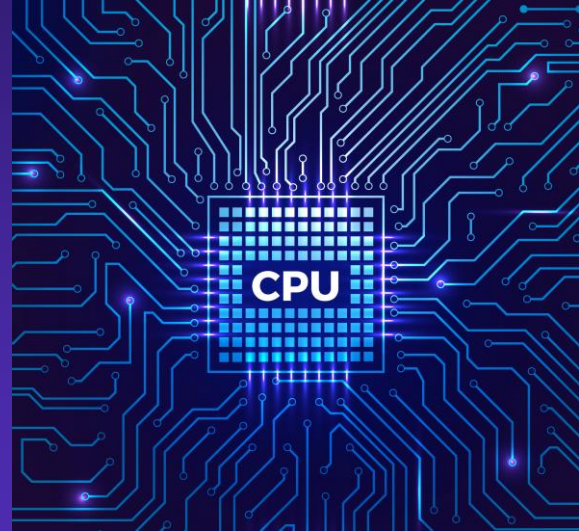


# Single cycle mips processor

x x x x x  
x x x x x  
x x x x x  
x x x x x



# Agenda

01

**What is MIPS?**

02

**synthesis**

02

**Architecture.**

04

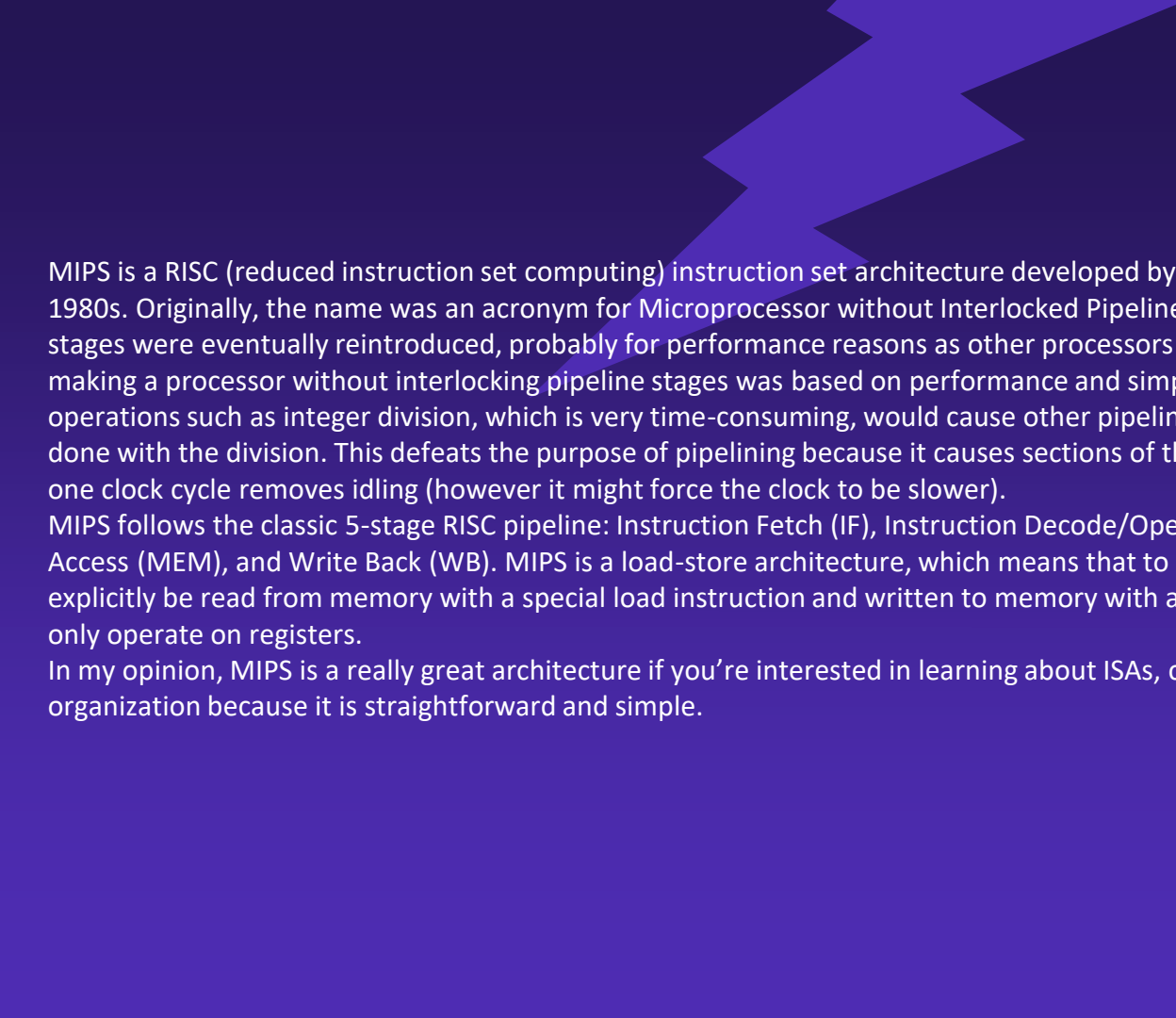
**Functional verification.**

x x x x x  
x x x x x  
x x x x x  
x x x x x  
x x x x x

# What is MIPS?

× × × × ×  
× × × × ×  
× × × × ×  
× × × × ×  
× × × × ×





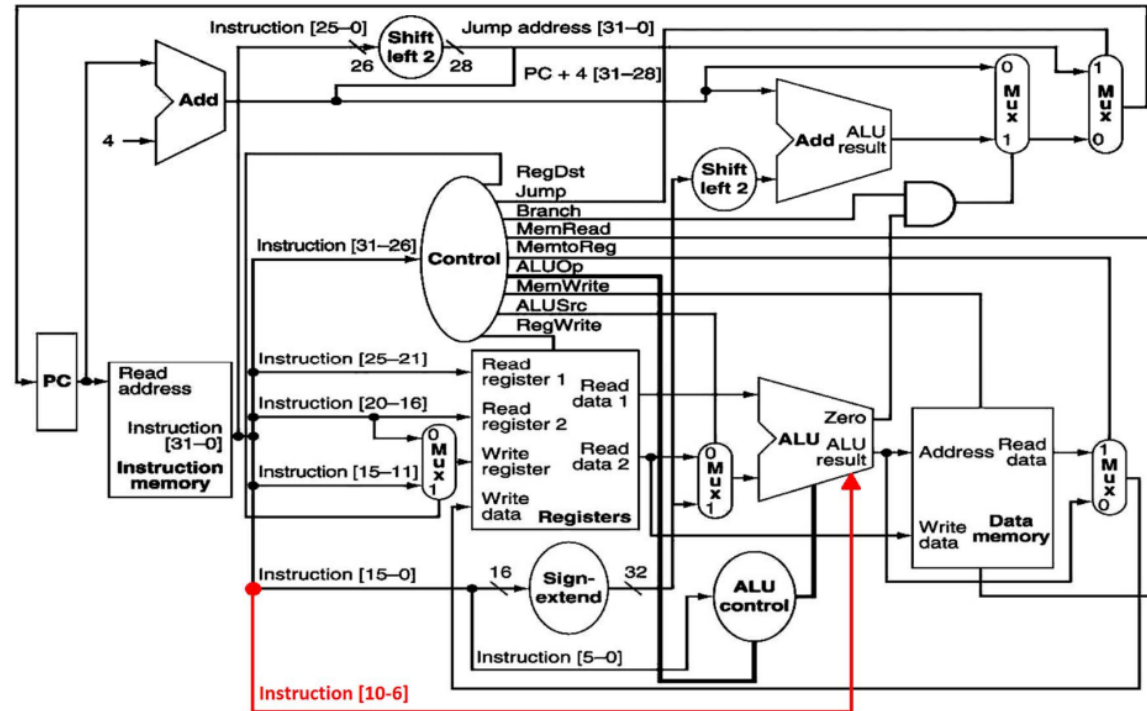
MIPS is a RISC (reduced instruction set computing) instruction set architecture developed by several Stanford researchers in the mid 1980s. Originally, the name was an acronym for Microprocessor without Interlocked Pipeline Stages, but interlocks between pipeline stages were eventually reintroduced, probably for performance reasons as other processors became more advanced. The decision for making a processor without interlocking pipeline stages was based on performance and simplicity of design. With interlocks, operations such as integer division, which is very time-consuming, would cause other pipeline phases to wait until the execute unit was done with the division. This defeats the purpose of pipelining because it causes sections of the processor to idle. Reducing all phases to one clock cycle removes idling (however it might force the clock to be slower).

MIPS follows the classic 5-stage RISC pipeline: Instruction Fetch (IF), Instruction Decode/Operand Fetch (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). MIPS is a load-store architecture, which means that to do arithmetic on data, values must explicitly be read from memory with a special load instruction and written to memory with a store instruction; arithmetic instructions only operate on registers.

In my opinion, MIPS is a really great architecture if you're interested in learning about ISAs, computer architecture, and computer organization because it is straightforward and simple.

# Architecture

x x x x x  
x x x x x  
x x x x x  
x x x x x  
x x x x x



This MIPS support the following instructions :

Instruction	Format	op	funct
<b>add</b>	R	0 <sub>10</sub>	32 <sub>10</sub>
<b>addi</b>	I	8	
<b>and</b>	R	0	36

<b>lw</b>	I	35	
<b>nor</b>	R	0	39

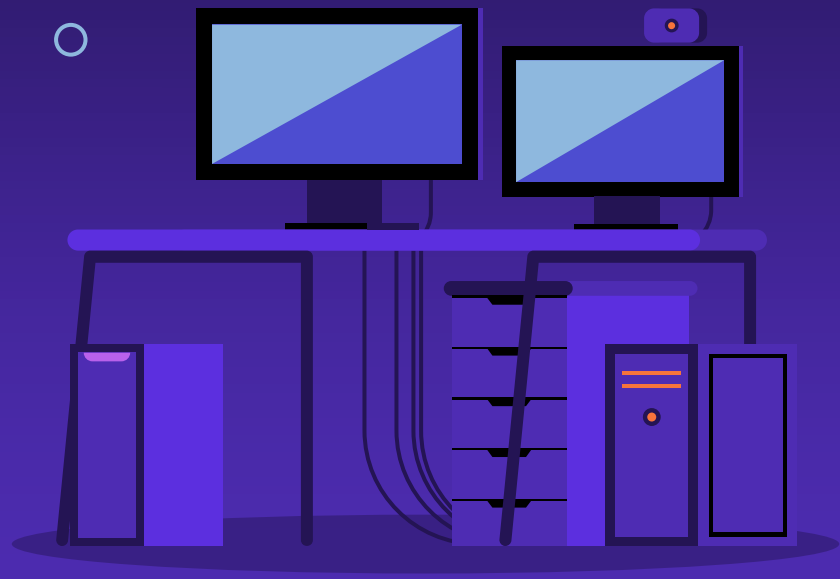
<b>j</b>	J	2	
----------	---	---	--

<b>beq</b>	I	4		<b>sll</b>	R	0	0
------------	---	---	--	------------	---	---	---

<b>or</b>	R	0	37
-----------	---	---	----

<b>sra</b>	R	0	3
<b>srl</b>	R	0	2
<b>sub</b>	R	0	34
<b>sw</b>	I	43	

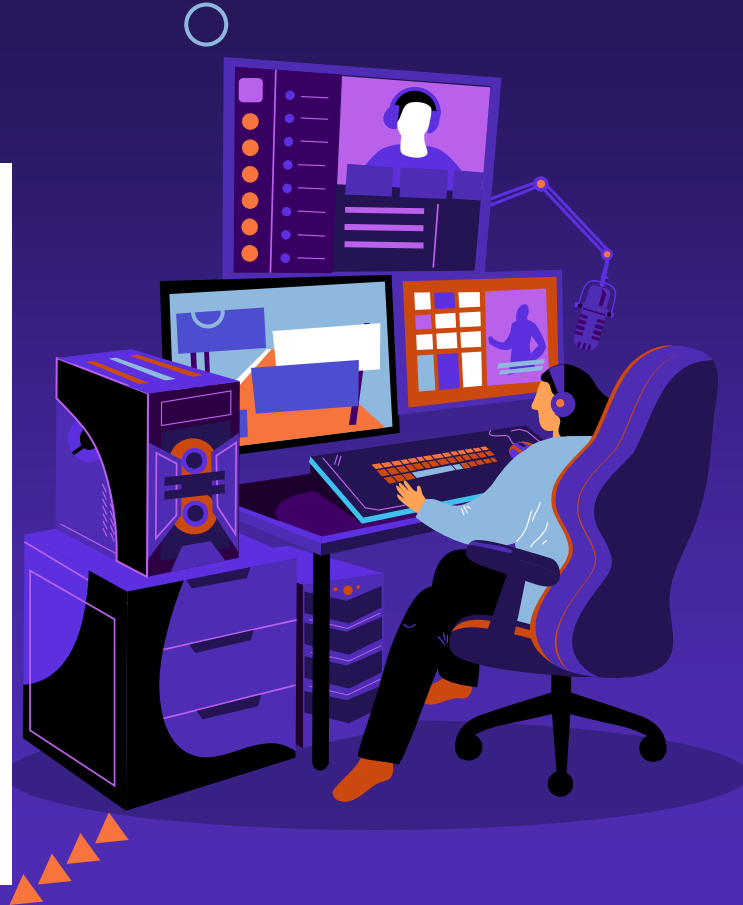
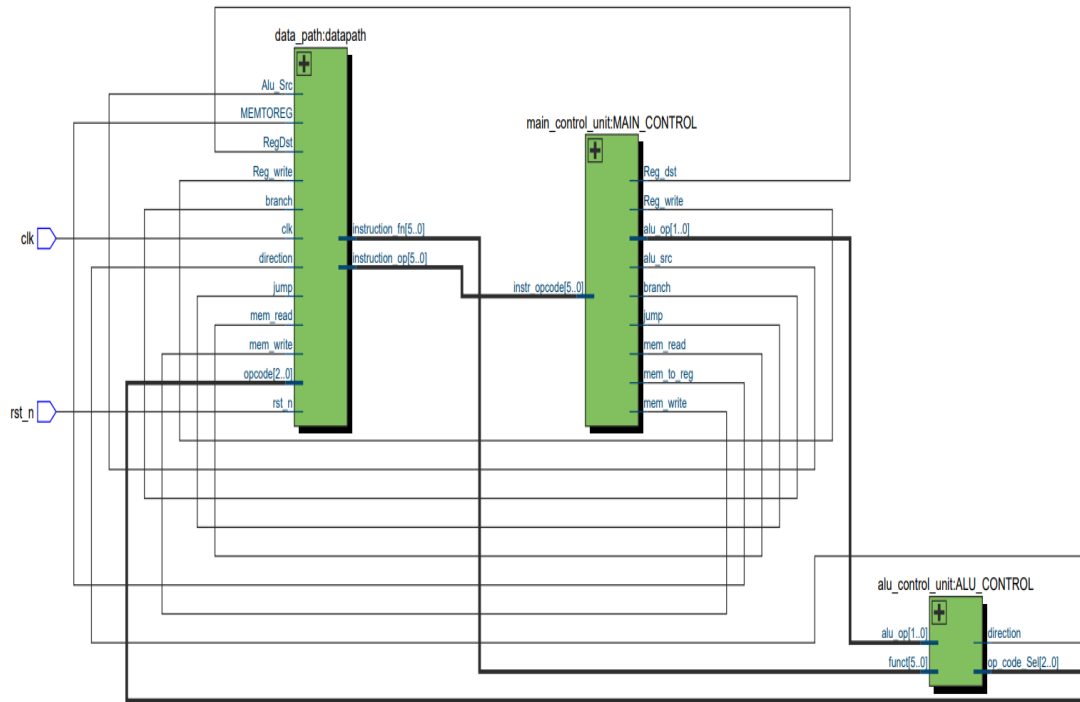
It also supports multiplication through the alu



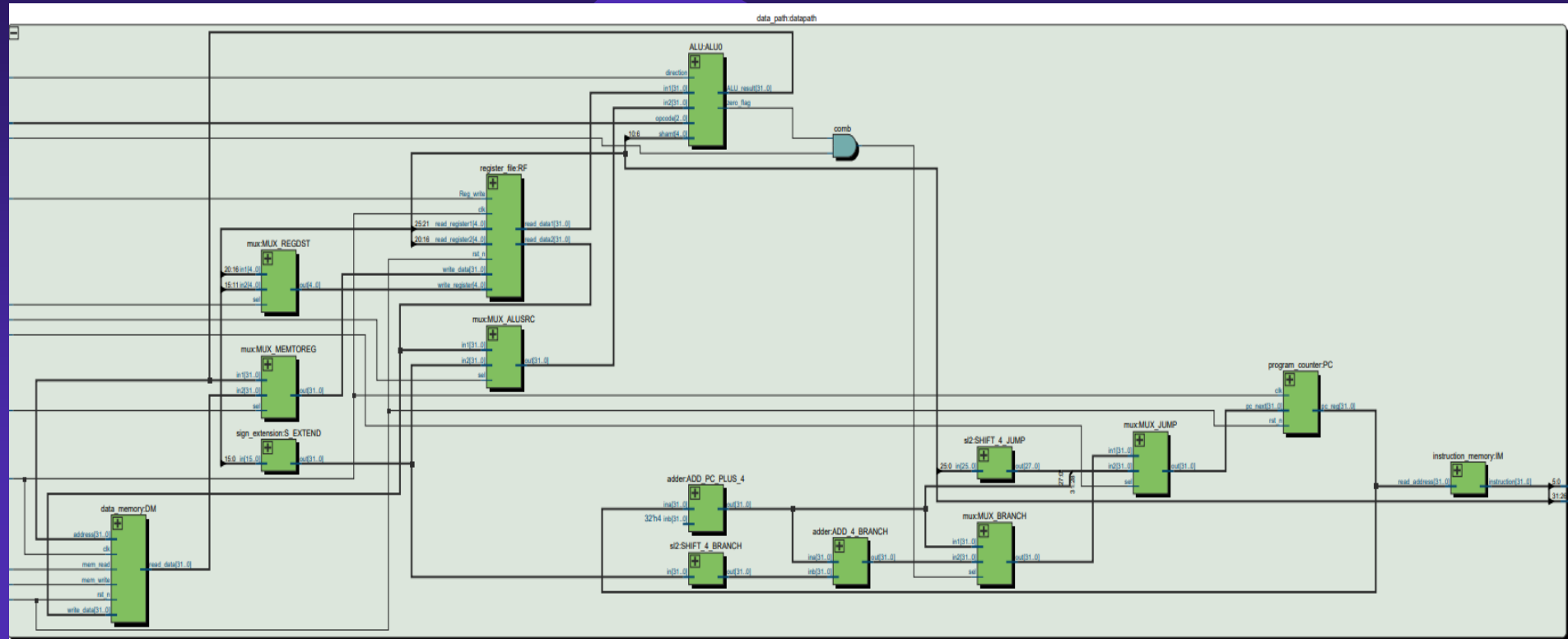
# synthesis



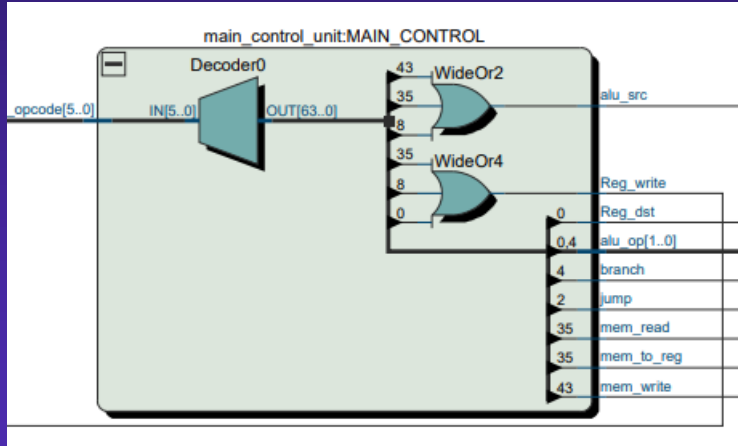
# Top module



# Data path



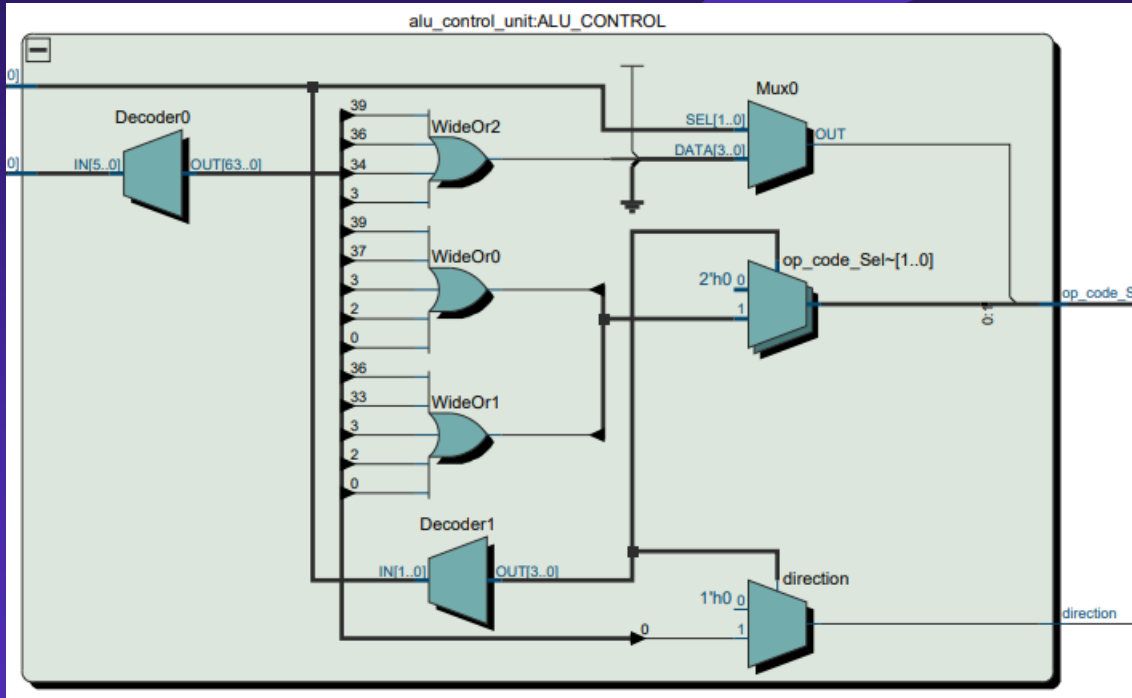
# Main control unit



× × × × ×  
× × × × ×  
× × × × ×  
× × × × ×  
× × × × ×

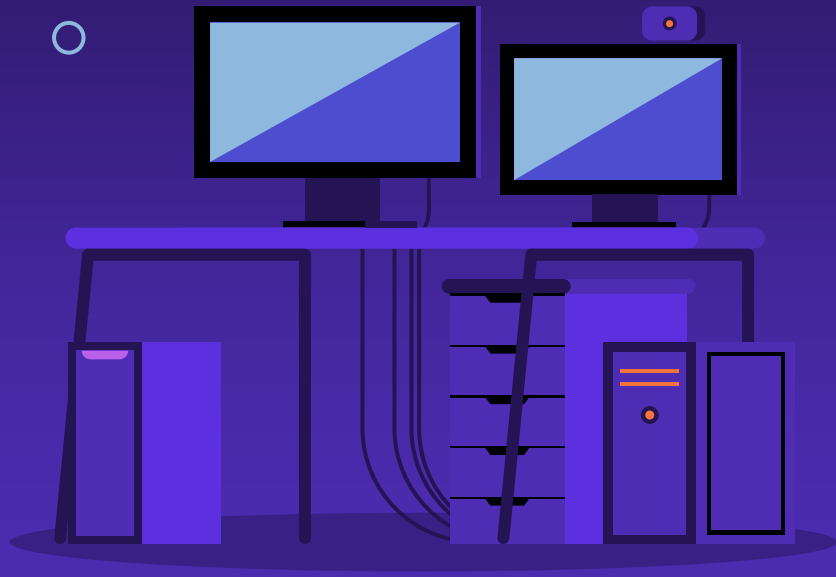


# Alu control unit



× × × × ×  
× × × × ×





# Functional verification

```
add $s0, $0, $0
addi $s0, $0, 7
add $s1, $0, $0
addi $s1, $0, 1
beq $s1, $0, 3
mul $s1, $s1, $s0
addi $s0, $s0, -1
J 4
sw $s1, 0($0)
```

**This program 1 that I used to test the MIPS  
and it calculates the factorial of 7**



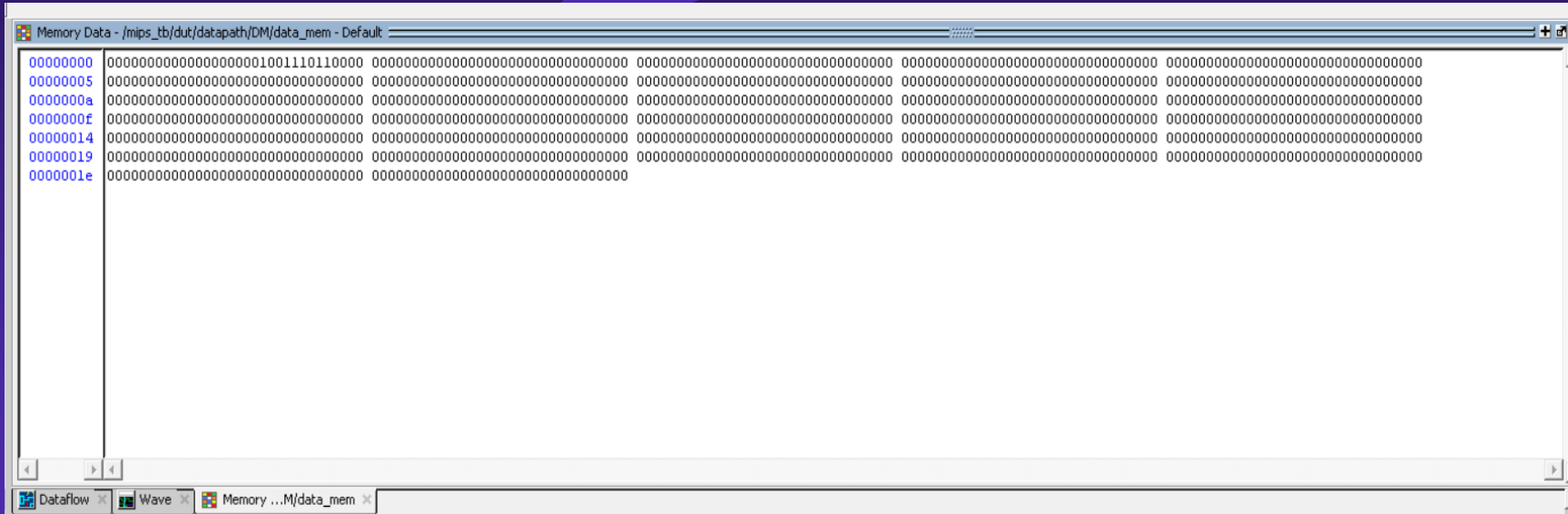
# Instruction memory



x x x x x  
x x x x x  
x x x x x  
x x x x x



# data memory



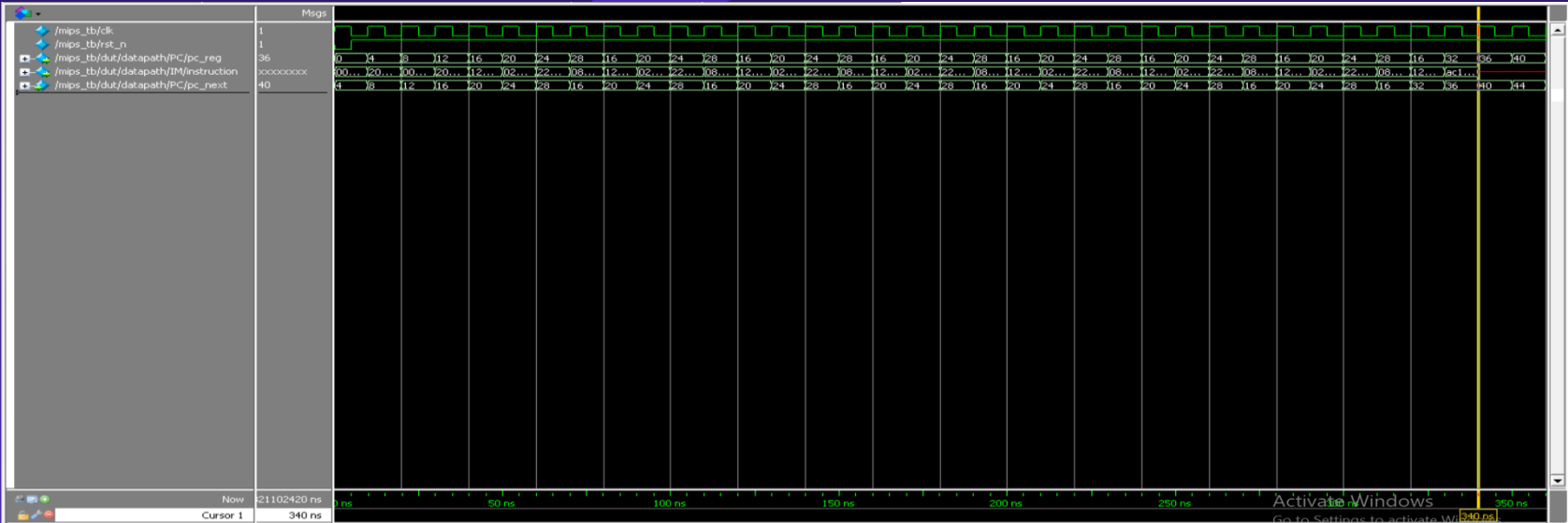
```
Memory Data - /mips_tb/dut/datapath/DM/data_mem - Default
00000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000005 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000000a 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000000f 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000014 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000019 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000001e 00000000000000000000000000000000 00000000000000000000000000000000
```

x x x x x  
x x x x x  
x x x x x  
x x x x x






# Snippet that shows how many cycles the program takes




× × × × ×  
× × × × ×  
× × × × ×  
× × × × ×





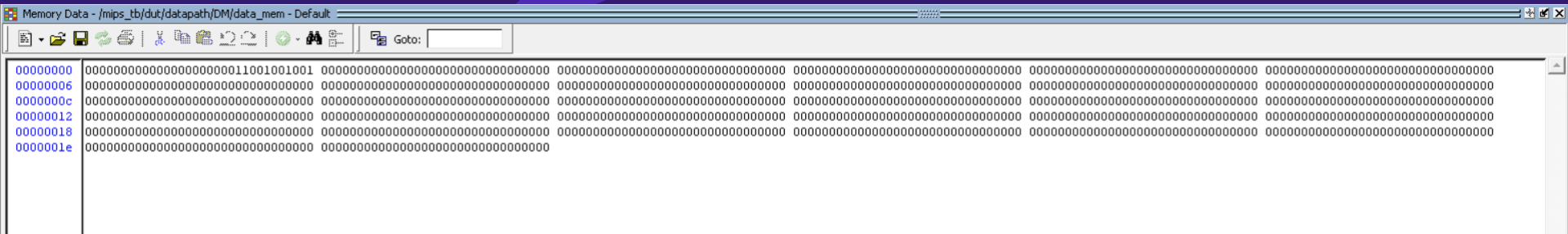
```
ADDI $t1 $t0 0x1926  
SLL $t0 $t1 1  
SRL $t0 $t1 1  
SRA $t0 $t1 2  
SW $t0 0($zero)
```

**This is program 2 that I used to test the MIPS  
and it tests the 3 shift instructions and sw**

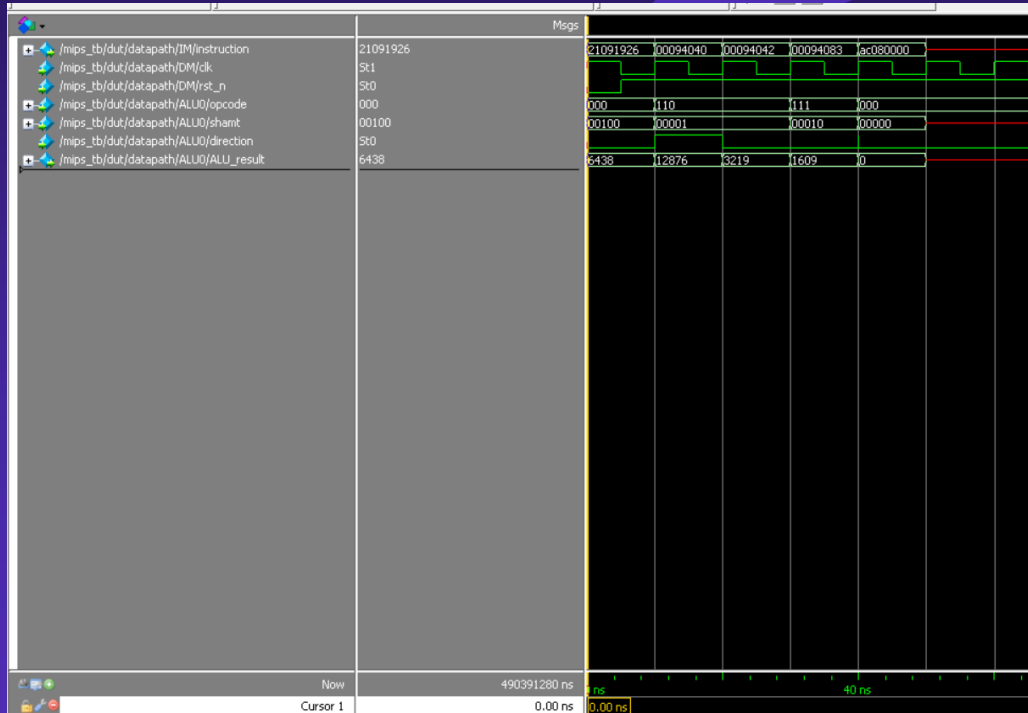




# data memory



## Snippet that shows how many cycles the program takes and the output of the Alu after every instruction



**Hint:** direction signal is high in SLL and low in SRL