Available online at www.sciencedirect.com

ScienceDirect

Computers & Education 49 (2007) 891–915

COMPUTERS & EDUCATION

www.elsevier.com/locate/compedu

# Applying a constructivist and collaborative methodological approach in engineering education

Lorenzo Moreno [*], Carina Gonzalez, Ivan Castilla, Evelio Gonzalez, Jose Sigut

*Escuela Tecnica Superior de Ingenieria Informatica, Universidad de La Leguna, 38200 La Laguna, Tenerife, Spain*

Received 9 September 2005; received in revised form 18 November 2005; accepted 15 December 2005

## Abstract

In this paper, a methodological educational proposal based on constructivism and collaborative learning theories is described. The suggested approach has been successfully applied to a subject entitled "Computer Architecture and Engineering" in a Computer Science degree in the University of La Laguna in Spain.

This methodology is supported by two tools: the Moodle platform as a collaboration framework among students and teachers and a free Instruction Level Parallelism (ILP) processor simulator called SIMDE, developed by the authors to promote the experience and help the understanding of superscalar and VLIW processors.

This work is described showing how the constructivist and collaborative approaches have been applied and how the activities have been structured temporarily in phases. This educational proposal has been validated and improved with the feedback of the students during two academic years.

Furthermore, the methodological procedure is also suitable to be used not only in subjects with contents which require the understanding of dynamic situations but also in subjects with other requirements.
© 2005 Elsevier Ltd. All rights reserved.

*Keywords:* Educational technology; Collaborative learning; Constructivism; Simulators; E-learning; Computer architecture

[*] Corresponding author.
*E-mail address:* lmoreno@ull.es (L. Moreno).

0360-1315/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compedu.2005.12.004

## 1. Introduction

A fundamental topic in teaching Computer Architecture is the ILP (Instruction Level Parallelism) subject. Concepts about ILP need the understanding of dynamic situations and implicit parallelism in the different functional units of processor. Therefore, a difficult challenge to be solved by the teacher is to show how the execution process of instructions is carried out.

Furthermore, another key point to be taken into account is the previous knowledge of the student and his misconceptions. Both things are essential for every constructivist approach. Studying these difficulties can help in the design of an effective learning process. Their background, i.e. the foundations upon which the students will construct their models, will clearly affect the way in which the new knowledge is assimilated. The instructor needs to know what to pull down, if necessary. Thus, it is necessary to carry out an exploratory learning, exploring their own ideas and beliefs about ILP processors. On the other hand, it is essential to identify their learning difficulties as a way to understand their cognitive process.

In previous courses, the students have been in contact with several aspects referred to Computer Architecture. They are familiar with concepts such as cache, float units, registers, etc., but they do not dominate concepts such as out of order execution, ILP, branch prediction, etc. From their programming experience, in both high and low level languages, a number of students seem to have adopted, among others, the following misconceptions:

(a) The processor cannot start to execute an instruction until it has not finished executing the preceding one.
(b) The effects of every processed instruction cannot be reversed, in other words, every instruction that enters into the processor is always executed.
(c) Although several instructions can be executed in a parallel way, there is only one "flow" of execution.
(d) The order of execution policy in every processor is similar to the execution flow tested in a Borland-like debugger.

From their experience for several years in teaching activities, the authors have identified three possible sources of learning difficulties when students face ILP processing:

(a) Parallelism in instruction execution.
(b) Out of order execution.
(c) A good number of parameters to define: cache levels, predictive strategies, Translation Lookaside Buffer (TLB), Branch Target Buffer (BTB), number of positions in the reorder buffer, number of functional units, number of reserve stations, etc.

These issues have been shown to be difficult to understand from the reading of a text book since they are too complex to be represented in students' mind. Even in highly acclaimed reference texts like "*Computer Architecture: A Quantitative Approach*" by Hennesy and Patterson (2003), each example depends on specific simplifications that make more difficult to put all concepts together. It is clear that a more visual way of representation (for example, simulators) is desired.

Traditionally, books and simulators have been used as the basic tools to explain these concepts. However, the use of a simulator as a didactic resource is necessary but not enough. It is necessary because it allows to display the internal structures of the processor during the execution process of instructions in the examples proposed by the teacher. Moreover, the student can easily work with different implementations and optimizations. On the contrary, it is not enough because other activities that permit knowledge elicitation, its understanding and the transference of theoretical concepts to the real world must be earned out. For this purpose, several activities on an e-learning platform that allows the collaborative work and a better adaptation to the learning styles of students have been designed. Anyhow, we think that face to face interactions among students in class as well as the tutoring are still essential to a better tracking of the student and the workgroup.

A simulator called SIMDE has been developed in the University of La Laguna to improve the understanding of the concepts mentioned above. SIMDE allows a visualization of the complete execution process of instructions in superscalar and Very Long Instruction Word (VLIW) processors. This simulator has been used in a Computer Architecture subject during two years. In the first year, the students had a more passive role in the learning using the simulator, because they only had to analyze characteristics of different processors in examples designed by teachers. In the second year, this simulator has been included as part of a constructional and collaborative educational methodology that permits learning by practice. In this way, the students had to create their own code optimizing it according to the techniques analyzed during the course.

This methodology allows to encourage the learning, to increase the experience, the active role and social abilities of students in the building of the knowledge, promoting the role of professor and the simulator as mediator.

In addition, in order to implement this type of learning in a virtual environment, a flexible e-learning platform that supports these pedagogical principles is needed. Also, this platform must facilitate the communication, the coordination of tasks, the tracking down and interpretation, the common work and the reuse of the obtained results. Moodle[1] is an open source e-learning platform which satisfies these requirements and this is the main reason why we have chosen it.

This paper is organized in the following way. In Section 2 the educational theories and tools that support the constructivism and designed collaborative environment will be described. Section 3 explains the proposed collaborative environment and the activities organized in phases. The exercises proposed with the simulator SIMDE are detailed in Section 4 together with the feedback from the students. The evaluation framework of the methodology will be reported in Section 5. Finally some conclusions are offered in Section 6.

## 2. Educational foundations

In this section the theoretical issues which support this work and the technological tools used to implement the collaborative environment will be described.

[1] Moodle. http://moodle.org/.

## *2.1. Theoretical framework*

Using correctly and effectively the technology in education require that the application be supported in proven pedagogical theories. The methodology that we have used in the Computers Architecture's subject combine theoretical and practical procedures based on CSCL (Computer Supported Collaborative Learning) (Gifford & Enyedy, 1999; Inaba, Supnithi, Ikeda, Mizoguchi, & Toyoda, 2000; Martinez, Dimitriadis, Rubia, Gómez, & de la Fuente, 2003) and the constructional learning (Bruner, 1991, 1997; Jonassen, 1999; Koschman, 1996 Jonassen) as follows:

*Feature I:* Constructivism proposes to give more significance to the *learning contexts* as an alternative to the memorization. This permits to *build knowledge*, doing activities closer to the real world and generally involves discussion groups (Crook, 1998). The significant contexts for the constructivist authors are *situations of the real world that help to put into practice the experience* (Knuth & Cunningham, 1991).
*Feature II:* The *learning environments must be flexible* and are characterized by the fact that the same knowledge can be represented in different ways. So, the students learn through the variety of proposals (Spiro, Feltovich, Jacobson, & Coulson, 1991).
*Feature III:* Regarding the role of the *computer* in the constructivist environments, the constructivist authors consider that it should not be used only merely to put out knowledge.[2] On the contrary, it must be a *supporter tool for the experimentation and building of knowledge.*
*Feature IV:* Martí (1992) made a proposal to the Papert's methods based on the double axis: *application to specific instructive situations of constructivism and mediation of learning through computers and people*. According to him, it is possible that through the individual exploration the student can acquire determined general schemes of knowledge, but it is much more difficult that they can achieve specific learning.
*Feature V:* Martí considers the necessity of *defining the instructive situation starting from the previous ideas of the students* and it is essential to define the type of intervention of other people: teacher and students.
*Feature VI:* Other main characteristic of a constructivist learning environment is that it proposes a "communitarian or collaborative learning", where the students work together helping each other, reinforcing the *social dimension of the education* (Vygotsky, 1979). This humanistic method is based on principles of experience as well. Through the activity and the experience, the best results are obtained. When in these environments there is a computer as mediation, we are talking about "Computer Supported Collaborative Learning (CSCL)". CSCL is focused on how collaborative learning supported by technology can enhance peer to peer interaction and work in groups, and how collaboration and technology facilitate sharing and distributing of knowledge and expertise among community members (Lipponen, 2002). So, computer resources act as mediators. From the CSCL view, the student is an active agent builder of his own learning process, a person who has to generate knowledge. CSCL has been successfully applied for teaching in the Computer Architecture area and several related software environments such as (Gogoulou, Gouli, & Grigoriadou, 2003; Gogoulou, Gouli, Grigoriadou, &

[2] Papert, S. http://www.papert.org/.

Samarakou, 2003; Grigoriadou, Toula, & Kanidis, 2003; Osuna et al., 2000; SYNERGO Collaborative Mapping Environment, 2004) have been developed.

On the other hand, Chickering and Gamson (1987) summarize the seven principles for good practice in undergraduate education which should be covered as much as possible by every teaching design. These principles can be enumerated as follows:

1. Encourages contact between students and faculty
2. Develops reciprocity and cooperation among students
3. Encourages active learning
4. Gives prompt feedback
5. Emphasizes time on task
6. Communicates high expectations
7. Respects diverse talents and ways of learning.

It is noted that these principles will be cited in the remainder of the paper using its order in this enumeration.

### 2.2. Technological resources

As we have mentioned above, the authors have made use of two tools: the Moodle platform (footnote 1) as a collaboration framework among the students and teachers and a free Instruction Level Parallelism (ILP) processor simulator called SIMDE, developed by us to promote the experience and help the understanding of superscalar and VLIW processors.

#### 2.2.1. Moodle's features

Moodle is a course management system (CMS). This free software package which has been designed using sound pedagogical principles, helps educators to create effective online learning communities, to participate actively in the learning process and to collaborate in groups. The collaborative activities often promote metacognitive processes such as reflection, self-explanation and self-regulation. This platform allows the management of contents, students and teachers and offers a large variety of resources and activities, such as quizzes, consults, diaries, workshops, SCORM objects, among others. One of the activities included in Moodle, the Wiki, was conceived as collaborative since its beginnings. Wiki is a software application that permits to create collectively documents on the web using a simple scheme of labels and marks. Also, it is not necessary any revision or acceptation of content for publishing it on Internet.[3]

As it is suggested by principle 1 in the Chickering and Gamson list, this platform increases the student–faculty contact time since a student is in contact with their peers and with the learning subjects for more time using both asynchronous (forums) and synchronous (chats) tools. On the other hand, it encourages the collaborative work among students, as stated in principle 2,

[3] Wiki definition. Wikipedia. http://es.wikipedia.org/wiki/Wiki.

allowing the sharing of ideas and research results. Principle 3 is also promoted. It is noted that students only remember the 20% of what they listen but the 90% of what they talk about and make (Felder, Felder, & Dietz, 1995). Moreover, the students can schedule their learning time, as mentioned in principle 5, since the web platform is available 24 h per day/7 days per week. Principle 7 is reached because the collaborative work in the platform offers the students the possibility of showing their talents and learning from the others. Although principles 4 and 6 are not directly offered by the Moodle platform, it is a basis for them. The Moodle offers auto-evaluation tools that can be administered by the professor and it can be a framework where the students download well-prepared and high-quality materials, usually prepared by the professor. In the designed approach both principles have been included.

The platform offers a discussion framework where the students can either complete or put the finishing touches to the meaning negotiation threads started in the class using the forums and wikis services. They can implement new activities, managed and administrated by them, about the ILP concepts or repeat the same activities developed in the class. In this sense, the advantages offered by the "Living Pipeline" activity may be increased in this virtual environment.

### *2.2.2. SIMDE simulator description*

The use of simulators in Computer Architecture and Organization courses is a usual practice to bridge the gap between theoretical knowledge and practical experience (Denhiere & Baudet, 1992; Vosniadou & Kolias, 2003). Simulators' potential is considered far more efficient than other conventional tools. A variety of simulators have been used for this purpose: Dinero Edler and Hill (1997) (cache), Scott (2005) (RISC), WinDLVVSim López and Calpé (1998) (vectorial), MIDAS Silhan and Fuss (1997) and SATSim Wolff et al. (2000) (superscalar), TMS320C6xxx Cuppu (1999)(VLIW), etc. Some of these simulators have been used for ILP teaching, but they still have some disadvantages such as an excess of specificity found in commercial simulators (TMS320C6xxx), lack of GUI (Simple Scalar) or not stable versions (MIDAS). Simulators offer simpler versions of real world, but in their design the developers should not fall in an excessive simplification that could imply new misconceptions.

SIMDE covers simultaneously two different approaches of ILP: superscalar and VLIW processors. This two-in-one approach makes possible to emphasize similarities and differences between both ILP Architectures. The focus of effort has been the definition of a basic structure which has been used as a shared basis for both superscalar and VLIW designs. The common structure consists of the following components:

- Instruction set: A MIPS IV-like instruction subset repertoire has been used, including arithmetic instructions (ADD, ADDI, SUB, MULT, ...), bit instructions (OR, AND, XOR,...), shift instructions (SLLV, SRLV), floating point instructions (ADDF, MULTF, ...), load/store instructions (LW, SF, ...) and branch instructions (BNE, BGT, ...).
- General Purpose Registers (GPRs): Sixty-four 32-bit registers (R0, R1, ..., R63) have been provided for integer operations. The value of R0 is always 0.
- Floating Point Registers (FPRs): Sixty-four additional 32-bit single precision registers for floating point operations.

- Memory: 1024 32-bit words compound main memory. Two specific caches (instruction and data) have been added to improve memory access. The instruction cache has been designed in such a way that programs (both sequential and VLIW code) are completely allocated on it. Thus, no instruction cache misses can happen and many simplifications in the general design can be assumed. On the other hand, the data cache allows cache misses in a user-defined proportion.
- Functional Units (FUs): Six specific pipelined FUs have been designed: integer adder, integer multiplier, floating adder, floating multiplier, memory and branch. The number of FUs of each type is a user-defined parameter. Each type of FU has a specific number of pipeline stages. This parameter (called FU latency) is also user-defined.
- Program Counter (PC): It has identical functionality in both processors: Superscalar and VLIW. The difference is that PC points to single instructions (or operations) in Superscalar processor and to long instructions in VLIW processor.

The superscalar processor is based on Tomasulo's algorithm (Tomasulo, 1967) in order to remark the differences with the software-based VLIW approach. Users can modify the issue rate from 2 to 16 instructions per clock. The designed processor implements multiple issues, branch prediction and speculation and it has the following components:

- Prefetch unit loads instructions from the instruction cache in a transparent way for the rest of hardware. This unit predecodes the instructions in order to look for branches. When a branch instruction is found, prefetch unit checks the branch prediction table for the result of this branch. If the result is false, it continues loading instructions in sequential order. Otherwise, prefetch unit loads instructions from the branch target.
- Decoder checks instructions for true data dependences (due to RAW hazards). Then, it distributes the instructions to the reorder buffer and among the reservation stations.
- Reservation stations (REs): There is one RE per type of FU. The instructions are buffered here until execution is finished. All of the reservation stations simultaneously monitor their source operands looking for data availability. When all the operands of an instruction are ready, the instruction may be issued (subject to hardware resource availability).
- Reorder Buffer (ROB) is a buffer implemented as a circular queue. It allows instructions to commit in-order and it supports speculative execution and register renaming.
- Branch prediction table: It uses a 2-bit dynamic branch prediction strategy.
- An extra address FU (a sort of integer adder FU) has been added in order to split memory access just as Tomasulo's algorithm does.

In contrast to superscalar design, hardware simplicity is the main goal in VLIW processor design. Only a few structures have been added in order to characterize VLIW processor:

- Predicate registers. Sixty-four 1-bit registers ($p0, p1, \ldots, p63$) have been provided for predication support. The value of $p0$ is always 1 (true). A full-predication scheme is applied: all operations have an associated predicate register ($p0$ if predication is not desired for this operation). Predicate registers are set/reset by branch operations. Thus, multiway branching is supported too.

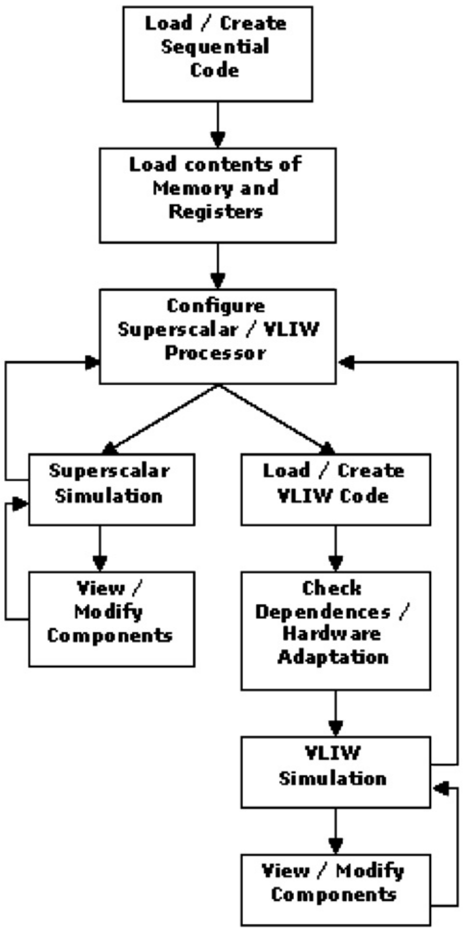

Load / Create Sequential Code
Load contents of Memory and Registers
Configure Superscalar / VLIW Processor
Superscalar Simulation
Load / Create VLIW Code
View / Modify Components
Check Dependences / Hardware Adaptation
VLIW Simulation
View / Modify Components

Fig. 1. Typical execution flow.

- One NaT (Not a Thing) bit has been added for each GPR and FPR. A NaT bit is set when a cache miss occurs in a load operation using its associated register as target. If another operation tries to use this register, an exception is raised and instruction issue is stalled. Instruction issue continues when the load operation has finished (the main memory access has finished).

The designed VLIW processor issues one long instruction per cycle. Long instruction format consists of as many single instructions (so-called operations in this context) as defined FUs. Branch operations are limited to one per instruction in order to simplify the design.

SIMDE is freely available for academic use from its website,[4] covering principles 5 and 7 of Chickering and Gamson. It has been designed with a grid-based graphic interface that achieves

[4] SIMDE website. http://www.cyc.ull.es/simde.

a clearer presentation of contents and a more intuitive interaction between the students and the processor components. The practical experience is reinforced with a contextual help easily accessible from any part of the simulator. This help includes not only a user guide for the simulator but a quick revision of the theoretical concepts that the students may need.

A custom procedure set flow is provided in Fig. 1. Sequential codes can be easily created by using any plain text editor such as Windows notepad. These codes can be used to be simulated on the superscalar processor or as a basis for a VLIW code.

The contents of the memory and the registers of both processors can be modified before and during simulations. These contents can be saved/loaded to/from a file. The users can tune several parameters of the simulator such as:

- Data cache miss probability
- Number of FUs of each type
- Latency of FUs of each type (number of pipeline stages)
- Maximum instruction issue rate (only Superscalar processor)

Both, Superscalar and VLIW processors, provide common simulation features such as:

- Customization of the execution window by hiding or showing components.
- Continuous and step-by-step execution. Breakpoints can be also used.

The VLIW simulation has the following additional features:

- VLIW codes can be created from scratch by using as basis a sequential code. Furthermore, they can be saved/loaded to/from a file.
- An easy code scheduling mechanism is achieved by dragging the operations from the sequential code panel and dropping them on an appropriate long instruction cell.
- Operations and long instruction words can be added and deleted by the user.
- Predication can be applied to any operation by simply double-clicking on it.
- Branches are completely customizable: target, predicate registers, etc.
- The area of influence of an operation (i.e. the amount of long instructions where a true-dependent operation should not be scheduled) can be coloured in order to help the students to avoid true dependences.
- An auto-check tool ensures code correctness.

The superscalar simulation allows colouring each instruction in the ROB in order to track it along its execution flow.

## 3. Description of the methodological procedure based on constructivist and collaborative principles

In this section, the authors describe the way in which the work has been structured and how the constructivist and collaborative approaches have been applied.

The methodological procedures have been organized in several phases and activities:

1. *Preparation phase:*
   (1) Theoretical classes about ILP processors: superscalar and VLIW.
   (2) Creation of the teams.
   (3) Resolution of some exercises in class.
   (4) Presentation of the SIMDE simulator.
   (5) Presentation of the Moodle platform.
   (6) "Living Pipeline".

2. *Experimentation phase:*
   (1) Using the SIMDE simulator in the problem solving.
   (2) Using the Moodle platform for the interaction/discussion inter/intra-group and interaction with the professor through forums, chats and email.
   (3) Generation of a WIKI.

3. *Presentation and demo phase:*
   (1) Group presentation in class.
   (2) Discussion about the presented works.
   (3) Generation of a final document about the ILP processor features and how these features are included in the analyzed commercial machines.
   (4) Demonstration of the analyzed optimization techniques in the assigned type of problem using the SIMDE simulator.
   (5) Resolution of surveys about SIMDE's technical aspects.
   (6) Resolution of surveys about SIMDE's educational aspects.

4. *Final phase:*
   (1) Auto-evaluation tests.
   (2) Final exercise about ILP architectures.
   (3) Analysis of the generated documents.
   (4) Analysis of the platform registers and how the students have taken part in the procedures/discussions.

Fig. 2 shows a scheme with the organization of the activities in phases and categories of analysis of the learning process which are involved in the evaluation method.

In order to achieve a constructivist learning approach in agreement with our methodological proposal, the following basic questions of constructivism must be answered:

(1) How is active learning reached or how can knowledge be built by means of our procedure?

Active learning is reached in our procedure through the following activities that include negotiation, mediation and experimentation:

- Several *debates are moderated by the professor*, after a period of two weeks in which the students are encouraged to read some fragments related to ILP processors from Hennesy

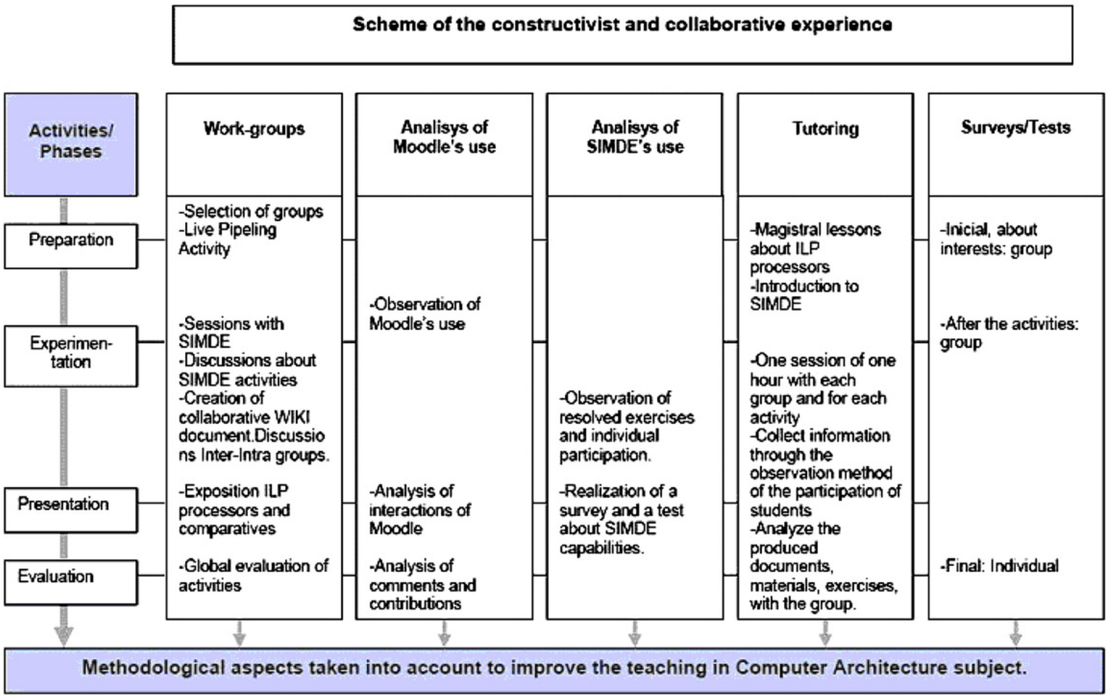

Scheme of the constructivist and collaborative experience
Activities/ Phases
Preparation
Experimen-tation
Presentation
Evaluation
Work-groups
-Selection of groups -Live Pipeling Activity
-Sessions with SIMDE -Discussions about SIMDE activities -Creation of collaborative WIKI document.Discussions Inter-Intra groups.
-Exposition ILP processors and comparatives
-Global evaluation of activities
Analisys of Moodle's use
-Observation of Moodle's use
-Analysis of interactions of Moodle
-Analysis of comments and contributions
Analisys of SIMDE's use
-Observation of resolved exercises and individual participation.
-Realization of a survey and a test about SIMDE capabilities.
Tutoring
-Magistral lessons about ILP processors -Introduction to SIMDE
-One session of one hour with each group and for each activity -Collect information through the observation method of the participation of students -Analyze the produced documents, materials, exercises, with the group.
Surveys/Tests
-Inicial, about interests: group
-After the activities: group
-Final: Individual
Methodological aspects taken into account to improve the teaching in Computer Architecture subject.

Fig. 2. Categories of analysis in the methodological approach.

and Patterson (2003) and from *Advanced Computer Architectures: A Design Space Approach* by Sima, Fountain, and Kacsuk (1997). In these debates students make a *negotiation of meaning* of ILP architecture concepts, first event of the students' construction.

- These debates are complemented with other activities such a "Living Pipeline", where some students play the role of different parts of a pipeline when executing a hypothetical program. This type of activities has been successfully applied to Computer Architecture teaching (Kris et al., 2004), taking advantage of the fact that students remember what they do more than what they listen to. In this activity, the rest of the class tries to deduce the effects on the pipeline of a new instruction. This is a first approximation to some circumstances such as bubbles or data dependence. As stated above, the professor plays a role of mere moderator, encouraging the negotiation among the students and trying to reduce their difficulties when the number of functional units in the architecture increases. This way, the student's understanding of those concepts improves considerably and some of the misconceptions described in the introduction are removed. If this first phase was not carried out, it would be nearly impossible that the student were able to get a proper knowledge about the ILP processing.
- Several auto-evaluation tests about specific concepts shown in the procedures were provided to the students. In these tests, each student was provided with the opportunity of access to the right answer as a complete explanation. This way, prompt feedback, as mentioned in principle 4, is given.
- The evaluation of these activities implied several factors: an individual proof, a one-hour session with each group covering several aspects shown with the simulator, a written report about the use of the simulator in the problem solving, the presentation in class about the assigned

commercial ILP architecture, how the student has taken part in the discussions, the monitoring of the interactions and the register analysis in the Moodle platform.

(2) How have we created the learning context in order to obtain a meaningful learning?
Materials and pedagogical tools alone do not warranty the knowledge construction. It is necessary to create an environment which allows the social interaction, the right use of media and the experimentation. For this reason, we have carried out the following actions:

- Several discussion groups were created. These groups were active during all the academic year: 45 students divided into 15 groups of there students. These discussion groups worked by themselves on specific concepts about current and commercial architectures and with other groups so that they were able to compare with other computer architectures, with similar characteristics and belonging to the same generation.
- These discussion groups worked on concrete problems about current and commercial architectures, for a later transfer when comparing with other computer architectures, with similar characteristics and belonging to the same generation. Each group has focused its work on a concrete architecture, comparing it with the other proposed machines: ITANIUM, HP PA-RISC 8000, Emotion Engine (PS2), AMD 64 OPTERON, Hitachi SH-4 (Dreamcast), Sun UltraSPARC III, MIPS R12000, Centrino, SGI Origin 2000 and 3000, Hyperthreading Intel Xeon, SG Origin 2000 and 3000, IBM G3–G5 and Alpha 21264. Students are encouraged to look for information (from journals like *IEEE Micro* or from the Internet) about their assigned architecture for a week and prepare a brief Powerpoint-like presentation of about 30–40 min. Once every group has presented their research results, the students should compare their assigned architecture with the rest. In this way, the students themselves carry a sort of review by peers, covering the principle 4 of Chickering and Gamson. After that collaboration, students are encouraged to cooperate to generate a high-quality document that will be used by other students in the next courses, covering this way the principle 6 in the Chickering and Gamson list.
- The individual and group works were carried out in an open learning environment, where Moodle e-learning platform and the use of SIMDE simulator allow the adaptation to different student learning methods and places as stated in principle 7. The assigned problems were solved using the simulator through the analysis of the clock cycles obtained from the different implementations and optimization techniques adapted to VLIW and superscalar architectures. By doing so, the simulator becomes a tool that makes the experimentation and knowledge construction easier. In previous experiences, the authors have realized that the success of discussion activities like "Living Pipeline" are often conditioned by two factors: lack of time and a sense of "fear to fail" in some students that do not take part in the discussions. The impact of both factors can be reduced with the use of pedagogical tools such as Moodle.
- The professors have played the role of mediator in the learning process. They set out the guidelines in the generation of the documents that the students should present in class as the exercises developed in the simulator. They solved any doubt and formulated some questions that helped the students to get a deeper knowledge about the concerned aspects.
- Furthermore, the professors, because of their large experience in teaching activities, have a background about the previous ideas and misconceptions that the students usually present. This background allows them to define proper activities to be carried out.

(3) How have we implemented the collaborative learning?

Collaborative learning was reached through inter-group and intra-group interactions consisting of discussion, reflection and making of decisions using both e-learning platform and class. The generation of the document was carried out in both inter-group and intra-group interactions. On the other hand, SIMDE activities only needed intra-group interaction. The five collaborative elements were applied as follows (González, González, Muñoz, & Sigut, 2005; Harvey, 2001):

- *Positive interdependence:* This is the core element. The students need each other in order to complete the group's task successfully – they "sink or swim" together. In this experience, the professors remarked the double responsibility: individual and from the group. In the same way, they encouraged discussion among groups.
- *Face to face interaction:* If the group wants to complete its task, the students need face to face interaction with the rest of the members. In the experience, they had to work together sharing resources and helping each other.
- *Individual accountability*: The individual accountability was reached through the assignation of different individual profiles to each member of the group. The proposed roles in the activity of generation of the ILP commercial architecture document were:
  - *Manager:* He/she should monitor the progress and efficiency of the team, limit the work and control that the tasks are carried out on time. He/she should carry out a task of coordination with the managers of the other groups about the comparative analysis among the different architectures.
  - *Scavenger:* The student that plays this role is responsible for looking for technical data (in the Internet, data sheets, reference books and magazines – *IEEE Micro, IEEE Transactions on Computers, etc.*) referred to the assigned computer architecture.
  - *Filter:* From the information supplied by the scavenger, this student should select the most significant one, organize it and draw up a document, according to what the group had set.

  On the other hand, the roles proposed in the SIMDE procedure set were chosen according to the optimization strategies that were used in the different problems to solve, i.e. dot product, DAXPY and linked list or vectors. Each group chose one of these problems and applied different optimization techniques. Each member of the group was responsible for the application of one strategy at least. These roles will be showed with more detail in Section 4.
- *Interpersonal and small group skills:* Social skills are essential for an effective collaborative work. Students should get involved simultaneously in the assigned task and the work in a team. In the proposed set of procedures, these skills are stimulated through the creation of teams with common interests and the promotion of their identity as group.
- *Group processing:* Finally, the evaluation of the group processing was carried out by students, members of the group and other groups. The documents generated by the groups were presented in class and published in the web page of the subject, as it was described above. Thus, the students themselves revised the task and identified (helped by the professor and their peers) their weak points as their main contributions. Each group presented a summary of the course, using the common document and other resources (specially designed for the presentation). The students were encouraged to present practical examples, illustrating the theoretical concepts remarked in class. Looking for the *positive interdependence* and the stimulation of the

Table 1
Task notebook of a groups indicating the type of interaction (I: individual, G: group, OG: other groups, P: professor) and the activity (description and features) of each task

| Tasks | Task notebook | | | | | |
|---|---|---|---|---|---|---|
| | Interaction | | | | Activity | |
| | I | G | OG | P | Description | Characteristics |
| 1. SIMDE | X | X | | | Problem solving | • Discovery learning<br>• Experimentation<br>• Simulator as mediator<br>• Contact with powerful ideas as objects to extrapolation and appropriation |
| 2. Discussion SIMDE | | X | | X | Explanation of proposals developed by the group to the professor | • Social construction of knowledge through the interactions teacher–learner–learner–teacher |
| 3. Preparation of the presentation about current commercial machines: WEBQUEST and classic bibliography search | X | X | X | | Search guided by objectives | • Social construction of knowledge through the interactions inter-group and intra-group<br>• Discovery learning |
| 4. Presentation and discussion about commercial current machines | X | X | X | X | Exposition of the assigned machine synthesis and comparison with others. | • Contact with powerful ideas as objects to extrapolation and appropriation |
| 5. WIKI of ILP concepts in current machines | X | X | | | Production of a collaborative document about concepts and characteristics of analyzed machines. | • Transfer and synthesis of studied concepts in order to consolidate the learning |
| 6. WIKI analysis | X | X | X | X | Discussion mediated by the teacher who observes common points and differences among analyzed machines | • Analysis, thoughts and learning about possible mistakes or misconceptions |

interaction, students were offered sufficient references where to collect information. These students should at least answer some questions formulated in the proposed work-guide. Furthermore, they were encouraged to look for technical data in the supplied documents, offering their information to other groups and discussing among the members of the group and with other groups. For these communication processes, the students should use the e-learning platform, in particular a chat and a forum, specially created for this activity.

In order to understand the collaborative part of the proposed procedure, we are going to analyze the interactions among the tasks and their characteristics in a notebook belonging to a particular group (Table 1).

As we can observe in our methodological approach, the role of SIMDE simulator as learning's mediator is essential to improve the educational experience in the concepts about ILP's processors. So, in the next section we will focus on the pedagogical activities carried out with this tool.

## 4. Improving the educational experience through SIMDE

The work proposed with SIMDE consists of three 2-h practical sessions in class and an exercise by group. The contents of each session and the exercise are described in the following sections.

### 4.1. Superscalar example

The objectives of the first session are as follows:

(1) To familiarize the students with the simulator environment and its basic functionalities.
(2) To consolidate students' knowledge about superscalar processors.

This session is focused on clarifying how Tomasulo's algorithm works and the way the instructions flow by the superscalar components: ROB, reservation stations, etc. Hence, it is fundamental to have explained all of these topics previously in theoretical sessions.

Firstly, a simple sequential code that represents a single loop example is loaded (see Example 1). This code adds a constant to each component from a 16-element array. The result is placed in another array.

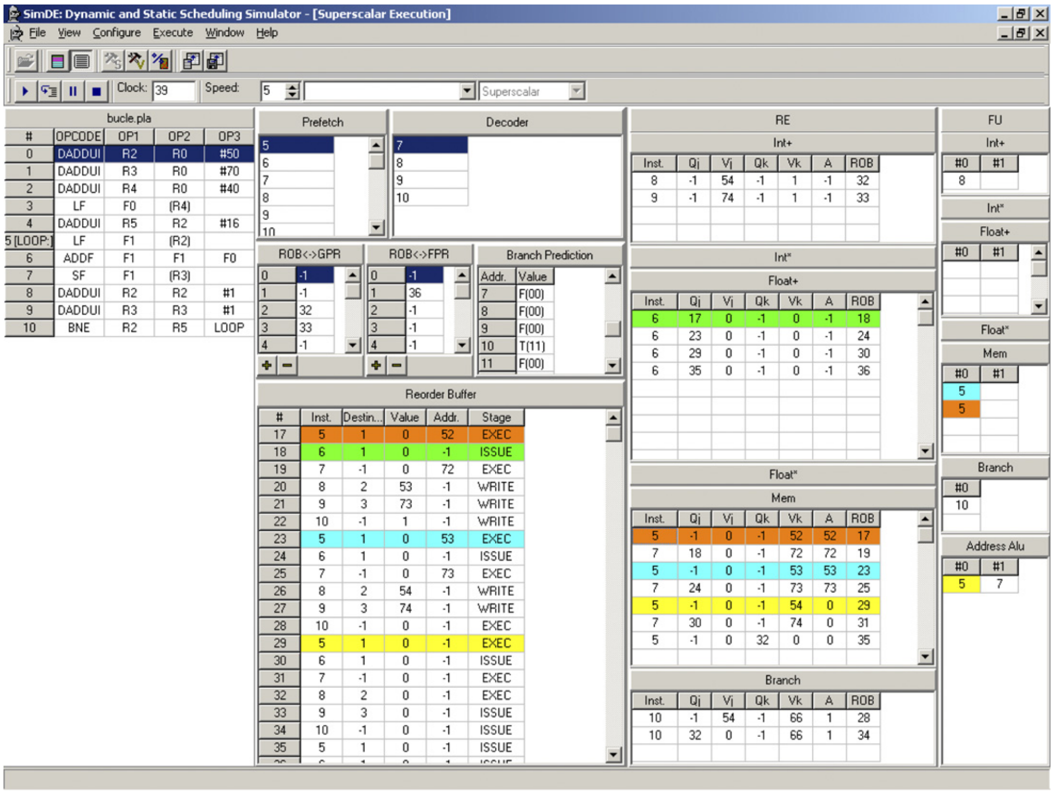
This example requires that the students fill appropriately the memory before they start the simulation by putting a constant value at memory address 40, and 16 floating point values from memory address 50–65.

The buttons at the execution toolbar allow the users to control the simulator. The students can start, pause, stop and restart the simulation, and they can check the amount of clock cycles passed from its initiation. The superscalar execution window shows all of the processor components: ROB, reservation stations, functional units, etc., in a way that the instruction flow is remarked (Fig. 3). The students can even colour some selected instructions to track them easily through their execution cycle. Therefore, some obscure details about Tomasulo's algorithm are clarified.

The students can check the clock at execution toolbar after each experiment to verify on their own the importance of the branch miss predictions or the influence of cache misses over the execution performance. The issue rate or the number and latency of FUs are also student-customizable parameters. These parameters can be modified to extract conclusions about the design of a superscalar processor, but the teacher should emphasize the cost of adding new FUs.

```
DADDUI R2 R0 #50
DADDUI R3 R0 #70
DADDUI R4 R0 #40
LF F0 (R4)
DADDUI R5 R2 #16
LOOP:
LF F1 (R2)
ADDF F1 F1 F0
SF F1 (R3)
DADDUI R2 R2 #1
DADDUI R3 R3 #1
BNE R2 R5 LOOP
```

Example 1. Sample code (loop.pla).


SimDE: Dynamic and Static Scheduling Simulator - [Superscalar Execution]
File View Configure Execute Window Help

Clock: 39 Speed: 5 Superscalar

bucle.pla

| # | OPCODE | OP1 | OP2 | OP3 |
|---|---|---|---|---|
| 0 | DADDUI | R2 | R0 | #50 |
| 1 | DADDUI | R3 | R0 | #70 |
| 2 | DADDUI | R4 | R0 | #40 |
| 3 | LF | F0 | (R4) | |
| 4 | DADDUI | R5 | R2 | #16 |
| 5 [LOOP:] | LF | F1 | (R2) | |
| 6 | ADDF | F1 | F1 | F0 |
| 7 | SF | F1 | (R3) | |
| 8 | DADDUI | R2 | R2 | #1 |
| 9 | DADDUI | R3 | R3 | #1 |
| 10 | BNE | R2 | R5 | LOOP |

| Prefetch |
|---|
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

| Decoder |
|---|
| 7 |
| 8 |
| 9 |
| 10 |

ROB<->GPR

| | |
|---|---|
| 0 | -1 |
| 1 | -1 |
| 2 | 32 |
| 3 | 33 |
| 4 | -1 |

ROB<->FPR

| | |
|---|---|
| 0 | -1 |
| 1 | 36 |
| 2 | -1 |
| 3 | -1 |
| 4 | -1 |

Branch Prediction

| Addr. | Value |
|---|---|
| 7 | F(00) |
| 8 | F(00) |
| 9 | F(00) |
| 10 | T(11) |
| 11 | F(00) |

Reorder Buffer

| # | Inst. | Destin... | Value | Addr. | Stage |
|---|---|---|---|---|---|
| 17 | 5 | 1 | 0 | 52 | EXEC |
| 18 | 6 | 1 | 0 | -1 | ISSUE |
| 19 | 7 | -1 | 0 | 72 | EXEC |
| 20 | 8 | 2 | 53 | -1 | WRITE |
| 21 | 9 | 3 | 73 | -1 | WRITE |
| 22 | 10 | -1 | 1 | -1 | WRITE |
| 23 | 5 | 1 | 0 | 53 | EXEC |
| 24 | 6 | 1 | 0 | -1 | ISSUE |
| 25 | 7 | -1 | 0 | 73 | EXEC |
| 26 | 8 | 2 | 54 | -1 | WRITE |
| 27 | 9 | 3 | 74 | -1 | WRITE |
| 28 | 10 | -1 | 0 | -1 | EXEC |
| 29 | 5 | 1 | 0 | -1 | EXEC |
| 30 | 6 | 1 | 0 | -1 | ISSUE |
| 31 | 7 | -1 | 0 | -1 | EXEC |
| 32 | 8 | 2 | 0 | -1 | EXEC |
| 33 | 9 | 3 | 0 | -1 | ISSUE |
| 34 | 10 | -1 | 0 | -1 | ISSUE |
| 35 | 5 | 1 | 0 | -1 | ISSUE |

RE

Int+

| Inst. | Qj | Vj | Qk | Vk | A | ROB |
|---|---|---|---|---|---|---|
| 8 | -1 | 54 | -1 | 1 | -1 | 32 |
| 9 | -1 | 74 | -1 | 1 | -1 | 33 |

Int*

Float+

| Inst. | Qj | Vj | Qk | Vk | A | ROB |
|---|---|---|---|---|---|---|
| 6 | 17 | 0 | -1 | 0 | -1 | 18 |
| 6 | 23 | 0 | -1 | 0 | -1 | 24 |
| 6 | 29 | 0 | -1 | 0 | -1 | 30 |
| 6 | 35 | 0 | -1 | 0 | -1 | 36 |

Float*

Mem

| Inst. | Qj | Vj | Qk | Vk | A | ROB |
|---|---|---|---|---|---|---|
| 5 | -1 | 0 | -1 | 52 | 52 | 17 |
| 7 | 18 | 0 | -1 | 72 | 72 | 19 |
| 5 | -1 | 0 | -1 | 53 | 53 | 23 |
| 7 | 24 | 0 | -1 | 73 | 73 | 25 |
| 5 | -1 | 0 | -1 | 54 | 0 | 29 |
| 7 | 30 | 0 | -1 | 74 | 0 | 31 |
| 5 | -1 | 0 | 32 | 0 | 0 | 35 |

Branch

| Inst. | Qj | Vj | Qk | Vk | A | ROB |
|---|---|---|---|---|---|---|
| 10 | -1 | 54 | -1 | 66 | 1 | 28 |
| 10 | 32 | 0 | -1 | 66 | 1 | 34 |

FU

Int+

| #0 | #1 |
|---|---|
| 8 | |

Int*

Float+

| #0 | #1 |
|---|---|

Float*

Mem

| #0 | #1 |
|---|---|
| 5 | |
| 5 | |

Branch

| #0 |
|---|
| 10 |

Address Alu

| #0 | #1 |
|---|---|
| 5 | 7 |

Fig. 3. Superscalar execution.

At the end of this session the students are encouraged to write its own sequential codes and test them over different superscalar configurations.

## 4.2. VLIW example

In the former session, the students had to deal with the superscalar processor. The second session is focused on the VLIW processor. Before the students start this session it is necessary for them to have learnt about the hardware resources and the software techniques related to VLIW processors in the theoretical classes.

A VLIW code can be created from scratch using as basis a sequential one or it can be loaded and then customized by the students. At this stage, it is advisable for the students to start loading a pre-created code. For this purpose, the "loop.vliw" file can be used together with the "loop.pla" example seen in the previous session. When the students load this sample code, the VLIW code window is shown.

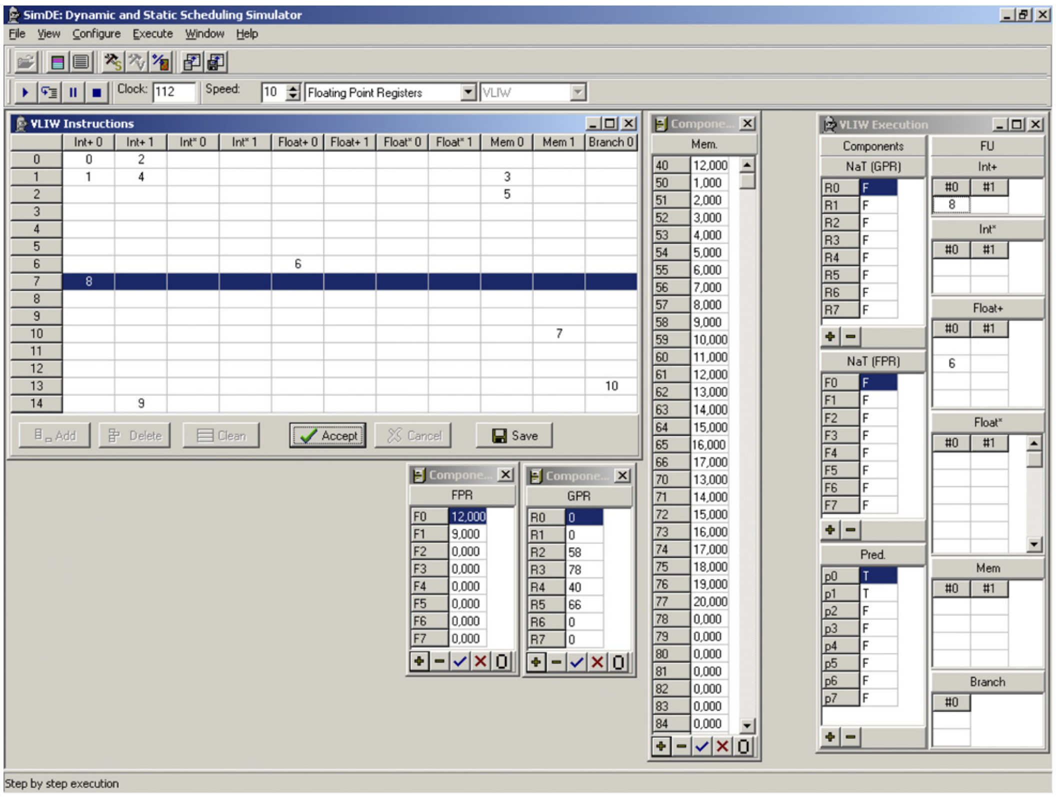
The VLIW code window is a table where columns represent the different FUs and rows represent the execution-ordered long instructions. The numbers in cells correspond with the identifiers of the sequential instructions. The code can be modified by the students by deleting any long instruction or operation, or by adding a new one. The students can also set a predicate register for each operation or modify any branch operation behavior (branch target and predication).

The students should notice that the instruction memory allocated for this code is much bigger than the corresponding sequential one. Since the long instruction word provides a field for each available FU and not all of the fields are used, there is a lot of wasted memory space.

The execution of the proposed example on the VLIW processor allows the teacher to remark its hardware simplicity. The VLIW execution window is much simpler than the superscalar one. It only consists of the FUs, the predicate registers and the NaT bits, as shown in Fig. 4.

The corresponding example executes in 212 clock cycles opposite to the 72 clock cycles obtained using the superscalar processor. Since the original sequential code consists of a few operations, it is not possible to obtain any performance improvement by changing the number of FUs in the processor parameters.

At this point, the students should be encouraged to reason about this low performance to deal with principle 3 of Chickering and Gamson. The logical consequence must be that no more ILP can be obtained from this sequential code. Then, the teacher can propose to rewrite the sequential code by applying loop unrolling, software pipelining and other software techniques



Clock: 112 Speed: 10 Floating Point Registers VLIW

VLIW Instructions

| | Int+ 0 | Int+ 1 | Int* 0 | Int* 1 | Float+ 0 | Float+ 1 | Float* 0 | Float* 1 | Mem 0 | Mem 1 | Branch 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | | | | | | | | | |
| 1 | 1 | 4 | | | | | | | 3 | | |
| 2 | | | | | | | | | 5 | | |
| 3 | | | | | | | | | | | |
| 4 | | | | | | | | | | | |
| 5 | | | | | | | | | | | |
| 6 | | | | | 6 | | | | | | |
| 7 | 8 | | | | | | | | | | |
| 8 | | | | | | | | | | | |
| 9 | | | | | | | | | | | |
| 10 | | | | | | | | | | 7 | |
| 11 | | | | | | | | | | | |
| 12 | | | | | | | | | | | |
| 13 | | | | | | | | | | | 10 |
| 14 | | 9 | | | | | | | | | |

| Mem. | |
|---|---|
| 40 | 12,000 |
| 50 | 1,000 |
| 51 | 2,000 |
| 52 | 3,000 |
| 53 | 4,000 |
| 54 | 5,000 |
| 55 | 6,000 |
| 56 | 7,000 |
| 57 | 8,000 |
| 58 | 9,000 |
| 59 | 10,000 |
| 60 | 11,000 |
| 61 | 12,000 |
| 62 | 13,000 |
| 63 | 14,000 |
| 64 | 15,000 |
| 65 | 16,000 |
| 66 | 17,000 |
| 70 | 13,000 |
| 71 | 14,000 |
| 72 | 15,000 |
| 73 | 16,000 |
| 74 | 17,000 |
| 75 | 18,000 |
| 76 | 19,000 |
| 77 | 20,000 |
| 78 | 0,000 |
| 79 | 0,000 |
| 80 | 0,000 |
| 81 | 0,000 |
| 82 | 0,000 |
| 83 | 0,000 |
| 84 | 0,000 |

| FPR | |
|---|---|
| F0 | 12,000 |
| F1 | 9,000 |
| F2 | 0,000 |
| F3 | 0,000 |
| F4 | 0,000 |
| F5 | 0,000 |
| F6 | 0,000 |
| F7 | 0,000 |

| GPR | |
|---|---|
| R0 | 0 |
| R1 | 0 |
| R2 | 58 |
| R3 | 78 |
| R4 | 40 |
| R5 | 66 |
| R6 | 0 |
| R7 | 0 |

Components

| NaT (GPR) | |
|---|---|
| R0 | F |
| R1 | F |
| R2 | F |
| R3 | F |
| R4 | F |
| R5 | F |
| R6 | F |
| R7 | F |

| NaT (FPR) | |
|---|---|
| F0 | F |
| F1 | F |
| F2 | F |
| F3 | F |
| F4 | F |
| F5 | F |
| F6 | F |
| F7 | F |

| Pred. | |
|---|---|
| p0 | T |
| p1 | T |
| p2 | F |
| p3 | F |
| p4 | F |
| p5 | F |
| p6 | F |
| p7 | F |

FU

| Int+ #0 | Int+ #1 |
|---|---|
| 8 | |

| Float+ #0 | Float+ #1 |
|---|---|
| | |
| 6 | |
| | |
| | |

Int*, Float*, Mem (#0, #1) and Branch (#0): empty

Step by step execution

Fig. 4. VLIW execution.

for code optimization, giving prompt feedbacks as it is proposed in principle 4 of Chickering and Gamson.

The rest of the session can be focused on a two-step methodology: the students apply optimizations to the sequential code (loop unrolling, software pipelining, instructions reordering, etc.) and then they schedule the sequential operations over the long instruction words. Loop unrolling can be used to illustrate the influence of code optimizations over the VLIW processor performance, as seen in Table 2.

Lastly in this session, it can be included the possibility of changing the VLIW structure by adding new FUs. Thus, the performance of the unrolled codes could be improved (i.e. the eight-unrolled iterations example executes in 30 clock cycles by using eight floating point adders and eight memory units). The emphasis can be placed on how much cheaper is to add a new FU to a VLIW processor than a superscalar one.

### 4.3. Advanced features of the VLIW simulator

The objectives of the last session are as follows:

(1) To remind the students about all the simulator features seen in the former sessions.
(2) To use advanced features such as predication.

Table 2
Compared performance of loop unrolling

| Iterations unrolled | Clock (superscalar) | Clock (VLIW) |
|---|---|---|
| 1 (single) | 72 | 212 |
| 2 | 68 | 108 |
| 4 | 61 | 60 |
| 8 | 43 | 36 |

```
DADDUI R10 R0 #10
DADDUI R1 R0 #0
DADDUI R2 R0 #1
LF F1 0(R10)
LF F2 1(R10)
BNE R32 R1 A
ADDF F3 F1 F0
BEQ R0 R0 FIN
A:
BNE R32 R2 B
ADDF F3 F2 F0
BEQ R0 R0 FIN
B:
ADDF F3 F2 F1
FIN:
SF F3 2(R10)
```

Example 2. Sequential code for illustrating predication.

Example 2 is a code that loads two floating point values from memory. The value in the register R32 indicates one of three possible values to be placed on a memory location: the first loaded value, the second one or the addition of both values.

This code scheduling requires that the students apply predication. A deeper knowledge of the simulator is needed to build an optimized code. Some aspects must be considered now such as the operation latencies and how the simulator handles predication. This code has been scheduled by assuming a two-cycle latency for branch operations. Regarding predication, any operation that reaches the top of a FU pipeline with a false predicate is cancelled. The simulator design ensures that branches are solved at the beginning of the cycle, while the evaluation of predicates is left as a final stage.

## 4.4. Selected exercise

Each group is provided with a sufficiently complex sequential code. These codes include well-known algorithms such as the dot product or DAXPY, and a few codes that operate with linked lists or vectors.

The exercise consists of:

(1) Studying the sequential code. The students must look for inputs that produce worst and best cases.
(2) Simulating this code on the superscalar processor. The students should test different inputs and different processor parameters in order to obtain the best performance.
(3) Applying software techniques for code optimization over the sequential code.
(4) Scheduling and simulating each of the optimized codes on the VLIW processor. Again, it is interesting to test different inputs and processor parameters.
(5) Comparing results. The results from both processors must be compared and the students should extract conclusions about performance. The students also should weigh up the relative costs of the processor design and the effort invested in playing the compiler role for the VLIW processor.

Once the students have finished, all their conclusions and results are gathered into a report. They are also asked to fill in a questionnaire about the tool in order to assess their experience and the simulator itself.

## 4.5. Feedback from the students

The statistical results that are described herein correspond essentially to the tracking made of using of the simulator and its efficiency as pedagogical tool.

Feedback from students is a priority task in order to verify the usefulness of the proposed exercises (Harvey, 2001). Some lacks or potential improvements of the simulator are obtained in this way too.

The simulator has been used during the last two years by students of last year of a Computer Engineering degree from University of La Laguna. The students are asked to fill in a questionnaire mentioned in the section before. This questionnaire is basically focused on three main aspects:

(1) The suitability of the simulator for the educational requirements from students.
(2) The features and functionality of the software.
(3) The technical aspects of the simulator: malfunctions, bugs, ...

There is an important difference between the results of the first and the second year. The simulator has been used by about 30 students the first year (SIMDE v. 1.0), and 45 the second one (SIMDE v. 1.2). The first year the authors were looking for a technical validation of the software and the educational value was left as a secondary issue. Thus, the students of the first year were used basically as beta-testers after an introductory session with the simulator. They were only asked to test the tool by using it at their discretion. The results of this year allowed to fix most of the detected bugs and to add several improvements to the simulator, but it was clear that the students needed a more structured procedure set to deal correctly with the simulator and improve their educational experience.

The second year the authors were able to focus on the educational aspects of the tool since the technical aspects had been sufficiently tested. Consequently, they designed the procedure set described in this paper and got a notorious improvement of the results. The dedication of the students during the first and second year is compared in Fig. 5. It is clear that students of the second year dedicated much more time in testing the simulator than the first year ones.

The main differences are referred to the interface validation. Several improvements were made to the simulator from the first year to the second one. The dedication of the students and a better scheduling of exercises and sessions are the fundamental influences in these results.

The students of the second year were able to understand more easily the contents of both processors, even when these contents had no changes from one year to the other.

Anyway, students of both years judged positively the teaching role of SIMDE. Their answers suggested that the simulator is a helpful tool to understand theoretical contents of ILP as seen in Fig. 6a.

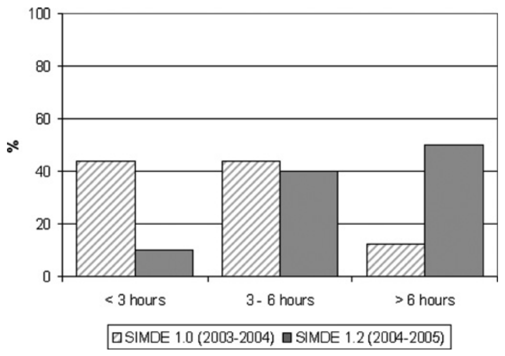


Fig. 5. Students' dedication to the simulator (results are showed as percentages for comparison purposes in this figure and successive).

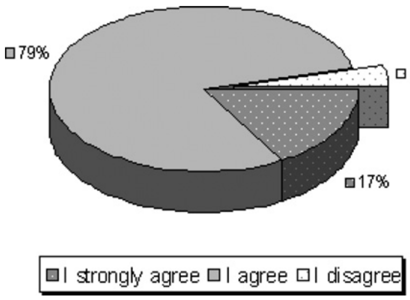


Fig. 6a. SIMDE helps to understand theoretical concepts.

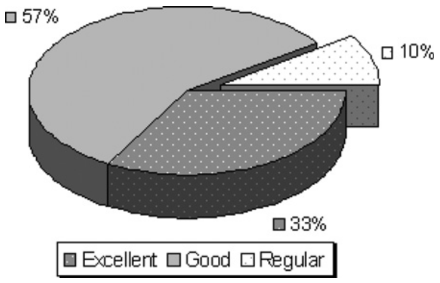


Fig. 6b. Students' opinion about the help facility.

The help facility provided by the application was well valuated too, since it correctly illustrates an overview of all the theoretical contents (see Fig. 6b). According to the students, this facility allows them to quickly understand the application functionalities. Some students criticized the short review of theoretical concepts mentioned above because they thought these topics needed to be treated in greater insight. In spite of this opinion, the authors consider that the application should be used as an assistant tool for Computer Architecture courses, and not as a substitute for books or theoretical classes.

As a complement of the validation tasks, the simulator was presented in an educational conference (Castilla, Moreno, Sigut, González, & González, 2004) and more opinions and suggestions were collected from several users that were able to test the simulator.

## 5. Evaluation framework of methodology

The mixed evaluation method used in the procedure is based on the following analysis categories and data collection (Table 3):

In the procedure presented here, three modes of analysis are combined: quantitative, applied to closed questions of quizzes; qualitative, obtained by the qualitative data sources (open quizzes, observations, group interviews) and social network analysis, with inputs of data from quizzes, observations and activity registers of Moodle. Fig. 2, presented above, shows the categories of analysis in the methodological approach organized in activities and phases.

Table 3
Analysis categories and data collection (Mixed Evaluation Method)

| Technique | Instrument |
|---|---|
| *Data collection* | |
| Observation | • Teacher notebook<br>• Map of interactions |
| Automatic registration | • Black box (Moodle) |
| Interview | • Discussion group |
| Quiz | • Group survey<br>• Individual test |
| **Method** | **Instrument** |
| *Analysis categories* | |
| Quantitative | • Closed quizzes<br>• Automatic registers |
| Qualitative | • Open quizzes<br>• Observations<br>• Interviews |
| Social networks | • Observations of face to face relations<br>• Interactions in Moodle |

The results obtained this academic year using the proposed methodology have improved significantly with respect to the ones obtained in the first year where only the simulator was used. The students have shown a deeper knowledge as well as a higher level of motivation. This motivation is shown when analyzing the course marks. The 85 per cent of the students have passed the course and the 60 per cent of these students have got A and B qualifications.

## 6. Conclusions and future work

A constructivism and collaborative methodological approach applied for teaching ILP Architectures to improve the learning process has been presented.

The described methodology has been tested by students from last year of a Computer Engineering degree in University of La Laguna.

The key aspects of the presented approach follow the principles of constructivism and collaborative pedagogical theories and are supported by two technological tools: an e-learning platform (Moodle) and a simulator of ILP architectures (SIMDE). So, a collaborative virtual learning environment, which includes this simulator and other learning resources, has been designed and developed.

The e-learning platform was selected because it satisfies the theoretical principles, allows us to implement an efficient communication among students and teachers, and offers a wide diversity of resources and activities.

A simulation tool of ILP architectures, covering dynamic (using a superscalar processor) and static scheduling (using a VLIW processor) has been specially designed and incorporated into the teaching strategy. The simulator offers a valuable tool for teaching ILP architectures in a way that no other available simulators can supply. The main achievement is that students can compare static scheduling versus dynamic scheduling using the same basic structure, components and instruction set.

The results from tests showed in the paper have been collected during the last two years and they have confirmed the utility of SIMDE as an educational tool to support teaching of ILP architectures. The first year the authors focused on a technical validation of the tool but last year the authors have been able to design a complete procedure set that improves the educational experience of the students.

With respect to the learning experience with the proposed methodology, the students have shown a deeper knowledge as well as a higher level of motivation, difficult to find in this type of courses.

The authors are currently working on adding new features to the simulator, such as more branch prediction schemes or a trace cache. They are also working on increasing the instruction set in order to let the students design more example codes that illustrate the functionalities of ILP architectures.

Due to this great interest and the obtained results, this methodological proposal will be extended to the design of other theoretical-procedure activities in the current academic course.

In the same way, this methodological procedure is suitable to be used in other subjects with contents which require to explicit dynamic complex concepts. In this case, we suggest the use of simulators that allow to study the behaviour of the processes. Even in the case where simulators are not necessary, the methodological proposal can be used with new activities according to the contents. In fact, our methodology has been used in other subjects of the Computer Science degree, such as Operating Systems, Computer Structure and Human–Computer Interaction with excellent results. The use of computer tools has been different in each case but keeping the constructivism and collaborative proposal. A simulator for the teaching of hierarchy memory concepts in Operating Systems and Computer Structures subjects has been developed (González, Alesanco, Castilla, & Moreno, 2005). This simulator has been used in combination with an adaptive tests tool (González et al., 2005) to promote the discovery learning. In the case of Human–Computer Interaction, the methodological proposal was the same, however we did not use simulators, but we worked with constructivist and collaborative activities such as WEBQUESTs, WIKIs among others (González, González et al., 2005).

## References

Bruner, J. (1991). *Actos de significado*. Madrid: Alianza, Bruner.

Bruner, J. (1997). *La educación, puerta de la cultura*. Madrid: Visor.

Castilla, I., Moreno, L., Sigut, J., González, C., & González, E. J. (2004). SIMDE: Un Simulador para el Apoyo Docente en la Enseñanza de las Arquitecturas ILP con Planificación Dinámica y Estática. In *Proceedings of X Jornadas de Enseñanza Universitaria de la Informática (JENUI 2004)* (pp. 505–508).

Chickering, A., & Gamson, Z. (1987). Seven principles for good practice in undergraduate education. *American Association for Higher Education Bulletin*.

Crook, Ch. (1998). *Ordenadores y aprendizaje colaborativo*. Madrid: Morata.

Cuppu, V. (1999). *Cycle Accurate Simulator for TMS320C62x, 8 way VLIW DSP Processor*. Available from http://citeseer.ist.psu.edu/416529.html.

Denhiere, G., & Baudet, S. (1992). *Lecture, comprehension de texte et science cognitive* (pp. 214–232). Paris: Presses Universitaires de France.

Edler, J., & Hill, M. (1997). *Dinero IV: Trace-Driven Uniprocessor Cache Simulator*. Available from http://www.cs.wisc.edu/~markhill/DineroIV.

Felder, R. M., Felder, G. M., & Dietz, E. J. (1995). A longitudinal study of Engineering student performance and retention vs comparisons with traditional taugh students. *Journal of Engineering Education, 87*(4), 469–480.

Gifford, B., & Enyedy, N. D., (1999). Activity centered design: towards a theoretical framework for CSCL. In *Proceedings of the third international conference on computer support for collaborative learning*. [En red] Available from www.gseis.ucla.edu/faculty/enyedy/pubs/Giffcrd&Enyvedy_CSCL2000.pdf.

Gogoulou, M., Gouli, E., Grigoriadou, M., & Samarakou, M., (2003). Exploratory + collaborative learning in programming: a framework for the design of learning activities. ICALT 2003 (pp. 350–351).

Gogoulou, M., Gouli, E., Grigoriadou, M., & Samarakou M., (2003). Supporting collaboration and adaptation in a CSCL environment. ICALT 2003 (p. 470).

González, C., Alesanco, F., Castilla, I., & Moreno, L. (2005). SIJEM: Una herramienta didáctica para la enseñanza de la Jerarquía de Memoria. SIECI 2005. Orlando, Florida, USA. July.

González, C., González, E., Muñoz, V., & Sigut, J. (2005). Una Experiencia de Aprendizaje Colaborativo en la Universidad Ulilizando Wikis en Moodle. SIECI 2005. Orlando, Florida, USA. July.

González, C., Pérez, Z., D́ıaz, D., Medina, F., Alesanco, F., & Moreno, L. (2005). Portal Web y Personaje 3D para la evaluación de alumnos utilizando tests adaptativos bayesianos en XML. SIECI 2005. Orlando. Florida. USA. July.

Grigoriadou, M., Toula, M., & Kanidis, E. (2003). Design and evaluation of a cache memory simulation program. ICALT 2003 (pp. 170–174).

Harvey, L. (2001). Student feedback: a report to the higher education funding council for England. Available from http://www.uce.ac.uk/crq/publications/studentfeedback.pdf, October.

Hennesy, J. L., & Patterson, D. A. (2003). *Computer architecture: a quantitative approach* (third ed.). San Mateo, CA: Morgan Kaufmann.

Inaba, A., Supnithi, T., Ikeda, M., Mizoguchi, R., & Toyoda, J. (2000). How can we form effective collaborative learning groups? Available from http://www.ai.sanken.osaka-u.ac.jp/indexe.html.

Jonassen, D. (1999). Design of constructivist learning environments. Available from http://tiger.coe.missouri.edu/~jonassen/courses/CLE/.

Knuth, R. A., & Cunningham, D. J. (1991). Tools for constructivism. In T. M. Duffy & D. H. Jonassen (Eds.), *Constructivism and the technology of instruction: a conversation* (pp. 163–187). Hillsdale, NJ: Lawrence Erlbaum Associates.

Koschman, T. (1996). *Theory and practice of an emerging paradigm*. Mahwah: N.J. Lawrence Erlbaum.

Kris, D. (2004). Powers: teaching computer architecture in introductory computing: why? and how? Sixth Australasian computing education conference (ACE 2004), Dunedin, New Zealand, January 18–22 (Living CPU) (pp. 255–260).

López, P., & Calpé, R. (1998). WinDLXVSim. Available from <http://dlxv.disca.upv.es/tools/dlxv.html>.

Lipponen, L. (2002). Exploring foundations for computer-supported collaborative learning. CSCL 2002. Colorado Boulder. USA. January.

Martí, E. (1992). *Aprender con ordenadores en la escuela*. Barcelona: ICE-Universitat de Barcelona/Horsori.

Martinez, A., Dimitriadis, Y., Rubia, B., Gómez, E., & de la Fuente, P. (2003). Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Computers and Education*.

Osuna, C. (2000). DELFOS: a telematic and educational framework based on layer oriented to learning situations. PhD tesis, Universidad de Valladolid, Valladolid, España.

Scott, M. (2005). WinMIPS64. Available from <http://www.computing.dcu.ie/~mike/winmips64.html>.

Silhan, J., & Fuss, C. (1997). MIDAS Beta-1. Available from http://icaro.eii.us.es/descargas/R10kSim.zip.

Sima, D., Fountain, T., & Kacsuk, P. (1997). *Advanced computer architectures: a design space approach*. Harlow, England: Addison-Wesley.

Spiro, R. J., Feltovich, P. J., Jacobson, M. J., & Coulson, R. L. (1991). Knowledge representation, content specification, and the development of skill in situation-specific knowledge assembly: some constructvist issues as they relate to cognitive flexibility theory and hypertext. *Educational Technology, 31*(9), 22–25.

SYNERGO collaborative mapping environment. (2004). Available from http://www.ee.upatras.gr/hci/synergo/.

Tomasulo, R. M. (1967). An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal Research and Development, 11*(1), 25–33.

Vosniadou, S., & Kolias, V. (2003). Using collaborative, computer-supported, model building to promote conceptual change in science. In E. De Corete, L. Verschaffel, N. Entwistel, & J. Van Merrienboer (Eds.), *Powerful learning environments: unravelling basic components and dimensions. Advances in learning and instruction*. Elsevier Press.

Vygotsky, L. S. (1979). *El desarrollo de los procesos psicológicos superiores*. Barcelona: Grijalbo.

Wolff, M., & Wills, L. (2000). SATSim: A Superscalar Architecture Trace Simulator Using Interactive Animation. Available from http://www.ece.gatech.edu/research/pica/SATSim/satsim.html, June.