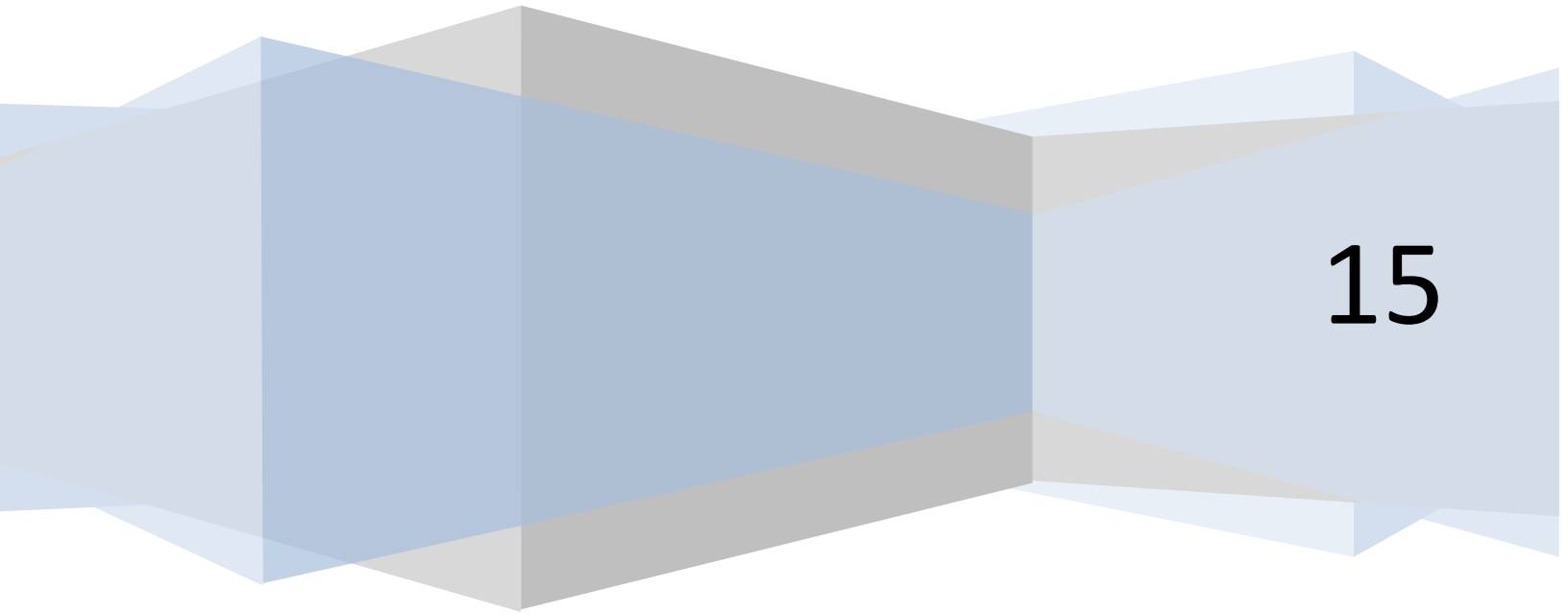
Single Cycle MIPS Processor

DR: CHERIF SALAMA



15

PROJECT TEAM MEMBERS:

1- **Amr Mohamed Ahmed Mahmoud**
2- **Marwa Abd El Rahman Mohamed**
3- **Mona Mohamed Ali Matar**
4- **Nouran Mustafa Mustafa**
5- **Omnia Ahmed Abd El Monaem**

An instruction set architecture is an interface that defines the hardware operations which are available to software.

Any instruction set can be implemented in many different ways:

— In a basic single-cycle implementation all operations take the same amount of time—a single cycle.

— Next, pipelining lets a processor overlap the execution of several instructions, potentially leading to big performance gains.

A single-cycle MIPS processor consists of:

1) Program counter:

The program counter or PC register holds the address of the current instruction. MIPS instructions are each four bytes long, so the PC should be incremented by four to read the next instruction in sequence.

2) Instruction Memory:

The Instruction Memory reads the address of instruction from the program counter.

3) Control unit:

The control unit is responsible for setting all the control signals so that each instruction is executed properly.

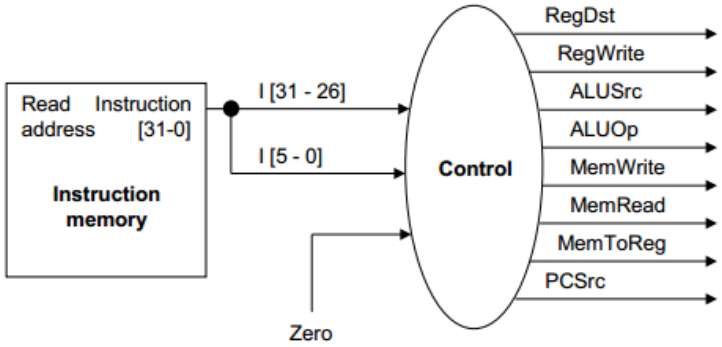
The control unit needs 13 bits of inputs.

— Six bits make up the instruction's opcode.

— Six bits come from the instruction's function field.

— It also needs the Zero output of the ALU. The control unit generates 9 bits of output (PCsrc=Zero) shown in figure below:

- The Output signals from the control unit:



4) Register File:

A register file is a means of memory storage within a computer's central processing unit (CPU). The computer's register files contain bits of data and mapping locations. These locations specify certain addresses that are input components of a register file.

5) Arithmetic-Logic Unit:

An arithmetic-logic unit (ALU) is the part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer instruction word. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU).

6) Multiplexer:

The multiplexer shortened to "MUX" is a combinational logic circuit designed to switch one of several input lines through a single common output line by the application of a control signal.

7) Adder:

One of them is used to increment the value of program counter by a constant of value equal 4 to make the program counter points to address of the next instruction that should by executed in the next clock cycle.

Another adder in the MIPS processor is used in preparing the address of branch instruction by adding two operands together { The sign extended offset shifted by two and the value of program counter+4}.

8) Data Memory:

Data cache is used to caching the values used in the program many times from memory to make execution faster because direct access to memory takes a long time.

9) ALU control unit:

This unit is very important because it controls the different types of operations performed in the ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less than	101010	set on less than	111

The different instructions supported by MIPS single-cycle processor are:

1. Arithmetic: Add, addi.
2. Load/store: lw, sw.
3. Logic: sll, and, andi, or, ori, nor.
4. Control flow: beq, jal, Jr.
5. Comparison: slt.

In addition to the implementation of single-cycle MIPS processor, we implement two of the bonus features:

1-Implementing a simple assembler to allow the user to supply programs in assembly language and the assembler will translate the programs into appropriate binary format and saving it.
2-writing set of test programs of different level of complexity.

The tools used in implementing the MIPS single-cycle processor are:

1-Modelsim program that helps in writing the code and simulate it.
2-Using reference {computer organization and design 4th edition} as a guide.
3-Searching for the information we want using the internet.
The language used for implementing and describing the hardware is verilog.

The Total work done by the team To Implement a single cycle MIPS processor.

First of all we started by making a kick off meeting to discuss the following items:

1- The scope of the project.
2- The tasks that should be done to finish the project on time.
3- The functions supported by the MIPS processor.
4- Discuss which two bonus features to implement in MIPS processor.
5- Making a good plan to manage the project perfectly.
6- Doing a suitable time schedule.

The scope of the project is to write and test a simplified verilog description of the single-cycle MIPS data path.

The Tasks of the project:

1-Writing the codes of each component included in the processor [Program counter, Instruction memory, Register file, Alu, Data memory, Multiplexer, Adder].

2-Testing the individual component to ensure that the unit performs the intended function.

3-Connect all the components using test bench.

4-Testing the processor as one entity.

5-Writing the programs used in testing the MIPS processor using machine language.

6-Writing the report that should include the following:

6.1-A brief description of the implementation including any bonus features

6.1.1-This should also include any tools/languages used for implementation or simulation.

6.2- A summary of how the work was split among the team members.

6.3-A user guide including a full simulation example step by step with snapshots.

6.4-A list of programs simulated.

6.5-Stating the number of clock cycles needed to simulate it to completion and describe its expected final state.

6.6-For each program show the actual final state obtained by running the simulation with snapshots.

7-Finally adding the bonus features to our MIPS single-cycle processor.

The functions supported by MIPS

1. Arithmetic: Add addi.
2. Load/store: lw, sw.
3. Logic: sll, and, andi, or, ori, nor.
4. Control flow: beq, jal, Jr.
5. Comparison: slt.

The plan was started by splitting the entire component on team members. The table below describes how the work was organized

Name	The task assigned
1-Mona Mohamed Ali [Team leader]	1-Manage the work of team 2- write the codes of -program counter -Multiplexers -PC adder -Shift left -Sign extend -Jump address module 3- writing the project report
2-Nouran Mustafa	1-Alu module 2-Alu control unit module 3-ext_zero module 4-Alu_branch module
3-Amr Mohamed	1-Register file 2-sign extend 3-Final design of test bench 4-Writes the code of the assembler
4-Omnya Ahmed	1-Data memory with initialization. 2-Instruction memory 3-writing programs and inverting them into machine code.
5-Marwa Abd El Rahman	1-Control unit 2- The Test programs{12 program}.

Shared Tasks:

1-The Integrated circuit was connected by {Nouran Mustafa & Mona Mohamed}.

2-All The Team memebers work in Testing The Single cycle MIPS processor Implementation.

User Guide To help In The Implementing Single-cycle Processor:

Overview of single cycle implementation processor:

Single Cycle MIPS processor is one of many processors used in computers.

1-1 The Performance Perspective:

Performance of a machine is determined by:

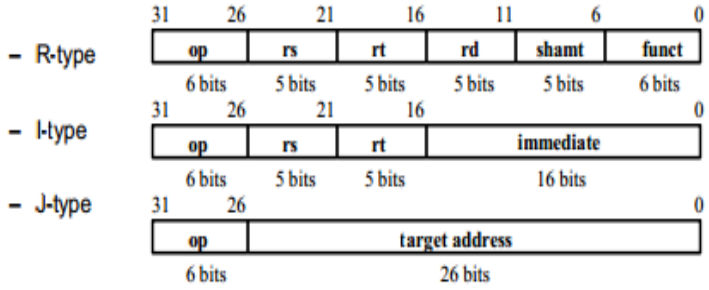
– Instruction count
– Clock cycle time
– Clock cycles per instruction

Processor design {Datapath &control} will determine:

– Clock cycle time
– Clock cycles per instruction
– Single cycle processor:
- **Advantage:** One clock cycle per instruction
- **Disadvantage:** long cycle time

1-2 The MIPS Instruction Formats:

All MIPS instructions are 32 bits long. The three instruction formats:
R-type , I type , J type.



The different fields are:

– OP: operation of the instruction.
– Rs, Rt, Rd: the source and destination register specifiers.
– Shamt: shift amount.
– Function fields: selects the variant of the operation in the “op” field.
– Address / Immediate: address offset or immediate value.
– Target address: target address of the jump instruction.

1-3 The supported instruction:

- Arithmetic : add , addi.
- Load/Store : lw, sw.
- Logic: sll, and , andi , or , ori , nor.
- Control flow : beq , jal , jr.
- Comparison : slt.

2- User Interface:

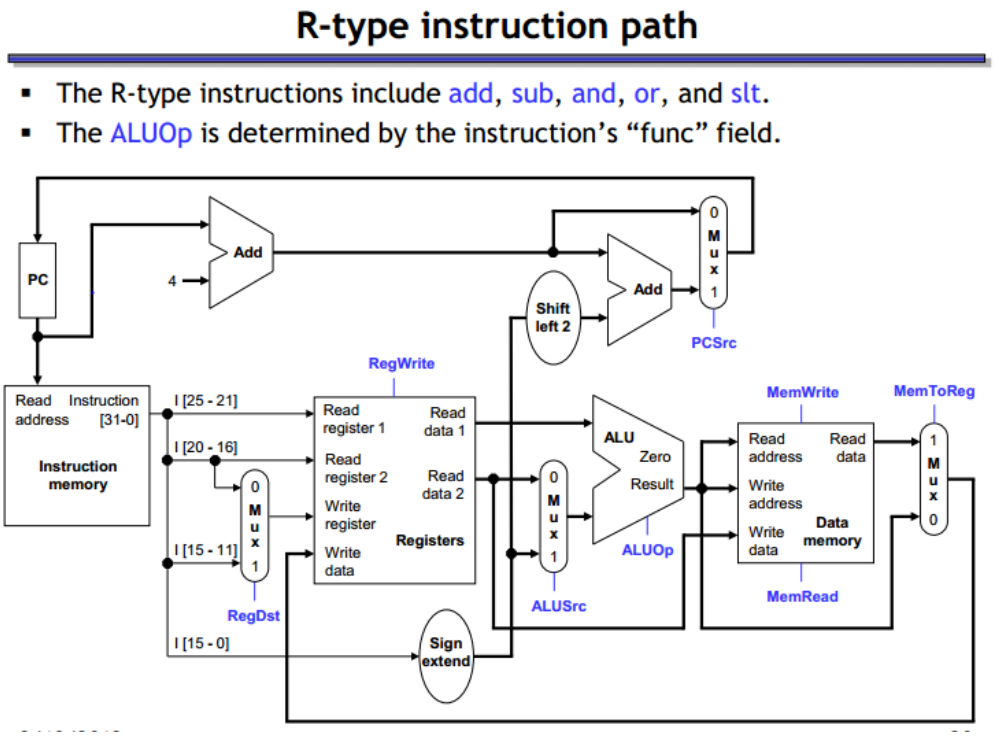
The user writes the program in MIPS then it is converted to machine language {0&1} by the assembler.

The converted instructions are saved in file and the file is saved in the instruction memory.

The program counter {PC} points each clock cycle at an instruction.

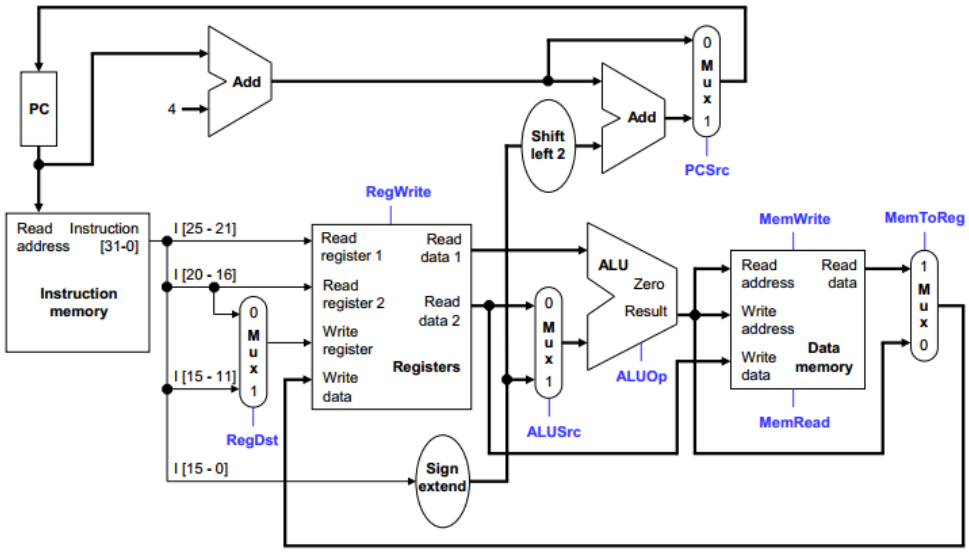
Instruction by instruction is executed and the final output is released.

3- The Design of Single-Cycle MIPS Implementation:

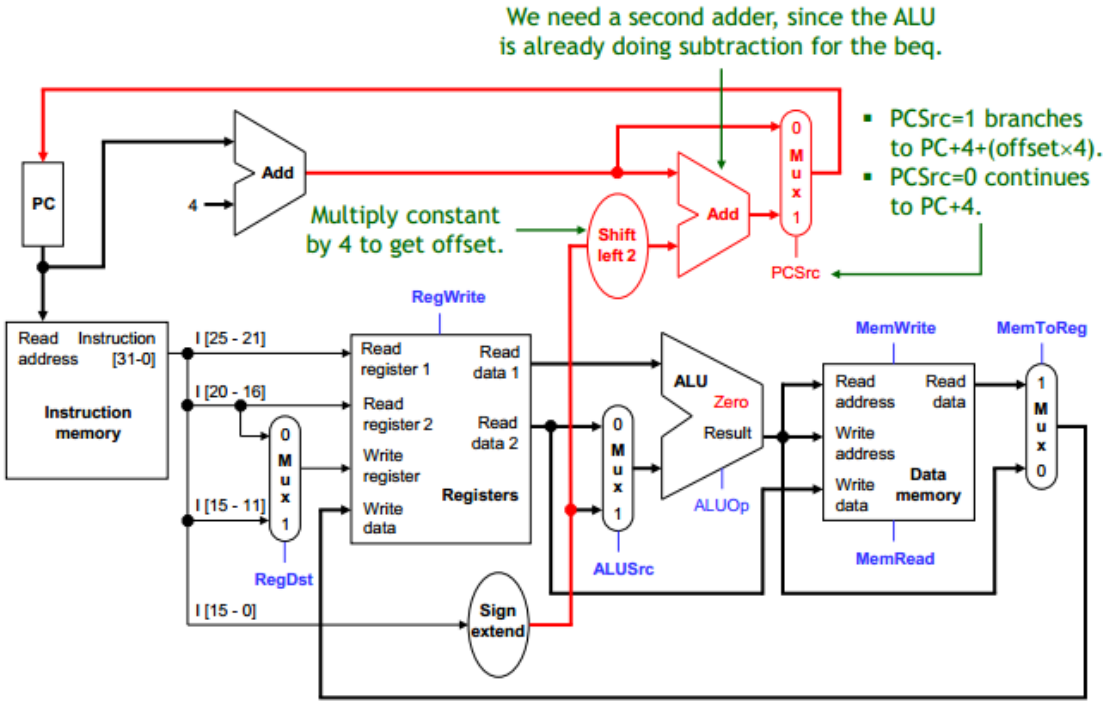


sw instruction path

- An example store instruction is sw $a0, 16($sp).
- The ALUOp must be 010 (add), again to compute the effective address.



Branching hardware



The steps in executing a beq

1. Fetch the instruction, like beq $at, $0, offset, from memory.
2. Read the source registers, $at and $0, from the register file.
3. Compare the values by subtracting them in the ALU.
4. If the subtraction result is 0, the source operands were equal and the PC should be loaded with the target address, PC + 4 + (offset x 4).
5. Otherwise the branch should not be taken, and the PC should just be incremented to PC + 4 to fetch the next instruction sequentially.

Outputs of the testing programs:

1-**Program 1**:

lw $t0 4 ($0)	$t0=2
add $t1 $t0 $0	$t1=2
addi $t1 $t1 3	$t1=5
sw $t1 0 ($0)	Storing $t1=5

Number of clock cycle=4

The simulated output is:

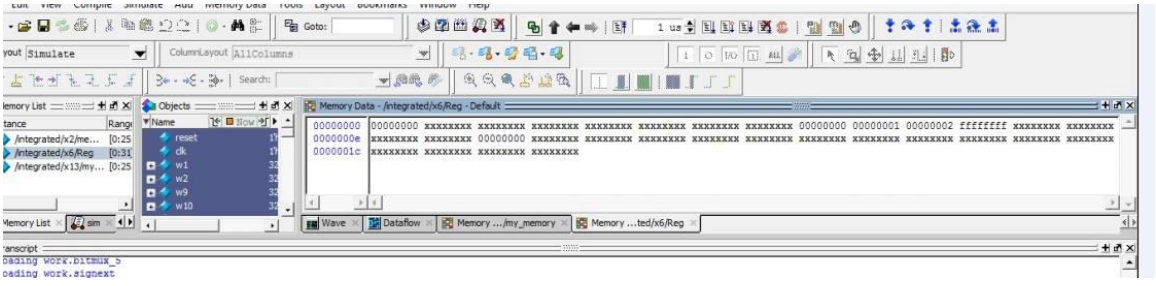
```
00000000 00000000 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 00000002 00000005
0000000a xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000014 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001e xxxxxxxx xxxxxxxx
```

2-Program 2:

Logic gate half adder:

Lw $t0,0 ($zero)	$t0=1
Lw $t2, 4($zero)	$t2=2
and $s7, $t0, $t2	$s7=0
Addi $s0,$zero,0	$s0=0
Nor $t1,$t0,$s0	
Nor $t3,$t2,$s0	
And $t4,$t0,$t3	
And $t5, $t2,$t1	
Or $t6,$t4,$t5	

Number of clock cycles= 9 cycles.



3-program 3:

lw $s1 4 ($0)	$S1= 2
lw $s2 8 ($0)	$S2=3
lw $s3 ,12($0)	$S3=4
lw $s4, 16($0)	$S4=5
sll $s2, $s2, 2	$S2=12
or $s1 ,$s1, $s2	$S1=14
sll $s3, $s3, 2	$S3=16
or $s1 ,$s1, $s3	$S4=30
sll $s4, $s4, 2	$S4=20
or $s1, $s1, $s4	$S1=30

Number of clock cycles = 10cycles

The simulated output is:

```
00000000 00000000 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000000a xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 0000001e 0000000c 00000010
00000014 00000014 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001e xxxxxxxx xxxxxxxx
```

4-Program 4:

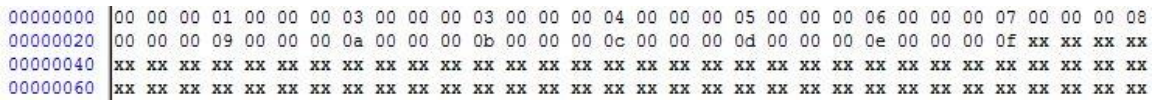
lw $t0 ,0($0)	$t0=1
lw $s2, 4($0)	$s2=2
lw $s4, 8($0)	$s4=3
lw $t1 ,12($0)	$t1=4
lw $s3 ,16 ($0)	$s3=5
add $t0, $s2, $t0	$t0=3
add $t1, $s4, $s4	$t1=6
add $t1, $t1, $t1	$t1=12
add $t1, $t1, $s3	$t1=17
sw $t0, 4($0)	Data memory[4]=3

Number of clock cycles = 10 cycles

value of registers in register file is

```
00000000 00000000 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 00000003 00000011
0000000a xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 00000002 00000005
00000014 00000003 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001e xxxxxxxx xxxxxxxx
```

value of data memory is

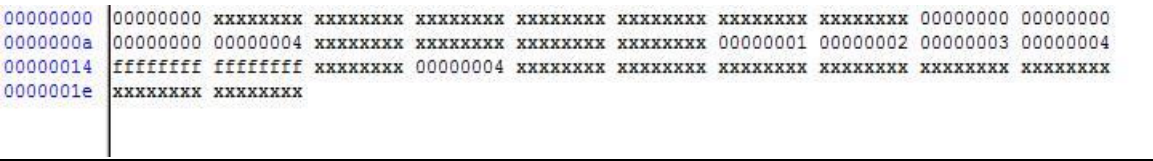


5-Program 5:

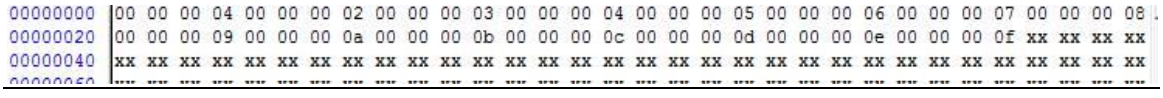
lw $s0 ,0($0)	**$s0=1**
lw $s1,4($0)	**$s1=2**
lw $s2 ,8($0)	**$s2=3**
lw $s3 12 ($0)	**$s3=4**
addi $s4 $0 0	**$s4=0**
addi $s5 $0 0	**$s5=0**
and $t0 $s0 $s4	**$t0=0**
and $t0 $t0 $s5	**$t0=0**
nor $s4 $s4 $0	**$s4=32'b1**
and $t1 $s1 $s4	**$t1=2**
and $t1 $t1 $s5	**$t0=0**
nor $s4 $s4 $0	**$s4=0**
nor $s5 $s5 $0	**$s5=32'b1**
and $t2 $s2 $s4	**$t2=0**
and $t2 $t2 $s5	**$st2=0**
nor $s4 $s4 $0	**$s4=32'b1**
and $t3 $s3 $s4	**$t3=4**
and $t3 $t3 $s5	**$t3=4**
or $s7 $t0 $t1	**$s7=0**
or $s7 $s7 $t2	**$s7=0**
or $s7 $s7 $t3	**$s7=4**
sw $s7 0 ($0)	**Storing in memory value of 4**

Number of clock cycle: 23 cycle

The simulated output of register file is:



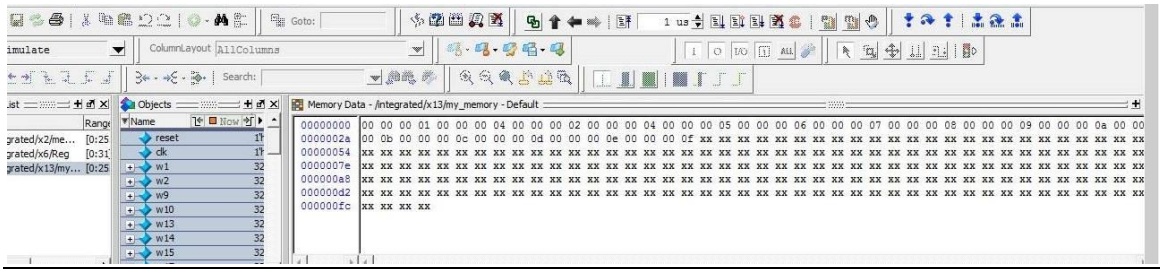
Memory output:



6-Program 6 :

lw $t1, 0 ($0)	$t1=1
lw $t2 ,4 ($0)	$t2=2
and $t1, $t1, $t2	
andi $t2 ,$t2 ,3	
ori $t1, $t1, 4	
sw $t1, 4($0)	
sw $t2 ,8 ($0)	

Number of clock cycles = 7 cycles.



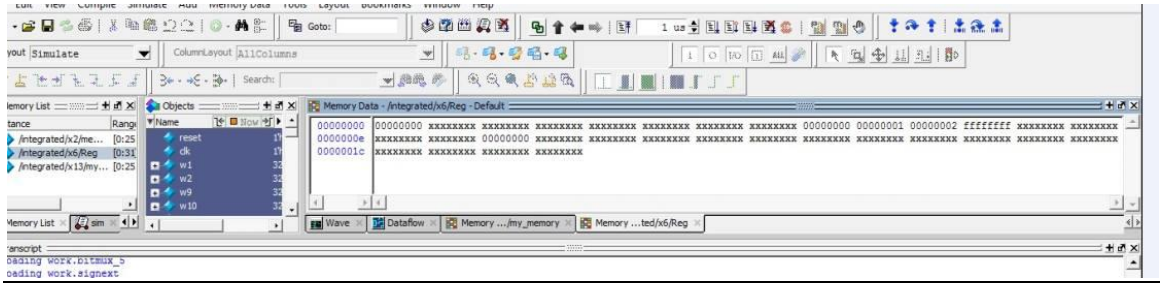
7-Program7:

logic gate (NAND)

lw $t1 0 ($0)	$t1=1
lw $t2 4 ($0)	$t2=2
addi $s0 $0 $0	$s0=0
and $t0 $t1 $t2	$t0=0
nor $t3 $t0 $s0	$t3=1

Number of clock cycle=5 cycle

The simulated output is:



8-Program8:

Logic gate xor

Lw $t0, 0($zero)	
Lw $t2, 4 ($zer0)	
Addi $s0, $zero, 0	
Nor $t1, $t0,$s0	
Nor $t3,$t2,$s0	
And $t4,$t0,$t3	
And $t5,$t2,$t1	
Or $t6,$t4,$t5	

Number of clock cycles=8 cycles