Toxic Processor

A simplistic 4-bit processor ready to synthesis

Version 2.0.0

Entropy Xu
entropy.xcy@protonmail.com

May 7, 2019

Contents

Chapter 1

Introduction

1.1 Intention

1.2 Advantages

1.3 Limitations

1.4 History of Revisions

Chapter 2

Design of the Processor

2.1 Overview

This processor is a **4-bit Stack machine** (0-address machine).

- The addressing width is configurable to be a multiple of 4.
- The stack depth is configurable to be greater than 16.
- The width of each Instruction is 4 bits. Thus, 16 Instructions in total.
- The width of each block inside the stack is 4 bits.
- Von Neumann Architecture: seperate data memory and code memory.
- Code memory and data memory are using the same addressing space.

2.2 Data Structure

The name **stack machine** or equivalently 0-address machine means that there is no addressable register in this machine neither do operands in the instructions.

In order to store temporary data in this processor, we use a hardware stack to replace the register file which is usually implemented by other popular processors.

The 4-bit block-data-width will not limit the scalability of this processor in that addressing width of this processor is 4-bits but a configurable width of a multiple of 4 (usually 8 bits or 12 bits or 16 bits).

2.2.1 Stack for Storing Temporary Data

We have a stack for storing temporary data. Stack is a LIFO (Last In First Out) data structure. Each block of the stack is a 4-bits register. The stack supports common operations like push and pop. For each Instruction we execute, we will have to read value from TOS (Top of Stack) and NTOS (Next Top of Stack), and push the result of the operations to the stack. (Refer to Figure 2.1 for the model of the stack)

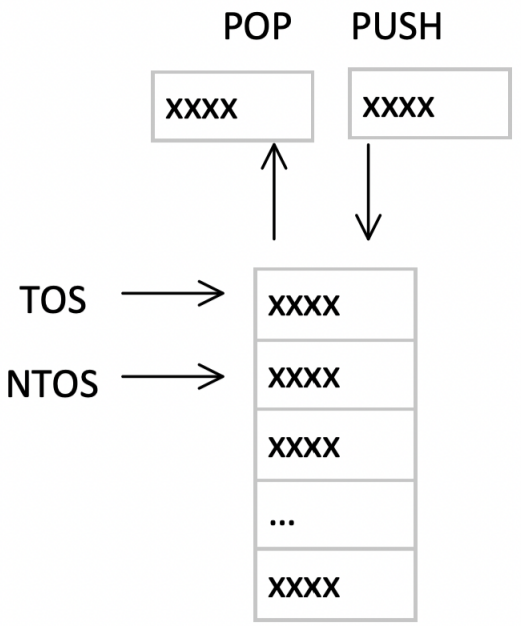

Figure 2.1: Toxic Stack Model

2.2.2 Queue for Addressing Bus

We have discussed that the addressing width for the Toxic processor is configurable and usually more than 4 bits. Thus, we establish a Queue as the data structure for storing the address of the Bus. For the length of the queue, we have $Length(Queue) = BitWidth(BusAddress)/4 * 2$

Queue is a FIFO data structure. However, the Queue data structure we used in the Toxic processor is similar to the common Queue but with some tweaks. Same as the stack, each block in the Queue is a 4-bits register. The

Queue should supports common operations like **enqueue** and **dequeue**. Refer to Figure 2.2 for the model of the queue and the connection with the BusAddress. Noted that, when we **enqueue**, all the blocks shift left one position relative to the BusAddress; when we **dequeue**, all the blocks shift right one position relative to the BusAddress.

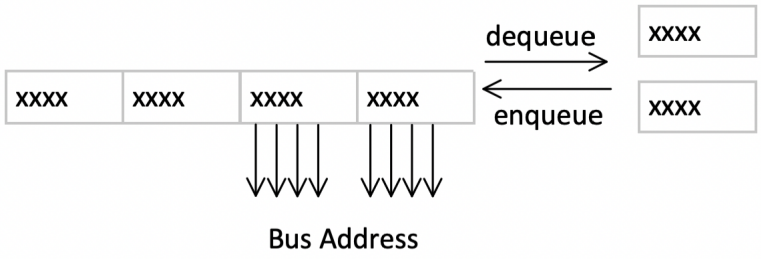

Figure 2.2: Toxic Queue Model

2.2.3 Memory

The memory of the Toxic processor is accessed through Bus and only takes part of the addressing space. The model of the Memory is different from common RAMs since for common RAMs, the block bit width is 1 bype (8 bits) while for the Toxic processor, the width of a block is half-byte (4 bits).

For the Toxic processor, we assign the width of each block of the memory to be 4 bits. This memory model can be easily implemented using a standard memory block and would be discussed in detail in Chapter 4. Detailed addressing space definitions can be found in detail in section 2.4.

2.3 Addressing Modes

There are three addressing modes in the Toxic processor:

- TOS (Top of Stack)
- NTOS (Next Top of Stack)
- Bus for memory and peripherals

2.4 Addressing Space

These definitions for the addressing space is for the standard version of the Toxic processor. More versions of definitions of implementations can be found in Chapter 4

2.4.1 Reserved: 0x0-0xf

2.4.2 Code Memory: 0x10-0xffff

2.4.3 Data Memory: 0x1000

2.4.4 Peripherals:

Chapter 3

Instruction Set Architecture

3.1 Instructions Map

1:0\3:2	00	01	11	10
00	P0	POP	ADD	SV
01	P1	DIS	NAND	LD
11	CMP	SWP	LS	B1
10	CD	RVS	RS	B0

3.2 Push Instructions

3.2.1 P0

3.2.2 P1

3.3 Stack and Bus Operation Instructions

3.3.1 POP

3.3.2 DIS

3.3.3 SWP

3.3.4 RVS

3.4 Numeric Computing Instructions

3.4.1 ADD

3.4.2 NAND

3.4.3 LS

3.4.4 RS

3.5 Memory Operations Instructions

3.5.1 SV

3.5.2 LD

3.5.3 CD

3.6 Branch Instructions

3.6.1 B1

3.6.2 B0

3.7 Special Instructions

3.7.1 CMP

Chapter 4

Implementation

4.1 Modules List

4.2 Data Path