



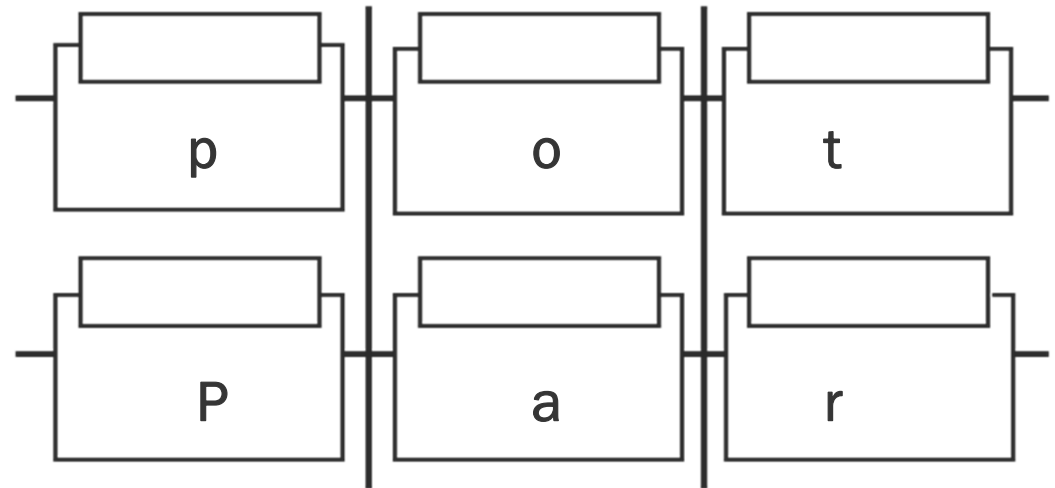
POLITECNICO
DI MILANO

INFORMATICA

La rappresentazione del
programma nel
calcolatore e la
dichiarazione delle
variabili

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base,  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

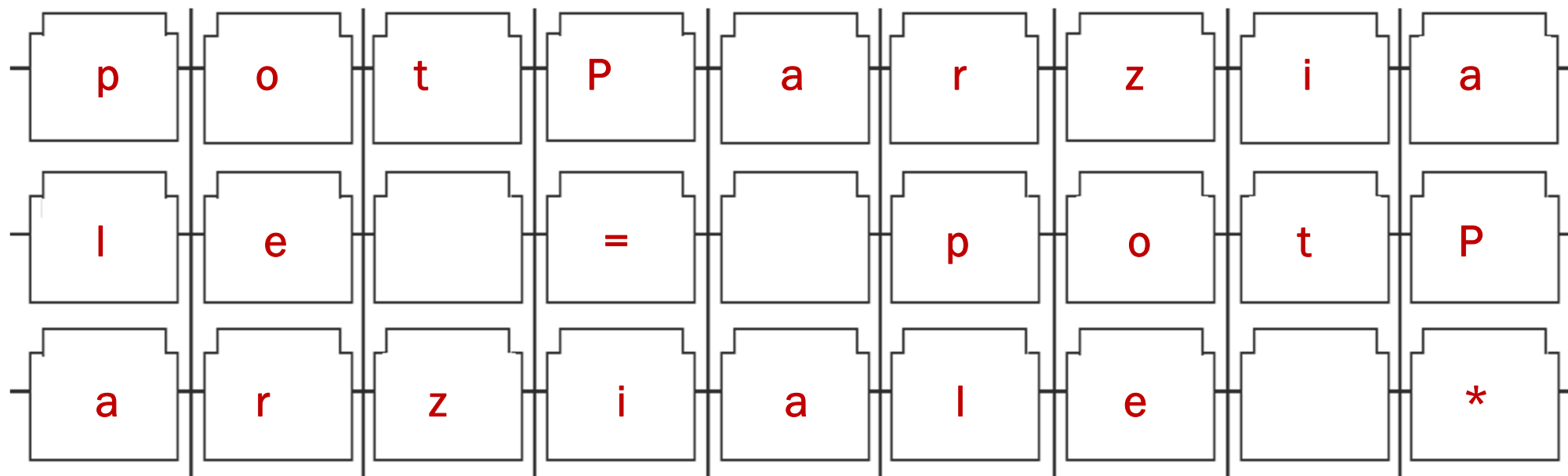
IL PROGRAMMA
IN MEMORIA

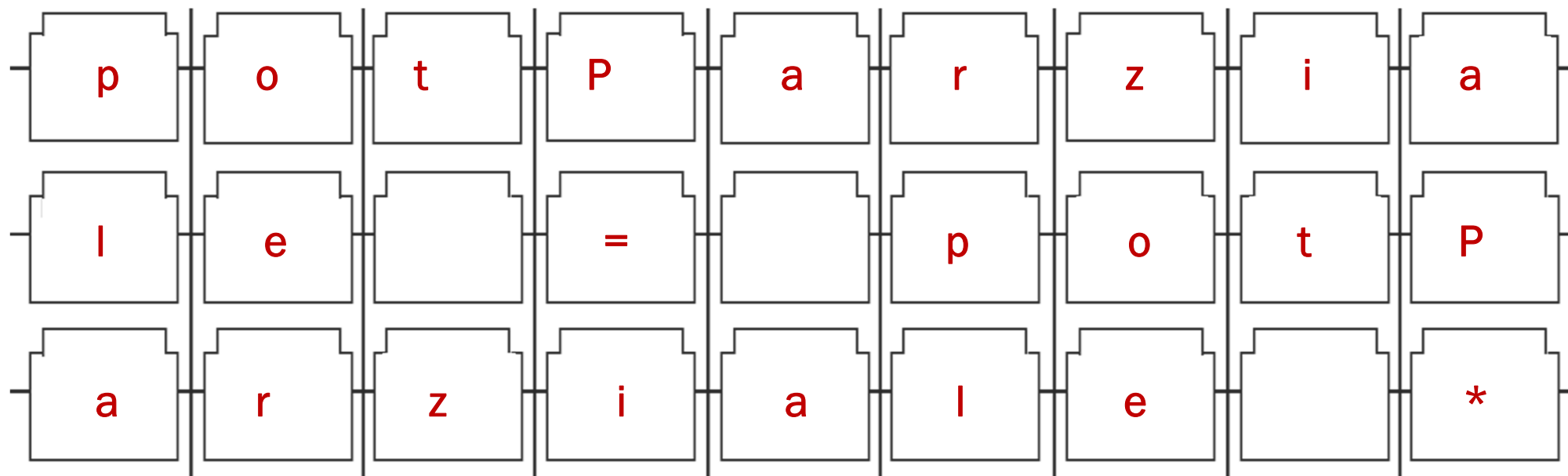
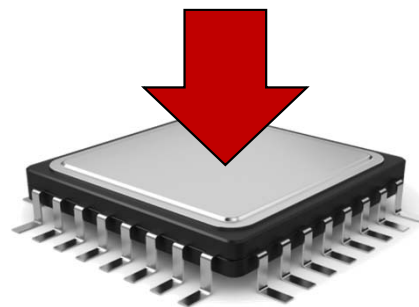


EDITOR

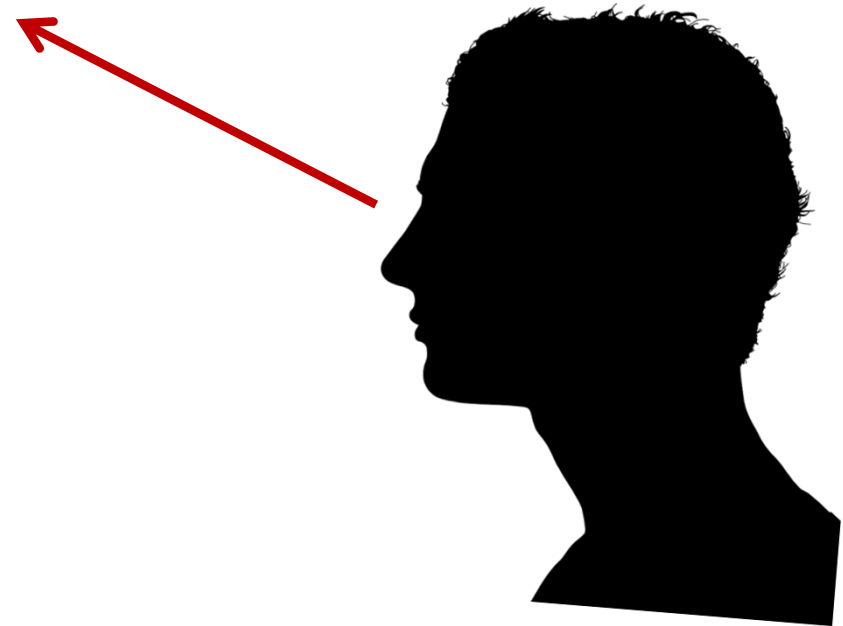


```
{ potParziale = potParziale * base;
```



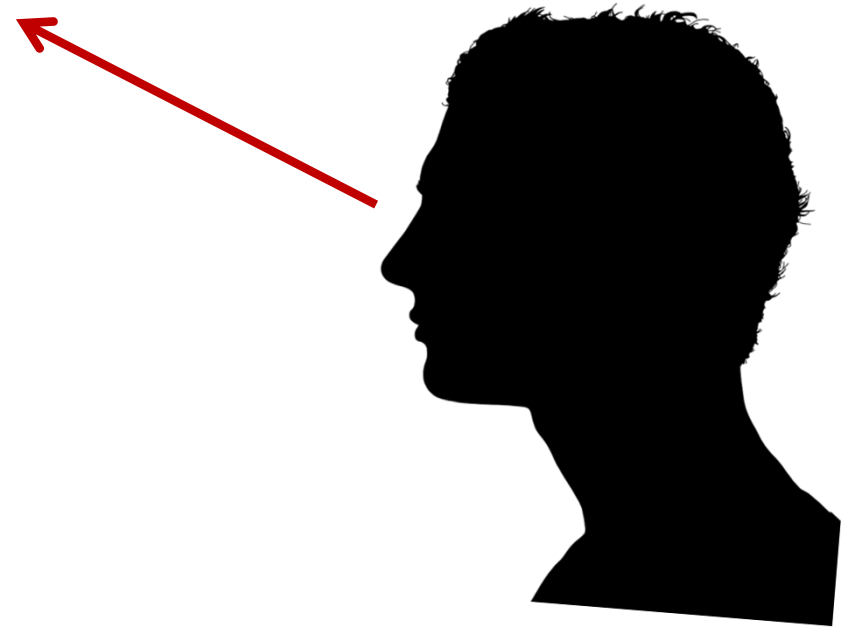
UNITÀ DI
CONTROLLO

```
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
} while (prodMancanti > 0);
```



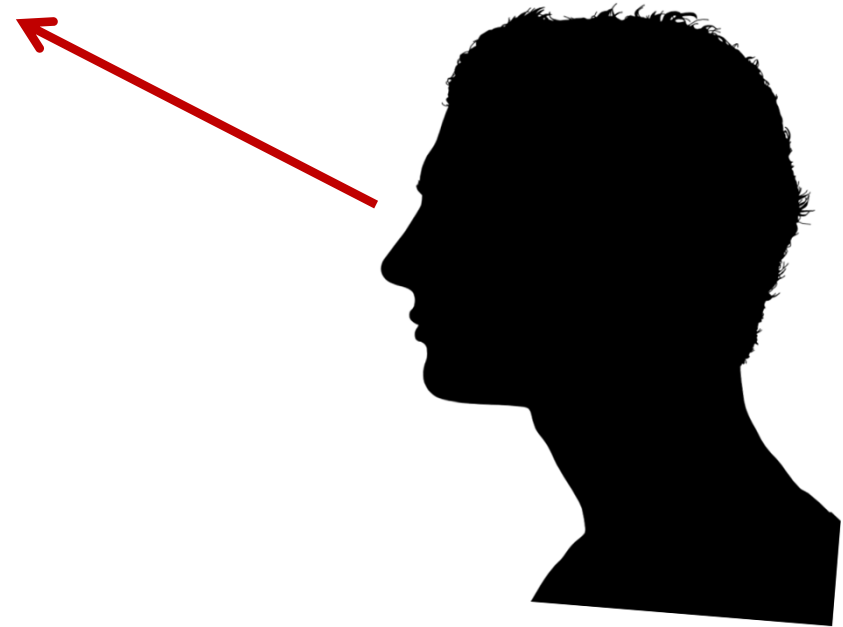
```
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
} while (prodMancanti > 0);
```

ASSEGNAZIONE



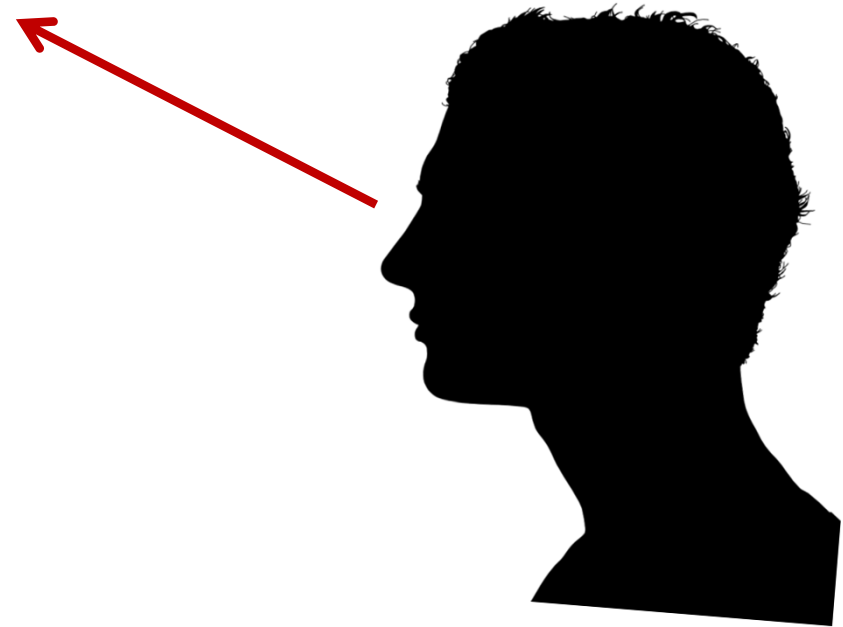
```
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
} while (prodMancanti > 0);
```

VARIABILE



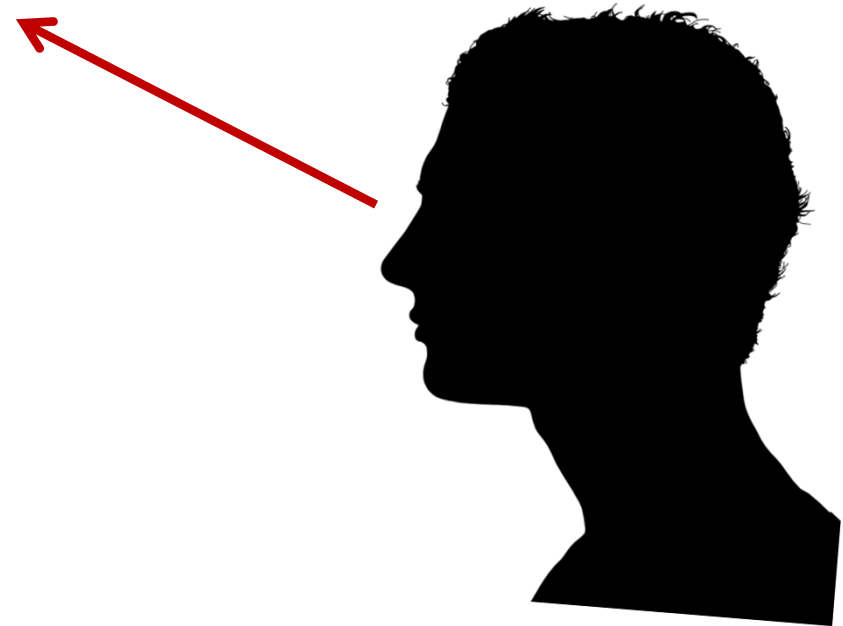
```
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
} while (prodMancanti > 0);
```

ESPRESSIONE




```
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
} while (prodMancanti > 0);
```

PRODOTTO

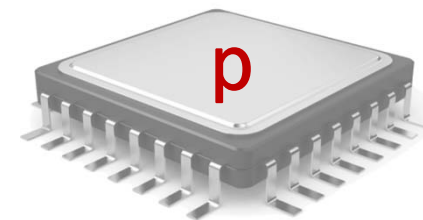


```
{ potParziale = potParziale * base;
```

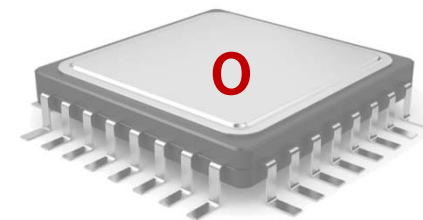
ELABORATORE



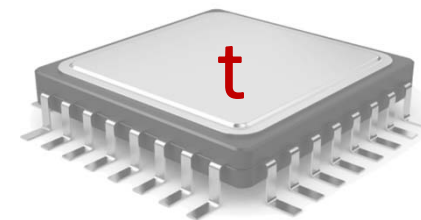
```
{ potParziale = potParziale * base;
```



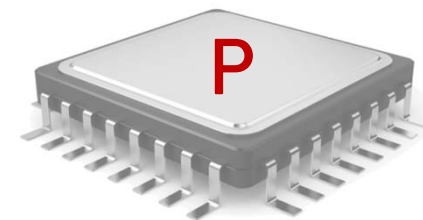
```
{ potParziale = potParziale * base;
```



```
{ potParziale = potParziale * base;
```

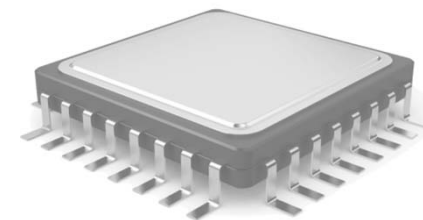


```
{ potParziale = potParziale * base;
```



{ **potParziale** = potParziale * base;

 nome

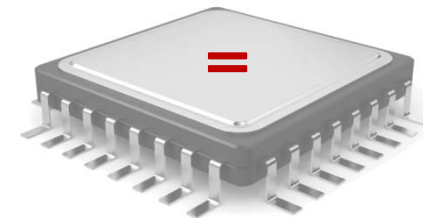


{ **potParziale** = potParziale * base;
 └──────────┘
 identificatore

IDENTIFICATORE



$\{ \text{potParziale} = \text{potParziale} * \text{base};$
 └──────────┘
 identificatore



{ potParziale ■ = potParziale * base;
└──────────┘
identificatore

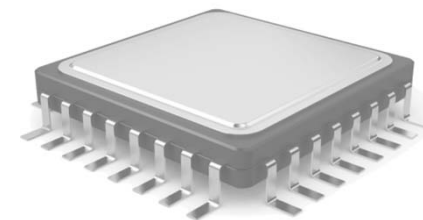
*Non sono ammessi spazi
all'interno dei nomi
delle variabili*



{ potParziale = potParziale * base;

└──────────┘ └┘

identificatore



```
{ potParziale = potParziale * base;
```

identificatore

operatore
di
assegnamento



`{ potParziale = potParziale * base;`

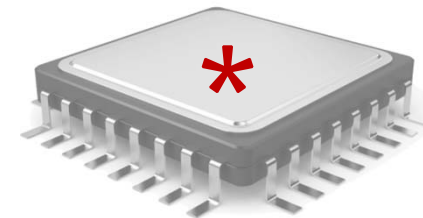
identificatore

operatore
di
assegnamento



{ potParziale = potParziale * base;

identificatore operatore di assegnamento identificatore operatore di prodotto

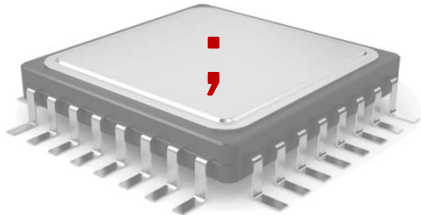


operatore
di
assegnamento



{ potParziale = potParziale * base;

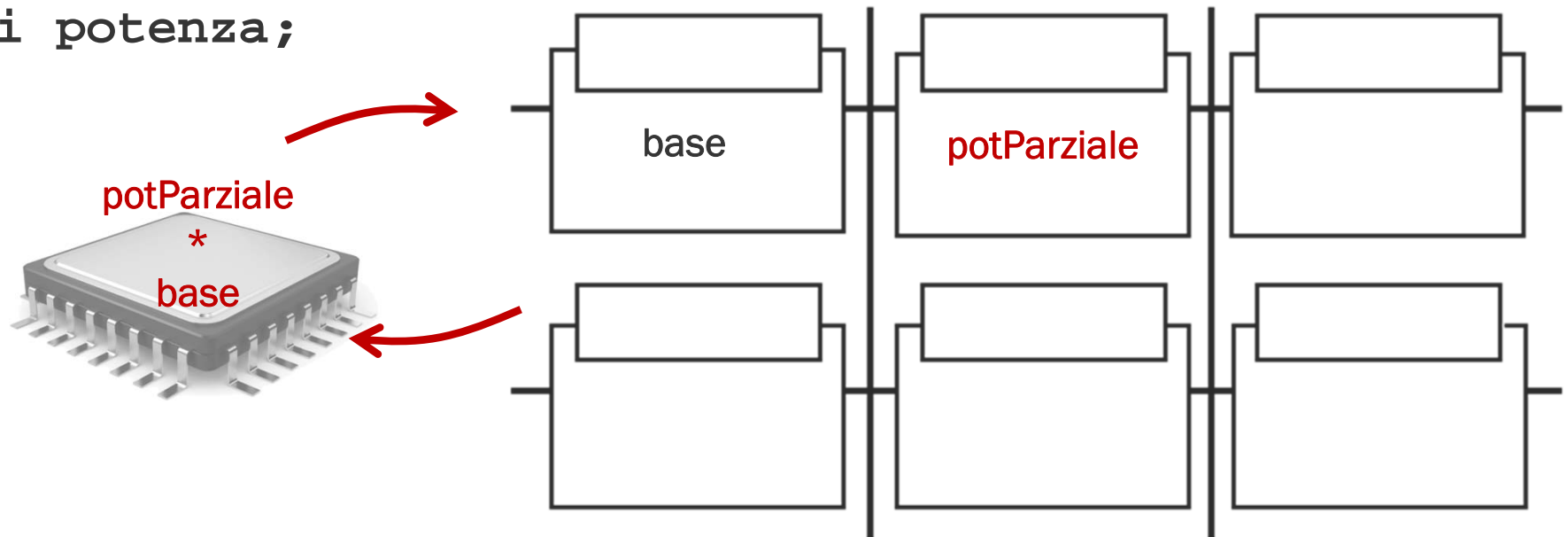
identificatore operatore di assegnamento identificatore operatore di prodotto identificatore.



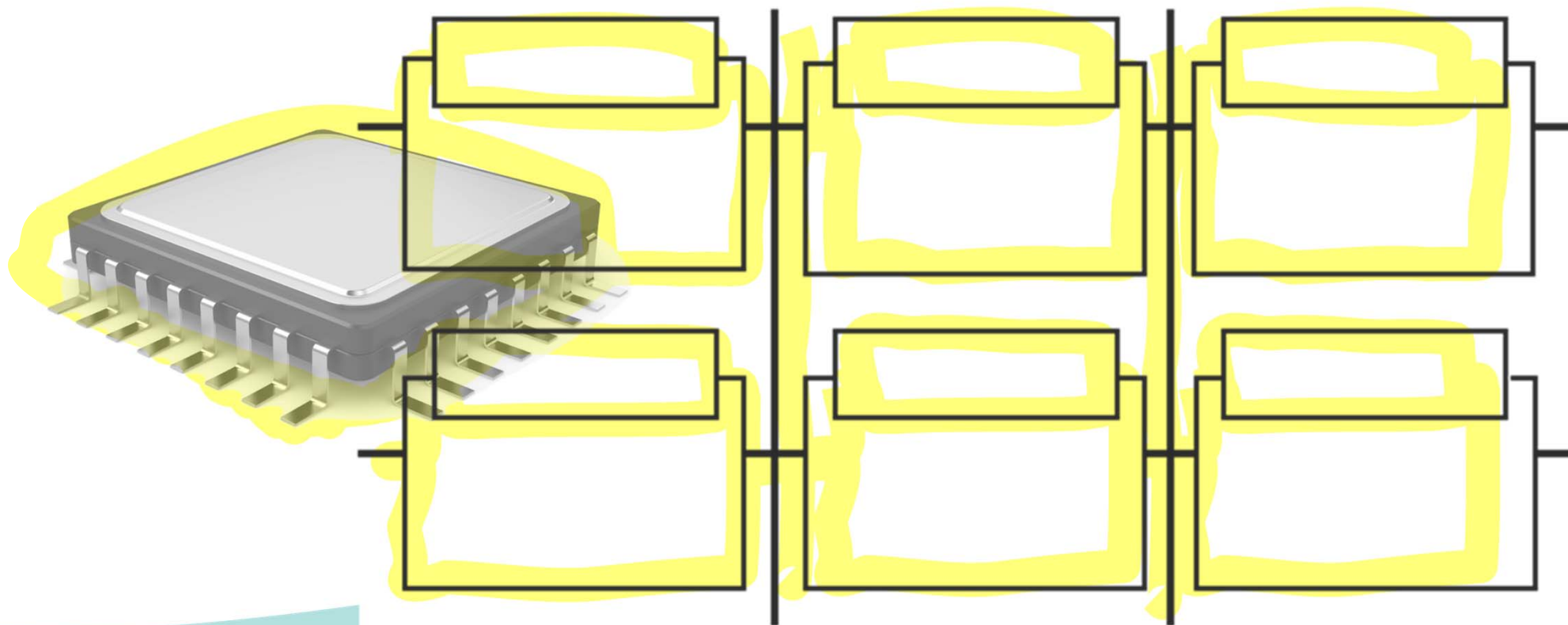
1. Termina
l'identificatore

2. Termina
l'istruzione


```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```



MEMORIA
CAPIENTE

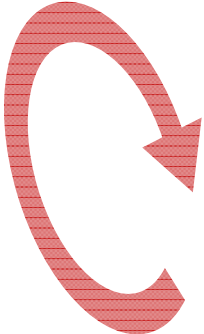


PROCESSORI
VELOCI

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

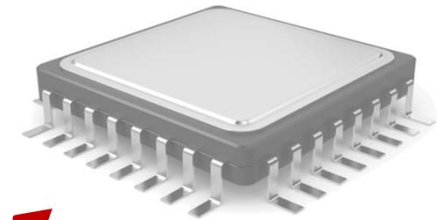
CICLO

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```



FORMATO SORGENTE

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```



FORMATO
SORGENTE



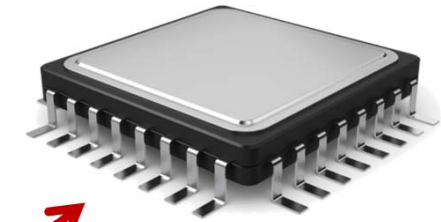
Editor

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

Trasformazione



PROGRAMMA
COMPILATORE

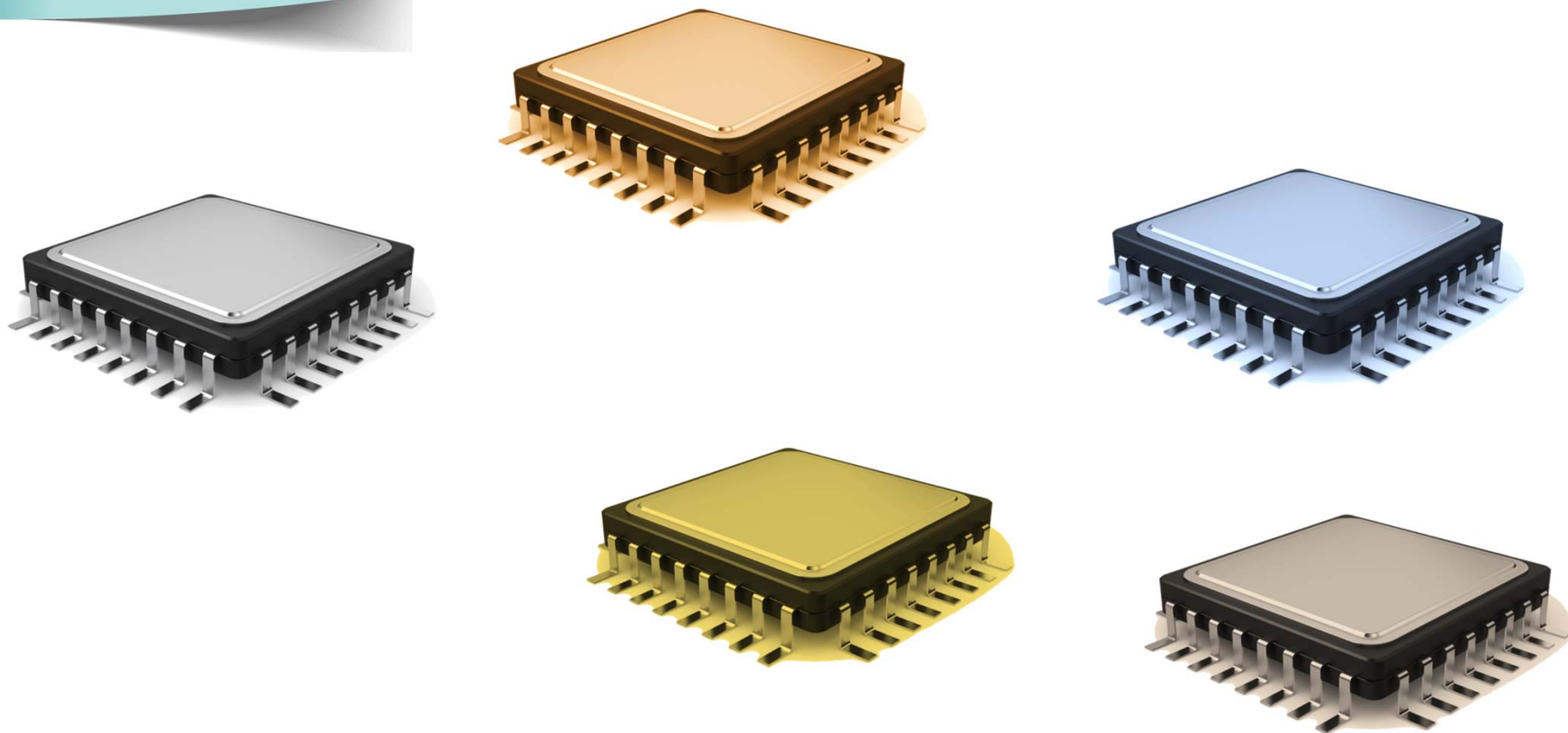


CODICE COMPILATO

CODICE SORGENTE

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

DIVERSI SET DI
ISTRUZIONI





Set di istruzioni
del processore

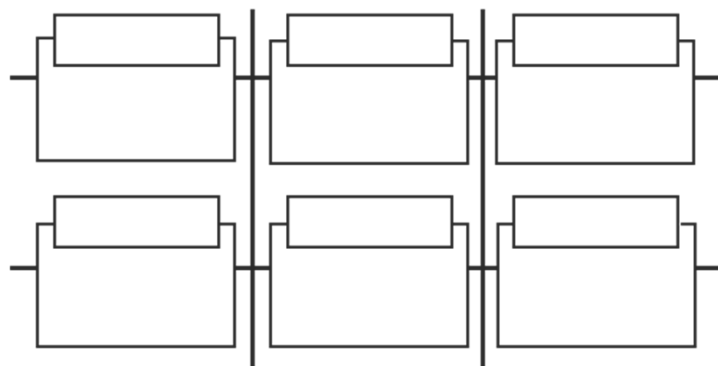
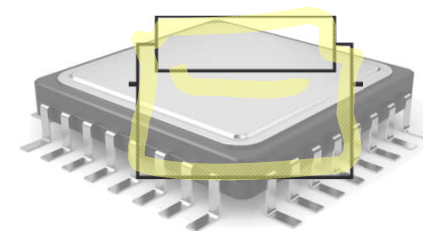
Programma compilatore

C++

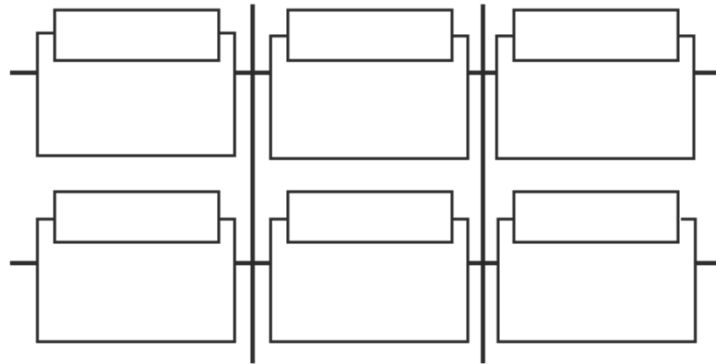
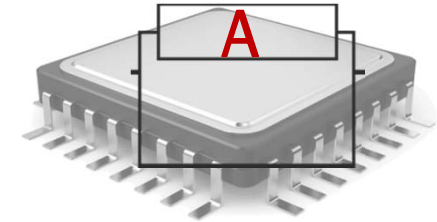
```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

REGISTRO

MEMORIA
TEMPORANEA



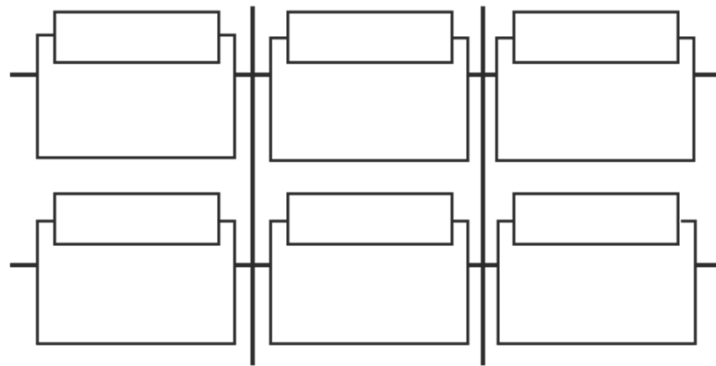
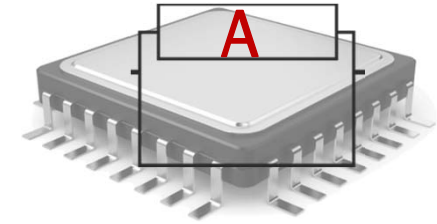
SINTASSI



LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

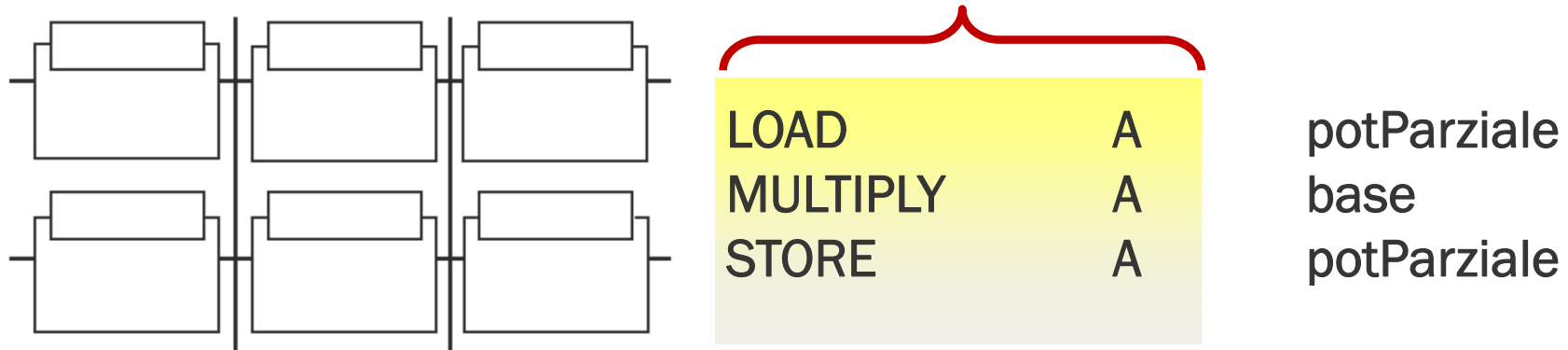
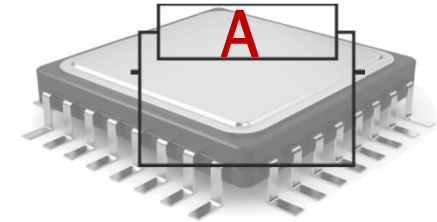
CAMPI



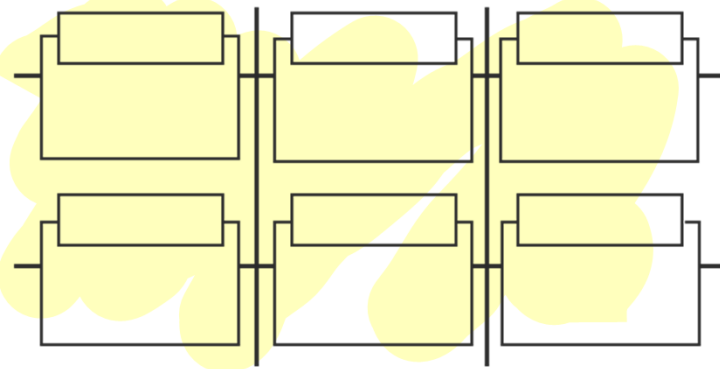
LOAD	A
MULTIPLY	A
STORE	A

potParziale
base
potParziale

```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

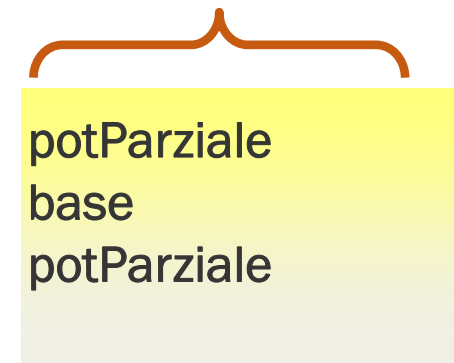
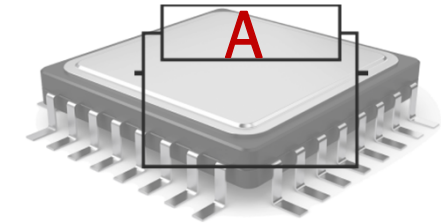
TIPO DI
ISTRUZIONE

```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

CASELLE DI
MEMORIA

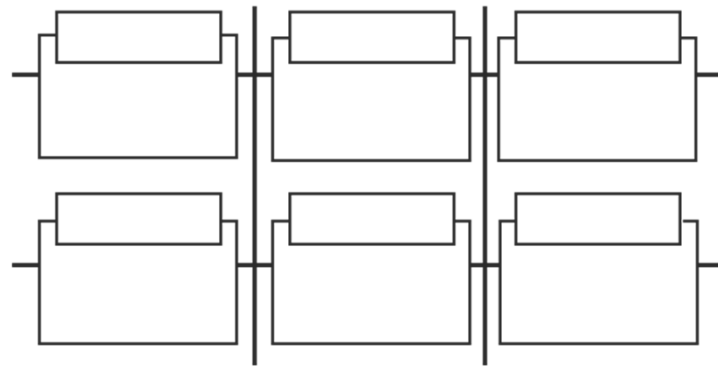
LOAD
MULTIPLY
STORE

A
A
A



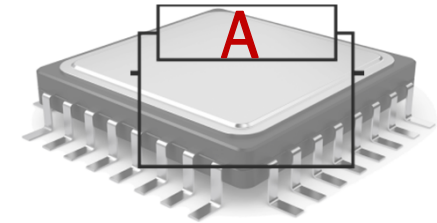
```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

SEMANTICA



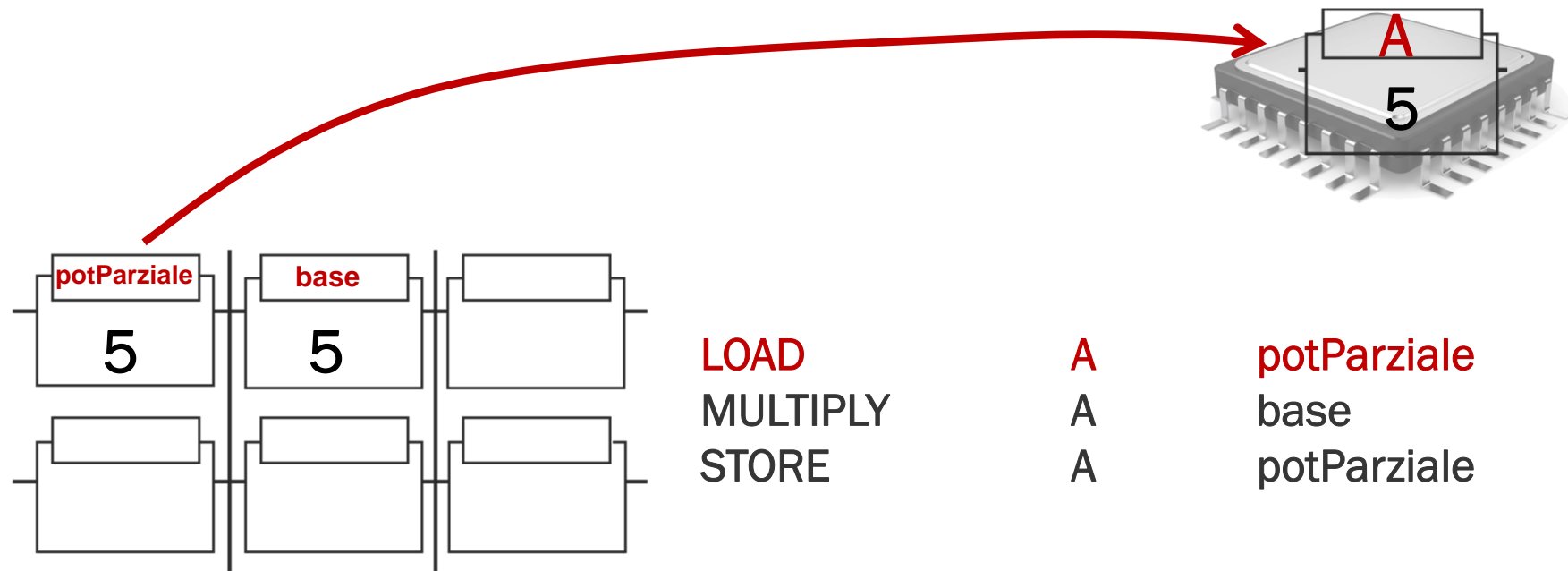
LOAD
MULTIPLY
STORE

A
A
A

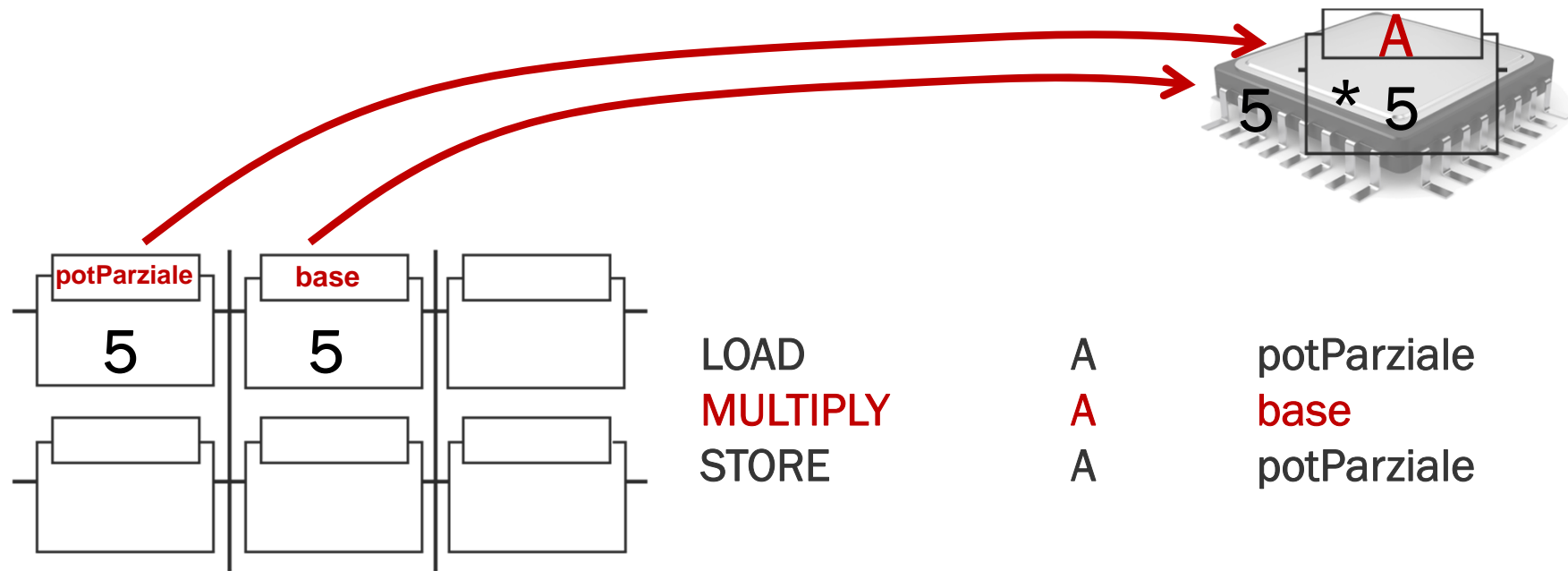


potParziale
base
potParziale

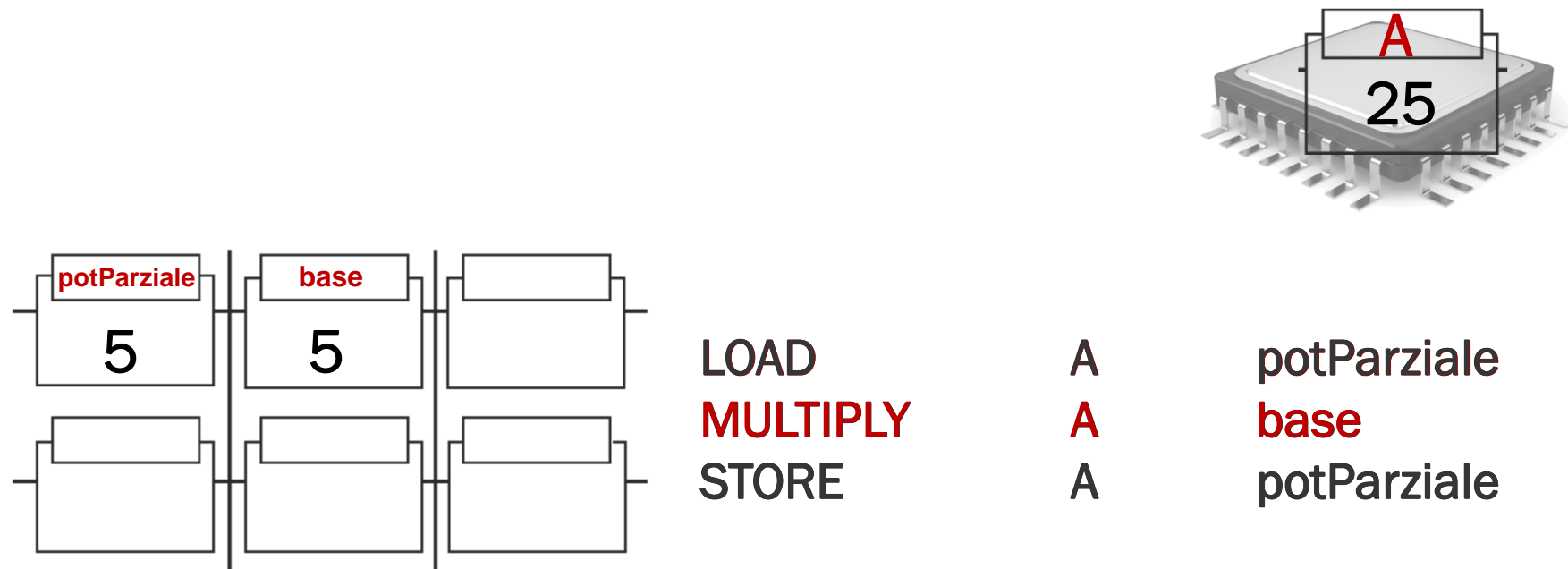
```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```



```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

```
do
{
    potParziale = potParziale * base;
    prodMancanti = prodMancanti - 1;
    ;
}
```



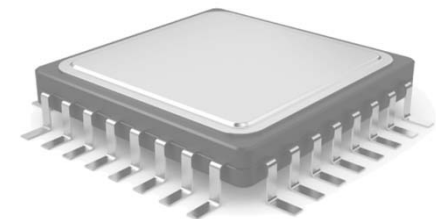
```
do
{ potParziale = potParziale * base;
  prodMancanti = prodMancanti - 1;
  ;
```

LINGUAGGIO ASSEMBLATORE

LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

SEQUENZA DI BIT

0110110010

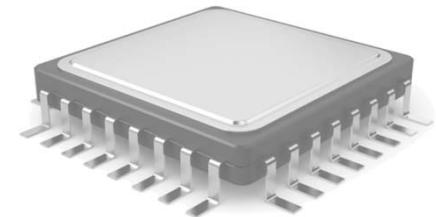


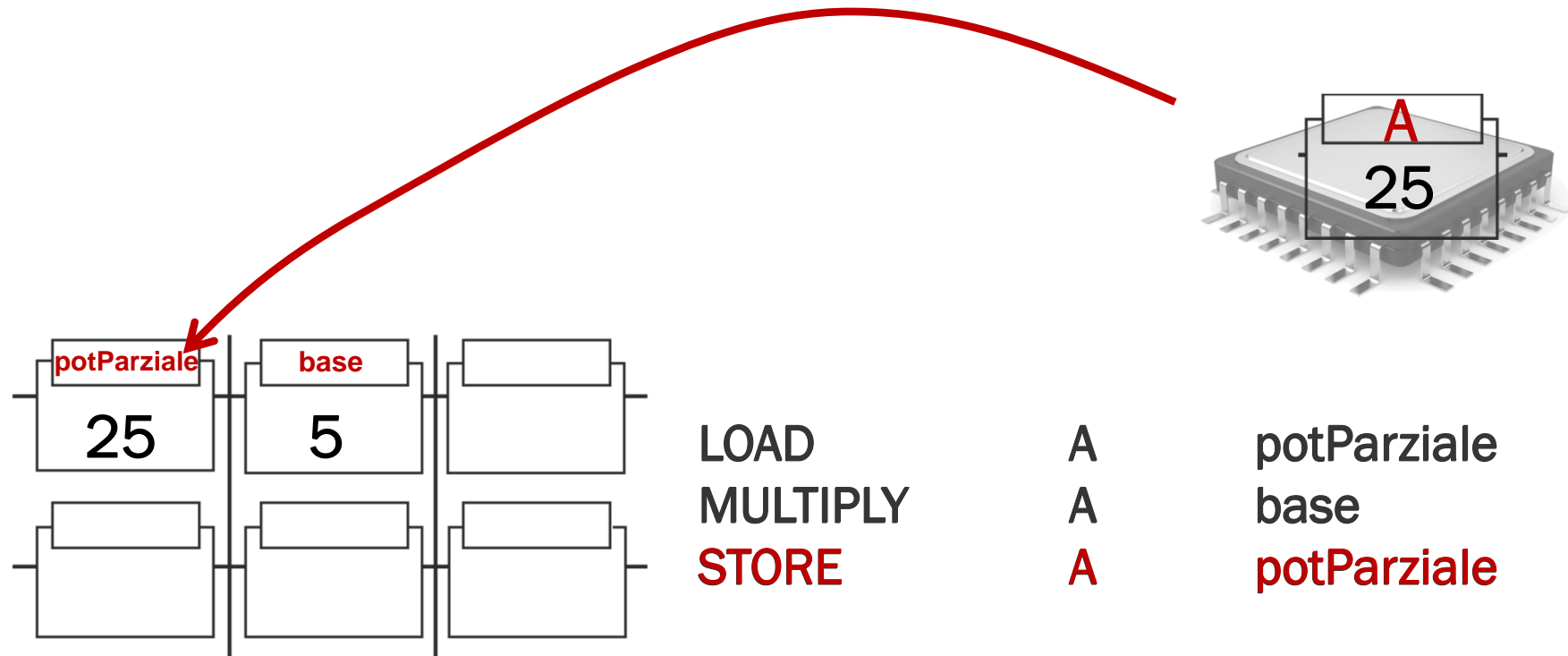
CAMPO CODICE
OPERATIVO

LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale
...		

128
istruzioni
= 7 bit

0110010





```
do  
{ potParziale = potParziale * base;  
  prodMancanti = prodMancanti - 1;  
  ;
```

POSIZIONE DI MEMORIA - CAMPO INDIRIZZO

LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

**0110010****01001101011001110**

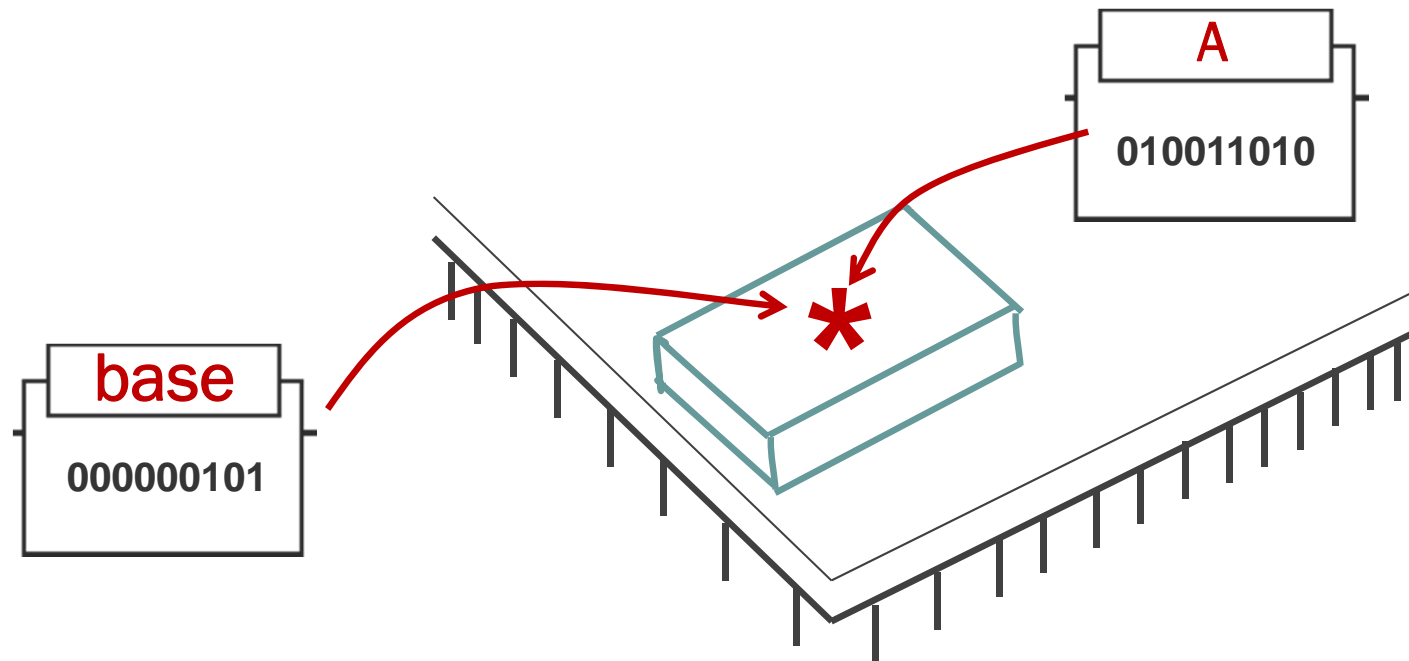
LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

0110010

0 001101011001110

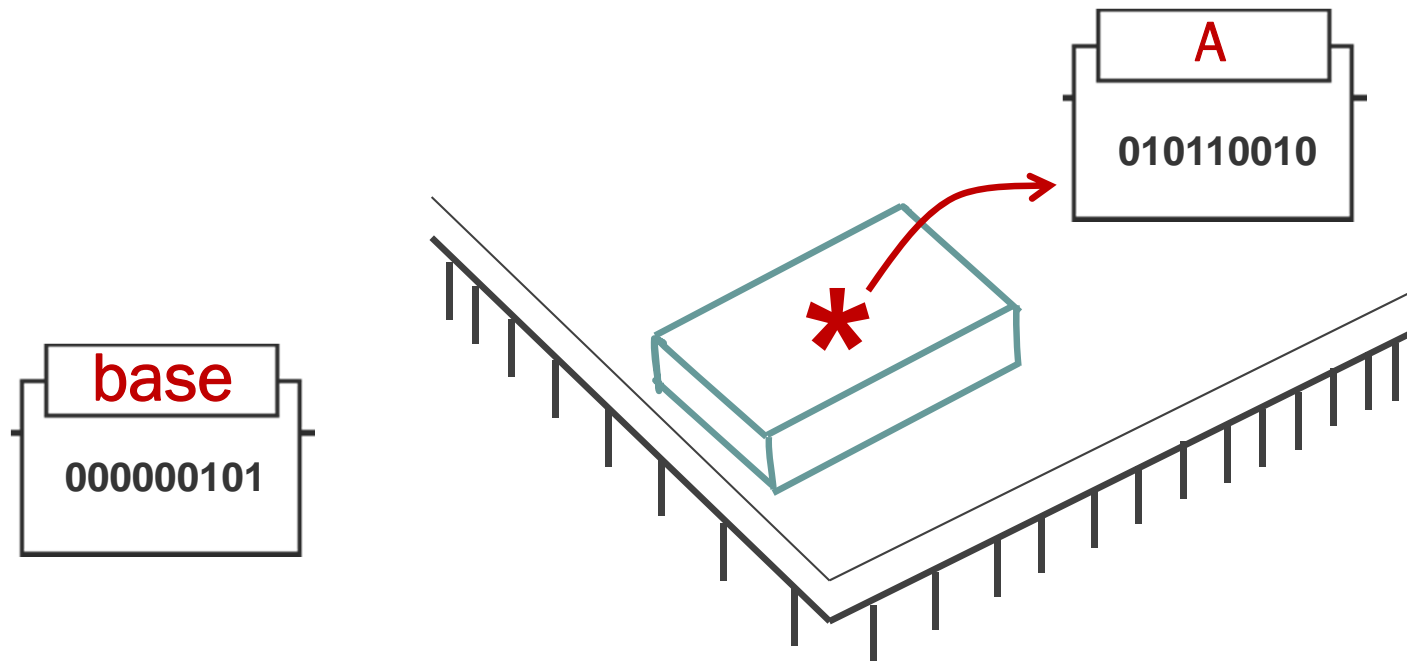
LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

A, base: interi con segno



LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

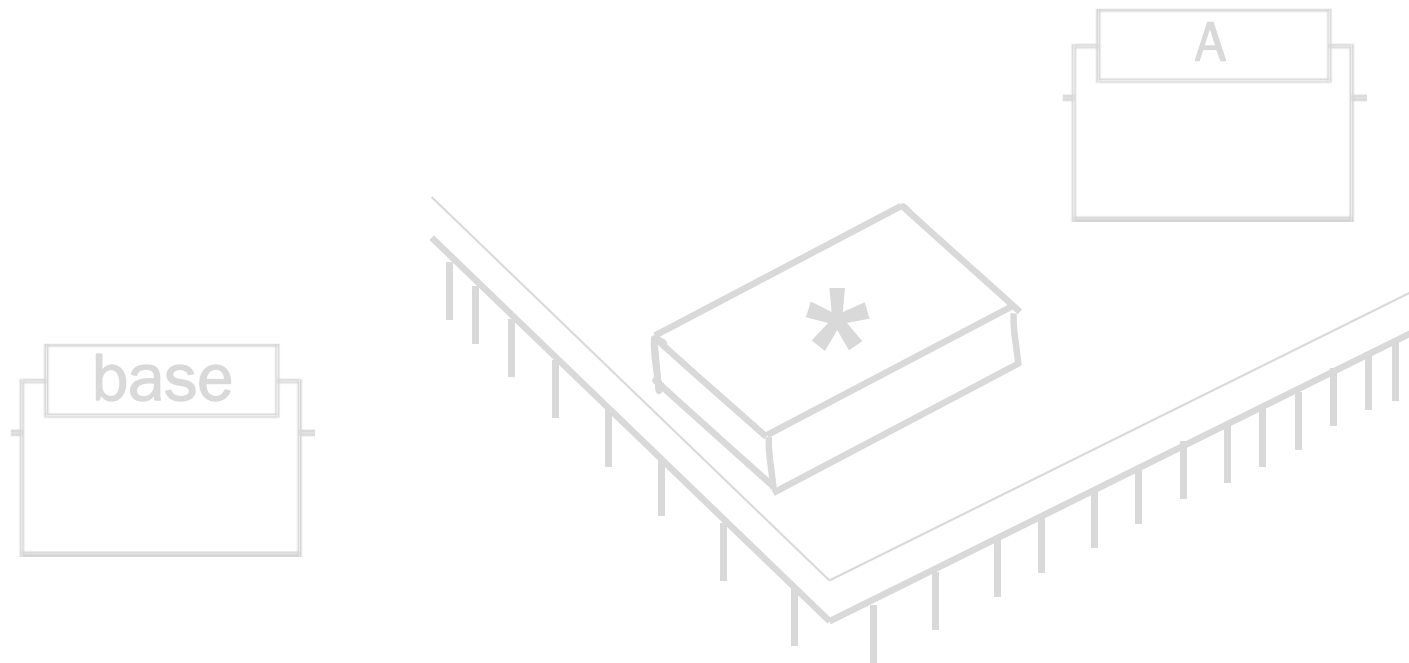
A, base: interi con segno



TIPO DEL VALORE

LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

A, base: interi con segno



TIPO DEL VALORE

LOAD A potParziale
MULTIPLY A base
STORE A potParziale

A, base: interi con segno

int

+	5
0	00000101



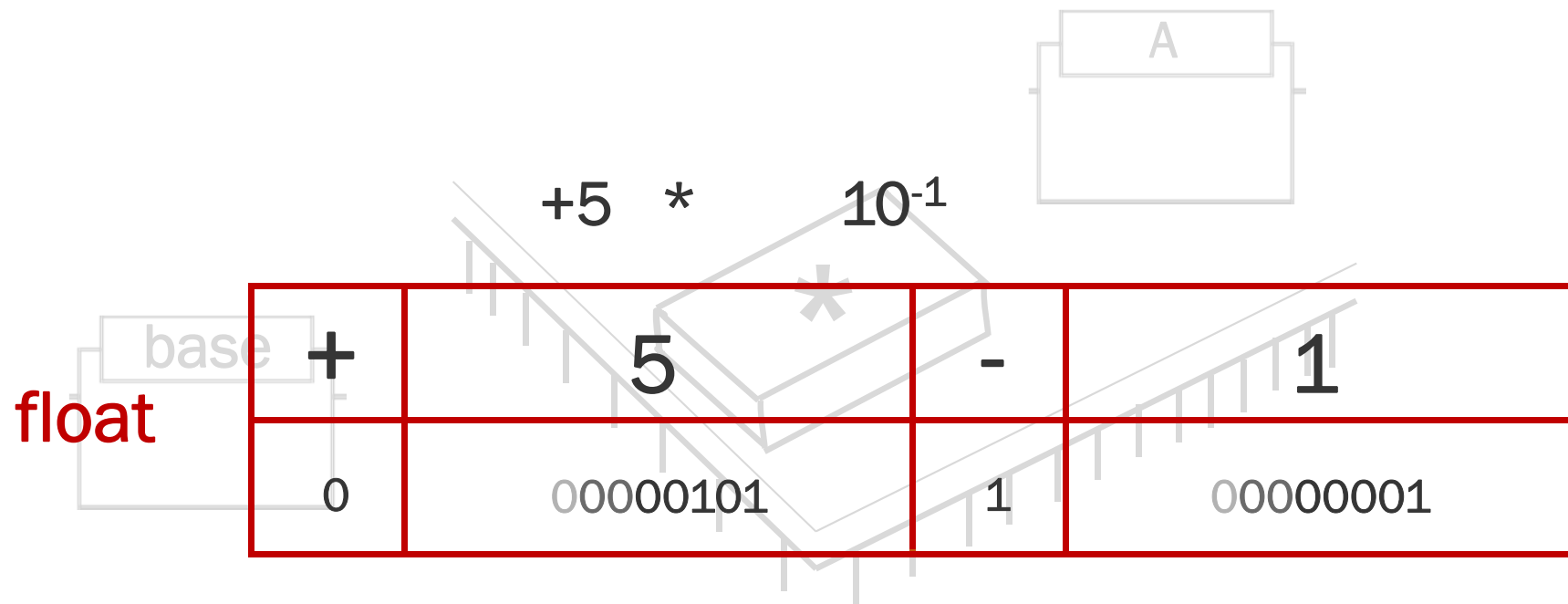
float

+	5	*	-	1
0	00000101	1	1	00000001

TIPO DEL VALORE

LOAD	A	potParziale
MULTIPLY	A	base
STORE	A	potParziale

A, base: floating point



LOAD
MULTIPLY
STORE

A potParziale
A **base** *A, base*
A potParziale

+	5
0	00000101

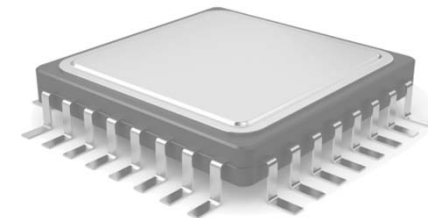
+	5	-	1
0	00000101	1	00000001

LINGUAGGIO OGGETTO

LOAD	A	potParziale	
MULTIPLY	A	base	<i>A, base</i>
STORE	A	potParziale	

0

00000101



0

00000101

1

00000001

DEFINIRE IL TIPO

```
void main()  
{ leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  { potParziale = potParziale * base;  
    prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```

INTERO DOTATO DI SEGNO

```
void main()  
{  int base, esponente, potenza; } int  
  int potParziale, prodMancanti;  
  leggi base;  
  leggi esponente;  
  potParziale = 1;  
  prodMancanti = esponente;  
  do  
  {  potParziale = potParziale * base;  
      prodMancanti = prodMancanti - 1;  
  } while (prodMancanti > 0);  
  potenza = potParziale;  
  scrivi potenza;  
}
```


IN VIRGOLA MOBILE

```
void main()  
    float base, esponente, potenza;  
    float potParziale, prodMancanti; } float  
    leggi base;  
    leggi esponente;  
    potParziale = 1;  
    prodMancanti = esponente;  
    do  
        { potParziale = potParziale * base;  
          prodMancanti = prodMancanti - 1;  
        } while (prodMancanti > 0);  
    potenza = potParziale;  
    scrivi potenza;  
}
```