ARCHITETTURA DEL CALCOLATORE PARTE 2

AVVERTENZE

Questi appunti non vogliono essere degni sostituti di un buon libro di testo e hanno il solo scopo di aiutare gli studenti dei corsi di informatica che affrontano lo studio dell'architettura di un computer. Gli argomenti affrontati in aula saranno dunque qui trattati in modo stringato, dando per scontata l'attiva partecipazione degli studenti alle lezioni.

Si consiglia quindi, durante lo studio, di integrare queste dispense con i propri appunti.

Per ogni dubbio, chiarimento, o segnalazione contattatemi via mail scrivendo a

francesco.capezi@gmail.com

Francesco Capezio

INSTRUCTION SET ARCHITECTURE

Una ISA (Instruction Set Architecture) è un insieme di **istruzioni** (scritte in **linguaggio macchina**) che una CPU è capace di decodificare ed eseguire. Queste istruzioni sono caratterizzate da un **Operation Code** (Op.Code) e da alcuni parametri (operandi) che servono per l'esecuzione dell'istruzione stessa. A queste istruzioni è anche associato un **codice mnemonico**, ovvero un nome che ne ricorda la funzione.

Una ISA di riferimento può essere, ad esempio:

Op.Code	Mnemonico	Operando 1	Operando 2	Descrizione
00	STORE	Indirizzo	Valore	Salva il valore (operando2) all'indirizzo assegnato (operando1)
10	LOADA	Indirizzo		Carica il **valore** presente all'indirizzo (operando1) nel **registro A**
11	LOADB	Indirizzo		Carica il **valore** presente all'indirizzo (operando1) nel **registro B**
20	OUT			Pone in uscita (sullo schermo ad esempio) il valore contenuto nel **RegistroA**
21	INPUT	Indirizzo		Salva il valore digitato dall'utente all'indirizzo dell'Operando1
30	JMP	Indirizzo		Il programma salta all'indirizzo assegnato
31	JE	Indirizzo		(Jump Equal) Se il valore nel **RegistroA** è **uguale** a quello del **RegistroB** il programma salta all'indirizzo passato, altrimenti prosegue
32	JG	Indirizzo		(Jump Greater) Se il valore nel **RegistroA** è **maggiore** di quello del **RegistroB** il programma salta all'indirizzo passato, altrimenti prosegue
50	END			Termina il programma

Esempio

"*Scrivere un programma che, presi due tasti numerici in ingresso dall'utente, stampi il simbolo > se il primo ingresso è maggiore del secondo, < se viceversa, = se i due valori sono uguali*"

Analizziamo il programma da scrivere

- Prima di tutto il programma deve prendere due ingressi dall'utente.
- Poi serviranno (memorizzati nella RAM) i simboli >, < e =. Memorizziamoli usando i loro **codici ASCII**: sono 60 (<); 61 (=) e 62 (>).
- E' necessario salvare i due ingressi nei registri A e B, in modo da poterli confrontare con le istruzioni JE e JG

- Per capire se il primo valore è maggiore del secondo utilizzo JG. Se è vero salto nella porzione di codice dove stampo >
- Se non è vero si prosegue con il programma e si confrontano i valori per vedere se sono uguali. Uso JE. Se è vero salto alla porzione di codice dove stampo =
- Se anche questo confronto risulta falso proseguo con il programma e in questo caso vuol dire che il primo valore è minore del secondo (non è maggiore ne uguale). Allora stampo < e salto alla fine del programma.
- Ricordiamo che per stampare un carattere è necessario prima caricarlo nel registro A

Ecco come appare il programma in memoria. Usiamo il codice **mnemonico** invece dell'Op.Code

Indirizzo della RAM	**Contenuto della cella**	**Descrizione del programma**
00	INPUT 17	Il programma attende fino a che l'utente non inserisce il primo numero. Questo sarà salvato all'indirizzo 17
01	INPUT 16	Il programma attende fino a che l'utente non inserisce il secondo numero. Questo sarà salvato all'indirizzo 16
02	STORE 15 **60**	Carico il carattere < (codice ascii 60) all'indirizzo 15
03	STORE 14 **61**	Carico il carattere = (codice ascii 61) all'indirizzo 14
04	STORE 13 **62**	Carico il carattere > (codice ascii 62) all'indirizzo 13
05	LOADA 17	Carico il primo numero dell'utente (salvato all'indirizzo 17) nel registro A
06	LOADB 16	Carico il primo numero dell'utente (salvato all'indirizzo 17) nel registro B
07	JG 0C	Confronto il primo numero e il secondo (registri A e B). Se A>B salto all'indirizzo **0C** dove stamperò >
08	JE 0F	Se A non è più grande di B (non sono saltato) provo a vedere se sono uguali. Se A=B salto all'indirizzo **0F** dove stamperò =
09	LOADA 15	A non è uguale a B (non ho saltato). Dunque sarà minore. Carico nel registro A il simbolo <, salvato all'indirizzo 15 dalla istruzione "02"
0A	OUTA	Stampo <, contenuto ora nel registro A
0B	JMP	Salto alla fine
0C	LOADA 13	Questa è la porzione di codice dove stampo >. Prima di tutto lo carico nel registro A (il simbolo è salvato all'indirizzo 13 dall'operazione "04")
0D	OUTA	Stampo >, contenuto ora nel registro A
0E	JMP	Salto alla fine
0F	LOADA 14	Questa è la porzione di codice dove stampo =. Prima di tutto lo carico nel registro A (il simbolo è salvato all'indirizzo 14 dall'operazione "03")
10	OUTA	Stampo =, contenuto ora nel registro A
11	END	Fine del programma
12		
13	>	Spazio dei dati
14	=	
15	<	

16	Secondo numero dell'utente	
17	(primo numero dell'utente	

NOTA

Le istruzioni facente parte di una ISA sono caricate in RAM in **codice macchina** scritto in binario.

Esempio l'istruzione all'indirizzo 03h della RAM è

STORE 14 "="

ovvero "*carica il carattere = all'indirizzo 14*". Abbiamo visto come l'**OpCode** di STORE sia 00, mentre il codice ASCII del segno = sia 61. L'istruzione può dunque essere scritta

00 14 61

Ogni elemento dell'istruzione (Op.Code e Operandi) è costituito da un **codice esadecimale** composto da due cifre (1 byte). **L'istruzione dunque è lunga 3 byte**. Possiamo dunque scrivere l'istruzione in binario, assegnando 4 bit ad ogni cifra esadecimale.

0 0 1 4 6 1

0000 0000 0001 0100 0110 0001

Ecco dunque l'istruzione "*carica il carattere < all'indirizzo 14*" scritta in linguaggio macchina.

NOTA

Le istruzioni JMP, JE e JG eseguono il "salto" del programma semplicemente **modificando il registro PC** (program counter), inserendo l'indirizzo al quale effettuare il salto (che diventa quindi l'indirizzo della prossima istruzione da eseguire)

ARCHITETTURE CISC E RISC

Le istruzioni di una ISA vengono eseguite dalla CPU grazie a dei **microprogrammi** che sono scritti a livello hardware (porte logiche, transistor etc.) all'interno del processore. Ogni processore ha un set di microprogrammi che permette di eseguire un preciso set di ISTRUZIONI.

Esistono due diverse tipologie di ISA: **RISC** (Reduced Instruction Set Computer) e **CISC** (Complex Instruction Set Computer).

CISC – Complex Instruction Set Computer

Nei primissimi calcolatori non esisteva ne sistema operativo ne compilatore. I programmatori dovevano scrivere i programmi direttamente in codice macchina o Assembly.

Nascono successivamente i primi processori dotati di un **set di molte istruzioni complesse** che dovevano aiutare i programmatori nel loro lavoro. Queste istruzioni dovevano avvicinarsi molto alle istruzioni dei linguaggi di alto livello (come il *C*). Ad esempio avevano **modalità complesse di accesso alla memoria** (il programmatore poteva elaborare direttamente i dati in memoria senza utilizzare i registri)

Una architettura CISC ha le seguenti caratteristiche:

- Una ISA complessa che richiede una **struttura complessa del microprocessore** (microprogrammi) che diventa quindi **difficile da progettare.**
- Le istruzioni sono **lente da eseguire**, sia perché i microprogrammi sono più complessi ma, soprattutto, perché accedono molto alla memoria (e con modalità complesse).
- I processori di una architettura CISC utilizzano molti transistor (necessari per eseguire i microprogrammi) e hanno dunque meno spazio per i registri.

Alcuni vantaggi di questa architettura sono:

- E' **facile compilare un software** per queste architetture. Le istruzioni complesse sono già simili a quelle di alto livello.
- I programmi generati dal compilatore risultano più **compatti** (utilizzano meno istruzioni perché sono più potenti) e quindi **occupano meno RAM**.

Questo tipo di architettura si è diffusa verso **la fine degli anni '70** quando la memoria RAM era molto costosa e il divario di velocità tra processore e bus non era molto elevato. Si è dunque preferito favorire il programmatore fornendo istruzioni complesse.

Un tipico esempio di questa architettura è la **INTEL X-86** nata con il processore **8086**

RISC – Reduced Instruction Set Computer

Le architettura RISC nascono successivamente alle CISC **(fine anni '80**) principalmente per due ragioni:

- I programmatori ignoravano le istruzioni particolarmente complesse delle CISC, preferendo utilizzare quelle più semplici
- Il divario di velocità tra CPU e RAM era diventato notevole e i progettisti cercano un modo per limitare l'accesso alla memoria

Una architettura RISC è composta da **poche semplici istruzioni**. Tipicamente l'accesso alla memoria RAM è consentito solo ad alcune istruzioni (ad esempio LOAD e STORE) mentre le altre elaborano i dati contenuti nei registri.

Un set di istruzioni RISC ha diversi vantaggi:

- Le istruzioni richiedono pochi microprogrammi per essere eseguite: i **microprocessori** saranno **molto più semplici da progettare** (conterranno meno transistor, necessari per creare i circuiti che eseguono i microprogrammi).
- Sul chip del microprocessore ci sarà più spazio per i registri
- Le istruzioni semplici saranno **più veloci da eseguire,** sia perché accedono poco e in modo semplice alla RAM (non utilizzo molto il bus) sia perché i microprogrammi sono semplici.

Gli svantaggi principali erano:

- il programma risultante dalla compilazione era piuttosto lungo (tanti istruzioni semplici) e occupava molta RAM
- Era difficile per i compilatori tradurre il linguaggio di alto livello in istruzioni RISC (troppo poco potenti)

I processori RISC più noti sono: gli **ARM** (utilizzati su alcuni palmari e smartphone); **MIPS** (utilizzati sulle console Playstation e Nintendo; **PowerPC** (utilizzati per i supercomputer IBM e, fino al 2006 , da Apple, successivamente passata a Intel Core Duo)

Confronto tra le due architetture

1) Le istruzioni CISC sono sicuramente più lente da eseguire in quanto accedono molto (e in modo complesso) alla memoria RAM

IMPORTANTE E' giusto notare come la parte lenta di calcolatore sia proprio la RAM:

- Il **clock** di una CPU ha ormai raggiunto i **2 GHz**
- L'accesso alla RAM tramite il bus è tipicamente sui **600MHz** (0.6 GHz)

(le velocità attuali sono: clock a 3 GHz e RAM a 1.6 GHz)

2) I processori RISC sono molto più semplici da progettare. E sono più veloci

Il primo RISC costruito (CDC 6600) aveva 74 Op.Code contro i 400 Op.Code diversi dell'8086.

Alcuni processori CISC degli anni '80 utilizzavano 100.000 transistor. Alcuni processori RISC della stessa epoca avevano 40.000 transistor e solo 39 istruzioni erano molto più veloci.

Il successo dei processori CISC dipende solo dalla predominanza sul mercato di **Intel** e **AMD** e dai loro processori con architettura CISC.

Per i consumatori, infatti, è molto più importante ricercare la **compatibilità software con le loro applicazioni** piuttosto che ricercare processori a prestazioni più elevate. Nessuna architettura RISC ha una base utenti così amplia da competere con Intel e, dunque, nessun produttore di software investirebbe su questo tipo di architetture.

Le moderne architetture Intel sono però un **ibrido tra CISC e RISC.** Le istruzioni più complesse delle architettura vengono infatti tradotte internamente in istruzioni più semplici.

PROCESSORI INTEL AD ARCHITETTURA COMPATIBILE X-86

1974	8080	Non ha ancora una architettura X-86 ma è il precursore dei successivi
1976	8086	Primo processore della famiglia X-86. 1MB di RAM e bus a 16bit E' il processore utilizzato sugli shuttle!
1976	8088	E' un processore simile all'8086 ma con bus a 8bit
1982	80286	16MB RAM Processore 12.5 MHz (ma i concorrenti arrivavano a 25 MHz
1985	80386	Primo processore a 32 bit (architettura IA-32, compatibile alla X-86) Primo processore con una cache L1 (diventa importante velocizzare l'accesso alla RAM per concorrere con le architetture RISC)
1989 1994	486	Architettura 32bit Primo processore con FPU (floating point unit) dedicata alle operazioni in virgola mobile Il 486DX2/66 era considerato la dotazione minima per giocare
1993	Pentium	Eseguiva due istruzioni in parallelo (pipeline) gestiva una cache di secondo livello L2 montata sulla motherboard aveva un BUG che ne ha fatto ritirare molti
1997	Pentium Pro	È il primo che può montare più di 4GB di RAM (bus indirizzi a 36bit=64GB) Aveva la cache L2 direttamente sul processore
1997	Pentium II	La Intel ha iniziato a dividere i processori in due fasce: Celeron: processore entry level (economico) senza cache L2 Xeron: processore per server con cache L2
1999	Pentium III	Primo processore Intel con clock maggiore di 1 GHz
2000	Pentium 4	Non tenne il passo coi tempi: era costoso e scaldavo molto. Non aveva prestazioni adeguate

IL PROCESSORE 8086

Primo processore Intel con architettura X-86, del 1976, aveva le seguenti caratteristiche principali:

- La CPU è divisa in due parti EU (Execution Unit) e BIU (Bus Interface Unit)
- Clock (velocità processore): 4,77 MHz
- RAM: 1 MB
- DBUS: 16bit
- ABUS: 20 bit (2^{20} = 1M di indirizzi)
- Spazio indirizzamento I/O: 16bit → 64KB
- 8 line di interrupt e canale DMA
- Il processore dispone di **14 registri** a 16 bit

Le line del DBUS sono **in comune** con quelle del ABUS. Le linee in totale sono 20 e un segnale del CBUS determina se sono utilizzate per un indirizzo (tutte e 20) o per un dato (solo le prime 16)

La ISA è composta da istruzioni lunghe da 1 a 4 byte (a seconda degli operandi). Essendo un'architettura di tipo CISC le **istruzioni accedono spesso alla memoria** per salvare i dati temporanei dei calcoli (non ci sono abbastanza registri)

Questo è stato fino al 2011 il processore montato nei computer dello Shuttle Nasa.

Questo processore è 400 volte più lento dei processori montati sugli attuali smartphone e ha 2000 volte meno RAM.

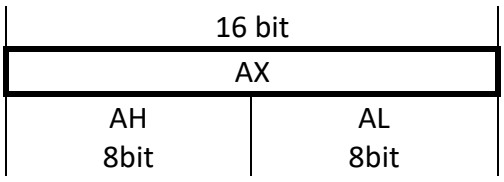
Registri ad uso generale

Sono 4 registri a 16 bit ma possono anche essere utilizzati come registri a 8 bit (ogni registro viene diviso in 2)

AX	Accumulatore	Utilizzato per i calcoli
BX	Base	E' l'unico dei registri generali che può contenere un indirizzo della RAM
CX	Contatore	Utilizzato per variabili di tipo contatore (ad esempio nei loop e nei cicli for)
DX	Registro I/O	Può contenere indirizzi delle periferiche I/O Utilizzato anche per moltiplicazioni e divisioni

Come detto, le istruzioni possono accedere a questi registri anche dividendoli in 2 da 8 bit.

In questo caso non si usa più la X ma **L** per gli 8 bit meno significativi e **H** per quelli più significativi



Discorso analogo vale per gli altri registri (BX = BH+BL etc)

Registri Indice

Sono utilizzati per la manipolazione degli array. Gli array sono un insieme di dati dello stesso tipo contenuti in una struttura lineare. L'array ha un determinato numero di celle e può contenere altrettanti dati, posizionati uno dopo l'altro.

Nella memoria RAM un array è una serie di celle contigue e viene identificato **dall'indirizzo della prima cella** e dal numero di celle.

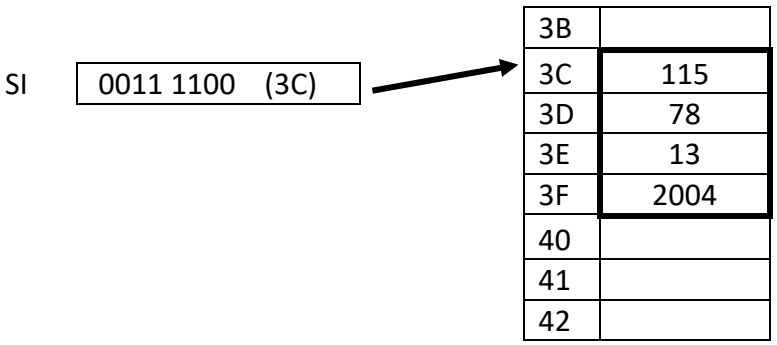
SI	Indice Sorgente	E' il registro che contiene l'indirizzo della prima cella dell'array
DI	Indice Destinazione	Contiene l'indirizzo di destinazione dell'array, al seguito delle operazioni che si vogliono fare

Esempio: Array composto da **4 valori interi**

V è il nome del'array (vettore) e identifica l'indirizzo di partenza.

V[0] è il contenuto della prima cella ovvero V[0]=115. Di conseguenza V[1]=78 V[2]=13 V[3]=2004.

Se volessimo operare su questo array dobbiamo utilizzare il registro indice SI che conterrà il valore 3C



Registri di Stack

Lo **Stack** è **l'aria di memoria dove vengono salvate le variabili relative ad un processo** in esecuzione.

I dati vengono salvati iniziando dal **fondo dello stack (base)** e rimossi (ad esempio quando una funzione del programma termina) iniziando dalla **cima.**

Questa politica di gestione dei dati si chiama LIFO: Last IN First OUT (l'ultimo ad essere inserito è il primo ad essere prelevato – rimosso).

BP	Base Pointer	è il registro che contiene l'indirizzo della base dello stack

SP Stack Pointer contiene l'indirizzo della cima attuale dello stack

I due registri indicano quindi qual è l'area di memoria (**dati**) che viene utilizzata attualmente dal programma in esecuzione.

Instruction Pointer

E' simile al già noto Program Counter. Il registro ha 16 bit ma l'area dati (ABUS) ha 20bit. Questo registro è quindi la **parte meno significativa** dell'indirizzo della prossima istruzione da eseguire

IP Instruction Pointer Serve per formare l'indirizzo della prossima istruzione da eseguire

Registri di Segmento

Questi sono i registri utilizzati per l'indirizzamento.

La memoria nell'architettura X-86 si dice **segmentata**, ovvero ogni indirizzo è composto da due parti: il **segmento** (la parte più significativa) e **l'offset** (quella meno significativa)

Un segmento quindi contiene più indirizzi (a seconda dell'offset) e, dunque, contiene diverse celle.

Esempio

Indirizzi a 5 bit: 3 bit di segmento e 2 di offset

000 00		Primo segmento: 000 Contenente 4 celle
000 01		
000 10		
000 11		
001 00		Secondo segmento 001 Contenente altre 4 celle
001 01		
001 10		
001 11		

Ogni segmento di memoria ha delle proprietà che valgono per tutte le sue celle. Ad esempio

- E' riservato per le istruzioni o per i dati (stack)
- E' protetto (accessibile solo al sistema) oppure no (accessibile alle applicazioni)
- Etc

1) Se l'indirizzo ha una parte di **segmento con pochi bit** e **tanti bit di offset**

- Si hanno pochi segmenti con molte celle per ognuno (grandi segmenti di memoria)
- Sono più facili da gestire per il Sistema Operativo
- Si ha uno spreco di memoria (il segmento non è utilizzato nella sua interezza)

2) Se l'indirizzo ha una parte di **segmento con tanti bit** e **pochi bit di offset**

- Si hanno tanti segmenti con poche celle (piccoli segmenti di memoria)
- La memoria è ottimizzata (poco spreco)
- E' più difficile da gestire per il Sistema Operativo

L'indirizzo di segmento è contenuto negli appositi registri:

CS	Code Segment	Contiene la parte più significativa dell'indirizzo della prossima istruzione da eseguire. L'offset è contenuto nel registro **IP** (Instruction Pointer)
DS	Data Segment	Contiene l'indirizzo di un segmento con i dati. Offset: BX, SI, DI
ES	Extra Segment	Contiene l'indirizzo di un segmento con i dati. Offset: BX, SI, DI BX contiene l'offset nel caso di un indirizzo generico SI e DI contengono l'offset nel caso si tratti di array
SS	Stack Segment	Contiene l'indirizzo di un segmento utilizzato come stack del programma. L'offset dell'indirizzo è contenuto in BP e SP nel caso che si tratti rispettivamente di un indirizzo della base o della cima dello stack

Schema architettura del processore Intel 8086