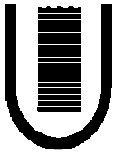
Virtual CPU – Eniac
parte 1



Università degli Studi di Roma “Tor Vergata”

Dr.ssa Veronica Marchetti

Rev. EN - 15gen08

Overview

- Capire perché è utile imparare a programmare a basso livello;
- Conoscere la struttura (lo schema circuitale) della CPU virtuale Eniac che adotteremo per programmare;
- Acquisire i concetti base a monte della programmazione utilizzando la CPU virtuale Eniac;
- Acquisire il set di istruzioni disponibile;
- Comprendere il funzionamento di piccoli programmi esistenti;
- Imparare a programmare, a basso livello, applicativi di piccola entità.

“Capire perché è utile imparare a programmare a basso livello”

Capire l'importanza della programmazione a basso livello

- Contro
  - Assenza quasi totale della gestione dei tipi di dato nei compilatori: è infatti il programmatore che si deve far carico del corretto utilizzo dei tipi;
  - Portabilità ridotta rispetto ai linguaggi più moderni;
  - Difficoltà nel comprendere programmi già scritti;
  - Difficoltà anche nello scrivere programmi.
  - Non è più utilizzato dalla maggior parte delle aziende che sviluppano software.

Capire l'importanza della programmazione a basso livello

- Pro
  - *Si percepisce come effettivamente si riescono a far svolgere delle attività ad un calcolatore;*
  - *Si ha la giusta percezione di cosa può e cosa non può fare un calcolatore;*
  - Si possono ottimizzare particolari routine di un applicativo, specialmente se scritto in C o in C++;
  - Fornisce la possibilità di intuire e ricostruire il codice sorgente a monte di un eseguibile preesistente (Reverse Engineering).

“Conoscere la struttura (lo schema circuitale) della CPU virtuale Eniac che adotteremo per programmare”

Un po' di terminologia

- Emulatore: è un software che ricrea l'hardware di un sistema, solitamente differente da quello di cui si dispone, permettendo di far eseguire programmi di vario genere come se fossero eseguiti nell'hardware emulato.
- Simulatore: è un software che imita l'hardware di un sistema.
- La differenza tra i due concetti risiede nel fatto che mentre il primo ha come obiettivo primario il funzionamento del software, per il secondo tale obiettivo è secondario: l'attenzione è infatti principalmente rivolta alla rappresentazione degli aspetti architetturali e funzionali dell'hardware emulato.

Un po' di terminologia

- **Assembly**: (definizione tratta dal sito di Wikipedia) è il linguaggio di programmazione più vicino al linguaggio macchina vero e proprio.

  Infatti, esiste una corrispondenza pressoché biunivoca tra gli mnemonici del linguaggio assembly ed i corrispondenti codici macchina corrispondenti ai bitfields (campi di bit) che compongono le istruzioni direttamente eseguibili dal dispositivo elettronico (in genere una CPU) che si sta programmando.

Un po' di terminologia

- **Assembler**: (definizione tratta dal sito di Wikipedia) è un programma compilatore che si occupa di tradurre in linguaggio macchina (ossia una serie di bit 0 e 1 che costituiscono l'unico modo per comunicare con dispositivi elettronici), una serie di comandi scritti in linguaggio Assembly.

  Oltre a questo compito base, un Assembler si occupa spesso di ausiliare il programmatore, ad esempio consentendo l'utilizzo nel codice sorgente del programma di nomi mnemonici al posto di indirizzi esadecimali che costituiscono l'esatta collocazione di una variabile o di una porzione di programma nella memoria centrale del computer.

Conoscere gli strumenti

- Durante queste lezioni prenderemo in considerazione un emulatore di una CPU virtuale progettata e realizzata dal Dott. Mauro Codella e Dott. Dario Dussoni ;
- L'emulatore preso in considerazione è Eniac
- Attualmente il software è reperibile alla seguente URL :

  http://sourceforge.net/projects/eniac/

Conoscere gli strumenti

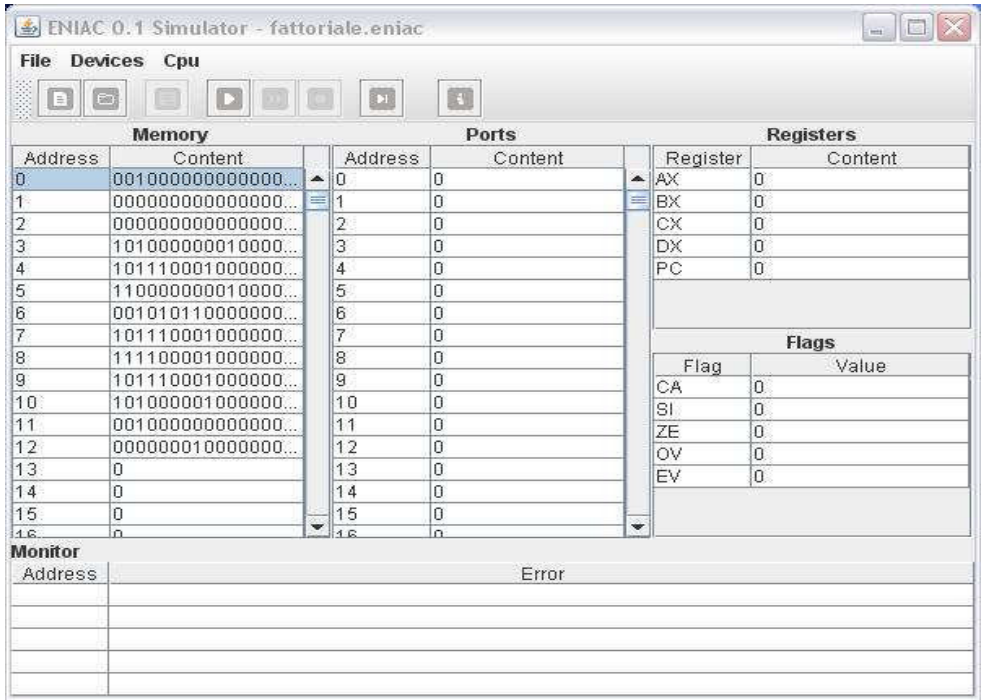
- E.N.I.A.C., è l'acronimo di Electronic Numerical Integrator And Computer, il primo computer elettronico della storia (1946).
- Nel 2006 si è celebrato il 60° anniversario, non solo di ENIAC ma della storia della intera programmazione;
- Poiché tale emulatore è stato realizzato proprio nel 2006 gli è stato dato questo nome come tributo.

Conoscere gli strumenti

- Lo scopo dell'emulatore è quello di rendere possibile sia l'esecuzione di programmi, che di rendere comprensibile tutto ciò che accade all'interno di un calcolatore.
- Esso mette a disposizione un'interfaccia grafica semplice e funzionale, che permetta di gestire agevolmente tutte quelle risorse coinvolte nell'esecuzione di un programma,quali memoria, porte, registri, flags, ma non solo: la GUI di ENIAC è stata pensata mirandola, in modo particolare, a scopo didattico

Conoscere gli strumenti

- Dispone di un editor di testo sensibile alla sintassi dell'assembly, per gestire tutte le celle di memoria, le porte, i registri, i flags.
- E' inoltre contemplato un campo per visualizzare eventuali errori.



Il perché della scelta

- Perché utilizzare un emulatore e non un assembler?
  - L'emulatore permette di analizzare le componenti salienti della CPU in modalità real-time: durante l'esecuzione delle istruzioni sono infatti visibili immediatamente le modifiche ai registri e ai flag;
  - Possiamo eseguire il nostro codice su un qualunque calcolatore in grado di eseguire l'emulatore;
  - Non vi è la necessità di ricompilare il programma ad ogni minima modifica al codice;
  - I vantaggi offerti dall'assembler (velocità ed ottimizzazione del codice) non sono pertinenti allo scopo di queste lezioni.

“Acquisire i concetti base a monte della programmazione assembly”

Rappresentazione in base 2 di un numero in base 10

- Vogliamo richiamare qui l'algoritmo per la rappresentazione in base 2 di un numero in base 10;
- Algoritmo di conversione:

I. scegliamo il numero in base 10 da convertire;

II. memorizziamo il numero in una variabile che chiameremo ***x***;

III. dividiamolo il valore in ***x*** per 2;

IV. riponiamo il resto della divisione in una pila ***P***;

V. se il risultato della divisione è uguale a 0 allora salta al passo VIII, altrimenti procedi con il passo successivo;

VI. riponiamo il risultato della divisione in ***x***;

VII. ritorniamo al passo III;

VIII. fine: nella pila troviamo il numero convertito in binario, con la cifra più significativa (MSB) sulla cima.

Rappresentazione in base 16 di un numero in base 10

- Vogliamo richiamare qui l'algoritmo per la rappresentazione in base 16 di un numero in base 10;
- Algoritmo di conversione:

I. scegliamo il numero in base 10 da convertire;

II. memorizziamo il numero in una variabile che chiameremo ***x***;

III. dividiamolo il valore in ***x*** per 16;

IV. riponiamo il <u>resto</u> della divisione in una pila ***P***;

V. se il risultato della divisione è uguale a 0 allora salta al passo VIII, altrimenti procedi con il passo successivo;

VI. riponiamo il <u>risultato</u> della divisione in ***x***; (*)

VII. ritorniamo al passo III;

VIII. fine: nella pila troviamo il numero convertito in esadecimale, con la cifra più significativa sulla cima.

Rappresentazione in base 16 di un numero in base 10

(*) incontriamo un problema: sapendo che, in generale, il resto di una divisione per ***d è pari ad un r nell'intervallo chiuso [0, d-1], come ci comportiamo quando dividiamo per 16? Il resto in questo caso è nell'intervallo [0, 15], ma nella pila p siamo costretti ad inserire una sola cifra... cosa fare?***

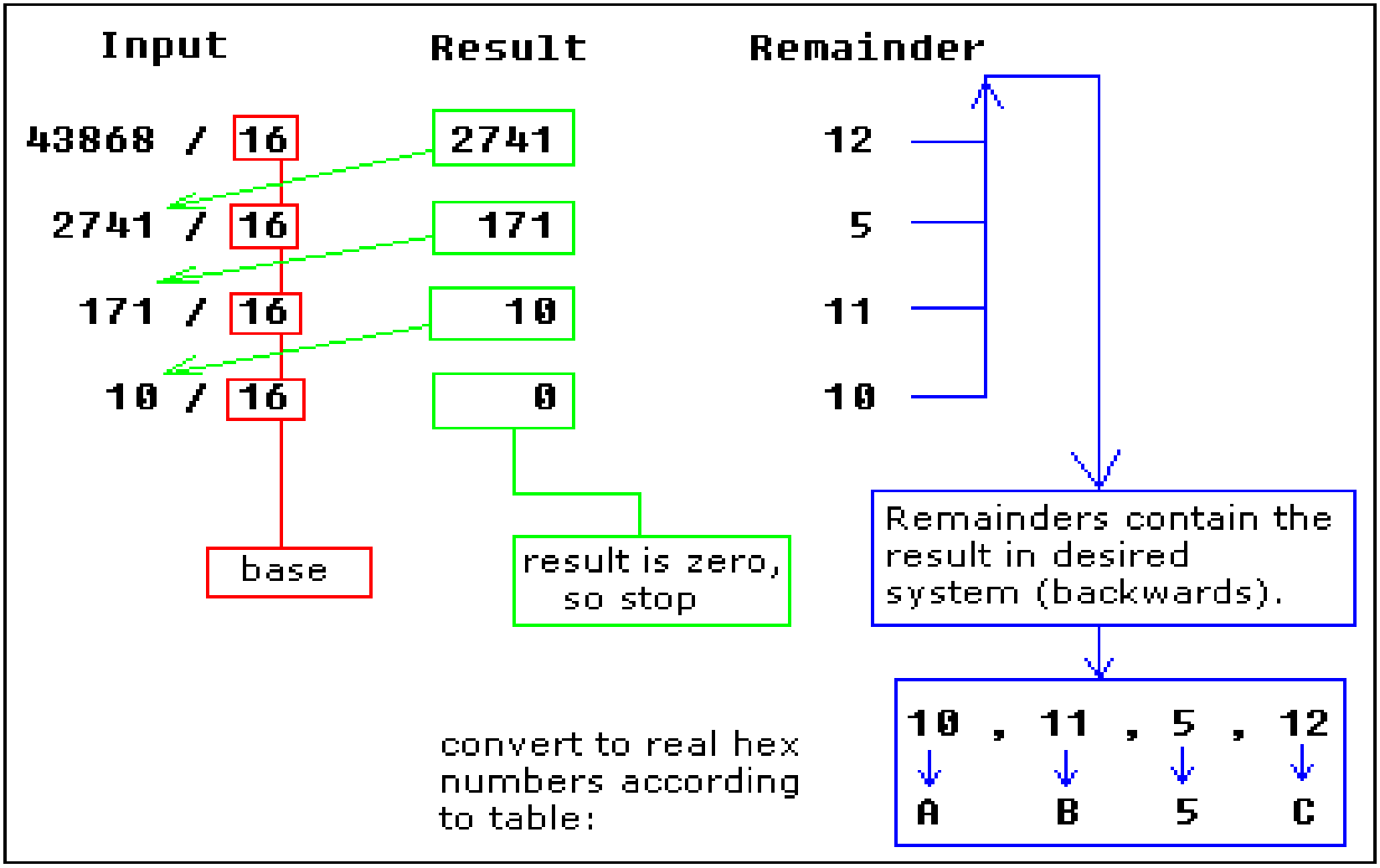
Rappresentazione in base 16 di un numero in base 10

(*) ... semplicemente associamo ai valori 10, 11, 12, 13, 14 e 15 dei simboli univoci che li rappresentano. In particolare si prende A per rappresentare 10, B per 11, C per 12, D per 13, E per 14 ed F per 15.

Tabella riassuntiva delle cifre esadecimali

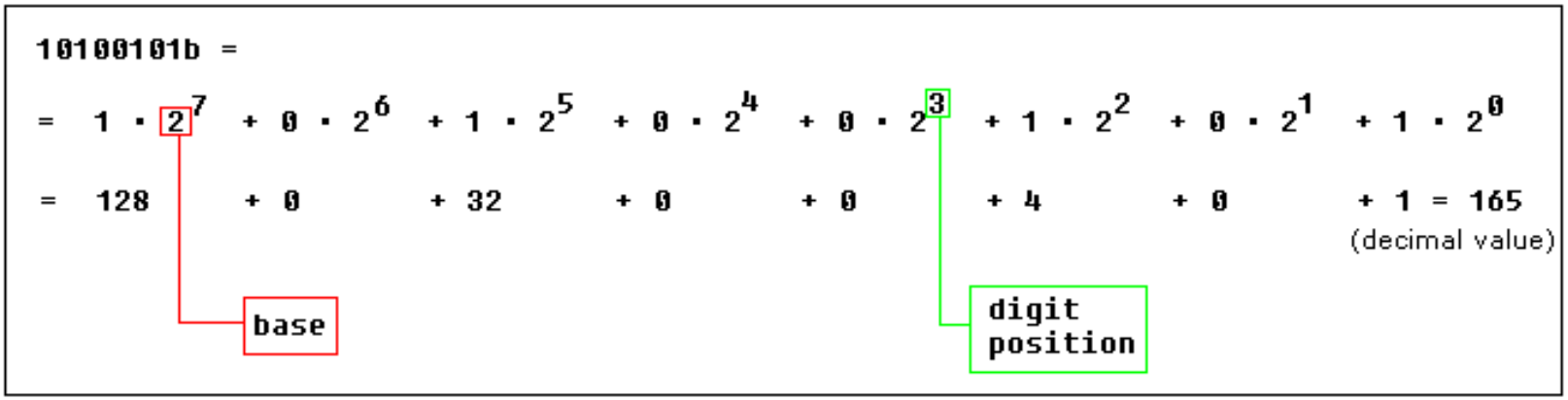
DEC	BIN	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Rappresentazione in base 16 di un numero in base 10

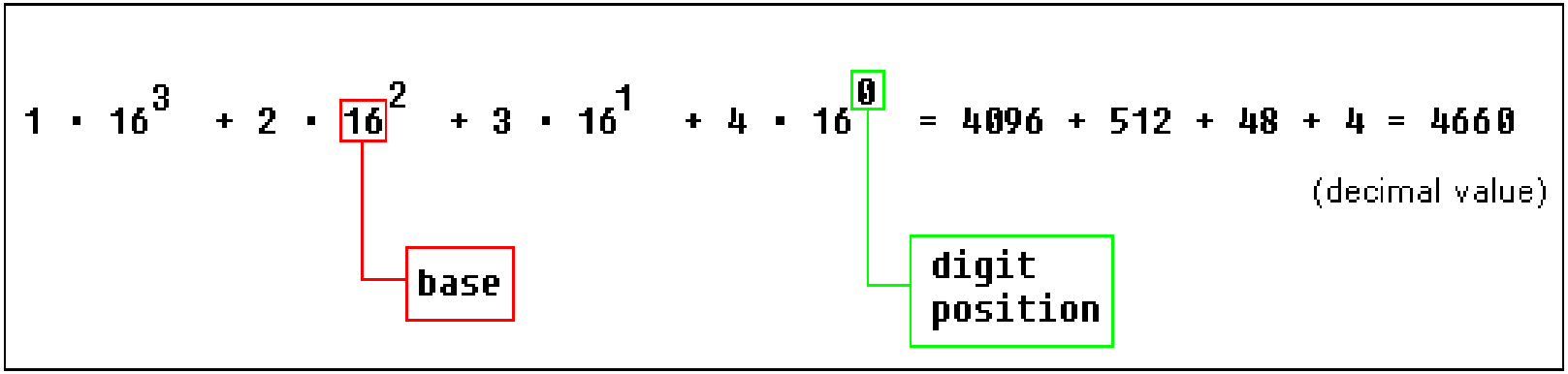


Conversioni da base 2, base 16 a base 10

- Da base 2 a base 10:

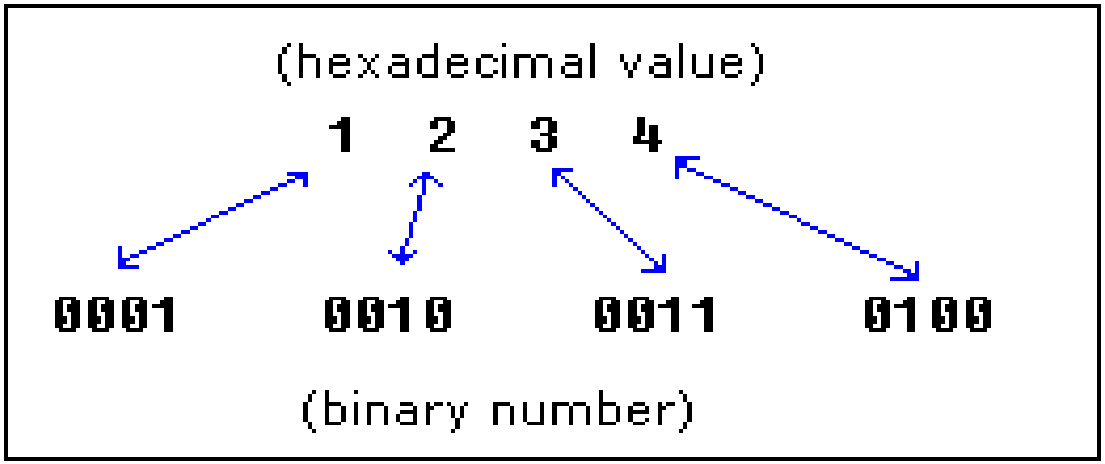


- Da base 16 a base 10:



Notiamo una certa caratteristica nella rappresentazione in base 16

- Dalla tabella riassuntiva delle cifre esadecimali si vede chiaramente che esiste una corrispondenza biunivoca tra codici binari a 4 bit e le cifre della rappresentazione esadecimale;
- Questo è il motivo chiave che ha spinto ad adottare la rappresentazione in base 16: permette infatti di rappresentare in forma compatta e facilmente leggibile i numeri binari di grande dimensione;
- Convertire un numero binario in uno esadecimale e viceversa è un'operazione banale. Infatti:



Convenzioni

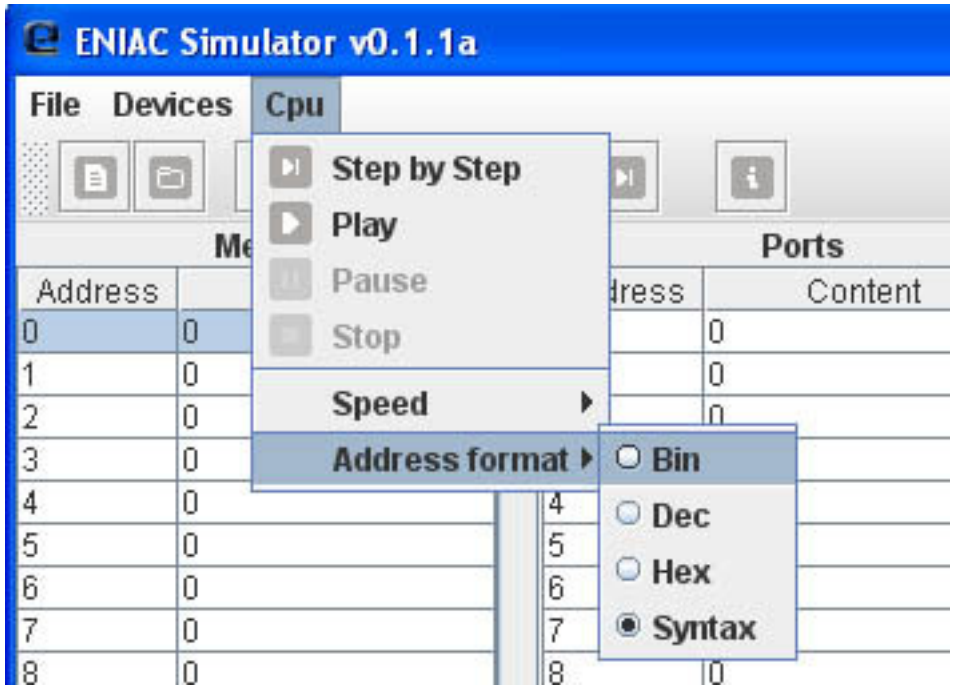
- Per indicare un numero in base 10, ad esempio ***x, si utilizzerà*** $(x)_{10}$***, o più semplicemente x;***
- ***Per indicare un numero in base 16, si utilizzerà la sintassi*** $(x)_{16}$ ***o più semplicemente xh;***
- ***Per indicare un numero in base 2, si utilizzerà la sintassi*** $(x)_2$***, o più semplicemente xb;***
- ***Sia x un carattere: di seguito si intenderà con 'x' il codice numerico associato alla codifica di x.***

Concludendo: perché tre basi diverse?

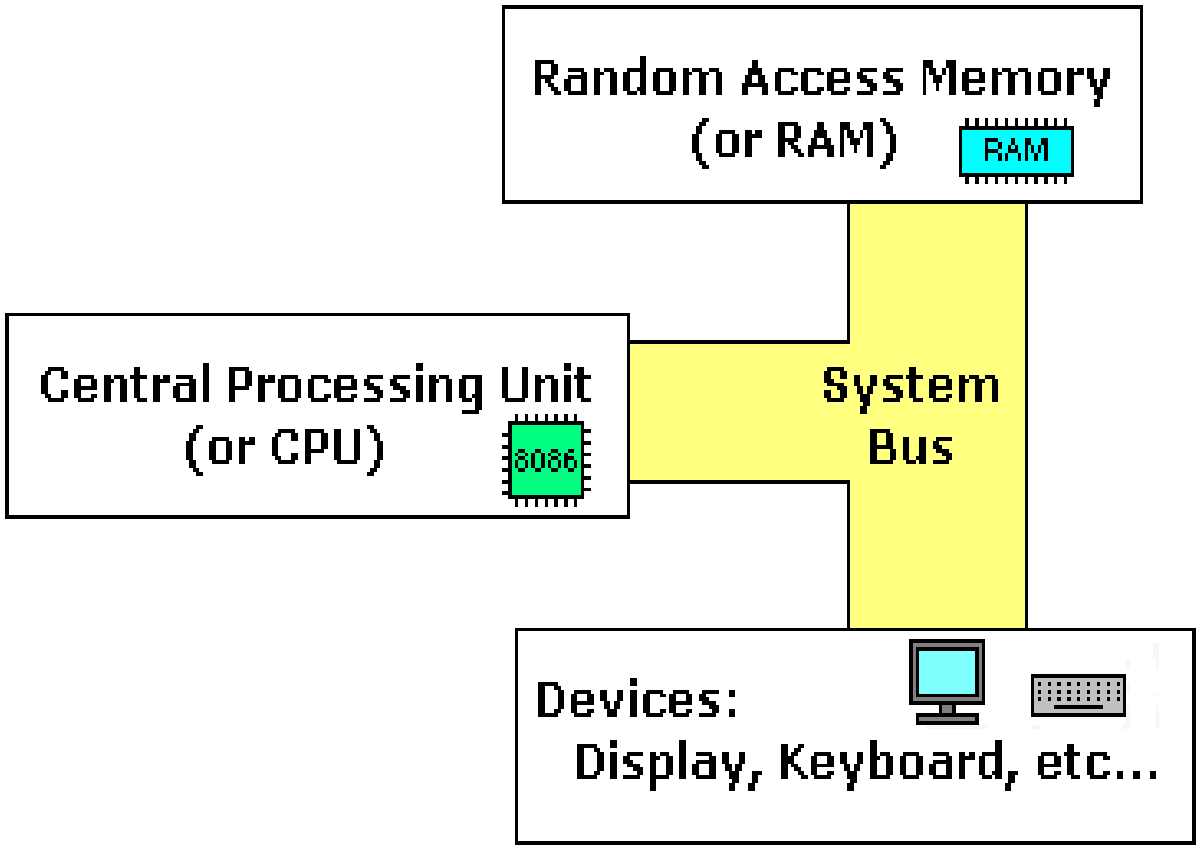
- Le ragioni che hanno spinto i programmatori a basso livello ad adottare tre basi diverse per indicare un numero sono le seguenti:
  - La base 10 permette di rappresentare i numeri nella forma che meglio conosciamo: sin dagli albori dell'umanità l'uomo ha contato utilizzando le dieci dita;
  - La base 2 permette di rappresentare i numeri nella forma canonica dei calcolatori: è particolarmente utile quando si vuole specificare una particolare sequenza di cifre binarie;
  - Spesso, le cifre binarie che si vogliono rappresentare sono molto lunghe: l'utilizzo della base 16 permette di rendere quattro volte più compatta la loro rappresentazione.

Ecco come ENIAC ci assiste in task di conversione di base

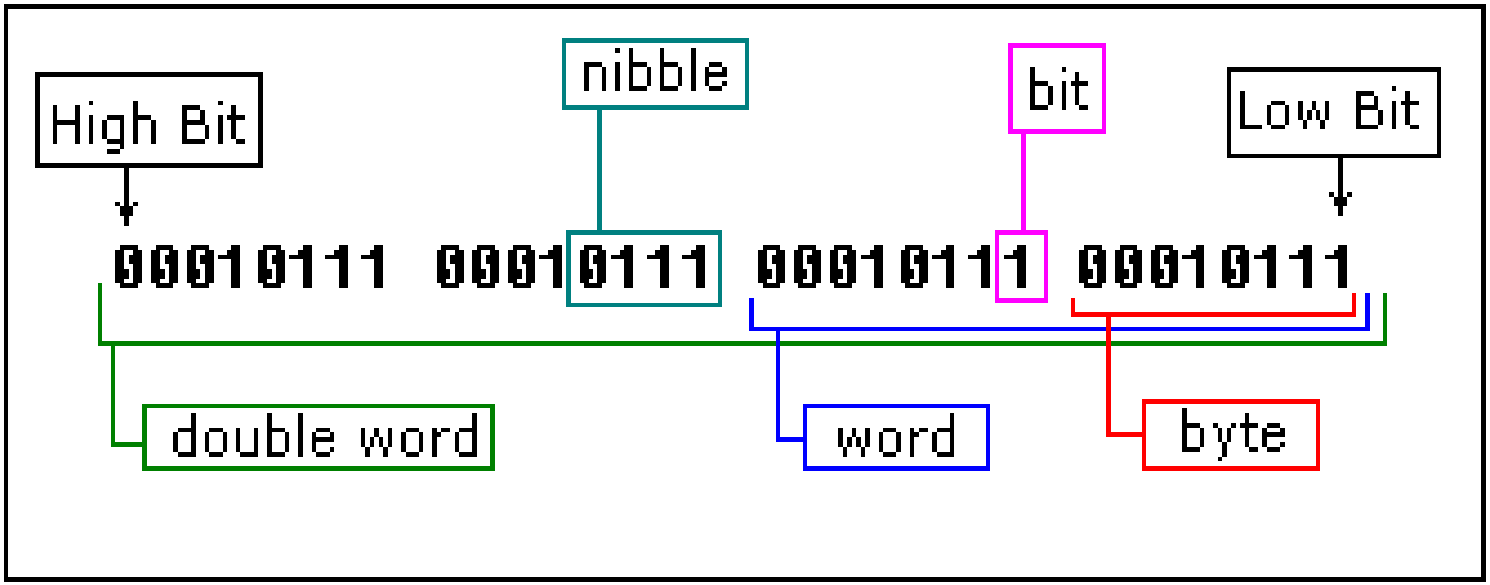
- In ENIAC disponiamo di un Sottomenu Address Format.
- Permette di cambiare le modalità di visualizzazione dei dati presenti all'interno delle risorse di ENIAC.



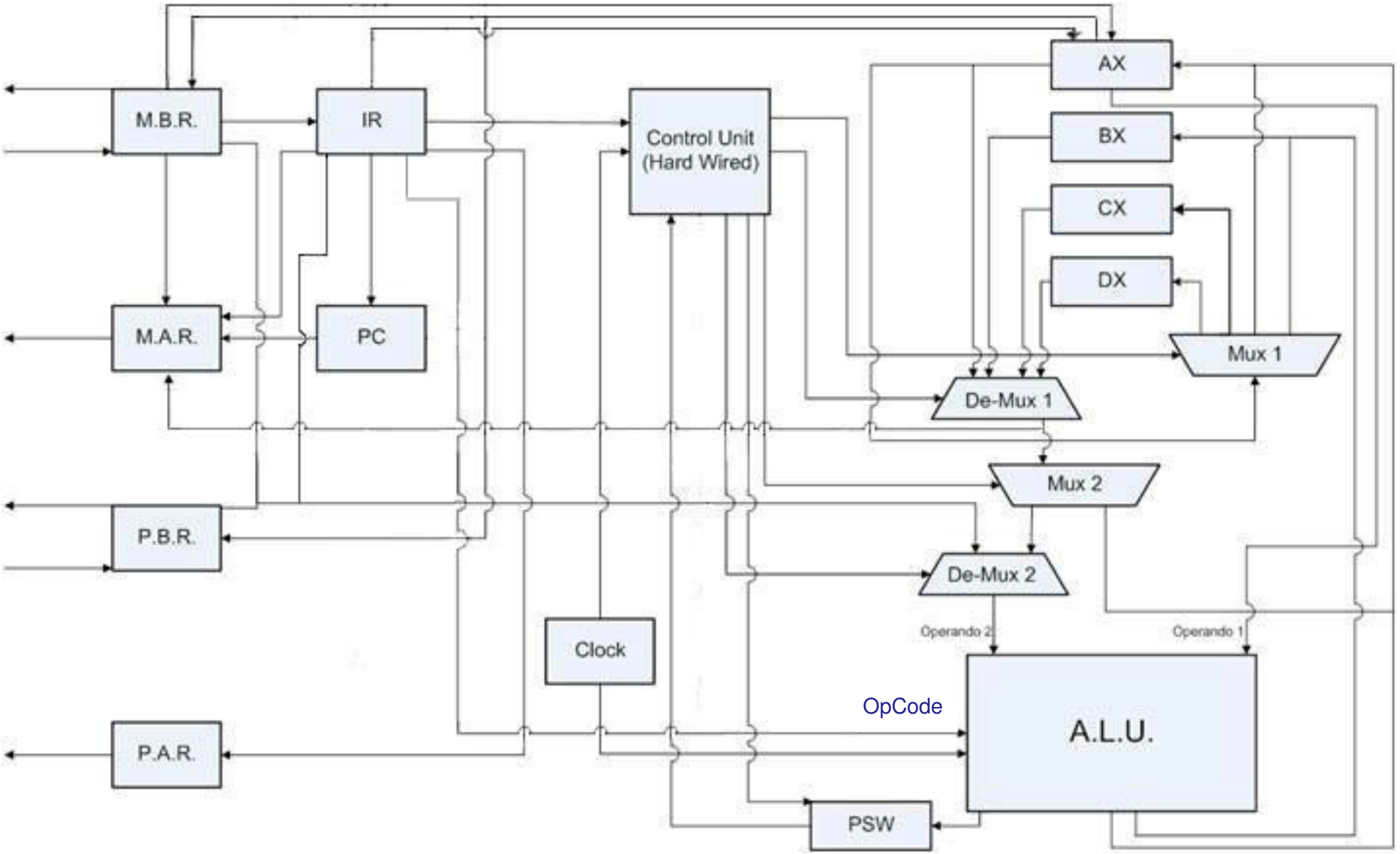
Come è fatto il nostro calcolatore



Un po' di terminologia



Struttura della vCPU ENIAC a 24 bit



Visione d'insieme

- ***MBR, MAR, PBR, PAR***, sono registri che rappresentano i punti di contatto tra la CPU e le risorse esterne.
  - ***MBR, MAR*** : memorizzano il dato in transito da o verso la memoria (MBR) e contengono l'indirizzo della cella in questione (MAR);
  - ***PBR, PAR*** : memorizzano il dato in transito da o verso le porte (PBR) e contengono il numero di porta alla quale ci si riferisce (PAR);
- ***Control Unit*** : decodifica l'istruzione ed esegue l'operazione associata ad essa.
- ***ALU*** (Arithmetic and Logic Unit) : permette di effettuare operazioni logiche e matematiche.
  - Può effettuare 10 operazioni differenti
- ***Le componenti comunicano mediante connessioni dirette e NON mediante bus***.

Visione d'insieme

- **AX**, l'accumulatore;
- **BX**, **CX**, **DX**, registri general purpose;
- **PSW** (Processor Status Word) **:** il registro dei flags;
- **PC** (il Program Counter) : contiene l'indirizzo della cella contenente l'istruzione successiva da eseguire (non utilizzabile direttamente dal programmatore).
- **IR** (il Registro delle Istruzioni) : contiene l'istruzione appena prelevata dalla memoria (non utilizzabile direttamente dal programmatore).

Ognuno di questi registri ha per semplicità la dimensione di una cella di memoria (24 bit) .

Flags

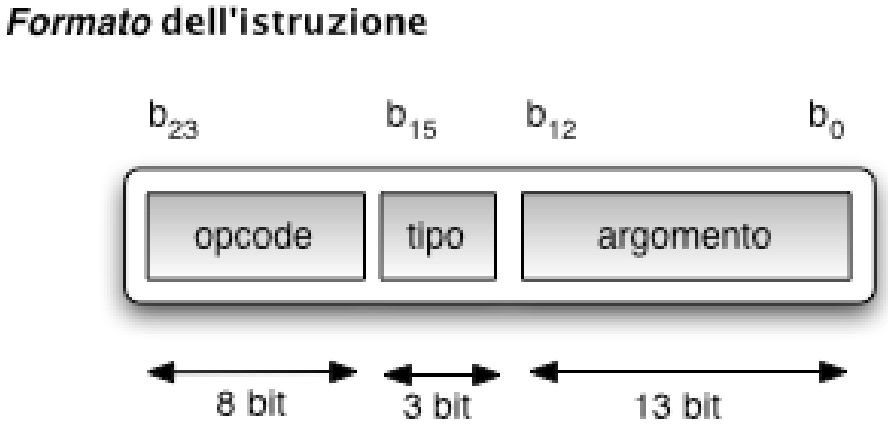
- I flag, cioè i bit del registro PSW, sono essenziali per prendere decisioni in merito all'esito delle istruzioni eseguite precedentemente, per poter cambiare il flusso del programma.
- Il valore dei flag dopo una computazione rappresentano una parte dello stato finale in cui è arrivata la ALU.
- I flag si distinguono in *semplici* e *complessi*. I flag *semplici* sono così chiamati in quanto il meccanismo che ne calcola il valore prescinde dal tipo di operazione effettuata. I flag *complessi,* invece, vengono calcolati con meccanismi che oltre al risultato si basano anche sull'operazione effettuata.

Memoria

- Le celle di memoria hanno una dimensione pari a quella del formato di istruzioni adottato (24 bit).
- Le celle sono associate univocamente a dei valori numerici contigui, crescenti a partire da zero, chiamati *indirizzi*.
- La memoria contiene 2^{12} celle, poiché abbiamo adottato la codifica del complemento a 2 per rappresentare gli interi e nel formato delle istruzioni abbiamo assegnato 13 bit all'argomento per contenere l'indirizzo.

Formato dell'istruzione

- Fissato a 24 bit.
- 8 bit dedicati al codice operativo (b_{23}…b_{16}).
- 3 bit per il tipo dell'argomento (b_{15}…b_{13}).
- 13 per l'argomento (b_{12}…b_0).



Tipologia di argomenti

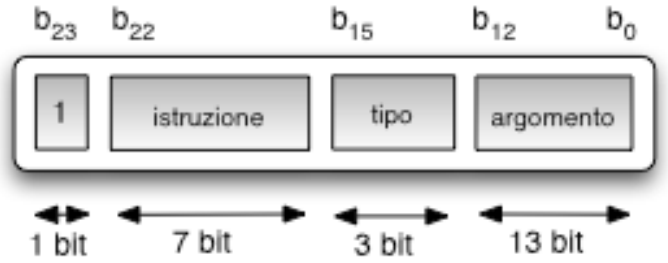
- L'argomento di una istruzione può essere di vari **tipi**: ad esempio:
  - Indirizzo della cella di memoria contenente l'operando necessario all'istruzione;
  - L'operando, direttamente fruibile dall'istruzione.
  - Il codice del registro contenente l'operando.
  - Altro ancora…
  - Il tipo viene determinato dal valore dei bit $b_{15}…b_{13}$

- Alcune istruzioni gestiscono argomenti di diversi tipi, mentre altre ancora usano sempre la stesso. Questa differenza divide le istruzioni in due classi:
  - Istruzioni che usano argomenti di diverso tipo.
  - Istruzioni che usano un argomento sempre dello stesso tipo.

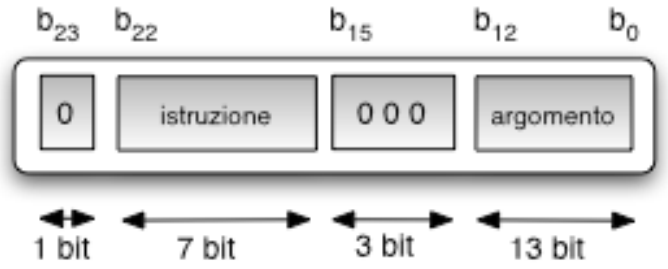
Classi di istruzioni

- La differenza tra le due categorie viene formalizzata per mezzo della specifica di due classi di istruzioni: la classe α e la classe β.
- Le due classi si distinguono per mezzo dell'MSB dell'istruzione: il bit b_{23}.
- Nella classe β i bit $b_{15}\ldots b_{13}$ sono sempre assegnati a zero in quanto inutilizzati. La tipologia dell'argomento è insita nel codice operativo.

Classe α



Classe β



Nella prossima lezione…

- Architettura interna della ALU