BIG 8051

Prepared By:

Sajid Amir
Computer Engineering '15
Elizabethtown College

Josiah Buxton
Computer Engineering '15
Elizabethtown College

Added resources from the works of: James Kelly and David Cain,
Computer Engineering Elizabethtown College
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. General information</td>
<td>3</td>
</tr>
<tr>
<td>2. Key features</td>
<td>4</td>
</tr>
<tr>
<td>3. Connecting the system to a power source</td>
<td>5</td>
</tr>
<tr>
<td>4. C8051 microcontroller</td>
<td>6</td>
</tr>
<tr>
<td>5. Programming microcontroller</td>
<td>7</td>
</tr>
<tr>
<td>6. Ethernet module</td>
<td>8</td>
</tr>
<tr>
<td>7. Piezo buzzer</td>
<td>9</td>
</tr>
<tr>
<td>8. DS 1820 temperature sensor</td>
<td>10</td>
</tr>
<tr>
<td>9. USB UART module</td>
<td>11</td>
</tr>
<tr>
<td>10. RS-232 modules</td>
<td>12</td>
</tr>
<tr>
<td>11. CAN module</td>
<td>13</td>
</tr>
<tr>
<td>12. ZigBee module</td>
<td>14</td>
</tr>
<tr>
<td>13. MMC/SD connector</td>
<td>15</td>
</tr>
<tr>
<td>14. Comparator</td>
<td>16</td>
</tr>
<tr>
<td>15. ADC module</td>
<td>17</td>
</tr>
<tr>
<td>16. DAC module</td>
<td>18</td>
</tr>
<tr>
<td>17. Memory modules</td>
<td>19</td>
</tr>
<tr>
<td>18. LEDs</td>
<td>20</td>
</tr>
<tr>
<td>19. Push buttons</td>
<td>21</td>
</tr>
<tr>
<td>20. I/O ports</td>
<td>22</td>
</tr>
<tr>
<td>21. 8051 Assembly introduction</td>
<td>23</td>
</tr>
<tr>
<td>22. Register addressing</td>
<td>24</td>
</tr>
<tr>
<td>23. Instruction set summary</td>
<td>25</td>
</tr>
<tr>
<td>24. Instruction by opcode</td>
<td>27</td>
</tr>
<tr>
<td>25. Disclaimer</td>
<td>28</td>
</tr>
</tbody>
</table>
General information

The BIG 8051 development system provides a development environment for programming and experimenting with 8051 microcontrollers.

Inside the box

- Development System: BIG8051
- CD: Product CD
- USB Cable
- 128x64 graphic LCD display
- 2x16 alphanumeric LCD display
- Piezo buzzer
- CAN
- ZigBee

System Specification

- **Power Supply:** over AC/DC connector / USB Cable (5V DC)
- **Power Consumption:** 50mA (with all modules off)
- **Dimensions:** 26.5 x 22 cm
- **Weight:** 420g

Features

- Full-featured development system for 8051
- Analog to digital conversion of voltage signal
- CAN communication module
**Key Features**

1. Ethernet module  
2. Piezo buzzer  
3. Connector for temperature sensor  
4. Connector for programmer  
5. USB UART module (USB power supply connector)  
6. RS-232B module  
7. Connector for MCU card  
8. RS232B module  
9. Comparator  
10. Jumpers used to select pull-up/pull-down resistors  
11. DIP switches used to turn on pull-up/pull-down resistors  
12. I/O ports  
13. A/D converter inputs  
14. D/A converter outputs  
15. Memory modules  
16. Touch panel controller  
17. Connector for GLCD display  
18. Connector for touch panel  
19. Push buttons  
20. Jumper used to shorten protective resistor  
21. Jumper used to select push buttons’ logic state  
22. Reset button  
23. Connector for MMC/SD cards  
24. 3.3 V voltage regulator  
25. Power supply module  
26. LEDs  
27. CAN module  
28. ZigBee module  
29. Connector for LCD display
Connecting to power supply

To power the development system on, we have to provide power supply voltage over AC/DC connector CN18. Before you turn the system on, it is necessary to place jumper J11 in the EXT position. Next, set the switch marked POWER SUPPLY to the ON position. As soon as the development system now powers on, a green LED labelled POWER will automatically illuminate.
C8051F040 microcontroller

The BIG8051 system has a 100-pin microcontroller C8051F040 in TQFP package. Today over fifty companies produce variations of the 8051. Several of these companies have over fifty variations of this system. To bring things into context, over 100 million 8051’s are sold each year. Our microcontroller appears to be soldered on the MCU card.

Originally, 8051 belongs to the MCS-51 family of microcontrollers. MCS-51 was developed by Intel but other manufacturers are second sources of this family. The MCS-51, now commonly referred to as the 8051 is a Harvard architecture, consisting of physically separate storage and signal pathways for instructions and data. Today, most processors implement such separate signal pathways for performance reasons. The original MCS-51 featured 8-bit ALU, registers, and data busses. The following table shows the features of the MCS-51 and the BIG8051:

<table>
<thead>
<tr>
<th>Feature</th>
<th>MCS-51</th>
<th>BIG8051</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K bytes internal ROM</td>
<td>4352 bytes internal RAM</td>
<td>64kB Flash</td>
</tr>
<tr>
<td>128 bytes internal RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Four 8-bit I/O ports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two 16-bit timers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64K external code memory space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64K external data memory space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>210-bit addressable locations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined instruction architecture executes 70% of instruction set in 1 or 2 system clocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Up to 25 MIPS throughout with 25MHz clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4352 bytes internal RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64kB Flash; in-system programmable 512-byte sectors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOSCH Controller Area Network (CAN 2.0B), hardware SMBus™ (I2C™ Compatible), SPI™, and two UART serial ports available concurrently</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal calibrated programmable oscillator: 3 to 24.5 MHz, etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The microcontroller can be replaced with another one if desired, but it is important to make sure that they are compatible with the pinouts. 1-wire serial communication enables data to be transferred over one single communication line, while the process itself is under control of the master microcontroller. The advantage of this communication is that only one microcontroller pin is used. All slave devices have unique ID code, which enables the master device to easily identify all devices sharing the same communication bus.
Programming microcontroller

The C8051F040 microcontroller must be programmed using a USB DEBUG ADAPTER which is provided with the box. It is always a good practice to make sure that the power is turned on, the USB DEBUG ADAPTER is connected to the development system through the cable as shown (CN23).
The Ethernet module lets us access the LAN network via the connector (RJ45). Using Serial Peripheral Interface, this module can communicate with the microcontroller.
Piezo buzzers are used for making beeps, tones, and alerts. To use, one pin has to be connected to the ground (in schematic) and the other pin to a square wave out from a timer or microcontroller. For the loudest tones, stay around 4KHz. This buzzer gives the BIG8051 development system the ability to emit audio signals. Best performance for our Piezo buzzer can be achieved with frequencies between 2KHz and 4KHz. To make a connection between the piezo buzzer and the microcontroller, it is necessary to set switch 8 on the DIP switch SW13 to the ON position. The figure currently shows it in off position.
Temperature sensor

Our BIG8051 development system comes with a DS1820 temperature sensor that uses 1-wire communication as mentioned earlier. It can be used to measure temperature in the range between –55 degrees Celsius to 125 degree Celsius with an accuracy of +/- 0.5 degrees Celsius. The sensor can convert temperature with 9-bit resolution and takes a maximum of 750ms. The communication between this module and the microcontroller is achieved via pin P2.7 as shown in the diagram. To establish a connection between them, it is necessary to set switch 8 on the DIP switch SW15 to the ON position.

DS1820 and microcontroller connection schematic [6]

Switch 8 on the DIP switch SW15 is set to OFF at the moment. To connect DS1820, set pin P2.7 to ON.

DS1820 temperature sensor connected. Make sure that the rounded side of the DS1820 matches the half circle on board.
USB UART module

The USB UART module is used to connect the microcontroller to an external USB device. To establish connection between the microcontroller and the USB UART module, it is important to set switches 1 and 3 (optionally 2 and 4) to the ON position.

The BIG8051 can also use this module to power itself. To achieve this, it is necessary to set the switch position of jumper J11 (found on the power module) from EXT to USB.
RS-232 modules

The RS-232 modules allow the development system to communicate to external devices in compliance with the RS-232 standard. The two RS modules in the BIG8051 can operate separately. To connect, switches 1 and 3 (optionally 2 and 4) on DIP switch SW11 should be set to the ON position.
Controller Area Network (CAN) is a communication standard primarily intended for use in the automotive industry. It is used when microcontrollers and devices need to communicate with each other in applications without a host computer. The modern automobile may have as many as 70 electric control units for various subsystems. Typically the biggest processor is the engine control unit. Others are used for transmission, airbags, ABS, cruise control, electric power steering, audio systems, power windows, doors, mirrors, etc. Some of these form independent subsystems, but communications among others are essential. A subsystem may need to control actuators or receive feedback from sensors. The CAN standard was devised to fill this need.

Each CAN node is able to send and receive messages, but not simultaneously. The devices that are usually connected by a CAN network are typically sensors, actuators, and other control devices. The host processor decides what the received messages mean and what messages it wants to transmit. Sensors, actuators and control devices can be connected to the host processor. CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor, usually by the CAN controller triggering an interrupt. For sending, the host processor sends the transmit messages to a CAN controller, which transmits the bits serially onto the bus when the bus is free.

To establish connection between this module and the microcontroller, it is necessary to set the switches 1 and 2 on the DIP switch SW12 to the ON position.
ZigBee module

The BIG805 enables you to connect the ZigBee module that is used for wireless communication. The module communicates with the microcontroller with the standard SPI protocols. It is necessary to set switches 1, 3, and 5 on the DIP switch SW10, as well as the switches 1, 2, 3, and 4 on the DIP switch SW15 to the ON position for functionality.

At the moment, our boards do not have any RF module attached to it. However, the MRF24J40MA ZigBee module can be connected to the development system. This module can be used as an ideal solution for wireless sensor networks, home automation, building automation, and consumer applications. The features of this particular radio transceiver module are:
- 2.4 GHz IEEE 802.15.4 Transceiver module supports ZigBee™, MiWi™, and MiWi P2P protocols
- Integrated PCB Antenna with simple four-wire SPI interface to PIC microcontroller
- Low current consumption
MMC/SD connector

The development system is able to read memory cards due to the on-board MMC/SD connector. The module uses microcontroller pins for serial communication. To connect this module, it is necessary to set switches 1, 3 and 5 (optionally 2, 4, and 6) on the DIP switch SW10, as well as switches 1 and 2 on the DIP switch SW14 to the ON position.
Comparator

The BIG8051 can compare voltage levels due to a comparator built into the microcontroller. Voltage signals are supplied via potentiometers P5 and P6. To enable voltage signals, set switches 5 and 6 on DIP switch SW15 to the ON position.
ADC module

The ADC module is used to convert an analog voltage level into the appropriate 12-bit digital value. The analog voltage signal is supplied via screw terminals CH0, CH1, CH2, and CH3. The ADC module is built-in into the microcontroller. The voltage supplied from the VREF pin is used as a voltage reference. In order to use this voltage, switch 8 on the DIP SW14 should be set to the ON position.
The DAC module attached to the development system can be used to convert 12-bit digital values into appropriate analog voltage values. The output analog voltage signals are delivered via screw terminals DAC0, and DAC1. Like the ADC module, this module also uses the VREF pin on the microcontroller as a voltage reference. To use this voltage, switch 7 on DIP switch SW14 should be set to the ON position.
Memory modules

The BIG8051 comes with Flash, RAM and EEPROM memory modules. This allows the microcontroller to expand its memory space.

The Flash module lets the microcontroller to use an additional 8Mbit flash memory via SPI. To establish connection between the Flash module and the microcontroller, switches 1, 3, and 5 (optionally 2, 4, and 6) on the DIP switch SW10, as well as switch 7 on the DIP switch SW15 to the ON position.

The RAM module enables the microcontroller to use an additional 64Kbit RAM memory via SPI. To enable this connection between the module and the microcontroller, it is necessary to set switches 1,3,5,7 and 8 (optionally 2,4,6) on the DIP switch SW10 to the ON position.

EEPROM module enables the microcontroller to use an additional 1Kbit EEPROM memory via I2C serial connection. Set switches 3 and 5 (optionally 4 and 6) on the DIP switch SW12 to the ON position.
LEDs

There are a total of 64 LEDs on the development board and can be used to visually indicate the state of each microcontroller I/O pin. When active, an LED indicates that a logic 1 is present on that particular pin. To enable LEDs to illuminate, it is necessary to select the appropriate port using DIP switch SW9.

There are 8 available ports labelled PORT0 through PORT7.
Push buttons

The provided push buttons on the development board can be used to change the logic state of all the microcontroller input pins. Jumper J10 is used to determine the logic state to be supplied on the appropriate microcontroller pin by pressing a push button. On the lower left side of the push buttons, a RESET button can be found to reset the microcontroller.
Input / Output ports

The BIG8051 has 10-pin connectors lined to the microcontroller I/O ports. Due to DIP switches SW1-SW8, every connector pin can be connected to one pull-up/pull-down resistor. It depends on the position of jumpers J1-J8 to control this aspect.

The pull-up resistor ensures that the signal will be a valid logic level if external devices are connected. A pull up resistor weakly “pulls” the voltage of the wire it is connected to towards its voltage source level when the other components on the line are inactive. A pull-down resistor works the same way but is connected to ground. It holds the logic signal near zero volts when no other device is connected.
8051 Assembly Introduction

A computer instruction is made up of an operation code (op-code) followed by either zero, one, or two bytes of operands.

The op-code identifies the type of operation to be performed while the operands identify the source and destination of the data.

If the instruction is associated with more than one operand, the format is always:

**Instruction**    **Destination, Source**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>MOV A, B</td>
</tr>
<tr>
<td>Direct</td>
<td>ADD A, @R0</td>
</tr>
<tr>
<td>Indirect</td>
<td>ADD A, #80H</td>
</tr>
<tr>
<td>Immediate Constant</td>
<td>SJMP +127/128 of PC</td>
</tr>
<tr>
<td>Relative*</td>
<td>AJMP within 2K</td>
</tr>
<tr>
<td>Absolute*</td>
<td>MOV C @A+PC</td>
</tr>
<tr>
<td>Long*</td>
<td>MOV D @A+PC</td>
</tr>
</tbody>
</table>

The memory map of the C8051 if the lower data RAM area. Addresses 0x00 through 0x1F are banked registers R0-R7. The active bank is controlled via bits in the Program Status Word. From the PSW, we can conclude that the bit addressable memory located from 0x20 through 0x2F which provides 128 bits of bit addressable memory. The upper portion is used general purpose RAM and can be accessed by any addressing mode (direct or indirect).

Special function registers (SFRs) have been added to the C8051 to that of the standard 8051 for enhanced peripherals. Upper data memory and SFR memory share the same address space but are accessed via different addressing modes (direct vs. indirect). The SFR portion can be accessed via direct addressing only.

There are eight modes of addressing available on the C805. The different addressing modes determine how the operand byte mentioned earlier to be selected.
Register Addressing

In CPU registers are used to store information temporarily. That information can be in the form of bytes of data to be processed or some address pointing to a data to be fetched. In 8051 there is only one data type: 8 bits. With an 8 bit data registers, any data more than 8-bit should be divided into 8 bit chunks before being processed.

The most widely used registers of 8051 are A (accumulator), R0, R1, R2, R3, R4, R5, R6, R7, DPTR (data pointer), PC (program counter). A register is used for all arithmetic and logic operations. These registers are divided into two groups:

- General purpose registers
- Special purpose registers

The register addressing instruction involves information transfer between registers

Example:

- MOV R0, A
- MOV A, #51H
- MOV R1, #0FFH
- MOV R2, #3BH

The first instruction above transfers the accumulator content into the R0 register. The register bank (Bank 0, 1, 2 or 3) must be specified prior to this instruction.

- # signifies that it is a number and not a byte address
- ‘H’ symbolizes that it is a hexadecimal number. Similarly if we write a binary number it should end with ‘B’
- The ‘0’ in the third instruction in 0FFH symbolizes that the ‘F’ is a hex number and not a letter. This is always necessary if we are writing any hex numbers. It should always start with a 0.
- All the register should be assumed to be of 1 byte unless otherwise mentioned.

Now to transfer contents of R1 to lets say, R6, the following code should do the trick:

MOV R6, R1 ; copy contents of R1 to T6

DIRECT ADDRESSING:

This mode allows you to specify the operand by giving its actual memory address (typically specified in hex format) or by giving its abbreviated name. It is also used for SFR accesses.

INDIRECT ADDRESSING:

In the indirect addressing mode, a register is used to hold the effective address of the operand. This register, which holds the address, is called the pointer register and is said to point to the operand. Only registers R0, R1 and DPTR can be used as pointer registers. DPTR is useful in accessing operands which are in the external memory.

The 8051 microcontroller instructions are divided among five functional groups

1. Arithmetic
2. Logical
3. Data transfer
4. Boolean variable
5. Program branching
### Instruction Set Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Operations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD A, Rn</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, direct</td>
<td>Add direct byte to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADD A, @Ri</td>
<td>Add indirect RAM to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ADD A, #data</td>
<td>Add immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, Rn</td>
<td>Add register to A with carry</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADDC A, direct</td>
<td>Add direct byte to A with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, @Ri</td>
<td>Add indirect RAM to A with carry</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SUBB A, Rn</td>
<td>Subtract register from A with borrow</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBB A, direct</td>
<td>Subtract direct byte from A with borrow</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SUBB A, @Ri</td>
<td>Subtract indirect RAM from A with borrow</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SUBB A, #data</td>
<td>Subtract immediate from A with borrow</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC Rn</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC direct</td>
<td>Increment direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC @Ri</td>
<td>Increment indirect RAM</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC Rn</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC direct</td>
<td>Decrement direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DEC @Ri</td>
<td>Decrement indirect RAM</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL AB</td>
<td>Multiply A and B</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DIV AB</td>
<td>Divide A by B</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Logical Operations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANL A, Rn</td>
<td>AND Register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, direct</td>
<td>AND direct byte to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, @Ri</td>
<td>AND indirect RAM to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, #data</td>
<td>AND immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL direct, A</td>
<td>AND A to direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL direct, #data</td>
<td>AND immediate to direct byte</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ORL A, Rn</td>
<td>OR Register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, direct</td>
<td>OR direct byte to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, @Ri</td>
<td>OR indirect RAM to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, #data</td>
<td>OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL direct, A</td>
<td>OR A to direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL direct, #data</td>
<td>OR immediate to direct byte</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>XRL A, Rn</td>
<td>Exclusive-OR Register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, direct</td>
<td>Exclusive-OR direct byte to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, @Ri</td>
<td>Exclusive-OR indirect RAM to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, #data</td>
<td>Exclusive-OR immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL direct, A</td>
<td>Exclusive-OR A to direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL direct, #data</td>
<td>Exclusive-OR immediate to direct byte</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPLA</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRA A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Instruction Set Summary (cont.)

#### Data Transfer

| Instruction          | Description                                      | CY | CY
|----------------------|--------------------------------------------------|----|----
| MOV A, Rn           | Move Register to A                                | 1  | 1  
| MOV A, direct       | Move direct byte to A                             | 2  | 2  
| MOV A, @Ri          | Move indirect RAM to A                            | 1  | 2  
| MOV A, @iabs        | Move indirect memory to A                         | 2  | 2  
| MOV Rn, A           | Move A to Register                                | 1  | 1  
| MOV Rn, direct      | Move direct byte to Register                      | 2  | 2  
| MOV Rn, @iabs       | Move indirect Register to A                       | 2  | 2  
| MOV direct, @a       | Move A to direct byte                             | 2  | 2  
| MOV direct, Rn      | Move Register to direct byte                      | 2  | 2  
| MOV direct, direct  | Move direct byte to direct byte                   | 3  | 3  
| MOV direct, @a       | Move indirect RAM to direct byte                  | 2  | 2  
| MOV direct, Rn      | Move immediate to direct byte                     | 3  | 3  
| MOV @Ri, A          | Move A to indirect RAM                            | 1  | 2  
| MOV @Ri, direct     | Move direct byte to indirect RAM                  | 2  | 2  
| MOV @Ri, @iabs      | Move indirect RAM to indirect RAM                 | 2  | 2  
| MOV @Ri, @iabs      | Move immediate to indirect RAM                    | 2  | 2  
| MOV @Ri, A          | Move A to indirect RAM                            | 2  | 2  
| MOV @Ri, direct     | Move direct byte to indirect RAM                  | 2  | 2  
| MOV R0 @Ri, A       | Move external data (8-bit address) to A           | 1  | 3  
| MOVX A, @Ri         | Move A to external data (8-bit address)           | 1  | 3  
| MOVX A, @DPTR       | Move external data (16-bit address) to A          | 1  | 3  
| MOVX A, @DPTR, A    | Move A to external data (16-bit address) to A     | 1  | 3  
| PUSH direct         | Push direct byte onto stack                       | 2  | 2  
| POP direct          | Pop direct byte from stack                        | 2  | 2  
| XCH A, Rn           | Exchange Register with A                          | 1  | 1  
| XCH A, direct       | Exchange direct byte with A                       | 2  | 2  
| XCH A, @Ri          | Exchange indirect RAM with A                      | 1  | 2  
| XCHD A, @Ri         | Exchange low nibble of indirect RAM with A        | 1  | 2  

#### Boolean Manipulation

| Instruction       | Description                                      | CY | CY
|-------------------|--------------------------------------------------|----|----
| CLR C             | Clear Carry                                      | 1  | 1  
| CLR bit           | Clear direct bit                                 | 2  | 2  
| SETB C            | Set Carry                                        | 1  | 1  
| SETB bit          | Set direct bit                                   | 2  | 2  
| CPL C             | Complement Carry                                 | 1  | 1  
| CPL bit           | Complement direct bit                            | 2  | 2  
| ANDI C, bit       | AND direct bit to Carry                          | 2  | 2  
| ANDI C, @iabs     | AND complement of direct bit to Carry            | 2  | 2  
| ORI C, bit        | OR direct bit to carry                           | 2  | 2  
| ORI C, @iabs      | OR complement of direct bit to Carry             | 2  | 2  
| MOV C, bit        | Move direct bit to Carry                         | 2  | 2  
| MOV C, @iabs      | Move Carry to direct bit                         | 2  | 2  
| JC rel             | Jump if Carry is set                             | 2  | 2/3 
| JNC rel            | Jump if Carry is not set                         | 2  | 2/3 
| JB bit, rel       | Jump if direct bit is set                        | 3  | 3/4 
| JNB bit, rel      | Jump if direct bit is not set                    | 3  | 3/4 
| JBCC bit, rel     | Jump if direct bit is set and clear bit          | 3  | 3/4 

#### Program Branching

| Instruction       | Description                                      | CY | CY
|-------------------|--------------------------------------------------|----|----
| CALL addr11       | Absolute subroutine call                         | 2  | 3  
| LCALL addr16      | Long subroutine call                             | 3  | 4  
| RET               | Return from subroutine                           | 1  | 5  
| RETI              | Return from interrupt                            | 1  | 5  
| AJMP addr11       | Absolute jump                                    | 2  | 3  
| LJMP addr16       | Long jump                                        | 3  | 4  
| JMP rel (short jump) | Jump indirect relative address                  | 2  | 3  
| JMP &@DPTR        | Jump indirect relative to @DPTR                  | 1  | 3  
| JZ rel            | Jump if A equals zero                            | 2  | 2/3 
| JNZ rel            | Jump if A does not equal zero                    | 2  | 2/3 
| CJNE A, direct, rel | Compare direct byte to A and jump if not equal | 3  | 3/4 
| CJNE A, @iabs, rel | Compare indirect to A and jump if not equal     | 3  | 3/4 
| CJNE Rn, @iabs, rel | Compare immediate to Register and jump if not equal | 3  | 3/4 
| CJNE @Ri, @iabs, rel | Compare indirect to @iabs and jump if not equal | 3  | 4/5 
| DJNZ Rn, rel      | Decrement Register and jump if not zero          | 2  | 3/4 
| DJNZ direct, rel  | Decrement direct byte and jump if not zero       | 3  | 3/4 
| NOP               | No operation                                     | 1  | 1  

---
<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOP</td>
<td>AJMP</td>
<td>LJMP</td>
<td>RR</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
</tr>
<tr>
<td>10</td>
<td>JBC</td>
<td>ACALL</td>
<td>LCALL</td>
<td>RRC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
</tr>
<tr>
<td>20</td>
<td>JB</td>
<td>AJMP</td>
<td>RET</td>
<td>RL</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
</tr>
<tr>
<td>30</td>
<td>JNB</td>
<td>ACALL</td>
<td>RETI</td>
<td>RLC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
<td>ADDC</td>
</tr>
<tr>
<td>40</td>
<td>JC</td>
<td>AJMP</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
<td>ORL</td>
</tr>
<tr>
<td>50</td>
<td>JNC</td>
<td>ACALL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
<td>ANL</td>
</tr>
<tr>
<td>60</td>
<td>JZ</td>
<td>AJMP</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
<td>XRL</td>
</tr>
<tr>
<td>70</td>
<td>JNZ</td>
<td>ACALL</td>
<td>ORL</td>
<td>JMP</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>80</td>
<td>SJMP</td>
<td>AJMP</td>
<td>ANL</td>
<td>MOVC</td>
<td>DIV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>90</td>
<td>MOV</td>
<td>ACALL</td>
<td>MOV</td>
<td>MOVC</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
<td>SUBB</td>
</tr>
<tr>
<td>A0</td>
<td>ORL</td>
<td>AJMP</td>
<td>MOV</td>
<td>INC</td>
<td>MUL</td>
<td>===</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>B0</td>
<td>ANL</td>
<td>ACALL</td>
<td>CPL</td>
<td>CPL</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
<td>CJNE</td>
</tr>
<tr>
<td>C0</td>
<td>PUSH</td>
<td>AJMP</td>
<td>CLR</td>
<td>CLR</td>
<td>SWAP</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
<td>XCH</td>
</tr>
<tr>
<td>D0</td>
<td>POP</td>
<td>ACALL</td>
<td>SETB</td>
<td>SETB</td>
<td>DA</td>
<td>DJNZ</td>
<td>XCHD</td>
<td>XCHD</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
<td>DJNZ</td>
</tr>
<tr>
<td>E0</td>
<td>MOVX</td>
<td>AJMP</td>
<td>MOVX</td>
<td>CLR</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>F0</td>
<td>MOVX</td>
<td>ACALL</td>
<td>MOVX</td>
<td>CPL</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
</tbody>
</table>
Writing your first Program

Download Silicon Laboratories IDE. Follow the link for the download:

Next, download an assembler. The recommended assembler is: Keil uVision5
- http://www.keil.com/c51/

After both the installers are downloaded, install the software and run the IDE first. Then proceed to write your program in assembly language.

Assembling your Code

After completing your code, click the assemble button as shown:

Next, you may be prompted to find an assembler. Browse for the assembler (Keil uVision 5) installed earlier. Click rebuild all as shown:
Connecting the IDE with the Development Board

Before establishing connection between the development board and the IDE, power on the system as instructed earlier. The jumpers J13 and J14 should be in the JTAG position.

- Click options and then select Connection Options from the drop down:
- In the Collection Options Dialog Box, select configuration as shown:
- Click Connect to establish connection:
Downloading and Running Code on 8051

- Click Download after connection has been established successfully
- Then click the Go button as shown
- Observe results on board
How to set up the 8051 Simulator

1) Make sure you have Java installed, you can download it at:
2) Run the installer you downloaded and make sure to uncheck installing the Ask Toolbar
3) Download http://www.edsim51.com/8051simulator/edsim51di.zip. If the software has been updated since this writing, the newer version can be downloaded at http://www.edsim51.com
4) Unzip the file
5) Inside edsim51di folder, run edsim51di.jar

- In the middle window are your main controls where you will be able to write and execute your code
- The bottom window allows you to manipulate virtual inputs to the 8051, and see outputs on its various LCDs and LEDs
- On the left is a representation of what is stored in the memory and registers of your 8051
  - Note: To clear some of these requires closing and re-opening the simulator
- In the black box on the right you’ll find a reference for how to access various “hardware” elements of the simulated 8051, such as switches and LEDs, and whether currently sees a logic 1 or 0
Coding the Simulator

1) Above the assembly code window, you will see the toolbar depicted below. Either select “New” to begin a new assembly or “Load” to import an existing one

2) Once a program has been written or loaded, select either “Assm” or “Run” to execute it. If the simulator identifies any errors in the code, the program will not run, and a message will appear indicating where the problem originated
   - “Assm” is used to manually step through the code and observe the individual steps. Once it has been clicked, the button will change to “Step” and must be pressed repeatedly to advance the program one line of code at a time
   - “Run” will run the program all the way to the end without stopping. At any point, the user may pause the program by clicking the button again.

3) If you wish to stop the program from running entirely, select “RST”. This will end the program and allow you to make modifications or simply begin the program again.

4) When finished, you can save your code as a .asm by clicking save, which will allow you to import your work at a later time.

Sample Code:

1) A simple program to display binary values from 0 to 255 on the LEDs (note, since the simulator considers logic 0 to be ‘lit’ for the LEDs, the program must decrement instead of increment)

   ```
   main:
   DEC P1 ;decrement instead of increment because logic 0 lights the LEDs
   JMP main
   ```

2) A Program to mirror the 8 push buttons onto the corresponding LEDs

   ```
   main:
   MOV P1, P2 ;move values on buttons into the LEDs
   JMP main
   ```
Simulation Video (Amir & Buxton)

https://youtu.be/x5dIODI726U
Sample Project (Cain & Kelly)

Below is a fully functional program that may be run on the simulator. The program is designed to control the water level in a bucket used as part of a hydroponic garden. Switches 1, 2 and 3 on the board serve as stand ins for a trio of float sensors that will activate when the water level reaches a set height. The first four LEDs on the board correspond to the four devices that the system needs to control. LED 0 represents the input valve, LED 1 represents an aerator that needs to run for 5 seconds each time a switch is flipped for the first time. LEDs 3 and 4 represent the drain valve that will allow the water out (3), and a light that is required to be turned on while the bucket empties (4). Recall that, as with the earlier example programs, logic 0 is required to “light” the LEDs, so CLR is used to light them instead of SETB.

```
main:
    JNB F0, initialize ; F0 is a flag that is set when initialization has been run once
    SETB P1.1 ; ensure that the aerator is off
    JB P0.3, drain ; if P0.3 is set, the bucket has finished filling and should begin to drain
    CLR P1.0 ; Begin filling the bucket
    JNB P2.3, Switch3 ; is the third switch tripped? (note that the last switch must be the first checked so as to not get stuck)
    JNB P2.2, Switch2 ; Is the second switch tripped?
    JNB P2.1, Switch1 ; Is the first switch tripped?
    JMP main ; continuously loop
Switch1:
    JB P0.0, main ; If this function has been executed before return to main
    SETB P0.0 ; set as flag that this function has been executed
    JMP aerate ; jump if this is the first time the switch read as pressed
Switch2:
    JB P0.1, main ; If this function has been executed before return to main
    SETB P0.1 ; set as flag that this function has been executed
    JMP aerate ; jump if this is the first time the switch read as pressed
Switch3:
    JB P0.2, main ; If this function has been executed before return to main
    SETB P0.2 ; set as flag that this function has been executed
    SETB P0.3 ; set to indicate that bucket is full
    JMP aerate ; jump if this is the first time the switch read as pressed
aerate:
    SETB P1.0 ; stop filling
    CLR P1.1 ; start aerator
    JMP delay ; jump to delay
delay:
    MOV R0,#20 ; set loop count to 20
    JMP countdown ; jump to countdown
countdown:
    DEC R0
    DJNZ R0,countdown ; keep looping until R0 hits 0
    JMP main ; return to main
drain:
    CLR P1.2 ; open drain valve
    CLR P1.3 ; turn on light
    JNB P1.3, drain ; keep looping until the water drops below the first sensor
    SETB P1.2 ; close drain valve
    SETB P1.3 ; turn off light
    JMP initialize ; Reset the flags so the program can restart automatically
initialize:
    ; This function sets flags to ensure certain functions are only run once (aerate once per switch, drain once)
    CLR P0.0 ; This will be set when the first sensor is tripped
    CLR P0.1 ; This will be set when the second sensor is tripped
    CLR P0.2 ; This will be set when the third sensor is tripped
    CLR P0.3 ; This will be set after the third switch is tripped to confirm that the bucket is full
    SETB F0 ; Set to confirm that this function has been run once
    JMP main ; return to main
```
Disclaimer

This manual was prepared strictly for academic use. All the information gathered here have been collected via research and aimed to guide any student to start using the BIG 8051 microcontroller. All the pictures of the development system were taken by Sajid Amir and should not be copied.

References:

- http://users.etown.edu/w/wunderjt/Xilinx%20instructions.pdf