

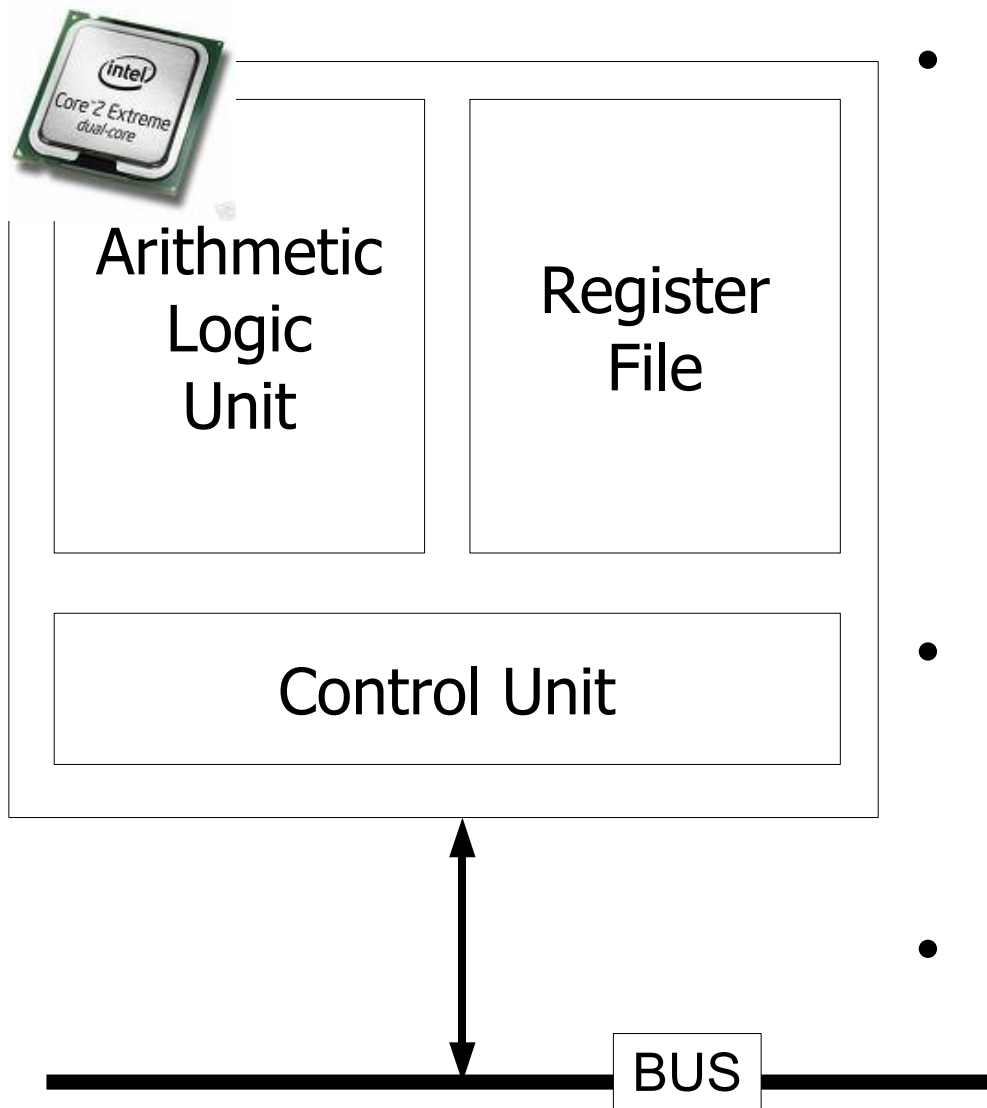


Architettura della CPU e linguaggio assembly

Corso di Abilità Informatiche
Laurea in Fisica

prof. ing. Corrado Santoro

Schema a blocchi di una CPU



- **Control Unit (CU)**
 - Si interfaccia con RAM/ROM e I/O
 - Legge le istruzioni dalla mem centrale (RAM/ROM) e le interpreta
 - Legge e scrive dati da e su mem centrale
- **Arithmetic Logic Unit (ALU)**
 - Esegue operazioni logico/matematiche e di confronto
- **Register File**
 - E' una piccola memoria utile al funzionamento della CPU

Control Unit: Funzionalità



- Opera secondo una sequenza periodica basata su tre fasi principali:
 - **FETCH**
 - **DECODE**
 - **EXECUTE**
- **FETCH**: viene prelevata dalla mem centrale il byte contenente l'istruzione da eseguire
- **DECODE**: l'istruzione viene "decodificata", preparando la circuiteria per la fase successiva
- **EXECUTE**: l'istruzione viene effettivamente eseguita.
- ***Ma da quale locazione di memoria la CU preleva l'istruzione da eseguire?***

Register File



- E' una memoria, con poche locazioni, che serve alla CPU per il suo normale funzionamento
- Le locazioni non sono indicizzate per numero ma per \Ogni locazione (nome simbolico) è denominato registro
- Una CPU ha almeno i seguenti registri:
 - **PC, Program Counter:**
 - Contiene l'indirizzo della locazione di memoria dove la CU preleverà la prossima istruzione
 - La sua dimensione (in bit) è tale da coprire tutta la memoria centrale indirizzabile dalla CPU
 - **A, Accumulator:** E' usato nelle operazioni matematiche e nei trasferimenti
 - Ogni trasferimento avviene da mem centrale a registro A e viceversa
 - Ogni operazione matematica avviene tra A e un operando specificato
 - **FLAGS (bit-mapped):** contiene lo "stato" della CPU e dell'ultima operazione eseguita (es. zero, numero negativo, etc.)

Control Unit (Revised)



- **FETCH:** viene prelevata, dall'indirizzo specificato nel registro PC, il byte contenente l'istruzione da eseguire
- **DECODE:** l'istruzione viene "decodificata", preparando la circuiteria per la fase successiva
- **EXECUTE:** l'istruzione viene effettivamente eseguita e il PC viene aggiornato in modo da "puntare" all'istruzione successiva
- Tipologie di istruzioni eseguite
 - Trasferimenti di uno o più byte da CPU a memoria e viceversa
 - Operazioni matematico-logiche
 - Confronti
 - "Salti" incondizionati e condizionati

Arithmetic Logic Unit



- Ha il compito di eseguire operazioni
 - **Matematiche** (somma, differenza, negazione, prodotto, divisione, modulo)
 - **Logiche** (AND, OR, NOT)
 - **Confronti** ($>$ $=$ $<$)
- Tutte le operazioni sono effettuate tra l'accumulatore e un operando che può essere:
 - Un valore immediato / costante
 - Il contenuto di una locazione di memoria specificata
- Il risultato è sempre memorizzato nell'accumulatore
- Dopo l'operazione, il registro FLAGS viene aggiornato sulla base del risultato (es. zero, numero negativo, etc.)

Sommiamo due numeri!



- Supponiamo di voler sommare i byte posti nelle locazioni 100 e 101, memorizzando il risultato nella locazione 102
 - $\text{MEM}(102) = \text{MEM}(100) + \text{MEM}(101)$
- La CPU non può farlo direttamente ma deve passare dall'accumulatore:
 - **Copia di MEM(100) su A**
 - $\text{MEM}(100) \rightarrow A$
 - **Somma di A con MEM(101)**
 - $A + \text{MEM}(101) \rightarrow A$
 - **Copia di A su MEM(102)**
 - $A \rightarrow \text{MEM}(102)$

Verso l'assembly



- **Copia di MEM(100) su A**
 - **MEM(100) $\rightarrow$ A**
- **Somma di A con MEM(101)**
 - **A + MEM(101) $\rightarrow$ A**
- **Copia di A su MEM(102)**
 - **A $\rightarrow$ MEM(102)**
- Tre tipologie di istruzioni:
 - LOAD A, (mem) mem $\rightarrow$ A
 - STORE A, (mem) A $\rightarrow$ mem
 - ADD A, (mem) A + mem $\rightarrow$ A

Primo programma in assembly



- $\text{MEM}(102) = \text{MEM}(100) + \text{MEM}(101)$
- **$\text{MEM}(100) \rightarrow A$**
- **$A + \text{MEM}(101) \rightarrow A$**
- **$A \rightarrow \text{MEM}(102)$**
- **LOAD A, (100)**
- **ADD A, (101)**
- **STORE A, (102)**
- **QUESTO E' LINGUAGGIO ASSEMBLY!!**
- **E' dipendente dalla CPU, cioè ogni tipo (famiglia) di CPU ha il suo proprio set di istruzioni assembly**

Altro esempio in assembly



- $R = X + Y * Z$
 - $MEM(100) = X$
 - $MEM(101) = Y$
 - $MEM(102) = Z$
 - $MEM(103) = R$
- **LOAD A, (101)** **;; Y -> A**
- **MUL A, (102)** **;; A * Z -> A**
- **ADD A, (100)** **;; A + X -> A**
- **STORE A, (103)** **;; A -> R**

Altro esempio in assembly



- **$R = X * Y + Z * K$**
 - $MEM(100) = X$
 - $MEM(101) = Y$
 - $MEM(102) = Z$
 - $MEM(103) = K$
 - $MEM(104) = R$
- **LOAD A, (100) ;; X -> A**
- **MUL A, (101) ;; A * Y -> A**
- **STORE A, (105) ;; A -> temp**
- **LOAD A, (102) ;; Z -> A**
- **MUL A, (103) ;; A * K -> A**
- **ADD A, (105) ;; A + temp -> A**
- **STORE A, (104) ;; A -> R**

Simboli o numeri?



- `LOAD A, (100) ; ; X -> A`
- `MUL A, (101) ; ; A * Y -> A`
- `STORE A, (105) ; ; A -> temp`
- ...
- Questo è un programma espresso in un linguaggio simbolico
- Ma la CPU capisce simboli o numeri?
- Il programma, che sta nella mem centrale è fatto di simboli o di numeri?
- Ogni istruzione mnemonica ha il suo corrispondente **codice numerico (in genere esadecimale)**! Es.
 - `LOAD = 80H`
 - `STORE = 81H`
 - `ADD = A0H`
 - `SUB = A1H`
 - `MUL = A2H`
- Questa mappatura dipende dall'assembly della specifica CPU

Simboli o numeri?



- Ogni istruzione mnemonica ha il suo corrispondente **codice numerico!**
Es.
 - LOAD = 80H
 - STORE = 81H
 - ADD = A0H
 - SUB = A1H
 - MUL = A2H
- Nella memoria, ogni istruzione è codificata dal suo “**opcode**” seguito da uno o più byte che sono gli **argomenti** dell'istruzione
- Esempio:
 - LOAD A, (100) -> 80H 64H
 - ADD A, (101) -> A0H 65H
 - ...

Codifichiamo il nostro programma



- **LOAD A, (100) ; ; X -> A**
 - **MUL A, (101) ; ; A * Y -> A**
 - **STORE A, (105) ; ; A -> temp**
 - **LOAD A, (102) ; ; Z -> A**
 - **MUL A, (103) ; ; A * K -> A**
 - **ADD A, (105) ; ; A + temp -> A**
 - **STORE A, (104) ; ; A -> R**
-
- **Secondo la codifica scelta, questo programma equivale alla sequenza di byte:**
 - **80 64 A2 65 81 69 80 66 A2 67 A0 69 81 68**