Elettronica

Roncella
e-mail: roberto.roncella@unipi.it
Ricevimento: giovedì pomeriggio o venerdì
050 2217669

25.09.2015

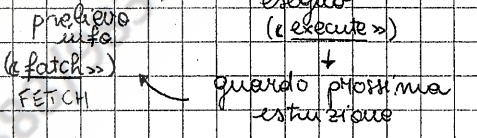
MICROCONTROLLORE

Impiegato in moltissimi apparecchi (ECG, tagliaerba..)
Passaggi per programmare: Scrittura programma → compilatore → macchina
Col linguaggio ASSEMBLY si riporta «pari pari» l'istruzione: ad ogni istruzione corrisponde un comando per il sistema digitale.

Il microcontrollore è un CIRCUITO INTEGRATO; è dato da una scodellina ($1,5 \times 1,5\ cm^2$) contenente una scheggia di silicio. Ve ne sono di vari tipi e vari prezzi.

COMPONENTI:
- CPU («core»): è il cervello, l'intelligenza del blocco. Vi vengono gestite le operazioni e il flusso di esecuzione, la sequenza.
  è il µProcessore
  ha anche piccoli contenitori, veloci, per avere "dati alla mano"
  ha { parte di controllo ⇒ eseguono istruzioni
  " operativa





- ALU: fa operazioni (somma, prod..) secondo logica booleana
- PERIFERICHE / CIRCUITI AUSILIARI: tantissime tipologie
- Contenitori di informazioni, ROM e RAM
  cioè MEMORIE

RAM: «Random access memory»; "casuale" cioè posso scegliere dove leggere, cancellare, scrivere.

ROM: «Read only memory»
contenitore di programma in sola lettura; vi si scrive all'inizio del programma sviluppato e cui si hanno cambi occasionali

- Firmware: programma in ROM, non modificabile; fa da "ponte" tra componenti logiche e fisiche, ovvero:
- Software: la sua esecuz. è resa possibile da hardware
- Hardware: parte fisica, oggetto "duro"

Storicamente

1) ARCHITETTURA VON NEUMANN (anni '70)

Siccome fili per collegare le componenti erano dispendiosi e la memoria era esterna, unica memoria (parte non volatile + parte modificabile) collegata al resto con un unico BUS (canale di comunicazione tra più dispositivi); esso trasporta sia istruzioni che dati elaborandi / elaborati.

Limiti: Rischio di sovrapposizioni tra dati e istruzioni; rischio di creare programmi che si automodificano

Unico BUS: collo di bottiglia e se prelevo istruzioni, non posso al contempo scrivere dati.

2) ARCHITETTURA HARVARD (anni '80)

Idea chiave: dispositivi digitali non solo per fare i conti, ma anche per acquisire dati una volta ottenuti, contemporaneamente. Fisab e acquisire tutto in tempo reale

Per velocizzare, si FRAZIONA

- mem. dati (RAM): accesso diretto ad ALU
- mem. istruzioni: • contenuto almeno immodificato • può essere realizzata con memoria non volatile

⇓

miglior efficienza, evitando colli di bottiglia.

Possibili ulteriori miglioramenti lavorando ulteriormente le fasi "fetch" ed "execute" o inserendo più bus

Pipeline ⇒ tecnologia per aumentare quantità di istruzioni eseguite in una data quantità di tempo per eseguire operazioni in parallelo (es: una esegue operazione mentre il sistema di controllo decodifica istruzione successiva)

per questo sono necessari anche REGISTRI INTERMEDI, ad esempio per i riporti nella somma (FLAG)

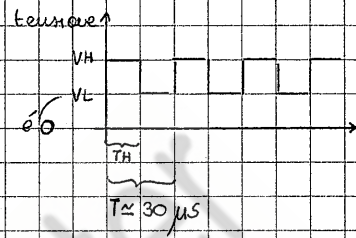
2 tipi di architetture: { RISC (Reduced Instruction Set Computer) → ridotto numero di istruzioni, effettuate in numero ridotto di microcicli (sequenziatore veloce e semplice)
CISC (Complex Instruction Set Computer) → istruzioni di elevata capacità, effettuate in elevato numero di microcicli (sequenziatore e ROM complessi)

→ PERIFERICHE di un μC

- Sottocircuiti dedicati ad applicazioni specifiche
- Interfaccia (= x far dialogare una componente con le altre o col μP) tra periferica e μP: inizialmente era specifica (dialogo apposito ∀ periferica), ma oggi sarebbe impossibile, a causa di troppe periferiche esistenti.
  Soluzione: ogni periferica finge di essere parte della MEMORIA; così ogni periferica può essere configurata, letta e trasmessa dal μP
  CPU ha ≃ 2^{16} contenitori. Si possono, ad es. dedicare tutti i registri da 0 → 0x2000 alle periferiche.
- Chip = componente elettronico su cui è presente il circuito integrato; esso comunica mediante i pin, piedini da cui è composto

- PIN (<<piedini>>) gestibile come fonte di info (in ingresso) o uscita (→ bassa resistenza)
  - alta resistenza
  - Interruzione sul livello o sulla variazione
  - Cioé controllo individuale dell'impedenza

- TIMER e CONTATORI / COUNT: • Conteggio di eventi (misurare di tempo e frequenza)
  - eventi periodici con frequenza nota / non nota
  - • Generazione di segnali periodici / con ciclo di lavoro variabile (cioé grandezze variabili continuamente) (PWM)
    - es: luci di Natale
    - onde rettangolari note:



$\frac{T_H}{T} = \delta$ = «duty cycle»

cambiando δ, cambio intensità media

- CONVERTITORI A/D (VOLTMETRI): leggono tensione su pin e la trasformano, traducono, in un valore digitale in un registro.
- CONVERTITORI D/A: generano tensione analogica su PIN di uscita corrispondente a valore su registro
- INTERFACCE di COMUNICAZIONE STANDARD: usb, can → c'é «assunzione» tra client (USB) e server (PC)
  - × far dialogare strumento con altro apparecchio. Si stabilisce un livello fisico, logico, elettrico → Standard stabiliti da aziende (Bluetooth), da stati (wi fi)
  - livello fisico, logico, elettrico
  - linguaggio

→ «Polling»: tecnica per reagire a periferica; controllo ciclico di ogni periferica, che é molto utile per una continua sorveglianza anche di eventi straordinari, ma molto oneroso in quanto gli eventi possono essere SPORADICI: inefficiente
- l'azione di interrogare

Soluzione: nel caso di un problema, la periferica stessa richiede interruzione del programma, della normale esecuzione. Solo a quel punto avviene l'«interrogazione» e poi si riprende con l'usuale corso
- esempio tale metodo può essere abilitato o meno

→ CIRCUITI AUSILIARI

- Clock: presupponiamo tutto già sincronizzato
- Supervisore di alimentazione: a pile o a corrente, l'importante é che sia valida, altrimenti meglio non eseguire (si pensi ad una presa)
  - Circuito di reset
  - Supervisore della modalità operative (iniziale / sleep): per le applicazioni portatili (indossabili/impiantabili) é necessaria un'attivazione selettiva per l'efficienza energetica
- Watch dog timer: Se un apparecchio non va, noi OFF → ON; ciò può avvenire anche «automaticamente» mediante un contatore con auto-incremento che ha un suo clock ed é legato al resto solo da fili di reset reciproci

Se il contatore raggiunge un valore particolare (x), si resetta il sistema: dunque il sistema deve, prima che si raggiunga x, resettare il watch dog timer, azzerando il punteggio.
Se sistema si intoppa, incoppa, non resetta il timer prima di x e dunque watch dog procede con reset del programma.

Studieremo Microcontrollore della ATMEL (fusa su con Dialog, Rivoira; www.Atmel.com)

- gruppo x-mega, 256A3BU (memoria); fascia entry level, fino a 32 megaHz
- 8 bit, AVR (numeri binari a 8 cifre); 32 registri
- Architettura Harvard, con ROM e RAM interne e separate (8 bit; 16 bit = $2^{16}-1$ info)
- Per il "Polling" ha 3 livelli di abilitazione.

NB • convers binario - decimale
• $0x2000$ numero esadecimale, 2×16^3

Esadec -> binario 9 A B .. F
15 = 1111
Quindi FF = 1111 1111

28 09 2015:

Di x-mega si può lavorare "in parallelo" con coppie di registri, quindi 16 bit: da 0 .. $2^{16}-1$. Anche in questo caso, "compl a 2"

Il proc lavora in virgola fissa (no mobile)

→ Famiglie di µC: Chip con stesso µP e diverse periferiche

28 09 2015

LINGUAGGIO ASSEMBLY

→ È lo strumento di programmazione più vicino alla realtà fisica-elettronica di un elaboratore (più vicino ci sono solo i bit): diretta corrispondenza tra elem. del linguaggio ed elem. della architettura del processore.
Traduzione da parte di ASSEMBLATORI (vs compilatori) che traducono in modo diretto i dati e li inseriscono nella memoria di programma dell'elaboratore per essere eseguiti.

Quindi A) Lunghezza codice assembler direttamente proporzionale a spazio occupato in memoria.

B) Ogni operazione ha un tempo preciso di esecuzione → posso conoscere così il tempo necessario per eseguire il task: ciò è permesso dall'assenza di passaggio intermedio di compilatore che, traducendo, tenta di ottimizzare tempi; ciò avviene in macchine di fascia alta, con programmazione ad alto livello in cui esistono apposite strategie per ottenerlo.

Programma ok se tempo e memoria ridotti

→ PROGRAMMA ASSEMBLY: lista di istruzioni contenenti elementi di vario tipo; ogni istruzione ha una lunghezza nota ed occupera un preciso indirizzo nella memoria di programma.

- Linee (che non iniziano con .) hanno 2 colonne + commento (introdotto da ;**)

  I^a: istruzioni verbi, individuate con "codice mnemonico" (Idi = «load immediate», eor = «ebxor»)

  II^a: "operandi" su cui agire (costanti / registri: x Atmel, l'addizione in Ave opera sono tra registri, da R0 a R31: OUT1 ed OUT0 sono 2 registri); non sono sempre presenti (es: RET non ha operandi)

  Talvolta a sx è presente anche una III^a colonna, contenente «LABEL»; cioè etichette (MULT, M1, M2): in termini fisici sono elementi indirizzi della memoria del programma, in sostituzione all'indirizzo «numerico»: sono importanti sia per orientarsi che per far aggiunte al programma anche successivamente alla scrittura nel primo tempo.
  Tali etichette si trovano anche tra operandi, per eseguire "salti"

- Linee che iniziano con . (es: .org 100): contengono «direttive» non per µProc ma per assemblatore. Es:
  - .org 100 (= origin): le seguenti istruzioni vanno messe in memoria a partire dall'indirizzo 100
  - .equ: x costante (.equ "nome" = "valore")
  - .def: per dare nome simbolico ad elem. del processore. (.def "pippo" = R2)

Ogni macchina ha «sfumature» personali per il suo assembly, caratterizzabili nel suo "instruction set".

«Problema dell'indirizzamento»: come una istruzione recupera i propri operandi

** su più righe, /* ... */