# AN ASSEMBLER PROGRAM FOR A 16 BIT SoC RISC PROCESSOR

*Ryan M. D. Rangel; Juliana Z. F. Diniz; Ricardo R. Linder; Geraldo M. Benício Jr.; Adson F. da Rocha; José C. da Costa*

Universidade de Brasília – Faculdade de Tecnologia – Departamento de Engenharia Elétrica
Campus Universitário Darcy Ribeiro – Caixa Postal 4386 – Brasília – DF – 70919-970 – Brasil
e-mail: camargo@ene.unb.br

**ABSTRACT**

An assembler program was developed with success for a 16 bit SoC (*System-on-Chip*) RISC CMOS processor. It was projected to manipulate an instruction set made by 16 micro-instructions and 44 pseudo-instructions. The software Montador v.1.1 is able to perform automatic syntax recognition and syntax error detection. The execution time is reduced by the use of the C++ language in implementation and the interface with the user is very friendly, based on standard text editors.

## 1. INTRODUCTION

A SoC design containing a 16 bit RISC CMOS processor has been developed [1]. Machine code and an assembly language [2] were developed to explore the capabilities of the processor.

In this study, an assembler program, the Montador – v.1.1, was developed in the C++ programming language, to translate the instructions from the assembly code (more friendly to a human user) to machine code (binary – which is the processor language). The program has the ability to check the syntax typed by the user and to show errors in syntax. Besides, the program was developed to be a very useful IDE tool, providing standard text editor facilities while keeping a compatibility with the Microsoft Windows platform.

## 2. THE RISC PROCESSOR'S INSTRUCTION SET

### 2.1. Micro-instructions

This RISC processor has 16 micro-instructions. These instructions have 16 bits of size, and are divided in:

a) Type R instructions: they use two registers as operands storing the result in another register.

b) Type I instructions : these instructions make comparison among two registers and are used for conditional branches.

c) Type J instructions : these instructions use a register and a constant in their format. They are used for unconditional jumps, shift operation and for the boolean Not.

### 2.2. Pseudo-instructions

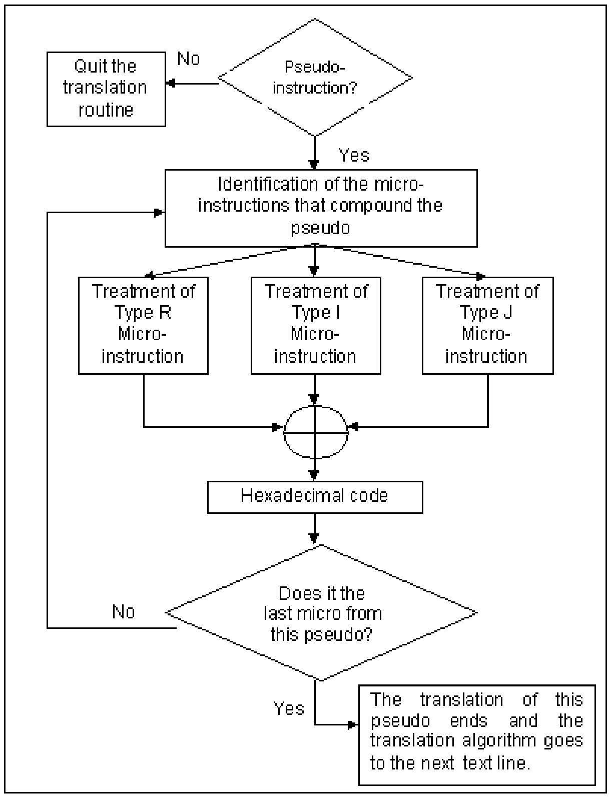
Those types of instructions are combinations of micro-instructions, implemented in software that allow the expansion of the instruction set from the original 16 micro-instructions to 60 (with the addition of 44 pseudo-instructions). The complete pseudo-instructions set [2,4] includes the micro-instructions, although in this case, the pseudo is the proper micro-instruction.

## 3. THE ASSEMBLER PROGRAM

To construct the software Montador – v.1.1 the C++ language was selected because it's a language that spends a small time to compile and execute, and it is well accepted in the Windows system, the platform chosen to the use of this software. For a better development of programs by the user, the interface (IDE) was constructed as a text editor tool, with some facilities that can be found in this environment. The assembly code must be finished with the "END" statement. If that statement is not included, the software will show an error message, in a dialog box, and the translation process will be stopped. It was used to provide an end of text indication to the C ++ text editor. After this test, the text lines will be interpreted, one by one.

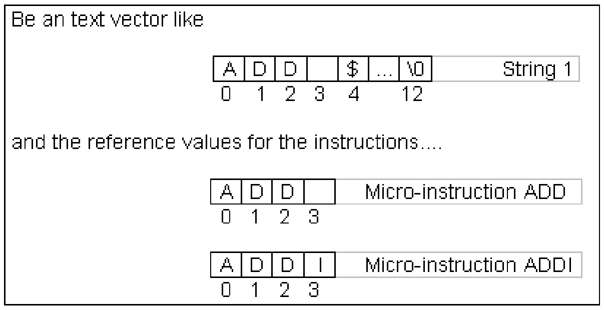
To interpret the assembly instructions, the method is identifying them as micro or pseudo-instructions. The micro-instructions will be treated according their format (R, I or J). If no match was found, then the pseudo-instructions algorithm is called. After the identification the pseudo will be decomposed into the micro-instructions that constitute it, and each one of them will be handled as a simple micro-instruction (this algorithm is presented in Fig.1.).

The comparison of entries and reference values is made considering the assembly instructions typed by the user as a vector of characters (string) and then tests are made position by position using pointers as in Fig. 2..



Pseudo-instruction?
No
Quit the translation routine
Yes
Identification of the micro-instructions that compound the pseudo
Treatment of Type R Micro-instruction
Treatment of Type I Micro-instruction
Treatment of Type J Micro-instruction
Hexadecimal code
Does it the last micro from this pseudo?
No
Yes
The translation of this pseudo ends and the translation algorithm goes to the next text line.

Fig.1 – Pseudo-instruction's treatment algorithm.



Be an text vector like
A D D $ ... \0 String 1
0 1 2 3 4 12
and the reference values for the instructions....
A D D Micro-instruction ADD
0 1 2 3
A D D I Micro-instruction ADDI
0 1 2 3

Fig. 2 – Comparison algorithm

In this example, the comparison is made until position 3, where the strings of reference are distinct between them, so the number of positions needed to be compared are determined by the number of possibilities that exists for a given character sequence. Each treatment routine tests the existence of the registers and the instruction field sizes typed by the user in the assembly commands, to increase the security level in the conversion. If an error occurs, the software identifies the line and shows the correspondent error message. If no errors happened, after each line translation, the generated hexadecimal code is written in the file "file_out.bin", and after all lines have been processed, the result will be showed to the user.

The assembler was developed and tested [3, 4] in the Borland C++ Builder 5.0 tool, a PC with an Intel PENTIUM® II 266Mhz processor, 64 MB of RAM memory, color monitor with 1024x768 of screen resolution and MS Windows 98® operational system.

## 4. CONCLUSIONS

An assembler software, with a reliable syntax analysis, was developed with success. It presents all the easiness of the common text editors andallows an easy code development.

The algorithms and functions developed in this software fulfilled the proposed specification. For future versions of this assembler, some code can be encapsulated creating new functions, and also reducing the code size.

A routine also could be added to identify the registers which will have their contents modified after a pseudo-instruction's execution. That routine shall warn the programmer about the situation.

A two step assembler can also be expanded from this software, so it will be able to recognize LABELS and commands of attribution as EQUATE (EQU) that can be used to simplify the use of constants. The RISC processor for which this assembler has been designed was successfully simulated in VHDL and a FPGA implementation was also carried out. In the future some new features may be required and the assembler's structure shall allow those modifications without difficulties.

## 5. REFERENCES

[1] G. M. Benicio Jr., *Design of a 16-bit RISC Microprocessor for a Wireless Communication System, Master´s thesis*, Dept. of Electrical Engineering, Univ. of Brasilia, Brasilia, Brazil, 2002 (in Portuguese).

[2] R. R. Linder*, A Machine Language for a Processor in a SoC, Master´s thesis*, Dept. Electrical Engineering, Univ. of Brasilia, Brasilia, Brazil, 2002 (in Portuguese).

[3] Diniz, Juliana Z.. *Development of a Basic Application for a wireless SoC.* Electrical Engineering Graduation Final Project Electrical Engineering Department , Univ. of Brasilia. Brasilia DF, Brazil, 2002 (in Portuguese).

[4] Rangel, Ryan M. D.. *Mobile Communication Application – Development of an Assembler Software to a RISC 16-bits processor.* Electrical Engineering Graduation Final Project Electrical Engineering Department , Univ. of Brasilia. Brasilia DF, Brazil, 2002 (in Portuguese).