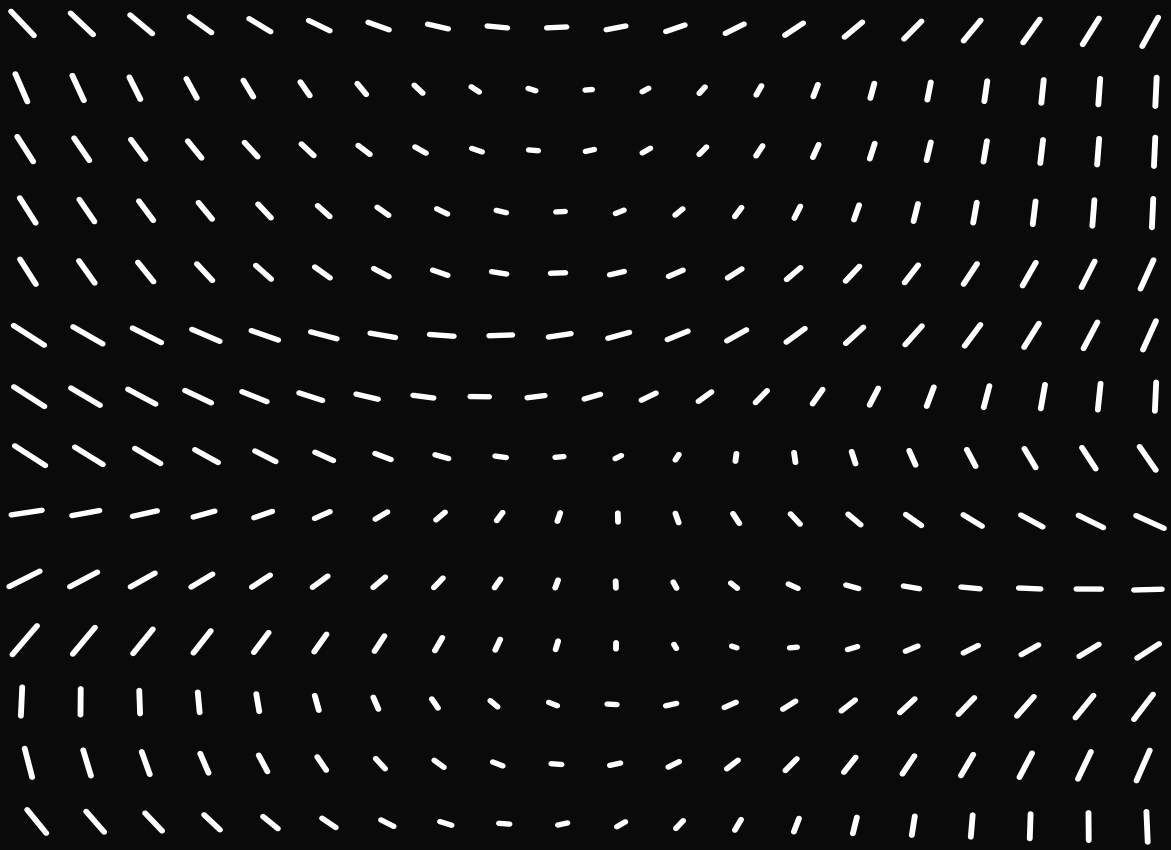
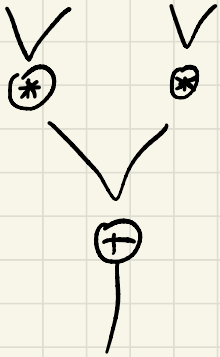
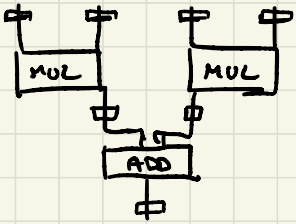


Introduzione

$f = a * b + c * d$

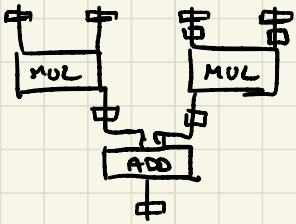




COMBINATORIO

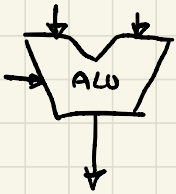
REGISTRI

CON 2 CICLO DI CLOCK



CON 3 CICLI DI CLOCK

ALU



(ATL) RESORCE SHARING CONTROLLATA DALLA CONTROL UNIT (CU)

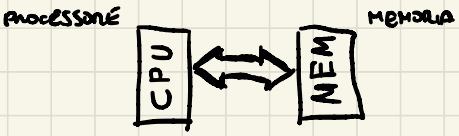
DATO CHE HO UN SOLO COMPONENTE IL COMPUTER DOVRÀ ESSERE RIPENSATO PER FARE UNA COSA LA LA VOLTA

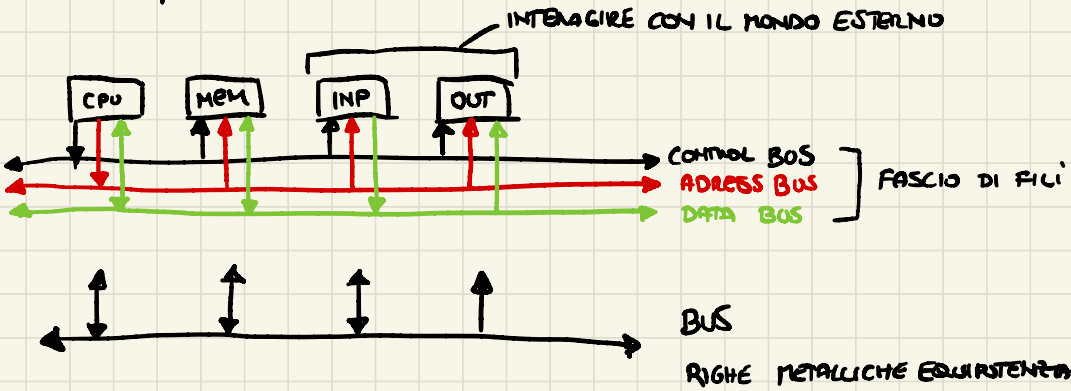
ARCHITETTURA VON-NEUMANN

- LEGGE ED ESEGUE INSTRUZIONI DA UNA MEMORIA
- VENGONO ESEGUITI UNO ALLA VOLTA NELLA CPU

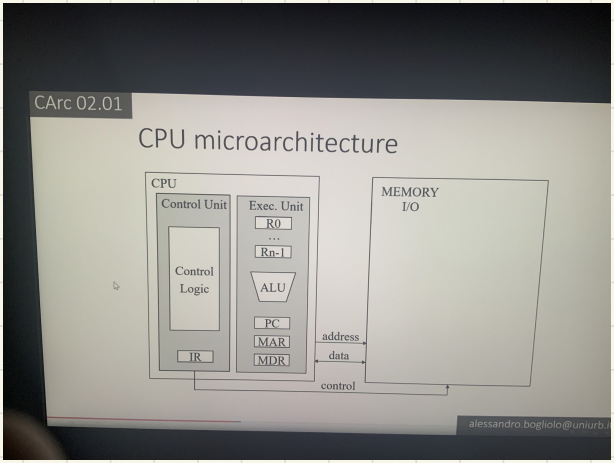
CPU

- LEGGE UN'ISTRUZIONE, ED ESEGUE IN UN CICLO SENZA FINE
  - FETCH
  - EXECUTION





CPU controlla il BUS CONTROL + ADRESS BUS unico componente che scrive



MICRO ARCHITETTURA

(IR) INSTRUCTION REGISTER (ULTIMA ISTRUZIONE LETTA)

(PC) PROGRAM COUNTER (PUNTATORE ALLA PROSSIMA ISTR.)

(MAR) MEMORY ADDRESS REGISTER

(MDR) MEMORY DATA REGISTER

(R_0 - R_{n-1}) GENERICI INDIRIZZI INDICIZZABILI

REGISTER FILE

SCRIVIBILI IN BASSO LIVELLO (ASSEMBLY)

ESECUZIONE DI UNA SOLA ISTRUZIONE (MICRO PROGRAMMA)

1 MAR ← PC
2 MDR ← MEM [MAR]
3 IR ← MDR
4 DECODE
5 PC ← PC+1
6 EXECUTE → SI PUÒ SOVRASCRIVERE IL PC (QUINDI SI POSSONO FARE SALTI)
7 GO BACK 1

= 1 IR ← MEM [PC] (INTERFACCIA DEDICATA ALLA MEMORIA)

(MEMORIA CI FORNISCE A QUESTO INDIRIZZO L'ISTRUZIONE)

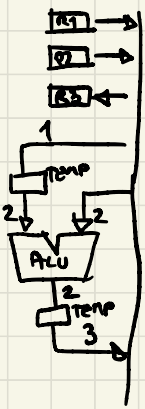
ADD R3, R1, R2 (R3 ← R1+R2) ADDIZIONE TRA REGISTRI INTERNI

① OPERAND FETCH
Bring the operands at the imputs of ALU

② EXECUTE
compute sum

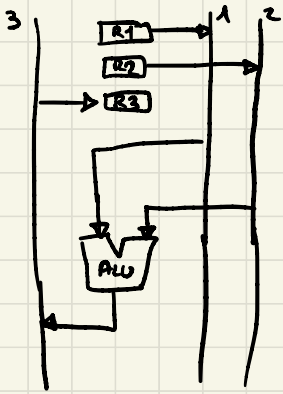
③ WRITE BACK
put the result in the destination register

SINGLE INTERNAL BUS



1. TMPin ← R1
2. TMP OUT ← TMPout+R2
3. R3 ← TMP OUT

3 INTERNAL BUS



1. R3 ← R1+R2

1/03/2023 (AUDIO 2)

ISTRUCTION SET

Caratteristiche generali: (CRITERI DI CLASSIFICAZIONE)

- Dimensione
  - → CISK (ISTRUCTION SET PIÙ COMPLESSO)
  - → RISK (ISTRUCTION SET PIÙ RISTRETTO)
    ISTRUZIONI PIÙ BREVI
    (POCHE COSE FATTE IN MANIERA EFFICIENTE, E VELOCE)

- Formato delle istruzioni

- Modello di esecuzione
  - ↳ RAPPORTO CON LA MEMORIA
    - ↳ MEM-MEM (PROC. LAVORA SULLA MEMORIA)
    - ↳ REG-REG (PRIMA CARICATI IN REGISTRI INTERNI)
    - ↳ MEM-REG (UNO DALLA MEMORIA UNO DAI REGISTRI)
    - ↳ STACK (PILA)

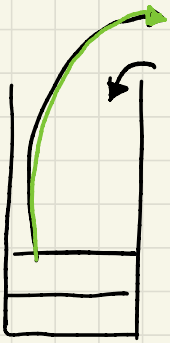
- INDIRIZZAMENTO DELLA MEMORIA

RISC → hanno tutte la stessa dimensione (istruzioni)
pochi formati standardizzati, OP CODE sempre all'inizio
OP CODE un solo formato

CISC → hanno differenti dimensioni (istruzioni)
OP CODE - posizioni variabili, formati variabili

RISC → processori più usati
CISC → ormai in disuso

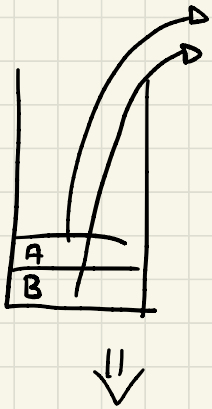
STACK (PILA)



un solo punto di accesso, si aggiunge sempre dal ultimo dato

deve prevedere quello che faremo

una volta prelevati vengono eliminati



A+B

codifica della somma = ADD

MEM - MEM (OP. PIÙ COMPRESSA)

DATI E RISULTATI DEVONO ESSERE SCRITTI IN MEMORIA (senza registri)

MEM - REG

POSSONO ESSERE SCRITTI SIA SU REGISTRI CHE MEMORIA (almeno un registro)

REG - REG (OP. ELEMENTARE)

SOLO SU REGISTRI

ESEMPI ESECUZIONI ARCHITETTURE

$D = A * B + C$

$\downarrow$

$D = A * B$
$D = D + C$

MEM - MEM

```
MUL  D, A, B
ADD  D, D, C
```

MEM - REG

```
LOAD  R1, A
MUL   R3, R1, B
ADD   R3, R3, C
STORE D, R3
```

MEM - REG - 1

```
LOAD  A
MUL   B
ADD   C
STORE D
```

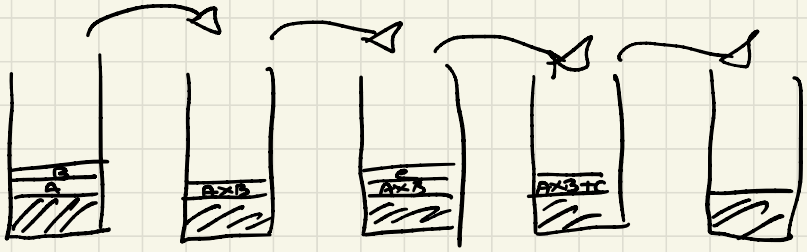
REG - REG

```
LOAD  R1, A
LOAD  R2, B
MUL   R3, R1, R2
LOAD  R1, C
ADD   R3, R3, R1
STORE D, R3
```

```
LOAD  R1, B
LOAD  R2, A
LOAD  R3, C
MUL   R1, R1, R2
ADD   R3, R1, R3
STORE D, R3
```

STACK

PUSH A	PUSH C
PUSH B	PUSH A
MUL	PUSH B
PUSH C	MUL
ADD	ADD
POP D	POP D



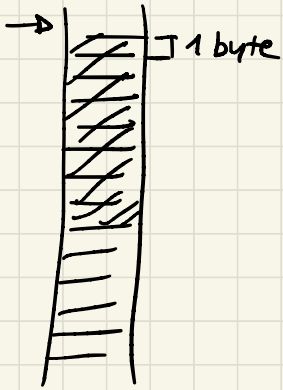
INDIRIZZAMENTO (MISURA BYTE)

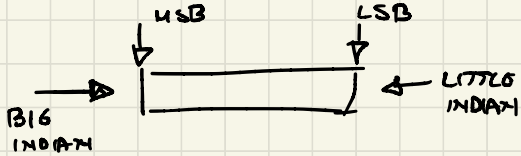
- Problema posizione e peso di bit (bit più signif e meno)

- LITTLE INDIAN (BIT MENO SIGNF.)

- BIG INDIAN (BIT PIÙ SIGNF.) (DEFAULT)

8 byte





LETTI IN UN SOLO CICLO DI BUS
SE A 64 bit = 8 byte

CALCOLARE INDIRIZZO DI MEMORIA

VAR → INDIRIZZO DI MEMORIA

DIRETTO

↳ LD R1, var R1 ← M[var]

INDIRETTO

↳ LD R1, (R2) R1 ← M[R2] (R2 contiene un indirizzo di memoria)

RELATIVO

↳ LD R1, var(R2) R1 ← M(R2 + var) (array)
(spostarsi senza aggiornare R2)

↳ LD R1, var(R2)(RX) → dimensione formato dei dati es: somma floating point ≠ somma interi

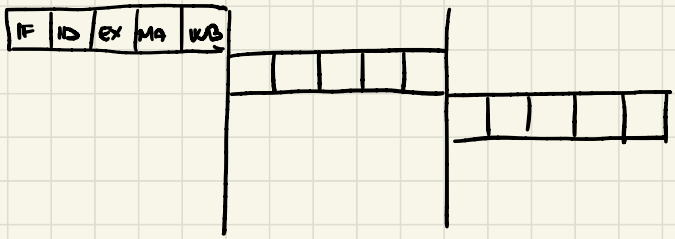
R1 ← M[R2 + var + RX * d]

2/03/2023 (AUDIO3)

1 FETCH ISTRUZIONE	IF	(PC MEM)	— IR
2 DECODIFICA	ID	(CU REG)	— OPERANDI, CONTROL
3 EXECUTE	EX	(ALU)	— RESULT, ADDRESS, CONTROL
4 MEMORY ACSESS	MA	(MEM)	— RESULT, CONTROL
5 WRITE BACK	WB	(REG)	

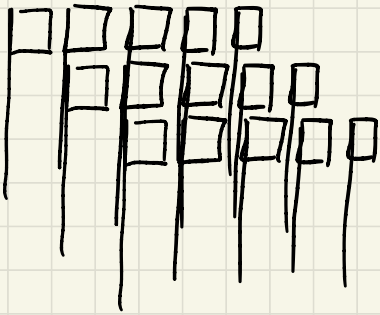
— REGISTRI, PER DIVIDERE LE FASI

Performance



SINGLE CYCLE

MULTI CYCLE



PIPELINING
(CICLO INFINITO)

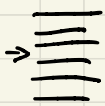
GROSSA NOVITÀ PER L'ARCHITETTURA DEGLI ELABORATORI

METRICHE PERFORMANCE PROCESSORI

CPUT [sec / program]	CPU-time	tempo misurato in secondi riferito all'esecuzione di un programma
CPUC [clock / program]	CPU-clock	frequenza misurata $\frac{1}{T}$ riferito all'esecuzione di un programma

Tclk [sec / clock]

$CPUT = CPUC \cdot Tclk$



CPI = 1 (IDEALMENTE) CPI ≥ 1

AGGIUNGIAMO UN'ALTRA ISTRUZIONE

IC numero di istruzioni da eseguire per completare l'esecuzione del programma

IC · CPI = CPUC (per eseguire l'intero programma)

CPUT = IC · CPI · TclK (tempo per eseguire l'intero programma)

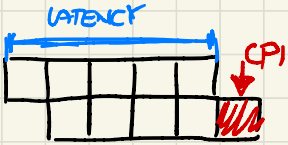
MIPS frequenza di istruzioni ≥ CPUC

↳ dipende dalla natura del programma e dal Istruction set

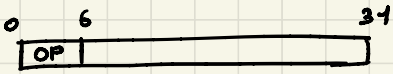
RMIPS = CPUT (ref) / CPUT * MIPS (ref) → riferito a un benchmarks

LATENCY [clocks] FORNIMO OPERANDI → MI DÀ IL RISULTATO = S

THROUGHPUT [ISTRUZIONI/SEC] numero di istruzioni eseguite al secondo

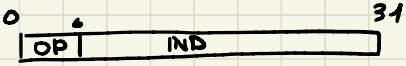


ISTRUCTION SET MINIMO (RISC) (32 bit)



↳ JUMP ISTRUCTION
CONDITIONAL BRANCH
MEMORY ACCESS
ARITMETIC/LOGIC OPERATION

JUMP ISTRUCTION (SALTO INCONDIZIONATO)



JMP Label (etichetta)

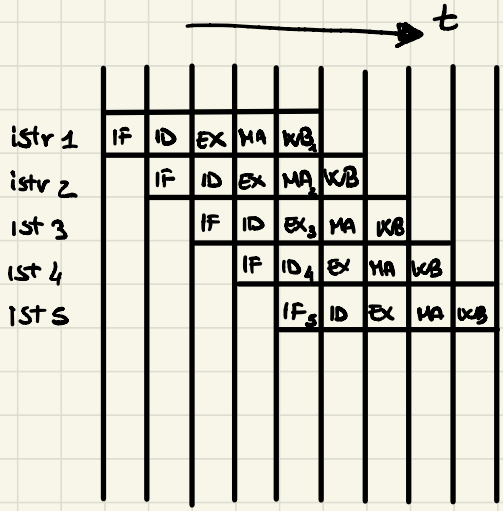
PC ← IND × 4 → A METÀ DI DUE ZERI

32 − 6 = 26 + 2 = 28 bit 2^{28} byte

STRUTTURA DI RIFERIMENTO PER ESEGUIRE DIVERSI TIPI DI ISTRUZIONI

7/03/2023 (AUDIO 4)

CONFLITTI FUNZIONAMENTO PIPELINE



VON-NEUMANN ARCHITECTURE
UN SOLO CICLO DI CLOCK ACCESSO ALLA MEMORIA DA DUE ISTRUZIONI
↓
IMPEGNAMENTO DELLA MEMORIA

SINGOLA MEMORIA PER ACCESSO AI DATI.

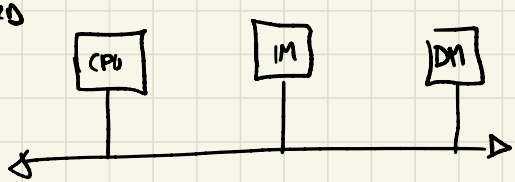
↓

COMPORTA SE CI SONO ACCESSI ALLA MEMORIA NELLO STESSO CICLO DI CLOCK BISOGNA FARE UNO STALLO

SOL. SEPARARE (DM) Memoria dei dati da memoria (IM) delle istruzioni

↓

ARCHITETTURA HARVARD



BISOGNA DUPLICARE ANCHE IL BUS

CONFLITTO DIPENDENZA DEI DATI (CONFLITTO LOGICO)

ADD	r1	r2	r3	IM	reg	ALU	DM	reg			
SUB	r4	r3	r1		IM	reg	ALU	DM	reg		
AND	r6	r1	r7			IM	reg	ALU	DM	reg	
OR	r8	r1	r9				IM	reg	ALU	DM	reg
ADD	r_{10}	r1	r_{11}					IM	reg	ALU	DM reg

ADD	r1	r2	r3	IM	reg	ALU	DM	reg			
SUB	r4	r3	r1		IM	--	--	--	reg	ALU	DM reg
AND	r6	r1	r7						IM	reg	ALU DM reg
OR	r8	r1	r9							IM	reg ALU DM reg
ADD	r_{10}	r1	r_{11}								IM reg ALU DM reg

NON CI POSSONO ESSERE DUE ISTRUZIONI CON IL SALTO

SOLUZIONE DATA FROM WARKING

↓

DOUBLE-RATE REGISTER

CONFLITTI CONTROLLO DI FLUSSO

CBranch - conditional

UC Branch - unconditional

SOL. (1,2)

CON HARDWARE DEDICATO SI RISPARMIA

2 CICLI NEL SALTO CONDIZIONALE

1 CICLO NEL SALTO INCONDIZIONATO

SOL. 3

PREDICT UNTAKEN

SOL. 4 → compromette molto il codice

DELAYED BRANCH

MULTIPLE - EXECUTION

ADDD F4, F0, F2 IF ID EX EX MA WB

IF ID - EX EX MA WB

repition time
2 cicli di clock

SOL PIPELINED EXECUTION UNITS
REPLICATED EXECUTION UNITS

08/03/2023 (AUDIO 5)

PIPELINE IDEALE CPI = 1 , CASO NORMALE C.P.I ≥ 1

CPU specification

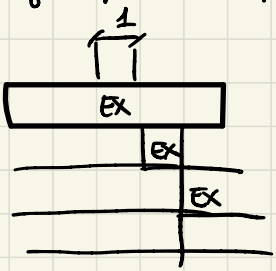
- FUNZIONALI → ISTRUCTION SET
- ELETTRICO → COME IL PROCESSORE SI INTERFACCIA CON IL MONDO
- PRESTAZIONI
  - → DURATA CICLO DI CLOCK (Tclk)
  - → numero di istruzioni per clock (C.P.I) → UNICA ISTRUZIONE (SPLIT) CON CPI = 2
  - → Latenza (numero di cicli che intercorrono tra ingressi e uscite)
  - → RIPETIZIONE MEDIA → dopo quanti cicli di clock è possibile eseguire una istruzione dello stesso tipo

tempo necessario a numero di cicli di clock che servono dagli input all output per le istruzioni successive

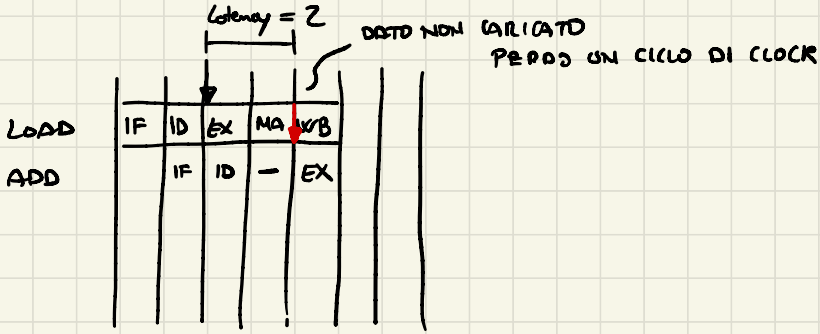
ADDD F4, F0, F2 IF ID EX EX MA WB (latency)

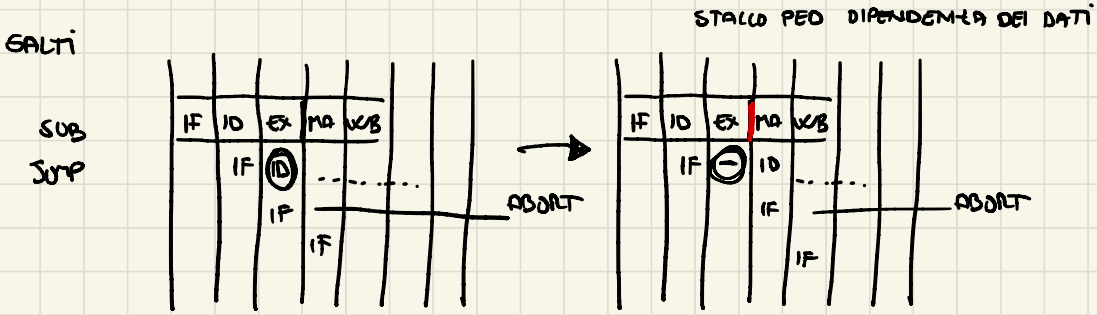
IF ID - EX EX MA WB

repition time
2 cicli di clock



Latenza = conflitti di dati
repition time =





SUB R1, R2, R3

JUMP R1, target

$$CPUC = CPI_{avg}\, IC + N_{STALLS}$$

IC → ISTRUCTION COUNT

N_{STALLS} → STALLI
- LATENCY
- REPITION TIME

$$CPI_{avg} > 1$$

$$CPUC > CPI_{avg}\, IC$$

$$CPI_{avg} = \frac{\sum_{i \in IS} CPI_i\, IC_i}{IC}$$

$$CPUC = \sum_{i \in IS} CPI_i\, IC_i + N_{STALLS}$$

13/03/2023 (AUDIO 6)

OTTIMIZZAZIONE STATICA DEL CODICE (A LIVELLO SOFTWARE)
(PROCESSORI ODIERNI OTTIMIZZAZIONI DINAMICHE)

ISTRUCTION REORDERING SCAMBIARE POSIZIONI SENZA MODIFICARE LA SEMANTICA

SI POSSONO METTERE DELLE ISTRUZIONI INDIPENDENTI IN UNO STALLO

SRODOLARE UN CICLO SERVE PER EVITARE SPRECI, RIUSCIRE A PROCESSARE PIÙ ELEMENTI

REGISTER RENAMING - TOGLIERE DIPENDENZE FALSE DIPENDENZE

PER USARE ISTRUCTION REORDERING

UN LOOP UNROLLING TOTALE COSTANTE SUFFICIENTEMENTE PICCOLO
SI PUÒ SCRIVERE PER TANTE LOTTE

15/03/2023 (AUDIO 7)

OTTIMIZZAZIONE STATICA CODICE ASSEMBLY

- LOOP UNROLLING
  - RIDUCE SALTI
  - ELIMINANO CONTATORI E PUNTATORI
  - SE TOTALE, RENDE CODICE LINEARE (SENZA SALTI)
- ISTRUCTION REORDERING
- REGISTER RENAMING

PROGETTO → SPECIFICA IN C (LINGUAGGIO) → TRADUZIONE IN ASSEMBLY

ISTRUCTION REORDERING → SPOSTARE ISTRUZIONI PER AVERE MENO STALLI POSSIBILI NELLA PAYPLINE (SENZA ALTERARE IL CODICE), IN UN PEZZO DI CODICE LINEARE

LOOP UNROLLING → ALLUNGARE LA SEZIONE DI CODICE LINEARE, PER APPLICARE LE ALTRE OTTIMIZZAZIONI

REGISTER RENAMING → DISACCOPPIARE DUE ISTRUZIONI NEL CICLO PER POI APPLICARE ISTRUCTION REORDERING

LOOP UNROLLING PARZIALE → AUMENTO DI UN CICLO PER PROCESSARE PIÙ ELEMENTI PER VOLTA

22/03/2023 AUDIO (8)

CPI < 1

↳ LIMITE TEORICO, NON SI PUÒ ANDARE SOTTO O UGUALE

↓

UNICO MODO

↳ AVERE PIÙ ISTRUZIONI NELLA STESSA FASE

SUPER SCALAR

PROCESSORI SUPER SCALARI

↳ PIÙ PIPELINE → PROCESSORE DEVE AVERE ABBASTA RISORSE

↳ CODICE SEQUENZIALE

↳ DECIDE IL PROCESSORE CHE DECIDE QUALI ISTRUZIONI MANDARE

↳ ISTRUZIONI IN CONFLITTO → SI MANDA IN CODA AL PROSSIMO CICLO DI CLOCK

↳ PER TUTTI I MOTIVI DETTI NELLE LEZIONI PREC. (BISOGNA EVITARLI)

↳ CPI IDEALE $\frac{1}{N}$

REALTÀ CPI > $\frac{1}{N}$

↳ LOGICA DIFFERENTE DALLE ARCHITETTURE DELLE SINGOLE PIPELING

↳ ISHUING → DECISIONE CHE L'ISTRUZIONE DEVE ANDARE AL CICLO DI CLOCK SUCCESSIVO

(AMA' UN PRE-FETCHING)

↳ PROCESSORI A ISHUING VARIABILE RATE (ISHUING MEDIO INVENTO CPI MEDIO)

↳ PIPELINE DIFFERENTI

↳ PIPELINE SPECIALIZZATE A FARE CERTE ISTRUZIONI ES. INTERA FLOATING POINT (AUMENTO PIPELINE SANO INDIPENDENTI)

↳ PROBLEMA → COMPOSIZIONE DEL CODICE COMPATIBILE CON LA COMPOSIZIONE DEL CODICE ALTRIMENTI UE' UNO SQUILIBRIO, PERDIAMO DI EFFICIENZA

↳ LATENZE DIFFERENTI

↳ ISHUING DINAMICO

SCHEDULING STATICO (NON SI PUÒ CAMBIARE) → PARALLEL ISTRUCTION COUNT

↳ STALLI → INTERO BLOCCO DI PIPELING (CICLO DI CLOCK PERSO)

ES.

$$CPUT = (PIC + SC)\, T_{clk} > IC/N * T_{clk}$$

SUPERSCALAR DLX → PIPLINE < INTERA / FLOATING POINT (LATENZA MAGGIORE)

- ↳ REGISTRI DIVERSI
- ↳ ACCESSO MEMORIA ESITO INTERO RISCHIA DI IMPATTARE UN ESECUZIONE FLOATING POINT → BISOGNA MANDARE UNO STALLO
- ↳ REORDERING
  - ↳ TENENDO CONTO DEL MASSIMIZZARE LA NATURA DELL'ARCHITETTURA

VLIM PROCESSORS → MOLTI PIPELINING PARALLELE (5 A 30)

- ↳ CODICE PARALLELO
- ↳ ISTRUZIONI DI CONTINAIA MIGLIAIA DI BIT
- ↳ STATIC SCHEDULING
  STATIC ISSUING
- ↳ LOGICA DI CONTROLLO PIÙ SEMPLICE → MAGGIOR DIFFICOLTÀ PER LA PROGRAMMAZIONE

ES.

PIPLINE → N S (SPECIALIZZATE)

- ↳ 2 FLOATING POINT
- ↳ 2 MEMORY ACCESS
- ↳ 1 SALTI

$$CPUT = (VLIWC + SC)\,Tclk > IC/N * Tclk$$

PROCESSORS OOO → DINAMICO SCHEDULING
DINAMICO ISSUING

- ↳ PARALLELISMO CON IL TRAFFICO STRADALE
- ↳ ALGORITMO DI TOMASULO → IN HARDWARE DI DECISIONE

29.03.2023 (AUDIO)

OTTIMIZZAZIONI DINAMICHE
- OOO
- SPECULATION
- SUB-CLOCK

DATA DEPENDENCIES (CONFLITTI DI DATI)
- RAW READ AFTER WRITE
- WAR WRITE AFTER READ
- WAW WRITE AFTER WRITE

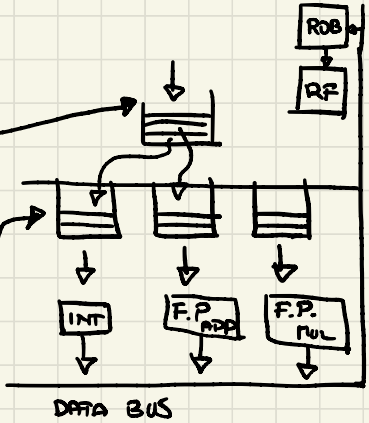
↓ DA PRIORITÀ ALLE ISTRUZIONI DI ESECUZIONE

↓ LIBERARSI DALLE FALSE DIPENDENZE

↓ FORNIRE I DATI SUBITO ALLA EXECUTION UNIT

↓ PRESERVARE LA COERENZA SEQUENZIALE (SOLO WRITE BACK)

OOO EXECUTION SUPPORT
- ISTRUCTION DECODE E REGISTER FETCH SEPARATI
- CODA ISTRUZIONI (SEMPRE PIENA)
- RESERVATION STATION
- COMMON DATA BUS
- REORDER BUFFER (ROB) (GESTITO IN AUTOMATICO)



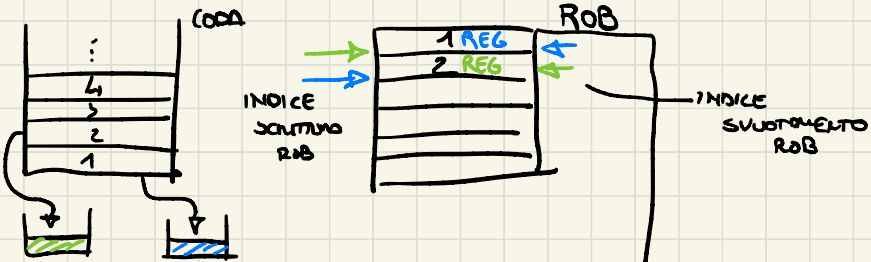
ALGORITMO TOMMASULO (REGISTER RENAMING AUTOMATICO)

ISSUE → LASCIA LA CODA E VIENE MESSA NELLE CODE DELLE RESERVATION STATION

EXECUTE → QUANDO ESTRA NELLE DIVERSE INT, $F.P_{ADD}$, $F.P_{MUL}$

WRITE RESULT → SCRITTURA NEL ROB (ANNOTARLI)

COMMIT → DATI DEL ROB VENGONO TRASCRITTI NEL RF



REGISTER RENAMING

LIMITE DIPENDE DALLA GRANDEZZA DEL ROB

SPECULAZIONE SALTI INCONDIZIONATI, CONDIZIONATI

SE JUMP SI SBAGLIA SVUOTO IL ROB
ALTRIMENTI PROSEGUO

- BRANCH PREDICTION BUFFER
- BRANCH PREDICTION TARGET BUFFER

5.04.2023 (AUDIO 10)

MEMORIE → ARCHITETTURA VON NEUMANN (COLLO DI BOTTIGLIA)
- CLASSIFICAZIONE
  - → VOLATILI (NON ↯)
  - → NON VOLATILI (PERSISTENZA DEI DATI A FRONTE DELL'ALIMENTAZIONE)

BILANCIAMENTO LETTURA E SCRITTURA
- → ROM → MEMORIA A SOLA LETTURA (NON VENGONO SOVRASCRITTI)
- → PROM → MEMORIA PROGRAMMABILE
- → EPROM → MEMORIA CANCELLABILE
- → EEPROM → MEMORIA CANCELLABILE ELETTRONICAMENTE (CANCELLABILI ANCHE DALL'SISTEMA)
- → FLASH → BASATI SU EEPROM NON-VOLATILE (DETENGONO IL DISPOSITIVO) ABBASTANZA VELOCE (A PREVALENTE LETTURA)
- → RAM → VOLATILI (HANNO LE STESSE PRESTAZIONI OVUNQUE LE LEGGIAMO)

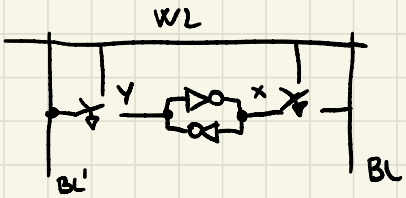
RAM STATICA (SRAM)

CIRCUITO BISTABILE

→ "1" o "0"

A LIVELLO TEORICO PUÒ CONTENERE UN BIT DI DATO AL SUO INTERNO, A LIVELLO PRATICO SI STABILIZZA SUBITO → IN UNO STATO O L'ALTRO

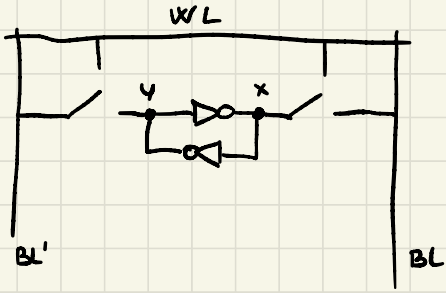
COME COLLEGARLO AL RESTO DEL SISTEMA?

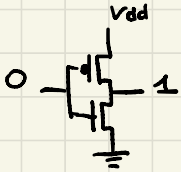


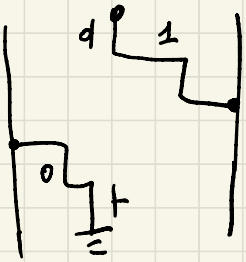
WL → BIT DI PAROLA

BL → BIT LINE

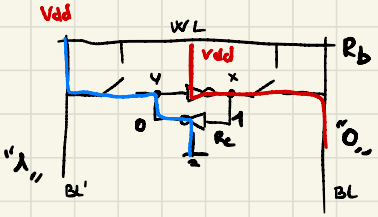
OPERAZIONE DI LETTURA

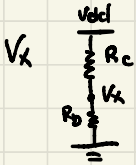






OPERAZIONE DI SCRITTURA





$$I = \frac{V_{dd}}{R_c + R_b}$$

$$V_x = R_b \cdot I$$

$$V_x = V_{dd} \frac{R_b}{R_b + R_c}$$

$$R_b \ll R_c$$

SONO PROCESSI PILOTATI (RAM STATICHE)

RAM DINAMICHE → USA CONDENSATORI $\neq$ $Q = C \cdot V$

↳ OPERAZIONE DI REFRESH PER CONDENSARE PERDITE DEI CONDENSATORI NON IDEALI

LETTURA E RISCRITTURE (SENZA L'AVVISO DEL PROCESSORE)

SCRITTURA PIÙ FACILE CHE LETTURA

$$\begin{cases} C \cdot Q_c = C_b \cdot Q_b \\ Q_c + Q_b = C \cdot V_{dd} \end{cases}$$

→ PRECARICA CON $\frac{V_{dd}}{2}$ per alzare / abbassare il suo valore

$$V_x = \frac{Q}{C_b + C} = V_{dd} \frac{C}{C + C_b}$$

$$V_{dd} = \frac{Q}{C}$$

serve amplificare (nota sbilanciamenti e li riporta al valore nominale leggibile)

lettura ricarica come refresh

refresh lettura

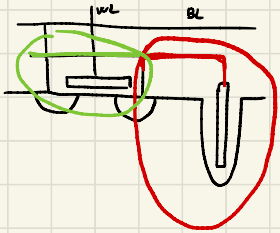
SRAM VS DRAM

06.04.2023 (AUDIO 11)

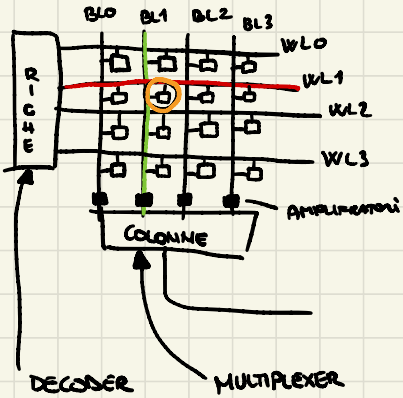
RAM STATICHE → MOLTO VELOCI → PIÙ ELEMENTI DI COSTRUZIONE → INVERTER
↳ MENO CAPACITIVE DELLE RAM DINAMICHE — 6 TRANSISTOR

RAM DINAMICHE → PIÙ LENTE DI QUELLE STATICHE
↳ POCHI ELEMENTI DI COSTRUZIONE — 1 TRANSISTOR E CONDENSATORE
↳ PIÙ CAPACITIVE

(STACKED E TRENCH CAPACITOR)



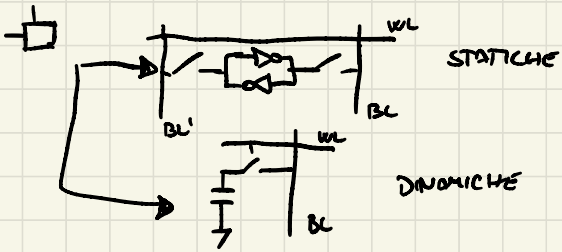
SCHEMA ARCHITETTURALE MEMORIE



DECODER

MULTIPLEXER (SELEZIONE DI UNA SOLA CELLA)

SCHEMA A MATRICE

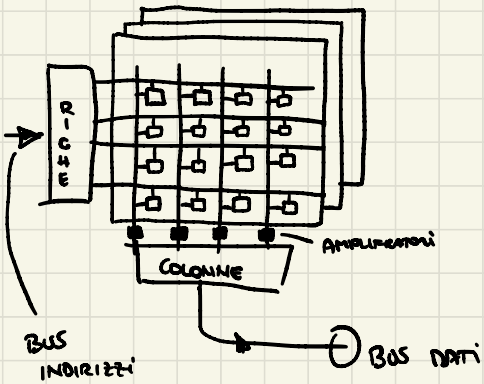


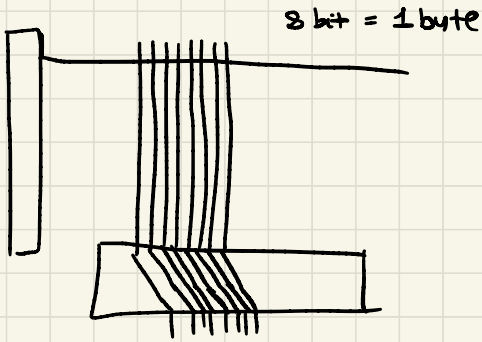
MSB → DECODER — 01
LSB → MULTIPLEXER — 01

0101 — CELLA DA CUI IL PROCESSORE VUOLE LEGGERE O SCRIVERE IL DATO

→ SEGNALE MANDATO DAL PROCESSORE

USIAMO IL PARALLELISMO PER PROCESSARE PIÙ DATI ALLA VOLTA





ORDINE DELLE FASI LETTURA, SCRITTURA

↳ CODIFICARE LA RIGA → PILOTARE LE BIT LINE (ASSUMONO IL VALORE NELLE CELLE CORRISPONDENTI)

↓

DECODIFICA INDIRIZZO COLONNA

↓

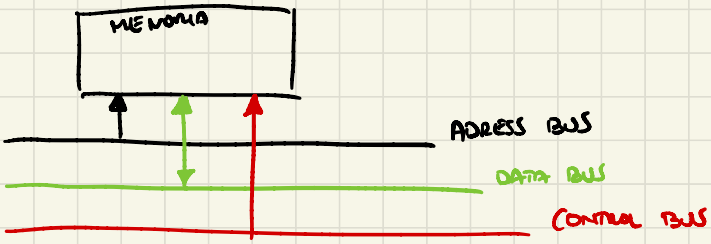
SELEZIONE COLONNA

↓

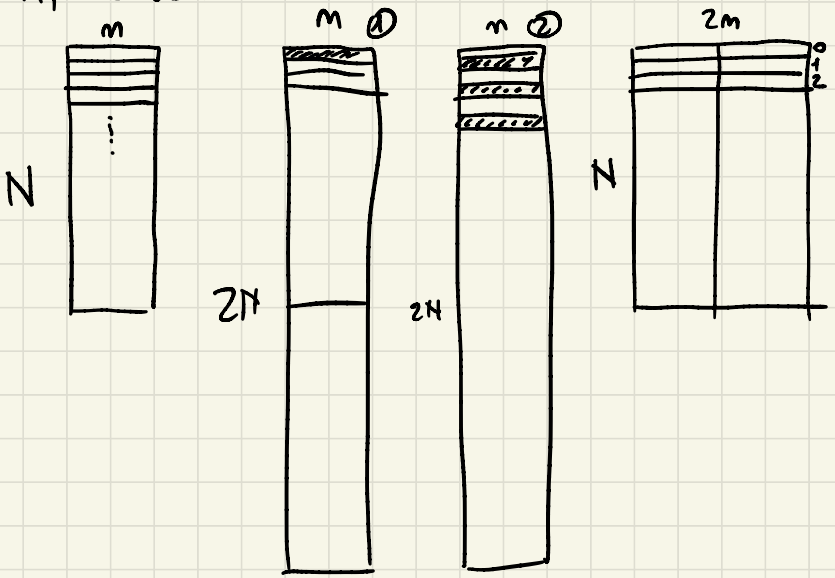
PILOTAGGIO BUS CORRISPONDENTE

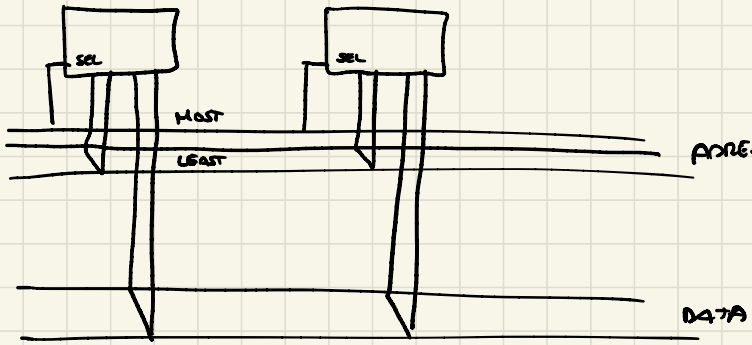
PAGE MODE NON UTILIZZA IL RANDOM ACCESS → INFORMAZIONI CONTIGUE

IL REFRESH → SCANNERIZZA TUTTE LE RIGHE PERIODICAMENTE SENZA CHE LA MEMORIA VENGA IN CONFLITTO CON L'ACCESSO ESTERNO
(UTILE POICHÉ SI FA UN REFRESH SU MILIONI DI CELLE)



N parole da m bit





BIT PIÙ SIGNIFICATIVO SERVE PER INDIRIZZARE QUALE DISPOSITIVO USARE ①

② BIT MENO SIGNIFICATIVO CONSENTE DI PER USARE IN PIPELINING LE MEMORIE

MEMORIE NON VOLATILI

ROM → NAND
→ NOR

19/04/2023

DA ROM A PROM ERASABLE → SI POSSONO CANCELLARE E RISCRIVERE I DATI

20/04/2023 (AUDIO 12)

CACHE → PICCOLE E MOLTO VELOCI

BLOCCHI DALLA RAM → ALLA CACHE

↓

SPOSTANO SEMPRE LE STESSE QUANTITÀ DI DATI

MAPPATURA DIRETTA (ZONA STESSE DIMENSIONE DELLA CACHE)

BLOCK ADDRESS		
TAG	INDEX	OFFSET

CACHE

	00	01	10	11
00				A
01				
10			B	
11				

TAG
01
11

RAM

	00	01	10	11
00 00				
00 01				
00 10				
00 11				

01 00				A
01 01				
01 10				
01 11				

10 00				
10 01				
10 10				
10 11				

11 00				
11 01				
11 10			B	
11 11				

CACHE ASSOCIATIVA

BLOCK ADDRESS	
TAG	OFFSET

	00	01	10	11
01 00				A

SI CONFRONTANO I DATI DI UNA RIGA TUTTI IN UNA VOLTA COME UN GROSSO MULTIPLEXER

RAM

		00	01	10	11
00	00				
00	01				
00	10				
00	11				

01	00				A
01	01				
01	10				
01	11				

10	00				
10	01				
10	10				
10	11				

11	00				
11	01				
11	10			B	
11	11				

POLITICHE RIMPIAZZO BLOCCHI

METTIAMO UN BLOCCO TIME FLAG ✓ PER SAPERE QUALE DATO È STATO USATO PER L'ULTIMO (PRINCIPIO DI COLOCALITÀ)

- BLOCCO USATO MENO DI RECENTE (LIST RECENTLY USE) LRU
- FIRS IN FIRST OUT → LISTE CIRCOLARI FIFO
- RANDOM RND

SET ASSOCIATIVA

BLOCK ADDRESS		
TAG	INDEX	OFFSET

CACHE

	00	01	10	11	TAG
0				A	010
1					

	00	01	10	11	TAG
0			B		111
1					

RAM

	00	01	10	11
00 00				
00 01				
00 10				
00 11				
01 00				A
01 01				
01 10				
01 11				
10 00				
10 01				
10 10				
10 11				
11 00				
11 01				
11 10			B	
11 11				

CACHE PERFORMANCE

SISTEMI DI LIVELLI GERARGICI LIVELLO 1 CACHE – LIVELLO 2 CACHE

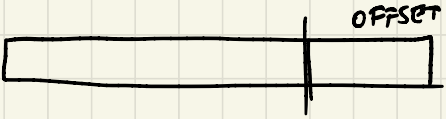
MEMORIA VIRTUALE

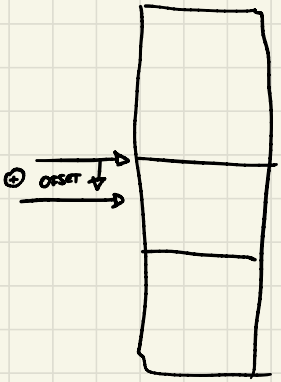
26/04/23 (AUDIO 13)

MEMORIA VIRTUALE → MAPPATO DINAMICAMENTE

↳ SPAZIO DI INDIRIZZAMENTO LOGICAMENTE SUPERIORE A QUELLO FISICO
QUESTO PERCHÉ NON VERRÀ MAI UTILIZZATO TUTTO CONTEMPORANEAMENTE

PAGING VS SEGMENTI

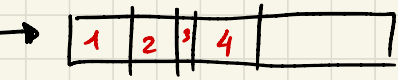




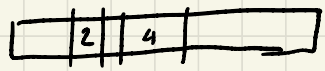
SEGMENTI OPERAZIONE DI SOMMA ARITMETICA

SEGMENTI DIMENSIONE VARIABILI

PAGINE → STESSE DIMENSIONE



1, 3 VENGONO DEALLOCATI



DEFRAMMENTAZIONE

NON RESE AD ALLOCARE

TROVARE UNA QUADRA TRA DIMENSIONE PAGINE IN MODO DA REALIZZARE IL COMPROMESSO TRA PRESTAZIONI E TRASFERIMENTI

FRAMMENTAZIONI PAGINE SUPERIORE
FRAMMENTAZIONI A SEGMENTI

COSTA MOLTO NON TROVARE UNA PAGINA

↓

BISOGNA TROVARLA SEMPRE

↓

IN CASO NO ABBIAMO DELLE POLITICHE DI RIMPIAZZAMENTO

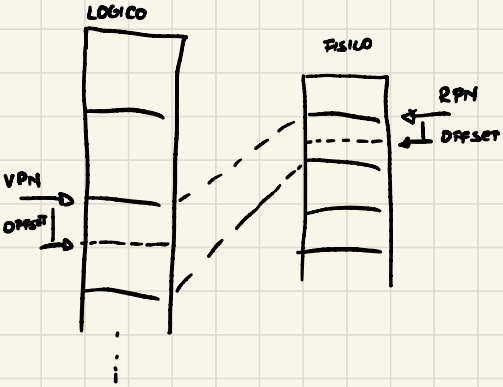
↓

ASSOCIATIVA CACHING → LISTA RECENTLY USE → POLITICHE DI RIMPIAZZAMENTO

RAPPORTO TRA INDIRIZZI LOGICI E FISICI (MEMORIA PAGINATA)

VIRTUAL PAGE NUMBER = VPN

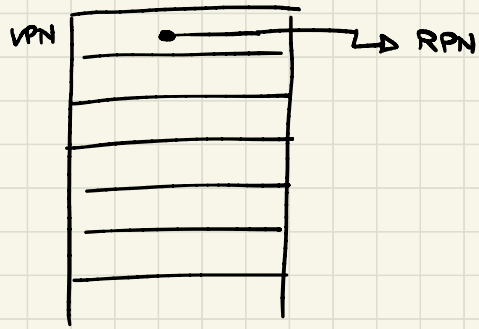
REAL PAGE NUMBER = RPN



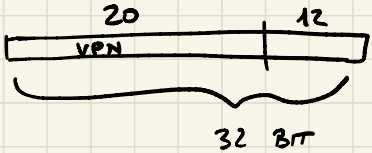
TRADUZIONE INDIRIZZI NON NECESSITA DELL' OFFSET

TRADUZIONE → TRAMITE TABELLA DI CONVERSIONE

TABELLA DELLE PAGINE

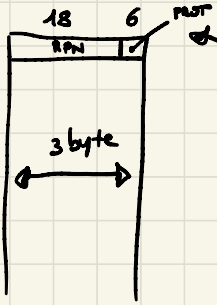


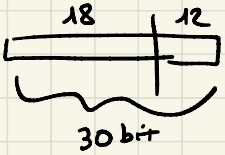
32 BIT



4Kbyte

2^{20} 1M DI RIGHE

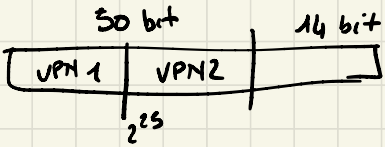




bit protezione = 6

PROT

64 16 KB



2^{25}

DUE TABELLE

$\downarrow$

2^{25}

TLB TRANSLATION LOOKSIDE BUFFER

CACHE FISICA – VIRTUALE INDIRIZZO CHE USIAMO PER INDIRIZZARLA

27/04/2023 (AUDIO 14) LINUX → LS CPU → INFORMAZIONI CPU

PERFORMANCE OPTIMIZATION

MICRO-BENCHMARKS → PROGRAMMA → MISURARE PERFORMANCE DEL PROCESSORE NEL ESEGUIRE QUEL CODICE

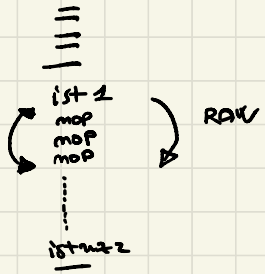
- SINTETICI → SOLO SCOPO DI MISURARE PERFORMANCE STANDARD DEL PROCESSORE
- NATURALI → PROGRAMMA NON CREATO PER BENCHMARKS MA USATO COME TALE

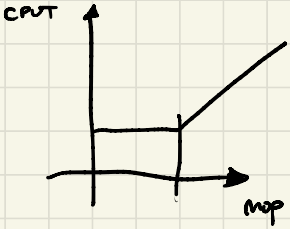
CARATTERIZZAZIONE SPERIMENTALE SAPP. TUTTO SUL PROCESSORE → OBBIETTIVO → CPI DI UN PROCESSORE

↓

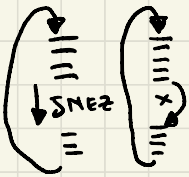
MISURARE LATENZA

	∅		
CPI	1	istruz 1	
CPI + LATENZA + CPI	2	int+ent 1	int+ent 2





CARATTERIZZARE I SACTI

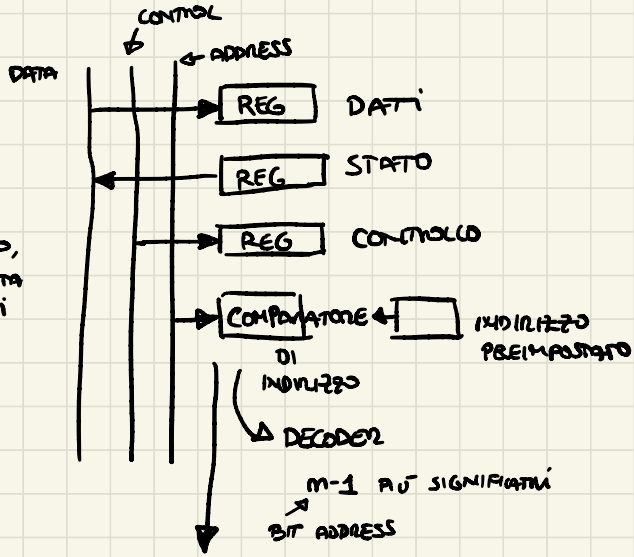


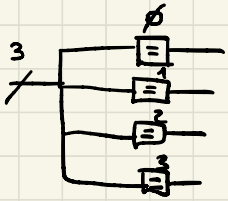
03/05/2023 (AUDIO 14)

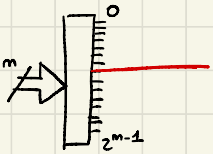
PERIFERICHE → GESTITE ATTRAVERSO IL BUS
↳ ALTRO BUS RISPETTO A QUELLO DEL PROCESSORE E MEMORIE
↓
SEMPRE CONTROLLABILE E RAGGIUNGIBILE DAL PROCESSORE

INTERFACCIA VERSO IL BUS ATTRAVERSO I REGISTRI
↳ LETTI E SCRITTI DAL PROCESSORE

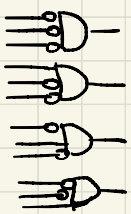
REGISTRO DI STATO
↳ SINCRONIZZAZIONE
SONO DISPOSITIVI ASINCRONI
NON PREVEDIBILE DAL PROCESSORE QUANDO USARLA
SERVE PADRONE A QUESTO IL REGISTRO, A MOSTRARE IL SUO STATO CHE È PRONTA A RICEVERE E MANDARE INFORMAZIONI







=

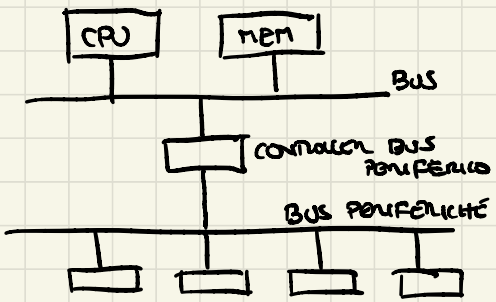


SPAZIO DI INDIRIZZAMENTO → MEMORY-MAPPED DEVICE
→ I/O - MAPPED DEVICE

NON DEDICATO
DEDICATO

→ DIPENDE DALL' ISTRUCTION SET SE È COMPRESA O NO

→ NON VIENE USATO IL CACHING



SINCRONIZZAZIONE → SI PUÒ ANCHE PROGETTARE LA GESTIONE A LIVELLO SOFTWARE NON HARDWARE

→ TEMPO ACCESSO → BUS PERIFERICO → TEMPO DI ACCESSO

TEMPO SERVIZIO → NON AVENDO UN CLOCK SINCRONIZZATO NON RIESCE IL PROCESS. AD ASPETTARE LA RISPOSTA (TRAMITE REGISTRO DI STATO SI FA SI CHE L'UNICO CANALE SIA IL REGISTRO DI STATO)

→ TEMPO DI SERVIZIO

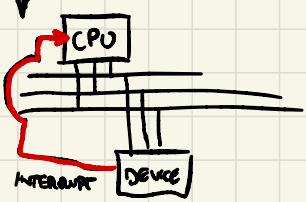
while (peripheral.status != READY) {} POLLING → TRAMITE POLLING IL PROCESSORE INTERROGA AD INTERVALLI IL REGISTRO DI STATO

→ (TRAMITE LETTURE E SCRITTURE, DEVE ESSERE LETTO DI CONTINUO)

PARADIGMI DI SINCRONIZZAZIONE → PROGRAMMA → SENZA SUPPORTO HARDWARE SOLO CON POLLING SVANT. COSTO TIENE IMP. CPU E BUS PER NON FARE NULLA

→ SUPPORTO HARDWARE

→ INTERRUPT → PER DAR ASCOLTO ALLA PERIFERICA

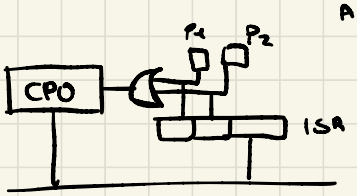


→ DMA → DIRECT MEMORY ACCESS

→ ASSUMONO MOMENTANEAMENTE IL CONTROLLO DEL BUS PER PRENDERE I DATI SENZA CHE SIA IL PROCESS. A GESTIRGLIELI

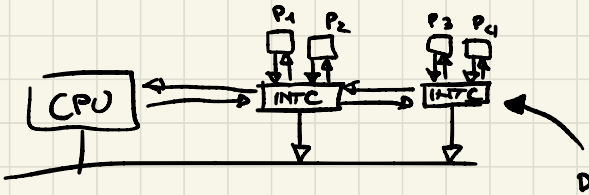
a)

b)



INTERRUPT STATUS REGISTER

c)



INTC DA SIA PRIORITÀ CHE SI PRECCUPA DI QUALE PERIFERICA SERVIRE AL PROCESSORE

DAISY CHAIN