Corso di Calcolatori Elettronici II

---

# ASIM

# Ambiente per la SImulazione dei sistemi a Microprocessore

(Attenzione: il testo è ancora in Bozza e da correggere)

Antonino Mazzeo

1 CARATTERISTICHE PRINCIPALI DI ASIM ..... 6

2 IL MECCANISMO DI SIMULAZIONE ..... 9

2.1 Il meccanismo di simulazione ..... 9

**2.1.1 Classi di oggetti attualmente simulabili in ASIM** ..... 10

**2.1.2 Classe dei dispositivi Processore** ..... 10

**2.1.3 Classe dei dispositivi Nodo** ..... 10

**2.1.4 Classe dei dispositivi Bus/Memoria** ..... 11

**2.1.5 Classe dei dispositivi device** ..... 11

3 Manuale d'uso di ASIM ..... 12

3.1 I Menù della barra di Controllo di ASIM ..... 12

3.1.1 Il menù "File". ..... 12

3.1.2 Il menù "Edit". ..... 13

3.1.3 Il menù **View (*)**. ..... 14

3.1.4 Il menù **Simulation**. ..... 15

3.1.5 Il menù **Window**. ..... 16

3.1.6 Il menù **Help**. ..... 16

3.2 La barra degli strumenti di ASIM ..... 16

4 CONFIGURAZIONE E SIMULAZIONE DI UN SISTEMA E DEGLI OGGETTI DI ASIM 18

4.1 Il dispositivo Processore ..... 19

4.1.1 Il processore M68000 e la gestione delle interruzioni ..... 20

4.2 Il dispositivo Bus/Memoria ..... 22

4.3 Il dispositivo 1TO4BusInterface ..... 24

4.4 Il dispositivo generatore di interruzioni 1to4INTGEN. ..... 26

4.4.1 Il generatore di interruzioni programmabile ..... 26

4.4.2 La sezione con i 4 timer ..... 27

4.5 IL Priority Interrupt Controller ..... 29

4.5.1 Modello di programmazione ..... 30

4.5.2 Modalità di selezione dei registri ..... 32

4.5.3 Collegamento del dispositivo con altri componenti di ASIM... 32
4.5.4 Programmazione del PIC ..... 33
4.6 Il dispositivo DMA ..... 36
4.6.1 Collegamento al processore ..... 41
4.6.2 Collegamento ad un device..... 41
4.6.3 Configurazione ASIM del DMA ..... 42
4.6.4 Modello di programmazione..... 43
4.6.5 Descrizione dei comandi ..... 45
4.6.6 Programmazione ..... 45
4.7 Il dispositivo PIA Periferal Interface Adapter ..... 49
4.7.1 Programmazione del dispositivo PIA..... 50
4.8 Il dispositivo video-tastiera TERMINAL ..... 54
4.9 Il dispositivo USART ..... 57
4.9.1 Protocollo di comunicazione asincrono ..... 58
4.9.2 Protocollo di comunicazione sincrono ..... 58
4.9.3 Generalità USART ..... 58
4.9.4 Interfacci verso il BUS ..... 60
4.9.5 Interfaccia verso altri dispositivi..... 61
4.9.6 Collegamento software..... 63
4.9.7 Modello di programmazione..... 63
4.9.8 Programmazione ..... 65
4.9.9 Il comportamento. ..... 67
4.9.10 Trasmissione asincrona ..... 67
4.9.11 Trasmissione sincrona..... 68
4.9.12 Ricezione asincrona ..... 69
4.9.13 Ricezione sincrona ..... 69
4.10 Dispositivo Tappo ..... 71
4.10.1 Collegamento al processore e ad un device..... 72

4.10.2 Collegamento software ..... 73

**4.10.3** Modello di programmazione ..... 74

4.11 Il dispositivo di collegamento tra nodi: ELABNODE. ..... 75

4.12 Il dispositivo nodo "data flow": ARTMNODE. ..... 76

# Parte I-descrizione funzionale e modalità d'uso di ASIM

# 1 CARATTERISTICHE PRINCIPALI DI ASIM

ASIM (**A**mbiente **SI**mulazione **M**icroprocessori) è un ambiente didattico per la simulazione di sistemi a microprocessore e che consente all'utente di definire proprie configurazioni di architetture hw e di estendere l'insieme originario di dispositivi presenti con nuovi componenti sviluppati in ambiente C++.

ASIM è stato sviluppato e si è evoluto a partire dal 1984 in seno ai corsi di Macchine per l'Elaborazione delle Informazioni e Calcolatori elettronici II, con la collaborazione degli studenti e con il lavoro prezioso di numerosi tesisti di cui voglio ricordare soltanto l'ing. L. Impagliazzo che ha dato un notevole contributo nel portare, ridisegnandolo strutturalmente, il simulatore nell'ambiente C++/Windows. La versione iniziale è stata scritta in C e operava in ambiente MSDOS, è in atto il porting in JAVA nelle piattaforme Linux e Windows.

ASIM è un ambiente di simulazione in cui i sistemi hw da simulare sono caratterizzati da configurazioni di architetture composte dall'interconnessione di oggetti (CPU, memorie, bus, device e nodi) e su cui poter operare sia in modalità mono che multiprocessore con programmi scritti in linguaggio assembler.

Poiché ASIM consente di specificare architetture di vario tipo (mono, multi processore, data flow, shared memory, message passing, etc.) è possibile utilizzare l'ambiente oltre che come un laboratorio per lo studio di sistemi a microprocessore, con l'impiego di programmazione in assembler, anche come un laboratorio per lo studio dei livelli architetturali superiori di sistemi (S.O., multiprogrammazione di basso livello, ecc.). Ad esempio i processori possono essere connessi con bus locali, globali e crossbar, oppure possono formare, insieme ad una memoria, dei nodi di elaborazione; questi possono poi essere connessi tra loro con link per formare strutture ad ipercubo o mesh.

Il limite alla complessità dei sistemi simulabili, in termini di componenti impiegati in una configurazione, è imposto dalla configurazione (capacità di memoria, velocità del processore in uso, ecc.).

Il simulatore ASIM è scitto in C++, opera in ambiente Microsoft Windows, di cui riflette l'organizzazione a finestre e utilizza pienamente le classi offerte da MFC (Microsoft Foundation Classes) per l'interazione con Windows stesso. Esso è organizzato come applicazione di tipo MDI (Multiple Document Interface) cioè un'applicazione con documenti e viste multiple. L'applicazione MFC è caratterizzata da quattro classi principali:

- CasimApp ← CwinApp
- CasimDoc ← Cdocument
- CasimFrame ← CMDIChildWnd
- CasimView ← Cview

A destra della freccia sono indicate le classi di MFC da cui sono ereditate le quattro classi di ASIM. Per i dettagli sull'architettura di ASIM di consulti il documento "ASIM Internal Design".

L'ambiente IDE (Integrated Development Environment) è caratterizzato da una finestra principale (Fig. 1-1 La finestra principale di ASIM) che delimita l'area di lavoro e presenta un menù da cui è possibile accedere a tutti i comandi. La barra di fondo (barra di stato) indica il numero di eventi (clock) schedulati nella simulazione e il punto di "break", relativamente al numero di clock da eseguire, cioè l'evento che determina l'arresto momentaneo della simulazione. In ASIM, infatti, la simulazione viene scandita dai segnali di "clock" o "passi" inviati dallo schedulatore a tutti gli elementi; l'intervallo minimo di simulazione è, quindi, il clock; il numero di azioni che ogni dispositivo compie in un colpo di clock dipende dalle caratteristiche del dispositivo stesso.

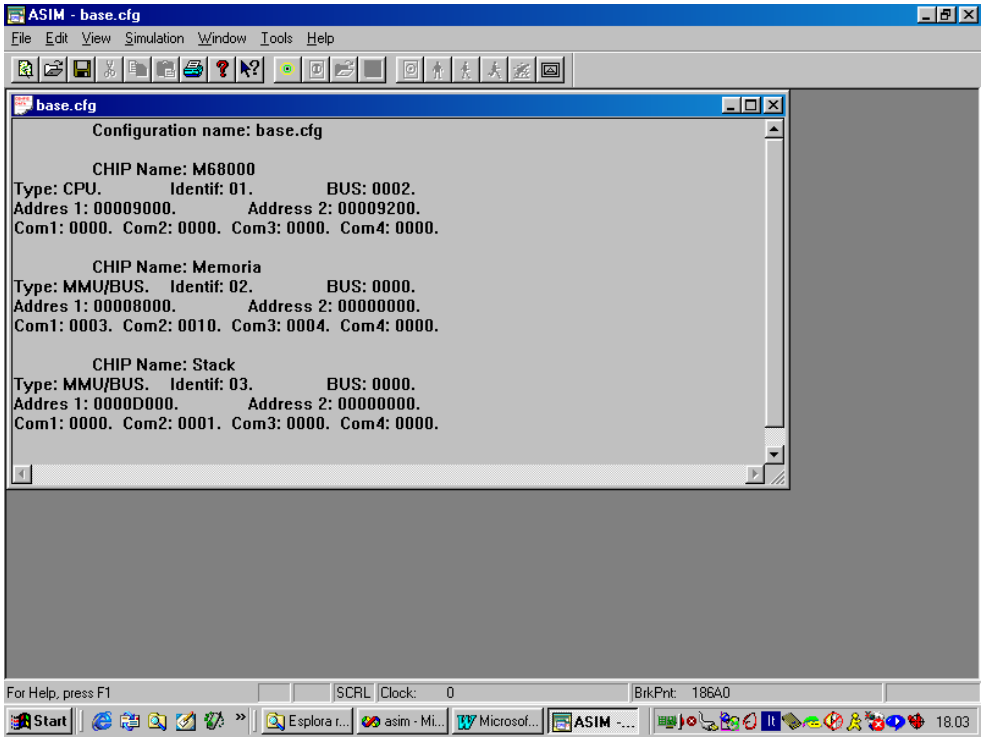


Fig. 1-1 La finestra principale di ASIM

Ad ogni oggetto componente una configurazione da simulare è associata una finestra in cui vengono presentati i dati che lo caratterizzano, coincidenti, in linea di massima, con il modello di programmazione del componente generico (ad es. il contenuto dei vari byte per una memoria o dei registri per un processore). Quando una finestra è attiva (selezionata), il menù sulla barra di controllo viene specializzato per le specifiche funzioni dell'oggetto associato. La finestra corrente può essere ridimensionata, spostata in ogni punto dell'area di lavoro o ridotta a icona (quest'ultima possibilità va sempre sfruttata nel corso di una simulazione, quando non si è interessati alla visualizzazione dei dati di un particolare elemento, in modo da aumentare la velocità della simulazione stessa).

In ASIM tutti i numeri sono rappresentati in notazione esadecimale, uniche eccezioni sono le rappresentazioni in binario per quei registri in cui assumono significato singoli o aggregati di bit (ad esempio il registro di stato di un processore).

Dalla versione 3.0 ASIM offre:

- la gestione di configurazioni multiple;
- l'accesso rapido alle configurazioni usate più di recente;
- il lancio del programma con la configurazione desiderata già aperta chiamando il relativo file *.cfg dal "FileManager" di Windows;

- la creazione di una configurazione mediante il meccanismo di "taglia e incolla" tipico di Windows;
- la visualizzazione completa dei dati relativi a ciascuna configurazione;
- la stampa di detti dati con font di caratteri, intestazione e piè di pagina selezionabili; numerazione automatica delle pagine;
- l'anteprima di stampa delle pagine da inviare alla stampante;
- l'esecuzione veloce di una simulazione;
- la simulazione contemporanea di due configurazioni;
- la simulazione in background mentre si lavora su un'altra configurazione;
- l'accesso rapido ai comandi mediante ToolBar;
- l'help in linea su barra di stato;
- l'help da indice o dipendente dal contesto;
- l'accesso alla "Guida alla Guida" in Windows;
- la protezione da errori d'uso mediante il meccanismo di mascheramento dei comandi.

# 2 IL MECCANISMO DI SIMULAZIONE

## 2.1 Il meccanismo di simulazione

Una simulazione in ASIM è caratterizzata da una specifica configurazione di sistema a microprocessori, contenente processori e dispositivi (oggetti) messi a disposizione dall'ambiente. Tutti gli oggetti inclusi in una configurazione vengono inseriti in una lista che il simulatore consulta per sviluppare i singoli passi di simulazione e per effettuare controlli di consistenza delle interfacce dei vari oggetti e i vincoli di interconnessione. Definita la configurazione della macchina da simulare (mediante l'ausilio dell'editor di configurazione) si provvede a "costruirla" premendo il bottone di "costruzione" e ad attivarla premendo il bottone di "accensione" (o l'equivalente voce ON/Reset del Menù Simulazione) della macchina realizzata (Fig. 2-1-La barra di controllo di ASIM).

All'attivazione di una configurazione, sono eseguiti controlli sulle connessioni predisposte tra i dispositivi, in particolare, la finestra principale invia un messaggio (messaggio bloccante Windows WM_RESET) a ciascun dispositivo interessato e i cui dati caratterizzanti sono raccolti in un oggetto della lista e aspetta che questo esegua i necessari controlli. Se viene riscontrato nel dispositivo qualche errore di configurazione, una finestra di dialogo segnala all'utente l'anomalia e viene restituito un codice di errore che interrompe ulteriori controlli e sospende la simulazione. In assenza di errori, la finestra principale invia lo stesso messaggio ad un altro *dispositivo* e, solo se tutti i controlli per ogni componentedanno esito positivo, abilita i comandi per la simulazione (per ulteriori dettagli si analizzino i sorgenti dei moduli CMainWindow e CBaseWnd).

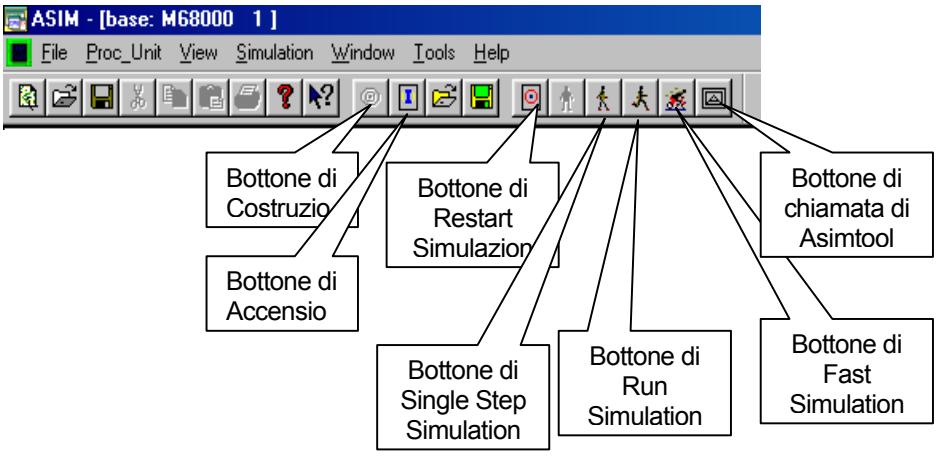



**Fig. 2-1-La barra di controllo di ASIM**

In modo analogo è eseguito un passo di simulazione ma con la differenza che i messaggi sono inviati in successione e in modo non bloccante.

Per controllare l'esecuzione di un passo di simulazione la finestra principale utilizza due parametri ClockHigh (TRUE se un passo di simulazione è in corso) e ChildBusy (che indica in ogni istante il numero di dispositivi che stanno eseguendo un passo di simulazione).

Alla richiesta di un passo di simulazione, ClockHigh è posto a TRUE e la finestra principale invia a tutti i dispositiviil messaggio WM_CLOCK senza attendere risposta (messaggio non bloccante); ChildBusy assume un valore pari al numero di componenti che formano la configurazione, ad indicare il fatto che tutti i dispositivi stanno eseguendo IN PARALLELO le operazioni previste dal passo di simulazione (operazioni che comportano in generale un'interazione tra i vari dispositivi).

Quando un dispositivo ha completato l'esecuzione del passo di simulazione, un messaggio (WM_COMPL ) segnala alla finestra principale (che decrementa ChildBusy) l'evento. Quando tutti i dispositivi hanno completato (ChildBusy è tornato a zero e ClockHigh è riportato a FALSE) un nuovo passo di simulazione può essere richiesto (per maggiori dettagli si consultino i sorgenti del modulo CMainWnd::TraceProg che riporta il codice richiesto per il lancio di un passo di simulazione).

ASIM consente, oltre all'esecuzione di singoli passi di simulazione (Single Step Simulation) su richiesta dell'utente, anche l'esecuzione dei programmi (Run Simulation) in modalità "trace" (come successione di singoli passi) e l'esecuzione completa (Fast Simulation) con cui eseguire velocemente la simulazione senza la visualizzazione dei dati di "trace" .

### 2.1.1 Classi di oggetti attualmente simulabili in ASIM

Un sistema calcolatore è costituito da un insieme di componenti (processori, memorie, device, ecc.) fra loro interconnessi. In Asimsono stati implementati un ridotto numero di modelli di componenti da cui poter derivare, secondo il meccanismo di ereditarietà tipica delle classi C++, specifici componenti. Questi, a seconda delle funzionalità svolte e delle modalità di interfacciamento con altri dispositivi, sono stati raggruppati in quattro classi principali. L'appartenenza di un dispositivo ad una di tali classi è subordinata al possesso di almeno una delle proprietà che la caratterizza (nel ricavare le proprietà si è fatto un confronto tra il modello proposto e le reali architetture dei sistemi esistenti). Sono descritte, di seguito, le classi allo stato supportate dal simulatore.

### 2.1.2 Classe dei dispositivi Processore

Processore: un qualsiasi dispositivo che ha lo stato interno rappresentato dal contenuto dei suoi registri, più un insieme di informazioni riguardanti il/i bus cui è connesso e le eventuali richieste di interruzioni pervenute e che gode delle seguenti proprietà:

- P1) può eseguire un programma contenuto in una memoria (interna o esterna al dispositivo stesso);
- P2) può accedere attraverso il/i bus cui è connesso ad una memoria (esterna) o ai registri di altri dispositivi per eseguire operazioni di lettura o scrittura;
- P3) può servire e/o gestire interruzioni provenienti da altri dispositivi, eventualmente assegnando ad esse un livello di priorità.

Oltre alle CPU, che sicuramente sono classificabili come processore in quanto godono di tutte le proprietà suddette, rientrano in tale insieme anche i processori video e di IO ed i coprocessori matematici (godono delle proprietà P1 e P2), i DMA (godono della proprietà P2), i "priority interrupt controller" (godono della proprietà P3) ed infine strutture "hardware" per la gestione prioritaria delle interruzioni come il "daisy chaining" (godono della proprietà P3). I processori supportati da ASIM appartengono a tale insieme.

### 2.1.3 Classe dei dispositivi Nodo

Nodo: è un dispositivo il cui stato interno raccoglie informazioni relative alle connessioni con altri nodi ed ai messaggi in transito da/verso questi e che gode delle seguenti proprietà:

N1) può consentire la connessione tra dispositivi dello stesso tipo;

N2) può gestire, come nodo intermedio, la connessione tra dispositivi dello stesso tipo, instradando la comunicazione;

N3) può avere capacità autonome di elaborazione e trasformare (secondo una logica qualsiasi) i messaggi o dati ricevuti.

N4) può essere connesso ad un bus/memoria.

### 2.1.4 Classe dei dispositivi Bus/Memoria

Bus/memoria: un dispositivo il cui stato interno è caratterizzato dai valori assunti da un insieme molto ampio di registri e/o da informazioni riguardanti i dispositivi che esso connette e che gode delle seguenti proprietà:

M1) può consentire a dispositivi di tipo processore di leggere o scrivere i suoi registri interni (la selezione avviene in base all'indirizzo);

M2) può consentire a dispositivi di tipo processore di leggere o scrivere i registri dei dispositivi di tipo device (la selezione avviene in base all'indirizzo);

M3) regola l'accesso di più dispositivi di tipo processore ai suoi registri ed ai device (in ogni istante al più un accesso è in corso, gli altri processori devono attendere);

M4) gestisce la memoria fisica effettuando il “mapping” degli indirizzi virtuali negli indirizzi fisici. M5) può consentire l'esecuzione di cicli non interrompibili di lettura-modifica;

M6) può connettersi ad altri dispositivi dello stesso tipo sia per costruire accessi da un altro bus sia verso un altro bus;

### 2.1.5 Classe dei dispositivi device

Device: tutti i dispositivi che non rientrano nelle tre precedenti classi e che godono delle seguenti proprietà:

D1) può essere connesso ad un bus/memoria in modo che un processore possa accedere ad esso;

D2) può essere in grado di generare delle interruzioni da inviare ad un processore; nel qual caso deve essere connesso in qualche modo al processore;

D3) può essere connesso e comunicare con un altro device della stessa macchina o di un'altra macchina;

D4) può connettere dispositivi di tipo processore e bus/memoria a dispositivi del tipo bus/memoria.

# 3 Manuale d'uso di ASIM

## 3.1 I Menù della barra di Controllo di ASIM

### 3.1.1 Il menù "File".

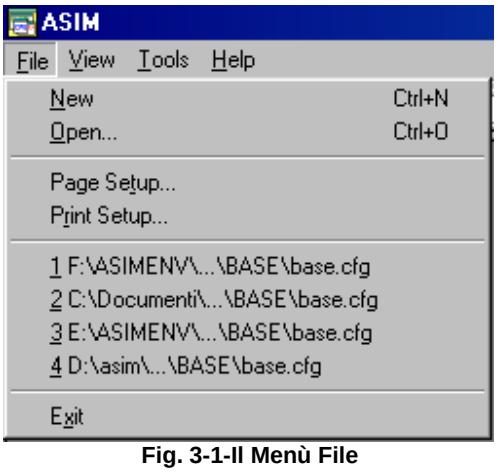



**Fig. 3-1-Il Menù File**

Dal menù **File** (Fig. 3-1-Il Menù File) è possibile accedere alle tradizionali funzioni di apertura (nuova o già esistente) di una configurazione, alla stampa della stessa e alla chiusura di ASIM.

**New** consente di creare una nuova configurazione aprendo una finestra ed attivando per essa il menù Edit (paragrafo 1.2); da questo menù sarà possibile selezionare tutti i comandi necessari a definire i componenti della nuova configurazione. Poiché è possibile avere più configurazioni aperte contemporaneamente, il comando New non chiude alcuna precedente configurazione.

**Open** apre un file di configurazione precedentemente salvato; non chiude alcuna precedente configurazione.

**Close** chiude la configurazione corrente; se vi sono state modifiche chiede se il file deve essere aggiornato.

**Save, Save As** salva il file di configurazione corrente; Save As consente di specificare un nuovo nome per il file che diventa anche il nome della configurazione corrente.

**Print** attivo solo se la finestra corrente è una di quelle che mostra i dati della configurazione; il comando consente di stampare questi dati con un set di caratteri selezionabile. Se è necessario utilizzare più di una pagina per stampare i dati, la paginazione è automatica.

**Page Setup** consente di impostare l'intestazione ed il piè di pagina; tipicamente questi riportano rispettivamente il nome del file di configurazione ed il numero di pagina.

**Print Preview** questo comando fornisce un'anteprima di stampa ovvero consente di vedere l'effetto della stampa della configurazione corrente sulla stampante e con le impostazioni selezionate.

**Print Setup** imposta la stampante.

**1 .. 4 (Most Recent Used)** nome dei file di configurazione usati più di recente; consente di aprire le relative configurazioni immediatamente, senza passare per il comando Open.

**Exit**, esce da ASIM; per tutte le configurazioni modificate viene richiesto se salvare o meno le modifiche.

### 3.1.2 Il menù "Edit".

Dal menù **Edit** è possibile accedere a tutti i comandi che consentono di definire una configurazione di sistema; tale configurazione rappresenta la macchina da simulare. Per convenzione i file che contengono le informazioni relative ad una configurazione hanno come estensione ".cfg".

Il menù si presenta come inFig. 3-2-Il Menù Edit.

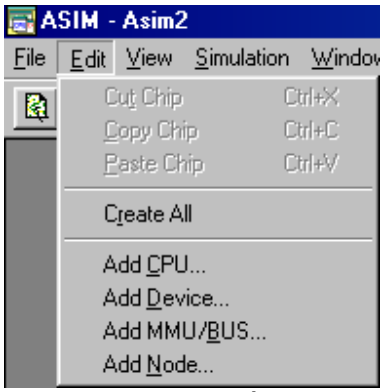



**Fig. 3-2-Il Menù Edit**

Questo menù è disponibile solo se la finestra corrente è quella che mostra i dati relativi alla configurazione (corrente); esso mette a disposizione tutti i comandi utili a costruire la macchina da simulare.

**Cut Chip** elimina il componente selezionato dalla configurazione corrente e salva i dati ad esso relativi nella Clipboard; con il comando Paste Chip sarà possibile inserire questo componente in un'altra configurazione. La selezione del componente avviene con un doppio click del mouse nell'area della finestra che contiene i dati del chip desiderato.

**CopyChip** vale quanto detto per Cut Chip con l'unica differenza che il componente selezionato non viene eliminato dalla configurazione corrente ma se ne crea una copia.

**Paste Chip** consente di inserire in una configurazione un componente i cui dati sono disponibili nella Clipboard; grazie a questo comando ed ai due precedenti è possibile creare nuove configurazioni a partire da precedenti in maniera molto semplice attraverso il tipico meccanismo "copia e incolla" di Windows.

**Create All** dalla versione 2.0 di ASIM in poi non necessariamente viene creata insieme al componente anche la finestra ad esso associata; ciò per evitare di affollare il video di finestre non utili. È possibile quindi costruire prima una macchina con i comandi di "copia e incolla" o di "Add ..." e poi creare tutte le finestre associate ai componenti con questo comando. È indispensabile eseguire Create All per poter rendere disponibili i comandi relativi alla simulazione.

**Add CPU, Add Device, Add MMU/BUS, Add Node** consentono di aggiungere alla configurazione corrente un nuovo componente. La finestra associata al nuovo componente è creata immediatamente solo se è stato dato in precedenza il comando Create All per la configurazione corrente.

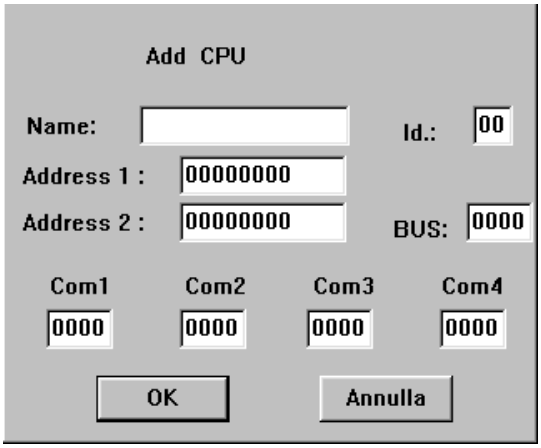



**Fig. 3-3-Menù di Configurazione Oggetti**

I comandi Add_CPU, Add_Device, Add_MMU/BUS, Add_Node, danno accesso ad una finestra di dialogo (Fig. 3-3-Menù di Configurazione Oggetti**Errore. L'origine riferimento non è stata trovata.**) dove possono essere specificati il nome, l' identificatore ed una serie di parametri che caratterizzano l'elemento da inserire nella configurazione corrente. L' identificatore deve essere un numero compreso tra $01 e $FF e ciascun dispositivo deve avere un identificatore diverso da ogni altro (ciò limita a 255 i componenti appartenenti ad una configurazione); ogni qual volta, nel definire le connessioni tra i vari dispositivi, è necessario riferirsi ad un certo elemento, ciò viene fatto specificandone l'identificatore. Il significato degli altri parametri è dipendente dal particolare componente e sarà chiarito di volta in volta.

### 3.1.3 Il menù **View (*)**.

Il menù View (Fig. 3-4-Il Menù View) abilita/disabilita alcune funzionalità offerte dalle classi MFC quali toolbar, barre di stato, help in linea, selezione dei font di caratteri.

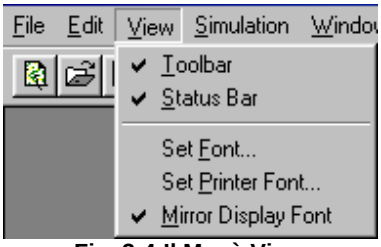



**Fig. 3-4-Il Menù View**

**Toolbar (*)**, mostra/nasconde la barra che consente l'accesso rapido ad alcuni comandi (toolbar); i tasti presenti nel toolbar consentono (nell'ordine da sinistra a destra) di attivare immediatamente i comandi: New, Open, Save, Cut, Copy, Paste, Print, Index, Help, Restart, Stop, Trace, Run, Fast Run, ASIM Tool.

**Status Bar(*)**, mostra/nasconde la barra di stato; quest'ultima mette a disposizione dell'utente un "help in linea" ovvero presenta a video una frase che indica l'effetto dell'azione che si sta compiendo o si sta per compiere (selezione di un comando, spostamento di una finestra, ecc.). La barra di stato mostra inoltre due informazioni relative alla configurazione corrente: numero dell'ultimo passo di simulazione eseguito (clock), numero dell'ultimo passo di simulazione da eseguire prima di interrompere la simulazione (BreakPoint); queste informazioni sono visibili solo se sono attivi, rispettivamente, i tasti "Caps Lock" e "Num Lock".

**Set Font (*)**, consente di selezionare il font di caratteri utilizzato dalla finestra che mostra i dati relativi alla configurazione; quest'ultima deve essere la finestra corrente affinché il comando sia disponibile.

**Set Printer Font (*)**, consente di selezionare il font di caratteri utilizzato dalla stampante; affinché il comando sia disponibile la finestra corrente deve essere quella contenente i dati relativi alla configurazione. È ammesso che il font selezionato per la stampante sia diverso da quello selezionato per lo schermo.

**Mirror Display Font (*)**, forza la stampante ad utilizzare lo stesso font di caratteri selezionato per il video con il comando Set Font.

### 3.1.4 Il menù **Simulation**.

Il menù Simulation (Fig. 3-5-Il Menù Simulazione**Errore. L'origine riferimento non è stata trovata.**) rende accessibili i comandi per il controllo della simulazione di configurazioni.


**Fig. 3-5-Il Menù Simulazione**

**ON/Reset**, accende/resetta la macchina da simulare, controlla le connessioni. ON/Reset è abilitato solo se, per la configurazione corrente, è stato dato il comando Create All del menù Edit (paragrafo 1.2).

**Restart** riporta la macchina nelle condizioni in cui si trovava prima di avviare la simulazione.

**RUN** lancia una simulazione. ASIM consente anche l'esecuzione di due simulazioni in parallelo oppure di lavorare con una configurazione mentre è in corso la simulazione per un'altra configurazione.

**STOP**, arresta la simulazione in corso.

**TRACE**, lancia un singolo passo di simulazione.

**Fast RUN** esegue le azioni previste dal comando RUN ma prima riduce ad icona tutte le finestre associate ai componenti presenti nella configurazione corrente; ciò consente di accelerare notevolmente l'esecuzione di una simulazione.

**BreakPoint**, fissa il numero di passi dopo cui bloccare automaticamente la simulazione.

### 3.1.5 Il menù **Window**.

Questo menù (Fig. 3-6-Il Menù Window) raccoglie i comandi utili alla gestione dell'ambiente a finestre multiple.

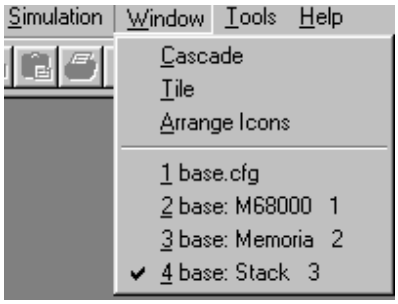



**Fig. 3-6-Il Menù Window**

**Cascade**, dispone le finestre in cascata.

**Tile**, dispone le finestre affiancate.

**Arrange Icons**, risistema le icone sul fondo della finestra principale di ASIM.

**1..n**, seleziona una finestra portandola in primo piano (finestra corrente).

### 3.1.6 Il menù **Help**.

Questo menù (Fig. 3-7-Il Menù Help) raccoglie i comandi utili ad ottenere informazioni su ASIM e sul suo utilizzo.

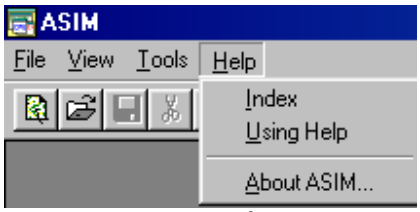



**Fig. 3-7-Il Menù Help**

**Index (*)**, consente di accedere alla guida all'uso di ASIM. Attualmente ASIM 2.0 offre tutto il supporto necessario a livello di codice per realizzare sia una guida all'uso sia un help dipendente dal contesto; resta da compilare il file di testo contenente le informazioni.

**Using Help (*)**, consente di accedere a tutte le informazioni necessarie per l'uso di una guida in ambiente Windows.

**About ASIM**, mostra le informazioni relative ad ASIM (autore, versione, ecc.).

## 3.2 La barra degli strumenti di ASIM

La barra degli strumenti di Asim (Fig. 3-8-La barra degli strumenti di ASIM) presenta 19 bottoni raggruppati in 3 gruppi. I 9 bottoni del primo gruppo servono a selezionare le tradizionali funzioni di Windows. I 4 bottoni del secondo gruppo servono ad attivare la configurazione e a ripristinare e salvare la configurazione di lavoro (workspace), i primi 5 bottoni del terzo gruppo servono a selezionare le funzioni di Asim di stop, arresto, esecuzione di un passo, esecuzione ed esecuzione veloce iconizzata di una simulazione. L'ultimo bottone serve ad attivare l'ambiente assembler detto Asimtool.

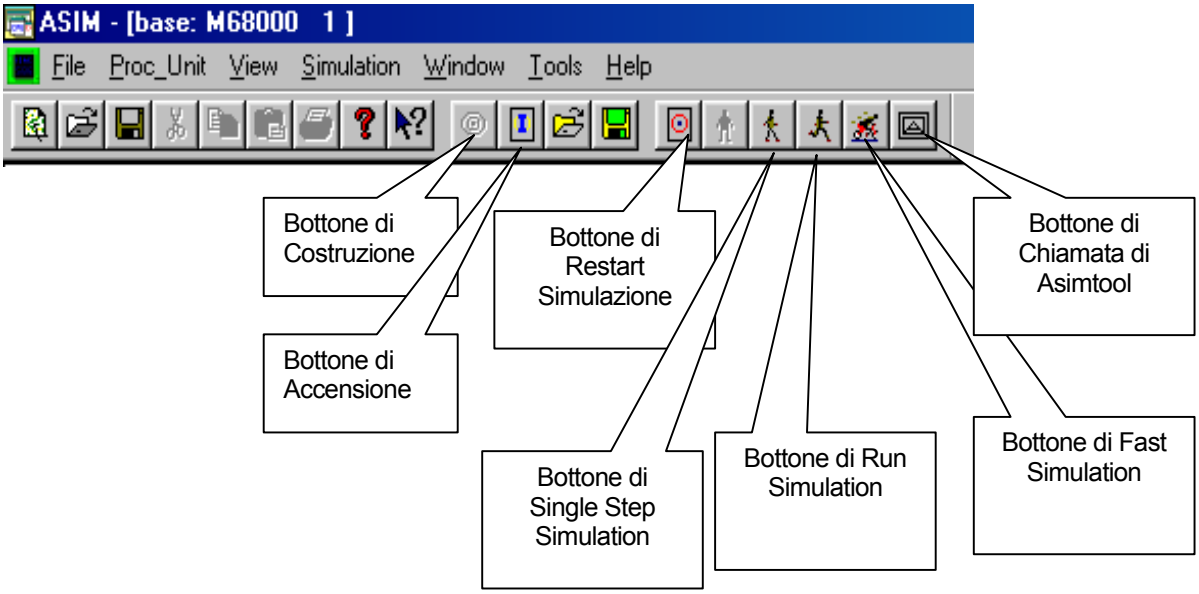



**Fig. 3-8-La barra degli strumenti di ASIM**

Taluni dei comandi indicati nei bottoni e nei Menù di Asim possono essere attivati ricorrendo all'uso dei tasti funzione così come riportato negli stessi menù. I comandi possono, infine, essere attivati premendo il tasto ALT e il carattere sottolineato riportato nel nome dei comandi nei vari Menù.

# 4 CONFIGURAZIONE E SIMULAZIONE DI UN SISTEMA E DEGLI OGGETTI DI ASIM

In Asim un sistema è descritto mediante una configurazione contenente un insieme di oggetti appartenenti ad una delle classi sopra descritte. L'editor di configurazioni di ASIM, per tramite di una "finestra di dialogo", consente di immettere tutti i parametri necessari a descrivere i singoli oggetti ed il loro interfacciamento. La finestra di dialogo presenta una maschera comune a tutti gli oggetti. Tale maschera è dotata di 9 campi che, a seconda dell'oggetto, codificano informazioni atte a caratterizzarlo all'interno della configurazione hw e, quindi, della simulazione. Il significato di tali campi è, in generale, il seguente:

- **configuration name:** nome della configurazione corrente;
- **chip name:** nome in codice dell'oggetto se esistono più oggetti in una classe; arbitrario se l'insieme contiene un unico dispositivo;
- **type:** classe di appartenenza dell'oggetto;
- **identif:** codice numerico che identifica lo specifico oggetto nell'ambito di una configurazione;
- **bus:** identificatore di un oggetto bus con cui lo specifico oggetto può interagire in qualche modo;
- **address:**indirizzi fisici compresi nello spazio indirizzi visto dallo specifico oggetto;
- **com1…com4:** quattro campi che codificano particolari caratteristiche di uno specifico oggetto.

Si riportano di seguito le descrizioni degli oggetti attualmente utilizzabili nelle configurazioni di ASIM. Ciascuno di essi è descritto mediante una scheda sintetica di configurazione, la descrizione funzionale e il modello di programmazione. Sono indicati con n.s. (non specificati) i campi che non hanno senso e che non devono essere utilizzati nella configurazione di taluni dispositivi. Nel capitolo successivo sono presentati concreti esempi di configurazione di sistemi ASIM in cui sono impiegati gli oggetti descritti. Gli esempi sono corredati di programmi scritti in linguaggio assembler Motorola MC68000 da assemblare con lo strumento Asimtool e utilizzare per il test delle configurazioni.

## 4.1 Il dispositivo Processore

Il dispositivo processore è uno dei componenti principali di una configurazione di sistema ASIM. Allo stato ASIM supporta soltanto il processore M68000 che simula il processore industriale Motorola MC68000. Un processore in ASIM è configurabile mediante le seguenti codifiche:

| | |
|---|---|
| Name | Nome dell'oggetto processore (chiave) {M68000,...} |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggettonella configurazione ASIM |
| Type | CPU |
| Address1 | indirizzo da assegnare al reset all'User Stack Pointer (USP) |
| Address2 | indirizzo da assegnare al reset al Supervisor Stack Pointer (SSP) |
| BUS | Identificatore del bus a cui è connesso il processore |
| COM1 | n.s. |
| COM2 | n.s. |
| COM3 | n.s. |
| COM4 | n.s. |

**Tabella 4-1 Tabella di configurazione di un processore**

Allo stato, l'unico processore supportato in Asim è il processore MC68000. L'ambiente di sviluppo è fornito di un assemblatore assoluto (asm.exe utilizzabile anche da una finestra MSDOS) che produce un file eseguibile (immagine di memoria) ed uno di listing utilizzato da ASIM durante la fase di debugging. Per quel che riguarda le caratteristiche del processore MC68000 e del suo linguaggio assembler si rimanda ai manuali Motorola riportati in bibliografia e al manuale assembler fornito con l'ambiente ASIMtool.

L'assemblatore, a partire da un file simbolico (*.a68) genera un assoluto (*.h68) in linguaggio macchina in formato S-File di Motorola e un file di listing (*.lis) utilizzato anche dal simulatore per effettuare il trace simbolico del programma durante il processo di simulazione.

Quando la finestra attiva è quella associata ad un MC68000, tra le varie voci di menù compare anche quella *Processore* grazie alla quale è possibile accedere a tutti i comandi precedentemente descritti.

Nella finestra del processore (mediante l'attivazione di *Mostra Registri*) è possibile visualizzare, oltre a tutti i registri interni, anche il numero di clock macchina consumati per eseguire tutte le istruzioni dall'inizio fino al punto corrente della simulazione (i valori dei cicli di clock impiegati per eseguire ciascun istruzione sono stati tratti dal manuale del processore. Sebbene questo numero non tenga conto di eventuali cicli di attesa per l'accesso in memoria, moltiplicandolo per il periodo di clock della macchina reale che si sta simulando, si ottiene una buona stima dei tempi di elaborazione "reali" di un dato programma).

Per inserire in una configurazione un MC68000 occorre selezionare dal menù *Configura* il comando *Aggiungi CPU* e specificare nella finestra di dialogo (Fig. 4-1 Finestra di configurazione del processore M68000), nel campo Name il valore "M68000"; nel campo "Address 1" l'indirizzo del puntatore allo stack utente; nel campo "Address 2" l'indirizzo del puntatore allo stack supervisore; nel campo in "BUS" l'identificatore del bus o memoria cui è connesso il processore.

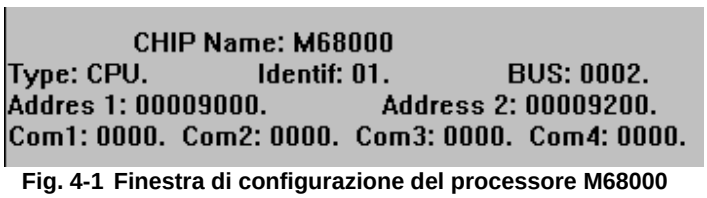



**Fig. 4-1 Finestra di configurazione del processore M68000**

I campi Com1..Com4 possono essere lasciati al valore zero non avendo per tale componente alcun significato.

Si osservi che il campo address2 contenente SSP (il puntatore allo stack supervisore) deve essere inizializzato con una locazione fisica di memoria RAM del processore; se così non fosse, in seguito ad un'operazione di accesso a tale indirizzo, verrebbe generato un "bus error" a cui il processore reagirebbe con il tentativo di salvataggio di alcuni registri sullo stack supervisore innescando, quindi, una condizione di stallo da cui non sarebbe possibile uscire se non con una operazione di "reset" del processore. Eventi anomali come "bus error", "istruzione illegale", "violazione di privilegio" vengono sempre segnalate.

Il parametro "BUS" fissa il collegamento del processore al bus di sistema mediante cui accedere agli altri componenti di una configurazione. Il parametro “Identif” è utilizzato dagli oggetti (device) della specifica configurazione per riferire il processore relativamente all’eventuale linea di interruzione ad esso connessa.

## 4.2 Modello di Princeton e di Harward

Negli anni '40 le ricerche rivolte alla costruzione di una macchina a programma in grado di eseguire calcoli in modo automatico, ha condotto alla formulazione di due architetture che hanno portato, negli anni seguenti, alla costruzione di calcolatori elettronici (prima a tubi elettronici, poi a transistor e successivamente a circuiti integrati, a varia scala di integrazione) noti come macchine di Harward e di Princeton (dal nome delle due università in cui venivano svolte tali ricerche). Alla formulazione della macchina di Princeton ha dato un determinante contributo J. v. Neumann (il modello, infatti, è comunemente noto come modello di v. Neumann). I due modelli, operano entrambi a programma memorizzato ma differiscono per quel che concerne il ruolo della memoria che, per la prima macchina, è logicamente e fisicamente composta da due memorie distinte di cui una, a sola lettura, destinata a memorizzare il programma da eseguire, e l’altra, a lettura-scrittura, i dati su cui quest'ultimo si trova ad operare. Il modello di Princeton utilizza, invece, un solo tipo di memoria a lettura-scrittura in cui vengono posti sia il programma che i dati. In tale macchina non si fa, pertanto, alcuna distinzione fra area dati e programmi. E' la fase elaborativa che determina la natura di una stringa di bit, memorizzata in una locazione di memoria, come istruzione o dato. Tale modello consentirebbe di effettuare a run-time la modifica del programma che si sta eseguendo.Tale ultimo modello è quello più diffusamente adottato per la costruzione dei processori che, pertanto, sono comunemente noti come processori di von Neumann e di cui il processore MC68000 ne è un esemplare.

### 4.2.1 Il processore M68000 e la gestione delle interruzioni

Un device può essere connesso a un dispositivo in grado di gestire un segnale di interruzione (un processore, un priority interrupt controller o genericamente un gestore di interruzioni). ASIM implementa il meccanismo base di gestione delle interruzioni di tipo vettorizzato. Tale meccanismo può essere utilizzato per la simulazione di un qualunque dispositivo in grado di gestire interruzioni quali uno specifico processore o un priority Interrupt Controller.

Il processore M68000, simulato in ASIM, impiega uno schema di interruzioni di tipo prioritario vettorizzato. Tre linee, IPL0, IPL1, IPL2 (IPL è acronimo di Interrupt request Priority Level) codificano l’evento interruzione in ingresso secondo il livello prioritario assegnato al dispositivo interrompente (un apposito encoder Hw esterno deve essere utilizzato per effettuare la codifica dei segnali di interruzione in uno dei livelli possibili). Il processore, in base al livello corrente memorizzato nella maschera delle interruzioni (bit 8-10) del suo registro di stato SR, abilita o meno l’evento di interrupt (PPL, Process Priority Level). Il 68000 gestisce sette livelli prioritari di interruzioni, il livello 7 è quello a maggiore prioritàin cui le interruzioni non sono mascherabili (NMI). Il livello 0 è quello a minore priorità, ovvero quello in cui le interruzioni sono completamente mascherate (disabilitate).Negli altri casi caratterizzati da una priorità 0<Pi<7, risultano abilitati tutti gli eventi interruzione di livello superiore al livello corrente memorizzato nel registro di stato (fa eccezione a tale regola, ovviamente, il livello 7 per le interruzioni NMI).

Il processore, all’arrivo di un’interruzione (sia essa di tipo Hw che Sw) da servire (e non mascherare) avvia un ciclo di bus mediante il quale acquisisce dalla periferica interrompente uno specifico numero di vettore espresso su 8 bit (l’area ROM dei vettori occupa, infatti, il primo K dello spazio indirizzi di memoria). Tale

numero, moltiplicato per 4 (mediante l'esecuzione di uno shift a sinistra di 2 posizioni), fornisce il puntatore ad una locazione di memoria ROM contenente il vettore associato all'Interrupt Service Routine (puntatore alla ISR). Il processore, una volta salvato il contesto minimo globale dell'elaborazione (registro di stato e PC), esegue l'istruzione in memoria puntata da tale vettore (cambio di contesto). Tutti i vettori sono di 32 bit (espressi su due word) e risiedono in memoria, in area dati dello stato supervisore (2^8<<4=1K). Fa eccezione il vettore di reset #0,che punta all'area programmi, di 4 word di cui 2 usate per inizializzare il PC e 2 word per inizializzare il supervisor stack pointer.

A supporto delle interruzione dei processori simulati, in ASIM è implementato un meccanismo di gestione interruzioni di tipo vettorizzato. Il meccanismo fa uso degli eventi del sistema operativo Windows. Ogni segnale di interruzione è associato ad un tale evento ed è descritto da una struttura dati di 2 byte contenenti i seguenti tre campi:*vector number* (bit b7-b0 del byte più significativo); *priorità*(bit b7-b4 del byte meno significativo);*linea d'interruzione*(bit b3-b0 del byte meno significativo).

Un dispositivo gestore di interruzioni (processore o PIC) può disporre di una o più linee di interruzione, a cui possono connettersi più sorgenti di interruzioni, a differente priorità. Se su una linea di interruzione, sono presenti più eventi interruzione simultanei, viene selezionato quello a maggiore priorità, e cioè quello il cui livello codificato è il più basso fra i 16 livelli ($0-$F) possibili. Al posto dei vettori si possono anche usare gli autovettori, che permettono di realizzare servizi di interruzione in cui il vettore viene generato internamente al processore (gli autovettori sono cablati nel processore, essi, di fatto, sono degli indirizzi memorizzati in hardware e associate staticamente alle linee di interruzione). Per essi, in Asim, il campo vector number non va specificato, essendo automaticamente associato al campo linea interruzione.

I 256 vettori di interruzione sono numerati da $00 a $FF esadecimale. Nel 68000, i vettori delle interruzioni dal numero 12 al 23 e dal numero 23 al 48, sono riservati (per future espansioni del processore), i vettori da 0 a 12, sono riservati alle eccezioni interne del processore, i 7 vettori 25-31 operano da pseudovettori (o autovettori). Essi sono posti agli indirizzi $64-$7C e assegnati staticamente alle linee di interruzione e hanno livello prioritario da 1 a 7. Tali vettori sono stati introdotti nel 68000 per supportare le periferiche operanti con la serie di processori a 8 bit 68xx e sono scomparsi nelle versioni successive del processore.

In ASIM gli autovettori vanno inizializzati direttamente nella configurazione dei vari dispositivi supportati. In particolare, la struttura dell'evento interruzione (vect-priorità-linea),precedentemente descritta, ha il campo vect posto a zero, il campo linea riporta un numero da 1 a 7 che indica l'autovettore mappato agli indirizzi da $60 a $67 in area ROM.

## 4.3 Il dispositivo Bus/Memoria

Il dispositivo Bus/Memoria è un componente di ASIM che simula il comportamento di un bus per l'interconnessione di dispositivi e dellememorie di tipo RAM e/o ROM. Il dispositivo offre differenti possibilità di configurazione a seconda dei valori posti nei vari campi di codifica il cui significato è il seguente:

| | |
|---|---|
| Name | MEMORIA (nome generico opzionale); |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto nella configurazione ASIM; |
| Type | MMU/BUS; |
| Address1 | Indirizzo base RAM; |
| Address2 | Indirizzo base ROM; |
| BUS | Identificatore di bus esterno verso cui esportare lo spazio indirizzi delle memorie RAM/ROM (dare visibilità del proprio spazio indirizzi verso dispositivi non appartenenti alla medesima configurazione BUS); |
| COM1 | Identificatore di bus esterno da cui importare la visibilità dello spazio di memoria RAM/ROM e anche degli oggetti device, cpu ad esso connessi (possibilità di avere visibilità di altro spazio indirizzi); |
| COM2 | dimensione (in esadecimale) della memoria RAM in blocchi da 1 Kbyte; |
| COM3 | dimensione (in esadecimale) della memoria ROM in blocchi da 1 Kbyte; |
| COM4 | n.s. |

**Tabella 4-2**

La classe Bus/Memoria è costituita da un unico oggetto bus. Il bus consente di effettuare lo scambio di dati, controlli e stati fra una risorsa sorgente ed una di destinazione nell'ambito di appositi cicli di trasferimento, sincronizzati da un clock o, se in modalità asincrona, da segnali di handshake In ASIM l'oggetto bus è anche "sostegno" per l'eventuale memoria (ROM/RAM)ad esso connessa. Un bus configurato senza memoria viene utilizzato per interconnettere altri oggetti (bus, memorie appartenenti ad altri bus, processori, ecc.).

Il dispositivo Bus/Memoria può essere configurato (Fig. 4-2-Finestra di configurazione del BUS/Memoria), inizializzando, opportunamente,i campi BUS, COM1, COM2 e COM3, come segue:

1. bus da utilizzare per connettere dispositivi di tipo qualsiasi;
2. modulo di memoria RAM e/o ROM;
3. bus con memoria RAM e/o ROM (unione dei casi 1 e 2);
4. interfaccia verso un altro bus, con o senza memoria RAM e/o ROM.

I campi Nome e Identif assumono il significato già discusso precedentemente. In particolare,a Nome va dato un valore a piacere che serve per chiamare il componente nella speifica configurazione e a distinguerlo da altri componenti simili. Identif è un campo in cui va posta la parola chiave MMU/BUS che individua il tipo di componente.

La memoria RAM e ROM configurabile è a parallelismo byte, ha uno spazio indirizzi definito mediante i campi Address1 e Address2, in cui vanno postirispettivamente gli indirizzi base della memoria RAM e di quella ROM solo se diversi da zero i campi COM2 e COM3che indicano le rispettive dimensioni in blocchi di 1K. Tali dimensioni sono espressein esadecimale. Il limite massimo di spazio di indirizzabilità ammesso in Asimper ciascun componente memoria è di $FFFF blocchi da 1K. Ciò corrisponde ad una memoria fisica di $3FF * $FFFF=64M locazioni. Nel definire le configurazioni delle memorie si deve fare attenzione a che gli spazi

indirizzi assegnati ai vari moduli di memoria non si sovrappongano. Se la RAM o la ROM non sono presenti,i campi COM2 o COM3 vanno posti a zero. Il parametro "Com4" non svolge alcuna funzione e valasciato al valorezero.

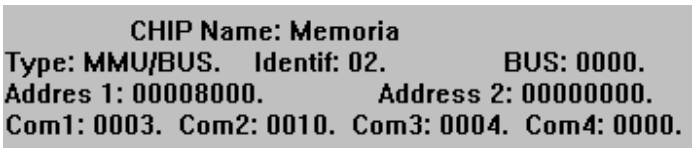



**Fig. 4-2-Finestra di configurazione del BUS/Memoria**

Il dispositivo Bus/Mem può essere usato, opportunamente configurato per suddividere il proprio spazio indirizzi fra banchi di memoria differenti (per realizzare, ad esempio, un banco di memoria da dedicare allo stack o banchi memoria contenenti sole aree dati o programmi) o per avere in un sistema S1 la visibilità dello spazio indirizzi di un sistema differente S2 (realizzando in tal modo una forma di condivisione della memoria fra S1 e S2). Per configurare un'architettura memoria del primo tipo, nel campo BUS va posto l'identificatore di tale componente Bus/Mem con sola funzione di memoria da interconnettere. Per configuarare, invece, un'architettura del secondo tipo, il sistema S1deve contenere nel campo COM1,il riferimento (l'identificativo) del componente Bus/Memdi S2da cui importare la visibilitàdelle memorie in esso presenti e degli altri oggetti device connessi al bus.

InFig. 4-3Esempio di visibilità del dispositivo MMU/BUSè riportata una configurazione con tre dispositiviBus/Mem aventi identificativi id-1, id-4, id-7interconnessi. In essa il sistema S1 con BUS-1, avendo i campi BUS=0 e COM1=0,non esporta il proprio spazio indirizzi e non accede a spazi indirizzi di altri sistemi. Il sistema S2 con BUS-4,avendo i campi BUS=7 e COM1=1, esporta verso il sistema S3 con Bus-7 il proprio spazio indirizzi e importa dal sistema S1 con BUS-1 lo spazio indirizzi delle memorie e degli oggetti RAM-STACK e Terminal a questo connessi. Il sistema S3 con BUS-7 ha BUS=4 e COM1=0 per cui esporta verso il BUS-4 lo spazio indirizzi delle RAM e ROM e non importa spazio indirizzi da alcun oggetto. Se il sistema S3 fosse stato configurato con BUS=0 e COM1=4, esso non esporterebbe il proprio spazio indirizzi RAM/ROM ma importerebbe dal sistema S2 con BUS4 lo spazio indirizzi delle memorie e degli oggetti a questo connessi. Tali funzionalità, unitamente a quelle messe a disposizione dall'oggetto 1TO4BUSInterface, espansore di bus, consentono di simulare configurazioni di sistemi complessi con varie gerarchie di BUS al loro interno.

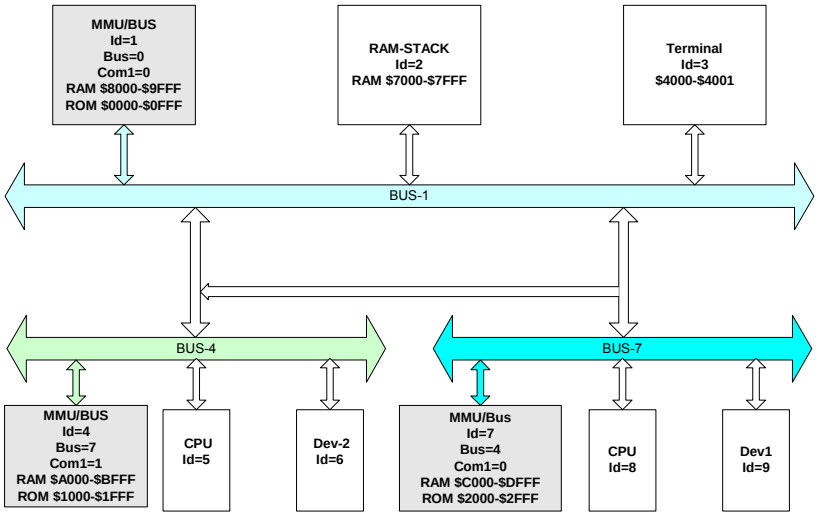



**Fig. 4-3Esempio di visibilità del dispositivo MMU/BUS**

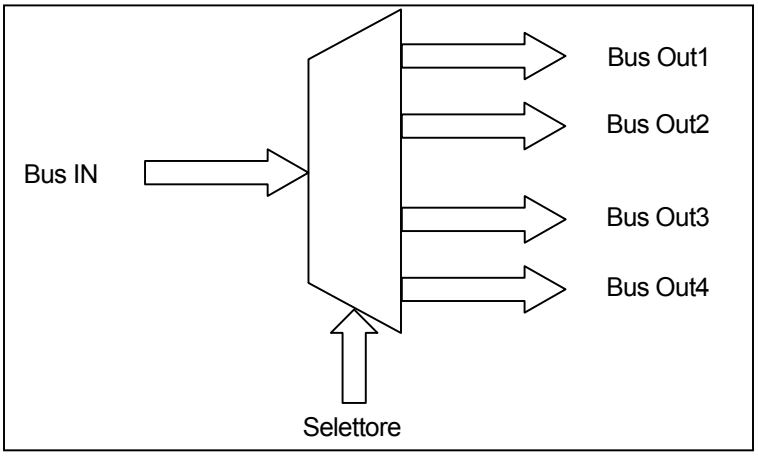
## 4.4 Il dispositivo 1TO4BusInterface

I campi relativi alla configurazione di un bus sono così codificati:

| | |
|---|---|
| Name | 1TO4BusInterface |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Valore iniziale dello spazio indirizzi gestito dall'oggetto; |
| Address2 | Valore finale dello spazio indirizzi gestito dall'oggetto; |
| BUS | Identificatore di bus esterno connesso all'ingresso; |
| COM1 | Identificatore di bus esterno connesso all'uscita 1, se 0 la linea è n.c.; |
| COM2 | Identificatore di bus esterno connesso all'uscita 2, se 0 la linea è n.c.; |
| COM3 | Identificatore di bus esterno connesso all'uscita 3, se 0 la linea è n.c.; |
| COM4 | Identificatore di bus esterno connesso all'uscita 4, se 0 la linea è n.c.; |

**Tabella 4-3**

L'oggetto 1TO4BusInterface è un demultiplexer (Fig. 4-4-1To4bus interface - demux di bus)con linee bus (1 ingresso-4 uscite) full duplex in grado di espandere le possibilità di interconnessione dell'oggetto BusMem e generare configurazioni ASIM multisistema. Esso è funzionale ad ASIM per la simulazione di configurazioni




complesse e non corrisponde ad alcun concreto dispositivo commerciale.

**Fig. 4-4-1To4bus interface - demux di bus**

La selezione della linea(selettore) su cui instradare la connessione è fatta in base all'appartenenza dell'indirizzo presente sul bus in ingresso ad uno dei 4 intervallidi indirizzi fisici definiti staticamentedai parametri posti nei campi COM1..COM4.

Impiegando tale meccanismo 1TO4BusInterface consente la simulazione dischemi architetturali multisistema e di strutture "crossbar".

Il dispositivo, limitandosi a partizionare lo spazio indirizzi in ingresso secondo un semplice schema (filtro) statico, non necessita di registri interni per la programmazione e, pertanto, non occupa alcuna locazione d

memoria e non va mappato. Il menù *Device* ad esso associato, diversamenteda quello utilizzato per gli altri componenti, presenta un unico comando *Trace Accessi* che consente di registrare e visualizzarei cicli di accesso in lettura, modifica e scrittura. La registrazione degli accessi viene avviata una volta selezionato il comando e permane fin quando non lo si deseleziona. Un buffer interno memorizzagli ultimi trenta cicli che possono essere visualizzati facendo scorrere i dati mediante l'apposita barra di scorrimento.

La definizione dei parametri del dispositivo avviene attraverso la finestra di dialogo cui si accede dal menù *Configura* con il comando *Aggiungi Device*. In particolare:

il "Nome" è da indicare; il campo Identificatore è 1TO4BUSINT;nel campo BUS va posto l'identificatore del bus connesso alla linea di ingresso di 1to4BUSINT;"Indirizzo1" e Indirizzo 2" delimitano lo spazio di indirizzabilità relativo al bus in ingresso. Una richiesta con indirizzo compreso in detto intervallo, è instradata verso uno dei quattro bus in uscita. Se l'operazione fallisce, per indisponibilità del bus (bus non connesso) è restituito al richiedente un "Bus error".

Gli identificatori dei bus o di altri 1to4BUSINT connessi sulle linee di uscita vanno postinei campi COM1..COM4. Le linee non connesse ad alcun bus devono avere i relativi COMi posti a zero e devono seguire nell'ordine le linee connesse se si intende suddividere fra esse l'intero spazio indirizzi. Lo spazio di indirizzabilità del bus in ingresso (indirizzo2-indirizzo1+1) è, infatti, suddiviso in intervalli uguali fra le linee di uscita effettivamente connesse a bus. Tale numero coincide con la posizione i dell'ultimo COMi non nullo, anche se talune linee, di posizione inferiore, non sono connesse. In Fig. 4-5-Intervalli dello spazio indirizzi al variare di parti N, si riporta lo schema di suddivisione dello spazio indirizzi fra le linee.

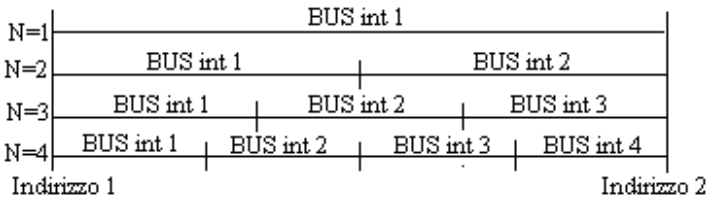



**Fig. 4-5-Intervalli dello spazio indirizzi al variare di parti N**

```
I=address1-adress2
Indirizzo di inizio intervallo i-esimo=(address1 + (i-1)*I/n)
Indirizzo di fine intervallo i-esimo=(address1 + i*I/n) -1
```

Il meccanismo di gestione degli indirizzi implementato nel dispositivo 1to4BUSINT consente di realizzare oltre che alla funzione di multiplexing anche il filtraggio dello spazio indirizzi. Si supponga, ad esempio,di configurare il dispositivo con COM1=COM2=COM3=0 e COM4 ≠0. In tal caso, lo spazio di indirizzabilità relativo al bus di ingresso verrà suddiviso in quattro parti uguali (BUS int1..BUS int4) ma soltanto gli indirizzi compresi nel quarto intervallo BUS int4, sarà ruotato verso le linee d'uscita. Gli indirizzi compresi negli intervalli BUS int1, BUS int2, BUS int3 genereranno un Bus error ove si dovessero presentare.

Il dispositivo è dotato, inoltre, della capacità di monitorare l'attività di accesso su una delle linee bus (sia in ingresso che in uscita). Ad esempio, 1to4BUSINT può essere utilizzato con una sola connessione in uscita al fine di monitorare gli accessi su un bus da parte di un processore. E' anche possibile assegnare più linee di uscita allo stesso bus (specificando più volte lo stesso identificatore). Ciò è utile quando ad un bus sono connessi una memoria e dei dispositivi con indirizzi separati e gli indirizzi intermedi sono, invece, assegnati ai dispositivi di un altro bus.

## 4.5 Il dispositivo generatore di interruzioni 1to4INTGEN.

Il dispositivo 1to4INTGEN è un generatore di interruzioni verso i dispositivi in grado di gestirli (in ASIM processori e PIC). Esso è composto da5 generatori: un generatore di interruzioni programmabile e quattro timer ed è visto come 20 ($13+1) locazioni consecutive di memoria.Tale dispositivo èimpiegatoper generare interruzioni ad intervalli di tempo prefissati (mediante la programmazione dei parametri dei timer) o per generare, a programma, i segnali di interruzione verso lo specifico oggetto gestore di interruzioni.

I campi relativi alla configurazione di 1to4INTGEN sono così codificati:

| | |
|---|---|
| Name | 1to4INTGEN. |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione; |
| Address1 | Valore iniziale dello spazio indirizzi dell'oggetto; |
| Address2 | Valore finale dello spazio indirizzi (Address1 + $13); |
| BUS | Identificatore di bus a cui il dispositivo è connesso; |
| COM1 | Identificatore dei dispositivi in grado di gestire le interruzioni, se 0 la linea è n.c.; |
| COM2 | Identificatore dei dispositivi in grado di gestire le interruzioni, se 0 la linea è n.c.; |
| COM3 | Identificatore dei dispositivi in grado di gestire le interruzioni, se 0 la linea è n.c.; |
| COM4 | Identificatore dei dispositivi in grado di gestire le interruzioni, se 0 la linea è n.c.; |

**Tabella 4-4**

Per connettere 1to4INTGEN ad una macchina da simulare, occorre specificare i parametri nella finestra di dialogo, cui si accede dal menù *Configura* con il comando *Aggiungi Device*.

"Nome elemento" è 1TO4INTGEN, "Address1" deve essere posto a indirizzo pari e rappresenta l'indirizzo del registro I (indirizzo relativo 00); tutti gli altri registri hanno indirizzo dato da "Address 1" + indirizzo relativo (i valori dell'indirizzo relativo sono quelli di Fig. 4-6Modello di programmazione di 1to4intgen). "Address 2" deve essere uguale ad "Address 1" + $13. In "BUS" va l'Identificatore del bus a cui il dispositivo è connesso. I parametri "Com1", "Com2", "Com3" e "Com4" (corrispondenti, rispettivamente, alle linee 1, 2, 3 e 4) consentono di specificare le connessioni verso i dispositivi gestoridelle interruzioni. Ad esempio Com2=3 assegna la linea di uscita 2al dispositivogestore di interruzioni di identificatore 3. Se una linea non è connessaad alcun dispositivo il relativo valore di COMi va posto a zero. Più linee possono essere connesse allo stesso dispositivo.

Il dispositivo 1to4INTGEN ha dieci registri r0-r9 a sedici bit di cui i primi due dedicati al generatore di interruzioni, i rimanenti 2x4 ai 4 timer. La Fig. 4-6Modello di programmazione di 1to4intgen, mostra l'insieme dei registri e la loro funzione, la Fig. 4-7Finestra associata al dispositivo, la finestra di stato-controllo e gli indirizzi corrispondenti a ciascun registro (gli indirizzi sono relativi all'indirizzo base).

### 4.5.1 Funzionamento del generatore di interruzioni programmabile

L'1to4INTGEN è dotato di 4 linee di interruzione e può connettersifisicamente con quattro dispositivi in grado di gestire interruzioni (processori o priority interrupt controller). Il registro **I**,posto all'indirizzo relativo $00 del dispositivo, contiene la struttura dell'evento interruzione che si vuole inviare. Un processore che vuole inviare un'interruzione ad un altro (o anche a se stesso) deve specificare il tipo di interruzione assegnando al registro **I**i valori della linea, priorità e del vector number (le due cifre esadecimali del byte meno significativo specificano linea e priorità; le due cifre esadecimali del byte più significativo specificano il vector number). Una volta definita l'interruzione, per inviarla, occorre settare l'apposito bit del registro di indirizzo relativo $02associato alle 4 linee di interruzione verso i gestori codificati in com1-com4.

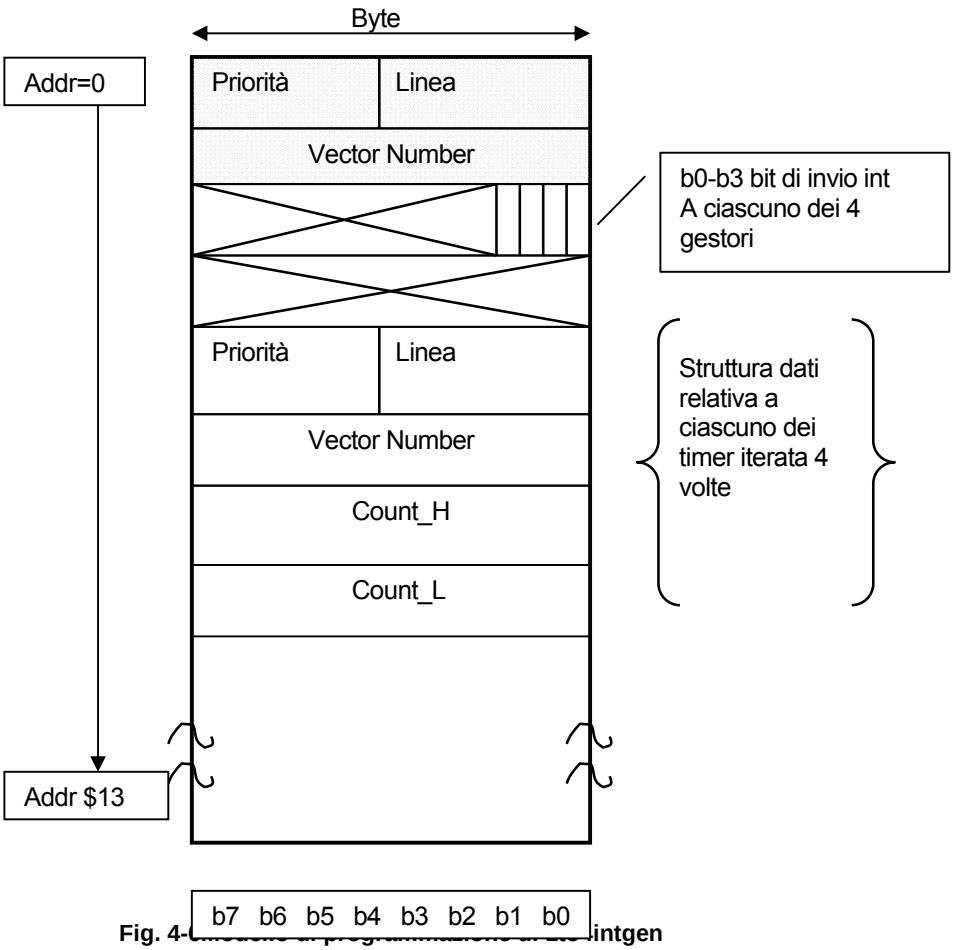



**Fig. 4-intgen**

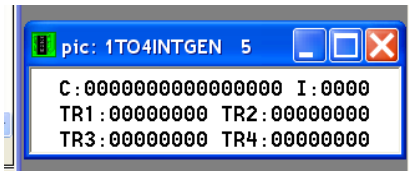



**Fig. 4-7Finestra associata al dispositivo**

E' possibile inviare, contemporaneamente, la stessa interruzione anche a tutti e quattro i dispositivi connessi, ponendo ad 1 tutti e quattro i bit del registro.Se ad una delle linee di uscita non è connesso alcun gestore delle interruzioni, l'interruzione non viene inviata; se più di una linea è connessa allo stesso dispositivo e vengono posti ad 1 i corrispondenti bit, vengono inviate al gestore tante interruzioni dello stesso tipo, quante sono le linee ad esso connesse.

### 4.5.2 La sezione con i 4 timer

I quattro timer TR1-TR4 sono identici e sono collegabili alle linee di uscita 1-4. A ciascuno di essi sono associate due word di cui la meno significativa, specifica l'interruzione (così come detto per il registro I) e la più significativa, definisce il valore iniziale del contatore. Se V è la velocità del clock associato al dispositivo (di default V = 1, ma può essere modificata), ogni V colpi di clock il valore del contatore si decrementa di 1. Quando questi arriva a zero, viene attivata un'interruzione sulla linea specificata e fermato il contatore (che, se programmato, viene re-inizializzato). Tale interruzione viene inviata al gestore d'ìinterruzioni , se questo è connesso sulla linea in questione.

Poiché i dispositivi connessi alle linee di uscita del 1to4INTGEN non devono necessariamente essere diversi, è possibile, ovviamente, associare allo stesso dispositivo anche tutti e quattro i timer. In tal caso la scelta di quale interruzione servire va fatta sulla base della priorità.

## 4.6 IL Priority Interrupt Controller

Il Priority Interrupt Controller (PIC) è un dispositivo in grado di estendere la capacità di gestione delle interruzioni propria del processore, effettuandone la gestione prioritaria e la mascheratura delle interruzioni. La configurazione di tale dispositivo in ASIM è caratterizzata dalla seguente codifica dei campi di configurazione del dispositivo:

| | |
|---|---|
| Name | I8259PIC |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Indirizzo base per il dispositivo |
| Address2 | Indirizzo base+1 |
| BUS | Ident. del bus a cui il dispositivo è connesso |
| COM1 | Ident. del dispositivo PIC o CPU gestore delle interruzioni |
| COM2 | \|vettore (b15-b8)\|priorità (b7-b4)\|linea Int (b3-b0)\| |
| COM3 | N.U. |
| COM4 | N.U |

**Tabella 4-5**

Nella tabella di configurazione:

**Nome** è utilizzato per specificare il tipo di componente da inserire, nel caso in esame deve essere *"I8259PIC";* **Identificatore** deve essere unnumero *compreso tra 01 ed FF*e viene utilizzato dal simulatore per riferire il dispositivo; **Indirizzo 1** rappresenta l'indirizzo più basso di selezione del componente; in questo caso deve essere un *numero pari;* **Indirizzo 2** definisce l'indirizzo più alto per indirizzare il componente, nel caso in esame deve essere *uguale ad Indirizzo1 + 1.*Questi ultimi due parametri permettono di definire per quali indirizzi il decodificatore di indirizzi di sistema attiva la linea **CS.**

**BUS** determina l'*Identificatore del bus* a cui è connesso il dispositivo. La connessione delle linee **bus dati**, **A0**, **RD** e **WR** del controllore delle interruzioni al processore è realizzata nel simulatore, scegliendo per **BUS** l'identificatore di un componente MMU/BUS, a cui è stato collegato il dispositivo di tipo CPU.

**COM1** definisce l'*Identificatore del gestore delle interruzioni*, cioè il componente (di tipo processore o PIC) al quale trasmettere le interruzioni non mascherate.**COM2** i 2 byte di tale campo (xxyz) ospitano un descrittore di interruzione (vector number, priorità e linea d'interruzione,). Il vector number è generato dal PIC a partire dal valore base del vettore, memorizzato in un suo registro,a cui si aggiunge lo spiazzamento dato dal numero della specifica interruzione pervenutain una delle sue 8 linee.Il campo vector number (ultime due cifre esadecimali), in tal caso, non va specificato.**COM3 e COM4**non sono utilizzati.

La finestra associata al PIC in ASIM è presentata in fig.4.8

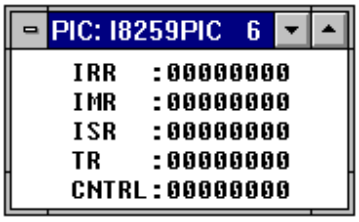



**Fig. 4-8Finestra ASIM con i registri del PIC**

Il PIC implementato è derivato, nella logica operativa, dal componente Intel 8259, ma presenta un numero di registri e di modi di funzionamento limitato rispetto ad esso. LaFig. 4-9modello di programmazione del PIC,riporta il modello di programmazione e le linee fisiche simulate. Tale dispositivo è stato progettato dall'Intel per la famiglia di processori 80x86 al fine di estendere le capacità di gestione delle interruzioni del processore. L'8259 presentaverso il processore una sola linea di interruzione INTR e una linea di acknowledge INTA epresenta in ingresso fino a 8 linee di interruzione differenti, con livelli di priorità multipli (fino ad 8 livelli di priorità). Più dispositivi 8259A possono essere connessi in cascata (fino a 8 per la gestione di max 64 linee di interruzione). Accetta richieste di interruzioni dai dispositivi di I/O connessi alle sue linee; determina, a seconda dell'algoritmo di gestione prioritaria selezionato, quale delle interruzioni simultaneamente attive, ha la priorità più alta per il servizio; trasmette l'interruzione al processore mediante la linea INTR; attende che il processore risponda con l'acknowledge INTA; trasmette al processore il vettore dell'interruzione associato alla linea selezionata. A questo punto Il processore avviail servizio dell'interruzione.

La gestione prioritaria si basa sui seguenti tre schemi programmabili per tramite di un'opportuna codifica del registro di controllo CNTRL:

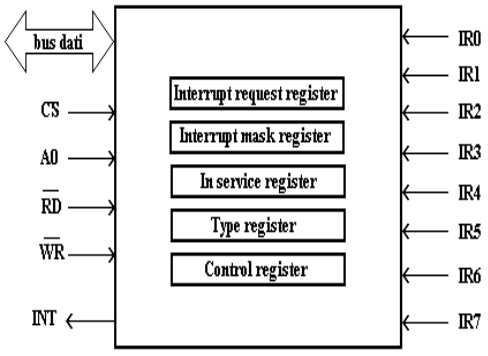
- Fully nested: le richieste di interruzione sono ordinate secondo uno schemaa priorità fissa che va da IR0 a IR9, con IR0 la linea più prioritaria;
- Round Robin: è uno schema prioritario a rotazione. La linea di interruzione più prioritaria servita diventa la meno prioritaria dopo il servizio;
- Maschera delle interruzioni: consente l'inibizione o l'abilitazione delle linee di interruzione a programma.

Il modo secondo cui può operare il PICdeve essere configurabilein fase di inizializzazione del dispositivo (dopo il suo reset) ma può essere cambiato anche dai programmi di gestione.

**Fig. 4-9modello di programmazione del PIC**

### 4.6.1 Modello di programmazione

Il dispositivo PIC simulato occupa due locazioni di memoria con cui indirizzare i 5 registri a 8 bit CNTRL, IRR, ISR, TR, IMRa disposizione del programmatore ed il cui significato è il seguente:




- Il registro CNTRL (Control Register) permettere di controllare il comportamento del dispositivo;
- Il registro IRR (Interrupt Request Register) memorizza i segnali di interruzione;

- Il registro ISR (In Service Register) presenta al gestore delle interruzioni i segnali interrompenti in modo ordinato rispetto alla maschera e alla priorità corrente;
- Il registro TR (Type Register) memorizza la parte base dell'indirizzo del vettore
- Il registro IMR (Interrupt Mask Register) permette di mascherare singolarmente le interruzioni.

Il banco di 8 registri da 1 bit, detti Interrupt Request Reg (**IRR**),riceve in ingresso le 8 linee di interruzione (IR0-IR7) provenienti dalle periferiche collegate,memorizzandone il valore. IRR è posto in cascata con un circuito per la gestione prioritaria delle interruzioni e con un altro banco di 8 registri da 1 bit, detto In Service Reg (**ISR**) in cui sono memorizzati solo i segnali di interruzione da servire. Fra questi è selezionato quello più prioritario. Il blocco addetto alla gestione prioritaria determina le priorità dei bit settati in IRR. Il bit più prioritario (la linea a priorità maggiore è la linea 0; la priorità decresce fino alla linea 7) viene selezionato e scritto nel registro ISR. Non possono essere attivate interruzioni in arrivo su linee meno prioritarie di quelle in cui è settato uno dei bit di ISR. Qualora sia abilitato l'apposito flag nel registro CNTRL, i bit di ISR sono cancellati automaticamente all'atto del servizio; in caso contrario sarà compito della routine che serve l'interruzione cancellare lo specifico bit mediante il registro CNTRL.

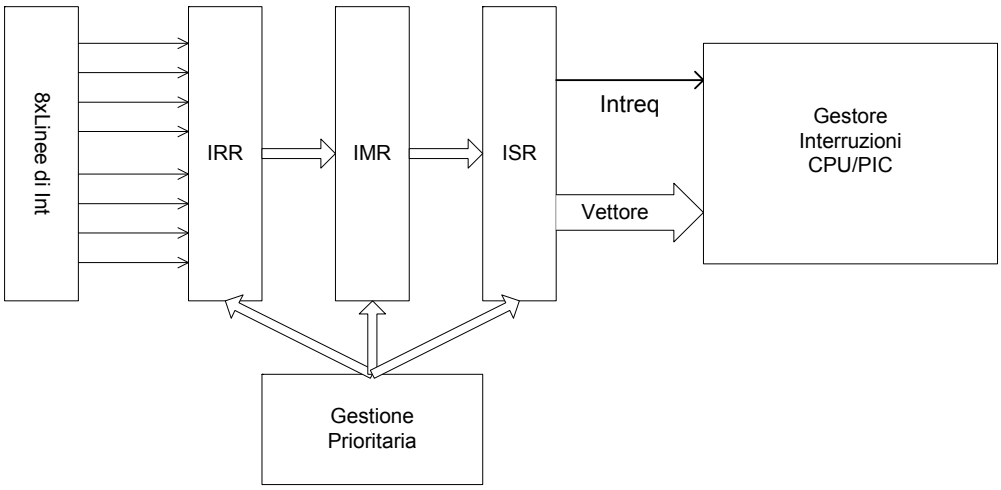



**Fig. 4-10Modell funzionale e di programmazione del PIC**

Il registroInterrupt Mask (**IMR**) di 8 bit,serve da mascheradelle interruzioni. I bit di mascheramento corrispondono in posizione a quelli del registro IRR; se il bit è settato a 1 la corrispettiva linea di interruzione è mascherata.Il registro Type Register (**TR**) di 8 bit memorizza (nel formato aaaaannn) nei 5 bit più significativi (in posizione aaaaa), il valore base del vector number (compreso tra 0 e 2**5=32); nei successivi 3 bit meno significativi (nnn) uno piazzamento settatoautomaticamente in base alla linea interrompente (ad esempio se è interessata la linea 5 i 3 bit saranno automaticamente settati ad 101). Pertanto, tutte le interruzioni associate ad uno specifico PIC presentano vettori consecutivi composti da un unico indirizzo base di 5 bit ed uno spiazzamento di 3.

Una richiesta d'interruzione che arriva sulla linea i-esima, pone ad 1 il relativo bit del registro **IRR** e, se non mascherataeselezionata,è spedita al gestore delle interruzioni collegato al PIC (una CPU o un altro PIC). Il PIC trasmette, quindi, il relativo vettore (nel formato aaaaannn). Il bit i-esimo di IRR è automaticamente cancellato per rimuovere la causa d'interruzione.

Tra tutte le interruzioni che arrivano da altri device sulle linee di richiesta IR0-IR7 del componente, solo quella con priorità maggiore e non mascherata sarà da esso scelta. Se su una linea sono collegati più device che generano interruzioni contemporaneamente, viene scelta quello a priorità maggiore. ASIMopera secondo un meccanismo che simula un circuito Daisy Chain con 16 livelli di priorità per linea; il valore del livello prioritario viene spedito al PIC insieme al numero della linea.

Selezionata l'interruzione con la più alta priorità non mascherata, il dispositivo modifica i tre bit meno significativi del registro **TR** per generare lo specifico vettore relativo alla linea di interruzione interessata da spedire al processore.

Dopo aver trasmesso al processore l'interruzione sulla linea i-esima, il bit i-esimo di **ISR** viene posto ad 1, il corrispondente di **IRR** viene posto a 0. Se il bit **AEOI** del registro **CNTRL** (bit 4), è pari ad 1, allora anche il bit i di **ISR** viene posto a 0 altrimenti, sarà compito della routineche serve l'interruzione annullarlo, per consentire il futuro servizio di interruzioni di livello prioritario pari o inferiore. Tale operazione va fatta settando opportunamente il valore di i nei tre bit meno significativi di **CNTRL** e ponendo il bit 3 **EOI** ad 1.

| bit | significato |
|---|---|
| 0,1,2 | determinano il bit n da cancellare in **ISR** |
| 3 | Bit **EOI** (End Of Interrupt), se posto ad 1 fa cancellare il bit n-esimo del registro **ISR** puntato dai bit b0-b2 |
| 4 | Bit **AEOI** (Automatic End Of Interrupt), se posto ad 1 fa cancellare automaticamente il bit in **ISR** dopo la trasmissione dell'interruzione |
| 5 | non utilizzato |
| 6 | Bit **RIS** (Register Interrupt Selector), se il bit **RR**=1 seleziona l'accesso in lettura del registro **ISR** se è posto a 1 e di **IRR** se a 0 |
| 7 | Bit **RR**, permette se posto ad 1 la lettura dei registri **ISR** o **IRR** |

**Tabella 4-6 - Significato dei bit di CNTRL**

### 4.6.2 Modalità di selezione dei registri

I registri TR ed IMR condividono lo stesso indirizzo dispari (bit0=1). Al reset è preselezionato il registro TR che può essere acceduto in sola scrittura, i successivi accessi a tale indirizzo sia in lettura che scrittura selezioneranno il registro IMR. L'accesso all'indirizzo pari del dispositivo (bit0=0) seleziona, in scrittura, il registro di controllo CNTRL, in lettura, il registro di controllo CNTRL se il bit7 è posto a 0, i due registri IRR e ISR se posto ad 1. La scelta fra tali ultimi due registri è fatta in base al valore del bit 6 che se posto a 1 seleziona il registro ISR se a 0 il registro IRR. In Tabella 4-6 - Significato dei bit di CNTRL, è riportata la struttura del registro di CNTRL, in Tabella 4-7-indirizzamento dei registri, lo schema di selezione dei registri sopra descritto.

| Indirizzo | Tipo di accesso | RR | RS | Registro |
|---|---|---|---|---|
| Pari | W<br>R | 0/1<br>0 | 0/1<br>0/1 | **CNTRL** |
| Pari | R | 1 | 0 | **IRR** |
| Pari | R | 1 | 1 | **ISR** |
| Dispari | W | 0/1 | 0/1 | **TR*** |
| Dispari | W/R | 0/1 | 0/1 | **IMR** |

** accessibile solo quando il dispositivo viene resettato*

**Tabella 4-7-indirizzamento dei registri**

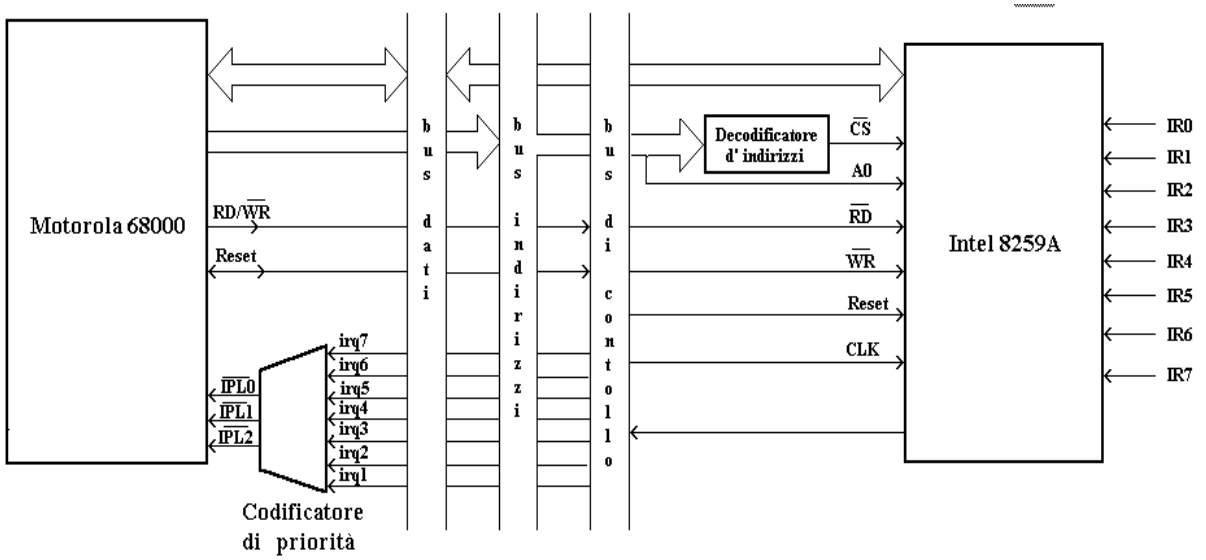
### 4.6.3 Collegamento del dispositivo con altri componenti di ASIM

Il PIC può collegarsi in una configurazione ASIM ad:

- un processore in grado di programmare il componente;
- un device,per la gestione delle proprie interruzioni;
- un altro PIC per creare una gerarchia di PIC ed aumentare il numero di linee di interruzioni da dedicare ai dispositivi.

In

Fig. 4-11 Collegamento del PIC al processore MC68000, è mostrata una configurazione tipica che impiega il priority interrupt controller. Il **bus dati** è utilizzato dal processore per il trasferimento dei dati da e verso il componente. La linea **CS** è utilizzata per la selezione del dispositivo; i registri interni sono selezionati dal bit meno significativo del bus indirizzo **A0**, dai segnali di lettura-scrittura **RD** e **WR**, oltre che dallo stato di taluni bit del registro di controllo. La linea d'interruzione **INT** trasmette al processore, o all'eventuale PIC, la richiesta




d'interruzione selezionata dal dispositivo, tra quelle presenti in ingresso. Il processore MC68000 commerciale

**Fig. 4-11 Collegamento del PIC al processore MC68000**

impiega un segnale di **intack** per avviare un ciclo di bus in cui la periferica fornisce il proprio vector number. Tale segnale non è simulato in ASIM in quanto non è gestito un simile ciclo di bus. Si sopperisce a ciò con un meccanismo che consente ad un dispositivo interrompente di inviare un messaggio di 16 bit contenente il valore della tripla vettore-priorità-linea precedentemente discussa.

Sulla destra dello schema sono mostrate le linee che collegano il PIC ad un device di I/O, 8 linee delle interruzioni a priorità decrescente da **IR0** a**IR7**.

### 4.6.4 Programmazione del PIC

L'uso del PIC è riportato nell'esempio fornito con ASIM. Esso si basa sulla configurazione, di seguito riportata, che simula un sistema composto da un processore 68000, una RAM di 16K, una ROM con inizializzati 8 vettori di interruzione, un PIC ed un 1to4intgen per la generazione delle interruzioni a programma e mediante uno dei 4 timer.

```
Configuration name: pic.cfg
CHIP Name: MEMORY
Type: MMU/BUS. Identif: 01. BUS: 0000.
Address 1: 00008000. Address 2: 00000000.
```

```
Com1: 0000. Com2: 0010. Com3: 0008. Com4: 0000.
CHIP Name: M68000
Type: CPU. Identif: 02. BUS: 0001.
Address 1: 00009000. Address 2: 00009200.
Com1: 0000. Com2: 0000. Com3: 0000. Com4: 0000.
CHIP Name: TERMINAL
Type: Device. Identif: 03. BUS: 0001.
Address 1: 00002000. Address 2: 00002001.
Com1: 0006. Com2: 0006. Com3: 0007. Com4: 0000.
CHIP Name: 1TO4INTGEN
Type: Device. Identif: 04. BUS: 0001.
Address 1: 00002004. Address 2: 00002017.
Com1: 0006. Com2: 0006. Com3: 0006. Com4: 0006.
CHIP Name: 1TO4INTGEN
Type: Device. Identif: 05. BUS: 0001.
Address 1: 00002018. Address 2: 0000202B.
Com1: 0006. Com2: 0006. Com3: 0006. Com4: 0006.
CHIP Name: I8259PIC
Type: Device. Identif: 06. BUS: 0001.
Address 1: 0000202C. Address 2: 0000202D.
Com1: 0002. Com2: 0001. Com3: 0000. Com4: 0000.
```

**Tabella 4-8 - Configurazione ASIM PIC**

La configurazione riportata in Tabella 4-8 - Configurazione ASIM PIC, e la Fig. 4-12Architettura utilizzata per l'esempio PIC, mostrano una configurazione composta da un processore 68000, un modulo di memoria RAM di 16 Kbyte e ROM di 1Kbyte inizializzata con i vettori delle interruzioni, un dispositivo Terminal, due generatori di interruzioni e un PIC.

L'inizializzazione del PIC (a seguito di un Reset) viene effettuata scrivendo

- nel registro **TR** il valore base del vector number da trasmettere al processore mediante un accesso in scrittura all'indirizzo dispari; dopo tale scrittura tutti gli accessi in scrittura all'indirizzo dispari selezioneranno sempre il registro **IMR**;

- all'indirizzo pari nel registro **CNTRL** un valore tale da definire il modo in cui viene cancellato il bit in **ISR** (cioè in modo automatico o dalla routine che serve l'interruzione). Come detto, se la cancellazione del bit in **ISR** non è automatica, è compito della ISR cancellarlo scrivendo un byte in CNTRL che contenga nei tre bit meno significativi il numero del bit da cancellare ed il valore 1 nel bit 3.

- il registro **IMR** all'indirizzo dispari per, eventualmente, mascherare le singole linee di richieste di interruzione.

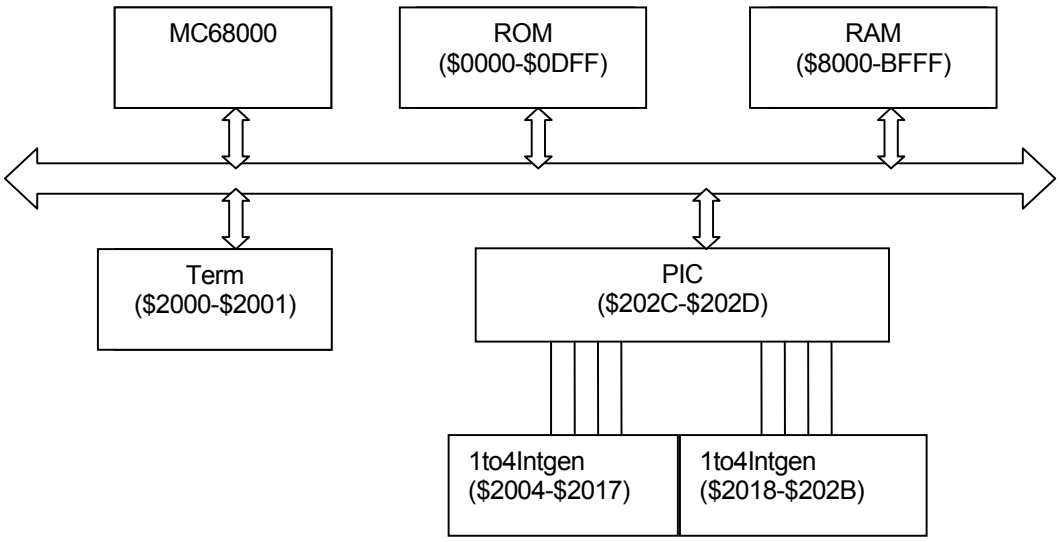



**Fig. 4-12Architettura utilizzata per l'esempio PIC**

Si riporta di seguito un esempio di una sequenza di inizializzazione del PIC in cui in A0 è posto l'indirizzo base del device:

```
move.b          #$40,1(A0)  //vector number $40 in TR
move.b          #$10,(A0)   //seleziona in CNTRL la cancellazione
                            //automatica del bit in ISR
move.b          #$F0,1(A0)  //scrive la maschera delle interruzioni
                            // per le linee da 4 a 7
```

Il dispositivo 1to4intgen presenta, come detto, al suo interno due componenti "generatori di interruzioni" programmabili. Ciascuno di essi permette di gestire 4 interruzioni verso un processore o PIC cui è collegato. Ciascuna linea di interruzione è caratterizzata dai tre parametri {vettore int (0-255), liv. prioritario (Irq0-Irq7), linea int (Ir0-Ir7)} e da un flag che se ad 1 attiva l'interruzione.

Il dispositivo contiene anche 4 timer indipendenti in grado di generare eventi interruzioni. Ciascun timer è programmabile mediante una coppia di registri di cui quello ad indirizzo più basso serve a specificare la linea di interruzione, così come descritto sopra, quello ad indirizzo più alto il valore di un contatore a decrescere che quando raggiunge lo zero genera un'interruzione sulla linea a cui è connesso. Ciascun dispositivo 1to4intgen consente, pertanto, mediante i suoi 4 timer, di generare 4 eventi autonomi ciclici e due eventi a programma.

Il dispositivo PIC acquisisce 8 linee di interruzioni dai due 1to4intgen e li invia codificate su 3 bit (IPL0, IPL1, IPL2) al processore.

Il codice assembler deve inizializzare i due 1to4intgen e il PIC e i vettori di interruzioni e definire le 8 ISR che dovranno gestire gli 8 eventi interruzione inviate dai generatori.

Le ISR sono identiche e si limitano a stampare al video di TERM l'identificativo dell'interruzione ricevuta e gestita.

# Il dispositivo DMA

Un processore esegue istruzioni ad una velocità legata al suo clock (ad esempio n cicli v.Neumann/k cicli di clock con k>n). Il numero di istruzioni al secondo è, allo stato attuale della tecnologia, dell'ordine di Giga istruzioni/sec ($10^9$ GIPS) per processori commerciali. La velocità del processore è quasi sempre, a parte taluni casi particolari, molto maggiore rispetto a quella di un dispositivo, specialmente di dispositivi elettromeccanici quali le stampanti, i plotter, ecc.

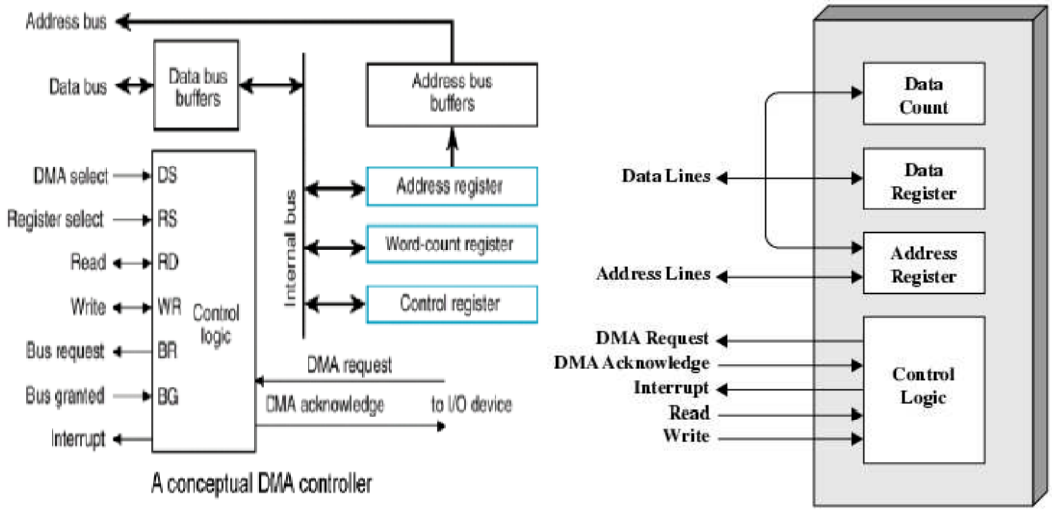
Le operazioni di trasferimento da e per un device/periferica, sono caratterizzate da un tempo di setup iniziale, necessario ad avviare l'operazione stessa e indipendente dal numero di bit o byte da trasferire; da un tempo di trasferimento del dato/dati; da un tempo di chiusura dell'operazione. I trasferimenti possono essere di tipo bit oriented, e cioè il trasferimento consiste nello scambio di una stringa di bit di lunghezza n spesso variabile ad ogni trasmissione, o di tipo byte oriented (o a carattere) in cui il trasferimento consiste nello scambio di n byte (singolo trasferimento di byte o trasferimento di n byte). Un trasferimento può essere effettuato o in modalità "polling" o in mediante "interrupt". Nel primo caso il processore è impegnato per tutta la durata del trasferimento o in attesa che esso avvenga (busy wait), nel secondo caso eventi asincroni interrompono il processore solo quando l'operazione è pronta ad essere eseguita. Da quanto detto è chiaro che la soluzione di busy wait è, particolarmente, onerosa se si gestisce una periferica lenta che tra un trasferimento e l'altro, o per un trasferimento, impiega molto tempo. La soluzione che fa uso degli interrupt può essere critica se il trasferimento interessa una periferica veloce in quanto la latenza del servizio delle interruzioni può essere di valore comparabile o maggiore dI tempo di reazione della periferica.

Supponendo che un processore operi ad una velocità di 100 M istruzioni/sec e supponendo che esso esegua un'istruzione/ciclo di clock, il suo clock avrà una frequenza di 100 MHz e un tempo di ciclo di 0,01 μsec.

Operazioni a carattere (ad es. processore-modem asincrono oppure processore-stampante a caratteri):

> Ipotesi: periferica che opera con una velocità di trasferimento di 10000 car/sec (ovvero di 1 carattere/10msec) e processore a 100 MHz. In 1 sec il processore esegue circa 100 M istruzioni per cui, se si opera in polling, 10 msec di attesa implicano la mancata esecuzione di 10000 istruzioni.

Il dispositivo Direct Memory Access (DMA) è un dispositivo in grado di operare da master di bus, in grado di gestire cicli bus di trasferimento dati ( in sostituzione del processore). Esso viene utilizzato per aumentare le prestazioni di un sistema di calcolo gestendo, in sostituzione del processore ma in modo più efficiente, le operazioni di trasferimento, consentendo, nel contempo, al processore di operare in parallelo. La maggiore efficienza è dovuta al fatto che mentre un processore per eseguire un trasferimento deve eseguire un'istruzione (ad es. un codice di move) attraverso una fase di fetch ed execute, un DMA gestisce il trasferimento utilizzando Hw specializzato appositamente progettato per realizzare tale unica funzione.



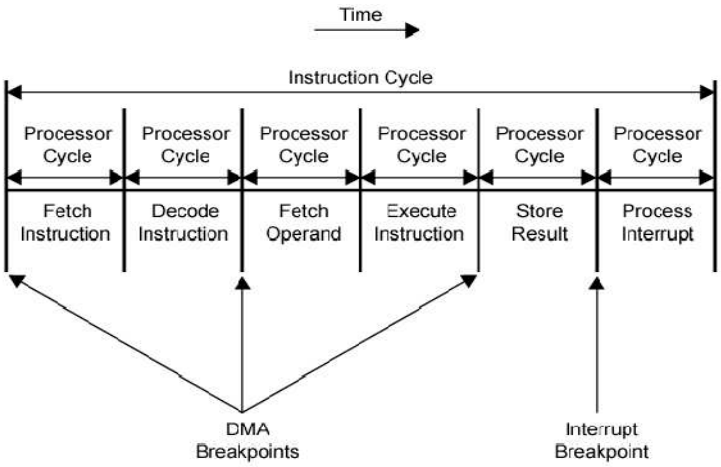

In un'operazione di DMA il processore attiva la linea R/W per indicare la lettura o scrittura, pone sul bus dati l'indirizzo fisico del dispositivo di I/O interessato e, successivamente, l'indirizzo iniziale in memoria del blocco dati coinvolto nell'operazione (da dove leggere o dove scrivere i dati) e la quantità di dati da trasferire, avviando in tal modo l'operazione di trasferimento del blocco dati. Il processore può, a questo punto, eseguire altri codici operativi (sempre che non si creino conflitti di accesso sul bus) mentre il controllore DMA si occupa del trasferimento dei dati colloquiando direttamente con la memoria centrale. Alla fine del trasferimento del blocco il controllore DMA invia un interrupt alla CPU al fine di avviare le operazioni di chiusura dell'operazione. Durante il trasferimento dati il DMA può accedere al canale dati o una parola alla volta, sottraendo di tanto in tanto alla CPU il controllo sul canale (cycle stealing) oppure per blocchi, prendendo possesso del canale per una serie di trasferimenti (burst mode).

La CPU è bloccata in entrambi i casi, ma il burst mode è più efficace perché la latenza di acquisizione del canale è comunque un'operazione onerosa.

Nel Caso di un trasferimento dati in cycle stealing il controllore DMA prende possesso del bus per un ciclo necessario a trasferire una parola (word) di dati. In tale tipo di operazione la CPU non cambia contesto (non è una interruzione), il suo ciclo viene temporaneamente ritardato prima che acceda al bus Ad esempio, prima del caricamento di un dato e/o operando o di una scrittura. Tale sospensione crea, di fatto, un rallentamento del processore ma non così tanto come nel caso in cui sia la CPU stessa ad occuparsi del trasferimento dati.

Breakpoint di DMA e di Interrupt durante un ciclo di istruzione

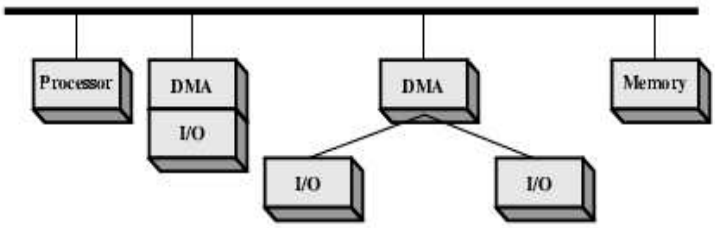
Un ciclo istruzione un processore è realizzato mediante la successione di fasi elaborative eseguite ad ogni ciclo di clock (più cicli di clock (o, in taluni casi più fasi di clock) realizzano un ciclo istruzione. Nella figura seguente si evidenzia il caso di un processore che esegue un ciclo v Neumann in 6 fasi (fetch istruzione, decodifica, fetch operando, esecuzione istruzione, memorizzazione risultato, controlla presenza interuzioni). In essa sono indicati i possibili punti di interruzione del ciclo (breakpoint) per operazioni di DMA e di interruzioni. Come è noto, la sospensione del ciclo richiede il salvataggio del contesto elaborativo e, per mantenere il contenuto informativo associato allo stato limitato e trattabile (anche ai fini delle prestazioni, dovendo il contesto essere salvato), si individuano punti ben precisi del ciclo (tipicamente posti alla fine di una fase operativa) come quelli indicati nell'esempio.




Un sistema può essere architettato in tanti modi differenti a seconda delle risorse hardware disponibili e impiegate. Elemento critico per una architettura a processore è il bus. Si possono definire architetture semplici basate su singolo bus o architetture più complesse basate su più bus. I bus possono essere specializzati per operazioni di I/O verso device lenti/veloci o per trasferimenti verso risorse quali la memoria centrale.

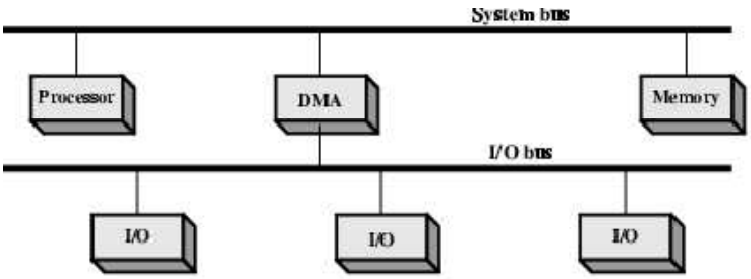
Ad esempio nell'architettura a semplice bus singolo di seguito riportata, la connessione fra DMA e I/O è fatta per tamite il bus: in tal caso il controller DMA legge o scrive dal device di I/O e scrive o legge verso la memoria. Ogni trasferimento usa il bus due volte: da I/O a DMA e poi da DMA alla memoria e la CPU perde il possesso del bus due volte.

Una soluzione alternativa più efficiente, basata sempre sull'uso di un Bus singolo, fa ricorso a un controllore DMA integrato con I/O e in grado di controllare più di un dispositivo. In tal caso ogni trasferimento usa il bus una volta (da DMA a memoria) e la CPU perde il controllo del bus una sola volta.



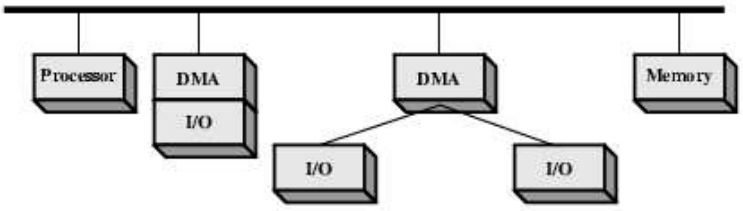

(b) Single-bus, Integrated DMA-I/O

Architetture più efficienti fanno uso di un Bus di I/O separato come nella figura di seguito mostrata. In tale tipo di architettura il DMA necessita di una sola interfaccia di I/O e per essa ogni trasferimento usa il bus di sistema una sola volta (da DMA a memoria), la CPU perde il controllo del bus una sola volta.




(c) I/O bus

Come nel caso del bus singolo, una soluzione più efficiente è data dalla seguente soluzione.




(b) Single-bus, Integrated DMA-I/O

| | |
|---|---|
| Name | I8237DMA |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Indirizzo base del dispositivo |
| Address2 | Indirizzo base+$F (DMA occupa 16 locazioni di memoria) |
| BUS | Ident. del bus a cui il dispositivo è connesso |
| COM1 | Ident. della CPU master per il trasferimento |
| COM2 | \|vettore (b15-b8)\|priorità (b7-b4)\|linea Int (b3-b0)\| |
| COM3 | Ident. Gestore interruzioni |
| COM4 | bit0-3 e bit4-7 ident. dispositivi connessi al canale 0 e 1 |

Le operazioni di trasferimento dati verso i moduli di I/O gestite mediante interruzioni o polling richiedono l'esecuzione di codici operativi da parte del processore per cui la velocità di trasferimento è limitata dalla velocità con cui il processore testa e serve il device e il processore è impegnato nell'eseguire un certo numero di istruzioni per ogni operazione di I/O. Quando grandi volumi di dati devono essere trasferiti, si riccorre ad un meccanismo più efficiente, quello basato sul DMA in grado di operare parallelamente al processore e gestire, mediante hardware specializzato le operazioni di trasferimento. Un dispositivo DMA controller è, infatti, capace, opportunamente programmato, di diventare il master del bus e supervisionare un trasferimento tra memoria e memoria o un'interfaccia periferica o una memoria di massa, senza intervento del processore. Mentre esegue un trasferimento, il DMA pone gli indirizzi di memoria sul bus e spedisce e riceve i segnali necessari per l'effettuazione di operazioni di lettura e scrittura in memoria e sulle interfacce periferiche.

Lo scopo di un DMA controller è, quindi, quello di realizzare una sequenza di trasferimenti mediante l'uso di cicli di bus sottratti al processore.

Come dispositivo commerciale da cui derivare il DMA didattico inserito nell'ambiente ASIM, si è scelto l'Intel 8237, un semplice controllore programmabile per l'accesso diretto in memoria a 4 canali impiegato nella famiglia Intel 808x e oggi disponibile come "Core" (C8237 core by Altera), specificato in VHDL o VERILOG e utilizzabile per lo sviluppo di ASIC e FPGA custom, unitamente a altri componenti MSI e VLSI.

Il DMA controller è organizzato in moduli funzionali così come rappresentato nel seguente diagramma a blocchi.

Il modulo Timing & Control genera la tempificazione interna al chip e dei segnali di controllo verso l'esterno. La tempificazione è derivata dal clock collegato esternamente al chip secondo i seguenti due tipi di ciclo: ciclo di idle (Si) e cicli attivi (S0, S1, S2, S3, S4). Tali cicli caratterizzano i trasferimenti verso dispositivi di I/O. I trasferimenti memoria-memoria occupano due canali alla volta e richiedono prima un trasferimento dalla memoria sorgente verso un registro temporaneo e un successivo trasferimento dal registro temporaneo alla memoria destinazione, utilizzando 8 (4+4) cicli di trasferimento (S11, S12, S13, S14) per la prima metà del trasferimento e (S21, S22, S23, S24) per la seconda metà. Ciascuno stato Si dura un periodo di clock.

Modulo di Fixed Priority & Rotating Priority Logic

In caso di competizione su due o più canali, il controllore DMA può selezionare il canale da attivare in base ad uno schema prioritario fisso, in cui l'ordine prioritario è legato alla posizione del canale (3 più prioritario e 0 meno prioritario) o dinamico a priorità ruotante (round-robin) in cui l'ultimo canale che ha ricevuto il servizio passa in posizione meno prioritaria.

I registri del DMA

Il DMA controller contiene al suo interno una memoria di 344 bit organizzata sotto forma di registri. L'accesso ai registri può avvenire in lettura/scrittura o, per taluni di essi o in sola lettura o scrittura. La selezione dei registri è fatta mediante le quattro linee A3-A0, i segnali di lettura (IORN) e scrittura (IOWN) attivi negati, solo se il chip è abilitato assertando il chip select negato (CSN=0).

Command Register: registro di sola scrittura posto all'indirizzo $8

Il registro CNTR controlla le operazioni del DMA, esso è programmato dal microprocessore ed è resettato dal segnale di Reset o dall'istruzione Master Clear.

Le richieste di trasferimento in arrivo sui vari canali sono gestite secondo logica prioritaria fissa o di tipo round-robin. Il trasferimento dei dati può avvenire in uno dei seguenti 4 modi:

- **Single**: il controllore dopo ogni trasferimento rilascerà il bus al processore per almeno un ciclo di bus, dopo inizierà di nuovo a testare la linea di richiesta e se attiva, procederà a "rubare" un altro ciclo;
- **Block**: posta e soddisfatta la richiesta del bus, sarà effettuato l'intero trasferimento del blocco;
- **Demand**: simile al modo **block,** con la differenza che il trasferimento del blocco continua fin quando la linea di richiesta è attiva ma, quando il trasferimento viene sospeso e poi ripreso, esso inizia dal punto in cui era stato sospeso;
- **cascade**- permette di realizzare, collegando più controllori 8237 in cascata, sistemi DMA con più di 4 canali.

Il blocco di dati che è possibile trasferire in un'unica operazione è di max 64 Kbyte. Alla fine di ogni trasferimento, la linea di fine conteggio segnala al processore (o alla periferica che aveva attivato) la fine del trasferimento).

Il DMA consente, inoltre, l'auto inizializzazione del conteggio associato al trasferimento, le operazioni di trasferimento di tipo memoria-memoria; la richiesta di DMA programmata e l'inibizione di un canale.

Il componente DMA simulato in ASIM differisce da quello commerciale per il minor numero di canali, che sono stati ridotti da 4 a 2 ed i modi di trasferimento che sono limitati a **single** e **block**. In Fig. 4-13 Schema di collegamento del DMA, è mostrato un esempio di interconnessione del DMA con un sistema basato sul processore M68000.

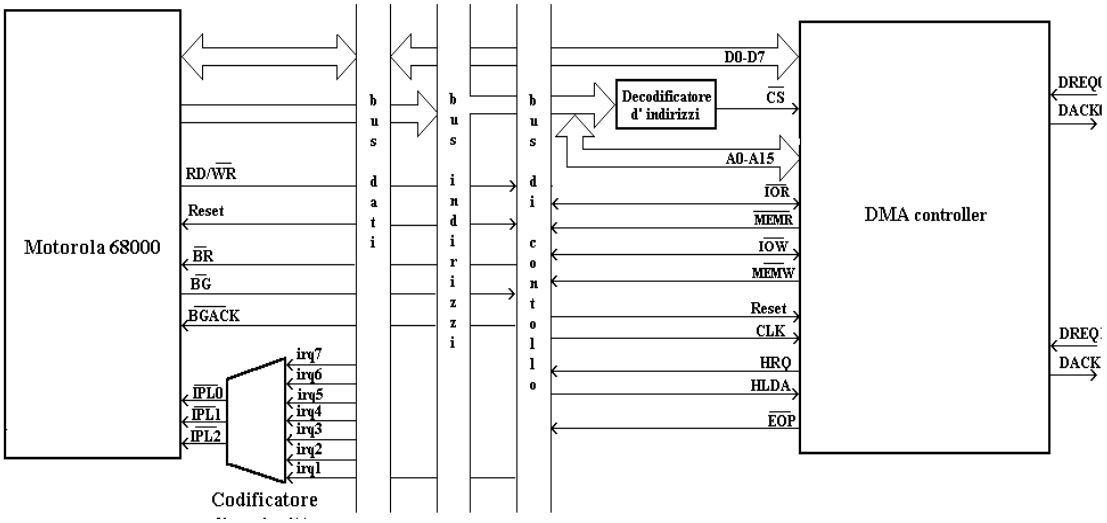



**Fig. 4-13 Schema di collegamento del DMA**

Un DMA controller in una configurazione ASIM si collega d un lato al processore, che avendo accesso ai suoi registri, attraverso il bus di sistema ne permette la programmazione delle operazioni e le periferiche con cui effettuare operazioni di trasferimento da e per la memoria.

## Collegamento al processore

Le linee di collegamento al processore, sono mostrate sulla sinistra dello schema di Fig. 4-13 Schema di collegamento del DMA. In particolare, **D0-D7** sono 8 linee che vanno connesse al bus dati per il trasferimento dei dati da e verso il componente; **CS** è utilizzata per la selezione del dispositivo mentre la selezione, dei registri interni è effettuata mediante le linee **A0-A3**del bus indirizzi e dai segnali di lettura-scrittura sul componente: **IOW** e **IOR**. Il controllore, per eseguire un trasferimento, deve attivare i segnali necessari per l'effettuazione di operazioni di lettura e scrittura in memoria (**MEMR** e **MEMW**) e sulle interfacce periferiche (**IOR** e **IOW**).

Sulle linee **CLK** e **Reset** arrivano rispettivamente il segnale dal generatore di clock e il segnale di reset che esegue il posizionamento al valore 0 di tutti i registri del controllore. La linea **HRQ** (Hold Request) è adoperata per spedire una richiesta di controllo del sistema bus. Essa normalmente è applicata all'ingresso HOLD della CPU. Un segnale, in arrivo dalla CPU, sulla linea **HLDA** (Hold Ack) indica che è stato acquisito il sistema bus. La linea d'interruzione **EOP** trasmette al processore, o a un eventuale gestore delle interruzioni, un interruzione per avvisare che il trasferimento di un blocco di memoria è stato completato.

## Collegamento ad un device

Sulla destra dello schema in Fig. 4-13 Schema di collegamento del DMA, sono mostrate le linee fisiche che collegano il DMA controller alle periferiche. In particolare, **DREQ0** e **DREQ1** sono le linee di richieste, usate dalle periferiche collegate ai rispettivi canali, per ottenere dei cicli DMA. Le richieste che arrivano su **DREQ0** hanno precedenza su quelle che arrivano su **DREQ1**. Le linee **DACK1** e **DACK2** informano la periferica,

connessa a quel canale, che è stata selezionata per un ciclo DMA. Queste linee si comportano, nei confronti dei componenti periferici che richiedono questo servizio, come un "chip select".

## Configurazione ASIM del DMA

In Fig. 4-14 Configurazione del DMA, è mostrato lo schema concettuale di interconnessione del dispositivo DMA industriale in un sistema. Per inserire il componente in una configurazione ASIM, bisogna attenersi alla stessa procedura seguita per gli altri componenti di tipo Device. I parametri presenti nella finestra **Aggiungi Device** permettono di gestire la connessione del controllore con gli altri componenti della configurazione. Il significato dei singoli parametri è il seguente:

**Nome Elemento** è utilizzato per specificare il tipo di componente da inserire, nel nostro caso deve essere I8237DMA.

**Identificatore** deve essere un numero compreso tra 01 ed FFe viene utilizzato dal programma per riferire il dispositivo;

**Indirizzo1** rappresenta l'indirizzo base del dispositivo per la selezione dei suoi registri;

**Indirizzo2**è dato da **Indirizzo1** + $F, essendo lo spazio indirizzabile di 16 byte.

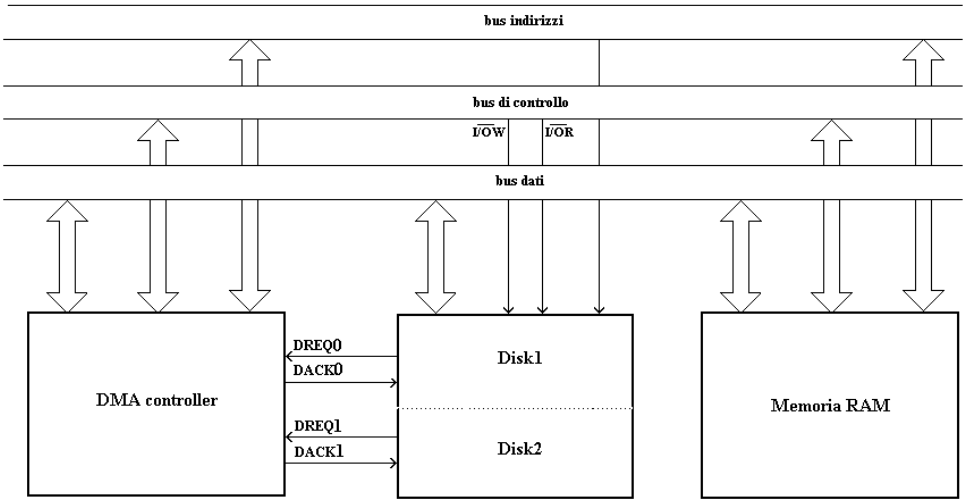



**Fig. 4-14 Configurazione del DMA**

**BUS:** determina l'Identificatore del bus a cui è connesso il dispositivo (per cui, si simula il collegamento a un componente MMU/BUS delle linee D0-D7, A0-A7, IOR, IOW, MEMR, MEMW, CLK e Reset del DMA).

**COM1:** definisce l'Identificatore della CPU alla quale richiedere il controllo del bus mediante un handshaking (si simula il collegamento logico delle linee HRQ e HLDA).

**COM2:** specifica del tipo di interruzione trasmessa al gestore delle interruzioni (CPU o PIC), indicato in COM3, alla fine del conteggio di un trasferimento di un canale. Delle quattro cifre esadecimali che definiscono COM2 la meno significativa individua la linea d'interruzione, la seconda definisce la priorità e le due più significative specificano il vector number da trasmettere al processore che gestisce l'interruzione. Se le interruzioni sono gestite da un PIC, queste due cifre non devono essere specificate. I due parametri COM2 e COM3 determinano, quindi, la connessione logica della linea d'interruzioni EOP al processore o al PIC.

**COM4:** le 2 cifre meno significative e le 2 più significative indicano, rispettivamente, l'identificatore della periferica collegata al canale 0 e al canale 1. Specificando il valore in COM4 si definisce, di fatto, a quali componenti collegare logicamente le coppie di linee (DREQ0, DACK0) e (DREQ1, DACK1).

## Modello di programmazione

La finestra di programmazione associata al dispositivo I8237DMA ed registri programmabili del controllore sono riportati inFig. 4-15 Modello di programmazione del DMA.

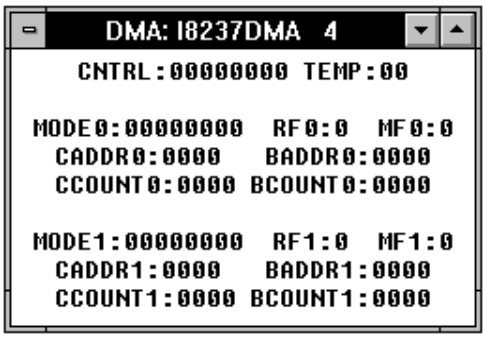



**Fig. 4-15 Modello di programmazione del DMA**

Il dispositivo è costituito da due canali DMA simmetrici programmabili mediante i registri BADDR (indirizzo base del trasferimento), CADDR (indirizzo corrente del trasferimento), BCOUNT (valore base del conteggio espresso come numero di byte da trasferire), CCOUNT (valore corrente del conteggio). A tali registri si aggiunge il registro MODE che codifica la modalità del trasferimento il flag RF che consente di avviare via software il trasferimento (senza la richiesta di una periferica) e MF per mascherare il trasferimento sul canale. Il registro di controllo/stato CNTRL è comune ai due canali, così come il registro TEMP utilizzato per appoggiare il dato da trasferire nei trasferimenti memoria-memoria. Di seguito si riporta il dettaglio di programmazione di detti registri:

- **CADDR0** e **CADDR1** (indirizzo corrente del trasferimento) del DMA: hanno parallelismo 16 bit e contengono l'indirizzo della locazione di memoria interessata al trasferimento. Nel caso del trasferimento da memoria a memoria **CADDR0** contiene l'indirizzo sorgente e **CADDR1** l'indirizzo destinazione. In ogni caso, entrambi i registri sono accessibili sia in lettura che in scrittura ed il loro indirizzo relativo è 0 per **CADDR0** e 2 per **CADDR1**.

- **BADDR0** e **BADDR1** sono i registri indirizzo di base ed hanno la funzione di memorizzare gli indirizzi iniziali rispettivamente di **CADDR0** e **CADDR1**. Essi sono a 16 bit e sono accessibili solo in fase di scrittura, infatti, quando viene scritto un valore in un registro indirizzo corrente, questo viene copiato anche nel relativo registro di base ed il valore rimane immutato fino a quando non si verifica un'altra scrittura.

- **CCOUNT0** e **CCOUNT1** sono i registri di conteggio correnti, anch'essi a 16 bit, memorizzano il numero di byte da trasferire; sono accessibili in lettura e scrittura ed il loro indirizzo relativo è 1 per **CCOUNT0** e 3 per **CCOUNT1**. Nel caso di trasferimento da memoria a memoria il conteggio viene effettuato da **CCOUNT1**.

- **BCOUNT0** e **BCOUNT1** sono registri a 16 bit che memorizzano il conteggio di base ed hanno la funzione di conservare gli indirizzi iniziali rispettivamente di **CCOUNT0** e **CCOUNT1**, essi sono accessibili solo in fase di scrittura. Quando viene scritto un valore in un registro di conteggio corrente questo viene copiato anche nel relativo registro di base ed il valore rimane immutato fino a quando non si verifica un'altra scrittura.

- **MODE0** e **MODE1** sono i registri che contengono le informazioni relative al modo di funzionamento dei rispettivi canali. Essi sono di sola scrittura ed hanno entrambi indirizzo relativo pari a $B; la selezione tra i

due avviene sul valore del bit meno significativo del dato: se 0, il dato viene scritto in **MODE0** altrimenti in **MODE1**. Il significato dei bit dei registri **MODE0** ed **MODE1** è illustrato nella Tabella 4-9-Significato *dei bit nei registri MODE***.**

- **RF0** e **RF1** sono i flag dove indirizzare le richieste di tipo software al DMA, queste producono gli stessi effetti di quelle provenienti dalle interfacce dei dispositivi. Questi flag sono accessibili solo in scrittura e l'indirizzo relativo per entrambi è $9. Anche in questo caso la selezione del canale avviene sul bit meno significativo del dato: 0 per il canale 0 ed 1 per il canale 1. Il valore che deve assumere il flag deve essere posto sul bit numero 3 del dato.

| bit | significato |
|---|---|
| 0 | Se posto a 0 selezione il canale 0 se posto a 1 il canale 1 |
| 1-2 | Bit non utilizzati |
| 3 | indica la **direzione di trasferimento**: 0 per trasferimenti da memoria ad interfaccia, 1 da interfaccia a memoria |
| 4 | il valore 1 abilita l'auto inizializzazione, cioè al termine del conteggio i registri indirizzo e conteggio corrente sono caricati con i valori dei rispettivi registri di base |
| 5 | il valore 1 abilita il decremento di una unità del valore contenuto in CADDR di quel canale dopo ogni trasferimento di un byte; deve essere posto a 0, se vogliamo l'**incremento** |
| 6 | Bit non utilizzato |
| 7 | determina il modo del trasferimento: 0 per il modo **Single** ed 1 per il modo **Block**; nel primo caso il bus viene rilasciato al processore alla fine di ogni trasferimento, viceversa, nel modo **block** il bus viene rilasciato dopo il trasferimento dell'intero blocco |

**Tabella 4-9-Significato dei bit nei registri MODE**

- **MF0** e **MF1** del dispositivo, accessibili in sola scrittura, mascherano le richieste dei rispettivi canali se il mask flag di quel canale è posto ad 1. L'indirizzo relativo di entrambi i flag è $A, il canale selezionato, come prima in base al valore del bit 0, il valore del flag va posto sul bit 2 del dato.

In un trasferimento da memoria a memoria il byte da spostare viene letto dalla locazione sorgente e spostato nel registro temporaneo,**TEMP** e da questo spostato nella locazione di memoria di destinazione. Questo registro può essere solo letto ed ha indirizzo relativo pari a $D.

Il registro **CNTRL** è suddiviso in 2 parti: i 4 bit meno significativi rappresentano i bit di stato del componente, mentre quelli più significativi i bit di controllo. Su questo registro possono essere fatti accessi, all'indirizzo relativo $8, sia in lettura che in scrittura, queste ultime però non influenzano i 4 bit di stato. Il significato dei bit di **CNTRL** é specificato in

| *bit* | *azione svolta se il bit è posto ad 1* |
|---|---|
| 0 | **TC0: termine conteggio per il canale 0** |
| 1 | **TC1: termine conteggio per il canale 1** |
| 2 | **DREQ0: richiesta inoltrata al canale 0** |
| 3 | **DREQ1: richiesta inoltrata al canale 1** |
| 4 | **non utilizzato** |
| 5 | **abilita trasferimento da memoria a memoria** |
| 6 | **in un trasferimento memoria-memoria, impone che l'indirizzo sorgente deve rimanere costante per trasferire un byte in più locazioni di memoria (inizializzazione di un blocco di memoria a valore costante)** |
| 7 | **abilita il DMA controller** |

**Tabella 4-10-Significato dei bit di CNTRL**

| *it* | *azione svolta se il bit è posto ad 1* |
|---|---|
| 0 | **TC0: termine conteggio per il canale 0** |
| 1 | **TC1: termine conteggio per il canale 1** |
| 2 | **DREQ0: richiesta inoltrata al canale 0** |
| 3 | **DREQ1: richiesta inoltrata al canale 1** |
| 4 | **non utilizzato** |
| 5 | **abilita trasferimento da memoria a memoria** |
| 6 | **in un trasferimento memoria-memoria, impone che l'indirizzo sorgente deve rimanere costante per trasferire un byte in più locazioni di memoria (inizializzazione di un blocco di memoria a valore costante)** |
| 7 | **abilita il DMA controller** |

Tabella 4-10-Significato dei bit di CNTRL.

Un quadro riassuntivo sull'indirizzamento dei registri e dei flag del componente è riportato in **Errore. L'origine riferimento non è stata trovata.**tab. 4-11.

| *Indirizzo relativo (Hex)* | *Tipo di accesso* | *Nome del registro o del flag* |
|---|---|---|
| 0 | W/R | **CADDR0** |
| 1 | W/R | **CCOUNT0** |
| 2 | W/R | **CADDR1** |
| 3 | W/R | **CCOUNT1** |
| 8 | W/R | **CNTRL** |
| 9 | W | **RF0/RF1** |
| A | W | **MF0/MF1** |
| B | W | **MODE0/MODE1** |
| D | R | **TEMP** |

**Tabella 4-11 Indirizzamento dei registri e dei flag del DMA controller**

## Descrizione dei comandi

Il dispositivo ha dei comandi attivabili scrivendo all'indirizzo indicato in *Tabella 4-12-Indirizzi dei comandi del DMA controller*. I comandi disponibili sono:

- **RESET:** riporta il componente nello stato iniziale, azzerando tutti i registri; per attivarlo basta accedere in scrittura all'indirizzo relativo $D;

- Clear Mask Flag (**CMF**): cancella tutti i flag MF, si attiva scrivendo all'indirizzo relativo $E;

- Write All Mask Flag (**WAMF**): permette di fissare contemporaneamente tutti i flag MF; la scrittura deve avvenire all'indirizzo relativo $F ed il valore di MF0 deve essere posto nel bit 0 del dato, mentre il valore di MF1 nel bit 1.

| Indirizzo relativo esadecimale | Tipo di accesso | Nome del comando |
|---|---|---|
| $D | W | **RESET** |
| $E | W | **CMF** |
| $F | W | **WAMF** |

**Tabella 4-12-Indirizzi dei comandi del DMA controller**

## Programmazione

Prima che sia effettuata la richiesta al DMA di un trasferimento dati si devono inizializzare i valori nei registri del canale interessato al trasferimento e nel registro di controllo.

Ad esempio, nell'ipotesi che il canale adoperato sia quello '0', le operazioni da eseguire sono le seguenti:

- scrivere all'indirizzo relativo $**0** (registro **CADDR0**) una word indicante l'indirizzo del primo byte, del blocco in memoria, da trasferire;
- scrivere all'indirizzo relativo $**1** (registro **CCOUNT0**) una word indicante il numero di byte che si vuole trasferire;
- scrivere all'indirizzo relativo $**B** esadecimale (registro **MODE0**) un byte che indichi il modo di funzionare del canale (direzione del trasferimento, autoinizializzazione, incremento o decremento di **CADDR0**, modo del trasferimento **Single** o **Block**);
- scrivere all'indirizzo relativo $**8** (registro **CNTRL**) un byte che abiliti il controllore.

Si riportano di seguito due esempi di codice assembler che inizializzano il controllore DMA, per un trasferimento di tipo memoria-dispositivo di I/O e memoria-memoria. Si suppone che A0 sia stato precedentemente caricato con l'indirizzo base del DMA controller. Il codice seguente è relativo a un trasferimento di tipo SINGLE dalla memoria all'interfaccia collegata al canale 0. Il numero di byte da trasferire è $20, gli indirizzi sono quelli crescenti a partire da $1000. Il trasferimento impiega l'autoinizializzazione che permette di ricaricare, al termine del trasferimento, i registri CADDR0 e CCOUNT0 con i rispettivi registri di base, rendendo così il componente pronto ad eseguire un nuovo trasferimento sulle stesse posizioni (ciò, ad esempio, ha senso nei casi in cui si debba fare un refresh ciclico di un'area di memoria come quella video).

```
MOVE.W#$1000,0(A0)      //indirizzo inizio blocco sorgente
MOVE.B#$20,1(A0)        //N.ro byte da trasferire
MOVE.B#$10,$B(A0)       //in MODE: 0x0100x0, modo singolo,inc
                        //conteggio,auto inizializzazione, trasferimento
                        //mem->interfaccia, canale 0
MOVE.B#$80,$8(A0)       //in CTRL abilitazione DMA
```

L'operazione di trasferimento inizia quando la periferica collegata al canale 0 spedisce una richiesta al DMA e quest'ultimo l'accetta. La stessa richiesta, invece che mediante un segnale hardware, potrebbe anche realizzarsi in software mediante l'uso del flag RF0 (o RF1 per l'altro canale) con la seguente istruzione:

```
MOVE.b      #$08,9(A0)          //fa uso del flag RF0 per avviare il DMA
```

Il secondo esempio è relativo al trasferimento memoria-memoria di un blocco di byte. In tal caso le operazioni da eseguire sono:

- scrivere all'indirizzo relativo $**0** (registro **CADDR0**) e $**2** (registro **CADDR1**) rispettivamente gli indirizzi della locazione iniziale del blocco in memoria da trasferire e dell'indirizzo di inizio della destinazione;
- scrivere all'indirizzo relativo $**3** (registro **CCOUNT1**) una word indicante il numero di byte che si vuole trasferire;
- scrivere due volte all'indirizzo relativo $**B** esadecimale (registri **MODE0** e **MODE1**) un byte che indichi il modo di funzionare del canale 0 (sorgente) e del canale 1 (destinazione) per quanto riguarda l'autoinizializzazione e l'incremento o il decremento di **CADDR0** e **CADDR1**);
- scrivere all'indirizzo relativo $**8** (registro **CNTRL**) un byte che abiliti il controllore ed il trasferimento da memoria a memoria.

Di seguito è riportato l'esempio di inizializzazione relativo a tale tipo di trasferimento di un blocco di 64 byte ($40) a partire dall'indirizzo (sorgente) $1000. Esso deve essere trasferito nella zona di memoria successiva all'indirizzo $2000 (destinazione). Infine, è stata disabilitata l'autoinizializzazione sia per il canale sorgente che per quello destinazione.

```
MOVE.W#$1000,$0(A0)          //inizio blocco sorgente
MOVE.W#$2000,$2(A0)          //inizio blocco destinazione
MOVE.B#$40,$3(A0)            //N.ro byte da trasferire in CCOUNT
MOVE.B#$00,$B(A0)            //in MODE0 modalità di trasferimento
MOVE.B#$01,$B(A0)            //in MODE1 modalità di trasferimento
MOVE.B#$A0,$8(A0)            //in CNTRL abilitazione DMA con
                             //operazione mem-to-mem
```

Quando un device collegato ad un canale del DMA invia una richiesta e il flag MF di quel canale è uguale a zero, viene settato il bit corrispondente in CNTRL (il bit 2 per il canale 0 e il bit 3 per il canale 1). Se il bit 7 del registro CNTRL (che abilita il dispositivo) è uguale a uno, il dispositivo spedisce al processore una richiesta di bus che viene automaticamente concessa.

La direzione del trasferimento dipende dal valore del bit 3 del registro MODE associato al canale; se pari a 0, il trasferimento è dalla memoria al device, altrimenti è nella direzione opposta. Se il bit 5 di CNTRL è uguale ad 1, il trasferimento è, invece, effettuato da memoria a memoria.

Preso il possesso del bus, il controllore effettua, per un trasferimento da Device a memoria, una lettura all'indirizzo più basso associato al device e una scrittura del dato letto all'indirizzo di memoria presente nel registro CADDR di quel canale, ovviamente, nel caso di trasferimento da memoria a device, le operazioni sono invertite. In entrambi i casi viene decrementato di una unità il registro CCOUNT del relativo canale mentre il registro CADDR viene incrementato o decrementato di una unità rispettivamente, se il bit 5 del registro MODE è pari a zero o ad uno.

Completato un ciclo di bus, questo viene rilasciato al processore se il valore del bit 7 del registro MODE è zero (trasferimento a singolo byte); se posto a 1, si susseguono, invece, tutti gli altri cicli necessari al trasferimento dell'intero blocco. Al termine del trasferimento di un blocco, cioè quando il valore **CCOUNT** diventa nullo, sono eseguite le seguenti operazioni:

- viene cancellato il flag **RF** (è il flag dove un processore può indirizzare le sue richieste ed hanno gli stessi effetti di quelle che arrivano dai device),
- è azzerato il bit 2 o 3 di **CNTRL** (bit delle richieste) a seconda che si sia utilizzato rispettivamente il canale zero o uno,
- è posto ad uno il bit 0 o 1 di **CNTRL** (bit di fine conteggio) rispettivamente per il canale zero e uno,
- infine viene inviata un'interruzione del tipo specificato in **COM2** al gestore delle interruzioni specificato in **COM3**.

Per un trasferimento da memoria a memoria viene effettuata, prima una operazione di lettura all'indirizzo contenuto in **CADDR0** ed il dato viene posto nel registro **TEMP**, poi questo valore viene scritto all'indirizzo di memoria contenuto in **CADDR1**.

Dopo un trasferimento il registro **CCOUNT1** viene decrementato di una unità mentre **CADDR0** e **CADDR1** vengono incrementati o decrementati di una unità, se il bit 5 dei rispettivi registri **MODE** è pari a zero o ad uno. Se però il bit 6 di **CNTRL** è posto ad 1 allora **CADDR0** rimane costante durante il trasferimento.

Il bus viene rilasciato al processore dopo il trasferimento dell'intero blocco, cioè quando il valore in **CCOUNT1** diventa nullo. Alla fine del trasferimento viene azzerato il bit 5 di **CNTRL**, il quale attiva i trasferimenti da memoria a memoria.

Qualunque sia il tipo di trasferimento, se il bit 4 di un registro **MODE** è fissato ad 1, al termine del trasferimento di un blocco i registri **CADDR** e **CCOUNT** di quel canale sono caricati con i valori rispettivamente di **BADDR** e **BCOUNT**. Essi,alla scrittura di **CADDR** e **CCOUNT,** assumono il loro stesso valore.

Appendice

<vedere data sheet allegato del dispositivo 82C237 commerciale>

## 4.8 Il dispositivo Periferal Interface Adapter (PIA)

I campi relativi alla configurazione del dispositivo sono così codificati:

| | |
|---|---|
| Name | MM6821PIA |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Indirizzo base |
| Address2 | Indirizzo base+3 |
| BUS | Identificatore di bus esterno a cui rendere visibile le memorie di bus/mem |
| COM1 | Identificatore dispositivo gestore delle interruzioni generate dall'oggetto |
| COM2 | Controllo linea Interruzione IRQA: XXYZ (XX=#vettore, Y=livello di priorita', Z=linea di interruzione |
| COM3 | Controllo linea Interruzione IRQB: XXYZ (XX=#vettore, Y livello di priorità, Z=linea di interruzione |
| COM4 | I due caratteri meno significativi contengono l'identificatore assoluto dell'oggetto a cui il porto parallelo e'connesso, i due caratteri più significativi la tipologia di connessione. |

**Tabella 4-13 Configurazione del dispositivo PIA**

Il dispositivo PIA (Periferal Interface Adapter) è un dispositivo parallelo, a parallelismo 8 bit, facente parte della famiglia dei processori Motorola 68xx. Il dispositivo contiene due sezioni quasi identiche, ciascuna dotata di 8 bit dati configurabili, anche singolarmente, come linee di ingresso o di uscita. Ciascun porto può, quindi, funzionare in ingresso, uscita o in configurazione mista (in-out). L'utilizzo tipico previsto è quello di controllore di una periferica con comunicazione full duplex e con possibilità di controllare le operazioni di I/O mediante interruzioni e schemi di handshaking preconfigurati.

Due coppie di linee di controllo sono utilizzate per gestire l'handshaking per la sincronizzazione della periferica connessa al dispositivo. Di queste, CA1 e CB1, in ingresso al dispositivo, acquisiscono eventi dalle periferiche e controllano la linea di interruzione IRQA e IRQB verso il processore; le altre due linee, CA2 e CB2, possono essere programmate per operare come segnali in ingresso (in tal caso si comportano in modo analogo alla CA1 e CB1) sia in uscita per implementare forme di handshaking. Il controllo delle operazioni realizzabili con le due linee è programmato, come di seguito mostrato, mediante i campi del registro di controllo.

Per aggiungere un M6821PIA ad una configurazione si deve selezionare dal menù Configura il comando Aggiungi Device e specificare tutti i parametri previsti nella finestra di dialogo.

Il "Nome Elemento" è M6821PIA. Indirizzo 1 deve essere pari, Indirizzo 2 deve essere uguale a "Indirizzo 1" + 3. "BUS" deve contenere l'Identificatore del bus cui il dispositivo è connesso. "Com1" contiene, se previsto, l'Identificatore del dispositivo gestore delle interruzioni. "Com2" e "Com3" definiscono le linee di interruzione IRQA e IRQB previste dal dispositivo (associate ai flag di interruzione, rispettivamente, dei registri CRA e CRB); i valori attribuibili a tali parametri sono quelli già più volte specificati; si osservi che, se "Com1" è diverso da zero, le linee di interruzione devono essere specificate e non possono essere nulle.

Infine, "Com4" contiene l'identificatore del dispositivo periferico connesso (nell'attuale versione di ASIM, l'unico dispositivo a cui un M6821PIA può essere connesso è un altro M6821PIA). L'identificatore del dispositivo da connettere è posto nelle due cifre meno significative di "Com4". Le altre due cifre servono per definire le connessioni di CA2 e CB2 alle rispettive linee di hadshake CA1, CB1,CA2 e CB2.In particolare, la cifra meno significativa può assumere uno dei seguenti valori:

- 0 linea CA2 connessa alla linea CA1 del dispositivo periferico;

- 1 linea CA2 connessa alla linea CA2 del dispositivo periferico;
- 2 linea CA2 connessa alla linea CB1 del dispositivo periferico;
- 3 linea CA2 connessa alla linea CB2 del dispositivo periferico;

nei primi due casi le linee dati A sono connesse alle linee dati A del dispositivo periferico; nei rimanenti due, le linee dati A sono connesse alle linee dati B del dispositivo periferico.

La più significativa può assumere uno dei seguenti valori:

- 0 linea CB2 connessa alla linea CA1 del dispositivo periferico;
- 1 linea CB2 connessa alla linea CA2 del dispositivo periferico;
- 2 linea CB2 connessa alla linea CB1 del dispositivo periferico;
- 3 linea CB2 connessa alla linea CB2 del dispositivo periferico;

nei primi due casi le linee dati B sono connesse alle linee dati A del dispositivo periferico; nei rimanenti due, le linee dati B sono connesse alle linee dati B del dispositivo periferico.

E' a cura di chi definisce la configurazione fare in modo che le connessioni dei due dispositivi siano coerenti. Inoltre, poiché all'accensione della macchina da simulare tutti i registri sono nulli, devono, opportunamente, essere inizializzati i dispositivi prima di effettuare un trasferimento dati.

### 4.8.1 Programmazione del dispositivo PIA

Il dispositivo PIA simulato in ASIM è derivato da quello commerciale MC6821 di cui ne conserva quasi tutte le caratteristiche funzionali e architetturali. In particolare, la struttura dei sei registri interni (3 per la sezione A e 3 per quella B) ad otto bit è la seguente::

- due registri per il trasferimento dei dati da e verso la periferica (PRA e PRB);
- due registri di controllo/stato (CRA e CRB);
- due registri, DRA e DRB, per il controllo della direzione dei dati (in input o in output).

Poiché il dispositivo impiega per la selezione dei registri interni solo quattro indirizzi, per accedere a tali registri viene utilizzato il meccanismo di indirizzamento indiretto per tramite di campi del registro controllo che vanno opportunamente settati prima dell'accesso:

| AD1 | AD0 | CRA2 | CRB2 | Registro Selezionato |
|---|---|---|---|---|
| 0 | 0 | 1 | X | PRA |
| 0 | 0 | 0 | X | DRA |
| 0 | 1 | X | X | CRA |
| 1 | 0 | X | 1 | PRB |
| 1 | 0 | X | 0 | DRB |
| 1 | 1 | X | X | CRB |

X : indifferente.

**Tabella 4-14-Indirizzamento dei registri interni**

Le due sezioni del dispositivo PIA sono simili e singolarmente programmabili. Il significato dei campi di controllo presenti nei rispettivi registri è di seguito riportato. Eventuali differenze nel comportamento delle due sezioni sono esplicitamente indicate.

Controllo delle linee di ingresso CA1 e CB1

CA1 consente di controllare il flag di interruzione (b7 del registro di stato) IRQA1. In particolare, tale bit va alto a seguito della transizione di fronte attiva di CA1 (la transizione attiva è programmabile mediante il bit b1, in particolare, è la variazione 1-0 se b1=0, 0-1 se b1=1). Lo stato del flag IRQA1 del registro di stato determina quello della linea fisica IRQA usata tipicamente per interrompere un processore con la transizione 1-0. Tale interruzione è mascherata se il bit b0 è posto a 0, abilitata se b0=1. Il flag b7 è automaticamente resettato all'atto del ciclo di lettura del registro dato da parte del processore.Quanto detto per CA1 vale anche per la linea CB1 relativa alla sezione B.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **CRA** | IRQA1 | IRQA2 | Controllo CA2 | | | Selezione DRA | Controllo CA1 | |
| **CRB** | IRQB1 | IRQB2 | Controllo CB2 | | | Selezione DRB | Controllo CB! | |

**Tabella 4-15-Formato dei registri controllo CRA e CRB**

Controllo delle linee CA2 e CB2

il controllo di CA2 è differente a seconda che tale linea sia stata programmata per operare come linea di ingresso o di uscita. In particolare, si hanno i seguenti due casi:

CA2 (o CB2) come linea di ingresso (b5=0): si comporta come CA1 con la differenza che il flag di interruzione interessato è IRQA2 ed i bit di programmazione sono b3 nelle funzioni di b0 e b4 nelle funzioni di b1. Quanto detto vale anche per CB2.

CA2 o (CB2) come linea di uscita (b5=1): in tal caso CA2 consente di controllare lo handshaking versola periferica. Sono previsti 3 possibili modi di sincronizzazione codificati mediante i bit b4 e b3:

La tabella 6.2**Errore. L'origine riferimento non è stata trovata.** mostra la codifica adottata per la selezione dei registri del dispositivo. AD1 e AD0 vanno collegati aidue bit meno significativi del bus indirizzi. Essendo PRA e DRA mappati sullo stesso indirizzo la selezione di ciascuno di essi è effettuata in modo indiretto per tarmite del bit 2 di CRA. così come indicato in tabella.

**Tabella 4-16-Controllo della linea CA1(CB1)**

| CRA1 (CRB1) | CRA0 (CRB0) | Flag Interruzione CRA7 (CRB7) | Richiesta Interr IRQA (IRQB) |
|---|---|---|---|
| 0 | 0 | Alto su high/low di CA1 (CB1) | Disabilitata |
| 0 | 1 | Alto su high/low di CA1 (CB1) | Inviata quando CRA7 (CRB7) diventa alto |
| 1 | 0 | Alto su low/high di CA1 (CB1) | Disabilitata |
| 1 | 1 | Alto su low/high di CA1 (CB1) | Inviata quando CRA7 (CRB7) diventa alto |

Il flag di interruzione CRA7 (CRB7) torna al valore basso in seguito ad un' operazione di lettura su PRA (PRB).

**Tabella 4-16-Controllo della linea CA1(CB1)**

I registri PRA, DRA, CRA consentono il controllo completo di un insieme di otto linee dato da connettere verso una periferica (linee A0..A7). Ciascuna linea può essere programmata separatamente come linea di uscita o di ingresso settando, rispettivamente ad 1 o a 0, il corrispondente bit del registro DRA. Il registro PRA rispecchia lo stato delle linee del bus dati del dispositivo. Nelle operazioni di scrittura di PRA, il valore dei singoli bit di PRA viene trasferito nelle corrispondenti linee dati programmate come linee in output; nelle operazioni di lettura, i singoli bit di PRA assumono i valori presenti sulle corrispettive linee programmate come input.

| CRA5 (CRB5) | CRA4 (CRB4) | CRA3 (CRB3) | Flag Interruzione CRA6 (CRB6) | Richiesta Interr IRQA (IRQB) |
|---|---|---|---|---|
| 0 | 0 | 0 | Alto su high/low di CA2 (CB2) | Disabilitata |
| 0 | 0 | 1 | Alto su high/low di CA2 (CB2) | Inviata quando CRA6 (CRB6) diventa alto |
| 0 | 1 | 0 | Alto su low/high di CA2 (CB2) | Disabilitata |
| 0 | 1 | 1 | Alto su low/high di CA2 (CB2) | Inviata quando CRA6 (CRB6) diventa alto |

Il flag di interruzione CRA6 (CRB6) torna al valore basso in seguito ad un' operazione di lettura su PRA (PRB).

**Tabella 4-17-Effetti delle variazioni sulla linea CA2 (CB2 ) in ingresso**

Per il gruppo di registri PRB, DRB e CRB vale quanto detto per il gruppo precedente; le differenze sono nelle funzioni attribuite alle linee di sincronizzazione come mostrato nelle tabelle.

**Controllo della linea di uscita CB2**

| CRB5 | CRB4 | CRB3 | CB2 BASSO | CB2 ALTO |
|---|---|---|---|---|
| 1 | 0 | 0 | Basso in seguito ad un' operazione di scrittura su PRB | Alto quando CRB7 va ad 1 per una variazione h/l o l/h di CB1 |
| 1 | 0 | 1 | Basso in seguito ad un' operazione di scrittura su PRB | Alto al primo colpo di clock successivo la scrittura su PRB |
| 1 | 1 | 0 | Basso quando CRB3 diviene basso in seguito ad un' operazione di scrittura su CRB | Sempre basso finche CRB3 è basso. Diviene alto se CRB3 va ad 1 per un' op. di scritt. su CRB |
| 1 | 1 | 1 | Sempre alto finchè CRB3 è alto Diviene basso se CRB3 va a 0 per un' op. di scrittura su CRB | Alto quando CRB3 diviene alto in seguito ad un' operazione di scrittura su CRB |

**Tabella 4-18 – Controllo della linea di uscita CB2**

La finestra associata a M6821PIA, fig.6.ne mostra tutti i registri. I valori di tali registri possono essere modificati, oltre che da programma, utilizzando il comando Modifica Valore del menù Device. Con i comandi Input da File e Output su File del menù Device è possibile, inoltre, redirigere i flussi di input e output verso un file anziché verso una periferica.

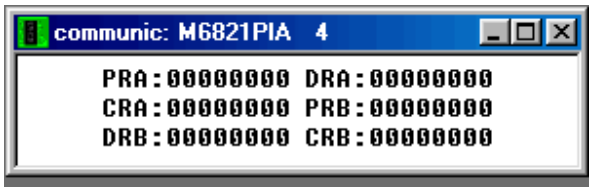



**Fig. 4-16 Finestra dei registri interni della PIA**

## 4.9 Il dispositivo video-tastiera TERMINAL

I campi relativi alla configurazione del dispositivo sono così codificati:

| | |
|---|---|
| Name | TERMINAL |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Indirizzo registro dati (tastiera in scrittura, video in lettura) |
| Address2 | Indirizzo registro stato-controllo |
| BUS | Identificatore di bus esterno a cui è connesso l'oggetto |
| COM1 | Identificatore dispositivo gestore delle interruzioni generate dall'oggetto |
| COM2 | Controllo Interruzione dovuta ad ENTER: XXYZ (XX=#vettore, Y=livello di priorità , Z=linea di interruzione |
| COM3 | Controllo Interruzione dovuta a Buffer Full |
| COM4 | n.s. va posto a 0 |

**Tabella 4-19**

ASIM mette a disposizione, come dispositivo base per realizzare l'I/O da e per un sistema, un dispositivo da utilizzare, appunto, come terminale video- tastiera denominato TERMINAL.

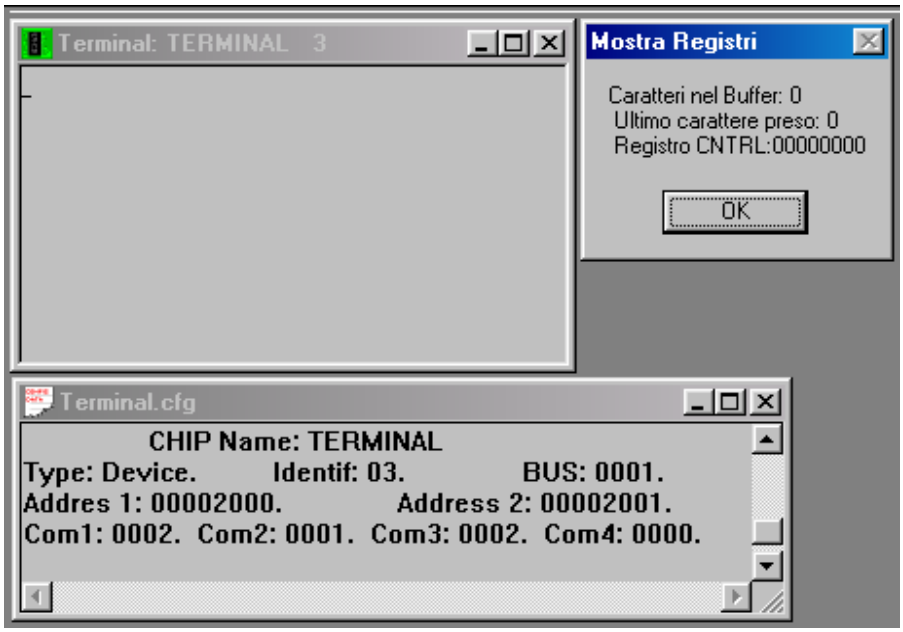



**Fig. 4-17-Finestre ASIM associate al TERMINAL**

Nella finestra associata a questo dispositivo (Fig. 4-17) appare l'output destinato al terminale compreso l'eco dell'input da tastiera. Il video simulato è alfanumerico con dieci righe di quaranta caratteri; la tastiera coincide con quella del PC IBM (eslusi i tasti speciali: Ctrl, Alt, Esc, tasti funzione e movimento cursore) e prevede i tasti di "backspace" ed "enter".

Quando la finestra di TERMINAL diviene quella corrente, la tastiera del PC (esclusi i tasti speciali che svolgono le solite funzioni previste da Windows) emula quella del dispositivo nel senso che un eventuale input da tastiera appare nella finestra e viene inviato al sistema che si sta simulando. Più precisamente, la tastiera accetta dei caratteri solo se è abilitata e l'input va a video solo se la funzione di eco è attiva. Dopo l'accensione

della macchina che si sta simulando sia la tastiera sia l'eco sono disabilitati; per attivarli occorre modificare il valore del registro di controllo e stato: CNTRL. Questo è un registro di un sol byte i cui bit, a partire dal meno significativo, svolgono, se posti al valore 1, le seguenti funzioni:

Bit di Controllo

- bit0 abilita interruzione su "Buffer full";
- bit1 abilita interruzione sull'"ENTER";
- bit2 cancella video;
- bit3 pulisci buffer di tastiera;
- bit4 abilita tastiera;
- bit5 abilita eco;

Bit di Stato

- bit6 stato di "Buffer full";
- bit7 stato di ENTER inviato.

Il contenuto del registro CNTRL può essere modificato o da programma o utilizzando il comando *Modifica Valore* del menù *Device.* Il comando *Mostra Registri* apre una finestra di dialogo in cui appare il registro CNTRL, il numero di caratteri presenti nel buffer di tastiera e la posizione nel buffer dell'ultimo carattere letto dal processore (Fig. 4-17).

Per comprendere il significato di questi due numeri è necessario conoscere il meccanismo di gestione dell'input e output da parte del dispositivo in esame.

TERMINAL ha due indirizzi d'accesso: il primo è associato al buffer di tastiera e al video, il secondo al registro CNTRL. Un'operazione di accesso in scrittura sul primo indirizzo comporta la stampa a video del carattere scritto; i caratteri validi sono quelli con codice ASCII compreso tra 32 e 126 (estremi inclusi) ed il carattere con codice 13 che provoca il ritorno a capo. Quando una riga è piena, il successivo carattere viene scritto nella riga seguente; se tutte le dieci righe sono piene il testo scorre di una riga verso l'alto. Ponendo ad 1 il bit 2 di CNTRL, si pulisce lo schermo e si riporta il cursore in alto a sinistra.

Un'operazione di accesso in lettura sul primo indirizzo consente di prelevare un carattere dal buffer di tastiera. Partendo dallo stato di buffer di tastiera vuoto e con tastiera ed eco abilitati, si digiti una frase qualsiasi. Per ogni carattere premuto, il corrispondente carattere va nel buffer di TERMINAL ed appare nella finestra; inoltre il numero di caratteri nel buffer si incrementa di un'unità. Se vengono battuti più di 256 caratteri la tastiera viene disabilitata, il bit 6 di CNTRL va ad 1 e, se il bit 0 è alto, viene inviata un'interruzione per segnalare al processore la condizione di "Buffer full". Quando si è completata la frase (con meno di 256 caratteri) va premuto il tasto di ENTER; ciò porta TERMINAL a disabilitare la tastiera, ad alzare il bit 7 di CNTRL ed ad inviare al processore un interruzione se il bit 1 di CNTRL è alto; ogni qualvolta il processore legge un carattere dal buffer, il numero di caratteri letti, inizialmente nullo, si incrementa di uno; tale numero rappresenta un puntatore al primo carattere non letto e, il fatto che si incrementi man mano, fa si che il processore legga, nel corretto ordine, tutti i caratteri nel buffer. L'ultimo carattere è sempre l'ENTER; per riattivare la tastiera e pulire il buffer occorre portare al valore 1 i bit 3 e 4 di CNTRL.

Ponendo a 0 il bit 5 di CNTRL si disabilita l'eco; ciò è utile quando si vuole nascondere l'input di tastiera come nel caso in cui venga digitata una password.

Per aggiungere un terminale ad una macchina da simulare va selezionato il comando *Aggiungi Device* del menù *Configura* e vanno specificati i parametri nell'apposita finestra di dialogo (figura 3). Il "Nome Elemento" è TERMINAL; l'"Indirizzo1" DEVE essere pari ed è quello associato al video e buffer di tastiera; l'"Indirizzo2" DEVE essere quello successivo ad "Indirizzo 1" ed è quello associato a CNTRL.

In "BUS" va posto l'Identificatore del bus cui è connesso il dispositivo. In "Com1" va posto, se previsto, l'Identificatore del gestore delle interruzioni. "Com2" e "Com3" vengono utilizzati per specificare le linee di interruzione, rispettivamente, per ENTER e "Buffer full"; delle quattro cifre esadecimali che definiscono questi parametri, la meno significativa individua la linea di interruzione, la seconda definisce la priorità e le due più significative specificano il "vector number". Se "Com1" è diverso da zero, la linea (cifra meno significativa di "Com2" e "Com3") DEVE essere diversa da zero; le altre ciftre possono essere nulle. Infine "Com4" DEVE essere lasciato al valore zero.

## 4.10 Il dispositivo USART

I campi relativi alla configurazione del dispositivo sono così codificati:

| | |
|---|---|
| Name | I8251USART |
| Type | Device |
| Identif | Intero ($01..$FF) che identifica univocamente l'oggetto in una configurazione |
| Address1 | Indirizzo base device (deve essere pari) |
| Address2 | Indirizzo base+1 |
| BUS | Identificatore di bus esterno a cui è connesso il device |
| COM1 | Identificatore del dispositivo gestore delle interruzioni generate dall'oggetto |
| COM2 | Controllo linea Interruzione Rx |
| COM3 | Controllo linea Interruzione Tx |
| COM4 | xxyz (xx=id device connesso; y=0 connessione completa; y=1 connessione semplice;z=n.c.) |

**Tabella 4-20**

Il dispositivo USART (Universal Synchronous-asynchronous Receiver Transmitter) consente di realizzare una connessione seriale fra dispositivi (ad es. il collegamento seriale di un sistema verso un dispositivo periferico quale un terminale, un modem, una stampante, etc., per effettuare un trasferimento di dati) secondo un protocollo sincrono o asincrono.

Per inserire un'USART in una configurazione ASIM i parametri precedenti vanno così configurati:

**Name**: va posto ad 8251USART.

**Identificatore** deve essere un *numero compreso tra 01 ed FF*e viene utilizzato dal programma per riferirsi a questo dispositivo.

**Indirizzo1** indirizzo base del dispositivo (deve essere pari).

**Indirizzo2** è dato da Indirizzo1+1 occupando il dispositivo 2 locazioni di memoria

Questi ultimi due parametri permettono di definire per quali indirizzi il decodificatore d'indirizzi attiva la linea **CS**. Infatti, se l'indirizzo sul bus indirizzo è pari ad uno dei due, il componente viene selezionato, cioè l'operazione di lettura o scrittura è eseguita su un suo registro.

**BUS** determina l'*Identificatore del bus* a cui è connesso l'interfaccia seriale. Quindi, scegliendo per **COM1** l'identificatore di un componente MMU/BUS si connettono le linee **bus dati**, **A0**, **RD**,**WR** e **Reset**, dell'interfaccia seriale, al bus suddetto.

**COM1** definisce l'*Identificatore del gestore delle interruzioni,* cioè il componente (di tipo CPU o I8259PIC) a cui trasmettere le interruzioni.

**COM2** viene utilizzato per specificare l'interruzione trasmessa al gestore delle interruzioni, quando viene ricevuto un carattere in **Receiver shift register** ed è copiato in **Data-in buffer register**,peressere letto dal processore. Delle quattro cifre esadecimali che definiscono **COM2** la meno significativa individua la *linea d'interruzione*, la seconda definisce la *priorità* e le due più significative specificano il "*vector number*" da trasmettere al processore che gestisce l'interruzione. Se le interruzioni sono gestite da un PIC, queste due cifre non devono essere specificate.

**COM3** specifica l'interruzione inviata al gestore delle interruzioni, quando viene copiato il carattere dal **Data-out buffer register** in **Transmitter shift register** ed inizia la sua trasmissione. Le quattro cifre esadecimali hanno lo stesso significato di quelle di **COM2**.

E' chiaro, quindi, che nel nostro simulatore, l'insieme dei parametri **COM1**, **COM2** e **COM3**, permettono di gestire la connessione delle linee d'interruzioni, **RxRDY** e **TxRDY**, con il processore o il PIC

**COM4** viene impiegato per specificare il device a cui l'interfaccia seriale è connessa ed il tipo di connessione. Precisamente, le 2 cifre meno significative specificano l'Identificatore del Device a cui è connesso l'interfaccia seriale. La terza cifra può assumere il valore '0'per realizzare un collegamento pieno o '1'per uno parziale. **COM4** consente di simulare la connessione delle **linee di handshaking**, di **Tx** e **Rx**, con un componente connettibile all'interfaccia seriale.

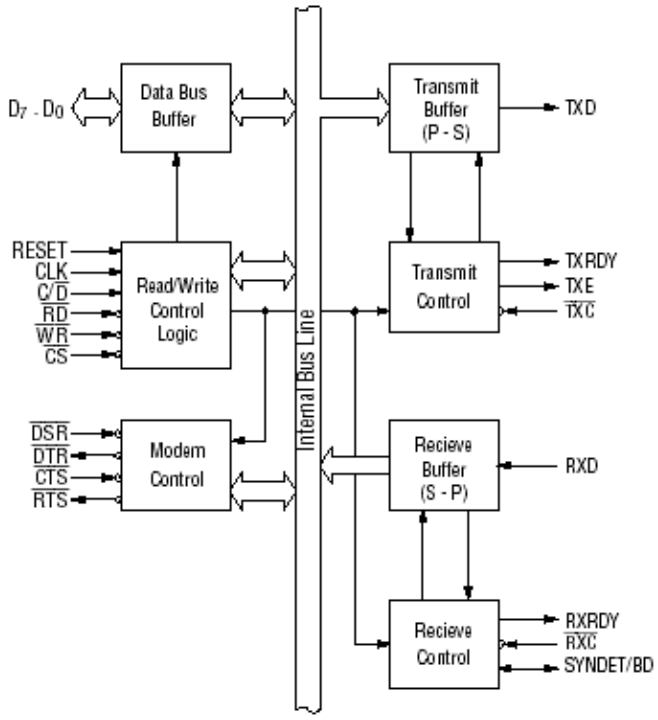
### 4.10.1 Protocollo di comunicazione asincrono

Il protocollo di comunicazione asincrono consente di effettuare una trasmissione a caratteri in modalità start-stop. I due interlocutori tempificano localmente il trasferimento, sincronizzando i relativi clock per ogni carattere trasmesso o ricevuto.

### 4.10.2 Protocollo di comunicazione sincrono

### 4.10.3 Generalità USART

Il componente commerciale di riferimento per simulare l'USART in ASIM è l'Intel 8251A. Il modello funzionale a blocchi dell'USART è riportato in fig.26a, quello di programmazione con le linee di I/O dell'UART simulato sono riportati in fig.26.



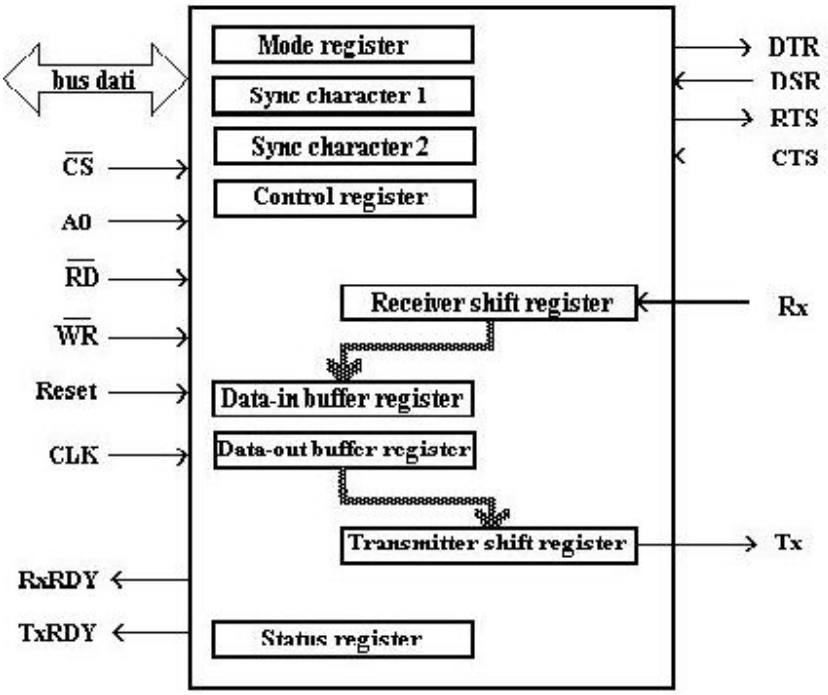



**Fig. 4-18 - Diagramma funzionale dell'USART I8251**

**Fig.26b - Modello del dispositivo USART simulato in ASIM**

Questo componente simula gli stessi registri e relative posizioni dei bit del componente commerciale. Per i dettagli tecnici è, pertanto, possibile riferire il data sheet dell'8251 riportati nell'apposito allegato.

L'USART è un dispositivo full duplex che accetta i caratteri dalla CPU in formato parallelo e poi li converte in un flusso continuo di dati seriali per trasmetterli in modo sincrono o asincrono. Simultaneamente, esso può ricevere un flusso di dati seriali e convertirli nel formato parallelo per essere letti dalla CPU.

Il componente segnala alla CPU quando esso è pronto ad accettare un nuovo carattere da trasmettere e quando ha ricevuto un carattere che può essere letto dalla CPU stessa; quest'ultima può leggere lo stato completo dell'interfaccia in qualsiasi momento.

Altre caratteristiche importanti sono:

- comunicazione di tipo sincrona o asincrona,
- trasmettitore e ricevitore full duplex con doppia bufferizzazione,
- caratteri da 5 ad 8 bit,

- inserzione automatica dei caratteri di sincronismo,
- rilevazione di errori di parità, di overrun e di framing.

### 4.10.4 Interfacci verso il BUS

Le linee fisiche, che collegano, attraverso il sistema bus, l'interfaccia seriale al processore, sono mostrate sulla sinistra dello schema, in Fig..

Il processore scambia i dati con il componente attraverso il **bus dati** e seleziona lo stesso mettendo sul bus indirizzi uno dei due indirizzi ad esso associati. Poi, un decodificatore di indirizzi, collegato al bus indirizzi, quando rivela uno degli indirizzi associati all'8259A, seleziona il componente attraverso la linea **CS**.

I registri interni sono selezionati dal bit meno significativo del bus indirizzo: **A0**; dai segnale di lettura-scrittura: **RD** e **WR**; oltre che dallo stato interno.

Un segnale sulla linea **Reset** riporta il componente nello stato iniziale cancellando tutti i suoi registri. Questo segnale può essere spedito sia dal processore che da un dispositivo esterno.

**RxRDY** e **TxRDY** sono due linee d'interruzione che trasmettono al processore o all'eventuale PICuna richiesta d'interruzione.

L'interruzione su **RxRDY** è inviata quando viene ricevuto un carattere in **Receiver shift register** ed è copiato in **Data-in buffer register**.

Su **TxRDY**, invece, l'interruzione viene inviata quando viene copiato il carattere dal **Data-out buffer** register in **Transmitter shift register** ed inizia la trasmissione.

Un esempio di collegamento dell'interfaccia seriale con un processore è mostrato in Fig. 4-19, dove sono presentate, del Motorola 68000, solo le linee coinvolte nel collegamento.

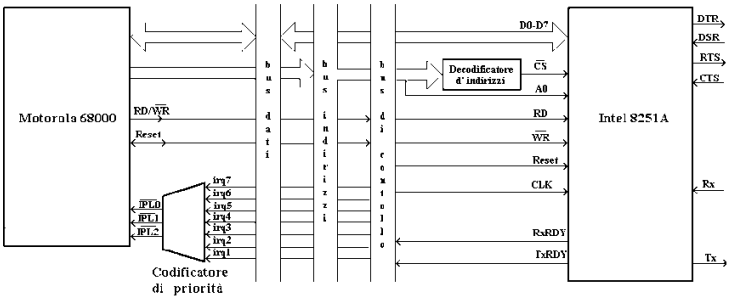


**Fig. 4-19 Collegamento 8251A-MC68000**

### 4.10.5 Interfaccia verso altri dispositivi

In generale due dispositivi sono collegabili quando hanno i segnali d'ingresso e uscita compatibili tra loro. Ciò significa che le uscite di un dispositivo devono essere dello stesso tipo delle entrate dell'altro e viceversa, inoltre, deve essere possibile rispettare i protocolli di handshaking di entrambi i dispositivi.

Sulla destra dello schema in figura 1 sono mostrate le linee fisiche che collegano l'interfaccia seriale ad un device. Le cinque linee poste più in alto vengono utilizzate per lo sviluppo del protocollo di handshaking ed il loro significato è fornito in **Errore. L'origine riferimento non è stata trovata.**.

Le altre due linee, **TX** e **RX**, sono utilizzate rispettivamente per trasmettere e ricevere i dati.

Le proprietà che deve possedere il componente che interfaccia la periferica seriale sono:

1) deve essere provvisto di linee compatibili alle seguenti 6 linee: **DTR**, **DSR**, **RTS**, **CTS**, **TX** ed **RX**, per una connessione piena con l'interfaccia seriale , altrimenti, se vogliamo realizzare una connessione parziale, deve disporre solo di linee compatibili a **TX** ed **RX**,

2) deve rispettare il protocollo di handshaking,

3) nel caso di comunicazione asincrona la trasmissione deve avere il formato presentato in **Errore. L'origine riferimento non è stata trovata.**.

**Tabella 4-21-Segnali per lo sviluppo del protocollo di handshaking**

| segnale | significato |
|---|---|
| **DTR** | l'interfaccia chiede al modem di connettersi alla linea |
| **DSR** | il modem segnala all'interfaccia che si è connesso alla linea |
| **RTS** | l'interfaccia chiede al modem di trasmettere |
| **CTS** | il modem invia in linea la portante e segnala all'interfaccia che è pronto a trasmettere |

**Tabella 4-21-Segnali per lo sviluppo del protocollo di handshaking**

In Fig. 4-21 sono mostrate le evoluzioni dei segnali di handshaking dell'interfaccia che trasmette i dati nel caso in cui dopo la trasmissione del messaggio non venga sconnesso il modem dalla linea.

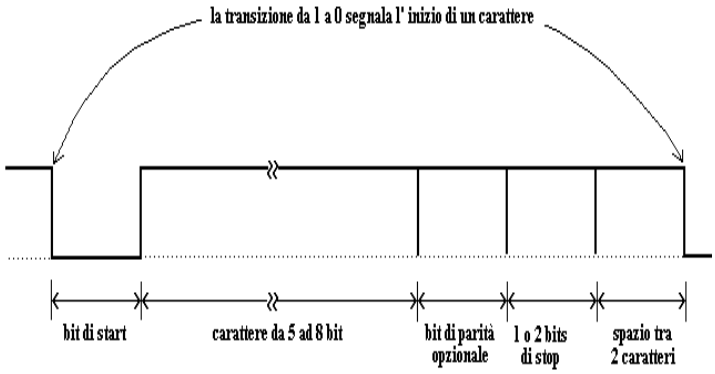



**Fig. 4-20 Formato frame della trasmissione asincrona**

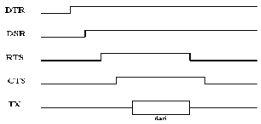



**Fig. 4-21 Handshaking per la trasmissione Tx**

Con riferimento all'esempio precedente sono mostrate in Fig. 4-22 le evoluzioni dei segnali di handshaking dell'interfaccia ricevente.


**Fig. 4-22 Segnali di hanshaking dell'interfaccia Rx**

Un esempio di connessione di due interfacce seriali è rappresentato in Fig. 4-23.

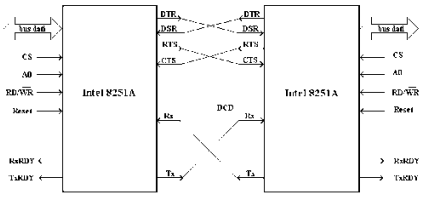


**Fig. 4-23 Connessione completa fra dispositivi seriali**

Oltre a questo tipo di connessione che citerò con l'aggettivo "piena" è possibile simulare anche una connessione "parziale", la quale coinvolge solo le linee **RX** e **TX** come è mostrato in Fig. 4-24.

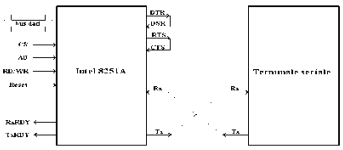


**Fig. 4-24-Connessione parziale fra dispositivi seriali**

## 4.10.6 Collegamento software

**Fig. 4-25-La finestra dei registri del dispositivo USART**

## 4.10.7 Modello di programmazione

I registri programmabili dell'interfaccia seriale sono quelli presenti in Fig. 4-25 con l'esclusione dei due shift register **TSHIFT** e **RSHIFT**.

Per accedere ad uno di questi 7 registri ad 8 bit occorre che l'indirizzo sia uguale a **Indirizzo1** o **Indirizzo2**. La selezione dei registri in questo dispositivo avviene attraverso il bit meno significativo dell'indirizzo, il tipo di accesso e lo stato del sistema.

Il registro che contiene le informazioni relative al genere di trasmissione che vogliamo realizzare è stato chiamato **MODE**. Esso può essere accesso solo in scrittura, all'indirizzo dispari e quando l'interfaccia è stata appena resettata. Il significato dei bit nel registro **MODE** sono illustrati nella Tabella 4-22.

| bit | significato |
|---|---|
| 0 | determina il **tipo di trasmissione**, ad 1 per quella sincrona e ad 1 per quella asincrona |
| 1 | non utilizzato |
| 2/3 | indicano il **numero di bit d'informazione** per ogni carattere trasmesso: 00 per 5 bit, 01 per 6, 10 per 7 e 11 per 8 |
| 4 | il valore 1 di questo bit abilita la **presenza del bit di parità** nel carattere trasmesso |
| 5 | il **tipo di parità** è pari se il bit ha valore 1 altrimenti è dispari |
| 6 | determina in una trasmissione asincrona il **numero di bit di stop** per ogni carattere trasmesso: 0 per un solo bit di stop e 1 per 2 |
| 7 | in una trasmissione sincrona definisce il **numero di caratteri di sincronismo** che devono essere trasmessi all'inizio della trasmissione: è posto a 0 per un carattere o ad 1 per due caratteri |

**Tabella 4-22-Significato dei bit nel registro MODE**

Se con il valore inserito in **MODE** abbiamo deciso di realizzare una trasmissione sincrona, la prossima scrittura all'indirizzo dispari ci permetterà di inserire il primo carattere di sincronismo nel registro **SYNC1**, invece , qualora abbiamo deciso che i caratteri di sincronismo devono essere due, dobbiamo effettuare una ulteriore scrittura all'indirizzo dispari per inserire il secondo carattere di sincronismo in **SYNC2**.

Dopo la scrittura nel registro **MODE,** se trasmettiamo in modo asincrono, o dopo l'inserimento del o dei caratteri di sincronismo, se, invece, trasmettiamo in modo sincrono, ogni ulteriore accesso in scrittura all'indirizzo

| bit | azione svolta se il bit è posto ad 1 |
|---|---|
| 0 | abilita il trasmettitore |
| 1 | attiva il segnale di handshaking DTR |
| 2 | abilita il ricevitore |
| 3 | non utilizzato |
| 4 | cancella i 3 bit d'errori nel registro STATUS |
| 5 | attiva il segnale di handshaking RTS |
| 6 | resetta l'interfaccia seriale |
| 7 | conduce il ricevitore nello stato "hunt", in questo stato il ricevitore cerca il o i caratteri di sincronismo memorizzati in SYNC1 e SYNC2 |

dispari viene eseguito nel registro **CNTRL**, il quale controlla il funzionamento dell'interfaccia seriale.

Il significato dei bit nel registro **CNTRL** sono illustrati nella Tabella 4-23.

Ovviamente, se resettiamo il componente, la sequenza di scrittura nei vari registri citati si ripete iniziando dal registro **MODE**.

Nel registro **DATIN** viene copiato il carattere che è stato ricevuto in **RSHIFT** per essere letto dal processore. **DATIN** è accessibile solo in lettura all'indirizzo pari. Un accesso in lettura in esso azzera il bit 1 di **STATUS**.

**Tabella 4-23-Significato dei bit nel registro CNTRL**

Nel registro **DATOUT** viene scritto dal processore il carattere che deve essere copiato in **TSHIFT** per essere trasmesso. **DATOUT** è accessibile solo in scrittura all'indirizzo pari; un tale tipo di accesso azzera il bit 0 di **STATUS**.

I registri **RSHIFT** e **TSHIFT** sono degli shift register non accessibili, ma vengono utilizzati dall'interfaccia per effettuare rispettivamente la trasformazione del formato serie/parallelo in ricezione e parallelo/serie in trasmissione.

Nel registro **STATUS** sono contenute informazioni sull'interfaccia che possono essere utilizzate dal programma che è in esecuzione; esso è accessibile in lettura all'indirizzo pari. Il significato dei bit nel registro **STATUS** sono illustrati nella Tabella 4-24.

Si riscontra un errore di parità, quando il numero di '1'nei bit d'informazione più il bit di parità non corrisponde

| bit | informazione indicata quando assume il valore 1 |
|---|---|
| 0 | è stato copiato il carattere da DATOUT in TSHIFT ed inizia la trasmissione, questo bit si azzera quando il processore scrive un nuovo carattere in DATOUT |
| 1 | è stato ricevuto un carattere in RSHIFT ed è stato copiato in DATIN, questo bit si azzera quando il processore legge il dato da DATIN |
| 2 | in una trasmissione sincrona il trasmettitore non ha nessun carattere da trasmettere |
| 3 | è stato rilevato un errore di parità |
| 4 | è stato rilevato un errore di overrun |
| 5 | è stato rilevato un errore di framing |
| 6 | è stato rilevato il o i caratteri di sincronismo previsti |
| 7 | è stato attivato il segnale di handshaking DSR |

al tipo di parità prevista.

**Tabella 4-24-Significato dei bit nel registro STATUS**

Un errore di overrun si verifica quando viene ricevuto un nuovo dato in DATIN, prima che il precedente sia stato letto dal processore, oppure quando il processore inserisce il prossimo carattere in DATOUT, prima che venga completamente trasmesso il dato precedente.

Un errore di framing si riscontra quando in una trasmissione asincrona il ricevitore si aspetta di rilevare un bit di stop e, invece, rileva uno zero.

Un quadro riassuntivo sugli indirizzamenti dei vari registri è mostrato in Tabella 4-25.

- il registro che viene selezionato dipende dallo stato del sistema.

| Indirizzo | Tipo di accesso | Registro |
|---|---|---|
| Pari | R | **DATIN** |
| Pari | W | **DATOUT** |
| Dispari | R | **STATUS** |
| Dispari | W | *** MODE, CNTRL, SYNC1 e SYNC2** |

**Tabella 4-25-Indirizzamento dei registri dell'interfaccia seriale**

### 4.10.8 Programmazione

Prima di iniziare una comunicazione, appena dopo aver resettato l'interfaccia, dobbiamo scrivere un byte all'indirizzo dispari, che ci permette di accedere al registro **MODE,** per determinare il tipo di comunicazione che vogliamo effettuare (sincrona o asincrona) e il formato del carattere (numero di bit d'informazione, bit di parità, tipo di parità, numero di bit di stop per comunicazione asincrona e numero di caratteri di sincronismo per comunicazione sincrona).

Se la comunicazione è sincrona, dobbiamo scrivere, sempre all'indirizzo dispari, i caratteri di sincronismo previsti in **SYNC1** ed eventualmente in **SYNC2**.

I prossimi accessi in scrittura all'indirizzo dispari ci permettono di accedere in **CNTRL** e, quindi, di abilitare il trasmettitore e/o il ricevitore, di attivare le linee di handshaking **RTS** e **DTR**, di cancellare i bit d'errore nel registro **STATUS**. Inoltre se l'interfaccia , in una comunicazione sincrona, è stata abilitata a ricevere, deve essere fissato ad 1 il bit 7 di **CNTRL** per iniziare la ricerca dei caratteri di sincronismo.

Se la comunicazione, invece, è asincrona, dopo aver caricato **MODE,** eseguendo una scrittura all'indirizzo dispari si accede direttamente a **CNTRL**. Essa ci permette di effettuare le stesse azioni descritte per la comunicazione sincrona tranne l'inserimento nel modo "hunt".

In entrambi i casi da questo momento in poi, tutti gli accessi in scrittura all'indirizzo dispari selezionano il registro **CNTRL**. In ogni istante è , comunque, possibile resettare il componente ponendo ad 1 il bit 6 di questo registro.

Esempi di sequenze di inizializzazione dell'interfaccia seriale sono mostrati nelle Tabella 4-26 e Tabella 4-27. In entrambi gli esempi si presuppone che A0 sia stato precedentemente caricato con l'indirizzo più basso associato all'interfaccia seriale.

```
move.b    #$5d,1(A0)
move.b    #$37,1(A0)
```

**Tabella 4-26**

In Tabella 4-26 la prima istruzione, scrivendo in **MODE**,

- imposta la trasmissione come di tipo asincrona,
- fissa il numero di  bit d'informazione ad 8,
- abilita la presenza del bit di parità dispari,
- fissa ad 1 il numero di bit di stop.
- La seconda istruzione scrive nel registro **CNTRL**, essa
- cancella i bit d'errore nel registro **STATUS**,
- abilita il trasmettitore ed il ricevitore,
- attiva i segnali di handshaking **DTR** ed **RTS**.

```
move.b    #$84,1(A0)
move.b    #$16,1(A0)
move.b    #$b7,1(A0)
```

**Tabella 4-27**

La prima istuzione in Tabella 4-27, scrivendo in **MODE**,

- imposta la trasmissione come di tipo sincrona,
- fissa il numero di  bit d'informazione a 6,
- disabilita la presenza del bit di parità,
- fissa ad 1 il numero dei caratteri di sincronismo.

La seconda istruzione memorizza il carattere di sincronismo nel registro **SYNC1**, il carattere scelto è pari a 16 esadecimale.

L'ultima scrittura avviene nel registro **CNTRL**, essa

- cancella i bit d'errore nel registro **STATUS**,
- abilita il trasmettitore ed il ricevitore,
- abilita la ricerca dei caratteri di sincronismo,
- attiva i segnali di handshaking **DTR** ed **RTS**.

Questo componente permette di realizzare operazioni di I/O sia in modo programmato che attraverso interruzioni.

In entrambi i casi, se desideriamo leggere i caratteri ricevuti, dobbiamo ogni volta controllare che i bit 3, 4 e 5 (questo solo per comunicazione asincrona) del registro **STATUS** siano pari a zero, cioè che non vi sia stato nessun errore nella trasmissione del carattere.

Nell'eseguire operazioni di I/O in modo programmato deve essere continuamente esaminato il valore di uno dei due bit meno significativi di **STATUS**.

In particolare, se vogliamo trasmettere un carattere, prima di inserirlo nel registro buffer **DATOUT**, dobbiamo verificare che il bit 0 sia pari ad 1, assicurandoci, in questo modo, che il carattere precedentemente inserito in **DATOUT** sia stato già trasferito in **TSHIFT**, evitando così un errore di *overrun*.

Invece, quando intendiamo leggere un carattere dal registro buffer **DATIN**, dobbiamo verificare che il bit 1 sia pari ad 1, questo significa che il carattere ricevuto in **RSHIFT** è stato trasferito in **DATIN**. Qualora il carattere non venga letto ed un nuovo carattere viene ricevuto, esso viene trasferito in **DATAIN** cancellando il carattere precedente e, quindi, causando un errore di *overrun*.

In alternativa le operazioni di I/O possono essere realizzate con l'ausilio delle interruzioni specificate nei parametri **COM2** e **COM3**. La routine associata alla prima interruzione deve assicurarsi di eventuali errori nella ricezione e quindi leggere il carattere da **DATIN**, mentre, quella associata all'interruzione specificata in **COM3** deve provvedere a scrivere in **DATOUT** il prossimo carattere da trasmettere.

### 4.10.9 Il comportamento.

In questo paragrafo verrà descritto il comportamento del dispositivo nei vari modi di funzionamento facendo implicito riferimento che le interruzioni vengono inviate al gestore delle interruzioni specificato in **COM1** e che le linee di handshaking e le linee di ricezione e trasmissione dati sono collegate al device specificato in **COM4**.

Dato che la frequenza del segnale di clock applicato all'interfaccia seriale è almeno 30 volte inferiore a quella applicata al processore, bisogna ridurre la frequenza di clock, simulata, applicata al modulo che simula l'interfaccia seriale.

Questo può essere realizzato con le seguenti operazioni:

1) attivare il menù **Device** dalla finestra principale;

2) scegliere il comando **Speed**;

3) inserire un valore maggiore di uno nella finestra che appare.

Maggiore sarà il valore inserito più bassa sarà la frequenza del segnale di clock applicata all'interfaccia. Il valore inserito indica dopo quanti colpi di clock, applicati al processore, viene mandato uno all'interfaccia.

Ovviamente, i colpi di clock cui faremo riferimento successivamente saranno quelli applicati all'interfaccia seriale e non al processore.

### 4.10.10 Trasmissione asincrona

Il trasmettitore è pronto a trasmettere solo dopo la sua abilitazione e dopo aver ricevuto i segnali **DSR** e **CTS**; tuttavia, fino a quando non viene inserito un carattere in **DATOUT**, sarà trasmesso il valore '1'per ogni colpo di clock.

Quando scriviamo un carattere in **DATOUT**, viene azzerato il bit 0 di **STATUS**, poi al prossimo impulso di clock:

- il valore in **DATOUT** viene copiato in **TSHIFT**,

- viene fissato ad 1 il bit 0 di **STATUS**,
- viene inviata un'interruzione sulla linea specificata in **COM3** ,
- infine viene trasmesso un bit '0', detto bit di start.

Successivamente per ogni impulso di clock viene trasmesso il bit 0 di **TSHIFT** e contemporaneamente il contenuto di questo registro viene shiftato di una posizione verso destra. Questo comportamento continua fin quando non vengono trasmessi tutti i bit d'informazione previsti in **MODE**. Se il carattere contiene più bit di quelli da noi previsti, allora i bit in eccesso non saranno presi in considerazione.

Il prossimo bit ad essere trasmesso, se esso è stato richiesto in **MODE**, è il bit di parità, con il giusto tipo di parità, altrimenti viene trasmesso un bit "1" citato come bit di stop. Se in **MODE** sono stati fisssati due bit di stop, al prossimo colpo di clock verrà trasmesso un'altro "1" riportandoci nello stato iniziale.

Se nel frattempo non è stato inserito un nuovo carattere in **DATOUT**, sarà trasmesso un "1" per ogni successivo colpo di clock.

### 4.10.11 Trasmissione sincrona

Dopo aver impostato la trasmissione come di tipo sincrona , dopo aver abilitato il trasmettitore e ricevuto i segnali **DSR** e **CTS**, il trasmettitore è pronto a trasmettere, ma fin quando non viene inserito un carattere in **DATOUT**, verrà posto ad 1 il bit 3 di **STATUS** e saranno trasmessi in continuazione i caratteri di sincronismo.

Precisamente il prossimo impulso di clock produce le seguenti azioni:

- copia del valore di **SYNC1** in **TSHIFT**,
- trasmissione del bit 0 di **TSHIFT**,
- shift verso destra di una posizione del valore in **TSHIFT**.

Successivamente per, ogni impulso di clock, vengono ripetute le ultime due operazioni fino alla completa trasmissione degli otto bit del primo carattere di sincronismo. Se deve essere trasmesso anche il secondo carattere di sincronismo, tutte le operazioni sono ripetute, con la differenza che in **TSHIFT** viene caricato **SYNC2** invece che **SYNC1**.

Se, nel frattempo, non è stato caricato in **DATOUT** nessun carattere, viene posto ad 1 il bit 3 di **STATUS** e vengono trasmessi di nuovo i caratteri di sicronismo previsti. In generale questo avviene ogni volta che il trasmettitore viene a trovarsi nella condizione di *underrun*.

Quando scriviamo un carattere in **DATOUT** viene azzerato il bit 0 di **STATUS** e al prossimo impulso di clock viene:

- copiato il valore di **DATOUT** in **TSHIFT**,
- posto ad 1 il bit 0 di **STATUS**,
- azzerato il bit 3 di **STATUS**,
- inviata un'interruzione sulla linea specificata in **COM3**,
- trasmesso il bit 0 di **TSHIFT**,
- shiftato verso destra di una posizione il valore in **TSHIFT**.

Le due ultime operazioni vengono ripetute ad ogni impulso di clock fin quando non vengono trasmessi tutti i bit d'informazione previsti in **MODE**. Anche in questo caso, se il carattere contiene più bit di quelli da noi previsti, i bit in eccesso non saranno presi in considerazione.

Il prossimo bit ad essere trasmesso, se esso è stato richiesto in **MODE**, è il bit di parità, con il giusto tipo di parità.

Se, nel frattempo, non è stato inserito un nuovo carattere in **DATOUT**,il trasmettitore viene a trovarsi nella condizione di *underrun* e quindi verrà posto ad 1 il bit 3 di **STATUS** e verranno trasmessi di nuovo i caratteri di sicronismo previsti.

### 4.10.12 Ricezione asincrona

Quando il ricevitore è stato abilitato ed è stato attivato il segnale **DSR**,esso è pronto a ricevere.

Se viene ricevuto un dato sulla linea **RX**, esso è inserito sempre nel bit più significativo di **RSHIFT**, dopo che quest'ultimo è stato shiftato di una posizione verso destra.

Il ricevitore inizia a considerare i bit in arrivo come bit d'informazione solo quando riceve il bit di start.

Una volta ricevuto l'ultimo bit d'informazione del carattere trasmesso, il valore in **RSHIFT** viene shiftato di un numero di posizioni verso destra pari a otto meno il numero di bit d'informazioni trasmesso, in modo da spostare il carattere al limite destro di **RSHIFT**.

A questo punto viene copiato il dato di **RSHIFT** in **DATIN** e se il bit 1 di **STATUS** è uno, cioè se il carattere ricevuto prima non è stato letto dal processore, viene segnalato un errore di *overrun* ponendo il bit 4 di **STATUS** ad uno.

Il prossimo bit ricevuto se presente è il bit di parità, se viene scoperto un errore di parità, esso viene segnalato ponendo ad uno il bit 3 di **STATUS**.

Successivamente il ricevitore si aspetta uno o due bit di stop, se, invece, viene ricevuto uno zero, esso segnala un errore di *framing* ponendo ad uno il bit 5 di **STATUS**.

Quando viene ricevuto l'ultimo bit di stop viene posto ad uno il bit 1 di **STATUS** ed è inviata un'interruzione del tipo specificata in **COM2**.

Quando poi il processore preleva il dato da **DATIN** il bit 1 di **STATUS** viene azzerato.

### 4.10.13 Ricezione sincrona

Anche in questo caso il ricevitore è pronto a ricevere, se esso è stato abilitato ed è stato attivato il segnale **DSR**.

Se viene ricevuto un dato, esso è inserito sempre nel bit più significativo di **RSHIFT** dopo che quest'ultimo è stato shiftato di una posizione verso destra.

Le differenze con la ricezione asincrona sono che non esistono nè bit di start, nè bit di stop e che, se il ricevitore è nello stato *hunt,* esso ricerca non i bit d'informazione ma i caratteri di sincronismo. Quando riconosce i caratteri di sincronismo memorizzati all'inizio in **SYNC1** ed eventualmente in **SYNC2,** esso azzera il bit7 di **CNTRL**, il quale fa terminare la ricerca, e pone ad uno il bit 6 di **STATUS**.

Una volta ricevuto tutti i bit d'informazione e l'eventuale bit di parità e dopo aver segnalato i possibili errori di *overrun* e di *parità,* in modo identico a quanto visto per il ricevitore asincrono, viene posto ad uno il bit 1 di **STATUS** ed è inviata un'interruzione del tipo specificata in **COM2**.

Anche per la ricezione sincrona, quando il dato è prelevato dal processore da **DATIN**, viene azzerato il bit 1 di **STATUS**.

## 4.11 Dispositivo Tappo

La funzione di questo dispositivo e di rispedire in modo opportuno, i segnali che gli arrivano in ingresso, all'interfaccia seriale o al terminale seriale che li aveva inviati.

In questo modo il terminale o l'interfaccia riceve i caratteri trasmessi precedentemente.

Nel collegare ad un'interfaccia seriale un dispositivo periferico, qual'è il bloccatore, sono possibili, come abbiamo visto in precedenza, due tipi di connessioni: piena e parziale.

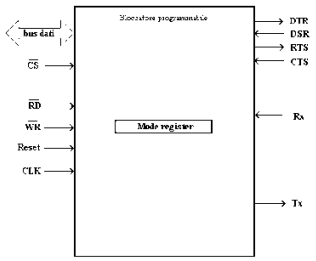



**Fig. 4-26**

E' stato simulato un bloccatore programmabile, il cui modello è presentato in Fig. 4-26, che permette di gestire sia connessioni parziali che piene, in funzione del valore presente nel registro di modo.

Se il componente è stato inizializzato per gestire una connessione piena, esso realizza i collegamenti tra le linee **DSR** e **DTR**, tra **CTS** ed **RTS**, e tra **Tx** e **Rx**. (figura 10). Invece, per una connessione parziale il bloccatore effettua il solo collegamento tra **Tx** ed **Rx** (figura 11).

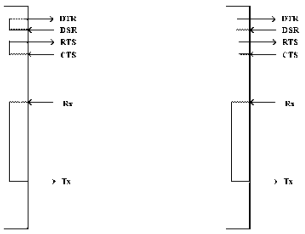



**Fig. 4-27**

Si è voluto simulare anche un bloccatore più semplice, che non è collegato al sistema bus e quindi non è programmabile, che connette le linee sempre come illustrato in Fig. 4-27. Il modello di questo tipo di bloccatore è mostrato in Fig. 4-28.

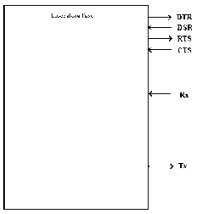



**Fig. 4-28**

Il nome del modulo che simula il bloccatore, sia esso programmabile o non, è STOPPER.

### 4.11.1 Collegamento al processore e ad un device

Il bloccatore programmabile presenta, rispetto all'interfaccia seriale descritta nel capitolo precedente, sostanzialmente le stesse linee verso il device e verso il processore. Quindi le proprietà per l'interfacciamento sono uguali a quelle descritte per l'interfaccia seriale.

Due esempi di collegamento sono mostrati in Fig. 4-29 e Fig. 4-30.

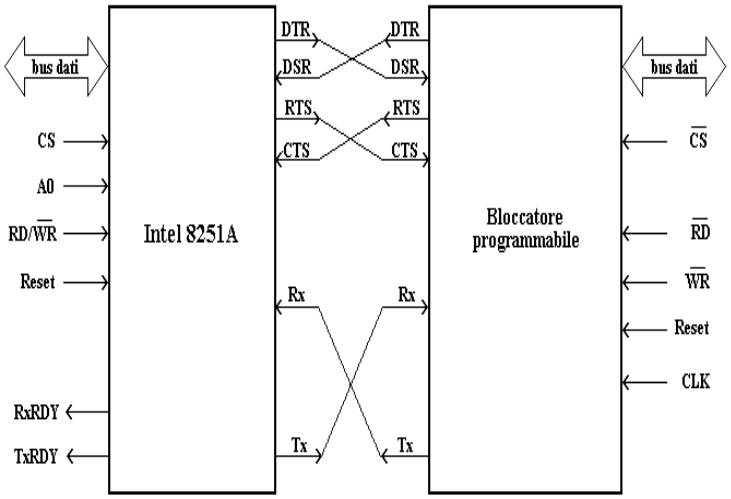



**Fig. 4-29**

I collegamenti illustrati in queste figure sono realizzabili anche con il bloccatore fisso, con la differenza che manca il suo collegamento al bus.

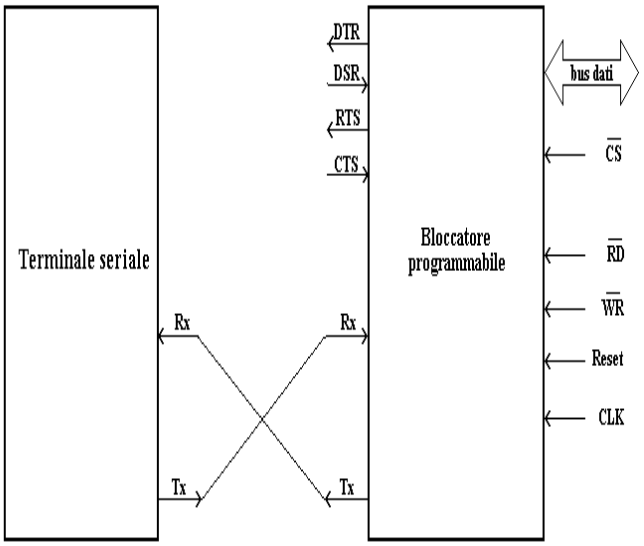



**Fig. 4-30**

### 4.11.2 Collegamento software

L'inserimento del bloccatore in una configurazione di ASIM si differenzia in funzione del tipo di bloccatore da simulare. In particolare, se vogliamo inserire il bloccatore fisso, dobbiamo definire solo i parametri **Nome Elemento**, **Identificatore** e **Com4**, invece, per quello programmabile bisogna fissare anche **Indirizzo1** e **BUS**.

**Nome Elemento** e **Identificatore** hanno sempre lo stesso significato devono essere pari rispettivamente a "*STOPPER*" e ad un *numero compreso tra 01 ed FF*.

**Indirizzo1** rappresenta l'indirizzo per accedere al componente. Questo parametro permette di definire l'indirizzo sul bus indirizzi che attiva la linea **CS** e quindi che seleziona il componente.

**BUS** determina l'*Identificatore del bus* a cui è connesso il bloccatore programmabile. Quindi, scegliendo per **BUS** l'identificatore di un componente MMU/BUS, a cui è stato collegato il dispositivo di tipo CPU, si determina la connessione delle linee **bus dati**, **RD**,**WR** e **Reset** al processore suddetto.

**COM4** specifica l'*Identificatore del Device* a cui è connesso il bloccatore, consentendo così di definire il collegamento delle linee **Tx** e **Rx** ed eventualmente delle linee **DSR**, **DTR**, **CTS** ed **RTS**, con un componente connettibile al bloccatore.

La finestra associata a STOPPER è quella in figura 15.

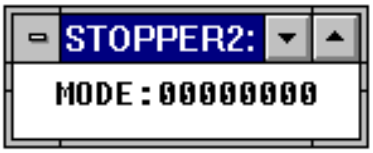



**Fig. 4-31**

### 4.11.3 Modello di programmazione

Il bloccatore programmabile ha un solo registro programmabile: il registro **MODE**. Per accedere a questo registro ad 8 bits occorre che l'indirizzo sia uguale a **Indirizzo1** e gli accessi possonoessere sia in scrittura che in lettura.

In **MODE** l'unico bit che ha significato è il meno significativo. Se esso è fissato a 0 il bloccatore realizza i collegamenti tra le linee **DSR** e **DTR**, tra **CTS** ed **RTS**, e tra **Tx** e **Rx**, altrimenti, esso effettua il solo collegamento tra **Tx** ed **Rx**.

Data la semplicità della programmazione di questo dispositivo si omette l'esempio relativo.

## 4.12 Il dispositivo di collegamento tra nodi: ELABNODE.

Con i dispositivi MMU/BUS e 1to4BUSINT, si è mostrato come in ASIM sia possibile costruire macchine con più processori; questi processori possono interagire tra loro scambiandosi informazioni e dati utilizzando aree di memoria comune, cioè rispettando il cosiddetto "modello a memoria comune". Esistono tipi di architetture per macchine MIMD che fanno invece riferimento ad un "modello a scambio di messaggi" e sono realizzate utilizzando nodi di elaborazione; questi sono costituiti tipicamente da processore e memoria e sono connessi attraverso un certo numero di link; ogni comunicazione tra i vari processori avviene attraverso la rete costituta da questi link; le tipologie più comuni sono, ad esempio, quelle ad ipercubo e mesh. Il dispositivo in esame è stato introdotto per simulare con ASIM anche questo tipo di architetture.

ELABNODE è un dispositivo che presenta quattro link in ingresso(I1, I2, I3, I4) e quattro in uscita (O1, O2, O3, O4); ad ogni link è associato un registro a trentadue bit che è indicato con lo stesso nome del link. Un'operazione di trasferimento dati consiste nell'inviare o ricevere il contenuto di uno di questi registri. La sincronizzazione nel trasferimento di un dato avviene sfruttando un registro di controllo/stato (CN) ad otto bit. I quattro bit meno significativi sono associati ai link in ingresso e, se alti, assumono il significato di dato pronto; i quattro bit più significativi sono associati ai link in uscita e, se alti, assumono il significato di link occupato.

ELABNODE può essere connesso ad un bus, occupando gli indirizzi da FFFFFFEF a FFFFFFFF; al primo indirizzo si può accedere solo con un'operazione di scrittura o lettura sul byte e corrisponde al registro CN; Inoltre è possibile accedere, solo con un'operazione di scrittura o lettura su doppia parola, agli indirizzi: FFFFFFF0, FFFFFFF4, FFFFFFF8, FFFFFFFC; a questi corrispondono, in lettura, i registri I1, I2, I3, I4 ed, in scrittura, i registri O1, O2, O3, O4.

ELABNODE può essere connesso anche ad un dispositivo per la gestione delle interruzioni; ad ogni link di ingresso corrisponde una differente linea di interruzione. Quando, ad esempio, sul link I1 arriva un dato, il bit 0 (corrispondente ad I1) del registro CN va ad 1 e viene inviata l'interruzione prevista; quando il processore legge il dato da I1, ELABNODE segnala al mittente l' avvenuta ricezione e quindi la disponibilità a ricevere un nuovo dato. Inoltre, riporta a 0 il bit 0 di CN.

Per inviare un dato, ad esempio sul link O1, il processore deve innanzitutto controllare che il link non sia occupato da un precedente trasferimento, cioè deve verificare che il bit 4 (corrispondente ad O1) di CN non sia 1; poi può scrivere il dato che ELABNODE provvede ad inviare dopo aver posto ad 1 il bit 4 di CN; tale bit torna a 0 a trasferimento completato segnalando così la disponibilità ad inviare un nuovo dato.

Vi sono due possibili modalità nell'invio dei dati: la prima tiene conto della tempificazione e prevede l'invio del dato durante il primo colpo di clock successivo alla scrittura (questo tipo di collegamento sarà detto, nel seguito, "tempificato"); la seconda non tiene conto della tempificazione e prevede l' invio del dato immediatamente dopo l'operazione di scrittura. Nel primo caso è possibile, variando la velocità con il comando *Velocità* del menù *Schedulatore*, simulare link con diverso tempo di trasferimento.

La finestra associata ad ELABNODE (Fig. 4-32) mostra il contenuto di tutti i registri; questi possono essere direttamente modificati con il comando *Modifica Valore* del menù *Nodo*. All'accensione della macchina da simulare, tutti i registri sono al valore zero.

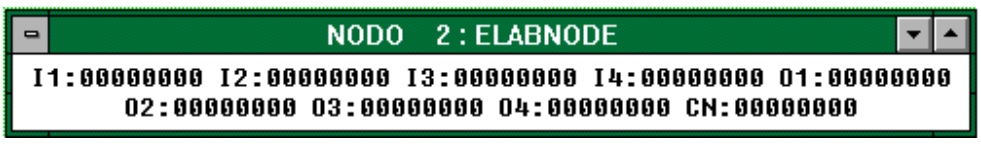



**Fig. 4-32**

Per costruire un nodo di elaborazione occorre definire almeno tre componenti: una CPU, un MMU/BUS ed un ELABNODE; si è detto almeno tre, perché nulla vieta di connettere al bus un numero qualsiasi di dispositivi anche se, tipicamente, il nodo di elaborazione è costituito da processore, memoria e link di connessione. Poiché gli indirizzi associati ai registri di ELABNODE sono fissi, non è possibile connettere ad un bus più di un dispositivo del tipo in esame. Si è già visto come definire CPU e MMU/BUS; per quanto riguarda ELABNODE occorre specificare i parametri che compaiono in una finestra di dialogo del tipo di quella mostrata in figura 3 cui si accede selezionando il comando *Aggiungi Nodo* del menù *Configura*.

Il "Nome Elemento" è ovviamente ELABNODE. "Indirizzo 1" e "Indirizzo 2" definiscono le interruzioni connesse ai link in questo modo:

quattro cifre meno significative di "Indirizzo 1" -> interruzione associata ad I1;

quattro cifre più significative di "Indirizzo 1" -> interruzione associata ad I2;

quattro cifre meno significative di "Indirizzo 2" -> interruzione associata ad I3;

quattro cifre più significative di "Indirizzo 2" -> interruzione associata ad I4.

Il significato delle quattro cifre è quello più volte presentato.

"BUS" consente di definire il bus (due cifre meno significative) e il gestore delle interruzioni (due cifre più significative) cui ELABNODE è collegato.

"Com1", "Com2", "Com3", "Com4" consentono di definire i dispositivi cui sono collegati i quattro link di uscita. Ad esempio, le due cifre meno significative di "Com1" sono destinate all'Identificatore del dispositivo da collegare ad O1; la terza cifra di "Com1" indica a quale link di ingresso del dispositivo a valle è connesso O1; i valori ammessi sono:

0 O1 è connesso al link I1 del dispositivo a valle;

1 O1 è connesso al link I2 del dispositivo a valle;

2 O1 è connesso al link I3 del dispositivo a valle;

3 O1 è connesso al link I4 del dispositivo a valle.

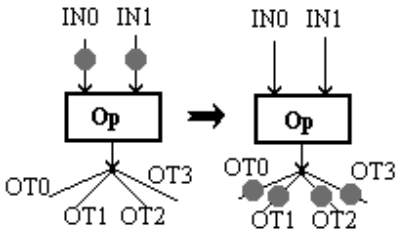
Infine la cifra più significativa di "Com1" consente di specificare se il collegamento è tempificato (cifra = 1) o non tempificato (cifra = 0).

Per "Com2", "Com3" e "Com4" vale quanto detto per "Com1" con riferimento, anziché ad O1, ad O2, O3 ed O4 rispettivamente.

Nell'attuale versione di ASIM i dispositivi collegabili ad un link sono ELABNODE e ARTMNODE.

### 4.13 Il dispositivo nodo "data flow": ARTMNODE.

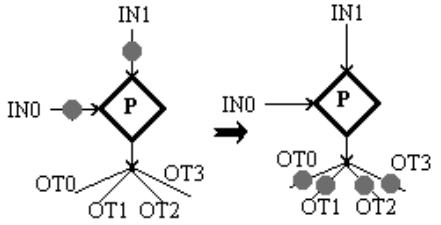
ASIM è un ambiente che consente di simulare anche architetture non Von Neuman come le macchine Data Flow. Il dispositivo ARTMNODE è stato introdotto per mostrare questa possibilità; esso definisce un nodo in grado di eseguire un operazione aritmetica su interi espressi su trentadue bit.




| Codice | Operatore | IN1 cost. |
|---|---|---|
| 00 | + | no |
| 01 | + | si |
| 02 | * | no |
| 03 | * | si |
| 04 | / | no |
| 05 | / | si |

**Fig. 4-33**

Per ogni nodo è possibile definire l'operazione che esso deve svolgere quando tutti gli operandi previsti si rendono disponibili; più nodi possono essere connessi tra loro per eseguire espressioni algebriche; grazie alla presenza di funzioni per il confronto, porte TRUE e FALSE e funzioni di Switch e Merge, è possibile definire macchine data flow in grado di eseguire semplici algoritmi; infine, essendo possibile collegare tra loro ELABNODE e ARTMNODE, si possono costruire macchine miste, combinazione di architetture convenzionali e data flow.




Il token in uscita è booleano: true se IN0 P IN1 è vero, false altrimenti. Se IN1 è costante, il token corrispondente è considerato sempre presente e basta l' arrivo di IN0 per lo scatto.

| Codice | Predicato | IN1 cost. |
|---|---|---|
| 10 | = | no |
| 11 | = | si |
| 12 | < | no |
| 13 | < | si |
| 14 | <= | no |
| 15 | <= | si |
| 16 | > | no |
| 17 | > | si |
| 18 | >= | no |
| 19 | >= | si |

**Fig. 4-34**

ARTMNODE presenta quattro registri collegati ai link in ingresso (IN0, IN1, IN2, IN3) e quattro registri collegati ai link in uscita (OT0, OT1, OT2, OT3); il numero di registri in ingresso, e quindi di link, utili dipende dalla funzione

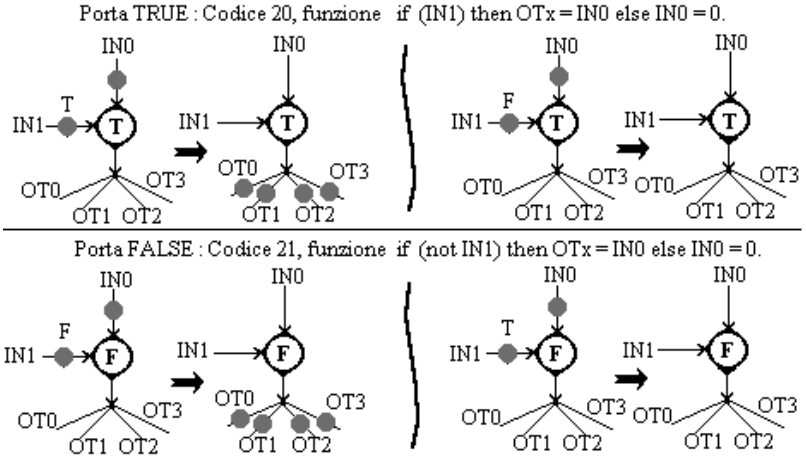



**Fig. 4-35**

svolta dal nodo. Ad esempio, se la funzione f prevede due operandi, solo IN0 e IN1 vengono utilizzati; quando sono arrivati entrambi i dati (token) sui link corrispondenti, viene eseguita l' operazione f(IN0,IN1); il

risultato viene contemporaneamente scritto sui registri di uscita ed inviato ad eventuali nodi connessi ai link; infine IN0 e IN1 vengono riportati al valore 0. Nel caso in cui una funzione prevede che una variabile sia costante, il registro che la contiene non viene azzerato. Se un nuovo dato arriva in un registro di ingresso prima che il dato precedente sia stato elaborato, il dato precedente viene perso. Le regole di scatto e le funzioni disponibili (con relativo codice) sono riportate nelle figure 13, 14, 15, 16.

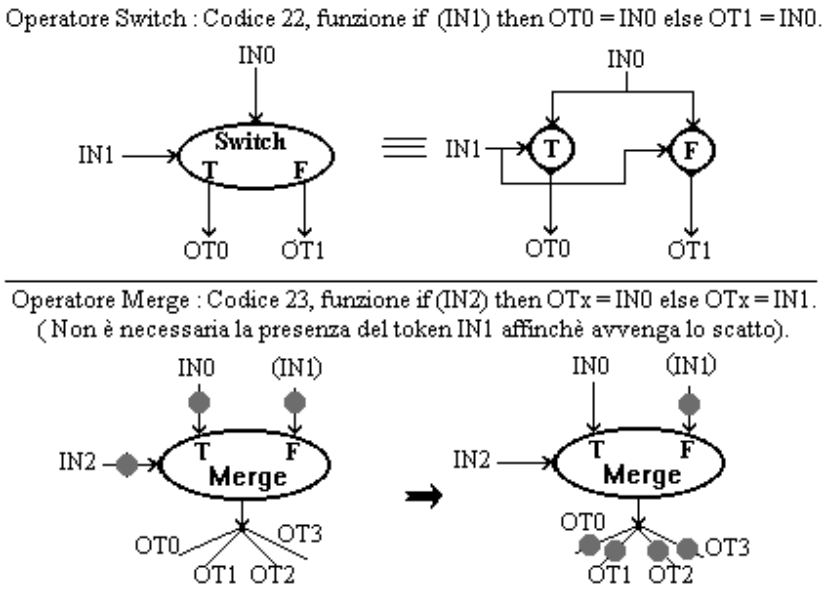



**Fig. 4-36**

La finestra associata ad ARTMNODE (17Fig. 4-37) riporta la funzione svolta dal nodo e tutti i registri (compresi quelli non utilizzati); il contenuto dei registri può essere modificato utilizzando il comando *Modifica Valore* del menù *Nodo*; si osservi che la modifica diretta è equivalente all' arrivo del dato sul link corrispondente al registro modificato, cioè produce gli stessi effetti.

Come nel caso di ELABNODE, vi sono due possibili modalità nell' invio dei dati (token): la prima tiene conto della tempificazione e prevede l' invio del dato durante il primo colpo di clock successivo alla scrittura sul registro di uscita; la seconda non tiene conto della tempificazione e prevede l' invio del dato immediatamente dopo l' operazione di scrittura. Nel primo caso è possibile, variando la velocità con il comando *Velocità* del menù *Schedulatore*, simulare link con diverso tempo di trasferimento.

Quando si simula un algoritmo utilizzando una macchina data flow e si vuole vedere l'esecuzione dell'algoritmo passo passo, è necessario tempificate qualche link di uno o più nodi. Solo in tal caso è, infatti, possibile utilizzare il comando *Passo* del menù *Schedulatore* più volte; altrimenti, l'intero algoritmo verrà eseguito in un sol passo. Se non viene tempificato nessun link si DEVE prevedere per l'algoritmo una condizione di terminazione, altrimenti non lo si potrà più fermare una volta avviato.

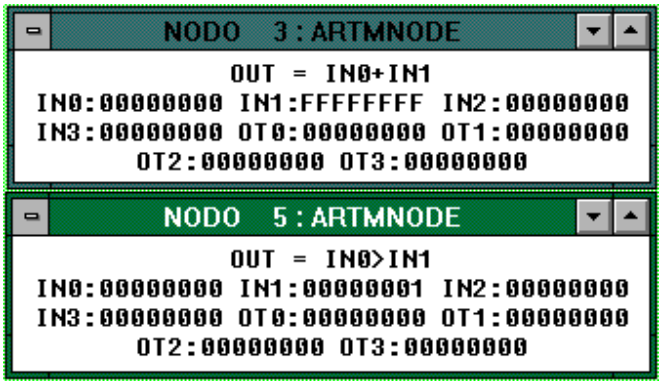



**Fig. 4-37**

Per definire un ARTMNODE occorre specificare i parametri che compaiono nella finestra di configurazione di ASIM e a cui si accede selezionando il comando *Aggiungi Nodo* del menù *Configura*.

Il "Nome Elemento" è ARTMNODE. "Indirizzo 1" consente di specificare la funzione che deve svolgere il nodo; il codice che corrisponde a ciascuna funzione è presentato nelle figure 13, 14, 15, 16. "BUS" non è utilizzato da questo dispositivo. "Com1", "Com2", "Com3", "Com4" consentono di definire i dispositivi cui sono collegati i quattro link di uscita. Ad esempio, le due cifre meno significative di "Com1" sono destinate all'Identificatore del dispositivo da collegare ad OT0; la terza cifra di "Com1" indica a quale link di ingresso del dispositivo a valle è connesso OT0; i valori ammessi sono:

0 OT0 è connesso al link IN0 del dispositivo a valle;

1 OT0 è connesso al link IN1 del dispositivo a valle;

2 OT0 è connesso al link IN2 del dispositivo a valle;

3 OT0 è connesso al link IN3 del dispositivo a valle.

Infine la cifra più significativa di "Com1" consente di specificare se il collegamento è tempificato (cifra = 1) o non tempificato (cifra = 0).

Per "Com2", "Com3" e "Com4" vale quanto detto per "Com1" con riferimento, anzichè ad OT0, ad OT1, OT2 ed OT3 rispettivamente.

# Parte II

# Esempi di configurazioni ASIM

# Parte III

## Progetti in ASIM

### Comunicazione parallela fra due sistemi a processore mediante dispositivi di I/O di tipo PIA (Parallel Interface Adapter)

Il sistema, riportato in fig.1, è costituito da due sistemi a processore autonomi e simmetrici nell'architettura e in grado di cooperare mediante porti paralleli.

In particolare, immettendo delle stringhe di caratteri per tramite della tastiera di uno dei dispositivi TERMINAL, a seguito di un invio volontorio (o di invio per buffer di tastiera full), si avvia il meccanismo di trasmissione dei dati verso il sistema gemello e, in particolare, verso il video del relativo dispositivo TERMINAL con ricorso ad un protocollo di handshaking (del tipo dato pronto-dato ricevuto), come mostrato in fig.2. La ricezione dei caratteri dalla tastiera fa uso di interruzioni.

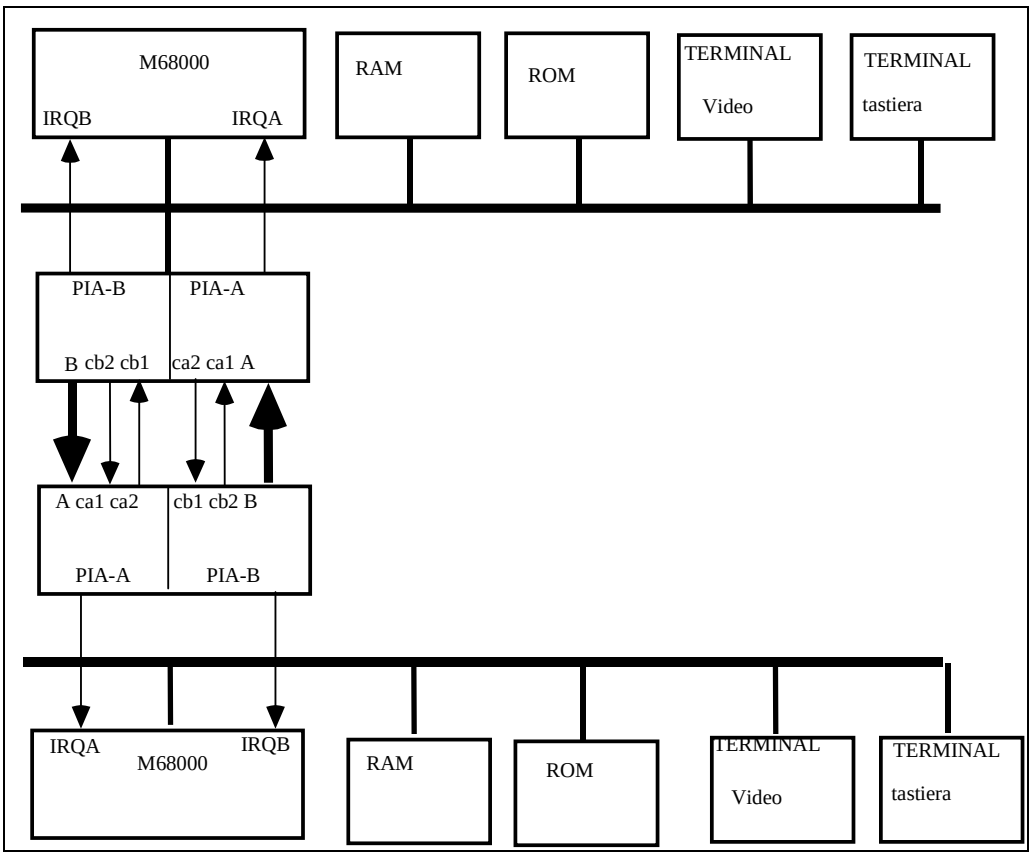


Figura 1 Configurazione ASIM del sistema di comunicazione parallela

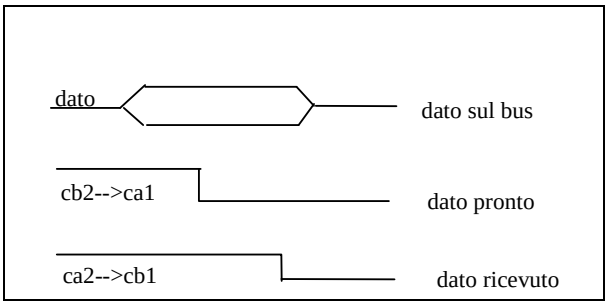



Figura 2 Protocollo di Handshaking

Il carattere ricevuto dal secondo porto parallelo viene, quindi, trasmesso al video (TERMINAL) del sistema ricevente mediante interruzioni generate sulla linea IRQA del porto parallelo.

Il programma handshaking (hands_semplice e hands_interall) serve a provare la configurazione "communic" costituita da due sistemi simmetrici ciascuno con un processore M68000, una ROM di 4K (addr $0-$3FFF), una RAM di 16K (addr $8000-$BFFF), un device parallelo PIA mappato a $2004, un device seriale di tipo TERMINAL mappato a $2000. I due PIA, interconnessi come in fig.1, mediante il protocollo di handshaking consentono ai due sistemi di scambiarsi i caratteri digitati sul dispositivo TERMINAL. I device interagiscono con i rispettivi processori mediante le linee di interruzione. In particolare, il dato immesso da tastiera è acquisito mediante interruzione (liv.2,autovettore 26 mappato in area ROM alla locazione $68 che punta alla ISR posta all'indirizzo $8500 in area RAM) e inviato alla sezione A del dispositivo parallelo PIA per la trasmissione verso il dispositivo PIA connesso all'altro sistema. Il carattere ricevuto dal PIA è gestito mediante interruzione. All'arrivo dell'interrupt la ISR lo acquisisce e lo invia al terminal per la visualizzazione.L'interruzione e' associata all'interrupt di liv. 1, #vect 25 mappato a $64 della ROM con ISR a $8700.

Ciascuno dei due sistemi ha un’architettura composta da un microprocessore MC68000 dotato di RAM, ROM e di dispositivi TERMINAL e PIA. I due sistemi si scambiano dati per tramite dei due porti paralleli (fig.1). In particolare,si fa uso di una ROM di 4K (addr $0-$3FFF), una RAM di 16K (addr $8000-$BFFF), un device parallelo PIA mappato a $2004, un deviceseriale di tipo TERMINAL mappato a $2000. Il datoimmesso da tastiera e' acquisito mediante interruzione (liv.2,autovettore 26 mappato in areaROM alla locazione $68 che punta alla ISR posta all'indirizzo $8600 in area RAM) e inviatoalla sezione A del dispositivo parallelo PIA per la trasmissione verso il dispositivo PIAconnesso all'altro sistema.Il carattere ricevuto dal PIA e' gestito mediante interruzione. All'arrivo dell'interrupt laISR lo acquisisce e lo invia al terminal per la visualizzazione.L'interruzione e' associataall'interrupt di liv. 1, #vect 25 mappato a $64 della ROM con ISR a $8500.

I parametri di configurazione in ASIM di tale sistema sono di seguito riportati:

```
     CHIP Name: MEMORY

RAM ad indirizzo base $8000 e di dimensione $10 kbyte.

Type: MMU/BUS.      Identif: 01.    BUS: 0000.
Addres 1: 00008000. Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

     CHIP Name: M68000
Type: CPU.          Identif: 02.    BUS: 0001.
Addres 1: 00009000. Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

     CHIP Name: TERMINAL
Type: Device.       Identif: 03.    BUS: 0001.
Addres 1: 00002000. Address 2: 00002001.
Com1: 0002.  Com2: 0001.  Com3: 0002.  Com4: 0000.

     CHIP Name: M6821PIA
Type: Device.       Identif: 04.    BUS: 0001.
Addres 1: 00002004. Address 2: 00002007.
Com1: 0002.  Com2: 0003.  Com3: 0004.  Com4: 0208.

     CHIP Name: MEMORY
Type: MMU/BUS.      Identif: 05.    BUS: 0000.
Addres 1: 00008000. Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

     CHIP Name: M68000
Type: CPU.          Identif: 06.    BUS: 0005.
Addres 1: 00009000. Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

     CHIP Name: TERMINAL
Type: Device.       Identif: 07.    BUS: 0005.
Addres 1: 00002000. Address 2: 00002001.
Com1: 0006.  Com2: 0001.  Com3: 0002.  Com4: 0000.

     CHIP Name: M6821PIA
Type: Device.       Identif: 08.    BUS: 0005.
Addres 1: 00002004. Address 2: 00002007.
Com1: 0006.  Com2: 0003.  Com3: 0004.  Com4: 0204.
```

Di seguito è riportato il codice in assembler di un programma per il test della configurazione. Il programma è composto da una parte che provvede all'inizializzazione del sistema e di tutti i suoi dispositivi e all'attrezzaggio delle ISR per la gestione delle interruzioni, e da una parte applicativa per il test della configurazione.

```
NAME communic       ;tale direttiva non e' allo stato supportata dall'assemblatore

            ORG     $8200

PIADA       EQU     $2004   ;indirizzo di PIA-A dato, usato in input
PIACA       EQU     $2005   ;indirizzo di PIA-A stato/controllo
PIADB       EQU     $2006   ;indirizzo di PIA-B dato, usato in output
PIACB       EQU     $2007   ;indirizzo di PIA-B controllo

TERD        EQU     $2000   ;indirizzo di TERMINAL registro dato
TERC        EQU     $2001   ;indirizzo di TERMINAL registro Stato-Controllo


BEGIN       JSR DVAIN       ;inizializza PIA porto A
            JSR     DVBOUT          ;inizializza PIA porto B
            JSR     DVTER           ;inizializza terminal
            MOVE.W  SR,D0           ;legge il registro di stato
            ANDI.W  #$D8FF,D0       ;maschera per reg stato (stato utente, int abilitati)
            ORI.w   #$8000,D0       ;messa per provare il bit di trace
```

```
            MOVE.W  D0,SR           ;pone liv int a 000
            ADD     d0,d1           ;messa per provare il bit    di trace

            LOOP    JMP LOOP        ;ciclo caldo dove il processore attende interrupt


*
DVAIN       MOVE.B  #0,PIACA                ;seleziona il registro direzione del porto A
            MOVE.B  #$00,PIADA              ;pone le linee di A a linee di input
            MOVE.B  #%11100101,PIACA        ;
            MOVE.B  PIADA,D0                ;lettura fittizia per inizializzare CA2 ad 1
            RTS

DVBOUT      MOVE.B  #0,PIACB                ;seleziona il registro direzione di PIA porto B
            MOVE.B  #$FF,PIADB              ;pone le linee di PIA B a linee di output
            MOVE.B  #%11111100,PIACB
            RTS

DVTER       MOVE.B  #$3f,TERC               ;seleziona indirizzo stato/controllo
*                                           ;00111111 {int abilitato su buff full, int ab. su enter,
*                                           ;canc. video, clear buff tastiera, abil. tastiera, abil. eco
            RTS



ISR per la gestione dato proveniente dalla tastiera di TERMINAL e spedito, per tramite del PIA porto B,
all'altro sistema.ISR associata all'interrupt di liv. 1, #vect 25 mappato a $64 della ROM con ISR a $8500

            ORG     $8500                   ;ricevi da tastiera
INT1        MOVE.L  A0,-(A7)                ;push di A0,A1,A2,D0,D1 in stack supervisor
            MOVE.L  A1,-(A7)
            MOVE.L  A2,-(A7)
            MOVE.L  D0,-(A7)
            MOVE.L  D1,-(A7)
            MOVEA.L #TERD,A0
            MOVEA.L #PIADB,A1
            MOVEA.L #PIACB,A2

INPUT       MOVE.B  (A0),D0                 ;acquisisci dato da terminal

*trasferisce il carattere letto alla PIA-A con handshaking

ciclo1      MOVE.B  (A2),D1         ;leggi stato/controllo di portoB
            ANDI.B  #$80,D1         ;test se IRQB1=1 (CB1 1->0)
            BEQ     ciclo1
            MOVE.B  D0,(A1)
            MOVE.B  #%11110100,(A2) ;CB2: 1->0
ciclo2      MOVE.B  (A2),D1
            ANDI.B  #$80,D1
            BEQ     ciclo2
            MOVE.B  #%11111100,(A2) ;CB2: 0->1

*fine trasferimento e handshaking

            CMP.B           #13,D0
            BNE             INPUT
            ORI.B           #$18,TERC       ;riabilita tastiera
            MOVE.L          (A7)+,D1        ;ripristino di D0,D1,A2,A1,A0
            MOVE.L          (A7)+,D0
            MOVE.L          (A7)+,A2
            MOVE.L          (A7)+,A1
            MOVE.L          (A7)+,A0
            RTE

*La pia-A ha ricevuto un carattere dalla pia-B partner, interrompe il processore checon la ISR riceve il
carattere e lo trasmette direttamente al proprio TERMINAL per la visualizzazione.ISR a $8700 associata
all' interrupt di liv. 3  #vect 27  mappato a $6C della ROM


            ORG $8700

INT3        MOVE.L  A1,-(A7)                ;salvataggio registri
```

```
        MOVE.L  A0,-(A7)
        MOVE.L  D0,-(A7)

        MOVEA.L  #TERD,A0             ;inizializzazione indirizzi device
        MOVEA.L  #PIADA,A1
        ANDI.B          #$FE,1(A1)    ;disabilita possibilita' interruzioni di IRQA
        MOVE.B  (A1),(A0)             ;acquisisce carattere e lo trasferisce a Terminal
        ORI.B   #$01,1(A1)            ;riabilita interruzioni

        MOVE.L  (A7)+,D0              ;ripristino registri
        MOVE.L  (A7)+,A0
        MOVE.L  (A7)+,A1
        RTE

END     BEGIN
```

Di seguito è riportato il listing prodotto dall'assemblatore

```
00008200                                20      ORG    $8200
00008200                                21
00008200  =00002004                     22  PIADA      EQU    $2004    ;indirizzo di PIA-A dato, usato
in input
00008200  =00002005                     23  PIACA      EQU    $2005    ;indirizzo di PIA-A
stato/controllo
00008200  =00002006                     24  PIADB      EQU    $2006    ;indirizzo di PIA-B dato, usato
in output
00008200  =00002007                     25  PIACB      EQU    $2007    ;indirizzo di PIA-B controllo
00008200                                26
00008200  =00002000                     27  TERD       EQU    $2000            ;indirizzo di TERMINAL
registro dato
00008200  =00002001                     28  TERC       EQU    $2001    ;indirizzo di TERMINAL registro
Stato-Controllo
00008200                                29
00008200                                30
00008200  4EB9 00008226                 31  BEGIN      JSR            DVAIN   ;inizializza PIA porto
A
00008206  4EB9 0000823E                 32      JSR            DVBOUT  ;inizializza PIA porto B
0000820C  4EB9 00008252                 33      JSR            DVTER   ;inizializza terminal
00008212  40C0                          34      MOVE.W  SR,D0   ;legge il registro di stato
00008214  0240 D8FF                     35      ANDI.W  #$D8FF,D0 ;maschera per reg stato (stato
utente, int abilitati)
00008218  0040 8000                     36      ORI.w   #$8000,D0 ;messa per provare il bit di trace
0000821C                                37
0000821C                                38
0000821C  46C0                          39      MOVE.W  D0,SR   ;pone liv int a 000
0000821E  D240                          40      add     d0,d1   ;messa per provare il bit    di trace
00008220                                41
00008220                                42
00008220  4EF9 00008220                 43  LOOP       JMP LOOP        ;ciclo caldo dove il processore
attende interrupt
00008226                                44
00008226                                45
00008226                                46  *
00008226  11FC 0000 2005                47  DVAIN      MOVE.B  #0,PIACA                ;seleziona il
registro direzione del porto A
0000822C  11FC 0000 2004                48      MOVE.B  #$00,PIADA      ;pone le linee di A a linee di
input
00008232  11FC 00E5 2005                49      MOVE.B  #%11100101,PIACA  ;
00008238  1038 2004                     50      MOVE.B  PIADA,D0                ;lettura fittizia per
inizializzare CA2 ad 1
0000823C                                51
0000823C  4E75                          52      RTS
0000823E                                53
0000823E  11FC 0000 2007                54  DVBOUT     MOVE.B  #0,PIACB                ;seleziona il
registro direzione di PIA porto B
00008244  11FC 00FF 2006                55      MOVE.B  #$FF,PIADB        ;pone le linee di PIA B a
linee di output
0000824A  11FC 00FC 2007                56      MOVE.B  #%11111100,PIACB
00008250  4E75                          57      RTS
00008252                                58
```

```
00008252  11FC 003F 2001                   59  DVTER      MOVE.B  #$3f,TERC      ;seleziona indirizzo
stato/controllo
00008258                                   60  *                          ;00111111 {int abilitato su
buff full, int ab. su enter,
00008258                                   61  *                          ;canc. video, clear buff
tastiera, abil. tastiera, abil. eco
00008258  4E75                             62      RTS
0000825A                                   63
0000825A                                   64
0000825A                                   65
0000825A                                   66
0000825A                                   67  *ISR per la gestione dato proveniente dalla tastiera di
TERMINAL e spedito, per tramite del
0000825A                                   68  * PIA porto B, all'altro sistema.
0000825A                                   69  *ISR associata all'interrupt di liv. 1, #vect 25 mappato a
$64 della ROM con ISR a $8500
0000825A                                   70
00008500                                   71      ORG     $8500          ricevi da tastiera
00008500  2F08                             72  INT1       MOVE.L  A0,-(A7)               push di
A0,A1,A2,D0,D1 in stack supervisor
00008502  2F09                             73      MOVE.L  A1,-(A7)
00008504  2F0A                             74      MOVE.L  A2,-(A7)
00008506  2F00                             75      MOVE.L  D0,-(A7)
00008508  2F01                             76      MOVE.L  D1,-(A7)
0000850A  207C 00002000                    77      MOVEA.L #TERD,A0
00008510  227C 00002006                    78      MOVEA.L #PIADB,A1
00008516  247C 00002007                    79      MOVEA.L #PIACB,A2
0000851C                                   80
0000851C  1010                             81  INPUT      MOVE.B  (A0),D0        acquisisci dato da
terminal
0000851E                                   82
0000851E                                   83  *trasferisci il carattere letto alla PIA-A con handshaking
0000851E  1212                             84  ciclo1     MOVE.B  (A2),D1        ;leggi stato/controllo
di portoB
00008520  0201 0080                        85      ANDI.B  #$80,D1        ;test se IRQB1=1 (CB1 1->0)
00008524  67F8                             86      BEQ     ciclo1
00008526  1280                             87      MOVE.B  D0,(A1)
00008528  14BC 00F4                        88      MOVE.B  #%11110100,(A2) ;CB2: 1->0
0000852C  1212                             89  ciclo2     MOVE.B  (A2),D1
0000852E  0201 0080                        90      ANDI.B  #$80,D1
00008532  67F8                             91      BEQ     ciclo2
00008534  14BC 00FC                        92      MOVE.B  #%11111100,(A2) ;CB2: 0->1
00008538                                   93  *fine trasferimento e handshaking
00008538                                   94
00008538  B03C 000D                        95      CMP.B          #13,D0
0000853C  66DE                             96      BNE            INPUT
0000853E  0038 0018 2001                   97      ORI.B   #$18,TERC      ;riabilita tastiera
00008544  221F                             98      MOVE.L  (A7)+,D1               ;ripristino di
D0,D1,A2,A1,A0
00008546  201F                             99      MOVE.L  (A7)+,D0
00008548  245F                            100      MOVE.L  (A7)+,A2
0000854A  225F                            101      MOVE.L         (A7)+,A1
0000854C  205F                            102      MOVE.L         (A7)+,A0
0000854E  4E73                            103      RTE
00008550                                  104
00008550                                  105  *La pia-A ha ricevuto un carattere dalla pia-B partner,
interrompe il processore che
00008550                                  106  *con la ISR riceve il carattere e lo trasmette
direttamente al proprio TERMINAL per la visualiz-
00008550                                  107  *zazione.
00008550                                  108  *ISR a $8700 associata all' interrupt di liv. 3  #vect 27
mappato a $6C della ROM
00008550                                  109
00008550                                  110
00008700                                  111      ORG $8700
00008700                                  112
00008700  2F09                            113  INT3       MOVE.L  A1,-(A7)               ;salvataggio
registri
00008702  2F08                            114      MOVE.L  A0,-(A7)
00008704  2F00                            115      MOVE.L  D0,-(A7)
00008706                                  116
00008706  207C 00002000                   117      MOVEA.L  #TERD,A0      ;inizializzazione indirizzi
device
```

```
0000870C  227C 00002004                   118     MOVEA.L  #PIADA,A1
00008712  0229 00FE 0001                  119     ANDI.B          #$FE,1(A1)      ;disabilita
possibilita' interruzioni di IRQA
00008718  1091                            120     MOVE.B  (A1),(A0)               ;acquisisce carattere e
lo trasferisce a Terminal
0000871A  0029 0001 0001                  121     ORI.B   #$01,1(A1)      ;riabilita interruzioni
00008720                                  122
00008720  201F                            123     MOVE.L  (A7)+,D0                ;ripristino registri
00008722  205F                            124     MOVE.L  (A7)+,A0
00008724  225F                            125     MOVE.L  (A7)+,A1
00008726  4E73                            126     RTE
00008728                                  127
00008728                                  128     END     BEGIN

No errors detected
No warnings generated
```

La figura successive mostra il codice eseguibile prodotto dall'assemblatore assoluto in formato s-file. Tale codice è pronto per essere caricato in memoria per l'esecuzione. Son rappresentate in grassetto le meta informazioni necessarie al caricatore per posizionare il codice agli indirizzi assoluti. Per la descrizione del formato s-file si veda l'appendice.

```
S004000020DB
S12582004EB9000082264EB90000823E4EB90000825240C00240D8FF0040800046C0D2404EF9CF
S1258222000822011FC0000200511FC0000200411FC00E52005103820044E7511FC00002007B7
S119824411FC00FF200611FC00FC20074E7511FC003F20014E75CB
S12585002F082F092F0A2F002F01207C00002000227C00002006247C0000200710101212020 1C0
S1258522008067F8128014BC00F412120201008067F814BC00FCB03C000D66DE0038001820017E
S10F8544221F201F245F225F205F4E7363
S12587002F092F082F00207C00002000227C00002004022900FE000110910029000100012 01F02
S1098722205F225F4E738C
S90382007A
```

# Parte IV Assembler MC68000

## 5 Linguaggio assembler

### 5.1 Generalità sui linguaggi assembler

Un linguaggio assemblativo (o assembler) è un linguaggio di basso livello in cui i codici operativi hanno una corrispondenza uno ad uno con i codici in linguaggio macchina del processore. L'assembler offre al programmatore tipi di dato primitivi analoghi a quelli supportati 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 "Download" ed eseguito dal processore (fig.1).

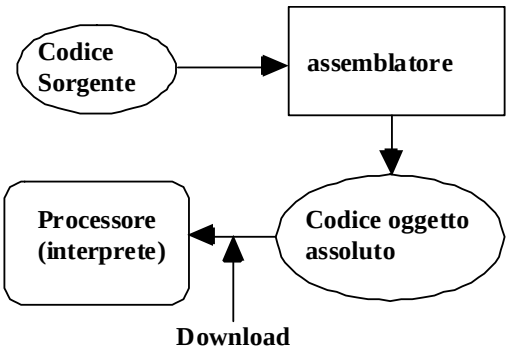



Fig.1 Ciclo di sviluppo con assembler assoluto

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 (ad es. la definizione di strutture dati e costanti, l'inizializzazione di un location counter, etc.).

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 ancora rilocabile, a meno di una costante di spiazzamento) che, 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, in ogni caso, ne restano mantenute sempre distinte le funzionalità.

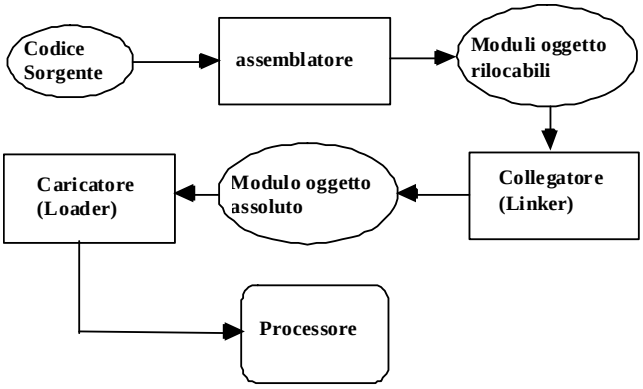



Fig.2 Ciclo di sviluppo con Assembler rilocabile

Esistono, inoltre, dei linguaggi assemblativi detti assembler strutturati o macro assembler che forniscono, oltre alle funzionalità presenti negli assembler tradizionali, istruzioni di controllo ad alto livello (con semantica dei costrutti if, while, repeat, for tipici dei linguaggi di programmazione quali C, Fortran, etc.), direttive per la creazione di strutture dati complesse e la loro inizializzazione, impiego di formati istruzione variabili e possibilità di definire ed usare macro istruzioni.

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

Un programma assembler può essere organizzato in aree o segmenti specializzati (classi di allocazione). 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 macro-assembler 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 il sistema di sviluppo (Hw e S.O base) MOTOROLA per il 68000 dispone di un macro-assembler che gestisce 16 moduli. Il Linker associato accorpa tali moduli al massimo in quattro classi di allocazione[1]. 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).

<inserire esempio dei listati dell'assembler-linker>

[1] 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.

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.

## 5.2 Programma Sorgente

E' 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 macro-assembler) anche su più linee.

## 5.3 Programmi assoluti e rilocabili

Un riferimento in memoria dati o programmi (indirizzo) è detto assoluto se esso viene definito a tempo di compilazione (o assemblaggio) relativo se definito in epoche 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'immagine. 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 indipendente 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 dai 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.

Nella scrittura di un programma assembler 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 corrispondenti a indirizzi fisici della 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 possibilmente 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 quasi concorrenza. 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. Cioè, 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" (position independent).

## 5.4 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".

## 5.5 Schema di rilocazione adottato da Assembler-Linker-loader della famiglia 68xxx

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.

I modi di indirizzamento supportati dal 6809 sono i seguenti:

-Immediato

-Relativo

-Diretto ed Esteso

-Indicizzato

## 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
```

# 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.

## 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 [2]::

```
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 posponendolo dopo il simbolo ".". Tale specificatore può assumere i valori {B=byte, W=word, L=long word, S=short}.

[2] 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.

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 **"*"**.

### 6.1.1 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.

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.

### 6.1.2 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.

### 6.1.3 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).

### 6.1.4 Costanti ASCII

Una costante ASCII consiste di una stringa di non più di 4 caratteri,codificati secondo il codice ASCII (si veda allegato I), racchiusi tra apici singoli (') che può essere memorizzata in una voce di 32 bit (1 byte/stringa). Un doppio apice (''), da non confondere con il simbolo di “, all' interno della stringa indica il singolo 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\| |

### 6.1.5 Modello di programmazione del processore

Il processore MC68000 ha un’architettura rivolta al programmatore composta da

- D0-D7 8 Registri dati
- A0-A7 8 Registri indirizzi
- A7,SP 1Stack pointer di sistema (si può indifferentemente utilizzare l'una o l'altra denominazione)
- USP 1 Stack pointer d'utente
- CCR 1 Registro Condition Code (parte bassa (bit 0-7) del registro di stato SR)
- SR 1 Registro di stato (modificabile nella sola parte alta se in stato utente)
- PC 1 Registro prossima istruzione, utilizzabile nei modi di indirizzamento relativi al PC.

## 6.2 Direttive di Assemblaggio

Sono di seguito illustrate le principali direttive d'assemblaggio organizzate per classi d'uso. Si ricorda che le direttive di assemblaggio non sono istruzioni per il processore (codici operativi) e sono processate dall'assemblatore a "tempo di compilazione". Esse non saranno mai processate a "tempo di esecuzione" in quanto non generano codici in linguaggio macchina.

### 6.2.1 Controllo delle operazioni di assemblaggio

#### 6.2.1.1 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.

#### 6.2.1.2 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).

### 6.2.2 Definizione simboli

#### 6.2.2.1 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.

#### 6.2.2.2 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.

#### 6.2.2.3 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.

### 6.2.3 Definizione dati-Allocazione memoria

#### 6.2.3.1 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
```

### 6.2.3.2 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                                18
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
```

### 6.2.3.3 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
```

## 6.3 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}

### 6.3.1.1 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].

<inserire file repertoriocodop.doc dopo averlo ristrutturato e formattato anche a partire dal file totcodici.doc>

## 6.4 Programmazione in assembler

In questo capitolo 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.

### 6.4.1 Esempi di traduzione in assembler di costrutti di controllo flusso ad alto livello

#### 6.4.1.1.1 Costrutto If

***if (C) then S1 else S2***

```
Bcc(NOT C) et1
S1
     BRA     et2
et1  S2
et2  ...
```

#### 6.4.1.1.2 Costrutto Repeat

***repeat S until C***

```
et1  S
     Bcc(NOT C)      et1
     ...
```

#### 6.4.1.1.3 Costrutto While

***while (C) do S***

```
BRA
et1  S
et2  Bcc(C)  et1
```

#### 6.4.1.1.4 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
et1  MOVE    D1,D2
     SUB     D0,D2
     BGT     et2
     S
     ADD     k,D0
     BRA     et1
et2  ...
```

#### 6.4.1.1.5 Costrutto case

```
case tag of
1:   S1;
2:   S2;
3:   S3;
4,5  S4;
end;
```

### 6.4.2 Esempio di sviluppo di un progamma assembler in ambiente ASIM

Il codice LINK and allocate consente di gestire il frame stack (detto anche activation record)che viene predisposto da un programma all'atto della chiamata a un sottoprogramma o funzione per gestire il passaggio dei parametri e il salvataggio del contesto operativo o stato dell'elaborazione. Il programma chiamante carica nel frame tutte le informazioni per gestire il protocollo di scambio dei parametri effettivi nei formali e salvataggio di variabili e registri di interesse).
Si riporta di seguito la descrizione del codice fornita nel manuale utente del processore Mc68K

**LINK - Link and allocate**

Operation: [SP] <-- [SP] - 4; [M([SP])]<-- [An];
[An] <--[SP]; [SP] <-- [SP] + d
Syntax: LINK An,#<displacement>
Sample syntax: LINK A6,#-12
Attributes: Size = word

**Description**
The contents of the specified address register are first pushedonto the stack. Then, the address register is loaded with theupdated stack pointer. Finally, the 16-bit sign-extendeddisplacement is added to the stack pointer. The contents of theaddress register occupy two words on the stack. A negativedisplacement must be used to allocate stack area to a procedure.At the end of a LINK instruction, the old value of address registerAn has been pushed on the stack and the new An is pointing atThe 68000's Instruction Set 29the base of the stack frame. The stack pointer itself has beenmoved up by d bytes and is pointing at the top of the stackframe. Address register An is called the frame pointer because it isused to reference data on the stack frame. By convention,programmers often use A6 as a frame pointer.Application: The LINK and UNLK pair are used to create local workspace on the top of a procedure.s stack.
Consider the code:

```
Subrtn LINK A6,#-12        Create a 12-byte workspace
MOVE D3,(-8,A6)            Access the stack frame via A6
UNLK A6                    Collapse the workspace
```

```
RTS                          Return from subroutine
Condition codes: X N Z V C
- - - - -
```

The LINK instruction does not affect the CCR.

**Esempio d'uso**

Tale programma è stato scritto in linguaggio C e compilato con l'opzione per produrre l'intermedio assembler. Il programma è riportato nel riquadro a fianco del listato assembler. Viene definita una funzione "Prodotto" che scambia i due parametri operandi della moltiplicazione e restituisce il loro prodotto nel nome della funzione.
Si noti come la traduzione automatica predisposta dal compilatore C fa ricorso a schemi di generazione di codici (ad es. il Movem.L D3, -(A7)) che in una traduzione manuale non si userebbero (si sarebbe utilizzato il semplice Move, in tal caso).

```
......
SECTION  9

PRODOTTO:
   LINK         A6,#0
   MOVEM.L      D3,-(A7)
   MOVE.L8      (A6),D0
   MOVE.L12     (A6),D1
   MULS         D1,D0
   MOVE.LD0,D3
   MOVE.LD3,D0

L_0:
   MOVEM.L      (A7)+,D3
   UNLK         A6
   RTS

MAIN:
   LINK         A6,#-54
   MOVEM.L      D3/D4/D5/D6/D7,-(A7)
   MOVE         #4,D4
   MOVE         #2,D3
   MOVE.LD3,-(A7)
   MOVE.LD4,-(A7)
   JSR          PRODOTTO
   ADD.W        #8,A7
   MOVE.LD0,-12(A6)
   MOVE.L-12(A6),D0
   ADD.W        D4,D0
   MULS         D3,D0
   MOVE.LD0,-32(A6)
   MOVE.L#128,D4
   MOVE         #10,D3
   BRA    L_1
L_2:
   MOVE.L-36(A6),-40(A6)
L_3:
   MOVE         #50,D7
   MOVE.LD7,D0
   JSR          _C_SWITCH

   DC.L         L_5,48
```

```
   DC.L        L_6,49
   DC.L        L_7,50
   DC.L        0,L_8

L_5:
   ADD.L       #1,D6
   BRA         L_4
L_6:
   ADD.L       #1,-20(A6)
   BRA         L_4
L_7:
   SUB.L       #1,-24(A6)
   BRA         L_4
L_8:
   SUB.L       #1,-28(A6)
   BRA         L_4
L_4:
L_9:
   CLR.L       -48(A6)
   TST.L       #0
   BEQ         L_10
   ADD.L       #1,D6
   BRA         L_9
L_10:
   MOVE        #0,D5
L_11:
   CMP.L       -52(A6),D5
   BGE         L_12
   ADD.L       #1,D5
   BRA         L_11
L_12:
L_1:
L_13:
   MOVEM.L     (A7)+,D3/D4/D5/D6/D7
   UNLK        A6
   RTS


   XDEF        PRODOTTO
   XDEF        MAIN
   XREF        _C_SWITCH
   END
```

# 7 Sottoprogrammi e scambio dei parametri

Un programma scritto in un qualsiasi linguaggio imperativo, sia esso ad alto che a basso livello, ricorre spesso, per problemi di strutturazione e economia di spazio memoria, a moduli detti sottoprogrammi che incapsulano al loro interno specifiche funzionalità che conviene mantenere isolate rispetto al flusso delle istruzioni di altri moduli. I programmatori affidano, secondo un protocollo convenzionato, a specifici sottoprogrammi, la realizzazione di funzioni specifiche. Il protocollo di interfaccia definisce la modalità con cui i dati di ingresso al sottoprogramma e i dati di uscita da esso restituiti al modulo chiamante, debbano essere scambiate. Tali dati detti parametri effettivi secondo l'ottica del chiamante e formali secondo quella del chiamato, possono essere scambiati fondamentalmente secondo le seguenti tre modalità:

per valore

per indirizzo

per nome

Nello scambio per valore

# Architettura e Ambiente Sviluppo ASIM

Il sistema di simulazione

La simulazione di una configurazione e dei relativi componenti si basa sul meccanismo primitivo offerto dal sistema operativo windows di scambio messaggi (sincrono e asincrono).

Una configurazione ASIM è trattata come una lista ordinata di oggetti (i singoli dispositivi componenti la configurazione) ciascuno appartenente ad una prefissata categoria (ad es. CPU, Device, MMU, Nodi). A ciascun oggetto è associato un descrittore, una struttura dati etichettata mediante un identificatore assoluto dell'oggetto stesso nella configurazione. I campi del descrittore sono comuni a ciascun oggetto e in essi sono codificati i parametri caratterizzanti l'oggetto stesso nella configurazione, la sua interconnessione con altri oggetti, lo spazio indirizzi occupato, le linee di interruzioni e l'eventuale gestore delle stesse.

Un dispositivo si presenta con tre interfacce caratterizzate ciascuna da una coppia di primitive:

## Device

## 7.1 Allegato I: Tabella ASCII

La tabella **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) che si pronuncia "aschii", è uno standard di codici a 7 bit proposto da ANSI nel 1963 e terminato nel 1968.

Lo standard di caratteri ASCII consiste di 128 numeri decimali (7 bit), da 0 a 127 assegnati a lettere, numeri, punteggiatura e caratteri speciali. L'insieme Esteso di caratteri ASCII consiste di altri 128 numeri decimali, da 128 a 255 (utilizzando dunque tutti gli 8 bit di un byte) e rappresentano caratteri addizionali e speciali, matematici, grafici e stranieri.

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

# Tabella dei caratteri ASCII

| Dec | Hex | Sym | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | NUL | 32 | 20 | | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | SOH | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | STX | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | ETX | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | EOT | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | ENQ | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | ACK | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | BEL | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | BS | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | TAB | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | LF | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | VT | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | FF | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | CR | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | SO | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | SI | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | DLE | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | DC1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | DC2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | DC3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | DC4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | NAK | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | SYN | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | ETB | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | CAN | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | EM | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | SUB | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | ESC | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | FS | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | GS | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | RS | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | US | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | |

# 8 Formato S-File Motorola

I file binari eseguibili sono prodotti a valle delle fasi di compilazione/assemblaggio/collegamento, come immagini di memoria assoluta. Dopo la fase di collegamento tutti i moduli rilocabili, o il modulo assoluto, che caratterizzano una specifica applicazione, confluiscono in un'immagina memoria assoluta, o ancora rilocabile, pronta per essere caricata in memoria centrale da un loader ai fini della sua successiva esecuzione.
Tali file eseguibili non sono visualizzabili a video direttamente in quanto i byte che li rappresentano non sono caratteri di stampa codificati, ad esempio, in ASCII. Per rappresentarli, si ricorre allora a rappresentazioni in cui le due quartine di un byte sono rappresentate con i caratteri
ASCII che ne caratterizzano il valore in esadecimale. Ad esempio il byte in codice macchina "11001010" è rappresentato dalle due quartine $B e $A sui due byte ASCII codificati come $42 e $41.
I file immagine sono caratterizzati dalle stringhe di byte con cui devono essere caricati determinate aree di memoria centrale. Il file, oltre ai dati immagine di memoria, porta con se dei metadati necessari al caricamento del file stesso in memoria e all'inizializzazione del registro prossima istruzione PC con il valore dell'indirizzo di entry point del programma rappresentato.
Sia l'Intel che Motorola, produttori dei primi e più diffusi sistemi a microprocessore hanno immesso sul mercato due loro formati (S.File per Motorola e ??-File per Intel, simili nella struttura se pur sintatticamente differenti).

L'S-File prevede l'esistenza di record di tipo differente etichettati come record di tipo S0, S1, S2, S3,S7,S8, e S9. Tali S-Record presentano tutti il seguente formato generale:

| Type[2] | Length[2] | addr[4,6,8] | data[0..2n] | check[2] |
|---|---|---|---|---|

Il campo addr e' di 2 byte se associato ad un record di tipo S1, 3 se ad uno di tipo S2 e 4 se ad uno di tipo S3.
L'S-File inizia sempre con un header record di tipo S0, il cui campo datacontiene un descrittore del programma od il nome del file;
I record di tipo S1, S2 ed S3 contengono blocchi di codici (o dati) da caricare in memoria a partire da un indirizzo espresso rispettivamente su 16, 24e 32 bit (2,3,4 byte);
I record di tipo S7,S8, ed S9 non contengo campo data e chiudono rispettivamente i blocchi data con indirizzo espresso rispettivamente su4, 3, 2 byte. Il campo address contiene in tal caso, se non vuoto, l'indirizzo (su 2,3o 4 byte) da caricare, come entry point, nel registro prossima istruzione PC
Ciascun blocco data deve, pertanto, cominciare e finire con una delle tre coppiedi record S1-S9, S2-S8, S3-S7.
Il byte di check e' calcolato sommando tutti i byte, escluso il byte di tipoe complementando ad 1 il risultato.

Una rappresentazione in XML degli S record è riportata di seguito:

<inserire XML e DTD>

Un esempio di S-Record:
<inserire un esempio di S-record di quelli riportati negli esempi rappresentati nel testo precedentemente>