SUBSYSTEMS

8-bit Computer Simulator

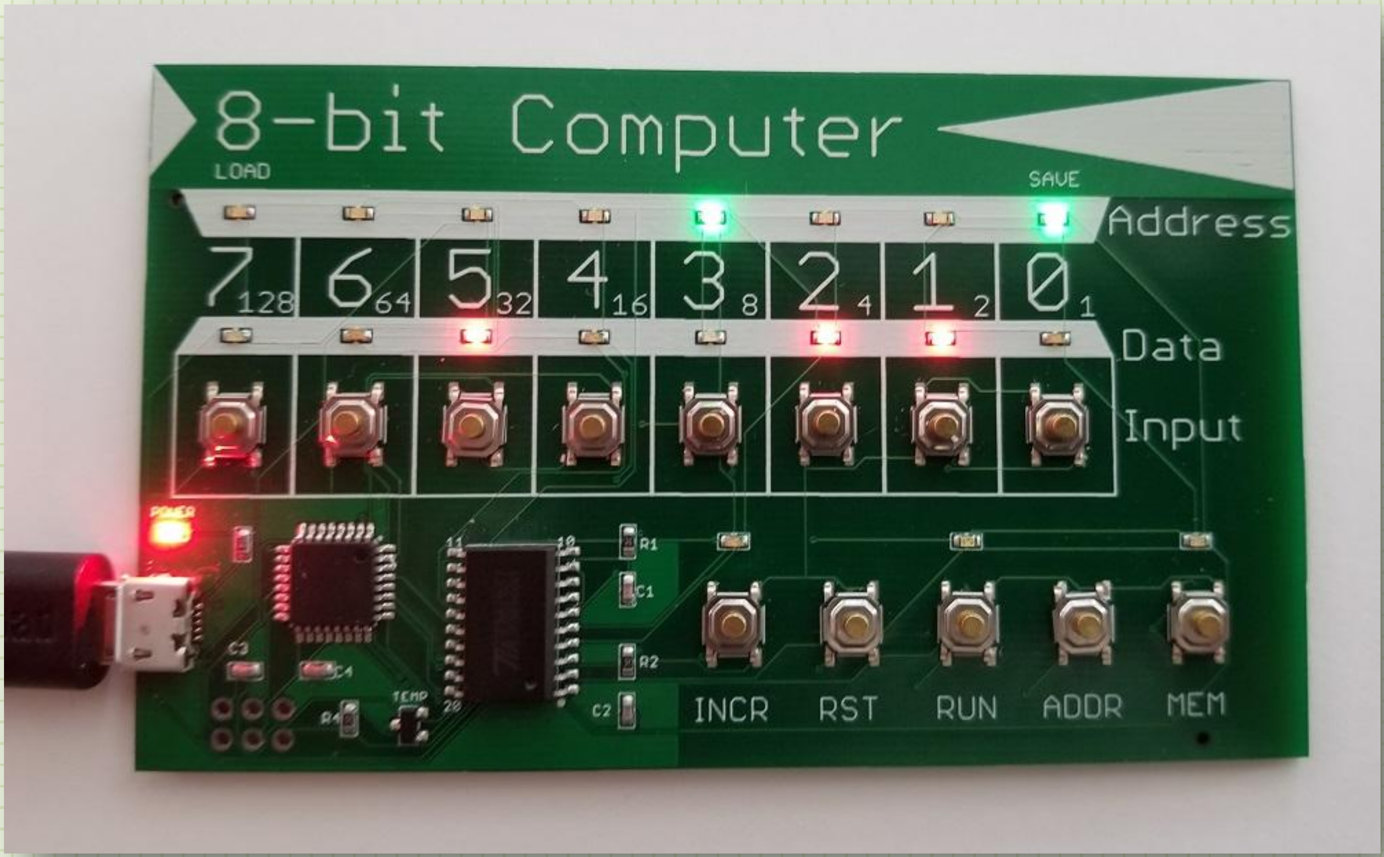


Table of Contents

Intro

8-bit computing

The language of computers today is far more evolved from where they began. Still, computers work fundamentally the same. They still just process 1s and 0s. This kit gets you back to the days of imputing programs by entering the instructions and data directly in memory.

Objectives

1) This instruction is intended to get you up and operating the computer. For an in depth curriculum on machine programming, see my 4-bit Microprocessor kit on Tindie.com and download the curriculum from the product page.

Section

1

Operating the Computer

The following are the basic instructions on key operation and how to run programs on the Computer simulator board.

Powering the Board

To power the computer board, use either a micro USB cable and plug it into a computer USB port or a phone charger. If plugged into a computer, it will only use the USB for power and there is no USB communication with the board. When power is applied, a red LED next to the USB port will light to show power is being supplied.

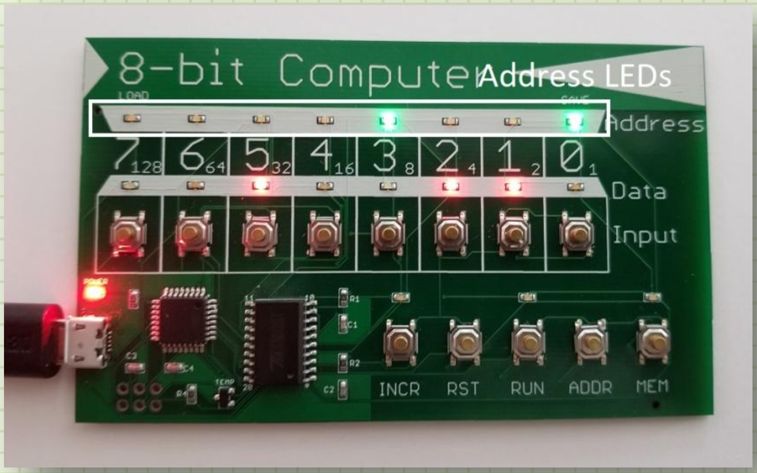
Basic Input / Output

The computer uses 2 sets of switches and 1 sensor to receive input. The temperature sensor is located on the bottom of the board and is labeled TEMP. It sends an analog signal to the computer which then makes reading available through the instruction set. It also has a row of LEDs for the DATA bus value and the ADDRESS value.

Below is a description of the Input / Output:

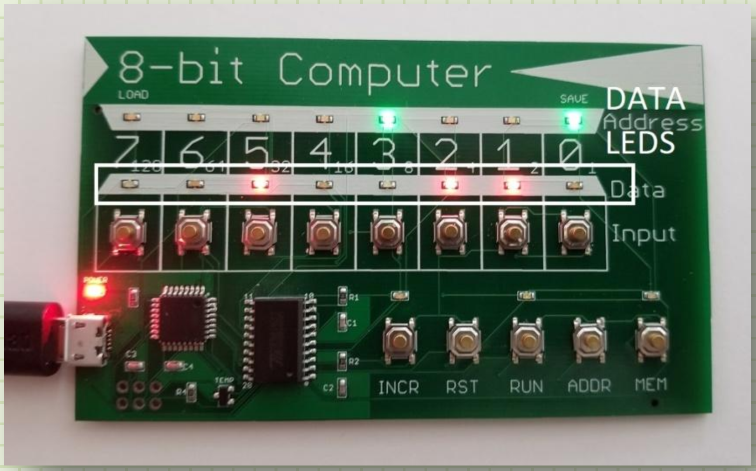
Address LEDs

The Address LEDs show the 8-bit representation of the current address. This is true whether the computer is in PROGRAM mode or RUN mode. The bit 0 and bit 7 LEDs also are used during program save and load operations (discussed later in the section for the MEM switch).



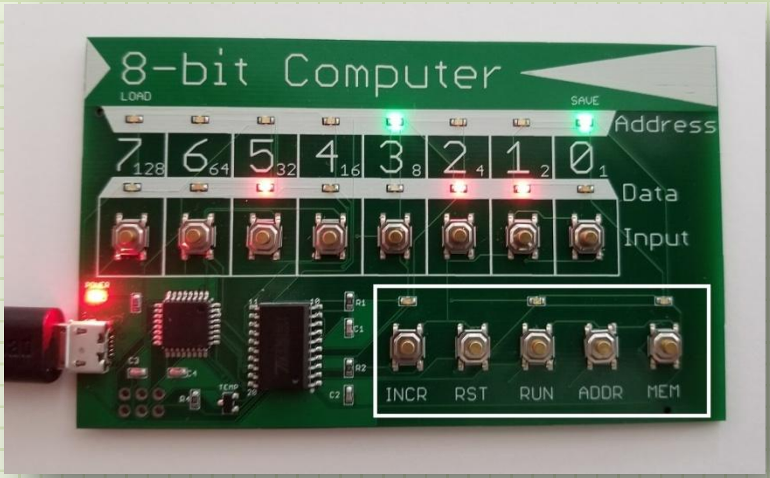
Data LEDs

In PROGRAM mode, these LEDs will show the current status of the DATA value at the current address or the new value you wish to load into the current address. During RUN mode, they are used as an output for displaying numeric and visual data.



INCR, RUN, and MEM LEDs

These special purpose indications will be discussed in their respective switch sections below.



Input Switches Data 0-7

These switches are used to create HIGH and LOW bit inputs to the computer. When programming, these are used to set the 8-bit binary number to be stored in the current memory location. Pressing these switches in PROGRAMMING mode will toggle the associated Data LED. Off is 0 (or LOW). On is 1 (or HIGH). These switches are also used in RUN mode to enter data into the computer during run time. Switch D0 and D1 also are used to select which bank of memory to save or load a program to or from.

Reset (RST)

The reset switch will return the program counter to 0 and display the current data located in that memory location. It is also used to break the operation of a currently running program.

Increment (INCR)

The INCR switch is used during PROGRAMMING mode to take the current value of the DATA and load it into the current memory location. After it saves the data, it advances to the next memory location. Programming is usually accomplished by resetting the computer (this will bring it to memory location 0). Enter the instruction or data using the DATA switches. Press the INCR switch to save this and move to the next location. Enter the next line of data. Press INCR, etc. Do this until you have completed entering all the instructions in memory. Remember to press increment for that final memory location. Just entering it into the DATA bus doesn't load it until INCR is pressed.

Run (RUN)

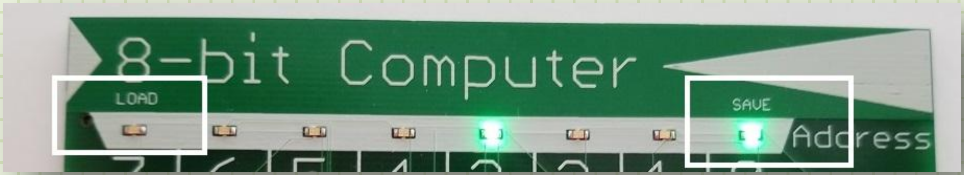
RUN will run a program from the current address. Normally, you will RESET before you RUN. This will ensure that you will start the program from memory location 0. But this feature allows you to run a program from any starting address.

Address (ADDR)

The ADDR switch will move the value of the DATA line into the Program Counter. This allows you to jump to any location in the 256 byte memory location. Enter the address you want into the DATA line using the data switches and press ADDR. The memory pointer 9program counter) will move to this address and display the contents. This is helpful to jump around your code to make changes or to start program execution from a specific address.

Memory (MEM)

The memory button allows you to load a program from memory or store a program to memory. There are two 256 byte memory banks to store programs. 256 bytes is the entire contents of the memory available for a program (address 00000000 – 11111111). Above the bit 7 address LED you will see the word "LOAD" and the word "SAVE" above the bit 0 LED.



<u>To Save Your Current 256 byte Program Space:</u>

1) Press the MEM button
2) Watch as the Address LEDs cycle down to the bit 0 LED

3) Verify the LED is lit under the word “SAVE” on the board
4) Choose which of the 2 memory banks to store your program (0 or 1) by pressing either the bit 0 or bit 1 Data switch. The corresponding DATA LED will light.
5) Verify the MEM green LED is lit and press the MEM switch to save the program

<u>To Load a Previously Saved Program from Storage:</u>

1) Press the MEM button
2) Watch as the Address LEDs cycle down to the bit 0 LED
3) Press the MEM switch again and watch the Address LEDs cycle up to the bit 7 LED
4) Verify the LED is lit under the word “LOAD” on the board
5) Choose which of the 2 memory banks to load your program from (0 or 1) by pressing either the bit 0 or bit 1 Data switch. The corresponding DATA LED will light.
6) Verify the MEM green LED is lit and press the MEM switch to load the program from memory into the current program RAM

Section

2

The Programming Resources

To understand the instruction set in the next section you have to first know the internal structure of the simulated computer. Our computer has the following:

Accumulators A and B (ACCA, ACCB)

These are general purpose 8-bit memory locations used to store data, perform arithmetic operations, and accept input.

Index Registers Y and Z (REGY, REGZ)

These are memory locations that hold an 8-bit address that can then be used to access and perform operations on memory.

Program Counter (PC)

The program counter holds the address of the next instruction to be executed.

Temperature Sensor

There is a temperature sensor on the board hooked up to the computers analog input port. Instruction TEMP loads the voltage value from the temperature sensor into ACCA. The small chip labeled “TEMP” on the board is the sensor. The value is proportional to the temperature but does not read out in a particular temperature unit.

The Timer

There is a timer that starts at zero when the computer runs a program and runs continuously up to 255 then recycles to 0. You can reset this Timer or load its value into ACCA. It returns the elapsed time in seconds.

Condition Codes

Condition codes are bits that are set or cleared based on the result of the instruction that was just executed. They are typically used to compare two numbers or jump to a different parts of the program dependant on a comparison.

Our computer has the following condition codes:

Carry/Overflow (ccC) – This is TRUE if the result of the operation is greater than 255.

Zero (ccZ) – This is TRUE if the result of the operation is zero.

Negative (ccN) – This is TRUE if the result of the operation is negative.

Memory Space

Program Memory: Our computer has 256 bytes of memory for a program and data. Memory locations 254 and 255 are also used as the 2 byte output of a multiply instruction.

Permanent Memory: There are two 256 byte storage locations to store the full program memory space. See MEM above for a description on how to LOAD and SAVE programs.

Section 3

The Instruction Set

The computer instruction set is comprised of typical instructions found in microprocessors and microcontrollers. It is intended to learn about the internal workings of computers and is not all inclusive. It is robust enough to give someone the basic knowledge of machine language programming or give the seasoned programmer a fun, nostalgic outlet.

Instructions by function

Input / Output Instructions

AOUT / BOUT / MOUT – these instructions transfer the value stored in the register (or memory location pointed to by the Y index register) to the Data LEDs.

INA - is used when you want the program to halt, input data into the DATA switches, and then start up the program with this number entered into ACCA. When the program reaches this point in your code, it will halt operation, clear the data LEDs and light the LED above the INCR switch. This is signaling that the computer is in data input mode. Use the data switches to enter the desired 8-bit number. When you are finished, press the INCR key. The program will load the number you placed on the input into ACCA and continue operation.

KEY - is used when you want the computer to input the data switch information that is present at that time during program execution, transfer it into ACCA, and immediately continue the program. This instruction doesn’t wait for you to enter a desired number.

TMR / RTMR –This places the timer value (the number of seconds since the program was run or the timer was reset by RTMR) into the ACCA.

BRT – this sets the brightness of the LEDs. The value can be from 0 (off) to 7 (max) and is the value stored in ACCA when this instruction is run.

TEMP – this loads the current temperature sensor voltage value into ACCA.

Accumulator Instructions

LDA / LDB / LDY – these instructions load the value stored at the next memory location into the associated accumulator.

CLA / CLB – this sets the associated accumulator to zero.

INCA /INCB / INCY – adds 1 to the associated accumulator. Resets to zero after 256 is reached.

DECA / DECB / DECY – subtract 1 from the associated accumulator. Resets to 255 when -1 is reached.

SWP – exchanges the contents of ACCA and ACCB.

SYZ - exchanges the contents of REGY and REGZ.

ROLA – Roll Left ACCA – shift all the bits in ACCA one place to the left. If a 1 is shifted out, it is fed back into the rightmost (LSB) bit.

RORA – Roll Right ACCA – shift all the bits in ACCA one place to the right. If a 1 is shifted out, it is fed back into the leftmost (MSB) bit.

CMPA / CMPB – Compare ACCA/ACCB – this instruction subtracts the value in the next memory location from ACCA/ACCB and sets the condition codes accordingly.

CMAB – Compare ACCA and ACCB - this instruction subtracts the value in ACCB from ACCA and sets the condition codes accordingly.

Memory Instructions

MIA – Memory into ACCA – transfers the contents of the memory location pointed to by REGY into ACCA.

AIM – ACCA into Memory – transfer the contents of ACCA into the memory location pointed to by REGY.

MIB – Memory into ACCA – transfers the contents of the memory location pointed to by REGY into ACCB.

BIM – ACCB into Memory – transfer the contents of ACCB into the memory location pointed to by REGY.

AAM – Add ACCA to Memory – add ACCA to the value in memory pointed to by REGY and store the result in that memory location.

AMA – Add Memory to ACCA – add ACCA to the value in memory pointed to by REGY and store the result in ACCA.

SAM – Subtract ACCA from Memory – subtracts ACCA from the value in memory pointed to by REGY and store the result in that memory location.

Flow control Instructions

JMP – JUMP to Address – load the value of the next memory location into the Program Counter. This will cause program execution to resume at this new address.

WAIT – this instruction will halt program execution and light the RUN LED. Execution will resume when you press the RUN key.

STOP – this stops program execution. The computer stays in RUN mode, but the program just continually loops this instruction.

DLYA / DLYB – Delay ACCA / ACCB – delays program execution by the value of the associated accumulator times 0.1 seconds. For example, if ACCA held the value of 12, DLYA would suspend program execution for 1.2 seconds. NOTE: Reset will not function during the time delay.

Branching Instructions

Branching instructions evaluate a condition based on the condition codes and branch to the address in the next memory location is the condition is met. If the condition is not met, program execution continues with the next instruction.

After an instruction that sets condition coeds is run, you can use the following branch statements:

BEQ – Branch if equal

BNE – Branch is not equal to

BLT – Branch if less than

BLE – Branch if less than or equal

BGT – Branch if greater than

BCS – Branch if ccC (overflow) is set

Math Functions

ADDA – Add to ACCA – add the value of the next memory location into ACCA

ADAB – Add ACCB to ACCA – add ACCB to ACCA and store the result in ACCA

SUBA – subtracts the value in the next memory location from ACCA and store in ACCA

SBA – subtracts ACCB from ACCA and store result in ACCA

SFLA – Shift Left ACCA – shift the bits in ACCA one bit left

SFLB – Shift Left ACCB – shift the bits in ACCB one bit left

SFRA – Shift Right ACCA – shift the bits in ACCA one bit right

SFRB – Shift Right ACCB – shift the bits in ACCB one bit right

ANDA – performs a logical AND between ACCA and the next memory location and stores the result in ACCA

ORA – performs a logical OR between ACCA and the next memory location and stores the result in ACCA

XORA – performs a logical XOR between ACCA and the next memory location and stores the result in ACCA

AAB – performs a logical AND between ACCA and ACCB and stores the result in ACCA

AOB – performs a logical OR between ACCA and ACCB and stores the result in ACCA

AXOB – performs a logical XOR between ACCA and ACCB and stores the result in ACCA

MULA – performs a byte multiplication between ACCA and the next memory location and stores the result in memory locations 254 (MSB) and 255 (LSB)

MAB – performs a byte multiplication between ACCA and ACCB and stores the result in memory locations 254 (MSB) and 255 (LSB)

RNDA – transfers a random number from 0 to 255 into ACCA

Instruction Set

Below is the chart of available instructions with amplifying information. The column headers are:

CMD – Command. This is just the decimal representing the instruction number.

Binary – This is the binary equivalent of the CMD and is what you enter to execute the particular instruction.

ASM – Assembly. This is the assembly language code for the instruction. It is often quicker to write a program in assembly and then hand compile it using the chart of instructions.

Description – This is a brief description of the instruction. The symbol "=>" means "transfers to" in the chart. So instruction 0, ACCA=>DISPLAY means transfer the contents of accumulator A to the Data display LEDs. A particular memory location is represented by M[REGY]. REGY is the pointer to the desired memory location and M[REGY] represents the data in that memory location. A "d" in the chart represents a number held in the next memory location.

BYTES – This shows the number of bytes (memory locations) used for this instruction. Some instructions need additional information and this will be placed in the memory location directly after the instruction. This will cause the instruction to use 2 bytes of memory.

CODES – This column displays the condition codes that may be affected by this instruction. Instruction 3, DATA=>ACCA, transfers a number from the data line input to accumulator A. Since there is no number you can transfer from the DATA line that is greater than 8 bits, the overflow condition code will never be true. And because the computer uses unsigned numbers (0-255), you cannot transfer in a negative number. You can transfer in 0 though, and that would set the ccZ bit so the column shows that the Z condition code could be affected.

CMD	Binary	ASM	Description	BYTES	CODES
0	**00000000**	**AOUT**	ACCA => DISPLAY	1	
1	**00000001**	**BOUT**	ACCB => DISPLAY	1	
2	**00000010**	**MOUT**	M[REGY] => DISPLAY	1	
3	**00000011**	**INA**	DATA => ACCA (wait for it)	1	Z
4	**00000100**	**KEY**	DATA => ACCA (real-time)	1	Z
5	**00000101**	**TMR**	Timer => ACCA elapsed secs	1	Z
6	**00000110**	**RTMR**	0 => Timer	1	
7	**00000111**	**BRT**	Set LED brightness (ACCA)	1	
8	**00001000**	**TEMP**	Temperature => ACCA	1	Z
9	**00001001**	**LDA**	d => ACCA	2	Z
10	**00001010**	**LDB**	d => ACCB	2	Z
11	**00001011**	**LDY**	d => REGY	2	Z
12	**00001100**	**CLA**	0 => ACCA	1	
13	**00001101**	**CLB**	0 => ACCB	1	
14	**00001110**	**INCA**	ACCA + 1 => ACCA	1	Z,C
15	**00001111**	**INCB**	ACCB + 1 => ACCB	1	Z,C
16	**00010000**	**INCY**	REGY + 1 => REGY	1	Z,C
17	**00010001**	**DECA**	ACCA - 1 => ACCA	1	Z,N
18	**00010010**	**DECB**	ACCB - 1 => ACCB	1	Z,N
19	**00010011**	**DECY**	REGY - 1 => REGY	1	Z,N
20	**00010100**	**SWP**	ACCA <=> ACCB	1	Z
21	**00010101**	**SYZ**	REGY <=> REGZ	1	Z
22	**00010110**	**ROLA**	ROLL ACCA 1 BIT LEFT	1	Z,C
23	**00010111**	**RORA**	ROLL ACCA 1 BIT RIGHT	1	Z,C
24	**00011000**	**CMPA**	(ACCA-d); Z if 0, N if <0	2	Z,N
25	**00011001**	**CMPB**	(ACCB-d); Z if 0, N if <0	2	Z,N

26	**00011010**	**CMAB**	(ACCA-ACCB); Z if 0, N if <0	2	Z,N
27	**00011011**	**MIA**	M[REGY] => ACCA	1	Z
28	**00011100**	**AIM**	ACCA => M[REGY]	1	Z
29	**00011101**	**MIB**	M[REGY] => ACCB	1	Z
30	**00011110**	**BIM**	ACCB => M[REGY]	1	Z,N
31	**00011111**	**AAM**	ACCA+M[REGY] => M[REGY]	1	Z,C
32	**00100000**	**AMA**	ACCA+M[REGY] => ACCA	1	Z,C
33	**00100001**	**SAM**	M[REGY]-ACCA => M[REGY]	1	Z,N
34	**00100010**	**JMP**	d=> PC	2	
35	**00100011**	**WAIT**	Stop until INC pressed	1	
36	**00100100**	**STOP**	Stop program execution	1	
37	**00100101**	**DLYA**	DELAY = ACCA x 0.1 sec	1	
38	**00100110**	**DLYB**	DELAY = ACCB x 0.1 sec	1	
39	**00100111**	**BEQ**	If Z, d => PC	2	
40	**00101000**	**BNE**	If not Z, d => PC	2	
41	**00101001**	**BLT**	If N, d => PC	2	
42	**00101010**	**BLE**	If Z or N, d => PC	2	
43	**00101011**	**BGT**	If not (Z or N), d => PC	2	
44	**00101100**	**BGE**	If not N, d => PC	2	
45	**00101101**	**BCS**	If C, d=>PC	2	
46	**00101110**	**ADDA**	ACCA+d => ACCA	2	Z,C
47	**00101111**	**ADAB**	ACCA+ACCB => ACCA	1	Z,C
48	**00110000**	**SUBA**	ACCA-d => ACCA	2	Z,N
49	**00110001**	**SBA**	ACCA-ACCB => ACCA	1	Z,N
50	**00110010**	**SFLA**	SHIFT LEFT ACCA 1 BIT	1	Z,C
51	**00110011**	**SFLB**	SHIFT LEFT ACCB 1 BIT	1	Z,C
52	**00110100**	**SFRA**	SHIFT RIGHT ACCA 1 BIT	1	Z,C
53	**00110101**	**SFRB**	SHIFT RIGHT ACCB 1 BIT	1	Z,C
54	**00110110**	**ANDA**	ACCA (AND) d => ACCA	2	Z
55	**00110111**	**ORA**	ACCA (OR) d => ACCA	2	Z

56	**00111000**	**XORA**	ACCA (XOR) d => ACCA	2	Z
57	**00111001**	**AAB**	ACCA (AND) ACCB => ACCA	1	Z
58	**00111010**	**AOB**	ACCA (OR) ACCB => ACCA	1	Z
59	**00111011**	**AXOB**	ACCA (XOR) ACCB => ACCA	1	Z
60	**00111100**	**MULA**	ACCA X d =>M[254,255]	2	Z
61	**00111101**	**MAB**	ACCA X ACCB => M[254,255]	1	Z
62	**00111110**	**RNDA**	RANDOM (0-255) => ACCA	1	Z

Section

4

Sample programs

Sci-Fi Switchboard

Ever notice in Sci-Fi movies there is always boxes with random lights flashing in the background? Let’s make one of those. We can load a random number into ACCA, display it, delay half a second using ACCB, and then loop back to display another number. First let’s create the program in assembly language. Assembly is when we use the ASM description to write code. This makes it more readable to humans.

Address	ASM	Machine Code
00000000	LDB	
00000001	5	
00000010	RNDA	
00000011	AO	
00000100	DLYB	
00000101	JMP	
00000110	00000010	

The first instruction at memory location 0 is Load into ACCB. This instruction loads the next memory location data into ACCB. Since we are using ACCB for our time delay, let’s load 5 into it. That will be a delay of 0.1x5 or half a second when we call the DLYB instruction. We load the 5 in memory location 1. At memory location 3 we load a random number into ACCA using the RNDA instruction. Next, we display the value in ACCA by calling the AO instruction. Now that the value is displayed, in memory location 4, we call our DLYB instruction that will give us a delay based on the value we loaded into ACCB. Finally, we JMP back to when we loaded a random number into ACCA (memory location 2) and let the program continue to loop. The JMP command jumps to the location represented by the data in the next memory location so we load 2 into memory location 6.

That is the whole program. Now using the chart of instructions, write in the machine code for all of the ASM instructions and copy over any straight data values.

Address	ASM	Machine Code
00000000	LDB	00001010
00000001	5	00000101
00000010	RNDA	00111110
00000011	AO	00000000
00000100	DLYB	00100110
00000101	JMP	00100010
00000110	00000010	00000010

With the machine code ready, we can now program the computer.

Plug in the computer and press RST to ensure the computer is reset to address 0. Enter the first command into the DATA switches (00001010 the 1s represent on LEDs and the 0s are off). When you verify you entered it correctly, press INCR. This will enter the data value into the address and automatically move to the next address. Keep doing this with the rest of the instructions until you have entered them all. Remember to press INCR after the last command to ensure it gets entered.

To run your program, press RST. This brings the computer back to address 0. Now press RUN. The computer will give you a light show worthy of old science fiction shows.

Robot Eye

Let’s look at some code that has to evaluate a condition. We will have our program light the LEDs from left to right.

Address	ASM	Machine Code
00000000	LDB	00001010
00000001	1	00000001
00000010	LDA	00001001
00000011	1	00000001
00000100	AO	00000000
00000101	DLYB	00100110
00000110	CMPA	00011000
00000111	128	10000000
00001000	BEQ	00100111
00001001	00001101	00001101
00001010	SFLA	00110010
00001011	JMP	00100010
00001100	00000100	00000100
00001101	SFRA	00110100
00001110	AO	00000000
00001111	DLYB	00100110
00010000	CMPA	00011000
00010001	2	00000010
00010010	BEQ	00100111
00010011	00000010	00000010
00010100	JMP	00100010
00010101	00001101	00001101

The program shifts the bit in ACCA to the left and checks to see if it has reached the last LED. When it does, it jumps to the segment of the program that shifts right. That continues until it reaches the second LED and then jumps to start the cycle over again. There are many ways to implement this program. Experiment with other ways. Also play with the delay value loaded into ACCB. This will change the speed of the LEDs.

Multiply Two Numbers

Address	ASM	Machine Code
00000000	INA	00000011
00000001	SWP	00010100
00000010	INA	00000011
00000011	MAB	00111101
00000100	LDB	00001010
00000101	10	00001010
00000110	LDY	00001011
00000111	254	11111110
00001000	MIA	00011011
00001001	AO	00000000
00001010	DLYB	00100110
00001011	INCY	00010000
00001100	MIA	00011011
00001101	AO	00000000
00001110	DLYB	00100110
00001111	CLA	00001100
00010000	AO	00000000
00010001	DLYB	00100110
00010010	JMP	00100010
00010011	00000110	00000110

When you run the program it will star in input mode. Enter a number into the DATA switches and press INCR. This loads that number into ACCA. The SWP instruction is run to switch ACCA and ACCB contents. Then the computer goes into input mode again.

Enter the second number you want to multiply and press INCR to load it into ACCA. The program then shows you the result as a sequence of three 1 second periods. The first second it shows the high 8 bits of the result. The second shows the lower 8 bits. The third it clears the screen. Then the cycle repeats. We could have used the command MOUT to just display the values in those memory locations but this program demonstrates transferring memory into ACCA.

You can test it with the following:

1100110 x 11010111 = 01010101 10101010

This will produce an output that is every other LED and shifts as it goes from the high to low bits.

60 Second Timer

Address	ASM	Machine Code
00000000	TMR	00000101
00000001	AO	00000000
00000010	CMPA	00011000
00000011	00111100	00111100
00000100	BLT	00101001
00000101	00000000	00000000
00000110	STOP	00100100

When you run this program, the display will count up to 60 seconds and then halt the program.