ADDITIONAL DATA TRANSFER INSTRUCTIONS — ASSIGNMENT 4

Object To use additional instructions from the data transfer group which involve the microprocessor and system memory.

Approximate time required Two hours

Memory is like a filing system. Although the information in it is not 'at your fingertips' so that it can be immediately worked on, it can be brought out, one item at a time, and the contents can be changed when required. Memory contains

the working instructions (always)

and (when required) –

information to be worked on (input data)
results of work done (processed data available for output)
any information too extensive to be held in the registers.

This assignment deals with communication to and from memory.

4.1 Introduction

The previous assignment was concerned primarily with basic data transfer instructions which involved only the internal processor registers and which used either the register or immediate addressing modes. This assignment is therefore concerned with some additional data transfer instructions which involve the system memory and which use either the direct or register indirect addressing modes.

4.2 Data Transfer Between the Processor and System Memory

The two addressing modes which provide for data transfer between the processor and system memory are direct (extended) addressing and register indirect addressing.

4.2.1 Direct Addressing

Using direct addressing an operand may either be read from or written to a memory location, the address of which is specified in the instruction itself. Since all the memory addresses are 16-bits, the address requires two bytes and it is for this reason that this mode is often referred to as extended addressing.

e.g. LDA 28EA

This results in the A-register being loaded with the contents of memory location 28EA (hex) and is expressed as

(A) ← (28EA)

This is a three byte instruction:

3A	Operation
EA	l.s. byte of memory address
28	m.s. byte of memory address

Similarly, the contents of the A-register may be stored in a specified memory location.

e.g. STA 28F2

This results in the contents of the A-register being stored in memory location 28F2 (hex) and is expressed as

(28F2) ← (A)

This is also a three byte instruction:

32	Operation
F2	l.s. byte of memory address
28	m.s. byte of memory address

As will be seen in the next section, the register pair H and L is frequently used to hold a combined 16-bit memory address and consequently two instructions are provided to enable the two registers (H and L) to be loaded using a single instruction and direct addressing:

e.g LHLD 28A2

This results in register L being loaded with the contents of memory location 28A2 and register H being loaded with the contents of the next consecutive memory location, i.e 28A3 in this example. This is therefore expressed as

(L)←(28A2)
(H)←(28A3)

and is a three byte instruction:

2A	Operation
A2	l.s. byte of memory address
28	m.s. byte of memory address

Similarly, SHLD 28AF

This results in the current contents of register L being stored in memory location 28AF and the contents of register H being stored in memory location 28B0. This is expressed as

(28AF)←(L)

(28B0)←(H)

and is again a three byte instruction:

22	Operation
AF	l.s. byte of memory address
28	m.s. byte of memory address

Program Example 4.1 Direct Addressing

The following program uses a combination of immediate and direct addressing first to store immediate data into two consecutive memory locations, then to load register pair HL with this data and finally to store the same data into a pair of different memory locations:

Assembly Instruction	Action
MVI A, FF	(A)←FF (hex)
STA 28A2	(28A2)←(A)
MVI A, EE	(A)←EE (hex)
STA 28A3	(28A3)←(A)
LHLD 28A2	(L)←(28A2) i.e FF (hex)
	(H)←(28A3) i.e EE (hex)
SHLD 28A4	(28A4)←(L)
	(28A5)←(H)

Exercise 4.1

List and code the above program example on a coding form and load it into memory starting at address 2800 (hex) using the 'substitute memory' command. Execute the program using the 'single step' command and examine the contents of the appropriate processor register or memory location using either the 'examine register' or 'substitute memory' command to verify correct execution of each program instruction.

Register Indirect Addressing

Using direct addressing only the A-register may be used to store or load a value to or from memory. Thus if a value was to be stored in a memory location from, say, the B-register, using direct addressing, it would first be necessary to transfer the contents from B to A before the store operation could be performed. A more efficient method, therefore, is to use register indirect addressing since with this mode data may be transferred between any of the processor registers and the system memory.

Using register indirect addressing the operand is either read from or written to the memory location, the address of which is currently stored in the register pair HL. The instruction does not contain the actual memory address itself, therefore, but instead *implies* that the address to be used is currently stored in the HL register pair. The actual memory address is therefore obtained *indirectly*.

e.g MOV A, M

This results in the A-register being loaded with the contents of the memory location whose address is specified in registers H and L. This is represented as

$$(A) \leftarrow ((H)(L))$$

and is a single byte instruction.

7E

Similarly, MOV M, B

This results in the contents of the B-register being transferred to the memory location whose address is in registers H and L. This is represented as

$$((H)(L)) \leftarrow B$$

and is also a single byte instruction

70

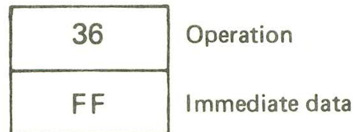
In addition to being able to load immediate data into a specified processor register, there is also an instruction to enable immediate data to be stored directly into a memory location. Again the memory address is stored in the register pair H and L. Thus

MVI M, FF

results in the value FF (hex) being stored in the memory location whose address is in registers H and L. This is represented as

((H)(L)) ← FF (hex)

and is a two byte instruction



Program Example 4.2 Register Indirect Addressing

The following program uses a combination of immediate and register indirect addressing. A memory address is first loaded into registers H and L using immediate addressing and then a value is loaded into this memory location using register indirect addressing. Finally, the value is loaded into two further registers again using register indirect addressing.

Assembly Instruction	Action
LXI H, 28A0	(H) ← 28 (hex)
	(L) ← A0 (hex)
MVI M, AA	((H)(L)) ← AA (hex)
	i.e (28A0) ← AA (hex)
MOV B, M	(B) ← (28A0)
	i.e (B) ← AA (hex)
MOV C, M	(C) ← (28A0)
	i.e (C) ← AA (hex)

Exercise 4.2

List and code the above program example on a coding form and load it into memory starting at address 2800 (hex) using the 'substitute memory' command. Execute the program using the 'single step' command and examine the contents of the appropriate processor register or memory location using either the 'examine register' or 'substitute memory' command to verify correct execution of each program instruction.

Exercise 4.3

Write an assembly language program to:

i) load the A-register with immediate data FF (hex)
ii) store this in memory at location 28FF (hex) using direct addressing
iii) load register pair H and L with immediate data 28FF (hex)
iv) load the B-register with the previously stored data using register indirect addressing
v) transfer the data in the B-register to registers C and D using register addressing.

List and code your program on a coding form and load it into memory starting at address 2800 (hex) using the 'substitute memory' command and examine the contents of the appropriate processor register or memory location to verify correct execution of each program instruction.

SUMMARY

This assignment was concerned with the transfer of data between the microprocessor and system memory. Two addressing modes were introduced: direct addressing and register indirect addressing. Using direct addressing the 16-bit (two byte) memory address is specified in the instruction itself. With register indirect addressing the memory address is implied as being the current contents of the register pair H and L.

DATA MANIPULATION I

ASSIGNMENT 5

Object To see how data is represented in a microcomputer and to use some arithmetic instructions from the data manipulation group.

Approximate time required Three hours

$$\begin{array}{r} 2 \\ +\underline{3} \\ 5 \end{array}$$

5.1 Introduction

Since microprocessors may be used in a wide variety of applications, it is often necessary to represent data within the system in a number of different forms. For example, in some applications a simple unsigned binary representation is adequate but in others it may be advantageous to represent the data in a binary coded decimal form. This assignment, therefore, first describes the different methods available for representing data within a microcomputer and then introduces some arithmetic instructions from the data manipulation group.

5.2 Data Representation

Before considering specific arithmetic instructions, it is necessary to consider the different ways numbers can be represented in a microcomputer. In the Intel 8085, for example, numbers may be represented in unsigned binary, signed binary or binary coded decimal (B C D) form. These are considered in turn.

5.2.1 Unsigned binary

This is the most basic and, for microprocessors, the most common form of number representation. All numbers are assumed positive and each byte is simply the 8-bit binary equivalent of the number.

Thus:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	=	Weighting
0	0	1	0	1	0	1	1	=	43 (decimal)
0	1	0	0	0	1	1	0	=	70 "
1	0	1	0	0	0	0	1	=	161 "
1	1	0	0	1	1	0	0	=	204 "

5.2.2 Signed binary

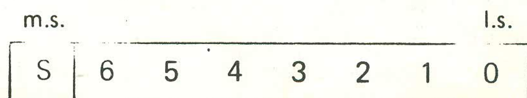
For some applications it is necessary to be able to represent both positive and negative numbers. Moreover, when performing arithmetic operations on the numbers it is necessary to produce the correct signed result. Thus with a signed binary form of number representation one bit, usually the most significant, is used to indicate the sign of the number. The simplest form of signed binary representation is *sign and magnitude* since in this form the most significant bit indicates the sign (0 positive, 1 negative) and the other seven bits the magnitude.

e.g. 0 0011010 = +26

1 1100100 = -100

Unfortunately, however, with this form it is not possible to perform simple arithmetic operations on numbers and automatically produce the correct signed result. It is for this reason that the *two's complement* form of representation is used since with this form, as will be shown later, performing arithmetic operations on two's complement signed numbers automatically produces the correct two's complement signed result.

As before, with two's complement the most significant bit of each number, S, is used as a sign bit:



S = 0 for positive numbers and zero

S = 1 for negative numbers

For positive numbers, the simple binary code is used to represent the magnitude. For negative numbers, however, the magnitude is represented in its two's complement form. To obtain the two's complement of a number, the number is first inverted (complemented) and the resulting number incremented by 1.

```
e.g.      +15   =   0 0 0 0 1 1 1 1
                                        Invert
                    1 1 1 1 0 0 0 0
                                        Increment
          -15   =   1 1 1 1 0 0 0 1

e.g.      +89   =   0 1 0 1 1 0 0 1
                                        Invert
                    1 0 1 0 0 1 1 0
                                        Increment
          -89   =   1 0 1 0 0 1 1 1
```

Note that after the complement process the most significant bit is automatically a 1. For an 8-bit machine, therefore, the range of possible numbers is summarised in the following table:

Decimal Number	Two's Complement Representation
+127	0 1 1 1 1 1 1 1
⋮	⋮
+ 3	0 0 0 0 0 0 1 1
+ 2	0 0 0 0 0 0 1 0
+ 1	0 0 0 0 0 0 0 1
0	0 0 0 0 0 0 0 0
- 1	1 1 1 1 1 1 1 1
- 2	1 1 1 1 1 1 1 0
- 3	1 1 1 1 1 1 0 1
⋮	⋮
-127	1 0 0 0 0 0 0 1
-128	1 0 0 0 0 0 0 0

It is important to note from the table that the maximum positive number that can be represented is +127 and the maximum negative number is -128. When performing arithmetic operations on two's complement signed numbers, therefore, it is important not to exceed this range otherwise incorrect answers will be obtained. For example, (+68) + (+76) will produce an incorrect result since the result (+144) exceeds the maximum possible positive number with 8-bits (+127). Similarly, (-68) — (+76) will also produce an incorrect result since the result (-144) is more than the maximum possible negative number (-128).

5.2.3 Binary Coded Decimal (B C D)

It is sometimes preferable in some applications to use decimal number representation and arithmetic within the micrcomputer. For example, if the input data is from a decimal keypad and the subsequent output data drives a decimal display. Most microcomputers, therefore, provide instructions for performing arithmetic on binary coded decimal (B C D) numbers. B C D representation

is a subset of the hexadecimal system introduced earlier and is summarised in the following table:

Decimal Digit	B C D code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Thus an 8-bit binary number may be used to store two B C D characters:

e.g. 1 0 0 0 0 1 1 0 = 86 (decimal)

0 1 0 1 0 0 0 1 = 51 (decimal)

Summarising, the user may choose one of three different forms of number representation. The type of representation being used, however, is in many ways transparent to the microprocessor since this simply treats the data as an 8-bit binary pattern and it is the responsibility of the programmer (i.e. the user) to process and interpret this data in a form necessary to solve his particular task.

Exercise 5.1

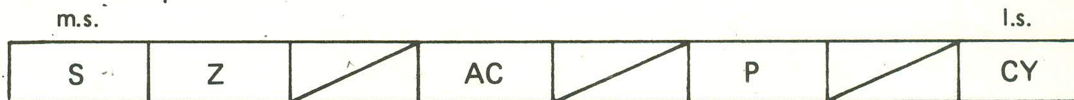
Derive the equivalent decimal numbers from the following 8-bit binary patterns interpreted in turn as unsigned binary, two's complement signed binary and binary coded decimal representations:

0 1 1 0 1 0 0 1
0 0 1 0 1 0 0 0
0 1 1 1 0 1 1 0
1 0 0 0 1 0 0 0
1 0 0 1 0 0 1 1
1 0 0 0 0 1 1 1

5.3 Arithmetic Instructions

The basic arithmetic instructions provided by a microcomputer are add, subtract, increment and decrement. In general these instructions always involve the A-register and either another processor register or a memory location. Since different users may require to interpret the data within the microcomputer in different ways, a microprocessor contains a number of different forms of the above instructions so that users can manipulate their data in the selected manner. In addition, the microprocessor contains a number of flags (status or condition bits) which are either set or cleared depending on the particular arithmetic instruction being carried out. The user is able to use and interpret these flags in order to manipulate his data in the selected way.

The individual flag bits are grouped together to form the flag register, F, and for the Intel 8085 it is made up as follows:



S = Sign Flag	This flag is intended for use when signed numbers are being used. It is set when the result of an arithmetic operation is negative – i.e. the m.s. bit of the A-register is a 1, otherwise it is cleared.
Z = Zero Flag	This flag is set if the result of an arithmetic operation in the A-register is zero, otherwise it is cleared. This is used with transfer of control instructions.
AC = Auxiliary Carry	This flag is intended for use when B C D number representation is being used. It is set when the result of an arithmetic operation produces a carry out from the fourth bit of the A-register, and cleared otherwise.
P = Parity Flag	This is used with the logical operations to be described in the next assignment. The flag is set if the result of a logical operation (AND, OR, XOR) produces an even number of 1's and cleared otherwise.
CY = Carry Flag	This flag is the carry out from the m.s. bit of the A-register. For example, C is set after an ADD instruction if a carry out was generated from the A-register (see below).

Some examples of arithmetic instructions are now given together with their effect on the individual flags in the flag register.

5.3.1 Add Instructions

The addition of two bits is summarised by the following table:

Bit 1	Bit 2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The table shows that the sum can only be 0 or 1 and consequently a carry may be generated which must be added to the next higher order pair of bits. Thus when adding two binary numbers together it is necessary to consider not only each pair of bits but also the carry digit from the previous pair.

The table below is a *truth table* which summarises all the possible combinations of two bits and a carry in and the resulting sum and carry out.

Bit 1	Bit 2	Carry-In	Sum	Carry-Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

e.g.

A	=	1 0 0 1 1 0 1 0	=	154 (decimal)
B	=	0 1 0 1 0 1 1 1	=	87 (decimal)
Carry	=	0 0 1 1 1 1 0 0		
A + B	=	1 1 1 1 0 0 0 1	=	241 (decimal)

All Add instructions use either register, immediate or register indirect addressing and affect all the flag bits.

Register addressing

ADD B

This results in the contents of the B-register being added to the current contents of the A-register. The result is placed in the A-register and the contents of the B-register are unchanged:

$$(A) \leftarrow (A) + (B)$$

This is a single byte instruction:

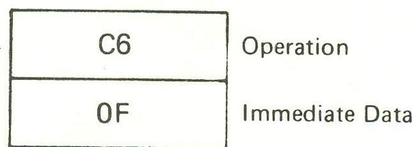
Immediate addressing

ADI 0F

This results in the immediate data 0F (hex) being added to the current contents of the A-register:

$$(A) \leftarrow (A) + 0F$$

This is a two byte instruction



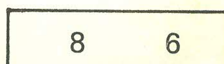
Register Indirect addressing

ADD M

This results in the contents of the memory location whose address is contained in the H and L registers being added to the current contents of the A-register. The result is placed in the A-register:

$$(A) \leftarrow (A) + \big((H)(L)\big)$$

This is a single byte instruction



5.3.2 Subtract Instructions

The subtraction of two binary numbers is similar to addition except that the carry is now replaced by a borrow. The subtraction of two bits is summarised in the following truth table:

Bit 1	Bit 2	Borrow-In	Difference	Borrow-Out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

e.g.

A	=	1 0 0 1 1 0 1 1	=	155 (decimal)
B	=	0 1 0 1 0 1 1 1	=	87 (decimal)
Borrow	=	1 0 0 0 1 0 0 0		
		0 1 0 0 0 1 0 0	=	68 (decimal)

The subtract instructions are identical to the above add instructions except that a subtraction operation is performed in place of the add operation. The corresponding subtraction instructions for each of the above examples are:

SUB	B	90 (hex)
SUI	0F	D6
SUB	M	96

5.3.3 Increment Instructions

The increment instructions use either register or register indirect addressing to increment the contents of either a processor register or a memory location by unity. All flags *except* the carry flag are affected.

Register addressing

INR A

results in the contents of the A-register being incremented by unity:

$$(A) \leftarrow (A) + 1$$

and is a single byte instruction

3	C

In addition to being able to increment the contents of a single register, a number of instructions are provided to increment the combined contents of a pair of registers.

e.g. INX H

This results in the combined contents of register pair H and L being incremented by unity:

$$(H)(L) \leftarrow (H)(L) + 1$$

Again it is a single byte instruction

Register Indirect addressing

INR M

This results in the contents of the memory location whose address is contained in the H and L registers being incremented by unity:

$$((H)(L)) \leftarrow ((H)(L)) + 1$$

This is also a single byte instruction

5.3.4 Decrement Instructions

These instructions are identical to the increment instructions except that the contents of either the processor register(s) or memory location is decremented by unity. The corresponding decrement instructions are:

DCR	A	3D (hex)
DCX	H	2B
DCR	M	35

Program Example 5.1 Unsigned Arithmetic

The following program example uses unsigned binary number representation. The two registers A and B are first loaded with immediate data and their contents are added together. A third number is then subtracted from the contents of A using immediate addressing and finally the new contents of A are decremented by unity.

Assembly Instruction		Action
MVI	A, 53	$(A) \leftarrow 53$ (hex) i.e 83_{10}
MVI	B, 3A	$(B) \leftarrow 3A$ (hex) i.e 58_{10}
ADD	B	$(A) \leftarrow (A) + (B)$
SBI	8C	$(A) \leftarrow (A) - 8C$ (hex) i.e. 140_{10}
DCR	A	$(A) \leftarrow (A) - 1$

Exercise 5.2

List and code program example 5.1 on a coding form and load it into memory. Verify correct execution of the program using the 'single step' command.

Program Example 5.2 Signed Arithmetic

The following program example is the same as the previous example except that the values have been changed to produce a negative result. Two's complement signed number representation is therefore used.

Assembly Instruction		Action
MVI	A, 23	(A) ← 23 (hex) i.e $+35_{10}$
MVI	B, B8	(B) ← B8 (hex) i.e -72_{10}
ADD	B	(A) ← (A) + (B)
SBI	DB	(A) ← (A) - DB (hex) i.e -37_{10}
DCR	A	(A) ← (A) - 1

Exercise 5.3

List and code program example 5.2 on a coding form and load it into memory. Verify correct execution of the program using the 'single step' command taking particular care to interpret the signs of the numbers in the correct way.

SUMMARY

This assignment has introduced three different ways numbers may be represented in a micro-computer: unsigned binary is used for those applications which do not require negative quantities to be considered, two's complement signed binary is used to represent numbers which have mixed signs and binary coded decimal is used for applications which involve decimal data. In addition, some basic arithmetic instructions have been examined together with the meaning of the various status bits which make up the flag register. Further familiarity with data manipulation instructions will be gained in the next assignment.

DATA MANIPULATION II

ASSIGNMENT 6

Object

To examine additional instructions from the data manipulation group: arithmetic instructions which use the carry flag, arithmetic instructions which use B C D number representation and logical instructions.

Approximate time required

Three hours

$$\begin{array}{r} \underline{11} \\ 29 \\ +\ \underline{187} \\ 216 \neq 215 \end{array}$$

6.1 Introduction

The previous assignment was concerned with different methods which may be adopted for representing numbers in a microcomputer. Some basic arithmetic instructions were introduced from the data manipulation group. This assignment is concerned firstly with some additional arithmetic instructions, particularly to note their use of the various flag bits, and secondly the logical instructions which are also part of the data manipulation group.

6.2 Multiprecision Arithmetic Instructions

Although 8 bits are sufficient to represent a data value for many microprocessor applications, some necessitate the use of 16 or more bits. The Intel 8085, therefore, provides a number of arithmetic instructions for manipulating numbers of more than 8 bits.

6.2.1 16-bit Arithmetic

The Intel 8085 provides some data transfer instructions introduced in an earlier assignment, for loading 16-bit (2 byte) immediate data into a register pair – BC, DE and HL. However, there are also instructions for incrementing and decrementing the combined 16-bit contents of a register pair and also for performing double length (16-bit) addition.

e.g.

INX B

0 3

This results in the combined contents of registers B and C being incremented by unity. No flags are affected:

$$(B)(C) \leftarrow (B)(C) + 1$$

e.g.

DCX D

1 B

This results in the combined contents of registers D and E being decremented by unity. No flags are affected:

$$(D)(E) \leftarrow (D)(E) - 1$$

e.g.

DAD B

0 9

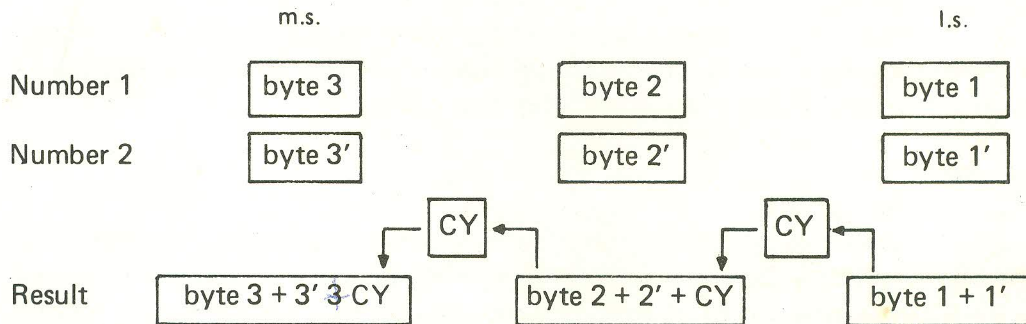
This results in the 16-bit contents of the register pair BC being added to the 16-bit contents of the register pair HL. The result is placed in the register pair HL. Only the carry flag is affected: it is set if there is a carry out from the most significant bit of H during the addition operation otherwise it is reset (i.e cleared).

$$(H)(L) \leftarrow (H)(L) + (B)(C)$$

Note The word 'reset' is often used by people in the microprocessor industry to mean the same thing as 'clear' or 'cleared', i.e 'make/made equal to zero'. It is also sometimes used to mean 'set back to a standard initial state' in which bits may be made equal to 1 or 0 as required. Its use in ordinary English can imply 'set once more'. So treat the word with great caution and avoid its use where possible. This manual uses both 'reset' and 'clear' in order to accustom you to the language in common use.

6.2.2 Multiprecision Arithmetic – The Carry Flag

If more than 16-bit accuracy is required, there are no single instructions available for performing arithmetic operations and instead a number of instructions must be used. For example, consider the addition of two 24-bit (3 byte) numbers. Each number would require three memory locations and also the total addition operation would require three separate 8-bit additions:



The above figure shows that it is also necessary to add the carry bit (CY) when adding together the second and third pair of bytes. The Intel 8085, therefore, provides additional add and subtract instructions which use the carry (borrow) bit.

e.g.

ADC M

8 E

This results in the contents of the memory location whose address is contained in registers H and L and the contents of the CY flag to be added to the contents of the A-register. The result is placed in the A-register and all flags are affected:

$$(A) \leftarrow (A) + ((H)(L)) + (CY)$$

Another example:

SBB M

9 E

This is the same as the above except that a subtraction operation is performed:

$$(A) \leftarrow (A) - ((H)(L)) - (CY)$$

Program Example 6.1 Multiprecision Arithmetic

The following program example adds together the 24-bit (3 byte) number which is stored in the three consecutive memory locations starting at address 28F0 to the 24-bit number which is stored in the three memory locations starting at address 28F3. The 24-bit result replaces the first number:

Assembly Instruction		Action
MVI	H, 28	Initialise HL to contain 28F3
MVI	L, F3	
LDA	28F0	Add 1st pair of bytes
ADD	M	
STA	28F0	
INX	H	Increment HL
LDA	28F1	Add 2nd pair of bytes
ADC	M	
STA	28F1	
INX	H	Increment HL
LDA	28F2	Add 3rd pair of bytes
ADC	M	
STA	28F2	

Exercise 6.1

List and code the above program and load it into memory starting at address 2800. Before the program is run, use the 'substitute memory' command to load the following data into memory starting at address 28F0:

```
28F0  =  6F
         A2        1st Number = 23A26F (hex)
         23

28F3  =  B1
         5E        2nd Number = 5A5EB1 (hex)
         5A
```

Execute the program using the 'single step' command and check the result of each addition as it is performed. The final result should be:

```
28F0  =  20
         01        Result = 7E0120 (hex)
         7E
```

6.3 BCD Arithmetic

As has been mentioned, it is advantageous in some microprocessor applications to represent numbers in binary coded decimal (BCD) form. The Intel 8085, therefore, contains an instruction which, together with the basic arithmetic instructions, allows binary coded decimal data to be manipulated.

Consider the addition of two 2-digit BCD numbers:

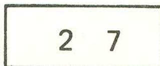
```
Ex.1   Number  1          0 1 1 0 0 0 1 0  =  62   (BCD)
       Number  2          0 0 1 0 0 1 0 1  =  25   (BCD)

   Normal binary sum  =   1 0 0 0 0 1 1 1
   Required BCD sum   =   1 0 0 0 0 1 1 1  =  87   (BCD)

Ex.2   Number  1          0 1 1 1 1 0 0 1  =  79   (BCD)
       Number  2          0 0 0 1 0 1 1 0  =  16   (BCD)

   Normal binary sum  =   1 0 0 0 1 1 1 1
   Required BCD sum   =   1 0 0 1 0 1 0 1  =  95   (BCD)
```

It can be deduced from these two examples that if the normal binary addition of each 4-bit group produces an answer which is 9 or less, the result is correct. If the result of the addition is greater than 9, however, a correction must be made. The correction can be readily seen to be the addition of +6 to the normal binary sum. Consider example 2 above:

Normal binary sum	=	1 0 0 0 1 1 1 1		
Add +6	=	+0 0 0 0 0 1 1 0		
Corrected BCD sum	=	1 0 0 1 0 1 0 1	= 95	(BCD)

The Intel 8085 contains the following instruction which may be used to automatically adjust the result produced by a normal binary addition operation when B C D data is being manipulated:

DAA (Decimal Adjust Accumulator)



This results in the 8-bit number in the A-register being adjusted to form two 4-bit BCD digits by the following process:

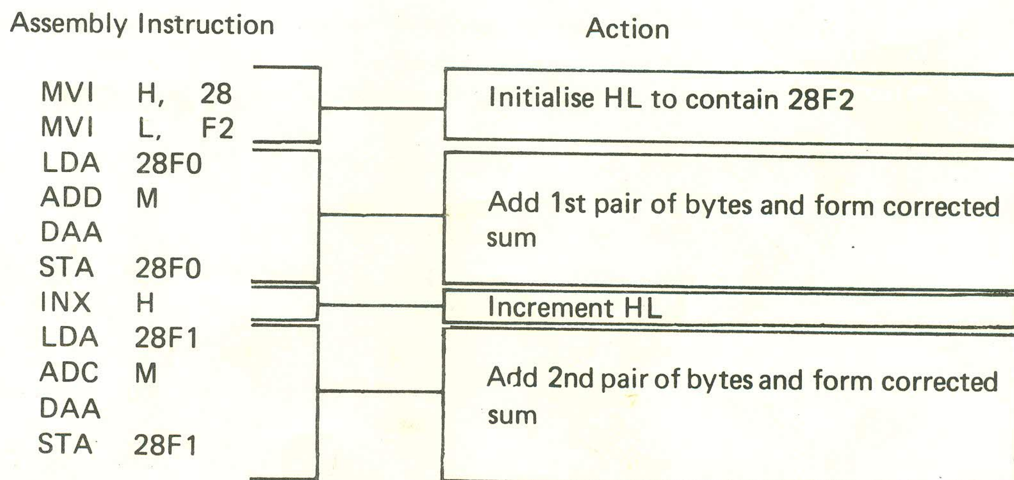
(i) If the value of the l.s. 4-bits of the A-register is greater than 9 *or* if the AC flag is set, 6 is added to the A-register.

(ii) If the value of the m.s. 4-bits of the A-register is now greater than 9 *or* if the CY flag is set, 6 is added to the m.s. 4-bits of the A-register.

Since both halves of the A-register are corrected and also both the carry (CY) and auxiliary (half) carry (AC) are affected, it is clearly possible to perform multiprecision BCD arithmetic using the same instruction. This is illustrated in the following example.

Program Example 6.2 BCD Arithmetic

The following program example adds together two 4-digit (16-bit) BCD numbers, and forms the corrected 4-digit BCD sum. The first number is stored in the two consecutive memory locations starting at address 28F0 and the second is stored in the two locations starting at address 28F2. The result replaces the first number.



Exercise 6.2

List and code the above program and load it into memory starting at address 2800. Load the following data into memory starting at address 28F0 using the 'substitute memory' command:

28F0	=	76	1st number	=	1276	B C D
		12				
28F2	=	99	2nd number	=	6799	B C D
		67				

Execute the program using the 'single step' command and check the result of each uncorrected and corrected addition as it is performed. The final corrected result should be:

28F0	=	75				
		80	Result	=	8075	B C D

6.4 Logical Operations

So far in this and the previous assignment it has been assumed that the data within the microcomputer is always representing a numerical value. In many applications, however, the data may simply be indicating the state of, say, a controlled system. For example, a single binary bit may indicate the state of a control valve: 0 = valve open, 1 = valve closed. Thus the 8-bit binary value 0 1 1 0 0 1 1 1 may mean control valves 1, 2, 3, 6 and 7 are closed whilst control valves 4, 5 and 8 are open. In addition to the arithmetic instructions already introduced a microprocessor has available a number of data manipulation instructions which are primarily included to manipulate non-numeric data of this kind. These are the logical instructions and some examples are now considered.

6.4.1 Logical AND

The logical AND instructions perform the bit-by-bit AND operation between the contents of the A-register and either immediate data or the contents of another processor register or a memory location.

The truth table for the AND function is:

Bit 1	Bit 2	Bit 1 *AND* Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

A typical application of this instruction is to test the state of a specific bit in a group of, say, 8 bits.

For example,

ANI 40

E6	Operation
40	Immediate data

This results in the bit-by-bit logical AND operation between the contents of the A-register and the immediate data 40 (hex). The result is placed in the A-register:

(A) ← (A) *AND* 40

e.g.

(A)	=	01100100	
40	=	01000000	
(A) *AND* 40	=	01000000	P is cleared (odd)

Thus if bit 7 of A is logical 1, the new contents of A are non-zero. Conversely, if bit 7 of A is logical 0, the new contents of A will be zero.

All logical instructions affect the parity flag, P. If the number of 1's in the result (the new contents of the A-register) is *odd* P is cleared (0). If the number of 1's is *even* P is set (1). Thus with the above example, P is reset.

6.4.2 Logical OR

The logical OR instructions are similar to the AND ones except that the bit-by-bit OR operation is performed. The truth table for the OR function is:

Bit 1	Bit 2	Bit 1 *OR* Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

For example, ORA B

B0

The contents of the A-register are OR-ed with the contents of the B-register. The result is placed in the A-register:

(A) (A) *OR* (B)

e.g.

(A)	=	01100100	
(B)	=	10010101	
(A) *OR* (B)	=	11110101	P is set (even)

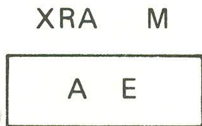
6.4.3 Exclusive OR (XOR)

The exclusive OR operation differs from the normal logical OR insomuch that with the former when both bits are logical 1's, the output is 0. The truth table for the exclusive OR function is:

Bit 1	Bit 2	Bit 1 *XOR* Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

The range of exclusive OR instructions provided is similar to the OR instructions and is used extensively in detecting and correcting errors that may occur when transmitting binary information.

For example, XRA M



This results in the bit-by-bit exclusive OR operation between the contents of the A-register and the contents of the memory location whose address is contained in the H and L registers. The result is placed in the A-register:

$$(A) \leftarrow (A)\ XOR\ ((H)(L))$$

e.g.

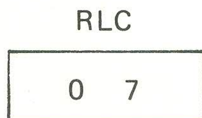
$$\begin{aligned} (A) &= 10011011 \\ ((H)(L)) &= 11001101 \\ (A)\ XOR\ ((H)(L)) &= 01010110 \end{aligned}$$

P is set (even)

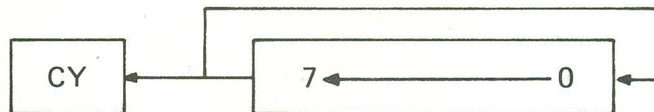
6.4.4 Rotate

The three previous instruction types – logical AND, OR, and XOR – performed the bit-by-bit logical operation between two 8-bit patterns. In addition, the logical group contains instructions to shift (rotate) a binary value left or right one place. This is useful, for example, when performing binary multiplication and division: a left shift is a x2 operation and a right shift is a ÷ 2 operation.

For example, RLC



The contents of the A-register are rotated left one place. The least significant bit and the carry flag are both set to the value shifted out of the most significant bit position:



e.g

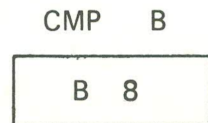
(A) = 1 0 0 1 0 1 1 0

RLC (A) = 0 0 1 0 1 1 0 1 (CY)←1

6.4.5 Compare

The compare instructions are very useful since they compare two values – the contents of the A-register and either immediate data or the contents of a processor register or memory location – without modifying either value. The result of the comparison affects the flags and as will be seen in the next assignment, these may be tested to determine the next operation to be performed.

e.g. CMP B



The contents of the B-register are compared with the contents of the A-register. The Z flag is set to 1 if the contents are equal, the CY flag is set to 1 if the contents of A are less than the contents of B.

Program Example 6.3 Logical Operations

The following program example first loads immediate data into registers A and B and then performs a series of logical operations. First the contents of A and B are compared, the contents of A are then rotated left, the new contents of A are then AND-ed with a constant and finally the resulting contents of A are OR-ed with the contents of B.

Assembly Instruction			Action
MVI	A,	F0	(A)←F0 (hex)
MVI	B,	0F	(B)←0F (hex)
CMP	B		Z←0, CY←0
RLC			(A)←E1, CY←1
ANI	81		(A)←81, CY←0
ORA	B		(A)←8F, CY←0

Exercise 6.3

List and code the above program and load it into memory starting at address 2800. Execute the program using the 'single step' command and check both the contents of the A-register and the contents of the Flag register after each instruction has been executed to verify its correct operation.

SUMMARY

This assignment has introduced some additional arithmetic instructions: the add/subtract with carry for multiple precision arithmetic and the decimal adjust for use with B C D arithmetic. Some familiarity with the logical instructions has been gained and the examples have given some indication of their use.