Assembler

> ILC = L'**Instruction Location Counter** è una variabile che viene usata per tenere traccia (in fase di esecuzione) dell'indirizzo di una istruzione. Questo indirizzo viene usando durante l'assegnamento di un valore ad un simbolo nel campo *etichetta* della *istruzione*.

Passi assemblatore:

1) L'assemblatore costruisce due tabelle:
    1) **Tabella delle costanti** che contiene le **costanti globali** e i **nomi simbolici** dei sottoprogrammi
    2) **Tabella dei Simboli** che contiene i **valori dei simboli** Per determinare la lunghezza di una istruzione usa la **Tabella degli Opcode**
2) Generazione di un **codice oggetto** del programma sorgente, che consiste di:
    - sostituzione dei riferimenti simbolici **risolti**
    - si marcano le istruzioni che contengono riferimento simbolici **non risolti**
    - le istruzioni vengono tradotte in *codice oggetto*
    - rilevata presenza di eventuali errori
    - vengono emesse le informazioni necessarie al **Linker** per collegare i moduli >Tutto il *secondo passo* viene fatto tramite la **Tabella dei simboli**

Linker A questo punto, dopo i due passi dell'assemblatore, i due moduli oggetto generati devono essere fusi insieme. Il **LINKER** fa questo mediante una tabella.

Per avere un programma correttamente eseguibile i passi che un codice sorgente attraversa sono:

1) Compilatore
2) Assemblatore
3) Linker
4) Loader

Con riferimento all'interprete micro-programmato Mic-1, qualli delle seguenti affermazioni sono vere? [**F**] A. Durante l'esecuzione della micro-istruzione Main1 viene sempre richiesto il fetch dell'argomento dell'istruzione in esecuzione. [**V**] B. Durante l'esecuzione della micro-istruzione Main1 può essere richiesto il fetch del codice operativo della prossima micro-istruzione. [**F**] C. Durante l'esecuzione di una micro-istruzione che richieda una lettura dalla memoria (rd) nessun'altra lettura dalla memoria o scrittura verso la memoria

deve essere già in corso. [**F**] D. Il valore dei flag N e Z dell'ALU non sono alterati dalla micro-istruzione MDR=TOS.

Quali delle seguenti affermazioni relative a MIC-2 sono vere: [**V**] a) non è necessario che almeno uno degli operandi dell'ALU debba provenire dal registro H [**F**] b) il microprogramma del MIC-2 è identico al microprogramma del MIC-1 [**V**] c) l'IFU recupera dall'area dei metodi 4 byte alla volta. [**V**] d) l'offset di 2 byte presente nell'istruzione IJVM *"GOTO offset"* viene prelevato direttamente dal registro MBR2 [**F**] e) utilizza la tecnica del pipelining

Nell'ambito dell'architettura MIC-1 si descriva la relazione fra gli indirizzi nel *control store* delle due micro-istruzioni raggiungibili come destinazioni alternative di una istruzione di tipo jump (JAMZ e/o JAMZ uguali ad 1):

Quali delle seguenti affermazioni sono vere? [**V**] A. Il registro PC contiene l'indirizzo di 1 byte. [**F**] B. Il registro PC contiene 1 byte. [**V**] C. Il registro SP contiente l'indirizzo di 1 parola da 4 byte [**F**] D. Il registro SP contiene 1 byte

Descrivere la rappresentazione dei numeri relativi in complemento a due su *n* bit, specificando:

a) come si ottiene la codifica dei numeri positivi e negativi;

- positivo: basta ricopiarlo così com'è
- negativo: bisogna complementarlo a 1 (negarlo) e sommargli 1.

b) qual è l'intervallo di rappresentazione

- l'intervallo di rappresentazione su n bit è [$-2^{(n-1)}$] -> [2^(n-1)-1]

c) quali sono i principali vantaggi di questa rappresentazione dei numeri relativi

- il vantaggio principale di questa rappresentazione è che si ha una sola rappresentazione dello 0

JMPC è impostato ad 1 quando si vuole abilitare l'input dell'indirizzo da MBR, quindi solo nel caso dell'istruzione Main1.

Modifiche all'emulatore:

1) aggiungere una nuova macro-istruzione nel file di configurazione ijvm.conf
2) aggiungere in mic1ijvm.mal le micro-istruzioni relative all nuova macro-istruzione

3) Definire l'indirizzo nella control store relativo alla prima micro-istruzione. L'indirizzo dev'essere uguale all'OPCODE della nuova *macro-istruzione*.

Modi per riconoscere overflow in Complemento A 2:

1) Somma tra addendi dello stesso segno dà un risultato con segno diverso (es: ++=- o –=+)
2) Il riporto della colonna *n-1* alla colonna *n* ed il riporto dalla colonna *n* a quella oltra la cifra più significativa sono discordi.

Cache Il contenuto della cache e ciò che è in memoria (RAM) è identico. La RAM ha molte più pagine (4096) di memoria rispetto alla Cache (128 *frame*). Ogni pagina della RAM sta in un frame della Cache.

Principio di località: - **Spaziale**: con questo principio si assume che, durante l'esecuzione di un'istruzione di un programma da parte della CPU, le istruzioni seguenti si trovino in un'area contigua nella memoria principale. In questo modo la **cache** può memorizzare quell'area riducendo i tempi di esecuzione totale del programma. - **Temporale**: con questo principio si tiene in considerazione la frequenza con cui le parti di codice vengono eseguite e si assume che ci siano parti che vengono eseguite più spesso di altre. Dunque mantenendo una copia in memoria cache dei dati più richiesti durante l'esecuzione del programma, si sfrutta tale principio. - ###### Direct mapping: La memoria principale viene divisa in blocchi da 128 pagine, quindi 32 blocchi. Ogni blocco ha un campo, (che chiamiamo) **tag**, numerato da 0 a 31. La Cache usa il numero del tag per identificare il blocco e l'indice dei **suoi** *frame* indica quale *pagina* verrà caricata:

Direct mapping

La memoria principale viene divisa in blocchi da 128 pagine, quindi 32 blocchi. Ogni blocco ha un *tag* numerato da 0 a 31. La Cache usa il numero del tag per identificare il blocco e l'indice dei **suoi** *frame* indicano quale *pagina* verrà caricata: **TAG** 3, **INDICE** 2: la *pagina* caricata nel *frame* è la pagina 2 del blocco 3 -> *pagina* 386 In questo tipo di mapping **non ci sono algoritmi di *Sostituzione delle pagine***.

- **Associative mapping - Fully Associative Mapping**
- **Set-associative mapping**

BUS

Il **BUS** è un insieme di linee elettriche che collegano i moduli di un elaboratore. Affinché i *moduli* collegati siano in grado di comunicare è necessario che essi interagiscano con il *BUS* tramite regole ben precise. L'insieme delle regole viene definito come il ***protocollo del BUS***

Le **linee del BUS** possono essere:

- Linee di **DATI**: la sua larghezza determina il numero di bit che possono essere trasmessi insieme.
  - Es: con n bit possono indirizzare 2ˆn indirizzi
- Linee di **INDIRIZZO**: permettono di inviduare la **sorgente/destinazione** dei dati
- Linee di **CONTROLLO**: **controllano l'accesso** delle linee di dati e di indirizzo.

I **BUS** possono essere: - **SINCRONI**: essi **hanno** un **clock principale*** pilotato da un oscillatore. **Tutte** le attività richiedono un numero **intero** di questi cicli. - **ASINCRONI**: essi **NON** hanno un clock principale. I cicli hanno lunghezza variabile in base alle necessità e **non devono** essere uguali per tutti i dispositivi. - I dispositivi collegati possono essere: - **MASTER** - **SLAVE**

INVOKEVIRTUAL L'istruzione **INVOKEVIRTUAL** predisponde l'ambiente per il passaggio al sottoprogramma: - Crea l'area di attivazione del chiamato sul top dello stack - Inserisce in quest'area le informazioni necessarie per il ritorno - Aggiorna i registri - Lancia l 'esecuzione del sottoprogramma —-

Calcolo dell'indirizzo della prossima istruzione: Espressione Booleana: > F = (JAMZ AND Z) OR (JAMN AND N) OR NEXT_ADDREDSS[8] (il bit più significativo)

se JAMN = 1 -> MPC[PIÙ SIGNIFICATIVO] OR N se JAMZ = 1 -> MPC[PIÙ SIGNIFICATIVO] OR Z

> In ogni caso MPC conterrà uno dei due: NEXT_ADDRESS NEXT_ADDRESS[8] OR 1

se JAMC = 1 -> MBR OR NEXT_ADDRESS > Normalmente, in questa condizione, i bit di NEXT_ADDRESS sono tutti 0 e l'**opcode** dell'istruzione ISA (dopo fetch in MBR) determina la prossima micro-istruzione (**inizio** di un nuovo microprogramma).