01 - STRUTTURA DI UN ELABORATORE

COMPONENTI E ORGANIZZAZIONE

PROGAMMAZIONE HARDWARE/SOFTWARE

MODI DI IMPLEMENTARE UN PROGRAMMA:

1. in *HARDWARE*
   Un programma è un opportuno circuito costruito *specificamente* per un determinato compito.

   PRO:

   - Permette di implementare un circuito di area minima
   - Esecuzione veloce
   - Risparmio energetico

   CONTRO:

   - Non può essere modificato
   - Costo di implementazione elevato
   - Approccio specifico e non generale
2. in *SOFTWARE*
   Una famiglia di funzioni logiche e aritmetiche è implementata in hardware.
   Dei segnali di controllo indicano quali funzioni attivare e in che ordine
   PRO:

   - Il programma *può* essere modificato
   - Costo di implementazione ridotto

- Universalità

CONTRO:

- Minor efficienza di calcolo
- Circuiti più estesi
- Maggior consumo energetico

STRUTTURA DI UN ELABORATORE ELEMENTARE

ELABORATORE: Sistema per la programmazione in software.

- Si assegna un *codice* univoco a ogni possibile sequenza di segnali,
- Si aggiunge una componente hardware per tradurre un codice in *segnali*,
- Ogni codice denota *un'istruzione* che determina in modo univoco i segnali da attivare,
- Un *programma* è definito dalla sequenza di istruzioni/codici (*software*).

CPU (CENTRAL PROCESSING UNIT)

Deve essere costituita da:

- **EXECUTION UNIT**
  Interpreta ed esegue le istruzioni, è suddivisa in
  - **ALU** (Arithmetic and Logic Unit)
  - **CU** (Control Unit)
- **Registri di memoria**
  La CPU legge e scrive dati/istruzioni in memoria tramite due registri:
  - **MAR** Memory Address Register, trasmette gli indirizzi delle locazioni di memoria,
  - **MBR** Memory Buffer Register, trasmette il contenuto delle locazioni.
- **Registri I/O**
  - **I/O AR** I/O Address Register,
  - **I/O BR** I/O Buffer Register.
- **IR Instruction Register**
  La CPU mantiene una copia locale dell'istruzione da eseguire.
- **PC Program Counter**
  Mantiene l'indirizzo in memoria della prossima istruzione da eseguire.

MODULO I/O

Modulo per immettere *input* ed emettere *output*.

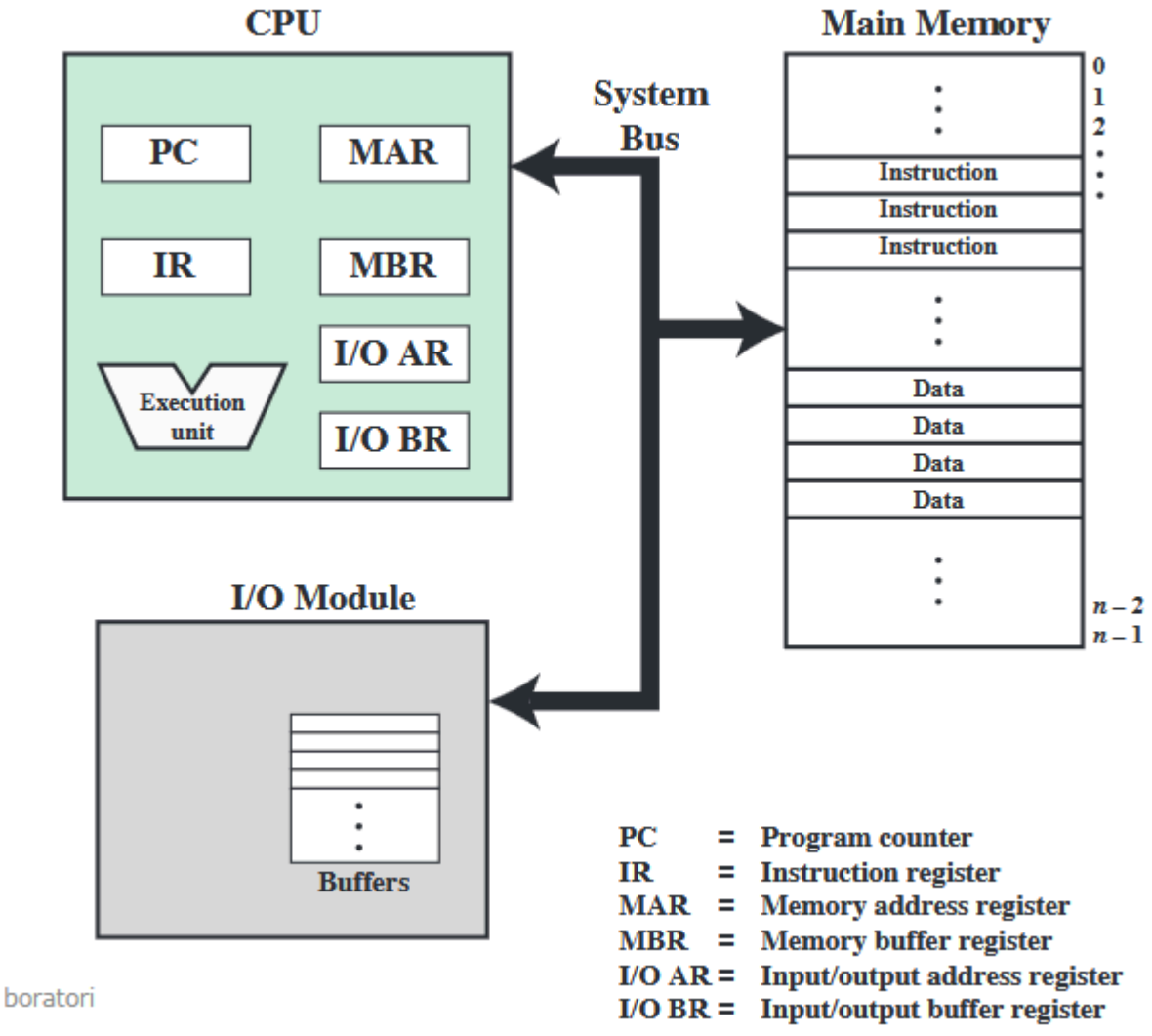
MEMORIA

- Serve per salvare *istruzioni e dati*.
- Permette di accedere a essi durante l'esecuzione dei programmi.
- E' divisa in *locazioni*, ognuna identificata da un indirizzo.

BUS

Per connettere i vari moduli.
Vedi INTERCONNESSIONI DI UN ELABORATORE.

SCHEMA ORGANIZZATIVO:



boratori

CICLO DI UN'ISTRUZIONE

1. Il PC contiene l'indirizzo in memoria della prossima istruzione da eseguire, che viene recuperata (*fetch*).
2. L'istruzione viene salvata nel registro IR.
3. Il registro PC viene incrementato per puntare alla prossima istruzione.
4. L'istruzione nel registro IR viene eseguita.
5. Si ritorna al punto 1.

OSSERVAZIONE: Alcune istruzioni potrebbero richiedere più dati, per questo nella CPU è presente anche un registro *accumulatore* **AC** per salvare dei dati temporanei.

ISTRUZIONI

Ogni istruzione è costituita da:

- **OPCODE**, che determina il tipo di operazione (es. load, store, add);
- **ADDRESS**, che contiene l'indirizzo in memoria.

INTERCONNESSIONI DI UN ELABORATORE

Le componenti devono essere connesse per supportare lo spostamento di dati/istruzioni:

- Da memoria a processore e viceversa,
- Da modulo I/O a processore e viceversa,
- Da modulo I/O a memoria e viceversa.

Tale interconnessione è realizzata tramite i bus:

- A ogni istante, solo *un* componente può *inviare* segnali in un bus.
- I segnali inviati da un componente sono *disponibili a tutti* gli altri componenti connessi allo stesso bus.

ESEMPI DI BUS:

1. **DATA BUS**: linee usate per trasmettere dati.
   Il numero di linee (*bus width*) indica il numero di bit che possono essere trasmessi contemporaneamente.

2. **ADDRESS BUS:** linee usate per trasmettere indirizzi.
   Per leggere un dato in memoria, il processore comunica l'indirizzo del dato tramite l'address bus. Il dato viene poi restituito tramite il data bus.
3. **CONTROL BUS:** linee usate per trasmettere segnali di controllo.
   Usato per coordinare l'accesso al bus dati/indirizzi tra i vari dispositivi a esso collegati, ad esempio attraverso segnali di temporizzazione che indicano quando un segnale nel bus dati è effettivamente valido.

MEMORIA RAM

- La memoria è divisa in *locazioni*.
- Ogni locazione è individuata da un *indirizzo*.
- Con un indirizzo da *n bit* sono indirizzabili un totale di 2^n locazioni.
  NOTA: $2^{10} = 1024 = 1K$ etc..
- Ogni locazione contiene *L bits*.
  Generalmente L=8 bits (1 byte) o, in alcuni casi, L=32 bits (4 byte).
- Una memoria può essere vista come un vettore di $N = 2^n$ elementi, dove ogni elemento occupa L bits.

TERMINOLOGIA:
La parola *word* è usata per denotare l'insieme di bit/byte che compongono un dato in un'architettura, la dimensione di una word è variabile.
Nel nostro caso, architettura *ARM 32bit*, la dimensione di una word è 32 bit, e quindi:

- *Halfword:* 16 bit (2 byte);
- *Doubleword*: 64 bit (8 byte).

ORDINAMENTO DEI BYTE IN MEMORIA

PROBLEMA: se la memoria è organizzata in locazioni da 1 byte, come fare per salvare delle word da 4 byte? *Es.:* 0xAABBCCDD
SOLUZIONI:
Salvare i byte della word in locazioni adiacenti, ma in che ordine?

- **BIG ENDIAN:** la parte *più significativa* viene memorizzata per prima (nella locazione con indirizzo minore).
- **LITTLE ENDIAN:** la parte *meno significativa* viene memorizzata per prima.

BIG ENDIAN		LITTLE ENDIAN	
Indirizzo	Contenuto	Indirizzo	Contenuto
0	0xAA	0	0xDD
1	0xBB	1	0xCC
2	0xCC	2	0xBB
3	0xDD	3	0xAA

TIPI DI MEMORIE

Esistono diversi tipi di memorie, con caratteristiche diverse:

1. **ROM - Read Only Memory**
   - Il contenuto della memoria è *permanente* e non può essere cambiato.
   - Non è richiesta alimentazione per mantenere i valori in memoria.
   - I dati sono cambiati nel chip durante il processo di fabbricazione.
   - L'inserimento dei dati è *costoso*.
2. **PROM - Programmable ROM**
   - Il contenuto della memoria è *permanente*.
   - Però la scrittura può essere effettuata *dopo* il processo di fabbricazione con opportuna apparecchiatura elettronica.
   - Non è richiesta alimentazione.
   - Maggior flessibilità e minor costo rispetto alla ROM.
3. **EPROM- Erasable Programmable ROM**
   - Si può cancellare l'intero contenuto con *radiazione UV*.
4. **EEPROM - Electrically Erasable PROM**
   - E' possibile cancellare singoli byte tramite segnali elettrici.
5. **FLASH MEMORY**
   - Costi e funzionalità intermedi tra EPROM ed EEPROM.
   - La cancellazione avviene per settori.
6. **RAM - RANDOM ACCESS MEMORY**
   - Permette accesso casuale (senza scorrimento di ogni locazione).
   - E' un tipo di memoria *volatile* (richiede alimentazione).
   - Si può sia leggere che scrivere.
   - Esistono due tipi principali di RAM: dinamica (*DRAM*) e statica (*SRAM*).

DRAM

- Il valore di un bit viene rappresentato tramite la presenza o meno di una carica in un condensatore.
- Richiede un *refresh* periodico per mantere il valore.
- Alta densità (circuito semplice) ma limitata velocità di accesso (*25-30ns*).

SRAM

- Il valore di un bit viene salvato tramite flip-flop.
- Mantiene il valore finchè è presente alimentazione.
- Bassa densità, alta velocità di accesso (*2-3ns*), utilizzata per la *CACHE*.

ORGANIZZAZIONE DELLA MEMORIA

Una memoria RAM con N locazioni da L bit può essere implementata tramite una struttura gerarchica:

- **CHIP di MEMORIA**: generalmente quadrati, consistono di N_1 locazioni da L_1 bit ciascuna.
- **MODULO**: costituito da *chip*, si ottengono N_1 locazioni da L bit ciascuna (è aumentata la dimensione delle locazioni).
- **BANCO**: ottenuto unendo i moduli, si aumenta il numero di locazioni da N a N_1.

ESEMPIO:

Implementazione di una memoria RAM di 16GB con 2G=2^{31} locazioni da 64 bit:

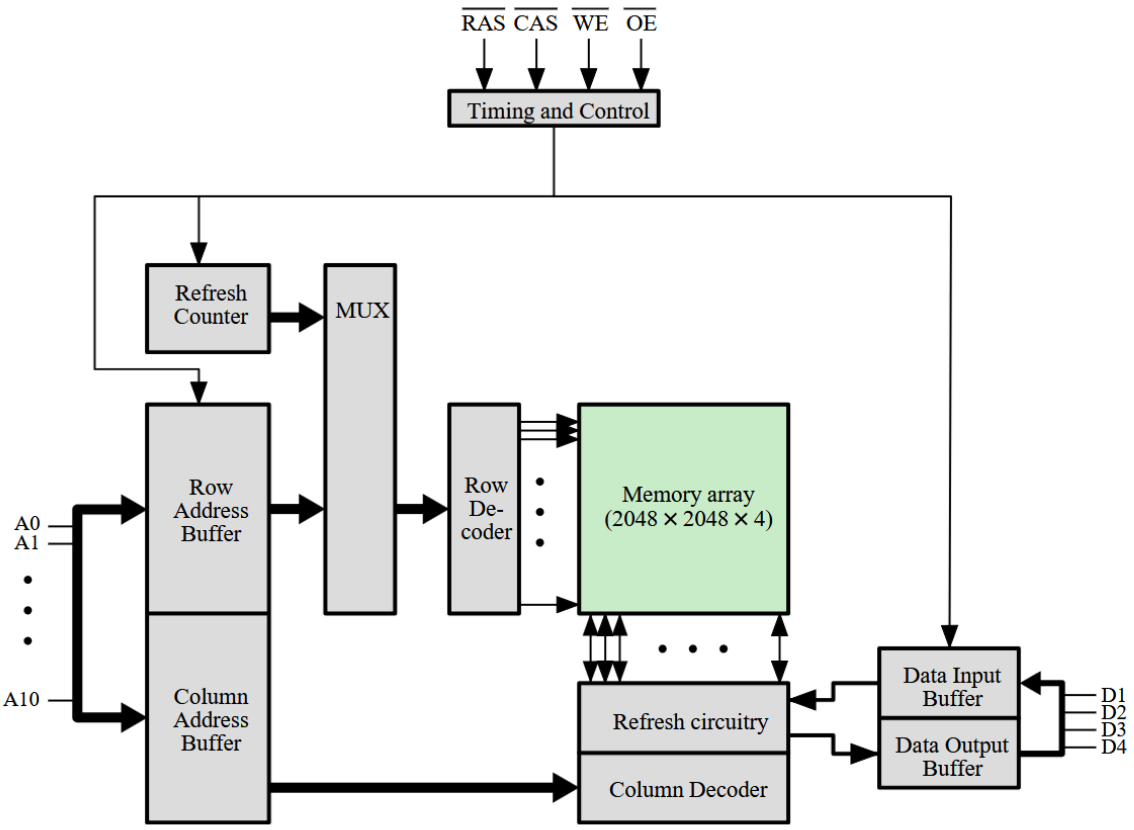
- Unità base: *chip* da 1G locazioni da 8 bit ciascuna;
- Si uniscono 8 chip in un *modulo* per ottenere 1G locazioni da 64 bit ciascuna;
- Si uniscono 2 moduli in un *banco* per ottenere 2G locazioni da 64 bit ciascuna.

ORGANIZZAZIONE DEI CHIP

Vediamo come organizzare un chip 4M x 4 (2^{22} locazioni da 4 bit):

- Posizioniamo le locazioni in una matrice da 2^{11} x 2^{11}: ciascuna cella è una locazione da 4 bit.
- Servono 11 bit per indirizzare la riga e altri 11 per la colonna.

NOTA: In un modulo (unione di chip), la stessa posizione in chip diversi ha lo stesso indirizzo (riga e colonna). In un banco, ogni modulo è individuato dai primi bit dell'indirizzo di memoria complessivo.



I 22 bit di indirizzo non sono trasferiti insieme, ma in *due parti*: prima l'indirizzo di riga, poi quello di colonna. Per questo sono presenti anche 4 *segnali di controllo*:

- **RAS**: (Row Address Select) indica che si sta leggendo l'indirizzo di riga.
- **CAS** (Column Address Select) indica che si sta leggendo la colonna.
- **WE**: (Write Enable) denota una scrittura.
- **OE**: (Output Enable) denota una lettura.

REFRESH DELLA MEMORIA

La *DRAM* è implementata con dei *condensatori* che perdono carica nel tempo, e richiedono quindi un refresh periodico per mantenere l'informazione.
Introduciamo le seguenti quantità:

- **PERIODO di REFRESH**: tempo disponibile per eseguire il refresh di ciascuna locazione, indicato con T_R.
- **TEMPO di REFRESH**: tempo a disposizione per eseguire il refresh di un'unità (cella o riga), indicato con t_R.

- **TEMPO di ACCESSO**: tempo necessario per un ciclo di lettura in memoria, indicato con t_a.

Prendiamo in esame un chip 64Kx1 (256 righe x 256 colonne), assumendo $T_R = 4ms$ e $t_a = 60ns$.
Se il refresh fosse operato per *singole celle*:

- $t_R = \frac{4ms}{2^{16}} = 61ns$
- Impegno %: $\text{Rfsh}_{\%} = 100\%$.

Operando il refresh *per righe*:

- $t_R = \frac{4ms}{256} = 16\mu s$
- $\text{Rfsh}_{\%} = \frac{60ns}{16\mu s} * 100 = 0.4\%$.