Zagazig University

Faculty of Engineering

Computer and Systems Dept

Report of

MIPS

Processor Implementation

For

Prof. Dr. Howaida Abdel Latif

- Team Members: -
  - كريم اسامة السيد عبد الرحمن محمود
  - مصطفى حسن جاب الله عطية محمود
  - مصطفى المهدى احمد عطوة
  - سامح سامي كمال غطاس
  - عبد الرحمن منصور السيد منصور

Students of Computer and Systems Dept.

Table of Content

Single-Cycle Processor

Introduction

- Overview

In the realm of computer architecture and digital design, the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture stands as a fundamental framework for understanding and developing modern processors. With its emphasis on simplicity, efficiency, and modularity, the MIPS architecture has been a cornerstone in both academic research and industrial applications.

This report presents the culmination of a project aimed at designing and implementing a Single-Cycle MIPS Processor. The

objective of this endeavor is to create a processor capable of executing instructions within a single clock cycle.

- Instruction Set Architecture

In this project, We have designed a simple 32-bit RISC processor with seven 32-bit general purpose registers: R1 through R31. R0 is hardwired to zero and cannot be written.

All instructions are only 32 bits.

There are three instruction formats, R-type, I-type, and J-type as shown below:

R-type format

6-bit opcode (Op), 5-bit two source register (Rs) & (Rt), 5-bit destination register, 5-bit shift value (shift) and 6-bit function field (f).

Op	Rs	Rt	Rd	shift	F

I-type format

6-bit opcode (Op), 5-bit two source register Rs & Rt and 16-bit constant value (immediate).

Op	Rs	Rt	immediate

J-type format

6-bit opcode (Op), 26-bit constant value (immediate).

Op	immediate

Register Use

For R-type instructions, Rs and Rt specify the two sources register numbers, and Rd specifies the destination register number. The function field F can specify functions of the instruction. We can ignore opcodes for R-type instructions.

For I-type instructions, Rs specifies a source register number, and Rt can be a second source or a destination register number. The immediate constant is 16-bits because of the fixed-size nature of the instruction. The size of the immediate constant is suitable for our uses. The 16-bit immediate constant can be signed or unsigned depending on the opcode. The immediate constant is signed (range is -32768 to +32767).

The J-type format is used by J (jump). The 26-bit immediate is used for PC-relative Addressing and constant formation. The immediate constant is unsigned (range is 0 to 2^26-1).

Instruction Description

Opcodes 0 are used for R-type instructions. Opcode 10 is used for the J (jump) instruction. Opcodes 1 through 9 are used for I-type instructions. The 16-bit immediate constant is zero-extended for ANDI, ORI, and XORI. It is sign-extended for the remaining instructions.

Processor Architecture

Our processor is meticulously constructed from six main components: Arithmetic Logic Unit (ALU), Register File, Instruction Memory, Data Memory, Control Units, and Program Counter (PC).

These components are intricately interconnected through a data-path to ensure the precise execution of all instructions. In the following subsequent sections, we will provide comprehensive descriptions of each component, elucidating their functionality and

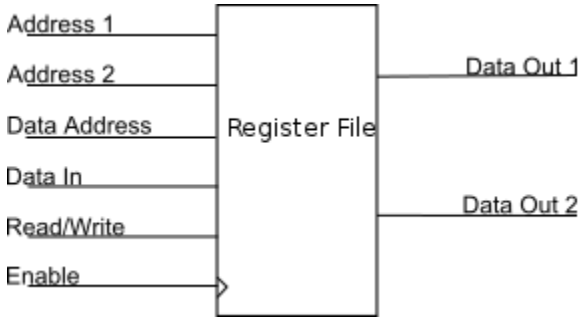
the intricate processes involved in executing instructions.

Single-Cycle Architecture

• Register File

Register File Overview

Our implemented Register File boasts seven 16-bit registers denoted as R1 to R31.
It has two read ports and one write port. Notably, R0 is hardwired set to zero, serving as a default value.



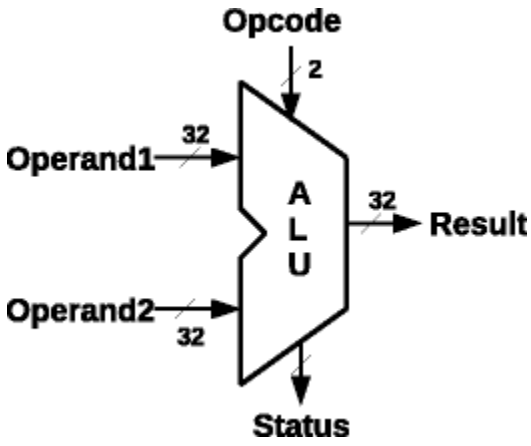
This register file features dual sources, a single destination input, and a data write input. Additionally, it is outfitted with two output buses to facilitate data output from the register file.

A pivotal control signal, RegWrite, is integrated to regulate the enabling or disabling of writing within the register file. Lastly, the Register File accommodates an input specifically designated for the Clock signal, ensuring synchronous operation.

• Arithmetic and Logic Unit (ALU)

ALU Overview

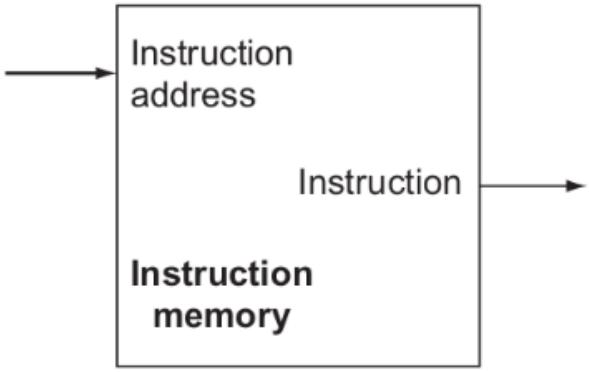
Embedded within the CPU, the Arithmetic and Logic Unit (ALU) executes arithmetic and logic operations on operands found in computer instruction words.



ALU has two inputs (A and B) and a single main output for result, so that ALU delivers the outcome of the operation.

It incorporates essential signal like Zero_Flag which used for branch instructions

• Instruction Memory



Instruction memory, depicted in the figure, functions as a ROM (Read-Only Memory) crucial for permanent data storage.

Stored data in ROM consists of eight hex numbers (words), providing essential instructions for the system.

It operates on a word-Addressable basis, with input A sourced from the next PC. ensuring access to the memory.

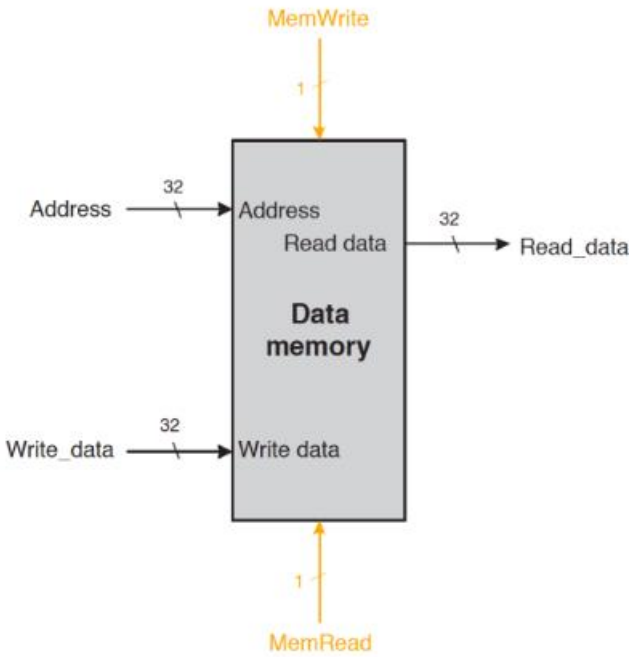
The output comprises 32 bits, representing:

- For R-type format: 6-bit opcode (Op), 5-bit destination register Rd and two 5-bit source registers Rs & Rt, 5-bit shift-amount shamt and 6- bit function field.

• For I-type format 6-bit opcode (Op), 5-bit destination register Rt, 5-bit source register Rs, and 16-bit immediate.

• For J-type format 6-bit opcode (Op) and 26-bit Immediate.

• Data Memory



In our Processor, Data Memory serves as the repository for storing data post-instruction execution, facilitating its retrieval for subsequent use. To ensure optimal performance and versatility, we opted for RAM (Random Access Memory) as our choice for Data

Memory. RAM's inherent speed and random access capabilities allow the processor to SWiftly access any location within the memory without the need to sequentially search through all Addresses.

Data manipulation within the Data Memory is orchestrated through the execution of LW (Load Word) and SW (Store Word) instructions, governed by the MemRd and MemWr control signals. Two primary inputs drive the Data Memory operations: firstly, the 32-bit result generated by the ALU determines the memory Address to be accessed; secondly, the Read Data 2, sourced from the Register File, is contingent upon the RegDst control signal, which designates whether it originates from Rd or Rt.

At the output stage of the Data Memory, a Multiplexer Selects between the ALU Result (when memory access is unnecessary) and the memory output (during data loading), guided by the MemtoReg control signal.

• Program Counter

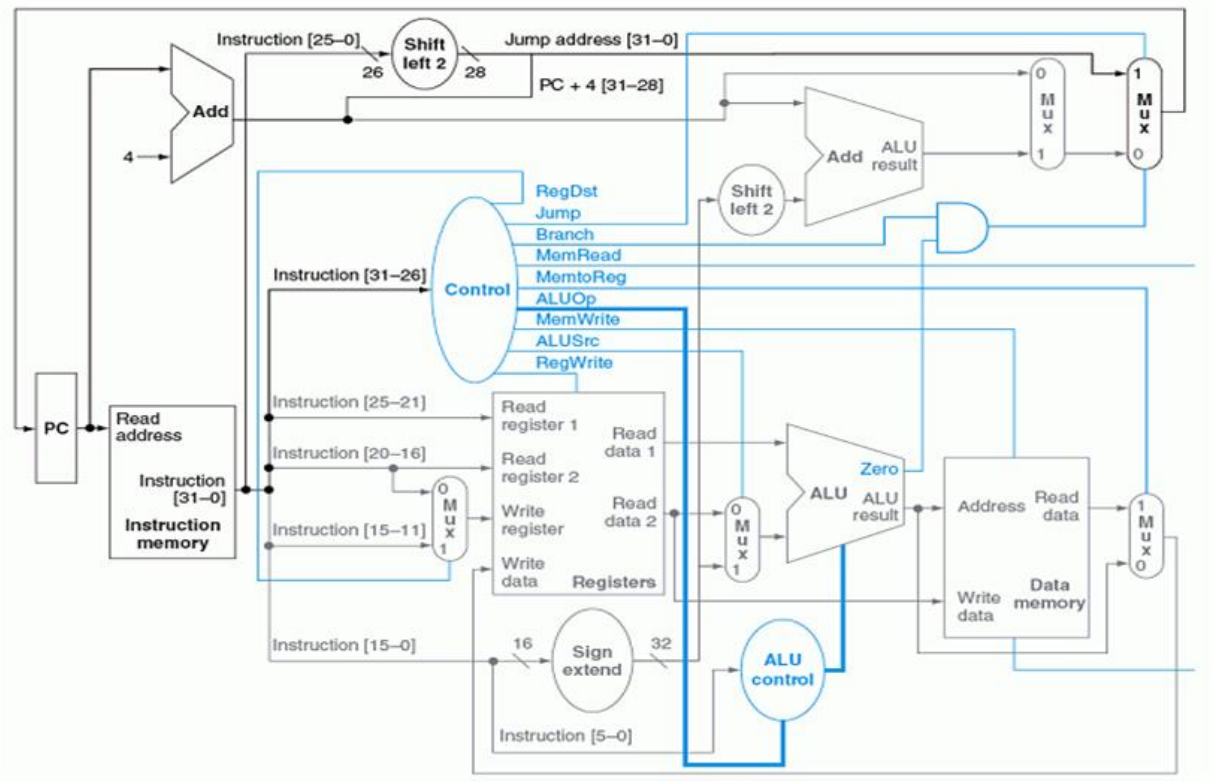
PC Overview

The Program Counter (PC) Block is an integral component of our architecture, housing a dedicated 16-bit register responsible for storing the Address of the current instruction residing in the Instruction Memory, awaiting execution. In Addition to its primary function as a storage unit for instruction Addresses, the PC Block encompasses a datapath tailored to handle specific instructions like Branch instructions (BEQ, BNE, BLT, BGE), Jump instructions (J and JAL), and Jr instruction.

• Data Path

interconnect components to construct the

complete processor, achieved through what we refer to as the Data Path.



• Control Units

Control units generate required signals which determine suitable data path to execute each instruction in the processor.

We have 2 control units each one generates special signals as follow:

1- Main Control Unit

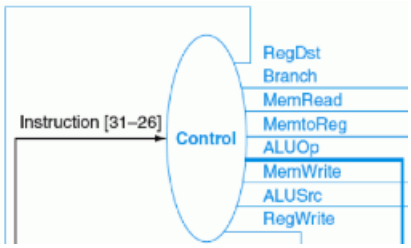
Signals: (RegWr, RegDst, Jump, branch, AluOp, ALUSrc, MemRd, MemWr, MemtoReg).

2- ALU Control Unit

Signals: (ALU-Ctrl).

• Main Control Unit

Overview



It is the unit responsible for Data Path signals. It has 1 input which is 6-Bit Op code and 9 outputs signals as follow:

RegWr: Enable/disable writing in Register File.

RegDst: Determine either Rt or Rd will be destination register.

J: Responsible for jumping to specific index of instruction.

Branch: Responsible for the offset instruction from next instruction.

ALUOp: Used for determination operation in R-type format.

ALUSrc : Determine either Rt or Imm will be directed to ALU.

MemRd : Enable loading from the Data Memory.

MemWr : Enable storing in the Data Memory.

MemtoReg : Selects either Data Memory output or the ALU result will be written back in the Register File

Instructions Truth Table

Instruction	OP	Reg Dst	J	Br	MEM Rwite	MEM TO REG	ALU OP	Mem Rwite	ALU Src	Reg Rwite
R-type	0	1	0	0	0	0	OP	0	0	1
ANDI	000001	0	0	0	0	0	OP	0	1	1
ORI	000010	0	0	0	0	0	OP	0	1	1
ADDI	000011	0	0	0	0	0	OP	0	1	1
SUBI	000100	0	0	0	0	0	OP	0	1	1
SLTI	000101	0	0	0	0	0	OP	0	1	1
NORI	000110	0	0	0	0	0	OP	0	1	1
LW	000111	0	0	0	1	1	OP	0	1	1
SW	001000	0	0	0	0	0	OP	1	1	0
BEQ	001001	0	0	1	0	0	OP	0	0	0
BNE	001010	0	0	1	0	0	OP	0	0	0
J	001011	0	1	0	0	0	OP	0	0	0

• ALU Control Unit

Overview

It is the unit responsible for ALU operations.

It has 2 inputs which are the 6-Bit Op code and 6-Bit Function, and 1 output 6-bit.

- Simulation and Testing

Sorting Array Test Code

Selection Sort is one of best algorithms for sorting Array that has few data, in addition it's simple to use, so We'll choose it for testing our MIPS Processor.

Code: -

i = 0; **j = 0;** **n = 6;** ***a = 4;** **Tmp = 20;**	**addi s0 zero 0;** **addi s1 zero 0;** **addi s2 zero 6;** **addi s3 zero 4;** **addi t0 zero 20;**

while(i<n){	L2 : slt t1 s0 s2; beq zero t1 L1;
a[i] = tmp;	addr s4 s3 s0; sw t0 0(s4);

tmp = tmp-1;
i = i+1;
}
	subi t0 t0 1;
	addi s0 s0 4;
	j L2

i = 0; while(i<n){	L1 : addi s0 zero 0; L7 : slt t1 s0 s2; beq zero t1 L3;
j = i+1 while(j<n){	addi s1 s0 4; L6 : slt t1 s1 s2; beq zero t1 L4;

if(a[j] < a[i]){	addr t2 s3 s1;
	addr t3 s3 s0;
	lw s4 0(t2);
	lw s5 0(t3);
	slt t1 s4 s5;
	beq zero t1 L5;

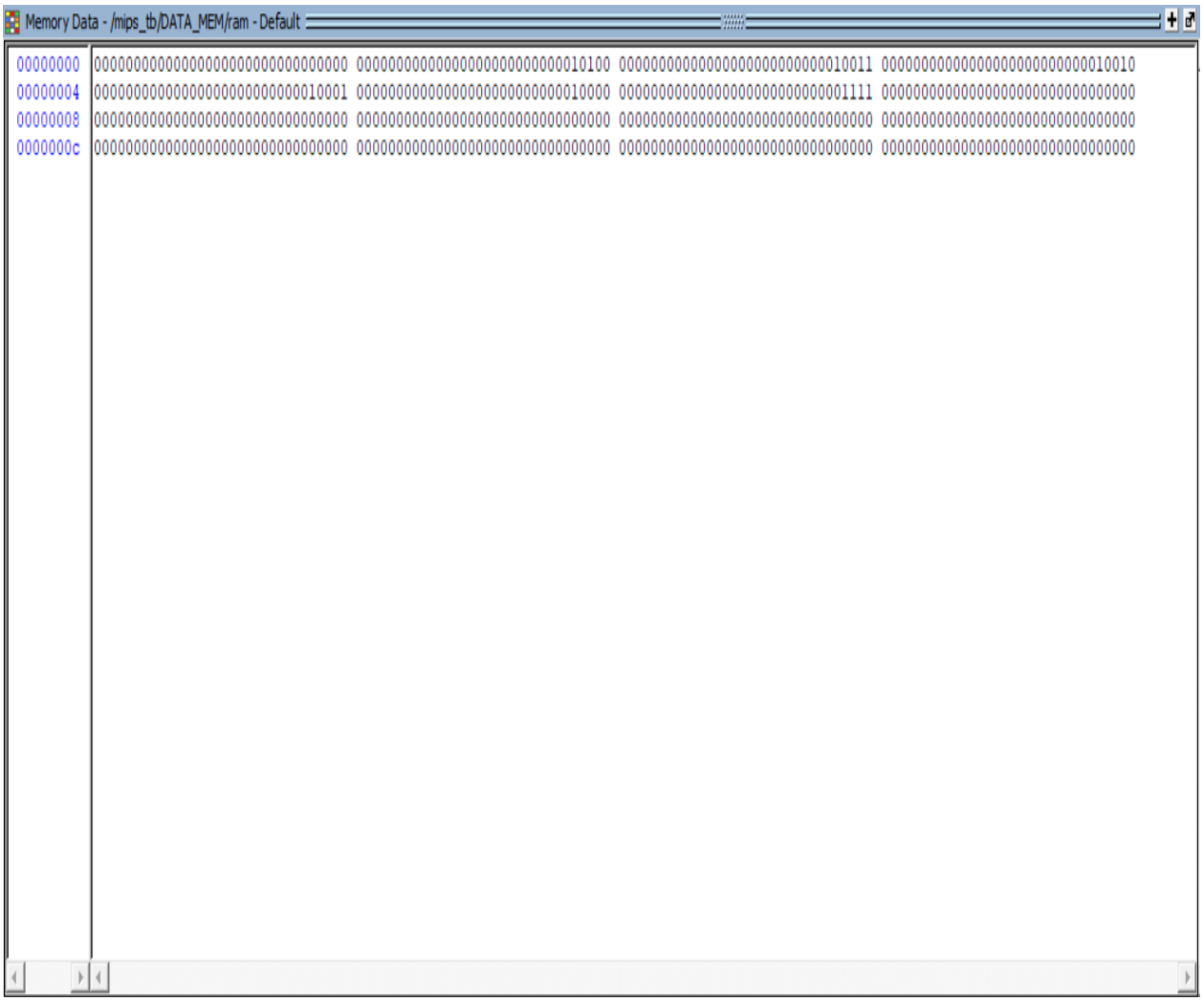
tmp = a[j];
a[j] = a[i];
a[i] = tmp;
}
	addr t0 zero s4;
	addr s4 zero s5;
	addr s5 zero t0;
	sw s4 0(t2);
	sw s5 0(t3);

j = j+1; }	L5 : addi s1 s1 4; j L6

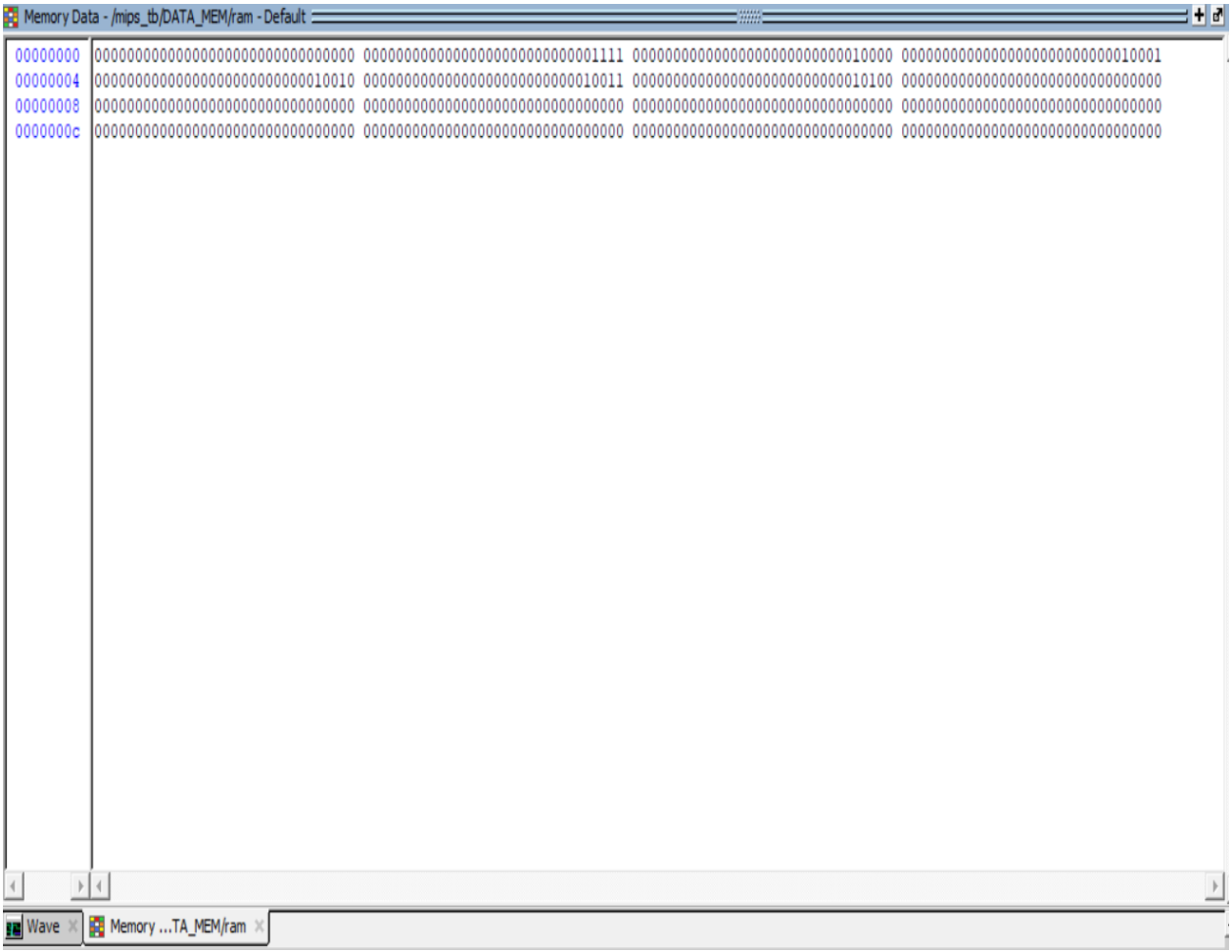
I = i+1; **}**	**L4 addi s0 s0 4;** **j L7;** **L3 : j L3;**

Test-Bench: -

- **Data-Memory Before Sorting: -**



- **Data-Memory After Sorting: -**


Memory Data - /mips_tb/DATA_MEM/ram - Default
00000000 00000000000000000000000000000000 00000000000000000000000000001111 00000000000000000000000000010000 00000000000000000000000000010001
00000004 00000000000000000000000000010010 00000000000000000000000000010011 00000000000000000000000000010100 00000000000000000000000000000000
00000008 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000000c 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
Wave
Memory ...TA_MEM/ram