1.8 Architettura interna della CPU

Architettura interna della CPU

In questa unità viene analizzata l'architettura interna di un CPU di tipo Von Neumann da un punto di vista sistemico, cioè attraverso schemi a blocchi di tipo funzionale senza entrare nel dettaglio delle caratteristiche elettroniche.
Lo schema si riferisce ad una generica CPU senza un preciso riferimento ad una specifica marca e modello e serve solo per capire i principi di funzionamento.
Per avere una idea della situazione di può pensare che questo modello descrive con una certa accuratezza le CPU degli anni '70 (Intel 8080, Zilog Z80, Motorola 65xx) quando questi componenti rispecchiavano abbastanza fedelmente il modello "Von Neumann". L'incremento della integrazione ha portato all'inserimento nelle moderne CPU di molte altre funzioni non previste in questo modello che saranno trattate nella successiva unità "Architetture non Von Neumann". Anche con queste aggiunte il principio di funzionamento resta però valido soprattutto perché dal punto di vista del programmatore gli elementi innovativi sono spesso invisibili e si percepiscono solo come un incremento di prestazioni di un modello che resta però quello tradizionale.
Nei paragrafi che seguono sono elencate alcune definizioni che spiegano il significato dei blocchi che compaiono nello schema.

Sezione di controllo
E' la parte di CPU, individuata dal rettangolo tratteggiato di sinistra, dedicata alle operazioni di controllo dei bus. E' un automa governato dal segnale di clock che interpreta le istruzioni, attua le modifiche sui bus dei dati e degli indirizzi controllandoli con il control bus e governa la sezione esecutiva.

Sezione esecutiva
E' la parte di CPU, individuata dal rettangolo tratteggiato di destra, dedicata alle operazioni di elaborazione dei dati. E' anch'essa un automa che opera governato dall'unità di controllo effettuando trasformazioni sui dati presenti all'interno della CPU.

Internal bus
E' il bus interno alla CPU che mette in comunicazione tutti gli elementi che fanno parte sia dell'unità di controllo che della unità esecutiva. Questo bus è unico (senza distinzione tra dati ed indirizzi) perché la CPU, che genera gli indirizzi di selezione delle periferiche, deve essere in grado di modificarli quindi internamente alla CPU un indirizzo è un dato come qualsiasi altro.

Registro interno
Il termine registro indica un blocco di memoria, equivalente nella sua struttura ad una cella di memoria ma contenuto all'interno della CPU. Le dimensioni (in bit) dipendono dal tipo di CPU e in genere sono in relazione con l'ampiezza di parola anche se non c'è sempre coincidenza tre i due valori.
La funzione dei registri è di realizzare una memorizzazione temporaneo di una informazione, per il solo tempo necessario per portare a termine l'operazione in cui l'informazione è coinvolta. Si può quindi immaginare un registro come una lavagna su cui viene scritta una informazione per la durata della loro spiegazione e poi viene cancellata quando si passa alla spiegazione successiva; se si vuole che l'informazione sia conservata permanentemente è necessario salvarla da qualche altra parte (appunti = variabile in memoria centrale).
Diversamente dalle celle di memoria i registri non hanno un indirizzo (non sono in memoria centrale ma dentro alla CPU); sono invece identificati da un nome. Alcuni registri sono accessibili al programmatore attraverso il loro nome (modello di programmazione) mentre altri sono invisibili e servono per attuare le azioni di controllo.

Modello di programmazione
E' la parte di CPU che è accessibile al programmatore (in linguaggio Assembly), evidenziata in colore più scuro nello schema, ed è formata dai registri e dalla ALU che è l'esecutore aritmetico/logico.
Alcuni registri sono specializzati, cioè svolgono solo una specifica funzione mentre altri sono di uso generale, possono cioè essere usati dal programmatore per un qualsiasi scopo.
Le trasformazioni dei dati contenuti neri registri vengono effettuate attraverso elaborazioni della ALU.

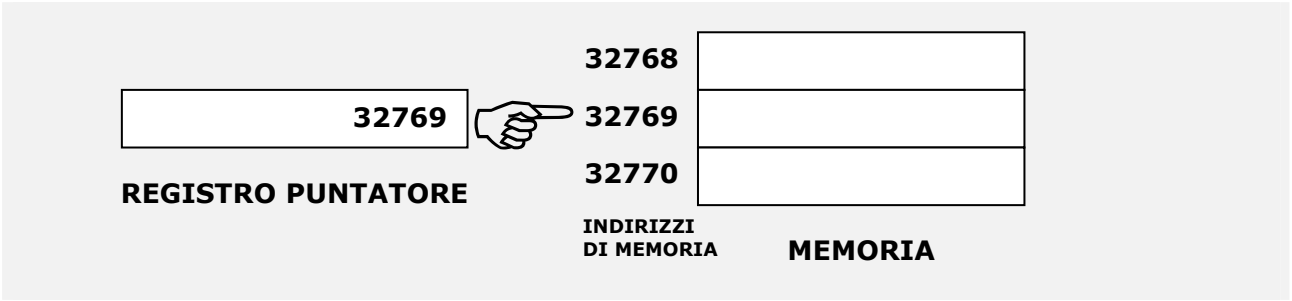
Registro specializzato
E' un registro destinato ad una specifica funzione, può essere visibile al programmatore E' la parte di CPU che è accessibile al programmatore (in linguaggio Assembly) ed è formata dai registri e dalla ALU che è l'esecutore aritmetico/logico.
Alcuni registri sono specializzati, cioè svolgono solo una specifica funzione mentre altri sono di uso generale, possono cioè essere usati dal programmatore per un qualsiasi scopo.
Le trasformazioni dei dati contenuti neri registri vengono effettuate attraverso elaborazioni della ALU.

MDR
Memory Data Register: è un registro interno, che si affaccia sul "data bus" attraverso un buffer bidirezionale tristate. E' invisibile al programmatore e contiene temporaneamente i dati campionati dalla CPU (lettura) o che vengono emessi dalla CPU (scrittura).

MAR
Memory Address Register: è un registro interno, che si affaccia direttamente sull' "address bus". E' invisibile al programmatore e contiene temporaneamente i gli indirizzi che la CPU ha elaborato e che devono essere emessi per selezionare di dispositivi periferici (memorie ed I/O).

IR
Instruction Register: è un registro interno, che riceve il codice operativo dal MDR durante la fase di interpretazione della istruzione (FETCH) si affaccia direttamente sull' "address bus". E' invisibile al programmatore e contiene temporaneamente il codice operativo della istruzione da eseguire.

PC
Program Counter: è un registro interno, che contiene in ogni fase di avanzamento del programma l'indirizzo di memoria in cui si trova l'istruzione da eseguire. Per questo motivo questo registro (ma anche molti altri che vedremo in seguito) viene anche chiamato **puntatore** (nelle CPU Intel si chiama Instruction Pointer → IP) nel senso che il dato contenuto nel registro si riferisce ("punta") ad un altro dato che si trova in memoria all'indirizzo corrispondente al valore contenuto nel puntatore. I seguenti modelli grafici servono per fissare visivamente questo nuovo concetto che è fondamentale per il funzionamento della CPU.



Viene incrementato automaticamente dalla CU (Control Unit) ogni volta che l'esecuzione di una istruzione è completata in modo da potere leggere in memoria l'istruzione successiva.
E' anche accessibile al programmatore che lo può usare per modificare il flusso sequenziale del programma. Infatti cambiando il valore di PC l'istruzione successiva viene prelevata da un diverso punto del programma (scelte, cicli, sottoprogrammi).

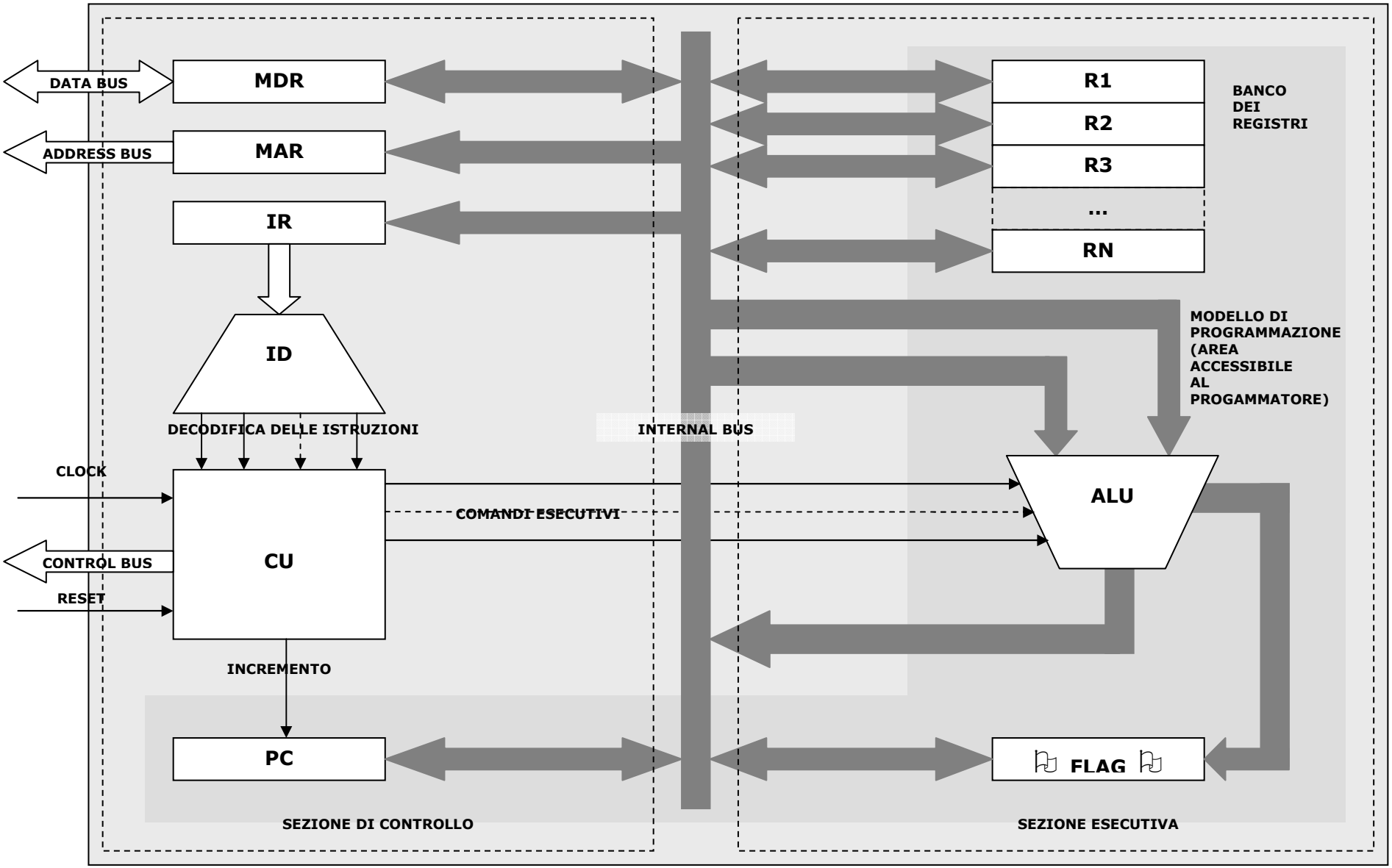
FLAG
Un flag (bandierina di segnalazione) è una informazione logica elementare (un bit) che fornisce informazioni sul risultato delle operazioni della sezione esecutiva.
Il registro dei flag quindi a differenza degli atri registri non ha un significato nel suo complesso ma ogni bit che lo compone ha un significato indipendente (ad esempio il bit ZF = Zero Flag segnala se il risultato della elaborazione è nullo o diverso da zero mentre il bit SF = Sign Flag segnala se il risultato della elaborazione è positivo e negativo. E' accessibile al programmatore che può avere informazioni sui flag e modificarli.

Banco dei registri
Il banco dei registri è un blocco di registri non specializzati, destinati ad ospitare temporaneamente i dati in corso di elaborazione; i dati contenuti nei registri provengono dalla memoria, vanno alla ALU per la elaborazione e da qui tornano ai registri da cui vengono riportati in memoria. Il banco dei registri è accessibile al programmatore. Ogni CPU ha una sua organizzazione del banco dei registri; in questo esempio generico è ipotizzata l'esistenza di N registri tutti uguali da R1 a RN.

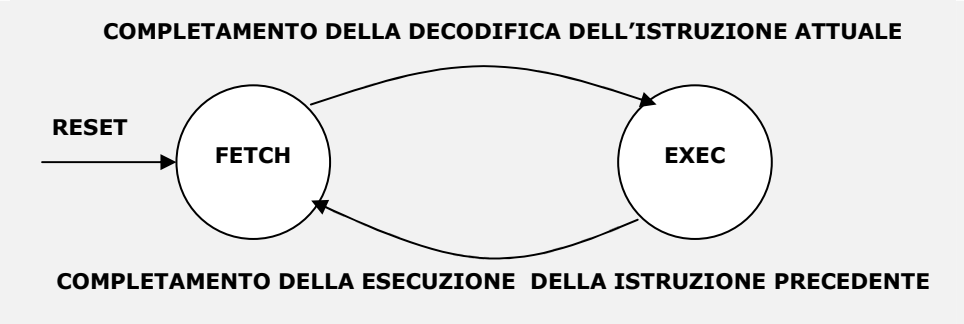
ID
Instruction Decoder: è il blocco che riceve il codice binario dell'istruzione (codice operativo o OPCODE) dall'IR e lo decodifica fornendo le informazioni all CU che deve attuare le azioni corrispondenti all'istruzione.

CU
Control Unit: è il blocco che, sincronizzato dal clock, genera i cicli di bus attraverso il control bus e invia i comandi esecutivi alla ALU in base alla decodifica dell'istruzione. In seguito all'esecuzione di ogni istruzione la CU incrementa l'indirizzo di memoria puntato dal PC in modo da predisposi all'esecuzione della istruzione successiva che si trova in genere all'indirizzo di memoria successivo rispetto all'istruzione precedente (programma sequenziale).

ALU
Arithmetic Logic Unit: è il blocco che esegue le trasformazioni sui dati. Poiché i dati, indipendentemente dal loro significato sono codificati attraverso numeri binari qualsiasi trasformazione da fare sui dati si riduce ad una operazione aritmetica o logica. Opera su due operandi che possono essere contenuti nei registri interni o provenire dalla memoria attraverso l'MDR e produce un risultato che può essere salvato su un registro interno o in memoria attraverso l'MDR.


DATA BUS
MDR
ADDRESS BUS
MAR
IR
ID
DECODIFICA DELLE ISTRUZIONI
CLOCK
CONTROL BUS
RESET
CU
INCREMENTO
PC
INTERNAL BUS
R1
R2
R3
...
RN
BANCO DEI REGISTRI
MODELLO DI PROGRAMMAZIONE (AREA ACCESSIBILE AL PROGAMMATORE)
COMANDI ESECUTIVI
ALU
FLAG
SEZIONE DI CONTROLLO
SEZIONE ESECUTIVA

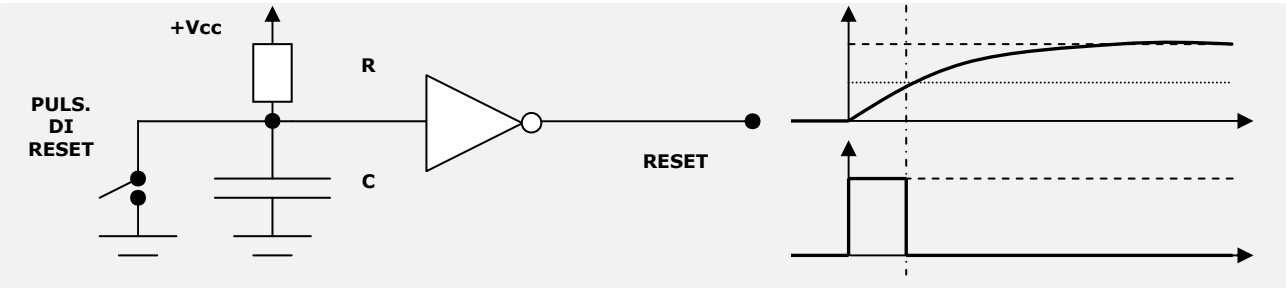
Automa esecutore
Per comprendere il funzionamento della CPU conviene partire da una automa a stati finiti molto semplice formato da due soli stati: FETCH ed EXECUTE.



Si può immaginare la CPU come un automa a stati finiti che si muove continuamente tra i due suoi stati:

- FETCH (INTERPRETAZIONE): Si entra in stato FETCH con l'evento "Completamento della esecuzione della istruzione precedente". Supponendo che PC stia puntando al codice operativo di una istruzione memorizzato in memoria centrale la CU comanda, attraverso i bus (PC copiato in MAR e RD+MEM attivi sul control bus), la lettura dalla memoria del codice operativo (OPCODE) della istruzione; questa informazione attraverso il MDR arriva all' IR che lo manda al circuito di decodifica ID per ottenere la decodifica della istruzione. Inoltre CU incrementa il PC in modo che al prossimo FETCH punti all'istruzione successiva e passa allo stato di EXECUTE.
- EXECUTE (ESECUZIONE): Si entra in stato EXECUTE con l'evento "Completamento della decodifica della istruzione attuale". Ottenuta la decodifica dell'istruzione la CU comanda ad ALU la parte esecutiva che può coinvolgere sia trasferimenti di bus (lettura di dati dalla memoria, scrittura di dati in memoria, acquisizione di dati dall'ingresso, emissione di dati sull'uscita) in cui l'altro estremo del trasferimento è un registro interno della CPU sia elaborazioni interne fra registri eseguite dalla ALU che portano a risultati memorizzati sui registri. Esistono anche elaborazioni miste che coinvolgono contemporaneamente entrambe le attività (ad esempio una somma tra il contenuto di un registro ed il contenuto di una cella di memoria che viene reso disponibile sul MDR mediante una lettura con risultato in un registro). Al termine della esecuzione l'automa si riporta in FETCH per interpretare l'istruzione successiva il cui indirizzo si trova in PC. E' possibile però che l'azione esecutiva abbia nel frattempo modificato il contenuto di PC (scelte, cicli, sottoprogrammi)

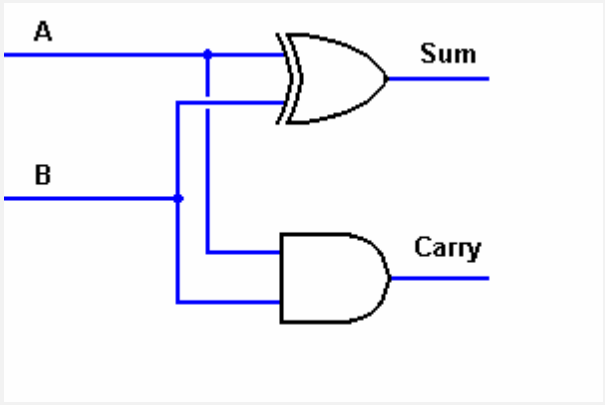
Come in ogni automa è indispensabile la definizione dello stato iniziale. Nel caso della CPU lo stato iniziale viene generato dal segnale esterno "RESET". Questo segnale di ingresso, che può essere generato da un apposito circuito di reset attivato dall'accensione del computer o dal pulsante di reset, forza la CPU in stato di FETCH ed inizializza il registro PC all'indirizzo del codice operativo della prima istruzione contenuta nel programma di bootstrap in ROM (non è detto che questo indirizzo sia 0 ma varia in funzione del tipo di CPU, ad esempio per le CPU Intel è FFFFFFFFFFFFFFF0 cioè nella parte più alta della memoria).
In questo modo ad ogni reset la CPU è obbligata a ripartire in fetch dalla prima istruzione del suo programma di caricamento.



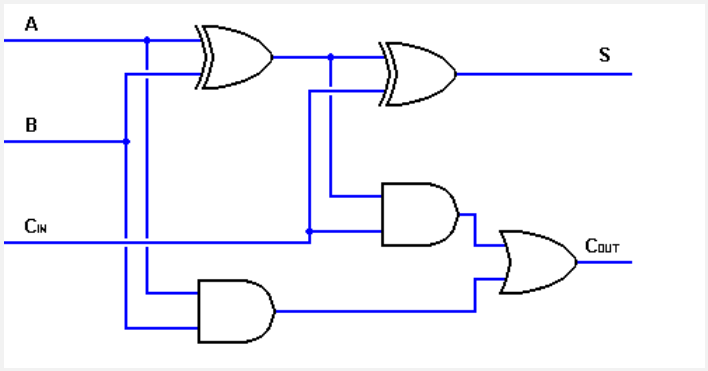
Questo modo di funzionare spiega perché non ci sia una distinzione in memoria tra codici operativi di istruzioni e variabili che sono memorizzati tutti insieme. E' il succedersi dei fetch sulle istruzioni che da significato di codici operativi ai dati contenuti in memoria tanto è vero che se per un qualche motivo questa sincronizzazione viene persa (ad esempio a causa di un errore funzionale di un programma) la CPU può passare ad eseguire il fetch su dati qualsiasi che non sono in effetti codici operativi e questa situazione porta ben presto ad errori fatali che a causa del sistema di protezione attuato dal sistema operativo in genere si traducono nel messaggio "Questo programma ha eseguito una istruzione non valida …".
Attenzione: non è la CPU che si accorge di questa situazione ma un meccanismo di gestione della memoria attuato dal sistema operativo di cui parleremo più avanti.
E' possibile anche simulare un reset via software caricando in PC l'indirizzo di bootstrap ma questa operazione non è esattamente coincidente con l'azione del segnale hardware di reset perché quest'ultimo inizializza l'intero hardware della CPU compreso il contenuto di tutti i registri interni.

ALU

In questo paragrafo viene analizzato nel dettaglio il funzionamento del blocco ALU
La ALU è la parte della CPU dedicata alle operazioni aritmetiche e logiche.
Poiché le informazioni elaborate dai programmi, indipendentemente dal loro significato, sono tutte codificate in forma numerica in realtà tutte le elaborazioni passano attraverso la ALU perché qualunque operazione può essere ricondotta ad una operazione aritmetico/logica.
Un paio di esempi:
ricordando che i caratteri alfabetici sono codificati nella codifica ASCII in ordine progressivo (A=41H, B=42H, C=43H ... a=61H, b=62H, c=63H …) un ordinamento alfabetico diventa un confronto di maggioranza/minoranza numerica (con le maiuscole che precedono le minuscole) mentre la trasformazione di un carattere da maiuscolo a minuscolo è una operazione di AND (A=41H OR 20H → 61H =a, a=61H AND DFH → 41H=A).
Sebbene le ALU siano circuiti logici molto complessi, da un punto di vista di principio se ne può realizzare un modello estremamente semplice. Si parte dal concetto di "insieme minimo" di operazioni che definisce l'insieme di operazioni indispensabili per potere costruire una qualunque operazione in un certo dominio di valori.
Questo concetto è già stato affrontato in logica elettronica dove si è visto che una qualunque operazione può essere realizzata disponendo di un insieme di operatori (sono insiemi minimi della logica le coppie AND+NOT, OR+NOT e gli operatori NAND e NOR).
Lo stesso concetto può essere esteso anche all'aritmetica individuando come insieme minimo la coppia "addizione"+"negazione". Attenzione a non confondere la "negazione" detta anche "complemento-a-due" con l'operatore logico NOT detto anche "complemento-a-uno". La negazione svolge nell'aritmetica binaria le funzioni di cambio di segno in assenza di un simbolo dedicato al segno.
Ogni altra operazione numerica può essere concettualmente derivata da questa coppia. Ad esempio una sottrazione è la combinazione di una "negazione" del "sottraendo" e di una "addizione" con il "minuendo" cioè a-b → a+(-b), la "moltiplicazione" è una ripetizione di "addizioni" del "moltiplicando" tante volte quanto indicato dal "moltiplicatore", la "divisione" è una ripetizione di "sottrazioni" del "divisore" dal "dividendo" fino a che il divisore è ancora contenuto nel dividendo avendo come "quoziente" il numero di ripetizioni e così via ...
Si può quindi immaginare la ALU come costituita solo dalla combinazione di operatori AND,NOT,addizionatori e negatori.
Nello schema che segue viene mostrata la struttura di principio di un addizionatore (HALF ADDER):

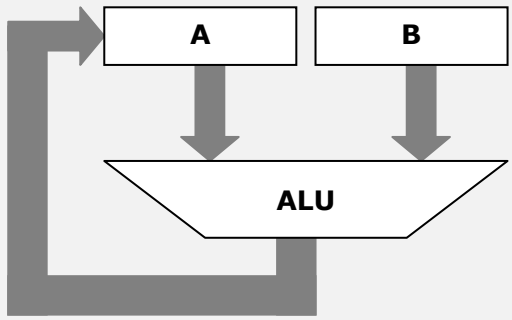


Lo schema si riferisce ad un singolo bit e va ripetuto per ogni bit del dato sommando il riporto al bit di peso superiore (FULL ADDER). Il sommatore diventa così a tre bit (un bit del primo operando, un bit del secondo operandoe ed il bit di carry dell'adder di peso inferiore) ed è rappresentato nello schema successivo.



L'operatore negatore può essere pensato, come si è già visto lo scorso anno dalla combinazione dell'operatore NOT (complemento-a-uno) e dell'addizionatore. Infatti il complemento-a-due è definito operativamente come complemento-a-uno + uno.

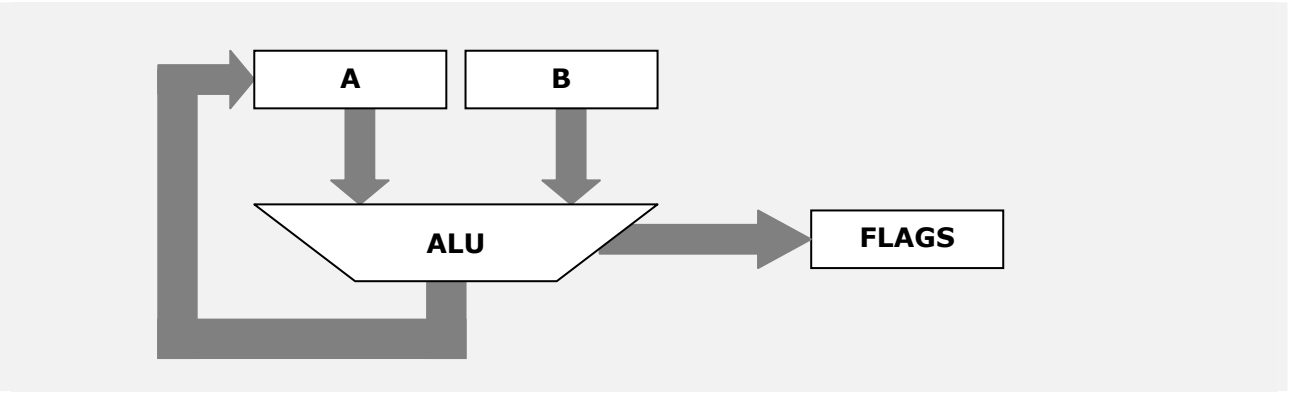
La ALU si comporta in genere da "operatore binario ad accumulo". Il termine "binario" in questo caso non sta ad indicare che i dati sono in codice binario ma che la ALU esegue le operazioni a partire da due operandi. Il termine "accumulo" sta ad indicare che il risultato dell'operazione va a finire in uno dei due operandi. In questo modo nell'operando che è dapprima origine e poi destinazione dei dati si forma un accumulo di dati.
Esempio: supponiamo di avere due registri A e B che svolgono le funzioni di operandi per una ALU e supponiamo che il risultato vada a finire in A. Se in A c'è il valore 1 e in B in valore 1 effettuando la somma A+B (1+1) il valore di A diventa 2, effettuando ancora la somma A+B (2+1) il valore di A diventa 3 e cosi via ... Il registro A si sta comportando da accumulatore, con il suo valore che è risultato di una operazione ed operando della successiva. In questo modo si possono realizzare delle operazioni concatenate con pochi trasferimenti di dati ottenendo presestazioni superiori.
Lo schema della esempio precedente è descritto dal seguente modello:



Gli operatori della ALU (aritmetici e logici) operano in parallelo in modo da elaborare contemporaneamenti tutti i bit omologhi dei due operandi producendo un risultato che ha la stessa ampiezza dei due operandi.
In genere, ma non sempre, l'ampiezza della ALU è uguale all'ampiezza del data bus.
Se una operazione ha più di due operandi, ad esempio A+B+C deve essere spezzata in due operazioni A+B che produce il risultato ad esempio in A e successivamente A+C applicando quindi la proprietà distributiva della somma.
Esistono anche operazioni "unarie" come la complementazione (NOT), la negazione, lo shift … che hanno un unico operando di ingresso.

FLAGS

Oltre al risultato della operazione la ALU produce anche altre informazioni logiche dette FLAGS. Queste informazioni formano un pacchetto di bit che riflettono con il loro stato informazioni sul risultato della ultima operazione aritmetico logica eseguita.
Quindi questi bit si aggiornano ad ogni operazione aritmetico/logica mentre restano inalterati al valore della ultima operazione aritmetico/logica mentre la CPU fa altri tipi di operazione che non coivolgono la ALU (ad esempio trasferimenti sul bus).
Queste informazioni sono essenziali per la costruzione degli algoritmi dei programmi. Infatti esiste una ampia gamma di istruzioni che valuta il contenuto dei flags ed effettua in funzione dello stato di un flag delle variazioni del contenuto del PC consentendo quindi di alterare il flusso sequenziale del programma per realizzare i costrutti di scelta ed iterazione e le chiamate ai sottoprogrammi.
Lo schema completo della ALU quindi diventa:



I principali flags sono:

- **ZERO (ZF)**: è vero quando il risultato dell'ultima operazione vale 0 mentre è falso quando il risultato dell'operazione è diverso da 0. E' il flag più usato. Ad esempio il costrutto if (a==b) {...} diventa a livello di CPU una sottrazione tra a e b per verificare se il risultato della sottrazione da zero (a e b uguali) controllando se ZF è vero. Per evitare confusioni usiamo sempre i termini "vero" e "falso" quando ci si riferisce ai flag evitando di usare i termini 1 e 0.
- **CARRY (CF)**: è vero quando c'è stato un riporto dal bit più significativo del risultato. Le operazioni di somma e le operarazioni derivate da questa (sottrazione, moltiplicazione ...) possono determinare un riporto da un bit al bit di peso successivo (vedere FULL ADDER). Se la somma del bit più significativo determina un riporto questo bit non può essere sommato al bit successivo e viene memorizzato nel CF per segnalare che l'operazione nel suo complesso ha determinato un riporto. Ci sono due impieghi principali: come nel caso precedente per i costrutti per testare maggioranza e minoranza tenendo conto che la sottrazione tra due numeri provoca un prestito (riporto negativo) se il minuendo è minore del sottraendo if (a>b) {...} diventa a livello di CPU una sottrazione tra a e b per verificare se a>b è vero (CF falso) o falso (CF vero), un altro caso di impiego è costituito dalle operazioni in cascata da effettuare qunado il dato supera la precisione della ALU (ad esempio dato a 64 bit in una ALU a 32 bit).
- **SIGN (SF)**: è vero quando il risultato è negativo. Materialmente viene riportato nel SF il bit più significativo del risultato che riflette il segno.
- **OVERFLOW(OF)**: è vero se il segno del risultato discorda dal segno degli operandi. Segnala la presenza di un overflow rispetto alla precisione; infatti se si sommano due operandi positivi il risultato dovrebbe essere sempre positivo, se il bit più significativo del risultato è negativo significa che il risultato ha un'ampiezza superiore alla capacità del risultato; la stessa cosa vale per due numeri negativi mentre se i due numeri hanno segni discordi un overflow non si può verificare quindi OF diventa sempre falso.
- **PARITY (PF)**: è vero quando il numero di bit a 1 del risultato è pari. Non riguarda quindi la parità aritmentica ma la parità logica e viene usato negli algoritmi di comunicazione per la gestione del rilevamento e correzione degli errori.