The 6502 ALU Dataset

Jonathan Tapson[1,2]

***Abstract*—We introduce a machine learning data set generated by the logical structure of the Arithmetic Logic Unit of the MOS Technology 6502 CPU. The purpose of this dataset is to provide a vehicle for understanding the effectiveness and limitations of methods such as supervised deep learning and backpropagation in elucidating the structure of computational platforms, including the brain. The ALU is a simple structure consisting of 208 transistors connected by binary weighted connections and carrying binary signals, and its structure implements stateless, bitwise elemental operations such as AND and OR. Unlike most machine learning datasets, the generating function is complete, deterministic and of exact precision, with approximately 1M unique input patterns producing approximately 2k output patterns. Nonetheless, simple models suggest that it is surprisingly hard to learn this model without error.**

***Index Terms*—6502 CPU, 6502 ALU, machine learning, neural structure, neural function**

I. Introduction

THE purpose of this work is to provoke thought about the strengths, weaknesses and appropriateness of various machine learning methods for modeling unknown systems. For example, there is a plethora of recent work in which supervised learning of deep networks is shown to be a simple and superior method for deriving models of unknown systems, and a strong theme in that work is that the less attention paid to intrinsic structure ("features") in the data, the better [1]. By producing a dataset with a simple generating function but an apparently large and complex input – output space, we have an opportunity to measure the efficiency and accuracy of this and other methods.

The vehicle for this work is a dataset composed of binary input and output signals for the Arithmetic Logic Unit (ALU) of a MOS Technology 6502 microprocessor. The 6502 was used in many early personal computers and game modules – most notably the Apple II, Atari 8-bit, Commodore 64 and BBC Micro series – and has become a standard theoretical platform in a number of research efforts where comparisons are made between the methods of neuroscience on living organisms, and equivalent methods applied to a silicon microprocessor. For example, Jonas and Kording [3] described the connectomics of the 6502, performed lesioning studies on the CPU, analysed the tuning curves of the transistors, measured local field potentials, and studied the spatial dynamics of activity. This new dataset aims to further that work, by investigating the extent to which a neural network model learnt from this data reveals anything useful about the underlying structure.

The particular virtues of this dataset are:

- It is relatively large, but is generated by a set of very simple binary functions.
- The underlying functions are not random; they are the canonical computational methods required for a microprocessor to function.
- It is entirely deterministic, complete (in that the output corresponding to every possible input can be generated), and the inputs and outputs are Boolean variables and hence have exact precision.
- The dataset is generated by a simple program, which has the pragmatic benefit that there is no large download required to access it.
- The real generating function of the dataset is known, and can be specified in terms of logic, or electronic circuitry, or functional structure, or software programs, allowing meaningful comparisons with size and complexity of the machine learning structures used to model it.

The advantage of using only the ALU of the 6502 (as opposed to the whole microprocessor, for example) is that the ALU is stateless, so can be modeled without time or sequence information. As our field progresses and more sophisticated datasets are required, a much larger and more complex dataset could be generated using simple programs running on the whole 6502.

II. The 6502 Arithmetic Logic Unit

The MOS 6502 was a typical 8-bit microprocessor for the period (it was designed and laid out by hand, by either a single person or a team of three – the history is controversial - in 1975). The original block diagram is shown in Figure 1.

1. GrAI Matter Labs, jtapson@graimatterlabs.ai
2. The MARCS Institute, Western Sydney University, j.tapson@westernsydney.edu.au

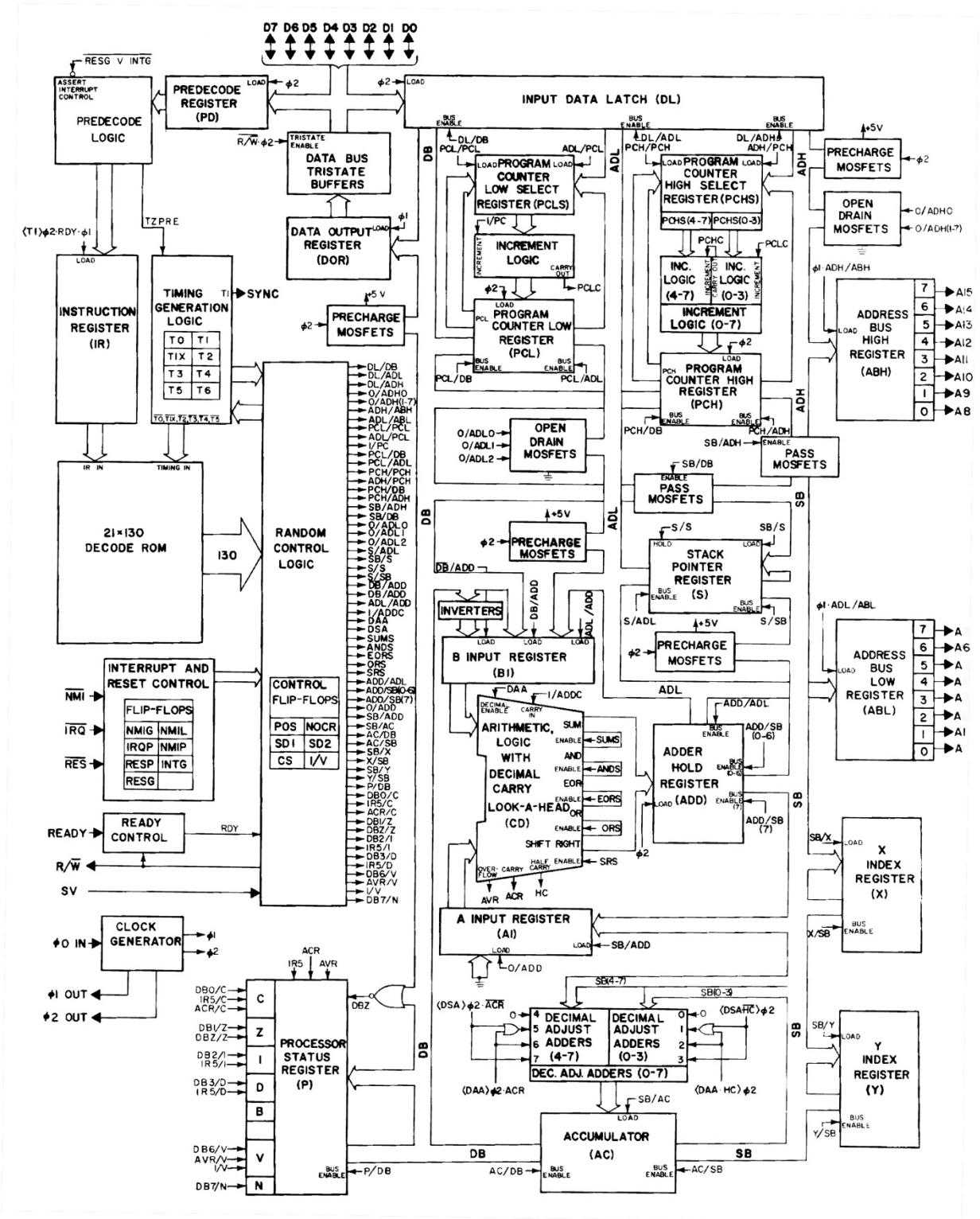

Figure 1: 6502 Microprocessor Block Diagram. The ALU, labeled "ARITHMETIC, LOGIC WITH DECIMAL CARRY LOOK-AHEAD" can be seen at lower center. It can be seen that the ALU takes inputs from the 8-bit AI and BI registers, as well as a number of single-bit instruction signals, and outputs to the 8-bit ADD register and three single-bit status flags.

It can be seen from Figure 1 that while the ALU is the computational heart of the 6502, it constitutes only a small part of the total functionality. This is reflected in the transistor count – although the whole microprocessor uses 4528 field-effect (FET) transistors (1018 of which are depletion mode FETs acting purely as resistors), the ALU uses approximately 208, of which 80 are depletion mode FETs. Figure 2 shows half of the ALU circuit in terms of logic gates. There are 94 logic gates in the ALU, each of which is instantiated with two or three FETs, of which one may be a depletion mode FET. There are also 40 pass transistors, as the circuit works by simultaneously generating the output of all five arithmetic instructions (SUM,

AND, OR, EOR or XOR, SHIFT RIGHT) and then multiplexing the correct answer onto the output lines.

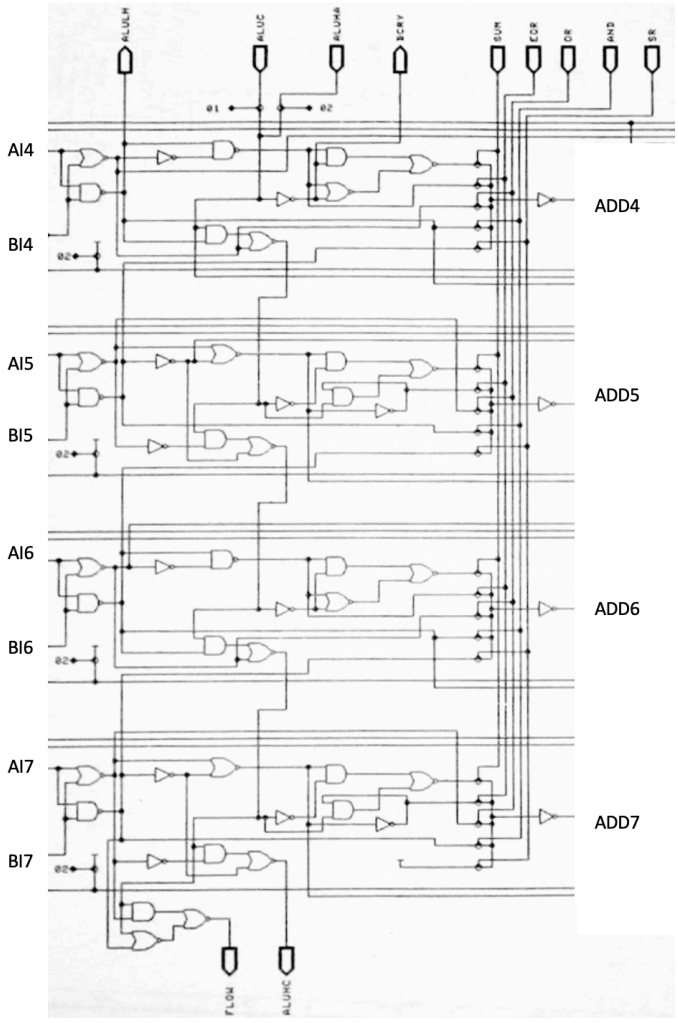

Figure 2: The logic circuit for half the ALU is shown. Bits from each data byte (AI_n and BI_n) are fed in at the left-hand side, and the instructions are input at top right. The mathematical result exits at the right as the ADD register, and the status flags at the bottom.

Notably, if we model the ALU as a neural network, it has six layers of binary logic gate "neurons", connected by binary weights, processing binary values. There are 16 neurons in the first layer, 20 in the second, 10 in the third, 16 in the fourth, 8 in the fifth layer, and 40 pass transistors in the output layer.

The ALU's inputs are as follows:

- Two input bytes of data, called AI and BI, from A and B Input Registers.
- An instruction, which is a binary signal on one of five lines – SUM, AND, EOR (exclusive or), OR and SHIFT RIGHT. Note that only one of these signals may be high at any one time.
- Two status inputs – Carry In (I/ADDC), and Decimal Enable (DAA). Decimal Enable causes the ALU to treat its inputs as binary-coded decimal (BCD) numbers.

This gives the ALU a total of 23 input lines. Given the mutual exclusivity of the instruction inputs, the input space has $5 \times 2^{18} = 1{,}310{,}720$ unique valid input samples.

The ALU's outputs are as follows:

- A byte of output data, stored in ADD.
- Three status bits – Overflow (AVR), Carry (ACR), and Half-Carry (HC). Overflow is a flag for signed arithmetic, which indicates that the output sign – which is stored as the most significant bit of the byte – has been corrupted by an overflow from the lower 7 bits. Half carry indicates a carry from the lower 4-bit "nybble" of the answer, to the upper. It is only used in BCD computation.

There are 11 output lines, giving an output space of $2^{11} = 2048$ unique output possibilities, all of which may be reachable from combinations of input data.

It can be seen that the ALU does not offer the full range of computation available to the programmer (for example, there is no subtraction instruction, and no signed computation). All of these more sophisticated possibilities can be generated by basic functions of the ALU. Note that we have not included the possibility of inverting the B input (the inverter can be seen in the block diagram, immediately above "B INPUT REGISTER"), which is required in order for the ALU to perform subtraction. It would be trivial to extend this dataset for that purpose.

The ALU's input-output functions can be expressed exactly in terms of which of the instruction inputs is high. The following pseudocode generates the entire dataset.

```
ACR = 0
AVR = 0
HC = 0

if SUM == 1
        if DAA==0
    [ACR ADD] = AI + BI + I/ADDC
    AVR = (AI(7) && BI(7) && ~ADD(7))||(~AI(7) && ~BI(7)
&& ADD(7))
  else
    [HC ADD(3:0)] = AI(3:0) + BI(3:0) + I/ADDC
    [ACR ADD(7:4)] = AI(7:4) + BI(7:4)
  end
else if AND == 1
      ADD = AI && BI
else if EOR == 1
      ADD = AI ^ BI
else if OR == 1
      ADD = AI || BI
else if SR == 1
```

```
        [ADD ACR] = AI >> 1
end
```

III. Dataset Structure

In order to generate training and testing data sets, we have shuffled the data randomly and then divided it into two halves. It might be argued that a 50:50 split is unusual, but this set is unique in that we have outputs for the entire input space, and there is no noise or ambiguity. A random 50% of this gives an exhaustive selection of input data on which to characterize the model. This is particularly so given that the essential computations are single-step bitwise operations such as AND, which have been trivially solvable since the invention of the perceptron in 1958. Of course, its inability to model XOR was famously (possibly exaggeratedly) responsible for the demise of the perceptron, but even in 1972 XOR had been solved using two-layer perceptrons. Therefore, this dataset should in principle be easily soluble without error.

The data can be downloaded from the repository in [5].

IV. Commentary on Accuracy

Notwithstanding the statement above, it is interesting to analyse the accuracy likely to be achieved on the dataset. The output data fall into two classes. 8 bits (the "data" bits) consist of the mathematical solution of a simple binary function, and therefore a random bit has approximately 50% chance of being correct (this is confirmed by the average value of these bits in the dataset, which is approximately 0.5). These bits should in principle be soluble with zero error, because the mathematical functions are very simple. The 3 flag and status bits, on the other hand, are very seldom set (2.5%, 10% and 5% of the time for AVR, ACR and HC respectively). A trivial solution could be implemented as follows:

1. Solve the 8 data bits – this should be straightforward with no more than a two-layer network.
2. Set the 3 flag bits to zero.

This solution will achieve approximately 98.4% accuracy. The challenge in this dataset is therefore to surpass this level.

There are some complications that can be expected in this endeavor. One is that the critical bits – the flag bits – are extremely unbalanced, being predominantly zeroes. Another is that the data is binary and noiseless, and hence does not intrinsically present a gradient for the purpose of descent. There are many well-known solutions to these problems in machine learning, but a naïve application of standard deep network methods may be surprisingly unhelpful.

V. Conclusion

The ALU of the 6502 microprocessor produces an input-output dataset which is simple to generate, either in terms of silicon structures, transistors, logic gates, or software code. It should be equally simple and straightforward to model the data using machine learning methods. We hope that this dataset inspires some reflection on the usefulness of machine learning in modeling computational structures in both electronics and neuroscience.

References

1. Yann LeCun, Yoshua Bengio & Geoffrey Hinton, *Deep learning*, Nature vol. 521, pp. 436–444 (2015)

2. The best source of information on the 6502 is http://6502.org/

3. Jonas E, Kording KP (2017) *Could a Neuroscientist Understand a Microprocessor?* PLoS Comput Biol 13(1): e1005268. doi:10.1371/ journal.pcbi.1005268

4. Marvin Minsky and Seymour Papert (1972) *Perceptrons: An Introduction to Computational Geometry* (2nd edition with corrections, first edition 1969), The MIT Press, Cambridge MA.

5. https://github.com/JTapson/The-6502-ALU-Dataset