BCD (ASCII) Arithmetic

- We will first look at unpacked BCD which means strings that look like '4567'.
- Bytes then look like: 34h 35h 36h 37h OR: 04h 05h 06h 07h
- x86 processors also have instructions for packed BCD where there are two decimal digits per byte
  99h = 99D
  3456 = 3456h
- With unpacked BCD we wish to add strings such as '989' and '486' and come up with the string '1475'.
- With BCD you can use the standard input and output routines for strings to get numbers into and out of memory without converting to binary
- BCD arithmetic uses the standard binary arithmetic operators and then converts the result to BCD using the various adjust operators.

Where and Why is BCD used?

- BCD obviously takes more space and time than standard binary arithmetic
- It is used extensively in applications that deal with currency because floating point representations are inherently inexact
- Database management systems offer a variety of numeric storage options; "Decimal" means that numbers are stored internally either as BCD or as fixed-point integers
- BCD offers a relatively easy way to get around size limitations on integer arithmetic

BCD Adjustment Instructions

- Four unpacked adjustment instructions are available:
  AAA (ASCII Adjust After Addition)
  AAS (ASCII Adjust After Subtraction)
  AAM (ASCII Adjust After Multiplication)
  AAD (ASCII Adjust Before Division)

- Except for AAD the instructions are used to adjust results after performing a binary operation on ASCII or BCD data
- AAD (in spite of the mnemonic) is used after a DIV instruction

Packed BCD, ASCII, Unpacked BCD

- AAA and AAS can be used with both ASCII and unpacked BCD
  9701 in ASCII (hex) 39 39 30 31
  9701 in unpacked BCD 09 07 00 01

- Two packed BCD operations are also available:
  DAA Decimal Adjust After Addition
  DAS Decimal Adjust After Subtraction

AAA

- This instruction has very complex semantics that are rarely documented correctly. Here are two examples:

```
IF AL > 9 OR AF = 1 THEN
    AL <- AL + 10
    AH <- AH + 1
    CF <- 1     AF <- 1
ELSE
    CF <- 0
    AF <- 0
END

IF AL > 9 OR AF = 1 THEN
    AL <- AL + 6
    AH <- AH + 1
    Bits 4-7 of AL set to 0
    AF and CF set
ELSE
    AF and CF clear
    Bits 4-7 of AL set to 0
END
```

Example

- Let’s see what happens if we try to add ASCII strings byte by byte:
  989 -> 39 38 39
  486 -> 34 38 36
  1475 6D 70 6F
  As you can see the result in binary does not look like what we want.
  When adding, AF is set whenever there is a carry from the lowest-order nibble to the next lowest nibble.
  Recall that a NIBBLE is one hex digit or 4 bits.
  When adding 9 and 6 in hex we get F and no binary carry.
  Note that AF is clear after the addition of 9h and 6h, because there was no carry from the low-order nibble to the next one
  If adding 9 and 7 we would get 10h, and AF would be set.
AAA Semantics

• AAA (ASCII Adjust for Addition) does the following things:

  IF AL > 9 THEN
  clear top nibble of AL
  AL = AL - 10
  AH = AH + 1
  CF = 1
  AF = 1

  ELSE
  CF = 0
  AF = 0
  clear top nibble of AL

  END

• Notes
  1. Addition result must be in AL in order for AAA to work
  2. Top nibble of AL always cleared so AAA will adjust for ASCII as well as unpacked BCD
  3. Either AH or CF can be used for decimal carries.

AAA Example with no Aux Carry

<table>
<thead>
<tr>
<th>AX=0000</th>
<th>BX=0000</th>
<th>CX=0012</th>
<th>DX=0000</th>
<th>SP=FFFE</th>
<th>BP=0000</th>
<th>SI=0000</th>
<th>DI=0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
<td>CS=2BC5</td>
<td>IP=0108</td>
<td>NV</td>
<td>UP</td>
<td>EI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B039</td>
<td>MOV</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0039</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AA</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0071</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0013</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
</tbody>
</table>

AAA Example with Aux Carry

<table>
<thead>
<tr>
<th>AX=0000</th>
<th>BX=0000</th>
<th>CX=0012</th>
<th>DX=0000</th>
<th>SP=FFFE</th>
<th>BP=0000</th>
<th>SI=0000</th>
<th>DI=0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
<td>CS=2BC5</td>
<td>IP=0108</td>
<td>NV</td>
<td>UP</td>
<td>EI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B039</td>
<td>MOV</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0039</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AA</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0071</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0013</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
</tbody>
</table>

Example with Unpacked BCD

<table>
<thead>
<tr>
<th>AX=0000</th>
<th>BX=0000</th>
<th>CX=0012</th>
<th>DX=0000</th>
<th>SP=FFFE</th>
<th>BP=0000</th>
<th>SI=0000</th>
<th>DI=0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
<td>CS=2BC5</td>
<td>IP=0108</td>
<td>NV</td>
<td>UP</td>
<td>EI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B009</td>
<td>MOV</td>
<td>AL,31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0009</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AA</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0009</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0009</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
</tbody>
</table>

Example with No Carries

<table>
<thead>
<tr>
<th>AX=0000</th>
<th>BX=0000</th>
<th>CX=0012</th>
<th>DX=0000</th>
<th>SP=FFFE</th>
<th>BP=0000</th>
<th>SI=0000</th>
<th>DI=0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
<td>CS=2BC5</td>
<td>IP=0108</td>
<td>NV</td>
<td>UP</td>
<td>EI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B031</td>
<td>MOV</td>
<td>AL,31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0031</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AA</td>
<td>AL,39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0031</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX=0031</td>
<td>BX=0000</td>
<td>CX=0012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DS=2BC5</td>
<td>ES=2BC5</td>
<td>SS=2BC5</td>
</tr>
</tbody>
</table>

BCD Addition Program

strl db '04989' ; leading 0 allows easy processing
str2 db '07486' ; in loop without concern for sum db 5 dup ? ; extra digit for carry out

... mov si, offset str1+4 ; point to LSD
move bx, offset str2+4
mov di, offset sum+4
mov cx, 5 ; digits to process
cic ; ensure of clear
BCD Addition Program (2)

LP1:
  mov al,[si]           ;get a digit from op1
  adc al,[bx]           ;add prev cf + op2 digit
  aaa                   ;adjust
  mov [di], al         ;save result
  dec bx                ;advance all 3 pointers
  dec si
  dec di
  loop LP1

  mov cx, 5             ;convert to ASCII
  inc di                ;adjust di back to MSD

LP2:
  or byte ptr[di], 30h   ;convert
  inc di
  loop LP2

What’s Wrong with This?
;this code tries to do it in a single loop
;but doesn’t work correctly
LP1:
  mov al,[si]           ;get a digit from op1
  adc al,[bx]           ;add prev cf + op2 digit
  aaa                   ;adjust
  or al, 30h            ;convert to ASCII
  mov [di], al         ;save result
  dec bx                ;advance all 3 pointers
  dec si
  dec di
  loop LP1

;How can we fix without using a 2nd loop?

AAS
• AAS (ASCII Adjust for Subtraction) works in a similar manner to AAA
• Note that negative results are expressed in 10’s complement.

10’s Complement
• The reason that 2’s complement is used in computers is that we can replace subtraction with addition
• We can apply the same principle to decimal arithmetic
• To obtain the 10’s complement of a number take the 9’s complement of each digit and then add 1
• Example:
  68 – 37 = ?
  So 99 – 37 = 62
  And 62 + 1 = 63
  Subtracting 68 from 37 is equivalent to 63 + 68 with the carry discarded, 63 + 68 = 131 = 31

10’s Complement (2)
• Complement notation is used with ‘fixed size’ integers.
• For a given size n digits, you can compute the 10’s complement by subtracting from n 9’s and then add 1
• Or you can subtract the number from 10^n-1
• Example: what is –77 in 5 digit 10’s complement?
  99999 – 00077 = 99922 + 1 = 99923
  Or 100000 – 77 = 99923
• Note that the leftmost digit is a “sign digit.” If it is >= 5 the result is negative

Other Adjustment Instructions
• There are four other instructions used in BCD arithmetic:
  • Unpacked BCD:
    AAM    ASCII Adjust After Multiplication
    AAD    ASCII Adjust before Division
  • Packed BCD:
    DAA    Decimal adjust after addition
    DAS    Decimal adjust after subtraction
• Note that there are no multiplication or division instructions for packed BCD.
• The BCD instructions can also be used for certain specialized conversions.
• We will take a brief look at AAM and AAD
AAM (ASCII Adjust after Multiplication)

- Used after multiplication of two unpacked BCD numbers (note: NOT ASCII).
- Semantics:
  \[ \text{AL} < \text{AL} \mod 10, \text{AH} < \text{AH}/10, \text{ZF} < \text{SF} \text{ or ZF} \]
  \[ \text{ZF} \text{ if AL} = 0 \]

- Examples:
  \[ I=0000 \]
  \[ I=0000 \]

- In the original 8086, there was not enough room in the
  Values other than 10 can be used (if the assembler will do
  \[ I=0000 \]
  \[ AH \times 10 + AL \]
  \[ I=0000 \]
  \[ \text{al}, \text{AL}/10 \]
  \[ ; \text{DOS get time function} \]
  \[ ; \text{get tens ASCII digit} \]

- Because many programs came to rely on this behavior,
  Intel retained it but never documented it

- Both are particularly useful with the value 16

- Undocumented Operations with AAM and AAD
  - In the original 8086, there was not enough room in the
    microcode for the constant 10d intended to be used for
    AAM and AAD.
  - Consequently the 10d is actually placed as an immediate
    value in the machine code, even though the instruction
    was documented only as a 0 operand instruction
  - AAM and AAD are assembled as 2-byte, one operand
    instructions AAD imm and AAM imm
  - Values other than 10 can be used (if the assembler will do
    it or if you are willing to patch the assembled code
    manually)
  - Both are particularly useful with the value 16
  - Because many programs came to rely on this behavior,
    Intel retained it but never documented it

2-Digit Decimal Conversions

- The zero operand AAD and AAM with implied base 10
  can be used for conversion of 2-digit decimal numbers
- Examples:
  \[ ; \text{binary to ASCII} \]
  \[ \text{mov ah}, 2ch \quad ; \text{DOS get time function} \]
  \[ \text{int 21h} \]
  \[ \text{mov al}, \text{cl} \quad ; \text{load minutes into AL} \]
  \[ \text{aam} \quad ; \text{al} < \text{al} \mod 10, \text{ah} < \text{al}/10 \]
  \[ \text{or ax, 3030h} \quad ; \text{convert to ascii} \]
Generalized BCD Addition

BCDAdd:
; si points to last digit of operand 1
; di points to last digit of operand 2
; bx points to last digit of storage for result
; cx contains digit count (same for both operands!)
clc ; ensure CF clear
LP1:
mov al,[si] ; get digit from op1
adc al,[di] ; add in corresponding digit from op2
aaa ; adjust
pushf ; save flags
or al,30H ; convert to ascii
popf ; restore CF
mov [bx],al ; and store
dec bx ; adjust pointers (CF unchanged!)
dec si
dec di
loop LP1
ret