Il livello architettura e set di istruzioni

Daniele Paolo Scarpazza
Dipartimento di Elettronica e Informazione
Politecnico di Milano

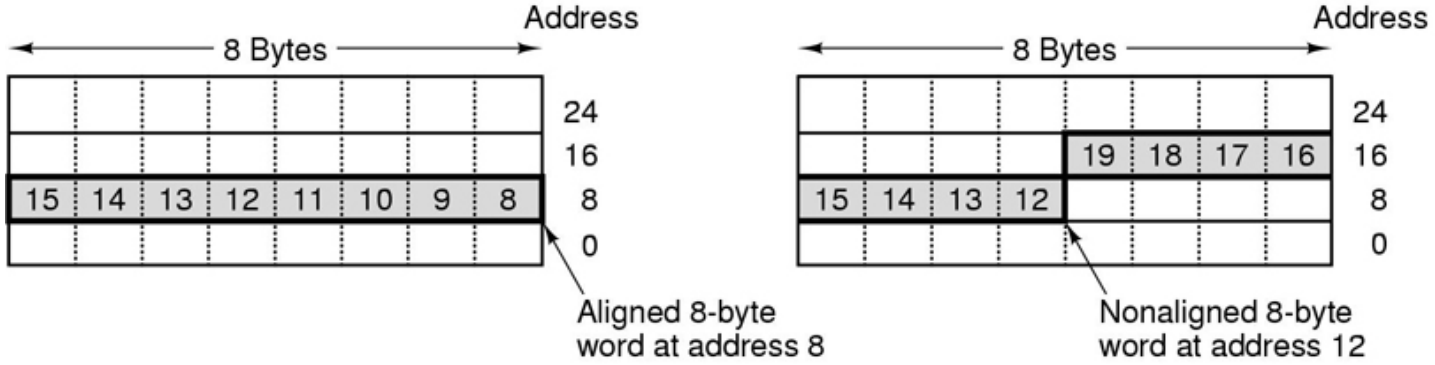
7 Giugno 2004

Il livello ISA

- Definisce le informazioni necessarie per
  - programmare una architettura in linguaggio macchina (= costruire un compilatore);
    - modello di memoria;
    - registri disponibili e loro dimensioni e usi specifici;
    - tipi di dati disponibili;
    - istruzioni disponibili;
  - non sono rilevanti a questo livello:
    - informazioni sull' architettura interna;
    - microprogrammazione
    - pipeline e stadi;
    - ...

Modello di memoria

- Memoria suddivisa in celle con indirizzi consecutivi;
  - solitamente celle da 8 bit, (architetture del passato 1-60 bit);
  - parola: gruppo di byte processabili simultaneamente; 32 bit = parole di 4 byte;
  - allineamento; una lettura disallineata richiede due accessi alla RAM: letture disallineate possono essere vietate o eseguite con costosa logica supplementare;



- Spazi di indirizzamento, scelte possibili:
  - uno solo, lineare, 0-2^{32} (scelta più diffusa)
  - due spazi distinti per dati e istruzioni; (RAM indirizzabile x2, impossibile corrompere le istruzioni);

Registri

- Tutte le architetture hanno registri visibili a livello ISA (molti registri della microarchitettura non sono visibili);
- Ve ne sono di due tipi:
  - ad uso generale (variabili locali, risultati intermedi delle operazioni)
  - per usi specifici (*program counter*, *stack pointer*, ...)
- I registri per uso generale possono essere:
  - equivalenti l'uno all'altro e completamente intercambiabili (programmatore e compilatore possono scegliere i registri a piacere);
  - differenziati per ruolo: (e.g. EAX accumulatore, ECX contatore)
- Registri speciali (e.g. FLAGS o PSW) contenenti:
  - informazioni hardware a disposizione del solo S.O. (modo utente/kernel, abilitazione interrupt, meccanismi di debugging);
  - codici di condizione (zero, negativo, overflow, parità, riporto) ad uso delle istruzioni di salto;

Istruzioni

- L'*instruction set* è forse la caratteristica che differenzia più fortemente una architettura da un'altra;
- Il set di istruzioni di una architettura definisce ciò che si può/non si può fare con quella macchina, e con quante istruzioni si codifica un dato algoritmo;
- Istruzioni tipiche:
  - istruzioni per lo spostamento dei dati fra registri (MOVE);
  - istruzioni per lo spostamento dei dati da e verso memoria (un qualche tipo di LOAD o STORE);
  - istruzioni aritmetico/logiche (ADD, SUB, MUL, DIV, AND, OR, NOT e loro varianti);
  - istruzioni di salto condizionato e incondizionato (JUMP o BRANC e loro varianti condizionate);

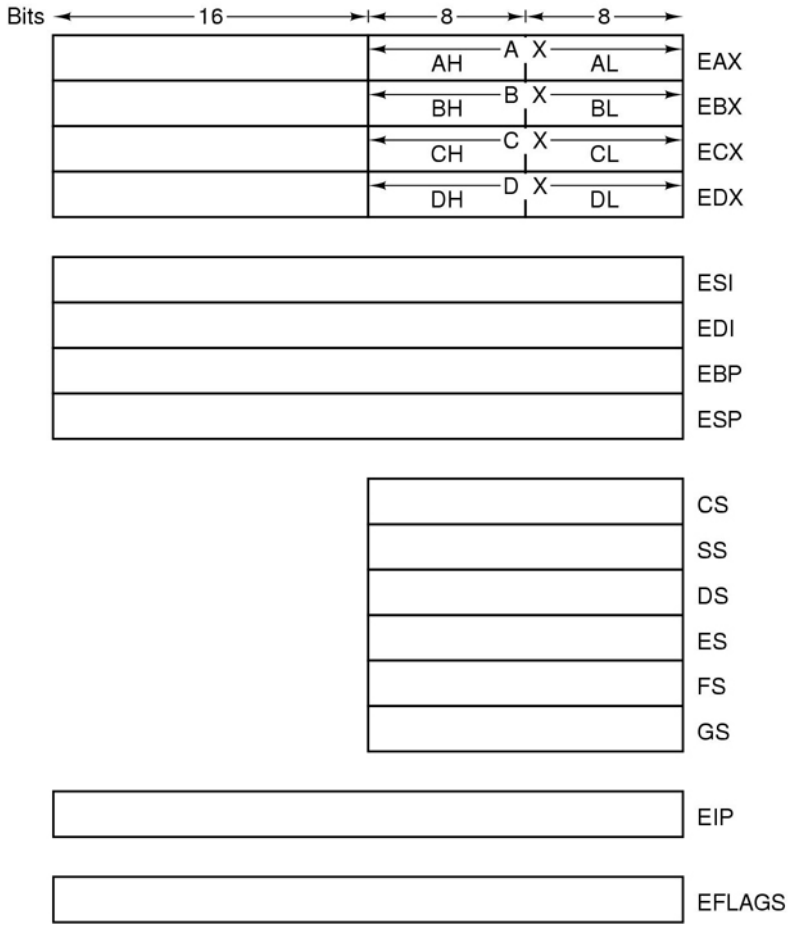
Esempio: architettura IA-32

- Riferimento:
  IA-32 Intel Architecture Software Developer's Manual, Volume 1,
  disponibile in PDF al sito http://developer.intel.com
- Nasce nel 1985 con il processore Intel 80386; arriva fino ad oggi comprendendo i processori: 486, Pentium, la famiglia P6 (Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon), Pentium 4, il Pentium Xeon e Pentium M;
- Tutti i processori IA-32 sono ancora in grado di eseguire il codice dei processori a 16 bit della famiglia precedente (8088 e 8086, giugno 1979);

Caratteristiche dell' IA-32

- Tre modi operativi: (differenze: cfr. Tanenbaum)
  - *real-address mode*: la macchina si comporta come un 8086, tutto quello che c'è di nuovo è disabilitato; è la modalità predefinita all'accensione;
  - *protected mode*: modo nativo del processore, con tutte le funzionalità attive. Fra le varie funzionalità, capacità di eseguire codice nativo 8086 in un ambiente protetto multi-tasking; impropriamente detto *virtual 8086-mode;*
  - *system management mode:* modo speciale per la gestione di energia e sicurezza
- Quattro livelli di privilegio: 0,1,2,3
  - 0 è la modalità con accesso completo alla macchina, riservata al kernel del sistema operativo;
  - 3 è la modalità utente, con limitazioni su istruzioni e segmenti indirizzabili, per garantire l'integrità del sistema;
  - 1,2 solitamente non sono usate;
- Spazio di indirizzamento:
  - 16k segmenti da 2^{32} byte ciascuno,
  - solitamente solo uno usato => spazio di indirizzamento lineare 2^{32};
  - parole memorizzate in formato *little-endian* (LSbyte per primo)

Registri



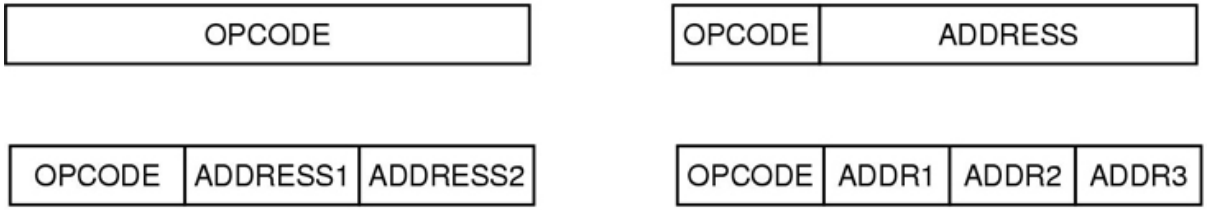
- EAX, EBX, ECX, EDX:
  - generali, ma con ruoli diversi:
    - EAX accumulatore aritmetico;
    - EBX registro "base"; puntatori;
    - ECX contatore
    - EDX ausiliario per MUL/DIV
  - ognuno ha versioni a 16 bit (AX, BX, CX, DX) risalenti all'8086;
  - a loro volta divise in parti alta e bassa (AH, AL, BH, BL, ...)
- ESI e EDI: sorgente e destinazione per le operazioni su stringhe;
- ESP: stack pointer
- EBP: puntatore alla base del frame
- CS, SS, ... GS: selettori di segmento;
- EIP: program counter;
- EFLAGS: registro di stato;

Tipi di dati

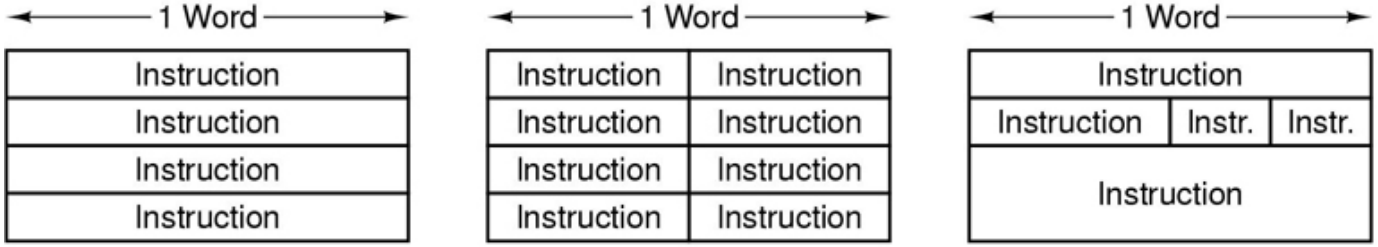
- Ciascuna istruzione di una architettura opera aspettandosi dati di un tipo ben definito e rispettandone le caratteristiche (e.g. bit di segno per gli interi, campi dei floating-point);
- Le operazioni sui tipi non supportati devono essere svolte *in software* appoggiandosi sulle istruzioni per i tipi supportati;
- Tipi di dati più comuni:
  - numerici:
    - interi (segnati e non segnati) da 8, 16, 32, 64, ... bit;
    - floating-point da 32 bit (float), 64 (double), 80, 96, 128 bit;
  - non-numerici;
    - caratteri ASCII (7bit), ASCII estesi (8) e UNICODE (16);
    - stringhe di tipo ASCIIZ;
    - booleani, bit e mappe di bit;
- IA-32 supporta: interi (segnati e non) da 8,16, 32 bit; BCD da 8 bit; numeri in virgola mobile da 32 e 64 bit;

Formato delle istruzioni

- Un'istruzione è formata da:
  - un *opcode,* che indica quale operazione deve essere svolta;
  - un campo che indica tipo ed eventualmente indirizzo degli operandi;
- la maniera con cui si designano gli operandi in una istruzione si dice *modalità di indirizzamento;*
- Una istruzione può comprendere 0,1,2,3 indirizzi;



- Le istruzioni possono essere di lunghezza fissa o variabile;

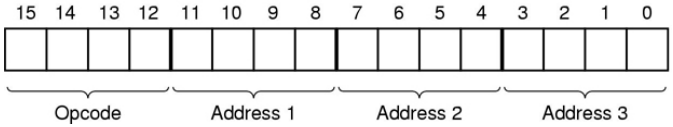


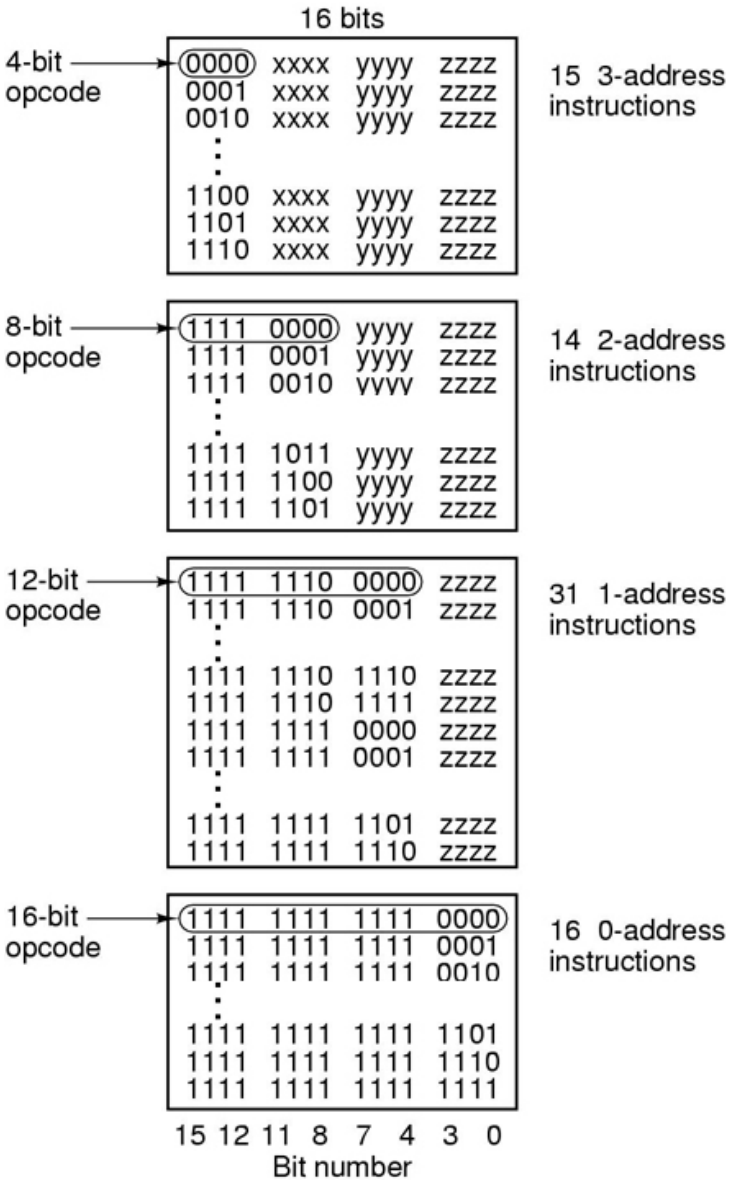
Scelta del formato delle istruzioni

- Obbiettivi:
  - espandibilità futura del set di istruzioni (opcode inutilizzati);
  - basso costo del dispositivo;
  - alte prestazioni;
  - efficienza in spazio; bassa occupazione dei codici oggetto;
- Vincoli del problema:
  - rappresentabilità di tutte le operazioni necessarie;
  - numero di registri;
  - ampiezza dello spazio di indirizzamento;
- Esempio di trade-off:
  - un formato di istruzioni complesso risparmia spazio e tempo di accesso alla RAM, ma richiede uno stadio di decode più complicato e costoso;
  - un indirizzamento a parole fa risparmiare 2 bit su tutti gli indirizzi, ma richiede aggiunte per gestire i singoli byte in una word;
  - lasciare pochi opcode liberi fa risparmiare bit ma preclude la possibilità di espandere il set di istruzioni nelle evoluzioni future;

Schema di *espansione dell'opcode*

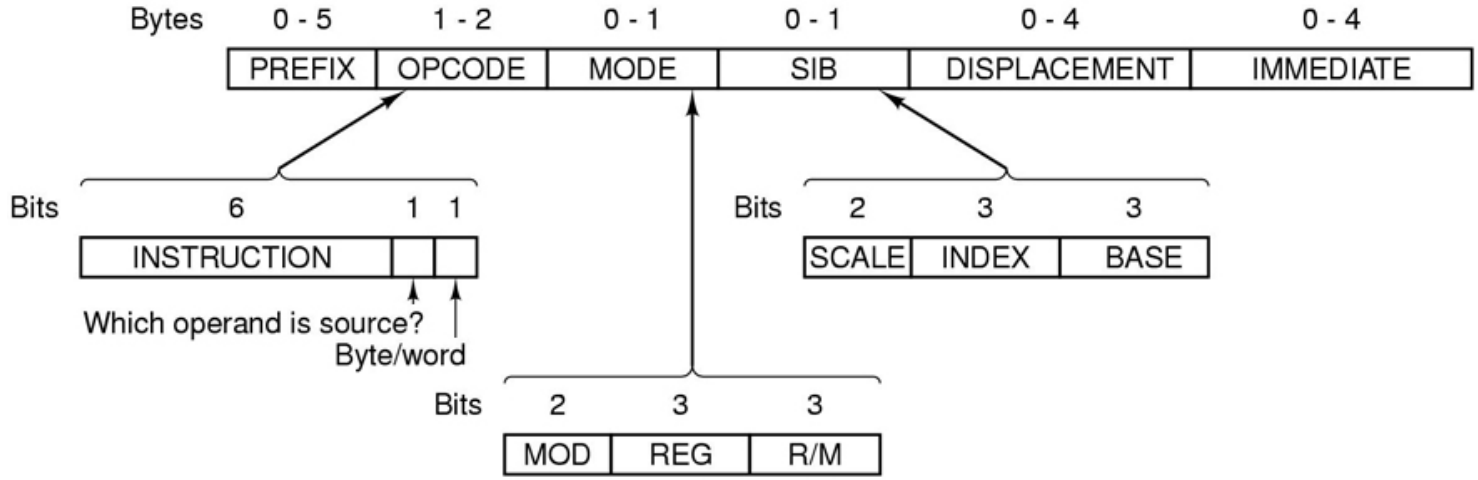
- È uno dei possibili schemi di attribuzione degli opcode;
- È un compromesso fra efficienza spaziale e facilità di decodifica;
- Istruzioni a lunghezza costante: "quelle con molti operandi hanno un opcode breve; quelle con pochi operandi hanno un opcode lungo";
- Esempio:





Formato delle istruzioni IA-32

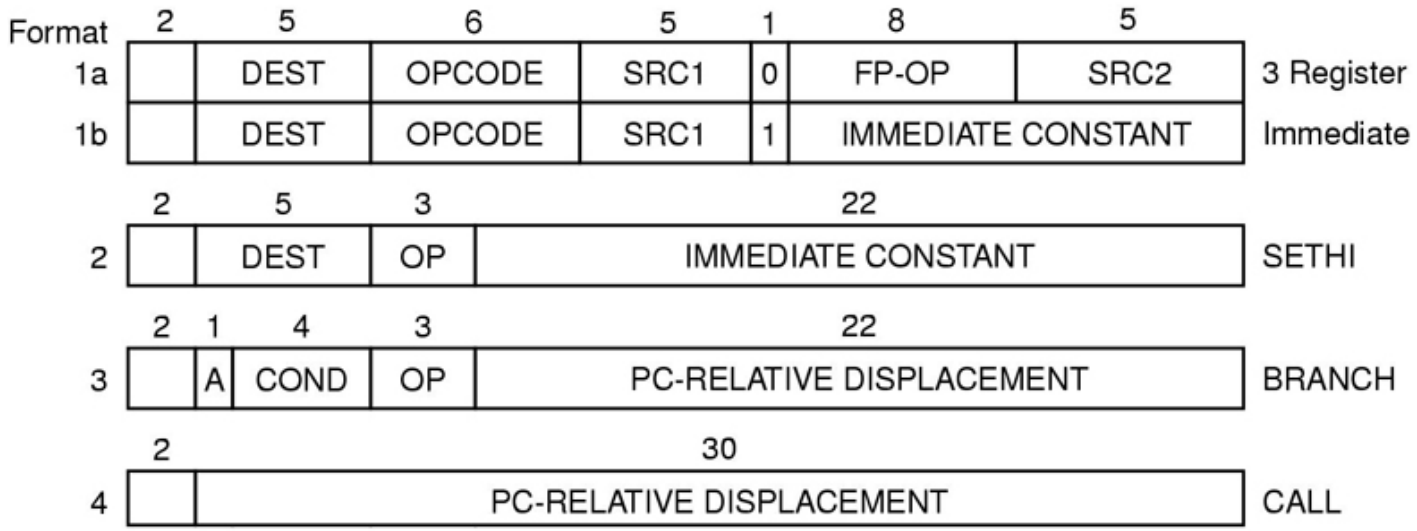
- Formato complesso e irregolare dovuto alla necessità di compatibilità con le vecchie generazioni;
- Fino a sei campi di lunghezza variabile; struttura generale:



- Il sottocampo instruction dell'opcode non si “spezza” => è necessaria una complessa decodifica prima di poter determinare quanto è lunga l'istruzione corrente, e dove inizia la successiva; degrado di prestazioni;
- Compromessi nelle regole di codifica impediscono l'uso di certe combinazioni di registri;

Formato delle istruzioni SPARC

- Formato originario:
  - regolare, progettato "a tavolino" ex novo;
  - istruzioni di lunghezza fissa (32 bit) allineate in memoria;
  - solamente 4 formati di istruzioni previsti:



- Evoluzioni storiche: nuovi formati, ottenuti ritagliando bit
- Al momento 31 formati, in possibile aumento;

Indirizzamento

- Le modalità di indirizzamento di una architettura indicano le combinazioni possibili degli operandi in una istruzione; e come codificare i bit che designano quali sono gli operandi;
- Scelte diverse possibili nel numero di operandi espliciti:
  - 3 indirizzi ADD R1, R2, R3 cioè R1 = R2 + R3 (Sparc)
  - 2 indirizzi ADD R1, R2 cioè R1 = R1 + R2 (Intel)
  - 1 indirizzo ADD R1 cioè ACC += R1 (...)
  - 0 indirizzi ADD (sulla pila) (IJVM)
- Modalità di indirizzamento possibili:
  - immediato
  - diretto
  - registro
  - registro indiretto
  - base + indice
  - ...

Modalità di indirizzamento (1)

- Indirizzamento **immediato**:
  - l'operando viene inserito nell'istruzione;
  - letta l'istruzione, l'operando è disponibile immediatamente;
  - e.g.: `MOV R1, 4`
- Indirizzamento **diretto**:
  - l'indirizzo in memoria dell'operando viene inserito nell'istruzione;
  - si accede sempre alla stessa locazione: utile solo per variabili globali;
- Indirizzamento **registro**:
  - indirizzamento più frequente e più compatto in spazio (l'unico disponibile per le operazioni nelle architetture load/store);
  - indica che l'operando è contenuto in un registro;
  - l'istruzione contiene il codice del registro, che è molto breve;
  - il compilatore cerca di tenere nei registri le variabili più spesso usate;
- Indirizzamento **registro indiretto**:
  - l'operando si trova in memoria, ad una locazione contenuta in un registro;
  - l'istruzione contiene il codice del registro a cui leggere l'indirizzo;
  - permette l'accesso alla RAM senza avere un indirizzo pieno in istruzione;

Indirizzamento: esempio

- Calcolo la somma degli interi del vettore A (1024 celle, ciascuna contenente un intero da 32 bit):

	Istruzioni	Modalità di indirizzamento	Commento
	`MOV R1, #0`	`; reg, imm`	azzera la somma iniziale
	`MOV R2, #A`	`; reg, imm`	indirizzo del vettore A in R2
	`MOV R3, #A+4096`	`; reg, imm`	primo indirizzo oltre il vettore A
`LOOP:`	`ADD R1, (R2)`	`; reg, reg ind`	accumulo in R1 la cella corrente
	`ADD R2, #4`	`; reg, imm`	avanzo di una cella (4 byte)
	`CMP R2, R3`	`; reg, reg`	il prossimo indirizzo cade ancora in A?
	`JLT LOOP`	`; imm (8bit rel)`	se è inferiore non ho finito

Modalità di indirizzamento (2)

- Indirizzamento **indice**:
  - l'operando si trova ad un indirizzo che si ottiene sommando un puntatore costante e un offset contenuto in un registro;
  - esempio: calcolo la somma elemento per elemento di due vettori ($C_i = A_i + B_i$)

	Istruzioni	**Commento**
	`MOV R1, #0`	; primo offset da usare
`LOOP:`	`MOV R2, A(R1)`	; copio *Ai*
	`ADD R2, B(R1)`	; sommo *Bi*
	`MOV C(R1), R2`	; salvo il risultato in *Ci*
	`ADD R1, 4`	; determino il prossimo offset
	`CMP R1, #4096`	; sono ancora negli offset validi ?
	`JLT LOOP`	; continuo

- Indirizzamento **base+indice:**
  - l'operando si trova ad un indirizzo che si ottiene sommando un puntatore ed un offset che si trovano entrambi in un registro;
  - esempio: `MOV R4, (R2+R5)`