Ingegneria Elettronica

Appunti di

S.E.P.

(appunti del corso del Prof. S. Di Pascoli)

a cura di Simone H.

master COPY COPISTERIA

master COPY







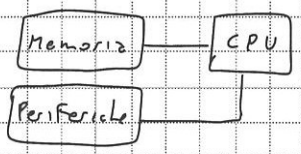
Simone H.
Appunti di S.E.P.
© Simone H., 2019
Tutti i diritti riservati

-computer: rete logica pensata a fare un elaborazione generica sui dati
-microprocessore: è il cuore di un computer, è l'implementazione fisica più comune della CPU
-microcontrollore: periferiche e memoria stanno in un singolo chip
-SOC (System On Chip): sinonimo di microcontrollore ma cambia l'applicazione; è più potente di un microprocessore e di un microcontrollore; è dotato di MMU(memoria virtuale)
-DSP (Digital Signal Processor): è un microcontrollore specializzato per elaborazione numerica dei segnali in tempo reale

come si sceglie un microcontrollore?
-classificazione: la potenza di calcolo cresce in maniera radicale con il numero dei bit
->8bit: prestazioni inferiori ma basso costo e più facile da usare; problema principale è fare l'aritmetica->devo spezzare i calcoli in più op. (in assembler). se i dati sono segnali allora in questo caso c'è una dinamica insufficiente
->16bit: è la scelta migliore ma si trova poco nel mercato
->32bit: dal punto di vista aritmetico è anche troppo 32bit(4miliardi circa); avanza spazio(spreco); a volte lo stesso programma è più efficiente in un 16bit rispetto al 32bit perchè si dice che in quest'ultimo c'è una cattiva "code density"
-storia: lo Z80(8bit) aveva circa 10mila transistor, il primo ARMV1 ne aveva 32mila(32bit); con il passare degli anni il numero dei bit che si utilizzava è diventato inefficiente e quindi ora siamo passati a 64bit. es alcuni 8bit vecchi ancora in uso: 8051, 8080, 6502. sono stati progettati per essere programmati in assemble e a mano, quindi si prestano male a essere programmati in C->sconsigliati per i progetti nuovi. esiste un compilatore in C in grado di gestire questi processori: SDCC(small device C compiler) ma non usa il C classico ma modificato perchè questi processori hanno una RAM piccolissima
-compilatore
->GCC: compilatore per il C gratis (di qualità)
->IAR: produce compilatori specializzati per micro(3000 euro); non ne vale la pena
-compatibilità con GCC
->8bit: AVR, AVRXMEGA, PIC(ha un compilatore strano)
->16bit: MSP430
->32bit: ARM, AVR32
-riassumendo: (per la scelta)
1)C o assembler
2)proprietario o opensource
3)normale o strano(per normale si intende simile al processore del PDP11) ||il pic ha un compilatore C strano
4)CISC(complex instrument set computer) o RISC(reduce instrument set computer) ||è più conveniente avere meno istruzioni semplici(RISC)
-RISC: le istruzioni devono essere eseguite in un ciclo di ck(quasi tutte le istruzioni dell'AVR). AVR e AVRXMEGA vengono eseguite in 1/2 cicli di ck
-CISC: (Z80) usa di 6 ai 23 cicli di ck
-processori
->AVR (64)
->ARM64: svantaggio: è troppo difficile per il principiante, quindi noi si parte con l'AVR che è più compatto. non si usa l'AVRxMEGA perchè anche questo è difficile

Architettura AVR

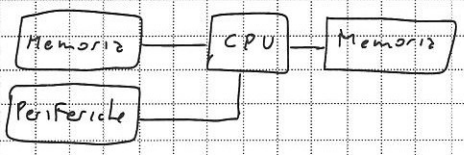
Von Neumann



-> Si possono incontrare 3 tipi di difficoltà:
1) CPU intensive → la CPU non è abbastanza potente per fare altri calcoli
2) Memory intensive: es. semplici su tanti dati: → la CPU ce la farebbe ma ha bisogno di più memoria
3) I/O intensive: trasferimento dati lento dall'ext

-l'architettura Von Neumann è buona per applicaz. general che di solito hanno prob. del tipo 1 e 2) → s/z ⇒

⇒ Architettura HARWARD: ha 2,3 memorie (non più di 3)



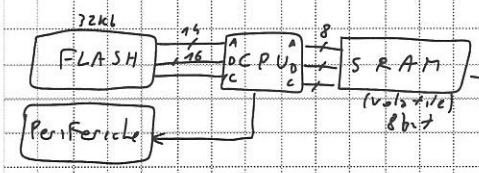
→ tipicamente una memoria contiene il codice e l'altra i dati (per caricarle contempor.)

-Inizialmente è stata inventata per i DSP e poi estesa agli AVR perché è utile avere più memorie fatte in maniera ≠ (volatile e non)

-Nei μC si ha l'esigenza di una memoria non volatile e anche volatile se fanno continuamente ricevere e mandare dati

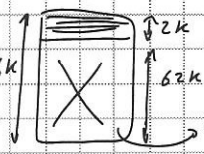
-Le 2 memorie sono ≠ perché sono fatte con due tipologie di tec

AVR (8bit)



→ con questo tipo di architettura si può comunicare con 2 memorie contemporaneam.

→ È a 8bit, le singole celle di memoria hanno capacità 8bit per cui di conseguenza hanno capacità di 8bit mentre quello per gli indirizzi (2k indirizzi) ha 16bit che permettono di indirizzare 64kb



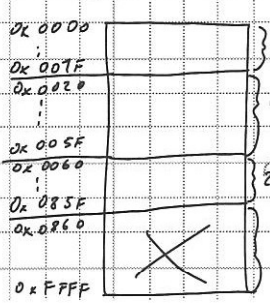
→ La Flash contiene le istruzioni ed è organizzata in 16k celle da 16bit (WORD)

→ Il BUS indirizzi è di 14bit $16 \cdot 2^{10} = 2^{14}$ (2048) sono le locazioni di memoria interna della SRAM

32kb = 16K × 16bit (celle)

→ la prima cella della FLASH ha indirizzo 0x0000 fino a 0x3FFF (16383 indirizzi)

→ la mappa di memoria della SRAM è più complessa



32 celle, le prime 32 celle contengono una copia di 32 registri (sono mappati in memoria)

64 celle per le periferiche

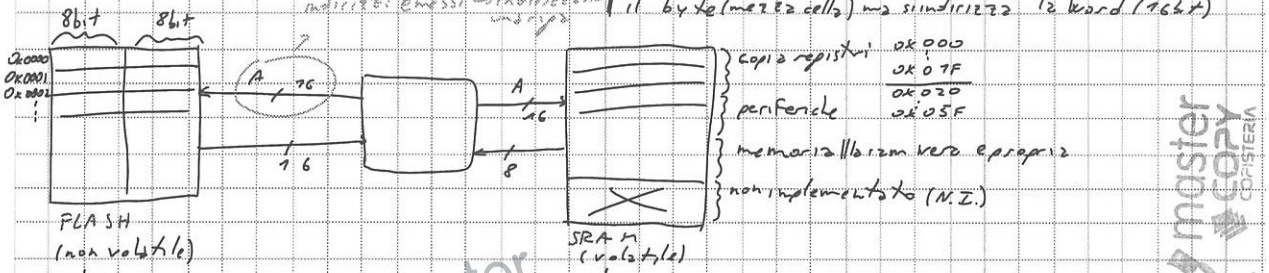
2048

2kb di memoria vera per i dati (RAM)

spazio non implementato, se ci scrivo sopra perdo i dati. Quando leggo mi viene sempre FF (tutti "1")

1/10/18

ATMEGA32



per il codice o dati permanenti (anche dati che si aggiornano raramente)

dati che cambiano molto frequentemente

FLASH → contiene il codice (le istruzioni 16bit) → la memoria è larga 16bit divisa in due rettangoli da 8bit

- Gli indirizzi indirizzano una riga da 16bit (una locazione da 16bit) → è strano perché di solito ci si riferisce ad una cella, qui no.

SRAM → organizzata a 8bit

Processore a 8bit → si chiama così perché il nome dipende dalla dimensione dei registri e della ALU che sono entrambi a 8bit
└→ organo che fa le op.
→ Se sommo num a 16bit devo spezzare le op. in due

Come scrivo in memoria? le celle di memoria hanno un indirizzo a 16bit
→ Se un processore RISK → ogni istruzione occupa una cella di memoria → le istruzioni normalmente sono grandi come il nome del processore (8bit); nel nostro caso non è vero:
→ Il processore è a 8bit ma le istruzioni sono grandi 16bit

→ Se un istruzione ha bisogno di avere l'indirizzo, questo deve essere messo dentro l'istruzione ma se l'istruzione è grande 16bit non posso mettere l'indirizzo di 16bit dentro l'istruzione → non mi resterebbe niente per descrivere il comando

→ Ogni istruzione ha: una descrizione di cosa deve fare il processore (codice operativo) e la descrizione degli operandi (materie prime sulle quali fare le op.) (indirizzo)

→ Se operando sta in memoria e l'indirizzo dell'operando è l'indirizzo di memoria ed è grande come l'istruzione → non è possibile mettere l'indirizzo dentro l'istruzione perché non c'è posto → S/Z → il nostro processore è un RISK con eccezioni ⇒ nel caso abbia bisogno di un indirizzo a 16bit, questo si mette in una seconda cella (violazione del principio del RISK) → l'istruzione occupa due celle: 1° → codice operativo e la 2° l'indirizzo

codice operativo
indirizzo

→ Se voglio accedere alla memoria ho bisogno di specificare l'indirizzo a 16bit → ci sono due istruzioni apposite:
LDS e STS