ASSIGNMENT NO. 2

AIM :- Write X86/64 ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment

APPARATUS :

- Core 2 duo/i3/i5/i7 - 64bit processor
- OS – ubuntu 32bit/64bit OS
- Assembler used –nasm (the netwide assembler)
- Editor Used – gedit

THEORY :

STRING INSTRUCTION

1. MOVSB/MOVSW

The string instructions help operations on sequences of bytes or words. No one take an explicit operand; instead, they all work implicitly on the source and/or destination strings. The current element (byte or word) of the source string is at DS : SI, and the current element of the destination string is at ES:DI. Each instruction works on one element and then automatically adjusts RSI and/or RDI; if the Direction flag is clear, then the index is incremented, otherwise it is decremented (when working with overlapping strings it is sometimes necessary to work from back to front, but usually you should leave the Direction flag clear and work on strings from front to back).

To work on an entire string at a time, each string instruction can be accompanied by a repeat prefix, either REP or one of REPE and REPNE (or their synonyms REPZ and REPNZ). These cause the instruction to be repeated the number of times in the count register, RCX; for REPE and REPNE, the Zero flag is tested at the end of each operation and the loop is stopped if the condition (Equal or Not Equal to zero) fails.

The MOVSB and MOVSW instructions have the following forms:

```
        MOVSB
or
        REP MOVSB

        MOVSW
or
        REP MOVSW
```

2. STOSB/STOSW

STOSB stores the byte in AL at [ES:DI] or [ES:EDI], and sets the flags accordingly. It then increments or decrements (depending on the direction flag: increments if the flag is clear, decrements if it is set) DI (or EDI).

STOSB - Store AL at address ES:(E)DI

STOSW - Store AX at address ES:(E)DI

3. STD/CLD

STD sets the direction flag. The STD instruction puts the DF flag in 1.

Syntax: STD

CLD clear the direction. This instruction turns off the corresponding bit to the address flag.

Syntax: CLD

ALGORITHM:

1. <u>Non overlapped</u> block transfer <u>without string instruction</u>

1. Start
2. Declare & initialize the variables in .data section.
3. Declare uninitialized variables in .bss section.
4. Declare Macros for Read, print and exit operation.

5. Define source block in data section.
6. Point rsi to source block
7. call display_block_ procedure
   7a. Initialize count
   7b. Move content of rsi in accumulator.
   7c. Push rsi
   7d. Call display_procedure to display number
   7e. Print space
   7f. Pop rsi and increment rsi and decrement count.
   7g. Repeat step b to f until count =0.
8. Repeat step 7 for destination_block
9. Call Block_transfer procedure
   9a. Mov content of rsi into accumulator and increment rsi
   9b. Mov content of accumulator into rdi and increment rdi.
   9c.Repeat a to b until count is 0.
10. Repeat step 3 for displaying source block and destination_block.
11. Using Macro terminate the process.
12. Stop.

2. <u>Non overlapped</u> block transfer <u>with string instruction</u>

1. Start
2. Declare and initialized the variable in .data section
3. Declare uninitialized variables in .bss section.
4. Declare Macros for Read, print and exit operation.
5. Define source block in data section.
6. Point rsi to source block
7. call display_block_ procedure
   7a. Initialize count
   7b. Move content of rsi in accumulator.

7c. Push rsi

7d. Call display_procedure to display number

7e. Print space

7f. Pop rsi and increment rsi and decrement count.

7g. Repeat step b to f until count =0.

8. Repeat step 7 for destination_block
9. Call Block_transfer procedure

   9a. Used MOVSB instruction to transfer the content and clear direction flag..

   9b.Repeat b until count is 0.
10. Repeat step 3 for displaying source block and destination_block.
11. Using macro terminate procedure.
12. Stop

3. <u>Overlapped</u> block transfer <u>without string instruction</u>

1. Start
2. Declare and initialized the variable in .data section
3. Declare uninitialized variables in .bss section.
4. Macros for Read, print and exit operation.
5. Define source block in data section.
6. Point rsi to source block
7. call display_block_ procedure

   7a. Initialize count

   7b. Move content of rsi in accumulator.

   7c. Push rsi

   7d. Call display_procedure to display number

   7e. Print space

   7f. Pop rsi and increment rsi and decrement count.

   7g. Repeat step b to f until count =0.

8. Repeat step 7 for destination_block
9. Call Block_transfer procedure
   9a. Set the rsi and rdi to specific location for overlapping the block.
   9b. Mov content of rsi into accumulator and decrement rsi
   9c. Mov content of accumulator into rdi and decrement rdi.
   9d. loop b to c until count is 0.
10. Repeat step 3 for displaying source block and destination_block.
11. Using macro terminate procedure.
12. Stop

4. Overlapped block transfer with string instruction

1. Start
2. Declare and initialized the variable in .data section
3. Declare uninitialized variables in .bss section.
4. Declare Macros for Read, print and exit operation.
5. Define source block in data section.
6. Point rsi to source block
7. call display_block_ procedure
   7a. Initialize count
   7b. Move content of rsi in accumulator.
   7c. Push rsi
   7d. Call display_procedure to display number
   7e. Print space
   7f. Pop rsi and increment rsi and decrement count.
   7g. Repeat step b to f until count =0.
8. Repeat step 7 for destination_block
9. Call Block_transfer procedure
   9a. Used MOVSB instruction to transfer the content and set direction flag..
   9b.Repeat b until count is 0.

Repeat step 3 for displaying source block and destination_block.

10. Using macro terminate procedure.
11. Stop

CONCLUSION: