Instruction Set Architecture Design and Implementation

Learning Outcomes

At the end of this lab, you should be able to:

1. understand and modify a simple ISA;
2. write a simple assembler

Contents

1 Resources

- Video: https://youtu.be/CzyXb_T-xgU.
- Source Codes: https://git.io/JU3al
- Online VHDL Tool: https://www.edaplayground.com/home

2 Discussion

The **processor(CPU)** is composed of the **datapath** and **control**. In the previous labs, you learned that combinational and sequential circuit elements are used as building blocks to create the functional components of the datapath and control. Examples of these functional elements include the **ALU, Register File, Program Counter, and Memory**. You also learned that a **clock** drives the execution and control is implemented using a **finite state machine** for **fetch-decode-execute cycle**. One question that we can answer next is: *How do we program the CPU?*

 ©JACHermocilla (CC NC-BY-SA 4.0)

2.1 Instruction Set Architecture

Instruction Set Architecture (ISA) is an **abstraction** between the hardware and the lowest-level software. It includes anything programmers need to know to make a binary machine language program work correctly. Typically it documents the set of instructions that can be performed by the processor, number and name of available registers, memory addressing modes, I/O, interrupt processing, etc.

ISA allows computer designers to talk about functions indepedently from the hardware that performs them. This abstact interface enables many implementations (aka **microarchitectures**) of varying costs and performance to run identical software.

Examples of ISA include the **IA-32** and **x86-64** which are commonly used in desktop and laptops. Intel implements these ISA in their Intel Core i5 product as **8th Generation aka as Kaby Lake Refresh**. AMD also implements these ISA in the Ryzen 5000 as **4th Gen aka Zen 3**. There are other implementations(aka generations) that vary in their performance characteristics.

For mobile devices, a popular ISA is the **ARMv8 A64**. MediaTek uses ARM's **Cortex-A73** and **Cortex-A53** implementations in their Octa-core Helio P70. Qualcomm also uses the same implementations in their Kryo 240 processor for Snapdragon SoC.

2.2 Application Binary Interface

Application Binary Interface (ABI) is a combination of the basic instruction set and the operating system interface provided for application programmers.

For general-purpose use such a desktops and laptops, programming a processor using only the basic instruction set is inefficient. Thus, operating systems perform an important role in the management and efficient use of hardware resources in addition to making it easier for users to use a computer.

ABI describes function-calling conventions, parameter passing, sizes of C data types, executable file formats (ELF, PE). Examples are the IA-32 and **x86-64 System V ABI** which is used in Linux and other Unix-type operating systems. In Windows, it uses its own **x64 ABI**.

2.3 ISA Taxonomy

We can categorize ISAs based on where operands in instructions are stored. **Stack-based** ISAs uses a stack(LIFO) where operations are performed on the operands on the top of the stack. In **accumulatr-based** ISAs, one register is designated as accumulator and its use in operations is implied. Modern ISAs are **general purpose** where operands are explicitly named in the instruction. Operations can be register-to-register, register-to-memory, or memory-to-register.

2.4 Considerations in ISA Design

- *Types/Class of instructions(Operations in the instruction set)* - arithmetic/logic, data movement, branching/control flow, I/O, etc.
- *Types and sizes of operands* - 8, 16, 32, 64, 128, floating point
- *Addressing modes* - register, direct, indirect, immediate, etc.
- *Addressing memory* - byte-addressable, word-addressable
- *Encoding and Instruction Formats* - opcode field, addresses field, mode field
- *Compiler-related issues* - optimization features

 ©JACHermocilla (CC NC-BY-SA 4.0)

2.5 Example Instruction Formats

Below are the instruction formats for x86-64 and ARMv8 taken from documentation manuals. The x86-64 format is more complex than that of the ARMv8.

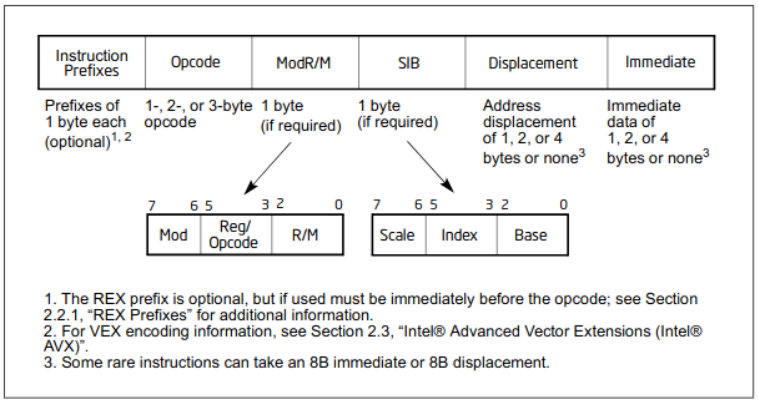
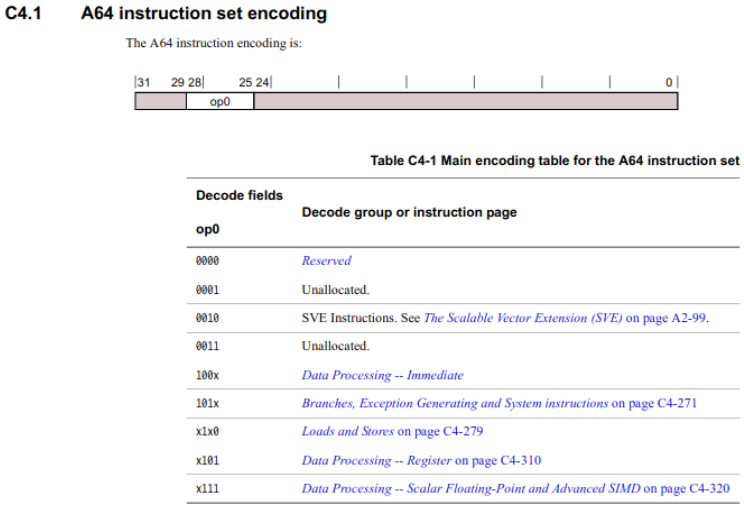


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Figure 1: x86-64 Instruction Format (CISC).



C4.1 A64 instruction set encoding

The A64 instruction encoding is:

31 29 28 25 24 0
op0

Table C4-1 Main encoding table for the A64 instruction set

Decode fields op0	Decode group or instruction page
0000	*Reserved*
0001	Unallocated.
0010	SVE Instructions. See *The Scalable Vector Extension (SVE)* on page A2-99.
0011	Unallocated.
100x	*Data Processing -- Immediate*
101x	*Branches, Exception Generating and System instructions* on page C4-271
x1x0	*Loads and Stores* on page C4-279
x101	*Data Processing -- Register* on page C4-310
x111	*Data Processing -- Scalar Floating-Point and Advanced SIMD* on page C4-320

Figure 2: ARMv8 Instruction Format (RISC).

2.6 The Optimal Machine Architecture (TOMA) ISA

2.6.1 Features

- Four 8-bit registers named $s0, $s1, $s2, $s3 in assembly code
- Instruction memory is 8 bytes(8x8), address line is 3 bits
- Three-bit Program Counter (PC)
- Single-cycle - completes instruction execution in one clock cycle
- No data memory, thus has no load and store instructions
- No control transfer instructions

 ©JACHermocilla (CC NC-BY-SA 4.0)

2.6.2 Instruction Format

The size of an instruction in TOMA is 8 bits divided into the configuration shown in Figure 3

Figure 3: TOMA instruction format.

 ©JACHermocilla (CC NC-BY-SA 4.0)

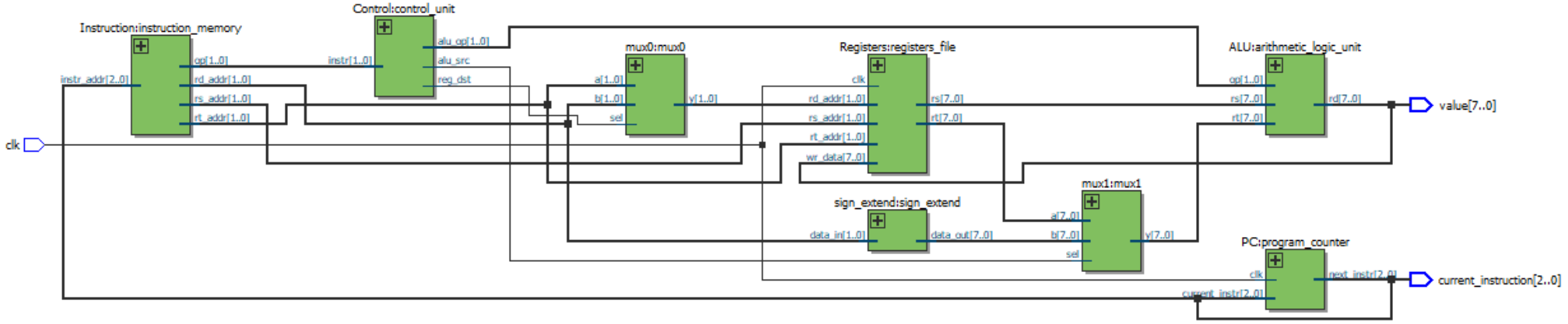


Figure 4: REDHORSE 500.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY clocker_tb IS
END clocker_tb;
ARCHITECTURE behavior OF clocker_tb IS
   --100Mhz
   CONSTANT frequency: integer := 100e6;
   CONSTANT period : time := 1000 ms / frequency;
   SIGNAL clk : std_logic := '0';
BEGIN
   clk <= not clk after period / 2;
   -- do some stuff here using clk as input
END ARCHITECTURE;
```

3 Summary

In this lab, you learned some of the sequential elements that are useful in the design of a processor as well as the importance of clocks. We also showed the design and implementation of a simple traffic light system using finite state machines since a simple truth table is not enough to characterize a sequential system.

You should now be able to tell whether a functional component of a datapath and control is composed of a combinational or sequential element.

4 Learning Activities

Download the source codes for this lab then try experimenting by adding more test cases in the testbenches. Submit a PDF document that shows screenshots of your modifications and runs.

5 Self-Assessment Questions

1. What is the main purpose of clocks in sequential circuits?
2. What is the difference between a clocked latch and a flip-flop?
3. Why can't a multiplexer be used in RAM?
4. Why is SRAM more expensive than DRAM?
5. If my CPU is clocked at 800 MHz, what is the period?

6 Deliverable

Your final deliverable for this lab is implement the RAM in Figure **??**. Submit the VHDL code including a testbench as well as images of the waveforms. NOTE: Enable lines should be connected to the output of the decoder and the rightmost Din in the figure should be Din[0].

7 Further Reading

- https://www.doulos.com/knowhow/vhdl/simple-ram-model/

 ©JACHermocilla (CC NC-BY-SA 4.0)

References

[1] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, ARM Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, arm edition, 2017.

 ©JACHermocilla (CC NC-BY-SA 4.0)