



CLASS: T.E. E &TC Engg.

SUBJECT: MC

EXPT. NO.: 1

DATE: 12/07/2024

TITLE: Memory Block Transfer

PROBLEM STATEMENT:

Write a program for non-overlapping (10 Bytes from 20H to 40H onwards) and overlapping (10 Bytes from 20 H to 25H onwards) memory block transfer.

OBJECTIVE:

- a. To study addressing modes
- b. To study concepts of overlapped and non-overlapped Memory.

S/W PACKAGES USED:

Keil IDE, Windows 7

THEORY

1. Addressing Modes of 8051

Addressing mode is a way to address an operand. Operand means the data we are operating upon (in most cases source data).

i. Immediate Addressing Mode

Let's begin with an example.

MOV A, #6AH

In general, we can write MOV A, #data. This addressing mode is named as "*immediate*" because it transfers an 8-bit data immediately to the accumulator (destination operand). This instruction is of two bytes and is executed in one cycle. This instruction is of two bytes and is executed in one machine cycle. The '#' symbol before 6AH indicates that operand is a data (8 bit). If '#' is not present then the hexadecimal number would be taken as address.

ii. Direct Addressing Mode

Here the address of the data (source data) is given as operand. Let's take an example.

MOV A, 04H

Here 04H is the address of register 4 of register bank0. When this instruction is executed, whatever data is stored in register 04H is moved to accumulator. Let register 04H holds the data 1FH. So, the data 1FH is moved to accumulator. This is a 2-byte instruction which requires 1 machine cycle to complete.

Note: We have not used '#' in direct addressing mode, unlike immediate mode. If we had used '#', the data value 04H would have been transferred to accumulator instead of 1FH.



iii. Register Direct Addressing Mode

In this addressing mode we use the register name directly (as source operand). An example is shown below.

MOV A, R4

At a time, registers can take value from R0, R1...to R7. There are 32 such registers. There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7. At a time only one register bank can be selected. Selection of register bank is made possible through a Special Function Register (SFR) named Program Status Word (PSW). Register banks are selected using PSW.3 and PSW.4. These two bits are known as register bank select bits as they are used to select register banks. The above instruction is 1 byte and requires 1 machine cycle for complete execution. This means using register direct addressing mode can save program memory.

iv. Register Indirect Addressing Mode

In this addressing mode, address of the data (source data to transfer) is given in the register operand.

MOV A, @R0

Here the value inside R0 is considered as an address, which holds the data to be transferred to accumulator. If R0 holds the value 20H, and we have a data 2F H stored at the address 20H, then the value 2FH will get transferred to accumulator after executing this instruction. This is a single byte instruction and requires 1 machine cycle for complete execution.

Note: Only R0 and R1 are allowed to form a register indirect addressing instruction. In other words, programmer must make any instruction either using @R0 or @R1. All register banks are allowed.

v. Indexed Addressing Mode

Let's see two examples first.

MOVC A, @A+DPTR and MOVC A, @A+PC

Where DPTR is data pointer and PC is program counter (both are 16-bit registers). Let's take the first example.

MOVC A, @A+DPTR

The source operand is @A+DPTR. It is nothing but adding contents of DPTR with present content of accumulator. This addition will result a new data which is taken as the address of source data (to transfer). The data at this address is then transferred to accumulator. The opcode for the instruction is 93H. For example, DPTR holds the value 01FE, where 01 is located in DPH (higher 8 bits) and FE is located in DPL (lower 8 bits). Accumulator now has the value 02H. A 16-bit addition is performed and now 01FE H+02 H results in 0200 H. Whatever data is in 0200 H will



get transferred to accumulator. The previous value inside accumulator (02H) will get replaced with new data from 0200H. This is a 1-byte instruction with 2 cycles needed for execution.

The other example **MOVC A, @A+PC** works the same way as above example. The only difference is, instead of adding DPTR with accumulator, here data inside program counter (PC) is added with accumulator to obtain the target address.

2. Overlapped and Non-overlapped Memory

In non-overlapping method, address of source is totally different from address of destination. So, no memory area is common in both arrays (blocks). Hence the name non overlapped memory data transfer.

The memory blocks are said to be overlapped if some of the memory locations are common for both the blocks.

In both methods, take the source block data from user.

e.g. A) Non overlap block data transfer:

Input:

Source (from 20H memory onwards): 10h, 20h, 30h, 40h, 33h, 0ffh, 44,55h, 23h, 45h

Destination (from 40H memory onwards): 00h, 00h, 00h, 00h, 00h, 00h, 00h,00h, 00h, 00h

Output:

Destination (from 40H memory onwards): 10h, 20h, 30h, 40h, 33h, 0ffh, 44,55h, 23h, 45h

B) Overlap block data transfer:

Input:

Source (from 20H memory onwards): 10h, 20h, 30h, 40h, 33h, 0ffh, 44h, 55h, 23h, 45h

Destination (from 25H memory onwards): 0ffh, 44h,55h, 23h, 45h, 00h, 00,00h, 00h, 00h

Output:

Memory form 20H onwards: 10h, 20h, 30h, 40h, 33h, 10h, 20h, 30h, 40h, 33h, 0ffh, 44h, 55h, 23h, 45h

3. Algorithm

3.1 Algorithm for Non-overlapped memory transfer

1. Start
2. Assign i range from 0 to 9
3. Assign *ptr (20H) as pointer to source memory.
4. Assign *ptr1 (40H) as pointer to destination memory.
5. Increment value of i.
6. Copy data from location *ptr+i to *ptr1+i.



7. If i less than 10 then go to step 5.
8. Stop

3.2 Algorithm for Overlapped memory transfer

1. Start
2. Assign i range from 0 to 9.
3. Assign *ptr (29H) as pointer to source memory.
4. Assign *ptr1 (2EH) as pointer to destination memory.
5. Increment value of i.
6. Copy data from location *ptr-i to *ptr-i.
7. If i less than 10 then go to step 5.
8. Stop

4. Source code (Attach separate Sheet)

5. Result & Conclusion

6. References:

- a. Mazidi, 8051 microcontroller & embedded system 3rd Edition, Pearson
- b. Datasheet of 8051 microcontroller



A. Source code :

1. Source code for Non-overlapped memory transfer :

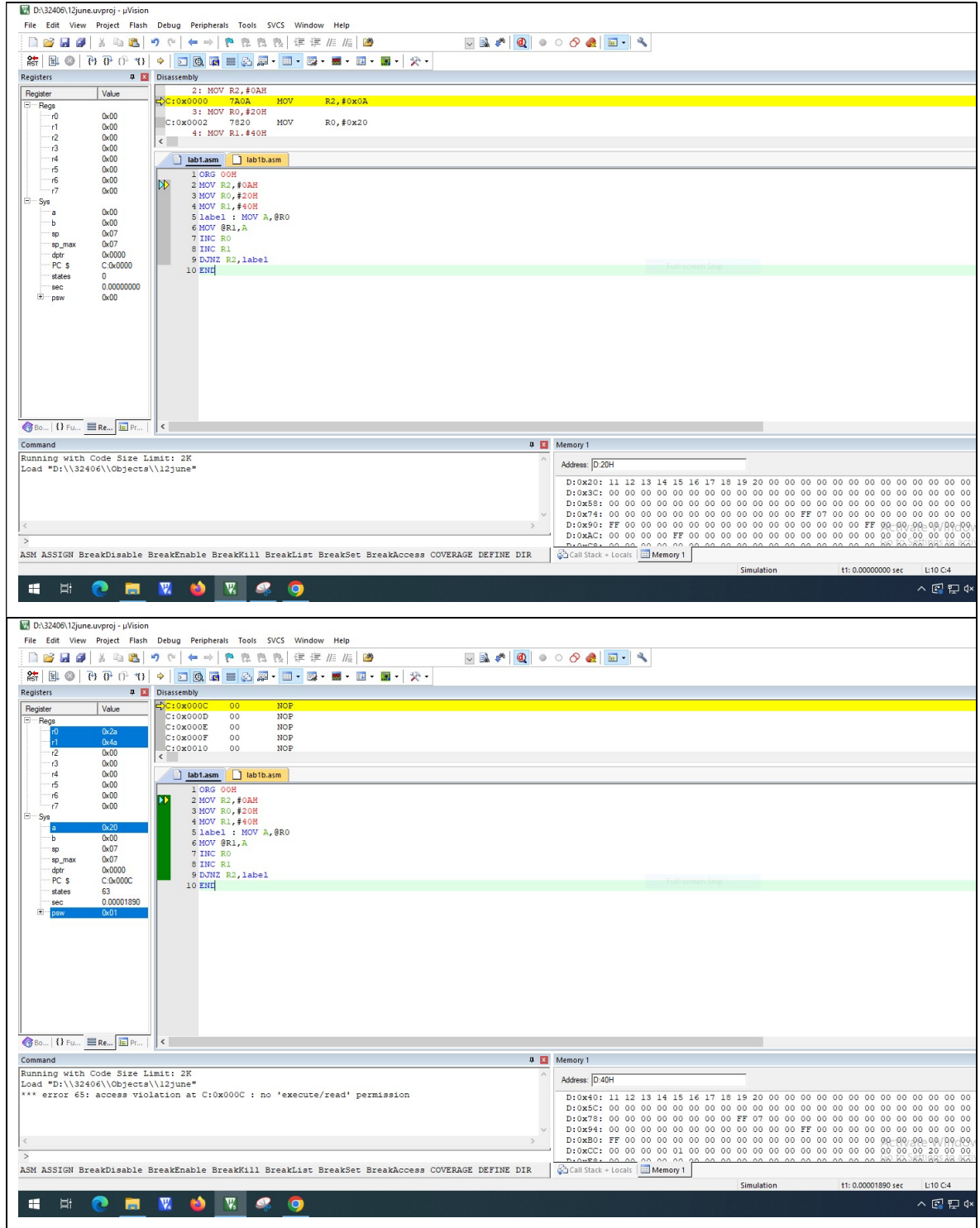
```
ORG 00H
MOV R2,#0AH
MOV R0,#20H
MOV R1,#40H
label : MOV A,@R0
MOV @R1,A
INC R0
INC R1
DJNZ R2,label
END
```

2. Source code for Overlapped memory transfer:

```
ORG 00H
MOV R2,#0AH
MOV R0,#29H
MOV R1,#2EH
label : MOV A,@R0
MOV @R1,A
DEC R0
DEC R1
DJNZ R2,label
END
```

B. Output :

1. Source code for Non-overlapped memory transfer :



The first screenshot shows the assembly code for a program that transfers data from memory address 0x0000 to 0x0020. The code is as follows:

```

1 ORG 00H
2 MOV R2,#0AH
3 MOV R0,#20H
4 MOV R1,#40H
5 label: MOV A,R0
6 MOV @R1,A
7 INC R0
8 INC R1
9 DJNZ R2,label
10 END

```

The second screenshot shows the same code after execution. The registers are updated, and the memory dump shows the data being transferred. The error message indicates an access violation at address 0x0000C due to no 'execute/read' permission.

