FAQ di Calcolatori Elettronici

G. Lettieri

8 luglio 2011

1 Esame

Q 1.1 *Che materiale è possibile portare alla prova pratica?*

(*Ordinamento 509*) Esclusivamente materiale che riguarda le istruzioni assembler e un manuale di Java (ad es. il libro del Prof. Frosini). Per quanto riguarda l'assembler, sono ammessi, ad esempio:

- libro di Reti Logiche sulle istruzioni assembler che riguardano la ALU;
- fotocopie (prese ad es. dal libro di Calcolatori Elettronici sulla corrispondenza tra C++ ed assembler) sulle istruzioni della FPU.

Non sono ammessi altri testi, in particolare non sono ammesse le parti del libro sulla corrispondenza tra C++ ed assembler che contengono gli esercizi svolti. Non sono ammessi fogli o appunti personali.

(*Ordinamento 270*) Materiale che riguarda le istruzioni assembler, più i volumi II e III di "Architettura dei Calcolatori". Per quanto riguarda l'assembler, sono ammessi, ad esempio:

- libro di Reti Logiche sulle istruzioni assembler che riguardano la ALU e la FPU;
- fotocopie (prese ad es. dal libro di Calcolatori Elettronici sulla corrispondenza tra C++ ed assembler) sulle istruzioni della FPU.

Non sono ammessi altri testi, in particolare non sono ammesse le parti del libro sulla corrispondenza tra C++ ed assembler che contengono gli esercizi svolti. Non sono ammessi fogli o appunti personali.

Q 1.2 *Per quanti appelli viene conservata la prova pratica?*

Per tutti gli appelli:

La prova scritta mantiene la sua validità per 5 appelli che prevedono prova scritta. La prova scritta può essere conservata per una volta, a discrezione della commissione, anche se la prova orale non ha avuto successo, fino al termine massimo della sua validità.

Per l'appello di Aprile:

Possono usufruire dell'appello di Aprile, che prevede la sola prova orale, solo gli studenti fuori corso. La prova scritta, che deve essere già stata sostenuta ed essere ancora valida, è stata conservata anche se la prova orale è stata sostenuta senza successo nel terzo appello di Gennaio-Febbraio, a prescindere dal fatto che fosse già stata conservata.

Q 1.3 *Quanto dura la prova pratica?*

Due ore.

Q 1.4 *Se ho già una prova pratica consegnata, posso provare a venire un'altra volta e decidere se mantenere la vecchia o la nuova?*

No. Non appena ci siede alla prova pratica, eventuali altre prove già consegnate in appelli precedenti vengono annullate.

Q 1.5 *Mi trovo nella situazione X, devo svolgere l'esercizio di programmazione in Java?*

(*Ordinamento 509*) Sì.
(*Ordinamento 270*) Java?!

Q 1.6 *Come devo fare per iscrivermi all'orale?*

Per gli appelli ordinari, basta presentarsi il primo giorno dell'orale (così come previsto dal calendario). In quel giorno viene fatto un appello, in base a tutte le prove pratiche ancora valide in nostro possesso. Se intendete fare l'orale in quella sessione, rispondete semplicemente all'appello. Per gli appelli straordinari, controllate sul sito di Ingegneria se è disponibile il form per l'iscrizione online.

Q 1.7 *A che ora comimcia l'orale?*

In genere alle 9:00. A volte ci sono ritardi imprevisti.

Q 1.8 *Se non dovessi superare la prova orale, la mia prova pratica viene conservata?*

Dipende da come è andato l'orale. In ogni caso, la prova pratica non viene conservata per più di una volta.

Q 1.9 *Qual è il voto minimo della prova pratica per l'ammissione all'orale?*

Non c'è un voto minimo. Però la prova orale è fortemente sconsigliata per chi ha ottenuto meno di 15 punti alla prova pratica.

2 Autocorrezione

Q 2.1 *Quanto tempo ho per portare a termine l'autocorrezione?*

È sufficiente che il compito sia stato corretto prima che vi presentiate all'orale.

Q 2.2 *Il mio compito risulta corretto sul sito, ma non mi compare nessun link per richiedere il voto!*

Se il vostro compito è corretto, non c'è alcun bisogno di richiedere il voto. Il voto va richiesto quando il compito era scorretto, lo avete modificato tramite il sito e siete soddisfatti delle modifiche apportate. Ad esempio, anche se, grazie alle modifiche, il compito risulta finalmente corretto, potreste voler eliminare o rendere più precisa qualcuna delle modifiche, per far capire meglio quale era l'errore commesso.

Q 2.3 *Il compito funzionava correttamente alla prova pratica, l'ho anche compilato a casa su Windows SuperXP/2015 Service Pack 36 e non mi dava nessun errore. Sul sito, invece, mi dice che non esegue!*

Leggete l'esempio due su questa pagina e poi la domanda 3.3 di questa FAQ.

Un'altra possibilità è che abbiate scritto "`jmpl`" invece di "`jmp`". In questo caso gcc interpreta l'istruzione come un salto indiretto che, molto probabilmente, non è ciò che volevate. Togliete la "l" e non vi preoccupate.

Q 2.4 *Il compito funzionava correttamente alla prova pratica, l'ho anche compilato a casa su Windows Vista firmato da Bill Gates e non mi dava nessun errore. Sul sito, invece, mi da errori di sintassi in tutte le istruzioni* `faddpl`*!*

Si tratta di una imprecisione di DJGPP: quelle istruzioni sono scorrette perché la "l" (che specifica operandi di tipo reale doppio) non ha senso: gli operandi sono entrambi nella FPU, e quindi sono di tipo reale esteso. gcc 2.7.2.1, che è quello che viene usato sul sito (e su cui la versione di DJGPP che usiamo è basata) non ha questa imprecisione, e quindi rileva l'errore. Si tratta, in ogni caso, di errori veniali. Correggeteli senza paura.

3 Assembler

Q 3.1 *Quando provo a compilare, mi compare il seguente errore:*

```
c:/djgpp/tmp/cbaaaaa(.text+0xac):fake: undefined reference to 'xxx'
```

Cosa vuol dire?

È un errore del linker: dice che non riesce a trovare la definizione dell'etichetta `xxx`. Succede, normalmente, se:

1. avete dimenticato il simbolo `%` davanti al nome di qualche registro (ve ne accorgete perché `xxx` è proprio il nome del registro in questione). L'assemblatore ha interpretato il nome del registro come una etichetta di cui, poi, il collegatore non ha trovato la definizione;

2. (più frequentemente) avete sbagliato la traduzione del nome di qualche funzione da C++ ad assembler, o non l'avete resa "global". Ricontrollate l'etichetta all'inizio della funzione, controllate di aver scritto la direttiva `.global` e che le due etichette coincidano; controllate infine le istruzioni `call`.

Q 3.2 *Quando provo a compilare, mi compare il seguente errore:*

```
xxx.s:Assembler messages:
xxx.s:0: Warning: end of file not at the end of a line; new line inserted
```

Cosa vuol dire?

Il file da assemblare deve terminare con un *a capo*. Se così non è, state attenti, perchè l'ultima riga del file (quella senza l'*a capo* finale) viene *ignorata*. Normalmente, quella riga conteneva una `ret`: se è così, il vostro programma, invece di tornare al chiamante, "tira dritto" e, generalmente, l'effetto è quello descritto nella domanda successiva.

Q 3.3 *Quando eseguo un programma, ottengo il seguente output:*

```
Exiting due to signal SIGSEGV
Stack fault at eip xxxxxxxx
```

e poi il contenuto in esadecimale di tutti i registri. Come posso individuare l'errore?

Il programma è stato interrotto per via di un accesso non consentito alla memoria e quella è la stampa dello stato del processore al momento dell'accesso scorretto. Ovviamente le possibili cause dell'errore sono molteplici, anche se, dai vostri compiti, ho visto che le più frequenti sono:

- dimenticare il dollaro prima di qualche costante (soprattutto, non so come mai, nell'istruzione `test`);

- usare una `movl` al posto di una `leal` o viceversa;
- dimenticarsi che `%edi` e `%esi` vengono modificati da `movsl`;
- a volte, errori nel prologo, come scrivere `movl %ebp,%esp` al posto di `movl %esp, %ebp`.

Ovviamente la lista non è completa, ma vi consiglio, per prima cosa, di controllare questi possibili errori. Indipendentemente da questo, un aiuto può essere dato dal sapere qual è l'istruzione che ha causato l'errore. Per far questo basta guardare il valore di `%eip` al momento dell'errore (quello mostrato nell'output). Usate poi il seguente comando:

```
objdump --disassemble a.exe > dis.txt
```

(al posto di `a.exe` scrivete il nome dell'eseguibile che vi da errore). In questo modo otteniamo il file `dis.txt` che contiene il disassemblato dell'eseguibile con, nella prima colonna, l'indirizzo di ogni istruzione. Qui potete cercare l'istruzione che si trova all'indirizzo che era in `%eip` quando si è verificato l'errore.

Purtroppo, anche se l'accesso scorretto è stato causato al 99.9999% da un errore nel *vostro* programma (e non in Windows, o in DJGPP, o ...), l'effetto dell'errore potrebbe non essersi manifestato subito, ma potrebbe aver causato un errore molto dopo, magari durante l'invocazione di una funzione di libreria. In questo caso, l'istruzione che corrisponde ad `%eip` potrebbe non essere una tra quelle che avete scritto voi, oppure non essere proprio all'interno dell'eseguibile. In questo caso la faccenda è molto più complicata e, senza un debugger, non resta che ispezionare meglio ciò che avete scritto.

Q 3.4 *Come mai, quando è stato ridefinito il costruttore di copia di una classe* `A`*,* `gcc` *passa per riferimento anche i parametri di tipo* `A` *dichiarati per valore?*

Immaginate di avere una classe `A` e le seguenti funzioni

```
void f(A) {}    // oggetto di classe A passato per valore
A g() { A a; return a; } //oggetto di classe A restituito
```

e supponiamo di dover tradurre l'espressione:

```
f(g());
```

la chiamata a `g` deve essere eseguita prima di quella a `f`, e bisogna passare a `g` l'indirizzo a cui scrivere l'oggetto di classe `A` da lei restituito. Sarebbe complicato impostare le cose in modo che `g` lo vada a scrivere direttamente nella posizione dello stack in cui poi lo vuole `f` (bisognerebbe cominciare a preparare il frame di `f` prima di quello di `g`, che però va chiamata per prima), quindi si preferisce allocare un oggetto temporaneo (una variabile locale, non dichiarata esplicitamente dal programmatore) e passare l'indirizzo di tale variabile a `g`. Terminata la chiamata a `g`, bisogna costruire il frame di attivazione di `f`.

Ora: se non è stato ridefinito il costruttore di copia, basta copiare bit a bit la variabile temporanea nel posto giusto del frame di `f`. Ma se è stato

ridefinito il costruttore di copia? evidentemente il programmatore ha ritenuto che la copia bit a bit, per questa classe, non ha senso, quindi non possiamo prenderci la libertà di farla, in generale. Ma nemmeno possiamo chiamare il costruttore di copia, perchè la necessità di questa copia è nata da una nostra esigenza implementativa, e non era nella semantica del linguaggio. Da qui il comportamento di `gcc`: se il costruttore di copia è stato ridefinito, la copia non viene fatta, e l'oggetto viene passato a `f` per riferimento.

4 Architettura

Q 4.1 *Per quale motivo il circuito di abilitazione che deve produrre i 3 segnali di uscita (in mutua esclusione) di abilitazione per il bus a 32 o 16 o 8 bit, ha in ingresso oltre agli indirizzi di linea, anche D/C?*

Perché, in generale, vogliamo anche riconoscere il ciclo di risposta alle richieste di interruzione esterne (che è composto da due cicli di lettura agli indirizzi 0 e 4 di IO — cioè M/IO=0 — e D/C=0). Dobbiamo infatti abilitare il bus corretto, in modo che il tipo dell'interruzione possa viaggiare dal controllore di interruzione al processore. In generale, il bus corretto da abilitare potrebbe essere diverso da quello da abilitare per gli stessi indirizzi di IO, ma con D/C=1 (normali operazioni di ingresso/uscita).

5 Memoria Virtuale

Q 5.1 *In che senso la memoria fisica è mappata in memoria virtuale? Perché il bit P deve essere sempre uguale a 1?*

Probabilmente il modo in cui ciò avviene vi sfugge perchè siete troppo legati alle astrazioni (pagine che si spostano, etc.). Volendo restare nell'astratto, potete immaginare che, all'interno della memoria virtuale, ci sia una finestra (grande quanto la memoria fisica) che permette di accedere alla memoria fisica, indipendentemente da ciò che essa (memoria fisica) contiene. Questa finestra la collochiamo a partire dall'indirizzo (virtuale) 0 in modo che, per leggere (o scrivere) alla locazione di memoria fisica x, basta leggere (o scrivere) all'indirizzo virtuale x. Se la memoria fisica è, ad es., di 512 M, questa finestra finisce all'indirizzo virtuale $2^{29} - 1$. Questa finestra non viene gestita come la normale memoria virtuale: è solo un modo (normalmente riservato al codice di sistema) per accedere a tutta la memoria fisica. Possiamo pensare che la normale memoria virtuale inizi subito dopo la fine di questa finestra (quindi, nell'esempio di prima, dall'indirizzo 2^{29} fino all'indirizzo $2^{32} - 1$).

Come viene creata questa finestra? qui conviene che abbandoniate le astrazioni e pensiate semplicemente a cosa sono (collettivamente) il direttorio più le tabelle delle pagine: una tabella che trasforma *ogni* indirizzo generato dal processore, prima che questo indirizzo arrivi alla memoria (quella fisica, l'unica che esiste davvero). Il bit P, che è associato ad ogni "pagina" di indirizzi virtuali (una pagina, in questo contesto, è solo un insieme contiguo di indirizzi virtuali) aggiunge un meccanismo particolare alla trasformazione: se, per un certo indirizzo, P=0, la MMU non effettua la trasformazione ma, invece, solleva una eccezione. Questa trasformazione può essere usata per vari scopi, la memoria virtuale è solo uno di questi (anche se è il più significativo). Nella memoria virtuale il bit P ci serve perchè non tutti gli indirizzi sono sempre validi (appunto, sono virtuali). La nostra finestra, invece, è una applicazione particolare di questa trasformazione: in pratica, una trasformazione "identità", che lascia gli indirizzi così come erano. Per questi indirizzi il bit P non ci serve. Lo poniamo sempre a 1 perchè non vogliamo che vengano generate eccezioni quando il processore accede agli indirizzi dentro questa finestra.

A questo punto, vi resta da capire che queste due trasformazioni "convivono" nella stessa tabella: nelle prime entrate ci sono le trasformazioni (identità) relative alla finestra, con bit P sempre = 1, nelle rimanenti le trasformazioni relative alla memoria virtuale, alcune con bit P=0, altre con bit P=1, a seconda di quali pagine della memoria virtuale sono "presenti in ram".

Notate che una pagina virtuale di indirizzo virtuale V, presente in RAM nella pagina di memoria fisica di indirizzo fisico F, sarà accessibile al processore da *due* indirizzi virtuali: V (tramite la memoria virtuale) e F (tramite la finestra).

Q 5.2 *Nel libro si dice che, poiché quando una tabella delle pagine viene scelta per essere rimpiazzata, non contiene descrittori significativi, non viene salvata su disco. Però, da un'altra parte, si dice che i descrittori con P=0 possono*

essere usati per memorizzare la posizione su disco della pagina corrispondente. Ma, allora, se rimpiazzo una tabella delle pagine, perdo queste informazioni?

È chiaro che le due ottimizzazioni non possono essere usate contemporaneamente. Quindi: o non usiamo i descrittori con P=0 (prevedendo una struttura dati a parte per ricordare l'associazione tra pagine virtuali e indirizzi su disco), oppure, se vogliamo usarla, abbiamo due possibilità:

1. il contenuto della tabella viene ricopiato in memoria di massa, come per le normali pagine
2. le tabelle delle pagine non vengono mai rimpiazzate (come in Linux).

6 Nucleo

Q 6.1 *Che differenza c'è tra un programma e un processo?*

Nessuno mi ha fatto esplicitamente questa domanda, ma vedo che c'è molta confusione su questo punto fondamentale.

Proviamo ad illustrare la differenza con una semplice metafora: in una pizzeria, il pizzaiolo riceve una ordinazione. Il pizzaiolo è inesperto e ha bisogno di avere la ricetta della pizza davanti a se. Stende la pasta, la condisce, la inforna, aspetta che sia cotta, la farcisce e serve la pizza.

Cos'è il programma?

Sono sicuro che nessuno di voi ha dubbi: il programma è la ricetta.

Cos'è il processo?

Qui credo che molti risponderanno: il pizzaiolo. No. Il pizzaiolo è il processore, cioè l'entità che interpreta la ricetta e la esegue.

Allora è la pizza? No. La pizza sono i dati da elaborare.

Il processo è un concetto un po' più astratto. È la *sequenza di stati* che il sistema "pizza + pizzaiolo" attraversa, passando dalla pasta alla pizza finale, secondo le istruzioni dettate dalla ricetta, eseguite pedissequamente dal pizzaiolo inesperto.

Uscendo dalla metafora, un processo è un programma in esecuzione su dei dati di ingresso. Questa esecuzione la possiamo modellare come la sequenza degli stati attraverso cui il sistema processore + memoria passa eseguendo il programma, su quei dai, dall'inizio fino alla conclusione. Notate che questa definizione si applica bene ai programmi di tipo *batch*, in cui gli ingressi vengono specificati tutti all'inizio, e il processo (generato dal programma in esecuzione su quei dati) prosegue indisturbato fino ad ottenere le uscite (ad esempio, pensate ad un programma per ordinare alfabeticamente un file). Una volta afferrato il concetto, però, in questo caso semplice, credo che non vi sarà difficile estenderlo ai programmi "interattivi" a cui ormai siete più abituati (praticamente tutti i programmi con interfaccia grafica, ma non solo quelli).

A prima vista, il processo potrebbe sembrare molto simile al programma: la ricetta dice "stendere la pasta" e nel processo vediamo il pizzaiolo stendere la pasta; nel rigo successivo la ricetta ricetta dice "versare il condimento" e nel processo vediamo, subito dopo, il pizzaiolo versare il condimento. È molto semplice confondere i due concetti, soprattutto se il programma è molto semplice, ma si tratta di due cose completamente distinte, per i seguenti motivi:

- uno stesso programma può essere associato a più processi: tanti clienti, in genere, chiedono lo stesso tipo di pizza. In questo caso, il programma è sempre lo stesso (la ricetta per quel tipo di pizza), ma ad ogni pizza corrisponde un processo distinto, che si svolge autonomamente nel tempo.

- in generale, non è esclusivamente il programma (la ricetta) a decidere attraverso quali stati il processo dovrà passare, ma anche la richiesta del cliente(l'input): la ricetta potrebbe, infatti, prevedere delle varianti (un `if`), che il cliente dovrà specificare. La ricetta conterrà istruzioni per

entrambe le varianti (con o senza carciofi), ma un particolare processo seguirà necessariamente *una sola* variante.

Ma c'è dell'altro, nella metafora del pizzaiolo, che ci può aiutare a capire altri punti fondamentali. Il processo, se lo guardiamo nella sua interezza, si svolge necessariamente nel tempo. Possiamo anche, però, guardarlo ad un certo istante, facendone una fotografia. La fotografia che ne facciamo deve contenere tutte le informazioni necessarie a capire come il processo si svolgerà nel seguito. Nel nostro esempio, la foto dovrà contenere:

- la pizza nel suo stato semilavorato (la memoria dati del processo);
- il punto, sulla ricetta, a cui il pizzaiolo è arrivato (il contatore di programma);
- tutte le altre informazioni che permettono al pizzaiolo di proseguire (da quel punto in poi) con la corretta esecuzione della ricetta, come, ad esempio, il tempo trascorso da quando ha infornato la pizza (i registri del processore).

Se la fotografia è fatta bene, contiene tutto il necessario per sospendere il processo e riprenderlo in un secondo tempo. Il pizzaiolo fa questa operazione continuamente, quando condisce, a turno, più pizze, o quando lascia una serie di pizze nel forno e, nel frattempo, comincia a prepararne un'altra (multiprogrammazione).