

Stefan Sokołowski

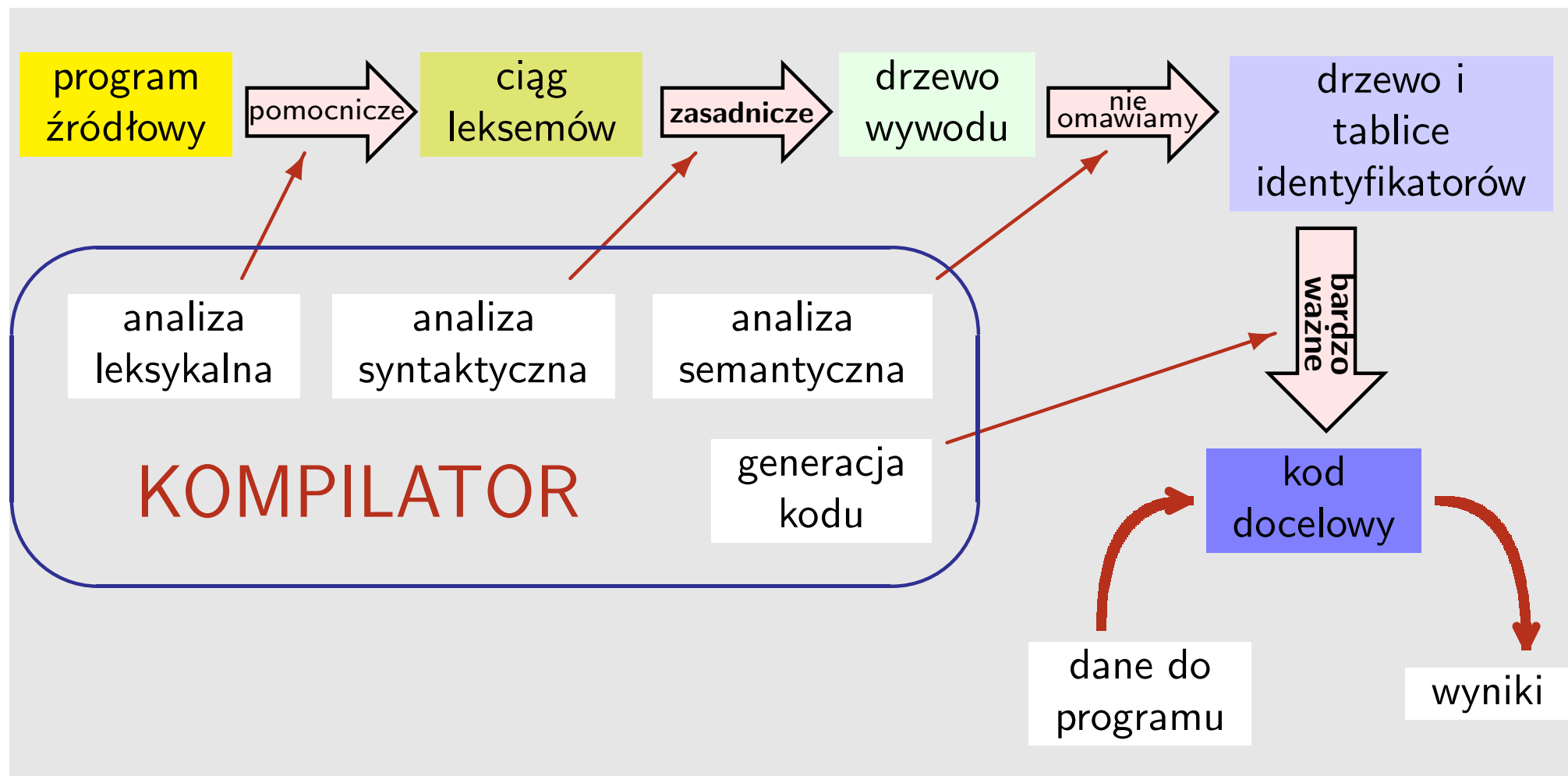
JĘZYKI FORMALNE I METODY KOMPILACJI

wykład 9

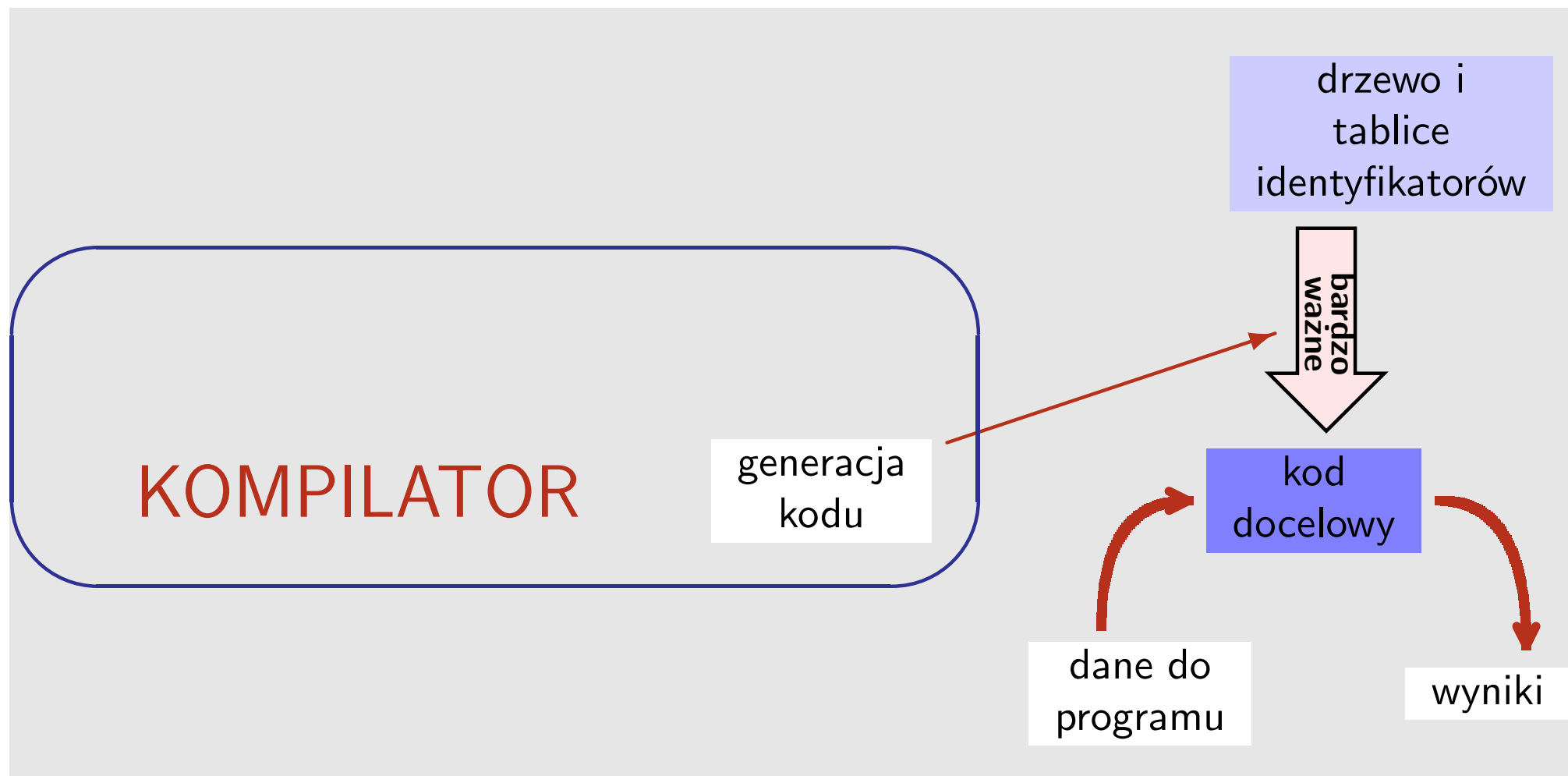
Inst. Informatyki Stosowanej, ANS

Elbląg, 2025/2026

Uproszczony schemat kompilacji

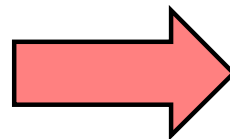


Uproszczony schemat kompilacji



Kompilacja

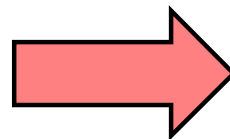
drzewo i
tablice
identyfikatorów



kod
docelowy

Kompilacja

drzewo i
tablice
identyfikatorów

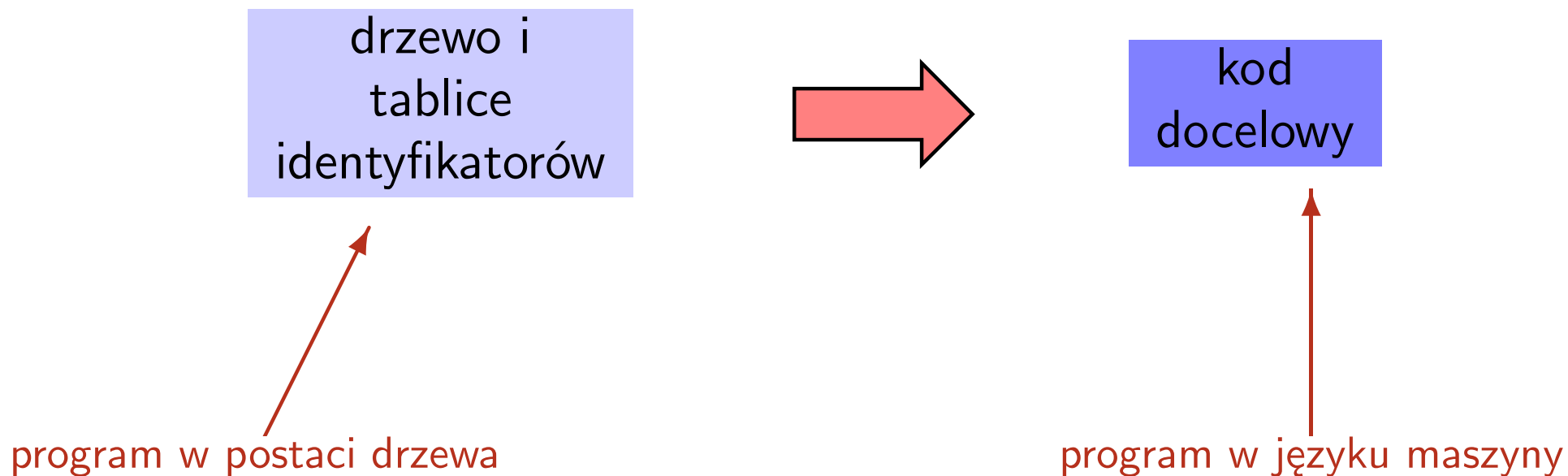


kod
docelowy

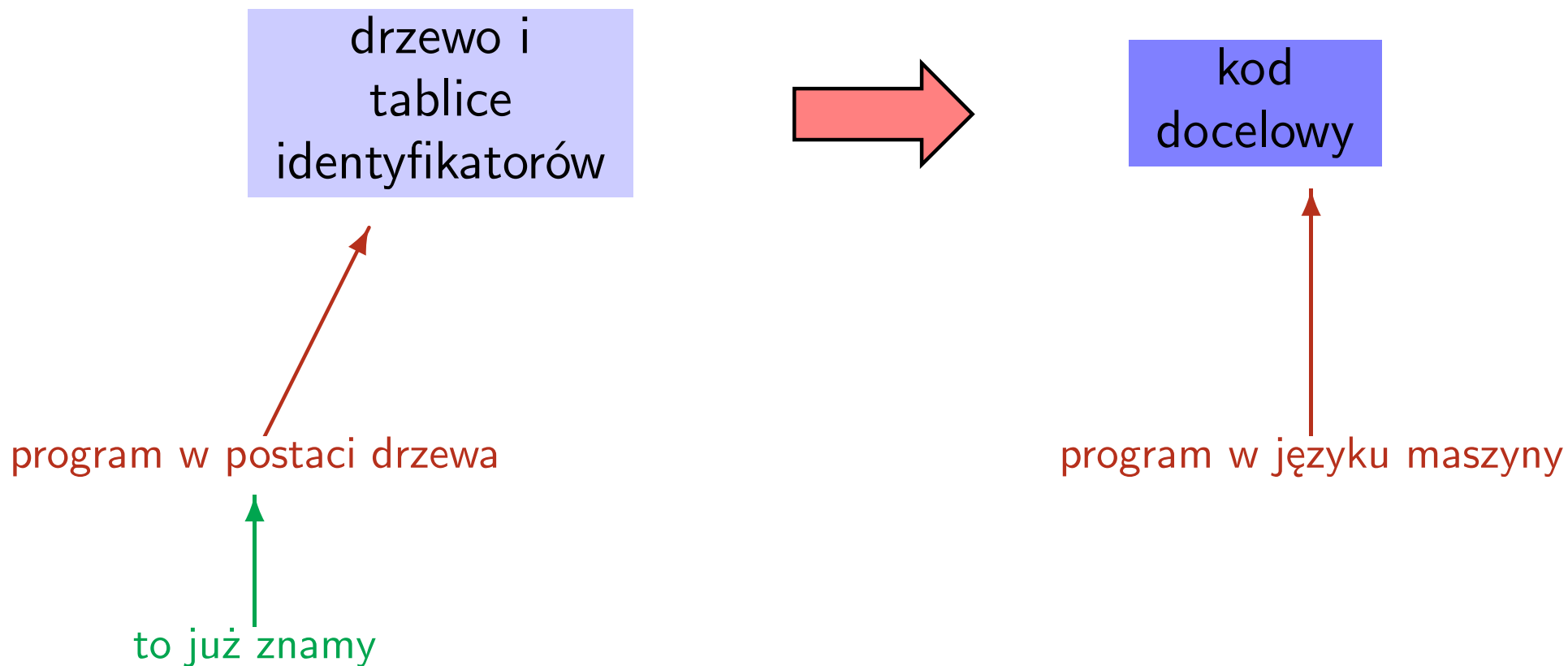
program w postaci drzewa



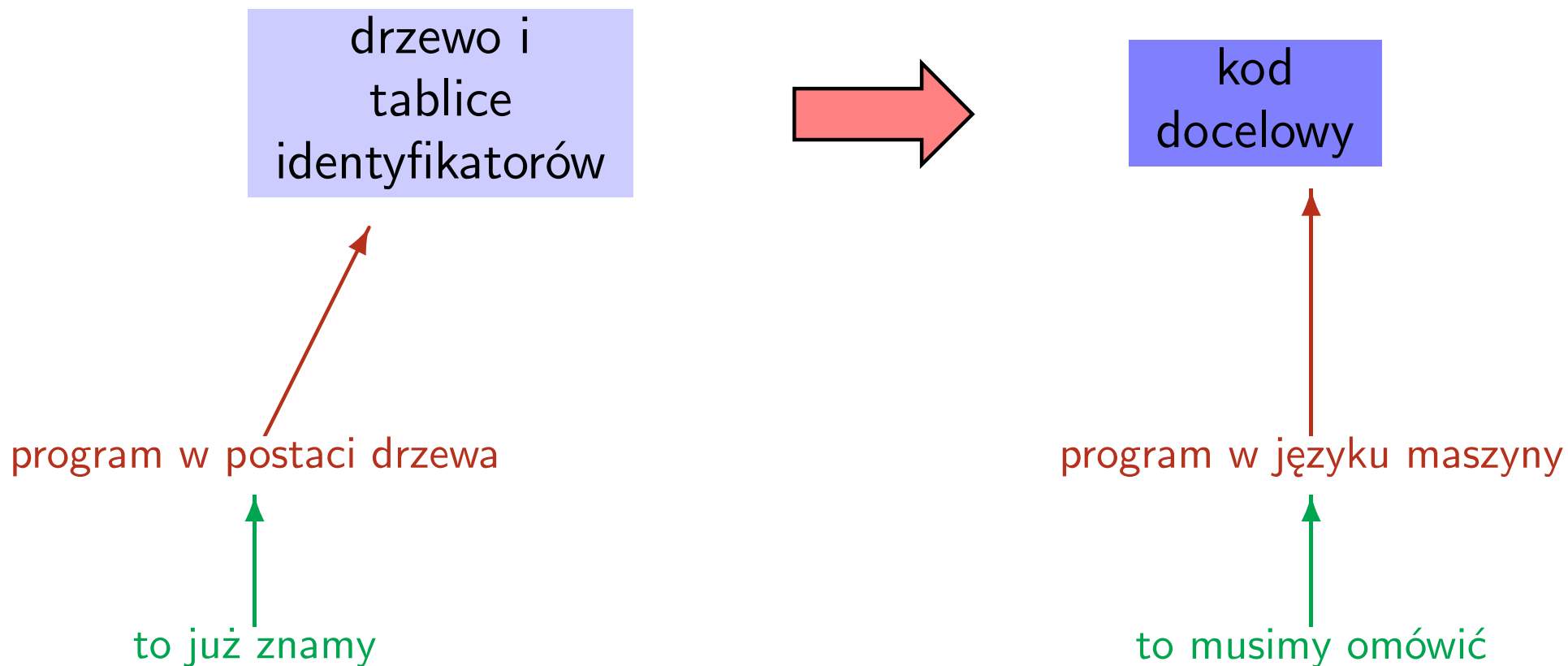
Kompilacja



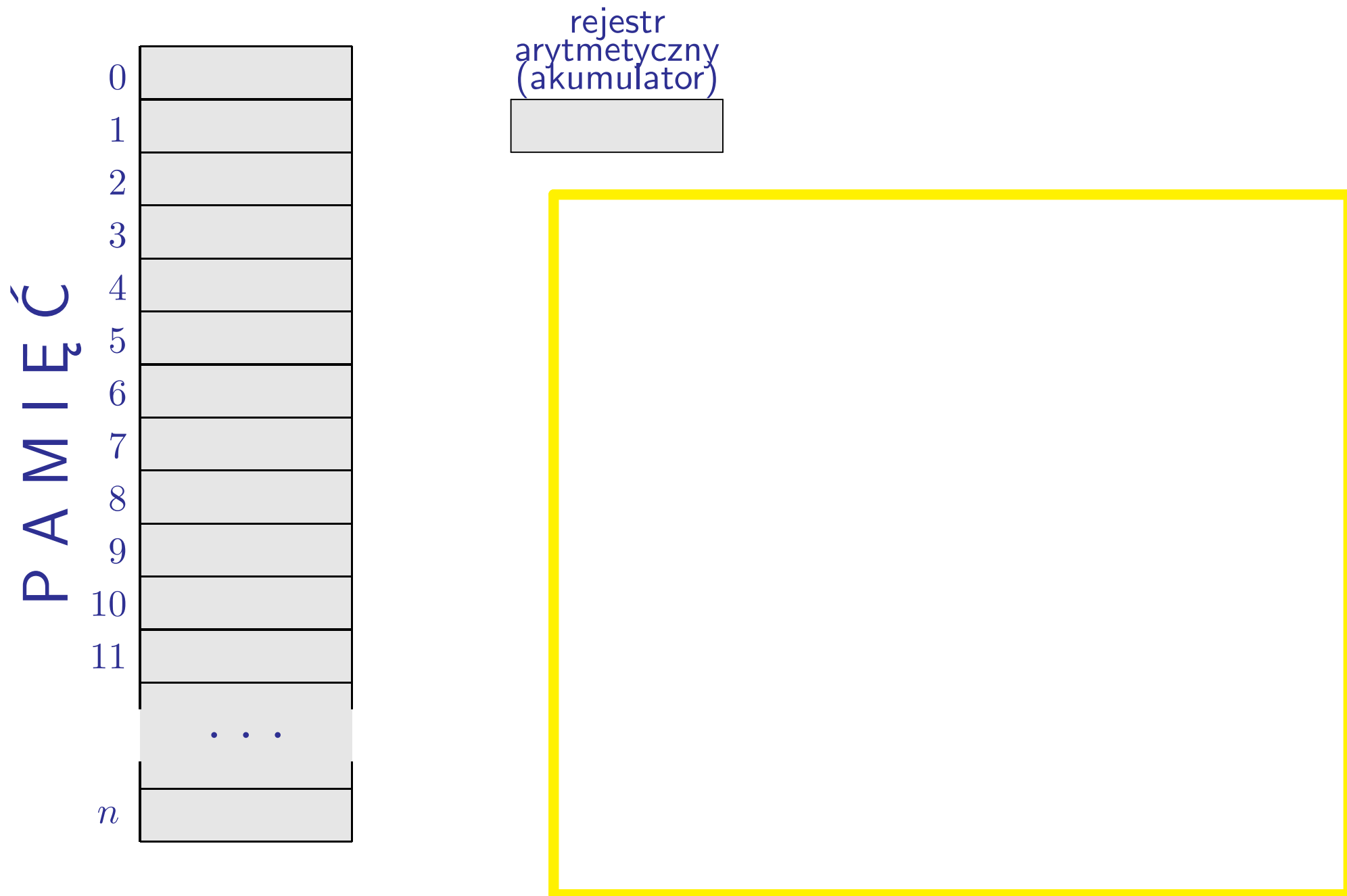
Kompilacja



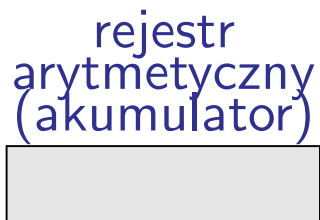
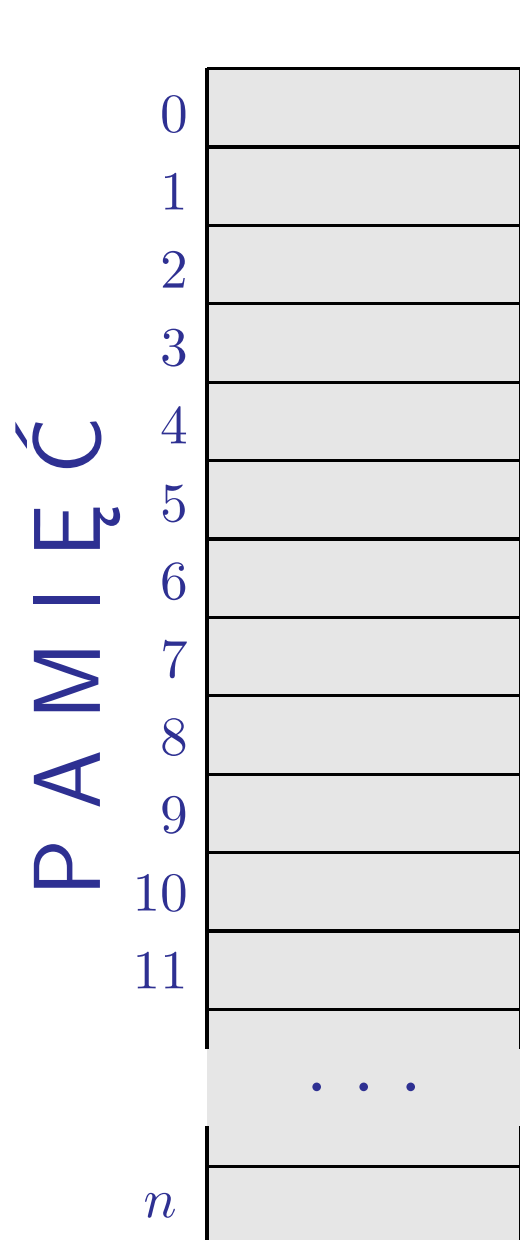
Kompilacja



Programowanie niskiego poziomu

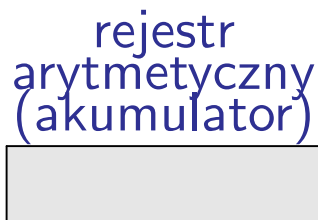
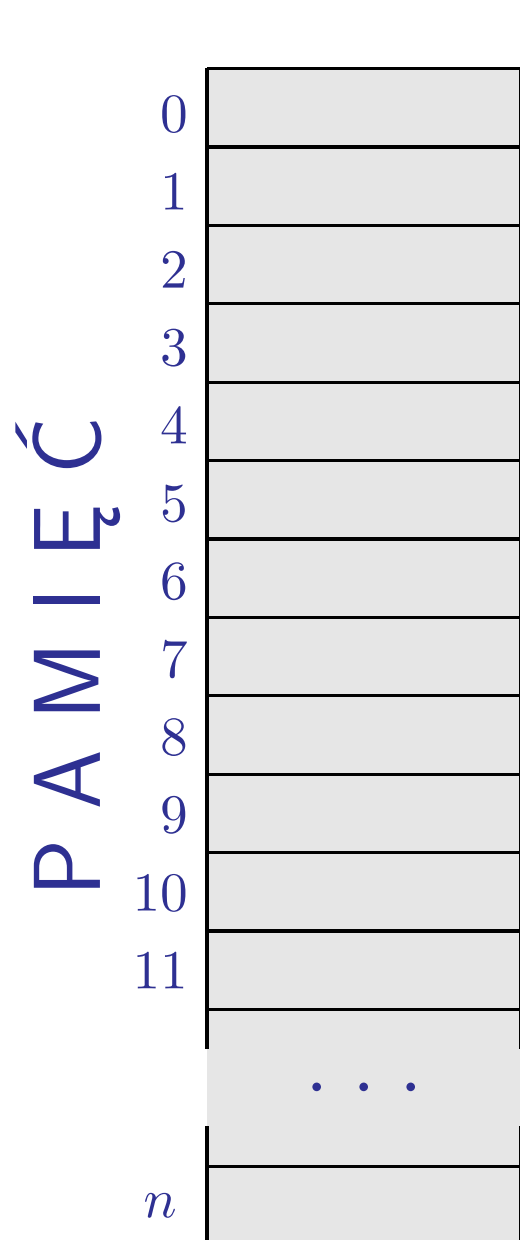


Programowanie niskiego poziomu



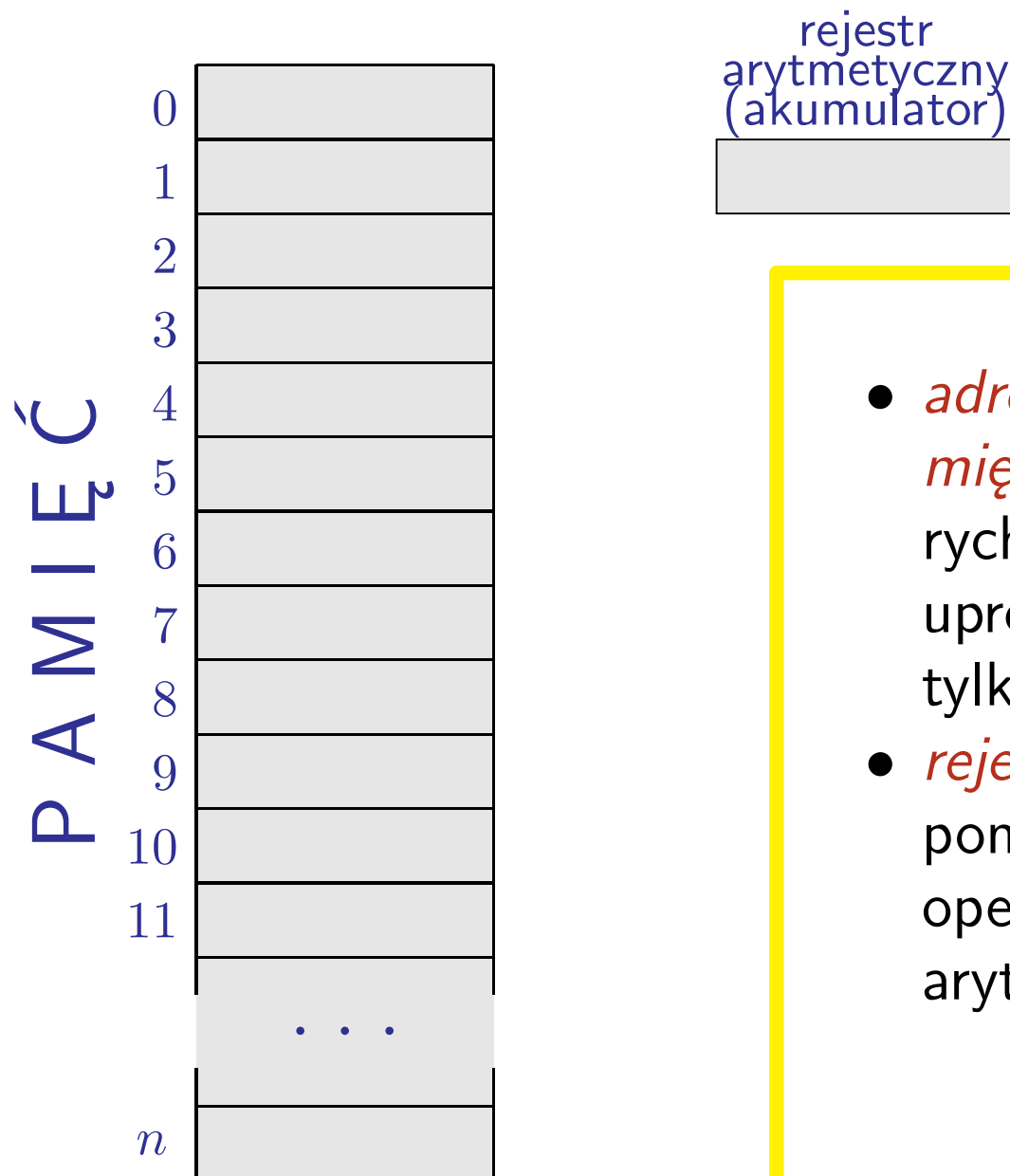
- *adresowana* (czyli numerowana) *pa-mięć* złożona z *komórek*, z których każda może pomieścić bajt

Programowanie niskiego poziomu



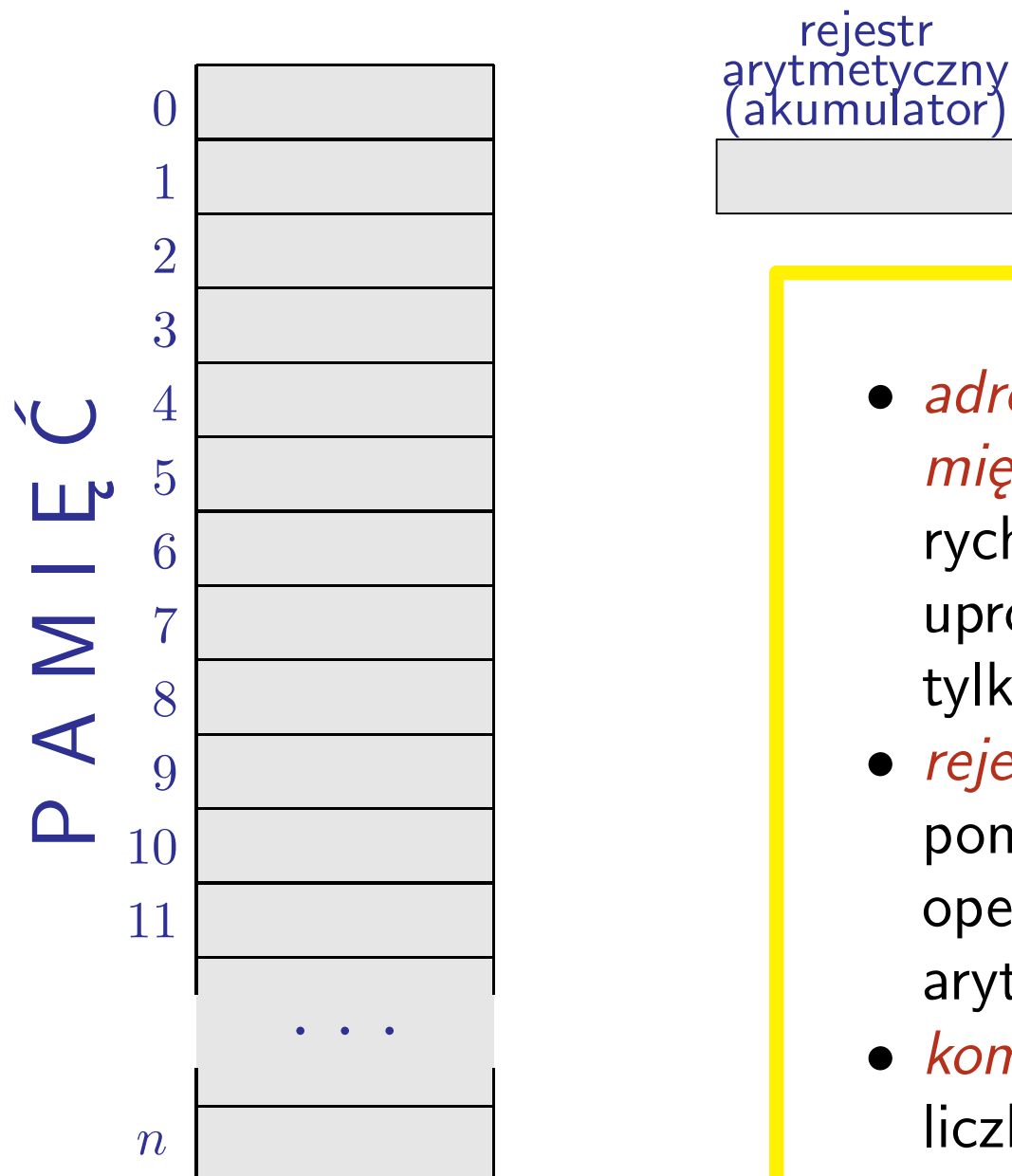
- *adresowana* (czyli numerowana) *pa-mięć* złożona z *komórek*, z których każda może pomieścić bajt; dla uproszczenia zakładam, że nie bajt tylko liczbę całkowitą

Programowanie niskiego poziomu



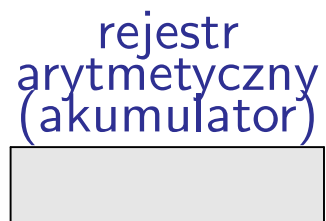
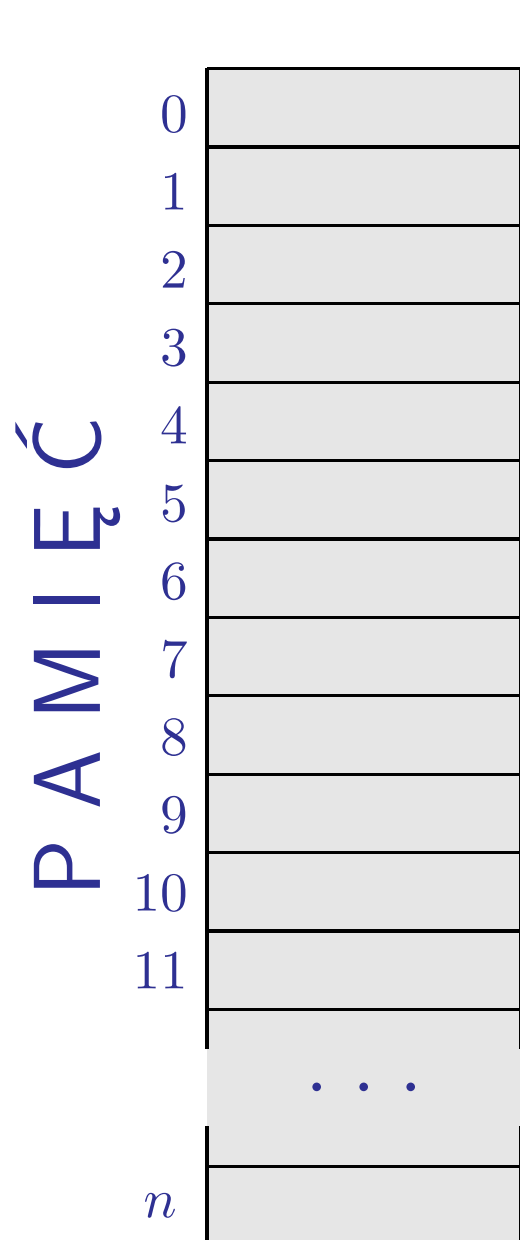
- *adresowana* (czyli numerowana) *pamięć* złożona z *komórek*, z których każda może pomieścić bajt; dla uproszczenia zakładam, że nie bajt tylko liczbę całkowitą;
- *rejestr arytmetyczny* także mogący pomieścić liczbę całkowitą; większość operacji odbywa się między rejestrem arytmetycznym a komórką pamięci

Programowanie niskiego poziomu



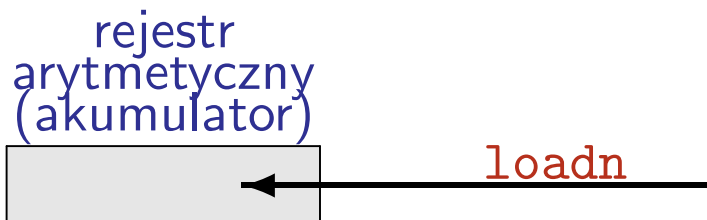
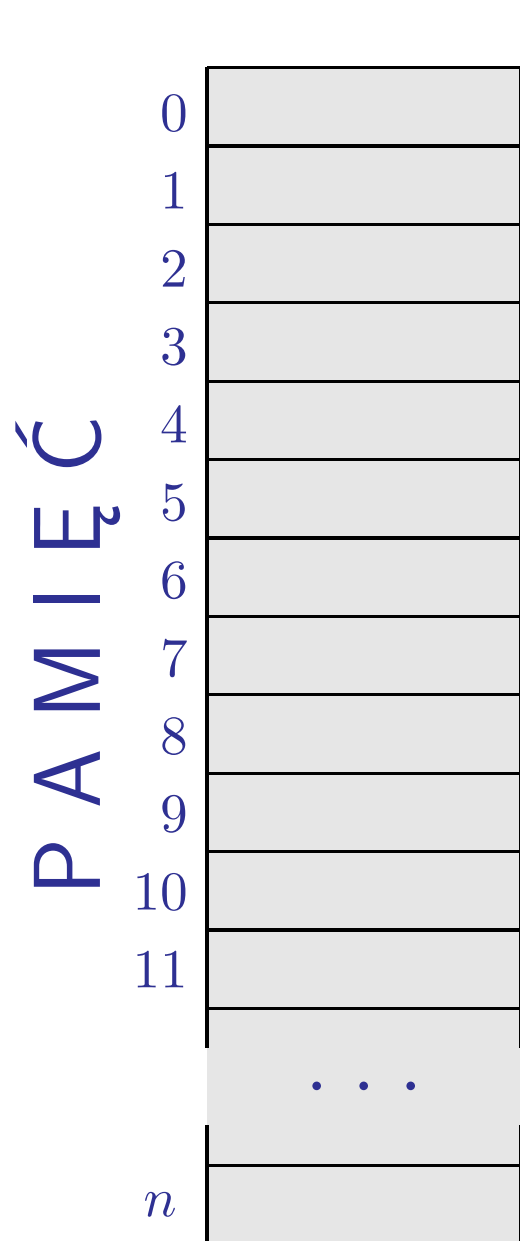
- *adresowana* (czyli numerowana) *pamięć* złożona z *komórek*, z których każda może pomieścić bajt; dla uproszczenia zakładam, że nie bajt tylko liczbę całkowitą;
- *rejestr arytmetyczny* także mogący pomieścić liczbę całkowitą; większość operacji odbywa się między rejestrem arytmetycznym a komórką pamięci;
- *komendy* (patrz niżej) są kodowane liczbami całkowitymi i wpisywane do komórek pamięci.

Programowanie niskiego poziomu



KOMENDY PRZESYŁANIA

Programowanie niskiego poziomu

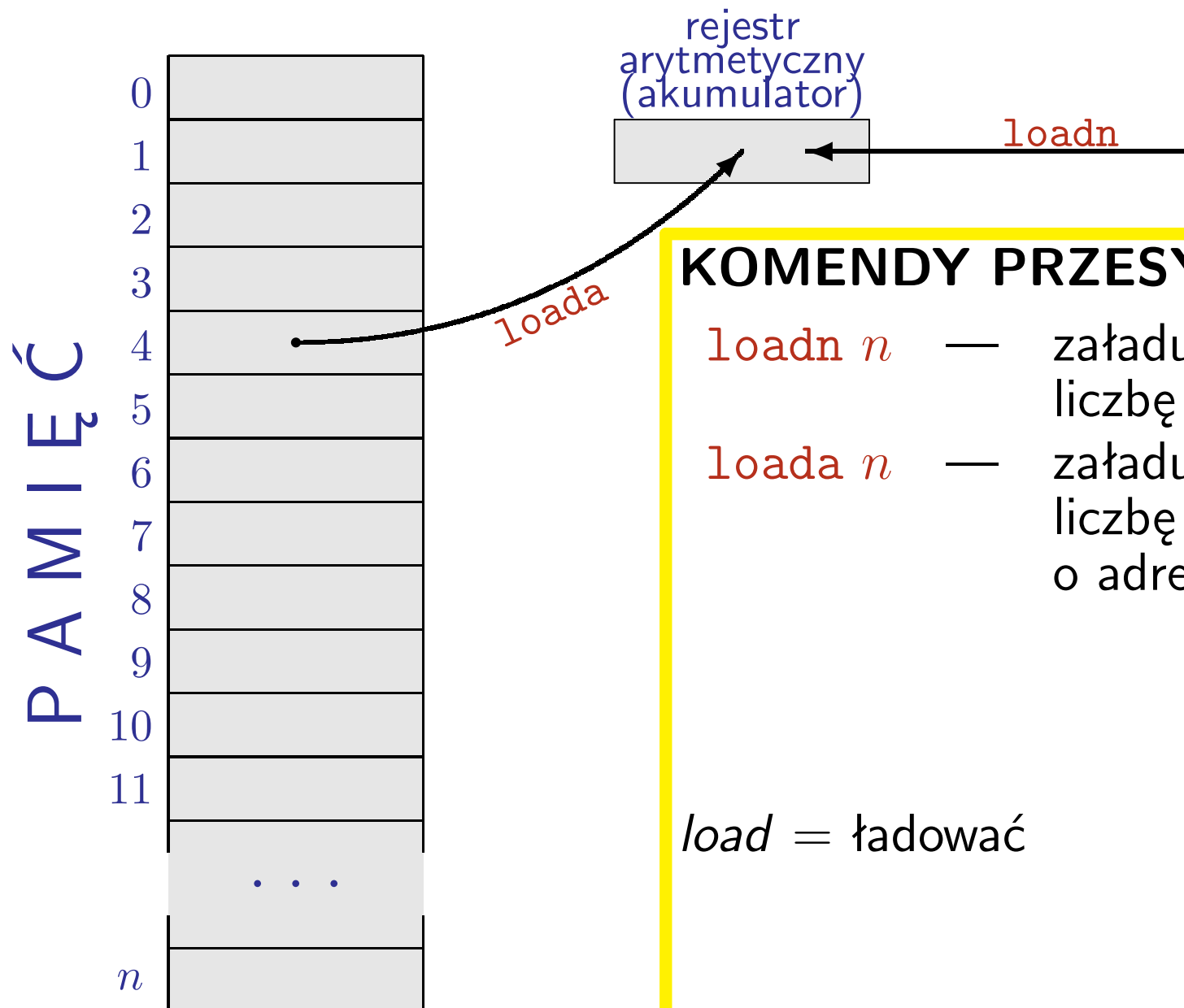


KOMENDY PRZESYŁANIA:

`loadn n` — załaduj do akumulatora liczbę *n*

load = ładować

Programowanie niskiego poziomu



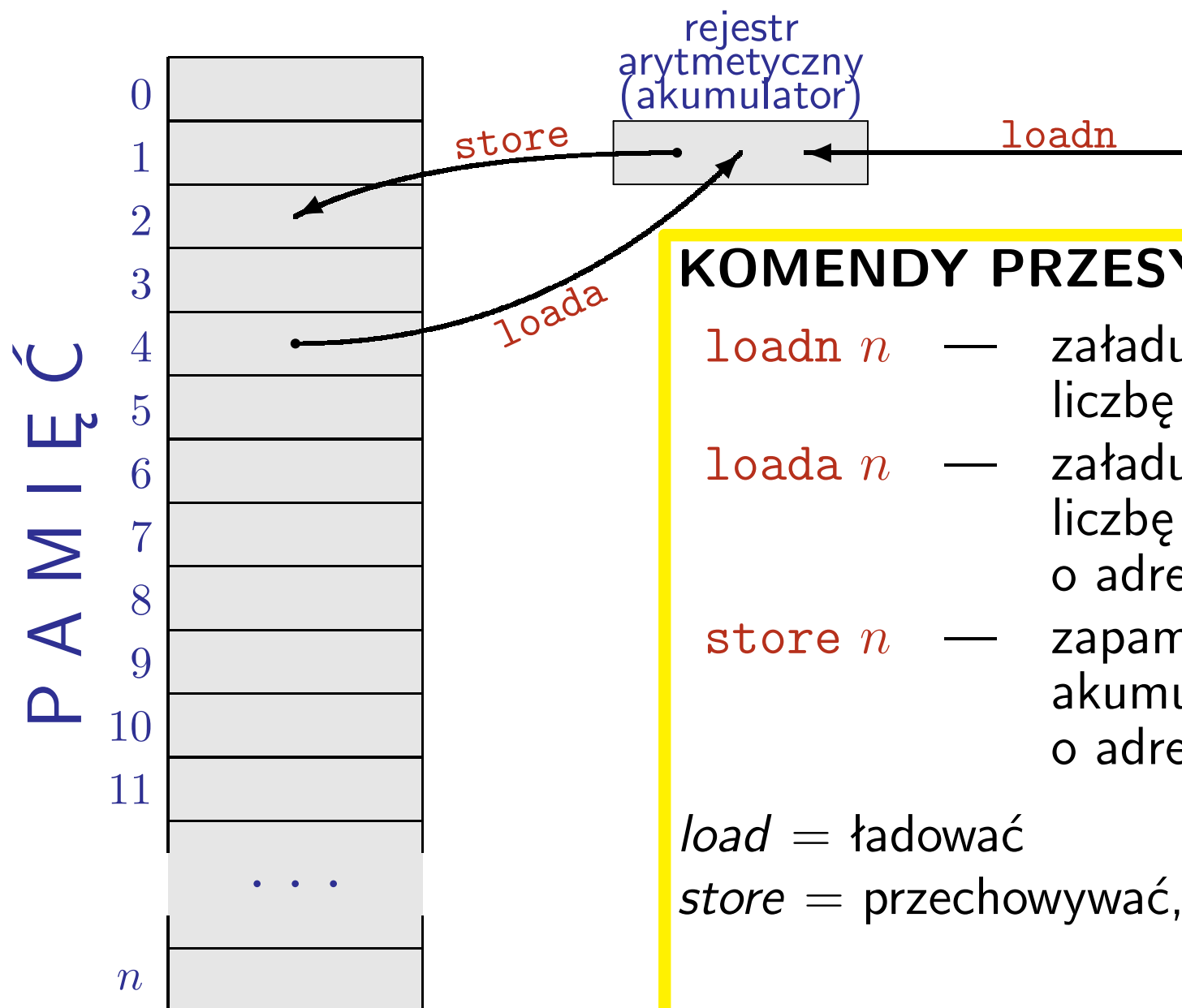
KOMENDY PRZESYŁANIA:

loadn n — załaduj do akumulatora liczbę n

loada n — załaduj do akumulatora liczbę z komórki o adresie n

load = ładować

Programowanie niskiego poziomu



KOMENDY PRZESYŁANIA:

loadn n — załaduj do akumulatora liczbę n

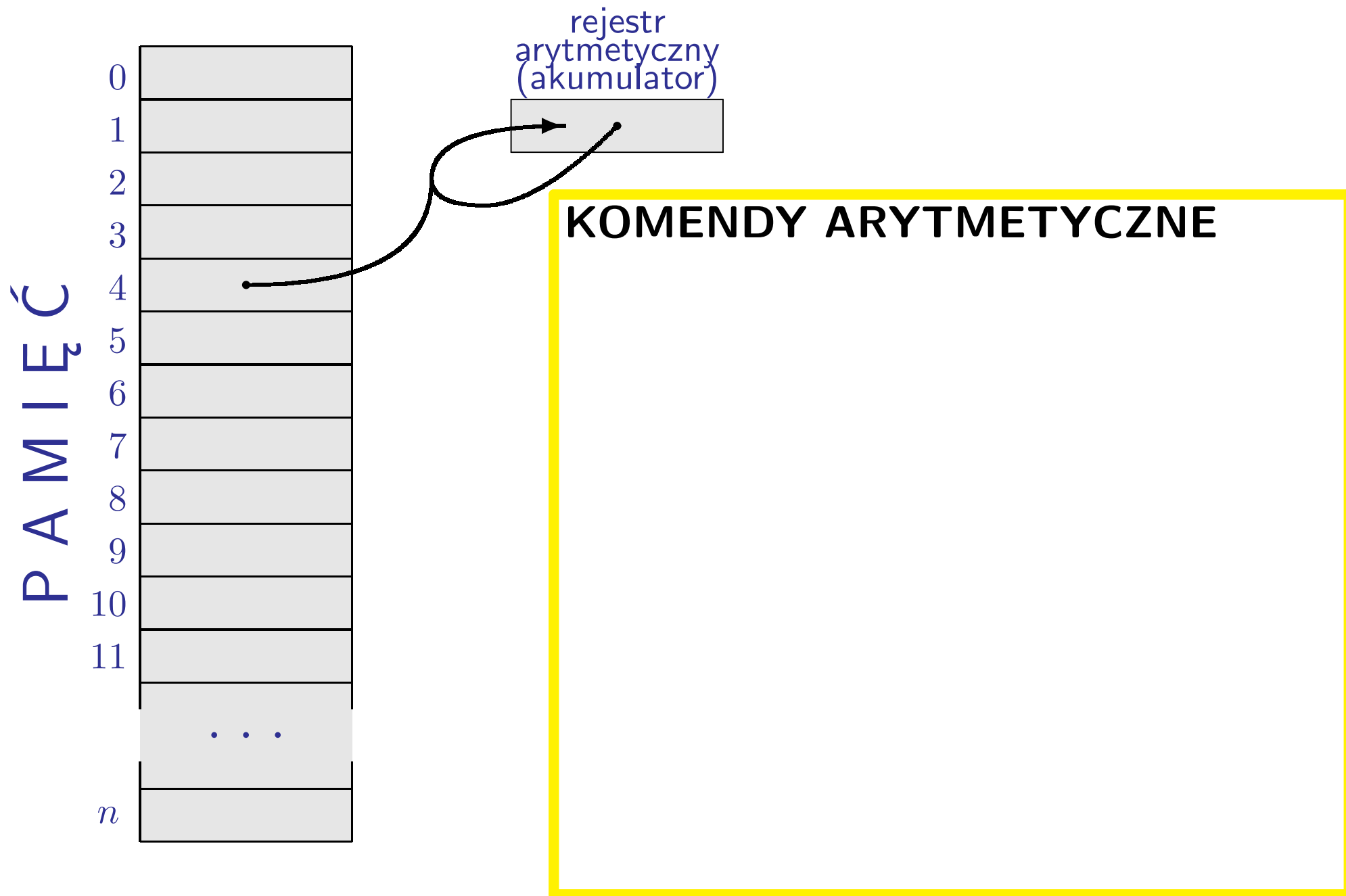
loada n — załaduj do akumulatora liczbę z komórki o adresie n

store n — zapamiętaj zawartość akumulatora w kom. o adresie n

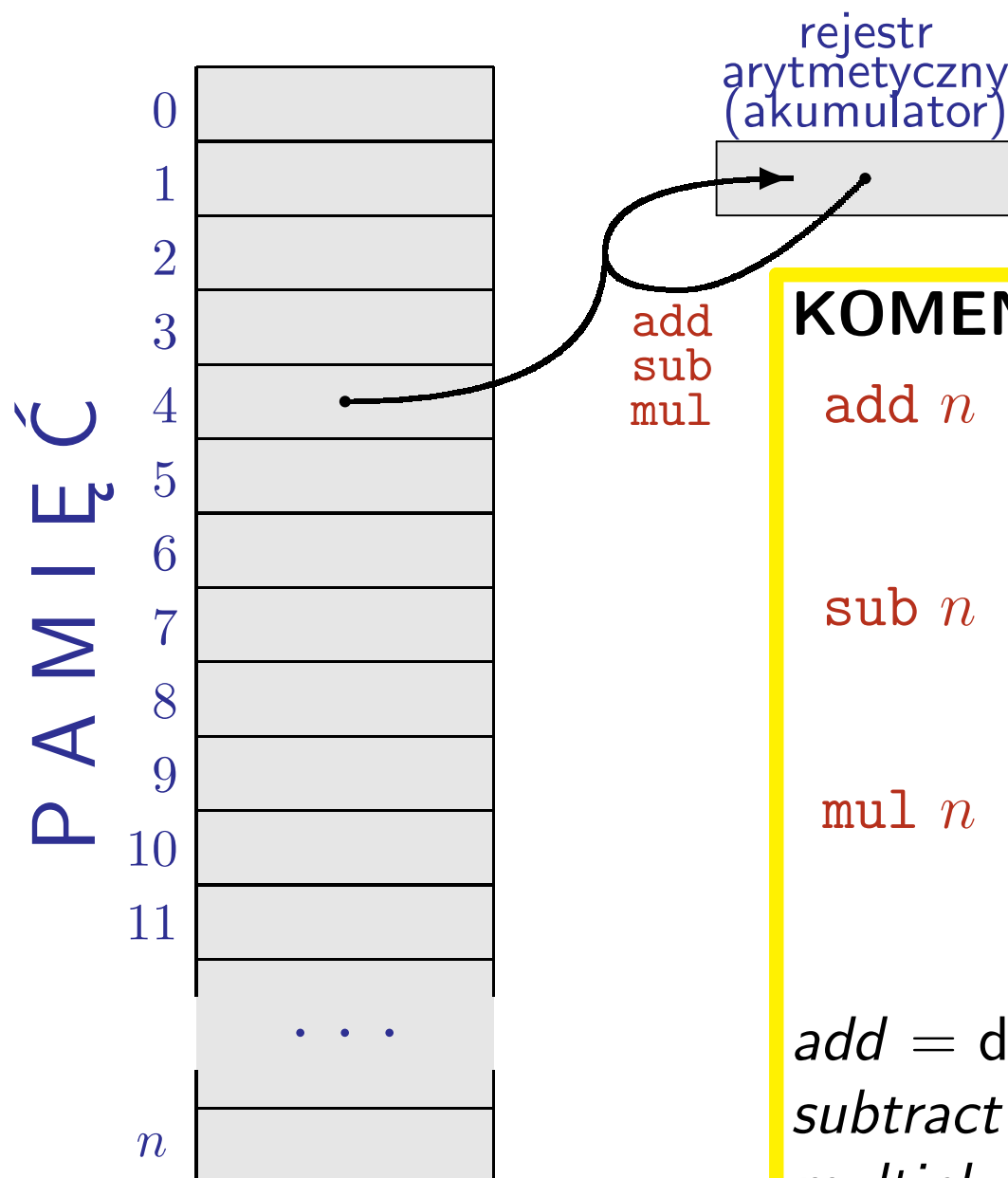
load = ładować

store = przechowywać, magazynować

Programowanie niskiego poziomu



Programowanie niskiego poziomu



KOMENDY ARYTMETYCZNE:

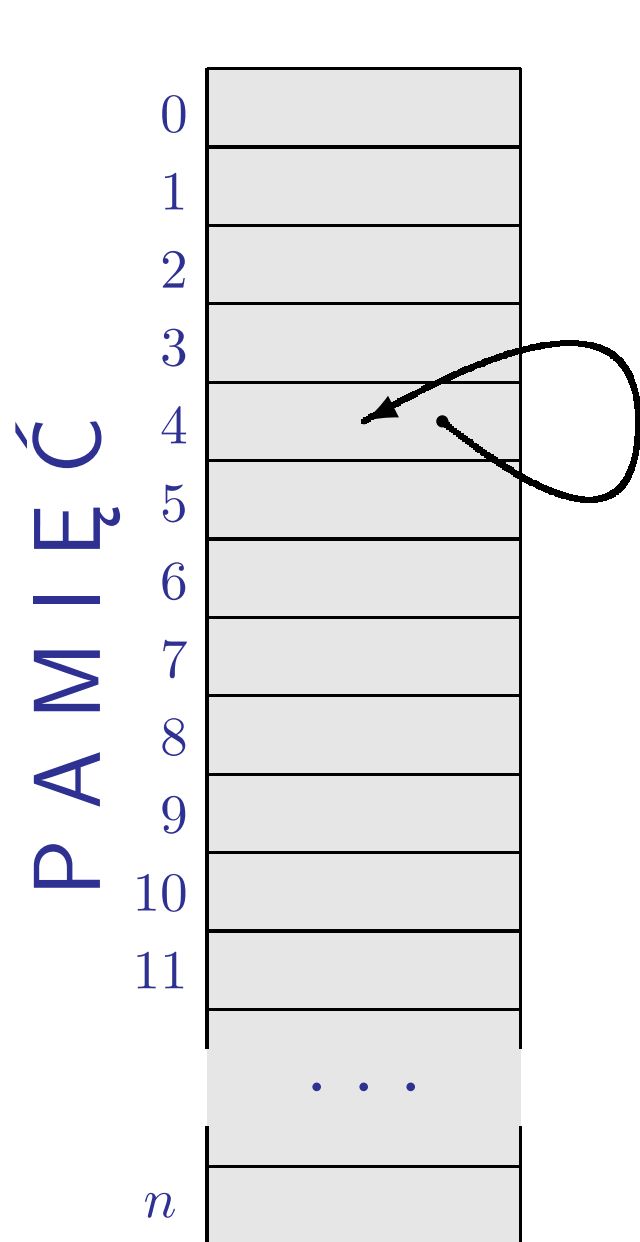
- add** n — dodaj do akumulatora liczbę z kom. o adresie n ; wynik w akumulatorze
- sub** n — odejmij od akumulatora liczbę z kom. o adresie n ; wynik w akumulatorze
- mul** n — pomnóż akumulator przez liczbę z kom. o adresie n ; wynik w akumulatorze

add = dodawać

subtract = odejmować

multiply = mnożyć

Programowanie niskiego poziomu

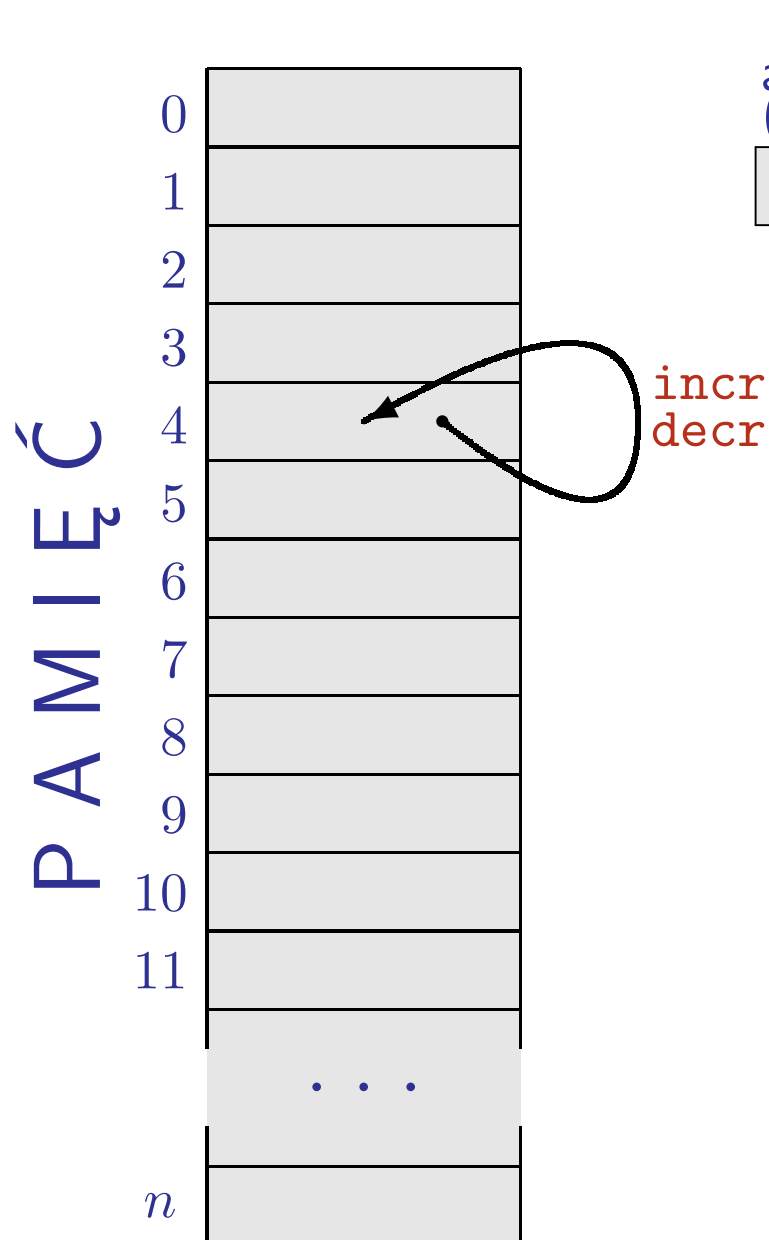


rejestr
arytmetyczny
(akumulator)



**KOMENDY DZIAŁAŃ
NA KOMÓRKACH**

Programowanie niskiego poziomu



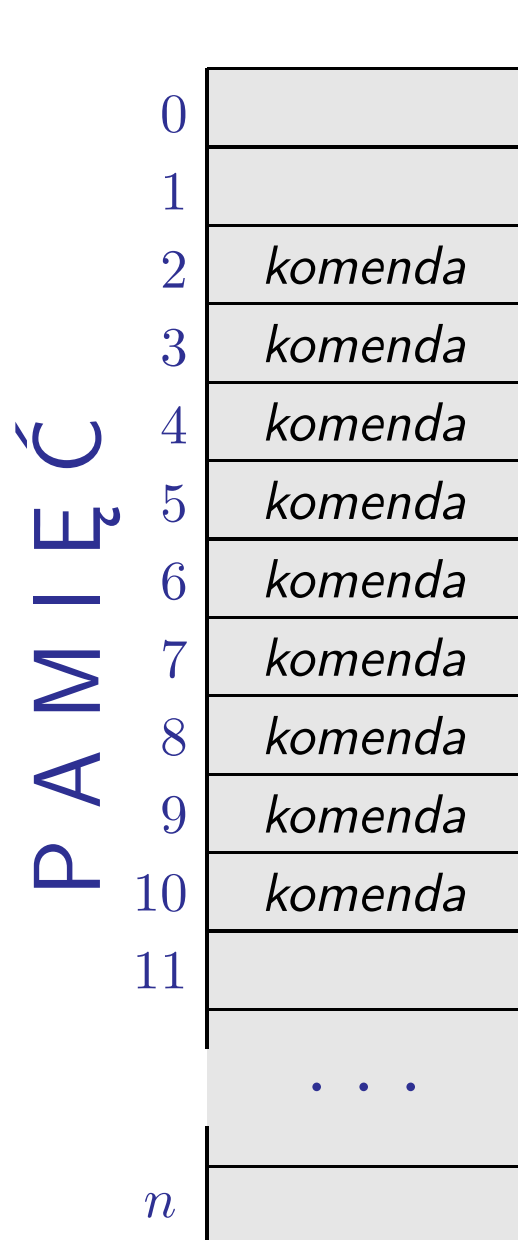
KOMENDY DZIAŁAŃ NA KOMÓRKACH:

- incr* n — powiększ o 1 zawartość kom. o adresie n
- decr* n — zmniejsz o 1 zawartość kom. o adresie n

increment = zwiększać

decrement = zmniejszać

Programowanie niskiego poziomu



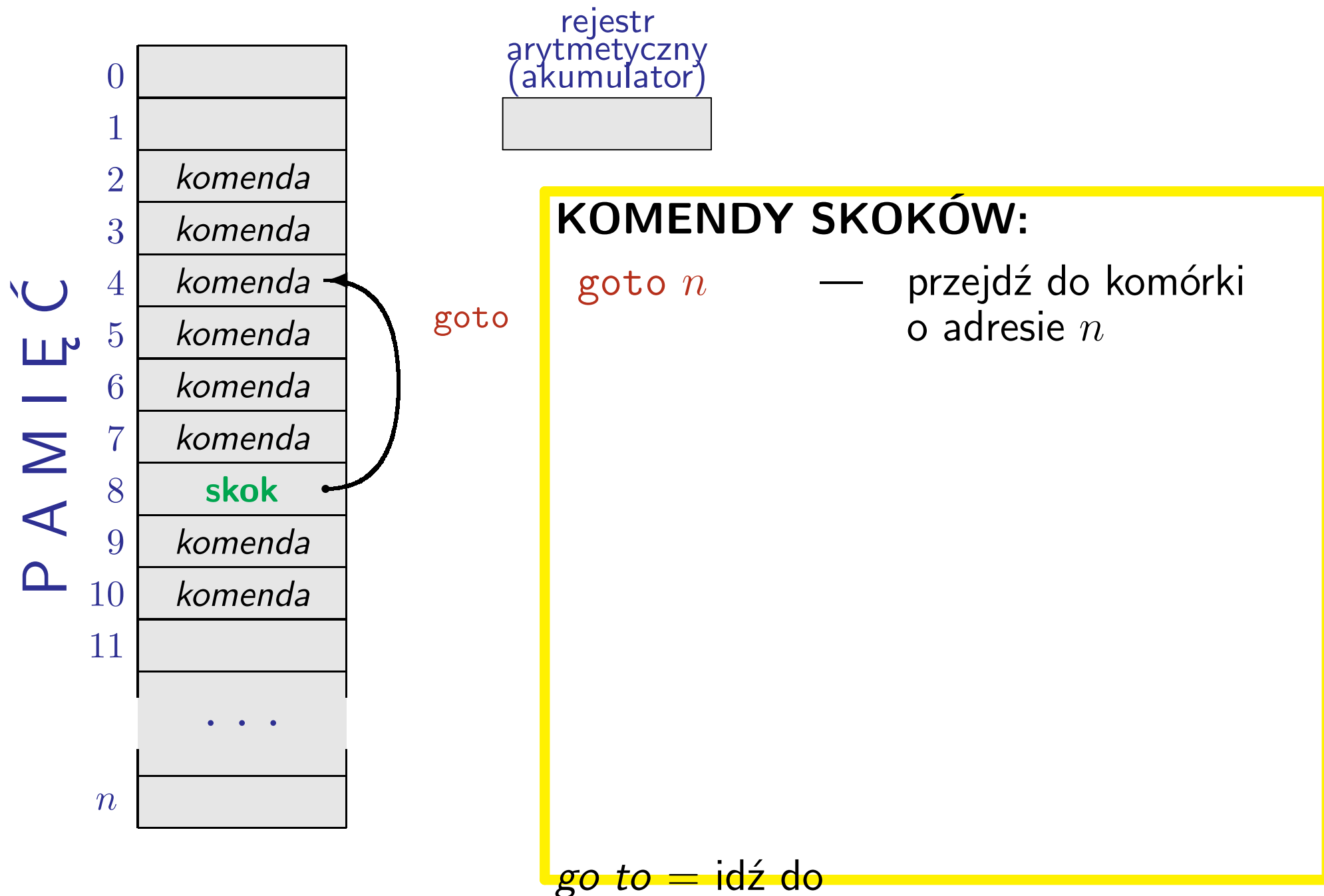
rejestr
arytmetyczny
(akumulator)



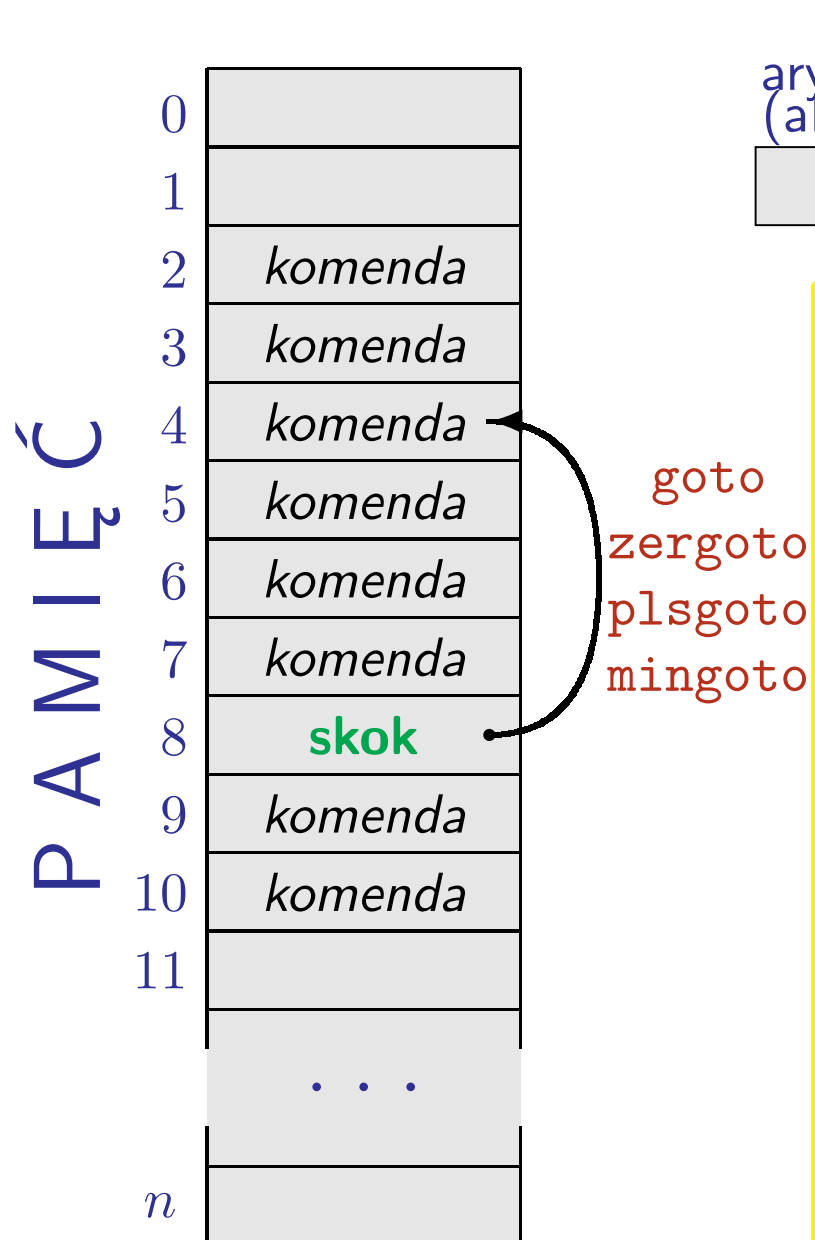
KOMENDY SKOKÓW

2010 = 10710

Programowanie niskiego poziomu



Programowanie niskiego poziomu

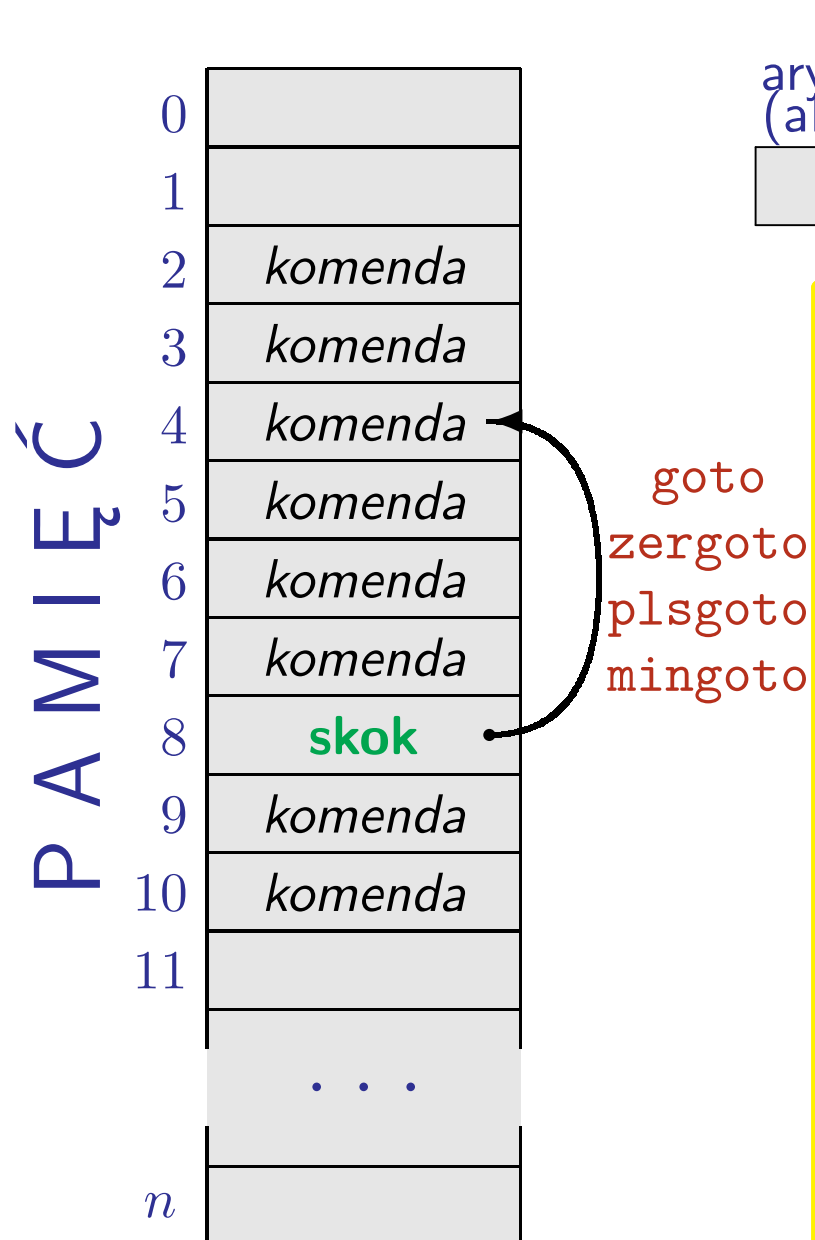


KOMENDY SKOKÓW:

- goto *n*** — przejdź do komórki o adresie *n*
- zergoto *n*** — jeśli w akumul. zero, to przejdź do kom. o adresie *n*
- plsgoto *n*** — jeśli akumul. dodatni, to przejdź do kom. o adresie *n*
- mingoto *n*** — jeśli akumul. ujemny, to przejdź do kom. o adresie *n*

go to = idź do

Programowanie niskiego poziomu

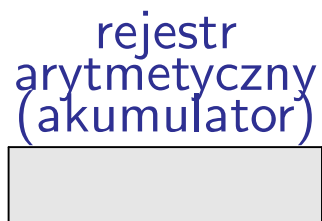
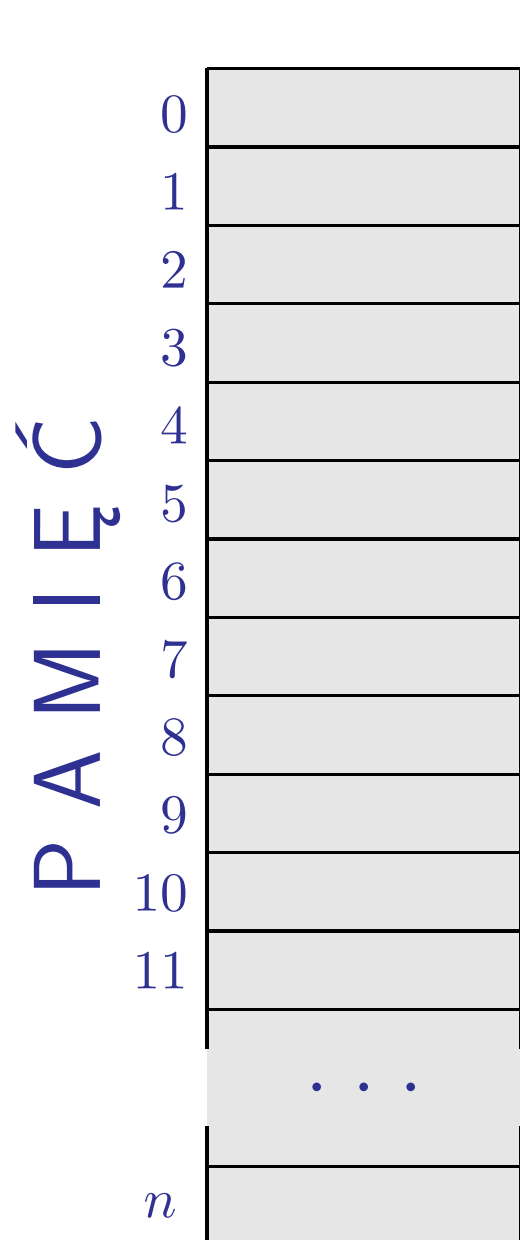


KOMENDY SKOKÓW:

- | | | |
|------------------|---|--|
| <i>goto n</i> | — | przejdź do komórki o adresie <i>n</i> |
| <i>zergoto n</i> | — | jeśli w akumul. zero, to przejdź do kom. o adresie <i>n</i> |
| <i>plsgoto n</i> | — | jeśli akumul. dodatni, to przejdź do kom. o adresie <i>n</i> |
| <i>mingoto n</i> | — | jeśli akumul. ujemny, to przejdź do kom. o adresie <i>n</i> |
| <i>stop</i> | — | zatrzymaj działanie |

go to = idź do

Programowanie niskiego poziomu

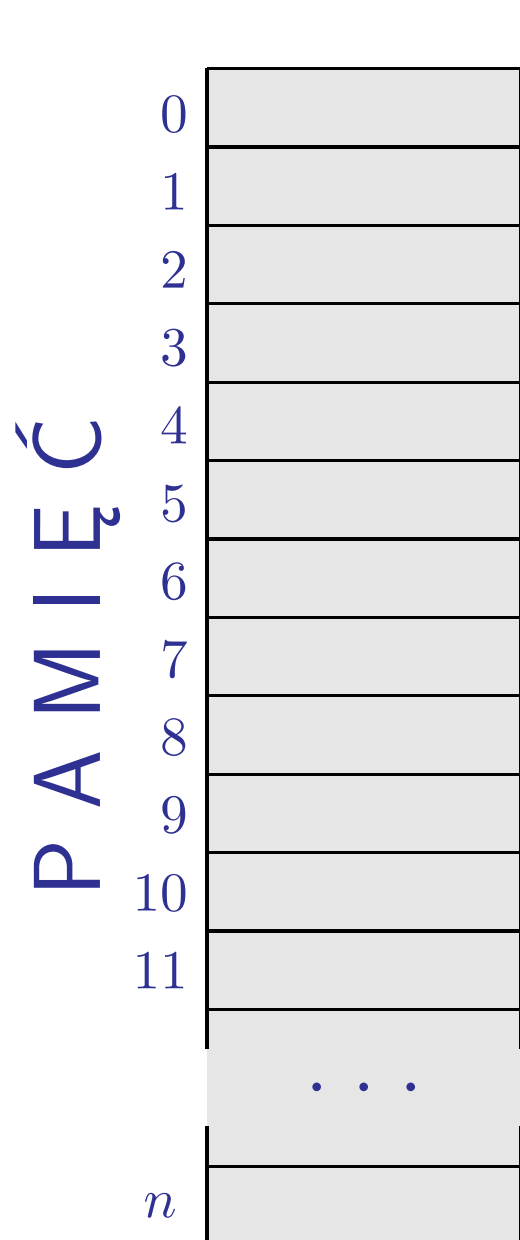


PRZYKŁAD:

Obliczenie wartości wielomianu

$$5x^2 - 3x + 1$$

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

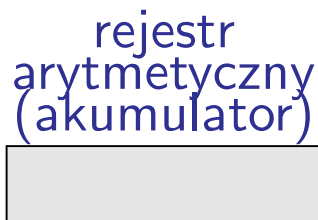
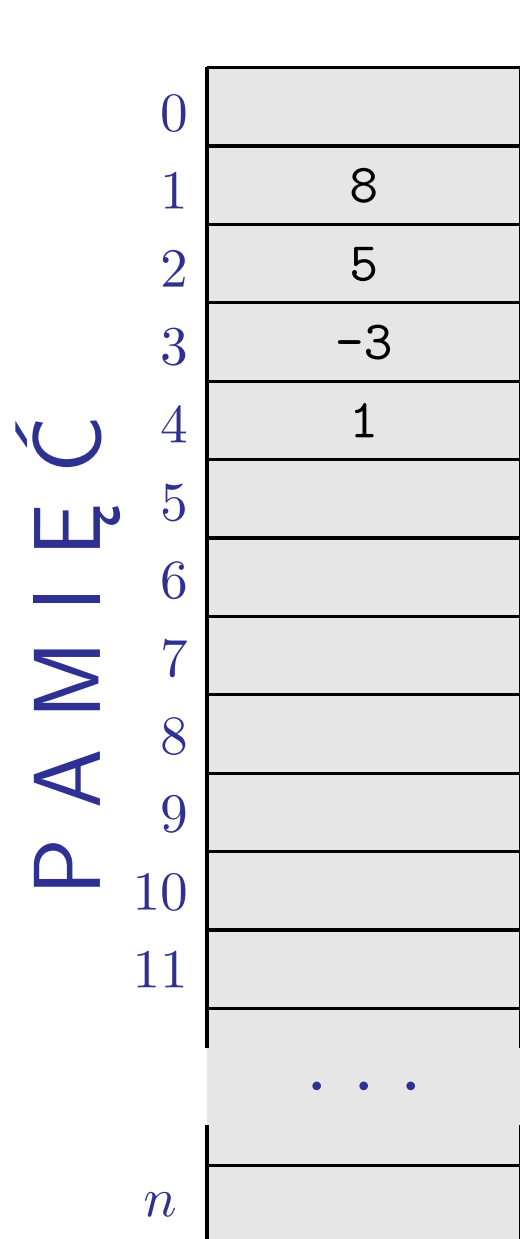


PRZYKŁAD:

Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

Programowanie niskiego poziomu



PRZYKŁAD:

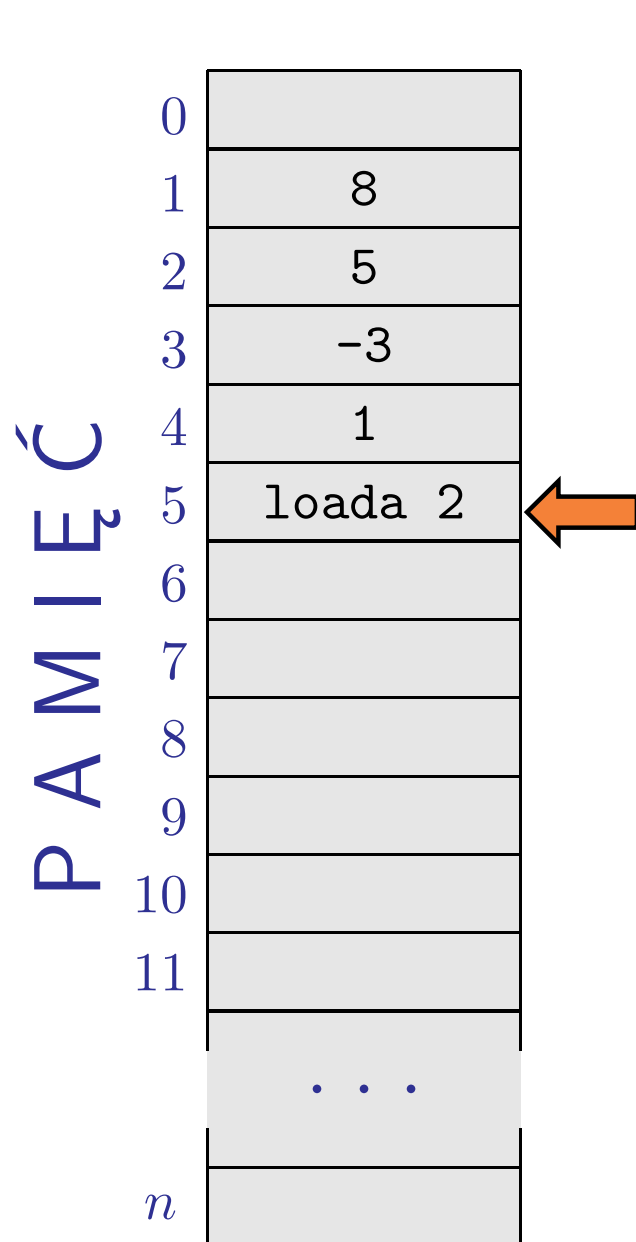
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)



PRZYKŁAD:

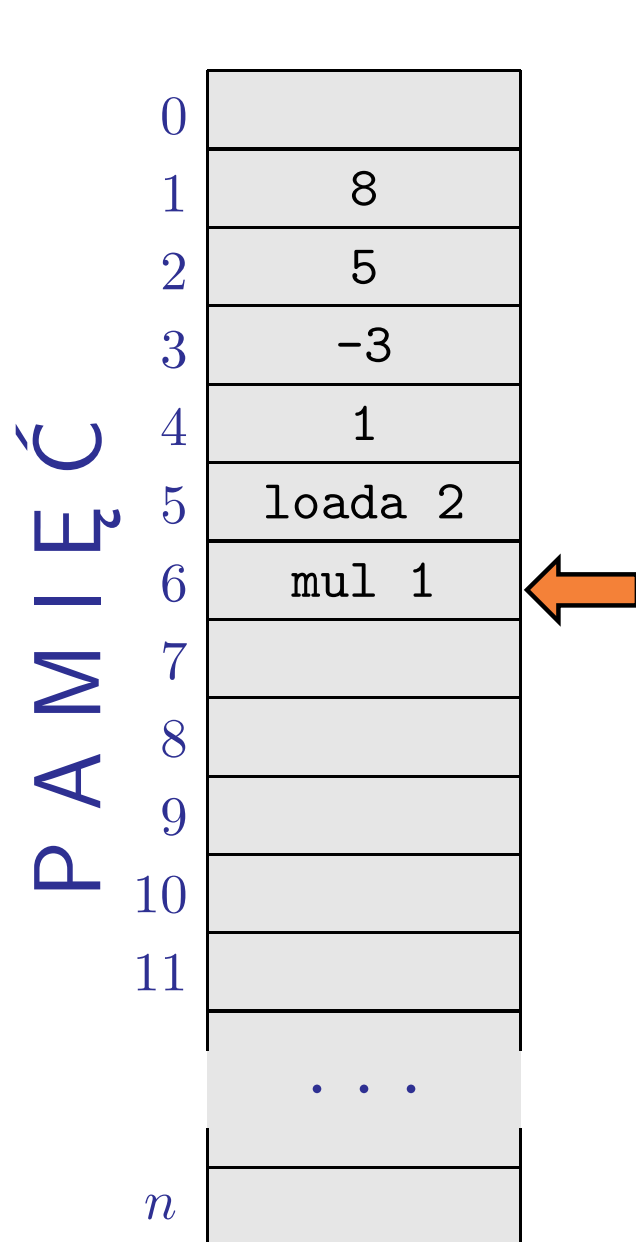
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

5

PRZYKŁAD:

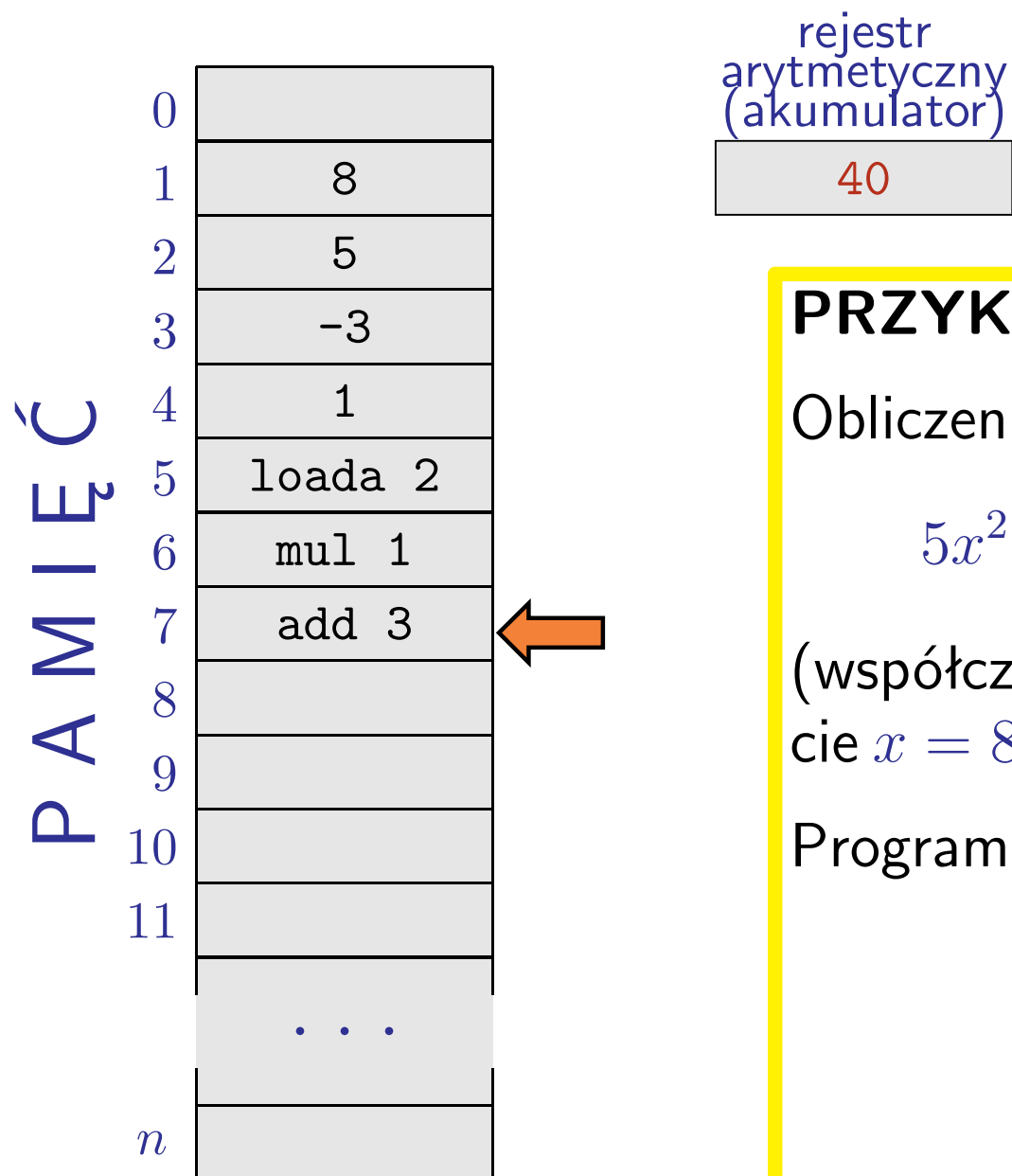
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



PRZYKŁAD:

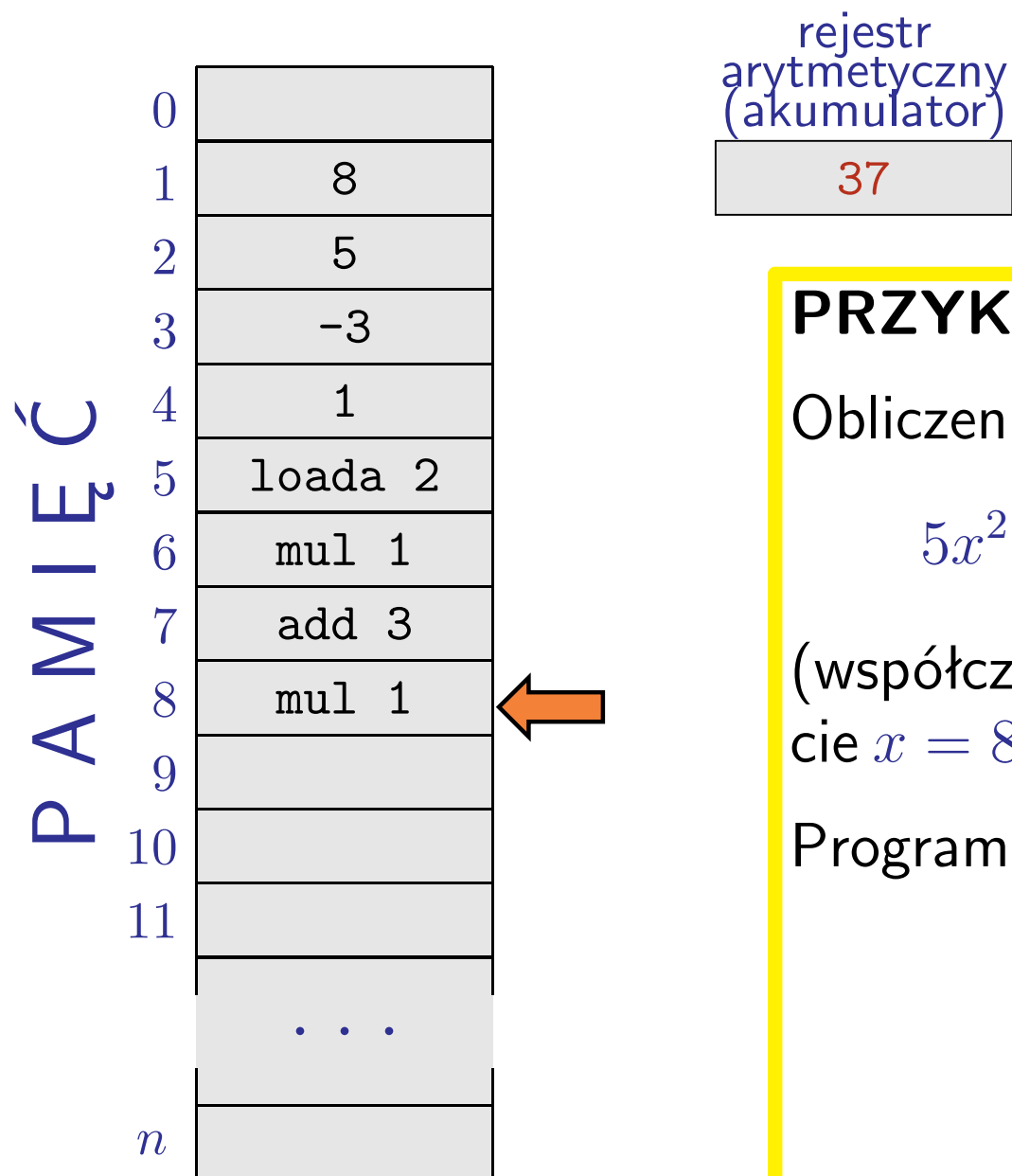
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



PRZYKŁAD:

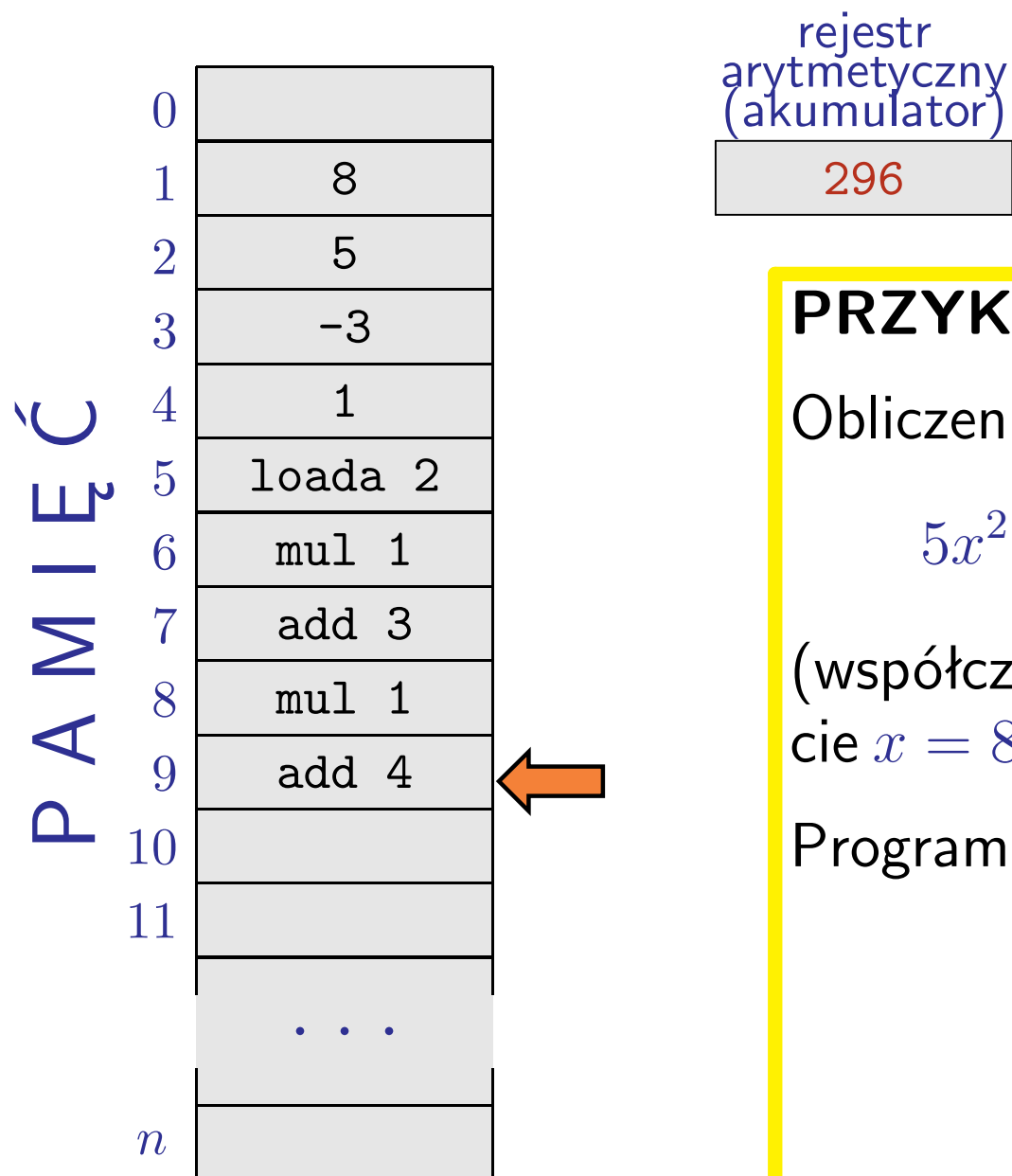
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



PRZYKŁAD:

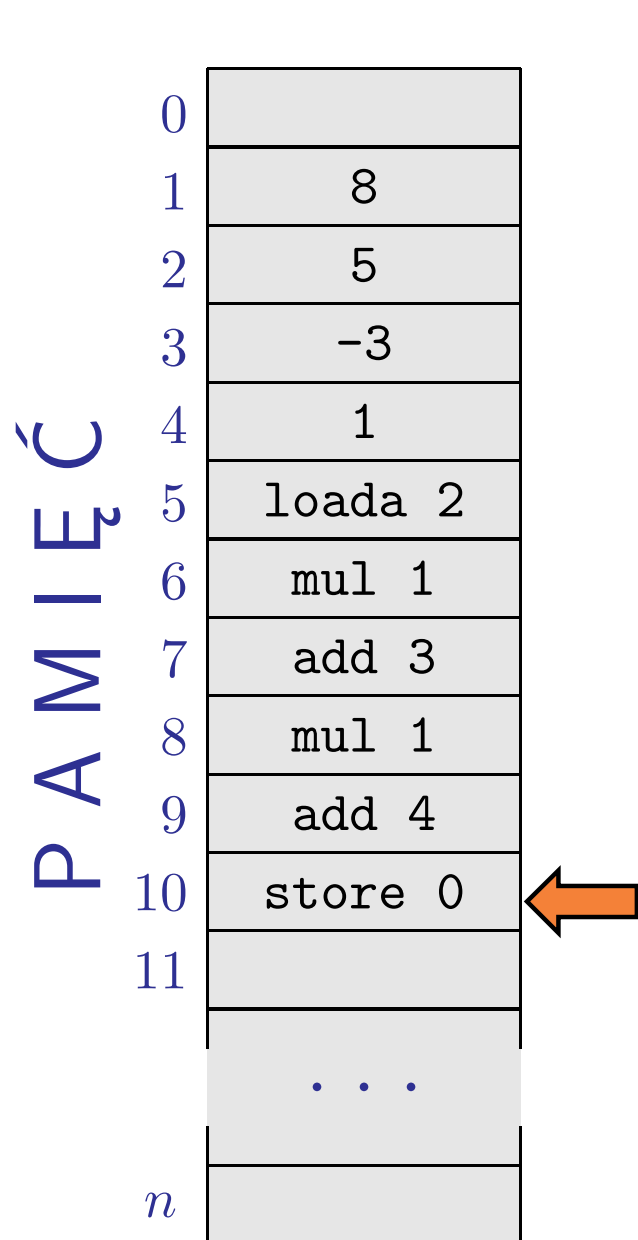
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

297

PRZYKŁAD:

Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu

PAMIĘĆ

0	297
1	8
2	5
3	-3
4	1
5	loada 2
6	mul 1
7	add 3
8	mul 1
9	add 4
10	store 0
11	stop
	...
<i>n</i>	



rejestr
arytmetyczny
(akumulator)

297

PRZYKŁAD:

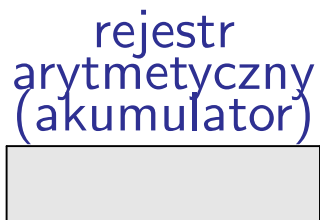
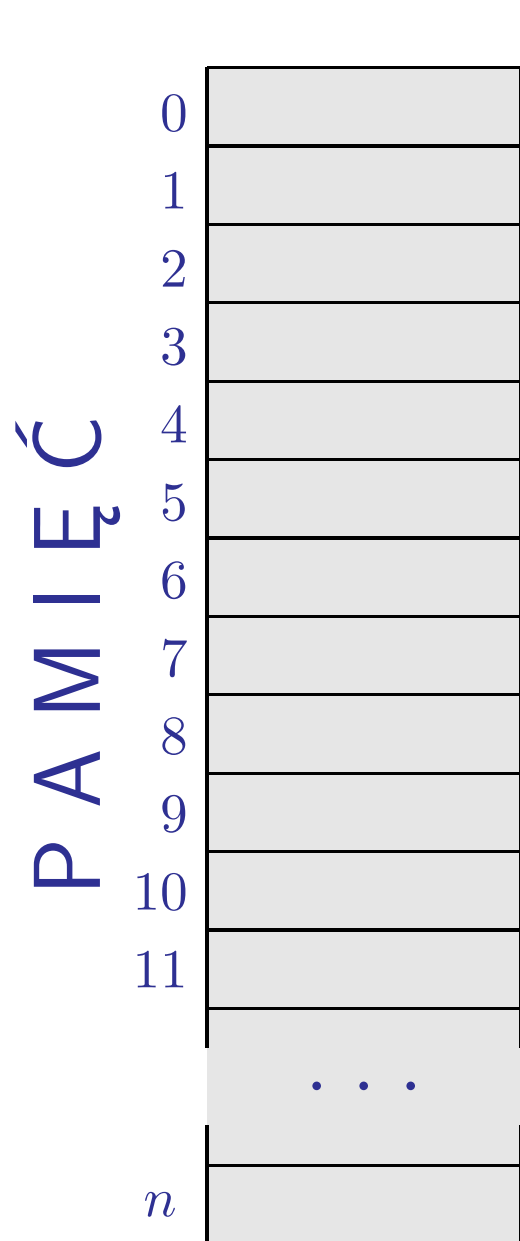
Obliczenie wartości wielomianu

$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(współczynniki w komórkach 2–4) w punkcie $x = 8$ (komórka 1); wynik w komórce 0.

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu

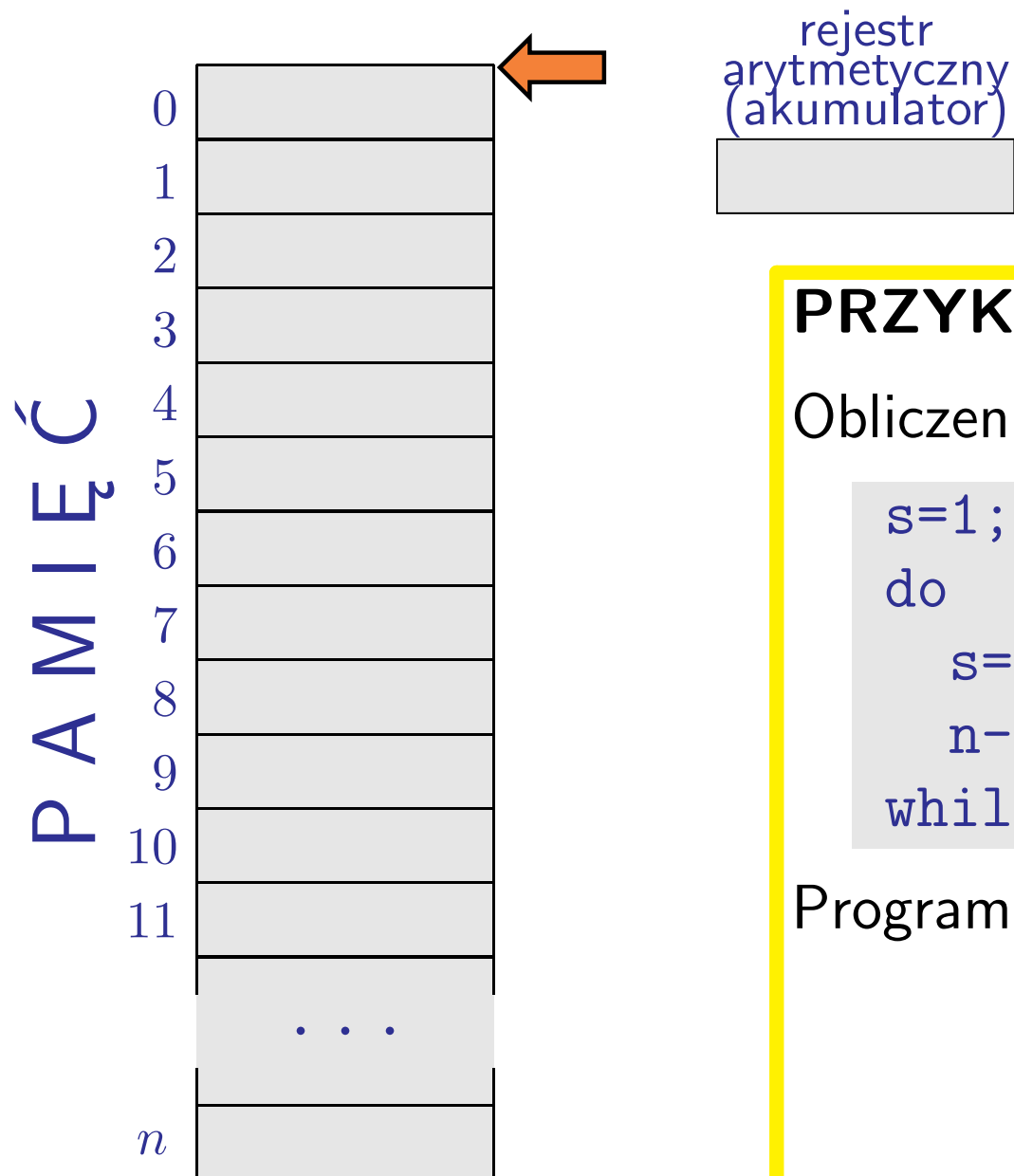


PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;  
do  
    s=s*n;  
    n--;  
while (n>0);
```

Programowanie niskiego poziomu



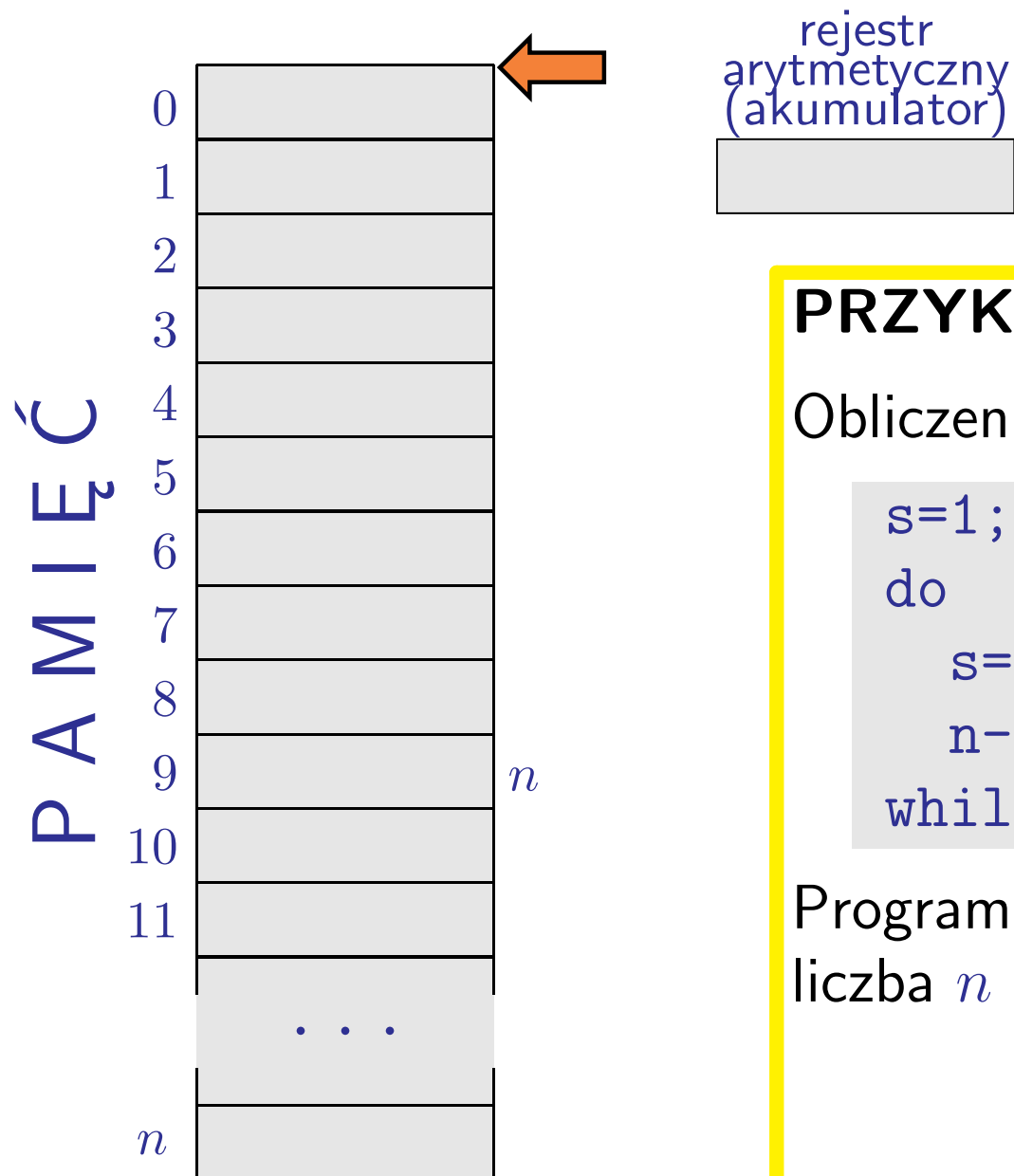
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;  
do  
    s=s*n;  
    n--;  
while (n>0);
```

Program należy uruchamiać od komórki 0

Programowanie niskiego poziomu



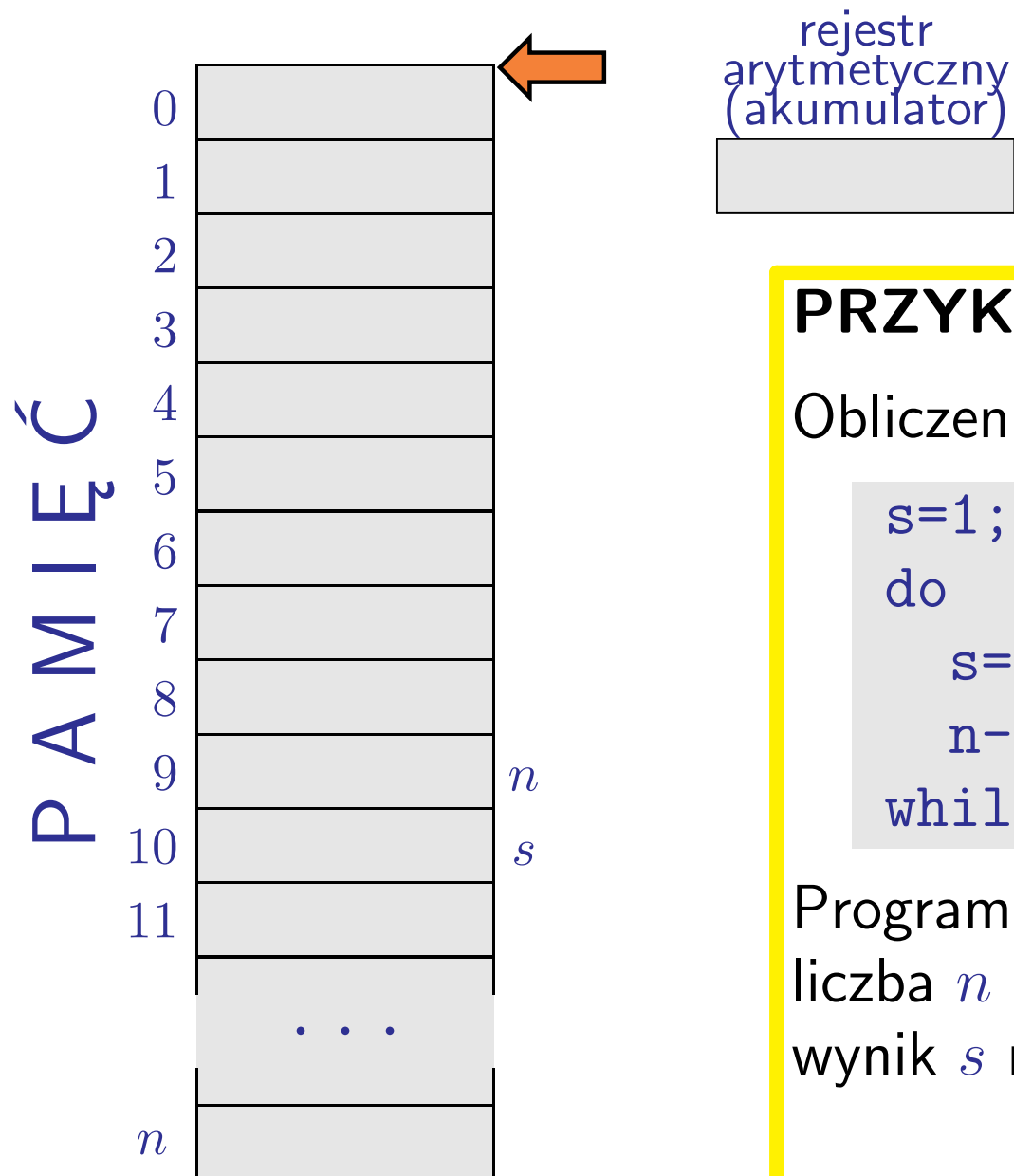
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9

Programowanie niskiego poziomu



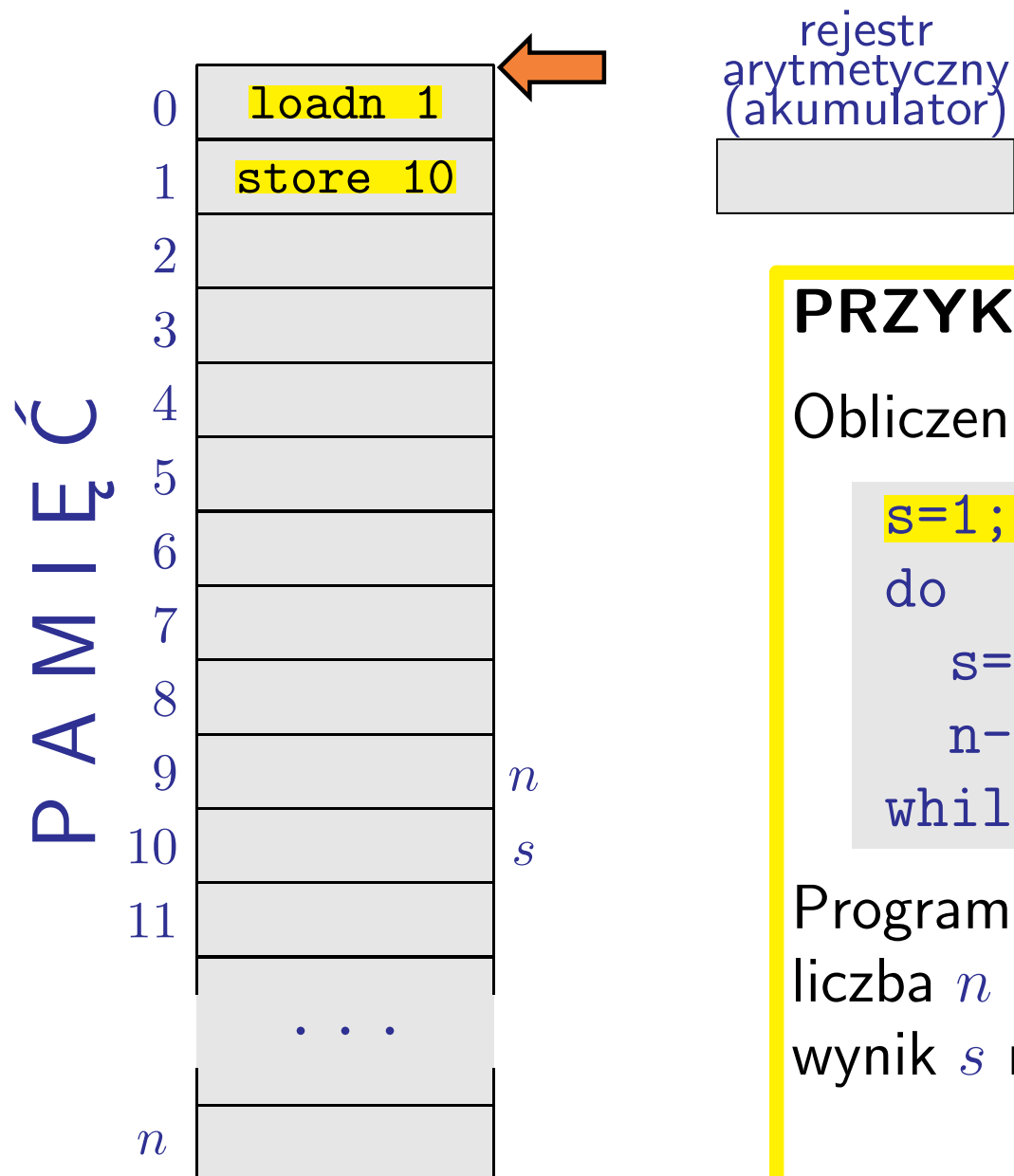
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
 liczba n ma być w kom. 9;
 wynik s ma być w kom. 10.

Programowanie niskiego poziomu



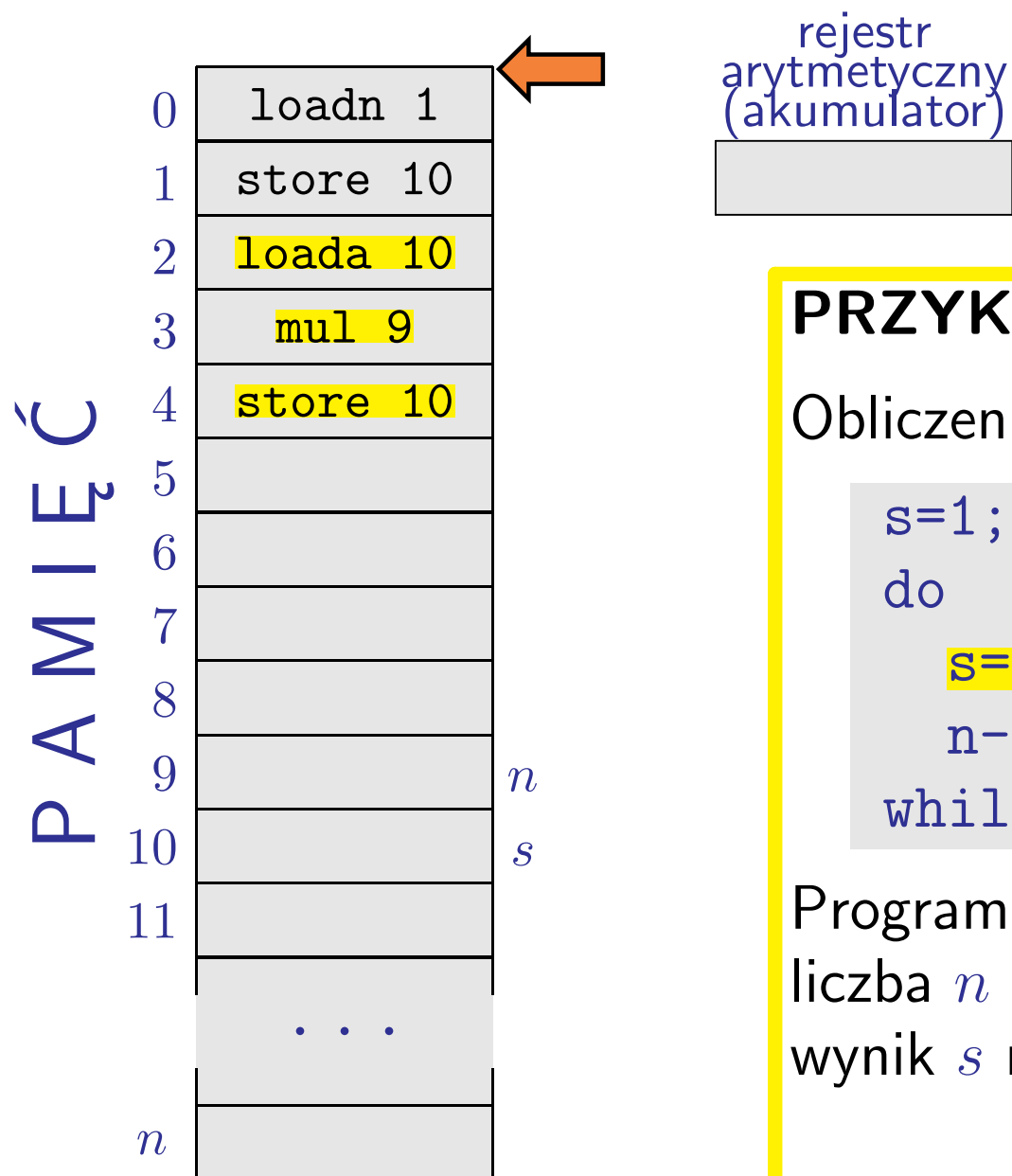
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Programowanie niskiego poziomu



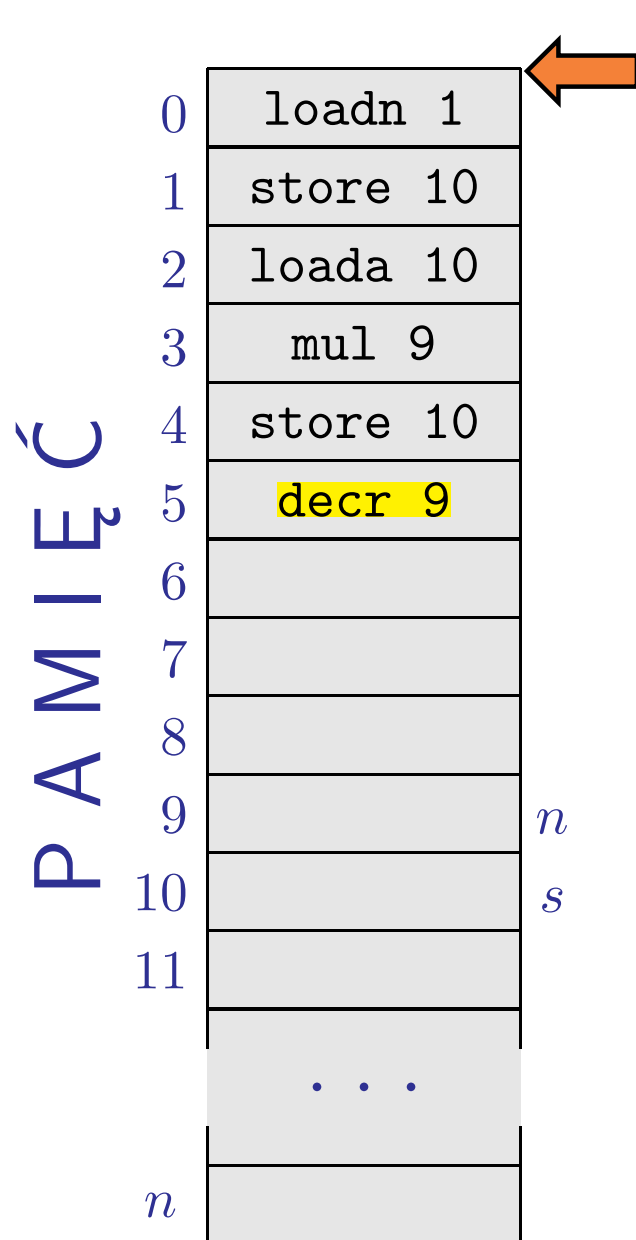
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

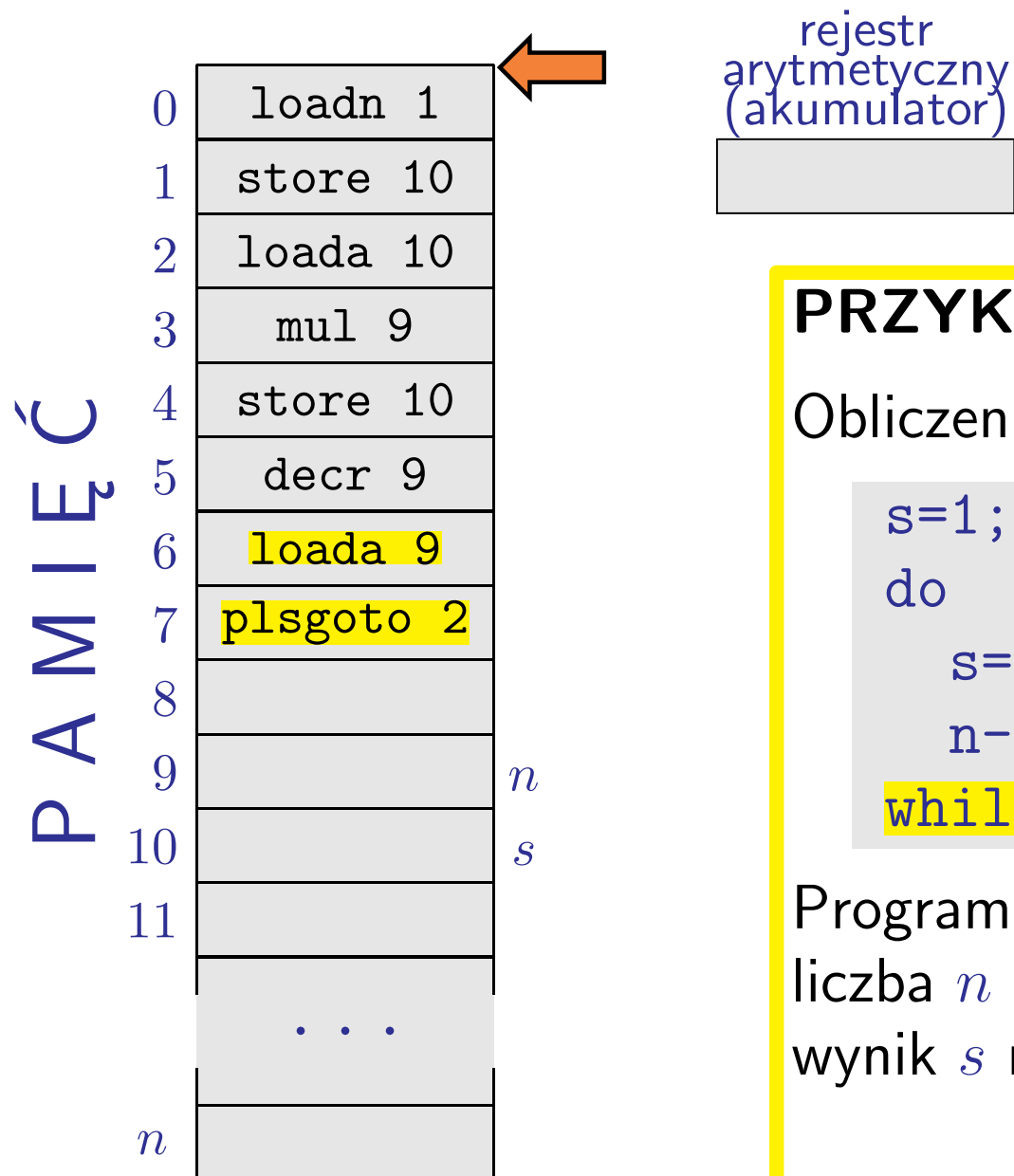
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Programowanie niskiego poziomu



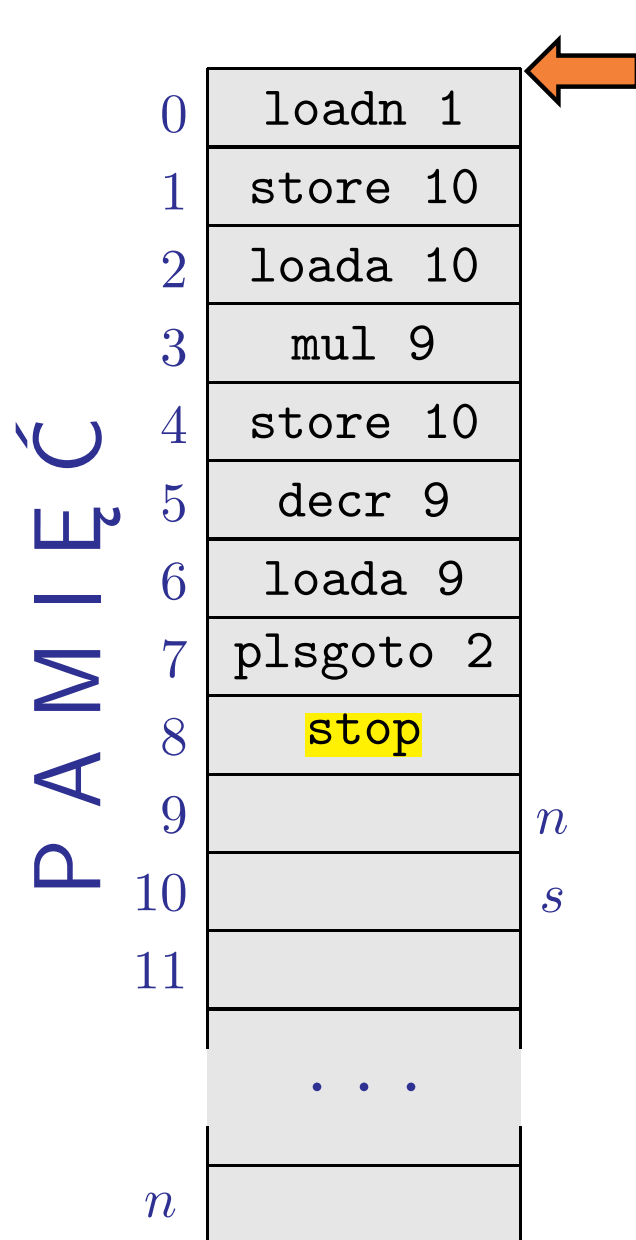
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)



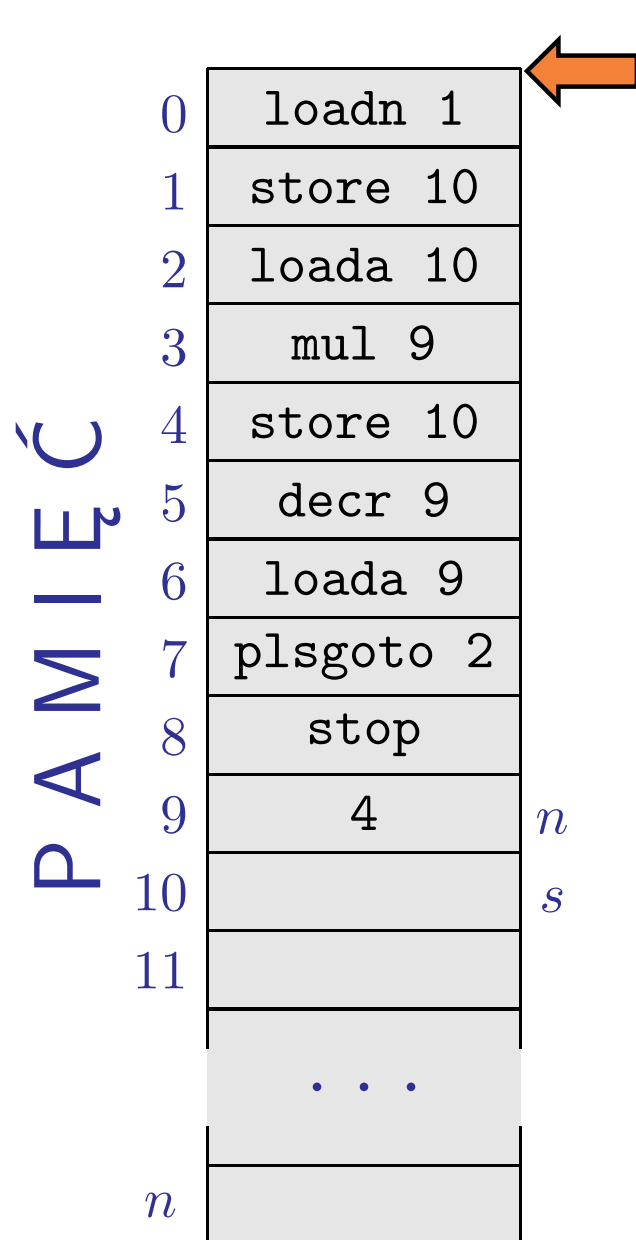
PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)



PRZYKŁAD:

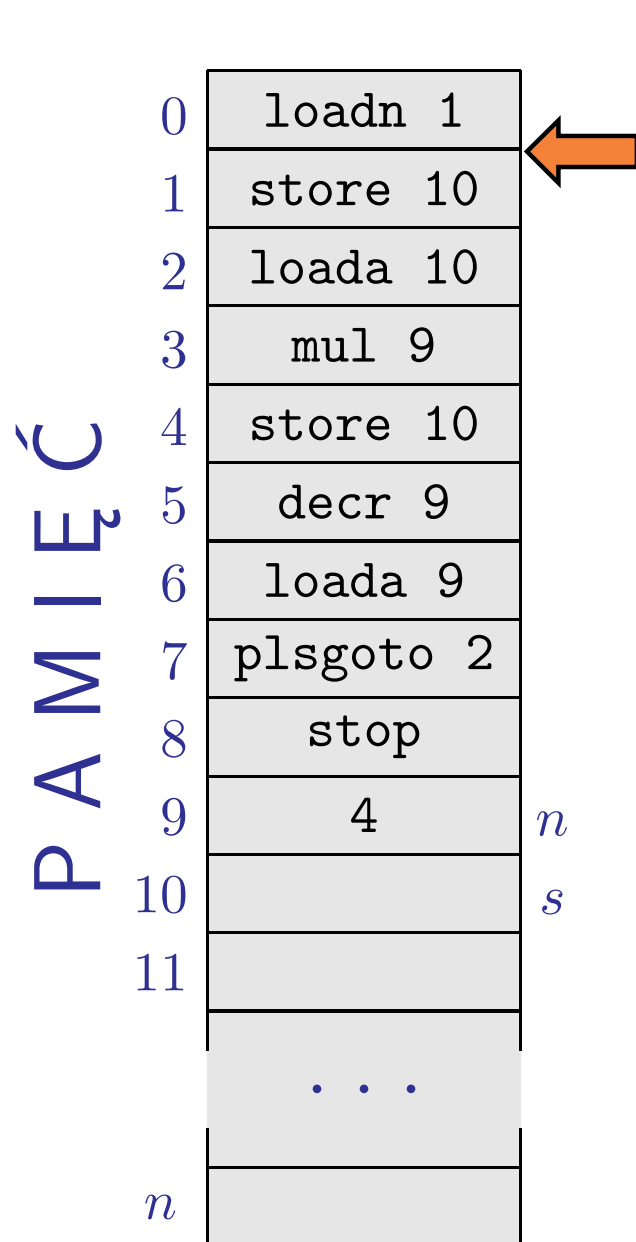
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

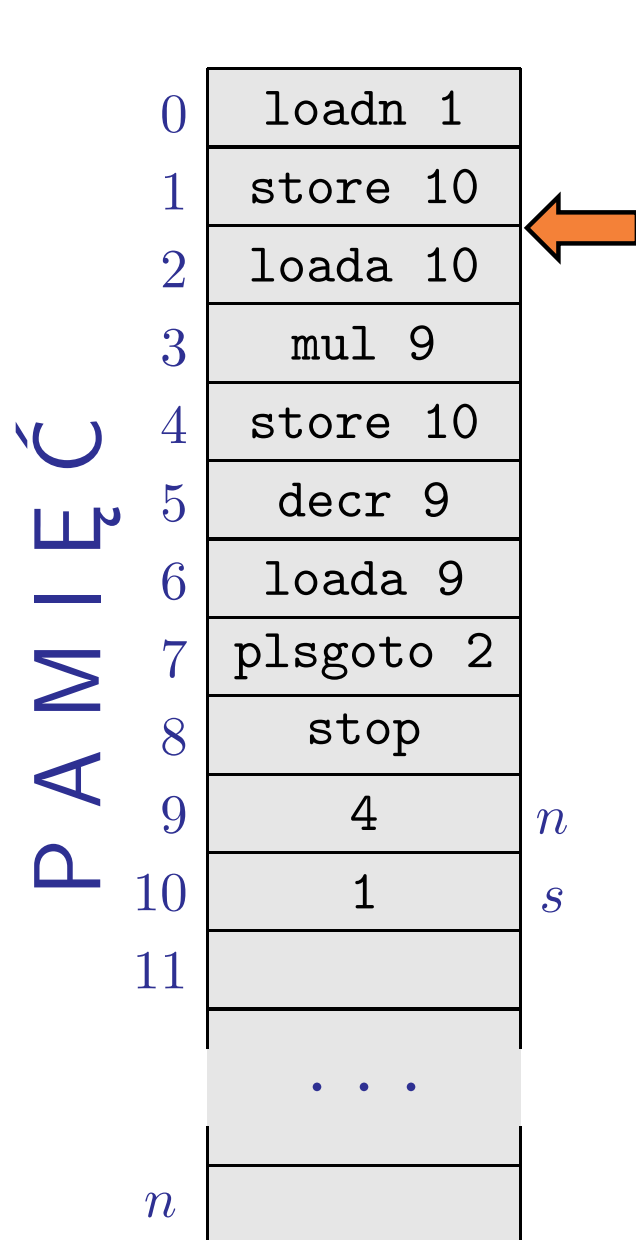
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

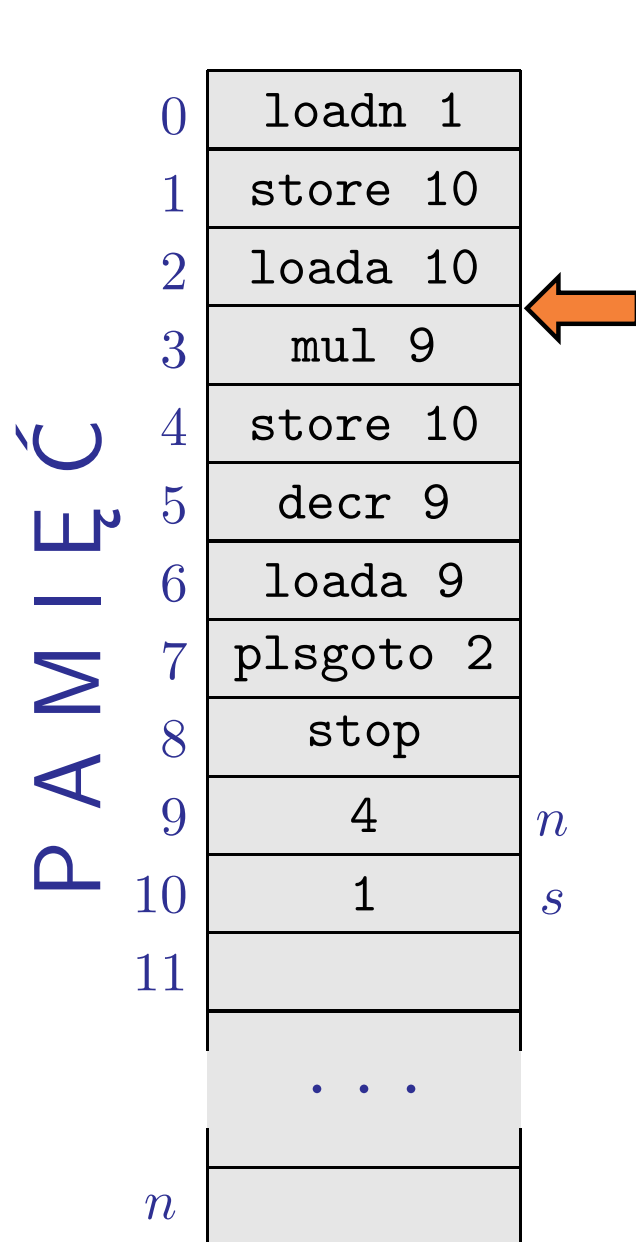
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

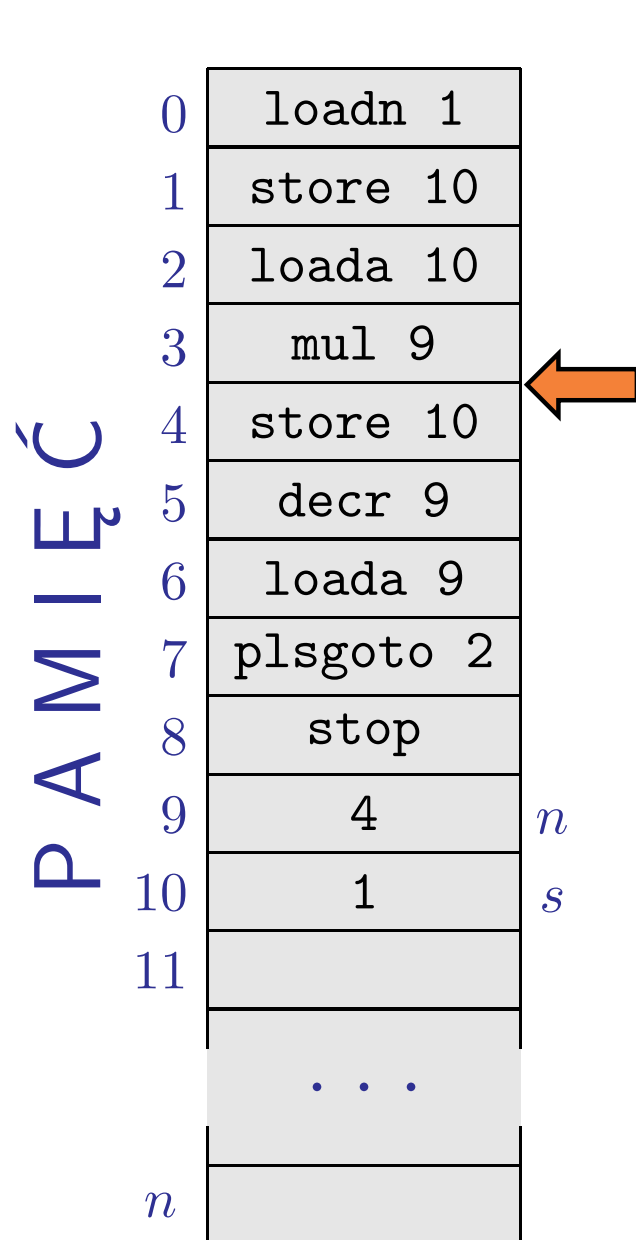
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

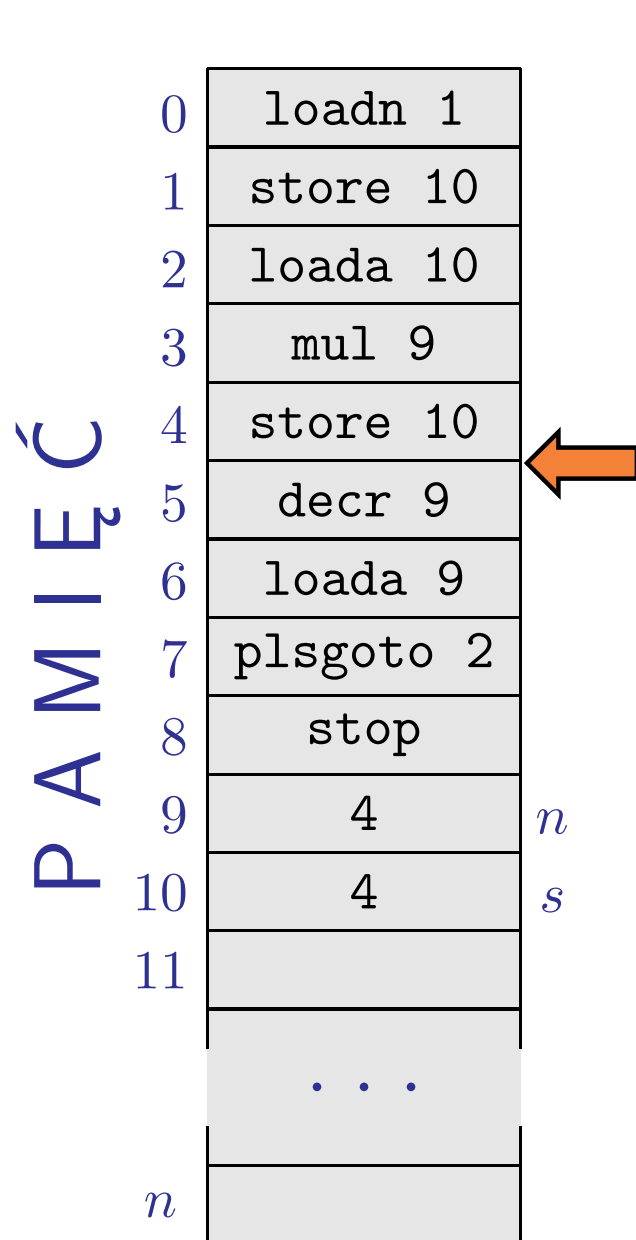
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

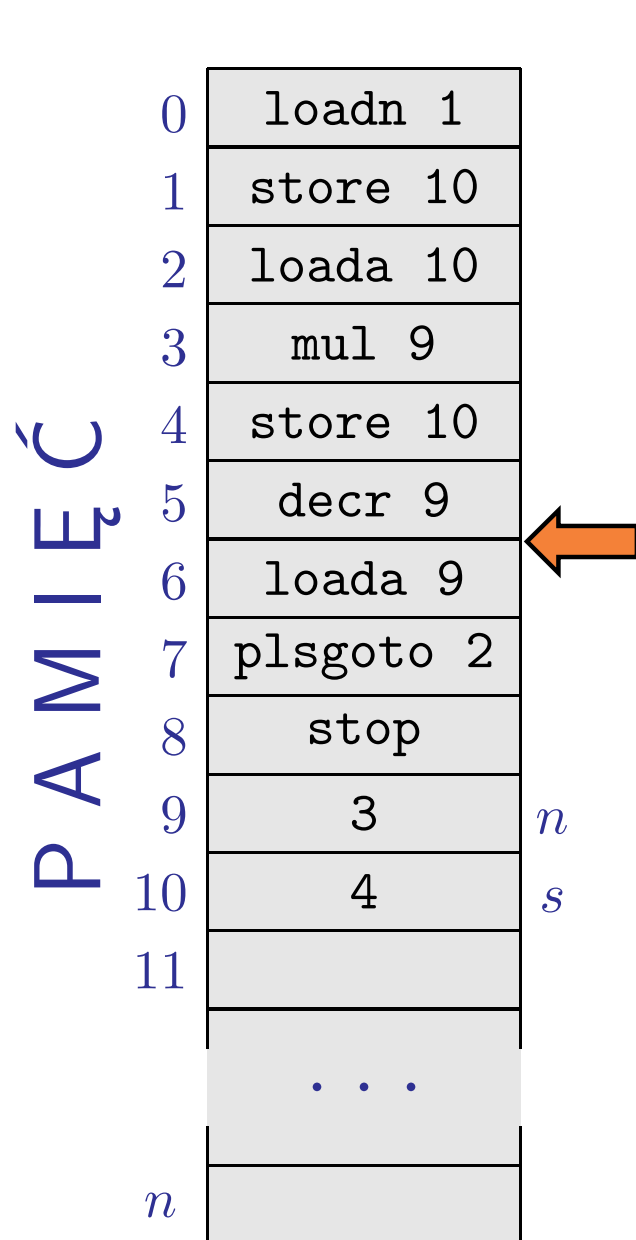
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

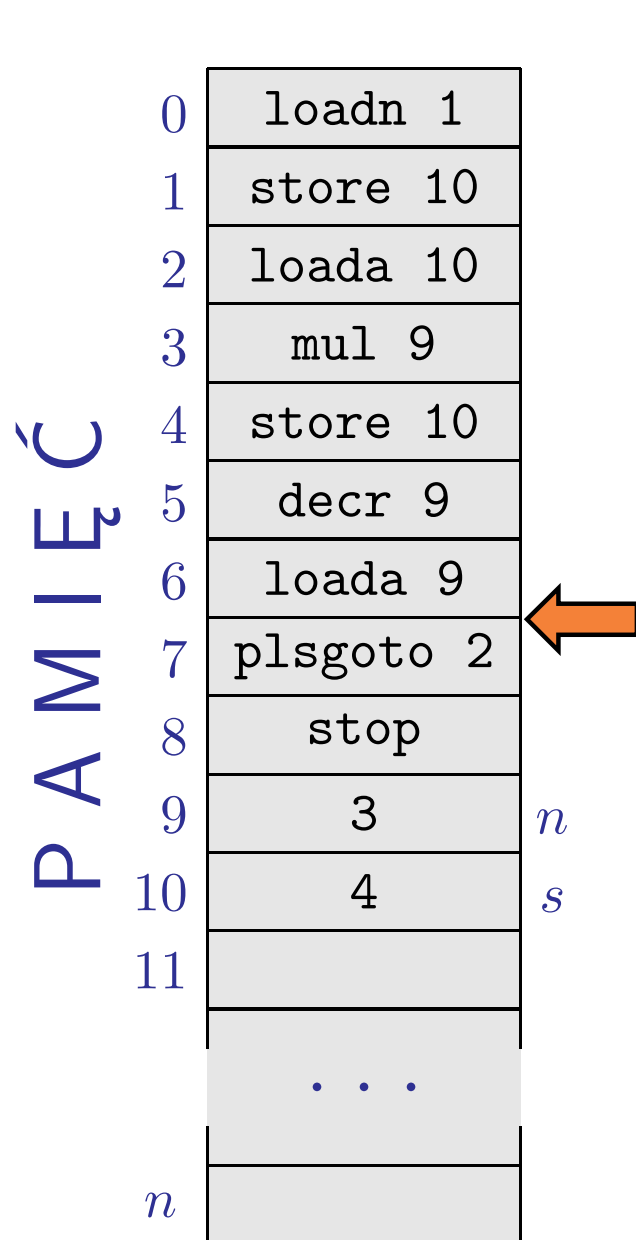
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

3

PRZYKŁAD:

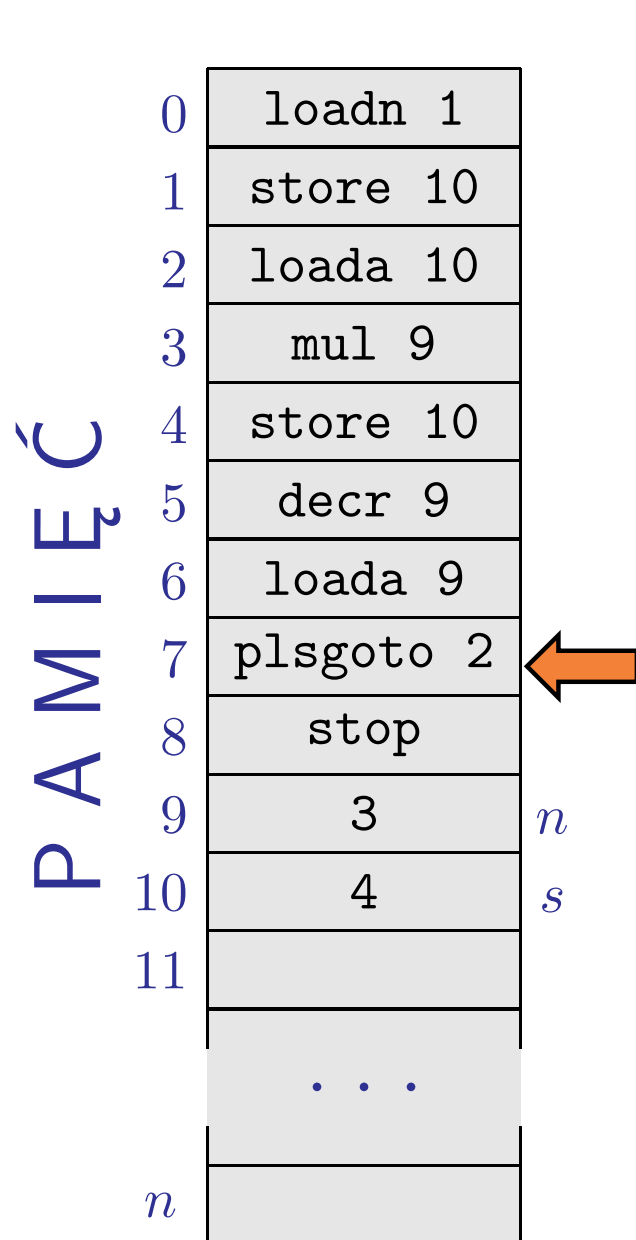
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

3

PRZYKŁAD:

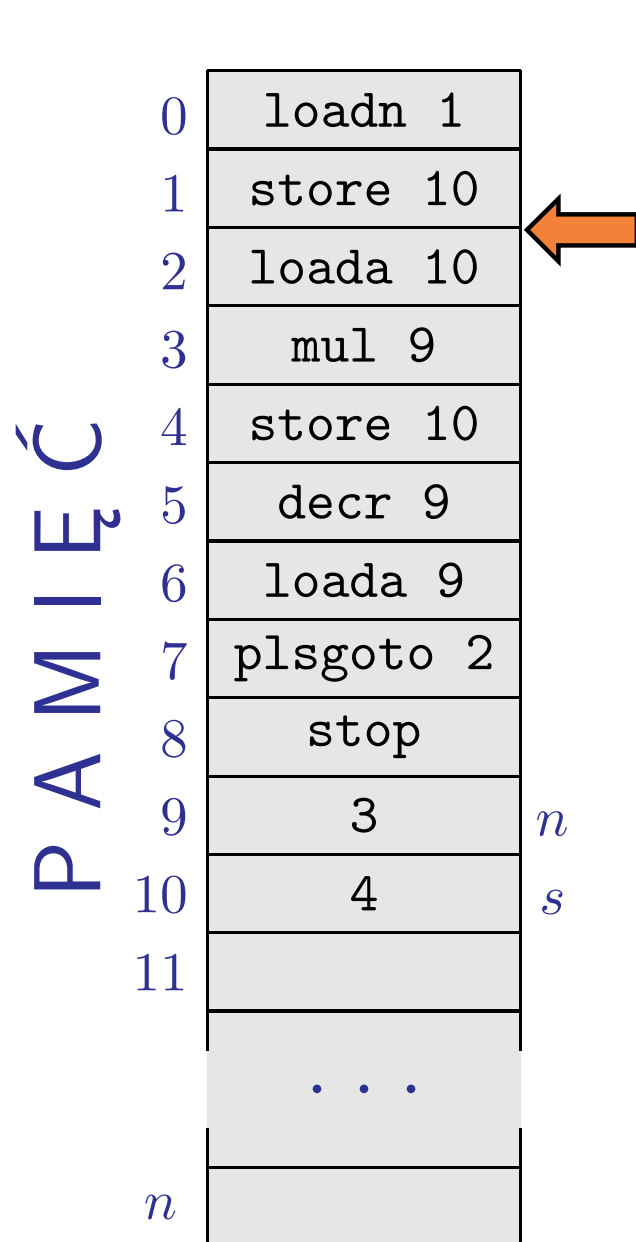
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

3

PRZYKŁAD:

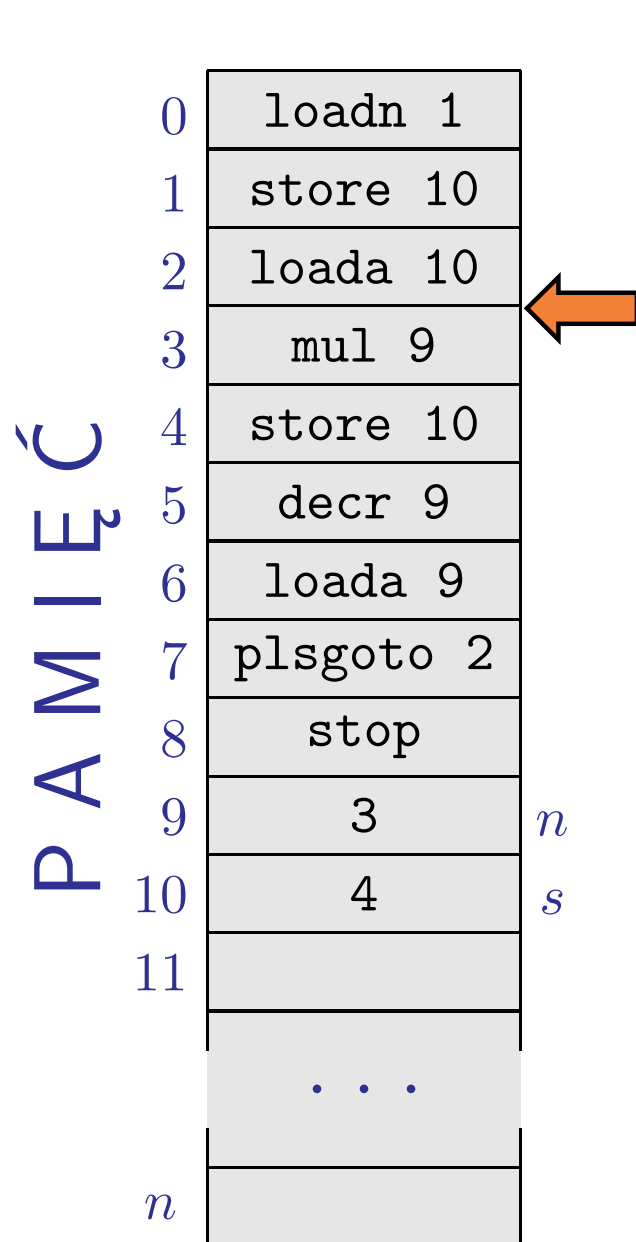
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

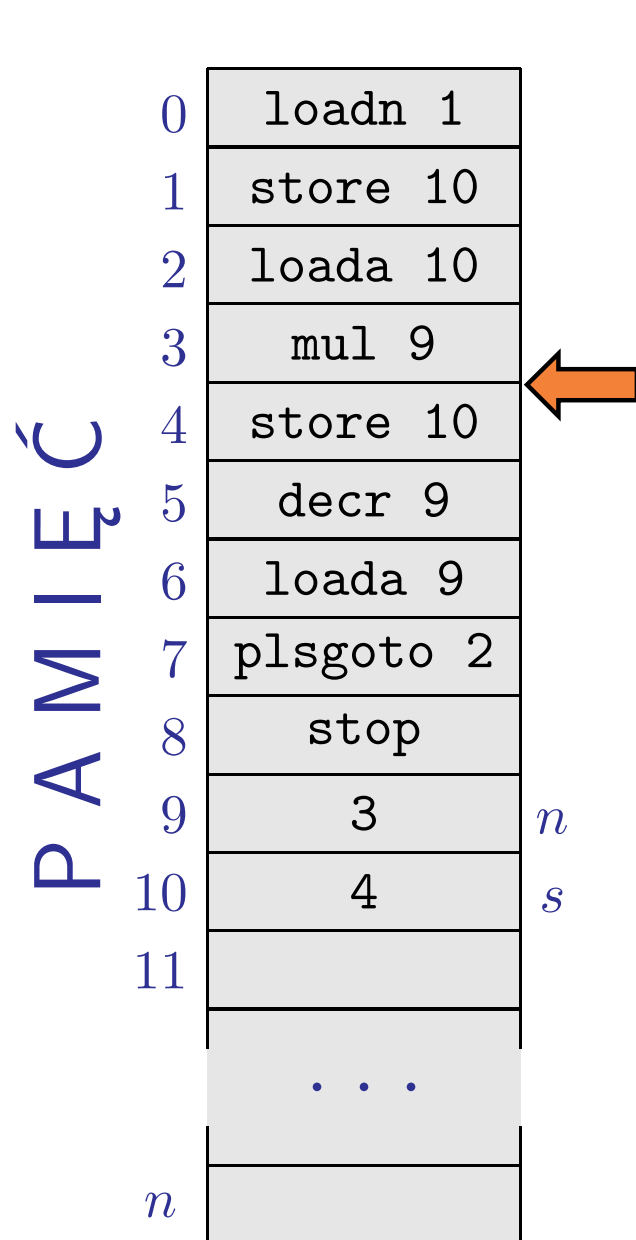
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

12

PRZYKŁAD:

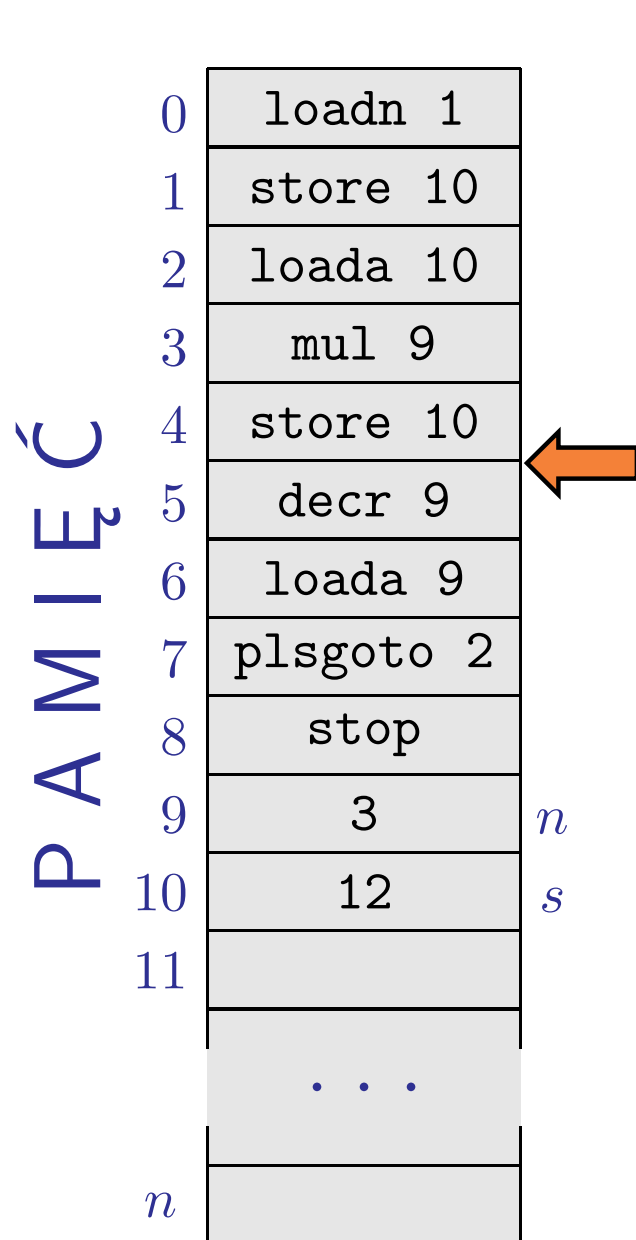
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

12

PRZYKŁAD:

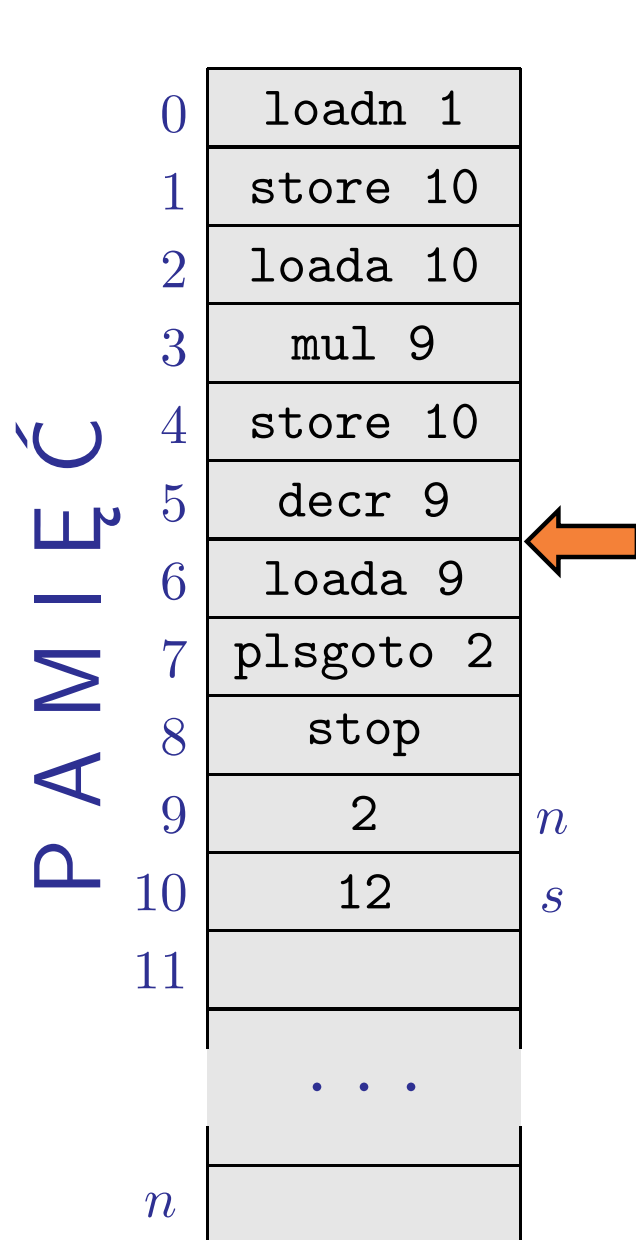
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

12

PRZYKŁAD:

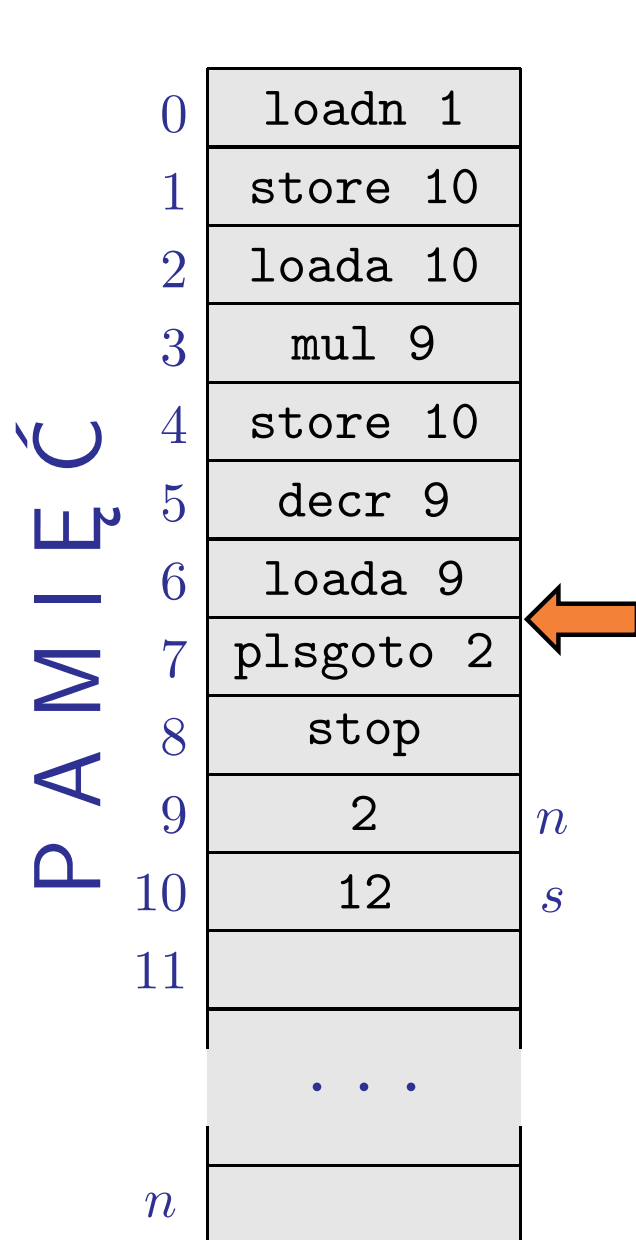
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

2

PRZYKŁAD:

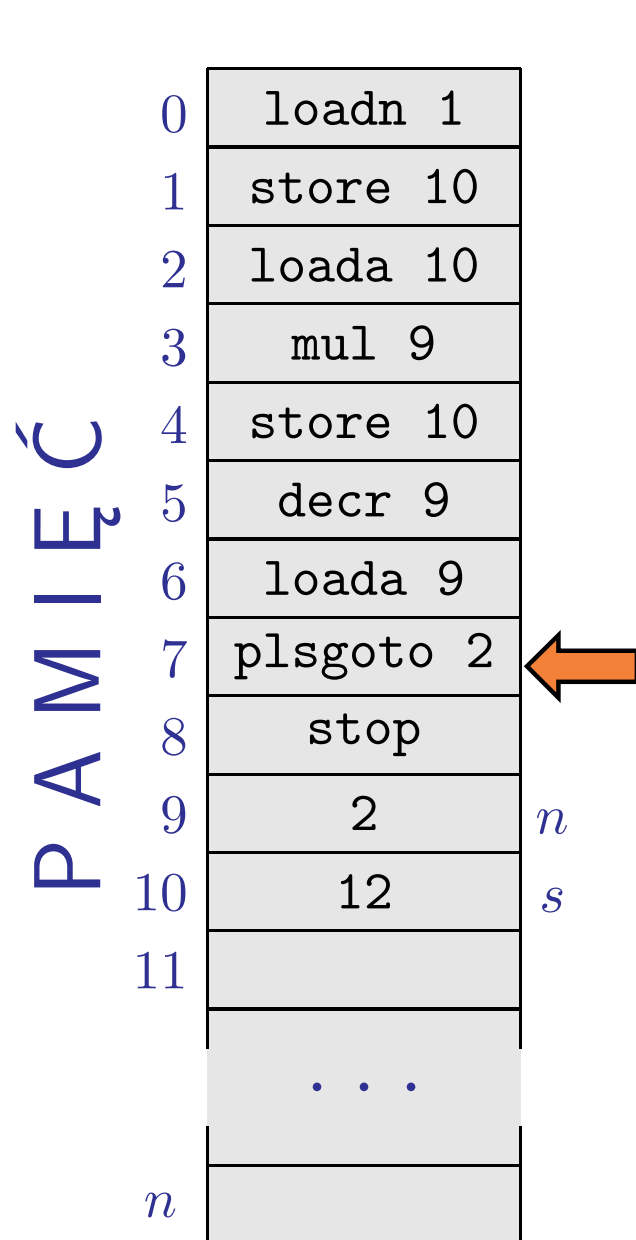
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

2

PRZYKŁAD:

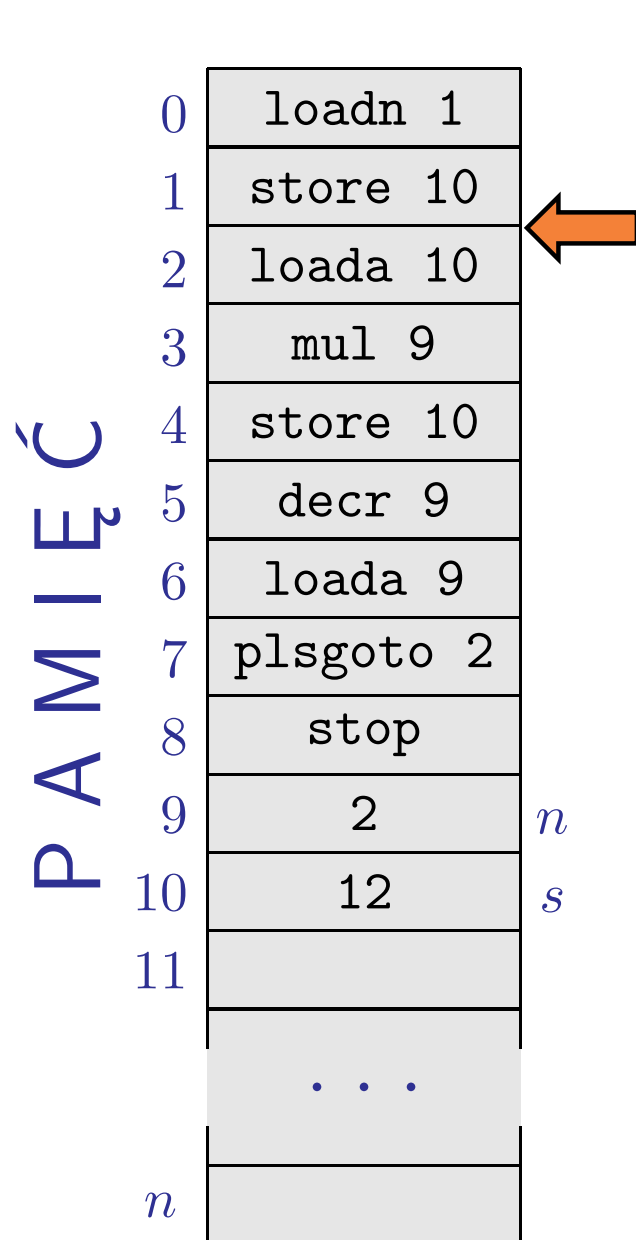
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

2

PRZYKŁAD:

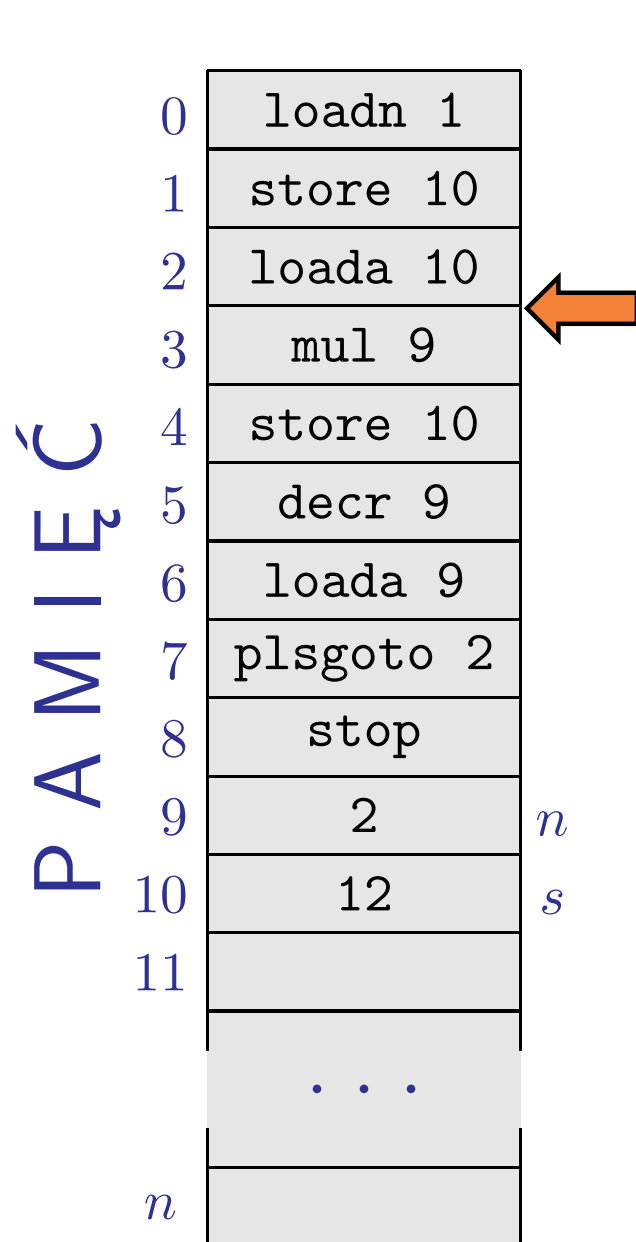
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

12

PRZYKŁAD:

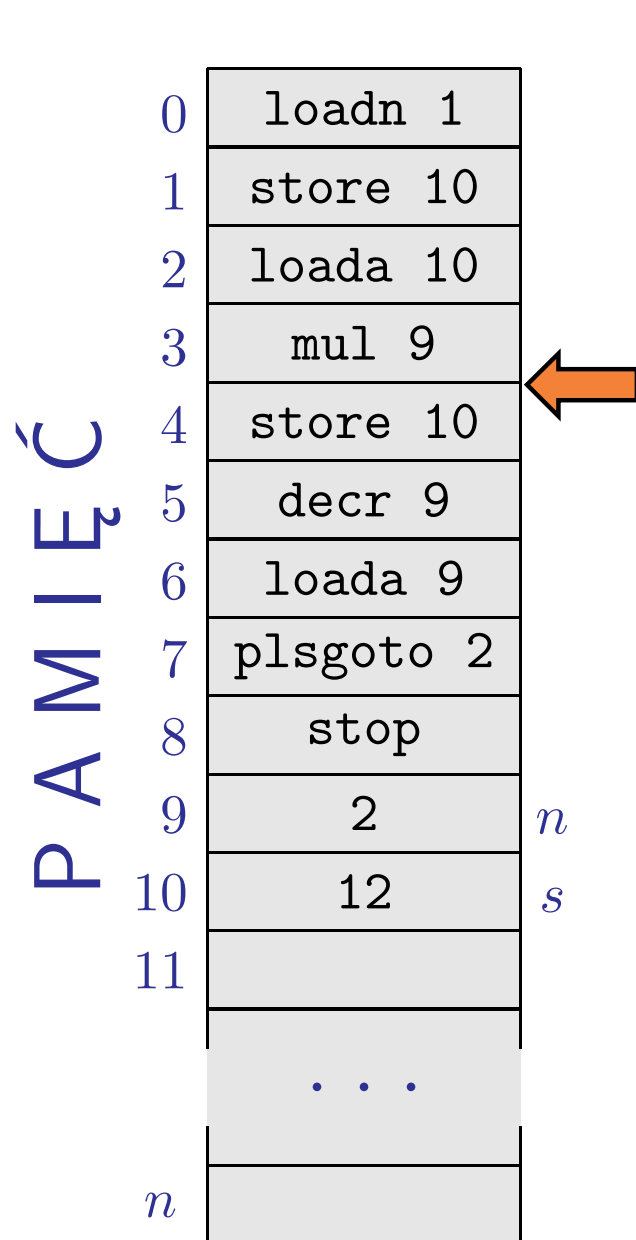
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

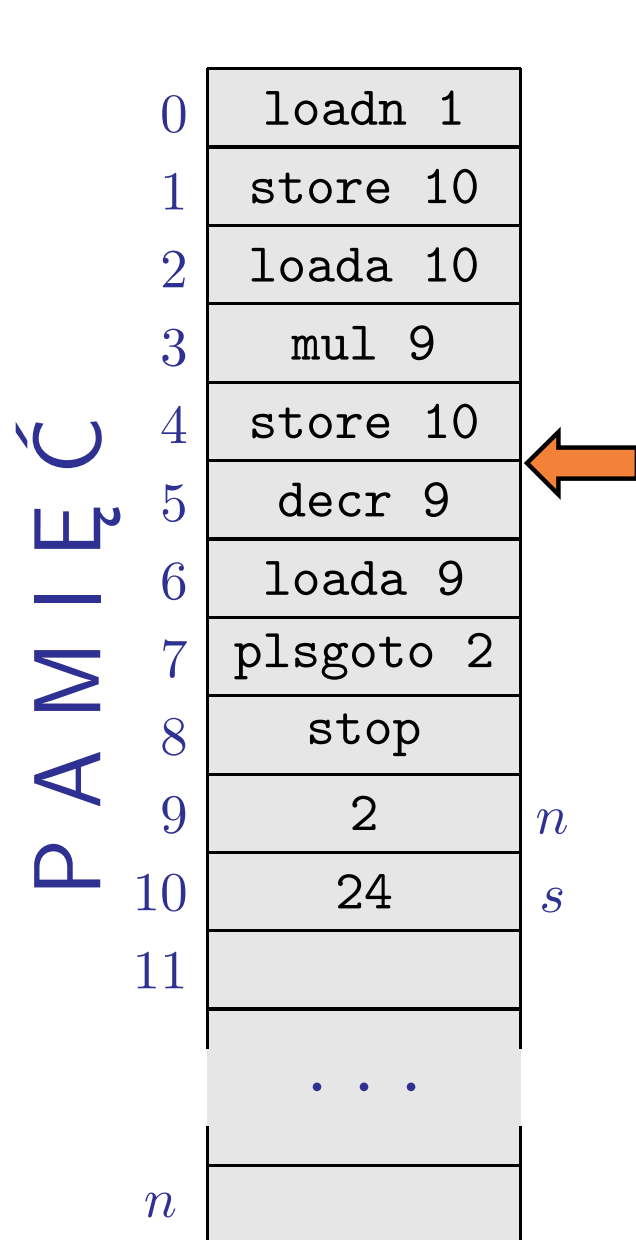
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

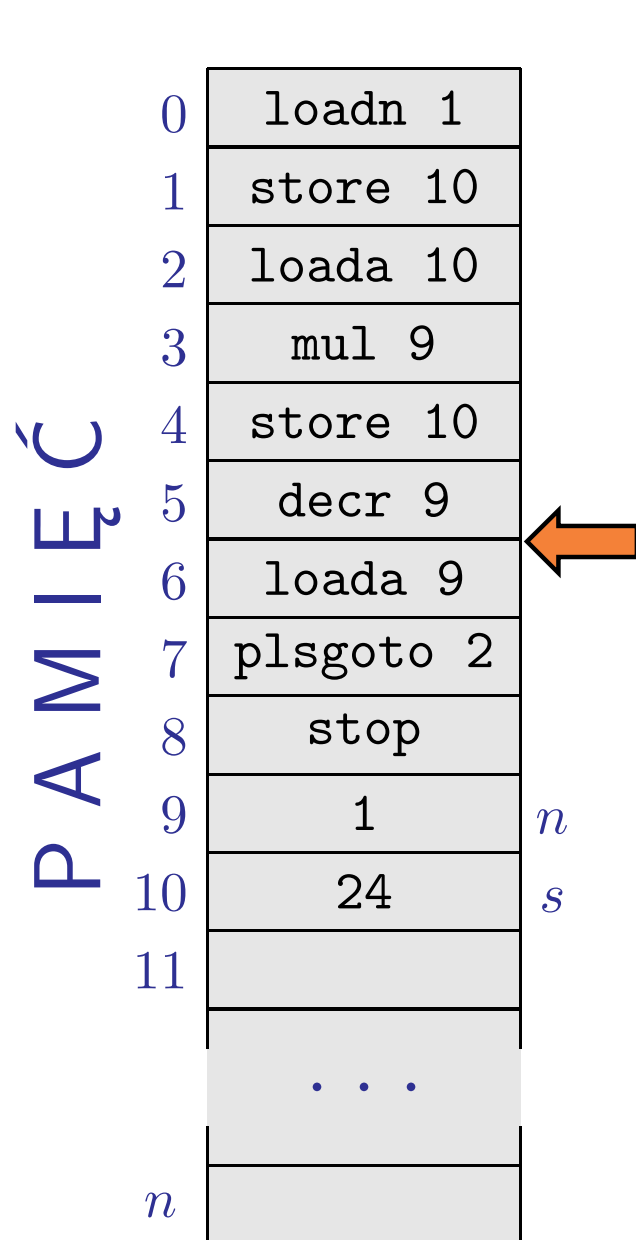
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

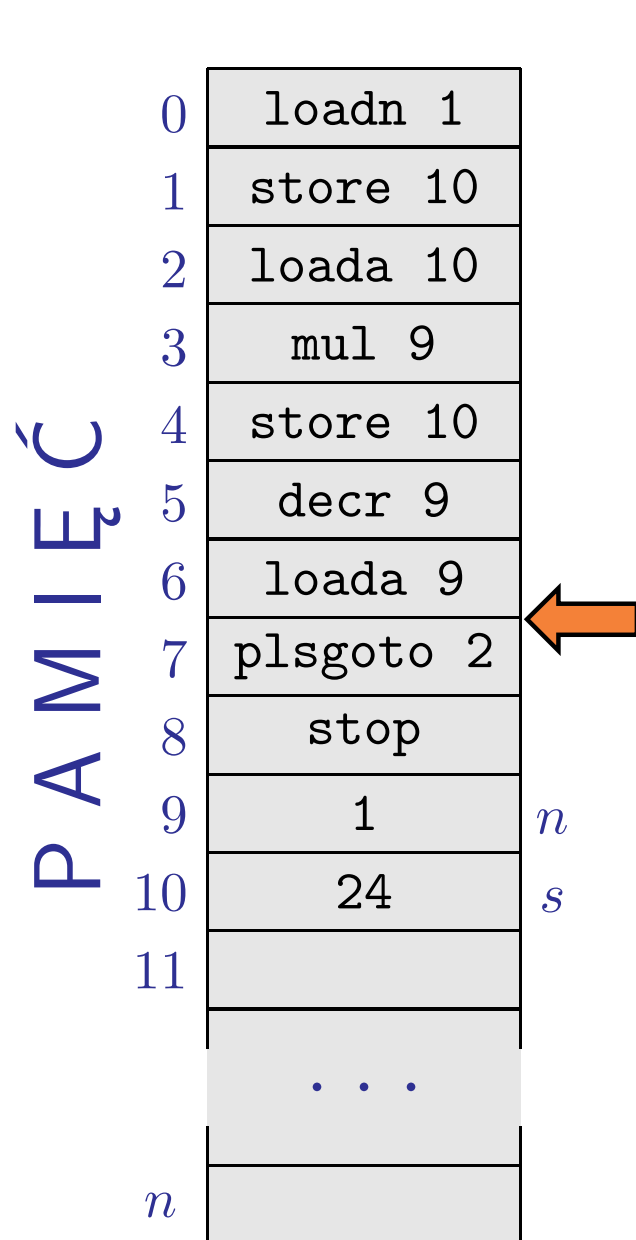
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

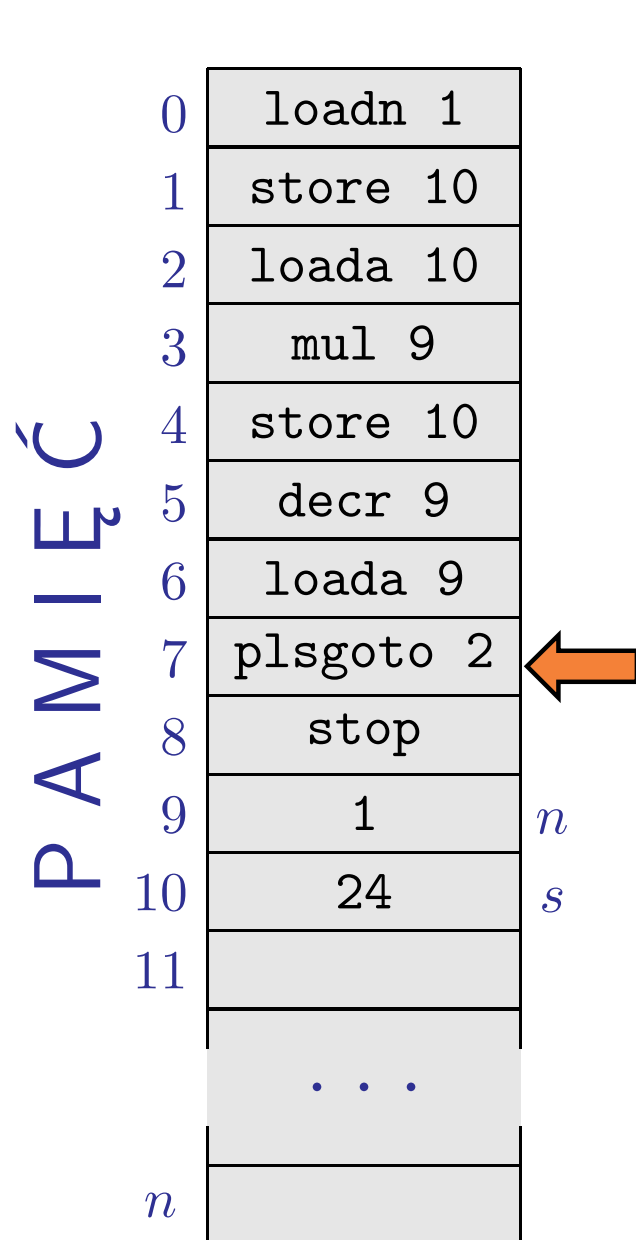
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

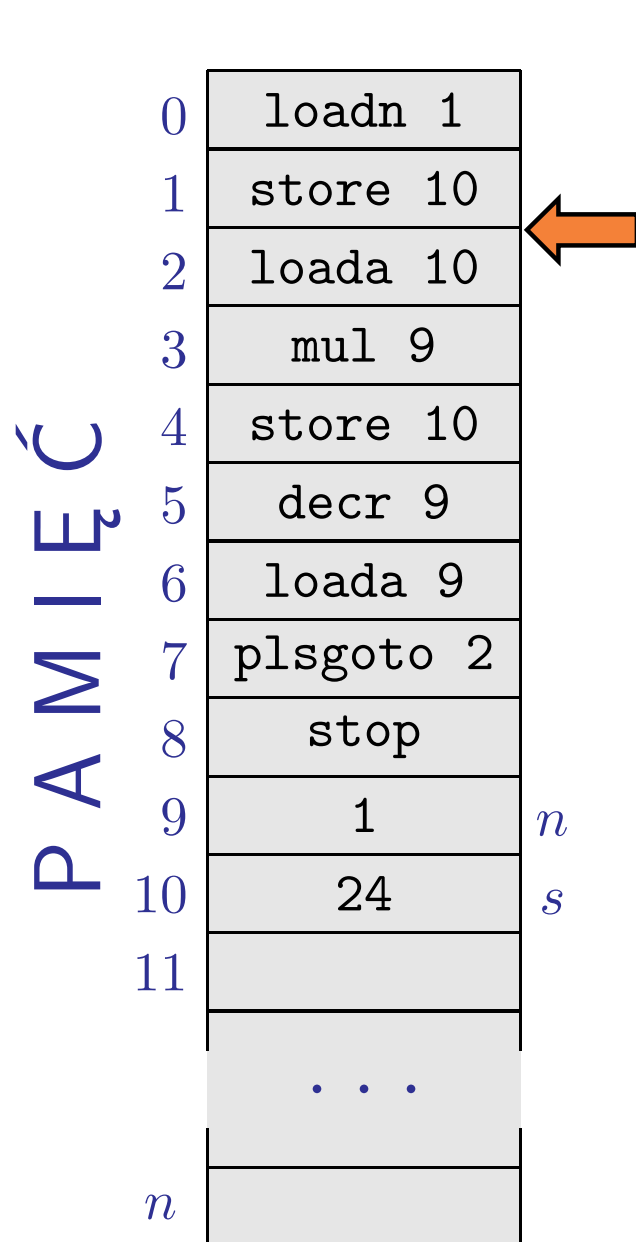
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

1

PRZYKŁAD:

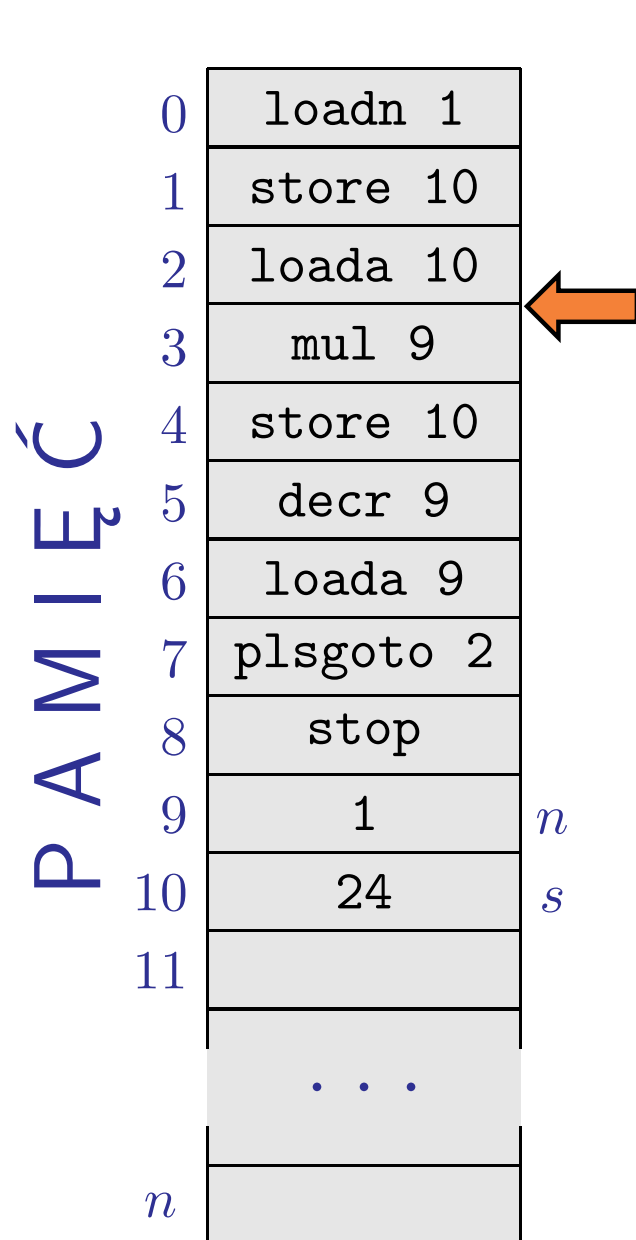
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

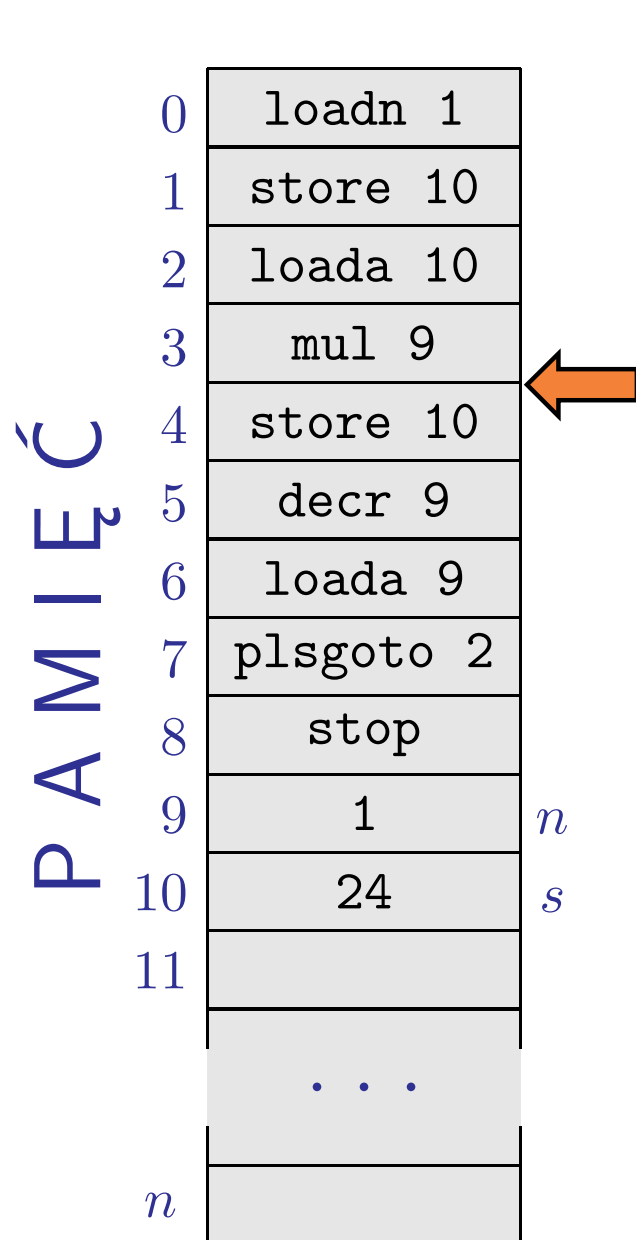
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

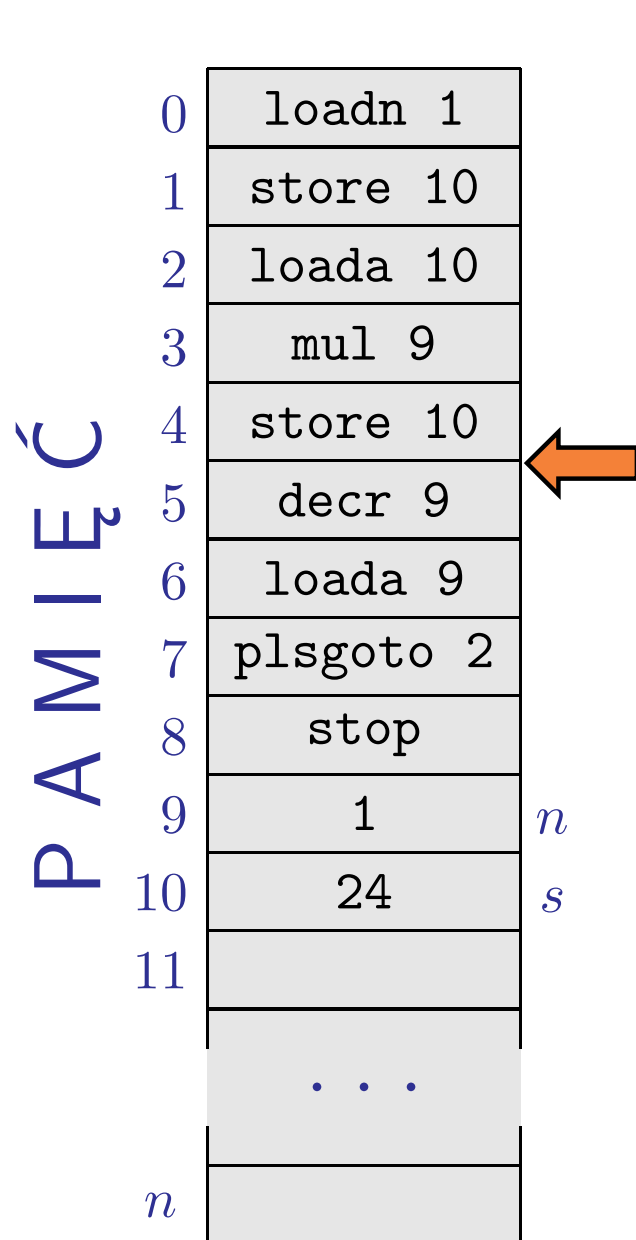
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

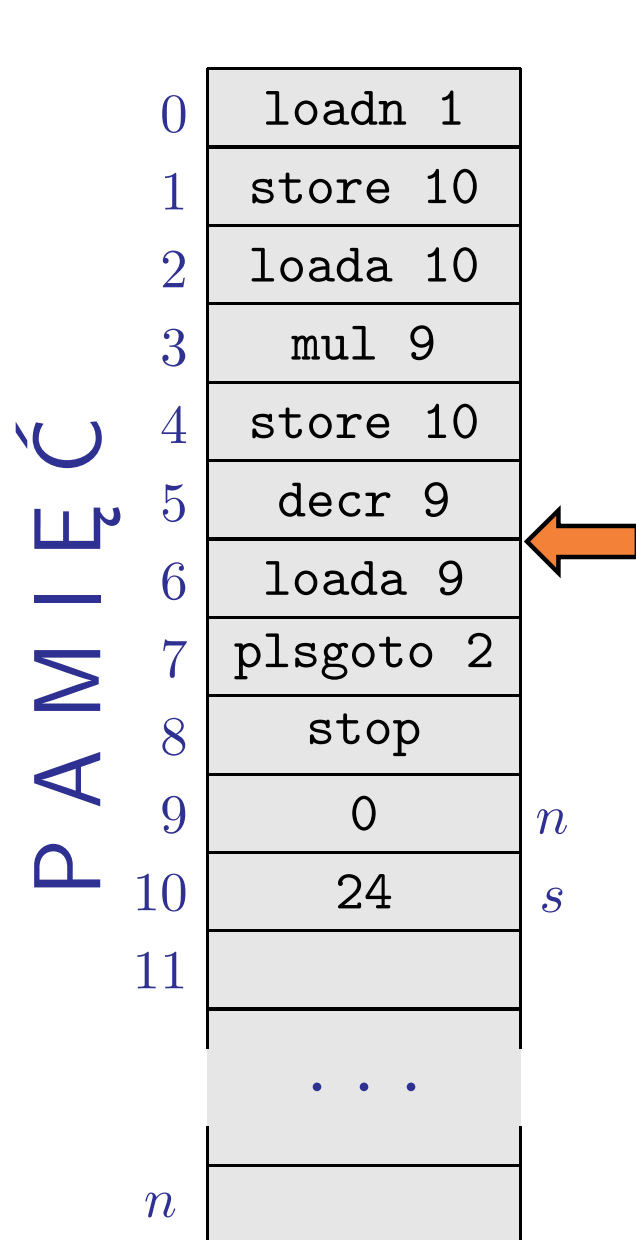
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

24

PRZYKŁAD:

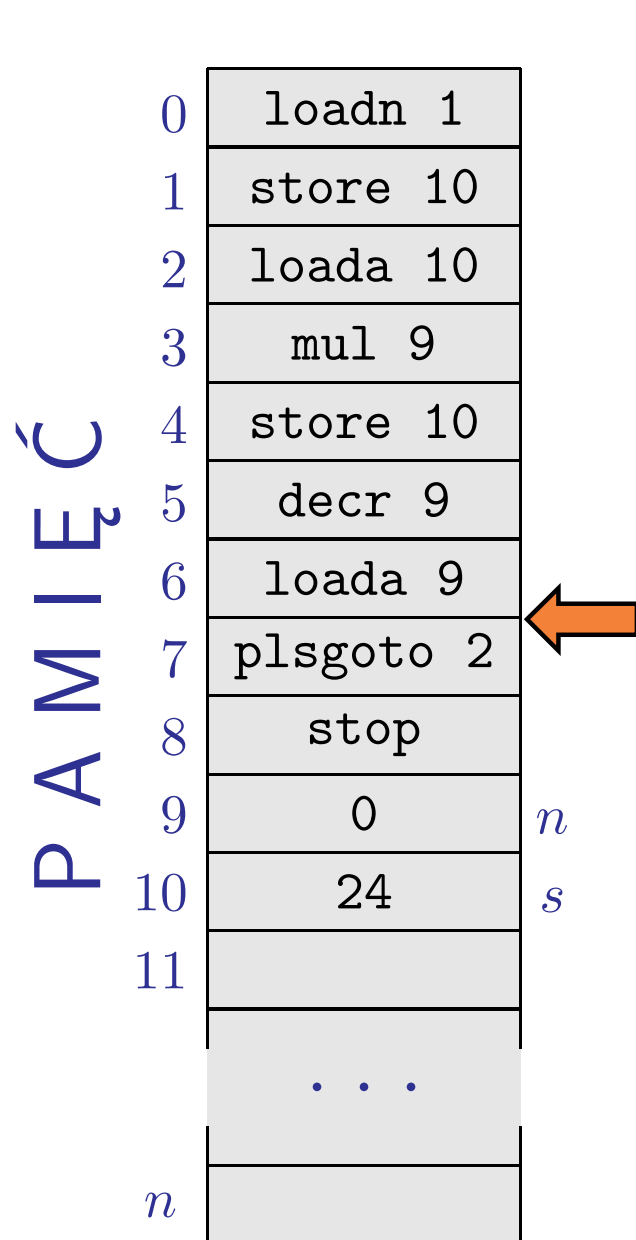
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

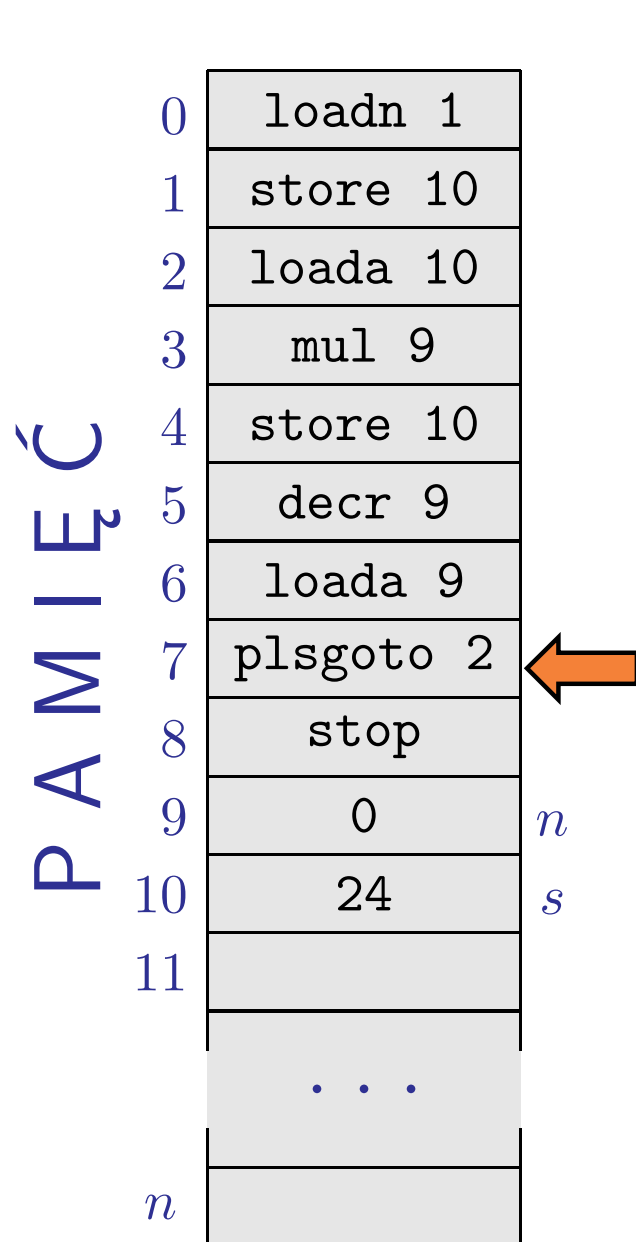
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

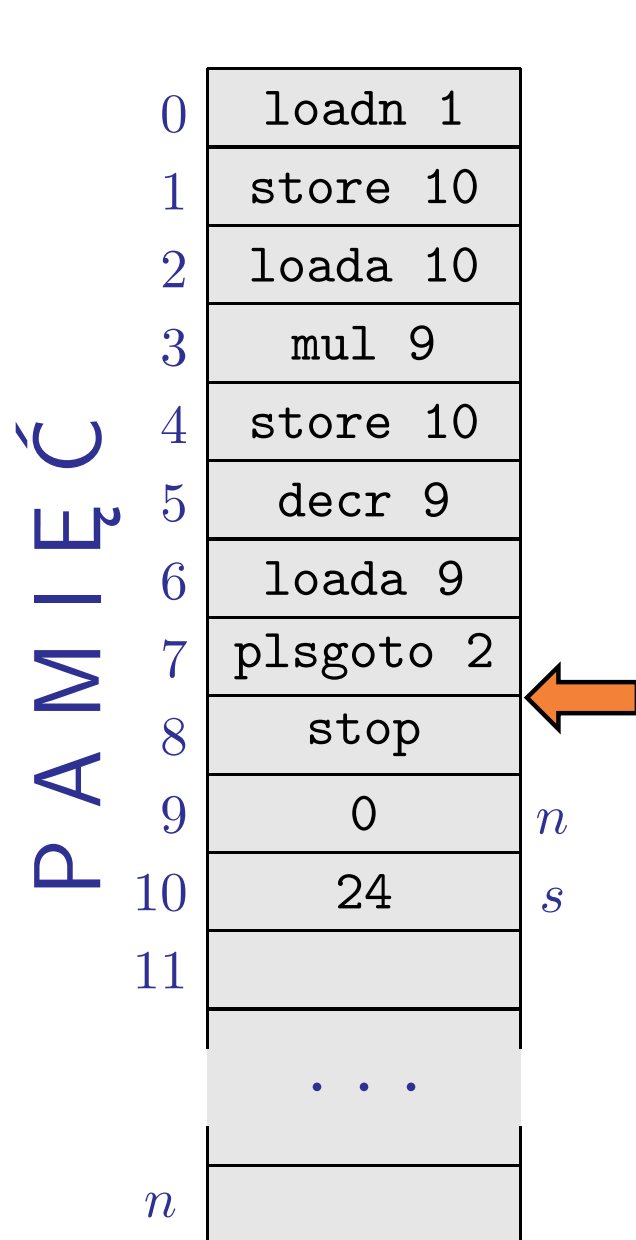
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

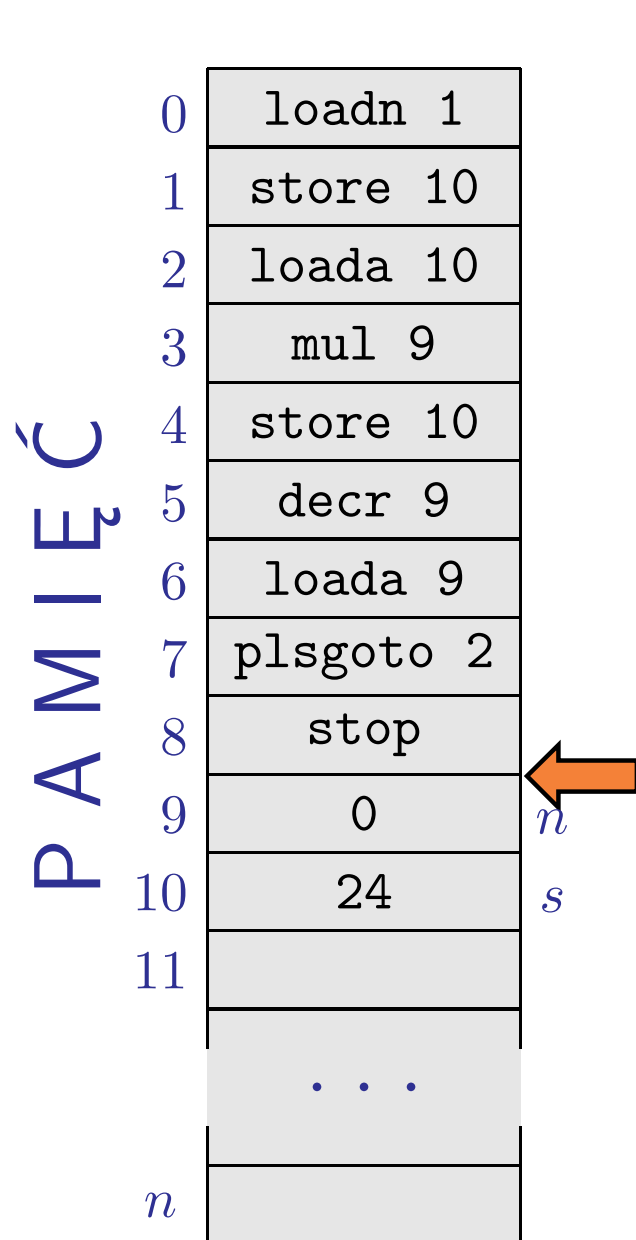
Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Programowanie niskiego poziomu



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

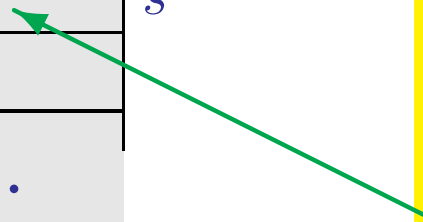
Programowanie niskiego poziomu

PAMIĘĆ

0	loadn 1
1	store 10
2	loada 10
3	mul 9
4	store 10
5	decr 9
6	loada 9
7	plsgoto 2
8	stop
9	0
10	24
11	
	...
n	

n

s



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Obliczenie silni dla $n \geq 1$:

```
s=1;
do
    s=s*n;
    n--;
while (n>0);
```

Program należy uruchamiać od komórki 0;
liczba n ma być w kom. 9;
wynik s ma być w kom. 10.

Policzyć silnię od 4.

Adresy

1 Prostý adres komendy

Adresy

1 Prostý adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500

Adresy

1 Prostý adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

Adresy

1 Prostý adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

2 Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k , należy tę zawartość do adresu n dodać.

Adresy

1 Prostý adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

2 Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k , należy tę zawartość do adresu n dodać.

Np. jeśli pod adresem 500 stoi liczba 1, to komenda `add 200+[500]` oznacza: dodaj do akumulatora zawartość komórki 201

Adresy

1 Prostý adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

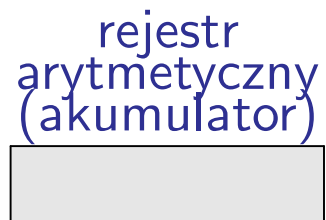
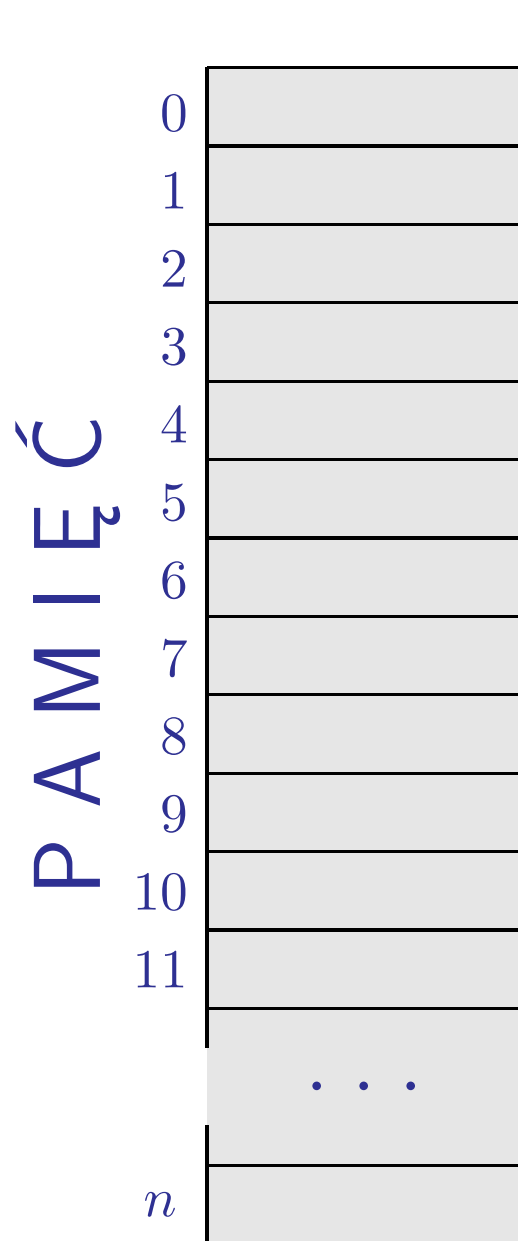
2 Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k , należy tę zawartość do adresu n dodać.

Np. jeśli pod adresem 500 stoi liczba 1, to komenda `add 200+[500]` oznacza: dodaj do akumulatora zawartość komórki 201;

a jeśli pod adresem 500 stoi liczba 5, to *ta sama* komenda `add 200+[500]` oznacza: dodaj do akumulatora zawartość komórki 205.

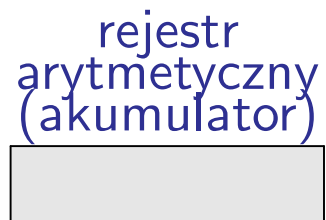
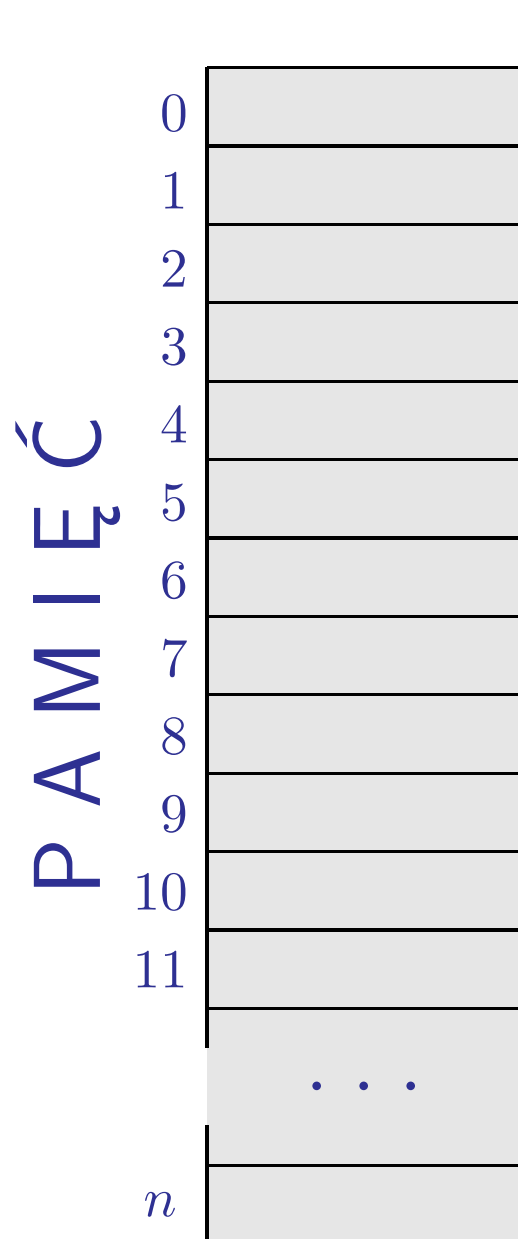
Adresy



PRZYKŁAD:

Zerowanie n komórek

Adresy

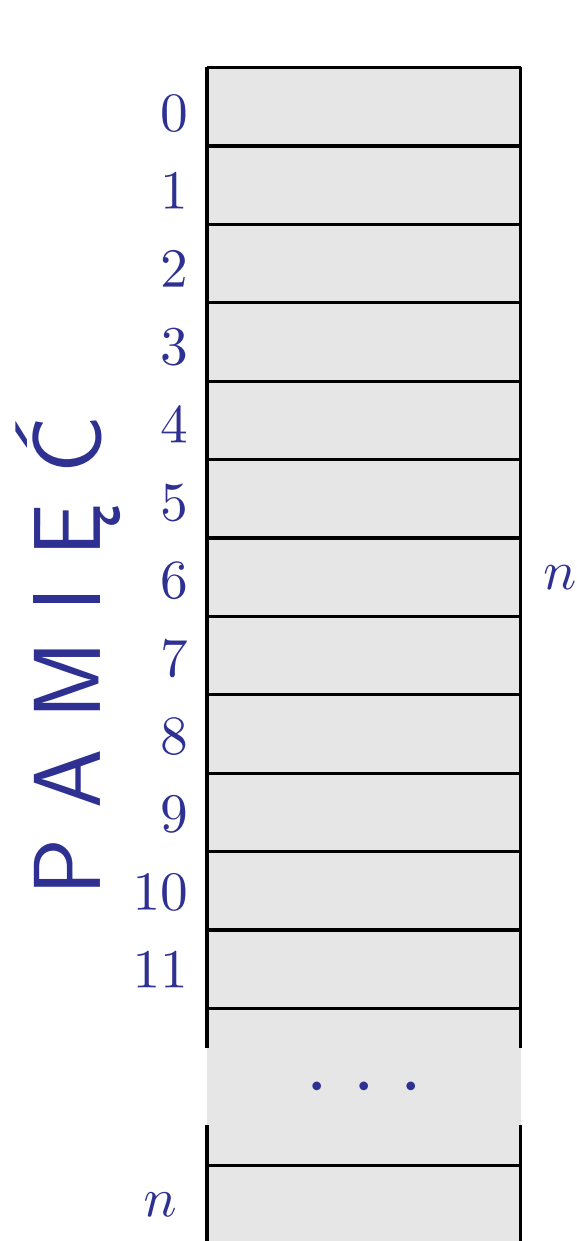


PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Adresy



rejestr
arytmetyczny
(akumulator)



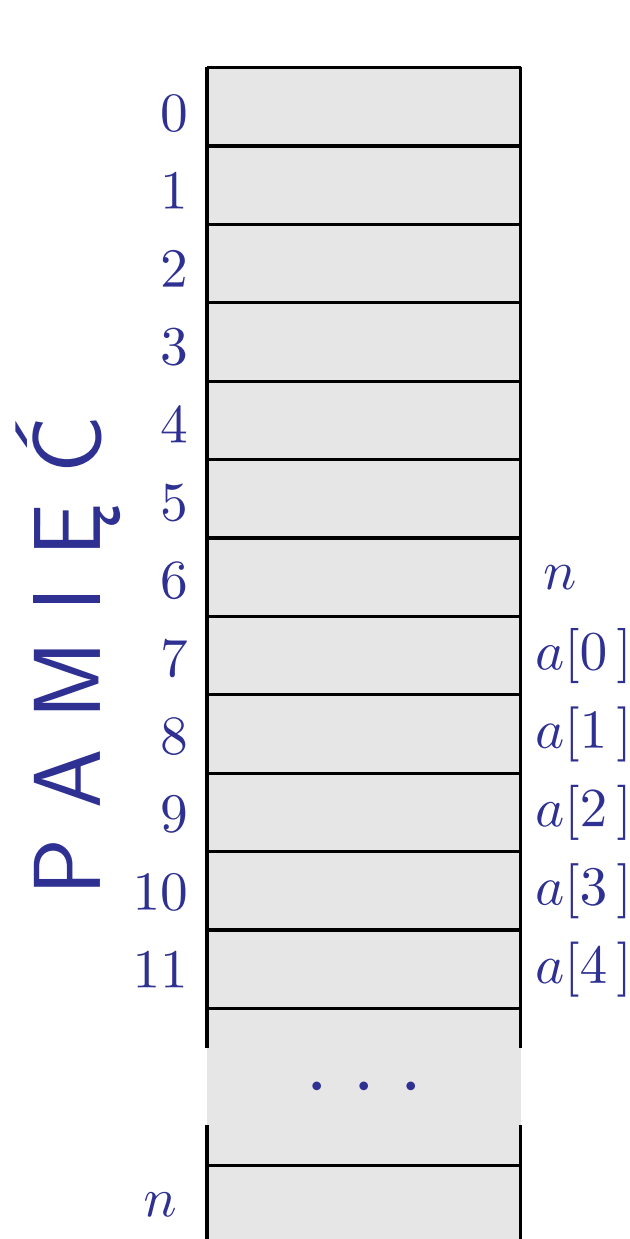
PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6

Adresy



rejestr
arytmetyczny
(akumulator)



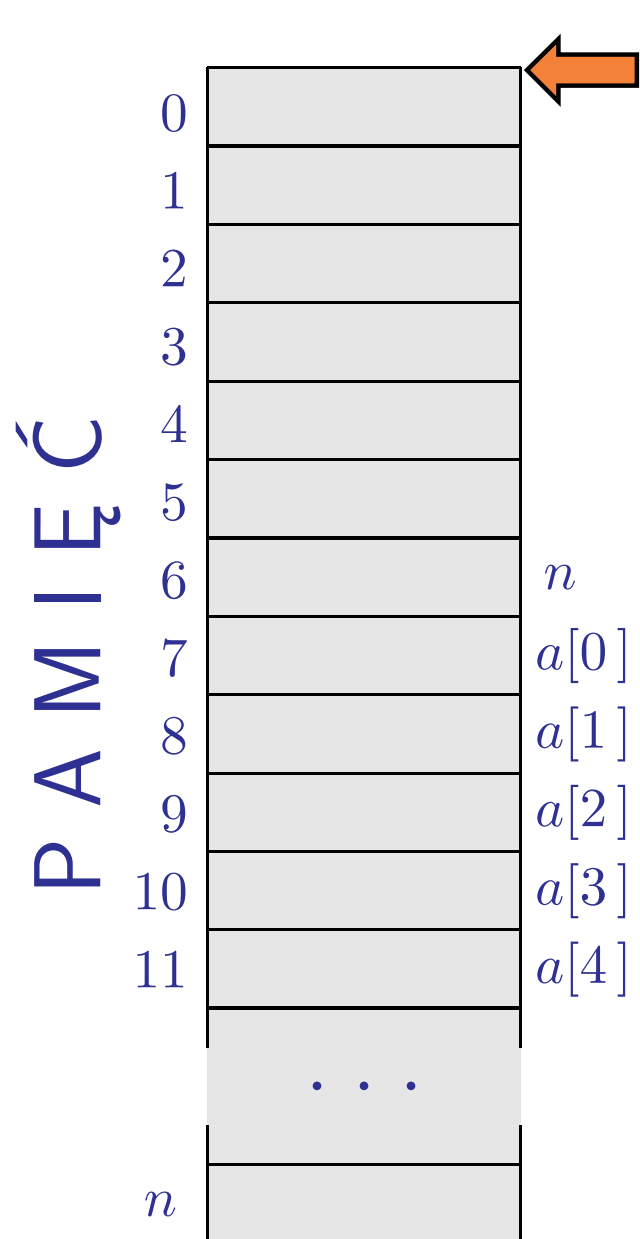
PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7

Adresy



rejestr
arytmetyczny
(akumulator)



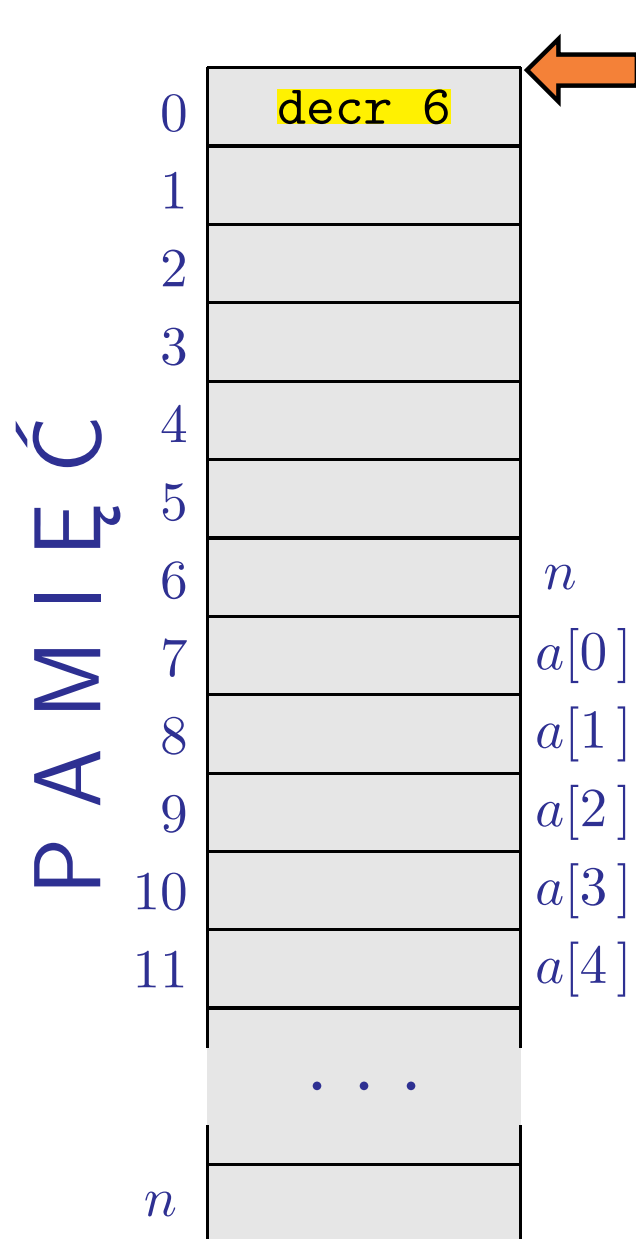
PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy



rejestr
arytmetyczny
(akumulator)

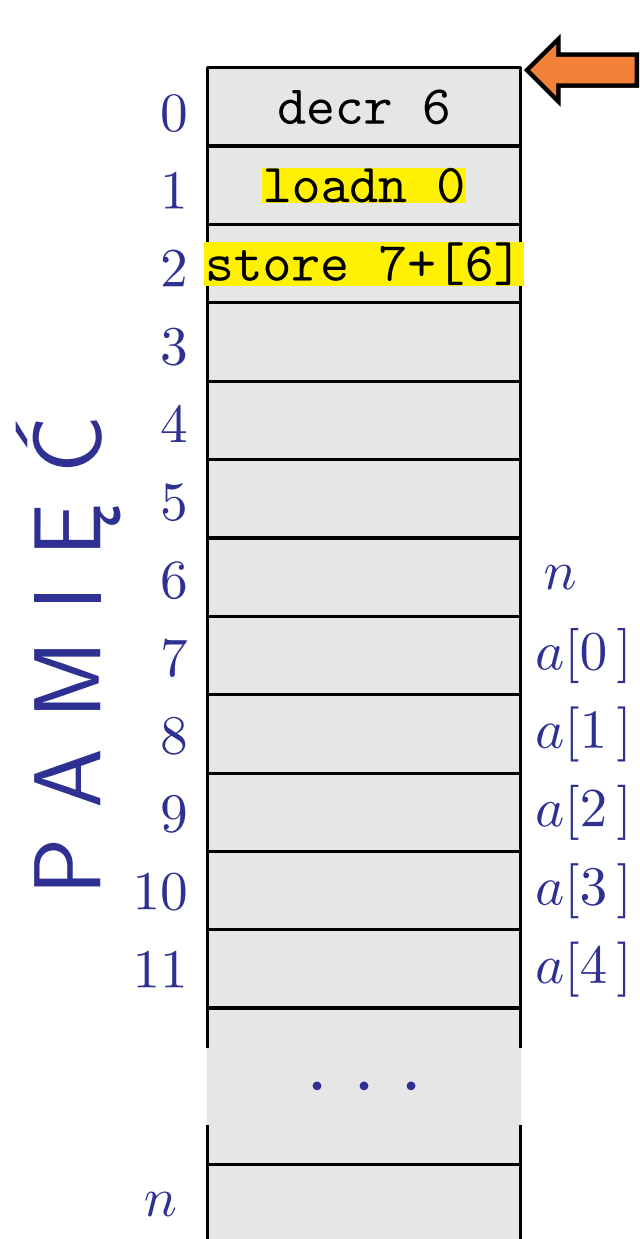
PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy



rejestr
arytmetyczny
(akumulator)



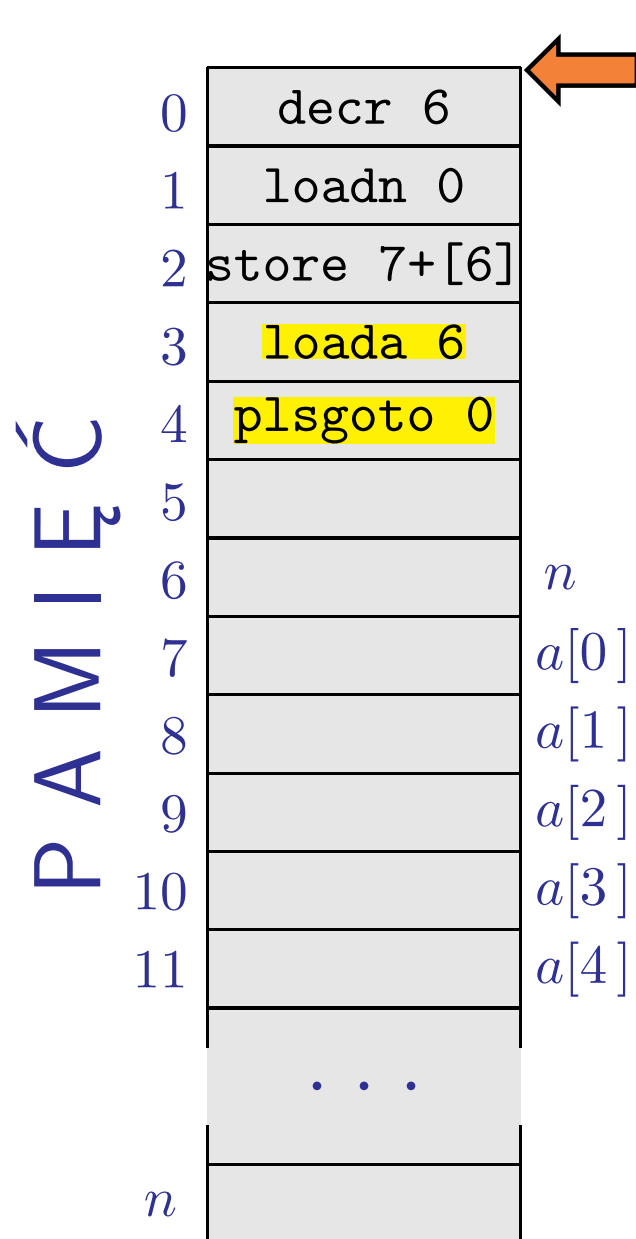
PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy



rejestr
arytmetyczny
(akumulator)



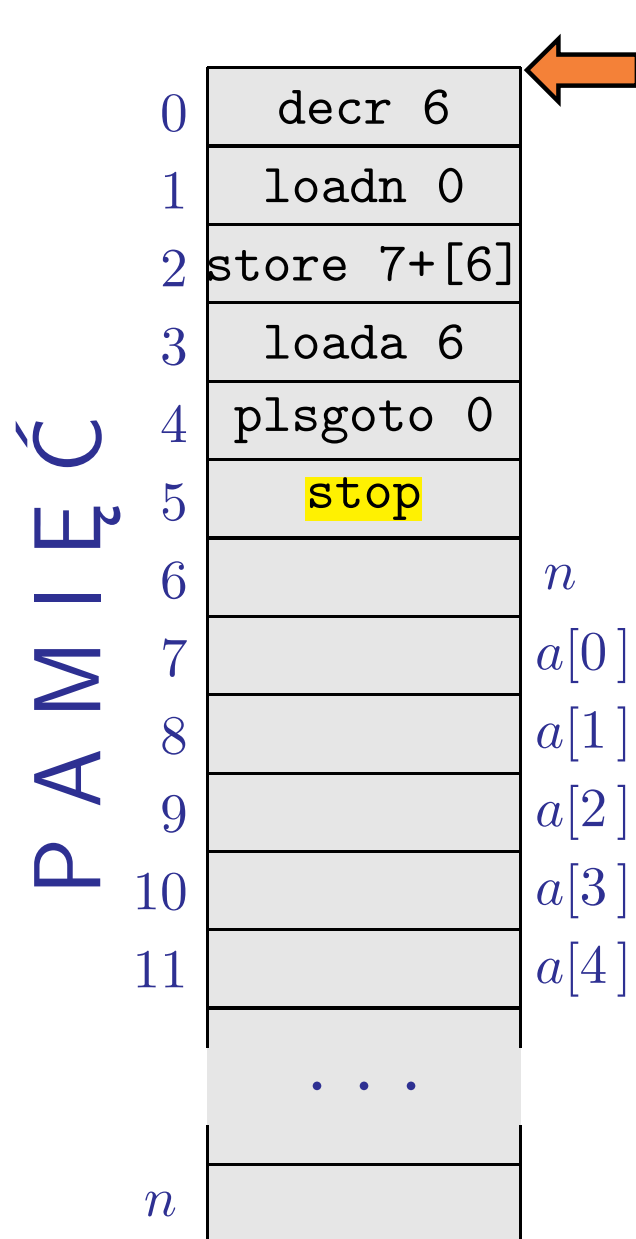
PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy



rejestr
arytmetyczny
(akumulator)



PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

PAMIĘĆ

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	5	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	-1	$a[4]$
	...	
n		

rejestr
arytmetyczny
(akumulator)



PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	-1	$a[4]$
	...	
n		

rejestr
arytmetyczny
(akumulator)



PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	-1	$a[4]$
	...	
n		



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		



rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	←
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		

rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

PAMIĘĆ

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	4	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		

rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

PAMIĘĆ

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	3	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		

rejestr
arytmetyczny
(akumulator)

4

PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

PAMIĘĆ

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	3	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	9	$a[3]$
11	0	$a[4]$
	...	
n		



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Adresy

P A M I Ę Ć

0	decr 6	
1	loadn 0	
2	store 7+[6]	
3	loada 6	
4	plsgoto 0	
5	stop	
6	3	n
7	10	$a[0]$
8	-12	$a[1]$
9	25	$a[2]$
10	0	$a[3]$
11	0	$a[4]$
	...	
n		



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Zerowanie n komórek

```
do
    n--;
    a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Jak programować w języku niskiego poziomu?

Jak programować w języku niskiego poziomu?

- Dużo drobniutkich kroczków — bardzo żmudne.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.
- Napisać program w języku wyższego poziomu i tłumaczyć komenda po komendzie.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.
- Napisać program w języku wyższego poziomu i tłumaczyć komenda po komendzie.
- Na razie używać „*adresów symbolicznych*” — nie numerów komórek tylko nazw zmiennych.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.
- Napisać program w języku wyższego poziomu i tłumaczyć komenda po komendzie.
- Na razie używać „*adresów symbolicznych*” — nie numerów komórek tylko nazw zmiennych.
- Dopiero na końcu ponumerować komendy i zmienne.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.
- Napisać program w języku wyższego poziomu i tłumaczyć komenda po komendzie.
- Na razie używać „*adresów symbolicznych*” — nie numerów komórek tylko nazw zmiennych.
- Dopiero na końcu ponumerować komendy i zmienne.
- Jeśli źle działa, może być łatwiej napisać od nowa niż poprawiać.

Jak programować w języku niskiego poziomu?

- Dużo drobniotkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.
- Napisać program w języku wyższego poziomu i tłumaczyć komenda po komendzie.
- Na razie używać „*adresów symbolicznych*” — nie numerów komórek tylko nazw zmiennych.
- Dopiero na końcu ponumerować komendy i zmienne.
- Jeśli źle działa, może być łatwiej napisać od nowa niż poprawiać.

Tak też działa kompilator.

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu $\rightarrow$ język wys. poziomu bez pętli

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu $\rightarrow$ język wys. poziomu bez pętli

```
p = 1;  
for (i=0; i<n; i++)  
    p = p*x;
```

koniec:

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu → język wys. poziomu bez pętli

```
p = 1;  
for (i=0; i<n; i++)
```

```
    p = p*x;
```

pętla:

koniec:

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu $\rightarrow$ język wys. poziomu bez pętli

```
p = 1;  
for (i=0; i<n; i++)  
  
    p = p*x;
```

```
p = 1;
```

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu → język wys. poziomu bez pętli

```
p = 1;  
for (i=0; i<n; i++)  
  
    p = p*x;
```

```
p = 1;  
i = 0;  
pętla:    if (n-i == 0) goto koniec;  
  
        i++;  
        goto pętla;  
  
koniec:
```

Jak programować w języku niskiego poziomu?

Przykład — potęgowanie

język wys. poziomu → język wys. poziomu bez pętli

```
p = 1;  
for (i=0; i<n; i++)
```

```
    p = p*x;
```

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;  
i = 0;  
pętla:  if (n-i>0) goto koniec;  
        p = p*x;  
        i++;  
        goto pętla;  
koniec:
```

pętla:

koniec:

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla:    if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
pętla:
```

```
koniec:
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;  
  
i = 0;  
  
pętla:   if (n-i == 0) goto koniec;  
  
p = p*x;  
  
i++;  
goto pętla;  
  
koniec:
```

```
loadn 1  
store p
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
loadn 1
```

```
store p
```

```
loadn 0
```

```
store i
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
loadn 1
```

```
store p
```

```
loadn 0
```

```
store i
```

```
pętla: loada n
```

```
sub i
```

```
zergoto koniec
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
loadn 1
```

```
store p
```

```
loadn 0
```

```
store i
```

```
pętla: loada n
```

```
sub i
```

```
zergoto koniec
```

```
loada p
```

```
mul x
```

```
store p
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
loadn 1
```

```
store p
```

```
loadn 0
```

```
store i
```

```
pętla: loada n
```

```
sub i
```

```
zergoto koniec
```

```
loada p
```

```
mul x
```

```
store p
```

```
incr i
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

```
p = 1;
```

```
i = 0;
```

```
pętla: if (n-i == 0) goto koniec;
```

```
p = p*x;
```

```
i++;
```

```
goto pętla;
```

```
koniec:
```

```
loadn 1
```

```
store p
```

```
loadn 0
```

```
store i
```

```
pętla: loada n
```

```
sub i
```

```
zergoto koniec
```

```
loada p
```

```
mul x
```

```
store p
```

```
incr i
```

```
goto pętla
```

Jak programować w języku niskiego poziomu?

język wys. poziomu bez pętli → język wewn. z adr. symbolicznymi

`p = 1;`

`i = 0;`

pętla: `if (n-i == 0) goto koniec;`

`p = p*x;`

`i++;`

`goto pętla;`

koniec:

`loadn 1`

`store p`

`loadn 0`

`store i`

pętla: `loada n`

`sub i`

`zergoto koniec`

`loada p`

`mul x`

`store p`

`incr i`

`goto pętla`

koniec: `stop`

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```
loadn 1
store p
loadn 0
store i
pętla: loada n
      sub i
      zergoto koniec
      loada p
      mul x
      store p
      incr i
      goto pętla
koniec: stop
```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```
loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop
```



```
loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop
```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

```



```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop
          # x -- podstawa

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

```



```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop
# x -- podstawa
# n -- wykładnik

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

```



```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

# x -- podstawa
# n -- wykładnik
# p -- wynik

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

```



```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

# x -- podstawa
# n -- wykładnik
# p -- wynik
# i -- pomocnicza

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**dopisanie zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

```



```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

# x -- podstawa
# n -- wykładnik
# p -- wynik
# i -- pomocnicza
END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(numeracja komend i zmiennych)

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**numeracja komend i zmiennych**)

```
loadn 1
store p
loadn 0
store i
pętla: loada n
      sub i
      zergoto koniec
      loada p
      mul x
      store p
      incr i
      goto pętla
koniec: stop
      # x -- podstawa
      # n -- wykładnik
      # p -- wynik
      # i -- pomocnicza
END
```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**numeracja komend i zmiennych**)

```

loadn 1
store p
loadn 0
store i
pętla: loada n
sub i
zergoto koniec
loada p
mul x
store p
incr i
goto pętla
koniec: stop

# x -- podstawa
# n -- wykładnik
# p -- wynik
# i -- pomocnicza

END

```



```

0: loadn 1
1: store p
2: loadn 0
3: store i
pętla: 4: loada n
5: sub i
6: zergoto koniec
7: loada p
8: mul x
9: store p
10: incr i
11: goto pętla
koniec: 12: stop

13: # x -- podstawa
14: # n -- wykładnik
15: # p -- wynik
16: # i -- pomocnicza

END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```
0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10:  incr i
11:  goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END
```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul 13
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul 13
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada 14
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul 13
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada 14
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul 13
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store i
pętla: 4:  loada 14
5:  sub i
6:  zergoto koniec
7:  loada 15
8:  mul 13
9:  store 15
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store i
pętla: 4:  loada 14
5:  sub i
6:  zergoto koniec
7:  loada 15
8:  mul 13
9:  store 15
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
pętla: 4:  loada 14
5:  sub 16
6:  zergoto koniec
7:  loada 15
8:  mul 13
9:  store 15
10: incr 16
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:   loadn 1
1:   store p
2:   loadn 0
3:   store i
pętla: 4:   loada n
5:   sub i
6:   zergoto koniec
7:   loada p
8:   mul x
9:   store p
10:  incr i
11:  goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:   loadn 1
1:   store 15
2:   loadn 0
3:   store 16
pętla: 4:   loada 14
5:   sub 16
6:   zergoto koniec
7:   loada 15
8:   mul 13
9:   store 15
10:  incr 16
11:  goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:  sub i
6:  zergoto koniec
7:  loada p
8:  mul x
9:  store p
10: incr i
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
pętla: 4:  loada 14
5:  sub 16
6:  zergoto 12
7:  loada 15
8:  mul 13
9:  store 15
10: incr 16
11: goto pętla
koniec: 12: stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:      sub i
6:      zergoto koniec
7:      loada p
8:      mul x
9:      store p
10:     incr i
11:     goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
pętla: 4:  loada 14
5:      sub 16
6:      zergoto 12
7:      loada 15
8:      mul 13
9:      store 15
10:     incr 16
11:     goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:   loadn 1
1:   store p
2:   loadn 0
3:   store i
pętla: 4:   loada n
5:   sub i
6:   zergoto koniec
7:   loada p
8:   mul x
9:   store p
10:  incr i
11:  goto pętla
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```



```

0:   loadn 1
1:   store 15
2:   loadn 0
3:   store 16
pętla: 4:   loada 14
5:   sub 16
6:   zergoto 12
7:   loada 15
8:   mul 13
9:   store 15
10:  incr 16
11:  goto 4
koniec: 12:  stop
13:      # x -- podstawa
14:      # n -- wykładnik
15:      # p -- wynik
16:      # i -- pomocnicza
      END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:      sub i
6:      zergoto koniec
7:      loada p
8:      mul x
9:      store p
10:     incr i
11:     goto pętla
koniec: 12:    stop
13:         # x -- podstawa
14:         # n -- wykładnik
15:         # p -- wynik
16:         # i -- pomocnicza
          END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
4:  loada 14    # pętla
5:  sub 16
6:  zergoto 12
7:  loada 15
8:  mul 13
9:  store 15
10: incr 16
11: goto 4
12: stop        # koniec
13:         # x -- podstawa
14:         # n -- wykładnik
15:         # p -- wynik
16:         # i -- pomocnicza
          END

```

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:      sub i
6:      zergoto koniec
7:      loada p
8:      mul x
9:      store p
10:     incr i
11:     goto pętla
koniec: 12:    stop
13:         # x -- podstawa
14:         # n -- wykładnik
15:         # p -- wynik
16:         # i -- pomocnicza
          END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
4:  loada 14    # pętla
5:  sub 16
6:  zergoto 12
7:  loada 15
8:  mul 13
9:  store 15
10: incr 16
11: goto 4
12: stop        # koniec
13:         # x -- podstawa
14:         # n -- wykładnik
15:         # p -- wynik
16:         # i -- pomocnicza
          END

```

514 == ?

Jak programować w języku niskiego poziomu?

zamiana adresów symbolicznych na fizyczne
(**zamiana na numery**)

```

0:  loadn 1
1:  store p
2:  loadn 0
3:  store i
pętla: 4:  loada n
5:      sub i
6:      zergoto koniec
7:      loada p
8:      mul x
9:      store p
10:     incr i
11:     goto pętla
koniec: 12:    stop
13:         # x -- podstawa
14:         # n -- wykładnik
15:         # p -- wynik
16:         # i -- pomocnicza
          END

```



```

0:  loadn 1
1:  store 15
2:  loadn 0
3:  store 16
4:  loada 14    # pętla
5:  sub 16
6:  zergoto 12
7:  loada 15
8:  mul 13
9:  store 15
10: incr 16
11: goto 4
12: stop        # koniec
13:  5          # x -- podstawa
14: 14          # n -- wykładnik
15:  0          # p -- wynik
16:  0          # i -- pomocnicza
          END

```

514 == ?