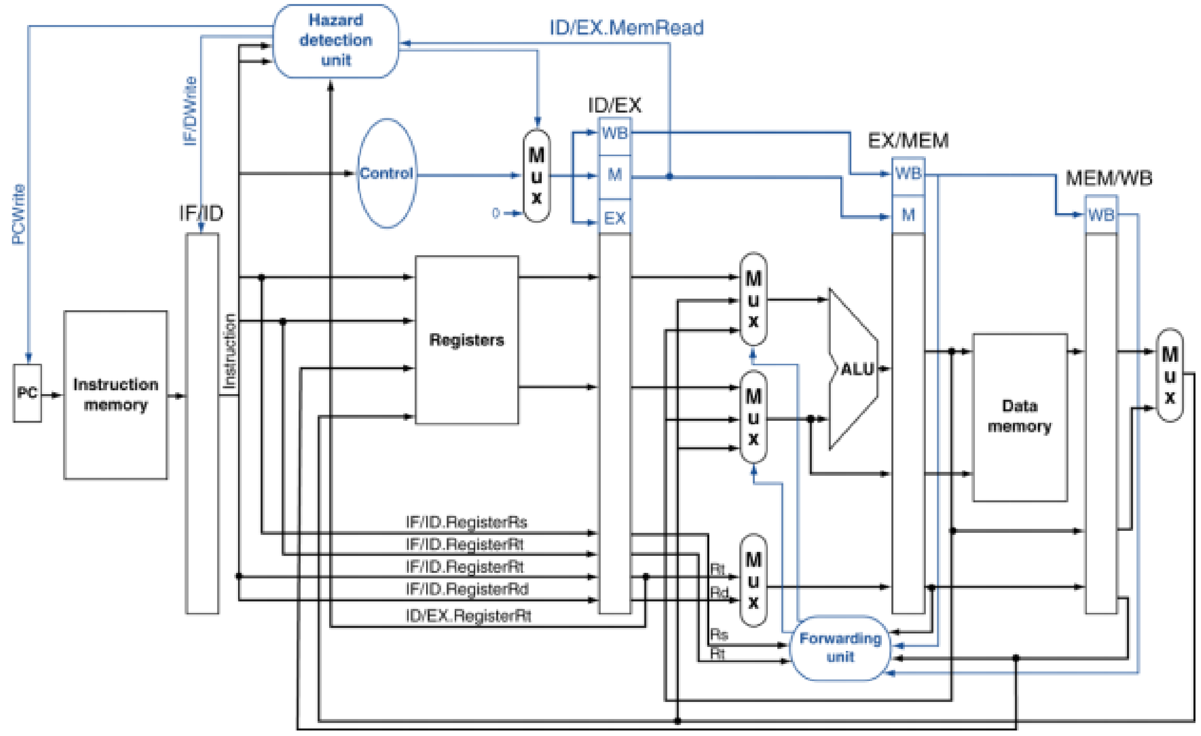
1. ***Pipeline concept.***

MIPS ARCHITECTURE The accompanying outline demonstrates the fundamental architecture of a MIPS-based framework. Basic MIPS Architecture Microprocessor without Interlocked Pipeline Stages (MIPS) is a RISC (Reduced Instruction Set Computing) architecture. Pipelined MIPS has five stages which are IF, ID, EX, MEM and WB. Pipelining means several operations in single data path at the same instant. Pipelining is used to enhance the capabilities of the RISC processor which is the reason for its utilization in this type of computer architecture. A multicycle CPU comprises of countless tasks. So if one task occurs, rather than waiting for the process to finish, at the same time another task is initiated in the same data path simultaneously without interfering with the previous task. The processes is thus divided into different pipelined stages. Following every clock a new operation is instigated in the pipeline stage to which the process is being fed to. The triggering is done without causing any interruptions to the past process. This makes simultaneous utilization of all stages in the data path possible. This thusly can increment the throughput of MIPS. 5-Stage Pipelined MIPS



MIPS processor has been executed utilizing five pipeline stages, which are Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory access (MEM) and Write Back (WB).The isolation of these stages is achieved by special registers known as pipeline registers. The aim of these registers is to isolate the stages of the instructions so that there is no inadmissible information because of various directions being executed all the while. They are named in the middle of each of these: IF/ID Register, EX/MEM Register and MEM/WB Register. The data path demonstrated in Fig. 0. is that of the MIPS pipelined processor.

1.1 Instruction Fetch (IF) .

The command relayed to the Program Counter (PC) to fetch the instruction from the cache memory is what instigates the primary pipelining operation of the IF stage. The storage of PC and Instruction for the successive clock cycle is done in the IF/ID pipelined register as RAM (Random Access Memory) Fig. 1. IF Stage representation IF stage for the most part relies on upon PC's represent value. On the basis of the PC value the processor gets the instructions from the cache and followed by which the Program Counter value is incremented by 1. Thus, the IF/ID register receives this information followed by which the information is relayed to the decoder unit. The Instruction Fetch (IF) stage operation has been represented in Fig. 1 .

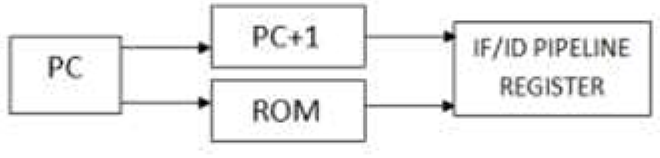


Fig1 .Instruction Fetch.

1.2 Instruction Decoder (ID)

ID Stage representation The Opcode is relayed to the decoder unit at the instant when the instruction is obtained from the IF stage. Instruction Decoder ID stage directs the controlling command to the various units of the MIPS processor examining the Opcode of the instructions. Thus the procurement of data from the MIPS registers is carried out by the Read register. The Branch unit is likewise incorporated into Instruction Decoder (ID) stage. The Input data of ID stage is received from IF stage as shown in Fig. 1.This decoding stage includes four different instructions: Register (R) type, Immediate (I)type, Jump (J) type and Input/Output (I/O) type instructions. Depending upon these instructions the function will be performed utilizing above mentioned formats. Fig.2. indicates Instruction Decode (ID) stage operation.

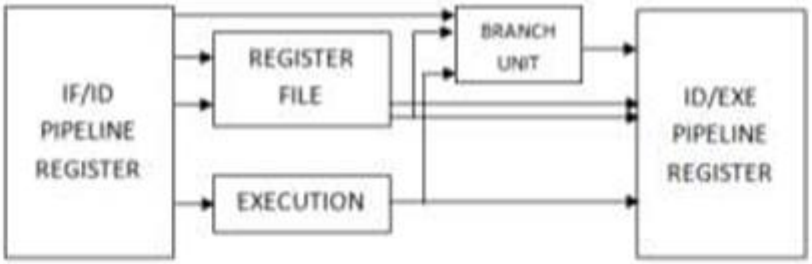


Fig2 Instruction Decoder

1.3 Execute (EX) .

Execution Stage (EX) representation Following the Instruction Decoder(ID), the instructions are sent to execute stage(EXE or EX). Execute (EX) stage performs Arithmetic and Logical Unit (ALU) processes. Execution of operations is the fundamental aspect of Execute (EX) stage, for instance arithmetic operations such as addition and difference and OR & AND. In particular, EX/MEM pipelined register receives the result upon the execution of specific instructions (i.e. FP ALU). Execute stage representation is shown in.

1.4 Memory Access &Input/Output (MEM).

Memory Access representation The storing and loading of values along with inputting and outputting data from the processor is the primary function of the memory access (MEM) stage. The outcome will be dispatched to the WB stage in a scenario where the instruction is neither memory nor IO instruction. After the result is figured the primary function is to store the data values in the destination register. The Memory Access (MEM) stage operation is demonstrated in Figure 4.

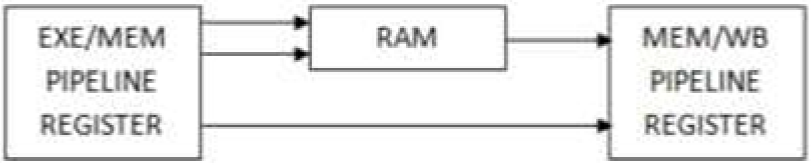


fig4 Memory Access.

1.5 Write Back (WB) Fig. 13. Write Back representation As per Fig. 5., the Write-Back (WB) operation is the final stage of the RISC based MIPS architecture which composes the result, store information and input data from and to the register . Writing the data that has been fetched from the MIPS register to the target register is the main aim of this stage.

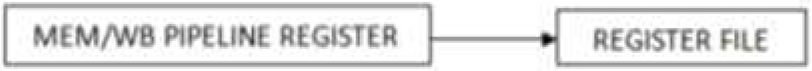
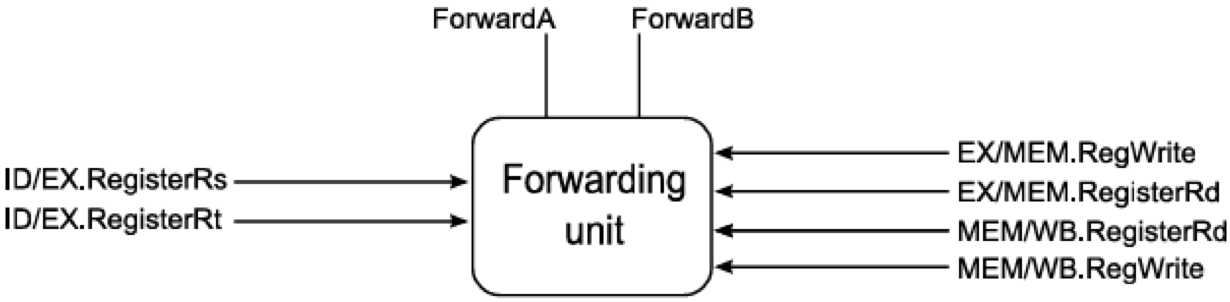


Fig5 Write Back

1.6 The forwarding unit

The forwarding unit takes a total of six input values, and produces two output values: Fig 6: The Forwarding Unit Interface The purpose of the forwarding unit is to guarantee that the instruction entering the EX stage of the pipeline receives the correct values for its register operands.

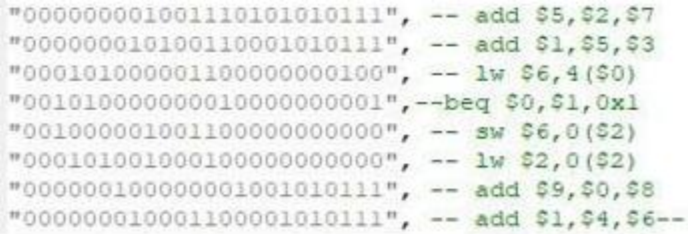


1.6 The hazard detection unit □

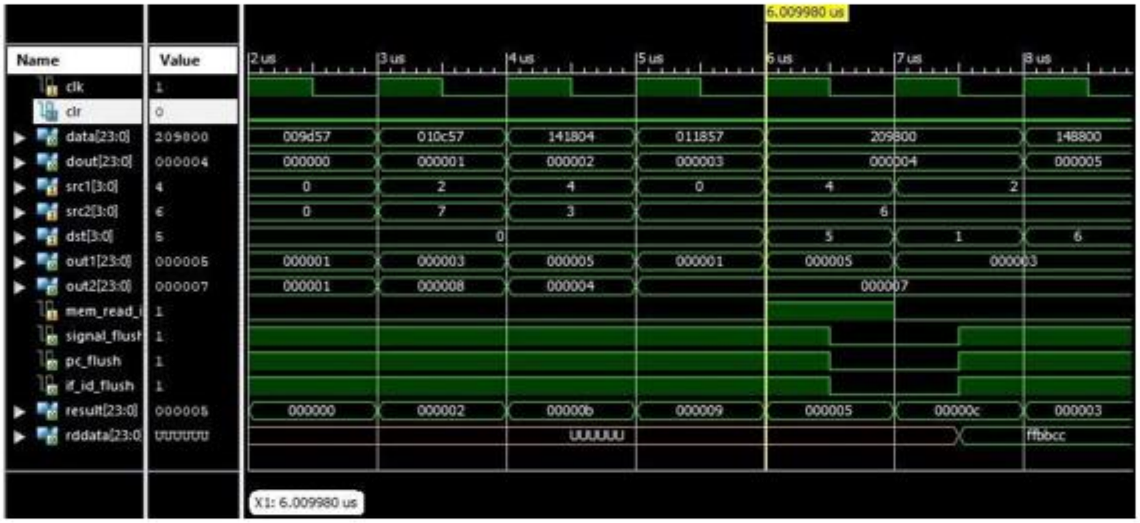
The hazard detection unit’s inputs are as follows. — IF/ID.RegisterRs and IF/ID.RegisterRt, the source registers for the current instruction. — ID/EX.MemRead and ID/EX.RegisterRt, to determine if the previous instruction is LW and, if so, which register it will write to. □ By inspecting these values, the detection unit generates three outputs. — Two new control signals PCWrite and IF/ID Write, which determine whether the pipeline stalls or continues. — A mux select for a new multiplexer, which forces control signals for the current EX and future MEM/WB stages to 0 in case of a stall.

1.7 Simulation Result

The Resulting output waves of 5-staged pipelined MIPS RISC processor is shown below:

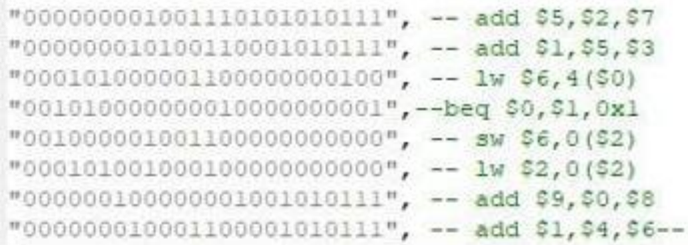


```
"000000001001110101010111", -- add $5,$2,$7
"000000010100110001010111", -- add $1,$5,$3
"000101000001100000000100", -- lw $6,4($0)
"001010000000010000000001",--beq $0,$1,0x1
"001000001001100000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
"000000100000001001010111", -- add $9,$0,$8
"000000001000110000101011l", -- add $1,$4,$6--
```

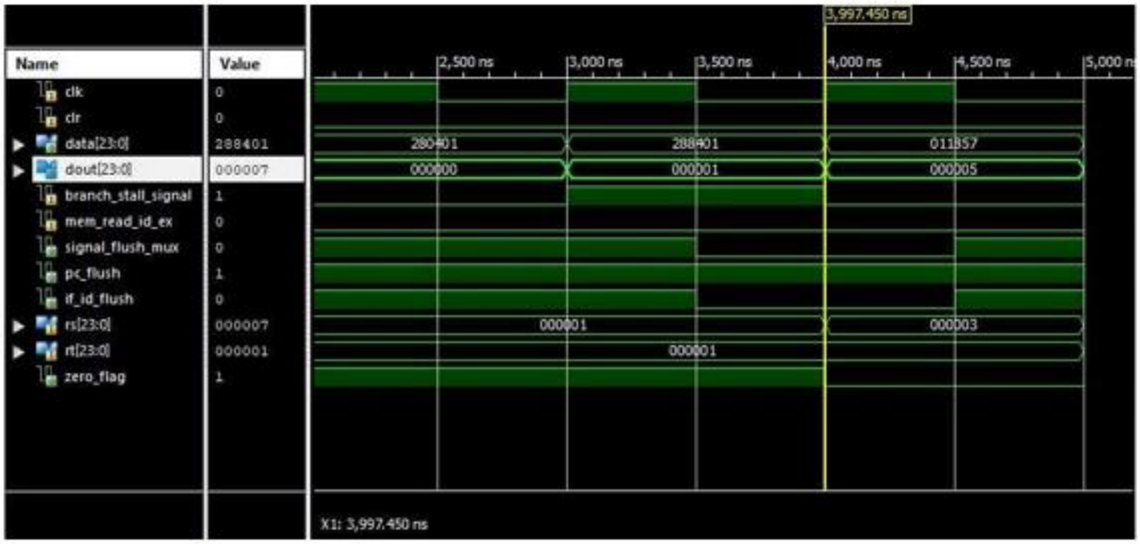


Name	Value
clk	1
clr	0
data[23:0]	209800
dout[23:0]	000004
src1[3:0]	4
src2[3:0]	6
dst[3:0]	5
out1[23:0]	000005
out2[23:0]	000007
mem_read_i	1
signal_flush	1
pc_flush	1
if_id_flush	1
result[23:0]	000005
rddata[23:0	UUUUUU

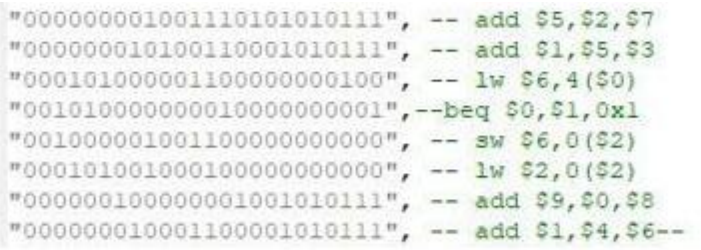
2-branch



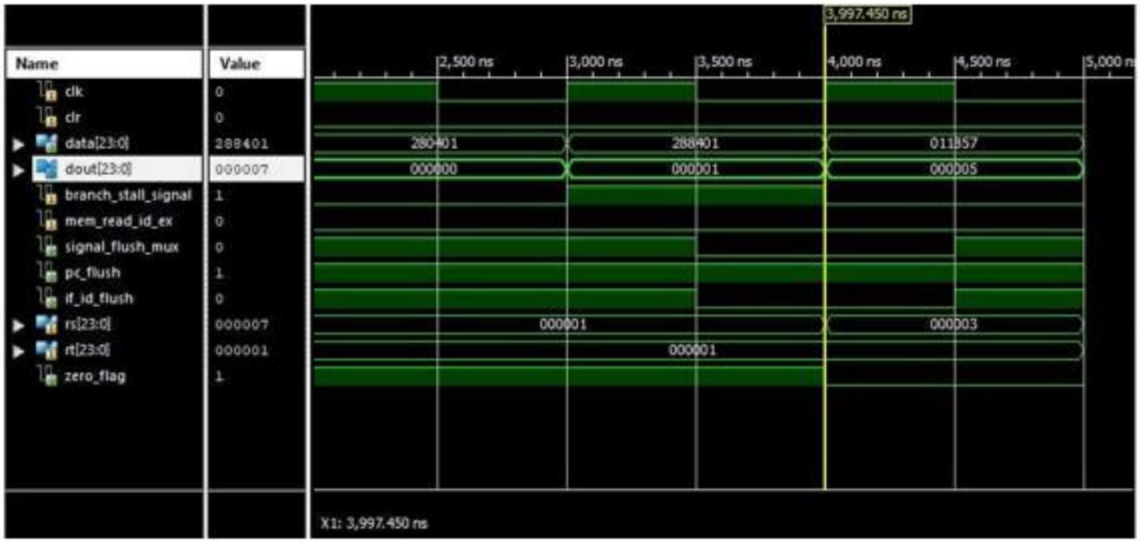
```
"000000001001110101010111", -- add $5,$2,$7
"000000010100110001010111", -- add $1,$5,$3
"000101000001100000000100", -- lw $6,4($0)
"001010000000010000000001",--beq $0,$1,0x1
"001000001001100000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
"000000100000001001010111", -- add $9,$0,$8
"000000001000110000101011l", -- add $1,$4,$6--
```

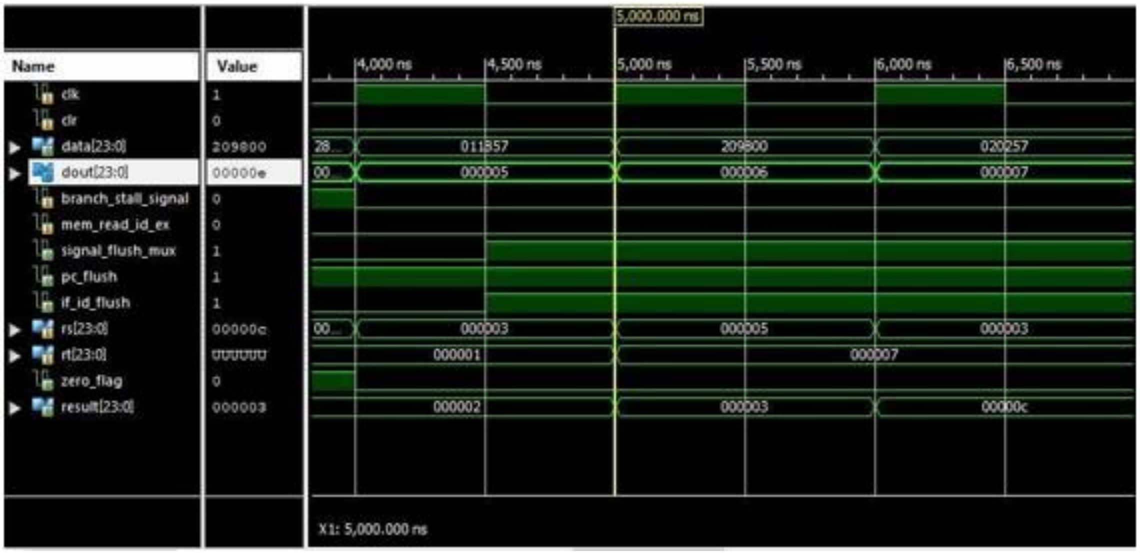


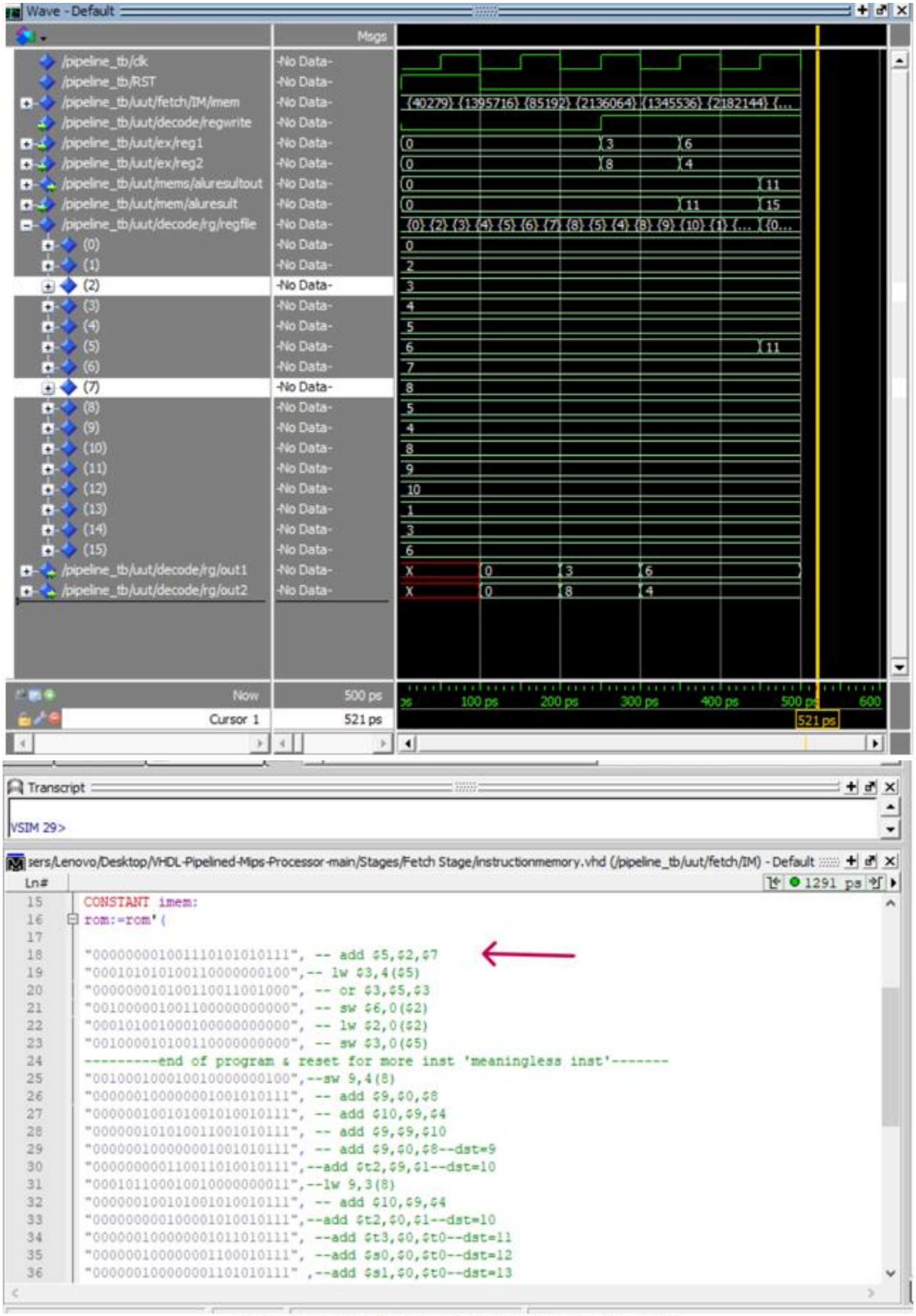
Name	Value
clk	0
clr	0
data[23:0]	288401
dout[23:0]	000007
branch_stall_signal	1
mem_read_id_ex	0
signal_flush_mux	0
pc_flush	1
if_id_flush	0
rs[23:0]	000007
rt[23:0]	000001
zero_flag	1



```
"000000001001110101010111", -- add $5,$2,$7
"000000101001100010101111", -- add $1,$5,$3
"000101000001100000000100", -- lw $6,4($0)
"001010000000010000000001",--beq $0,$1,0x1
"001000000100110000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
"000000100000000100101011 1", -- add $9,$0,$8
"000000010001100000101011 1", -- add $1,$4,$6--
```

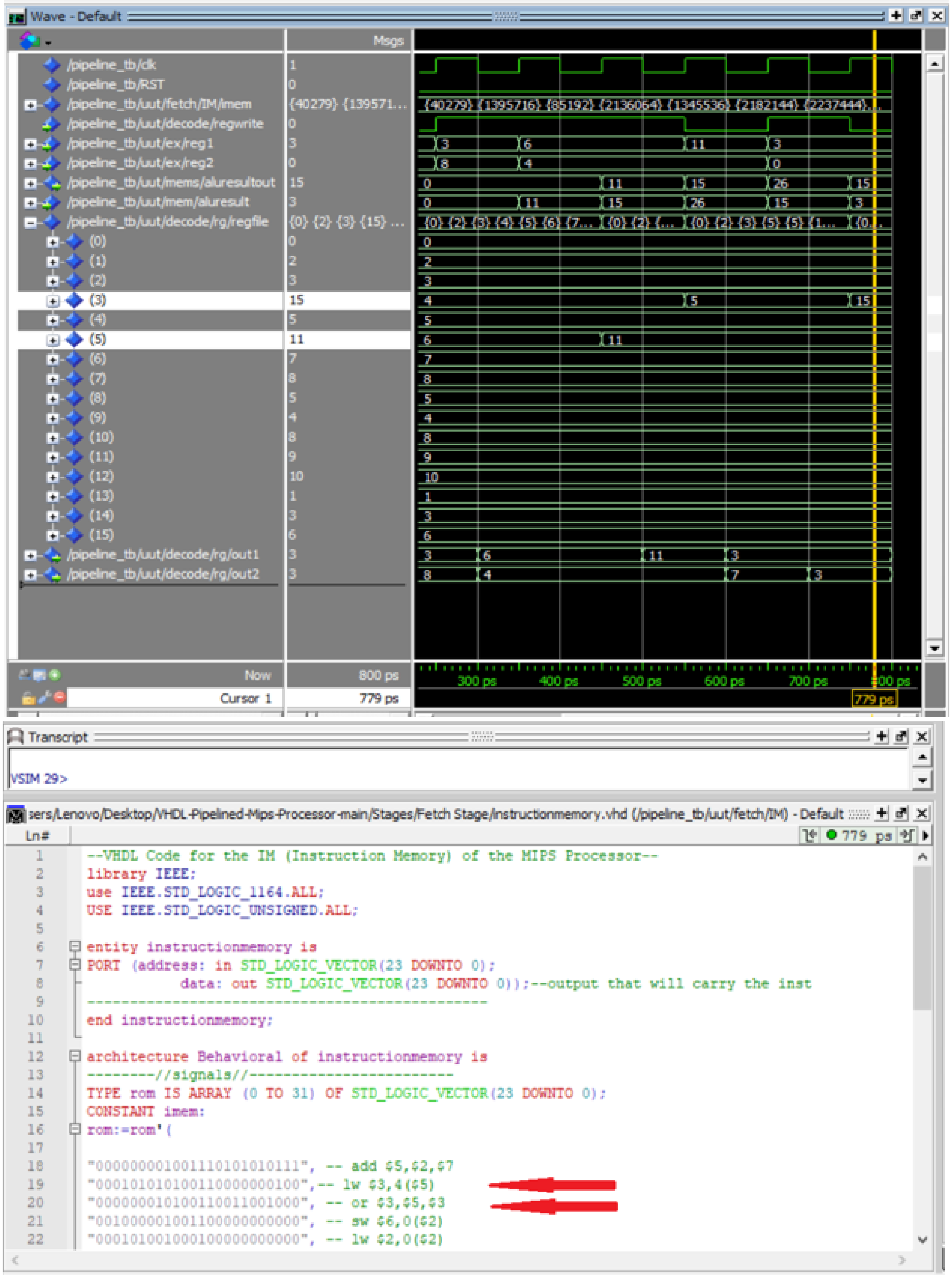






```
CONSTANT imem:
rom:=rom'(

"00000000100111010101010111", -- add $5,$2,$7
"00010101010011000000000100",-- lw $3,4($5)
"00000001010011001100100 0", -- or $3,$5,$3
"00100000100110000000000000", -- sw $6,0($2)
"00010100100010000000000000", -- lw $2,0($2)
"00100000101001100000000000", -- sw $3,0($5)
---------end of program & reset for more inst 'meaningless inst'-------
"00100010001001000000000100",--sw 9,4(8)
"00000010000000010010101111", -- add $9,$0,$8
"00000010010100101001010111", -- add $10,$9,$4
"00000010101001100101010111", -- add $9,$9,$10
"00000010000000010010101111", -- add $9,$0,$8--dst=9
"00000000001100110100101111",--add $t2,$9,$1--dst=10
"00010110001001000000000011",--lw 9,3(8)
"00000010010100101001010111", -- add $10,$9,$4
"00000000001000010100101111",--add $t2,$0,$1--dst=10
"00000010000000010110101111", --add $t3,$0,$t0--dst=11
"00000010000000011000101111", --add $s0,$0,$t0--dst=12
"00000010000000011010101111" ,--add $s1,$0,$t0--dst=13
```

Fig1.code and wave form of instruction 1 (add $5,$2,$7).



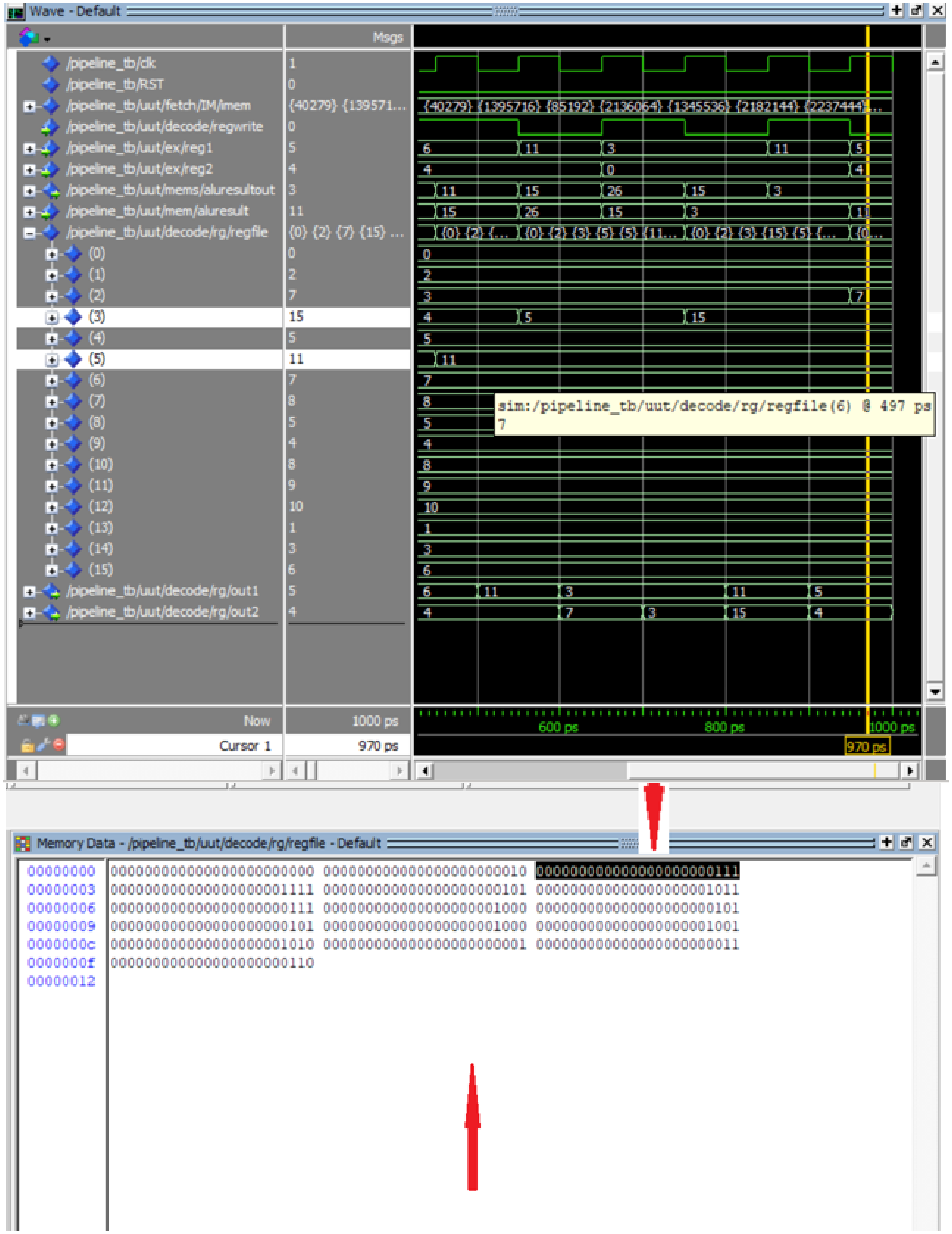
	Msgs
/pipeline_tb/clk	1
/pipeline_tb/RST	0
/pipeline_tb/uut/fetch/IM/imem	{40279} {139571...
/pipeline_tb/uut/decode/regwrite	0
/pipeline_tb/uut/ex/reg1	3
/pipeline_tb/uut/ex/reg2	0
/pipeline_tb/uut/mems/aluresultout	15
/pipeline_tb/uut/mem/aluresult	3
/pipeline_tb/uut/decode/rg/regfile	{0} {2} {3} {15} ...
(0)	0
(1)	2
(2)	3
(3)	15
(4)	5
(5)	11
(6)	7
(7)	8
(8)	5
(9)	4
(10)	8
(11)	9
(12)	10
(13)	1
(14)	3
(15)	6
/pipeline_tb/uut/decode/rg/out1	3
/pipeline_tb/uut/decode/rg/out2	3

```
VSIM 29>
```

sers/Lenovo/Desktop/VHDL-Pipelined-Mips-Processor-main/Stages/Fetch Stage/instructionmemory.vhd (/pipeline_tb/uut/fetch/IM) - Default

```
--VHDL Code for the IM (Instruction Memory) of the MIPS Processor--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity instructionmemory is
PORT (address: in STD_LOGIC_VECTOR(23 DOWNTO 0);
           data: out STD_LOGIC_VECTOR(23 DOWNTO 0));--output that will carry the inst
-----------------------------------------------
end instructionmemory;

architecture Behavioral of instructionmemory is
---------//signals//------------------------
TYPE rom IS ARRAY (0 TO 31) OF STD_LOGIC_VECTOR(23 DOWNTO 0);
CONSTANT imem:
rom:=rom'(

"000000001001110101010111", -- add $5,$2,$7
"000101010100110000000100",-- lw $3,4($5)
"000000010100110011001000", -- or $3,$5,$3
"001000000100110000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
```

Fig2. assembly code and .wave form of instruction 1,2 (lw $3,4($5)) ------- (or $3,$5,$3).



Wave - Default

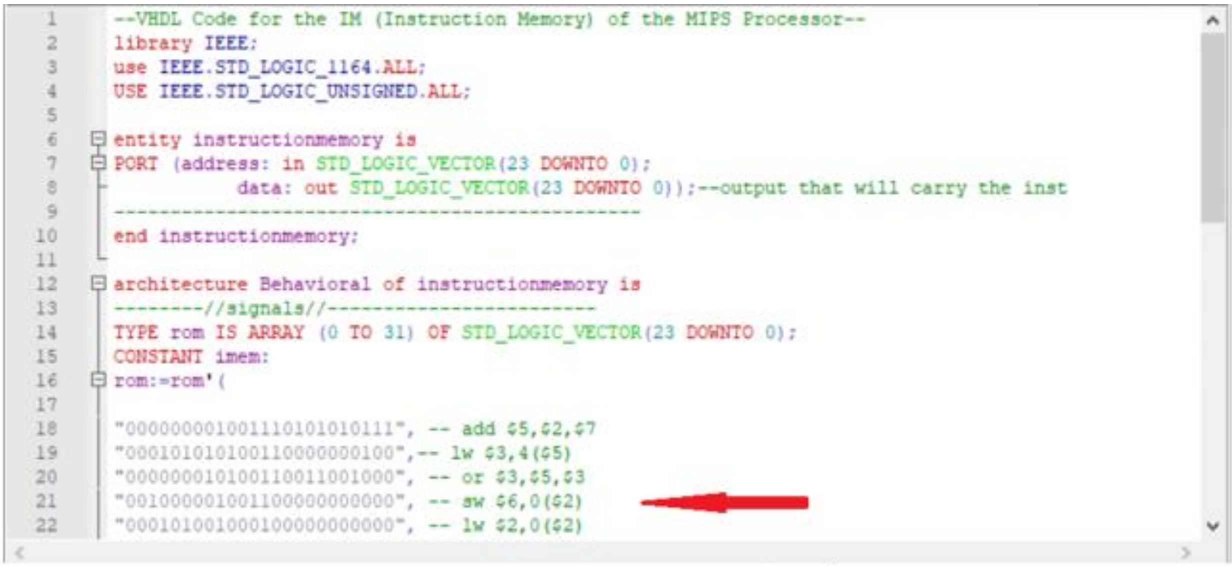
```
                                         Msgs
/pipeline_tb/clk                         1
/pipeline_tb/RST                         0
/pipeline_tb/uut/fetch/IM/imem           {40279} {139571...
/pipeline_tb/uut/decode/regwrite         0
/pipeline_tb/uut/ex/reg1                 5
/pipeline_tb/uut/ex/reg2                 4
/pipeline_tb/uut/mems/aluresultout       3
/pipeline_tb/uut/mem/aluresult           11
/pipeline_tb/uut/decode/rg/regfile       {0} {2} {7} {15} ...
  (0)                                    0
  (1)                                    2
  (2)                                    7
  (3)                                    15
  (4)                                    5
  (5)                                    11
  (6)                                    7
  (7)                                    8
  (8)                                    5
  (9)                                    4
  (10)                                   8
  (11)                                   9
  (12)                                   10
  (13)                                   1
  (14)                                   3
  (15)                                   6
/pipeline_tb/uut/decode/rg/out1          5
/pipeline_tb/uut/decode/rg/out2          4

Now                                      1000 ps
Cursor 1                                 970 ps
```

sim:/pipeline_tb/uut/decode/rg/regfile(6) @ 497 ps
7

Memory Data - /pipeline_tb/uut/decode/rg/regfile - Default

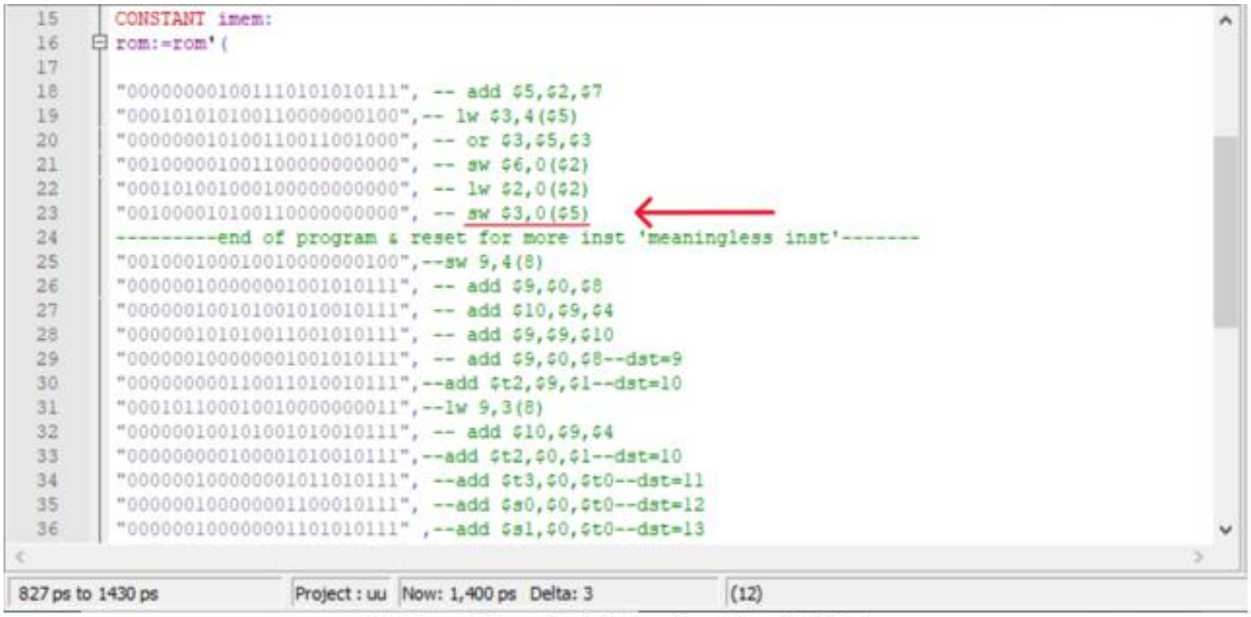
```
00000000  000000000000000000000000  000000000000000000000010  000000000000000000000111
00000003  000000000000000000001111  000000000000000000000101  000000000000000000001011
00000006  000000000000000000000111  000000000000000000001000  000000000000000000000101
00000009  000000000000000000000101  000000000000000000001000  000000000000000000001001
0000000c  000000000000000000001010  000000000000000000000001  000000000000000000000011
0000000f  000000000000000000000110
00000012
```

Fig4.data memory content of instruction 4 (sw $6,0($2)).



```
--VHDL Code for the IM (Instruction Memory) of the MIPS Processor--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity instructionmemory is
PORT (address: in STD_LOGIC_VECTOR(23 DOWNTO 0);
          data: out STD_LOGIC_VECTOR(23 DOWNTO 0));--output that will carry the inst
-----------------------------------------------
end instructionmemory;

architecture Behavioral of instructionmemory is
--------//signals//-----------------------
TYPE rom IS ARRAY (0 TO 31) OF STD_LOGIC_VECTOR(23 DOWNTO 0);
CONSTANT imem:
rom:=rom'(

"000000001001110101010111", -- add $5,$2,$7
"000101010100110000000100",-- lw $3,4($5)
"000000010100110011001000", -- or $3,$5,$3
"001000001001100000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
```

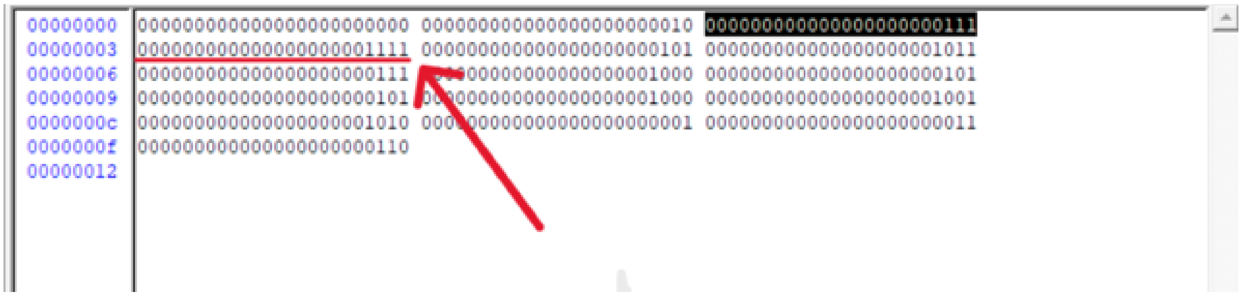
Fig5 .assebly code of intrucrion 4 (sw $6,0($2))



```
CONSTANT imem:
rom:=rom'(

"000000001001110101010111", -- add $5,$2,$7
"000101010100110000000100",-- lw $3,4($5)
"000000010100110011001000", -- or $3,$5,$3
"001000001001100000000000", -- sw $6,0($2)
"000101001000100000000000", -- lw $2,0($2)
"001000010100110000000000", -- sw $3,0($5)
---------end of program & reset for more inst 'meaningless inst'-------
"001000100010010000000100",--sw 9,4(8)
"000000100000001001010111", -- add $9,$0,$8
"000000100101001010010111", -- add $10,$9,$4
"000000101010011001010111", -- add $9,$9,$10
"000000100000001001010111", -- add $9,$0,$8--dst=9
"000000000110011010010111",--add $t2,$9,$1--dst=10
"000101100010010000000011",--lw 9,3(8)
"000000100101001010010111", -- add $10,$9,$4
"000000000100001010010111",--add $t2,$0,$1--dst=10
"000000100000001011010111", --add $t3,$0,$t0--dst=11
"000000100000001100010111", --add $s0,$0,$t0--dst=12
"000000100000001101010111" ,--add $s1,$0,$t0--dst=13
```

827 ps to 1430 ps | Project : uu | Now: 1,400 ps Delta: 3 | (12)

Fig5 .assebly code of intrucrion 5 (sw $3,0($5))



```
00000000  000000000000000000000000 000000000000000000000010 000000000000000000000111
00000003  000000000000000000001111 000000000000000000000101 000000000000000000001011
00000006  000000000000000000000111 000000000000000000001000 000000000000000000000101
00000009  000000000000000000000101 000000000000000000001000 000000000000000000001001
0000000c  000000000000000000001010 000000000000000000000001 000000000000000000000011
0000000f  000000000000000000000110
00000012
```

Fig6 .data memory content of instruction 5 (sw $3,0($5)).

2- Conclusion

MIPS processor is widely used RISC processor in industry and research area. In this paper, we have successfully designed and synthesized a basic model of pipelined MIPS processor. The design has been modeled in VHDL and functional verification policies adopted for it. The simulation results show that maximum frequency of pipeline processor is increased from 100MHz to 200MHz.