AAS, AAM, AAD

The AAS (ASCII adjust after subtraction) instruction is used to adjust the result of a subtraction operation when the result is negative.

It is typically used after a SUB or SBB instruction that has subtracted one unpacked decimal value from another and stored the result in the AL register.

The AAS instruction works by first checking the Carry flag. If the Carry flag is set, it means that the subtraction resulted in a negative number.

In this case, the AAS instruction subtracts 1 from the AH register and sets the Carry flag again.

It also sets the AL register to the ASCII representation of the negative number.

If the Carry flag is not set, it means that the subtraction resulted in a positive number.

In this case, the AAS instruction simply sets the AL register to the ASCII representation of the positive number.

Here is an example of how to use the AAS instruction:

```
83 mov ah, 0 ; clear AH before subtraction
84 mov al, 8
85 sub al, 9 ; subtract 9 from 8
86 aas ; adjust the result
87 or al, 30h ; convert AL to ASCII
```

After the above code has executed, the AL register will contain the ASCII representation of the number -1, which is 45h.

The AAS instruction can be useful for performing arithmetic operations on ASCII decimal strings.

For example, it can be used to subtract two ASCII decimal strings,

even if they have different lengths.

Here is an example of how to use the AAS instruction to subtract two ASCII decimal strings:

```
095 mov esi, offset first_number
096 mov edi, offset second_number
097 mov ecx, length_of_first_number
098
099 loop:
100 mov ah, 0 ; clear AH before subtraction
101 mov al, [esi]
102 sub al, [edi]
103 aas ; adjust the result
104 or al, 30h ; convert AL to ASCII
105 mov [esi], al
106
107 inc esi ; increment the first number pointer
108 inc edi ; increment the second number pointer
109 dec ecx ; decrement the loop counter
110
111 cmp ecx, 0
112 jne loop ; continue looping if the loop counter is not zero
```

This code will subtract the two ASCII decimal strings starting at the least significant digits and working their way up to the most significant digits.

The AAS instruction is used to adjust the result of each subtraction operation to ensure that it is in a valid ASCII decimal format.

The AAS instruction is a powerful tool for performing arithmetic operations on ASCII decimal strings. It is easy to use and can be used to implement a variety of arithmetic algorithms.

==========

AAM (ASCII adjust after multiplication)

==========

he AAM (ASCII adjust after multiplication) instruction is used to convert the binary product produced by the MUL instruction to unpacked decimal format.

The MUL instruction must be used to multiply two unpacked decimal values.

The AAM instruction works by dividing the product by 100 and storing the quotient in the AH register and the remainder in the AL register.

The quotient represents the most significant digit of the unpacked decimal result, and the remainder represents the least significant digit.

Here is an example of how to use the AAM instruction:

```
116 mov bl, 5 ; first operand
117 mov al, 6 ; second operand
118 mul bl ; AX = 001Eh (binary product)
119 aam ; AX = 0300h (unpacked decimal result)
```

After the above code has executed, the AX register will contain the unpacked decimal representation of the product of 5 and 6, which is 30.

The AAM instruction can be useful for performing arithmetic operations on ASCII decimal strings. For example, it can be used to multiply two ASCII decimal strings, even if they have different lengths.

Here is an example of how to use the AAM instruction to multiply two ASCII decimal strings:

```
mov esi, offset first_number
mov edi, offset second_number
mov ecx, length_of_first_number

loop:
mov bl, [esi]
mov al, [edi]
mul bl ; AX = binary product of two digits
aam ; AX = unpacked decimal representation of product

mov [esi], al ; store the least significant digit of the product
mov [edi], ah ; store the most significant digit of the product

inc esi ; increment the first number pointer
inc edi ; increment the second number pointer
dec ecx ; decrement the loop counter

cmp ecx, 0
jne loop ; continue looping if the loop counter is not zero
```

This code will multiply the two ASCII decimal strings starting at the least significant digits and working their way up to the most significant digits.

The AAM instruction is used to convert the binary product of each multiplication operation to unpacked decimal format.

The AAM instruction is a powerful tool for performing arithmetic operations on ASCII decimal strings. It is easy to use and can be used to implement a variety of arithmetic algorithms.

==========

AAD (ASCII adjust before division)

==========

The AAD (ASCII adjust before division) instruction is used to convert an unpacked decimal dividend in AX to binary in preparation for executing the DIV instruction. This is necessary because the DIV instruction can only divide binary numbers.

The AAD instruction works by multiplying the AL register by 100 and

adding the result to the AH register.

This ensures that the AH register contains the most significant digit of the unpacked decimal dividend and the AL register contains the least significant digit.

Here is an example of how to use the AAD instruction:

```
149 mov ax, 0307h ; dividend
150 aad ; AX = 0025h
151 mov bl, 5 ; divisor
152 div bl ; AX = 0207h
```

After the above code has executed, the AX register will contain the quotient and remainder of the division operation, respectively. The quotient is stored in the AL register, and the remainder is stored in the AH register.

The AAD instruction can be useful for performing arithmetic operations on ASCII decimal strings. For example, it can be used to divide two ASCII decimal strings, even if they have different lengths.

Here is an example of how to use the AAD instruction to divide two ASCII decimal strings:

```
mov esi, offset first_number
mov edi, offset second_number
mov ecx, length_of_first_number

loop:
mov bl, [esi]
aad ; AX = unpacked decimal representation of first number

mov ah, 0 ; clear AH before division
mov al, [edi]
div bl ; AX = quotient and remainder of division

mov [esi], al ; store the quotient
mov [edi], ah ; store the remainder

inc esi ; increment the first number pointer
inc edi ; increment the second number pointer
dec ecx ; decrement the loop counter

cmp ecx, 0
jne loop ; continue looping if the loop counter is not zero
```

This code will divide the two ASCII decimal strings starting at the most significant digits and working their way down to the least significant digits.

The AAD instruction is used to convert the unpacked decimal representation of the first number to binary before each division operation.

The AAD instruction is a powerful tool for performing arithmetic operations on ASCII decimal strings. It is easy to use and can be used to implement a variety of arithmetic algorithms.

Questions:

Question: Write a single instruction that converts a two-digit unpacked decimal integer in AX to ASCII decimal.

Answer: To convert a two-digit unpacked decimal integer in AX to ASCII decimal, you can use the AAM (ASCII Adjust AX After Multiplication) instruction:

```
aam
```

Question: Write a single instruction that converts a two-digit ASCII decimal integer in AX to unpacked decimal format.
Answer: To convert a two-digit ASCII decimal integer in AX to unpacked decimal format, you can use the AAD (ASCII Adjust AX Before Division) instruction:

```
aad
```

Question: Write a two-instruction sequence that converts a two-digit ASCII decimal number in AX to binary.
Answer: To convert a two-digit ASCII decimal number in AX to binary, you can use the following two-instruction sequence:

```
186 sub al, 30h    ; Convert the tens digit to binary
187 aam            ; Adjust AX to form the binary value
```

Question: Write a single instruction that converts an unsigned binary integer in AX to unpacked decimal.
Answer: To convert an unsigned binary integer in AX to unpacked decimal, you can use the AAD (ASCII Adjust AX Before Division) instruction:

```
aad
```