CONTENTS

ABSTRACT

Jarvis 7 is a 24 bit microprocessor with a 4 stage pipeline and a minimalist RISC instruction set. The motivation behind its design being that an efficient Instruction Set Architecture when implemented over simplified hardware can increase performance by leaps and bounds.

A compact ISA can cause a drastic reduction in hardware and subsequent decrease in power consumption and cost, besides lessening the code footprint and giving additional flexibility to the assembly programmer.

The report is an insight into the development process right from the innovations in the ISA to the implementation in VHDL, a birds-eye view of Jarvis 7 against a backdrop of constraints and trade-offs, its strengths and uniqueness pitched against commercially available microprocessors.

INTRODUCTION

“Success will not lower its standards to us,

We must raise our standards to success.”

The conception of Jarvis 7 took place at the beginning of the EC-252 Computer Architecture and Microprocessors course with the aim to accentuate the theory with practice. Yet, over the course of its making, Jarvis 7 has substantially increased our understanding of microprocessors, their working, the intricacies of their instruction sets, the philosophy behind their design and provided us with a memorable lesson in teamwork.

The name Jarvis stands for “**Just A Rather Very Intelligent System**”, the personal Artificial Intelligence program of Tony Stark, protagonist of Iron Man, which assists him in the construction and programming of the Iron Man suit. “7” in the name signifies the strength of the group.

- The project began with the brainstorming over an efficient Instruction Set Architecture which laid the foundation of the processor. All the optimization done in the ISA would later trickle down as efficacies in the hardware. This was the most important step in the entire design process.
- The next step forward was the design of a datapath for executing the developed ISA. It was followed by creation of datapath modules on VHDL for testing and simulation. This step was the most time consuming as it included numerous sessions of code creation, refactoring and debugging.
- Once the alpha release was tested, the project proceeded to the survey and comparison stage. Its importance cannot be overlooked as it evaluated the processor. The findings were rather surprising and the unique selling point of the project.

This report is a firsthand account of the entire design and development process and hopes to provide all details of Jarvis 7 in as comprehensive a manner as possible.

PURPOSE

The goal of our study and related research work was to accomplish the following objectives during the period of our learning -

1) Design and Simulation of a pipelined processor using VHDL.

2) Developing a reduced instruction set for this processor and an efficient ISA.

3) Study of advanced features like Branch Prediction, high performance cache memory and Instruction Level Parallelism and scope of their inclusion.

4) Implementation and testing of this design on a Xilinx FPGA Spartan 3E kit.

5) Comparison with a market available processor of similar features.

In order to accomplish the stated purpose, we divided the work under the following objective heads and devoted our energies in realizing one objective at a time.

OBJECTIVES

1) Instruction Set Architecture and Datapath Design.
    a. Developing a minimalist ISA
    b. Selecting minimum number of Instructions.
    c. Selecting optimum number of general purpose registers.
    d. Developing the datapath to implement it.

2) VHDL Implementation.
    a. Implementing and Testing the Design on a Xilinx FPGA Spartan 3E kit.

3) Survey and Comparison.
    a. Pitching JARVIS-7 against commercially available Processors.

LEARNING OUTCOMES

The take home from the project is the knowledge of ISA development, real time implementation and development of a microprocessor, pipelining and its incorporation into the datapath. The following are the learning outcomes listed in greater detail:

1) UNDERSTANDING THE CONCEPTS OF COMPUTER ARCHITECTURE

"Because if you have a strong foundation, then you can build or rebuild anything on it. But if you've got a weak one you can't build anything"

-Jack Scalia

We studied various computer architectures from stack based machines to accumulator machines to load store machines, analyzed their pros and cons and selected a load store machine as it best served our purpose.

2) STUDY OF DIFFERENT INSTRUCTION SET ARCHITECTURES

We studied in detail the various aspects of CISC (Complex Instruction Set Computers) and RISC (Reduced Instruction Set Computers) and understanding of the differences between the two.
Jarvis 7 embraced the RISC philosophy, the motive being to break down complex instructions into smaller ones and increasing the clock frequency to increase performance.

3) DEVELOPMENT OF INSTRUCTION SET ARCHITECTURE

Finalizing the various ISA aspects of JARVIS- 7

- Deciding the length of the instruction. JARVIS 7 processor has a fixed instruction length of 24 bits (lesser than the MIPS' 32 bits) dividing the instruction formats into three types.
- This step involved using the optimum number of general purpose registers, the length of opcode, and the number of instructions.
  Optimum number of GPRs=16. Bits required to address them=4
  Optimum number of instructions=18.
- Addressing Modes: Direct addressing, Indirect addressing, Immediate addressing

- Operand Size: JARVIS 7 has a varying opcode field as compared to MIPS' fixed opcode length allowing us to have a larger offset value thereby increasing the outreach while accessing memory.
- Operations: We have included basic instructions like add, subtract, branch if equal, branch if set, etc. which can be used to accomplish further complicated instructions.
- Conditional Code Implementation :
  Each instruction is conditionally executed depending upon the condition code bits and the content of the status register.

4) DATAPATH DESIGNING

- Designing of the datapath once the instruction set and formats were finalized.

5) INCORPORATION OF PIPELINING IN THE MICROPROCESSOR DESIGN

- Study of pipelining and various hazards involved in its implementation.
- Development of pipelined Datapath for JARVIS – 7.

6) DEVELOPMENT OF VHDL CODE FOR REAL TIME IMPLEMENTATION OF THE DESIGNED MICROPROCESSOR

- Development of codes for the various individual components.
- Integration of the individual components into a single microprocessor unit.
- Modifying the basic microprocessor to incorporate the pipelined design.

7) SURVEY AND COMPARISON

- Pitched the various processors present in the market today against each other.
- Also, compared the performance of JARVIS-7 with the processors available in the market.

PART I

ISA AND DATAPATH DESIGN

SYSTEM DESCRIPTION

DESIGN PHILOSOPHY

We began by exploring the benefits and drawbacks that the different architectures (with stored program concept) offer to us. We began a SWOT analysis, ruling out possibilities based on our design goals.

(i) STACK MACHINE :

Advantages of Stack Machine Instruction Sets

- Very Compact Object Code due to simple opcodes. The two operands are implied in each instruction thereby reducing the instruction length.
- Simple Compilers
- Simple Interpreters
- Minimal Processor State : A machine with an expression stack can get by with just two visible registers, the top-of-stack address and the next-instruction address. The minimal hardware implementation has few bits of flipflops or registers.

Performance Disadvantages of Stack Machines

- More Memory References
- Factoring Out Common Sub expressions Has High Cost :
- Rigid Code Order
- Hides a Faster Register Machine Inside
- More Instructions, Slower Interpreters

(ii) ACCUMULATOR BASED MACHINES

Performance advantage of Accumulator based machines

- One operand implied
- Faster than stack machines

Performance disadvantages of Accumulator Based Machines:

- Pipelining cannot be implemented.

- Provide no huge benefit over stack based machines

(iii) THAT BRINGS US TO LOAD STORE MACHINES WITH A REGISTER FILE.

- Has to take care of different types of instruction formats but gives more flexibility and ease to the programmer.
- Also, pipelining can be implemented.

INSTRUCTION SET ARCHITECTURE OF THE LOAD STORE MACHINE

(i) CISC (Complex Instruction Set Computers) :

- More flexibility,
- Complex implementation

(ii) RISC (Reduced Instruction Set Computers):

- All instructions of same length,
- Breaking down complex instructions into minimalist instructions and speeding up the clock to increase performance.
- Easier to implement.

Thus, after going over all the advantages and disadvantages of the various architectures available , using RISC as the foundation for JARVIS-7 microprocessor architecture was decided as the most logical choice.

INSTRUCTION SET ARCHITECTURE

- No. of registers : 16
- Length of registers : 24 bits
- Register classification :
    - Program Counter
    - Status register
    - 4 argument registers
    - 5 saved registerss
    - 7 temporary registers

INSTRUCTION FORMATS

R-Format (ARITH FORMAT)

C	F	RS1	RS2	Rd	Opcode
2 bits	2 bits	4 bits	4 bits	4 bits	8 bits

IMMEDIATE FORMAT

C	F	RS1	Rd	Offset	Opcode
2 bits	2 bits	4 bits	4 bits	8 bits	4 bits

BRANCH FORMAT

C	F	Offset	Opcode
2 bits	2 bits	18bits	2 bits

DESCRIPTION OF FIELDS

(i) F , The '*Format field' :* depicts the format of the instruction on which the opcode length is to be decided.

(ii) C, The *'Condition code field'* : depicts the condition code which decide whether the instruction need to be executed or not.

(iii) Rd : Destination Address/location

(iv) Rs : Source Address/location

(v) Offset: offset to address

(vi) Opcode: operation code

INSTRUCTION LENGTH

JARVIS-7 processor has a fixed instruction length of 24 bits (lesser than the MIPS' 32 bits) dividing the instruction formats into three types depending on opcodes:

- Arithmetic
  Add, sub, and, or, not, slt, seq

- Immediate
  Addi, subi, andi, ori, slti, seqi, lw, sw, beq

- Branch
  B, BL

Opcode Length : JARVIS VII has a varying opcode field(2,4 and 8 bits depending on the instruction) while MIPS has a fixed opcode length of 8 bits for every instruction.

Varying opcode field sizes have enabled to have a larger offset value thereby increasing the outreach while accessing memory.

INSTRUCTION SIZE

- JARVIS -7 has a reduced instruction size of 24 bits compared to MIPS 32 bits thereby enabling lesser instruction memory consumption.

- Further because of the varying opcode field size,we succeeded in increasing the offset field size to 18 bits (in case of Branch Format)thereby increasing the memory reach to a greater extent, which if had not been there would lead to a pretty smaller offset field size of just 12 bits

S NO.	INSTRUCTION	MNEMONIC	FORMAT	OPCODE
1	Add	add	AR	00000000
2	Sub	sub	AR	00000001
3	And	and	AR	00000100
4	Or	or	AR	00000101
5	Nor	nor	AR	00000111
6	Set if less than	slt	AR	00001100
7	Set if equal	Seq	AR	00001111
8	Branch	B	BR	00
9	Branch and Link	BL	BR	01
10	Add Immediate	addi	IM	0000
11	Sub Immediate	subi	IM	0100
12	And Immediate	andi	IM	0001
13	Or Immediate	ori	IM	0101
14	Nor Immediate	nori	IM	0111
15	Set if Less than Immediate	slti	IM	1100
16	Set if equal to Immediate	Seqi	IM	1111
17	Load Word	lw	IM	1000
18	Store Word	sw	IM	1001
19	Branch if equal	Beq	IM	1011

TABLE 1: INSTRUCTION SET ARCHITECTURE FOR JARVIS-7

AR-> Arithmetic (format bits=00)
BR-> Branch (format bits=01)
IM-> Immediate (format bits=10)

IMPORTANT FEATURE: CONDITIONAL EXECUTION OF EACH INSTRUCTION

- CC=01 => execute instruction if status register is set (i.e. if SR=00000000000000000000000001)
- CC=00 => execute instruction if status register is reset (i.e. if SR=00000000000000000000000000)
- CC=11 => always execute instruction

ADVANTAGES

(i) Decreases code footprint
(ii) Gives additional flexibility to assembly programmer
(iii) Reduces Hardware.

DECREASED CODE PRINT

Sample program to depict advantage of conditional execution of each instruction:

C CODE	MIPS CODE	JARVIS-7 CODE
w,a,b	slti $t0,$s0,5	slti $SR, $s3, 5
if(w<5)	beq $t0,$zero,L1	addis $s1,0
{ a=1; b=0;	addi $s1,0	addins $s1,1
}	addi $s2,1	addis $s2,1
Else	L1 : addi $s1,1	addins $s2,0
{ a=0; b=1;	addi $s2,0	
}		

For the given C Code, MIPS has 6 instructions while JARVIS 7 has 5 instructions thereby decreasing the code foot print.

	PASCAL	C
IF	29%	43%

TABLE 2: DYNAMIC OCCURRENCE OF CONDITIONAL EXECUTION

On an average, the JARVIS-7 ISA reduces one instruction per if-else block.
Assuming, average number of instructions per if-else block to be 6.

	PASCAL	C
IF	(1/6)*29% = 4.83%	(1/6)*43% = 7.16%

TABLE 3: SAVINGS AS COMPARED TO MIPS ISA(TOTAL NO. OF INSTRUCTIONS)

***The values depict the consequent reduction in code length.**

ADDED FLEXIBILITY

Using condition codes in the instructions gives the assembly programmer the convenience of writing the conditionally executable instructions in any order, once the flag is set by evaluating the condition.
//flexibility to assembler programmer

C CODE	MIPS CODE	JARVIS-7 CODE
	slti $t0,$s0,5	slti $SR, $s3, 5
if(w<5)	beq $t0,$zero,L1	addis $s1,0
{ a=1; b=0;	addi $s1,0	addins $s1,1
}	addi $s2,1	addis $s2,1
Else	L1 : addi $s1,1	addins $s2,0
{ a=0; b=1;	addi $s2,0	
}		

REDUCING HARDWARE

As a consequence of using condition codes in the instruction formats, we can implement unconditional jump using a branch instruction.

We have three branch instructions (branch if set, branch if not set, simple branch)

(i) CONDITIONAL BRANCHING:

It is implemented by breaking the conditional branch instruction into two parts:

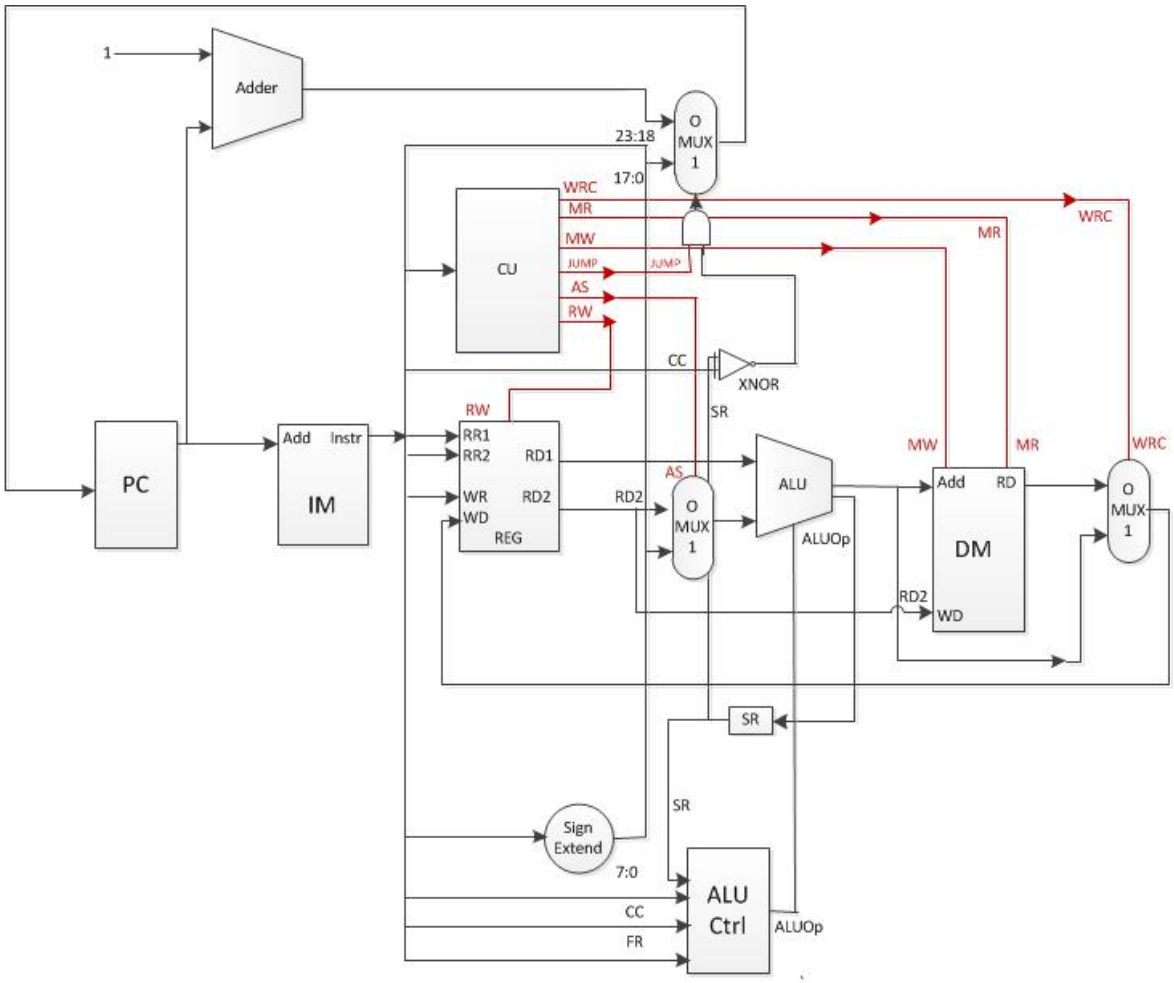
- A condition testing instruction that sets (or does not set) a register after condition evaluation.
- A branch if set (or not set) instruction with the specified address : the address here, being specified in (18/24) bits

(ii) UNCONDITIONAL BRANCHING

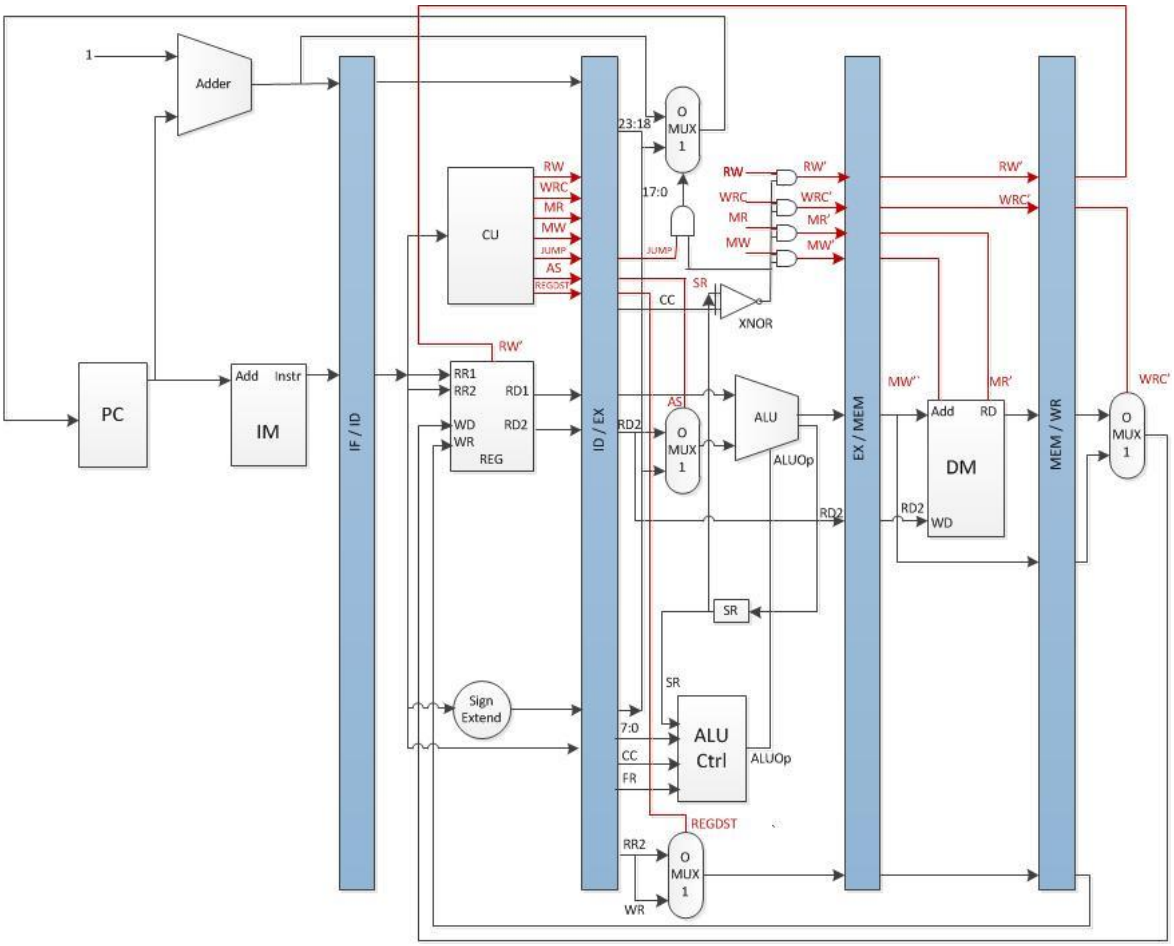
Uses simple branch instruction and specify the address (18/24) bits.
MIPS has an entirely different instruction format just for implementing unconditional jump (26 bit offset/32 bit)

- This improvisation reduces the additional hardware associated with the unconditional jump instruction (in MIPS) and utilizes the hardware for branching to implement the same.

DESIGNING THE DATAPATH



JARVIS 7 DATAPATH



JARVIS 7 PIPELINED DATAPATH

The figure depicts the data path of our self-designed microprocessor. The various components of the data path are:

1) Program counter: This register holds the address of the next instruction to be executed and supplies to the instruction memory. It is incremented after each clock cycle depending upon the type of instruction being executed.
2) Instruction memory: This unit receives the address of the instruction from the program counter and provides the instruction at the corresponding address for its execution. The capacity of IM in the processor is 128 x 24 bits.

3) Register file: This is basically the collection of registers which hold the values of data items or address of any location in the memory during the execution. The register file provides the facility to read and write the registers depending upon the type of instruction.
4) Control unit: This unit provides the function of providing control signals to all the functional units for proper execution of the instruction.
5) ALU: The arithmetic logic unit performs two operations. First, it is programmed to function as an adder to calculate the contents of the program counter. Second, it performs arithmetic and logical calculations for the R-type instructions and addition and subtraction for the MEM Type and BR Type instructions.
6) Data memory: This memory receives the address from the ALU and it provides the content of the location specified by the address. This memory can also be written by the contents of the selected register.
7) ALU control: This entity provides control signals for the ALU which informs the latter the kind of operation to be performed.

The flow through the datapath can be easily understood by the basic knowledge of datapath flow across any processor for example MIPS. Here we list out the features that starkly differentiates datapath of JARVIS-7 from other microprocessors.

1. As we are using word addressable memory, to go to the next location in memory we need to increment PC by just 1, provided there is no branch or jump instruction.
2. As mentioned earlier ISA of JARVIS-7 uses instruction format having variable opcodes. Thus, to determine the ALUop 4 inputs are sent to the ALU Ctrl – 7:0 bits , Condition Codes, Format bits, and the status in the status register.
3. Also, since the control unit in the decode stage doesn't know the state of the status bit which falls in the execution stage, it sends the controls solely on the basis of the type of instruction, irrespective of whether the condition code and status bit contents match or not.
   To allow for conditional execution of instructions, we have XNORed the status bit with the condition code. This gives a high output when the two are same, i.e. the condition for execution is met. Otherwise the output is low. This output is ANDED with controls from CU to generate the new set of controls which are then fed into different units and the next pipeline register.

An attempt was made to remove the multiplexer placed just before the register file in the MIPS data path design. Consequently, the instruction format was modified as:

C	F	Rd	Rs1	Rs2	opcode
23:22	21:20	19:16	15:12	11:8	7:0

The register destination has been modified to be just after the file format such that the destination of register is known beforehand and the presence of multiplexer can be avoided. But unfortunately, this design demanded the placement of a multiplexer before the source registers which would create further complications because source registers are read in parallel along with the functioning of the control unit. So this approach was dropped as it could have resulted in unnecessary delays.

Thus the finalized instruction format adopted was:

C	F	Rs1	Rs2	Rd	opcode
23:22	21:20	19:16	15:12	11:8	7:0

PART II

VHDL IMPLEMENTATION

DEVELOPMENT OF CODE FOR INDIVIDUAL COMPONENTS

The VHDL implementation of our datapath and instruction set architecture started with the coding of different components

Xilinx code was, therefore, developed for –

- Memory : data memory and instruction memory
- Control Unit
- ALU and ALU Control
- MUX
- Sign Extender
- Pipeline registers
- Register file

RTL SCHEMATIC AND TESTBENCH WAVEFORMS OF INDIVIDUAL COMPONENTS.

MEMORY

Memory Size of the JARVIS microprocessor is 128 X 24 bits

RTL Schematic

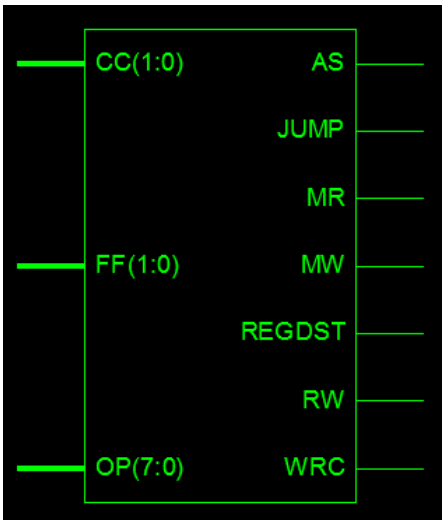
The description of various signals are as follows :

(i) address : Address to be accessed.
(ii) data : data to be written
(iii) clk : Memory clock
(iv) en : Enable
(v) rd : Read Enable
(vi) wr : Write enable

CONTROL UNIT DESIGN

Once the datapath design was complete, implementing the control unit was just a matter of setting a few control signals high or low based on the type of instruction (as decoded by the control unit)

These signals when applied to the datapath modules would synchronize their working according to the instruction received.



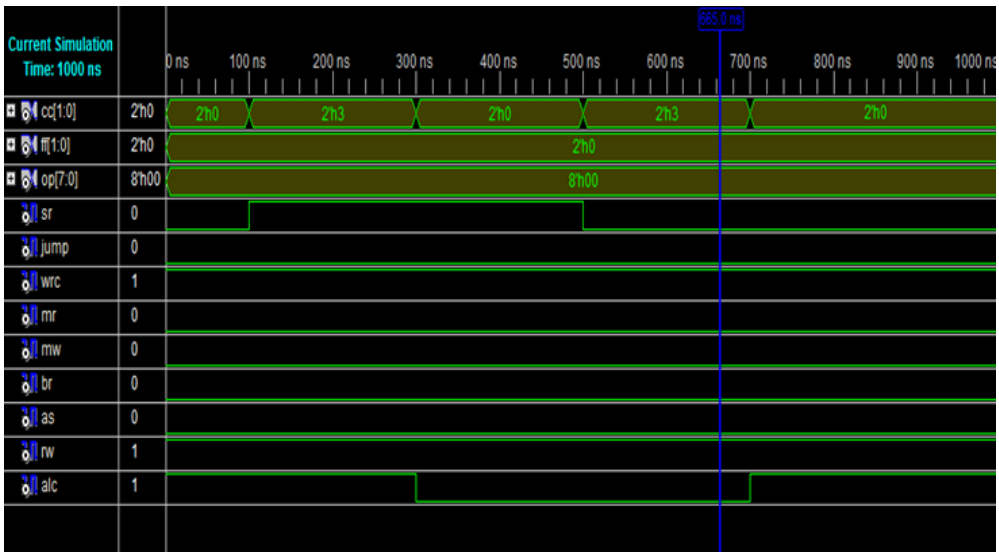
RTL Schematic

The description of various input and output signals are as follows :

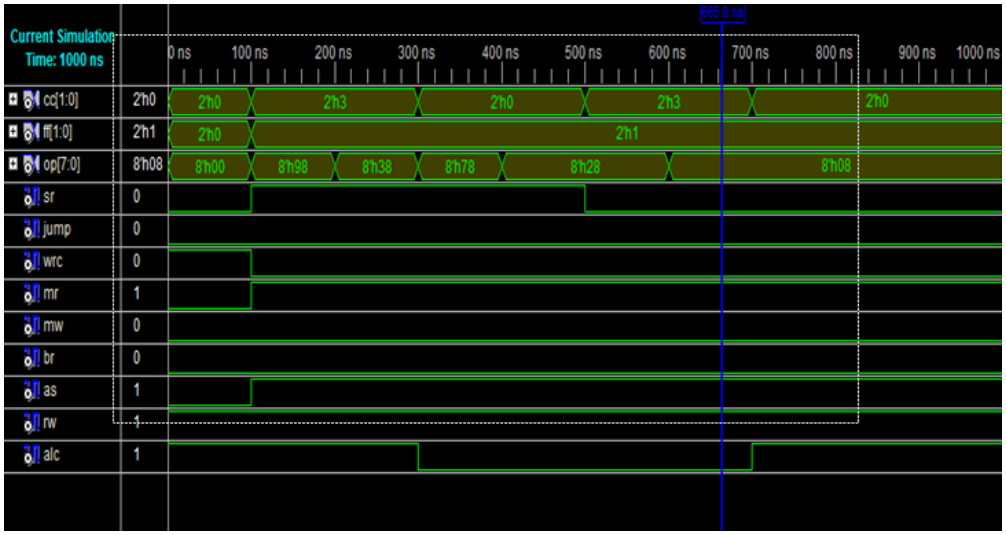
(i) CC : Condition code bits of the instruction
(ii) FF : Instruction format bits of the instruction
(iii) OP : Opcode bits of the instruction
(iv) SR : Status Register bit of the instruction
(v) AS : Bits sent to the MUX deciding the source data of the ALU
(vi) JUMP : Set when there is a JUMP instruction
(vii) MR : To enable Memory Read
(viii) MW : To enable Memory Write
(ix) RW : To enable Register Write
(x) WRC : To set whether the output of ALU is to written back to the register.
(xi) REGDST : Decides the register to be written on the basis of instruction format

SOME TESTBENCH WAVEFORMS :

TYPE I INSTRUCTION : ADD

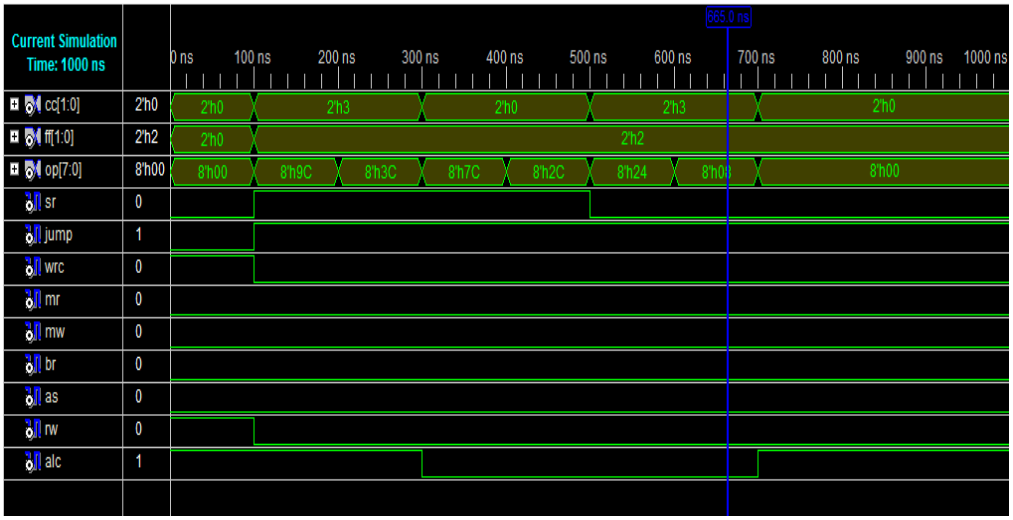


TYPE II INSTRUCTION : LW



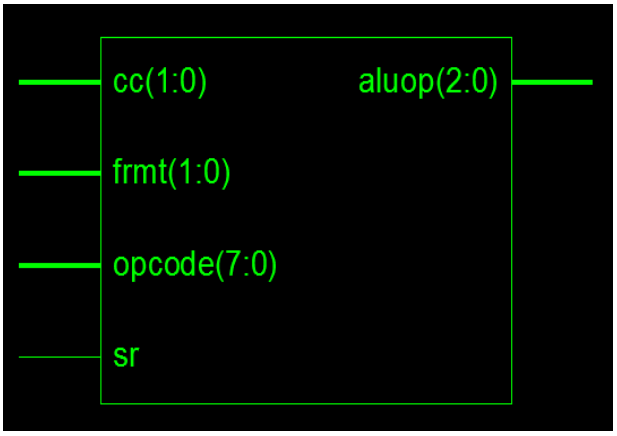
rw is set to enable register write of the data to be brought from memory.
mr is set to enable memory read.

TYPE III INSTRUCTION : JUMP



jump bit is set to 1 to load the PC with the destination address given in the instruction.

ALU CONTROL DESIGN



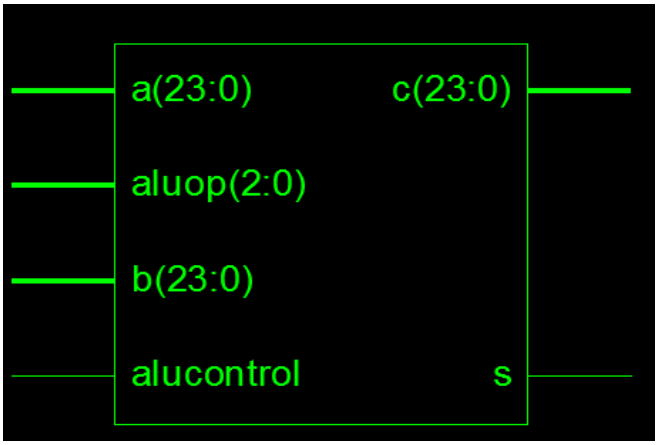
RTL Schematic

The description of various signals are as follows :

(i) cc : Condition code bits
(ii) frm : format bits
(iii) opcode : Operation Code
(iv) sr ; Status bit
(v) aluop : To drive ALU
(vi) wr : Write enable

This entity scans the condition code provided in the instruction and compares it with the status bit to decide whether the arithmetic logic unit needs to be involved in the execution or it will remain idle for one clock cycle. Further, depending upon the format of the instruction being executed, it assigns to the arithmetic logic unit a unique code which governs the operation of the computing entity.

ALU



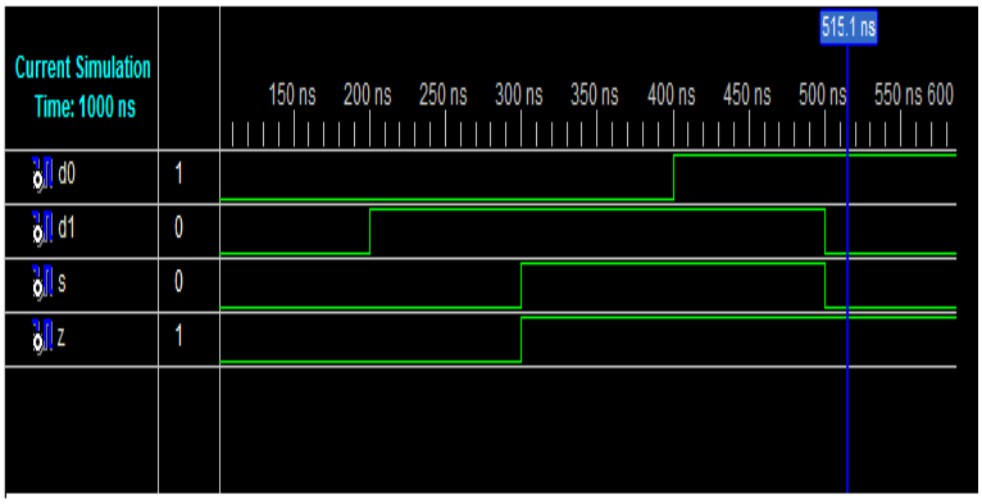
RTL Schematic

The description of various signals are as follows :

(i) a : Source data
(ii) b : Source data
(iii) aluop : Bits from the ALU Control

The arithmetic logic unit receives an exclusive code from the ALU CONTROL. This code administers the operation of the entity. Provided the operands as inputs to the ALU, it performs an arithmetic or logical operation depending upon the code and provides the output. Besides computation, ALU also sets the status bit as per as the requirements of the instruction and the output produced

2x1 MUX



DATA INPUT

D0 - 1

D1 - 0

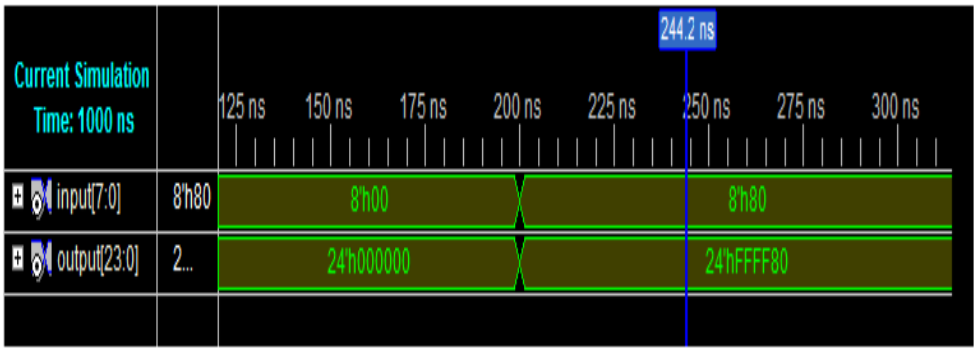
S - 0

DATA OUTPUT

Z – 1

EXPLANATION : Since S i.e. the status is 0, the output Z is same as the D0

SIGN EXTENDER



DATA INPUT

Input : 10000000

DATA OUTPUT

Output : 111111111111111110000000

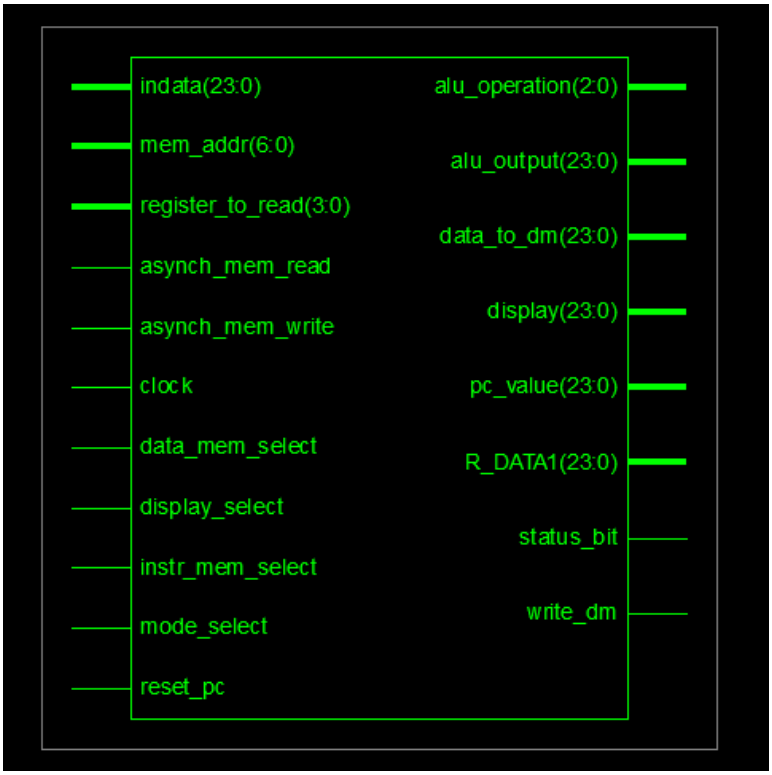
EXPLANATION : The input data is multiplexed.

VHDL IMPLEMENTATION OF THE ENTIRE PROCESSOR

Once the working of each individual component was tested and verified , the next step involved synchronization of each component so that the entire processor worked as a single unit – JARVIS.

The final implemented and synchronized code for JARVIS is listed in the appendix. Here, we present the RTL Schematic and Timing Diagrams generated by testing some sample programs on the JARVIS 7.

TOP MODULE : THE USER-JARVIS INTERFACE



RTL Schematic

INPUTS

(i) Indata(23:0)
the 24 bit data you want to write to the instruction or data memory asynchronously.

(ii) Mem_addr(6:0)
The 7 bit address for the memory location that you wish to manipulate.

(iii) Register_to_read(3:0)

The 4 bit address of the register you want to read anytime.

(iv) Asynch_mem_read

When you put this high, you can read the selected memory unit (instruction or data memory).

(v) Asynch_mem_write

When you put this high, you can write the selected memory unit (instruction or data memory).

(vi) Clock

One single external clock for synchronization.

(vii) Data_mem_select

Put this high if you want to operate on data memory.

(viii) Display_select

When high, it displays the data in the given address location of the **data memory** .When low, it displays the contents of the given memory location of the **instruction memory.**

(ix) Instr_mem_select

Make this high to be able to operate on instruction memory.

(x) Mode_select

The user mode:

When high, you (the user) can perform asynchronous operations on the state elements of the microprocessor.

The JARVIS mode:

When low, JARVIS takes over the control and executes what you have instructed it to.

(xi) Reset_PC

To reset the Program Counter contents (to all zeros).

OUTPUTS

(i) Alu_operation(2:0)*

The 3 bit aluop that is generated by ALU_Control and fed to the ALU to instruct it what operation to perform.

(ii) Alu_output(23:0)*

The output of ALU.

(iii) Data_to_dm(23:0)*

This is the content of the bus that serves as the data input to data memory.

(iv) Display(23:0)

This bus carries the data read from instruction or data memory.

(v) Pc_value(23:0)

The contents of Program counter.

(vi) R_DATA1(23:0)

The contents of the register that user wants to read.

(vii) Status_bit

The state of the status (set when high and reset when low).

(viii) Write_dm*

The state of the write enable of data memory.

Note: *marked outputs are solely for the purpose of keeping track of what the JARVIS does when in command. These checks help in error identification and debugging.

EXPLANATION

The top module was created in order to allow the user to

(i) Enter the instructions to be executed in the instruction memory **before the execution begins.**

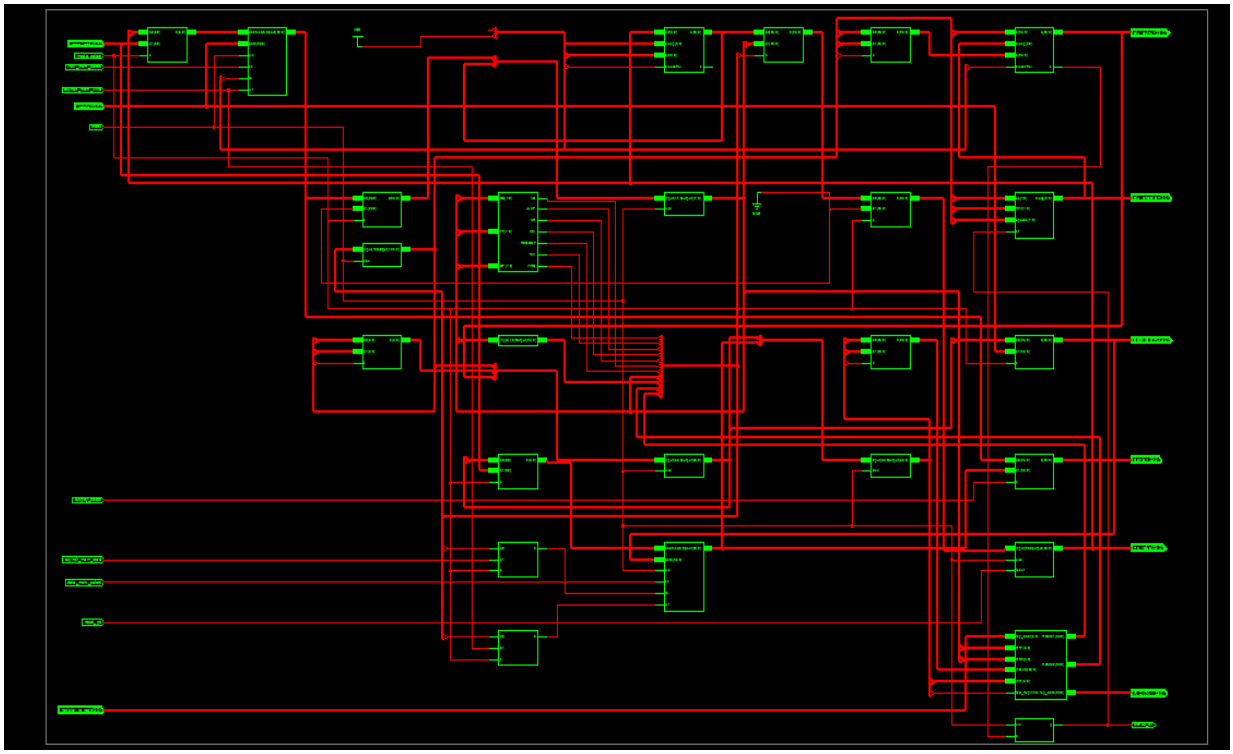
(ii) Keep an eye over the changing values of various signals **during the execution of the instructions by the processor.**

(iii) Read the contents of data memory and registers to match the output against the expected one, **after the program execution has been completed.**

User begins by setting **mode_select** high(the user mode). He can then select the instruction memory and feed in the instructions at the desired memory locations using the various appropriate inputs. When the program is ready to be executed, user can set the **mode_select** low (the JARVIS mode), wherefrom, the microprocessor starts executing the instructions.

User can track the progress of the execution through various outputs and finally verify whether the correct result has been produced by the processor.

RTL SCHEMATIC: INSIDE THE TOP MODULE



DESCRIPTION:

The schematic is an inside view of the TOP MODULE. It consists of several individual functional units (the green boxes) interconnected by signals (the red lines).
Following are the various functional units inside the processor:

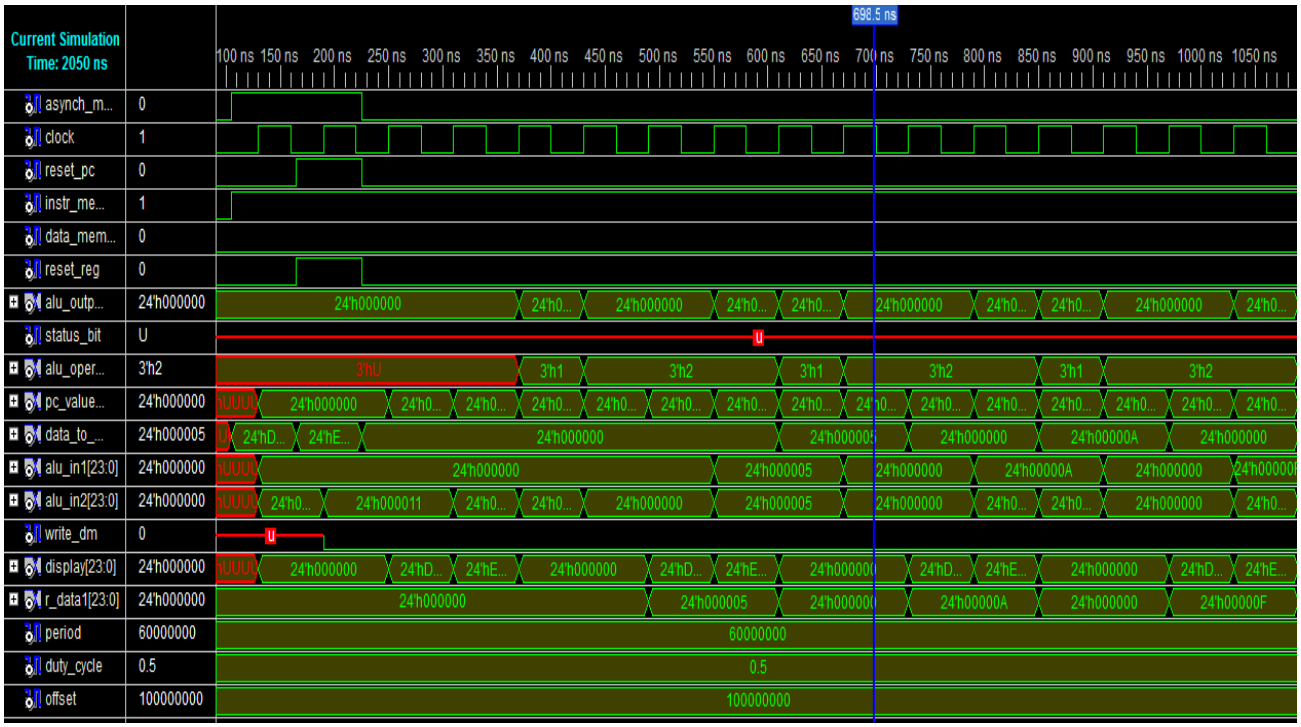
1. Control unit
2. Data memory
3. Instruction memory
4. Register file
5. ALU
6. ALU control
7. Sign extender
8. Multiplexors
9. Pipeline registers
10. Status bit latch
11. Program counter register

Each of these units was coded behaviorally. The top module is nothing but a structural port mapping of these units to come up with a working data path of the microprocessor.

SAMPLE PROGRAMS

1) A SIMPLE ADDITION LOOP

```
addi  R0,R0,5     D00050   // R0 <- R0 + 5 (R0 initially has 0)
b     0x0000      E00000   //  Branch to address 0 i.e. to the addi instruction
```



TEST BENCH WAVEFORM

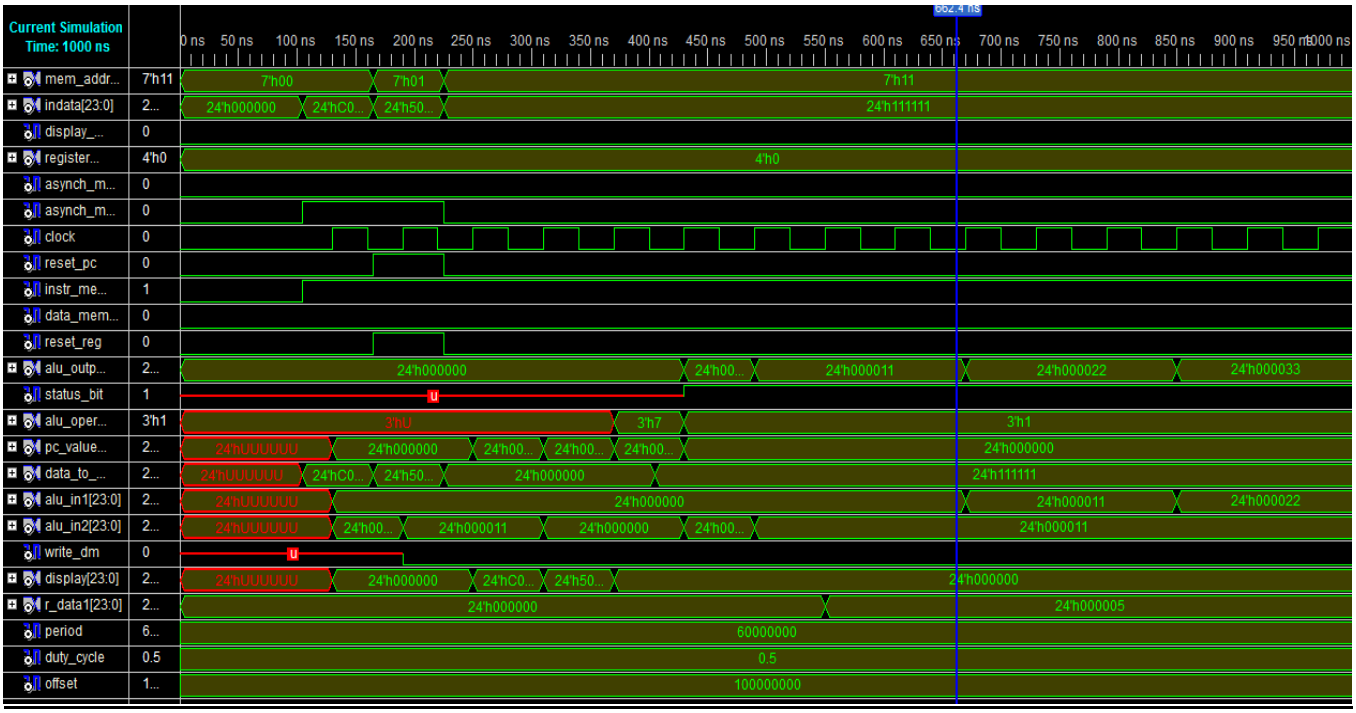
EXPLANATION:

After 5 clock cycles from the beginning of execution, the contents of the register R0 change from 0 to 5 (see the r_data output above). 5 clock cycles because we have 5 pipeline stages. The program counter increments every clock cycle. After 4 clock cycles from the beginning of execution, the program counter contents change from 3 to 0(the B instruction makes a jump to 0 for looping). The process repeats in a loop to add 5 to the register R0. The contents of R0 can be seen increasing from 5 to A and then to F.

2) CONDITIONAL ADDITION

seq R0,R1 C0100F // set the status bit if (R0) = (R1)

addis R0,R0,5 500050 // add 5 to R0 if status bit is set



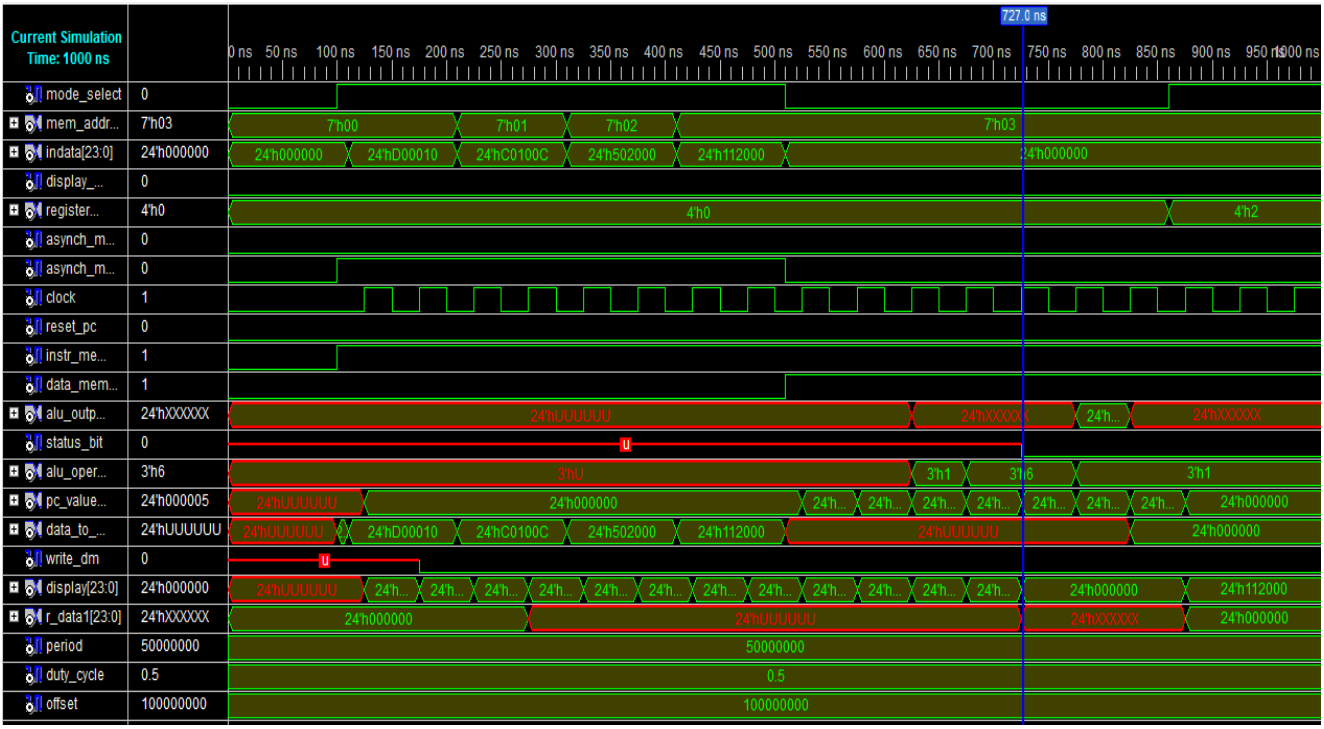
TEST BENCH WAVEFORM

EXPLANATION:

Since the contents of register R0 and R1 are initially 0 (and thus equal), the status bit will be set by the seq instruction. This can be seen as a transition of the status_bit output from unknown (U) to high in the waveform above. After two clock cycles the value of register R0 changes from 000000 to 000005 (r_data above). Thus the expected output has been obtained.

3) MINIMUM OF TWO NUMBERS

```
addi R1,R1,1          // Loading 1 into register R1 ; R0 has by default 0 value
slti R1,R0            // Comparing R1 with t0
addis  R2,R1,0        // If R1 is smaller,load R1 in the register R2
addins R2,R0,0        //If R0 is smaller,load R0 in the register R2
```



TESTBENCH WAVEFORM

EXPLANATION:

The minimum value i.e. 0 is stored as the result in the register R2 at the end of 850ns.Thisis indeed the minimum of the two values 0 and 1.

PART III

SURVEY AND COMPARISON

COMPARISON OF JARVIS – 7 WITH OTHER COMMERCIALLY AVAILABLE PROCESSORS

	Alpha	MIPS I	PA-RISC 1.1	PowerPC	SPARC V8	JARVIS VII
Date announced	1992	1986	1986	1993	1987	2012
Instruction size (bits)	32	32	32	32	32	24
Condition Code	N	N	N	N	N	Y
Frequency		100 Mhz				70 Mhz
Address space (size, model)	64 bits, flat	32 bits, flat	48 bits, segmented	32 bits, flat	32 bits, flat	24
Data alignment	Aligned	Aligned	Aligned	Unaligned	Aligned	Aligned
Data addressing modes	1	1	5	4	2	2
Protection	Page	Page	Page	Page	Page	---
Minimum page size	8 KB	4 KB	4 KB	4 KB	8 KB	---
I/O	Memory mapped	Memory mapped	Memory mapped	Memory mapped	Memory mapped	---
Integer registers (number, model, size)	31 GPR x 64 bits	31 GPR x 32 bits	31 GPR x 32 bits	32 GPR x 32 bits	31 GPR x 32 bits	16 GPR x 24 BITS

SUMMARY OF DATA ADDRESSING MODES SUPPORTED BY THE DESKTOP ARCHITECTURES.

PA-RISC also has short address versions of the offset addressing modes. MIPS V has indexed addressing for floating-point loads and stores.

Addressing mode	Alpha	MIPS V	PA-RISC 2.0	PowerPC	SPARC V9	JARVIS VII
Register + offset (displacement or based)	Y	Y	Y	Y	Y	Y
Register + register (indexed)	N	Y(FP)	Y(Loads)	Y	Y	N
Register + scaled register (scaled)	N	N	Y	N	N	N

SUMMARY OF CONSTANT EXTENSION FOR DESKTOP RISCS

The constants in the jump and call instructions of MIPS are not sign-extended since they only replace the lower 28 bits of the PC, leaving the upper 4 bits unchanged. PA-RISC has no logical immediate instructions.

Format: instruction category	Alpha	MIPS V	PA-RISC 2.0	PowerPC	SPARC V9	JARVIS VII
Branch: all	Sign	Sign	Sign	Sign	Sign	--
Jump/call: all	Sign	---	Sign	Sign	Sign	--
Register-immediate: data transfer	Sign	Sign	Sign	Sign	Sign	Sign
Register-immediate: arithmetic	Zero	Sign	Sign	Sign	Sign	Sign
Register-immediate: logical	Zero	Zero	---	Zero	Sign	Sign

COMPARISON OF THE INSTRUCTION SET ARCHITECTURE USED IN VARIOUS PROCESSORS

1) **R-Format (ARITH FORMAT)**

JARVIS VII

C	F	Rd	RS1	RS2	opcode
2 bits	2 bits	4 bits	4 bits	4 bits	8 bits

OTHER DESKTOP RISC ARCHITECTURES

	31	25	20	15	4	0
ALPHA	op	Rs1	Rs2	Op		Rds

MIPS	op	Rs1	Rs2	Rd	consts	Op

POWER-PC	op	Rd	Rs1	Rs2	shamt

PA-RISC	op	Rs1	Rs2	Rd	funct

SPARK	op	Rd	Op	Rs1	0	Op	Rs2

2) IMMEDIATE FORMAT

JARVIS VII

C	F	Rd	RS1	offset	Opcode
2 bit	2 bits	4 bits	4 bits	8 bits	4 bits

OTHER DESKTOP RISC ARCHITECTURES

ALPHA	op	Rd	Rs1	const.

MIPS	op	Rs1	Rd	const.

POWER-PC	op	Rd	Rs1	const.

PA-RISC	op	Rs2	Rd	const.

SPARK	op	Rd	Op	Rs1	1	Const

3) BRANCH FORMAT

JARVIS VII

C	F	Offset	Opcode
2 bits	2 bits	18bits	2 bits

CONCLUSION

This report introduced the Jarvis 7 microprocessor, blueprints of its design and details of its working. It was an attempt to chalk out the entire process of “How to design a microprocessor”. We hope that this report provides a sufficient startup material to anyone who refers to it or anyone interested in building their own microprocessor.

Jarvis 7 has increased our understanding of the field computer architecture substantially. More so, it has taught us work ethics and teamwork. Besides giving us hands on experience, it has opened up new avenues of further research.

SCOPE FOR FURTHER WORK

We have outlined the strengths and unique features of Jarvis 7 in the report. This project can be extended further to improve upon its features.

- Increasing the number of pipeline stages
- Functionality of combining registers to allow floating point operation
- Superscaling the microprocessor
- Incorporating branch prediction techniques
- Improving upon the fetch and write mechanism for instruction and data cache

BIBLIOGRAPHY

1.Computer Organization and Design by David Patterson and John Hennessey

2.Computer Organization and Architecture by William Stallings

3.Structured Computer Organization by Andrew Tannenbaum

4.www.wikipedia.org

5.www.MIPS.com

6.www.webopedia.com/TERM/P/pipelining.html

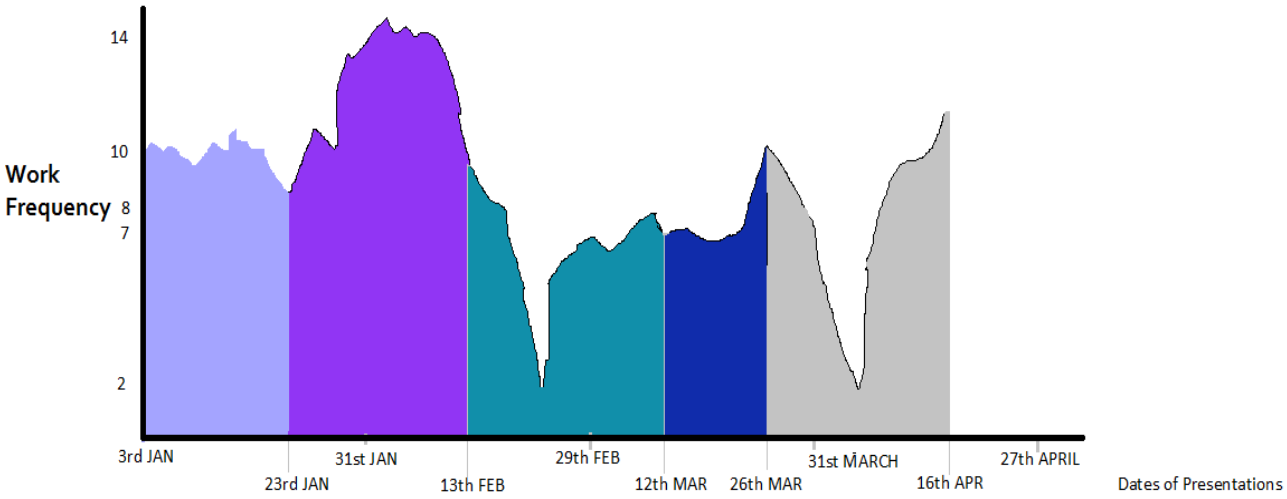
7.http://en.wikibooks.org/wiki/Microprocessor_Design/Instruction_Set_Architectures

8.http://webster.cs.ucr.edu/AoA/Windows/HTML/ISA.html

9.www.xilinx.com

10.http://office.microsoft.com/en-in/visio/

TIMELINE OF OUR WORK



Y axis denotes the combined work frequency of all the members of the group in hours/day

X axis shows the dates on which we displayed our progress as presentations

- From **3rd Jan** to **23rd Jan**, we studied in detail various concepts related to developing and using microprocessors like pipelining, ISA, datapath design , superscalar features , etc.
- **23rd Jan** to **13th Feb** was a very busy time during which we designed our own ISA, datapath and other components which were required in JARVIS 7. This required a lot of time and sincere efforts.
- Then from **13th Feb** to **12th March**, we coded individual units required for JARVIS 7 like ALU, ALU Control, Data Memory, Sign Extender etc. on XILINX.
- The time from **12th March** to **26th March** was utilized in synchronizing all the individual modules and developing a top module. This was the final stage of JARVIS 7 development.
- The period after **26th March** was spent on a survey of various commercially available microprocessors and comparing them with Jarvis 7 in various fields such as processor performance, code lengths, power consumption and hardware required for implementation.

The two sharp minimas on our timeline graph correspond to the Mid - Term Examinations. Yet the brighter aspect is that the work never stopped and JARVIS 7 was on the floor in time.