Architettura degli Elaboratori

Federico Matteoni

Indice

1 Introduzione **2**

2 Cosa riguarda il corso **2**

3 Struttura a livelli **2**
3.1 Macchine Virtuali . . . 2
3.2 Compilazione vs Interpretazione . . . 3

4 Assembler D-RISC **4**

5 Reti Combinatorie **4**
5.1 Algebra Booleana . . . 4
5.1.1 AND . . . 4
5.1.2 OR . . . 4
5.1.3 NOT . . . 4
5.2 Tecnica della Somma di Prodotti, o codifica degli 1 . . . 5

6 26-09-2019 **5**

7 moduli operativi/unità funzionali **5**

1 Introduzione

Appunti del corso di **Architettura degli Elaboratori** presi a lezione da **Federico Matteoni**.

Prof.: **Maurizio Bonuccelli**, maurizio.angelo.bonuccelli@unipi.it
Riferimenti web:

- *http://pages.di.unipi.it/bonuccelli/aeb.html*
- *didawiki.cli.di.unipi.it/doku.php/informatica/ae/start*

Ricevimento: Martedì 10-12, stanza 294 DE
Esame: **scritto** (*closed book*) e **orale**. I compitini sono validi solo per la sessione invernale (gen-feb)
Libri

- M. Vanneschi *Architettura degli Elaboratori*, Pisa University Press
- D. A. Patterson *Computer Organization & Design - The Hardware/Software Interface*

2 Cosa riguarda il corso

Consiste in come sono fatti pc internamento da un punto di vista di sottosistemi senza scendere nei dettagli elettrici.
Il corso è diviso in quattro parti:

- Fondamenti e strutturazione firmware (I Compitino)
- Macchina assembler (D-RISC) e processi
- Architetture General-Purpose
- Architetture parallele (II Compitino)

3 Struttura a livelli

Quando voglio costruire qualcosa di complesso lo faccio a pezzi, partendo da comp elementari messe insieme o studiate ad altro livello, messe ulteriormente insieme ecc.
Ogni livello lo chiameremo **macchina virtuale** o MV, seguito da un numero che indica il numero di livello.
Due approcci fondamentali:

- **Linguistico**: stabilisce i livelli in base ai linguaggi usati
- **Funzionale**: stabilisce i livelli in base a cosa fanno

3.1 Macchine Virtuali

[disegno]
MV_i realizza politica P_i con linguaggio L_i e risorse R_i.
Utilizza le funzionalità che il livello MV_{i-1} (primitive) fornisce attraverso l'interfaccia

Supporto a tempo di esecuzione o **Runtime Support**: insieme dei livelli sottostanti. Nell'esempio, MVi ha come runtime support i livelli MVi-1 ... MV0. Una macchina virtuale è modulare perché devo poterla modificare, deve essere portabile (riutilizzabile in più contesti possibili).

> **MV_4** Applicazioni
> L_4: Java, C
> R_4: costrutti
>
> *Interfaccia*: chiamate di sistema
>
> **MV_3** Sistema Operativo
> L_3: C
> R_3: variabili condivise, risorse condivise
>
> *Interfaccia*: istruzioni assembler

$\mathbf{MV}_2$ Macchina assembler
L_2: assembler (D-RISC)
R_2: registri, memoria, canali di comunicazione

Interfaccia: istruzioni firmware per l'assembler

$\mathbf{MV}_1$ Firmware
L_1: microlinguaggio
R_1: sommatore, commutatore

Interfaccia: hardware

$\mathbf{MV}_0$ Hardware
L_0: *funzionamento dei circuiti elettronici*
R_0: circuiti elettronici elemntari (AND, OR, NOT)

Il corso riguarderà principalmente i livelli $MV_2 \rightarrow MV_0$ incluse, comprese le istruzioni assembler.
Il livello firmware sarà fatto da **memoria**, **processore** e **dispositivi I/O**. I/O comunica bilaterale con memoria e Processore comunica bilaterale con memoria. Opzionalmente I/O comunica bilaterale direttamente con processore. Questa è l'architettura standard in maniera estremamente semplicistica. Vedremo processore e memoria, non i dispositivi I/O perché troppo complessi.

3.2 Compilazione vs Interpretazione

Compilatore: è **statico**, vedendo tutto il codice può ottimizzarlo. Sostanzialmente è l'opera di un traduttore, che può leggersi il testo più volte per tradurlo alla perfezione.

Interprete: è **dinamico**, quindi non può ottimizzare. Il firmware riceve un'istruzione alla volta quindi la interpreta.

Entrambe servono per tradurre il **codice sorgente** nel **programma oggetto** o **eseguibile**.

Suppongo programmi:

```
   for i=0; i++; i<n
A[i] = A[i] + B[i];
```

```
   for i=0; i++; i<n
B[i] = B[i] + C;
```

Ricevendo i due blocchi di istruzioni, il compilatore riconosce che sono diverse e le compila in modo diverso. Però in entrambi i casi sono del tipo *oggetto = somma due oggetti*, quindi produce una sequenza di istruzioni analoga (a meno di registri e dati, ovviamente).

Parte del secondo pezzo di codice, ad esempio, verrà tradotto in questa maniera:

Microlinguaggio corrispondente
`LOAD` R_{base}`,` R_I`,` R_1	`M[R[base] + R[I]]` $\rightarrow$ `R[1]`
`ADD` R_1`,` R_2`,` R_1	`R[1] + R[2]` $\rightarrow$ `R[1]`
`STORE` R_{base}`,` R_I`,` R_1	`R[1]` $\rightarrow$ `M[R[base] + R[I]]`
`INC` R_I	`R[I] + 1` $\rightarrow$ `R[I]`
`IF<` R_I`,` R_N`, LOOP`	

4 Assembler D-RISC

Istruzioni lunghe 32bit, primi 8bit per identificativo istruzione. Poi tre blocchi di 6Bit (R_i, R_j, R_h, in ogni blocco vi è mem semplicemente l'indice i, j o h). Poi 6 bit tipicamente inutilizzati (per estensioni future, istruzioni particolare e per riempire le locaz. di mem che sono tutte a 32 bit).
$2^6 = 64$ registri generali nel processore
Ad esempio `ADD` R_i`,` R_j`,` R_h significa `M[R[i] + R[j]]` $\rightarrow$ `R[h]`, e `ADD` è memorizzato con un determinato codice identificativo.
Per l'inizializzazione, ho il registro R_0 che contiene sempre 0.

Esempio di RTS `MV3 C = A + B`
Su MV_2 diventa `ADD` R_A`,` R_B`,` R_C
Su MV_1 ho registro A, registro B verso addizionatore/sottrattore (con alfa che indica operazione) e porta in C (con beta che indica scrittura attiva o meno)
Su MV_0 i vari componenti sono costruiti da una serie di gate (`AND`, `OR`, `NOT`).

PO Parte Operativa
PC Parte Controllo
`roba eventuale`

5 Reti Combinatorie

In una rete combinatoria si ha una serie di segnali in input (X_1 ... X_n) che vengono trasformati in una serie di segnali in output (Y_1 ... Y_m). A seconda delle varie componenti presenti sulla rete combinatoria, un insieme di segnali 0/1 viene trasformato in un altro insieme di segnali 0/1 seguendo le regole dell'**algebra booleana**.
Elettricamente, quando un segnale vale 1 significa che la tensione è circa 5V.

5.1 Algebra Booleana

L'algebra booleana è computata su **due valori** e **tre operatori**:

`false`	`AND`
`true`	`OR`
	`NOT`

Esistono anche altri operatori, derivati dai tre precedenti: `XOR`, `NAND`, `NOR` ecc..

Proprietà Vale la proprietà distributiva anche per la somma rispetto alla moltiplicazione, oltre il viceversa, quindi: `A(B+C) = AB + AC`, ma anche `A + BC = (A + B)(A + C)`.
Inoltre si hanno le cosiddette **proprietà di DeMorgan**:

- $\overline{A+B}$ `=` $\overline{A}$ `*` $\overline{B}$
- $\overline{AB}$ `=` $\overline{A}$ `+` $\overline{B}$

5.1.1 AND

Anche detta **moltiplicazione logica**.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

5.1.2 OR

Anche detta **somma logica**.

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

5.1.3 NOT

Anche detta **negazione logica**.

Y	Z
0	1
1	0

Per costruire una **rete combinatoria** esistono varie tecniche. Quella che useremo si chiama **somma di prodotti**.

5.2 Tecnica della Somma di Prodotti, o codifica degli 1

La tecnica nel dettaglio Partendo dalla **tabella di verità**, identifico le uscite che valgono 1. Di quelle uscite, **moltiplico (`AND`)** tra loro le entrate **sulla stessa riga**, **nego le entrate che valgono 0** e **sommo (`OR`) tra loro le diverse righe**.

Un esempio con la somma algebrica Partendo dalla seguente tabella di verità.

X	Y	Z	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sfruttando la tecnica descritta sopra ottengo le seguenti espressioni per le due uscite:

`Z =` $\overline{X}$ `* Y + X *` $\overline{Y}$

`R = X * Y`

Alternativamente, posso anche realizzare la **funzione complementare**, ovver fare il solito procedimento ma per le uscite che valgono 0 per poi negarle.

X	Y	$\overline{Z}$	R
0	0	1	0
0	1	0	0
1	0	0	0
1	1	1	1

`Z =` $\overline{\overline{X} * \overline{Y} + X * Y}$

`R = X * Y`

6 26-09-2019

S1	S2	X	Y	S1*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

s1≐ ns1*ns2*x*y + ns1*s2*x*y + s1*ns2*nx*ny + s1*ns2*x*ny + s1*ns2*x*y + s1*s2*nx*y
+ s1*s2*x*ny + s1*s2*x*y
mappa di carnaut
00 01 11 10, così che tra due colonne cambi un solo bit
prendere multipli di due "uni", cioè 2/4/8... uni
Gli estremi sono logicamente collegati (colonna 00 e colonna 10 sono adiacenti quindi posso formare rettangoli anche tra loro)

7 moduli operativi/unità funzionali

Parte operativa: produce l'output
Parte controllo: dice alla PO *come controllare* i suoi componenti (es. produe gli alfa (dicono cosa fare ai componenti) e i beta (quali registri in scrittura e quali no)). La PO porta alla PC le **variabili di condizionamento**, che istruiscono

la PC su *come* produrre alfa e beta.
PO e PC sono reti sequenziali.
In generale le mealy sono migliori: mediamente costano non di più e sono non più lente.
in mealy x va anche in omega.

1. PO –var condiz–¿ PC
2. PC –alfa, beta–¿PO
3. PO –output Z–¿ *fuori*

PO la faccio M-
PC la faccio M-
PO moore (ma non sarà automa), PC mealy

Funzionalmente Mealy e Moore sono **equivalenti**. Moore rispodne dopo un clock, mealy risponde subito.
var condiz: info che PO passa alla PC affincché P generi alfa e beta.

microlinguaggio uao