INGEGNERIA PER LA SOSTENIBILITA' INDUSTRIALE - UnivPM

Fondamenti di Informatica

Prof. Ing. Loris Penserini, PhD

elpense@gmail.com

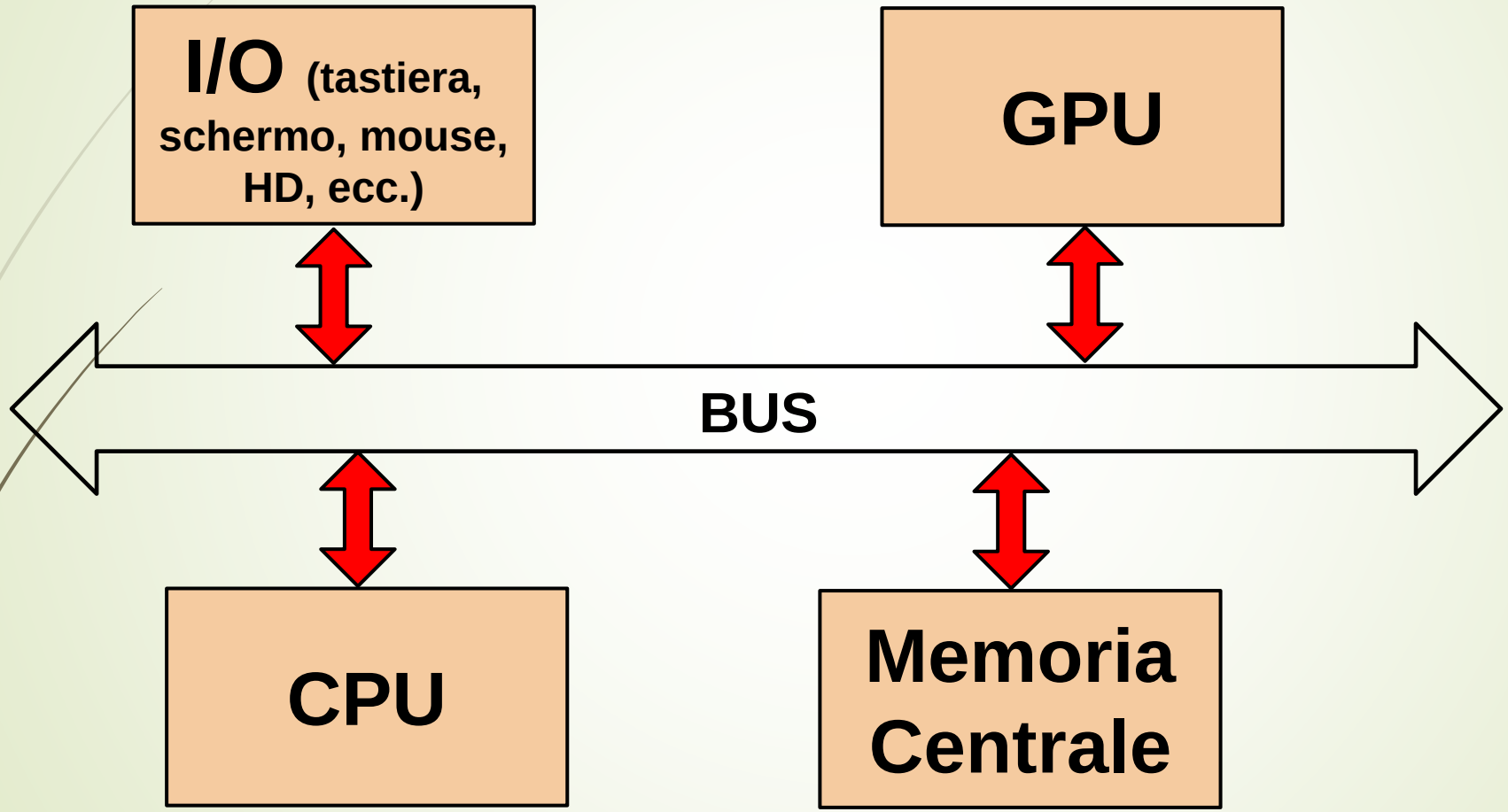
https://orcid.org/0009-0008-6157-0396

Materiale:

https://github.com/penserini/Lezioni_UnivPM.git

Eseguire un Algoritmo nel Sistema di Elaborazione (overview)

Dal punto di vista hardware



Macchina di Von Neumann

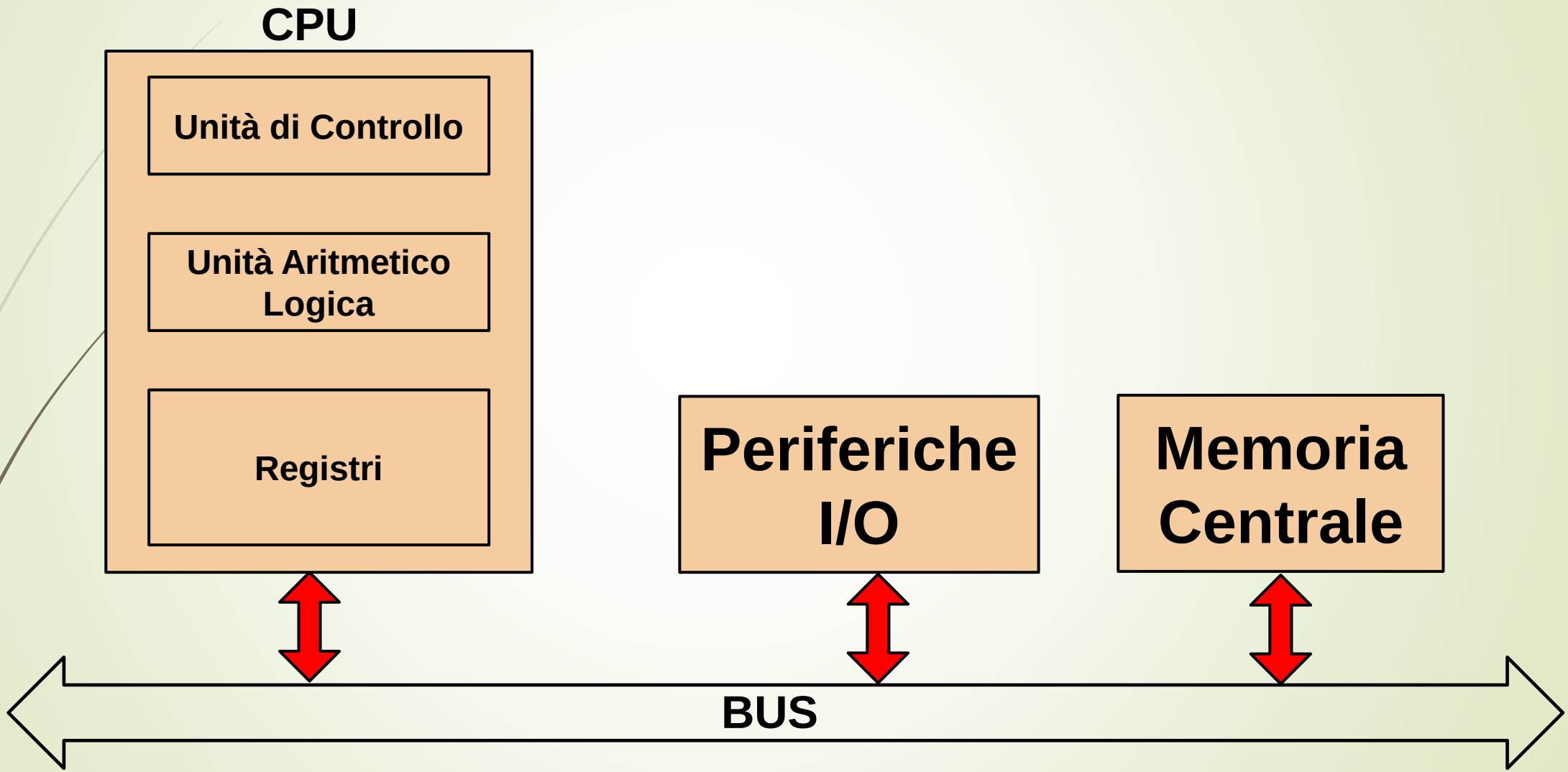
Tipi di BUS

BUS DATI: utilizzato per trasferire i dati tra RAM e CPU e tra CPU e Periferiche I/O

BUS INDIRIZZI: identifica la posizione delle celle di memoria in cui la CPU va a leggere/scrivere

BUS DI CONTROLLO: percorso da segnali di controllo che consentono di selezionare le unità coinvolte in un trasferimento dati (sorgente e destinazione), e quindi definire la direzione dello scambio (scrittura o lettura).

Il Processore



CPU

Unità di Controllo: legge le istruzioni dalla memoria e ne determina il tipo

Unità Aritmetico Logica (ALU): Esegue le operazioni necessarie per eseguire le istruzioni

Registri

- Sono memorie ad alta velocità usate per mantenere risultati temporanei
- Determinano il parallelismo (pipelining) della CPU
- Esistono vari tipi di registri con funzioni specifiche come ad esempio: Program Counter (PC), Instruction Register (IR), Program Status Word (PSW) che contiene info sullo stato di esecuzione del programma, MAR contiene l'indirizzo della locazione di memoria da leggere/scrivere, MDR necessario per scambiare informazioni tra memoria e CPU, ecc.

Frequenza di Clock: Velocità con cui la CPU scandisce le operazioni. Si misura in Hertz (Hz) che significa «volte al secondo»: 2Hz => 2 volte al secondo

Periodo di clock: intervallo di tempo tra un fronte e l'altro della sequenza di impulsi (inverso della frequenza di clock)

Il periodo di clock deve essere sufficientemente lungo da consentire a tutti i segnali elettrici di arrivare.

Esecuzione delle Istruzioni

L'esecuzione di una qualsiasi istruzione segue una sequenza di azioni che tecnicamente prende il nome di ***«Ciclo di Fetch-Decode-Execute»*** come segue:

- **FETCH**: prende l'istruzione corrente dalla memoria e la inserisce nel registro delle istruzioni IR
- Incrementa il program counter (PC) in modo che contenga l'indirizzo dell'istruzione successiva
- **DECODE**: Determina il tipo di istruzione corrente
- Se l'istruzione usa una parola in memoria determina dove si trova
- Se necessario, carica la parola in un registro della CPU
- **EXECUTE**: esegue l'istruzione
- Ricomincia dal nuovo valore del PC

La Macchina di Turing (overview)

La Tesi di Church-Turing

L'uso della Macchina di Turing (MdT) introduce in modo intuitivo l'idea che un problema sia risolubile attraverso un **procedimento** quando sia possibile costruire una MdT in grado di eseguire un algoritmo che descrive, attraverso passi elementari, il **procedimento** risolutivo.

Per cui, il concetto di algoritmo è il procedimento risolto dalla MdT astratta.

La Tesi di Church-Turing afferma che: La classe delle funzioni computabili, secondo il concetto intuitivo di algoritmo, coincide con la classe delle funzioni Turing-computabili, cioè computabili con una MdT.

Sulla base della Tesi di Church-Turing possiamo quindi individuare all'interno dell'insieme di tutti i possibili problemi il sottoinsieme dei problemi che sono computabili, cioè quelli ai quali può essere associata una MdT che risolve il problema. Inoltre l'algoritmo risolutivo di un problema computabile può essere identificato con la sequenza delle azioni che vengono eseguite da una MdT.

Automa

Abbiamo precedentemente affermato che un «esecutore» o «elaboratore elettronico» deve eseguire in modo **automatico** l'algoritmo.

Per cui definiamo con il termine «**automa**» un dispositivo in grado di eseguire da solo, cioè in modo automatico, senza l'intervento di una persona, una sequenza di azioni stabilite in precedenza. Inoltre, l'automa è in grado di acquisire informazioni dall'esterno e produrre informazioni verso l'esterno e, infine, al suo interno può assumere stati diversi.

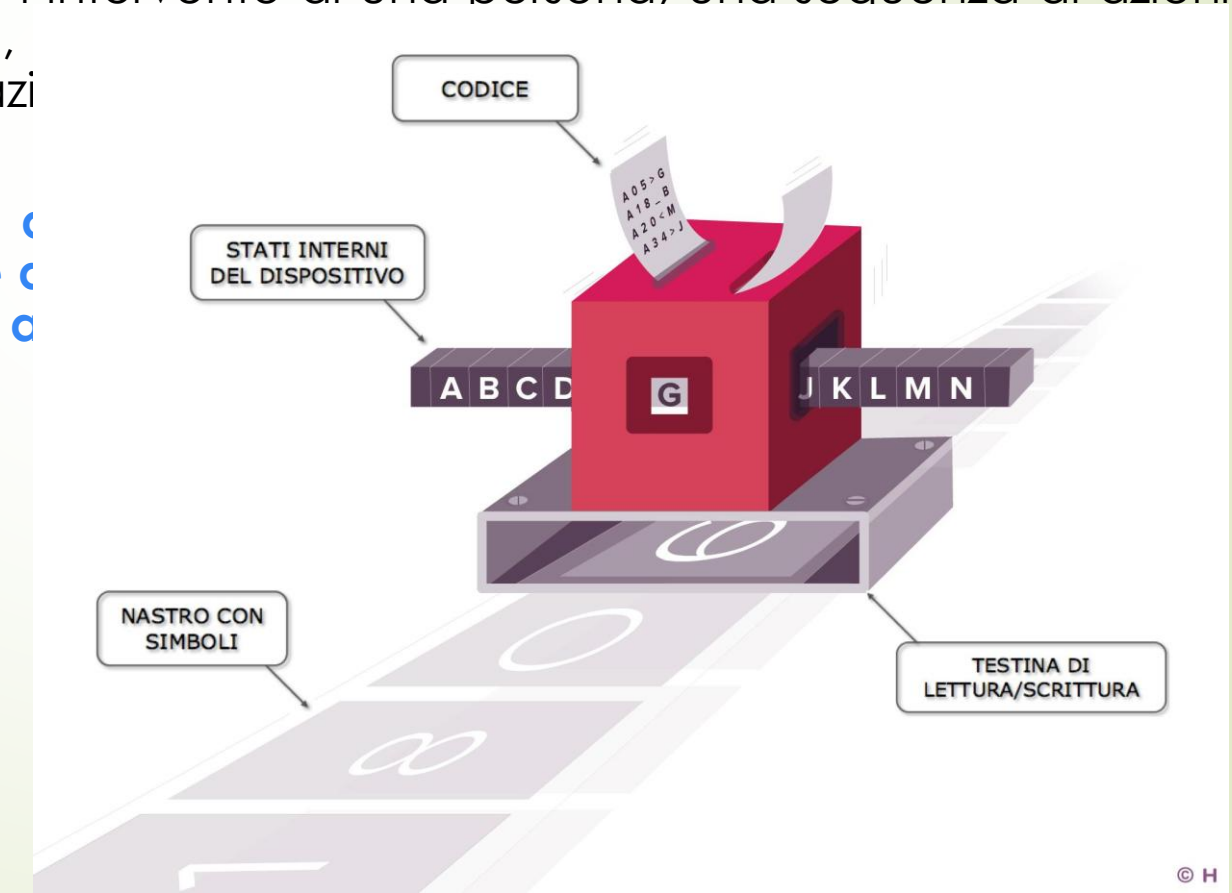
La MdT è un automa costituita da un meccanismo di controllo, da un nastro di lunghezza infinita nei due sensi e da una testina di lettura/scrittura che può effettuare due tipi di movimento a sinistra o a destra.

Automa

Abbiamo precedentemente affermato che un «esecutore» o «elaboratore elettronico» deve eseguire in modo **automatico** l'algoritmo.

Per cui definiamo con il termine «**automa**» un dispositivo in grado di eseguire da solo, cioè in modo automatico, senza l'intervento di una persona, una sequenza di azioni stabilite in precedenza. Inoltre, dall'esterno e produrre informazi assumere stati diversi.

La MdT è un automa costituita lunghezza infinita nei due sensi e due tipi di movimento a sinistra o



Formalizzazione della MdT

MdT = (A, I, S, s_0, F, d)

A: alfabeto dei simboli utilizzati dalla MdT (es. «blank» rappresenta la cella vuota)

I: l'insieme dei simboli di input che possono essere letti dal nastro

S: l'insieme degli stati della macchina

s_0: lo stato iniziale della macchina

F: l'insieme degli stati finali della macchina (sottoinsieme di S → S ⊃ F)

d: funzione di transizione che associa ad una coppia (simbolo-input, stato) una terna (stato, simbolo-output, movimento-testina):

$$(i_t, s_{t-1}) \rightarrow (s_t, u_t, m_t)$$

s_t : stato successivo; u_t : simbolo scritto sul nastro; m_t : movimento testina

Viene prodotta una «**matrice di transizione**» che schematizza con una tabella la «**funzione di transizione**».

Formalizzazione della MdT

Matrice delle transizioni.

Stati / Input	s_1	s_2	s_3
i_1	s_{11},u_{11},m_{11}	s_{12},u_{12},m_{12}	s_{13},u_{13},m_{13}
i_2	s_{21},u_{21},m_{21}	s_{22},u_{22},m_{22}	s_{23},u_{23},m_{23}
i_3	s_{31},u_{31},m_{31}	s_{32},u_{32},m_{32}	s_{33},u_{33},m_{33}
i_4	s_{41},u_{41},m_{41}	s_{42},u_{42},m_{42}	s_{43},u_{43},m_{43}

In ogni istante la MdT assume una configurazione esprimibile con la quintupla:

{simbolo letto, stato, nuovo stato, simbolo scritto, movimento della testina}

E' chiamata «**mossa**» della MdT il passaggio da una configurazione all'altra.

Esempio di utilizzo della MdT

Problema: calcolare il quoziente e il resto della divisione di un numero per 2

Definizione formale della MdT per risolvere il «problema»

A = (blank, #, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) , con # situazione di fine input

I = (#, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

S = (R0, R1, FINE) con R0: lo stato iniziale e FINE: lo stato finale

F = (FINE)

All'inizio, sul nastro si trovano scritte le cifre che compongono il dividendo. Alla fine dell'elaborazione, si trovano le cifre del quoziente e, immediatamente a destra, quella del resto (che sarà 0 oppure 1). La MdT opera la divisione partendo dalle cifre più significative del dividendo (da sinistra a destra) e scrivendo al posto di ogni cifra il quoziente della cifra diviso 2, tenendo conto dell'eventuale resto precedente.

R0 -> stato con resto precedente uguale a zero

R1 -> stato con resto precedente uguale a uno

Funzione delle Transizioni della MdT

La **funzione di transizione** opera in questo modo:

(R0, cifra pari) -> (R0, cifra pari / 2, Destra)

(R1, cifra pari) -> (R0, (cifra pari + 10) / 2, Destra)

(R0, cifra dispari) -> (R1, cifra pari : 2, Destra)

(R1, cifra dispari) -> (R1, (cifra pari + 10) / 2, Destra)

(R1, #) -> (FINE, 1, Destra)

(R0, #) -> (FINE, 0, Destra)

	R0	R1
#	FINE, 0, D	FINE, 1, D
0	R0, 0, D	R0, 5, D
1	R1, 0, D	R1, 5, D
2	R0, 1, D	R0, 6, D
3	R1, 1, D	R1, 6, D
4	R0, 2, D	R0, 7, D
5	R1, 2, D	R1, 7, D
6	R0, 3, D	R0, 8, D
7	R1, 3, D	R1, 8, D
8	R0, 4, D	R0, 9, D
9	R1, 4, D	R1, 9, D

Spiegazione della Tabella delle Transizioni

Riga # (fine input)
- In **R0**: (FINE, 0, D) → significa che la macchina si ferma scrivendo **0** (perché il resto finale è 0).
- In **R1**: (FINE, 1, D) → la macchina si ferma scrivendo (resto finale 1)

Qui viene scritto il resto della divisione.

Riga 0
- In **R0**: (R0, 0, D) → dividere 0 con resto 0 dà quoziente 0, resto nuovo 0.
- In **R1**: (R0, 5, D) → se avevo resto 1, considero "10+0 = 10"; diviso 2 fa quoziente 5, nuovo resto 0.

Riga 1
- In **R0**: (R1, 0, D) → 1 ÷ 2 dà quoziente 0, resto 1 → passo in stato R1.
- In **R1**: (R1, 5, D) → (10+1=11)/2 = 5 con resto 1 → rimango in R1, scrivo 5.

Riga 2
- In **R0**: (R0, 1, D) → 2 ÷ 2 = 1, resto 0 → rimango in R0.
- In **R1**: (R0, 6, D) → (10+2=12)/2 = 6, resto 0 → passo in R0.

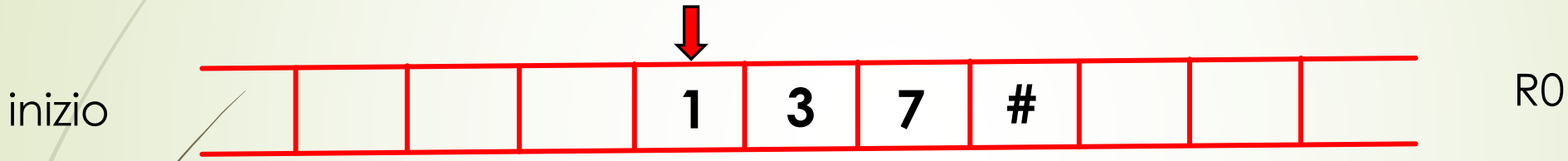
Ecc.

	R0	R1
#	FINE, 0, D	FINE, 1, D
0	R0, 0, D	R0, 5, D
1	R1, 0, D	R1, 5, D
2	R0, 1, D	R0, 6, D
3	R1, 1, D	R1, 6, D
4	R0, 2, D	R0, 7, D
5	R1, 2, D	R1, 7, D
6	R0, 3, D	R0, 8, D
7	R1, 3, D	R1, 8, D
8	R0, 4, D	R0, 9, D
9	R1, 4, D	R1, 9, D

Project Work

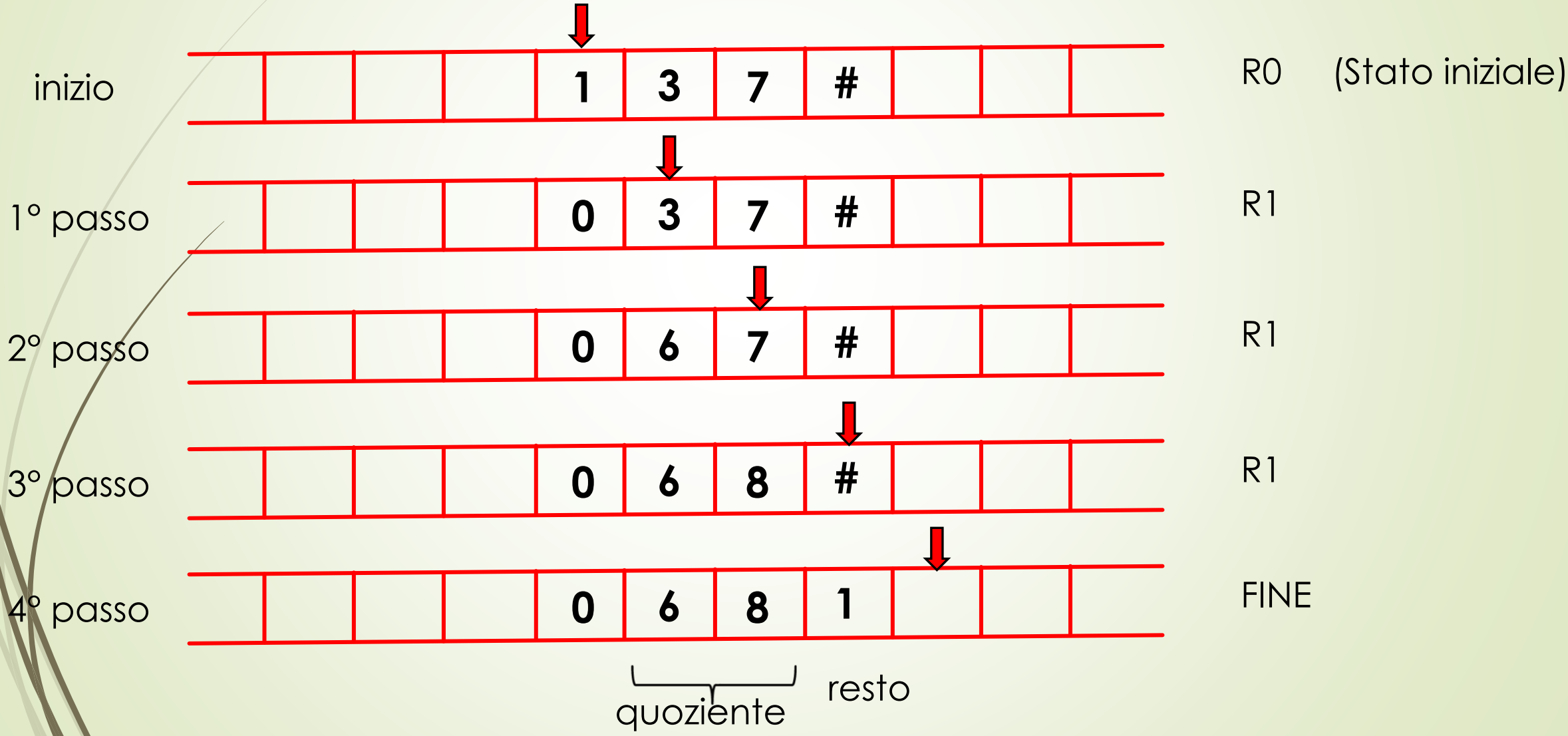
Problema: calcolare il quoziente e il resto della divisione per 2 del numero 137

Scandire le operazioni necessarie per la MdT definita precedentemente per questo tipo di problemi. Scrivere l'esecuzione dettagliata.



Project Work - Soluzione

Problema: calcolare il quoziente e il resto della divisione per 2 del numero 137



Project Work - Soluzione

ESECUZIONE DETTAGLIATA

Passo 1 — Stato **R0**, sotto la testina la MdT legge: 1

- Calcolo: [1/2]=0, nuovo resto =1 ⇒ nuovo stato **R1**
- Azione: scrivi 0, muovi a destra
- Nastro: [0][3][7][#], stato **R1**, testina su 3

Passo 2 — Stato **R1**, sotto la testina la MdT legge: 3

- Calcolo con “riporto 10”: [(10+3)/2]=[13/2]=6, nuovo resto =13 mod 2=1 ⇒ rimane in **R1**
- Azione: scrivi 6, muovi a destra
- Nastro: [0][6][7][#], stato R1, testina su 7

Passo 3 — Stato **R1**, sotto la testina la MdT legge: 7

- Calcolo: [(10+7)/2[=[17/2]=8, nuovo resto =1 ⇒ rimane in **R1**
- Azione: scrivi 8, muovi a destra
- Nastro: [0][6][8][#], stato R1, testina su #

Passo 4 — Stato **R1**, sotto testina: # (fine input)

- Azione: scrivi il resto corrente (1) e vai nello stato di **FINE**
- **Nastro finale: [0][6][8][1] (HALT)**

Project Work

Problema: **formalizzare una MdT che calcoli il MCD**

Ricordiamo l'approccio risolutivo di Euclide per due numeri **a** e **b** (**a ≥ b > 0**):

- Calcola il resto **r = a mod b**
- Se r = 0, allora MCD(a, b) = b (FINE)
- Altrimenti: sostituisci a = b, b = r, e **ripeti**

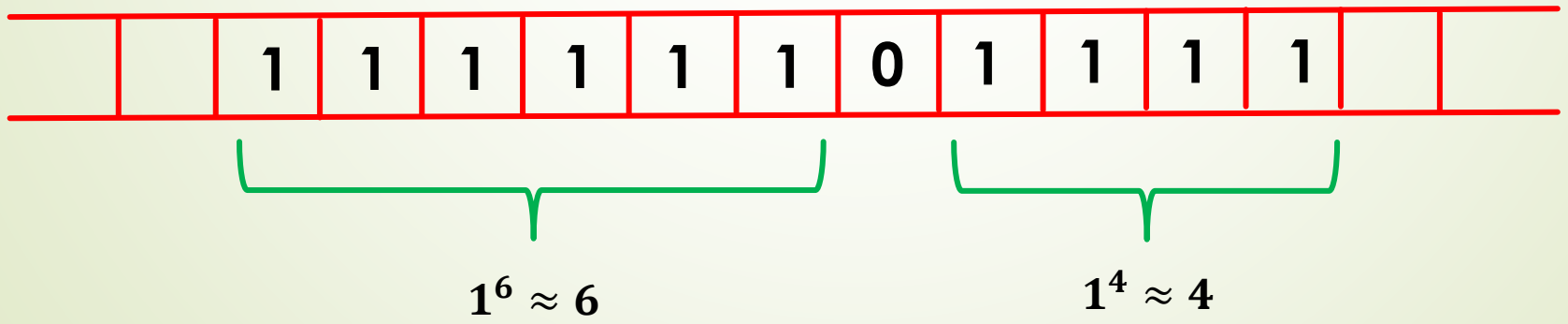
Project Work - Soluzione

Idea di rappresentazione sulla MdT

Dobbiamo decidere come codificare i numeri sul **nastro**:

- Adottiamo un sistema «**unario**» (più semplice per una MdT).
  Es: il numero 4 è rappresentato come 1111 (una sequenza di 4 simboli).
- Separiamo i due numeri con un simbolo speciale, ad esempio 0.
  Es: per **(a, b) = (6, 4)** scriviamo **11111101111** sul nastro.

Alla fine, sul nastro rimarrà il **MCD in unario** (una sequenza di 1).



Project Work - Soluzione

Stati e funzionamento generale

La MdT deve implementare la **divisione con resto**:

- **Inizio**: nastro con **a0b** (dove a e b sono sequenze di 1, e 0 il separatore).
- **Divisione di a per b**:
  - Sottrai iterativamente blocchi di **b** da **a** (cioè "cancelli" tanti 1 in **a**, da sinistra a destra, quanti 1 ci sono in **b**).
  - Se riesci a sottrarre esattamente, il resto è 0. Cioè l'algoritmo termina.
  - Se non riesci a sottrarre completamente, quello che rimane di **a** è il **resto r**.
- **Aggiornamento**: riscrivi sul nastro **b0r** (cioè nuovo dividendo = vecchio divisore, nuovo divisore = resto).
- **Itera** finché il resto non è nullo.
- **Uscita**: il numero rimasto è il MCD.

Project Work - Soluzione

Definizione formale minimale/semplificata della MdT
La MdT è una funzione:

$$\mathbf{M=(Q,\Sigma,\Gamma,\delta,q_0,q_{acc})}$$

- **Alfabeto di input:** $\Sigma=\{1,0\}$, Codifica: $\mathbf{1}^a\ \mathbf{0}\ \mathbf{1}^b\ con\ a \geq b > 0$
- **Alfabeto del nastro:** $\Gamma=\{1,0,\_,X,Y\}$, dove _ è il blank; **X**, **Y** marcatori temporanei
- **Stati** $Q=\{q_0,q_{check},q_{sub},q_{rest},q_{update},q_{acc}\}$
  - q_0: stato iniziale; porta la testina al separatore **0**
  - q_{check}: verifica se **b=0** (caso base); se dopo 0 c'è _ (cioè b=0) ⇒ q_{acc}; altrimenti avvia una divisione (ripeti sottrazione)
  - q_{sub}: esegue sottrazione iterativa di **b** da **a**
  - q_{rest}: se non è stato possibile sottrarre completamente, il contenuto di **a** è il **resto**; ripristina i marcatori in **b**
  - q_{update}: riscrive configurazione **b0r** (**a=b e b=r**) e riposiziona la testina al separatore **0** per un nuovo ciclo ⇒ torna a q_{check}
  - q_{acc}: stato finale di accettazione (halt), il nastro contiene $\mathbf{1}^{\mathbf{MDC(a,b)}}$, cioè il MCD
- **Funzione di transizione** δ: implementa i passi di cancellazione e ricopiatura
- **Stato accettazione** q_{halt}

Project Work - Soluzione

La divisione con resto segue queste fasi:

- **q_{sub}**: esegue **divisione con resto** r=a **mod** b tramite sottrazioni ripetute:
  - abbina ogni **1** di **b** con un **1** di **a** marcando Y (in **b**) e X (in **a**);
  - se in un giro tutti gli **1** di **b** sono serviti ⇒ cancella gli X (cioè **a = a−b**) e ripeti il giro;
  - se **a** finisce prima ⇒ vai a **q_{rest}** (giro fallito, resto trovato, ripristino marcatori).

Alcune invarianti:

- All'ingresso in q_{check} il nastro ha sempre la forma **1^a 0 1^b** e la testina è pronta a verificare b
- Se **b=0** ⇒ termina in q_{acc} lasciando **1^a**
- Altrimenti: **b ≠ 0** ⇒ **r = a mod b** viene calcolato in q_{sub}; poi q_{update} prepara il prossimo ciclo con la coppia **(a,b) = (b,r)**
- L'algoritmo di Euclide garantisce che dopo un numero finito di cicli **b** diventa **0** ⇒ la MdT entra in q_{acc}

Project Work – Soluzione applicata

Consideriamo l'esempio con **a=6 e b=4**:
Codifica sul nastro: **1^a 0 1^b ⇒ 111111 0 1111**
(Internamente la MdT usa marcatori temporanei X,Y mentre lavora, ma qui mostro solo gli "scatti" significativi senza i marcatori.)

Passo 0 — Inizio

- **Stato**: $q_0 \rightarrow q_{check}$
- **Nastro**: 111111 0 1111
- **Azione**: controlla **b**. Dopo **0** c'è **1** ⇒ **b≠0** ⇒ vai a q_{sub}

Passo 2 — 2° giro (fallito) ⇒ resto trovato

- **Stato**: $q_{sub} \Rightarrow q_{rest}$
- **Motivo**: ora **a=2** < **b=4**, quindi non posso sottrarre ancora: **resto r=2**
- **Nastro**: (immutato) 11 0 1111
- **Azione**: ripristina le eventuali marcature e vai a q_{update}.

Project Work – Soluzione applicata

Passo 3 — Aggiornamento (swap)

- **Stato**: q_update
- **Azione**: riscrivi **b 0 r ⇒ 1111 0 11** (cioè a=4, b=2)
- **Vai** a q_{check}

Passo 4 — Controllo

- **Stato**: q_{check}
- Dopo **0** c'è **1** ⇒ **b≠0 ⇒** continua in q_{sub}

Passo 5 — Nuovo ciclo, 1° giro (riuscito)

- **Stato**: q_{sub}
- **Effetto**: a = 4−2 = 2
- **Nastro**: **11 0 11**

Passo 6 — Nuovo ciclo, 2° giro (riuscito)

- **Stato**: q_{sub}
- **Effetto**: a = 2−2 = 0
- **Nastro**: **0 11**

Project Work – Soluzione applicata

Passo 7 — Tentativo successivo (fallisce subito) ⇒ r = 0

- **Stato**: $q_{sub} \Rightarrow q_{rest} \Rightarrow q_{update}$
- **Resto**: r = 0
- **Swap**: scrivi **b 0 r ⇒ 11 0** (ora a=2, b=0).
- **Vai** a q_{check}

Passo 8 — Arresto

- **Stato**: q_{check}
- Dopo **0** c'è blank ⇒ **b = 0** ⇒ **q_{acc} (HALT)**.
- **Nastro finale**: **11**

Risultato

Il nastro contiene 11 ⇒ **MCD(6,4) = 2**.

Spunti Bibliografici dell'Autore

[Morandini et al., 2017] Mirko Morandini, Loris Penserini, Anna Perini, Alessandro Marchetto: Engineering requirements for adaptive systems. Requirements Engineering Journal, 22(1): 77-103 (2017)

[Morandini et al., 2009] Morandini M., Penserini L., and Perini A. (2009b). Operational Semantics of Goal Models in Adaptive Agents. In 8th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'09). IFAAMAS.

[Morandini et al., 2008] Morandini, M., Penserini, L., and Perini, A. (2008b). Automated mapping from goal models to self-adaptive systems. In Demo session at the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), pages 485–486.

[Aldewereld et al., 2008] Huib Aldewereld, Frank Dignum, Loris Penserini, Virginia Dignum: Norm Dynamics in Adaptive Organisations. NORMAS 2008: 1-15

[Penserini et al., 2007a] Loris Penserini, Anna Perini, Angelo Susi, John Mylopoulos: High variability design for software agents: Extending Tropos. ACM Trans. Auton. Adapt. Syst. 2(4): 16 (2007)

[Penserini et al., 2007b] Loris Penserini, Anna Perini, Angelo Susi, Mirko Morandini, John Mylopoulos: A design framework for generating BDI-agents from goal models. AAMAS 2007: 149

[Pagliarecci et al., 2007] Francesco Pagliarecci, Loris Penserini, Luca Spalazzi: From a Goal-Oriented Methodology to a BDI Agent Language: The Case of Tropos and Alan. OTM Workshops (1) 2007: 105-114

[Penserini et al., 2006a] Loris Penserini, Anna Perini, Angelo Susi, John Mylopoulos: From Stakeholder Intentions to Software Agent Implementations. CAiSE 2006: 465-479

[Penserini et al., 2006b] Loris Penserini, Anna Perini, Angelo Susi, John Mylopoulos: From Capability Specifications to Code for Multi-Agent Software. ASE 2006: 253-256

[Panti et al., 2003] Maurizio Panti, Loris Penserini, Luca Spalazzi: A critical discussion about an agent platform based on FIPA specification. SEBD 2000: 345-356

[Panti et al., 2001] Maurizio Panti, Luca Spalazzi, Loris Penserini: A Distributed Case-Based Query Rewriting. IJCAI 2001: 1005-1010

GRAZIE!