SYLLABUS

1. Information regarding the programme

1.1 Higher education institution	**Babeş Bolyai University**
1.2 Faculty	**Faculty of Mathematics and Computer Science**
1.3 Department	**Department of Computer Science**
1.4 Field of study	**Mathematics**
1.5 Study cycle	**Bachelor**
1.6 Study programme / Qualification	**Mathematics and Computer Science (in english)**

2. Information regarding the discipline

2.1 Name of the discipline				**Computer Systems Architecture**			
2.2 Course coordinator				**Lect. PhD. Coroiu Adriana Mihaela**			
2.3 Seminar coordinator				**Lect. PhD. Coroiu Adriana Mihaela**			
2.4. Year of study	**2**	2.5 Semester	**1**	2.6. Type of evaluation	**E**	2.7 Type of discipline	**Compulsory**

3. Total estimated time ((hours/semester of didactic activities)

3.1 Hours per week	4	Of which: 3.2 course	2	3.3 seminar/laboratory	1 sem + 1 lab
3.4 Total hours in the curriculum	56	Of which: 3.5 course	28	3.6 seminar/laboratory	28
Time allotment:			hours		
Learning using manual, course support, bibliography, course notes			20		
Additional documentation (in libraries, on electronic platforms, field documentation)			10		
Preparation for seminars/labs, homework, papers, portfolios and essays			20		
Tutorship			4		
Evaluations			14		
Other activities:					
3.7 Total individual study hours			32		
3.8 Total hours per semester			100		
3.9 Number of ECTS credits			4		

4. Prerequisites (if necessary)

4.1. curriculum	-
4.2. competencies	-

5. Conditions (if necessary)

5.1. for the course	◎ projector
5.2. for the seminar /lab	◎ Laboratory with computers

6. Specific competencies acquired

61. Professional competencies	C6.1 Identification of basic concepts and models for computer systems and computer networks. C6.2 Identification and description of the basic architectures for the organization and management of systems and networks.
6.2 Transversal competencies	CT1 Application of organized and efficient work rules, of responsible attitudes towards the didactic and scientific domain, for the creative exploitation of their own potential according to the principles and rules of professional ethics CT3 Use of effective methods and techniques of learning, information, research and development of the capacity to exploit knowledge, to adapt to the requirements of a dynamic society and communication in Romanian language and in a foreign language

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	Knowledge of the computer architecture models, processor functioning, computer information representation usage
7.2 Specific objective of the discipline	- Understanding by the students of the computer architecture models, processor functioning, computer information representation usage - Initiation in assembler language programming, which will assure the comprehension of the microprocessor architecture and functioning - Understanding the basic functions of a computer's architectural components and its native low-level workflow. Awareness of the architectural impact on designing and implementing high level programming languages. - Understanding the impact of the 80x86 processor architecture on Windows functioning and limitations. Awareness of the triade computer architecture – operating systems – programming languages and their interactions as the basic core of Computer Science.

8. Contents

8.1 Course	Teaching methods	Remarks
1. Data representation – part 1 **2**. **Data representation** – part 2 **3. Computing systems architecture** and **The 80x86 microprocessor's architecture** **4. Assembly language basic elements:** the source line format, location counter, labels, expressions, accessing the operands, operators. Temporary non-destructive conversions and their specific operators. **5. Assembly language basic instructions:** transfer instructions, signed and unsigned arithmetic operations, bitwise shifting and rotating, logical bitwise operations. **6**. **The 80x86 microprocessor's Eflags** register. The flags role and classifications. Examples of usage and case studies in conjuction with basic arithmetic operations. **7. Conversions classification.** Signed vs unsigned conversions instructions. Non destructive operators vs. destructive instructions conversions in assembly language. Examples and case studies. **8. The Bus Interface Unit** (BIU) of the 80x86 microprocessor: the address registers, segment registers, machine instructions representation. The address computation mechanism, addressing modes, far addresses and near addresses. The offset specification formula on 32 bits vs. on 16 bits. **9. Directives** for defining segments, data definition directives, the EQU and INCLUDE directives. Data types and the impact of data type interpretations and little-endian representation on accessing memory data.	Exposure, description, explanation, examples, discussion of case studies	

10. Overflow analysis. the overflow concept in mathematics vs. practical memory overflow in a Computing System. The 80x86 architecture reactions to an overflow for each of the four basic arithmetic operations. **11. String instructions.** Conditional and unconditional jump instructions, looping instructions, string parsing in assembly language with non specific instructions. Specific strings instructions and their efficiency. Examples and case studies. **12. Windows Input/Output Function Calls** (printf and scanf) and **Text files** (fopen, fread, fscanf, fprintf, fclose) processing operations callable from NASM assembler **13. Multi-module programming** in assembly language. Import – export mechanisms and shared resources between separate modules in assembly language. **14. Review of** theoretical aspects and **additional problems**: integration of the concepts already presented: data type, little-endian and directives with specific instructions for signed and unsigned representations.		

Bibliography

1. Al. Vancea, F. Boian, D. Bufnea, A. Andreica, A. Darabant, A. Navroschi – Arhitectura calculatoarelor. Limbajul de asamblare 80x86., Editura Risoprint, Cluj-Napoca, 2014.
2. Al. Vancea, F. Boian, D. Bufnea, A. Gog, A. Darabant, A. Sabau – Arhitectura calculatoarelor. Limbajul de asamblare 80x86., Editura Risoprint, Cluj-Napoca, 2005.
3. A. Gog, A. Sabau, D. Bufnea, A. Sterca, A. Darabant, Al. Vancea – Programarea în limbaj de asamblare 80x86. Exemple si aplicatii., Editura Risoprint, Cluj-Napoca, 2005.
4. Randal Hyde – The Art of Assembly Programming, No Starch Press, 2003. (http://homepage.mac.com/randyhyde/webster.cs.ucr.edu/www.artofasm.com/DOS/index.html)
5. Boian F.M. Vancea A. Arhitectura calculatoarelor, suport de curs. Facultatea de Matematica si
Informatica, Centrul de Formare Continua si Invatamânt la Distanta,. Ed. Centrului de Formare Continua si Invatamânt la Distanta, Cluj, 2002
6. Irvine, K.R., 2015. *Assembly language for x86 processors*.

7. Kusswurm, D., 2014. *Modern X86 Assembly Language Programming*. Springer.
8. Carter, P.A., 2004. *PC Assembly Language*. Github: (http://pacman128.github.io/static/pcasm-book.pdf)
9. Cavanagh, J., 2013. *X86 Assembly Language and C Fundamentals*. CRC Press.
10. Guide, P., 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part, 2*, p.11. (http://www.facweb.iitkgp.ac.in/~goutam/compiler/readingMaterial/intelXeon/253665.pdf)

8.2 Seminar/Laboratory	**Teaching methods**	**Remarks**
Seminars: S1: Introduction to the IA-32 assembly language. Converting numbers between numbering bases 2, 10, 16. Representation of integer numbers in the computer's memory. S2: Signed and unsigned instructions. Arithmetic instructions (addition, substraction, multiplications and divisions). Signed and unsigned conversions. S3: Little-endian representation of data in memory. Conditional and unconditional jumps. String operations. S4. Bitwise instructions (Bitwise logical operations, Shift and rotate operations) S5: Specific string instructions. Complex string problems. S6: Library functions call (printf, scanf, fread, fscanf, fprintf, fclose)	Exposure; Description; Explanation; Examples; Discussion of case studies; Practical projects.	Seminar is structured as 2 hour classes every second week

S7: Multi-module programming in assembly language. **<u>Laboratories</u>** L1: Converting between different number bases. Bit. Sign bit. Complementary code. Representing signed integers. Tools for laboratories. Structure of a NASM program in assembly. L2: Arithmetic expressions based on arithmetic instructions (additions, substractions, multiplications, divisions, little-endian, signed and unsigned conversions, declaring variables/constants) L3: Complex arithmetic expressions and bitwise operations. L4: Specific string operations (Instructions for comparisons, conditional jumps and repetitive loops and Instructions working on strings of bytes, words, doublewords and quadwords). L5: Function calls: Function libraries, Using external functions. Call conventions, Calling a system function. Standard msvcrt functions L6: Text file operations (open, write, read, close).		Laboratory is structured as 2 hour classes every second week. Laboratory problems assigned at a lab, have to be presented in the next lab.

L7: Multi-module programming in assembly language.		

Bibliography

1. Al. Vancea, F. Boian, D. Bufnea, A. Andreica, A. Darabant, A. Navroschi – Arhitectura calculatoarelor. Limbajul de asamblare 80x86., Editura Risoprint, Cluj-Napoca, 2014.
2. Al. Vancea, F. Boian, D. Bufnea, A. Gog, A. Darabant, A. Sabau – Arhitectura calculatoarelor. Limbajul de asamblare 80x86., Editura Risoprint, Cluj-Napoca, 2005.
3. A. Gog, A. Sabau, D. Bufnea, A. Sterca, A. Darabant, Al. Vancea – Programarea în limbaj de asamblare 80x86. Exemple si aplicatii., Editura Risoprint, Cluj-Napoca, 2005.
4. Randal Hyde – The Art of Assembly Programming, No Starch Press, 2003. (http://homepage.mac.com/randyhyde/webster.cs.ucr.edu/www.artofasm.com/DOS/index.html)
5. Boian F.M. Vancea A. Arhitectura calculatoarelor, suport de curs. Facultatea de Matematica si
Informatica, Centrul de Formare Continua si Invatamânt la Distanta,. Ed. Centrului de Formare Continua si Invatamânt la Distanta, Cluj, 2002
6. Irvine, K.R., 2015. *Assembly language for x86 processors*.
7. Kusswurm, D., 2014. *Modern X86 Assembly Language Programming*. Springer.
8. Carter, P.A., 2004. *PC Assembly Language*. Github: (http://pacman128.github.io/static/pcasm-book.pdf)
9. Cavanagh, J., 2013. *X86 Assembly Language and C Fundamentals*. CRC Press.
10. Guide, P., 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, *2*, p.11. (http://www.facweb.iitkgp.ac.in/~goutam/compiler/readingMaterial/intelXeon/253665.pdf)

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

The course exists in the studying program of all major universities in Romania and abroad;
◎ The content of the course is considered by the software companies as important for average programming skills

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Course	Testing the basic principles of the domain and their interactions	Written exam	45 %
	Verifying the understanding of the assembly language basic operations and mechanisms	Moodle midterm online multiple choice test	15 %
	Application of the 32 bits assembly language principles for problem solving;	Average grade received for the laboratory work	15 %
10.5 Lab/Seminar activities	Developing and implementing an assembly language code solution for a given problem	Practical exam	15 %
	Evaluating the students activities during the seminaries	Seminar activity	10 %
10.6 Minimum performance standards	- For participating at the written exam, a student must have at least 5 seminar attendances and 6 laboratory attendances. - Knowledge of the basic concepts. Each student has to prove that he/she has acquired an acceptable level of knowledge and understanding of the domain, that he/she is capable of expressing the acquired knowledge in a coherent form, that he/she has the ability of using this knowledge for problem solving. - For successfully passing the examination, a student must have at least 5 for the laboratory average, for the written exam, for the practical exam, and minimum 5 as a final grade.		

Date 14.04.2020

Signature of course coordinator
Lect. PhD Adriana Mihaela COROIU

Signature of seminar coordinator
Lect. PhD. Adriana Mihaela COROIU

Date of approval

Signature of the head of department
Prof. PhD. Anca ANDREICA

assembler
is a utility program that converts source
code programs from assembly language into machine language. A
linker
is a utility program that
combines individual files created by an assembler into a single executable program. A related
utility, called a
debugger
, lets you to step through a program while it's running and examine registers
and memory.

Assembly language programmers deal with data at the physical level, so they must be adept at examining memory and registers. Often, binary numbers are used to describe the contents of computer memory; at other times, decimal and hexadecimal numbers are used. You must develop a certain fluency with number formats, so you can quickly translate numbers from one format to another.
Each numbering format, or system, has a *base*, or maximum number of symbols that can be assigned to a single digit. Table 1-2 shows the possible digits for the numbering systems used most commonly in hardware and software manuals. In the last row of the table, hexadecimal numbers use the digits 0 through 9 and continue with the letters A through F to represent decimal values 10 through 15. It is quite common to use hexadecimal numbers when showing the contents of computer memory and machine-level instructions.

1.3.1 Binary Integers

A computer stores instructions and data in memory as collections of electronic charges. Representing these entities with numbers requires a system geared to the concepts of *on* and *off* or *true* and *false*.
Binary numbers are base 2 numbers, in which each binary digit (called a *bit*) is either 0 or 1. ***Bits*** are numbered sequentially starting at zero on the right side and increasing toward the left. The bit on the left is called the *most significant bit* (MSB), and the bit on the right is the *least significant bit* (LSB). The MSB and LSB bit numbers of a 16-bit binary number are shown in the following figure: