Euskal Herriko Unibertsitatea
Universidad del País Vasco

Konputagailuen Arkitektura eta Teknologia Saila
Departamento de Arquitectura y Tecnología de Computadores

eman ta zabal zazu

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

The Computer Input/Output Subsystem Education in an undergraduate introductory course: a Multiperspective Study

Dissertation Presented to the department of Computer Architecture and Technology in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in Computer Science by
Edurne Larraza-Mendiluze

PhD Advisor
Nestor Garay-Vitoria

Donostia, December 20, 2013

Servicio Editorial de la Universidad del País Vasco (UPV/EHU)
Euskal Herriko Unibertsitateko Argitalpen Zerbitzua (UPV/EHU)
University of the Basque Country - Editorial Service (UPV/EHU)
ISBN: 978-84-9082-010-0

Laburpena

Tesi hau informatikaren irakaskuntzaren ikerkuntza lerroaren barruan kokatzen da. Oinarri gisa, teknologia, pedagogia eta edukien ezagutza eredua (TPACK model, bere ingeleseko sigletatik) hartzen du. Ikerketa honetan, aipatutako ereduaren osagai bakoitza ikuspuntu desberdin gisa erabilia izan da konputagailuaren S/I azpisistemaren irakaskuntza aztertzeko.

Edukien ezagutza osagaiaren ikuspuntutik, testuliburuak, unibertsitateetako programak eta ikerkuntza bibliografia aztertu dira eta gaia irudikatzeko hurbilpen bat baino gehiago badaudela aurkitu da.

Teknologiaren ezagutza osagaiaren ikuspuntutik, Nintendo DS makinaren S/I azpisistemaren funtzionamendua aztertu da.

Pedagogiaren ezagutza osagaiaren ikuspuntutik, proiektuetan oinarritutako ikaskuntza eta beste hainbat metodologia aktibo ikasi egin dira ondoren aplikatu ahal izateko.

Txosten honek azaltzen du nola konbinatu diren osagai horietako ezagutzak bestelako ezagutzak sortzeko: **teknologia eta edukien** ezagutza, **teknologia eta pedagogia** ezagutza, **pedagogia eta edukien** ezagutza, eta azkenik, **teknologia, pedagogia eta edukien** ezagutza. Ezagutza berri hauekin konputagailuaren S/I azpisistemarentzako hezkuntza ingurune eraginkor bat sortzeko asmoz. Horretarako pauso hauek jarraitu dira:

- Edukiak irudikatzeko hurbilpen bat aukeratzea,
- Nintendo DS makina eduki zehatza lantzeko prestatzea eta
- Proposatutako teknika pedagogikoen baliozkotzea.

TPACK ereduan nolabait ezkutatuta baldin badago ere, hezkuntzarako eredu bat den neurrian, ikasleak ere hartzen ditu ikuspuntu gisa. Eta lan hau TPACK ereduan oinarritzen den neurrian, ikasleak ere hartzen ditu ikuspuntu gisa, eta haien ezagutza aztertu egiten du horretarako.

Lan honek informatikaren irakaskuntzaren ikerkuntza lerroan ate asko irekitzen ditu. Izan ere, informatikaren irakaskuntzaren ikerkuntzak konputagailuen S/I aspisistemari ez baitio arreta handirik ipini.

Abstract

This dissertation is framed in the Computing Education Research area. As background, the Technological, Pedagogical and Content Knowledge framework (TPACK model) has been used. Each construct of the mentioned framework serves as a different study perspective, where the topic analysed is the computer I/O subsystem.

From the **content knowledge** perspective, textbooks, university syllabi and research bibliography have been analysed and different approaches found.

From the **technological knowledge** perspective, the I/O subsystem of the Nintendo DS machine has been studied.

From the **pedagogical knowledge** perspective, project-based learning (PBL), and other active learning methodologies have been learned for its latter application.

The dissertation explains how these constructs have been combined into **technological content** knowledge, **technological pedagogical** knowledge, **pedagogical content** knowledge, and finally **technological, pedagogical and content** knowledge in order to define an effective educational environment for the computer I/O subsystem topic. The following steps are followed:

- The selection of a content representation approach,
- The preparation of the Nintendo DS machine for its use this the specific content, and
- The validation of the proposed pedagogical techniques.

Although somehow hidden, the TPACK model, as an educational framework, includes the student as a perspective, and also does this work, with the analysis of students' knowledge.

This work opens many doors in the field of Computing Education Research towards the computer I/O subsystem, which has been barely payed attention.

Eskaintza

Familiari

Zentzurik zabalenean,
zaretenoi eta izan gabe zaudetenoi,
izandakoei eta izango direnei,
tarte batzuetan egon ez banaiz ere,
egongo naizelako.

Muxu.

A la familia

En el sentido más amplio,
A quienes sois y a quienes sin serlo estáis,
A quienes fueron y a quienes serán,
Porque, aunque a veces no he estado,
Estaré.

Muxu.

Eskerrak

Azkenean ez da ba hau izango tesiko atalik zailena? Nik den denak eskertu nahi zaituztet. Inguruan zaudeten guztiek izan duzue nolabaiteko eragina tesi honen garapenean, animoak emanez besterik ez bada ere, eta hori eskertzekoa da. Hala ere, izenak jarri behar dira, eta zerrenda ez da motza inondik inora.

Lehenik, esker mila Nestor. Zuk onartu zenuen nere tesia zuzentzea nik planteatzen nuena bertako normatik ateratzen bazen ere. Beste askok arazoak ikusten zituzten lekuan, zuk aurrera egiteko erabakia hartu zenuen. Harez gero, urteak pasa dira, era guztietako gora beherak izan dira, tesiaren proiektua erabat aldatu egin da, eta zuri esker amaitzea lortu dut.

Eskerrik asko ere finantzio propiorik izan ez duen proiektu hau ekonomikoki bideratzen lagundu duzuenoi, Konputagailuen Arkitektura eta Teknologia Saila eta Egokituz taldeari.

Eskerrik asko PIE/HBPetan parte hartu duzuen guztioi, Txelo, Natxo, Javi, Iratxe, Nestor, Lukas, Santiago, Teresa eta Ibai. Hor egindako lanak ere berebiziko garrantzia izan du tesi honengan.

Bazkalorduko taldetxoa ezin ahaztu. Horko tertuliak eta fotosintesi gabe seguruenik indar falta izango nintzateke.

Ezin ahaztu tesian zehar egindako esperimentuetan parte hartu duten guztiak. Batetik ikasleak, metodologia eta teknologia berriak probatu behar izan dituzue eta jarraitu beharrekoa ez da beti arrosa bidea izan. Bereziki nirekin 2010/2011, 2011/2012 eta 2012/2013 ikasturteetan egon zaretenek sufritu behar izan duzue tesi hau, baina aurreko urteetan, oraindik ere plan zaharrean nirekin izan zineten ikasleak ere ez zaituztet ahazten. Bestetik irakasleak, zuen ezagutza kontzeptu mapen bidez adierazteko eskatu nizuen batzuei eta jakin badakit ez zela lan makala izan, eskerrak beraz Gonzalo, Teresa, Javi, Natxo, Iratxe, Txelo, Clemente eta Nestor.

Roberto, eskerrik asko Latex-ekin laguntzeagatik, auskalo nolako txostena edukiko nukeen zure txantiloi eta aholkurik gabe.

Ana, zuri ere eskerrik asko bulegokide aparta izanagatik, eman didazun laguntza eta animoengatik.

GALAN taldekoei, bereziki Iñakiri ere eskerrik asko kontzeptu mapekin emandako laguntzagatik eta aplikazioan gauzak nire beharretara egokitzeagatik.

Oro har, Informatika Fakultateari ere eskertu nahi dizkiot bertan jaso ditudan ezagutzak. Lehenik nire irakasle izan zineten asko, orain nire lankide zarete eta behar izan dudan guztietan egon da norbait laguntzeko eta animoak emateko prest.

Eta nola ez, eskerrak familiari zuen animo eta laguntzagatik eta bereziki Oier, Izaro, Ander, eskerrik asko etxea alaitzeagatik eta etxean gauzak ondo joan zitezen emandako laguntza guztiagatik. Kiki, ez naiz kapaz tesi hau aurrera atera zedin egin duzun guztia hitzez adierazteko, mila esker!

Acknowledgements

This work would have been impossible to accomplish without the help of so many people. I would like to thank Maria José Santofimia for trying our methods with her students and giving us the results.

Also all the people in the ICER doctoral consortium that helped me by redirecting the objectives of the thesis. It was very nice to be able to talk to people that could perceive what I was doing from a different perspective. And the same acknowledgement goes also for the community of the Koli Calling conference. Thank you very much!

Capitolina, although finally it become very hard to add a gender perspective to this work (there are so few girls in our classrooms!) thank you for helping when you could and for understanding all the changes.

This last acknowledgement was written also in Basque, but since some of the people will not be able to understand, here it goes again. Thank to the professors that accepted our request to draw concept maps for this research. I know that it was not an easy task, thanks therefore to Gonzalo, Teresa, Javi, Natxo, Iratxe, Txelo, Clemente and Nestor.

"If we knew what we were doing, it wouldn't be called research."

Like so much of what Einstein said, this quotation communicates a very important idea in a very simple way. The path to discovery is seldom straight. Rarely does the reader of the final research have the opportunity to understand how the journey of discovery unfolded.

By Mark Cotteleer[1]

[1] At http://dupress.com/articles/its-called-research-the-exceptional-company-collection/

Preface

Mark Cotteleer's previous quote gives me the opportunity to introduce the reader to the journey of this dissertation.

Most of the people I see around me do their research within a research group. Although a PhD dissertation is an individual work, many times it is necessary to explain which parts out of all the things reported in the dissertation are the PhD student's work and which are the group's work.

In this case there is also a group of people supporting the work, but it is not a research group, it is a group of teachers that have been teaching the computer I/O subsystem in the last years.

Thanks to that group, and also to two colleagues from the school of pedagogy, I have been able to get some funds from the Educational Counselling Service (SAE/HELAZ from its initials in Spanish and Basque) for two Educational Innovation Projects (PIE/HBP from its initials in Spanish and Basque) in which I have been the leader.

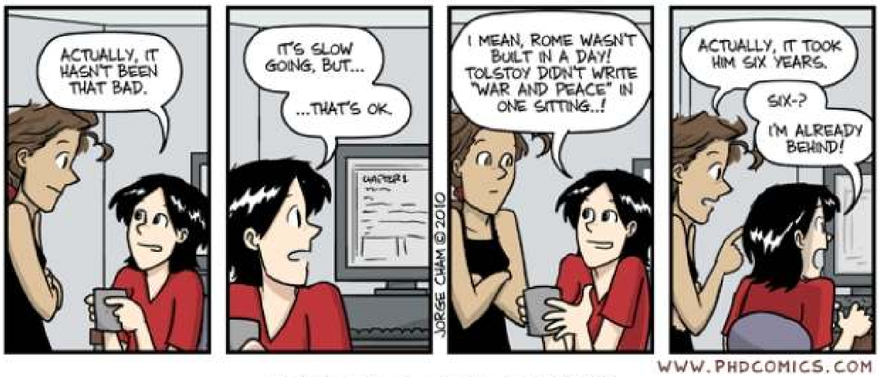
That group, where my PhD advisor can be found, has supported my decisions, has brought some ideas, and has helped writing some of the papers, but without any background on Computing Education Research, since each of them have their own research area.

Thanks to ITICSE, but mostly ICER and Koli Calling conferences I have been able to understand some of the intricacies of this research area, which I am sure will still surprise me many times.

I hope this short explanation can help the reader imagine the very twisted path of this dissertation.

Thanks again to my PIE colleagues and to the CER community.

Piled Higher and Deeper ***by Jorge Cham*** **www.phdcomics.com**


ACTUALLY, IT HASN'T BEEN THAT BAD.
IT'S SLOW GOING, BUT...
...THAT'S OK.
CHAPTER 1
JORGE CHAM © 2010
I MEAN, ROME WASN'T BUILT IN A DAY! TOLSTOY DIDN'T WRITE "WAR AND PEACE" IN ONE SITTING..!
ACTUALLY, IT TOOK HIM SIX YEARS.
SIX-?
I'M ALREADY BEHIND!
WWW.PHDCOMICS.COM

title: "Write in Pieces" - originally published 8/4/2010

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

THIS dissertation is all about the educational process of the computer I/O subsystem. Therefore it is important to have at least a rough idea of the topic itself, the reasons why it is important to teach the computer I/O subsystem to computer scientists and engineers, and what does the computing curricula say about the computer I/O subsystem.

Next, in this chapter some changes which took place at the University of the Basque Country (UPV/EHU) will be introduced, then the research questions proposed for the dissertation will be outlined, and finally, the organization of the document will be summarized.

1.1 The computer I/O subsystem - What is it?

As Stallings [2012] defines, *"The computer system's I/O architecture (or subsystem) is its interface to the outside world". "I/O subsystem architecture deals with the organization of concurrent processing activities within the devices, controllers, channels, and I/O processors that comprise an I/O subsystem. In addition, it deals with the coordination of I/O subsystem processing activity as a whole with respect to the activity of the central processor"*, say Buzen [1975]. A computer can not do anything without data from the outside world, and serves to nothing if it can not bring the results of its computations to the outside world.

The aim of the I/O subsystem, as an undergraduate introductory topic is to teach the basics of the communication between the computer system and the I/O architecture or subsystem, in order to reach the outside world.

It is very easy to see how a mechanical typewriter converts a key press into a letter in the paper, but most computer users will not be able to say which actions happen between the moment a person presses a key in a computer keyboard and the moment the letter appears in the display.

1.2 Why is it important to teach the computer I/O subsystem to computer scientist and engineers?

The I/O subsystem is one of the main parts of a computer system together with the CPU, the memory unit, and the buses which have the responsibility of linking all the three units, CPU, memory and I/O subsystem. The techniques used to synchronize the I/O subsystem with the CPU and memory, and the correct functioning of it can highly influence the performance of the computer. The ACM/IEEE-CS joint task force [CC2004] points out that: *"students must understand how various peripheral devices interact with, and how they are interfaced to a CPU"*.

The use of interrupts to synchronize the I/O subsystem with the CPU opens the door to introduce students to concepts such as multiprogramming and concurrency in the execution of the programs that run in computing systems.

This type of execution will increase the performance of the computer systems with respect to monoprogramming and synchronization via polling.

On the other hand, I/O subsystem can be considered as it forms the lowest level in Human Computer Interaction (HCI). When designing interfaces based on non-standard devices (those which are not the classical keyboard, screen and mouse), developers have to know other I/O devices and their characteristics [Garay-Vitoria, 2006].

The use of microprocessors is becoming ubiquitous. *"Almost every electronic appliance and device today uses embedded systems. Cell phones, automobiles, toasters, televisions, airplanes, medical equipment, and a host of other devices, products, and applications use embedded systems"* [CC2004]. In these embedded systems, the Input/Output (I/O) subsystem takes usually a considerable role. *"The various I/O techniques are essential to software developers designing operating systems, network software, database systems, embedded systems, process control systems, and real-time systems"* [Krishnaprasad, 2002]. However, lecturers in these embedded systems courses feel that in previous courses the I/O subsystem is not treated sufficiently: *"Today's computer architecture courses spend more time on instruction set design and pipelining and less time on topics like I/O. As a result, before getting to the significant examples in embedded computing, a course must first cover some basic principles and techniques that have not been learned in other courses"* [Wolf and Madsen, 2000].

1.3 What do the computing curricula recommendations say about the I/O subsystem topic?

The ACM/IEEE-CS Joint Task Force has been developing curriculum recommendations for the different and emerging computing disciplines since the 1960s. They developed a volume for Computer Science (CS) in 2001 [CC2001], a revision of it in 2008 [CC2008], and a new draft was published in February 2013 [CC2013].

The curricula for CS say that: *"The computer lies at the heart of computing. Without it most of the computing disciplines today would be a branch of theoretical mathematics. To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic. All students of computing should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions."* [CC2001; CC2008]. Moreover, the ACM/IEEE-CS joint task force's curriculum for Computer Engineering (CE) [CC2004] remarks that: *"Computer architecture is a key component of computer engineering and the practising computer engineer should have a practical understanding of this* *topic. It is concerned with all aspects of the design and organization of the central processing unit and the integration of the CPU into the computer system itself.".* But, Computer Architecture (CA) is a very wide area covering topics such as: Digital logic and digital systems, Machine level representation of data, Assembly level machine organization, Memory system organization and architecture, Interfacing and communication, Functional organization, Multiprocessing and alternative architectures, Performance enhancements, Architecture for networks and distributed systems, and much more. The subject that this curricula recommendations devote to the I/O topic is called *"Interfacing and communication"* [CC2001; CC2013] or *"Interfacing and I/O Strategies"* [CC2008].

In [CC2008] the ACM/IEEE-CS Joint Task Force tried a change, but since in the [CC2013] draft the subject comes back to be the same as in [CC2001], we will not consider here the [CC2008] proposal. The [CC2013] proposal has the same topics and learning outcomes as [CC2001], but with a little more detail. It devotes a minimum coverage time of 2 hours and states as follows:

"Topics:

- *I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O*
- *Interrupt structures: vectored and prioritized, interrupt acknowledgement*
- *External storage, physical organization, and drives*
- *Buses: bus protocols, arbitration, direct-memory access (DMA)*
- *Introduction to networks: networks as another layer of access hierarchy*
- *Multimedia support*
- *RAID architectures"*

"Learning outcomes:

1. *Explain how interrupts are used to implement I/O control and data transfers [Knowledge].*
2. *Identify various types of buses in a computer system [Knowledge].*
3. *Describe data access from a magnetic disk drive [Knowledge].*
4. *Compare common network organizations, such as ethernet/bus, ring, switched vs. routed [Knowledge].*

5. *Identify interfaces needed for multimedia support, from storage, through network, to memory and display [Knowledge].*
6. *Describe the advantages and limitations of RAID architectures [Knowledge]."*

In 2004 the ACM/IEEE-CS Joint Task Force developed a curriculum for CE [CC2004]. It devotes a minimum coverage time of 10 hours and the differences with the CS curriculum can be seen in the topics and learning outcomes as follows:

"Topics:

- *I/O fundamentals: handshaking, buffering,*
- *I/O techniques: programmed I/O, interrupt-driven I/O, DMA*
- *Interrupt structures: vectored and prioritized, interrupt overhead, interrupts and reentrant code*
- *Memory system design and interfacing*
- *Buses: bus protocols, local and geographic arbitration"*

" Learning outcomes:

1. *Explain how to use interrupts to implement I/O control and data transfers.*
2. *Write small interrupt service routines and I/O drivers using assembly language.*
3. *Identify various types of buses in a computer system.*
4. *Describe data access from a magnetic disk drive.*
5. *Analyze and implement interfaces."*

As it can be seen, the major difference is that while the CS curriculum [CC2013] stays at the knowledge level, the CE curriculum [CC2004] demands as learning outcomes the ability to write small Interrupt Service Routine (ISR) and I/O drivers, and the ability to analyse and implement interfaces.

1.4 The computer I/O subsystem topic at the University of the Basque Country (UPV/EHU)

At the UPV/EHU the subject called *"Computer Architecture 1"* was taught in the third semester out of the ten semesters of the studies in Computer Engineering. This subject comprised the central processing unit (CPU) (the control unit and the arithmetic-logic unit), the memory organization, the computer I/O subsystem, and the interconnection buses, a detailed introduction to the Von Neumann architecture. The *"Computer Architecture 1"* subject concerned the lecturers because of its high dropout rates (31%) and low pass rates (50% out of the taken exams, 34% out of the enrolment). A problem area was detected in the course of the years: the computer I/O subsystem. Students drastically left the subject during the teaching of the computer I/O subsystem. Moreover, the 50% pass rate previously mentioned droped to 44% considering only the computer I/O subsystem part of the exam.

Due to the adoption of the European Higher Education Area (EHEA), the computing degree taught at the UPV/EHU has recently undergone several changes. Now the degree in Computer Engineering is eight semesters long. The *"Computer Architecture 1"* subject has disappeared, and the computer I/O subsystem has moved from a third semester subject *"Computer Architecture 1"* to a second semester subject *"Computer Structure"* (CSt to make it different from Computer Science (CS)). In this subject, the computer I/O subsystem shares centre stage with assembly language and low-level programming. At that point, the concern of the lecturers towards the computer I/O subsystem topic increased since the difficulty in understanding a topic would be greater in the first year.

A change was needed, and the aim of this dissertation is to report the changes made, the data obtained empirically, and the methods implemented in order to be able to know the real learning outcomes of the topic.

1.5 Research questions

In this chapter, the research questions that are going to be answered in the course of the dissertation are going to be expressed.

Q1: How well can a handheld game console be adapted to the teaching needs of the I/O subsystem topic?

Q2: How well can a PBL methodology help in the learning process of the I/O subsystem topic?

Q3: How do students understand the I/O subsystem topic?

1.6 Organization of the Document

Chapter 1. In the first chapter we offer some hints about general concepts of the computer I/O subsystem.

Chapter 2. In the second chapter a justification has been given for this dissertation to be part of the Computing Education Research. Moreover, the whole work carried out in this research is framed in the TPACK model. The dissertation considers the lecturer's technological knowledge, pedagogical knowledge and content knowledge, and searches for the deficiencies on TPACK that lies in students' knowledge.

Chapter 3. The third chapter shows the results of a survey of the computer I/O subsystem as a topic in different universities, textbooks, and research bibliography. Four different approaches have been found to exist and they have been linked to the universities and textbooks that are using them. Moreover the research literature has been analysed in order to find more on how the computer I/O subsystem is taught.

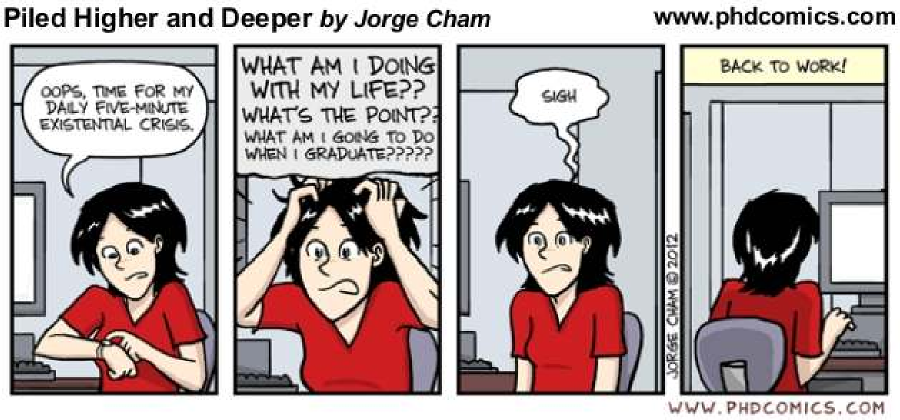
Chapter 4. This chapter introduces the Nintendo DS as a machine to reinforce the computer I/O subsystem learning.

Chapter 5. The fifth chapter reports the application of the Project-based learning methodology during three years, explaining each years settings. An example of the projects developed by the students each year has also been detailed.

Chapter 6 Here, the data gathered empirically during the three years in which the Nintendo DS has been used in the Project-based learning methodology are analysed.

Chapter 7. The seventh chapter reports the study where the students' knowledge is analysed in order to have a better perspective of how to deal with the students understanding of the topic.

Chapter 8. This chapter wraps up the results presented in this work and suggests directions for future work.


Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
OOPS, TIME FOR MY DAILY FIVE-MINUTE EXISTENTIAL CRISIS.
WHAT AM I DOING WITH MY LIFE?? WHAT'S THE POINT?? WHAT AM I GOING TO DO WHEN I GRADUATE?????
SIGH
JORGE CHAM © 2012
BACK TO WORK!
WWW.PHDCOMICS.COM

title: "The Daily Routine" - originally published 4/20/2012

CHAPTER 2

Link to relevant theory

THIS chapter is devoted to explain in which way this dissertation can be integrated in the emerging field of Computing Education Research. This term often appears as Computer Science Education Research. However, the computer I/O subsystem has a stronger consideration in the field of computer engineering as can be seen in the curricula recommendations [CC2001, CC2004, CC2008, and CC2013]. Therefore, since the term computing endorses the topic better than the term computer science, it is the one used in this dissertation, considering that all the statements about computer science education research are equally applicable.

Fincher and Petre [2004] identified ten broad areas that motivate researchers in Computing Education.

- **Student understanding:** where students' mental and conceptual models, their perceptions and misconception are studied;
- **Animation, visualization and simulation:** where software tools and environments are used in order to affect students' learning;
- **Teaching methods:** where the studies analyse the ways in which teachers can "build bridges" for students, the ways they can scaffold their students' learning, helping them to make sense of the subject,

or the ways teachers control the dynamics of the teaching interaction to make it profitable;

- **Assessment:** where the types or validity of assessment, or how to automate grading are studied;
- **Educational technology:** where the advantages that new devices and technologies offer are harnessed, some times as assistive technology, and sometimes, applied to Computing subject matters;
- **Transferring professional practice into the classroom:** where academics seek inspiration in the work of professionals;
- **Incorporating new developments and new technologies:** where due to the fast industry development, the new incorporations are often difficult to transmit;
- **Transferring from campus-based teaching to distance education:** where many generic issues about Web-based learning and appropriate transfer of educational interaction are studied;
- **Recruitment and retention:** where the reasons for students to come into computing or stay there are analysed, paying special attention to diversity and gender issues; and,
- **Construction of the discipline:** where as well as curricula construction, also questions concerning the nature of the discipline (whether it is engineering, mathematics, design, business, or something else altogether) are posed.

The work developed in this dissertation touches the following of these areas:

- the construction of the discipline, when the curricula and different approaches to teach the computer I/O subsystem are analysed, in order to answer the question *"How is the computer I/O subsystem taught?", "What different approaches are taken?"*;
- animation, visualization and simulation, or incorporating new developments and new technologies, when the Nintendo DS machine is incorporated to the teaching and the benefits and disadvantages of using it are analysed;

- teaching methods, when the PBL methodology is used and the results analysed depending on the different settings of several parameters;
- educational technology, when the technology of a specific machine is turned into the infrastructure to support students' understanding of the topic;
- the student understanding area has its place here where concept maps have been used in order to picture the understanding of the topic and the end of its teaching.

From the educational point of view that a research project in Computing Education should have [Berglund, 2005], this dissertation is framed by the Technological, Pedagogical and Content Knowledge (TPCK or TPACK) model first introduced by Koehler, Mishra, Yahya, and Yadav in [2004], as the complex interplay of content (C), pedagogy (P), and technology (T), build upon the Pedagogical Content Knowledge model of Shulman [1986]. In [Mishra and Koehler, 2005], the authors defined the model as *"a framework for thinking about what teachers need to know about technology"*.

The model has been defined by the image in Figure 2.1.

This model has been reported with several weaknesses [Graham, 2011], such as:

- building on an unsure foundation;
- hiding behind its simplicity, a deep underlying level of complexity, in part because all of the constructs being integrated are broad and ill-defined;
- imprecise definition of its constructs;
- imprecise definition of technology itself;
- imprecise definition of whether its constructs relate in an integrative or transformative way;
- imprecise definition of its constructs boundaries;

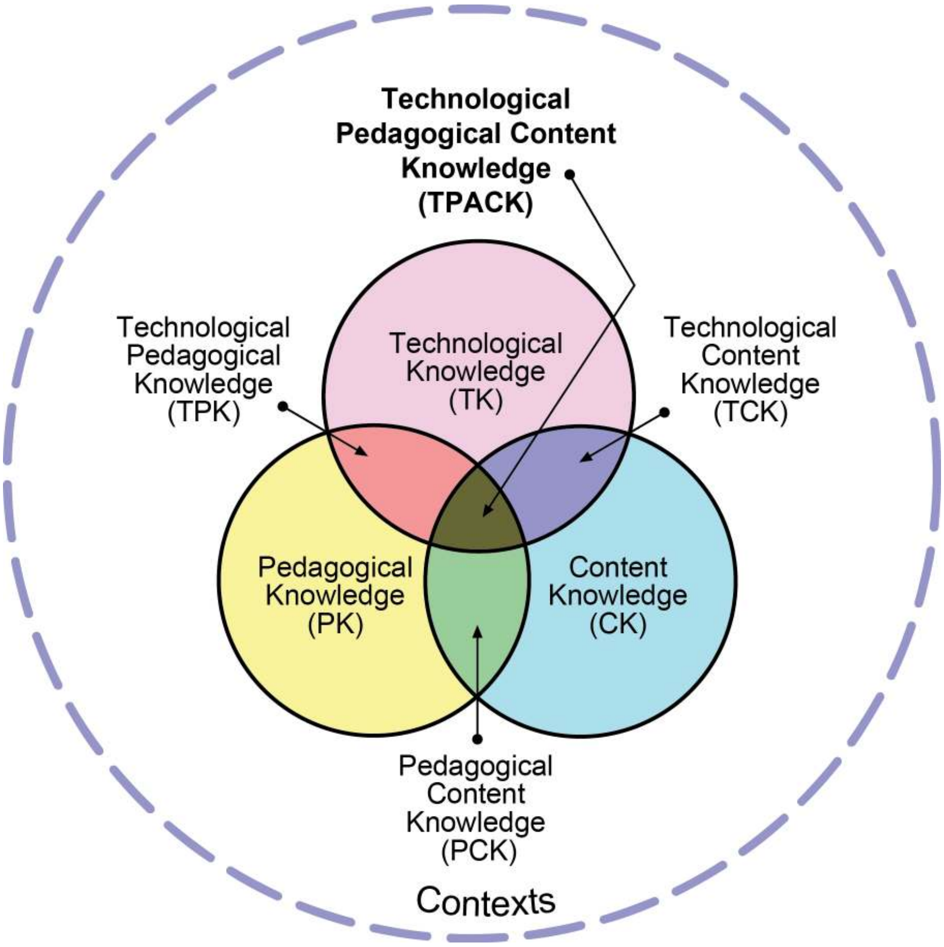

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.1: The TPACK Image. Reproduced with permission of the publisher, © 2012 by tpack.org

- the possibility of conflating TPACK with technology integration;
- insecure prescriptive value.

However Cox [2008] had already specified the constructs and its boundaries as follows:

- **Pedagogical Knowledge (PK):** Knowledge of the general pedagogical activities that a teacher might utilize, where general activities are independent of a specific content or topic (meaning they can be used with any content) and may include strategies for motivating students, communicating with students and parents, presenting information to students, and classroom management among many other things. Additionally, this category includes general activities that could be applied across all content domains such as discovery learning, cooperative learning, problem-based learning, etc.

- **Content Knowledge (CK):** Knowledge of the major facts and concepts within a field and the relationships among them. Knowledge of the possible topic-specific representations in a given subject area. This knowledge is independent of pedagogical activities or how one might use those representations to teach.

- **Technological Knowledge (TK):** Knowledge of how to use emerging technologies, considering technology any tool or collection of tools.

- **Technological Content Knowledge (TCK):** Knowledge of the technology-content interaction, independent of pedagogy.

- **Technological Pedagogical Knowledge (TPK):** Knowledge of the technology-pedagogy interaction independent of topic-specific representations or content-specific instructional strategies.

- **Pedagogical Content Knowledge (PCK):** In order to facilitate student learning, knowledge of activities and knowledge of representations are combined here. The knowledge of pedagogical activities is content-specific rather than general because PCK is situated in a particular subject area.

- **Technological Pedagogical Content Knowledge (TPACK):** knowledge of how to coordinate the use of subject-specific activities or topic-specific activities with topic-specific representations using emerging technologies to facilitate student learning.

Most of the research developed using the TPACK [Koehler et al., 2004] or the PCK [Shulman, 1986] models has taken place in the non university teacher education context. Moreover, to our knowledge, when it has been used in the field of computing, it has been applied to secondary education [Saeli, 2011] and [Hubwieser, Magenheim, Mühling, and Ruf, 2013].

That is not the case of this dissertation. In this case, the use of the model has been applied to a university level computing topic where the lecturers are struggling to get good results from their students. Therefore, in a topic of the second semester of the undergraduate studies, they are trying to use new pedagogical techniques and new technologies in order to achieve better results.

The following sections will summarize how the different steps of the research apply to the constructs of the TPACK model in Figure 2.1. However, in order to use such a model, it has to be understood and internalized. Figure 2.2 shows the way in which the model has been understood to develop this research, by means of a concept map.

2.1 Content Knowledge (CK), Pedagogical Knowledge (PK) and Technological Knowledge (TK), the basic constructs.

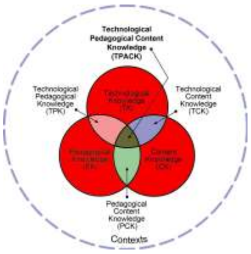

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Content Knowledge (TCK)
Technological Knowledge (TK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.3: Basic constructs

The basic constructs, on their own, do not help much with the teaching of a specific subject. Does the lecturer know the topic she has to teach? She is supposed to, since it is a requirement of her workplace. In this specific case, the content to be taught is the computer I/O subsystem. What it is and its importance has been explained in Chapter 1.

However, different representations have also been found to exist. Section 3.1 describes 4 different approaches to understand the content.

Does the lecturer know any pedagogical technique? At least in our university this is not a requirement. However, just knowing the pedagogical techniques is not sufficient to lead the students towards a significant learning. This knowledge must be aligned with the content to teach. In any case, during students' first year studies in the university, it has been

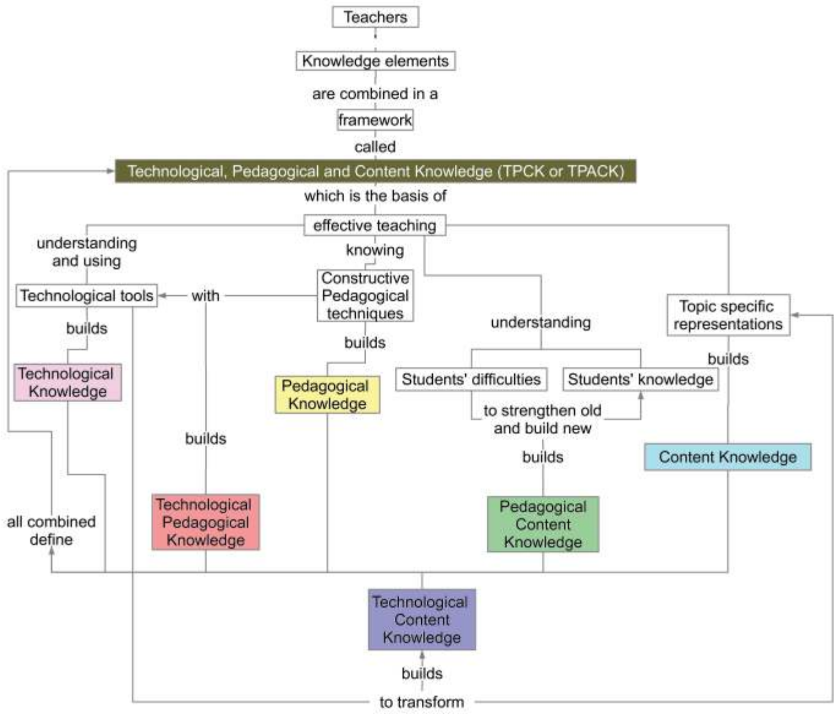


Figure 2.2: The TPACK model as understood and used in this dissertation. For the constructs, the same colours have been used as in Figure 2.1

found that teachers need to have a more pedagogical approach than in the other years of studies in the university.

Does the lecturer know technological tools? Since we are in a very technological environment, the faculty of computing, probably the lecturer will know many technological tools, but in what extent are these tools related to the topic or to the applied pedagogical techniques?

The other constructs of the model are the ones that combine the basic constructs, and the ones that will lead the research of this dissertation.

2.2 Technological Content Knowledge (TCK)

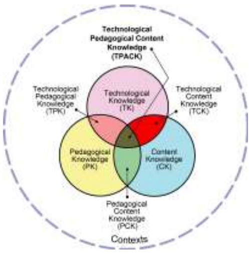

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.4: TCK construct

Being the computer I/O subsystem the content, the boundary with the technological construct is not clear at all. The content itself is technological. Every machine has an I/O subsystem and the lecturers, based on their content knowledge are able to decipher the specificities of the I/O subsystem of each machine.

Before the changes on the subject started, the PC was used as infrastructure, as mentioned in Subsection 5.1. Chapter 4 explains the characteristics of the Nintendo DS machine considering the computer I/O subsystem, which is the content at hand.

2.3 Technological Pedagogical Knowledge (TPK)

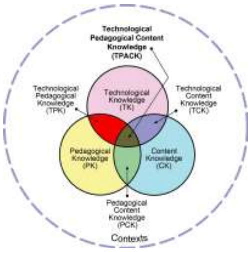

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.5: TPK construct

This construct is independent of content-specific representations. The technology analysed in this dissertation is the Nintendo DS. Originally it is a handheld game console devoted to play games. However, the so called serious games have been broadly used with pedagogical purposes as an extensive bibliography shows (see for example [Hanakawa, Yamamoto, Tashiro, Tagami, and Hamada, 2008], [Shirali-Shahreza, 2008]) and have also been proved to be more effective in terms of learning [Wouters, van Nimwegen, van Oostendorp, and van der Spek, 2013].

Not only video games have been used as a pedagogical tool but also video game development. Wu and Wang [2012] have done a good literature review on game development frameworks and also on how game development-based learning has been used in different learning areas. The laboratories explained in Setion 5.1.2, and the PBL assignments of Section 5.2 are based on this kind of knowledge.

2.4 Pedagogical Content Knowledge (PCK)

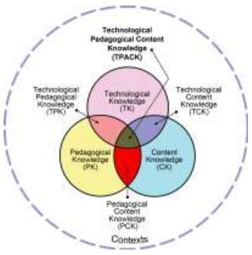

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.6: PCK construct

Project-based learning is a pedagogical methodology very popular nowadays. However, project work has always been used in computing because *"they can give insight into the sort of systems that students observe on their desktop computers, and can help bridge the gap between systems they use and systems they construct"* [Fincher, Petre, and Clark, 2001].

This assertion is also valid in the context of computer I/O subsystem. A project in which the students are able to manage the I/O devices can help them overcome the abstractness of something that always happens in that magic box inside the computer.

The TCK construct has been analysed in different ways. First, Chapter 3, Sections 3.3.1 and 3.3.2 review the technology used to teach the same topic by other lecturers. In Chapter 4, a new option is proposed and then analysed from the point of view of the specific content of the topic.

2.5 Technological Pedagogical Content Knowledge (TPCK or TPACK)

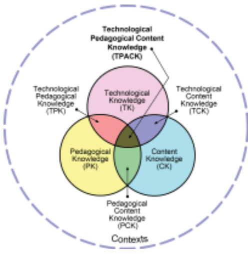

Technological Pedagogical Content Knowledge (TPACK)
Technological Pedagogical Knowledge (TPK)
Technological Knowledge (TK)
Technological Content Knowledge (TCK)
Pedagogical Knowledge (PK)
Content Knowledge (CK)
Pedagogical Content Knowledge (PCK)
Contexts

Figure 2.7: TPACK construct

Chapters 3 and 5 combine all the knowledge of the previous constructs in order to analyse the use of project-based learning with different settings (PK) with the NDS (TK) to teach the computer I/O subsystem (CK). Chapter 6 analyses the empirical data gathered during the application of all that knowledge.

However in the pedagogical construct there is a very important point that has not been touched so far. This point is not clear in the TPACK Figure 2.1, although it appears in its definition in

[Mishra and Koehler, 2006]. This points considers the students' knowledge, with the following sentences in the definition of TPACK:

- knowledge of what makes concepts difficult or easy to learn and how technology can help redress some of the problems that students face;
- knowledge of students' prior knowledge and theories of epistemology;
- knowledge of how technologies can be used to build on existing knowledge and to develop new epistemologies or strengthen old ones.

After two years of applying the PBL methodology, the results obtained made evident the need of analysing students' knowledge. The third year of the research students' knowledge was elicited at the end of the term, using concept maps for that objective. Chapter 7 shows the results obtained from that study.

In the conclusions of this dissertation some changes to the methodology will be proposed after considering the results of the analysis conducted with the concept maps. However a door opens after this dissertation to further analyse students' difficulties in order to be able to propose new techniques.

2.6 Conclusions

In this chapter, it has been explained how this dissertation is integrated within the area of Computing Education Research (CER). This integration has been done mainly by means of the TPACK model. The different elements that are covered in the educational process of the computer I/O subsystem are related with the constructs of the TPACK model.

In addition, this chapter has served to relate the chapters of this dissertation with the areas that motivate researchers in Computing Education.

CHAPTER 3

Background and related work

As a component of the computer system, the I/O subsystem is often identified as an introductory topic [Brylow and Ramamurthy, 2009a; Stojcev, Milentijevic, Kehagias, Drechsler, and Gusev, 2003]. However, there are at least two problems surrounding the educational process for this topic. The first is the level of abstraction with which students are faced. The second is the level of confidence of the people teaching in this subject area. Cassel, Holliday, Kumar, Impagliazzo, Bolding, Pearson, Davies, Wolffe, and Yurcik [2001] report that 82 individuals responded to a survey distributed to people teaching in the computer architecture and organization area and found that *"Fault Handling & Reliability and Synchronization & Handshaking are apparently poorly understood, but are taught by only a small percentage of instructors. [...] Bus Systems & DMA, I/O Control Methods, Interrupts, and External Storage [...] sub-areas are taught by about one-half of respondents and the low self-confidence index of a large portion of them indicates this is an area where instructors could use help."* This people could take advantage of an up to date review of the literature for the computer I/O subsystem as a teaching topic, as this one is. Next, the main teaching approaches are evaluated, and the technology available for teaching, both software and hardware, are presented.

3.1 Different approaches to teaching the computer I/O subsystem

3.1.1 The data gathering and analysis process

Three studies were conducted which led the author of this work to define a classification of 4 different learning approaches for the computer I/O subsystem. The first study was on exam questions, the second on course syllabi and the last on textbooks. Next in this chapter, first the studies and then the 4 approaches will be described.

The author of this work has previously analysed data from 6 different Spanish universities. The data were exam questions to test students' knowledge about the computer I/O subsystem [Larraza-Mendiluze and Garay-Vitoria, 2013b]. This analysis showed that pure or applied knowledge recall was needed to answer 30% of the exam questions, some coding skills were needed to answer 30% of the questions, and analysis of performance was required in 40% of the questions.

For the purpose of this chapter, the authors wanted to identify the approach taken by different universities around the world; but which universities and why those universities? The decision was taken to search the syllabi of the universities ranked among the top 100 universities in all 6 well known university rankings [Shanghai_Jiao_Tong_University, 2013; QS, 2013; Thomson_Reuters, 2013; Cybermetrics_Lab, 2013; Informatics_Institute_of_Middle_East_Technical_University, 2012; Hirst, 2008]. 36 universities were found to meet this requirement. Then, undergraduate courses including the computer I/O subsystem were searched (looking for their syllabus, assignments and exams). In 8 cases, there was either no information found or the information was too scarce to be considered. In 9 more universities the I/O subsystem was found to be taught in Operating Systems courses or from an operating systems point of view and these were therefore not considered for this study, which has a clear computer architecture and organization perspective. As a first approach, the syllabi, assignments and exams available online from the remaining 19 universities were analysed. The universities and names of the subject analysed can be found in Table 3.1.

Finally, some widely used textbooks were considered. As a starting point, the latest editions of the textbooks presented in [Cassel et al., 2001] as the most widely used were selected [Patterson and Hennessy, 2009; Tanenbaum and Austin, 2012; Stallings, 2012; Mano, 1993; Scragg, 1992; Hamacher, Vranesic, Zaky, and Manjikian, 2011]. From that starting point, it was found that [Mano, 1993] and [Scragg, 1992] were last published in 1993 and 1992 respectively. Initially, for this study they were considered to be too old and they were not going to be taken into account. However, a search was carried out to determine whether the books appeared in new syllabi and the finding was that [Mano, 1993] is still widely used and it was therefore reconsidered. On the other hand, [Tanenbaum and Austin, 2012] does not say much about the computer I/O subsystem. Tanenbaum and Austin mainly describe how certain peripherals are built, then mention the I/O techniques, and finally talk about I/O in different operating systems. Therefore [Tanenbaum and Austin, 2012] was not considered.

The next subsections will define each approach, mentioning the universities and textbooks that have been considered to use this approach and giving the reasons for that consideration, based on the information found. Please bear in mind that both universities and textbooks can take more than one approach.

3.1.2 The purely descriptive approach (PDA)

In this approach the computer I/O subsystem is described to the students, who are expected to be able to describe the concepts and, at best, to identify relationships between them. This is an easy way to introduce the topic, as required in [CC2001]. It could be used even with students that are not majoring in Computer Science or Engineering. However, it has to be considered that it stays at a low level in the knowledge taxonomies (see [Bloom, 1956] or [Biggs and Collis, 1982]).

Any of the textbooks in the most widely used list [Patterson and Hennessy, 2009; Stallings, 2012; Mano, 1993; Hamacher et al., 2011] could be considered for this approach. All the books describe the topic although the examples and exercises specify which of the following approaches each book takes.

With respect to the universities, some of them did not have any exercises devoted to the computer I/O subsystem in their exercise list, nor was there any assignment or exercise found that was devoted to the computer I/O subsystem in their exams. They were therefore considered to be presenting the I/O subsystem descriptively, intending their students' learning outcomes to be at a knowledge level [Bloom, 1956]. The list of universities following this approach, with the corresponding course can be seen in Table 3.1.

3.1.3 The performance approach (PeA)

The computer I/O subsystem is a bottleneck in the computer system. Its design has a major effect on the computer's performance. In this approach, the students are asked to calculate the performance of a computer system considering using different I/O techniques. This kind of questions are considered higher order questions since they require the application of knowledge and often the evaluation of results to determine which technique is most appropriate for the context [Bloom, 1956].

Performance is a key feature in computing and students must be aware of it. However, in the I/O subsystem, there is often a person at the end of the line, and when that happens it is difficult to justify why one or other option has been taken. In that case the bottleneck may not be improved from the computing perspective. Moreover, there is place enough in many other subjects in the CA branch to deal with performance.

The textbooks best suited for this approach are [Patterson and Hennessy, 2009; Stallings, 2012], since the questions and problems they contain are of this kind.

As for the universities, only the Computation Structures course from Massachusetts Institute of Technology (MIT) was found to follow this approach (see Table 3.1). In their course website they propose tutorial problems that ask the student the questions such as the time devoted to servicing and interrupt, or the time that a program will last if some specific interrupts are enabled. This kind of problems have not been found in any other course information.

3.1.4 The programming approach (PrA)

Considering the recent trend in computing towards embedded systems, the importance of human computer interaction (HCI), where the I/O subsystem is very important [Garay-Vitoria, 2006], and the last Computer Engineering curriculum of the ACM/IEEE-CS Joint Task Force [CC2004], it is also important to see the functioning of the computer I/O subsystem from a programming point of view, accessing the I/O registers, programming the Interrupt Service Routines (ISR), etc.

This approach helps introducing very abstract concepts such as interrupts or concurrency. Programming the computer I/O subsystem can help making these concepts clear. However the problem is that in undergraduate introductory courses the programming level of the students can be not high enough, or low level programming is not very appealing.

The textbook in the most widely used list that best introduces this approach is [Hamacher et al., 2011], which treats embedded systems directly.

Of the universities analysed, those listed in Table 3.1 have labs or projects in which students have to program Interrupt Service Routines (ISR) (sometimes in assembler and, sometimes in higher order languages) in order to for example print a string in the display, program an alarm clock with Arduino, or program a game for the Game Boy Advance (GBA).

3.1.5 The datapath-signal approach (DSA)

Although the last edition of [Mano, 1993] is from 1993, it was found to be the fourth most widely used textbook among people teaching computer organization and architecture in [Cassel et al., 2001] and a quick search on the Internet showed that it is still being used in several courses. This book presents a different approach, focusing on the path of the data going into and out of the computer, and the signals used to synchronize devices, CPU, and memory. This approach includes a lot of block and timing diagrams and flowcharts.

This approach can be very helpful to envision how the hardware works. However it rests at an understanding level, since getting students to make changes in the hardware can get really hard. This could be something to do in a more advanced level.

It has not been found among the universities analysed, but probably the universities that follow Mano [1993], also follow this approach.

Any of the abovementioned approaches could be taught in a classical way, by lectures and paper exercises. However, there are authors who claim that students need help to make the cognitive leap required to connect their theoretical knowledge with their practical experience [Djordjevic, Milenkovic, and Grbanovic, 2000], for which other educational tools such as simulators or real machines could be used. Section 3.3 will review the publications in that context.

3.2 How do textbooks present the I/O topic?

Considering the textbooks selected in section 3.1.1 we will analyse how do they treat the topics considered in the curricula mentioned in section 1.3. At the end, tables 3.2 and 3.3 show where in each book appear each of the topics and learning outcomes proposed in the curricula, and Table 3.4 shows the approach of each textbook.

3.2.1 Computer Organization and Architecture 9th Edition

Stallings [2012] devotes the 7th chapter of his book to the I/O subsystem. Indeed, he entitles the chapter *"Input/Output"*. It is a very descriptive book, in which the concepts of the I/O subsystem and their function are described, i.e.: peripheral, I/O modules, I/O registers, Programmed I/O (a.k.a polling), interrupt-driven I/O, Direct Memory Access (DMA), memory-mapped I/O, isolated I/O, interrupt controller, I/O channels, I/O processors, the buses used for the communication between a peripheral and an I/O module, and the types of buses used for that communication.

The last section, the one devoted to the questions and problems shows the intended outcomes of the chapter. The problems proposed are

University	Subject	Approach	Book
Cornell University	CS 3410: Computer System Organization and Programming	PDA	PatHen
Duke University	CS 250: Computer Architecture	PDA	PatHen
National University of Singapore	CS 2100: Computer Organization	PDA	PatHen
New York University	CSCI-UA 0436: Computer Architecture	PDA	PatHen
Northwestern University	EECS 361: Computer Architecture 1	PDA	PatHen
Swiss Federal Institute of Technology Zurich	Systems programming and computer architecture	PDA	PatHen
University of California San Diego	CSE 141: Computer Architecture	PDA	PatHen
University of Edinburgh	INF2c: Computer Systems	PDA	PatHen
University of Michigan	EECS 370: Introduction to computer organization	PDA	Others
University of Texas at Austin	CS 352: Computer Systems Architecture	PDA	PatHen
University of Washington	CSE 352: Hardware Design and Implementation	PDA	Others
Massachusetts Institute of Technology (MIT)	Computation Structures	PeA PrA	No textbook
California institute of Technology (Caltech)	EECS 51: Principles of Microprocessor Systems	PrA	Others
Georgia Institute of Technology	CS 2110: Computer Organization and Programming	PrA	Not found
McGill University	COMP 273: Introduction to Computer Systems	PrA	PatHen
Purdue University	CS 250: Computer Architecture	PrA	No textbook
University of Amsterdam	Architectuur & Computerorganisatie BINARCO6	PrA	Others
University of North Carolina Chapel Hill	COMP 541: Digital Logic and Computer Design	PrA	Others
University of British Columbia	CPSC 213: Introduction to Computer Systems	PrA	Others

Table 3.1: Universities, courses, the approach they follow, and the book from our list that they use (PatHen refers to [Patterson and Hennessy, 2009]; Others means that the textbooks they use are not in our list; No textbook means that the syllabus specifies that there is no textbook to be followed; Not found means we have not found in the syllabus the textbook they follow.)

related to performance and data transfer rates, addressing of the I/O registers, and use of the different I/O techniques. There is not reference to manipulation of the I/O registers or programming ISRs.

This book chapter makes a clear introduction to the theory of the I/O subsystem, leaving out external storage, buses, networks, memory system design, and programming specificities.

3.2.2 Computer Organization and Design 4th Edition

This book [Patterson and Hennessy, 2009] also devotes a unique chapter to the I/O subsystem. It is called *"Storage and Other I/O Topics"*. The chapter has lost the networks part in its fourth edition, which means that unlike the CS curricula, this book has decided to treat separately the interfaces and the communications.

This book makes great emphasis in storage and in dependability, reliability and availability, devoting four sections to it.

In section five they start talking about connecting processors, memory, and I/O devices, i.e. section five talks about buses. Section six is the one talking about interfacing I/O devices to the processor, memory and Operating System (OS). For that purpose they define all the concepts such as I/O device or peripheral, memory-mapped I/O and special I/O instructions (used to access isolated I/O), polling, interrupt-driven I/O, interrupt handling (what an interrupt controller does), and DMA.

As for the examples, they are all concentrated on the impact of I/O on the system performance and latency and bandwidth constraints.

3.2.3 Computer Organization and Embedded Systems 6th Edition

Hamacher et al. [2011] spread the I/O subsystem in different chapters of the book. Firstly, chapter three introduces the basics of the I/O subsystem, talking about I/O registers, memory-mapped I/O, device interface or I/O module, polling, interrupts, enabling and disabling them, handling multiple devices, nested interrupts and simultaneous requests, all of it from a very low level point of view, showing block diagrams of the systems and programs in assembly language. Chapter seven *"Input/Output Organization"* presents the need for the buses, the protocols they use, different arbitration techniques and several standards. Chapter ten talks about *"Embedded Systems"* referring also to their I/O subsystems and relating to the concepts seen in previous chapters. Finally, appendixes B, C, D and E present different processor families (Altera NIOX II, Coldfire, ARM, and Intel IA-32), where the I/O operations of their assembly languages are explained.

The exercises at the end of each chapter ask the students to make changes in I/O registers in order to obtain an specific result or block the interrupts of a specific device, etc.

This book does not mention external storage or networks in an specific manner. Also DMA is treated as part of the memory, instead of as part of the I/O subsystem. On the other hand, unlike the books reviewed so far, its perspective is the one of the system programmer at low level.

3.2.4 Computer System Architecture 3rd Edition

Mano [1993] devotes 2 sections and a whole chapter to the computer I/O subsystem. In chapter 5: *Basic Computer Organization and Design*, section 7 talks about *Input-Output and Interrupt*. This first section on the computer I/O presents the need of peripheral devices to get information stored in the computer and get the solutions of the computations to the users. Mano [1993] uses an example of a terminal unit with a keyboard and a printer, to show how the information to be transferred is stored in registers, how special instructions are needed for the I/O process, and how the programmed control transfer is inefficient and the use of interrupts can somehow solve this inefficiency.

The book continuously refers to signals and control bits. In chapter 6: *Programming the Basic Computer*, section 8 talks about *Input-Output programming*. As the name of the section indicates, here Mano [1993] introduces instructions to program the I/O and in the cases where interrupts are used, *Interrupt Service Routines* (ISR) are presented.

Finally, chapter 11: *Input-Output organization* delves deeper into the issue introducing synchronization for the transfer of information, and all the details of the I/O interface, interrupt controllers, interrupt priority, DMA, etc.

3.3 What is the community trying to do in order to improve students' understanding of the I/O topic?

"Computer Science Education Research (CSER) is an emergent area and is still giving rise to a literature." said Fincher and Petre [2004] almost a decade ago. After this, a lot of work has been done, and although CER is still a young research area, several categorizations and overviews have been published [Valentine, 2004; Pears, Seidman, Eney, Kinnunen, and Malmi, 2005; Berglund, Daniels, and Pears, 2006; Randolph, 2007; Simon, Carbone, De Raadt, and Lister, 2008; Simon, 2009; Malmi, Sheard, Simon, Bednarik, Helminen, Korhonen, Myller, Sorva, and Taherkhani, 2010; Kinnunen, Meisalo, and Malmi, 2010, among others], and also several PhD dissertations have been presented [Berglund, 2005; Eckerdal, 2009; Boustedt, 2010; Elliot Tew, 2010; Hewner, 2011; Seppälä, 2012; Sorva, 2012, among others].

Few of the studies performed in CER focus on CA. There is a specific CA education workshop, the Workshop on Computer Architecture Education (WCAE) [[WCAE]]. Most of the papers presented in the WCAE could be classified as "Marco Polo" [Pears et al., 2005], i.e. papers describing experiences and observations after having applied a method, tool, etc. The journal Computer Science Education, Vol.17, Issue 2, is devoted to the teaching of hardware/software fundamentals. Only one paper does not focus on methods of teaching or tools for teaching or for practical exercises for the students. Yehezkel, Ben-Ari, and Dreyfus [2007] use a new tool, but they analyse the students' mental models before and after using the tool and found that the students' understanding was better after using it.

Another issue we have found while searching for bibliography on the I/O subsystem is that often the broad term CA or Computer Organization (CO) is used, but the paper is really about a part of what it is considered as CA and CO within the curricula. For example, in his paper on why and how teach computer architecture in introductory

Book CC2013 (CS) - Topics	**Stallings [2012]**	**Patterson and Hennessy [2009]**	**Hamacher et al. [2011]**	**Mano [1993]**
I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O	7.1; 7.2; 7.3; 7.4	6.6	3.1; 3.2	5.7; 6.8; 11.2–11.4
Interrupt structures: vectored and prioritized, interrupt acknowledgement	7.4	6.6	3.2	11.5
External storage, physical organization, and drives	6	6.3; 6.4	8.10	N/A
Buses: bus protocols, arbitration, direct-memory access (DMA)	3.4; 7.7; 7.5	6.5; 6.6	7; 8.4	11.6
Introduction to networks: networks as another layer of access hierarchy	N/A	6.11 (in CD)	12.3	N/A
Multimedia support	N/A	7.6	E.10	N/A
RAID architectures	6.2	6.9	N/A	N/A
Learning outcomes				
Explain how interrupts are used to implement I/O control and data transfers [Knowledge].	7.4	6.6	3.2	11.4
Identify various types of buses in a computer system [Knowledge].	3.4	6.5	7	4.3
Describe data access from a magnetic disk drive [Knowledge].	6.1	6.3	8.10	N/A
Compare common network organizations, such as ethernet/bus, ring, switched vs. routed [Knowledge].	N/A	6.11 (in CD)	N/A	N/A
Identify interfaces needed for multimedia support, from storage, through network, to memory and display [Knowledge].	N/A	7.6	N/A	N/A
Describe the advantages and limitations of RAID architectures [Knowledge].	6.2	6.9	N/A	N/A

Note. N/A stands for Not Available, and it has been used when to topic referred has not been found in the book.

Table 3.2: Where in each book appear the topics and learning outcomes of CC2013

Book **CC2004 (CE) - Topics**	**Stallings [2012]**	**Patterson and Hennessy [2009]**	**Hamacher et al. [2011]**	**Mano [1993]**
I/O fundamentals: handshaking, buffering	7.1; 7.2	6.6	3.1	5.7; 6.8; 11.2
I/O techniques: programmed I/O, interrupt-driven I/O, DMA	7.3; 7.4; 7.5	6.6	3.1; 3.2; 8.4	11.4; 11.6
Interrupt structures: vectored and prioritized, interrupt overhead, interrupts and re-entrant code	7.4	6.3; 6.4	3.2	11.5
Memory system design and interfacing	6	6.5; 6.6	8	12
Buses: bus protocols, local and geographic arbitration	3.4; 7.4; 7.7	6.11 (in CD)	7	4.3
Learning outcomes				
Explain how to use interrupts to implement I/O control and data transfers.	7.4	6.6	3.2	11.4
Write small interrupt service routines and I/O drivers using assembly language.	N/A	6.5	3.2	6.8
Identify various types of buses in a computer system. data access from a magnetic disk drive.	3.4	6.3	7	4.3
Describe data access from a magnetic disk drive.	6.1	6.11 (in CD)	8.10	N/A
Analyze and implement interfaces.	N/A	7.6	7	N/A

Note. N/A stands for Not Available, and has been used when to topic referred has not been found in the book.

Table 3.3: Where in each book appears the topics and learning outcomes of CC2004

computing, Powers [2004] gives good arguments to answer to the why question. For the how question, he proposes *"The Living CPU"* in which a human representation of the Central Processing Unit (CPU) shows the data path and how to address memory. The I/O subsystem is not mentioned. Elliott Tew, Dorn, Leahy, and Guzdial [2008] present *"results of a comparative study examining student performance in a conventional organization course and in one that has been contextualized using a personal gaming platform as the pedagogical architecture"*. Although the I/O subsystem is considered in the contextualized course, it is not in the traditional, and therefore, the I/O subsystem stayed out of the list of elements used to assess students (i.e. arrays, basic machine models, bitwise operations, character strings, data-types and their hardware representations, memory and pointers, and number base conversions). Another example is the paper called *"Bridges to computer architecture education"* [Marwedel and Sirocic, 2004], where they try to provide bridges in order to motivate students to learn about the internals of computers. They propose two bridges, one for the MESI multiprocessor cache coherency protocol, and the other one for processor pipelines.

The aim of the next subsections is to present the reader with an up-to-date review of the tools being used to teach the computer I/O subsystem in introductory level courses. There is no intention of presenting a comparison of tools. The information used is based solely on material presented in the literature that reports the use of these tools and that are currently available. Previous surveys, such as [Wolffe, Yurcik, Osborne, and Holliday, 2002; Nikolic, Radivojevic, Djordjevic, and Milutinovic, 2009], have not been considered because they do not report the use of the tools used to teach the computer I/O subsystem. Work prior to the year 2000 has not been considered either, because tools for education are expected to be attractive to the students and in that context earlier work cannot compete with newer technology.

3.3.1 Simulators

Simulators have been widely used for teaching computer architecture and organization because *"they allow students to gain valuable hands-on experience in design and programming, without the problems involved in*

approach \book	[Patterson and Hennessy, 2009]	[Stallings, 2012]	[Mano, 1993]	[Hamacher et al., 2011]
purely descriptive	X	X	X	X
performance	X	X		
programming				X
datapath-signal			X	

Table 3.4: Approach followed by each textbook

maintaining a hardware lab" [Donaldson, Salter, and Punch, 2011a] and because they *"give students a better appreciation for the complexity of their own computers"* [Black and Komala, 2011]. As far as the computer I/O subsystem is concerned, the use of simulators usually lies in *the datapath-signal approach* (Subsection 3.1.5 in this document. The simulators show the signals used during the I/O process and show how data is exchanged via registers. Some of them give the option to program a poll or a simple ISR in assembler. It could therefore be said that they also follow the programmming approach.

Scott [2000a] reports the use of a simulator to show the difference between programmed and interrupt-driven I/O. The simulator is based on the simple computer presented in [Mano, 1993]. It was written in Borland C++ by a student for their senior project. A poll or an ISR can be programmed in assembly language (fictitious) and the simulator shows the changing values of the registers and memory while the instructions are executed. The abovementioned paper does not report usage results. The simulator can be found in [Scott, 2000b].

Far newer is [Donaldson et al., 2011a]. In this work Donaldson et al. present an extensible, visual simulation platform that contains plug-in components representing simulated CPUs, memory modules, I/O devices, and other supporting chips. They also present some already implemented circuits such as a single-bus computer with CPU, memory, and I/O console, where the CPU uses polling to interact with the console. Two more consoles can be controlled using interrupts and there is also a DMA-capable disk. The platform contains several plug-ins which simulate I/O devices that could be added to these implementations. They also suggest some exercises. As for the I/O subsystem, they suggest connecting several I/O devices to a single interrupt request line using daisy chain or modifying the CPU plug-in so as to separate the I/O address space from memory address space. The platform can be found in [Donaldson, Salter, and Punch, 2011b]. The system was not tried out with the students at the time of publication, and the authors of this paper could not find any later data.

Finally, Black and Komala [2011] report the building of a simulator after finding that in existing simulators "*peripherals and I/O are generally neglected*" and they do not allow students to run existing software, to develop assembly software that can run natively on their own computers, to observe the interplay between devices, and generally, to see how their actual personal computer works. In contrast, the simulator reported in [Black and Komala, 2011] "*has been tested with MS-DOS, FreeDOS, Windows 3, and Minix 1*". It is written in Java and can be downloaded as a single JAR file or run from a web-page as an applet. For the I/O subsystem they have developed a lab where *"students write a simple graphical animation program (a moving car) in x86 assembly. This involves configuring the interval timer, modifying the interrupt vector table and making their own timer interrupt routine, switching the video table and making their own timer interrupt routine, switching the video to graphics mode, and writing to video memory. Because the simulator accurately models a PC the students can then run the same program on their own laptops."* Later, Black and Waggoner [2013] report the integration of the x86 full system simulator, with a simulator to teach CPU design, a data-path builder, a control builder and a processor wizard. In this case the simulator was used so that the students could learn about finite state machines by designing an elevator controller and a traffic light controller, and then simulate their own RISC processor. Black and Waggoner [2013] report usage experience from the point of view of the students, but does not report learning benefits. The simulator can be found at [Black, 2013].

3.3.2 Real Machines

Despite the extensive use of simulators, some authors argue that *"students must touch, feel and smell the real hardware in a computer organization course"* [Brorsson, 2002a] and that the emulator or simulator approach, which is the most extensive one, *"fails on concreting theoretical concepts into real ones"* [Santofimia and Moya, 2009]. This section will survey the use of real machines for teaching the computer I/O subsystem.

Ellard, Holland, Murphy, and Seltzer [2002] present the Ant-32 architecture. This is a 32-bit RISC architecture designed specifically for educational purposes. They do not use a real architecture, arguing that they are *"too complicated and require mastery of too many arcane details in order to accomplish anything interesting"*. The implementation of the architecture they present uses a simple (but full-featured) bus architecture that was originally designed for use with the MIPS processor architecture, which allows the use of simulators for devices already written for that bus. Interrupts and exceptions are enabled and disabled via special instructions. Interrupts from external devices are treated as a special kind of exception. Interrupts can be disabled independently of exceptions. A tutorial for the Ant-32 assembly language can be found in [Ellard, 2003].

Brorsson [2002a] presents the MipsIt system, which consists of a development environment, a hardware platform and a series of simulators, targetted at the Windows (95-XP) platform as host machine. The evaluation board (containing an IDT 36100 micro controller with a MIPS32 ISA processor core) and the simulators are carefully explained in the above-mentioned paper. As for the I/O subsystem he developed *"a simple daughter board containing one eight-bit and one 16-bit parallel bi-directional I/O port"*, which *"also contains a simple interrupt unit with three interrupt sources (two push-buttons and one adjustable pulse source) that could also be read as a six-bit parallel input port"*. There is also a simulator of the whole system which can be found in [Brorsson, 2002c], and the lab exercises proposed for the I/O subsystem topic can be found in [Brorsson, 2002b].

In Teller, Nieto, and Roach's [2003] proposal, *"after acquiring some competence at programming the 68HC11"*, using the Visual 6811 simulator (a simulator for the Motorola HC6811 microprocessor), *"students are challenged to program small robots that are controlled by 68HC11s"*. *"The first robot lab has students use the serial communication interface to display the contents of the robot's memory (the 68HC11) on the monitor of the host PC. Pressing a key on the host's keyboard, which instructs the 68HC11 to read from or write to a memory byte or word, drives the related program" [. . .] The first final project had students program robots to* *navigate a maze. [. . .] Another challenging final project was to program two robots equipped with IR detectors: a wimp and a follower."*

In [Teller et al., 2003] the authors assessed the use of the tools via questionnaires, where they discovered that the majority of the students thought it was a good idea to use the robots in the course and that programming the robots helped to reinforce the concepts they learned and helped them to see (physically) the power of the 68HC11 architecture in action.

Brylow and Ramamurthy [Brylow and Ramamurthy, 2009a] report the use of the Linsys WRT54GL family of wireless routers in two different courses. In this paper the focus will be on the sophomore level course at Marquette University. The system *"contains a little-endian embedded MIPS 32 processor [. . .], with 16 MB of RAM, and 4MB FlashROM. [. . .] easily accessible serial port connections on the main board that allow direct access to the device firmware."* Among the exercises they include in for a sophomore course on embedded systems, the two most related to the I/O subsystem state the following: *"The third laboratory exercise, writing and testing a basic device driver, begins the experiences specifically aimed at building embedded systems design competence. Rather than view a device driver through the lens of a heavily abstracted operating system layer, students work directly with the memory-mapped control and status registers of a serial device to build input and output primitives that they will reuse for the remaining of the term. This assignment serves not only to practice reading and producing technical documentation for a peripheral I/O device, but also to use C language struct pointers to manage interaction with external devices, a common motif in embedded systems. [. . .] The eighth exercise extends the simple, synchronous serial driver to build a fully-buffered, asynchronous serial driver. Working directly with I/O interrupts, hardware FIFOs and multiple devices, students confront the genuine complexity of interaction between embedded processors and their peripherals. A modern serial UART is itself best described by a finite state machine with some timed transitions, and a simple teletype (TTY) protocol on top of this adds another state machine at a different level of abstraction."* This study again reports students' satisfaction, but contains nothing on learning outcomes. More information on this project can be found in [Brylow and Ramamurthy, 2009b].

Santofimia and Moya [Santofimia and Moya, 2009] report changing the PIC16F84 for a Nintendo DS (NDS) gaming console. In this case, the interrupt concept is introduced via timers and scrolls. *"Implementation of counters can be accomplished by using timers"*. Moreover, interrupts generated when line drawing are necessary to display new data on the screen. Students were given the option to choose between PIC16F84 and NDS. *"Over thirty percent of the students chose the NDS platform, what provided an acceptable feedback for comparing result regarding the acquired knowledge."* The data gathered while using the NDS showed *"an important improvement in terms of the contents understood and retained by students."*

Larraza-Mendiluze, Garay-Vitoria, Martín, Muguerza, Ruiz-Vázquez, Soraluze, Lukas, and Santiago [Larraza-Mendiluze et al., 2013] also use NDS in their approach, in this case switching from a PC-based laboratory to a NDS-based laboratory. The paper has a section that *"shows the extent to which the NDS I/O subsystem was used in [the] study and explains the various elements involved in the I/O process."* The paper shows data gathered in two subsequent years, where the experimental group used the NDS in both years and the control group used the PC during the first year and NDS during the second year. While in the first year the scores obtained in the control group were lower than those in the experimental group, the year in which both groups used the NDS the scores were almost the same. It is noteworthy that the experimental group also had lower drop-out ratios.

3.3.3 Using different educational approaches

Martínez-Monés, Gómez-Sánchez, Dimitriadis, Jorrín-Abellán, Rubia-Avi, and Vega-Gorgojo [2005] propose the use of multiple case studies to enhance PBL in a CA course. It looks like a good approach where they carry a quantitative and qualitative evaluation of the results for four years, concluding that *"students generalize well their acquired knowledge"*. However, in regard to the I/O subsystem, they have to resort to an analytical study *"because of the lack of simulators"*.

Ramachandran and Leahy Jr [2007] propose an integrated approach to teaching computer systems architecture, teaching in the same course CA and OS. One of the five modules they propose is devoted to the

I/O subsystem and software concepts related to devices and device controllers. Although they explain the project students would develop during the I/O module, the paper does not refer much about how students understand the topic.

Teller et al. [2003] report the implementation of a course using participatory learning strategies. First of all students learn to program a specific processor using a simulator and then students are challenged to program small robots that are controlled by the learned processor. In order to control the robot the students are required to *“download programs, via the serial communication interface, to a 68HC11 microprocessor. These programs interface with the 68HC11’s I/O ports and permit students to analyse the behaviour of, among other things: memory-mapped I/O, the serial communication interface, motors connected to the I/O ports, digital and analogue sensors, programmable timers and counters, and interrupts.”*. This paper has been reported in Subsection 3.3.2, however, it has also been included here because the authors stress from the beginning and during the whole paper the participatory learning strategies used.

3.4 The I/O topic at several Spanish universities

For this section only Spanish universities have been considered because the similarities in the syllabus of these different universities made it easier to locate the I/O topic within the subjects. The universities considered are University of Castilla-La Mancha (UCLM), University of Granada (UGR), Technical University of Catalonia (UPC), Technical University of Madrid (UPM), Polytechnic University of Valencia (UPV), UPV/EHU, and Unizar. Table 3.5 shows, for each university that have been considered, the subject in which the I/O subsystem is treated, the semester in which the mentioned subject is taught, the number of ECTS credits of the subject, the books they propose among the books we have referred in section 3.2 (it is possible that the edition does not concur), and the syllabi that these universities propose for the I/O subsystem topic. The subject whole programs for each university can be found at [UCLM, 2013; UGR, 2013; UPC, 2013; UPM, 2013; UPV, 2013; UPV/EHU, 2013; Unizar, 2013].

University	Subject Name	Semester	# of credits	References	Syllabus
UCLM	Computer Structure	2	6	Patterson and Hennessy Stallings	Input/Output system - I/O modules - I/O modes: programmed, interruptions and DMA - Buses
UGR	Computer Structure	3	6	Hamacher et al. Stallings Patterson and Hennessy	Input/Output and buses - I/O system funtions. I/O interfaces - Programmed I/O - Interrupts - DMA (Direct Memory Access) - Basic bus structures - A bus specification: Transfer. Timing. Arbitration - Examples and standards
UPC	Computer Structure	1 or 2	7.5	Patterson and Hennessy	Exceptions/Interruptions - Basic concepts and hardware support for MIPS - An exception's detailed functioning and example of a generic service routine - Specific cases: TLB faillure. System requests. Interruptions
UPM	Computer Architecture	3 or 4	6	Patterson and Hennessy	Input/Output system
UPV	Computer Structure	3 and 4	9	Patterson and Hennessy Stallings Hamacher et al.	Input/Output unit - I/O adaptors and interfaces - I/O synchronization mechanisms - I/O transfer techniques
UPV/EHU	Computer Structure	2	6	Hamacher et al. Stallings Patterson and Hennessy	The I/O subsystem - Description of the I/O interface - Communication and synchronization: polling and interrupt - Peripheral management in the Nintendo DS - DMA: Direct Memory Access
Unizar	Architecture and Computer Organization 1	2	6	Stallings	The I/O subsystem - Generic model of device controller registers - Basic methods of synchronization and transfer - Exceptions - Integration of peripherals in microcontrollers

Table 3.5: How the I/O topic is taught in several Spanish universities

3.5 The approach taken at the University of the Basque Country

At the University of the Basque Country (UPV/EHU), when the topic was taught in the third semester, all the approaches were included. The purely descriptive approach was overrun by the practical part. The programming approach was used in labs and also in paper exercises. Paper exercises were also used to work on the performance of the computer system. Finally, a simulator was developed in a graduate thesis that was used in lectures to show the datapath-signal approach [Garay, Larraza, Martín, Ruiz, and Soraluze, 2010].

However, when the EHEA was introduced and the curricula changed, the time devoted to the subject was reduced significantly. This made impossible covering all the approaches. The programming approach was selected because the performance of the computer systems is treated in other subjects, and the datapath-signal approach was considered only to help understanding the functioning of the I/O subsystem in an undergraduate introductory course.

3.6 Conclusions

In this chapter, a survey related to the educational process for the computer I/O subsystem at undergraduate introductory level is presented. Textbooks, course syllabi, and conference and journal papers were surveyed. In the analysis of textbooks, course syllabi of universities classified in high positions in different university rankings and the authors previous work, it was found that there are different educational approaches in regard to the computer I/O subsystem at undergraduate introductory level.

Pros and cons of the four different educational approaches have been given in order to help future lecturers choose. The purely descriptive approach stays at a low level knowledge taxonomy stage, being accessible for students of different majors. As for the other approaches, performance, although an important issue in the I/O subsystem, is not unique to this topic and can be worked in other subjects. The datapath-signal approach can help understanding the topic, but does not deepen in the knowledge taxonomy. Finally, the programming approach can be difficult to deal with when the students are not very good at programming, but is an approach where abstract concepts can be very well treated.

Also research literature has been analysed. In conference and journal papers related to teaching the computer I/O subsystem, several tools (such as simulators and real machines) are presented. In some cases little data are provided about the use of these tools in the classroom. However, there is a lack of references that show what students have learned after using the aforementioned tools. Moreover, it has to be said that most of the references found in the literature show single experiences that generally lack continuity over several years.

The survey was limited to introductory courses that are taught at undergraduate level. It would be interesting to analyse how the educational process for the computer I/O subsystem evolves in more advanced courses.

CHAPTER 4

Deciding on the educational infrastructure

DURING many years the laboratories to endorse the learning process of the computer I/O subsystem at the UPV/EHU have been developed on PCs. In these labs students used to access registers mapped in memory or in the I/O space, program polls to serve device requests and control the interrupt vector for accessing their own interrupt service routines. The laboratory sessions based on PCs are detailed in Subsection 5.1.2 of this disseretation.

Although the opinion of the course lecturers is that these laboratory sessions are of great help for the assimilation of the subject material, a progressive decrease has always been observed in the attendance.

Moreover, operating systems, in the pursue of security, started making use of the different accessing modes of the hardware such as user-mode or kernel-mode [Nutt, 2004]. This way some of the elements that need to be accessed for the control of peripherals stopped being directly accessible. They were only accessible through the kernel or nucleus of the operating system. The easiest overcome to the problem was to use virtual machines with old operating systems, working over the machine in full mode. However, using out of date operating systems is not attractive for the students and we started looking for a new infrastructure for our labs. The idea of more attractive labs brought us to search among handheld game consoles.

The rest of this chapter is devoted to show the path between this search to the end using the Nintendo®DS (NDS) console. In the first section of this chapter (section 4.1) the analysis of several handheld consoles will be shown. The second section (section 4.2) will introduce the NDS machine. The third section (section 4.3) will describe the peripherals used in the new laboratories and how they have been adapted for their use in the course. Finally section 4.4 will make some conclusions on the use of the NDS handheld console during the educational process of the computer I/O subsystem in undergraduate introductory courses.

What we present in this chapter has previously been partially published in [Larraza-Mendiluze, Garay-Vitoria, Martín, Muguerza, Ruiz-Vázquez, Soraluze, Lukas, and Santiago, 2012] and [Larraza-Mendiluze et al., 2013].

4.1 Selection of the handheld game console

An extensive analysis of handheld game consoles is out of the scope of this work. However before starting we did roughly analyse some of the (at that moment) known handheld consoles: the PlayStation® Portable (PSP), Nintendo® Dual Screen (NDS), PANDORA, and GP2X. The reasons that brought us to continue working with the NDS where the following:

- At that time, while version 5.53 of the firmware for the PSP had been released, homebrew applications could only be run under versions 1.0 or 1.5 of the firmware [Accarrino, 2005; Torrone, 2005]. Probably later updates have been released, but at that time it was a problem.

- PANDORA being a console running under open source software was a very attractive option. However, it is very difficult to obtain [Pandora]. Although it may be possible to obtain one for the development of the system this could generate problems when it comes to testing the system with the students.

- GP2X also runs on open source software. The main problem with this console at the time of making the selection for this project was its lack of connectivity. No Wifi is incorporated to allow the consoles to be connected (not even in the newest version at the moment-GP2X Wiz7) [GP2Xb]. At that time there was another project of a collaborative system for learning and this lack was a problem. A wifi possibility has been later added to the GP2X Caanoo [GP2Xa].

- The price of the consoles was also considered and in that aspect, the NDS was the most competitive one.
- All other consoles have been ruled out due to the lack of support they provided.

4.2 The NDS console

Although the main objective of this study was to adapt the subject to the use of the NDS machine, sometimes the functioning of the system needed to be adapted to the situation (the development was to be done by first year students). A few drawbacks needed to be solved and therefore a template was handed out to the students with the short-cuts and other system specifications that were not directly related to the I/O subsystem. This subsection is intended to show the extent to which the NDS I/O subsystem has been used in the Computer Structure course at the UPV/EHU. The different elements involved in the I/O process will be explained in this subsection.

It must be borne in mind that in order to program the NDS machine the devkitPro toolchains [devkitPro] have been used and therefore, some differences could appear when some other toolchains are used.

4.2.1 The memory

It has to be considered that the NDS machine's objective is gaming. For the last generation games the graphical interface is very important. It needs 2D and 3D rendering, and many other aspects that are out of the scope of the subject. As a result, the memory system is not as straightforward as it could be desirable in this context, see figure 4.1. Although

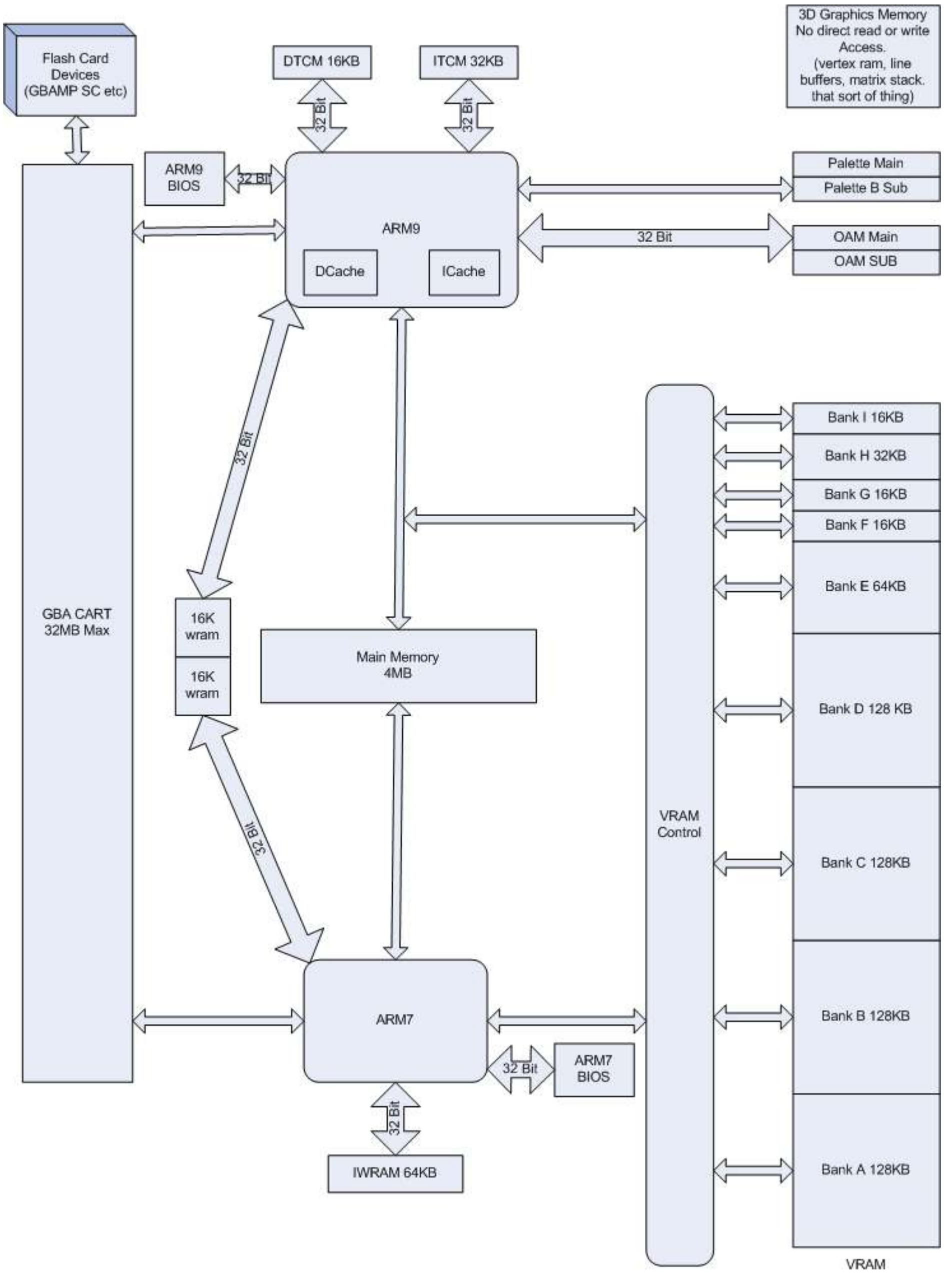
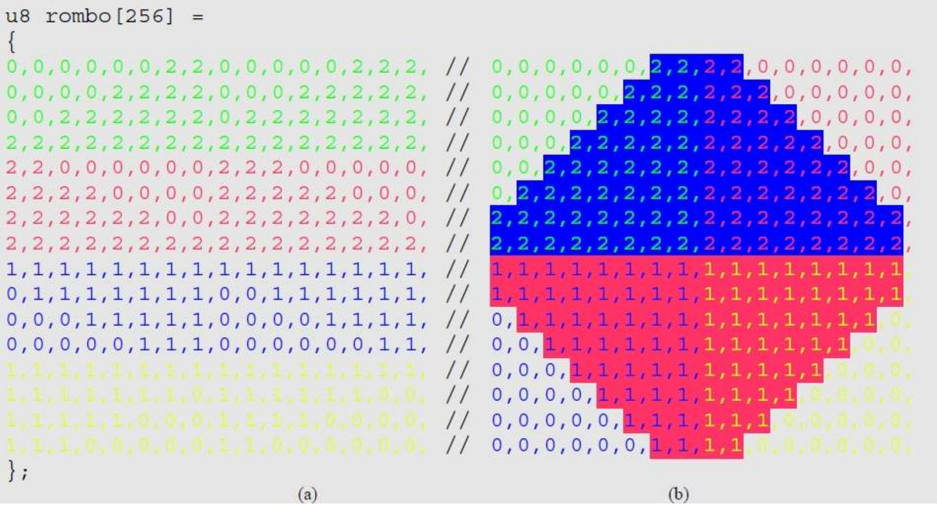


Figure 4.1: Nintendo DS memory layout
Source: http://dev-scene.com/NDS/Tutorials_Day_2#Memory_Layout. Retrieved in 11/25/2013

```
void initVideoMemory() {
    /* Map memory in order to show images in both displays. */

    vramSetMainBanks(VRAM_A_MAIN_BG_0x06000000,
                   VRAM_B_MAIN_BG_0x06020000,
                   VRAM_C_SUB_BG_0x06200000,
                   VRAM_E_LCD);

    vramSetBankE(VRAM_E_MAIN_SPRITE);
    vramSetBankD(VRAM_D_SUB_SPRITE);

    /* Set two modes for main memory */
    videoSetMode(MODE_5_2D | // set 5th video mod
                DISPLAY_BG2_ACTIVE | // activate 2nd background
                DISPLAY_BG3_ACTIVE); // activate 3rd background

    /* Set video mode for 2. level display */
    videoSetModeSub(MODE_5_2D | // set 5th video mod
                  DISPLAY_BG3_ACTIVE);
    // activate 3rd background
}
```

Figure 4.2: Source code of the routine for initializing video memory, from examples of Dovoto and Jaeden Ameron in the devkitPro environment [devkitPro]



```
u8 rombo[256] =
{
0,0,0,0,0,0,2,2,0,0,0,0,0,2,2,2,  //  0,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,
0,0,0,0,2,2,2,2,0,0,0,2,2,2,2,2,  //  0,0,0,0,0,2,2,2,2,2,2,0,0,0,0,0,
0,0,2,2,2,2,2,2,0,2,2,2,2,2,2,2,  //  0,0,0,0,2,2,2,2,2,2,2,2,0,0,0,0,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,  //  0,0,0,2,2,2,2,2,2,2,2,2,2,0,0,0,
2,2,0,0,0,0,0,0,2,2,2,0,0,0,0,0,  //  0,0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,
2,2,2,2,0,0,0,0,2,2,2,2,2,0,0,0,  //  0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,
2,2,2,2,2,2,0,0,2,2,2,2,2,2,2,0,  //  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,  //  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,  //  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,  //  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
0,0,0,1,1,1,1,1,0,0,0,0,1,1,1,1,  //  0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,
0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,  //  0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,  //  0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
1,1,1,1,1,1,1,0,1,1,1,1,1,1,0,0,  //  0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,
1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,  //  0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,
1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,  //  0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,
};
```

(a) (b)

Figure 4.3: Definition of a sprite, (a) array containing the value of each pixel, (b) interpretation of the information contained in the array.

the aim is to get to the systems low level, some library functions have been used to overcome that problem (see Figure 4.2).

The students are required to draw a sprite pixel by pixel and store it in memory, so that they can see that changing the content of a memory location changes what they get on the screen. As an example of how to do that, the code in figure 4.3(a) (an array of 256 positions) defines the diamond that can be seen in figure 4.3(b) (the number stored in each position of the array defines a color of the sprite. The first 64 positions of the array define the upper left quadrant of the sprite, an so on). Using the functions of figure 4.4 the sprite is stored in memory.

4.2.2 The double processor

Another drawback of the NDS for the purpose of this experience is that it has two processors, one mainly devoted to the graphic processing and the other one mainly devoted to the I/O subsystem. The conversational process between the two processors is not the objective of the subject and, therefore, the templates handed out to students make

```
u16* gfxerronbo;

void SpriteakMemorianGorde(){

int i;
       for(i = 0; i < 16 * 16 / 2; i++)
       {
              gfxerronbo[i] = rombo[i*2] |
                 (rombo[(i*2)+1]<<8);
       }
}
```

Figure 4.4: Source code for saving the sprite in memory.

some arrangements so that they can work as if they were using just one processor.

Only the graphical processor (necessary to get something on the screen) is programmed. Luckily, this processor also has access to the timers' and keyboards' registers, and some code in the templates grabs the data about the touch screen from the other processor.

4.2.3 The I/O registers

The I/O registers are means by which the peripherals and the processor communicate with each other and are one of the main concepts to learn in the I/O subsystem topic. I/O registers can be mapped into memory or into the I/O space. The latter is a more secure method and is that mainly used in PCs. However, a gaming console is less prone to attack and I/O registers need to be easily accessible, since peripherals are widely used. These could be the reasons why the NDS console has all its I/O registers mapped into memory. The registers to be used in the lab sessions are defined so that the students have no need to use the addresses all the time (see Figure 4.5). Remember that the registers of the touch screen are not directly accessible from the processor being used. Section 4.3.3 will show how touch screen data is accessed.

```
//Registers for interrupt control

#define IME           (*(vuint32*)0x04000208)
//Interrupt Master Enable
#define IE            (*(vuint32*)0x04000210)
//Interrupt Enable
#define IF            (*(vuint32*)0x04000214)
//Interrupt Flag

//Registers for the keyboard
#define TEKLAK_DAT    (*(vu16*)0x4000130)
//keyboard data register
#define TEKLAK_KNT    (*(vu16*)0x4000132)
//keyboard control register

//Registers for timer0
#define DENBO_DAT  (*(vuint16*)0x04000100)
//timer0 data register
#define DENBO_KNT (*(vuint16*)0x04000102)
//timer0 control register
```

Figure 4.5: Definition of the I/O registers.

```
void interrupts()
{
        irqSet(IRQ_TIMER0, DenbEten);
        irqSet(IRQ_KEYS, TekEten);
}
```

Figure 4.6: Setting interrupt service routines for each interrupt

4.2.4 The interrupt management

There is a processor devoted solely to the inputs and outputs and, therefore, almost all the I/O is controlled by polling. In this way polling becomes more natural and therefore easier to understand for the students. However, it is also possible to control some peripherals by interrupt. A software interrupt controller called "*The interrupt dispatcher*" can be found in the devkitPro toolchain [devkitPro], *libnds* library. This is programmed in assembler and it is therefore worth for reviewing the concepts explained in the first part of the subject.

In the lab sessions the interrupt controller is used as it is, although there is always the possibility to change it. As it is:

- the programmer sets the priorities of the interrupts, which will be the same as the order in which the ISRs for each Interrupt Request (IRQ) are set (e.g. in figure 4.6 timer0's interrupts will have more priority than keyboard's interrupts because they are defined first).
- only single-level interrupts are accepted; i.e. an interrupt cannot be interrupted (see figure 4.7).

When coding it is also possible to use the Interrupt Master Enable (IME) register to accept or deny interrupts in general, and the Interrupt Enable (IE) register in order to accept or deny specific interrupts. The Interrupt Flag (IF) register is only used by the interrupt controller or dispatcher to know the interrupts that have been requested.

```
mov    r12, #0x4000000
// base address for registers for interrupt control

ldr    r1, [r12, #0x208]     // r1 = IME
cmp    r1, #0
// if interrupts are disabled execution ends
bxeq lr

mov    r0, #0
str r0, [r12, #0x208]
// otherwise, value 0 is stored in IME, disabling interrupts
```

Figure 4.7: Code in the *Interrupt dispatcher* that avoids multiple level interrupts.

$$latch = 65536 - \frac{1}{interruptfrequency} * countingfrequency$$

Figure 4.8: Formula to calculate the value of the timer data register

4.3 Some of the NDS peripherals

4.3.1 The timer

The NDS console has four timers for each processor. These timers are 16 bit programmable counters. These counters can be stated on by activating bit 7 of its control register. Bit 6 is used to tell the timer whether it has to produce an IRQ when an overflow occurs. The counting frequency is also programmable using bits 0 to 2 and, therefore, the interrupt frequency is controllable. It is also possible to connect the timers with each other in order to have a bigger amount of interrupt frequencies (see table 4.1).

The data register can be used to establish the number from which the counter will start counting. The formula used to calculate this initial value can be seen in figure 4.8.

In the lab sessions the students have to program at least one timer and code the ISR that will be run when the timer interrupts.

Bit	Value	Definition
7	1	Activate timer
6	1	Generate interrupt when overflow
2	1	Connect timers. Start counting when previous timer overflows. Can not be used in timer0
0-1	0	Frequency divider 1 (Max.F. 33554432 Hz – Min.F. 512 Hz.)
0-1	1	Frequency divider 64 (Max.F. 524288 Hz – Min.F. 8 Hz.)
0-1	2	Frequency divider 256 (Max.F. 131072 Hz – Min.F. 2 Hz.)
0-1	3	Frequency divider 1024 (Max.F. 32768 Hz – Min.F. 0.5 Hz.)

Table 4.1: Explanation of the bits in the control register of the timer

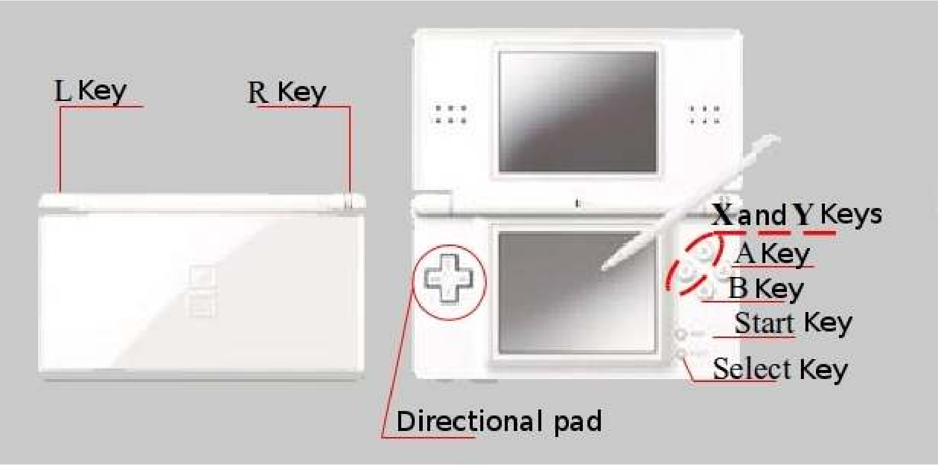


Figure 4.9: Keyboard of the Nintendo DS

4.3.2 The keyboard

The NDS has only 12 keys, as can be seen in Figure 4.9. Two of these keys, the X and Y keys, are controlled separately (only accessible from the processor that is not being used during the lab sessions). Therefore only the other 10 keys are to be used.

The data register is 16 bits long, and uses the bits 0 to 9 to tell which key or keys have been pressed. The control register is also 16 bit long. Bit 14 can be used to tell whether the keyboard will interrupt. If not, the data register can also be read by poll to determine the pressing of a

Bit	Definition
0	A key
1	B key
2	Select key
3	Start key
4	Right key (directional pad)
5	Left key (directional pad)
6	Up key (directional pad)
7	Down key (directional pad)
8	R key
9	L key
10-13	not used
14	Activate interrupts (1 active / 0 inactive)
15	Interrupt condition (1 AND / 0 OR)

Table 4.2: Explanation of the bits in thecontrol register of the keyboard. Bit 0 to 9 are the same in the data register.

key. Bit 15 of the control register is used to tell whether the interrupt will be caused by the pressing of a single key or whether several keys must be pressed in order to interrupt. Bits 0 to 9 can be used to say which keys are allowed to interrupt and finally bits 10 to 13 are not used (See Table 4.2).

During the lab sessions the students have to control some keys by polling and others by interrupt, setting the control register correctly for this and coding the polling and the ISR.

4.3.3 The touch screen

This peripheral is only accessible from the processor that is not being controlled in this context. However, it was also desirable to cover it because otherwise the work to do would be too restrictive for the students. The way of overcoming this problem was to make an abstraction of the communication process. The routine in the *libnds* library used can be seen in figure 4.10.

The students only need to define a variable of the `touchPosition` type, as in figure 4.11. This variable is a two value datum (the x and

```
//------------------------------------------
void touchRead(touchPosition *data) {
//------------------------------------------

       if ( !data ) return;

       data->rawx = __transferRegion()->touchX;
       data->rawy = __transferRegion()->touchY;
       data->px = __transferRegion()->touchXpx;
       data->py = __transferRegion()->touchYpx;
       data->z1 = __transferRegion()->touchZ1;
       data->z2 = __transferRegion()->touchZ2;
}
```

Figure 4.10: Routine that takes the information of the touch screen from the processor devoted to I/O

```
       touchPosition pos_screen;
```

Figure 4.11: Definition of a variable of the `touchPosition` type.

y coordinates of where the screen was touched). If these two values are 0, it means that the screen has not been touched and, therefore, the polling must carry on until these values are different to 0, as in Figure 4.12.

During the lab sessions the students have to program the above-mentioned polling, and combine it with the polling for the keyboard.

4.4 Conclusions

The aim of this chapter has been to show the characteristics of the Nintendo DS machine that make it useful for a computer I/O subsystem course in an undergraduate introductory level. The Nintendo DS is a bare machine, i.e. it has no operating system. Its I/O registers are memory-mapped. Its devices are usually polled for I/O. However, it offers the possibility of managing interrupts in some cases. It has an

```
        while(1)
        {
                touchRead(&pos_screen);
                while(pos_screen.px==0 && pos_screen.py==0)
                        touchRead(&pos_screen);
                [...]
        }
```

Figure 4.12: Polling the touch-screen

interrupt controller called the *Interrupt Dispatcher*, which shows clearly how the interrupts are controlled in ARM assembly language.

In this chapter, it has been explained how to use the timer, the keyboard, and the touch-screen. The specifics of the DMA have, for the moment, been left out of the scope of this study, but they will be introduced in the future, since it is another possibility offered by the NDS machine.

The next chapter will explain how NDS has been used in class along three consecutive school years.

CHAPTER 5

The Project Based Methodology applied to the computer I/O subsystem education: a three year outline

WHILE in chapter 4 we have analysed how to use the Nintendo DS machine as infrastructure for the educational process of the computer I/O subsystem in undergraduate introductory courses, this chapter will analyse how to improve students' learning process from a methodological point of view.

Project-based learning (PBL) is gaining confidence as a teaching method over more traditional methodologies in which lectures and traditional practical exercises prevail and students passively collect the information that the lecturer transmits. In contrast, in PBL, the students themselves take responsibility for their learning process and become promoters of their own learning. They have to develop a project that is relevant to the real world. They need to obtain the relevant information and learn all of the necessary concepts while the lecturer provides guidance and counselling in the process (i.e., they learn the concepts by doing). Thus, the project stimulates learning. Moreover, PBL is a learning methodology based on collaboration. Students have to work in teams, which forces them to develop other skills [Thomas, 2000].

There are many literature references on PBL in engineering education [Díaz Lantada, Lafont Morgado, Munoz-Guijosa, Muñoz Sanz, Echávarri Otero, Muñoz García, Chacón Tanarro, and De La Guerra Ochoa, 2013; Kali Prasad, 2013] that stress its importance and effectiveness. Related to computer science and computer engineering, in [Ramachandran and Leahy Jr, 2007] they provided a good list of possible projects to carry out in the area of computer architecture. [Berglund and Eckerdal, 2006; Martínez-Monés et al., 2005; Stanley, Wong, Prigmore, Benson, Fishler, Fife, and Colton, 2007; Urness, 2007] describe interesting implementations of the PBL methodology in the area of computer architecture and systems.

With all that in mind, we modified our teaching methods by transitioning from classical lecture-based passive classes to PBL active methods. As explained in chapter 4 we also decided to use the NDS for practical work, a machine that matches perfectly with the objective of doing attractive projects.

Section 5.1 will be used to describe the old approach in order to make the reader aware of the change. Section 5.2 will describe the first setting of the PBL approach, followed by the changes applied to this setting in the following years. Finally, Section 5.3 will show an example of project developed by the students from each school year. The data obtained in terms of grades and satisfaction will be described in Chapter 6.

5.1 The previous methodology

The conventional teaching methodology was based on lectures in which several topics were explained, including: the need for I/O controllers; how these controllers interact with the CPU via registers that may or may not be memory mapped; different synchronization options (polling and interrupt), DMA and how the machine performs differently with the use of the different synchronization options. All this theory was supported by paper problems and Personal Computer (PC)-based laboratory sessions. These, and the related assessment method will be explained in the next three subsections.

5.1.1 Paper problems

Paper problems in the computer I/O subsystem topic were assignments in which a new PC-controlled system had to be developed. This system had several peripherals other than the standard ones explained in the classroom (screen, keyboard and clock). Students had to design a State Machine (SM) that would perform the requirements of the assignment. Then they had to write the program needed to achieve a useful system. The program should include the main program, the polling (if required) and all the required ISRs [Garay et al., 2010].

Typically, the peripherals used in these problems were not real, and considering that students were only in their second year, the situations were simplified; however, they showed that peripherals can be controlled in very different ways and demonstrated that a link with the real world can be established. Several of these paper problems used to be done and corrected in class.

However, it is not easy to understand how everything happens, such as when an ISR is going to be executed or what a strobe sequence is for. The lab sessions explained in the next subsection were designed to enhance understanding of these details.

5.1.2 PC-based lab sessions

Four or five lab sessions were conducted while the computer I/O subsystem was taught. The aim was to teach students to directly control the standard peripheral devices of PCs (i.e. the screen, keyboard and clock). All these labs can be found in [Garay et al., 2010].

The first lab session was used to learn the control of a text screen, a memory mapped peripheral. Also the cursor had to be controlled, which is not mapped in memory. Students had to program low-level routines to:

- write a character in a specific point in the screen;
- erase the whole screen;
- read the character of a specific point in the screen;
- make a scroll;
- set the cursor in a specific point; and

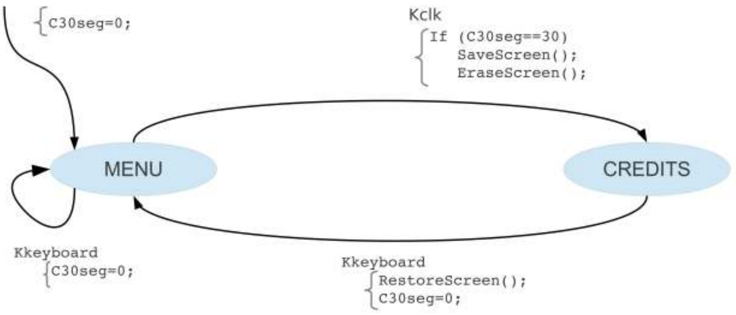


Figure 5.1: The SM used in the fourth laboratory session (a screen saver)

- read the position of the cursor and to establish the shape of the cursor.

These exercises were designed to help them understand how to work with memory mapped devices and show text and cursor wherever they wanted in the screen.

In the second lab session, the students would control the keyboard by polling, considering that the keyboard on a PC does not have a status register, and therefore is usually controlled by interrupt. In order to control the PC keyboard by poll, interrupts from the keyboard should be disabled and then, poll the Interrupt Request Register (IRR) of the Interrupt Controller (IC).

Controlling the keyboard via interrupt, and the difference between both, polling and interrupt, was addressed in the third lab session.

The fourth lab introduced the clock and the concept of real-time processing by means of a screen-saver. This lab session put students, for the first time, in front of a simple but real SM (see Figure 5.1). Students had to program low-level routines to save and restore the screen, to update the time and draw it on the screen, and to use all of them in the clock service-routine.

The fifth session was used to put everything together, programming another clear screen function operated by the combination of some keys. Usually not all the students were able to finish the first four lab sessions in time and many used the fifth session to finish the previous lab assignments.

5.1.3 The assessment

The assessment of the topic consisted of a closed book paper final exam. The computer I/O subsystem took a 45% of the whole exam. 10% of the score usually was devoted to theoretical questions, 5% to the lab sessions (could be graded either in the labs, if attended, or in the exam), and 30% a paper problem (brand new problem, but with the same structure).

This last paper problem in the final exam became a problem on itself. Although we were not able to prove it, we constantly felt that the students learned the structure of the problems and were able to solve them without understanding the basics of the computer I/O subsystem.

5.2 The new PBL methodology

The directed PC lab sessions, where all the students had to do the same lab assignment were replaced by a project in a PBL environment. This project must be defined by each team in a specified context.

PBL is a collaborative learning method with two important features: positive interdependence and individual accountability [Johnson, Johnson, and Holubec, 1998].

Positive interdependence requires that all members of a working team participate actively and contribute to the project, avoiding that a member absent himself from the team activity. An adequate working load is essential to reach this objective. However, in a environment that was new for the lecturers, it has been very difficult to measure this load and define an even work load for all the students. Subsections 5.2.1, 5.2.2, and 5.2.3 describe, among other parameters, how the team structure and working load has been adapted along the three years, in order to better satisfy the project needs.

Individual accountability requires that all the team-mates learn with the project all the topics involved. In this case this is not ensured with an even work load because the project has a contextualization work that could or could not be evenly distributed. For example, one of the team-mates could be doing all the images design leaving its part of the

I/O control to someone else. For this reason we decided that all the students had to take an examination and get a minimum mark on it, in order the project score to be taken into account in their final evaluation.

In adhering to PBL methodology, the leadership that a lecturer takes in the classroom is minimised and reserved only for moments where a special explanation is necessary. The rest of the time, the students' work is crucial and performed either individually or in teams with the lecturer playing the roll of counsellor.

Plagiarism is another issue overcome with this kind of projects. Homebrew games, which code could be found online are never programmed in such low level. They always use libraries for doing what the students in this subject are asked to do, controlling the I/O from and towards the peripherals. The problem comes when all the teams are asked to develop the same project, sometimes even year after year. There have been cases in which the students have changed the code of a previous year project, but have not recompile it. Great error. However, all this is avoided when each team has to design their own project. Even in such things that are the equal for every project, there are always small subtleties that each team will have redress.

As for the assessment, the value of the computer I/O subsystem topic is still 45% of the total. As it is natural in a PBL environment, the project itself has its load. 20% of the grade is taken from the work devoted to the project. However there is a minimum theoretical knowledge that the students are required to get. This theoretical knowledge takes the 10% of the grade and is assessed with a close book multiple choice test. Finally the 15% remaining is assessed via a closed book exam with practical exercises very close to the work developed in the project.

This study is nowadays three years old and we think that the work done warrants some reflection in order to get a better setting of the PBL methodology. The context changes from year to year, and even when the context is the same or very similar the setting of the methodology can be changed, trying to get better results or to increase students' motivation. It has to be beard in mind that in this scenario teachers are learners too [Boss and Krauss, 2007].

The grades obtained each year, the students' satisfaction questionnaires and the lecturer's observations have been considered in order to make changes in the PBL settings. The following subsections will tell about each year's setting. Please while reading bear in mind that all the data mentioned in these subsections is reported in Chapter 6.

5.2.1 The first school year 2010-2011

The first year, the amount of students enrolled in the "Computer Structure" course was of 60. However, only 41 students got to enrol for the project. The rest of the students, either had already drop the studies or the subject, or had chosen the final exam option.

The 41 students enrolled in the project where divided in teams of 5 mates, and one team of 6 mates. Most of the teams managed very well, but there were two teams where some of the team-mates were not regularly coming to class. In a few weeks, the 11 students of these two teams had either stopped coming to class or were reassigned to fill vacancies that students had left available in other teams. 29 students got to the end of the project in 5 teams of 5 mates and 1 team of 4 mates.

Once the teams were created, the students were handed out the project definition. The teams were asked to design an emulator of vending machine which could work on a Nintendo DS machine. At this moment, the reader must be wondering why, using an NDS machine, the project was not the develop a game. The fact is that using a NDS machine to develop the project was a challenge for the lecturers because of the lack of handbooks. For example, the management of the graphic memory is quite complicate and therefore, during the first year students were only allowed to use static images, which was a big handicap to design a game.

The students were instructed on building state machines so that they could use this tool to graphically visualize the design. Meanwhile, during the lectures, a question was proposed: *"What do we need in order to make/solve the project?"*. A lot of needs were made evident from students, several were related to the I/O topic itself, such as how to control the buttons or the screen, the communications between computer components, or how to control timing, whereas others were not, such as a programming environment. At that point they were willing to read the course notes in order to be able to answer the questions. However, since we were following the jigsaw technique, they only got part of the notes, because in the jigsaw technique, each student studies part of the topic at hand, becoming an “expert” on that part. Then there is an “expert” meeting to get to an agreement on the studied part, and finally the team meets again, and each “expert” explains his or her part to the rest of the team [Aronson, Blaney, Stephin, Sikes, and Snapp, 1978]. While they were developing the project and they started needing more information on the others’ parts, they got the whole set of class notes.

The five parts in which the course notes were divided in order to hand them out to each member of the team where as follows:

- I/O controllers and I/O registers (See Section A.1 in Appendix A, in Basque)
- Polling and touch screen (See Sections A.2 and A.2.1 in Appendix A and Section B.1.3 in Appendix B, in Basque)
- Polling and keyboard (See Section A.2 and A.2.1 in Appendix A and Section B.1.4 in Appendix B, in Basque)
- Interrupt and keyboard (See Section A.2 and A.2.2 in Appendix A and Section B.1.4 in Appendix B, in Basque)
- Interrupt and timer (See Section A.2 and A.2.2 in Appendix A and Section B.1.5 in Appendix B, in Basque)

The rest of the work, that is programming the Interrupt Controller (See Section A.3 in Appendix A and Section B.2 in Appendix B, in Basque), and putting it all together would be a team work. On the other hand, some of the reading was not directly connected with the work to do in the project and was to be done by everybody (see Sections A.4, B.1.1, and B.1.2.

The 4 member team could choose between treating the keyboard by polling or by interrupt.

The six teams that got to finish the project developed vending machine emulators controlling touch screen, keyboard and timer, as explained in [Larraza-Mendiluze et al., 2013]. In these vending machines it was possible to buy the following items:

- sandwiches,
- drinks,
- candy,
- pictures,
- train tickets,
- petrol.

With regard to the achieved results, only 6 out of the 29 students that got to finish the project did not get the minimum score in the exam in order to pass the topic. That makes a pass rate of 79%, which is a much better result than the previous 44%.

However, considering satisfaction, items #7, #16, #20, and #22 of the gathered satisfaction questionnaires show that the students felt helpless. They are used to be said what to do every moment and having to work it out on their own was very hard. They complained about not knowing anything about the theory before starting with the project, although they were working on every part of the theory while trying to address the project. They neither liked depending too much on what their team-mates had to say about their part of the theory. With a little help most of them got the idea of what the computer I/O subsystem, polling and interrupt-driven I/O are. However, the idea of how all of it works together was not so clear.

As for the lecturer, it was hard to deal with the team size. There were not too many teams and this was helpful, but with a bigger group in which there will be more teams, the management of the teams could become a problem.

Therefore, a new proposal was made in order to try to overcome the problems detected. On the one hand, the teams will be formed by at least one member less. This was a tricky decision because the group was going to be bigger. However, even for a bigger group, the lecturer thought that it would be easier to manage within team conflicts. On the other hand, the timing to introduce the theory could also change, as will be explained in next subsection, in order to cover students' demands.

5.2.2 The second school year 2011-2012

The second year, the amount of students enrolled in the "Computer Structure" course was of 80, out of which 57 students got to enrol for the project. Again, the rest of the students, either had already drop the studies or the subject, or had chosen the final exam option.

The 57 students enrolled in the project where mostly divided in teams of 4 members, but there also were a few teams of 5 members, and one team of 3. It has to be said that the students of the 3 members team accepted on their own responsibility the fact that they would have to work harder in order to finish the project. Two teams dropped. The first one arguing that they wanted to devote more time to the subjects they had drop in the first semester. The second team just stopped coming. In the rest of the teams there were some movements because some members also dropped. Finally there were 4 teams with 3 members, 5 teams with 4 members, and 3 teams with 5 members, that is to say 47 students in 12 teams altogether.

This time, the project definition was not handed out to the students until after working on the theory. The method to work on the theory was again the jigsaw puzzle, but when they finished with the last step of the jigsaw puzzle they all got the whole set of class notes.

During the previous year the lecturers were also improving their control over the machine and its resources, and at the end we were able to give the students some routines in order to move sprites. This opened the door for designing games, which we thought would be more attractive for the students. Therefore, we changed the definition of the project. During 2011/2012 academic year students had to implement a simple game using the same NDS devices of the previous year.

Since the theory was already introduced, the question to inspire willingness could not be the same as in the previous year. This time the students were asked the following: *"How can this be done in the NDS"*. They got say things like *"we will need the peripherals registers' address"*, or *"how are interrupts managed in the NDS?"*. But this time they were eager to start developing THEIR game.

As we had hypothesized students were very attracted by the idea of developing a game for the NDS console. They were really engaged to the project.

The twelve teams that got to finish the project developed the twelve games describe below:

- Anaconda, a snake game in which the snake did not grow;
- Aracno jump, a platform game in which a spider had to jump from platform to platform;
- Arkanoid, an arcanoid game in which the ball only moved vertically an only once in the bar could be moved horizontally;
- Bold Eye, a quick vision game;
- Mineswipper, just it;
- Chombs, a labyrinth game with enemies to avoid;
- Gravity, a platform game where the gravity changes;
- Moveblock, a labyrinth game;
- Nyan space, an obstacle avoiding game;
- Parkour King, another obstacle avoiding game;
- Super Nacho, a platform game where the avatar is a nacho;
- Total quiz, a quiz game.

It was very nice to have the students so involved in their project, but this time there was also one problem (at least). Teams were not very open minded as to their project definitions. Some of them were classified as hard to develop by the lecturer, but the students would not change their mind. Only the team that finally developed the Bold Eye game switched from a very ambitious graphical adventure game. Not only had some of the definitions a strong programming requirement, but the students would not go for a graphically poor environment. They would dedicate too much effort on these issues and much less to I/O issues. Most of the issues related to what they have to learn in this topic were achieved via trial and error. Of course this was reflected on the scores they got. This time, only 54% of the students that got to finish the project did also got the minimum grade at the exam. The value is still above the 44% obtained with the previous methodology, but much below the 83% of the precedent school year.

The satisfaction questionnaires also reflected a bigger dissatisfaction, although not statistically significant. However, items #7, #15, #20, and #22 did not get better grades. Working on the theory before telling the students what they were supposed to do in their project did not make them feel more secure on the theoretical concepts, neither they felt supported during the development of the project, and they still do not think that the project itself is enough practice to understand the content of the subject.

From the lecturer's point of view, that year was very demanding, the deadline was close and most of the teams did not have the project ready because they had spent too much time developing graphics. Also during the theoretical part, when dealing with the jigsaw puzzle, the lecturer felt too difficult to guide the discussion groups as the teams were too big or too many. Moreover, the teams that finally got to have only three members did not have much difficulties to end their projects up. Therefore, the conclusion is that the project can be assigned to teams of three members, where the management of the team is easier. The decisions taken in order to overcome these problems were as follows:

- The biggest size of the teams should be of 4, however teams of 3 are preferred.

- Problems derived from the jigsaw puzzle technique could be overcome using an inquiry-based learning technique [Kahn and O'Rourke, 2005], where questions are proposed and answered in different ways.

- The too ambitious game designs should be avoided since the beginning in the project definition.

5.2.3 The third school year 2012-2013

That year, the amount of students enrolled in the "Computer Structure" course was of 78, out of which 40 students got to enrol for the project. As allways, the rest of the students, either had already drop the studies or the subject, or had chosen the final exam option.

The 40 students enrolled in the project where divided in teams of 3 members. However there were 4 students that would rather work in pairs than form a 4 member team and therefore there were 12 teams of 3 members and 2 teams of 2 members. In this occasion only one students dropped because he decided to go for a design degree. Whether the reason for the teams to remain firm was that it is more difficult for the students to leave their partners when the team is smaller, or that the students that year were more serious, can not be extracted from the data at hand.

In order to develop the project the tasks were divided as follows:

- Interrupt controller and I/O controllers (See Sections A.1 and A.3 in Appendix A and Section B.2 in Appendix B, in Basque)
- Polling (See Section A.2.1 in Appendix A and Sections B.1.3 and B.1.4 in Appendix B, in Basque)
- Interrupt-driven I/O (See Section A.2.2 in Appendix A and Sections B.1.4 and B.1.5 in Appendix B, in Basque)

As in the previous years, the reading that was not directly connected with the work to do in the project was to be done by everybody (see Sections A.4, B.1.1, and B.1.2.

In order to follow the inquiry-based learning technique, lecturers proposed questions that students had to answer based on their own knowledge. This opened the way to a discussion where the theoretical concepts of the topic were outlined. Finally the students had to answer to the questions on their own with the course-notes.

Another tool was introduced that year, the concept maps. Although the objective was to obtain an image of what the students get to know after studying the topic, the hypothesis was that concept maps would help the students on their process to learn the topic. More on the study carried out with the concept maps can be seen in Chapter 7.

Restrictions were stablish in that year's project definition. The students were proposed to build a meta-project. Each team would have to develop a mini-game to open a door in a 100-doors like game. The definition of the mini-games was carefully revised by the lecturer, so they were not too complicated to develop. Of course, the students' initiative must not be repressed and they were given the possibility for improving their projects once the I/O was controlled as the project required. Most of the games were of the kind of the one described in Section 5.3.3, but with different puzzles to open the door.

Only one student dropped, and only four failed to get the minimum grade in the exam. 90% of the students who followed the methodology passed. This is the best results obtained so far. However, the satisfaction dropped dramatically with statistical significance in 16 items. And why would the students be so unsatisfied with the subject? They explicitly made the lecturer know that they did not like concept mapping. It was a new tool requiring lots of effort to which they did not think they could take advantage of.

However, there were two items that got a better score with a significant difference. The first one said that the methodology was useful to link theory and practice. The second one that the methodology was useful to relate the contents of the course and obtain and integrated view.

From the lecturer's point of view everything worked better than in previous years, although it has to be considered that the grades had a centralization tendency which should be avoided, maybe defining tasks to give the opportunity to raise grades.

5.3 The Projects

The projects proposed to the students in the three consecutive school years have been different. The following subsections will show one of the outcomes of each year showing images of the SMs and the final products.

5.3.1 School year 2010/2011

In the 2010/2011 school year students had to implement vending machine simulators that would run on a NDS machine. No restrictions were made on the type of vending machine students should design, only on the peripheral devices that had to be used at least to implement it on the NDS. In fact, the students designed very different machines, including machines that dispensed train tickets, sandwiches and soft drinks.

Figs. 5.2 and 5.3 show the revised SM designed by one of the teams and screen-shots for each transition for the train tickets vending machine.

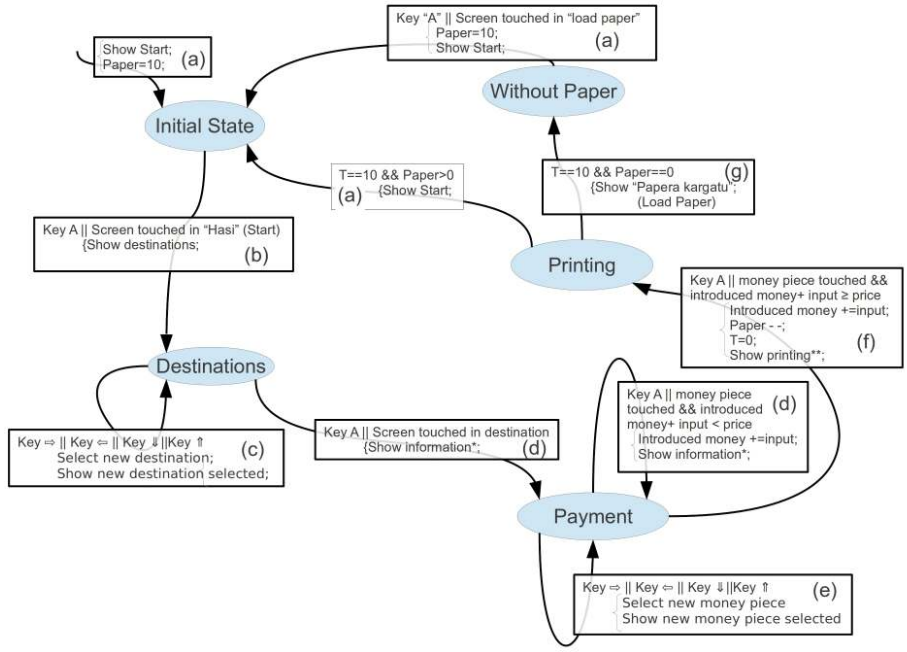
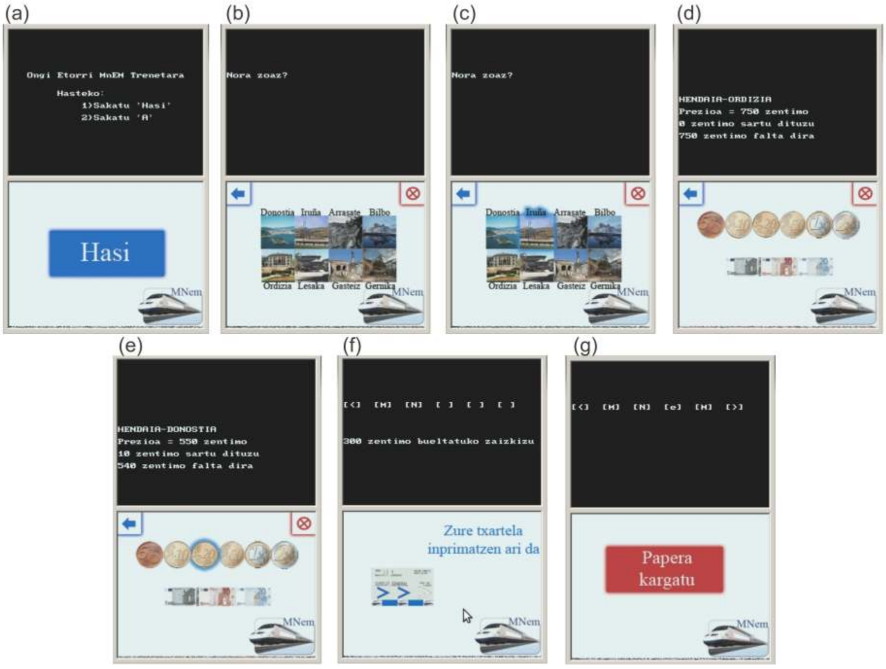

Show Start;
Paper=10; (a)
Key "A" || Screen touched in "load paper"
Paper=10;
Show Start; (a)
Without Paper
Initial State
T==10 && Paper>0
(a) {Show Start;
T==10 && Paper==0 (g)
{Show "Papera kargatu";
(Load Paper)
Key A || Screen touched in "Hasi" (Start)
{Show destinations; (b)
Printing
Key A || money piece touched &&
introduced money+ input ≥ price
Introduced money +=input;
Paper - -;
T=0;
Show printing**; (f)
Destinations
Key A || money piece
touched && introduced (d)
money+ input < price
Introduced money +=input;
Show information*;
Key ⇨ || Key ⇦ || Key ⇩||Key ⇧ (c)
Select new destination;
Show new destination selected;
Key A || Screen touched in destination
{Show information*; (d)
Payment
Key ⇨ || Key ⇦ || Key ⇩||Key ⇧ (e)
Select new money piece
Show new money piece selected

Figure 5.2: State machine for the train ticket vending machine developed for the NDS during the school year 2010-2011. *The information shown is the destination, the ticket price, the introduced quantity, and the quantity remaining to be introduced. Letters in transitions are links for screen-shots in Figure 5.3

In the initial state the system shows a screen inviting to start the process, which could be done either by pressing button "A" or by touch-



```
Ongi Etorri MnEM Trenetara
   Hasteko:
      1)Sakatu 'Hasi'
      2)Sakatu 'A'
```

```
Nora zoaz?
```

```
HENDAIA-ORDIZIA
Prezioa = 750 zentimo
0 zentimo sartu dituzu
750 zentimo falta dira
```

```
HENDAIA-DONOSTIA
Prezioa = 550 zentimo
10 zentimo sartu dituzu
540 zentimo falta dira
```

```
[<]  [M]  [N]  [ ]  [ ]  [ ]

300 zentimo bueltatuko zaizkizu
```

Zure txartela inprimatzen ari da

```
[<]  [M]  [N]  [e]  [M]  [>]
```

Figure 5.3: Screen-shots for the train ticket vending machine developed for the NDS during the school year 2010-2011.

ing "HASI" (START in Basque) in the touch screen (Figure 5.3(a)). Once starting the process all the possible destinations are shown (Figure 5.3(b)). It is possible to select the destination by touching the screen, directly, or moving around with the direction keys and pushing key A when the desired destination is highlighted (Figure 5.3(c)). Once the destination is selected, the payment process will start. The upper screen will show the destination, the price of the ticket, the money introduced so far, and the difference between the price and the money introduced. The bottom screen will be showing the money pieces with which the payment can be done (Figure 5.3(d)). In a real vending machine, at this point we will have to introduce the coins or bills, but in this case we only have to select the money piece by touching the screen, or moving around with the direction keys and pushing key A when the desired money piece is highlighted (Figure 5.3(e)). Once enough money has been introduced to pay the ticket, a screen will be showing the message “printing the ticket” (Figure 5.3(f)) for 10 seconds. If the machine runs out of paper after printing the ticket, it will show a message asking to be paper loaded (Figure 5.3(g)). Once this is done (in this case just touching the screen in the “load paper” message is enough) the system returns to its initial state (Figure 5.3(a)).

5.3.2 School year 2011/2012

During the school year 2011/2012, the students had to develop a game. As we hypothesised, the students where very attracted to that idea. On the other hand, most of the games they thought about instantly were out of their programming skills, and therefore, the lecturers had to make a great effort to maintain the projects at a reachable level for the students.

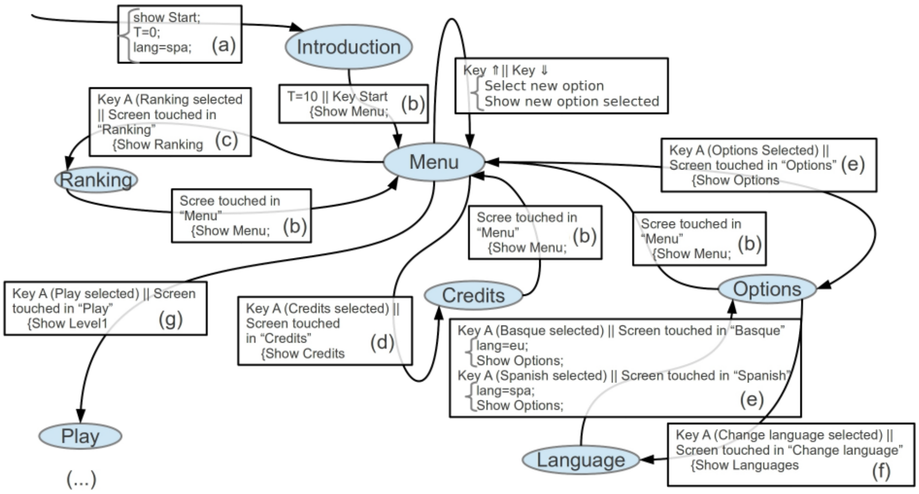


Figure 5.4: State machine for the SuperNacho game developed for the NDS during the school year 2011-2012. Letters in transitions are links for screen-shots in Figure 5.5

Figs. 5.4 and 5.5 respectively show the SM and screen-shots of one of the projects developed during the 2011/12 school year. It is a platform game where a nacho (the popular Mexican food) has to recollect as much cheese as he can while running through different scenarios or levels. The SM starts after a presentation. When the final image of the presentation is shown, two possibilities are offered, either pushing button *Start* or waiting 10 seconds (see Figure 5.5(a)) to show the game menu (see Figure 5.5(b), which needs both NDS screens). Selection of the options in the menu can be done either by touching the screen or with the up and down keys. Option *Ranking*: students can not save information for later games, therefore they just show an image of an imaginary ranking (see Figure 5.5(c)). Option *Credits*: show an image with the names of the team mates (see Figure 5.5(d)). Option *Options*: a new menu is shown with the only options of changing language (see Figure 5.5(e)). By selecting this option the player has the possibility to change the language from Spanish to Basque, going back directly to the options menu (see Figure 5.5(f)). Going back to initial menu after selecting one of these three options is done by touching the little button in the top left corner of the screen *"menu"*(see Figs. 5.5(c,d,e)). Option *Play*: the first level is shown (see Figure 5.5(g), which needs both NDS screens). The upper screen says that button A can be used to jump, that the directional pad can be used to move around, that button B is used to go back to the menu. Finally, it explains that to go to the next level the player needs to catch all the cheese pieces. 15 levels are shown one after the other (see Figure 5.5(h)) once the Nacho has taken all the cheese pieces and gets to the right hand side of the screen. After the 15th level a winning screen is shown (see Figure 5.5(i)). The button *Start* needs to be pressed to go back to the initial menu.

As it can be appreciated, all the control of the *Play* state has been excluded. What has been shown should be enough to see that the dimension of the project exceeds the scope of the project and it is not completely devoted to the I/O subsystem.

5.3.3 School year 2012/2013

Figs. 5.6 and 5.7 respectively show the SM and screen-shots of one of the projects developed during the 2012/13 school year adapted for explanation in this dissertation. It is a mini-game where the player needs to solve a puzzle to get the elevator door open (see Figure 5.7(a)). The player at the beginning does not have any information and therefore

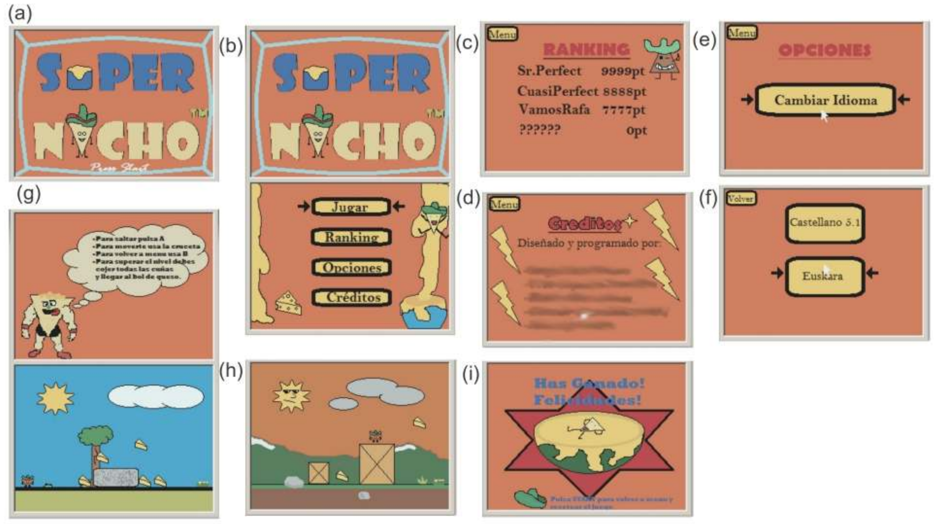


Figure 5.5: Screen-shots for the train ticket vending machine developed for the NDS during the school year 2011-2012.

probably will start touching the screen. When the door is touched, a warning is showed saying that there is no electricity, and a red light appears (see Figure 5.7(b)). Then the fuse box need to be touched to be able to see the fuses (see Figure 5.7(c)). The player has 20 seconds to set the fuses correctly. If the wrong fuse is touched, it will be necessary to press key B to start again (see Figure 5.7(d)). The same will happen if the time gets to the limit (see Figure 5.7(e)). But, if all the fuses are correctly set, then the player will have to press key A to go back to the elevator room (see Figure 5.7(f)). This time the electricity will be on, and the light in the elevator green (see Figure 5.7(g)). Touching the door will then open it (see Figure 5.7(h)), and get to the end of the minigame. In Appendix C, the source code of this project is shown.

5.4 Conclusions

After these three trials in which thirty-four projects were developed, the conclusion is that as we hypothesised, the students are very attracted

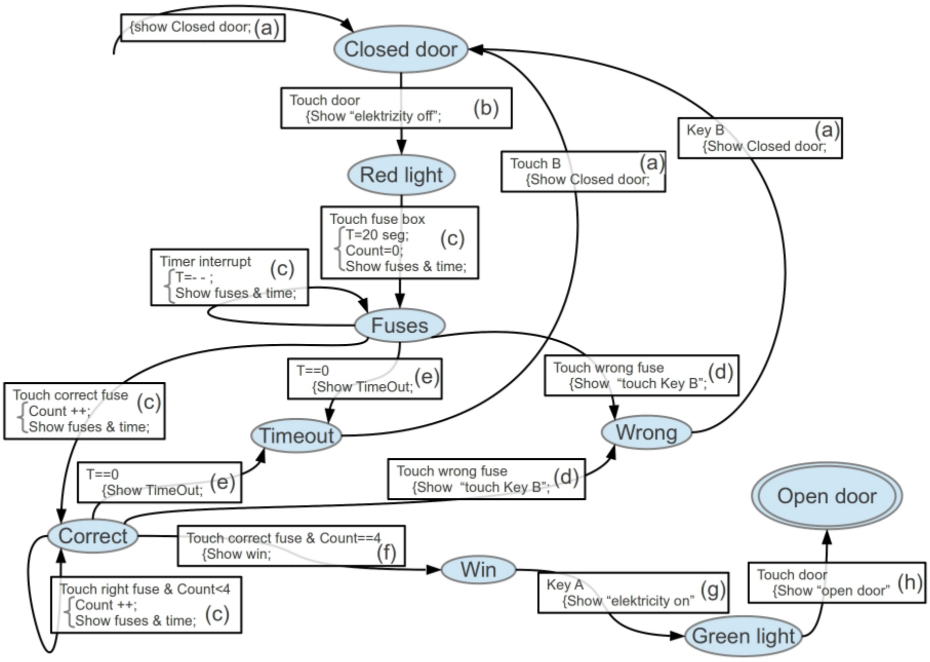
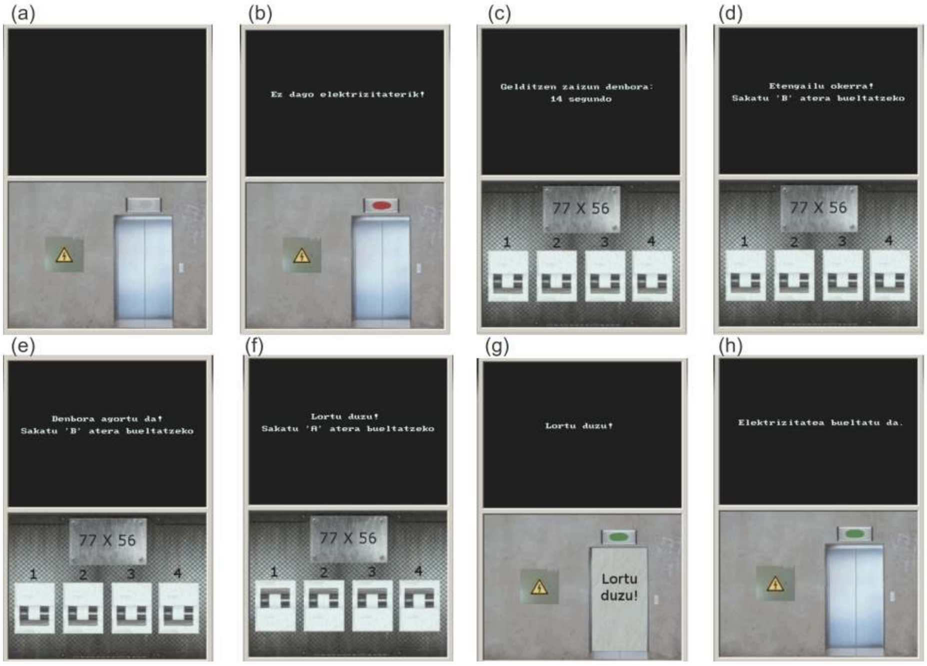

{show Closed door; (a)
Closed door
Touch door {Show "elektrizity off"; (b)
Red light
Touch B {Show Closed door; (a)
Key B {Show Closed door; (a)
Touch fuse box {T=20 seg; Count=0; Show fuses & time; (c)
Timer interrupt {T=- - ; Show fuses & time; (c)
Fuses
T==0 {Show TimeOut; (e)
Touch wrong fuse {Show "touch Key B"; (d)
Touch correct fuse {Count ++; Show fuses & time; (c)
Timeout
Wrong
T==0 {Show TimeOut; (e)
Touch wrong fuse {Show "touch Key B"; (d)
Open door
Correct
Touch correct fuse & Count==4 {Show win; (f)
Win
Key A {Show "elektricity on" (g)
Touch door {Show "open door" (h)
Touch right fuse & Count<4 {Count ++; Show fuses & time; (c)
Green light

Figure 5.6: State machine for the "*Elevator door*" minigame developed for the NDS during the school year 2012-2013. Letters in transitions are links for screen-shots in Figure 5.7

towards programming a game for a game console. However, it is very hard to control the programming difficulties they can have being in their first study year. As an example, during the course 2011-2012, a team decided to develop an "arkanoid" like game. The lecturer advised the team to be careful with this idea since the angles and velocity of the ball should be considered. Indeed, the students had trouble with this, they spent to much time on it and finally their "arkanoid" ball moved only vertically, and the horizontal move was done statically in the bar. This led the students to discourage and to miss a lot of detail in issues really related to the computer I/O subsystem. Therefore, we extracted from here a recommendation to carefully delimit the scope of each project.



Ez dago elektrizitaterik!

Gelditzen zaizun denbora:
14 segundo

Etengailu okerra!
Sakatu 'B' atera bueltatzeko

Denbora agortu da!
Sakatu 'B' atera bueltatzeko

Lortu duzu!
Sakatu 'A' atera bueltatzeko

Lortu duzu!

Elektrizitatea bueltatu da.

Figure 5.7: Adapted state machine the "*Elevator door*" minigame developed for the NDS during the school year 2012-2013.

Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
YES, I GOT IT TO WORK!
I HAVE A POSITIVE RESULT!
SAMPLE SIZE: N = 1!
JORGE CHAM © 2012
GRAD SCHOOL: GETTING EXCITED ABOUT INSIGNIFICANT SIGNIFICANCE.
a few hundred more to go!
WWW.PHDCOMICS.COM

title: "Significance" - originally published 11/28/2012

CHAPTER 6

Analysis of the data obtained during the three years

THE experiment was conducted, each course, only in one of the two groups in which we teach the subject “Computer Structure”. This subject is taught in the first year of the computing engineering degree. It is worth remembering that it is a mandatory course, so every student in the first year must enrol in this course, but not all of them attend it or follow it actively. Only the students who have been following it actively have been considered in this study.

While developing the study, a few steps were followed. Once PBL had been applied and the project had been developed using NDS machines, some data were gathered for quantitative and qualitative analysis. On the one hand, the grades obtained by the students in the computer I/O subsystem topic where considered for a quantitative analysis. On the other hand, satisfaction was analysed both quantitative and qualitatively. This data has been partially published in [Larraza-Mendiluze et al., 2013].

6.1 Analysis of the grades obtained by the students in the computer I/O subsystem topic

The mark-grade system adopted in Spanish universities recognizes four common grades. The students’ performance is assessed using a 10-point grading scale and the grades are expressed as follows: outstanding (mark ≥ 9), remarkable ($7 \leq$ mark < 9), pass ($5 \leq$ mark < 7), and fail (mark < 5). Some of the students that achieve 10 points are eligible for a special grade denoted outstanding with honours. On the other side, some of the students enrolled do not take the exam (drop-out).

Figure 6.1 shows the grades obtained by the students in the I/O subsystem topic along the three years where the NDS and the PBL has been applied. Fail and drop-out rate is the inverse of overall pass rate. Compared to the 44% pass rate of the previous years, it is obvious that the new setting, at its last configuration is very beneficial. However, there is still space for improvement, since most of the grades of the last year (59%) are just pass grades and it would be convenient to displace the grades towards remarkable.

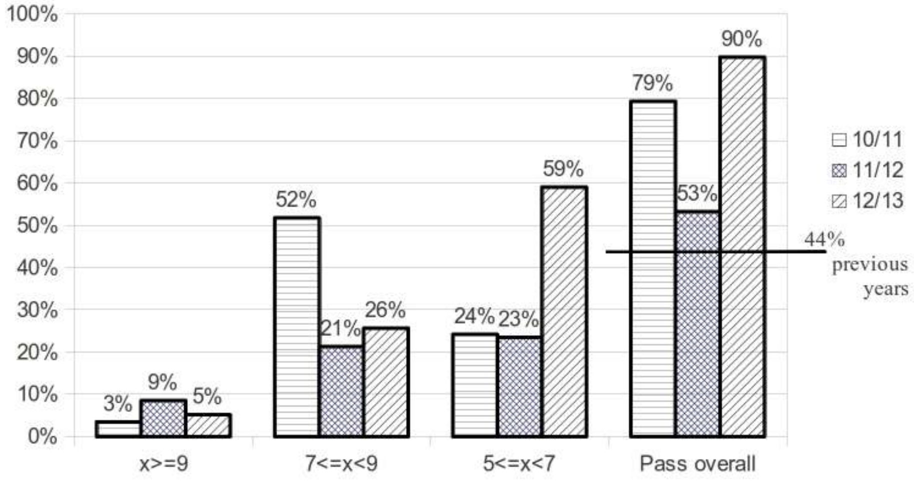


Figure 6.1: Grades obtained by the students in the I/O subsystem topic

There is also a group in which all these changes have not been applied in the same way. At the beginning the possibility of using that group as an experimental group was considered. However along the three years, in that group to many parameters (machine used as infrastructure, methodology, lecturer, etc) have been changing and therefore the comparison has become too difficult. However it is also interesting to see the pass rates obtained in that group, which can be seen in Figure 6.2.

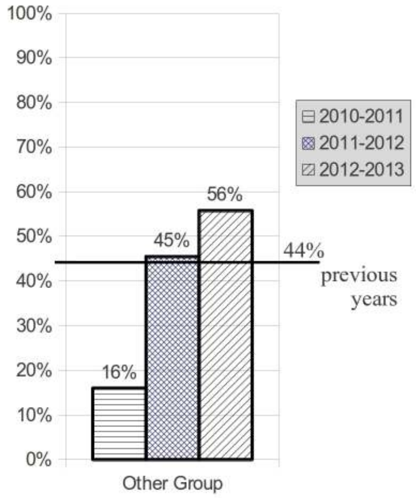


Figure 6.2: Pass rates obtained in the I/O topic in the parallel group

During school year 2010/2011, the parallel group was using the same infrastructure and methodology than in previous years. The pass rate that year was of 16%. This rate is much bellow the pass rate of the previous years. This confirms our fear of passing the I/O subsystem topic from a third semester course to a second semester course. The students in their first year of studies need a different consideration.

The second year, 2011/2012, this group started using the NDS and some active learning techniques. The pass rate then rose to 45%. It has to be borne in mind that here all the students who tried the exam were considered, an not only the ones who followed the active learning methodologies.

The last year, 2012/2013, the lecturer changed and this fact could have influenced the rise of the pass rate, but also using still more active learning techniques could have influenced.

After reading this data, it is possible to conclude that using both the NDS and the active learning methodologies is beneficial for the learning process of the I/O subsystem.

6.2 Analysis of the satisfaction among the students while learning the computer I/O subsystem

6.2.1 The satisfaction questionnaire

The objective of the questionnaire was to determine the level of student satisfaction with the educational methodology. The questionnaire contained 32 items, which were Likert-type items on a scale from 1 to 5.

The number of students that responded to the questionnaire was of 29, 42, and 48 for the school years 10/11, 11/12, and 12/13 respectively. Figures 6.3, 6.4, 6.5, and 6.6 show the average results obtained each year for each item.

- Items in purple background are to be considered because they show a very low grade;
- Items in yellow background are to be considered because they show a statistically significant difference from the first year to the last one having the average result decreased;
- Items in green background also show a statistically significant difference but improving the results from the first year to the last one.

The last year the questionnaire changed a little bit. Question 8. which asked about the use of technological and multimedia resources was converted into 5 new questions asking for each of the resources, concept maps, project, notes, moodle diary, and moodle forums. The data obtained in those new questions can be found in table 6.1.

These values show that the concept mapping has not been well received by the students, and it can be the reason of the overall values decrease of the questionnaire.

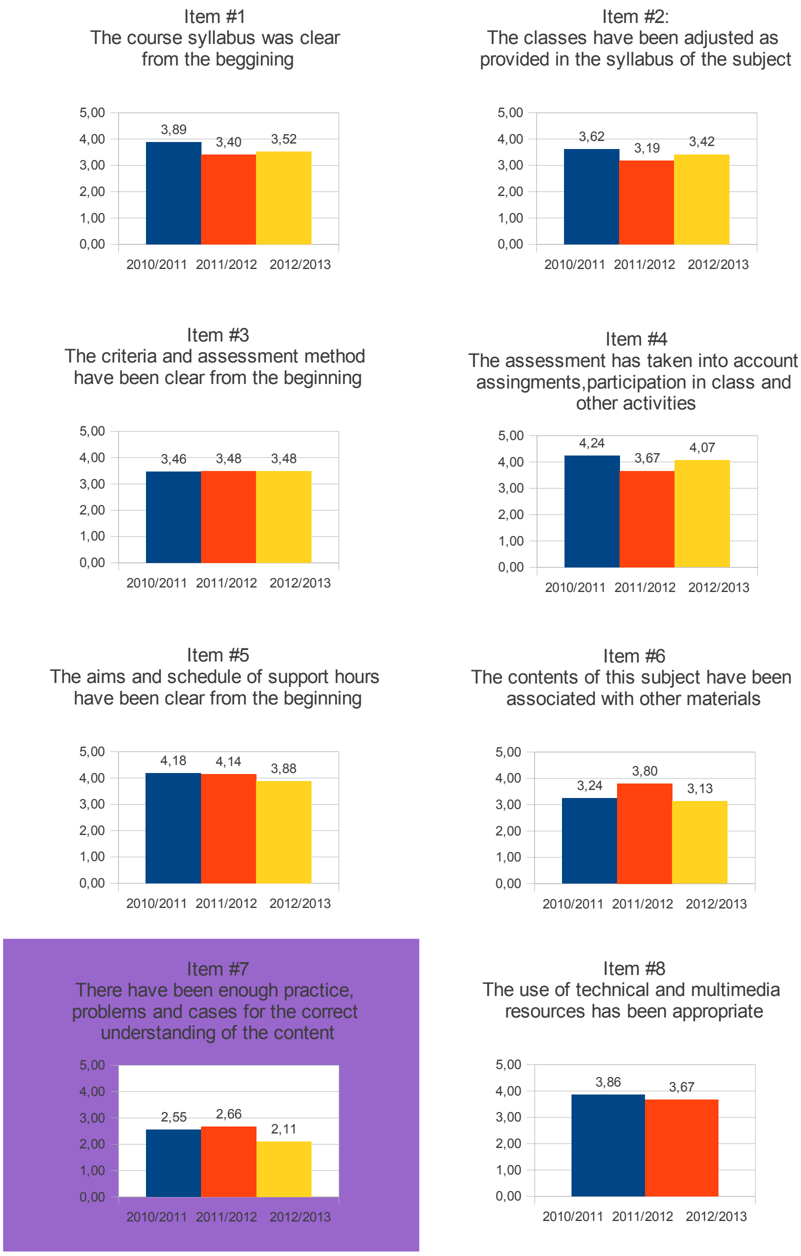


Figure 6.3: Satisfaction questionnaire, charts for items 1 – 6

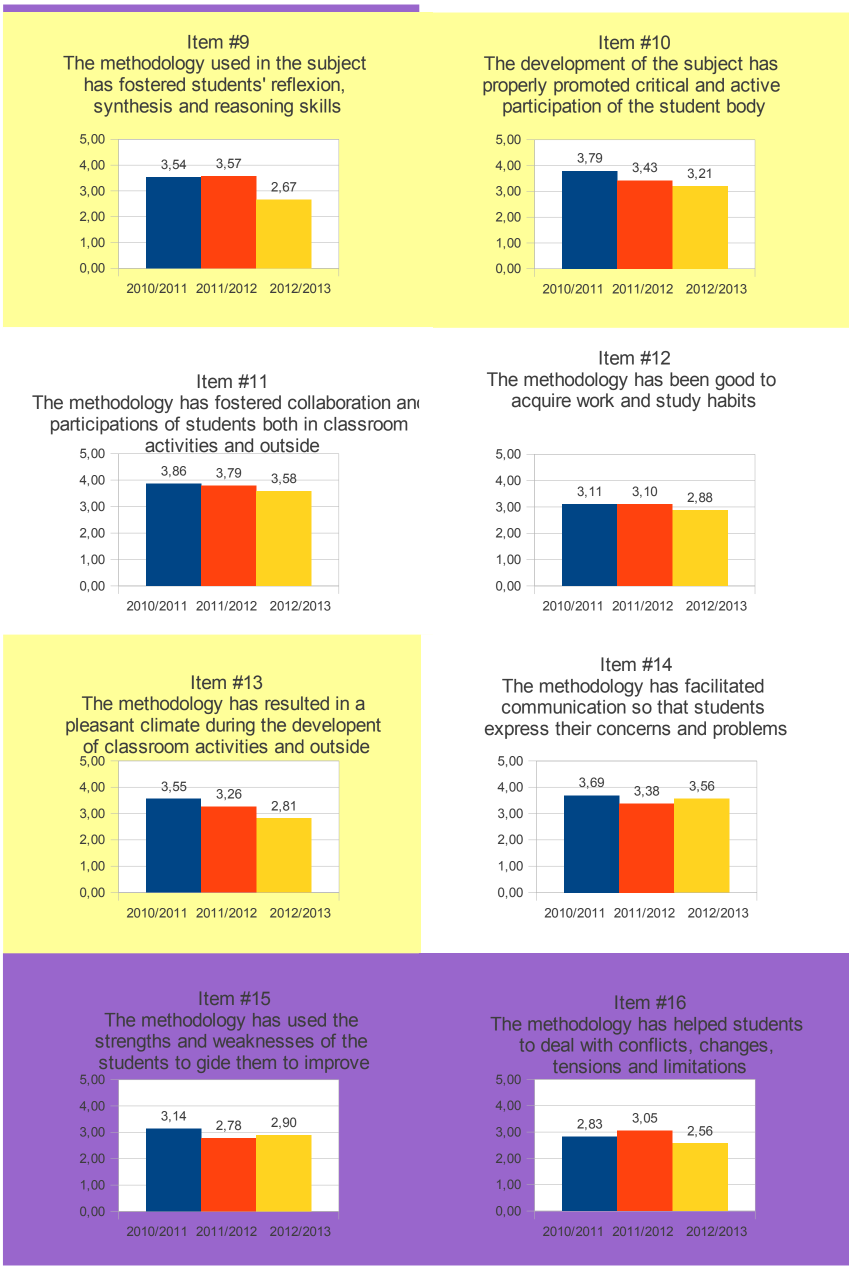
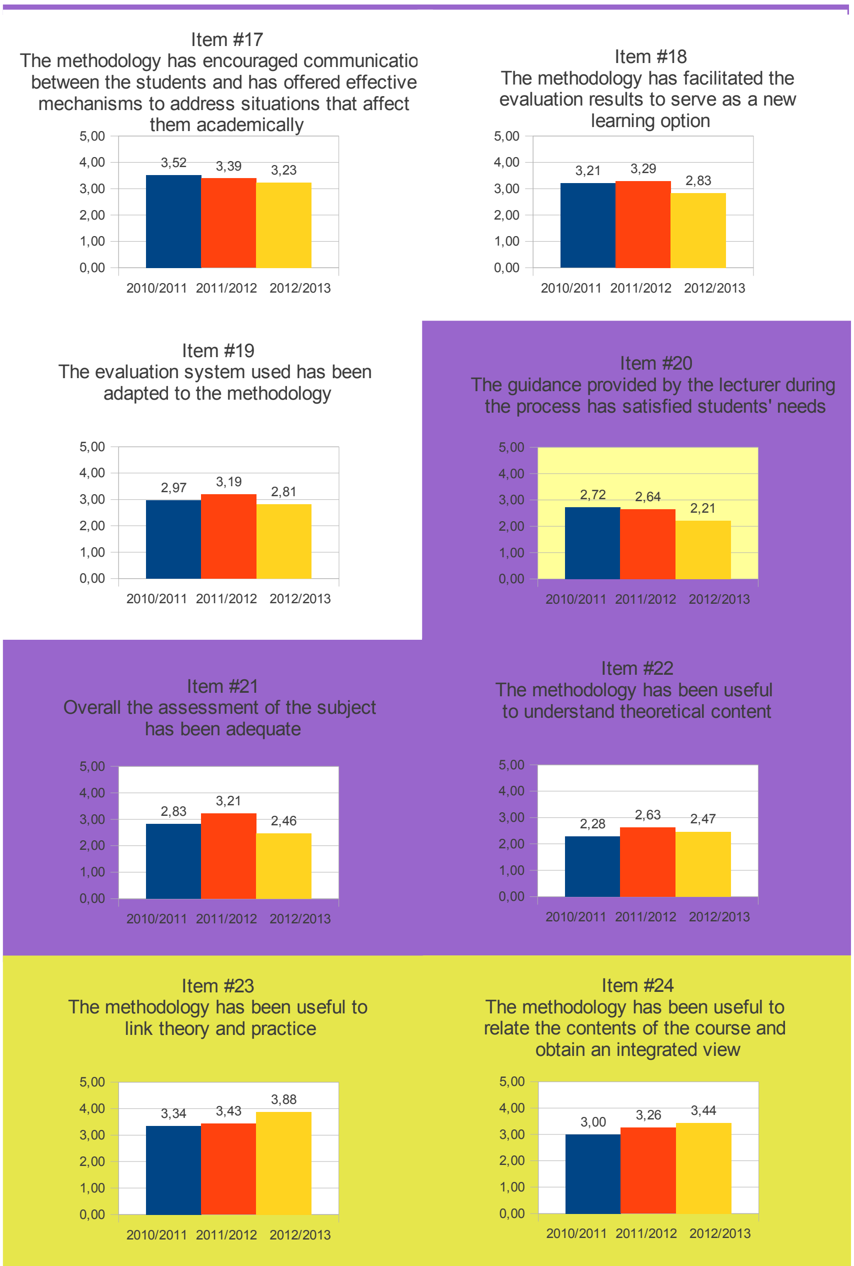


Figure 6.4: Satisfaction questionnaire, charts for items 7 – 12



Item #17
The methodology has encouraged communicatio between the students and has offered effective mechanisms to address situations that affect them academically

Item #18
The methodology has facilitated the evaluation results to serve as a new learning option

Item #19
The evaluation system used has been adapted to the methodology

Item #20
The guidance provided by the lecturer during the process has satisfied students' needs

Item #21
Overall the assessment of the subject has been adequate

Item #22
The methodology has been useful to understand theoretical content

Item #23
The methodology has been useful to link theory and practice

Item #24
The methodology has been useful to relate the contents of the course and obtain an integrated view

Figure 6.5: Satisfaction questionnaire, charts for items 13 – 18

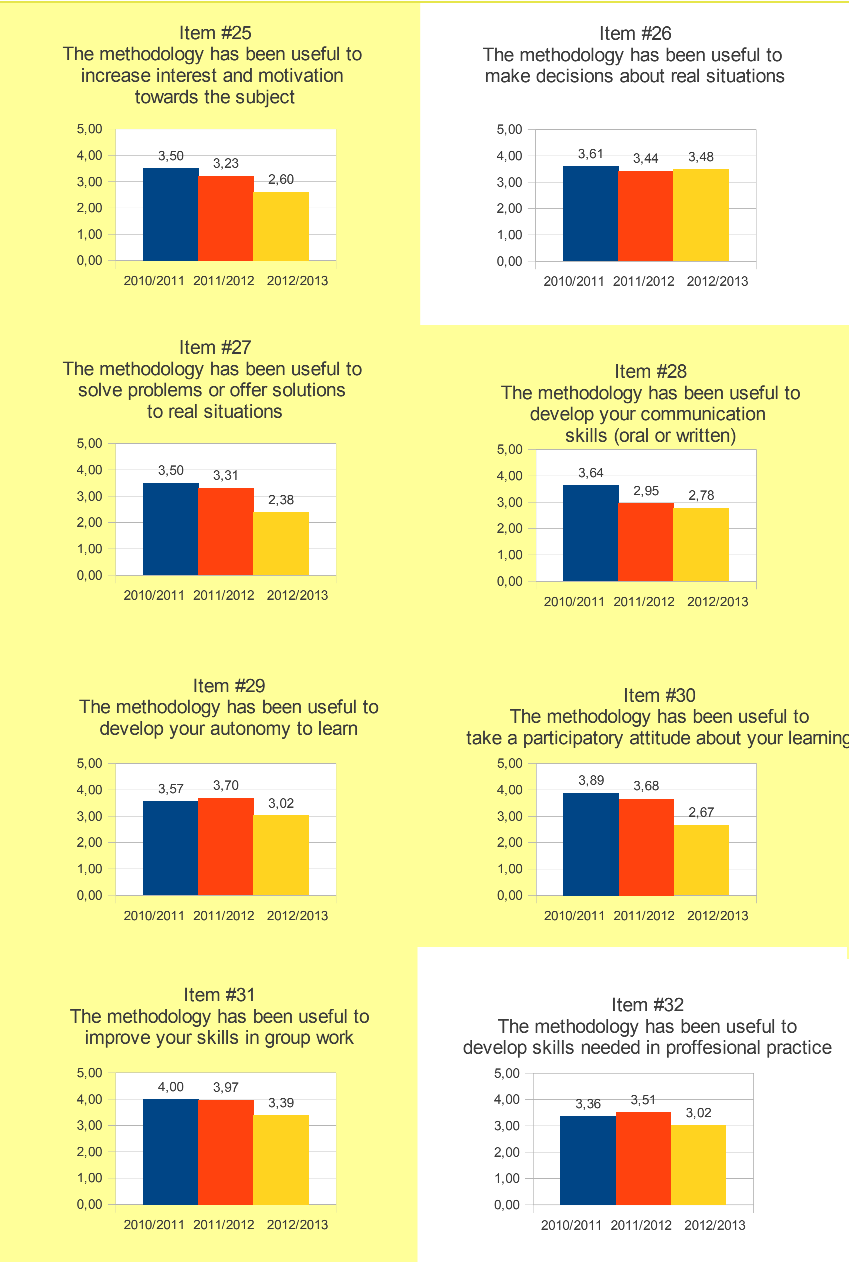


Figure 6.6: Satisfaction questionnaire, charts for items 19 – 24

	2012/2013		
	Avg.	STD	N
1. Concept maps	2.47	1.18	47
2. Project	3.88	0.91	48
3. Class notes	3.44	0.92	48
4. Moodle diary	2.60	0.94	48
5. Moodle forums	3.48	1.13	48

Table 6.1: Items of the new questions in the questionnaire with their average responses (Likert scale [1–5])

6.2.2 Data for a qualitative analysis of the satisfaction

The satisfaction questionnaire had also the following two open questions which answers could be considered for a qualitative analysis:

1. Would you change anything? Can you think of any suggestions for improvement?
2. Write any other opinion or aspect that you consider relevant that is not collected on these indicators.

Besides these two questions, a discussion group was used the first year as a qualitative tool to gather information about the satisfaction among students with relation to the methodology and the new platform (NDS instead of PC). In a discussion group, a group of people discusses a topic and expresses their points of view, as well as their method of understanding and perspective. A discussion group was formed with 7 members of the group. Each member was part of a different working team to ensure that a more comprehensive point of view would ensue. All of this has been very useful in appreciating the view and attitude of each participant towards the topic.

Two colleagues from the Faculty of Education Science conducted the discussion in which the students were asked to speak about the following topics:

- Clarity of the program
- Achievement of objectives
- Adequacy of the practices and methodology in order to achieve the objectives
- Adequacy of student assessment
- Adequacy of technological and other resources to the educational process
- Quality of course notes
- Assessment of the platform (NDS)
- Individual work versus group work
- Student motivation
- Lecturer's support
- Relationship of this course with others
- Students' participation in the development of the subject (content, practices, evaluation)
- Positive and negative aspects of the methodology in the subject
- Proposals for change and improvements
- Level of involvement in the subject (from the students' point of view)
- Any additional points to be added

The results of the discussion group did not differ much from the results obtained in the open questions, and it required much effort. Therefore, there was not a discussion group during the second and third years of the experiment.

Next, data obtained from the first course's discussion group and the open questions of the satisfaction questionnaires are going to be explained.

The process after the discussions and analysis of the questionnaires allowed us to obtain information about needs and expectations and to explain how the situation is perceived. The overall feeling towards the subject has been positive, both in the questionnaire and in the discussion group. Students were satisfied because the course development adhered closely to the syllabus presented at the outset of the course. "...the initial schedule was followed almost to the letter, without big changes." "It was pretty clear". Regarding the assessment criteria, students thought that the work performed was taken into account (item 4), but the assessment used was not totally adapted to the methodology (item 19); this was something to consider. The projects were developed in groups, and most of the weight of the topic was assigned to the group deliveries and final product, while there was an individual examination. Students were very critical of the fact that their colleagues' work would affect their grades. This is something they are not used to; however, the positive interdependence (i.e., a dependence on one another to reach a goal) is a characteristic of cooperative work, and students need to learn it.

In the first year, the new methodology based on PBL was initially met with reluctance, but the students finally said: "We feel comfortable.", "We learn more." This is corroborated by the results of the questionnaire. Items 22 to 32 show different aspects that students thought this methodology helped them to develop. Among them, only item 22 had a score below 3; in fact, this is the aspect that students criticised most, both in open questions and in the discussion group, with statements such as: "With this methodology, theoretical concepts are not explained as it should be.", "We have missed some more theoretical explanations... sometimes we got lost.", and "We did not have lectures. We got the notes and worked them by ourselves. But did we understand?" Moreover, students thought that this was an obstacle to developing the project: "I would explain more theory because I had many doubts with some of the practices.", and "We haven't seen some examples in class to learn how to apply them.". The lecturers were providing guidance, support and counselling (see item 5), and, in the discussion group, they agreed that the questions raised in the office and via e-mail were answered properly. Furthermore, they agreed that the evaluation process was adequate and allowed students to improve their assignments.

In the third year, also the open questions revealed a difference from the first year. Many students did express their distaste for the concept maps. Just a few felt comfortable with the methodology and realized that "at the end I had a good view of the topic", unlike many of them who thought that more lectures were needed and specially, "more exercises should be revised during the lectures to better face the exams". Are not they asking for a recipe to pass the exam? Others do not like being the one who has to work, and say that "the lecturer should explain more, and not be us who have to read, who have to understand, and who have to do everything".

As for the motivation, there is no doubt that the used platform (i.e., the Nintendo® DS) motivates the students: "It could be done in a PC, but it would not be so attractive. Much better with the Nintendo.", "To learn everything counts.", "This way you are more careful and you pay more attention, it is more satisfying.", "You see that you are learning.", "The motivation is kept during the whole course." Although the last year the satisfaction questionnaire results were not so good, and of course the lecturers have to consider it and keep trying in order to get both good knowledge outcomes and satisfied students, there was a comment that is worth mentioning: "In my opinion the most important thing, not only in CS but in every course, is the motivation, and I would like to deepen on that. In some courses the lecturers take the lead and if you are not interested they do not mind you. The key is: lecturers should generate interest towards the subject and CS is exemplary on that matter, other lecturers could learn a lot from CS, mostly the theoretical ones."

6.2.3 Attraction and retention

Attraction and retention are two more parameters that can be used to analyse students' satisfaction, and raising these numbers is always one of the objectives a lecturer should have.

In this case, interpreting the numbers of the students enrolled in the PBL methodology (attraction) is not trivial because many variables are influencing that number. We are not counting only on the willingness of the students to enrol, but also in the scores they got in the previous parts of the subject. If they do not get a minimum score then they are not considered inside the active methodology.

Considering all the previous statements, we have analysed the retention of the students in the PBL methodology during the topic related to the computer I/O subsystem. Table 6.2 shows the numbers obtained during the three years:

	10-11	11-12	12-13
Students enrolled in PBL	41	57	40
Students that ended the project	29	47	39
Percentage of students that ended the project with respect to those who started it	70.73%	82.46%	97.5%

Table 6.2: Students following and finishing in the PBL methodology

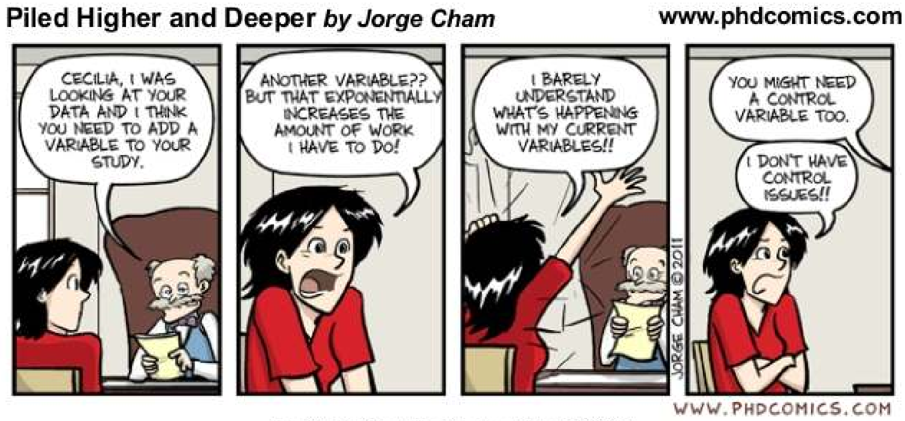
What Table 6.2 shows is a progressive increase of the retention, which means that the changes made in the settings of the project have been beneficial.

6.3 Conclusions

The proposed methodology and platform appear to be suitable for our purposes. This study shows that better results are obtained with this combination of methodology and platform, although there is still space for improvement. Grades should be displaced towards remarkable and more students should be attracted and maintained in the active learning methodology.

On the other hand the students are not totally satisfied with the methodology. They consider that there is not enough practice, problems and cases for the correct understanding of the subject, that the guidance provided by the lecturer during the process does not satisfy students' needs, and that the methodology is not useful to understand theoretical content. They are used to get everything almost done, exercise examples that could be rebuilt in the exam, careful explanations of every detail that could be asked in the exam, and most of them are thinking in passing the exam instead of learning and therefore being able to pass the exam. This methodology asks an extra effort from the students, but maybe, being a subject located in the first course, some of these requirements should be fulfilled.

On behalf the concept maps, the students did not like the effort of having to build them, neither what they felt as being told how to learn. They articulate this feeling both personally and by means of the satisfaction questionnaire (see item 1 in Table 6.1, it has the fourth worst score among all the items). Moreover, we think that this fact affected the overall assessment, and that this is the reason of the overall score decrease of the last year. However, the students were not aware of the increase in the percentage of students passing the computer I/O subsystem topic that year.


Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
CECILIA, I WAS LOOKING AT YOUR DATA AND I THINK YOU NEED TO ADD A VARIABLE TO YOUR STUDY.
ANOTHER VARIABLE?? BUT THAT EXPONENTIALLY INCREASES THE AMOUNT OF WORK I HAVE TO DO!
I BARELY UNDERSTAND WHAT'S HAPPENING WITH MY CURRENT VARIABLES!!
JORGE CHAM ©2011
YOU MIGHT NEED A CONTROL VARIABLE TOO.
I DON'T HAVE CONTROL ISSUES!!
WWW.PHDCOMICS.COM

title: "Variable Control" - originally published 12/2/2011

CHAPTER 7

Students' understanding about the computer I/O subsystem?

The point of this chapter differs from the previous ones. In the TPACK model this chapter is devoted to cover the students' understanding. The aim is, in fact, to build a better TPACK of the lecturer in order to be able to better align it with the content knowledge of the students. We would like to identify exactly where the problem lies when students try to understand the computer I/O subsystem. Therefore, the question we would like to empirically answer in this chapter is: How do students understand the I/O topic?

This chapter is intended to present to the community the work we have carried out in order to answer the above question. Showing first the results of a preliminary study, and then, the quantitative and qualitative analysis of concept maps, the research point to where the preliminary study lead. Both the preliminary study and the quantitative analysis have been published in [Larraza-Mendiluze and Garay-Vitoria, 2012] and [Larraza-Mendiluze and Garay-Vitoria, 2013a] respectively.

The chapter is divided into six sections. Section 7.1 will review the bibliography on this topic. Section 7.2 will report on the preliminary study that lead the final study on concept maps, a work published in [Larraza-Mendiluze and Garay-Vitoria, 2012]. Section 7.3 will detail the investigation itself, the students we selected to participate in the investigation and the steps followed to obtain the information. Section 7.4 will report the data obtained during the study and the quantitative analysis of that information, a work published in [Larraza-Mendiluze and Garay-Vitoria, 2013a]. Section 7.5 will take a different look at the data and report the qualitative analysis. Finally, Section 7.6 presents the conclusions and outlines future work.

7.1 Related work

The literature studying the difficulties in learning programming has grown in recent years as can be seen in conference series such as [ICER; Koli-Calling; ITICSE]. The literature studying the issues in computer architecture education can be found at the Workshop on Computer Architecture Education ([WCAE]), and also in the previously mentioned conference series. The computer I/O subsystem topic has not been very widely treated. There are several references that include the I/O unit within computer architecture, either in simulators such as those presented in [Donaldson et al., 2011a] and [Black and Komala, 2011] or integrated in new teaching approaches [Ramachandran and Leahy Jr, 2007], but nothing was found on the understanding of the computer I/O subsystem.

The different approaches to treating I/O topics found in the textbooks in the literature is also reflected in the different universities. It is possible to analyse teaching guides to validate this (see Section 3 in this dissertation). Other approaches to identify the differences between what is taught in different centres are by analysing exam questions and obtaining the relevant concepts and problems to solve that are proposed by teachers in the corresponding universities. In order to do this, the collaboration of the teaching groups is needed, as has been stated in [Larraza-Mendiluze and Garay-Vitoria, 2013b].

In this step of the investigation, we want to find out how the students view the I/O topic at the end of the semester, using the teaching/learning methodology described by Larraza-Mendiluze et al. [2013]. This will enable us to detect misconceptions that need to be addressed during subsequent teaching/learning processes.

One of the tools used to depict acquired knowledge is concept mapping. Concept maps are defined by Novak and Cañas [2008] as *"graphical tools for organizing and representing knowledge. They include concepts, usually enclosed in circles or boxes of some type, and relationships between concepts indicated by a connecting line linking two concepts. Words on the line, referred to as linking words or linking phrases, specify the relantionship between the two concepts"*. As Sanders, Boustedt, Eckerdal, McCartney, Moström, Thomas, and Zander [2008] point out, concept maps have been used for different purposes, such as:

- helping students to learn;
- measuring changes in students understanding; and,
- what really interests us, obtaining a static picture of what students know.

We have found some works on concept maps applied to the teaching of Object Oriented Programming [Sanders et al., 2008] and [Hubwieser and Mühling, 2011] and a paper on mental models in computer architecture [Yehezkel, Ben-Ari, and Dreyfus, 2005], where the researchers study the way in which the students understand the different types of interactions between the CPU, Memory and Input and Output units in an assembly language course.

Concept map show individual knowledge, but in the second study we wanted to analyse the whole students group. Therefore, software tools for electronic concept mapping from the learner's perspective [Mühling and Hubwieser, 2012] were not suitable for this research.

Software that could be used to analyse all the concept maps together was not easy to find. During the search, most of the solutions we found were valid only for individual concept map assessment. However, McLinden [2013] showed us the similarities between concept maps and social networks. For social network analysis there are many tools used to find similar patterns between users. We tried some of these software tools; UCINET [Freeman, Everett, and Borgatti], and PAJEK [Vlado, 2005].

Since we started working with concepts maps we have used the tool CM-ED which has been described at [Arruarte, Elorriaga, and Rueda, 2001] and can be found at [GaLan]. Due to the similarities with the text documents generated by the aforementioned tool, PAJEK [Vlado, 2005] was selected as the social network analysis software for this project and its use was learned thanks to [de Nooy, Mrvar, and Batagelj, 2005; Batagelj and Mrvar, 2011].

7.2 The preliminary study

Previous to the step of having all the students building concept maps, we carried out a preliminary study that would tell us whether there was a real problem to pursue.

The tool used for this study was not a pure concept map as defined by they creators [Novak and Cañas, 2008], but just concepts linked among them. First of all several Computer Architecture lecturers were asked to build three different, let us call them, “concept maps”. Each of the “concept map” had to be built with different concepts and following the next criteria:

- a classification of concepts,
- the communication between elements of the I/O subsystem,
- steps that must be followed when using different I/O techniques (polling, interrupt-driven I/O, and DMA.

The results obtained from the lecturers with a big consensus after a validation step are shown in Figures 7.1, 7.2, and 7.3. In Figure 7.1, concepts in square boxes are classified into concepts in oval boxes. In the case of I/O registers, there are two possible classifications, to the right the type of register, to the left where the registers are stored.

In Figure 7.2, dashed lines reflect that the concept at the origin of the arrow is part of the concept at the destination of the arrow, whilst continuous lines express a communication between both concepts.

In Figure 7.3, each technique is expressed on the top in a square, while steps are expressed in rounded squares in the same order they use to happen.

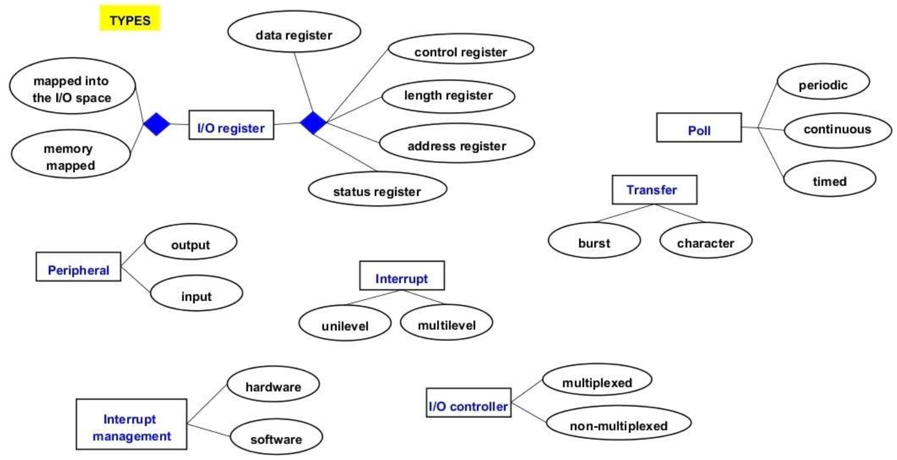


Figure 7.1: Classification of I/O subsystem concepts

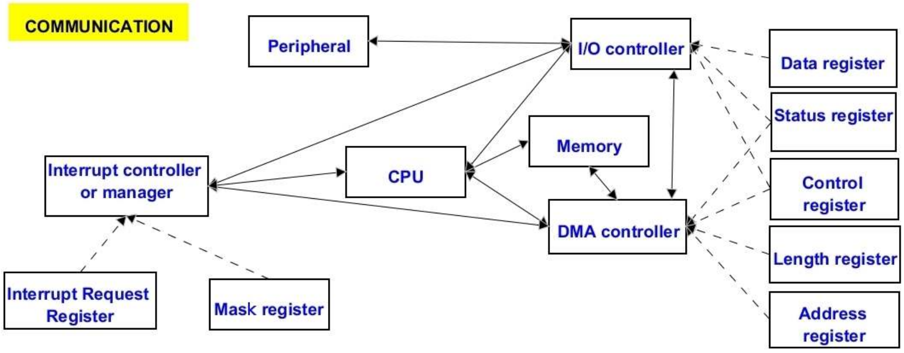


Figure 7.2: Communication between the elements of the I/O subsystem

Twelve students were selected for participating in the study out of the 81 that had follow the subject. There were students from both, the Basque and the Spanish groups, and within all the achieved grades ranges. However, three of the students with lower grades either they did not show to the appointment or their “concept map” was too poor to be considered. Therefore we finally worked with the “concept maps” built by 9 students.

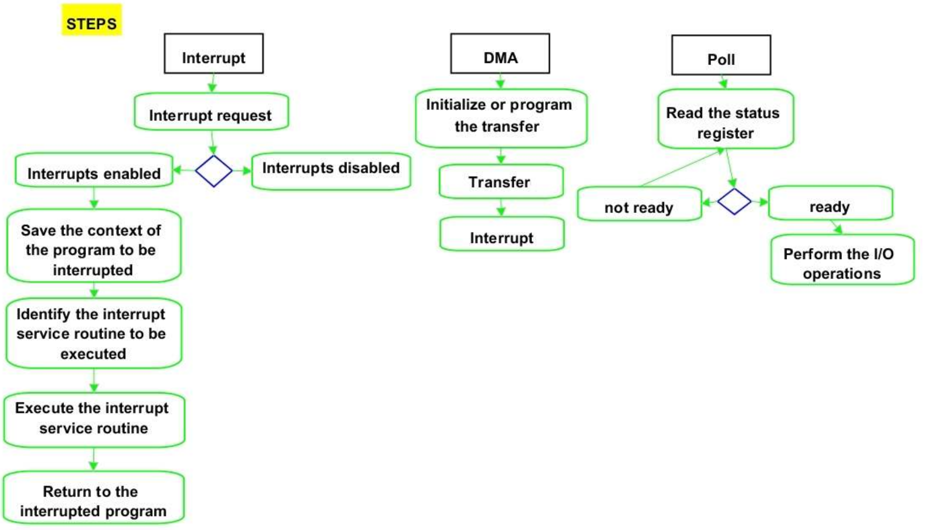

STEPS
Interrupt
Interrupt request
Interrupts enabled
Interrupts disabled
Save the context of the program to be interrupted
Identify the interrupt service routine to be executed
Execute the interrupt service routine
Return to the interrupted program
DMA
Initialize or program the transfer
Transfer
Interrupt
Poll
Read the status register
not ready
ready
Perform the I/O operations

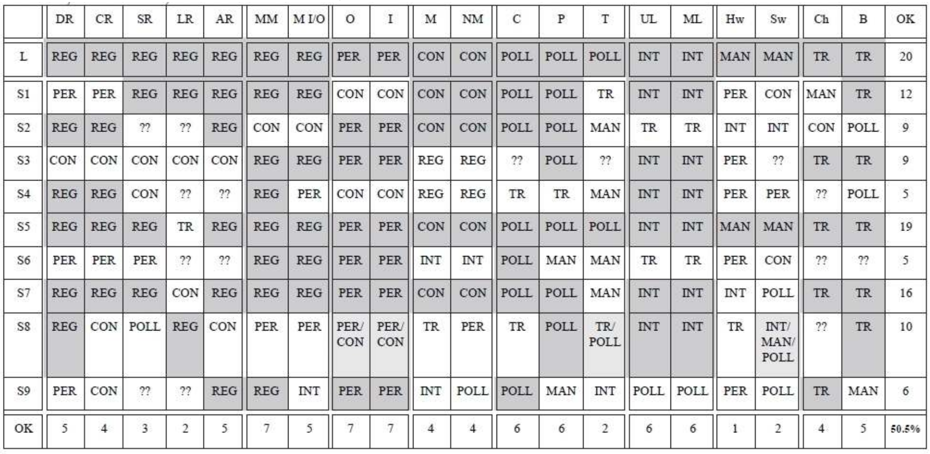
Figure 7.3: Steps that must be followed while using different I/O techniques (polling, interrupt-driven I/O, and DMA)

Students got the concepts shown in Figures 7.1, 7.2, and 7.3, and they only had to stablish the links, without linking words.

The links that the students established are shown in Tables 7.1, 7.2, 7.3, and 7.4. Relationships shown in the "concept map" belonging to Figure 7.1 is shown in Table 7.1. The "concept map" in Figure 7.2 needed to tables in order to be analysed. Table 7.2 shows the relationship of components of the I/O subsystem that communicate among each other, while Table 7.3 shows the hierarchy among the components of the computer I/O subsystem, i.e. which "is part of" which. Finally the relations that appeared in the "concept map" shown in Figure 7.3 can be seen in Table 7.4.

One of the conclusion after this study (others can be found in [Larraza-Mendiluze and Garay-Vitoria, 2012]) was that the students could not see the whole picture of how the computer I/O subsystem works and therefore a further study was planned for the next year. The following sections will explain the method followed in this study, the results obtained, and finally the conclusions derived from the study.

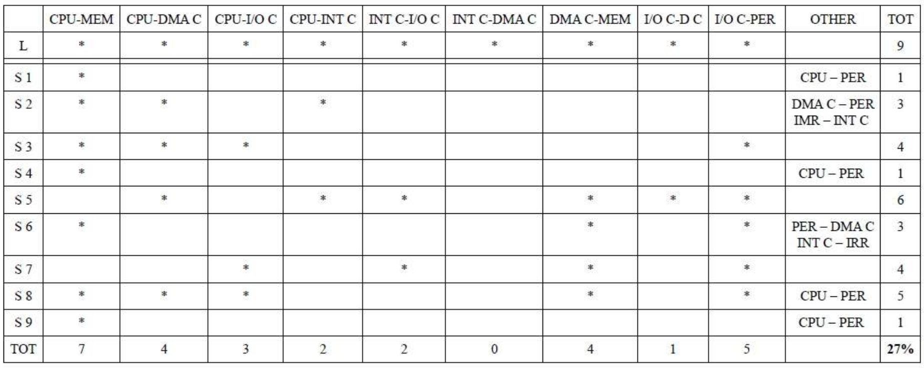
The abbreviations are as follows: DR: Data Register, CR: Control Register, SR: Status Register, LR: Length Register, AR: Adress Register, MM: Memory Mapped, M I/O: Mapped into the I/O space, O: Output, I: Input, M: multiplexed, NM: Non-Multiplexed, C: Continuous, P: Periodic, T: Timed, UL: Unilevel, ML: Multilevel, Hw: by Hardware, Sw: by Software, Ch: by Character, B: by Burst, REG: I/O Register, PER: Peripheral, CON: I/O Controller, POLL: Poll, INT: Interrupt, MAN: Interrupt Management, TR: Transference, L: Lecturers, S_i:Student #i, OK: Total correct answers.



	DR	CR	SR	LR	AR	MM	M I/O	O	I	M	NM	C	P	T	UL	ML	Hw	Sw	Ch	B	OK
L	REG	REG	REG	REG	REG	REG	REG	PER	PER	CON	CON	POLL	POLL	POLL	INT	INT	MAN	MAN	TR	TR	20
S1	PER	PER	REG	REG	REG	REG	REG	CON	CON	CON	CON	POLL	POLL	TR	INT	INT	PER	CON	MAN	TR	12
S2	REG	REG	??	??	REG	CON	CON	PER	PER	CON	CON	POLL	POLL	MAN	TR	TR	INT	INT	CON	POLL	9
S3	CON	CON	CON	CON	CON	REG	REG	PER	PER	REG	REG	??	POLL	??	INT	INT	PER	??	TR	TR	9
S4	REG	REG	CON	??	??	REG	PER	CON	CON	REG	REG	TR	TR	MAN	INT	INT	PER	PER	??	POLL	5
S5	REG	REG	REG	TR	REG	REG	REG	PER	PER	CON	CON	POLL	POLL	POLL	INT	INT	MAN	MAN	TR	TR	19
S6	PER	PER	PER	??	??	REG	REG	PER	PER	INT	INT	POLL	MAN	MAN	TR	TR	PER	CON	??	??	5
S7	REG	REG	REG	CON	REG	REG	REG	PER	PER	CON	CON	POLL	POLL	MAN	INT	INT	INT	POLL	TR	TR	16
S8	REG	CON	POLL	REG	CON	PER	PER	PER/ CON	PER/ CON	TR	PER	TR	POLL	TR/ POLL	INT	INT	TR	INT/ MAN/ POLL	??	TR	10
S9	PER	CON	??	??	REG	REG	INT	PER	PER	INT	POLL	POLL	MAN	INT	POLL	POLL	PER	POLL	TR	MAN	6
OK	5	4	3	2	5	7	5	7	7	4	4	6	6	2	6	6	1	2	4	5	50.5%

Table 7.1: Students' concept classification relations

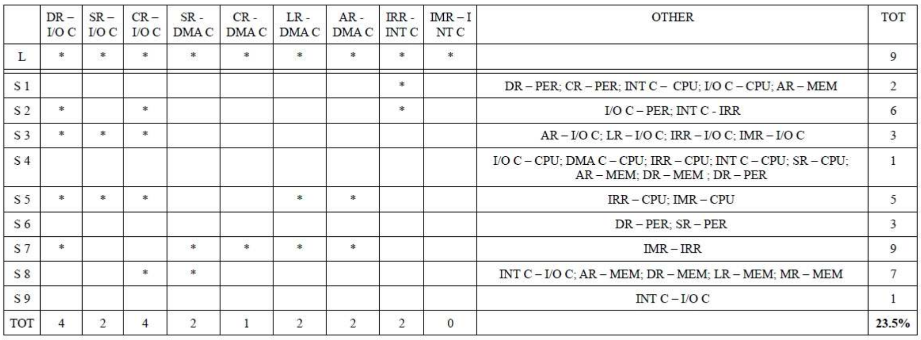
The abbreviations are as follows: CPU: Central Processing Unit, MEM: Memory, D C: DMA Controller, I C: I/O Controller, INT C: Interrupt Controller or Manager, PER: Peripheral, OTHER: wrong connections, L: Lecturers, S_i:Student #i, TOT: Total correct connections.



	CPU-MEM	CPU-DMA C	CPU-I/O C	CPU-INT C	INT C-I/O C	INT C-DMA C	DMA C-MEM	I/O C-D C	I/O C-PER	OTHER	TOT
L	*	*	*	*	*	*	*	*	*		9
S 1	*									CPU – PER	1
S 2	*	*		*						DMA C – PER IMR – INT C	3
S 3	*	*	*						*		4
S 4	*									CPU – PER	1
S 5		*		*	*		*	*	*		6
S 6	*						*		*	PER – DMA C INT C – IRR	3
S 7			*		*		*		*		4
S 8	*	*	*				*		*	CPU – PER	5
S 9	*									CPU – PER	1
TOT	7	4	3	2	2	0	4	1	5		27%

Table 7.2: Communication relations from the point of view of the students

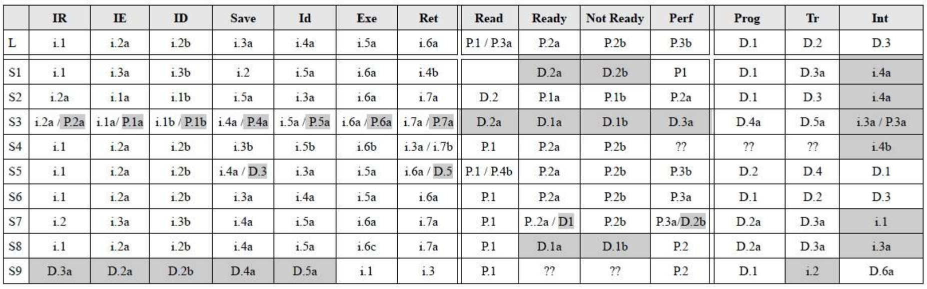
The abbreviations are as follows: DR: Data Register, SR: Status Register, CR: Control Register, LR: Length Register, AR: Address Register, IRR: Interrupt Request Register, IMR: Interrupt Masc Register, I/O C: I/O Controller, D C: DMA Controller INT C: Interrupt Controller or Manager, OTHER: wrong connections, L: Lecturers, S_i:Student #i, TOT: Total correct connections.



	DR – I/O C	SR – I/O C	CR – I/O C	SR - DMA C	CR - DMA C	LR - DMA C	AR - DMA C	IRR - INT C	IMR – INT C	OTHER	TOT
L	*	*	*	*	*	*	*	*	*		9
S 1								*		DR – PER; CR – PER; INT C – CPU; I/O C – CPU; AR – MEM	2
S 2	*		*					*		I/O C – PER; INT C - IRR	6
S 3	*	*	*							AR – I/O C; LR – I/O C; IRR – I/O C; IMR – I/O C	3
S 4										I/O C – CPU; DMA C – CPU; IRR – CPU; INT C – CPU; SR – CPU; AR – MEM; DR – MEM ; DR – PER	1
S 5	*	*	*			*	*			IRR – CPU; IMR – CPU	5
S 6										DR – PER; SR – PER	3
S 7	*			*	*	*	*			IMR – IRR	9
S 8			*	*						INT C – I/O C; AR – MEM; DR – MEM; LR – MEM; MR – MEM	7
S 9										INT C – I/O C	1
TOT	4	2	4	2	1	2	2	2	0		**23.5%**

Table 7.3: *Is_part_of* relations from the point of view of the students

The abbreviations are as follows: IR: Interrupt Request, IE:Interrupt Enabled, ID: Interrupt Disabled, Save: Save the context of the program to be interrupted, Id: Identify the interrupt service routine to be executed, Exe: Execute the ISR, Ret: Return to the interrupted program, Read: Read the status register, Ready: device is ready, Not Ready: device is not ready, Perf: Perform the I/O operations, Prog: Initialize or program the transfer, Tr: Transfer, Int: Interrupt.
Each box shows the position where this step was situated, i stands for interrupt, P for Poll and D for DMA, a and b for the branches in the decision points and the number reflects the order of the step.



	IR	IE	ID	Save	Id	Exe	Ret	Read	Ready	Not Ready	Perf	Prog	Tr	Int
L	i.1	i.2a	i.2b	i.3a	i.4a	i.5a	i.6a	P.1 / P.3a	P.2a	P.2b	P.3b	D.1	D.2	D.3
S1	i.1	i.3a	i.3b	i.2	i.5a	i.6a	i.4b		D.2a	D.2b	P1	D.1	D.3a	i.4a
S2	i.2a	i.1a	i.1b	i.5a	i.3a	i.6a	i.7a	D.2	P.1a	P.1b	P.2a	D.1	D.3	i.4a
S3	i.2a / P.2a	i.1a/ P.1a	i.1b / P.1b	i.4a / P.4a	i.5a / P.5a	i.6a / P.6a	i.7a / P.7a	D.2a	D.1a	D.1b	D.3a	D.4a	D.5a	i.3a / P.3a
S4	i.1	i.2a	i.2b	i.3b	i.5b	i.6b	i.3a / i.7b	P.1	P.2a	P.2b	??	??	??	i.4b
S5	i.1	i.2a	i.2b	i.4a / D.3	i.3a	i.5a	i.6a / D.5	P.1 / P.4b	P.2a	P.2b	P.3b	D.2	D.4	D.1
S6	i.1	i.2a	i.2b	i.3a	i.4a	i.5a	i.6a	P.1	P.2a	P.2b	P.3a	D.1	D.2	D.3
S7	i.2	i.3a	i.3b	i.4a	i.5a	i.6a	i.7a	P.1	P..2a / D1	P.2b	P.3a/D.2b	D.2a	D.3a	i.1
S8	i.1	i.2a	i.2b	i.4a	i.5a	i.6c	i.7a	P.1	D.1a	D.1b	P.2	D.2a	D.3a	i.3a
S9	D.3a	D.2a	D.2b	D.4a	D.5a	i.1	i.3	P.1	??	??	P.2	D.1	i.2	D.6a

Table 7.4: Order the students gave to the steps of an interrupt, a poll and a DMA transfer

7.3 Method used for the complete study

7.3.1 Subjects

The research was carried during 2012-2013 course. Although 78 students were enrolled, the experiment was carried out with only the 39 students that actively completed the whole course. Out of the 39 concept maps gathered, 6 were discarded; 2 because they did not have enough information; 3 because they were too confusing and difficult to interpret; 1 because the writing was impossible to understand.

7.3.2 Procedure

Training the students to build concept maps

It is very important to train students in the use of a new tool; in this case, the concept maps. Even the way in which they are trained can influence the final result [Santhanam, Leach, and Dawson, 1998]. Therefore, the training phase was very carefully planned. Since what we wanted to obtain was the picture of the topic the students had at the end of the semester, the concept mapping training period was extended for the same time devoted to teach/learn the topic.

During the presentation of the topic, the importance we were going to give to the concept maps was emphasized. Then, as an example, the evaluation method for this half (the computer I/O subsystem topic) of the Computer Structure course was explained using a concept map (see Figure 7.4). We wanted the students to be able to build a concept map at the end of the semester and, we therefore made the delivery of the concept maps mandatory. However, we did not want the students to learn the concept maps by rote. This was the tricky part. We clearly stated that concept maps would not be part of the examination. Moreover, the percentage of the score given to the concept maps delivered was very low, 3.5%. This approach carried the risk that the student would not take the concept maps seriously and would build them up without paying attention. In order to avoid this, we explained that concept maps built without any care would be taken as not delivered.

The students were asked to deliver four different concept maps all along the ten week the computer I/O subsystem topic lasted. The concept maps to deliver should be answering the following questions:

1. What is the Von Neumann structure? (Out of the given concepts). (First week)

2. What is needed for I/O to occur, and how does the synchronization work? (Second week)

3. Complete the previous concept map with what you now know. (Fourth week)

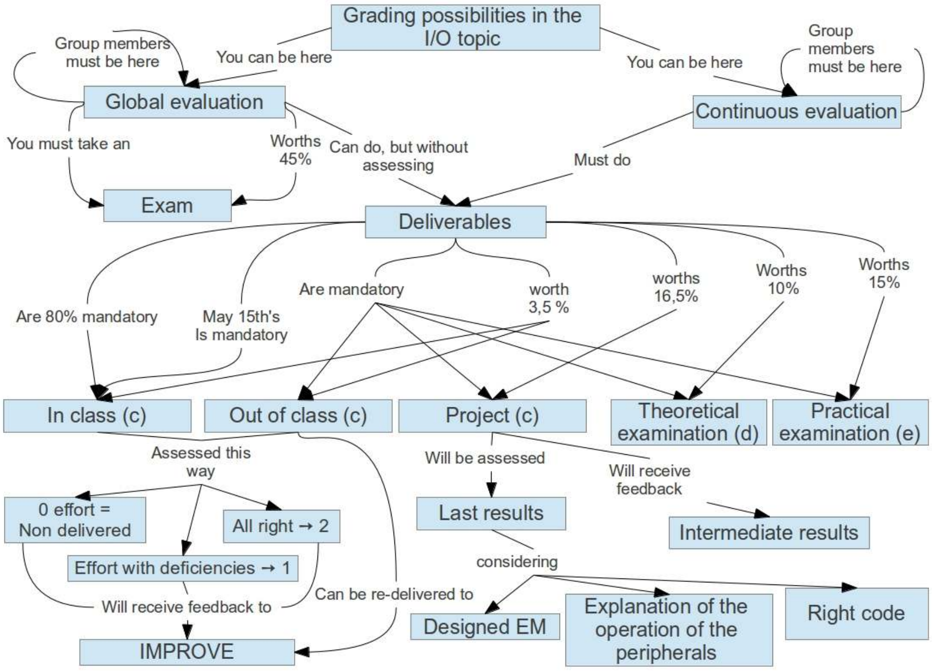


Figure 7.4: Concept map handed out to the students to show the different evaluation options.

4. Complete the previous concept map answering the question: How does DMA work? (Eight week)

All the students received feedback after completing each concept map. We did not want the partial construction of the concept maps and our intervention to have too much effect on the final result. Indeed, it would have been better to use another topic during the training phase, but we had too many time restrictions. Therefore, we decided the feedback would be almost exclusively on the construction of the map. The only feedback statements related to the content itself were as follows:

1. You need to expand the topic further.
2. A concept such as “controller” is too broad, Please be more specific.
3. This is just a classification. You need to try to focus more on the description and the operation level. For example, you said

> there are two synchronization methods, but what do you need synchronization for?

The main feedback given to the students concerning the construction of the map was due to the lack of linking phrases (see Figure 7.5), the construction of block diagrams instead of concept maps (see Figure 7.6), or the use of concepts that were not concepts but whole phrases (even paragraphs taken from the course notes); in other words, a schema that looks like a concept map (see Figure 7.7).

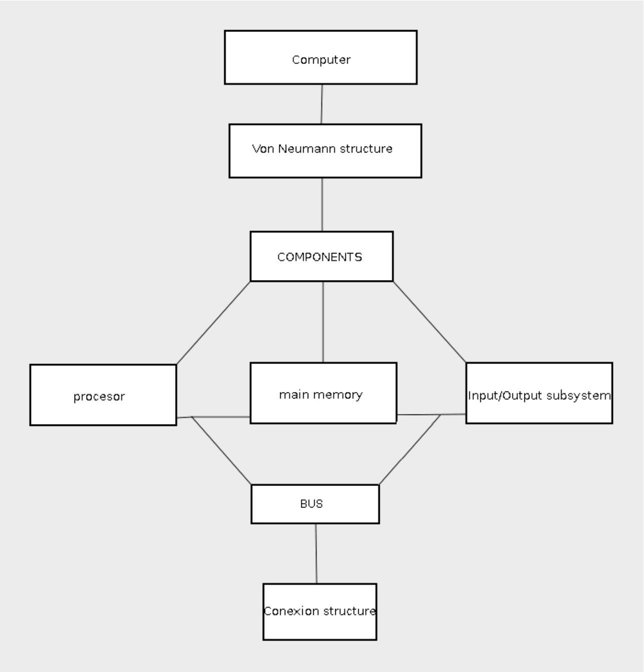


Figure 7.5: A student's concept map to answer the question "What is the Von Neumann structure", without any linking phrase. Translated into English from the original in Basque.

The last concept map

For the last concept map that the students built on their own, we had to tell them to please try to show their own knowledge, because most of them were building the concept maps while reading the course notes.

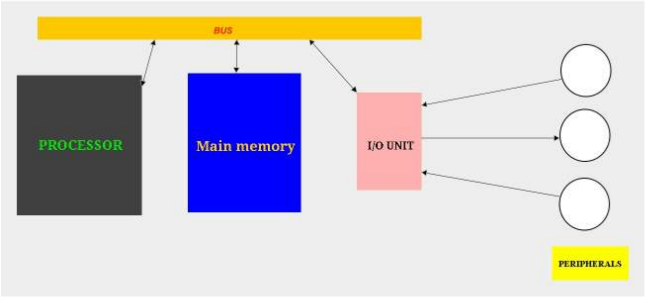


Figure 7.6: A student's trial concept map to answer the question "What is the Von Neumann structure" that turned out to be a block diagram. Translated into English from the original in Basque.

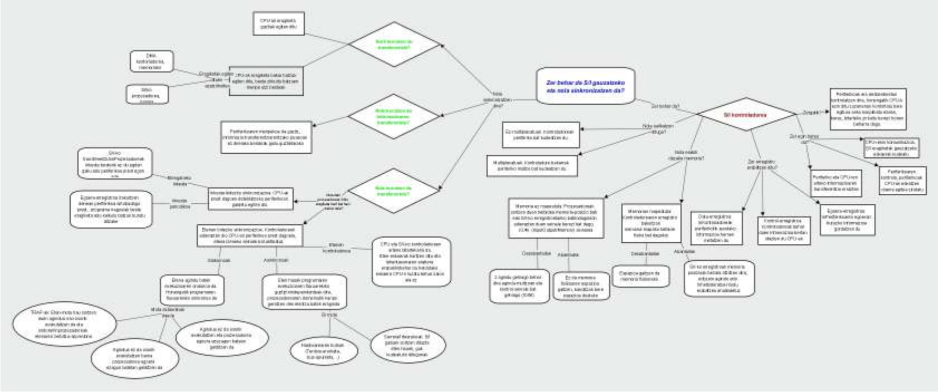


Figure 7.7: Although impossible to read, this is a good example of schema that looks like a concept map.

It is obvious when the concept maps are built that way, because they contain too many specific details.

Finally, the day after the exam, all the students were called to the classroom. They did not know they were going to be asked to build a concept map. They were given twenty selected concepts and they were told they could add more concepts if needed. The students were placed in exam conditions, in order to avoid copying, and they built their concept maps on paper.

The twenty concepts given to the students to build the last concept map were taken from the course notes, according to the weighting they had during the course (e.g. we talked much more about interrupts

Interrupt Service Routine	I/O interrupt
Enable/Disable interrupts	Polled I/O
Read control instructions	I/O register
Interrupt Controller	DMA controller
Analyse status	DMA
Interrupt identification	Peripheral
Interrupt-driven I/O	CPU
Nested interrupts	Subroutine
Interrupt priority	Memory
I/O instructions	I/O controller

Table 7.5: The 20 concepts given to the students to build the last concept map.

and different kinds of interrupts than about DMA, and therefore the percentage of the concepts about interrupts is bigger than the percentage of concepts about DMA). We also considered that the concepts selected appeared in the most used text books in the area [Stallings, 2012; Patterson and Hennessy, 2009; Hamacher et al., 2011; Mano, 1993], analysed in Chaper 3, Section 3.2, although some times the term used changed. The selected concepts can be seen in Table 7.5, in a random order.

Students' feedback

The students were not very happy about having to build the concept maps. Some of them told us personally and it was also reflected in a satisfaction survey they filled out. To average to the question "Was concept mapping helpful in your learning process?", which was delivered in a Likert scale (1: strongly disagree to 5: strongly agree) was 2.47. A chart with the percentage of every possible answer can be seen in Figure 7.8. However, the data obtained with the students' concept maps was essential for this research.

Analysis of the concept maps

Step 1 All the concept maps built by the students in their last deliverable (the one in exam conditions) were digitalized with CM-ED

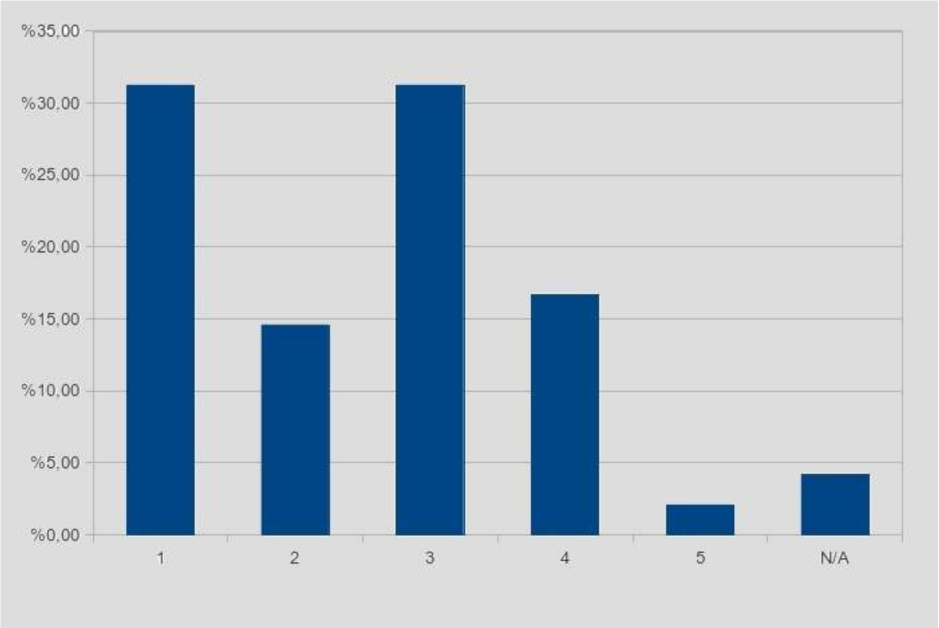


Figure 7.8: Answers to the question "Was concept mapping helpful in your learning process?" in a Likert scale.

[GaLan] software. One big difference between concept maps and social networks is that while in concept maps it is possible to use n-ary relationships, social networks apparently only accept binary relationships. Therefore, some of the relationships had to be changed during this first step. Mainly two kinds of changes where made, as can be seen in Figure 7.9.

Step 2 Once all the concept maps had been digitalized, the essential information that defines the concept maps was extracted from the XML document and converted to PAJEK's format (nodes or concepts vs. vertices, links vs. arcs(directed) or edges (non directed)).

Step 3 All the concept maps were merged. The students had the possibility to add concepts that were not on the list (See Table 7.5). Therefore, it was very important to bear in mind that not all the concept

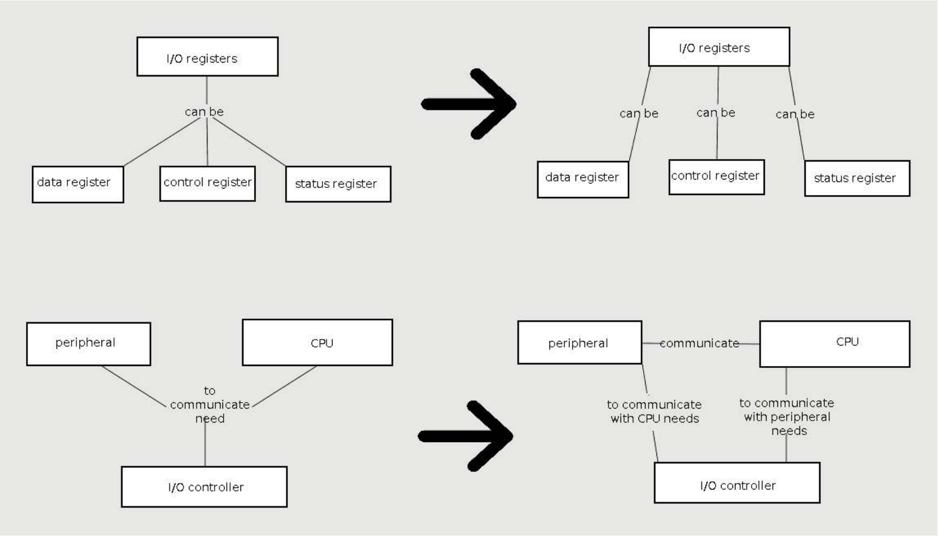


Figure 7.9: Conversions needed to adapt the concept maps for social network analysis.

maps had the same amount of concepts and that the same concept should always be numbered the same.

Step 4 Use PAJEK to improve the readability of the graph.

7.4 Quantitative analysis of the concept maps

Figure 7.10 shows the picture of all the concept maps merged. It is almost impossible to read all the links in that image. However it is clear enough to see that there are some strong links and a lot of very weak links. Moreover, most of the weak links are related to at least one of concept that has been added by the students. These weak relationships hamper reading, and therefore we decided to prune the graph by removing the relationships used by three or less students. We stopped there because if we removed the relationships used by four students, at least one of the concepts of the given list would be left an orphan. The resulting graph is shown in Figure 7.11, where most of the weak connections have been pruned out and only 7 of the 39 concepts added by the students remain connected.

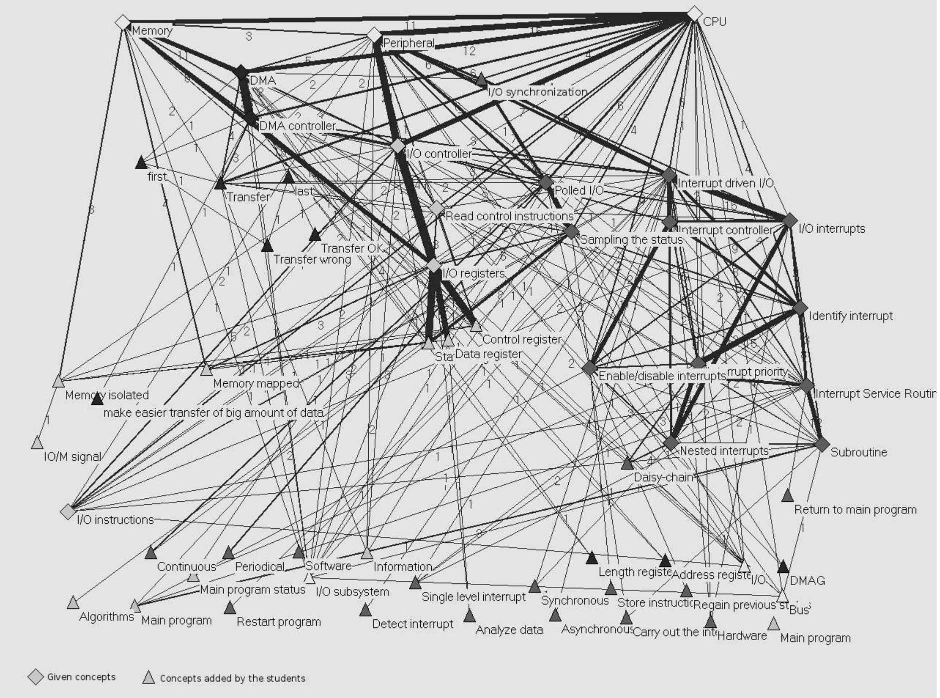


Figure 7.10: Graph showing all the relationships of the students' concept maps.

The concepts given to build the concept map can be divided into four subtopics; the system itself (*CPU, memory, and peripheral*) represented in white; the I/O controller (*I/O controller, I/O registers, read control instructions, I/O instructions*), represented in light grey; synchronization (*polled I/O, interrupt-driven I/O, sampling the status, I/O interrupts, interrupt controller, identify interrupt, enable/disable interrupts, interrupt priority, Interrupt Service Routine, nested interrupts, subroutine*) represented in dark grey; and DMA (*DMA, DMA controller*), represented in black. The concepts added by the students were also classified into these subtopics.

We wanted to determine whether the students were able to correctly relate these subtopics. We shrunk the subtopics in order to be able to depict in a graph (see Figure 7.12) all the connections between subtopics. The graph in Figure 7.12 shows that the "System" subtopic is strongly connected to the other subtopics, but the "I/O controller",

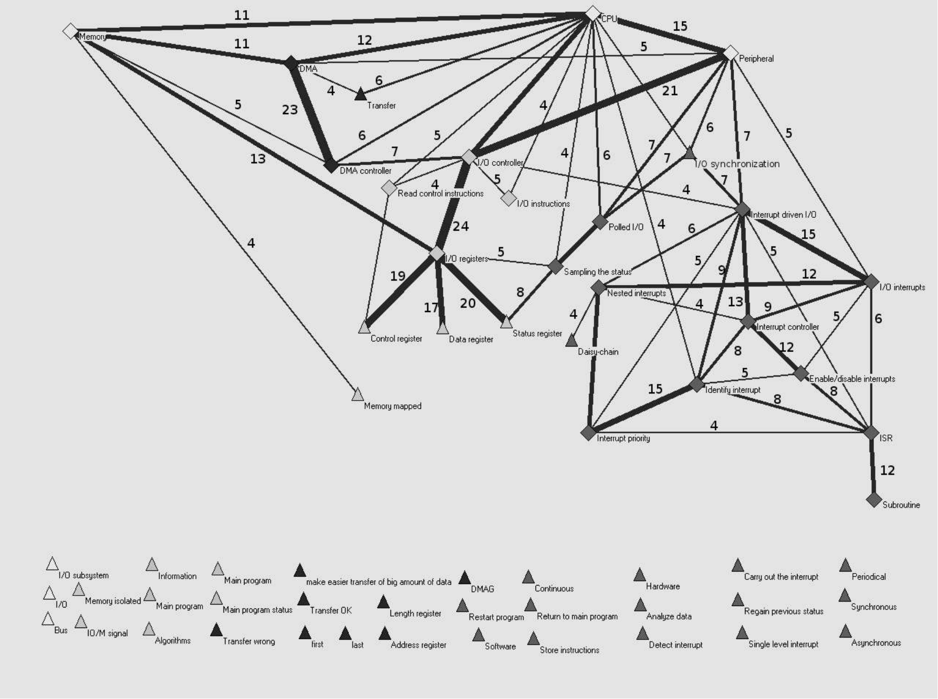


Figure 7.11: Graph showing the concept map links used by more than three students.

"Synchronization", and "DMA" subtopics are either not connected at all or are weakly connected. For example, the CPU in the "System" subtopic and the I/O controller in the "I/O controller" subtopic need synchronization in order to connect, but there are only 17 links between these two subtopics. The DMA controller in the "DMA" subtopic is in fact an I/O controller, but there are only 7 links between these two subtopics. Finally, the DMA controller needs to synchronize with the CPU in order to allow data transfer between memory and a peripheral, but there is no link between the "DMA" and the "Synchronization" subtopics.

In the following subsections, we are going to look more closely at each of the subtopics.

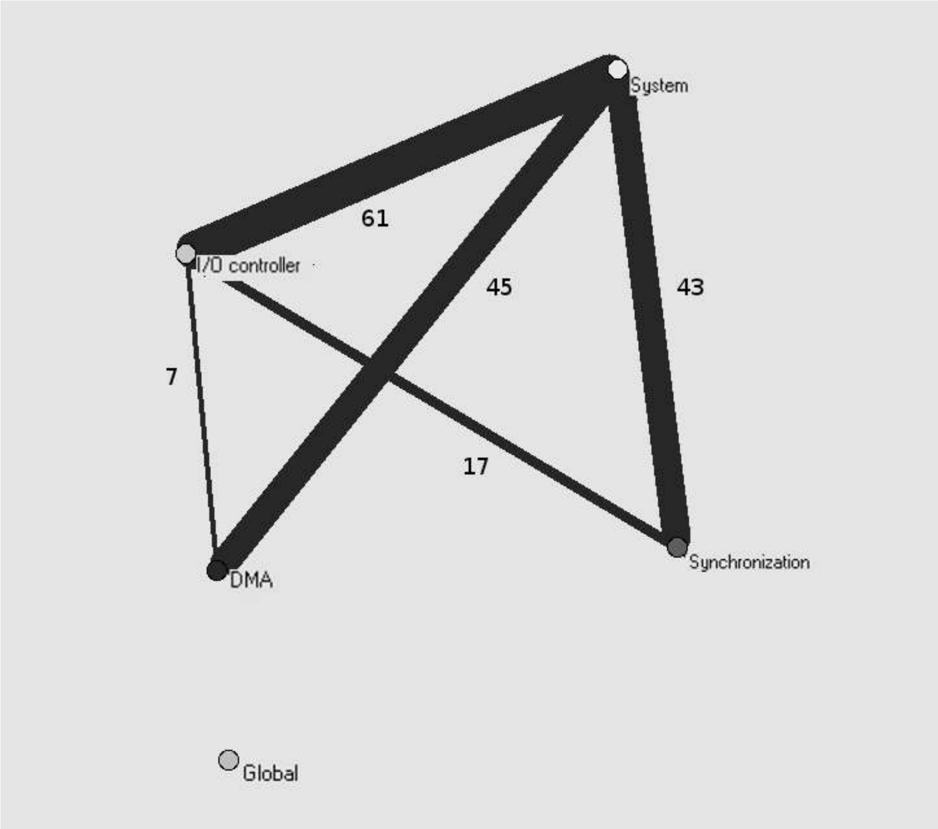


Figure 7.12: Graph showing the links between subtopics.

7.4.1 The system

The computer I/O subsystem is a part of a computer system as are the CPU or the memory, all of them connected by the buses. A program's orders and data are usually stored in memory, but many times data must be introduced by the user via a peripheral device. CPU will control the data transfer helped by the I/O controller. Also it will be the CPU who decides whether the input will be stored in memory or not, or the data going to the output devices comes directly from the processing unit or from memory. But, how do our students see this relation?

"Computer system", *"CPU"*, *"Memory"*, *"Peripheral"*, and *"I/O controller"* could be the concepts to define the computer I/O subsystem if we were

considering the previous definition. The concept *"Computer system"* was not among the concepts given to the students to build their concept map, and therefore we will not consider it here neither. The concept maps built by the students could show many different combinations of those concepts, and indeed they do but some of the options are most repeated and these are the ones we will talk about.

As can be seen in Figure 7.11, 11 students have linked *"CPU"* with *"Memory"*, and 15 with *"Peripheral"*, while *"CPU"* and *"Peripheral"* are linked to *"I/O controller"* by 12 and 21 students respectively. Other weaker links can be found linking these concepts with the other subtopics, but these will be analysed in the next subsections.

7.4.2 Connections inside the I/O controller subtopic

Figure 7.13 shows the relationships inside the "I/O controller" subtopic and the links from this subtopic to the others.

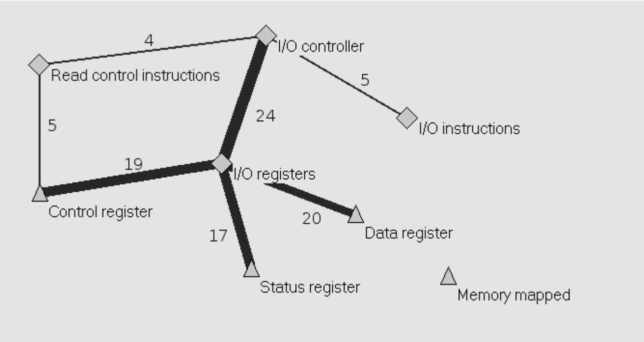


Figure 7.13: Graph showing the links inside the I/O controller subtopic.

Inside the subtopic we find four very strong connections the link between the *I/O controller* and the *I/O registers*. 24 students used this connection in their concept maps (almost 73% of the students). Also, although these concepts were not in the given concept list, 19 (57.5%), 17 (51.5%), and 20 (60.6%) students linked the *I/O registers* to the *control register*, *data register* and *status register* respectively.

While most of the students correctly link the *I/O controller* to its *I/O registers* and even specify that these are the *control, data, and status* *registers*, only 6 (18%) said that the *control instructions are read* from the *control register*, and 4 (12%) that the *I/O controller* is what performs this operation.

Moreover, at least 4 (12%) students used the concept *memory mapped* but did not link it to any of the concepts in its own subtopic, they only linked it to the concept *memory* in the "System" subtopic.

7.4.3 Connections inside the Synchronization subtopic

Figure 7.14 shows the relationships inside the "Synchronization" subtopic. Two *I/O synchronization* methods clearly appear in this graph; *polled I/O* and *interrupt-driven I/O*. Figure 7.14 shows only 7 links (21.2%) to *Polled I/O* and *interrupt-driven I/O*. This is because the graph only shows direct links inside the subtopic. However, when we also consider also the links from the "System" subtopic (see Figure 7.11), the total number of links to *Polled I/O* is 20 (60.6%), and to *interrupt-driven I/O* is 18 (54.5%). Therefore, more than half of the students have distinguished the two synchronization methods.

Polled I/O was linked by 14 students (42.4%) to *Sampling the status*, which, remember, was linked to *I/O registers* by 5 students (15.1%) and to *status register* by 8 students (24.2%); i.e. 13 students in total (39.4%). There are no other significant links to or from these nodes.

The net formed around the *interrupt-driven I/O* concept is much bigger. It has 10 nodes in total, where only one of them was inserted by the students; *Daisy chain*. This makes it difficult to quantitatively analyze, just the direct links between concepts, because two concepts could very well be connected by indirect links that are not considered in this case. However, considering the strongest links (those used by at least 8 students (24.2%)), the graph can be read as follows: *interrupt-driven I/O* needs *I/O interrupts*. The interrupts can be *nested interrupts*. Both kind of interrupts (nested and simple) are controlled by *interrupt controllers*. *Interrupt controllers* can *enable/disable interrupts* and *identify interrupts*, and according to *interrupt priority*, execute *ISRs*. *ISRs* are in fact *subroutines*.

This reading seems good, but, a deeper analysis of the linking words is needed, because, for example, the previous sentence states that *ISRs* are *subroutines*, but the *CPU* executes them directly without a *subroutine* call. Is that what the linking words imply or is it something else? The qualitative analysis needed to be able to say that is detailed in Section 7.5.

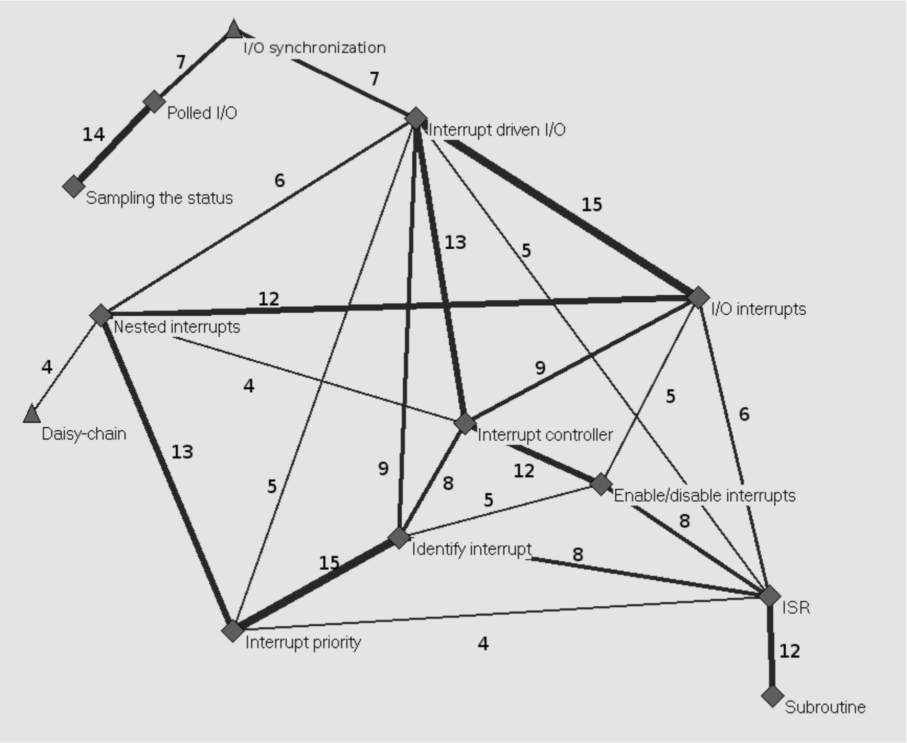


Figure 7.14: Graph showing the links inside the synchronization subtopic.

7.4.4 Connections inside the DMA subtopic

Figure 7.15 shows the relationships inside the “DMA” subtopic, where 23 students (69.7%) link *DMA* to *DMA controller*. A concept added by the students appears in Figure 7.15 (*transfer*). DMA is indeed used for large and continuous transfers of data from a peripheral to the CPU, but only 4 students (12%) used this link.

The following subsections are going to look more closely at these connections between subtopics.

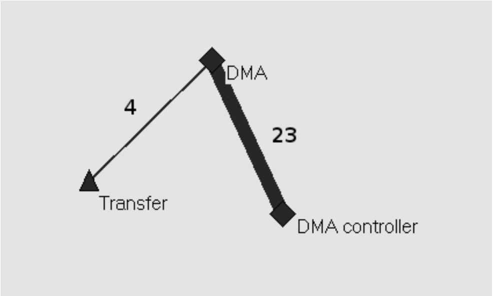


Figure 7.15: Graph showing the links inside the DMA subtopic.

7.4.5 Links between the "I/O controller" and the "synchronization" subtopics

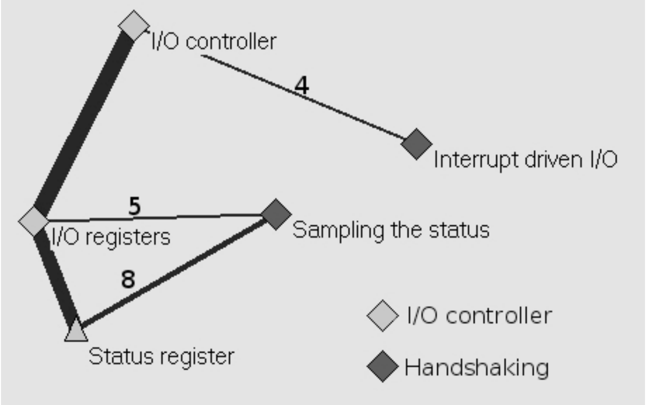


Figure 7.16: Graph showing the links between the I/O controller and the synchronization subtopics.

Figure 7.16 shows that although there are 17 links between the "I/O controller" and the "Synchronization" subtopics these links are divided into the two synchronization methods presented in the subject; *Polled I/O* and *interrupt-driven I/O*. Only 4 students (12%) linked the *I/O controller* to the *interrupt-driven I/O*, while 13 (39.4%) see the relationship the *I/O controller* has with *sampling the status* of the peripheral. Moreover these students see that this has been done via *I/O registers*, and 8 students (24.2%) know that the *I/O register* used for that purpose is the *status register*.

7.4.6 Links between the "I/O controller" and the "DMA" subtopics

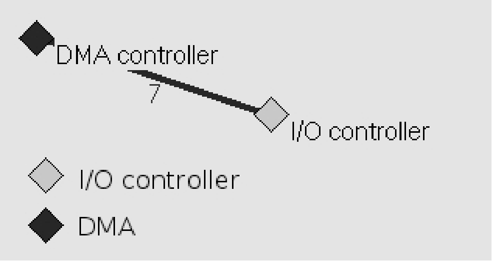


Figure 7.17: Graph showing the links between the I/O controller and the DMA subtopics.

Figure 7.17 shows that 7 students (21.2%) know that the *DMA controller* is in fact an *I/O controller*.

7.4.7 Links between "DMA" and "synchronization" subtopics

Nobody linked the "DMA" subtopic with the "synchronization" subtopic. The *DMA controller* needs to synchronize with the CPU once the transfer is finished. This synchronization is usually done by *interrupt-driven I/O* because of the ability of the *DMA controller* to generate *I/O interrupts*. However the students' concept maps show that this mental model [Gentner and Stevens, 1983] has not been created in the students' minds.

7.5 A more qualitative reading of the concept maps

Now, we would like to look closer at the concept map links. It is important to see not only whether the concepts are linked, but also whether the links are correct. It is difficult to analyse and compare the whole concept maps, and therefore we did it in chunks, as in the quantitative analysis in Section 7.4. All the explanations in the next subsections will be referring to the images of the concept maps built by the students and that can be seen in Apendix D.

7.5.1 The system

We have found several different concept grouping in the analysis of the elements of a computer system.

First of all, the triangle that links the CPU, the peripherals and the I/O controllers. When this option is used, the students tend to specify that the I/O controller is needed in order to manage the communication between the CPU and the peripherals. See Figures D.1, D.4, D.16, D.17, D.18, D.20, D.21, D.22, D.23, D.27, D.31, D.33.

When this option is not used, most of the concept maps do not connect the concepts, or connect them in a disordered way. See Figures, D.2, D.3, D.5, D.6, D.7, D.8, D.12, D.29.

As for the memory, it can be found connected to the CPU, but is mainly found connected only to the I/O registers (see Figures D.1, D.10, D.11, D.14, D.16, D.20, D.22, D.23, D.26, D.31), as the place where to find them, or connected only to the DMA (see Figures D.4, D.9, D.11, D.13, D.17, D.18, D.24, D.27, D.28, D.30). This case will be further analysed in section 7.5.4 where connections inside the DMA subtopic are explained.

Now, let us take a look at the different subtopics and then we will look at the connections between subtopics.

7.5.2 Connections inside the I/O controller subtopic

We have seen that several students know that the CPU and the peripherals communicate with each other helped by the I/O controller. But, what is exactly the I/O controller? An I/O controller can control one or more peripherals and it is composed of several I/O registers that could be mapped into memory or not. Do the students know that?

Most of the students do correctly relate the *"I/O controller"* with the *"I/O registers"*, either directly and saying that the *"I/O controller"* is composed of *"I/O registers"* (see Figures D.2, D.4, D.5, D.6, D.8, D.9, D.12, D.14, D.16, D.17, D.18, D.19, D.20, D.21, D.24, D.25, D.27, D.28, D.30, D.31, D.32, D.33) or that the *"I/O controller"* uses or controls the *"I/O registers"* (see Figures D.1, D.11) or indirectly saying that the I/O controller controls the peripherals by means of the I/O registers (see Figures D.7, D.22). Therefore, it is possible to conclude that most of the students got a correct idea of the *"I/O controller"*. Moreover, most of them also correctly specified the number of the usual *"I/O registers"*, the *"control register"*, the *"data register"*, and the *"status register"*.

The remaining students, do not show a pattern in order to say that there could be a misconception. Only in one case I/O controller and I/O registers are somehow connected, but just because I/O controller reads control instructions in the control register, which is an I/O register (see Figure D.29). For the rest of the students, I/O controller and I/O registers are just not related (see Figures D.3, D.13, D.15, D.23). Some only say that I/O registers are in memory, without saying what they do at all (see Figures D.10, D.26).

Two more concepts, these ones very specific, where related with the I/O controller subtopic. The first one, the *"I/O instructions"*. The I/O instructions were defined as the instructions needed to access I/O registers not mapped into memory. Most of the students did not put this concept into the concept map (see Figures D.2, D.4, D.5, D.6, D.7, D.8, D.11, D.12, D.13, D.17, D.18, D.19, D.20, D.21, D.23, D.24, D.27, D.28, D.30, D.31, D.32). As for the other students, no one used the concept properly (see Figures D.1, D.3, D.9, D.10, D.14, D.15, D.16, D.22, D.25, D.26, D.29, D.33). Therefore it has to be considered that this concept was not understood at all.

The second concept was *"read control instructions"*. This concept was introduced in the concept list in order to see whether the students knew who gives or writes control instructions into the control register, and who reads those instructions. Once again, many students did not use this concept in their concept maps (see Figures D.1, D.5, D.6, D.8, D.11, D.13, D.15, D.16, D.18, D.19, D.21, D.23, D.25, D.27, D.30, D.31). However in this case there was at least one student who did say that it is the peripheral who reads control instructions from the CPU in the control register (see Figure D.20). Two more students said that it is the I/O controller who reads control instructions from the control register in order to know what should the peripheral be doing (see Figure D.22, D.29). Being the control register who reads the control instructions has a few followers (see Figures D.3, D.4, D.9, D.28), as does the option of being the CPU who reads the control instructions (see Figures D.2, D.10, D.17, D.32).

7.5.3 Connections inside the Synchronization subtopic

The synchronization between the CPU and the peripheral, in order the information exchange to be performed correctly, can be done in at least two ways, by polling and by interrupt. Let us then consider each of them separately in the following subsections.

Polled I/O

Considering polled I/O, we could be asking three different questions: What is it for? How is it done? Who does it?

Answering to the first question, **what is polled I/O for?**, we would say that it is a way to synchronize the communication between the CPU and the peripherals, as has been graphically expressed in Figures D.13, D.17, D.20, D.22, D.24, D.27, and D.28. Some other students say that it is a way of synchronizing the I/O subsystem (see Figures D.1, D.3, D.9, D.32, D.14, D.15, D.31), which would be a good answer with a clear idea of what the computer I/O subsystem is. Another option has been saying that it is a way to control or synchronize peripherals (see Figures D.11, D.19, D.25). This option misses who needs the peripheral to be synchronized with.

A surprising option is the one that says the *"polled I/O"* is a kind of I/O interrupt (see Figures D.16, D.23, D.29). This is something that needs to be cleared up. The difference between interrupts and polls should be stressed.

And, **how is this synchronization be done?**. Analysing the peripheral's status, by continuously reading the I/O registers (more specifically the status register) (see Figures D.2, D.19, D.24, D.30, D.31). Many students know that it is necessary to analyse the status, but what the concept maps do not reflect is whether they know what does this mean. Analysing the status of whom? (see Figures D.1, D.6, D.12). Where is this analysis performed? (See Figures D.22, D.25). However again the biggest problem encountered here is the mixing up of interrupts and polling (see Figures D.16, D.18, D.29).

Finally, it is also important to know **who performs the analysis of the status of the peripheral**, and this is something that most of the students just do not mention (see Figures D.3, D.5, D.6, D.7, D.8, D.9, D.10, D.11, D.13, D.14, D.15, D.16, D.17, D.18, D.20, D.22, D.23, D.24, D.25, D.27, D.28, D.29, D.32, D.33), and only a few ones get to say that it is the CPU the responsible of that action (see Figures D.2, D.12, D.19, D.21, D.30, D.31).

Interrupt-driven I/O

For interrupt-driven I/O we could formulate the same questions as for polled I/O: What is it for? How is it done? Who does it?

For the first question, we get more or less the same answers as in the same question of the previous section (Section 7.5.3), since, as a way of synchronizing the communication between the CPU and the peripherals, polled I/O and interrupt-driven I/O are parallel.

We will now first look at the **who**, which, in this case is double question. *Who interrupts* and *who is interrupted*. This is something that has not been expressed in the students' concept maps. The majority of the students did not answer any of the questions and just a few said that it is the peripherals who interrupt (see Figures D.1, D.2, D.11, D.19).

As for the **how**, the steps to be followed after an interrupt request are not clear for the students. Figure 7.14 shows that there is not a specific organization and looking to the linking phrases does not improve the situation.

The interrupt controller should be analysing whether interrupts are enabled or disabled, identify the interrupt request, if multiple interrupts have been requested the interrupt controller should analyse their priority, and if everything is permitting the interrupt, then execute the ISR corresponding to the request, disabling interrupts if nested interrupts are not allowed. However for some students the only task of the interrupt controller is to enable/disable interrupts (see Figures D.4, D.13, D.16, D.33). Some others state that together with the DMA controller, the interrupt controller is a kind of I/O controller (see Figures D.3, D.12, D.15).

The ISR is a subroutine executed asynchronously, triggered by the CPU and not by the program. The nearest to that, what we can find in the concept maps is ISRs operate similar to subroutines (see Figure D.12). Then some students say that ISRs are subroutines (see Figures D.2, D.7, D.15, D.22, D.29, D.33).

7.5.4 Connections inside the DMA subtopic

DMA is used for big transfers. When using DMA, peripherals and memory get connected directly so that the CPU does not need to take care of the transfer of each little bit of information. The CPU programs the peripheral's I/O controller and the DMA controller, and continues working until at the end of the transfer, the DMA controller triggers an interrupt request. Therefore, inside the subtopic, as shows Figure 7.15 the only substantial link is between the DMA and the DMA controller. However, many students missed that link (see Figures D.1, D.2, D.7, D.12, D.16, D.21, D.24, D.31).

Most of the links of the DMA subtopic should be with other subtopics, mainly with the synchronization subtopic and with the system. These links between subtopics will be considered in subsections 7.5.6 and 7.5.7.

7.5.5 Links between the I/O controller and the synchronization subtopics

These two subsections should be linked. Indeed as the synchronization subtopic encompasses the polled I/O and the interrupt-driven I/O, the link between the two subtopics will be considered also in two parts, the link between the I/O controller and the polled I/O on the one hand, and the I/O controller and the interrupt-driven I/O on the other hand.

Links between the I/O controller and the polled I/O

The links between the I/O controller and the polled I/O have already been analysed in section 7.5.3, when talking about the how question, since how polled I/O is performed is what links it with the I/O controller; the CPU samples the status of the peripheral by asking the status register whether the peripheral is ready or not to perform the I/O action. The analysis shows that many students do not know what sampling the status is, since they do not link it with polled I/O (see Figures D.3, D.4, D.7, D.8, D.17, D.20, D.28, D.32), neither do they say the status of whom should be sampled. Some of those who know it, do not know how to do it (see Figures D.10, D.12, D.16, D.22, D.25).

Links between the I/O controller and the interrupt-driven I/O

Should there exist a link between the I/O controller and the interrupt-driven I/O? Of course there should, but maybe not a direct link as the one shown in Figure 7.16. These links say that the I/O controller is a kind of interrupt-driven I/O (see Figures D.6, D.33) or that interrupt-driven I/O is a way of synchronizing the I/O controller (with whom?) (see Figure D.21).

The I/O controller should be linked with the ISR, since once the interrupt has been accepted, the ISR that is executed is the one accessing the I/O registers of the I/O controller in order to be able to decide what the peripheral should be doing or to get the data the peripheral and the CPU are interchanging. This link has been shown only in one concept map, and this link says that the I/O controller reads the ISR (see Figure D.23).

7.5.6 Links between the DMA controller and the I/O controller

The DMA controller is indeed an I/O controller that connects directly memory and peripheral's I/O controllers and it has its own I/O registers. Some students get this idea partially, saying that DMA controller is connected to I/O controllers (see Figures D.2, D.12). Some others get to say that the DMA controller controls direct communication between DMA and peripherals (see Figures D.24, D.30), and some other say that DMA controller has its I/O registers (see Figures D.5, D.9, D.32). The rest of the connections between DMA controller and I/O controller are either wrong (see Figures D.3, D.6, D.7, D.23), or non-existent (see Figures D.1, D.4, D.8, D.10, D.11, D.13, D.14, D.15, D.16, D.17, D.18, D.19, D.20, D.21, D.22, D.25, D.26, D.27, D.28, D.29, D.31, D.33).

7.5.7 Links between the DMA and the synchronization subtopics

As seen in Figure 7.12, there is no connection between the DMA and the synchronization subtopics used by more than three students. However one student correctly pointed that the DMA controller, at the end of the data transfer, synchronizes by interrupt- driven I/O (see Figure D.6), and another students that DMA controller controls DMA either by polled I/O or interrupt-driven I/O (see Figures D.26).

7.6 Conclusions

In this research we used concept maps to identify strengths and weaknesses in students' understanding of the computer I/O subsystem. Tools from social network analysis were used for the quantitative analysis of the merged concept maps, and the results are shown.

Those results show that in the students' minds there are three mostly unconnected subtopics: the “I/O controller”, the “Synchronization”, and the “DMA”. The qualitative analysis brought a deeper insight showing that there is mostly one clear idea, which is that the I/O controller has I/O registers. The rest is overall not clear:

- system's concepts are connected disorderedly, and sometimes memory is only connected to I/O registers or DMA;
- I/O instruction is an unknown concept;

- I/O registers or I/O controller being the readers of the control instructions is an spread misconception;
- Polling and interrupts are often mixed up;
- The processes in order to poll the I/O or to manage interrupts are not clear;
- DMA and its management have not been understood at all.

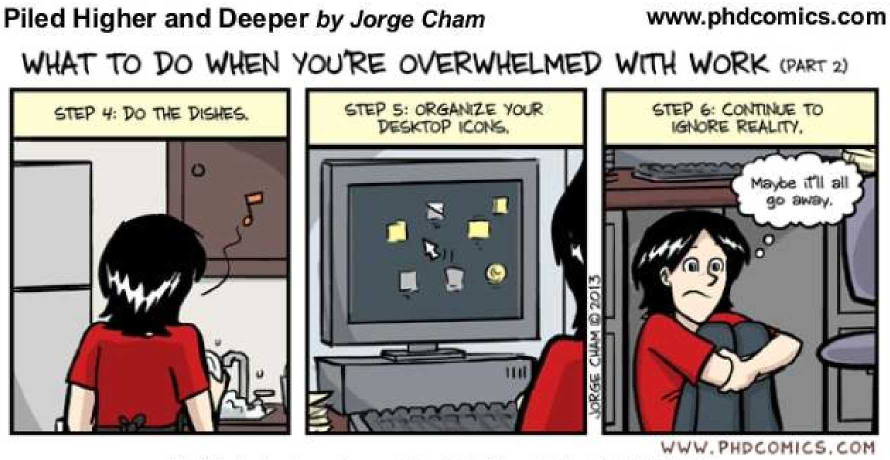
The quantitative picture that the merged network reveals can be very helpful when individual concept maps have to be analysed in order to give feedback to the students. However, this process should be somehow automatized in order to be able to give that feedback in a reasonable amount of time.

It was a shame that concept mapping was so badly taken by the students. At least, the data obtained were very descriptive about the knowledge of the students about the topic, and will be very useful to improve the teaching. Achieved data will be used in subsequent academic years in order to better clarify the worst understood concepts and then reinforce I/O subsystem learning.

In Chapter 6 the dissatisfaction of the students towards this tool has been made evident. However it has been very useful to see what do they get to understand, and the year where they were used, the passing rate raised. Therefore, although some adaptations could be required in the use of the concept maps, we will probably continue using it, until more evidence is gathered.

Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
WHAT TO DO WHEN YOU'RE OVERWHELMED WITH WORK
STEP 1: MAKE A LIST OF ALL THE THINGS YOU HAVE TO DO.
STEP 2: CATEGORIZE THEM BASED ON THEIR IMPORTANCE.
TO DO:
IMPORTANT FOR MY THESIS!
IMPORTANT FOR MY FUNDING!
IMPORTANT FOR MY BILLS!
IMPORTANT FOR MY PROFESSOR!
IMPORTANT FOR MY CAREER!
IMPORTANT FOR MY SANITY!
JORGE CHAM © 2013
STEP 3: REALIZE YOU JUST MADE IT WORSE.
AAH! THEY'RE ALL IMPORTANT!!
WWW.PHDCOMICS.COM

title: "What to do when you\'re overwhelmed" - originally published 11/20/2013


Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
WHAT TO DO WHEN YOU'RE OVERWHELMED WITH WORK (PART 2)
STEP 4: DO THE DISHES.
STEP 5: ORGANIZE YOUR DESKTOP ICONS.
JORGE CHAM © 2013
STEP 6: CONTINUE TO IGNORE REALITY.
Maybe it'll all go away.
WWW.PHDCOMICS.COM

title: "What to do when you're overwhelmed, Part 2" - originally published 11/22/2013

CHAPTER 8

Overall Conclusions and Future Work

8.1 Conclusions

The TPACK model described in Chapter 2 draws a whole picture of the work carried out for this dissertation. Technological, pedagogical, and content knowledge have been studied and grouped in order to be able to perceive the needs of the computer I/O subsystem as a teaching topic.

In order to strengthen the content knowledge I searched the bibliography (both textbooks and research literature) and different universities' syllabi for the topic (see Chapter 3). In this dissertation I found that four different approaches are being used in order to teach the computer I/O subsystem.

- The first one is purely descriptive, and it does not expect a deep knowledge from the students.
- The second one lies in the fact that the computer I/O subsystem is a bottleneck in the computer system, and it wants the student to be able to calculate the performance of the system while using different settings of the I/O subsystem.

- The third approach intends the students to be able to control the peripherals of a computer system by configuring and using their controllers' registers.
- The fourth approach places a great emphasis on the hardware level control signals needed for the I/O subsystem to work correctly.

The technological and pedagogical constructs of the TPACK model are too large to be considered without the limits of the content to be taught. The research literature in this context has been analysed and found that several simulators and real machines are being used to teach the computer I/O subsystem. As with the pedagogical construct, the literature found makes a commitment towards participatory learning strategies and project-based learning.

I chose the Nintendo DS machine to move forward with the research. From a technological content knowledge perspective I analysed the system and prepared it for its use in the subject (see Chapter 4).

All this technological, pedagogical and content knowledge was combined and empirically tried out during a period of three school years, using the NDS machine, following the programming approach and a PBL methodology. Chapter 5 describes the development and changes in the course of the years, while in Chapter 6 the data gathered is analysed. The data show that the combination of game-design and project-based learning approach is attractive to the students and make pass rates improve, answering to the two first research questions:

Q1: How well can a handheld game console be adapted to the teaching needs of the I/O subsystem topic?

Q2: How well can a PBL methodology help in the learning process of the I/O subsystem topic?

However, the model revealed a very important hole in the knowledge to teach the topic effectively. This hole centres in students' knowledge.

I then chose to follow this path and analyse the students' knowledge at the end of the subject. A combination of concept maps and social network analysis was used to first to elicit students' knowledge on the relationships among the concepts studied in the topic, and then analyse the concept maps gathered.

This study revealed a very interesting knowledge. The students were unable to make relationships between different subtopics in the subject. This gives us some ideas about what is missed in the educational process of the I/O subsystem and which are the concepts that have to be reinforced next years, and answers to the third research question:

Q3: How do students understand the I/O subsystem topic?

8.2 Future work

In an almost untouched field as the educational process of the computer I/O subsystem is, this dissertation is no more than a door opened to a corridor full of new doors asking to be opened.

The programming approach has been selected for this research. One of the doors in the corridor is looking forward for someone to deeper study the other approaches.

During the dissertation I have remarked on the fact that the study was carried out in an undergraduate introductory level. Behind one of the doors the computer I/O topic spreads towards more advanced subjects such as operating systems, embedded systems, real time systems, etc. How does the approach selected affect the understanding and learning process in these subjects? This is another path to go through.

When opening these first mentioned doors a wide open area is found. However, there is a whole set of doors that open towards the same computer I/O subsystem at the undergraduate introductory level in which this dissertation has been working on. There are still questions to answer there. For example:

- How to triangulate the pre and post subject students' knowledge and the work they have been doing?
- How to set the subject so that a progressive learning is ensured?
- How well can serious games (since we are already using a game console) help in that endeavour? In order to start the path to give an answer to that question, the design of a question and answer collaborative game have been presented in [Larraza-Mendiluze and Garay-Vitoria, 2009] and [Larraza-Mendiluze and Garay-Vitoria, 2010].

- How can this progressive knowledge be assessed? The path to answer this last question has already been started in [Larraza-Mendiluze and Garay-Vitoria, 2013b]. However, this is no more than a starting point. To be able to answer the question, more than the analysis of the questions used in the exam of a few universities will be needed.

There is still another much more technical door that opens towards a study on how to combine in one the concept mapping tool and the social network analysis tool, and then spread away in order to be able to compare knowledge in different universities and even in different subjects.

Another door opens towards widening the technological knowledge, by studying whether mobile devices can be used to teach the computer I/O subsystem as we use the Nintendo DS.

8.3 Publications obtained from this work

1. E. Larraza-Mendiluze, N. Garay-Vitoria, J. Martín, J. Muguerza, T. Ruiz-Vázquez, I. Soraluze, J. Lukas, and K. Santiago. *Nintendo DS projects to learn computer input-output*. In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, ITiCSE '12, pages 373–373, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1246-2.
doi: 10.1145/2325296.2325388. (Classified in category A in the CORE 2013 ranking).

2. E. Larraza-Mendiluze and N. Garay-Vitoria. *A comparison between lecturers' and students' concept maps related to the input/output topic in computer architecture*. In Proceedings of the 12th Koli Calling International Conference on Computing Education Research, Koli Calling '12, pages 57–66, New York, NY, USA, 2012. ACM Press. doi: 10.1145/2401796.2401803. (Classified in category B in the ERA 2010 ranking).

3. E. Larraza-Mendiluze and N. Garay-Vitoria. *The Learning Outcomes of the Exam Question in the Input/Output Topic in Computer Architecture*, pages 212–215. LaTiCE '13. Institute of Electrical and

Electronics Engineers, IEEE, 2013b. ISBN 978-1-4673-5627-5. doi: 10.1109/LaTiCE.2013.13.

4. E. Larraza-Mendiluze and N. Garay-Vitoria. *Use of concept maps to analyze students' understanding of the I/O subsystem*. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, pages 67–76, New York, NY, USA, 2013a. ACM Press. doi: 10.1145/2401796.2401803. (Classified as category B in the ERA 2010 ranking).

5. E. Larraza-Mendiluze, N. Garay-Vitoria, J. Martín, J. Muguerza, T. Ruiz-Vázquez, I. Soraluze, J. Lukas, and K. Santiago. *Game-Console-Based Projects for Learning the Computer Input/Output Subsystem*. IEEE Transactions on Education, 56(4):453–458, Nov. 2013. ISSN 0018-9359. doi: 10.1109/TE.2013.2255877. (Classified as Q2 in the category Education, Scientific disciplines in the 2012 ISI/JCR Science Edition with an impact factor of 0.950 and 5-years impact factor of 1.177).

6. E. Larraza-Mendiluze, N. Garay-Vitoria. *Approaches and Tools used to Teach the Computer Input/Output Subsystem at Undergraduate Introductory Computer Architecture and Organization Courses*. IEEE Transactions on Education, –Under review, minor changes needed– (Classified as Q2 in the category Education, Scientific disciplines in the 2012 ISI/JCR Science Edition with an impact factor of 0.950 and 5-years impact factor of 1.177).

title: "Written" - originally published 7/5/2013

Bibliography

J. Accarrino. Install Homebrew Games on Your Sony PSP, 2005. URL `http://www.methodshop.com/gadgets/tutorials/pspinstallgames/index.shtml`. [Retrieved 10/18/2013].

E. Aronson, N. Blaney, C. Stephin, J. Sikes, and M. Snapp. *The jigsaw classroom.* Sage Publishing Company, 1978.

A. Arruarte, J. Elorriaga, and U. Rueda. A template-based concept mapping tool for computer-aided learning. In *Advanced Learning Technologies, 2001. Proceedings. IEEE International Conference on*, pages 309–312, 2001. doi: 10.1109/ICALT.2001.943931.

V. Batagelj and A. Mrvar. Pajek, Program for Analysis and Visualization of Large Networks, Reference Manual, 2011. URL `http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf`. [Retrieved 07/03/2013].

A. Berglund. *Learning computer systems in a distributed project course: The what, why, how and where*. PhD thesis, Uppsala University, 2005.

A. Berglund and A. Eckerdal. What do CS Students Try to Learn? Insights from a Distributed, Project-based Course in Computer Systems. *Computer Science Education*, 16(3):185–195, 2006.

A. Berglund, M. Daniels, and A. Pears. Qualitative research projects in computing education research: an overview. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 25–33, Darlinghurst, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-34-1.

J. Biggs and K. Collis. *Evaluating the quality of learning: The SOLO Taxonomy*, volume 1. Wiley Online Library, 1982.

M. Black. EmuMaker 86, 2013. URL `http://http://www.emumaker86.org/`. [Retrieved 09/23/2013].

M. Black and P. Komala. A full system x86 simulator for teaching computer organization. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 365–370, Dallas, TX, USA, 2011. ACM. ISBN 978-1-4503-0500-6. doi: 10.1145/1953163.1953272.

M. Black and N. Waggoner. Emumaker86: a hardware simulator for teaching cpu design. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 323–328, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445294.

B. Bloom. *Taxonomy of educational objectives: the classification of educational goals*. Number v. 1 in Taxonomy of Educational Objectives: The Classification of Educational Goals. D. McKay, 1956.

S. Boss and J. Krauss. *Reinventing Project-Based Learning*. International Society for Technology in Education, 2007.

J. Boustedt. *On the road to a software profession: Students' experiences of concepts and thresholds*. PhD thesis, Umeå University, 2010. URL `http://uu.diva-portal.org/smash/record.jsf?pid=diva2:309647`. [Retrieved 06/28/2013].

M. Brorsson. Mipsit: a simulation and development environment using animation for computer architecture education. In *Proceedings of the 2002 workshop on Computer architecture education: Held in conjunction with the 29th International Symposium on Computer Architecture*, WCAE '02, New York, NY, USA, 2002a. ACM. doi: 10.1145/1275462.1275479.

M. Brorsson. MipsIt Laboratory Exercise 3, 2002b. URL `http://www.elsevierdirect.com/companions/9780123744937/exercises/03~3_Assembly_Language_Programming_Interrupts_and_the_OS_Interface.pdf`. [Retrieved 09/25/2013].

M. Brorsson. MipsIt simulator, 2002c. URL `http://www.elsevierdirect.com/companions/9780123744937/Mipsit.zip`. [Retrieved 09/25/2013].

D. Brylow and B. Ramamurthy. Nexos: a next generation embedded systems laboratory. *SIGBED Review*, 6(1):7, 2009a.

D. Brylow and B. Ramamurthy. Nexos: a next generation embedded systems laboratory., 2009b. URL `http://www.cse.buffalo.edu/nexos/index.html`. [Retrieved 09/25/2013].

J. Buzen. I/O subsystem architecture. *Proceedings of the IEEE*, 63(6):871–879, 1975. ISSN 0018-9219. doi: 10.1109/PROC.1975.9852.

L. B. Cassel, M. Holliday, D. Kumar, J. Impagliazzo, K. Bolding, M. Pearson, J. Davies, G. S. Wolffe, and W. Yurcik. Distributed expertise for teaching computer organization & architecture. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '00, pages 111–126, New York, NY, USA, 2001. ACM. doi: 10.1145/571968.571973.

CC2001. *Computing Curricula 2001 Computer Science, Final Report*. The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery), 2001. URL `http://www.acm.org/education/education/education/curric_vols/cc2001.pdf`. [Retrieved 06/28/2013].

CC2004. *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery), 2004. URL `http://www.acm.org/education/education/curric_vols/CE-Final-Report.pdf`. [Retrieved 06/28/2013].

CC2008. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery), 2008. URL `http://www.acm.org/education/curricula/ComputerScience2008.pdf`. [Retrieved 06/28/2013].

CC2013. Computer science curricula 2013 (ironman draft v.0.8), 2013. URL `http://ai.stanford.edu/users/sahami/CS2013//ironman-draft/cs2013-ironman-v0.8.pdf`. [Retrieved 06/28/2013].

S. Cox. *A conceptual analysis of technological pedagogical content knowledge*. PhD thesis, 2008. URL `http://contentdm.lib.byu.edu/cdm/ref/collection/ETD/id/1486`. [Retrieved 02/12/2013].

Cybermetrics_Lab. Ranking Web of Universities, 2013. URL `http://www.webometrics.info/en/world`. [Retrieved 09/24/2013].

W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Structural Analysis in the Social Sciences. Cambridge University Press, 2005. ISBN 9780521602624. URL `http://books.google.es/books?id=beRRM_GH1YkC`. [Retrieved 07/03/2013].

devkitPro. URL `http://devkitpro.org`. [Retrieved 07/19/2013].

A. Díaz Lantada, P. Lafont Morgado, J. Munoz-Guijosa, J. Muñoz Sanz, J. Echávarri Otero, J. Muñoz García, E. Chacón Tanarro, and E. De La Guerra Ochoa. Towards Successful Project-Based Teaching-Learning Experiences in Engineering Education. *International Journal of Engineering Education*, 29(2):476–490, 2013.

J. Djordjevic, A. Milenkovic, and N. Grbanovic. An integrated environment for teaching computer architecture. *Micro, IEEE*, 20(3):66–74, 2000.

J. Donaldson, R. Salter, and R. Punch. DLSys: A Toolkit for Design and Simulation of Computer System Architecture. In *Proceedings of the 2011 workshop on Computer architecture education*, WCAE'11, pages 0–6, San Antonio, TX, USA, 2011a.

J. Donaldson, R. Salter, and R. Punch. DLsim, 2011b. URL `http://www.cs.oberlin.edu/~rms/dlsim.com`. [Retrieved 09/23/2013].

A. Eckerdal. *Novice programming students' learning of concepts and practise*. PhD thesis, Uppsala University, 2009. URL `http://user.it.uu.se/~annae/FullAvh-Spikenheten.pdf`. [Retrieved 06/28/2013].

D. Ellard. Ant-32 Assembly Language Tutorial, 2003. URL `http://ellard.org/dan/www/pubs/ant32_tutorial.pdf`. [Retrieved 09/25/2013].

D. Ellard, D. Holland, N. Murphy, and M. Seltzer. On the design of a new CPU architecture for pedagogical purposes. In *Proceedings of the 2002 workshop on Computer architecture education: Held in conjunction with the 29th International Symposium on Computer Architecture*, WCAE '02, New York, NY, USA, 2002. ACM. doi: 10.1145/1275462.1275471.

A. Elliot Tew. *Assessing fundamental introductory computing concept knowledge in a language independent manner*. PhD thesis, Georgia Institute of Technology, 2010. URL `http://hdl.handle.net/1853/37090`. [Retrieved 06/28/2013].

A. Elliott Tew, B. Dorn, W. Leahy, Jr., and M. Guzdial. Context as support for learning computer organization. *J. on Educational Resoures in Computing*, 8(3):8:1–8:18, Oct. 2008. ISSN 1531-4278. doi: 10.1145/1404935.1404937.

S. Fincher and M. Petre. *Computer Science Education Research*. Taylor & Francis, 2004. ISBN 9789026519697.

S. Fincher, M. Petre, and M. Clark, editors. *Computer Science Project Work: Principles and Pragmatics*. Springer-Verlag, London, UK, UK, 2001. ISBN 1-85233-357-X.

L. Freeman, M. Everett, and S. Borgatti. UCINET software. URL `https://sites.google.com/site/ucinetsoftware/home`. [Retrieved 07/03/2013].

GaLan. CM-ED (Concept Maps EDitor). URL `http://galan.ehu.es/Galan/node/34`. [Retrieved 07/03/2013].

N. Garay, E. Larraza, J. Martín, T. Ruiz, and I. Soraluze. Arquitectura de computadores. *Open Course Ware - Universidad del País Vasco (UPV/EHU)*, 2010. URL `http://ocw2010.ehu.es/file.php/103/arquitectura/arquitectura_com/Course_listing.html`. [Retrieved 07/19/2013].

N. Garay-Vitoria. Hci design is not only software design. In *Proceedings CONVIVIO Faculty Forum: Teaching Design for HCI*, 2006.

D. Gentner and A. Stevens. *Mental Models*. Cognitive Science - Lawrence Erlbaum Associates. Lawrence Erlbaum Associates, 1983. ISBN 9780898592429. URL `http://books.google.es/books?id=QFI0SvbieOcC`. [Retrieved 07/03/2013].

GP2Xa. GP2X Caanoo. URL `http://gp2xwiz.info/caanoo.asp`. [Retrieved 10/18/2013].

GP2Xb. GP2X Wiz Portable Arcade & Console Emulator! URL `http://gp2xwiz.info`. [Retrieved 10/18/2013].

C. R. Graham. Theoretical considerations for understanding technological pedagogical content knowledge (TPACK). *Computers & Education*, 57(3):1953 – 1960, 2011. ISSN 0360-1315. doi: 10.1016/j.compedu.2011.04.010.

C. Hamacher, Z. Vranesic, S. Zaky, and N. Manjikian. *Computer organization and embedded systems*. McGraw-Hill Science/Engineering/Math, 6th edition, 2011. ISBN 9780073380650. URL `http://highered.mcgraw-hill.com/sites/0073380652/`.

N. Hanakawa, G. Yamamoto, K. Tashiro, H. Tagami, and S. Hamada. p-HInT: Interactive Educational environment for improving large-scale lecture with mobile game terminals. In *Proceedings of the 16th International Conference on Computers in Education*, pages 629–634, 2008.

M. Hewner. *Student conceptions about the field of computer science*. PhD thesis, Georgia Institute of Technology, 2011. URL `http://hdl.handle.net/1853/45890`. [Retrieved 06/28/2013].

P. Hirst. The G-Factor International University Ranking, 2008. URL `http://web.archive.org/web/20090226044237/http://universitymetrics.com/gfactor2006top300`. [Retrieved 09/24/2013].

P. Hubwieser and A. Mühling. What students (should) know about object oriented programming. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 77–84, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0829-8. doi: 10.1145/2016911.2016929.

P. Hubwieser, J. Magenheim, A. Mühling, and A. Ruf. Towards a conceptualization of pedagogical content knowledge for computer science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 1–8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2243-0. doi: 10.1145/2493394.2493395.

ICER. ICER Conference. `http://icer.hosting.acm.org/`. [Retrieved 07/09/2013].

Informatics_Institute_of_Middle_East_Technical_University. University Ranking by Academic Performance, 2012. URL `http://www.urapcenter.org/2012/world.php?q=MS0yNTA=`. [Retrieved 09/24/2013].

ITICSE. ITICSE Conferences. `http://www.sigcse.org/events/iticse`. [Retrieved 07/09/2013].

D. Johnson, R. Johnson, and E. Holubec. *Cooperation in the classroom*. Edina, Minn : Interaction Book, Co., Upper Saddle River, NJ, USA, 1998.

P. Kahn and K. O'Rourke. Understanding enquiry-based learning. 2005.

N. Kali Prasad. Towards Successful Project-Based Teaching-Learning Experiences in Engineering Education. *International Journal of Engineering Education*, 29(1):17–22, 2013.

P. Kinnunen, V. Meisalo, and L. Malmi. Have we missed something?: identifying missing types of research in computing education. In *Proceedings of the Sixth international workshop on Computing education research*, ICER '10, pages 13–22, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0257-9. doi: 10.1145/1839594.1839598.

M. Koehler, P. Mishra, K. Yahya, and A. Yadav. Successful teaching with technology: The complex interplay of content, pedagogy, and technology. In *Society for Information Technology & Teacher Education International Conference*, volume 2004, pages 2347–2354, 2004.

Koli-Calling. Koli Calling international conference on computing education research. `http://cs.joensuu.fi/kolistelut/`. [Retrieved 07/09/2013].

S. Krishnaprasad. Relevance of computer hardware topics in computer science curriculum. *J. Comput. Sci. Coll.*, 18(2):328–336, Dec. 2002. ISSN 1937-4771.

E. Larraza-Mendiluze and N. Garay-Vitoria. Una propuesta de impartición de competencias asociadas a la entrada/salida del computador desde el punto de vista colaborativo. In *Actas del congreso Fomento e Innovación con Nuevas Tecnologías en la Docencia de la Ingeniería*, FINTDI 2009, pages 193–198. IEEE, Sociedad de Educación: Capítulos Español y Portugués, 2009. ISBN 978-84-8158-463-9.

E. Larraza-Mendiluze and N. Garay-Vitoria. Changing the learning process of the input/output topic using a game in a portable console. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 316–316, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-820-9. doi: 10.1145/1822090.1822193. URL `http://doi.acm.org/10.1145/1822090.1822193`.

E. Larraza-Mendiluze and N. Garay-Vitoria. A comparison between lecturers' and students' concept maps related to the input/output topic in computer architecture. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 57–66, New York, NY, USA, 2012. ACM Press. doi: 10.1145/2401796.2401803.

E. Larraza-Mendiluze and N. Garay-Vitoria. Use of concept maps to analyze students' understanding of the I/O subsystem. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 67–76, New York, NY, USA, 2013a. ACM Press. doi: 10.1145/2401796.2401803.

E. Larraza-Mendiluze and N. Garay-Vitoria. *The Learning Outcomes of the Exam Question in the Input/Output Topic in Computer Architecture*, pages 212–215. LaTiCE '13. Institute of Electrical and Electronics Engineers, IEEE, 2013b. ISBN 978-1-4673-5627-5. doi: 10.1109/LaTiCE.2013.13.

E. Larraza-Mendiluze, N. Garay-Vitoria, J. Martín, J. Muguerza, T. Ruiz-Vázquez, I. Soraluze, J. Lukas, and K. Santiago. Nintendo DS projects to learn computer input-output. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, ITiCSE '12, pages 373–373, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1246-2. doi: 10.1145/2325296.2325388.

E. Larraza-Mendiluze, N. Garay-Vitoria, J. Martín, J. Muguerza, T. Ruiz-Vázquez, I. Soraluze, J. Lukas, and K. Santiago. Game-Console-Based Projects for Learning the Computer Input/Output Subsystem. *IEEE Transactions on Education*, 56(4):453–458, Nov. 2013. ISSN 0018-9359. doi: 10.1109/TE.2013.2255877.

L. Malmi, J. Sheard, Simon, R. Bednarik, J. Helminen, A. Korhonen, N. Myller, J. Sorva, and A. Taherkhani. Characterizing research in computing education: a preliminary analysis of the literature. In *Proceedings of the Sixth international workshop on Computing education research*, ICER '10, pages 3–12, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0257-9. doi: 10.1145/1839594.1839597.

M. M. Mano. *Computer System Architecture*. Prentice Hall PTR, Englewood Cliffs, NJ, USA, 3rd edition, 1993. ISBN 0131755633.

A. Martínez-Monés, E. Gómez-Sánchez, Y. Dimitriadis, I. Jorrín-Abellán, B. Rubia-Avi, and G. Vega-Gorgojo. Multiple case studies to enhance project-based learning in a computer architecture course. *Education, IEEE Transactions on*, 48(3):482–489, 2005.

P. Marwedel and B. Sirocic. Bridges to computer architecture education. In *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, WCAE'10, page 12. ACM, 2004.

D. McLinden. Concept maps as network data: Analysis of a concept map using the methods of social network analysis. *Evaluation and Program Planning*, 36(1):40 – 48, 2013. ISSN 0149-7189. doi: 10.1016/j.evalprogplan.2012.05.001. <ce:title>Special Section: Rethinking Evaluation of Health Equity Initiatives</ce:title>.

P. Mishra and M. Koehler. Educational technology by design: Results from a survey assessing its effectiveness. In *Proceedings of Society for information Technolgy and Teacher Education International Conference*. C. Crawford, et al. (Eds.), 2005.

P. Mishra and M. Koehler. Technological pedagogical content knowledge: A framework for teacher knowledge. *The Teachers College Record*, 108(6):1017–1054, 2006.

A. Mühling and P. Hubwieser. Towards software-supported large scale assessment of knowledge development. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 145–146, New York, NY, USA, 2012. ACM Press. ISBN 978-1-4503-1795-5. doi: 10.1145/2401796.2401818.

B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *Education, IEEE Transactions on*, 52(4):449–458, 2009.

J. Novak and A. Cañas. The theory underlying concept maps and how to construct them. *Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition*, 2008. URL `http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf`. [Online; accessed 28/06/2013].

G. J. Nutt. *Operating systems*, volume 3. Pearson Education, 2004.

Pandora. Pandora official boards. URL `http://www.openpandora.org`. [Retrieved 10/18/2013].

D. Patterson and J. Hennessy. *Computer Organization and Design*. Morgan Kaufmann, 4th edition, 2009. ISBN 0123744938.

A. Pears, S. Seidman, C. Eney, P. Kinnunen, and L. Malmi. Constructing a core literature for computing education research. *SIGCSE Bull.*, 37(4):152–161, Dec. 2005. ISSN 0097-8418. doi: 10.1145/1113847.1113893.

K. Powers. Teaching computer architecture in introductory computing: why? and how? In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ACE '04, pages 255–260, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

QS. QS World University Rankings, 2013. URL `http://www.topuniversities.com/university-rankings/world-university-rankings/2013#sorting=rank+region=+country=+faculty=+stars=false+search=`. [Retrieved 09/24/2013].

U. Ramachandran and W. Leahy Jr. An integrated approach to teaching computer systems architecture. In *Proceedings of the 2007 workshop on Computer architecture education*, WCAE'07, pages 38–43, New York, NY, USA, 2007. ACM.

J. Randolph. *Computer science education research at the crossroads: a methodological review of computer science education research, 2000–2005*. PhD thesis, Logan, UT, USA, 2007. AAI3270886.

M. Saeli. Pedagogical content knowledge in programming education for secondary school. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 145–146, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0829-8. doi: 10.1145/2016911.2016943.

K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, L. Thomas, and C. Zander. Student understanding of object-oriented programming as expressed in concept maps. *ACM SIGCSE Bulletin*, 40(1):332, Feb 2008. doi: 10.1145/1352322.1352251.

E. Santhanam, C. Leach, and C. Dawson. Concept mapping: How should it be introduced, and is there evidence for long term benefit? *Higher Education*, 35(3):317–328, 1998. doi: 10.1023/A:1003028902215.

M. Santofimia and F. Moya. Nintendo DS: A Pedagogical Approach to Teach Computer Architecture. In H. R. Arabnia and A. M. G. Solo, editors, *ESA*, pages 269–273, Las Vegas, Nevada, USA, 2009. CSREA Press. ISBN 1-60132-102-3.

T. A. Scott. Illustrating programmed and interrupt driven I/O. *J. Comput. Sci. Coll.*, 16(1): 230–238, Oct. 2000a. ISSN 1937-4771.

T. A. Scott. Mano Computer Simulator, 2000b. URL `http://hopper.unco.edu/course/CS222/CS222S2000/lab7.html`. [Retrieved 09/19/2013].

G. W. Scragg. *Computer organization: a top-down approach*. McGraw-Hill, Inc., New York, NY, USA, 1992. ISBN 0-07-055843-4.

O. Seppälä. *Advances in assessment of programming skills*. PhD thesis, 2012. URL `https://aaltodoc.aalto.fi/handle/123456789/4446`. [Retrieved 07/22/2013].

Shanghai_Jiao_Tong_University. Academic Ranking of World Universities: Computer Science, 2013. URL `http://www.shanghairanking.com/SubjectCS2013.html`. [Retrieved 09/24/2013].

M. Shirali-Shahreza. Aiding speech-impaired people using nintendo ds game console. In *Proceedings of the 1st International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '08, pages 83:1–83:3, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-067-8. doi: 10.1145/1389586.1389681.

L. S. Shulman. Those who understand: Knowledge growth in teaching. *Educational researcher*, 15(2):4–14, 1986.

Simon. Ten years of the Australasian Computing Education Conference. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*, ACE '09, pages 157–164, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc. ISBN 978-1-920682-76-7.

Simon, A. Carbone, M. De Raadt, and R. Lister. Classifying computing education papers: process and results. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 161–171, 2008. ISBN 9781605582160.

J. Sorva. *Visual Program Simulation in Introductory Programming Education*. PhD thesis, 2012. URL `http://lib.tkk.fi/Diss/2012/isbn9789526046266/isbn9789526046266.pdf`. [Retrieved 07/22/2013].

W. Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall Press, Upper Saddle River, NJ, USA, 9th edition, 2012. ISBN 978-0132936330.

T. D. Stanley, L. K. Wong, D. Prigmore, J. Benson, N. Fishler, L. Fife, and D. Colton. From archi torture to architecture: Undergraduate students design and implement computers using the multimedia logic emulator. *Computer Science Education*, 17(2):141–152, 2007.

M. Stojcev, I. Milentijevic, D. Kehagias, R. Drechsler, and M. Gusev. Computer architecture core of knowledge for computer science studies. *Cyprus Computer Society Journal*, 5(4):39–42, 2003.

A. S. Tanenbaum and T. Austin. *Structured Computer Organization*. Prentice Hall Press, Upper Saddle River, NJ, USA, 6th edition, 2012. ISBN 9780132916523.

P. Teller, M. Nieto, and S. Roach. Combining learning strategies and tools in a first course in computer architecture. *Symposium on Computer Architecture*, 2003.

J. W. Thomas. A review of research on project-based learning. *San Rafael, CA: Autodesk Foundation*, 2000. URL `http://www.ri.net/middletown/mef/linksresources/documents/researchreviewPBL_070226.pdf`. [Retrieved 29/11/2013].

Thomson_Reuters. Times Higher Education World University Rankings, 2013. URL `http://www.timeshighereducation.co.uk/world-university-rankings/2012-13/subject-ranking/subject/engineering-and-IT`. [Retrieved 09/24/2013].

P. Torrone. Run Homebrew Apps on Your PlayStation Portable (PSP), 2005. URL `http://admin.makezine.com/extras/40.html`. [Retrieved 10/18/2013].

UCLM. Computer Structure, 2013. URL https://guiae.uclm.es/vistaPrevia/6389/999. [Retrieved 07/22/2013].

UGR. Computer Structure, 2013. URL http://grados.ugr.es/informaticaymatematicas/pages/infoacademica/guiasdocentes/201213/segundo/estructura_computadores/! [Retrieved 07/22/2013].

Unizar. Architecture and Computer Organization 1, 2013. URL http://titulaciones.unizar.es/asignaturas/30205/actividades13.html. [Retrieved 07/22/2013].

UPC. Computer Structure, 2013. URL http://www.fib.upc.edu/fib/estudiar-enginyeria-informatica/assignatures/EC.html. [Retrieved 07/22/2013].

UPM. Computer Architecture, 2013. URL http://www.dia.eui.upm.es/Asignatu/Curso12-13/ICarq_com.htm#programa. [Retrieved 07/22/2013].

UPV. Computer Structure, 2013. URL http://www.upv.es/pls/oalu/sic_asi.Busca_Asi?p_codi=11552&p_caca=2012&P_IDIOMA=c&p_vista=. [Retrieved 07/22/2013].

UPV/EHU. Computer Structure, 2013. URL http://www.ikasketak.ehu.es/p266-shmastct/eu/pls/entrada/plew0040.htm_asignatura_next?p_sesion=&p_cod_idioma=CAS&p_en_portal=S&p_cod_centro=226&p_cod_plan=GINFOR20&p_anyoAcad=act&p_pestanya=3&p_menu=guia&p_cod_asig=26015&p_ciclo=X&p_curso=1&p_dpto=&p_vengo_de=asig_cursos&p_centro_ori=226&p_plan_ori=GINFOR20. [Retrieved 07/22/2013].

T. Urness. Teaching computer organization/architecture by building a computer. In *Proceedings of the 2007 workshop on Computer architecture education*, WCAE'07, pages 72–76. ACM, 2007. ISBN 9781595937971.

D. Valentine. Cs educational research: a meta-analysis of sigcse technical symposium proceedings. *SIGCSE Bull.*, 36(1):255–259, Mar. 2004. ISSN 0097-8418. doi: 10.1145/1028174.971391.

A. Vlado. Networks / Pajek: Program for Large Network Analysis, 2005. URL http://vlado.fmf.uni-lj.si/pub/networks/pajek/. [Retrieved 07/03/2013].

WCAE. Workshop on Computer Architecture Education. http://www4.ncsu.edu/~efg/wcaes.html. [Retrieved 07/11/2013].

W. Wolf and J. Madsen. Embedded Systems Education for the Future. *Proceedings of the IEEE*, 88(1), 2000. ISSN 0018-9219. doi: 10.1109/5.811598.

G. S. Wolffe, W. Yurcik, H. Osborne, and M. A. Holliday. Teaching computer organization/architecture with limited resources using simulators. *SIGCSE Bull.*, 34(1):176–180, Feb. 2002. ISSN 0097-8418. doi: 10.1145/563517.563408.

P. Wouters, C. van Nimwegen, H. van Oostendorp, and E. D. van der Spek. A meta-analysis of the cognitive and motivational effects of serious games. *Journal of Educational Psychology*, 105(2):249, 2013.

B. Wu and A. I. Wang. A guideline for game development-based learning: A literature review. *Int. J. Comput. Games Technol.*, 2012:8:8–8:8, Jan. 2012. ISSN 1687-7047. doi: 10.1155/2012/103710. URL `http://dx.doi.org/10.1155/2012/103710`.

C. Yehezkel, M. Ben-Ari, and T. Dreyfus. Computer architecture and mental models. *SIGCSE Bull.*, 37(1):101–105, Feb. 2005. ISSN 0097-8418. doi: 10.1145/1047124.1047390. URL `http://doi.acm.org/10.1145/1047124.1047390`.

C. Yehezkel, M. Ben-Ari, and T. Dreyfus. The contribution of visualization to learning computer architecture. *Computer Science Education*, 17(2):117–127, June 2007. ISSN 0899-3408. doi: 10.1080/08993400601165545.

Appendices

APPENDIX A

Notes on the theory of the computer I/O subsystem for the students

This appendix shows the lecture notes on the theory of the computer I/O subsystem, that students are handed out. The appendix is in Basque, one of the languages used in the course.

A.1 Sarrera/Irteera-ko interfazearen deskribapena.

Gai honetan, CPU edo prozesadorea kanpoaldearekiko nola komunikatzen den aztertuko dugu. Era berean, komunikazio hori makina lengoaia mailan nola kontrolatzen den ikusiko dugu. Komunikazio hori lortu ahal izateko ***periferikoen*** laguntza behar du prozesadoreak.

Periferikoak mota askotakoak izan daitezke:

- Datuak aurkezteko: pantaila, inprimagailua...
- Datuak hartzeko: teklatua, sentsoreak,...
- Informazioa metatzeko: diskoak, zintak,..
- Ingurunea aldatzeko: motoreak, balbulak, ...

Periferikoak elektronikoki oso era desberdinetan kontrolatzen dira. Honengatik CPU-k zuzenean kontrolatu beharko balitu, kontrol-programak eta periferikoen konexioak hiru busekin konplexuegiak izango lirateke. Beraz, bitarteko zirkuitu berezi bat sartzen da: ***Sarrera/Irteerako kontrolagailua***. Sistemak A.1 Irudikoaren itxura hartuko du.

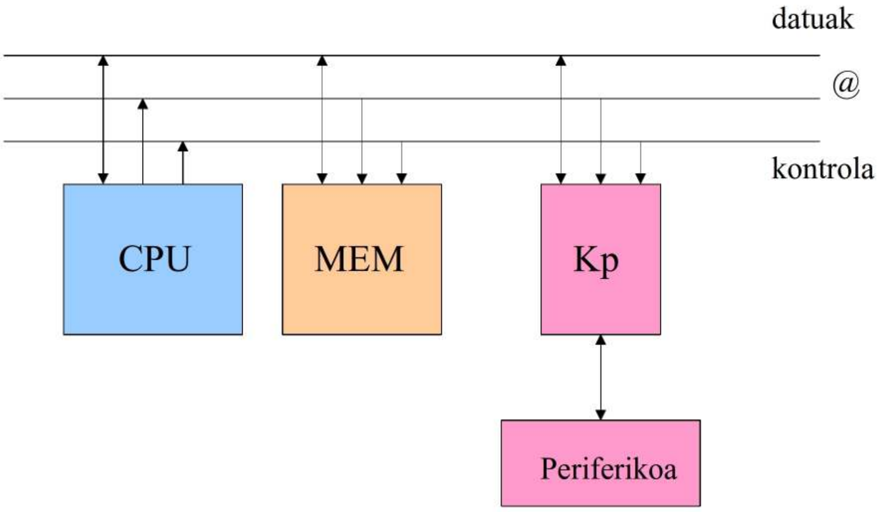


Figure A.1: Sistemaren itxura periferikoaren kontrolagailua txertatu ondoren.

Kontrolagailua periferiko eta sistemako busen artean konektatzen den gailu elektronikoa baino ez da eta makina-mailan erregistro-multzo bat bezala ikusten du programatzaileak. Prozesadorea erregistro horien bitartez komunikatzen da periferikoarekin.

A.1.1 S/I-ko kontrolagailuaren ikuspegi funtzionala

Hauexek dira S/I-ko kontrolagailuaaren betebeharrak:

- CPU-rekin komunikazioa (sarrera/irteerako eragiketak burutzeko CPU-k egindako eskaerak hartzen ditu).
- Periferikoaren kontrola, honek CPU-k eskatutakoa bete dezan.

- Periferiko eta CPU-ren arteko informazioaren transferentzia erraztea.

Eskematikoki kontrolagailua zer den A.2 Irudian ikus daiteke.

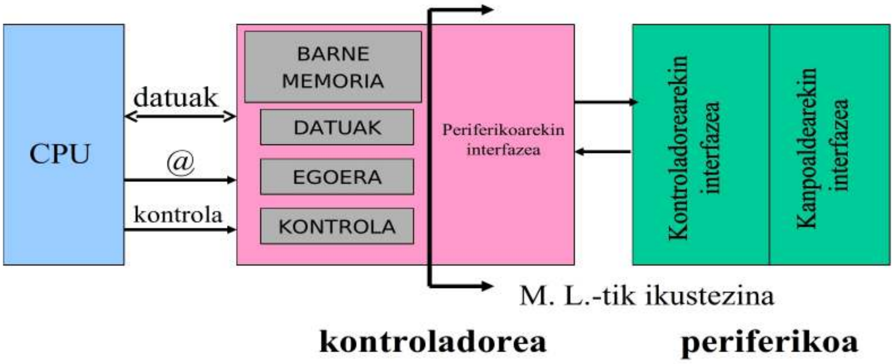


Figure A.2: Sistemaren itxura periferikoaren kontrolagailua txertatu ondoren.

A.1.2 Sarrera/irteera-ko kontrolagailuaren erregistroak

Kontrolagailuak dituen erregistroak prozesadoreak makina-lengoaiaren aginduen bidez atzitzen ditu, helburu orokorreko erregistroak edota memoria-posizioak atzitzen dituen era berean. Kontrolagailuaren erregistroak hauexek izan ohi dira:

Egoera-Erregistroa:

Periferikoaren egoerari buruzko informazioa gordetzen du. Adibidez, tekla bat sakatu den edo informazio konkretu bat inprimatu den. CPU-k informazio hori behar duenean erregistro honen edukia **irakurtzen** du.

Kontrol-Erregistroa:

Erregistro honetan CPU-k **idatzi** egiten du ekintza konkretu bat aurrera eraman ahal izateko kontrolagailuak behar duen informazioa; adibidez, nola egin behar duen lana periferikoak edo zein den burutu behar den eragiketa.

Datu-Erregistroa:

Kontrolagailuak periferikotik jasotako informazioa hemen metatzen du CPU-k hortik har dezan edo alderantziz. Adibidez, tekleatutako karakterea edo diskoan idatzi nahi den karakterea.

Bi aukera desberdin daude prozesadoreak kontrolagailuaren erregistroak atzi ditzan, erregistro hauen “kokapenaren” arabera:

- Erregistroak memorian mapeatuta.
- Erregistroak memorian ez-mapeatuta edo S/I bereiztua.

A.1.3 Sarrera/Irteera memorian mapeatuta

Kontrolagailuaren erregistro bakoitzari memoria-mapako helbide fisiko bat dagokio, memoria fisikoaren posizioekin gertatzen den bezala. Hau da, kontrolagailuaren erregistroak memoria-posizioak bezalaxe irakurri eta idazten dira, A.3 irudian ikus daitekeen moduan.

Ikus dezagun adibide bat non, 2^6 hitzeko RAM memoria bat eta 3 erre-gistrodun teklatua konektatu nahi ditugun makina batean. Helbide fisikoak honakoak izango dira:

```
@ datu-erregistroa       = 1XXXX11 (irakurketa)
@ kontrol-erregistroa    = 1XXXX11 (idazketa)
@ egoera-erregistroa     = 1XXXX10
@ RAM                    = 0XXXXXX
```

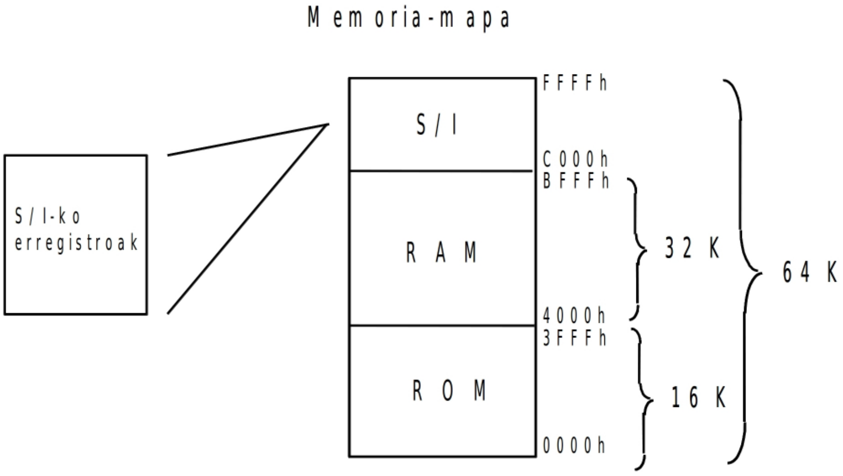


Figure A.3: Memorian mapeatutako S/I erakusten duen memoria-mapa.

Makina honen memoriako helbideak 7 bitekoak dira, beraz makinaren helbide-ratze-espazioa 2^7 posiziokoa da. Zeroz hasten den memoria-helbide bat atzitzen denean (nahiz irakurtzeko nahiz idazteko) RAM memoria atzitzen da (beraz, helbideratzeko 7 bit erabili arren, pisu altuenekoa beti 0 izango da eta horregatik memoriaren tamaina $2^6 = 64$ hitzetakoa da). 1XXXX01 helbidean egindako irakurketa batean datu-erregistroa irakurtzen da. 1XXXX11 helbidean idazten bada, orduan kontrol-erregistroan idazten da. 1XXXX10 helbidean irakurtzen denean kontrolagailuaren egoera-erregistroa irakurtzen da.

Diseinua egin den moduan, aurreikusi da teklatua sarrerako periferikoa dela eta inoiz ez dugula datu-erregistroan, hau da 1XXXX11 helbidean idatziko. Era berean kontrol-erregistroan burutuko den eragiketa irakurketa izango denez, posiblea da bi erregistro hauek helbide berean mapeatzea. Beste aukera bat izango litzateke bi erregistro horiek helbide desberdinetan egotea, horrela bietan idatzi eta irakurri ahal izango litzateke. Helbideak honela geratuko lirateke:

- datu-erregistroa 1XXXX01 helbidean mapeatuta,
- kontrol-erregistroa 1XXXX11helbidean mapeatuta,

- egoera-erregistroa 1XXXX10 helbidean mapeatuta.

A.4 Irudian, erregistroen konexioa nola gauzatzen den ikus dezakegu.

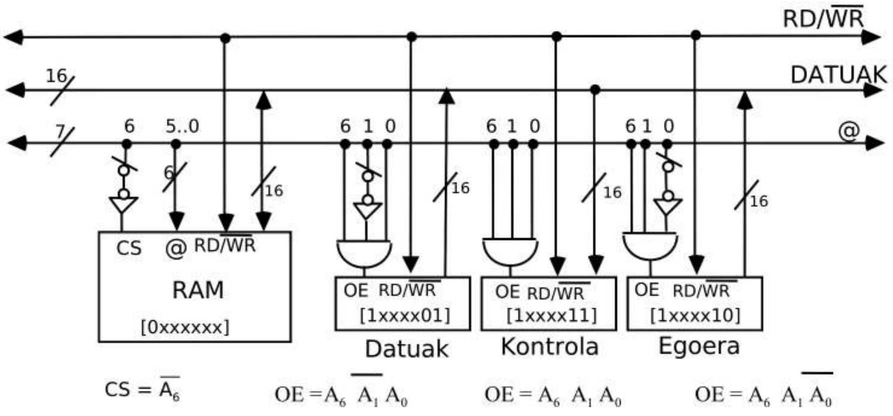


Figure A.4: Erregistroen konexioaren eskema memorian mapeatuta daudenean

Ondoren, S/Iko erregistroak memorian mapeatzearen abantaila eta desabantailak aurkeztuko ditugu:

Abantailak:

S/I-ko erregistroak memoria posizioak bezalaxe atzitzen dira, hots, edozein agindu eta edozein helbideratze-modu erabil daitezke. Izan ere, memorian mapeatutako erregistroak memoria fisikoaren posizioak bailira maneiatzen dira eta CPUa ez da enteratzen erregistro ala memoria-posizioa den.

Desabantailak:

Espazioa galtzen da memoria fisikorako.

Oso zaila da erabiltzaileak S/Iko erregistroak atzitu ditzan galaraztea. Eta normalean Sistema-Eragilea (SE) da horretaz arduratu behar dena. Nolabait erabiltzailearen atzipena galarazi behar da ezjakintasunagatik zerbait hondatu dezakeelako. Kasu honetan hardware bitartez detektatu behar da programa zein motakoa den: erabiltzaile arrunt batena edo SEarena eta lehenengo motakoa bada transferentzia galarazi. Normalean prozesadoreek seinale berezi bat izaten dute kontrol-busean exekutatzen ari den programa mota adierazteko: erabiltzaile arrunta edo berezia. Hori horrela bada aginduak beharrezkoak dira mota hori aldatu ahal izateko eta egoera-hitzean (PSW) gorde beharko da exekutatzen ari den programa mota.

A.1.4 Sarrera/Irteera bereiztua edo memorian ez mapeatua

Prozesadoreak sortzen duen helbidea memoria-posizio bati edo S/I-ko erregistroetariko bati dagokion adierazten duen seinale berezi bat dago. Seinale hau IO/M da *(InputOutput/Memory)*. A.5 irudian ikus daitekeen moduan, seinale honi esker, memoria-mapa bat eta S/I-ko mapa bat desberdindu daitezke.

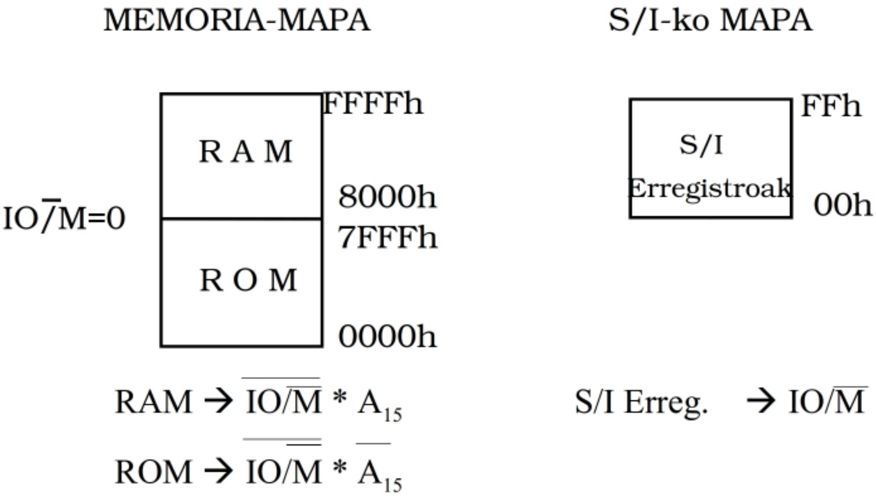

MEMORIA-MAPA
S/I-ko MAPA
FFFFh
RAM
8000h
7FFFh
ROM
0000h
$IO\overline{/}M=0$
S/I
Erregistroak
FFh
00h
RAM → $\overline{IO/\overline{M}} * A_{15}$
ROM → $\overline{IO/\overline{M}} * \overline{A_{15}}$
S/I Erreg. → $IO/\overline{M}$

Figure A.5: S/I bereiztua duen sistema baten memoria-mapa

Memoria atzitzen duen agindu bat exekutatzen denean, MOV, ADD eta INC bezalakoak, IO/M seinalea desaktibatuta dago (IO/M = 0); era honetan, prozesadoreak sortutako helbidea memoria fisikoari dagokiola jakin daiteke.

Programak S/Iko portuak (erregistroak) atzitu nahi baditu, agindu berezi bat exekutatu behar du. Sarrera/irteerako agindu bereziak hauexek izan ohi dira:

```
IN     barne_erregistro_kodea, portu_kodea
OUT    portu_kodea, barne_erregistro_kodea
```

Bi agindu hauek exekutatzen direnean IO/M seinalea aktibatzen da (IO/M = 1). Bi parametro dituzte: alde batetik, prozesadorearen erregistro bat eta beste aldetik, S/Iko portu edo erregistro bat. Agindua IN edo OUT izatearen arabera, portu horretan irakurri edo idatzi egiten da. Laburbilduz, aurreko teknikarekiko desberdintasuna hauxe da: bi espazio desberdin daude, memoria-mapa eta sarrera/irteerako mapa, eta hauek atzitzeko agindu desberdinak erabiltzen dira.

Lehen erakutsitako adibidearekin jarraiki, orain, helbide-espazioko 7 bitak erabil daitezke memoria bera helbideratzeko, beraz, memoriaren tamaina 2^7 posiziokoa izan daiteke. IO/M seinaleak 0 balioa hartu beharko du RAM memoria atzitzeko eta 1 balioa sarrera/irteera erregistroak atzitzeko, A.6 irudian ikus daitekeen bezala.

Jarraian, lehen egin den moduan aukera honen abantaila eta desabantaila nagusiak aipatuko dira:

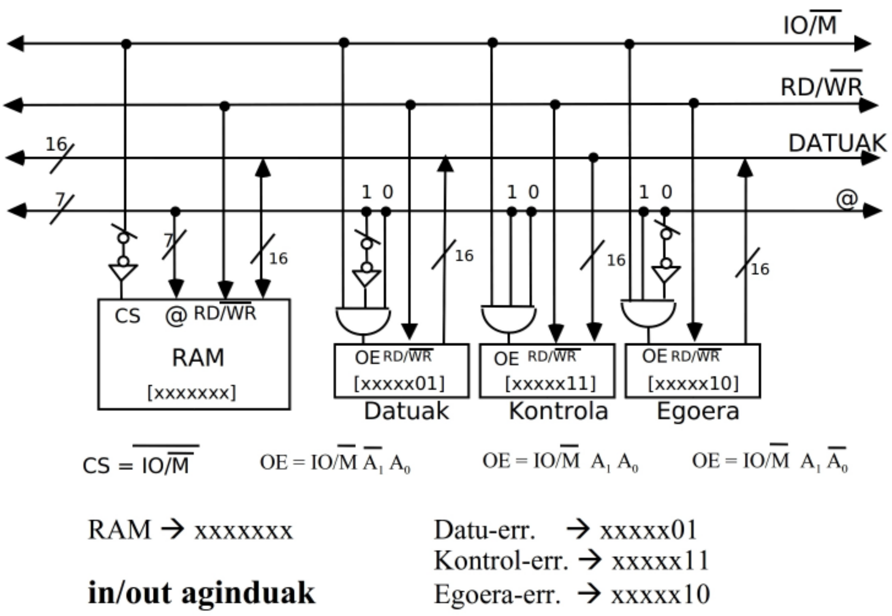


Figure A.6: Erregistroen konexioaren eskema hauek ez daudenean memorian mapeatuta

Abantailak:

Ez da memoria fisikoaren espazioa galtzen bi espazioak bereiztuta daudelako.

Erraza da programatzaileak erregistroak atzitu ditzan galaraztea, IN eta OUT aginduak SEak bakarrik erabiltzen baditu.

Desabantailak:

2 agindu gehiago behar dira agindu-multzoan eta kontrol-seinale bat gehiago (IO/M). Hauek direla eta, CPU-ren diseinua pixka bat konplexuagoa suertatzen da.

Zenbait makinetan bi metodoak nahasten dira: adibidez, i8086-an oinarritutako ordenadore pertsonaletan pantaila memorian mapeatuta dago baina teklatua, inprimagailua, eta bestelako periferikoak ez. Hau da, pantailan karaktere bat idazteko nahikoa da memoria-posizio batean idaztea (pantailako puntu edo "pixel" bakoitzari memoria-espazioko posizio bat dagokio), baina beste periferikoen erregistroak atzitzeko IN eta OUT aginduak erabili behar dira.

A.1.5 S/Iko kontrolagailuen sailkapena

Kontrolagailuak honelaxe sailka daitezke:

Ez-multiplexatuak: Kontrolagailuak periferiko bakar bat hartzen du bere gain, A.7 Irudian ikus daitekeen bezala.

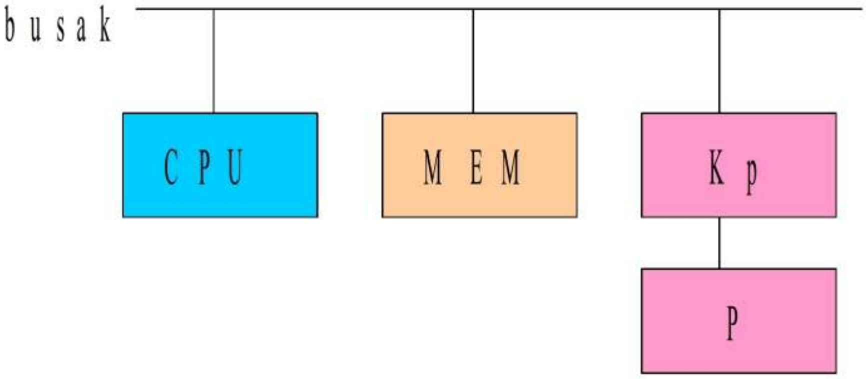


Figure A.7: Kontrolagailu ez-multiplexatua erakusten duen eskema

Hau periferikoa azkarra denean erabiltzen da, honela kontrolagailua nahiko lanpetuta mantenduko delarik. Adibidez: disko azkarrak, pantaila grafikoak,...

Multiplexatuak: Kontrolagailu batek periferiko batzuk hartzen ditu bere gain (A.8 Irudia).

Kasu honetan multiplexazioa karakterez edo blokez egin daiteke:

- Karakterez multiplexatua: Kontrolagailuak periferiko bati kasu egiten dio karaktere baten transferentziak dirauen bitartean. Periferiko motelekin erabiltzen da. Adibidez, oso normala da kontrolagailu bakar batek teklatua eta pantaila, biak batera, kontrolatzea, teklatuak atentzioa noizbehinka baino eskatzen ez baitu, tekla bat (karaktere bat) sakatzen denean, alegia.

- Blokez multiplexatua: Kontrolagailuak periferiko bati kasu egiten dio karaktere-bloke baten transferentziak dirauen bitartean. Periferiko azkarragoekin erabiltzen da. Adibidez, kontrolagailu bat eta zenbait zinta-unitate.

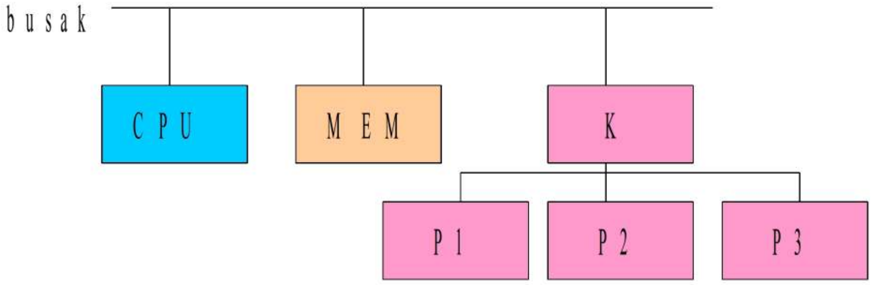


Figure A.8: Kontrolagailu multiplexatua erakusten duen eskema

A.2 Komunikazioa eta sinkronizazioa Sarrera/Irteerako eragiketetan

Konputagailu-sitema batetan S/Iko eragiketak nola burutzen diren ongi ulertzeko ondoko hiru galderak erantzun behar ditugu:

- Nola burutzen da informazioaren transferentzia?
- Nork burutzen du transferentzia?
- Noiz burutzen da transferentzia?

Nola burutzen da informazioaren transferentzia?

Hau erantzun ahal izateko zein periferiko erabiltzen ari den jakitea guztiz beharrezkoa da, informazio-transferentzia burutzeko jarraitu behar diren urratsak periferikoaren menpekoak baitira.

Adibidez, i8086 makinan oinarritutako konputagailu batetik inprimagailu batetara datu transferentzia nola egiten den ikusiko dugu. Inprimagailuaren kontrolagailuak memorian ez mapeatutako 3 erregistro ditu. Erregistro hauen helbideak honakoak izan daitezke (hau inprimagailuarekin bateragarria den konputagailuaren fabrikatzailearen araberakoa izan daiteke):

Erregistroa	**Helbidea**
datuak	378H
egoera	379H
kontrol	37AH

Erregistro hauek 8 bitekoak dira. Egoera-erregistroko 7. bita 1ekoa izateak adierazten du inprimagailua inprimatzeko prest dagoela. Egoera horretan, karaktere bat inprimatu nahi izanez gero, karaktere horren ASCII-kodea datu-erregistroan idatzi eta STROBE sekuentzia bat egin behar da. STROBE sekuentziak egiten duena zera da: kontrol-erregistroaren n bitean 1ekoa idatzi ondoren, bit beraren gainean 0koa idazten da. Eragiketa hau beharrezkoa da konputagailuak datua onar dezan. Edozein modutan, kontrolagailuak jasotako informazio hau ez da inprimatuko, <CR> eta <LF> karaktereak bidali arte. Bitartean, kontrolagailuak barne buffer batean gordeko du.

Erabat desberdina den beste kontrolagailu baten adibidea, pantailaren kontrolagailuarena da. Kasu honetan, pantailaren posizio bakoitzari (80x25 karaktereentzako lekua du testuzko pantaila arrunt batek) hitz bat dagokio bufferrean. Hitz horretan, posizio horretan erakutsiko den karakterearen ASCII-kodea eta karakterea nola erakutsiko den adierazten duen atributua gordetzen dira. Atributua adierazteko bytean balio desberdinak sar daitezke. Adibidez:

07H	normala
70H	alderantzikatua
87H	aldizkakoa

Pantailaren kontrolagailu hau memorian mapeatuta egoten da, 0B0000H edo 0B8000H helbidetik aurrera (txartel-grafikoaren arabera).

Pantaila horren (i,j) posizioan karaktere bat erakutsi nahi badugu ondorengo posizioetan karakterearen ASCII-kodea eta atributua idatzi beharko dira:

```
(i,j) karakterearen @ = oinarri @ + 2(80i+j)
(i,j) atributuaren @ = oinarri @ + 2(80i+j) + 1
```

Pantailan esaldi bat idazteko, nahikoa litzateke karaktere eta atributu bakoitza dagokion memoriako posizioan idaztea.

Noiz burutzen da transferentzia?

Galdera honi erantzuna eman ahal izateko hauxe da jakin behar dena: nola sinkronizatzen da CPU S/Iko gailuekin? Hau da, nola daki prozesadoreak S/Iko eragiketa berri bat has daitekeela?

Kasu batzuetan erantzuna oso erraza da:CPU-k transferentzia berri bati hasiera ematen dio berak nahi duenean. Adibidez: pantailan karaktere berri bat aurkezteko ez du zertan egoera berezi baten zain egon beharrik, nahi duenean karaktere berria memoria-posizio batean uzten du. Beste kasu limitea hauxe da: CPU-k erabiltzaileak tekleatutako karaktere bat irakurri behar duenean. Kasu honetan CPU-k erabiltzaileak tekla sakatu arte itxaron behar du.

Kasurik orokorrenean, CPU-k gailua transferentzia burutzeko prest dagoen detektatu beharko du nola edo hala. Detekzio hau bi eratan egin daiteke. Hau da, badaude bi metodo CPU eta S/I-ko gailuak sinkronizatzeko:

- **Inkesta bidezko sinkronizazioa:** CPU-k gailua prest dagoen detektatzeko galdetu egiten dio inkesta moduko batez, bere erregistroak aztertuz.
- **Etenen bidezko sinkronizazioa:** Kasu honetan kontrolagailua da CPU-ri periferikoa prest dagoela esaten diona. Hau egiteko **etena** izeneko seinale bat aktibatzen du.

Nork burutzen du transferentzia?

Nork burutzen du kontrolagailu eta memoriaren arteko informazio-transferentzia? (orokorrean periferiko batetik etorritako informazioa memorian metatzen da eta alderantziz).

Bi aukera daude:

a) CPU-k S/Iko ekintza guztiak burutzen ditu.

b) CPU-k ekintza batzuk baino ez ditu burutzen, beste guztiak kanpo zirkuitu berezi batek burutzen dituelarik. Bi zirkuitu espezializatu mota daude:

  - DMA kontrolagailua (memoriarako atzipen zuzena).
  - S/Iko prozesadorea (kanala).

A.2.1 Inkesta bidezko Sarrera/irteera

Kasu honetan, S/I-ko transferentzia prozesadoreak kontrolatzen du; hau da, prozesadoreak ematen dio hasiera transferentziari periferikoa prest dagoela detektatu ondoren. Hau egoera-erregistroaren bitartez jakiten da: egoeraren inkesta edo egoeraren azterketa.

Egoera bidezko bi inkesta mota daude:

Etengabeko inkesta (edo inkesta jarraitua): S/Iko eragiketa bat burutzeko programa nagusiak exekutatu behar duen algoritmoa hauxe da:

```
 irakurri egoera-erregistroa
bitartean ez-prest egin
    irakurri egoera-erregistroa
ambitartean
burutu S/I-ko transferentzia
```

Prozesadoreak inkesta besterik ez du egiten gailu edo periferikoa prest egon arte.

Inkesta periodikoa (edo aldizkako inkesta): Egoera-erregistroa irakurtzen denean periferikoa ez badago prest, orduan programa nagusiak beste eragiketa edo kalkulu batzuk burutu ditzake eta hauek bukatzean berriro aztertu egoera-erregistroa periferikoa prest dagoen ikusteko. Programa nagusian exekutatu behar den algoritmoa:

```
 irakurri egoera-erregistroa
bitartean ez-prest egin
    burutu beste eragiketa batzuk
     /*edo itxaron denbora-tarte bat */
    irakurri egoera-erregistroa
ambitartean
burutu S/I-ko transferentzia
```

Kasu honetan arazo bat sor daiteke: programa nagusiak beranduegi detektatzea transferentzia burutu daitekeela. Adibidez, programa itxaroten ari den bitartean edo beste kalkuluak egiten ari diren bitartean, erabiltzaileak bi tekla sakatzen ditu eta lehenengoa galdu egiten da.

Egoera bidezko inkestaz gain beste aukera bat ere badago inkesta burutzeko: **inkesta denborizatua**. Honetarako periferikoak transferentzia bat burutzeko zenbat denbora behar duen jakin behar da.

Denbora-tarte hau igaro ondoren suposatzen da periferikoa prest dagoela beste transferentzia berri bat burutzeko. Kasu honetan, beraz, ez da beharrezkoa kontrolagailuak egoera-erregistro bat edukitzea (honetarako baino erabiltzen ez bada, behintzat). Garbi dago teknika hau ezin dela erabili erantzun-denbora ezezaguna duten periferikoekin. Adibidez, teklatua: erabiltzailea da bi tekla-sakatze kontsekutiboren arteko denbora finkatzen duena eta hau ez da konstantea!

A.2.2 Etenen bidezko Sarrera/Irteera

Etena, prozesadorearen aditasuna behar duen ez-ohiko gertaera bat baino ez da. Gertaera hauek honela sailkatzen dira:

Sinkronoak (Softwarezkoak): Etena agindu baten exekuzioaren ondorioa da. Honengatik programaren fluxuarekiko sinkronoa da. Mota batzuk daude:

- **TRAP-ak:** Eten-mota hau sortzen duen agindua oso-osorik exekutatzen da eta ondoren prozesadoreak etenaren zerbitzu-azpirrutina exekutatuko du. Adibidez, koma higikorreko eragiketa batean OVF sortzen denean, edo zerorekin zatiketa egiten denean, eta abar.

- Agindua ez da osorik exekutatzen baina prozesadorea egoera ezagun batetan gelditzen da. Honela, hutsa ebatzi ondoren, etena sortu duen aginduaren exekuzioa hasten da berriro. Adibidez: paritate-errorea memoria-irakurketa batetan.

- Agindua ez da osorik exekutatzen eta prozesadorea egoera ezezagun batean gelditzen da. Honela guztiz ezinezkoa da programa berreskuratzea. Adibidez, memoria-posizio pribilegiatu bat atzitu nahi denean.

Asinkronoak (Hardwarezkoak): Eten hauek programaren exekuzioaren fluxuarekiko guztiz independenteak dira, prozesadoretik at gertatutako ekintza baten ondorioak direlarik. Bi mota:

- **HARDWAREAREN HUTSAK:** Tentsio-erorketa, bus-apurketa,...

- **SARRERA/IRTEERAKOAK:** Sarrera/irteerako gailuek sortzen dituzte eten hauek. Hauek dira guri interesatzen zaizkigunak.

Prozesadoreak eten-sarrera batzuk izaten ditu periferikoen eten-eskaerak jasotzeko. Periferikoa transferentzia bat burutzeko prest dagoenean, bere kontrolagailuaren egoera-erregistroko bit bat aktibatzen da (inkesta bidezko sinkronizazioan aztertzen dena hain zuzen); une berean INT irteera **(eten-eskaera)** aktibatzen da, irteera hau CPU-ren sarreretariko batekin konektatuta dagoelarik. Hardwarearen eskema A.9 irudikoa da.

CPU-k sarrera hauetariko bat aktibatu dela detektatzen duenean (eten-eskaera) eten hori momentu horretan bertan zerbitzatuko duen ala ez erabakitzen du (etena galarazita ala baimenduta dagoen aztertu behar da edo une horretan burutzen ari den lana S/I-ko eragiketa baino garrantzitsuagoa den). Baiezkoan programaren exekuzioa gelditzen da eta **etenaren zerbitzu-azpirrutinara** jauzten da. Etenaren tratamendua bukatzen denean prozesadoreak etendako programaren exekuzioa berreskuratzen du.

Beraz, etenaren zerbitzu-errutina kanpo gertaera batek aktibatutako errutina baino ez da. Errutina honek prozesadorea eta periferikoaren arteko informazio-transferentzia burutzen du eta S/I-ko eragiketa eskatu zuen programari transferentziaren emaitzari buruzko informazioa pasatzen dio. Guzti honen eskema A.9 irudian ikus daiteke.

Lehentxeago aipatu denez, metodo honen abantaila hauxe da: prozesadoreak ez du denborarik galtzen inkesta egiten eta beste ekintza batzuk burutu ditzake. Desabantaila, makinaren hardwarea pixka bat konplexuagoa dela INT kontrol-unitatearen sarrera-berria dela eta, bere egoera algoritmoa konplexuagoa izanik.

A.3 Etenen kudeaketa

Eten bat sortzen denean burutzen diren ekintzak hauexek dira:

- Eten-eskaeraren detekzioa.
- Gorde etendako programaren egoera.

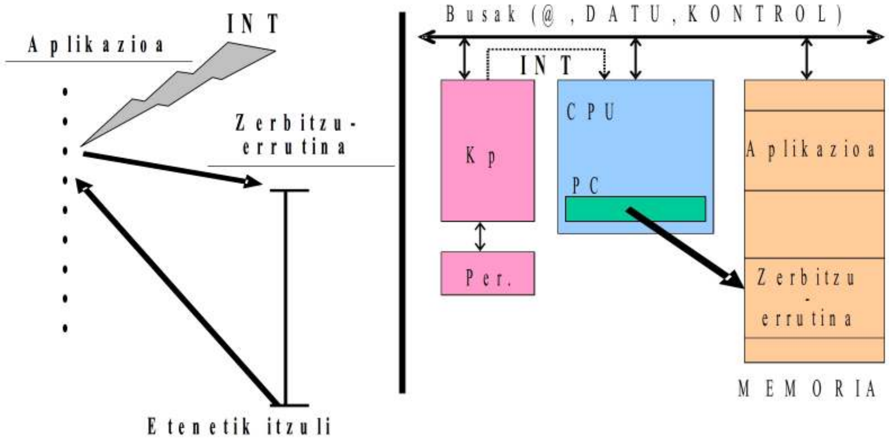


Figure A.9: Etenei erantzuteko mekanismoaren eskema

- Exekutatu behar den zerbitzu-errutinaren identifikazioa (periferikoaren menpekoa).
- Zerbitzu-errutinaren exekuzioa.
- Etendako programaren egoera berreskuratu.

Azter ditzagun orain banan banan.

A.3.1 Eten-eskaeraren detekzioa

Eten-eskaera CPU-ren sarrera baten aktibazioa da. CPU-k sarrera bat edo gehiago izan dezake. Normalean etenak sor ditzaketen periferikoen kopurua CPU-ren sarreren kopurua baino handiagoa izan ohi da. Honengatik periferikoak sarrera horien artean banatu behar dira.

CPU-k sarrera hori aztertu beharko du periodikoki. Normalean agindu baten exekuzioa bukatzen denean aztertzen du, hurrengo aginduaren bilaketari ekin baino lehen. Aginduak luzeak direnean, karaktere-kateen tratamenduak kasu, karaktere baten gaineko eragiketa burutu ondoren aztertu ohi da. Normalean eten-eskaerek berehala behar dute aditasuna.

Honengatik ez da oso egokia eten-seinalearen azterketa luzeegia izan daitekeen agindu baten exekuzioaren ondoren egitea.

Oso normala da CPU-k etenen kontra babesteko baliabideren bat izatea, adibidez: oso programa garrantzitsua exekutatzen ari denean. Orokorrean egoera-hitzaren bit batzuk etenak baimenduta ala galarazita dauden adierazi ohi dute. Eten-seinale bakarraren kasuan, hardwarea nolakoa izan daitekeen erakusten duen eskema ikus daiteke A.10 irudian.

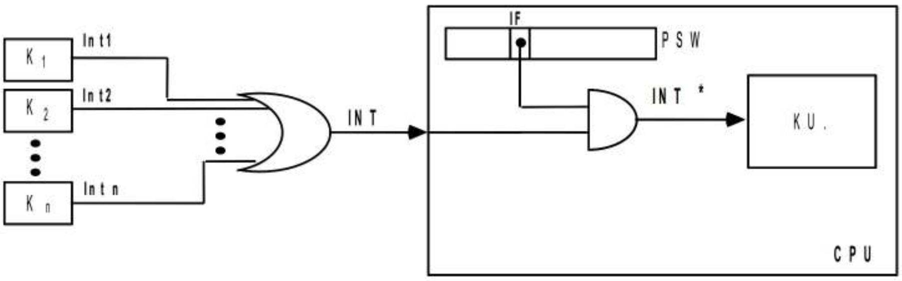


Figure A.10: Eten-seinale bakarra dagoen kasurako hardware eskema

IF etenaren adierazlea (Interrupt Flag) da. Kontrol-unitateak aztertzen duen seinalea INT* da, periferikoetatik heltzen den INT eten-eskaeraren seinalea "maskaratuz" lortua. IF=0 denean etenak galarazita (disabled) daudela esaten da. IF=1 denean, aldiz, etenak baimenduta daude (enabled). Eten-eskaera ez-maskaragarriak existitu ohi dira (NMI = Non Maskable Interrupt).

Eten-seinale batzuk daudenean posible da bakar batzuk baino ez galaraztea, A.11 irudian ikus daitekeen bezala.

Makina-lengoaietan IF bita eta maskara maneiatzeko agindu bereziak existitu ohi dira; Horrela, programatzaileak uneoro hainbat etenetaz babesteko aukera izango du.

Suposatuko dugu, etenaren zerbitzu-errutinaren exekuzioa hasten denetik amaitu arte beste eten-eskaerak galarazita daudela. Hau da, CPU-k eten bat zerbitzatuko duela erabakitzen duenean, IF bita automatikoki 0ra jarriko du eta ez da berriro aktibatuko zerbitzu-errutina amaitu arte.

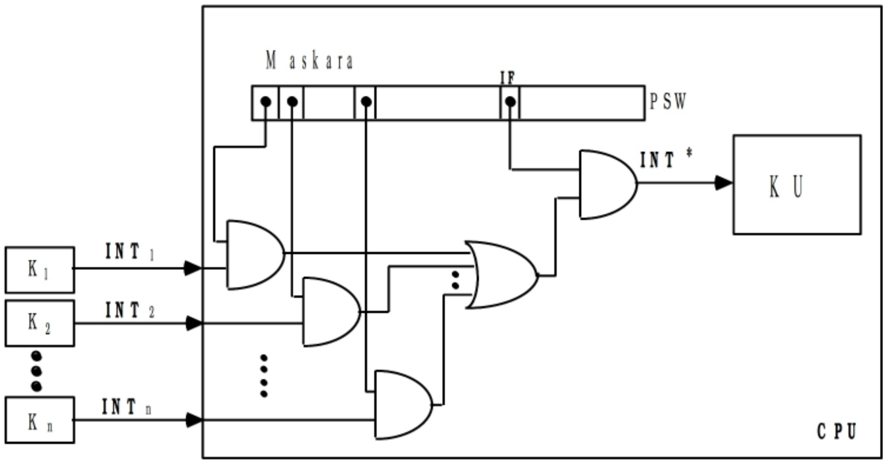


Figure A.11: Eten-seinale bat baino gehiago dagoen kasurako hardware eskema

A.3.2 Gorde etendako programaren egoera

Teorikoki etenaren zerbitzu-errutinaren exekuzioa amaitzen denean etendako programaren exekuzioa berreskuratu behar da. Hau egin ahal izateko beharrezkoa da programaren egoera gordetzea etena onartzen denean.

Automatikoki gorde behar den gutxienezko informazioa hauxe da:

- PC: zerbitzu-errutinatik bueltatzerakoan exekutatu behar den hurrengo agindua zein den jakiteko.
- PSW (baldintza-bitak edo egoera-bitak): zerbitzu-errutinatik bueltatzerakoan, hau exekutatzen hasi aurretik zegoen egoera berean jarraitu dezan.

Zerbitzu-errutinak, bere aldetik, berak erabiltzen dituen erregistroen edukia gorde beharko du. Zerbitzu-errutina ezin dela eten esan dugunez (maila bakarreko etenak), prozesadorearen egoera posizio finkotan gorde daiteke. Geroxeago ikusiko dugu prozesadore gehienetan maila anitzeko etenak onartzen direla, eta horrexegatik prozesadorearen egoera pilan gorde beharko dela.

A.3.3 Zerbitzu-errutina edo periferikoaren identifikazioa

Etenak sor ditzaketen periferikoak desberdinak direnez eta bakoitzak tratamendu egokitua behar duenez, etena detektatu (eta onartu) ondoren beharrezkoa da periferiko eten-sortzailea identifikatzea berari dagokion zerbitzu-errutinara jauzteko. Arazo hau agertzen da periferiko batzuk eten-sarrera bakarra erabiltzen dutenean. Arazo hau ebazteko ondoko galderak egin behar dizkiogu gure buruari:

- nork sortu du etena? Identifikazioa hardware bidez zein software bidez egin daiteke.
- etena periferiko batek baino gehiagok eskatzen badu une berean, zeini egingo zaio kasu lehenik? Lehentasunak ezarri behar dira.

Software bidezko identifikazioa:

Etenak baimenduta daudela, INT seinalea aktibatzen den bakoitzean, CPU zerbitzu-errutina konkretu batetara jauzten da (zerbitzu-errutina bana eten-sarrerarako): zerbitzu-errutina orokorra. Errutina honen lehenengo aginduen helburua hauxe da: eten-eskaera sortu duen periferikoaren identifikazioa egitea, inkesta baten bidez, kontrolagailuen egoera-erregistroak aztertuz. Inkestaren ordenak periferikoen lehentasuna adierazten du. Periferikoa identifikatua izan denean, berari dagokion zerbitzu-errutinara jauzten da.

Ebazpen hau oso sinplea da baina motelegia izan daiteke. Hau erabiltzen duen makina bat: Nintendo DS.

Hardware bidezko identifikazioa:

Kasu honetan periferikoa bera identifikatzen da prozesadorearen aurrean. Horretarako, CPU-k irteera bereziak baditu eten eskaerak onartzeko, INTA hain zuzen (etenaren onarpena), A.12 irudian ikus daitekeen bezala. Irteera hauen kopurua INT sarreren kopuru bera da [INT1,...,K ====> INTA1,...,K].

CPU-k eten-eskaera onartzen duenean aktibatu den sarrerari dagokion INTA irteera aktibatzen du. Seinale hau kontrolagailuen sarrera izango da. Etena eskatu duen periferikoak seinale hau aktibatu dela detektatzean, CPU-ri informazioa bidaliko dio honek jakin dezan zein den exekutatu behar duen zerbitzu-errutina.

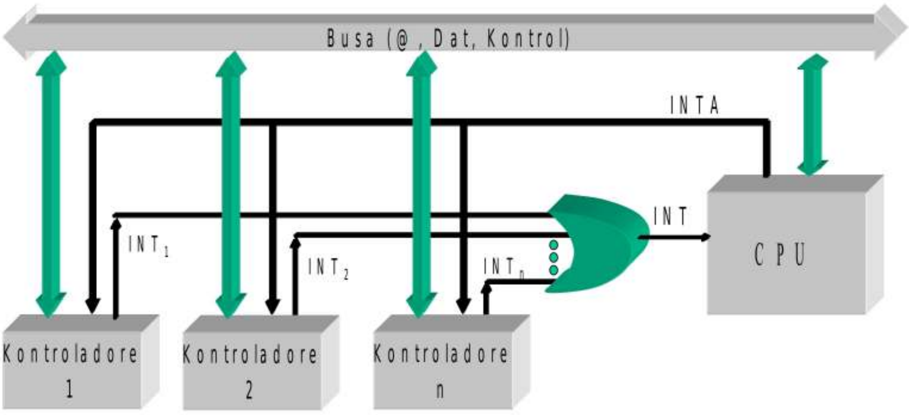


Figure A.12: Kontrolagailuaren identifikazioa egiten duen hardwarearen eskema

Kontrolagailuak prozesadoreari bidaltzen dion informazioa ondokoetako bat izan daiteke:

- Zerbitzu-errutinaren helbidea (SIGNETICS 2650).
- Prozesadoreak exekutatu behar duen aginduaren eragiketa-kodea; agindu hau, logikoki, zerbitzu-errutinarako jauzia da (i8080).
- Prozesadoreak exekutatu behar den zerbitzu-errutinaren helbidea gordetzen duen taula bat atzitzeko erabiliko duen identifikadorea. Teknika honi **“eten bektorizatua”** izena ematen zaio (i8086).

Edozein kasutan ere, bigarren arazoa ebazteke daukagu, hots: zer egin une berean periferiko batek baino gehiagok eten-eskaera sortzen badu? Aurreko eskemari ez badiogu ezer gehitzen, etena eskatu zuten periferiko guztiek, INTA seinalea detektatzean, dagokien informazioa bidali nahi izango liokete prozesadoreari, guztiek batera! Arazo hau ebazteko erabiltzen den aukeretako bat ikusiko dugu, **Daisy-chain edo margarita-katea** (ikusi A.13 Irudia).

CPU-k eten-eskaera hartu ondoren onartzea erabakitzen duenean INTA irteera aktibatzen du. Kontrolagailu bakoitzak INTA sarrera eta irteera bana baditu. Kontrolagaily batek bere INTA sarrera aktibatuta detektatzen duenean, berak ez badu eten-eskaera luzatu, orduan bere INTA

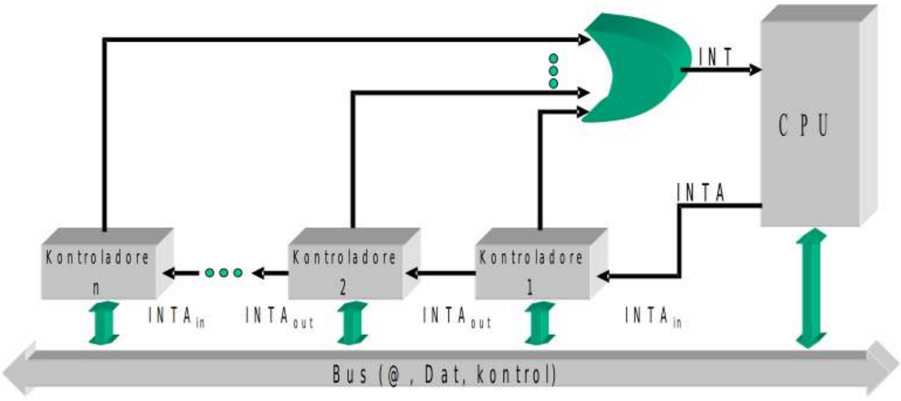


Figure A.13: Margarita-katearen hardware eskema

irteera aktibatzen du hurrengo kontrolagailuak azter dezan; prozesu hau errepikatzen da etena eskatu duen kontrolagailuraino heldu arte. Honek ez du bere INTA irteera aktibatzen eta prozesadoreari errutina egokia exekuta dezan behar duen informazioa bidaltzen dio.

Teknika honen bitartez, prozesadoretik hurbilen dauden periferikoak dira lehentasun gorenekoak (INTA seinalea lehenago hartzen baitute). Ikusi A.13 Irudia.

A.3.4 Zerbitzu-errutinaren exekuzioa

Exekutatu behar den zerbitzu-errutina zein den aztertu ondoren, burutu behar diren eragiketak periferikoaren eta lortu nahi den emaitzaren menpekoak dira.

A.3.5 Etendako programaren egoera berreskuratu

Zerbitzu-errutinaren azken agindua itzuliarena da. Itzulera aginduak etendako programaren egoera (PC eta PSW) berreskuratzen du bere exekuzioa berriro has dadin.

A.3.6 Maila anitzeko etenak

Konputagailu gehienetan CPU eten daiteke aurreko eten baten tratamendurako zerbitzu-errutina exekutatzen ari den bitartean. Kasu honetan, zerbitzu-errutina berrira jauzten da eten berria aurrekoa baino lehentasun handiagokoa bada. Beraz, n lehentasuna duen etena sortzen denean, kasu egingo zaio etenak baimenduta badaude eta momentu horretan lehentasun handiagoko zerbitzu-errutinaren bat exekutatzen ari ez bada. Hau da, lehentasun mailak ezartzen dira. Kasu honetan IF ez da 0-ra jarriko automatikoki.

A.14 Irudian, lehen ikusitako Daisy-Chain hardwarearen eskeman, maila anitzeko etenekin lan egin ahal izateko egin beharreko aldaketak zeintzuk diren ikus daiteke.

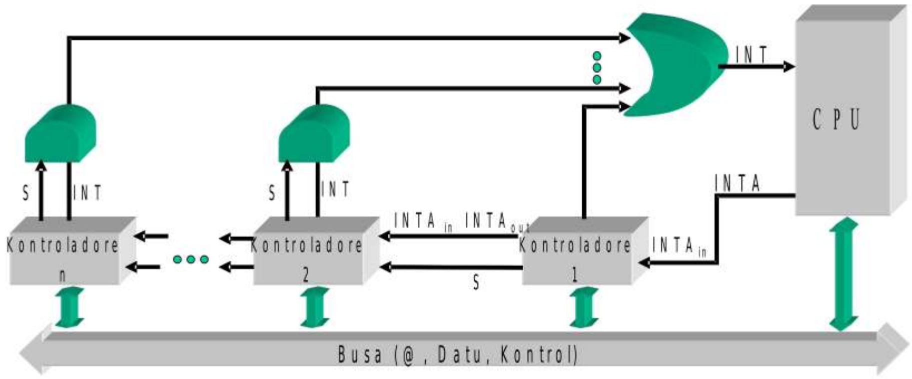


Figure A.14: Daisy-chain hardwarea maila-anitzeko etenak jasateko aldaketekin

K_i kontrolagailuak, etena eskatu duenak hain zuzen, INTA seinalea hartzen duenean CPU-ri zerbitzu-errutina identifikatzeko behar duen informazioa bidaltzen dio eta Si seinalea desaktibatzen du; honela adierazten du berari dagokion zerbitzu-errutina exekutatzen ari dela. Orain K_{i+1} kontrolagailuak etena eskatzen badu INTA seinalea ez zaio iritsiko S_i seinalea desaktibatuta dagoenez prozesadorera iristen den INT seinalea ez baita aktibatuko; beraz, lehentasuna errespetatzen da. Etena K_{i-1} kontrolagailuak eskatzen badu berriz zerbitzatuko da. Zerbitzu-errutinaren exekuzioa amaitzen denean Si seinalea aktibatu beharko da.

A.3.7 Etenen kontrolagailua

Etenen kontrolagailua CPU eta S/I-ko kontrolagailuen arteko bitartekaria da, bera dela medio etenen kudeaketa eraginkorra lortzen delarik. Eten-eskaerak hartzen ditu eta lehentasunaren arabera erabaki behar du heldutako eskaera CPU-ri luzatu behar zaion ala ez. Eten-eskaera CPU-k onartzen duenean, etenen kontrolagailuak zerbitzu-errutina identifikatzeko behar den informazioa bidaltzen dio CPU-ri.

Etenen kontrolagailuaren beste aukerak hauexek dira:

- etenak bereiztuta maskaratzea,
- lehentasunak programatzea (dinamikoki),
- etenen kontrolagailuak kateatzea.

A.4 Memoriarako Atzipen Zuzena (DMA – Direct Memory Access)

Memoria eta periferikoen artean datu-kopuru itzela transferitu behar denean edo *abiadura handiko transferentzia* bat egin behar denean orain arte ikusitako metodoa ez da batere egokia. Datu guztiek prozesadoretik pasa behar badute, denbora asko erabiliko da S/I eragiketetan.

Demagun disko eta memoriaren arteko transferentzia bat egin nahi dugula (ad. sektore bat memoriara pasa nahi dugula):

- Datu asko idatzi behar dugunez gero, prozesadoreak denbora luzea emango du lan hau betetzen, beste gauzarik egin gabe.
- Datu guztiek prozesadoretik pasa behar dutenez gero, transferentzia ez da "oso azkarra" izango eta, ondorioz, ez ditugu periferiko eta memoriaren abiadura-ezaugarriak aprobetxatuko.

Daukagun eskema A.15 Irudian ikus daitekeena da.

Transferentzia hori azkarragoa izan dadin, hardware berezi bat sartuko dugu sisteman horrelako datu-mugimenduez arduratzeko (kontuan

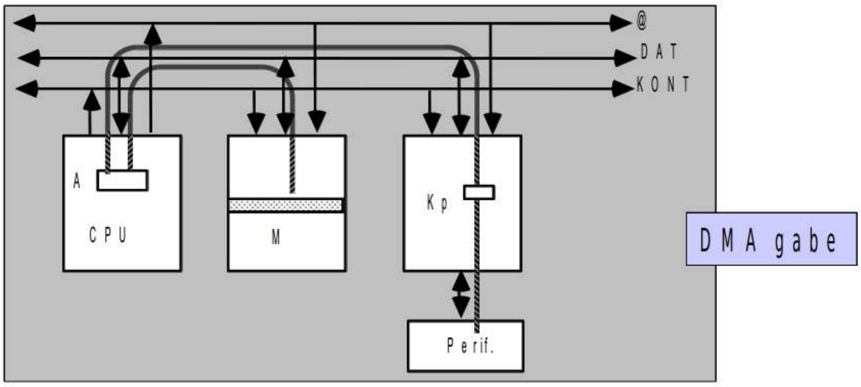


Figure A.15: Memoriako atzipena DMA gabe

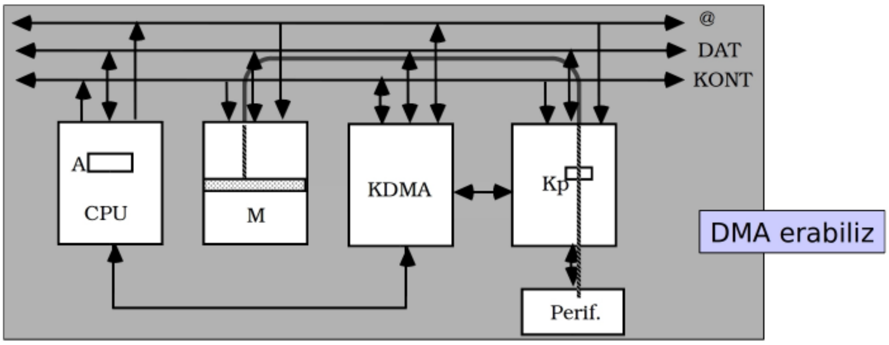


Figure A.16: Memoriako atzipena DMA erabiliz

hartu oso lan erraza dela). Zirkuitu hau DMA (memoriarako atzipen zuzena) kontrolagailua da. Aurreko eskema A.16 irudian ikusten den bezala geratuko litzateke:

DMA-ren bidezko sarrera/irteera bakarrik erabiltzen da memoria-posizio kontsekutiboetan dauden datuak transferitu behar direnean. Tipikoenak adibidez, kanpoko memoria-biltegi bat eta memoria nagusiaren arteko datu-mugimenduak dira.

DMA moduko idazketa/irakurketa prozesuaren hasieran **prozesadoreak**

DMA kontrolagailua (KDMA) programatuko du lan horretaz ardura dadin. Sarrera/irteera kontrolatzeko egin behar dena, normalean, bi gauza dira: helbidea helbide-busean jarri eta RD/WR seinalea aktibatu kontrol-busean. Ondoren, helbidea eta bidali behar den datu-kopurua gaurkotzen dira eta datu berri bat bidaltzen da, guztiekin bukatu arte.

Hala izanik, **busak kontrolatzen dituzten bi gailu daude** orain: alde batetik prozesadorea, normalean busak kontrolatzen dituena, eta bestaldetik, KDMA, S/I-ko prozesu hauetan busen kontrola bere gain hartzen duena. Biek batera, prozesadoreak eta DMA kontrolagailuak, memoria atzitzea ezinezkoa denez gero, soluzio bat aurkitu behar da.

Aukera bat portu anitzeko memoria bat erabiltzea da. Ikusi A.17 irudia.

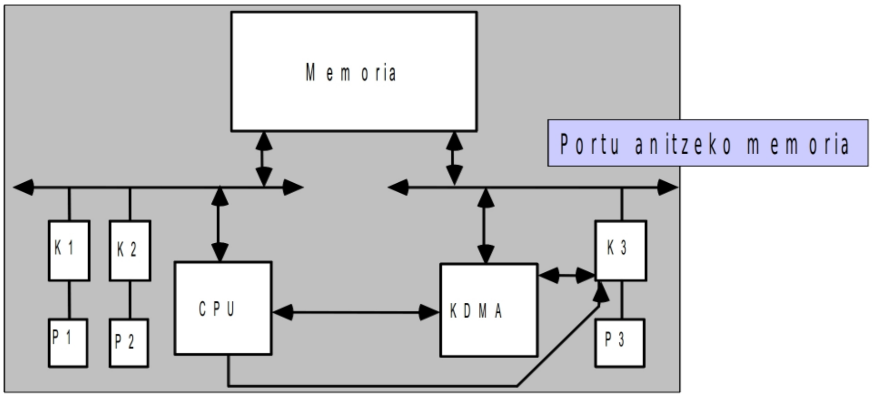


Figure A.17: Portu anitzeko memoria

Kasu honetan bi bus daude eta CPUk bat kontrolatzen duen bitartean, KDMA-k bestea kontrolatzen du. Horrela, P3 (irudian) eta memoriaren arteko transferentzia bat egiten ari den bitartean, prozesadoreak, bere aldetik, bere lanarekin jarraitzen du. Transferentzia osoa bukatuta dagoenean, DMA kontrolagailuak abisatuko dio prozesadoreari eta kito! Sinkronizazio hau, beti bezala, inkestaren bidezkoa (CPU-k burutzen duena) edo etenen bidezkoa (KDMA-k burutzen duena) izan daiteke. Azken hau erabiltzen bada, CPUa behin bakarrik eteten da.

Portu anitzeko memoriak garestiak dira. Horregatik, prozesadoreak eta KDMA-k bus bera erabiltzen dute askotan memoria atzitzeko. Ondorioa hauxe da, batek busak erabiltzen dituenean besteak deskonektatuta egon behar du, inpedantzia altuko egoeran. Bigarren eskema A.18 Irudian ikus daiteke.

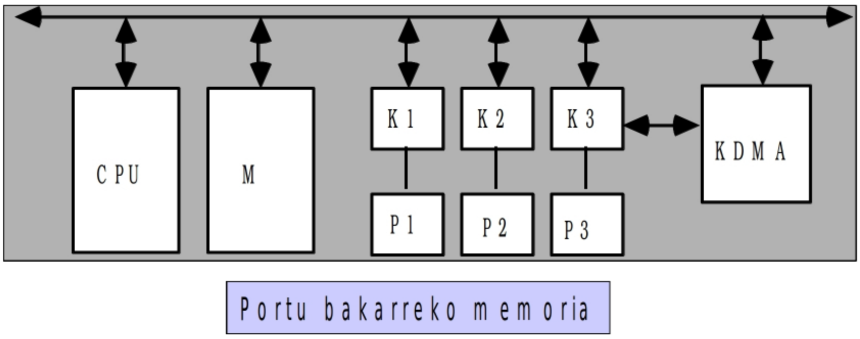


Figure A.18: Portu bakarreko memoria

Jakina, paralelotasuna askoz txikiagoa da orain, bata lana egiten ari denean (memoria irakurri edo idatzi) bestea lanik egin gabe (memoriarekin behintzat) geratuko baita.

A.4.1 DMA kontrolagailua

DMAko kontrolagailu arrunt batean erregistro batzuk agertzen dira normalean. Haien bidez zirkuituaren funtzionamendua programatzen da. Hauen artean daude:

- **Helbide erregistroa:** erregistro honetan transferentziaren hurrengo datuaren helbidea daukagu.
- **Luzera erregistroa:** transferitzeko geratzen diren datu kopurua gordetzen du.
- **Kontrol erregistroa:** prozesua kontrolatzeko informazioa gordetzen du, hala nola:

  - sarrera- ala irteera-prozesua den (rd - wr)
  - inkrementatu/dekrementatu: helbideak kontrolatzeko, hau da, datu bat bidali eta gero helbidea inkrementatu edo dekrementatu behar ote den.
  - sinkronizazio-modua: prozesua bukatu dela adierazteko etenak bai ala ez.
  - prozesu-mota: ziklo-lapurketa, blokeka, . . .

- **Egoera erregistroa:** prozesua ondo bukatu ote den jakiteko.

DMA kontrolagailuaren eskema A.19 Irudian ikus daiteke.

A.4.2 DMA bidezko transferentzia

DMA bidezko sarrera/irteera bat egiteko hauek dira jarraitu behar diren pausuak:

- Transferentziaren hasieraketa.
- Transferentzia.
- Transferentziaren amaiera.

Ikus ditzagun hiru pauso hauek zehatzago:

Transferentziaren hasieraketa

Prozesadoreak kontrolagailuak programatu behar ditu. Horretarako, lehenik kontrolagailu hauek prest dauden jakin beharko du sarrera/irteerako beste eragiketetan bezala.

- Aztertu KDMA eta KPER prest ote dauden
- Programatu KDMA
  - hasierako memoria-helbidea bidali.
  - datu-kopurua adierazi
  - kontrol-informazioa idatzi (norantza, etenak, transferentzia mota)

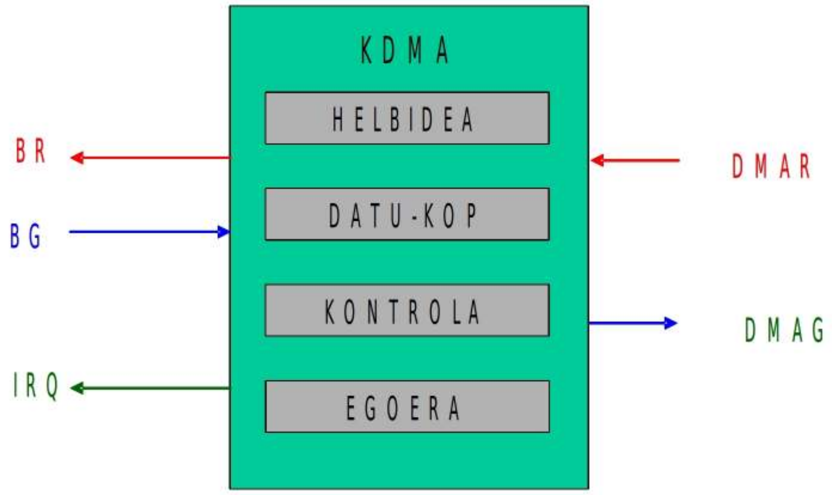


Figure A.19: DMA kontrolagailuaren erregistroak

- Programatu KPER (periferikoaren araberakoa).

Transferentzia

Transferitu behar den datu bakoitzeko honako pausoak jarraitu beharko dira (ikusi zenbakiak A.20 Irudian):

1. KPER kontrolagailuak **DMAR** (DMA Request) seinalea aktibatzen du transferentzia bat egiteko prest dagoela adierazteko: datu bat dauka edo datu bat hartzeko prest dago.
2. KDMA kontrolagailuak **BR** (Bus Request) seinalea aktibatzen du CPU-ri busen kontrola eskatzeko. erikoaren araberakoa).
3. CPU-k bus zikloa amaitzen du eta **BG** (Bus Grant) seinalea aktibatzen du, busen kontrola KDMA-ri emanez eta bere irteera lerroak inpedantzia altuko egoeran jarriz.
4. KDMA kontrolagailuak helbidea kokatzen du helbide busean, R/W seinalea aktibatzen du eta **DMAG** (DMA Grant) seinalea bidaltzen dio KPER-eri datu bat irakur edo idatz dezan datu busean.

5. Eragiketa amaitzerakoan, KDMA-k BR seinalea desaktibatzen du CPUri busen kontrola emateko.

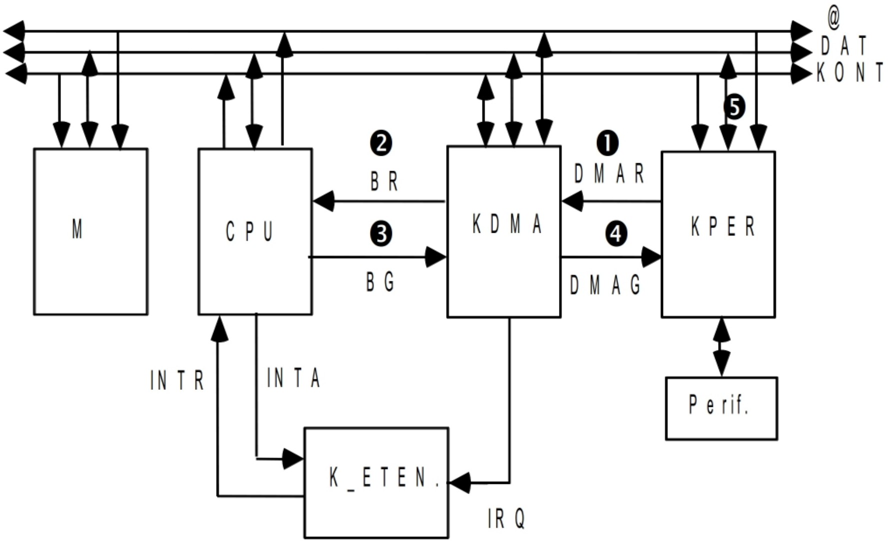


Figure A.20: DMA kontrolagailuaren funtzionamendua

Pausu hauek transferitu behar den datu bakoitzeko errepikatzen direnean transferentzia **ziklo-lapurketa** (*cycle stealing*) motakoa dela esaten da, noizbehinka DMA-k ziklo bat lapurtzen diolako CPU-ri. Periferikoaren abiadura handia bada, KDMA-k etengabe eskatuko dio prozesadoreari ziklo bat datu-transferentzia bat egiteko eta teknika hau ez da oso eraginkorra izango.

Orduan beste teknika bat erabil daiteke: **blokekako transferentzia** (*burst*). Prozedura honen bidez KDMA-k busen kontrola lortzen duenean datu-bloke oso bat bidali ondoren bueltatzen dio busen kontrola prozesadoreari.

Transferentziaren amaiera (sinkronizazioa)

Blokearen transferentzia amaitzen denean, CPU eta KDMA sinkronizatu egin behar dira. Normalean etenen bidez egiten da sinkronizazio hau (baina hau ere programa daiteke). KDMA-ren zerbitzu-errutinak egoera-erregistroa aztertzen du KDMA transferentzia ondo joan den egiaztatzeko. Eten-eskaera sortzen duena bai KDMA bai KPER izan daiteke.

- **Inkestaren bidezko sinkronizazioa:**

  Periferikoari eragiketa-ordena bidali ondoren programa nagusiak KDMA-ren egoera-erregistroan inkesta egiten du honek eragiketa bukatu dela (datu guztiak transferitu dituela) adierazten dion arte.

  Garbi dagoenez aukera hau ez da oso logikoa CPU-k ez baitu lanik egiten informazio- transferentziak dirauen bitartean. Beraz, ez da ezer irabazten lan guztia CPU-k burutzen duen kasuarekin konparatuta. Zerbait irabaztekotan abiaduraren aldetik izango litzateke KDMA-ren abiadura handiagoa baita CPUrena baino.

- **Etenen bidezko sinkronizazioa:**

  Programa nagusiak kontrolagailuak programatzen ditu hasieran informazio-transferentziari hasiera emateko baina gero transferentziak dirauen bitartean beste kalkulu batzuk (memoria erabiltzen ez dutenak noski) egin ditzake. KDMA-k datu guztiak transferitu direla detektatzen duenean CPUri eten-eskaera bidaltzen dio transferentziaren bukaera adierazteko.

  KDMA-ak CPU etengo transferentzia bukatzean. Etena sortzen denean exekutatu behar den errutina honelakoxea izango da:

  ```
  irakurri egoera-erregistroa
  baldin errorea orduan
      programatu KDMA berriro
      programatu KDISKO berriro
  bestela bukatu
  ambaldin
  ```

  Honela CPU-ri S/I-ko lan astunaren zati bat kentzen zaio, orain beste zirkuitu batek (KDMA) burutzen duelarik lan hori.

APPENDIX B

Notes on the specifics of the Nintendo DS for the students

This appendix shows the lecture notes on the functioning of the NDS devices, that students are handed out. The appendix is in Basque, one of the languages used in the course.

SARRERA/IRTEERA NDS-AN

Aurreko gaian NDS makinarekin lan egin dugu, mihiztadura lengoaian programatuz. Konpiladoreak egiten duen lana aztertu dugu, eta goi mailako lengoaia batean idatzitako programa bat nola itzultzen den mihiztadura lengoaiara edo makina lengoaiara ere. Orain arte ordea, ez dugu sarrera/irteerako azpisistemarekin zerikusia duen ezer landu.

Gai honetan, sarrera/irteerako funtzionamenduaren teoria orokorra ezagutzen dugunez, Nintendo DS-aren sarrera/irteera aztertuko dugu. Horretarako, garrantzitsuak iruditzen zaizkigun makina honen hardwarearen atalak deskribatuko ditugu, alde batetara utziz ikasturte aurreratuago batean landu daitezkeen zehaztasunak.

B.1 Hardwarearen deskribapena

B.1.1 Prozesadoreak

Dakigun bezala, NDSak bi prozesadore ditu, ARM9 (66 MHz) bata eta ARM7 (33 Mhz) bestea. Nintendok NDSrako diseinu batean integratu ditu biak. ARM9 prozesadorea batez ere grafikoen kontrolaz arduratzen da eta ARM7 prozesadorea periferikoen kontrolaz.

Bi prozesadore izateak hainbat zailtasun dakartza berarekin, bi prozesadoreek elkarrekin lan bat burutzen dutenean beraien arteko komunikazioa ezinbestekoa delako. Gure lanerako sinplifikazio bat egingo dugu prozesadore bakarrarekin lan egiten dugula suposatuz. ARM7 prozesadoreak ezin ditu grafikoak tratatu. ARM9 prozesadorea ordea, periferiko garrantzitsuenen informazioa atzitu dezake. Beraz, azken hau izango da gure proiektuan zehar erabiliko duguna. Hala ere, badago periferikoren bat bakarrik ARM7ak kontrolatzen duena. Horrelako periferiko bat atzitzea nahi dugunean liburutegiko funtzioak erabiliko ditugu, prozesadoreen arteko komunikazioa nola burutzen den aztertu gabe.

Gaur egun gure inguruan aurkitu ditzakegun beste hainbat sistema txertatutan gertatzen den bezala, makina honek ez du sistema eragilerik. NDSa PCtik desberdintzen duen ezaugarri honek makina behe mailan kontrolatzeko aukera ematen digu, sarrera/irteerako erregistroak, etenen kudeatzailearen erregistroak eta eten-taula atzituz.

B.1.2 Memoria

B.1 Irudian ikus daitekeen bezala NDSaren memoria-sistema konplexua da. Guzti honetatik guk jakin behar duguna da sarrera/irteerako kontroladoreen erregistroak eta etenen kudeaketarako erregistroak beti memorian mapeatuta daudela, eta beraz, ez dugula agindu berezirik behar erregistro horiek atzitzeko.

Ondoren ikusiko dugu klasean erabiliko dugun periferiko bakoitzaren kontrolagailuen erregistroak zein memoriako helbidetan mapeatzen diren eta nola konfiguratu dezakegun memoria, fondo grafikoak eta sprite-ak erabiltzeko.

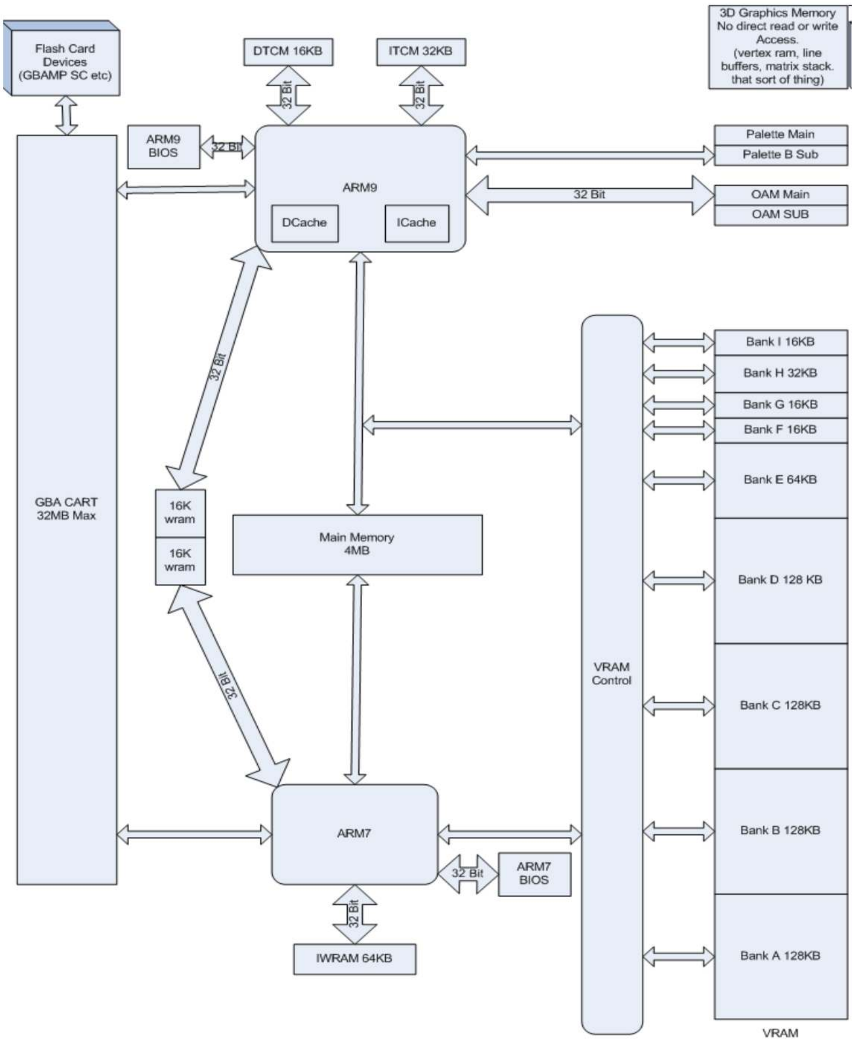


Figure B.1: NDSaren hardwarearen eskema

B.1.3 Pantailak

NDS makinek badaukate jolasteko beste makina eramangarrietatik bereizten dituen ezaugarri bat; bi pantaila dituela. Beheko pantaila gainera, ukimen pantaila bat da. Kontrolatzeko moduari dagokionez, pantaila grafikoa eta ukimen pantaila periferiko desberdinak balira bezala kontrolatzen dira. Horregatik dokumentu honetan bananduta aztertuko ditugu.

Pantaila grafikoak

Bi pantaila grafikoak LCD (*Liquid Crystal Display*) motakoak dira. Biak tamaina berekoak dira, 256x192 pixel, eta trukatu ditzaketen bi motor grafiko erabiltzen dituzte. Bi motorretako batek hiru dimentsiotako irudiak tratatu ditzake.

Irudiak bitarrera itzultzen dira eta programarekin batera gordetzen dira. Irudi bitar hauek pantailan atera daitezen pantaila mapeatzeko erabiltzen den VRAM (Video RAM) memoriako bankuan kopiatu behar dira.

Baina, zer nahi dugu esatea pantaila memoriako banku batean **mapeatzen** dela diogunean? bada, pantailari memoriako banku hori esleitzen ziola, eta beraz, banku horretan idazten dena izango dela pantailan aterako dena.

Pantailan erakutsi daitezkeen irudiak gutxienez bi motatakoak izan daitezke: **fondoak** eta **sprite**-ak. Fondoak, desplazatu, biratu, eraldatu eta dimentsioz alda daitezkeen arren, mugimendu gaitasun gutxiago dute sprite-ak baino. Azken hauek, pantaila guztian zehar mugitzen diren animazioak izan daitezke.

Grafikoak maneiatzea oso lan konplexua denez, nahiz eta grafikoekin lan egiteko ezagutu behar den memoria helbideratzea ikasgairako interesgarria izan, grafikoen tratamendurako kode gehiena praktikarako txantiloi batean emango da.

FONDOAK B.1 Taulan ikus dezakegu ze banku dauzkan VRAM memoriak. Banku hauek, pantailak mapeatu eta fondoak erakusteko erabil daitezke. B.2 Irudian ikusten da nola mapeatu daitezkeen pantailak memoriako bankuetan, klasean erabiliko dugun txantiloian agertzen den bezala.

B.2 Irudian ikus daitekeen bezala, pantaila nagusia A eta B bankuetan mapeatzen da eta bigarren pantaila C bankuan. E bankua beste kontu batzuetarako erabiliko da.

bankua	kontrol erregistroa	tamaina	erabilerak
VRAM_A	VRAM_A_CR	128 KB	VRAM_A_LCD VRAM_A_MAIN_BG_0x6000000 = VRAM_A_MAIN_BG VRAM_A_MAIN_BG_0x6020000 VRAM_A_MAIN_BG_0x6040000 VRAM_A_MAIN_BG_0x6060000 VRAM_A_MAIN_SPRITE VRAM_A_TEXTURE_SLOT0 = VRAM_A_TEXTURE VRAM_A_TEXTURE_SLOT1 VRAM_A_TEXTURE_SLOT2 VRAM_A_TEXTURE_SLOT3
VRAM_B	VRAM_B_CR	128 KB	VRAM_B_LCD VRAM_B_MAIN_BG_0x6000000 VRAM_B_MAIN_BG_0x6020000 = VRAM_B_MAIN_BG VRAM_B_MAIN_BG_0x6040000 VRAM_B_MAIN_BG_0x6060000 VRAM_B_MAIN_SPRITE VRAM_B_TEXTURE_SLOT0 VRAM_B_TEXTURE_SLOT1 = VRAM_B_TEXTURE VRAM_B_TEXTURE_SLOT2 VRAM_B_TEXTURE_SLOT3
VRAM_C	VRAM_C_CR	128 KB	VRAM_C_LCD VRAM_C_MAIN_BG_0x6000000 VRAM_C_MAIN_BG_0x6020000 VRAM_C_MAIN_BG_0x6040000 = VRAM_C_MAIN_BG VRAM_C_MAIN_BG_0x6060000 VRAM_C_ARM7 VRAM_C_SUB_BG_0x6200000 = VRAM_C_SUB_BG VRAM_C_SUB_BG_0x6220000 VRAM_C_SUB_BG_0x6240000 VRAM_C_SUB_BG_0x6260000 VRAM_C_TEXTURE_SLOT0 VRAM_C_TEXTURE_SLOT1 VRAM_C_TEXTURE_SLOT2 = VRAM_C_TEXTURE VRAM_C_TEXTURE_SLOT3
VRAM_D	VRAM_D_CR	128 KB	VRAM_D_LCD VRAM_D_MAIN_BG_0x6000000 VRAM_D_MAIN_BG_0x6020000 VRAM_D_MAIN_BG_0x6040000 VRAM_D_MAIN_BG_0x6060000 = VRAM_D_MAIN_BG VRAM_D_ARM7 VRAM_D_SUB_SPRITE VRAM_D_TEXTURE_SLOT0 VRAM_D_TEXTURE_SLOT1 VRAM_D_TEXTURE_SLOT2 VRAM_D_TEXTURE_SLOT3 = VRAM_D_TEXTURE
VRAM_E	VRAM_E_CR	64 KB	VRAM_E_LCD VRAM_E_MAIN_BG VRAM_E_MAIN_SPRITE VRAM_E_TEX_PALETTE VRAM_E_BG_EXT_PALETTE VRAM_E_OBJ_EXT_PALETTE

continued on next page

continued from previous page

bankua	kontrol erregistroa	tamaina	erabilerak
VRAM_F	VRAM_F_CR	16 KB	VRAM_F_LCD VRAM_F_MAIN_BG VRAM_F_MAIN_SPRITE VRAM_F_TEX_PALETTE VRAM_F_BG_EXT_PALETTE VRAM_F_OBJ_EXT_PALETTE
VRAM_G	VRAM_G_CR	16 KB	VRAM_G_LCD VRAM_G_MAIN_BG VRAM_G_MAIN_SPRITE VRAM_G_TEX_PALETTE VRAM_G_BG_EXT_PALETTE VRAM_G_OBJ_EXT_PALETTE
VRAM_H	VRAM_H_CR	32 KB	VRAM_H_LCD VRAM_H_SUB_BG VRAM_H_SUB_BG_EXT_PALETTE
VRAM_I	VRAM_I_CR	16 KB	VRAM_I_LCD VRAM_I_SUB_BG VRAM_I_SUB_SPRITE VRAM_I_SUB_SPRITE_EXT_PALETTE

Table B.1: VRAM memoriako bankuak eta beraien erabilera posibleak.

```
vramSetMainBanks(VRAM\_A\_MAIN\_BG\_0x06000000, \\
                 VRAM\_B\_MAIN\_BG\_0x06020000, \\
                 VRAM\_C\_SUB\_BG\_0x06200000, \\
                 VRAM\_E\_LCD); \\
```

Figure B.2: Pantaila grafikoak VRAM bankuetan mapeatzearen adibidea.

SPRITEAK B.1 Irudiko eskuineko goiko partean, 3D grafikoetarako memoriaren azpian, spriten koloreen paletarako memoria dago eta honen azpian OAM memoria ikus daiteke, sprite-ak gordetzen dituen memoria. Memoria honen hasieratzea B.3 Irudian ikus daiteke.

Hasieratze honetan koloreen formatua 256 koloretakoa izatea aukeratu da, eta horregatik 8 bit beharko dira kolorea identifikatzeko. Koloreen paletan 0 eta 255 arteko balio bakoitzari kolore bat esleitzen zaio. Kolore hori RGB-15 formatuan adierazten da, 15 bit erabiltzen dituena kolore bat adierazteko, 5 bit osagai (gorria, berdea, urdina) bakoitzaren intentsitaterako, 31 balioa izanik intentsitate handienekoa. Adibidez proiekturako erabiliko den txantiloian, paletako 1 balioari kolore gorria esleitu zaio eta 2 balioari urdina, B.4 Irudian ikusten den

```
void initSpriteMem() {
   oamInit(&oamMain, SpriteMapping_1D_32, false);
   oamInit(&oamSub, SpriteMapping_1D_32, false);
   gfx=oamAllocateGfx(&oamMain,SpriteSize_16x16,
SpriteColorFormat_256Color);
   gfxSub=oamAllocateGfx(&oamSub,SpriteSize_16x16,
SpriteColorFormat_256Color);
}
```

Figure B.3: Sprite-n memoria hasieratzeko kodea.

bezala. Paletako 0 balioari esleitutako kolorea beti gardena da.

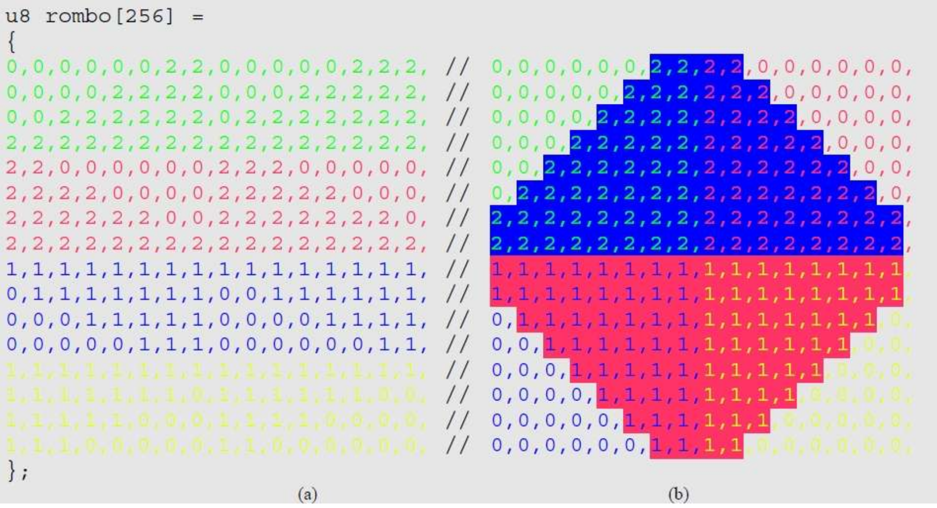
```
void establecerPaletaPrincipal() {

       SPRITE_PALETTE[1] = RGB15(31,0,0);
//1 baliodun pixelak gorriak dira
       SPRITE_PALETTE[2] = RGB15(0,0,31);
//2 baliodun pixelak urdinak dira
}
}
```

Figure B.4: Sprite-n koloreen paleta nola definitzen den ikusteko adibidea.

Beste aukera bat 16 koloretako paleta bat erabiltzea da. Modu honetara, 16 paleta desberdin erabil daitezke. Lehenengo aukera erabiliko dugu bere tratamendua sinpleagoa delako.

Kontuan hartu behar den beste gauza bat ondokoa da: memoriako bankuko byten lehen laurdena irudiko goiko ezkerreko koadrantean erakusten dela, byten bigarren laurdena goiko eskuineko koadrantean, byten hirugarren laurdena beheko ezkerreko koadrantean eta byten laugarren laurdena beheko eskuineko koadrantean. B.5b Irudian ikus dezakegu nola erakutsiko liratekeen NDSaren pantailan B.5a Irudiko bektorearen bidez adierazitako datuak.

Pixel bakoitza byte baten bidez adierazten den arren, bideo memoriako posizio bakoitza 2 bytekoa da, eta horregatik, bideo memorian idazketak aldiko 16 bit idatziz egin behar dira. B.6 Irudian ikusten da nola idatziko litzatekeen bideo memorian B.5 Irudiko erronbo bektorea. Kontuan izan,



```
u8 rombo[256] =
{
0,0,0,0,0,0,2,2,0,0,0,0,0,2,2,2,  //  0,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,
0,0,0,0,2,2,2,2,0,0,0,2,2,2,2,2,  //  0,0,0,0,0,2,2,2,2,2,2,0,0,0,0,0,
0,0,2,2,2,2,2,2,0,2,2,2,2,2,2,2,  //  0,0,0,0,2,2,2,2,2,2,2,2,0,0,0,0,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,  //  0,0,0,2,2,2,2,2,2,2,2,2,2,0,0,0,
2,2,0,0,0,0,0,0,2,2,2,0,0,0,0,0,  //  0,0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,
2,2,2,2,0,0,0,0,2,2,2,2,2,0,0,0,  //  0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,
2,2,2,2,2,2,0,0,2,2,2,2,2,2,2,0,  //  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,  //  2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,  //  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,  //  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
0,0,0,1,1,1,1,1,0,0,0,0,1,1,1,1,  //  0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,
0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,  //  0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,  //  0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
1,1,1,1,1,1,1,0,1,1,1,1,1,1,0,0,  //  0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,
1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,  //  0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,
1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,  //  0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,
};
```

Figure B.5: Sprite baten errepresentazioa osokoen bektore baten bidez.

makina little-endian izanik, biteak alderantzizko ordenean kopiatzen direla memorian.

```
for(i = 0; i < 16 * 16 / 2; i++)
   gfx[i+(16*16/2)] = erronbo[i*2] | (erronbo[(i*2)+1]<<8);
}
```

Figure B.6: Sprite baten pixelak spriten memoriara kopiatzen dituen adibidea.

Ukimen pantaila

Ukimen pantaila, lehen azaldu den bezala, bakarrik AMR7 prozesadoretik atzi daiteke. Hala ere, *libnds* liburutegiak hainbat funtzio eskaintzen ditu ukimen pantaila zein posiziotan sakatu den jakiteke balio dutenak. B.7 Irudian ikus daiteke nola egin dezakegun ukimen pantailari buruzko inkesta bat nahiz eta bere erregistroak ezin ditugun zuzenean atzitu.

```
touchPosition pos_pantalla; // aldagaiaren definizioa

touchRead(&pos_pantalla); // posizioaren irakurketa
           while(pos_pantalla.px==0 && pos_pantalla.py==0)
              // inkesta
       touchRead(&pos_pantalla); // posizioaren irakurketa
```

Figure B.7: Ukimen pantailaren kontrola inkesta bidez.

Pantaila programen arazketarako

Pantailak aurreko ataletan azaldu den bezala definitzen baditugu, bakarrik balio dute grafikoak erakusteko. Hori horrela izanda, pantailan testu bat erakustsi nahi badugu, testu horrek grafikoetan idatzita egon beharko du, horrela fondo moduan erakutsiko delarik.

Programazio inguruneak eskaintzen digun araztailea oso erabilgarria izan daiteke zenbait kasutan, baina kontutan hartu behar da erabilitako programazio modua gertaerei zuzendutakoa dela eta programan jartzen ditugun geldiune puntuak sortzen diren gertaeretan eragina izan dezaketela (gertaerak izateko aukerak murriztu ditzakete). Horregatik, hainbat kasutan oso erabilgarria suerta daiteke pantailan zuzenean zerbait idatzi ahal izatea. Horretarako, *libnds* liburutegiak *consoleDemoInit()* prozedura eskaintze du. Prozedura hau erabiliz bigarren pantailan idatzi ahal izango da *printf()* funtzioa erabiliz. Kasu horretan, pantaila horretako fondoa erakutsiko ez delarik.

B.1.4 Teklatua

NDSaren teklatua nahiko berezia da, oso tekla gutxi ditu, B.8 Irudian ikus daitekeen bezala.

X eta Y teklen erregistroak, ukimen pantaila bezala, bakarrik ARM7 prozesadoretik atzi daitezke. Hau dela eta, eta gainerako teklak atzigarri ditugunez, X eta Y teklak ez ditugu erabiliko. NDSak tekla gutxi dituenez, PCak ez bezala, ez da ASCII taularik behar sakatutako tekla eta bere esanahiaren arteko itzulpena egiteko. NDSan, tekla bakoitzari bit bat esleitzen zaio teklatuaren datu-erregistroan, eta horrela, zein

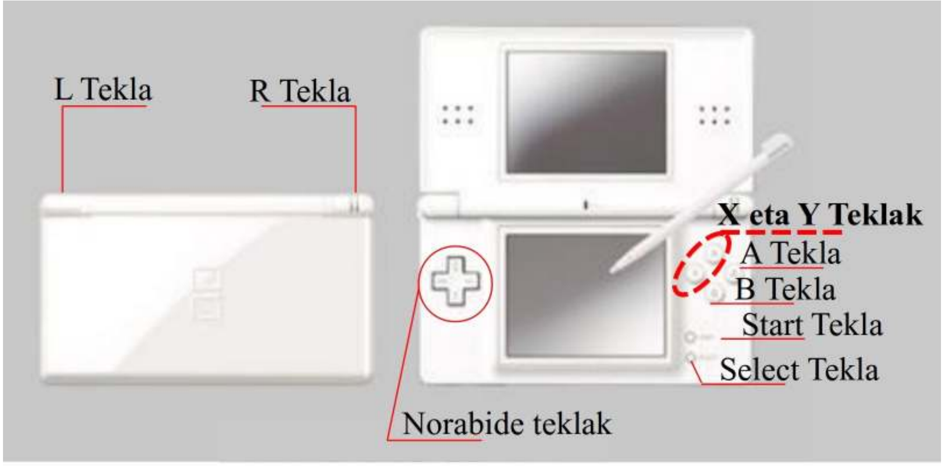


Figure B.8: NDSaren teklak

tekla sakatu den jakin daiteke. Teklatuak datu-erregistro bat eta kontrol-erregistro bat ditu. Erregistro hauek, praktikan erabiliko diren gainerako erregistroak bezala, "defineak.h" fitxategian definituta daude. Definizio hauek B.9 Irudian ikus daitezke.

```
// Teklatuaren erregistroen definizioa
#define TECLAS_DAT    (*(vu16*)0x4000130) //datu erregistroa
#define TECLAS_CNT    (*(vu16*)0x4000132) //kontrol
    erregistroa
```

Figure B.9: Teklatuaren erregistroen helbideen definizioa

Teklatuaren sinkronizazioa inkesta bidez edo etenen bidez egin daiteke. Are gehiago, tekla batzuk etenen bidez kudeatu daitezke eta beste batzuk inkesta bidez. Konfigurazio hau kontrol-erregistroaren bidez egiten da. Etenen bidez tratatu nahi dugunean tekla bat kontrol-erregistroan tekla horri dagokion bitari 1 balioa eman behar zaio. B.2 Taulan ikus dezakegu tekla bakoitzari teklatuaren kontrol-erregistroko zein bit esleitzen zaion.

Tekla bakoitzari esleitzen zaion bitaz gain, kontrol-erregistroak baditu erabiliko ditugun beste bi bit. 14 bitari 1 balioa eman behar zaio

Bita	Deskribapena
0	A tekla
1	B tekla
2	Select tekla
3	Start tekla
4	Eskuinerako norabidea
5	Ezkerrerako norabidea
6	Goranzko norabidea
7	Beheranzko norabidea
8	R tekla
9	L tekla
10-13	ez dira erabiltzen
14	Etena aktibatu (1 aktibatuta / 0 desaktibatuta)
15	Etenaren baldintza (1 AND / 0 OR)

Table B.2: Teklatuaren kontrol erregitroaren biten erabilera

teklatuak etenak sortu ahal izateko. Bit honek 0 balioa hartzen badu, teklei dagozkien gainerako bitetan 1ekoak egon arren, ez da inolako etenik sortuko tekla horiek sakatzean. 15 bita berriz ondoko aukeraketa egiteko erabiltzen da: 0 balioa ematen bazaio tekla bakar bat sakatzean eten bat sortuko da; 1 balioa badu berriz, etena sortuko da hainbat tekla batera sakatzen direnean. Etena sor dadin batera sakatu behar diren teklak kontrol-erregistroan 1 balio hartzen duten guztiak dira.

Teklaren bat sakatzen denean (bai teklatua inkesta bidez eta bai etenen bidez sinkronizatzen denean), datu-erregistroak adieraziko du zein den sakatu den tekla, teklari dagokion bitean 0 balioa erakutsiz. Datu-erregistroan tekla bakoitzari esleitzen zaion bit zenbakia kontrol-erregistroan esleitzen zaion bera da.

B.1.5 Denboragailuak

Denboragailuak maiztasun zehatz batean lan egiten duten kontagailuak dira eta denbora kontrolatzeko balio digute. NDSko prozesadore bakoitzak 4 denboragailu ditu (*Timer0, Timer1, Timer2 eta Timer3*) eta beraien artean seriean konektatu daitezke (*Timer0-tik Timer3-ra*, ordenean) kontrolatu nahi den denbora handiagotzeko. Hauetako kontagailu bakoitza 16 bitekoa da, eta beraz, 65536 arte kontatu dezakete. Beraien maiztasun handiena 33 Mhz da, zehazkiago 33554432 Hz.

Denboragailuek kontrol-erregistro bat izaten dute (beraien helbideak 3.4.11 irudian ikus daitezke). Kontrol-erregistro hauen bidez ondoko

Bita	Balioa	Deskribapena
7	1	Tenporizadorea aktibatu
6	1	Etenak sortu kontaketa bukatzean (gainezkatzean)
2	1	Tenporizadoreak lotu, aurrekoak kontaketa bukatzean hasi kontaketa. Ezin da erabili 0 tenporizadorean.
0-1	0	Maiztasuna zati 1 (M.Max. 33554432 Hz – M.Min. 512 Hz)
0-1	1	Maiztasuna zati 64 (M .Max. 524288 Hz – M.Min. 8 Hz)
0-1	2	Maiztasuna zati 256 (M .Max. 131072 Hz – M.Min. 2 Hz)
0-1	3	Maiztasuna zati 1024 (M .Max. 32768 Hz – M.Min. 0,5 Hz)

Table B.3: Tenporizadoreen kontrol-erregistroaren biten erabilera

konfigurazio lanak burutu daitezke: kontagailuak martxan jartzea, kontagailuak seriean lotzea, kontaketa bukatzean eten bat sor dezatela adieraztea, eta kontagailuen kontaketarako maiztasuna murriztea (ikusi B.3 Taula).

```
#define TIMER0_CNT (*(vuint16*)0x04000102)
#define TIMER1_CNT (*(vuint16*)0x04000106)
#define TIMER2_CNT (*(vuint16*)0x0400010A)
#define TIMER3_CNT (*(vuint16*)0x0400010E)

#define TIMER0_DAT  (*(vuint16*)0x04000100)
#define TIMER1_DAT  (*(vuint16*)0x04000104)
#define TIMER2_DAT  (*(vuint16*)0x04000108)
#define TIMER3_DAT  (*(vuint16*)0x0400010C)
```

Figure B.10: Tenporizadorearen erregistroen helbideen definizioa

Tenporizadoreen datu-erregistroek ere badituzte beraien helbideak 3.4.11 Irudian ikus daitezke. Datu-erregistroen bidez, kontagailuek zein baliotik aurrera hasiko duten kontaketa adierazten da (latch), beti ere 16 biteko balioak ditugula kontuan hartuta.

B.11 Irudiko formula erabilita kalkulatu dezakegu datu-erregistroa zein baliorekin hasieratu behar dugun (kontaketaren hasiera, latch) eta zein maiztasunekin kontatu behar duen kontagailuak lortzea nahi dugun eten maiztasunaren arabera (segundoko zenbat aldiz nahi dugun etetea da eten maiztasuna).

Suposatu adibidez segundoko 5 aldiz nahi dugula kontagailu edo denboragailuek etetea. Maiztasun maximoa utzita (33554432 lehenago esan dugun bezala) ondokoa izango genuke:

$$latch = 65532 - \frac{1}{etenmaiztasuna} * kontaketamaiztasuna$$

Figure B.11: Zein baliotatik hasi behar den kontatzen eta zein maiztasunekin kontatu behar den kalkulatzeko formula.

$$latch = 65532 - \frac{1}{5} * 33554432 = -6645354,4$$

Kontaketa maiztasuna altuegia da, zatitu dezakegu.

$$latch = 65532 - \frac{1}{5} * 33554432/64 = -39325,6$$

Oraindik ere altuegia da, beste zatitzaile handiago bat erabiliko dugu.

$$latch = 65532 - \frac{1}{5} * 33554432/256 = 39317,6$$

Beraz, denboragailuaren datu erregistroan gorde behar dugun balioa 39318 da. Balio horretatik hasiko da kontatzen kontagailua eta 65536-39322=26214 kontaketa egiten dituen bakoitzean eten bat sortuko du. Bestetik finkatu behar duguna da zein azkar egingo duen kontaketa hori. Segundoko 5 aldiz etetea nahi badugu, segundo batean bost aldiz egin behar ditu 26214 kontaketa, beraz, 26214*5 = 131070 kontaketa egin behar ditu segundoko. Kontaketa maiztasun hori maiztasun maximoari 256 zatitzailea jarrita lortzen da 33554432 / 256= 131070 Hz. Beraz, kontrol-erregistroko 0-1 bitetan 2 balioa idatzi beharko da.

B.2 Etenen kudeaketa

PC-an etenen kudeaketa osoa hardware bidez egiten da, eta lan hori burutzen duen zirkuitua etenen-kontroladorea deitzen da. NDSan berriz, etenen kudeaketaren parte handi bat software bidez burutzen da, eta lan hori egiten duen software edo programa *textbfInterrupt Dispatcher* izenarekin ezagutzen da. Etenen kudeaketarekin lotutako hainbat erregistro aurkituko ditugu, baina etenen kontrol logika guztia *Interrupt Dispatcher*rak burutuko du. Bera arduratuko da adibidez etenen kudeaketarekin lotutako erregistroak irakurtzeaz etenak baimenduta dauden edo ez aztertzeko. Etenak baimenduta daudenean eta eten bat sortzen denean *Interrupt Dispatcher*rak atzituko du **eten bektorea** exekutatu behar den zerbitzu errutinaren helbidea lortzeko, makinako egoera erregistroa gordeko du zerbitzu errutinaren exekuzioa bukatzean makinaren egoera berreskuratzeko eta zerbitzu errutinaren exekuzioa abiaraziko du bere helbidea PCan kargatuz.

B.2.1 Etenen kudeaketarako erregistroak (etenen kudeatzailea)

DevkitPro inguruneko [devkitPro] *libnds* liburutegian *Interrupt Dispatcher*raren inplementazioa aurki dezakezue (mihiztadura lengoaian dago). Kode horretan ikus daiteke eten-kudeaketarako erregistroez *Interrupt Dispatcher*rak egiten duen erabilera. Ikus daiteke ere, etenen kudeaketarako erregistro guztiak, sarrera/irteerako beste erregistroak bezala, memorian mapeatuta daudela.

Ondokoak dira etenen kudeaketarako aurkitzen ditugun erregistroak:

- **IME (Interrupt Master Enable)** erregistroa: etenak orokorrean baimenduta dauden edo ez adierazten du. 1 balioa badu erregistro honek etenak baimenduta egongo dira eta 0 balioa eman beharko zaio etenak galaraztea nahi ditugunean.
- **IE (Interrupt Enable)** erregistroa: maskara erregistroa da, eten zenbaki bakoitzeko bit bat izango du, eta beraz, periferiko zehatz bakoitzerako etenak baimenduta dauden edo ez adieraziko du: i

bitak 1 balio badu i etena baimenduta dago. 32 biteko erregistroa da.

- **IF (Interrupt Flag** erregistroa: eten eskaeren erregistroa da, zein periferikok egin duten eten eskaera bat adierazten du. Dituen 32 bitetatik 25 erabiltzen dira, B.4 Taulan ikus daitekeen eten-lerroen taularen arabera.

B.12 Irudian ikus ditzakegu erregistro hauen definizioak eta esleitzen zaizkien memoria helbideak.

```
#define IME          (*(vuint16*)0x04000208)
#define IE           (*(vuint16*)0x04000210)
#define IF           (*(vuint16*)0x04000214)
```

Figure B.12: Etenen kudeaketarako erregistroen definizioa

Adierazi den bezala IE eta IF erregistroek bitez bit eten lerro desberdinak identifikatzen dituzte. B.4 Taulan ikus daiteke bit bakoitzak ze eten-lerro identifikatzen duen eta zein izenekin definitu den *libnds* liburutegian (praktikan erabili ahal izateko).

B.2.2 Eten-bektore edo eten-taula

Eten-taulak, etenen tratamendurako erabiltzen diren zerbitzu-errutinen helbideak gordetzen ditu. Taula hau memorian gordeta egoten da, baina ez helbide finko batean. Eten-taula gordetzeko memoriako hasierako helbidea dinamikoa da eta programa memorian kargatzen denean esleitzen zaio hasierako helbide hori.

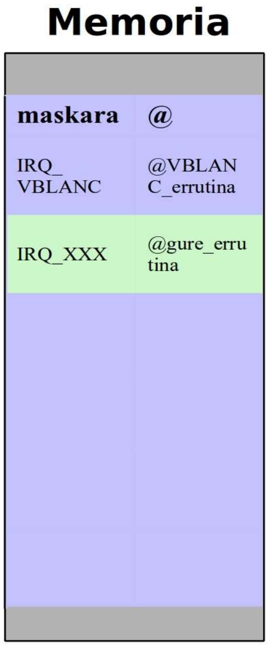
Eten-taulako posizio bakoitza 4 bytekoa da: 2 byte erabiltzen dira zerbitzu errutinaren helbidea gordetzeko eta beste bi byte eten maskararako. Eten-maskarak eten zenbakia identifikatzen du. Hau guztia C.2 Irudian ikus daiteka.

Zerbitzu errutina bat idazten dugunean gure programan, eten taulan gorde beharko dugu bere helbidea. Eten-taulan informazioa idazteko *libnds* liburutegian funtzio bat aurkitu dezakegu lan hori errazten diguna, **irqSet** funtzioa. Funtzio honek bi parametro jasotzen ditu, gure

Bita	Maskara – *libnds*-ko definizioa	Deskribapena
0	IRQ_VBLANC(PANTALLA)	Pantailaren freskatze bertikalaren etena
1	IRQ_HBLANC (PANTALLA)	Pantailaren freskatze horizontalaren etena
2	IRQ_VCOUNT (PANTALLA)	VCOUNT-ekin parekatzeagatik sortutako etena
3	IRQ_TIMER0	0 tenporizadorearen etena
4	IRQ_TIMER1	1 tenporizadorearen etena
5	IRQ_TIMER2	2 tenporizadorearen etena
6	IRQ_TIMER3	3 tenporizadorearen etena
7	IRQ_NETWORK	Serie atakaren etena
8	IRQ_DMA0	DMA0ren etena
9	IRQ_DMA1	DMA1ren etena
10	IRQ_DMA2	DMA2ren etena
11	IRQ_DMA3	DMA3ren etena
12	**IRQ_KEYS**	Teklatuaren etena
13	IRQ_CART	GBA kartutxoaren etena
14	—	
15	—	
16	IRQ_IPC_SYNC	IPCarekin sinkronizatzeko etena
17	IRQ_FIFO_EMPTY	Bidaltze FIFOa hutsik dagoen etena
18	IRQ_FIFO_NOT_EMPTY	Bidaltze FIFOa hutsik ez dagoen etena
19	IRQ_CARD	DS kartutxoaren etena
20	IRQ_CARD_LINE	eten lerroa
21	IRQ_GEOMETRY_FIFO	Geometria FIFOaren etena
22	IRQ_LID	Estalkiaren bandaren etena
23	IRQ_SPI	SPIaren etena
24	IRQ_WIFI	WIFIaren etena

Table B.4: eten-lerroen identifikazio eta definizioa

etenaren maskara eta gure zerbitzu errutinaren izena (izenetik bere helbidea lortuko du eta hori da eten-taulan gordeko duena). Adibidez, teklaturako:



maskara	@
IRQ_VBLANC	@VBLANC_errutina
IRQ_XXX	@gure_errutina

Figure B.13: NDSaren eten-taula

IrqSet(IRQ_KEY, Teklatuaren_ZE_izena)

B.2.3 Interrupt Dispatcher-a

Etenen kudeaketaren barruan, esan dugun bezala, *Interrupt Dispatcher* softwarea da etenen kudeaketarekin lotutako lan guztia egiteaz arduratzen dena.

DevkitPro inguruneko [devkitPro] *libnds* liburutegian aurki dezakegu *Interrup Dispatcher*aren kodea mihiztadura lengoaian. Kodea hori aztertuta *Interrupt Dispatcher*rak etenen kudeaketarako jarraitzen dituen pausoak ikus daitezke. Orokorrean hauek dira etenen kudeaketarako jarraitzen diren pausoak:

1. CPUak exekutatzen duen agindu bakoitzeko, exekuzio faseren batean, Interrupt Dispatcherra exekutatzen du.
2. *Interrupt Dispatcher*rak bere exekuzioan:
   (a) IME erregistroa erabiliz eten guztiak galarazten ditu,
   (b) IE eta IF erregistroak konparatzen ditu egiaztatzeko onartutako eten eskaeraren bat dagoen,
   (c) Ez badago onartutako eten eskaerarik programaren exekuzioarekin jarraitzen da,
   (d) Onartutako eten eskaeraren bat badago, eten-taulan exekutatu behar den zerbitzu errutinaren helbidea bilatzen du eta CPUko PC errerregistroan kargatzen du. Aldi berean egoera erregistroa gordetzen da zerbitzu errutinaren exekuzioaren bukaeran aurreko egoera berreskuratu ahal izateko.
3. Zerbitzu errutina exekutatzen da,
4. Zerbitzu errutinaren exekuzioa bukatzean aurreko programaren egoera berreskuratzen da.

Ondoko ondorioak atera ditzakegu aurreko azalpenetatik:

- **etenen kudeaketa modu honek ez ditu maila anitzeko etenak onartzen** Hau ondorioztatzeko *Interrupt Dispatcher*raren kodea begiratu behar da, eta ikusiko dugu, bere exekuzioaren hasieran, eten guztiak galarazten dituela. Hau dela eta, zerbitzu errutina bat exekutatzen ari den bitartean ez zaie kasurik egingo sortzen diren etenei.
- Etenen lehentasunak programatzaileak ezarriko ditu. Tratatu beharreko etenen bat dagoenean, exekutatu behar den zerbitzu-errutina bilatzen da eten-taulan. Bilaketa hori eten-taulako 0 posizioan hasten da, modu sekuentzialean aztertuz bere posizioak, eskaera

batekin bat datorren sarrera bat aurkitu arte. Beraz, eten-taulan informazioa (zerbitzu-errutinen helbideak beraien identifikazioarekin) idazteko erabili den ordenak finkatzen ditu lehentasunak, eta informazioa programatzaileak idazten du eten-taulan irqSet funtzioa erabiliz; irqSet-en ordenak ezartzen ditu etenen arteko lehentasunak.

Ezaugarri hauek desberdinak izateko nahikoa da *Interrupt Dispatcher*ra beste modu batean idaztea.

APPENDIX C

Source-code of one of the projects

This appendix shows the source-code of the project explained in section 5.3.3. It was developped by Asier Santos, Markel Sanz, and Alejandro Reyes.

In order to develop the project, students got a folder with all the code and header files they would need. This code they get can be directly compiled. It is a small version of the project they have to develop. A closed door that opens when touching the screen at any point. Any further control of the touch screen or any other peripheral is what the students have to program. What they have to do is already stablished because they get the names and specification of all the routines they have to code, but they are all empty, and this is the work they have to do.

Next in this appendix, figures will be showing different parts of the project, while text will be explaining what was given and what was the students work.

C.1 The main program

Figure C.1 shows only the part that students had to code, the main loop of the program. Previous to that, the code would have shown all the definitions needed to have the graphics work correctly. These definitions are given to the students.

As for the routines they have to use to show backgrounds on the screen, they are called `erakutsiXXX();`. For example, `erakutsiAteaGorri()` shows the door with a red light, while `erkutsiAteaItxita()` shows the door closed. They get an example and they build the rest.

In that main loop it is easy to distinguish the states of the state machine they have designed and that can be seen in Figure 5.6 (translated into English). The state named Correct in Figure 5.6, comprises the states "BatOndo", "BiOndo", and "HiruOndo" of the code, since the only difference between them is the amount of correct fuses.

In every state except in *"LauOndo"* (win), the touch screen is controlled and do different things when different parts of the screen are touched. In state *"LauOndo"* (win) the keyboard is controlled by polling.

C.2 Defining the Interrupt-table

Students get the routine in Figure C.2 empty and they fill it with the IRQ settings needed. In this case, the IRQ lines and ISR names for Timer0 and the keyboard.

C.3 The keyboard

In order to control the keyboard students get a set of routines, all of them empty, with a simple explanation of the aim of the routine. For example the routine in Figure C.3 is to detect whether a key has been pressed, and the on in Figure C.8 is to configure the functioning of the keyboard.

C.4 The timer

The same way as for the keyboard, also for the timer students get a set of empty routines with its definition. For example the routine in Figure C.10 should enable the timer's interrupts. In this case students have add a few routines that were not required (see code in Figures C.12 and C.13, in order to start and stop the timer).

```
        bool amaitua = false;

        while(amaitua == false) {//Closed door in Figure 5.6
                if(egoera == Itxita) {
                        erakutsiAteaItxita();
                        touchRead(&pos_pantaila);
                        while(pos_pantaila.px==0 &&
                            pos_pantaila.py==0)
                                touchRead(&pos_pantaila);
                        if (pos_pantaila.px < 212 &&
                            pos_pantaila.px > 136 &&
                            pos_pantaila.py < 192 &&
                            pos_pantaila.py > 42) {
                                egoera = ItxitaGorri;
                                    //Red light in Figure 5.6
                                erakutsiAteaGorri();
                                consoleClear();
                                iprintf("\x1b[10;4HEz dago
                                    elektrizitaterik!");
                        }
                }
                if(egoera == ItxitaGorri) {
                        //Red light in Figure 5.6
                        touchRead(&pos_pantaila);
                        if (pos_pantaila.px < 97 &&
                            pos_pantaila.px > 46 &&
                            pos_pantaila.py < 118 &&
                            pos_pantaila.py > 72) {
                                while (pos_pantaila.px != 0 &&
                                    pos_pantaila.py != 0) {
                                        touchRead(&pos_pantaila);
                                }
                                erakutsiZeroOndo();
                                egoera = ZeroOndo;
                                //Fuses in Figure 5.6
                                denb = 20;
                                consoleClear();
                                iprintf("\x1b[9;3HGelditzen
                                    zaizun denbora:");
                                iprintf("\x1b[11;10H %i
                                    segundo", denb);
                                tenporizadoreaHasi();
                        }
                }
```

Figure C.1: The main program of one project. Code continues

```
if(egoera == ZeroOndo) {//Fuses in Figure 5.6
        while (pos_pantaila.px == 0 &&
            pos_pantaila.py == 0 && egoera ==
            ZeroOndo)
                touchRead(&pos_pantaila);
        if (pos_pantaila.px < 246 &&
            pos_pantaila.px > 193 &&
            pos_pantaila.py < 157 &&
            pos_pantaila.py > 90 && egoera ==
            ZeroOndo) {
                while (pos_pantaila.px != 0 &&
                    pos_pantaila.py != 0)
                        touchRead(&pos_pantaila);
                egoera = BatOndo;
                    //Correct in Figure 5.6
                erakutsiBatOndo();
        }else if (egoera == ZeroOndo){
                egoera = Trantsizioa;
                    //Wrong in Figure 5.6
                tenporizadoreaGelditu();
                consoleClear();
                iprintf("\x1b[9;8HEtengailu
                    okerra!");
                iprintf("\x1b[11;2HSakatu ’B’
                    atera bueltatzeko");
        }
}
if(egoera == BatOndo) {//Correct in Figure 5.6
        while (pos_pantaila.px == 0 &&
            pos_pantaila.py == 0 && egoera ==
            BatOndo)
                touchRead(&pos_pantaila);
        if (pos_pantaila.px < 185 &&
            pos_pantaila.px > 133 &&
            pos_pantaila.py < 157 &&
            pos_pantaila.py > 90 && egoera ==
            BatOndo) {
                while (pos_pantaila.px != 0 &&
                    pos_pantaila.py != 0)
                        touchRead(&pos_pantaila);
                egoera = BiOndo;
                //Correct in Figure 5.6
                erakutsiBiOndo();
```

Figure C.1: The main program of one project. Code continued and continues

```
               }else if (egoera == BatOndo){
                       //Correct in Figure 5.6
                       egoera = Trantsizioa;
                       // Wrong in Figure 5.6
                       tenporizadoreaGelditu();
                       consoleClear();
                       iprintf("\x1b[9;8HEtengailu
                          okerra!");
                       iprintf("\x1b[11;2HSakatu 'B'
                          atera bueltatzeko");
               }
        }
        if(egoera == BiOndo) { //Correct in Figure 5.6
               while (pos_pantaila.px == 0 &&
                   pos_pantaila.py == 0 && egoera ==
                   BiOndo)
                       touchRead(&pos_pantaila);
               if (pos_pantaila.px < 59 &&
                   pos_pantaila.px > 6 &&
                   pos_pantaila.py < 157 &&
                   pos_pantaila.py > 90 && egoera ==
                   BiOndo) {
                       while (pos_pantaila.px != 0 &&
                          pos_pantaila.py != 0)
                              touchRead(&pos_pantaila);
                       egoera = HiruOndo;
                          //Correct in Figure 5.6
                       erakutsiHiruOndo();
               }else if (egoera == BiOndo){
                       egoera = Trantsizioa;
                          //Wrong in Figure 5.6
                       tenporizadoreaGelditu();
                       consoleClear();
                       iprintf("\x1b[9;8HEtengailu
                          okerra!");
                       iprintf("\x1b[11;2HSakatu 'B'
                          atera bueltatzeko");
               }
        }
```

Figure C.1: The main program of one project. Code continued and continues

```
if(egoera == HiruOndo) {
       //Correct in Figure 5.6
       while (pos_pantaila.px == 0 &&
           pos_pantaila.py == 0 && egoera ==
           HiruOndo)
              touchRead(&pos_pantaila);
       if (pos_pantaila.px < 122 &&
           pos_pantaila.px > 69 &&
           pos_pantaila.py < 157 &&
           pos_pantaila.py > 90 && egoera ==
           HiruOndo) {
              tenporizadoreaGelditu();
              while (pos_pantaila.px != 0 &&
                  pos_pantaila.py != 0)
                     touchRead(&pos_pantaila);
              egoera = LauOndo;
                  //Win in Figure 5.6
              consoleClear();
              iprintf("\x1b[9;10HLortu duzu!");
              iprintf("\x1b[11;2HSakatu 'A'
                  atera bueltatzeko");
              erakutsiLauOndo();
       }else if (egoera == HiruOndo){
              egoera = Trantsizioa;
                  //Wrong in Figure 5.6
              tenporizadoreaGelditu();
              consoleClear();
              iprintf("\x1b[9;9HEtengailu
                  okerra!");
              iprintf("\x1b[11;2HSakatu 'B'
                  atera bueltatzeko");
       }
}

if(egoera == LauOndo) {      //Win in Figure 5.6
       if (IrakurriTeklatuInkesta() == A) {
              consoleClear();
              egoera = ItxitaBerde;
              //Green light in Figure 5.6
              erakutsiAteaBerde();
              iprintf("\x1b[10;3HElektrizitatea
                  bueltatu da.");
       }
}
```

Figure C.1: The main program of one project. Code continued and continues

```
              if(egoera == ItxitaBerde) {
                     //Green light in Figure 5.6
                     touchRead(&pos_pantaila);
                     while(pos_pantaila.px==0 &&
                         pos_pantaila.py==0)
                            touchRead(&pos_pantaila);
                     if (pos_pantaila.px < 212 &&
                         pos_pantaila.px > 136
                                   && pos_pantaila.py < 192
                                       && pos_pantaila.py >
                                       42) {
                            erakutsiAteaIrekita();
                            consoleClear();
                            iprintf("\x1b[10;10HLortu
                                duzu!");
                            egoera = Irekita;
                                //Open door in Figure 5.6
                     }
              }

              if(egoera == Irekita) {
                     //Open door in Figure 5.6
                     amaitua = true;
              }

              if(egoera == Trantsizioa) {
              //Wrong in Figure 5.6
              }

       }
```

Figure C.1: The main program of one project. Code continued

```
void etenak()
{
       irqSet(IRQ_TIMER0, DenbEten);
       irqSet(IRQ_KEYS, TekEten);
}
```

Figure C.2: Code to define the interrupt-table

```
int TeklaDetektatu()
{
        //TRUE itzultzen du teklaren bat sakatu dela
        //detektatzen badu
        int tekla=TEKLAK_DAT;
        bool aurkitua=false;
        while(tekla>=0 && aurkitua==false){
                        if((tekla % 2)==0)
                        {
                        aurkitua=true;
                        }
                        tekla=tekla/2;
                }
        return aurkitua;
}
```

Figure C.3: Routine that returns true if the keyboard has been pressed

```
int SakatutakoTekla()
{
        //Sakatutako teklaren balioa itzultzen du:
        //A=0;B=1;Select=2;Start=3;Esk=4;Ezk=5;
        //Gora=6;Behera=7;R=8;L=9;
        int tekla=TEKLAK_DAT;
        bool aurkitua=false;
        int i=-1;
        while(tekla>=0 && aurkitua==false)
        {
                if((tekla % 2)==0)
                {
                aurkitua=true;
                }
                tekla=tekla/2;
                i++;
        }
        return i;
}
```

Figure C.4: Routine that returns which key has been pressed

```
int IrakurriTeklatuInkesta()
{
        //inkesta tekla bat sakatu dela detektatu arte
        //teklaren balioa itzuli
        while(TeklaDetektatu()==false)
        {
                TeklaDetektatu();
        }
        return SakatutakoTekla();
}
```

Figure C.5: Routine that polls the keyboard and returns the pressed key

```
void TekEtenBaimendu()
{
        //Teklatuaren etenak baimendu
        //Lan hau burutzeko lehenengo eten guztiak galarazi
        //behar dira eta bukaeran baimendu
        EtenakGalarazi();
        IE=IE|0x1000;
        EtenakBaimendu();
}
```

Figure C.6: Routine to enable keyboard interrupts

```
void TekEtenGalarazi()
{
        //Teklatuaren etenak galarazi
        //Lan hau burutzeko lehenengo eten guztiak galarazi
        //behar dira eta bukaeran baimendu
        EtenakGalarazi();
        IE=IE&0xEFFF;
        EtenakBaimendu();
}
```

Figure C.7: Routine to disable keyboard interrupts

```
void konfiguratuTeklatua()
{
       // Teklatuaren konfigurazioa bere S/I erregistroak
       // aldatuz.
       TEKLAK_KNT=0x4002;
       TekEtenBaimendu();
}
```

Figure C.8: Routine to configure keyboard

```
void TekEten()
{
       //Teklatuaren zerbitzu errutina, teklatuaren etenak
       //tratatzeko.
       tekla=SakatutakoTekla();
       if (tekla==B && egoera == Trantsizioa)
       //Wrong in Figure 5.6
       {
              consoleClear();
              iprintf("\x1b[10;7HSaia zaitez berriz");
              egoera=Itxita;        //Closed door in Figure
                  5.6
              erakutsiAteaItxita();
       }
}
```

Figure C.9: ISR of the keyboard

```
void DenbEtenBaimendu()
{
//Denboragailu baten etenak baimendu (Timer0)
//Horretarako lehenengo eten guztiak galarazi eta bukaeran
//berriro baimendu
       EtenakGalarazi();
       IE=IE|0x0008;
       EtenakBaimendu();
}
```

Figure C.10: Routine to enable timer interrupts

```
void DenbEtenGalarazi()
{
//Denboragailu baten etenak galarazi (Timer0)
//Horretarako lehenengo eten guztiak galarazi eta bukaeran
//berriro baimendu
       EtenakGalarazi();
       IE=IE&0xFFF7;
       EtenakBaimendu();
}
```

Figure C.11: Routine to disable timer interrupts

```
void tenporizadoreaHasi()
{
       //Denboragailua aktibatu kontatzen has dadin
       DENBO_KNT = DENBO_KNT | 0x0080;
}
```

Figure C.12: Routine to start timer

```
void tenporizadoreaGelditu()
{
       //Denboragailua desaktibatu kontatzen geldi dadin
       DENBO_KNT = DENBO_KNT & 0xFF7F;
}
```

Figure C.13: Routine to stop timer

```
void konfiguratuTenporizadorea()
{
       /* Tenporizadorearen konfigurazioa bere S/I
           erregistroak aldatuz */
       DENBO_KNT = 0x0043;
       DENBO_DAT = 32764;
       DenbEtenBaimendu();
}
```

Figure C.14: Routine to configure timer

```
void DenbEten()
{
        //Denboragailuaren (Timer0) etenaren tratamendurako
        //zerbitzu errutina
        if (denb > 0) {
                denb--;
                consoleClear();
                iprintf("\x1b[9;3HGelditzen zaizun denbora:");
                iprintf("\x1b[11;10H %i segundo", denb);
        }
        else{
                egoera=Trantsizioa;
                //Timeout in Figure 5.6
                tenporizadoreaGelditu();
                consoleClear();
                iprintf("\x1b[9;7HDenbora agortu da!");
                iprintf("\x1b[11;2HSakatu 'B' atera
                    bueltatzeko");
        }
}
```

Figure C.15: ISR of the timer

APPENDIX D

Students' concept maps

This appendix shows the concept maps generated after the concept maps that the students last built at the end of course 2012/2013, with the adaptations needed for their use with the social network analysis tools. Images are in Basque, the language in which lectures are given. The process followed to analyse the concept maps is detailed in Chapter 7.

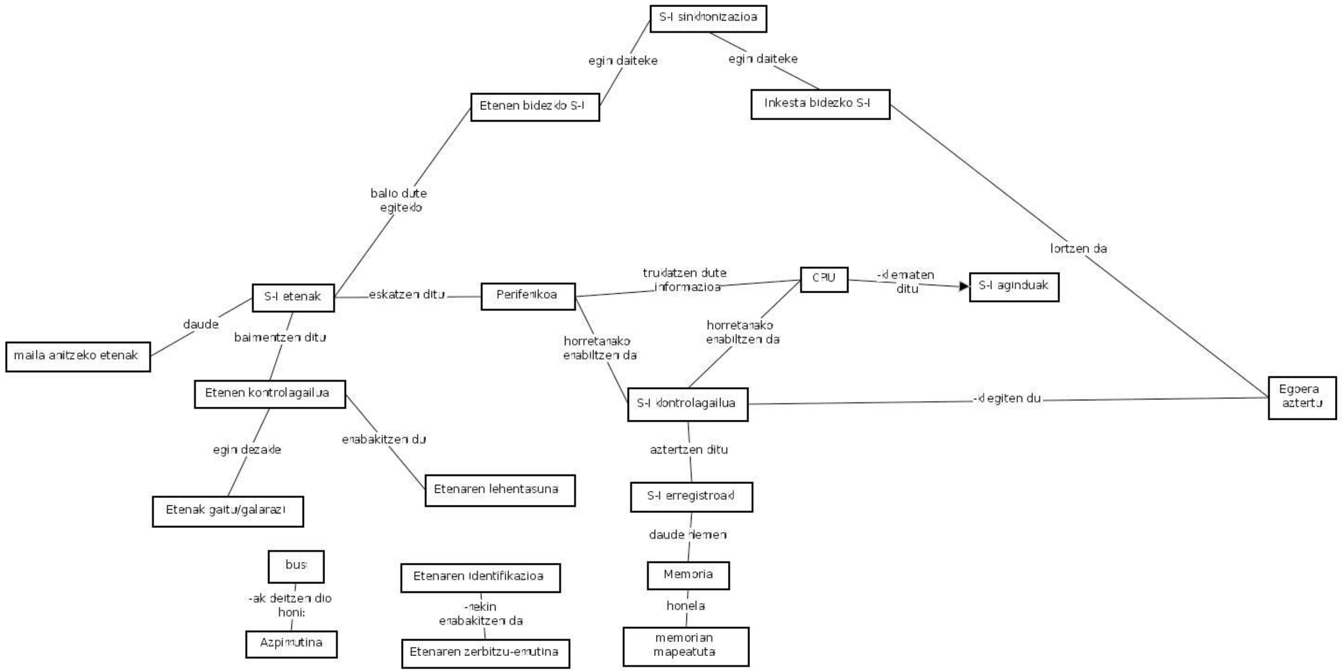


Figure D.1: Student concept map #1

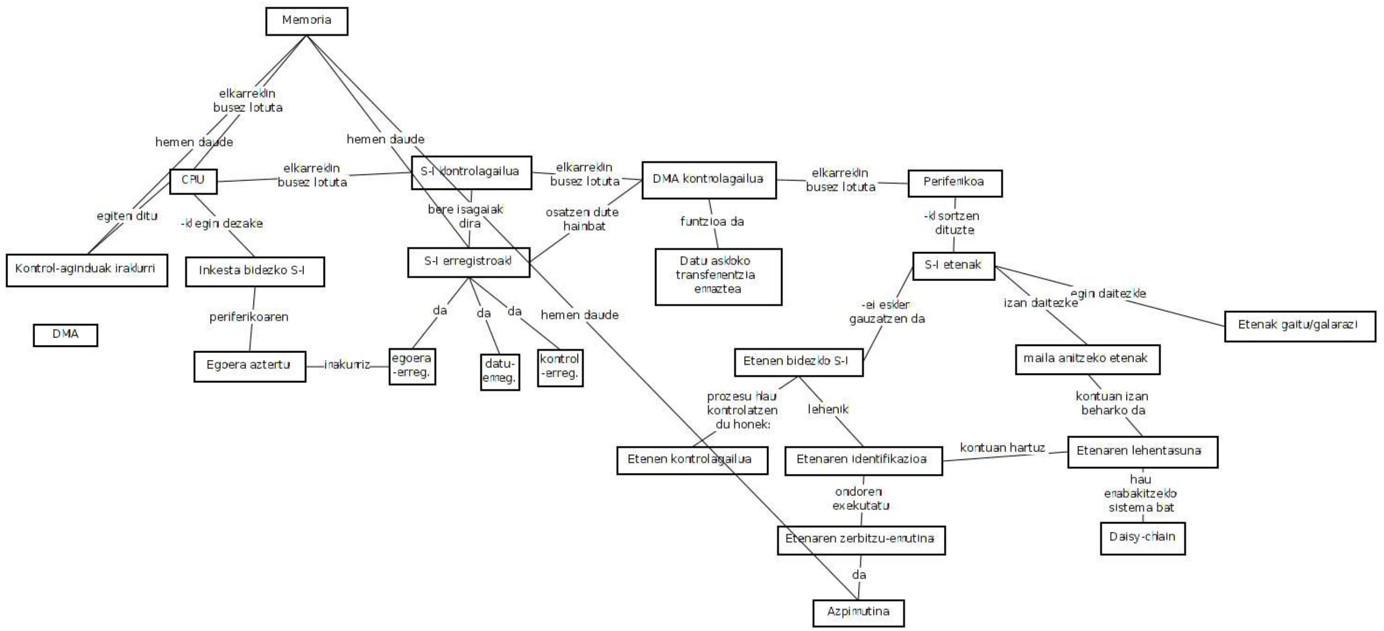


Figure D.2: Student concept map #2

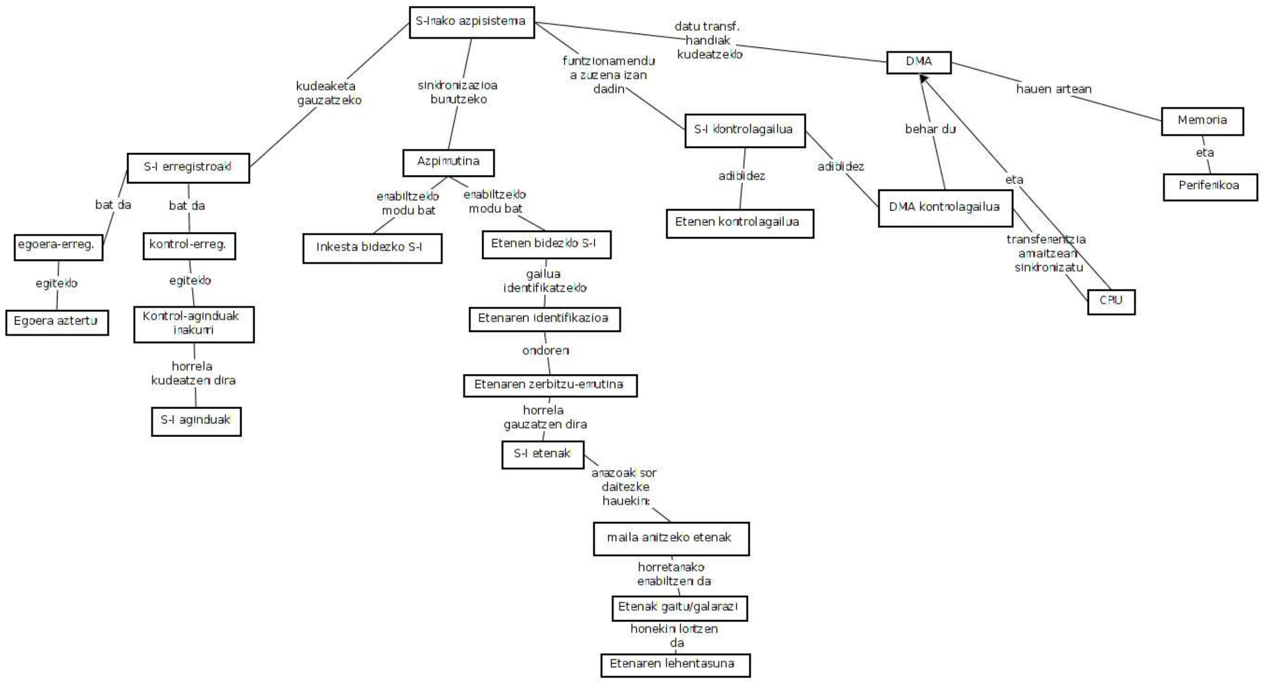


Figure D.3: Student concept map #3

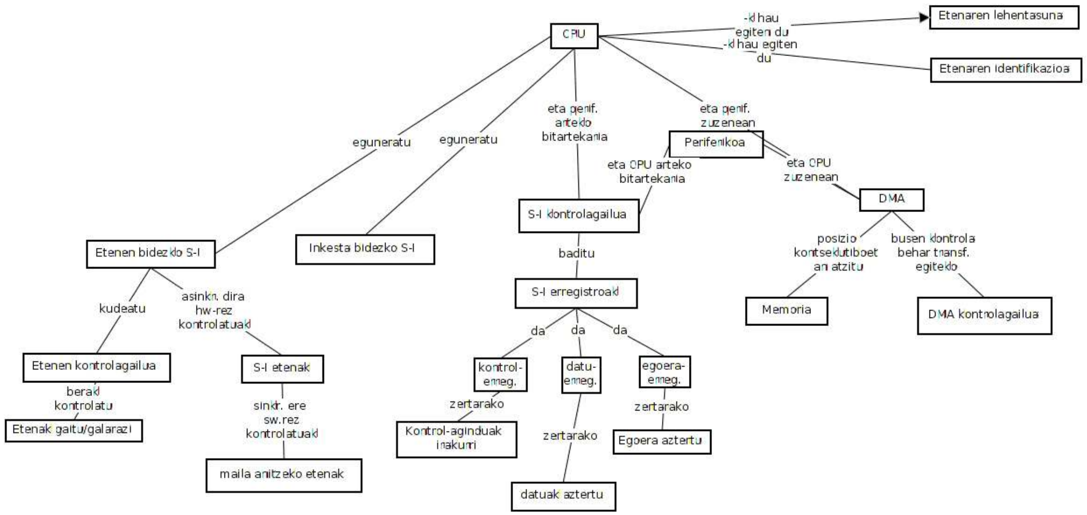


Figure D.4: Student concept map #4

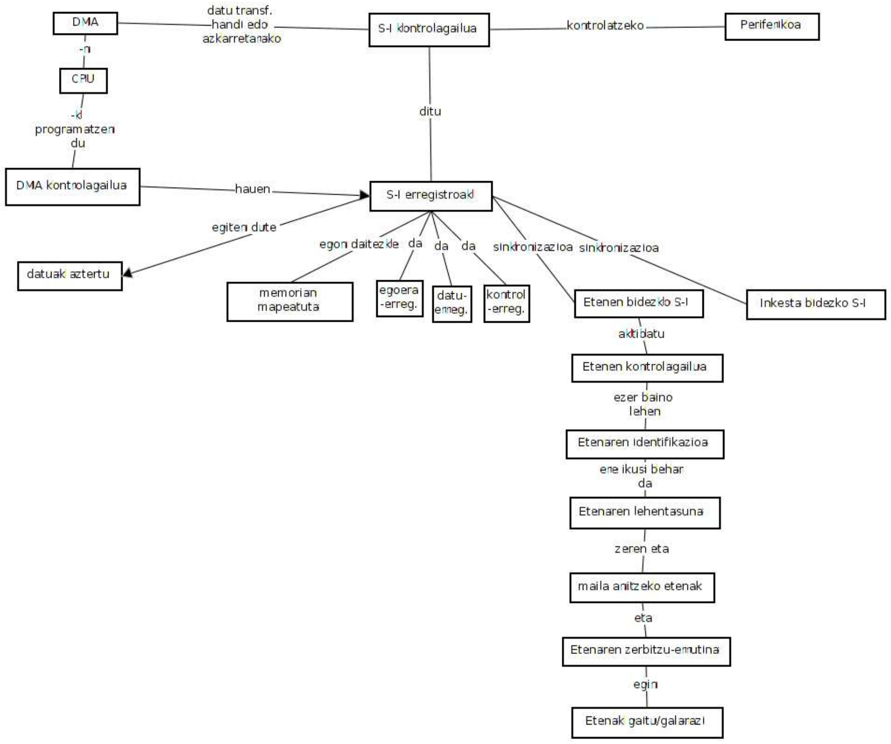


Figure D.5: Student concept map #5

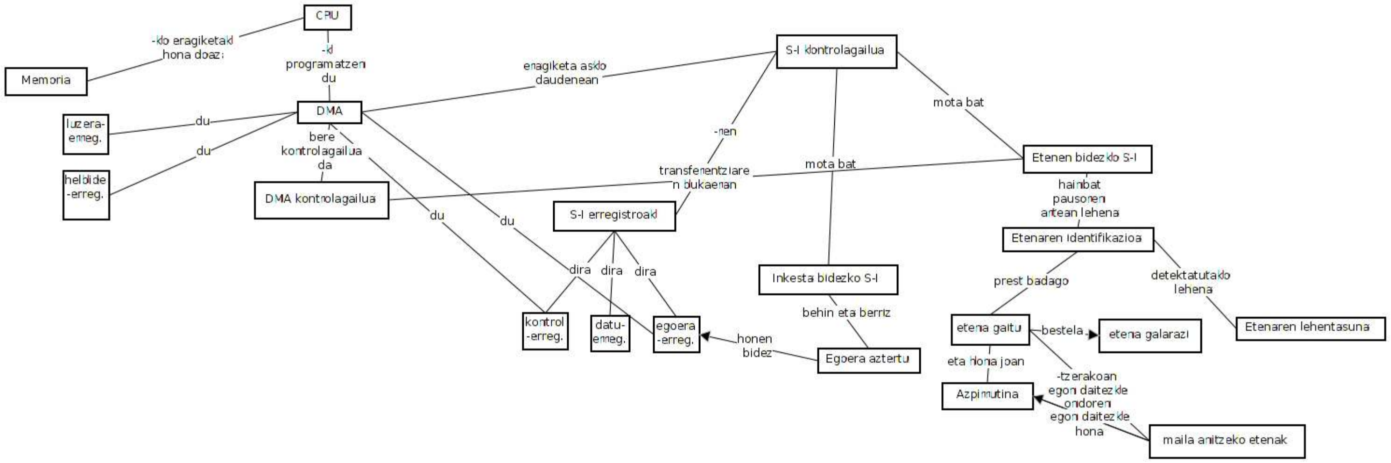


Figure D.6: Student concept map #6

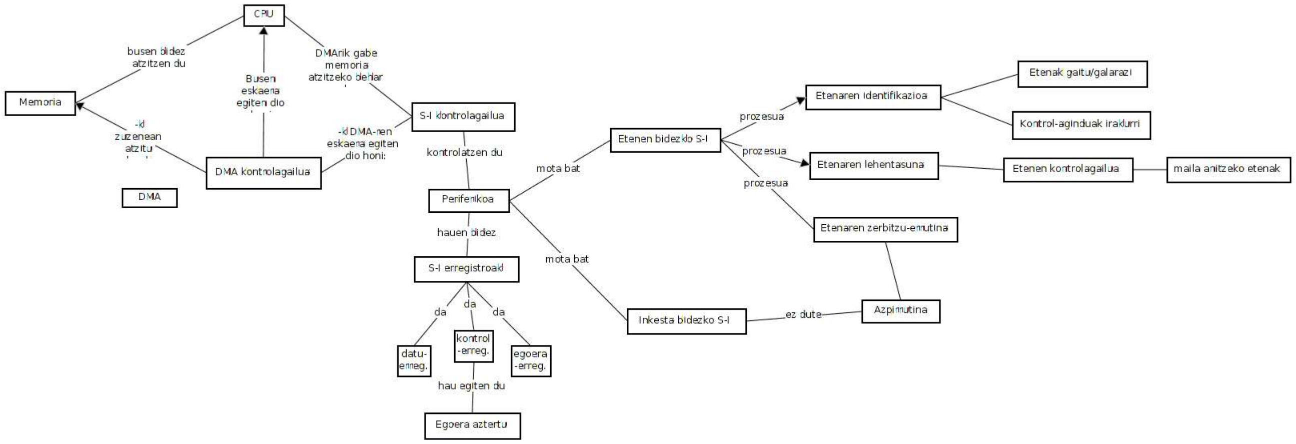


Figure D.7: Student concept map #7

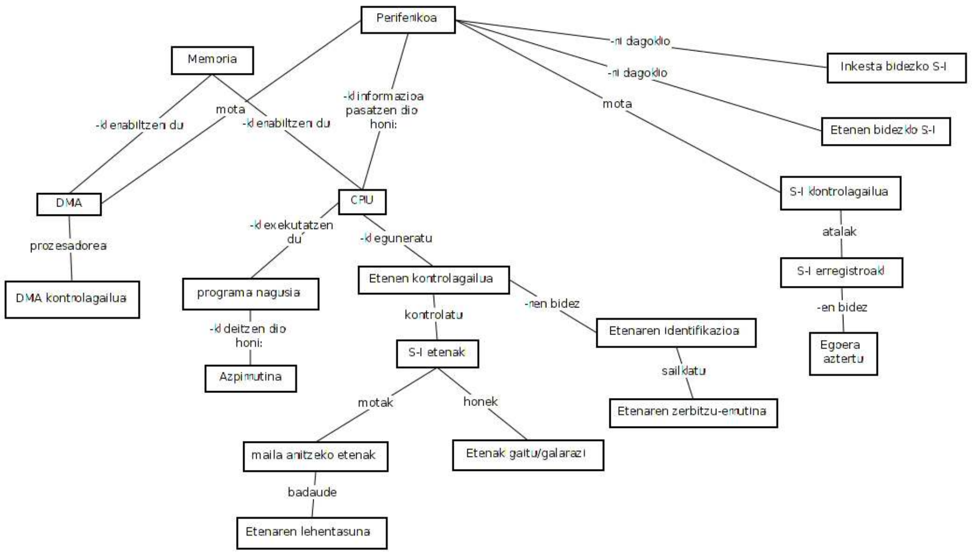


Figure D.8: Student concept map #8

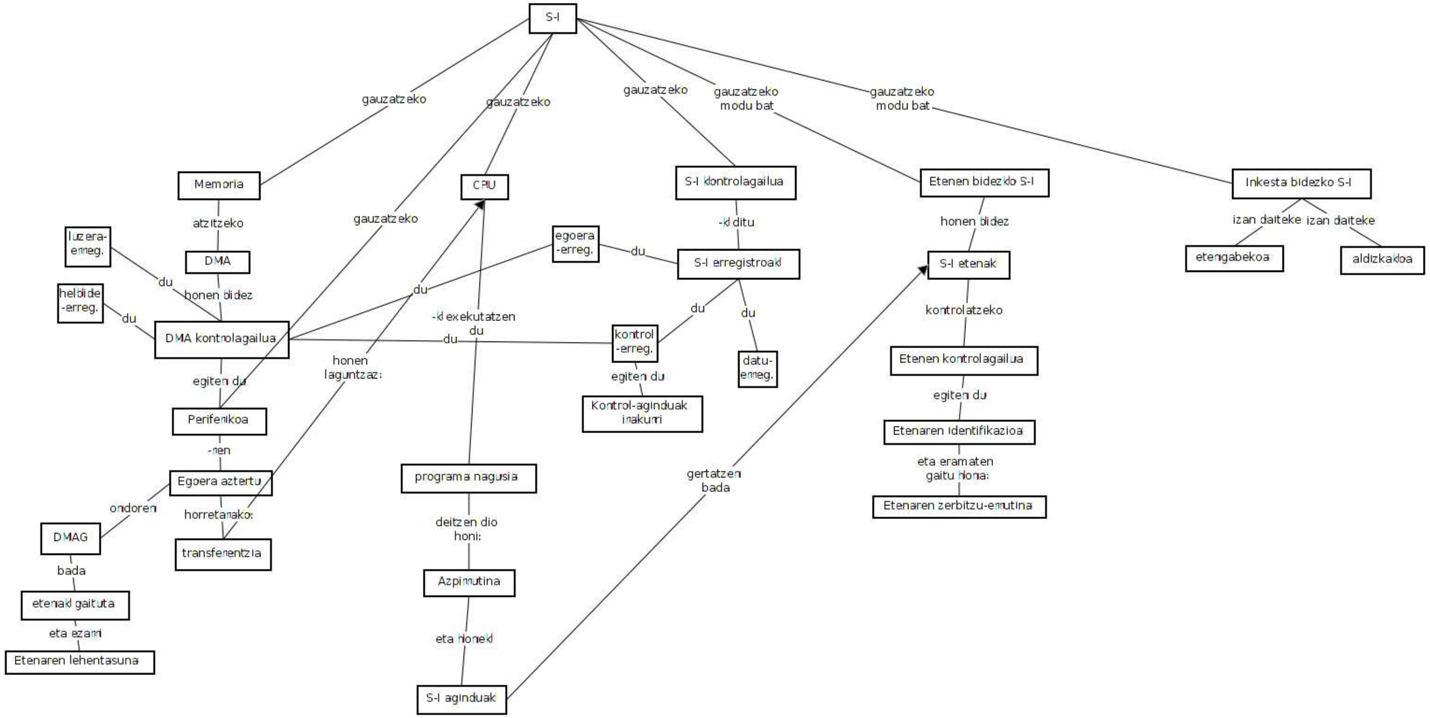


Figure D.9: Student concept map #9

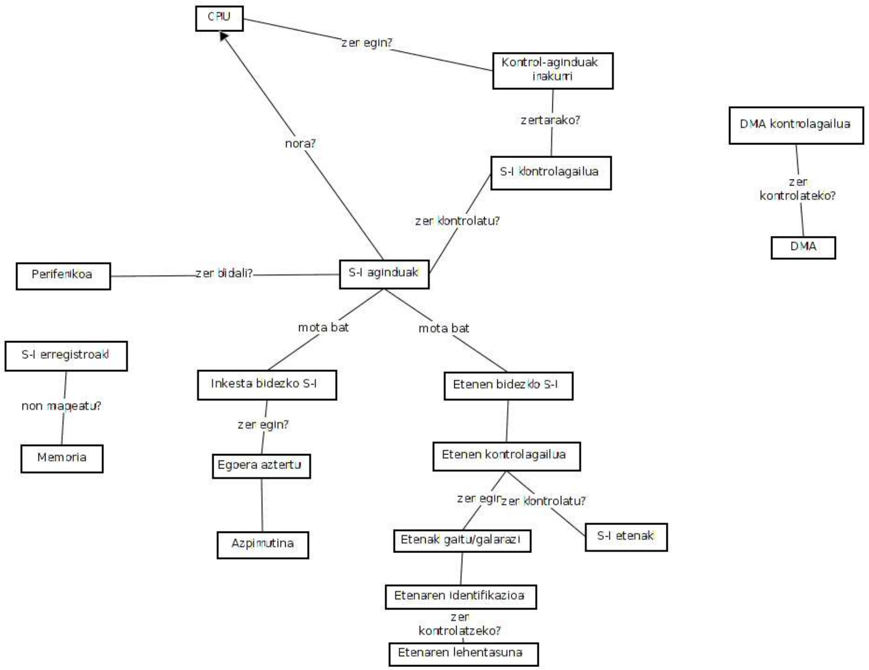


Figure D.10: Student concept map #10

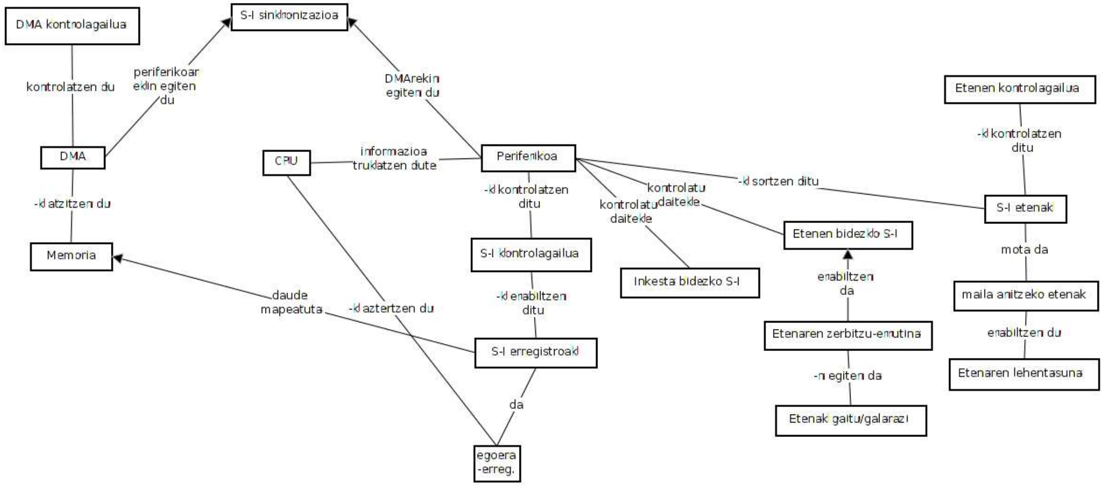


Figure D.11: Student concept map #11

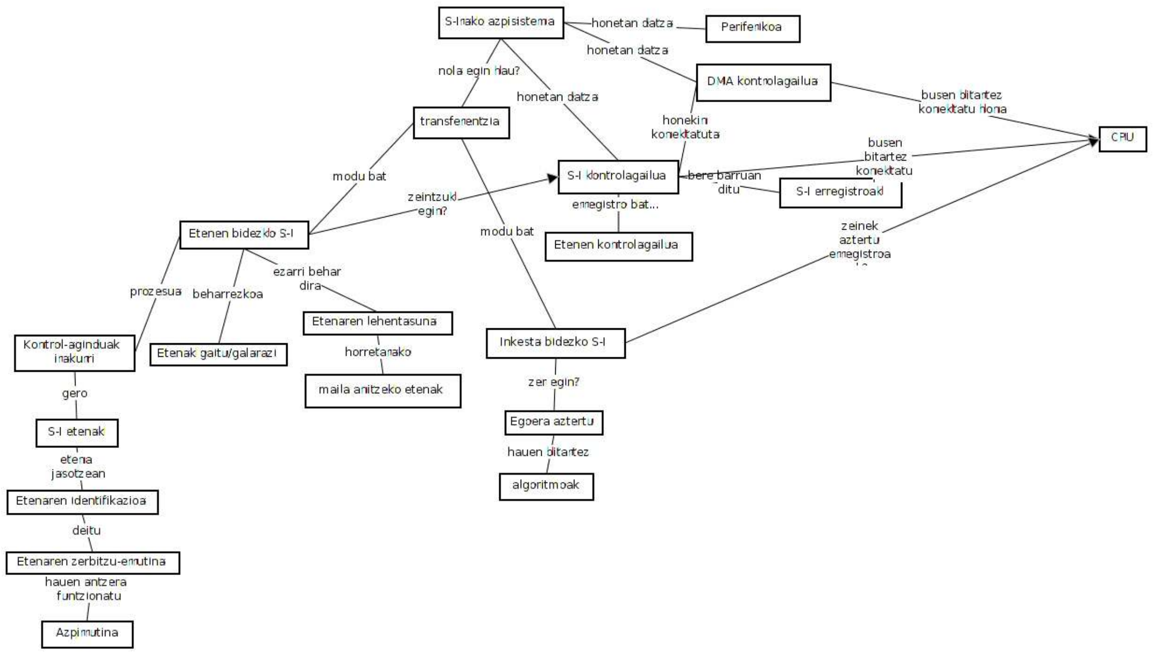


Figure D.12: Student concept map #12

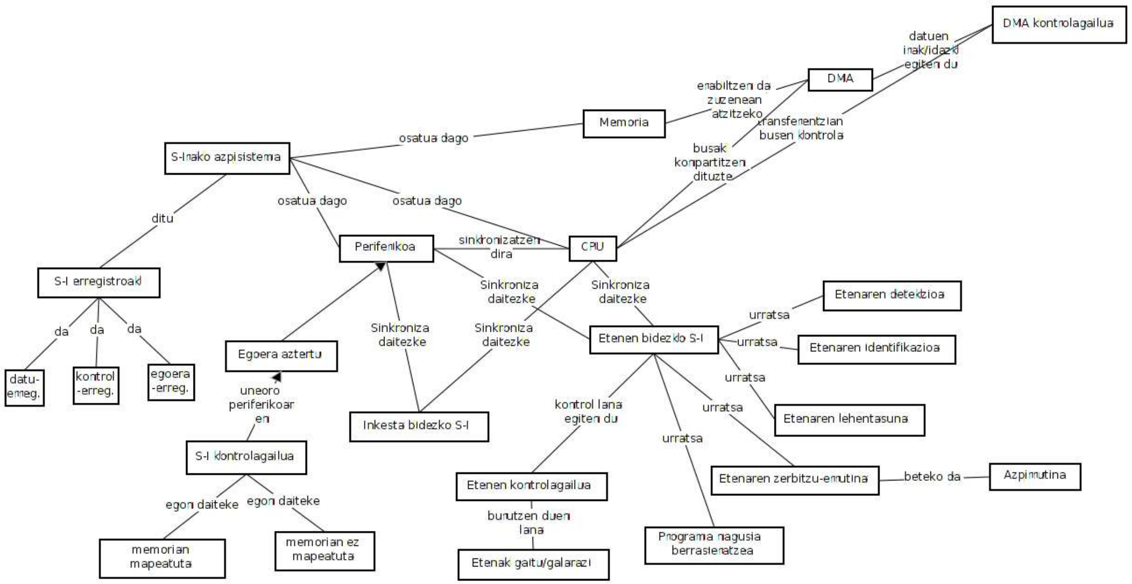


Figure D.13: Student concept map #13

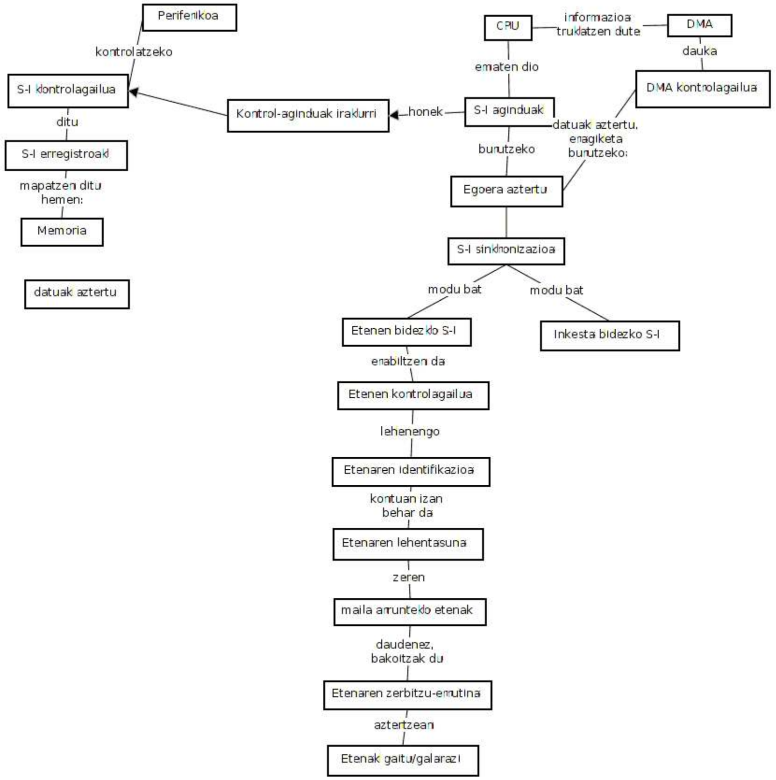


Figure D.14: Student concept map #14

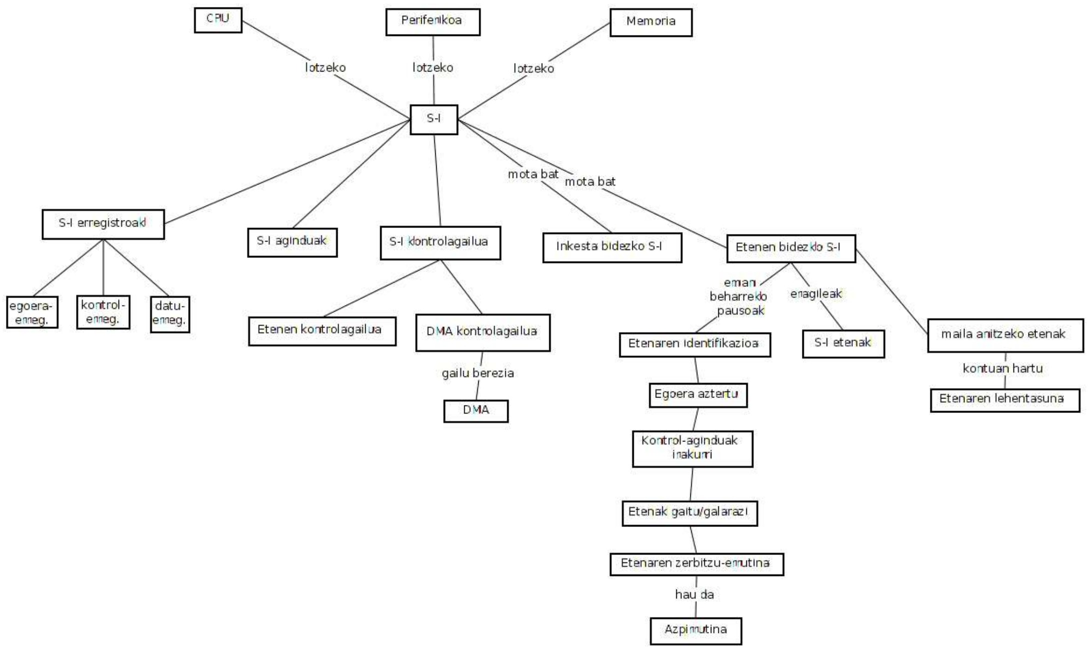


Figure D.15: Student concept map #15

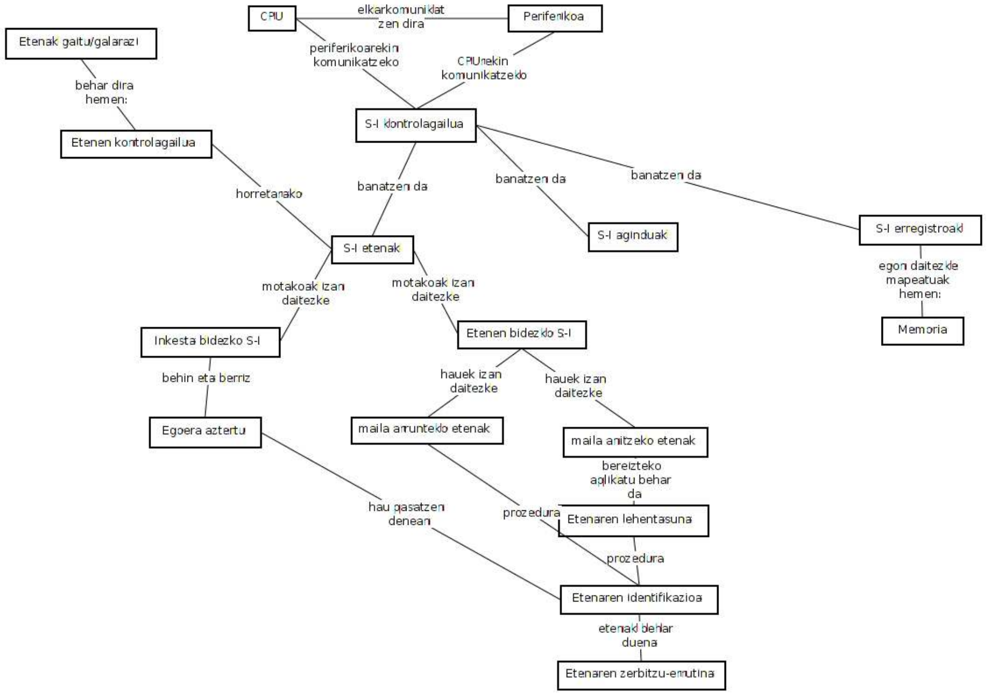


Figure D.16: Student concept map #16

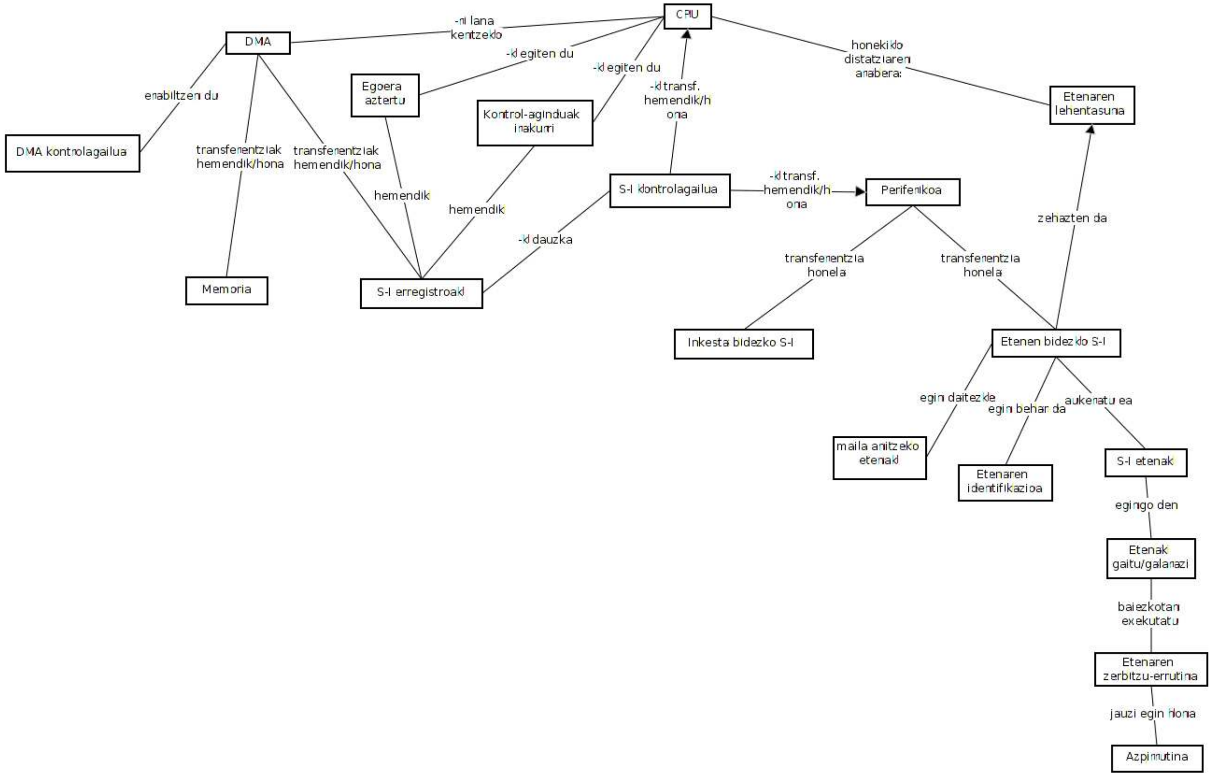


Figure D.17: Student concept map #17

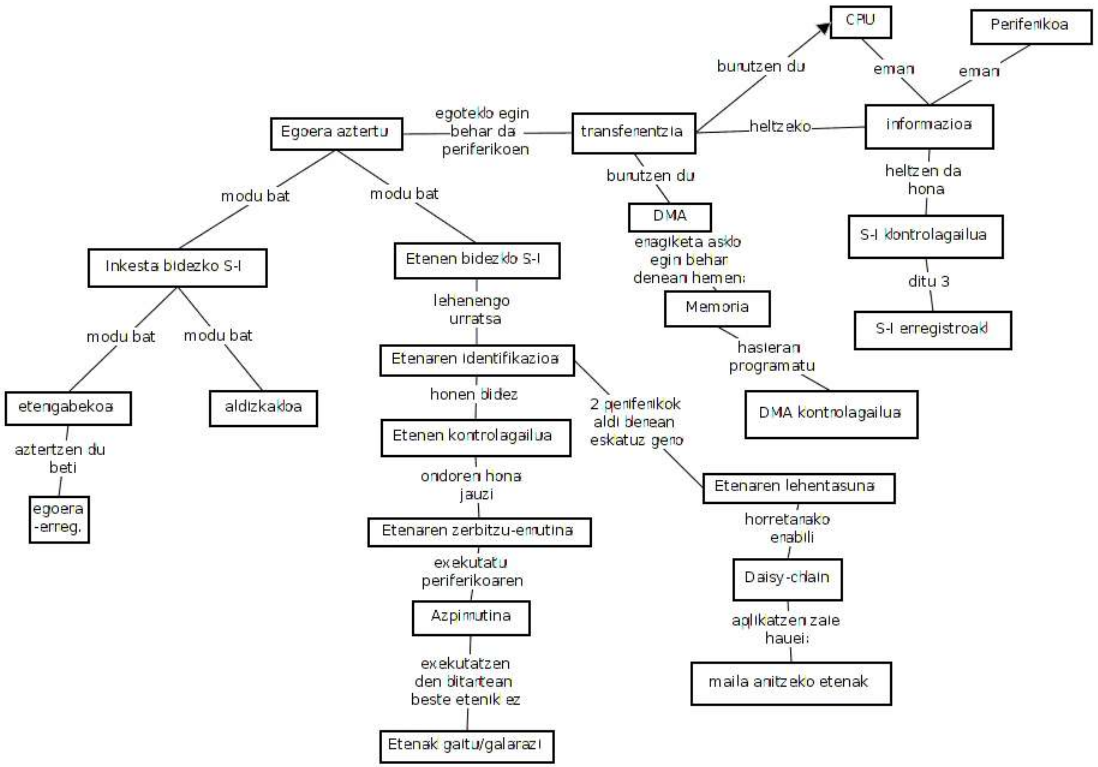


Figure D.18: Student concept map #18

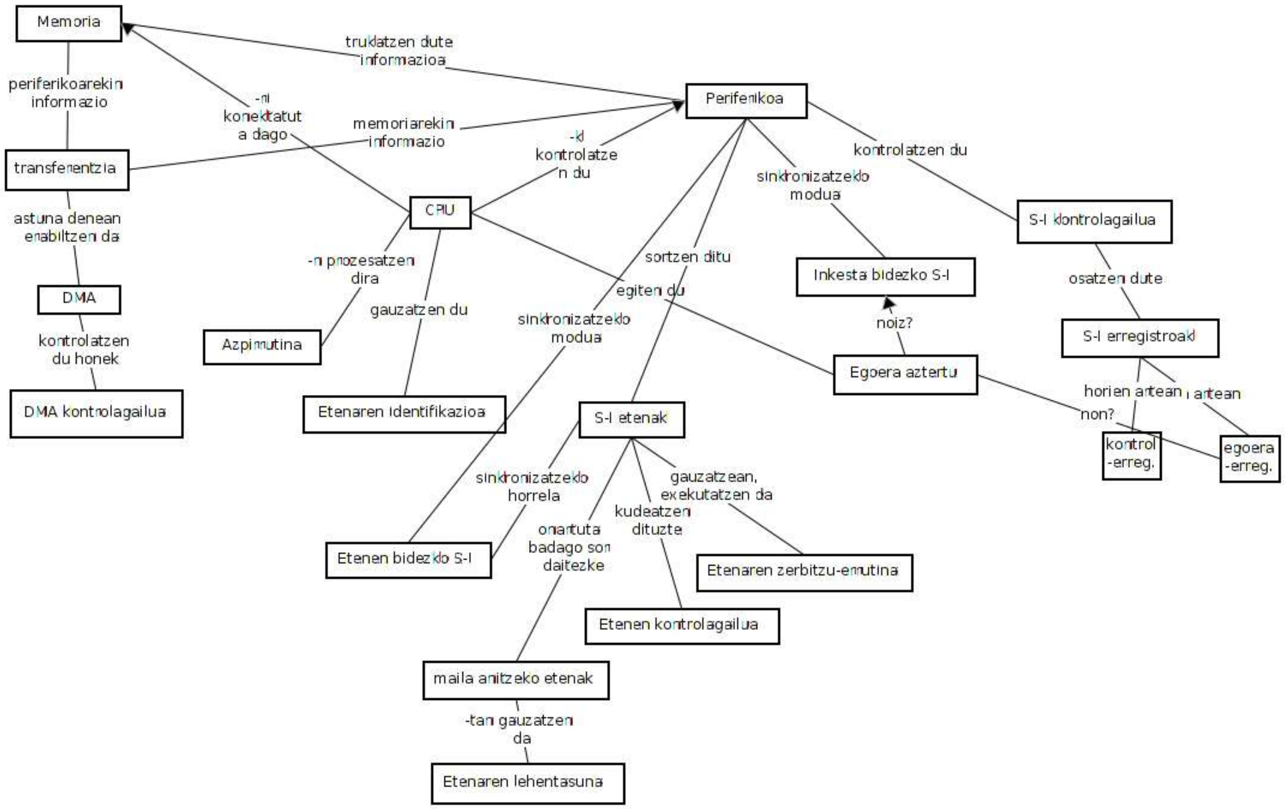


Figure D.19: Student concept map #19

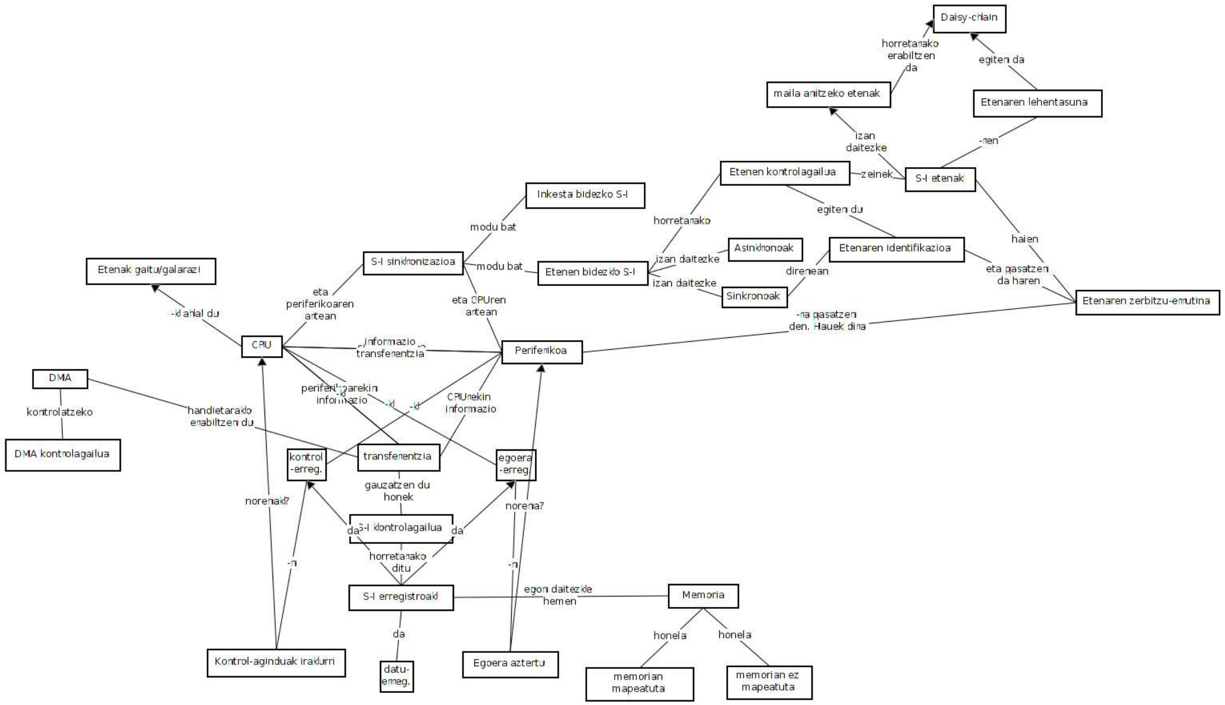


Figure D.20: Student concept map #20

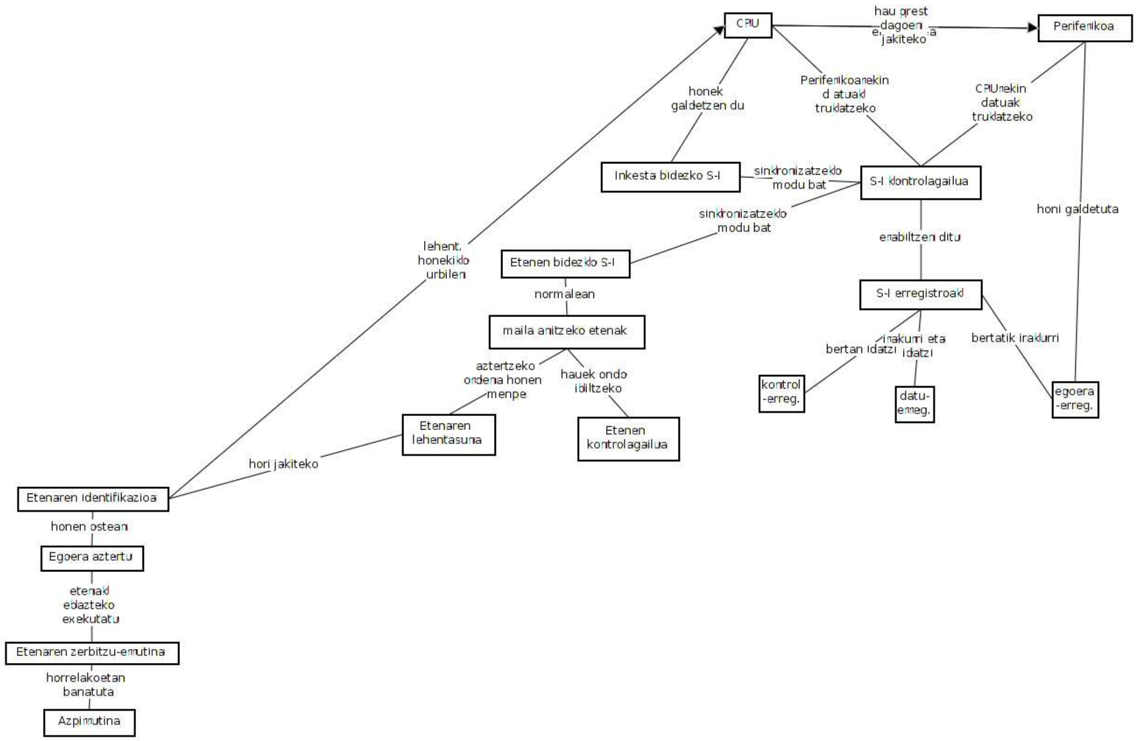


Figure D.21: Student concept map #21

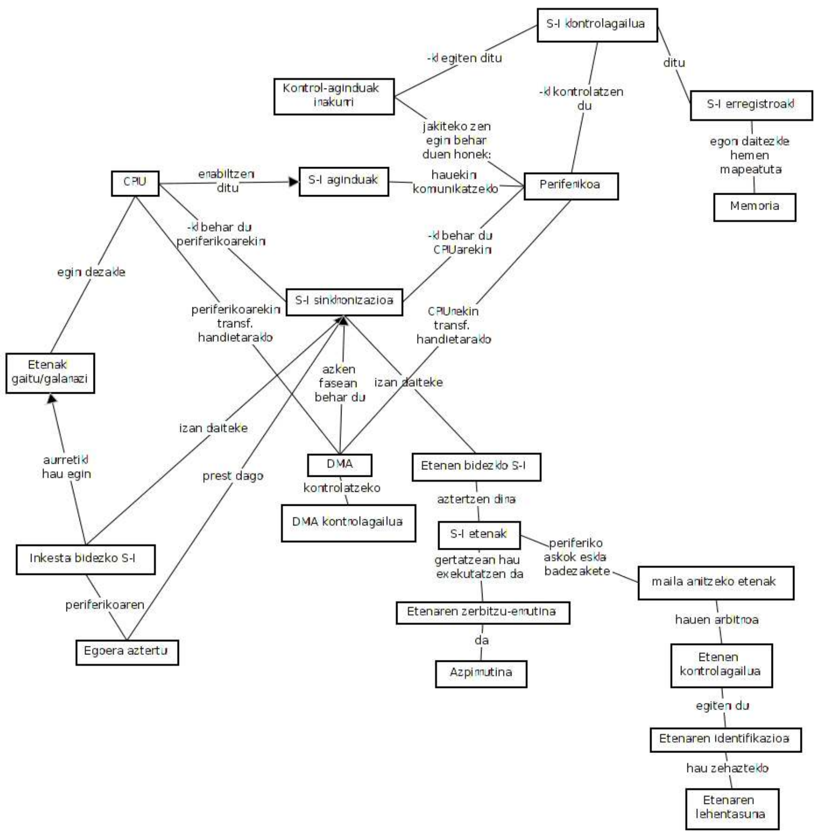


Figure D.22: Student concept map #22

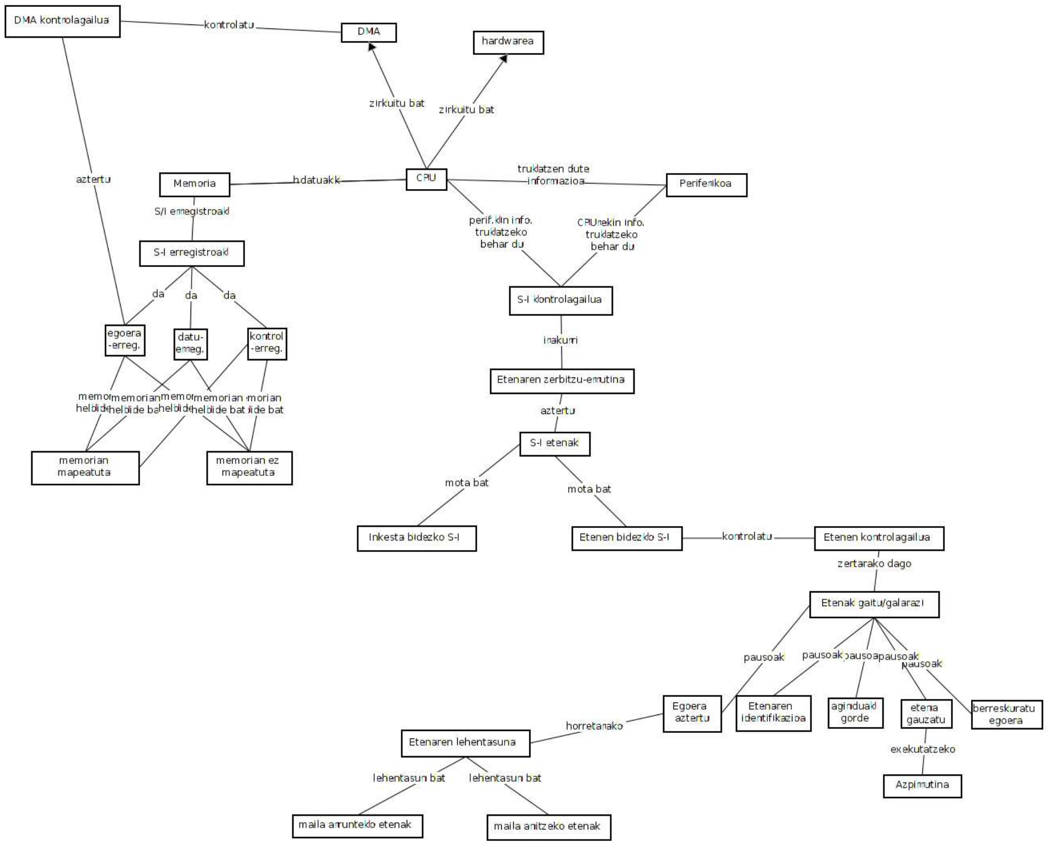


Figure D.23: Student concept map #23

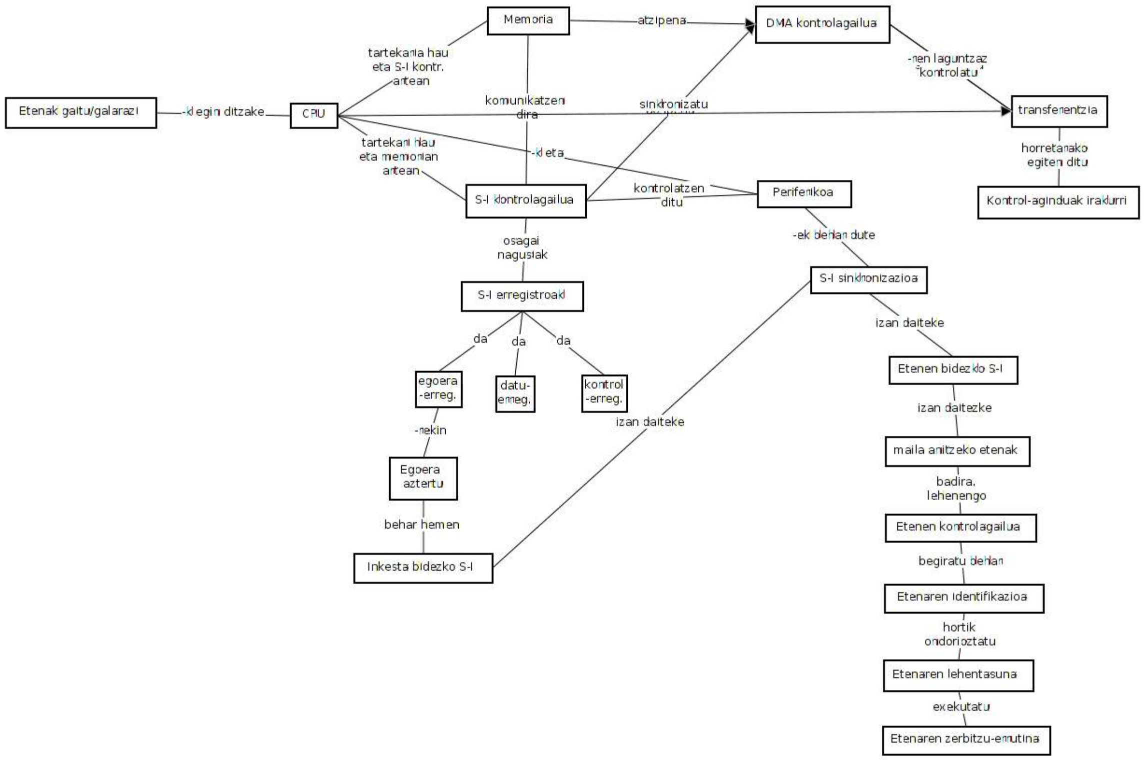


Figure D.24: Student concept map #24

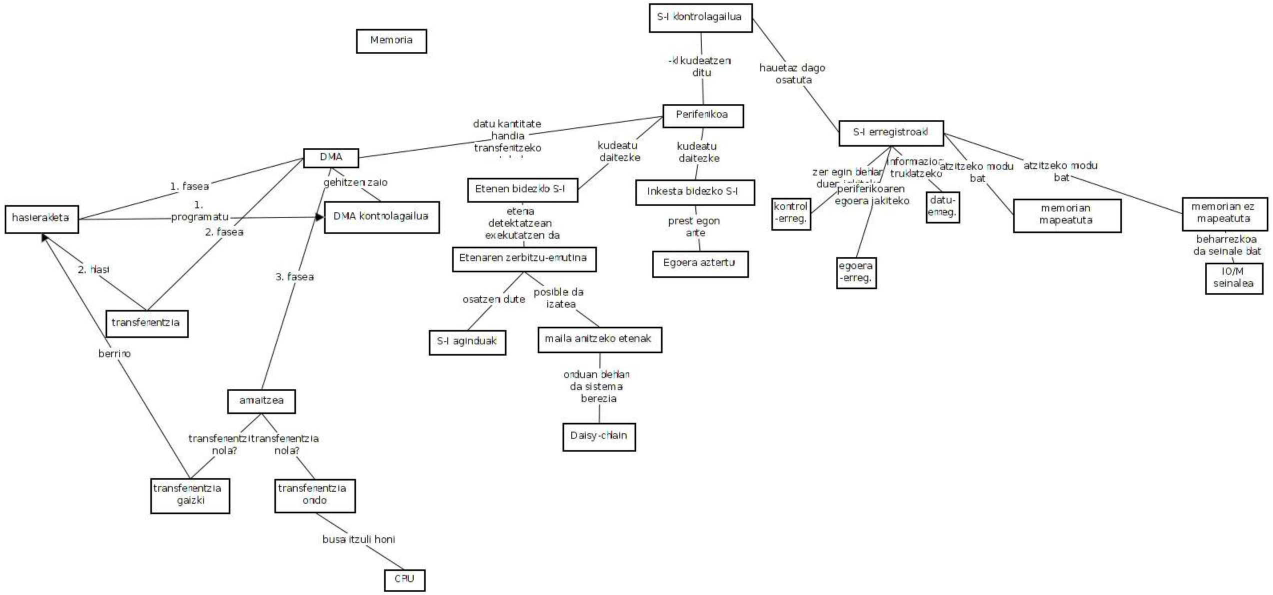


Figure D.25: Student concept map #25

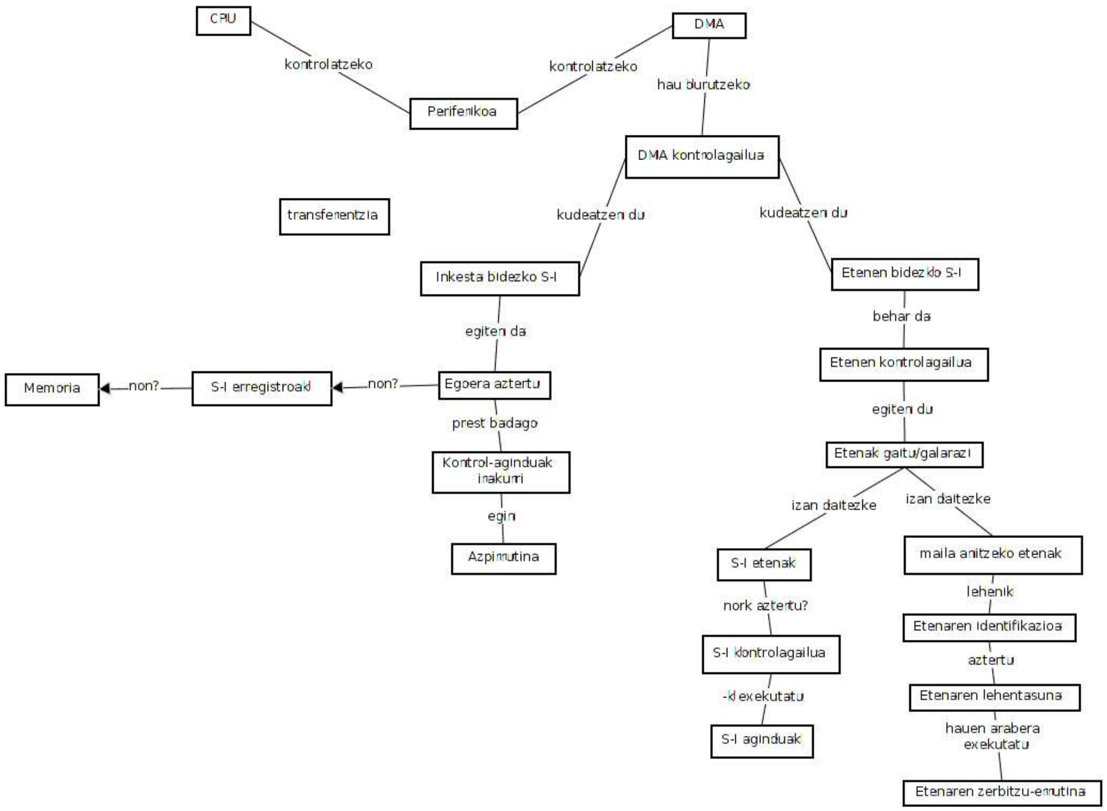


Figure D.26: Student concept map #26

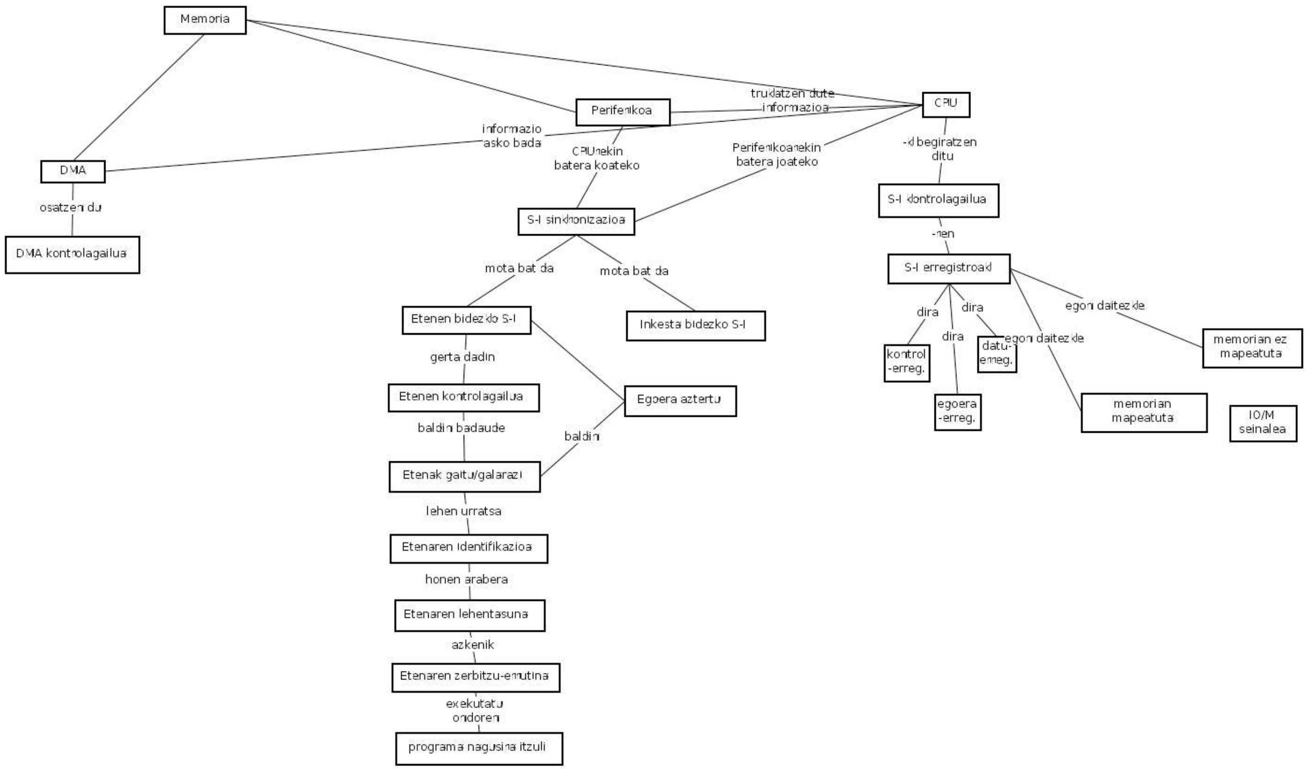


Figure D.27: Student concept map #27

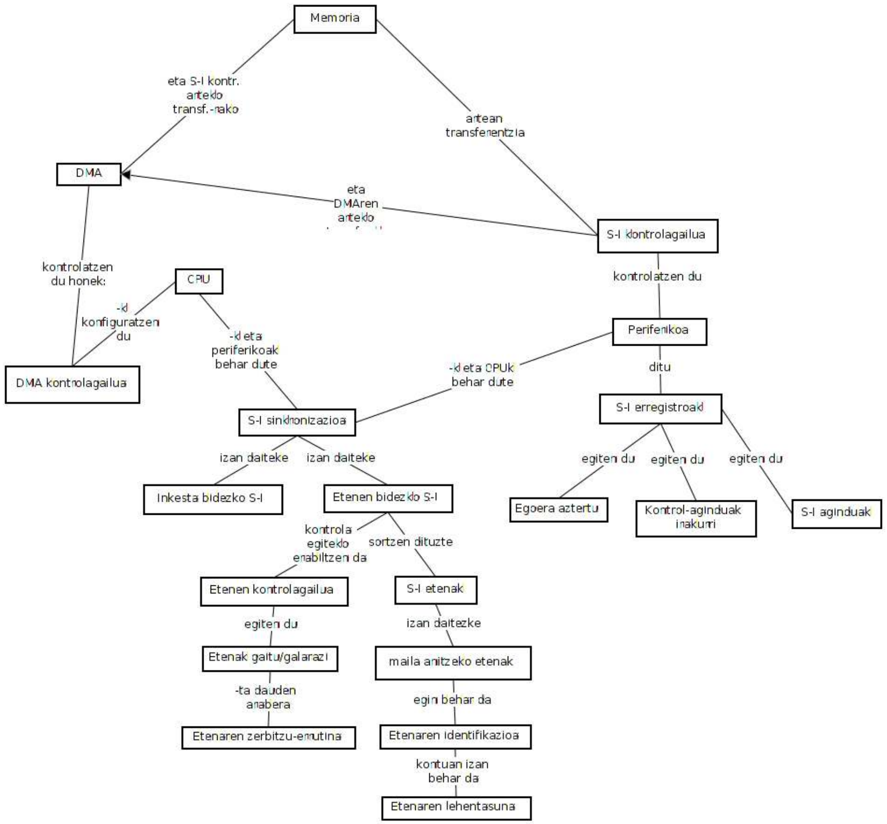


Figure D.28: Student concept map #28

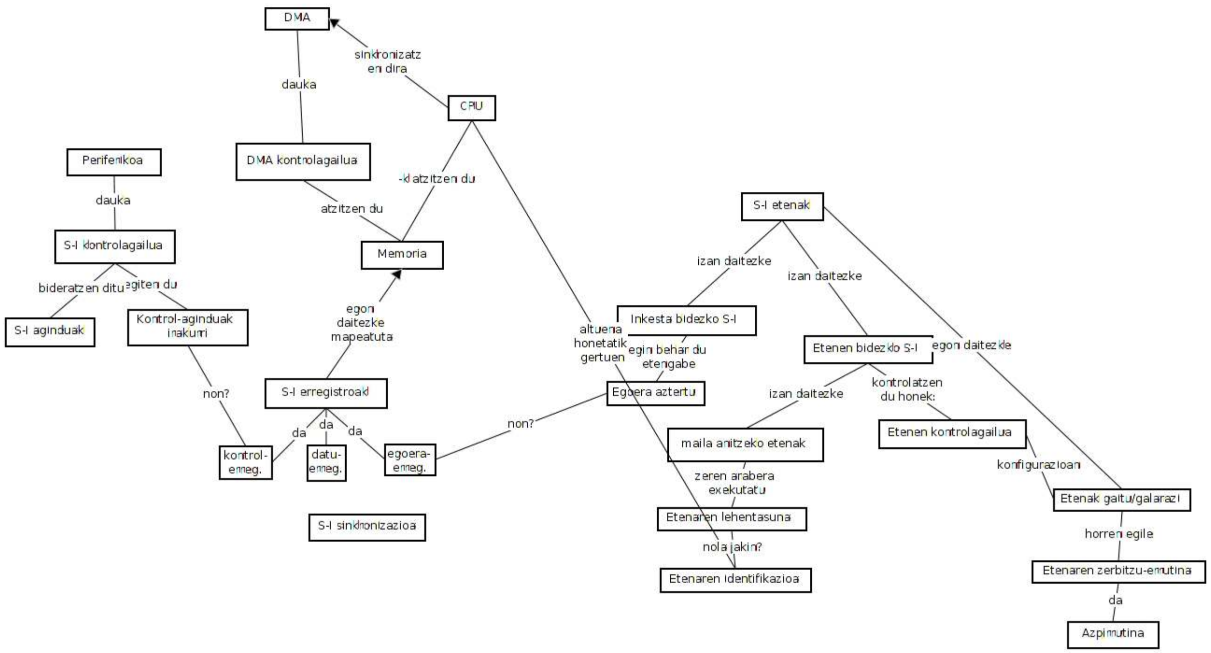


Figure D.29: Student concept map #29

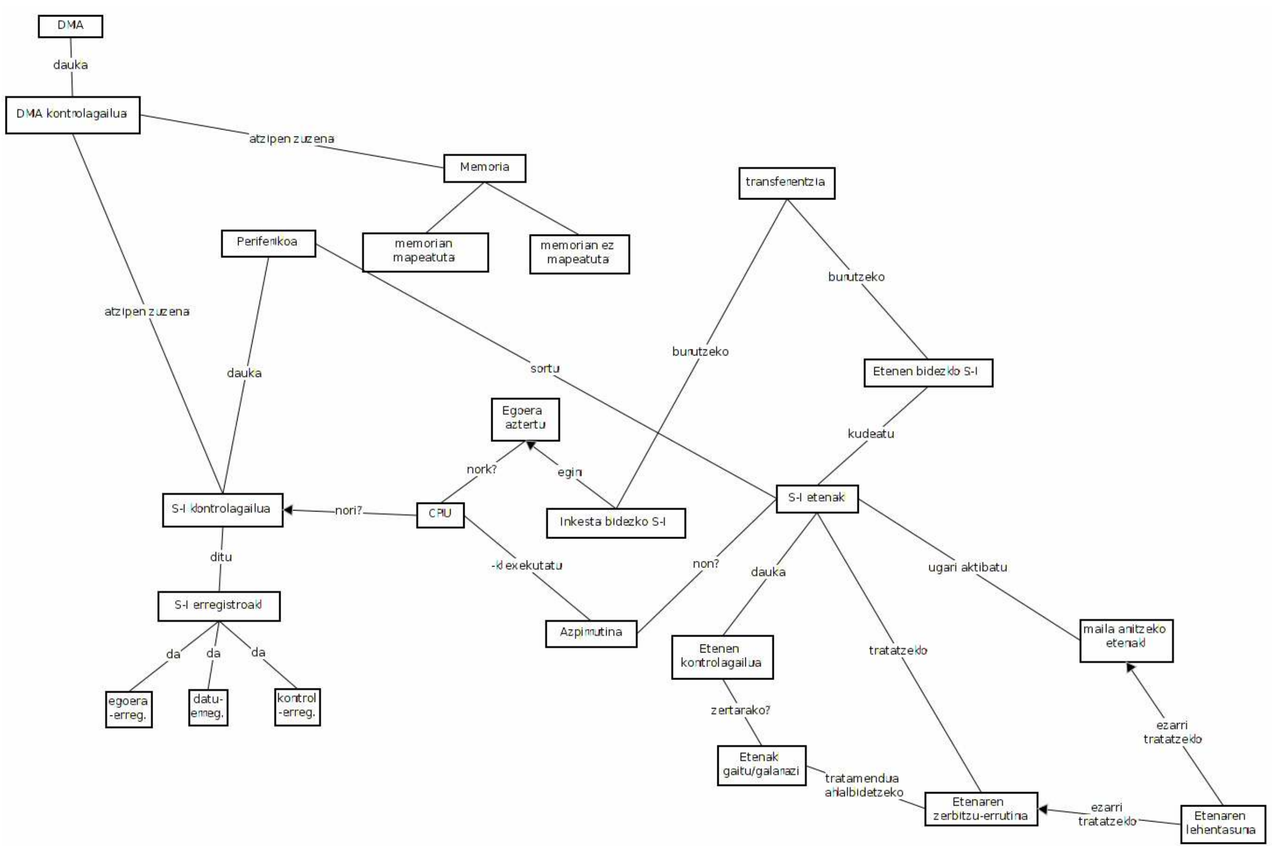


Figure D.30: Student concept map #30

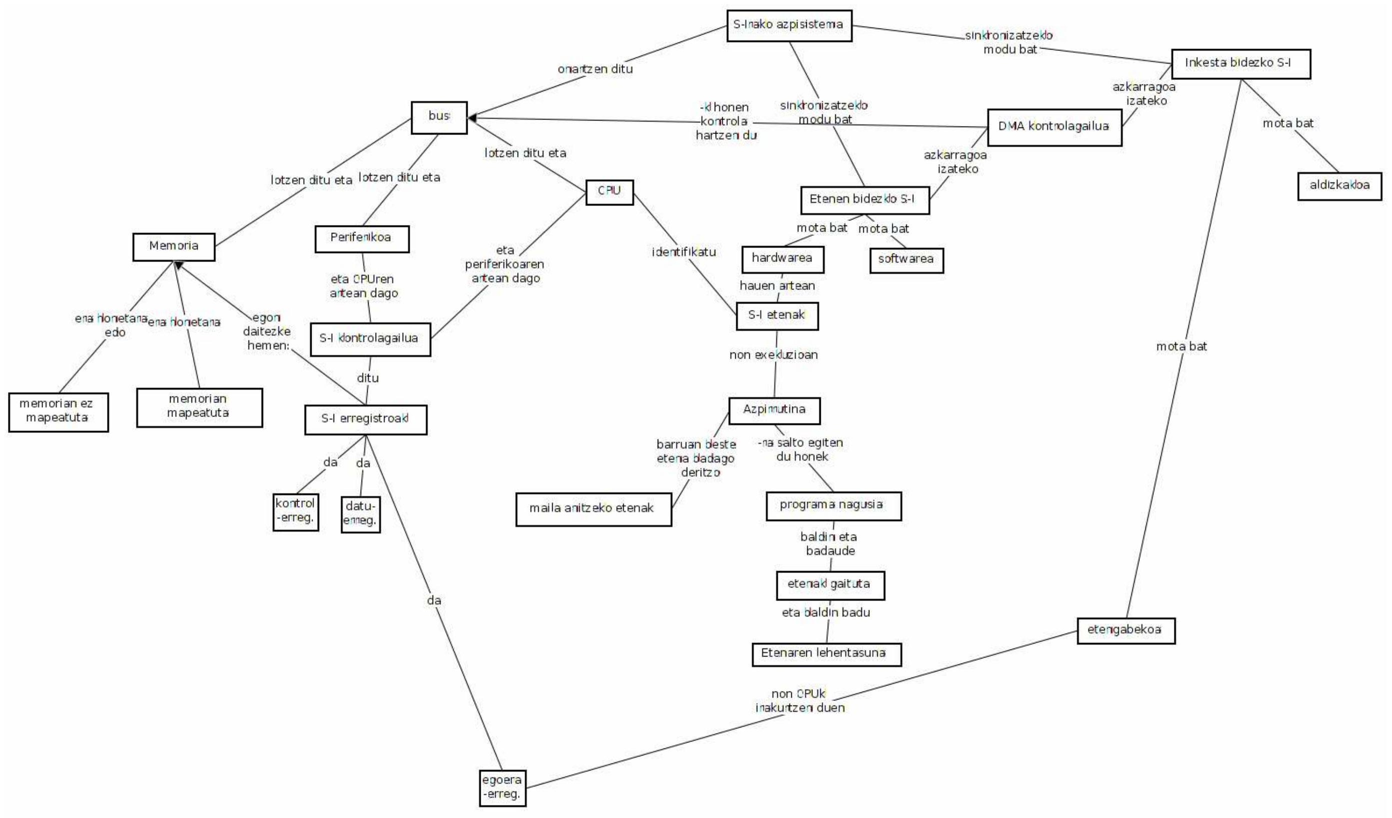


Figure D.31: Student concept map #31

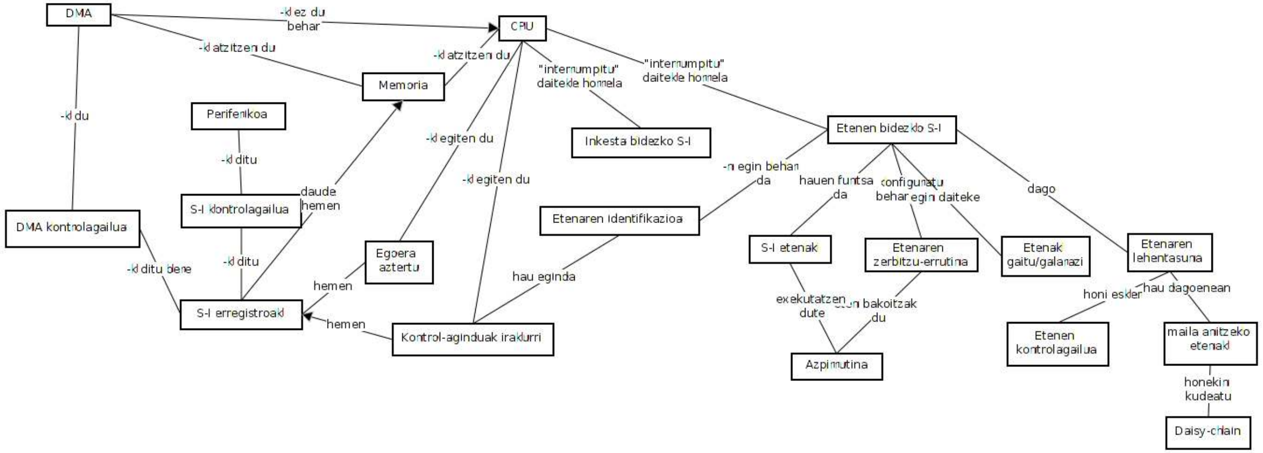


Figure D.32: Student concept map #32

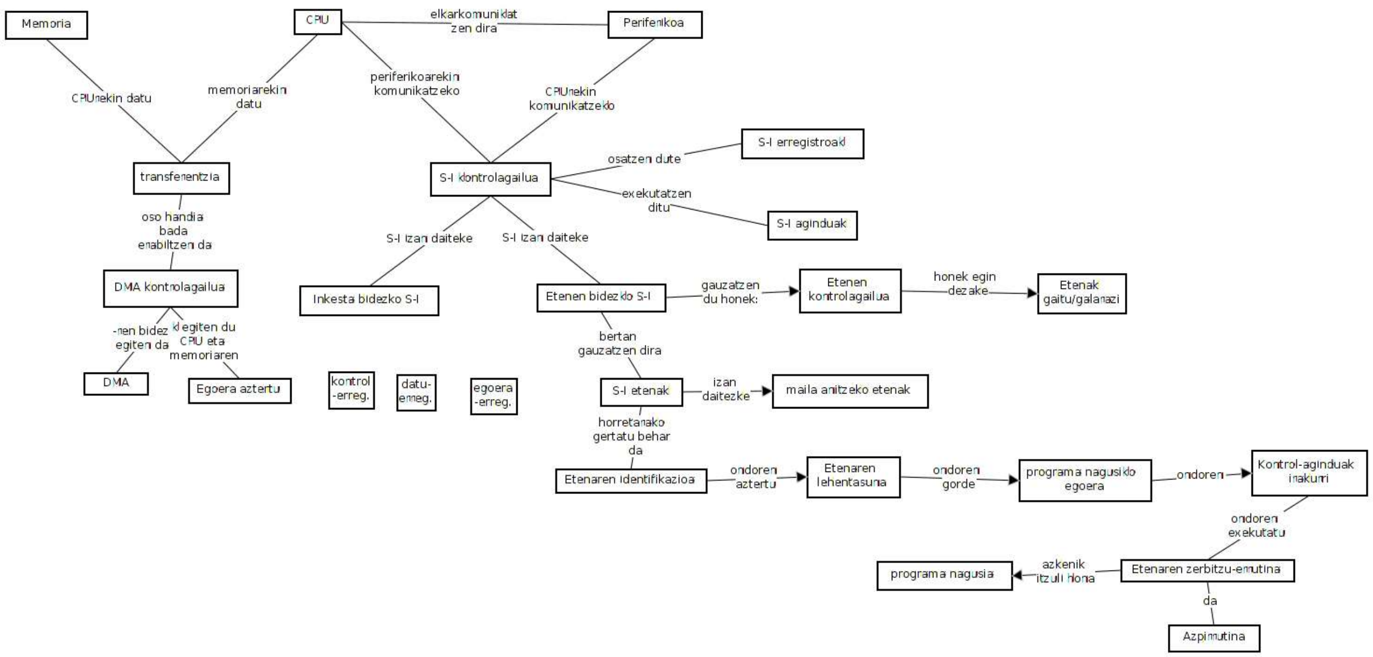


Figure D.33: Student concept map #33

Servicio Editorial de la Universidad del País Vasco (UPV/EHU)
Euskal Herriko Unibertsitateko Argitalpen Zerbitzua (UPV/EHU)
University of the Basque Country - Editorial Service (UPV/EHU)
ISBN: 978-84-9082-010-0