ASCII and unpacked decimal arithmetic

This is a type of arithmetic that can be performed on ASCII decimal strings, without requiring them to be converted to binary.

There are two advantages to using ASCII arithmetic:

Conversion from string format before performing arithmetic is not necessary.

Using an assumed decimal point permits operations on real numbers without danger of the roundoff errors that occur with floating-point numbers.

However, ASCII arithmetic does execute more slowly than binary arithmetic.

There are four instructions that deal with ASCII addition, subtraction, multiplication, and division:

- **AAA (ASCII adjust after addition)**
- **AAS (ASCII adjust after subtraction)**
- **AAM (ASCII adjust after multiplication)**
- **AAD (ASCII adjust before division)**

These instructions are used to adjust the sum, difference, product, or quotient, respectively, to ensure that it is in a valid ASCII decimal format.

Here is an example of how to use ASCII addition to add the numbers 3402 and 1256:

```
; Load the first operand into the AL register.
mov al, 3
; Load the second operand into the AH register.
mov ah, 4
; Add the two operands together.
adc al, 0
; ASCII adjust the sum.
aaa
; Store the sum in the AX register.
mov ax, al
```

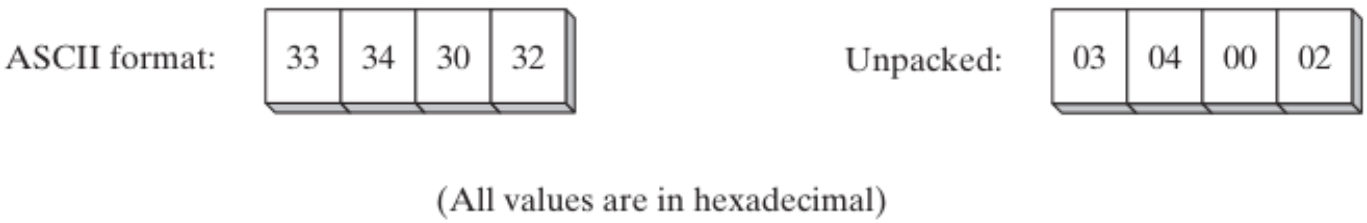
This code will add the two numbers together and store the sum in the AX register.

The AAA instruction is used to adjust the sum to ensure that it is in a valid ASCII decimal format.

ASCII subtraction can be performed in a similar way, using the AAS instruction to adjust the difference.

ASCII multiplication and division are also possible, but they are more complex and require the use of the AAM and AAD instructions, respectively.

ASCII arithmetic can be a useful tool for performing arithmetic on ASCII decimal strings. It is important to note, however, that ASCII arithmetic is slower than binary arithmetic.



(All values are in hexadecimal)

This means that the numbers in the block diagram are represented in two different formats: ASCII and unpacked decimal.

ASCII is a character encoding standard that assigns a unique code to

each letter, number, and symbol. The ASCII codes for the digits 0 through 9 are 30 through 39, respectively.

Unpacked decimal is a binary representation of decimal numbers, where one byte is used to represent each digit. The unpacked decimal representation of the number 12 is 000000010010, or 0x0302 in hexadecimal.

The image shows that the four numbers in the block diagram are the same in both ASCII and unpacked decimal formats. This is because the ASCII codes for the digits 0 through 9 are the same as the unpacked decimal representations of those digits.

Digit	ASCII code	Unpacked decimal
0	30	00
1	31	01
2	32	02
3	33	03
4	34	04
5	35	05
6	36	06
7	37	07
8	38	08
9	39	09

Here is a table showing the ASCII codes and unpacked decimal

representations of the four numbers in the image:

Number	ASCII code	Unpacked decimal
3	33	03
4	34	04
0	30	00
2	32	02

This image is useful for understanding the relationship between ASCII and unpacked decimal formats. It is also a reminder that all numbers are ultimately represented in binary form, regardless of how they are displayed or stored.

As you can see, the ASCII codes for the digits 0 through 9 are simply the decimal values of the digits shifted by 4 bits. This makes it easy to convert between ASCII and unpacked decimal representations of numbers.

For example, to convert the ASCII code 34 to an unpacked decimal representation, we simply shift the ASCII code right by 4 bits. This gives us the unpacked decimal representation 04, which is the decimal value of 4.

To convert the unpacked decimal representation 05 to an ASCII code, we simply shift the unpacked decimal representation left by 4 bits. This gives us the ASCII code 35, which is the ASCII code for the digit 5.

The fact that the ASCII codes for the digits 0 through 9 are the same as their unpacked decimal representations makes it easy to perform arithmetic on ASCII decimal strings.

For example, we can add two ASCII decimal strings by simply adding

the corresponding ASCII codes for each digit.