Pseudo-operatori

Lo pseudo-operatore ORG

- Viene usato per inizializzare il Program Location Counter (PLC)

Sintassi:

```
ORG    $HEXADDR
```

Lo pseudo-operatore END

- Viene usato per terminare il processo di assemblaggio e saltare all'entry point del programma

Sintassi:

```
END    TARGETLAB
```

Pseudo-operatori

DS

Viene usato per incrementare il Program Location Counter (PLC), in modo da riservare spazio di memoria per una variabile

Sintassi: LABEL DS.W NUMSKIPS

DC

Viene usato per inizializzare il valore di una variabile

Sintassi: LABEL DC.W VALUE

EQU

Viene usato per stabilire un'identità

Sintassi: LABEL EQU VALUE

L’istruzione CLR

CLR Clear an operand

Operation: `[destination] ← 0`

Syntax: `CLR <ea>`

Sample syntax: `CLR (A4)+`

Attributes: Size = byte, word, longword

Description: The destination is cleared — loaded with all zeros. The `CLR` instruction can't be used to clear an address register. You can use `SUBA.L A0,A0` to clear A0. Note that a side effect of `CLR`'s implementation is a *read* from the specified effective address before the clear (i.e., write) operation is executed. Under certain circumstances this might cause a problem (e.g., with write-only memory).

Condition codes:

X	N	Z	V	C
-	0	1	0	0

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔		✔	✔	✔	✔	✔	✔	✔			

L'istruzione ADD

ADD Add binary

Operation: [destination] ← [source] + [destination]

Syntax: ADD <ea>,Dn
ADD Dn,<ea>

Attributes: Size = byte, word, longword

Description: Add the source operand to the destination operand and store the result in the destination location.

Condition codes:

X	N	Z	V	C
*	*	*	*	*

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Destination operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✓		✓	✓	✓	✓	✓	✓	✓			

L'istruzione SUBQ Subtract quick

Operation: [destination] ← [destination] - <literal>

Syntax: `SUBQ #<data>,<ea>`

Attributes: Size = byte, word, longword

Description:

Subtract the immediate data from the destination operand. The immediate data must be in the range 1 to 8.

Word and longword operations on address registers do not affect condition codes. A word operation on an address register affects the entire 32-bit address.

Condition codes:

X	N	Z	V	C
*	*	*	*	*

L'istruzione MOVE

MOVE Copy data from source to destination

Operation: [destination] ← [source]

Syntax: MOVE <ea>,<e>

Sample syntax:

```
MOVE (A5),-(A2)
MOVE -(A5),(A2)+
MOVE #$123,(A6)+
MOVE Temp1,Temp2
```

Attributes: Size = byte, word, longword

Description: Move the contents of the source to the destination location. The data is examined as it is moved and the condition codes set accordingly. Note that this is actually a *copy* command because the source is not affected by the move. The move instruction has the widest range of addressing modes of all the 68000's instructions.

Condition codes:

X	N	Z	V	C
-	*	*	0	0

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Destination operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔		✔	✔	✔	✔	✔	✔	✔			

L’istruzione Compare

Operation: [destination] - [source]

Syntax: `CMP <ea>,Dn`

Sample syntax: `CMP (A6),D2`

Attributes: Size = byte, word, longword

Description:

Subtract the source operand from the destination operand and set the condition codes accordingly. The destination must be a data register. The destination is not modified by this instruction.

Condition codes:

X	N	Z	V	C
–	*	*	*	*

Compare memory with memory

Operation: [destination] - [source]
Syntax: `CMPM (Ay)+,(Ax)+`
Attributes: Size = byte, word, longword

Description:
Subtract the source operand from the destination operand and set the condition codes accordingly. The destination is not modified by this instruction. The only permitted addressing mode is address register indirect with post-incrementing for both source and destination operands.
Application:
Used to compare the contents of two blocks of memory.

Condition codes:

```
X N Z V C
- * * * *
```

L'istruzione Branch on condition

Operation: IF cc = 1 THEN [PC] $\leftarrow$ [PC] + d

Syntax: `Bcc <label>`

Attributes: Bcc takes an 8-bit or 16-bit offset (displacement)

Description:

If the specified logical condition is met, program execution continues at location [PC] + displacement, d.

The displacement is a two's complement value.

L'istruzione Branch on condition

Bcc — Branch on condition cc

Operation: `If cc = 1 THEN [PC] ← [PC] + d`

Syntax: `Bcc <label>`

Sample syntax:
```
BEQ Loop_4
BVC *+8
```

Attributes: BEQ takes an 8-bit or a 16-bit offset (i.e., displacement).

Description: If the specified logical condition is met, program execution continues at location [PC] + displacement, d. The displacement is a two's complement value. The value in the PC corresponds to the current location plus two. The range of the branch is -126 to +128 bytes with an 8-bit offset, and -32K to +32K bytes with a 16-bit offset. A short branch to the next instruction is impossible, since the branch code 0 indicates a long branch with a 16-bit offset. The assembly language form BCC *+8 means branch to the point eight bytes from the current PC if the carry bit is clear.

BCC	branch on carry clear	$\overline{C}$
BCS	branch on carry set	C
BEQ	branch on equal	Z
BGE	branch on greater than or equal	$N.V + \overline{N}.\overline{V}$
BGT	branch on greater than	$N.V.\overline{Z} + \overline{N}.\overline{V}.\overline{Z}$
BHI	branch on higher than	$\overline{C}.\overline{Z}$
BLE	branch on less than or equal	$Z + N.\overline{V} + \overline{N}.V$
BLS	branch on lower than or same	$C + Z$
BLT	branch on less than	$N.\overline{V} + \overline{N}.V$
BMI	branch on minus (i.e., negative)	N
BNE	branch on not equal	$\overline{Z}$
BPL	branch on plus (i.e., positive)	$\overline{N}$
BVC	branch on overflow clear	$\overline{V}$
BVS	branch on overflow set	V

Condizioni cc da cui dipende Bcc

Single bit	
BCS branch on carry set	$C = 1$
BCC branch on carry clear	$C = 0$
BVS branch on overflow set	$V = 1$
BVC branch on overflow clear	$V = 0$
BEQ branch on equal (zero)	$Z = 1$
BNE branch on not equal	$Z = 0$
BMI branch on minus (i.e., negative)	$N = 1$
BPL branch on plus (i.e., positive)	$N = 0$
Signed	
BLT branch on less than (zero)	$N \oplus V = 1$
BGE branch on greater than or equal	$N \oplus V = 0$
BLE branch on less than or equal	$(N \oplus V) + Z = 1$
BGT branch on greater than	$(N \oplus V) + Z = 0$
Unsigned	
BLS branch on lower than or same	$C + Z = 1$
BHI branch on higher than	$C + Z = 0$

DBcc Test condition, decrement, and branch

Operation: [Dn] ← [Dn] - 1 {decrement loop counter}
IF [Dn] != 0
[PC] ← [PC] + d {take branch}
ELSE [PC] ← [PC] + 2 {goto next instruction}

Syntax: `DBcc Dn,<label>`

Attributes: Size = word

Description: The DBcc instruction provides an automatic looping facility and replaces the usual decrement counter, test, and branch instructions. Three parameters are required by the DBcc instruction: a branch condition (specified by ‘cc’), a data register that serves as the loop down-counter, and a label that indicates the start of the loop. The 14 branch conditions supported by Bcc are also supported by DBcc, as well as DBF and DBT (F = false, and T = true). Note that many assemblers permit the mnemonic DBF to be expressed as DBRA (i.e., decrement and branch back).

L’istruzione JMP

JMP — Jump (unconditionally)

Operation: [PC] ← destination

Syntax: JMP <ea>

Attributes: Unsized

Description: Program execution continues at the effective address specified by the instruction.

Application: Apart from a simple unconditional jump to an address fixed at compile time (i.e., JMP label), the JMP instruction is useful for the calculation of *dynamic* or *computed* jumps. For example, the instruction JMP (A0,D0.L) jumps to the location pointed at by the contents of address register A0, offset by the contents of data register D0. Note that JMP provides several addressing modes, while BRA provides a single addressing mode (i.e., PC relative).

Condition codes:

X	N	Z	V	C
-	-	-	-	-

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
		✔			✔	✔	✔	✔	✔	✔	

Esempio - Moltiplicazione di due interi

```
*   Programma per moltiplicare MCND e MPY
*
      ORG       $8000
*
MULT  CLR.W        D0          D0 accumula il risultato
      MOVE.W    MPY,D1      D1 e' il contatatore di ciclo
      BEQ       DONE        Se il contatore e' zero e' finito
LOOP  ADD.W     MCND,D0     Aggiunge MCND al prodotto
parziale
      ADD.W     #-1,D1      Decrementa il contatore
      BNE       LOOP        e ripete il giro
DONE  MOVE.W    D0,PROD     Salva il risultato

PROD  DS.W      1           Riserva spazio di memoria per
PROD
MPY   DC.W      3           Definisce il valore di MPY
MCND  DC.W      4           Definisce il valore di MCND
      END          MULT       Fine ass., salto a entry point
```

Esempio – Somma dei primi N numeri

```
START   CLR.W   SUM
        MOVE.W  ICNT,D0
ALOOP   MOVE.W  D0,CNT
        ADD.W   SUM,D0
        MOVE.W  D0,SUM
        MOVE.W  CNT,D0
        ADD.W   #-1,D0
        BNE     ALOOP
        JMP     SYSA
SYSA    EQU     $8008
CNT     DS.W    1
SUM     DS.W    1
IVAL    EQU     17
ICNT    DC.W    IVAL
```

Esempio – prodotto scalare fra 2 vettori

```
        ORG     $8000
START   MOVE.L #A,A0
        MOVE.L #B,A1
        MOVE.L #N,D0
        SUBQ    #1,D0
        CLR     D2
LOOP   MOVE    (A0)+,D1
        MULS   (A1)+,D1
        ADD     D1,D2
        DBRA    D0,LOOP
        MOVE    D2,C
DONE    JMP     DONE
N       EQU     $000A
        ORG     $80B0
A       DC.W    1,1,1,1,1,1,1,1,1,1
        ORG     $80D0
B       DC.W    1,1,1,1,1,1,1,1,1,1
C       DS.L    1
```

Esempio – ricerca token in stringa

```
                ORG         $8000
START       MOVEA.L     #STRING,A0
            MOVE.B      #TOKEN,D0
LOOP        CMP.B       (A0)+,D0
            BNE         LOOP
FOUND       SUBQ.L      #1,A0
            MOVE.L      A0,TOKENA

            ORG         $8100
TOKEN   EQU         ':'
STRING      DC.B    'QUI QUO:QUA'
TOKENA      DS.L    1
```