A Comparative Analysis of the Performance of Emulated Ternary vs Binary Logic Systems

Thomas Timmons
Bachelor Hons. Computer Science
tlt33@bath.ac.uk

March 19, 2025

Abstract

Here I will provide an informative and concise summary of the report and it's findings.

Contents

Acknowledgements

Thank you to my Supervisor, Dr. James Davenport, for his guidance and support throughout this project along with Dr. Paola Bruscoli whose feedback has significantly enhanced the quality of this work.

1 Introduction

Binary Logic has been the defacto standard for digital systems since the inception of the transistor. However, due to the imminent end of Moore's Law, researchers have been looking for alternative methods to increase the performance of digital systems. Multi-valued logic (MVL) has emerged as appealing subject where this paper will attempt to realise the theoretical performance gains of ternary logic within an emulator. We can then measure and project the performance on the assumption it should be possible to implement these circuits in hardware and discuss possible implications.

1.1 Ternary Algebra

Points

- Discuss the relationship between computing in different bases and complexity theory/-turing machines.
- Critically compare this fields such as quantum computing with recent advances.

The cost or complexity of number systems can be evaluated through representation efficiency, computational overhead, and implementation requirements.

Representation efficiency refers to how digits or symbol are required represent any given range of values N, with the base of the number system represented by R and the necessary number of digits required is d (rounded upwards to the nearest integer):

$$N = R^d \tag{1}$$

This directly impacts the memory usage and storage costs in a computing system.

The computational overhead involves the complexity of performing basic arithmetic operations. Larger bases require the managing of more states, requiring the processing of a larger set of possible combinations. The number of possible combinations is given by:

$$C = R^i \tag{2}$$

Where i is the number of inputs for the operation and R is the base of the number system as defined before. For example, the number of possible combinations for addition is $C = R^2$ which in binary is $C = 2^2 = 4$ creating the set $\{0+0, 0+1, 1+0, 1+1\}$. In ternary the set would be $C = 3^2 = \{0+0, 0+1, 0+2, 1+0, 1+1, 1+2, 2+0, 2+1, 2+2\} = 9$ which requires implementing more logic circuits to handle the additional combinations.

Implementation requirements consider the the practical aspects of building a system that works with a given base. This includes error detection and correction using approaches such as Hamming Code's in binary logic As this paper aims to explore the performance of ternary logic systems in comparison to binary systems within an emulator, we will not be considering the practical challenges of implementing ternary logic in hardware.

When considering the representation efficiency of a number system, looking at the formula's you would assume that increasing the base would increase the efficiency but the more digits also require more space. Therefore we can make the assumption that c is directly proportional to the capacity of the digits ($R \cdot d$). (Hurst also discusses the cost assuming that the cost is independent of the radix R which would lead us to believe that the larger the base the more efficient the system.)

Therefore for some constants k to represent the direct proportionality we can substitute $d = \frac{\log N}{\log R}$ from Equation 1 into the formula (Hurst 1984):

$$c = k(R \cdot d) = k\left(R \cdot \frac{\log N}{\log R}\right) = k \log N \left(\frac{R}{\log R}\right) \tag{3}$$

Minimising this cost c can be done by differentiating with respect to R and setting the result to 0:

$$\frac{\partial c}{\partial R} = k \log N \cdot \frac{d}{dR}\left[\frac{R}{\log R}\right] = k \log N \cdot \frac{\log R - 1}{(\log R)^2} = 0 \tag{4}$$

After removing constants $\log R = 1$ and solving for $R = e = 2.718$. This work by Hurst in 1984 at the University of Bath shows that the natural base is the most efficient radix for implementation of switching circuits (Bitra et al. 2018). Since as stated in Equation 1, the radix R must be an integer then we are left with $R = 3$ to be able to perform computation (Jaber 2020).

In binary logic set of possible values is denoted as $\mathbb{B} = \{0, 1\}$. Ternary numeral systems can either unbalanced ternary values $\mathbb{T} = \{0, 1, 2\}$ or balanced ternary values $\mathbb{T} = \{-1, 0, 1\}$ with a variety of glyphs used to represent the values where $\overline{1}$ or T is sometimes used to represent -1.

Balanced ternary has the elegant property that of representing negative numbers intrinsically without the need for a separate signed bit. This leads to more symmetric carry rules due to a more even distribution around zero as shown in Figure 1.

$+$	T	0	1
T	$T1$	T	0
0	T	0	1
1	0	1	$1T$

$-$	T	0	1
T	0	T	$T1$
0	1	0	T
1	$1T$	1	0

$\times$	T	0	1
T	1	0	T
0	0	0	0
1	T	0	1

$\div$	T	1
T	1	T
0	0	0
1	T	1

Figure 1: Balanced Ternary Single-Trit Addition, Subtraction, Multiplication and Division Tables

For subtraction and division which are not commutative, the first operand is given to the left of the table and the second at the top.

Balanced ternary has a range of advantages when it comes to computation over binary

logic such as the reduction in the carry rate for addition/subtraction cuts down the carry rate in multi-digit multiplication as demonstrated in Figure 1 with the values being 2/9 and 1/4 respectively. Also notice that it is possible to perform subtraction by negating the second operand and adding the two values together which will useful for the implementation of the ternary logic gates.

Therefore, for this paper and emulator we will be using balanced ternary logic and any mention of 'ternary logic' is referring to 'balanced ternary logic'.

To calculate the value of a ternary number T with a radix point in decimal we can use the following formula:

$$v = (a_n a_{n-1} \cdots a_0 . c_1 c_2 c_3 \cdots)_3 = \sum_{i=0}^{n} a_i 3^i + \sum_{i=1}^{\infty} c_i 3^{-i} \tag{5}$$

Where n and m are the number of digits to the left and right of the radix point respectively and v is the value in decimal given a vector of values containing a radix point. Notice that each position can have a negative contribution to the total due to $a, c \in \{-1, 0, 1\}$. The symmetry creates the elegant property where negation can be performed by negating each trit within the word which has the benefits of only needing an addition circuit as mentioned above.

To directly compare between the binary and ternary systems we need to consider how we can encode binary values into ternary values and vice versa. Using continued fractions, the best approximations to rationals are given by convergents to the continued fraction, alternating between overshooting and undershooting the target value (Davenport 2008).

Since we are looking for $2^a \approx 3^b$ where $a, b \in \mathbb{N}$. We can take the logarithm of both sides to get the following:

$$\frac{a}{b} \approx \frac{\log 3}{\log 2} \tag{6}$$

Where $2^a < 3^b$ to ensure that the binary value can be encoded into a ternary value.

This continued fraction expansion can be calculated by computing the convergents which are as follows:

$$\frac{\log 3}{\log 2} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \cdots}}} \tag{7}$$

$$a_0; a_1, a_2, a_3, a_4 \cdots = 1; 1, 1, 2, 2, \cdots \tag{8}$$

This gives us:

$$\frac{2}{1} = 1 + \frac{1}{1} \Rightarrow 2^2 \approx 3^1 \text{ (but ternary worse than binary: } 2^2 > 3^1)$$
$$\frac{3}{2} = 1 + \frac{1}{1 + \frac{1}{1}} \Rightarrow 2^3 \approx 3^2$$
$$\frac{8}{5} = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}} \Rightarrow 2^8 \approx 3^5 \text{ (but ternary worse than binary: } 2^8 > 3^5)$$
$$\frac{19}{12} = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}} \Rightarrow 2^{19} \approx 3^{12}$$

$\frac{19}{12}$ provides a good approximation with an efficiency of 98.65%. However using a 19 bit word is not practical due to it being too large.

$\frac{11}{7}$ also provides a good approximation with an efficiency of 93.64%. This is created by summing the 2nd and 3rd convergents $\frac{3+8}{2+5}$ which provides a good balance between efficiency and word size.

This is the same ratio that Micron uses to encode 11 bits of binary data into 7 trits in their GDDR7 memory bus using PAM3 (Micron Technology 2024).
For our emulator we will use the same ratio in an attempt to simulate the performance potential of running this bus directly into a ternary ALU.

1.2 Ternary Logic Gates

Gates can be represented in matrix form where the following Binary AND truth table has the has the equivalent matrix representation:

$A \times B$	0	1
0	0	0
1	0	1

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Figure 2: Binary AND Gate Truth Table and Matrix Representation

1.2.1 Single-Input Gates

Single-input gates can be represented as a vector of length 2 for binary logic and length 3 for ternary logic. Combination theory shows that the number of possible combinations for a single input gate is n^n where n is the number of inputs. Therefore, the number of possible combinations for a single input gate is $2^2 = 4$ and $3^3 = 9$ for binary and ternary logic gates. Most of these are redundant so only the useful gates have been listed in Figure **??**.

Matrix	Description	Name	Symbol
$\begin{bmatrix}1\\0\end{bmatrix}$	Identity, Buffer, Pass	Buffer	A
$\begin{bmatrix}1\\0\end{bmatrix}$	Inverter	NOT	$\overline{A}$

Figure 3: Useful Single Value Binary Gates

Matrix	Description	Name	Symbol
$\begin{bmatrix}T\\0\\1\end{bmatrix}$	Identity, Buffer, Pass	BUF	A
$\begin{bmatrix}1\\0\\T\end{bmatrix}$	Inverter	NOT	$\overline{A}$
$\begin{bmatrix}1\\1\\T\end{bmatrix}$	Positively Biased Inverter	PNOT	$\hat{A}$
$\begin{bmatrix}1\\T\\T\end{bmatrix}$	Negatively Biased Inverter	NNOT	$\check{A}$

Figure 4: Useful Single Value Binary and Ternary Gates

1.2.2 Two-Input Gates

Matrix	Description	Name	Symbol
$\begin{bmatrix}0&0\\0&1\end{bmatrix}$	And	AND	$A \times B$
$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	Or	OR	$A + B$
$\begin{bmatrix}0&1\\1&0\end{bmatrix}$	Exclusive Or	XOR	$A \oplus B$

Figure 5: Useful Two-Input Binary Gates

All of the gates in Figure 5 and 6 have inverted equivalents that can be created by negating the output will be denoted with an N proceeding the gate name. e.g. NAND, NOR, NANY.

These gates can be combined to build more complex circuitry such as adders, multipliers and ALU's in their respective bases with a possible implementation shown by (Keshavarzian & Sarikhani 2014) while (Kam et al. 2022) details an possible ternary RISC-based instruction set. This paper will build upon their designs to implement an instruction set that is comparable between base systems.

Matrix	Description	Name	Symbol
$\begin{bmatrix} T & T & T \\ T & 0 & 0 \\ T & 0 & 1 \end{bmatrix}$	And/Minimum	AND	$A \times B$
$\begin{bmatrix} T & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Or/Maximum	OR	$A + B$
$\begin{bmatrix} T & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Consensus	CONS	$A \boxtimes B$
$\begin{bmatrix} T & T & 0 \\ T & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$	Any	ANY	$A \boxplus B$
$\begin{bmatrix} 1 & 0 & T \\ 0 & 0 & 0 \\ T & 0 & 1 \end{bmatrix}$	Multiplication	MUL	$A \otimes B$
$\begin{bmatrix} 1 & T & 0 \\ T & 0 & 1 \\ 0 & 1 & T \end{bmatrix}$	Addition	SUM	$A \oplus B$

Figure 6: Useful Two-Input Ternary Gates

(Clarke 1993) discusses the implementation of floating point arithmetic in MVL systems. As this is a stretch goal of the project this resource may come in useful when implementing the floating point arithmetic.

1.2.3 Assumptions

Researchers have attempted to investigate this field using two distinct approaches. One is from a more theoretical perspective where they investigate the potential gains of ternary logic. The other is from a more practical approach where Ternary test benches have been created in order to attempt to validate the theoretical performance gains.

For this project we will assume that comparable binary and ternary logic gates have the same propagation delay. This will allow us to to directly compare the performance of the two systems. Implications of this assumption will be discussed in the conclusion.

$$\begin{aligned} AND &\equiv \mathbb{T}AND \\ OR &\equiv \mathbb{T}OR \\ NOT &\equiv \mathbb{T}NOT \end{aligned} \tag{9}$$

This will allow us to create a basic binary and ternary emulator whose performance will be directly comparable.

2 Literature and Technology Survey

Points

- Research into the polish school of logic and see what's going on there?
- Relationship to other areas of research such as symbolic ternary system models?
- Discuss relevance of these other areas of research to the project and possible implications.

2.1 Previous Ternary Implementations

Setun was a sequential computer developed in 1958 by Sergei Sobolev and Nikolay Brusentov at Moscow State University. It was the most modern ternary computer using the balanced ternary numeral system consisting of 81 words of memory where each word was composed of 18 trits. It has a one-address architecture with one index register and was able to handle both positive and negative floating point calculations (Weatherby 2018) (Brousentsov et al. 2002).

The instruction set consisted of 24 instructions including performing mantissa normalisation for floating point calculations, shift, combined multiplication and addition.
In 1970, the Setun-70 was developed that built on idea of its predecessor. It had a short instruction set that was developed and implemented independently from the RISC architecture principles but did implement Dijkstra's ideas of of structured programming. Both of these computers where only used for research purposes and were never used commercially. The interval range for the mantissa value of a normalised number was changed to $\{0.5 < |m| < 1.5\}$ whereas it had been $\{0.1666... < |m| < 0.5\}$ in its predecessor (Brousentsov et al. 2002).

During a thawing period of the Cold War, visits were allowed between scientists from the different nations resulting in TERNAC. This was an emulator written in FORTRAN by the State University of New York at Buffalo and was able to deliver on technical promise of ternary systems. As Setun when implementing the hardware was lacking a switch that could encode the two values in one setting and instead used two separate gates (Weatherby 2018).

Weatherby summarises this history with the following statement:

> "Ternary computing, in other words, has always been emulation rather than implementation—its history is in large part imaginary."

And while his article is not a scientific paper and is written in a stylised and opinionated manner, it does raise a valid point. Fundamentally it may be impossible to implement a true ternary system in hardware due to fundamental laws around digital logic. However, we need not concern ourselves as we are only looking at the implications of ternary logic should it be possible to implement in hardware.

Brousentov takes an opposing point of view in his paper where he argues that the Setun computer did realise the theoretical benefits despite the two separate gates (Brousentsov

et al. 2002):

> "This experience convincingly confirms practical preferences of ternary digital technique."

This also should be viewed with caution as the paper was written by Russian scientists in 2002 and while it gives a few details on the architecture of Setun it does not contain any quantitative measures. It does contain a lot uncited claims with a heavy use of adjectives of almost a persuasive nature.

A more recent publication by Zahoor et al. in 2024 provides a more favourable view on the possible future of ternary. The paper discusses the benefits and possible implementations of ternary logic using novel post-CMOS devices due to advances in material science and nanotechnology. It cites several researcher who have reported ternary arithmetic logic unit designs which provide increased computation capability along with high energy efficiency and reduced interconnect complexity (Zahoor et al. 2024).

One example of these papers from 2009 by Keshavarzian et al. details the design of a ternary full adder (TALU) using Carbon Nanotube Field Effect Transistors (CNTFET) to get a 53% increase in Power-Delay Product (PDP) at 250MHz over previous CNTFET designs but does not make a direct comparison to a binary equivalent (Keshavarzian & Sarikhani 2014).

2.2 Alterative Uses of Ternary Values

The use of multiple values has been implemented in industrial solutions due to limitations in bandwidth because of interconnection difficulties. This relates to the shoreline problem where if you consider a chip of side length n with a perimeter of $4n$ and an area of n^2. If we double the side length to $2n$ the perimeter becomes $8n$ and the area becomes $4n^2$. The perimeter has doubled but the area has quadrupled. This also applies to the number of pins on a chip where you begin to run into interconnection difficulties (Zahoor et al. 2024). Inherently increasing the base of a number system will reduce the demand for bandwidth resulting in a smaller necessary interface area due to more information being transmitted per pin (Zahoor et al. 2024).
NVIDIA 30 Series GPU's make use of ternary values to encode 3 different pulse amplitudes on a single wire within the memory bus of their cards and will be used in their next generation of GPUs and is currently in development by Micron (Micron Technology 2024). PAM3 encoding is used to increase the single pin throughput 32Gb/s compared to its predecessor GDDR6 which had a throughput of 18Gb/s. This is achieved by encoding the sequential binary values into ternary values represented by 3 different voltage levels. This allows for 3 bits of information to be transmitted on a single wire within two clock cycles (1.5 bits per cycle) (Micron Technology 2024) (Nam et al. 2024).

Non-classical computing provide potential ways to implement ternary logic. One of these is optical computers (Photonic Computers), which use light to perform computation using photons. Infrared light with a wavelength of $1.50\mu m$ is used to transmit data long distances

within optical fibres but light with wavelengths around $570nm$ can can be used to perform computation. Logic gates are built using interference patterns to define possible states where polarization provides a way to encode ternary values.

This technology is mainly being used to data transfer but interest is growing due to the exponential demand for parallel processing with the rise of AI and machine learning.

2.3 Emulation of Circuits

There exist many tools that can be used to emulate circuits but the following have been evaluated for this project:

- **Circuit Diagram** is a free online tool that allows for the design and simulation of electronic circuits.
- **Logisim** is a free and open-source software that is available on Windows, Mac and Linux and allows for the design and simulation of digital logic circuits.
- **QUCS** (Quite Universal Circuit Simulator) is a free software that allows for the simulation of electrical circuits.
- **Verilog** is a Hardware Description Language (HDL) that can be used to describe the behavior of digital circuits.
- **Custom Rust Emulator** will be built to directly compare the performance of binary and ternary logic systems.

Other propriety tools can be used to emulate circuits but these require licenses and are not free to use, therefore will not be considered for this project.

Both QUCS and Circuit Diagram are more focused on the simulation of electrical circuits while Logisim is more focused on the implementation of logic gates from a educational perspective. Circuit Diagram is a good tool for creating simple circuits but is not suited to larger projects due to its online nature and lack of features. Within Logisim it is possible to simulate ternary behavior by using multiple binary gates together with multiple bits needed to represent the possible states making the designs more complex than needed. QUCS is more flexible and could also achieve the desired goal using analog components it is again not suited to purpose. Voltage levels for each ternary value along with switching thresholds would create more boilerplate than necessary.

HDL (Hardware Description Language) simulators are software tools that allow for the design and verification digital circuit designs. They utalise languages such as VHDL and Verilog to describe the behavior of the circuit. Verilog allows you to define the behavior of a module at 3 different levels of abstraction; Gate Level, Dataflow and Behavioural with the latter being the most abstract.

When compiling higher level of abstractions are converted into logic gates and flip-flops. This allows developers to design complex circuits without needing to understand the underlying hardware. (Bitra et al. 2018) used Verilog to implement a ternary ALU showing that this tool provides the flexibility needed to implement the required ternary logic gates although the paper does not appear to reach a clear conclusion.

Being able to work at the gate level is a necessary requirement for this project. The Verilog compiler does not support higher level abstractions of ternary logic but implementing this in our own custom system would require a lot of overhead building the boilerplate from the ground up. Furthermore Verilog is specifically designed to simulate and test digital circuits including a plethora of performance metrics to directly compare the produced emulators. Therefore, we will be using Verilog to implement the ternary logic gates and the binary equivalents.

3 Experimental Hypotheses

Points

- Theoretical performance gains of ternary logic have been shown in the literature.
- Can we achieve these performance gains in practice?
- Lots of research has been done into producing MVL logic gates, can we show if these endeavors are worth it?
- Expect a performance gain of around 40% for napkin maths but how will the extra circuitry affect the complexity/number of gates required?
- Analysing complexity which algorithms will benefit from these performance gains?

4 Experimental Design

4.1 Specification

- Discuss goals and objectives of the project and why they have been included and why certain goals have been excluded.

4.2 Design

- Design two CPU architectures, one using binary logic and the other using ternary logic.
- Briefly discuss the lifecycle of each operation in the CPU as well as the choice of endian.
- Discuss in detail the design choices for each split of opcode, operand and their respective efficiencies. (Be Critical)
- Discuss extending the work on Ternary full adders as well as the instruction set architecture from RISC based ternary architecture
- Investigate epsilon error when computing floating point numbers in ternary logic compared to binary logic.
- Discuss clamping the ranges of min of both the binary and ternary systems to allow for direct comparison. (Be Critical)
- Demonstrate how you will compile assembly language into machine code for both systems and discuss design choices.

4.3 Implementation

- Implement high level behavioral logic for both systems (these will be similar) along with design choices.
- Implement the low level logic gates for computationally expensive ALU operations for both systems e.g. Addition.
- Discuss which circuits were implemented and which were not in the interest of time. Justify these decisions.

4.4 Testing

- Discuss the testing strategy for the project and how you will ensure that the results are valid.
- Discuss the metrics you will be using to quantify the performance of the two systems. e.g. number of gates, propagation delay.
- Discuss the expected results and how you will analyse the data to draw conclusions.

Num	Type	9-trit instructions	Operation
0	R	MV Ta,Tb	TRF[Ta] = TRF[Tb]
1	R	PTI Ta,Tb	TRF[Ta] = PTI(TRF[Tb])
2	R	NTI Ta,Tb	TRF[Ta] = NTI(TRF[Tb])
3	R	STI Ta,Tb	TRF[Ta] = STI(TRF[Tb])
4	R	AND Ta,Tb	TRF[Ta] = TRF[Ta] $\wedge$ TRF[Tb]
5	R	OR Ta,Tb	TRF[Ta] = TRF[Ta] $\vee$ TRF[Tb]
6	R	XOR Ta,Tb	TRF[Ta] = TRF[Ta] $\oplus$ TRF[Tb]
7	R	ADD Ta,Tb	TRF[Ta] = TRF[Ta] + TRF[Tb]
8	R	SUB Ta,Tb	TRF[Ta] = TRF[Ta] $-$ TRF[Tb]
9	R	SR Ta,Tb	TRF[Ta] = TRF[Ta] $\gg$ TRF[Tb][1:0]
10	R	SL Ta,Tb	TRF[Ta] = TRF[Ta] $\ll$ TRF[Tb][1:0]
11	R	COMP Ta,Tb	TRF[Ta] = compare(TRF[Ta],TRF[Tb])
12	I	ANDI Ta,imm	TRF[Ta] = TRF[Ta] $\wedge$ imm[2:0]
13	I	ADDI Ta,imm	TRF[Ta] = TRF[Ta] + imm[2:0]
14	I	SRI Ta,imm	TRF[Ta] = TRF[Ta] $\gg$ imm[1:0]
15	I	SLI Ta,imm	TRF[Ta] = TRF[Ta] $\ll$ imm[1:0]
16	I	LUI Ta,imm	TRF[Ta] = {imm[3:0],00000}
17	I	LI Ta,imm	TRF[Ta] = {TRF[Ta][8:5],imm[4:0]}
18	B	BEQ Ta,B,imm	PC = PC + imm[3:0] if TRF[Ta][0] = B
19	B	BNE Ta,B,imm	PC = PC + imm[3:0] if TRF[Ta][0] $\neq$ B
20	B	JAL Ta,imm	TRF[Ta] = PC+1, PC = PC + imm[4:0]
21	B	JALR Ta,Tb,imm	TRF[Ta] = PC+1, PC = PC + imm[4:0]
22	M	LOAD Ta,Tb,imm	TRF[Ta] = TDM[TRF[Tb]+imm[2:0]]
23	M	STORE Ta,Tb,imm	TDM[TRF[Tb]+imm[2:0]] = TRF[Ta]

Table 1: 9-trit Instruction Set Architecture

References

Bitra, Chitra, A., Guntanala, I. & Hareesh (2018), 'Implementation of ternary alu using verilog', *International Research Journal of Engineering and Technology* **5**(3).

Brousentsov, N., Maslov, S., Ramil, A. J. & Zhogolev, E. (2002), 'Development of ternary computers at moscow state university', *Russian Virtual Computer Museum. Ed. Eduard Proydakov* **11**.

Clarke, C. T. (1993), The implementation and applications of multiple-valued logic, PhD thesis, University of Warwick.

Davenport, J. (2008), 'The higher arithmetic: 8th. ed. revised with a new chapter by j.h. davenport', *The Cambridge University Press* .

Hurst, S. L. (1984), 'Multiple-valued logic—its status and its future', *IEEE Transactions on Computers* .

Num	Type	16-bit instructions	Operation
0	R	MV a,b	RF[a] = RF[b]
2	R	NOT a,b	RF[a] = NOT(RF[b])
4	R	AND a,b	RF[a] = RF[a] $\wedge$ RF[b]
5	R	OR a,b	RF[a] = RF[a] $\vee$ RF[b]
6	R	XOR a,b	RF[a] = RF[a] $\oplus$ RF[b]
7	R	ADD a,b	RF[a] = RF[a] + RF[b]
8	R	SUB a,b	RF[a] = RF[a] $-$ RF[b]
11	R	COMP a,b	RF[a] = compare(RF[a],RF[b])
16	I	LUI a,imm	RF[a] = {imm[7:0],00000000}
17	I	LI a,imm	RF[a] = {RF[a][15:8],imm[7:0]}
22	M	LOAD a,b,imm	RF[a] = TDM[RF[b]+imm[4:0]]
23	M	STORE a,b,imm	TDM[RF[b]+imm[4:0]] = RF[a]

Table 2: 16-bit Instruction Set Architecture

Breakdown	Bits	Binary Range	Trits	Ternary Range	$\mathbb{T} \cup \mathbb{B}$ Range	Efficiency
Opcode	5	$[0-31]$	3	$[0-26]$	$[0-26]$	84.38%
Register	3	$[0-7]$	2	$[0-8]$	$[0-7]$	88.89%
Immediate	8	$[0-255]$	4	$[0-80]$	$[0-80]$	68.75%
Shift	2	$[0-3]$	2	$[0-8]$	$[0-3]$	37.50%

Table 3: 16-bit Instruction Set Architecture

Jaber, R. A. (2020), 'Multiple-valued logic circuit design and data transmission intended for embedded systems', *ResearchGate* .

Kam, D., Min, J. G., Yoon, J., Kim, S., Kang, S. & Lee, Y. (2022), Design and evaluation frameworks for advanced risc-based ternary processor, *in* '2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)', IEEE, pp. 1077–1082.

Keshavarzian, P. & Sarikhani, R. (2014), 'A novel cntfet-based ternary full adder', *Circuits, Systems, and Signal Processing* **33**, 665–679.

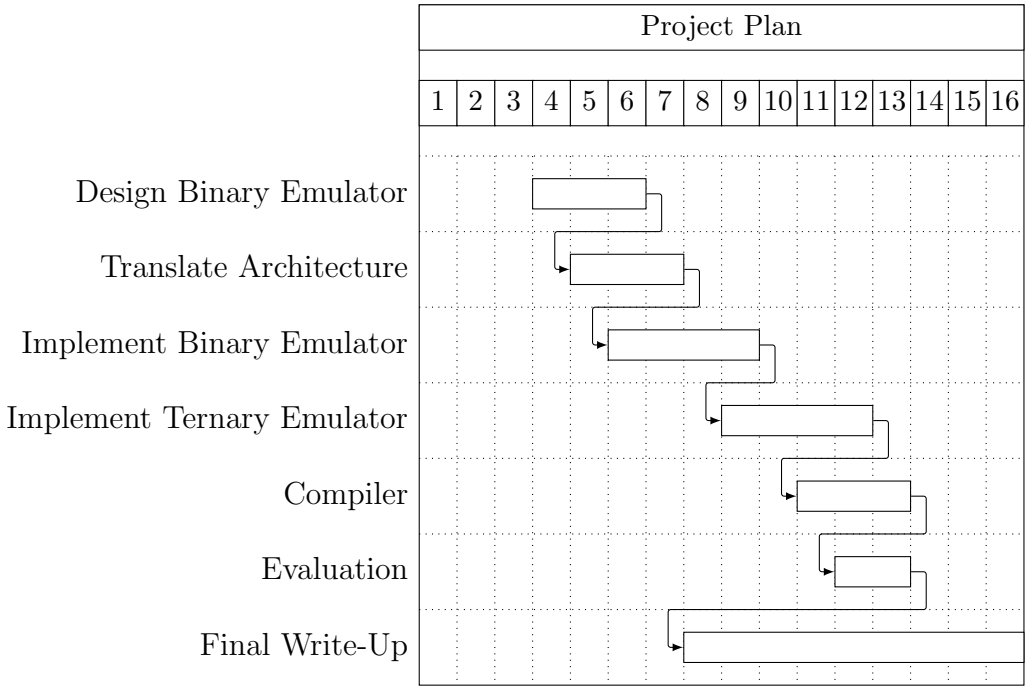
Micron Technology, I. (2024), 'Micron gddr7 memory product brief', *Micron* .

Nam, J., Han, J. & Kim, H. (2024), Low-power encoding for pam-3 dram bus, *in* '2024 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)', IEEE, pp. 1–4.

Weatherby, L. (2018), 'The imaginary history of ternary computing', *cabinetmagazine.org* .

Zahoor, F., Jaber, R. A., Isyaku, U. B., Sharma, T., Bashir, F., Abbas, H., Alzahrani, A. S., Gupta, S. & Hanif, M. (2024), 'Design implementations of ternary logic systems: A critical review', *Results in Engineering* p. 102761.

5 Appendix


Project Plan
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Design Binary Emulator
Translate Architecture
Implement Binary Emulator
Implement Ternary Emulator
Compiler
Evaluation
Final Write-Up