MIPS 1 in VHDL

HDL Lab - SS 2015

Bahri Enis Demirtel, Carlos Minamisava Faria, Lukas Jäger, Patrick Appenheimer

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik
Fachgebiet Integrated Electronic Systems Lab

Contents

1 Introduction

In the HDL Lab is a practical exercise of a hdl implementation. This semester the task is the implementation of a MIPS I microcontroller in vhdl. A requirement to this laboratory is the lecture HDL: Verilog and VHDL by Prof. Dr.-Ing. Klaus Hofmann.
MIPS is an acronym for Microprocessor without interlocked pipline stages. The MIPS instruction set is a reduced instruction set computer (RISC). There are available references for 32 and 64-bit with many revisions.
This structure was developed in the 80s with the intent to take fully advantage of pipelines. Nowadays this instruction set and structure is often used as an hdl first project. Commercially it is used embedded systems such as Windows CE devices, routers, residential gateways and video consoles such as Nintendo 64, Sony Playstation, Playstation 2 and Playstation Portable.

1.1 Task

The objective is to design, implement, synthesise and test a MIPS-I specified processor core on FPGA. The hardware description langugage is VHDL and the target technology is a Virtex5 from Xilinx. The synthesised microcontroller must be able to run at 50 MHz with a desirable frequency of 200 MHz. The microcontroller must use a pipeline of a minimum of 2 and maximum of 6 stages. The following subcomponents are mandatory: ALU, datapath and controlpath.
A counter test program shall run on the synthesised microcontroller, outputing the counter value to eight LEDs on the FPGA board.

2 Design

This chapter describes the design of a MIPS 1 microcontroller. A microcontroller consists mostly of a processor core, memory and programmable inputs/outputs peripherals. This design implementation focus only on the processor core design.

This laboratory requires a microcontroller structure of at least a CPU containing a controlpath, datapath and an ALU. The created design uses this base with a 5-staged pipeline. The CPU interacts with two external memories and has also a clock and a reset input. All of which are considered external to this design. The CPU is divided in two base components: a control and a datapath block, as shown in Figure 2.1.

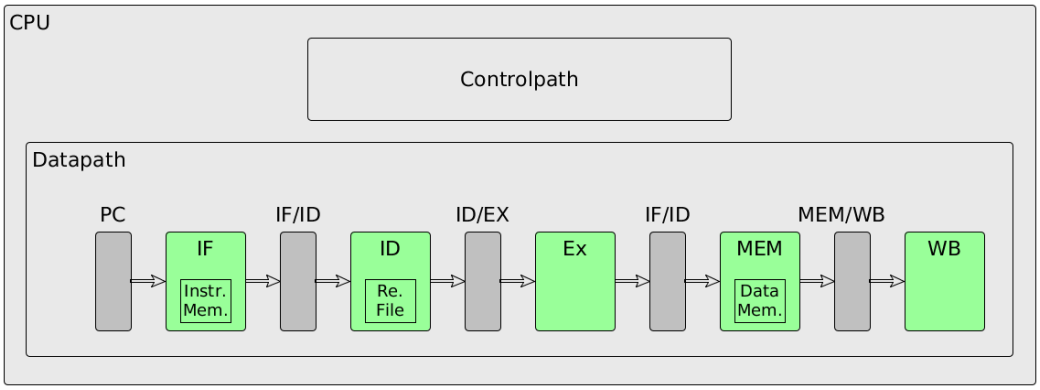

Figure 2.1: Datapath pipeline

In the datapath there is the pipeline made of five blocks: instruction fetch (IF), instruction decode (ID), execution (EX), memory stage (Me) and writeback (WB).

The following sections describe the central component ALU and the two base components: datapath and controlpath. The cpu components structure is shown in Figure 2.2:

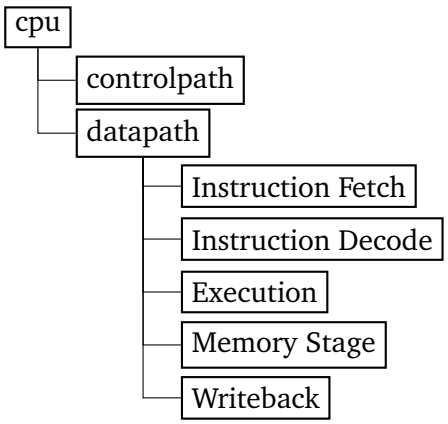

Figure 2.2: CPU component structure

2.1 ALU

2.2 Datapath

This section describes the datapath and its internal elements. The datapath is the component that connects the pipeline components within itself as well as with cpu inputs and outputs and the controlpath.

This MIPS implementation works with a 5 stage pipeline in order to achieve a fast clock. The datapath consists of instruction fetch, instruction decode, execution, memory stage and writeback. The datapath controls the information flow from one pipeline stage to the next with registers. These writing process occur on the positive edge of the clock when the pipeline stage input from the controlpath. That is, the registers forwards information synchronously. The datapath forwards the controlpath signals asynchronously, contrary to the pipeline to pipeline signals.

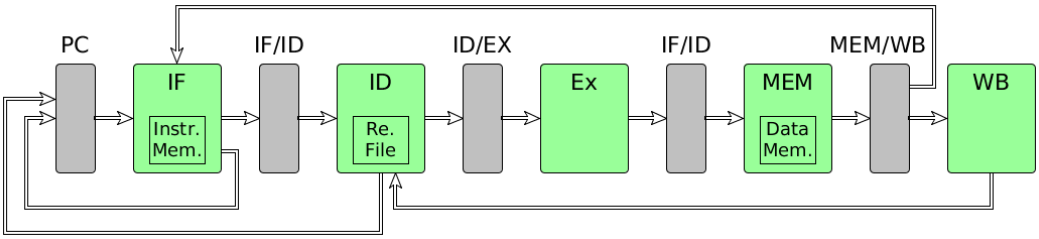

Figure 2.3: Datapath pipeline

The program counter (PC) is programmed into the datapath. Its function is the store the current program address, which is mostly counted up. It has a multiplexer controlled by the controlpath to choose the input. The two possible inputs are to count up (PC+4) from instruction fetch and the jump or branch from instruction decode. The controlpath chooses always the instruction decode input in the case of jump or branch.

The following subsections describe the pipeline components as well as its functions and IOs.

2.2.1 Instruction fecth

This first block of the pipeline is the instruction fetch. The main task of this block is to fetch the next instruction and pass it further to the pipeline.

The program counter is a 32-bit input, which is directly outputted as instruction address to fetch an instruction. The instruction fetch inputs the program memory's instruction data, with the actual 32-bit instruction. This value is directly forwarded to the pipeline.

The program counter is also incremented by four, because the used memory is 8-bit long. This incremented value is given back to PC.

2.2.2 Instruction decode

The second block of the pipeline if the instruction decode. Its main tasks are to divide the instruction into its pieces, manage the register file and manage branches.

The main input is the instruction from the instruction fetch stage. This instruction is 32-bit long and can be of three types. These are shown in Table 2.1 [1].

The **opcode** indicates the operation or arithmetic family of operations. Opcode equals zero are the R-type operations. The field **funct** provides an specific operation. **rs**, **rt** and **rd** provide sources or destinations register addresses. **shamt** indicates the shift amount for shift operations. **immediate** carries a relative address or constant, which is zero or sign extended to 32-bits. **address** is an absolute address.

The main outputs are register A, register B, shift, immediate and IP. Other than IP, all outputs depend on the instruction decoding.

Table 2.1: MIPS instruction types

Type	format (bits)					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shat (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Register File

The register file is a set of 32 general purpose 32-bit registers. These have the advantage, comparing to the ram memory, that they can always be accessed within one clock cycle. The access to these registers is made with five bits, which allows multiple registers to be referenced per instruction. All loaded memory values are stored in a register for later use.

The registers are numbered from $0 through $31. There is also a convention for using these registers, which must be enforced by assembly language and follow Table 2.2 [2]:

Table 2.2: MIPS registers

Register Number	Conventional Name	Usage
$0	$zero	Hard-wired to 0
$1	$at	Reserved for pseudo-instructions
$2 -$3	$v0, $v1	Return values from functions
$4 - $7	$a0 - $a3	Arguments fo functions - not preserved by subprograms
$8 - $15	$t0 - $t7	Temporary data, not preserved by subprograms
$16 - $23	$s0 - $s7	Saved registers, preserved by subprograms
$24 - $25	$t8 - $t9	More temporary registers, nor preserved by subprograms
$26 - $27	$k0 - $k1	Reserved for kernel. Dot not use.
$28	$gp	Global Area Pointer (base of global data segment)
$29	$gp	Stack pointer
$30	$sp	Frame Pointer
$31	$ra	Return Address

This implementation of MIPS does not have a FPU. In case of FPUs another 32 32-bit register set is used.

The register file is writen on the clock's negative edge with writeback information. The register file require a 5-bit destination register address and the 32-bit word to be written into the register.

Branch Logic

The branch and jump instructions require just one clock between instruction fetch and the jump itself. Due to this constrain, there is the need of a branch logic inside the instruction decode part. The output of this operation is the next instruction address for PC.

On the jump command, PC will receive the jump value. In case of a branch, PC will receive either the branch value or PC+4, depending on the instruction decode decision. This behaviour allows for the controlpath always to activate the instruction decode input in cases of jumps and branches.

Forwarding

Often calculated or memory read values are used in the following instructions. Due to the pipeline, the values are not ready in the register file, causing a data hazard. In order to avoid this conflict a data forwarding system is integrated. The data forwarding provide separeted inputs for the 5-bit destination register address and the 32-bit word for the ALU, memory stage and writeback.

2.2.3 Execution

This stage of the pipeline takes care of the actual mathematical operations. It provides two main multiplexers, one for each value input of the ALU. The inputs of the first multiplexer are the zero padded shift input, the number four (32-bit) and the register A from instructino decode. The second multiplexer provides register B, the immediate value and IP as inputs.

Both multiplexers are controlled by the controlpath.

2.2.4 Memory Stage

The memory stage is the fourth block of the pipeline and has the main task of fetch or save in the memory.

For memory operations the execution stage outputs two 32-bit values: aluResult_in, which works as the memory address, and data_in, which is data to be written in the memory. On read operations, the data_to_cpu input delivers the 32-bit memory value.

This stage has one multiplexer choosing the pipeline stage output from aluResult_in or data_to_cpu.

2.2.5 Writeback

This writeback stage is the fifth and last stage of the pipeline. Its main task is just to hold the calculated values, as well as the values read from the memory so they can be written the register file.

2.3 Controlpath

3 Evaluation

The cpu evaluation is done with Modelsim from MentorGraphics. The simulations provide a timed analysis of the code. It provides information about the timing relations and allows for bug identification still in a simulation environment, without the need of a complete synthesis and programming of the FPGA. This is a powerful tool to speed up the development process evaluating the design in an early stage.

Individual testbenches test each separate cpu components on all hierarchical levels up to the complete cpu. All component's simulation passed, including the complete cpu with a simulated perfect memory. Furthermore the simulation with a simulated real memory passed. The tests prove the complete implementation up to real conflict cases of instruction and data access stalls.

For the implementation on the FPGA a hdllab code was prepared with the cpu, memory, UART and pll components as well as LEDs, clock and reset interface with the FPGA already integrated. The cpu passed this simulation with the counter program, outputting the counter value to the LEDs output.

The evaluation phase was successful, proving exhaustively the correct behaviour of each component separately and as a piece of the whole cpu. The functional test of a counter program serve also as a prove of concept.

4 Synthesis

5 Conclusion

In this laboratory a 32-bit MIPS I cpu with restricted instruction set is successfully designed in vhdl, implemented, synthesized and test a MIPS-I specified processor core on FPGA. The required components of ALU, datapath and controlpath are present. The implementation on the FPGA passed a functional test with the counter sample program, running at the desired speed of 50 MHz and blinking the LEDs with the 8-bit counter value.

The cpu design comprises a controlpath and a datapath with five pipeline stages. Each component was designed and simulated separately and together up to as a complete cpu. The simulations in Modelsim included test cases for a perfect memory and a real one with instruction and data stalls. The simulation of a functional test with the program counter passed outputting the counter value as an 8-bit LED array.

The synthesis is done with Xilinx ISE for the FPGA Virtex5. With configurations for the fastest clock, the synthesis reports a maximum running frequency of over 70 MHz. The clock configuration for this laboratory is set at 50 Mhz. The synthesized code passes the functional test with the counter program on the Virtex5. This counter uses the planned instruction set.

This work shows an implementation of a restricted MIPS instruction set. The expansion of this instructions is expected to lead to a full MIPS 32-bit compliant microcontroller.

Bibliography

[1] MIPS Technologies Inc. *MIPS32TM Architecture For Programmers Volume I: Introduction to the MIPS32TM Architecture*. 2001.

[2] Jason W. Bacon. Computer science 315 lecture notes, 2011.