1

0.1 Preface

This textbook is an open-source document that meets the requirements of COMP 142 Computer Architecture and Organization. The goal of the class is to study the organization and behavior of real computer systems at the assembly-language level, the mapping of statements and constructs in a high-level language onto sequences of machine instructions is studied, as well as the internal representation of simple data types and structures. It also covers numerical computation, noting the various data representation errors and potential procedural errors.

The general topics covered by the class are:

1. Bits, bytes, and words
2. Numeric data representation and number bases
3. Fixed- and floating-point systems
4. Signed and twos-complement representations
5. Representation of nonnumeric data (character codes, graphical data)
6. Representation of records and arrays
7. Basic organization of the von Neumann machine
8. Control unit; instruction fetch, decode, and execution Instruction sets and types (data manipulation, control, I/O)
9. Assembly/machine language programming
10. Instruction formats
11. Addressing modes
12. Subroutine call and return mechanisms
13. I/O and interrupts

Chapter 1

Number Bases

A positional system is a numeral system in which the contribution of a digit to the value of a number is the value of the digit multiplied by a factor determined by the position of the digit. In early numeral systems, such as Roman numerals, a digit has only one value: I means one, X means ten and C a hundred (however, the value may be negated if placed before another digit). In modern positional systems, such as the decimal system, the position of the digit means that its value must be multiplied by some value: in 555, the three identical symbols represent five hundreds, five tens, and five units, respectively, due to their different positions in the digit string.

Among the earliest systems was the Babylonian numeral system. It used base 60. It was the first positional system to be developed, and its influence is present today in the way time and angles are counted in tallies related to 60, such as 60 minutes in an hour and 360 degrees in a circle. Today, the Hindu–Arabic numeral system (base-10) is the most commonly used system globally. However, the binary numeral system (base-2) is used in almost all computers and electronic devices because it is easier to implement efficiently in electronic circuits.

1.1 Base-2

You can think of a number base as the way to represent a number. The value of the number does not change when transitioning between number bases. Most people use a base of 10, also known as decimal. For example, 667 is a decimal number. When you work problems that have numbers of different bases, write the number in subscript to avoid confusion, for example: 667_{10} the number 667 written in decimal. Binary numbers are base 2. 667_{10} in

Western Arabic	0	1	2	3	4	5	6	7	8	9
Eastern Arabic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
East Asian	零	一	二	三	四	五	六	七	八	九

Figure 1.1: Many cultures use base-10 for arithmetic, differing only by the symbol used to represent each digit. Numerals used in Western culturals are formally called Western Arabic numerals

base 2 is representated as 1010011011_2.

When spoken, binary numerals are usually read digit-by-digit, in order to distinguish them from decimal numerals. For example, 1010011011_2 is pronounced *one zero one zero zero one one zero one one*. It would be confusing to refer to the number as *one billion ten million eleven thousand and eleven* which represents a different value. Note that the number base should not change the intrinsic value of a number.

The order of the digits in a binary number represents their power of two. The right-most digit is called the least significant bit (LSB) represents the power of 2^0. The left-most digit is called the most significant bit (MSB) represents the power of 2^{n-1} where n is the length of the sequence. For example, with 1010011011_2 the LSB is 1, representing 2^0, and the MSB is 2^9 because the sequence is 10 bits long.

Example 1.1 Convert 1010011011_2 to decimal by expanding the powers of two. The most straightforeward way to convert this number is to expand it in of powers of two:

$$1 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (1.1)$$

Note that we write out all powers of two starting with $n - 1$. In this case $n = 10$ because there ten digits. Begining with the MSB and ending with the LSB copy 0 or 1 based on the corresponding digit in the binary representation of the number. This may seem cumbersome but after some time you will memorize the powers of 2 and it will become easier.

Expanding the number in terms of powers of two is the simplest way to convert a decimal number to binary (such as in Equation 1.1). To do so, prepare a table with the power of two that is just less than the magnitude of the number you are converting. For 667_{10}, this is 512_{10} or 2^9:

Power	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	512	256	128	64	32	16	8	4	2	1
Digit										

Start from left to right. If you can subtract the number without it becoming negative, indicate 1 for the digit. Then, carry out the subtraction and use this new value for the next column. If you cannot carry out the subtraction without the number becoming negative, skip to the next column. Repeat this procedure for the next column.

$667 - 512 = 155$, so we can indeed subtract 512_2 from 667_2. We can note 1 in the digit, and use the value of 155 for the next column. $155 - 256 < 0$. We cannot subtract without the number becoming negative, so we note 0 as the digit for this current column. However, $155 - 128 = 27$. So we can note 1 in the digit for this next column. And so on until the last digit. If you have any remaining value beyond the right-most column you have made a mistake, check your work.

Power	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	512	256	128	64	32	16	8	4	2	1
Digit	1	0	1	0	0	1	1	0	1	1

There is a more formal way to carry out this procedure, called the **divide-by-2** method. Essentially, with the previous algorithm the digit column is noting if there would be a remainder when carrying out integer division by its respective power of 2. An alternative way to convert the number is as follows.

Start with the number you want to convert. Perform an integer division by 2. Note that an integer division does not produce a fractional result. It should produce an integer and a remainder if the divisor cannot cleanly divide the dividend. Write the remainder to the right of the calculation, and write the result *below* your calculation. Continue dividing by 2 until you reach a dividend of 1. The binary representation of the number is read from top-to-bottom of the remainder values.

Homework Questions

1.1 Conduct a search on the historical basis for binary numbers before they were used in computing and explain how they were used by society.

1.2 Explain in your own words the following concepts:

   (a) Most significant bit
   (b) Least significant bit
   (c) The largest unsigned number that can be stored in n bits.

1.3 Define the divide-by-2 method in pseudo-code.

1.4 What is the largest unsigned number that can be stored in the following data types?

   (a) `char`
   (b) 2-byte `int`
   (c) 48-bit integer

1.5 Convert the following decimal numbers to binary:

   (a) 0
   (b) 1
   (c) 23
   (d) 59
   (e) 100
   (f) 1200
   (g) 1092
   (h) 1000000

1.6 Convert the following binary numbers to decimal using the table method:

   (a) 0
   (b) 1
   (c) 1010
   (d) 1111
   (e) 1110101
   (f) 1010100101
   (g) 1000000000
   (h) 1000100101

1.7 Repeat Question 1.6 using the divide-by-2 method.

Chapter 2

sec:datarepresentation

Computers use binary numbers because they are made up of electronic components called transistors, which can be either in an “on” or “off” state. By using a binary numbering system, which only has two digits (0 and 1), computers can represent and process information using these “on” and “off” states of transistors.

A binary number is a number expressed in the base-2 numeral system, a method of mathematical expression which uses only two symbols: typically 0 or 1. In more mathematical or logical contexts 0 is treated as False and 1 is treated as True.

The base-2 numeral system is a positional notation with a base of 2. Each digit is referred to as a bit, or binary digit. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used by almost all modern computers and computer-based devices, as a preferred system of use, over various other human techniques of communication, because of the underlying transitor-level components.

The modern binary number system was studied in Europe in the 16th and 17th centuries by Thomas Harriot, Juan Caramuel y Lobkowitz, and Gottfried Leibniz. However, systems related to binary numbers have appeared earlier in multiple cultures including ancient Egypt, China, and India. Leibniz was specifically inspired by the *I Ching*, a classical Chinese text from the 9th century BCE.

The purpose of thise chapter is to understand binary numbers. Examples will describe how to convert from decimal numbers to binary. Alternative representations, such as hexadecimal, will also be covered. Finally, TODO: how they are grouped in the computer using bytes or bibytes,

It is important to note the largest positive integer that can be stored for a binary sequence of a given length to avoid a concept called overflow. For unsigned numbers this is defined as:

$$2^n - 1 \tag{2.1}$$

Where n is the number of digits in the binary sequence.

Example 2.2 What is the largest number that you can store in an unsigned C-language `int` data type? Note that `int` is 32 bits, so $n = 32$

$$2^n - 1$$
$$2^{32} - 1 = 4,294,967,295$$

Perhaps this is somewhat shocking. A C-language `int` cannot hold numbers greater than four billion.

Example 2.3 Suppose that you want to use binary numbers to encode specific letters of the Spanish language alphabet which has 27 letters. At least how many bits will you need to uniquely encode each letter?

$$\begin{aligned}
&2^n - 1 \geq 27\\
&2^n \geq 28\\
&n \geq \log_2 28\\
&n \geq 4.8 \approx 5
\end{aligned}$$

You cannot have a fractional number of digits, so it must be at least 5 bits.

Chapter 3

Microprocessor Architecture

The goal of a microprocessor is to execute a given program. Programs are a sequence of instructions stored in a binary format, in a particular language, called an instruction set architecture (ISA). Generally one ISA is incompatible with another ISA. For example, if you had a program for a PowerPC Macintosh, it would not run on an Acorn RISC Machine (ARM64) Macintosh. The ISA defines the interface for a set of instructions, but the underlying hardware implementation can vary. For example, AMD and Intel microprocessors can vary in performance due to hardware optimizations.

The microprocessor is fed instructions from the program line-by-line as input. It performs some action based on the directions given in the instruction. It provides some output, also in binary. Though, some operations such as control of flow may not result in a distinct output from the system.

All ISAs regardless of hardware-level implementation are based on the von Neumann architecture–also known as the von Neumann model or Princeton architecture. The von Neumann architecture was defined in a 1945 technical description of Electronic Discrete Variable Automatic Computer (EDVAC), one of the earliest digital computers, designed by John von Neumann and others. A von Neumann architecture is a digital computer with the following components:

- A processing unit with both an arithmetic logic unit (ALU) and processor registers,
- A control unit including an instruction register and a register called the instruction pointer (IP),
- A single memory containing data and instructions,
- Input and output mechanisms, and
- External mass storage.

Before the invention of the von Neumann architecture, computers used discrete plugboard wiring or fixed control circuitry to store the entire program. Examples of this persist in modern times, such as simple non-graphing calculators where the software cannot be updated. Reprogramming early fixed-program machines required redesign of the whole system. The von Neumann architecture allowed for programs to be defined by a set of instructions (ISA) and for the program to be stored in a memory to be executed. The term stored program computer was coined to refer to computers that could a program stored in its memory.

An overview is given in Figure 3.1. The main components are connected by a shared bus. The term von Neumann architecture has evolved to refer to any stored program computer in which an

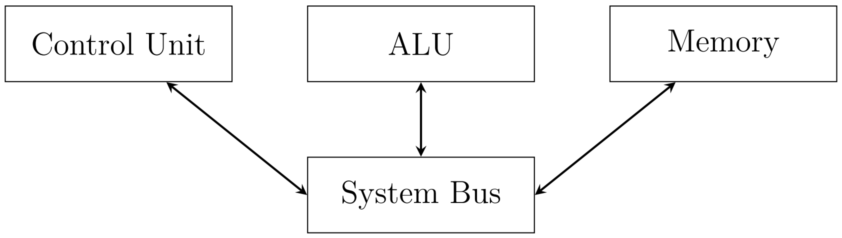

Figure 3.1: Description of the von Neumann architecture.

instruction fetch and a data operation cannot occur at the same time (since they share a common bus). This is referred to as the von Neumann bottleneck, which often limits the performance of the corresponding system. There are other architecture designs such as the Harvard architecture which distinguishes between memory for data and instructions. It called for separate busses to access instructions and data. However, the vast majority of microprocessors developed in the 21st century have followed the von Neumann architecture.

3.1 Arithmetic Logic Unit

In computing, an arithmetic logic unit is a digital circuit that performs arithmetic and bitwise operations on integer binary numbers. Because binary numbers are a positional numbering system it implements positional arithmetic algorithm (carry-in/out) at the hardware level. This is in contrast to a floating point unit (FPU), which operates on floating point numbers which is beyond the scope of this class. arithmetic logic units are a fundamental building block of many types of computing circuits, including the microprocessor, FPU, and graphics processing unit (GPU).

Some of the operations a ARM64 processor can carry out are:

- Simple arithmetic operations, such as addition, subtraction, division, negation, etc.
- Complex arithmetic operations consisting of multiple types of operation. For example, the `madd` operation multiplies two operands, adds a third, and stores the result in a forth.
- Instructions to move values between registers.
- Evaluating the expression of a conditional branch.

The inputs to an ALU are the data to be operated on, called operands, and a operation code, also known as opcode, indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs and control lines. Status inputs or outputs convey information about a previous operation or the current operation, respectively, between the ALU and external status registers. Control lines are additional data beyond the opcode to ensure the correct operation is carried out. With load-and-store architectures, the type of microprocessor architecture we will study in class, the arithmetic logic unit can only perform operations on values in registers, a special type of memory.

3.1.1 Registers

Registers are a concept perhaps unknown to individuals who have just completed their introductory programming courses. A processor register is a quickly accessible location available to a computer's

Table 3.1: List of registers in ARM64 and their purpose.

Address(es)	Purpose
31	Stack pointer (SP) or reference zero depending on context
30	Return address (RA), used to return from subroutines
29	Frame pointer (FP)
19-29	Scratch registers, saved by the callee
16-18	Reserved
9-15	Scratch registers, unsaved by the callee
8	Reserved
0-7	Arguments passed for input and output with subroutines

processor. It is faster than memory due to modern microprocessors using SRAM memory technology for registers. There are a fixed number of registers. Each register is fixed in size. Registers used by the program are general purpose, although some registers have specific hardware functions, and may be read-only or write-only. Some common purposes for registers are given below:

- Scratch registers are for general purpose use.
- Reserved registers used by the operating system that should not be modified by the program.
- Flag registers that cannot be modified directly but hold important information such as if overflow occured. These are often used when handling exceptions.
- Reference values that will always be a fixed value when used as an operand, such as zero.
- Pointers to important parts of memory, e.g. the next instruction, the bottom of the stack, where to return from a subroutine.
- Registers for passing arguments between subroutines.

The guidelines for using registers are defined by a calling convention. The registers for ARM64 are given in 3.1. In the past, ISAs assigned memory addresses to registers. With modern systems, registers have their own addressing system, ussually a compact addressing system with as few bits as needed. Note that the register addresses specified in Table 3.1 would be given in binary.

When a program references datum in memory there is a high probability the value will be used or modified again. This is called temporal locality. Random access of memory is comparatively slow in modern microprocessors due to the type of technology used (dynamic random access memory (DRAM)). You will always need to reference memory at least once per datum, but can use registers as an intermediate structure to hold their value, reducing the number of references performed per program. Holding frequently used values in registers can be critical to a program's performance. Register allocation is performed either by a compiler in the code generation phase, or manually by an assembly language programmer.

3.1.2 Reduced Instruction Set Computing (RISC)

In the past, there was a need to distinguish between two different types of ISAs: reduced instruction set computer (RISC) and complex instruction set computer (CISC). A RISC is a computer architecture designed to simplify the individual instructions given to the computer to accomplish tasks. Compared to the instructions given to a CISC, a RISC might require more instructions (more code) in order to accomplish a task because the individual instructions are written in simpler code. The goal is to offset the need to process more instructions by increasing the speed of each instruction, in particular by implementing an instruction pipeline, which may be simpler given simpler instructions.

The key operational concept of the RISC computer is that each instruction performs only one function (e.g. copy a value from memory to a register). The RISC computer has tens of general-purpose registers and the code uses a load-and-store architecture in which the code for the performing arithmetic are separate from the instructions that grant access to memory of the computer. The design of the CPU allows RISC computers few simple addressing modes and predictable instruction times that simplify design of the system as a whole.

Consider the C++ instruction `i++`. In ARM64, a type of RISC, this requires three instructions:

```
ldr w9 [x0] ; Dereference a pointer
add w9 w9 #1 ; Add one
str w9 [x0] ; Place result back into memory
```

The first instruction fetches the value of `i` from memory. The second instruction adds one to the value. The third instruction stores the modified value in memory. In AMD64, a type of complex instruction set computer (CISC), the same instruction is:

```
add (%rbx), 1 ; Do everything
```

With the later example, the microprocess is instructed to increment a value in memory. It fetches the value, increments it, then stores the result back in memory. This is carried out using a single instruction.

It may not seem obvious why one would want a RISC. However, bundling many atomic operations in a single hardware-level instruction increases the complexity of the design. CISCs tend to be more expensive, and are hard to study. With a RISC, the design is simple enough to draw on paper. This is the why we focus on ARM64 for this introductory class.

With modern microprocessors, the lines between RISC and CISC are blurred. Almost all computers, whether load-and-store architecture or not, load data from a larger memory into registers where it is used for arithmetic operations and is manipulated or tested by machine instructions. Manipulated data is then often stored back to main memory, either by the same instruction or by a subsequent one. Finally, ARM64 has a few instructions that violatic the principle of single-function instructions. For example, a common instruction we will see in the future is:

Aside: SRAM vs. DRAM It may come as quite a shock to PC builders, but the main memory on your motherboard is too slow for the microprocessor. Current memory technologies are SRAM, DRAM and flash. Flash is mostly used for secondary storage. SRAM is a type of memory technology where a single bit is constructed from logic gates. It is fast, but has a comparatively high number of transistors per bit. Thus, it requires a lot of physical space on the microprocessor and costs more than other technologies. DRAM on the other hand uses a single transistor and capacitor to represent a bit. However, due to capacitor's tendency to dissipate over time, DRAM must be refreshed periodically. This is done in chunks, called rows. The need to organize data in rows leads to a not-so-straightforeward method to randomly access data. So, it is slower than SRAM, but it is cheaper and requires less physical space.

```
stp x29, x30, [sp, -32]!
```

This instruction moves the SP by 32 bytes, and performs two register-to-memory storage operations, as a single hardware-level instruction. It would require more instructions to do this in AMD64 and it has no single equivalent instruction.

Acronyms

CISC complex instruction set computer. 12

LSB least significant bit. 4

MSB most significant bit. 4

Glossary

AMD Advanced Micro Devices, a semiconductor manufacturer, developed the AMD64 ISA. 9

AMD64 The 64-bit extension of the x86-architecture, backward compatible with x86, also known as x86-64. 12, 13

arithmetic logic unit The component of a microprocessor responsible for computing arithmetic. 9, 10

ARM64 The 64-bit extension of the Advanced/Acorn RISC machine (ARM) architecture family, also known as ARM64. 9–12

callee When a subroutine calls another subroutine, the callee is the subroutine that has been called. 11

calling convention A set of requirements based on the operating system and ISA defining standards for how to use the registers and stack. 11

CISC A complicated type of microprocessor with a large instruction set where a single instruction may be capable of performing multiple operations at once. 11, 12

control of flow A type of operation that causes the microprocessor to execute some other instruction, other than the next one. 9

control unit The component of a microprocessor that controls the operation of the microprocessor. 9

DRAM Dynamic random access memory, a simple type of memory technology that has difficulties with random access. 11, 12

EDVAC Electronic Discrete Variable Automatic Computer, one of the earliest electronic computers. 9

FP Frame pointer, pointing the the procedure call's current frame. 11

FPU A special purpose ALU that operates on floating-point values. 10

GPU A special purpose ALU optimized for executing math operations commonly used in computer graphics applications. 10

Harvard architecture A model for computing that differs from the von Neumann architecture by distinguishing between instruction memory and data memory. 10

instruction pointer A register in the microprocessor that holds the address of the next instruction to be executed. 9

instruction register A register in the microprocessor that holds the value of the current instruction. 9

Intel A semiconductor manufacturer, developed the x86 ISA. 9

ISA Instruction set architecture. The term for a specific microprocessor design, and its set of instructions. 9, 11

load-and-store architecture A type of architecture where memory operations are distinct from other types of operations. 10, 12

opcode Input to the ALU description the type of operation to be carried out. 10

overflow Exceeding the capacity of a binary number with fixed number of digits. 7

PowerPC A microprocessor made by IBM briefly used as the microprocessor for Macintosh computers. 9

RA The return address, points to where a procedure call should return on completion. 11

reference zero A special register that will be read as zero when used as an operand, or will throw away the result if used as a destination. 11

register Fast temporary storage physically located on the microprocessor. 9–12

RISC A simple microprocessor where instructions generally perform only singular actions resulting in a smaller instruction set. 11, 12

SP Stack pointer, pointing the the next available space on the stack. 11, 13

SRAM Static random access memory, a fast but comparatively expenstive type of memory technology. 11, 12

stored program computer A system that executes programs in memory, in contrast to computers designed to execute fixed programs. 9

temporal locality The concept that if a value from memory is accessed it has a high probability of being accessed again. 11

unsigned A data type that allows for only positive numbers or operation that assumes the operands are positive numbers. 7

von Neumann architecture An electronic digital computer possessing an arithmetic logic unit, a control unit, memory, input and output. 9, 10