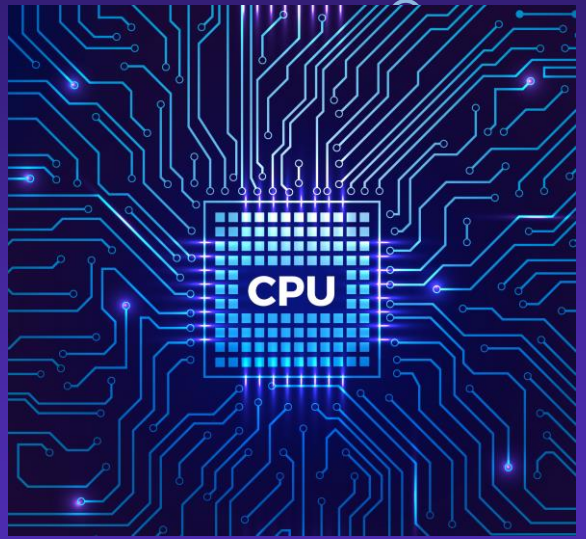
Single cycle mips processor



Agenda

01 What is MIPS?

02 Architecture.

02 synthesis

04 Functional verification.

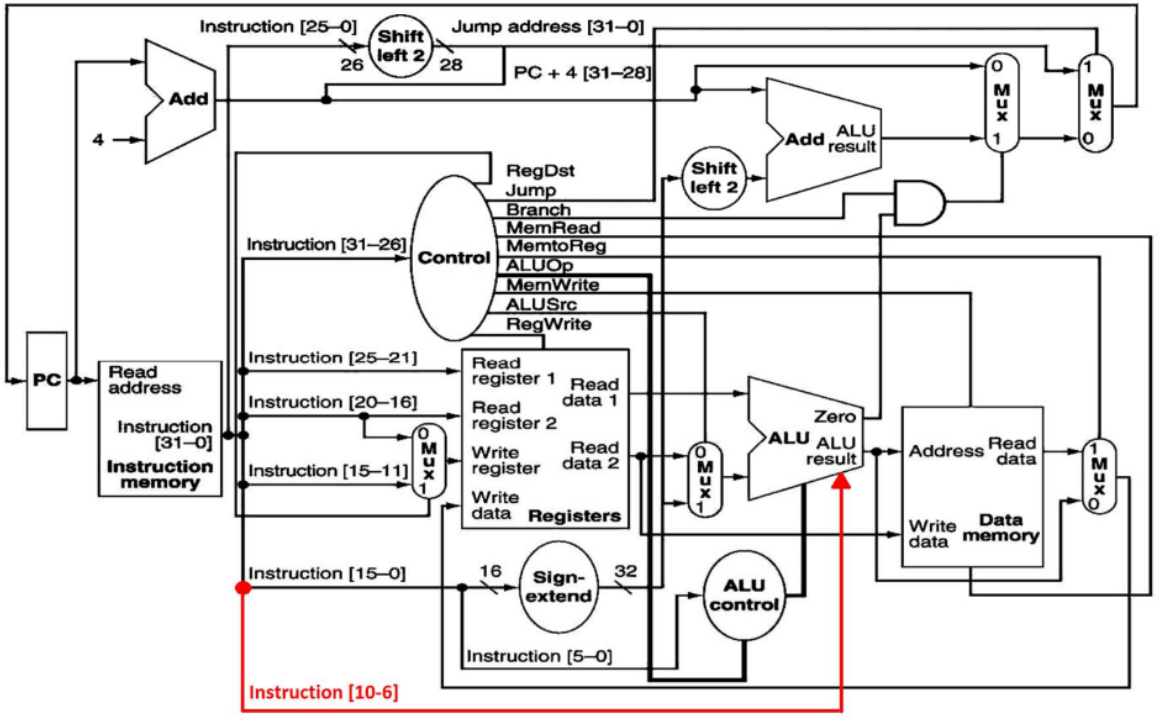
What is MIPS?

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.

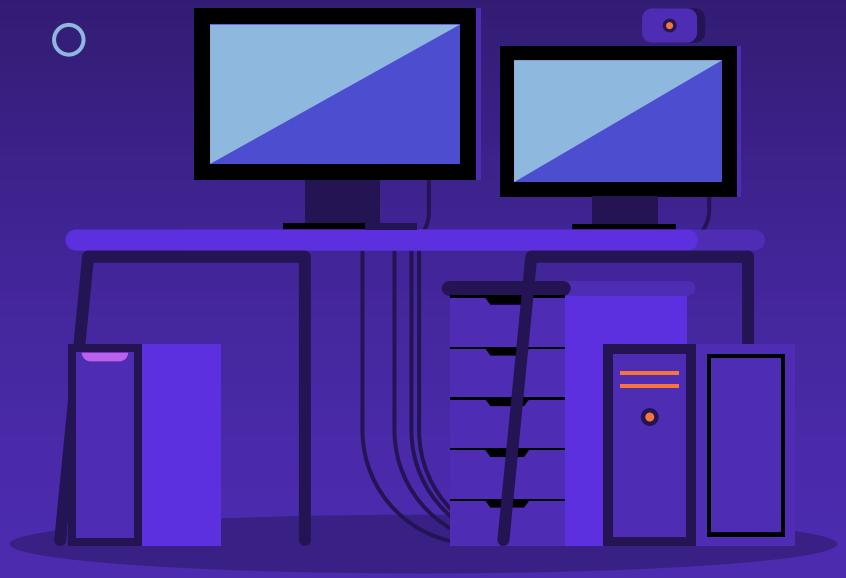
Architecture


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0 M u x 1
1 M u x 0
Add ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Instruction [10-6]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
16
Sign-extend
32
ALU control

This MIPS supports the following instructions :

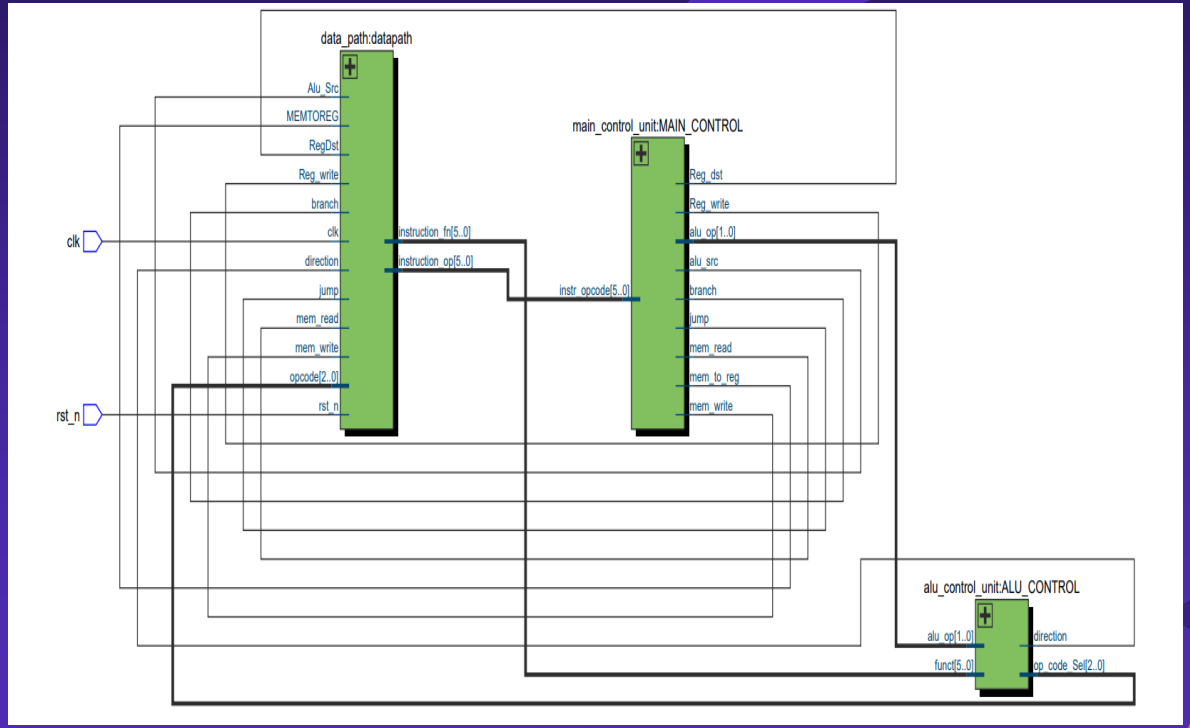
Instruction	Format	op	funct
add	R	0_{10}	32_{10}
addi	I	8	
and	R	0	36
lw	I	35	
nor	R	0	39
j	J	2	
beq	I	4	
or	R	0	37
sra	R	0	3
srl	R	0	2
sub	R	0	34
sw	I	43	
sll	R	0	0

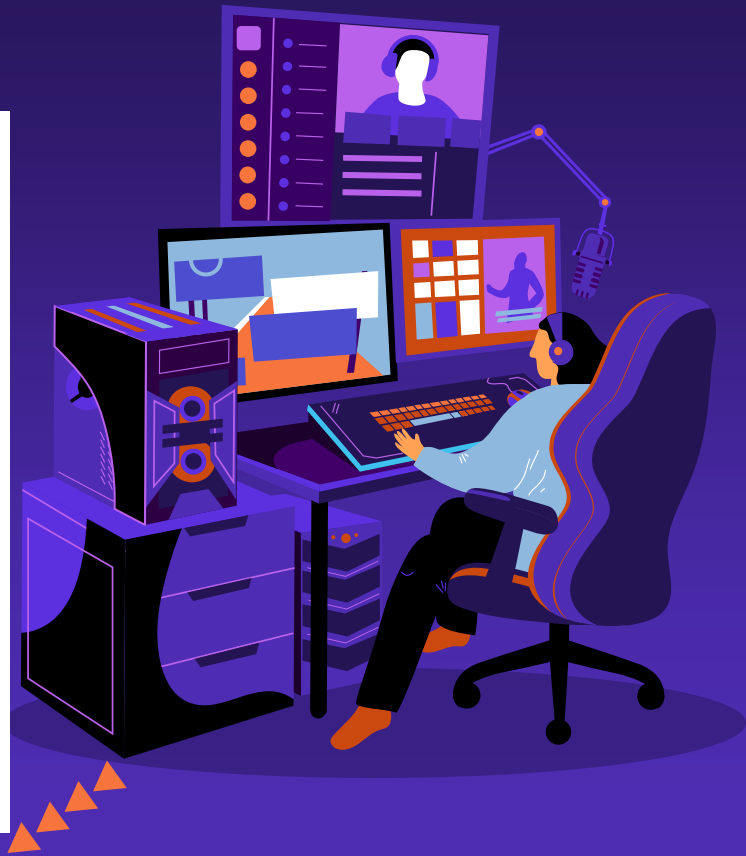
It also supports multiplication through the alu



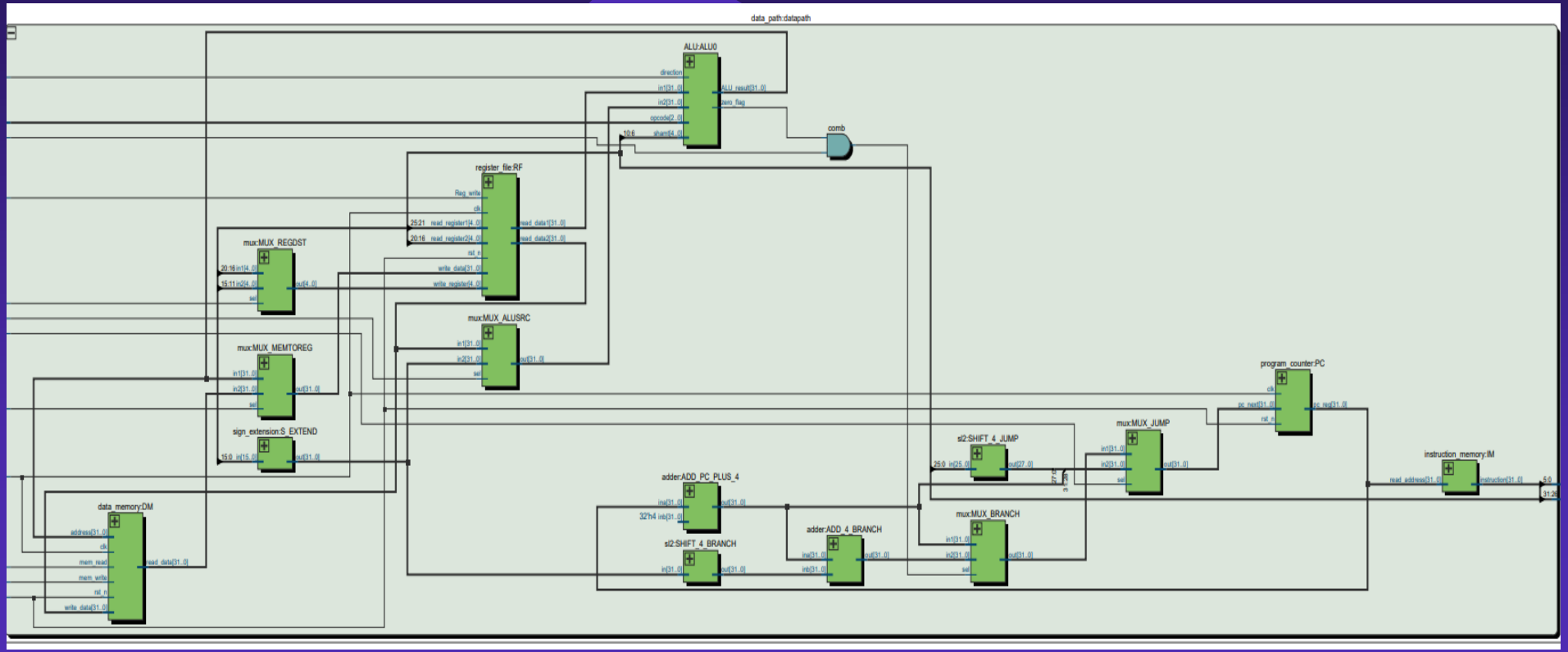
synthesis

Top module

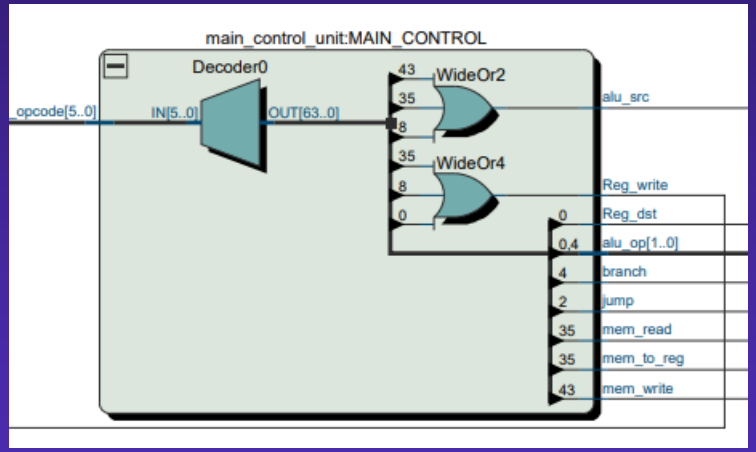

data_path:datapath
Alu_Src
MEMTOREG
RegDst
Reg_write
branch
clk
direction
jump
mem_read
mem_write
opcode[2..0]
rst_n
instruction_fn[5..0]
instruction_op[5..0]
main_control_unit:MAIN_CONTROL
instr_opcode[5..0]
Reg_dst
Reg_write
alu_op[1..0]
alu_src
branch
jump
mem_read
mem_to_reg
mem_write
alu_control_unit:ALU_CONTROL
alu_op[1..0]
funct[5..0]
direction
op_code_Sel[2..0]
clk
rst_n



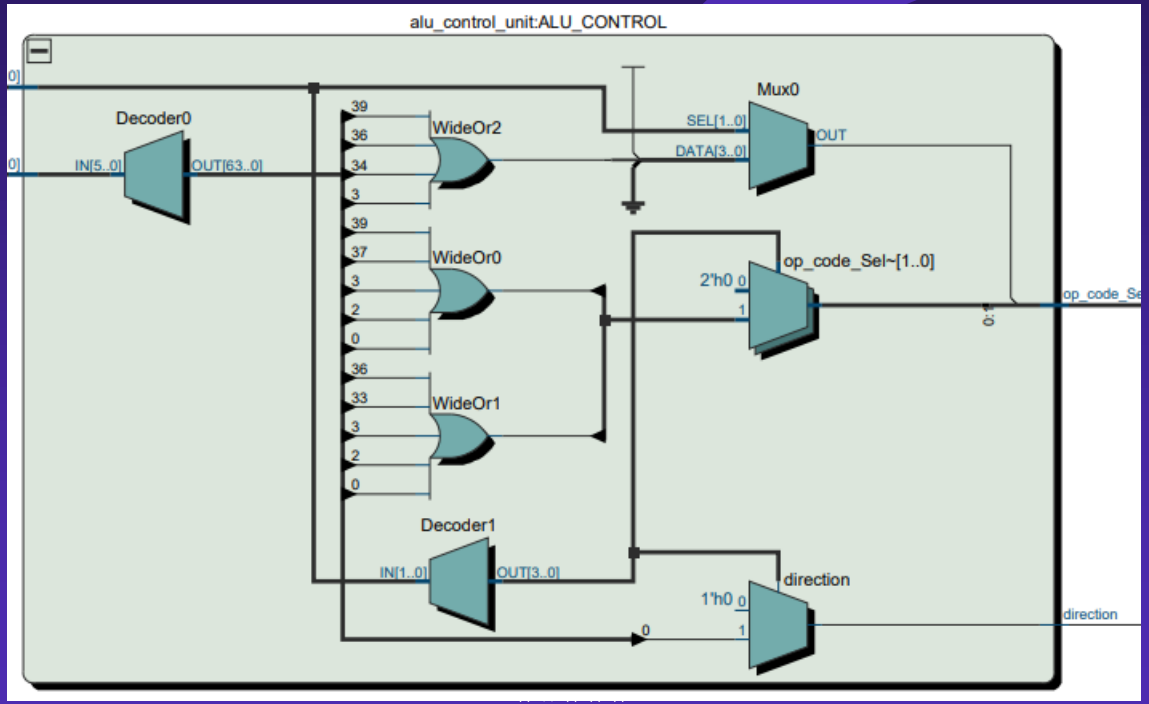
Data path

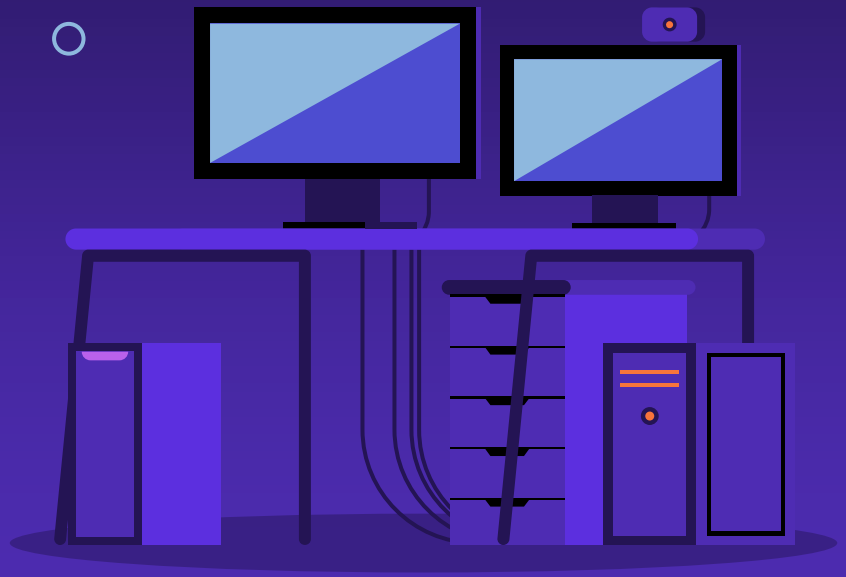

data_path:datapath
ALU:ALU0
comb
register_file:RF
mux:MUX_REGDST
mux:MUX_ALUSRC
mux:MUX_MEMTOREG
sign_extension:S_EXTEND
data_memory:DM
adder:ADD_PC_PLUS_4
sl2:SHIFT_4_BRANCH
adder:ADD_4_BRANCH
mux:MUX_BRANCH
sl2:SHIFT_4_JUMP
mux:MUX_JUMP
program_counter:PC
instruction_memory:IM

Main control unit



Alu control unit



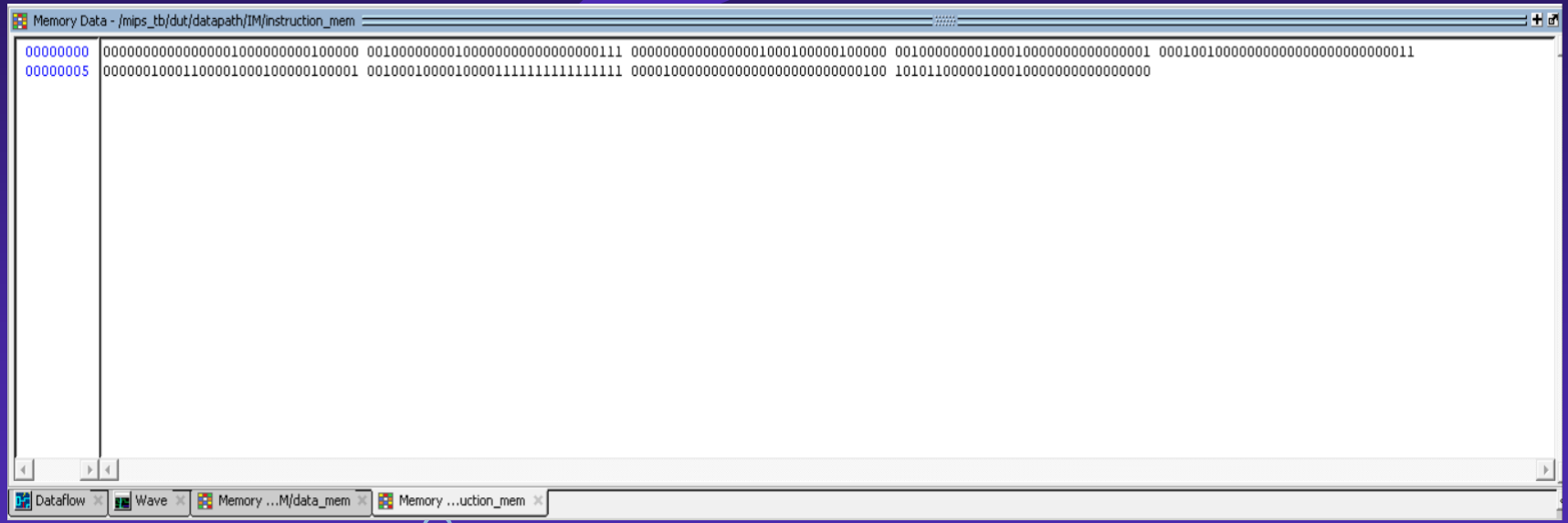


Functional verification

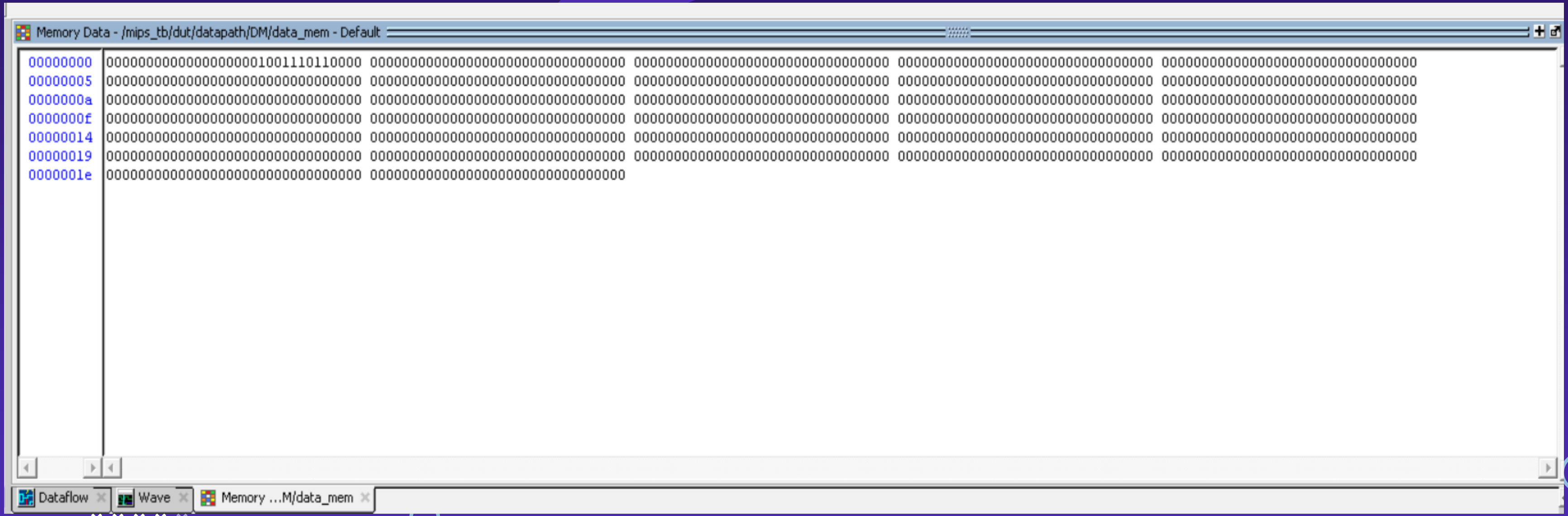
```
add $s0, $0, $0
addi $s0, $0, 7
add $s1, $0, $0
addi $s1, $0, 1
beq $s1, $0, 3
mul $s1, $s1, $s0
addi $s0, $s0, -1
J  4
sw $s1, 0($0)
```

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

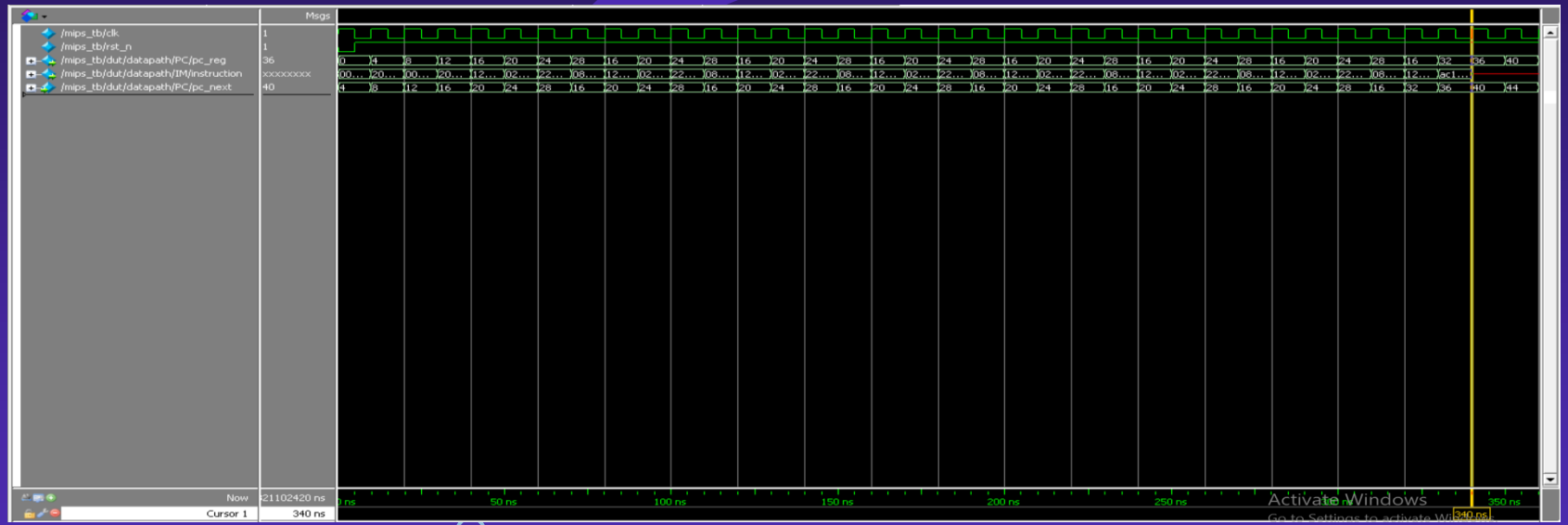
Instruction memory



data memory


Memory Data - /mips_tb/dut/datapath/DM/data_mem - Default
00000000 00000000000000000001001110110000 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
Dataflow
Wave
Memory ...M/data_mem

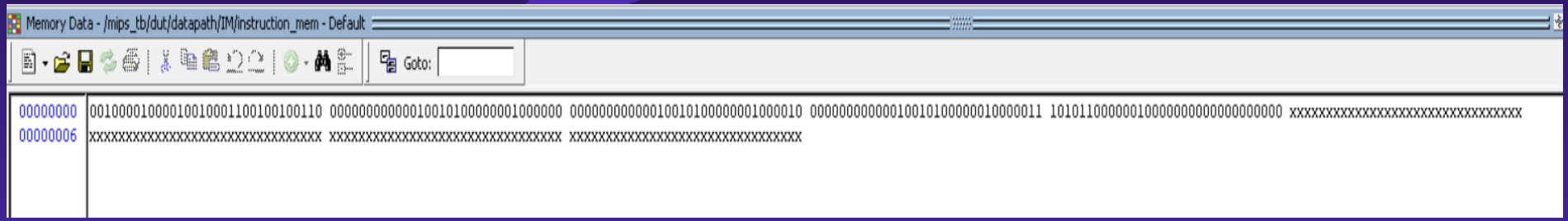
Snippet that shows how many cycles the program takes



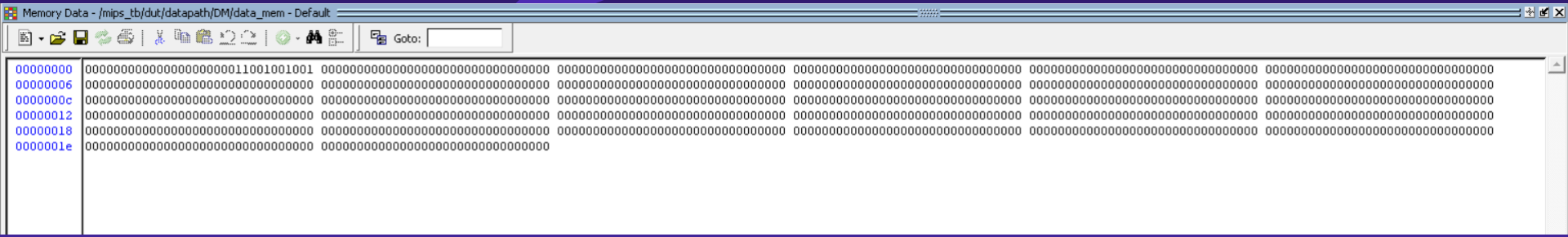
```
ADDI $t1 $t0 0x1926
SLL $t0 $t1 1
SRL $t0 $t1 1
SRA $t0 $t1 2
SW $t0 0($zero)
```

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

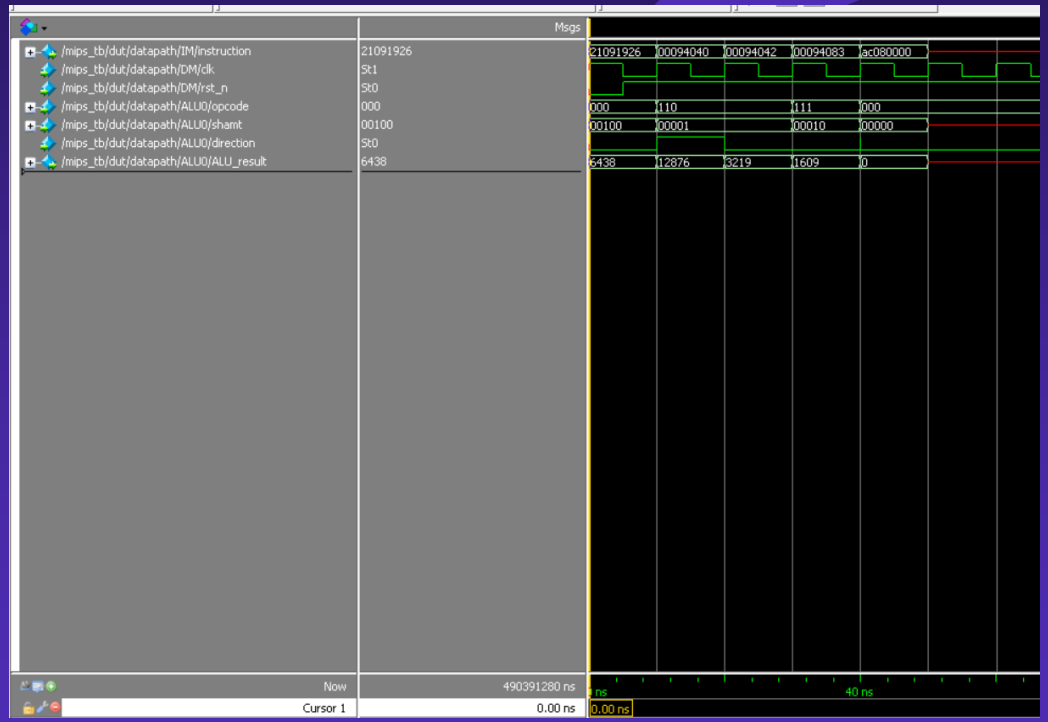
Instruction memory



data memory


Memory Data - /mips_tb/dut/datapath/DM/data_mem - Default
Goto:
00000000 00000000000000000000000011001001001 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000006 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000000c 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000012 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
00000018 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
0000001e 00000000000000000000000000000000 00000000000000000000000000000000

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