Tecniche di Programmazione
A.A. 2015/2016

Introduzione all'Architettura dei Calcolatori

Giorgio Grisetti Luca Iocchi
email:{grisetti,iocchi}@dis.uniroma1.it

21 febbraio 2016

La maggior parte dei dispositivi elettronici che usiamo nella vita di tutti i giorni sono divenuti veri e propri computer che eseguono complessi programmi e sono governati da sofisticati sistemi operativi. Si pensi a smartphone, televisori automobili etc. La qualità ed il successo di un prodotto è determinata dalle funzioni che esso può svolgere. Ciascuna di queste funzioni è implementata sul dispositivo mediante uno o più programmi che interagiscono tra loro.

Programmare significa essere in grado di sintetizzare procedure automatiche per la soluzione di problemi (algoritmi). Per essere eseguiti da un calcolatore, tali algoritmi vanno espressi in una forma ad esso comprensibile, attraverso uno dei tanti linguaggi di programmazione. La sintesi di un algoritmo richiede di pensare ad un livello di astrazione adeguato al linguaggio scelto, anche se alla fine tutto deve essere eseguito dalla CPU dell'elaboratore. In questo corso si utilizzeranno i linguaggi C e C++, che rappresentano una delle scelte dominanti nello sviluppo di applicazioni in cui l'efficienza è un requisito essenziale.

L'intangibilità di un programma ed il ridotto (se non nullo) costo di replicazione, ha portato inizialmente a considerare i programmi diversi da altri prodotti industriali, come automobili o telefoni. Con il tempo ci si è resi conto che un programma deve rispettare gli stessi criteri (se non di più) di ogni altro prodotto, quali correttezza, affidabilità, robustezza, efficienza, usabilità, manutenibilità, ecc.

Tali considerazioni hanno portato allo sviluppo di linguaggi di programmazione che semplificano la produzione di software che soddisfi tali criteri. Rimane comunque al programmatore il compito di scrivere un buon software. Un buono scalpello non serve a nulla in mano ad uno scultore non dotato.

1 Struttura del Calcolatore

Un calcolatore è una macchina che esegue un programma specificato dall'utente. Il programma viene eseguito da un circuito (la CPU), che è in grado di leggere e scrivere su una memoria temporanea ed interagire con delle periferiche di I/O. Memoria e dispositivi di I/O interagiscono con la CPU attraverso dei "canali di comunicazione", detti bus. Le informazioni (o dati) da leggere e scrivere viaggiano sul bus dati. Ad esempio, il valore di una variabile letta da/scritta nella memoria viene trasferita tra la CPU e la memoria (o viceversa) attraverso il bus dati. Le informazioni sul "dove" il dato deve essere scritto viaggiano sul bus indirizzi. Infine, le informazioni su "cosa" fare con le informazioni presenti negli altri due bus viaggiano sul bus di controllo. Ad esempio, se la CPU vuole scrivere un certo dato in una locazione di memoria manderà sul bus dati il valore della variabile da scrivere, scriverà sul bus indirizzi la locazione di memoria in cui la variabile deve essere scritta e specificherà sul bus di controllo che l'operazione che deve essere compiuta è una scrittura (e non, per esempio, una lettura).

La figura 1 illustra un semplice diagramma a blocchi di tale architettura.

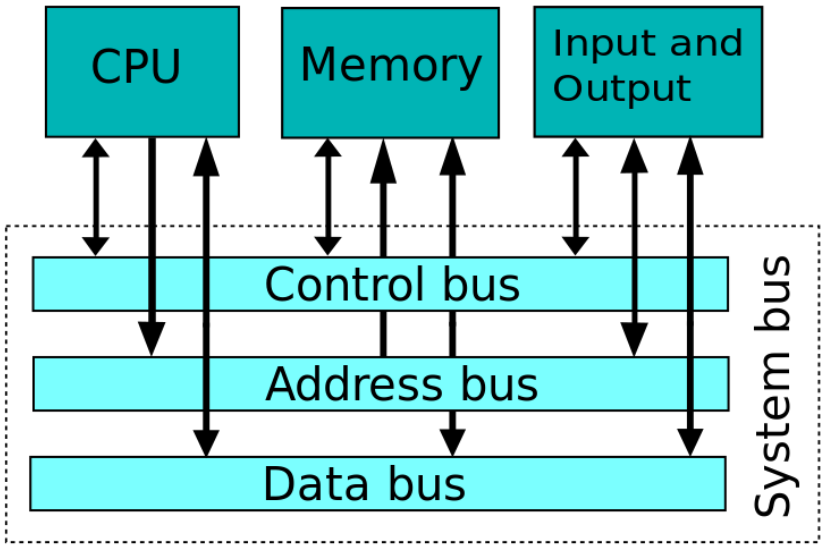

Figura 1: Architettura Schematica di un Calcolatore. (fonte, Wikipedia)

1.1 CPU

La CPU è uno speciale circuito che è in grado di eseguire un programma. Un programma si presenta in memoria come una sequenza di numeri (detti istruzioni). Ogni istruzione consiste in:

- **opcode**: che dice alla CPU cosa deve fare (ad esempio una sottrazione, un'addizione, un confronto od un "salto");
- **operandi**: che rappresentano i dati su cui la CPU opera (ad esempio le locazioni di memoria dei dati da sommare, i loro valori, o l'indirizzo del programma da cui l'esecuzione deve continuare nel caso l'opcode sia un "salto").

In figura 2 è illustrata l'architettura di una diffusa CPU ad 8 bit. Sebbene antiquata, tale architettura contiene tutti gli elementi fondamentali.

All'interno della CPU sono presenti i seguenti circuiti:

- **banco di registri**: che sono locazioni di memoria temporanea molto veloce che mantengono i risultati temporanei del programma. Tra i registri, del banco, alcuni hanno funzioni particolari:
  - **Program Counter**: è uno speciale registro che contiene l'indirizzo dell'istruzione corrente del programma. Il program counter è incrementato ad ogni istruzione, per puntare alla successiva e viene direttamente alterato dalle istruzioni di "salto".
  - **Stack Pointer**: è uno speciale registro che contiene l'indirizzo della locazione corrente dello stack. Lo stack è una porzione di memoria gestita in modo particolare (vedi Sezione 1.2) e deputata a contenere lo stato delle variabili temporanee, il valore dei parametri di funzioni e l'indirizzo di ritorno dalle funzioni.
  - **Flag Register**: è un registro i cui bit vengono alterati a seguito dell'esecuzione di particolari istruzioni. Ad esempio, un'istruzione di confronto tra due numeri imposterà ad uno un determinato bit del flag register, se i numeri sono uguali oppure ne imposterà un altro ad uno se il primo operando è maggiore del secondo, ecc.
- **ALU**: o unità aritmetico logica, esegue le operazioni aritmetiche e logiche sui dati provenienti dai registri o dalla memoria. Il risultato viene scritto in memoria o su un registro, a seconda della particolare combinazione di opcode ed operandi, ed il flag register viene modificato di conseguenza.

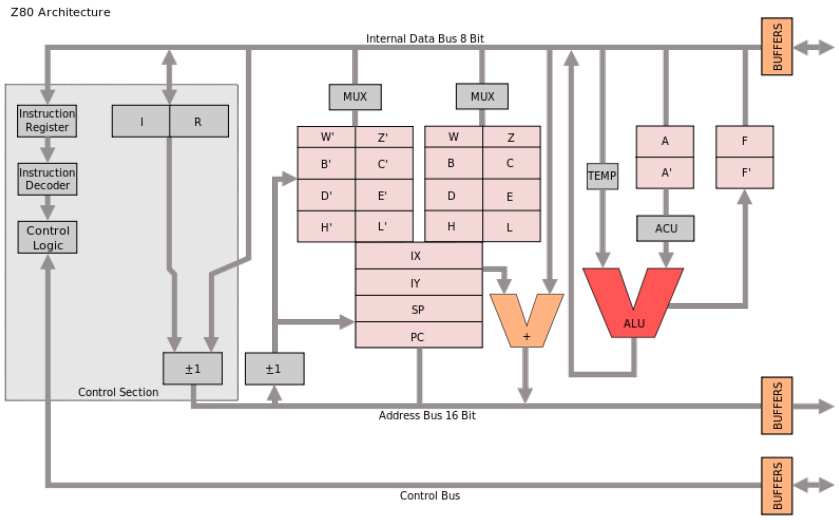

Figura 2: Schema a blocchi di una diffusa CPU ad 8 bit (Z80). (fonte, Wikipedia). PC: Program Counter; F, F': Flag Registers, SP: Stack Pointer, A,B,C,..., A',B',C',...: banco di registri; Control Section: unità di controllo.

- **unità di controllo**: è un circuito logico che governa l'interazione tra i vari dispositivi all'interno della CPU e controlla i bus. L'unità di controllo è responsabile:
  - del caricamento delle istruzioni dalla memoria, e della loro decodifica,
  - del caricamento dei dati (operandi) dalla memoria,
  - dell'esecuzione dell'istruzione,
  - della scrittura dei risultati su memoria o su registro,
  - del controllo del flusso del programma, interagendo con il program counter e lo stack pointer.

  L'unità di controllo è inoltre responsabile della gestione del DMA e degli interrupt, ovvero dell'invocazione di particolari routines a seguito del verificarsi di eventi particolari (ad esempio, la pressione di un tasto della tastiera).

1.2 Memoria e Stack

All'interno di un calcolatore esistono diversi tipi di memoria. La memoria RAM la memoria principale e può essere vista come una sequenza di byte organizzati in un array (vettore). I dati sono memorizzati nelle locazioni (celle) di tale array. L'indirizzo di un dato in memoria corrisponde all'indice di tale array. Dati più grandi di un byte sono memorizzati in locazioni contigue.

Una particolare regione di memoria viene gestita dalla CPU in modo diretto, per memorizzare informazioni necessarie alla corretta esecuzione del programma. In particolare, ci consente di memorizzare informazioni che non possono essere memorizzate all'interno della CPU in quanto di dimensioni potenzialmente troppo grandi.

Tale regione di memoria è chiamata "Stack" (ovvero "pila") e funziona esattamente come tale. Uno stack può essere visto come un array di dimensioni variabili, in cui si possono compiere solamente tre operazioni:

- $push(d)$, che inserisce un dato d in ultima posizione ed aumenta di uno la dimensione dell'array;
- $top()$, che ritorna l'elemento in ultima posizione, senza modificare l'array;
- $pop()$, che elimina l'elemento in ultima posizione.

È attraverso lo stack che la CPU "ricorda" dove il flusso di esecuzione di un programma deve continuare quando una funzione termina.

La parte restante della memoria che non è usata come stack si chiama "heap" e viene gestita dal sistema operativo (che altro non è che un programma, vedi Sezione 2). La memoria heap viene allocata e rilasciata/ceduta dai programmi che ne fanno richiesta attraverso l'invocazione di funzioni come `malloc()` o `free()`. L'uso dell'heap nel nostro modello semplificato di CPU non richiede particolari registri. Tuttavia, nelle moderne CPU esistono meccanismi di protezione della memoria che impediscono ad un programma di leggere i dati di un altro, ma la loro gestione è trasparente al programmatore non di sistema.

1.3 Dispositivi di Input/Output

Questa classe di dispositivi comprende hard disk, schede di rete, tastiere, mouse e tutte le periferiche che possono essere connesse al sistema di base costituito da CPU e RAM. In genere è compito del sistema operativo garantire interfacce "uniformi" alle periferiche.

2 Concetto di sistema operativo

I primi calcolatori fornivano all'utente un accesso diretto all'hardware. In prima istanza, questo può sembrare buono, ma richiede una conoscenza profonda dell'hardware sottostante per eseguire le funzioni più semplici. La stampa di un carattere a video poteva richiedere la scrittura di complesse funzioni che interagivano con l'hardware del monitor sincronizzandosi con tempi di tracciamento vari per evitare fastidiosi sfarfallii. La lettura di un carattere da tastiera comportava la scrittura di funzioni di gestione delle interruzioni. Ben presto ci si è resi conto che per rendere più fruibile l'hardware occorreva offrire uno strato software che da un lato ne mascherasse la complessità e dall'altro ne esponesse le potenzialità in modo semplice. In poche parole occorreva fornire un'*astrazione* all'hardware per renderlo più usabile. In seguito ci si è resi conto che, sfruttando tali astrazioni, i programmi potevano essere facilmente portati da un sistema all'altro: se la funzione `printChar(char c)` fa la stessa cosa sul mio VAX e sul tuo PDP-11, possiamo scambiarci il codice sorgente dei programmi.

Inoltre i computer erano costosi e non tutti potevano averne uno. Era perciò comune che più persone ne condividessero l'uso. All'inizio i programmi degli utenti venivano fatti girare "in sequenza", a lotti. L'utente portava il suo programma in un ufficio sotto forma di comode schede perforate, l'impiegato inseriva le schede in un pratico lettore e congedava l'utente. Il giorno successivo l'utente passava a ritirare un listato che conteneva il risultato del suo programma.

Successivamente, le schede perforate caddero in disuso ed i computer iniziarono ad essere dotati di monitor e tastiere che permettevano all'utente di interagire con in computer in maniera diretta. Ma i computer erano sempre pochi, e più utenti volevano avere un accesso concorrente alle risorse. A tale scopo i sistemi operativi che consistevano all'inizio di semplici librerie di funzioni ed un interprete di comandi in grado di lanciare un programma alla volta, vennero estesi in modo da fornire funzionalità di multi-tasking. Un sistema operativo multi-tasking è in grado di fornire una percezione che più programmi girino concorrentemente. In realtà i programmi sono eseguiti uno alla volta, ovvero uno per "core", per brevi istanti di tempo. La velocità dell'alternarsi dà l'impressione che l'esecuzione sia concorrente.

Più persone che usavano lo stesso elaboratore cominciarono a lamentarsi che, a causa di un errore nel programma del vicino che aveva scritto nella locazione di memoria sbagliata, il loro processo era stato interrotto. Per prevenire questo, i sistemi operativi multi-tasking furono dotati di meccanismi di protezione della memoria che impedivano ai programmi di "disturbarsi" tra loro e di disturbare il sistema operativo.

È bene notare che nell'accezione più pura del termine, un sistema operativo è un programma che gestisce solamente:

- **processi**: ovvero i programmi in esecuzione, permettendo di lanciarli ed interromperli;
- **memoria**: attraverso opportune chiamate di sistema (che in C sono invocabili attraverso `malloc()`, `free()` e derivati);
- **file system**: ovvero la creazione, cancellazione, lettura, scrittura di file e directory, e la gestione dei permessi. Tali funzioni sono accessibili in C attraverso `fopen()`, `fclose()` e derivati);

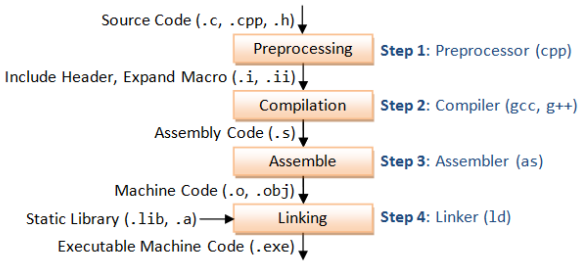

Figura 3: Processo di compilazione

- **dispositivi di I/O**: attraverso la definizione di interfacce uniformi che mascherano lo specifico dispositivo. Il sistema operativo è tipicamente esteso attraverso particolari programmi deputati alla gestione di hardware specifico (ad esempio, una scheda video NVIDIA). Tali programmi sono chiamati driver.

Si noti che le interfacce grafiche sono comunemente confuse con i sistemi operativi, altre volte per specifiche scelte di marketing le due funzionalità sono "fuse".

3 Alcuni strumenti per la programmazione

Di seguito menzioniamo alcuni strumenti utili per la programmazione dei calcolatori.

- **Editor**: Un editor è un programma per scrivere, leggere e modificare file di testo. Tali file di testo contengono il codice sorgente del nostro programma. Ogni programmatore usa l'editor che preferisce. I file sorgenti in C/C++ sono comunemente dotati di estensione `.c`, `.cc` o `.cpp`, mentre i file di intestazione (header) hanno estensione `.h`, `.hh` o `.hpp`

- **Compilatore**: Un compilatore è un programma che traduce un file sorgente (`.c`) in un file "oggetto" (con estensione `.o`). Un file oggetto contiene delle sezioni in linguaggio macchina, che potrebbero essere direttamente eseguibili dalla CPU

- **Linker**: Un linker è un programma che, dati un insieme di file oggetto, li "collega". Più precisamente, a partire da più file oggetto, il linker genera un singolo eseguibile od una libreria "fondendo" i file oggetto tra loro, risolvendo i simboli e verificando che tutte le funzioni invocate siano effettivamente state definite all'interno di uno dei file oggetto. Se una funzione `f()` nel file oggetto `f.o`, chiama una funzione `g()`, definita nel file oggetto `g.o`, in linker metterà l'indirizzo di partenza di `g()` al posto del simbolo, in corrispondenza della chiamata.

- **Debugger**: un debugger è uno strumento per analizzare i programmi durante la loro esecuzione alla ricerca di errori.

- **Make**: è uno strumento per la compilazione condizionale. Permette di risparmiare tempo nella compilazione di progetti complessi costituiti da tanti file, ricompilando solo quelli che sono cambiati. Per usare Make è necessario scrivere semplici script, noti come Makefile.

4 Processo di Compilazione

Il processo di compilazione di un codice sorgente C è illustrato in figura 3.

Un programma C ecostituito da uno o più file `.c` e (opzionalmente) dai corrispondenti file header `.h`. Per compilare un file sorgente C in un file oggetto si usa il seguente comando:

```
g++ -c -o mioFile.o mioFile.c
```

L'opzione `-c` serve a generare un file oggetto, e non un eseguibile (nel caso la porzione di programma non contenga una funzione `main`). L'opzione `-o` serve a dire al compilatore che il nome del file oggetto da generare è `mioFile.o`.

È possibile generare un unico oggetto da più file `.c` con il comando:

```
g++ -c -o mioFile.o mioFile1.c mioFile2.c ... mioFileN.c
```

Prima di eseguire la compilazione vera e propria di un file, il compilatore esegue una fase di "preparazione" del sorgente, eseguendo le direttive del compilatore, contrassegnate nel codice sorgente dal carattere iniziale `#`.

Una volta generati tutti i file oggetto del progetto, si possono linkare assieme, attraverso il seguente comando:

```
g++ -o mioFile  mioFile1.o mioFile2.o ... mioFileN.o
```

È possibile eseguire la compilazione ed il linking in un solo comando:

```
g++ -o mioFile  mioFile1.c mioFile2.c ... mioFileN.c
```

Come detto precedentemente, per la compilazione di progetti complessi, è conveniente l'uso di strumenti adeguati (ad esempio, Make).

4.1 Individuazione degli errori di un programma

Un ruolo fondamentale di un compilatore quello di verificare se il programma è scritto rispettando le regole del linguaggio di programmazione. Quando ciò non avviene, il compilatore interrompe il processo di compilazione e genera dei messaggi che indicano la presenza di errori nel codice sorgente. La comprensione e la correzione di questi errori è un processo importante dello sviluppo del software.

In alcuni casi, il compilatore genera anche degli avvisi (warning), che pur non essendo errori e quindi non impedendo la generazione dei file oggetto e dei file eseguibili, avvisano il programmatori di possibili problemi che potrebbero verificarsi durante l'esecuzione del programma. È buona norma quindi risolvere anche i warning prima di procedere all'esecuzione del programma.