# Brunel University - Faculty of Technology and Information Systems
# Department of Electronic & Computer Engineering

## Electronic, Electrical & Microprocessor Applications (EE2082A)

## Problem Sheet Solution *(Dr M. K. Darwish)*

1) ***CPU:*** The microprocessor is generally referred to as the central processing unit (CPU). It is responsible for executing arithmetic and logic operations on the binary data being processed and also for controlling the timing and sequencing of operations in the system. The internal structure (the term architecture is used) of a microprocessor depends on the microprocessor concerned. In general, microprocessor architecture will consist of *Registers*, *ALU* and *Control unit*.

   ***Registers:*** These are memory locations within the microprocessor and are used to store information involved in program execution. A microprocessor will contain a group of registers, each type of register having a different function. The number, size and types of registers vary from one microprocessor to another. Some of these registers are: *accumulator*, *status register* (or flag register), *program counter register* (PC), *general-purpose registers*, *memory address register*, *stack pointer register* and *instruction register*.

      ***Accumulator:*** The accumulator register (A) is where data for an input to the arithmetic and logic unit is temporarily stored.

      ***Status register or flag register:*** It contains information concerning the result of the latest process carried out in the arithmetic and logic unit. It contains individual bits, called flags, with each bit having special significance. The status of the latest operation is indicated by each flag being set or reset.

      ***Program counter register (PC) or instruction pointer (IP):*** This is the register used to allow the microprocessor to keep track of its position in a program. This register contains the address of the memory location that contains the next program instruction. As each instruction is executed the program counter register is updated so that it contains the address of the memory location where the next instruction to be executed is stored. The program counter is incremented by one each time so that the microprocessor executes instructions sequentially unless an instruction, such as JUMP or BRANCH, changes the program counter out of that sequence.

      ***General-purpose registers:*** They may serve as temporary storage for data or addresses and they may be used in operations involving transfers between various other registers.

      ***Memory address register:*** is used to store addresses.

      ***Stack pointer register:*** The *stack* is a special area of the memory in which program counter values can be stored when a subroutine part of a program is being used. In some microprocessors the stack pointer points to the first free location in the stack; in other microprocessors it points to the last used location.

      ***Instruction register (IR):*** This stores an instruction. After fetching an instruction from the memory, the CPU stores it in the instruction register. It can then be decoded and used to execute an operation.

   ***ALU:*** acronym for arithmetic and logic unit. The hardware unit of a central processor which is responsible for data manipulation and carries out arithmetic operations of addition, subtraction and logic operations of AND, OR, NOT and EXCLUSIVE-OR.

   ***Control Unit:*** This controls the timing of operations within the system.. It carry out selection and retrieval of instructions from storage in sequence, interpret them and initiate the required operation.

---

2) ***Memory:*** The memory unit stores binary data and takes the form of one or more integrated circuits. The memory elements in a unit consist essentially of a large numbers of storage cells with each cell capable of storing either a 0 or a 1 bit. The storage cells are grouped in locations with each location capable of storing one word. There are two main types of memory used with a microprocessor system: read-only memory (ROM) and random access memory (RAM). The major difference between these two types of memory is that ROM does not lose its stored data when the power is switched off but RAM does.

   ***ROM:*** There are two types of read-only memory; they are termed masked ROM and programmable ROM (PROM). PROM can also occur as EPROM and EEPROM:

      ***Masked ROM:*** Masked ROMs are programmed with the required contents during the manufacture of the integrated circuit. The data can only be read and is used for mixed for fixed programs such as computer operating systems and programs for dedicated microprocessor applications.

***PROM:*** The term *programmable ROM* (PROM) is used for ROM chips that can be programmed by the user. Initially every memory cell has a fusible link which keeps its memory at 0. The 0 is permanently change to 1 by sending a current through the fuse to permanently open it. Once the fusible link has been opened the data is permanently stored in the memory and cannot be further changed.

***EPROM:*** The term *erasable and programmable ROM* (EPROM) is used for ROMs that can be programmed and then altered at a later date. A typical EPROM chip contains a series of cells, small electronic circuits which can store charge. The program is stored by applying voltages to the integrated circuit connection pins and producing a pattern of charged and uncharged cells. The pattern remains permanently in the chip until erased by shining ultraviolet light through a quartz window on the top of the device. This causes all the cells to become discharged and the entire stored program erased. The chip can then be reprogrammed.

***EEPROM:*** Electrically erasable PROM (EEPROM) is similar to EPROM. Erasure is by applying a relatively high voltage rather than using ultraviolet light. Unlike an EPROM, an EEPROM does not have to be completely erased before it can be reprogrammed.

***RAM:*** Temporary data, i.e. data currently being operated on, is stored in a read/write memory referred to as a *random access memory* (RAM). Such a memory can be read or written to. There are two forms of RAM, termed static and dynamic.

***SRAM:*** Static RAM (SRAM) is based on the use of transistor flip-flop elements for temporary storage of 0 or 1 states.

***DRAM:*** Dynamic RAM (DRAM) uses small capacitors as temporary storage elements. This enables there to be more memory elements per square millimetre of chip than SRAM, where each memory element consists of four or six transistor. SRAM does, however, have the advantage of shorter access time than DRAM. Charged leakage from the capacitors means a loss of stored data and so this type of memory has to be refreshed at frequent intervals, typically every 2 ms, if data is not to be lost.

---

3) ***Address bus:*** a bus used only for the transmission of address data. The more lines in the address bus the more memory locations that can be addressed (memory locations = $2^{\text{address lines}}$).

***Data bus:*** is used to carry data and instructions from one unit to another. The number of lines on the bus will be determined by the microprocessor word length.

***Control bus:*** is used in any microprocessor system to control the flow of information between units. No standard format exists for this bus with its function and number of lines varying considerably between different types of processors.

---

4) ***I/O unit:*** Input/output unit communicates with the outside world. I/O includes two types of devices: input and output; these I/O devices are also known as peripherals. The input devices such as keyboard, switches, and A/D converter transfer binary information (data and instructions) from the outside world to the microprocessor. The output devices transfer data from the microprocessor to the outside world. They include devices such as D/A converter, a cathode-ray-tube (CRT), video screen, printer, plotter, etc..

***Interrupt:*** A temporary break in the sequence of a program, initiated externally and causing control to pass to another routine (interrupt service routine). After completion of this routine, the CPU registers are returned from the stack and the main program continues from the point it left off.

***DMA:*** The CPU is usually involved in every data transfer, and this can be very time-consuming. It is possible to add a direct memory access (DMA) feature to most systems where data can be transferred directly between an input/output channel and memory without passing through the CPU.

---

5) ***Machine code:*** is a binary code, which represents instructions or data. Also known as computer code, instruction set, machine instruction code, machine language code.

***Hexadecimal code:*** because it is tedious and prone to errors to write programs in machine code, programs are, for convenience, written in hexadecimal code and entered in a single-board microcomputer by using Hex keys. A monitor program of the system translates the hexadecimal code into machine code.

***Assembly language:*** even though the instructions can be written in hexadecimal code, it is still difficult to understand a program written in hexadecimal numbers. Therefore, each manufacturer of microprocessors has devised a symbolic code for

each instruction, called assembly language. In order to convert an assembly language program into machine code a program called *assembler* is used.

***High-level language:*** Programming languages that are intended to be machine-independent are called high-level languages. These include such languages as FORTRAN, BASIC, PASCAL, C, and C++, all of which have certain sets of rules and draw on symbols and conventions from English. A program called either a *compiler* or an *interpreter* is used to convert the high level language into machine code.

***Comparison between assembly & high-level languages:*** Compilers and interpreters require large memory space because instruction in English requires several machine codes to translate it into binary. On the other hand, there is one-to-one correspondence between the assembly language mnemonics and the machine code. Thus, assembly language programs are compact and require less memory space. They are more efficient than the high-level language programs. The primary advantage of high-level languages is in troubleshooting (debugging) programs. It is much easier to find errors in a program written in a high-level language than to find them in a program written in assembly language.

---

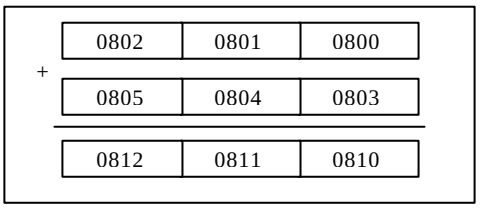
6) Write an 8085 assembly language programme, which adds the contents of memory location 0800 to the contents of memory location 0801 and stores the result in memory location 0802.

```
        ORG=0900      ; Start the program from address 0900
        LXI SP, 0950  ; Reserve address 0950 for the stack use

START   LXI H, 0800   ; Store the address 0800 in registers H & L
        MOV B, M      ; Move the contents of the memory location, which its address is stored in H & L, into reg. B
        INX H         ; Increment the contents of H & L by 1. (now H&L contain 0801)
        MOV A, M      ; Move the contents of memory location 0801 into the accumulator (reg.A)
        ADD B         ; Add the contents of reg. B to the contents of reg. A. The answer goes automatic into reg.A
        INX H         ; Increment the contents of H & L by 1. (now H&L contain 0802)
        MOV M, A      ; Store the answer (which is in 'A') into memory location 0802
        END           ; End the program
```

---

7)




```
        ORG=0900      ; Start the program from address 0900
        LXI SP, 0950  ; Reserve address 0950 for the stack use

START   LXI H, 0800   ; Put the address 0800 in registers H & L
        MOV A, M      ; Move the contents of the memory location, which its address is stored in H & L, into reg. A
        LXI H, 0803   ; Put the address 0803 in registers H & L
        ADD M         ; Add the contents of location 0800 to the contents of reg. A. The answer is in 'A'.
        LXI H, 0810   ; Put the address 0810 in registers H & L.
        MOV M, A      ; Move the answer of (0800)+(0803) which is in 'A' into 0810.

        LXI H, 0801   ; Put the address 0801 in registers H & L
        MOV A, M      ; Move the contents of the memory location, which its address is stored in H & L, into reg. A
        LXI H, 0804   ; Put the address 0804 in registers H & L
        ADC M         ; Add (0801) to the contents of 'A' pluse the contents of the carry flag. The answer is in 'A'.
        LXI H, 0811   ; Put the address 0811 in registers H & L.
        MOV M, A      ; Move the answer of (0801)+(0804) which is in 'A' into 0811.

        LXI H, 0802   ; Put the address 0802 in registers H & L
        MOV A, M      ; Move the contents of the memory location, which its address is stored in H & L, into reg. A
        LXI H, 0805   ; Put the address 0805 in registers H & L
        ADC M         ; Add (0802) to the contents of 'A' pluse the contents of the carry flag. The answer is in 'A'.
        LXI H, 0812   ; Put the address 0812 in registers H & L.
        MOV M, A      ; Move the answer of (0802)+(0805) which is in 'A' into 0812.
```

```
        MVI A, 00       ; Put 00 into the 'A' and add it to 00 plus whatever in the carry flag.
        ACI 00
        LXI H, 0813
        MOV M, A        ; Put the answer in 0813
```

---

8)

```
        ORG=0900        ; Start the program from address 0900
        LXI SP, 0950    ; Reserve address 0950 for the stack use

START   LXI H, 0800     ; Put the address 0800 in registers H & L
        MVI A, F0       ; Move immediate 11110000 into 'A' register.
        ANA M           ; LOGIC AND the contents of 0800 with F0.
        MOV B, A        ; Store the answer into B register.
        INX H           ; Point to the number in 0801.
        MVI A, F0       ; Move immediate 11110000 into 'A' register.
        ANA M           ; LOGIC AND the contents of 0801 with F0.
        ADD B           ; Add the contents of 'B' to the contents of 'A'.
        LXI H, 0810     ; Point to the number in 0810.
        MOV M, A        ; Move the answer into 0810.
        MVI C, 00       ; Move immediate 00 into 'C' register.
        ADC C           ; Add 'C' + 00 + carry (in case there is a carry from the last addition).
        INX H           ; Point to the number in 0811
        MOV M, A        ; Move the 2nd part of the answer into 0811
        STOP            ; Stop the program.
```

---

9)

```
        ORG=0900        ; Start the program from address 0900
        LXI SP, 0950    ; Reserve address 0950 for the stack use

START   LXI H, 0800     ; Store the address 0800 in registers H & L
        MOV A,M         ; Move the number from Memory location 0800 into 'A' register.
        RAR             ; Rotate the contents of 'A' register.  The lsb is now in the 'CY'
        JC ODD          ; Jump to subroutine "ODD" if the carry contains '1' (i.e. the number is odd).
EVEN    LXI H, 0810     ; Store the address 0810 in registers H & L.
        MVI M, FF       ; Move immediate FF into memory location 0810.
        HLT             ; Halt the program.
ODD     LXI H, 0810     ; Store the address 0810 in registers H & L.
        MVI M, 00       ; Move immediate 00 into memory location 0810.
        STOP            ; Stop the program.
```

---

10)

```
        ORG=0900        ; Start the program from address 0900
        LXI SP, 0950    ; Reserve address 0950 for the stack use

START   LXI H, 0800     ; Put the address 0800 in registers H & L
        MOV A, M        ; Move the value of the 1st sensor in furnace 1 into 'A'.
        LXI H, 0810     ; Point to the memory location of the 1st sensor in furnace 2.
        MOV B, M        ; Move the value of the 1st sensor in furnace 2 into 'B'.
        CMP B           ; subtract the value of the two temperature from each other (0800-0810)
        JM EMERG        ; If the value of 0810 is > than the value of 0800, go to EMERG.

        LXI H, 0801     ; Put the address 0801 in registers H & L.
        MOV A, M        ; Move the value of the 2nd sensor in furnace 1 into 'A'.
        LXI H, 0811     ; Point to the memory location of the 2nd sensor in furnace 2..
        MOV B, M        ; Move the value of the 2nd sensor in furnace 2 into 'B'.
```

```
        CMP B       ; subtract the value of the two temperature from each other (0801-0811)
        JM EMERG    ; If the value of 0811 is > than the value of 0801, go to EMERG.

        LXI H, 0802 ; Put the address 0802 in registers H & L.
        MOV A, M    ; Move the value of the 3rd sensor in furnace 1 into 'A'.
        LXI H, 0812 ; Point to the memory location of the 3rd sensor in furnace 2..
        MOV B, M    ; Move the value of the 3rd sensor in furnace 2 into 'B'.
        CMP B       ; subtract the value of the two temperature from each other (0802-0812)
        JM EMERG    ; If the value of 0812 is > than the value of 0802, go to EMERG.

        LXI H, 0803 ; Put the address 0803 in registers H & L.
        MOV A, M    ; Move the value of the 4th sensor in furnace 1 into 'A'.
        LXI H, 0813 ; Point to the memory location of the 4th sensor in furnace 2..
        MOV B, M    ; Move the value of the 4th sensor in furnace 2 into 'B'.
        CMP B       ; subtract the value of the two temperature from each other (0803-0813)
        JM EMERG    ; If the value of 0813 is > than the value of 0803, go to EMERG.

        LXI H, 0804 ; Put the address 0804 in registers H & L.
        MOV A, M    ; Move the value of the 5th sensor in furnace 1 into 'A'.
        LXI H, 0814 ; Point to the memory location of the 5th sensor in furnace 2..
        MOV B, M    ; Move the value of the 5th sensor in furnace 2 into 'B'.
        CMP B       ; subtract the value of the two temperature from each other (0804-0814)
        JM EMERG    ; If the value of 0814 is > than the value of 0804, go to EMERG.

        LXI H, 0850 ; The program will only come to this location if all temperatures in furnace 1 are
        MVI M, 01   ; larger than the corresponding ones in furnace 2.
        JMP START   ; Repeat the program.

EMERG   LXI H, 0850 ; The program will come here if any of the temperatures in furnace 2 is
        MVI M, FF   ; larger than the corresponding one in furnace 1.
        RET         ; Return from the subroutine.
```

---

11)

```
        ORG=0800    ; Start the program from address 0800
        LXI SP, 0850; Reserve address 0850 for the stack use
        MVI A,01    ; Put number 01 into the accumulator
        OUT 10      ; Configure port 11 as output port

START   MVI B,00    ; Reset the contents of B register to 0
        IN 12       ; Take the votes from port 12
        RLC         ; Push the msb (bit 7) into the carry flag
        CC YES      ; Go to YES if bit 7 is '1'
        RLC         ; Push the original bit 6 into the carry flag
        CC YES      ; Go to YES if the original bit 6 is '1'
RLC     ; Push the original bit 5 into the carry flag
        CC YES      ; Go to YES if the original bit 5 is '1'
        RLC         ; Push the original bit 4 into the carry flag
        CC YES      ; Go to YES if the original bit 4 is '1'
        RLC         ; Push the original bit 3 into the carry flag
        CC YES      ; Go to YES if the original bit 3 is '1'
        RLC         ; Push the original bit 2 into the carry flag
        CC YES      ; Go to YES if the original bit 2 is '1'
        RLC         ; Push the original bit 1 into the carry flag
        CC YES      ; Go to YES if the original bit 1 is '1'
        RLC         ; Push the original bit 0 into the carry flag
        CC YES      ; Go to YES if the original bit 0 is '1'
        MOV A,B     ; move the count from B to A
        CPI 04      ; Compare it with 4
        JZ CHAIR    ; If the number is 4 go and find out the chair's vote
        JP AGREE    ; If what is in B reg. is larger than four then go to AGREE
        JM DSAGR    ; If what is in B reg. is less than four then go to DSAGR
```

```
        HLT

CHAIR   IN 12           ; Take the votes from port 12
        RLC             ; Push the msb (bit 7) into the carry flag
        JC AGREE        ; If there is '1' in bit 7 then go to AGREE
        JMP DSAGR       ; Otherwise go to DSAGR

YES     INR B           ; Increment the counter in B
        RET
AGREE   MVI A,FF        ; Send FF to port 11
        OUT 11
        JMP START
DSAGR   MVI A,00        ; Send 00 to port 11
        OUT 11
        JMP START
        HLT
```

---

12)

```
        ORG=0800        ; Start the program from address 0800
        LXI SP, 0850    ; Reserve address 0850 for the stack use
        MVI A,01        ; Put number 01 into the accumulator
        OUT 10          ; Configure port 11 as output port

START   MVI A, 24       ; Put number 24 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 24 to port 11
        MVI E, 0F       ; Put 15 into E register so that the delay routine is repeated 15 times
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 44       ; Put number 44 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 24 to port 11
        MVI E, 01       ; Put 1 into E register so that the delay routine is executed only once
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 84       ; Put number 84 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 84 to port 11
        MVI E, 01       ; Put 1 into E register so that the delay routine is executed only once
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 86       ; Put number 86 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 86 to port 11
        MVI E, 01       ; Put 1 into E register so that the delay routine is executed only once
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 81       ; Put number 81 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 81 to port 11
        MVI E, 0F       ; Put 15 into E register so that the delay routine is repeated 15 times
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 82       ; Put number 82 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 82 to port 11
        MVI E, 0F       ; Put 15 into E register so that the delay routine is repeated 15 times
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, 84       ; Put number 84 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of 84 to port 11
        MVI E, 01       ; Put 1 into E register so that the delay routine is executed only once
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)

        MVI A, C4       ; Put number C4 into the accumulator (equivalent to the 1st line in the table)
        OUT 11          ; send the binary equivalent of C4 to port 11
        MVI E, 01       ; Put 1 into E register so that the delay routine is executed only once
        CALL DELAY      ; Call DELAY routine (which takes 1sec x the contents of E)
```

```
        JMP START     ; Go back to START and repeat the program

DELAY   MVI A, 03     ; Put 03 into 'A' register
D1      MVI B, BA     ; Put BA into 'B' register
D2      MVI C, FF     ; Put FF into 'C' register
D3      DCR C         ; Decrement C by 1
        JNZ D3        ; If the contents of A is not zero go back to D3
        DCR B         ; Decrement B by 1
        JNZ D2        ; If the contents of B is not zero go back to D2
        DCR A         ; Decrement A by 1
        JNZ D1        ; If the contents of A is not zero go back to D1
        DCR E         ; Decrement E by 1 (E contains either '1' or 'F')
        JNZ DELAY     ; If the contents of E is not zero go back to DELAY
        RET           ; Return from the subroutine
```

---

13)

i) The current in the msb = 10V / 10 kΩ = 1 mA

ii) The resistor in the lsb = 1280 kΩ
The current in the lsb = 10V / 1280 kΩ = 7.8 μA

iii) When both lsb and msb are on the total current is simply the current in the msb plus the current in the lab branches = 1 mA + 7.8 μA = 1.0078 mA

iv) The maximum current will be achieved when all switches are on:
(10V/10 kΩ) + (10V/20 kΩ) + (10V/40 kΩ) + (10V/80 kΩ) + (10V/160 kΩ) + (10V/320 kΩ) + (10V/640 kΩ) + (10V/1280 kΩ) = 1.9921875 mA

---

14)