Gulliver – Guided Navigation inside the Computer

Alfio Andronico[1], Giuseppe Di Palma[2] and Pasquale Lo Schiavo[3]

[1] University of Siena, Department of Mathematics, Siena, Italy
[2] Via Giotto, Livorno, Italy
[3] I.T.I.S. "T. Sarrocchi", Siena, Italy

Gulliver was born from the requirement for a tool able to develop computers architecture projects primarily for didactic purposes. Using Gulliver it is possible:

– to define computer architectures in an incremental way, starting from a minimal structure building up to more complex ones;

– to define symbolic program languages coherently with the model of the architecture implemented;

– to simulate functionality of the architecture and the language proposed for each stage of the development.

Such a system is of considerable utility both for the teacher, who can strengthen the effect of a traditional lesson by simple experiments directed at the topic to be deepened, and for the student as an instrument of study and/or for the autoevaluation of the learning level acquired in the subject. Finally, Gulliver can be considered to be the nucleus of a development tool for system architectural projects.

Keywords: assembly, simulation, education, interactivity, microprocessor, virtual machines

1. Introduction

The use of a symbolic language at Assembler level is widely known as a good method for computers architecture learning, because it allows the student to interact with every device without using digital logic (cfr. [Eriksson, Reijonen 90]). However, the use of a real machine of Assembler level results in practical problems which often cause the teacher to abandon doing lab experiments in favor of traditional lessons which are exclusively verbal. The simulation of a microprocessor work, accomplished with the help of a computer, can avoid a lot of these problems and permit the realization of simple laboratory experiments with virtual machines at the assembler level.

Taking advantage of the graphic capacities of modern computers, it is possible to create significant and effective models for process observation.

Gulliver permits visualization of a succession of elementary steps actually followed by a typical computer at the execution of a single instruction, showing the flow of data through the devices of the microprocessor and accenting "how" the computer reaches certain result independently of that same result.

The interactivity provided by the computer permits a type of "constructive" approach that is fundamental in many learning processes (cfr. [Carlucci Aiello, 91]). Starting from a "base" structure, Gulliver allows the defining of new devices and new instructions. So the user takes part in the definition of a more complex structure microprocessor in an incremental way.

Gulliver allows to define computer architectures in an incremental way, starting from a minimal strucuture building up to more complex one, to define symbolic program languages coherently with the model of the architecture implemented, to simulate functionality of the architecture and the language proposed for each stage of the development.

In microprocessor simulators on the market examined by the authors, the importance of good graphics is underestimated; moreover the simulated microprocessor, real or hypothetical, is complete and not modifiable. Such examples

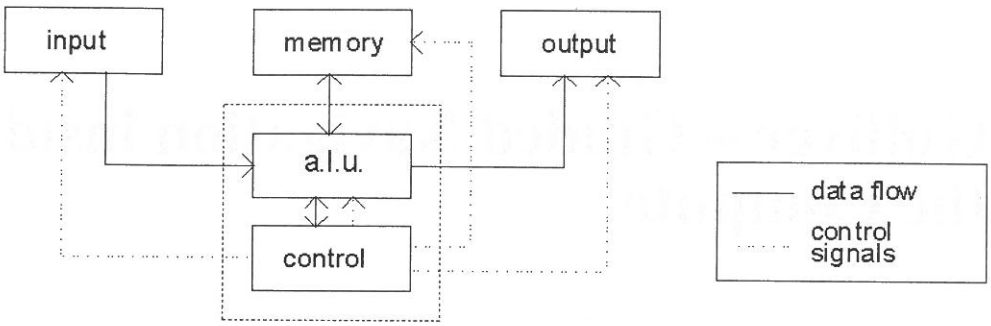

Fig. 1. Von Neumann's model.

are found in [Diab, Demashkieh 91], [Eriksson, Reijonen 90], [Forti, 93], [Hamblen, .. 90], [Lo Schiavo, Forti 94], [Reid, 91], [Silvester, 91], [Cioffi, 94].

2. The created model

Real microprocessors, to satisfy the need for minimization of costs and maximization of efficiency, are developed with very particular architectures and even with very complicated specific solutions (cfr.[Preparata 85]). However, at the base of its planning, there are principles and ideas which are generally very simple and common to all microprocessors. Gulliver's aim is to demonstrate these core concepts.

The model of the proposed computer is similar to Von Neumann's model, shown in figure 1. The control unit and the Arithmetic and Logic Unit (ALU) can be considered "internal" to microprocessor; the *input unit*, the *output unit* and the *memory* are to be considered "external" and we are not interested in investigating their structures within this context.

Globally, the microprocessor is seen as only one entity, communicating with the outside by means of three buses: the data bus, bidirectional, for data exchange between microprocessor and external devices, the address bus, unidirectional, to transfer addresses to memory in read and write operations and the control bus, bidirectional, to coordinate data interchange between the processor and the external devices. Such a scheme is visible in figure 2.

The devices inside the microprocessor are numerous and different. Following common models (cfr. [De blasi, 90]) such devices are grouped according to features and purposes in the following units: data unit, address unit, instruction unit, bus unit, control unit, arithmetic & logic unit.

– The data unit is composed of data registers used by the processor as a "cache" memory to speed microprocessor internal operations. The Accumulator register belongs to the data unit.

– The address unit is composed of those registers and devices that make reference to memory addresses. A register usually called Program Counter belongs to the address unit. A register's adder and a register's shifter can be a part of the address unit, too (cfr. [De Blasi, 90]).

– The instruction unit is composed of all the registers aimed at instructions memorization and decoding.

– The bus unit is composed of that set of signals and registers which the microprocessor shares

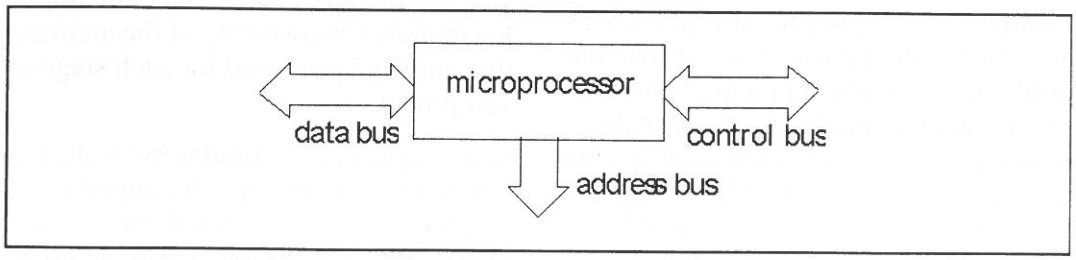

Fig. 2. Microprocessor seen in general.

	read	write	mem. request	I/O request
memory reading	1	0	1	0
memory writing	0	1	1	0
I/O reading	1	0	0	1
I/O writing	0	1	0	1

Table 1. The control signals

with external devices. The register that interfaces the data bus and the one that interfaces the address bus belong to the bus unit and are usually called Read & Write Register and Memory Address Register.

– The control unit supervises the operation of the whole microprocessor and is so connected to all other internal devices. That function, in this job, remains "understood" so as not to make the graphics dull. Instead of the control signals that supervise the comunication protocol between the microprocessor and the outside are highlighted: read signal, write signal, memory request signal, I/O request signal. Their behaviour is highlighted in the table 1, where "1" means "active signal".

The arithmetic and logic unit (A.L.U.) is here seen as a simple calculus unit and for this reason it contains only temporary registers, used in all real microprocessors in order to memorize the data waiting to carry out particular operations on them. Next to the A.L.U. there is the Flags Register, reporting the logical state of the Accumulator after the last operation is completed. There are four highlighted flags:

- carry flag: active if the operation has caused a carry;
- overflow flag: active if the operation has supplyed a non- representable result, producing an overflow error;
- negative flag: active if the result is negative;
- zero flag: active if the result is null.

Units are connected with each other by the internal data bus, by the internal address bus and by the internal control bus. The logical structure described above is shown in figure 3 (cfr. [Tanenbaum, 91]).

3. Graphics and animation in Gulliver

The visual simulation of a system necessarily involves graphic replication of its real components. In this case the system to simulate (the microprocessor) is not really visible. This leaves us freedom in the choice of the graphic to use. We decided to remain rather faithful to the scheme suggested in figure 3.

To study a microprocessor thoroughly it is also necessary to observe how it interacts with the

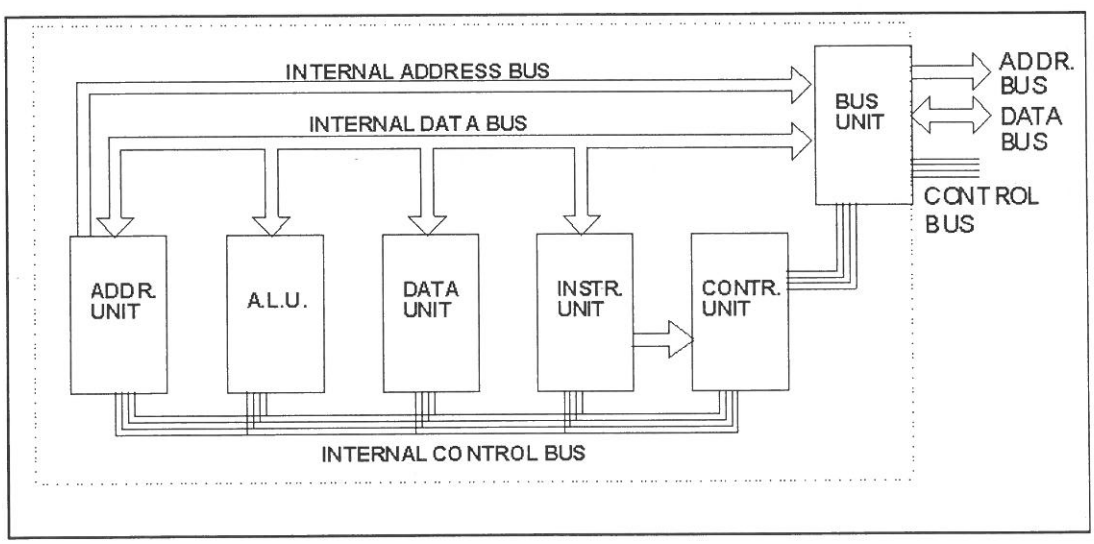

Fig. 3. The internal structure of a microprocessor.

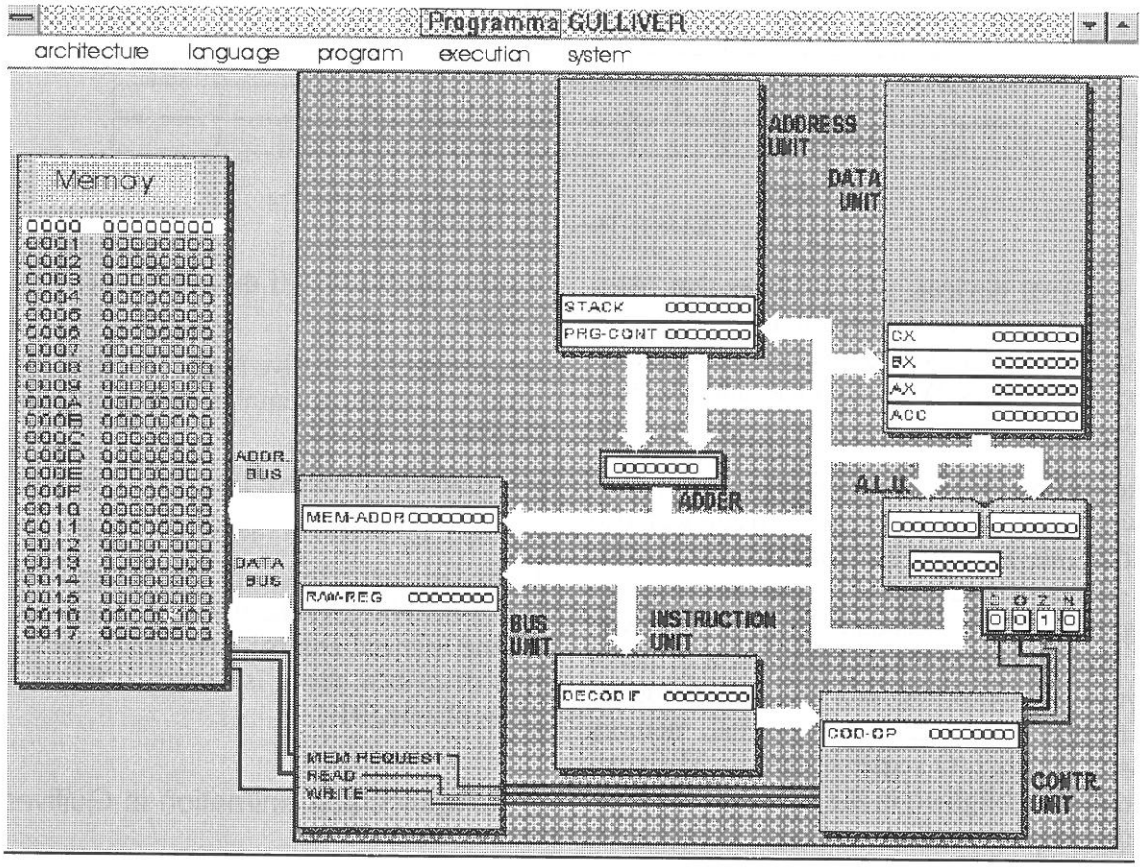

Fig. 4. The scheme of a microprocessor simulated by Gulliver.

external devices, that is, with the memory and with the input and output devices. Therefore it has also been taken necessary to simulate those devices and their connections with the microprocessor, constituited by the data bus, by the address bus and by the control signals.

The previous considerations and the problems of space utilization on the screen have led to definition of the graphic shown in figure 4.

In the simulation phase of a program execution, Gulliver shows every micro-operation of the processor, highlighting the devices and the bus involved in the data transfer. The system also shows, with a suitable coloring, the control signals and the active flags, the memory cell involved and the window that shows the program list and the instruction in execution. Finally a button with the "continue" sign enables the timing of the simulation process according to the user's need. A step of the execution of program simulation is visible in figure 8.

The simulation program is formed using a Windows-like user interface easy to use even for those who have no experience with Windows systems.

The whole system has been realized in Object Pascal for Windows.

4. Using Gulliver

In order to show the functions and the capabilities of Gulliver it is useful to take as an example one of its possible uses in a didactic environment. For example, to demonstrate the behaviour of a microprocessor in comparison with different ways of addressing, with particular reference to *direct by register addressing* and *direct by memory addressing*.

In a traditional verbal lesson it is difficult to appreciate the functional difference between these two methods, because the two methods are conceptually very similar. In both cases the instruction specifies the data address. Use of common simulators of microprocessors do not improve

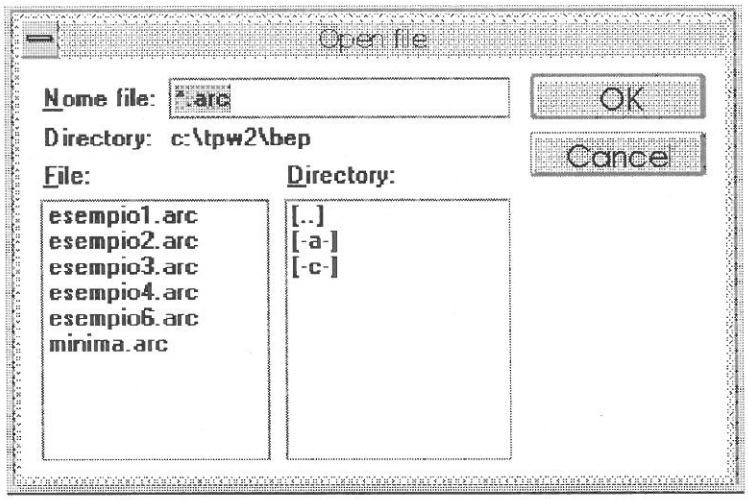

Fig. 5. Architecture's loading window.

the situation: they always underline the final result of each instruction and not the process followed by the microprocessor to execute that instruction.

Outlined below is the utilization of Gulliver to simulate this process. The system allows us to define the physical structure necessary for the experiment (the architecture), the language, the program, and at last to simulate the execution of the program.

4.1. The architecture

The first thing to define is the physical structure of the microprocessor to be used. The minimal physical working structure includes the Accumulator in the data unit, the Program Counter in the address unit, the Read and Write Register and the Memory Address Register in the bus unit, an Instrucion Register, the control unit, the A.L.U., and the Flags Register. A better look at the definition of the minimal architecture is given in [Di Palma, 94]. To prepare the experiment above described, starting from that minimal working structure, it is necessary to insert only one more register in the data unit, which will contain the data in the case of direct register addressing. Because of this we have to load up the minimal structure existing in Gulliver system and modify it by adding a data type register inside the data unit. From the main menu (pop-up style) we select "architecture" and afterwards the option "load". At this point the system shows the list of available architectures, which could contain those previously created and saved by the user. Amongst these there is always an architecture called "minima.arc". When selected, the system shows its structure. At this point we can select the options in the following order "architecture", "modifies", "add register", and "data type". The system will ask to insert the name of the new register. In this instance REG1.

The system shows in real time all modifications to the architecture. The physical structure just defined is sufficient for the appointed purposes. The modified architecture may be saved and given a name e.g. ARC1 as in figure 5.

4.2. The language

Via a guided pop-up menu the system allows the user to define the most appropriate symbolic language for the user's purposes. Real microprocessors and microprocessor simulators on the market have languages with a complete and static set of instructions. However, Gulliver allows the definition of languages with open instruction sets, a feature that forms a useful beginning for other types of exercises. However in the case of this example it is sufficient to use a minimal language. Under "minimal language" we mean the smallest set of instructions allowing the calculation of any computable numerical function. A better look at the definition of the minimal language is in [Di Palma, 94]. The system provides the minimal language that can be loaded in the same way as the architecture is loaded.

Name	*Type*	*Semantics*
INC X	arithmetic	Acc:=(X)+1
NOT X	logic	Acc:=not(X)
AND X	logic	Acc:=(Acc) and (X)
LOAD X	data movement	Acc:=(X)
STORE X	data movement	X:=(Acc)
JUMP-ZERO X	prog. control	jumps to X if fl.zero=1
JUMP-NEG X	prog. control	jumps to X if fl.neg=1
READ X	I/O	X:=(input)
WRITE X	I/O	output:=(X)
ALT	pseudo-istr.	stops execution

Table 2. An example of minimal language.

A teacher that has created exercises with a distinctive language, real or hypothetical, could however decide to alter the associated names of the instruction set in such a way as to avoid the students having to learn another language, thus maintaining a close focus on the exercise at hand. In this case a set of pop-up menus allows modifications of the language instructions with the desired names, removal of instructions, and saving of the language thus modified. For example the language defined in table 2 may be saved as LANG1.

4.3. The program

After the definitions of the architecture and of the language, the next step is drafting of the program. This level of the system is reached by selecting "edit" from the main menu.

During the editing phase the system shows a window containing the list of the language instructions and their meaning. Moreover the system provides a set of keys that enables a quick use of normal editing functions (i.e.: cut, copy,

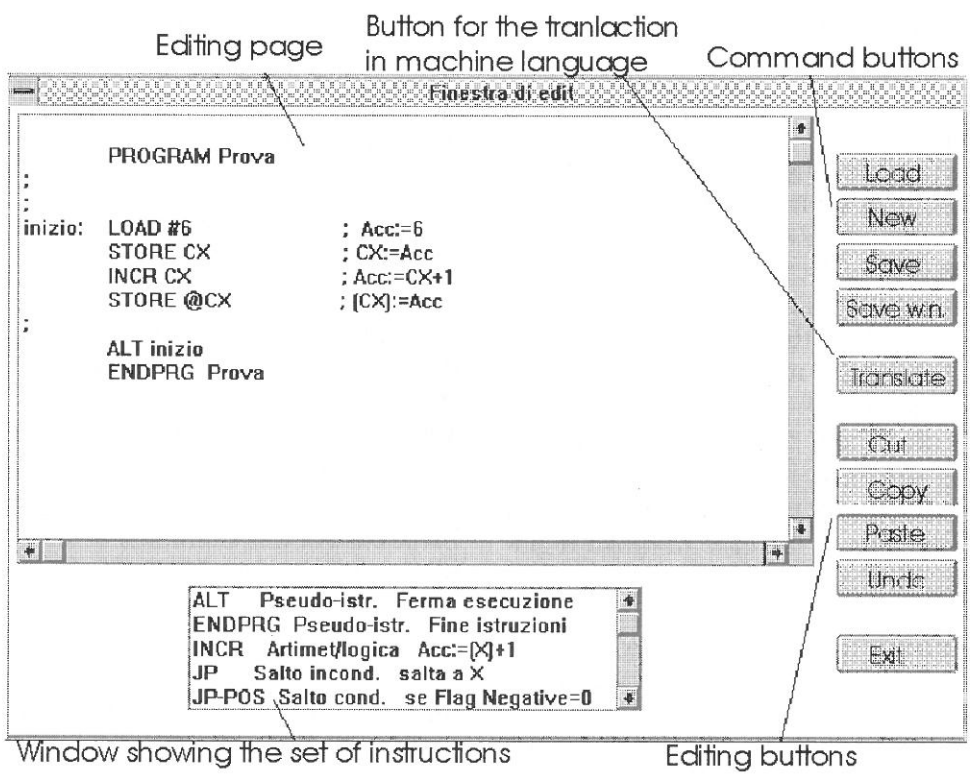

Fig. 6. The editing window.

paste, erase) and system functions (load, new, save, . . .).

In the editor window there is a button that calls the assembler translator. A current program conceived for certain architecture and certain language may generate errors if assembled with a different kind of language or architecture. In the case of an assembling mistake, the system shows a window containing the row in question and indicate the kind of error.

The program in figure 7, written in LING1 language, for ARC1 architecture, is adequate for this didactic purpose:

As we can see, the program is extremely concise, in fact it has no directives or pseudo-instructions which complicate its reading. In this way the student can concentrate on the subject observed during the exercise without being distracted by other problems.

At the end of drafting and after a correct assembly phase, the program can be saved with "save as. . . ". E.g.: PROG1.

4.4. The simulation of the program execution

When the phases of definition of architecture, language and program have been completed, the system is ready to simulate the execution. If the focus is on the simulation the previous architecture, language and program will be loaded without modifcation. Once these elements are loaded, the system will be ready for the simulation phase.

Choosing the execution option "step by step" the system scans the execution of each instruction in order to have the succession of elementary steps. It is possible to alter the time alloted to each execution step in order to suit the teaching needs best. In the simulation phase, besides showing the devices involved, Gulliver also shows a window in which the phase in execution is briefly described; in this way the student can also use the system in an automonous manner for passing over or deepening the exercise done with the teacher.

In the case of the current example, once the system is started, the teacher loads ARC1 architecture, the LING1 language and the PROG1 program previously saved and orders the program assembly. At this point the teacher can select the option "execution step by step" from the main menu.

The system will graphically show the structure of the ARC1 architecture; in the window a part of the program written in the language LING1 appears. The hexadecimal code created by the assembler appears in the memory.

At this point the teacher has the possibility to show the phases of the execution of each instruction in a dynamic way and the student observes the flow of the data among the devices.

In the "execution step by step" mode the system also simulates the fetch phase of each instruction; the teacher can decide to remove this phase (equal for each instruction) by selecting the "execution without fetch" modality.

In this example the teacher will note how the execution of an instruction with a direct-through registers addressing is quite different from the execution with a direct-through memory addressing, even if these two methods are conceptually equal.

Making reference to the program shown in figure 7, in the case of the instruction STORE REG1, that uses a direct-through registers addressing, after the fetch and decoding phases, the real execution consists of only one passage of data between the accumulator and the register REG1.

```
begin:  LOAD #10          ; Acc:=10 (16)
        STORE REG1        ; REG1:=(Acc)
        STORE LABEL       ; LABEL:=(Acc)
end:    ALT begin         ; interrupt the execution
LABEL:                    ; defines the label LABEL
```

Fig. 7. The program PROG1

In the case of the instruction STORE LABEL, that uses a direct-through memory addressing, the microprocessor has to:

- recover memory cell address labelled LABEL;

- copy the memory cell in the Address Register;

- copy the accumulator's contents in the Read & Write Register;

- effect a write operation in the memory.

Gulliver points out this difference in an easy and effective way; other simulators, showing in every instruction only the starting and finishing states of the microprocessor, don't permit such underlining of the essential difference between these two addressing methods.

5. Didact trials with Gulliver

To verify the didact validity of Gulliver we tested the system within a university environment. Below are the results of the tests carried out with the students at I.T.I.S. "Tito Sarrocchi" in Siena.

The students first study these subjects (assembler languages, microprocessor and architectures) at the end of the third year. The trials took place at the beginning of the fourth scholastic year.

5.1. First trial

For the first experience, the students of the class were divided at random into two groups, A and B, in equal number. Group A followed a traditional lesson for about twenty minutes, conducted by a teacher. Simultaenously group B set out to learn the same concepts with a computer's and Gulliver's help.

Later on all the student undertook a test composed of eight questions to which they had to answer in a discorsive manner. In order to provide a control measure, two of these questions concerned topics covered neither during the traditional lesson nor in that using Gulliver. This

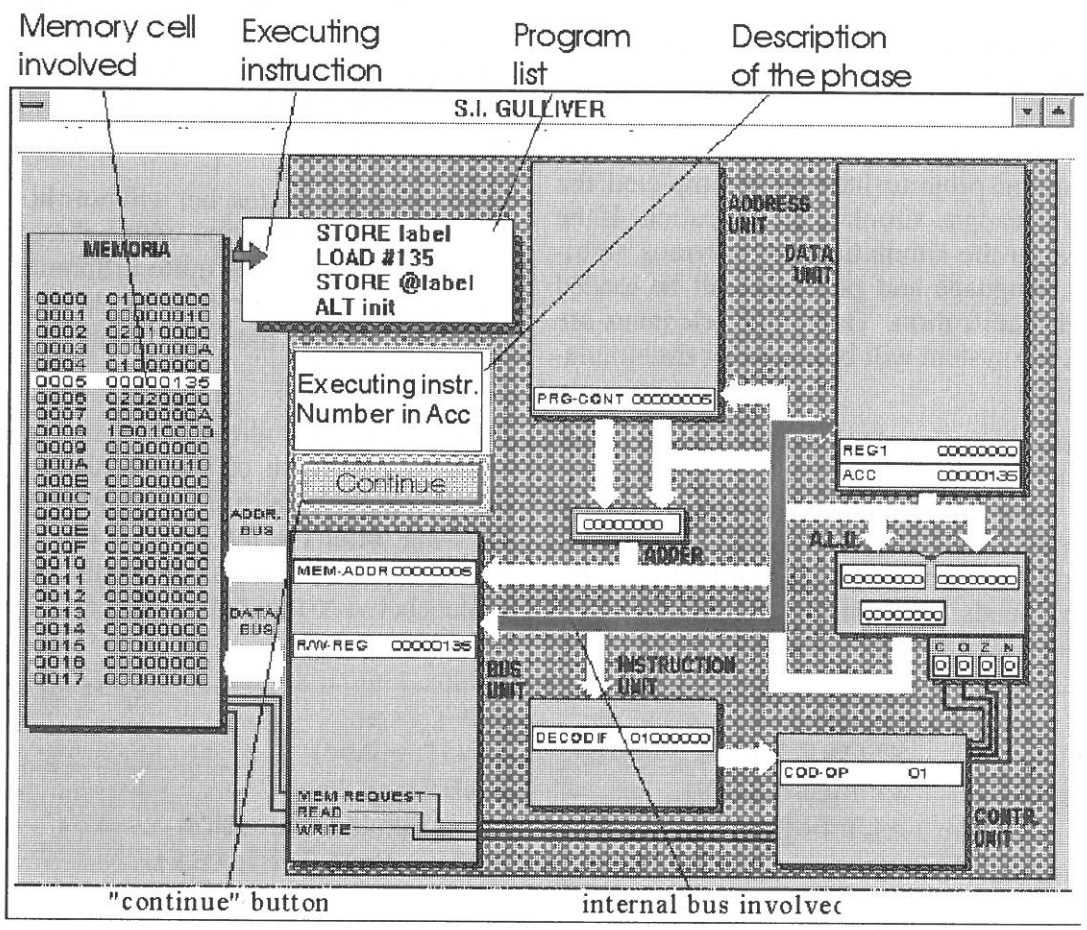

Fig. 8. A step of the program execution simulation.

was done to compare in a significant way the test results obtained from the students of group A to those obtained from the group B.

1) *Which physical devices are necessary for the operating of a microprocessor?*

2) *Describe minutely the fetch phase of an instruction.*

3) *What is the functional difference between direct-through register addressing and indirect-through registers addressing?*

4) *In terms of efficency (velocity of execution) which between the two previous addressings is preferable and why?*

5) *Which are the instructions that check the state of the flags?*

6) *Which micro-operations are done by the microprocessor during a jump instruction?*

7) *Which control signals are involved in a memory reading operation? And in one of writing?*

8) *Which devices, besides the essentials, are necessary to develop a relative to the program counter addressing?*

Looking first at the two control questions it was expected that the responses of the two groups should have given similar results, because the topics in question hadn't been treated that day. The general results obtained in relation to these two questions are shown below in table 3.

Answer	*group A*	*group B*
Correct/complete	15	14
General/partial	4	2
Lacking/wrong	5	8

Table 3. Results of the general character questions.

The results of other questions, whose topics had been treated during the traditional type lesson and during the lesson using Gulliver, are shown in table 4.

Answer	*group A*	*group B*
Correct/complete	5	39
General/partial	44	30
Lacking/wrong	23	3

Table 4. Summary results of the first trial.

The results summarized in table 3 indicate that preparation of the students in the two groups was similar. However, table 4 shows a big difference in the results obtained in the test. The students who worked with Gulliver's help have generally furnished more detailed responses, but above all, these were complete explanations. The use of Gulliver was effective in particular to understand the fetch phase, the difference among the various addressing methods, and the main devices for a micoprocessor. Moreover, using Gulliver the students seemed able to generalize some basic concepts.

5.2. Second trial

In the second trial the class were divided in eight group of three students. In the first phase they were asked to write a program that could calculate the sum of two numbers using various addressing methods allowed by Gulliver. Gulliver provided all students with a minimal language and an architecture including minimal devices and with two registers of general use.

The aim of the first phase of this trial was to clarify the running of a standard microprocessor and to make the differences between various addressing methods appreciated. The students had no major difficulty in writing programs that would add two numbers using the various addressing methods provided by the system and in the simulation phase of the various programs excution they spontaneously commented on the conceptual and, above all, functional differences between various methods.

In the second phase of the second trial the students were asked to write a program that would calculate the product of two numbers through sum operations. Besides the minimal language, this time the students were provided with the minimal architecture and they were allowed to add the devices that they considered handy.

The aim of that particular phase of experience was to make the students interact more with the system enabling them to modify the microprocessor structure to work on. In fact, the students, some immediately, and others after a

short discussion, decided to insert into the simulated microprocessor a general use register to use like a counter. Several students underlined how introduction of the counter register instead of the use of the memory provided the possibility of both writing and executing the program more briefly. At the end of the experience, 5 groups were able to obtain a correct program and 3 groups not.

The lesson effected with the use of Gulliver had, in comparison to a traditional lesson, an indisputabily more active participation of the students. In particular graphic description of the processes seems to have been didactically valuable. Many students appreciated the possibility to modify the microprocessor structure, underlining the difference between this simulator and a normal debugger.

6. Possible developments

Gulliver has been developed in such a way as to work in Windows and it offers a user interface similar to any application of this system. It has also been designed in a way as to be able to be enlarged without excessive difficulty, because of its highly structured nature.

A possible evolution would be the simulation of the model of microprocessors with two addresses instructions, not because this is of didactic importance but rather to provide the system with the possibility of emulating a larger range of microprocessors. A further development is the possible introduction in Gulliver of a grading tool to measure the efficiency and complexity of the algorithms. In assisted didactics an important step could be the addition of a verifying section with the introduction of an interactive problem generator.

References

CARLUCCI AIELLO L., Intelligenza artificiale e formazione, in *GOLEM* Anno III, n. 1/2, Gennaio/Febbraio 1991;

CIOFFI G. AND JORNO A., VILLANI T., *Il processore PD32: architettura assembler e simulatore*, Masson S.p.a., ed. ESA, Milano, 1994;

DE BLASI M., *Computer architecture*, Addison-Wesley Publishing Company, 1990;

DI PALMA G., S.I. Gulliver: un sistema interattivo per lo studio di microprocessori, Thesis, Università degli Studi di Siena, A.A. 1993/94;

DIAB H.B. AND DEMASHKIEH I, A Computer-Aided Teaching Package for Microprocessor Systems Education, in *IEEE Trans. Education*, n.2, May 1991;

ERIKSSON I. AND REIJONEN P., Training computer-supported work by simulation, in *Education & Computing*, n.1,2, July 1990;

FORTI G., Un sistema per la progettazione e la simulazione di microprocessori, Thesis, Università degli Studi di Siena, A.A. 1992/93;

HAMBLEN J.O., PARKER A. AND ROHLING G.A., An Istructional Laboratory to Support Microprogramming, in *IEEE Trans. Education*, n.4, Nov. 1990;

LO SCHIAVO P. AND FORTI G., Un sistema interattivo per la definizione e la simulazione di microprocessori, in *Didamatica '94 – Atti* (A. Andronico, G. Casadei and G. Saceroti editors), Cesena;

PREPARATA F.P., *Introduzione alla organizzazione e alla progettazione di un elaboratore elettronico*, Franco Angeli Libri s.r.l., Milano, 1985;

REID R.J., Computer-Aided Engineering for Computer Architecture Laboratories, in *IEEE Trans. Education*, n.2, May 1991;

SILVESTER P.P., Introducing Computer Structure by Machine Simulation, in *IEEE Trans. Education*, n.1, Feb. 1991;

TANEMBAUM A.S., *Computer architecture*, Milano, Gruppo Editoriale Jackson, 1991.

Received: September, 1997
Accepted: January, 1998

Contact address:
Alfio Andronico
University of Siena
Department of Mathematics
Via del Capitano 15
53 100 Siena
Italy
phone: 0577-263742

Giuseppe Di Palma
Via Giotto, 20
57 100 Livorno
Italy
phone: 0586-859056

Pasquale Lo Schiavo
I.T.I.S. "T. Sarrocchi"
Via C. Pisascane
53 100 Siena
Italy
phone: 0577-49080

Alfio Andronico is full professor of foundations of computer science at the Engineering faculty of the University of Siena. He is head of steering committee for engineering diplomas in Informatics and Telecommunications. He has published approximately 100 works including books edited and coedited, technical reports in computer science research, methodology, applications in different fields, operational research and education. He is scientific coordinator for an Italian National Conference named DIDAMATICA and head of the AICA's (Italian Association for Information Processin) Working Group dealing with the Computer in Education.

Giuseppe Di Palma obtained his degree in Mathematics at Siena University in 1994. He has interests in educational technology and informatics. He is presently teaching informatics at the Livorno Naval Academy. He is developing a project regarding computer aided teaching in scientific field.

Pasquale Lo Schiavo has a degree in physics and is a systems teacher at a secondary school in Siena. He has been working for years in educational technology and informatics at the University of Siena where he has contributed to publications on these subjects. In recent years, he has devoted himself to the planning, construction and experimentation of educational software, and in the "Gulliver System" described in this paper.