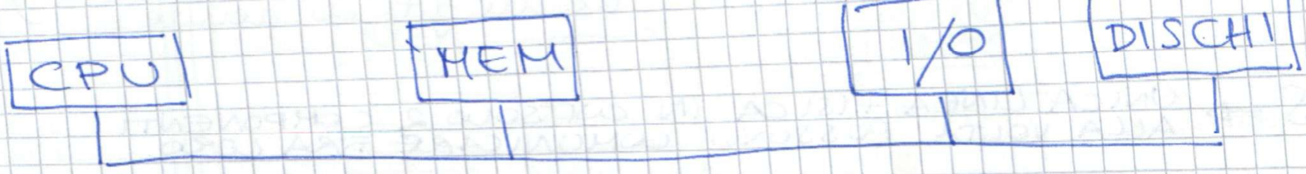
una descrizione del calcolatore ... strutturale

cuore del processore (vi è anche una parte di memoria) secondo Neumann ie CPU elabora le istruzioni base del programma

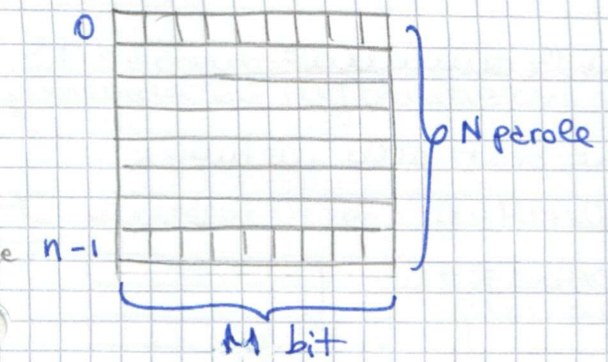
memoria principale: banca dati per memorizzare sia le istruzioni del programma (Algoritmo) che i dati.
[RAM] → Random Access Memory

input/output
varie periferiche



La memoria contiene informazioni in binario e contiene cellette di 1 bit

PAROLA: sequenza di bit che rappresentano un'informazione



tante parole tutte di M bit
{nell'architettura le parole sono tutte lunghe uguale}

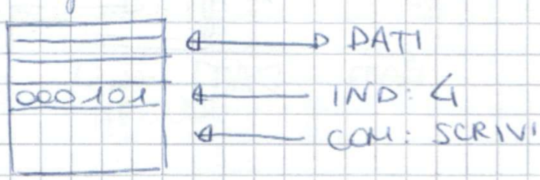
LA MEMORIA È GRANDE M · N

PORTE UNIDIREZIONALI: In genere la memoria ha una porta chiamata INDIRIZZO (es. voglio la parola 3) e una chiamata COMANDO (leggi/scrivi)

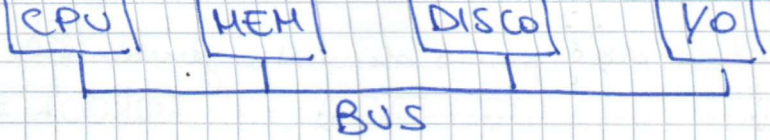
PORTA BIDIREZIONALE: porta Dati

es. voglio leggere la parola 2

es voglio scrivere 000101 nella parola 4



$\log_2 N$ per l'INDIRIZZO: di quanti bit abbiamo bisogno



BUS: serie di linee di rame, collegamenti fisici che mettono in comunicazione le parte
(linee condivise a K bit)

{ ciascuna porta ha n bit diversi per poter trasmettere l'informazione intera e non sequenziale
↓
ogni filo è 1 bit quindi ho n fili di rame }

BUS → UNICA LINEA FISICA IN CUI SOLO 2 COMPONENTI ALLA VOLTA POSSONO COMUNICARE FRA LORO

(questa architettura è la più flessibile possibile)

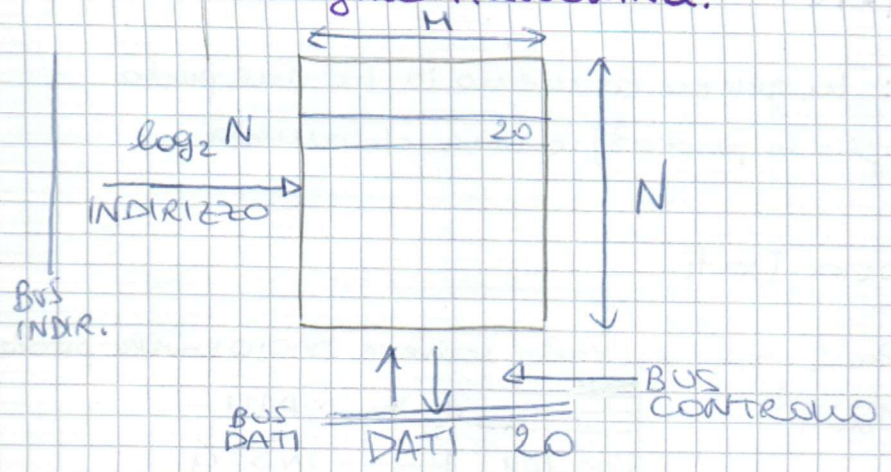
BUS → 3 gruppi di linee:
- bus DATI (bidirezionale)
- bus INDIRIZZO } monodir.
- bus COMANDO }

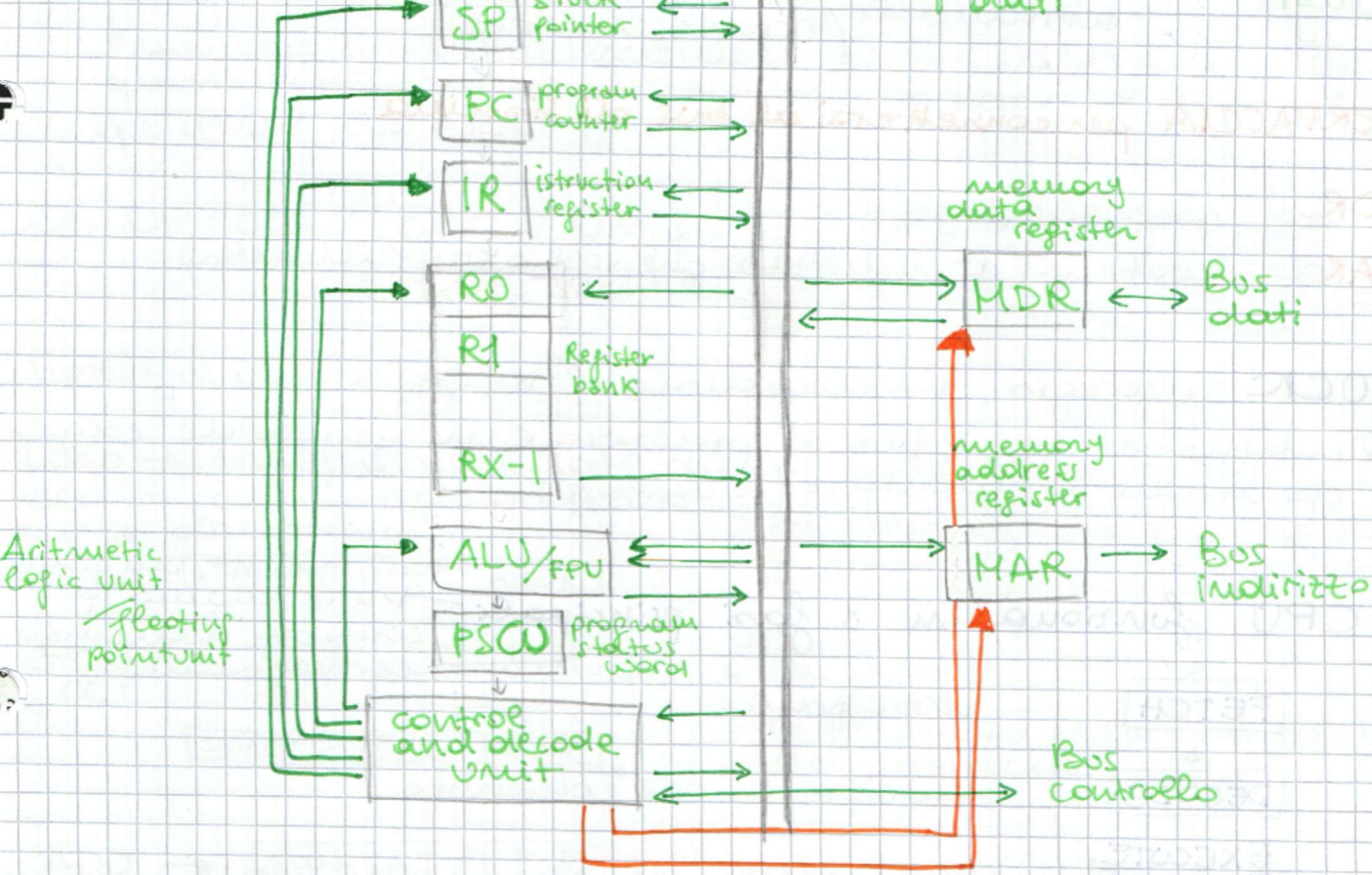
la porta indirizzo è connessa al bus indirizzo

MASTER: unità che utilizza il bus, che "comanda"
↓
CPU → unità che decide di chiedere dati alla memoria

SLAVE: chi riceve le richieste

immagine riassuntiva:





HO ACCESSO DIRETTO A TUTTE LE CELLE (NO SEQUENZIALE)

Il CPU è composto da una serie di registri, memorizza la stessa quantità di informazione di una riga di memoria

PC è memorizzato in MEMORIA come una serie di istruzioni

ESECUZIONE SEQUENZIALE (dal I all'ultimo)
(per accedere alla memoria)

IR → è un registro della CPU
memorizza le istruzioni che stiamo eseguendo
(che è stata prelevata in quell'istante dalla MEM)

REGISTER BANK in genere 32 bit, contiene X parole diverse, che sono gli elementi che stiamo elaborando.

ALU è in grado di eseguire operazioni algebrico-logiche
(somma, sottrazione...)

la ALU ha 2 uscite < risultato → ||
di STATO → PSW

PSW contiene informazioni sul risultato:
se c'è overflow, se è un numero positivo o negativo,

control & decode unit — componente che comunica con tutti gli altri attivandoli (CONTROLLA ANCHE IL BUS)

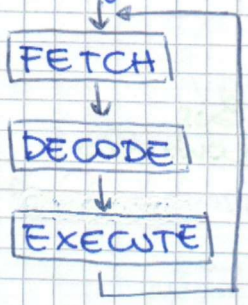
INTERFACCIA per connettersi al bus di sistema

MDR

MAR contiene l'indirizzo che vogliamo richiedere

CLOCK: orologio che scandisce il tempo; all'interno di ogni tick il processore è in grado di eseguire un'operazione elementare (micropassaggi)

Il CPU funziona in 3 fasi principali → questo ciclo si ripete all'infinito, da quando accendiamo il computer a quando lo spegniamo



estrazione

interpretazione del contenuto

comando dei vari componenti

PROGRAMMA: una serie di istruzioni una in fila all'altra

I FASE: 4 operazioni controllate dalla CPU

PC → MAR

Mem[MAR] → MDR

MDR → IR

{ PC → ALU → PC (PC+1) ; (PC+1) → PC } → per continuare con la sequenza con quello addizionato di 1 cioè per tornare al PC e RICOMINCIARE IL PERCORSO

NON SONO ISTRUZIONI, SONO REALIZZATE DALLA CIRCUITERIA DI CONTROLLO

II FASE: intanto in IR ho i miei dati

IR va al control unit & decode unit per capire cosa fare

IR → CONTROL & DECODE UNIT

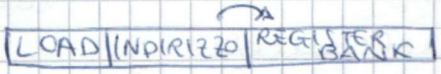
↳ numero in codifica binaria naturale, in base a questo CODE l'unità di controllo sa capire il tipo di operazione da eseguire

es. OP CODE: 000 = lettura
111 = scrittura

→ bit dedicati al OP CODE $\log_2 N$ (N = n° di istruzioni, arrotondato all'eccesso)

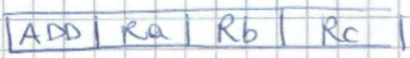
Tipologie di codice

LOAD → trasferisce un dato da una cella di memoria ad un register bank → La macchina non è in grado di lavorare direttamente sui dati nella memoria
000



STORE → fa il processo inverso alla load
001

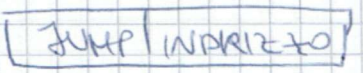
trasferisce un dato dal bank alla memorie

ADD → 

010

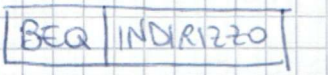
è una somma
$Ra + Rb = R_{r1}$
$R_{r1} + R_c = R_{r2}$

JUMP → istruzione per manipolare il program counter



esegue sempre un salto

BEQ → Branch if equal



(es. salta se all'interno del PSW il bit = 0 se no, continua)

SALTA SOLO SE AVVIENE UNA DETERMINATA POSSIBILITÀ

III FASE:
Vengono eseguite le operazioni comandate dall'unità di controllo

| 0110 | 00001 | 00010 | 00000 |

→ la ALU fa la somma e scrive il risultato nel registro 0 } SCRITTURA

ADD

OP_1 è il registro 1 — OP_2 registro 2 — LETTURA

| LOAD | IND | Rx |

$OP_1(IR)$ $OP_2(IR)$

$OP_2(IR) \rightarrow MAR$

$MEM[MAR] \rightarrow MDR$

$MDR \rightarrow REG[OP_2(IR)]$

| STORE | Rx | IND |

$OP_1(IR)$ $OP_2(IR)$

numero registro banco da cui prendo i dati

$REG[OP_1(IR)] \rightarrow MDR$

$OP_2(IR) \rightarrow MAR$ → inviamo l'indirizzo al mar

$MDR \rightarrow MEM[MAR]$ la III parte della scritta si invia al mar

→ Dall'IR si passa alla decode che instrada i bit

L'informazione dell'IR va contenuta nel decode

↳ L'informazione da cui si parte tutto è nell'IR

| JUMP | INDIRIZZO |

$OP_1(IR) \rightarrow PC$

Noi lavoriamo in sequenza finché non cambiamo l'indirizzo della cosa da leggere con Jump o Beq

Indirizzamento a registro: l'operando è contenuto nel registro

| ADD | R_1 | R_2 | R_3 |

Se voglio lavorare con una costante utilizzo una variante

| ADDI | R_1 | C | R_2 |

$REG[OP_1(IR)] + OP_2(IR) \rightarrow REG[OP_3(IR)]$

questo non è l'indirizzo di un registro, ma è già il valore che voglio sommare

"INDIRIZZAMENTO IMMEDIATO"

è indicato direttamente nell'istruzione
per una LOAD o STORE

LOAD	IND	R_x

INDIRIZZAMENTO INDIRETTO

ADD	IND_MEM	R_2	R_3

è l'unione della LOAD e la ADD

solitamente è contenuta nelle

CISC
complex istruction set computer

l'opposto è le RISC

ND INDICIZZATO

operan

Il processore è in grado di eseguire le istruzioni macchina (le più semplici) - ASSEMBLER

C linguaggio che permette di specificare in modo rigoroso il nostro algoritmo

↓

OMPILATORE è in grado di convertire da questa specifica comprensibile ad una comprensibile alla macchina

↓

rogramma in singole istruzioni assembler } IN SET DI ISTRUZIONI RIDOTTO

X

X l'operando si recupera dalla memoria calcolando - - -

ADD COST(R_1) R_2 R_3

$(C_{COST} + R1) + R_2 \Rightarrow R_3$

OFFSET = DISCOSTAMENTO

al cambiare del contenuto di R_1 ip mi sposterò sempre di c linee più in giù

SP contiene l'indirizzo

qual è il FRAME di memoria in cui sono contenuti in quell'esecuzione i dati di programma

INDIRIZZAMENTO ASSOLUTO ALL'INTERNO DEL FRAME

LOAD 100(R_1) R_2

{ il dato si trova sempre 100 linee dopo l'inizio dei dati del mio programma

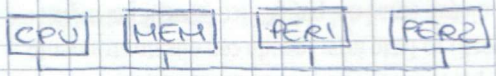
Es.) programma che esegua $(a+5) \times (c+d)$

il computer non sa farlo in 1 passaggio, ma deve scomporlo in micropassaggi eseguibili in memoria

200	LOAD 1000 R_0	SEZ. ISTRUZIONI
201	LOAD 1001 R_1	
202	ADD R_0 R_1 R_2	
203	LOAD 1002 R_0	
204	LOAD 1003 R_1	
205	ADD R_0 R_1 R_3	
206	MULT R_2 R_3 R_0	
	STORE R_0 1004	
1000	2	SEZ. DATI
1001	4	
1002	5	
1003	1	
1004	30	

memoria tutti i dati

RB solo i dati effettivi in quel momento (che stiamo elaborando)



POLLING — CONTROLLO DA PROGRAMMA

Periodicamente si interrompe il programma e si controlla le richieste da parte delle periferiche con un altro programma di servizio (ROUTINE)

INTERRUPT

Il bus di controllo ha anche una linea che va in direzione opposta che può essere sfruttata dalla periferica per comunicare direttamente alla CPU che ha bisogno di essere letta o scritta (con il comando interrupt)

a quel punto la CPU esegue un salto dal flusso sequenziale del programma (vengono salvati i registri nel frame) come una fotografia

A quel punto il PC viene salvato in memoria e viene caricato al suo posto l'indirizzo di routine

① SI FA UN SALTO ALL'ISTRUZIONE DI ROUTINE

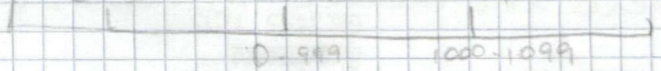
② si comunica con la periferica, la routine legge e scrive i registri per comandare la periferica stessa

BUFFER in cui la periferica deve porre i dati
AREA DI MEMORIA ABBASTANZA LIMITATA

③ Il programma prende i dati dalla memoria periferica e li spedisce nella MEM affinché nella mem periferica BUFFER si possano sovrascrivere altri dati

Comandi la periferica leggendo e scrivendo appositi registri comandi

DMA



Direct memory access — piccolo componente hardware programmabile, avente un master ed è in grado di trasferire dati da periferica a memoria e viceversa

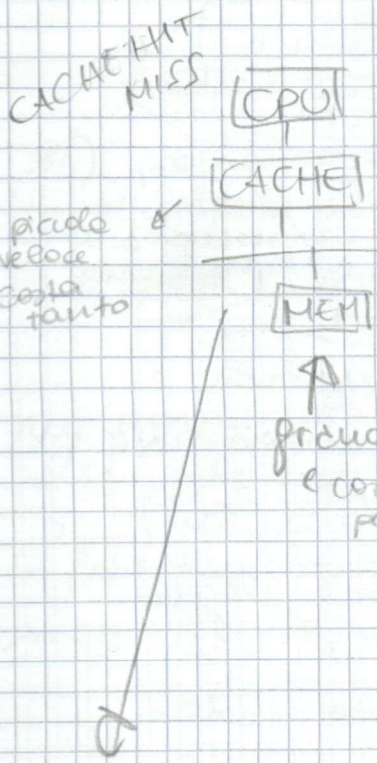
④ Poi prima di uscire dalla routine recupero la fotografia del mio programma principale e ricomincio

il programma principale e il DMA quello di routine → CONFLITTO SUL BUS, la precedenza va al DMA

Terminata la routine il DMA da un altro segnale di interrupt e la CPU ricomincia a lavorare

come si possono accelerare le cose)

GERARCHIA DI MEMORIA



realizzata con una tecnologia molto più veloce, solitamente è contenuta nel processore

Riusciamo ad utilizzare un piccolo pezzo di CACHE

Memorizziamo in CACHE solo i dati che stiamo utilizzando in quel momento

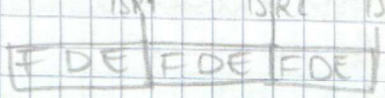
Così ottengo una memoria grande e veloce ed economica

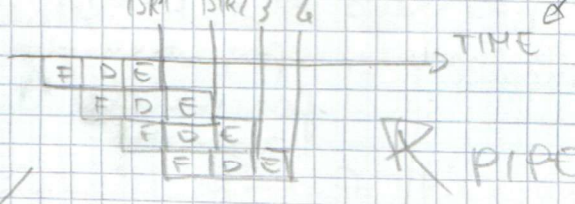
la prima volta che vado che nella CACHE non c'è un dato copio un BLOCCO da MEM a CACHE in cui c'è anche quel dato

(→ se CACHE è piena viene eliminato un blocco (trasferito alla memoria se il blocco era stato modificato) e viene preso il nuovo blocco

Il processore non vede niente, se nella CACHE manca un dato lo recupero lei dalla memoria nel nuovo blocco

Nella CACHE teniamo i dati più frequentemente utilizzati

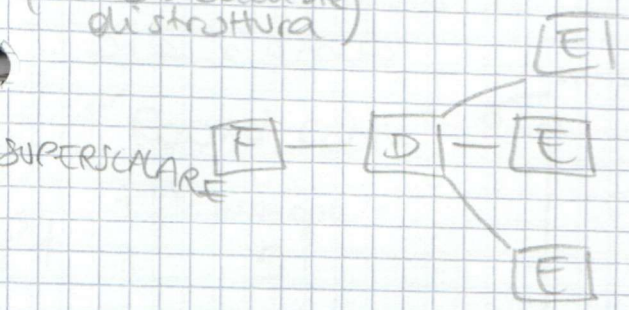




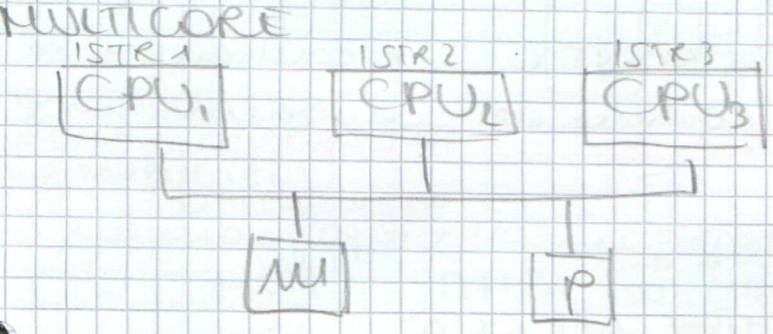
PIPELINE

Per poter realizzare questa struttura dobbiamo duplicare tutti gli elementi che sono condivisi dai 3 stadi per non creare conflitto

Sdoppio le PORTE e la ALU
(le uniche cambiate di struttura)



le istruzioni eseguono eseguite in parallelo ma il problema è che i risultati potrebbero finire non in ordine corretto, quindi serve poi riordinare.



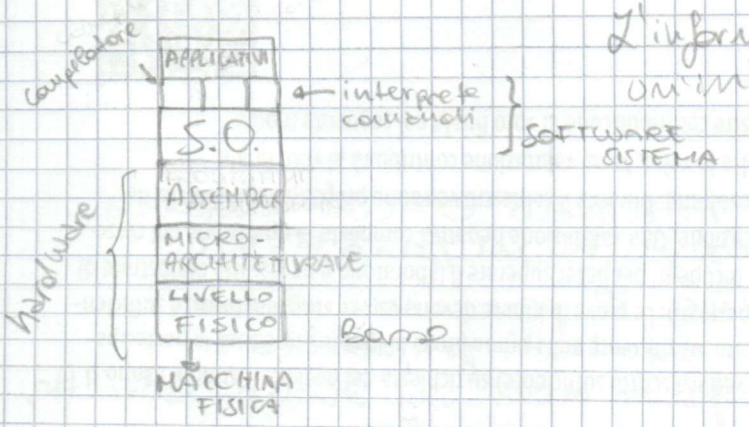
ogni CPU ha la sua CACHE

S.O.

risorse condivise
CPU
MEM principale
I/O
DISCO

è stato introdotto il sistema operativo che gestisce le risorse fornendo una versione più astratta

Livello di astrazione



L'informatica ha un'interfaccia e astratti

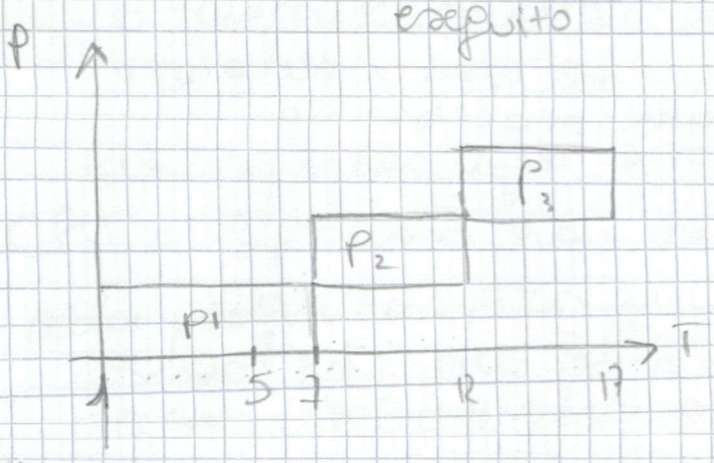
nocciolo
Il cuore del S.O. si chiama KERNEL e contiene tutte le funzioni di base per la gestione dei componenti

PROCESSO: serie di istruzioni come il programma e in più è rappresentato dallo stato - tutti i dati che lo sta elaborato il programma

non solo informazioni di istruzioni, ma tutti i dati elaborati, è contenuto nelle celle di memoria dedicate ad un programma (frame

Time sharing -> la CPU è condivisa nel tempo

SCHEDULING → utilizza un parametro per dare la precedenza
↳ elenco dei processi che deve essere eseguito



1 → 7 P1 running
P2, P3 ready
7 → 12 P2 running, P1, P3 ready

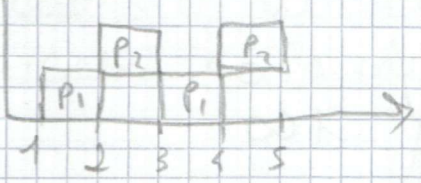
"First come first serve" → usato negli anni 70' 80', ma è inaccettabile (es sistemi Batch)

che è usato "Round robin"

nei sistemi di tipo interattivo

per un istante eseguo uno, per un istante eseguo l'altro

	T	indice (i)
P1	1	5
P2	1	5



→ P2 P3 P1 → dopo che P1 è eseguito per un quanto

→ P1 P3 P2

Sistemi desktop interattivi con l'operatore si preoccupa il blocciamento

CPU MEM TIMER → SO ordina un tempo al timer, quando scade il timer lancio un interrupt

IN QUESTO MODO CAMBIAMO IL CONTESTO

Le routine è un pezzo di sistema operativo che passa il controllo allo scheduler

Linux esegue varie code diverse, in base all'urgenza (priorità)

e nei computer di bordo

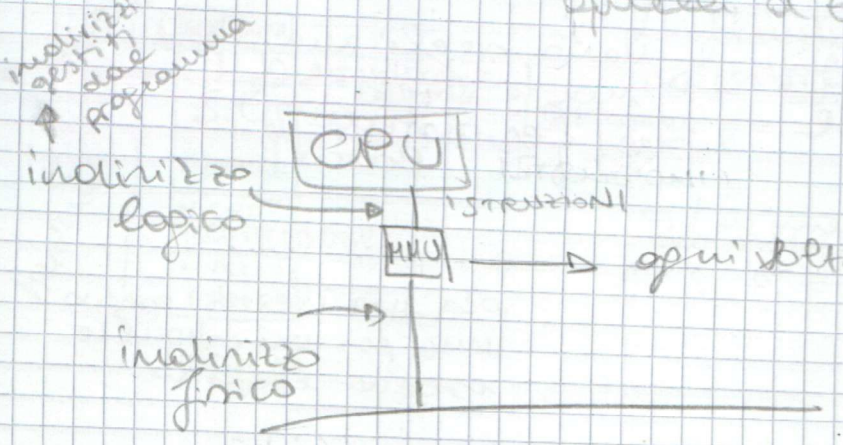
es nel satellite c'è un sistema real time (il programma deve terminare entro il corso di tempo reale, perché il satellite ci sposta) o l'altro non serve

SCHEDULING DEADLINE → esegue chi ha più

metodo che si basa su un componente hardware connesso alla CPU, il memory management unit

permette di nascondere a ciascun processo la memoria degli altri processi

vede una porzione degli indirizzi sul bus, che sono quelli a lui riservati

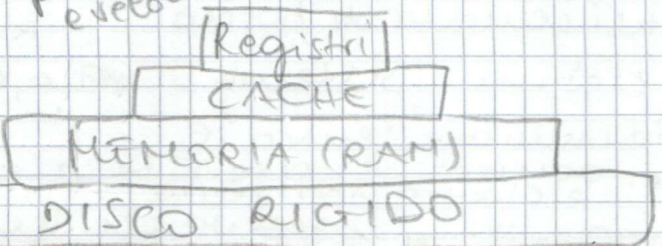


INDIRIZZI ALL'INTERNO DELL'ESEGUIBILE (LOGICI) { 0 ... 100

↕ discostamento gestito dal MMU

INDIRIZZI TRASLATI IN MEMORIA (FISICI) { 1000 ... 1100 → quali aree di memoria sono utilizzate da chi

Con il cambio di contesto la MMU gestisce fornendo ad un altro programma un altro pezzo di indirizzi, per non creare sovrapposizione di dati diversi in memoria.



si trasferisce su disco perché, interrotto, non verrà utilizzato nell'immediato

CPU o il processo non può accedere direttamente alla periferica

le utilizza tramite dei moduli del S.O.

l'unico che non ha limitazioni, vede tutto

{ se un processo richiedesse un indirizzo non suo, il MMU accide il programma }

Appositi comandi del S.O. invocano il driver elaborano la richiesta e la inviano alla periferica

è importante la gestione delle risorse condivise

(2 processi vogliono avere accesso alla stessa periferica ex 2 stampe)

SYSCALL

programmi che richiedono interventi del S.O. (interfaccia del S.O.)

anche in questo caso abbiamo una coda per priorità

→ [R2] [R1] →

DISCO RIGIDO

file e cartelle (file system)

organizza dati e cartelle

ripartizione il contenuto del disco in file e gestisce le cartelle con una gerarchia ad albero

accanto una tabella di ciò contenuto