KING'S College LONDON

University of London

6CCS3PRJ Background Specification

Progress Report
Breadboard Computer Architecture

Final Project Report

Author: Luca-Dorin Anton

Supervisor: Christian Urban

Student ID: 1710700

December 13, 2019

Abstract

Computers are becoming smaller, faster, more efficient and complex. At the same time, great advancements in both hardware and software provide both developers and end-users with increasingly higher levels of abstraction from the bare hardware. Whilst this allows computer users to focus on the task at hand and ignore any implementation details of the machine they are using which might get in the way, it also means that most people, including developers, software engineers and computer scientists are viewing computers as "magic black boxes". This project focuses on specifying, designing and building a simple and understandable Turing-complete machine architecture, as well as developing the necessary software tools to operate it, bridging the conceptual gap between silicon and lines of code.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Luca-Dorin Anton

December 13, 2019

Acknowledgements

I'd like to thank my supervisor, Dr. Christian Urban, for providing great mentorship, prodiving great ideas and suggestions and greatly heling towards keeping the goals of the project acheivable.

Contents

1 Introduction **2**
1.1 Inspiration and Motivation 4
1.2 Objectives 4
1.3 Project Structure 5

2 Background **6**
2.1 Computer Architecture 6
2.2 Implementing individual modules 10
2.3 8-Bit Computer Architecture Designed by Ben Eater [3] 14

3 Specification & Design **21**
3.1 Specification Guidelines 21
3.2 Major Architecture Changes 22
3.3 Operational Enhancments 22
3.4 I/O Enhancements 22
Bibliography 23

Chapter 1

Introduction

As the demand for high speed, low power, efficient and cheap computers rose over the past three decades, manufacturers invested heavily into improving production processes, shrinking transistors, pipelining instructions, creating new aggressive branch prediction models and implementing more and more functionality into the hardware directly. Moore's prediction on the number of electronic components doubling on the same surface area every two years held up well until recently when issues of quantum tunnelling started to arise. This hasn't stopped the continuous enhancement of microprocessor and microsystems though. Now, instead of making components smaller, manufacturers are installing more processing cores onto a single computing chip package.

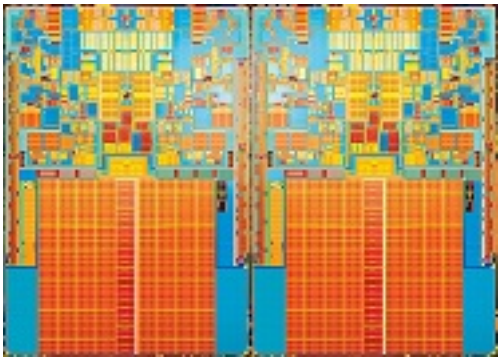

Figure 1.1: Intel quad-core 45nm CPU die

Another way of improving hardware is by implementing complex functionality, which would have been achieved traditionally through software, directly in hardware. An example of this is the implementation of the *Advanced Encryption Standard* (AES) by *Intel* directly in their lineup of CPUs through the *AES-NI* instruction set extension [6]. While the advantages for

modern society of the continuous and accelerated development of hardware cannot be doubted, there are also some worrying disadvantages. Such an advanced level of complexity in microprocessor design has been reached, that system and chip designers have started to increasingly rely on abstraction tools like *High-Level Synesthesis* to accommodate the advanced design requirements and meet user needs, as noted by Coussy in the *User Needs* chapter [4]. On the one hand, this increasing complexity of hardware poses challenges for operating system and compiler developers, who need to constantly stay up to date with the newest improvements in the hardware space and integrate them into their products, to ensure user satisfaction and sustained quality over time. One the other hand, people, including developers, are presented with no need to understand the underlying machines with which they are interacting. To quote Bruce Scheiner, a famous cryptographer and computer scientist:

> People don't understand computers. Computers are magical boxes that do things. People believe what computers tell them. [8]

As a possible solution to this general human trend towards treating computers as magical boxes", this project proposes a simple and understandable *practically implementable* machine architecture which has the same computational capabilities as a Turing machine. Some core features of the architecture are:

- 16-bit word length
- variable clock speed for live execution visualization
- single clock step function
- simplified input and output
- hardware addition and subtraction implementation

The rest of the report will go through the steps involved in specifying, designing, building and testing the machine and the software to go along with it in great detail.

1.1 Inspiration and Motivation

The main inspiration for this project is a YouTube series created by *Ben Eater* titled *Building an 8-bit breadboard computer!"* [3]. After going through the content, it became apparent that such a computer could serve as a great learning medium for developing a better understanding of computers, even more so with some hardware improvements and a sizable effort in writing some bespoke software for it. *This project is largely based around Eater's design.* It serves not only as the motivational source for the project, but it also provides a solid foundation for which extension, enhancement and improvement can be within the scope of a final year project.

1.2 Objectives

The main goal of this project is the physical implementation of a 16-bit computer system modelled after an architecture designed around the ideas of *explainability and ease of understanding* down to the transistor level. Breaking down this objective by specific hardware and software requirements, the following can be stated: The main hardware objectives are:

1. 16 bit word length
2. appropiatley sized memory space
3. memory read/write capability
4. capability to decode and execute instructions sequentially
5. program counter alteration (jumps)
6. I/O functionality
7. hardware-implemented ability to perform basic arithmetical operations
8. simple branching
9. variable speed clock
10. single step clock function

The main software objectives are:

1. adequate microcode for the control mechanisms
2. assembly mnemonics
3. assembler package to turn assembly files into binaries
4. compiler for the WHILE-language to breadboard computer binaries

1.3 Project Structure

The report begins with an in-depth circuit specification, design and analysis literature review. These core skills lay the theoretical foundation necessary for understanding the reasoning for choices when designing the hardware. The next section is concerned with a detailed analysis of the computer built by Ben Eater [3]. Eater's computer serves as the template from which the design of the computer described in this report will originate. As such, it makes sense to analyse it carefully and classify its capabilities.

This will be followed by a specification of what the new computer should achieve in contrast to the capabilities of the existing computer.

The next chapter will cover the updated design of the computer. Important design decisions will be scrutinised and held against the main goals of the project. Both high-level, as well as in-depth design choices, will be taken into account. Main design challenges will be discussed and appropriate solutions presented.

After the design stage is complete, the following chapter will document the build phase. A detailed chronological breakdown of build progress will be presented. Testing will be executed in parallel with the building, so testing documentation will be found in this chapter as well.

With the build phase complete, the software development phase will follow. The design and implementation of the various software tools necessary for running the breadboard computer will be documented in this chapter. With the software as well as the hardware ready, a rigorous testing phase will follow, coupled with an in-depth evaluation of the end product in comparison to the success criteria set at the start of the report.

Finally, the conclusion chapter summarises everything done so far and highlights the learning outcomes of the project and the possible continuation paths for future work.

Chapter 2

Background

Since this project is largely focused around designing and building a new hardware architecture, it is necessary to go through the existing material surrounding this topic. Hardware design can be structured in many different ways. For the purposes of this report, a structuring based on increasing abstraction levels will be used. Since the implementation objectives of the project aim to be educational in nature, it is crucial to start off with as few assumptions about the existing systems as possible. As such, the following explanations assume zero previous knowledge. Besides this, the educational outcomes largely focus on developers and computer scientists, professionals who would benefit from a better understanding of the computer but who are not necessarily familiar with the field of logic design. As such, the definitions and explanations will be kept as brief as possible, to avoid possibly superfluous levels of detail.

2.1 Computer Architecture

Computer architecture refers to organization, functionality and implementation design details of a computer system. It can be generally split into two main categories: *Instruction Set Architecture* and *Microarchitecture*.

2.1.1 Instruction Set Architecture (ISA)

The *Instruction Set* is the set of unique operations the computer is capable of performing. It is generally independent of the physical implementation of the system and it serves as an interface against which assembler, compiler and operating system designers and engineers can structure their products. ISA design will be a crucial part of this project. By building a hardware

architecture from scratch, the opportunity for many different ISA design choices will present itself. Many of those choices will be presented, implemented and analysed in this report.

2.1.2 Microarchitecture

A computing system's microarchitecture refers to concrete and detailed implementation choices for the hardware which is to implement the Instruction Set defined through the ISA. For the purposes of this project, the microarchitecture design will come first, which will be done against a general set of requirements and then the ISA design will follow based on the hardware design choices made. The ISA design will serve as a stepping stone towards the software architecture stage of the project.

2.1.3 General High-Level Architecture

Generally speaking, all computers share some high-level design features:

- *A Processor,* or *Arithmetic-Logic Unit (ALU),* which performs operations on some data
- *A Memory* or *Storage Unit,* which stores both data and instructions
- *A Control Unit,* which decodes instructions and issues control signals
- *Input/Output (I/O) devices,* to communicate with the outside world
- *A Clock Pulse Generator,* which keeps all other modules running synchronously
- *A Data Bus,* to facilitate the transfer of information between modules

Each individual component will be discussed in great detail in this report.

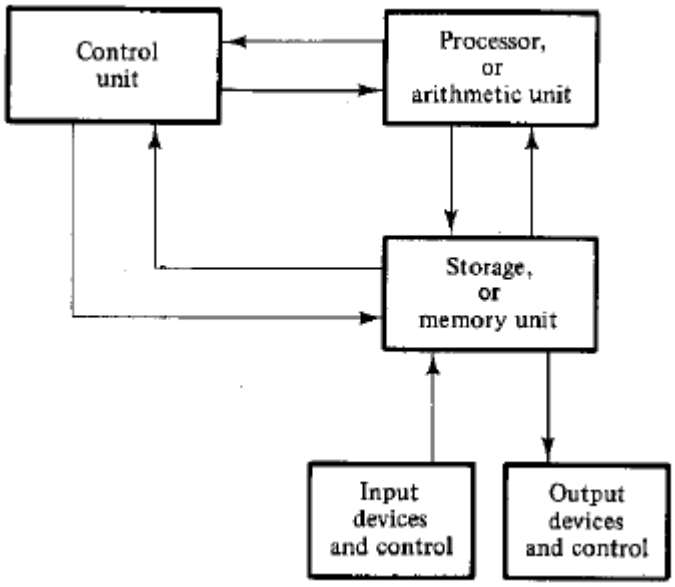

Figure 2.1: Block diagram of a digital computer, adapted from *Digital Logic and Computer Design* by M. Morris Mano [7]

Figure 2.1 is a block diagram based on the previous listing of modules. The Data Bus is represented as the double-ended arrows connecting the different components together. The clock is omitted.

2.1.4 The Processor

The processor module is tasked with executing certain operations on data. Its mode of operation is only dependent on two inputs: the data to be operated on and the operation to be applied to that data. As such, the processor does not have to hold any kind of internal state; its output will always be the same for a certain input. This makes the processor a *cobinatorial circuit*, meaning that it just implements some (albeit complex) logic function and does not have any internal state or memory.

2.1.5 Memory

Memory serves the purpose of storing data and instructions and returning the stored information when requested. By nature, it is a *sequential* circuit, meaning that it has some internal state besides the logical function implementations used to communicate with the rest of the computer. Memory is organized in addresses, each address storing a word of information.

2.1.6 The Control Unit

The Control Unit oversees all other modules and ensures that everything is happening according to the present instruction. It also has the task of decoding the present instruction to correctly select the control signals which have to be issued next. There are two main design choices to be made when constructing a Control Unit. One option is to *hardwire* the logic. Whilst more efficient, this often proves to be tedious and very hard to alter. Another common and more accessible approach is the use of *microcode*. Microcode control units use some sort of *Read-Only Memory (ROM)* as a *lookup table* to decide which control signals to switch on at a given time step of a given instruction. This lookup table is microcode. As it is implemented through a ROM, it can be easily reprogrammed or swapped out for a different ROM, making the maintenance of the Control Unit much more accessible.

2.1.7 Input/Output Devices

Input devices are used to inject instructions and data into the computer. Output devices communicate calculated results back to the user. I/O devices can take many forms and usually also require the computer to implement some sort of *interrupt* system to notify the control logic that an external event is taking place. In the case of the computer built for this project, I/O devices will be abstracted as simple registers. The computer will read from and write to those registers.

2.1.8 The Clock Pulse Generator

Computers rely on a master clock to synchronise the activity of all other components. In modern microcomputers, this is usually accomplished through a *crystal osscilator* which vibrates at a predetermined frequency. There are also other ways to achieve a steadily pulsating clock signal. In the case of the computer built in this report, a pair of voltage comparators will be used.

2.1.9 The Data Bus

Given a large number of modules present in a computer, it comes off as impractical to have each module communicate with each other module directly. In this situation, the data bus presents itself as an adequate solution. All modules connect both their input and their output terminals to the bus through some guard or buffer which allows them to disconnect from the bus as needed. Then, at any given clock pulse, only one device is allowed to connect and output

to the bus, whilst any devices interested in receiving that information can connect and input from the bus. As long as only one device writes to the bus per clock cycle, the bus functions properly.

2.1.10 Other Important Components

Besides the main modules listed above, there are a few more components which are crucial to the optimal operation of a computer.

Registers

Registers are small memory units which can store only one word of memory. A computer system normally has a very limited amount of registers. In modern computers, registers have much shorter access times than memory. Besides storing data and programs, registers can have special functions, for example, input and output registers, registers tied to certain operations, flags registers and instruction registers.

Power Supply

Since the focus of this project is electrical computers, some sort of electrical power supply will be necessary. A simple solution for a power supply based on a mobile phone charger will be presented in a later section of the report.

2.2 Implementing individual modules

With the general architecture of a computer system in place, the next design step is to create and implement design for each individual module. This can be achieved by using circuit design theory and best practices.

2.2.1 Types of electrical circuits

Electrical circuits can be broadly classified into two main categories: *combinatorial* and *sequential* circuits. *Combinatorial circuits* are circuits without any internal state (no memory) which implement a certain logic function. A logic function maps binary inputs to binary outputs. They are implemented as cascading layers of *logic gates.* The main techniques which can be used to transform a logic function into a combinatorial circuit are standard form reductions,

map creations and adequate use of don't care conditions (input/output conditions which are considered to be invalid/ will never happen). *Sequential circuits* are composed of two parts:

1. A memory structure usually built out of *flip flops*
2. One ore more combinatorial circuits, implementing some logical functions

2.2.2 Logic Gates

Logic gates are electronic devices usually built out of transistors which perform certain logic functions. A *logic function* is a function which maps binary inputs to binary outputs.

2.2.3 The Transistor

A transistor is an electronic device which allows the control of the flow of one current source through a second, potentially smaller current source. This serves as the basic component for creating more advanced electronic components like logic gates. Today, the most popular type

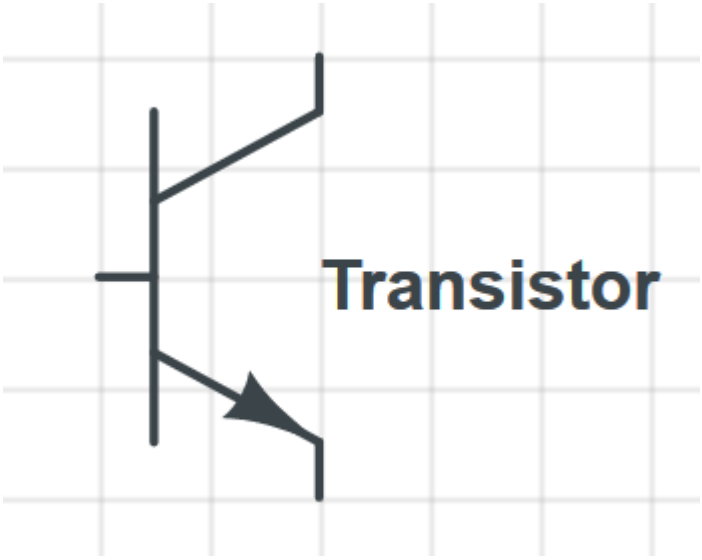

Figure 2.2: Transistor Symbol

of transistor and the most widley produced device in the world is the MOSFET [5].

2.2.4 Buffers

Buffers are simple logic gates which just pass through the signal they receive.

A	B
0	0
1	1

Table 2.1: Truth table of a buffer

TO DO: ADD CIRCUIT DIAGRAM

2.2.5 Inverters

Inverters are similar in complexity to buffers. They invert the signal they receive.

A	B
0	1
1	0

Table 2.2: Truth table of an inverter

TO DO: ADD CIRCUIT DIAGRAM

2.2.6 AND Gates

And gates output a logic 1 only if all of their inputs are logic 1's.

A	B	O
0	0	0
1	0	0
0	1	0
1	1	1

Table 2.3: AND Gate Truth Table

TO DO: ADD CIRCUIT DIAGRAM

2.2.7 OR Gates

OR gates output a logic 1 if either of their inputs or both of them are logic 1's.

A	B	O
0	0	0
1	0	1
0	1	1
1	1	1

Table 2.4: OR Gate Truth Table

TO DO: ADD CIRCUIT DIAGRAM

2.2.8 XOR Gates

Exclusive OR, or XOR gates output a logic 1 only if either of their inputs is a 1, but not both.

A	B	O
0	0	0
1	0	1
0	1	1
1	1	0

Table 2.5: XOR Gate Truth Table

TO DO: ADD CIRCUIT DIAGRAM

2.2.9 NAND Gates

A NAND gate is an inverted AND gate. This means it outputs a logic 1 in all situations except for when all of its inputs are logic 1's.

A	B	O
0	0	1
1	0	1
0	1	1
1	1	0

Table 2.6: NAND Gate Truth Table

TO DO: ADD CIRCUIT DIAGRAM

2.2.10 NOR Gates

A NOR Gate is an inverted OR gate. This means it outputs a logic 1 only if all of its inputs are logic 0's. NAND and NOR gates are considered *universal gates*. This means that any logic circuit can be built exclusivley out of NAND or out of NOR gates.

A	**B**	**O**
0	0	1
1	0	0
0	1	0
1	1	0

Table 2.7: NOR Gate Truth Table

TO DO: ADD CIRCUIT DIAGRAM

2.2.11 Flip Flops

Flip Flops are circuits usually built out of logic gates which can store one bit of information, i.e. they can be either on or off. There are many types of flip flops: RS-flip-flops (Reset-Set), D-flip-flops (Data), JK-flip flops (refined RS flip flops) and T flops (single input JK flip-flops). Each flip flop performs best in a certain scenario. Since they all implement de storage and retrieval of one bit of information, only the D-flip-flop (the most commonly used one) will be discussed.

TODO: ADD D FLIP FLOP DESC.

2.3 8-Bit Computer Architecture Designed by Ben Eater [3]

This section concerns itself with the detailed analysis of the 8-bit computer architecture designed by Ben Eater in his YouTube tutorial series [3]. This computer serves as the starting design for the computer discussed in this report in the later section, as such, it presents itself as a good topic for discussion.

2.3.1 Main Features

The main features of Bean Eater's 8-bit computer on a breadboard can be broken down as follows:

1. 16 by 8 memory space (4-bit addresses, 8-bit words)
2. adjustable and manually single-steppable clock
3. two data registers: A and B
4. ALU which implements addition, subtraction and simple branching based on zero and carry conditions
5. Decimal output through three 7-segment displays
6. Microcode-based control logic using *EEPROMs* (electronically erasable and programmable read-only memory)
7. Common 8-bit data and instruction bus

2.3.2 High-Level Overview

Figure 2.3 is a high-level block diagram displaying the inner workings of the 8-bit computer built by Ben Eater[3]. The following observations can be made when observing this diagram:

1. Each module is connected directly to the common data bus. This means that every module can output information to the bus each clock cycle. After a judicious inspection of the microcode[1] it is clear that no two modules output to the bus at the same time.
2. The clock signal and the inverse clock signal are distributed throughout the computer to each module. Also, notice how there is a control signal for the clock as well. This *HLT* (Halt) signal allows the computer to halt the clock, and implicitly halt the execution, for example after finishing a calculation, to allow it to be displayed.
3. Besides the main connections through the bus, there are some additional *special connections* between certain modules. For example, the *A register* and the *B register* have a direct connection to the *ALU*, the *MAR* (Memory address register) has a direct connection to the *RAM* (Random Access Memory) and the *IR* (Instruction Register) has a direct connection to the control logic.

4. All control signals originate from the *Control Logic* and spread out throughout the computer. Each module has at least one control signal

5. This computer is severely limited in terms of memory. While 16 bytes can be sufficient for some demonstrational trivial programs (like Factorial or Fibonacci), it is insufficient for anything else.

6. Another major limitation is the fact that it can only operate on signed integers. The architecture represents data in *big endian, two's complement integer* format. No other data format or type is supported.

2.3.3 Module Design Conventions

Ben Eater follows some essential design conventions when designing and building each of the modules for his computer. The following subsections describe those conventions.

Simplicity of understanding over cost and efficiency

In many situations where a simpler solution from cost or efficiency makes itself notices, Eater often chooses to go for a more pragmatical approach which focuses on the simplicity of understanding. Since his computer mostly serves as an educational tool, it makes sense to pursue solutions which are easy to understand, over solutions which might be slightly cheaper or more efficient. A good example of this is in the design of the clock module **??**.

For the combinatorial circuit responsible for selecting a clock signal (either the automatic or the manual one) and also filtering out the clock when the *HLT* (Halt) signal is active, Eater could have opted for a circuit built out of *NAND* (Not And) gates instead of a circuit of AND, OR and inverter gates. This is because NAND gates are universal gates, which means that any combinatorial circuit can be built exclusively out of NAND gates. In this case, this would have had a net effect on cost, since the circuit could have been implemented with only two NAND ICs (integrated circuits), instead of three. The choice was made to use AND, OR and Inverter gates since the function of those gates is more intuitive and as such, the entire circuit is easier to understand.

Connection to the Bus

Eater's computer features an 8-bit common bus for both data and instructions. Most modules are tied to this bus directly. For the bus to function properly, only one device should be allowed

to output to the bus at a time. Without some guards, connecting to the bus directly would mean that all modules would inadvertently drive the bus either high or low, depending on their output. The solution to this is the use of *Tri-State buffer gates.* This gates can be set to be in three states, either on or off, depending on the signal passing through them, or in a *high-impedance* state in which the two terminals of the buffer are essentially disconnected from each other. This is activated through a separate control signal. All modules which output to the bus do so through an IC (integrated circuit) containing such gates. An example of this can be seen on the A register 2.5.

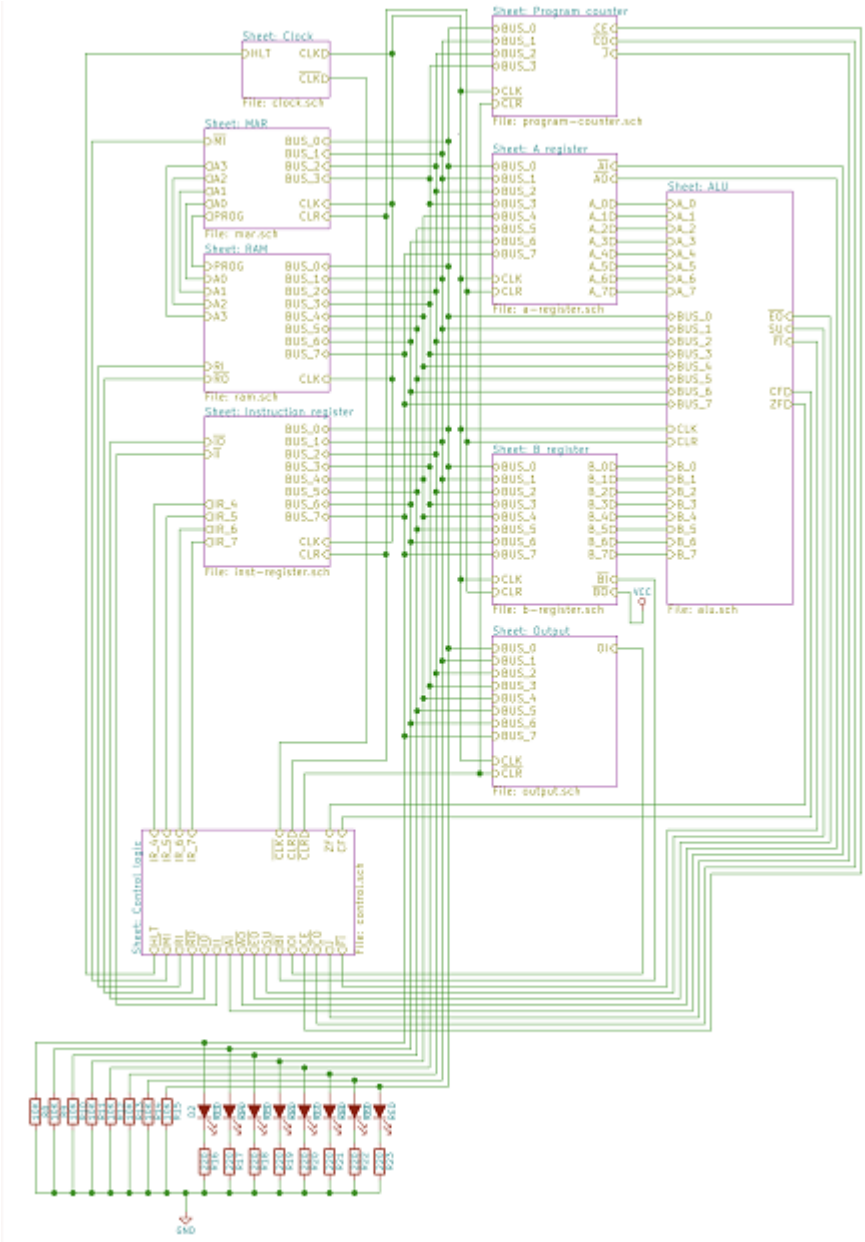

Figure 2.3: Block diagram of the 8-bit computer built by Ben Eater [2]

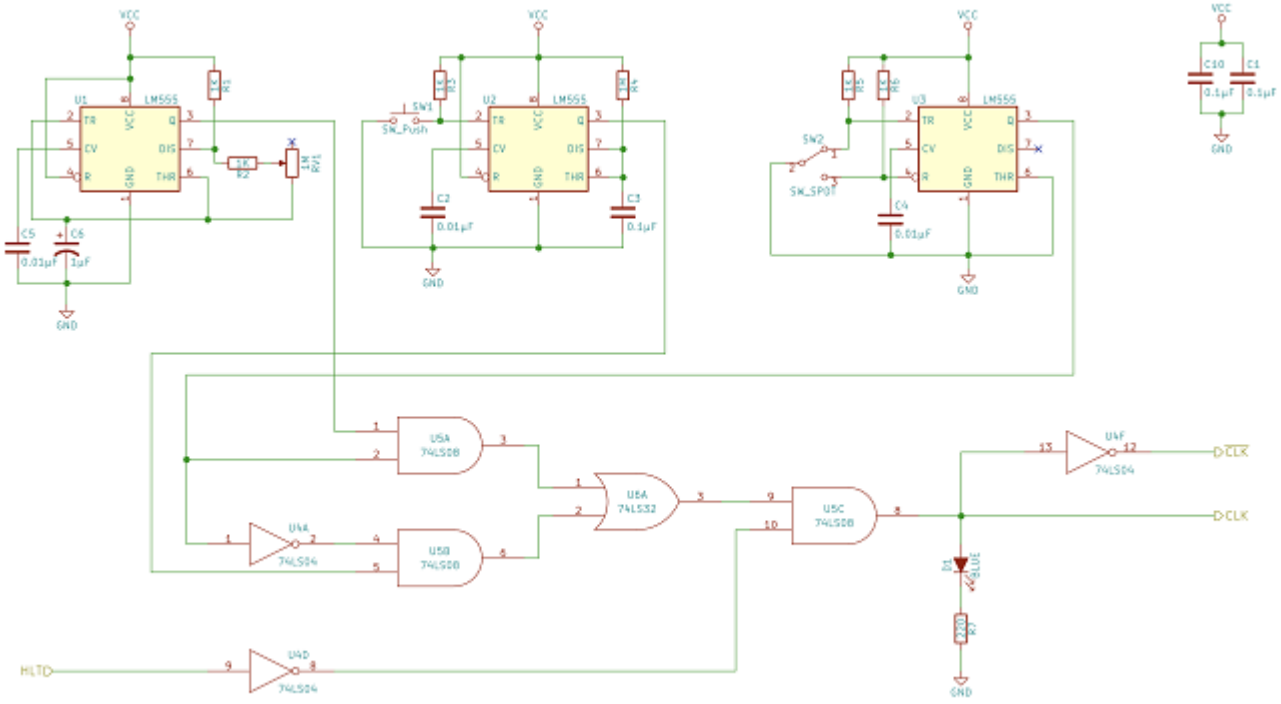

Figure 2.4: Schematic of the clock module in Bean Eater's 8-bit computer [2]

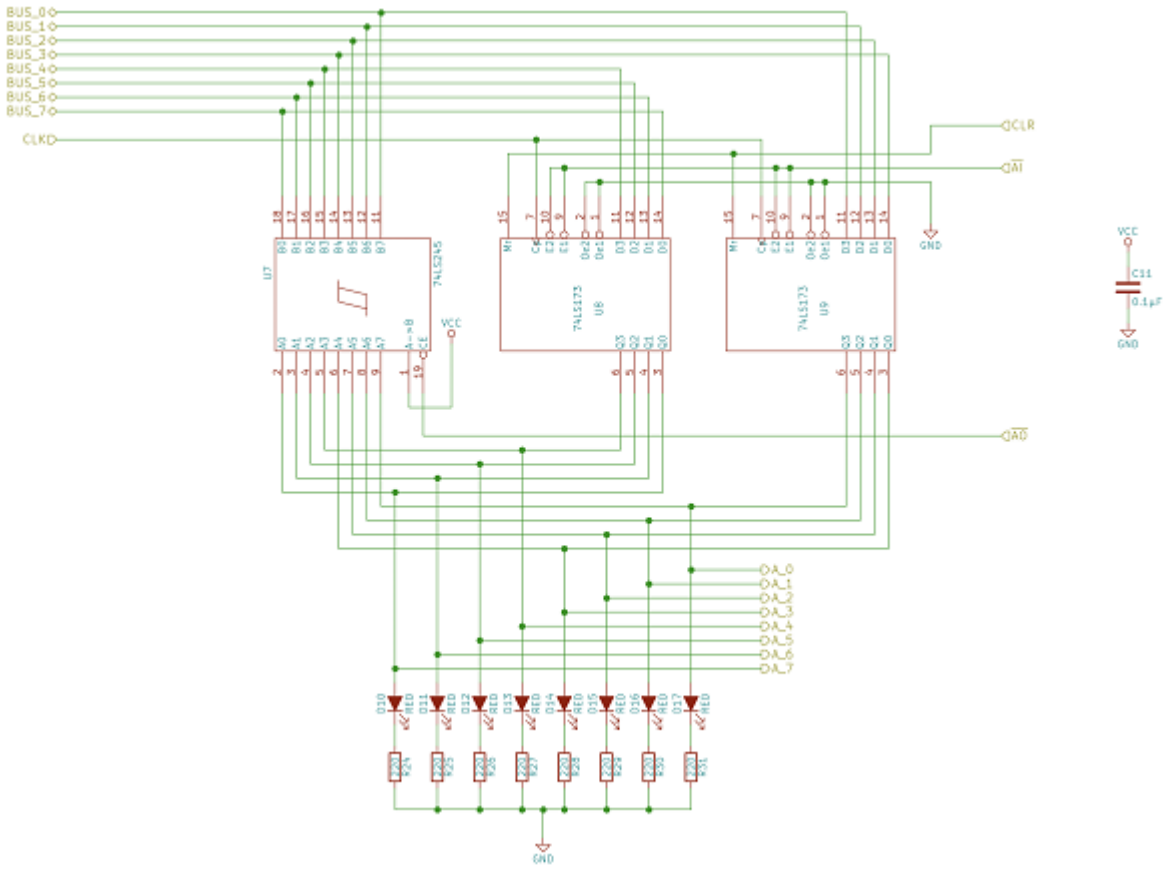

Figure 2.5: Schematic of the A register in Bean Eater's 8-bit computer [2]

Chapter 3

Specification & Design

This chapter of the report focuses on creating a comprehensive set of specifications for the computer to be built as a part of the project and then provides a detailed listing of the design choices made to satisfy the specifications provided. Since the design will be based on Bean Eater's 8-bit breadboard computer [3], most specification criterions will be phrased as additions and enhancements to the existing designs.

3.1 Specification Guidelines

The specifications which are about to be presented serve the purpose of adding functionality to the 8-bit breadboard computer such that its educational potential is harnessed more effectively, while at the same time avoiding over-complication and over-extensions of scope. As such, it makes sense to list some of the features which will fall out of the scope of this build for practical and time considerations.

- Interrupts and Interrupt handling
- Processes (The computer will only run one process, there will be no threading interface/ no operating system)
- Floating-Point operations
- Support for any kind of advanced in-hardware operations (for example encryption)
- Native support for signed integers over 16 bits
- Graphical User Interfaces (GUIs)

- Input through traditional peripheral (mouse and keyboard)

3.2 Major Architecture Changes

The computer should broadly follow the architecture o the 8-bit computer designed and built by Bean Eater [3]. The major architectural difference should be *the extension to 16-bit words*. Since the word length of the computer should be 16 bits, its bus and most of its modules should be extended to accommodate this extra capacity.

3.3 Operational Enhancments

Besides Addition and Subtraction, the computer should also implement *bit-shifting* (both left and right, with the option to choose to insert a 0 or a 1 after the shift). Bit-shifting is a crucial operation which is very often performed to increase the efficiency of certain operations (for example, multiplication and division by 2 can be expressed in binary as a left or a right shift of 1 bit)

3.4 I/O Enhancements

Currently, the only way to provide input to the 8-bit computer is by *manually* programming each memory address through dip-switches. This is slow, clunky and prone to errors. There should exist a mechanism to quickly program the computer, for example through an external *microcontroller* like an Arduino. Besides this, there should also exist a way for the computer to request input *while executing* from an external device like a microcontroller. Similarly, to provide persistence to the values from calculated by the computer, instead of being able to display only one value at a time, the computer should also have the ability to communicate with an existing external device like a separate microcontroller, providing it with the values it has calculated. In turn, the microcontroller a the be connected to a regular personal computer and then programmed to display those values to the screen.

References

[1] Eater Ben. Conditional jump instructions. https://www.youtube.com/watch?v=Zg1NdPKoosU, 2018. Accessed: 2019-12-13.

[2] Eater Ben. 8-bit computer high level architecture diagram. https://eater.net/8bit/schematics, 2019. Accessed: 2019-12-13.

[3] Eater Ben. Building an 8-bit breadboard computer! https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU, 2019. Accessed: 2019-07-15.

[4] Philippe Coussy and Adam Morawiec. *High-level synthesis: from algorithm to digital circuit*. Springer Science & Business Media, 2008.

[5] Laws David. Mosfet. https://computerhistory.org/blog/13-sextillion-counting-the-long-winding-road-to-the-most-frequently-manufactured-human-artifac?key=13-sextillion-counting-the-long-winding-road-to-the-most-frequently-manufactured-human-ar 2018. Accessed: 2019-12-8.

[6] Rott Jeffrey. (AES-NI) intel advanced encryption standard instructions. https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni, 2012. Accessed: 2019-12-8.

[7] M Morris Mano. *Digital logic and computer design*. Pearson Education India, 2017.

[8] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2011.