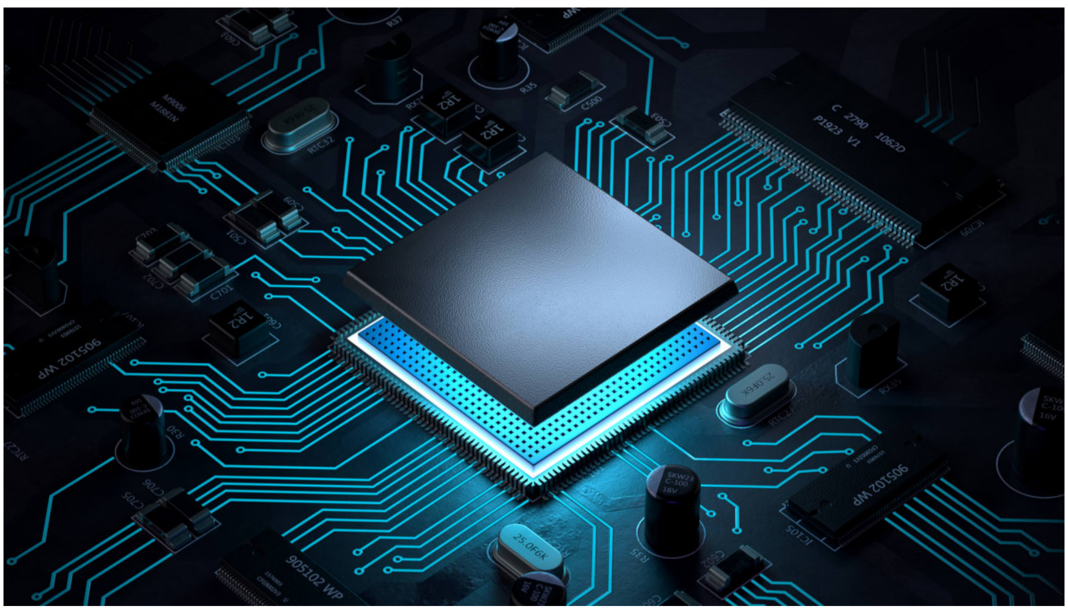
Computer Architecture and Embedded Systems
Semester Assignment

Vasileios Nastos[1] and **Panagiotis Koromilias**[2]

[1]MSc in department of informatics and telecommunications, University of Ioannina, **Id:137**
[2]MSc in department of informatics and telecommunications, University of Ioannina, **Id:150**

Date:30/01/2023



1 Introduction

Listing 1: Assignment basic code

```
.data
message: .asciiz "Result = "
.text
.globl main

main:
    la $4, message # $a0 <- start of welcome message
    li $v0, 4 # $v0 <- service #4
    syscall # call to system service
    nop # not operation
    move $20, $zero # $s4 <- 0, initialize accumulator
    move $17, $0 # $s1 <- 0, initialize iterations counter
# Next two instructions mean
#"While $s1 < 100 Do"
loop:
    slti $18, $17, 101 # $s1 < 100 => $s2 <- 1
    beq $18, $0, end_lop # $s2 = 0 => go to end_loop
    add $20, $20, $17 # $s4 <- $s4 + $s1, add number
    add $17, $17, 1 # $s1 <- $s1 + 1, update counter of iterations
    j loop # go to loop
nop
end_lop:
    move $4, $20 # $a0 <- $s4, load result of sum
    li $v0,1 # $v0 <- service #1 (data is already in $a0)
    syscall # call to system service
    _exit: # main program exit
    li $v0,10 # $v0 <- service #10
    syscall # call to system service
nop
```

The code 1 is written in MIPS assembly language, which is used to program microprocessors in the MIPS architecture. It performs a simple task of adding the numbers from 0 to 100 and displays the result. The code starts by initializing the accumulator register ($20) to 0, and the iterations counter register ($17) to 0. A while loop is created to repeatedly add the current iteration number to the accumulator and increment the iteration counter until it reaches 100. Finally, the result of the sum is stored in the argument register ($4) and displayed with a system service call (syscall). The program ends with an exit syscall service to terminate the program.

2 Topic 1

1. **What is the purpose of the project?**
   An: The objective of this program is to calculate the sum of the first 100 positive integers and print the result on the screen. This program is written in MIPS Assembly language, specifically for a MIPS architecture computer. It performs the following steps:

   - Loads the message "Result = " into $a0, and prints it using system call 4.
   - Initializes an accumulator ($s4) with 0, and a counter of iterations ($s1) with 0.
   - Enters a loop, which continues while the counter of iterations ($s1) is less than 100.
   - Inside the loop, the program adds the value of the counter ($s1) to the accumulator ($s4).
   - The counter is then incremented by 1.
   - The program jumps back to the start of the loop.
   - After the loop, the final result is loaded into $a0, and printed using system call 1.
   - The program exits with system call 10.

   The program performs arithmetic operations, loads and stores values from/to memory, and interacts with the system through system calls.

2. **What is the meaning of the SPIM directive ".asciiz"?**
   An:.The ".asciiz" directive in SPIM (a MIPS simulator) is used to define a null-terminated ASCII string. It allocates space in memory for a string of characters and stores the string along with a null terminator (ASCII value 0) at the end. The ".asciiz" directive is used to define a string "Result = " in memory, with a null terminator added at the end. The label "message" is assigned to the starting address of this string in memory, so that it can be referred to elsewhere in the code.

3. **What is the purpose of the SPIM ".data" and ".text" directives?**
   An: The ".data" and ".text" directives in SPIM are used to separate the data and code segments of a program. .data - This directive starts the data segment of the program, where global variables and constant data are defined. All data values declared within the .data section are stored in the data segment of the program's memory. .text - This directive starts the code segment of the program, where the actual instructions are stored. The code segment contains the program's executable instructions and all the functions and subroutines. The separation of data and code into separate segments makes it easier to manage the program's memory and enables the system to execute the instructions more efficiently. Additionally, it allows for better control over the read/write permissions of the program's data and code.

4. **What does a label indicate?**
   An: A label in assembly language is a symbol that represents a specific memory address. Labels are used to give a name to a specific location in memory, which can then be referred to by other instructions in the program. The label message is assigned to the starting address of the string "Result = " in memory, which was defined in the .data segment. Later in the code, the value stored at message is loaded into the register $a0usingtheinstructionla4$, message to print the string. The use of the label makes the code more readable and easier to maintain, as it gives a descriptive name to a specific location in memory.

3 Topic 2

1. **List and explain the "programming block (instruction set)" required, to send messages to the console?**
   An: To send messages to the console in MIPS assembly, the following instruction sets are required:

   - li $v0, 4: This instruction sets the value of $v0 to 4, which is the service number for printing a string.
   - la $a0, message: This instruction loads the address of the message to be printed into $a0.
   - syscall: This instruction triggers a system call to print the message stored in $a0 to the console.
   - li $v0, 1: This instruction sets the value of $v0 to 1, which is the service number for printing an integer.
   - syscall: This instruction triggers a system call to print the integer stored in $a0 to the console.
   - li $v0, 10: This instruction sets the value of $v0 to 10, which is the service number for terminating the program.
   - syscall: This instruction triggers a system call to terminate the program.

Listing 2: Code for printing a message through the command line

```
        .data
        message: .asciiz "Hello, world!\n"

        .text
        .globl main

        main:
```

```
        la $a0, message # Load address of message into $a0
        li $v0, 4 # Load system call code for printing strings into $v0
        syscall # Call the system to print the string stored in $a0
        li $v0, 10 # Load system call code for exit into $v0
        syscall # Call the system to exit the program
```

Listing 3: Equivelant code in C

```
#include <stdio.h>

int main(void) {
  printf("Hello, world!\n");
  return 0;
}
```

2. **Indicate the ”programming block” required to import data from the keyboard**
   An: In code presented in lstlisting 4, li $v0, 5 loads the system call code 5, which is used to read an integer from the keyboard. The syscall instruction invokes the system call, and the integer entered by the user is stored in register $v0. Finally, the integer is moved to register $s0 for future use.

Listing 4: code for question 2 of Topic 2

```
li $v0, 5 # Load system call code for reading an integer
syscall # Call the system to read integer
move $s0, $v0 # Store the input integer in register $s0
```

3. **Once the data is retrieved from the keyboard, where is it placed?**
   An: Listing 5.Once data is recovered from the keyboard in MIPS, it's typically stored in a register or memory location. For example, the syscall instruction is used to read data from the keyboard into a register in MIPS. Then the value can be stored in a memory location. It's important to note that in MIPS, memory is addressed by its address, not by its content, so it's necessary to use a register to hold the memory address before storing or loading data from memory.

Listing 5: Data storage example

```
li $v0, 5 # specify that we want to read an integer
syscall     # read integer from keyboard into $v0
sw $v0, 0($t0) # store the integer read from keyboard into memory
```

4. **Create a program to add two integers that have been entered from the keyboard and display the result as follows: ”Sum= result”.**
   An: The program in Listing 6 first reads two integers from the keyboard using the syscall instruction with $v0 set to 5, then stores the result in register $t0. The sum of the two integers is stored in $t0. The string ”Sum = ” is displayed using the syscall instruction with $v0 set to 4 and $a0 set to the address of sum_message. Finally, the sum is displayed using the syscall instruction with $v0 set to 1 and $a0 set to $t0. The program terminates using the syscall instruction with $v0 set to 10.

Listing 6: Code for question 4 in topic 4

```
.data
sum_message: .asciiz "Sum = "

.text
.globl main
main:
    li $v0, 5 # read integer into $v0
    syscall
    move $t0, $v0 # move the first integer into $t0

    li $v0, 5 # read integer into $v0
    syscall
```

```
    add $t0, $t0, $v0 # add the second integer to the first one in $t0

    li $v0, 4 # specify print_str syscall
    la $a0, sum_message # load address of sum_message into $a0
    syscall

    li $v0, 1 # specify print_int syscall
    move $a0, $t0 # move the sum into $a0
    syscall

    li $v0, 10 # specify exit syscall
    syscall
```

4 Topic 3

1. **Convert the following code from C to Assembly**

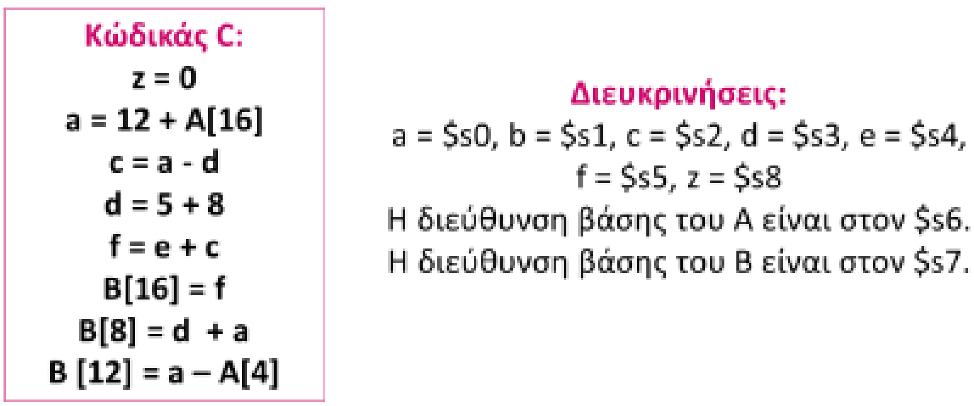


Figure 1: Topic 3 question 1 clarifications

An: The code in lstlisting 7 executes the following steps:

- Loading a constant value of 0 into register $s8.
- Loading a value from memory at address 64 + $s6 into register $t0.
- Adding a constant value of 12 to register $t0 and storing the result in register $s0.
- Subtracting the contents of register $s3 from register $s0 and storing the result in register $s2.
- Adding 5 and 8 to the contents of register $s3.
- Adding the contents of register $s4 and register $s2 and storing the result in register $s5.
- Storing the contents of register $s5 in memory at address 64 + $s7.
- Adding the contents of register $s3 and register $s0 and storing the result in register $t1.
- Storing the contents of register $t1 in memory at address 32 + $s7.
- Loading a value from memory at address 16 + $s6 into register $t2.
- Subtracting the contents of register $t2 from register $s0 and storing the result in register $t3.
- Storing the contents of register $t3 in memory at address 48 + $s7.

Listing 7: Answer to the Topic 3 question 1

```
li $s8, 0
lw $t0, 64($s6)
addi $s0, $t0, 12
sub $s2, $s0, $s3
addi $s3, $s3, 5
addi $s3, $s3, 8
add $s5, $s4, $s2
sw $s5, 64($s7)
add $t1, $s3, $s0
sw $t1, 32($s7)
lw $t2, 16($s6)
sub $t3, $s0, $t2
sw $t3, 48($s7)
```

2. **Convert the previous Assembly code 7 into a machine code table in decimal system.**

<table>
<tr><th>op</th><th>rs</th><th>rt</th><th>rd</th><th>Shamt</th><th>funct</th></tr>
<tr><td>8</td><td>24</td><td>24</td><td colspan="3">0</td></tr>
<tr><td>35</td><td>22</td><td>8</td><td colspan="3">64</td></tr>
<tr><td>8</td><td>8</td><td>16</td><td colspan="3">12</td></tr>
<tr><td>0</td><td>16</td><td>19</td><td>18</td><td>0</td><td>34</td></tr>
<tr><td>8</td><td>19</td><td>19</td><td colspan="3">5</td></tr>
<tr><td>8</td><td>19</td><td>19</td><td colspan="3">8</td></tr>
<tr><td>0</td><td>20</td><td>18</td><td>21</td><td>0</td><td>32</td></tr>
<tr><td>43</td><td>21</td><td>23</td><td colspan="3">64</td></tr>
<tr><td>0</td><td>19</td><td>16</td><td>9</td><td>0</td><td>32</td></tr>
<tr><td>43</td><td>9</td><td>23</td><td colspan="3">32</td></tr>
<tr><td>35</td><td>22</td><td>10</td><td colspan="3">16</td></tr>
<tr><td>0</td><td>16</td><td>10</td><td>11</td><td>0</td><td>34</td></tr>
<tr><td>43</td><td>11</td><td>23</td><td colspan="3">48</td></tr>
</table>

Figure 2: Machine code for Assembly code in Listing 7

3. **For each line of code the format to which it belongs will be declared, the table will be of (depending on the format)**

op	rs	rt	rd	shamt	funct

Format	op	rs	rt	rd	Shamt	funct
I	8	24	24	0		
I	35	22	8	64		
I	8	8	16	12		
R	0	16	19	18	0	34
I	8	19	19	5		
I	8	19	19	8		
R	0	20	18	21	0	32
I	43	21	23	64		
R	0	19	16	9	0	32
I	43	9	23	32		
I	35	22	10	16		
R	0	16	10	11	0	34
I	43	11	23	48		

Figure 3: Answer to topic 3 question 2

5 Topic 4

1. **Implement a short dialogue, of your choice, with the user.**
   An: The code in Listing 8 This code is a MIPS assembly language program that prompts the user for their name, field of study, and academic year, and stores the input in memory. The program uses the syscall service to perform input/output operations. The syscall service is a mechanism for interacting with the operating system from within a MIPS program. The code defines data sections "value", "msg1", "msg2", "msg3", and "msg4". The "value" section is an array of 3 words (integer values) used to store the user inputs. The "msg1", "msg2", "msg3", and "msg4" sections are strings used to print messages on the screen. The main program first loads the address of the "value" data into register $t0. It then performs four sets of operations to prompt the user for input and store the input in memory: Print the message "whats your name;". Read a string from the user and store it in register $v0. Store the contents of register $v0 in memory, at the first word of the "value" array. Repeat steps 1 to 3 for the fields of study and academic year, with different messages and different storage locations in the "value" array.

Listing 8: Answer to Topic 4 question 1

```
.data
    value: .word 0, 0, 0
    msg1:  .asciiz "Hello what's your name; \n"
    msg2:  .asciiz "Which is your department affiliation; \n"
    msg3:  .asciiz "Which is your semester; \n"
    msg4:  .asciiz "Thank you for the inspiring chat\n"


.text
.globl main

main:
  la $t0, value
  li $v0, 4
  la $a0, msg1
  syscall
  li $v0, 8
  syscall
  sw $v0, 0($t0)
  li $v0, 4
  la $a0, msg2
  syscall

  li $v0, 8
  syscall
  sw $v0, 8($t0)
  li $v0, 4
  la $a0, msg3
  syscall

  li $v0, 8
  syscall
  sw $v0, 12($t0)
  li $v0, 4
  la $a0, msg3
  syscall
  li $v0, 4
  la $a0, msg4
  syscall
  li  $v0, 10
  syscall
```

2. **Modify the program given in the description to iteratively execute the original procedure, but this time by squaring all the unnecessary numbers**

Listing 9: Answer to Topic 4 question 2

```
.data
    message: .asciiz "Result for add = "
    message2: .asciiz "\nResult for ^2 = "
.text
.globl main
main:
 la $4, message        # $a0 <- start of welcome message

 li $v0, 4           # $v0 <- service #4
 syscall           # call to system service


 nop             # not operation
 move $20, $zero       # $s4 <- 0, initialize accumulator
 move $21, $zero       # $s5 <- 0, initialize accumulator

 move $17, $0          # $s1 <- 0, initialize iterations counter
 move $22, $0          # $s6 <- 0, initialize iterations counter


# Next two instructions mean
#"While $s1 < 100 Do"
```

```
addi $s6,2

loop: slti $18, $17, 101     # $s1 < 101 => $s2 <- 1
 beq $18, $0, end_lop        # $s2 = 0 => go to end_loop

 div $17, $s6
 mfhi $t0

beq    $t0, $0, if_else_label
  mul $23, $17, $17        # $s6 <- $s1 * $s1, ^2 number
    add $21, $21, $23          # $s5 <- $s5 + $s6, add number
  add $17, $17, 1          # $s1 <- $s1 + 1, update counter of iterations
  j if_end_label
if_else_label:
    add $20, $20, $17          # $s4 <- $s4 + $s1, add number
  add $17, $17, 1          # $s1 <- $s1 + 1, update counter of iterations
if_end_label:

 j loop              # go to loop
 nop



end_lop:
 move $4, $20        # $a0 <- $s4, load result of sum
 li $v0,1              # $v0 <- service #1 (data is already in $a0)
 syscall             # call to system service

 la $4, message2       # $a1 <- message2
 li $v0, 4           # $v1 <- service #4
 syscall           # call to system service

 move $4, $21            # $a0 <- $s5, load result of ^2
 li $v0,1              # $v0 <- service #1 (data is already in $a0)
 syscall             # call to system service



_exit:               # main program exit
 li $v0,10             # $v0 <- service #10
 syscall             # call to system service
 nop
```

6 Topic 5

Write and execute a program in MIPS assembly code that determines the number of prime numbers that are less than some positive integer N. This number is usually referred to as π(N). e.g., π(1) = 0, π(2) = 1, π(3) = 2, π(4) = 2, π(5) = 3, ..., π(10) = 4, etc. Remember that one (1) is not a prime number. Follow the steps below: The program takes as input a positive integer,

- A matrix of integers from 2 to N is defined,
- The smallest integer i in the table, which is not marked as a multiple of any of the integers already checked (of course 2 at the beginning), is identified, counted as a prime number, and then its multiples are marked, i.e., i, 2i, 3i, etc.

Step 3 is repeated. When $i > \sqrt{N}$, the program terminates and prints the number of prime numbers. E.g., for N = 20, the table is: 2, 3, 4, 5, 6, 7,8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 Nothing has been marked yet, and the smallest integer is 2. Therefore, 2 is counted as a prime number (shown underlined), while its multiples are highlighted (shown in bold): 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 The smallest unsigned integer is now i = 3. 3 is counted as a prime number and then its multiples are highlighted: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 A number can be marked more than once, such as 6, but it does not mean anything different, it is still not considered a prime number. The next smallest integer that is not marked is 5. However, it is greater than the square root of 20, and the program terminates after counting in the number of prime numbers (2 so far) the number of unlabeled integers in the table, i.e. the 6 numbers 5, 7, 11, 13, 17, 19. Thus, in total, π(20) = 8. Tip: Implement the matrix of integers 2, ..., N as a bit vector in the MIPS data area. In other words, define a vector of length N-1, where the 1st bit corresponds to 2, the 2nd bit to 3, and so on. Initially, the values of the vector will all be 1. Therefore, when a position needs to be marked, its value will change from 1 to 0. At the end of the process, all bits that have remained 1 will correspond to prime numbers. At the start of the program, the console output should be "Enter Integer:" A positive integer N should be able to be entered as a decimal number. The next output (after enter) should be "pi(N) =" followed by the correct value of pi(N), given as a decimal number. Example of execution:

- Enter Integer: 5000
- Enter: Enter: Enter: pi(5000) = 669

Listing 10: Answer to Topic 5

```
.data
lastm: .asciiz "\n Semester Assignment \n"
firstms: .asciiz "\nInsert Integer:\n"
primes1: .asciiz    "\nPrime numbers are between "
primes2:    .asciiz " and are: "
nl:     .asciiz "\n"
comma:  .asciiz ", "
ms1: .asciiz "p("
ms2: .asciiz ") = "
memory: .word 0


.text

main:
  la $t0, memory

  li $v0, 4
    la $a0, lastm
    syscall

    li $v0, 4
    la $a0, firstms
    syscall

    li $v0, 5
    syscall
    move $s0, $v0
  sw $s0, 0($t0)
    mul $a0, $s0, 4
    li $v0, 9
    syscall
    move $s1, $v0

    li $s2, 0
    bl_loop:
    sb $zero, 0($s1)
    beq $s0, $s2, endbl
    addi $s1, $s1, 1
    addi $s2, $s2, 1
    j bl_loop
    endbl:
    li $s2, 1
    ot_loop:
    addi $s2, $s2, 1
    mult $s2, $s2
    mflo $s3
    bgt $s3, $s0, exit

    #if prime[counter] == 0
    lb $s4, 0($s1)
    bnez $s4, ot_loop
    mul $s5, $s2, $s2
```

```
      $s2

      in_loop:
          bgt $s5, $s0, ot_loop
          add $s6, $s5, $s1
          sb  $s2, 0($s6)
          add $s5, $s5, $s2
          j in_loop
          j ot_loop           #

    exit:
      li $v0, 4
      la $a0, primes1
      syscall

      li $v0, 1
      move $a0, $s0
      syscall

      li $v0, 4
      la $a0, primes2
      syscall

      li $s2, 1
    li $s5, 0

    print:
          addi $s2, $s2, 1
        bgt $s2, $s0, done
          add $s3, $s1, $s2
          lb $s4, 0($s3)

          bnez $s4, print
          li $v0, 1
          move $a0, $s2
          syscall

          li $v0, 4
          la $a0, comma
        addi $s5, $s5, 1
        syscall

      j print
          done:
```