Manual for
Computer Organization and Assembly Language

Department of Computer Science
National University of Computer and Engineering Sciences
Chiniot-Faisalabad Campus

Outline

- String Primitive Instructions
  - MOVSB, MOVSW, and MOVSD
  - CMPSB, CMPSW, and CMPSD
  - SCASB, SCASW, and SCASD
  - STOSB, STOSW, and STOSD
  - LODSB, LODSW, and LODSD

1 String Primitive Instructions

The x86 instruction set has five groups of instructions for processing arrays of bytes, words, and doublewords. Although they are called string primitives, they are not limited to character arrays. Each instruction in Table 1 implicitly uses ESI, EDI, or both registers to address memory. References to the accumulator imply the use of AL, AX, or EAX, depending on the instruction data size. String primitives execute efficiently because they automatically repeat and increment array indexes.

Table 1: String Primitive Instructions.

Instruction	Description
MOVSB, MOVSW, MOVSD	**Move string data:** Copy data from memory addressed by ESI to memory addressed by EDI.
CMPSB, CMPSW, CMPSD	**Compare strings:** Compare the contents of two memory locations addressed by ESI and EDI.
SCASB, SCASW, SCASD	**Scan string:** Compare the accumulator (AL, AX, or EAX) to the contents of memory addressed by EDI.
STOSB, STOSW, STOSD	**Store string data:** Store the accumulator contents into memory addressed by EDI.
LODSB, LODSW, LODSD	**Load accumulator from string:** Load memory addressed by ESI into the accumulator.

Using a Repeat Prefix

By itself, a string primitive instruction processes only a single memory value or pair of values. If you add a repeat prefix, the instruction repeats, using ECX as a counter. The repeat prefix permits you to process an entire array using a single instruction.

Table 2: Repeat Prefixes.

Prefix	Description
REP REPZ, REPE REPNZ, REPNE	Repeat while ECX > 0 Repeat while the Zero flag is set and ECX > 0 Repeat while the Zero flag is clear and ECX > 0

Direction Flag

String primitive instructions increment or decrement ESI and EDI based on the state of the Direction flag (see Table 3). The Direction flag can be explicitly modified using the CLD and STD instructions:

```
CLD;clearDirectionflag(forwarddirection)
STD;setDirectionflag(reversedirection)
```

Table 3: Direction Flag Usage in String Primitive Instructions.

Values of the Direction Flag	Effect on ESI and EDI	Address Sequence
clear	Incremented	Low-high
Set	Decremented	High-low

The size of the increment/decrement is shown in the following table:

Table 4: Size Increment/Decrement.

Instruction	Value Added or Subtracted from ESI and EDI
SB instruction	1
SW instruction	2
SD instruction	4

1.1 MOVSB, MOVSW, and MOVSD

The MOVSB, MOVSW, and MOVSD instructions copy data from the memory location pointed to by ESI to the memory location pointed to by EDI. The two registers are either incremented or decremented automatically (based on the value of the Directionflag):

Suppose we want to copy 20 doubleword integers from source to target. After the array is copied, ESI and EDI point one position (4 bytes) beyond the end of each array:

```
.data
        sourceDWORD20DUP(0FFFFFFFFh)
        targetDWORD20DUP(?)
.code
        cld;direction=forward
        movecx,LENGTHOFsource;setREPcounter
        movesi,OFFSETsource;ESIpointstosource
        movedi,OFFSETtarget;EDIpointstotarget
        repmovsd;copydoublewords
```

1.2 CMPSB, CMPSW, and CMPSD

The CMPSB, CMPSW, and CMPSD instructions each compare a memory operand pointed to by ESI to a memory operand pointed to by EDI:

```
.data
        sourceDWORD1234
        h
        targetDWORD5678
        movesi,OFFSETsource
        movedi,OFFSETtarget
        cmpsd;comparedoublewords
        jaL1;jumpifsource>target
```

To compare multiple doublewords, clear the Direction flag (forward direction), initialize ECX as a counter, and use a repeat prefix with CMPSD:

```
 movesi,OFFSETsource
 movedi,OFFSETtarget
cld;direction=forward
movecx,LENGTHOFsource;repetitioncounter
repecmpsd;repeatwhileequal
```

1.3 SCASB, SCASW, and SCASD

The SCASB, SCASW, and SCASD instructions compare a value in AL/AX/EAX to a byte, word, or doubleword, respectively, addressed by EDI. The instructions are useful when looking for a single value in a string or array. Combined with the REPE (or REPZ) prefix, the string or array is scanned while ECX > 0 and the value in AL/AX/EAX matches each subsequent value in memory. The REPNE prefix scans until either AL/AX/EAX matches a value in memory or ECX = 0.

In the following example we search the string alpha, looking for the letter F. If the letter is found, EDI points one position beyond the matching character. If the letter is not found, JNZ exits:

```
.data
    alphaBYTE"ABCDEFGH",0
.code
    movedi,OFFSETalpha;EDIpointstothestring
    moval,'F';searchfortheletterF
    movecx,LENGTHOFalpha;setthesearchcount
    cld;direction=forward
    repnescasb;repeatwhilenotequal
    jnzquit;quitifletternotfound
    decedi;found:backupEDI
```

JNZ was added after the loop to test for the possibility that the loop stopped because ECX = 0 and the character in AL was not found.

1.4 STOSB, STOSW, and STOSD

The STOSB, STOSW, and STOSD instructions store the contents of AL/AX/EAX, respectively, in memory at the offset pointed to by EDI. EDI is incremented or decremented based on the state of the Direction flag. When used with the REP prefix, these instructions are useful for filling all elements of a string or array with a single value. For example, the following code initializes each byte in string1 to 0FFh:

```
.data
    Count= 100
    string1BYTECountDUP(?)
.code
    moval,0FFh;valuetobestored
    movedi,OFFSETstring1;EDIpointstotarget
    movecx,Count;charactercount
    cld;direction=forward
    repstosb;fillwithcontentsofAL
```

1.5 LODSB, LODSW, and LODSD

The LODSB, LODSW, and LODSD instructions load a byte or word from memory at ESI into AL/AX/EAX, respectively. ESI is incremented or decremented based on the state of the Direction flag. The REP prefix is rarely used with LODS because each new value loaded into the accumulator overwrites its previous contents. Instead, LODS is used to load a single value.