Transactor Lessons

Walter Cazzola

COLLABORATORS

	TITLE : Transactor Lessons	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Walter Cazzola	February 14, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Transactor Lessons

1.1 Assembler Terza Lezione

Terza Lezione: Modi di Indirizzamento (seconda parte)

Finiamo di vedere i modi di indirizzamento indiretto a registro indirizzi (la volta scorsa me n'è sfuggito uno, sorry!). Il quinto è:

- Indiretto a registro indirizzi con indice

Vediamo subito la sintassi: b(Ax,Rx.w) o b(Ax,Rx.l). Con Rx indichiamo un generico registro dati o indirizzi.

Questo modo, simile rispetto al precedente con spiazzamento, restituisce l'indirizzo effettivo dell'operando sommando al valore contenuto nel registro indirizzi Ax:

- una costante di spiazzamento "b" con segno avente dimensione 8 bit (quindi varia da -128 a +127)

- il contenuto di un registro indice (registro dati o registro indirizzi).

Attenzione! Dalla sintassi notiamo un particolare: questa volta abbiamo la possibilità di definire il tipo di dato oltre che nell'opcode anche insieme al registro indice Rx. Ciò ci permette di sommare tutto il registro indice o solo la word bassa di esso (comunque MAI solo il byte basso, anche se il registro indice fosse un registro dati). Con un unico accorgimento: se la word bassa di Rx ha valore compreso tra $8000 e $ffff, il 68000 automaticamente espanderà il segno e nella somma Rx verrà considerato come se fosse compreso tra $ffff8000 e $ffffffff. Se si vuole evitare il rischio, assicurarsi che la word alta sia nulla e sommare Rx.l.

Vediamo alcuni esempi:

PRIMA DOPO

1) move.l 6(A0,A1.w),D0 A0=$00020000 A0=$00022220

A1=$abcd1500 A1=$abcd1500 <--non ci sarà estensione!

D0=$aaaabbbb D0=$12345678

$21506=#$12345678 $21506=#$12345678 <-- A0+A1+6=$21506

2) move.w -4(A1,D2.w),A0 A1=$00001000 A0=$00001000

D2=$00108056 D2=$00108056 <--ci sarà estensione!

A0=$ffffffff A0=$00001234 <--risultato esteso

$ffff9052=#$1234 $ffff9052=#$1234 <-- A1+D2-4 = 1000+ffff8056-4=$ffff9052

Gli esempi sono abbastanza chiari (il risultato in A0 viene esteso comunque, vi ricordate il modo diretto a registro indirizzi?), comunque il meccanismo è molto semplice: sommando il contenuto di Ax+Rx+"b" otteniamo l'indirizzo da cui prelevare il dato da mettere nell'operando destinazione. Tenere sempre a mente l'eventuale estensione del segno!

INDIRIZZAMENTO RELATIVO AL PROGRAM COUNTER

Esistono 2 modi:

- Relativo al PC con spiazzamento

- Relativo al PC con indice

Gli indirizzamenti relativi al PC sono fondamentali nello sviluppo di programmi di tipo PIC (Position Indipendent Code), ossia dei programmi cosiddetti rilocabili. Scrivere del codice che funzionerà sempre indipendentemente dalla zona di memoria in cui viene caricato è uno dei requisiti più importanti per lo sviluppo di applicazioni in OS multitasking. In questi 2 modi l'indirizzo effettivo da cui reperire operandi viene calcolato aggiungendo un valore all'indirizzo contenuto nel PC. Potremo quindi accedere ad operandi locati un certo numero di byte più in basso o più in alto rispetto all'istruzione corrente. Tranne che con istruzioni di salto, il valore del PC non sarà modificato.

- Relativo al PC con spiazzamento

Questo modo di indirizzamento è identico a quello indiretto a registro indirizzi con spiazzamento, con l'unica differenza che anzichè esserci un registro indirizzi nell'istruzione troviamo il PC. La logica di funzionamento per il resto è identica.

La sintassi è la seguente: w(PC). La costante 'w' è uno spiazzamento di dimensione di 16 bit con segno; si potrà quindi accedere a dati limitatamente a -32768 e +32767 bytes dalla locazione puntata dal PC.

Esempio:

PRIMA DOPO

1) move.w 6(PC),D0 PC=$00024002 PC=$00024006

D0=$aaaa0000 D0=$aaaaffff

$24008=#$ffff $24008=#$ffff

Attenzione! Il PC è cambiato non perchè l'istruzione ne ha variato il contenuto, ma perchè è stato aggiornato a puntare all'istruzione successiva alla 1): la 1) occupa in tutto 4 bytes.

Qualcuno nell'esempio avrà notato che, se il dato da mettere in D0 si trova dopo parecchie istruzioni, senza essere a conoscenza dell'indirizzo in cui sia situato tale dato o in cui si trova l'istruzione corrente, devo perdere parecchio tempo a calcolare la distanza in byte tra la locazione del dato e il valore del PC (immaginate il lavoro necessario se ad esempio tra l'istruzione in corso e l'indirizzo del dato ci fossero cento istruzioni: sarebbe una pazzia andare a calcolare la somma dei bytes occupati da ogni istruzione). Oltretutto, se modificassi il mio programma inserendo o escludendo alcune istruzioni che si trovano fra l'istruzione in corso ed il dato dovrei rifarmi tutti i conti! è per questo che, in mancanza di indirizzi assoluti, si fà ampio uso delle LABEL. Una label è una vera e propria etichetta sistemata, per la sintassi, immediatamente prima dell'istruzione:

LABEL |OPCODE |OPERANDI

move.w prova(pc),D0

..... <-- Qui ci sono varie istruzioni!

prova dc.w "11"

è compito dell'assemblatore calcolare quale sia l'effettivo valore di "prova" nella prima istruzione (che è uguale alla differenza tra la locazione in cui stá l'istruzione dc.w "11" e la locazione in cui stá move.w prova(pc),D0). Oltretutto anche inserendo o togliendo istruzioni tra

move.w prova(pc),d0

..... <--(qui inserisco o tolgo istruzioni!)

prova dc.w "11"

non ci dovremo preoccupare di niente, perchè come prima sarà l'assemblatore ad occuparsi di calcolare il nuovo valore di "prova". Più in là riincontreremo le label.

Gli assemblatori più diffusi, tipo Devpac (ottimo!), accettano SOLO label, niente costanti numeriche di spiazzamento col PC!

Qualcuno potrebbe notare: ma se scrivessi move.w prova,D0? La risposta è: funziona uguale, ma scrivendo così si occupano 6 bytes anzichè 4 e la cpu è impegnata per più cicli. è anche una questione di convenienza!

- Relativo al PC con indice

Di nuovo, questo modo è per logica di funzionamento del tutto identico all'indirizzamento indiretto a registro indirizzi con indice, cioè il quinto della precedente categoria, solo che al posto di Ax avremo il PC. La sintassi? Facile: b(PC,Rx.w) o b(PC,Rx.l).

L'indirizzo effettivo è dato dalla somma tra: PC+Rx+b. 'b' è la solita costante di spiazzamento di dimensione 8 bit con segno, variabile tra -128 e +127 e per Rx.w valgono gli stessi accorgimenti di prima riguardo l'estensione del segno. Un paio di rapidi esempi e concludiamo questo importante modo di indirizzamento:

PRIMA DOPO

1) move.w (PC,D1.l),A0 PC=$00020050 PC=$00020054 <--Se la costante è assente viene calcolata 0 nella somma

D1=$0001a000 D1=$0001a000

A0=$aaaaffff A0=$00001234 <--risultato esteso

$3a050=#$1234 $3a050=#$1234 <--PC+D1+0=$3a050

2) move.w 4(PC,A0.w),A1 PC=$00012000 PC=$00012004

A0=$00008000 A0=$00008000

A1=$00000000 A1=$ffffb000 <--risultato esteso

$a004=#$b000 $a004=#$b000

NB Nel secondo esempio sottolineamo il fatto che A0 viene considerato nella somma come $ffff8000, per via dell'estensione del segno. Quindi la somma sarà: PC+A0+b=$12000+$ffff8000+4=$a004.

L'uso delle label, anche in questo modo di indirizzamento, non solo è consigliatissimo ma come prima in certi casi (Devpac e altri) è obbligatorio.

INDIRIZZAMENTO ASSOLUTO

Due modi di indirizzamento in questo gruppo:

- Assoluto corto

- Assoluto lungo

In questi modi l'operando sorgente o quello destinazione od entrambi rappresentano indirizzi di memoria da cui prelevare o scrivere dati. Una nota: in un OS multitasking questi modi di indirizzamento non potranno MAI essere usati e il motivo è ovvio: scrivendo in un indirizzo deciso a priori (assoluto) potrei rovinare un altro programma, oppure potrei leggere dei dati che credevo diversi....

- Assoluto corto

Questo modo ricorda molto l'indirizzamento zero-page del 6510: si può accedere a delle zone di memoria usando meno spazio di quello normalmente richiesto per accedere ad altri indirizzi e in maniera più veloce.

È il momento di riprendere un discorso lasciato in sospeso: ricordate all'inizio della scorsa lezione parlammo di come il 68000 gestisca gli indirizzi di memoria come longwords tranne nel caso in cui l'indirizzo sia compreso tra $0000 e $7fff o tra $ffff8000 e $ffffffff? Ebbene, l'indirizzamento assoluto corto sfrutta questa caratteristica trattando gli indirizzi in quegli intervalli di memoria come words (2 bytes): basta specificare ".w" dopo l'operando che rappresenta l'indirizzo assoluto di memoria.

Vediamo qualche esempio:

PRIMA DOPO

1) move.l $1234.w,D0 $1234=#$ffaaffaa $1234=#$ffaaffaa

D0=$00000000 D0=$ffaaffaa

2) move.w $ffff8100.w,D1

$ffff8100=#$2c00 $ffff8100=#$2c00

D1=$ffffffff D1=$ffff2c00

Chiariamo bene una cosa: posso applicare l'indirizzamento assoluto corto SOLO se la memoria su cui lavoro è compresa tra $0000 e $7fff o tra $ffff8000 e $ffffffff. In questi casi specificando .w accanto all'operando-indirizzo il 68000 risparmia 2 bytes e qualche ciclo di cpu. Qualunque assemblatore davanti ad un'istruzione del tipo move.w $8100.w,D0 o move.w $ffff5400.w,D0

risponde con un errore di dimensione degli operandi. RICORDIAMO BENE gli intervalli di memoria in cui si può applicare questo modo!

Ma che succede se non specifico .w quando potrei? La risposta viene dal prossimo modo.

- Assoluto lungo

In maniera assolutamente identica al modo precedente questo indirizzamento gestisce operandi come indirizzi di memoria su cui lavorare. L'unica differenza col modo precedente è che, occupando 2 bytes in più e occupando la cpu per più cicli, posso accedere a qualunque indirizzo in memoria, anche a quelli accessibili dal modo assoluto corto, con l'unica differenza che verrano gestiti come longwords anzichè words!

Esempi:

PRIMA DOPO

1) move.w A0,$23400 A0=$ff00ff00 A0=$ff00ff00

$23400=#$aabb $23400=#$ff00

2) move.w $ffff8100,D1

$ffff8100=#$aaaa $ffff8100=#$aaaa

D1=$0000ffff D1=$0000aaaa

Il primo esempio è abbastanza chiaro: preleva la word bassa di A0 e scrivila a partire dall'indirizzo $23400. Il secondo esempio ci serve da confronto col precedente modo: anche se potevamo, non abbiamo specificato .w accanto all'indirizzo. Cosa cambia? A livello di risultati niente, ma ora verranno usati 2 bytes in più e un pò più di tempo per eseguire l'istruzione!

INDIRIZZAMENTO IMMEDIATO

A questa categoria appartengono 2 modi:

- Immediato

- Immediato rapido

In questi 2 modi l'operando sorgente è una costante. Sono molto semplici e il secondo ha una caratteristica interessante.

- Immediato

In questo indirizzamento una costante, che può avere dimensioni di byte, word o longword, viene trasferita in un registro generico (NB solo word e longword se registro indirizzi). Se il tipo di dato della costante è byte o word, lo spazio usato dall'opcode (detto anche op-word, dato che come sappiamo i comandi del 68000 sono lunghi 2 bytes) sommato agli operandi sarà 4 bytes. Se il dato ha dimensioni pari a una longword, in tutto useremo 6 bytes tra opcode e operandi. I dati verranno inoltre estesi di segno se la destinazione è un registro indirizzi e la dimensione del dato da spostare è una word.

Vediamo due esempi:

PRIMA DOPO

1) move.w #$8000,A0 A0=$aaaa0000 A0=$ffff8000 <-- esteso!

2) move.l #$8000,A0 A0=$aaaa0000 A0=$00008000

3) move.w #$12,A0 A0=$aaaa0000 A0=$00000012 <-- esteso!

4) move.b #$c4,D0 D0=$12345678 D0=$123456c4

Questo indirizzamento è veramente semplice, per cui passiamo subito all'altro modo di questa categoria.

- Immediato rapido

Questo modo è applicabile solo con 3 istruzioni, che ora diamo e sulle quali torneremo le prossime volte:

1) moveq (move quick),

2) addq (add quick),

3) subq (sub quick).

Il vantaggio offerto da queste 3 istruzioni (che in effetti costituiscono questo modo!) è di far stare opcode e operandi in 2 bytes di spazio, praticamente includendo il dato nell'opcode, a patto di rispettare certe condizioni. Addq e subq permettono di sommare o sottrarre ad un registro qualunque o ad una locazione di memoria un valore compreso tra 1 e 8. Sono a tutti gli effetti le istruzioni di incremento e decremento del 68000 (l'equivalente di INX e DEX o INY e DEY del 6510 ma molto più potenti perchè posso incrementare o decrementare da 1 a 8 bytes alla volta). Moveq consente, utilizzando 2 soli bytes, di spostare SOLO in un registro dati una costante con segno di dimensioni pari a 1 byte estendendone il segno: quindi la costante varia da -128 a +127 e il registro dati assumerà corrispondentemente un valore variabile tra $ffffff80 e $0000007f. Mi rendo conto che abbiamo sconfinato un pò troppo nelle istruzioni ancor prima di iniziare a spiegarle, ma era l'unico modo per affrontare questo modo di indirizzamento.

Vediamo un pò di esempi:

PRIMA DOPO

1) addq.w #$2,A0 A0=$a0a0b0b0 A0=$a0a0a0a2

2) subq.b #$3,D0 D0=$12345678 D0=$12345675

3) moveq #$6e,D1 D1=$aaaabbbb D1=$0000006e

4) moveq #-$70,D1 D1=$12345678 D1=$ffffff90

E con questi esempi abbiamo terminiamo i modi di indirizzamento. La prossima volta inziamo (sul serio!) con le istruzioni.

Piccole curiosità: come impedire l'accesso agli utenti di cartuccie tipo Action Replay?

Ve le ricordate? Premendo un tastino potevo bloccare l'esecuzione del gioco e provare ad avere vite infinite e altre cose!

Perchè si blocca l'esecuzione? Perchè è previsto che la cpu, in presenza di certi segnali software o hardware, possa interrompere il programma che sta eseguendo, salvare sullo stack il PC e lo SR attuale per riprenderli quando rinizierà l'esecuzione del programma interrotto e, tramite una routine di gestione dell'interruzione (meglio noto in inglese come Interrupt) eseguire del codice che voglio (o che vuole l'OS, se stiamo sotto OS, ma questo non è il nostro caso). È bene sapere che esistono 7 diversi livelli di interruzione, da 1 a 7, il più alto, regolati dai 3 flag dello SR I0, I1, I2. L'int. di livello 7 è l'NMI, interrupt non mascherabile, cioè un interrupt alla presenza del quale la cpu deve assolutamente bloccarsi e dedicare attenzione anche se tramite particolare accorgimenti è possibile bloccare tutti gli altri interrupt (quindi tutti quelli fino a livello 6). Ebbene, un interrupt del genere era quello che veniva prodotto dalla Action Replay. Come fermare gli intrusori?

Anzitutto dobbiamo sapere che se disabilitiamo tutti gli Interrupt e quindi scriviamo del codice che non li sfrutta possiamo fare in modo che lo Stack Pointer (ci ricordiamo tutti cos'è, vero?) punti ad un indirizzo dispari: situazione questa del tutto anomala per il 68000, che come sappiamo può gestire solo indirizzi pari pena enormi GURU. Nel momento in cui il cartucciaro infame (!) spinge il famoso tastino che succede? Il 68000 avvia la normale procedura hardware di gestione dell'interruzione: prende cioè il PC e lo SR e li mette sullo stack....ma non può, perchè lo stack pointer punta ad un indirizzo dispari! A questo punto il 68000 genererà un errore di indirizzo, che è anch'esso un interrupt! Quindi di nuovo proverà a mettere il PC e lo SR sullo stack e di nuovo non potrà...e così via all'infinito. Il 68000 entrerà in un tremendo loop dal quale si potrà uscire solo riavviando tutto, facendo quindi perdere informazioni preziose all'intrusore mancato! Carino, vero?

Lezione Precedente

Indice Assembler Indice Corsi