G9 Processor

Modules:

- ProgramCounter(clk, reset,branch, pc_branch_target,pc_4, pc) contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence.

- SignExtend(input [15:0] imm16, output [31:0] imm32): fits the 16 bit signed binary value to 32 bit signed binary value.

- ControlUnit(input[31:0] instruction, output reg[2:0] alu_op, output reg mem_read,mem_write,alu_src,mem_to_reg,reg_write, b,br,bz,bnz,bcy,bncy,bs,bns,bv,bnv,Call,Ret) tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been received.

- RegisterFile(clk, reset , reg_write, read_reg_1, read_reg_2, write_register, write_data, read_data_1, read_data_2, led_output) used to stage data between memory and the functional units on the chip.

- ArithmeticLogicUnit(alu_control, operand0, operand1, ALUResult, carryflag, signflag, overflowflag, zeroflag) contains the logical circuit to perform mathematical operations like subtraction, addition, multiplication, division, logical operations and logical shifts on the values held in the processors registers. A 32-bit processor is one with a 32-bit ALU.

- DataMemory(clk,reset,address,mem_write,mem_read,write_data,read_data) serves for storing and keeping data required for the proper operation of the programs.

- InstructionMemory(clka, rsta, addra, douta) holds the instruction currently being executed or decoded.

Timing Report:

Minimum period: 13.939ns (Maximum Frequency: 71.743MHz)
Minimum input arrival time before clock: 5.412ns
Maximum output required time after clock: 6.306ns
Maximum combinational path delay: No path found

OPCODES used for instruction memory:

op1	**rs**	**rt**	**imm**
6 bits	5 bits	5 bits	16 bits

op2	**rs/ra**	00000	**imm**
6 bits	5 bits	5 bits	16 bits

op3	00000	00000	**imm**
6 bits	5 bits	5 bits	16 bits

OP1		
OPERATION		**OPCODE**
ARITHMETIC	*add*	000000
	comp	000010
LOGICAL	*and*	000100
	xor	000101
SHIFT	*shllv*	001110
	shrlv	010000
	shrav	010010
MEMORY	*lw*	000100
	sw	000101

OP2		
OPERATION		**OPCODE**
ARITHMETIC	*addi*	000001
	compi	000011
SHIFT	*shll*	001100
	shrl	001101
	shra	010001
BRANCH	*br*	010101
FUNCTION	*Call*	011110
	Ret	011111

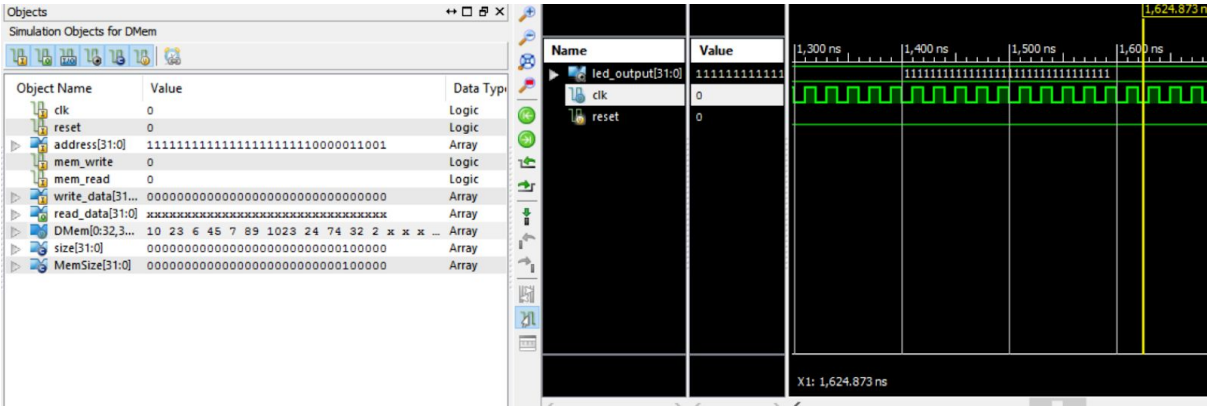
OP3		
OPERATION		**opcode**
BRANCH	*b*	010100
	bz	010110
	bnz	010111
	bcy	011000
	bncy	011001
	bs	011010
	bns	011011
	bv	011100
	bnv	011101

REGISTER FILE:

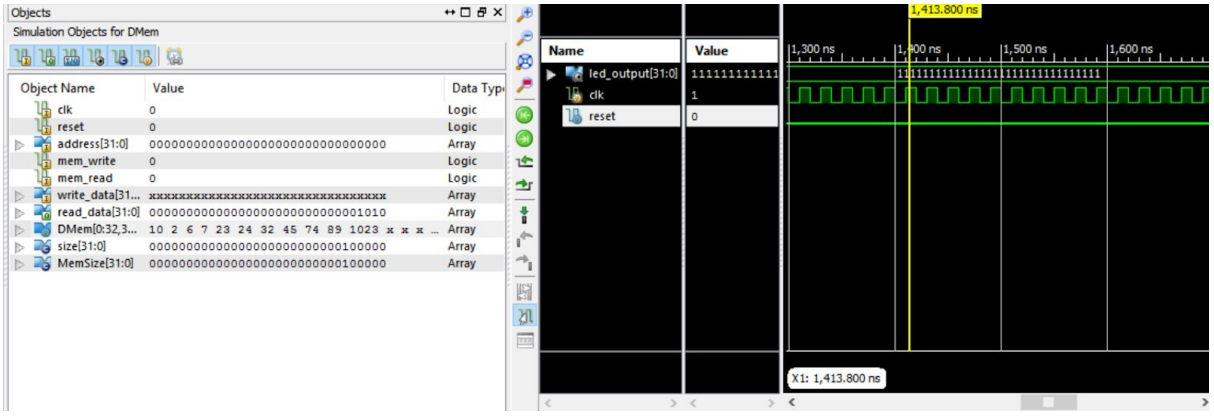
REGISTER	Value
rs	00000
rt	00001
ra	11111
a0-a3	00010-00101 (2-5)
v0-v3	00110-01001 (6-9)
t0-t7	01010-10001 (10-17)
s0-s7	10010-11001 (18-25)

Data Memory before Sorting:

First element represents the number of integers in an array.



Data Memory after Sorting:



Verifying Implementation :

Input : clk
Output : sorted numbers every 4th clk cycle *
Implemented Algo : Bubble sort
*Initial output is 11111111111111111111111111111111 which means sorting is not yet completed. After sorting is done the sorted numbers are returned every 4th clock cycle.

Bubble Sort

```
## function  params ##
xor $a1,$a1
lw $a0, 0($a1)          # num of elements
addi $a1,1              # base address of array in mem
########################
# BUBBLE SORT Starts #
########################
comp $s1,$a0
xor $t0,$t0             # OUTER LOOP Begins
add $t0,$a1             # ptr to base address of array
xor $t1,$t1
add $t0,$zero           # INNER LOOP Begins
lw $t2, 1($t0)
lw $t3, 0($t0)
xor $t4,$t4
add $t4,$t2
comp $t5,$t3
add $t4,$t5
bns 4                   # branch NO_SWAP
sw $t2, 0($t0)
sw $t3, 1($t0)
xor $t1,$t1
addi $t1,1
addi $t0,1              # NO_SWAP
xor $t6,$t6
add $t6,$t0
add $t6,$s1
bs -17                  # branch INNER LOOP
add $t1,$zero
bnz -22                 # branch OUTER LOOP
#####################
# BUBBLE SORT Ends #
#####################
lw $s7, 0($a1)          #load result elems
add $s7,$zero
addi $a1, 1
addi $a0,-1
bnz -5
```

In Binary:

00010100011000110000000000000000
00100000011000100000000000000000
00000100011000000000000000000001
00001010011000100000000000000000
00010101010010100000000000000000
00000001010000110000000000000000
00010101011010110000000000000000
00000001010000000000000000000000
00100001010011000000000000000001
00100001010011010000000000000000
00010101110011100000000000000000
00000001110011000000000000000000
00001001111011010000000000000000
00000001110011110000000000000000
01101100000000000000000000000100
00100101010011000000000000000000
00100101010011010000000000000001
00010101011010110000000000000000
00000101011000000000000000000001
00000101010000000000000000000001
00010110000100000000000000000000
00000010000010100000000000000000
00000010000100110000000000000000
01101000000000001111111111101111
00000001011000000000000000000000
01011100000000001111111111101010
00100000011110010000000000000000
00000011001000000000000000000000
00000100011000000000000000000001
00000100010000001111111111111111
01011100000000001111111111111011
01011100000000001111111111111011