Un'ISA indica un insieme di istruzioni macchina per una determinata architettura in un elaboratore.
E' come il computer viene visto dal programmatore o dal compilatore.

Si possono dividere in base al tipo di memoria utilizzata:

- **Stack**
- **Accumulatore**
- **Registers:**
  - **Register-memory**
  - **Register-register**
  - **Memory-memory**

Endianess

Little: Il byte con l'indirizzo più basso viene messo a destra(LSB).
Big: Il byte con l'indirizzo più basso viene messo a sinistra(MSB).

Memory Alignment

Si tratta dell'accesso *allineato* alla memoria.
Dal momento che un elaboratore salva i dati in *word*(32 bit ad esempio), se accedo un'area di memoria che non è un multiplo di 32 bit, potrebbero esserci problemi.

- **Aligned access:** Permettere soltanto un accesso *allineato* è una limitazione
- **Misaligned access:** Gli accessi non *allineati* introducono un *overhead* di hardware e performance maggiore.

Addressing Modes

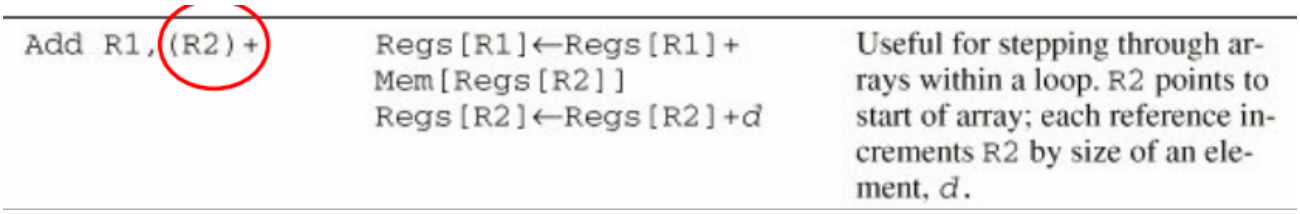
Sono le varie modalità di indirizzamento. Si dividono il base al tipo di dato che si accede:

- **Register mode:** Quando il valore specificato è un registro. *Add R4,R3*
- **Immediate mode:** Quando il valore è una costante. *Add R4,#3*
- **Displacement mode:** Si accede alla memoria tramite un offset applicato ad un valore di un registro.
  Add R4,100(R1) che equivale a *Reg[R4] = Reg[R4] + Mem[100+Reg[\R1]]* . Usato per accedere le variabili locali.
- **Register Deferred/Indirect mode:** Come per la **displacement mode** ma senza offset.
  Add R4,100(R1) che equivale a *Reg[R4] = Reg[R4] + Mem[Reg[\R1]]*

- **Indexed Mode:** Come per la **displacement mode** ma si usa come offset un altro registro. Usato per accedere gli array, dove il primo registro è la base dell'array e il secondo l'indice.
  Add R4,(R1+R2) che equivale a *Reg[R4] = Reg[R4] + Mem[Reg[R1]+Reg[\R2]]*
- **Direct/Absolute mode:** Si accede direttamente alla memoria tramite indirizzo assolute. Non viene quasi mai utilizzata questa modalità.
  Add R1,(1001)
- **Memory Indirect mode:** Si accede tramite l'indirizzo puntato dal contenuto di un registro(come se fosse un puntatore in **C**).

Add R1,@(R3)	Regs[R1]←Regs[R1]+ Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields *p.

- **post autoincrement mode:** Dopo aver eseguito l'istruzione incrementa il valore del registro. Utile per ciclare in un array.

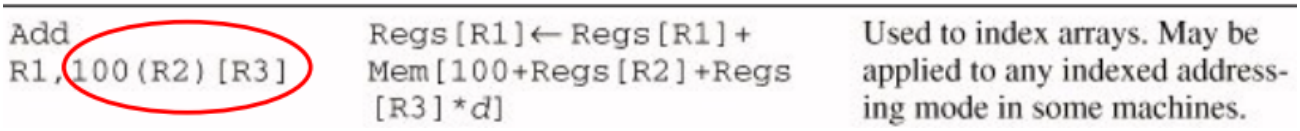


Add R1,(R2)+	Regs[R1]←Regs[R1]+ Mem[Regs[R2]] Regs[R2]←Regs[R2]+*d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*.

- **pre autodecrement mode:** Come il **post autoincrement.** Si può usare come push/pop per implementare uno stack.

Add R1,-(R2)	Regs[R2]←Regs[R2]-*d* Regs[R1]←Regs[R1]+ Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/ pop to implement a stack.

- **Scaled mode:** Usato per indicizzare gli array.



Add R1,100(R2)[R3]	Regs[R1]← Regs[R1]+ Mem[100+Regs[R2]+Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some machines.

d è la dimensione della word.

Destination address

Nelle istruzioni di **jump**, bisogna specificare anche l'indirizzo di destinazione, ovvero l'indirizzo dove risiede la prossima istruzione da eseguire.
Potremmo mettere direttamente l'indirizzo dell'istruzione, tuttavia nella maggior parte dei casi, la prossima istruzione è abbastanza vicina a quella precedente(basti pensare un *if*).
Dunque possiamo usare un **displacement**(o offset).
In questo modo il codice diventa anche **position-indipendent.**

Register indirect jump

Permette di saltare ad un indirizzo specificato dal contenuto di un determinato registro. Ad esempio: **salta** a **r5**.

Evaluating branch conditions

Condition Code: Vengono settati dei bit speciali durante le operazioni della **ALU**. Usato da ARM e 80x86.
Condition register: vengono testati due registri a scelta tramite un confronto. Usato da Alpha e VAX.
Compare and branch: il confronto fa parte dell'istruzione. Usato da MIPS e VAX.

Procedures

Bisogna salvare determinate informazioni:

- l'indirizzo di ritorno
- registri acceduti:
  - quelli usati dal **chiamante(caller)**
  - quelli usati dal **chiamato(callee)**

Instruction Set Encoding

Ci sono vari modi:

- **Variable:** Ogni istruzione ha una lunghezza variabile. Usato da **80x86** e **VAX.** Il codice è più **piccolo** tuttavia ha una performance peggiore, dal momento che ogni istruzione ha una lunghezza diversa e non si può ottimizzare ciò(ogni volta devo capire quanto è lunga).
- **Fixed:** Ogni istruzione ha una lunghezza fissa. Usato da ARM, PowerPC, Sparc etc. Ovviamente il codice è più grande ma ha performance migliori.
- **Ibride**

Register Allocation

Bisogna avere il giusto numero di registri.
Questo problema si basa su *graph coloring* e si risolve in maniera ottimale con un numero di registri > 16

Soluzioni per chi scrive compilatori

Rendere più **veloce** il caso più **comune** e quello più raro corretto.
Rendere le **quantità note** come **costanti.**