A Rudimentary Machine. Experiences in the Design of a Pedagogic Computer

Enric Pastor, Fermín Sánchez and Anna M. del Corral

Abstract—
This paper describes a pedagogic computer named "Máquina Rudimentaria". This computer has been designed to be used in a first course on logic design or computer architecture; orthogonality and simplicity have been the major goals. The description of the computer includes the definition of both the machine and the assembly language, the implementation of the datapath and the optimization of the control unit. A programming framework has been designed to allow the development of small programs in assembly language, their compilation and detailed simulation on the proposed architecture.

I. Introduction

The *Máquina Rudimentaria* (MR) [1] intends to be a simple pedagogic computer. The main objective pursued in the design is to guarantee the orthogonality and simplicity. The MR may be used as a tool to teach basic concepts about structure and architecture of computers in the first courses of engineering or computer science studies. Since the design of the MR only includes basic concepts about digital systems, it can also be used as a complex case study in VLSI design courses. In that sense, the MR allows a natural progression from the study of logic design to the basic concepts of computer architectures and further into VLSI design. The MR has been design by professors in the Dept. of Computer Architecture (UPC) by using the experience acquired in the design of a simpler computer [2].

In order to use a processor in an initial course on computer architecture, old commercial products like the VAX [3] or Motorola series should be disregarded because they include large instruction sets and complex addressing modes. Processors in the Intel 80x86 family [4] should be also disregarded because of their lack of orthogonality and the number of side concepts that must be introduced (even the old 8086 is too cumbersome). More recent RISC processors, like the SPARC family, still are beyond the required complexity. Existing pedagogic processors like the DLX [5] are typically well documented and allow to introduce advanced concepts on them. However, they are also still far too complex to be easily assimilated by students with very rudimentary knowledge about computer architecture.

II. The Architecture of the MR

A. Basic Structure

The MR is a classic von Neumann computer without any I/O facility (for the sake of simplicity). The memory is organized in 256 words of 16 bits, with separated ports for

Instruction	Operation
ADDI Rs, #immediate, Rt	Rt := Rs + immediate
SUBI Rs, #immediate, Rt	Rt := Rs − immediate
ADD Rs1, Rs2, Rt	Rt := Rs1 + Rs2
SUB Rs1, Rs2, Rt	Rt := Rs1 − Rs2
ASR Rs, Rt	Rt := Rs >> 1
AND Rs1, Rs2, Rt	Rt := Rs1 $\wedge$ Rs2
LOAD base_addr(Ri), Rt	Rt := M[base_addr + Ri]
STORE Rs, base_addr(Ri)	M[base_addr + Ri] := Rs

Instruction	Branch condition	
BR target_addr	unconditional	1
BEQ target_addr	equal	Z
BL target_addr	less	N
BLE target_addr	less or equal	$N \vee Z$
BNE target_addr	different	$\overline{Z}$
BGE target_addr	greater or equal	$\overline{N}$
BG target_addr	greater	$\overline{N \vee Z}$

Fig. 1. The assembly language for the MR.

reading and writing, and a read/write control signal. The CPU is clearly divided into a datapath and a centralized Control Unit, described in sections III and IV respectively.

The instructions of the MR work on integer numbers of 16 bits using two's complement encoding. Addresses have 8 bits matching with the size of the memory. The CPU is able to perform basic integer operations on data stored in the 8 registers available in a Register File. Register R0 is dedicated to store the constant 0. Additionally, there exist two condition *flags* for the sign (N) and for zero (Z).

B. The Assembly Language of the MR

The instructions are of a fixed length of 16 bits (the word of the computer). The operation code is always stored in the two more significant bits, describing four types of instructions: arithmetic-logical instructions, data transfer instructions (two codes) and branch instructions. The MR offers four basic addressing modes: *register* mode, *immediate* mode, *relative* mode (base address + offset) and *absolute address* mode.

The arithmetic-logical instructions can use both the register and immediate addressing modes. When using the register mode the register file is accessed. When using the immediate mode a 5 bits constant coded in the instruction in two's complement is accessed. All arithmetic-logical instructions store their result into the register file (Rt) and activate the condition flags N and Z. If the instruction writes the result into the register R0 (constantly set to 0) no result is stored, but the condition flags are updated.

According to the source operands, two types of arithmetic-logical instructions exist: the first type reads all its source operands from the register file; the second reads

Authors are with the Department of Computer Architecture Universitat Politècnica de Catalunya, 08034 Barcelona, Spain. E-mail: enric, fermin, anna@ac.upc.es.

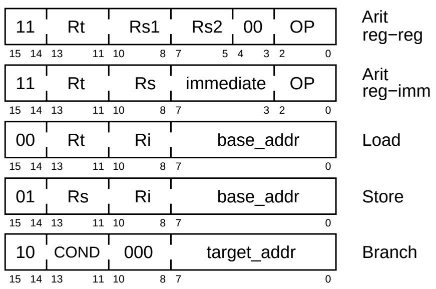


Fig. 2. Machine language format for the MR.

one operand from the register file and the second one from the instruction itself (immediate mode). The MR offers six arithmetic-logical instructions, as shown in Figure 1.

There exists two data transfer instructions: load and store (see Figure 1). These instructions use the register and relative addressing modes. The target memory address is computed by adding the contents of the eight less significant bits of an index register (Ri), the offset, to an 8 bits address stored in the instruction (base_addr), the base. The detailed format of each instruction is described in Figure 2.

All branch instructions use the absolute mode (8 bits) to indicate the branch target address. None of the branching instructions modifies the value stored in the condition flags. There exists six conditional branch instructions and one unconditional branch instruction, as shown in Figure 1.

III. Datapath

The set of instructions described in Section II can be sequentially executed (not in a pipelined way) in the datapath described in Figure 3. The datapath contains six special registers:
- The program counter (PC),
- the instruction register (IR),
- the address register (R@),
- the ALU register (RA) and
- the sign and zero flags (RN and RZ).

Four parts can be distinguished in the datapath:
- **Register file control:** The register file has 8 16-bit registers. R0 is a special register that always contains a 0. The register file provides one read port and one write port that can be simultaneously used.
- **Arithmetic-logical computation:** This part includes:
  - the Arithmetic-Logical Unit: the operation to be performed is controlled by means of three bits. The *OPERATE* bit proceeds from the control unit.
  - a 16-bit register to store the first operand (it is always a register from the register file)
  - a mux to select the source of the second operand, either from the register file, the memory or the *IR*. The input of the mux is selected by the control unit.
  - the flags Z and N.
- **Branch evaluation circuit:** This specific combinational circuit generates one bit that reports to the control unit the result of the evaluation of the branch condition.
- **Memory references calculation:** The memory references performed during the execution of a program are calculated in this module.

Memory references can be generated by three events:
- **Fetching consecutive instructions**: Instructions are 16-bit length. Since a memory reference contains a 16-bit word, two consecutive instructions are always stored in consecutive addresses.
- **Execution of memory-reference instructions**: Load and store instructions reference a memory word by using the relative addressing mode. Then, the address of such a word is calculated by adding an offset (contained in the bits 7-0 of the instruction) to a base address (stored in a register from the register file).
- **Read the contains of a branch target address**: In order to store the branch target address in *R@*, bits 7-0 of the branch instruction (branch target address) are added to zero (taken from *R0*) in the *ADRAD adder*.

IV. Control Unit

The control is modeled by means of a state diagram, which indicates the sequence of the operations performed in the datapath to execute each machine instruction. The control takes steps to load the registers (Ld_RA, Ld_IR, Ld_PC, Ld_R@, Ld_RZ and Ld_RN) and the register file (ERt), decides whether data are read from or written to memory ($\overline{\mathrm{R}}$/W), indicates to the ALU if an operation must be performed (OPERATE) and selects the appropriate data in the multiplexors ($\overline{\mathrm{PC}}$/@ and CRs). The control is modeled as a Moore synchronous state machine [6] (see Figure 4). The next state is decided by using both the operation code (bits 14 and 15 of the instruction) and the branch evaluation bit.

V. Simulator and assembler of the MR

A graphic simulator of the MR architecture has been developed in the Dept. of Computer Architecture. The simulator includes an assembler program to translate assembly code to machine code. The assembly language contains the basic elements to define instructions, labels, directives, expressions and macros [1].

The simulator is available via `ftp` at the address `ftp://ftp.ac.upc.es/pub/archives/mr`. Versions for DOS and WINDOWS environments can be freely downloaded.

The simulator depicts simultaneously the datapath, the state diagram of the control unit and the program in execution. The contents of the register file and the memory are presented at each moment. The buses that are active at each step are highlighted, as well as the contents of all the registers of the datapath and the value of the output signals of the control unit. The current state is shadowed in the state diagram.

The user may expand or unexpand macros (macros may be nested) and put or get break points at any place in the

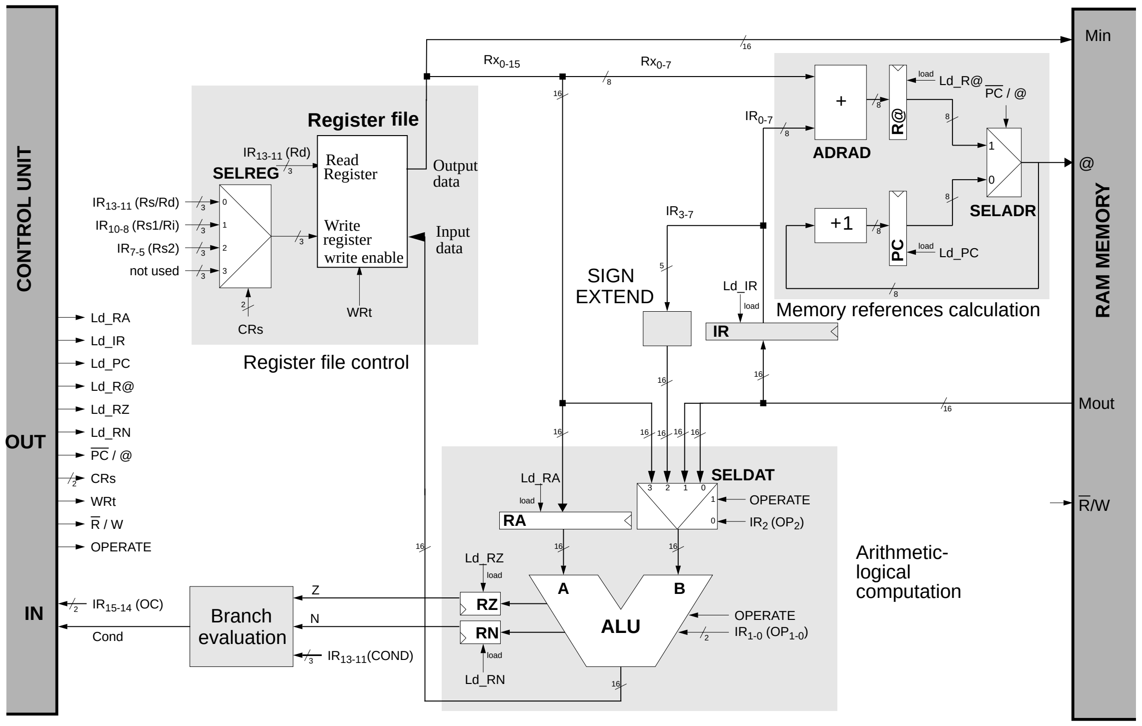
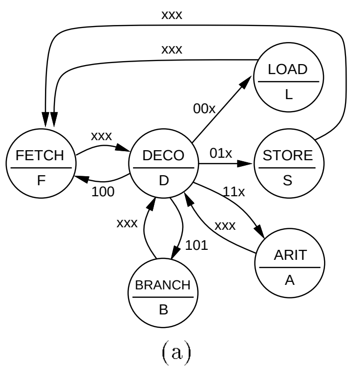


Fig. 3. Datapath of the MR.



Outputs UC	F	D	A	L	S	B
Ld_RA	0	1	0	0	0	0
Ld_IR	1	0	1	0	0	1
Ld_PC	1	0	1	0	0	1
Ld_R@	0	1	0	0	0	0
Ld_RZ	0	0	1	1	0	0
Ld_RN	0	0	1	1	0	0
WRt	0	0	1	1	0	0
$\overline{\text{R}}$/W	0	0	0	0	1	0
$\overline{\text{PC}}$/@	0	x	0	1	1	1
CRs	x	1	2	x	0	x
OPERATE	x	x	1	0	x	x

(b)

Fig. 4. State diagram and output table for the control unit.

program. The execution of a program can be also stopped at any clock cycle or after finishing every instruction.

The simulator also allows to build timing diagrams with the control signals and buses of the datapath selected by the user. This feature makes easy to the students understanding in detail how the MR works.

VI. Conclusions

This work has presented an overview of the architecture and design of the pedagogic computer named *Máquina Rudimentaria*. The MR is a useful tool to introduce the basic principles about computer structure and architecture in a first course in an Engineering or Computer Science degree. Its simple design allows a natural and fluent connection from the study of digital circuits to assembly language and computer architecture. The existence of a programming environment and a detailed simulator eases the compression of its operation (available at `ftp://ftp.ac.upc.es/pub/archives/mr`).

References

[1] R. Hermida, A.M. del Corral, E. Pastor, and F. Sánchez, *Fundamentos de Computadores*, Ed. Síntesis, 1998.
[2] E. Aiguadé, J.J. Navarro, and M. Valero, *La Máquina Sencilla: Introducción a la Estructura Básica de un Computador*, Edicions UPC, Col.lecció Aula, Num. 26, 1992.
[3] E. F. Sowell, *Programming in Assembly Language VAX-11*, Addison-Wesley, 1987.
[4] Barry B. Brey, *Programming the 80286, 80386, 80486 and Pentium-based Personal Computer*, Mac Millan Pub Co., 1996.
[5] J. L. Hennessy and D. A. Patterson, *Computer architecture. A Quantitative Approach*, Morgan Kaufmann Publishers. San Mateo, Calif., 1990.
[6] R. S. Sandige, *Modern Digital Design*, McGraw-Hill, 1990.