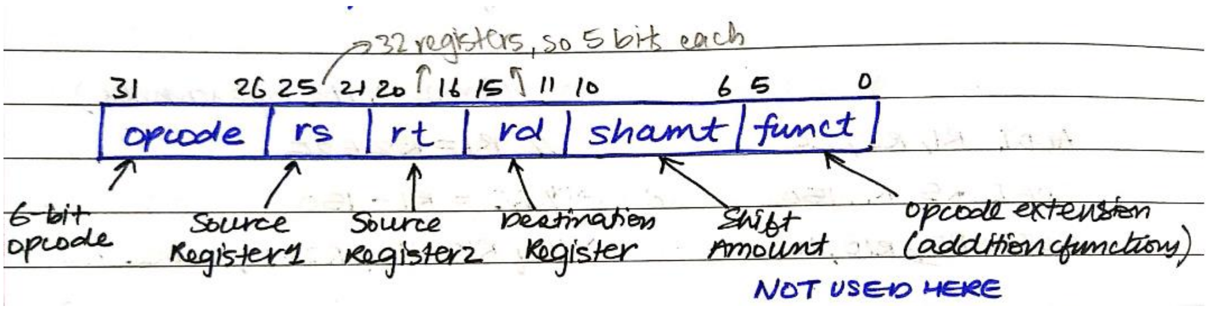
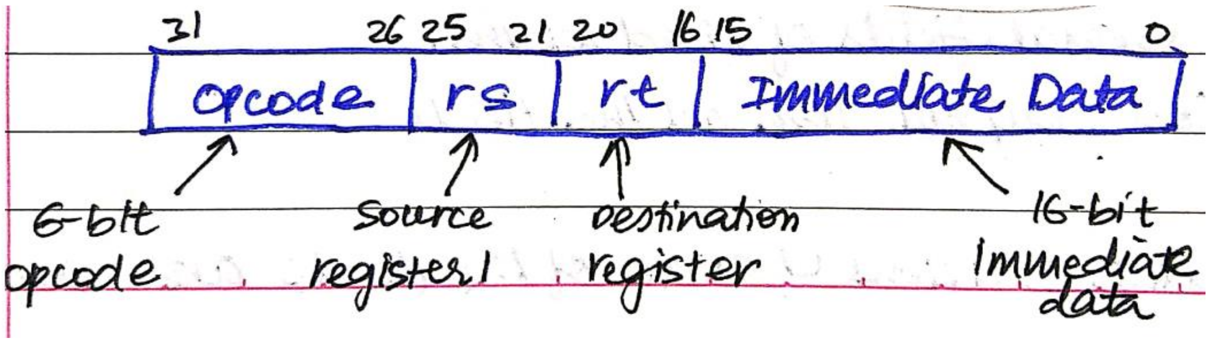
Reduced MIPS -32 pipelined architecture

(Verilog code in separate file)

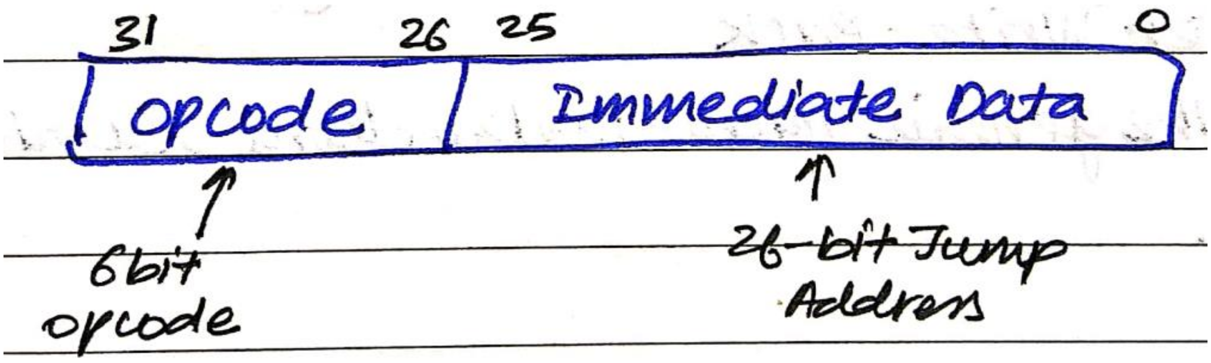
R-Type Instruction:



I – Type Instruction



J- Type Instruction



MIPS 32 Instruction cycle

We divide the instruction execution cycle into 5 steps :

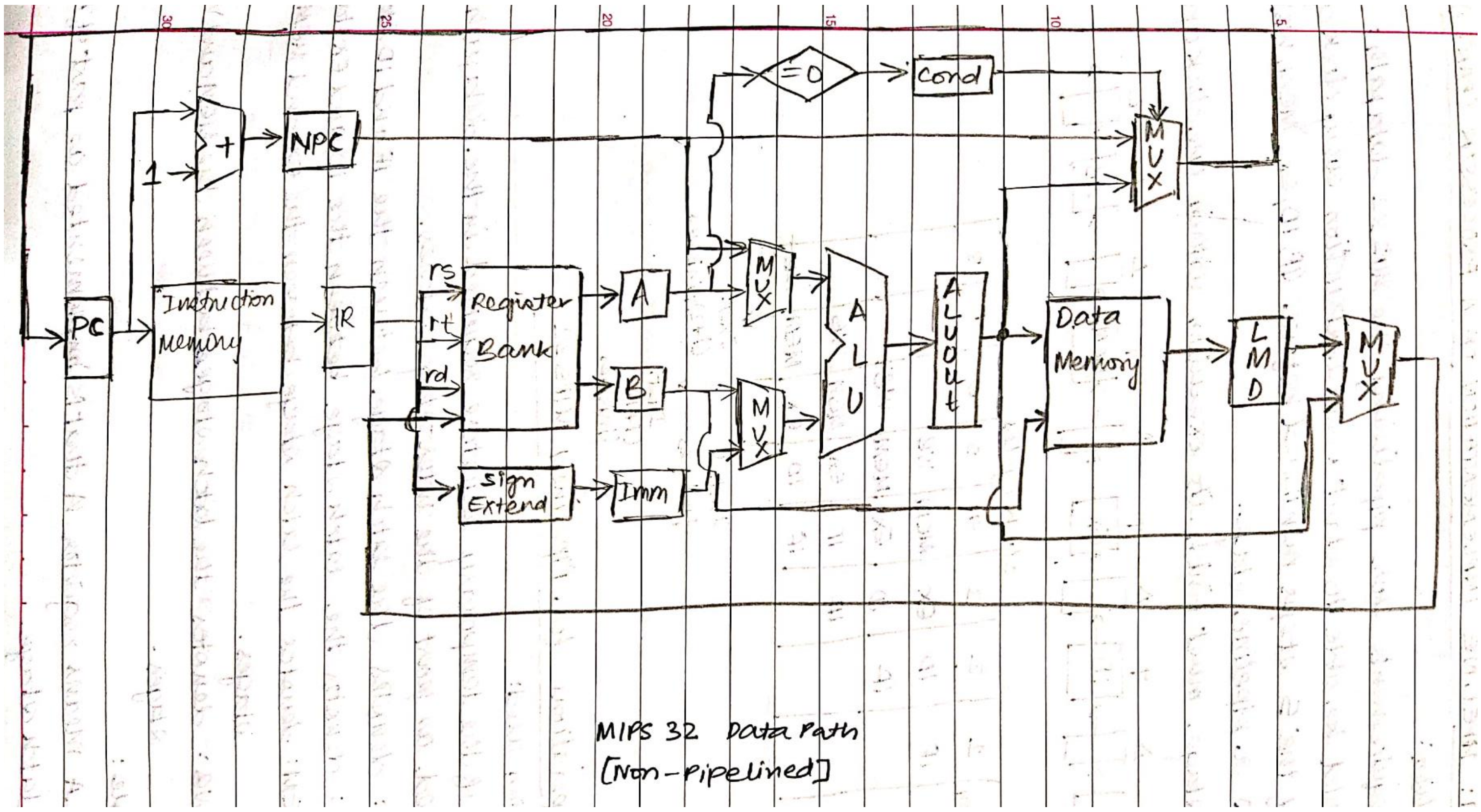
(a) IF : Instruction fetch

(b) ID : Instruction Decode / Register Fetch

(c) EX : Execution / Effective Address Calculation

(d) MEM : Memory Access / Branch Completion

(e) WB : Register Write-Back

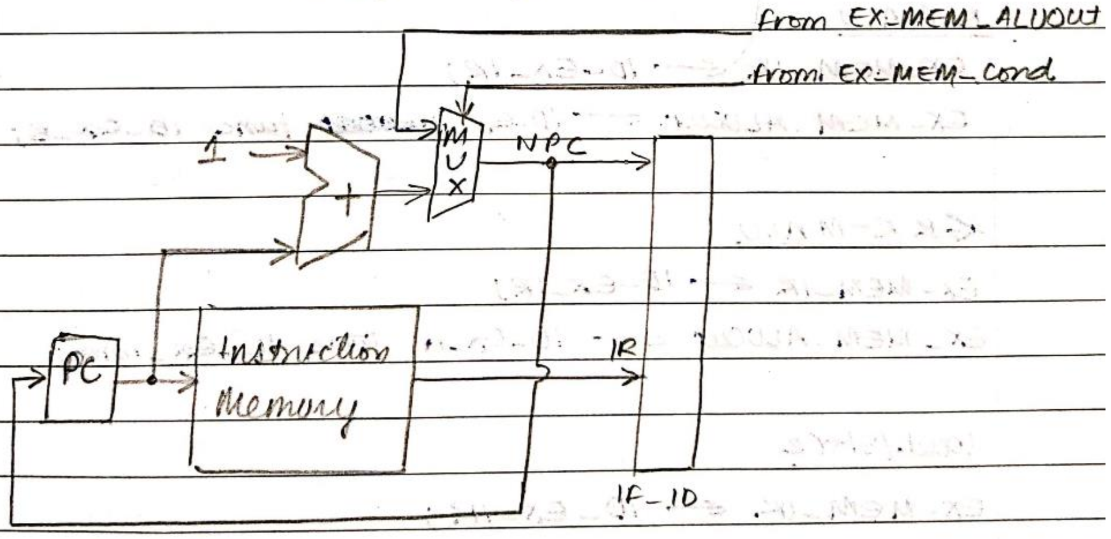


MIPS 32 Data Path
[Non-pipelined]

(a) Micro-operations for Pipeline Stage IF

IF_ID_IR ← Mem [PC]

IF_ID_NPC, PC ← (if ((Ex_MEM_IR [opcode] == branch) &
                                Ex_MEM_cond)
                                { EX_MEM_ALUOut }
                                else {PC+1});



(b) Micro-operations for Pipeline stage ID.

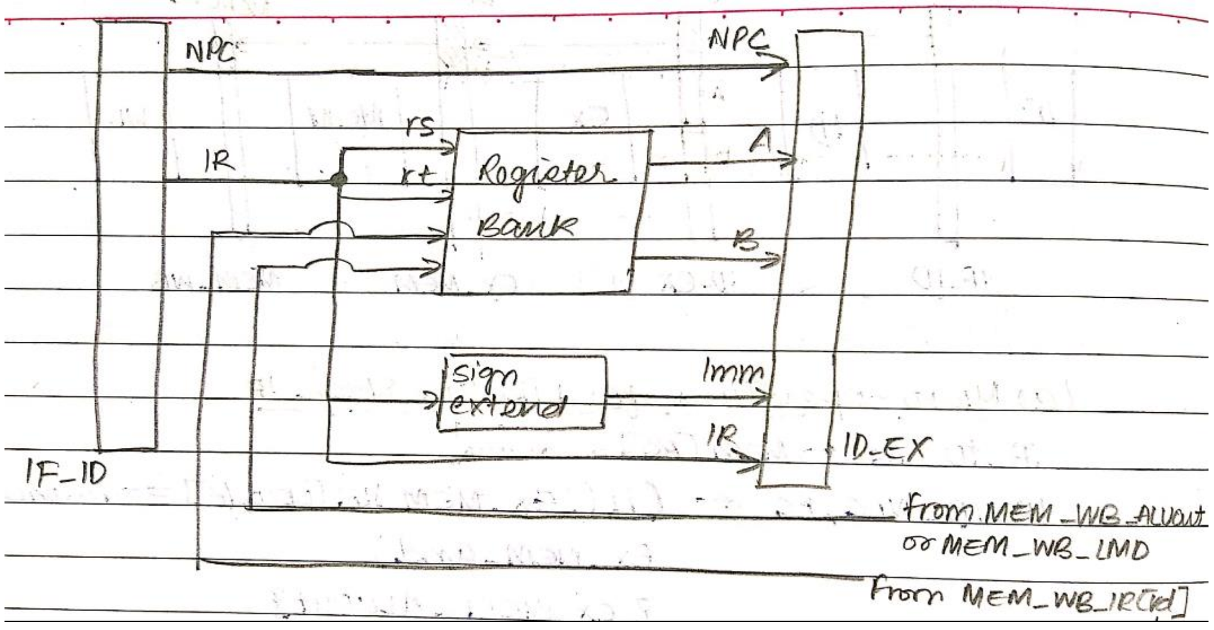
ID_EX_A ← Reg [IF_ID_IR [rs]];

ID_EX_B ← Reg [IF_ID_IR [rt]];

ID_EX_NPC ← IF_ID_NPC;

ID_EX_IR ← IF_ID_IR;

ID_EX_Imm ← Sign_extend (IF_ID_IR$_{15..0}$);



(b) Micro-operations for Pipeline stage ID.

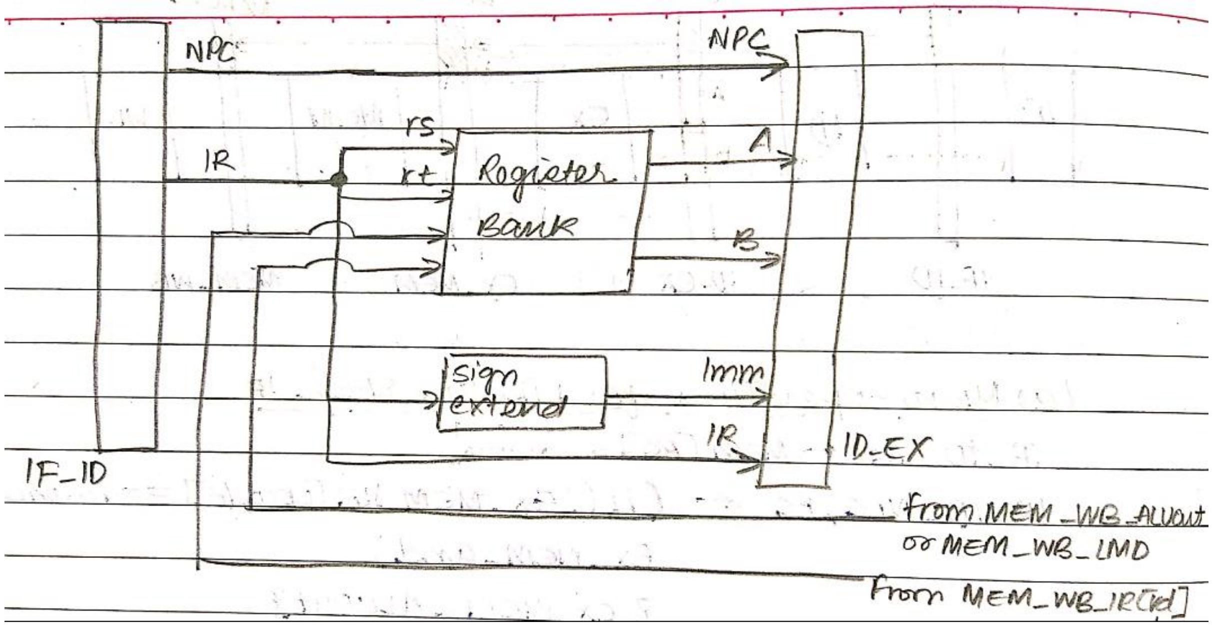
ID_EX_A ← Reg [IF_ID_IR [rs]] ;

ID_EX_B ← Reg [IF_ID_IR [rt]] ;

ID_EX_NPC ← IF_ID_NPC ;

ID_EX_IR ← IF_ID_IR;

ID_EX_Imm ← Sign_extend (IF_ID_IR$_{15..0}$);



(c) Micro-Operations for Pipeline Stage Ex.

R-R ALU

EX_MEM_IR ← ID-EX_IR;

EX-MEM_ALUout ← ID-EX_A[redacted] func ID-EX_B;

~~RR~~ R-M ALU

EX-MEM_IR ← ID-EX_IR;

EX_MEM_ALUOut ← ID-EX_A func ID_EX_imm;

Load/Store

EX-MEM_IR ← ID_EX_IR;

EX_MEM_ALUout ← ID-EX_A + ID-EX_Imm;

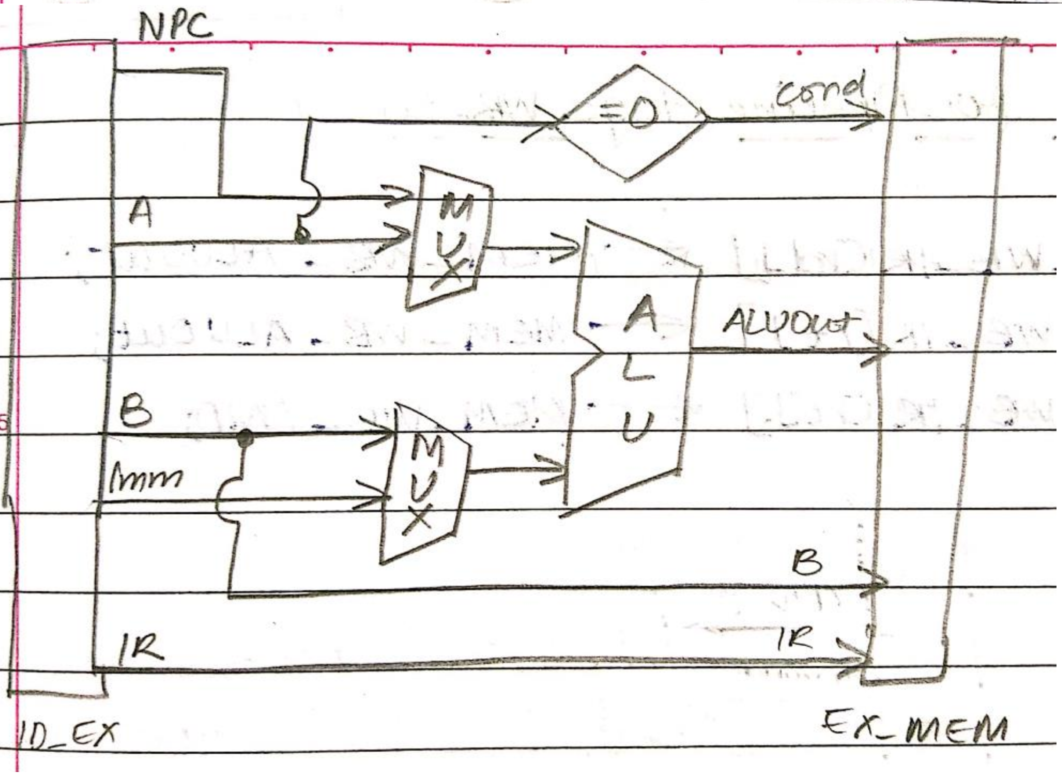
EX_MEM_B ← ID-EX_B;

Branch

EX_MEM_ALUout ← ID_EX_NPC + ID_EX_Imm;

EX_MEM_cond ← (ID_EX_A == 0);

EX_MEM_IR ← ID_EX_IR;



(d) Micro-operations for Pipeline Stage MEM

ALU:

MEM_WB_IR ← EX_MEM_IR;

MEM_WB_ALUOut ← EX_MEM_ALUOut;

Load:

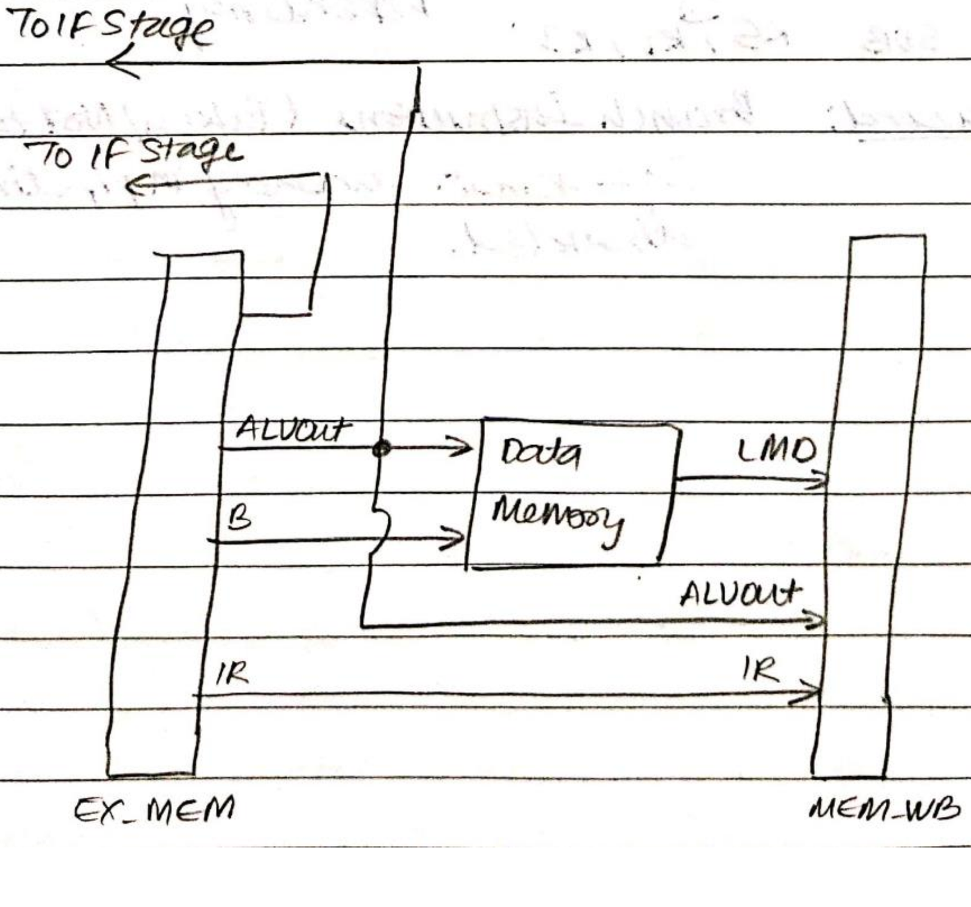
MEM_WB_IR ← EX_MEM_IR;

MEM_WB_LMD ← Mem[EX_MEM_ALUout];

Store:

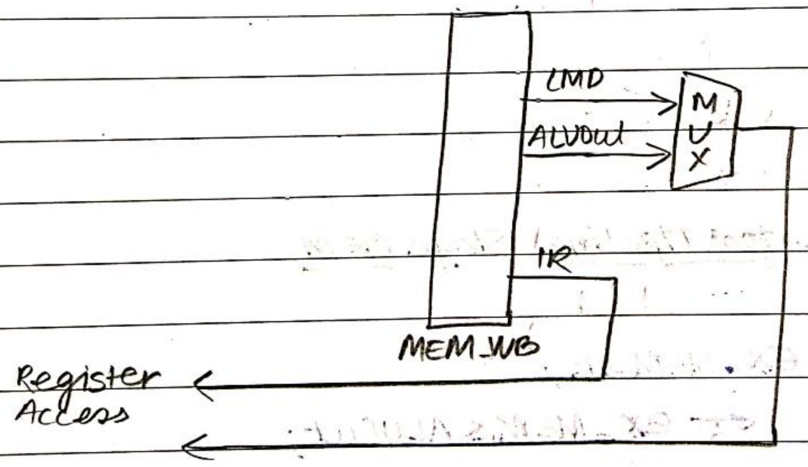
MEM_WB_IR ← EX_MEM_IR;

mem[EX_MEM_ALUout] ← EX_MEM_B;

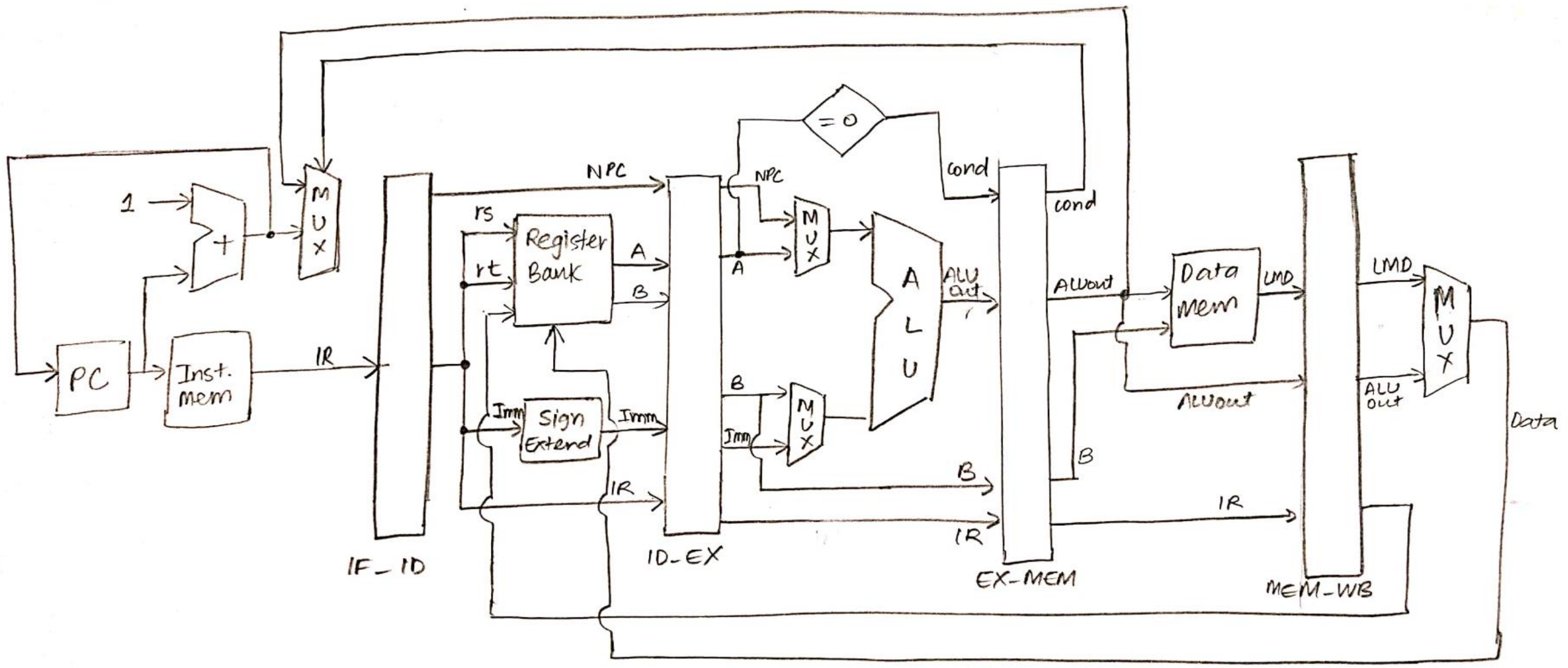


(e) Micro - Operations for Pipeline Stage WB

R-R ALU: Reg[MEM_WB_IR [rd]] ← MEM_WB_ALUOut;
R-M ALU: Reg [MEM_WB_IR [rt]] ← MEM_WB_ALUOut;
LOAD : Reg [MEM_WB_IR [rt]] ← MEM_WB_LMD;

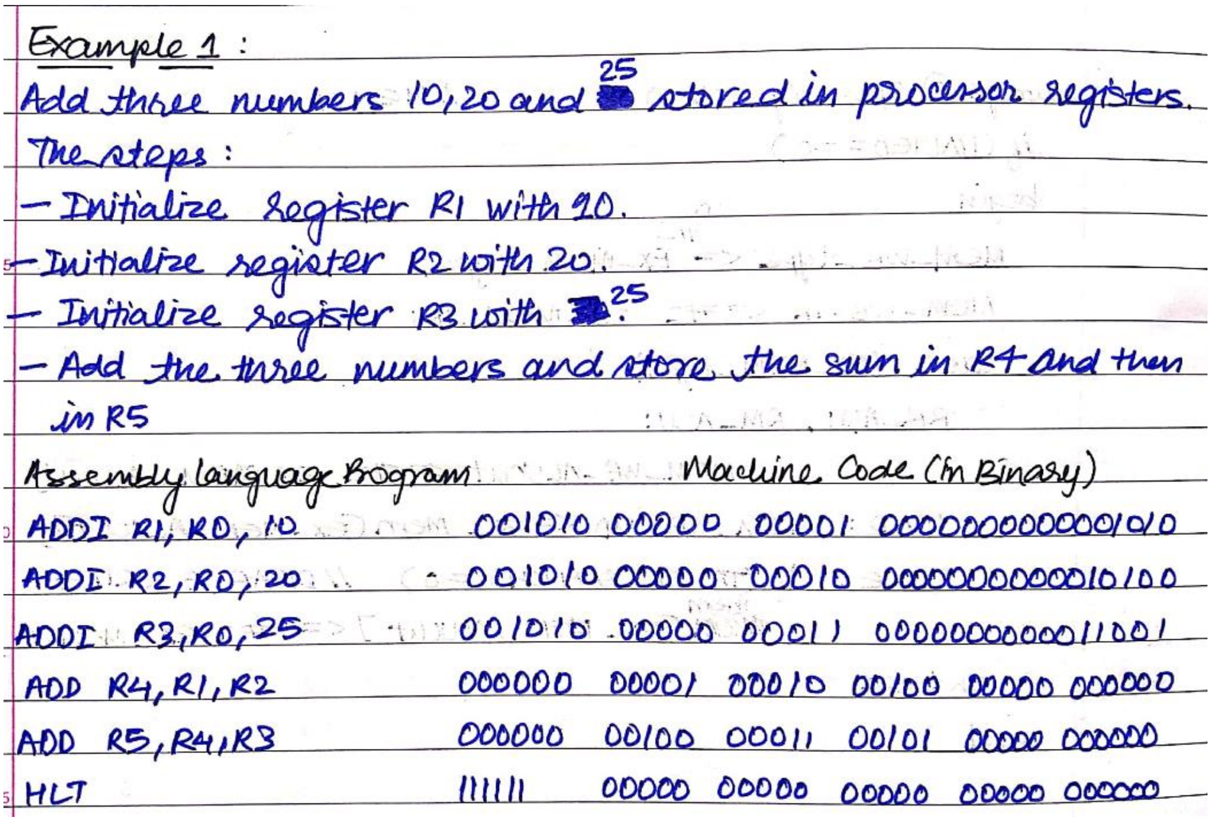


Two special 1-bit variables are used:
- HALTED :: Set after a HLT instruction executes and reaches the WB stage.
- ~~[illegible]~~ TAKEN_BRANCH :: Set after the decision to take a branch is known. Required to disable the instructions that have already entered the pipeline from making any state changes.



PIPELINED - MIPS 32

Test Benches 1:



Example 1 :

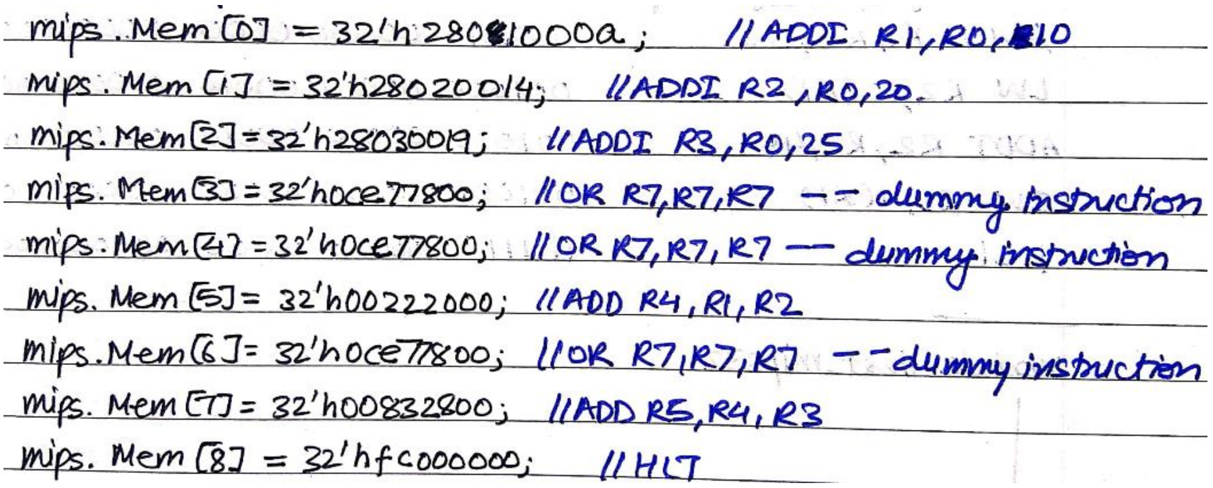
Add three numbers 10, 20 and 25 stored in processor registers.

The steps :

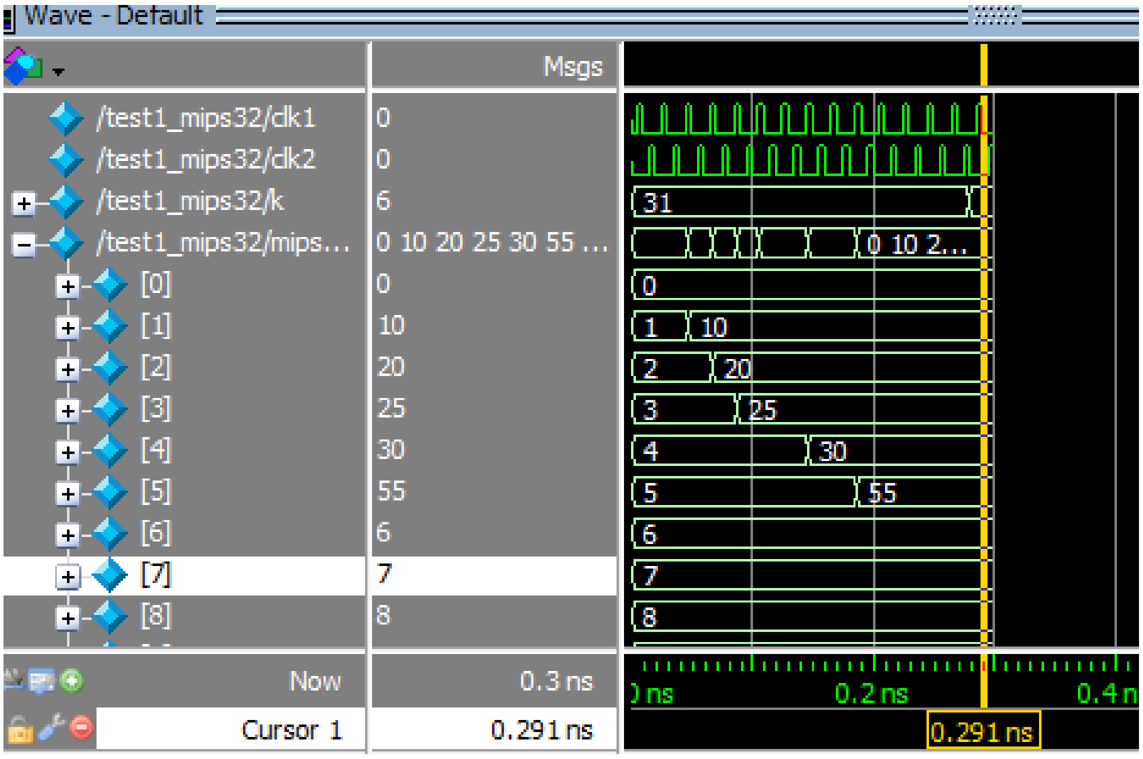
- Initialize register R1 with 10.
- Initialize register R2 with 20.
- Initialize register R3 with 25.
- Add the three numbers and store the sum in R4 and then in R5

Assembly language Program	Machine Code (In Binary)
ADDI R1, R0, 10	001010 00000 00001 0000000000001010
ADDI R2, R0, 20	001010 00000 00010 0000000000010100
ADDI R3, R0, 25	001010 00000 00011 0000000000011001
ADD R4, R1, R2	000000 00001 00010 00100 00000 000000
ADD R5, R4, R3	000000 00100 00011 00101 00000 000000
HLT	111111 00000 00000 00000 00000 000000

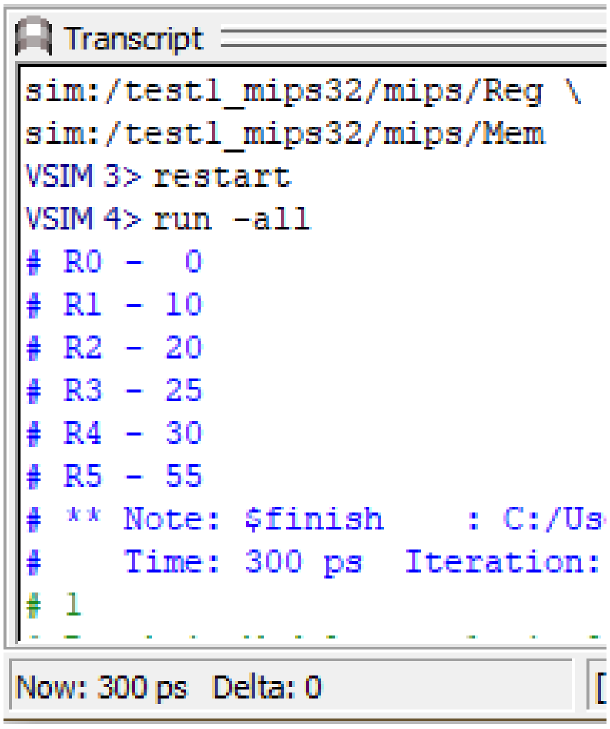
Actual Program:



```
mips.Mem[0] = 32'h2801000a;   // ADDI R1,R0,10
mips.Mem[1] = 32'h28020014;   //ADDI R2,R0,20.
mips.Mem[2] = 32'h28030019;   //ADDI R3,R0,25
mips.Mem[3] = 32'h0ce77800;   //OR R7,R7,R7 -- dummy instruction
mips.Mem[4] = 32'h0ce77800;   //OR R7,R7,R7 -- dummy instruction
mips.Mem[5] = 32'h00222000;   //ADD R4,R1,R2
mips.Mem[6] = 32'h0ce77800;   //OR R7,R7,R7 -- dummy instruction
mips.Mem[7] = 32'h00832800;   //ADD R5,R4,R3
mips.Mem[8] = 32'hfc000000;   //HLT
```



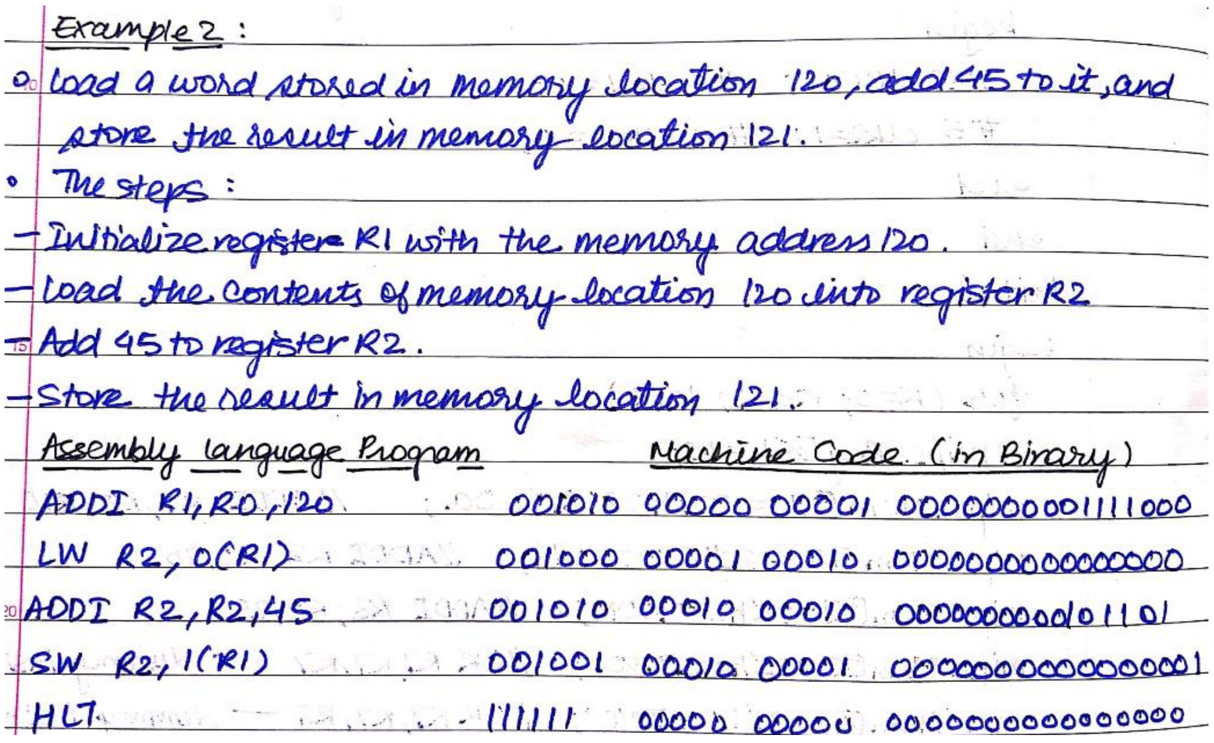
```
Wave - Default
                                Msgs
/test1_mips32/clk1              0
/test1_mips32/clk2              0
/test1_mips32/k                 6
/test1_mips32/mips...           0 10 20 25 30 55 ...
  [0]                           0
  [1]                           10
  [2]                           20
  [3]                           25
  [4]                           30
  [5]                           55
  [6]                           6
  [7]                           7
  [8]                           8
Now                             0.3 ns
Cursor 1                        0.291 ns
```



```
Transcript
sim:/test1_mips32/mips/Reg \
sim:/test1_mips32/mips/Mem
VSIM 3> restart
VSIM 4> run -all
# R0 -   0
# R1 - 10
# R2 - 20
# R3 - 25
# R4 - 30
# R5 - 55
# ** Note: $finish    : C:/Us
#    Time: 300 ps  Iteration:
# 1

Now: 300 ps   Delta: 0
```

Test Bench 2:



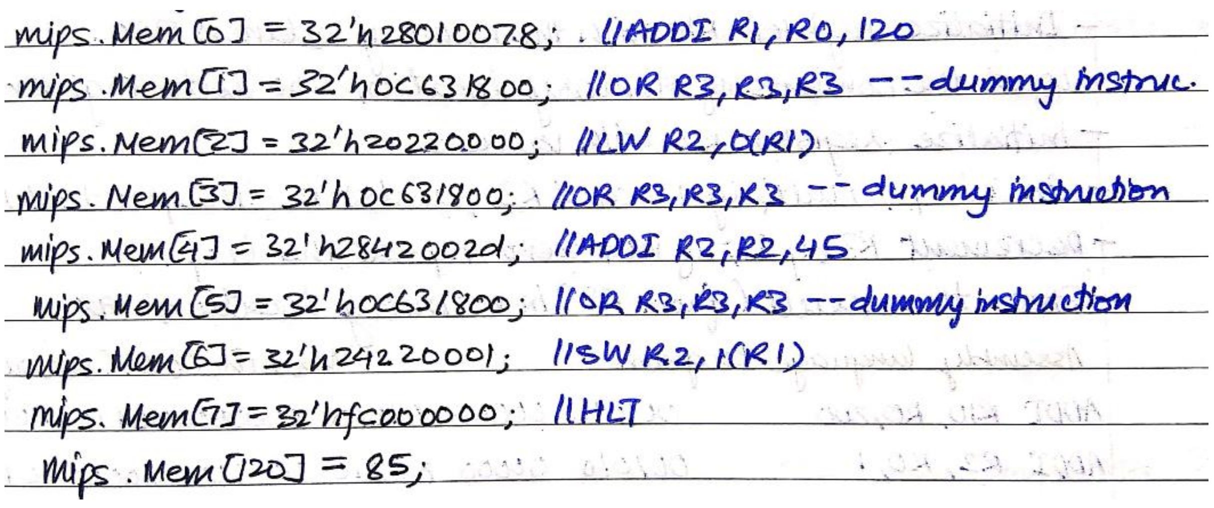
Example 2 :

- Load a word stored in memory location 120, add 45 to it, and store the result in memory location 121.
- The steps :

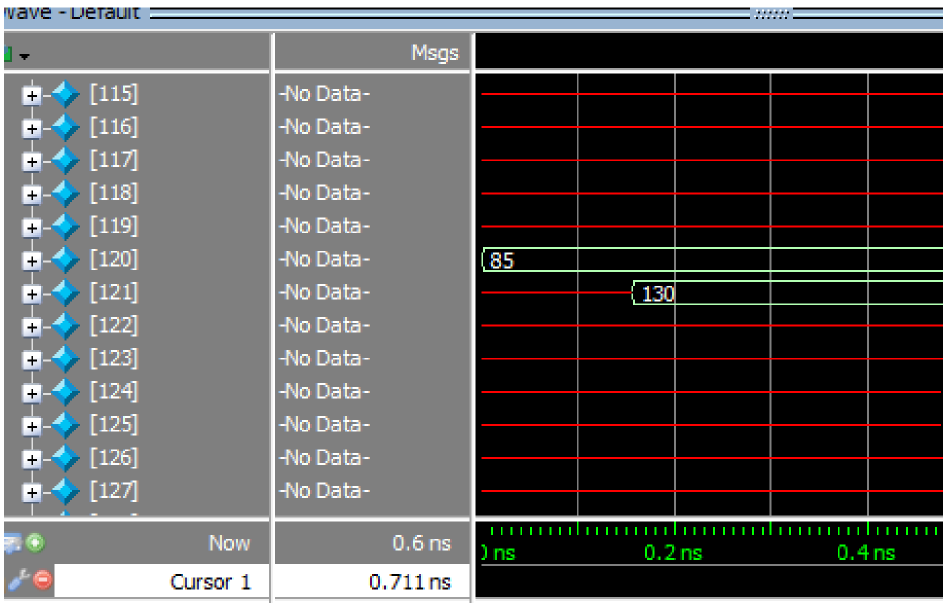
- Initialize register R1 with the memory address 120.
- Load the contents of memory location 120 into register R2
- Add 45 to register R2.
- Store the result in memory location 121.

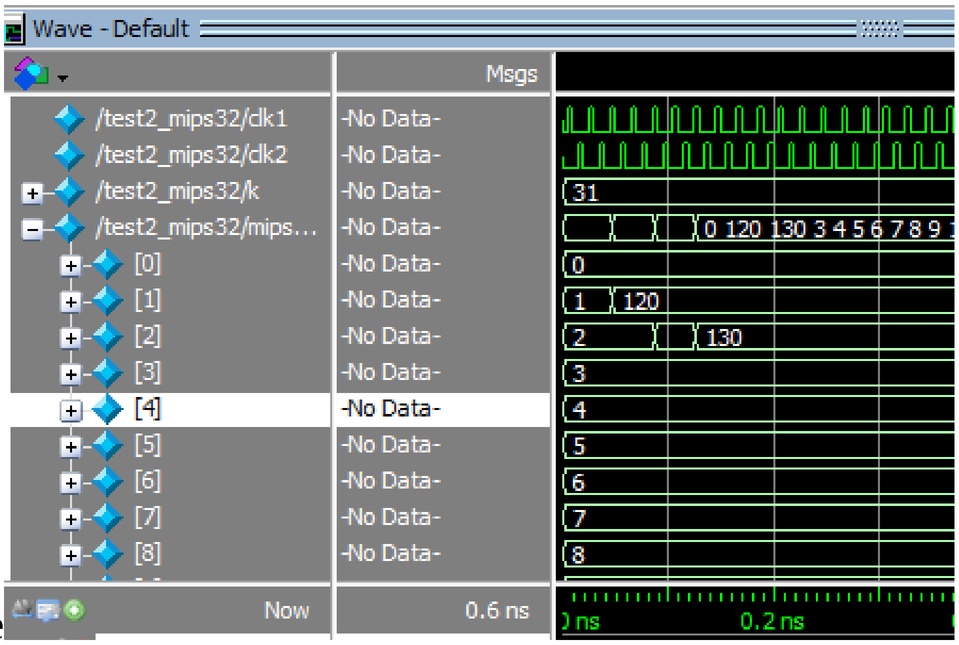
Assembly language Program	Machine Code (in Binary)
ADDI R1, R0, 120	001010 00000 00001 0000000001111000
LW R2, 0(R1)	001000 00001 00010 0000000000000000
ADDI R2, R2, 45	001010 00010 00010 0000000000101101
SW R2, 1(R1)	001001 00010 00001 0000000000000001
HLT	111111 00000 00000 0000000000000000

Actual Program:

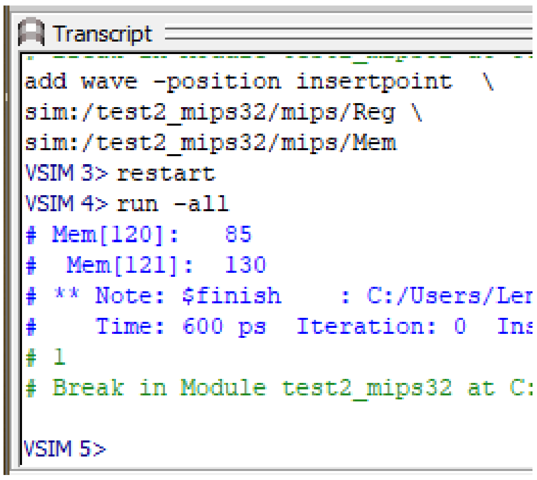


```
mips.Mem[0] = 32'h28010078;  //ADDI R1, R0, 120
mips.Mem[1] = 32'h0c631800;  //OR R3, R3, R3  -- dummy instruc.
mips.Mem[2] = 32'h20220000;  //LW R2, 0(R1)
mips.Mem[3] = 32'h0c631800;  //OR R3, R3, R3  -- dummy instruction
mips.Mem[4] = 32'h2842002d;  //ADDI R2, R2, 45
mips.Mem[5] = 32'h0c631800;  //OR R3, R3, R3  -- dummy instruction
mips.Mem[6] = 32'h24220001;  //SW R2, 1(R1)
mips.Mem[7] = 32'hfc000000;  //HLT
mips.Mem[120] = 85;
```



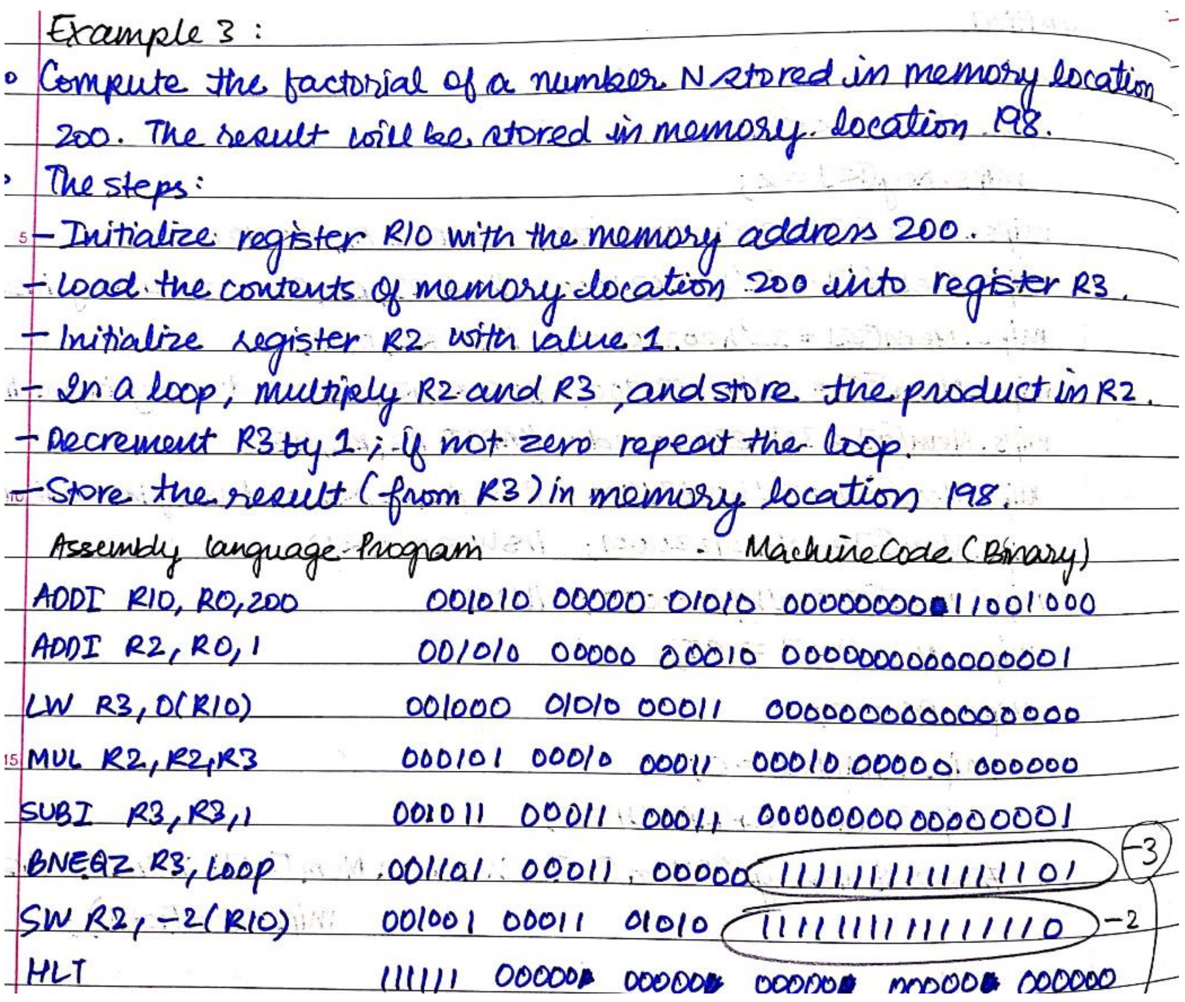


e


Transcript
```
add wave -position insertpoint  \
sim:/test2_mips32/mips/Reg \
sim:/test2_mips32/mips/Mem
VSIM 3> restart
VSIM 4> run -all
# Mem[120]:    85
#  Mem[121]:   130
# ** Note: $finish     : C:/Users/Ler
#    Time: 600 ps  Iteration: 0  Ins
# 1
# Break in Module test2_mips32 at C:

VSIM 5>
```

Test Bench 3:



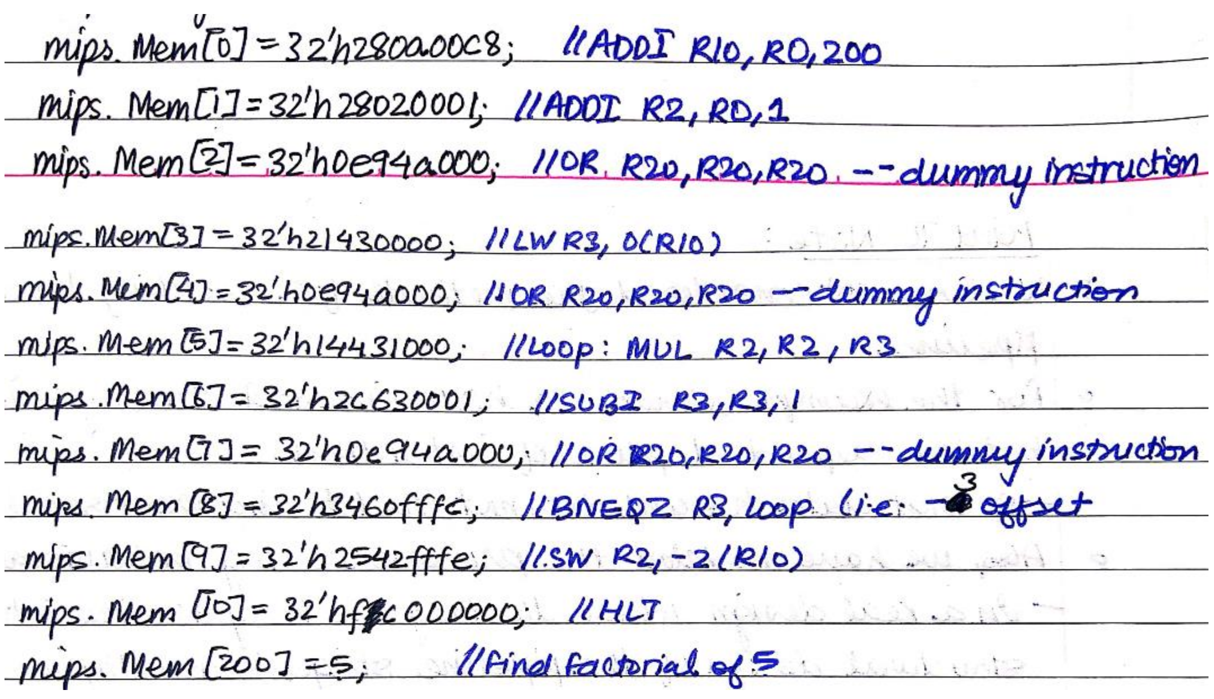
Example 3 :

- Compute the factorial of a number N stored in memory location 200. The result will be stored in memory location 198.
- The steps:

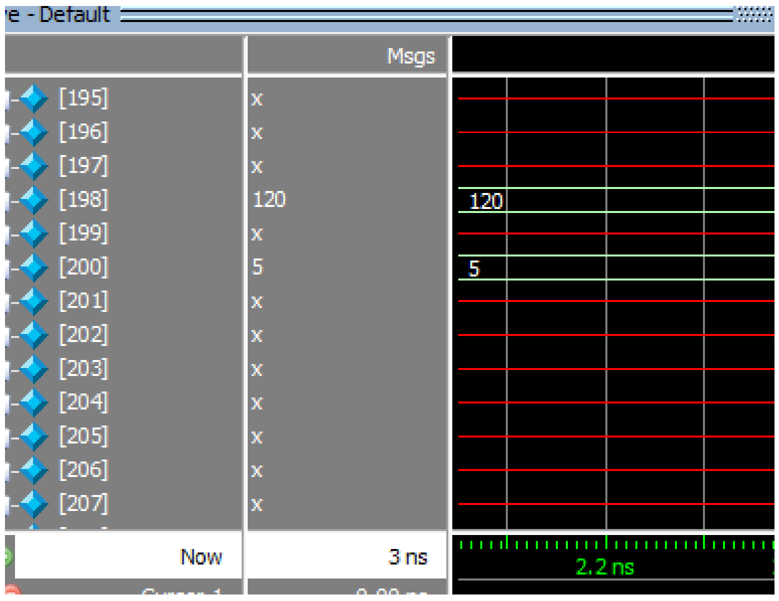
– Initialize register R10 with the memory address 200.
– Load the contents of memory location 200 into register R3.
– Initialize register R2 with value 1.
– In a loop, multiply R2 and R3, and store the product in R2.
– Decrement R3 by 1; if not zero repeat the loop.
– Store the result (from R3) in memory location 198.

Assembly language Program	Machine Code (Binary)
ADDI R10, R0,200	001010 00000 01010 0000000011001000
ADDI R2, R0, 1	001010 00000 00010 0000000000000001
LW R3, 0(R10)	001000 01010 00011 0000000000000000
MUL R2, R2,R3	000101 00010 00011 00010 00000 000000
SUBI R3, R3,1	001011 00011 00011 0000000000000001
BNEQZ R3, Loop	001101 00011 00000 1111111111111101 −3
SW R2, −2(R10)	001001 00011 01010 1111111111111110 −2
HLT	111111 000000 000000 000000 000000 000000

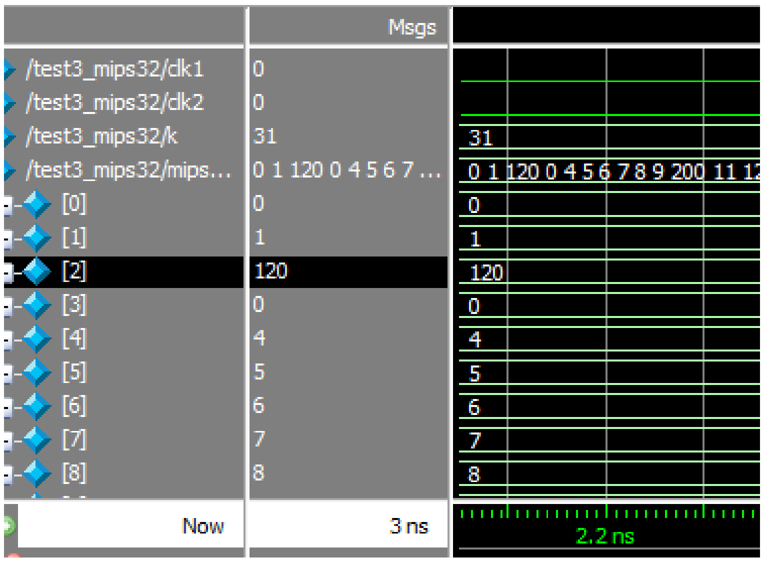
Actual Program:



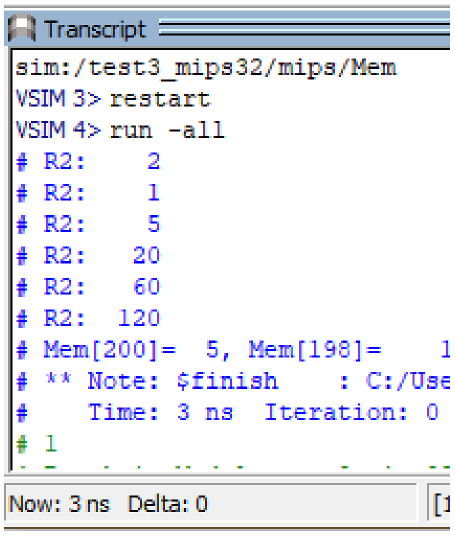
```
mips.Mem[0] = 32'h280a00c8;   //ADDI R10, R0, 200
mips.Mem[1] = 32'h28020001;   //ADDI R2, R0, 1
mips.Mem[2] = 32'h0e94a000;   //OR R20, R20, R20 --dummy instruction
mips.Mem[3] = 32'h21430000;   //LW R3, 0(R10)
mips.Mem[4] = 32'h0e94a000;   //OR R20, R20, R20 --dummy instruction
mips.Mem[5] = 32'h14431000;   //Loop: MUL R2, R2, R3
mips.Mem[6] = 32'h2c630001;   //SUBI R3, R3, 1
mips.Mem[7] = 32'h0e94a000;   //OR R20, R20, R20 --dummy instruction
mips.Mem[8] = 32'h3460fffc;   //BNEQZ R3, Loop (i.e. -3 offset
mips.Mem[9] = 32'h2542fffe;   //SW R2, -2(R10)
mips.Mem[10] = 32'hfc000000;  //HLT
mips.Mem[200] = 5;            //Find factorial of 5
```



```
e - Default
                    Msgs
[195]               x
[196]               x
[197]               x
[198]               120          120
[199]               x
[200]               5            5
[201]               x
[202]               x
[203]               x
[204]               x
[205]               x
[206]               x
[207]               x
Now                 3 ns         2.2 ns
```



```
                         Msgs
/test3_mips32/clk1       0
/test3_mips32/clk2       0
/test3_mips32/k          31                      31
/test3_mips32/mips...    0 1 120 0 4 5 6 7 ...   0 1 120 0 4 5 6 7 8 9 200 11 12
[0]                      0                       0
[1]                      1                       1
[2]                      120                     120
[3]                      0                       0
[4]                      4                       4
[5]                      5                       5
[6]                      6                       6
[7]                      7                       7
[8]                      8                       8
Now                      3 ns                    2.2 ns
```



```
Transcript
sim:/test3_mips32/mips/Mem
VSIM 3> restart
VSIM 4> run -all
# R2:     2
# R2:     1
# R2:     5
# R2:    20
# R2:    60
# R2:   120
# Mem[200]=   5, Mem[198]=     1
# ** Note: $finish    : C:/Use
#    Time: 3 ns  Iteration: 0
# 1
Now: 3 ns   Delta: 0
```