Experiment 5: Logic and Bit Instructions, Hex Keypad, Stepper Motor

Instructional Objectives:

- To implement logic, bit, and shift instructions to use more complicated I/O peripherals.
- To use bit test instructions to change the output of an I/O peripheral given an input.
- To learn how to properly debounce inputs from mechanical switches.

Introduction:

The following procedure will introduce students to logic and bitwise operations that are necessary to implement more complex I/O peripherals, such as the keypad, and learn how to use these instructions to set and clear bits and implement *if-else*, and *while* structures.

Logic, Shift, and Rotate Instructions:

Table 5-8. Boolean Logic Instructions

Mnemonic	Function	Operation
ANDA	AND A with memory	(A) • (M) ⇒ A
ANDB	AND B with memory	(B) • (M) ⇒ B
ANDCC	AND CCR with memory (clear CCR bits)	(CCR) • (M) ⇒ CCR
EORA	Exclusive OR A with memory	(A) ⊕ (M) ⇒ A
EORB	Exclusive OR B with memory	(B) ⊕ (M) ⇒ B
ORAA	OR A with memory	(A) + (M) ⇒ A
ORAB	OR B with memory	(B) + (M) ⇒ B
ORCC	OR CCR with memory (set CCR bits)	(CCR) + (M) ⇒ CCR

Logic instructions are instructions that allow for logic operations to be performed between an 8-bit register and the contents of a memory location or immediate value. These instructions allow for individual bits to be set or cleared to start or end events, or to set or clear specific bits of an output port. In addition, individual bits can be checked to determine the status of individual bits of an input port.

An `AND` instruction allows for specific bits to be cleared. To clear or mask a specific bit, perform an `AND` operation with a `0` on that specific bit. To check the status of a specific bit, mask off unnecessary bits and examine only specific bits. See the example below.

```
LP:   LDAA  PTT   ;This instruction loads accumulator A with the contents of Port T
      ANDA  #$08  ;This instruction masks bits 0-2 and bits 4-7 of Port T
      BNE   LP    ;This instruction branches back to LP if bit 3 of Port T is set
```

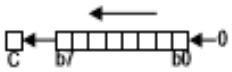
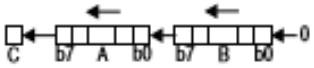
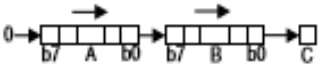
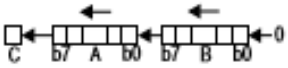
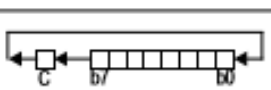
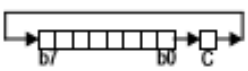

An `OR` instruction allows for specific bits to be set. To set a specific bit, perform an `OR` operation with a `1` on that specific bit. See the example below.

```
LP:   LDAA  PTS   ;This instruction loads accumulator A with the contents of Port S
      ORAA  #$10  ;This instruction sets bit 4 of Port S while leaving the other bits alone
      STAA  PTS   ;This instruction stores the new value back to Port S
```

An `XOR` instruction allows for specific bits to be toggled. To toggle a specific bit, perform an `XOR` operation with a `1` on that specific bit. See the example below.

```
LP:   LDAA  PTS   ;This instruction loads accumulator A with the contents of Port S
      EORA  #$AA  ;This instruction toggles bits 1,3,5,7 of Port S while leaving the other bits alone
      STAA  PTS   ;This instruction stores the new value back to Port S
```

Table 5-12. Shift and Rotate Instructions

Mnemonic	Function	Operation
Logical Shifts		
LSL LSLA LSLB	Logic shift left memory Logic shift left A Logic shift left B	
LSLD	Logic shift left D	
LSR LSRA LSRB	Logic shift right memory Logic shift right A Logic shift right B	
LSRD	Logic shift right D	
Arithmetic Shifts		
ASL ASLA ASLB	Arithmetic shift left memory Arithmetic shift left A Arithmetic shift left B	
ASLD	Arithmetic shift left D	
ASR ASRA ASRB	Arithmetic shift right memory Arithmetic shift right A Arithmetic shift right B	
Rotates		
ROL ROLA ROLB	Rotate left memory through carry Rotate left A through carry Rotate left B through carry	
ROR RORA RORB	Rotate right memory through carry Rotate right A through carry Rotate right B through carry	

Shift and rotate instructions simultaneously shift all bits either right right or left. Shift instructions can be used to double a value in an accumulator by shifting to the left or halving a value in an accumulator by shifting to the right. Logical shift instructions are for unsigned values while arithmetic shift instructions are for signed values. Rotate instructions can be used to check each and every bit of a register sequentially. To do this, shift and check the carry flag of the CCR using `BCC` or `BCS`. See the example below.

```
      LDAA  PTT    ;This instruction loads accumulator A with the contents of Port T
      LDAB  #0     ;This instruction initializes a counter in accumulator B
LP:   INCB         ;This instruction increments the counter by 1
      ROLA         ;This instruction performs a rotate operation on Port T
      BCC   LP     ;This instruction branches back to LP if the nth bit of Port T is clear
                   ;Accumulator B will have the value of the rightmost switch that is on
```

Bit Test, Manipulation, and Condition Branch Instructions:

Table 5-11. Bit Test and Manipulation Instructions

Mnemonic	Function	Operation
BCLR	Clear bits in memory	$(M) \bullet (\overline{mm}) \Rightarrow M$
BITA	Bit test A	$(A) \bullet (M)$
BITB	Bit test B	$(B) \bullet (M)$
BSET	Set bits in memory	$(M) + (mm) \Rightarrow M$

In addition to logic operations, there are also instructions that bit test instructions that can check a bit of a register without changing that specific bit of the destination register. There are also bit modify instructions that can set or clear specific bits in memory. See the example below.

```
MyConstants:    SECTION
DDRSMASK    EQU    $FF
MASKBIT0    EQU    $01
MASKBIT1    EQU    $02

MyCode:         SECTION
Entry:BSET  PTS,#DDRSMASK ;This instruction initializes all pins of Port S as outputs
LP:   LDAA  PTT    ;This instruction loads accumulator A with the contents of Port T
      BITA  #$01   ;This instruction checks the status of switch 1 and only switch 1
      BEQ   LED1   ;This instruction branches if switch 1 is low
      BITA  #$02   ;This instruction checks the status of switch 2 and only swtich 2
      BEQ   LED2   ;This instruction branches if switch 2 is low
      BRA   LP     ;This instruction branches back to LP to check the switches again
LED1: BCLR  PTS,#MASKBIT0 ;This instruction clears the rightmost LED (bit 0 of Port S)
      BRA   LP     ;This instruction branches back to LP to check the switches again
LED2: BCLR  PTS,#MASKBIT1 ;This instruction clears the next LED (bit 1 of Port S)
      BRA   LP     ;This instruction branches back to LP to check the switches again
```


Table 5-19. Bit Condition Branch Instructions

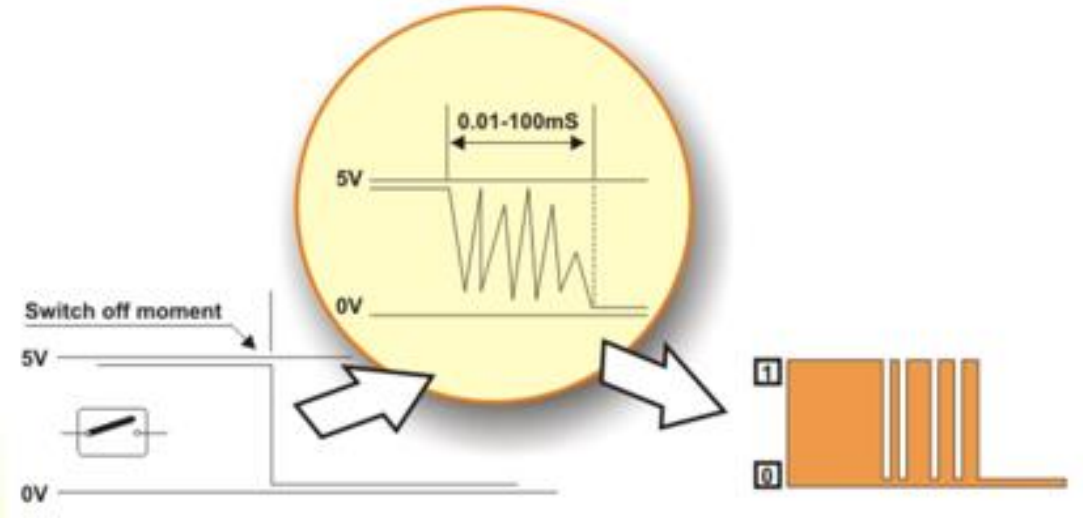
Mnemonic	Function	Equation or Operation
BRCLR	Branch if selected bits clear	(M) • (mm) = 0
BRSET	Branch if selected bits set	$(\overline{M})$ • (mm) = 0

There are also bit condition branch instructions that allows for the program to check the specific value of a memory location or port and branch if the necessary condition is met. The following code is identical to the example above.

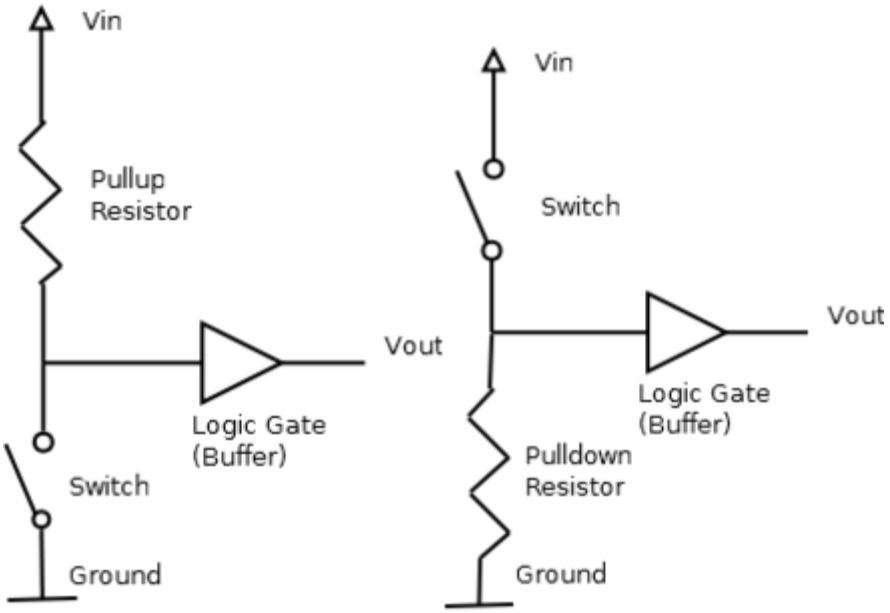
```
MyConstants:    SECTION
DDRSMASK    EQU     $FF
MASKBIT0    EQU     $01
MASKBIT1    EQU     $02

MyCode:         SECTION
Entry:BSET  PTS,#DDRSMASK ;This instruction initializes all pins of Port S as outputs
LP:   BRCLR PTT,MASKBIT0,LED1 ;This instruction branches if switch 1 is low
      BRCLR PTT,MASKBIT1,LED2 ;This instruction branches if switch 2 is low
      BRA   LP    ;This instruction branches back to LP to check the switches again
LED1: BCLR  PTS,#MASKBIT0 ;This instruction clears the rightmost LED (bit 0 of Port S)
      BRA   LP    ;This instruction branches back to LP to check the switches again
LED2: BCLR  PTS,#MASKBIT1 ;This instruction clears the next LED (bit 1 of Port S)
      BRA   LP    ;This instruction branches back to LP to check the switches again
```

Switch Debouncing, Pull up/down devices, and the Hex Keypad

All mechanical contacts bounce when making and breaking connections, including mechanical switches and the pushbuttons used for the keypad on the lab boards. Bouncing means that for one switching event, several changes may be noticed by the CPU. See the figure below.



To deal with bouncing, it is required to add a routine to debounce the value of the mechanical switch. This involves adding a short software delay that delays until the bouncing subsides whenever a change in the switch is detected for the first time. For most switches, a one millisecond delay is sufficient.



Inputs can be configured to function as a pull up device or a pull down device. For a pull up device, when the switch is open, the value of Vout will equal Vin. When the switch is closed, the value of Vout will equal zero. For a pull down device, when the switch is open, the value of Vout will equal zero. When the switch is closed, the value of Vout will equal Vin.

3.3.7.5 Port U Pull Device Enable Register (PERU)

	7	6	5	4	3	2	1	0
R W	PERU7	PERU6	PERU5	PERU4	PERU3	PERU2	PERU1	PERU0
Reset	0	0	0	0	0	0	0	0

Figure 3-46. Port T Pull Device Enable Register (PERT)

Read: Anytime. Write: Anytime.

This register configures whether a pull-up or a pull-down device is activated on configured input pins. If a pin is configured as output, the corresponding Pull Device Enable Register bit has no effect.

Table 3-33. PERT Field Descriptions

Field	Description
7:0 PERU[7:0]	**Pull Device Enable Port U** 0 Pull-up or pull-down device is disabled. 1 Pull-up or pull-down device is enabled.

3.3.7.6 Port U Polarity Select Register (PPSU)

	7	6	5	4	3	2	1	0
R W	PPSU7	PPSU6	PPSU5	PPSU4	PPSU3	PPSU2	PPSU1	PPSU0
Reset	0	0	0	0	0	0	0	0

Figure 3-47. Port U Polarity Select Register (PPSU)

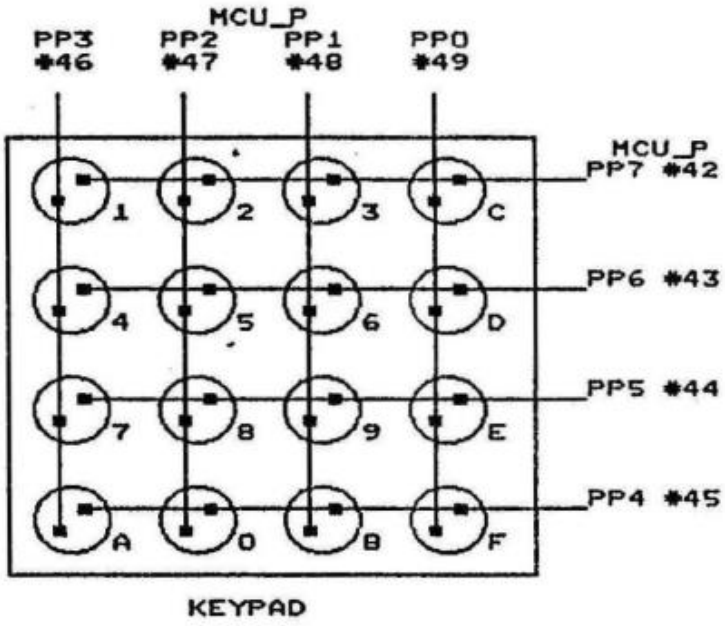
Read: Anytime. Write: Anytime.

The Port U Polarity Select Register selects whether a pull-down or a pull-up device is connected to the pin. The Port U Polarity Select Register is effective only when the corresponding Data Direction Register bit is set to 0 (input) and the corresponding Pull Device Enable Register bit is set to 1.

Table 3-34. PPST Field Descriptions

Field	Description
7:0 PPSU[7:0]	**Pull Select Port U** 0 A pull-up device is connected to the associated port T pin. 1 A pull-down device is connected to the associated port T pin.

In addition to the Data Direction Register (DDR), each I/O port has a Pull Device Enable Register (PER) and a Polarity Select Register (PPS). The Pull Device Enable Register enables a pull-up or pull-down device. To enable a pull device on an input pin of an I/O port, write a `1` to the corresponding bit of the port's PER. The Polarity Select Register decides whether the pull device is pull-up or pull-down. To make a pull device a pull-up, write a `0` to the corresponding bit of the port's PPS. To make a pull device a pull-down, write a `1` to the corresponding bit of the port's PER.



Above is the keypad used in Laboratory 5.2. Since there are sixteen keys on the keypad, it is not possible to read each single distinctively as an input on an 8-bit I/O port. To efficiently read the sixteen keys, the lower nibble (bits 0-3) of Port U (at address `$268`) are connected to the four columns of the keypad and are configured as inputs. The upper nibble (bits 4-7) of Port U are connected to the four rows of the keypad and configured as outputs. The bits of the Pull Up Enable Register of Port U (at address `$26C`) that correspond to the input pins of Port U must be set to enable pull devices on that pin. If using the inputs of the keypad as a pull-up device, write a `0` to the corresponding bits of the Polarity Select Register of Port U (at address `$26D`). If using the inputs of the keypad as a pull-down device, write a `1` to the corresponding bits of the Polarity Select Register of Port U.

Reading the keypad is a two-step process. First, continuously sent a sequence to Port U to check each row of the keypad. For a pull-up configuration, the first sequence value to read row 1 is `%01110000=$F0`. For a pull-down configuration, the first sequence value is to read row 1 is `%10000000=%80`. After a sequence value is written to the keypad, debounce and read the value back from Port U.

In between each sequence value, read back the value from Port U. The value in the lower nibble of Port U will determine if a key is pressed or not. For example, for a pull-up configuration, the value on the lower nibble corresponding with the four columns of the keypad will read `%0111`. For a pull-down configuration, the value on the lower nibble will read `%1000`. If no key is pressed, for a pull-up configuration, the value on the lower nibble corresponding with the four columns of the keypad will read `%1111`. For a pull-down configuration, the value on the lower nibble will read `%0000`. In that case, the next sequence value must be written to the upper nibble of Port U.

After writing a value, the program must wait for the key to be released before going back to scan for the next key. This is because the E-Clock of the HC(S)12 is so fast, that multiple keypresses can be read as one press. This problem can be overcome by debouncing and then checking the lower nibble of Port U to see if a key has been released.

Note: Save the keypad routine from Laboratory 5.2 step 2 below. This routine will be used again for the final project.

Experimental Procedure:

Laboratory 5.1.1: Logic Instructions

1. Write a program which will read variable `VAR_1` (with initial value set to `$EC`) into accumulator A, then clear bit 6 of `VAR_1` (and only bit 6) and then store this value back into `VAR_1`. Use a logic operation to clear the bit. Verify that the program works properly. `VAR_1` must be initialized each time the program is run (either in the assembly code or by editing the value in the debugger).

2. Add code to the above program to set bit 4 of `VAR_1` and store it back in `VAR_1`.

3. Using an `AND` instruction and the appropriate branch instruction add code to the above program which will wait for bit 1, and only bit 1 (switch 2) of the dipswitches at Port T (address `$240`). When switch 2 goes low, send VAR_1 to the LEDs at Port S (at address `$248`)

4. Modify the above program to wait for switch 2 to go high after going low.

Laboratory 5.1.2: Bit Instructions

1. Using the `BSET` instruction, initialize the DDR for Port S (address `$24A`).

2. Using the `BCLR` instruction, turn off the LEDs at Port S (address `$248`).

3. Using either the `BRSET` or `BRCLR` instruction, write a program that waits for switch 2 (bit 1) to go low, then high, then lights LED 6 (Using either the `BSET` or `BCLR` instruction).

4. Put a `NOP` instruction at the end of the program. Set a breakpoint to stop the program after the wait loop.

Laboratory 5.2: Hex Keypad

1. Initialize Port U (address `$268`) so the upper four bits are outputs and the lower four bits are inputs.

    - Write value `$F0` to the Data Direction Register of Port U (address `$26A`) to make bits 0-3 inputs and bits 4-7 outputs.

    - Write value `$F0` to the Polarity Select Register of Port U (address `$26D`) to set the pins 0-3 as a pull-up device.

    - Write value `$0F` to the Pull Device Enable Register of Port U (address `$26C`) to activate the pull-up device on pins 0-3.

2. Check each row of the keypad by writing a logic low to the output pin of Port U connected to that row and a logic high to the other output pins of Port U. Then read back the value and check for a match.

    - The sequence that needs to be sent to Port U continuously to scan all four rows of the keypad is `$70, $B0, $D0, $E0`.

    - After sending a value from the sequence to Port U, wait for Port U to debounce (use a 1 millisecond delay), then read back the value in Port U.

    - If no key in that row is pressed, value `$xF` will be read from the lower nibble (bits 0-3) of Port U. Use an `AND` instruction to mask the upper nibble (bits 4-7) of Port U. If this value is read, branch back and scan the next row using the sequence above.

    - If a key is pressed use the lookup table routine from Laboratory 4.2 to check for a match. The index value of this sequence corresponds to the key that is being pressed (i.e., if a two is pressed on the keypad, the upper nibble with read value `%0111` and the lower nibble will read `%1011`). If a key is selected, send the index value to Port S (address `$248`). This value should be displayed on the lower nibble of the LEDs (bits 0-3 of Port S), and nothing should be read on the upper nibble of the LEDs (bits 4-7 of Port S). **THE LEDS SHOULD NOT FLICKER!!!**

    - After sending the value, wait for the key to be released before returning to scan the keypad. To do this, wait for Port U to debounce, then read the value from Port U and use an `AND` instruction to mask the upper nibble. If value `$xF` is read from the lower nibble, then the key has been released and the program should branch back to read the four rows of Port U continuously again. If any other value is read

from Port U, then the key has not been released and the program should wait for Port U to debounce again.

3. Alter the code above so that the index value is shown on the left four LEDs while keeping the right four LEDs off. This can be accomplished by using a shift or rotate instruction.

4. Alter the code above so that the first keypress value displays on the right four LEDs, the second keypress displays on the left four LEDs while maintaining the value of the first keypress, the third keypress displays on the right four LED while maintain the value of the second keypress, and so on. This can be accomplished using `AND`, `OR`, and shift instructions.

5. A subroutine **MUST BE** used for the debounce delay. In addition, the main keypad subroutine **MUST BE** contained in its own subroutine.

Extra Credit: perform the following two tasks:

1. Configure the keypad to function as a pull-down device instead of a pull-up device.

2. Not counting pushing and pulling registers to the stack, or the lines of code executed by a subroutine, write the keypad in 20 lines of code or less.

Laboratory 5.3: Stepper Motor

1. Modify the code from Laboratory 4.4 to add direction and speed control. The program should read the value from the dipswitches at Port T (address `$240`).

    - If switch 1 (bit 0) and switch 2 (bit 1) are both high or both low, the motor should not turn.

    - If switch 1 is low, the motor should turn clockwise. If switch 2 is low, the motor should turn couter-clockwise.

    - Give the motor two speeds. If switch 3 (bit 2) is low, the motor should turn slowly. If switch 3 is high, the motor should turn fast.

2. For the speed, use a delay of 15 milliseconds between each step of the sequence for the fast speed and a delay of 60 milliseconds between each step for the slow speed. Check switch 3 and load a 16-bit register with the count to generate a proper delay. An *if-else* structure can perform this task. The same delay subroutine must be used for both speeds.

3. For the direction, put the sequence from Laboratory 4.4 into an array and use an index register (X or Y) to step through the data. This pointer will track the motor sequence value even if the direction changes. If the motor is to step counter-clockwise, decrement the index register as the sequence values are being sent to Port P (address `$258`). If the motor is to step clockwise, increment the index register as sequence values are being sent. The best way to do this is to place a terminator at each end of the sequence. If the terminator value is read, reset the index register to point to the end of the sequence if turning counter-clockwise, or at the beginning of the sequence if turning clockwise, and then decrementing or incrementing the index register accordingly.

