1 Linguaggio assembler

1.1 1. Generalità sui linguaggi assembler

Un linguaggio assemblativo (o assembler) è un linguaggio di basso livello, relativamente alla semantica dei codici che hanno una corrispondenza diretta con codici operativi del relativo processore. Un linguaggio assembler supporta i tipi di dato gestiti dal processore. Il formato istruzione è fisso. Mediante un tale linguaggio è, quindi, possibile tradurre un programma sorgente direttamente in linguaggio macchina. Nel caso di assembler detti assoluti il modulo oggetto prodotto è già in formato eseguibile e può essere caricato in memoria mediante un'operazione di "Down Load" ed eseguito dal processore (fig.1).

Un linguaggio assembler possiede, inoltre, dei comandi di servizio per l'assemblatore, detti "Direttive di Assemblaggio" o " codici", che servono ad istruire l'assemblatore su come assemblare il programma e a fornire al programmatore istruzioni di più alto livello in grado di semplificare la programmazione (definizione di strutture dati e desiderata di altro tipo).

I linguaggi assemblativi possono generare, all'occorrenza, moduli oggetto in formato rilocabile. Tali moduli, mediante l'azione di uno strumento detto collegatore (Linker), sono raggruppati in modo da formare un unico modulo assoluto eseguibile (o talvolta rilocabile a meno di una costante di spiazzamento). Tale modulo, per tramite dell'azione di un ulteriore strumento, detto caricatore (Loader), può essere caricato nella memoria programma del processore ed essere, quindi, eseguito (fig.2). In certi casi gli strumenti linker e loader possono essere fusi in un unico strumento che, in ogni caso, ne mantiene distinte le funzionalità.

Esistono, inoltre, linguaggi assemblativi detti assembler strutturati o macro assembler che forniscono, oltre alle funzionalità presenti negli assembler tradizionali, istruzioni di controllo ad alto livello (if, while, repeat, for), direttive per la creazione di strutture dati complesse e la loro inizializzazione, l'impiego di formati istruzione variabili e la possibilità di definire ed utilizzare macro.

I macro assemblatori (macroassembler), unitamente all'azione dei collegatori-caricatori (linker-loader) consentono di strutturare i programmi assembler prodotti da un programmatore o da compilatori secondo schemi che garantiscono oltre alla modularità anche la rilocabilità di tali oggetti.

Un programma assembler può essere organizzato in aree o segmenti specializzati. Tali segmenti possono contenere dati o programmi. Di solito un programma processato da un assemblatore e collegato da un linker dispone di una sola area programmi e di più aree dati (ad es. area dati globale (area common), area dati comune a più segmenti (common labellato), aree dati locali a procedure di tipo dinamico (create all'atto dell'istanziazione di una procedura e rilasciate alla fine della procedura) poste in strutture stack.

I macroassembler offrono al programmatore la possibilità di disporre di un numero fisso di moduli con relativi "location counter" per sviluppare programmi organizzati in sezioni a cui corrisponderanno un numero uguale o inferiore di classi di allocazione gestite successivamente dal linker.

Ad esempio i sistemi di sviluppo MOTOROLA per il 68000 dispongono di un macroassembler che gestisce 16 moduli. Il Linker associato accorpa tali moduli al massimo in quattro classi di allocazione. Tale valore è dovuto al fatto che il sistema di sviluppo utilizza per la gestione della memoria un dispositivo di "Memory Management Unit" MMU dotato proprio di quattro segmenti. Nello sviluppo dell'applicazione il programmatore struttura il programma in moduli (max 16) e passa da un modulo all'altro nominando il relativo location counter ($0-$15). Il linker provvederà ad aggregare moduli appartenenti allo stesso location counter e tramite l'uso di direttive di collegamento accorperà più

moduli in un'area associata ad uno dei quattro segmenti di programma. Ciascuno dei segmenti può essere etichettato come RO (sola lettura) o R/W (a lettura scrittura).

Un linguaggio assemblativo è caratterizzato da un insieme di oggetti mnemonici e simboli che rappresentano:

- operazioni eseguibili (codici operativi);
- direttive di assemblaggio (pseudo codici);
- nomi simbolici (label);
- operatori;
- simboli speciali.

La sintassi e la semantica del linguaggio assembler descrivono pienamente il modo in cui tali oggetti possono essere aggregati dall'utente programmatore per sviluppare programmi assembler per uno specifico processore.

Al fine di pervenire a linguaggi assembler il più possibile standardizzati ed indipendenti dallo specifico processore, è stato appositamente definito uno standard IEEE [...] che stabilisce le linee guida per la definizione di un generico linguaggio assemblativo.

1.2 2. Programma Sorgente

è costituito da una sequenza di istruzioni in formato simbolico (tipicamente formato testo in ASCII) che descrive uno specifico algoritmo. Le istruzioni possono essere: di controllo flusso, di trasferimento o di trasformazioni di un generico dato (semplice o strutturato).

Se l' assembler è a formato fisso ogni istruzione assembler è rappresentata su di una linea di testo (tipicamente non superiore ad 80 caratteri) se a formato libero (assembler strutturati e macroassembler) anche su più linee.

1.3 3. Programmi assoluti e rilocabili

Un riferimento in memoria (indirizzo) è detto assoluto se esso viene definito a tempo di compilazione (o assemblaggio) relativo se definito in epoco diverse, a seguito dell'azione di un linker (relativo statico) o a tempo di esecuzione (relativo dinamico).

Un programma assemblativo (o in breve un programma) è di tipo assoluto se tutti gli indirizzi (riferimenti) dei dati adoperati e dei salti a istruzioni sono assoluti. Un tale programma, quando assemblato, produce un modulo oggetto assoluto che si presenta in formato immagine di memoria ciò in quanto, quando caricato in memoria per una esecuzione, occuperà le stesse posizioni presenti nell'imagine. Un programma assoluto utilizza sempre una direttiva ORG (ORIGIN) che fissa il valore del location counter. Si fa osservare che il program counter, per tale tipo di programmi, assumerà gli stessi valori del location counter limitatamente alle istruzioni eseguite.

Con rilocazione si intende quel processo che consente di creare una corrispondenza fra uno spazio indirizzi logico (creato a tempo di compilazione) ed uno fisico o virtuale (creato a tempo di collegamento o di esecuzione).

Un programma assemblativo è di tipo rilocabile se l'indirizzo di caricamento del programma può essere determinato a tempo di caricamento (rilocabile staticamente) o a tempo di esecuzione

(rilocabile dinamicamente). Un codice è rilocabile se è indipente dalla posizione in cui esso viene posto in memoria.

La rilocabilità statica si rende necessaria ad esempio nel caso in cui si vogliano produrre librerie di programmi da memorizzare in memorie a sola lettura o nel caso in cui un programma debba operare su una classe di sistemi simili ma differenti nella configurazione o nel caso in cui un programma è composto da moduli che dovranno essere posizionati in tempi differenti e negli stessi spazi o nel caso in cui si dispone di un S.O. che carica i programmi dopo aver ottenuto in modo statico un'area di memoria da un apposito gestore (ad es. il S.O MSDOS).

La rilocabilità dinamica si rende necessaria quando un modulo eseguibile deve essere spostato in memoria nel tempo e nello spazio (swap) a causa di esigenze derivanti dall'uso di sistemi di tipo multitasking. In tal caso, infatti, essendo le risorse memoria e processore condivise fra più task di uno stesso programma o di programmi differenti, e non potendo essere quindi assegnate in modo permanente richiedono il riposizionamento dei moduli eseguibili in aree di memoria diverse.

Una rilocazione di tipo statico potrebbe essere realizzata modificando ogni qualvolta si vuole rilocare il codice la direttiva ORG. Ciò richiederebbe, però, la ricompilazione del programma. Alternativamente l'assemblatore può produrre un modulo oggetto contenente un numero di informazioni aggiuntive necessarie per consentire al linker la generazione del modulo oggetto rilocabile. Quest'ultimo provvede a sviluppare una azione di modifica degli indirizzi generati dall'assemblatore all'atto del caricamento di un programma.

All'atto della scrittura di un programma vengono adoperati dei nomi simbolici (nomi di variabili, procedure locali e di libreria, label) che prima di una esecuzione del programma dovranno essere convertiti in indirizzi di memoria collegati ad indirizzi fisici di memoria centrale. Tale collegamento può essere fatto in diversi momenti e precisamente:

Per Allocazione Statica

all'atto della stesura di un programma

all'atto della traduzione del programma

durante la fase di collegamento per tramite di un linker

durante la fase di caricamento

Per Allocazione Dinamica a tempo di esecuzione

polarizzando mediante costanti di rilocazione (costanti di spiazzamento) gli indirizzi logici

collegando procedure e nomi esterni, non noti all'inizio dell'esecuzione del programma.

I meccanismi da adoperare per l'assolutizzazione di un programma dipendono dal sistema calcolatore in uso e dal suo sistema operativo. L'architettura di un semplice sistema di controllo processo, ad esempio non fa alcun uso dello strato di sistema operativo in quanto su esso opererà direttamente sullo strato hardware un'unica applicazione. In tal caso l'assoluto eseguibile sarà generato staticamente, non dovendo subire alcuna forma di rilocazione dinamica.

Un sistema di controllo più complesso opera, ad esempio, su di un sistema multitasking, in cui lo strato hardware è affiancato da un livello di supporto a tempo di esecuzione in grado di offrire le funzionalità necessarie per il controllo dei task. In un tale sistema l'applicazione, anche se unica, viene strutturata in moduli da assegnare a task che opereranno in regime di quasiconcorrenza. Non esisterà, quindi, un unico programma eseguibile ma più moduli eseguibili che possibilmente non potranno essere tutti simultaneamente residenti in memoria, ma parte residenti e parte posti in file su

memorie secondarie (swap su dischi). In tal caso i moduli eseguibili non possono essere assoluti (con tutti i riferimenti già collegati agli indirizzi fisici di memoria) ma dovranno poter godere della proprietà della rilocabilità. Cioé ciascuno di essi dovrà essere assolutizzato a meno di una costante di spiazzamento dipendente, all'atto del caricamento in memoria (swap-in) dall'indirizzo iniziale in cui il modulo verrà caricato (load point o load address). Tale costante dovrà essere sommata a tutti i riferimenti in memoria per ottenere i riferimenti assoluti corretti. Se tale azione viene fatta una tantum (rilocazione statica) la modifica può essere fatta direttamente dal linker, ma se deve avvenire dinamicamente per far fronte ad operazioni di swap-in e swap-out la modifica verrà fatta direttamente durante la preparazione degli indirizzi, sommando all'indirizzo posto nell'istruzione il contenuto di un registro su cui l'esecutivo (il RTS) ha posto la costante di spiazzamento all'atto del caricamento del modulo eseguibile o del task.

Lo schema diventa ancora più complicato se si dispone di un sistema di calcolo dotato di sistema operativo multitasking e multiuser (ad esempio UNIX) e con gestione segmentata o paginata della memoria o, più in generale, con l'impiego della memoria virtuale. In tal caso la presenza di indirizzi di memoria virtuali richiede due livelli di assolutizzazione: una prima assolutizzazione dello spazio degli indirizzi del modulo eseguibile rispetto agli indirizzi virtuali e una seconda assolutizzazione degli indirizzi virtuali rispetto agli indirizzi fisici. Se la gestione della memoria è segmentata il procedimento di assolutizzazione composto dai due passi precedenti è a carico del programmatore. Se la memoria è virtuale, il programmatore dovrà occuparsi di utilizzare strumenti in grado di assolutizzare i riferimenti soltanto rispetto agli indirizzi virtuali, in quanto l'assolutizzazione di tali indirizzi in indirizzi fisici è a carico del S.O.

Un ulteriore meccanismo che garantisce la rilocabilità di un programma, rispetto ad uno spazio di indirizzi virtuali, senza ricorrere ad interventi esterni di un S.O. consiste nel generare, facendo uso di opportuni modi di indirizzamento, programmi detti "indipendenti dalla posizione".

1.4 4. Indipendenza dalla posizione

un codice è detto indipendente dalla posizione se al suo interno non esistono indirizzi effettivi coincidenti con riferimenti assoluti di memoria. L'indipendenza dalla posizione di un codice si può ottenere non facendo uso della direttiva ORG e ricorrendo a modi di indirizzamento di tipo relativo o polarizzato con o senza impiego di indicizzazione. Va, inoltre tenuto conto che:

- Una costante numerica è di tipo assoluto
- Il Location Counter è rilocabile
- Un simbolo associato ad una label di uno statement che non sia una direttiva EQU è rilocabile
- Un simbolo che compare nel campo label di una direttiva EQU ha lo stesso attributo del valore riportato nel campo operando dell'istruzione
- Una espressione che interessa soltanto costanti assolute e simboli porta ad un valore assoluto
- Espressioni che coinvolgono simboli rilocabili ed assoluti producono valori rilocabili nei seguenti casi: R, R+A, R-A. Valori assoluti nei seguenti: R-R.
- Non sono ammessi operatori differenti da "+" e "-" e le forme "R+R", "A-R".

1.5 5. Schema di rilocazione adottato da Assembler-Linker-loader della famiglia 680xx

Lo schema di rilocazione e collegamento per tale famiglia di processori che include tutti i processori ad 8 bit prodotti da Motorola (6800, 6801,6805 e 6809) è stato progettato per fornire le seguenti funzionalità:

- rilocazione di un programma;
- collegamento di più moduli programma;
- facile sviluppo di programmi per essere allocati su RAM/ROM;
- facile specifica di qualunque modo di indirizzamento;
- possibilità di impiego di un'Area Common di tipo bianco non inizializzata;
- possibilità di impiego di un'Area Common etichettata (o con nome) inizializzata.

Un programma assembler per tale famiglia di processori può essere organizzato secondo uno schema che prevede le seguenti cinque differenti sezioni:

- ASCT Absolute Section: è questa una sezione non rilocabile che il programmatore può utilizzare per definire variabili assolute ed inizializzare aree di memoria. E' questa, ad esempio, la sezione dove possono essere dichiarate le variabili associate a dispositivi di I/O "memory mapped".
- BSCT Base Section: è questa una unica sezione rilocabile utilizzata per allocarvi variabili che devono essere riferite utilizzando il modo di indirizzamento diretto. Tale sezione è limitata alle prime 256 locazioni di memoria (0-255).
- CSCT Blank Common Section: è questa una unica sezione rilocabile non inizializzabile dall'utente. Tale sezione è usata per realizzare un'area Common così come previsto da linguaggi come il FORTRAN.
- DSCT Data Section: è un'unica sezione rilocabile utilizzata per allocare in memoria RAM variabili riferite mediante il modo di indirizzamento extended.
- PSCT Program Section: è un'unica sezione rilocabile simile alla DSCT destinata a contenere l'area programmi (da mettere ad es. in ROM).
- Le sezioni common con nome possono essere allocate (interamente) sia in BSCT che in DSCT o PSCT.

1.5.1.1.1.1.1 I modi di indirizzamento supportati dal 6809 sono i seguenti:

-Immediato

-Relativo

-Diretto ed Esteso

-Indicizzato

1.5.1.2 5.1 Esempio di programma assembler organizzato in moduli rilocabili

```
MAIN            IDNT          1,0    MODULO DI ESEMPIO
          OPT           CRE
          XREF.S 9:THISLINE,ARRAY
          XREF          FIND

ALLREG    REG           D0-D7/A0-A7
TEMP            SET           44
FIXED           EQU           @36
COLUMN    EQU           4

          OFFSET 0
ROW1            DS.L          4
ROW2            DS.L          4
ROW3            DS.L          4
ROW4            DS.L          4

          SECTION       1
START:
          LEA           ARRAY,A5
          MOVE.L ROW2(A5),D3
          ADD.L         ROW4+COLUMN(A5),D3
          BEQ.L         NOFIND
          MOVEM         ALLREG,-(A7)
          MOVE.B 'A',D0
          BSR.L         FIND
          MOVEM         (A7)+,ALLREG
NOFIND    CLR.L         D7
          ADDA          #TEMP,A3
TEMP            SET           TEMP+FIXED
          CMP           #4,D0
          BHI           NOFIND
          END           START
```


2 6. Assembler 68000

Le seguenti note descrivono l'assembler assoluto del processore Motorola MC68000 supportato dallo strumento ASIMTOOL. L' assembler è in grado di generare codici in linguaggio macchina (moduli oggetto assoluti) che possono essere eseguiti mediante lo strumento di simulazione di sistemi ASIM su configurazioni di sistemi dotati di processore MC68000 o su altri sistemi fisici compatibili con le configurazioni adottate in ASIM.

Lo scopo del presente manuale è quello di mettere lo studente in condizione di sviluppare semplici programmi in linguaggio assembler. Per quel che concerne il dettaglio dei codici operativi si rinvia al manuale tecnico d' utente del processore edito direttamente da Motorola [Motorola MC68000UMIAD]. In appendice A è, comunque, fatta un breve riepilogo di detti codici sia in termini sintattici che semantici.

2.1.1.1 6.1 Formato istruzione del linguaggio assembler per MC68000

Il formato istruzione supportato dall' assembler 68000 di ASIM, così come altri tipi di assembler, è a formato fisso, ogni istruzione è codificata in una linea di massimo 80 caratteri e ha il seguente formato [1]::

```
Line Number      LABEL OPERATION      OPERAND,OPERAND      COMMENT
```

ad es.

```
                              12 start: MOVE.B A0,2(A3)
```

Line Number: è un campo generato dall' assemblatore che fa precedere ciascuna linea con un numero di sequenza fino a quattro cifre decimali.

LABEL: tale campo è costituito da un nome simbolico (costruito secondo le regole di formazione di nomi vedi §xx) può iniziare in colonna 1 e terminare con uno spazio o essere posizionato in qualunque colonna della linea e terminare con il carattere ":". Una linea può essere contenere la sola label. Alla label viene assegnato il valore corrente del location counter. Essa può essere usata ovunque nell' ambito del sorgente in cui è definita.

OPERATION: il campo operation segue il campo label da cui è separato da almeno uno spazio. Esso può contenere: un codice operativo mnemonico del repertorio dei codici del 68000 (si veda app. xx); una direttiva di assemblaggio; un NULL, in tal caso viene definita in tabella simboli la label, assegnandole il valore del location counter. Il campo non può iniziare in colonna 1 (verrebbe interpretato come label). Ad esso può essere associato uno specificatore della lunghezza del campo dato postponendolo dopo il simbolo ".". Tale specificatore può assumere i valori {B=byte, W=word, L=long word, S=short}.

OPERAND: il campo operandi, se presente, segue il campo operation e ne è separato da almeno uno spazio. Se l' istruzione prevede l' uso di più operandi, questi sono fra loro separati da virgole e senza inclusione di spazi. Nel caso più comune viene utilizzato un formato a due operandi di cui il primo fa le funzioni di campo sorgente ed il secondo di campo destinazione (attenzione la notazione è opposta a quella usata per i processori Intel). Un esempio di tale istruzione è il seguente:

```
loop:          MOVE.W <sorg>,<dest> dest-->sorg
```

COMMENT: Il campo commento inizia dopo il campo operandi, da cui è separato da almeno uno spazio, e può contenere un qualunque carattere del set ASCII stampabile. Esso è opzionale; anche se presente è ignorato dall' assembler. Viene solo utilizzato per essere incluso nel file di listing prodotto a valle di una fase di compilazione. Un campo commento può interessare l' intera linea se questa inizia con il carattere "*". Inoltre, un campo commento può iniziare anche dopo il campo label, sempre facendolo precedere dal simbolo "*".

2.1.1.2 6.2 Simboli

I simboli riconosciuti dall' assembler consistono di uno o più caratteri del set ASCII. Sebbene questi possono essere di lunghezza qualunque l'assembler ne considera significativi soltanto i primi 8. Un simbolo deve iniziare o con uno dei seguenti caratteri: ("A..Z", ".",). I restanti caratteri possono essere oltre ai simboli precedenti anche i simboli di "_" e "$". L'assemblatore non fa alcuna differenza fra caratteri maiuscoli e minuscoli. Tutti i caratteri sono comunque considerati maiuscoli per cui non c'è differenza ad esempio fra MOVE e move, CICLO e ciclo, etc.

[1] Si fa presente che l'assembler in oggetto, a differenza di altri linguaggi, non fa alcuna distinzione fra caratteri maiuscoli e minuscoli. Il programmatore è libero di usare, ai fini di una maggiore leggibilità, il formato che più gli aggrada.

I simboli sono rappresentati come interi espressi su 32 bit.

I simboli possono appartenere a due classi differenti (essi vengono, di fatto, utilizzati in due contesti differenti: nel campo operation di una istruzione, ad es. direttive assembler e mnemonici delle istruzioni, e nei campi label ed operand, ad es. label definite dal programmatore e nomi di registri. Stante i differenti contesti in cui tali simboli vengono adoperati, è possibile utilizzare lo stesso simbolo per denotare cose differenti come ad es. nella seguente istruzione: MOVE MOVE D0,D5, in cui si fa uso della stessa stringa MOVE per denotare la label ed il codice operativo. Nell'ambito di ciascuna classe non è, ovviamente, ammessa la ridefinizione di un simbolo. I simboli riservati dell'assembler sono da intendersi tali, limitatamente alla classe di appartenenza e non per l'altra classe.

2.1.1.3 6.3 Espressioni

Un' espressione è composta da uno o più simboli ed è caratterizzata da operandi composti mediante operatori.

L'assembler riconosce i seguenti operatori suddivisi nelle seguenti 6 classi in ordine di precedenza decrescente:

A)	(...)	sottoespressioni in parentesi
B)	-	meno unario
	~	negazione bit -a-bit (complemento ad uno)
C)	<<	scorrimento a sinistra
	>>	scorrimento a destra
D)	&	and bit-a-bit
	!	or bit-a-bit
E)	*	prodotto
	/	divisione intera
	\	resto modulo
F)	+	somma
	-	sottrazione

Tutti gli operatori sono valutati da sinistra a destra ad eccezione di quelli del gruppo B) in cui la valutazione avviene da destra a sinistra.

2.1.1.4 6.4 Base di numerazione

L' assembler è in grado di riconoscere numeri espressi in base:

decimale	stringa di cifre (0..9), ad es. 1298;
esadecimale	stringa di cifre (0..9,A..F) preceduta dal carattere "$, ad es. $B129F;

ottale	stringa di cifre (0..7) preceduta dal carattere "@", ad es.	@2754;
binaria	stringa di cifre (0,1) preceduta dal carattere "%", ad es.	%10010110

L' assemblatore gestisce, indipendentemente dalla base, numeri che possono essere rappresentati su 32 bit (in caso contrario genera un warning).

2.1.1.5 6.5 Costanti ASCII

Una costante ASCII consiste di una stringa di non più di 4 caratteri racchiusa tra apici (') che può essere memorizzata in una voce di 32 bit (1 byte/stringa). Un doppio apice (") all' interno della stringa indica l' apice ('). Le stringhe ASCII sono giustificate a sinistra e riempite di zeri (se necessario). Ad esempio se la costante è di un solo carattere, questo è posto nel byte meno significativo della voce di 32 bit, se di due, nei 2 byte meno significativi, se di quattro nell' intera voce, se di tre nei primi 3 byte con 0 nel byte meno significativo. Si osservi che una costante stringa è differente dal valore stringa generato mediante la direttiva "DC" (si veda nel seguito) in quanto quest' ultima non è limitata nella lunghezza. Esempi di stringhe e rappresentazioni:

'ABCD'	|41|42|43|44|
'ABC'	|41|42|43|00|
'AB'	|00|00|41|42|
'A'	|00|00|00|41|

Modello di programmazione del processore

D0-D7	Registri dati
A0-A7	Registri indirizzi
A7,SP	Stack pointer di sitema (si può indifferentemente utilizzare l'una o l'altra denominazione)
USP	Stack pointer di utente per programmi in stato supervisore
CCR	Registro Condition Code parte bassa (bit 0-7) del registro di stato SR
SR	Registro di stato modificabile, nella parte alta, solo in stato utente
PC	Registro prossima istruzione. tale simbolo è da usarsi per modi di indirizzamento relativi al PC.

2.1.1.6 6.6 Direttive di Assemblaggio

2.1.1.6.1.1 6.6.1 Controllo delle operazioni di assemblaggio

ORG Origine assoluta di un programma

```
<label>     ORG    <espressione>
esempio:    areap  ORG    $8000
```

Tale direttiva consente di assegnare al Location Counter un valore espresso su 32 bit. Il valore da assegnare (riferendosi ad un'area programmi in generale) deve essere pari, se così non fosse gli viene sommato +1. Se il campo label è presente, al simbolo associato all'ORG viene assegnato il nuovo valore del location counter.

END Fine di un programma

END <label>
esempio: END START

Tale direttiva serve a segnalare all'assemblatore la fine del programma sorgente. Tutto il testo presente in un file sorgente dopo tale direttiva verrà ignorato dall'assembler. Il campo etichetta che segue la direttiva indica all'assembler l'istruzione di inizio del programma (entry point o strating address).

2.1.1.6.1.2 6.6.2 Definizione simboli

2.1.1.6.1.3 EQU Assegna valori permanenti al simbolo del campo label

<label> EQU <espressione>
esempio: MAX EQU 100

Tale direttiva assegna un valore ad un simbolo. Il valore assegnato al simbolo è permanente e non può essere ridefinito mediante un'altra direttiva EQU. Il campo espressione non deve contenere riferimenti futuri ed il valore associato deve poter essere valutato all'atto in cui la direttiva è esaminata dall'assemblatore.

SET Assegna valori temporanei

<label> SET <espressione>
esempio: MAX SET 100

Tale direttiva è simile all'EQU, con la differenza che i simboli impiegati possono essere ridefiniti da altre direttive SET.

REG Definisce una lista di registri

<label> REG <reg list> [commento] ed in cui <reg list> ha la forma R1[-R2]{/R3[-R4]}
esempio: SALVA A2-A4/D1/D2-D5/

Tale direttiva consente di associare ad un simbolo una lista di nomi di registri del processore. la lista è libera, cioé i registri possono essere nominati in un ordine qualunque. Il formato impiegato per esprimere la lista è lo stesso che può essere adoperato con il codice operativo MOVEM di trasferimento multiplo di dati registro da e per la memoria.

2.1.1.6.1.4 6.6.3 Definizione dati-Allocazione memoria

DC Definisce una costante

<label> DC.<size> <item>,<item>, ...

esempio: ST2 DC.B 'ASSEMBLER'

La direttiva DC consente di definire una costante in memoria.

Al simbolo del campo <label>, se presente, viene assegnato l'indirizzo di memoria di inizio delle costanti allocate.

Il campo <size> può assumere i valori "B, W, L" ed indica all'assembler l'allineamente a livello byte, word o long word delle costanti da creare. Se si omette il size l'assembler per default assume allineamente word.

La direttiva può avere uno o più operandi separati da una virgola. Ciascun operando può essere una valore costante (di tipo decimale, esadecimale o appartenente al set ASCII) o un simbolo o un'espressione a cui l'assembler può assegnare un valore.

Operando di tipo stringa: è delimitato da singoli apici; a ciascun carattere della stringa, rappresentata su 7 bit, viene assegnato un byte di memoria in cui il bit ottavo è posto sempre a 0. La stringa non è ultimata da alcun carattere ma è considerata semplicemente una successione di caratteri non ammettendo il processore alcun tipo stringa.

La rappresentazione in memoria delle costanti generate dalla direttiva è caratterizzata essenzialmente da due fatti: le costanti sono rappresentate in modo contiguo ed a partire da qualunque indirizzo (pari o dispari) solo se è presente il size .B; la sincronizzazione a pari degli indirizzi viene effettuata ogni qualvolta si adopera il size .W o .L; eventuali indirizzi di memoria non usati dalle costanti per problemi di sincronizzazione (stante la scelta .W o .L fatta) sono poste a 0. Sono date di seguito vari esempi e relative le rappresentazioni in memoria di costanti generate con la direttiva DC. Le costanti generate con .B sono non rilocabili al contrario di quelle generate con .W e .L.

```
00000000  41 53 53 45 4D 42 4C 45 52              1  st1 dc.b   'ASSEMBLER'
0000000A  4153 5345 4D42 4C45 5200              2  st2 dc.w   'ASSEMBLER'
00000014  41535345 4D424C45 52000000            3  st3 dc.l   'ASSEMBLER'
00000020  41 53 53 45 4D 42 4C 45 52            4  st4 dc.b
   'A','S','S','E','M','B','L','E','R'
0000002A  4100 5300 5300 4500 4D00 ...          5  st5 dc.w
   'A','S','S','E','M','B','L','E','R'
0000003C  41000000 53000000 53000000 ...        6  st6 dc.l
   'A','S','S','E','M','B','L','E','R'

00000000  41 53 53 45 4D 42 4C 45 52         1  st1 dc.b   'ASSEMBLER'
0000000A  4153 5345 4D42 4C45 5200           2  st2 dc.w   'ASSEMBLER'
00000014  41535345 4D424C45 52000000         3  st3 dc.l   'ASSEMBLER'
00000020  41 53 53 45 4D 42 4C 45 52         4  st4 dc.b
   'A','S','S','E','M','B','L','E','R'
0000002A  4100 5300 5300 4500 4D00 ...       5  st5 dc.w
   'A','S','S','E','M','B','L','E','R'
0000003C  41000000 53000000 53000000 ...     6  st6 dc.l
   'A','S','S','E','M','B','L','E','R'
00000060                                     7
00000060  00 01 02 03 04 05 06 07 08         8  nm1 dc.b   0,1,2,3,4,5,6,7,8
0000006A  0000 0001 0002 0003 0004 ...       9  nm2 dc.w   0,1,2,3,4,5,6,7,8
0000007C  00000000 00000001 00000002 ...    10  nm3 dc.l   0,1,2,3,4,5,6,7,8
```


2.1.1.6.1.5 DCB Definisce un blocco di costanti

<label> DCB.<size> <lunghezza>,<valore>

La direttiva DCB serve a generare un blocco di valori in memoria di dimensione espressa nel campo <lunghezza> ed inizializzati al valore posto nel campo <valore>. Alla stringa posta nel campo <label>, se presente, viene assegnato l'indirizzo di inizio blocco. Relativamente a problemi di sincronizzazione degli indirizzi, vale quanto detto per le direttive precedenti.
Esempi:

```
000000E6                                19  cb1 dcb.b  10,'ABCD'
000000F0                                20  cb2 dcb.w  10,'ABCD'
00000104                                21  cb3 dcb.l  10,'ABCD'
0000012C                                22  cb4 dcb.b  10,$FF
```

DS Definisce un area di memoria

<label> DS.<size> <lunghezza>

La direttiva DS serve a riservare spazio di memoria in quantità indicata nel campo <lunghezza> non inizializzato ad alcun valore.
Al campo <label>, come al solito e se esistente, viene assegnato l'indirizzo iniziale del blocco.
Il campo <lunghezza> può essere un'espressione intera valutabile nel primo passo di compilazione (non deve fare, quindi, riferimento a simboli futuri).
Per quel che concerne la sincronizzazione degli indirizzi vale quanto già detto per la direttiva DC. L'effetto della direttiva DS è quello di incrementare il location counter della quantità di spazio di memoria da riservare. Va ricordato che la forma DS 0 può essere usata per sincronizzare il location counter ad un indirizzo pari.

Esempi:

```
000000A0                                12  ds1 ds.b   10
000000AA                                13  ds2 ds.w   10
000000BE                                14  ds3 ds.l   10
000000E6                                15  ds4 ds.b   0
000000E6                                16  ds5 ds.b   0
000000E6                                17  ds6 ds.b   0
```


2.1.2 6.4 Repertorio Codici operativi processore MC68000

Il repertorio dei codici operativi del processore Motorola MC68000 comprende 83 codici operativi suddivisi nelle seguenti nove classi

Trasferimento dati **Mo** [11 codici]
{EXG, LEA, LINK, MOVE, MOVEA, MOVEM, MOVEP, MOVEQ, PEA, SWAP, UNLK}

Arithmetica di interi **Ar** [18 codici]
{ADD, ADDA, ADDQ, ADDI, ADDX, CLR, DIVS, DIVU, EXT, MULS, MULU, NEG, NEGX, SUB, SUBA, SUBI, SUBQ, SUBX}

Test/Compare **TC** [6 codici]
{TAS, TST, CMP, CMPA, CMPM, CMPI}

Aritmetica di decimali in BCD **BD** [3 codici]
{ABCD, SBCD, NBCD}

Logiche **BO** [7 codici]
{AND, ANDI, OR, ORI, EOR, EORI, NOT}

Shift/Rotate **SR** [8 codici]
{ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR}

Controllo programma **CT** [9 codici]
{Bcc, DBcc, Scc, BRA, BSR, JMP, JSR, RTR, RTS}

Operazioni su bit **BI** [4 codici]
{BTST, BSET, BCLR, BCHG}

Operazioni di controllo sistema **MI** [17 codici]
{RESET, RTE, STOP, ORI to SR, MOVE USP, ANDI to SR, EORI to SR, MOVE EA to SR, TRAP, TRAPV, CHK, ANDI to CCR, EORI to CCR, MOVE EA to CCR, ORI to CCR, MOVE SR to EA, MOVE EA to USP}

2.1.3 6.5 Repertorio Codici operativi in ordine alfabetico

Per ogni codice operativo viene fornita una breve descrizione, la sintassi assembler, la codifica in linguaggio macchina e lo stato dei flag X N Z V C {0, 1, U per non definito, - per non interessato, * per modificato a 0 oppure 1 a seconda della logica}, dopo l'esecuzione della stessa. Per una completa descrizione del repertorio dei codici operativi si faccia riferimento al manuale del processore[MOTO-MC68000UMADI].

2.2 ABCD Add decimal with extend

```
1100 Rx 10000 R/M Ry
dest<=*dest10+*sorg10

X,N,Z,V,C
* U ? U ?
Z=Z·¬Rm·....·¬R0
C,X=1 se si genera riporto (decimale)

ABCD      Dy,Dx
ABCD      -(Ay),-(Ax)
size: B
```

2.2.1.1.1.1.1 esempi

```
Programma che somma due word in formato decimale mediante il codice ABCD. La
somma è eseguita mediante una subroutine WABCD che prende gli indirizzi delle
due word in A0 ed A1 e restituisce il risultato all'indirizzo di A1.


* Inizio codice
   ORG    $4000                      Allocazione del codice
VAR1      DS.W   1                          Riserva una word per var1
VAR2      DS.W   1                          Riserva una word per var2

* Inizio subroutine WABCD
* Questa subroutine effettua la somma decimale delle variabili agli indirizzi
* contenuti in A0, A1 e registra il risultato in A1. Un eventuale riporto è
segnalato
* dal flag X e da quello C nel registro di condizione ( CCR ).
* La somma è effettuata mediante il codice ABCD che somma due byte in formato
decimale
* tenendo conto dell'eventuale riporto generato nelle somme precedenti e
memorizzato
* nel flag X del registro di condizione.
WABCD     MOVE.W #0,CCR                     Inizializza registro di
condizioni
   ADDA.L #2,A0                     A0:=A0+2
   ADDA.L #2,A1                     A1:=A1+2
   ABCD   -(A0),-(A1)               (A1-1) := (A1-1) + (A0-1) + X

   ABCD   -(A0),-(A1)               (A1-1) := (A1-1) + (A0-1) + X
WABCDF    RTS

* Inizio programma
MAIN      MOVE.W #$3980,VAR1                Inizializzazione delle variabili
VAR1, VAR2
   MOVE.W #$7432,VAR2
   LEA.L  VAR1,A0                           Carica indirizzi completi delle
due
   LEA.L  VAR2,A1                            variabili in A0, A1.
   JSR    WABCD                     Effettua la somma VAR2=VAR2+VAR1
   END    MAIN                      Fine del codice
```

```
Questo programma esegue la somma decimale di due numeri memorizzati in 4 byte
ognuno a partire dalla locazione $8300 e restituisce il risultato in
$8300..$8304 sfruttando il modo "memory to memory" dell'istruzione ABCD

    ORG $8000
```

```
START MOVE.L     #EOP1,A0       in A0 indirizzo base 1° addendo
    MOVE.L       #EOP2,A1   in A1 indirizzo base 2° addendo
    MOVE.L       #COUNT,D1  in D1 il valore della variabile
*                               di conteggio del ciclo

LOOP  ABCD-(A1),-(A0)   Memory to Memory
    DBEQ  D1,LOOP
    MOVE.L       (A0),D2

*Dati a partire dalla locazione $8300
    ORG   $8300
OP1 DC.B  $85,$39,$43,$89
EOP1       EQU     *
OP2 DC.B  $27,$47,$21,$84
EOP2  EQU   *
COUNT EQU   EOP2-EOP1-1

    END START
```

Questo programma effettua la somma di due numeri, in formato BCD, posti in memoria su un numero di byte scelto dall'utente. $VER: bcdsum.asm 1.0 (27-11-96)*

```
* NOME
*     BCDSUM -- Somma BCD in multiprecisione.
*   FUNZIONE
*     Questo programma effettua la somma di due numeri, in formato BCD, posti
*     in memoria su un numero di byte scelto dall'utente.
*   NOTA
*     Il byte piu` alto e` il meno significativo del numero BCD.
*   INGRESSI
*     D0 - Numero di byte che compongono un addendo in formato BCD ( cifre
*           decimali / 2).
*     A0 - Contiene l'indirizzo del byte meno significativo del 1° addendo
BCD.
*     A1 - Contiene l'indirizzo del byte meno significativo del 2° addendo
BCD.
*   RISULTATO
*     (A1) - Contiene l'indirizzo del risultato della somma in formato BCD.
*     C, X - Indicatori di riporto( od overflow se non si intende
ulteriormente
*            estendere la lunghezza del risultato).
*  ESEMPIO
   ORG     $1000
BYTES          EQU      6
ADDENDO1       DS.B     BYTES        1° addendo di (2 * BYTES) cifre
decimali.
ADDENDO2       DS.B     BYTES        2° addendo di (2 * BYTES) cifre
decimali.

BCDSUM         SUBQ     #1,D0            Allineamento del contatore del
numero
*                                        di byte degli addendi.
               MOVE     #0,CCR                Azzera il valore dei codici
cond
*                                              del registro di stato del
proc.
               ADDA     D0,A0            Allinea i puntatori agli addendi sui
               ADDA     D0,A1                 loro byte meno significativi.
```

```
REPEAT          ABCD      -(A0),-(A1)       Esegue la somma binario decimale di
*                                                 una coppia di byte degli
addendi e
*                                                  punta alla coppia di byte succ.
                DBRA      D0,REPEAT         Ripete la somma fino all'esaurimento
*                                           del numero di coppie di byteda
sommare.             RTS                              Ritorna al programma
chiamante.

INIZIO          MOVEM.L   D0/A0-A1,-(SP)          Salva sullo stack il contenuto
*                                                       dei registri usati.
                MOVE      #BYTES,D0               Inizializza il contatore del
num *                                                   di byte degli addendi.
              LEA       ADDENDO1,A0             Punta al primo addendo.
                LEA       ADDENDO2,A1              Punta al secondo addendo.
                BSR       BCDSUM                   Esegue l'addizione.
                MOVEM     (SP)+,D0/A0-A1          Ripristina il contenuto prec.
*                                                   nei registri serviti per
oper.
                END       INIZIO
```

2.3 ABCD Addizione decimale in multiprecisione

Addiziona l'operando sorgente all'operando destinazione tenendo conto del bit di estensione, e memorizza il risultato nella locazione di destinazione. l'addizione è eseguita usando l'aritmetica decimale codificata in binario.


```
1100 Rx 10000 R/M Ry
dest<=*(dest)10 + *(sorg)10 + X

X,N,Z,V,C
* U ? U ?

C= Riporto decimale
Z= Z * ¬Rm * ... * ¬R0
X= C

ABCD      Dy,Dx
ABCD      -(Ay),-(Ax)
size: B
```

```
2.4    * $VER: bcdsum.asm 1.0 (27-11-96)
2.5    * NOME
2.6    *     BCDSUM -- Somma BCD in multiprecisione.
2.7    *   FUNZIONE
2.8    *     Questo programma effettua la somma di due numeri, in formato BCD, posti
2.9    *     in memoria su un numero di byte scelto dall'utente.
2.10   *   NOTA
2.11   *     Il byte piu` alto e` il meno significativo del numero BCD.
2.12   *   INGRESSI
2.13   *     D0 - Numero di byte che compongono un addendo in formato BCD ( cifre
2.14   *          decimali / 2).
2.15   *     A0 - Contiene l'indirizzo del byte meno significativo del 1° addendo BCD.
2.16   *     A1 - Contiene l'indirizzo del byte meno significativo del 2° addendo BCD.
2.17   *   RISULTATO
2.18   *     (A1) - Contiene l'indirizzo del risultato della somma in formato BCD.
2.19   *     C, X - Indicatori di riporto( od overflow se non si intende ulteriormente
2.20   *           estendere la lunghezza del risultato).
2.21   *  ESEMPIO
```

```
    ORG      $1000
BYTES          EQU      6
ADDENDO1       DS.B     BYTES        1° addendo di (2 * BYTES) cifre decimali.
ADDENDO2       DS.B     BYTES        2° addendo di (2 * BYTES) cifre decimali.

BCDSUM         SUBQ     #1,D0          Allineamento del contatore del numero
*                                      di byte degli addendi.
               MOVE     #0,CCR              Azzera il valore dei codici cond
*                                           del registro di stato del proc.
               ADDA     D0,A0          Allinea i puntatori agli addendi sui
               ADDA     D0,A1               loro byte meno significativi.
REPEAT         ABCD     -(A0),-(A1)    Esegue la somma binario decimale di   *
                                         una coppia di byte degli addendi e
*                                       punta alla coppia di byte succ.
               DBRA     D0,REPEAT      Ripete la somma fino all'esaurimento
*                                      del numero di coppie di byteda sommare.
       RTS                           Ritorna al programma chiamante.

INIZIO         MOVEM.L  D0/A0-A1,-(SP)       Salva sullo stack il contenuto   *
                                                 dei registri usati.
               MOVE     #BYTES,D0            Inizializza il contatore del num *
di byte degli addendi.                       LEA      ADDENDO1,A0             Punta al
primo addendo.
               LEA      ADDENDO2,A1            Punta al secondo addendo.
               BSR      BCDSUM                 Esegue l'addizione.
               MOVEM    (SP)+,D0/A0-A1       Ripristina il contenuto prec.
*                                             nei registri serviti per oper.
               END      INIZIO
```

2.22 ADD Add binary

```
1101 R mode ea
dest<=*dest+*sorg

X,N,Z,V,C
* * * ? ?
V=Sm·Dm·¬Rm+¬Sm·¬Dm·Rm
C=Sm·Dm+¬Rm·Dm+Sm·¬Rm

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L
```

2.22.1.1.1.1.1 esempio

```
*Il programma calcola il prodotto di due numeri con l'algoritmo delle
addizioni successive allo scopo di testare il codice operativo ADD
inizializzando il prodotto a zero e aggiungendo MPY volte MCND a se'
stesso.L'istruzione ADD.B usata effettua la somma su due operandi su 8 bit
usando l'addizione binaria con carry iniziale nullo e memorizza la somma su 8
bit nell'op.dest. e setta i conditions bit in base al valore degli operandi.
ADD e' valido sia per numeri signed two's complement che per numeri unsigned.
Per i signed,N e V indicano lo stato del risultato,mentre per gli unsigned, C
indica un riporto che si estende oltre la     *posizione
dell'MSB.L'istruzione ADD setta consistentemente tutti questi bit di
condizione.Per quanto riguarda il campo <ea> (effective address),si ha che se
nella locazione specificata vi e' un operando sorgente,allora tutti i modi di
indirizzamento sono concessi.Se il size e'il byte,allora il modo address
register direct non e' permesso.
*
*Descrizione dello SR durante l'esecuzione del seguente listato:
*-------------------------------------------------------------------------
-------------------------------------------------------------
*  moltiplicando       12         20           30         40         -35
-127               d1(inizialmente contiene MPY)
*-------------------------------------------------------------------------
-------------------------------------------------------------
* add.b  D1,D0      00000    00000    00000   00000   01000   01000
* add.b   #-1,d1     10001    10001    10001   10001   10001   10001
6
*-------------------------------------------------------------------------
-----
*                   00000     00000    00000   00000   11001   10011
*                   10001     10001    10001   10001   10001   10001
5
*-------------------------------------------------------------------------
-----
*                   00000     00000     00000   00000  11001    01000
*                   10001     10001     10001   10001  10001    10001
4
*                         ---------------------------------------------
-----------------------------------------------------------------
*                   00000      00000    00000    01010   10011   10011
*                   10001      10001    10001    10001   10001   10001
3
*                         ---------------------------------------------
-----------------------------------------------------------------
*                            00000      00000     01010    01000   10001
01000
```

```
*                              10001      10001     10001    10001   10001
10001        2
*                             ------------------------------------------------
-----------------------------------------------------------------
*                               00000     00000      01000    01000   10001
10011
*                               10001     10001      10001    10001   10001
10001        1
*                             ------------------------------------------------
-------------------------------------------------------
*                               00000     01010      01000    10001    10001
01000
*                               10101     10101      10101    10101    10101
10101       0
*                              -----------------------------------------------
-------------------------------------------------------
Non sono stati riscontrati errori nel posizionamento dei flag del registro
SR.

org $8200
main
              lea          mcnd,a0
              bsr          mult
done      move.b   d0,prod
              cmp.w     #mcnde,a0
              bne         mult
              stop        #$ff00

mult
              move.b   (a0)+,d2
              move.b    mpy,d1
              clr.w       d0
loop       add.b      d2,d0
              add.b      #-1,d1
              bne         loop
              beq         done
              rts
              org          $9000
mpy       dc.b         7
moltiplicatore fisso
prod       ds.b         1
mcnd     dc.b         12,20,30,40,-35,-127            moltiplicandi
utilizzati
mcnde   equ          mcnd+6

              end main
```


2.23 ADD Add binary

1101 R mode ea
*dest<=*dest+*sorg

X,N,Z,V,C
* * * ? ?
$V=S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L

```
esempio
*Il seguente listato calcola il prodotto di due numeri con l'algoritmo delle
addizioni uccessive allo scopo di testare il
*codice operativo ADD inizializzando il prodotto a zero e aggiungendo MPY
volte MCND a se' stesso.L'istruzione
*ADD.B usata effettua la somma su due operandi su 8 bit usando l'addizione
binaria con carry iniziale nullo
*e memorizza la somma su 8 bit nell'op.dest. e setta i conditions bit in base
al valore degli operandi.
*ADD e' valido sia per numeri signed two's complement che per numeri
unsigned.
*Per i signed,N e V indicano lo stato del risultato,mentre per gli unsigned,C
indica un riporto che si estende oltre la     *posizione
dell'MSB.L'istruzione ADD setta consistentemente tutti questi bit di
condizione.Per quanto riguarda
*il campo <ea> (effective address),si ha che se nella locazione specificata
vi e' un operando sorgente,allora
*tutti i modi di indirizzamento sono concessi.Se il size e'il byte,allora il
modo address register direct non e' permesso.
************************************************************************
**********************
*
*Descrizione dello SR durante l'esecuzione del seguente listato:
*-----------------------------------------------------------------------
-------------------------------------------------------------
*  moltiplicando       12         20           30         40         -35
-127               d1(inizialmente contiene MPY)
*-----------------------------------------------------------------------
-------------------------------------------------------------
* add.b  D1,D0      00000    00000    00000   00000   01000   01000
* add.b   #-1,d1     10001    10001    10001   10001   10001   10001
6
*                         ---------------------------------------------
-------------------------------------------------------------
*                            00000     00000    00000   00000   11001   10011
*                            10001     10001    10001   10001   10001   10001
5
*                         ---------------------------------------------
-------------------------------------------------------------
*                            00000     00000     00000   00000  11001
01000
*                            10001     10001     10001   10001  10001
10001          4
*                         ---------------------------------------------
-------------------------------------------------------------
*                            00000      00000    00000    01010   10011
10011
*                            10001      10001    10001    10001   10001
10001        3
*                         ---------------------------------------------
-------------------------------------------------------------
*                            00000      00000     01010    01000   10001
01000
```

2.24 ADDA Add address

```
1101 R mode ea
dest<=*dest+*sorg

X,N,Z,V,C
- - - - -

ADDA      <ea>,An
size: W,L


esempio
```

2.25 ADDI Add immediate

00000110 size ea + 1,2 ext word
dest<=*dest+imm

X,N,Z,V,C
* * * ? ?
$V=S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$

ADDI #<imm>,<ea>
size: B,W,L

esempio

2.26 ADDI Add immediate

00000110 size ea + 1,2 ext word
dest ← *dest+imm

X,N,Z,V,C
* * * ? ?
$V=S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$

ADDI #<imm>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- Immediate
- Relative
- Relative indexed

esempio :

```
* Questo segmento di programma estrae una colonna da una matrice di
long word e * la trasferisce in una certa area di memoria.
* D0 contiene l'indice della colonna e A1 contiene l'indirizzo di
partenza       * dell'area di memoria in cui deve essere trasferita
la colonna.
*
   org     $8000
START    subq.l   #1,D0          individua l'indirizzo di partenza
   mulu.w   #4,D0          del primo elemento della colonna
   addi.l  #TABINI,D0 selezionata;
   moveq.l       #3,D1          inizializza il contatore;
LOOP     move.l  D0,A0          consente di utilizzare
l'indirizzamento indiretto;
   move.l  (A0),(A1)+ trasferisce l'elemento e aggiorna il puntatore;
   addi.l  #16,D0    seleziona il successivo elemento della colonna;
   dbf     D1,LOOP
*
TABINI   org     $8020          inizializza l'area della tabella
*
RIGA1    dc.l    $004531A0,$0103F34A,$0005E781,$120056BC
RIGA2    dc.l    $17C3B001,$A9D361C0,$01050641,$F20D501C
```

```
RIGA3    dc.l    $1045F380,$D010520A,$0005E000,$AB00D12C
RIGA4    dc.l    $030581A0,$0AB3F51A,$0503D081,$1263A0BC
    END      START
```

2.27

2.28 Note :

- Il modo di indirizzamento ‘Address-direct-register’non è permesso, contrariamente a quanto è scritto sia sul manuale (‘Only data alterable addressing modes are allowed’) sia sul Wakerly (‘a_dst’).

2.29 ADDQ Add quick

```
0101 data 0 size ea
dest<=*dest+data

X,N,Z,V,C
* * * ? ?
V=Sm·Dm·¬Rm+¬Sm·¬Dm·Rm
C=Sm·Dm+¬Rm·Dm+Sm·¬Rm

ADDQ      #data,<ea>
size: B,W,L


esempio
```

2.30 ADDQ Add quick

```
0101 data 0 size ea
dest<=*dest+data

X N Z V C
* * * ? ?
V=Sm·Dm·¬Rm+¬Sm·¬Dm·Rm
C=Sm·Dm+¬Rm·Dm+Sm·¬Rm

ADDQ  #data,<ea>
size: B,W,L

ESEMPIO

Questa subroutine permette il calcolo dei bit alti di un dato passato
in D0,restituisce il risultato in d1.

ORG     $8000   indirizzo di partenza

ESADDQ MOVE.L #32,D1 Inizializza a 32 d1
       MOVE.L #32,D2 Inizializza il contatore
LAB1   ROL     #1,D0 Shifta a sinistra D0
       BCS      LAB2 Controlla il primo bit di D0
                SUBQ      #1,D1
LAB2   SUBQ    #1,D2
       BNE      LAB1
       RTS      Ritorno al chiamante

MAIN
       MOVE.L #$10000003,D0  Metti in D0 il dato
       JSR    ESADDQ Salto a sottoprogramma
       END    MAIN
```


ADDQ Somma

				Data			Effective Address

ADDQ #<data>,<ea>

Somma al valore contenuto in dst un valore immediato compreso tra 1 ed 8. La destinazione può essere un indirizzo effettivo alterabile. Sono permessi i size BWL tranne che per gli An dove non è permesso utilizzare il size Byte.

X	N	Z	V	C
*	*	*	?	?

X è settato allo stesso modo di C
N è settato se il risultato è negativo
Z è settato se il risultato è zero
$V = S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C = S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$

Sm bit più significativo del sorgente
Dm bit più significativo della destinazione
Rm bit più significativo del risultato

L'istruzione ADDQ torna utile laddove serve incrementare un puntatore ad una word (2 unità) o ad una long word (4 unità).

Differenze con ADDI: è più corta ed è ad essa preferita quando si ha a che fare con parole in doppia precisione ; può usare un An come destinazione ed, in tal caso, si comporta come un ADDA con un dato immediato (sono consentite come dimensioni, in questo caso, soltanto W e L e i bit di stato restano intatti).

2.31 ADDX Add extended

1101 Rx 1 size 00 R/M Ry
dest<=*dest+*sorg+X

X,N,Z,V,C
$V=S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$
$Z=Z \cdot \neg R_m \cdot \ldots \cdot \neg R_0$

```
ADDX      Dy,Dx
ADDX      -(Ay),-(Ax)
size: B,W,L
```

esempio

2.32 ADDX Add extended

1101 Rx 1 size 00 R/M Ry
dest<=*dest+*sorg+X

X,N,Z,V,C
* * ? ? ?
$V=S_m \cdot D_m \cdot \neg R_m + \neg S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot D_m + \neg R_m \cdot D_m + S_m \cdot \neg R_m$
$Z=Z \cdot \neg R_m \cdot \ldots \cdot \neg R_0$

```
ADDX      Dy,Dx
ADDX      -(Ay),-(Ax)
size: B,W,L
```


esempio

```
*Esempio 2 includente operazioni in multipla precisione sfruttando addx.l;
*D5 contiene l'high order longword somma di D1 e D3,mentre D6 contiene
*la low order longword somma di D0 e di D2.
*Al fine di testare anche il modo di indir. data reg. to data reg.,la somma
di D0
*e di D2 è stata infine salvata in D7.
*TST1=D1-D0=11111100001010101111010100101111-                    *
11111100001110101111011110001001
*TST2=D3-D2=01111010001010101010001110101010-                    *
01101010101001001011101001100111

            org        $8000
start
            move.l    #$fc2af52f,d0 load registers D0-D1 and *
D2-D3 to do two longwords
            move.l    #$fc3af789,d1
            move.l    #$6aa4ba67,d2
            move.l    #$7a2aa3aa,d3
            move.l     d0,TST1L
            move.l     d1,TST1H
            move.l     d2,TST2L
            move.l     d3,TST2H
            lea.l      TST1+8,a0  addressing for auto-decrement   *
access
```

```
            lea.l      TST2+8,a1  addressing for auto-decrement   *
access
            move.w    #0,ccr                          clear X
              *                                            SR: XNZVC
                  addx.l      -(a1),-(a0)                   10001 *   adds
TST1 to TST2 one longword at a time
                  move.l     (a0),d5                        10000 *   saves
result of D1+D3 in D5(it isn't change X)
                  addx.l      -(a1),-(a0)                   10001 *   adds
with X-Flag getted by the first addx
                  move.l      (a0),d6                       10000 *  saves
result of D0+D2 in D6(X is unchanged)
                  move.l       d6,d7                        10000 *  and now
D7 contains D0+D2
                  addx.l       d5,d6                        01010 *   this
instruction has only the task to test addx
*   for data register to data register.
           stop             #$ff00
           org              $8500
TST1          ds.l              2
TST2          ds.l              2
TST1L         equ            TST1
TST1H       equ              TST1+4
TST2L        equ              TST2
TST2H       equ               TST2+4
                  end           start
```

2.33 AND And logical

```
1100 R mode ea
dest<=*dest & *sorg

X,N,Z,V,C
- * * 0 0

AND<ea>,Dn
AND Dn,<ea>
size: B,W,L


esempio
```

2.34 AND And binary

```
1100 R mode ea
dest=dest and sorg

X,N,Z,V,C
- * * 0 0
N=1 se il bit più significativo di dest è 1
Z=1 se dest=0


AND<ea>,Dn
AND Dn,<ea>
size: B,W,L


esempio:
Il seguente frammento di programma conta i bit alti in un byte
        org         $8000
mask    dc.b        1,2,4,8,16,32,64,128    ognuno di
*                    questi byte ha un solo bit 1
pippo   ds.b        1                       byte da analizzare
init    move.b      #%10011001,pippo            fornisce un
*                           valore al byte pippo

cont    move.b      #8,d2                   inizializza il
*                           contatore del ciclo
        movea.l     #mask,a0
loop    move        pippo,d0        d0 verrà modificato
*                                           dalla and
        add.b       #-1,d2          decrementa il
*                                           contatore
        and.b       0(a0,d2),d0
        beq         zero
        add.b       #1,d1                   il bit di posizione
*                                           d2 è 1
zero    cmp         #0,d2                   controlla la fine del
*                                           ciclo
        bne         loop
        end         init                    d1 contiene il numero
*                            di bit alti del byte pippo
```

2.35 AND And logical

```
1100 R mode ea
dest<=*dest & *sorg
```

```
X,N,Z,V,C
- * * 0 0
N=1 se MSB del risultato=1
Z=1 se il risultato=0

AND<ea>,Dn
AND Dn,<ea>
size: B,W,L

se <ea> è sorg. sono permessi solo modi
di indirizamento “data”
se <ea> è dest. sono permessi solo modi di ind. “alterable memory”  (cfr.
MC68000,Table B-1)


esempio
* Questo programma conta il numero di bit alti in un byte * (NUM); allo scopo
di testare il bit i-esimo di NUM     * esegue una AND tra NUM ed un byte-
maschera in cui tutti * i bit sono settati bassi tranne l’i-esimo che è alto.
* Se il risultato produce Z=0 (AND <> 0) vuol dire che   * il corrispondente
bit di NUM è alto, per cui si        * incrementa di 1 un contatore. I byte-
maschera vengono  * memorizzati tutti in uno stesso registro semplicemente  *
shiftando il suo contenuto a sinistra in ogni ciclo di * confronto.

    ORG $8000
NUM DC.B  $F1       numero da processare

BEGIN CLR.B D3
    MOVE.B #1,D1   in D1 la prima maschera
    MOVE.B (NUM),D0
    MOVE.B D0,D2

LOOP  MOVE.B D2,D0
    AND D1,D0
    BEQ NOADD     se Z=0 non inc. il contatore
    ADD #1,D3     se Z=1 inc. il contatore
NOADD LSL.B #1,D1    prepara prossima maschera
    TST.B D1      controlla se il byte è finito
    BNE LOOP       altrimenti ripete il ciclo

    END BEGIN

* in D3 è contenuto il numero di bit alti del byte NUM
```


2.36 AND And logico

And logico tra <destinazione> e <sorgente>, il risultato va in <destinazione>; uno dei due operandi deve essere un registro dati.

```
1100 R mode ea
dest<=*dest & *sorg

X,N,Z,V,C
- * * 0 0

N=1 se MSB(*dest)=1
Z=1 se *dest=0
```

AND<ea>,Dn
AND Dn,<ea>
size: B,W,L

```
*Esempio di AND

   org $1000

*conta i bit alti di una word.
*In d0 viene passato il dato per valore e sempre in d0 viene
*restituito il numero di bit alti.

contb    move.w #1,d1 maschera per d0
   move.b #16,d2     contatore del ciclo
   move.b #0,d3 num. di bit alti
ciclo    move.w d0,d4 in d4 ho una copia di d0
   and.w d1,d4
   beq noinc         il bit d2-esimo è 0
   add.b #1,d3
noinc    asl.w #1,d1       cambio maschera
   sub.b #1,d2
   bne ciclo
   move.w d3,d0
   rts

start    andi.w #$00ff,sr
   move.w #$3f33,d0  il risultato deve essere 10
   jsr contb
   end start
```

2.37 ANDI And immediate

```
00000010 size ea + 1,2 ext word
dest<=*dest & imm

X,N,Z,V,C
- * * 0 0
N=1 se MSB(*dest)=1
Z=1 se *dest=0

ANDI      #<data>,<ea>
size: B,W,L


esempio
```

2.38 ANDI And immediate

```
00000010 size ea + 1,2 ext word
dest ← *dest & imm

X,N,Z,V,C
- * * 0 0
N=1 se MSB(*dest)=1
Z=1 se *dest=0

ANDI      #<data>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- Immediate
- Relative
- Relative indexed


esempio :
```



```
* Questo segmento di programma costruisce una Longword con blocchi di
bit conte- * nuti in BYTE, WORD e LWORD. Tali blocchi sono estratti a
mezzo dell'istruzione * ANDI e sono assemblati in D0.
*
    ORG  $8000
*
    MOVE.B         BYTE,D0         Sposta BYTE in D0
    ANDI.B         #$3C,D0               Estrae il blocco di bit da 2
a 5
    MOVE.W         WORD,D1         Sposta WORD in D1
    ANDI.W         #$7C00,D1       Estrae il blocco di bit da 10 a 14
    OR.W    D1,D0           Impacchetta i 2 blocchi
    MOVE.L   LWORD,D2       Sposta LWORD in D2
    ANDI.L         #$F0700000,D2  Estrae i blocchi da 20 a 22 e da 28
a 31
    OR.L     D2,D0          Impacchetta i 4 blocchi
*
BYTE       DC.B    %01101001
WORD       DC.W     %1010010110111000
LWORD      DC.L    %11101100100100011111101001000111
```

2.39 Note :

- Il modo di indirizzamento ‘Address-direct-register’non è permesso, in accordo con Wakerly (‘d_dst’) ma contrariamente a quanto è scritto sul manuale (‘Only data alterable or the status register addressing modes are allowed’) .

2.40 ASL Arithmetic shift left

```
1110 count|Reg dr size i/r 00 R
dest<=dest<<count

X,N,Z,V,C
* * * * *
N=1 se dest=1
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ASL Dx,Dy
ASL #<data>,Dy
ASL <ea>
size: B,W,L


esempio
```

2.41 ASR Arithmetic shift rigth

```
1110 count|Reg dr size i/r 00 R
dest<=dest<<count

X,N,Z,V,C
* * * * *
N=1 se dest=1    RIVEDERE DA QUI IN POI FLAG !!!!!!
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ASR Dx,Dy
ASR #<data>,Dy
ASR <ea>
size: B,W,L


esempio
```

2.42 ASR Arithmetic shift rigth

```
1110 count|Reg 0 size i/r 00 R (Register Shifts)
dest<=dest>>count

1110 000 011 ea                ( Memory Shifts )
dest<=dest>>1

X,N,Z,V,C
* * * 0 ?
C= D count-1

X,N,Z,V,C ( count == 0 )
- * * 0 0

ASR Dx,Dy
ASR #<data>,Dy
ASR <ea>
size: B,W,L


esempio
*********************************************************************
*Il seguente programma testa il codice operativo ASR eseguendo il controllo
*della divisibilità per due di un numero incluso in tst.
*Gli shift aritmetici trattano i loro operandi come signed two's complement;
*lo shift aritmetico ASR equivale alla divisione per due.
*Nell'operazione ASR,l'overflow aritmetico è impossibile,così il bit di flag
V
*è sempre zero.
*Nel seguente listato,la divisione per due di un numero dispari eseguita con
*ASR.W genera X:=C con C:=1 così come dovrebbe;
*ASR.W per numeri pari genera X:=C con C:=0  (ok);
*ASR.W per numeri negativi pari genera X:=C con C:=1;N:=1;
*ASR.W per numeri negativi dispari genera X:=C con C:=1;N:=1.
*********************************************************************


            org            $8000
main
```

```
            lea               test,a0
            bsr               divideby2
done1       move.b        #0,d3                     pone 0 in d3 se il
numero non è divisibile per 2
            bsr               newtst
done2       move.b        #1,d3                     pone 1 in d3 se il
numero è divisibile per 2
newtst      cmp.l         #testend,a0
            bne             divideby2
            stop            #$ff00
divideby2
                move.w         (a0)+,d1
                move.w         d1,d2                          salva numero
attuale in d2
                asr.w             #1,d1                          effettua
la divisione per 2
                mulu.w          #2,d1
                cmp.w           d1,d2
                bgt                done1
                beq              done2
                rts
                org               $8700
test          dc.w              4,7,20,21,120,125,1000,2307,-124,-1005
numeri per il test
testend     equ               test+20
                end               main
```

2.43 Bcc Branch conditionally

```
0110 Cond 8bit_disp+[16bit_disp]
if (cc) PC+disp=>PC

X,N,Z,V,C
- - - - -
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

Per i cc si veda la tabella xx
Bcc<label>
size: B,W
```

esempio

B_{CC} Diramazione su condizione

				Condition	8-bit Displacement
16-bit Displacement if 8-bit Displacement = 0					

Bcc <label>

operazione effettuata : if cc then PC + spiazzamento -> PC

Consentiti i size di byte e word.

Non sortisce alcun effetto sul CCR (bit di flag).

Condizioni implementate :

	Test su			Test su	
	0100 -	¬C		0011 -	C | Z
	0101 -	C		1101 -	(N&
	0111 -	Z		1011 -	N
	1100 -	(N&V) | (¬N&¬V)		0110 - Not	¬Z
	1110 -	(N&V&¬Z)|(¬N&		1010 -	¬N
	0010 - High	¬C&¬Z		1000 -	¬V
	1111 - Less	Z|(N&¬V)|(¬N&V		1001 -	V

Quando si devono confrontare numeri dotati di segno si usano BLT e BGE.

BLT si comporta come BMI finché non si ha Overflow, altrimenti è equivalente a BPL.

2.44 Bcc Branch conditionally

```
0101 Condition 11001 8-bit Displacement
```

16-bit Displacement if 8-bit Displacement=0

```
if cc then Pc+d=>Pc
Bcc <label>

X N Z V C
- - - - -

Bcc <label>

Size (Byte,Word)
```

ESEMPIO

Questa subroutine restituisce in D0 il valore assoluto di un byte passato in D0 , setta il Flag v =1 se D0 = -128.

```
ORG $8000   indirizzo di partenza

ESBCC
      CLR    D1     Inizializza a zero D1
      CMP.B  #0,D0  Controlla se D0 < 0
      BGE    LAB1
      SUB.B  D0,D1  Calcola D1 = 0 - D0
      MOVE.L D1,D0
      CMP.B  #-128,D0 Controlla se D0 = - 128
           BNE    LAB1
      MOVE.W #$2,CCR  Attiva il flag v
LAB1
      RTS  Ritorno al chiamante

MAIN
      MOVE.L #-128,D0 Metti in D0 il dato
      JSR    ESBCC    Salto a sottoprogramma
      END    MAIN
```

2.45 BCHG Test a bit and change

```
1101 R mode ea
dest=dest+sorg

- - ? - -
Z=¬Dn

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.46 BCHG Test a bit and change

```
0000 R 101 ea                          (bit number dynamic)
0000100001 ea + 1 ext word             (bit number static)
Z ← ~(<bit number> of dest)
(<bit number> of dest)← ~(<bit number> of dest)

X,N,Z,V,C
- - ? - -
Z = ~Dn

BCHG      Dn,<ea>
BCHG      #<data>,<ea>
size: B,L

Modi di indirizzamento non permessi :
- diretto con registro dati
- diretto con registro indirizzo
- relativo
- relativo indicizzato
- immediato


esempio :

* L'esempio permette di complementare bit per bit un byte ottenendone la
* rappresentazione per complementi diminuiti .
* In D2 è riportato il numero di 1 presenti nel byte complementato .
*
          org    $8000
BYTE      dc.b   %00110100     byte da complementare
CNTdc.b   7             contatore
*
INIZIO    org    $8020
   clr.l  D0
   clr.l  D1
   clr.l  D2
   move.b CNT,D1        inizializza il contatore
   move.b BYTE,D0
*
LOOP      bchg.l D1,D0         complementa a partire dal bit 7 fino al bit 0
   beq    NUM1          salta NUM1 se è avvenuta la transizione 0-1
   dbf    D1,LOOP
   bra.s  FINE
```

```
NUM1      addi.b #1,D2         aggiorna il numero di 1 del complemento
diminuito
*   cmpi.b #-1,D1
    dbf    D1,LOOP
*
FINE      nop
*
    END    INIZIO
```

2.47 BCLR Test a bit and clear

```
1101 R mode ea
dest=dest+sorg

- - ? - -
Z=¬D_n

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.48 BCLR Test a bit and clear

```
0000 R 110 ea                          (bit number dynamic)
0000100010 ea + 1 ext word             (bit number static)
Z ← ~(<bit number> of dest)
<bit number> of dest ← 0

X,N,Z,V,C
- - ? - -
Z = ~Dn

BCLR      Dn,<ea>
BCLR      #<data>,<ea>
size: B,L

Modi di indirizzamento non permessi :
- diretto con registro dati
- diretto con registro indirizzo
- relativo
- relativo indicizzato
- immediato


esempio :

* L'esempio permette di eseguire una operazione OPER1 solo se si riscontra il
* passaggio dal valore 1 al valore 0 del bit in posizione 4 del byte CONTROL
.
* Si noti l'analogia con l'esempio del codice BSET .
*
          org    $8000
CONTROL           dc.b   %10110101     byte di controllo
CNT       dc.b   10            contatore
*
INIZIO            org    $8020
          clr.l  D0
          clr.l  D1
          move.b #10,D3        inizializza il contatore
          move.b CONTROL,D7
*
LOOP              bclr.l #4,D7         se il bit 4 passa da 1 a 0 il flag Z
rimane basso
          bne    OPER1         salta ad OPER1 se è avvenuta la transizione
          addi.b #1,D1         altrimenti esegue altre operazioni
          cmpi.b #4,D1
```

```
          bne.s  SALTA
          ori.b  #16,D7        riporta a 1 il bit 4 del byte di controllo
SALTA             dbf    D3,LOOP              decrementa il contatore
          bra    FINE
OPER1             addi.b #2,D0
          dbf    D3,LOOP
*
FINE              nop
*
          END    INIZIO
```

2.49 BRA Branch always

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L
```

esempio

BRA Salta sempre

								8-bit Displacement
16-bit Displacement if 8-bit Displacement = 0								

BRA <label>

operazione effettuata : PC + spiazzamento -> PC

Consentiti i size di byte e word.

Non sortisce alcun effetto sul CCR (bit di flag).

A differenza dell'istruzione JMP che effettua un salto ad un indirizzo assoluto, la BRA salta ad un indirizzo relativo alla posizione corrente del PC.

2.50 BSET Test a bit and set

```
1101 R mode ea
dest=dest+sorg

- - ? - -
Z=¬Dn

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.51 BSET Test a bit and set

```
0000 R 111 ea                          (bit number dynamic)
0000100011 ea + 1 ext word             (bit number static)
Z ← ~(<bit number> of dest)
<bit number> of dest ← 1

X,N,Z,V,C
- - ? - -
Z = ~Dn

BSET      Dn,<ea>
BSET      #<data>,<ea>
size: B,L

Modi di indirizzamento non permessi :
- diretto con registro dati
- diretto con registro indirizzo
- relativo
- relativo indicizzato
- immediato


esempio :

* L'esempio permette di eseguire una operazione OPER1 solo se si riscontra il
* passaggio dal valore 0 al valore 1 del bit in posizione 4 del byte CONTROL
.
* Si noti l'analogia col l'esempio del codice BCLR .
*
          org    $8000
CONTROL           dc.b   %10100101     byte di controllo
CNT       dc.b   10            contatore
*
INIZIO            org    $8020
          clr.l  D0
          clr.l  D1
          move.b #10,D3        inizializza il contatore
          move.b CONTROL,D7
*
LOOP              bset.l #4,D7         se il bit 4 passa da 0 a 1 si alza il
flag Z
          beq    OPER1         salta ad OPER1 se è avvenuta la transizione
          addi.b #1,D1         altrimenti esegue altre operazioni
          cmpi.b #4,D1
```

```
          bne.s  SALTA
          andi.b #239,D7              riporta a 0 il bit 4 del byte di
controllo
SALTA             dbf    D3,LOOP              decrementa il contatore
          bra    FINE
OPER1             addi.b #2,D0
          dbf    D3,LOOP
*
FINE              nop
*
          END    INIZIO
```

2.52 BSR Branch to subroutine

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L
```

esempio

BSR Salto a sottoprogramma

								8-bit Displacement
16-bit Displacement if 8-bit Displacement = 0								

BSR <label>

operazione effettuata : PC->SP@- ; PC + spiazzamento -> PC

Consentiti i size di byte e word.

Non sortisce alcun effetto sul CCR (bit di flag).

BSR salva il PC all'indirizzo antecedente a quello contenuto in A7 (indirizzo di ritorno).
Modifica il PC in maniera tale da saltare all'etichetta della subroutine. L'indirizzo di ritorno viene salvato in cima alla pila puntata da SP.

In virtù dell'utilizzo dello Stack Pointer la BSR può essere richiamata più volte anche all'interno di porzioni di codice a cui si è giunti da un'altra BSR. Ad ogni BSR il valore di SP è decrementato di 4 unità e il nuovo indirizzo di ritorno viene memorizzato nella nuova posizione dello stack.

2.53 BTST Test a bit

```
1101 R mode ea
dest=dest+sorg

- - ? - -
Z=¬Dn

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.54 BTST Test a bit

```
0000 R 100 ea                          (bit number dynamic)
0000100000 ea + 1 ext word             (bit number static)
Z ← ~(<bit number> of dest)

X,N,Z,V,C
- - ? - -
Z = ~Dn

BTST      Dn,<ea>
BTST      #<data>,<ea>
size: B,L

Modi di indirizzamento non permessi :
- diretto con registro indirizzo
- immediato
- BTST.B non può avere come dest un registro dati


esempio :

* Tale programma consente di contare il numero di bit '1' in un byte e di
* valutarne la parità .
*
   org    $8200
BEGIN     move.b BYTE1,D0
   clr.l  D2                  Azzera il contatore dei bit '1'
        move.b   #7,D1                Inizializza bitnum a 7
LOOP      btst.l D1,D0                Test di D0[D1] dinamico
   beq.s  NEXT          Salta se Z=1
   addq.b #1,D2         Incrementa il contatore se Z=0
NEXT      dbf    D1,LOOP
   btst.l #0,D2         Setta Z se il numero di bit è pari (test statico)

FINE      nop
   org    $8500
byte1     dc.b   214           in binario    11010110
   end    BEGIN
```

Note sulle istruzioni di bit manipulation

Innanzitutto in teoria le istruzioni di manipolazione di bit possono operare solo su operandi di size byte e long word, e non su word come indicato sul manuale.

```
          org    $8000
CONTROL           dc.l   $AEBA8423
INDICE            dc.l   70
INIZIO            org    $8020
          move.l CONTROL,D7
          move.l INDICE,D0
LOOP1             btst.l D0,D7
          dbf    D0,LOOP1
          move.l CONTROL,D7
          move.l INDICE,D0
LOOP2             bset.l D0,D7
          dbf    D0,LOOP2
          move.l CONTROL,D7
          move.l INDICE,D0
LOOP3             bclr.l D0,D7
          dbf    D0,LOOP3
          move.l CONTROL,D7
          move.l INDICE,D0
LOOP4             bchg.l D0,D7
          dbf    D0,LOOP4
          *
FINE              nop
*
          END    INIZIO
```

Provando a simulare questo programma con Asim, osservando i registri influenzati e l'SR, è possibile evidenziare alcuni malfunzionamenti delle istruzioni di bit manipulation se il size è long word :

- BTST opera correttamente solo sugli 8 bit meno significativi di una long word, settando il flag Z=1 per tutti gli altri ;
- BSET,BCLR operano correttamente solo sui 16 bit meno significativi, lasciando inalterati quelli più significativi. In particolare se si manipola il bit in posizione 15 si nota che :

- BSET setta tutti i 16 bit più significativi ;
- BCLR pone a 0 tutti i 16 bit più significativi ;

- BCHG opera correttamente solo sugli 8 bit meno significativi, lasciando inalterati quelli più significativi. In particolare se si manipola il bit in posizione 15 si nota che vengono settati i 16 bit più significativi, ed inoltre quelli dalla posizione 15 alla 8 sono trattati come se in origine fossero tutti 0.

2.55 CHK Check register against bounds

```
1101 R mode ea
dest=dest+sorg

- * U U U

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.56 CHK Check Register against Bounds

```
0100 Reg 110 EA
If Dn<0 or D>(<ea>) then TRAP

X N Z V C
- * U U U
N - Settato se Dn<0, Azzerato se Dn>(<ea>).Indefinito negli altri casi

CHK <EA>,Dn

size: WORD


esempio
```

```
*il programma seguente mostra l'utilizzo del codice chk
*il programma gestisce uno coda interna di qlen  word, ogni volta che si cerca di fare
un inserimento
*attraverso chk si controlla se la coda e' piena, se lo e' si genera una  eccezione che
*manda a video la scritta di queue overflow.
    org     $8000  start at address $8000
*procedure enqueue: inserisce un dato in coda
enq         movea.l tail,a0         carica in a0 il puntatore alla coda della struttura
coda
    move.b      d0,(a0)         se non è pieno mette il valore d0 in coda
    adda.l  #2,a0           ed incrementa il puntatore alla coda
*                           controlla se il blocco di memorizzazione è pieno
    add.l   #1,d1
    chk     #qlen,d1
eend        rts

start       move            $0000,sr
    lea.l   stacke,sp
    lea.l   queue,a0
    move.l  a0,head
    move.l  a0,tail
    clr     d1
rep     jsr enq
    jmp rep
    stop    #$2000          istruzione privilegiata illecita, genera
*                   eccezione a 8 a $20
```

```
    org     $8300
ter     equ     $a000
ans     dc.b    'error: queue overflow ',0
qlen    equ     20      lunghezza della coda
queue   ds.w    qlen    blocco di memorizzazione
head    ds.l    1
tail ds.l   1
stack   ds.l    50      spazio per gli indirizzi di ritorno
stacke  equ     *
    org     $8500   isr per la gestione della exception
    stop    #$a000


    org             $8600       isr  per la gestione della ecezione generata da chk
    movea.l  #ter,a2
    movea.l  a2,a0
        add.l       #1,a2
    move.b          #$30,(a2)  setta il registro di controllo del terminale
    lea.l   ans,a1
output      move.b          (a1)+,d0
    cmp     #0,d0    se il carattere da inviare è 0 termina

    beq     fine
    move.b  d0,(a0)          manda il carattere d0 a video
    bcc     output
fine        rte


    end     start
```


2.57 CHK Check Register against Bounds

```
0100 reg 110 ea
if Dn<0 or Dn>(<ea>) then TRAP

X N Z V C
- * U U U

N : settato se Dn<0; resettato se Dn>(<ea>);
    indefinito altrimenti.

CHK <ea>,Dn
Size : W

reg - Specifica il registro dato il cui contenuto viene testato.
ea  - Specifica l'operando word che rappresenta il limite
      superiore del confronto. Sono concessi solo data addressing
      modes.
*



esempio
INSTR  ORG   $9200
*Questa ISR preleva una stringa dal buffer di tastiera, trasforma eventuali *caratteri minu
caratteri maiuscoli e la porta nella memoria centrale.
```

```
*Essa è associata ad una linea di interruzione con priorità 1 e quindi il suo *indirizzo de
all' indirizzo $64 della ROM.

         MOVEM.L A0/A1/A2/D0,-(SP)  vengono salvati i registri usati
         MOVE.L    #TRDAT,A0
         MOVE.L    #TRCTR,A1
         LEA       BUF,A2
while    MOVE.B   (A0),D0          il carattere viene portato in D0
         CMP.B     #13,D0          si controlla se la stringa è terminata
         BEQ       fin
         SUB.B     #$41,D0         si controlla se il carattere sia maiuscolo
         CHK       #12,D0          in caso contrario viene generata una TRAP
         ADD.B     #$41,D0         viene rinormalizzato il codice
         MOVE.B    D0,(A2)+        si inserisce il carattere nel buffer
         BRA       while
fin      MOVE.B    D0,(A2)+        l'enter viene inserito nel buffer e
*                                  funge da terminatore

*A scopo di prova il risultato viene displayato
OSTR
        ORI.B      #$0C,(A1)           pulisce il buffer di tastiera e
*                                      effettua il clearscreen
        LEA     BUF,A2
while1  MOVE.B  (A2)+,D0
        CMP.B   #13,D0
        BEQ     fine
        MOVE.B  D0,(A0)
        BRA     while1
fine    MOVE.B  D0,(A0)+
        MOVE.B  #$33,(A1)              si ripristinano le condizioni iniziali *
del terminale

        MOVEM.L  (SP)+,A2/A1/A0/D0  i registri salvati vengono ripristinati
        RTE


CONV   ORG  $9300
*Questa routine controlla se in DO vi sia un carattere minuscolo
*nel qual caso lo trasforma in maiuscolo
*La routine è chiamata da una trap generata dal codice CHK il cui
*Vector Number è 7 e quindi il suo indirizzo deve comparire allo
*indirizzo $18 della ROM

             CMP.B   #$20,D0
             BLT       rit
             CMP.B   #$39,D0
             BGT       rit
             SUB.B   #$20,D0
rit          RTE

*Esempio di programma chiamante

MAIN   ORG  $9500
BUF    DS.B   256
TRDAT  EQU  $2000
TRCTR  EQU  $2001

VIA    MOVE.L #$11111111,A0      viene messo un valore arbitrario nei registri
       MOVE.L #$22222222,A1      usati dalla ISR per provare il corretto funz.
       MOVE.L #$33333333,A2      del codice MOVEM
```

```
       MOVE.L #$44444444,D0
       ANDI       #$D8FF,SR      passaggio a stato utente

       MOVE.B #$33,(TRCTR)       inizializzazione terminale: si abilitano echo *
tastiera e interruzioni

LOOP   JMP    LOOP               attesa di CR
       END    VIA
*
```

2.58 CLR Clear an operand

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.59 CLR Clear an operand

```
01000010 Size Ea
dest=0

X N Z V C
- 0 1 0 0

CLR<Ea>

Size: B,W,L


esempio
```

2.60 CLR Clear an operand


```
01000010 Size ea
dest=0

X N Z V C
- 0 1 0 0

CLR <ea>
size: B,W,L

EA può essere solo un “Data alterable addressing mode”
*


esempio
      ORG $9420
      LOC DS.B 2

*Il codice viene provato per alcuni dei modi
*di indirizzamento consentiti
VIA   MOVE #$FFFF,D0
      CLR D0
      LEA.L LOC,A0
      MOVE.B #$FF,(A0)
      CLR.B (A0)
      MOVE.B #$FF,(A0)
      CLR.B (A0)+
      MOVE.B #$FF,(A0)
      CLR.B -(A0)
      MOVE.B #$FF,1(A0)
      CLR.B 1(A0)
      MOVE.B #$FF,1(A0,D0)
```

```
      CLR.B 1(A0,D0)
      MOVE.B #$FF,LOC
      CLR.B LOC
      END VIA
Le prove hanno dato risultati correti
*
```

2.61 CMP Compare

```
1011 R mode ea
dest-sorg

X,N,Z,V,C
- * * * *
N=1 se il bit più significato è 1, 0 altrimenti
Z=1 se i due dati sono uguali
V=1 se il risultato di dest-sorg genera overflow
C=1 se il risultato di dest-sorg genera un carry

CMP <ea>,Dn
size: B,W,L
```

esempio :

```
* Programma che utilizza la routine di inizializzazione di array
monodimensionali
* ARRIN. Questo programma non fa altro che dichiarare tre array ;
* il primo di byte, il secondo di word ed il terzo di long word.
* dopo la dichiarazione li inizializza con tre diversi valori.

   ORG    $4000               Localizza il programma
SIZE1     EQU    35                  Dimensione primo array
SIZE2     EQU    6                   Dimensione secondo array
SIZE3     EQU    8                   Dimensione terzo array
ARR1      DS.B   SIZE1               Riserva spazio per primo array
ARR2      DS.W   SIZE2               Riserva spazio per secondo array
ARR3      DS.L   SIZE3               Riserva spazio per terzo array
VAL1      EQU    53                  Primo valore inizializzazione
VAL2      EQU    32001               Secondo valore inizializzazione
VAL3      EQU    1000000                    Terzo valore inizializzazione

*** Procedura ARRIN : effettua l'inizializzazione di un array.
* L'indirizzo dell'array è posto in A0.
* La dimensione dell'array è posta in D2.
* il tipo di dato è identificato da un valore posto in D0 :      1 = Byte
*
   2 = Word
*                                  4 = Long
* Nel caso in D0 sia presente un valore diverso da questo allora la routine
* procederà come se il dato fosse un Byte.
* Il valore di inizializzazione è posto in d1.

ARRIN     CMP.B  #1,D0               if D0=1 then goto INIB
   BEQ    INIB
   CMP.B  #2,D0               if D0=2 then goto INIW
   BEQ    INIW
   CMP.B  #4,D0               if d0=4 then goto INIL
   BEQ    INIL
INIB      MOVE.W #0,D0               d0:=0
INIB1     MOVE.B D1,(A0,D0)          ^a0+d0:=d1
   ADD.W  #1,D0               d0:=d0+1
   CMP.W  D2,D0               if d0<d2 then goto inib1
   BLT    INIB1               else goto arrend
   BGE    ARREND
```

```
INIW      MULU   D0,D2              d2:=d2*d0
   MOVE.W #0,D0                 d0:=0
INIW1     MOVE.W D1,(A0,D0)          ^a0+d0:=d1
   ADD.W  #2,D0                 d0:=d0+2
   CMP.W  D2,D0                 if d0<d2 then goto iniw1
   BLT    INIW1                 else goto arrend
   BGE    ARREND
INIL      MULU   D0,D2              d2:=d2*d0
   MOVE.W #0,D0                 d0:=0
INIL1     MOVE.L D1,(A0,D0)          ^a0+d0:=d1
   ADD.W  #4,D0                 d0:=d0+4
   CMP.W  D2,D0                 if d0<d2 then goto inil1
   BLT    INIL1                 else goto arrend
ARREND    RTS                        fine subroutine



* Inizio codice
MAIN      MOVE.W #SIZE1,D2           D2 := SIZE1
   MOVE.B #1,D0                 D0 := 1
   MOVE.B #VAL1,D1                  D1 := VAL1
   LEA.L  ARR1,A0                   A0 := &ARR1
   JSR    ARRIN                 Inizializza ARR1
   MOVE.W #SIZE2,D2             D2 := SIZE2
   MOVE.B #2,D0                 D0 := 2
   MOVE.W #VAL2,D1                  D1 := VAL2
   LEA.L  ARR2,A0                   A0 := &ARR2
   JSR    ARRIN                 Inizializza ARR2
   MOVE.W #SIZE3,D2             D2 := SIZE3
   MOVE.B #4,D0                 D0 := 4
   MOVE.L #VAL3,D1                  D1 := VAL3
   LEA.L  ARR3,A0                   A0 := &ARR3
   JSR    ARRIN                 Inizializza ARR3
   END    MAIN                  Fine del codice
```


2.62 CMP Compare

```
1011 R mode ea
(destination)-(source)

- * * * *
N=1 se il risultato è <0
Z=1 se il risultato è 0
V=1 se si genera overflow
V=¬S_m·D_m·¬R_m + S_m·¬D_m·R_m
C=1 se si genera borrow
C=S_m·¬D_m + R_m·¬D_m + S_m·R_m
X non viene modificato

CMP <ea>,Dn
size: B,W,L

Quando il size dell'operazione è "byte" il modo di
indirizzamento "registro diretto" (con riferimento
all'operando sorgente) non è permesso.


esempio

* Questo programma trova il max in un vettore
```

```
* di words

   ORG $9000
VET DC.W  $0105,$05F0,$5F01,$0003,$0000

BEGIN  LEA.L VET,A1       indirizzo vettore in A1
     MOVE.W  (A1)+,D0   primo elemento in D0
CICLO  MOVE.W  (A1)+,D1   elemento generico in D1
       CMP.W  D0,D1           confronta elementi
      BLE  CICLO1       se D1>D0 allora
      MOVE.W D1,D0       il max va in D0
CICLO1 TST.W D1        il vettore è finito?
      BNE CICLO
     END BEGIN
```

2.63 CMP Comparazione

Confronta <destinazione> con <sorgente> effettuando la sottrazione; il risultato non viene memorizzato ma viene usato per modificare i bit di flag. <Destinazione> può essere solo un registro dato.

```
1011 R mode ea
*dest-*sorg

X,N,Z,V,C
- * * ? ?

N=1 se MSB(*dest)=1
Z=1 se *dest=0
V=¬S_m·Dm·¬R_m+Sm·¬D_m·Rm
C=S_m·¬Dm+R_m·¬Dm+S_m·Rm

CMP<ea>,Dn
size: B,W,L
```



```
*Esempio di CMP

   org $1000

dim      equ 5
vect     ds.w dim
vmax     ds.w 1
vmin     ds.w 1

*Verifico se gli elementi di un vettore sono contenuti
*in un intervallo specificato, se è così z=1.
*In a0 è contenuto l'indirizzo iniziale del vettore (di dimensione
dim),
*in d0 e d1 sono contenuti rispettivamente estremo inferiore e
superiore
*dell'intervallo.

contrv   move.b #dim,d2
ciclo    cmp.w (a0),d0      elemento del vettore deve essere
   bgt outr           >d0
   cmp.w (a0)+,d1
   blt outr           <d1
```

```
   sub.b #1,d2
   bne ciclo          z=1 se esco dal ciclo
   bra fine
outr     andi.b #$fb,ccr    pongo z=0
fine     rts

start    andi.w #$00ff,sr
   *carico il vettore vect, vmax e vmin
   move.w vmin,d0
   move.w vmax,d1
   move.w #vect,a0
   jsr contrv
   end start
```

2.64 CMPI Compare immediate

1101 R mode ea
dest=dest+sorg

\- * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$ ADD <ea>,Dn

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L

esempio

2.65 CMPI Compare immediate

00001100 size ea + 1,2 ext word
*dest-imm

X,N,Z,V,C
* * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

CMPI #<imm>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- Immediate
- Relative
- Relative indexed

esempio :

```
* Questo segmento di programma cerca la stringa 'SYNC' all' interno
di un buffer  * di caratteri.
* Nel caso in cui la stringa viene trovata, si salta ad una
subroutine (MSGOUT) * che stampa un messaggio appropriato.
*
INIZIO   org   $800
   move.l#BUFADD,A0         inizializza il puntatore del buffer;
*
CERCAS   move.l      A0,D0              aggira 'Error:invalid
addressing mode';
   cmpi.l#BUFADD+BUFSIZE-4,D0     verifica se è ancora possibile
trovare'SYNC'
   move.lD0,A0              aggira 'Error:invalid addressing mode';
   bgt.s FINE
   cmpi.b#'S',(A0)+         confronta 'S' con il carattere puntato;
   bne.s CERCAS
CERCAY   cmpi.b#'Y',(A0)          confronta 'Y' con il carattere
puntato;
   bne.s CERCAS
   addq.l#1,A0              incrementa il puntatore;
```

```
CERCAN   cmpi.b#'N',(A0)          confronta 'N' con il carattere
puntato;
   bne.s CERCAS
   addq.l#1,A0              incrementa il puntatore;
CERCAC   cmpi.b#'C',(A0)          confronta 'C' con il carattere
puntato;
   bne.s CERCAS
*  jsr   MSGOUT
FINE     nop
*
   org   $8100
BUFADD   dc.b  'DSFGSYBNSYNKHSYNCAEXCLSYN'
BUFSIZE  equ   25
   end   INIZIO
```

2.66 Note :

- Il modo di indirizzamento ‘Address-direct-register’non è permesso, contrariamente a quanto è scritto sia sul manuale (‘Only data alterable addressing modes are allowed’) sia sul Wakerly (‘a_dst’).

2.67 CMPM Compare memory

```
1101 R mode ea
dest=dest+sorg

- * * ? ?
V=¬S_m·D_m·¬R_m+S_m·¬D_m·R_m
C=S_m·¬D_m+R_m·¬D_m+S_m·R_m ADD  <ea>,Dn

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.68 CMPM Compare memory

```
1011 Rx 1 Size 001 Ry
CMP Ay@+ TO Ax@+

X, N, Z, V, C
-  *  *  *  *

N = 1 se il bit più significativo di dest-sorg è 1
Z = 1 se i due operandi sono uguali
V = 1 se dest-sorg genera un overflow
C = 1 se dest-sorg genera un riporto

CMPM (Ay)+,(Ax)+
size: B,W,L


esempio
* Programma che utilizza la subroutine di confronto fra due stringhe COMPST


   ORG    $4000               Localizza il codice
ST1DC.B   'CIAO AMICO',13
   DS.W   0
ST2DC.B   'CIAO AMICO',13

*** Subroutine COMPST ***
* Questa subroutine confronta due stringhe di caratteri terminate da un
carattere
* di ENTER ( codice ASCII 13).
* In A0 ed A1 si trovano gli indirizzi di inizio delle due stringhe da
confrontare.
* In D2 viene restituito 0 se le stringhe sono differenti altrimenti viene
restituito 1
* Il contenuto dei registri D0 e D1 viene modificato dalla subroutine.
* Per il confronto dei caratteri vengono usati i codici CMP e CMPM.


COMPST    MOVE.L #1,D2               Inizializzazione di D2 al valore 1
CST1      MOVE.B (A0),D0                    d0 := (a0)
   MOVE.B (A1),D1                   d1 := (a1)
   CMP.B  #13,D0              if d0 = ENTER then goto ENT1
   BEQ    ENT1
   CMP.B  #13,D1              if d1 = ENTER then goto ENT2
```

```
   BEQ    ENT2
CONFR     CMPM   (A0)+,(A1)+          if (a0)+ = (a1)+ then goto CST1

   BEQ    CST1
DIVERSE   MOVE.L #0,D2                d2 := 0
   BRA    CSFINE               goto CSFINE
ENT1      CMP.B  #13,D1               if d1 = ENTER then goto CSFINE
   BEQ    CSFINE               else goto DIVERSE
   BNE    DIVERSE
ENT2      CMP.B  #13,D0               if d0 = ENTER then goto CSFINE
   BEQ    CSFINE               else goto DIVERSE
   BNE    DIVERSE
CSFINE    RTS                         Fine subroutine

MAIN      LEA.L  ST1,A0
   LEA.L  ST2,A1
   JSR    COMPST
   END    MAIN
```

2.69 CMPM Compare memory

```
1011 Rx 1 size 001 Ry
(dest)-(source)

- * * * *
N=1 se il risultato è <0
Z=1 se il risultato è 0
V=1 se si genera overflow
V = ¬S_m·D_m·¬R_m + S_m·¬D_m·R_m
C=1 se si genera borrow
C = S_m·¬D_m + R_m·¬D_m + S_m·R_m
X non viene modificato

CMPM (Ay)+,(Ax)+
size: B,W,L

CMPM effettua la sottrazione tra due operandi referenziati mediante il modo
di indirizzamento
indiretto con postincremento.


esempio

       ORG $8000
DATI        DC.W $01F4,$01B3
BEGIN  LEA.L DATI,A0
     LEA.L DATI+2,A1
     CMPM.W  (A0)+,(A1)+
    END BEGIN

Dopo l'istruzione CMPM:
Z=V=0;
N=1 (risultato negativo);
C=1 (si è generato borrow);
```


2.70 CMPM Comparazione in memoria

Confronta <destinazione> con <sorgente> effettuando la sottrazione; il risultato non viene memorizzato ma viene usato per modificare i bit di flag. Viene sempre usato l'indirizzamento

postincrementato, gli operandi quindi sono sempre dei registri di memoria il cui indirizzo è contenuto nei registri indirizzo specificati nell'istruzione.

```
1011 Rx 1 size 001 Ry
*dest-*sorg

X,N,Z,V,C
- * * ? ?

N=1 se MSB(*dest)=1
Z=1 se *dest=0
V=¬S_m·D_m·¬R_m+S_m·¬D_m·R_m
C=S_m·¬D_m+R_m·¬D_m+S_m·R_m ADD  <ea>,Dn

CMPM      (Ay)+,(Ax)+
size: B,W,L
```

```
*Esempio di CMPM

   org $1000

dim      equ 20
vect1 ds.w dim
vect2    ds.w dim

*Dati due vettori stabilire se sono uguali
*se lo sono settare il bit z
*L'indirizzo iniziale dei vettori deve essere posto in a0 e a1.

compv    move.b #dim,d0          contatore per il ciclo
ciclo    cmpm (a0)+,(a1)+
   bne diversi             z=0 se sono diversi
   sub.b #1,d0
   bne ciclo               z=1 esco dal ciclo (sono uguali)
diversi  rts

start    move.w #$0000,sr        inizializzo sr
   *inserisco dati significativi nei due vettori
   move.w #vect1,a0        carico indirizzo iniziale
   move.w #vect2,a1        dei due vettori
   jsr compv
   end start
```

2.71 DBcc Test condition, decrement and branch

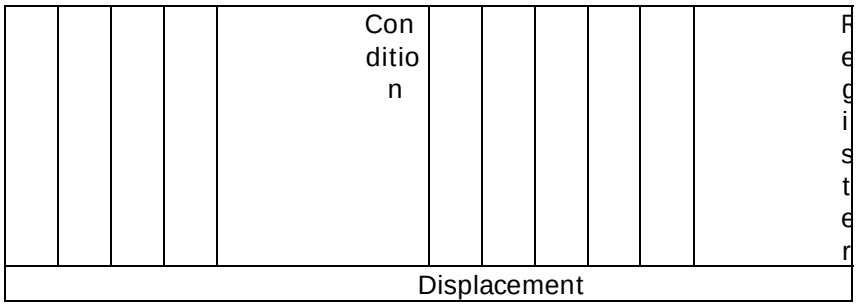
```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.72 DBcc Test condition, decrement and branch

```
0101 Condition 11001 Register
       Displacement(d)
if ¬cc then Dn-1=>Dn;if Dn<>-1 then Pc+d=>Pc

X N Z V C
- - - - -

DBcc Dn,<label>


ESEMPIO

Questa subroutine trova la prima occorenza di un dato passato in D0
in un array il cui indirizzo e' passato in A0 e la cui lunghezza e'
data in D1.
Se il dato e' trovato il flag z=1 ed A0 punta alla locazione
succesiva al dato altrimenti z=0.

ORG      $8000 Indirizzo di partenza

ESDBCC
       BRA LAB2
LAB1   CMP.B (A0)+,D0
LAB2   DBEQ   D1,LAB1
       RTS

ORG    $9000   Area dati
BUFFER  DC.B   128,30,40,40
BUFSIZ   EQU   4

MAIN
     LEA.L BUFFER,A0 Carica in A0 l'indirizzo
                     dell'array
     MOVE.L #40,D0   Carica in D0 il dato da
                     cercare
     MOVE.L #BUFSIZ,D1  Carica in D1 la dimenzione
                        dell'array
     JSR    ESDBCC   Salto a sottoprogramma
```

```
END MAIN
```

DB_{CC} Decrementa, controlla e salta



DBcc Dn,<label>

operazione effettuata : if ¬cc then Dn=Dn-1 ; if Dn<>-1 then PC + spiazzamento -> PC

Unico size consentito è word.

Non sortisce alcun effetto sul CCR (bit di flag).

Condizioni implementate :

	Test su			Test su	
	0100 -	¬C		0011 -	C | Z
	0101 -	C		1101 -	(N&
	0111 -	Z		1011 -	N
	0001 -			0110 - Not	¬Z
	1100 -	(N&V) | (¬N&¬V)		1010 -	¬N
	1110 -	(N&V&¬Z)|(¬N&		0000 -	
	0010 - High	¬C&¬Z		1000 -	¬V
	1111 - Less	Z|(N&¬V)|(¬N&V		1001 -	V

L'istruzione DBcc verifica prima il codice condizione e poi esegue decremento e salto se la conddizione non è soddisfatta.

Nel caso in cui la condizione non è mai soddisfatta, ovvero DBF, l'assembler accetta l'istruzione DBRA, che vuole un registro dati come sorgente e una etichetta come destinazione.
In questo caso il registro viene comunque decrementato, e quindi, alla fine del ciclo, non conterrà un valore nullo.
Se il valore del registro è zero si esegue l'istruzione successiva, altrimenti si esegue il salto all'etichetta specificata.

2.73 DIVS Signed divide

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
V=overflow di divisione

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.74 DIVS Signed divide

```
1000 R 111 ea
dest=dest/sorg

- * * * 0

N=1 se il quoto e' negativo. Nullo altrimenti. Indefinito se c'e' OVERFLOW.
Z=1 se il quoto e' nullo. Nullo altrimenti. Indefinito se c'e' OVERFLOW.
V=1 overflow di divisione. Nullo altrimenti.

DIVS      <ea>,Dn
size: W

esempio
* Esempio d'uso delle istruzioni: DIVS, MOVEQ, SUB, DIVU;
* tramite la configurazione terminal.cfg .
* Il programma esegue la divisione con segno tra due numeri interi
* immessi da tastiera, il risultato sara' mostrato sul video del terminale.
* Range valori applicabili:    Numero A ----> +/- 131071
*                              Numero B ----> +/-  32767
* N.B.
* Per questa versione non sono previsti forme particolari
* di controllo dell'input
* pertanto bisogna porre attenzione durante l'immissione dei dati.
* Tale esempio e' usato fondamentalmente per mostrare l'uso
* della istruzione DIVS del Motorola 68000,
* ma e' anche utile notare l'uso della MOVEQ per azzerare i registri,
* operazione che eseguita tramite tale istruzione risulta molto veloce.
* Sono usate anche le operazioni DIVU e SUB, la prima si usa come
* divisore decimale mentre la seconda e' usata in modo particolare per
* trasformare una cifra ASCII in una cifra binaria, la cui differenza nella
* codifica e' che la cifra BINARIA si ottiene sottraendo un
* numero (una specie di DISPLACEMENT) dalla cifra ASCII,
* Il valore di tale numero e' proprio pari allo '0' ASCII=$30,
* e' chiaro che sottraendo $30 dalla cifra ASCII ottengo proprio
* il numero 0 BINARIO (stessa cosa per le altre 9 cifre).
* Da notare la forma usata per l'istruzione SUB:
* invece di scrivere "SUB #$30,d4" si e' preferito usare la forma
* "SUB #'0',d4" che ha lo stesso effetto, dato che rappresenta
* lo stesso valore, ma in questa forma migliora la comprensione del codice.
* Elaborazione: Gruppo 10 - 96/97.

   ORG     $8200
TEREQU     $2000
MEMEQU     $8000
```

```
   MOVEQ.L      #0,D2  *  azzera registri  *
   MOVE.L D2,D4
   MOVE.L D4,D5
   MOVE.L D4,D6
   MOVE.L D4,D7
NUMOUT    DS.L   2      * definiz. zona mem per risultato in output *
* --------------------- definiz. stringhe di output ------------------------
*
NUM1      DC.B   'Immetti numero A = ',0    * ( ---- range +/- 131071 ---- )
*
NUM2      DC.B   'Immetti numero B = ',0    * ( ---- range +/- 32767 ----- )
*
TOTDC.B   'Risultato A / B = ',0
BEGIN     MOVEA.L #TER,A2      * inizializzazione terminal *
   MOVEA.L A2,A0
   ADD.L   #1,A2 * attivazione reg. controllo e stato *
   MOVE.B  #$30,(A2)    * abilitaz. buffer tastiera ed eco *
SCELTA    CMP.B #0,D6   * se d6=0 siamo all'input del primo num *
          BEQ    FIRST  * quindi va alla first                    *

                         * altrimenti d6=1: siamo al secondo num *
   LEA.L  NUM2,A1       * carica ind 2' num in A1         *
   BRA    OUTPUT * prosegue saltando ad OUTPUT     *
FIRST             LEA.L  NUM1,A1       * carica ind 1' num in A1         *
OUTPUT    MOVE.B  (A1)+,D0     * richiesta a video dei numeri su cui operare
   *
   CMP.B   #0,D0 * controllo per fine stringa  *
   BEQ     CONT  * e' 0 ?  SI: vai a cont *
   MOVE.B  D0,(A0)      * altrimenti continua a mostrare i caratteri a video
*
   BCC     OUTPUT       * salta ad output *
CONT      MOVE.B  (A2),D1      * input numero da tastiera *
   AND.B   #$80,D1      * controlla se e' premuto il <cr> *
   BEQ     CONT  * NO: vai a cont e continua input *
   MOVEA.L #MEM,A1      * carica indirizzo di mem in a1 *
IN1MOVE.B  (A0),D0      * mette in d0 ogni cifra memorizzata nel buffer *
   CMP.B  #13,D0 * controlla se <cr> premuto, cioe' D0 e vuoto ? *
   BEQ    INMEM         * Si: salta a inmem *
SWITCH    CMP.B  #'-',D0       * controlla se la prima cifra ascii e' il '-'
*
          BEQ    Segno  * altrimenti prosegue da switch   *
* -----   inizio conversione ascii2bin             -----*
   MOVE.B D0,D4  * trasferisce la cifra ascii in d4 per convertirla *
   SUB.B  #'0',D4       * sottrae lo '0' ascii e la converte cosi' in binario
*
   MULS   #10,D5 * moltiplica d5 per 10, aumentanto il peso decimale *
   ADD.L  D4,D5         * addiziona la cifra binaria a d5 *
   BRA    IN1           * salta a in1 *
SEGNO     MOVE.B #1,D2  * il numero e' negativo, pone 1 in d2 come reminder *
   BRA    IN1           * prosegue da inmem *
*--------------  fine conversione --------------------*
INMEM     CMP.B  #1,D2  * d2=1 ? Cioe' il numero e' negativo? *
   BNE    VAI1   * NO: salta a vai1 *
   NOT.L  D5     * SI: complementa a 2 il numero in d5 (lo nega)  *
   ADDI   #1,D5  * (e poi gli addiziona 1) *
   MOVEQ  #0,D2  * azzera d2, il reminder di num negativo *
VAI1      CMP.B  #1,D6  * controlla se d6=1  *
   BEQ    CONT0  * quindi salta a cont0   *
   MOVE.L D5,D7  * altrimenti carica D7 col primo numero convertito  *
   MOVEQ  #1,D6  * pone d6=1 quindi indica che si deve passare al 2' num *
```

```
   MOVEQ  #0,D5  * azzera D5 che serve di nuovo per la convers ascci2bin  *

   BRA    BEGIN  * ripeti il ciclo di input a video *
                 * Al termine, in D5 resta il 2' numero convertito, *
                 * il 1' numero convertito e' in D7  *
CONT0      DIVS   D5,D7  * * operazione che il programma deve eseguire * *
   MOVE.W  SR,D5 * sr in d5, per conservarlo prima di manipolare i bit *
   MOVE.L D7,D3  * porta il numero: MSW=resto + LSW=quoto in d3 *
   SWAP   D3     * inverte MSW con LSW *
   ANDI.L #$FFFF,D3  * cancella il quoto da d3 che ora e' il resto *
   ANDI.L #$FFFF,D7  * cancella il resto da d7 che ora e' il quoto *
   MOVE.W D5,SR  * restituisce lo sr orig. prima di manipolarne i bit *
   BPL    CONV   * salta a conv se il risultato non e' negativo (bit N=0) *
   NOT.W  D7     * complementa a 1 d7 che era negativo *
   ADDI   #1,D7  * addiziona 1 a d7, il numero e' ora positivo*
                 * pronto per essere convertito in ascii
   MOVEQ  #1,D2  * indica segno *
CONV       LEA    NUMOUT,A3           * va alla fine di numout *
   move.B #$0D,(A3)+    * inserisce il carattere <cr> come fine output *
LOOP       DIVU   #10,D7 * divide per 10 il numero, il risultato restera' in
d7 *
                         * ris:  MSW=resto div  LSW=quoziente div *
   MOVE.L D7,D5      * carica in d5 il contenuto di d7  *
   SWAP   D5     * inverte MSW con LSW in modo che ora LSW=resto div *
   ANDI.L #$FFFF,D5   * azzera l'MSW, cosi' lascia in d5  *
                              * solo il resto della divs *
   ANDI.L #$FFFF,D7   * azzera l'MSW, cosi' lascia in d7 *
                              * solo il quoziente della divs *
   ADD.W  #'0',D5       * somma $30 a d5 cosi' conv. da BIN ad ASCII la cifra
*  MOVE.B D5,(A3)+      * carica la cifra in ASCII in memoria *
   CMP    #0,D7  * quoziente div=0 significa: fine cifre numero bin *
   BNE    LOOP   * salta se le cifre bin sono ancora disponibili in d7 *
   CMP    #1,D2  * verifica segno *
   BNE    POS    * salta se e' positivo *
   MOVE.B #'-',(A3)+    * inserisce il segno in mem *
POS LEA.L   TOT,A1       * presenta a video la stringa di output *
OUT1       MOVE.B  (A1)+,D0     * carica la stringa cifra per cifra *
   CMP.B   #0,D0        * termina quando trova lo 0 finale *
   BEQ     OUT2  * salta a cont *
   MOVE.B  D0,(A0)      * continuo l'output a video del risultato *
   BCC     OUT1  * torna a out1 finche' chi sono caratteri nel buffer *
OUT2       MOVE.B  -(A3),D0     * pone in d0 il risultato, cifra per cifra *
   MOVE.B  D0,(A0)      * lo trasferisce a video *
   CMP.B   #13,D0       * finche' non trova il carattere di <cr> *
   BNE     OUT2  * altrimenti ripete da out2 *
   STOP   #$2000 * arresta il programma  *
   END     BEGIN
```

2.75 DIVS Signed divide

```
1000 R 111 EffAddr
dest=dest/sorg

- * * ? 0
V=overflow di divisione
N= dest<0
Z= dest=0

DIVS      <ea>,Dn
size: B,W,L
```

Esempio: **retta.a68**

```
*   Utilizzo del codice operativo: DIVS
*   Programma per il calcolo dei punti appartenenti ad un segmento.
*   Dati:
*          punto iniziale [x1,y1]
*          punto finale [x2,y2]
*   Gli assi del piano vanno da [-200,200] per le x e da [-200,200] per le y
*   I punti vengono calcolati attraverso la formula:
*   y=[[y2-y1]*x + y1*x2-y2*x1] / [x2-x1]
*   Questo programma può essere utilizzato come programma di libreria
*   in un sistema operativo che gestisce il video.
*
   ORG    $8000
X1 DC.W   2      ORDINATA DEL PRIMO PUNTO
Y1 DC.W   2      ASCISSA DEL PRIMO PUNTO
X2 DC.W   6      ORDINATA DEL SECONDO  PUNTO
Y2 DC.W   9      ASCISSA DEL SECONDO  PUNTO
BUFFER    DS.W   800    SPAZIO IN MEMORIA RISERVATO A CONTENERE ASCISSA E
*                        ORDINATA DEI PUNTI CALCOLATI

START     MOVE   Y2,D1
   SUB    Y1,D1          (Y2-Y1)
   MOVE   X2,D2
   MULS   Y1,D2          (Y1)*(X2)
   MOVE   X1,D3
   MULS   Y2,D3          (Y2)*(X1)
   SUB.L  D3,D2          (Y1)*(X2)-(Y2)*(X1)
   MOVE   X2,D3
   SUB    X1,D3          (X2-X1)
   MOVE   X1,D0
   MOVE.L #BUFFER,A0
   MOVE.W D0,(A0)+              SALVA L'ASCISSA DEL PUNTO DATO, X1
   MOVE.W Y1,(A0)+              SALVA L'ORDINATA DEL PUNTO DATO, Y1
LOOP      ADDQ.B #1,D0                 INCREMENTA LA X DI 1
   MOVE.L D0,D5                 REGISTRO DI APPOGGIO
   MULS   D1,D5          (Y2-Y1]*X
   ADD.L  D2,D5          (Y2-Y1]*X+ Y1*X2-Y2*X1
   DIVS   D3,D5          [[Y2-Y1]*X + Y1*X2-Y2*X1] /  [X2-X1]
   MOVE.W D0,(A0)+              X IN BUFFER
   MOVE.W D5,(A0)+              Y IN BUFFER
   CMP.W  X2,D0                       RIPETI IL LOOP FINO A QUANDO  X=X2
   BMI    LOOP

   STOP   #$2000

   END    START
```

2.76 DIVU Unsigned divide

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
V=overflow di divisione

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.77 DIVU Unsigned divide

```
1000 R 011 ea

dest/sorg-->dest

- * * * 0

N=1 se il quoziente ha il bit più significativo pari a uno
Z=1 se il quoziente è nullo
V=1 overflow di divisione

DIVU      <ea>,Dn

size: W


esempio
*     Il programma effettua il MCD tra due numeri, ognuno *
*  contenuto in una WORD ;di conseguenza il range di valori *
*  applicabili e' compreso tra 1 & 32767           *
*     Elaborazione: Gruppo 10 - 96/97

   ORG    $8000
DD1EQU    32765
DV2EQU    360
   *      assegna il valore minimo al registro d0  *
START     MOVE.W #DD1,D0       *  inizializza il registro d0 *
          CMP.W  #DV2,D0       *  confronta con l'altro valore*
          BLE    MIN           *  salta se e' minore o uguale *
          MOVE.W #DV2,D0       *     dv2 e' piu' piccolo  *
          MOVE.W #DD1,D1
          BRA           INIZIO
MIN       MOVE.W #DV2,D1       *      dv2 e' piu' grande   *
INIZIO    MOVE.W D1,D2         *      d2 e' di appoggio          *
          DIVU          D0,D2          *      divisione unsigned   *
          ANDI.L #$0000FFFF,D2 *  seleziona solo il quoziente *
          MULU          D0,D2          *      moltiplicazione unsigned  *
          MOVE.W D1,D3         *      d3 e' di appoggio          *
          SUB.W         D2,D3          *      calcolo della diff.  *
          CMP           #0,D3          *      se d3=0 dd1 e dv2 sono      *
          BEQ           MCD                   *      multipli di  d0      *
          MOVE.W D0,D1         *      per  iterare il calcolo
          MOVE.W D3,D0         *      finche'       d3<>0 aggiorna i  *
          BRA           INIZIO *      registri      d1 e d2       *
```

```
MCD       MOVE.W D0,MCD1       *      reg. il valore in memoria*
MCD1             DS.W          1
          END           START
```

2.78 DIVU Unsigned divide

```
1000 R 011 EffAddr
dest=dest/sorg

- * * ? 0
V=overflow di divisione
N= bit più sign. alto
Z= dest=0

DIVU      <ea>,Dn
size: B,W,L
```

Esempio: decbcd.a68

```
*  Utilizzo dei codici operativi: DIVU e OR.
*  Conversione binario->BCD
*  Può essere utilizzato come routine per rappresentare un numero su un
*  display.
*
   ORG    $8000
DECIM     EQU    5240               Numero da convertire
BASE      EQU    10                 Base BCD
START     CLR.L  D4                 Azzera peso shift
   MOVE.L #DECIM,D1

*Conversione mediante succesive divisioni per 10

LOOP      DIVU   #BASE,D1
   SWAP   D1                  Porta il resto nella LSW
   LSL.W  D4,D1               Effettua lo shift a sinistra:0->unità  4-
>decine
*                             8->centinaia...
   OR.W   D1,D2               Compone il numero finale
   SWAP   D1                  Riporta il quoto nella LSW
   AND.L  #$0000FFFF,D1 Azzera il resto
   ADDQ.B #4,D4         Incrementa lo shift
   CMP.W  #BASE,D1
   BPL    LOOP                Il ciclo termina quando il quoto è minore di
10
   LSL.W  D4,D1
   OR.W   D1,D2               Termina la composizione del numero shiftandolo
*                             al peso finale
   END    START
```

2.79 EOR Exclusive or logical

```
1101 R mode ea
dest=dest+sorg

- * * 0 0

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.80 EOR Exclusive OR logical

```
1011 R Op-mode ea
EOR Dn TO <ea>

X, N, Z, V, C
-  *  *  0  0

EORDn,<ea>
size: B,W,L


esempio :

*** Subroutine che inverte il bit 4 del registro di controllo del device
*   TERMINAL di ASIM. L' indirizzo del registro è memorizzato in A0.
*   Viene modificato il valore di D0

TTOGGL    MOVE.W #%00010000,D0
   EOR.W  D0,(A0)
   RTS
```


2.81 EOR Exclusive or logical

```
1011 R mode ea
dest=dest+sorg

- * * 0 0
X,N,Z,V,C
N=1 se MSB del risultato=1
Z=1 se il risultato=0

EOR Dn,<ea>
size: B,W,L

per <ea> sono permessi solo modi di indirizzamento
di tipo "data alterable"


esempio1
* Questo programma esegue il complemento a 256
* di un byte usando EOR

    ORG $8000

BYT   DC.B  $F0
```

```
START MOVE.B (BYT),D0
    EOR     #$FF,D0   effettua l'operaz.D0=256-D0
    END START

esempio2

    ORG $9200
DATI  DC.W $FFFF,$FFFF
BEGIN MOVE.W (DATI),D0
    MOVE.W (DATI+2),D1
    EOR.W D0,D1
    END BEGIN

Dopo l'istruzione EOR:
N=0;
Z=1  (FFFF xor FFFF = 0)
```

2.82 EOR Or esclusivo logico

Or esclusivo tra <destinazione> e <sorgente>, il risultato viene posto in <destinazione>;<sorgente> può essere solo un registro dati.

```
1011 R mode ea
dest<=*dest XOR *sorg

X,N,Z,V,C
- * * 0 0

N=1 se MSB(*dest)=1
Z=1 se *dest=0

EOR Dn,<ea>
size: B,W,L
```



```
* $VER: tglcase 1.0 (11-12-96)
*
*   NOME
*     A2D -- Converte, in una stringa, le maiuscole in minuscole e viceversa.
*
*   FUNZIONE
*     Questo programma converte i caratteri di una stringa, che rappresentano
*     lettere, da maiuscole a minuscole e viceversa.
*
*   INGRESSI
*     A0 - Punta alla stringa da convertire. Peraltro la stringa e' intesa
*           terminata con il carattere NULL (ASCII).
*
*   RISULTATO
*      A0 - Punta alla stringa convertita come spiegato in FUNZIONE.
*
*  ESEMPIO
*
               org  $1000
STRINGA        DS.B      255                   Buffer per la stringa da convertire.
          DS 0

TGLCASE        MOVE.B   (A0),D0               Preleva dalla memoria il prossimo
*                                              carattere da manipolare.
               BEQ      FINE_STRINGA          Se il carattere corrente e` il
*                                              catarrere NULL, la stringa e`
*                                              terminata.
               CMP.B    #$41,D0               Verifica se il carattere e` maiuscolo:
               BCS      NON_LETTERA            se il carattere corrente precede il
```

```
                CMP.B    #$5A,D0              carattere 'A' oppure segue il carattere
                BLS      LETTERA               'Z', allora non e` maiuscolo.
                CMP.B    #$61,D0              Verifica se il carattere e` minuscolo:
                BCS      NON_LETTERA           se il carattere corrente precede il
                CMP.B    #$7A,D0              carattere 'a' oppure segue il carattere
                BHI      NON_LETTERA           'z', allora non e` ne' minuscolo ne'una
*                                              lettera.
LETTERA         EOR.B    #$20,D0              Se il carattere corrente e` una
                MOVE.B   D0,(A0)+             lettera maiuscola diventa minuscola e
*                                              vicerversa se e` minuscola diventa
*                                              maiuscola.
NON_LETTERA     BRA      TGLCASE
FINE_STRINGA    RTS

*

INIZIO          MOVEM.L  A0,-(SP)             Salva sullo stack il contenuto del
*                                              registro che serve per operare.
                MOVE      #STRINGA,A0        Punta alle locazioni dei fattori da
*                                              moltiplicare.
                BSR      TGLCASE               Esegue la conversione della stringa.
                MOVEM    (SP)+,A0             Ripristina il contenuto precedente
*                                              nel registro servito per operare.
                END      INIZIO
```

2.83 EORI Exclusive or immediate

```
1101 R mode ea
dest=dest+sorg

- * * 0 0

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.84 EORI Exclusive or immediate

```
00001010 size ea + 1,2 ext word
dest ← dest ⊕ sorg

X,N,Z,V,C
- * * 0 0
N=1 se MSB(*dest)=1

Z=1 se *dest=0
EORI      #<data>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- diretto con registro indirizzo
- relativo
- relativo indicizzato
- immediato


esempio :

* L'esempio permette di complementare i bit delle long word di un vettore,
* fornendone così la rappresentazione per complementi diminuiti .
*
VETINI    org    $8000         indirizzo del primo elemento del vettore
VETTORE   dc.l   $ABCD1234,$EF567890,$0A0B0C0D,$12345678,$FEDCBA98
NUMEL     dc.b   5             numero degli elementi del vettore
*
INIZIO    org    $8050
   moveq.l      #0,D0         D0 contiene l'indice del primo elemento
   move.l #VETINI,A0
LOOP      move.l D0,D1
   mulu.w #4,D1         D1 contiene lo spiazzamento dell'elemento da settare
   move.l 0(A0,D1),D2
   eori.l #$FFFFFFFF,D2 effettua la complementazione di tutti i bit
   move.l D2,0(A0,D1)   aggiorna il vettore in memoria
   addq.l #1,D0         aggiorna l'indice dell'elemento
   cmp.b  NUMEL,D0      controlla se sono esauriti gli elementi del vettore
   bne.s  LOOP
*
FINE      nop
   END    INIZIO
```

2.85 EXG Exchange register

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.86 EXG Exchange register

```
1100 R 1 mode R
 Rx↔Ry

X,N,Z,V,C
- - - - -

EXG Rx,Ry
size: L
```


esempio:

```
Il seguente sottoprogramma riceve in ingresso tre valori in d1,d2,d3 e li
ordina in maniera crescente a partire da d1

ordina      cmp         d1,d2
   ble      sec
   exg      d1,d2       d2 è più piccolo di d1
sec cmp     d1,d3
   ble      ter
   exg      d1,d3       d3 è più piccolo di d1
ter cmp     d2,d3
   ble      fine
   exg      d2,d3       d3 è più piccolo di d2
fine        rts
```

2.87 EXG Exchange register

```
1100 Rx 1 mode Ry
dest⇔sorg

- - - - -
X,N,Z,V,C

EXG Rx,Ry
size: L

Se lo scambio è tra Dn e An allora Rx è sempre Dn
ed Ry è sempre An


esempio
```

```
* Il seguente programma inverte il contenuto di una
* stringa servendosi del codice EXG.
* Lo scambio dei singoli bytes costituenti i caratteri
* della stringa da invertire avviene dagli estremi al
* centro della stessa.

    ORG $8000
STRIN DC.B 'Questa stringa sarà invertita',0

START LEA.L   STRIN,A1     indirizzo stringa in A1
    MOVE.L  A1,A3        e in A3
LOOP1 MOVE.B  (A1)+,D0   questo ciclo calcola
    CMP.B   #O,D0      l'indirizzo dell'ultimo
    BNE       LOOP1      carattere della stringa

    MOVE.L  A1,A4
LOOP  MOVE.B (A3),D0    carica i caratteri
    MOVE.B (A4),D1    nei registri-dato D0 e D1
    EXG D0,D1         scambia caratteri
    MOVE.B  D0,(A3)+  registra i caratteri
    MOVE.B  D1,-(A4)  in memoria
    CMPA    A3,A4     confronta indirizzi
*                          caratteri da scambiare
    BGT LOOP               se A4<=A3 il centro
      END START         stringa è raggiunto
```

2.88 EOR Or esclusivo logico

Or esclusivo tra <destinazione> e <sorgente>, il risultato viene posto in <destinazione>;<sorgente> può essere solo un registro dati.


```
1011 R mode ea
dest<=*dest XOR *sorg

X,N,Z,V,C
- * * 0 0

N=1 se MSB(*dest)=1
Z=1 se *dest=0

EOR Dn,<ea>
size: B,W,L
```

```
* $VER: tglcase 1.0 (11-12-96)
*
*   NOME
*     A2D -- Converte, in una stringa, le maiuscole in minuscole e viceversa.
*
*   FUNZIONE
*     Questo programma converte i caratteri di una stringa, che rappresentano
*     lettere, da maiuscole a minuscole e viceversa.
*
*   INGRESSI
*     A0 - Punta alla stringa da convertire. Peraltro la stringa e' intesa
*          terminata con il carattere NULL (ASCII).
*
*   RISULTATO
*      A0 - Punta alla stringa convertita come spiegato in FUNZIONE.
*
*  ESEMPIO
*
```

```
                org  $1000
STRINGA         DS.B      255                    Buffer per la stringa da convertire.
           DS 0

TGLCASE         MOVE.B    (A0),D0               Preleva dalla memoria il prossimo
*                                                carattere da manipolare.
                BEQ       FINE_STRINGA           Se il carattere corrente e` il
*                                                catarrere NULL, la stringa e`
*                                                terminata.
                CMP.B     #$41,D0               Verifica se il carattere e` maiuscolo:
                BCS       NON_LETTERA            se il carattere corrente precede il
                CMP.B     #$5A,D0               carattere 'A' oppure segue il carattere
                BLS       LETTERA                'Z', allora non e` maiuscolo.
                CMP.B     #$61,D0               Verifica se il carattere e` minuscolo:
                BCS       NON_LETTERA            se il carattere corrente precede il
                CMP.B     #$7A,D0               carattere 'a' oppure segue il carattere
                BHI       NON_LETTERA            'z', allora non e` ne' minuscolo ne'una
*                                                lettera.
LETTERA         EOR.B     #$20,D0               Se il carattere corrente e` una
                MOVE.B    D0,(A0)+              lettera maiuscola diventa minuscola e
*                                                vicerversa se e` minuscola diventa
*                                                maiuscola.
NON_LETTERA     BRA       TGLCASE
FINE_STRINGA    RTS

*

INIZIO          MOVEM.L   A0,-(SP)              Salva sullo stack il contenuto del
*                                                registro che serve per operare.
                MOVE       #STRINGA,A0         Punta alle locazioni dei fattori da
*                                                moltiplicare.
                BSR       TGLCASE               Esegue la conversione della stringa.
                MOVEM     (SP)+,A0              Ripristina il contenuto precedente
*                                                nel registro servito per operare.
                END       INIZIO
```

2.89 EXT Signed extend

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.90 EXT Sign-extend

```
01000100 Op-mode 000 Reg
(dest) Esteso in▶segno  Dest

X N Z V C
- * * 0 0

N=1 se il risultato è negativo, azzerato altrimenti
Z=1 se il risultato è zero, azzerato altrimenti
V è sempre azzerato
C è sempre azzerato
X non viene modificato

EXTDn

Size: W,L


esempio
```



```
* LE SEGUENTI LINEE DI PROGRAMMA
* MOSTRANO L'UTILIZZO DEL CODICE EXT
* IL CODICE PUO' ESSERE UTILIZZATO PER
* OPERAZIONI DI CASTING.

* N.B.: OCCORRE PORRE ATTENZIONE AL CASO IN CUI
*       GLI OPERANDI SON UNSIGNED ED HANNO IL BIT
*       PIU' SIGNIFICATIVO PARI AD 1

* ESEMPIO: SOMMA DI UN BYTE ED UNA WORD SIGNED

*area dati
     ORG $8300
       wrd: dc.w $1234

*area programma
    ORG $8100
 start: move.w wrd,d0
        move.b #$07,d1
        ext.w d1          *d1 ora contiene il valore $0007
        add.w d1,d0       *somma in complementi: d0= $123B
        end start
```

ESEMPIO 2
(somma di un byte ad una word)

```
*area dati
      ORG $8300
        wrd: dc.w $1234

*area programma
     ORG $8100
 start: move.w wrd,d0
        move.b #$80,d1  * d1=10000000
        ext.w d1            * d1=$FF80
        add.w d1,d0       * somma in complementi: d0=$11b4
        end start

* se d1 era unsigned la somma avrebbe dovuto produrre d0=$12B4
* si puo ovviare al problema introducendo dopo la linea ext.w d1
* l'istruzione andi.w #$00FF,d1
```

ESEMPIO 3
(Somma di un byte ed una word signed)

```
*area dati
      ORG $8300
        wrd: dc.w $1234

*area programma
     ORG $8100
 start: move.w wrd,d0
        move.b #$80,d1  * d1=10000000
        ext.w d1            * d1=$FF80
        andi.w #$00FF,d1
        add.w d1,d0       * somma in complementi: d0=$11b4
        end start

* se d1 era unsigned la somma avrebbe dovuto produrre d0=$12B4
* si puo ovviare al problema introducendo dopo la linea ext.w d1
* l'istruzione andi.w #$00FF,d1
```


2.91 EXTW Sign-extend low order byte of data register to word

```
0100100010000 Dn
Dn - sign-extend -> Dn

X N Z V C
-   *  * 0 0
set N if the result is negative.Cleared otherwise
set Z if the result is zero.Cleared otherwise

EXTW Dn
*


esempio
```

2.92 EXTL Sign-extend low order word of data register to Long

```
0100100011000 Dn
Dn - sign-extend -> Dn

X N Z V C
-   *  * 0 0
set N if the result is negative.Cleared otherwise
set Z if the result is zero.Cleared otherwise

EXTL Dn
*


esempio
```

2.93 JMP Jump

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.94 JMP Jump

```
0100111011 EA

dest =>PC

X N Z V C
- - - - -

Jump <ea>
size: Unsized


esempio
```


2.95 JMP Jump

```
0100 1110 11 ea
PC=Destinazione

X N Z V C
- - - - -

Tutti i flag sono not affected

JMP <ea>
Unsized

ea - Specifica l'indirizzo della prossima istruzione
    (destinazione). Sono concessi solo control addressing
     modes.


esempio
```

2.96 JSR Jump to subroutine

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.97 JSR Jump to Subroutine

```
0100111010 Effective Address

PC=>SP@-
Destination=>PC

X N Z V C
- - - - -

JSR <ea>

size:Unsized


esempio
```


2.98 JSR Jump to Soubroutine

```
0100 1110 10 ea
SP@-=PC ; PC=Destinazione

X N Z V C
- - - - -

Tutti i flag sono not affected

JSR <ea>
Unsized

ea - Specifica l'indirizzo della prossima istruzione
    (destinazione). Sono concessi solo control addressing
     modes.


esempio
Usato in più esempi.
```

2.99 LEA Load effective address

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.100 LEA Load Effective Address

```
0100 Register 111 Effective Address
Destination=>An
LEA <ea> into Am

X N Z V C
- - - - -
size: Long


esempio
```


2.101 LEA Load Effective Address

```
0100 reg 111 ea
lea <ea> into An

X N Z V C
- - - - -

Tutti i flag sono not affected

LEA <ea>,An
Size : L

reg - Spesifica il registro indirizzo in cui sarà caricato
      l'effective address.
ea  - Specifica l'operando il cui effective address andrà caricato
      in An. Sono concessi solo control addressing modes.
*


esempio
```

2.102 LINK Link and allocate

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.103 LINK Link and allocate

```
0100111001010 Reg
→       An  SP@-→ SP      An ,SP+d       SP

X N Z V C
- - - - -

Nessun flag di CC viene modificato


LINK      An,#<displacement>

Size: Unsized


esempio
```



```
* LE SEGUENTI LINEE DI PROGRAMMA MOSTRANO L'UTILIZZO
* DEI CODICI LINK E UNLK.
* TALI CODICI SONO UTILI PER L'ALLOCAZIONE DINAMICA
* DEI PARAMETRI SULLO STACK.iL REGISTRO A6 SARA'
* UTILIZZATO COME FP.

    ORG $8000       *AREA DATI
frst  equ 0
last equ 4
size equ 2
num dc.w  1,2,5,4,3
eanum equ num-(frst*size)

   ORG $9000    *AREA STACK
stk ds.w 40
stke equ *

  ORG $8100 *AREA SUBROUTINE
* l'array contiene i valori 1 2 5 4 3
* la routine cerca il max e la sua posizione lasciando l'output sullo stk

cercamax  link a6,#0  *salva il fp,aggiorna fp e non alloca var locali
```

```
                movea.w  indnum(a6),a0
                move.w #frst,d0
                move.w d0,d1
                mulu.w #size,d1
                move.w d1,a1
                adda.w a0,a1
                move.w #frst,d2  *si assume che il max sia il primo
                move.w (a1),d3
iloop           add.w #1,d0         *si incrementa l'indice d0
                move d0,d1
                mulu.w #size,d1
                move.w d1,a1
                adda.w  a0,a1       *a1 punta al successivo
                cmp.w (a1),d3       *confronto col max corrente
                bge skip
                move.w (a1),d3      *a1 punta al nuovo max
                move.w d0,d2        *l'indice d0 va salvato in pos
skip            cmpi.w #last,d0
                ble iloop         *a fine ciclo d2 e d3 contengono il max e la
                                   *sua posizione
                move.w d2,pos(a6)
                move.w d3,max(a6)
                unlk a6

indnum equ 8
pos     equ 10
max     equ 12

                rts

* area programma
    ORG $8400
start  andi.w #$DFFF,sr  *pone il processore in stato utente
       movea.l  #stke,sp
       movea.l  sp,a6       * a6 frame pointer
       adda.w #-4,sp        *riserva area parametri di output sullo stk
       move.w #eanum,a0
       move.w a0,-(sp)     *passa l'array per indirizzo
       jsr cercamax
       move.w -(a6),d0     *max
       move.w -(a6),d1     *pos del max
       end start
```


2.104 LINK **Link and Allocate**

```
01001110010 Register Displacement
Push(An) ; An=SP ; SP+d→SP

X N Z V C
- - - - -

LINK An,#<displacement>
size: unsized
```

-Register: specifica il registro indirizzo attraverso il quale viene realizzato il link
-Displacement: Specifica l’intero espresso in complementi che deve essere sommato allo start pointer
*

2.105 LSL Logical shift left

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.106 LSL Logical shift left

```
1110 count|Reg 1 size i/r 01 R (Register Shifts)
dest<=dest<<count
1110 001 111 ea               (Memory Shifts)
dest<=dest<<1

X,N,Z,V,C
* * * 0 ?
C= D m-count+1

X,N,Z,V,C ( count == 0 )
- * * 0 0

LSLDx,Dy
LSL#<data>,Dy
LSL<ea>

size: B,W,L ( Register Shifts )
size: W     ( Memory Shifts )


esempio

   ORG    $8000

* Si mostra come sia possibile implementare l'operazione di divi-
* sione, unsigned, di dividendo a 32 bit con divisore a 16 bit
* utilizzando operazioni ai SUB e di LSL.
* L'esempio è strutturato come subroutine che effettua l'operazio-* ne
desiderata, a questo proposito si assume che in D0 vi sia il * dividendo e
nella parte bassa di D1 vi sia il divisore.
* La routine restituisce in D0 il risultato dell'operazione, nella * word
bassa il quoto e nella word alta il resto; D1 rimane inal- * terato. Se il
divisore è nullo oppure il quoto non è rappresen- * tabile in 16 bit D0
rimane inalterato mentre i bit 16 e 17 ri-  * spettivamente di D1 vengono
posti ad 1

start

* DIVISIONE PER ZERO
   MOVE.L #$FE351234,d0
```

```
   MOVE.L #0,d1
   jsr    DIVIDE

* OVERFLOW DI DIVISIONE
   MOVE.L #$FE351234,d0
   MOVE.L #$1234,d1
   jsr    DIVIDE

* DIVISIONE “NORMALE”
   MOVE.L #$FE351234,d0
   MOVE.L #$FF34,d1
   jsr    DIVIDE

   STOP   #$80

* ROUTINE DI DIVISIONE
DIVIDE
   MOVE.L d2,-(a7)      Salvataggio sullo stack di D2

   CMP.W  #0,d1  Test divisione per 0
   BNE    NOOVFL1
* divisione per zero => set del bit #16 di D1 ad 1 ed uscita dalla * routine

*  BSET.L       #16,D1 istruzione non funzionante per cui usiamo
   OR.L   #$10000,d1
   BRA    ENDDIV
NOOVFL1
   SWAP.W d1     porta il divisore nella word alta di D1
   CMP.L  d1,d0  determina se il quoto sia rappresentabile  *
in 1 word
   BCS    NOOVFL2
* overflow nella rappresentazione del quoto => set del bit #17 di  * D1 ed
uscita dalla routine
   SWAP.W d1     riporta il divisore nella word bassa di D1
*  BSET.L       #17,D1 istruzione non funzionante per cui usiamo
   OR.L   #$20000,d1
   BRA    ENDDIV
NOOVFL2
* utilizziamo il size L perché nell'attuale implementazione DBF   * che
decrementa il registro D2 lo decrementa a 32 bit => se uti- * lizzassimo il
size B o W senza "pulire" la parte restante del   * registro il ciclo
potrebbe non effettuare il numero di iterazio-* ni previsto
   MOVE.L #15,d2 D2 = loop count
* ciclo di divisione; la precondizione sempre verificata è:
* D0[31..16] < D1[31..16]
FORDIV
* Shift Left del dividendo (corrisponde ad abbassare una cifra nel * classico
algoritmo di divisione)
* nel LSB di D0 verranno man mano memorizzate le cifre del quoto
   LSL.L  #1,d0
   BCS    CSETDIV1
* il MSB di D0 prima del LSL era 0
   CMP.L  d1,d0  determina se D1[31..16] < D0[31..16]
   BCS    CSETDIV2
  * D1[31..16] >= D0[31..16]
   SUB.L  d1,d0  D0[31..16] =  D0[31..16] - D1[31..16]
   BSET.L #0,d0  la cifra attuale del quoto è 1
   BRA    LABDIV
CSETDIV2
  * D1[31..16] < D0[31..16]
```

```
   BCLR.L #0,d0  la cifra attuale del quoto è 0
   BRA    LABDIV
CSETDIV1
* il MSB di D0 prima del LSL era 1
   SUB.L  d1,d0  D0[31..16] =  D0[31..16] - D1[31..16]
   BSET.L #0,d0  la cifra attuale del quoto è 1

LABDIV    DBF    d2,FORDIV
ENDDIV
   MOVE.L (a7)+,d2      Ripristino di D2
   RTS

   end start
```

2.107 LSR Logical shift rigth

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.108 LSR Logical shift rigth

```
1110 count|Reg 0 size i/r 01 R (Register Shifts)
dest<=dest>>count

1110 001 011 ea               (Memory Shifts)
dest<=dest>>1

X,N,Z,V,C
* * * 0 ?
C= D count-1

X,N,Z,V,C ( count == 0 )
- * * 0 0

LSRDx,Dy
LSR#<data>,Dy
LSR<ea>

size: B,W,L ( Register Shifts )
size: W     ( Memory Shifts )


esempio

   ORG    $8000

* Esempio di utilizzo dell'istruzione LSR
* Si mostra come sia possibile implementare l'operazione di mol-
* tiplicazione, unsigned, di operandi a 16 bit con risultato a 32 * bit
utilizzando operazioni ai ADD e di LSR.
* L'esempio è strutturato come subroutine che effettua l'operazio- * ne
desiderata, a questo proposito si assume che nella parte
* bassa di D0 vi sia il moltiplicatore, a 16 bit, e che nella
* parte alta vi sia il moltiplicando.
* La routine restituisce in D1 il risultato dell'operazione
* Per un descrizione dell'algoritomo vedi Wakerly pag.103

start

   MOVE.L #$12341234,d0
   MOVE.L #12345,d2
```

```
   jsr    MULTIPLY2

   MOVE.W #$FF34F234,d0
   jsr    MULTIPLY2

   STOP   #$80


MULTIPLY2
   MOVE.L d2,-(a7)      Salvataggio sullo stack di D2
* utilizziamo il size L perché nell'attuale implementazione DBF   * che
decrementa il registro D2 lo decrementa a 32 bit => se uti- * lizzassimo il
size B o W senza "pulire" la parte restante del   * registro il ciclo
potrebbe non effettuare il numero di iterazio-* ni previsto
   MOVE.L #15,d2 D2 = loop count

   MOVE.L d0,d1  Inizializzazione D1 = risultato parziale
   AND.L  #$0000FFFF,d1 Pulisce la parte alta di D1
   AND.L  #$FFFF0000,d0 Pulisce la parte bassa di D0


FOR2      BTST.L #0,d1  testa il LSB del moltiplicando shiftato
   BEQ           NOADD2
   ADD.L  D0,D1   somma il moltiplicando shiftato al
*                        risultato parziale
   BCS    CARSET2
* il LSB del moltiplicatore shiftato era nullo
NOADD2
   LSR.L  #1,d1   shift a destra del risultato parziale e
*                         del moltiplicando
   BRA    LAB2

CARSET2          * l'addizione ha generato carry
   LSR.L  #1,d1   shift a destra del risultato parziale e
*                         del moltiplicando

*  BSET.L        #31,d1 istruzione non funzionante per cui usiamo
   OR.L   #$80000000,d1   pone il MSB del prodotto parziale ad 1

LAB2      DBF    d2,FOR2

   MOVE.L (a7)+,d2      Ripristino di D2
   RTS

   end start
```

2.109 MOVE Move data from source to destination

```
00 size (reg mode)D (mode reg)S
dest<=*source

- * * 0 0

MOVE <ea>,<ea>
size: B,W,L


esempio
```

2.110 MOVE Move data from source to destination

```
00 size (reg mode)D (mode reg)S
dest ← *source

X,N,Z,V,C
- * * 0 0

MOVE <ea>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- relativo
- relativo indicizzato
- immediato
esempio :

*Sottrazione tra due operandi memoria attraverso registri data
*
          org    $8200
INIZIO            MOVE.L MINUENDO,D0
          MOVE.L SOTTRAENDO,D1
          SUB.L  D0,D1         *RISULTATO IN D1
          MOVE.L D1,RISULTATO

FINE              NOP
*
MINUENDO  DC.L   $9E2A127C
SOTTRAENDODC.L   $2AB35481
RISULTATO DS.B   1
*
          END    INIZIO

*Scambio tra due vettori in ordine inverso
*
          org    $8200
INIZIO            LEA    VETTORE1,A0          IND. DI INIZIO DI VETTORE1 IN A0
                  LEA    VETTORE2+10,A1              IND. DI FINE    DI
VETTORE2 IN A1
          MOVE.B #10,CNT                      INIZIALIZZAZIONE DEL CONTATORE

CICLO             MOVE.B        (A0)+,-(A1)
          SUBI.B #1,CNT
          BNE    CICLO

FINE              NOP

VETTORE1  DC.B   $9E,$2A,$12,$7C,$2E,$F4,$72,$8C,$A2,$B4
```

```
VETTORE2  DS.B   10
CNT       DC.B   0

          END    INIZIO
```

*Sposta in un vettore di word la word + significativa di una serie di longword.
*

```
          org    $8200
INIZIO    LEA    SORGENTE,A0   *IND. DI SORGENTE IN A0
          LEA    DESTINAZIONE,A1     *IND. DI DESTINAZIONE IN A1
          MOVE   #0,D0

CICLO     MOVE   (A0,D0),(A1)
          ADDI.B #4,D0
          ADDA   #2,A1
          CMP    #20,D0
          BNE    CICLO

FINE      NOP

SORGENTE  DC.L   $9E347B2C,$21E3F60A,$17AB7B20,$11AE37F6,$31FFE10A
DESTINAZIONE     DS.W   5

          END    INIZIO
```

2.111 MOVE to CCR Move to condition codes

```
0100010011 ea
CCR<=*sorg

? ? ? ? ?
I flag saranno settati in accordo al valore posto in sorg

MOVE <ea>,CCR
size: W


esempio
```

2.112 MOVE to CCR Move to condition codes

```
0100010011 Ea
CCR<=(sorg)

X N Z V C
* * * * *
I flag saranno settati in accordo al valore posto in sorg

MOVE <Ea>,CCR

Size: W


esempio :
```



```
*************************************************************************************
*PROGRAMMA DI PROVA DEL  CODICE MOVE to CCR
*Il programma seguente mostra l'utilizzo del codice MOVE to CCR  per settare in
*maniera opportuna i codici di condizione in una procedura di ricerca lineare di una
*word in un vettore di word. In particolare MOVE to CCR viene utilizzato per
*abbassare il flag Z utilizzato dalla procedura Srch come indicatore dell'esito
*della ricerca:  Z=1  -> esito postivo
*           Z=0  -> esito negativo
*************************************************************************************
*Area Programma a partire dalla locazione $8000
    ORG   $8000
START     move.w          sr,d0              legge il registro di stato
    andi.w  #$d8ff,d0               maschera per reg stato (stato utente, int abilitati)
    move.w          d0,sr              pone liv int a 000
    move.l  #$0d300,sp     inizializza stack pointer

    lea       Array,a0
    move.w          #$800,d0
    jsr       Srch
    move.w          #$810,d0
    lea       Array,a0
    jsr       Srch
    stop        #$FF00

Srch        nop
*
```

```
*
*Ricerca nel vettore Array (il cui indirizzo base è in a0) la word contenuta in d0
*ed esce con Z=1 se l'elemento cercato è presente.
*
*
ciclo       move.w      (a0)+,d2
            cmp.w  d0,d2
            beq.s  fine             elemento presente -> esce con Z=1
    cmp.w  #riemp,a0
            bne.s  ciclo
    jsr     resetZ          esito della ricerca negativo -> esce con Z=0
fine        rts

*
*Routine per azzerare il flag Z
*
resetZ      move   sr,d1
    andi    #$fffb,d1
    move    d1,ccr
    rts

*Area dati a partire dalla locazione $9000
    org     $9000
Array       dc.w
    $8000,$4000,$2000,$1000,$800,$400,$200,$100,$80,$40,$20,$10,$8,$4,$2,$1
riemp       equ     Array+32
    end     START
```


2.113 MOVE to CCR

```
0100010011 ea
CCR=SORG

X N Z V C
? ? ? ? ?
I flag saranno sostituiti dai bit di sorg

MOVE <ea>,CCR
size: W

EA può essere solo un “Data alterable addressing mode”
*


esempio

      ORG $9420

VIA   MOVE #$FFFF,CCR
      MOVE #0,CCR
      MOVE #$000F,CCR
      END VIA
```

2.114 MOVE to SR Move to the status register (istr. priv.)

```
0100011011 ea
SR<=*sorg

? ? ? ? ?
I flag saranno settati in accordo al valore posto in sorg

MOVE <ea>,SR
size: W


esempio
```

2.115 MOVE to SR move to status register (istr.priv.)

```
0100011011ea
SR=(source)

X N Z V C
*  *  *  *  *
Tutti i flag sono settati in accordo al valore posto in source

MOVE <ea>,SR
size: W

ea - specifica la locazione dell'operando source.
       Sono concessi solo data addressing modes.
*


esempio
```

2.116 MOVE from SR Move from the status register

```
0100000011 ea
dest<=SR

X,N,Z,V,C
- - - - -

MOVE SR,<ea>
size: W
esempio
```

2.117 MOVE from SR

```
01000011 Size ea
dest=SR

X N Z V C
- - - - -
Tutti i flag sono not affected.

MOVE SR,<ea>
size: W

EA può essere solo un “Data alterable addressing mode”
*


esempio
```


2.118 MOVE from SR Move from the status register

```
0100000011 Ea
dest<=SR

X N Z V C
- - - - -

MOVE SR,<Ea>

Size: W

esempio :
   Vedi esempio su MOVE to CCR


MOVE USP         Move user stack pointer
010011100110 dr reg
An<=USP; USP<=An

- - - - -

MOVE USP,An
MOVE An,USP
size: L
esempio
```

2.119 MOVEA Move address

```
1101 R mode ea
dest=dest+sorg

- - - - -

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.120 MOVEM Move multiple register

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.121 MOVEM Registers to EA Move multiple registers to effective address

```
010010001 Sz Ea
Dest=Registers

X N Z V C
- - - - -

MOVEM     <Register list>,<Ea>

Size: W,L


esempio :
```


2.122 MOVEM Registers to EA Move Multiple Registers to EA

```
010010001 Size ea
<Registers List Mask>
Dest=Registers

X N Z V C
- - - - -
Tutti i flag sono not affected

MOVEM < Register list>,<ea>
Size= W,L

-Sz                 Specifica la lunghezza dei registri che devono
                    essere trasferiti
-ea                 Specifica l'indirizzo della locazione di memoria a
                    partire dalla quale i registri saranno trasferiti
                    in memoria . Sono concessi solo control alterable
                    addressing modes oppure l'indir. per predecremento.
-Register List Mask Specifica i registri che saranno trasferiti.
                    L'ordine di trasferimento è il seguente:
                    D0,D1,...,D7,A0,A1,...,A7 per i control modes,
                    A7,A6,...,A0,D7,D6,...,D0 per il modo predecremento.
*
```

2.123 MOVEM EA to Registers Move effective address to multiple registers

```
010011001 Sz Ea
Registers=(Sorg)

X N Z V C
- - - - -

MOVEM     <Ea>,<Register list>

Size: W,L


esempio
```

2.124 MOVEM EA to Registers Move Multiple EA to Register

```
010011001 Sz ea
<Register List Mask>

Registers=(Source)

X N Z V C
- - - - -
Tutti i flag sono not affected

MOVEM <ea>,<register list>
Size= W,L

-Sz                 Specifica la lunghezza dei registri che devono
                    essere operati
-ea                 Specifica l'indirizzo della locazione di memoria a
                    partire dalla quale si trasferirà nei registri.
                    Sono concessi solo control alterable
                    addressing modes oppure l'indir. per postincremento.
-Register List Mask Specifica i registri che saranno trasferiti.
                    L'ordine di trasferimento è il seguente:
                    D0,D1,...,D7,A0,A1,...,A7.
*
```



```
esempio (MOVEM)

MOVM ORG $8000
*Questa routine prova il codice MOVEM per i modi d'indirizzamento "control alterable"; i mo
postincremento e preincremento sono stati usati in vari
altri esempi.

BUF  DS.L 4               buffer per il salvataggio dei registri

VIA
*---------------------------------------------------------------------------
*Prova per Size = Long
         MOVE.L  #$12121212,A0     si caricano valori arbitrari
         MOVE.L #$34343434,A1      nin alcuni registri
         MOVE.L #$56565656,A2
```

```
         MOVE.L #$78787878,D0

         MOVEM.L A0/A1/A2/D0,BUF   i registri vengono salvati
*                                  nel buffer
         MOVE.L #0,A0              vengono azzerati
         MOVE.L #0,A1
         MOVE.L #0,A2
         CLR.L    D0

         MOVEM.L BUF,A0/A1/A2/D0   vengono ripristinati

*---------------------------------------------------------------------
*Prova per Size = Word
         MOVE    #$1212,A0
         MOVE    #$3434,A1
         MOVE    #$5656,A2
         MOVE    #$7878,D0

         MOVEM   A0/A1/A2/D0,BUF

         MOVE.L #0,A0
         MOVE.L #0,A1
         MOVE.L #0,A2
         CLR.L    D0

         MOVEM   BUF,A0/D0/A1/A2   notiamo che l'ordine in cui i registri
*                                  vengono elencati non è importante
*---------------------------------------------------------------------

         END VIA

NOTA: Con Size=Long il codice: "carica da memoria a registri", su alcuni
      computer non funziona per nessuno dei due modi di indirizzamento
      consentiti quando size=Long;infatti carica sempre la stessa costante a
      prescindere dal valore presente in memoria.
*
```

2.125 MOVEP Move peripheral data

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.126 MOVEP

```
0000 R mode 001 R
dest ← (source)

X,N,Z,V,C
- - - - -

MOVEP     Dx,d(Ay)
MOVEP     d(Ay),Dx
size :W,L

Modi di indirizzamento :
- based è l'unico permesso


esempio
*Questo esempio trasporta i byte di 3 longword
*in byte consecutivi.
*
*

   ORG    $8400
INIZIO    LEA    ALTERNATI,A0  INDIR. INIZIALE DI ALTERNATI IN A0
          LEA    MATRICE,A1    INDIR. INIZIALE DI MATRICE IN A1
   MOVE.L #0,D3         CONTATORE POSTO A ZERO

CICLO     MOVE.L       (A1,D3),D1    IN D1 CI SONO 4 BYTE DELLA MATRICE
   MOVEP.L       D1,0(A0)            I 4 BYTE VANNO IN ALTERNATI
   ADD.L  #4,D3         PREDISPOSIZIONE AL PROSSIMO TRASFERIMENTO
   ADDA.L #8,A0         BYTE DI DESTINAZIONE SUCCESSIVI
   ADD.L  #4,D3
   CMPI.B #12,D3
   BNE    CICLO
*
*
MATRICE          DC.L   $71AB2C43,$31A52F4A,$1EAF2144
ALTERNATI DS.B   24
*
*
FINE      NOP
   END    INIZIO
```

note
I trasferimenti su indirizzi dispari non sono consentiti.

2.127 MOVEQ Move quick

```
1101 R mode ea
dest=dest+sorg

- * * 0 0

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.128 MOVEQ Move quick

```
0111  REG  0 data

#<data> --> Dn

- * * 0 0

N=1 se il dato è negativo
Z=1 se il dato è nullo

MOVEQ  #<dato>,Dn

size: L


esempio:
*  Compatta due long-word in unica long-word se la prima
*  long-word è con word 0 nella parte alta e la seconda nella
*  parte bassa.  *
*  Si effettua un preventivo controllo delle due long-word
*  per vedere se e' possibile effettuare il compattamento
*  E'un esempio di utilizzo di MOVEQ(sia per 'pulire' un registro
*  e sia per effettuare inizializzazioni di contatore)e della OR.
*  Elaborazione: Gruppo 10 - 96/97

   ORG    $8000
B1 EQU    $0000EFA1
B2 EQU    $9F6C0000
START     MOVE.L #B1,D1
   MOVE.L #B2,D2
*CONTROLLO DELLA PRIMA LONG:VERIFICA SE E' DEL TIPO UPPER-WORD =0*
          MOVEQ  #0,D7  * PULISCE REGISTRO D7 *
          MOVEQ  #16,D0 * INIZIALIZZA CONTATORE DO A 16*
POP1             LSL.L  #1,D1  * SHIFTA IL REGISTRO DI UNA
                        * POSIZIONE A SINISTRA
          MOVE.W SR,D7     * MASCHERA  SUL  REGISTRO DI STATO *
          ANDI.W #1,D7
          CMP           #1,D7
          BEQ           ESCI   *LA PAROLA NON PUÒ ESSERE COMPATTATA*
          SUBI          #1,D0  * DECREMENTA CONTATORE *
          CMP           #0,D0
          BNE           POP1
          MOVE.L #B1,D1 *  RIPRISTINA IL VALORE DI D1     *
* CONTROLLO DELLA SECONDA LONG:VERIFICA SE E' DEL TIPO LOW-WORD =0 *
          MOVEQ  #0,D7  * PULISCE REGISTRO D7 *
```

```
          MOVEQ  #16,D0 * INIZIALIZZA CONTATORE DO A 16*
POP2            LSR.L  #1,D2  * SHIFTA IL REGISTRO D2 DI UNA POSIZIONE A
DESTRA
          MOVE.W SR,D7  * MASCHERA  SUL  REGISTRO DI STATO *
          ANDI.W #1,D7
          CMP           #1,D7
          BEQ           ESCI   *LA PAROLA NON PUÒ ESSERE COMPATTATA*
          SUBI          #1,D0  * DECREMENTA CONTATORE *
          CMP           #0,D0
          BNE           POP2
          MOVE.L #B2,D2 * RIPRISTINA IL VALORE DI D2      *
          OR.L          D1,D2         * COMPATTA *
          MOVE.L D2,AREA       * REGISTRA IL VALORE IN MEMORIA *
AREA            DS.L          1
ESCI            END           START
```

2.129 MOVEQ Move quick

```
0111 R 0 Data
<Data> -> dest

- * * 0 0

MOVEQ     #<data>,Dn
size: L
```

2.130 UNLK Unlink


```
01001110011 Register
SP=An; POP(An)

X N Z V C
- - - - -

UNLK An
size: unsized

-Register:     specifica il registro indirizzo attraverso il quale
               viene realizzato l’ unlink
*



esempio  (LINK-UNLK)
NSUB    ORG $9200
*Questo esempio è pensato per mostrare l'utilizzo dei codici
*LINK e UNLINK.
*---------------------------------------------------------------
*Subroutine ricorsiva che nega un numero espresso su N word.
*Il numero da negare, N e i flags  sono passati nello stack.
*Il numero negato e i flag risultato vengono restituiti ancora nello stack.

      LINK A6,#0
      MOVE.W LEN(A6),D0       preleva dallo stack il numero di word
                              che la subroutine deve negare
      CMP.L #1,D0             se il numero di word da negare è  1 la
*                             subroutine provvede alla sua negazione
      BEQ unaw

*Abbiamo più di una (n) word da negare:
```

```
*la routine richiama se stessa con n-1 word da negare
      SUB.L #1,D0
      MOVE.L D0,D1            D1 contiene il numero di word (n-1) da
*                             passare alla nuova chiamata della subroutine
      MULU.W #2,D1
ciclo MOVE.W DAT(A6,D1),-(SP) passa le n-1 word da negare
      SUB.L  #2,D1
      CMP.L #2,D1
      BGE ciclo
      MOVE.W FLAG(A6),-(SP)   passa i flags correnti
      MOVE.W D0,-(SP)         passa il numero di word da negare
      JSR NSUB

* Il numero di word da negare è 1
*Preleva dallo stack i risultati delle chiamate precedenti.
unaw  MOVE.W LEN(A6),D1
      SUB.L #1,D1
      BEQ nega
      ADDA.L #2,SP           rilascia  la lunghezza del dato(input)
      MOVE.W (SP)+,FLAG(A6)  preleva i flag precedenti
      MOVE.L #2,D2           spiazzamento per puntare alle word già negate
cicl1 MOVE.W (SP)+,DAT(A6,D2)copia nella propria area dati le word già
*                            negate dalla precedente chiamata della subroutine
      ADD.W #2,D2
      SUB.W #1,D1
      BNE cicl1

*Viene eseguita la negazione
nega      MOVE FLAG(A6),CCR
          NEGX.W DAT(A6)
          MOVE.W SR,FLAG(A6)
RET       UNLK A6
          RTS


*-----------------------------------------------------------------------


       ORG $9500
DATOL  DC.L $00880077
DATOH  DC.L  $00AA0099         numero da negare su 4 WORD
N           DC.W 4             numero di word su cui è espresso il dato

* Spiazzamenti rispetto al FP
LEN       EQU 8                numero di word su cui è espresso il dato
FLAG      EQU 10               word in cui è memorizzato il CCR corrente
DAT       EQU 12               dato da negare

MAIN      ANDI     #$DFFF,SR       passaggio a stato utente
          MOVEA.L #$00000000,A6    inizializzazione FP
*                                 (solo a scopo di riferimento)
*Caricamento dei parametri nello stack
          MOVE.L   DATOL,-(SP)
          MOVE.L   DATOH,-(SP)
          MOVE.W #$0004,-(SP)  poniamo nello stack i valori iniziali dei flag:
*                              il  flag X (riporto entrante) è 0;
*                              il flag Z è 1
          MOVE.W   N,-(SP)
          JSR NSUB             chiama la subroutine
          END MAIN
```

*

2.131 MULS Signed multiply

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.132 MULS Signed multiply

```
1100 R 111 ea
dest=dest*sorg

X N Z V C
- * * 0 0
N=1 se dest<0
Z=1 se dest=0

MULS <ea>,Dn
size: W
```

2.133 MULS Signed multiply


```
1100 R 111 ea
(dest)*(source)=dest

- * * 0 0

N=1 se il risultato è <0
Z=1 se il risultato è 0

MULS <ea>,Dn
size: W

MULS moltiplica due interi con segno rappresentati
in complementi alla base su 16 bit salvando il
risultato nel registro destinazione e rappresen-
tandolo in complementi alla base su 32 bit.


esempio1

    ORG $9200
DATI  DC.W -$0005,$0005

BEGIN MOVE.W (DATI),D0
    MOVE.W (DATI+2),D1
    MULS D0,D1
   END BEGIN

Dopo l'istruzione MULS:
N=1 (risultato negativo);
Z=0;
```

```
esempio2
* Questo programma implementa il prodotto scalare
* di due vettori numerici mediante il codice MULS

     ORG $9200
VET1        DC.W  $1325,$16BD,-$155A,-$517E,-$1111,$3CA1,$0000
VET2        DC.W  -$4317,$73A9,-$2A3F,$09FB,$25AB,$74A3

BEGIN  CLR.L D2      azzera la somma
     LEA.L VET1,A0    carica gli indirizzi
     LEA.L VET2,A1    dei vettori

LOOP        MOVE.W (A0)+,D0  scarica gli elementi
     CLR.L D1        da moltiplicare nei
     MOVE.W (A1)+,D1  registri-dato D0 e D1
     MULS.W D0,D1     moltiplica con segno
     ADD.L D1,D2     aggiorna somma
     TST.W (A0)      il vettore è finito?
     BNE LOOP
     END BEGIN
```

2.134 MULS Moltiplicazione con segno

Esegue la moltiplicazione con segno tra due operandi a 16 bit, di cui l'operando destinazione è un registro dati; il risultato è a 32 bit ed è memorizzato, ovviamente, nello stesso registro dati.

```
1100 R 111 ea
dest<=*(dest)  x  *(sorg)

X,N,Z,V,C
- * * 0 0

N=1 se MSB *(dest)=1
Z=1 se *(dest)=0

MULS      <ea>,Dn
size: W
```



```
* $VER: domuls.a 1.1 (26-11-96)
*   NOME
*     DOMULS -- Moltiplicazione con segno a 16 bit.
*   FUNZIONE
*  Questo programma effettua la moltiplicazione con segno
*  di due stringhe di 16 bit contenute in memoria; il risultato e` a 32 bit e   *
viene posto anch'esso in memoria.
*   INGRESSI
*     A0 - Contiene l'indirizzo di memoria a 32 bit delle due
*     stringhe di 16 bit, poste in memoria in locazioni adiacenti.
*     A1 - Contiene l'indirizzo di memoria a 32 bit in cui siceglie di         *
depositare il risultato della moltiplicazione.
*   RISULTATO
*     (A1) - Contiene il risultato della moltiplicazione.
*  ESEMPIO
    org $1000
FATTORI        DS.W      2
RISULTATO      DS.L      1

domul      move.w (A0)+,D0    preleva dalla memoria il primo
*                          fattore da moltiplicare e punta
*                          al secondo fattore.
    muls.w (A0),D0     Preleva il secondo fattore ed
```

```
*                          effettua la moltiplicazione.
       move.l D0,(A1)        Salva il risultato nella loc.
*                            di memoria ad esso destinata.
    rts

INIZIO          MOVEM.L   D0/A0-A1,-(SP)        Salva sullo stack il contenuto
*                                               dei registri che servono per op.
                MOVE.W  #FATTORI,A0        Punta alle locazioni dei fattori da
moltiplicare
                MOVE.W  #RISULTATO,A1      Punta alla locazione in cui memor. il
risultato
                BSR       DOMUL                      Esegue la moltiplicazione.
                MOVEM     (SP)+,D0/A0-A1        Ripristina il contenuto prec. nei reg.

                END       INIZIO
```

2.135 MULU Unsigned multiply

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.136 MULU Unsigned multiply

```
1100 R 011 ea
dest=dest*sorg

X N Z V C
- * * 0 0
N=1 se il bit più significativo di dest è 1
Z=1 se dest=0

MULU <ea>,Dn
size: W
```


Esempio:
Il seguente sottoprogramma moltiplica due numeri da 16 bit contenuti nei registri d1 e d2 e pone il risultato nei 32 bit di d3

```
mul         movem.l       d2-d4,-(sp)       salva i registri
            move.w        d1,d3                   d1L in d3
            move.w        d2,d4                   d2L in d4
            swap          d1                      d1H in d1
            swap       d2                         d2H in d2
* crea i prodotti
            mulu          d3,d2                   d2=d1L*d2H
            mulu          d4,d1                   d1=d2L*d1H
            mulu          d4,d3                   d3=d2L*d1L
* soma i termini incrociati
            add.w         d2,d1
* pone i termini incrociati nella cifra più alta del
* risultato
            swap          d1
            clr           d1
* inserisce la cifra più bassa
            add.l         d3,d1
            movem.l       (sp)+,d2-d4       ripristina i
*                                 registri usati
rts
```

2.137 MULU Unsigned multiply

```
1100 R 011 ea
(dest)*(source)-->dest

* * * * *
N=1 se l'MSB di dest
Z=1 se dest=0
V è sempre settato basso
C è sempre settato basso
X non viene influenzato

MULU <ea>,Dn
size: W

MULU esegue la moltiplicazione di due numeri
interi "unsigned" su 16 bit producendo un risulta-
to "unsigned" memorizzato su 32 bit.
A causa dell'ampiezza del campo in cui si salva
il risultato non si verifica mai overflow.


esempio

* Questo programma è una variante di quello pre-
* sentato a proposito del codice AND. In questo
* caso le maschere per I confronti vengono gene-
* rate moltiplicando per 2 iterativamente il con-
* tenuto di un registro inizialmente caricato con
* il valore 1

    ORG $8000
NUM   DC.B $F1       numero da processare

BEGIN CLR.B D3
      MOVE.W #1,D1
LOOP  MOVE.B (NUM),D0
    AND D1,D0
    BEQ NOADD       se la AND è <>0
    ADD #1,D3
NOADD MULU #2,D1      prepara prossima maschera
    CMP #$0100,D1   il byte è finito?
    BNE LOOP
      END BEGIN
```


2.138 MULU Moltiplicazione senza segno

Esegue la moltiplicazione senza segno tra due operandi a 16 bit, il risultato è a 32 bit ed è memorizzato in un registro dati.L'operazione è eseguita usando l'aritmetica senza segno, il registro dati contiene (nei primi 16 bit) uno dei due fattori quando inizia l'operazione.

```
1100 R 011 ea
dest<=*dest  x  *sorg

X,N,Z,V,C
- * * 0 0

N=1 se MSB(*dest)=1
```

Z=1 se *dest=0

MULU <ea>,Dn
size: W

```
*Esempio di MULU

   org $1000

*Effettua la moltiplicazione unsigned tra fattori a 32 bit
*il risultato è a 64 bit;i parametri sono passati tramite stack.

f1a equ 8
f1b equ 10
f2a equ 12
f2b equ 14
res equ 8
stato      equ -2

mltip      link a0,#-2
   move.w sr,stato(a0)
   move.w f1a(a0),d0
   move.w f2a(a0),d1
   mulu.w d1,d0           moltiplico le parti alte
   move.w f1b(a0),d1
   move.w f2b(a0),d2
   mulu.w d2,d1           moltiplico parti basse
   move.w f1a(a0),d2
   move.w f2b(a0),d3
   mulu.w d3,d2           p. alta di f1 * p. bassa di f2
   move.w f1b(a0),d3
   move.w f2a(a0),d4
   mulu.w d4,d3           p. bassa di f1 * p. alta di f2
   add.l d3,d2                   sommo parti miste
   move.l d2,d3
   move.b #15,d5
   roxr.l #1,d2           prendo i 16 bit + significativi
   lsr.l d5,d2                   per sommarli al prodotto delle p. alte
   add.b #1,d5
   lsl.l d5,d3                   mentre i 16 bit - sign. li sommo al prod. delle p.
                                 basse
   add.l d3,d1
   addx.l d2,d0
   move.l d0,res(a0)
   move.l d1,res+4(a0)
   move.w sr,d0           setto i bit di flag
   andi.b #$0c,d0
   or.b stato(a0),d0
   move.w d0,ccr
   unlk a0
   rts

start      andi.w #$00ff,sr
   move.w #$32fc,-(sp)           003212de*44de32fc
   move.w #$44de,-(sp)
   move.w #$12de,-(sp)
   move.w #$0032,-(sp)
   jsr mltip
   end start
```

2.139 NBCD Negate decimal with extend

```
1101 R mode ea
dest=dest+sorg

* U ? U ?
C=prestito decimale
Z=Z·¬R_m·....·¬R_0

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.140 NBCD Negate decimal with extend

```
0100100000 Ea
dest=0-(Dest)_10 -X

X N Z V C
* U * U *
N è indefinito
Z è invariato se il risultato è zero, azzerato altrimenti
V è indefinito
C=1 se si è generato un riporto decimale di sottrazione, azzerato altrimenti
X=C

NBCD      <Ea>

Size: B


esempio
```



```
*******************************************************************************
*PROGRAMMA DI PROVA DEL  CODICE NBCD
*Il programma seguente mostra l'utilizzo del codice NBCD  per realizzare la
*sottrazione OP1-OP2 in precisione multipla in aritmetica di decimali.
*L'algoritmo utilizzato è generico e consente di effettuare la sottrazione con
*operandi di lunghezza generica, specificabile tramite le variabili lung e
*definendo quindi opportunamente anche la lunghezza del risultato.
*
*N.B.       L'eventuale riporto di sottrazione sarà  il negato del flag C
*    e in caso di risultato negativo sarà N=1
*******************************************************************************
*Area Programma a partire dalla locazione $8000
    ORG   $8000
START   move.w      sr,d0          legge il registro di stato
    andi.w #$d8ff,d0          maschera per reg stato (stato utente, int abilitati)
    move.w      d0,sr          pone liv int a 000

    movea.l     #$d300,sp
    move.w      lung,d0
    lea     strnd,a1
    lea     mnd,a2
```

```
    movea.l     #result,a3      a3 punta al LSB del risultato
    adda.w lung,a1
    adda.w lung,a2
    subq   #1,d0
    move.w      #$0010,-(sp)    salva i codici di condizione sullo stack
ciclo       move.w      #$0010,ccr      set X
    nbcd   -(a1)
    move.b (a1),d1
    move.b -(a2),d2
    move   (sp)+,ccr                ripristina i codici di condizione del passo
precedente
    abcd   d1,d2            esegue la generica sottrazione x-y come x+(-y)
    move   sr,-(sp)         salva i codici di condizione sullo stack
    move.b d2,-(a3)
    dbra   d0,ciclo
    move   (sp)+,ccr                ripristina i codici di condizione dell'ultima
operazione

*Area Dati a partire dalla locazione $8300
    org    $8300
strnd       dc.l    $12345678,$98765432
mnd         dc.l    $98765432,$12345678
res ds.b    8               lunghezza risultato  = 8byte = 16 cifre
result      equ     *
lung        dc.w    8               lunghezza operandi = 8byte = 16 cifre
    END    START
```


2.141 NBCD Negate Decimal with Extend

```
0100100000 ea
Destination=0-Destination-X

X N Z V C
* U * U *

Z : Z·¬D_m...·¬D_0
C : Decimal borrow
X =C

NBCD <ea>
Size= B

ea - Specifica l'operando Destinazione
       Sono concessi solo data alterable addressing modes
*


esempio
NEGD ORG $9500
*Questa subroutine nega un numero bcd di una data lunghezza.
*Essendo nbcd un operazione esclusivamente sul byte si suppone
*che il numero occupi in ogni caso un numero intero di byte (eventualmente
*si aggiungo zeri in testa).
*La lunghezza del numero è passata in D0 (è corrisponde al numero di cifre
*su cui esso è epresso diviso 2); l'indirizzo di partenza dell'operando è
```

```
*passato in A0.

          ADDA.L D0,A0             il puntatore al dato viene spostato
*                                  al suo byte meno significativo
          SUBQ.L #1,D0             si tiene conto che il codice
*                                  DBcc salta a 0 e non a 1

          MOVE #$0004,CCR        viene settato il bit zero resettati gli altri
ciclo     NBCD -(A0)
          DBEQ D0,CICLO
          RTS

*Esempio di programma chiamante
MAIN   ORG $9200
OP      DC.B $40,$00,$00

VIA     ANDI #$DFFF,SR            passaggio a stato utente

          LEA OP,A0
          MOVEQ.L #3,D0
          JSR NEGD
          END VIA

NOTA: NBCD non opera correttamente sullo zero
*
```

2.142 NEG Negate

1101 R mode ea
dest=dest+sorg

* * * ? ?
$V=D_m \cdot R_m$, $C=D_m+R_m$
$V=D_m \cdot R_m$, $C=D_m+R_m$
$Z=Z \cdot \neg R_m \cdot .... \cdot \neg R_0$ADD <ea>,Dn

ADD Dn,<ea>
size: B,W,L

esempio

2.143 NEG Negate

01000100 Size Ea
dest :=0-(dest)

X N Z V C
* * * * *
N=1 se il risultato è negativo (N:=R_n), azzerato altrimenti
Z=1 se il risultato è zero, azzerato altrimenti
V=1 se si è generato un overflow, azzerato altrimenti
C=1 se si è generato un riporto di sottrazione, azzerato altrimenti
X=C

NEG<Ea>

Size: B,W,L

esempio

```
*******************************************************************************
*PROGRAMMA DI PROVA DEL  CODICE DI NEGAZIONE NEG
*Il programma seguente mostra l'utilizzo del codice NEG  per realizzare la
*sottrazione OP1-OP2 modulo 2^8 e modulo 2^16 mediante addizione modulo 2^8
*e 2^16 rispettivamente.
*******************************************************************************
*Area Programma a partire dalla locazione $8000
    ORG   $8000
START     move.w        sr,d0          legge il registro di stato
    andi.w #$d8ff,d0              maschera per reg stato (stato utente, int abilitati)
    move.w        d0,sr           pone liv int a 000

    neg.b  OP2b
    move.bOP2b,d0
    add.b  d0,OP1b
    neg.w  OP2w
    move.w        OP2w,d1
    add.w  d1,OP1w
*Area Dati a partire dalla locazione $8300
    org    $8300
OP1b      dc.b    $85
```

```
OP2b    dc.b    $12
OP1w    dc.w    $9876
OP2w    dc.w    $5432
   END  START
```

2.144 NEG Negate

```
01000100 Size ea
dest=0-dest

X N Z V C
* * * ? ?
X è settato come C
N=1 se il risultato è negativo; 0 altrimenti
V=Dm·Rm,
C=Dm+Rm
Z=Z·¬Rm·....·¬R0

NEGX <ea>
size: B,W,L

-ea specifica l'operando destinazione.Sono permessi solo i modi
    di indirizzamento "data alterable".
*
```

2.145 NEGX Negate with extend

```
1101 R mode ea
dest=dest+sorg

* * * ? ?
V=Dm·Rm, C=Dm+Rm
V=Dm·Rm, C=Dm+Rm
Z=Z·¬Rm·....·¬RøADD        <ea>,Dn

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.146 NEGX Negate with extend

```
01000000 Size Ea
dest :=0-(dest)-X

X N Z V C
* * * * *
N=1 se il risultato è negativo (N:=Rn) , azzerato altrimenti
Z è invariato se il risultato è zero, azzerato altrimenti
V=1 se si è generato un overflow, azzerato altrimenti
C=1 se si è generato un riporto di sottrazione, azzerato altrimenti
X=C

NEG<Ea>

Size: B,W,L

esempio
```



```
*******************************************************************************
*PROGRAMMA DI PROVA DEL  CODICE NEGX
*Il programma seguente mostra l'utilizzo del codice NEGX  per realizzare la
*sottrazione OP1-OP2 per operandi di 64 bit. Il risultato è contenuto nei registri
*d2(long word  più significativa) e d3(long word meno significativa).
*
*N.B.        L'eventuale riporto di sottrazione sarà  il negato del flag C e in caso di
*    risultato negativo sarà N=1
*
*******************************************************************************
*Area Programma a partire dalla locazione $8000
    ORG   $8000
START    move.w        sr,d0            legge il registro di stato
    andi.w #$d8ff,d0              maschera per reg stato (stato utente, int abilitati)
    move.w        d0,sr            pone liv int a 000

    movea.l       #$d300,sp
    move.w        lung,d0
    divs    #4,d0
    lea     strnd,a1
    lea     mnd,a2
```

```
    movea.l     #result,a3      a3 punta al LSB del risultato
    adda.w lung,a1
    adda.w lung,a2
    subq   #1,d0
    move.w      #$0010,-(sp)    salva i codici di condizione sullo stack
ciclo       move.w      #$0010,ccr      set X
    negx.l  -(a1)
    move.l  (a1),d1
    move.l  -(a2),d2
    move    (sp)+,ccr               ripristina i codici di condizione del passo
precedente
    addx.l  d1,d2           esegue la generica sottrazione x-y come x+(-y)
    move    sr,-(sp)        salva i codici di condizione sullo stack
    move.l  d2,-(a3)
    dbra    d0,ciclo
    move    (sp)+,ccr               ripristina i codici di condizione dell'ultima
operazione

*Area Dati a partire dalla locazione $8300
    org     $8300
mnd         dc.l    $02345678,$98765432
strnd       dc.l    $10000000,$12345678
res ds.l    2               lunghezza risultato  = 8byte = 64bit
result      equ     *
lung        dc.w    8               lunghezza operandi = 8byte = 64bit
    END     START
```


2.147 NEGX Negate with extend

```
01000000 size ea
dest=0-(dest)-X

X N Z V C
* * ? ? ?
X è settato come C
N=1 se il risultato è negativo; 0 altrimenti
V=D_m·R_m
C=D_m+R_m
Z=Z·¬R_m·....·¬R_0

NEGX <ea>
size: B,W,L

EA può essere solo un “Data alterable addressing mode”
*



esempio (NEG NEGX)
*Questa subroutine fornisce un esempio di utilizzo dei codici  NEG  e NEGX. *Essa nega un n
espresso su D0 elementi di D1 word ciascuno.
*-------------------------------------------------------------------------
*NOTA 1 : Prima della negazione in multipla precisione il flag Z deve
*         esere 1. Dopo la negazione esso sarà 0 se il numero negato
*         è diverso da zero.
*NOTA 2 : Dopo la negazione flag V=0 indica che il risultato è corretto.
```

```
*         Se V=1 la negazione ha dato risultato errato. Ciò accade nel
*         solo caso in cui il numero da negare è -(b**n)/2 ove
*         b=2 e n è il numero di bit su cui è espresso il numero.


NEGMUL ORG $9200
       MOVE  D0,D2
       SUBQ #2,D0           modifichiamo D0 per il suo uso nel ciclo FOR
       MULU D1,D2
       MULU   #2,D2        ora D2 contiene il numero di byte su cui è espresso
*                  il numero da negare
       MOVEA.L SP,A0     usiamo A0 come stack-pointer
       ADD.L #DAT,A0     A0 punta al numero da negare
       ADD.L D2,A0       A0 punta alla locazione successiva
*                        all'elemento meno significativo del numero
       ORI #$04,CCR        poniamo  Z=1
       NEG -(A0)
FOR    NEGX -(A0)
       DBF D0,FOR
FINE   RTS

*Un esempio di programma chiamante di NEGMUL
MAIN   ORG $9300
* numero da negare
NUM3   DC.W  $FFFF       elemento più significativo
NUM2   DC.W  $FFFF             "
NUM1   DC.W  $FFFF           "
NUM0   DC.W  $FFFF       elemento meno significativo
LEN    DC.W  4      numero di elementi in cui è suddiviso il numero
SIZE   DC.W  1           numero di word che compongono un elemento
DAT    EQU 4               spiazzamento per la subrotine

VIA   ANDI #$DFFF,SR     passaggio a stato utente
      MOVE SIZE,D1
      MOVE LEN,D0
      MOVE NUM0,-(SP)    passiamo il numero da negare tramite lo stack
      MOVE NUM1,-(SP)
      MOVE NUM2,-(SP)
      MOVE NUM3,-(SP)
      JSR NEGMUL
      END VIA

*
```

2.148 NOP No operation

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.149 NOP No operation

```
0100111001110001
No operation
X N Z V C
- - - - -
NOP

size: Unsized


esempio:
BLOCCO1:    .
     .
     .
   JMP BLOCCO3
BLOCCO2:    .
     .
     .
BLOCCO3:  NOP
          END
```


2.150 NOP No Operation

```
0100111001110001

X N Z V C
- - - - -
Tutti i flag sono not affected

NOP
Unsized
*


esempio
```

2.151 NOT Logical complement

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.152 NOT Logical complement

```
01000110 Size Ea
dest=-(dest)

X N Z V C
- * * 0 0
N=1 se il risultato è negativo, azzerato altrimenti
Z=1 se il risultato è zero, azzerato altrimenti
V è sempre azzerato
C è sempre azzerato
X è invariato

NOT<Ea>

Size: B,W,L


esempio
```


2.153 NOT Logical complement

```
01000110 Size ea
dest=not(dest)

X N Z V C
- * * 0 0

NOT <ea>
size: B,W,L

-ea specifica l'operando destinazione.Sono permessi solo i modi
    di indirizzamento "data alterable".
*


esempio
*Questa routine simula un semisottrattore realizzato tramite un *semiaddizionatore.Essa cos
esempio di prova del codice NOT.
*Gli operandi sono passati in D0 (minuendo) e D1 (sottraendo).
*Il risultato e restituito in D1.
*Il numero di bit della aritmetica (8,16,32) è passato nei flag V C :
*  V  C   NBIT
*  1  -     16
*  0  1      8
*  0  0     32
```

```
*I flag restituiti al programma chiamante sono coerenti con quelli forniti
*dal codice SUB del 68000

HALF_SUB      ORG $8000
              BVS WRD
              BCS BYT
*LONG
              CMP.L #0,D1       nel caso in cui il sottraendo è 0
              BEQ ADDL          half_sub ed half_add sono equivalenti
              NOT.L D1          complemento a (b^n)-1 del sottraendo
              ADD.L #1,D1       complemento a b^n        "      "
ADDL          ADD.L D0,D1       sottrazione
              BRA SFL


WRD           CMP.W #0,D1
              BEQ ADDW
              NOT.W D1
              ADD.W #1,D1
ADDW          ADD.W D0,D1
              BRA SFL


BYT           CMP.B #0,D1
              BEQ ADDB
              NOT.B D1
              ADD.B #1,D1
ADDB          ADD.B D0,D1

*I bit Z,N ed O non hanno bisogno di correzioni
*Il resto del sottrattore è uguale al negato di
quello dell'addizionatore; (X=C)
SFL             BCS CX1                 se C e X sono 0
                ORI  #$11,CCR           vengono messi ad 1
                BRA FINE                se sono 1
CX1             ANDI  #$EE,CCR          vengono messi a 0

FINE            RTS

*Esempio di programma chiamante
MAIN          ORG $8200
              ANDI #$DFFF,SR            passaggio a stato utente

 *Chiamata per Size=byte
              MOVE.B #$33,D0
              MOVE.B #$32,D1
              ORI        #$01,CCR
              JSR HALF_SUB

*Chiamata per Size=Word
              MOVE.W #$3333,D0
              MOVE.W #$3334,D1
              ORI        #$02,CCR
              JSR HALF_SUB

*Chiamata per Size=Long
              MOVE.L #$10000000,D0
              MOVE.L #$00000001,D1
              ANDI        #$F6,CCR
              JSR HALF_SUB
```

```
                END MAIN

NOTA:
E’ stato riscontrato un errore di ASIM nel settaggio del bit
di overflow della addizione (vedi chiamata per Size=Long)
*
```

2.154 OR Inclusive or logical

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

OR Inclusive or logical

```
1000 REG op-mode ea

SORG or DEST -->DEST

- * * 0 0

N=1 se il bit più significativo del risultato è 1
Z=1 se il risultato è nullo

OR <ea>,Dn
OR Dn,<ea>

size: B,W,L


esempio :
*  Questo programma compatta due word in una long-word *
*  mettendo la prima word nella parte alta della long-word e la *
*  seconda nella parte bassa.  Seconda versione. *
*  E' un esempio di utilizzo  della OR .  *
*  Elaborazione: Gruppo 10 - 96/97

   ORG    $8000
B1 EQU    $EFA1
B2 EQU    $9F6C

START     move.w #b1,d1
   move.w #b2,d2
   swap   d1     * sposta la word nella parte alta del registro d1
   or.l   d2,d1
   END    START
```


2.155 OR Inclusive or logical

```
1000 R OpMode EffAddr
dest=dest OR sorg

- * * 0 0

OR <ea>,Dn
OR Dn,<ea>
size: B,W,L
```

2.156 ORI Inclusive or immediate

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.157 ORI Inclusive or immediate

```
00000000 size ea + 1,2 ext word
dest ← dest + sorg

X,N,Z,V,C
- * * 0 0
N=1 se MSB(*dest)=1
Z=1 se *dest=0

ORI#<data>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- diretto con registro indirizzo
- relativo
- relativo indicizzato
- immediato


esempio :

* Tale segmento di programma consente di settare il bit più significativo
degli
* elementi di posto pari o dispari di un vettore di long word .
* D7 deve contenere il bit di controllo per determinare i posti dispari o
pari .
*
   org    $8000
INIZIO    move.l #VETINI,A0    carica A0 per indirizzare gli elementi del
vettore
   btst.l #0,D7         testa il bit meno significativo del registro
   beq.s  PARI          se il bit è 0 si opera sugli elementi di posto pari
DISPARI   moveq.l      #1,D0         D0 ha l'indice del primo elemento di
posto dispari
   bra.s  LOOP
PARI      moveq.l      #0,D0         D0 ha l'indice del primo elemento di
posto pari
LOOP      cmp.b  NUMEL,D0      controlla se sono esauriti gli elementi del
vettore
   bge.s  FINE
   move.l D0,D1
   mulu.w #4,D1         D1 contiene lo spiazzamento dell'elemento da settare
   move.l 0(A0,D1),D2
   ori.l  #$80000000,D2 effettua il settaggio del bit più significativo
   move.l D2,0(A0,D1)   aggiorna il vettore in memoria
   addq.l #2,D0         aggiorna l'indice dell'elemento
```

```
   bra.s  LOOP
*
VETINI    org    $8040         indirizzo del primo elemento del vettore
VETTORE   dc.l   $004531A0,$0103F34A,$0005E781,$120056BC,$D4C6F002
NUMEL     dc.b   5             numero degli elementi del vettore
*
FINE      nop
   END    INIZIO
```

2.158 PEA Push effective address

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.159 PEA Push effective address

```
0100100001 Ea
-(SP)=dest

X N Z V C
- - - - -

Nessun flag di CC viene modificato

PEA<Ea>

Size: L


esempio
```



```
*LE SEGUENTI LINEE MOSTRANO L'UTILIZZO DEL CODICE PEA
*TALE CODICE RISULTA UTILE NEL PASSAGGIO DEI PARAMETRI PER
*INDIRIZZO, COME NEL CASO DEGLI ARRAY, QUANDO I PARAMETRI SONO
*SCAMBIATI TRAMITE STACK.PUO' ESSERE USATO IN COMBINAZIONE
*CON I CODICI LINK ED UNLK

*ESEMPIO:INIZIALIZZAZIONE DI UN ARRAY DI WORD
*L'ARRAY E' SCAMBIATO PER INDIRIZZO TRAMITE STK

       org $8500   *area dati

frst      equ 1
last      equ 5
size     equ 2
vect     ds.b ((last-frst+1)*size)
eavect equ vect-(frst*size)

       org $9000     *area stk
stk ds.w 40
stke equ *

      org $8100 *area subroutine
init       link a6,#0
```

```
        move.l par(a6),a1
        move.l #frst,d0
iloop   move.l d0,d2
        mulu.w #size,d2
        move.l d2,a2
        add.l a1,a2
        move.w #0,(a2)
        add.w #1,d0
        cmpi.l #last,d0
        ble iloop
        unlk a6

par equ 8
        rts

    org $8300 *area codice

start andi.w #$DFFF,sr
      move.l #stke,sp
      move.l sp,a6
      move.l #eavect,a0
      pea (a0)
      jsr init
      end start
```

2.160 PEA Push effective address


```
0100100001ea
SP@-=Destinazione

X N Z V C
-  -   -  -  -
Tutti i flag sono not affected

PEA <ea>
Size= L

ea - Specifica l'indirizzo che deve essere messo in testa allo stack
       Sono concessi solo control addressing modes
*


esempio
```

2.161 RESET Reset external devices

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.162 RESET Reset External Device

```
0100111001110000
RESET

X N Z V C
- - - - -
X N Z V C,=Not affected

size:
Esempio
```

2.163 RESET Reset External Devices

```
0100111001110000

X N Z V C
- - - - -
Tutti i flag sono not affected

RESET
Unsized
*


esempio
```

2.164 ROL Rotate left without extend

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.165 ROR Rotate rigth without extend

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.166 ROR Rotate rigth without extend

```
1110 count|Reg 0 size i/r 11 R (Register Shifts)
dest = dest >> count
1110 011 011 ea               (Memory Shifts)
M[ea] = M[ea] >> 1


X,N,Z,V,C
- * * 0 ?
C= D count-1

X,N,Z,V,C ( count == 0 )
- * * 0 0

RORDx,Dy
ROR#<data>,Dy
ROR<ea>

size: B,W,L ( Register Shifts )
size: W     ( Memory Shifts )


esempio
   ORG    $8000

* Esempio di utilizzo dell'istruzione ROR ( ROtate Right )
* Si mostra come sia possibile implementare l'operazione di mol-
* tiplicazione, unsigned, di operandi a 16 bit con risultato a 32
* bit utilizzando operazioni ai ADD di ROR e di LSR.
* L'esempio è strutturato come subroutine che effettua l'operazio-* ne
desiderata, a questo proposito si assume che in D0, nei 16   * LSBs vi sia il
moltiplicando e che in D1, nei 16 LSBs, vi sia il * moltiplicatore.
* La routine restituisce in D2 il risultato dell'operazione,      *
modificando consistentemente i vari FLAG.
* Per un descrizione dell'algoritomo vedi Wakerly pag.

start

   MOVE.W #$1234,d0
   MOVE.W #$1234,d1
   jsr    MULTIPLY
```

```
   MOVE.W #$FF34,d0
   MOVE.W #$F234,d1
   jsr    MULTIPLY
   STOP   #$80

MULTIPLY
   MOVE.L d3,-(a7)      Salvataggio sullo stack di D3
* utilizziamo il size L perché nell'attuale implementazione DBF   * che
decrementa il registro D3 lo decrementa a 32 bit => se uti- * lizzassimo il
size B o W senza "pulire" la parte restante del   * registro il ciclo
potrebbe non effettuare il numero di iterazio-* ni previsto
   MOVE.L #15,d3 D3 = loop count
   MOVE.L #0,d2  Inizializzazione D2 = risultato parziale

FORBTST.L #0,d0  testa il LSB del moltiplicando shiftato
   BEQ    NOADD
   ADD.W  D1,D2  somma il moltiplicando shiftato al
                      * risultato parziale
   BCS    CARSET
* il LSB del moltiplicatore shiftato era nullo
NOADD     ROR.L  #1,d2  * shift a destra del risultato
                            * parziale e salvataggio del suo
          BRA         LAB1          * LSB nel MSB di D2

CARSET    * l'addizione ha generato carry
   ROR.L  #1,d2  * shift a destra del risultato parziale e
                        * salvataggio del suo LSB nel MSB di D2
*  BSET.L        #31,D2 istruzione non funzionante per cui usiamo
   OR.W   #$8000,d2          pone il MSB del prodotto parziale ad 1

LAB1      LSR.W  #1,d0  Shift a destra del moltiplicatore
   DBF    d3,FOR

   MOVE.L (a7)+,d3      Ripristino di D3
   SWAP.W d2     Correzione del risultato
   RTS

   end start
```

2.167 ROXL Rotate left with extend

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.168 ROXL Rotate left with extend

```
1110 count|Reg 1 size i/r 10 R (Register Shifts)
1110 010 111 ea               (Memory Shifts)


X,N,Z,V,C
* * * 0 ?
C= D m-count+1

X,N,Z,V,C ( count == 0 )
- * * 0 ?
C= X

ROXL      Dx,Dy
ROXL      #<data>,Dy
ROXL      <ea>

size: B,W,L ( Register Shifts )
size: W     ( Memory Shifts )


* Esempio di utilizzo dell'istruzione ROXL ( ROtate with X Left )
* Si mostra come sia possibile implementare l'istruzione ROL per
* operandi di dimensionioni maggiori di una LONGWORD (LW).
* L'esempio è strutturato come subroutine che effettua l'operazio-* ne
desiderata, a questo proposito si assume che il dato su cui
* effettuare il ROL sia di 64-bit memorizzato in D0-D1 ( D0 LW
* più significativa ).
* Viene mostrato come effettuare il ROL di 1 bit e poi tale
* operazione viene utilizzata per effettuare il ROL di più bit

   ORG    $8000
start
* operazione di ROL generica
   MOVE.l #$2AAAAAAF,d0    d0 = 00101010101010101010101010101111
   MOVE.l #$7D6A80AF,d1    d1 = 01111101011010101000000010101111
   JSR    ROL1M
* questi dati servono per verificare la corretta modifica di Z
   MOVE.l #$80000000,d0   d0 = 10000000000000000000000000000000
   MOVE.l #$00000000,d1   d1 = 00000000000000000000000000000000
   JSR    ROL1M
```

```
   MOVE.l #$00000000,d0   d0 = 00000000000000000000000000000000
   MOVE.l #$01000000,d1   d1 = 00000001000000000000000000000000
   JSR    ROL1M
   STOP   #$80

ROL1M
   LSL.L  #1,d1         X = C = ex D1[31]
   BEQ          ZERO               Salta se la prima LW è = 0

* La prima LW è risultata essere non nulla
   ROXL.L #1,d0         X = C = ex D0[31]
   BCC.S  END1          salta se  C = 0
   BSET   #0,d1         D1[0] = C = 1
END1      ANDI.B #$FB,CCR      Z = 0  La prima LW era != 0
   BRA.S  ENDROL1M

* La prima LW del dato è risultata essere nulla
ZERO      ROXL.L #1,d0         X = C = ex D0[31]
   BCC.S  ENDROL1M      salta se  C = 0
   BSET   #0,d1         D1[0] = C = 1
* a questo punto bisogna modificare il flag Z in modo consistente * con la
dimensione del dato: infatti le singole operazioni di LSL * e ROXL modificano
i flag in modo consistente con le dimensioni  * BYTE WORD e LW, mentre noi
stiamo operando su un dato di 2 LW;
* i flag X C V N risultano già modificati correttamente per il         * dato
a 64 bit,mentre il flag Z potrebbe dare un'informazione    * sbagliata in
quanto esso indica solo se D0, cioè parte del dato, * è nullo. Un modo
semplice per settare correttamente Z è l'ag-                 * giunta
dell'istruzione
   ANDI.B #$FB,CCR      Z = 0
* in quanto se in D1[0] entra 1 certamente il numero è !=0
ENDROL1M
   RTS
   END    START
```

2.169 ROXR Rotate rigth with extend

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.170 ROXR Rotate rigth with extend

```
1110 count|Reg 0 size i/r 10 R (Register Shifts)
1110 010 011 ea               (Memory Shifts)


X,N,Z,V,C
* * * 0 ?
C= D count-1

X,N,Z,V,C ( counto == 0 )
- * * 0 ?
C= X

ROXR      Dx,Dy
ROXR      #<data>,Dy
ROXR      <ea>

size: B,W,L ( Register Shifts )
size: W     ( Memory Shifts )


* Esempio di utilizzo dell'istruzione ROXR ( ROtate with X Right )
* Si mostra come sia possibile implementare l'istruzione ROR per
* operandi di dimensioni maggiori di una LONGWORD (LW).
* L'esempio è strutturato come subroutine che effettua l'operazio-* ne
desiderata, a questo proposito si assume che il dato su cui
* effettuare il ROL sia di 64-bit memorizzato in D0-D1 ( D0 LW
* più significativa ).
* Viene mostrato come effettuare il ROR di 1 bit.

   ORG    $8000
start
* operazione di ROR generica
   MOVE.l #$2AAAAAAF,d0    d0 = 00101010101010101010101010101111
   MOVE.l #$7D6A80AF,d1    d1 = 01111101011010101000000010101111
   JSR    ROR1M
* questi dati servono per verificare la corretta modifica di Z
   MOVE.l #$00000000,d0   d0 = 10000000000000000000000000000000
   MOVE.l #$00000001,d1   d1 = 00000000000000000000000000000000
   JSR    ROR1M
   MOVE.l #$01000000,d0   d0 = 00000000000000000000000000000000
```

```
   MOVE.l #$00000000,d1   d1 = 00000001000000000000000000000000
   JSR    ROR1M
   STOP   #$80

ROR1M
   LSR.L  #1,d1         X = C = ex D1[0]
   BEQ    ZEROR         Salta se la prima LW è = 0

* La prima LW è risultata essere non nulla
   ROXR.L #1,d0         X = C = ex D0[0]
   BCC.S  ENDR          salta se  C = 0
   BSET.L #31,d1        D1[31] = C = 1
ENDR      ANDI.B #$FB,CCR      Z = 0  La prima LW era != 0
   BRA.S  ENDROR1M

* La prima LW del dato è risultata essere nulla
ZEROR     ROXR.L #1,d1         X = C = ex D0[31]
   BCC.S  ENDROR1M      salta se  C = 0
   BSET   #31,d1        D1[31] = C = 1
* a questo punto bisogna modificare il flag Z in modo consistente * con la
dimensione del dato: infatti le singole operazioni di LSL * e ROXL modificano
i flag in modo consistente con le dimensioni         * BYTE WORD e LW, mentre
noi stiamo operando su un dato di 2 LW;
* i flag X C V N risultano già modificati correttamente per il      * dato a
64 bit, mentre il flag Z potrebbe dare un'informazione   * sbagliata in
quanto esso indica solo se D0, cioè parte del dato, * è nullo. Un modo
semplice per settare correttamente Z è l’ag-   * giunta dell'istruzione

   ANDI.B #$FB,CCR      Z = 0

* in quanto se in D1[31] entra 1 certamente il numero è != 0
ENDROR1M
   rts

   END    start
```

2.171 RTE Return from exception

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.172 RTE Return from Exception

```
0100111001110011
SP@+=>SR
SP@+=>PC

X N Z V C
* * * * *
X N Z V C - modificati a secondo del contenuto della word sullo stack

RTE

size: Unsized


esempio:Vedi l'esempio del codice Trap
```


2.173 RTE Return from Exception - privileged instruction

```
0100111001110011
SP@+ -> SR
SP@+ -> PC

X N Z V C
*  *  *  * *
X,N,Z,V,C : set according to the content of the word on the stack


esempio
Usato in più esempi.
```

2.174 RTR Return and restore condition codes

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.175 RTR Return and Restore Condition Code

```
0100111001110111

SP@+=>CC
SP@+=>PC
RTR

X N Z V C
* * * * *
X,N,Z,C,V - Settati In Funzione Dell'operando

size: Unsized


esempio
```


2.176 RTR Return and Restore Condition Codes

```
0100111001110111
CC=SP@+ ; PC=SP@+

X N Z V C
* * * * *

Tutti i flag sono accordati secondo il contenuto della word
nello stack

RTR
Unsized
*


esempio
```

2.177 RTS Return from subroutine

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.178 RTS Return from Subroutine

```
0100111001110101
SP@+=>PC

X N Z V C
- - - - -

RTS
size: Unsized


esempio:Vedi l’esempio del codice CHK
```

2.179 RTS Return from Subroutine

```
0100111001110101
SP@+ -> PC

X N Z V C
- - - - -
Tutti i flag sono not affected

RTS
Unsized
*


esempio
Usato in più esempi.
```

2.180 SBCD Subtract decimal with extend

```
1101 R mode ea
dest=dest+sorg

* U ? U ?
C=prestito decimale
Z=Z·¬Rm·....·¬R0

N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.181 SBCD Subtract decimal with extend

```
1000 Rx 10000 R/M Ry
dest=dest-sorg

* U * U *
X=settato come il carry
Z=0 se il risultato e' diverso da 0. Altrimenti resta invariato.
C=1 se si verifica il borrow (decimale). Altrimenti e' 0.

SBCD      Dy,Dx
SBCD  -(Ay),-(Ax)
size: B


esempio
* Il programma esegue una DIVU in BCD tra due numeri (codificati in BCD)
* aventi la stessa dimensione (in byte), utilizzando il codice operativo SBCD
* con operandi del tipo indirizzo predecrementato.
* Il quoto è memorizzato nel registro D2 ed il resto (sempre BCD)
* e' nel registro D3.
* Il controllo di fine ciclo viene realizzato ispezionando il flag X (=C)
* per vedere se il resto è negativo in seguito ad un CPMI con lo zero.
* Elaborazione: Gruppo 10 - 96/97.

*Area Programma a partire dalla locazione $8000
   ORG    $8000
START     MOVEQ  #0,D2               Inizializza il quoto in D2
INIT      MOVE.L (A0),D3       Inizializza il resto in D3
   MOVE.L #FOP1,A0      In A0 indirizzo base 1^ addendo
   MOVE.L #FOP2,A1      In A1 indirizzo base 2^ addendo
   MOVE.L #DIST,D1      In D1 il valore della var. di cont. del ciclo SBCD
LOOP      SBCD   -(A1),-(A0)         Una sottrazione BCD per ogni coppia di
nibble
   DBEQ   D1,LOOP
   MOVE.W SR,D4  Sposta SR in D4 per fare il controllo con la maschera
   CMP    #$2711,D4     Controllo con la maschera
   BEQ    FINE
   ADDI   #1,D2         In D2 il conto dei cicli per DIV, e cioè il quoto
   BRA    INIT          Deve effettuare ancora almeno una SBCD
```

```
FINE      STOP   #$2000 Istruzione illecita in quanto privilegiata. Serve
*                       per fermare l'esecuzione del programma.
*-------------------------------------------------------------------------
*
*Area Dati a partire dalla locazione $8032
   ORG    $8032
OP1DC.B   $96,$22,$00,$17      Definizione del dividendo
FOP1      EQU    *                Fine 1^operando
OP2DC.B   $17,$99,$41,$44      Definizione del divisore
FOP2      EQU    *                Fine 2^operando
DIST      EQU    FOP2-FOP1-1   Calcolo del numero di cifre del divisore
   END    START
```

2.182 SBCD Subtract decimal with extend

```
1000 Rx 10000 R/M Ry
dest=dest-sorg

* U ? U ?
C=prestito decimale
Z=Z·¬R_m·....·¬R_0

N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

SBCD      Dn,Dn
SBCD      -(An),-(An)
size: B
```


Esempio: bcdbin.a68

```
*  Utilizzo del codice operativo: SBCD
*  Conversione di un numero di due cifre BCD in Binario.
*  Incrementa il registro D2, che conterrà il risultato, ogni volta
*  che viene decramentato di uno il numero da convertire.
*
   ORG    $8200
UNOEQU    1
NUMEQU    $25           Numero da convertire
START     MOVEQ  #UNO,D0
   MOVEQ  #NUM,D1
   CLR    D2            Azzera contatore
LOOP      SBCD   D0,D1
   ADDQ   #UNO,D2
   CMP    #0,D1
   BNE    LOOP
   STOP   #$2000
   END    START
```

2.183 Scc Set according to condition

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.184 SCC Set according to condition

```
0101 Condition 11 ea
if cc then 1's=>dest else 0's => dest

X N Z V C
- - - - -

Scc <ea>
size: B


ESEMPIO

Questa sobroutine effettua l'estenzione in segno su due byte di un
dato caricato nel byte piu' basso di una word puntata da A0

ORG      $8000 Indirizzo di partenza

ESSCC MOVE.W 1(A0),D0 Carica byte basso dato
      BTST   #7,D0    Controlla il segno del dato
      SNE    (A0)     Estendi in segno
      RTS

ORG   $9000        Area Dati
BUFFER  DC.W       128

MAIN
LEA.L BUFFER,A0   Carica in A0 l'indirizzo dato
JSR   ESSCC       Salto a sottoprogramma
END   MAIN
```

S_{CC} Poni byte a 1 su condizione

				Condition			Effective Address

Scc <ea>

operazione effettuata : if cc then poni a 1 destinazione else poni a zero destinazione.

Consentito il solo size di byte.

Non sortisce alcun effetto sul CCR (bit di flag).

Condizioni implementate :

		Test su			Test su
	0100 -	¬C		0011 -	C | Z
	0101 -	C		1101 -	(N&
	0111 -	Z		1011 -	N
	0001 -			0110 - Not	¬Z
	1100 -	(N&V) | (¬N&¬V)		1010 -	¬N
	1110 -	(N&V&¬Z)|(¬N&		0000 -	
	0010 - High	¬C&¬Z		1000 -	¬V
	1111 - Less	Z|(N&¬V)|(¬N&V		1001 -	V

Questa istuzione verifica i valori dei bit di stato : se la condizione è soddisfatta il byte destinazione viene caricato con $FF, altrimenti lo stesso byte viene azzerato.

L'istruzione Scc è particolarmente utile per memorizzare lo stato di un particolare codice condizione in attesa di verificarlo in seguito

2.185 STOP Load status register and stop

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.186 STOP Load Status Register and Stop

```
1100111001110010 Immediate data
Immediate data =>SR
STOP

X N Z V C
* * * * *
X N Z V C - Modificati a secondo dell'operando
size: unsized


esempio
```

2.187 STOP Load Status Register and Stop- privileged instruction

```
0100111001110010
 <Immediate Data>

Immediate Data -> SR; STOP

X N Z V C
* * * * *
X,N,Z,V,C : set according to the immediate operand

Imm. Field: Specifica il dato da caricare nel registro di stato
*


esempio
```

2.188 SUB Subtract binary

1101 R mode ea
dest=dest+sorg

* * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L

esempio

2.189 SUB Subtract binary

1001 R opmode ea

dest=dest-sorg

* * * * *

X=settato come il carry
N=1 se il risultato e' negativo. Nullo altrimenti.
Z=1 se il risultato e' nullo. Nullo altrimenti.
V=1 se e' generato OVERFLOW. Nullo altrimenti.
C=1 se e' generato un BORROW. Nullo altrimenti.

SUB<ea>,Dn
SUBDn,<ea>

size: B,W,L

esempio

*** VEDI ESEMPI ISTRUZIONI: DIVS & DIVU ***

2.190 SUB Subtract binary

1001 R OpMode EffAddr
dest=dest-sorg

* * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

SUB<ea>,Dn
SUBDn,<ea>
size: B,W,L

Esempio: **restr.a68**

* Utilizzo dei codici operativi:MOVEQ e SUB
* **Restoring**: effettua la divisione mediante sottrazioni successive.

```
*  Esegue D0/D1: 25/3
*
   ORG    $8000
DVD EQU    25
DVS EQU    3

START     MOVEQ  #0,D2         Azzera il registro D2 che funge da contatore.
   MOVEQ  #DVD,D0       D0 è il dividendo
   MOVEQ  #DVS,D1       D1 è il divisore

INIZIO CMPD1,D0         Confronto D1 e D0 e salto a FINE se dovessi
effettuare
*                       una divisione illecita (DVS>DVD), oppure se ho
finito.
   BLT    FINE
   SUB.L  D1,D0         Il registro D0 alla fine conterrà il resto.
   ADDQ.L #1,D2  Il registro D2 alla fine conterrà il quoto.
   JMP    INIZIO
FINE      NOP                 No operation nel caso in cui DVS>DVD, oppure
se ho        *                       finito di sottrarre.
   STOP   #$2000
   END    START
```

2.191 SUBA Subtract address

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.192 SUBI Subtract immediate

1101 R mode ea
dest=dest+sorg

* * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L

esempio

2.193 SUBI Subtract immediate

00000100 size ea + 1,2 ext word
dest ← *dest-imm

X,N,Z,V,C
* * * ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

SUBI #<imm>,<ea>
size: B,W,L

Modi di indirizzamento non permessi :
- Immediate
- Relative
- Relative indexed

esempio :

```
* Questo segmento di programma estrae una colonna da una matrice di
long word e * la trasferisce in una certa area di memoria con
l'ordine invertito per gli      * elementi.
* D0 contiene l'indice della colonna e A1 contiene l'indirizzo di
partenza dell' * area di memoria in cui deve essere trasferita la
colonna.
*
   org    $8000
START    subq.l #1,D0          individua l'indirizzo di partenza
   mulu.w #4,D0          dell' ultimo elemento della colonna
   addi.l #TABINI+3*16,D0  selezionata;
   moveq.l      #3,D1          inizializza il contatore;
LOOP     move.l D0,A0          per utilizzare l'indirizzamento
indiretto;
   move.l (A0),(A1)+    trasferisce l'elemento e aggiorna il
puntatore;
   subi.l #16,D0              seleziona il successivo elemento della
colonna;
   dbf    D1,LOOP
*
TABINI   org    $8020          inizializza l'area della tabella
```

```
*
RIGA1    dc.l   $004531A0,$0103F34A,$0005E781,$120056BC
RIGA2    dc.l   $17C3B001,$A9D361C0,$01050641,$F20D501C
RIGA3    dc.l   $1045F380,$D010520A,$0005E000,$AB00D12C
RIGA4    dc.l   $030581A0,$0AB3F51A,$0503D081,$1263A0BC
    end     START
```

2.194 Note:

- Il modo di indirizzamento ‘Address-direct-register’ non è permesso, contrariamente a quanto è scritto sia sul manuale (‘Only data alterable addressing modes are allowed’) sia sul Wakerly (‘a_dst’).

2.195 SUBQ Subtract quick

```
1101 R mode ea
dest=dest+sorg

* * * ? ?
```
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

```
ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.196 SUBQ Subtract quick

```
0101 data 1 size ea
dest<=*dest-sorg

X N Z V C
* * * ? ?
```
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

```
SUBQ #data,<ea>
size: B,W,L


ESEMPIO

Questa subroutine permette il calcolo dei bit alti di un dato passato
in D0,restituisce il risultato in d1.

ORG     $8000   indirizzo di partenza

ESSUBQ  MOVE.L #32,D0 Inizializza a 32 d1
        MOVE.L #32,D2 Inizializza il contatore
LAB1    ROL    #1,D0  Shifta a sinista D0
        BSC    LAB2   Controlla il primo bit di D0
        SUBQ   #1,D1
LAB2    SUBQ   #1,D2
        BNE    LAB1
        RTS           Ritorno al chiamante

MAIN
        MOVE.L #$10000003,D0 Metti in D0 il dato
        JSR    ESSUBQ Salto a sottoprogramma
        END    MAIN
```


SUBQ Sottrazione

				Data			Effective Address

SUBQ #<data>,<ea>

Sottrae al valore contenuto in dst un valore immediato compreso tra 1 ed 8. La destinazione può essere un indirizzo effettivo alterabile. Sono permessi i size BWL tranne che per gli An dove non è permesso utilizzare il size Byte.

X	N	Z	V	C
*	*	*	?	?

X è settato allo stesso modo di C
N è settato se il risultato è negativo
Z è settato se il risultato è zero
$V = \neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C = S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$

Sm bit più significativo del sorgente
Dm bit più significativo della destinazione
Rm bit più significativo del risultato

L'istruzione SUBQ torna utile laddove serve decrementare un puntatore ad una word (2 unità) o ad una long word (4 unità).

Differenze con SUBI: è più corta ed è ad essa preferita quando si ha a che fare con parole in doppia precisione ; può usare un An come destinazione ed, in tal caso, si comporta come un SUBA con un dato immediato (sono consentite come dimensioni, in questo caso, soltanto W e L e i bit di stato restano intatti).

2.197 SUBX Subtract with extend

1101 R mode ea
dest=dest+sorg

* * ? ? ?
$V=\neg S_m \cdot D_m \cdot \neg R_m + S_m \cdot \neg D_m \cdot R_m$
$C=S_m \cdot \neg D_m + R_m \cdot \neg D_m + S_m \cdot R_m$
$Z=Z \cdot \neg R_m \cdot \ldots \cdot \neg R_0$

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L

esempio

2.198 SUBX Subtract with extend

1001 Rx 1 size 00 R/M Ry

dest=dest-sorg-x

* * * * *

X=settato come il carry
N=1 se il risultato e' negativo. Nullo altrimenti.
Z=1 se il risultato e' nullo. Inalterato altrimenti.
V=1 se e' generato OVERFLOW. Nullo altrimenti.
C=1 se e' generato un riporto. Nullo altrimenti.

SUBX Dx,Dy
SUBX -(Ay),-(Ax)

size: B,W,L

esempio

```
*  Esempio d'uso di SUBX per la sottrazione in multiprecisione TOT=num0-num1
*  num0 = 3.459.327.408.245.962.467 e quindi occupa due longword, cosi' come
*  num1 = 1.153.390.864.253.429.473
*  Il risultato, solo per una verifica veloce anche dei bit di flag,
*  e' salvato anche nei registri D0,D4
*  NB: Data la scelta precisa dei due numeri num0 e num1,
*  si verifichera' il BORROW durante la SUBX.
*  Di conseguenza la parte LOW del risultato sara' un num negativo.
*  E' quindi logico che spostando tale parte nel registro D4,
*  verra' settato il bit N dell'SR.
*  Elaborazione: Gruppo 10 - 96/97

   ORG    $8000         *
START     MOVEA.L       #0,A0         *  AZZERA IL REGISTO  A0
   MOVEA.L       A0,A1         *  AZZERA IL REGISTRO A1
NUM0H     EQU    $3001FFF3     *  SPLITTA L'INTERO NUMERO IN DUE LONGW: NUM0H
(HIGH)
NUM0L     EQU    $1245F6E3     *  E NUM0L (LOW)
NUM1H     EQU    $1001AAE1     *  STESSA OPERAZIONE PER NUM1
NUM1L     EQU    $4F01A6E1
```

```
N0 DS.L   2      *  RISERVA LO SPAZIO PER ALLOCARE IN MEMORIA I DUE NUMERI
N1 DS.L   2
   MOVE.L #NUM0H,N1     * PONE NUM0H IN MEMORIA
   MOVE.L #NUM0L,N1+4   * PONE NUM0L IN MEMORIA, DOPO NUM0H
                              * (QUINDI 4 BYTES DOPO NUM0H)
   MOVE.L #NUM1H,N0     * STESSA OPERAZIONE PER NUM2
   MOVE.L #NUM1L,N0+4   *
   LEA.L  N1+8,A0             * CARICA IN A0 L'INDIRIZZO DI NUM0
   LEA.L  N0+8,A1             * CARICA IN A1 L'INDIRIZZO DI NUM1
   MOVE.W  #0,CCR             * AZZERA LO SR
   SUBX.L  -(A1),-(A0)  * EFFETTUA LA SOTTRAZIONE TRA NUM0L E NUM1L
                              * IN MODO PREDECREMENTATO
   SUBX.L  -(A1),-(A0)  * SOTTRAE LE PARTI HIGH DEI DUE NUMERI COMPRESO
                              * L'EVENTUARE BORROW DELLA SOTTRAZIONE
PRECEDENTE
   MOVE.L (A0)+,D0      * PONE LA PARTE HIGH DEL RISULTATO IN D0
   MOVE.L (A0)+,D4      * PONE LA PARTE LOW DEL RISULTATO IN D4
                              * (N=1 SE C'E' STATO BORROW)
   END    START               * FINE.
```

2.199 SUBX Subtract with extend

```
1001 Rdest 1 Size 00 R/M Rsrc
dest=dest-sorg-X

* * ? ? ?
V=¬S_m·D_m·¬R_m+S_m·¬D_m·R_m
C=S_m·¬D_m+R_m·¬D_m+S_m·R_m
Z=Z·¬R_m·....·¬R_0

SUBX      Dy,Dx
SUBX      -(Ay),-(Ax)
size: B,W,L
```


Esempio: subx.a68

```
*Utilizzo del codice operativo:SUBX
*Il programma seguente mostra l'utilizzo del codice SUBX con l'indirizzamento
*con predecremento. Il minuendo e' il numero op1=67381185, il sottraendo
*op2=22419153.
*La differenza dei due numeri è 44F68032, e viene calcolata, tenendo conto
del prestito memorizzato nei flag X e C, componendo la sottrazione parziale
delle *coppie di cifre decimali contenute in ciascuno dei 4 byte in cui
ciascun numero *e'memorizzato.
*La differenza è posta nelle stesse locazioni di op1 e nel registro D2.

*Area Programma a partire dalla locazione $8000
   ORG    $8000
START     MOVE.L #EOP1,A0            In A0 indirizzo base 1^ addendo
   MOVE.L #EOP2,A1            In A1 indirizzo base 2^ addendo
   MOVE.L #COUNT,D1           In D1 il valore della variabile di conteggio *
                                    del ciclo

LOOP      SUBX.B -(A1),-(A0)
   DBEQ   D1,LOOP                   Quando D1 vale 0 ho finito di
sottrarre,           *                                        altrimenti
torno a LOOP
```

```
   MOVE.L (A0),D2
   STOP   #$2000

*Area Dati a partire dalla locazione $8300
   ORG    $8300
OP1DC.B   $67,$38,$11,$85
EOP1       EQU    *
OP2DC.B   $22,$41,$91,$53
EOP2       EQU    *
COUNT      EQU    EOP2-EOP1-1                Inizializza contatore
   END    START
```

2.200 SWAP Swap register halves

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.201 SWAP Swap register halves

```
0100100001000 Reg
Reg[31:16]   Reg[15:0]

X N Z V C
- * * 0 0

N=1 se il bit più significativo del risultato (a 32 Bit) è alto
Z=1 il risultato è zero, azzerato altrimenti
V è sempre azzerato
C è sempre azzerato
X è invariato

SWAP      Dn

Size: W


esempio
```



```
*******************************************************************************
*PROGRAMMA DI PROVA DEL  CODICE SWAP
*Il programma seguente mostra l'utilizzo del codice SWAP  per lo scambio delle
*word di un registro per contare il numero di bit alti in una long-word utilizzando
*una procedura per contare il numero di bit alti in una word.
*******************************************************************************
*Area Programma a partire dalla locazione $8000
    ORG   $8000
START     move.w        sr,d0          legge il registro di stato
    andi.w #$d8ff,d0              maschera per reg stato (stato utente, int abilitati)
    move.w        d0,sr           pone liv int a 000
    move.l #$0d300,sp             inizializza stack pointer

    move.l lword,D0
           jsr      cntword
           swap     d0
    move.bD1,nHlword
    jsr       cntword
```

```
    add.b   D1,nHlword
    stop      #$FF00

cntword     lea       masks,A0
*
*
*Conta il numero di bit alti in una word. Riceve la word in D0
*e fornisce il risultato in D1
*
*
            clr.b     D1
iciclo      move.w          (A0)+,D2
            and.w   D0,D2
            beq.s   continua
            add.b   #1,D1
continua    cmp.w #maske,A0
            bne.s   iciclo
            rts

    org     $9000
lword       dc.l      $8421FF01
nHlword     dc.b      0
masks       dc.w
    $8000,$4000,$2000,$1000,$800,$400,$200,$100,$80,$40,$20,$10,$8,$4,$2,$1
maske       equ       masks+32
    end     START
```


2.202 SWAP Swap Register Halves

```
0100100001000 reg
Registro [31:16] <-> Registro [15:0]

X N Z V C
- * * 0 0

N : Alto se dopo lo swap il bit più significativo del registro è 1. Basso altrimenti.
Z : Alto se il risultato dello swap è zero. Basso altrimenti.

SWAP Dn
Size= W

reg- Specifica il data register da swappare
*



esempio
*Questo esempio è un semplice test del codice
     ORG $9200
VIA  ANDI #$DFFF,SR
     MOVE.L #$0000FFFF,D0

*    MOVE.L #$0000FFFF,A0
*    SWAP A0
*Giustamente l'assemblatore da errore se si cerca di
*swappare un registro indirizzo
```

```
      ORI.B #$1F,CCR   viene settato il ccr per verificare la
*                      coerenza nel settaggio dei bit
      SWAP D0
      END VIA

Il CCR risulta settato correttamente
*
```

2.203 TAS Test and set operand

```
1101 R mode ea
dest=dest+sorg

- * * 0 0
ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.204 TAS Test and set operand

```
0100101011 Ea
(dest) Testato   CC ;1  bit#7 di dest

X N Z V C
- * * 0 0
N=1 se il bit più significativo dell'operando è alto, azzerato altrimenti
Z=1 se l'operando è zero, azzerato altrimenti
V è sempre azzerato
C è sempre azzerato
X è invariato

TAS<Ea>

Size: B


esempio
```



```
* Il codice TAS (Test And Set) e' particolarmente utile nel caso in cui piu' processori
* condividono un blocco di memoria in lettura/scrittura.
* Le seguenti linee ne mostrano un semplice utilizzo. Si suppone che piu' processori
* accedono al blocco di memoria comune COMN e la sincronizzazione tra i
processori
* e' regolata dal semaforo LOCK che indica quando l'area puo' essere acceduta in
quanto
* nessun processore la sta' utilizzando.
* Quando un processore deve utilizzarla mediante il codice TAS testa il semaforo e
cerca
* di impegnarlo. Se il semaforo era zero si puo' accedere a COMN. Al termine
dell'accesso
* deve essere sbloccato il semaforo resettandolo.

* Nell'esempio si suppone semplicemente che il processore voglia inizializzare a zero
* l'area COMN.

   org $8200 *area stack

stk ds.w 20
stke equ *

   org $8400 *area dati
```

```
lock ds.b 1 * 1 byte per il semaforo

  ds.w 0   *allinea PLC

frst equ 0
last equ 5
size equ 2
comn ds.b (last-frst+1)*size * 6 word in comune
eacomn equ comn-(frst*size) *effective address di comn

   org $8600 * area subroutine

*inizializza comn: l'array e' scambiato passando l'indirizzo base in A0

init     move.w #frst,d0  *indice del primo
iloop  move.l  d0,d1
         movea.l a0,a1
         mulu.w #size,d1 * indirizzo dell'elemento d0 rispetto ad a0
         adda.l d1,a1        * a1 ora contiene l'indirizzo effettivo dell'elemento
         move.w #1,(a1)
         add.w #1,d0
         cmpi.l #last,d0
         ble iloop
         clr.b lock  *sblocca il semaforo
         rts

   org $8800 *area main program

main    andi.w #$DFFF,sr *stato utente
          move.l #stke,sp
          clr.b lock *istruzione fittizia serve a testare TAS
          move.l #eacomn,a0
waitup  tas lock          *attesa attiva: attende che sia libera l'area comn
           bne waitup
           jsr init
           end main
```


2.205 TAS **Test and set an operand**

```
0100101011 <ea>
Tested -> CC
1 -> #7 of Destination (size: byte)

X N Z V C
- * * 0 0
set N if  MSB of the operand is set.Cleared otherwise
set Z if the operand is zero.Cleared otherwise

TAS <ea>

EA può essere solo un “Data alterable addressing mode”



esempio
```

2.206 TRAP trap

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.207 TRAP Trap

```
010011100100 vector

PC=>SSP@-
SR=>SSP#-
(VECTOR)=>PC

X N Z V C
- - - - -
TRAP #<VECTOR>

size: UNSIZED


esempio
```



```
*Il programma seguente mostra l'utilizzo del codice TRAP
*Il programma chiede la lettura di una linea attraverso una tastiera
*e la manda modificata ,sul video di un'altro terminale.
*vengono utilizzate due interruzioni software parametrizzate,una per la lettura
*ed un'altra per la scrittura.


    org     $8000  start at address $8000
start       move          $0000,sr
          movea.l  #ter,a2
          lea.l     quest,a1
    trap    #2                    scrive quest sul terminale 1
    movea.l       #ter,a2
    trap    #3                    legge il nome dal terminale 1
    movea.l       #ter2,a2
    lea.l   ans,a1
    trap    #2                    scrive ans sul terminale 2
    movea.l       #ter2,a2
    movea.l       #mem,a1
    trap    #2                    scrive il nome sul terminale 2
    stop    #$2000        istruzione privilegiata illecita, genera
*                 eccezione a 8 a $20
```

```
    org     $8300
ter     equ     $d000
ter2        equ     $d00a
mem         equ     $9000
quest   dc.b        'dimmi il tuo nome: ',0
ans         dc.b    'il suo nome e: ',0

    org     $8500   isr per la gestione della exception
    stop    #$a000


    org             $8600       isr  per la gestione della trap 2
    movea.l  a2,a0
            add.l       #1,a2
    move.b          #$30,(a2)  setta il registro di controllo del terminale
output      move.b          (a1)+,d0
    cmp     #0,d0    se il carattere da inviare è 0 termina

    beq     fine
    move.b  d0,(a0)          manda il ccarattere d0 a video
    bcc     output
fine        rte


    org             $8900       isr  per la gestione della trap 3
    movea.l  a2,a0
            add.l       #1,a2
    move.b          #$30,(a2)       setta il registro di controllo
cont        move.b(a2),d1
    and.b   #$80,d1                 se il bit di satto e' alto significa che ho inserito una
striga
    beq     cont
    movea.l  #mem,a1
ini  move.b(a0),d0          memorizza il carattere dal buffer
    move.bd0,(a1)+
    cmp     #13,d0
    bne  ini
            rte

    end     start
```


2.208 TRAP

```
010011100100 vector
push(PC); push(SR); PC=vector

X N Z V C
- - - - -
Tutti i flag sono non affected

TRAP #<vector>
size: unsized
*
```

esempio

2.209 TRAPV Trap on overflow

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.210 TRAPV Trap on Overflow

```
0100111001110110

if V then Trap

X N Z V C
- - - - -

TRAPV

size: Unsized


esempio:
```



```
*il programma seguente mostra l'utilizzo del codice trapv
*la somma provoca un overflow e il codice trapv manda in eccezione il
*processore che continua l'esecuzione dal'indirizzo caricato nel vettore delle
interruzioni
*all'indirizzo $1c.in questo esempio il programma manda a video una segnalazione
*di overflow

    org     $8000  start at address $8000
start       move     $0000,sr
          move.w  xx,d1
          add.w    sum,d1
    trapv
    stop    #$2000          istruzione privilegiata illecita, genera
*                   eccezione a 8 a $20

    org     $8300
xx  dc.w    $e500
sum         dc.w    $9000
ter       equ       $d000
err       dc.b   'error: overflow'

    org     $8500    isr per la gestione della exception
    stop    #$a000
```

```
    org             $8600       isr  per la gestione della exception di overflow
    movea.l         #ter,a2
          movea.l a2,a0
          add.l     #1,a2
    move.b          #$30,(a2)
    lea.l     err,a1
output      move.b          (a1)+,d0
    cmp     #0,d0
    beq     fine
    move.b  d0,(a0)
    bcc     output
fine        rte

    end     start
```

2.211 TRAPV Trap on Overflow

```
0100 1110 0111 0110
if V then TRAP

X N Z V C
- - - - -

Tutti i flag sono not affected

TRAP
Unsized
```



```
esempio
*Questa ISR stampa a video un messaggio di errore dovuto ad un overflow.
*Si suppone che TERDAT e TERCRT siano gli indirizzi dei due porti del
*dispositivo TERMINAL.

                ORG $9500
                MOVEA.L #TERDAT,A0
                MOVEA.L #TERCRT,A1
                MOVE.B #%00101100,(A1)   inizializzazione del terminale
                LEA MES,A2
ciclo           MOVE.B (A2)+,D0          stampa il messaggio di errore
                CMP.B #0,D0
                BEQ fine
                MOVE.B D0,(A0)
                JMP ciclo
fine            RTE
MES             DC.B 'Errore: Overflow!',0

MAIN            ORG $8000
TERDAT          EQU   $2000
TERCRT          EQU   $2001

VIA             ORI  #$02,CCR            settiamo il bit di overflow
                TRAPV
                END VIA

Un esempio di utilizzo di questa routine può essere l'esempio esplicativo di NEG e NEGX, do
essere usata per segnalare
```

a video l’errore dovuto all’overflow
*

2.212 TST Test an operand

```
1101 R mode ea
dest=dest+sorg

- * * 0 0

ADD<ea>,Dn
ADDDn,<ea>
size: B,W,L


esempio
```

2.213 TST Test an operand

```
01001010 Size Ea
(dest)—Testato   CC

X N Z V C
- * * 0 0
N=1 se l’operando è negativo, azzerato altrimenti
Z=1 se l’operando è zero, azzerato altrimenti
V è sempre azzerato
C è sempre azzerato
X è invariato

TST<Ea>

Size: B,W,L


esempio
```



```
*LE SEGUENTI LINEE MOSTRANO L'UTILIZZO DEL CODICE TST.
*ESSO PUO' RISULTARE UTILE PER VERIFICARE CONDIZIONI
SULL'OPERANDO
*ED EFFETTUARE DI CONSEGUENZA OPPORTUNE ISTRUZIONI DI SALTO
*PER EVITARE AD ESEMPIO OPERAZIONI ILLECITE COME UNA
*DIVISIONE PER ZERO.

* ESEMPIO

   ORG $8100
        dvsrOK: dc.w $8080
        dvsrNOTOK:dc.w $0000

   ORG $8000

   START: move.l #$12345678,d0
            move.w dvsrOK,d1
            tst.w d1            * BIT N=1 BIT Z=0
            beq skip
            divu.w d1,d0      *QUOZIENTE IN D0.L RESTO IN D0.H
            move.w dvsrNOTOK,d1
            tst.w d1            * BIT N=0 BIT Z=1
            beq skip
            divu.w d1,d0
```

```
SKIP:   nop
        end start

* SI FA PRESENTE CHE LE ISTRUZIONI DI TST PER VERIFICARE SE IL
DIVISORE
* E' ZERO NEL CASO SPECIFICO DELL'ESEMPIO POTEVANO ESSERE
OMESSE
* DATO CHE L'ISTRUZIONE DI MOVE SETTA OPPORTUNAMENTE I FLAG
```

2.214 TST Test an operand

```
01001010 size <ea>
Tested -> CC

X N Z V C
- * * 0 0
set N if operand is negative.Cleared otherwise
set Z if operand is zero.Cleared otherwise

TST <ea>
size: B,W,L

EA può essere solo un “Data alterable addressing mode”
*


esempio
```

2.215 UNLK Unlink

```
1101 R mode ea
dest=dest+sorg

* * * * *
N=1 se dest<0
Z=1 se dest=0
V=1 se si genera overflow
C,X=1 se si genera riporto (decimale)

ADD<ea>,Dn
ADD Dn,<ea>
size: B,W,L


esempio
```

2.216 UNLK Unlink

```
0100111001011 Reg
An SP ;—SP@+     An

X N Z V C
- - - - -

Nessun flag di CC viene modificato


UNLK      An

Size: Unsized


esempio
```

```
* LE SEGUENTI LINEE DI PROGRAMMA MOSTRANO L'UTILIZZO
* DEI CODICI LINK E UNLK.
* TALI CODICI SONO UTILI PER L'ALLOCAZIONE DINAMICA
* DEI PARAMETRI SULLO STACK.iL REGISTRO A6 SARA'
* UTILIZZATO COME FP.

    ORG $8000       *AREA DATI
frst  equ 0
last equ 4
size equ 2
num dc.w  1,2,5,4,3
eanum equ num-(frst*size)

   ORG $9000     *AREA STACK
stk ds.w 40
stke equ *

  ORG $8100 *AREA SUBROUTINE
* l'array contiene i valori 1 2 5 4 3
* la routine cerca il max e la sua posizione lasiando l'outpyt sullo stk
```

```
cercamax   link a6,#0  *salva il fp,aggiorna fp e non alloca var locali
           movea.w  indnum(a6),a0
           move.w #frst,d0
           move.w d0,d1
           mulu.w #size,d1
           move.w d1,a1
           adda.w a0,a1
           move.w #frst,d2  *si assume che il max sia il primo
           move.w (a1),d3
iloop      add.w #1,d0        *si incrementa l'indice d0
           move d0,d1
           mulu.w #size,d1
           move.w d1,a1
           adda.w  a0,a1      *a1 punta al successivo
           cmp.w (a1),d3      *confronto col max corrente
           bge skip
           move.w (a1),d3     *a1 punta al nuovo max
           move.w d0,d2       *l'indice d0 va salvato in pos
skip       cmpi.w #last,d0
           ble iloop        *a fine ciclo d2 e d3 contengono il max e la
                             *sua posizione
           move.w d2,pos(a6)
           move.w d3,max(a6)
           unlk a6  *ripristina A6

indnum equ 8
pos     equ 10
max     equ 12

           rts

* area programma
   ORG $8400
start  andi.w #$DFFF,sr  *pone il processore in stato utente
       movea.l  #stke,sp
       movea.l  sp,a6       * a6 frame pointer
       adda.w #-4,sp        *riserva area parametri di output sullo stk
       move.w #eanum,a0
       move.w a0,-(sp)     *passa l'array per indirizzo
       jsr cercamax
       move.w -(a6),d0     *max
       move.w -(a6),d1     *pos del max
       end start
```


2.217 UNLK Unlink

```
01001110011 Register
SP=An; POP(An)

X N Z V C
- - - - -

UNLK An
```

```
size: unsized

-Register:     specifica il registro indirizzo attraverso il quale
               viene realizzato l' unlink
*


esempio  (LINK-UNLK)
NSUB    ORG $9200
*Questo esempio è pensato per mostrare l'utilizzo dei codici
*LINK e UNLINK.
*-----------------------------------------------------------------
*Subroutine ricorsiva che nega un numero espresso su N word.
*Il numero da negare, N e i flags  sono passati nello stack.
*Il numero negato e i flag risultato vengono restituiti ancora nello stack.

      LINK A6,#0
      MOVE.W LEN(A6),D0       preleva dallo stack il numero di word
                              che la subroutine deve negare
      CMP.L #1,D0             se il numero di word da negare è  1 la
*                             subroutine provvede alla sua negazione
      BEQ unaw

*Abbiamo più di una (n) word da negare:
*la routine richiama se stessa con n-1 word da negare
      SUB.L #1,D0
      MOVE.L D0,D1            D1 contiene il numero di word (n-1) da
*                             passare alla nuova chiamata della subroutine
      MULU.W #2,D1
ciclo MOVE.W DAT(A6,D1),-(SP) passa le n-1 word da negare
      SUB.L  #2,D1
      CMP.L #2,D1
      BGE ciclo
      MOVE.W FLAG(A6),-(SP)   passa i flags correnti
      MOVE.W D0,-(SP)         passa il numero di word da negare
      JSR NSUB

* Il numero di word da negare è 1
*Preleva dallo stack i risultati delle chiamate precedenti.
unaw  MOVE.W LEN(A6),D1
      SUB.L #1,D1
      BEQ nega
      ADDA.L #2,SP           rilascia  la lunghezza del dato(input)
      MOVE.W (SP)+,FLAG(A6)  preleva i flag precedenti
      MOVE.L #2,D2           spiazzamento per puntare alle word già negate
cicl1 MOVE.W (SP)+,DAT(A6,D2)copia nella propria area dati le word già
*                            negate dalla precedente chiamata della subroutine
      ADD.W #2,D2
      SUB.W #1,D1
      BNE cicl1

*Viene eseguita la negazione
nega      MOVE FLAG(A6),CCR
          NEGX.W DAT(A6)
          MOVE.W SR,FLAG(A6)
RET       UNLK A6
          RTS


*-----------------------------------------------------------------------
```

```
       ORG $9500
DATOL  DC.L $00880077
DATOH  DC.L  $00AA0099         numero da negare su 4 WORD
N           DC.W 4             numero di word su cui è espresso il dato

* Spiazzamenti rispetto al FP
LEN       EQU 8                numero di word su cui è espresso il dato
FLAG      EQU 10               word in cui è memorizzato il CCR corrente
DAT       EQU 12               dato da negare

MAIN      ANDI     #$DFFF,SR       passaggio a stato utente
          MOVEA.L #$00000000,A6    inizializzazione FP
*                                 (solo a scopo di riferimento)
*Caricamento dei parametri nello stack
          MOVE.L   DATOL,-(SP)
          MOVE.L   DATOH,-(SP)
          MOVE.W #$0004,-(SP)  poniamo nello stack i valori iniziali dei flag:
*                              il  flag X (riporto entrante) è 0;
*                              il flag Z è 1
          MOVE.W   N,-(SP)
          JSR NSUB             chiama la subroutine
          END MAIN
*
```

2.217.1 Programmazione in assembler

In questo paragrafo saranno presentati alcuni esempi in linguaggio assembler allo scopo di mostrare le principali tecniche di programmazione.

Perché e quando sviluppare un programma in assembler?

La programmazione di una applicazione può essere effettuata ricorrendo a differenti linguaggi, ad alto o più basso livello. La potenza espressiva delle primitive offerte dai vari linguaggi gioca un ruolo fondamentale relativamente al tempo di sviluppo ed all'efficienza del codice generato. E' noto che un linguaggio ad alto livello (HLL) consente di ridurre i tempi di sviluppo ma, non avendo primitive in grado di controllare direttamente tutte le risorse hw di un processore, rende difficoltasa (o talvolta impossibile) lo sviluppo di programmi di sistema. Il codice macchina generato, essendo prodotto da un compilatore, può risultare non efficiente in talune parti critiche. Un linguaggio assembler (LLL), al contrario, possedendo le primitive per il completo controllo del processore, consente , se ben impiegato, la scrittura di codici particolarmente efficiente. Ma proprio il basso livello delle primitive fa si che i tempi di sviluppo e manutenzione di un programma risultino elevati rispetto allo sviluppo con HLL.

Non tutti le istruzioni di un programma hanno la stessa probabilità di essere eseguite

Soluzione mista HLL+assembler per la scrittura delle sole parti critiche. Ricordarsi che l'uso di compilatori con ottimizzazione del codice prodotto consente di generare codici, in taluni casi, più efficienti di quelli prodotti a mano.

Portabilità del codice possibile con HLL e non con assembler.

2.217.2 Esempi di traduzione in assembler di costrutti di controllo flusso ad alto livello

2.217.2.1.1 Costrutto If

```
if (C) then S1 else S2

Bcc(NOT C) et1
S1
   BRA    et2
et1S2
et2...
```

2.217.2.1.2 Costrutto Repeat

```
repeat S until C

et1S
   Bcc(NOT C)    et1
   ...
```

2.217.2.1.3 Costrutto While

```
while (C) do S
BRA
et1S
et2Bcc(C) et1
```

2.217.2.1.4 Costrutto For

```
for i:=inic to finec step k do S

i:=inic;
temp:=finec;
et1:      if(i>temp) then goto et2;
S;
i:=i+k
goto et1;
et2:      ...;
   move   inic,D0
   move   finec,D1
et1MOVE   D1,D2
   SUB    D0,D2
   BGT    et2
   S
   ADD    k,D0
   BRA    et1
et2...
```

2.217.2.1.5 Costrutto case

```
case tag of
1: S1;
2: S2;
3: S3;
4,5S4;
end;
```


2.217.3 Esempio di sviluppo di un progamma assembler in ambiente ASIM