Report on ARM Architecture

1. Register Organization

The ARM architecture is designed with a highly efficient register organization that adapts to different processor modes, such as User, FIQ (Fast Interrupt Request), IRQ (Interrupt Request), Supervisor, and others. Below are the key aspects of its register structure:

1.1 Number of Registers

ARM processors typically include **37 registers** in total:

- **31 general-purpose registers (R0-R30)**: These are used for arithmetic, logical operations, and data storage.
- **6 status registers**: Include the **CPSR (Current Program Status Register)** and **SPSRs (Saved Program Status Registers)** for exception handling.

However, only **16 registers (R0-R15)** are visible at any given time in most operating modes. The visibility depends on the mode the processor is currently executing in.

1.2 Register Length

- In **ARMv7** and earlier architectures, all registers are **32 bits wide**.
- In **ARMv8**, the introduction of the AArch64 execution state extends the register width to **64 bits**. The registers are labeled **X0-X30** for 64-bit operations, while **W0-W30** represent the lower 32 bits of these registers.

1.3 Special Purpose Registers

Certain registers have dedicated roles based on their function:

- **R13 (SP)**: The **Stack Pointer** is used to manage the stack, which stores temporary data like return addresses and local variables.
- **R14 (LR)**: The **Link Register** holds the return address for subroutine calls, enabling the program to resume execution after a function completes.
- **R15 (PC)**: The **Program Counter** points to the address of the next instruction to be executed.
- **CPSR**: The **Current Program Status Register** contains flags (**N**, **Z**, **C**, **V**) for condition checking, interrupt disable bits, and mode information.
- **SPSR**: The **Saved Program Status Register** is used in exception modes to save the CPSR during interrupts or exceptions.

These special-purpose registers ensure efficient management of control flow, stack operations, and processor state.

2. ARM Instruction Set

The ARM instruction set is designed for efficiency and flexibility, supporting a wide range of operations through fixed-length 32-bit instructions. Below are the details of its structure, types, data support, and addressing modes.

2.1 Instruction Format

ARM instructions are encoded in a consistent 32-bit format, with fields varying based on the instruction type. Key formats include:

1. Data Processing Instructions

- **Format**: Opcode | S | Rn | Rd | Operand2
  - **Opcode (4 bits)**: Specifies the operation (e.g., ADD, SUB).
  - **S (1 bit)**: Determines whether the CPSR flags are updated.
  - **Rn (4 bits)**: Source register.
  - **Rd (4 bits)**: Destination register.
  - **Operand2 (12 bits)**: Second operand, which can be a register or an immediate value.
- **Example**: ADD R1, R2, R3 adds the contents of R2 and R3, storing the result in R1.

2. Branch Instructions

- **Format**: Condition | Opcode | Offset
  - **Condition (4 bits)**: Specifies the condition under which the branch occurs (e.g., EQ for equal, NE for not equal).
  - **Opcode (4 bits)**: Identifies the branch operation.
  - **Offset (24 bits)**: Signed offset relative to the current PC.
- **Example**: B loop jumps to the label "loop."

3. Load/Store Instructions

- **Format**: Opcode | Rn | Rd | Offset
  - **Opcode (6 bits)**: Specifies the memory operation (e.g., LDR for load, STR for store).
  - **Rn (4 bits)**: Base register.
  - **Rd (4 bits)**: Source/Destination register.
  - **Offset (12 bits)**: Offset from the base address.

- **Example**: LDR R1, [R2, #4] loads the value at [R2 + 4] into R1.

2.2 Instruction Types

ARM supports several types of instructions:

- **Arithmetic**: Perform mathematical operations.
  - Example: ADD R1, R2, R3 adds R2 and R3, storing the result in R1.
- **Logical**: Perform bitwise operations.
  - Example: AND R1, R2, R3 performs a bitwise AND between R2 and R3.
- **Control Flow**: Manage program execution.
  - Example: BNE label branches to "label" if the Zero flag is not set.
- **Memory Access**: Handle data transfer between registers and memory.
  - Example: STR R1, [R2] stores the value in R1 at the memory address in R2.

2.3 Data Types

ARM supports various data types, including:

- **Integers**: 8-bit (byte), 16-bit (half-word), 32-bit (word), and 64-bit (double-word) signed and unsigned integers.
- **Floating-Point**: Single-precision (32-bit) and double-precision (64-bit) IEEE 754 floating-point numbers (introduced in ARMv8).

2.4 Addressing Modes

ARM provides multiple addressing modes to optimize memory access:

- **Immediate**: Uses a constant value.
  - Example: ADD R1, R2, #5 adds 5 to R2, storing the result in R1.
- **Register**: Uses a register as an operand.
  - Example: ADD R1, R2, R3 adds R2 and R3, storing the result in R1.
- **Register Indirect**: Uses a register as a pointer to memory.
  - Example: LDR R1, [R2] loads the value at the memory address in R2 into R1.
- **Base + Offset**: Adds an offset to a base register.
  - Example: LDR R1, [R2, #4] loads the value at [R2 + 4] into R1.
- **PC-relative**: Uses an offset relative to the program counter.
  - Example: LDR R1, [PC, #100] loads the value at [PC + 100] into R1.

3. Cache Organization

ARM processors employ advanced cache hierarchies to minimize memory latency and improve performance. Key features include:

3.1 Multi-Level Caches

- **L1 Cache**: Split into separate instruction (I-cache) and data (D-cache). Typical sizes are **32 KB each**.
- **L2 Cache**: Unified cache shared between instructions and data. Sizes range from **256 KB to 2 MB**.
- **L3 Cache**: Optional, shared across multiple cores in multi-core systems.

3.2 Cache Coherency

ARMv8 introduces hardware coherency mechanisms, such as the **CCI-400** (Cache Coherent Interconnect), ensuring consistency across cores in multi-core systems.

3.3 Cache Policies

- **Write-Back**: Writes data to the cache first; main memory is updated later.
- **Write-Through**: Writes data to both the cache and main memory simultaneously.

3.4 Example

The Cortex-A72 processor exemplifies ARM's advanced caching:

- **L1 Cache**: 48 KB I-cache and 32 KB D-cache.
- **L2 Cache**: 1 MB unified cache.
- **Cache Policy**: Write-back by default.

Conclusion

The ARM architecture excels in efficiency, scalability, and versatility, making it a cornerstone of modern computing. Its register organization provides a robust foundation for computation, while its instruction set supports a wide range of operations. Advanced cache features further enhance performance by reducing memory access delays. These attributes make ARM processors ideal for applications ranging from embedded systems to high-performance servers.