Specification and Implementation of the Didactic Processor P4 and Its Development Environment

Dinis Pedro Pinto Marcos Madeira
dinismadeira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

April 2019

Abstract

The "Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores" [1] book presents two didactic processors, the Small Pedagogical Processor (P3), a CISC processor, and the Small Pedagogical Processor with Pipeline (P4), a RISC processor. The P3 processor has been implemented before in hardware as a learning platform for the laboratory classes of the Computer Architecture course for the Computer Science degree at Instituto Superior Técnico. Considering an evolution of the course from a focus on CISC to a focus on RISC, the work on this paper consists of the specification of the P4, and its implementation on a reconfigurable hardware, supporting a set of I/O devices, including LEDs, switches, push buttons, hexadecimal displays, an alphanumeric display, a VGA display, a PS/2 keyboard and a 3-axis accelerometer. Additionally, the creation of the development environment based in a web platform, including an assembly editor, an assembler, debugging tools and a simulator, which can be accessed online with a web browser, with no need to download files or installing software in the user's system.
Keywords: Processor, P4, RISC, Learning Platform.

1. Introduction

The processor is the component in the computer responsible for performing arithmetic, logic and control operations. It also controls the input and output from storage devices, memories, caches, peripherals, and so on. The operations performed by the processor are specified as machine-language instructions, which consist of sets of bits.

Each processor has an architecture that can be characterized by the length of the data units it works on, the instruction set, what operations these instructions perform, how complex they are, the ability to directly perform these operations with operands in memory or with registers only, etc.

The Instruction Set Architecture (ISA) is one of the first abstractions of the processor, providing an interface between hardware and software at the lowest level. An ISA may have different hardware implementations, but it is still possible to run the software that has been developed for an ISA in all processors that implement it.

There are two types of ISA: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). The CISC architectures are characterized by having single instructions that perform many operations possibly requiring several clock cycles to complete. On the other hand, the RISC architectures usually implement only instructions that perform simple operations that usually execute on average close to a single clock cycle per instruction because of pipelining, which consists in dividing the execution of each instruction into several sequential steps corresponding to different stages. This means that there are several instructions being executed at the same time, each one in a different stage, like an assembly line in a factory.

In Figure 1 we can see how a 4 stage pipeline works while executing 6 instructions in nine clock cycles. Each instruction takes 4 clock cycles to execute, however, once the pipeline is filled, there are 4 instructions being executed simultaneously at different stages, thereby achieving an average of one instruction executed per clock cycle.

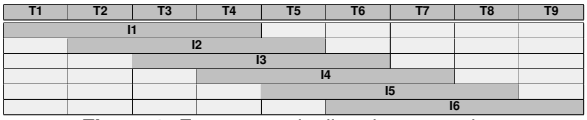

Figure 1: Four stage pipeline demonstration.

However, the fact that the execution of an instruction begins before the previous instructions complete may lead to conflicts when the execution of an instruction depends on the results of the pre-

vious instructions. Resolving these conflicts is one of the issues addressed in the hardware specification and implementation of the P4.

The main differences between CISC and RISC architectures are:

- Number and complexity of the instructions: CISC architectures support instructions for complex operations, whereas RISC architectures only provide instructions for simpler and more commonly used operations.
- Instruction formats: RISC architectures have instructions with a regular and fixed length format, which allows for simpler instruction decoding circuits, whereas the CISC architectures have more complex and variable length instruction formats.
- Addressing Modes and Memory Access: RISC architectures have a smaller number of addressing modes. Processors based on architectures RISC are described as load-store machines, since in order to perform operations, the operands are first loaded to registers and then the result is stored in memory, unlike CISC architectures, which usually have addressing modes that allow operations to be made directly with values in memory.

CISC architectures have the advantage of requiring a smaller number of assembly instructions to perform some task, which means programs usually use less memory compared to RISC architectures, since the later often uses several instructions to perform the same operations performed by a single instruction of a processor CISC. The greater complexity of the operations performed by the instructions in CISC architectures also means that the process of compiling a high level language to assembly requires less work from the compiler [2].

On the other hand, RISC processors have a simpler architecture, requiring less hardware resources, which translates to faster circuits. Although more instructions are required to perform the same task, RISC processors can achieve higher performance than CISC processors. This is due to the fact the operations performed by these instructions are simpler and have a constant execution time, which makes it easier to use pipelining. Load-store architectures can reduce memory accesses when several instructions are going to use the same operand, as once it is loaded from the memory, it will stay in a register until some instruction writes to that register.

Because of its simplicity over CISC processors, RISC processors are good objects of study in an introduction to computer architecture. However, a commercial processor capable of supporting an operating system such as Linux or Windows has to meet requirements that go well beyond what is desirable in a first approach, because these requirements translate into an increase in complexity that makes it difficult to understand the system as a whole and the basic principles that are intended to be studied.

The P4 is a processor that was initially proposed in the book “Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores” [1], along with P3, a CISC processor. The P3 is fully specified in the book and was already implemented in reconfigurable hardware, used to experiment in laboratory classes. The P4 can be seen as a RISC version of the P3, but unlike the P3, it is not completely specified in the book and had never been implemented in hardware before.

This paper describes the specification and development of a didactic system for P4. The didactic system consists of a development board with a reconfigurable device (FPGA) with several input and output peripherals that allows the user to interact with it. As part of this solution, a development environment was created with tools for the development of assembly programs for P4, which can be tested and debugged in a simulator, or can be sent to the board in order to test them in a real system.

This solution results in a set of tools that make it possible to study in depth a processor with a RISC architecture, since its implementation at register-transfer level, up to the level of its programming in assembly, thus making the bridge between hardware and software.

2. Development Platforms

This work encompasses the specification and implementation of the P4 in hardware, and also the development environment for the P4, which includes an assembly editor, an assembler, debugger, simulator and a tool to interface with the FPGA.

The P4 was implemented in reconfigurable hardware, and the development environment was created on a web platform.

2.1. Hardware Implementation

The platform chosen for the hardware realization of P4 was the Terasic DE10-Lite board based on an Intel FPGA MAX 10 [3], shown in Figure 2.

This board was chosen because it has an affordable cost to equip the laboratory, includes several peripherals useful for a didactic system, and also has a connection for Arduino Uno compatible expansion boards, which can be used to add more peripherals. Additionally, the FPGA technology allows other pedagogical uses, such as the realization of logic circuits designed by the students. This means the board can be used in other courses be-

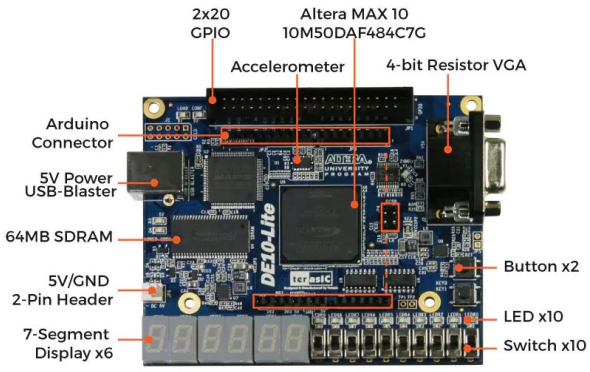

Figure 2: Photograph of the upper side of the Terasic DE10-Lite board.

sides Computer Architecture, such as Digital Systems. The main features of this board are:

- FPGA Altera MAX 10 10M50DAF484C7G.
  - 50,000 programmable logic elements.
  - 1638 Kb of memory organized in 9 kb blocks (M9K).
- 10 red LEDs.
- 10 switches.
- 2 push buttons.
- 6 7-segment hexadecimal displays.
- 64 MB SDRAM, x16 bits data bus.
- 4-bit VGA output.
- 3-Axis Accelerometer.
- Powered and reprogrammed via USB.

To provide more peripherals, an expansion board, shown in Figure 3, was added to this board, providing an additional six push buttons and a 2-line 16-character alphanumeric LCD.

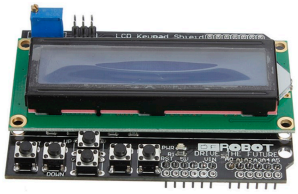

Figure 3: Photograph of the upper side of the expansion board.

A prototype for a second expansion board with a USB to allow the connection of a PS/2 compatible keyboard to the board was also developed.

One advantage of this FPGA is the ease of specifying the hardware and programming it using the Quartus Prime Lite software provided by Intel. Many of the components of P4 were specified using the Schematic Design tool available in Quartus. This tool provides a graphical environment where you can specify logic circuits using block diagrams. These diagrams are created using logic gates, registers and other components, which are then interconnected with lines and buses.

It is easy to use block diagrams to create simple circuits that depend only on the interconnection of several blocks, thus creating a hierarchical structure that is easy to visualize. On the other hand, for circuits that implement more complex sequential logic, the VHDL language [4], a hardware description language, was used because it allows specifying these circuits with a higher level of abstraction and a greater focus on their functionality.

2.2. Development Environment

We have chosen to create the tools for the development environment using web technologies: HTML5, JavaScript and CSS. These are technologies that are constantly evolving, with great availability of documentation and libraries due to their wide use in web and, increasingly, in other platforms such as mobile and desktop applications.

With these technologies it is possible to implement graphical interfaces and functionalities in a simpler and faster way than with other technologies traditionally used in the development of native applications. In addition, this technology is not platform specific, which means the development environment will work on several platforms.

For the user, the main advantage is being able to use the tools by simply accessing a web page using an Internet browser, with no need to install additional software on the user's system.

The development environment can also be installed in the user's system so the tools can be used even without an Internet connection. This offline version was developed with the NW.js [5] framework. The distributions for Linux, Windows and macOS platforms are automatically created using the nwjs-builder-phoenix [6] tool.

The offline version also includes a command-line assembler for P4.

3. P4 Specification

The P4 is a 16-bit RISC processor. Its main characteristics are:

- A set of 38 instructions.
- There are three addressing modes.
- Fixed size instructions (16 bits).
- There are three instruction formats.
- Each instruction uses at most two addressing modes.
- It has a set of 8 general purpose registers.
- It has a 4-stage pipeline.
- Supports interrupts.

3.1. Instruction Set

The P4 has instructions for performing arithmetic, logic, shift, control, transfer, and generic operations. The instruction set of P4 is listed in Table 1 with the instructions grouped by the type of operation they execute.

Table 1: P4 Instruction Set.

Arithmetic	Logic	Shift	Control	Transfer	Generic
NEG	COM	SHR	BR	MOV	NOP
INC	AND	SHL	BR.*cond*	LOAD	ENI
DEC	OR	SHRA	JMP	STOR	DSI
ADD	XOR	SHLA	JMP.*cond*	MVI	STC
ADDC	TEST	ROR	JAL	MVIH	CLC
SUB		ROL	JAL.*cond*	MVIL	CMC
SUBB		RORC	RTI		
CMP		ROLC	INT		

3.2. Pipeline

The P4 is structurally organized in a 4-stage pipeline:

- 1st stage: instruction fetch.
- 2nd stage: instruction decode and register fetch.
- 3rd stage: execute and memory access.
- 4th stage: register write back.

In more detail, in the *first stage* there is the Program Counter (PC) register, which stores the address of the instruction to be executed. The instruction to be executed is fetched from the program memory using the value in the PC register as the address to be fetched.

In the *second stage*, the instruction decoder evaluates the instruction and generates several control signals, including the signals to fetch the operands from the register file.

In the *third stage* the operation for the instruction is performed by the Arithmetic Logic Unit (ALU), unless it is a `LOAD` or `STOR` instruction which will load or write to the data memory or to a peripheral.

In the *fourth stage* the result is written to the register file. Other than the register file, there is only a multiplexer in this stage to select the result source to be written. It can be either a result from the ALU, memory or peripheral, or the PC.

3.3. Registers

The P4 has 8 general purpose registers, R0 to R7, able to store 16-bit values, the PC register, and the state register: the R0 register is always zero and cannot be changed; the PC has the address of the instruction to be executed; and the state register stores a set of flags set by the arithmetic, logic and shift operations. All registers are set to zero when the processor is reset.

3.4. State Flags

The following flags are used in the P4 processor:

- O: overflow. Indicates whether the result is not valid when interpreted as a two's complement value because it exceeds the capacity of the register where it was stored.
- N: negative. Indicates the result is a negative value when interpreted as a two's complement, i.e., the most significant bit is 1.
- C: carry. In arithmetic operations it indicates the result is not valid when interpreted as an unsigned value because it exceeds the capacity of the register where it was stored. On shift operations it corresponds to the bit that was shifted out. This flag can also be set by the instructions `STC`, `CLC` and `CMC`.
- Z: zero. It indicates the result was zero.
- E: enable interrupts. The interrupt handler is only called when this flag is activated. This flag can be set by the instructions `ENI` and `DSI`.

3.5. Memory

The P4 uses two separate memories for program and data. Each memory can store 32768 words with a length of 16 bits each. Addresses higher than 7FFFh are mapped to 15 bit addresses ignoring the most significant bit. For instance, the address 8000h is mapped to the same memory position as 0000h.

3.6. Addressing Modes

The P4 uses three addressing modes:

- *Register direct*: It is the only addressing mode used in arithmetic, logic and shift operations. The operands are contained in registers. E.g.: `INC R1`; `ADD R1, R2, R3`.
- *Immediate*: This mode is used by the instructions used to store literals in registers. It is also used in branch instructions and in the `INT` instruction. The operand is the literal. E.g.: `BR 127`
- *Register indirect*: It is used by `LOAD` and `STOR` instructions. The operand is the content at the memory location pointed by the register. E.g.: `LOAD R1, M[R2]`.

3.7. Instruction Formats

All instructions are 16-bit wide and there are only 3 instruction formats: format I, used for arithmetic, logic and shift instructions; format II, used for control operations; and format III, used for other instructions. There are two sub-divisions for the formats II and III defining different uses for the 8 least significant bits of the instruction. The reduced number of instruction formats means a simpler instruction decoder circuit can be used.

The `NOP` instruction is encoded as an instruction with all bits set to zero. This overlaps the encoding of relative branches, however a condition with all bits set to zero is never fulfilled, therefore, the `NOP` instruction is encoded as a relative branch that is never taken.

3.8. Interrupts

Interrupts allow the processor to react to the events that require immediate attention. In P4 there are 9 possible interrupt sources: pressing one of the 7 push buttons, pressing a key in the keyboard and when the timer count ends. Anytime

there is an interrupt, the bit corresponding to the E flag, and the corresponding bit in the interrupt mask are tested. When both are 1, the interrupt handler is called. The corresponding handler must be in the address computed as 7F00h plus the interrupt vector shifted 4 bits to the left. This shift means each handler can use up to 16 contiguous memory positions without overlapping other handlers.

3.9. IO

Addresses between FF00H and FFFFh are reserved for peripherals which can be interfaced using ports mapped to these memory addresses.

4. Development Environment

Assembly programs can be created and assembled in a graphical interface available on a web platform, developed in JavaScript, HTML5 and CSS, which runs directly in an Internet browser entering the address `https://web.tecnico.ulisboa.pt/dinismadeira/p4/`. This solution is similar to an already existing solution for the P3, the P3JS Assembler and Simulator [7].

There is also a native application for Windows, Linux and macOS, that can be downloaded from the online version.

4.1. Assembler

The assembler generates machine code from an assembly program. In order to run that program, the machine code must be written to the P4's program memory. In addition, the assembly programs can also initialize data memory locations with values specified in the program.

The contents to be written to these memories are specified by files in the format Memory Initialization File (MIF) [8]. Specifically, the assembler generates a prog.mif file that specifies the contents of program memory and a data.mif file that specifies the contents of the data memory. These two files are compressed into a single ZIP file with a p4z extension. There is also a command line assembler using the Node.js [9] environment.

4.2. Simulator

The simulator facilitates the development of programs for P4 by providing a way to run and debug programs in a graphical interface that represents the real system. The following features are supported:

- View the contents of the data memory, the value of the registers, the Program Counter, and the state flags.
- View the contents of program memory and the disassembled instructions.
- Set breakpoints and run the program step-by-step.
- Set the clock frequency for the simulation.
- View the total number of simulated clock cycles.
- View the assembly statement being executed.
- Emulation of the peripherals in an interactive graphical interface similar to the real system that allows the user to define and observe the inputs and outputs.

4.3. FPGA Interface

It is possible to interact with the FPGA using the application as long as the Intel Quartus software is installed on the system. This feature is not available in the online version.

Using this interface the user can run an assembled program on the P4, which is done by updating the contents in the program and data memory. It is also possible to change the terminal font and the colour palette, updating the respective memories.

The P4 processor has a control memory whose content can be changed using the FPGA interface in order to control the behaviour of the processor, such as stopping the execution of a program, execute step-by-step, restart the processor or execute any instruction.

4.4. Additional Tools

These tools are useful to easily create content to be displayed in the terminal, and further customize it with a custom font.

4.4.1 Terminal Text Editor

This tool allows the editing of the contents of the text window using a graphical interface, including changing the colour of the text and the background. It generates an assembly program that prints this content in the terminal by encoding it in memory, which uses control sequences to control the cursor position and set the foreground and background colour.

4.4.2 Terminal Font Editor

The font used in the terminal is stored in the font memory. This tool allows to customize the default font by drawing the pixels of each character. It then generates a MIF file with the custom font that can be loaded to the P4.

4.5. Settings

The language of the application can be set to English or Portuguese. All settings can be exported to a file or imported from a previously exported file.

5. P4 Implementation

The hardware implementation was done using a bottom-up approach, that is, the most basic components were first created and validated and then used as black boxes as part of other more complex components.

In this chapter we present in more detail the implementation of some components of P4, with spe-

cial emphasis on the development process, indicating some challenges and solutions found.

5.1. ALU

The Arithmetic Logic Unit (ALU) is the circuit that performs arithmetic and bitwise operations. It has four main components: the arithmetic unit, the logic unit, the shift unit, and the octet selection unit. All work in parallel, but an operation code is used to control a multiplexer which selects the output of the unit that performs the requested operation.

5.1.1 Arithmetic Unit

The main component of the arithmetic unit is a 16-bit adder, which has a carry-in input, a carry-out output, and also an overflow output. The carry-out output is the carry-out of the 1-bit adder that computes the most significant bit of the result, while the overflow output is computed as a XOR of the carry-outs of the 1-bit adders that compute the two most significant bits of the result.

The arithmetic unit also performs subtractions. Binary subtractions can be computed by performing the two's complement on the second operand. Thus, subtractions are accomplished by XORing each bit of the second operand with 1 and activating the carry-in of the 16-bit adder.

For `INC` and `DEC` instructions, the second operand is replaced by 1 or -1 by a multiplexer.

Circuits like this are simple enough to be tested directly on the board. The switches allow us to set the inputs and the outputs can be seen in the LEDs.

5.1.2 Logic Unit

The logic unit performs four operations: `XOR`, `OR`, `AND` and `NOT`, which correspond to the exclusive disjunction, disjunction, conjunction, and negation.

Each operation is performed by a circuit made of 16 logic gates that compute each bit of the result.

5.1.3 Shift Unit

The shift unit is made of multiplexers that compute the bit n of output by selecting from the input the bit $n-1$ when shifting left, or $n+1$, when shifting right. For the most and least significant bits there is some additional logic: when performing rotations, the inserted bit must come from the shifted out bit; in operations with carry, the inserted bit corresponds to the carry bit from the state register; in right arithmetic shifts the most significant bit remains unchanged.

5.1.4 Octet Selection Unit

We have chosen to implement the `MVI`, `MVIH` and `MVIL` instructions in the ALU. These operations load an 8-bit constant into the 8 most significant or least significant bits of a register, which is performed by this circuit.

This circuit also implements the `MOV` instruction by outputting the input unchanged, thus copying the value from one register to another.

5.1.5 State Control

Whenever an operation is performed by the ALU, the state register must be updated with the flags generated by the specific operation. The state control is a circuit that receives the current value in the state register and updates the bits for the flags generated by the performed operation. The updated value is then stored in the state register.

This circuit also updates the carry bit with the proper value for the `CLC`, `STC` and `CMC` instructions.

5.2. Register File

Using the bottom-up approach, the first components created for the register file were 16-bit registers made of D-type flip-flops. They were then used to create a file of seven 16-bit registers. Each register is connected to two multiplexers, which allow the register file to output two registers at once. Writing to the register file is possible with a decoder that activates the write enable of a single register according to a selection signal.

This circuit has also been tested directly on the board, using a push button as a clock signal, the switches to set the inputs, and the LEDs to show the output.

5.3. State Register

The state register is a 4-bit register that stores the signal, transport, excess, and zero flags. Its value is updated every clock cycle with the value provided by the state control circuit.

There is an additional register to backup the contents of the state register whenever an interrupt is handled. When the interrupt handler returns, the state register contents is recovered from this additional register.

5.4. Program Counter

The Program Counter is a register that increments one unit each clock cycle. It is also possible to load it with a given value.

5.5. Branch Control

The branch control unit evaluates when a branch should occur. Every condition is evaluated and used as an input to a multiplexer, which uses the condition code in the instruction to select the branch condition that should be used to evaluate the branch. The result is then XORed with the least significant bit of the condition code, thus, whenever this bit is 1, meaning it is a negative condition, the result is negated by the XOR operation. When the result is 1, the branch should occur and the program counter is loaded with the branch target address.

5.5.1 Delay Slot

Because the branch evaluation happens in the second stage of the pipeline, the instruction following the branch instruction enters the pipeline before the branch has been evaluated.

To prevent the performance hit of flushing every instruction that entered the pipeline after a branch that was taken, some processors use these instructions as delay slots, which are instructions following a branch that are always executed despite the branch being taken or not.

In P4, the instruction following a branch instruction is a delay slot. This simplifies the circuit as there is no need to flush the pipeline when a branch is taken, and has educational value for the concept of branch delay slots.

5.6. Instruction Decoder

The instruction decoder is a circuit that generates several control signals according to the instruction that is being executed.

This circuit was first created with support for some instructions only, allowing us to test the P4 with basic functionality. Then, new instructions were added and tested before adding more instructions.

Each control signal is computed by performing logical operations on several bits of the instruction, and, in some cases, using multiplexers also controlled by the results of these operations.

Even though these are not complex operations, using block diagrams makes it difficult to understand how the signals are being generated, which also makes it hard to debug and add new instructions. For this reason, a VHDL description of the circuit was made to replace the block diagram. This way signals are described with simple statements in a high level language.

5.7. Data Forwarding

Data forwarding is a technique used to improve the performance of the processor by avoiding pipeline stalls caused by data hazards. This is, when an instruction depends on the results of some previous instruction and must wait for its execution to complete.

The data forwarding unit detects data hazards by storing information about the registers where the results of the instructions in the 3rd and 4th stage of the pipeline are going to be stored, and comparing them with the registers that are going to be used as operands by the instruction in the 2nd stage. When they match, the result in the later stage of the pipeline is forwarded to the 2nd stage to be used as the operand, instead of using the current value of the register that does not yet reflect the result of the instructions in the later stages.

5.8. IO

Memory operations, IO operations and interrupt generation and acknowledgement, are controlled by the IO component.

When writing to an IO reserved address, a decoder is used to activate a writing flag in the corresponding peripheral for the port corresponding to that address. The outputs of the several peripherals are used as input signals in a multiplexer controlled by the address.

5.9. Terminal

The terminal receives input from a PS/2 keyboard and outputs to a VGA display.

5.9.1 Keyboard

Since the P4 board does not have a keyboard input, we developed a prototype of an expansion board with a USB port allowing the connection of keyboards with PS/2 compatibility mode or, alternatively, any PS/2 keyboard using a passive USB to PS/2 adapter to make the connection.

The USB port is connected to 4 pins of the GPIO input in the board. For correct operation, both the clock and data lines were connected to a 2kΩ pull-up resistor. In addition, 120Ω resistors were also used to interface the keyboard (5V) with the board (3.3V).

5.9.2 VGA Display Controller

The VGA Display Controller circuit is based on the P3's controller written in VHDL. There were made some changes in order to support higher resolution and colour output. The controller not only generates the VGA signals to display a text window, but also supports the control commands to update the contents in the text window.

The resolution of the P4 text window is 1920x1080 pixels using a 24-pixel font, an upgrade from the 640x480 pixels used in the P3 which used an 8-pixel font. The text window in the P4 is 80 characters wide with 45 lines, whereas the text window in the P3 has 24 lines and is also 80 characters wide. The P4 text window is also available in the 1280x960 resolution for compatibility reasons, which can be set using the control port for the terminal.

It is possible to set the background and foreground colour for each character printed to the text window using a configurable 256-colour 12-bit palette. The 256-character font used in the text window is also configurable.

5.10. Interrupts

Whenever an interrupt is emitted, the bit in the pending interrupts register corresponding to the interrupt source is set to 1. The interrupt encoder will use the pending interrupts register and the interrupt mask register to compute the next interrupt

to be handled, if any, and activate an interrupt flag telling the interrupt unit to handle the interrupt. On the other hand, when the interrupt handler returns, the interrupt unit activates the interrupt acknowledgement (IAK) flag and the corresponding bit in the pending interrupts register is set to 0.

5.10.1 Interrupt Mask

The interrupt mask is a 16-bit register where each bit corresponds to an interrupt vector. Writing to the interrupt mask port sets the contents of this register.

5.10.2 Interrupt Encoder

The interrupt encoder is a priority encoder that determines the next interrupt to be handled, making the logical conjunction between the interrupt mask and the pending interrupt register. If the resulting value is non-zero, there are pending interrupts, and the next to be handled is the one corresponding to the least significant non-zero bit, i.e., the interrupt whose vector is the lowest.

5.11. Push Buttons

The P4 board has 8 push buttons. There is a reset button that sets the Program Counter and all registers to zero. The remaining buttons generate an interrupt when pressed.

5.11.1 Expansion Board Buttons

With the exception of the reset button, the buttons on the expansion board are connected to a single analogue pin. The Analogue-to-Digital Converter (ADC) in the FPGA generates a 12-bit value from this input.

Whenever a button is individually pressed, the input value is within a certain range. These ranges were mapped for every button in order to create a circuit able to tell which button is being pressed, if any.

5.11.2 Debouncing

Due to the mechanical nature of the buttons, when a button is pressed, the input changes several times for a short period before stabilizing [10], which would make programs to behave as if the button had been pressed several times by the user.

To avoid this situation a debouncing filter is applied. This filter discards momentary variations of the input. To do this, a 19-bit counter driven by a 50 MHz clock is reset whenever the input changes, and the output is only updated when the counter reaches zero, which means the input was stable for at least 10 ms.

5.12. LEDs

The LEDs are driven by a 10-bit register. Each bit is connected to an individual LED. Writing to the LEDs port updates the contents of this register.

5.13. Timer

The timer is a circuit described in VHDL with two counters. There is a port to set the value for the main counter, which is the timer duration in deciseconds. When this timer reaches zero, an interrupt is issued and the counting is disabled. This timer is decremented every time the other counter reaches zero, which happens every 100 ms. There is also another port to enable or disable the counting.

5.14. Alphanumeric LCD

There is a circuit for the low-level control of the LCD responsible for initializing the LCD, serializing the commands and ensuring the timing requirements of the LCD are met. This circuit is controlled by another circuit that is able to understand the values written to the LCD ports and translates them to the proper commands for the LCD controller.

Since the LCD operates at a slower speed than the P4, it can't accept new commands for several clock cycles after receiving a command or when it is initializing. For this reason, writes to the LCD ports are queued in a 64-position FIFO queue. The commands in the queue are then consumed as the LCD is available to receive more commands.

This allows the programmer to make several consecutive writes to the LCD ports despite the time the LCD controller may take between each command, as long as they don't fill the queue.

5.15. 7-Segment Hexadecimal Display

Each display is connected to a circuit, created in block diagrams using AND, OR and NOT gates, that translates a 4-bit input into a 7-bit output that controls each individual segment to create a pattern for a hexadecimal digit that represents the 4-bit input.

Writing to the 7-segment display port updates the content of a 4-bit register used as the input of the former circuit.

5.16. Accelerometer

The accelerometer in the board uses an SPI interface [11], a serial communication interface used for communication between microprocessors and small peripherals. This is implemented using components described in Verilog downloaded from the Intel Design Store [12] with some changes in order to read all 3 axis instead of just the X axis as it originally does. These values are stored in registers that can be read using the port for the corresponding axis.

6. Evaluation

The system was validated and evaluated in terms of used resources, performance and operation. Some sample assembly programs were developed

demonstrating the operation of the system and the peripherals.

6.1. Functional Simulation

Due to the reconfigurable property of the hardware used, many circuits were tested on the board using the board's switches to define the possible input values for the circuits and observe the results through the LEDs and the 7-segment hexadecimal display. When the processor was already capable of running programs, it was tested by running small programs specifically created to test the processor and the peripherals.

However, this form of testing is limited to the outputs of the board and does not give full view of the internal state of the system. In order to observe the signals inside the circuit, the system was simulated using the Simulation Waveform Editor tool in Quartus.

This tool simulates the system and generates a diagram with the evolution of the signals for several clock cycles, allowing for a more complete testing and validation of the hardware implementation of P4.

6.2. Portability

In order to assess the portability of the P4 to other Intel FPGA boards, it was ported to the Terasic DE0-CV [13] board based on an Altera Cyclone V FPGA. In order to do this, the design configuration in the Quartus project, as well as the pin mapping, were updated for this device. Since the new pin mapping uses the same names for the pins that have the same function, only a few pins had to be updated within the project. It was however necessary to remove the ADC circuit, since this board does not have it. In addition, the board also has no accelerometer or connection to the expansion board used in the P4 system, so these peripherals would not be available on a system using this board.

6.3. Max. Frequency and Critical Path

The clock frequency of P4 is 80 MHz. The TimeQuest Timing Analisys tool estimates a maximum operating frequency of 83.08 MHz in the Slow 1200mV 85C model with a critical path between the data memory and the PC, through three multiplexers, which is the data forwarding circuit that makes possible for a `LOAD` instruction to be followed by `JMP` or `JAL` using the same register.

In order to reduce the critical path a register was added to the `Ld` signal that controls the loading of a new value into the PC. With this change, the estimated maximum operating frequency is 102.97 MHz, about 24% faster. On the other hand, this solution adds a delay to jumps and also makes the operation of the processor more complex. For this reason it was decided discard this change and keep the previous solution.

6.4. Performance Evaluation

The processor performance can be calculated as the total number of instructions executed to perform a given task, multiplied by the average number of clock cycles per instruction (CPI) and the inverse of the clock frequency.

Due to the pipelining mechanism used in P4 and its simpler architecture, a considerable increase in performance over P3 is expected. Because it is a RISC processor, P4 should have a CPI close to 1.

In Table 2 there is an assembly program to add the first 64 values in memory and write the result into memory, for both P3 and P4.

Table 2: Assembly program to add the first 64 values in memory and write the result into memory.

P3			P4		
	MOV	R2, 0040h		MVI	R2, 0040h
	MOV	R3, 0000h		MVI	R3, 0000h
L1:	ADD	R3, M[R2]	L1:	LOAD	R4, M[R2]
	DEC	R2		DEC	R2
	BR.NZ	L1		BR.NZ	L1
				ADD	R3, R3, R4
L2:	MOV	M[R1], R3	L2:	STOR	M[R1], R3

In P3, the program executes a total of 195 instructions, which take 2070 clock cycles, while in P4 it executes a total of 259 instructions, which take 259 clock cycles, not considering the time it takes to fill and empty the pipeline. This gives a CPI of approximately 10.6 for P3 and 1 for P4.

Since the operating frequency of the P4 is 80 MHz, the program takes 3.24 μs to execute, which corresponds to 80 MIPS. In the P3, at a frequency of 6.25 MHz, is takes 331.2 μs, corresponding to about 0.6 MIPS.

Although the P4 has to execute more instructions to accomplish the same task, its performance can be 100 times higher than the P3.

6.5. Demonstration

On the web platform there are some programs that demonstrate the operation of P4 and its peripherals. For example, the Keyboard Interrupts demo activates the terminal interrupts, and whenever a key is pressed, it will read the value for that key and write it back to the terminal, the alphanumeric LCD, the 7-segment hexadecimal display and the LEDs. For instance, if the user presses the `P` and `4` keys, it will display the `P4` text in the alphanumeric LCD, the `34h` value in the 7-segment display, and the `0000110100b` pattern in the LEDs, which is the ASCII value of the last key pressed, as shown in Figure 4.

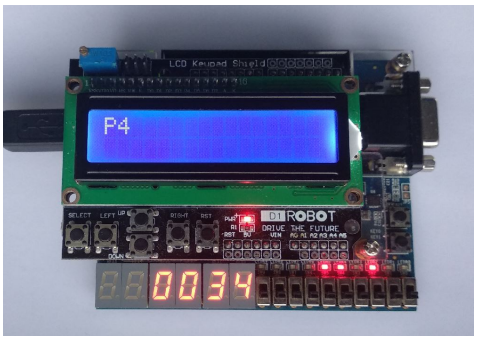

Figure 4: Keyboard Interrupts demonstration program running in the P4.

7. Conclusion & Future Work

The P4, proposed in the book, is almost fully specified. For its implementation in hardware, it was necessary to define some architectural questions that were not considered in the book. Some changes have also been made because they allow for simpler hardware realization. It was also implemented an interrupt system that was not planed for this processor in the book.

It was implemented in reconfigurable hardware, which was described at register-transfer level, using mainly visual block diagrams interface and also using the VHDL hardware description language.

In addition to the hardware implementation, it was also created a development environment based on a web platform, which includes tools to create, assemble and debug assembly programs, as well as to simulate them with a virtual graphical interface representing the real system with its peripherals.

This solution makes possible to study a RISC processor from its architecture and implementation at register-transfer level up to a higher level with programs developed in assembly being executed in a real system.

Being a non-commercial processor, created only for the purpose of being implemented in a didactic system, together with the fact that it is a RISC processor, it has a complexity level sufficiently accessible for its study in the Introduction to Computer Architecture course, unlike other existing solutions with a higher level of complexity that makes it more difficult to understand the basic operation of a processor.

This system comes to innovate the teaching of Computer Architecture, presenting a RISC architecture as an alternative to P3, which has a CISC architecture. Besides that, P3 is implemented in older hardware that becomes increasingly difficult to maintain in the laboratory as its interfaces are becoming obsolete. In the future, P3 can be adapted to the FPGA board used for P4, which would render a didactic system for a CISC and RISC processor using the same hardware solution.

The hardware implementation and software tools are available in a GitHub repository, available at `https://github.com/dinismadeira/p4`, which will facilitate future changes and independent contributions.

References

[1] Guilherme Arroz, José Monteiro e Arlindo Oliveira: Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores. 3rd Edition. IST Press (dezembro 2014)

[2] RISC vs CISC, `https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/`. Accessed: 2019/04/15.

[3] Intel FPGAs, `https://www.altera.com/`. Accessed: 2018/05/29.

[4] What is VHDL?, `https://www.doulos.com/knowhow/vhdl_designers_guide/what_is_vhdl/`. Accessed: 2019/04/30.

[5] NW.js, `https://nwjs.io/`. Accessed: 2019/03/08.

[6] nwjs-builder-phoenix, `https://github.com/evshiron/nwjs-builder-phoenix`. Accessed: 2019/03/08.

[7] P3JS Assembler and Simulator, `https://p3js.goncalomb.com/`. Accessed: 2018/05/30.

[8] Memory Initialization File (.mif), `https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/reference/glossary/def_mif.htm`. Accessed: 2019/04/07.

[9] Node.js, `https://nodejs.org/`. Accessed: 2019/02/02.

[10] Switch Debouncing, `https://electrosome.com/switch-debouncing/`. Accessed: 2019/04/27.

[11] Serial Peripheral Interface (SPI), `https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all`. Accessed: 2019/04/10.

[12] GSensor Example in Design Store for Intel FPGAs, `https://fpgacloud.intel.com/devstore/platform/16.1.0/Standard/gsensor-max10-de10-lite/`. Accessed: 2019/01/18.

[13] Terasic DE0-CV, `http://de0-cv.terasic.com`. Accessed: 2019/03/11.