1 Bootstrap

All'accensione della macchina, tutti i registri e la memoria si troveranno in uno *stato iniziale* (detto anche *stato di reset*). Nello stato iniziale usualmente tutti i registri hanno contenuto uguale a zero , pertanto anche il registro program counter conterrà il valore zero. La macchina può eseguire a partire da questa configurazione iniziale, andando ad accedere, decodificare ed eseguire l'istruzione memorizzata all'indirizzo zero. In questo indirizzo dovrà trovarsi la prima istruzione del *programma di inizializzazione* del sistema. Ma al momento dell'accensione la locazione di memoria a questo indirizzo conterrà un valore iniziale che non necessariamente è un'istruzione significativa.

Per ovviare a questo problema, una zona della memoria, a partire dall'indirizzo zero, viene implementata da una memoria a sola lettura (ROM - Read Only Memory) in cui è stato scritto, una volta per tutte al momento della costruzione della macchina, il programma di inizializzazione. Questo programma, spesso detto anche *monitor*, tipicamente farà partire una lettura da disco (hard disk o floppy disk) del codice del sistema operativo, che verrà scritto in una opportuna zona di memoria; alla fine della lettura verrà eseguito un salto alla prima istruzione di tale codice.

Questo processo di inizializzazione prende il nome di *bootstrap* o *start-up* del sistema.

2 Altri set di istruzioni

Il set di istruzioni presentato è molto limitato, anche se è già rappresentativo dei set di istruzioni che si trovano in macchine reali. Per avere una migliore idea di cosa si può incontrare in altri set di istruzioni, ricordiamo i seguenti punti:

- *formato variabile.* Nel nostro caso tutte le istruzioni sono rappresentate su una parola di memoria; spesso in un set di istruzioni troviamo istruzioni rappresentate su un numero di byte o parole diversi, a seconda del numero degli operandi e del loro modo di indirizzamento, al fine di rendere più compatto il codice, dal punto di vista della sua occupazione di memoria. Si parla in questo caso di *formato variabile*, perchè il formato delle istruzioni può essere diverso, ad esempio a seconda del loro codice operativo. La decodifica delle istruzioni viene resa più complessa dall'uso di un formato variabile perchè occorrerà prelevare dalla memoria e decodificare un numero variabile di segmenti di istruzione. Inoltre, anche la scrittura di programmi in linguaggio macchina risulta più complessa perchè, variando l'occupazione di memoria della singola istruzione, varia anche la distanza tra due istruzioni successive.

- *altre istruzioni aritmetiche.* Frequentemente si trovano altre istruzioni aritmetiche, come le istruzioni che eseguono spostamenti (*shift*) a sinistra o a destra, che possono corrispondere rispettivamente alla moltiplicazione o alla divisione per una potenza di due. Alcune macchine distinguono tra spostamenti logici (corrispondenti alla moltiplicazione e divisione di numeri unsigned), spostamenti aritmetici (corrispondenti alla moltiplicazione e divisione di numeri rappresentati in complemento a 2) e rotazioni (cioè spostamenti in cui l'ultimo bit che verrebbe perso viene riscritto nel primo bit). Nella classe delle istruzioni aritmetiche si fanno rientrare anche le classiche operazioni logiche (and, or , not).

- *virgola mobile.* Tra le altre istruzioni aritmetiche che si trovano spesso disponibili nei moderni processori troviamo quelle che operano su numeri rappresentati in virgola mobile. L'esecuzione di queste istruzioni viene spesso delegata ad un processore specializzato, detto *coprocessore matematico* o *FPU - Floating Point Unit.* Questa unità è (logicamente) esterna alla CPU: nei microprocessori delle ultime generazioni essa è comunque integrata sullo stesso chip del processore, mentre nei processori meno recenti era costituita da un chip separato.

- *istruzioni di controllo.* Abbiamo visto una forma limitata di istruzioni di controllo, quali il salto incondizionato e il salto condizionato per accumulatore maggiore-uguale a 0; è possibile trovare istruzioni di salto su condizioni diverse. Un caso particolare sono le istruzioni che permettono di verificare se nella precedente operazione numerica effettuata (ad es. somma) vi siano state condizioni di overflow, o comunque altre condizioni di errore che siano state memorizzate dalla CPU in appositi registri (normalmente registri di 1 bit detti *flag.*

- *istruzioni di I/O.* Le istruzioni di ingresso/uscita, considerate sono molto semplici. Spesso si trovano varie istruzioni specializzate per il colloquio con dispositivi esterni;

in altri casi, una zona della memoria viene considerata come appartenente ai dispositivi di ingresso/uscita (*memory-mapped I/O*) e leggere/scrivere parole di quella zona significa interagire direttamente con i dispositivi di I/O. Inoltre, la gestione dei dispositivi di I/O si effettua spesso tramite il meccanismo delle interruzioni (vedasi gli appunti su tale argomento).

- *Modi di indirizzamento* Il modo di indirizzamento specifica come ottenere l'operando a cui fa riferimento l'istruzione. Si noti che questo operando può essere, nel caso di istruzioni di controllo, un indirizzo. Vi sono quattro possibili modi di indirizzamento:

  Modo registro: l'operando dell'istruzione è contenuto nel registro specificato nel campo *SecondaParola*. Questo modo non può essere utilizzato nelle istruzioni di controllo.

  Modo registro differito: se l'operando richiesto dall'istruzione è un indirizzo, questo si trova nel registro specificato dal campo *SecondaParola*. Altrimenti, l'operando si trova nella parola di memoria il cui indirizzo è il contenuto del registro specificato dal campo *SecondaParola*.

  Modo immediato: indica che l'operando è direttamente il contenuto del campo *SecondaParola* dell'istruzione. Questo modo non può essere utilizzato per ottenere un indirizzo.

  Modo diretto: se l'operando richiesto dall'istruzione è un indirizzo, questo è specificato dal campo *SecondaParola*. Altrimenti, l'operando si trova nella parola di memoria il cui indirizzo è specificato dal campo *SecondaParola*.

- *indirizzamento al byte*. Nel nostro esempio gli indirizzi individuano direttamente delle locazioni di memoria; talvolta, si usano memorie che hanno locazioni di 8 bit; quindi l'indirizzamento è al byte, mentre gli operandi delle istruzioni e i registri sono parole di 16 bit, o anche doppie parole (32 bit). Quindi risulta più complessa la gestione degli indirizzi e risulta più difficile la scrittura di programmi in linguaggio macchina.

- *microcontrollori*. Una classe particolare di processori è quella detta dei "microcontrollori". Questi processori vengono utilizzati per controllare processi fisici, spesso con sarsa interazione con un utente umano: ne sono esempi la centralina elettronica di iniezione delle automobili, i computer che controllano lavatrici od altri elettrodomestici, o i controllori digitali che hanno spesso sostituito in molte applicazioni i contollori elttronici analogici.
  I microcontrollori, spesso caratterizzati da quantit di memoria RAM e ROM limitate e direttamente integrate sul chip del processore, contengono unità specializzate quali convertitori analogici/digitali e modulatori ad ampiezza di impulso, che consentono di acquisire direttamente segnali ananlogici proventienti da sensori di misura, e di fornire in uscita tensioni e frequenze proporzionali a valori contenuti in locazioni di memoira e registri. I microcontrollori presentano quindi una serie di istruzioni macchina atte a gestire tali unità. Usualmente il programma eseguito da un microcontrollore contenuto in una ROM (PROM o EPROM) e non fa riferimento ad unit

di memoria di massa (dischi o altro), in modo che venga direttamente eseguito ad ogni accensione della macchina.

3 Assiomi per la definizione dei tipi di dati astratti

La definizione di tipo di dato astratto come terna (S,F,C), cioè (insieme dei domini, insieme delle operazioni, insieme della costanti) richiede una definizione delle operazioni del tipo di dato astratto; tale definizione è data nel libro di testo riportando la sintassi delle operazioni, data in termini della loro funzionalità (ad es. $push : pila \times elemento \rightarrow pila$), e definendo la semantica delle operazioni a parole.
Un modo più rigoroso di definire la semantica delle operazioni è tramite la definizione di assiomi che definiscono le equivalenze tra diverse espressioni composte dalle operazioni definite (ad es. si può dire che: $push(pop(pila.vuota)) = pila.vuota$).

Di seguito diamo gli assiomi per i principali tipi studiati, seguendo la nomenclatura delle operazioni adottata nel testo.

Per ogni tipo, diamo anche una serie di condizioni di errore indicando quali operazioni ritornano valori erronei. Per essere rigorosi, dovremmo definire qual'è la semantica delle operazioni quando hanno come operando un valore erroneo, ma questo richiede un'approfondimento di alcune conoscenze di semantica algebrica che non possediamo. D'altra parte, la non segnalazione di condizioni di errore porterebbe a considerare, ad es. $pop(pila.vuota)$ come un valore legale del dominio $pila$. Un altro modo di trattare le condizioni di errore è quello di ritornare un valore convenzionale in casi erronei (ad es. $pop(pila.vuota) = pila.vuota$). Questa scelta può portare però a effetti laterali sulla semantica delle operazioni in condizioni non erronee (ad es. da questo assioma e da quello visto precedentemente sarebbe possibile derivare sia $push(pop(pila.vuota)) = pila.vuota$ che $push(pop(pila.vuota)) = push(pila.vuota)$, ottenendo perciò un sistema di assiomi *inconsistente*). Una rappresentazione concreta di un tipo di dato astratto dovrà di volta in volta trattare le condizioni di errore come meglio risulta da un punto di vista concreto (utilizzo di una variabile booleana di ritorno, stampa di un messaggio di errore, ecc..).

- INSIEME (pag. 95)
  $test.insieme.vuoto(insieme.vuoto) = true$
  $test.insieme.vuoto(inserisci(I, x)) = false$
  $test.appartenenza(insieme.vuoto) = false$
  $test.appartenenza(inserisci(I, x), x) = true$
  $test.appartenenza(cancella(I, x), x)) = false$
  $cancella(inserisci(a, x), x) = a$
  $inserisci(cancella(a, x), x) = a$

- MATRICE (pag. 103)
  $accedi(memorizza(m, i, x), i) = x$

- LISTA (pag. 112)
  $null(lista.vuota) = true$
  $null(cons(a, x)) = false$
  $car(cons(a, x)) = a$
  $cdr(cons(a, x)) = x$

$cons(car(x), cdr(x)) = x$

$car(lista.vuota) = error$
$cdr(lista.vuota) = error$

- PILA (pag. 147)
  $test.pila.vuota(pila.vuota) = true$
  $test.pila.vuota(push(p, x)) = false$
  $pop(push(p, x)) = p$
  $top(push(p, x)) = x$
  $push(pop(x), top(x)) = x$

  $top(pila.vuota) = error$
  $pop(pila.vuota) = error$

- CODA (pag. 151)
  $test.coda.vuota(coda.vuota) = true$
  $test.coda.vuota(in.coda(q, x)) = false$
  $out.coda(in.coda(coda.vuota, x)) = coda.vuota$
  $out.coda(in.coda(q, x)) = in.coda(out.coda(q), x)$, per $q \neq coda.vuota$
  $primo(in.coda(coda.vuota, x)) = x$
  $primo(in.coda(q, x)) = primo(q)$, per $q \neq coda.vuota$

  $primo(coda.vuota) = error$
  $out.coda(coda.vuota) = error$

- ALBERO BINARIO (pag. 154)
  $test.albero.vuoto(albero.vuoto) = true$
  $test.albero.vuoto(costruisci(x, a, y)) = false$
  $radice(costruisci(x, a, y)) = a$
  $sinistro(costruisci(x, a, y)) = x$
  $destro(costruisci(x, a, y)) = y$
  $costruisci(sinistro(x0, destro(x), radice(x)) = x$

  $radice(albero.vuoto) = error$
  $sinistro(albero.vuoto) = error$
  $destro(albero.vuoto) = error$