ARM Processor Design

From Simple to Pipelined Architecture

Project Report

Course:	CIE 439 - Computer Architecture and Assembly Language
Institution:	Zewail City of Science and Technology
Semester:	Fall 2025

Team Members

Mohammed Soliman	202402280
Mohammed Abdelrahman	202300110
Mohamed Ashraf	202401017

December 25, 2025

Abstract

This report presents the design and implementation of three ARM processor architectures: a simple 8-bit multicycle processor (ARM1), a 32-bit single-cycle processor (ARM32), and a 32-bit 5-stage pipelined processor (ARMPIP). Each design demonstrates increasing complexity in computer architecture, from basic accumulator-based computation to advanced hazard detection and forwarding mechanisms. The implementations are verified through comprehensive testbenches in SystemVerilog.

Contents

1 Introduction

This project implements three progressively complex processor architectures to demonstrate fundamental concepts in computer organization and design:

1. **ARM1**: A simple 8-bit multicycle processor with accumulator architecture
2. **ARM32**: A 32-bit single-cycle ARM processor supporting data processing and memory operations
3. **ARMPIP**: A 32-bit 5-stage pipelined ARM processor with full hazard handling

Design Goals:

- Demonstrate the evolution from simple to complex processor architectures
- Implement core ARM instruction set including data processing, memory access, and branching
- Handle pipeline hazards through forwarding, stalling, and flushing mechanisms
- Provide comprehensive verification through simulation testbenches

2 ARM1: Simple Multicycle Processor

2.1 Architecture Overview

ARM1 is an 8-bit accumulator-based processor with a 4-state finite state machine control unit. It uses a unified memory for both instructions and data, with a 4-bit address space (16 locations).

2.2 Datapath Components

Table 1: ARM1 Datapath Components

Module	**Width**	**Description**
PC	4-bit	Program Counter
IR	8-bit	Instruction Register (4-bit opcode + 4-bit address)
AC	8-bit	Accumulator Register
B	8-bit	Operand B Register
O	8-bit	Output Register
ALU	8-bit	Arithmetic Logic Unit
Memory	8-bit × 16	Unified instruction/data memory

2.3 Instruction Set

Table 2: ARM1 Instruction Set

Opcode	Binary	Mnemonic	Operation
0000	0x0	ADD	AC ← AC + B
0001	0x1	SUB	AC ← AC - B
0010	0x2	OR	AC ← AC | B
0011	0x3	AND	AC ← AC & B
1010	0xA	OUT	O ← AC
1100	0xC	LDA	AC ← Mem[addr]
1101	0xD	LDB	B ← Mem[addr]
1110	0xE	STR	Mem[addr] ← AC
1111	0xF	HLT	Halt execution

2.4 Control Unit State Machine

The control unit implements a 4-state FSM:

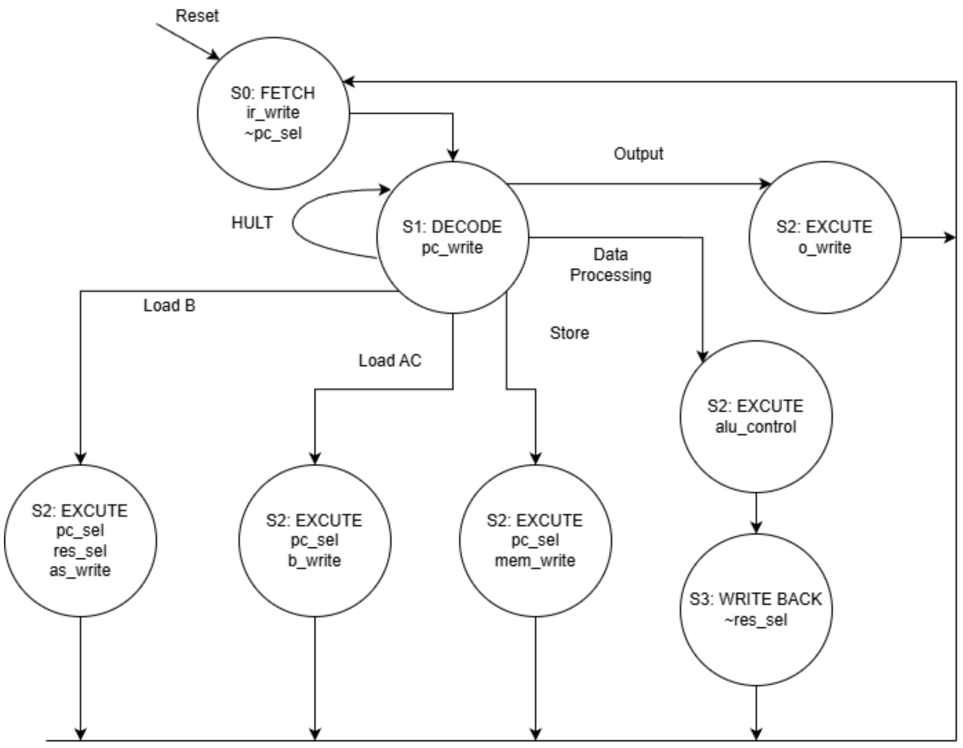


Figure 1: ARM 1 FSM

State Descriptions:

- **S0 (Fetch)**: Read instruction from Mem[PC] into IR
- **S1 (Decode)**: Decode opcode, increment PC
- **S2 (Execute)**: Perform operation (memory access, ALU, or I/O)
- **S3 (Writeback)**: Write ALU result back to AC (data processing only)

2.5 Simulation Output

```
# Time | State | PC | Opcode | Addr | AC   | B    | O    | Operation
# -------------------------------------------------------------------
# Time: 25 | Reset released
#
#   31 | S1    | 00 | LDA   | c    | -- | -- | -- | Decode
#   51 | S2    | 01 | LDA   | c    | 00 | 00 | 00 | Load AC
#   71 | S0    | 01 | LDA   | c    | -- | -- | -- | Fetch
#   91 | S1    | 01 | LDB   | d    | -- | -- | -- | Decode
#  111 | S2    | 02 | LDB   | d    | 05 | 00 | 00 | Load B
#  131 | S0    | 02 | LDB   | d    | -- | -- | -- | Fetch
#  151 | S1    | 02 | ADD   | 0    | -- | -- | -- | Decode
#  171 | S2    | 03 | ADD   | 0    | 05 | 03 | 00 | Add
#  191 | S3    | 03 | ADD   | 0    | 05 | 03 | 00 | Add
#  211 | S0    | 03 | ADD   | 0    | -- | -- | -- | Fetch
#  231 | S1    | 03 | OR    | 0    | -- | -- | -- | Decode
#  251 | S2    | 04 | OR    | 0    | 08 | 03 | 00 | OR
#  271 | S3    | 04 | OR    | 0    | 08 | 03 | 00 | OR
#  291 | S0    | 04 | OR    | 0    | -- | -- | -- | Fetch
#  311 | S1    | 04 | STR   | e    | -- | -- | -- | Decode
#  331 | S2    | 05 | STR   | e    | 0b | 03 | 00 | Store
#  351 | S0    | 05 | STR   | e    | -- | -- | -- | Fetch
#  371 | S1    | 05 | OUT   | 0    | -- | -- | -- | Decode
#  391 | S2    | 06 | OUT   | 0    | 0b | 03 | 00 | Output

# === Final Results ===
# AC: 0b (11)
# B:  03 (3)
# O:  0b (11)
# Memory[14]: 0b (11)
```

2.6 Hardware Implementation

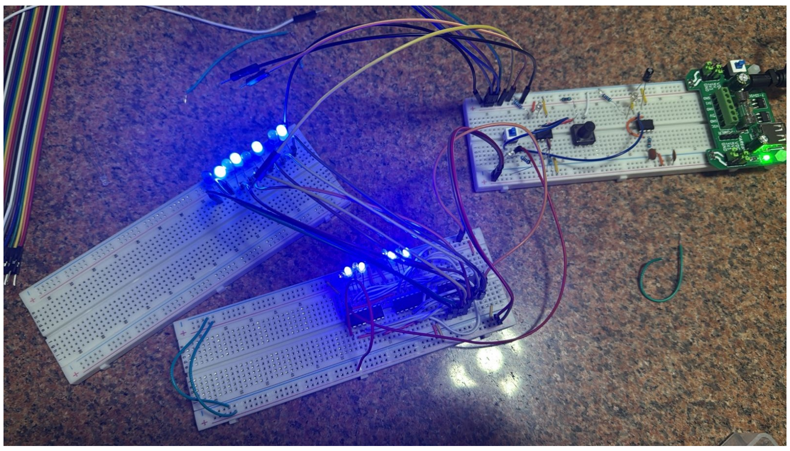


Figure 2: ARM 1 Hardware

3 ARM32: Single-Cycle Processor

3.1 Architecture Overview

ARM32 implements a subset of the 32-bit ARM instruction set in a single-cycle architecture. Each instruction completes in one clock cycle, with separate instruction and data memories.

3.2 Supported Instructions

Table 3: ARM32 Instruction Set

Type	Instructions	Format	Description
Data Processing	ADD, SUB, AND, ORR	DP	Register/Immediate arithmetic
Memory	LDR, STR	Mem	Load/Store with offset
Branch	B	Branch	Unconditional branch

3.3 Instruction Encoding

3.3.1 Data Processing Instructions

$$\underbrace{\text{Cond}}_{31:28}\ \underbrace{00}_{27:26}\ \underbrace{\text{I}}_{25}\ \underbrace{\text{Opcode}}_{24:21}\ \underbrace{\text{S}}_{20}\ \underbrace{\text{Rn}}_{19:16}\ \underbrace{\text{Rd}}_{15:12}\ \underbrace{\text{Src2}}_{11:0}$$

3.3.2 Memory Instructions

$$\underbrace{\text{Cond}}_{31:28}\ \underbrace{01}_{27:26}\ \underbrace{\text{IPUBWL}}_{25:20}\ \underbrace{\text{Rn}}_{19:16}\ \underbrace{\text{Rd}}_{15:12}\ \underbrace{\text{Offset12}}_{11:0}$$

3.3.3 Branch Instructions

$$\underbrace{\text{Cond}}_{31:28}\ \underbrace{101}_{27:25}\ \underbrace{\text{L}}_{24}\ \underbrace{\text{Offset24}}_{23:0}$$

3.4 Datapath Architecture

Table 4: ARM32 Major Components

Component	Description
Program Counter	32-bit PC with branch support
Instruction Memory	1024-word ROM
Register File	15 general-purpose registers + R15 (PC)
ALU	ADD, SUB, AND, ORR with flags
Data Memory	1024-word RAM
Extend Unit	8-bit, 12-bit, and 24-bit sign extension
Control Unit	Main decoder + conditional logic

3.5 Conditional Execution

ARM32 supports all 15 condition codes:

Table 5: Condition Codes

Code	Suffix	Meaning	Flags
0000	EQ	Equal	Z=1
0001	NE	Not Equal	Z=0
0010	CS/HS	Carry Set	C=1
0011	CC/LO	Carry Clear	C=0
1010	GE	Greater or Equal	N=V
1011	LT	Less Than	N≠V
1100	GT	Greater Than	Z=0 and N=V
1101	LE	Less or Equal	Z=1 or N≠V
1110	AL	Always	–

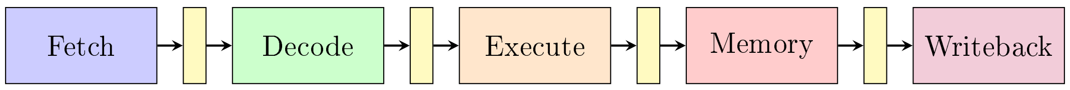
4 ARMPIP: 5-Stage Pipelined Processor

4.1 Architecture Overview

ARMPIP extends ARM32 with a 5-stage pipeline architecture to improve throughput. The pipeline stages are:

1. **Fetch (F)**: Read instruction from memory
2. **Decode (D)**: Decode instruction, read registers
3. **Execute (E)**: Perform ALU operation
4. **Memory (M)**: Access data memory
5. **Writeback (W)**: Write result to register file

4.2 Pipeline Diagram



4.3 Extended Instruction Set

Table 6: ARMPIP Instruction Set

Opcode	ALUControl	Instruction	Operation
0100	000	ADD	Rd ← Rn + Src2
0010	001	SUB	Rd ← Rn - Src2
0000	010	AND	Rd ← Rn & Src2
1100	011	ORR	Rd ← Rn | Src2
1110	100	BIC	Rd ← Rn & ~Src2
0001	101	EOR	Rd ← Rn ⊕ Src2
1101	110	MOV	Rd ← Src2
–	–	LDR	Rd ← Mem[Rn + offset]
–	–	STR	Mem[Rn + offset] ← Rd
–	–	B	PC ← PC + 8 + offset

4.4 Hazard Detection and Resolution

4.4.1 RAW (Read After Write) Hazard

Data hazards occur when an instruction depends on the result of a previous instruction still in the pipeline.

Solution: Data Forwarding

Listing 1: Forwarding Logic

```
// Forward from Memory stage (priority)
if (Match_1E_M) ForwardAE = 2'b10;
// Forward from Writeback stage
else if (Match_1E_W) ForwardAE = 2'b01;
// No forwarding needed
else ForwardAE = 2'b00;
```

4.4.2 LDR Hazard

When an instruction immediately following LDR needs the loaded value, forwarding is insufficient because the data isn't available until the Memory stage.

Solution: Pipeline Stall

Listing 2: LDR Stall Detection

```
assign LDRStall = ((RA1D == WA3E) | (RA2D == WA3E))
                  & MemtoRegE & RegWriteE;
assign StallF = LDRStall;
assign StallD = LDRStall;
assign FlushE = LDRStall | BranchTakenE;
```

4.4.3 Control Hazard

Branch instructions cause control hazards because the branch target is not known until the Execute stage.

Solution: Pipeline Flush

Listing 3: Branch Flush Logic

```
assign FlushD = PCSrcW | BranchTakenE;
assign FlushE = LDRStall | BranchTakenE;
```

4.5 Hazard Unit Summary

Table 7: Hazard Resolution Mechanisms

Hazard Type	Detection	Resolution	Penalty
RAW (register)	Register match	Forwarding	0 cycles
RAW (LDR)	MemtoRegE & match	Stall + Forward	1 cycle
Control (branch)	BranchTakenE	Flush F, D, E	2 cycles

5 Implementation Details

5.1 Module Hierarchy

5.1.1 ARMPIP Module Structure

```
top
+- arm
|   +- controller
|   |   +- decoder
|   |   +- condcheck
|   |   +- pipeline registers (floprc, flopr, flopenr)
|   +- datapath
|   |   +- regfile
|   |   +- alu
|   |   +- extend
|   |   +- mux2, mux3
|   |   +- adder
|   |   +- pipeline registers
|   +- hazard
+- imem
+- dmem
```

5.2 Key Design Decisions

1. **Register File Initialization**: All registers initialized to zero to prevent X propagation in simulation.

2. **MOV Instruction**: Implemented as a dedicated ALU operation (ALUControl = 110) that passes Src2 directly, avoiding dependency on uninitialized Rn.
3. **Branch Target Calculation**: Uses PC+8+offset following ARM convention where PC during execution points to current instruction + 8.
4. **Forwarding Priority**: Memory stage forwarding has priority over Writeback stage to provide the most recent value.

6 Verification and Testing

6.1 Test Program

The comprehensive test program validates all required functionality:

Listing 4: Test Program Assembly

```
; Test ADD, SUB with RAW hazards
MOV R0, #0          ; E3A00000
MOV R1, #10         ; E3A0100A
MOV R2, #3          ; E3A02003
ADD R3, R1, R2      ; E0813002 (R3 = 13, forward R1,R2)
SUB R4, R3, R1      ; E0434001 (R4 = 3, forward R3)

; Test AND, ORR, BIC, EOR
MOV R5, #0xFF       ; E3A050FF
MOV R6, #0x0F       ; E3A0600F
AND R7, R5, R6      ; E0057006 (R7 = 15)
ORR R9, R8, R6      ; E1889006 (R9 = 175)
BIC R2, R0, R1      ; E1C02001 (R2 = 240)
EOR R5, R3, R4      ; E0235004 (R5 = 255)

; Test STR, LDR
STR R1, [R0, #0]    ; E5801000 (Mem[0] = 42)
LDR R6, [R0, #0]    ; E5906000 (R6 = 42)

; Test LDR Hazard (stall required)
LDR R8, [R0, #8]    ; E5908008
ADD R9, R8, #1      ; E2889001 (must stall)

; Test Branch (control hazard)
B skip              ; EA000001
MOV R10, #99        ; SKIPPED
MOV R10, #98        ; SKIPPED
skip:
MOV R10, #7         ; E3A0A007 (branch target)

; Final verification
MOV R0, #100        ; E3A00064
STR R10, [R0, #0]   ; E580A000 (Mem[100] = 7)
```

6.2 Expected Results

Table 8: Test Results Verification

Test	Expected	Validates
Mem[0] = 42	STR instruction	Memory write
Mem[4] = 255	EOR result stored	Bitwise XOR
Mem[8] = 42	LDR hazard setup	Memory operations
Mem[100] = 7	Final result	Branch + all hazards

6.3 Simulation Output

```
============================================================================
CIE 439 - Project 2: ARM Pipelined Processor Test
============================================================================
Testing: ADD, SUB, AND, ORR, BIC, EOR, LDR, STR, B
Hazards: RAW (forwarding), LDR (stall), Control (flush)
============================================================================
MEM[  0] =  42 (0x0000002a)  -> TEST 8a PASSED
MEM[  4] = 255 (0x000000ff)  -> TEST 8b PASSED
MEM[  8] =  42 (0x0000002a)  -> TEST 10 SETUP
MEM[100] =   7 (0x00000007)  -> TEST 13 PASSED
============================================================================
SUCCESS! All critical tests passed.
============================================================================
```

7 Performance Analysis

7.1 CPI Comparison

Table 9: Cycles Per Instruction Comparison

Architecture	Best CPI	Worst CPI	Average CPI
ARM1 (Multicycle)	3 (Mem/IO)	4 (Data Proc)	∼3.5
ARM32 (Single-cycle)	1	1	1
ARMPIP (Pipelined)	1	3 (branch)	∼1.2

7.2 Pipeline Efficiency

For the pipelined processor:

- **Ideal throughput**: 1 instruction per cycle
- **LDR hazard penalty**: 1 cycle stall
- **Branch penalty**: 2 cycles (flush 2 instructions)
- **RAW hazards**: 0 cycles (resolved by forwarding)

8 Conclusion

This project successfully demonstrated the progression from simple to complex processor architectures:

1. **ARM1** introduced fundamental concepts: instruction fetch-decode-execute cycles, accumulator architecture, and finite state machine control.
2. **ARM32** expanded to a full 32-bit architecture with register file, conditional execution, and separate instruction/data memories.
3. **ARMPIP** achieved higher throughput through pipelining while correctly handling all three types of hazards: data hazards (forwarding), load-use hazards (stalling), and control hazards (flushing).

The implementations were verified through comprehensive testbenches that validate each instruction type and hazard scenario. The final pipelined processor correctly executes test programs that exercise all supported instructions and hazard conditions.

9 References

1. Harris S., Harris D. *Digital Design and Computer Architecture ARM Edition 2016*

2. Course materials from CIE 439, Zewail City of Science and Technology