Arithmetic Instructions

ADD

Examples:

```
4000    DE46 add.w d6,d7
4002    D679      add.w $2000,d3
4004    0000
4006    2000
4008    D979      add.w d4, $2002
400A    0000
400C    2002
```

Arithmetic Instructions

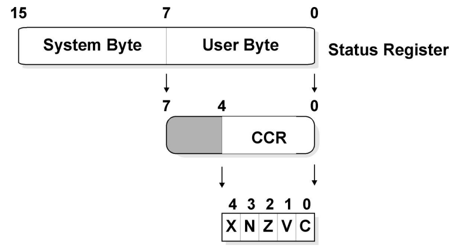
Subtract

Examples:

```
4000    9E46    sub.w d6,d7
4002    9679    sub.w $2000,d3
4004    0000
4006    2000
4008    9979    sub.w d4, $2002
400A    0000
400C    2002
```

The Condition Code Register

The **c**ondition **c**ode **r**egister (**CCR**) (least significant 5 bits of the status register) provides extra information about the result of instruction execution.



Condition Code Register Update

The **CCR** is updated after **EACH** instruction is executed. The effect on the CCR depends on the instruction.

Example:

```
move.b  #$ff,d0   clears C
add.b   #1,d0     sets C
```

C - Carry Bit

- Set if previous operation caused a carry (corresponds to borrow in the case of subtract).
- It is Cleared otherwise.

Before:

D0	D1
$ff	$02

CCR:

N	Z	V	C
0	0	0	0

Instruction: `add.b d0,d1`

After:

D0	D1
$ff	$01

CCR:

N	Z	V	C
0	0	0	1

V - oVerflow Bit

Indicates if the previous operation caused an overflow.

-> Result is outside signed number range.

It is Cleared otherwise.

Before:

D0	D1
$7f	$02

CCR:

N	Z	V	C
0	0	0	0

Instruction: `add.b d0,d1`

After:

D0	D1
$7f	$81

CCR:

N	Z	V	C
1	0	1	0

Z - Zero Bit

- Indicates if the last operation resulted in Zero result. It is cleared otherwise.
- Note: If Z = 1 then the result was 0.

Before:

D0	D1
$00	$F2

CCR:

N	Z	V	C
0	0	0	0

Instruction: `move.b d0,d1`

After:

D0	D1
$00	$00

CCR:

N	Z	V	C
0	1	0	0

N - Negative Bit

- Set if previous operation produced a negative result.
- It is cleared otherwise.

Before:

D0	D1
$02	$04

CCR:

N	Z	V	C
0	0	0	0

Instruction: `sub.b d0,d1`

After:

D0	D1
$02	$FE

CCR:

N	Z	V	C
1	0	0	0

X - eXtend Bit

- The eXtend flag is used for multiple precision operation where it acts as a carry.
- Precision: **Detail used to represent a measurement.**
- Usually specified by the number of significant digits.
- 68332 can handle 32-bit operations. Use of **X** allows arbitrary presision arithmetric.

Example: 64-bit Operation

```
  0000 0000  ffff ffff
+ 0000 0000  0000 0001
  0000 0001  0000 0000
```

We must split this into **two** 32-bit operations.

Split Operation

```
move.l #$00000000,d0        *upper 32-bits
move.l #$ffffffff,d1     *lower 32-bits
move.l #$00000000,d2
move.l #$00000001,d3

add.l d3,d1                 *lower 32-bits
add.l d2,d0                 *upper 32-bits
```

The result is: d0 = $00000000
d1 = $00000000 -> **clearly wrong**

Why?

- Didn’t carry across the carry between 1st and 2nd operation.
- The second operation should be:
  - **add.l [d2 + carry from previous operation], d0**
- The eXtend flag records the carry bit from the last operation. We add its value to the 2nd opeartion using the new instruction **addx.**

Addx?

```
move.l #$00000000,d0     *upper 32-bits
move.l #$ffffffff,d1     *lower 32-bits
move.l #$00000000,d2
move.l #$00000001,d3

add.l d3,d1              *lower 32-bits
addx.l d2,d0             *upper 32-bits + carry
```

We do not use the **C-flag** as it is cleared by the **move** instruction which is often used during a calculation.

Example: X - eXtend Bit

Before:

D0	D1
$01	$ff

CCR:	X	N	Z	V	C
	0	0	0	0	0

Instruction: `add.b d0,d1`

After:

D0	D1
$01	$00

CCR:	X	N	Z	V	C
	1	0	1	0	1

CCR-flags

- CCR flags behave as follows after execution of an instruction:
  - **-** -> not affected by the operation
  - ***** -> set according to the result
  - **0** -> cleared
  - **1** -> set
  - **?** -> undefined after the operation

Example: How move affects the CCR

CCR:	X	N	Z	V	C
	-	*	*	0	0