C.S. Project

INTEL 8085

* INTRODUCTION ABOUT INTEL 8085:-

The **Intel 8085** is an [8-bit](http://en.wikipedia.org/wiki/8-bit) [microprocessor](http://en.wikipedia.org/wiki/Microprocessor) introduced by [Intel](http://en.wikipedia.org/wiki/Intel) in 1977. It was binary-compatible with the more-famous [Intel 8080](http://en.wikipedia.org/wiki/Intel_8080) but required less supporting hardware, thus allowing simpler and less expensive [microcomputer](http://en.wikipedia.org/wiki/Microcomputer) systems to be built.

The "5" in the model number came from the fact that the 8085 required only a +5-[volt](http://en.wikipedia.org/wiki/Volt) (V) power supply rather than the +5V, -5V and +12V supplies the 8080 needed. Both processors were sometimes used in computers running the [CP/M operating system](http://en.wikipedia.org/wiki/CP/M_operating_system), and the 8085 later saw use as a [microcontroller](http://en.wikipedia.org/wiki/Microcontroller), by virtue of its low component count. Both designs were eclipsed for desktop computers by the compatible [Zilog Z80](http://en.wikipedia.org/wiki/Zilog_Z80), which took over most of the CP/M computer market as well as taking a share of the booming [home computer](http://en.wikipedia.org/wiki/Home_computer) market in the early-to-mid-1980s.

The 8085 had a long life as a controller. Once designed into such products as the [DECtape](http://en.wikipedia.org/wiki/DECtape) controller and the [VT100](http://en.wikipedia.org/wiki/VT100) video terminal in the late 1970s, it continued to serve for new production throughout the life span of those products (generally longer than the product life of desktop computers).

The 8085 is a conventional [von Neumann](http://en.wikipedia.org/wiki/Von_Neumann_architecture) design based on the Intel 8080. Unlike the 8080 it had no state signals multiplexed onto the data bus, but the 8-bit [data bus](http://en.wikipedia.org/wiki/Data_bus) was instead multiplexed with the lower part of the 16-bit [address bus](http://en.wikipedia.org/wiki/Address_bus) to limit the number of pins to 40. Pin #40 is used for the power supply (+5v) and pin #20 for ground. Pin #39 is used as the hold pin. Pins #15 to #8 are generally used for address buses. The processor was designed using [nMOS](http://en.wikipedia.org/wiki/NMOS) circuitry and the later "H" versions were implemented in Intel's enhanced nMOS process called HMOS, originally developed for fast static RAM products. Only a 5 Volt supply was needed, like competing processors and unlike the 8080. The 8085 used approximately 6,500 [transistors](http://en.wikipedia.org/wiki/Transistor).[[1]](http://en.wikipedia.org/wiki/Intel_8085#cite_note-0)

The 8085 incorporated the functionality of the 8224 (clock generator) and the 8228 (system controller), increasing the level of integration. A downside compared to similar contemporary designs (such as the Z80) was the fact that the buses required demultiplexing; however, address latches in the Intel 8155, 8355, and 8755 memory chips allowed a direct interface, so an 8085 along with these chips was almost a complete system.

The 8085 had extensions to support new interrupts: It had three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one [non-maskable interrupt](http://en.wikipedia.org/wiki/Non-maskable_interrupt) (TRAP), and one externally serviced interrupt (INTR). The RST n.5 interrupts refer to actual pins on the processor, a feature which permitted simple systems to avoid the cost of a separate interrupt controller.

Like the 8080, the 8085 could accommodate slower memories through externally generated [wait states](http://en.wikipedia.org/wiki/Wait_state) (pin 35, READY), and had provisions for [Direct Memory Access](http://en.wikipedia.org/wiki/Direct_Memory_Access) (DMA) using HOLD and HLDA signals (pins 39 and 38). An improvement over the 8080 was that the 8085 can itself drive a [piezoelectric crystal](http://en.wikipedia.org/wiki/Piezoelectric_crystal) directly connected to it, and a built in clock generator generates the internal high amplitude two-phase clock signals at half the crystal frequency (a 6.14 MHz crystal would yield a 3.07 MHz clock, for instance).

(SOURCE:- <http://en.wikipedia.org/wiki/Intel_8085>)

Memory Structure and Functions:-

In our programming of functions, we initially defined a global variable 'int' array 'memory', where all our data has been stored. According to directories as in 8085 simulator we allocated memory as follows:

memory[0] => A

memory[1] => B

memory[2] => C

memory[3] => D

memory[4] => E

memory[5] => H

memory[6] => L

memory[7] & memory[8] => PC

memory[9] & memory[10] =>SP

memory[11] => F

The major problem with the 8085 simulator is that memory is limited and while storing of data, we have to be careful and manipulative.

memory 7 and 8 ...&... 9 and 10 are always used together as a 16 bit (or 2 byte) memory position, whereas all others are 8 bit.

wecordinated the directories to 'memory' using "int address(char ch)"....

in simple words, on passing the name of the directory (eg. B), we get the output as 2.

8085 simulator requires the hexadecimal data added by the user to be converted to binary form..

.....the function --- "void binaryArr(intnum,int workspace[8])" was thus defined.

This function extracted data from the data stored in the diectories(i.e. 'memory'), then stored in the binary form in the ‘workspace’ array.

One more interesting and important to note is the use of **unsigned char** and not normal **char.**

Our data in the **workspace** array is to be stored in the **1 byte(or 2 nibbles)**, due to constraint of memory space.

Unsigned char helps store data in the of 8 b its (which is enough for our utily, we can also use the 16 bits directories). This is possible only in two pre defineddatatypes – char & unsigned char.

But the problem arises when we use char datatype. When the 1st bit of char has value as ‘1’ (which is called the MSB-most significant bit), then its value is automatically(unnecessarily) changed to another value

(eg. For binary form 1 0 0 0 1 0 0 1, 0 0 0 0 0 0 0 1 is added to it. Its NOT it taken. The minus(-) of this binary form is set. )

Another important aspect during arithmetic operations in 8085 is the 5 set of carry Flags stored in the 8 bit directory. It includes: S,Z,P,AC,CY. These are define in a binary number.

example 1010 0110

1. S: It is starting bit (most significant bit) of binary number.
2. If binary number is 0000 0000 than z=1. Else z=0.
3. P is parity which stores 1 when the number of 1’s in the binary is even. Else it stores 0.
4. Ac is the auxiliary carry.(i.e the carry while addition of the 2nd nibble.
5. Cy is the final carry after addition of the two 8 bit numbers.

We finally come to the programming of the Arithmetic functions:

1. ADD: add register to accumulator

void ADD(char)

This is a simple addition function. In this function, by default, addition is done on the A directory. The input char represents the name of the directory. The number stored is then extracted and then added to the numerical stored in A.

Here, 1st both numbers are converted to binary form. The numbers are added using simple carry concept. However a unit carry at the last place means we need another bit to store the number(which is not possible). This last carry is stored in the carry Flag.(Cy)

1. ADC: add register to accumulator with carry

void ADC(char)

This is almost same function as ADD,the difference between ADC and ADD is that ADC has a role of CY(carry) as it includes carry as a addition. The previous carry flag is reset.

It is very useful in the addition of 16 bit numbers. The carry in the addition of the 2nd 8 bit in added in the 1st 8 bits. Hence the carry is used up.

1. ADI: add immediate to accumulator

void ADI(char)

In this function the value to be added to A is given as an operand, and the addition is done just like an ADD function. The flags are affected exactly in the same way as the ADD function.

However instead of a directory name being passed, a hexadecimal no. Is passed directly

1. ACI: add immediate to accumulator with carry

void ACI(char)

This is a combination of ADI & ADC. Here too an immediate value is passed(like ACI), but it is useful in 16 bit addition due to transfer of carry (Cy) just like ADC.

1. ANA:

void ANA(char)

This applies the **bitwise logical AND operator** on A and the directory named with ANA.

1. ORA:

void ORA(char)

This applies the **bitwise logical OR operator** on A and the directory named with ANA.

1. XRA:

void XRA(char)

This applies the **bitwise logical Exclusive OR operator(XOR)** on A and the directory named with ANA.

1. MOV:

void MOV(char, char)

This function copies the contents of the 1st directory and assigns the value to the 2nd directory.

1. MVI:

void MVI(char,int)

This function assigns the integer value to the directory name entered.

1. INR:

void INR(char)

This function added a ‘1’(unit binary no.)using the ADD function.

1. CMP:

void CMP(char)

This function compares the value of directory with that of the accumulator. The operation carriedis the subtraction of register from the accumulator. The flags are affected in the usual way.

1. SUB:

voidSUB(char)

As the name suggests, the function subtracts the value of directory from that of the accumulator, using the concepts of two’s complement.

* Working:-

Our algorithms and ideas have been implemented as follows:-

The user is required to enter the INTEL 8085 code. The code read is then divided into substrings by identifying the spaces in the input line. The various substrings are then stored in a 2-dimensional array. We decided to divide our program in 2 parts. In the first part we convert the input text in hexadecimal form going through binary code phase. Suppose we write a code as follows:-

MOV A B

First of all the various sub strings will be stored in the 2-D array as follows:-

First row of the array will have MOV, second row will have A and the third row will have B. Now we pass this array through a function called binarygenerator. The code for MOV is “00” and then by calling a function called binary we find out the code for substrings A and B. Then the 3 generated binary codes are put together to form the complete 8 bit binary code for the code MOV A B.

This binary code is converted into hexadecimal format by breaking the binary code into 2 different parts and then converting them into hex form which is then merged together using an array. Now this code is entered into a file called hex.txt. This is used to generate the hexview of the simulator. Meanwhile the functions are called according to the code generated by passing appropriate parameters through the function.