
Z80

Table of Contents

Introduction....3
The Adventure Begins....4
The ALU (Arithmetic Logic Unit)....6
    ALU Slice....6
Registers....7
PLA (Programmable Logic Array)....8
Control....9
Putting it all together....10
Making it run....11
A quick dip into FPGA....12
Making it real on a DE0-Nano....13
Making it real on the DE10-Light....14

Introduction

The first computer I had the Timex Sinclair 1000, the US variant of the ZX81, which had a Z80 CPU. I quickly found out that I could bypass the limitations of Basic by using assembly language. The only limits now were the hardware. Since I learned 6502 assembly in college I quickly picked up on the Z80 assembly. I spent many hours pushing the limits of the Timex Sinclair 1000.

A few years later when I moved onto the PC I was hired as an electrical engineer for a company that made its own Z80 based computer used in quality control for the plastic molding industry. Working with the code rekindled my interest in the Z80. By that time my Timex Sinclair 1000 was long gone but the PC was more than capable of emulating it so I could once again enjoy coding Z80 assembly on the Timex Sinclair 1000.

Around 2010 I was introduced to FPGAs (Field Programmable Gate Arrays). The ability to construct hardware using Hardware Defined Language was intriguing. When I found out that you could also use logic symbols and simulate 74 series TTL chips it made me flash back to the days when I found the article for the Elf 1802 computer. I couldn't afford the 1802 on my lawn mowing profits of my neighbors but my room was littered with logic diagrams for expansion of the long sought after Elf II. I had to get a FPGA development board.

My first project was adapting a ZX81 written in VHDL to my new board. I felt triumphant when I finally got the flashing K in the corner of the screen and was able to play Mazogs on it. After that the next step was to actually design a cpu from scratch, either the 1802 or the Z80. That is where things ended. I didn't have much time to devote to it so from time to time I dabbled with my FPGA over the years, not accomplishing much. I even got another FPGA board with more bells and whistles to motivate me, which didn't work.

Over the last few years through the internet I found sites that had high resolution pictures of the Z80 substrate. Sites that reversed engineered the substrate into schematics and used them to make a Z80 on the FPGA. I also found a net level Z80 simulator and a digital simulator that could analyze a circuit and export hdl code. This sparked my interest again.

One thing I found lacking is more detailed explanation of the logic inside the Z80. So I am writing this to document what I found in my adventure.

So now armed with these resources we are ready to start our adventure by taking the deep dive into the Z80.

The Adventure Begins

The purpose of this book is to take us on an adventure through the inner workings of the Zilog Z80 computer processing unit (cpu) henceforth will be mentioned as the Z80. We will analyze the logic that makes up the Z80. All code related to this guide can be found on GitHub at

Diving Deeper into the Z80

Luckily for us there have been ardent adventurers that have blazed the trail before us. We give great thanks for this as this makes our travels easier. These adventurers are as follows;

- *Visual6502.*
  - Z80 Die shots
  - This site is hosted by a group that has decapped the Z80 and made high resolution shots of the dies which others have used for reverse engineering of the logic. It might be worth a look to see the other cpus and peripherals they have made die shots for.
- *Ken Shirriff's blog*
  - Index
  - Ken reverse engineers critical parts of the Z80 dies. He explains in detail the logic of the parts listed above. He has also reverse engineered various other cpus and shares their highlights as well.
- *Baltazar Studios*
  - Baltazar Studio
  - Goran Devic hosts this site and he has many articles where he has reversed engineer the Z80. The site mentions his Github site that has the A-Z80 a FPGA soft core made up of the logic he reversed engineered from the die. He has broken down the logic into the schematics that we use in this book. He also has coded a Z80 simulator, the Z80 Explorer, that is a netlist simulator. You can watch the die in action and drill into the logic of any part of the die.

Now any adventurer is only as good as his skills and to go on this adventure you should have a basic understanding of digital logic, reading schematics and optionally FPGAs.

We must equip ourselves for the adventure ahead so these items must be added to or gear bag.

- Digital
  - Digital on GitHub
  - This will be our go to tool to delve into the mysteries of the Z80. You can find the code for the logic diagrams shown in this guide on my GitHub page. This tool will let you simulate the logic flow in the logic diagram, generate truth tables, run test scripts, generate Verilog code with test bench script and VHDL code with test bench script. Their are other options as well for deeper understanding of the logic diagrams.
- Z80 Explorer
  - Z80 Explorer on GitHub
  - This is an optional item but well worth having if you want to dive into the depths of the Z80. This is a netlist simulator that will run assembly and highlight the areas of the die that is being used. You can also get a netlist diagram of the logic for any selected section.

Our itinerary for our journey will take us first through the fundamental parts of the Z80. The Arithmetic Logic Unit (ALU), Registers, Programmable Logic Array (PLA) and Control Logic sections will be covered in their own sections. We will then construct a simple Z80 based computer in Digital and make a simple operating system. For the truly adventurous we will take the generated Verilog and VHDL code to program a DE0-Nano and DE10-Lite FPGA development board.

The first stop, the ALU (Arithmetic Logic Unit)

The first stop in our adventure is the heart of the Z80, the ALU. This is where all the action is.

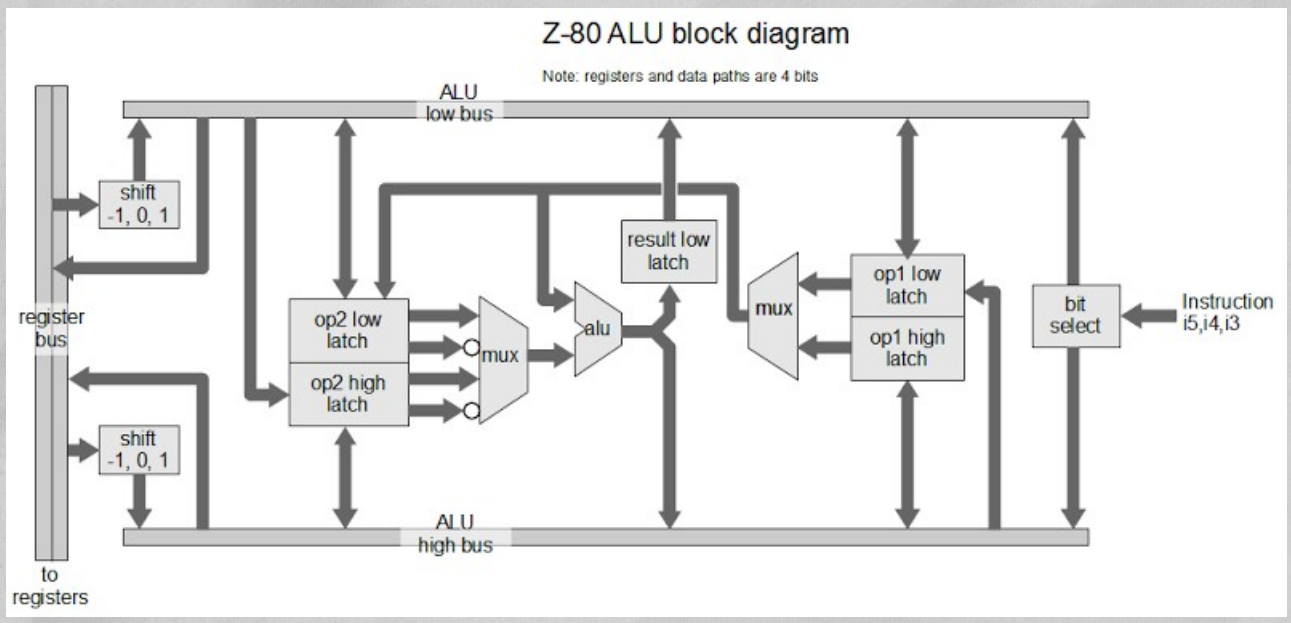
Wikipeada states:

An arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

A **combinational digital circuit** is a logic circuit that is not dependent on a clock signal to function. It is mainly made up of AND, OR, XOR, NAND, NOR, XNOR and NOT gates

The alu performs adding, subtracting, shifting, bit setting, and the gate functions AND, OR, XOR and NOT functions.

One very interesting fact is even though the Z80 is an eight bit CPU, the ALU is only four bits. This means each of the functions of the ALU must first be performed on the lower 4 bits then on the upper 4 bits. In arithmetic functions the Carry Out bit of the lower 4 bits must be retained so it can be used as the Carry In bit for the upper 4 bits.



You will notice the alu appears in the center of the diagram with support logic around it. This is because the ALU only adds, ands, ors and xors the two operands. All other functions are handled by other logic blocks.

On the left is the register bus. This bus allows the registers to directly access the ALU. The memory can also access the ALU high and low busses. Then from the register bus it goes through the shift logic. This will do a shift left one bit, right one bit or none before placing it on the high and low ALU busses. So as the data from the registers gets loaded it is shifted as needed as well.

Once the data is loaded on the ALU busses it is accessible to the op1 and op2 high and low latches. You will noticed the ALU high bus can load the op1 low latch and the ALU low bus can load the op2 high latch. This enables 4 bit binary coded decimal (BCD) shifts RRD and RLD.

The muxes are multiplexers that allow the switching from low 4 bit to high4 bit latches since the alu only performs 4 bit functions. Because of this, a result low latch needs to latch the results onto the ALU low bus so once the high bits result is is placed on the ALU high bus all 8 bits are available. The op2 mux also permits the data of the op2 latch to be inverted. This permits subtraction by using 2s complement addition.

2s Complement is done by inverting the second operand then adding it to the first operand then adding 1 then discarding any carry. For example

```
 10 - 8 =2
    1010  (10)
 +  0111  (8 inverted)
 +  0001 (1)
 =  0010 (2)
```

Finally the result is can have each individual bit of the 8 bits to be set or cleared. The bits 3, 4, and 5 of the instruction byte designate which bit to perform the operation on.

ALU Slice

Registers

PLA (Programmable Logic Array)

Control

Putting it all together

Making it run

A quick dip into FPGA

Making it real on a DE0-Nano

Making it real on the DE10-Light