COA Assignment 1

Samarth Gupta
Roll No. 1910110338
B.Tech., CSE
Shiv Nadar University
Greater NOIDA, U.P.
sg384@snu.edu.in

April 15, 2021

1 Intel x86 processor vs. ARM processor

A CPU works when given an instruction, say, to move data between registers and memory or to perform a calculation. For machines, the set of machine language instructions that a computer can follow is called the Instruction set architecture (ISA). Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC) are categories of processor or Instruction Set Architecture (ISA). RISC refers to a streamlined version of its predecessor, CISC. The ISA interface is the boundary between hardware and software. Both application programs and utilities access the ISA directly. However, mobile applications are not written using CPU instructions as they run across a large variety of chips. They are written in various higher-level programming languages (like Java or C++) that are compiled for specific instruction sets so they run correctly on the underlying ISA. These instructions are further decoded into microcode ops within the CPU, which requires silicon space and power. For the lowest power CPU, keeping the instruction set simple is paramount. However, higher performance can be obtained from more complex hardware and instructions at the expense of power (*Arm vs x86: Instruction sets, architecture, and all key differences explained*, n.d.).

We are going to comparing the CPU architectures, Intel x86 and ARM, based on the features of data types, operation types, addressing modes, instruction formats/sets. Table 1 gives some of the high level differences. x86 is the name given to Intel processors starting from 8086 processor in 1978. x86 chips are easy to program, use memory efficiently, have limited number of registers, and provide a complex set of instructions. x86 processors are compatible with OS like Windows, LINUX, Android and are widely used in laptops, desktops, servers. ARM processors, first known as Acron RISC machine, now known as Advanced RISC machine, were first developen in 1970s, 1980s. ARM is RISC based while Intel (x86) is CISC based. Since ARM is based on RISC architecture,

it utilizes a small, optimized simple set of instructions and is compatible with OS like LINUX, Android and is used in smartphones, iPads, tablets. ARM's CPU instructions are atomic, with a close correlation between the number of instructions and micro-ops. CISC, by comparison, offers more instructions, many of which execute multiple operations (like optimized math and data movement). This leads to better performance, but more power consumption decoding these complex instructions. The fundamental difference between ARM's and Intel's approaches to CPU design is that x86 targets peak performance while ARM processors aim for better energy efficiency. (Blem, Menon, & Sankaralingam, 2013; K & Arur, n.d.).

1.1 Data Types

Intel x86 Supports data types of 8 (byte), 16 (word), 32 (doubleword), 64 (quadword), and 128 (double quadword) bits in length. The signed integers are in two's complement representation and may be 16, 32, or 64 bits long. The floating point type refers to a set of types that are used by the floating-point unit and operated on by floating-point instructions. The three floating-point representations (Single precision, Double precision, and Double extended precision floating point) conform to the IEEE-754 standard. The words need not be aligned at even numbered addresses; doublewords need not be aligned at addresses evenly divisible by 4; and so on. When data is accessed across a 32-bit bus, data transfers take place in units of doublewords, beginning at addresses divisible by 4. x86 uses the little-endian style; that is, the least significant byte is stored in the lowest address. Byte, word, doubleword, quadword, and double quadword are referred to as general data types. Other supported data types are Integer, Ordinal, Unpacked binary coded decimal (BCD), Packed BCD, Near pointer, Far pointer, Bit field, Bit string, Byte string, floating point packed SIMD (single instruction, multiple data) (Stallings, 2003).
ARM supports data types of 8 (byte), 16 (halfword), and 32 (word) bits in length. For all three data types (byte, halfword, and word) an unsigned interpretation is supported, in which the value represents an unsigned, nonnegative integer. All three data types can also be used for twos complement signed integers. The majority of ARM processor implementations do not provide floating point hardware, which saves power and area. If floating-point arithmetic is required in such processors, it must be implemented in software. ARM does support an optional floating-point co-processor that supports the single- and double-precision floating point data types defined in IEEE-754. Normally, halfword access should be halfword aligned and word accesses should be word aligned. For nonaligned access attempts, the architecture supports three alternatives. In the default case, the address is treated as truncated, with address bits[1:0] treated as zero for word accesses, and address bit[0] treated as zero for halfword accesses. Alignment checking is the second alternative where the appropriate control bit is set, a data abort signal indicates an alignment fault for attempting unaligned access, the third alternative is the processor uses one or more memory accesses to generate the required transfer of adjacent bytes transparently to the

programmer. ARM Endian Support—Word Load/Store with E-Bit (Stallings, 2003).

1.2 Operation Types

X86 provides a complex array of operation types, including specialized instructions. In **Intel x86**, load and store are incorporated in instructions itself. Most of these are the conventional instructions found in machine instruction sets like data movement (MOV, PUSH, In, OUT), arithmetic and logical (ADD, SUB, MUL, AND, ROL/ROR), control transfer (JMP, JE/JZ, LOOPE/LOOPZ), string (MOVS, LODS) operations. It also provides several types of instructions, tailored to the x86 architecture. Call/Return instructions support procedure call/return(CALL, ENTER, LEAVE, RETURN). Memory Management instructions deal with memory segmentation and allow local and global segment tables (descriptor tables) to be loaded and read, and for the privilege level of a segment to be checked and altered. In the x86 architecture, status flags are set by arithmetic and compare operations. The compare operation subtracts two operands, as does a subtract operation, the difference being that a compare operation only sets status flags, whereas a subtract operation also stores the result of the subtraction in the destination operand. In 1996, Intel introduced MMX technology into its Pentium line. MMX is set of optimized instructions for multimedia tasks. There are 57 new instructions that treat data in a SIMD (single-instruction, multipledata) fashion, which makes it possible to perform the same operation, like addition or multiplication, on multiple data elements at once. Each instruction takes a single clock cycle to execute. For the proper application, these fast parallel operations can yield a speedup of two to eight times over comparable algorithms that do not use MMX instructions (Stallings, 2003).

ARM architecture provides a collection of operation types including load and store, branch, data-processing, multiply, parallel addition, subtraction, extend, and status register access instructions. In ARM, only load and store instructions access memory. The ARM architecture supports two types of instruction that load or store the value of a single register, or a pair of registers, from or to memory: (1) load or store a 32-bit word or an 8-bit unsigned byte, and (2) load or store a 16-bit unsigned halfword, and load and sign extend a 16-bit halfword or an 8-bit byte. Arithmetic and logical instructions are performed only on registers and immediate values encoded in the instruction. This limitation is characteristic of RISC design. The ARM architecture defines four condition flags that are stored in the program status register: N, Z, C, and V (Negative, Zero, Carry and Overflow), with similar meanings as the S, Z, C, and V flags in the x86 architecture. These four flags constitute a condition code in ARM. In addition to the normal data processing and multiply instructions, there are a set of parallel addition and subtraction instructions, in which portions of two operands are operated on in parallel. For example, ADD16 adds the top halfwords of two registers to form the top halfword of the result and adds the bottom halfwords of the same two registers to form the bottom halfword of the

result. These instructions are useful in image processing applications, similar to the x86 MMX instructions. ARM supports a branch instruction that allows a conditional branch forwards or backwards up to 32 MB. As the program counter is one of the general-purpose registers (R15), a branch or jump can also be generated by writing a value to R15. A subroutine call can be performed by a variant of the branch instruction. As well as allowing a branch forward or backward up to 32 MB, the Branch with Link (BL) instruction preserves the address of the instruction after the branch (the return address) in the LR (R14). Branches are determined by a 4-bit condition field in the instruction (Stallings, 2003).

1.3 Addressing Modes

The **x86** has a variety of addressing modes which allow efficient execution of high-level languages. For the immediate mode, the operand is included in the instruction. The operand can be a byte, word, or doubleword of data. For register operand mode, the operand is located in a register. For general instructions, like data transfer, arithmetic, and logical instructions, the operand can be one of the 32-bit general registers, one of the 16-bit general registers, or one of the 8-bit general registers. There are also some instructions that reference the segment selector registers. The addressing modes that reference memory locations are Displacement, Base, Base with displacement, Scaled index with displacement, Base with index and displacement, Base scaled index with displacement, and Relative addressing modes. A segment register is used for all except relative (PC). In the displacement mode, the operand's offset is contained in the instruction as an 8-, 16-, or 32-bit displacement. It can lead to long instructions, especially for 32 bit. Can be used to reference global variables. Indirect addressing means that the address portion of the instruction tells the processor where to look, to find the address. The base mode specifies that one of the 8-, 16-, or 32-bit registers contains the effective address (Stallings, 2003).
A typical RISC characteristic is a small and simple set of addressing modes. **ARM** departs from this convention with a relatively rich set of addressing modes: Load and Store (done indirectly through base register plus offset), Data Processing Instruction (either register addressing or a mixture of register and immediate addressing), Branch Instruction (only form of addressing is immediate addressing), Load/Store multiple. Load and Store are the only instructions that can reference memory. This is always done indirectly through a base register plus offset. There are three alternatives with respect to indexing, namely, Offset, Preindex, and Postindex. ARM refers to as a base register acts as an index register for preindex and postindex addressing. The offset value can be an immediate value stored in the instruction or in another register. If the offset value is in a register, another useful feature is: scaled register addressing. The value in the offset register is scaled by one of the shift operators: Logical Shift Left, Logical Shift Right, Arithmetic Shift Right, Rotate Right, or Rotate Right Extended (which includes the carry bit in the rotation). The amount of the shift is specified as an immediate value in the instruction. Data processing instructions

use either register addressing or a mixture of register and immediate addressing. The only form of addressing for branch instructions is immediate addressing. Load Multiple instructions load a subset (possibly all) of the general-purpose registers from memory. Store Multiple instructions store a subset (possibly all) of the general-purpose registers to memory (Stallings, 2003).

1.4 Instruction Formats/Sets

In **x86**, instructions are made up of from zero to four optional instruction prefixes, a 1- or 2-byte opcode, optional address specifier (consists of ModR/M byte and Scale Index Base byte) optional displacement, and an optional immediate field. The prefix bytes are Instruction prefixes, Segment override, Operand size, and Address size. The instruction includes the following fields: Opcode, ModR/M, SIB, Displacement, and Immediate. In x86, the addressing mode is provided as part of the opcode sequence rather than with each operand. Because only one operand can have address-mode information, only one memory operand can be referenced in an instruction. x86 instructions are therefore more compact. x86 allows the use of not only 1-byte, but also 2-byte and 4-byte offsets for indexing. Although the use of the larger index offsets results in longer instructions, this provides flexibility. For example, it is useful in addressing large arrays or large stack frames. The encoding of x86 instruction set is complex. This has to do partly with the need for backward compatible with 8086 machine and partly with a desire of the designers to provide assistance to the compiler writer in producing efficient code.(Stallings, 2003).
All instructions in **ARM** are 32 bits long and follow a regular format. First four bits are condition codes, next three specify general type of instruction. For most instructions other than branches next five bits are opcode and/or modifier bits, remaining 20 bits are for operand addressing. The regular structure simplifies the design of instruction decode units. To achieve a greater range of immediate values, data processing immediate format specifies an immediate value and a rotate value. The 8-bit immediate value is expanded to 32 bits and then rotated right by a number of bits equal to twice the 4-bit rotate value. The Thumb instruction set is a re-encoded subset of the ARM instruction set. Thumb is designed to increase the performance of ARM implementations that use a 16-bit or narrower memory data bus and to allow better code density than the ARM instruction set. Thumb contains a subset of the ARM 32-bit instruction set recoded into 16-bit instructions (Stallings, 2003).

1.5 Example

Considering the example of multiplying two numbers in main memory at locations M1 and M2 and storing back the result in the location M1 (K & Arur, n.d.).
Intel x86 will use the following program:
MULT M1, M2
ARM will use the following program:
LDR A, M1

LDR B, M2
MUL A, B
STR M1, A
In x86, after running the program, if any operand needs to be used for another computation, processor must reload data from memory into register while the operand will remain in register until a new value is loaded for ARM. We can observe that x86 achieves the result using a shorter code, little RAM space required for storage of instructions while ARM program has more lines of code and requires more RAM space. But x86 takes multiple clock cycles to execute the code, requires more transistors, and more hardware space, therefore power consumption is more. ARM requires less transistors, less hardware space, therefore power consumption is less, and also minimizes the number of clock cycles per instruction.

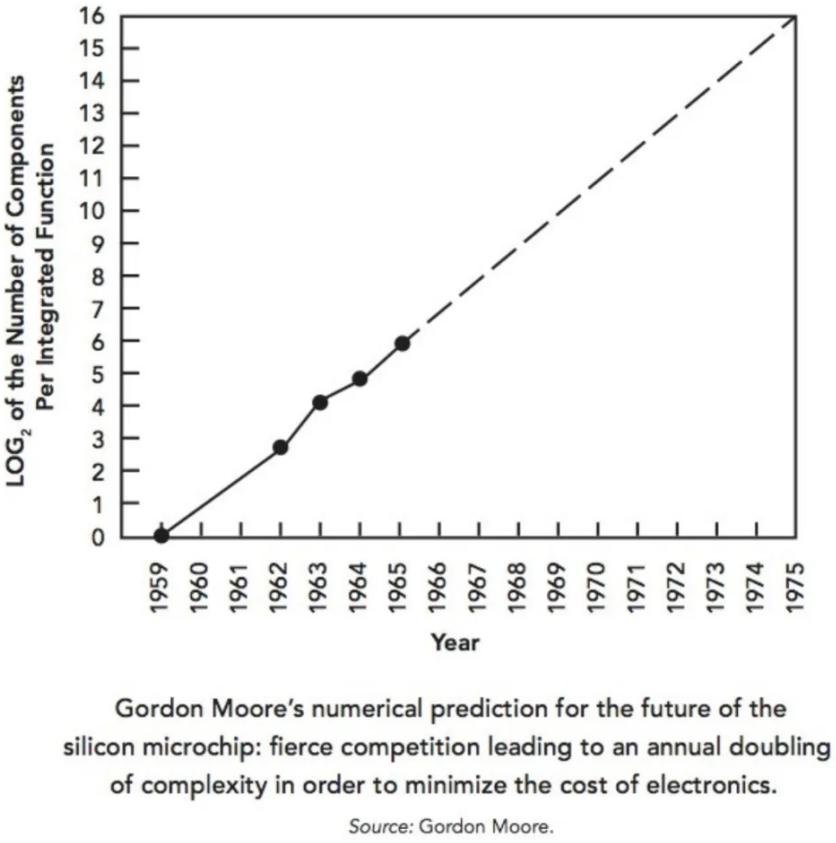

Figure 1: Moore's initial projection for the decade 1965-1975 (*Moore's Law*, n.d.-a)

Table 1: Differences between ARM/RISC and x86/CISC (*Arm vs x86: Instruction sets, architecture, and all key differences explained*, n.d.; Blem et al., 2013; K & Arur, n.d.)

Feature	**ARM**	**x86**
Name	Generic name given to Intel processors	Advanced RISC machine
Year of Development	1978	1970s, 1980s
Architecture	RISC	CISC
Focus	Speed and performance	Power consumption
Operations	Simple, single function operations Single cycle	Complex, multi-cycle instructions Transcendentals Encryption String manipulation
Operands	registers, immediates	memory, registers, immediates
Addressing modes	Few	Many
Registers	16 general purpose registers * 8 32b	6 16b registers
Instruction Format	Fixed length instructions Relatively simple encoding ARM: 4B, THUMB(2B, optional)	Variable length instructions Common insts shorter/simpler Special insts longer/complex x86: from 1B to 16B long
Execution Speed	Executes single instruction per cycle	Executes complex instruction, one at a time and it takes more than a cycle
Approach to performance optimization	Software focused	Hardware approach
Memory usage	Requires less registers, more memory	Uses more registers, less memory
Pipelining	Pipelining of instructions is the unique feature	Less pipelined
Execution time	Faster execution of instructions	Takes more time to execute
Complex addressing	Managed by software	Inherently designed to handle complex addresses
Operation Management	Compiler plays a key role	Microprogram does the trick
Execution of instructions	Multiple instructions are generated from complex one and executed individually	Can execute complex statement
Code expansion	Is difficult	Is easily managed
Instruction decoding	Easy	Complex
Calculations	Uses available memory	Needs supplement memory
Deployed in	mobile devices where size, power consumption, speed matters	servers, desktops, laptops where high performance and stability matters
Example CPU	Cortex-A8, Cortex-A9, Intel Atom, Sandybridge i7 microprocessors	Pentium and AMD processors, Cyrix 386/486S/DLC
Compatible OS	Most OS like Windows, LINUX, Android	OS like LINUX, Android
Used in	laptops, desktops, servers	smartphones, iPads, tablets

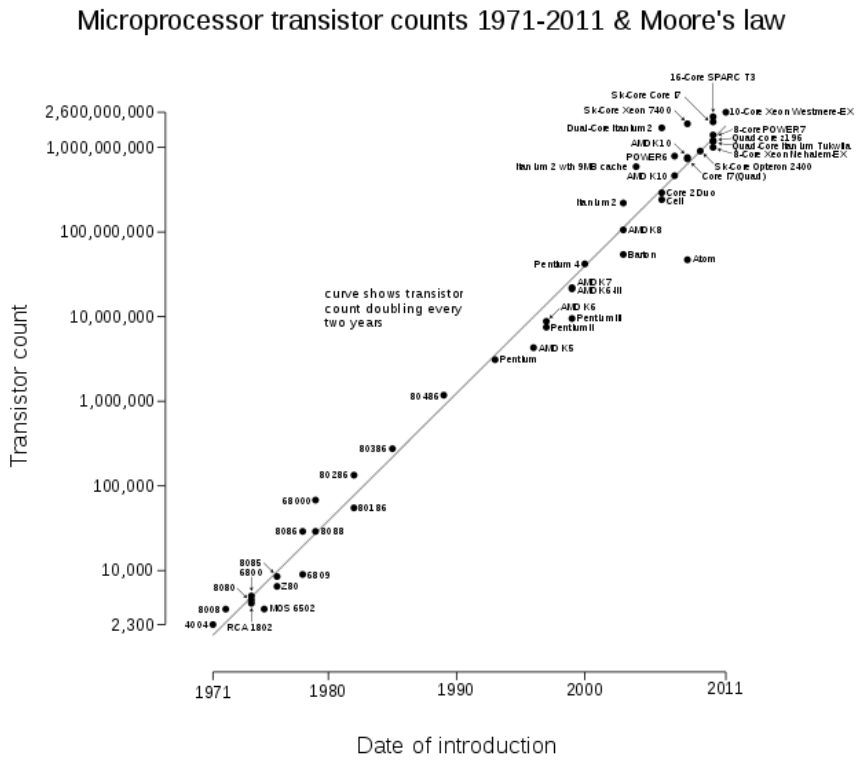

Figure 2: Plot of CPU transistor counts against dates of introduction. Note the logarithmic vertical scale; the line corresponds to exponential growth with transistor count doubling every two years (*Moore's Law*, n.d.-b).

2 Moore's Law

Moore's Law is the prediction that the number of transistors in a dense integrated circuit doubles every two years as technological progress advances. The law is named after Intel co-founder Gordon E. Moore, who described the trend in his 1965 paper. The paper noted that the number of components in integrated circuits had doubled every year from the invention of the integrated circuit in 1958 until 1965 and predicted that the trend would continue "for at least ten years" (Figure 1) (*Moore's Law*, n.d.-a). His prediction has proven to be uncannily accurate, in part because the law is now used in the semiconductor industry to guide long-term planning and to set targets for research and development (Figure 2 (*Moore's Law*, n.d.-b).

2.1 History of Moore's Law

1965: Moore wrote: "The cost per component is nearly inversely proportional to the number of components," so the more the number of transistors, the lower the cost per transistor (*Moore's Law*, n.d.-a). The period often quoted as "18 months" is due to Intel executive David House, who predicted that period for a doubling in chip performance (being a combination of the effect of more transistors and their being faster) (*Moore's Law*, n.d.-b).
1975: Gordon Moore's prediction held for 10 years, later becoming Moore's Law. He then revised his prediction and stated that the number of transistors would double every two years moving forward (*Moore's Law*, n.d.-a).
1975-2015: Moore's Law was applied widely during the decades, making more or less accurate predictions about the number of transistors that could fit on a single integrated circuit. However, many computer scientists, including Moore himself, predicted that the law was coming to an end (*Moore's Law*, n.d.-a).
2015-Present: Over the last few years, the growth of the number of transistors on each IC is declining, falling much lower than what Moore's Law predicts (*Moore's Law*, n.d.-a).
History shows Moore's law is a fluid concept, as opposed to an unchanging one as found in a physical or political law. In fact, Moore's article never used the phrase "Moore's law" or even the word "law." The article did include a cartoon of computers being sold as consumer items in the future, a projection that came to pass for microprocessor-based products. Of course, microprocessors weren't invented until 19714 and it wasn't until 1979 that Carver Mead even coined the phrase "Moore's law." (DeBenedictis, 2017).

2.2 Impact of Moore's Law

Moore's law is based on a 1965 Electronics magazine article by Gordon that makes a research contribution to IC manufacturability but also includes statements that could be considered either technical projections or tantalizing possibilities (DeBenedictis, 2017).
The most widely understood projection from Moore's 1965 paper is that the number of transistors on an IC would double every few years. The article said "shrinking dimensions on an integrated structure makes it possible to operate the structure at higher speed for the same power per unit area." This critical property enabled improvements in IC-based products. Rising device counts and higher speeds or clock rates allowed more features in successive product generations while retaining the product's package and power source (DeBenedictis, 2017).
The capabilities of many digital electronic devices are strongly linked to Moore's law: processing speed, memory capacity, sensors and even the number and size of pixels in digital cameras. All of these are improving at (roughly) exponential rates as well. This exponential improvement has dramatically enhanced the impact of digital electronics in nearly every segment of the world economy. Moore's law describes a driving force of technological and social change in the late 20th and early 21st centuries (*Moore's Law*, n.d.-b).

2.3 Is Moore's Law Dead?

The trend predicted in the Morre's law has continued for more than half a century. Sources in 2005 expected it to continue until at least 2015 or 2020. However, the 2010 update to the International Technology Roadmap for Semiconductors has growth slowing at the end of 2013, after which time transistor counts and densities are to double only every three years (*Moore's Law*, n.d.-b).

Some industry experts believe Moore's Law is no longer applicable. "It's over. This year that became really clear," said Charles Leiserson, a computer scientist at MIT and a pioneer of parallel computing, told MIT Technology Review in February. Moore's Law, Leiserson said, was always about the rate of progress, and "we're no longer on that rate" (*Moore's Law*, n.d.-c). In 2019, Nvidia CEO Jensen Huang declared that Moore's Law is dead and now it's more expensive and more technically difficult to double the number of transistors driving the processing power. That sentiment was also proclaimed a year earlier by Mike Muller, chief technology officer at chip designer ARM (*Moore's Law*, n.d.-c).

Others say not so fast. It may be slowing down, but "the trend is still there," said Karen Panetta, an IEEE fellow and dean of graduate engineering at Tufts University. Many people are holding to the true definition that it has to be transistors on silicon and they have to double every two years, and in that case, that is not happening, she acknowledged. "The name of the game now is the technology may not be traditional silicon transistors; now it may be quantum computing, which is a different structure and nano-biotechnology, which consists of proteins and enzymes that are organic," Panetta said. That means the essence of Moore's Law will likely change given that with quantum computing, "you may end up with exponentially more than" processing power doubling every two years, as well as the use of different materials, she said. Moore's Law will probably be replaced within the next five years—or maybe upgraded based on what comes out of nanobiology or quantum computing, Panetta said (*Moore's Law*, n.d.-c).

Mario Morales, a program vice president at IDC, said he also believes the law is still relevant, in theory. "If you look at what Moore's Law has enabled, we're seeing an explosion of more computing across the entire landscape,'" Morales said. "It used to be computing was centered around mainframes and then it became clients and now edge and endpoints, but they're getting more intelligent, and now they're doing AI inferencing, and you need computing to do that. So Moore's Law has been able to continue to really push computing to the outer edge" (*Moore's Law*, n.d.-c).

Echoing Panetta, Morales said there's been a shift in how we think about Moore's Law. "We're going beyond it now when we think about incremental improvements of software," he said. Because so much of computing now includes AI and machine learning, changes are happening much faster than the previous 18- to 24-month period, Morales said. Morales doesn't think it will be replaced, but rather, augmented. "Moore's Law has been in place for 55 years and it's

still going," he said. "It gets more and more challenging to push the envelope in processing technology—it's still happening, but it's getting more expensive" (*Moore's Law*, n.d.-c).

3 RISC vs. CISC Instruction Set

RISC vs. CISC wars raged in the 1980s when chip area and processor design complexity were the primary constraints and desktops and servers exclusively dominated the computing landscape. Today, energy and power are the primary design constraints and the computing landscape is significantly different: growth in tablets and smartphones running ARM (a RISC instruction set architecture (ISA)) is surpassing that of desktops and laptops running x86 (a CISC ISA). Further, the traditionally low-power ARM ISA is entering the high-performance server market, while the traditionally high-performance x86 ISA is entering the mobile low-power device market (Blem et al., 2013).

3.1 RISC vs. CISC

Considering the example of multiplying two numbers in memory locations M1 and M2 (Section 1.5).

The CISC Approach: The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction, say, MULT. When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction: MULT M1, M2. MULT is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of M1 and "b" represent the value of M2, then this command is identical to the C statement "a = a * b". One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly into the hardware (*RISC vs. CISC*, n.d.-a).

The RISC Approach: RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the MULT command in CISC could be divided into three separate commands: LOAD, which moves data from the memory bank to a register, PROD, which finds the product of two operands located within the registers, and STORE, which moves data from a register to the memory banks. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly

(Section 1.5). At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form (*RISC vs. CISC*, n.d.-a).

However, the RISC strategy also brings some very important advantages. Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle MULT command. These RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers. Because all of the instructions execute in a uniform amount of time, pipelining is possible (*RISC vs. CISC*, n.d.-a).

Separating the LOAD and STORE instructions actually reduces the amount of work that the computer must perform. After a CISC-style MULT command is executed, the processor automatically erases the registers. If one of the operands needs to be used for another computation, the processor must re-load the data from the memory bank into a register. In RISC, the operand will remain in the register until another value is loaded in its place (*RISC vs. CISC*, n.d.-a).

The Performance Equation: The following equation is commonly used for expressing a computer's performance ability:

$$\frac{time}{program} = \frac{time}{cycle} X \frac{cycles}{instruction} X \frac{instructions}{program}$$

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program (*RISC vs. CISC*, n.d.-a).

Despite the advantages of RISC based processing, RISC chips took over a decade to gain a foothold in the commercial world. This was largely due to a lack of software support (*RISC vs. CISC*, n.d.-a).

Although Apple's Power Macintosh line featured RISC-based chips and Windows NT was RISC compatible, Windows 3.1 and Windows 95 were designed with CISC processors in mind. Many companies were unwilling to take a chance with the emerging RISC technology. Without commercial interest, processor developers were unable to manufacture RISC chips in large enough volumes to make their price competitive (*RISC vs. CISC*, n.d.-a).

Another major setback was the presence of Intel. Although their CISC chips were becoming increasingly unwieldy and difficult to develop, Intel had the resources to plow through development and produce powerful processors. Although RISC chips might surpass Intel's efforts in specific areas, the differences were not great enough to persuade buyers to change technologies (*RISC vs. CISC*, n.d.-a).

Today, the Intel x86 is arguable the only chip which retains CISC architecture. This is primarily due to advancements in other areas of computer technology. The price of RAM has decreased dramatically. In 1977, 1MB of DRAM cost about $5,000. By 1994, the same amount of memory cost only $6 (when adjusted for inflation). Compiler technology has also become more sophisticated, so that

the RISC use of RAM and emphasis on software has become ideal.
The short answer is that RISC is perceived by many as an improvement over CISC. There is no best architecture since different architectures can simply be better in some scenarios but less ideal in others. RISC-based machines execute one instruction per clock cycle. CISC machines can have special instructions as well as instructions that take more than one cycle to execute. This means that the same instruction executed on a CISC architecture might take several instructions to execute on a RISC machine. The RISC architecture will need more working (RAM) memory than CISC to hold values as it loads each instruction, acts upon it, then loads the next one.
The CISC architecture can execute one, albeit more complex instruction, that does the same operations, all at once, directly upon memory. Thus, RISC architecture requires more RAM but always executes one instruction per clock cycle for predictable processing, which is good for pipelining. One of the major differences between RISC and CISC is that RISC emphasizes efficiency in cycles per instruction and CISC emphasizes efficiency in instructions per program. A fast processor is dependent upon how much time it takes to execute each clock cycle, how many cycles it takes to execute instructions, and the number of instructions there are in each program. RISC has an emphasis on larger program code sizes (due to a smaller instruction set, so multiple steps done in succession may equate to one step in CISC).
The RISC ISA emphasizes software over hardware. The RISC instruction set requires one to write more efficient software (e.g., compilers or code) with fewer instructions. CISC ISAs use more transistors in the hardware to implement more instructions and more complex instructions as well.
RISC needs more RAM, whereas CISC has an emphasis on smaller code size and uses less RAM overall than RISC. Many microprocessors today hold a mix of RISC- and CISC-like attributes, however, such as a CISC-like ISA that treats instructions as if they are a string of RISC-type instructions.
Some major differences between CISC and RISC architectures are listed in Table 3

4 I/O interface: Infiniband

The InfiniBand Architecture (IBA) is a new industry-standard architecture for server I/O and inter-server communication. It was developed by the InfiniBand Trade Association (IBTA) to provide the levels of reliability, availability, performance, and scalability necessary for present and future server systems, levels significantly better than can be achieved with bus-oriented I/O structures (Pfister, 2001).

4.1 Reasons for the InfiniBand Architecture

While busses have the major advantage of simplicity, and have served the industry well up to this point, bus-based I/O systems do not use their underlying electrical

CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

Figure 3: RISC vs CISC (*RISC vs. CISC*, n.d.-b)

technology well to provide data transfer (bandwidth) out of a system to devices. There are several reasons for this. First, buses are inherently shared, requiring arbitration protocols on each use, a "tax" that increases with the number of devices or hosts. As the clock speed of a bus increases, and its long burst bandwidth increases, the negative effects of arbitration overhead are magnified. A second reason why busses are inefficient users of electrical technology actually is really a function of how they are used. Since common industry standard busses are memory-mapped, short sequences, such as command transmission and the reading status, are performed using processor load and store operations. For store operations, this is not a major problem since modern processors allowing out-oforder completion can overlap substantial additional processing with store processing. Unfortunately, most devices require load instructions for operations like reading status and retrieving small amounts of data. Loads, unlike stores, usually cannot proceed very far without making the processor stop to wait for the

requested data. This can be a very serious efficiency problem. Internal analysis by the author of a competitor's system executing a commercial benchmark requiring many fairly small I/O operations—the TPCC benchmark—indicated that nearly 30% of processor execution time was spent waiting on I/O loads in this fashion. In addition to the above problems, busses also do not provide the level of reliability and availability now being required of server systems. A single device failure can inhibit the correct operation of the bus itself, causing all the devices on the bus to become unavailable, including those attached by bridges. Finding out which device is at fault, or whether there is a failure of the bus itself, often becomes an aggravating and time-consuming exhaustive search.(Pfister, 2001)

4.2 An InfiniBand Architecture Overview

These problems mentioned above have been solved before by individual server vendors in a number of ways. However, all such solutions have been wholly or partially proprietary, thereby incurring significant costs. IBA is, instead, an industry-standard architecture, which should achieve significant volumes and hence much lower costs. It avoids the problems with busses discussed above through two basic characteristics: (Pfister, 2001)
Point-to-point connections: All data transfer is point to-point, not bussed. This avoids arbitration issues, provides fault isolation, and allows scaling to large size by the use of switched networks.
Channel (message) semantics: commands and data are transferred between hosts and devices not as memory operations but as messages. Like any modern communication system, IBA is a stack divided into physical, link, network, and transport layers (Pfister, 2001). The smallest complete IBA unit is a *subnet*, illustrated in Figure 4 (Pfister, 2001). *IBA links* are bidirectional point-to-point communication channels, and may be either copper and optical fibre. *IBA switches* route messages from their source to their destination based on routing tables that are programmed with forwarding information during initialization and network modification. *IBA endnodes* are the ultimate sources and sinks of communication in IBA. The interface between an endnode and a link is a *Channel Adapter (CA)*. IBA provides several different types of communication services between endnodes (Pfister, 2001): Reliable Connection (RC), (Unreliable) Datagram (UD), Unreliable Connection (UC), Reliable Datagram (RD). *IBA management* is defined in terms of managers and agents.

4.3 Summary

- Recent I/O specification aimed at high-end server market
- First version released early 2001
- Standard for data flow between processors and intelligent I/O devices
- Intended to replace PCI bus in servers

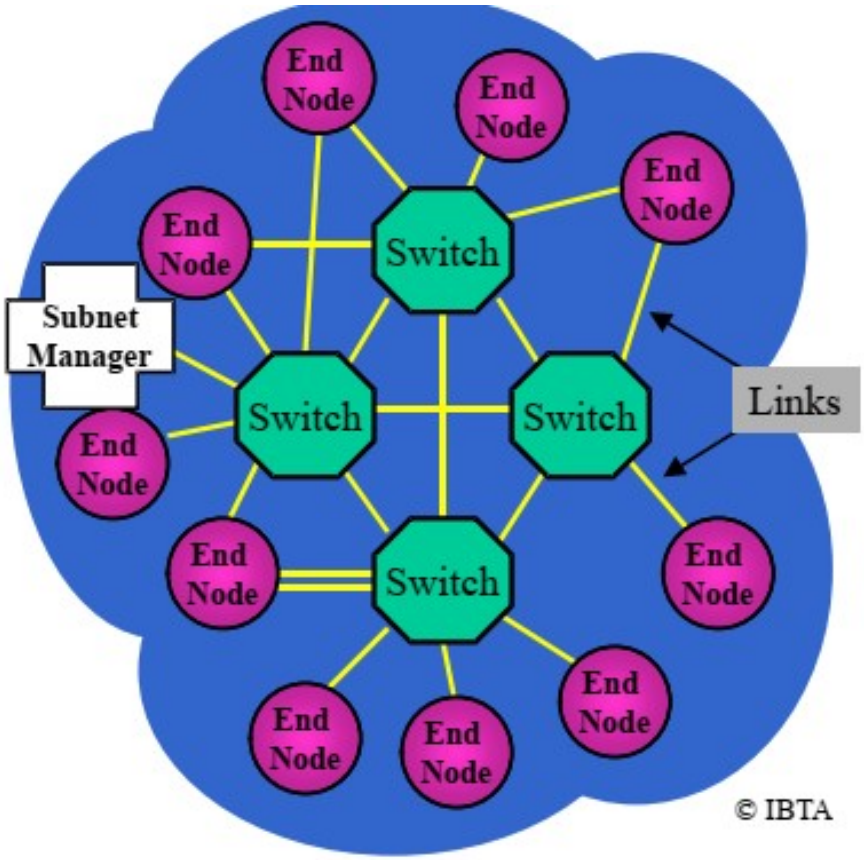

Figure 4: An InfiniBand Architecture Subnet (Pfister, 2001)

- Greater capacity, increased expandability, enhanced flexibility
- Connect servers, remote storage, network devices to central fabric of switches and links
- Greater server density
- Independent nodes added as required
- I/O distance from server up to 17 meters using copper, 300 meters using multimode optical fiber, 10 kilometers using single-mode optical fiber
- Transmission rates up to 30 Gbps (Putnam, n.d.)

InfiniBand Operations:

- 16 logical channels (virtual lanes) per physical link

- One lane for fabric management – all other lanes for data transport
- Data sent as a stream of packetsVirtual lane temporarily dedicated to the transfer from one end node to another
- Switch maps traffic from incoming lane to outgoing lane (Putnam, n.d.)

5 I/O Interface: Thunderbolt

The most recent, and fastest, peripheral connection technology to become available for general-purpose use is Thunderbolt, developed by Intel with collaboration from Apple. One Thunderbolt cable can manage the work previously required of multiple cables. The technology combines data, video, audio, and power into a single high-speed connection for peripherals such as hard drives, RAID (Redundant Array of Independent Disks) arrays, video-capture boxes, and network interfaces. It provides up to 10 Gbps throughput in each direction and up to 10 Watts of power to connected peripherals. Although the technology and its associated specifications have stabilized, the introduction of Thunderbolt-equipped devices into the marketplace has, only slowly begun to develop. This is because a Thunderbolt-compatible peripheral interface is considerably more complex than that of a simple USB device. The first generation of Thunderbolt products are primarily aimed at the prosumer (professional-consumer) market such as audiovisual editors who want to be able to move large volumes of data quickly between storage devices and laptops. As the technology becomes cheaper, Thunderbolt will find mass consumer uses, such as enabling very high-speed data backups and editing high-definition photos. Thunderbolt is already a standard feature of Apple's MacBook Pro laptop and iMac desktop computers (Stallings, 2003).

6 I/O Interface: Firewire

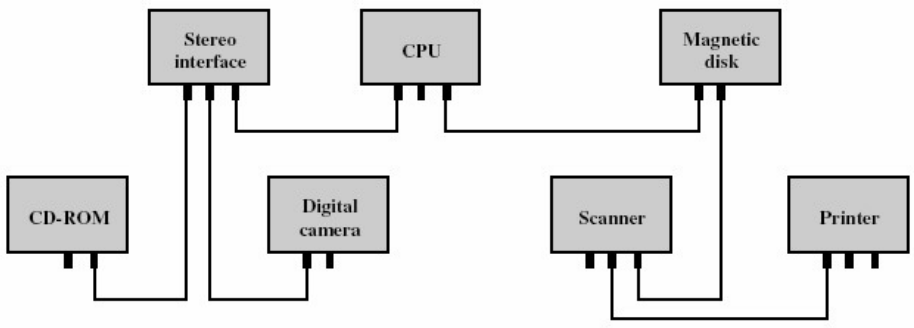

Figure 5: Firewire.

FireWire Serial Bus – IEEE 1394 (Putnam, n.d.):

- Very high speed serial bus
- Low cost
- Easy to implement
- Used with digital cameras, VCRs, and television (Figure 5)

FireWire Configurations (Putnam, n.d.):

- Daisy chain
- 63 devices on a single port – 64 if you count the interface itself
- 1022 FireWire busses can be interconnected using bridges
- Hot plugging
- Automatic configuration
- No terminations
- Can be tree structured rather than strictly daisy chained

FireWire three layer stack (Putnam, n.d.):

- Physical layer:
  - Defines the transmission media that are permissible and the electrical and signaling characteristics of each
  - 25 to 400 Mbps
  - Converts binary data to electrical signals
  - Provides arbitration services
  - Based on tree structure
  - Root acts as arbiter
  - First come first served
  - Natural priority controls simultaneous requests – nearest root
  - Fair arbitration
  - Urgent arbitration
- Link layer:
  - Describes the transmission of data in the packets
  - Asynchronous
    1. Variable amount of data and several bytes of transaction data transferred as a packet

2. Uses an explicit address
3. Acknowledgement returned

– Isochronous

1. Variable amount of data in sequence of fixed sized packets at regular intervals

2. Uses simplified addressing

3. No acknowledgement

- Transaction layer :
  - Defines a request-response protocol that hides the lower-layer detail of FireWire from applications

References

Arm vs x86: Instruction sets, architecture, and all key differences explained. (n.d.). Retrieved from `https://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/` (Available at https://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/, Accessed: 2021-04-9)

Blem, E., Menon, J., & Sankaralingam, K. (2013). A detailed analysis of contemporary arm and x86 architectures. *UW-Madison Technical Report*.

DeBenedictis, E. P. (2017). It's time to redefine moore's law again. *Computer*, *50*(2), 72–75.

K, M., & Arur, V. (n.d.). *Performance comparison between x86 and arm assembly.* Retrieved from `https://www.slideshare.net/ManasaSushmitha/x86-and-arm-performance-comparison` (Available at https://www.slideshare.net/ManasaSushmitha/x86-and-arm-performance-comparison, Accessed: 2021-04-9)

Moore's law. (n.d.-a). Retrieved from `https://corporatefinanceinstitute.com/resources/knowledge/other/moores-law/` (Available at https://corporatefinanceinstitute.com/resources/knowledge/other/moores-law/, Accessed: 2021-04-9)

Moore's law. (n.d.-b). Retrieved from `https://www.kth.se/social/upload/507d1d3af276540519000002/Moore%27s%20law.pdf` (Available at https://www.kth.se/social/upload/507d1d3af276540519000002/Moore%27s%20law.pdf, Accessed: 2021-04-10)

Moore's law. (n.d.-c). Retrieved from `https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant/` (Available at https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant/, Accessed: 2021-04-9)

Pfister, G. F. (2001). An introduction to the infiniband architecture. *High performance mass storage and parallel I/O*, *42*(617-632), 102.

Putnam, R. C. (n.d.). *Computer organization & architecture lecture #19.* University Lecture. Retrieved from `https://www.ecs.csun.edu/~cputnam/Comp546/Input-Output-Web.pdf` (Available at https://www.ecs.csun.edu/ cputnam/Comp546/Input-Output-Web.pdf, Accessed: 2021-04-9)

Risc vs. cisc. (n.d.-a). Retrieved from `https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/` (Available at https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/, Accessed: 2021-04-10)

Risc vs. cisc. (n.d.-b). Retrieved from `https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/` (Available at https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/, Accessed: 2021-04-10)

Stallings, W. (2003). *Computer organization and architecture: designing for performance*. Pearson Education India.