

RIASSUNTO

- 1. SISTEMI
 - GENERAL PURPOSE
 - SPECIAL PURPOSE (ESPERI)
 - $I \rightarrow SOC \rightarrow D$, SOC: SYSTEM ON A CHIP \rightsquigarrow SUPERSCARICATI: PIÙ ISTRUZIONI. 4 CLK
 - PROCESSORI
 - CISC: UN COLOPO DI CLOCK PER UN'ISTRUZIONE, >100 ISTRUZ.
 - RISC: 1 COLOPO DI CLOCK, 1 ISTRUZIONE, 2-100 ISTRUZ. (es. 32), PIPELINE, SISTEMA MM. REGISTRI
 - ASIC: APPLICATION SPECIFIC INTEGRATED CIRCUIT \rightsquigarrow SINTESI HW

2. PROGETTAZIONE DI CIRCUITI:

DIAGRAMMA A BLOCCHI \wedge DI TRASFORMAZIONE

INTERAZIONE: OGNI COMPONENTE DEVE CONOSCERE DI ESSERE IL PIÙ INFLU. POSSIBILE \rightarrow PARTE MODIFICA
- IN FORMA LASSI NON SI CONOSCE LA STRUTTURA INTERNA DI UN CIRCUITO, MA SOLO I/O

LIVELLI DI PROGETTO: FAETRICO \rightarrow TRANSISTORI \rightarrow PORTA LOGICA \rightarrow RISULTATI \rightarrow SISTEMA

- SISTEMI
 - ① COMBINATORI: I VAL. D'USCITA DIPENDONO ESCLUSIVAMENTE DA I IN t₀
 - ② SEQUENZIALI: I VAL. D'USCITA DIPENDONO DA INPUT a t₀ + VALORE DI t < t₀

① DESCRIZIONE TRAMITE di BOOLEANA o TABLEAU DI USCITA

② USO DIAGRAMMA DEGLI STATI, MA POSSO STAT. LO STATO DEL SIST. CALCOLATO SOLO I/O

SISTEMI
SEQUENZIALI

\hookrightarrow di transizioni: $\forall f: X \times Y \rightarrow Y \times Z$

• FSM (FAIR STATE MACHINE) / MODE: 0 APRENTE SOLO QUELLE VOCI CHE CORRISPONDENTI VOCI DI STATO

• PROGETTO A LIVELLO FAETRICO

• Transistor:

B: BASE/GATE
C: COLLETTORE
E: FUORITONDO

• MOSFET: $I = 0$

• INVERTER NMOS: $\begin{array}{c|c} I & 0 \\ \hline 0 & 1 \\ 1 & 0 \end{array}$

if $V_{IN} < V_{SOGNA}$: $V_{OUT} = V_{CC}$
else ($V_{IN} > V_{SOGNA}$): $V_{OUT} = V_{DD}$ \rightsquigarrow GROUND

• INVERTER CMOS: INTEGRA ENTRATE

• PORTE LOGICHE

• AND:

$$z = x_1 x_2 = x_1 \wedge x_2$$

• OR:

$$z = x_1 + x_2 = x_1 \vee x_2$$

• NAND = NOT AND D_o

• NOR:

OR EXCLUSIVE

• EXOR:

$$z = x_1 \oplus x_2$$

0	0	0
0	1	1
1	0	1
1	1	0

• NOT: $x \rightarrow D_o = \bar{x}$

• EXNOR, EXOR) D_o

VON-NEUMANN: MEMORIA UNICA PER DATI E ISTRUZIONI
HARVARD: MEMORIA SEPARATA PER DATI E ISTRUZIONI

\rightsquigarrow SUPERSCARICATI: PIÙ ISTRUZIONI. 4 CLK

CISC: UN COLOPO DI CLOCK PER UN'ISTRUZIONE, >100 ISTRUZ.

RISC: 1 COLOPO DI CLOCK, 1 ISTRUZIONE, 2-100 ISTRUZ. (es. 32), PIPELINE, SISTEMA MM. REGISTRI

ASIC: APPLICATION SPECIFIC INTEGRATED CIRCUIT \rightsquigarrow SINTESI HW

PROGETTAZIONE DI CIRCUITI:

DIAGRAMMA A BLOCCHI \wedge DI TRASFORMAZIONE

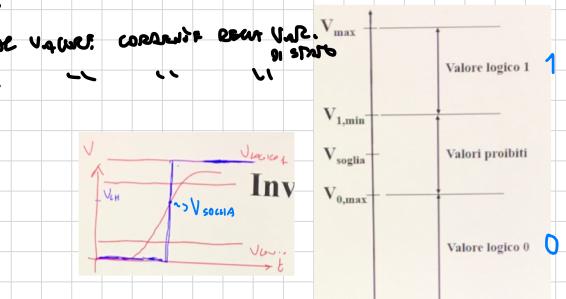
INTERAZIONE: OGNI COMPONENTE DEVE CONOSCERE DI ESSERE IL PIÙ INFLU. POSSIBILE \rightarrow PARTE MODIFICA
- IN FORMA LASSI NON SI CONOSCE LA STRUTTURA INTERNA DI UN CIRCUITO, MA SOLO I/O

LIVELLI DI PROGETTO: FAETRICO \rightarrow TRANSISTORI \rightarrow PORTA LOGICA \rightarrow RISULTATI \rightarrow SISTEMA

- SISTEMI
 - ① COMBINATORI: I VAL. D'USCITA DIPENDONO ESCLUSIVAMENTE DA I IN t₀
 - ② SEQUENZIALI: I VAL. D'USCITA DIPENDONO DA INPUT a t₀ + VALORE DI t < t₀

① DESCRIZIONE TRAMITE di BOOLEANA o TABLEAU DI USCITA

② USO DIAGRAMMA DEGLI STATI, MA POSSO STAT. LO STATO DEL SIST. CALCOLATO SOLO I/O



• BUFFER PMOS: $\begin{array}{c|c} I & 0 \\ \hline 0 & 0 \\ 1 & 1 \end{array}$

if $V_{IN} > V_{SOGNA}$: $V_{OUT} = V_{CC}$

else ($V_{IN} < V_{SOGNA}$): $V_{OUT} = V_{DD}$

0	0	0
0	1	1
1	0	1
1	1	0

• NOT: $x \rightarrow D_o = \bar{x}$

- INSISTE COMPLETO: SOTTO (RIPROV) DI PORTA / LOGICA, IN BASE A) GENERARE TABELLE DI COMBINATORIALE
- CIRCUITI COMBINATORI (1):
 - di combinatoria: $Z: \mathbb{B}^n \rightarrow \mathbb{B}$ / $\mathbb{B} = \{0,1\}$
 - CCDF: circuiti "SEN FORTI": 5 prototipi (innanzitutto: NO CICLI)

- FANIN: N° segnali in ingresso di una porta
- FANOUT: N° uscite verso altri gate

MINIMIZZAZIONE:

SEGUONO i diversi modi per semplificare il circuito \rightarrow LO RISOLVE PIÙ SEMPLICE ED EFFICIENTE

CIRCUITO A 2 VIESSI, ESERCIZIO DI RICONOSCIMENTO A:

$$\begin{aligned} \sum x_1 x_2 \dots x_m &= ab + b'c + \dots + \rightarrow \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \geq \text{OR} \\ \prod x_1 + x_2 + \dots + x_n &= (a+b)(c+d) \dots () \rightarrow \begin{array}{c} \text{OR} \\ \text{AND} \end{array} \leq \text{AND} \end{aligned}$$

- CIRCUITO COMBINATORIO MINIMO \Rightarrow (MIN N° DI PORTE) \vee (metodo di KARNAUGH)
- METODO DI KARNAUGH .

CREA UNA MAPPA CON LE USCITE \rightarrow SEMPLIFICA AUTOMATICO UNI 1 + GRUPPI DI 2, 4, 8, 16, ...
 \rightarrow GRUPPI DI 2, 4, 8, 16, ...

VARIANTI "DON'T CARE": "X" o "-" , où un si può scegliere se è 0 o 1

- RITARDO: l'uscita da una porta richiede UN TEMPO K PER FAR SI CHE IL SEGNALE SI SPREAD(2x)
 \rightarrow È TUO NON APPLICARE NUOVI INPUT FINO A CHE IL TEMPO NUOVO FINISCE

\rightarrow È POSSIBILE CONOSCERE IL RITARDO ASSOCIARE AL CIRCUITO \rightarrow RITARDO DI N° 0, PENSARE = PROFONDITÀ DEL CIRCUITO

CAMMINO (ROUTE): CAMMINO MINIMO CON IL RITARDO È MAX

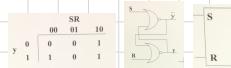
LIVELLO: DEFINISCE LA MISURA DI UN CIRCUITO

CIRCUITI SEQUENZIALI (2):

di dipendenze dal TEMPO: $f(t + \Delta) \rightarrow$ Mentre COME SEI DIPENDI (INTESA)

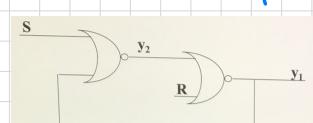
FLIP-FLOP: realizzata UNA SINTESI PER UN PROBLEMA DI TEMPO, SISTEMA DI SET-RESET SR

/ Asincrono: INDEPENDENTI DAL TEMPO



\ sincrono: DIPENDENTI DAL TEMPO \rightarrow USA IL CLK

$S=R=0$: configurazione del valore in FF
 $S=0, R=1$: $X, Y = 0$
 $S=1, R=0$: $X, Y = 1$
 $S=1, R=1$: config. VERSATA
 $\hookrightarrow Y_n = 0/1$ in base
 al ritardo REALE 2 porte



FLIP-FLOP D: realizza il valore di UN SINGOLO BIT DURANTE CLOCK = 1
 \hookrightarrow NON SONO PREMESSI AI STADI $R=S=0, R=S=1 \rightarrow R=S$

CIRCUITI SINCRONI:

\rightarrow IL CLOCK DA IL SEGNALE

\rightarrow MODELLA DI MUFTIGATE: $\Delta t_{clock} >$ RETRASO MAX NELLA RETE

COSTRUZIONE TRAMER STAGE

MINIMIZZAZIONE

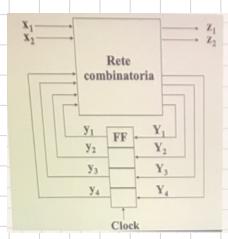
ASSERIRE UN STAGE \rightarrow COMBINATORI

COSTR. TRAMER DI STAGE

SINTESI

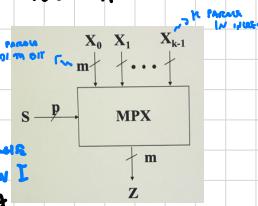
$$\rightarrow T > \Delta + S$$

$T_{clock} > S_{max}$ elementi
 \hookrightarrow ritardo FLIP-FLOP



Componenti

- Componenti combinatori
 - Porte logiche operanti su parole
 - Multiplexer
 - Decodificatori e codificatori
 - Moduli aritmetici (ALU, sommatori, ecc.)
- Componenti sequenziali
 - Registri
 - Contatori
 - FPGA
 - Bus
 - Memorie.

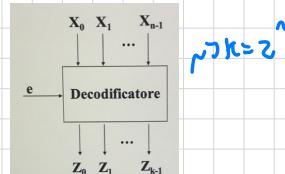


• 3. • PROGETTAZIONE A LIVELLO DI REGISTRI:

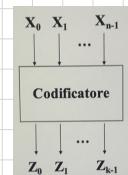
APPROF:
 $p(K=1)$
 $\sum K_i = i > K$
 $K = 2^P$
 $p = \log_2 K$
 \hookrightarrow BIT DI S

- MANIPOLAZIONE DI PAROLE, PROGETTO RESISTITO IN BLOCCHI
- MULTI PLEXER: \hookrightarrow IMPRESA ZONE DI OUTGOING $Z = f(i_0, i_1, \dots)$ works
 USO PER CONNETTARE LE PAROLE X_i AD UNA DESTINAZIONE, RESISTERE ALLA SECONDA S \hookrightarrow SECONDA PAROLA X_i IN I
 \rightarrow PUÒ ESISTERE L'INCERIMENTAZIA QUALESiasi f COMBINATORIA

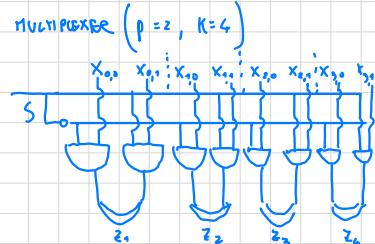
- DECODIFICATORI: TRASPORTO $N_2 \rightarrow N_{10}$
 n INGRESSI E 2^n USCITE
 \hookrightarrow E: ENABLE \hookrightarrow IN QUEL CASO



- CODIFICATORI: TRASPORTO $N_{10} \rightarrow N_2$ $m=2^k$
 2^k INGRESSI E K USCITE



- PRIORITY: SE PIÙ INGRESSI UTILLI \rightarrow SEI DAI PIÙ DEI PIÙ
 QC BIT DI PESO MAGGIORI \rightarrow CI PUÒ ESSERE SOLO 1 $X_i = 1$

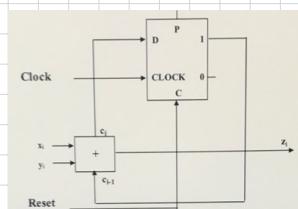


• NUOVI ARITMETICI:

• SOMMATORI:

- SOMMATORI:
 - MINIMO NUBERO DI PORTE LOGICHE MAX t gli disponibili
 \rightarrow FF ritornate a causa di UN BIT a LO USA PUR IL CICLO SUCCESSIVO

ΣIMMA n BIT
 \hookrightarrow n CLK



• CONTINATORI:

- CIRCUITO A 2 STABILIMENTI, M BIT D'INGRESSO, M+1 BIT D'USCITA
 \rightarrow SI RICHIESTA UNA TABELLA DI VERITÀ
 \rightarrow OTTIME IN COSTO DI HW E USOLOGIA

\hookrightarrow SOMMA // DI n BIT

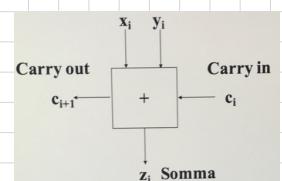
• MULATORES:

- SOMMA X E Y IN X_i, Y_i : SOMMA A PARTE DEL LSB
 \rightarrow BASATO SUL FULL ADDER

• FULL-ADDER: (NUOVO PASSO) \hookrightarrow IN I

- CIRCUITO CHE SOMMA DI $X_i, Y_i + C_i$, PRODUENDO $Z_i + C_{i+1}$

$$C_{i+1} = X_i Y_i + X_i C_i + Y_i C_i \quad \hookrightarrow C_{out} = 1 \quad (\Rightarrow \text{ALmeno 2 ADDERI} = 1)$$



• RIPETERE CARRY-ADDER: *

SI PUÒ SCRIVERE

- CIRCUITO NEL CASCATA DEL ULTIMO C_n
 \hookrightarrow Ogni ADDER È UNA CONTINENTE

V: bit dopo = bit -> 2 SEGNALE:

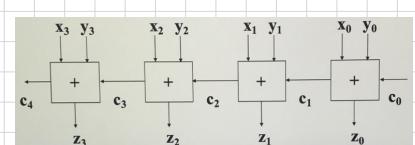
- GENERAZIONE: $y_i = x_i \oplus v_i$

• PROPAGAZIONE: $p_i = x_i + v_i$

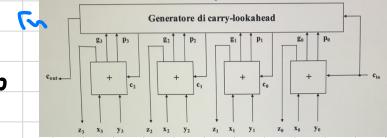
\hookrightarrow C'è un'altra

VERSIONE DI CIRCUITO CON I x : IL bit DI CARRETTA È GENERATO

ESENTE ASSEGNARE UNA PAROLA PRECEDENTE



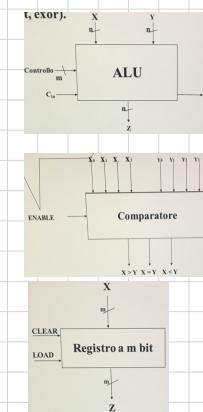
\hookrightarrow LAVORO IN // DEI FULL ADDER



SI PUÒ SCRIVERE
 SEGUENTE UN APPROSSIMAZIONE
 TRA I 2
 \rightarrow $c_i = x_i \oplus y_i \oplus p_i$
 $c_i = x_i \oplus y_i \oplus p_i + c_{i-1}$
 $c_i = x_i \oplus y_i \oplus p_i + c_{i-1} + p_i c_{i-1}$

ALU:

COMPONENTE COMBINATORIALE CHE SVOLGE OPERAZIONI ARITMETICHE E LOGICHE



COMPARATORI IN SERIE DI 16 BIT
L'USO CONVERGA A 4 BIT

COMPARATORI:

3 USCITE $X > Y$
 $X = Y$
 $X < Y$

PRODUCE COME RISULTATO IL CONFRONTO TRA NUMERI

REGISTRO A M-BIT

LOAD: ALIMENTA DATI \rightarrow CONSEGNA AL CLOCK

CLEAR: RESET DEL FFD \sim INIZI. TRACCIATO

A SIGHANNO: FFD DIPENDENTI TRA loro

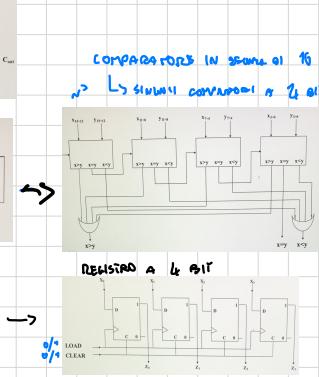
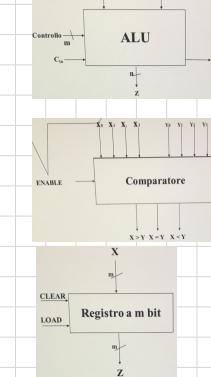
\rightarrow SE SHIFT ENABLE \rightarrow SPOSTAMENTO DI SX IN BASE A X
00X

CONTATORI:

FUNZIONANO SULLE K POSIZIONI IN INDIRIZZO L'ESITO FFT COMBINAZIONE
DEI DATTI I CLK

ASINCRONI: funziona DIPENDENTI DAL CLOCCHIO

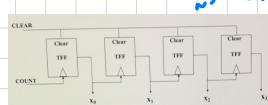
SINCRONI: USATE LE USCITE ASSISTONO LO STESSO VARIANTE



USI: DATA FIFO, CONV. SERIE-//O MULTIV., ETC.



MODO DI PROGRAMMAZIONE



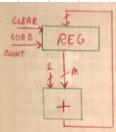
CONTATORE

PCF#FPI

REGISTRO #FPI

ASINCRONICO

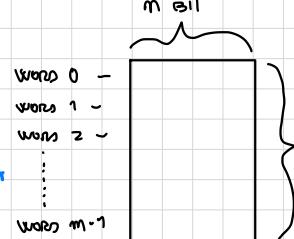
SINCRONICO



K BIT V INDIREZZO

$m = 2^k = \# \text{ registri}$

$m \text{ WORDS} / k \text{ WORDS} \rightarrow 1 \text{ WORD}$
 $\rightarrow 1 \text{ WORD} = n \text{ bit}$



WORD 0 -
WORD 1 -
WORD 2 -
...
WORD m-1

MEMORIE:

INDIRIZZO DI USCITE / 1 CELLA = 1 BIT

USCITA ORGANIZZATA IN PAROLE / PAROLA = m BIT

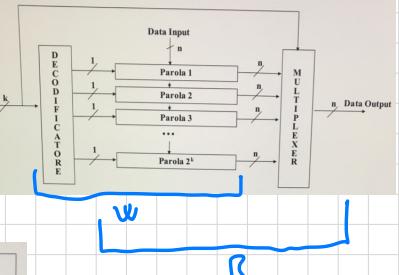
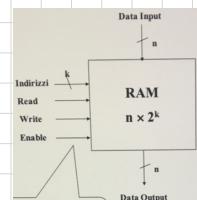
\rightarrow SE PAROLA \rightarrow 1 WORD
L'USCITA È SOGLIA
GURANTE IN m BIT
BIT MA VULTA

DIM. NECESSARIA = $m \cdot n = n \cdot 2^k$

RAM:

RANDOM ACCESS MEMORY

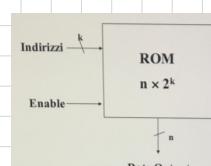
• CICLO DI LETTURA E SCRITTURA



ROM:

SOLA LETTURA

\rightarrow IL VIVO CONTENUTO È SCRITTO NEL MOMENTO
DATA MANUFACTURER E NON COMPOSTO



W
R

4. L'INSTRUMENTO MIPS

- ESECUZIONE ISTRUZIONI: FETCH \rightarrow EXECUTE \rightarrow CICLO DI ISTRUZIONE
 - \hookrightarrow dall'IR
 - \hookrightarrow DESCRIZIONE ESECUSIONE

- REGISTRI: MEMORIE VELOCI ALL'INTERNO DELLA CPU

ASSEMBLY

- SOMMA: add \$a, \$b, \$c $\left(\begin{array}{l} \text{in } C : a = b + c \\ \text{out } a \end{array} \right)$
- SOTTRAZIONE: sub \$a, \$b, \$c $\left(\begin{array}{l} \text{in } C : a = b - c \\ \text{out } a \end{array} \right)$

• REGISTRI MIPS: 32 + 32-BIT

\rightarrow IMMAGINE CON "\$"

• GLI INDIRIZZI DI UNA WORD IN MEMORIA SONO DIVISI IN 4

• ESTRAZIONE IN MEMORIA: lw \$s0, 8(\$t1) $\left(\begin{array}{l} \text{load word} \\ \text{register destination} \\ \text{immagine di indirizzo } \$t1 + 8 \end{array} \right)$

• SCOTTURA IN MEMORIA: sw \$t4, 0x8(\$0) $\left(\begin{array}{l} \text{store word} \\ \text{register source} \\ \text{immagine di indirizzo di destinazione: } \$0 + 8 \end{array} \right)$

• ENDIANESS: / LITTLE ENDIAN: N° BYTE INIZIA DA LSB

/ BIG ENDIAN: N° BYTE INIZIA DA MSB

• lb: LOAD BYTE \rightarrow lb \$s0, 1(\$0)

• sb: STORE BYTE \rightarrow sb

• UN DATO NON ALIGNATO IN MEMORIA (SU DUE INDIRIZZI) È ILLEGGIBILE IN MIPS

• OPERANDI IMMEDIATI: lw e sw USANO OPERANDI IMMEDIATI

\rightarrow addi \$s0, \$s0, 4 $\left(\text{in } C : a = a + 4 \right)$

FORMATI DI ISTRUZIONI:

32-BIT, 3 TIPI:

R-TYPE (REGISTER TYPE)

- OP: DEFINISCE L'OPERAZIONE
- RS-RT: REGISTRI SOUPPORTE
- RD: " DESTINAZIONE "
- FUNCT: DETTAGLI ASSI' OPERAZIONE
- SHAMT: " SHIFT AMOUNT ", SHIFT IN MEMORIA

I-TYPE (IMMEDIATE TYPE)

- IMM: COMPLEMENTO IMMEDIATO SU 16-BIT

J-TYPE (JUMP TYPE):

- INDIRIZZO DI SALTO

• SIGN EXTENSION: SE IL N° DI SOMME È NEGATIVO: ACCORDANDO 1 NEI PARTECIPANTI VERSO DEI SEGUENTI BITI

R-Type					
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
add \$s0, \$s1, \$s2 sub \$t0, \$t3, \$t5	0 17 0 11	18 16 13 8	0 0 0 0	32 34	

Assembly Code

Field Values

op	rs	rt	rd	shamt	funct
add \$s0, \$s1, \$s2	0	17	18	16	0 32
sub \$t0, \$t3, \$t5	0	11	13	8	0 34

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Type

op	addr
6 bits	26 bits

• PC: TIENE TRACCE DEL' ISTRUZIONE CORRENTE (UNIFORMANTE RAGGIRO DELL'ADRES 0x00400000)

• R-TYPE: IL CORDO OP-CODE È TUTTO 0

L'INDICA L'OPERAZIONE DA ESEGUIRE

• ZERO EXTENSION: LE OP. LOGICHE POSIZIONANO ZERI PER COMPLETARE ISTRUZIONI

• AND: MULSISTRUMENTO BITS

• OR: COMBINATORIO BITS

• NOR: INVERSO BITS

• SHIFT:

SPOSTAMENTO DEL BIT IN MEMORIA

• SPOSTAMENTO LOGICO IN MEMORIA:

- sll \$t0, \$t1, 5 ^{→ PASSO}
↳ SHIFT LEFT $\frac{t_0}{t_1}$ $\frac{5}{0}$

- srl \$t0, \$t1, 5 \rightarrow DIVISIONE PER 2^5

↳ SHIFT RIGHT $\frac{t_0}{t_1}$ $\frac{5}{0}$

• SPOSTAMENTO ARITMETICO IN MEMORIA: sra \$t0, \$t1, 5 \rightarrow VIENE PRESO IN CONSIDERAZIONE IL SEGNO

• sllv, sriv, srauv: SPOSTAMENTO DI VARIABILI

• 32-BIT COSTANTI: DIVIDONO IL NUMERO IN 2 PARTI DI 16-BIT

↳ $a = 0xFEDC\ 8765 \rightarrow lui \$s0, 0xFEDC$ \rightarrow cosa upper immagine
ori \$s0, \$s0, 0x8765

• lo/hi:

USATI PER MOLTIPLICAZIONE E DIVISIONE, NON DISSTANZIATE ACCESSIBILI

• MOLTIPLICAZIONE:

mult \$s0, \$s1 \rightarrow 32x32 mantenendo risultato su file int in hi/lo

• DIVISIONE:

div \$s0, \$s1 \rightarrow QUOTIENTE IN LO, RESTO IN HI

\rightarrow PER MUOVERE I RISULTATI IN ALTRI REGISTRI: mfls/mfhi \$s..

• PSEUDO-ISTRUZIONI: NON IMPLEMENTATE NELLA HARDWARE, UTILI PER DEFINIRE IC CODE PIÙ REALEABILI

• BRANCHING:

ESECUZIONE DI ISTRUZIONI NON SEQUENZIALI

• beg: BRANCH IF EQUAL {CONDIZIONE}

• bne: BRANCH IF NOT EQUAL

↳ beg/bne \$s0, \$s1, TARGET

↳ ENTRA: Esiste
 \Rightarrow CONDIZIONE TRUE ($s_0 = s_1$)

• J: JUMP \rightarrow SINTO DI 26 BIT
• Jn: JUMP REGISTER $\left\{ \begin{array}{l} \text{NON} \\ \text{CONDIZIONAVI} \end{array} \right.$
• jal: JUMP AND LINK

↳ j/jal TARGET \rightarrow JR \$s0

↳ ENTRA: Esiste
 \Rightarrow CONDIZIONE TRUE ($s_0 = s_1$)
SUGGERISCE COST.

• if/else:

• while:

C Code

```
if (i == j) {
    f = g + h;
    f = f - i;
    L1: sub $s0, $s0, $s3
    done:
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
bne $s0, $s1, L1
add $s0, $s1, $s2
j done
```

VISUALIZZATA IN CONDIZIONE
OPPOSTA

C Code

```
// determines the power
// of x such that  $2^x = 128$ 
int pow = 1;
int s0 = 0;
int i = 0;

while (pow != 128) {
    if (i == 128) {
        add $s0, $s0, 1
        addi $s0, $s0, 128
    }
    pow = pow * 2;
    x = x + 1;
}
done:
```

MIPS assembly code

```
# $s0 = pow, $s1 = x
add $s0, $s0, 1
add $s0, $s0, 128
addi $s0, $s0, 1
j while

while: beq $s0, $s1, done
sll $s0, $s0, 1
addi $s0, $s1, 1
j while
```

↳ ENTRA:

done:

C Code

MIPS assembly code

for (i=0; i!=10; i = i+1) {

sum = sum + i;

}

MIPS assembly code

\$s0 = sum, \$s1 = i

add \$s0, \$s0, 0

add \$s0, \$s0, \$s1

addi \$s0, \$s0, 10

addi \$s1, \$s1, 1

j for

RISULTATO DI ISB COMBINAZ. VERSI

slt/slti rd, rs, rt/cost.

↳ if(rs < rt/cost.) rd=1

else rd=0

- BRAUN PSEUDO INSTRUCTION: bye, blt, byt, ble \rightarrow set beg/bne ...
 ↳ various version unknown: bye4, ...
 - SWITCH:

• SWITCH :

1. SALVO L'INDIREZZO DI JUMP TABUE IN \$t4
 2. SALVO IN \$t7 IL VARIANTE DI \$S5 ($K_1 \times 2^2 \rightarrow$ IN \$t1 CI SONO L'OPERE DI K)
 3. SALVO IN \$t7 IL NUOVO INDIREZZO, CIOÈ L'INDIRETTO DI LO/L1/L2
 4. CARICA IL CONTENUTO IN \$t6 E USO CA JR PER ESEGUIRE IL CASE DELL'ULTIMA COMMA

\rightarrow `void *m[10]` contains $\$f_0 = \&L + \&\text{JUMPTABLE}$, in which some
contains the instruction $L0, L1, L2$

• ARRAY :

INN111220 1995: 0x1234800 (anher [0])

→ own business result strategy is aggressive: anonymous 4

821 IN 012120 01 MURWRA PROJECTIVE

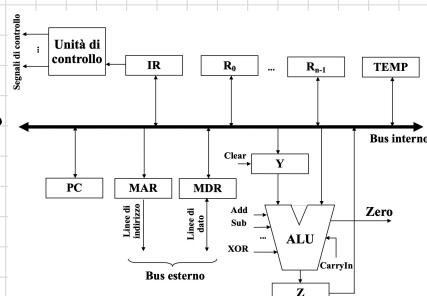
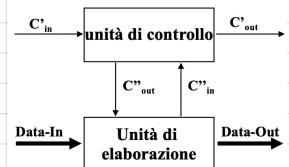
↳ Ques \$61, 4(\$50)
 ~> zero in \$61 array [1]

0x12340010	array[4]
0x1234800C	array[3]
0x12348008	array[2]
0x12348004	array[1]
0x12348000	array[0]

- **STRUCTURE :** . ASCII " STRNBUF "
 - **CONTRATE :** . byte char "
 - SYSCALL: permette di INFORMARE col SO
 → IL TIPO DI AZIONE E' CONTENUTO IN \$V0

Name	Register Number	Usage
\$0	0	the constant value 0
\$at	1	assembler temporary
\$v0-\$v1	2-3	function return values
\$a0-\$a3	4-7	function arguments
\$t0-\$t7	8-15	temporaries
\$s0-\$s7	16-23	saved variables
\$t8-\$t9	24-25	more temporaries
\$k0-\$k1	26-27	OS temporaries
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	function return address

procedure	code \$v0	result
read int	5	\$v0 contains the number
print int	1	\$a0 contains number to print
print string	4	\$a0 address of string
exit	10	End of program
print char	11	\$a0 contains char to print
read char	12	\$v0 contains the char



- #### - 4 OPERAZIONI PRINCIPALI DELL'CPV:

- ① **FETCH**: PRECIEVA DI ISTRUZIONE / DATO DA MEMORIA E LO INVIAVA IN REGISTRI
 - ② **STORE**: IN MEMORIA USC CONTENUTO DI UN REGISTRO
 - ③ **TRASFERIMENTO TRA REGISTRI**
 - ④ **ESECUZIONE**: DI OPERAZIONI ARITMETICO / LOGICHE
o risultato in memoria

- MAR : MEMORY ADDRESS REGISTER → CONTAINS THE ADDRESS OF RAM & ROM IN CPV

- MDR: " DATA " -> " RAM ECONOMIZARE PENTRU CPU, și cu 2 ACU și ACUSSR

- ③ Ogni registro è collegato al bus, controllato da 2 segnali:

R_{IN} : il rendono carico il varore nei bus

R_{OUT}: 16 CONTENUTO DEC REGISTRO TRASFERITO SUL PGS

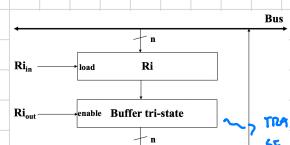
SEE ATTIVI

→ BUFFERED TEL-STATE: SI COMPRA CON UNA POISI → CHIUSA SE RICHIESTO → ALTA IN PONTE

- TRASFORMATORES ON R1 + R24: ATTRAVERSO R1 OUT, ATTRAVERSO R26 IN

L, DATA SU QUESTO

↳ memorizzato il valore su guy



TRANSIENTS
SE WESTSIE
SIA R OUT W
SUL BUS

- $V_{clock} \rightarrow 1$ solo R_i OUT ATTivo \rightarrow GENERA COLLISIONI SENSO $\left\{ \text{VINCOLI} \right.$
- $V_{clock} \rightarrow 1$ solo R_o attiva e sostituisce dato sullo stesso R_i

ACCESSO ALL'ADDRESS BUS (CONNESSIONE DI MAR):

SEGUONO LE MIGLIORI ISTRUZIONI $MAR \leftarrow R_j$:

- \rightarrow ATTivo R_1 OUT, ATTivo MAR_{in} ATTivo MAR_{out}
 \hookrightarrow Ogni sul BUS \hookrightarrow Bus di memoria \hookrightarrow Goto sul BUS

② (CONNESSIONE DI MDR):

2 ② $MDR \leftarrow \text{BUS INTERNO}$

ACCESSO IN SCRITURA

- $\rightarrow SEL = 1$, ATTivo MDR_{in} , ATTivo MDR_{out} , ACCESSO IN SCRITURA

RETURNA: $MDR \leftarrow \text{DATA BUS}$

- $\rightarrow SEL = 0$, ATTivo MDR_{in} , ATTivo MDR_{out} , ATTivo R_i , IN

LINEA DI SELEZIONE: $0 \rightarrow$ RETURNA DA SCRITURA

$1 \rightarrow$ SCRITURA DI SCRITURA

TEMPORIZZAZIONE: ciascun segnale ha una propria PARTE DI TEMPO DI ESECUZIONE ECC' ISTRUZIONE, considerati anche i vari ritardi fra i componenti

CPU - MEMORIA:

• LA MEMORIA IMPLICA QUESTO TIPO DI SCHEMI / SCHEDE:

- \rightarrow QUANDO COMPIETTA ATTIVA MFC (MEMORY FUNCTION COMPUTER) \hookrightarrow BUS ASINCRONO

• RETURNA PAROLA DATA MEMORIA:

LOAD $R_2, [R_1]$: $MAR \leftarrow R_1$, ATTIVO SEGNAL RETURNA, ATTIVATO MFC, MDR \leftarrow BUS ESTERNO, $R_2 \leftarrow MDR$

• SCRITURA PAROLA IN MEMORIA:

STORE $R_1, [R_2]$: $MDR \leftarrow R_1$, $MAR \leftarrow R_2$, ATTIVO SEGNAL RETURNA, ATTIVATO MFC

①

- SEGNALI ISTRUZIONE SUCCESSIVA
- AGGIORNAMENTO PC

$\rightarrow MAR \leftarrow PC$, $Y = 0$, $COREN = 1$, $Z = PC + Y + CARRY$, ATTIVO SEGNAL,

PC $\leftarrow Z$, ATTIVATO MFC, MDR \leftarrow BUS ESTERNO, IR \leftarrow MDR

\hookrightarrow SOGLI CONTROLLI VENUTI NELL'IR SONO DOPO MFC
 $(SSC = 0, MDR_{in}, MDR_{out}, IR_{in})$

• SALTO INCONDIZIONATO:

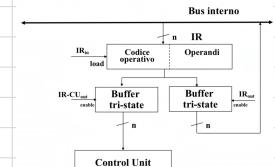
- $\rightarrow PC \leftarrow (\text{campo IR})_{out}$

SEGNALE: $(\text{campo IR})_{out}, PC_{in}$

• PC: CONTIENE L'INDIRIZZO DELL'ISTRUZIONE

• IR: CONTIENE L'ISTRUZIONE

Connessione di IR



• SALTO CONDIZIONATO:

• CI SONO DUE TIPI DI CONDIZIONE: ZP (ZERO FLAG): 1 se risultato nullo

SP (SIGN FLAG): 1 se risultato negativo

- $Y \leftarrow R_1$
- $Z \leftarrow Y - R_2$
- $Y \leftarrow PC$, if Zero = 0 then
 - $Z \leftarrow (\text{Campo operando di IR})_{out} + Y$
 - $PC \leftarrow Z$

5. UNITÀ DI CONTROLLO:

- PROGETTAZIONE: DIAGRAMMA A STAM

- IMPLEMENTAZIONE, 2 CASI - CU CABARET: (U COME circuito sequenziale con regoli trasizionali predefiniti)

CU microprogrammato: A operazione \rightarrow ISTRUZIONE

CU CABARET:

MORSO HUFFMAN

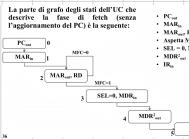
mentata come una macchina a STAM FINITI

\rightarrow UN STATO SI DEFINISCE LA COMBINAZIONE D'USCITA

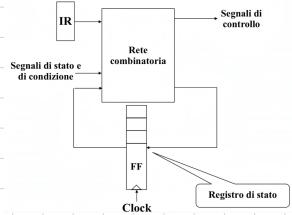
\hookrightarrow VERSO UNITÀ DI SCELZAMENTO

\hookrightarrow SI LENDO POI UNA TABELLA DI STAM

* SVANTAGGI: LA DURATA DEI PROGRAMMI E' TROPPO GRANDE PER PUÒ ESSERE RISERVATA
MODULARE ESTREMAMENTE CONNESE, MAX VELOCITÀ, MIN COMPLESSITÀ



\hookrightarrow FETTO DI TRASMETTE
UN TUTTO DEGLI STATI



CU MICROPROGRAMMATO:

CIASCUA ISTRUZIONE CONTIENE NELLE MICROISTRUZIONI, RENDIMENTO IN UNA APPOSITA MEMORIA

FUNZIONAMENTO:

• LEGGE DA ①, TRAMITE L'INDIRIZZO CONTENUTO IN ②

MEMORIA DI
 \hookrightarrow Microcodice

ISTRA. SUCCESSIVA

INDIRIZZO. EST.

INDIRIZZO DI SOTTO

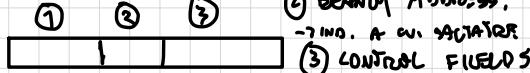
• ③ CORRISPONDENTI ALL'INDIRIZZO IN ③

• ③ PILOTA I SEZIONI PER ④ E PER LA ISTRUZIONE SUCCESSIVA (NOME DI PILOTA DI SUC. ISTRUZIONE)

\rightarrow FINITO IL FUNZIONAMENTO DURANTE UN SOLO COUPON DI CLK

MISTRUZIONI:

• FONDATO: MISTRU. SUCCESSIVA + CODICIA SERVIZI DI CONTROLLO



\hookrightarrow LE ISTRUZIONI DI SOTTO HANNO UN CAMPO ADDIZIONALE PER L'INDIRIZZO DI SOTTO

• SE MISTRU. SUCCESSIVA: INDIRIZZO MISTRU. CONSO. $t=1$

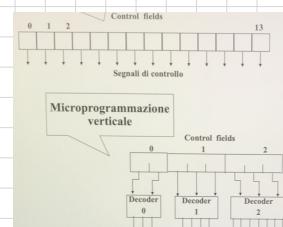
\hookrightarrow LINEE DI USCITA DI INIZIAZIAZIONE DI SECUZIONE

• ORIZZONTALE: 7 BIT PER 7 SERVIZI DI CONTROLLO \rightarrow 7 SERVIZI 1 BIT

• MICROPROGRAMMAZIONE / VERSIONE: 7 MISTRUZIONI \rightarrow CODICE, IL DECODER ESEGUE MISTRU. E LE CARICA
 \hookrightarrow PRESTAZIONE (PIÙ ALTA), MA VELOCITÀ E COSTO NO

• COMPATIBILITÀ: 2 SERVIZI DI CONTROLLO SONO COMPATIBILI SE NON SONO MAI ATTIVATI NELLA STESSA ISTRUZIONE

\hookrightarrow CLASSI DI COMPATIBILITÀ: GRUPPI DI SERVIZI SOTTO COMBINABILI



\hookrightarrow ORIZZONTALE.

\hookleftarrow VERTICALE: I SERVIZI DA ATTIVARE SONO CODICI DI INIZIAZIONE

6. • ARCHITETTURA MIPS

- DATA PATH (Nero), CV (Blu)

ESTRADA DA MIPS

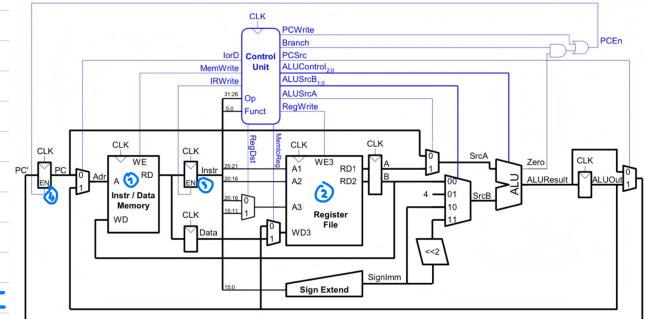
① MEMORIA: Lavora ad OOM CLK, ESEGUE UNA WRITE
O READ, IN BASE AL SEUWE WE

② REGISTER FILE: Lavora ad OOM CLK, ESEGUE 2 WE
O SCRITTURA, IN BASE A WE3

- 3 TIPI DI ISTRUZIONI:

③ IR

④ PC



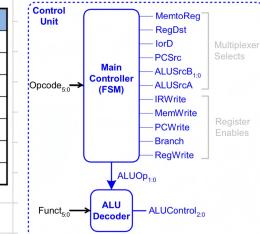
- R-TYPE: AND, OR, ADD, SUB, SLT
- MEMORY INSTRUCTION: LW, SW
- JUMP: BEQ, JR.

• PROCESSORE MIPS:

ALU:

ALUOp _{1:0}	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

ALUOp _{1:0}	Funct	ALUControl _{2:0}
00	X	010 (Add)
X1	X	110 (Subtract)
IX	100000 (add)	010 (Add)
IX	100010 (sub)	110 (Subtract)
IX	100100 (and)	000 (And)
IX	100101 (or)	001 (Or)
IX	101010 (slt)	111 (SLT)



• SINGLE CYCLE: 1 CLK → 1 ISTR.

• MULTICYCLE: 1 ISTR. ESEGUITA SU PIÙ CLK, DIVISA IN STEP MINORI

• PIPELINED: VARIANTE MULTICYCLE → + ISTR. ESEGUTE MOLTI VOLTI

• MIPS STATE ELEMENT: ASSISTINTA AL STATO DI ESECUZIONE DI UN PROCESSORE → WRITE SYNCHRONO CON CLK
↳ READ ASINCRONA

• FETCH: ~2 CLK

1. SEND ADDRESS TO MEMORY
2. WRITE INSTR. IN LR AND UPDATE PC

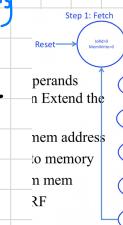
PRESEN IN: LW, SW, BTB: BOUND TO ADDRESS
R-TYPE INSTRUCTION, BEQ, JR [rs], [rt], PC+PC



• LW: ~7 CLK

REGISTERS
FILE

LW rt, offset(rs)



• SW: ~5 CLK

SW rt, offset(rs)

• R-TYPE: ~5 CLK

1.2.

3. READ SOURCE OPERAND FROM RF/WRITE SOURCE OPERANDS IN REG. A/B
4. PERFORM ALU OP
5. SEND RESULT TO RF

BEQ, JR ~3 CLK

• PERFORMANCE PROCESSORE:

$$\text{TEMPO E SEC.} = (\# \text{istr.}) \left(\frac{\text{n° celle}}{\text{istr.}} \right) \left(\frac{\text{secondi}}{\text{cycle}} \right)$$

• MULTICYCLE OLTRE PATTI: $T_C = t_{\text{pcy}} + t_{\text{mux}} + \max(t_{\text{alu}}, t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}$

7. - MEMORIE :

• SEGNALI DI CONTROLLO:

- TIPO DI OPERAZIONE RICHIESTA (R o W) {
 - DISPONIBILITÀ DI INDIRIZZI SUL BUS
 - DISPONIBILITÀ RAM SU BUS (W only)
- INDIRIZZO RAM
SARIE AVANZO
DARRESE
- SINCRONO
ASINCRONO
- ASINCRONO

(ASINCRONO)

- CS (chip select): ATTIVATO PER LETTURA O SCRITTURA
- OE (output enable): .. per lettura su un bus comune
- WE (write enable): genera
- SEGNALI DI STATO: ERRORE: PAROLE CON DATI ERROSI
MFL: W/R COMPLETATA
- DIMENSIONE: \uparrow PAROLA \times m BIT \downarrow PAROLA

A VETTORE: \forall CELLA \rightarrow PAROLA, N PAROLE POSS LE N CELLE.

• STRUTTURA: $\hookrightarrow \log_2 N \rightarrow N$ DECINA

A MATRICE BI-DIMENSIONALE: \forall INDIRIZZO \Rightarrow $\text{HO } R_x \times Q_y \rightarrow$ IDENTIFICA LE COORDINATE DELLA CELLA
 $\hookrightarrow \log_2 \sqrt{N} \rightarrow \sqrt{N}$ DECINA (2 DECINA)

\rightarrow PER OTTIMIZZARE IL N. DI SCANI IN INGRESSO, ALFINE METTENDO A RUSSO: SPRAVITAM ' SEGNALI RAS E CAS;

\rightarrow L'INDIRIZZO È FORNITO IN 2 FASI DISTINTE:

- SEGNALI DI INDIRIZZO E RECODIFICAZIONE PAR2 CELLE + SEGNALI RAS (ROW ADDRESS SIGNAL) \hookrightarrow IN ALLEVA
- " " " " " " " " " " " " CASS (COLUMN ADDRESS SIGNAL) \hookrightarrow MEMORIA

• PARSE MODE: IN CASO SI DESIETE AGIRE SUI INDIRIZZI CONSERVATI IN MEMORIA: NON RICHIESTO RAS MA IL 2° ULTERIO

\hookrightarrow FAST PAGE MODE: ACCESSO A BLOCCHI DI PAROLE SULLA STESSA PAG.

• MEMORIE A SEMICONDUTTORI:

\rightarrow LE CELLE SONO COSTITUITE DA TRATTORI DI SEMICONDUTTORE (ved. SISTEMA)

• ROM (read only memory): IL CONTENUTO È STABILITO DALLA PRODUZIONE \rightarrow NON È RIDUCIBILE

• PROM (programmable ROM): UNA VOLTA PROGRAMMATI, SONO PROGRAMMATI UNA VOLTA PER SEMPRE (BRUCIAMO IL PULIBRISE P)

• EPROM (ELETTRICAL PROM): RIPROGRAMMABILI, PREVIA CANCELLAZIONE SOLO SOTTO RADICI UV \hookrightarrow PIÙ COSTOSO

• EEPROM (ERASABLE EPROM): RIPROGRAMMABILI BYTIE PER BYTIE

• FLASH: 1 BIT \rightarrow 1 TRANSISTOR (FLOATING GATE TRANSISTORE)

\hookrightarrow MOSTRA

\hookrightarrow USARE PER LE PARTICOLE DI MASSA

- LETTURA: ATTIVO (1) \rightarrow (3) PORGIATO A ACTA/BASE TRASLONE IN PAGE AL CONTINUA IN (2)
- SCRITTURA: INTRODUCEO E' IN (2)

• NAND FLASH:

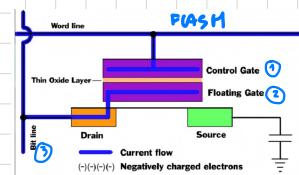
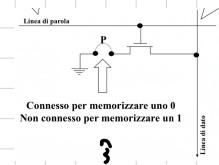
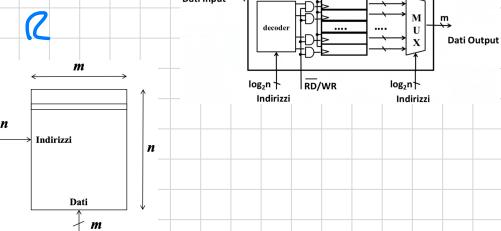
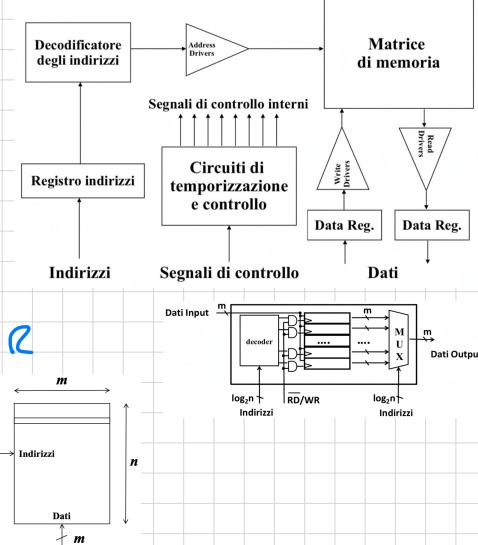
SLC (single level cell):

\forall CELLA \rightarrow 1 BIT \sim 2 VALORI POSSIBILI

MLC (multiple level cell):

\forall CELLA \rightarrow 2 BIT \sim 4 VALORI POSSIBILI

Tipo	Categoria	Cancellazione	Scrittura	Volatilità
RAM	read/write	elettricamente	elettricamente	volatile
ROM	read-only	impossibile	in fase di produz.	non-volatile
PROM	read-only	impossibile	elettricamente	non-volatile
EPROM	read-mostly	luce UV	elettricamente	non-volatile
EEPROM	read-mostly	elettricamente	elettricamente	non-volatile
Flash	read-mostly	elettricamente	elettricamente	non-volatile



RAM:

- \ STATIQUE (SRAM): CEEUA \rightarrow 1 FLIP-FLOP 1 FF SR
- \ DINAMIQUE (DRAM): CEEUA \rightarrow 1 CONDENSATORE + 1 TRANSISTOR, RAM DINAMICO USANO UNO O PIÙ CONDENSATORI, RINFRESCO PERDONO 25% INFORMAZIONI 1 COND. + 1 TRANS.

SRAM: CEEUA: 6T

PER LEGGERE/SCRIVERE DOVETE ATTIVARE LA LINEA DI PAROLA, ATTIVATA IN PP RITRICE RENDONO IL VADOO

• LETTURA / SCRITTURA: ATTIVA LINEA DI PAROLA
e FORZA I BIT SULLE 2 LINEE DI DATO

• IMPLEMENTAZIONE CMOS: CIASUNA CELLA DI RAM RICHIESTE 6 TRANSISTORI (6T-RAM)

DRAM: CEEUA: 1T

• LETTURA: VERRÀ ATTIVATA ①, IN QUESTO MOMENTO DI SCRITTURA NEL C, IL DATO È PASSATO SU ②

• SCRITTURA: VERRÀ ATTIVATA ②, I DATI VERRANNO SCRITTI IN BASE A CIO CHE C'È SU ②

• RINFRESCO: IL VAGLIO IN C NECESSITA DI ESSERE ACCESO, QUANDO QUESTO TIPO DI RAMA È ACCESO, VERRANNO FATTE DUE LETTURE ELETTRICHE, IN MODO CHE C SI COMUNI, \rightarrow IL DATO NON VERRÀ POK TRAVERSO SU ② ↗ R

• AFFIDABILITÀ: SE SONO EFFETTUO I RINFRESCHI, LE DRAM POSSONO SUSTENERE INFORMATI IN MEMORIA FINO A 20 ANNI

\rightarrow SI USANO DEI CODICI DI PROTEZIONE

CODICI DI PROTEZIONE: ~ 7 BIT DI PARITÀ

• CODICE DI PARITÀ:

UN UNICO BIT

• QUANDO SCRIVI UNA PAROLA CREA UN BIT DI PARITÀ E LO SALVI
 \rightarrow IN LEGGERIA, LEGGI UN BIT DI PARITÀ, E CONTROLLA SE IL BIT È CORRETTO

\rightarrow ERRORE SENNO

• CODICE DI HAMMING: $\sim n$ BIT DI PARITÀ $\rightarrow 1 + \log_2 n$ BIT DI PARITÀ

È POSSIBILE RILEVARE & CORREGGERE TUTTI I TIPI DI ERRORE (SOLI SE SINGOLI, ERROTI DOPPI)
(RILEVATI MA NON CORRETTI)

TUTTE LE DRAM HANNO OGNI ECC (CORTELLA CORREZIONE ERRORE) (es. HAMMING)

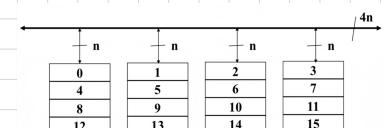
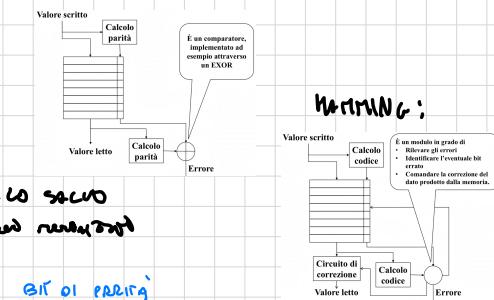
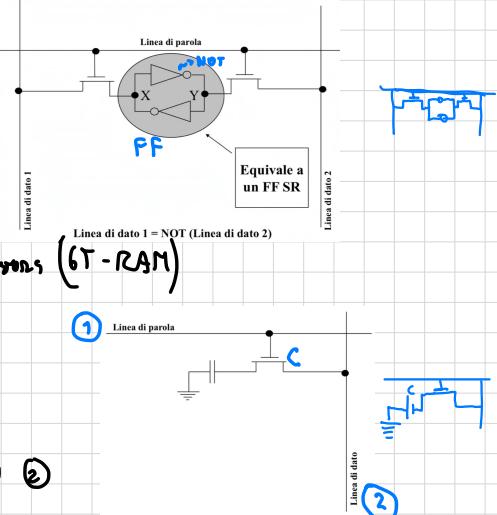
SRAM (rispetto alle DRAM):

- + PIÙ VECCHI (ACCESO 10 ns VS 100 ns)
- PIÙ COSTO
- + PIÙ STABILI NELL'UMIDITÀ
- + PIÙ APPROPRIATI

RISORSE INTERACCIAZIONI:

MOLTI PIÙ RISORSE DI RISPOSTA \rightarrow WORKS DISSERTE IN DUE ACCESSI

\rightarrow PER ACCEDERE A PAROLE CONSECUTIVE SI PUÒ UTILIZZARE UN ACCESO IN PARALLELO \rightarrow PIÙ RISORSE



• MIPS FUNCTION CALLS:

- CALLER: f_1 CHIAMA CHIAMATO DI f_2
- CALLEE: CHIAMATO DA f_1

jal PUNCT

:

Jr \$t0

- jal (JUMP AND LINK): SERVIRE PER CHIAMARE UNA FUNZIONE

-> es. jal FUNCTION

-> MODIFICA PC PER SVOLTARE A PROC &

-> COPIA IN \$t0 L'INDIRIZZO A UN RETORNO, CHE È CALLER

jal proc

31 26 25 proc 0

- JR: RETURN DELLA FUNZIONE

-> es. JR \$t0

- \$a0 - \$a3: ARGOMENTI DA PASSARE ALLA FUNZIONE

- \$v0: VALORE DI RITORNO

• STACK:

\$sp: MEMORIZZA L'INDIRIZZO DELLO STACK POINTER

CHE INIZIA CON CRESCEndo VERSO IL BASSO AL CRESCERE DEGLI PUSH

• STACK FRAME: SEZIONE DELLA p che SERVIRÀ PER SVOLGERE LE SUE VAR.

(LEAF FUNCTION: NON CHIAMA ALTRE FUNZIONI)

NON LEAF FUNCTION: CHIAMA ALTRE FUNZIONI

↳ PRIMA DI CHIAMARE ALTRE \neq NECESSITA DI SALVARE I VALORI DI \$t0 - \$t6 e \$a0 - \$a3
 ↳ LE SALVA NELLO STACK, PER Poi ELIMINARSI

(METODO ACCIARANNO!
 GUARDA SO.
 ES 1-2 UB. 5)

PUSH:

addi \$sp, \$sp, -12] ALLOCARE MEMORIA
 sw \$s0, 8(\$sp)] (3 WORDS)
 sw \$t0, 4(\$sp)] SALVARE 3 VAR. NELLA
 sw \$t1, 0(\$sp)] STACK

POP:

lw \$t1, 0(\$sp)] CARICO LE 3 VAR IN
 lw \$t0, 4(\$sp)] STACK NEI REGISTRI
 lw \$s0, 8(\$sp)] OBBLIGO DI SPAZIO
 addi \$sp, \$sp, 12] \Rightarrow \$sp TORNA ALLO SP

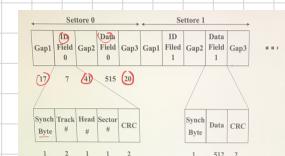
8. - MEMORIE SECONDARIE E OFF-LINE:

DISCHI MAGNETICI \rightarrow HD o SSD

• MEMORIA A DISCO MAGNETICO:

\rightarrow CONTROLLO INTERNO LO STESSO TI DI BLT

DISCO MAGNETICO CON TRACCE CONCENTRICHE \rightarrow Ogni TRACCIA È DIVISA IN SETTORI
 OGNI SETTORE HA UNA TESTINA CHE PUÒ MUOVERSI RADIALMENTE \rightarrow Z canali / DATI



• LETTURA: POSIZIONA LA TESTINA SULLA TRACCIA, ESTRAE DATI DA TESTINA \rightarrow LATENCY

• TEMPO DI ACCESSO: $t_A = t_s + t_L + t_o$ / $t_s = SEEK TIME$ \rightarrow t per IC Poi SUGLIO DATI CORRETTI
 $t_L = LATENCY TIME$ \rightarrow t Dopo TESTINA SUO SETTORE

• VELOCITÀ TRANSFER: $T \times V / T$ DENSITÀ DI NUMERAZIONE (bit/in) $t_o = DATA TRANSFER TIME$
 $V = VELOCITÀ TESTINA SUO TRACCIA (cm/sec)$

• DISK DRIVER: CONNETTE DISCO AL DISK CONTROLLER \rightarrow SI INTERFACCIA CON PROCESSORI E MEMORIA

\rightarrow INCHIUSO: DATA BUFFER E INTERFACCIA

• UNITÀ A STATO SOLIDO (SSD)

\rightarrow NON SONO AD ACCESSO SEQUENZIALE
 \rightarrow DI TIPO FLASH

• CODIFICA DI PHASE (o MANGIATORE):

\rightarrow BIT MIGRATORI IN MAMMA ASSORDENTE

O DISGENDENTE



NOR FLASH: MEMORIA CONCESA IN UN SoC

NAND FLASH

Architettura SSD:

① CONTROLLER CON CACHE DI SUPPORTO

② INTERFACCIA

- moduli di memoria Flash

①

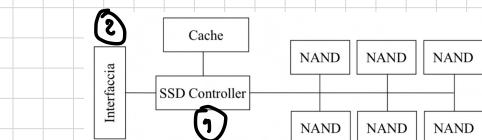
• SSD CONTROLLER:

covrono la NAND Flash e l'interfaccia di I/O verso l'esterno

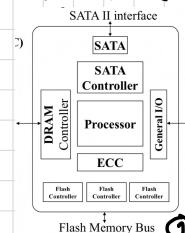
• SSD FLASH:

~> con moduli da 1+512 GB

una o più moduli di memoria Flash, 1 ECC + pagina

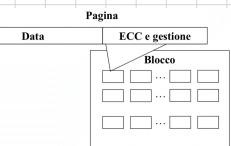
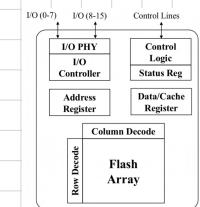


SSD CONTROLLER



Flash Memory Bus

SSD FLASH



② INTERFACCIA SSD:

covrono l'SSD al sistema di elaborazione

512B-4KB



92-128 Pagine



• DE NAND Flash sono organizzate in pagine e Ogni pagina è organizzata in blocchi

↳ R/W eseguiti sulle stesse pagine, ma la conservazione avviene per blocchi

↳ SD, compact flash, USB, ecc.

Vantaggi	
• Alte prestazioni: significativamente maggiori rispetto a HDD	
• Seek time ridotto: fino a 60x più veloci degli HDD	
• Maggiore affidabilità: nessuna parte meccanica in movimento, fino a 1500Gs di resistenza a shock meccanici	
• Minore consumo di potenza	
• Operatività silenziosa	
• Leggere: perfette per dispositivi portabili	
• Temperatura di esercizio maggiore	

MEZZI DI STORICO:

• MEZZI DI STORICO TRADIZIONALI:

• DATI IN PIASTRA SU CUI INCIDERE/LEGGERE TRACCE PARALLELE (es. g di solito)

↳ ORBITA DI UNA SUTTA FISSA -> SOTTOVIA

IN CONTINUA ROTAZIONE

MEZZI DI STORICO OTTICI:

Sono dei dischi ottici

↳ CAV (velocità incisiva cost.): disco con tracce concentriche -> tracce minori verso l'esterno

↳ CLV (velocità lineare cost.): disco legge le stesse tracce con traiettoria a spirale
↳ CD, DVD, ecc.

• CD: non reversibili, dati audio, ~60 min. di registrazione

• CD-RW: 1 sola traccia a spirale, info reversibile in forma elettronica -> rayo laser interattivo rosso/azzurro

• CD-RW: <0 rigonfiamenti ~3 minuti riproduzione: Vsinistra x Vintervento x Vverso, Lavoro così raffigurato D (RW)

• DVD: tracce minori di 135 mm

• Blu-ray: 1000 Mbit/s di dati

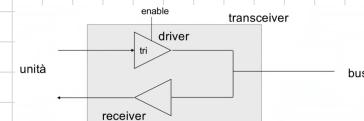
SSD	HHD
NAND Flash	Tecnologia
64GB	Capacità
75gm	Peso
Lettura: 100MB/s	Prestazioni
Scrittura: 80MB/s	
1W	Potenza
1500Gs	Resistenza Shock
0°C - 70°C	Temperatura di esercizio
MTBF > 2M ore	Robustezza
	MTBF < 0.7M ore

9. BUS:

TROPIETÀ I OAM

2 ISTRUZIONI

DRIVER: PIROVA LINEE DEL BUS
RECEIVER: LEGGE I VOLTATI SUL BUS



* POSSONO ESSERE: INTERRUTTORI CIRCUITO INTEGRATO / CONDENSATORI CAVO INTRICO / CONNESSIONE DI SCHERMI (BACKPLANE)

* 3 GRUPPI DI SEGNALE:

* di DATI: multipli di 8, BIODIREZIONALI O UNIDIREZIONALI dBUS

* di INDIRIZZO: identifico lo SLAVE con cui il MASTER vuole comunicare. aBUS

* di CONTROLLO: STATO, TEMPORIZZAZIONE, ecc. cBUS

* BUS MULTIMEDIALE: diversi tipi di dati, in tempi diversi succedono su

* CONTROLLARE IL RETROPERA:

- COMMUTAZIONE TDZ: necessarie a causa dei latimenti

- GESTIONE PRIORITÀ

- ATTIVARE SEGNALI DI TEMPORIZZAZIONE;

NEI SISTEMI PIÙ AVANZATI CPU È MASTER

* MASTER: segue lo SLAVE con cui comunica e invia il TRASF. DATA UNITÀ COMUNICANTE

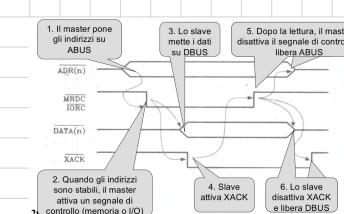
* SLAVE: risponde al comando del master nEL BUS

* GESTIONE DELLE TEMPIGLIUS:

- BUS SINCRONI: LE UNITÀ CONNESEZIONI SFRUTTANO LO STESSO CLK

- BUS ASINCRONI: INIZIANO L'HANDSHAKING \rightarrow FEEDBACK BIODIREZIONALE PER LA COMUNICAZIONE
↳ MIGLIOR PRESSIONE

SINGOLI
MULTIPLO



* CICLI DI WAIT: permettono agli SLAVE di OTTENERE PIÙ TIPO DI ESSERE UN'OPERAZIONE

* ARBITRAZIONE:

ENTRA IN PUNZ. QUANDO 2 UNITÀ RICHIUDONO DI ESSERE MASTER CONTEMPORANEO

CONTRACCEDIMENTO: 1 SOLO ACCESSO PER UNITÀ

DISPENSAMENTO: 1 UNITÀ \rightarrow SOLO UNITÀ CHE PUÒ ESSERE MASTER

BUS SCSI:

* 8 LINEE A PARITÀ (7 P. DATI)
 \rightarrow FISSO A 8 DISE. SI POSSONO CONNETTERE \rightarrow UTILIZZARE POI UNICI PER PRIORITÀ

* INTRACOMBINO:

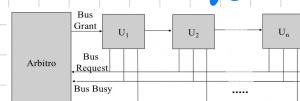
* DAISY CHAINING: UN'UNITÀ ACCONE IL BUS (BUS REQUEST) \rightarrow SE BUS Busy = 0, ABILITATO

3 SEMAFORI ATTRAVERSO BUS (GRANT): LE U_i SI PASSANO IL SEMAFORO

NO MODIFICHE POSSIBILI NO ELEVATO IN GIRO. ANCHE U_X CHE HA FATTO RICHIESTA IN QUESTA MISTRA BUS Busy = 1

SENZA ATTESA

DC



B REQUEST
B BUSY
B GRANT

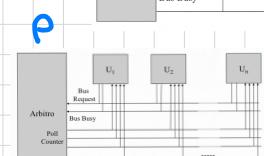
* POLLING: LE UNITÀ VENDONO MESE IN SEDE SU POLL COUNTER

62 + 8^n SEMAFORI \rightarrow C'È UNA UNITÀ DI BUS PER COMUNICARE

PRIORITÀ MODIFICABILE

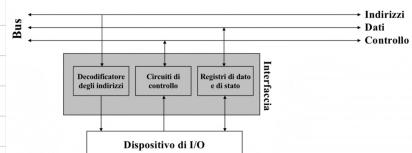
* RICHIESTE INDIPENDENTI: UN'UNITÀ \rightarrow C'È UNA UNITÀ DI BUS PER COMUNICARE

\rightarrow 2^n+1 SEMAFORI ($\forall U_i : BR \in BG_i, + BB \text{ per tutti}$)



PRIORITÀ MODIFICABILE

20. I/O



✓ dispositivo di I/O è UN'INTERFACCIA

che permette il comunicazione AL BUS DI SISTEMA

- PORTA: PERMETTE LA COMUNICAZIONE TRA DISPOSITIVO DI I/O E PROCESSORE (SISTEMICO)
- ↳ È UN REGISTRO

• CONNESSIONE /
 memory mapped I/O: strutturato LO STESSO BUS DELL'MEMORIA

↳ isolated I/O: separato BUS diversi → ABBONATE IN e OUT

→ I DIVERSI DISPOSITIVI HANNO VELOCITÀ DI OPERAZIONE DIVERSE; IL PROCESSORE DEVE SINCRONIZZARSI CON MECCANISMI:

① I/O PROGRAMMATO

② INTERRUPT

③ DMA (DIRECT MEMORY ACCESS)

①

GESTIONE DEMANDATA TOTALEMENTE DAL CPU: ESPERNA POLLING PER TESTARE IL REGISTRO DI STATO

→ POCO COSTOSO MA POCO EFFICIENTE → USATO IN SISTEMI PICCOLI

↳ I/O → R_{i(cpu)} → [MEM]

②

È UN SEGNALE ASINCRONO CHE IL DISPOSITIVO DI I/O INVIA DAL CPU CAVANDO DAL BUS DI DATI DI UN SERVIZIO

↳ NON FORZA TEMPO PER IL POLLING

- QUANDO ARRIVA UN SEGNALE, LA CPU INTERROMPE L'ISTR. CORRENTE E ESEGUE ISR (MANOVRA DI SERVIZIO DELL'INTERROGAZIONE)
- IC (CONTROLLER PER 'INTERRUPT'): GESTISCE TUTTE LE RICHIESTE DI INTERRUPT E PILOTA IL SEGNALE CHE VA AL PROCESSORE

→ ESSO CHE L'INTERROGATORE È UN ARRESTO DI ESECUZIONE → STAVI VEDI STACK IL PC SE PARTE DI STATO, PER poi ESEGUIRE

• POSSIBILITÀ DI GESTIRE DIVERSI DISPOSITIVI DI I/O: → SPESO PIÙ DI RISP. I > SCHEMI

UNICO
I/O CPU

PWI GESTE: • POLLING: 1 SOLO SEGNALE PER LE RICHIESTE DI INTERRUPT → CPU SCANNARE IL PERIFERICO DI STATO PER RILEVARE LA RICHIESTA → LATENZA RILEVATA

• INPUT VETTORIZZATO:

↳ PIÙ
URIBILO

• INTERRUPT ATTIVUOLIGUE: PROCESSORE PRONTO + GENERA IL RICHIESTA

• PERIFERIA METTE SUL BUS IL SUO CODICE IDENTIFICATIVO

• PROCESSORE USA IL CODICE COME CODICE INIZIALE DEL VETTORE CONTINENTE UN'INDRIZZO DI ESECUZIONE

↳ INT: INTERRUPT VECTOR TABLE

INTERRUPT

↓

↓

COD. ID

↓

INT

→ SPESO LE DIVERSE RICHIESTE POSSONO ESSERE ASSERVITE DALE PRATICAMENTE

• EQUAZIONI: VENGONO GESTITE DAL CPU COME RICHIESTE DI INTERRUPT DALE PERIFERICI

• EQUAZIONI DI I/O

• " " DEBUG

• " " DI PRATICAMENTO

• " " RICHIESTA

1. Un perifero attiva la richiesta di interrupt verso l'IC
2. L'IC attiva la richiesta di interrupt verso la CPU
3. La CPU completa l'esecuzione dell'istruzione corrente
4. La CPU invia il segnale Interrupt Acknowledgment
5. L'IC manda alla CPU il codice del perifero che ha fatto richiesta
6. La CPU invia al PC il registro di stato
7. La CPU accede alla INT e ne estrae l'indirizzo della ISR
8. Parte l'esecuzione della ISR
9. Alla fine della ISR
 - il registro di stato viene ripristinato
 - il programma interrotto riprende l'esecuzione

③ SI USA PER GRANDE MOLTI DI DATI, GESTITO DAL DMA CONTROLLER (DMA)

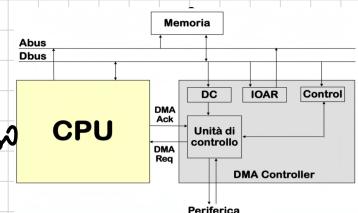
• DMA: INTER DI RICHIESTA E DI ACKNOWLEDGE, EGALITÀ DC E IOAR, GARantisce il controllo

↳ USO: PROGRAMMAZIONE (CPU carica DC e IOAR (immagine + n° WORDS da trasferire) e Usa)

• TIPO DI TRASFERIMENTO: A BLOCCHI, CYCLE STEALING, TRANSPORT DMA

↳ DIVISIBILE DATI IN BLOCCHI, CON SEGUENTI CAPTURETTI

↳ PUÒ GIUSTIFICARE RICHIESTA DI BUS



↳ TRASFERIMENTO SOLO GUARDO AL CPU SENZA

17. ARQUITETURA PIPELINE:

ARQUITETURA RISC necessaria modificare per aumentare l'efficienza

RISC: $\frac{1}{\text{clock}} \rightarrow 1 \text{ ISR} \rightarrow \text{CPI} \approx 1$
 Velco di clock, i processori / Superpipelined: $\frac{1}{\text{clock}} \rightarrow + \text{ ISR} \rightarrow \text{CPI} < 1$

CISC: $\text{CPI} > 1$

• CPI: CLOCK PER INSTRUCTION

• PIPELINE: SIEUE IN // OPERAZIONI DIVERSE SULLO STESSO FUSO DI DATI

• SE SI VERIFICA UNO STALL: VENNO PRODOTTI LE OPERAZIONI A RIUNTA

\rightarrow SI CHIAMA BUBBLE acc' INTRO DELL PIPELINE

\rightarrow CONCETTOLE ISTRUZIONI: UNICO (\rightarrow FETTO NON TUTTI HANNO ISR). \rightarrow SI CHIAMA CONCETTO PIP

\rightarrow NECESSITA CHE NON CI SIANO DELAYNTE TRA I DATI OMBRE (simboli) \rightarrow ATTIVITÀ DI CAVO SW o HW

\hookrightarrow HW: PERMETTE SPECIFICI REGOLI COME BUBBLING, SW: CONDIZIONI INTERNE ISR. NOP \rightarrow NO OPERATION

• SE ISTRUZIONI J POSSONO ESSERE DANNOSE: RISERVE + CASO DI CLOCK PER TUTTE LE NUOVE ISR.

\hookrightarrow RESERVE RECALL SLOT

HW: SVUOTO LA PIPELINE

\rightarrow SOLUZIONE:

'SW: ZERO RITORNATO \rightarrow INTRODUCE ISR. NOP OPPURE OTTIMIZZAZIONE

\hookrightarrow J ANTICIPATO

PROCESSORI RISC: GENERA SEMPRE, N RIDOTTO DI ISR., N FUORI DI REGISTRI, SENZA PIPELINE.

• 2 SOLO ISTRUZIONI: LOAD & STORE

, + ISR, - dimensione ISR. , CV costante

• POSSONO ISTRUZIONI FISSO \rightarrow OPERAZIONE DISCIPLINA PIÙ VERSATILE, CV ANCHE, FETCH OTTIMIZZATA

• PIÙ REGISTRI \rightarrow RETROAR. IL N DI ACCESSI IN PARALELO

SUPERPIPELINED:

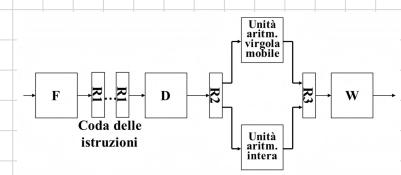
ESECUZIONE // DI UN'ISTRUZIONE O ANCHE SOVRAPPOSTE

\rightarrow GARANTIRE LO STESSO (che le operazioni) VENGANO ESEGUITE IN ORDINE

• ILP: INSTRUCTION LEVEL PARALLELISM

• TLP: THREAD LEVEL PARALLELISM \rightarrow INCREMENTARE NELLA PRESTAZIONE

F	D	O	W
F	D	O	W
F	D	O	W
F	D	O	W
F	D	O	W
F	D	O	W
F	D	O	W
F	D	O	W



$6 \cdot 2^{20}$

$512 \cdot 2^{20}$

$4 \cdot 2^{20}$

$512 \cdot 2^{20}$

$\frac{2}{1024 \cdot 4}$

$\frac{2}{512}$

• 12. LE MEMORIE CACHE

• PICCOLE DIMENSIONI, MA ELEVATA VELOCITÀ, TRA PROCESSORE E RAM, PRINCIPALE

→ SPORTA LA LOCALITÀ DEI RIFERIMENTI PER MIGLIORARE LE PRESTAZIONI:

✓ LOCALITÀ TEMPORALE: SE IL PROG. FA ACCESO AD UNA CELLA ALL'ISTANTE t , È MOLTO PROBABILE DI FARLO ANCHE ALL'ISTANTE $t + \Delta$

✓ LOCALITÀ SPAZIALE: SE A t ACCESO ALLA CELLA X , È MOLTO PROBABILE CHE A $t + \Delta$ IL PROG. FARÀ ACCESO A $X + e$

• TEMPO MEDIO DI ACCESO CPU: $T_{\text{medio}} = hC + (1-h)M$

$$\begin{cases} C: \\ M: \end{cases}$$

\hookrightarrow h: % RETRO NELLA CACHE
C: t successivo allo stesso
M: t successo in memoria se il dato
non è in cache

• STRUTTURA:

• AL SUO INTERNO CONTIENE UN CERTO NUMERO DI LINEE

✓ LINEA → 1 BLOCCO DI RIFERIMENTO + 1 TAG (INDICA IL BLOCCO DI MEMORIA PRESENTE)

• FUNZIONAMENTO:

Ogni volta che la CPU vuole accedere alla memoria, LA CARICA INIZIALMENTE L'INDIRIZZO:

VERIFICA SE IL BLOCCO È PRESENTE IN CACHE (cassa TAG) \checkmark HIT: ESTRAE IL PARCELLA E LA FORNISCE ALLA CPU

✓ SOLITAMENTE LA CASSA È POSTA TRA LA CPU E IL BUS MISS: CASSA IN CACHE L'INTERO BLOCCO DI CUI FA PARTE LA PARCELLA

✓ IN ALCUNI CASI SI DISTINGUE IL CIRCUITO PER I DATI E PER LE ISTRUZIONI \Rightarrow ARQUITETTURA MARVATTO

• DIMENSIONI:

POCHI KB → OVACOME MB \Rightarrow AL CRESCERE NEGLI ELEMENTI DIMINUISCONO LE PRESTAZIONI

• FUNZIONE DI TRADUZIONE (MAPPING):

→ POSSIBILE IN DIVERSE FORME NEGLI STESSI CIRCUITI È POSSIBILE UN BLOCCO DI MEMORIA

• MECCANISMI DI MAPPING: ① DIRECT MAPPING ② ASSOCIATIVE MAPPING ③ SET ASSOCIATIVE MAPPING

① ✓ BLOCCO I CORRISPONDE UNA LINEA K : $K = i \bmod N / N$: NUMERO DI LINEE DI CACHE

VANTAGGI: FACILEMENTE IMPLEMENTABILE IN HW,

Svantaggi: SE ACCESO A 2 BLOCCI CONDIVISI DA STessa LINEA → MISS

② ✓ BLOCCO → NON ESSERE RICORDATO IN UN DIVARIOASI BLOCCO DELLA MEMORIA

VANTAGGI: MASSIMA PLESSIVITÀ

Svantaggi: COMPLICATÀ HW DI RICERCA

③ N LINEE DI CACHE DIVISE IN S INSERIMENTI DI W LINEE Ogni LINEA

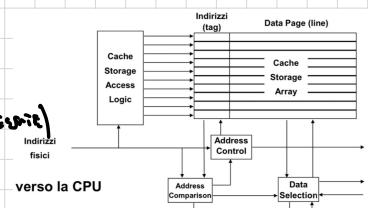
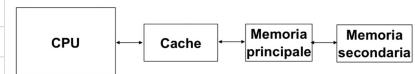
$K = i \bmod S$, i PUÒ ESSERE MESSO IN UNA DIVARIALE DELLE W LINEE

k-way set associative mapping
 $\# \text{SETS} = (\# \text{Entries}) / k$
 $\text{Offset} = \log_2(\text{Entries size})$
 $\text{Index} = \log_2(\# \text{SETS})$
 $\text{TAG} = \text{Address} - \text{Index} - \text{Offset}$

$K = VIE$

ACCESSI IN MEMORIA

BIT: ADDRESS - OFFSET



Summary of address mapping techniques

• Direct mapping
 $\# \text{Entries} = \text{Total memory} / \text{Entry size}$
 $\text{Offset} = \log_2(\text{Entries size})$

$$\text{Index} = \log_2(\# \text{Entries})$$

$$\text{TAG} = \text{Address} - \text{Index} - \text{Offset}$$

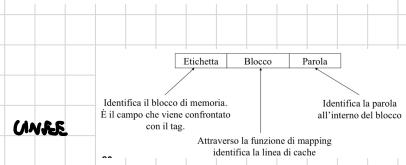
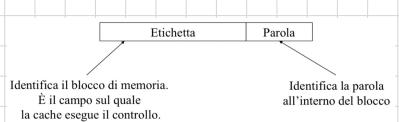
$$L_1, L_2, \dots, (m, n, m, n, m, n, \dots, m, n)$$

• Full associative mapping
 $\text{Offset} = \log_2(\text{Entries size})$

$$\text{TAG} = \text{Address} - \text{Offset}$$

$$\text{ENTRIES} = \text{LINEE}$$

$$K = VIE$$



• ALGORITMO DI RIMPIAZZAMENTO:

DEFINISCE COME UNA UNITÀ PER NECESSITARE DI ELIMINARE IL BLOCCO DI DATI IN CASO DI (2) O (3)

- LRU (LAST RECENTLY USED) / FIFO (IL PIÙ ECONOMICO) / LFU (LEAST FREQUENTLY USED) / RANDOM

• ALGORITMICO: MISURA, ADATTATORE:

• WRITE BACK: USO UN PUNT (DIRTY BIT) PER INDICARE SE UN BLOCCO È STATO MODIFICATO O MEM
 L₂ & DB = 1 → ORNO → DUNO IL BLOCCO E' ELIMINATO PUNT = 1 A VEDERE COPIATO IN MEMORIA
 AGGIORNARE LA MEMORIA? EMMA È' ACCESO IN CIRCUITO SE IL SUO DATO NON È AGGIORNATO

→ POCO EFFICIENTE

• WRITE THROUGH: A SCRIPIRE DIRETTA CPU → VENNE GUARDO SE IL DATO È STATO IN CACHE CUS IN MEMORIA

• DIMENSIONE DEI BLOCCI (4 - 32 BYTES).

↑ DIM. BLOCCO → ↑ MISURA, POI ↓ MISURA (DIMINUISCE IL NOME DI BLOCCI IN CACHE)

• BIT DI VARIABILITÀ: ✓ LINEA, STAMPALE SE IL VARIANTE DEL BLOCCO IN CACHE È GUARDO DA GUARDO IN MEMORIA

→ DISATTIVATO

• BUS WATCHING & WRITE-THROUGH: IL CONTROLLER DI CLASSEMENTO INTERNA ILLE WRITE-THROUGH SUL BUS E INVIA LA LINEA CORRESPONDENTI IN CACHE

• NON CACHABLE MEMORY

usato

usando

13. MEMORIA VIRTUALE

• PER ACESSO AL UNO SPAZIO MATERIALE DI QUELLO ESISTENTE, INDIREZI LOGICI UBICANO TRAVERSAMENTE IN MMU

• PROCESSORE ^{1. MMU} → ^{1. LOGICI} MEMORIA → DATO → TRASMISSIONE FATTA DA MMU (MEMORY MANAGEMENT UNIT)

• FUNZIONAMENTO MMU:

SE È PRESENTE L'INDIRIZZO IN MEMORIA PRINCIPALE → MMU PRODUCE L'INDIRIZZO FISICO CORRISPONDENTE

→ IN CASO CONTRARIO (PAGE DEFAULT) INTERNALE IC SO

→ LA MEMORIA È suddivisa IN BLOCCHI CHIAMATI PAGINE, ENTRO LA CPU ACCESO A UN DATO IN UNA PAGINA NON IN MEM. PRINCIPALE, VENNE TRASFORMATO L'INTERO PAGINA

• MAT: MEMORY ADDRESS TABLE → TRASFORMAZIONE DI INDIREZI LOGICI IN FISICI

→ LE PAGINE PIÙ ATTIVE SONO TUTTOVIA SERVITE DAL TLR → CAPO DELLA MAT

→ TRANSLATION LOOKAHEAD BUFFER

→ VANTAGGIO: RENDERE I PROGRAMMI INDEPENDENTI DAGLI CONFIGURAZIONI RESE DELLA MEMORIA

