Suez Canal University

Faculty of **Faculty of Computers**

Design ARM (Advanced RISC Machines) Processor with 32-bit computer architecture. By using a 32-bit instruction set called the ARM instructions. In Single, Multi-cycle, and pipeline implementation.

Student name	Amir Haytham Muhammad Salama
Faculty	Faculty of Computers and Informatics
Level	Third Level
Department	Computer Science
National ID	29906261900473

Student code	1714103022
Program	Computer Science Program
Course	Computer Architecture

1. Introduction

What is ARM architecture? The ARM architecture is an architecture of the Reduced Instruction Set Computer (RISC). architecture. Also, ARM, originally Acorn RISC Machine, later Advanced RISC Machine, is a family of computer processor-configured, RISC (Reduced Instruction Set Computing) architectures for different environments. British corporation ARM Holdings designs the architecture and licenses it to other companies who design their own products to implement one of those architectures — including on-chip systems (SoC).

Typical features of RISC architecture present in architecture for ARM:

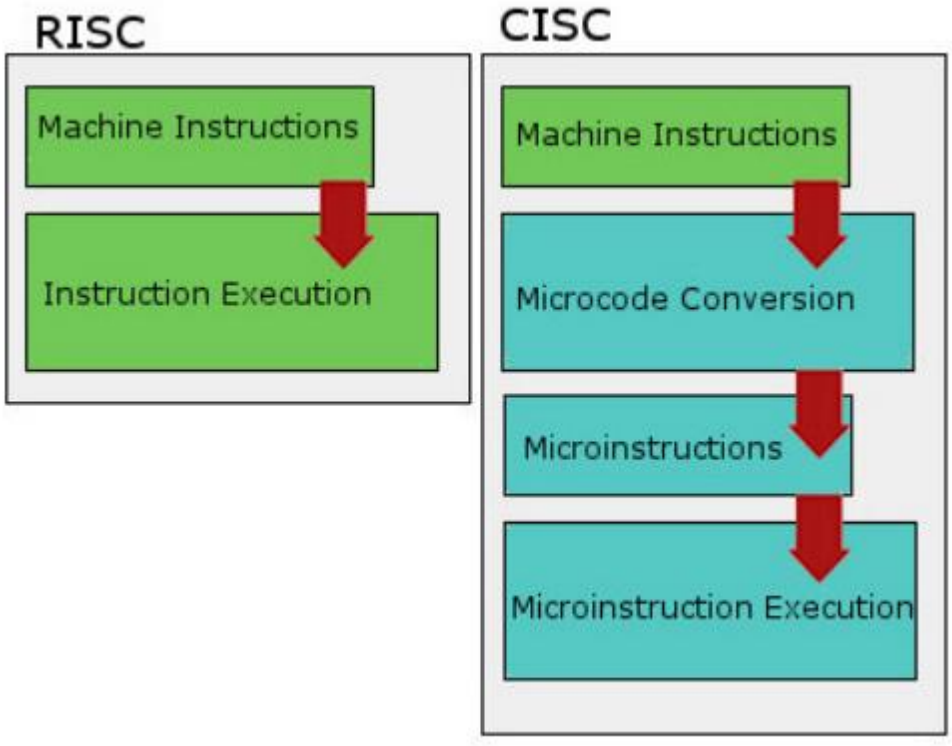
- a large uniform register file
- Load / store architecture (in this case data processing only for registered content)
- simple addressing modes, with all load / store addressed determined from the contents of the registry and instruction fields only

ARM architecture specifies that ARM processors can achieve a good balance of high performance, small code size, low power consumption and low power consumption according to the ARM Architecture Reference Handbook silicon area:

- Instructions that combine a shift with a logical or arithmetic operation
- Auto-increment and self-decrement of the modes for optimizing program loops
- Loading and storing multiple instructions to maximize data passage
- Conditional implementation of nearly all instructions for maximizing the execution throughput

ARM is a computer instruction set architecture with 32-bit reduced instruction set. The abbreviation for the ARM was known as the Advanced RISC Machine and the Acorn RISC Machine. ARM Processors became quite dominant in the mobile industry and embedded electronics market because of its suitability for low power applications. From mobile phones, to PDA, calculators, MP3 players, routers and other small electronics we can find ARM processors anywhere at this time.

If you've been paying any attention to smartphones and tablets you've probably heard of the term "ARM" used to refer to the inside hardware. It's tossed around left and right, mostly as a distinguishing point from laptops and desktops, which use Intel x86. The RISC is the Key To ARM.



RISC is a philosophy of designing processors in its broadest form. It stems from a assumption that a processor that has a fairly simple set of instructions would be more effective than a more complicated one. The

concept was first used back in the 1980s for a research project named Berkeley RISC that examined the possibilities of developing this approach and then produced processors based on it.

All ARM processors are considered RISC designs, but this doesn't mean much because RISC itself is simply a design approach, rather than a technological standard or architecture for processors. Still, ARM is properly framed by a fundamental understanding of RISC.[13,14]

2. Research items

- Computer Design and The Central Processing Unit
- ARM's Basics
- Advanced RISC Machine (ARM)
- Definition of RISC
  - RISC architecture & Organization
- ARM Instruction set
  - ARM Instruction set FOR RISC
  - LOAD and STORE instructions
- Single Cycle & Multicycle and Piplining
  - Example on ADD instruction set with an Explanation
- Control Bus & Data Bus
- Addressing Mode
- Instruction Format (op code)

3. Research

- # Computer Design and The Central Processing Unit [9]

The performance of a computer system is measured by the time that it takes to execute programs, the shorter the elapsed time the higher the performance rating. In order to optimize the output, a designer needs to find ways to suit the output of each part on the machine, to create a balanced device. As technology

changes and new discoveries are made, the performance bottle-neck is becoming different parts of a computer.

The architecture of a computer defines the major attributes of the design. The number of registers, their layout, the instructions and addressing modes that the computer understands are all part of the architecture, while the number of clock cycles taken to Run each instruction, the type of transistor logic used to build the CPU, and the memory layout are all part of the implementation.

The Central Processing Unit (CPU), as shown in the following Figure, is the part of the computer that executes instructions. The CPU consists of several specialized functional units (e.g., the Arithmetic Logic Unit, or ALU). The Instruction Decoder controls these functional units which activates the necessary sections of each unit to perform the operation specified by each instruction. Each functional unit is connected via data paths, through which flows data, sections of decoded instructions, and information on internal control.

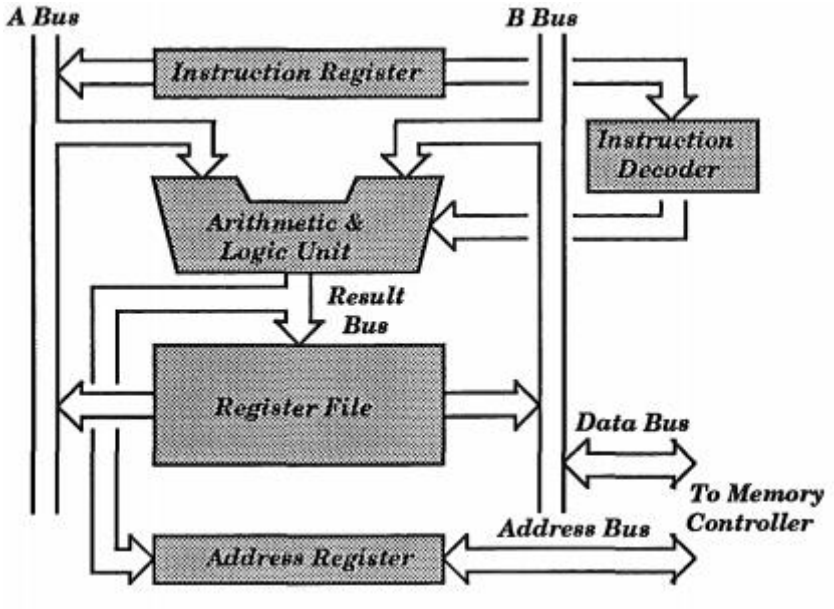


Fig: A Simple CPU

• ARM's Basics [15]

ARM refers to itself as an architecture which, when contrasted with Intel, can cause confusion. Intel gives each new chip design its own unique code and talks about each as a different architecture – even though there are so several similarities and everybody uses the same instruction set (x86). ARM, on the other hand, treats its designs as an unbroken family. Updates remain an integral part of ARM architecture. They're just given a new version number.

The most important feature for customers is not the micro-architecture (the chip 's physical design) but rather the collection of instructions. The set of instructions is the basic set of capabilities and features the software is made available by a processor. This decides what arithmetic to use, how to assign the cache and the order in which instructions will be executed. Software designed for one instruction set can't be used on another unless it's revised.

You can not separate micro-architectures and instruction sets because the architecture is a physical expression of the instruction set. For this reason processors based on ARM tend to be small, efficient and relatively slow. The simple set of instructions calls for a small , simple design with fewer transistors. Transistors consume power and increase die size (which increases production cost), so having as few as possible is ideal when selecting a processor for a smartphone or tablet.

• Advanced RISC Machine (ARM)[15]

The ARM is an instruction set architecture (ISA) 32-bit reduced instruction set computer (RISC), developed by ARM Holdings. It was known as the Advanced RISC Computer, and the Acorn RISC Computer, before that.

- The ARM architecture is the most commonly used 32-bit ISA in terms of the produced numbers.
- corn Computers originally conceived them as a processor for personal desktop computers, a market now dominated by the x86 family used by computers compatible with IBM PCs.
- The relative simplicity of the ARM processors has made them ideal for applications with low power.
- As relatively low cost and small microprocessors and microcontrollers this has made them dominant in the mobile and embedded electronics market.

Fig: Advanced RISC Machine (ARM) Microprocessor developed by Conexant Computers

• Definition of RISC [9,11,12]

RISC, or Reduced Instruction Set, is a microprocessor type Architecture that uses a small set of highly optimized instructions rather than a more specialized set of instructions that are often found in other architectural types.

Evolution/History

The first RISC projects emerged in the late 70s and early 80s from IBM, Stanford, and UC-Berkeley. The IBM 801, Stanford MIPS, and RISC 1 and 2 of Berkeley were all developed with a common concept that has become known as RISC. Of most RISC processors, certain design features were characteristic

Large Number of Registers

In general, the RISC architecture theory integrates a larger number of registers to avoid large numbers of memory interactions

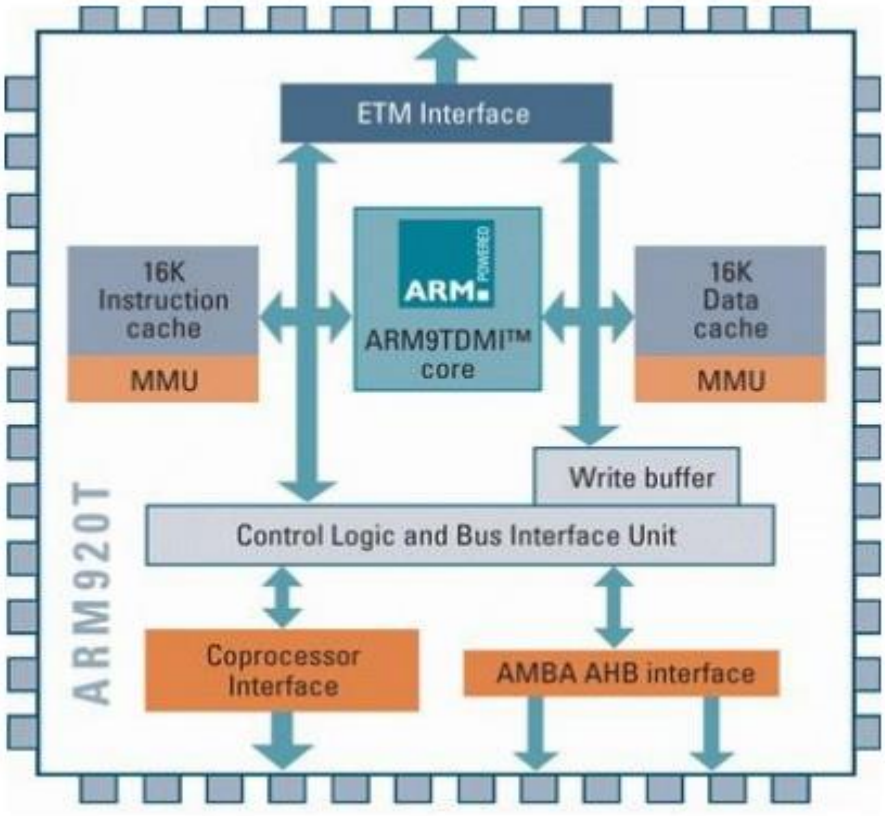


Fig: Advanced RISC Machine (ARM)

RISC architecture:

Berkeley incorporated the architecture of a Reduced Instruction Set (RISC) computer.

– A fixed instruction size (32-bit), with few formats;

- processors typically had instruction sets of variable lengths with lots of formats.

– A load / store architecture was instructions that data processing works only on registers and is distinct from memory access instructions;

- CISC processors usually permitted memory values to be used as operands in the instructions for data processing.

– A wide register bank of thirty-two 32-bit registers, all of which could be used for any purpose, enabling efficient operation of the load and store architecture;

• CISC register sets were growing, but none of them were bigger and most had different registers for different purposes.

RISC organization:

- Hard-wired instruction decode logic
  – CISC processor decoded large microcode ROMs with their instructions
- Pipelined execution
  – CISC processors allowed little, if any, overlap (though they do now) between consecutive instructions
- Single-cycle execution
  – Typically CISC processors took many clock cycles to complete a single instruction

• Instruction Set [10,18,19]

Computer instruction sets are important factors in the machine 's overall price / performance. Will CPU instruction must be implemented using custom-designed digital logic (this is addressed later using "microprogramming" to replace some logic by sacrificing performance). This "custom silicon" is extremely labour-intensive to manufacture, making it very costly to implement a broad and/or complex set of instructions in hardware. Emulating some instructions with software is less expensive, but lowers the performance of the computer, due to the inefficiency of using combinations of the existing instructions to emulate missing instructions.

While machines exist whose instruction set is tailored to a specific high-level language, such a design would be inappropriate for a microcomputer with a general purpose -8-. Table 1 illustrates a simple instruction set. An instruction set must be able to efficiently support the many unique features specific to different high level languages, although programming languages have major features in common.

- **arithmetic operations for integer and floating point data.**
  An instruction set wxill need operations to move data to and from the memory and registers, and be able to perform some simple arithmetic on that data. Integer addition and subtraction instructions are found in the simplest of CPUs, while multiply and divide are quite common in more complex architectures. The results of such instructions usually update the processor's condition flags: a negative result will set the negative flag, a zero result will set the zero flag, a carry or borrow from additions and subtractions will set the carry flag, and an overflow will set an overflow flag. Floating Point operations are usually carried out in a totally separate processor, the Floating Point Unit (FPU), but the functions it performs are similar to the integer unit.

- **operations on Boolean data.**

Boolean values, arrays of Boolean values, and sets require bit-wise logical operators like And, Or and Exclusive Or. The And operator is used to clear a bit, Or is used to set a bit, and Exclusive Or is used to toggle a bit. Operations on bit fields such as Not, Left Shift, Right Shift and Rotate can be used to build values for comparison with Boolean data.

Instruction	Mnemonic	Operation
Add	ADD	Dest := Src1 + Scr2
Subtract	SUB	Dest := Src1 – Src2
Logical AND	AND	Dest := Src1 AND Src2
Logical OR	OR	Dest := Src1 OR Src2
Logical EOR	EOR	Dest := Src1 EOR Src2
Logical NOT	NOT	Dest := NOT Src1
Logical Shift Left	LSL	Dest := Src1 * 2^{Src2}
Logical Shift Right	LSR	Dest := abs(Src1 / 2^{Src2})
Rotate	ROT	Dest := Src1 Rotated Src2 bits
Arithmetic Shift Right	ASR	Dest := Src1 / 2
Compare	CMP	Src1 - Src2
Move	MOV	Dest := Src1
Multiply	MULT	Dest := Src1 * Src2
Divide	DIV	Dest := Src1 / Src2
Jump	JMP	PC := Dest
Procedure Call	CALL	Dest := PC, PC := Dest
Procedure Return	RET	PC := Dest
Conditional Branch	Bcc	IF cc PC:=PC+offset

Where : Dest , Src1 and Src2 are registers or memory addresses
PC is the program counter
cc is a condition code
offset is an address offset

Table 1: A Simple Instruction Set

- **support for the conditional execution of instructions depending on some previous condition.**

Can be used to compare two pieces of data using a comparison instruction. The condition flags are usually set to reflect the result of the last compare instruction, the negative flag indicating which operand was the larger and zero flag if they were equal. A branch instruction will jump to a different part of the program depending on the state of one - 10- or more condition flags, for example "Branch on Greater Than or Equal to". Many architectures combine the instructions for comparing and branching, as they are widely used together. These instruction are used to implement IF statements, conditional loops (FOR, WHILE etc.) and CASE type statements.

- **jump instructions to change the flow of execution.**
  High level language constructs like infinite loops, premature loop terminators and GOTO statements require a Jump instruction to unconditionally alter the value held in the program counter.

- **constructs to implement procedure calls.**
  By storing the current value of the program counter, and using a jump instruction, a program can execute a procedure and then continue execution just after the point of call by jumping back to the value in the stored program counter. If the return address is placed onto a stack, it is possible to nest procedural calls, and procedure recursion is possible.

- **addressing modes to access data structures held in memory.**
  Data structures such as arrays and records from Pascal allow the processor to be able to load data from and to a given address by adding an offset to a base address. The base address holds the address of the start of the array or record and the offset holds the distance of the required element from the beginning. These same addressing modes can be used to access data held in the stack frame of a procedure held on the stack. - 11- Of course, all programming languages have their own characteristic features, and these must be covered by a general purpose architecture. The microprocessors designed in the early 1980's added many instructions and addressing modes to the simple instruction set shown in Table 1 to add hardware support for the features of many high level languages. This has the unfortunate side

effect that in some situations some instructions will be quite useless, effectively a waste of "silicon real estate" on the CPU chip.

ARM Instruction Set for RISC:[9]

ALU: Two registers for a third Register & immediately extended Ops signed include ADD, SUB, AND, OR. Immediate variants of the operations Arithmetic ADDU signed and unsigned

Load/Store: Register source (base register) and an Store immediate field (offset). Sum makes the effective address. The source or sink is second register.

Branch/Jumps: Conditional transfer of control Jumps Unconditional transfer. Destination is a signed extended offset to be added to the PC

Basic Instructions:

Data Movement: LOAD, STORE, MOVE
Arithmetic and Logical: ADD, SUB, AND, XOR, SHIFT
Branch: JUMP (unconditional), JZ, JNZ (conditional)
Procedure Call: CALL, RETURN
Input Output: Memory-mapped I/O*
Miscellaneous: NOP, EI (enable interrupt)

LOAD and STORE instructions: [17]

- Load instructions take a single value from memory and write it to a general purpose registry.
- Store instructions read the value from the general-purpose register and store it in memory.

Only LOAD and STORE instructions access the memory. All other instructions use register operands. Used in all RISC machines.

If X, Y, Z is a memory operand then X:= Y Z will be used as:

```
1  LOAD r1, Y
2  LOAD r2, Z
3  ADD r1, r2, r3
4  STORE r3, X
```

Performance improves if the operand(s) can be kept in registers for most of the time. Registers are faster than memory.[1]

- **Load and store instructions with a single instruction format**

  LDR r9,[r12,r8,LSL #2]
  - R12 :--base register
  - R8 :--an offset value created by shifting register r8
  - LSL :-- logical shift left by #2 bit
  - **Effective address:-base register offset value**.
  - For example
    r12 :-- 0x4000
    r8 :-- 0x20
    effective address :0x4080
    The address used to access memory

• Single Cycle & Multicycle and Piplining with an Example

- **Single Cycle**: The RISC processors have a single-cycle CPI (clock by instruction). This is because of the optimization of each CPU instruction.
  - Single Cycle Problems:
    • Single-cycle instruction is not found in current computers
    • Disadvantages: – Inefficient in both performance and HW costs – The clock cycle must have the same length for each instruction – The clock cycle must be as long as possible. –In the case of complicated instruction such as floating point, it would be more difficult – Functional units must be duplicated, as each can only be used once per cycle.[10,8,12]

- **Pipelining:** A technique allowing the simultaneous execution of parts, or stages, of instructions to process instructions more efficiently[5,12,16]

- **Mulitcycle:[4,8]**

  - Each instruction shall be divided into a number of steps
  - Each step shall be performed in one clock cycle
  - Advantages: – Makes the usage of usable systems more than once – Help to rising HW costs
  - Major data path differences from single cycle: – Single memory unit Used for instruction as well as data – ALU single instead of three – In each large operating unit, one or more registers are added . Keep that unit output until the next clock input is used Cycle.
  - Added registers – Instruction register (IR) • Save instruction read output from memory – Memory data register (MDR) • Save data read output from memory – A and B registers • Keep the registry operand read from registry file – ALUOut registry • Keep ALU efficiency

Example on Single Cycle:

The problem: P = Q + R + S
Assume that r1 = Q, r2 = R, r3 = S. The result of Q is going to go in r0.

```
AREA Example1, CODE, READONLY
ADD r0,r1,r2
ADD r0,r3
Stop    B    Stop
END
```

- **Explain the program**

Line1 :- AREA Example1, CODE, READONLY:

- Assembler directive and program setup required.
- It is a development system characteristic and not an ARM assembly language.
- AREA refers to the code segment, Example 1 is the name given to that segment
- CODE includes functional application, not files
- READONLY state that it cannot be modified at run time
- Line2 :-

```
ADD r0,r1,r2
ADD r0,r3
```

Simplify the problem :-

R0=R1+R2+R3

First :- know R0 store the value

Second : used ADD instruction to solve the problem

- Stop B Stop

Stop :- is a label that can be used to refer to that line.

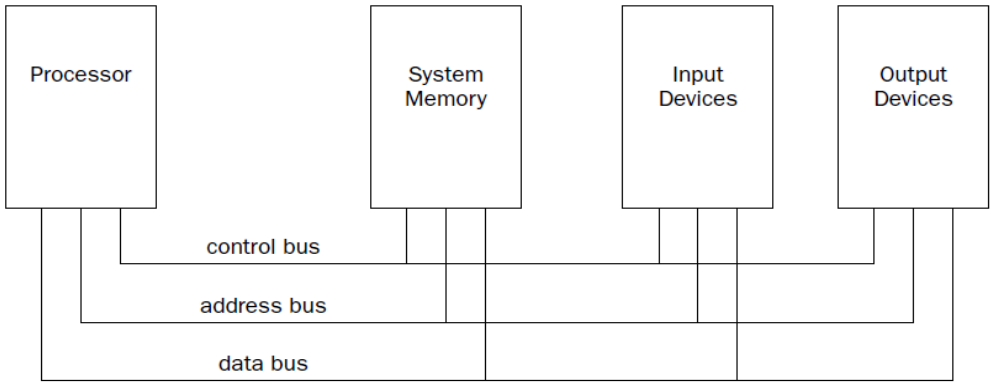
- The Stop B Stop phrase means 'Branch to the line called Stop' which is used to construct an endless loop.
- This is a convenient way to end programs in plain examples such as these.
- END
- END :- is an assemble directive that tells the assembler there is not more code to follow. It ends the program.

• Control Bus & Data Bus

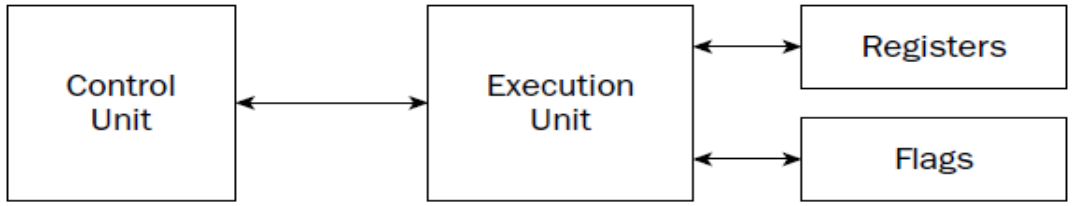
The processor contains the hardware and instruction codes that control the operation of the computer. It is connected to the other elements of the computer (the memory storage unit, input devices, and output devices) using three separate buses:

- a control bus
- an address bus
- a data bus

The control bus is used to synchronize the functions between the processor and the individual system elements. The data bus is used to move data between the processor and the external system elements. An example of this would be reading data from a memory location. The processor places the memory address to read on the address bus, and the memory storage unit responds by placing the value stored in that memory location on the data bus for the processor to access.[16]



And the processor itself consists of many components. Each component has a separate function in the processor's ability to process data. Assembly language programs have the ability to access and control each of these elements. As the following Figure:


Control Unit
Execution Unit
Registers
Flags

• Addressing Mode

The term addressing modes refers to the way an instruction operand is specified. The addressing mode specifies a rule that interprets or modifies the instruction 's address field before the operand is actually executed. In RISC, Simple addressing modes allow fast address computation of operands. Because RISC processors employ instructions for register-to-register, most instructions use register-based addressing. Only the instructions for loading and storing require memory-addressing mode. RISC designs provide very few addressing modes: often just one or two. They provide indirect addressing mode for the basic register, often allowing for a small displacement, either relative or absolute.[5]

• The memory is handled by the register and by the offset. LDR R0 [R1] LDR R0, [R1] @ [R1] mem[R1]

- Three ways to specify offsets:

– Immediate @ mem[R1+4]

```
LDR R0, [R1, #4]
```

– Register @ mem[R1+R2]

```
LDR R0, [R1, R2]
```

– Scaled register @ mem[R1+4*R2]

```
LDR R0, [R1, R2, LSL #2]
```

- # Instruction Format (op code)

As we know that the model for execution of ARM instructions is different from the MIPS, PowerPC, and SPARC models. This is somewhat similar to the Itanium one in that most ARM instructions are executed conditionally [20]. We used predicate registers in the Itanium architecture to evaluate whether or not an instruction should be executed. If the specified predicate register is 1, the instruction is executed; otherwise, it is treated as a nop (no operation). The ARM architecture uses a similar scheme. However, ARM uses a field of 4-bit condition code to express the condition that the instruction should be executed under. In each instruction, the most significant four bits (bits 28–31) are reserved for this purpose (see the following Figure). As shown in the following Table, these four bits define a number of conditions. This table gives the condition code field value and the mnemonic used in the instructions along with a description and the condition tested. As shown in this table, the four condition code flags are used to test for the condition specified in the condition field of the instructions.

The mnemonic given in the second column is appended to the instruction mnemonic to indicate the condition. For instance, we use beq to branch on equal, whereas we use blt to branch on the less than condition. Out of the 16 values, two are unconditional. In particular, if the condition field is 1110, the instruction is always executed. This is the default in the sense that if no condition is specified in the instruction, it is assumed to be AL and the instruction is unconditionally executed. Thus we can define either b or bal to branch unconditionally. We give more examples later on. The last clause is known as "Never" and the instruction is never executed, which renders it basically a nop. Latest versions, however, use it for other purposes (e.g., in DSP instructions) and thus do not follow this rule. The ARM instruction format isn't as straightforward as MIPS architecture. ARM supports an extensive range of instruction formats. What we have shown in the following Figure is a small sample of the formats used by ARM. The A encoding shown in this figure is used by most arithmetic and logical instructions. The field opcode specifies such operation as add and cmp. The Rd field identifies the register of destination which receives the result. One of the source operands is

in the source register Rn. Since we have 16 general purpose registers at our disposal, each of these fields is 4 bits long. The second source operand is provided by the shifter-operand field. This field takes the least significant 12 bits and is used to specify the second source operand. How the second source operand is specified depends on the addressing mode and whether the operand is to be manipulated prior to using it. The I bit distinguishes between the immediate and register forms used [2,3]

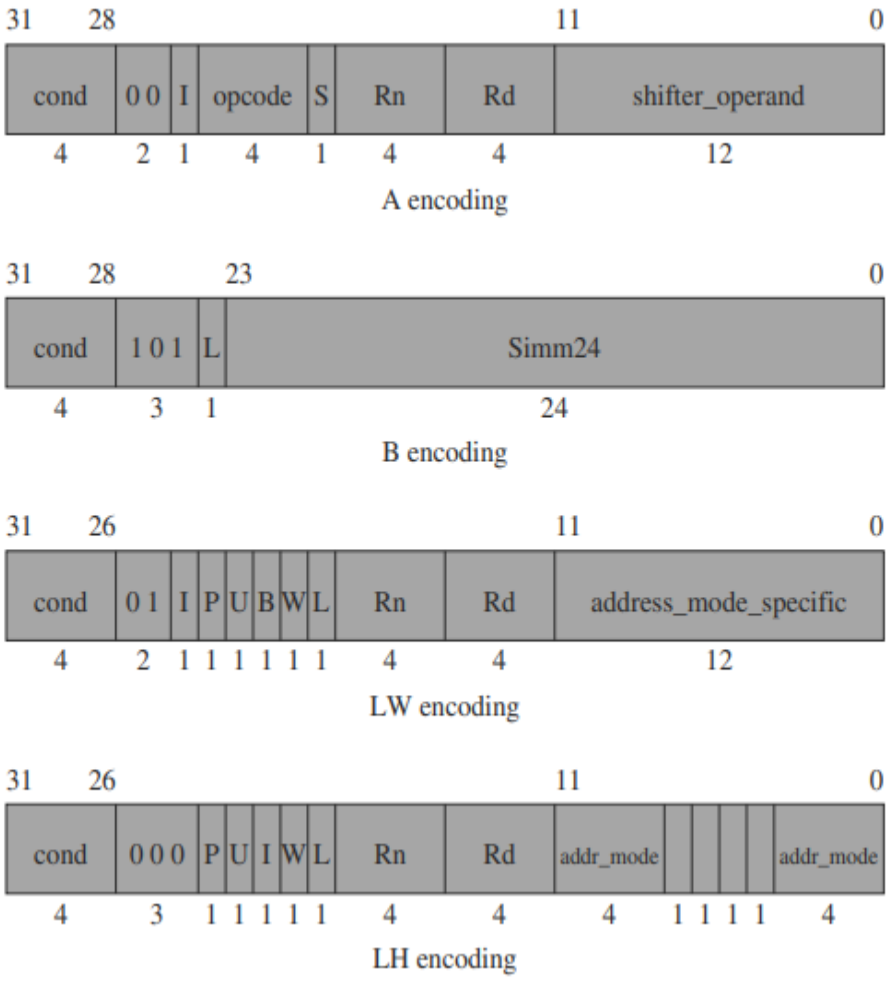


Fig: Selected ARM instruction formats

cond	Mnemonic	Description	Condition tested
0000	EQ	Equal	Z = 1
0001	NE	Not equal	Z = 0
0010	CS/HS	Carry set/unsigned higher or same	C = 1
0011	CC/LO	Carry clear/unsigned lower	C = 0
0100	MI	Minus/negative	N = 1
0101	PL	Plus/positive or zero	N = 0
0110	VS	Overflow	V = 1
0111	VC	No overflow	V = 0
1000	HI	Unsigned higher	C = 1 AND Z = 0
1001	LS	Unsigned lower or same	C = 0 OR Z = 1
1010	GE	Signed greater than or equal	N = V
1011	LT	Signed less than	N ≠ V
1100	GT	Signed greater than	Z = 0 AND N = V
1101	LE	Signed less than or equal	Z = 1 OR N ≠ V
1110	AL	Always (unconditional)	—
1111	(NV)	Never (unconditional)	—

Table: ARM condition codes

4. Conclusion

So let's conclude what I was talking about, In RISC Principles, We have introduced important features that distinguish RISC designs from their counterparts at CISC. The CISC designs have detailed guidance and a wide variety of approach modes. The rationale for this complexity is the need to close the semantic distance between languages of high level and machine languages. Efficient use of processor and memory resources had been critical in the early days. Complex instructions help to reduce the memory requirements. However, empirical research suggested that compilers do not use such complex instructions; rather, simple instructions are used to synthesize complex instructions. Such observations led to the designers taking a fresh look at the philosophy of processor design. RISC principles were proposed as an alternative to the CISC, based on empirical studies of CISC processors. Most of the processor designs currently in use

are based on these RISC principles. The next part looks at several RISC architectures. Also in Addressing Modes, We have described the addressing modes that we can use to write the assembly language programs. Some of these modes are provided by the assembler, not by the processor. We have presented some examples to illustrate the use of these addressing modes. These examples have shown how the memory addressing modes are used in writing the assembly language programs.

And in the end, ARM Architecture, The ARM architecture is remarkably different from the other architectures we have seen. It shares some features with the Itanium architecture. One feature of the ARM instruction set that sets it apart is that its instructions are executed conditionally. If the specified condition is not true, the instruction acts as a nop. Although Itanium also uses conditional instruction execution, the ARM uses a 4-bit condition code as opposed to the predicate bits of Itanium. Another interesting feature of ARM is that arithmetic and logical instructions can pre-process (shift/rotate) one of the input operands before operating on it. In this format, the destination register also supplies a source operand. Even though the ARM is a RISC architecture, it does not strictly follow the RISC principles as does the MIPS. Some ARM instructions like ldm and stm, for instance, aren't simple instructions. In addition, it provides a large number of addressing modes and uses a somewhat complex instruction format. However, having looked at five different RISC designs, we will also see a lot of commonality among these architectures and the RISC principles. Then, Explaining the Instruction set and the ARM Instruction set Single Cycle & Multicycle and Piplining with an Example on simple cycle the usage of The control bus & control bus then addressing mode with an example to explain load memory address. At the end, Instruction format (op code).

5. References

1. Britton, R. L. (2004). *MIPS assembly language programming*. Pearson/Prentice Hall.
2. Divivier, R. J., Haines, R., Nemirovsky, M. D., & Perez, A. (1998). *U.S. Patent No. 5,752,269*. Washington, DC: U.S. Patent and Trademark Office.

3. Furber, S. B., & Thomas, A. R. P. (1990). ARM3—a study in design for compatibility. *Microprocessors and Microsystems*, *14*(6), 407-415.
4. Vijay, J. V., & Bansode, B. (2015). ARM processor architecture. *International journal of science, engineering and technology research (ijsetr)*, *4*(10).
5. Furber, S. B., & Wilson, A. R. (1987). The Acorn RISC Machine ዘ an architectural view. *Electronics and Power*, *33*(6), 402-405.
6. Jaggar, D. V. (1990). A Performance Study of the Acorn RISC Machine.
7. Dandamudi, S. P. (2005). *Guide to RISC processors: for programmers and engineers*. Springer Science & Business Media.
8. Patterson, D. A., & Ditzel, D. R. (1980). The case for the reduced instruction set computer. *ACM SIGARCH Computer Architecture News*, *8*(6), 25-33.
9. Mano, M. M. (2003). *Computer system architecture*. Prentice-Hall of India.
10. Stallings, W. (2003). *Computer organization and architecture: designing for performance*. Pearson Education India.
11. https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/
12. https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/armchip.html#processors
13. https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/armchip.html#architecture
14. https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/history.html
15. https://en.wikipedia.org/wiki/Berkeley_RISC
16. https://en.wikipedia.org/wiki/ARM_architecture#Pipelines_and_other_implementation_issues
17. http://en.wikipedia.org/wiki/RISC
18. http://www.arm-development.com/
19. http://infocenter.arm.com/help/index.jsp?topic=index.html
20. http://www.arm.com/