String operations

Chapter 7

String Operations

- Used for string and array processing, with the goal of writing efficient code.
- The x86 instruction set has five groups of instructions for processing arrays of bytes and words.
- Although they are called string primitives, they are **not limited to character arrays**
- Next slide show these instruction and registers associated with them.

String Operations

Instruction	Description
MOVSB, MOVSW	Move string data - Copy data from memory at DS:SI to memory at ES:DI and update SI and DI to point to the next locations.
LODSB, LODSW	Load into accumulator (AL or AX) the contents of memory at DS:SI, and update SI to point to the next location.
STOSB, STOSW	Store the accumulator (AL or AX) contents into memory at ES:DI, and update DI to point to the next location.
CMPSB, CMPSW	Compare strings - Compare the contents of two memory locations at DS:SI and ES:DI, and update SI and DI to point to the next locations. Flags are also updated accordingly.
SCASB, SCASW	Scan string - Compare the accumulator (AL or AX) to contents of memory at ES:DI, and update DI to point to the next location. Flags are also updated accordingly.

Update in SI and DI

- Either one or both of SI/ DI are updated with string instructions.
- If Direction Flag (DF)=0
  - In byte mode SI+1, DI+1, and in word mode SI+2, DI+2
  - That is, pointers move forward towards the end of block.
- If DF=1
  - In byte mode SI-1 , DI-1, and in word mode SI-2, DI-2
  - That is, pointers move backward towards the start of block.

MOVS

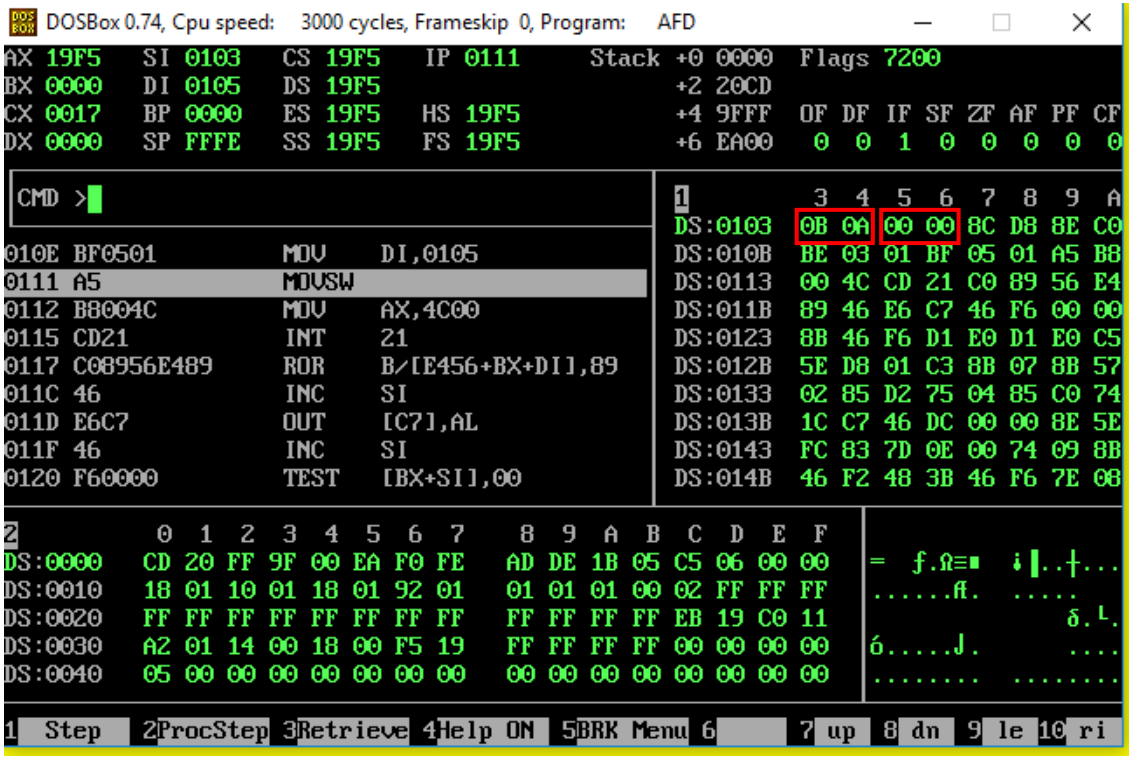
- The MOVSB and MOVSW instructions copy data from the memory location pointed to by DS:SI to the memory location pointed to by ES:DI.
- The two registers are either incremented or decremented automatically (based on the value of the Direction flag)
  - MOVSB Move (copy) bytes
  - MOVSW Move (copy) words

Example: MOVS

```
;Example 1: copy one word from Data segment to data segment
[org 0x0100]

jmp start

num1: dw 0A0Bh
num2: dw 0 ; copy num1 in num2

start:
;source segment is DS
;setting destination segment, i.e making ES= datasetgment
mov ax, ds
mov es, ax ;
mov si, num1 ; si points to offset of source in source segment
mov di, num2 ;si points to offset of destination  in destination segment
movsw ; not that we have directly moved a number from memory to memory


mov ax, 0x4c00
int 21h
```

Before and after execution of MOVSW

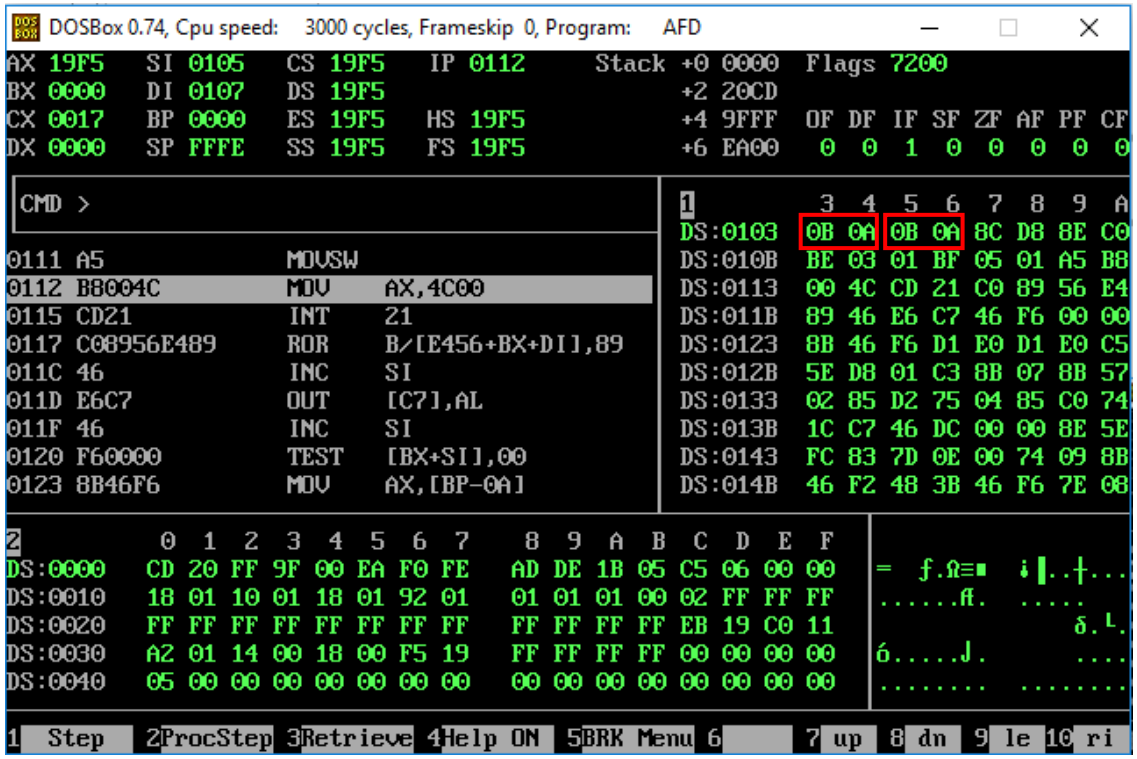
- As DF is zero, SI and DI are incremented by 2.
- Data is moved to destination



before

Before and after execution of MOVSW

- As DF is zero, SI and DI are incremented by 2.
- Data is moved to destination



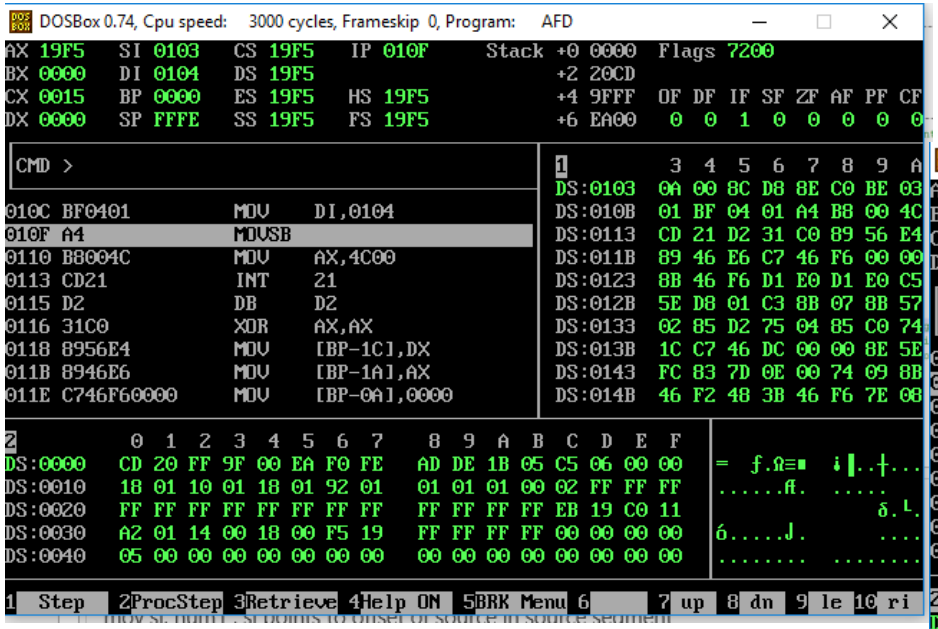
after

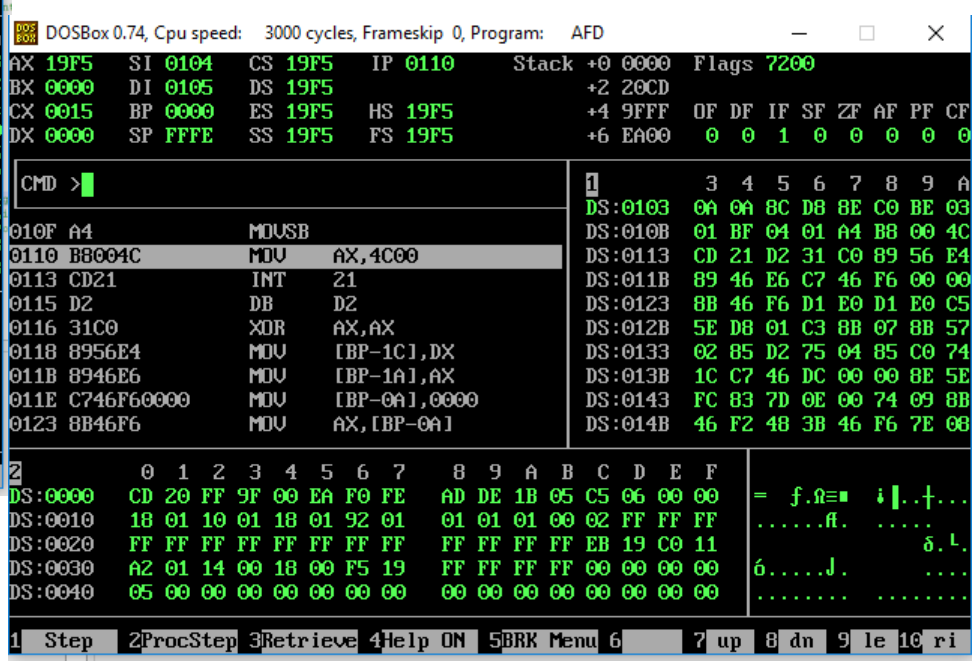
Example: MOVSB

```
;Example 2: copy one byte from Data segment to data segment
[org 0x0100]

jmp start

num1: db 0Ah
num2: db 0 ; copy num1 in num2

start:
;source segment is DS
;setting destination segment, i.e making ES= datasetgment
mov ax, ds
mov es, ax ;
mov si, num1 ; si points to offset of source in source segment
mov di, num2 ;si points to offset of destination  in destination segment
movsb ; not that we have directly moved a number from memory to memory


mov ax, 0x4c00
int 21h
```

Before and after execution of MOVSB

- As DF is clear, SI and DI are incremented by 1.
- Data is moved to destination





REP prefix

- REP repeats the following string instruction CX times.
- The use of CX is implied with the REP prefix
- CX is decremented after each repetition
- The decrement in CX doesn’t affect any flags

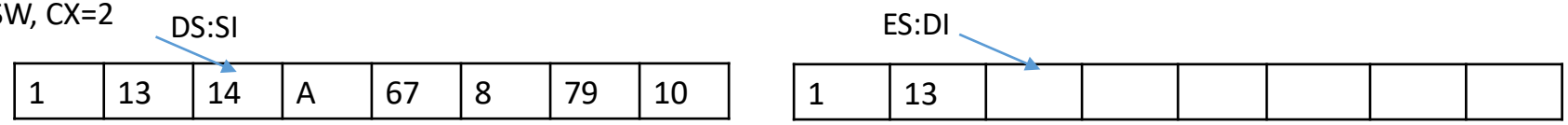
MOVS with REP

- We can used MOVS and REP together to copy an chunk of data from source to destination
- Depending on DF, Data can be copied from end to start of block or start to end of block
- Steps to mov n words from source to destination from start to end
  - Clear DF so that data is copied from start to end
  - Put source address in DS:SI
  - Put destination address in ES:DI
  - Set CX to n, i.e. number of words to be moved
  - REP MOVSW - this instruction will run MOVSW n times.
    - After each iteration, it decrements CX by 1, increments both SI and DI by 2

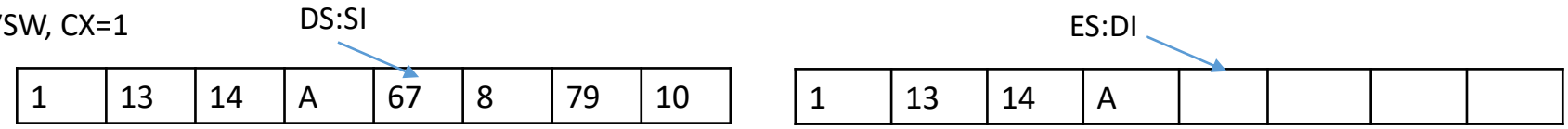
MOVSW and REP with DF=0


```
MOV cx, 3 ; number of words to be moved
REP MOVSW ; movsw will run 3 times now
```

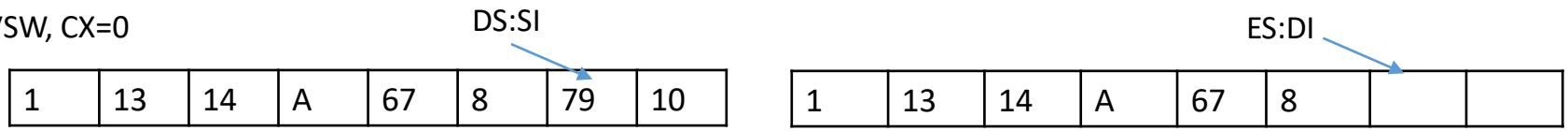
1 MOVSW, CX=2



2 MOVSW, CX=1



3 MOVSW, CX=0



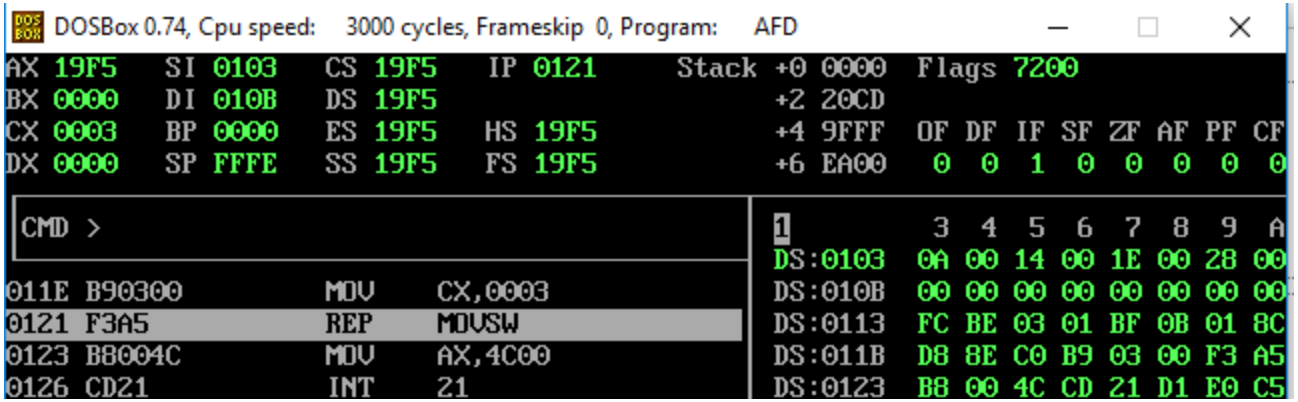
What will you change if you want to copy all 4 words?

Example 3: MOVSW with REP

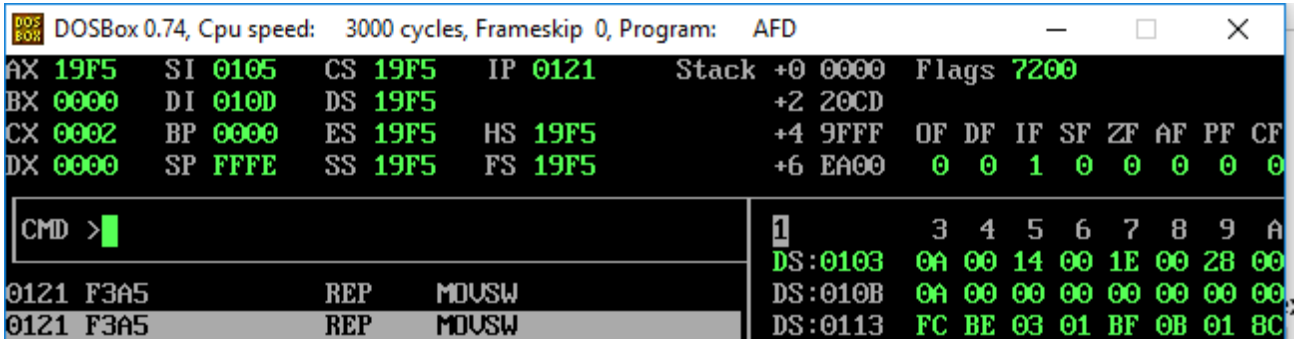
```
; copy and array of words  from Data segment to data segment
[org 0x0100]

jmp start

num1: dw 10, 20, 30, 40
num2: dw 0, 0, 0 ,0


start:
cld
mov si, num1
mov di, num2
mov ax, ds
mov es, ax ;
mov cx, 3 ; copy three words
rep movsw ; note that we have directly moved a number from memory to memory;
;you can use F1 to see each iteration of F2 to step over this instruction
; note the values of SI and DI at each repetition

mov ax, 0x4c00
int 21h
```

Debugging Example 3

Notice the values of DI, SI, CX and memory state before and after execution of each step



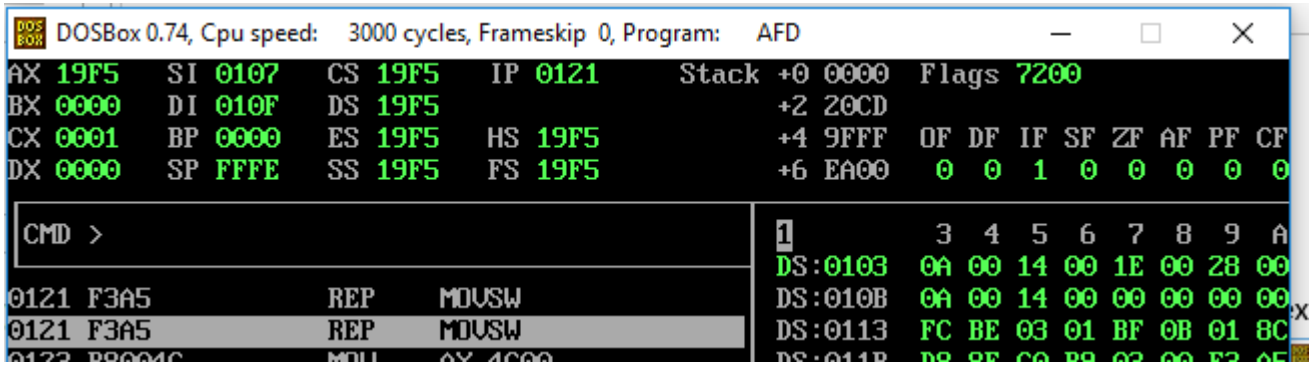
Initial state



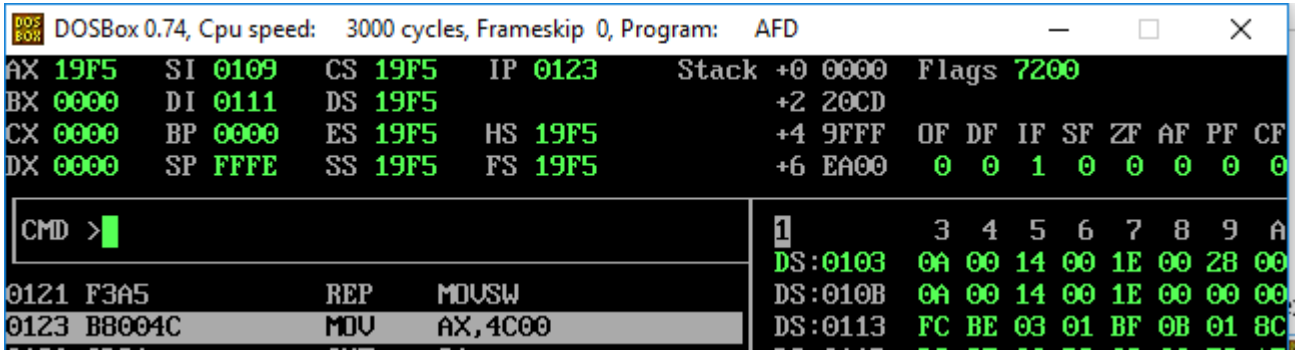
After one MOVSW

Debugging Example 3

Notice the values of DI, SI, CX and memory state before and after execution of each step



After 2nd MOVSW



After 3rd MOVSW

MOVSB with REP

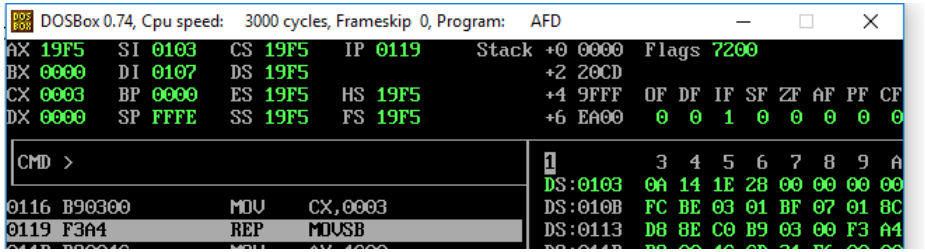
- MOVSB with REP works in same way as MOVSW with REP
- Only difference is SI, DI are incremented or decremented by 1
- CX will be number of bytes you want to copy

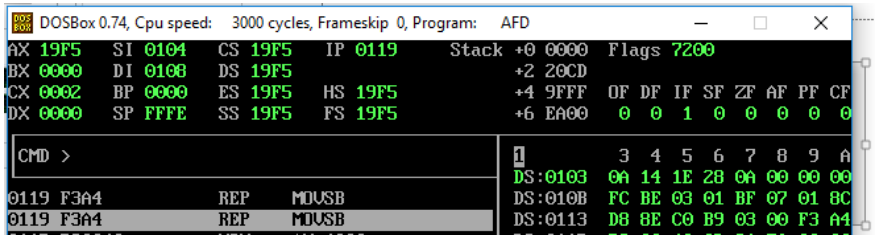
Example 4: MOVSB with REP

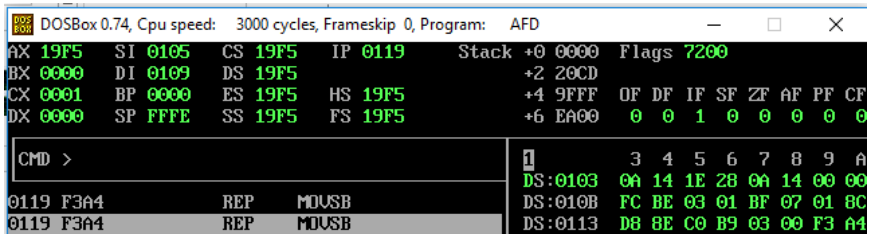
```
; copy and array of bytes  from Data segment to data segment
[org 0x0100]

jmp start

num1: db 10, 20, 30, 40
num2: db 0, 0, 0 ,0


start:
cld
mov si, num1
mov di, num2
mov ax, ds
mov es, ax ;
mov cx, 3 ; copy three bytes
rep movsb ; note that we have directly moved a number from memory to memory;
;you can use F1 to see each iteration of F2 to step over this instruction
; note the values of SI and DI at each repetition

mov ax, 0x4c00
int 21h
```

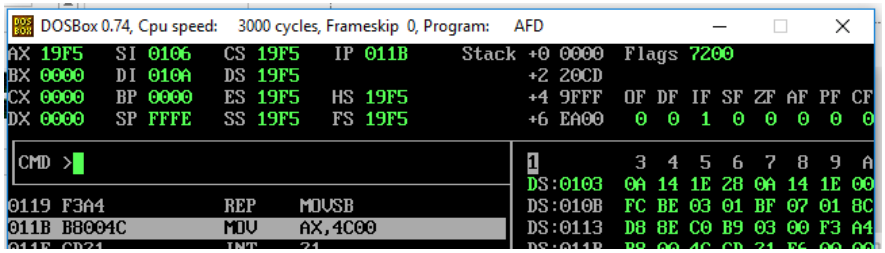
Debugging Example 4

Notice the values of DI, SI, CX and memory state before and after execution of each step









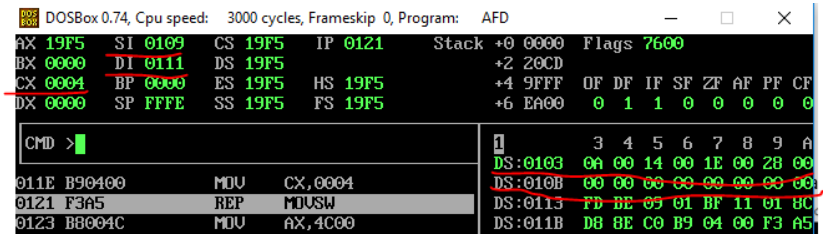
Order of moving data using MOVS

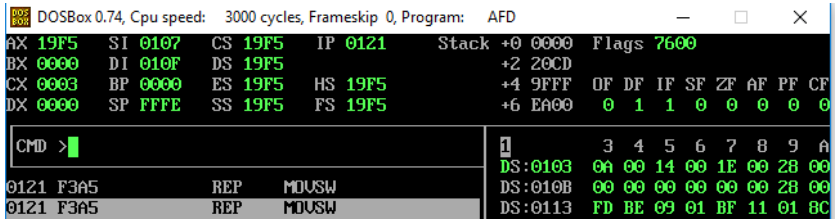
- To copy block of data from end to start set DF using STD instruction.

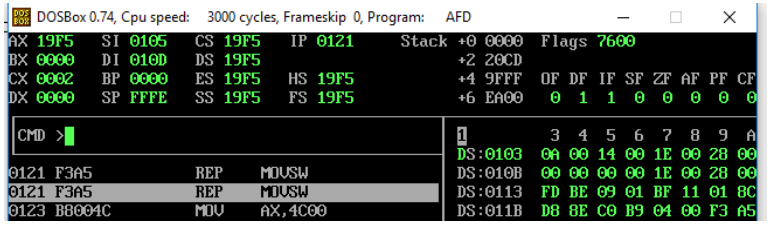
```
;Example 5: copy and array of word  from Data segment to data segment end to start
[org 0x0100]

jmp start

num1: dw 10, 20, 30, 40
num2: dw 0, 0, 0 ,0


start:
std
mov si, num1+6 ; end index of source array
mov di, num2+6; end index of destination array
mov ax, ds
mov es, ax ;
mov cx, 4 ; copy four words
rep movsw ; note that we have directly moved a number from memory to memory;
;you can use F1 to see each iteration of F2 to step over this instruction
; note the values of SI and DI at each repetition

mov ax, 0x4c00
int 21h
```

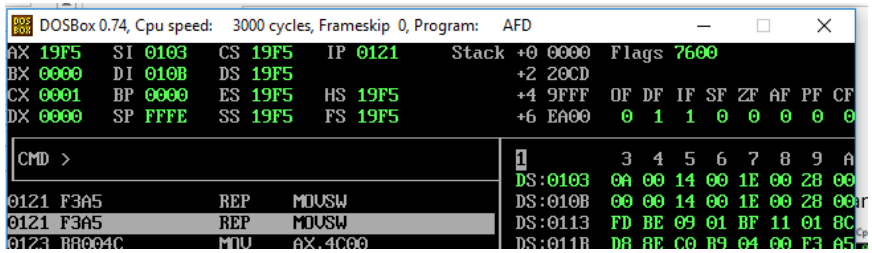
Debugging Example

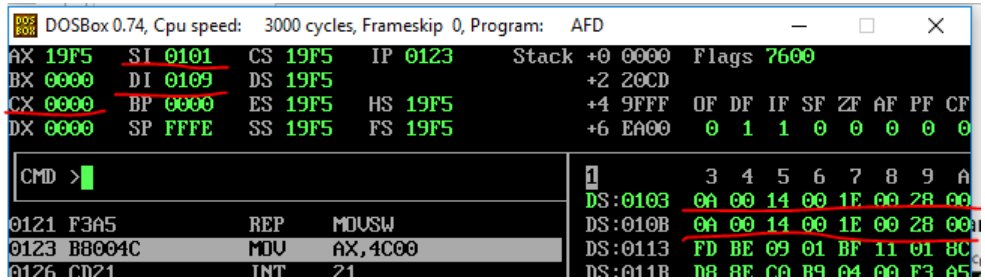
Notice the values of DI, SI, CX and memory state before and after execution of each step











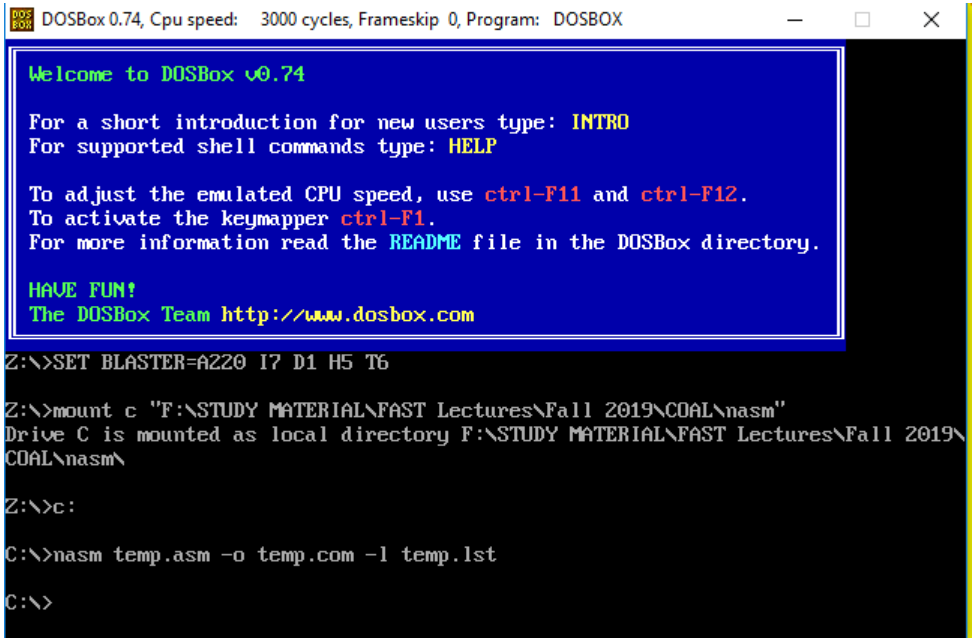
Why direction flag is needed?

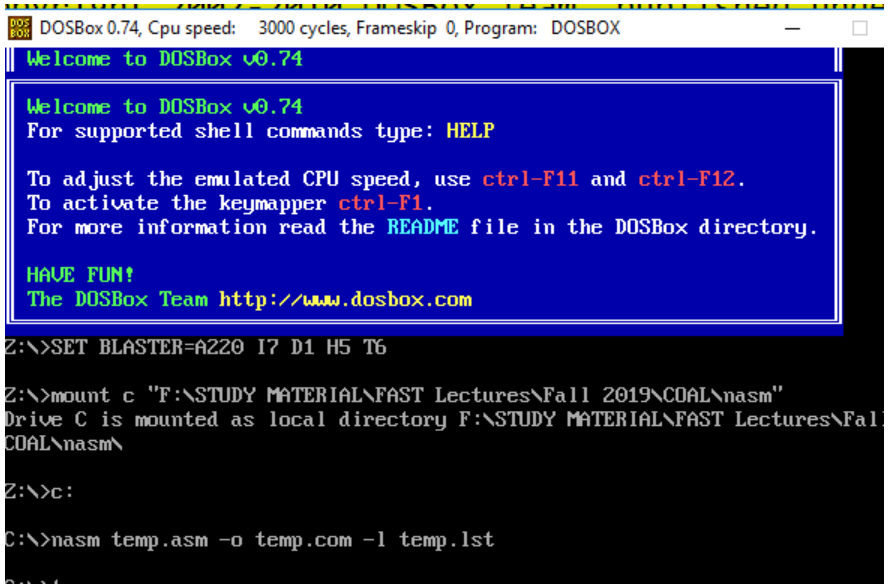
- The DF is important in the case of overlapping blocks.
- For example when the source and destination blocks overlap and the source is below the destination copy must be done upwards.
- While if the destination is below the source copy must be done downwards

MOVS Exercise

- Write a code the will take first line of screen (whatever is written their) and will copy the same to second line.

Hint: You can change DS register to point to B8000





STOS – store string

- STOS transfers a byte or word from register AL or AX to the string element addressed by ES:DI and updates DI to point to the next location.
- STOS is often used to clear a block of memory or fill it with a constant.
- If DF is zero, DI will be incremented by one or two depending of whether STOSB or STOSW is used.
- If REP is used before this instruction, the process will be repeated CX times.
- Note that values of AX/AL cannot be changed during one REP STOS.

STOS steps to use

- cld/std
- Set destination
- Set AX/AL as the value to be copied to destination
- Set CX to size of block
- REP STOSB (or STOSW)

Example: STOSW to clear screen

```
; clear screen using stosw
[org 0x0100]

cld
; set destination starting index
mov ax, 0xb800
mov es, ax
mov di, 0
; set element to be copied
mov ax, 0720h
mov cx, 80*25
rep stosw

mov ax, 0x4c00 ; terminate program
int 0x21
```

Exercise

- Change the code given in previous slide to color the screen as shown

Example: Copy -1 in whole array

```
[org 0x0100]
jmp start
num1: dw 10, 20, 40 ,50

start:

mov ax, ds
mov es, ax
mov di, num1

mov ax, -1
mov cx, 4
rep stosw

mov ax, 0x4c00 ; terminate program
int 0x21
```

LODS — Load string

- LODS transfers a byte or word from the source location DS:SI to AL or AX and updates SI to point to the next location.
- Again, SI is incremented/decremented depending on DF and by 1 or 2 depending on LODSB or LODSW
- LODS is generally used in a loop and not with the REP prefix
  - since the value previously loaded in the register is overwritten
  - if the instruction is repeated with REP, then only the last value of the block remains in the register

Example: Print string using LODS

```
;print string on screen using lods and stos
[org 0x0100]
jmp start

string: db 'hello world'
len: dw 11

start:
; in this example the string will be loaded element by element in al using lodsb
; ah will be 07h
; ax will be stored to display memory
cld
mov si, string ; set source for lod
mov di, 0 ; set destination for store
mov ax, 0xb800
mov es, ax ;
mov ah, 07h
mov cx, [len];
printChr:
lodsb
stosw
loop printChr ; the loop will execute will cx times

mov ax, 0x4c00 ; terminate program
int 0x21
```

- See this same example in book as subroutine
- Loop instruction will jump to label until cx becomes zero (cx-- in each iteration).

CMPS – Compare strings

- Compares blocks of data given at source and destination addresses
  - CMPW: Compare blocks word by word
  - CMPSB: Compare blocks byte by byte
- Used in pair with REPE or REPNE and conditional jump
  - REPE: repeats as long as blocks are same (can be used to check equality of blocks) or until CX > zero
  - REPNE: repeats as long as blocks are different (can be used to see check common elements in string) or until CX > zero
  - Flags are updated accordingly and conductional jumps can be used afterwards.
- DF has same use here as in MOVS, i.e. to control direction of comparison

CMPSW

- Steps to use
  - cld/ std
  - Set source and destination segments
  - Set source starting index
  - Set destination starting index
  - Set CX to length of block
  - Use REPE/REPNE CMPSW
  - Conditional Jump
- Note that SI and DI are auto incremented (or decremented) by 2 and CX is also decremented, as you saw in MOVSW

CMPSW Example

```
; compare array 1 with array2 if not equal set ax=1
[org 0x0100]
jmp start

array1: dw 10, 20, 30, 40
array2: dw 10, 20, 30, 40

start:
    cld
    mov si, array1
    mov di, array2
    mov ax, ds
    mov es, ax ;
    mov ax, 0
    mov cx, 4
    repe cmpsw;
    jne terminate

    mov ax, 1

terminate:
    mov ax, 0x4c00 ; terminate program
    int 0x21
```

Run this example with different values of arrays.
Note that SI and DI are auto incremented here and CX is zero after execution.

CMPSB

- Compares blocks byte by byte
- SI and DI are incremented / decremented by 1
- Rest is same as CMPW
- See example 7.7 from BH for string comparison

Exercise

Find a substring in a string and place the starting point of substring in AX. If substring is not found place -1 in ax.

Sample runs

- string=ABCABD, substring=ABD then ax should be 3
- string=ABCABX, substring=ABD then ax should be -1
- string=ABCABD, substring=ABC then ax should be 0

SCAS – Scan string

- Compares value in accumulator to all elements in a block starting at ES:DI
  - SCASW: Compares AX with all words in block one by one (DI +/- 2)
  - SCASB: Compares AL with all bytes in block one by one (DI +/- 1)
- REPE
  - Continue scanning the array/string while CX > 0 and the value in AL/AX matches each subsequent value in memory. Stop on finding a mismatch, or earlier if CX becomes 0
- REPNE
  - Scan until either AL/AX matches a value in memory. Stop when a match is found, or earlier if CX becomes 0
- Flag are updated accordingly. Use conductional jump afterwards

SCASW

- Steps to use:
  - cld/ std
  - Set destination segments
  - Set destination starting index
  - Set AX to element to be found
  - Set CX to length of block
  - REPE/REPNE SCASW
  - Conditional Jump
- DI is auto incremented (or decremented) by 2 and CX is also decremented in each iteration.

SCASW Example

```
; scan to match a charater in stringl mov 1 to bx if found
[org 0x0100]
jmp start

string: db 'ABCDEFGH'

start:
    cld
    mov di, string ; string to be compared
    mov ax, ds
    mov es, ax
    mov aL, 'F' ; element to be found
    mov bx, 0
    mov cx, 8
    repne scasb;  repeat till not found or CX!=0
    jne terminate; quit id letter not found

    mov bx, 1

terminate:
    mov ax, 0x4c00 ; terminate program
    int 0x21
```

- Try this example with different values of string and AL
- Note that after execution, DI points to one element **beyond** where ‘F’ was found.

SCASW Exercise

- Write a code that will scan an array of word-sized numbers, looking for a specific number, say 25. The code should place a 0 in BX if number is not found and 1 if found
- Read and run Example 7.3 from BH to use SCAS to find the length of string ending with null for example:
  - message: db 'hello world', 0
- Write a code the will find the frequency of ‘B’ in string using SCAS.
  - For example string: db ‘AAABCDBB’
  - At the end of your code BX should be 2

SCAS Example 2

```
; find how many F are at start of string, place that value in bx
[org 0x0100]
jmp start

string: db 'FFFAAAA'

start:
    cld
    mov di, string ; string to be compared
    mov ax, ds
    mov es, ax
    mov aL, 'F' ; element to be found
    mov bx, 0
    mov cx, 8
    repe scasb;  repeat till not found or CX!=0

    mov bx, cx; cx will be 4 at this point as it has scanned 4 element out of which 1 was !="F"
    dec bx

terminate:
     mov ax, 0x4c00 ; terminate program
     int 0x21
```

LES and LDS instructions

- In all of string instructions source and destination are segment offset pair.
- LES and LDS make it easier to copy values to ES:DI and DS:SI in one statement by loading segment and general purpose registers from consecutive memory locations in one instruction.
- Syntax
- LES register, mem location
  - Will load value on mem location to reg and mem+2 value to ES
- LDS register, mem location
  - Will load value on mem location to reg and mem+2 value to DS

LES and LDS instructions

- These instructions are mainly helpful when a subroutine receives source and destination string addresses as parameters on stack.
- e.g. If caller pushed value of ES and then of DI
- Within subroutine, value of DI is at BP+4 and value of ES is at BP+6. Following instruction will load these two correctly
  - LES di, [bp+4] ; [bp+4] to DI, [bp+6] to ES
- Similarly, “LDS si, [bp+4]” will load SI from BP+4 and DS from BP+6.