Project Description

The objective of the project is to design, build, and demonstrate a simple 4-bit computer processor that is capable of performing a number of arithmetic operations. This processor is able to load, add, subtract, divide by 2, increment by 1, and clear the values contained in the 4-bit register.

Project Requirements

The computer processor is built around a 4-bit register, w (with bits w3, w2, w1, and w0), and includes the necessary logic to perform several operations (computer instructions) on the contents of w and, possibly, external input data. The external 4-bit data is called L (for "literal"), these bits are represented by S3, S2, S1, S0 in this project. The operation performed during a given clock cycle is specified by 3 control inputs (an "instruction word"), X2, X1, X0. 2's complement data representation and math are to be used. The processor is to perform the operations (instructions) shown in Table 1 below.

Table 1

Instruction (Description)	Mnemonic	Notation
Add L to w (store in w)	ADDLW	w ← w + L
Subtract L from w (store in w)	SUBLW	w ← w – L
Increment w (store in w)	INCW	w ← w + 1
Arithmetic right shift w (store in w)	ASHRW	w ← ashr of w
Clear w	CLRW	w ← 0
Load L into w	MOVL	w ← L

In the construction of the processor, dip switches S3, S2, S1, and S0 are used for external inputs. Switches S7, S6, and S5 are used for control signals X2, X1, and X0, respectively. LEDS L3, L2, L1, L0 are used to display the corresponding bits of register w. LEDS L7, L6, L5, and L4 are used to display the external inputs. (w0 and S0 are the least significant bits of the two quantities) Push button PB1 is used as the manual clock. Each time the PB1 is pressed and released, the instruction specified by the current settings of X2, X1, and X0 is executed. The processor will derive a sequence of instructions that will computer a result for the following high-level expression: (x+1)/2 – y + z. The six instructions shown in Table 1 must all be used at least once in computing the result of the expression. The first instruction mnemonic to be executed must be 'CLRW." Register w will hold the result of the computation once the program has been executed. The values of x, y, and z that this processor will use an inputs are 2, 4, and 2 respectively. The result should be '1111' in bits or -1 numerically.

Design Approach

In designing the processor, the first thing that needs to be determined is the register-level building blocks that are required as well as their interconnections. The implemented layout shown later in Diagram 1 requires four 4-1 multiplexers, a 4-bit adder, a 4-bit register D flip-flop, nine NAND gates, and five inverters. The control signals used in this processor are shown on the following page in Table 2.

Table 2

Control Signal Operation	X2 switch	X1 switch	X0 switch
CLRW	0	1	0
MOVL	0	0	1
INCW	1	1	0
ASHRW	0	0	0
SUBLW	1	1	1
ADDLW	1	0	1

The processor works in the following explanation. Each external input bit value L is sent to one line of each of the 4 multiplexers. Its complement value is also sent to another line. One line of the multiplexer is grounder to always give an input of zero. The final line input of the multiplexer comes from the D flip flop 4-bit register. X1 and X0 are the switch selection lines for the multiplexers. Selection output is then sent to the 4-bit adder. The carry in input of the adder is implemented using switches X2 and X1. They are inputted to a NAND gate that is inverted so that, under a certain operation value, it will carry in a value of 1. The second adder input comes from the D flip flops in conjunction with a set of NAND gates that act as 2-1 multiplexers. This multiplexer has the ability to be set to all 0 or to the output already in the register. The 4-bit register receives all bit values for processing. A manual push button clock is on this register to perform each operation act at the user's decided time.

always show k-map for any logic you implement -1

Specific Specification of Processor Operation

There are six control instructions that are implemented in the processor. Each instruction performed in the processor is detailed below:

The 'CLRW' operation inputs '10' into the selection lines of the multiplexers (X1, X0). The input line into these multiplexers is grounded to zero, resulting in an output of zero from the multiplexers. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '01' results in an output of 0. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as an input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flop outputs. These inputs are selection lines, such that a value of zero is outputted to the second adder line of the 4-bit adder. The final operation is that it clears the register bits to 0s.

The 'MOVL' operation inputs '01' into the selection lines of the multiplexers (X1, X0). The input line into these multiplexers is set to a bit of each external input. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '00' results in an output of 0. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as an input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flops outputs. These inputs are selection lines, such that a value of zero is outputted to the second adder line of the 4-bit adder. The final operation is that it sets the register bits to the external input bits.

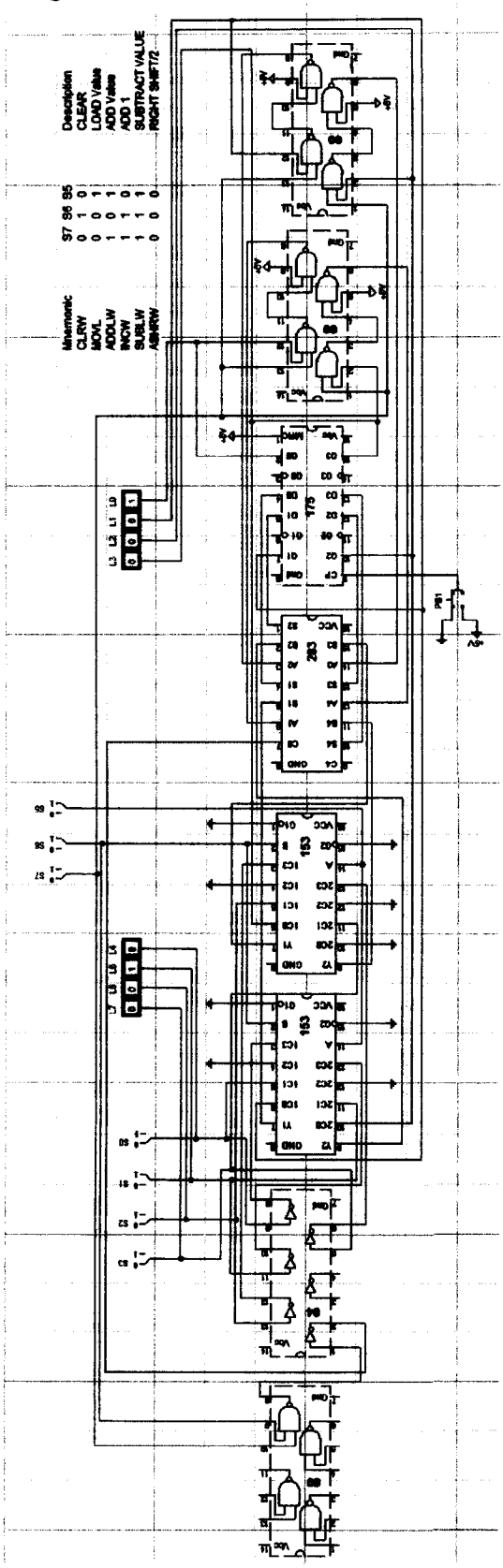
The 'INCW' operation inputs '10' into the selection lines of the multiplexers (X1, X0). This operation uses the same multiplexer input line as CLRW. The input line into these multiplexers is grounded to zero, resulting in an output of zero from the multiplexers. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '11' results in an output of 1. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flop outputs. These inputs are selection lines, such that in this case, the register value is selected to output to the second adder line of the 4-bit adder. The final operation is that it adds 1 to the register bits.

The 'ASHRW' operation inputs '00' into the selection lines of the multiplexers (X1, X0). The input line into these multiplexers is set to a bit of each of the register outputs. The register has bits 3, 2, 1, 0 therefore, bit 1 is outputted to bit 0 of the multiplexer input, bit 2 is outputted to bit 1, and bit 3 is outputted to bit 2. Bit 3 of the multiplexer input is grounded to 0. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '00' results in an output of 0. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flop outputs. These inputs are selection lines, such that in this case, the register value is selected to output to the second adder line of the 4-bit adder. The final operation shifts bits to the right, performing a divide by 2 operation.

The 'SUBLW' operation inputs '11' into the selection lines of the multiplexers (X1, X0). The input line into these multiplexers is set to a bit of each external input which has been inverted. This is because subtraction requires the addition of a complemented value and adding 1. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '11' results in an output of 1. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as an input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flops outputs. These inputs are selection lines, such that in this case, the register value is selected to output to the second adder line of the 4-bit adder. The final operation is that the bits in the register are added to the complemented external input form and added with 1. The operation of subtraction is performed.

The 'ADDLW' operation inputs '01' into the selection lines of the multiplexers (X1, X0). This operation uses the same multiplexer input line as MOVL. The input line into these multiplexers is set to a bit of each external input. Switches X2 and X1 are connected to a NAND gate and inverted, in this case the input of '10' results in an output of 0. This is used for the carry in for the 4-bit adder. The multiplexer output is inputted into each of the 4-bit adder inputs. The second input of the adder comes from the D flip-flop. Switch X2 is sent as an input to four 2-1 multiplexers. (Built from 8 NAND gates) The second input of these multiplexers comes from the D flip-flops outputs. These inputs are selection lines, such that in this case, the register value is selected to output to the second adder line of the 4-bit adder. The final operation is that the bits in the register are added to the external input.

Diagram 1


Mnemonic
CLRW
MOVL
ADDLW
INCW
SUBLW
ABWRW
S7 S6 S5
0 0 0
0 1 0
0 0 1
1 0 1
1 1 0
1 1 1
0 0 0
Description
CLEAR
LOAD Value
ADD Value
ADD 1
SUBTRACT VALUE
RIGHT SHIFT/2
283
153
153
175

Timing Data

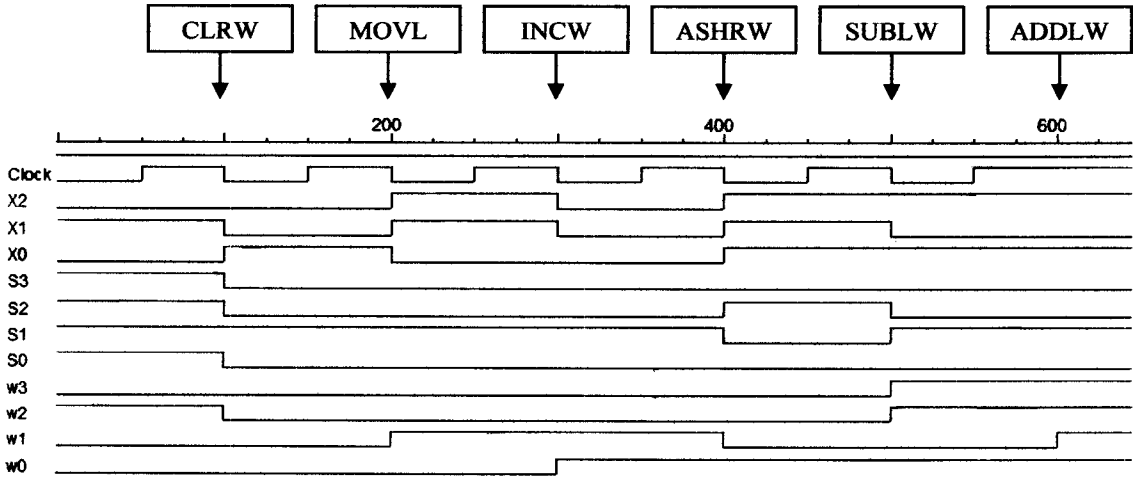
In Logic Works, having to toggle a large number of binary switches in a complex circuit can become tedious lead to error, or just end up taking a lot of time, especially if you loose your place and have to start over. Logic Works provides a means to automate the simulation of a circuit, through the use of timing files. A timing file is a way of telling the Simulator what signal you want to apply to a given input at a given point in time. Data 1 below shows a timing file. The first column represents the current point in time. Columns X0, X1, X2, represent the control signals used to select the appropriate logic operations. Columns S0, S1, S2, and S3 are the external inputs. S0 and X0 are the least significant bits of the two quantities. The last column represents the manual clock from which the control signals are implemented.

Data 1

$T	$O X0	$O X1	$O X2	$O S0	$O S1	$O S2	$O S3	$O Clock
0	0	1	0	1	1	1	1	0
50	0	1	0	1	1	1	1	1
100	1	0	0	0	1	0	0	0
150	1	0	0	0	1	0	0	1
200	0	1	1	0	1	0	0	0
250	0	1	1	0	1	0	0	1
300	0	0	0	0	1	0	0	0
350	0	0	0	0	1	0	0	1
400	1	1	1	0	0	1	0	0
450	1	1	1	0	0	1	0	1
500	1	0	1	0	1	0	0	0
550	1	0	1	0	1	0	0	1

Data 1 represents a numerical look at the signals being processed and implemented in the processor. Below in Data 2, a visual time graph of Data 1 is shown. The following marks below show the points at which the output signals are fully implemented, using the specific control signals. In addition to Data 2, Blue represents a logic signal of 0 and Red represents a signal of 1.

Data 2



In Data 2, the clock alternates between 0 and 1. The clock needs to fall and rise for each successful operation in the timing file. Therefore, for every change in external inputs S and X, Clock must change twice. The 4-bit register is represented by w3, w2, w1, and w0. The least significant bit is w0 in the registers. Data 2 represents the sequence of instructions that will computer a result for the following high-level expression: (x+1)/2 – y + z. (Where x = 2, y = 4, and z = 2) At the point 'CLRW', bits in the 4-bit register are cleared to '0000'. 'MOVL' loads a number into the registers, in this case '0010' (2) is loaded. 'INCW' adds 1 to the value within the registers. 'ASHRW' shifts the bits to the right, which divides the number in the registers by 2. The register is now '0001' (1). Next 'SUBLW' performs the subtraction of 1 – 4, which results in '1101' (-3). The final operation 'ADDLW' adds 2 to the number contained in the register. Therefore, the result is '1111' (-1).

Time Delay Analysis

The time delay analysis is the worst-case time delay of the circuit for the execution of each of the 6 instructions used in the processor. In each delay calculation, the individual gate delays are shown below in Data 3.

Data 3

IC TYPE	Delay (ns)
7400	10
7404	10
74153	22
74175	15
74283	16

The worst-case time delay fixes the minimum time that the overall system must allow for an instruction to complete – thus determining the maximum number of instructions that the computer can execute per unit time. The 'CLRW', 'MOVL', 'ADDLW', 'INCW', and 'ASHRW' delays include the following chips: 74153, 74175, and a 74283. Each has a resulting delay of 22 ns + 15 ns + 16 ns = 53 ns. The 'SUBLW' delay includes the following chips: 7404, 74153, 74175, and a 74283. Its resulting delay is 10 ns + 22 ns + 15 ns + 16 ns = 63 ns. It has a larger delay because its input has to be inverted, requiring a 7404 chip.

Evaluations, Conclusions, and Observations

The implementation used in this project is not the only one. There are probably several other ways to implement the processor. Time delay could be decreased, fewer chips could be used, and the processor could cost less to build. The time delay seems to be as minimum as possible in this processor. It was easy to see when implementing my processor that the carry in bit would require a separate NAND gate connected to two switch inputs to determine when a 1 should be carried in. The rest of the design was fairly straight forward as has been described. The first design just happened to be the final design. Granted the processor was fairly complex, it took an adequate amount of time to build.