

Faculty of Engineering

Computer and Systems Engineering Department

**CSE 311: Computer Organization (2)**

Instructor: Dr Cherif Salama

Mips Processor Single Cycle Implementation

**Submitted by**

Magdy Mohamed Abdel Moneim Hafez ID: 33800

Mohamed Ismail Mohamed Hafez ID: 33808

Sherif Ahmed Hassan Kotb ID: 33762

Shady Hany El Sayed ID: 33760

Marco Emile Ramzy ID: 33794

Fall (2015-2016)

1. **Description of the implementation :**

**Mips consists of :**

1. **PC:**

The program counter is a 32 bit D-flip flop register. It selects the address of the upcoming instruction. The PC is incremented after fetching the instruction and hold the memory address of the next instruction that will be executed.

**b) Instruction memory:**

It is a memory composed of a 32 bit width and in our implementation it has 40 entries depth. It holds the instructions that should be executed.

**c) Control Unit:**

It generates all control signals to all components of the processor .These control signals jump , branch , memread, memtoreg, aluop, memwrite , alu src ,regwrite and regdst.Its input is the opcode .

**d) Register File:**

It is an array of processor registers in a CPU.It has 32 registers each is of 32 bits. It holds the values of the registers on which operations are executed.

**e) ALU:**

It is the Arithmetic Logic Unit, a digital electronic circuit which performs bitwise arithmetic and logical operations.

It has two inputs .The first is Read Data 1 from the register file and the second is the output of the MUX that contains the Read Data 2 from register file and output from the sign extension unit.

**f) ALU control:**

Its inputs is the least six significant bits(function field ) and the other is the ALU op.It generates a control signal as an input to the ALU to select the operation to be executed.

**g) Data Memory:**

It has 256 registers each of width 32 bits.It holds the necessary data for instruction execution.

**h) Multiplexers:**

They are used to select one of its inputs to be forwarded to the next component depending on selector. Number of inputs is 2selectors .

**I) Sign extension:**

Its input is a 16 bit input and output is of 32 bits.It repeats the most significant bit of the input 16 times to be the most significant of the output.

**K) Full adder:**

It is a 32 bit full adder.

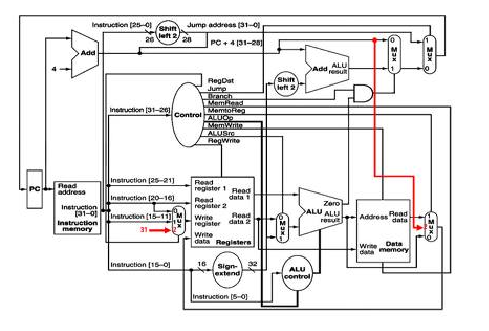
**L) Shift by two:**

Shifts the input by two.

**M) Adder(pc+4):**

Increments the program counter by four.

**2)Data path :**

****

We added two changes:

1. The instruction [10:6] is a shift amount input for the ALU.
2. Jump register control signal is an output from ALU control unit to a third MUX whose output is an input to the program counter.
3. Adding a 2\*1 multiplexer having a 1bit two inputs and a 1bit ouput,it's inputs are : regwrite signal(1bit) from control unit and a zero(1bit) and it's output is an input to register file(which determines whether to write in the register file or not).It's selector is jumpreg. If the jumpreg value = 0(i.e it's not a jr instruction),then multiplexer output will be regwrite signal from control unit.If the

jumpreg value = 1(i.e it's a jr instruction),then multiplexer output will be zero because it's an R-type instruction and all R-type instructions write in register file but for jr instruction(which is an R-type) we don't want to write in the register file.

1. Adder that increment the program counter will increment it by 1 not 4, since the instruction memory width in our implementation is 32 bits(1 word) not 8 bits(1 byte).

**3)How the work was split:**

1-Magdy Mohamed Abdel Moneim Hafez

* Instruction memory.
* Multiplexors.
* Program counter.

2-Mohamed Ismail Mohamed Hafez:

* ALU.
* ALU Control Unit.

3-Sherif Ahmed Hassan Kotb:

* Data memory.
* Shift by two unit.
* Full Adder.

4-Shady Hany El Sayed Mohamed:

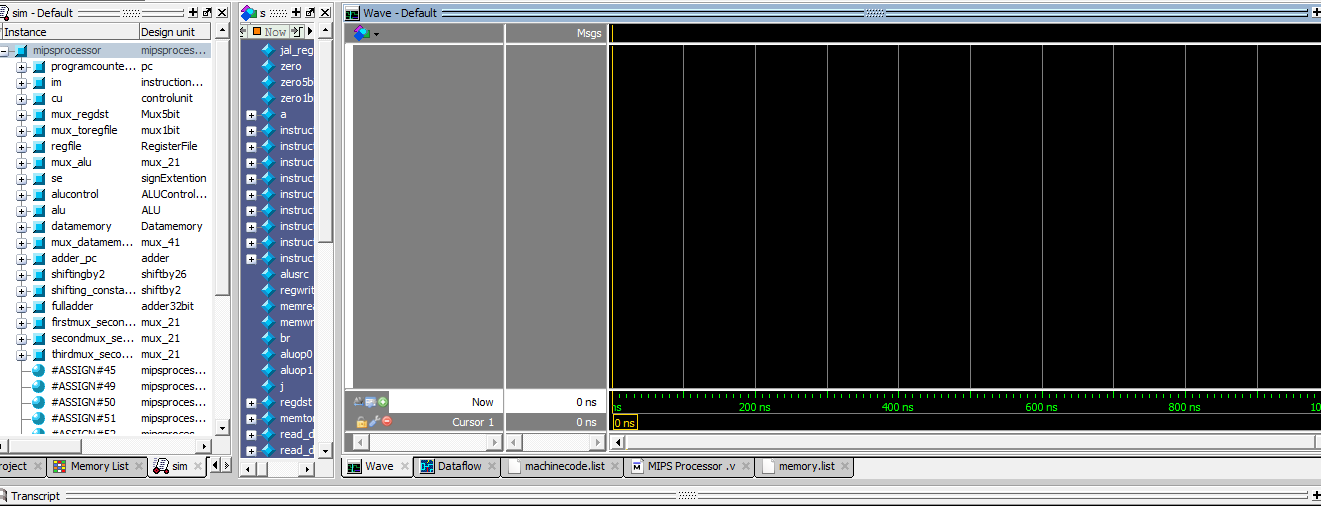
* Control Unit.
* (PC+4) Adder.

5-Marco Emile Ramzy Andraws:

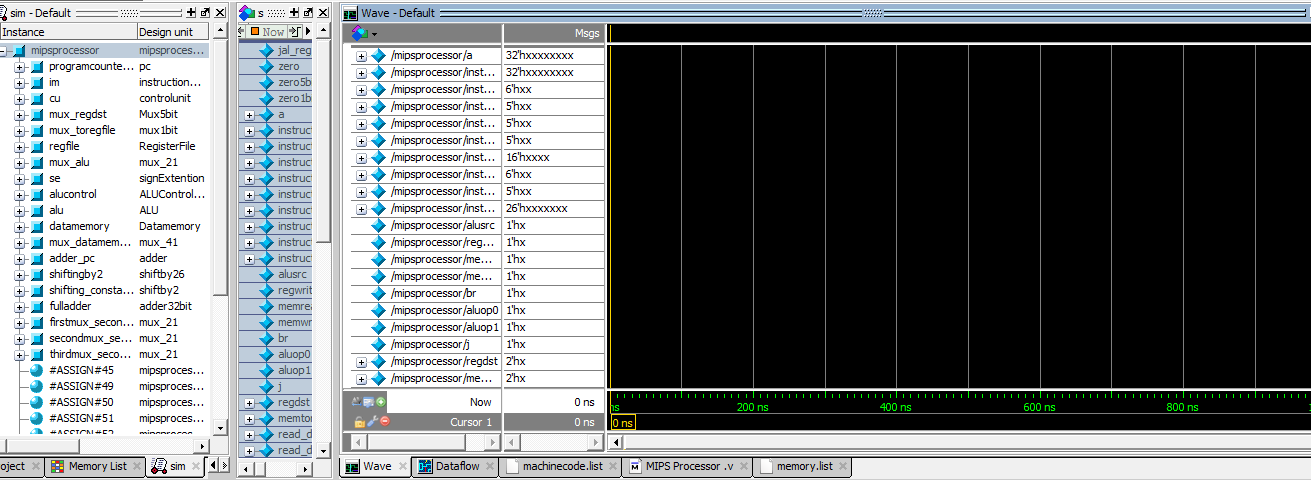
* Register File.
* Sign extension.
* The whole team contributed in the integration, testing and simulation of the MIPS.
* Troubleshooting By Magdy Mohamed, Mohamed Ismail and Sherif Ahmed.

**4)How to simulate:**

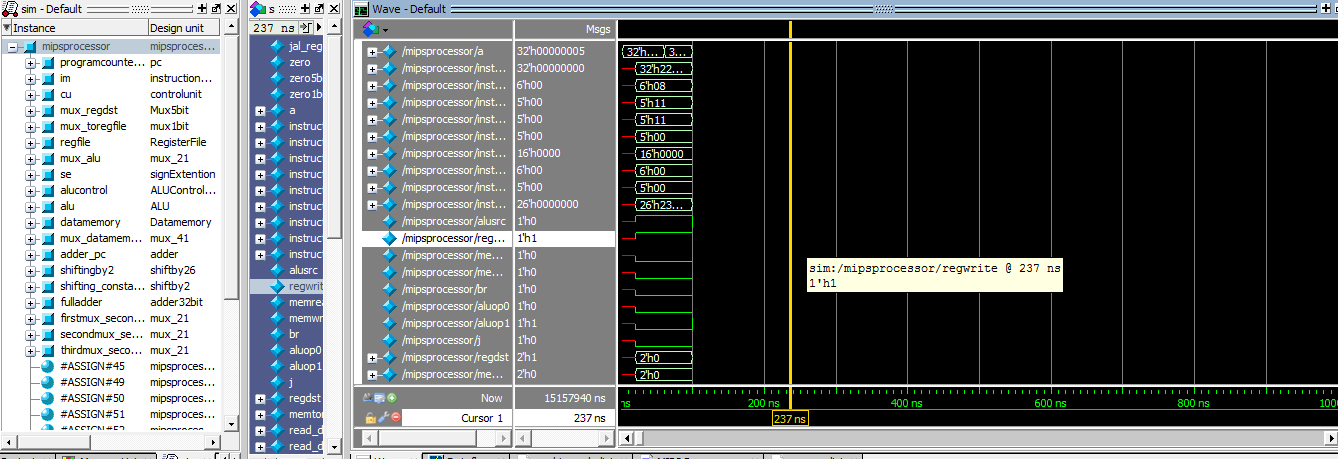
* From simulation menu choose start simulation
* Choose file you want to simulate , the following appears:



* Add all signals to wave



* The form simulation menu choose run🡪run all or run 100:



**5)Test programs :**

* Program one: This program adds the numbers between A and B, inclusive, where A originally resides at address 4 in data memory and B resides in address 8 in data memory. The result is placed at data memory location 0. Since there is no halt instruction on our MIPS processor, the program, after computing the result, will just loop (until you turn off the clock).
* The machine code :

00000022   
8c010000   
8c020004   
8c030008   
00842022   
00822020   
0043282a   
10a00002   
00411020   
1000fffb   
ac040000   
1000ffff

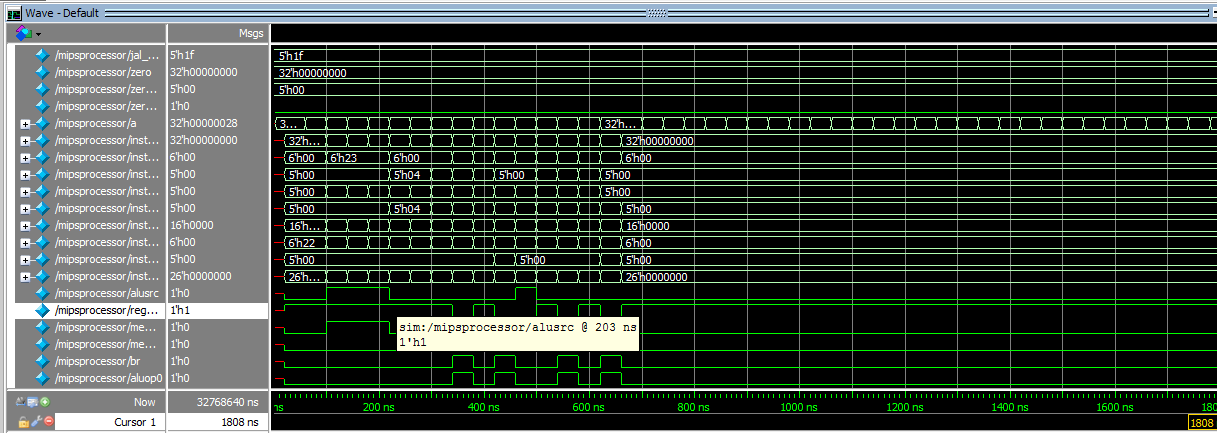
* The assembly:

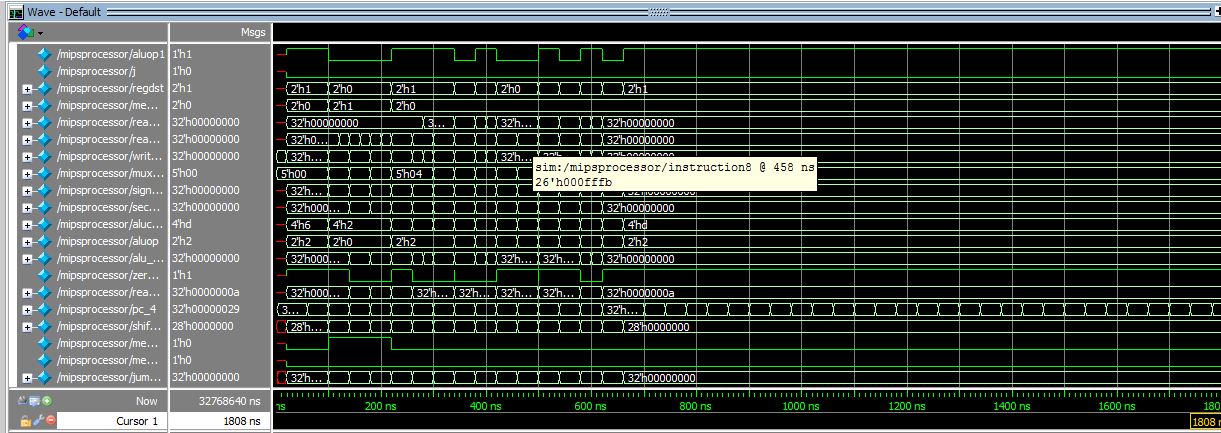
sub r0,r0,r0     ; set reg[0] to 0, use as base    
lw  r1,0(r0)     ; reg[1] <- mem[0] (= 1)    
lw  r2,4(r0)     ; reg[2] <- mem[4] (= A)    
lw  r3,8(r0)     ; reg[3] <- mem[8] (= B)   
sub r4,r4,r4     ; reg[4] <- 0, running total    
add r4,r2,r4     ; reg[4]+ = A    
slt r5,r2,r3     ; reg[5] <- A < B    
beq r5,r0,2      ; if reg[5] = FALSE, go forward 2 instructions    
add r2,r1,r2     ; A++    
beq r0,r0,-5     ; go back 5 instructions    
sw  r4,0(r0)     ; mem[0] <- reg[4]    
beq r0,r0,-1     ; program is over, keep looping back to here

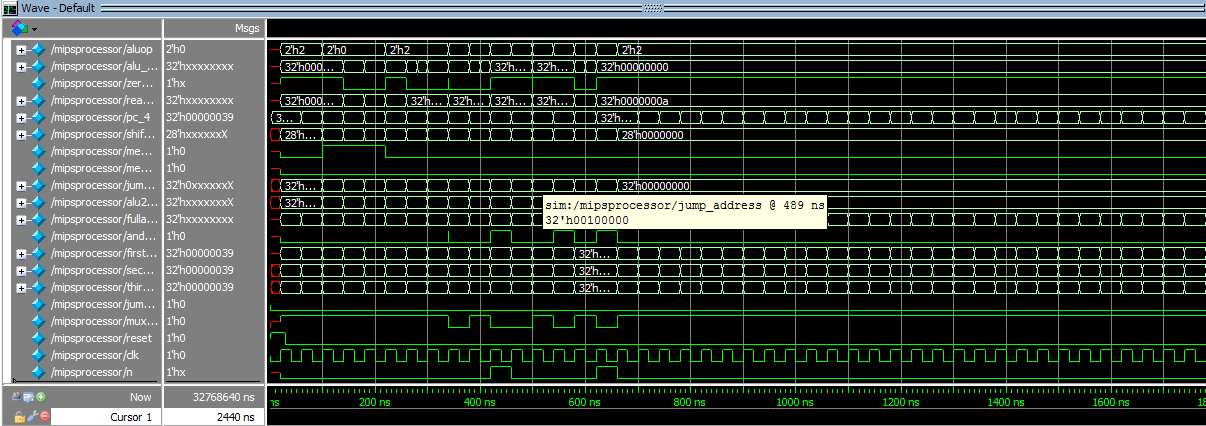
* Memory initialization :

00000001   
00000001   
0000000a

* Output screenshots:







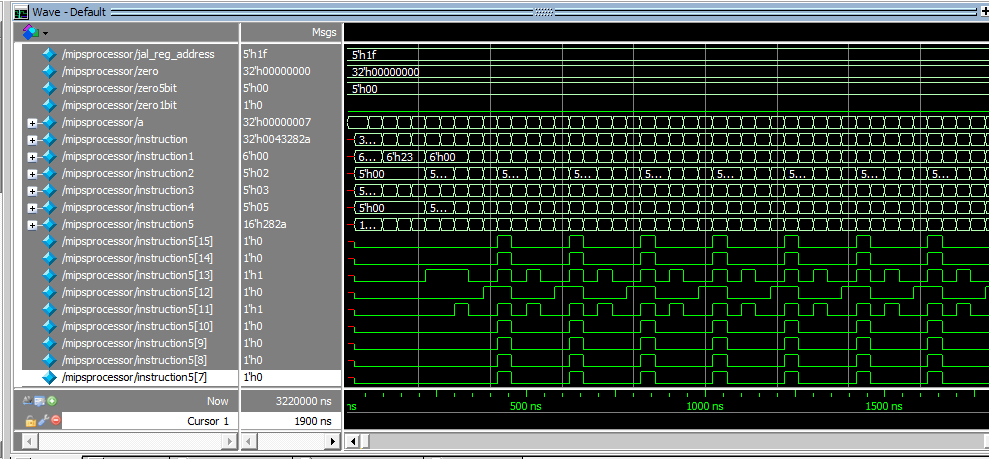
* Program Two: This program tests the processor's AND and OR operations, as well as testing addressing using offsets from a base register. The program starts with two variables, A and B, and does the following:   
     
  • computes A AND B and places the result in memory location 4   
  • computes A OR B and places the result in memory location 8
* The machine code :

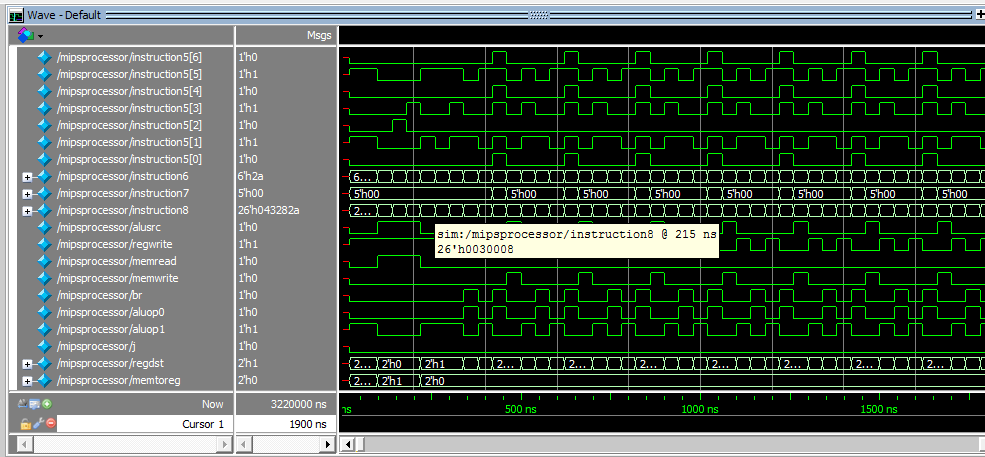
00000022   
8c010000   
8c220000   
8c230004   
00432024   
00432825   
ac040004   
ac050008   
1000ffff

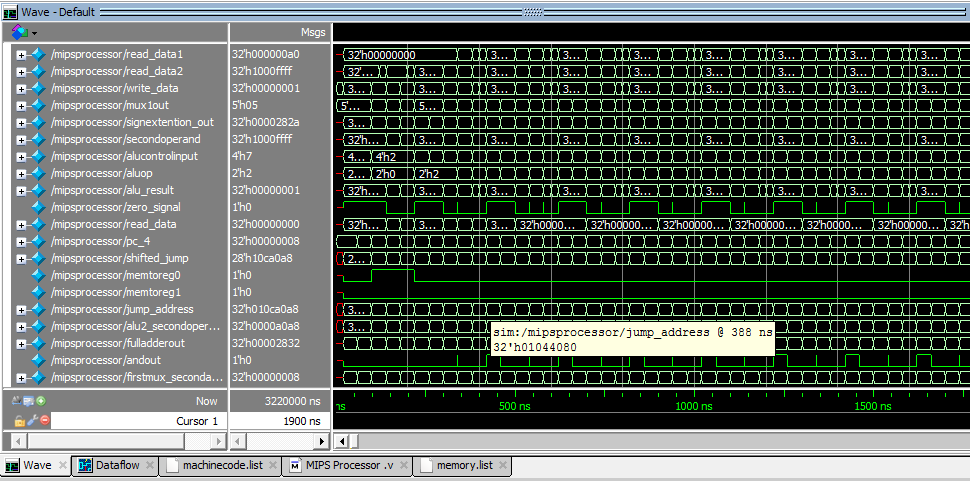
* The assembly:

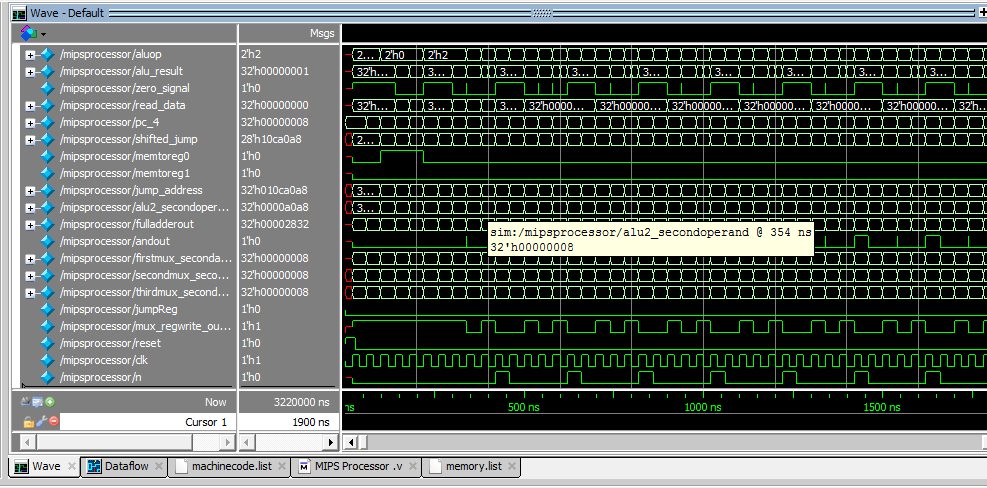
sub r0,r0,r0      ; set reg[0] to 0   
lw  r1,0(r0)      ; reg[1] <- mem[0] (= 20)    
lw  r2,0(r1)      ; reg[2] <- mem[20]    
lw  r3,4(r1)      ; reg[3] <- mem[24]   
and r4,r2,r3      ; reg[4] <- reg[2] AND reg[3]    
or  r5,r2,r3      ; reg[5] <- reg[2] OR reg[3]    
sw  r4,4(r0)      ; mem[4] <- reg[4]    
sw  r5,8(r0)      ; mem[8] <- reg[5]    
beq r0,r0,-1      ; program is over, loop back here

* Memory initialization :
* 00000014   
  00000000   
  00000000   
  00000000   
  00000000   
  430a1f9b   
  728cd2e3
* Output screenshots:

****

****

****

****

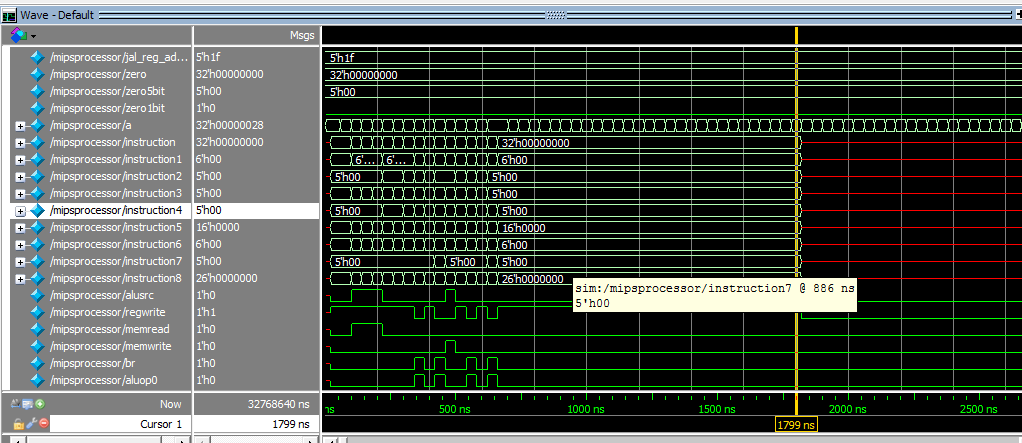
**6) Number of cycles :**

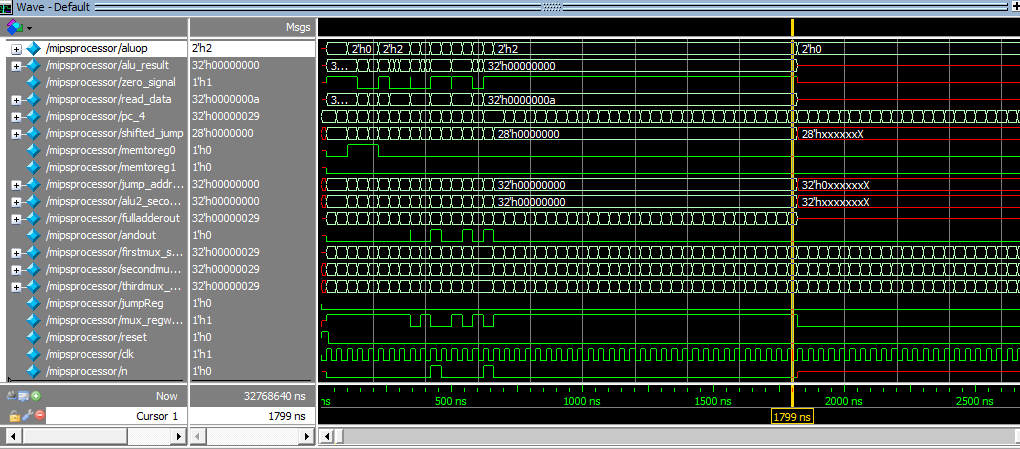
* Program one: Infinite due to loop.
* Program two:

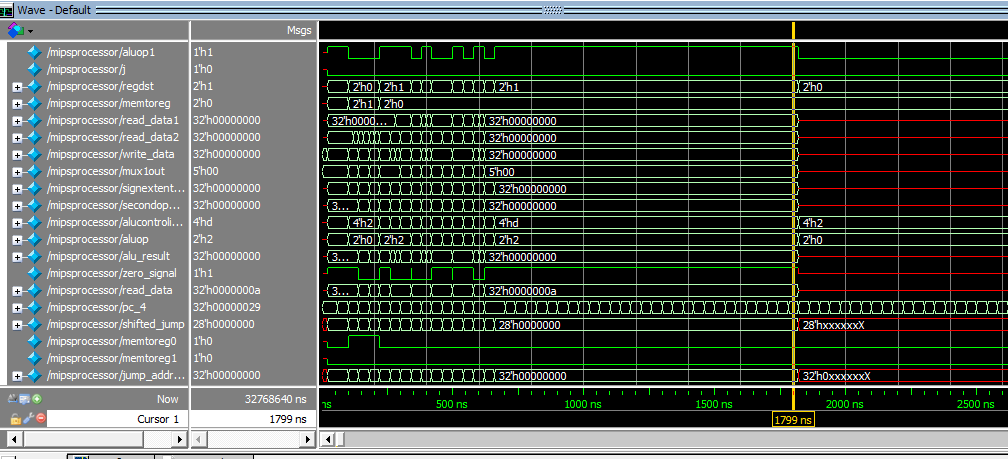
Eight cycles for execution and will not end because of the infinite loop.

**7) Final state:**

* Program one:







* Program two: Same as point 5.