

Project II – Final Report

**Course Name:** Computer Organization

**Course Number:** (**60610MEEOS-CME0437**)

**Department:** Computer Engineering

**Lecturer:** Prof. Dr. Ercan SOLAK

**Year** (2022-2023)

**Students Name:**

Beyza Serap Sevim - 1804010009

Leyla Abdullayeva - 1904010038

Yusef Murad Rayyan - 1904010031

**Contents**

[General Description 2](#_heading=h.gjdgxs)

[Instruction Set Architecture (ISA) 2](#_heading=h.30j0zll)

[Coding and Explanations 3](#_heading=h.1fob9te)

[Binary Format 4](#_heading=h.3znysh7)

[Code binary format 6](#_heading=h.2et92p0)

[Datapath Diagram 12](#_heading=h.tyjcwt)

[Explanation of the components 14](#_heading=h.3dy6vkm)

[Video Link 15](#_heading=h.1t3h5sf)

# General Description

The processor is an 8-bit single-cycle machine. It has 3 registers: a program counter (PC) that holds the memory address of the next instruction to be executed, an address register (AR) that holds the memory address for load/store operations and the target address for jumps, and a general-purpose register (GPR) that holds data values and intermediate results of operations. The instructions and memory words are 8 bits wide, allowing a maximum of 256 different instructions and 256 words of memory. The processor has a basic set of 6 instructions for performing arithmetic and memory operations.

# Instruction Set Architecture (ISA)

The instruction set architecture (ISA) of the processor includes the following 6 instructions:

1.   **GET:** Load a value from memory into the general-purpose register.

●        **Example**: GET 0001 (loads the value at address 0001 into the general-purpose register)

2.   **SET:** Store a value from the general-purpose register into memory.

●        **Example**: SET 0001 (stores the value in the general-purpose register at address 0001)

3.   **PUSH:** Add a value from memory to the value in the general-purpose register.

●        **Example**: PUSH 0001 (adds the value at address 0001 to the value in the general-purpose register)

4.   **PULL:** Subtract a value from memory from the value in the general-purpose register.

●        **Example**:  PULL 0001 (subtracts the value at address 0001 from the value in the general-purpose register)

5.   **JUMP:** Jump to a specific memory address.

●        **Example**:  JUMP 0010 (sets the program counter to address 0010)

6.   **JUMPZ:** Jump the instruction, if the value of the register is 0.

* **Example:** JUMPZ

# 

# Coding and Explanations

Sample program to declare and sum the elements of an array, keep in mind ending the program is done by writing the 01000100 number into the $GPR register.

.data

            Array: . word 1,2,3,4,5

.text                           # The Text segment.

Main:

GET $GPR, 0 # Load the first element of the array into the GPR.

SET $GPR, 6 # Store the value in the GPR into memory location 6.

 # Increment the counter.

bitload:

PUSH $GPR, $GPR, 7 # Add the value in memory location 7 to the GPR.

SET $GPR, 7   # store the value in the GPR back into memory location 7.

 # Check if the counter is less than the length of the array.

GET $GPR, 5   # Load the value 5 (length of the array) into the GPR.

PULL $GPR, $GPR, 7 # Subtract the value in memory location 7 from the GPR.

JUMPZ END # If the result is 0, jump to instruction 19 (END).

         # Load the next element of the array into the GPR.

PUSH $GPR, $GPR, 8 # Add the value 8 (size of each element in the array) to the GPR.

GET $AR, $GPR      # Load the value in the AR (memory address) into the GPR.

PUSH $GPR, $GPR, $zero # Add the value in the GPR to the GPR.

SET $AR, $GPR       # Store the value in the GPR back into the AR.

GET $AR, $GPR   # Load the value at the memory location specified by the AR into the GPR.

# Add the value in the GPR to the value in memory location 6.

PUSH   $GPR, $GPR, 6       # Add the value in memory location 6 to the GPR.

SET $GPR, 6   # Store the value in the GPR back into memory location 6.

# Increment the counter.

PUSH $GPR, $GPR, 7 # Add the value in memory location 7 to the GPR.

SET $GPR, 7   # Store the value in the GPR back into memory location 7.

# Jump back to the beginning of the loop.

JUMP Bitload    # Jump to instruction 4.

END:

GET $GPR, 01000100 # This value is representing the $v0, 10 in MIPS.

SYSCALL

# Binary Format

1. **GET:** The GET instruction has the following format: GET $GPR, address Binary format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The first 3 bits (000) can be used to identify the instruction as a GET instruction. The next 5 bits can be used to encode the memory address to be loaded into the GPR.

1. **SET:** The SET instruction has the following format: SET $GPR, address Binary format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

The first 3 bits (001) can be used to identify the instruction as a SET instruction. The next 5 bits can be used to encode the memory address where the value in the GPR will be stored.

1. **PUSH:** Binary format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

The first 3 bits (001) can be used to identify the instruction as a PUSH instruction. The next 5 bits can be used to encode the memory address from which the value will be added to the value in the GPR.

1. **PULL:** The PULL instruction has the following format: PULL $GPR, $GPR, address Binary format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

The first 3 bits (010) can be used to identify the instruction as a PULL instruction. The next 5 bits can be used to encode the memory address from which the value will be subtracted from the value in the GPR.

1. **JUMP:** The JUMP instruction has the following format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

The first 3 bits (100) can be used to identify the instruction as a JUMP instruction. The next 5 bits can be used to encode the memory address to which the program counter will be set.

1. **JUMPZ** instruction has the following format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

The first 3 bits (101) can be used to identify the instruction as a JUMPZ instruction. The next 5 bits can be used to encode the memory address to which the program counter will be set if the value in the GPR is 0.

# Code binary format

**GET $GPR, 0 # Load the first element of the array into the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The first 3 bits (000) identify the instruction as a GET instruction. The next 5 bits (00000) encode the memory address 0, which is the first element of the array.

**SET $GPR, 6 # Store the value in the GPR into memory location 6.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

The first 3 bits (001) identify the instruction as a SET instruction. The next 5 bits (00110) encode the memory address 6, where the value in the GPR will be stored.

**bitload:**

**PUSH $GPR, $GPR, 7 # Add the value in memory location 7 to the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The first 3 bits (001) identify the instruction as a PUSH instruction. The next 5 bits (00111) encode the memory address 7, from which the value will be added to the value in the GPR.

**SET $GPR, 7 # store the value in the GPR back into memory location 7.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The first 3 bits (001) identify the instruction as a SET instruction. The next 5 bits (00111) encode the memory address 7, where the value in the GPR will be stored.

**GET $GPR, 5 # Load the value 5 (length of the array) into the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

The first 3 bits (000) identify the instruction as a GET instruction. The next 5 bits (00101) encode the memory address 5, which is the length of the array.

**PULL $GPR, $GPR, 7 # Subtract the value in memory location 7 from the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The first 3 bits (010) identify the instruction as a PULL instruction. The next 5 bits (00111) encode the memory address 7, from which the value will be subtracted from the value in the GPR.

**JUMPZ END # If the result is 0, jump to instruction 19 (END).**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

The first 3 bits (101) identify the instruction as a JUMPZ instruction. The next 5 bits (00100) encode the memory address 20, which is the address of the END label.

**Load the next element of the array into the GPR.**

**PUSH $GPR, $GPR, 8 # Add the value 8 (size of each element in the array) to the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

The first 3 bits (001) identify the instruction as a PUSH instruction. The next 5 bits (01000) encode the memory address 8, from which the value will be added to the value in the GPR.

**GET $AR, $GPR # Load the value in the AR (memory address) into the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

The first 3 bits (000) identify the instruction as a GET instruction. The next 5 bits (00110) encode the memory address 6, which holds the value in the AR (memory address).

**PUSH $GPR, $GPR, $zero # Add the value in the GPR to the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The first 3 bits (001) identify the instruction as a PUSH instruction. The next 5 bits (00000) encode the memory address 0, which is the address of the zero register.

**SET $AR, $GPR # Store the value in the GPR back into the AR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

The first 3 bits (001) identify the instruction as a SET instruction. The next 5 bits (00110) encode the memory address 6, which is the address of the AR.

**GET $AR, $GPR # Load the value at the memory location specified by the AR into the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

The first 3 bits (000) identify the instruction as a GET instruction. The next 5 bits (00110) encode the memory address 6, which is the address of the AR.

**Add the value in the GPR to the value in memory location 6.**

**PUSH $GPR, $GPR, 6 # Add the value in memory location 6 to the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

The first 3 bits (001) identify the instruction as a PUSH instruction. The next 5 bits (00110) encode the memory address 6, from which the value will be added to the value in the GPR.

**SET $GPR, 6 # Store the value in the GPR back into memory location 6**.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

The first 3 bits (001) identify the instruction as a SET instruction. The next 5 bits (00110) encode the memory address 6, where the value in the GPR will be stored.

**Increment the counter.**

**PUSH $GPR, $GPR, 7 # Add the value in memory location 7 to the GPR.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The first 3 bits (001) identify the instruction as a PUSH instruction. The next 5 bits (00111) encode the memory address 7, from which the value will be added to the value in the GPR.

**SET $GPR, 7 # Store the value in the GPR back into memory location 7.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The first 3 bits (001) identify the instruction as a SET instruction. The next 5 bits (00111) encode the memory address 7, where the value in the GPR will be stored.

**Jump back to the beginning of the loop.**

**JUMP Bitload # Jump to instruction 4.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

The first 3 bits (100) identify the instruction as a JUMP instruction. The next 5 bits (00010) encode the memory address 4, which is the start of the loop.

**END:**

**GET $GPR, 01000100 # This value is representing the $v0, 10 in MIPS.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

The first 3 bits (000) identify the instruction as a GET instruction. The next 5 bits (100100) encode the memory address 36, which holds the value 01000100.

# Datapath Diagram

detailed Datapath diagram for the program.

              +----------------+

              | Array       |

              | Memory    |

              +----------------+

                      |

                      |

            +--------------------+

            | Array       |   |

            | Element     | GPR|

            | Register    |---|

            +--------------------+

                         |

                      |

            +--------------------+

            |   Memory   | 6   |

            |   Location |-----|

            +--------------------+

                      |

                      |

               +--------------------+

            | Counter    |   |

            | Register   | GPR|

            |           |---|

            +--------------------+

                      |

                      |

            +--------------------+

            |   Memory   | 7   |

            |   Location |-----|

            +--------------------+

                      |

                      |

            +--------------------+

            | Array       |   |

            | Length      | GPR|

            | Register    |---|

            +--------------------+

                      |

                      |

            +--------------------+

            | Counter    |   |

            | Register   | GPR|

            |           |---|

            +--------------------+

                      |

                      |

            +--------------------+

               | Memory   | END |

            | Location |------|

            +--------------------+

                      |

                      |

            +--------------------+

            |   Array       |   |

            |   Element     | GPR|

            |   Address     |---|

            |   Register    | AR |

            +--------------------+

                      |

                      |

           +--------------------+

            |   Memory   |   |   |

            |   Location | GPR| AR|

            |         |---|---|

            +--------------------+

                      |

                      |

            +--------------------+

            |   Memory   | 6   |

            |   Location |-----|

            +--------------------+

                      |

                      |

            +--------------------+

            |   Memory   | 7   |

               |   Location |-----|

            +--------------------+

                      |

                      |

            +--------------------+

            | Memory   | PC |

            | Location |-----|

            +--------------------+

                      |

                      |

            +--------------------+

                      |

                      |

              +----------------+

              | System      |

              | Calls     |

              +----------------+

# Explanation of the components

The Array Memory stores the array that is being processed by the CPU.

The Array Element Register stores the current element of the array being processed.

The Memory Location 6 stores the intermediate result of the sum of the array elements.

The Counter Register stores the current counter value, which is being used to keep track of the

number of iterations of the loop.

The Memory Location 7 stores the next counter value.

The Array Length Register stores the length of the array.

The Array Element Address Register stores the memory address of the current element of the array.

The Memory Location END stores the address of the instruction to jump to when the counter reaches the length of the array.

The Program Counter (PC) stores the address of the next instruction to be executed.

The System Calls component is responsible for executing system calls, such as the SYSCALL instruction at the end of the code.

# Video Link

[Second\_project\_video\_1804010009\_1904010038\_1904010.mp4](https://drive.google.com/file/d/1ps5QPXIw69ynQQHh4O19XiXMtZbaTTjL/view?usp=share_link)