ANEM - A Didactic 16 Bit Microcontroller

Geraldo José Travassos de Arruda Filho

José Rodrigues de Oliveira Neto

Prof. Dr. João Paulo Cerquinho Cajueiro

Universidade Federal de Pernambuco - UFPE

Abstract

This paper concerns the development and implementation of a didactic 16 bit Harvard microcontroller based on the MIPS architecture. We describe its instruction set and its architecture, emphasizing the pipelined implementation. Access to memory and peripheral is described. This microcontroller was succesfully implemented on an ALTERA FPGA, with an example software to demonstrate peripheral access and its interruption capabilities.

1 INTRODUCTION

Due to the advances that digital eletronics have been facing in the last decades, hardware description languages such as VHDL becomes more relevant every day. The use of CPLDs and FPGAs allows engineers to progress from concept to functional silicon very quickly. Furthermore, microcontrollers are now the key elements of many applications, while SSI and MSI components, that have been serving as building blocks for almost forty years, are now nearly obsolet.

In this way, virtually all digital elements in a project, including the microcontroller itself, can be integrated in a single FPGA, allowing considerable gain in performance, reliability, efficiency and size. The so-called *soft-core processors*, processors written entirely in a hardware description language, are now common and the number of cells in FPGAs has reached a point in which it's possible for multi-processor systems to be synthetized in a single device. Another advantage of a soft-core is its extreme flexibility, given that its parameters can be easily changed using simple device reprogramation[6].

Today there are many comercial soft-cores, being Altera®'s Nios II®[2] and Xilinx®'s Micro Blaze®[7] worth mentioning. However, these soft-cores do not allow a detailed knowledge of its implementation. Moreover, they are also intelectual property of its developers, being mandatory the use of their platforms for synthesis. They are also not interesting didactic-wise, mainly because the source-codes are not made available. This paper describes in detail the development of a simple soft-core, based on MIPS[5] architecture and with microcontroller-like features. It was named ANEM (portuguese acronym for *ANEM não é MIPS*, which translates to "ANEM is not MIPS") and it can serve the purpose of being a didactic tool for under-graduate courses on microprocessors, microcontrollers or programmable logic devices.

2 INSTRUCTION SET

In MIPS, instructions have three operands, requiring the address for three registers. Considering a set of sixteen registers, it would be needed four bits for each one's addressing. As Anem16 has instructions only 16 bits wide, it was chosen to use only two operands per instruction, where an operand is both input and output, like an 8086 [4].

There are five types of instructions in Anem16: Types R, S, W, L and J. Type R instructions are arithmetic instructions. Type S instructions are for shift and rotation operations, they include a SHAMT (SHift AMounT) field that indicates the number of shifts/rotations to be done.

	0 3	4 7	8 11	12 15
Type R	OPCODE	RA	RB	FUNC
Type S	OPCODE	RA	SHAMT	FUNC
Type W	OPCODE	RA	RB	OFST
Type L	OPCODE	RA	BYTE	
Type J	OPCODE	ADDRESS		

Figure 1: ANEM Instruction types.

Type W instructions include a four bit offset field instead of the func field. This offset is considered to be signed, and because of that values from -8 up to 7 are accepted. Type L instructions have only one operand and an immediate field that is 8 bit wide.

Type J (jump) instructions need a 12 bit field to tell the jump address. Jumps of this kind affect only the 12 less significant bits of PC (Program Counter).

The ANEM instruction set is shown on tables 1, 2, 3 and 4. In comparison to MIPS, different instructions are LIL (*Load Immediate Lower byte*) and LIU (*Load Immediate Lower byte*). Such instructions were added

to allow immediate 16 bits values to be loaded directly from instructions.

Table 1: Arithmetic Instructions

Instruction	Type	Opcode	Func
ADD	R	0000	0010
SUB	R	0000	0110
AND	R	0000	0000
OR	R	0000	0001
XOR	R	0000	1111
NOR	R	0000	1100
SLT	R	0000	0111

Table 2: Jump Instructions

Instruction	Type	Opcode
J	J	1000
JAL	J	1001
HAB	J	1111
JR	W	0111
BEQ	W	0110

Table 3: Shift Instructions

Instruction	Type	Opcode	Func
SHL	S	0001	0010
SHR	S	0001	0001
SAR	S	0001	0000
ROL	S	0001	1000
ROR	S	0001	0100

Table 4: Memory Instruction

Instruction	Type	Opcode
LW	W	0100
SW	W	0101
LIU	L	1100
LIL	L	1101

3 HARDWARE ORGANIZATION

ANEM was designed to be used as a microcontroller. One of its main features is the memory access system based on *Harvard Architecture*. This architecture consists on the use of physical separation between data and program memories, permitting simultaneous access to both of them, resulting in a better performance when compared to *Von Neumann Architecture*. In the present section, the major topics on the hardware organization used on ANEM project will be detailed.

4 PIPELINE

The ANEM instruction inherited from MIPS' a load of features adequate for pipeline. First of all, all instructions are the same width, what eases instruction fetch. The register fields are always in the same position, allowing to read the register file at the same time in which the type of instruction is determined. Another important feature is that memory operations only occur within load and store instructions and there are many (16) registers, so memory is seldom accessed. This allows memory address to be calculated in one pipeline stage and to complete the access in another. Still regarding the memory, data need to be aligned, always as 16 bit words, permiting the access to be always done within one cycle.

To prompt an increase in ANEM's performance, instruction execution was divided in five steps: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Write Black (WB). This scheme is similar to the one described for MIPS in [5]. This way, it is possible to execute up to five instructions simultaneously. The output of each stage is to be stored in a register for use in the following stage. For instance, signals generated in ID are stored in ID/EX register for use in the EX stage in the next clock cycle.

In the first stage of *datapath*, Instruction Fetch, an instruction is read from memory in the address pointed by PC (*Program Counter* register). In ID, the instruction is decoded: all the control signals needed for the next steps are generated and the register file is accessed.

In EX, ALU (Arithmetic and Logical Unit) operations, address decodification and jump operations (made to PC at the end of this cycle) occurs. In MEM, RAM read and write operations are done. Finally, in WB, results are stored back on the register file.

The technique known as *forwarding* or *bypassing*[5] that creates alternative paths for data along pipeline was widely used to solve hazards. There are no structural hazards since it is an Harvard Machine. In fact, this was the main reason that determined the adoption of this kind of memory archictecture, in oposition to what is used in MIPS, in which data and program memory share the same physical memory.

5 MEMORY AND PERIPHERALS

In memory, all words have 16 bits and need to be aligned. This allows the processor to address up to 128 KB (64 Kwords) to the program and another 128 KB for data. However, not all data memory range is available for general use because the peripherals are mapped as memory. The address range from FFD0 up to FFFF (in hexadecimal) are reserved to peripherals, physically located outside RAM memory.

There is a component in ANEM called *RamPerifController*, which is responsible for atending any data memory access request from the processor. This unity decodes the address received to determine if the request should be sent to RAM memory or to the *peripherals* entity, which centers all peripherals and the interruptions unity in the processor. When an address is located below the FFD0 (in hexa), the signals are sent to an external SRAM memory chip.

5.1 Program Memory

To simplify the design, the program memory in ANEM was implemented using blocks of internal RAM available in the FPGA itself. ANEM has a backchannel that permits the program memory content to be changed through the serial port. For this to happen, the programer must be enabled (*Prog* pin must be kept in high logical level), and a software must send the code already in binary format following a simple algorithm. A software was developed using *Processing* language [3] to execute the programming task. Source code and executable binaries for Linux, Mac and Windows are available.

6 EXCEPTIONS AND INTERRUPTIONS

External interruptions are called simple interruptions, while internal ones are called exceptions, as in the MIPS. In this implementation of the ANEM, an interruption or exception forces a jump to an interruption vector and saves the PC return address in register *$15*. A stack must be implemented in software in case of nested interruptions or sub-rotines.

Interruptions are executed as a new instruction inserted into the pipeline, while exceptions can be generated by ALU when executing an instruction of subtraction (*overflow* and *borrow*) and addition (*overflow* and *carry out*). Since the ALU is located in the EX stage while PC *Program Counter* is located in the IF stage, it became mandatory the creation of a special register to store the instruction address that generated the last exception. This register is located in EX and is called EPC (*Exception Program Counter*). This way, every time an arithmetic exception occurs (and is not masked), the address to be saved in the register *$15* is available in EPC (in opposition to other types of interruption, in which the address to be stored is in PC).

When an interruption occurs, the interruption unit disables the ocurrence of other interruption and sends a command to the pipeline control unit. This command determines if the interruption was generated by an exception or if it is external and contains the jump address to treat this interruption. Upon receiving these signals, the control unity (ID stage of pipeline) stalls the program execution (in a similar manner used to avoid control hazards) and inserts a JAL instruction in the datapath. The jump address is that given by the interruption unity and the link address is given by PC or EPC (in exceptions). The interruptions can only be re-asserted by software, using the *HAB* instruction.

7 EXAMPLE

A simple assembler was developed for programming the ANEM. As a practical example for design testing, the classic memory game *Genius* was assembled and programmed to be executed by ANEM. Using Altera®'s DE2 development board, the game was implemented using leds and push-buttons. A random sequence is generated by the microcontroller using the duration of time in which the player pushes the button. The sequence is displayed using the leds and the player must repeat it using the push-buttons. Additionally, data is sent to a computer using a RS232 [1] interface. A software developed in *Processing* can then be used as a graphical user interface.

With this simple example, it was possible to use interruptions and some peripherals: *timers* for delays and sequence generation and the UART for communication with a computer. The code is composed of 699 words of instructions, from 723 lines of code, using virtually all instructions available, and so accomplishing a quick validation of the microprocessor design. A software based stack for nested sub-routines using the *JAL* instruction was also implemented.

8 CONCLUSION

The microcontroller was developed entirely in VHDL and implemented in a DE2 development board donated by Altera®. It has a reduced instruction set of constant size. Peripherals are disposed in a modular way that permits easy addition or deletion of new elements using a simple mapping. A game of memory was programmed into the proccessor to show its funcionality. The project design aiming simplicity makes it suitable for didactic purposes.

The instruction set proved to be quite adequate to program a 723 line assembly program, with the exception of the reduced range of the BEQ instructions, which permits branches of -8 to +7, while in the MIPS the range is from -128 to 127.

References

[1] Electronic Industries Association. *EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange*. Telebyte Technology, 1985.

[2] Altera Coorporation. Nios ii embedded processor. Available online:

http://www.altera.com/products/ip/processors/nios2/ni2-index.html. Accessed February 2011.

[3] Ben Fry and Casey Reas. Processing language. Available online: http://processing.org/about/. Accessed February 2011.

[4] Randall Hyde. *The Art of Assembly Language*. No Starch Press, 2nd edition, 2003.

[5] David A. Patterson and John L. Hennessy. *Computer Organization And Design*. Morgan Kaufmann Elsevier, 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 4th edition, 2009.

[6] Franjo Plavec. Soft-core processor design. Master's thesis, University of Toronto, 2004.

[7] Xilinx. Microblaze soft processor. Available online: http://www.xilinx.com/tools/microblaze.htm. Accessed February 2011.