IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A COMPARATIVE STUDY ON THREE STAGE AND FIVE STAGE PIPELINING OF ARMs

[1]Nimisha Rai,[2]Pallavi Kadam ,[3]Chaitali Tikhe

[1]Assistant Professor

[1] Department of Electronics, Dr. D. Y. Patil ACS College, Pimpri, Pune, India

[2]Assistant Professor

[2]Department of Electronics, Dr. D. Y. Patil ACS College, Pimpri, Pune, India

[3]Assistant Professor

[3]Department of E & TC, Pimpri Chinchwad Polytechnic, Akurdi, Pune, India

Abstract: Pipelining is a critical technique in modern computer architecture that enhances CPU performance by overlapping the execution of instructions. This paper presents a comparative study between three-stage and five-stage pipelining architectures. By examining factors such as throughput, latency, complexity, and hazard management, the study aims to highlight the trade-offs and benefits of each pipeline depth.

Index Terms - Pipelining stages, Architecture, RISC, Throughput, Latency, Complexity, Hazards.

I. INTRODUCTION

Pipelining is a technique used in computer architecture to enhance the throughput of a CPU by dividing the execution process into distinct stages, allowing multiple instructions to be processed simultaneously. This method contrasts with non-pipelined architectures, where each instruction must complete all execution stages before the next instruction begins. Optimizing instruction execution through pipelining is crucial for improving CPU performance. By enabling parallelism within the processor, pipelining increases instruction throughput, reduces instruction latency, and maximizes resource utilization. As the number of pipelining stages is increased it affects the complexity, efficiency, and susceptibility to hazards.

II. PIPELINING IN ARM

ARM is an Advanced RISC Machine. RISC (Reduced instruction set computer) employs pipelining to improve processor efficiency. Pipelining is a design method or procedure that improves the efficiency of data processing by keeping the CPU in a continuous fetching, decoding, and execution process known as (the F&E cycle). Pipelining in ARM boosts execution speed, by applying parallelism in which one instruction is executed while other instructions are being decoded and fetched at the same time. It efficiently uses resources like memory, ALU and CPU to run them continually. The different arm processors have different stages of pipelining. To understand the concept we will have a look how instruction is executed in Non- Pipelined and Pipelined architecture.

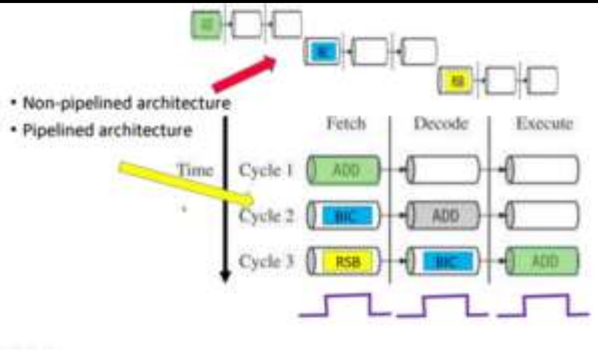


Figure 1: Non-Pipelined and Pipelined Architecture

In Non-Pipelined Architecture instruction is executed in sequence. Instruction 1 is first fetched, decoded and executed then only second instruction is fetched. So each instruction requires three clock cycles to complete it's exection. So three instructions will require nine clock cycles to complete execution. In pipelined architecture parallelism is achieved by overlapping many operations in single clock cycle. As shown in figure 1 in pipelined architecture when first instruction moved to decoding stage, the second instruction is fetched and so on. Hence it will take only five clock cycles to execute three instructions. Assume clock duration of each clock cycle is 1 sec then average time to execute one instruction in non-pipelined architecture is 9/3 = 3 sec whereas as for pipelined architecture is 5/3=1.67sec. In this way pipelining increases overall throughput of the system. Reduces processor cycle time by processing more instruction simultaneously in single clock cycle. Pipelining also allows processor to work at higher frequencies.

III. ARCHITECTURE OF ARM7 AND ARM9

3.1. Three- Stage Pipeline Architecture of ARM 7

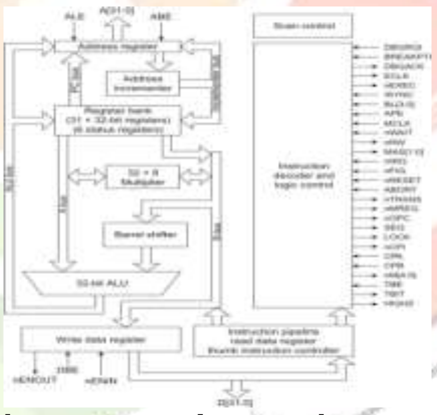


Figure 2: Three stage pipelining architecture of ARM7.

ARM 7 has Von Neumann architecture with a 32 bit data bus, which carries both instruction and data. Only load/ store/swap instruction access data from memory. The data operations are performed on registers instead of memory, uniform and fixed length instruction fields make decoding simpler. ARM7 processor uses three stages pipelining to increase the execution speed by performing many operations simultaneously and keeping CPU and memory to work continuously. In three sage pipelining instructions are executed in three stages.

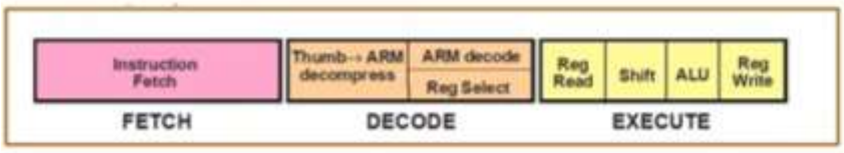


Figure 3: Three Stage Pipelining in ARM 7

1. Fetch: In this cycle instructions are fetched from memory and placed into the instruction register whose address is given by PC. After fetching the instruction PC is incremented by four as length of each instruction is fixed.

2. Decode: In this cycle instruction is decoded to identify which operation is going to be performed and appropriate control signals are generated and register are selected which store the operands.

3. Execute: In this cycle data transfer, arithmetic or logical operation are performed and result is stored in register or memory.

3.2 Five- Stage Pipeline Architecture of ARM 9

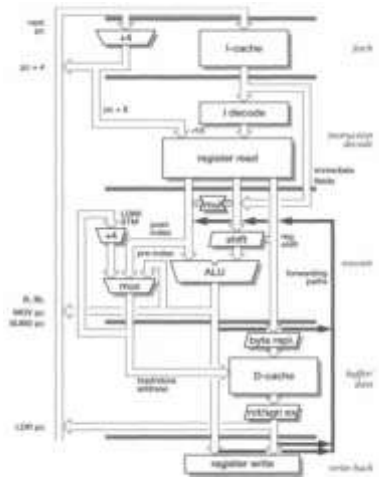


Figure 4: Five stage pipelining architecture of ARM9 processor.

It has a Harvard Architecture. It has separate 4 KB of data and instruction cache memory to reduce execution time with separate data and address bus. It supports 5 stage pipelining fetch, Decode, Execute, Memory and write back.

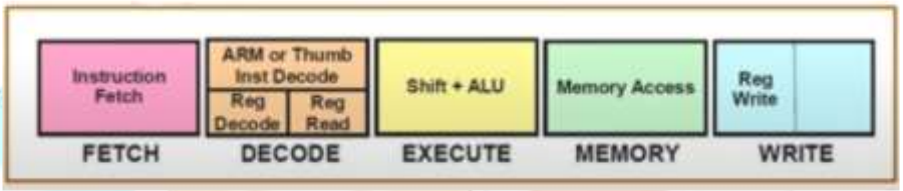


Figure 5: Five stage pipelining of ARM

The five stages of pipeline are

1. Fetch – In this stage the instructions are fetched from the memory and stored in the instruction register.
2. Decode – This stage decodes the instruction to identify that which operation is going to be performed and generates the appropriate control signals and takes the necessary steps for the next execution stage. It also performs register read in this stage.
3. Execute – This stage performs arithmetic and logical operation and generates the result. For memory-related instructions such as load or store, the memory address is calculated by the ALU.
4. Buffer/Data – Data memory is accessed if required. Otherwise, the ALU output is temporarily stored for a single clock cycle.
5. Write back – The result generated by the ALU is written back to the register file, including any data loaded from memory.

IV. HAZARDS IN PIPELINING

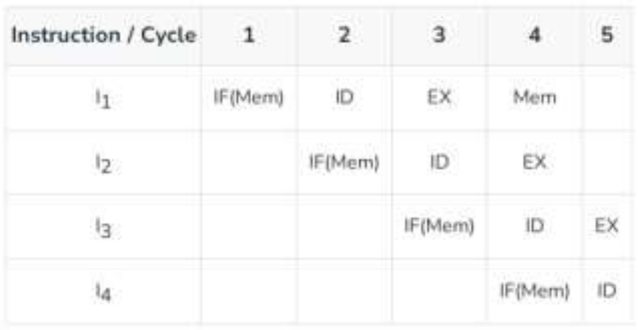
In pipelining there is a possibility that some set of instructions have some type of dependency. Due to these dependencies some instructions cannot be executed in their designated time slot this is called as Hazards in pipelining. There are three types of hazards.

1. Structural Hazards
2. Data Hazards
3. Control Hazards

4.1. Structural Hazards

Structural hazards occur in a pipelined processor when two or more instructions require the same hardware resource at the same time. This resource contention can lead to delays in instruction execution, thereby diminishing the overall efficiency of the pipeline. This type of situation cannot be handled by the hardware in an overlapped pipelined execution.

Example



Instruction / Cycle	1	2	3	4	5
I_1	IF(Mem)	ID	EX	Mem	
I_2		IF(Mem)	ID	EX	
I_3			IF(Mem)	ID	EX
I_4				IF(Mem)	ID

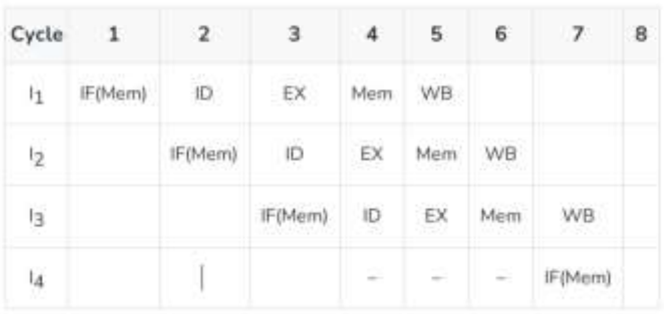
Table 1: Structural Hazard in pipelining

From the table it is observed that in cycle4 Instruction 1 and Instruction4 needs the same resource i.e. memory to complete their operation. This offers resource conflict.

Solution for Structural Hazards

a) Introduce stall in pipelining

- When resource conflict arises the instruction which results in conflict must be delayed and all other subsequent instruction must be delayed until the resource becomes free.
- Due to this delay the more number of clock cycles are required to execute the instruction which decreases CPU efficiency.



Cycle	1	2	3	4	5	6	7	8
I_1	IF(Mem)	ID	EX	Mem	WB			
I_2		IF(Mem)	ID	EX	Mem	WB		
I_3			IF(Mem)	ID	EX	Mem	WB	
I_4				-	-	-	IF(Mem)	

Table 2: Stall in pipelining

From the above table it is observed that I4 is fetched in cycle7 when the resource becomes free.

b) To Increase Structural Resources

1. By increasing the number of pipelining stages it reduces the bourdon of pipelining.
2. Using different data and instruction cache memory.
3. Separate ALU unit must be used for Address generation and floating point arithmetic.
4. By using a register file with multiport access.

4.2. Data Hazards

Data hazards occur when an instruction depends on the result of a previous instruction that is still in the pipeline. If not managed correctly, these dependencies can lead to incorrect results.

Classification is based on the order of READ or WRITE operations on registers:

1.**RAW (Read After Write):** This occurs when an instruction reads data that is produced by a preceding instruction. The dependent instruction must wait until the previous instruction finishes its write operation and stores the data in the register or memory.

2.**WAR (Write After Read):** This less common hazard occurs when a subsequent instruction writes to a register before a previous instruction reads from it. It is typically found in systems with complex and special instructions.

3. **WAW (Write After Write):** This happens when two parallel instructions write to the same register, and their order of execution is important. Ensuring that the second instruction writes its result after the first instruction completes its write operation is essential.

Solution for Data Hazards

1. **Data Forwarding:** This technique involves passing the result of a previous instruction directly to the functional unit that needs it. This makes the result available earlier to the dependent instruction, reducing stalls and improving pipeline efficiency.
2. **Compiler Optimizations:** Compilers can detect data dependencies during code optimization. They may reorder instructions or insert NOP (No Operation) instructions to reduce data hazards when generating executable code.
3. **Register Renaming:** Modern CPUs use register renaming to avoid WAW and WAR hazards. By allocating different physical registers for instructions that use the same logical register, these hazards can be prevented.
4. **Out-of-Order Execution:** Some CPUs employ out-of-order execution, allowing instructions to be executed in a sequence different from their original program order. This enables the processor to execute independent instructions concurrently, reducing stalls caused by data hazards.
5. **Multiple Stages of Pipelines:** CPUs with deeper pipelines can alleviate data hazards by breaking down instruction execution into more stages

4.3 Control Hazard

Control hazard, also known as branch hazards, occurs due to branching instructions such as looping branching or conditional statements in the program. Branching instructions alters the execution of program flow, hence fetching of next instruction is uncertain in the pipelining. Fetching of the next instruction in the pipeline depends upon the result of branching instruction.

Solutions for Control Hazards

1. **Stalling the pipeline:** The pipeline must be stalled as soon as branch instruction is decoded and fetching of further instruction is prevented unless the branch decision is resolved.
2. **Branch Prediction:** Instead of stalling (delaying) , the fetching of further subsequent instructions is continued by predicting the result of branching instruction. If prediction is correct pipelining continues without any disturbance but if prediction is wrong the pipeline must be flushed and correct instructions must be fetched.

V COMPARATIVE ANALYSIS

5.1 Performance of processor

The performance of the processor depends upon the time required to execute a program. It is given by

$$T_{prog} = N_{inst} \times \frac{CPI}{f_{clk}}$$

Where Ninst = Number of instruction in the program

CPI = Average number of clock cycles per instruction

f_{clk} = Processor clock frequency

To increase the performance of the processor we have to either reduce CPI or to increase f_{clk}. In three stage pipelining in the execute stage so many operations has to be performed in a single clock cycle, such as data transfer(memory to register , register to ALU), data processing and result writing. Hence the time period must be sufficiently long hence f_{clk} cannot be increased above a certain limit. In three stage pipelining the execute stage is divided further into memory and write back. Hence each stage has to perform small operations in a single cycle. So we can reduce the time period of one clock cycle and correspondingly increase f_{clk}.

5.2 Throughput

- Three-Stage Pipeline: Limited by fewer stages, resulting in lower instruction throughput.
- Five-Stage Pipeline: Higher throughput due to more stages processing instructions concurrently.

5.3 Latency

- Three-Stage Pipeline: Lower latency as each instruction passes through fewer stages.
- Five-Stage Pipeline: Higher latency since each instruction must pass through more stages.

5.4 Complexity

- Three-Stage Pipeline: Simpler design and control logic.
- Five-Stage Pipeline: Increased complexity in design and control logic due to more stages and the need for efficient hazard management.

5.5 Hazard Management

5.5.1 Data Hazards

- Three-Stage Pipeline: Fewer stages mean fewer opportunities for data hazards.
- Five-Stage Pipeline: More stages increase the likelihood of data hazards, requiring techniques like forwarding and stalling.

5.5.2 Control Hazards

- Three-Stage Pipeline: Easier to manage with fewer stages.
- Five-Stage Pipeline: More challenging due to the increased depth, requiring effective branch prediction and speculative execution.

5.5.3 Structural Hazards

- Three-Stage Pipeline: Fewer stages reduce the likelihood of resource conflicts.
- Five-Stage Pipeline: More stages increase the potential for resource conflicts, necessitating resource duplication or time-multiplexing.

VI CONCLUSION

From the comparative study of three-stage and five-stage pipeline architectures we understand that ARM7 has Von Neumann Architecture while ARM9 has Harvard architecture with separate data and instruction cache. Five-stage pipelining has more CPI as compared to three-stage pipelining, since pipelining reduces the number of clock cycle to execute the instruction. Operating frequency of ARM9 is approximately double that of ARM7. Higher throughput and better resource utilization, it also introduces increased complexity and a greater likelihood of hazards. Three-stage pipelining, on the other hand, provides a simpler and less hazardous design at the cost of reduced throughput. The choice between these pipeline depths depends on the specific requirements and constraints of the application, including performance goals, design complexity, and resource availability.

VII. REFERENCES

[1] Joshi Vaibhav Vijay, Balbhim Bansode, “ARM Processor Architecture” IJSETR, volume 4, Issue 10, October 2015.

[2] Technical Reference Manual, “ARM11 MPCore processor”, copyright 2005, 2006, 2008, ARM limited, ARM DDI 0360E.

[3] RealView, “ARM Architecture Overview”, tools by ARM.

[4] Ashton Raggatt McDougall, “ARM architecture”, retrieved in 2011.

[5] Grisenthwaite, Richard (2011), “ARM v8-A Technology Preview”, Retrieved 31 October 2011.

[6] “Procedure call standard for the ARM Architecture”, ARM holdings. 30 November 2013. Retrieved 27 May 2013.

[7]] K. P. K and V. Prakash A. M, "Designing and Implementation of 32-bit 5 stage Pipelined MIPS based RISC Processor Capable of Resolving Data Hazards," 2021 IEEE International Conference on Mobile Networks and Wireless Communications (ICMNWC), 2021, pp. 1-6, doi: 10.1109/ICMNWC52512.2021.9688435