ASCII and BCD Arithmetic

Chapter 11
S. Dandamudi
Outline

• Representation of Numbers
  * ASCII representation
  * BCD representation
    » Unpacked BCD
    » Packed BCD
• Processing ASCII numbers
  » ASCII addition
  » ASCII subtraction
  » ASCII multiplication
  » ASCII division
  * Example: Multidigit ASCII addition
• Processing packed BCD numbers
  * Packed BCD addition
  * Packed BCD subtraction
  * Example: Multibyte packed BCD addition
• Performance: Decimal versus binary arithmetic
Representation of Numbers

- Numbers are in ASCII form
  * when received from keyboard
  * when sending to the display
- Binary form is efficient to process numbers internally
Representation of Numbers (cont’d)

• Requires conversion between these two number representations
  » We have used GetInt/GetLint and PutInt/PutLint to perform these two conversions

• In some applications, processing of numbers is simple (e.g. a simple addition)
  » Does not justify the input and output conversion overheads
  » In this case, it is better to process numbers in the decimal form

• Decimal numbers can be represented in
  » ASCII
  » BCD
Representation of Numbers (cont’d)

• ASCII representation
  * Numbers are stored as a string of ASCII characters
    » Example: 1234 is stored as 31 32 33 34H
      ↗ ASCII for 1 is 31H, for 2 is 32H, etc.

• BCD representation
  * Unpacked BCD
    » Example: 1234 is stored as 01 02 03 04H
      – Additional byte is used for sign
        ↗ Sign byte: 00H for + and 80H for –
  * Packed BCD
    » Saves space by packing two digits into a byte
      – Example: 1234 is stored as 12 34H
Processing ASCII Numbers

• Pentium provides four instructions
  aaa  – ASCII adjust after addition
  aas  – ASCII adjust after subtraction
  aam  – ASCII adjust after multiplication
  aad  – ASCII adjust before division

  * These instructions do not take any operands
    » Operand is assumed to be in AL
Processing ASCII Numbers (cont’d)

ASCII addition

Example 1

34H = 00110100B
35H = 00110101B
69H = 01101001B

Should be 09H
Ignore 6

Example 2

36H = 00110110B
37H = 00110111B
6DH = 01101101B

Should be 13H
Ignore 6 and add 9 to D

• The aaa instruction performs these adjustments to the byte in AL register
Processing ASCII Numbers (cont’d)

• The **aaa** instruction works as follows:
  * If the least significant four bits in AL are > 9 or if AF =1, it adds 6 to AL and 1 to AH.
    – Both CF and AF are set
  * In all cases, the most significant four bits in AL are cleared
  * Example:

    ```assembly
    sub    AH, AH    ; clear AH
    mov    AL, '6'   ; AL := 36H
    add    AL, '7'   ; AL := 36H+37H = 6DH
    aaa    ; AX := 0103H
    or     AL, 30H   ; AL := 33H
    ```
Processing ASCII Numbers (cont’d)

ASCII subtraction

- The `aas` instruction works as follows:
  * If the least significant four bits in AL are > 9 or if AF = 1, it subtracts 6 from AL and 1 from AH.
    - Both CF and AF are set
  * In all cases, the most significant four bits in AL are cleared

- This adjustment is needed only if the result is negative
Processing ASCII Numbers (cont’d)

• Example 1: Positive result

sub    AH,AH    ; clear AH
mov    AL,'9'   ; AL := 39H
sub    AL,'3'   ; AL := 39H–33H = 6H
aas             ; AX := 0006H
or     AL,30H   ; AL := 36H

• Example 2: Negative result

sub    AH,AH    ; clear AH
mov    AL,'3'   ; AL := 33H
sub    AL,'9'   ; AL := 33H–39H = FAH
aas             ; AX := FF04H
or     AL,30H   ; AL := 34H
Processing ASCII Numbers (cont’d)

ASCII multiplication

• The `aam` instruction adjusts the result of a `mul` instruction
  * Multiplication should not be performed on ASCII
    » Can be done on unpacked BCD

• The `aam` instruction works as follows
  * AL is divided by 10
  * Quotient is stored in AH
  * Remainder in AL

• `aam` does not work with `imul` instruction
Processing ASCII Numbers (cont’d)

• Example 1

  mov AL, 3     ; multiplier in unpacked BCD form
  mov BL, 9     ; multiplicand in unpacked BCD form
  mul BL       ; result 001BH is in AX
  aam           ; AX := 0207H
  or AX, 3030H  ; AX := 3237H

• Example 2

  mov AL, '3'   ; multiplier in ASCII
  mov BL, '9'   ; multiplicand in ASCII
  and AL, 0FH   ; multiplier in unpacked BCD form
  and BL, 0FH   ; multiplicand in unpacked BCD form
  mul BL       ; result 001BH is in AX
  aam           ; AX := 0207H
  or AL, 30H    ; AL := 37H
Processing ASCII Numbers (cont’d)

ASCII division

• The `aad` instruction adjusts the numerator in AX before dividing two unpacked decimal numbers
  *
  The denominator is a single unpacked byte

• The `aad` instruction works as follows
  *
  Multiplies AH by 10 and adds it to AL and sets AH to 0
  *
  Example:
  » If AX is 0207H before `aad`
  » AX is changed to 001BH after `aad`

• `aad` instruction reverses the changes done by `aam`
Processing ASCII Numbers (cont’d)

• Example: Divide 27 by 5

```assembly
mov AX, 0207H ; dividend in unpacked BCD form
mov BL, 05H   ; divisor in unpacked BCD form
aad           ; AX := 001BH
div BL        ; AX := 0205H
```

• `aad` converts the unpacked BCD number in AX to binary form so that `div` can be used

Example: Multidigit ASCII addition

* ASCIIADD.ASM

* Adds two 10-digit numbers

  » Adds one digit at a time starting with the rightmost digit
Processing Packed BCD Numbers

- Two instructions to process packed BCD numbers
  - **daa** – Decimal adjust after addition
    - Used after **add** or **adc** instruction
  - **das** – Decimal adjust after subtraction
    - Used after **sub** or **sbb** instruction

* No support for multiplication or division
  - For these operations
    - Unpack the numbers
    - Perform the operation
    - Repack them
Processing Packed BCD Numbers (cont’d)

Packed BCD addition

Example 1

\[
\begin{align*}
29H &= 00101001B \\
69H &= 01101001B \\
92H &= 10010010B \\
\hline
\text{Should be 98H (add 6)}
\end{align*}
\]

Example 2

\[
\begin{align*}
27H &= 00100111B \\
34H &= 00110100B \\
5BH &= 01011101B \\
\hline
\text{Should be 61H (add 6)}
\end{align*}
\]

Example 3

\[
\begin{align*}
52H &= 01010010B \\
61H &= 01100001B \\
B3H &= 10110010B \\
\hline
\text{Should be 13H (add 60H)}
\end{align*}
\]
Processing Packed BCD Numbers (cont’d)

• The **daa** instruction works as follows:

  * If the least significant four bits in AL are > 9 or
    if AF = 1, it adds 6 to AL and sets AF
  * If the most significant four bits in AL are > 9 or
    if CF = 1, it adds 60H to AL and sets CF

**Example:**

```
mov    AL, 71H
add    AL, 43H    ; AL := B4H
daar                  ; AL := 14H and CF := 1
```

  * The result including the carry (i.e., 114H) is the correct answer
Processing Packed BCD Numbers (cont’d)

Packed BCD subtraction

• The **das** instruction works as follows:
  * If the least significant four bits in AL are > 9 or if AF = 1, it subtracts 6 from AL and sets AF
  * If the most significant four bits in AL are > 9 or if CF = 1, it subtracts 60H from AL and sets CF

Example:

```
  mov    AL, 71H
  sub    AL, 43H ; AL := 2EH
  das    ; AL := 28H
```
Processing Packed BCD Numbers (cont’d)

Example: Multibyte packed BCD addition

• Adds two 10-digit numbers
  » Adds two digits at a time starting from the rightmost pair

• For storage of the two input numbers and the result, we can use DT (Define Ten-byte) directive
  * DT stores in packed BCD form
  * Example:
    
    \[
    \text{DT 1234567890}
    \]

    is stored as
    
    \[
    90 78 56 34 12H
    \]
Performance: Decimal vs Binary Arithmetic

- Tradeoffs associated with the three representations

<table>
<thead>
<tr>
<th>Representation</th>
<th>Storage overhead</th>
<th>Conversion overhead</th>
<th>Processing overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>Nil</td>
<td>High</td>
<td>Nil</td>
</tr>
<tr>
<td>Packed BCD</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>ASCII</td>
<td>High</td>
<td>Nil</td>
<td>High</td>
</tr>
</tbody>
</table>
Performance: Decimal vs Binary Arithmetic

![Graph showing the comparison between decimal and binary arithmetic execution times.](image)

- **Execution time (seconds)**
- **Number of calls (in thousands)**
  - ASCII add
  - Binary add
  - BCD add
  - ASCII add

1998 © S. Dandamudi

To be used with S. Dandamudi, “Introduction to Assembly Language Programming,” Springer-Verlag, 1998.
Performance: Decimal vs Binary Arithmetic

![Graph showing execution time vs number of calls for ASCII add, BCD add, and Binary add.](image)

- **Execution time (seconds)** vs **Number of calls (in thousands)**
  - ASCII add
  - BCD add
  - Binary add

To be used with S. Dandamudi, “Introduction to Assembly Language Programming,” Springer-Verlag, 1998.