COEN 210 PROJECT REPORT

PARTH SHAH - W1549569
AKSHAY PARKALA - W1590736
SANTOSH VISWA MOHAN NANDURI - W1597096

TABLE OF CONTENTS

1. LIST OF ASSUMPTIONS

- Byte addressable architecture.
- Number of elements is assumed to be a multiple of 4 while implementing Loop Unrolling.
- X6 has the base address of the array.
- X5 maintains the additions.
- X4 has the number of elements in the array.
- For the Super instruction, only one register(x6 in our example) is used to store the base address and as the destination register for the final sum.

2. INSTRUCTION SET ARCHITECTURE

2.1 Instruction format

Const.	Rd	Rs1	Rs2	Opcode
6 bits	7 bits	7 bits	7 bits	5 bits

Instruction length = 6+7+7+7+5 = 32 bits.

2.2 List of all instructions

INSTRUCTION	SYMBOL	OPCODE	RD	RS 1	RS 2	IMM	FUNCTION
Load	LOAD	00 011	rs1	rs2	x	x	rs1 ← rs2
Store	STORE	01 011	x	rs1	rs2	x	rs2 ← rs1
Add	INC	11 001	rd	rs1	rs2	x	rd =rs1 + rs2
Add Immediate	INCI	00 001	rs1	rs2	imm		rs1 = rs2+ imm
Subtract	DEC	10 010	rd	rs1	rs2	x	rd = rs1 - rs2
AND	AND	10 011	rd	rs1	rs2	x	rd = rs1 AND rs2
And Immediate	ANDI	00 100	rs1	rs2	imm		rs1 = rs2 AND imm
OR	OR	10 101	rd	rs1	rs2	x	rd = rs1 OR rs2
NOR	NOR	10 110	rd	rs1	rs2	x	rd = rs1 NOR rs2
Jump and Link	JAL	11 000	rs1	imm			PC = imm rs1 = PC + 4
Jump and Link Register	JALR	11 001	rs1	rs2	imm		PC= rs2+imm rs1 = PC + 4

Branch if equal	DIFF	11 010	imm	rs1	rs2	imm	If rs1 > rs2 => PC = PC + imm
Super	ADDMTX	11 111	reg	reg	reg	imm	reg = sum of array with base address in reg and 'imm' number of elements

2.3 Addressing modes

1. Register addressing:
   - Inc x1, x2, x3 : [x1] = [x2] + [x3]
   - Dec x1, x2, x3 : [x1] = [x2] - [x3]
   - And x1, x2, x3 : [x1] = [x2] and [x3]
   - Or x1, x2, x3 : [x1] = [x2] or [x3]
   - Nor x1, x2, x3 : [x1] = [x2] nor [x3]
   - Load x1, x2 : [x1] = mem[x2]
   - Store x1, x2 : [x2] = mem[x1]

2. Immediate addressing:
   - Inci x1, x2, 100 : [x1] = [x2] + 100
   - Andi x1, x2, 100 : [x1] = [x2] & 100
   - Addmtx x6, n : [x6] = $\sum_{i=0}^{n-1} mem[x6 + i]$

3. PC-relative addressing:
   - Jal x1,2000 : PC = 2000; x1 = PC+4
   - Jalr x1, x2, 2000 : PC = x2+2000; x1 = PC+4
   - Diff x1, x2, 100 : PC = PC + 100 if x1 = x2 else PC = PC + 4

2.4 Justification of the proposed Instruction Set Architecture

Instructions like Load and Store instructions don't have immediate. Because of this, the Execution stage and the Memory stage can be combined into one single stage where either only one of ALU or Data Memory is used for an instruction. This reduces the cycle time. Also since every instruction uses only one maximum time consuming units, the clock period is kept to a minimum of 2NS.

Pros:

1. Less execution time because the pipeline architecture combined both execution and memory stages.
2. Since the lower 3 bits of the opcode are used as control lines to the ALU, we don't need an extra "ALU Control" unit like in RISC V architecture to let the ALU know the operation that needs to be run.
3. Because the "Execution+Memory" stage has ALU, Data Memory and Adder together, Super instruction can be implemented so that all three units are run together in parallel. Thus, this design boasts an execution time of the benchmark program that is unprecedented.
4. The Forwarding Unit is utilised effectively to implement the Super instruction, thereby removing the need to access the register file to fetch the partial sum of the array for each accumulation of a new element.

Cons:

1. Because the Load and Store instructions don't support any immediate, the source register needs to have the address from which we plan to load to or store from. This means that we need an extra instruction before the actual Load or Store instructions to move the address into the source register.
2. The Control Unit must include hardware to support the n+5 cycles of the Super instruction, and thus would be costlier than a conventional control unit. Though, since ALU control is not needed by this design, the cost remains unaffected drastically.

Difference between RISC V and proposed Architecture:

1. Number of pipeline stages is one stage less compared to RISC-V .
2. A new functional block called "Jump Branch Unit" is added to drive the next Program Counter and to determine when the pipeline buffers need to be flushed.
3. A new super instruction called "ADDMTX" is implemented to add contents of an array when the base address and the number of contents of the array are given.
4. The need for an ALU control unit is removed.
5. There are two cycles lost when the control takes a branch, compared to only one in the RISC-V architecture.

3. APPLICATION

3.1 Assembly Program without Loop Unrolling

```
INCI X6, X0, 1000      // X6 has base address of the array
INCI X5, X5,  0        // X5 maintains the additions
INCI X4, X0, N         // X4 has number of elements in the array
LOAD X7, X6
INCI X6, X6, -4
INC X4, X4, -1
INC X4, X5, X7
DIFF X4, X0 , -16      //This takes the instruction execution back to Load instruction
NOP
NOP
```

3.2 Assembly Program with Loop Unrolling

Every time the loop is unrolled, 2 instructions INC, DIFF and 2 NOP instructions in the branch delay slot can be reduced. In the following assembly code where the loop is unrolled by 4, a total of 12 instructions can be reduced. Performance improvement using Loop Unrolling is shown in PERFORMANCE ANALYSIS

```
INCI X6, X0, 1000      // X6 has base address of the array
INCI X5, X5,  0        // X5 maintains the additions
INCI X4, X0, N         // X4 has number of elements in the array
LOAD X7, X6
INCI X6, X6, -4
INCI X5, X5, X7
LOAD X7, X6
INCI X6, X6, -4
INC X5, X5, X7
LOAD X7, X6
INC X6, X6, -4
INC X5, X5, X7
LOAD X7, X6
INC X6, X6, -4
INC X4, X4, -4
INC X4, X5, X7
DIFF X4, X0      -52   //This takes the instruction execution back to the first Load
NOP
NOP
```

3.4: Assembly Program for the Benchmark

Design supports a Super Instruction called “ADDMTX” that is used to implement the benchmark code. This is explained in a latter chapter 4.3 Datapath to implement Super Instruction

ADDMTX X6, N

3.4 Emulation of RISC V instructions:

Instruction	Emulation
BEQ x1, x2, 1000	DIFF x1, x2, 1000
JAL x1, 2000	JAL x1, 2000
JALR x1,24(x2)	JALR x1,24(x2)
ADDI x1, x2, 100	INCI x1, x2, 100
ANDI x1, x2, 100	ANDI x1, x2, 100
ADD x1,x2,x3	INC x1,x2,x3
SUB x1, x2, x3	DEC x1, x2, x3
AND x1, x2, x3	AND x1, x2, x3
OR x1, x2, x3	OR x1, x2, x3
NOR x1, x2, x3	NOR x1, x2, x3
SD x1, imm(x2)	ADDI x2, x2, imm STORE x1, x2 DECI x2, x2, imm
LD x1, imm(x2)	ADDI x2, x2, imm LOAD x1, x2 DECI x2, x2, imm

4. CPU ORGANIZATION

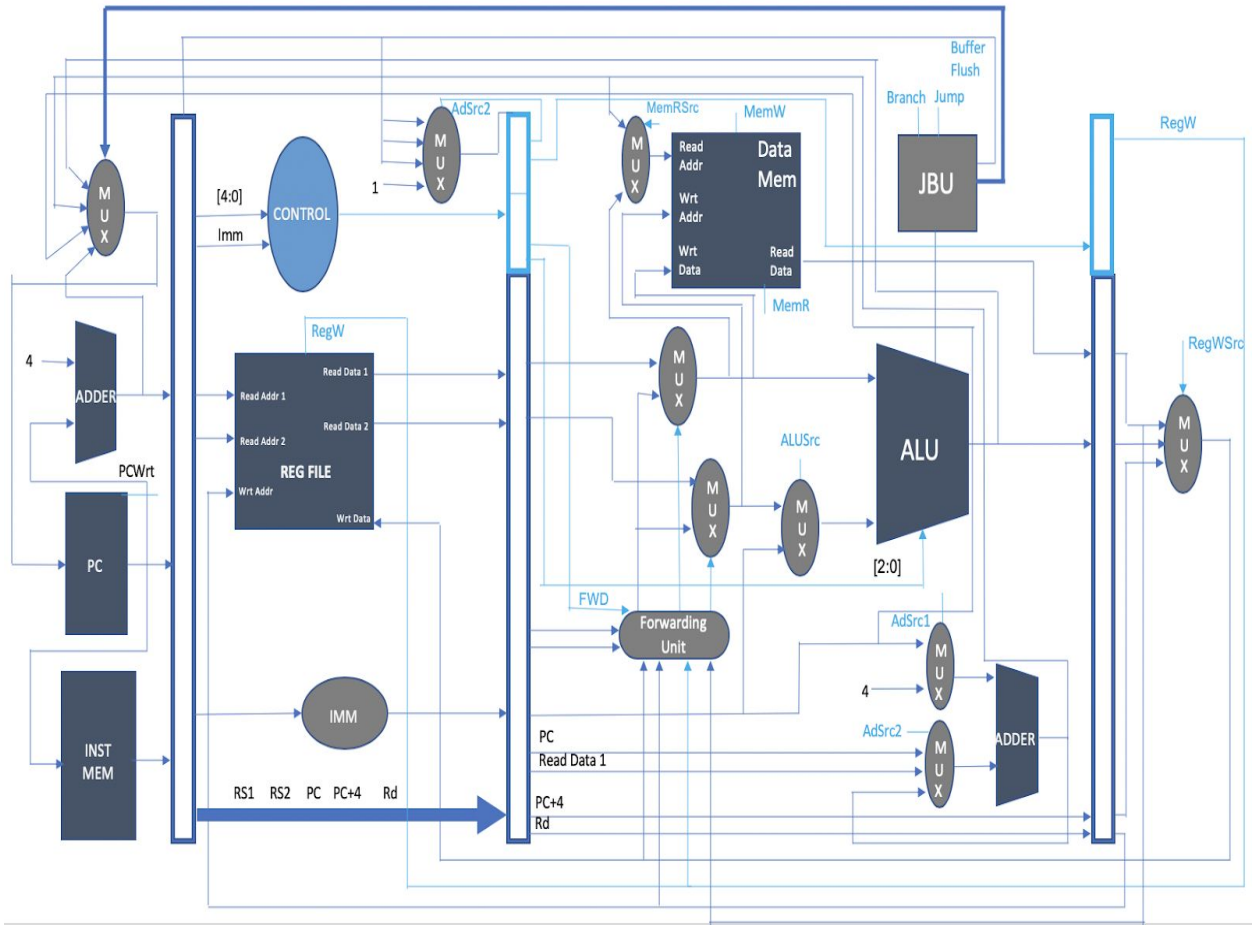
4.1 List of Hardware Components

1. PC: Program Counter
2. Instruction Memory
3. Register File
4. Data Memory
5. ALU
6. Adder
7. Immediate Generator
8. Adder
9. JBU
10. Multiplexers
11. Control Unit

One important aspect of this design is that we removed the need for an ALU Control Unit by incorporating the ALU OpCode into the Opcode for an instruction: the first 3 bits of the opcode for every instruction act as the ALU Control bits.

There is a new hardware unit(zero delay) called the JBU, which stands for Jump and Branch Unit. The inputs to this unit are the Jump, Branch control signals and the zero flag. The outputs are the PCSrc.

4.2 Datapath with Pipeline Implementation, Forwarding & Flushing



Pipeline Stages:

In this design, both the EX stage and MEM stage are clubbed to be executed parallely, since the instruction set architecture does not support immediate addressing for the load instructions. This enabled the 4-stage pipeline: IF(Instruction Fetch), ID(Instruction Decode), EX/MEM(Execution/Memory), WB(Register Write-Back). Consequently, the register buffers are IF/ID, ID/EXMEM, EXMEM/WB.

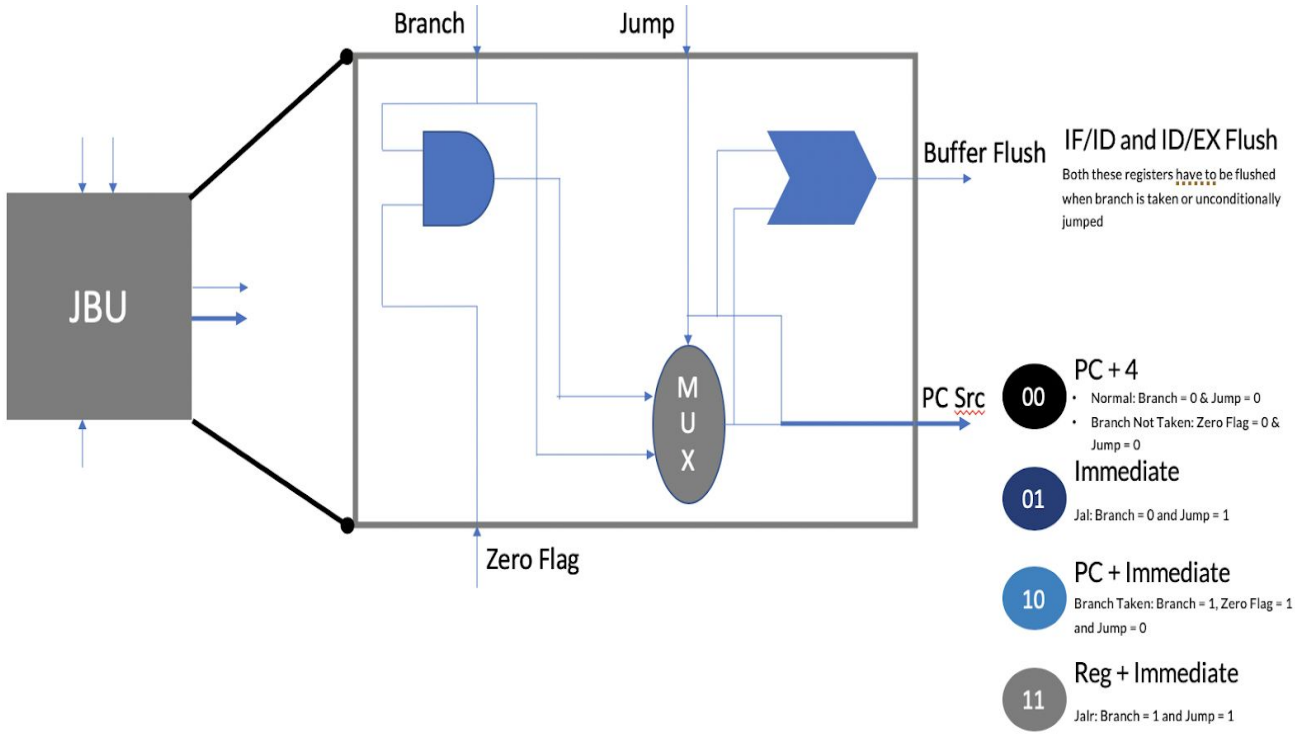
Flushing:

Flushing is implemented with the help of JBU. When either the branch is taken(PC + Imm) or when the control needs to be unconditionally jumped (Imm, Reg + Imm), the JBU asserts the Buffer Flush line which in turn clears the IF/ID and ID/EXMEM buffers.

Forwarding Unit:

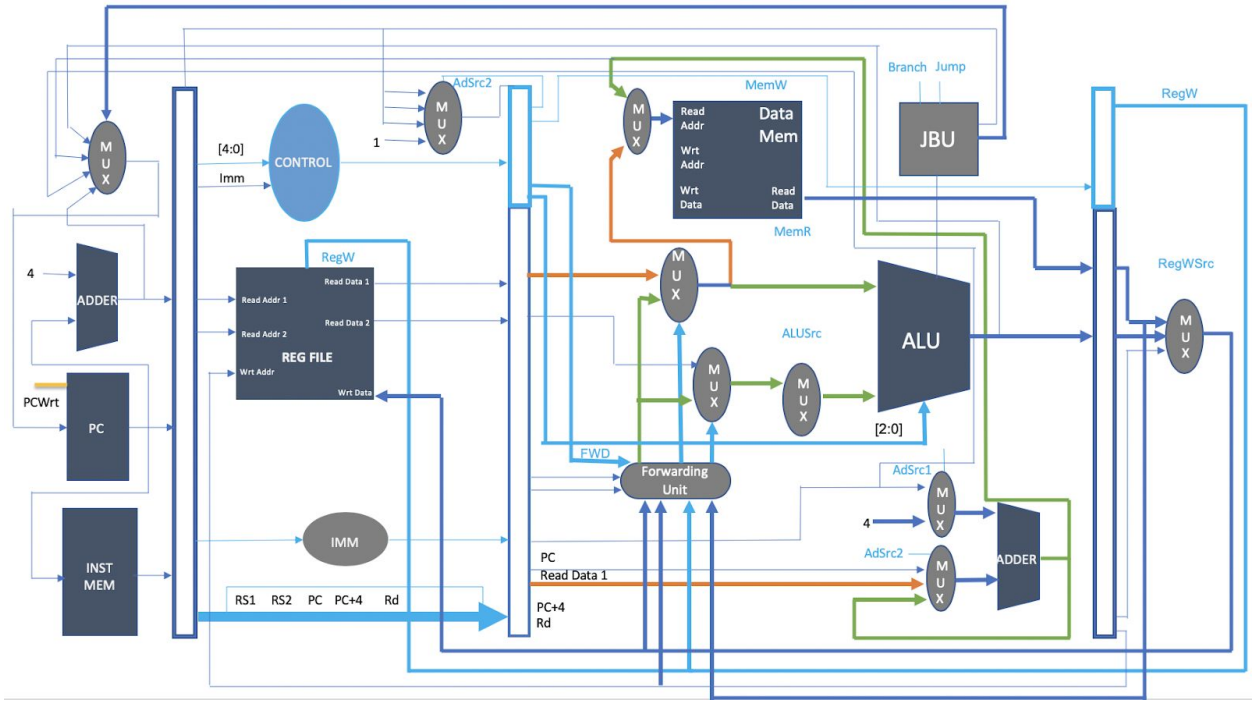
In this design, the Forwarding Unit plays a major role in the enabling of super instruction. It has a control signal FWD, which is only asserted for the super instruction.
For the rest of the instructions, the FWD signal is deasserted and the function of the unit is similar to the RISC-V datapath.

Jump and Branch Unit:



The JBU is a zero delay hardware unit which outputs the PCSrc control signals and the buffer flushing. It is placed in the EX/MEM stage and thus, on unconditional jumps and branching, there are two cycles lost in the pipeline. For these two instructions, the JBU asserts the Buffer Flush signals and clears the IF/ID and ID/EXMEM buffers.

4.3 Datapath to implement Super Instruction



1st Cycle

2 to n Cycles

n+1st Cycle

The Super instruction effectively utilises the EX/MEM stage and Forwarding unit in the pipeline process. This instruction, as any other, takes 2 cycles to reach the EX/MEM stage. After the first two cycles, the instruction continuously loops in the EX/MEM and WB stage.

Control Unit:

The control unit outputs three sets of control signals(2nd, 3 to n+1st, n+2nd cycle) for the super instruction to be executed. The corresponding truth table is mentioned in section 4.4. A point to be noted here is that, only for this instruction, the control unit manipulates the ALU control bits present in the instruction format. The following table elicits the changes in the control signals:

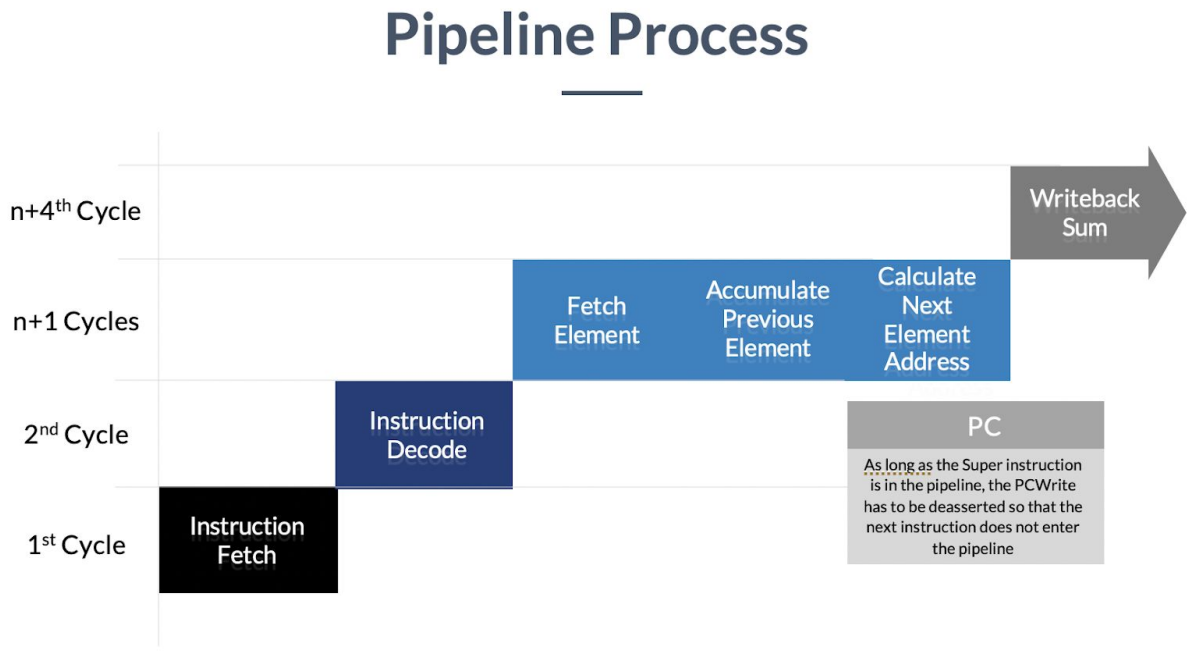
CYC	ALUOp	PCWrt	MemR	MemRSrc	AdSrc1	AdSrc2
2	111 The ALU outputs 0 in the first cycle of EX/MEM stage	0 The PCWrt is deasserted when the super instruction reaches the ID stage	1 The data memory outputs an element each in the first n cycles	0 The base address is delivered by “Read Data 1”	1 The first input to the Adder in the EXMEM stage is 4, for it to be able to calculate the address of the next element. Note: The adder delivers the read address for the data memory only from the second cycle	01 The base address is delivered as the second input to the adder in the first cycle
3 to n+1	001 The ALU accumulates the element generated in the previous cycle for the next n cycles			1 The address of all the other elements are delivered by the output of the adder in the EXMEM stage		10 From the second cycle onwards, its own output is delivered as the second input
n+2		1 PCWrt is reasserted when the final sum is accumulated in the EX/MEM stage. Therefore, a bubble is introduced in the pipeline via the ID/EXMEM buffer	0	X	0	X(11) The don’t care here is used as 11 to manipulate the flushing implementation. The mux which outputs the flushing signal for ID/EXMEM buffer uses AdSrc2 as control lines, and the 11 input to that mux is 1. For the rest of the three inputs, the usual Buffer flush signal which is output by the JBU is used as input.

EX/MEM Stage:

The super instruction engages this stage for a total of n+2 cycles, the n+2nd cycle being a bubble. The data memory, ALU and adder all run in parallel: in each cycle, data memory outputs the next element of the array, ALU accumulates the previous element and the adder generates the address of the next element in the array.

WB Stage:

In the first cycle, the initial sum of 0 is written to the register mentioned in the instruction. In the following n cycles, an element of the array is accumulated in each cycle and written to this register. In the n+4th cycle, the final sum is written.



Forwarding Unit

FWD Control Signal	Function
1 (only Super instruction)	Forward RegWrtOutput and EXMEM/WB.MemoryOutput to MUX1 and MUX2 respectively
0(All other instructions)	Like RISC-V

4.4 Control

Truth Tables:

ALU Control Lines	Function
111/000	Do Nothing/Output 0
001	Add
010	Subtract
011	Input 1
100	And
101	Or
110	Nor

INSTRUCTION	SYMBOL	OPCODE (ALUOp)	ALU Src	Branch	Mem Read	Mem Wrt	Reg W Src	Reg Wrt	Jump	Mem R Src	Ad Src1	Ad Src2	Fwd	PC Src	PC Wrt
Load	ld	00 011	x	0	1	0	00	1	0	1	0	00	0	00	1
Store	sd	01 011	x	0	0	1	x	0	0	x	0	00	0	00	1
Jump and Link Register	jalr	11 001	1	0	0	0	10	1	1	x	0	00	0	11	1
Add	add	11 001	0	0	0	0	01	1	0	x	0	00	0	00	1
Subtract	sub	10 010	0	0	0	0	01	1	0	x	0	00	0	00	1
And	and	10 011	0	0	0	0	01	1	0	x	0	00	0	00	`1
And Immediate	addi	00 100	1	0	0	0	01	1	0	x	0	00	0	00	1
Or	or	10 101	0	0	0	0	01	1	0	x	0	00	0	00	1

NOR	nor	10 110	0	0	0	0	01	1	0	x	0	00	0	00	1
Add Immediate	addi	00 001	1	0	0	0	01	1	0	x	0	00	0	00	1
Jump and Link	jal	11 000	x	0	0	0	10	1	1	x	0	00	0	01	1
Branch if equal	beg	11 010	0	1	0	0	x	0	0	x	0	00	0	10	1

Super Instruction:

CYC	ALUOp	ALU Src	MemR	MemR Src	MemW	Ad Src 1	Ad Src 2	Reg W	RegW Src	PCWrt	PC Src	Jump	Branch	Fwd	CYC
2	000	0	1	0	0	1	01	1	01	0	x	0	0	1	1
3 to n+1	001	0	1	1	0	1	10	1	01	0	x	0	0	1	2 to n
n+2	001	0	0	X	0	0	X	1	01	1	00	0	0	1	n + 1

5. PERFORMANCE ANALYSIS

5.1 Detailed Performance Analysis

1) Cycle time
2) Instruction count
3) Total number of cycles in terms of n
4) Optimization

5.2 Execution Time without Loop Unrolling:

Instruction Count	=	3 + 7N
Cycle Count	=	4 + (3 + 7N - 1) = 6 + 7N
CPI	=	#Cycles / #Instructions = (6 + 7N) / (3 + 7N) ≈ 1
Clock Period	=	2ns
Total Executions time	=	CPI * Cycle Count * Clock Period
	=	(1 * (6 + 7N) * 2) ns
	=	(12 + 14N) ns

5.3 Execution Time with Loop Unrolling:

Instruction Count	=	3 + (16n/4) = 3 + 4N
Cycle Count	=	4 + (3 + 4N - 1) = 6 + 4N
CPI	=	#Cycles / #Instructions = (6 + 4N) / (3 + 4N) ≈ 1
Clock Period	=	2ns
Total Execution time	=	CPI * Cycle Count * Clock Period
	=	(1 * (6 + 4N) * 2) ns
	=	(12 + 8N) ns
Improvement in performance	=	$\frac{Execution\ time\ without\ Loop\ Unrolling}{Execution\ time\ with\ Loop\ Unrolling}$
	=	(12 + 14N) / (12 + 8N) ≈ 1.75

5.4 Super Instruction Performance:

Instruction Count	=	1
Cycle Count	=	5 + N
CPI	=	#Cycles / #Instructions = (5 + N) / (1) = 5 + N
Clock Period	=	2ns
Total Executions time	=	CPI * Cycle Count * Clock Period
	=	(1 * (5 + N) * 2) ns
	=	(10 + 2N) ns

Performance improvement using Super Instruction:

$$= \frac{Execution\ time\ with\ Loop\ Unrolling}{Execution\ time\ with\ Super\ Instruction}$$

= (12 + 8N) / (10 + 2N) ≈ 4
