CSE 590: Computer Architecture

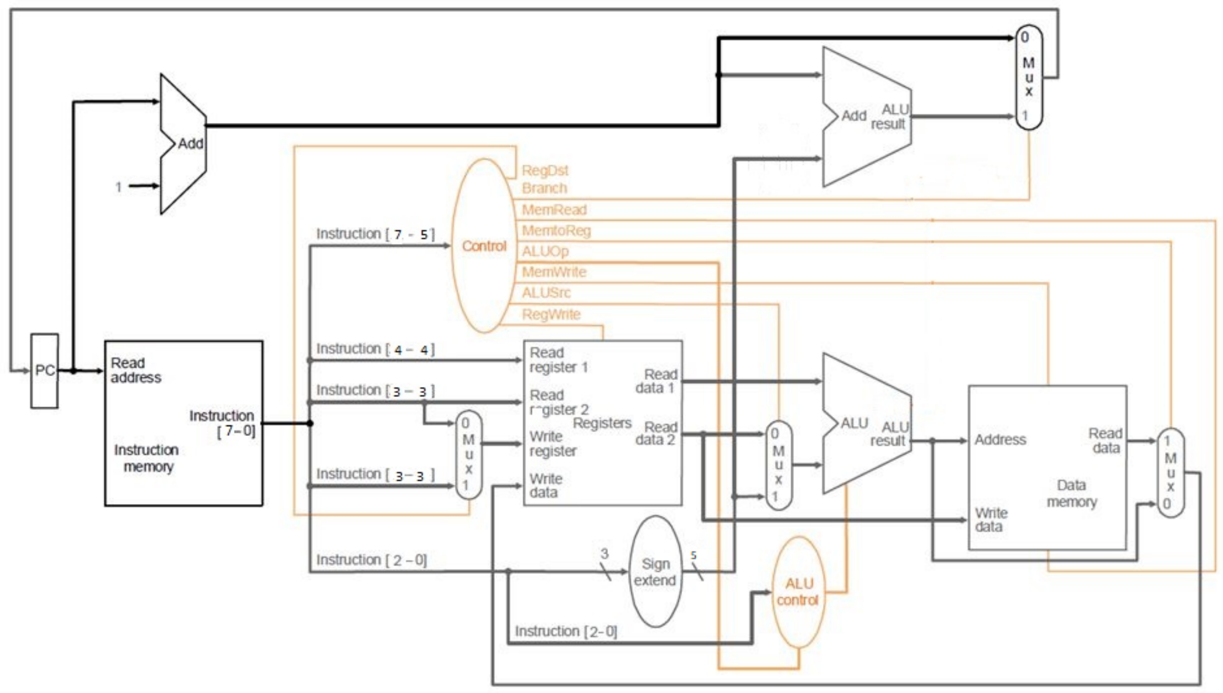
Project 1 Report

Team Members:
CSE 590

Sri Abinaya Gunasekaran | 50292993
Karthik Vikram Kuduva Gopal Kabirdoss | 50290281

8 Bits MIPS Processor Block Diagram

Block Diagram of MIPS processor



Control Unit :

- The control unit is responsible for setting all the control signals so that each instruction is executed properly.
- The control unit's input is the 3-bit instruction word.
- The signals are generated from the instruction opcode.
- It receives the opcode from the instruction memory and sends it signals to the corresponding unit based on the opcode.
- Below given table, shows the signals for all the opcodes and operation.

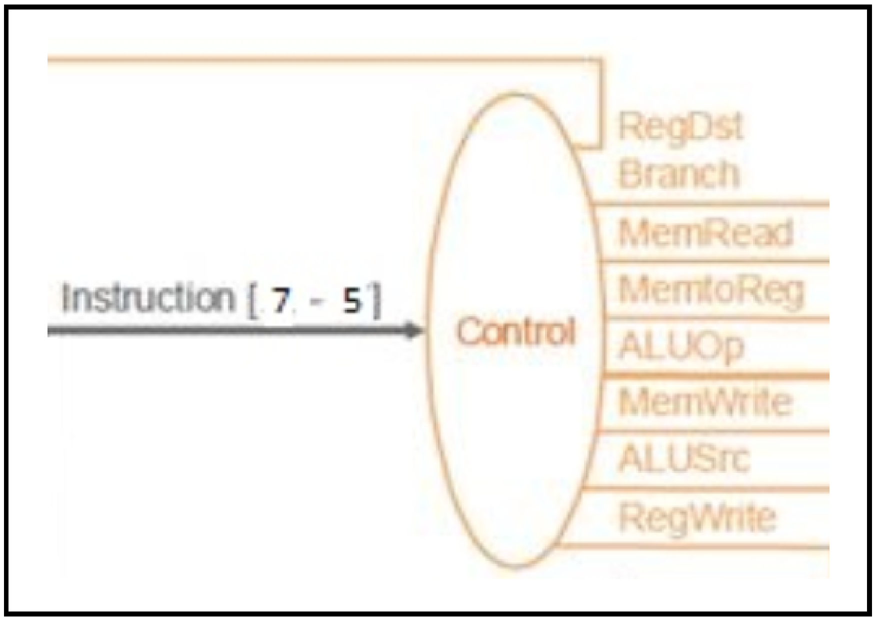


Fig: Block Diagram of the Instruction Memory Unit

Opcode	Operation	RegDst	Jump	RegWrite	ALUSrc	ALUOp	MemWrite	MemRead	MemToReg
000	**lw**	0	0	1	1	1	0	1	1
001	**sw**	0	0	0	1	1	1	0	0
010	**add**	1	0	1	0	0	0	0	0
011	**addi**	1	0	1	1	1	0	0	0
100	**sub**	1	0	1	0	0	0	0	0
101	**jmp**	0	1	0	0	0	0	0	0

RegDst - It sends the signal to the MUX connected between the Register and Instruction Memory.
Jump - It sends a signal to the MUX connected to the Program Counter. If the signal is ON, the MUX sends the address specified next to the jump instruction.If the signal is OFF, the MUX sends the next address to the program counter. It is ON only in case of the jump instruction.
RegWrite - It sends a signal to the Register. If the signal is ON, the register will write the content in the register after computation. If the signal is OFF, the register will not write the content in the register after computation.
ALUSrc - It sends the signal to the MUX connected between Register/Sign Extend and ALU. If the signal is ON, It sends the result from the sign extend. If the signal is OFF, It sends readdata2 from registers.
ALUOp - In general, MIPS processor, it sends bit input but for this processor, we send '0' for R-type instruction and it sends the functionality input to the ALU. it sends '1' for I type instruction to send default signal of addition.
MemWrite - It sends the signal to the Data Memory. If the signal is ON, the data memory will write the content in the data memory. If the signal is OFF, the data memory does not write anything.
MemRead - It sends the signal to the Data Memory. If the signal is ON, the data memory will read the content in the data memory. If the signal is OFF, the data memory does not read anything.
MemToReg - It sends the signal to the MUX connected between Data Memory and Register. If the signal is ON, it sends data from the Data memory to the Register. If the signal is OFF, it sends data from the ALU output to the Registers.

Instruction Memory

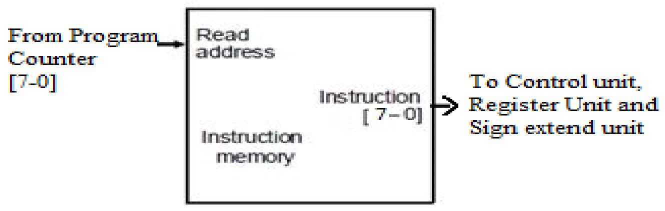

From Program Counter [7-0]
Read address
Instruction [7–0]
Instruction memory
To Control unit, Register Unit and Sign extend unit

Fig: Block Diagram of the Instruction Memory Unit

- The instruction memory component stores the bits of the instruction in the form of a continuous array which stores 8-bit data. Arrays can be indexed in Verilog like lists in python.
- The instruction memory takes in 8-bit address from the Program Counter and fetches the instruction corresponding to the address. The address stored in the 8-bit format is given to the corresponding components later.
- The instructions were formed using the guidelines mentioned in the project description.
- The following table gives the opcode that has to be used for different instructions.

- The **R- Type instructions** were formed using the format given below

Opcode			**Rt/Rd**	**Rs**	**Unused**		
7	6	5	4	3	2	1	0

Where,
Opcode – Specifies the 3 bits required for indicating whether it is the add or sub instruction.
Rt/Rd – Specifies whether the source and the target is the register R0 or R1
Rs- Specifies whether the second operand is register R0 or R1.

- The **I- Type instructions** were formed using the format given below

Opcode			**Rd**	**Rs**	**Immediate**		
7	6	5	4	3	2	1	0

Where,
Opcode – Specifies the 3 bits required for indicating whether it addi , lw or the sw instruction.
Rt – Specifies whether the first operand is the register R0 or R1
Rs - Specifies whether the second operand is register R0 or R1
Immediate – Specifies the immediate data in the case of addi instruction or the offset bits in case of the lw/sw instructions.

- The **J- Type instructions** were formed using the format given below

Opcode			**Immediate**				
7	6	5	4	3	2	1	0

Where,
Opcode – Specifies the 3 bits required for indicating it is the Jump instruction
Immediate – Specifies the address of the location where the program counter has to make a Jump to.

Program Counter

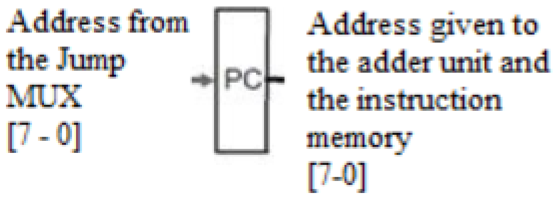

Address from the Jump MUX [7 - 0]
PC
Address given to the adder unit and the instruction memory [7-0]

Fig: Block Diagram of the Program Counter

- The Program counter in our 8 bit MIPS processor does nothing but gets the input from the jump MUX and gives it to the adder unit and the instruction memory unit.
- The Program counter unit simply serves to store the address of the current instruction getting executed by the MIPS processor.

ALU Unit

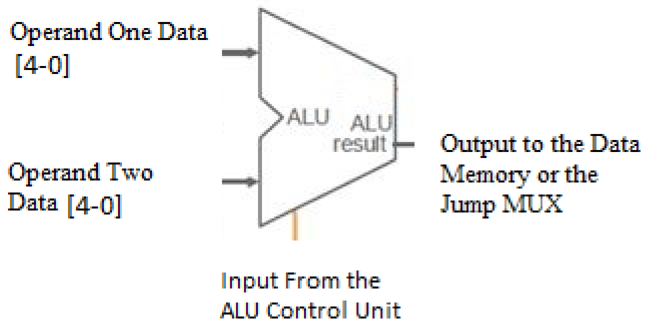

Operand One Data [4-0]
Operand Two Data [4-0]
ALU
ALU result
Output to the Data Memory or the Jump MUX
Input From the ALU Control Unit

Fig. Block Diagram of the ALU Unit

- The ALU unit serves to do two kinds of operations.
- The ALU can perform either bitwise addition or bitwise subtraction on the two 5 bit operands involved.
- The ALU unit decides upon whether it has to addition or subtraction operation based on the input it receives from the ALUSel input from the control unit
- The ALU unit gets one of its operands directly from the read register's content and the other from the ALU mux which decides from either the data from the second read register or the output of the sign extend unit which comes directly from the immediate part of the instruction.
- The ALU operates on a variety of data. During the execution of the ADD and SUB instruction, it adds or subtracts data from both the registers.
- During the execution of the ADDI instruction, it adds the data from one register and the sign extended data from the other instruction.
- During the execution of the LoadWord and StoreWord instructions it serves to add the data from one of the two registers and the offset associated with it.
- The result from the ALU is given either to the data memory unit in case of an address or the jump m,ux in case of the jump instruction exeuction.
- In the case of the ADD or SUB instructions the result is given to be written to one of the registers.

Sign Extend Unit

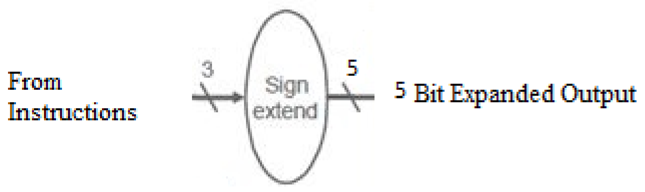
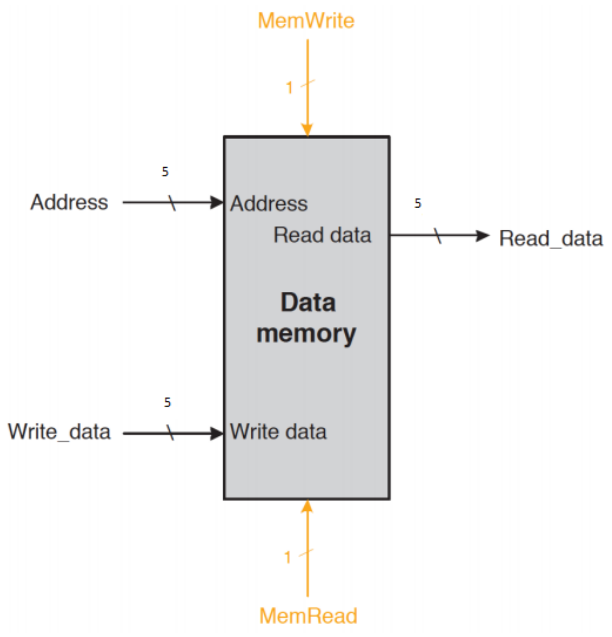


Fig. Block Diagram of Sign Extend Unit

- The sign extend unit gets a 3-bit input directly from the instruction memory and appends 2 0's in front of the 3 bits to convert it to a 5 bit address. Since the 0's are added to the MSB we can only give offsets up to 8 positions.

Data Memory

- Data memory has an 8-bit memory address.
- It can store data from 0 to 256.
- It initializes itself as 0 every time when MIPS processor starts and it replaces when it is written into the memory.
- When MemRead is 1, it reads the 8-bit address and goes to the particular address and reads the data and sends it output using Readdata.
- When MemWrite is 1, it reads the 8-bit address and gets the data from the ALU output and writes the data in the particular 8-bit address memory.



Multiplexer

- With two input signals and one output signal, the device is referred to as a 2-to-1 multiplexer.
- There is also a select signal to decide which input should be given as an output.
- In our processor, we have used 4 MUXes .Below given are the select signals used for the different MUXes.

  ALUsrc
  MemToReg
  Jump
  RegToDest

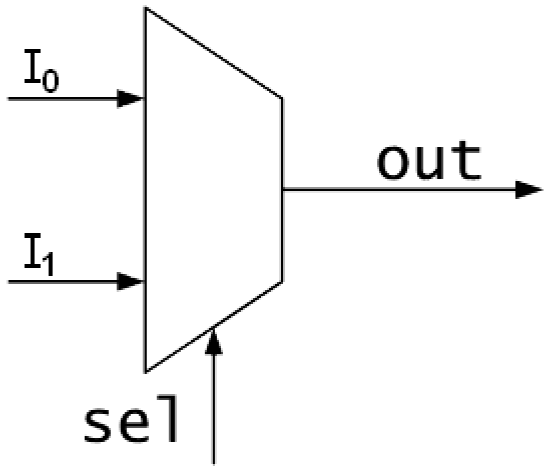


Fig. Block Diagram of 2 to 1 MUX

RegisterFile

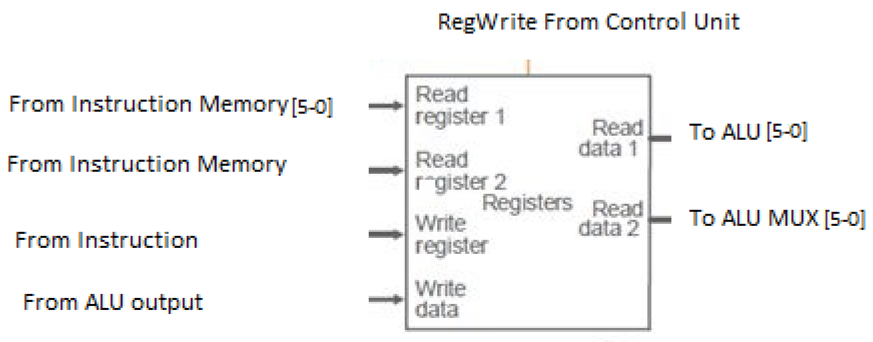
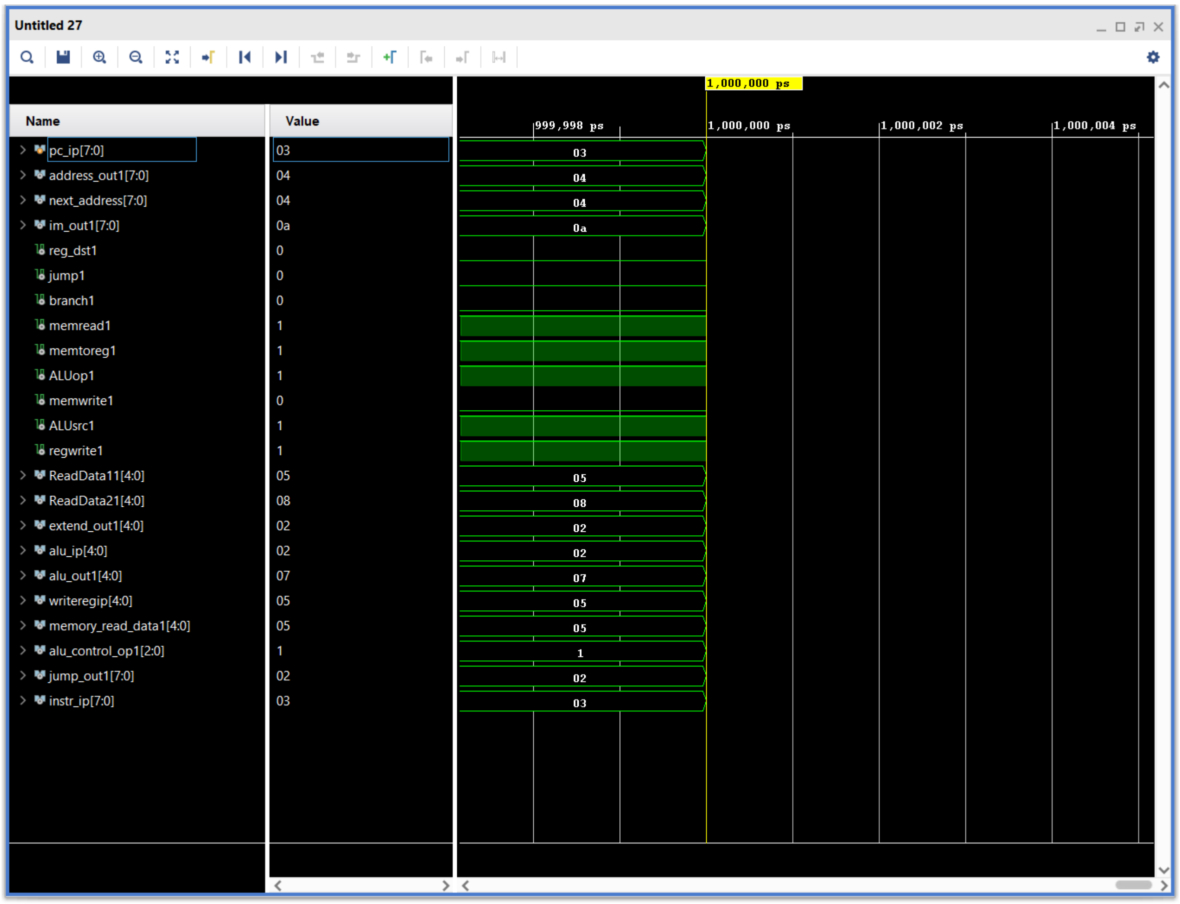


Fig. Block Diagram of Register Bank

Simulation Results of Individual Instructions

1- Simulation of Load Word Instruction

rs – R0 – 01000 - 8
rt – R1 – 00101 – 5
immediate – 00010 - 2
Instruction - 8'b00001010
We have stored in memory address 7 – 5'b00101
lw R0 2(R1)



Name	Value
pc_ip[7:0]	03
address_out1[7:0]	04
next_address[7:0]	04
im_out1[7:0]	0a
reg_dst1	0
jump1	0
branch1	0
memread1	1
memtoreg1	1
ALUop1	1
memwrite1	0
ALUsrc1	1
regwrite1	1
ReadData11[4:0]	05
ReadData21[4:0]	08
extend_out1[4:0]	02
alu_ip[4:0]	02
alu_out1[4:0]	07
writeregip[4:0]	05
memory_read_data1[4:0]	05
alu_control_op1[2:0]	1
jump_out1[7:0]	02
instr_ip[7:0]	03

Fig. Simulation Results of LoadWord instruction

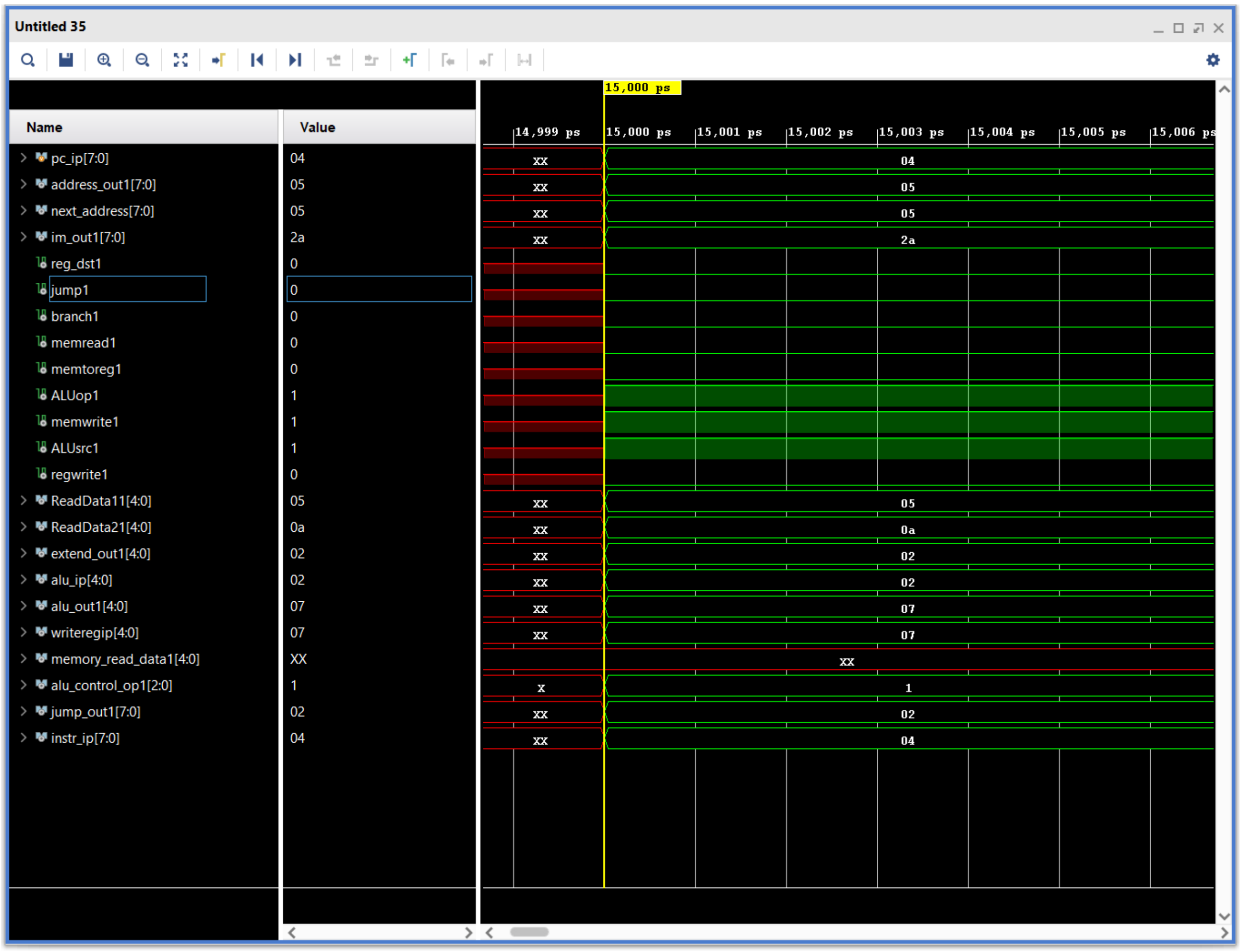
- Our load instruction is lw R0 2(R1)
- The R1 register contains the data 00101 (5).
- The ALU adds the offset with the contents of R1 to form the correct address to look for.

- The data in the memory location 7 is fetched and is loaded it into the register R0 as shown above.
- The memory location 7 contains the number 00101 (5) in it.

2 - Simulation of the Store Word instruction

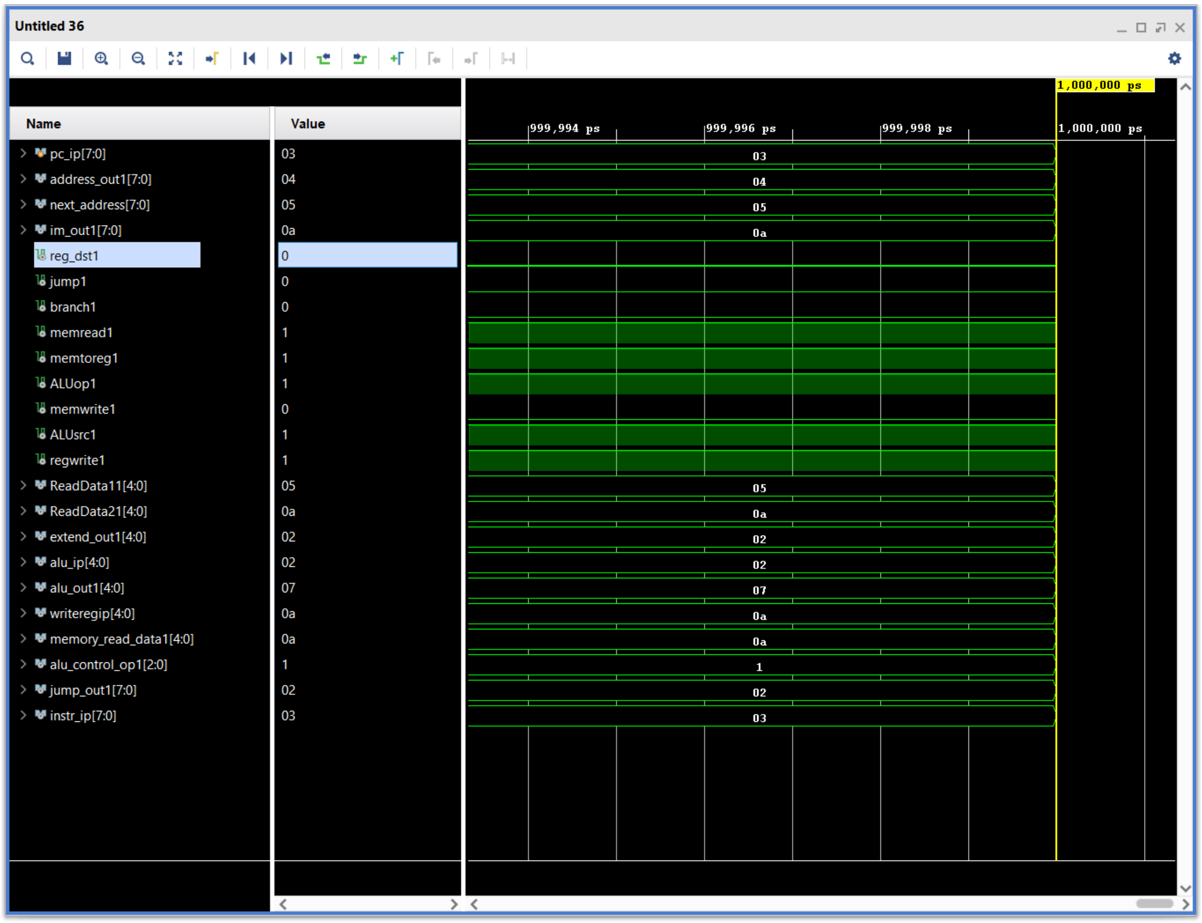
rs – R0 – 01000 - 8
rt – R1 – 00101 – 5
immediate – 00010 - 2
Instruction - 8'b00101010
SW R0 2(R1)

- The store word instruction being executed is sw R0 2(R1)
- The contents of the register , 5 (R1) is read and is added with the offset from the immediate bits of the instructions 2.
- The ALU adds the offset to form the correct address to write at the address formed in the data memory.
- The memory location 7 is written with the content 8 of the register R0



Name	Value
pc_ip[7:0]	04
address_out1[7:0]	05
next_address[7:0]	05
im_out1[7:0]	2a
reg_dst1	0
jump1	0
branch1	0
memread1	0
memtoreg1	0
ALUop1	1
memwrite1	1
ALUsrc1	1
regwrite1	0
ReadData11[4:0]	05
ReadData21[4:0]	0a
extend_out1[4:0]	02
alu_ip[4:0]	02
alu_out1[4:0]	07
writeregip[4:0]	07
memory_read_data1[4:0]	XX
alu_control_op1[2:0]	1
jump_out1[7:0]	02
instr_ip[7:0]	04

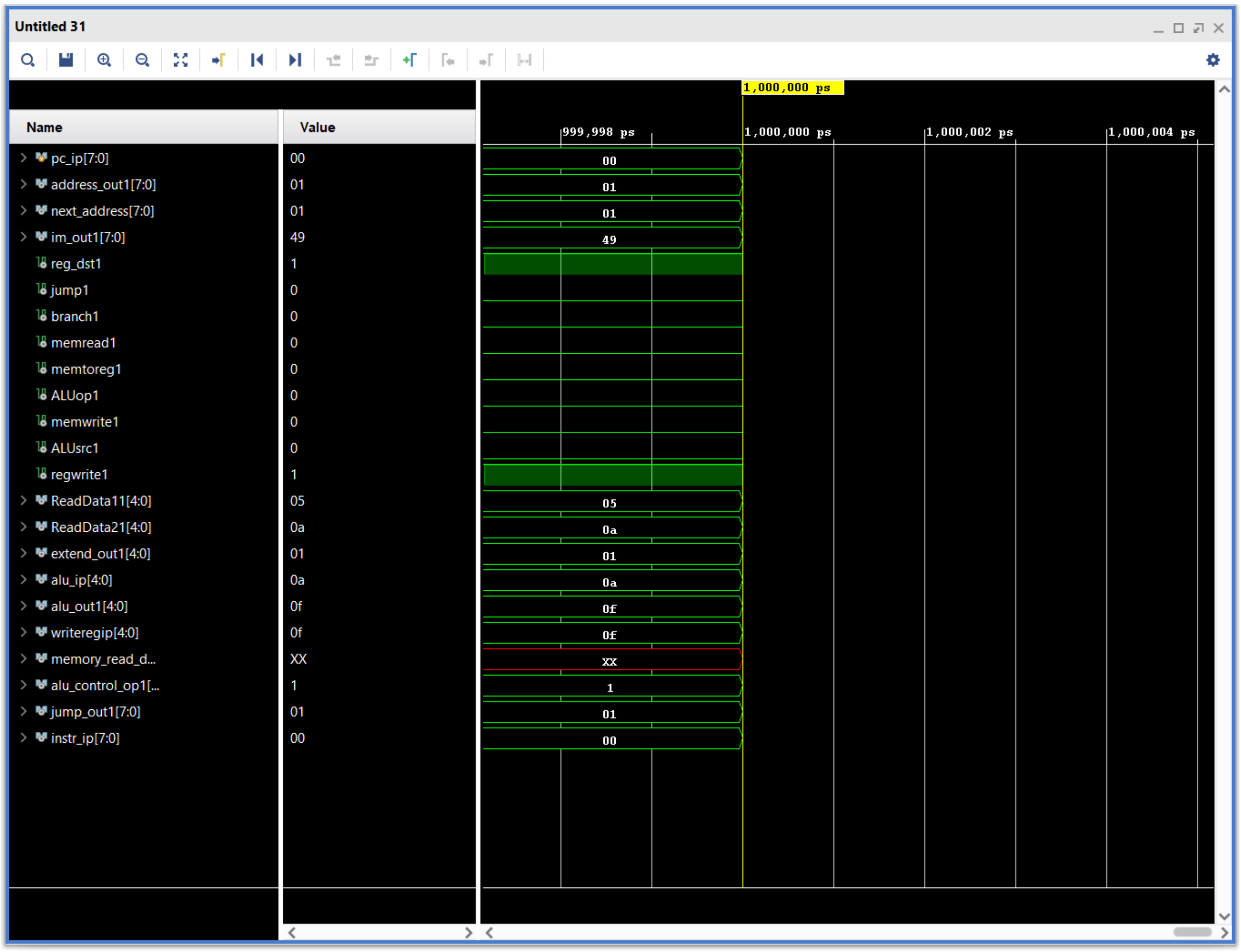
Load after Store



```
pc_ip[7:0]                03
address_out1[7:0]         04
next_address[7:0]         05
im_out1[7:0]              0a
reg_dst1                  0
jump1                     0
branch1                   0
memread1                  1
memtoreg1                 1
ALUop1                    1
memwrite1                 0
ALUsrc1                   1
regwrite1                 1
ReadData11[4:0]           05
ReadData21[4:0]           0a
extend_out1[4:0]          02
alu_ip[4:0]               02
alu_out1[4:0]             07
writeregip[4:0]           0a
memory_read_data1[4:0]    0a
alu_control_op1[2:0]      1
jump_out1[7:0]            02
instr_ip[7:0]             03
```

3 - Simulation of the ADD instruction

rt – R0 – 01010 - 10
rs – R1 – 00101 - 5
Write register - 01111 - 15 - 0F
Instruction - 8'b01001001
ADD R0 R1



Name	Value
pc_ip[7:0]	00
address_out1[7:0]	01
next_address[7:0]	01
im_out1[7:0]	49
reg_dst1	1
jump1	0
branch1	0
memread1	0
memtoreg1	0
ALUop1	0
memwrite1	0
ALUsrc1	0
regwrite1	1
ReadData11[4:0]	05
ReadData21[4:0]	0a
extend_out1[4:0]	01
alu_ip[4:0]	0a
alu_out1[4:0]	0f
writeregip[4:0]	0f
memory_read_d...	XX
alu_control_op1[...	1
jump_out1[7:0]	01
instr_ip[7:0]	00

- The instruction executed is the ADD R0 R1.
- The content of R0 is 10 and R1 is 5
- As you can see from the waveform, the result of the ALUop is 0F which is the HEX code for the number 15.
- The same results are taken to the writeregip to write the results back to the register R0.

4 - Simulation of the ADDI Instruction

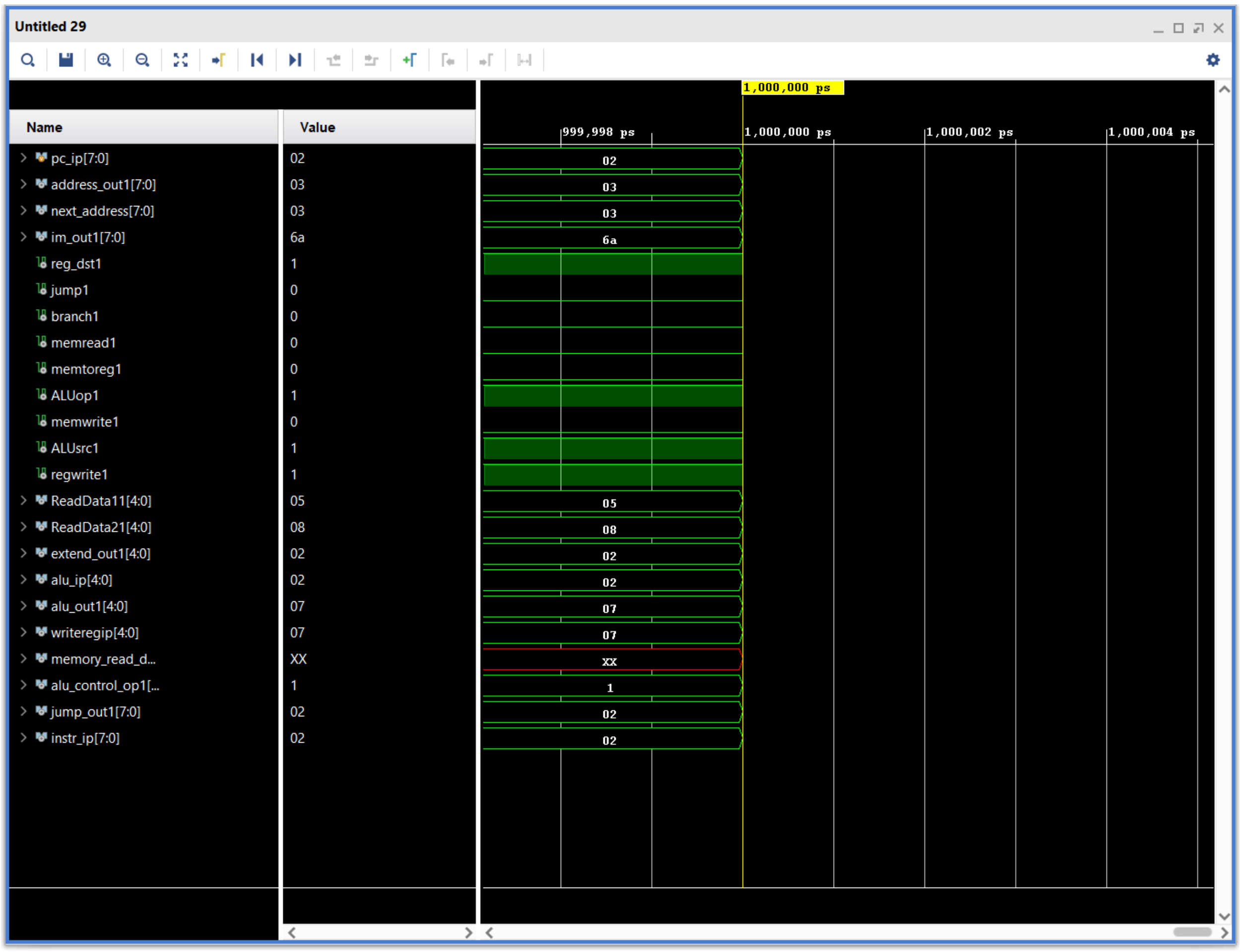
rs – R0 – 01000 - 8

rt – R1 – 00101 – 5

immediate – 010 - 2

Instruction - 8'b01101010

ADDI R0 R1 2

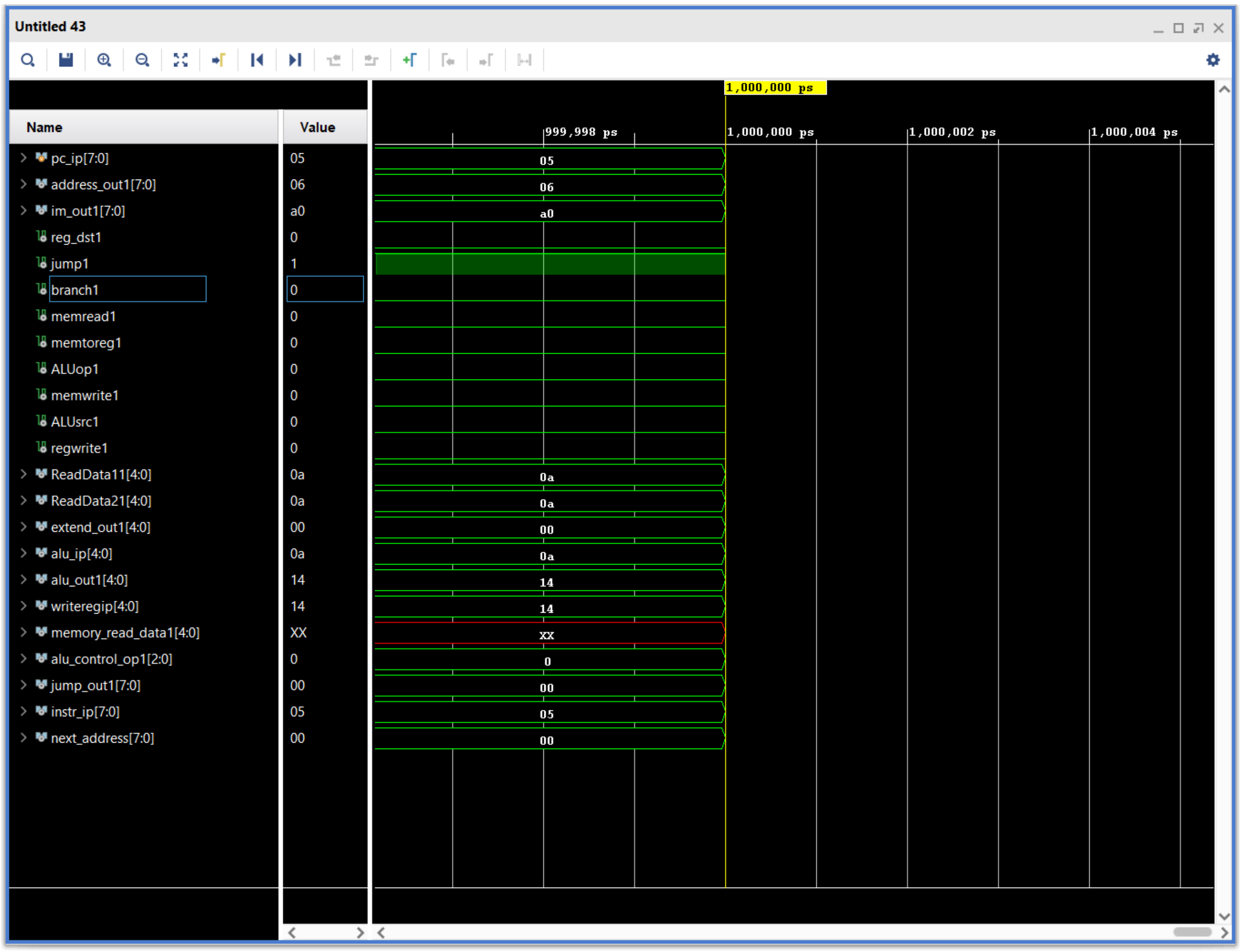


Name	Value
pc_ip[7:0]	02
address_out1[7:0]	03
next_address[7:0]	03
im_out1[7:0]	6a
reg_dst1	1
jump1	0
branch1	0
memread1	0
memtoreg1	0
ALUop1	1
memwrite1	0
ALUsrc1	1
regwrite1	1
ReadData11[4:0]	05
ReadData21[4:0]	08
extend_out1[4:0]	02
alu_ip[4:0]	02
alu_out1[4:0]	07
writeregip[4:0]	07
memory_read_d...	XX
alu_control_op1[...	1
jump_out1[7:0]	02
instr_ip[7:0]	02

- The instruction getting executed is ADDI R0 R1 2.
- The content of the Register R1 is 5. Immediate data 5 is added to the contents of the register R1 and later stored in the register R0.
- As seen from the waveform the output from the ALU is taken to the writeregip to be written to the register R0

5 - Simulation of the Jump Instruction

Instruction - 8'b10100000

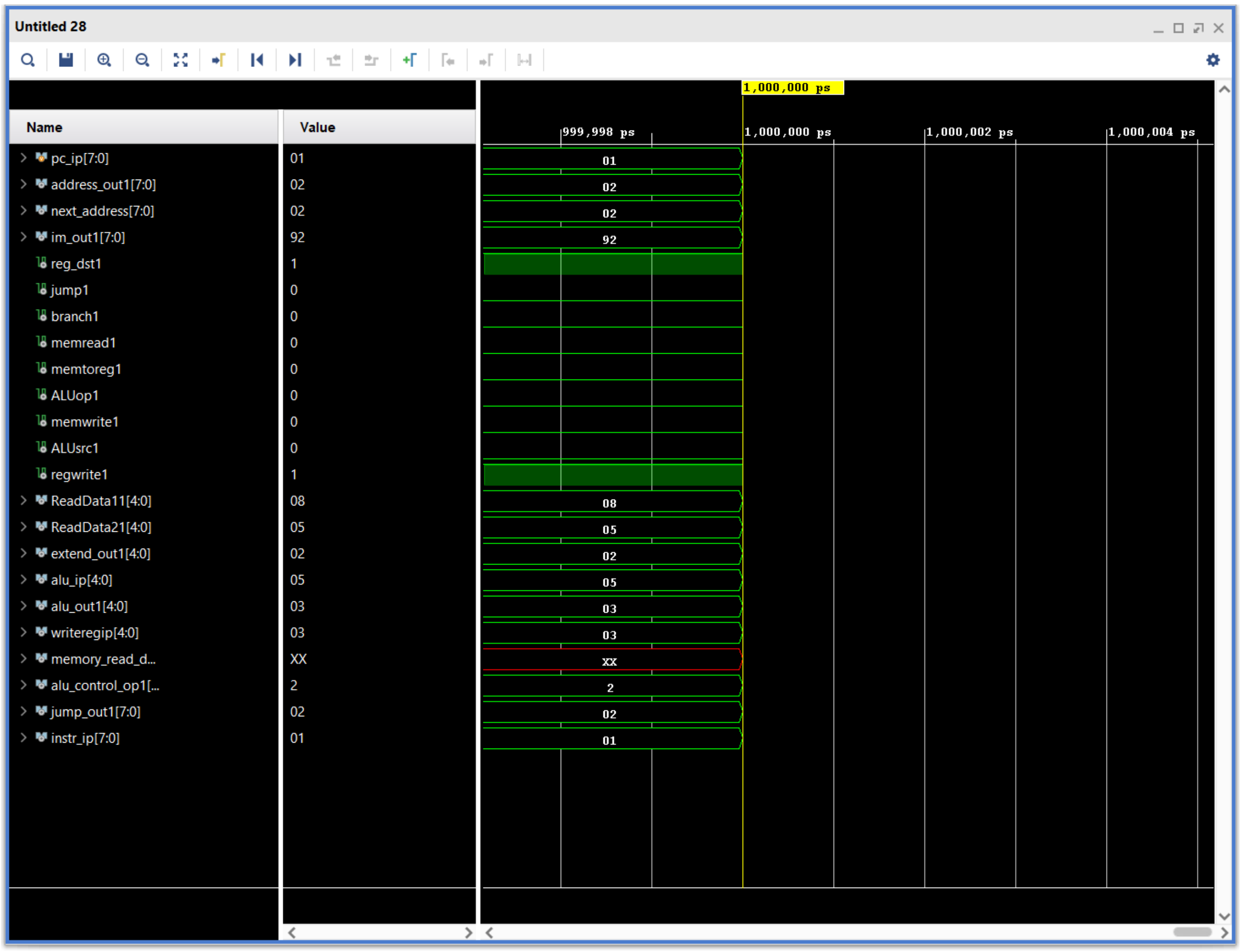


Name	Value
pc_ip[7:0]	05
address_out1[7:0]	06
im_out1[7:0]	a0
reg_dst1	0
jump1	1
branch1	0
memread1	0
memtoreg1	0
ALUop1	0
memwrite1	0
ALUsrc1	0
regwrite1	0
ReadData11[4:0]	0a
ReadData21[4:0]	0a
extend_out1[4:0]	00
alu_ip[4:0]	0a
alu_out1[4:0]	14
writeregip[4:0]	14
memory_read_data1[4:0]	XX
alu_control_op1[2:0]	0
jump_out1[7:0]	00
instr_ip[7:0]	05
next_address[7:0]	00

- The instruction executed is jmp 000.
- As seen in the simulation results, the next address has been selected as 000 which is the immediate part of the jmp instruction.
- The Jump ALU receives the jump signal from the control unit and selects between the ALU result and the result of the jump ALU.

6 - Simulation of the SUB Instruction

rs – R0 – 01000 - 8
rt – R1 – 00101 - 5
Write register - 00011 - 03
Instruction - 8'b10010010
SUB R0 R1



Name	Value
pc_ip[7:0]	01
address_out1[7:0]	02
next_address[7:0]	02
im_out1[7:0]	92
reg_dst1	1
jump1	0
branch1	0
memread1	0
memtoreg1	0
ALUop1	0
memwrite1	0
ALUsrc1	0
regwrite1	1
ReadData11[4:0]	08
ReadData21[4:0]	05
extend_out1[4:0]	02
alu_ip[4:0]	05
alu_out1[4:0]	03
writeregip[4:0]	03
memory_read_d...	XX
alu_control_op1[...	2
jump_out1[7:0]	02
instr_ip[7:0]	01

- The instruction executed is the SUB R0 R1.
- The content of R0 is 8 and R1 is 5
- As you can see from the waveform, the result of the ALUop is 03 which is the HEX code for the number 3.
- The same results are taken to the writeregip to write the results back to the register R0.

Work Distribution Among Members of the Team

The work was evenly distributed among both of the team members. Each one was individually held responsible for designing a module of the MIPS processor ,drawing the block diagram, typing the corresponding part of the report. Later the entire processor was put together and tested for simulation results, synthesis, implementation and port assignment.