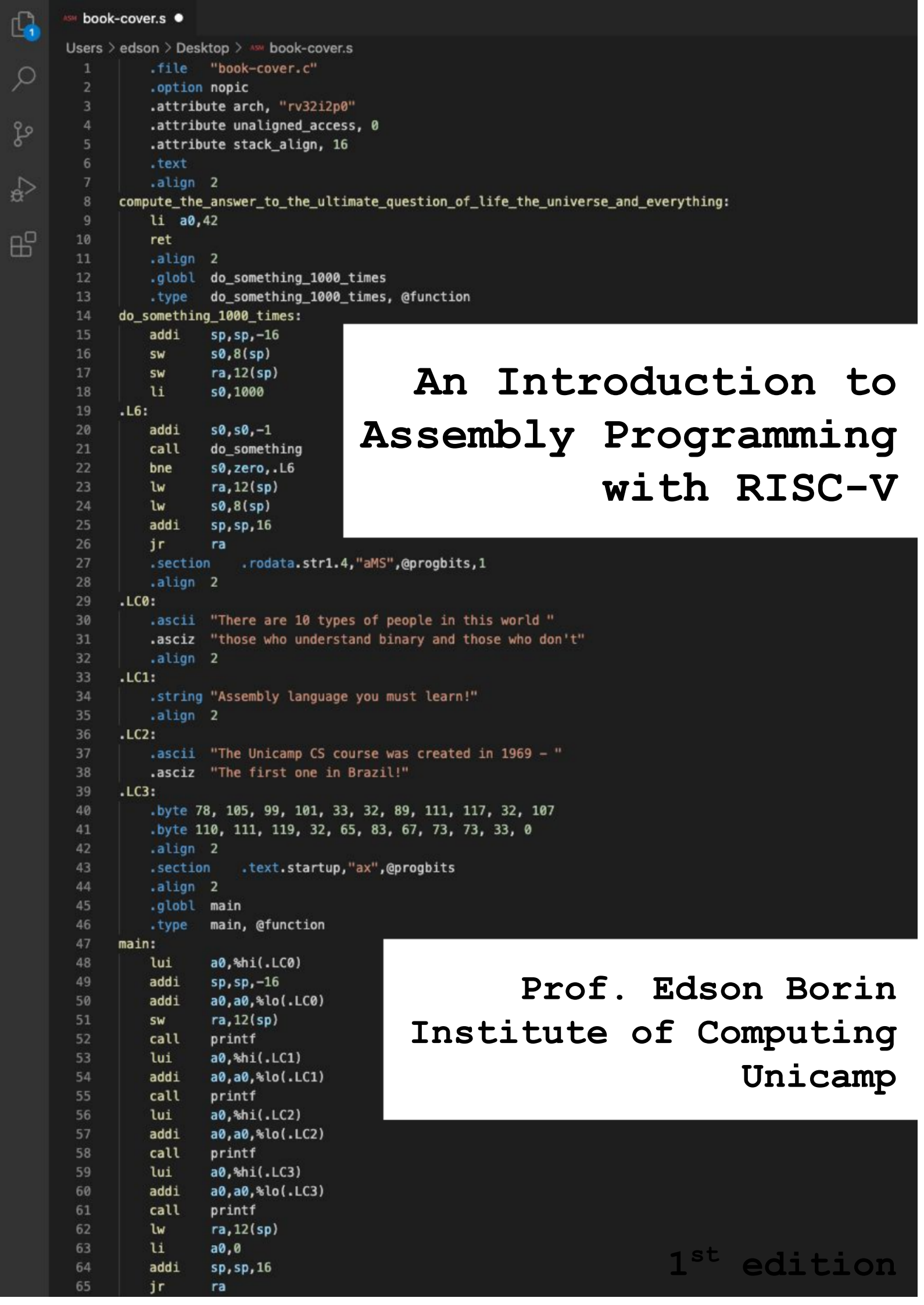

An Introduction to Assembly Programming with RISC-V

Prof. Edson Borin
Institute of Computing
Unicamp

1st edition

An Introduction to Assembly Programming with RISC-V

ISBN:978-65-00-15811-3
First edition 2021

Edson Borin
Institute of Computing - University of Campinas
Av. Albert Einstein, 1251
Cidade Universitária Zeferino Vaz
Barão Geraldo - Campinas - SP - Brasil
`www.ic.unicamp.br/~edson`
13083-852

An updated version of this book and other material may be available at: `riscv-programming.org`

Foreword

This book focuses on teaching the art of programming in assembly language, using the RISC-V ISA as the guiding example. Towards this goal, the text spans, at an introductory level, the organization of computing systems, describes the mechanics of how programs are created and introduces basic programming concepts including both user level and system programming. The ability to read and write code in low-level assembly language is a powerful skill to be able to create high performance programs, and to access features of the machine that are not easily accessible from high-level languages such as C, Java or Python, for example to control peripheral devices.

The book introduces the organization of computing systems, and the mechanics of creating programs and converting them to machine-readable format suitable for execution. It also teaches the components of a program, or how a programmer communicates her intent to the system via directives, data allocation primitives and finally the ISA instructions, and their use. Basic programming concepts of control flow, loops as well as the runtime stack are introduced.

Next the book describes the organization of code sequences into routines and subroutines, to compose a program. The text also addresses issues related to system programming, including notions of peripheral control and interrupts.

This text, and ancillary teaching materials, has been used in introductory classes at the University of Campinas, Brazil (UNICAMP) and has undergone refinement and improvement for several editions.

Mauricio Breternitz
Principal Investigator & Invited Associate Professor
ISTAR ISCTE Laboratory
ISCTE Instituto Universitario de Lisboa
Lisbon, Portugal

Contents

Acronyms

ABI Application Binary Interface. v, 47, 71, 76–79, 83, 85, 86, 104

ASCII American Standard Code for Information Interchange. v, 14–16

bit Binary digit. v, 2–4, 10–16, 18, 19, 22, 24, 25, 30, 32, 34, 37–39, 41–43, 86, 91–96, 100, 101, 104

CPU Central Processing Unit. v, 2–5, 27, 31, 42, 45, 47, 90–92, 94–105, 107–110

CSR Control and Status Register. v, 100–104, 107–110

ISA Instruction Set Architecture. v, 3, 4, 21, 22, 33, 34, 42, 43, 46–49, 53–57, 59–61, 63, 64, 70, 91, 92, 100, 101, 103, 104, 106–108

ISR interrupt service routine. v, 98–100, 103–105

PC program counter. v, 4, 5, 23, 27, 98, 101, 102, 104, 107–109

UTF-8 Universal Coded Character Set (or Unicode) Transformation Format - 8-bit. v, 14–16

Glossary

32-bit address space is the set of addresses represented by 32-bit unsigned numbers. v, 46

binary digit is a digit that may assume one of two values: "0" (zero) or "1" (one). v, 10

bus is a communication system that transfers information between the computer components. This system is usually composed of wires that are responsible for transmitting the information and associated circuitry, which are responsible for orchestrating the communication. v, 2, 90, 91, 94

byte addressable memory is a memory in which each memory word stores a single byte. v, 2–4, 15–19, 47

Central Processing Unit (CPU) is the computer component responsible for executing the computer programs. v, 2, 3

column-major order specifies that the elements of a two-dimensional array are organized in memory column by column. In this context, the elements of the first column are placed first then the elements of the second column are placed after the elements of the first one and so on. v

Control and Status Register (CSR) is an internal CPU register that exposes the CPU status to the software and allows software to control the CPU behavior. v, 100, 101, 107, 109

endianness refers to the order in which the bytes are stored on a computing system. There are two common formats: little-endian and big-endian. The little-endian format places the least significant byte on the memory position associated with the lowest address while the big-endian format places the most significant byte on the memory position associated with the lowest address. v, 16, 39, 54, 56, 57

exceptions are events generated by the CPU in response to exceptional conditions when executing instructions. v

external interrupts are interrupts caused by external (non-CPU) hardware, such as peripherals, to inform the CPU they require attention. v

hardware interrupts are events generated by hardware, such as peripherals, to inform the CPU they require attention. v

immediate value is a number that is encoded into the instruction encoding. As a consequence, it is a constant. v, 49–58, 61, 112, 113

Instruction Set Architecture (ISA) defines the computer instructions set, including, but not limited to, the behavior of the instructions, their encoding, and resources that may be accessed by the instructions, such as CPU registers. v, 3, 42, 46, 52, 100, 106

integer overflow occurs when the result of an arithmetic operation on two integer m-bit binary numbers is outside of the range that can be represented by an m-bit binary number. v, 13, 14

interrupt service routine (ISR) is a software routine that handles interrupts. It is also known as interrupt handler. v, 98, 99, 101–103, 105, 107–110

interrupt vector table is a table that maps interrupt/exception identifiers to routines that must be invoked to handle the interrupt/exception. The interrupt vector table is usually stored in main memory and accessed by the CPU hardware to invoke the proper routine when handling an interrupt/exception. v, 109

ISA native datatype is a datatype that can be naturally processed by the ISA. v, 47, 55

load instruction is an instruction that loads a value from main memory into a register. v, 48

Load/Store architecture is a computer architecture that requires values to be loaded/stored explicitly from/to main memory before operating on them. v, 48

machine language is a low-level language that can be directly processed by the computer's central processing unit (CPU). v, 22

main memory is a storage device used to store the instructions and data of programs that are being executed. v, 2–5, 15, 17–19, 23, 27, 28, 30, 31, 41, 90–92, 94, 96–100, 103, 104, 107

native program is a program encoded using instructions that can be directly executed by the CPU, without help from an emulator or a virtual machine. v, 2, 21, 22

numeral system is a system used for expressing numbers. v, 6–10

opcode (operation code) is a code that indicates the operation that an instruction must perform. It is usually encoded as a binary number into the instruction. v, 49

peripherals are input/output, or I/O, devices that are connected to the computer. Examples of peripheral devices include video cards (also known as graphics cards), USB controllers, network cards, *etc*. v, 2, 90

persistent storage is a storage device capable of preserving its contents when the power is shut down. Hard disk drives (HDDs), solid state drives (SSDs), and flash drives are example of persistent storage devices. v, 2, 90

positional numeral system is a numeral system in which the value of a digit d_i depends on the its position on the sequence. v, 6–10

privilege level defines which ISA resources are accessible by the software being executed. v, 100, 106

privilege mode defines the privilege level for the currently executing software. v, 106–109

program counter (PC) is the register that holds the address of the next instruction to be executed. In other words, it holds the address of the memory position that contains the next instruction to be executed. It is also known as instruction pointer, or IP, in some computer architectures. v, 48

pseudo-instruction is an assembly instruction that does not have a corresponding machine instruction on the ISA, but can be translated automatically by the assembler into one, or more, alternative machine instructions to achieve the same effect. v, 34, 49, 50, 58

register is a small memory device usually located inside the Central Processing Unit (CPU) for quick read and write access. v, 3

row-major order specifies that the elements of a two-dimensional array are organized in memory row by row. In this context, the elements of the first row are placed first then the elements of the second row are placed after the elements of the first one and so on. v, 17

stack pointer is a pointer that points to the top of the program stack. In other words, it holds the address of the top of the program stack. In RISC-V, the stack pointer is stored by the `sp` register. v

store instruction is an instruction that stores values into main memory. v

unprivileged ISA is the sub-set of the ISA that is accessible by the software running on unprivileged mode. v, 48, 106

unprivileged mode is the privilege mode with least privileges. In RISC-V, it is the User/Application privilege mode. v, 106, 107

unprivileged registers are a set of registers accessible on the unprivileged mode. v, 48

user application is an application designed to be executed at user-mode on a system managed by an operating system. v

user-mode on RISC-V, the user-mode is equivalent to the User/Application mode. v, 106

Part I

Fundamental concepts

Chapter 1

Execution of programs: a 10,000 ft overview

There are several ways of encoding a computer program. Some programs, for example, are encoded using abstract instruction sets and are executed by emulators or virtual machines, which are other programs designed to interpret and execute the abstract instruction set. Bash scripts, Java byte-code programs, and Python scripts are common examples of programs that are encoded using abstract instruction sets and require an emulator or a virtual machine to support their execution.

A **native program is a program encoded using instructions that can be directly executed by the computer hardware, without help from an emulator or a virtual machine**. In this book, we focus our discussion on native programs. Hence, from now on, whenever we use the term "program", unless stated otherwise, we are referring to native programs.

Native program instructions usually perform simple operations, such as adding or comparing two numbers, nonetheless, by executing multiple instructions, a computer is capable of solving complex problems.

Most modern computers are built using digital electronic circuitry. These machines usually represent information using voltage levels that are mapped to two states, HIGH and LOW, or "1" (one) and "0" (zero). Hence, the basic unit of information on modern computers is a binary digit, *i.e.*, "1" or "0". Consequently, information and instructions are encoded as sequences of **bi**nary digi**t**s, or **bit**s.

1.1 Main components of computers

Computers are usually composed of the following main components:

- **Main memory**: The main memory is used to store the instructions and data of programs that are being executed. The main memory is usually volatile, hence, if the computer is turned off, its contents are lost.
- **Central Processing Unit**: the Central Processing Unit, or CPU, is the component responsible for executing the computer programs. The CPU retrieves programs' instructions from the main memory for execution. Also, when executing instructions, the CPU often reads/writes data from/to the main memory.
- **Persistent storage**: Since the main memory is volatile, there is usually a persistent storage device to preserve the programs and data when the power is shut down. Hard disk drives (HDDs), solid state drives (SSDs), and flash drives are example of persistent storage devices.
- **Peripherals**: Peripherals are input/output, or I/O, devices that are connected to the computer. Examples of peripheral devices include video cards (also known as graphics cards), USB controllers, network cards, *etc.*

- **Bus**: The bus is a communication system that transfers information between the computer components. This system is usually composed of wires that are responsible for transmitting the information and associated circuitries, which orchestrate the communication.

Figure 1.1 illustrates a computer system in which the CPU, the main memory, a persistent storage device (HDD), and two I/O devices are connected through a system bus.

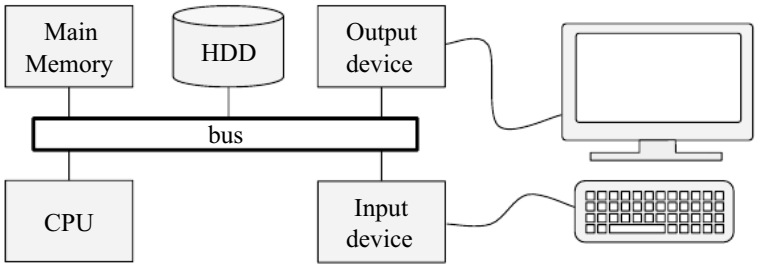

Figure 1.1: Computer system components connected through a system bus.

1.1.1 The main memory

The computer main memory is a storage device used to store the program instructions and data, and it is composed of a set of memory words. Each memory word is capable of storing a set of bits (usually eight bits) and is identified by a unique number, known as the memory word address. A byte addressable memory is a memory in which each memory word (a.k.a. memory location) stores a single byte and is associated with a unique address. Figure 1.2 illustrates the organization of a byte addressable memory. Notice that the memory word identified by address 5 (or simply "memory word 5") contains the value 11111111_2 while memory word 0 contains the value 00110110_2.

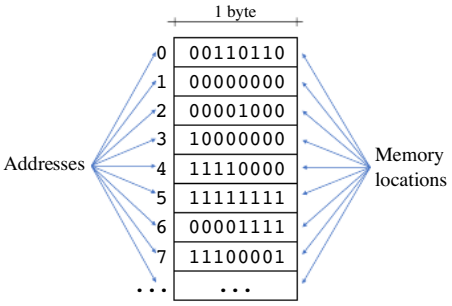

Figure 1.2: Organization of a byte addressable memory with its contents represented as a sequence of bits.

1.1.2 The CPU

The Central Processing Unit is the component responsible for executing the computer programs. There are several ways of implementing and organizing a CPU, however, to understand how programs are executed, it suffices to know that the CPU contains:

- **Registers**: a CPU register is a small memory device located inside the CPU. The CPU usually contains a small set of registers. RISC-V processors, for example, contain thirty-one 32-bit registers[1] that can be used by programs to store information inside the CPU. Computers often contain

[1] A 32-bit register is a register that is capable of storing 32 bits, *i.e.*, values composed of 32 bits.

instructions that copy values from the main memory into CPU registers, known as "load" instructions, and instructions that copy values from the CPU registers into the main memory, known as "store" instructions.

- **A datapath**: the CPU datapath is responsible for performing operations, such as arithmetic and logic operations, on data. The datapath usually performs the operation using data from the CPU registers and store the results on CPU registers.
- **A control unit**: the control unit is the unit responsible for orchestrating the computer operation. It is capable of controlling the datapath and other components, such as the main memory, by sending commands through the bus. For example, it may send a sequence of commands to the datapath and to the main memory to orchestrate the execution of a program instruction.

Accessing data on registers is much faster than accessing data on the main memory. Hence, programs tend to copy data from memory and keep them on CPU registers to enable faster processing. Once the data is no longer needed, it may be discarded or saved back on the main memory to free CPU registers.

The Instruction Set Architecture, or ISA, defines the computer instructions set, including, but not limited to, the behavior of the instructions, their encoding, and resources that may be accessed by the instructions, such as CPU registers. A program that was generated for a given ISA can be executed by any computer that implements a compatible ISA.

ISAs tend to evolve over time, however, ISA designers try to keep newer ISA versions compatible with previous ones so that legacy code, *i.e.*, code generated for previous versions of the ISA, can still be executed by newer CPUs. For example, a program that was generated for the 80386 ISA can be executed by any processor that implements this or any other compatible ISAs, such as the 80486 ISA.

1.2 Executing program instructions

As discussed previously, modern computers usually store the program that is being executed on main memory, including its instructions and data. The CPU retrieves programs' instructions from the main memory for execution. Also, when executing instructions, the CPU may read (write) data from (to) the main memory. To illustrate this process we will consider a CPU that implements the RV32I ISA.

The RV32I ISA specifies that instructions are encoded using 32 bits. Hence, assuming the system has a byte addressable memory[2], each instruction occupies four memory words. Also, it specifies that instructions are executed sequentially[3], in the same order they appear in the main memory.

Let us consider a small program generated for the RV32I ISA that is composed of three instructions and is stored in main memory starting at address 8000. Since each instruction occupies four bytes (*i.e.*, 32 bits) and instructions are stored consecutively on main memory, the first instruction is located at addresses 8000, 8001, 8002, and 8003, the second one on addresses 8004, 8005, 8006, and 8007, and the third one on addresses 8008, 8009, 8010, and 8011. Figure 1.3 illustrates the instructions stored on the main memory.

The CPU usually contains a register to keep track of the next instruction that needs to be executed. This register, called Program Counter, or PC, on the RV32 ISA, stores the starting address of the sequence of memory words that store the next instruction to be executed. For example, before executing the first instruction of the code illustrated at Figure 1.3, the PC contains the value 8000. Once the instruction stored at address 8000 is fetched, the value of the PC is updated by adding four to its contents so that the next instruction (at address 8004) can be fetched for execution once the current instruction is completed. Algorithm 1 illustrates the execution cycle performed by a simple RV32I CPU. First, the CPU uses the address in the PC to fetch an instruction (a sequence of four memory words, *i.e.*, 32 bits) from main memory and store it on an internal register called `IR`. Then, it updates the PC so it points to the next instruction in memory. Finally, it executes the instruction that was fetched from

[2]This is usually the case in modern computers.

[3]As discussed in Section 5.7, control-flow instructions may change the normal execution flow.

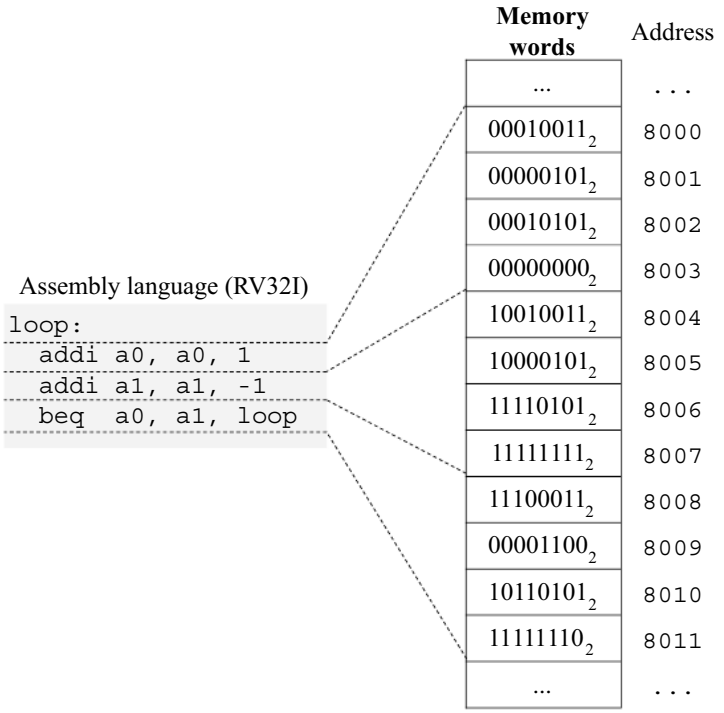

Figure 1.3: Three RV32I instructions stored on a byte addressable memory starting at address 8000.

memory. Notice that when executing the instruction, the CPU may also access the main memory to retrieve or store data.

Algorithm 1: RV32I instructions execution cycle.

```
1 while True do
      /* Fetch instruction and update PC */
2     IR ← MainMemory[PC];
3     PC ← PC+4;
4     ExecuteInstruction(IR);
5 end
```

To execute a program, the operating system essentially loads the program into the main memory (*e.g.*, from a persistent storage device) and sets the PC so it points to the program entry point.

1.3 The boot process

Since the main memory is volatile, whenever a computer is powered on, it contains garbage. As a consequence, at this point, the CPU may not retrieve instructions from the main memory. In this context, on power on, the PC is automatically set so that the CPU starts by retrieving instructions from a small non-volatile memory device, which stores a small program that performs the boot process[4]. This program sets up the basic computer components, checks the boot configuration (also stored on a non-volatile memory), and, based on its settings, loads into main memory the operating system boot loader from a persistent storage device (*e.g.*, the hard disk drive).

Once the operating system boot loader is loaded into memory, the CPU starts executing its code, which, in turn, finishes setting up the computer and loading the primary operating system modules into

[4]In old personal computer systems this program is known as the Basic Input/Output System, or BIOS. More modern computers use the Unified Extensible Firmware Interface, or UEFI, standard.