NSC-8 Architecture Report

Prepared by: Nolan Su-Hackett

Student in Faculty of Engineering and Applied Science

Queen's University

Table of Contents

1. Introduction

The Simple As Possible-1 (SAP-1) architecture is detailed by Malvino and Brown used to teach the foundations of computer architecture and the digital logic that supports it [1]. The purpose of this report is to detail Nolan's Simple Calculator-8 (NSC-8) which serves as an expansion to the SAP-1 adding 5 new instructions to the original 5. This report demonstrates how new instructions can be integrated with existing hardware through control logic and timing design.

This report begins with an overview of the NSC-8 architecture, touching on the large components of the CPU as well as general data path principles. This section includes a block diagram with associated controls and the complete instruction set. The following section of the report outlines the fetch cycle, specific controls, timing states, and digital logic for each new instruction implemented in the NSC-8. The final section of the report goes over testing that was conducted for the NSC-8 and how correct implementation of instruction and logic was verified.

2. System Overview

The NSC-8 is compatible with the SAP-1 architecture detailed by Malvino and Brown and so it closely follows the general timing flow and shares many common controls with it. This section showcases how the NSC-8 adds to the architecture and illustrates its block diagram with its respective controls. Along with the SAP-1's LDA, ADD, SUB, JUMP, and HALT instructions NSC-8 has integrated digital logic to support the following additional 5 instructions: LDAI, STA, ADDI, SUBI, and JUMP. Each of the added instructions introduces their own specific controls, detailed in Section 3 of this report. Descriptions and high-level details of the instruction set can be seen in Table 1. As for the major components of the NSC-8 it contains the same 10 major components as SAP-1. Figure 1 displays a block diagram with each main component of the NSC-8 with control instructions connected to their respective blocks. Each component of this architecture has its own role in the program control flow, instruction decoding, and data path execution.

2.1. Program Control Flow

The program counter, random access memory (RAM), and memory address register (MAR) are crucial in the program control flow, as these components drive the setup for each instruction. The program counter is a 16-modulo counter meaning on clock and enable pulses it counts in increments of 1 up to 15 and then resets to 0. The RAM is a 16x8 memory unit, meaning that it has 16 rows each comprised of 8 bits of information, in reality this unit is comprised of two 16x4 units each for upper and lower bits respectively. The MAR is a 4-bit register which is used to address one of the 16 rows of the RAM. The purpose of the program counter is to keep track of the natural progression of program execution and communicate to the MAR what row in memory the next program is stored. The role of the program counter is crucial as it allows instructions to be executed in a top-down manner, enabling a set of instructions to be completed. For instance,

instructions in RAM at address 0, 1, and 2 will be executed sequentially in that order as a result of the counting capability of this component coupled with relevant controls. The main role of the RAM in the NSC-8 is to store both instructions and data. This includes values needed for operations as well as empty memory locations used to store results, such as the output of an addition.

2.2. Instruction Decoding

The instruction register (IR) and the controller/sequencer are the parts that govern the identification of instructions, allowing the CPU to know what instruction is being executed, passing the relevant information and controls necessary to complete it. The instruction register is made up of two 4-bit registers responsible for storing the current instruction and the operand. One of the 4-bit registers is used to store the upper nibble or operation code and the other is used to store the lower nibble or operand for the instruction. The descriptions for each of the 10 instructions as well as the purpose of the operand can be seen in Table 1. The controller is responsible for decoding the instruction of the upper nibble in the instruction register, to send out specific controls to other components in the CPU. With the decoded instruction it is only possible to know what controls the controller will need to output but not what times, using a J-K 6-state ring counter the NSC-8 can combine timing and instruction information to know what controls to send out as well as its sequence.

2.3. Data Path Execution

Data path execution gets into the real execution of instructions after they have been decoded and processed by the controller. The components that are included are accumulator A, adder/subtractor, B register, and the bus although it is used throughout all processes. The bus is an 8-bit wide wire that runs vertically down the middle of the NSC-8 as seen in Figure 1, it provides a shared pathway for each component of the NSC-8 to communicate without direct connection. Accumulator A is made up of two 4-bit registers, it serves a dual-purpose, first as an operand to an equation or instruction and second as the default location for storing the result of operations like additions and subtractions. The B register contains the second operand of an equation and based on the instruction, A and B will be used to produce the result of logic operations with the adder/subtractor unit. The output component is used to show through a display what value is currently in the A accumulator.

Table 1: NSC-8 Instruction Set

Name	OP Code (Bits 4-7)	Operand (Bits 3-0)	Description
LDA	0000	Address in RAM	Writes data from passed address to accumulator A
ADD	0001	Address in RAM	Writes data from passed address to B register, performs add operation with contents of B register with what is currently in A, writes result to accumulator A
SUB	0010	Address in RAM	Writes data at passed address to B register, performs subtract operation with contents of B register and what is currently in A, writes result to accumulator A
LDAI	0011	Immediate Value	Writes immediate value to accumulator A
STA	0100	Address in RAM	Writes value from accumulator A to passed address in memory
ADDI	0101	Immediate Value	Reads immediate value to B register, performs add operation with contents of B register and what is currently in A, outputs result to accumulator A
SUBI	0110	Immediate Value	Reads immediate value to B register, performs subtract operation with contents of B register and what is currently in A, outputs result to accumulator A
JUMP	0111	Immediate Value	Writes immediate value to program counter in preparation to skip to a certain address in memory on following fetch cycle
OUT	1110	Irrelevant	Writes value from accumulator A to output register
HLT	1111	Irrelevant	Stops the program, pauses all clock signals while active.

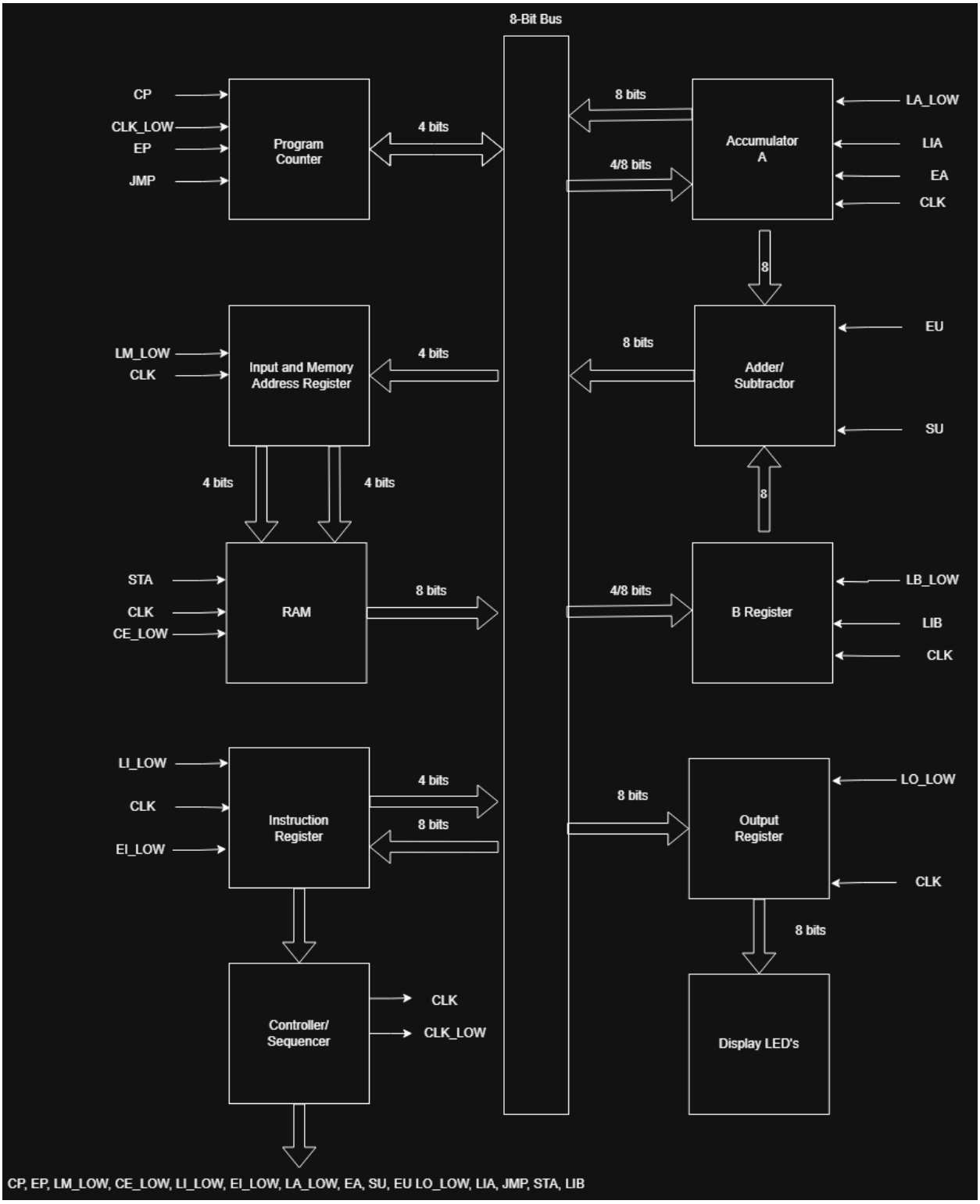

Figure 1: Complete block diagram of NSC-8 with respective controls

3. Control Logic and Design

This section details how the new instructions implemented in NSC-8 were designed and integrated into the SAP-1 architecture. The main components of control logic and design will be explained including the general fetch cycle, timing cycles for each new instruction, relevant controls, and digital design changes to the circuit.

3.1. Fetch Cycle

The fetch cycle of the NSC-8 is identical to that of the SAP-1 as the NSC-8 is an extension of it, this cycle details how instructions are fetched from memory in a top-down manner and describes the process from the program counter, to memory, to instruction register. The ring counter is made up of 6-time states as mentioned in Section 2.1, and the fetch cycle is split into the first three states. In time state 1 the value of the program counter is sent out on the bus using signal EP, then is received by the MAR from the bus with signal LM_LOW and addresses the RAM through direct connection. In time state 2 the program counter is incremented using the signal CP. In time state 3 the contents of the RAM go out onto the bus using signal CE_LOW, then is received by the instruction register from the bus with signal LI_LOW, its upper nibble is automatically transferred to the controller via direct connection. An image illustrating this process with respective controls and timing states can be seen in Figure 2. Each of the following instructions will detail the timing flow and controls after this process as the fetch cycle is repeated in the beginning of every instruction cycle.

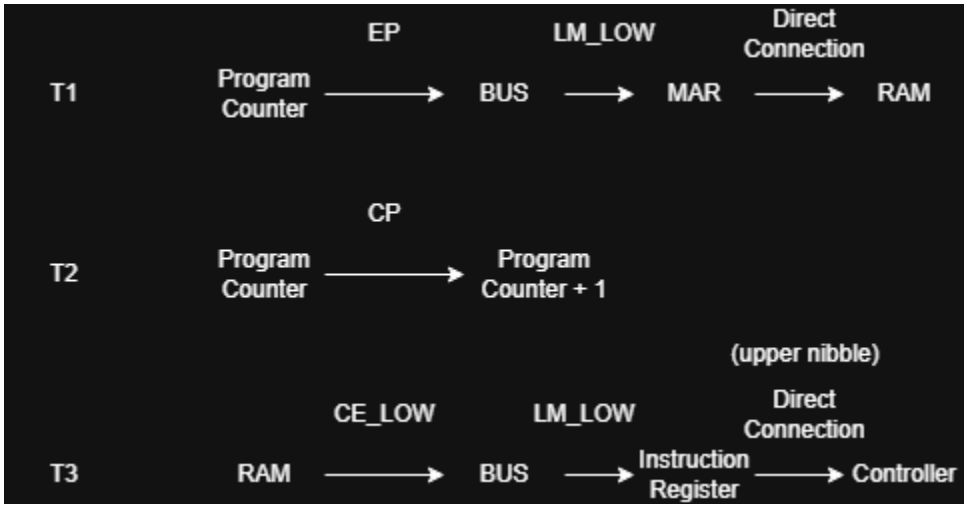

Figure 2: Illustration displaying fetch cycle timing states and controls

3.2. LDAI – Load Immediate

The load immediate instruction loads the bottom 4 bits of the instruction register directly to the A accumulator instead of sending addressed contents of the RAM like the normal load. On T4 the lower nibble of the instruction register is sent onto the bus through signal EI_LOW, then is received by the A accumulator with signal LIA. This is accomplished with digital logic by using an OR gate with inputs as LA_LOW or LIA as either can cause a write to this register, for this instruction the upper nibble is always 0 as there is not enough information. As a result of this, the signal LIA is connected directly to the clear of the upper nibble A register ensuring that it is always set to 0 on load immediate instruction. Nothing

happens on states 5 and 6, although the clock still runs through these timings as will be the case for many instructions that end before T6.

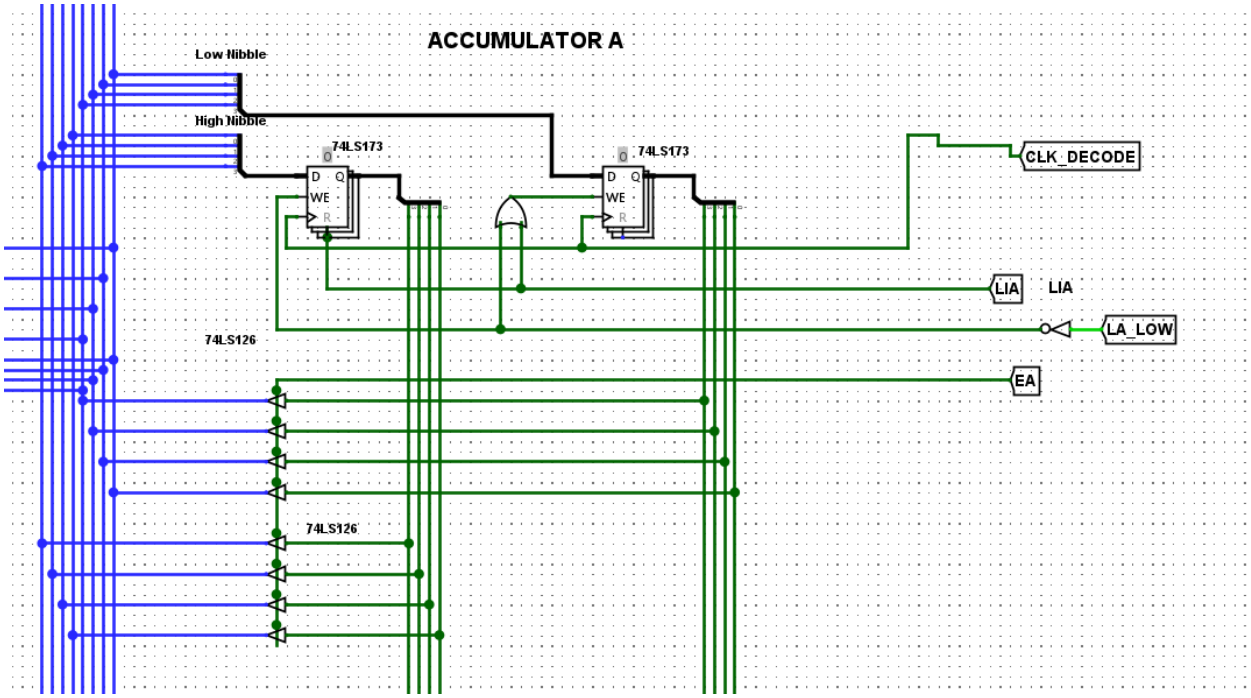

Figure 3: Logisim diagram displaying digital logic implementation of load immediate

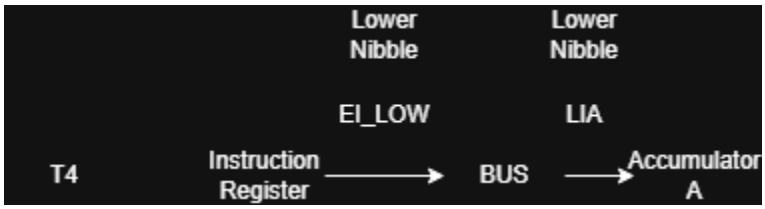

Figure 4: Illustration displaying load immediate timing states and controls

3.3. STA – Store Accumulator A

The store accumulator A instruction sends the contents of register A to an address in the RAM specified by the lower nibble of the passed instruction. On T4 the lower nibble of the instruction register is passed onto the bus using signal EI_LOW, these bits are received from the bus by the MAR using signal LM_LOW which then addresses the RAM through direct connection. On T5 the contents of the A register are sent onto the bus using signal EA, those contents are received from the bus by the RAM using signal STA and is stored at the specified address, an illustration can be seen in Figure 6. SAP-1 does not support non manual writes to memory and so the manual and bus inputs on T5 must be multiplexed using STA as the signal as shown in Figure 5. This multiplexed data is used as the input to the RAM and the write enable is dependent on the OR gate with inputs as signal STA and the manual write enable signal.

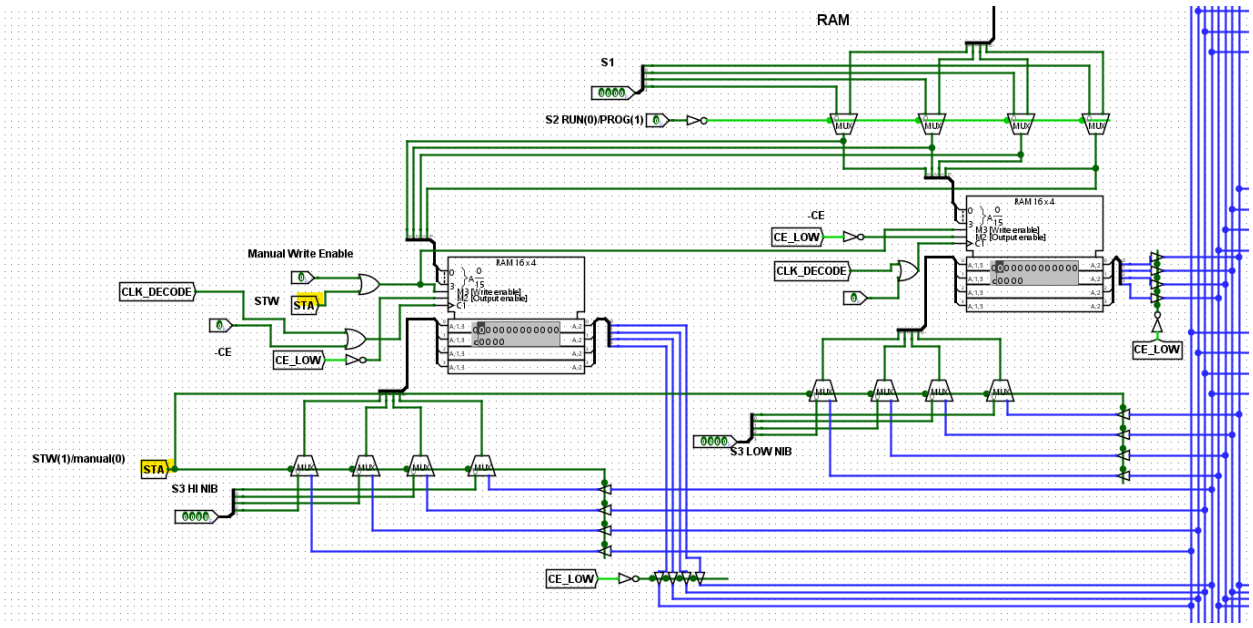

Figure 5: Logisim Diagram displaying digital logic implementation of store instruction

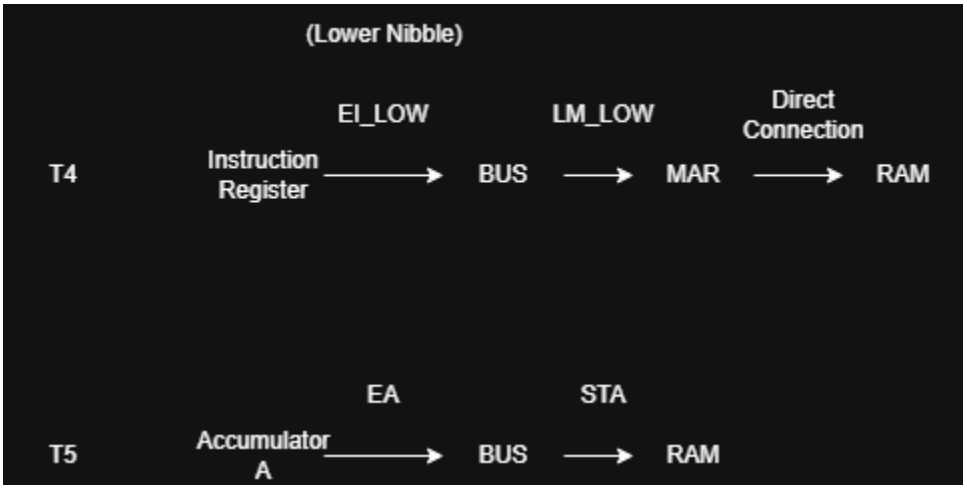

Figure 6: Illustration displaying store instruction timing states and controls

3.4. ADDI – Add Immediate

The add immediate instruction sends the lower nibble of the instruction as an immediate value to add with the current contents of accumulator A. On T4, the lower nibble of the instruction register is passed to the bus using signal EI_LOW, these lower bits are transferred from the bus to the B register using signal LIB which is directly connected to the adder/subtractor component. On T5, the output of this unit is passed to the bus using signal EU, these bits are transferred from the bus to accumulator A via signal LA_LOW. An illustration of the timing states and controls can be seen in Figure 8. The digital logic design for the add and subtract immediate instructions are like those needed for the load immediate above. There is an OR gate for the write enable of the lower nibble register as it is written to during signals LB_LOW or LIB. During an immediate add the upper nibble of the B register will always be 0 so LIB is connected to the clear of the upper nibble register. This digital logic implementation can be seen for both instructions in Figure 7.

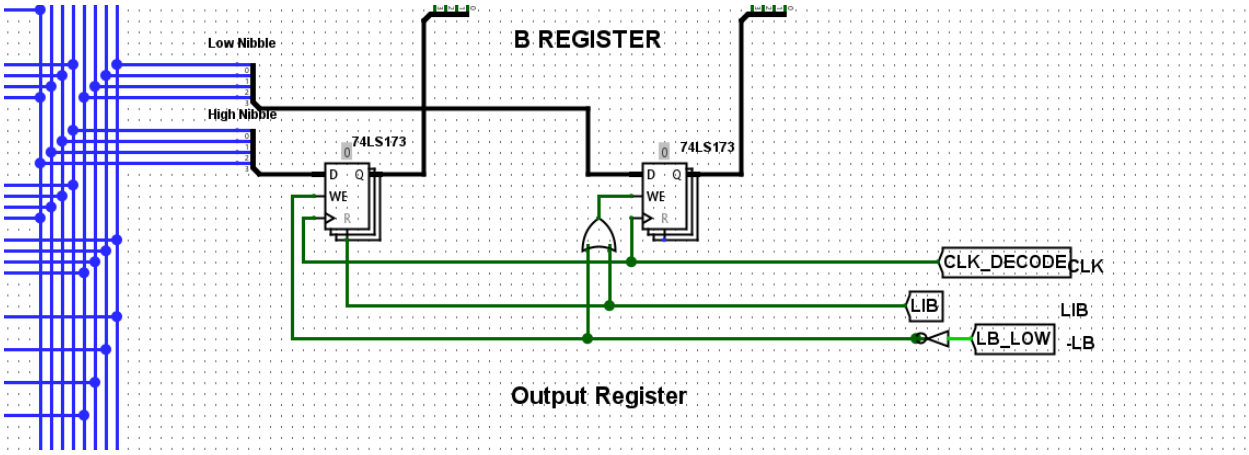

Figure 7: Logisim Diagram displaying digital logic implementation of sub and add immediate instructions

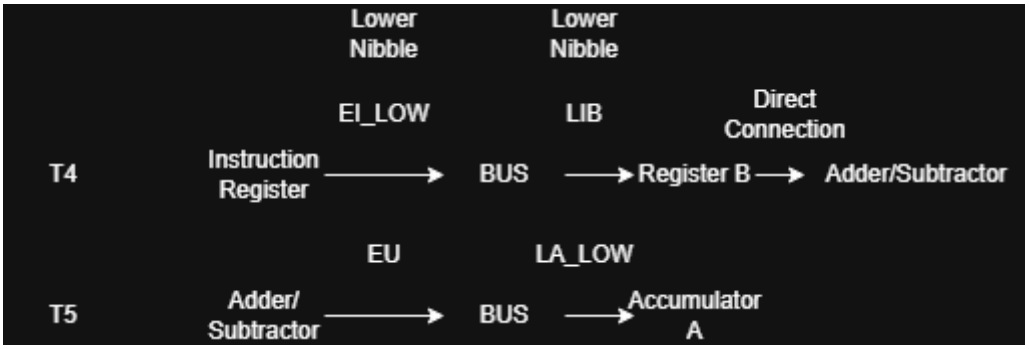

Figure 8: Illustration displaying add immediate instruction timing states and controls

3.5. SUBI – Subtract Immediate

The subtract immediate instruction sends the lower nibble of the instruction as an immediate value to subtract from the current contents of accumulator A. On T4, the lower nibble of the instruction register is passed to the bus using signal EI_LOW, these lower bits are transferred from the bus to the B register using signal LIB. On T5 the signal SU is enabled, this is to negate the value of b which is being passed to the adder/subtractor unit directly, the output of this unit is passed to the bus using signal EU, these bits are transferred from the bus to the accumulator via signal LA_LOW. An illustration of the timing states and controls can be seen in Figure 9, as well as the digital logic changes from SAP-1 in Figure 7.

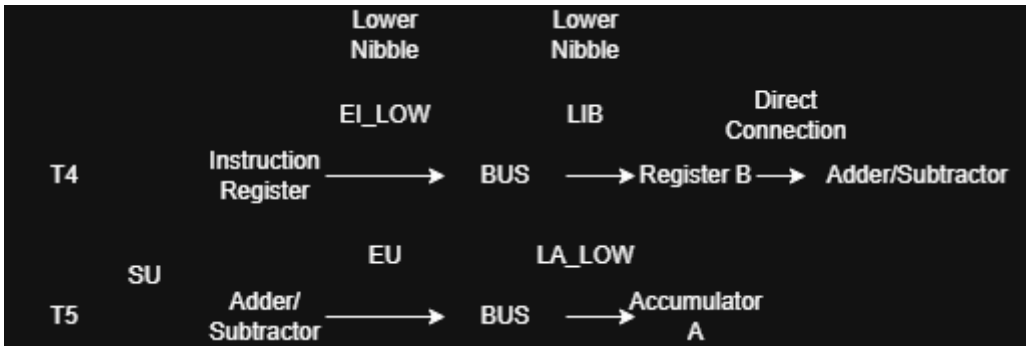

Figure 9: Illustration displaying sub immediate instruction timing states and controls

3.6. JMP – Unconditional Branch

The jump instruction functions as an unconditional branch to an address in memory using the lower nibble of the instruction, generally but not limited to being used as the next address in RAM to read instructions from. This instruction is extremely important in execution of large programs as it is the beginning of modular subroutine capability, this is a strong tool when combined with other memory management capabilities like the stack pointer. On T4, the lower nibble of the instruction register is passed to the bus via signal EI_LOW, the 4 lower bits are transferred to the program counter via signal JMP. An illustration of the timing state and associated controls can be found in Figure 11. Digital logic was implemented to set or reset each flip flop depending on the value of the bus, only setting or resetting when the signal JMP is enabled, this implementation can be seen in Figure 10.

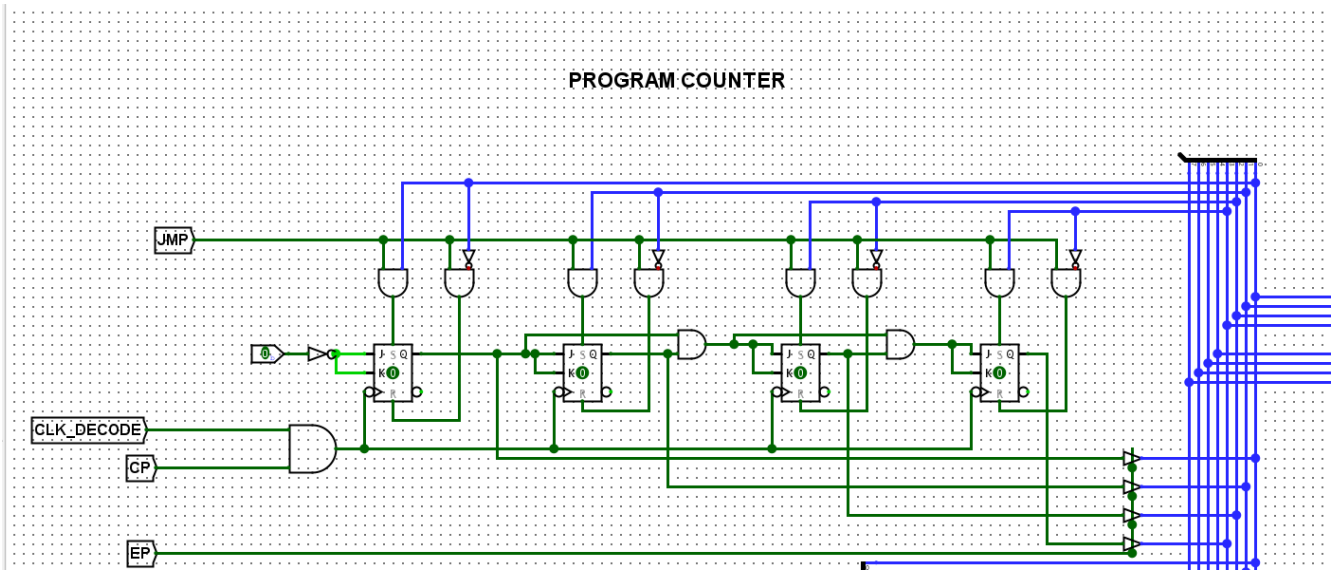

Figure 10: Logisim Diagram displaying digital logic implementation of jump instruction

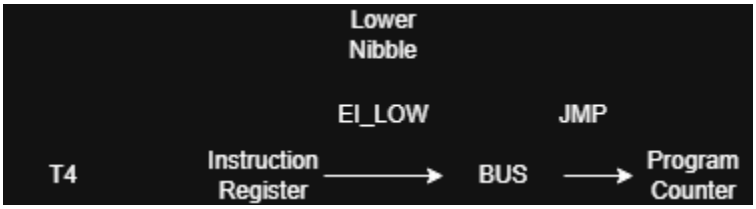

Figure 11: Illustration displaying jump instruction timing states and controls

4. Testing

Testing is crucial when completing any engineering project, in this case it verifies that additional instructions work with existing hardware and that the addition of new instruction or hardware do not disrupt previously tested instructions through regression testing. As each instruction was added to the NSC-8 they were tested individually, and previously

implemented instructions were re-tested as new ones were added. At the end of adding all instructions, example programs were tested which included a chain of instructions, the first included all original instructions in the SAP-1 architecture, the second program was created to test all new instructions to the architecture, the tables below showcase the programs. The respective scripts that can be directly entered into RAM can be found in the supporting documentation of the NSC-8 and can be seen in Table 2 and Table 3 below.

Table 2: Example program containing original instructions

Address (Hex)	**Instruction**	**Machine Code (Hex)**	**Binary (8-bit)**	**Explanation**
0H	LDA 9H	19	0001 1001	Load value from address 9H
1H	ADD AH	2A	0010 1010	Add value from address AH
2H	ADD BH	2B	0010 1011	Add value from address BH
3H	SUB CH	3C	0011 1100	Subtract value from address CH
4H	OUT	E0	1110 0000	Output accumulator value
5H	HLT	F0	1111 0000	Halt CPU
6H	XX	00 (Unused)	0000 0000	Unused
7H	XX	00 (Unused)	0000 0000	Unused
8H	XX	00 (Unused)	0000 0000	Unused
9H	Data	10	0001 0000	Value loaded by LDA
AH	Data	14	0001 0100	Value added to accumulator
BH	Data	18	0001 1000	Value added to accumulator

CH	Data	20	0010 0000	Value subtracted from accumulator

Table 3: Example program for newly added instructions

Address (Hex)	**Instruction**	**Machine Code (Hex)**	**Binary (8-bit)**	**Explanation**
0H	LDAI 4H	34	0011 0100	Load 4 into A
1H	ADDI 2H	52	0101 0010	Add 2 to A
2H	ADDI 1H	51	0101 0001	Add 1 to A
3H	SUBI 2H	62	0110 0010	Subtract 2 from A
4H	JUMP 9H	79	0111 1001	Jump to address 9
5H	XX	00 (Unused)	0000 0000	Unused
6H	XX	00 (Unused)	0000 0000	Unused
7H	XX	00 (Unused)	0000 0000	Unused
8H	XX	00 (Unused)	0000 0000	Unused
9H	STA FH	4F	0100 1111	Store A to address F
AH	OUT	EF	1110 1111	Output value in A
BH	HLT	FF	1111 1111	Halt program

5. Conclusion

The NSC-8 successfully extends the SAP-1 architecture with five new instructions that integrate seamlessly with existing hardware. Control logic and timing states were implemented and tested using Logisim. Future work may include deploying the architecture on an FPGA or adding conditional branching capabilities.

References

[1] A. P. Malvino and J. A. Brown, Digital Computer Electronics 3rd ed, New York: Glencoe/McGraw-Hill, 1993.