wide range of computing tasks.

The hybrid processor incorporates the ARM architecture's efficient cache system, which includes multi-level caches designed to reduce access times for frequently used instructions and data. Simultaneously, it utilizes the Harvard architecture's program-data separation to enable concurrent access to instructions and data, thereby improving execution speed and throughput.

This project includes detailed design specifications, focusing on the integration points of the two architectures. It covers the design of critical components such as the instruction fetch unit, data path, and cache hierarchy.

By combining the strengths of ARM and Harvard architectures, this hybrid processor aims to set new standards in performance, efficiency, and versatility, addressing the evolving demands of modern computing applications.

Table of Contents

List of Figures

1. INTRODUCTION

In the realm of processor architecture, innovation often arises from the fusion of established paradigms. Our project delves into the design and development of a hybrid processor, seamlessly blending the renowned ARM (Advanced RISC Machine) architecture with the efficient Harvard architecture. This endeavor seeks to combine the strengths of both architectures, leveraging the cache system typical of ARM processors with the distinct program and data memory separation characteristic of Harvard architecture. Through this synergistic integration, we aim to create a processor that not only excels in performance but also offers versatility in various computing environments.

1.1 Bridging Two Worlds

The ARM architecture is lauded for its simplicity, power efficiency, and widespread use in mobile and embedded systems. Its reduced instruction set computing (RISC) principles allow for streamlined processing and low power consumption, which are critical in battery-powered devices. ARM processors also feature advanced caching systems that significantly enhance performance by reducing the time needed to access frequently used data.

On the other hand, the Harvard architecture distinguishes itself by using separate memory spaces for instructions and data. This separation allows for simultaneous access to both, eliminating the bottleneck commonly found in von Neumann architectures where instructions and data share the same bus. The result is improved execution speed and efficiency, particularly in data-intensive applications where parallel processing is advantageous.

2. METHODOLOGY

2.1.ARM Architecture

ARM (Advanced RISC Machine) architecture is a family of Reduced Instruction Set Computing (RISC) architectures for computer processors, configured for various environments. Developed by ARM Holdings, it is widely known for its power efficiency and has become the dominant architecture in mobile devices, embedded systems, and increasingly in servers and supercomputers.

Key Features of ARM Architecture:

1. RISC Principles

Simplified Instruction Set: ARM architecture follows the RISC (Reduced Instruction Set Computing) principles, which means it uses a simplified set of instructions that are designed to be executed within a single cycle. This contrasts with CISC (Complex Instruction Set Computing) architectures, which use more complex instructions that may take multiple cycles to execute.

Uniform Instruction Length: ARM instructions are of uniform length, typically 32 bits. This uniformity simplifies the decoding process and allows for more predictable and efficient instruction execution.

Load/Store Architecture: ARM architecture uses a load/store model where operations are performed on registers rather than directly on memory. Data must be loaded into registers before being manipulated and then stored back into memory.

2. Instruction Set

Core Instructions: ARM's instruction set includes arithmetic operations (addition, subtraction), logical operations (AND, OR, XOR), data movement (load/store), control flow (branching), and more.

Conditional Execution: ARM instructions can be conditionally executed based on the status of flags in the condition code register. This reduces the need for branch instructions, improving performance and reducing code size.

Thumb and Thumb-2: The Thumb instruction set provides a compressed 16-bit encoding of the 32-bit ARM instructions. Thumb-2 extends this with a mix of 16-bit and 32-bit instructions, offering a balance between code density and performance.

3. Power Efficiency

Optimized for Low Power: ARM processors are designed to be power-efficient, which is essential for battery-powered devices like smartphones, tablets, and IoT devices. Techniques include dynamic voltage and frequency scaling, clock gating, and efficient transistor design.

Reduced Heat Generation: Lower power consumption also results in less heat generation.

4. Thumb Instruction Set

Higher Code Density: Thumb instructions are 16 bits long, compared to the 32-bit ARM instructions. This increases code density, making it possible to fit more instructions into the same memory space, which is beneficial for memory-constrained

applications.

Improved Performance in Constrained Environments: By reducing the memory footprint, Thumb instructions can improve performance in environments with limited memory bandwidth and storage.

5. Scalability

Wide Range of Applications: ARM architecture scales from tiny microcontrollers used in embedded systems to high-performance processors used in servers and supercomputers. This scalability is due to the modularity and flexibility of the ARM design.

Functioning of ARM

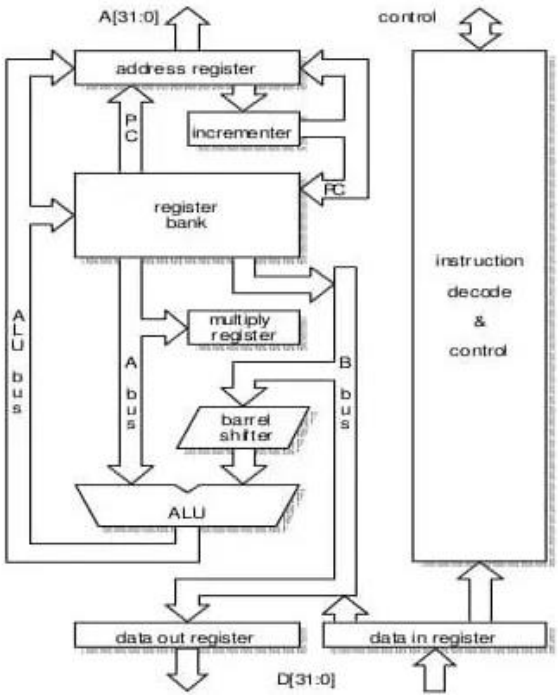


Figure 2.1.1 ARM Architecture

Instruction Fetch:

1.Program Counter (PC): Initially holds the memory address of the next instruction to be executed.

The PC fetches the instruction from memory, and simultaneously, it's incremented to point to the next memory location.

2. Decode and Operand Fetch:
Register Bank: The fetched instruction typically contains information about which registers or memory locations it needs to operate on.
The selected operands are fed into the ALU for the next stage of processing.

3. Execute:
Arithmetic Logic Unit (ALU): Receives the operands and performs the specified operation (e.g., addition, subtraction, logical AND/OR, etc.).
The ALU produces a result based on the operation and operands provided.

4. Memory Access (for Load/Store Operations):
Address Register: Holds the memory address of the data being accessed.
Data In/Data Out Registers: Used for reading data from memory (data in) or writing data to memory (data out).
During load/store operations, data flows between memory and registers.

5. Write Back:
Register Bank: Receives the result from the ALU or data from memory.
The result is written back to the designated register if it's an ALU operation.
If it's a load operation, the data retrieved from memory is stored in the specified register.

6.Control Flow:
Incrementor: Updates the PC to point to the next instruction in memory.
This completes the cycle, and the process begins again with the fetching of the next instruction.

7.Barrel Shifter:

Although not directly involved in the primary data flow, the barrel shifter is essential for performing shift operations on data, such as left shifts or right shifts by a specified number of bits.

It assists in various arithmetic, logical, and bitwise operations, enhancing the capabilities of the ALU.

2.2Harvard Architecture

Harvard architecture is a computer architecture with separate storage and signal pathways for instructions and data. Harvard architecture employs separate buses for instructions and data, allowing them to be fetched simultaneously. This separation of pathways enables more efficient and parallel processing, making Harvard architecture particularly well-suited for embedded systems, digital signal processing, and real-time applications.

Features of Harvard Architecture:

1.Separate Instruction and Data Memory:

In Harvard architecture, instructions and data are stored in physically separate memory units.

2.Parallel Instruction Fetch and Data Access:

The use of separate instruction and data buses allows the CPU to fetch instructions and access data simultaneously. This parallelism enhances the overall throughput and performance of the system, especially in applications requiring frequent

access to both instructions and data.

3.Improved Performance:
By eliminating the need to alternate between fetching instructions and accessing data on a single bus, Harvard architecture can achieve higher performance. This is particularly beneficial for applications with strict timing requirements or high computational demands.

4. Enhanced Security:
The separation of instruction and data memory provides inherent security benefits. For example, Harvard architecture makes it more difficult for certain types of security vulnerabilities, such as buffer overflow attacks, to exploit the system by executing malicious code stored in data memory.

5.Simplified Instruction Pipeline:
Harvard architecture typically features a simplified instruction pipeline, as instructions and data can be fetched independently. This simplification can lead to lower complexity and power consumption in processor design, making it suitable for resource-constrained embedded systems.

6.Reduced Instruction/Data Conflicts:
Since instructions and data are stored in separate memory units, Harvard architecture reduces the likelihood of conflicts between instruction fetches and data accesses. This can lead to more predictable and deterministic behavior in real-time systems, where timing constraints are critical.

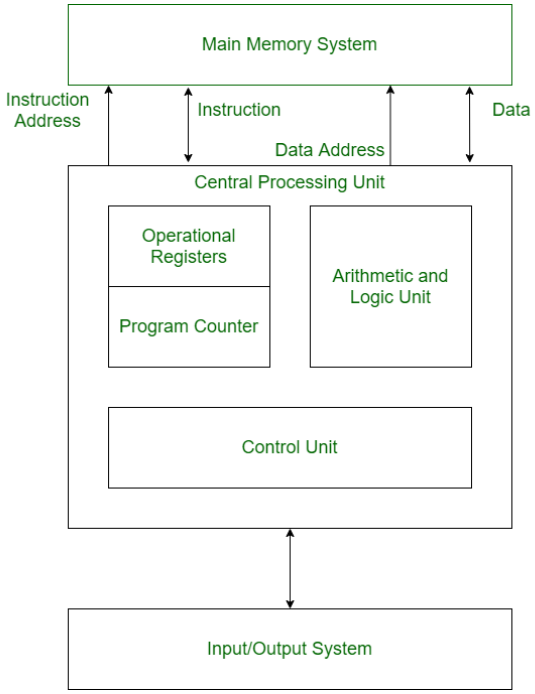


Figure 2.2.1 Harvard Architecture

- Main Memory System: This represents the memory where data and instructions are stored. It serves as a central repository for both program code and data.

- Central Processing Unit (CPU): The CPU is the core processing unit responsible for executing instructions. It consists of several essential components: Operational Registers: These registers temporarily hold data during computation. The ALU (Arithmetic Logic Unit) operates on data from these registers. Program Counter (PC): The PC keeps track of the memory address of the next instruction to be executed. After fetching an instruction, the PC increments to point to the next memory location.

- Arithmetic and Logic Unit (ALU): The ALU performs mathematical operations (addition, subtraction, etc.) and logical operations (AND, OR, etc.). It computes results using data from registers.

- Control Unit: The control unit manages how programs and applications run.
- It allocates system resources and coordinates interactions between different components.
- Input/Output System: The I/O system facilitates communication between the computer and external devices. Input devices (e.g., keyboard, mouse) provide data to the CPU. Output devices (e.g., display, printer) receive processed results from the CPU.
- Data Flow: Instructions and data flow from the memory to the CPU. The CPU processes data using the ALU and registers. Results are either displayed or sent to output devices.

2.3. The Hybrid Processor

The Hybrid Processor implements a novel processor architecture based on Harvard architecture with integrated features inspired by the ARM (Advanced RISC Machine) architecture. The Harvard architecture separates the instruction memory (program memory) from the data memory, enabling simultaneous access to both types of memory. In contrast, ARM processors are known for their efficient cache systems, which reduce memory access latency and enhance overall performance. By combining elements of both architectures, the Hybrid Processor seeks to leverage the advantages of program-data separation and cache-based performance optimization.

The figure given below displays the Hybrid processor architecture.

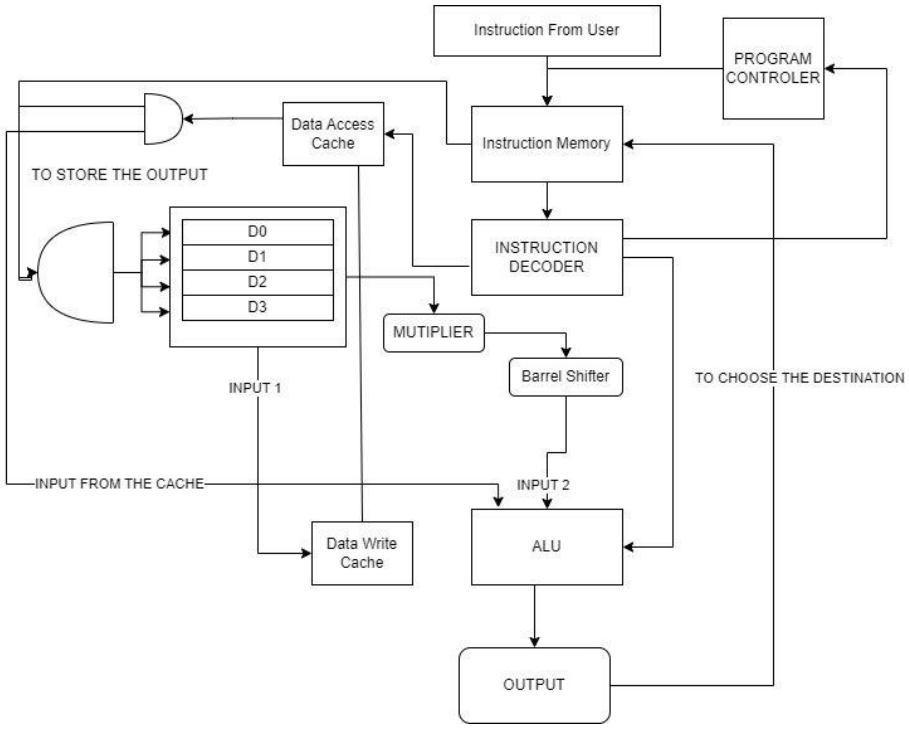


Figure 2.3.1 Hybrid Processor Architecture

1.Program Controller:

The Program Controller manages the execution flow within the processor. It determines which instructions to fetch from memory and how to sequence them.

2.Instruction Memory:

The Instruction Memory stores program instructions.

The Program Controller fetches instructions from here.

3.Instruction Decoder:

The Instruction Decoder interprets fetched instructions and decides the control bits for the ALU.

4.Barrel Shifter:

The Barrel Shifter performs shift operations on data.
It can shift data left or right by a specified number of bits.

6.Arithmetic Logic Unit (ALU):
The ALU is the core component responsible for performing arithmetic (addition, subtraction, etc.) and logical (AND, OR, etc.) operations.
It operates on data received from either data memory or cache. It receives its control bits from the instruction decoder.

7. Data Access Cache and Data Write Cache:
These caches improve memory access speed by not having to search the entire data memory.
The "Data Access Cache" stores frequently accessed data registers for faster retrieval.
The "Data Write Cache" stores frequently used registers to write the data back into memory.

8. Output:
The output from the ALU goes back to the Instruction Memory to get the destination where the output has to be stored.

2.4. Hybrid Processor- Functional Units

Program Counter

The Program Counter (PC) has an Arithmetic Logic Unit (ALU), and multiplexer (Mux). The PC holds the address of the current instruction, while the ALU adds a predetermined value to this address, preparing for the next sequential instruction. When a non-sequential instruction is required, a jump address is introduced, and the Mux selects between the sequential

address and the jump address. This selection is then fed back into the PC, dictating the address of the next instruction to be executed, thereby ensuring the processor executes instructions in the correct order or jumps as dictated by the program logic.

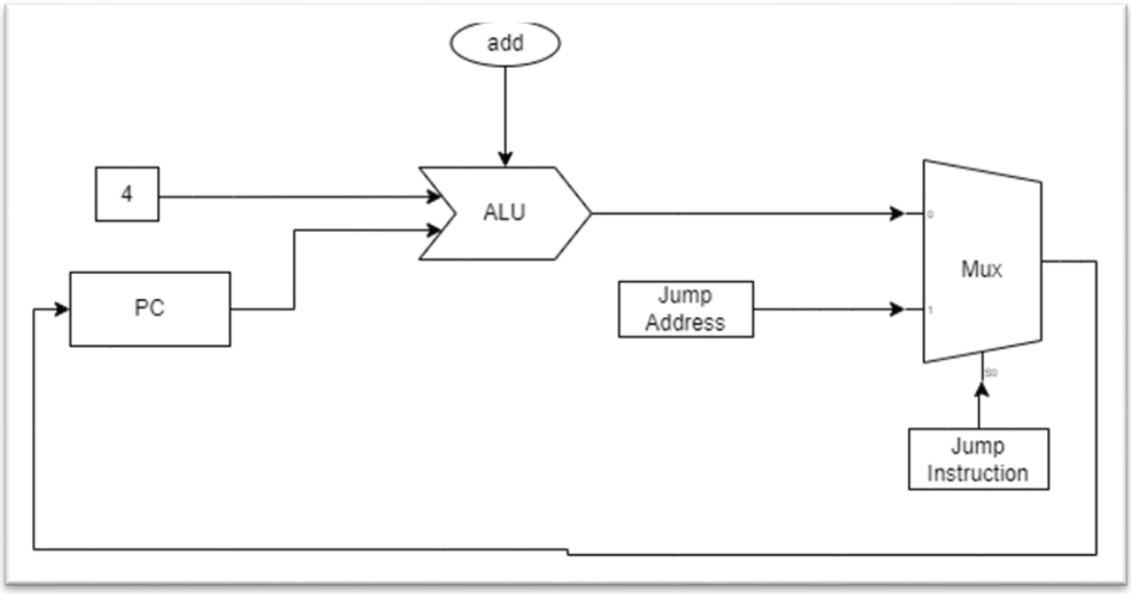


Figure 2.4.1 Hybrid Processor-PC

Cache

The cache memory system is a critical component in modern computing architectures designed to reduce the average time to access data from the main memory. The process initiates with a data request, where the Read/Write Register Address is compared against the cache's directory tags. Upon a successful match, known as a 'hit', the data is swiftly retrieved from the corresponding cache line. Each cache line is associated with status bits indicating its validity and whether it has been modified. In the event of a 'miss', where the requested data is not found in the cache, it is fetched from the main memory and loaded into the cache, updating the directory with the new tag and status bits. This mechanism significantly enhances system performance by

leveraging the temporal and spatial locality of data access patterns. The diagram given below succinctly illustrates the interaction between the cache controller and the cache memory, highlighting the efficiency of data retrieval in a high-speed computing environment.

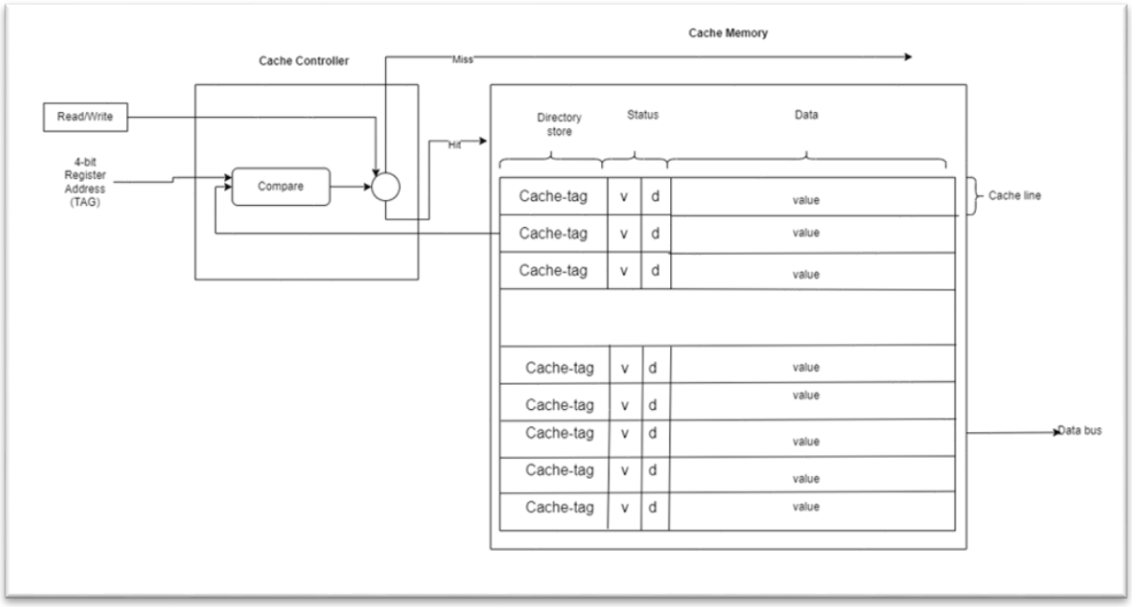


Figure 2.4.2 Hybrid Processor-Cache System

Compare

The Tag represents a portion of the address that uniquely identifies a specific memory block or data segment. It serves as a reference point for comparison.

The Cache-Tag, on the other hand, is a subset of the address that directly interacts with the cache memory.

Tag Comparison: The Tag undergoes a subtraction operation within an Arithmetic Logic Unit (ALU). Specifically, it is subtracted from another tag value obtained from the Cache-Tag.

The result of this subtraction is then fed into an OR gate along with another signal. If either input (the result of the subtraction or the additional signal) is true, the OR gate produces a true output. This signifies a cache miss—i.e., the requested data is not currently stored in the cache.

Cache-Tag Selection: Simultaneously, the Cache-Tag portion of the address enters a demultiplexer (Demux).The Demux selects the appropriate cache line (also known as a cache entry) from the CACHE MEMORY based on the Cache-Tag value. If there is no cache miss (as indicated by the OR gate output being false), the data can be retrieved directly from the selected cache line. This bypasses the need to access the slower DATA MEMORY (main memory). The diagram below illustrates this process.

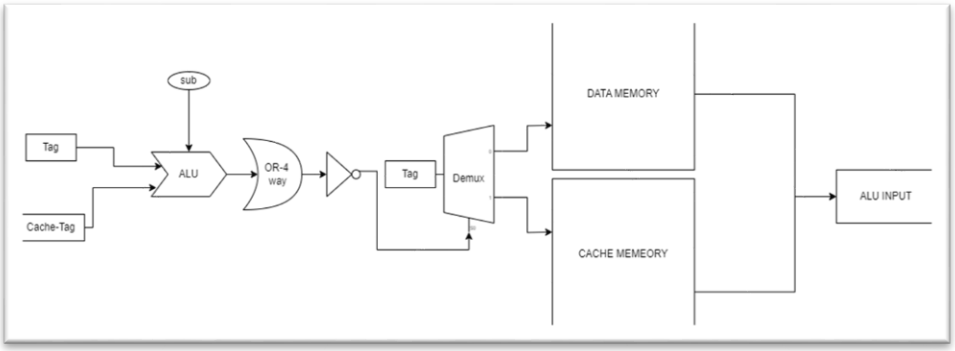


Figure 2.4.3 Hybrid Processor- compare

Writing the Output:

Once the output is arrived after the computing from the Alu it will be stored in the R0 register (0000). The R15 (1111) will take the address of the Rd from the current instruction and the current instruction in the Instruction decoder will change the instruction as it takes the value of the R15 and place it at the INS(13-16) and the data which is stored at the R0 will be moved to the INS(4-12) and the INS(3) will be emptied and the INS(2) will take the value of the “1” to write the instruction and the opcode changed to “01”. The instruction which is changed will take the form of storing the data in the register’s address.

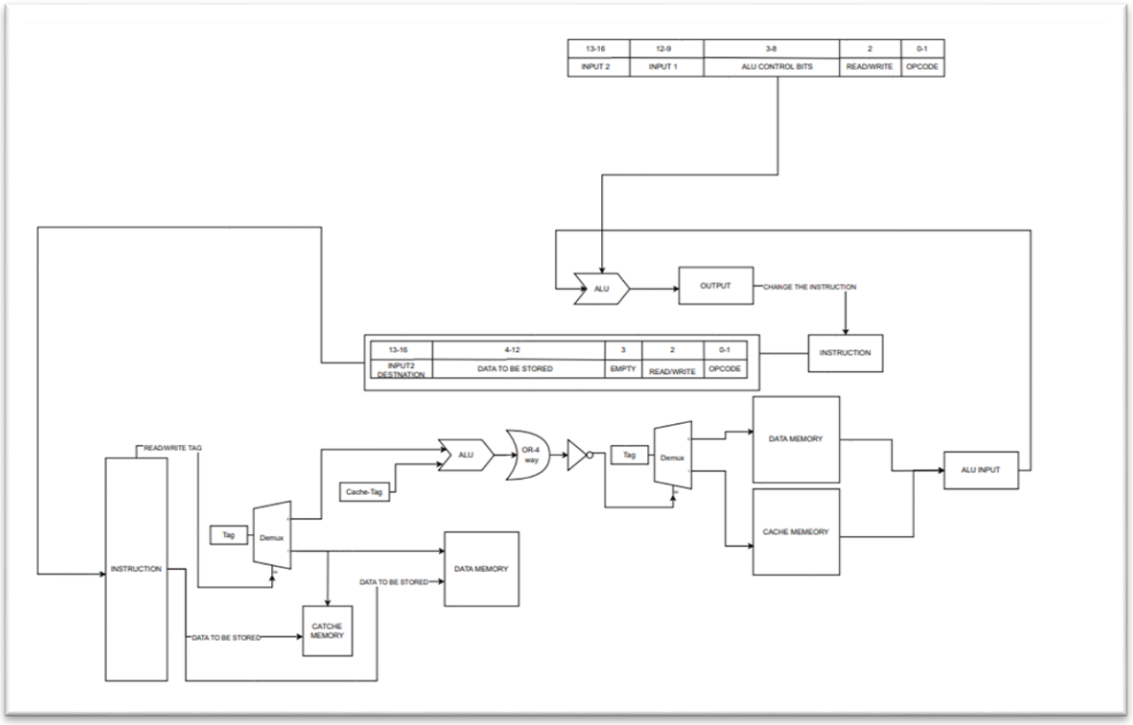


Figure 2.4.4 Hybrid Processor - Writing the Output

Memory

The memory system is a critical component in modern computing architectures. The process initiates with a data request, where the Read/Write Register Address is compared against the data's directory tags. Upon a successful match, known as a 'hit', the data is swiftly retrieved from the corresponding cache line. Each cache line is associated with status bits indicating its validity and whether it has been modified. In the event of a 'miss', where the requested data is not found in that tag, it iterates to the next tag. This mechanism significantly enhances system performance by leveraging the temporal and spatial locality of data access patterns. The diagram given below succinctly illustrates the interaction between the memory controller and the memory, highlighting the efficiency of data

retrieval in a high-speed computing environment.

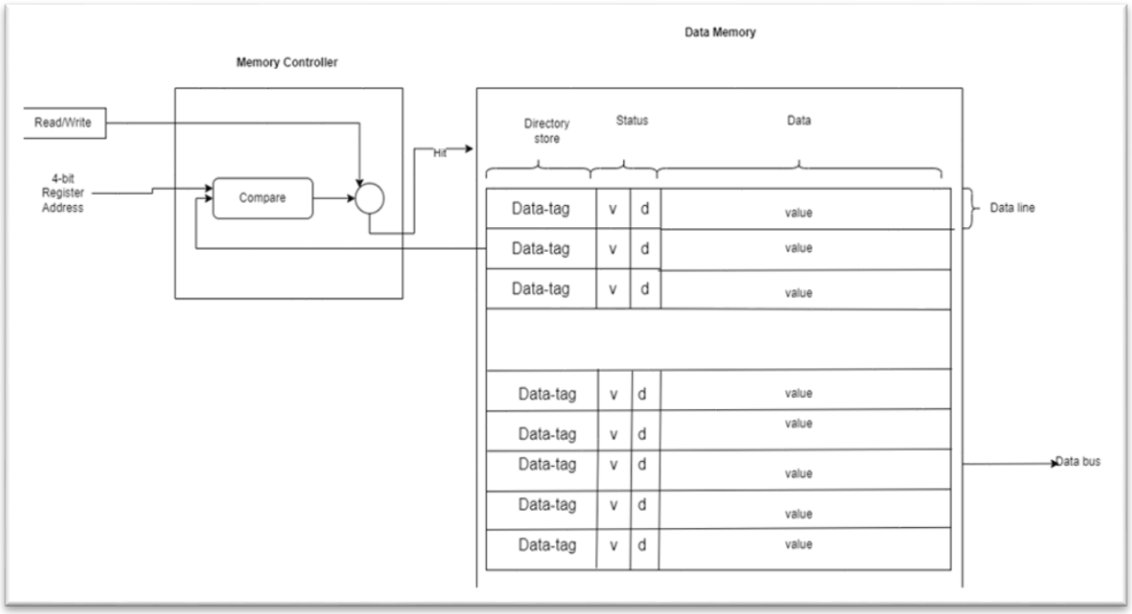


Figure 2.4.5 Hybrid Processor - Memory

ALU

The Hack ALU (Arithmetic Logic Unit) is a critical component of the Hack computer architecture, responsible for executing arithmetic and logical operations. It takes two 16-bit inputs, labeled as x and y, and can perform a variety of functions such as addition, bitwise AND, bitwise OR, negation, and conditional selection based on control bits. The control bits determine the specific operation by specifying whether to zero the inputs, negate them, or compute their sum. The ALU outputs a 16-bit result and two flags: a zero flag, which is set if the output is zero, and a negative flag, which is set if the most significant bit of the output is 1, indicating a negative result in two's complement representation.

The six control bits are zx, nx, zy, ny, f, and no, and their roles are as follows: **zx** (zero the x input): When zx is set to 1, the x input is replaced with 0. If zx is

0, the x input remains unchanged.

nx (negate the x input): When nx is set to 1, the x input is bitwise negated (inverted). If nx is 0, the x input remains as it is (or as zero if zx is 1).

zy (zero the y input): When zy is set to 1, the y input is replaced with 0. If zy is 0, the y input remains unchanged.

ny (negate the y input): When ny is set to 1, the y is bitwise negated (inverted). If ny is 0, the y input remains as it is (or as zero if zy is 1).

f (function code): The f control bit determines whether the ALU performs an arithmetic addition or a bitwise AND operation on the processed x and y inputs. When f is 0, the ALU computes the bitwise AND of the inputs. When f is 1, the ALU computes the arithmetic sum of the inputs.

no (negate the output): When no is set to 1, the output of the ALU operation (either the sum or the AND result) is bitwise negated. If no is 0, the output remains as computed.

In addition to the primary 16-bit output, the Hack ALU produces two single-bit output flags: the zero flag (zr) and the negative flag (ng). These flags provide essential information about the result of the ALU operation:

Zero Flag (zr): This flag is set to 1 if the 16-bit output of the ALU operation is 0. In other words, if all bits of the output are zero, the zr flag will be 1. If the output is any non-zero value, the zr flag will be 0. This flag is useful for conditional operations that depend on whether the result of an operation is zero.

Negative Flag (ng): This flag is set to 1 if the most significant bit (MSB) of the 16-bit output is 1. In a two's complement representation, this indicates that the output is a negative number. If the MSB is 0, indicating a non-negative (either positive or zero) result, the ng flag will be 0. This flag helps in determining the sign of the result, which is crucial for signed arithmetic operations.

The combination of the 16-bit result and the zr and ng flags allows the Hack ALU to support a wide range of computational tasks and conditional operations, providing the necessary information for subsequent processing stages within the

Hack computer architecture.

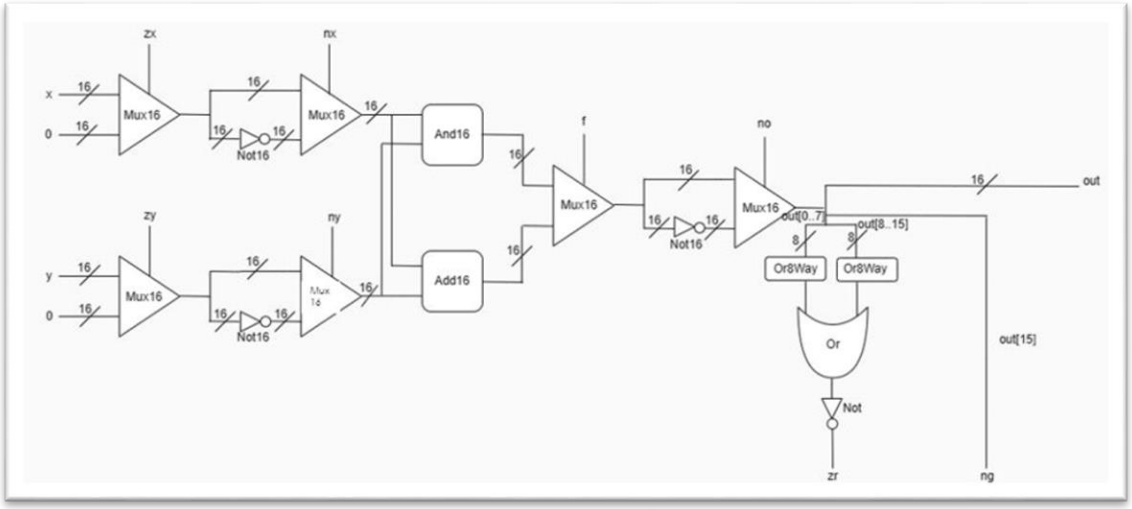


Figure 2.4.6 Hybrid Processor – ALU

2.5. Hybrid Processor- Instruction Set

The Hybrid processor design has four types of instruction sets. Each instruction is 17 bits.

The INS(0-1) bits are reserved for the opcode. The INS(2) is reserved for the reading and writing operation.

- J type
- I type
- R type
- D type

J type:

It contains a set of 4 instructions that help in the jump and branch functions. Where INS(0-12) is reserved for the instruction line it should jump. And the INS(13-16) is for the Branch code.

I type:

It contains INS(3-8) which is reserved for the immediate value that is to be operated and The register address to be operated will be

stored in the INS(9-12). The branch code takes the value from INS(13-16). It contains the 7 operations.

R type:

It contains INS(3-6) reserved for the Rd address and INS(7-10 for the R2 register being operated. The INS(11-16) contains the ALU code for the 4 operations and the remaining four Operations will take the branch code. The shift operations have a special code to differentiate from the other instructions. It contains 10 instructions.

D type instructions:

It will help to return the output value to store the output in the destination. The other instructions are

3.OUTCOMES

Combining the Harvard architecture with an ARM processor to create a hybrid processor yields several significant outcomes. This integration leverages the strengths of both architectures, resulting in improved performance, efficiency, and versatility.

1. Enhanced Performance: The separation of instruction and data memory inherent in the Harvard architecture allows for simultaneous access to instructions and data. This parallelism reduces bottlenecks and increases the overall execution speed of the processor, enhancing performance in both simple and complex applications.

2. Increased Efficiency: By combining the efficient instruction set and power-saving features of the ARM processor with the Harvard architecture's distinct memory pathways, the hybrid processor

achieves greater energy efficiency. This is particularly advantageous in embedded systems and mobile devices where power consumption is a critical concern.

3. Flexibility and Scalability: The ARM processor's modular and scalable architecture, coupled with the Harvard design, allows for flexible implementation across a wide range of applications, from small, low-power devices to high-performance computing systems. This adaptability makes the hybrid processor suitable for diverse use cases, including IoT devices, smartphones, and industrial automation.

4. Improved Memory Management: The Harvard architecture's separate memory spaces for instructions and data simplify memory management and can lead to more optimized memory usage. This separation can enhance security by preventing certain types of attacks that exploit unified memory spaces, adding an extra layer of protection.

5. Advanced Features and Capabilities: Integrating ARM's advanced features such as enhanced security support for SIMD (Single Instruction, Multiple Data) instructions, and efficient handling of complex operations with the Harvard architecture's fast and parallel memory access provides a powerful hybrid processor. This combination supports modern applications requiring high processing power and efficient resource utilization.

Overall, the hybrid processor resulting from this combination offers a robust solution that maximizes the benefits of both the Harvard

architecture and ARM processors, providing high performance, efficient power consumption, flexible application, and enhanced security.

4.CONCLUSION

In conclusion, the design of the Hybrid Processor represents a significant achievement in merging the principles of the Harvard architecture with ARM-inspired features. Through meticulous design specification, component design, the processor demonstrates its capability to offer high performance, efficiency, and versatility. By integrating Harvard architecture's separation of instruction and data memory with ARM's caching mechanisms and pipeline stages, the processor promises to deliver enhanced performance and resource utilization.

The successful implementation of this hybrid processor opens new avenues for addressing the evolving demands of modern computing environments. Its ability to handle simultaneous instruction and data processing efficiently makes it a formidable contender for applications ranging from embedded systems to high-performance computing. The energy-efficient design, coupled with robust security features, ensures that the processor can meet the stringent requirements of various industries, including mobile technology, IoT, and cybersecurity.

Moving forward, further refinements and optimizations can be explored to maximize the processor's potential and address specific application requirements. Areas of focus may include enhancing the processor's adaptability to different workloads, optimizing power consumption for ultra-low-power applications, and integrating advanced features such as machine learning accelerators or specialized cryptographic units. Additionally, ongoing collaboration with the open-source community can provide valuable insights and foster innovation, driving continuous improvement and broader adoption.

BIBLIOGRAPHY

[1] Berkeley, E.C. & Jensen, R.A. (1950). World's Smallest Electric Brain. Radio-Electronics, Oct.1950.

[2] Illinois State University (2000). Little Man Computer. http://www.acs.ilstu.edu/faculty/javila/lmc/

[3] Bell Laboratories Record (1969). Cardboard "Computer" Helps Students, 216.

[4] Yutaka, S. Paper Processor. Available: https://sites.google.com/site/kotukotuzimiti/

[5] Langdon, G.G. (1982). Computer Design. Computeach press, California.

[6] Hennessy, J.L. & Patterson, D.A. (1990). Computer Architecture: A quantitative Approach. Morgan Kaufmann San Mateo CA.

[7] El Aarag, H. (2009). A complete design of a RISC processor for pedagogical purposes. Journal of Computing Sciences in Colleges, Vol. 25, No.2, pp. 205–213.

[8] Rodriguez, B.J. (1994). A minimal TTL processor for

architecture exploration. Proceedings of the 1994 ACM symposium on applied computing, pp. 338–340.

[9] Massachussetts Institute of Technology. MIT 6.004 Beta Architecture Information. Available: http://6004.csail.mit.edu/Spring98/Beta.

[10] Verplaetse, P. & Campenhout, J. (1999). ESCAPE: Environment for the Simulation of Computer Architecture for the Purpose of Education. IEEE Technical Committee on Computer Architecture Newsletter, pp. 57–59.