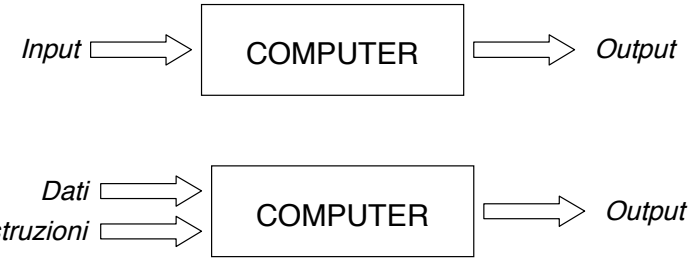
INFORMATICA E COMPUTER : INTRODUZIONE

- **Informatica**: dal francese, *informatique* – *informat*(ion) (automat)*ique* [termine coniato dall'ingegnere francese Philippe Dreyfus nel 1962] è la scienza che studia i metodi e le tecniche per il trattamento (acquisizione, memorizzazione, processamento, trasmissione) automatico delle informazioni tramite **computer**.

- **Computer**, o **calcolatore elettronico**: è una macchina che produce dati in uscita (output) sulla base delle informazioni che riceve in ingresso (input). Per un computer, sono input tanto i dati sui quail deve lavorare, quanto le istruzioni che deve eseguire per trasformare i dati in output



IL CONCETTO DI ALGORITMO

- Il calcolatore elettronico per risolvere un problema utilizza un algoritmo, cioè un insieme di azioni (o istruzioni) che, eseguite secondo un ordine prestabilito, permettono di trovare il risultato cercato sulla base dei dati in ingresso

- Il concetto di algoritmo è uno dei concetti di base dell'intera matematica: i più semplici ed antichi algoritmi sono le regole per eseguire le operazioni dell'aritmetica elementare, formulate dal matematico arabo medioevale *Al-Khuwarizmi*, da cui deriva appunto il nome di algoritmo.

- Un computer è un rapidissimo esecutore di sequenze di istruzioni (gli algoritmi)

ESECUZIONE DI UN ALGORITMO SU COMPUTER

algoritmo — *Procedura* di trasformazione di un insieme di *dati iniziali* in un insieme di *risultati finali* mediante una sequenza di istruzioni.

linguaggio di programmazione — *Linguaggio* (insieme di simboli e regole) *per rappresentare* le istruzioni di un algoritmo e la loro *concatenazione*

Programma — *algoritmo scritto in un linguaggio di programmazione al fine di comunicare al calcolatore elettronico le azioni da eseguire*

Processo — programma in *esecuzione* sul computer

REQUISITI DI UN ALGORITMO PER IL CALCOLATORE

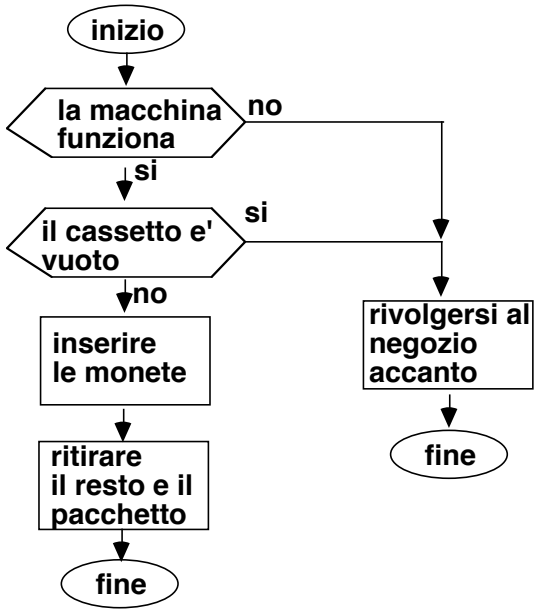
- ❏ istruzioni non ambigue
- ❏ istruzioni eseguibili in un tempo finito
- ❏ terminazione dell'esecuzione dopo un numero finito di passi

Descrizione intuitiva di un algoritmo

un formalismo grafico: i *diagrammi a blocchi* (*flow chart*).

- rappresentano la sequenza di istruzioni con linee di connessione orientate
- simboli grafici diversi per rappresentare tipi diversi di istruzioni
  - un esagono (o un rombo) : un *blocco di controllo* (condizione o *test*)**;**
  - un rettangolo : un *blocco azione* (cioè un'istruzione imperativa);
  - il *blocco iniziale* e il *blocco finale* sono rappresentati con ellissi.

DESCRIZIONE CON DIAGRAMMA A BLOCCHI



- riduzione di ambiguità/incertezze tramite l'introduzione di test
- i test *decidono* quale è la sequenza delle istruzioni successive
- i test provocano una *bi-dimensionalità*
- problema delle azioni ripetute

CALCOLATORE ELETTRONICO COME ESECUTORE

- Il calcolatore elettronico è un esecutore di algoritmi : un algoritmo per il calcolatore elettronico deve essere scritto tenendo conto delle istruzioni che il calcolatore elettronico è capace di eseguire

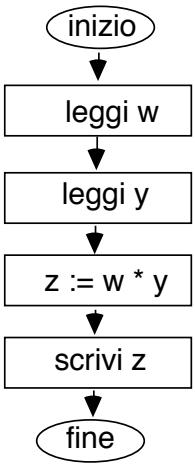
Ipotesi sull'esecutore:

1. comprende un linguaggio simbolico
2. è in grado acquisire numeri interi in ingresso e produrre numeri interi in uscita
3. è in grado di eseguire soltanto somme e sottrazioni tra due operandi e di verificare se un operando è uguale a 0

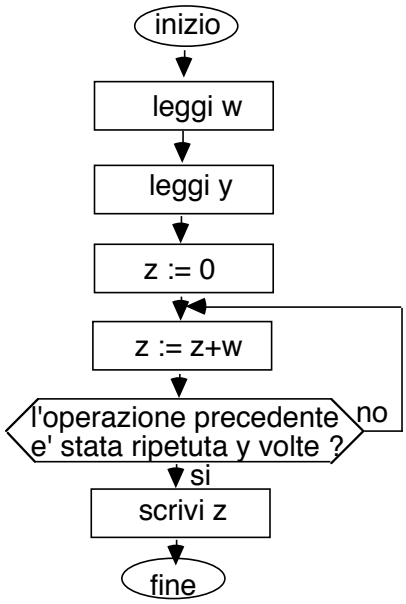
Problema: scrivere un algoritmo per l'esecuzione della **moltiplicazione** di due numeri interi

Schemi a blocchi (non corretti) dell'algoritmo della moltiplicazione

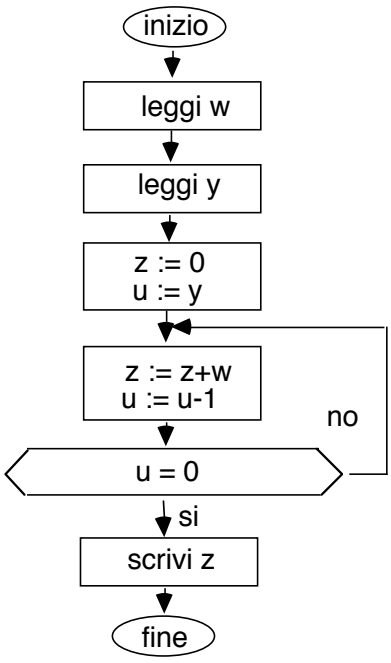
condizioni 1 e 2



condizioni 2 e 3 (ma non 1)

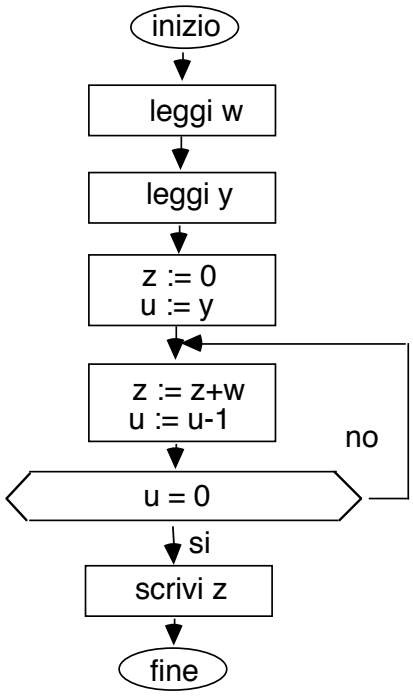


Schema a blocchi corretto dell'algoritmo della moltiplicazione



- soddisfa tutte le condizioni enunciate
- opera su una qualsiasi coppia di numeri interi (Y strettamente positivo)
- i valori acquisiti dall'esterno sono memorizzati nelle *variabili* w e Y; il risultato è memorizzato in Z
- 0 e 1 sono *costanti*
- la variabile U è utilizzata come contatore del numero di iterazioni
- l'operazione di *assegnamento* calcola l'espressione a destra del simbolo **:=** e ne assegna il valore alla variabile a sinistra
- l'istruzione <leggi> è un particolare tipo di assegnamento che preleva valori dall'*unità* di ingresso

Algoritmo della moltiplicazione : rappresentazione con linguaggio di alto livello



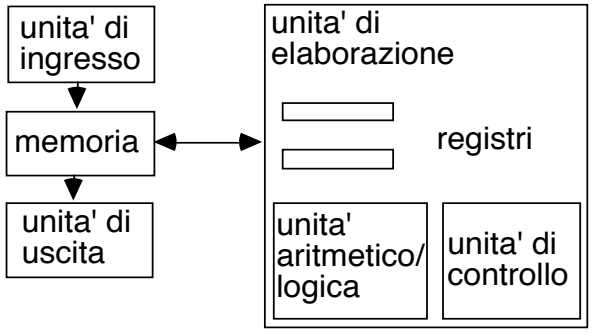
- si introducono strutture per poter esprimere la *ripetizione* delle istruzioni

```
inizio
  leggi W
  leggi Y
  Z := 0
  U := Y
  ripeti
       Z := Z+W
       U := U-1
  fino a quando U <> 0
  scrivi Z
fine
```

- ripeti ….. fino a quando U<>0 è una istruzione di *ripetizione condizionata*

REQUISITI DELL'ESECUTORE

- disponibilità dei valori delle *variabili* in ogni momento dell'esecuzione: unità di memoria
- capacità di elaborazione per somme, sottrazioni, confronti con 0: unità aritmetico/logica
- capacità di colloquiare con l'esterno: unità di ingresso (input) ed uscita (output)
- comunicazione tra le varie unità



Macchina di Von Neumann

UNITÀ DI ELABORAZIONE (CENTRAL PROCESSING UNIT – CPU)

- **registri**
  - dispositivi elettronici capaci di memorizzare pochi byte
    - rispetto alla memoria: molto *più piccoli* ma molto *più veloci*

- **unità di controllo**
  - gestisce la sequenzializzazione delle operazioni
  - gestione del **clock**: segnale di sincronizzazione per tutto il sistema
    - si misura in cicli/secondo [Hz]
    - Esempio: 2 GHz = 2 x 10^9 Hz: un ciclo è eseguito in 0.5 x 10^{-9} = 0.5ns

- **unità aritmetico/logica** (Arithmetic Logic Unit – **ALU**)
  - esegue operazioni aritmetiche e logiche

LA MEMORIA DELL'ELABORATORE

- è suddivisa in *celle*, di dimensione fissa, individuabili tramite un *indirizzo*
- è una struttura *monodimensionale*
- ogni dato occupa un numero intero di celle

Indirizzo	*Celle*
0	DAT0
1	DAT0
2	
N	

- operazioni:
  - **lettura** (non distruttiva) del contenuto di una cella
  - **scrittura** (distruttiva) in una cella

MEMORIZZAZIONE SU CALCOLATORE

- Il trattamento automatico delle informazioni sul calcolatore elettronico avviene in formato binario, cioè l'informazione è rappresentata da un insieme di bit. Un bit rappresenta l'elemento minimo d'informazione che può essere trattata da un calcolatore elettronico (così come un carattere è l'elemento minimo d'informazione che può essere trattato nel testo scritto).
- L'unità atomica è il **bit** (**bi**nary digi**t**) → valori <0> e <1> (dispositivi *binari*)
- Una sequenza di 8 bit è detto **byte**
- Altre forme di memorizzazione: Word (16 bit), Double-word (32 bit), Quad-word (64 bit)
- **bit**: Numero di configurazioni: 2 →intervallo di variabilità: [0-1]
- **byte**: Numero di configurazioni: 256 (2^8)
  →intervallo di variabilità: (dipende dal tipo di memorizzazione)

  Modulo: 256 configurazioni, [0, 255]

  Modulo e segno: 256 configurazioni, [-127, +127]

  Complemento a 2: 256 configurazioni, [-128, +127]

Esempio: codifica dei numeri da 0 a 31(tipo di memorizzazione: modulo)

0	**00000**	8	**01000**	16	**10000**	24	**11000**
1	**00001**	9	**01001**	17	**10001**	25	**11001**
2	**00010**	10	**01010**	18	**10010**	26	**11010**
3	**00011**	11	**01011**	19	**10011**	27	**11011**
4	**00100**	12	**01100**	20	**10100**	28	**11100**
5	**00101**	13	**01101**	21	**10101**	29	**11101**
6	**00110**	14	**01110**	22	**10110**	30	**11110**
7	**00111**	15	**01111**	23	**10111**	31	**11111**

5 BIT → 2^5 = 32 VALORI DISTINTI

LA MEMORIA DELL'ELABORATORE

- anche l'indirizzo è un numero codificato in binario
- una cella corrisponde ad un byte

Indirizzo	*Celle*
00000	DAT0
00001	DAT0
00010	
11111	

- Capacità della memoria:
  - Con un indirizzo lungo 10 bit ho 2^{10} = 1024 byte → 1 **Kilo** byte
  - Con un indirizzo lungo 20 bit ho 2^{20} = 1048576 byte → 1 **Mega** byte
  - Con un indirizzo lungo 30 bit ho 2^{30} = 1073741824 byte → 1 **Giga** byte

CODIFICA DEL PROGRAMMA ALL'INTERNO DEL CALCOLATORE

Programma cablato

l'unità di esecuzione è strutturata per eseguire un solo compito specifico.
Es: controllo di un ascensore

Programma memorizzato – (concetto introdotto da **Von Neumann**)

istruzioni e **dati** sono codificati e caricati nella memoria del calcolatore, con la possibilità di eseguire programmi diversi → elaboratore elettronico

→ la memoria ha una struttura mono-dimensionale

→ le istruzioni sono memorizzate in celle adiacenti: l'esecutore, dopo ogni istruzione, *cerca l'istruzione seguente*, salvo diversa indicazione

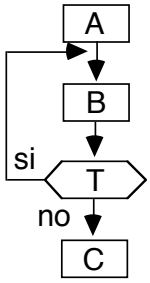
→ trasformare uno schema a blocchi in una struttura mono-dimensionale: *linearizzazione di un programma*

→ è necessaria una istruzione di *salto*

A, B e C sono istruzioni imperative

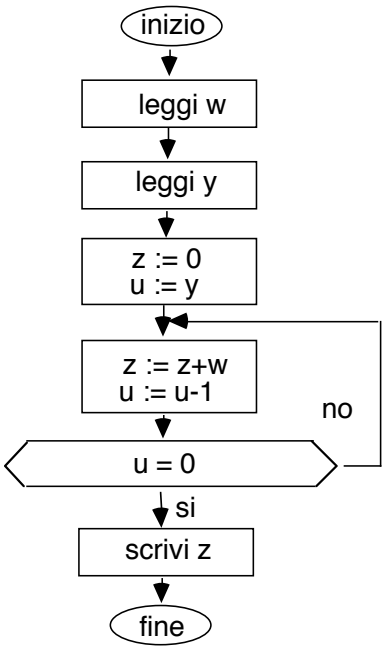
T è un'istruzione di salto

Schema a blocchi	Programma linearizzato in memoria



Indirizzo simbolico	
	A
CICLO	B
	Se T salta a CICLO
	C

Esempio : **programma della moltiplicazione**




```
        leggi W
        leggi Y
        Z := 0
        U := Y
PROD    Z := Z + W
        U := U – 1
        se U ≠ 0 salta a    PROD
        scrivi Z
        fine
```

IL LINGUAGGIO ASSEMBLATIVO

Il programma deve essere riportato in memoria in forma binaria:

Come codificare *le istruzioni di un programma in forma binaria* ?

Occorre una descrizione più dettagliata del calcolatore.

Linguaggio assemblativo:

È un linguaggio che definisce le operazioni elementari effettivamente disponibili sul calcolatore.

Ogni operazione è definita da

un **codice operativo** simbolico

un (eventuale) **indirizzo di memoria** simbolico

Il Linguaggio assemblativo dipende dallo specifico calcolatore.

Semplice esempio di Linguaggio assemblativo

- caricamento operando – LOAD: LOAD in R1 A (LOAD in R2 A)
  carica in R1 (R2) il dato memorizzato alla cella il cui indirizzo è A
- operazioni aritmetiche sui registri - ADD, SUB :
  somma (sottrai) il contenuto di R2 a quello di R1 e memorizza il risultato in R1
- operazioni di test - JUMP to A:
  se R1 ≠ 0 la prossima istruzione da eseguire è nella cella di indirizzo A, altrimenti procede in sequenza
- memorizzazione dei registri – STORE: STORE R1 in A (STORE R2 in A)
  memorizza il contenuto di R1 (R2) nella cella di indirizzo A
- acquisizione di un dato dall'esterno - READ: READ A
  leggi un dato e memorizzalo in A
- invio di un dato all'esterno - WRITE: WRITE A
  scrivi il valore della cella A
- arresto dell'esecuzione (STOP)

Traduzione in Linguaggio assemblativo

Una generica istruzione complessa deve essere tradotta in sequenze di operazioni elementari del Linguaggio assemblativo

Esempio

Z := Z + W

{
load in R1 Z
load in R2 W
add
store R1 in Z

RAPPRESENTAZIONE DEL PROGRAMMA IN MEMORIA

Linguaggio binario / Linguaggio macchina / Linguaggio assoluto

Ogni istruzione in linguaggio assemblativo è trasformata in **una istruzione** in linguaggio binario

- ogni istruzione è codificata come sequenza di bit
- ogni istruzione è costituita di due parti:
  - **codice operativo**, operazione ed eventualmente i registri interessati
  - **indirizzo dell'operando**, posizione in memoria del dato
- Nell'esempio precedente sono state individuate 10 operazioni
  → almeno 10 sequenze di bit distinti per codificare i codici operativi
  → 4 bit di codifica ($2^4 = 16$) → il codice operativo deve essere di 4 bit
- nelle istruzioni senza operando l'indirizzo dell'operando viene ignorato

Programma in memoria

Indirizzo Cella	Codice Operativo	Indirizzo Operando
00000	1000	10010
00001	1000	10011
00010	0010	10100
00011	0110	10101
00100	0010	10011
00101	0110	10110
00110	0010	10101
00111	0011	10010
01000	0101	-----
01001	0110	10101
01010	0010	10110
01011	0011	10111
01100	0100	-----
01101	0110	10110
01110	0010	10110
01111	1100	00110

Indirizzo Cella	Codice Operativo	Indirizzo Operando
10000	1001	10101
10001	0000	-----
10010	----	-----
10011	----	-----
10100	0000	00000
10101	----	-----
10110	----	-----
10111	0000	00001
11000	----	-----
11001	----	-----
11010	----	-----
11011	----	-----
11100	----	-----
11101	----	-----
11110	----	-----
11111	----	-----

LIVELLI DI RAPPRESENTAZIONE DEL PROGRAMMA

1. definizione del problema	
2. rappresentazione bidimensionale (vari livelli di dettaglio)	linguaggi ad alto livello
3. rappresentazione linearizzata	
4. utilizzo delle sole istruzioni eseguibili dalla specifica macchina, con codice operativo simbolico ed indirizzi simbolici	linguaggi assemblativi
5. istruzioni con codici operativi ed indirizzi binari	linguaggi assoluti

LINGUAGGI DI ALTO LIVELLO ⇔ MAGGIORE POTENZA ESPRESSIVA

PROD Z := Z + W U := U - 1 se U <> 0 salta a PROD	ripeti Z := Z + W U := U - 1 fino a che U = 0

OSSERVAZIONI

- ❏ i linguaggi di alto livello sono (in larga misura) indipendenti dallo specifico elaboratore
- ❏ i linguaggi assemblativi sono fortemente legati allo specifico elaboratore
- ❏ 1 istruzione ad alto livello ⇔ molte istruzioni in linguaggio assemblativo
- ❏ 1 istruzione in linguaggio assemblativo ⇔ 1 istruzione in linguaggio assoluto

TRADUZIONE DI UN PROGRAMMA IN LINGUAGGIO ASSOLUTO

PL1 - programma in linguaggio di alto livello

⇓ ← programma **traduttore**

PL2 - programma in linguaggio assoluto

Compilazione:

- Un programma, chiamato **compilatore,** prende in ingresso il programma PL1 (codice sorgente) e lo trasforma nel programma PL2 (codice binario) direttamente eseguibile sullo specifico calcolatore

Intepretazione:

- Un programma, chiamato **interprete,** prende in ingresso il programma PL1 e lo esegue "al volo" man mano che le istruzioni di PL1 sono lette, ovvero per ogni istruzione di PL1 fa eseguire al calcolatore le corrispondenti istruzioni in linguaggio assoluto

Classificazione dei linguaggi di programmazione

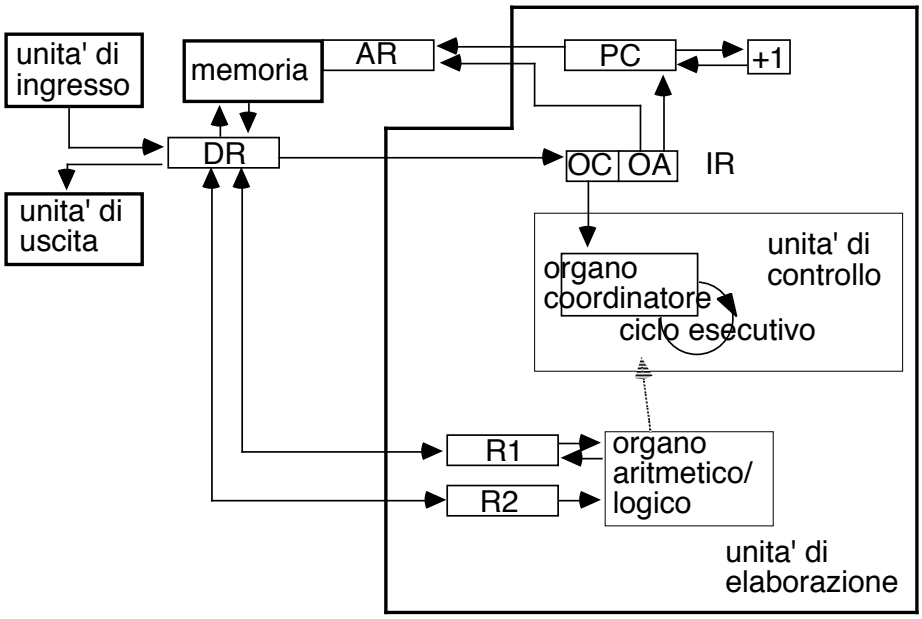
PER APPLICAZIONI:

- **scientifiche**
  FORTRAN (FORmula TRANslator)
- **commerciali**
  COBOL (COmmon Business Oriented Laanguage)
- **uso universale**
  PASCAL, BASIC, **C**, C++, JAVA
- **speciali**
  SQL basi di dati relazionali
  LISP (LISt Processing) risoluzione di problemi in forma simbolica
  PROLOG (PROGramming with LOGic) risoluzione di problemi in forma logica

PER CARATTERISTICHE

- **imperativi**
  programmazione libera
  FORTRAN (vecchie versioni), COBOL, BASIC
  programmazione strutturata
  PASCAL, C, FORTRAN (S), COBOL (S), BASIC (S)
- **orientati a oggetti**
  C++, JAVA
- **logico**: PROLOG
- **funzionale**: LISP

Struttura dell'elaboratore



ESECUZIONE DEL PROGRAMMA MEMORIZZATO

Unità di controllo: opera sul programma memorizzato in celle *consecutive.*

Ciclo di funzionamento:

1. acquisisce una istruzione in memoria
2. attiva i comandi per realizzare l'azione specificata dal codice operativo
3. individua l'istruzione successiva da acquisire (ed eseguire)

Necessari:

- un registro per contenere la posizione in memoria dell'istruzione da acquisire
  → PC = Program Counter
- un registro per contenere l'istruzione acquisita
  → IR = Instruction Register
- la possibilità di incrementare in valore di PC, per individuare la prossima istruzione in sequenza
- la possibilità di caricare un valore in PC, per eseguire le istruzioni di salto

CICLO DI ESECUZIONE DI UNA ISTRUZIONE

1. Fase interpretativa: (FETCH)
   acquisisce una istruzione e predispone l'indirizzo della seguente

i1)	(PC) → AR	comunicazione tra registri
i2)	(M^{AR}) → DR	prelievo dalla memoria
i3)	(DR) → IR	acquisizione dell'istruzione
i4)	(PC) + 1 → PC	incremento di registro

Notazioni:

(x) contenuto del registro o della cella x

M^{AR} cella di memoria il cui indirizzo è contenuto nel registro AR

→ Y trasferimento nel registro o nella cella Y del risultato di una espressione

2. Fase esecutiva:
   il registro IR è diviso in codice operativo (OC) e operando (OA)
   un organo coordinatore interpreta OC e compie le azioni opportune.
   La fase esecutiva dipende quindi dal codice operativo

e_{LD1}	e_{ST1}	e_{JN0}
1) (OA) → AR	1) (OA) → AR	$1_{\neq 0}$) (OA) → PC
2) (M^{AR}) → DR	2) (R1) → DR	$1_{=0}$) nulla
3) (DR) → R1	3) (DR) → M^{AR}	

e_{ADD}	e_{WRT}	e_{RDX}
1) (R1) + (R2) → R1	1) (OA) → AR	1) (OA) → AR
	2) (M^{AR}) → DR	2) input → DR
	3) (DR) → output	3) (DR) → M^{AR}