

ARCHITETTURA DEGLI ELABORATORI

MA TIPO SOLO LE COSE IMPORTANTI PERCHÈ MI STO FACENDO UNA CAPA TANTA E NON HO NEMMENO INCLUSO BASI E CONVERSIONI, SAD

COMPONENTI DI UN COMPUTER

UNITÀ DI CONTROLLO decodifica le istruzioni e le invia alla ALU

UNITÀ DI INGRESSO → SCRIVE → MEMORIA → LEGGE → UNITÀ DI USCITA

REGISTER FILE insieme di registri letti o scritti fornendo il numero del registro

ALU svolge le operazioni aritmetico-logiche:

- recupera i dati;
- li processa nell'ACCUMULATORE
- salva nel registro di uscita

PROGRAM COUNTER conserva l'indirizzo di memoria della prossima istruzione, è un puntatore.

È in ciclo fetch-execute:

- caricamento dell'istruzione
- aggiornamento
- esecuzione dell'istruzione caricata

TIPI DI ISTRUZIONI MIPS

R-Type OP - RS - RT - RD - SHAMT - FUNCT

I-Type OP - RS - RT - IMMEDIATO

J-Type OP - JUMP ADDRESS

DIMENSIONI DI MEMORIA

MAR 32 bit

BYTE 8 bit

HALF-WORD 16 bit

WORD 32 bit

RETI

COMBINATORI le uscite dipendono esclusivamente dalle entrate

SEQUENZIALI le uscite dipendono sia dalle entrate che dallo stato interno del circuito

DECODER n bit in ingresso

2^n uscite

Tra due l'ingresso nel suo valore decimale, quindi una sola uscita può essere 1, tutte le altre 0.

MULTIPLEXER n ingressi

$\log_2 n$ selettori

la sua uscita è uguale ad uno degli ingressi, scelto mediante un segnale di controllo. Seleziona quale segnale in ingresso mandare in uscita.

È composto da:

- un decoder che genera n segnali, corrispondenti ai valori dell'ingresso di selezione
- n porte AND, combina gli ingressi con i segnali del decoder
- porta OR che raccoglie le uscite delle porte AND

PLA implementazione della somma di prodotti. Composto da

- ingressi e relativi valori complementati
- porte AND
- porte OR

ROM read-only-memory, in cui ogni ingresso (=indirizzo) corrisponde un'uscita (=contenuto della cella di memoria di quell'indirizzo)

- N input
- memory height: 2^h celle di output
- memory width: N bit ↗

PROM rom programmabili

EPROM rom programmabili e cancellabili

PLA VS ROM

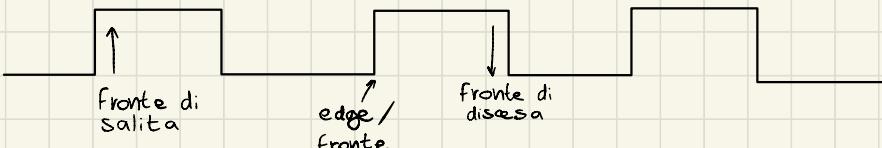
- Rom fully decoded, PLA partially decoded
- Rom di dimensione più grande
- PLA più efficienti
- Rom possono implementare qualsiasi funzione logica, senza modificare la dimensione

SEGNALE DI CLOCK

Servono per determinare quando debbono essere aggiornati gli elementi che contengono informazioni di stato.

Evvole con un **PERIODO** determinato e costante

FREQUENZA DI CLOCK inverso del periodo



Tutti i cambiamenti di stato avvengono in corrispondenza di un fronte.

SR-LATCH Set and reset. Composto da due porte NAND o due porte NOR con collegamenti incrociati.

I bit immagazzinato è portato all'uscita **Q**
e a **\bar{Q}** suo complemento

D-LATCH ingressi: D valore da memorizzare

C segnale di clock \rightarrow indica quando leggere e memorizzare D

Stato: **APERTO** clock asserted, Q assume valore D

CHIUSO clock non affermato, Q rimane fissato all'ultimo valore memorizzato

D-FIP-FLOP viene attivato sul fronte di salita o di discesa del clock.

L'uscita viene memorizzata sul fronte del clock, quindi D deve rimanere valido in un opportuno intervallo di tempo immediatamente prima e dopo il fronte del clock.

REGISTER FILE

Insieme di registri che possono essere letti o scritti, fornendo il numero di registro a cui fare accesso.

Realizzato con:

- decodificatore per ciascuna porta di lettura o scrittura
- matrice di registri realizzati con flip-flop di tipo D.

LETTURA DI UN REGISTRO numero del registro → dato contenuto nel registro

SCRITTURA DI UN REGISTRO numero di registro

dato da scrivere

segnale di clock

È quindi composto da:

READ REG #1 primo registro da leggere

READ REG #2 secondo registro da leggere

READ DATA 1 valore del primo registro letto

READ DATA 2 valore del secondo registro letto

WRITE REG # numero del registro su cui scrivere

WRITE DATA valore del registro da scrivere

ALCUNE RAM

SRAM static RAM, usata per realizzare memorie veloci. Non è possibile scrivere e leggere contemporaneamente

DRAM dynamic RAM, è più capiente ma più lenta. Ogni bit è memorizzato tramite un condensatore che mantiene la carica per pochi millisecondi, è quindi necessario effettuare un **REFRESH** che legge i valori e li riscrive subito dopo.

PROCESSO DI COMPILAZIONE

COMPILATORE alto livello → linguaggio assembler

ASSEMBLER linguaggio assembler → linguaggio macchina

} l'intero processo è la **COMPILAZIONE**

LINKE collega tra loro vari moduli che compongono lo stesso programma

- il programma sorgente è suddiviso in più file che vengono compilati separatamente creando diversi **FILE OGGETTO**

↳ il linker li collega, producendo un file contenente il codice eseguibile

In particolare:

- pone simbolicamente in memoria istruzioni e dati
- determina gli indirizzi dei dati e delle etichette dei salti
- Unisce i riferimenti interni ed esterni delle procedure

LOADER si occupa di far partire il programma:

- determina dimensione di segmento testo e segmento dati
- copia istruzioni e dati in memoria
- copia nello stack i parametri del main, se presenti
- inizializza registri e stack pointer
- invoca la procedura main

MACCHINE A STATI FINITI

Usate per descrivere i circuiti sequenziali:

Composte da:

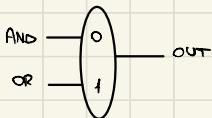
- un set di stati
- due funzioni:

- **NEXT STATE** determina lo stato successivo partendo dallo stato corrente e dai valori in ingresso
- **OUTPUT** produce un insieme di risultati partendo dallo stato attuale e dai valori in ingresso

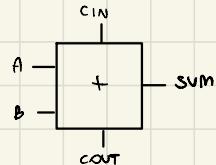
ALU 1 BIT e 32 BIT

Implementa operazioni aritmetiche e logiche, come di seguito:

- **NOT** effettuato su ognuno dei due ingressi A e B, tramite una porta NOR
- **AND/OR** effettuato sugli ingressi, l'operazione da fare è scelta tramite un selettori



- **ADDITIONE** somma A e B, considerando anche il carry
FULLY-ADDER in grado di gestire il ripporto.



- **SOTTRAZIONE** si ottiene negando B in CA2. B è negato tramite la linea di controllo detta **B-INVERT** (anche A ha la **A-INVERT**)
- **NOR** Si implementa con le **LEGGI DI DE MORGAN**
 - $\overline{A+B} = \overline{A} \cdot \overline{B}$
 - $\overline{A \cdot B} = \overline{A} + \overline{B}$

- **OPERAZIONI DI CONFRONTO** tra A e B:

- **SLT** set-on-less-then se $a < b \rightarrow 1$, altrimenti 0
 $a < b \Leftrightarrow a - b < 0$ nella ALU-31 (MSB)
- **BEQ** branch-on-equals, confronto di uguaglianza
 $a - b = 0 \Leftrightarrow a = b$, utilizzo della sottrazione

ALU A 32 BIT

Suddivise come:

- 01-30 - $c_{in} = c_{out}_{i-1}$ (carry precedente in)
 - less = 0
- 00 - $c_{in} = 1$ (per la sottrazione)
 - less
- 31 (MSB) - $c_{in}_{31} = c_{out}_{30}$
 - less = 0
 - gestisce i casi di overflow

CARRY LOOKAHEAD anticipare i bit di carry in , utile perché aspettare i riporti comporta un ritardo

Per farlo: $c_{out} = a \cdot cin + b \cdot cin + a \cdot b = (a+b) \cdot cin + a \cdot b$, generalizzando:

$$c_{i+1} = (a_i \cdot c_i) + (b_i \cdot c_i) + (a_i \cdot b_i) = \\ = g_i + p_i \cdot c_i$$

\downarrow \rightarrow i-esimo elemento

generate \nwarrow propagate

Quindi:

- se $g_i = 1$, carry out indipendente dal carry in
- se $p_i = 1$, non generato ma propagato dal carry precedente