Architettura degli Elaboratori
11 - Il Livello ISA

Ugo Dal Lago

Dipartimento di Scienze dell'Informazione
Università degli Studi di Bologna

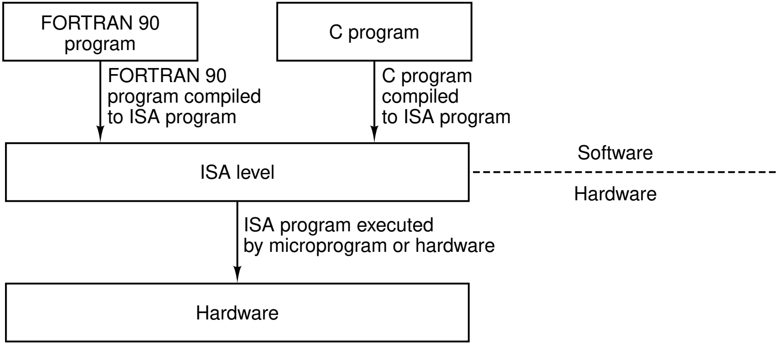
Anno Accademico 2006/2007

Sommario

Il Livello ISA

- Il livello ISA si trova tra il livello della microarchitettura e quello del sistema operativo.
- La rilevanza del livello ISA dipende, tra l'altro, dal fatto di costituire **l'interfaccia** tra hardware e software.
  - Eseguire programmi in linguaggi di alto livello direttamente con hardware non sarebbe una buona idea.
  - I programmi scritti in linguaggi di alto livello vengono in genere **tradotti** (tramite programmi detti **compilatori**) in programmi del livello ISA.
- In linea di principio, la progettazione del livello ISA coinvolge sia i progettisti hardware che i progettisti dei compilatori.
  - In pratica, molto spesso occorre rendere il nuovo ISA **retrocompatibile** con gli ISA precedenti.
  - Un buon ISA dovrebbe definire un insieme di istruzioni che possa essere **implementato efficientemente** con tecnologie presenti e future.
  - D'altra parte, occorre garantire la possibilità di **compilare efficientemente** codice di alto livello nel linguaggio che si sta progettando.

Il Livello ISA come Interfaccia tra Compilatori e Hardware



Il Livello ISA come Interfaccia Tra Sistema e Mondo Esterno

- Il livello ISA è il primo livello pubblico.
- Molto raramente i programmatori utilizzano **direttamente** il linguaggio del livello ISA.
  - D'altra parte, chi scrive compilatori, deve conoscere bene il livello ISA con cui avrà a che fare.
- In alcuni casi, il livello ISA è specificato attraverso un documento formale di definizione, contente **sezioni normative** e **sezioni informative**.
  - Questo è il caso dell'architettura SPARC, na non dell'architettura Pentium 4.
- La maggior parte dei processori è dotata di due modalità di esecuzione: la **modalità kernel** e la **modalità utente**.
  - In modalità kernel sono eseguibili tutte le istruzioni.
  - In modalità utente non è possibile invece eseguire alcune istruzioni.

Modelli di Memoria

- La memoria è suddivisa in **celle indirizzabili**, la cui dimensione è attualmente pari a 8 bit (ovvero a 1 byte).
- I byte vengono raggruppati in parole di 4 o 8 byte. Per manipolare parole esistono istruzioni apposite, che spesso richiedono che le parole siano **allineate**.
  - Il requisito dell'allineamento semplifica il progetto del livello logico-digitale.
- Lo **spazio degli indirizzi** è l'insieme di tutti gli indirizzi utilizzabili nei riferimenti alla memoria centrale.
- In genere, esiste un **unico** spazio degli indirizzi in ogni architettura. In alternativa, possiamo avere spazi degli indirizzi distinti per le **istruzioni** e per i **dati**.
- Possono esistere diverse **semantiche della memoria**, visto che in presenza di microarchitetture complicate, la memoria può non avere il comportamento atteso.
  - Da un lato, tutte le richieste di accesso alla memoria possono essere **serializzate**, dando luogo alla semantica più semplice.
  - D'altra parte, la semantica può **non dare alcuna garanzia**.

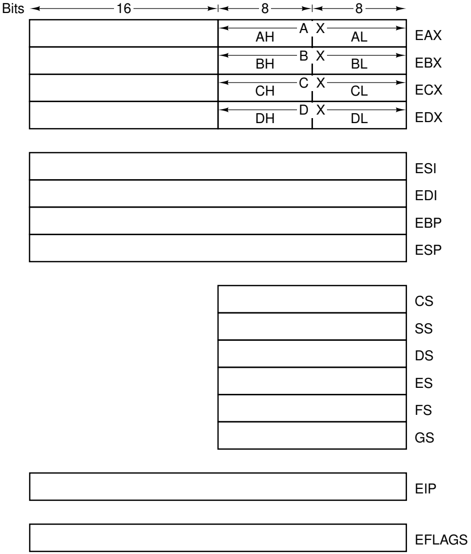
Registri

- Alcuni dei registri del livello sottostante il livello ISA sono visibili anche al livello ISA e possono quindi essere utilizzati nelle istruzioni.
- I registri del livello ISA sono suddivisibili in due categorie:
  - **Registri specializzati**, che comprendono il puntatore allo stack, il program counter e altri registri dedicati a funzioni specifiche.
  - **Registri di uso generale**, che contengono i risultati parziali del calcolo e le variabili locali più importanti.
- Alcuni tra i registri di uso generale possono avere ruoli particolari nell'esecuzione delle istruzioni.
  - Per esempio, un registro di uso generale potrebbe essere l'operando implicito di un'istruzione aritmetica.
- Esistono un certo numero di registri specializzati accessibili solo in modalità kernel.
- Un esempio è il cosiddetto **registro flag**, detto anche `PSW` (program status word), che contiene alcuni bit tra i quali i cosiddetti **codici di condizione**.

Panoramica del Livello ISA del Pentium 4

- L'ISA utilizzato nel processure Pentium 4 (e in tutti i suoi antenati a partire dall'80386) è chiamato **IA-32**.
- L'IA-32 mantiene un completo supporto per l'esecuzione di programmia scritti nel linguaggio ISA del processore 8088.
- Il processore Pentium 4 è dotato di tre modalità operative:
  - La **modalità reale**, in cui tutte le caratteristiche introdotte dopo l'8088 sono disattivate.
  - La **modalità virutale 8086**, che consente di eseguire vecchi programmi dell'8088 in modo protetto. da
  - La **modalità protetta**, in cui tutte le funzionalità del Pentium 4 sono disponibili. Esistono 4 livelli di privilegi, dal livello 0 (utilizzabile solo dal sistema operativo) al livello 3 (utilizzato dai programmi utente)
- Lo spazio degli indirizzi è suddiviso in 2^{14} segmenti di 2^{32} byte ciascuno.
  - La maggior parte dei sistemi operativi supporta un solo segmento.

I Registri Principali del Pentium 4



Panoramica del Livello ISA dell'UltraSPARC III

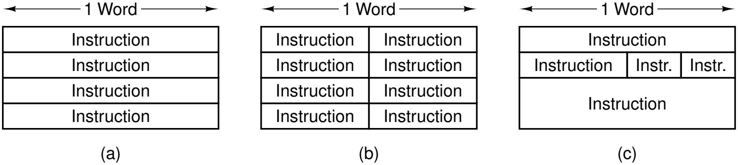
- In origine l'architettura SPARC era un'architettura a 32 bit, ma dalla versione 9 si è passati ad un'architettura a 64 bit.
- Lo spazio degli indirizzi è costituito da 2^{64} byte.
- L'UltraSPARC III ha due gruppi di registri:
  - 32 registri di uso generale da 64 bit, chiamati `R0`,`R1`,...,`R31`.
  - 32 in virgola mobile.
- I registri di uso generale possono assumere nomi diversi, che li fanno diventare registri specializzati.
  - Per esempio i registri `R8`,...,`R13` diventano i registri da `O0`,...,`O5` e vengono utilizzati per il passaggio dei parametri.
- L'architettura UltraSPARC III è un'architettura **load/store**. In altre parole, nello spirito RISC, le uniche due istruzioni che accedono alla memoria sono `LOAD` e `STORE`.

Tipi di Dati

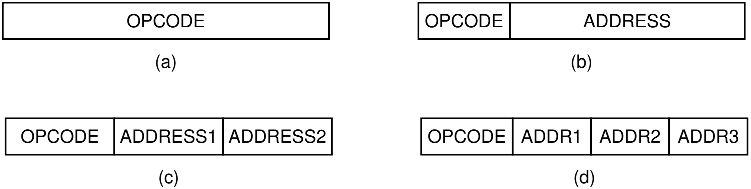
- Le operazioni di manipolazione dei dati messe a disposizione da un instruction-set assumono che le sequenze di bit contenute nei registri o in memoria codifichino i dati secondo schemi ben precisi.
  - In questo senso parliamo di **tipo di dato**.
- Esistono prima di tutto i **tipi di dati numerici**, che servono a rappresentare numeri.
  - Possiamo avere interi **con segno** oppure **senza segno**.
  - Possiamo avere anche numeri in **virgola mobile**.
  - Talvolta si utilizzano schemi di codifica inusuali come il cosiddetto schema **BCD**.
- Esistono poi i **tipi di dati non numerici**, che servono a rappresentare caratteri, sequenze di caratteri (che vengono dette **stringhe**) o valori booleani.
  - Si possono per esempio rappresentare 32 valori booleani tramite 4 byte. Parliamo in tal caso di **bit map**.
  - Un altro tipo di dato non numerico è il **puntatore**. In altre parole, si interpreta la sequenza di bit come un indirizzo.

Formati d'Istruzione

- Un'istruzione consiste in un **opcode** (o **codice operativo**), eventualmente corredato da **altre informazioni**, che servono a determinare quale siano i dati coinvolti nell'operazione da eseguire.
- Le istruzioni sono rappresentate tramite sequenze di bit.
  - In alcune macchine, tutte le istruzioni hanno la stessa lunghezza.
  - In altre, possiamo avere istruzioni di lunghezza diversa.
- In generale, la lunghezza di un'istruzione può essere maggiore, uguale o minore della dimensione di una parola.



Alcuni Formati d'Istruzione



Criteri Progettuali per i Formati d'Istruzione

- Prima di tutto, occorre tenere presente che **istruzioni corte sono preferibili ad istruzioni lunghe**.
  - Raddoppiare la dimensione delle istruzioni implica raddoppiare la dimensione dei programmi!
  - D'altra parte, istruzioni più corte potrebbero risultare più difficili da decodificare o da sovrapporre.
  - La larghezza di banda della memoria (e della cache) sono spesso la variabile critica.
- Poi, il formato delle istruzioni deve essere progettato in modo da prevedere **un numero di bit sufficiente** a tutte le operazioni necessarie, anche tenendo conto di possibili sviluppi futuri.
  - Se le operazioni sono n, avremo bisogno di codici operativi lunghi almeno $\lceil \log_2(n) \rceil$.
- Occorre infine determinare in modo opportuno il **numero di bit da destinare alle informazioni aggiuntive** (come ad esempio agli indirizzi degli operandi).
  - Una migliore **risoluzione di memoria** si paga al prezzo di indirizzi più lunghi (e quindi di istruzioni più lunghe).

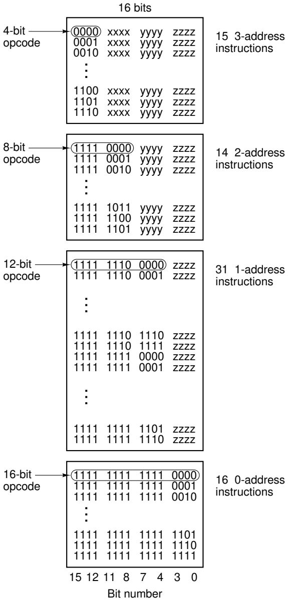
Codice Operativo Espandibile

- Fissata una lunghezza m per le istruzioni, possiamo decidere di attribuire n bit all'opcode e k bit agli indirizzi (in modo tale che $m = n + k$).
  - Diminuendo n, diminuiamo il numero di istruzioni rappresentabili, mentre diminuendo k diminuiamo la risoluzione di memoria.
- Illustriamo ora il concetto di **codice operativo espandibile** tramite un semplice esempio:
  - Supponiamo di lavorare con istruzioni a 16 bit, suddivisi in 4 bit per l'opcode, e tre indirizzi da 4 bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode				Address 1				Address 2				Address 3			

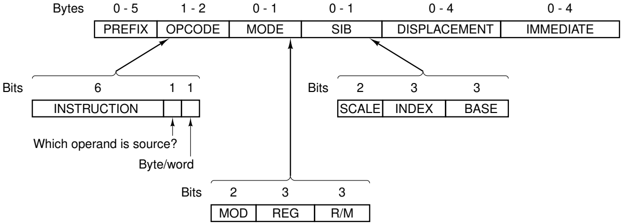
  - In questo modo, potremmo avere solo $16 = 2^4$ opcode diversi.
  - Supponiamo, però, di avere bisogno di 15 istruzioni con tre operandi, 14 con due operandi, 31 con un operando e 16 senza alcun operando.
  - Possiamo evitare di aumentare la lunghezza delle istruzioni riservando ai codici operativi un numero di bit variabile.

Un Opcode Espandibile



Formati delle Istruzioni di Pentium 4

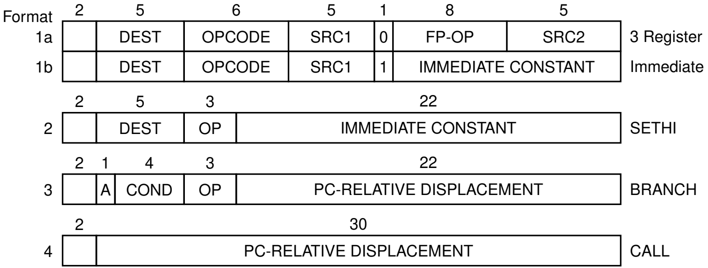
- Le istruzioni del Pentium 4 hanno un formato molto complesso e irregolare:



- Vi sono **poche regole generali**. Per esempio, nelle istruzioni con due operandi, al più uno degli operandi si trova in memoria.
- I singoli bit degli opcode danno **poche informazioni** riguardo la struttura delle istruzione: occorre decodificare completamente l'opcode per capire quale sia la lunghezza delle istruzioni.
  - Ciò rende **più complicato** il precaricamento dell'istruzione successiva.

Formati delle Istruzioni di UltraSPARC III

- L'ISA dell'UltraSparc III contiene esclusivamente istruzioni di 32 bit.
- Lo SPARC originale disponeva di soli cinque formati di istruzione:



- Oggi ne possiamo contare almeno 31.
- I primi 2 bit concorrono a determinare il formato dell'istruzione.

Modalità di Indirizzamento

- **Indirizzamento Immediato**.
  - Nella parte di istruzione riservata ad un operando, troviamo l'**operando stesso** e non il suo indirizzo (o qualunque altra informazione).
  - Per caricare una costante in un registro, potremmo per esempio usare un'istruzione come `MOV R1 4`
- **Indirizzamento Diretto**.
  - L'operando è specificato tramite il suo **indirizzo completo**.
  - Serve solo ad accedere a variabili globali.
- **Indirizzamento a Registro**.
  - Concettualmente identico all'indirizzamento diretto, ma si specifica un registro invece che un indirizzo di memoria.
  - L'esecuzione dell'istruzione sarà molto veloce.
- **Indirizzamento a Registro Indiretto**
  - L'operando proviene o è diretto alla memoria, ma il suo indirizzo è contenuto in un registro piuttosto che nell'istruzione.
  - In questo caso, l'indirizzo prende il nome di **puntatore**.
  - Una modalità di indirizzamento del genere risulta utile per accedere a parole di memoria con indirizzi consecutivi.

Modalità di Indirizzamento

- **Indirizzamento Indicizzato**.
  - L'indirizzo dell'operando si ottiene dall'indirizzo contenuto in un registro, sommando ad esso uno **spiazzamento**, contenuto invece nell'istruzione.
  - Ad esempio, le istruzioni `ISTORE` e `ILOAD` di IJVM utilizzano l'indirizzamento indicizzato.
  - Un altro esempio è l'istruzione `MOV R4,4(R2)`
- **Indirizzamento Indicizzato Esteso**.
  - L'indirizzo dell'operando si ottiene dalla somma degli indirizzi contenuti in due registri e sommando ad essa uno **spiazzamento**, contenuto nell'istruzione.
- **Indirizzamento a Stack**.
  - Si utilizza un modello di memoria basato sullo stack e l'indirizzo degli operandi è implicitamente determinato.
  - L'indirizzamento a stack risulta molto utile nella valutazione di espressioni in **notazione polacca inversa**.

Ortogonalità dei Codici Operativi e Delle Modalità d'Indirizzamento

- Le istruzioni e le modalità di indirizzamento dovrebbero presentare una **struttura regolare**, in cui tutti gli opcode consentano tutte le modalità di indirizzamento.
  - Si parla in questo senso di **ortogonalità** dei codici operativi e delle modalità di indirizzamento.
- Un esempio di progetto per i formati di istruzione è il seguente:

<table>
<tr><td>Bits</td><td>8</td><td>1</td><td>5</td><td>5</td><td>5</td><td>8</td></tr>
<tr><td>1</td><td>OPCODE</td><td>0</td><td>DEST</td><td>SRC1</td><td>SRC2</td><td></td></tr>
<tr><td>2</td><td>OPCODE</td><td>1</td><td>DEST</td><td>SRC1</td><td colspan="2">OFFSET</td></tr>
<tr><td>3</td><td>OPCODE</td><td colspan="5">OFFSET</td></tr>
</table>

- Nel caso in cui gli operandi siano due, si potrebbe utilizzare il formato seguente:

<table>
<tr><td>Bits</td><td>8</td><td>3</td><td>5</td><td>4</td><td>3</td><td>5</td><td>4</td></tr>
<tr><td></td><td>OPCODE</td><td>MODE</td><td>REG</td><td>OFFSET</td><td>MODE</td><td>REG</td><td>OFFSET</td></tr>
<tr><td></td><td colspan="7">(Optional 32-bit direct address or offset)</td></tr>
<tr><td></td><td colspan="7">(Optional 32-bit direct address or offset)</td></tr>
</table>

Modalità di Indirizzamento del Pentium 4 e di UltraSparc III

- L'indirizzamento, nel Pentium 4, è assai **complesso**.
  - Possono essere utilizzate le modalità immediata, diretta, a registro, a registro indiretto, più una modalità d'indirizzamento speciale per i vettori.
  - **Non tutte** le modalità si applicano a tutte le istruzioni.
  - Il byte `MODE` controlla la modalità di indirizzamento.
  - Uno dei due operandi è sempre un registro.
- In UltraSPARC, invece, l'indirizzamento è relativamente **semplice**:
  - Tutte le istruzioni si avvalgono dell'indirizzamento immediato o a registro.
  - L'unica eccezione a tal principio è rappresentata dalle istruzioni che fanno riferimento alla memoria (`LOAD`, `STORE` e un'istruzione per la sincronizzazione in ambito multiprocessore).

Istruzioni di Trasferimento Dati

- Una delle operazioni fondamentali è la copia di una sequenza di bit da una locazione ad un'altra, che chiamiamo **trasferimento**.
- I dati possono provenire da **due sorgenti** (memoria o registri) ed essere destinati a **due locazioni** (memoria o registri), vi sono quattro tipi di trasferimenti possibili.
  - Alcune architetture dispongono di quattro istruzioni diverse.
  - Altre hanno una sola istruzione.
  - Altre utilizzano `LOAD` per trasferire i dati dalla memoria ai registri, `STORE` per trasferire i dati dai registri alla memoria e `MOVE` per i trasferimenti tra i registri.
- Le istruzioni di trasferimento dati devono specificare in qualche modo la quantità di dati da trasferire.
  - Normalmente, si trasferisce una sequenza di bit lunga quanto una parola di memoria.
  - Lunghezze diverse vengono gestite ai livelli superiori, anche se qualche architettura CISC mette a disposizione apposite istruzioni al livello ISA.

Operazioni Binarie

- Le **operazioni binarie** producono un risultato dalla combinazione di due operandi.
- Tra le operazioni binarie vi sono sempre operazioni aritmetiche, come l'**addizione**, la **sottrazione**, la **moltiplicazione** e la **divisione**.
- Tra le operazioni binarie troviamo anche le operazioni logiche, che corrispondono ai connettivi dell'algebra booleana.
- L'operazione AND è utile per **estrarre** una sottosequenza di bit da una parola.
  - Supponiamo di vole estrarre il terzo bit dalla parola a 32 bit.
  - Possiamo procedere nel modo seguente:

10110111	10111100	11011011	10001011	A
00000000	11111111	00000000	00000000	B
00000000	10111100	00000000	00000000	A AND B

- L'operazione OR, invece, può essere utilizzata per **impacchettare** bit in una parola.
- Molti architetture supportano anche operazioni aritmetichein virgola mobile.

Operazioni Unarie

- Le **operazioni unarie** producono un risultato a partire da un singolo operando.
- Le operazioni per lo **scorrimento** o la **rotazione** (verso destra o verso sinistra) sono operazioni unarie spesso presenti in più varianti.
  - Nelle operazioni di scorrimento i bit che fuoriescono dalla parola sono persi.
  - Nelle operazioni di rotazione, i bit che fuoriescono da un lato riappaiono dall'altro.
- Gli scorrimenti verso destra sono spesso utilizzati in associazione con l'estensione del segno.
- Applicazione dello scorrimento sono la moltiplicazioni e la divisione per **potenze di** 2.
- Le operazioni di scorrimento sono anche utili per l'impacchettamento e lo spacchettamento di sequenze di bit.
- Alcune operazioni binarie sono così frequentemente invocate con un certo valore per uno degli operandi che a volte gli ISA dispongono di istruzioni unarie per svolgerle più velocemente.
  - L'istruzione `INC` è un caso particolare dell'istruzione `ADD`.

Confronti e Salti Condizionati

- È essenziale che vi siano istruzioni capaci di alterare la sequenza di esecuzione delle istruzioni in base al verificarsi di certe condizioni.
- Molto spesso, esiste un'istruzione di salto che esamina una condizione e:
  - Se la condizione **è verificata**, effettua il salto.
  - Se la condizione **non è verificata**, passa all'istruzione successiva.
- Tra le condizioni che vengono utilizzate più frequentemente nei salti incondizionati, vi è il valore di un certo bit interno alla macchina (per esempio uno dei bit del registro `PSW`).
  - Per esempio, molte macchine tengono traccia in `PSW` del verificarsi di una condizione di overflow.
- A volte, occorre eseguire un salto solo se il confronto tra due operandi ha un certo esito.
  - In alcuni casi, il confronto viene eseguito **contestualmente al salto** e quindi avremo un'istruzione con tre operandi.
  - In altri casi, esiste un'**apposita istruzione** di confronto che modifica alcuni bit interni alla macchina (per esempio qualche bit del registro `PSW`).

Invocazione di Procedura e Ritorno da Procedura

- Utilizzeremo **procedura**, **subroutine** e **metodo** come sinonimi.
- Esiste sempre almeno un'istruzione per l'**invocazione** di una procedura, che trasferisce il controllo alla procedura.
  - La successiva istruzione sarà la prima istruzione del corpo della procedura.
  - Se nell'esecuzione della procedura si incontra un'istruzione di **ritorno** da procedura, la successiva istruzione sarà quella che segue l'istruzione di invocazione.
  - Occorre quindi che al momento della chiamata si salvi l'indirizzo dell'istruzione successiva alla chiamata, che viene detto **indirizzo di ritorno**.
- L'indirizzo di ritorno può essere salvato in aree diverse:
  - In una **locazione di memoria fissa**.
  - In **una locazione di memoria relativa alla procedura**. Sarebbe però impossibile la **ricorsione**.
  - In un **registro** in modo che la procedura stessa possa poi salvarlo.
  - Nello **stack**, in modo che al momento del ritorno possa essere recuperato.

Istruzioni di Ciclo

- Spesso occorre eseguire una sequenza di istruzioni un numero prefissato n di volte.
  - Si parla in questo caso di un **ciclo di istruzioni**.
  - Molte architetture mettono a disposizione apposite istruzioni per costruire cicli.
- Esistono sostanzialmente due tecniche per costruire i cicli utilizzando le istruzioni di salto:
  - Ciclo **con verifica della condizione in testa**.
  - Ciclo **con verifica della condizione in coda**.
- Il primo dei due schemi è più efficiente del secondo (nel senso che necessita di meno istruzioni al livello ISA).
- Nel caso di cicli con verifica della condizione in coda, il corpo del ciclo viene eseguito almeno una volta, anche se $n = 0$.

Due Tipologie di Cicli

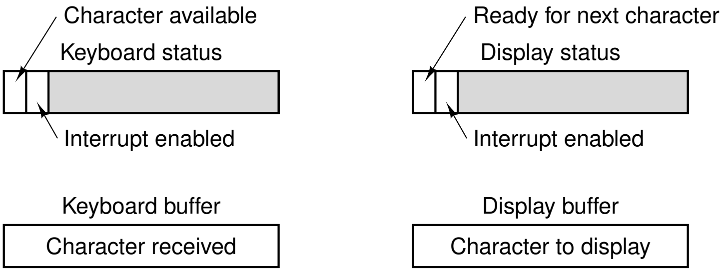
```
      i=0;
L1:   PrimaIstruzione
      ⋮
      UltimaIstruzione
      i=i+1;
      if (i<n) goto L1;
```

```
      i=1;
L1:   if (i>n) goto L2;
      PrimaIstruzione
      ⋮
      UltimaIstruzione
      i=i+1;
      goto L1;
L2:
```

I/O

- Al livello ISA occorre anche fare in modo che i programmi comunichino con i **dispositivi di I/O**.
- Ciò è vero in particolare in modalità kernel.
- Esistono sostanzialmente tre schemi diversi per l'I/O, relativamente al livello ISA.
- Prima di tutto, possiamo avere il cosiddetto **I/O programmato con attesa attiva**.
  - Una volta iniziata l'operazione di I/O, la CPU aspetta che l'operazione abbia fine, controllando il contenuto di alcuni registri.
  - La CPU passa gran parte del suo tempo in **attesa attiva**, e questo è proprio il principale problema di questo schema.
- Possiamo poi avere **I/O interrupt driven**.
  - Alla fine di ogni operazione di I/O il dispositivo genera un interrupt.
  - La CPU può nel frattempo elaborare altre istruzioni.

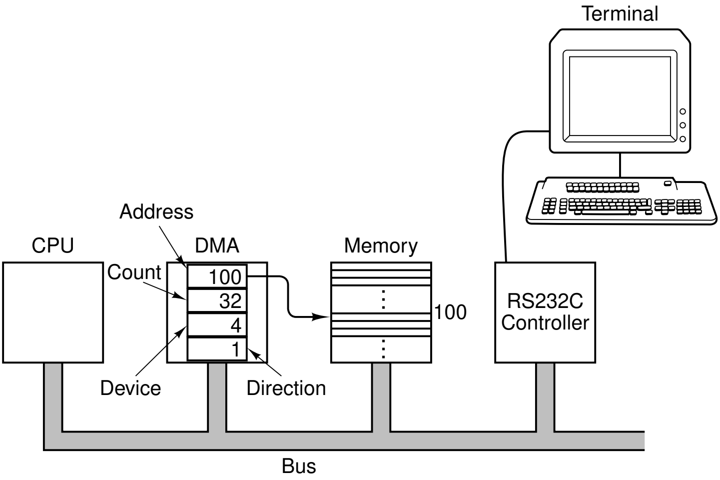
Registri dei Dispositivi



I/O

- Esiste poi la possibilità di affidarsi al **DMA**.
- I dati vengono trasferiti direttamente tra il dispositivo e la memoria centrale senza l'intervento diretto della CPU (se non all'inizio e alla fine del trasferimento).
- Esiste un chip specifico (chiamato **chip DMA**) che fa le veci della CPU e che dispone di un certo numero di registri accessibili direttamente dalla CPU stessa.
  - Vi sono almeno quattro registri interni al chip DMA corrispondenti ad altrettante informazioni: il **primo indirizzo di memoria**, la **lunghezza del blocco** da trasferire, il **codice del dispositivo di I/O** e il **tipo di operazione** (lettura o scrittura).
- I dati vengono trasferiti in blocco.
- Anche in questo caso alla fine di un trasferimento, viene generato un interrupt (questa volta dal chip DMA).
- Il chip DMA ha priorità sulla CPU nell'accesso al bus e può quindi sottrarre preziosi cicli di bus alla CPU.
  - Si parla in tal caso di **appropriazione di cicli** o **cycle stealing**.

Chip DMA in un Sistema di Calcolo

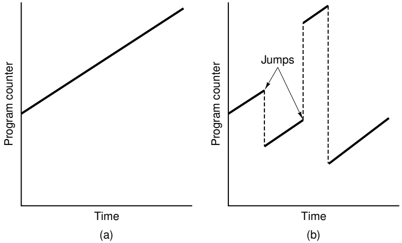


Istruzioni di Pentium 4 e UltraSPARC III

- Il Pentium 4 è una classica macchina CISC a 32 bit e a due indirizzi.
  - Le modalità di indirizzamento sono **molteplici** e molto irregolari.
  - Molte istruzioni permettono di accedere alla memoria.
  - Il più importante principio ispiratore dell'architettura è la retrocompatibilità.
- L'UltraSPARC III è invece un progetto RISC a 64 bit.
  - Le modalità di indirizzamento sono **poche**.
  - Soltanto tre istruzioni permettono di accedere alla memoria.
  - Molte macchine RISC contemporanee tendono a somigliare molto all'architettura SPARC.
- Per i dettagli sui due insiemi di istruzioni, si faccia riferimento al libro di testo.

Controllo del Flusso

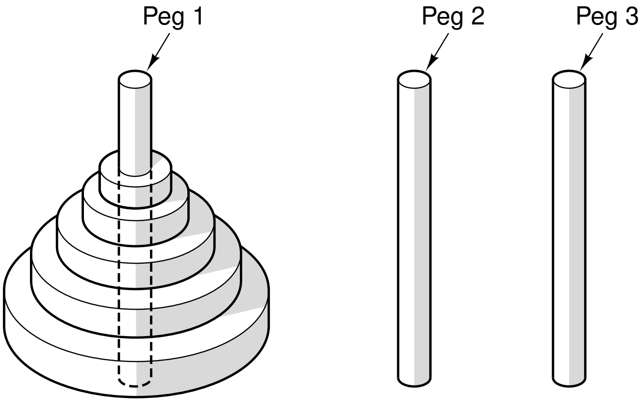
- Le istruzioni vengono eseguite di norma **nello stesso ordine** con cui si susseguono in memoria.
- Le uniche istruzioni che modificano questa situazione sono le seguenti:
  - I **salti incondizionati** e i **salti condizionati**.
  - Le **chiamate di procedura** e i **ritorni da procedura**.
  - Le **coroutine**.
  - Le **trap**.
  - Gli **interrupt**.
- In scenari diversi, il contenuto del registro `PC` evolve in modo diverso:



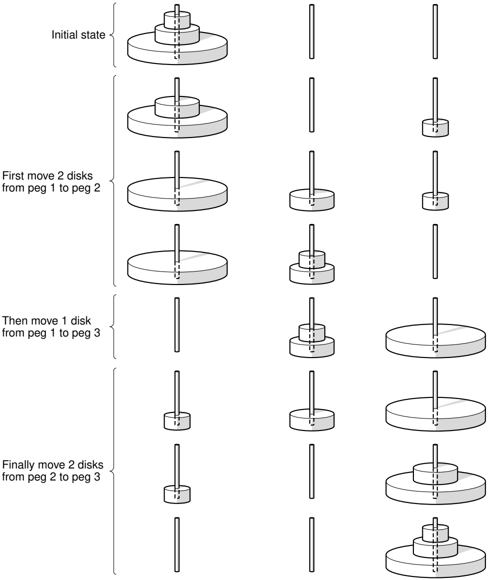
Procedure

- Le procedure sono la tecnica per la strutturazione dei programmi di gran lunga più importante.
- Le istruzioni di chiamata e di ritorno di procedura possono essere viste come salti incondizionati.
- Da un altro punto di vista, però, il corpo di ogni procedura può essere visto come un'istruzione singola.
- Le **procedure ricorsive** sono un tipo di procedure particolarmente interessanti. Le illustreremo tramite una soluzione ricorsiva al famoso problema della **torre di Hanoi**.
- La soluzione che consideremo è la seguente (supponendo n il numero di dischi da trasferire):
  1. Prima di tutto si spostano $n-1$ dischi dal primo piolo al secondo piolo.
  2. Poi si sposta il disco più grande dal primo piolo al terzo piolo.
  3. Poi si spostano $n-1$ dischi dal secondo al terzo piolo.

Configurazione Iniziale della Torre di Hanoi



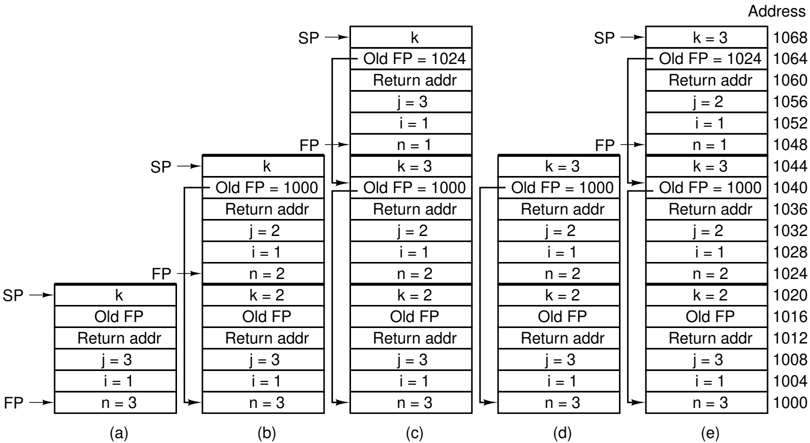
Soluzione con Tre Dischi



Procedura per la Soluzione del Problema della Torre di Hanoi

```
public void towers(int n,int i,int j)
{
int k;
if(n==1)
  System.out.println("Sposta un disco da "+i+" a "+j);
else
  {
  k=6-i-j;
  towers(n-1,i,k);
  towers(1,i,j);
  towers(n-1,k,j);
  }
}
```

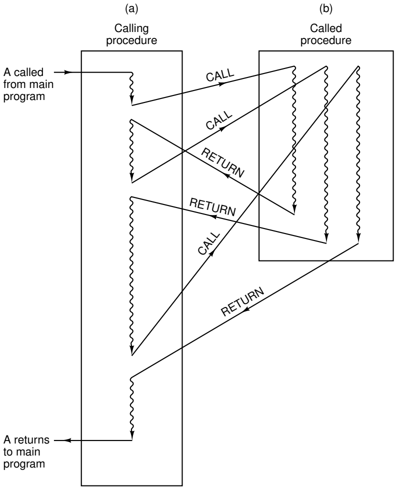
Stack Durante l'Esecuzione di `towers`



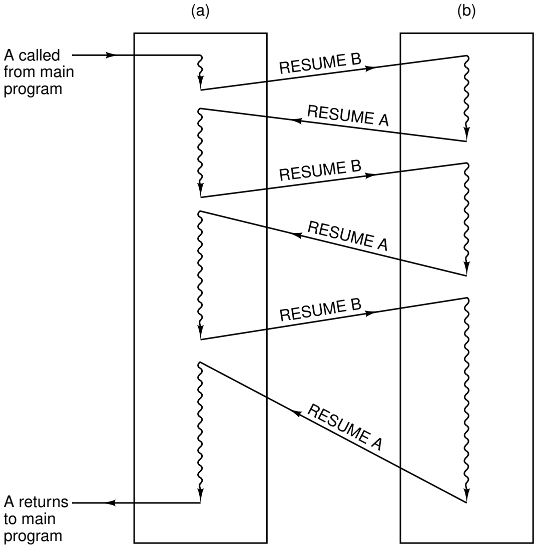
Coroutine

- Nelle **procedure** il passaggio del controllo tra la procedura chiamante e la procedura chiamata è **asimmetrico**.
  - Esiste da una parte l'istruzione di **chiamata** (o **invocazione**) di procedura e dall'altra l'istruzione di **ritorno** da procedura.
  - L'invocazione di procedura trasferisce il controllo alla **prima** istruzione del corpo della procedura chiamata.
  - Il ritorno trasferisce il controllo all'istruzione (della procedura chiamante) che **segue** l'istruzione di invocazione che aveva precedentemente attivato la procedura chiamata.
- Nelle **coroutine**, invece, le procedure coinvolte si scambiano il controllo in modo **simmetrico**.
  - Vi è un'**unica** istruzione di invocazione, che trasferisce il controllo alla prima istruzione non eseguita tra quelle della coroutine chiamata.
  - Questo schema risulta utile per simulare l'elaborazione parallela su una singola CPU.

Esecuzione di una Procedura



Esecuzione di una Coroutine



Trap

- Una **trap** è una chiamata di procedura automatica effettuata quando si verificano condizioni rilevanti, ma di occorrenza relativamente rara.
  - Esempi di condizioni che possono causare trap sono gli overflow, gli underflow, le violazioni di protezione, gli opcode non definiti, etc.
- Se si verifica una delle condizioni che causano una trap, il controllo è trasferito ad una procedura detta **gestore di trap**, che svolge le azioni appropriate al caso.
  - Per esempio, il gestore di trap potrebbe generare un messaggio di errore, o interrompere il programma attualmente in esecuzione.
- Un metodo alternativo all'utilizzo delle trap consiste nel tener traccia nel `PSW` delle condizioni di cui sopra.
  - Con quest'altra tecnica, però, il programma del livello ISA deve periodicamente controllare che le condizioni non siano verificate.
- Le trap possono essere realizzate al livello della microarchitettura oppure direttamente dall'hardware.

Interrupt

- Abbiamo già incontrato gli **interrupt** (o **interruzioni**) in altre parti del corso. È venuto il momento di parlarne più in dettaglio.
- Come le trap, gli interrupt interrompono il programma in esecuzione e trasferiscono il controllo a un **gestore** deputato a svolgere le azioni appropriate.
  - Una volta che il gestore ha svolto il suo compito, restituisce il controllo al programma interrotto.
- Mentre le trap sono **sincrone**, gli interrupt sono **asincroni**.
  - È sempre il programma ISA a causare una trap (per esempio eseguendo istruzioni illegali).
  - Le interruzioni, invece, si verificano in corrispondenza di eventi (quali la fine di un'operazione di I/O) il verificarsi dei quali non dipende dal programma attualmente in esecuzione.
- Al verificarsi di un'interruzione fanno seguito una serie di azioni hardware e software.
- Un gestore di interrupt si dice **trasparente** se è progettato in modo da far riprendere l'esecuzione esattamente dallo stato che aveva prima dell'invocazione dell'interrupt.

Interrupt

- Nei sistemi di calcolo moderni, vi sono più dispositivi di I/O.
- Esiste una probabilità non nulla che un secondo dispositivo di I/O voglia generare un interrupt proprio mentre una routine di interrupt è in esecuzione.
- Da una parte, si può fare in modo che ogni routine di interrupt **disabiliti** qualunque altro interrupt. In questo modo, però, ai dispositivi sono richieste attese anche lunghe.
- Uno soluzione migliore consiste nell'assegnare una **priorità** ad ogni dispositivo di I/O: alta per i dispositivi molto critici e bassa per quelli meno critici.
  - Durante l'esecuzione di una routine di priorità n, ogni tentativo di interruzione di dispositivi di priorità inferiore viene ignorato, mentre gli interrupt provenienti da dispositivi a priorità maggiore sono trattati senza attesa.
  - È consigliabile che tutti gli interrupt siano trasparenti.
- A partire dall'8088, i processori Intel sono dotate di due livelli di priorità: **mascherabile** e **non mascherabile**.
  - Gli interrupt **non mascherabili** servono a segnalare eventi quasi catastrofici.

Sequenza di Interrupt