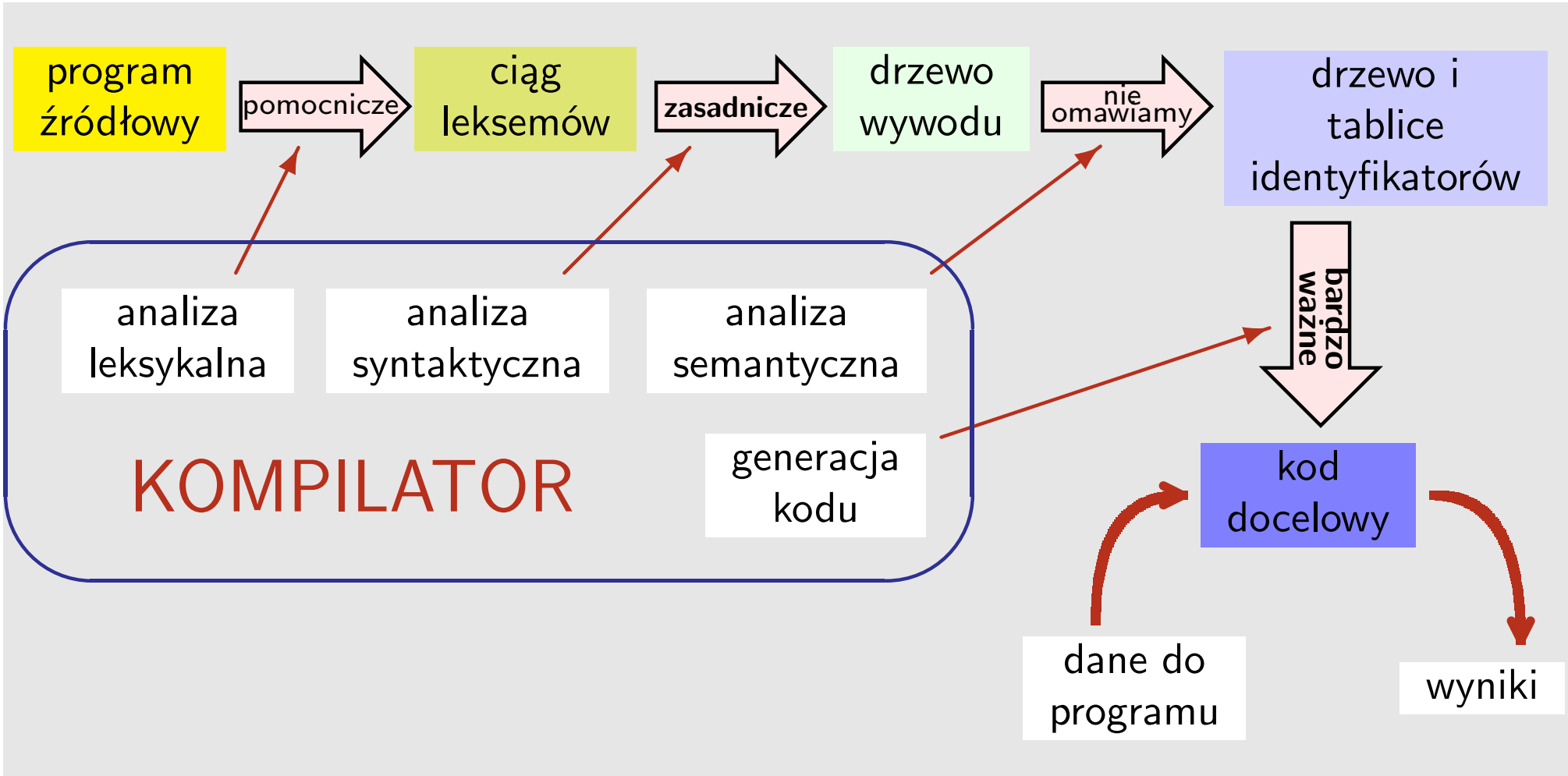
Stefan Sokołowski

JĘZYKI FORMALNE I METODY KOMPILACJI

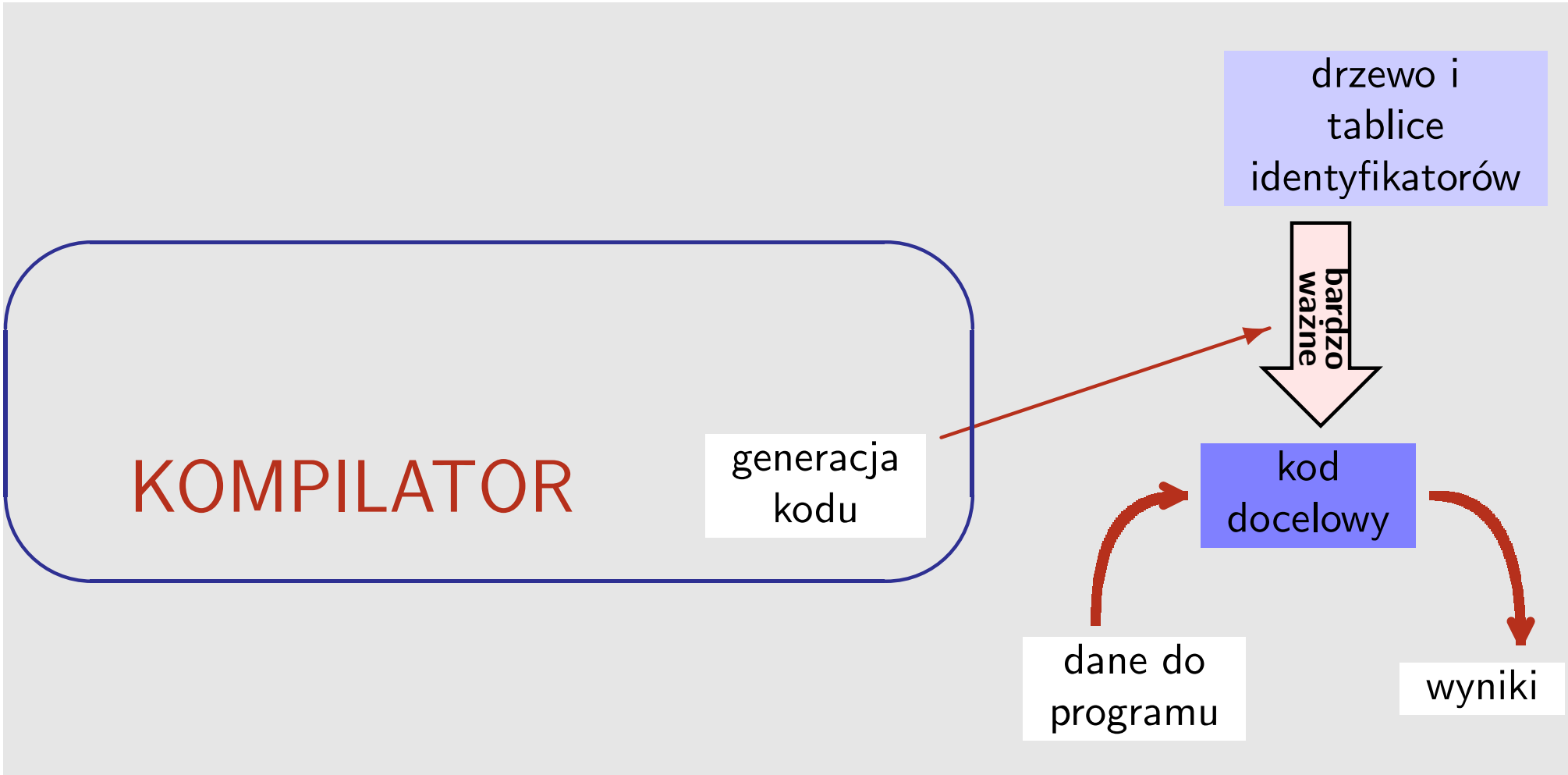
wykład 9

Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025

Uproszczony schemat kompilacji

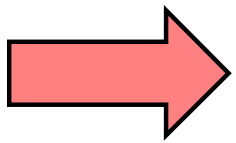


Uproszczony schemat kompilacji



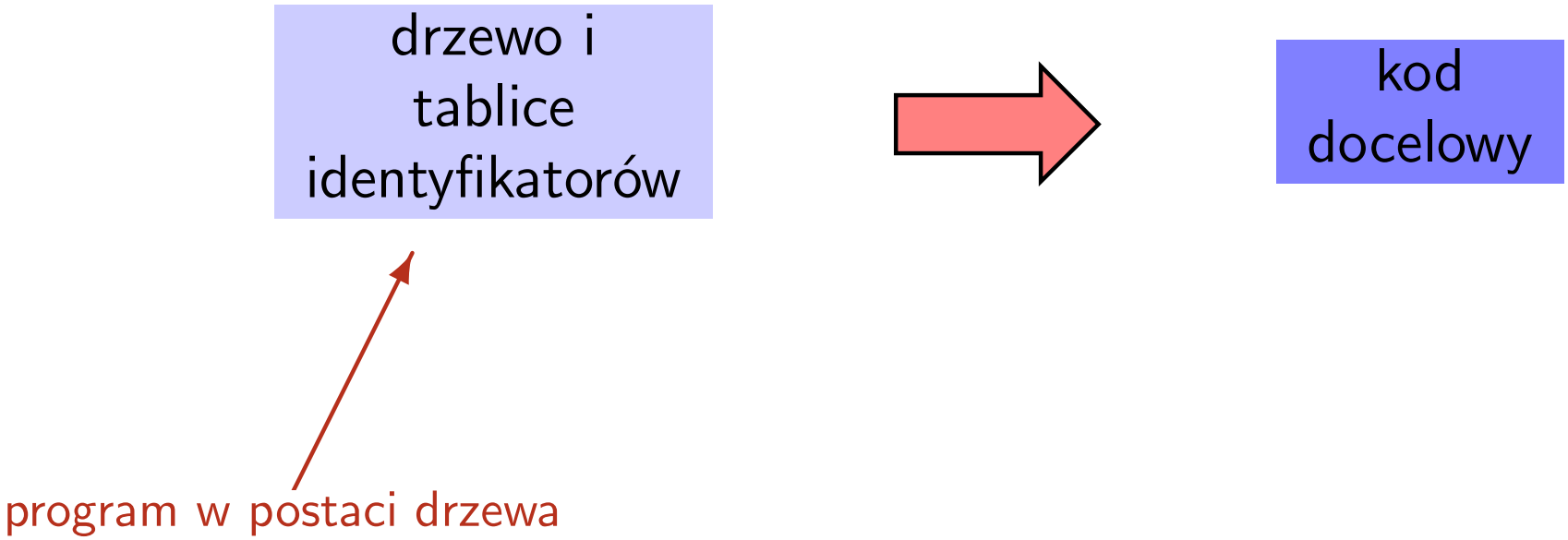
Kompilacja

drzewo i tablice identyfikatorów

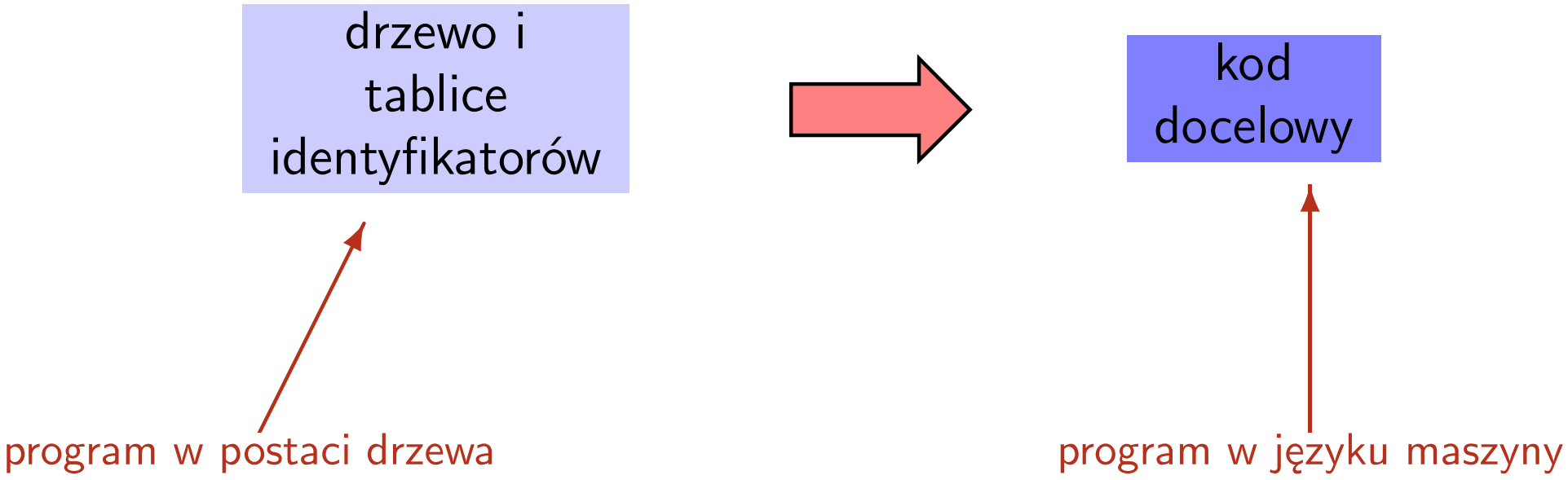


kod docelowy

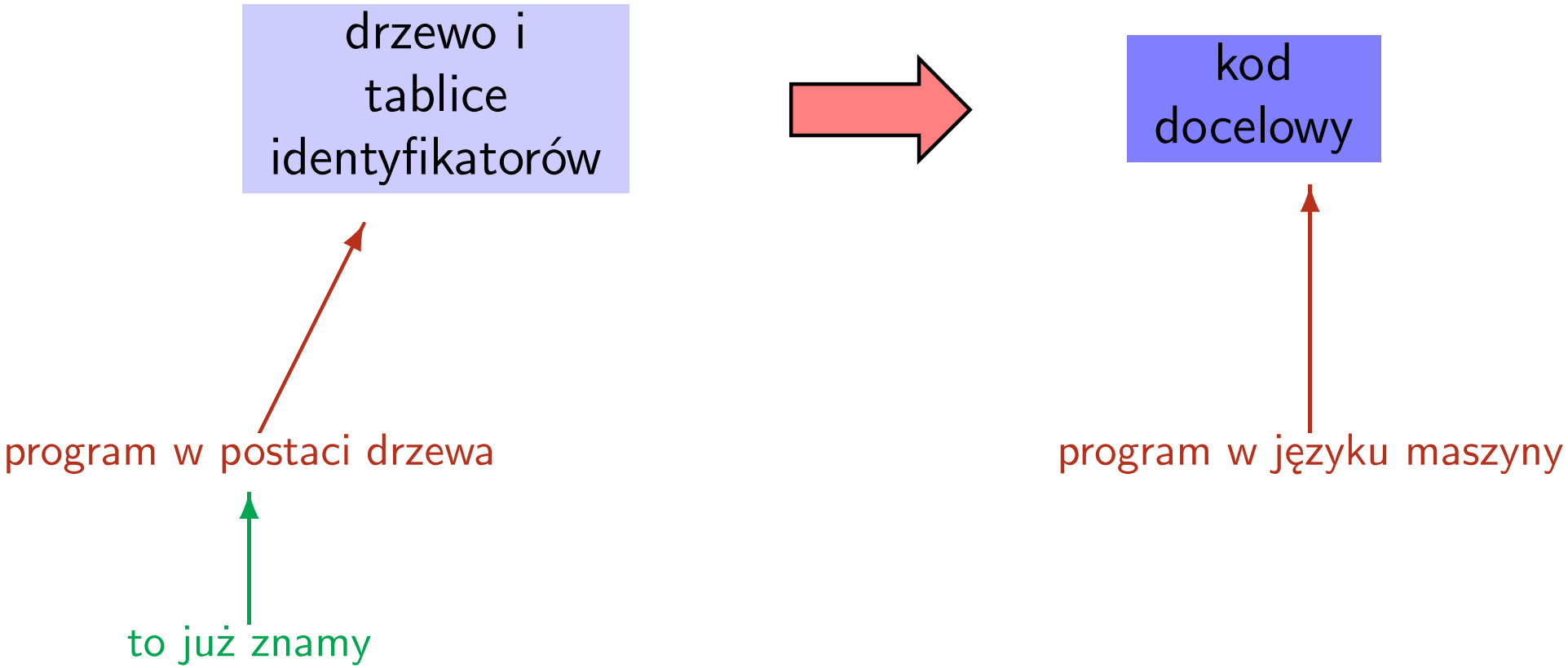
Kompilacja


drzewo i
tablice
identyfikatorów
kod
docelowy
program w postaci drzewa

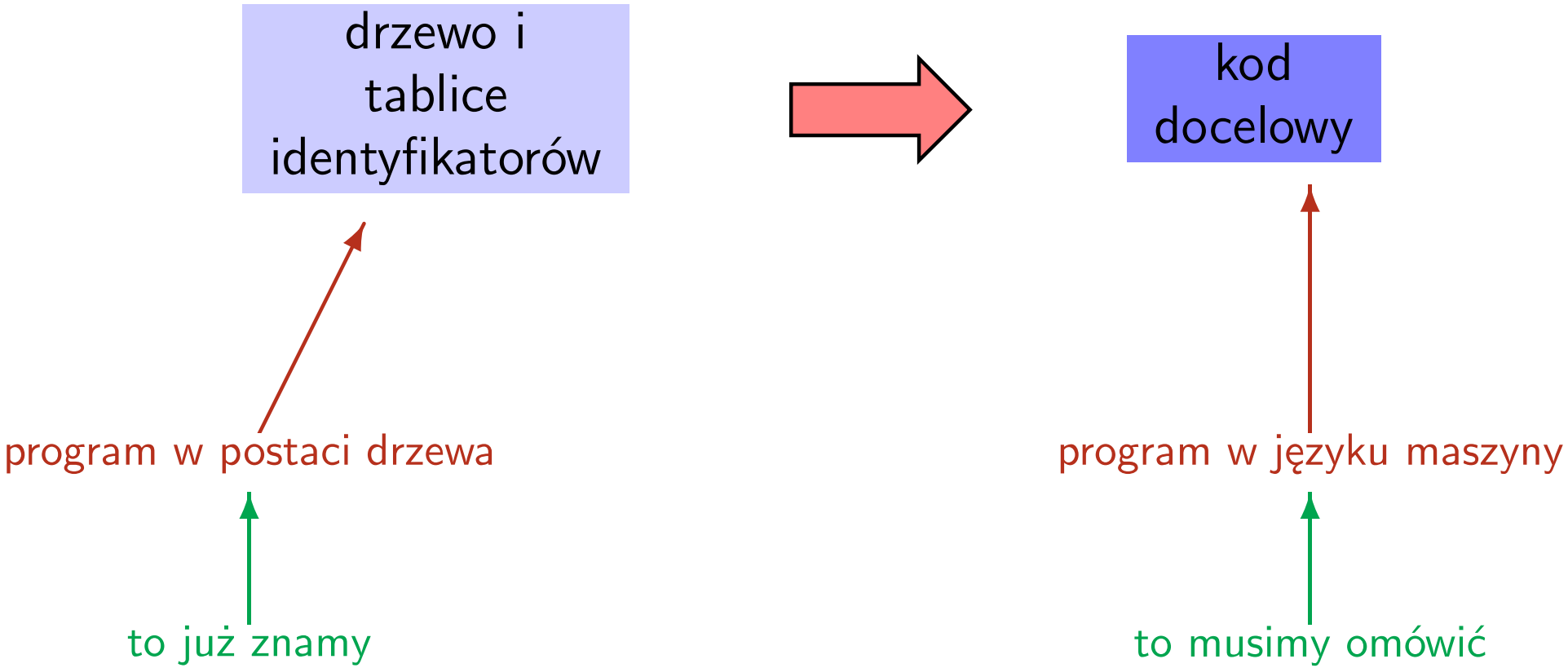
Kompilacja


drzewo i
tablice
identyfikatorów
kod
docelowy
program w postaci drzewa
program w języku maszyny

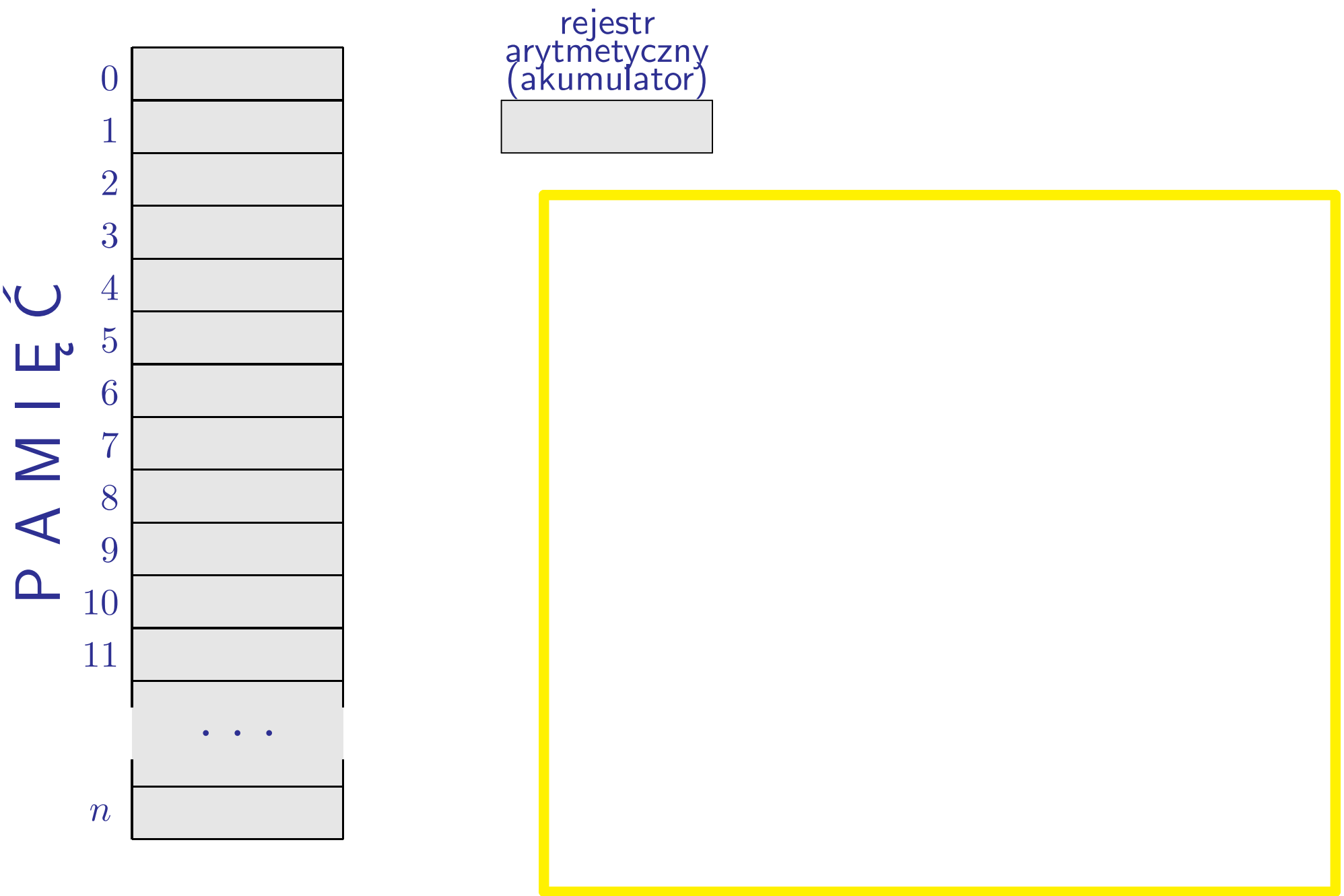
Kompilacja


drzewo i
tablice
identyfikatorów
kod
docelowy
program w postaci drzewa
program w języku maszyny
to już znamy

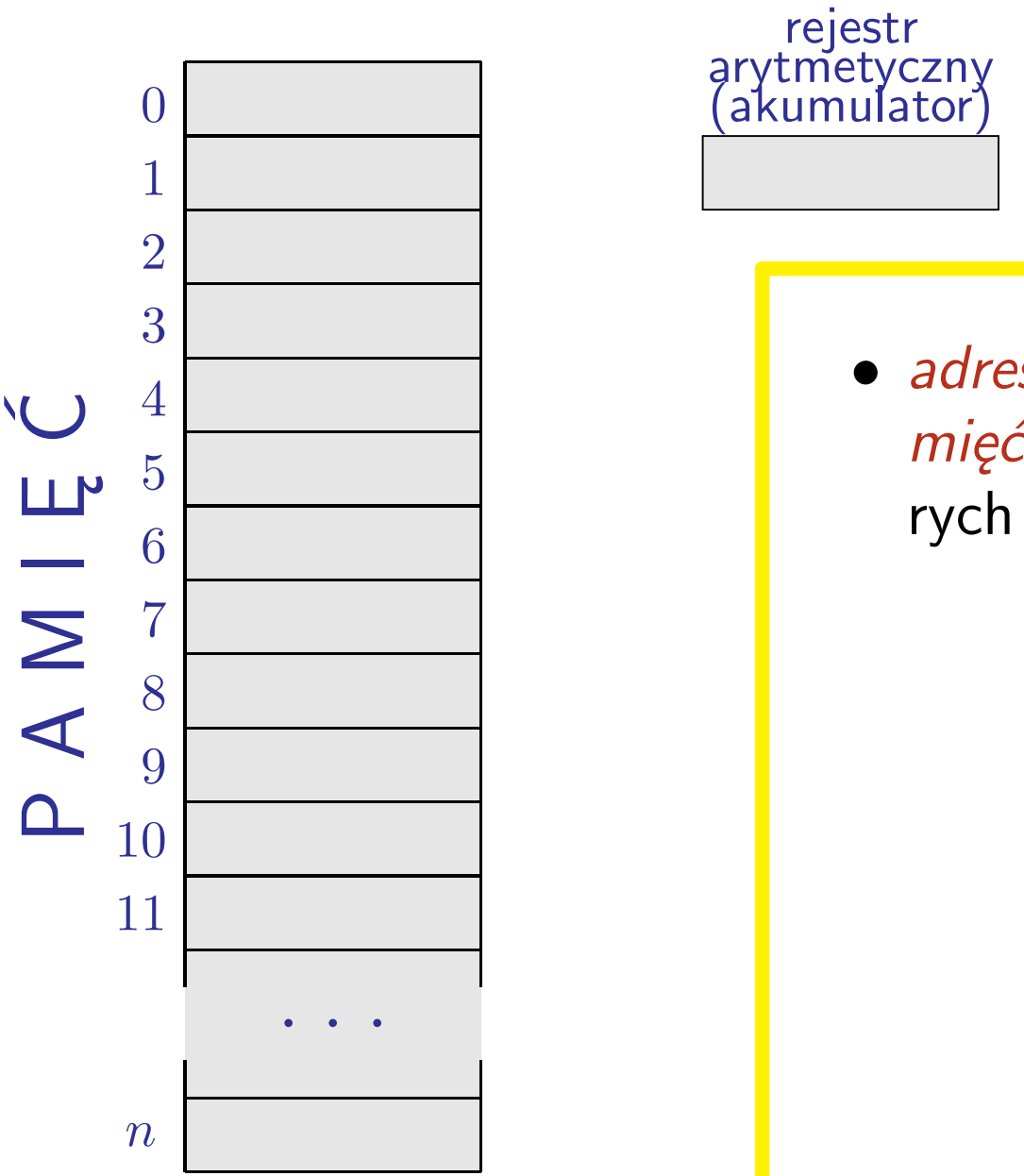
Kompilacja


drzewo i
tablice
identyfikatorów
kod
docelowy
program w postaci drzewa
program w języku maszyny
to już znamy
to musimy omówić

Programowanie niskiego poziomu

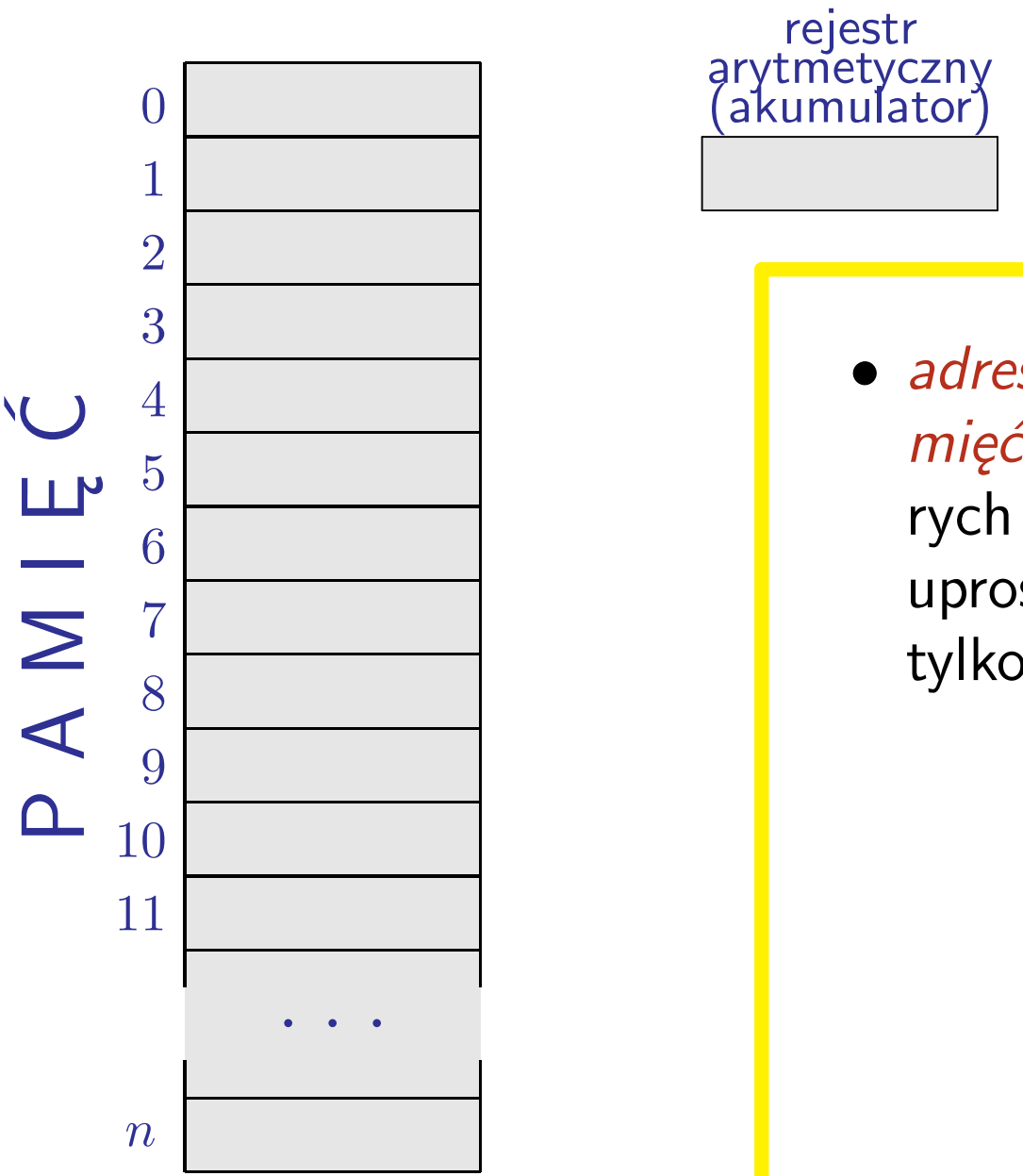


Programowanie niskiego poziomu



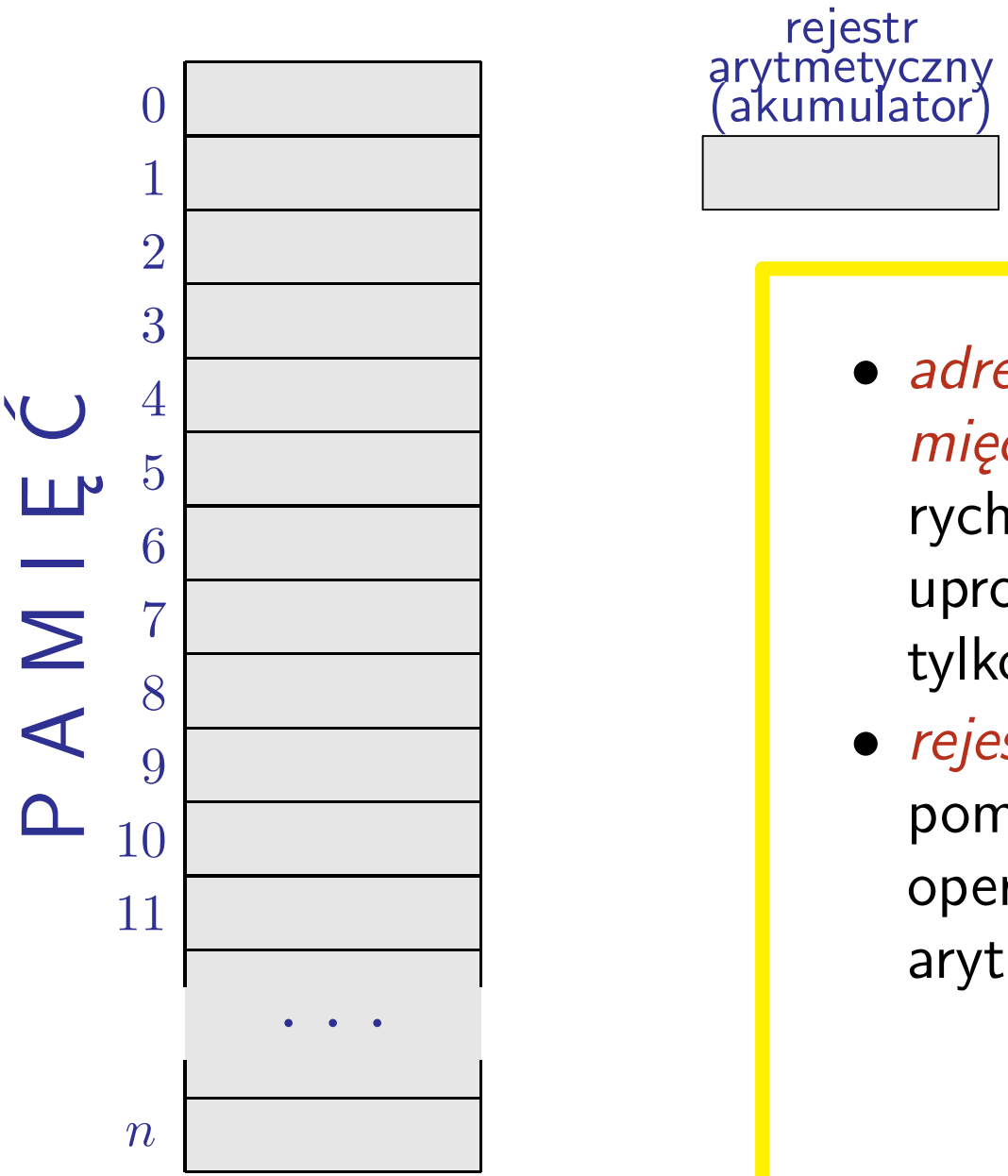
- *adresowana* (czyli numerowana) *pamięć* złożona z *komórek*, z których każda może pomieścić bajt

Programowanie niskiego poziomu



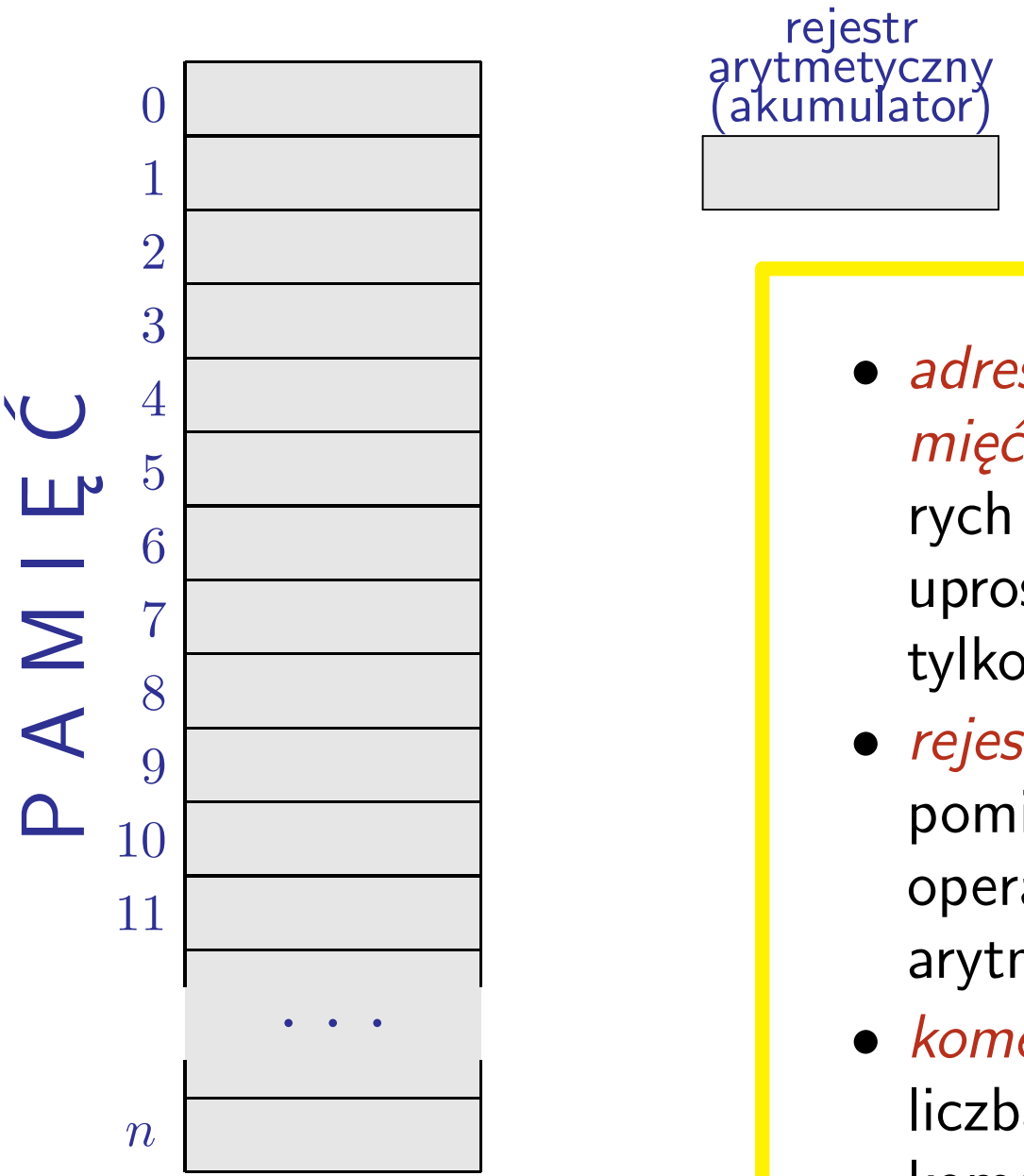
- *adresowana* (czyli numerowana) *pamięć* złożona z *komórek*, z których każda może pomieścić bajt; dla uproszczenia zakładam, że nie bajt tylko liczbę całkowitą

Programowanie niskiego poziomu



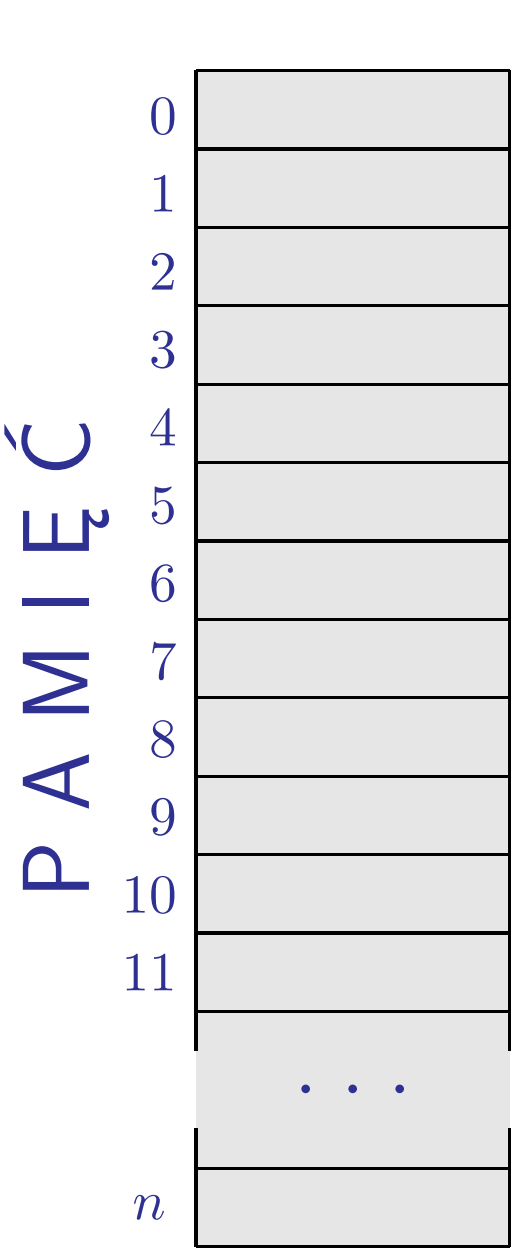
- *adresowana* (czyli numerowana) *pamięć* złożona z *komórek*, z których każda może pomieścić bajt; dla uproszczenia zakładam, że nie bajt tylko liczbę całkowitą;
- *rejestr arytmetyczny* także mogący pomieścić liczbę całkowitą; większość operacji odbywa się między rejestrem arytmetycznym a komórką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

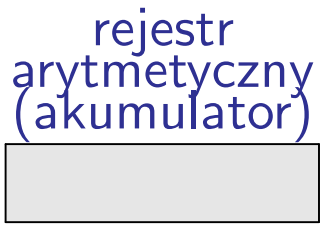
Programowanie niskiego poziomu



- *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.

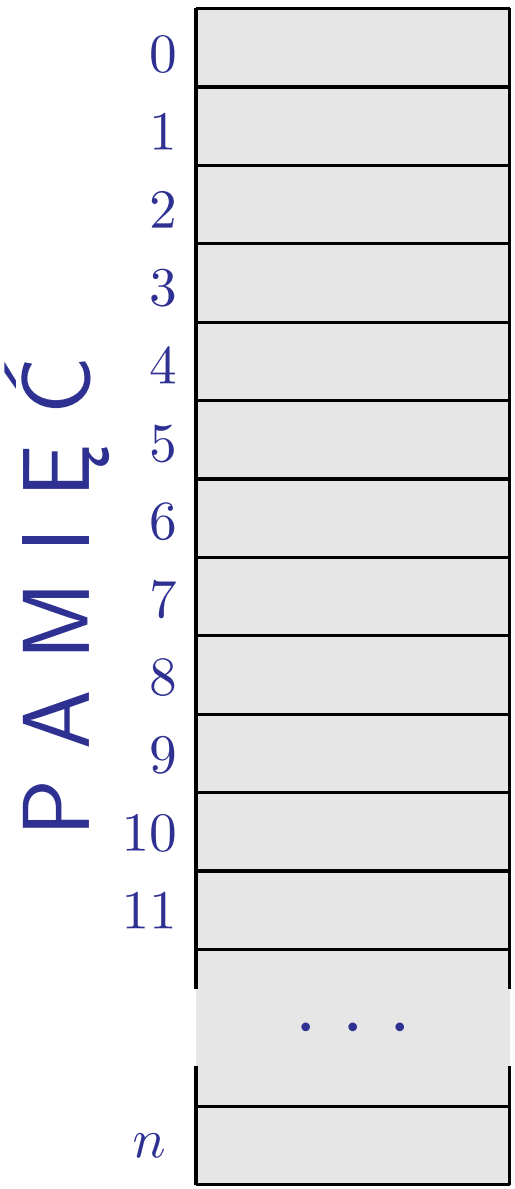
Programowanie niskiego poziomu

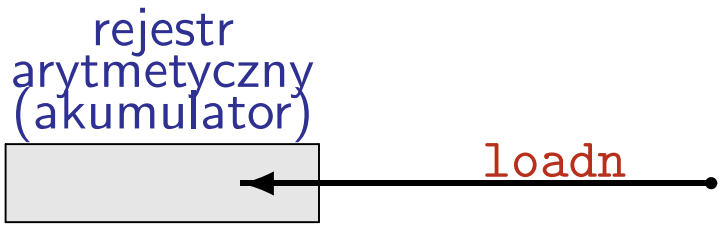




KOMENDY PRZESYŁANIA

Programowanie niskiego poziomu



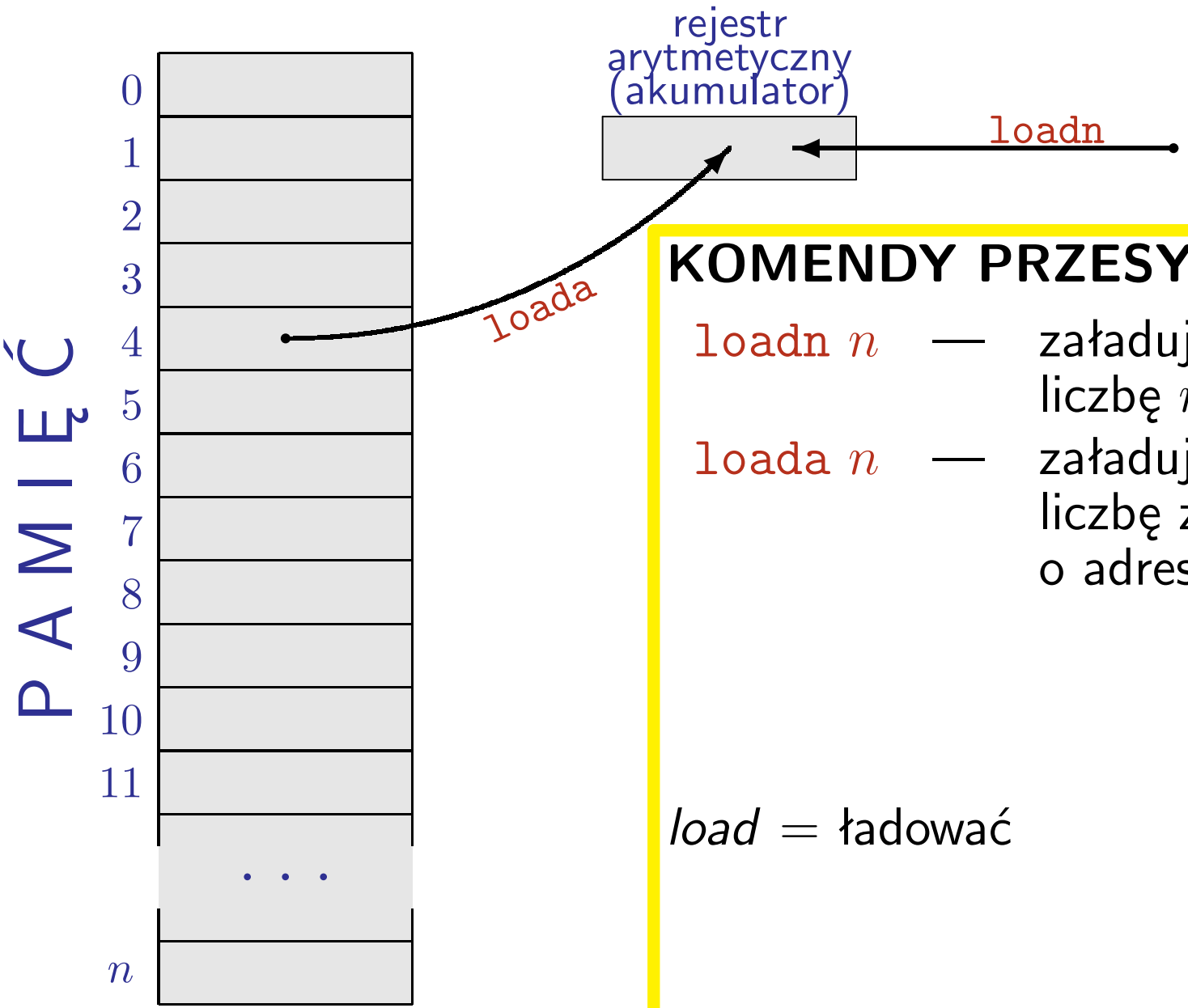


KOMENDY PRZESYŁANIA:

`loadn` n — załaduj do akumulatora liczbę n

load = ładować

Programowanie niskiego poziomu



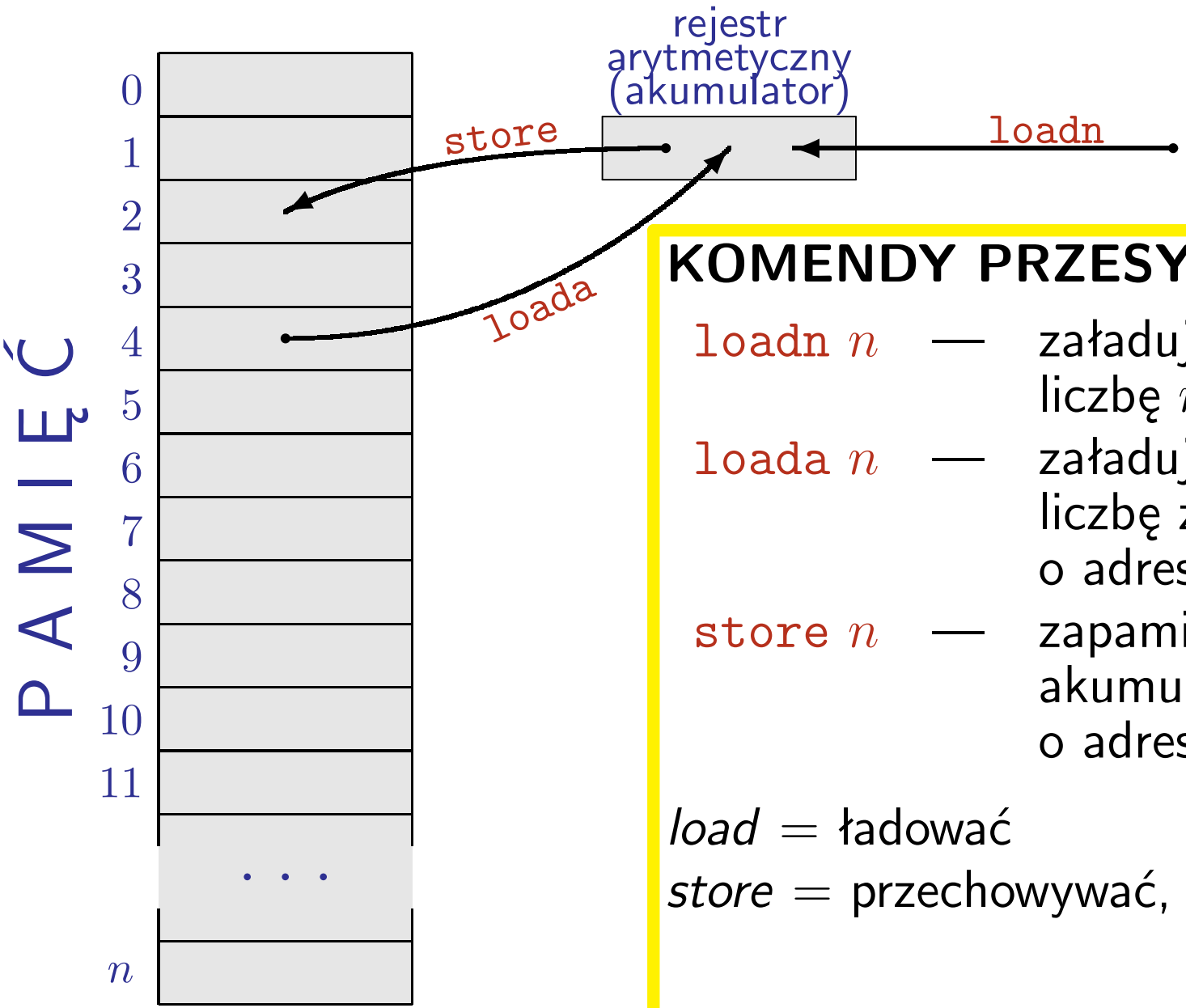
KOMENDY PRZESYŁANIA:

`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

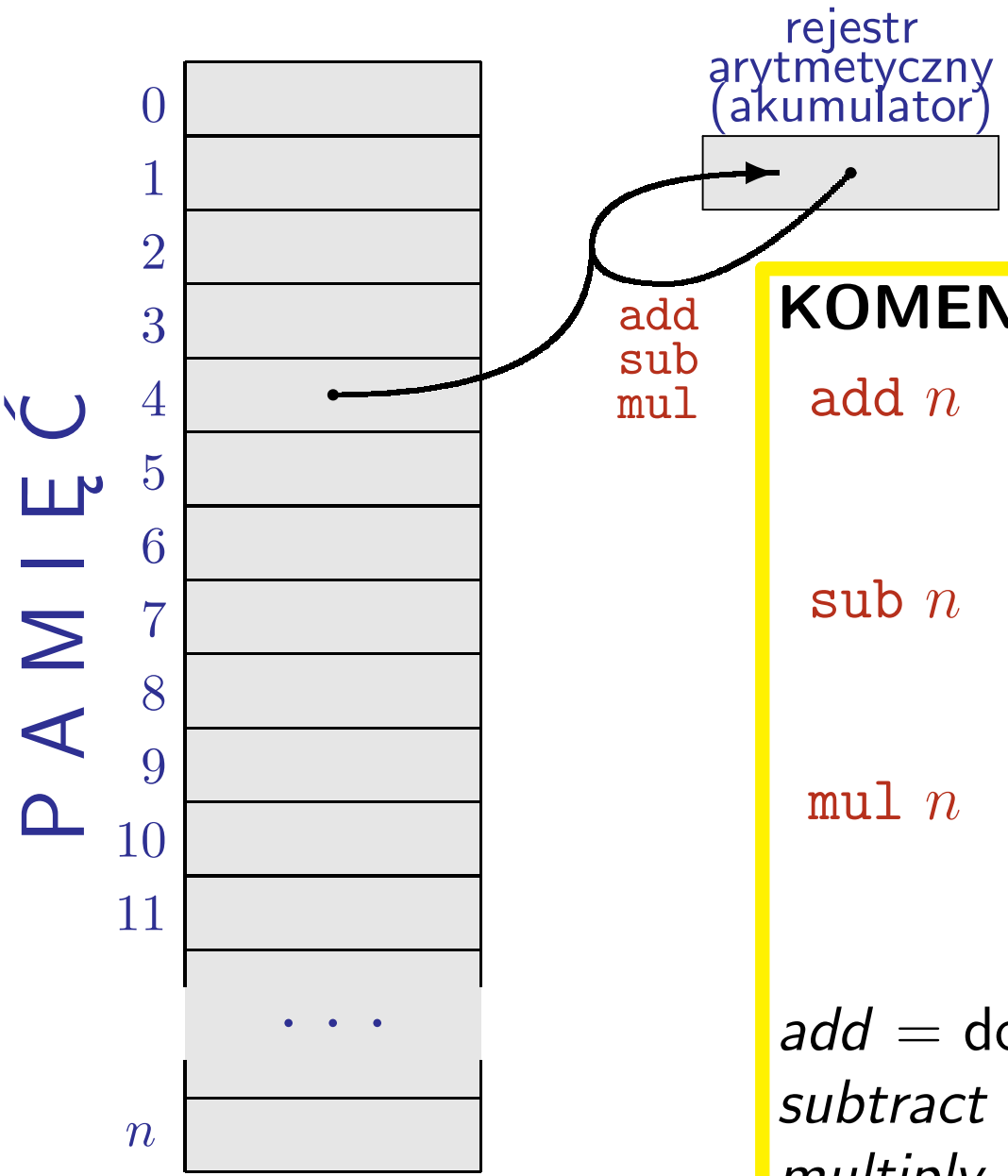
`loada` n — załaduj do akumulatora liczbę z komórki o adresie n

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

load = ładować

store = przechowywać, magazynować

Programowanie niskiego poziomu



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

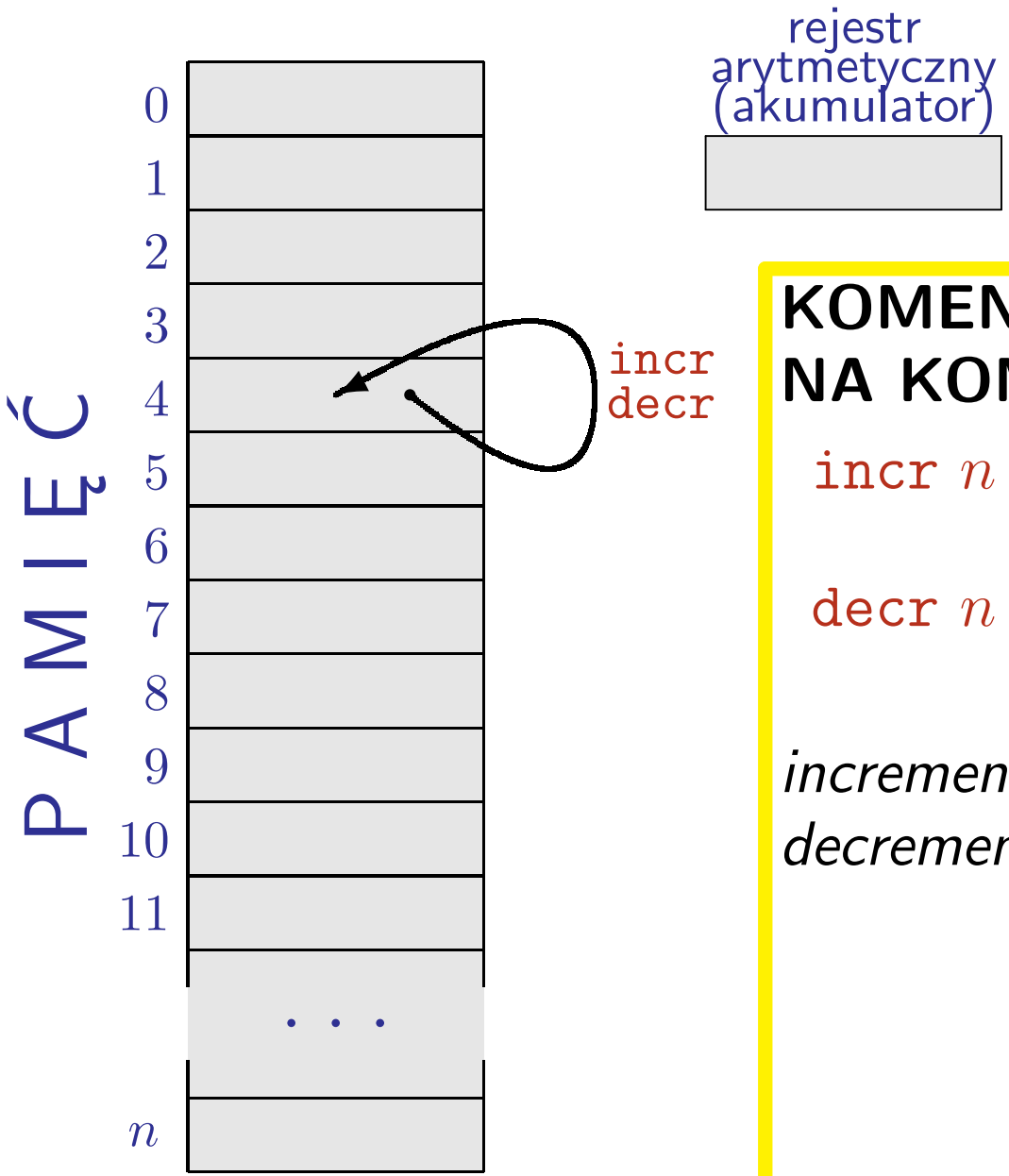
`mul` n — pomnóż akumulator przez liczbę z kom. o adresie n; wynik w akumulatorze

add = dodawać

subtract = odejmować

multiply = mnożyć

Programowanie niskiego poziomu



KOMENDY DZIAŁAŃ NA KOMÓRKACH:

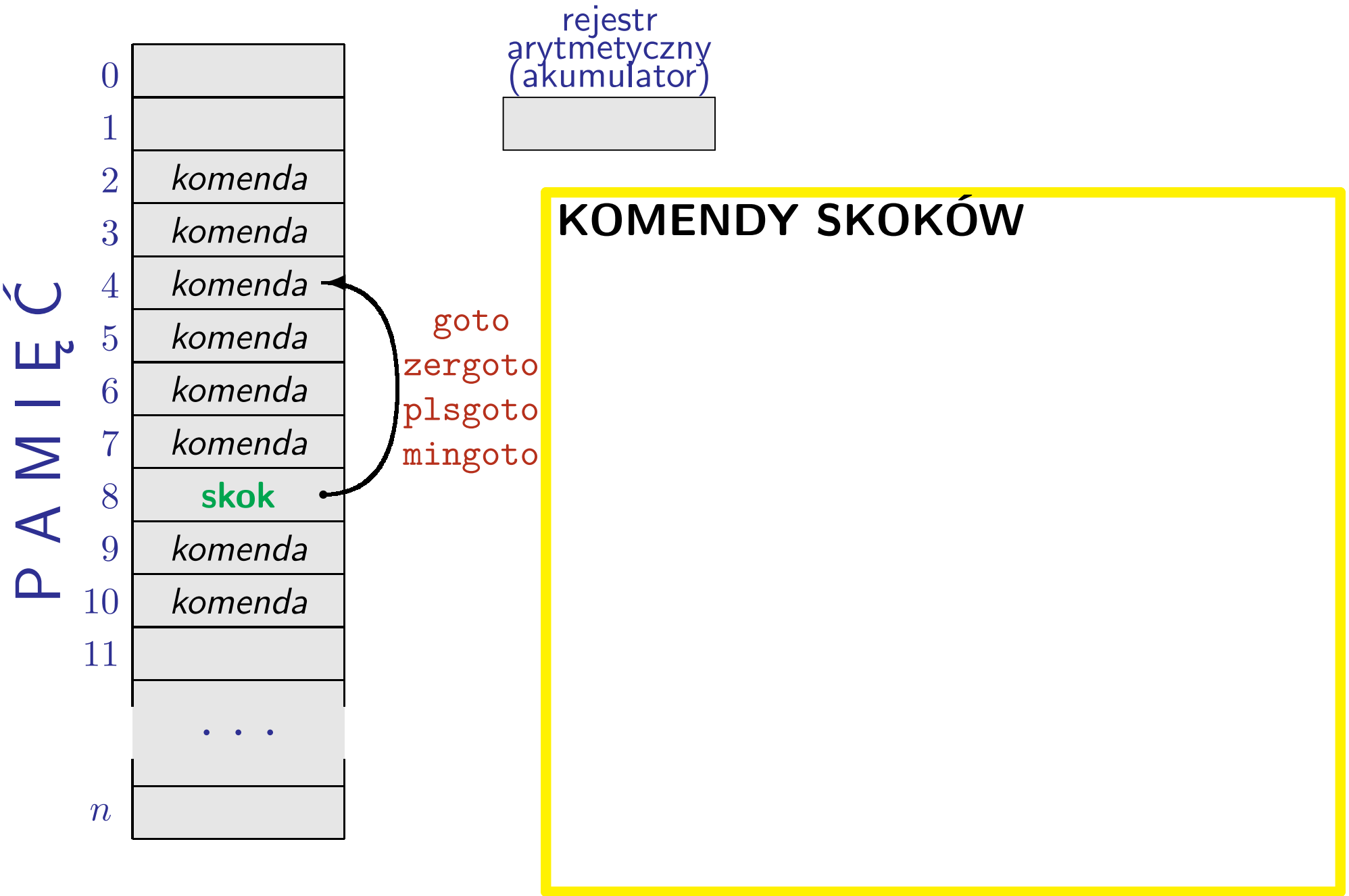
`incr` n — powiększ o 1 zawartość kom. o adresie n

`decr` n — zmniejsz o 1 zawartość kom. o adresie n

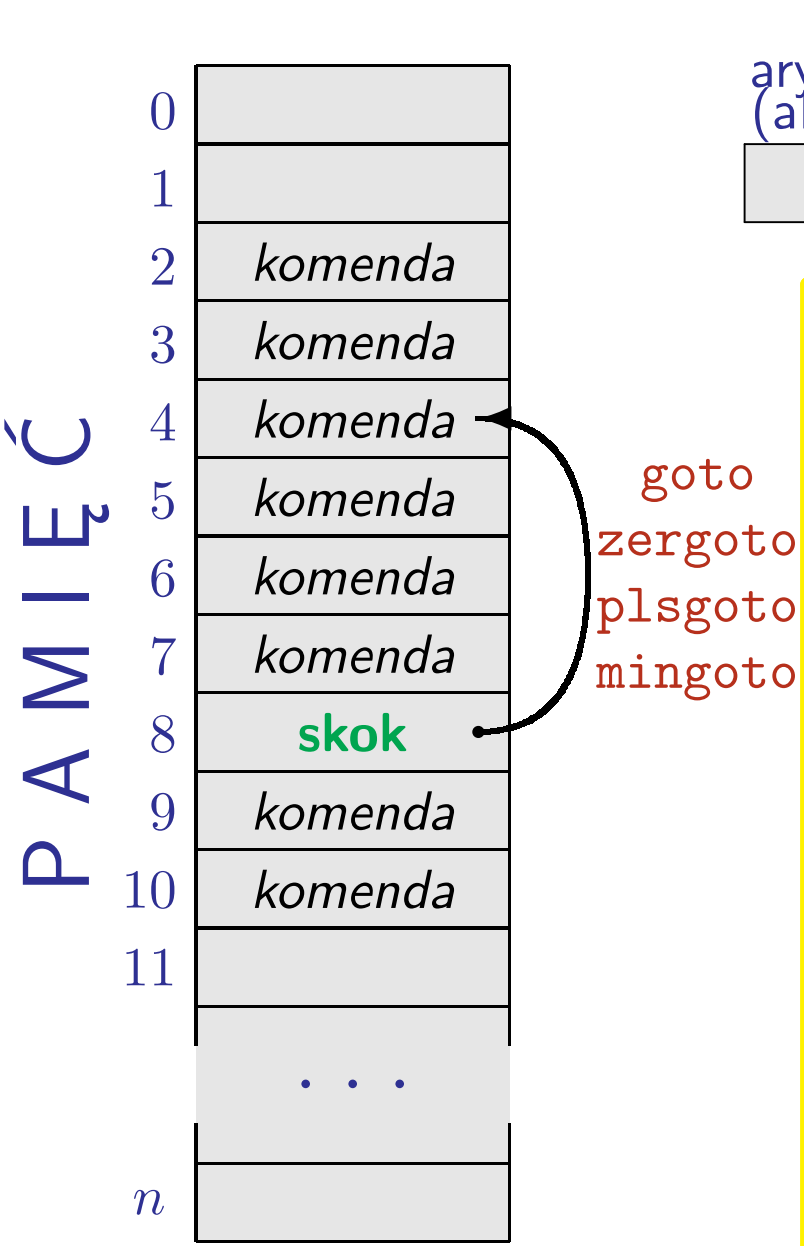
increment = zwiększać

decrement = zmniejszać

Programowanie niskiego poziomu



Programowanie niskiego poziomu

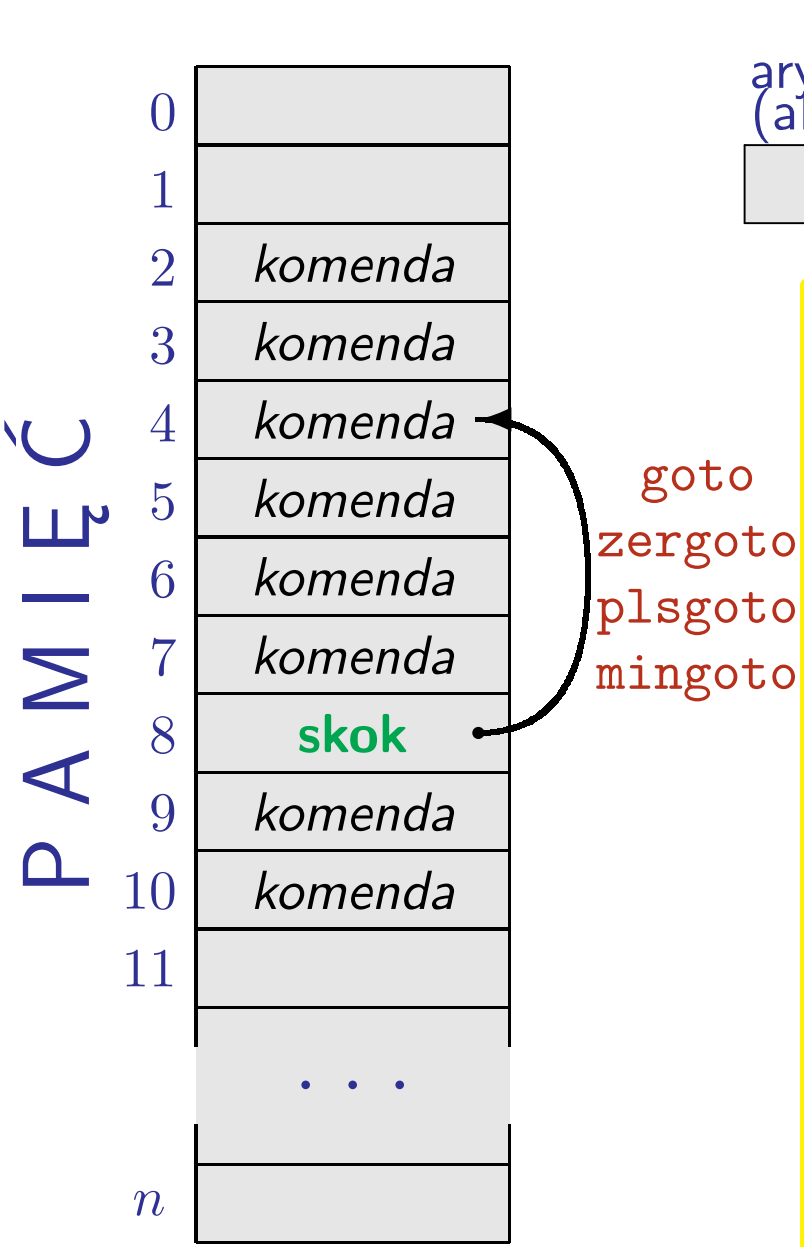


KOMENDY SKOKÓW:

`goto` n — przejdź do komórki o adresie n

go to = idź do

Programowanie niskiego poziomu

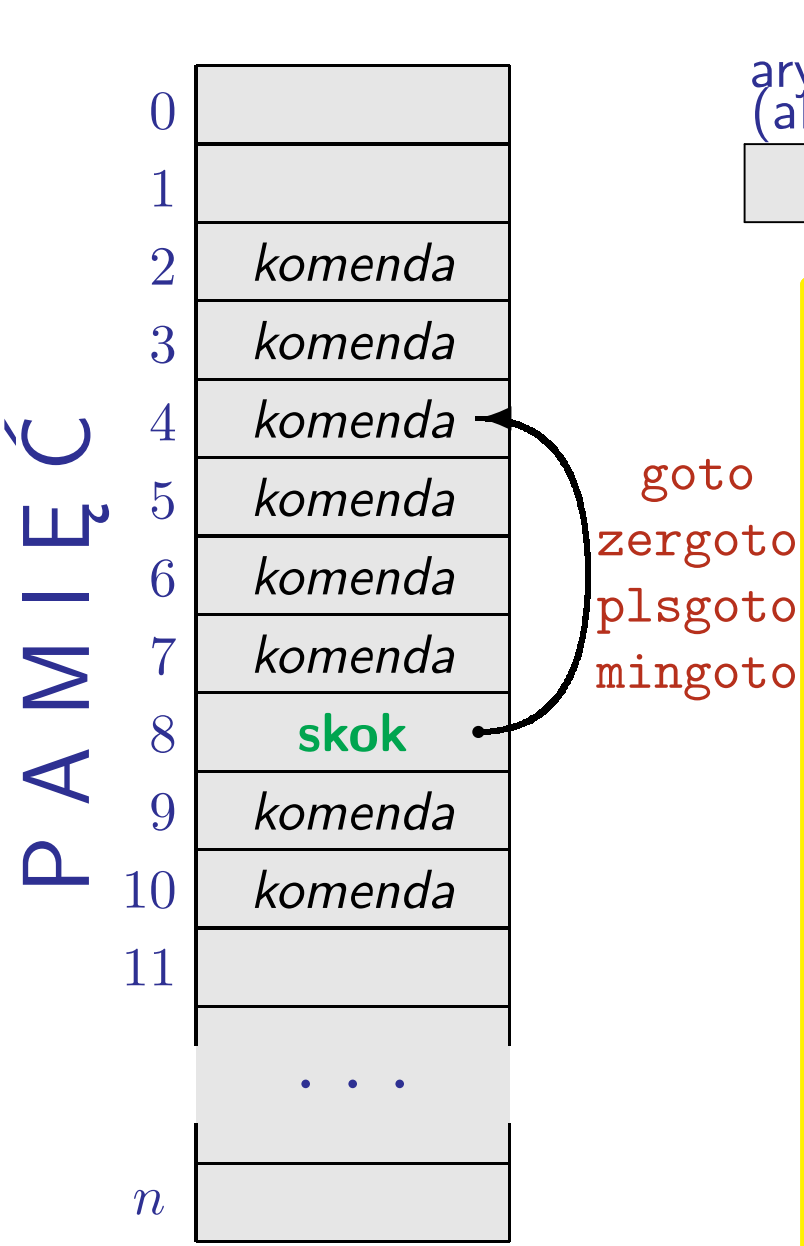


KOMENDY SKOKÓW:

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

go to = idź do

Programowanie niskiego poziomu

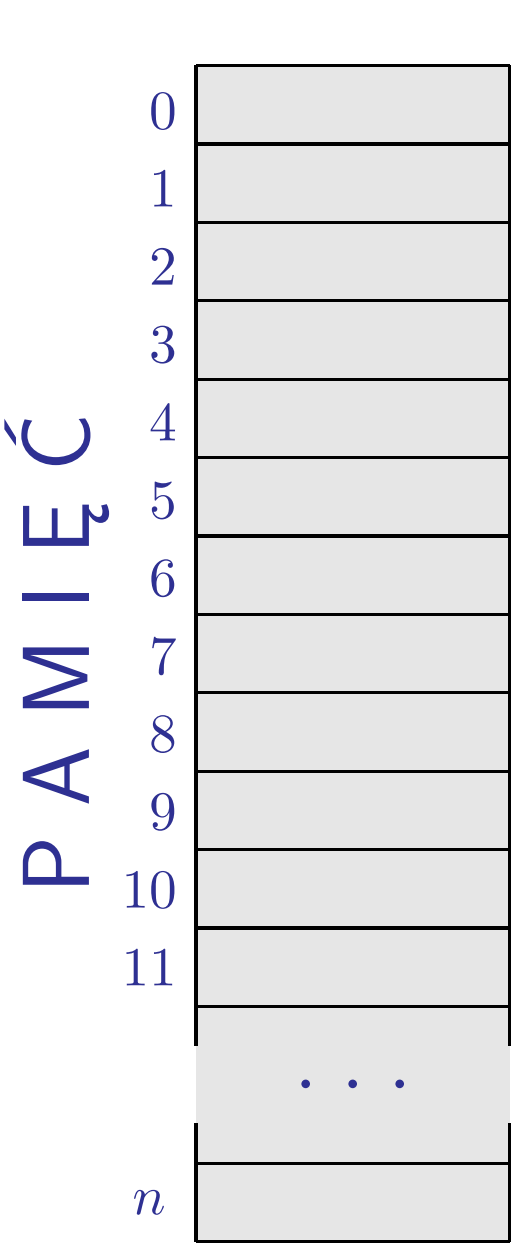


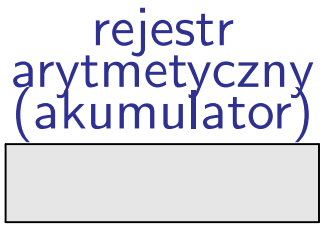
KOMENDY SKOKÓW:

`goto` n	—	przejdź do komórki o adresie n
`zergoto` n	—	jeśli w akumul. zero, to przejdź do kom. o adresie n
`plsgoto` n	—	jeśli akum. dodatni, to przejdź do kom. o adresie n
`mingoto` n	—	jeśli akum. ujemny, to przejdź do kom. o adresie n
`stop`	—	zatrzymaj działanie

go to = idź do

Programowanie niskiego poziomu



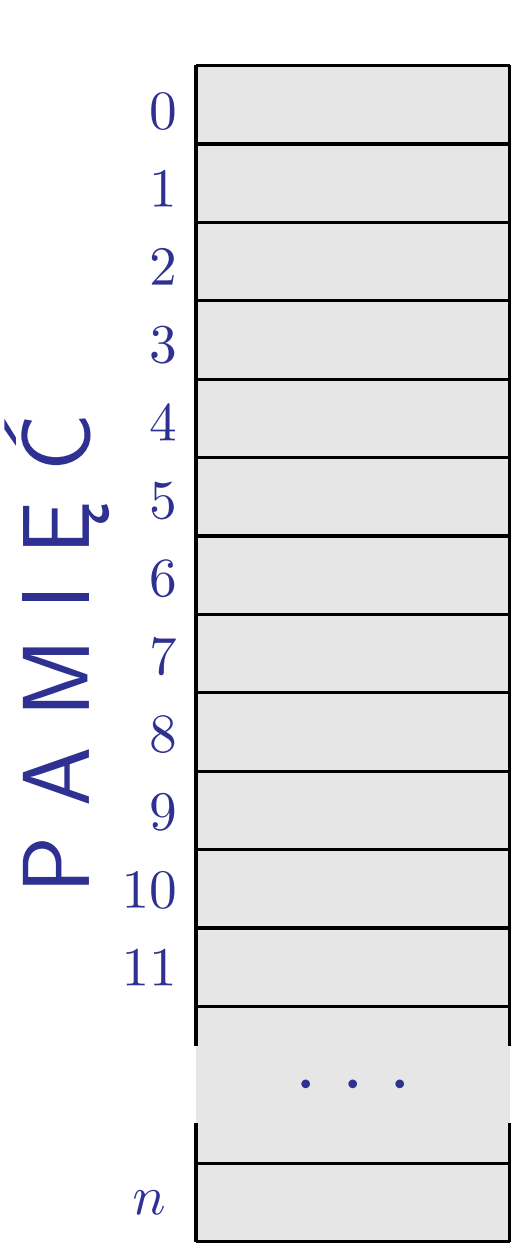


PRZYKŁAD:

Obliczenie wartości wielomianu

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

Programowanie niskiego poziomu



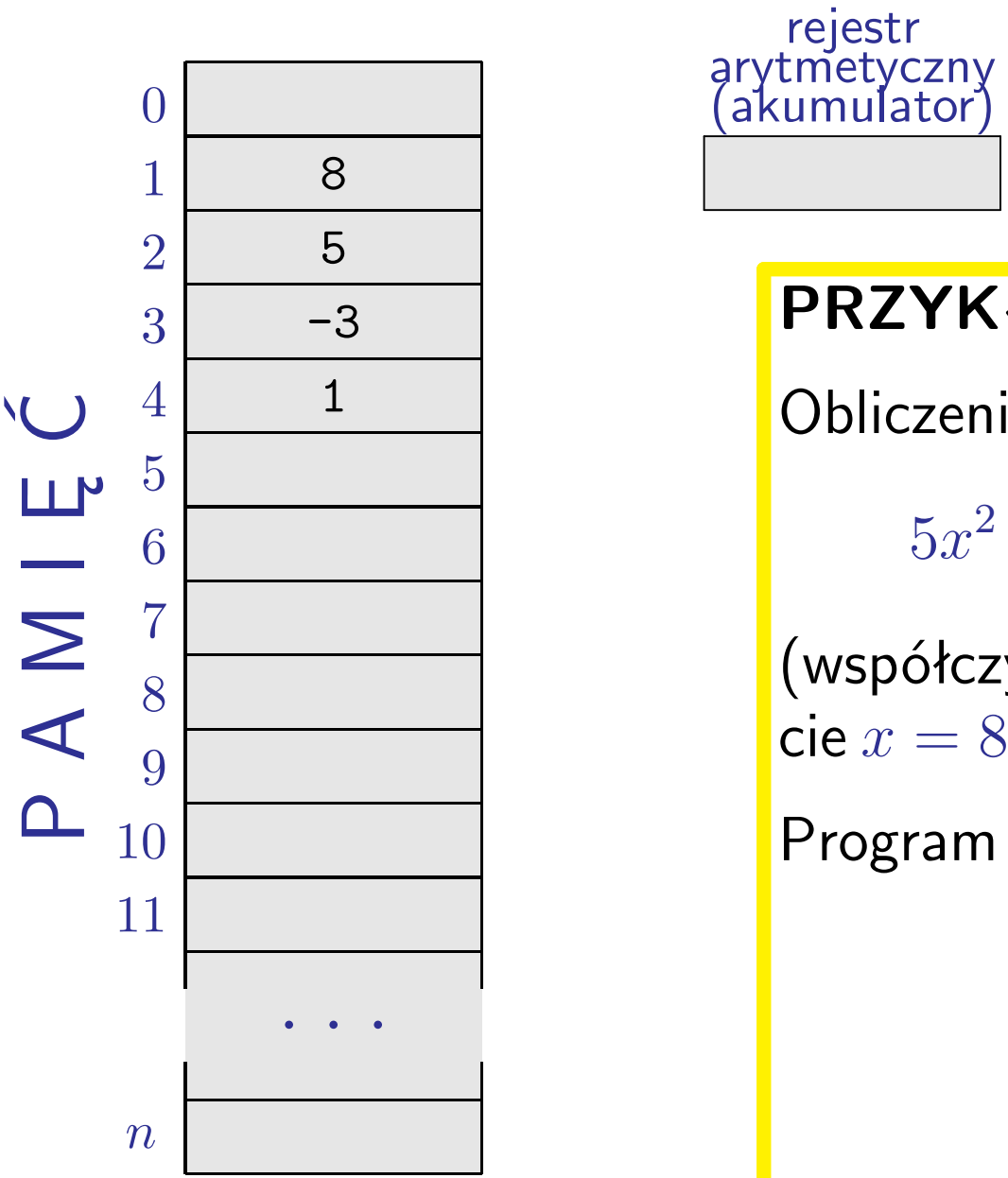
rejestr arytmetyczny (akumulator)

PRZYKŁAD:

Obliczenie wartości wielomianu

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

Programowanie niskiego poziomu



PRZYKŁAD:

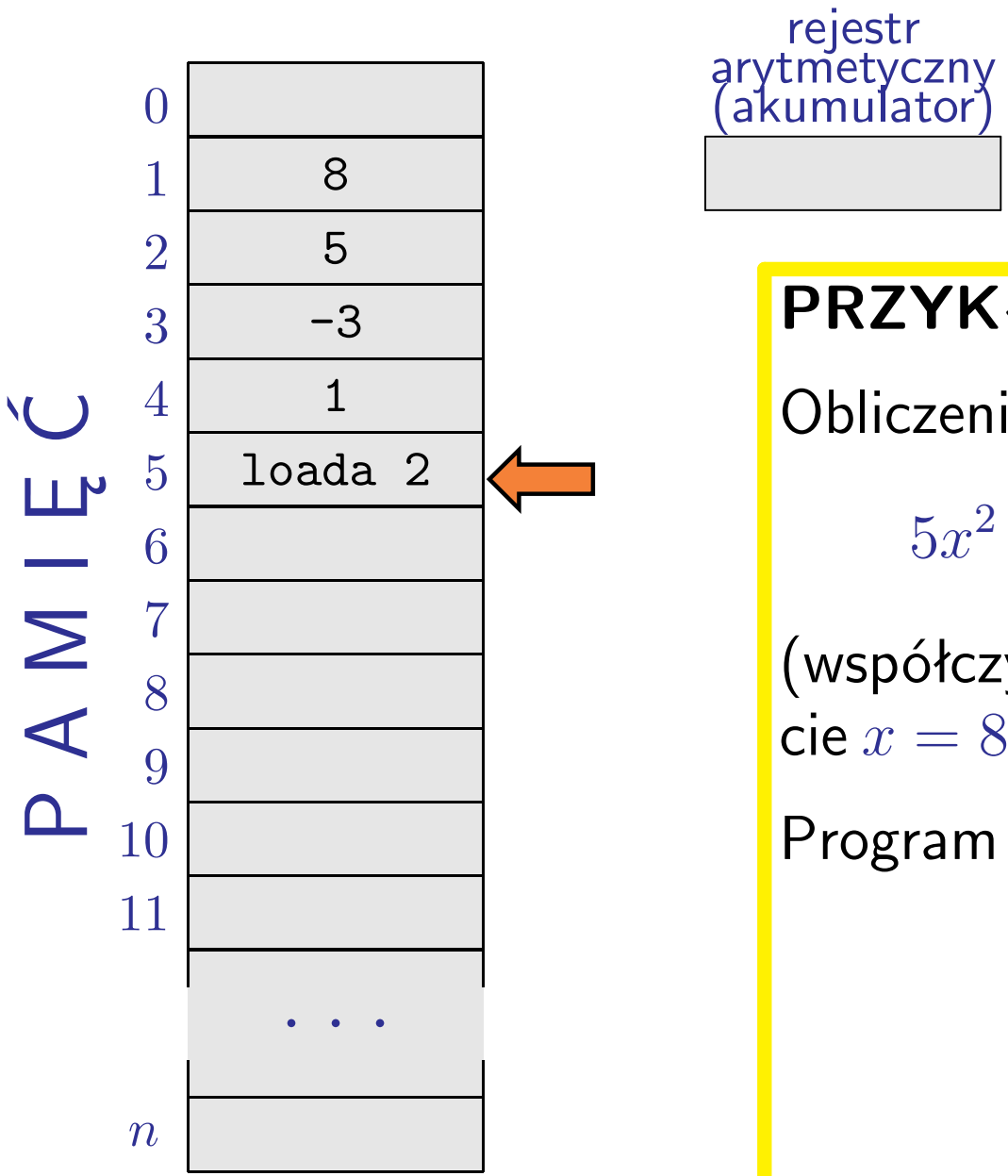
Obliczenie wartości wielomianu

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

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

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



PRZYKŁAD:

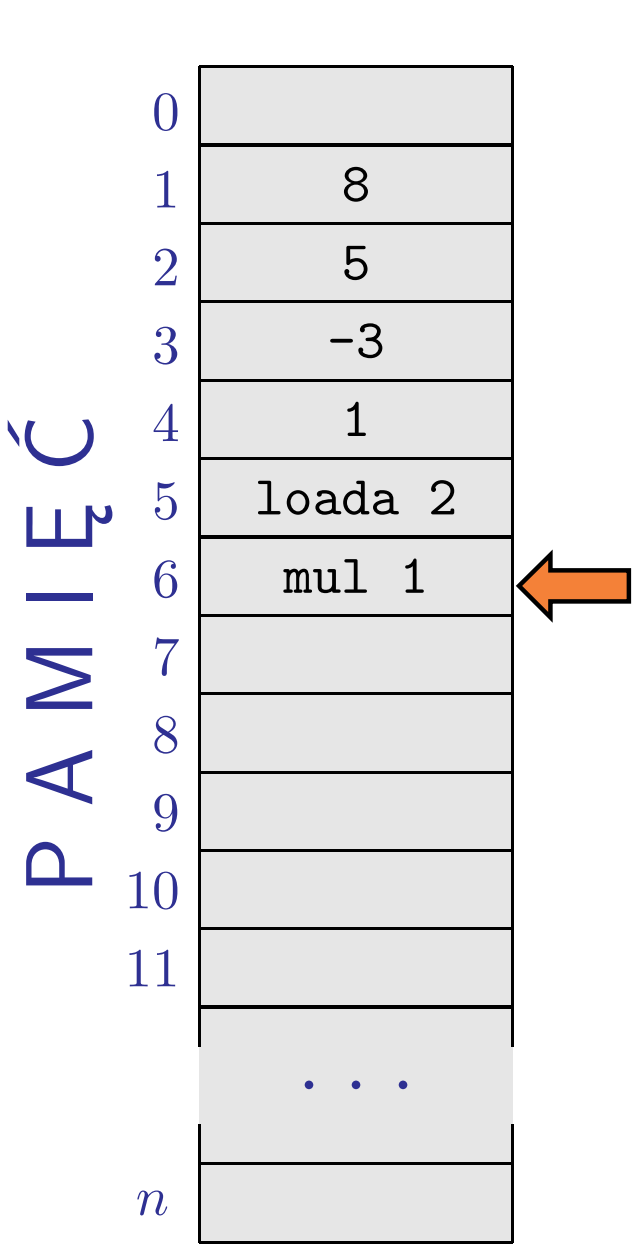
Obliczenie wartości wielomianu

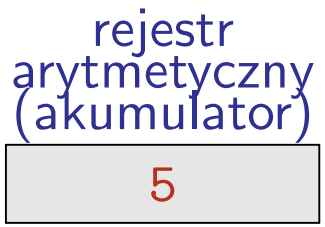
$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

(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:

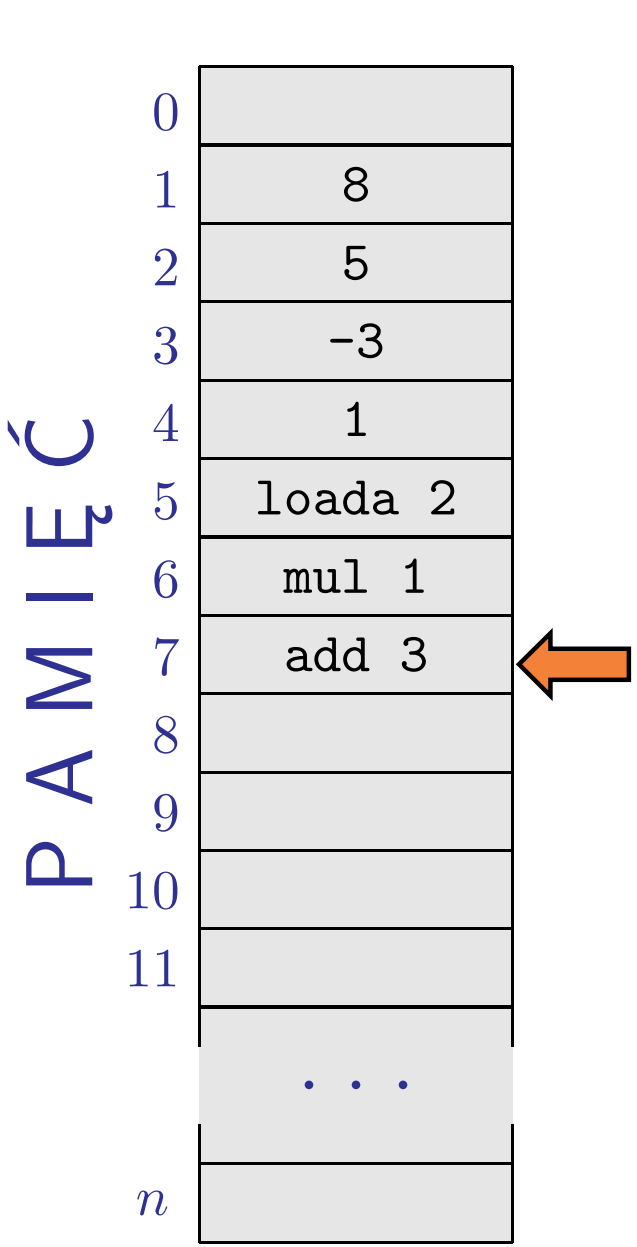
Obliczenie wartości wielomianu

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

(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)

40

PRZYKŁAD:

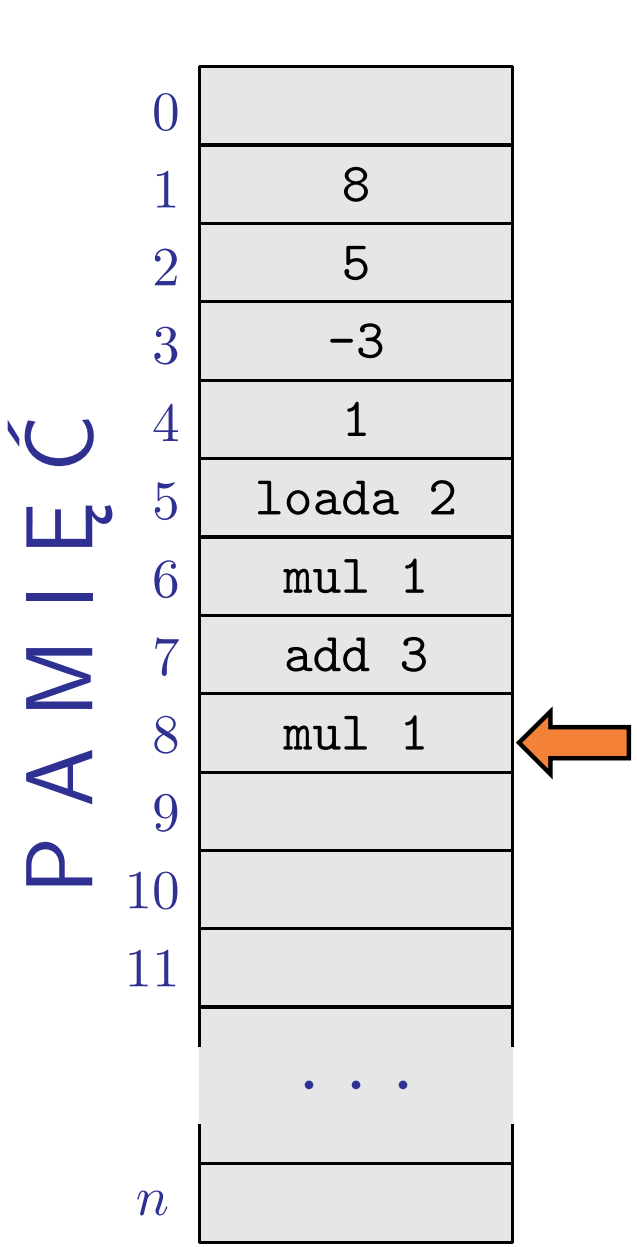
Obliczenie wartości wielomianu

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

(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)

37

PRZYKŁAD:

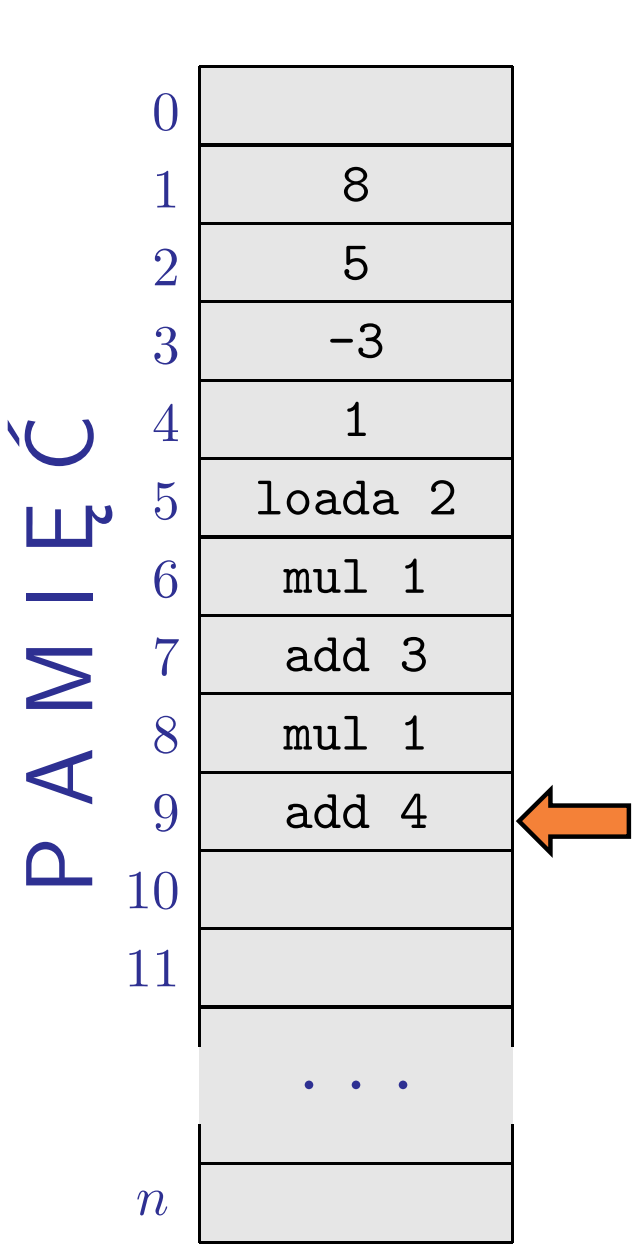
Obliczenie wartości wielomianu

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

(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)

296

PRZYKŁAD:

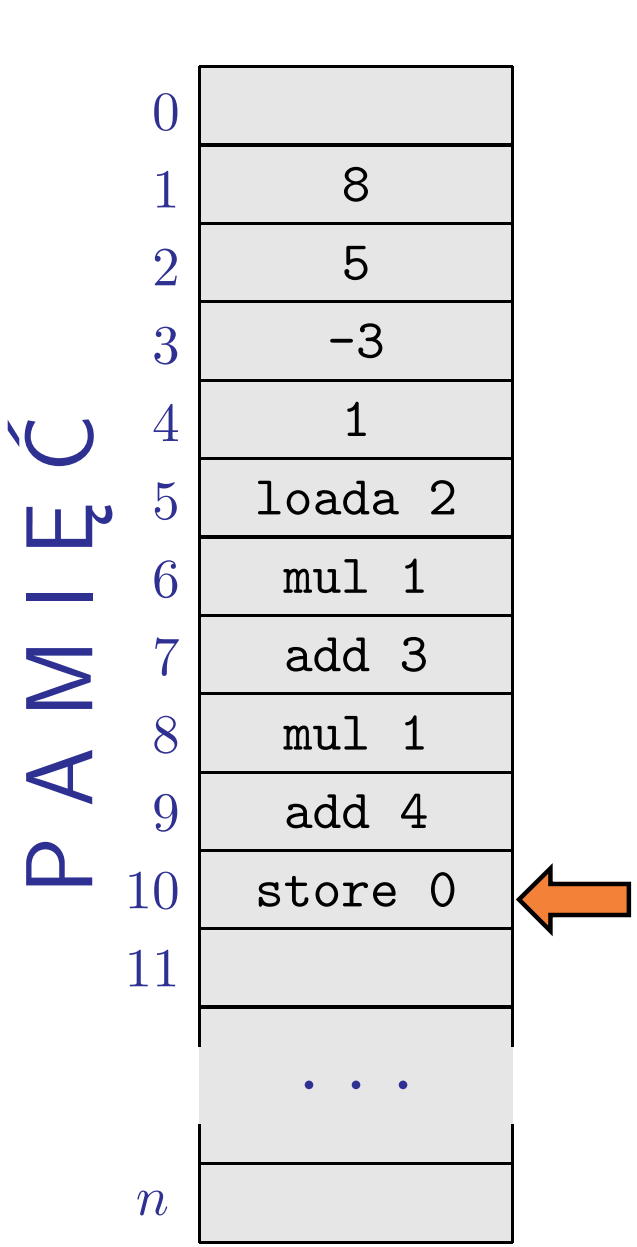
Obliczenie wartości wielomianu

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

(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:

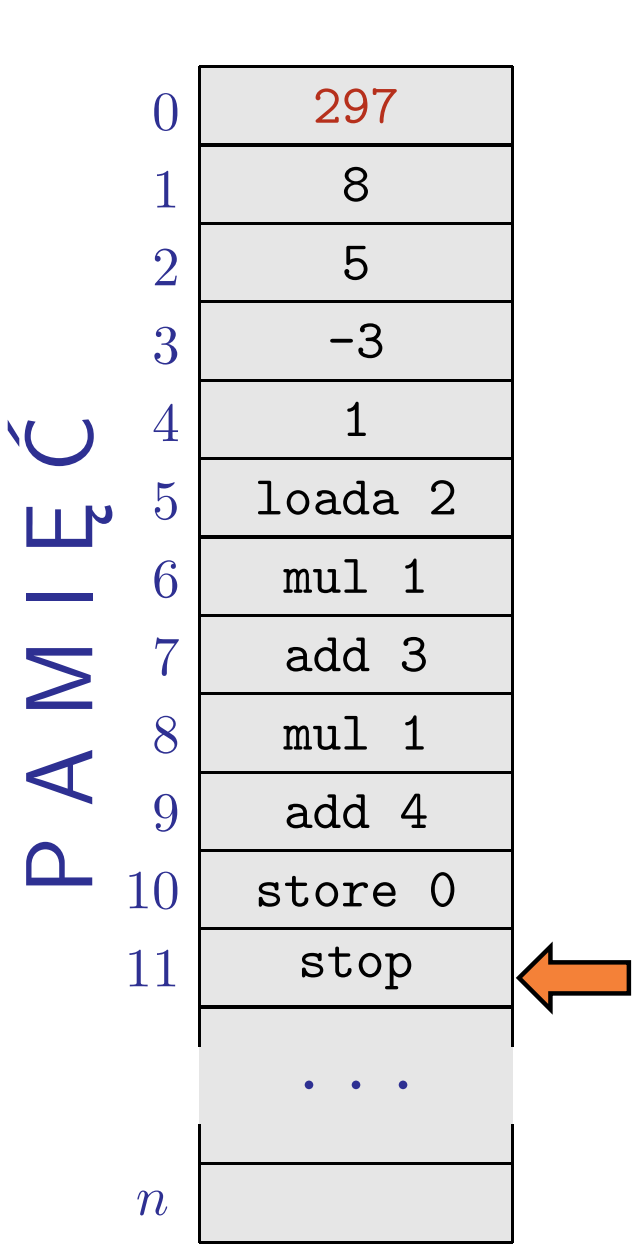
Obliczenie wartości wielomianu

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

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

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

297

PRZYKŁAD:

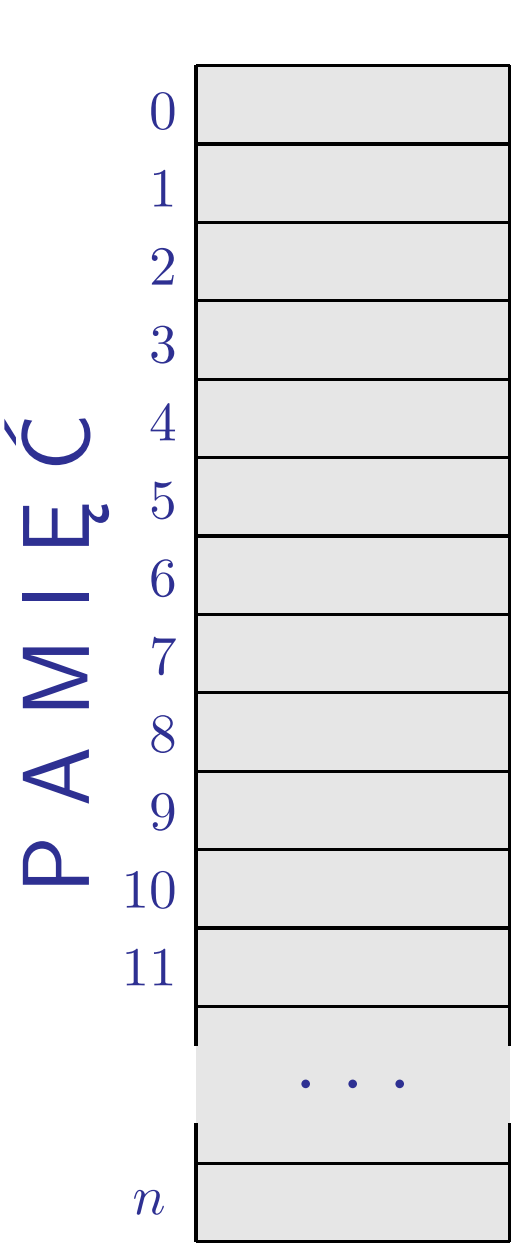
Obliczenie wartości wielomianu

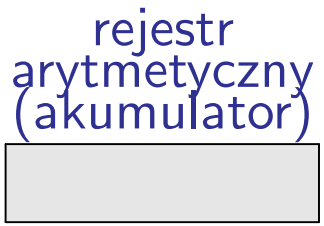
$$5x^2 - 3x + 1 = (5 \cdot x - 3) \cdot x + 1$$

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

Program należy uruchomić od komórki 5.

Programowanie niskiego poziomu



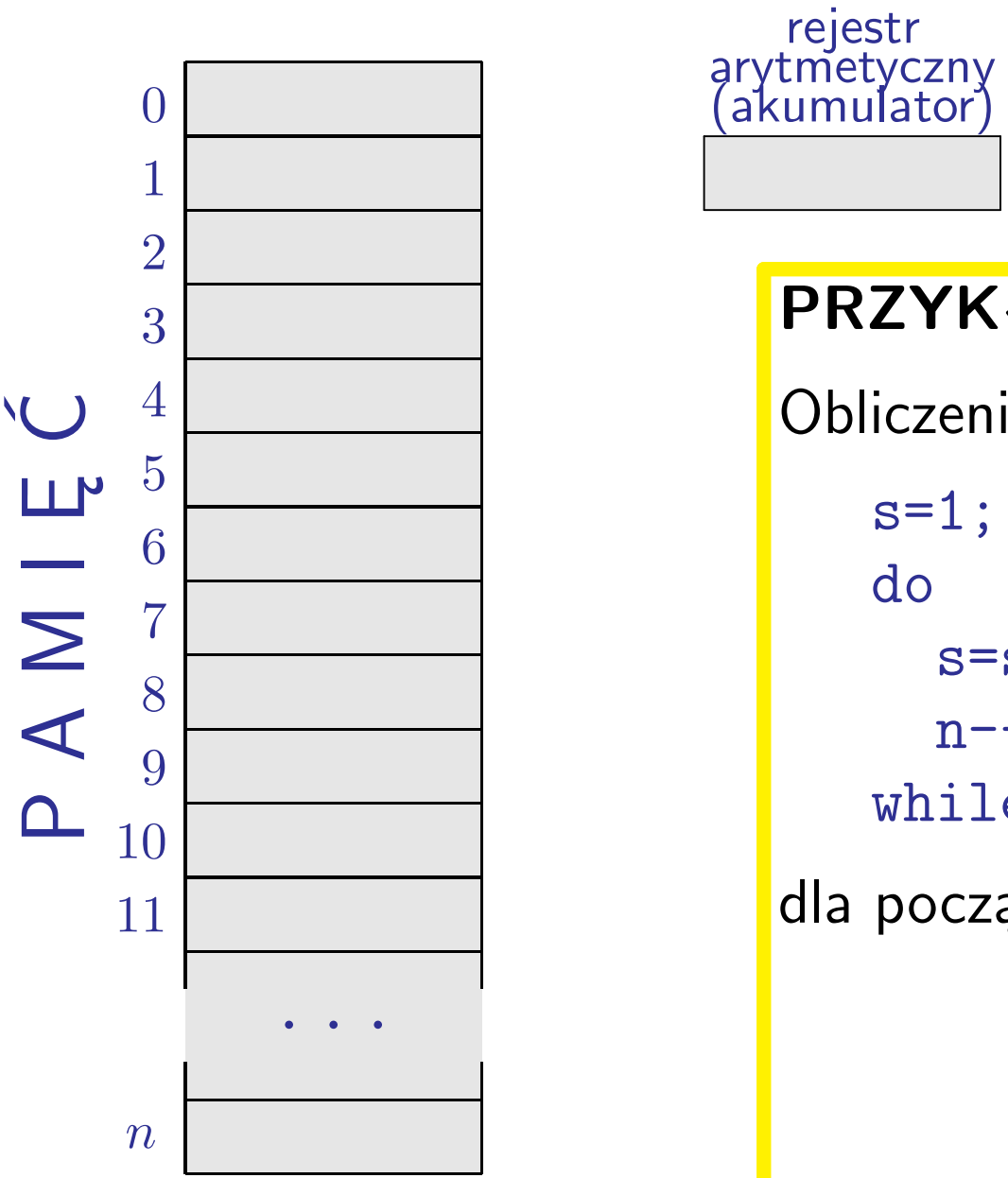


PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

Programowanie niskiego poziomu



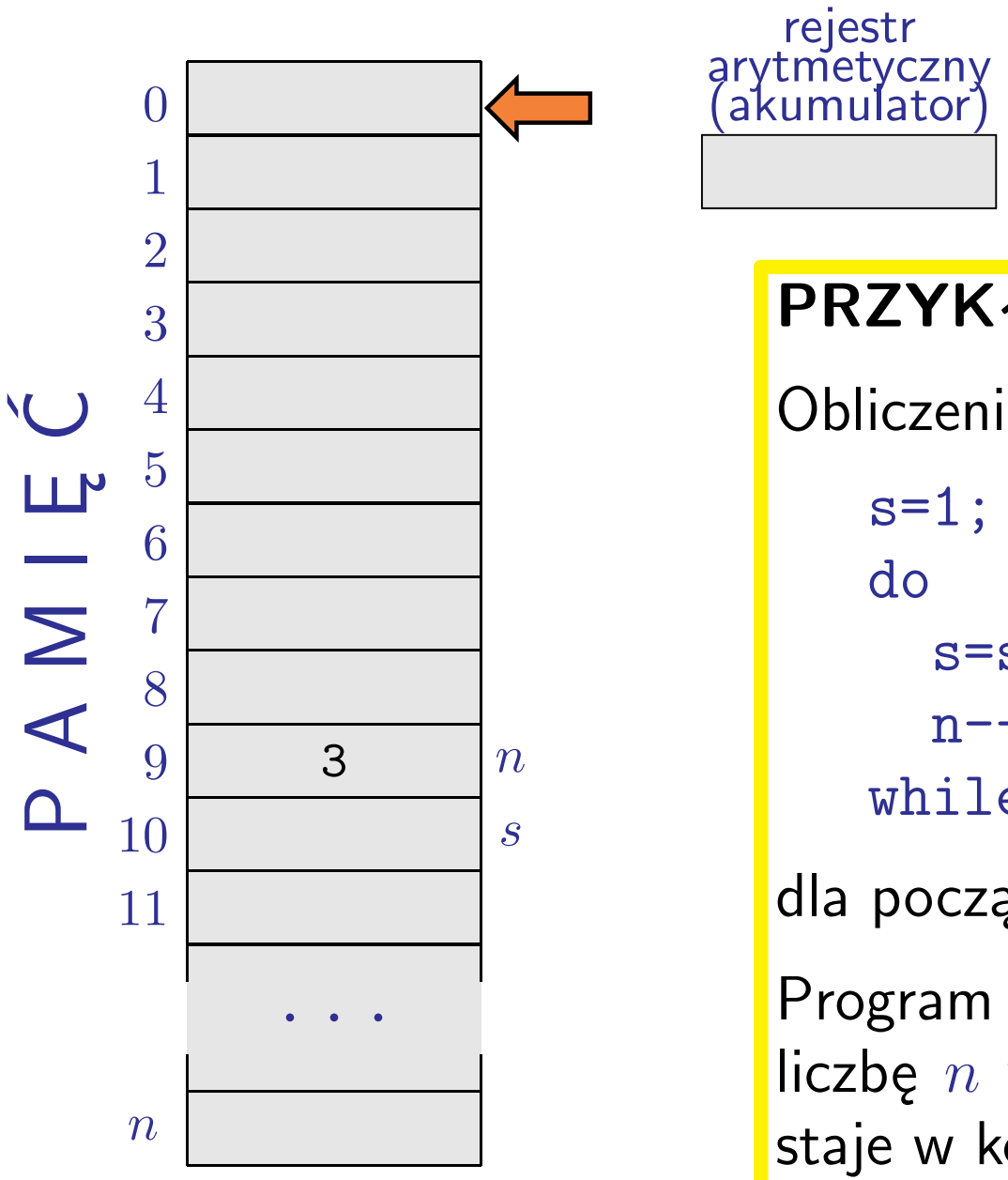
PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Programowanie niskiego poziomu



PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu

PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu

PRZYKŁAD:

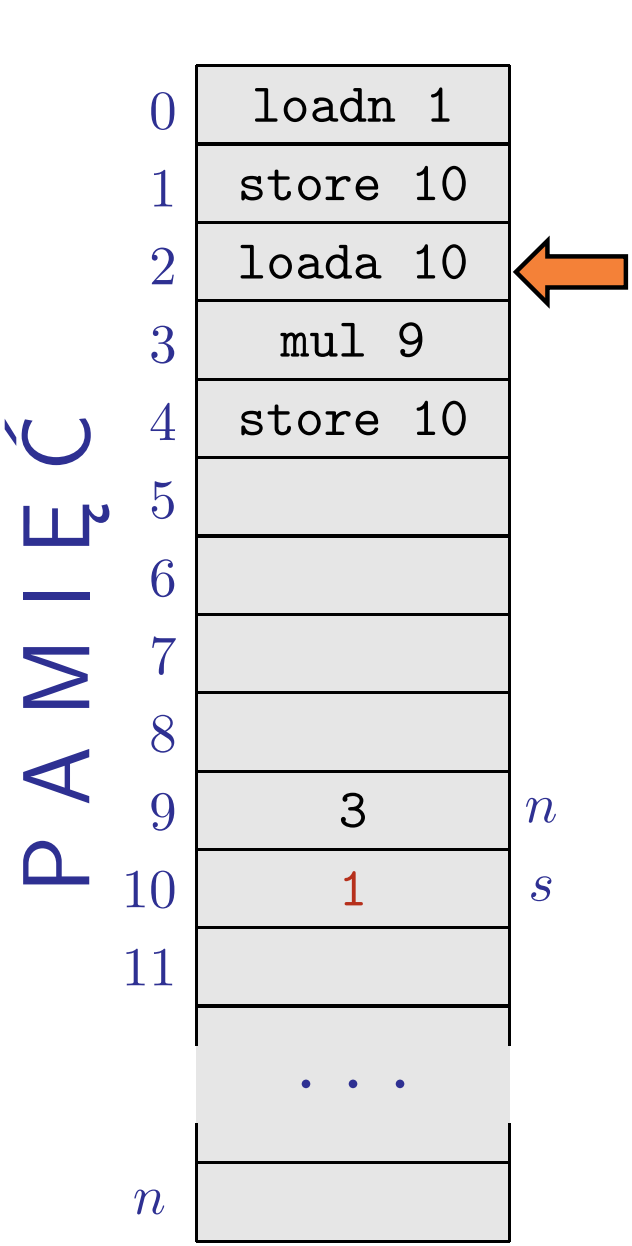
Obliczenie silni dla $n \geqslant 1$:

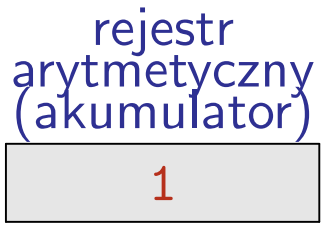
```
s=1;
do
   s=s*n;
   n--;
while (n>0);
```

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu





PRZYKŁAD:

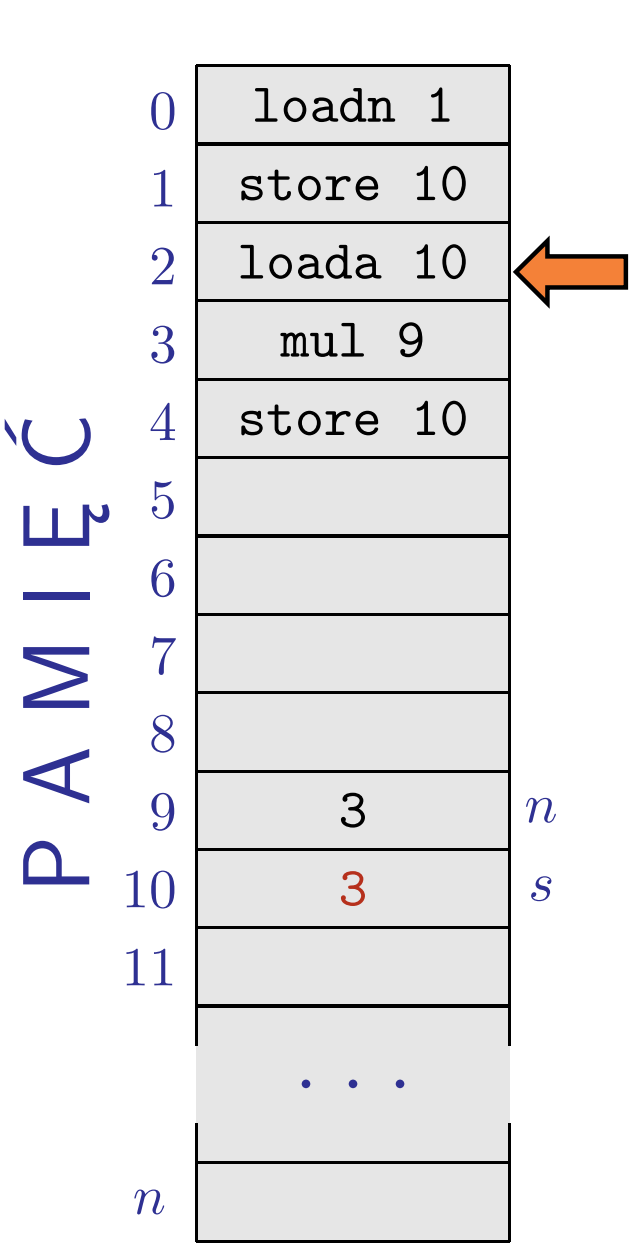
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

3

PRZYKŁAD:

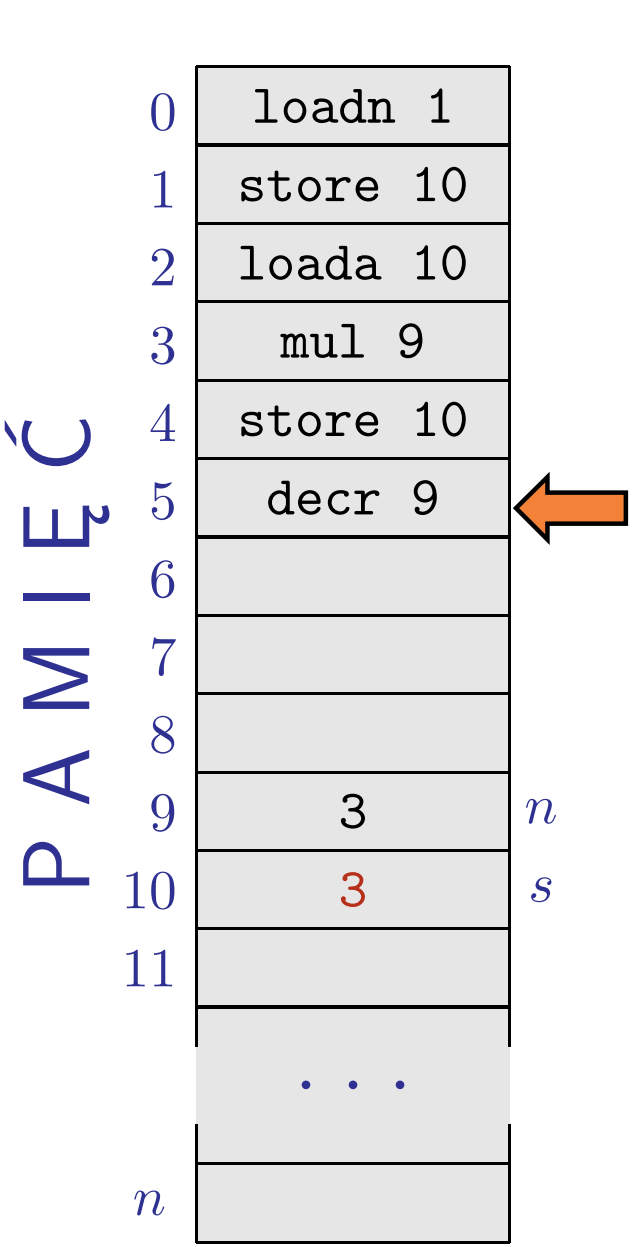
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

3

PRZYKŁAD:

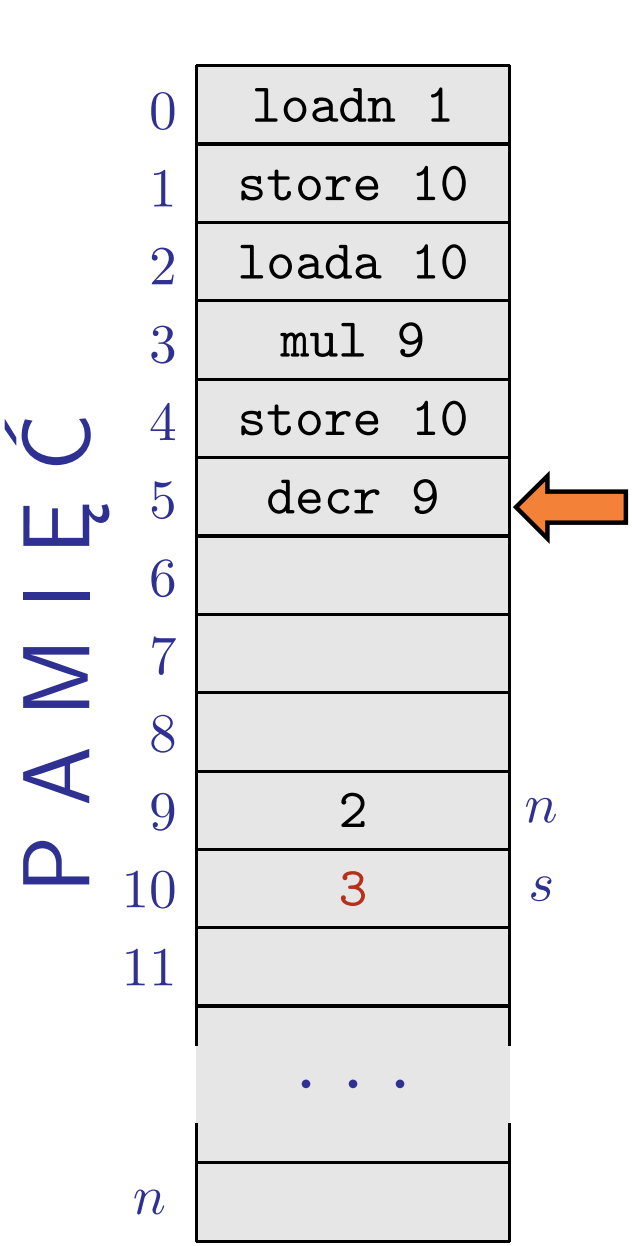
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

3

PRZYKŁAD:

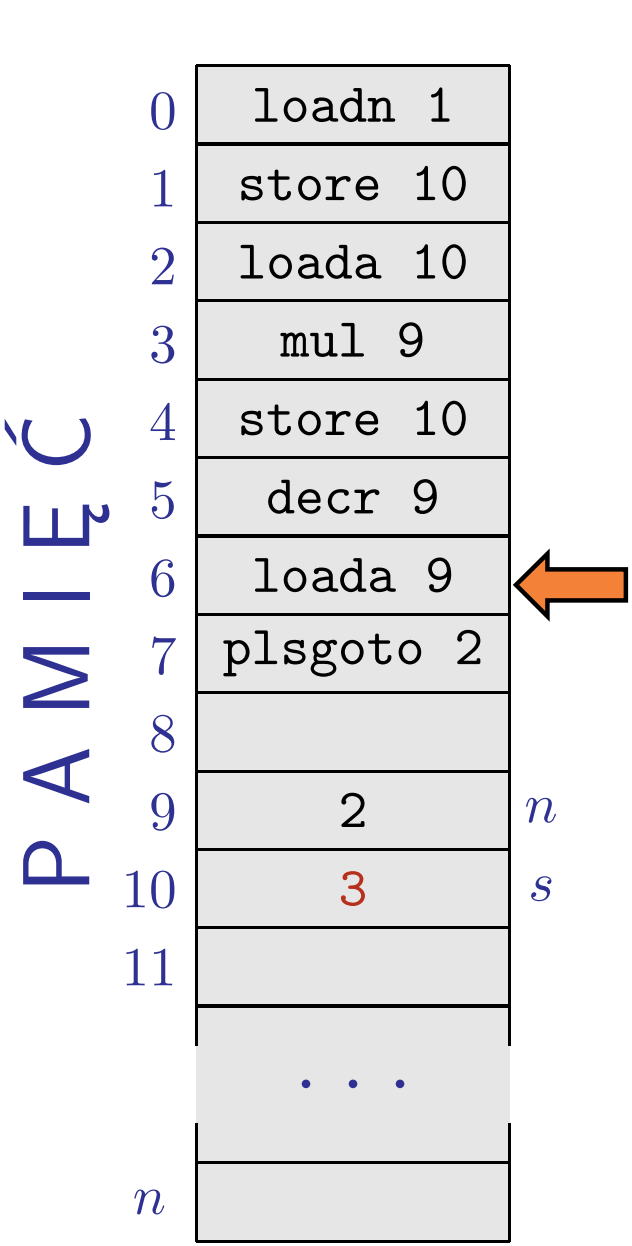
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

3

PRZYKŁAD:

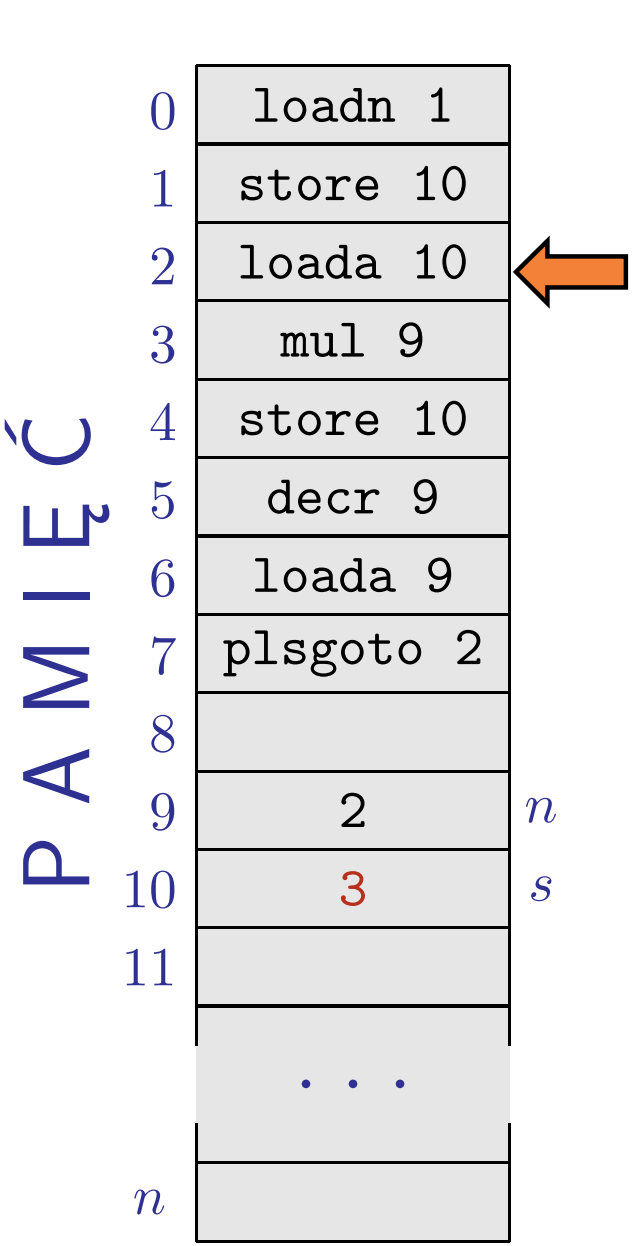
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

2

PRZYKŁAD:

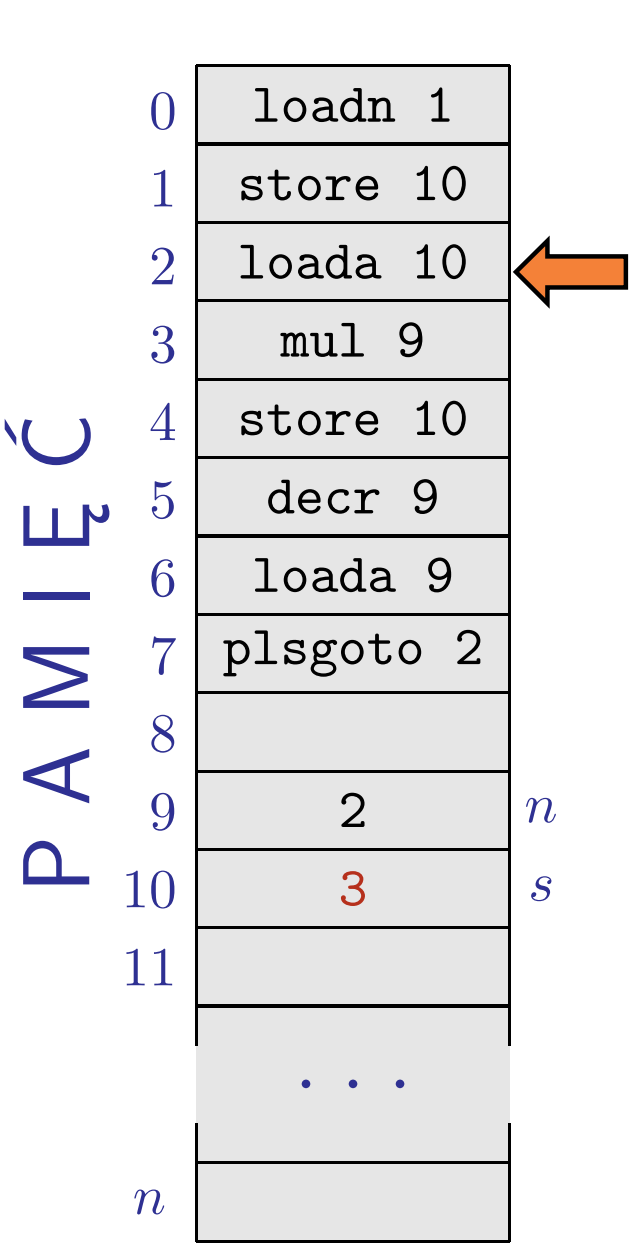
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

3

PRZYKŁAD:

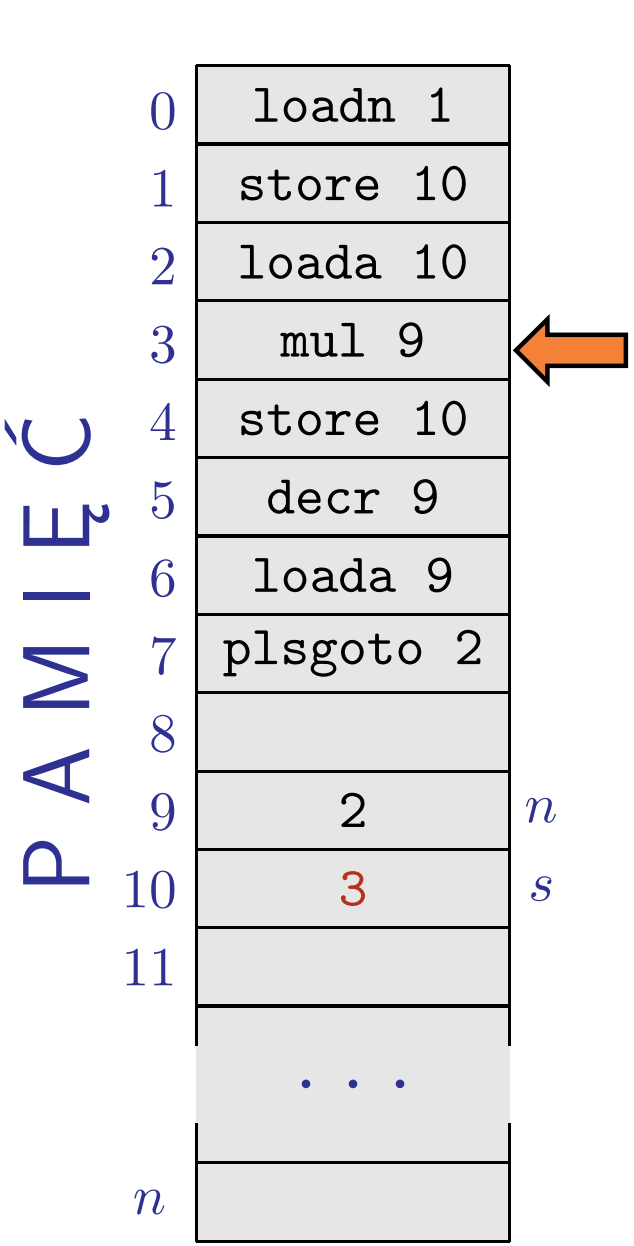
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

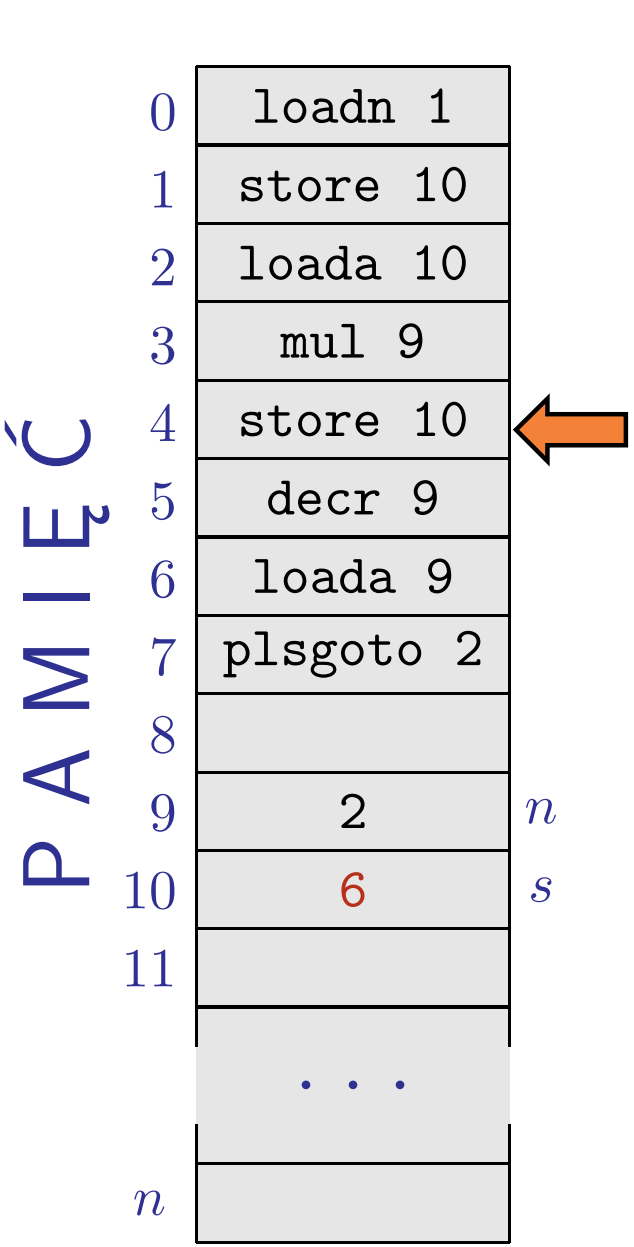
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

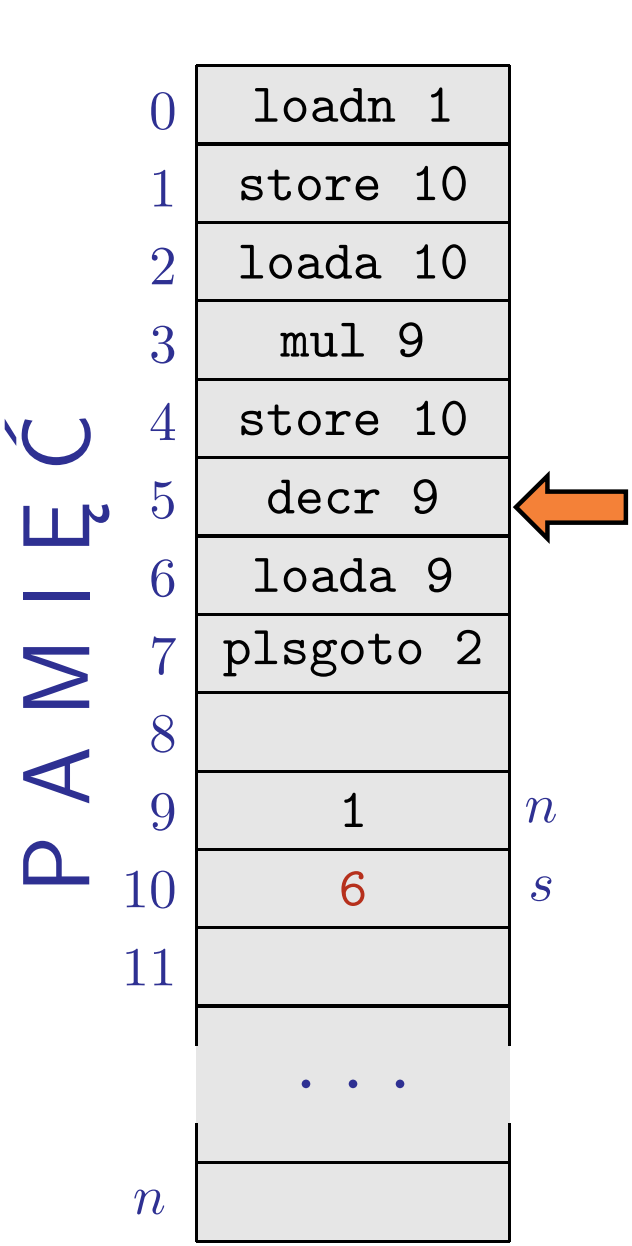
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

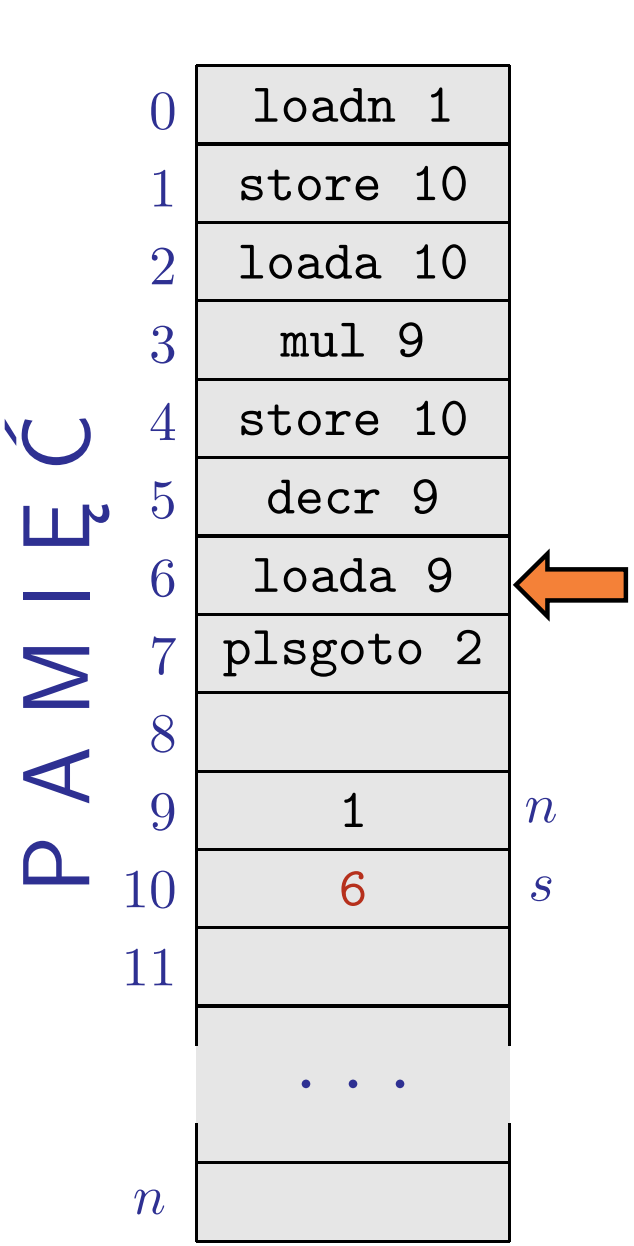
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

1

PRZYKŁAD:

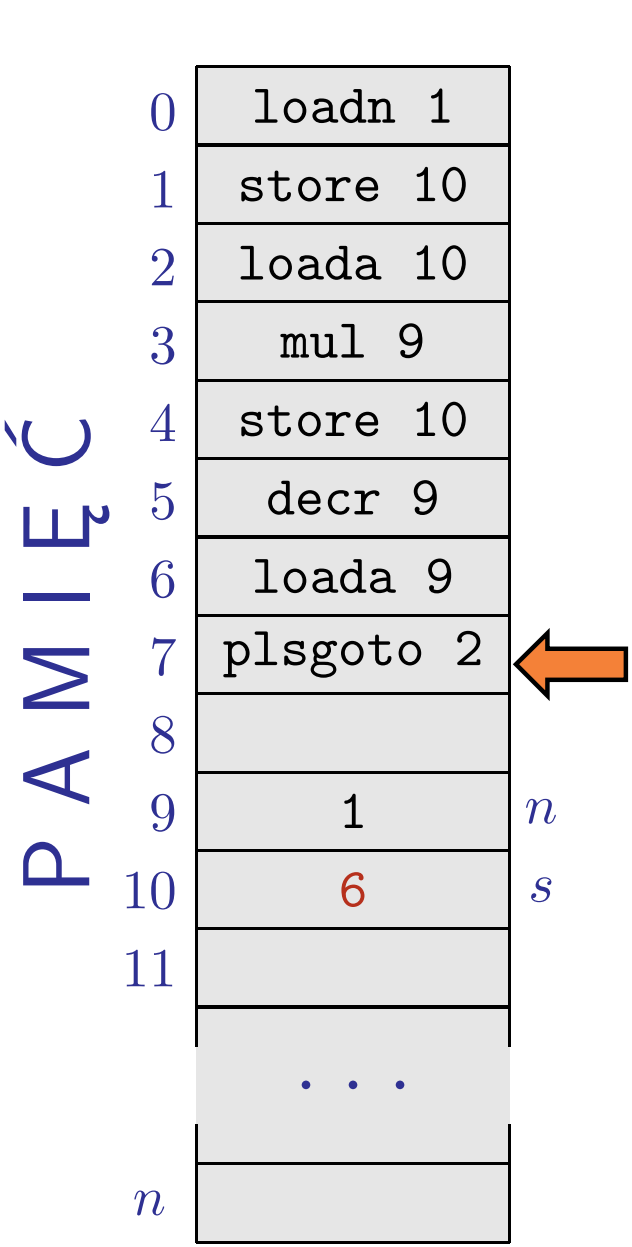
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

1

PRZYKŁAD:

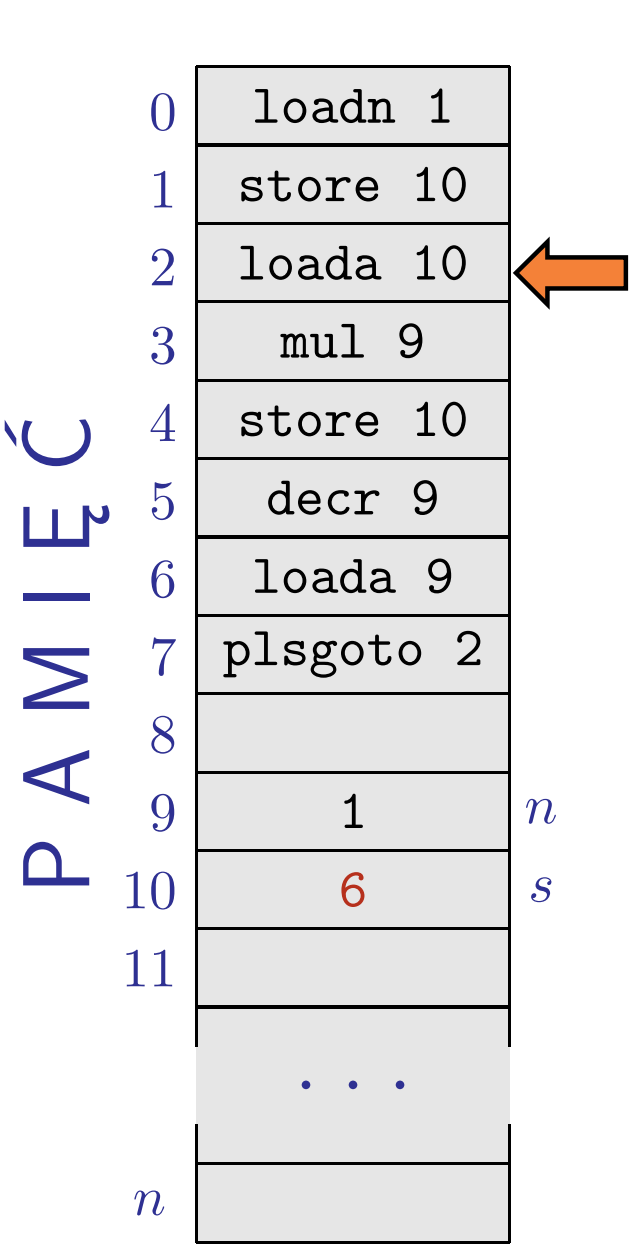
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

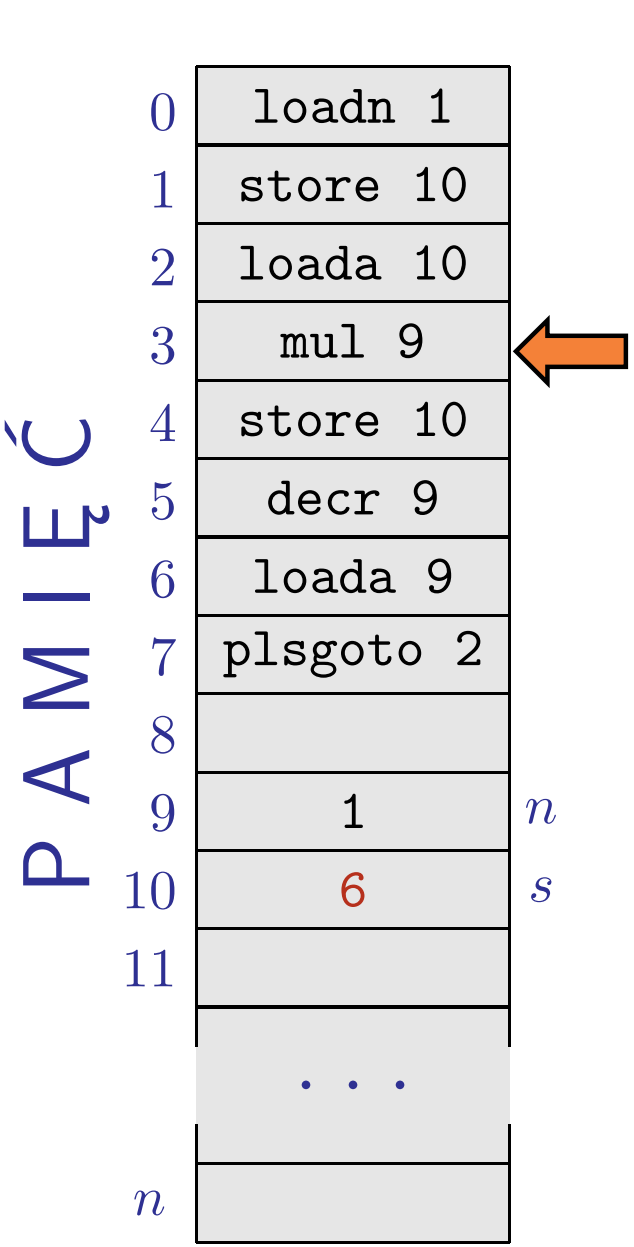
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

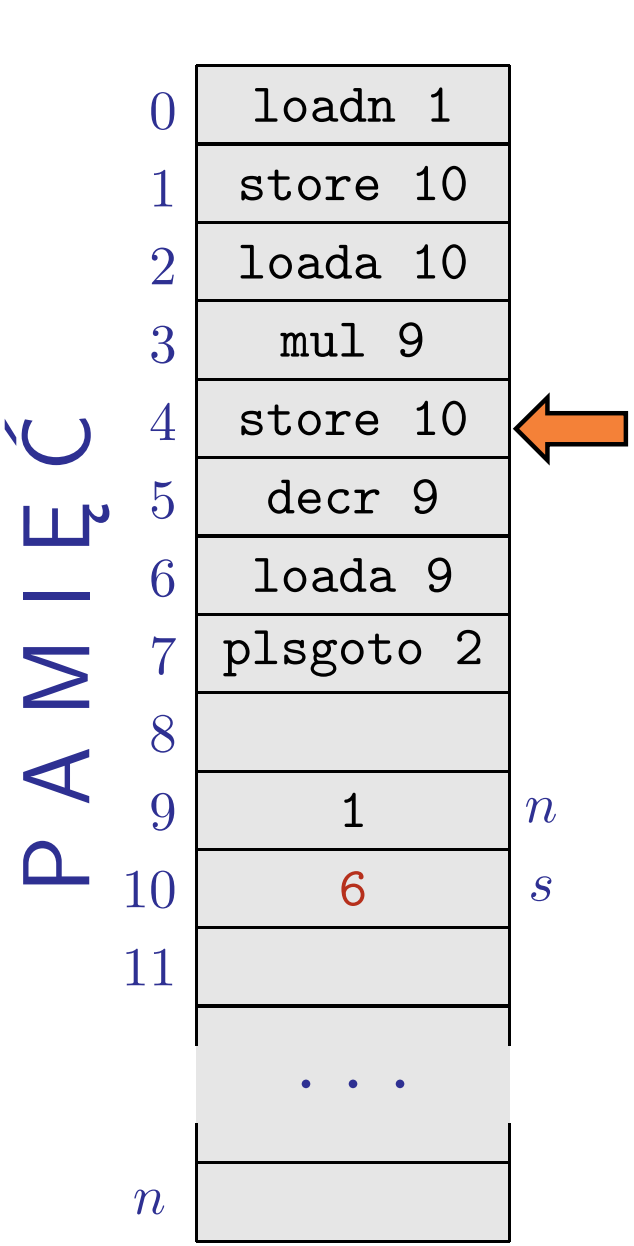
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

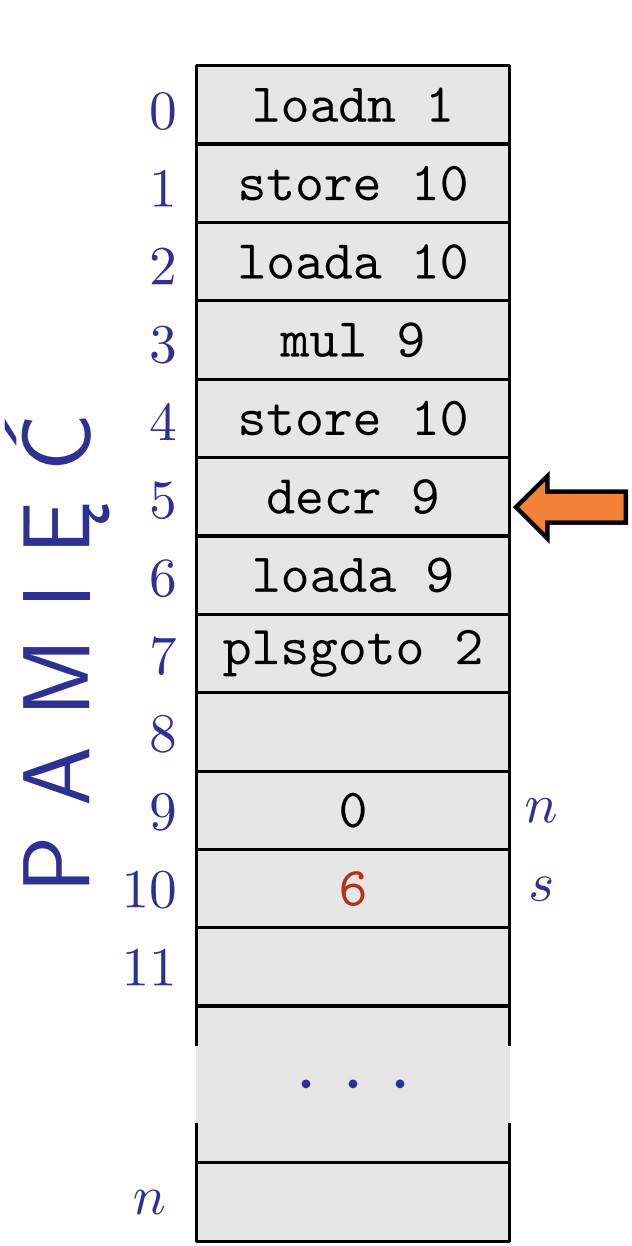
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

6

PRZYKŁAD:

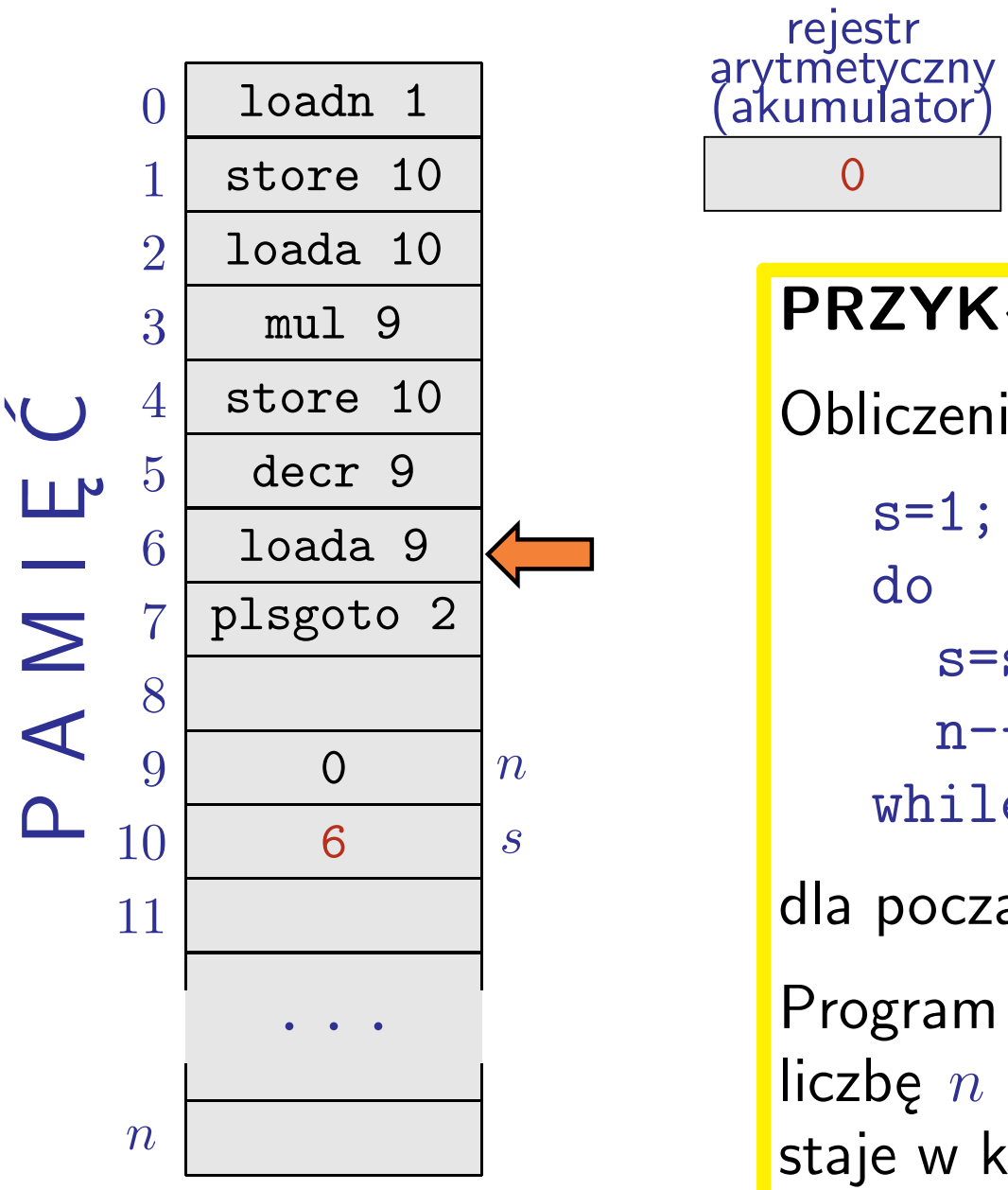
Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu



rejestr arytmetyczny (akumulator)

0

PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu

PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu

PRZYKŁAD:

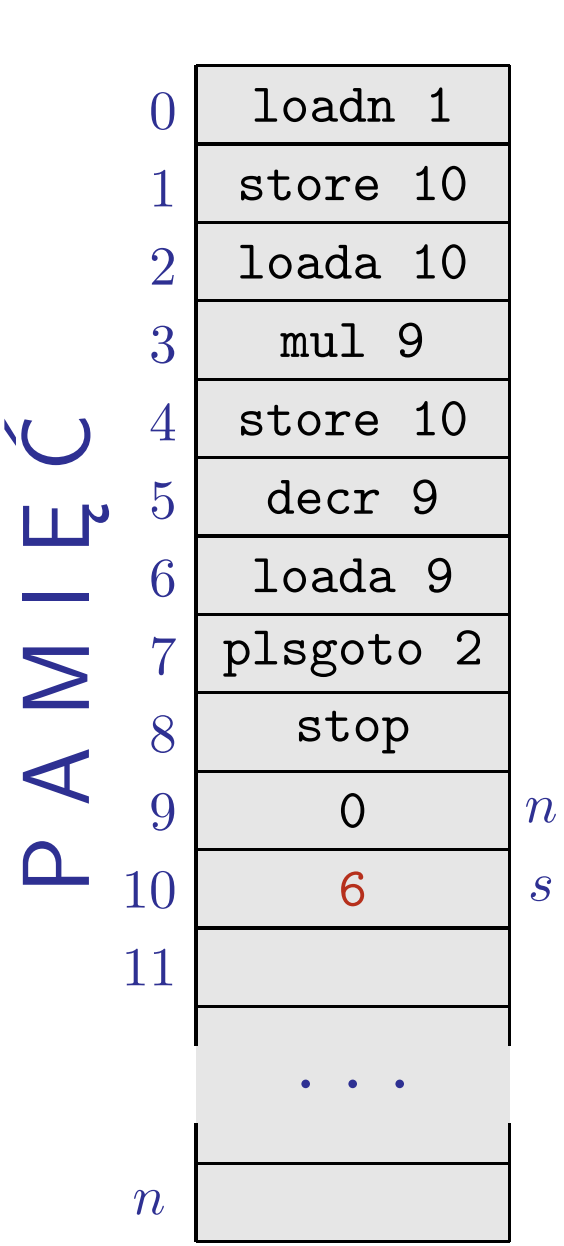
Obliczenie silni dla $n \geqslant 1$:

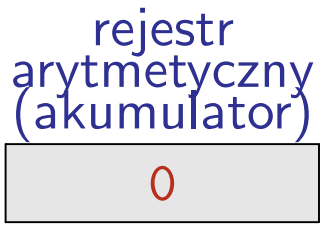
```
s=1;
do
   s=s*n;
   n--;
while (n>0);
```

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Programowanie niskiego poziomu





PRZYKŁAD:

Obliczenie silni dla $n \geqslant 1$:

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

dla początkowej wartości $n = 3$.

Program należy uruchomić od komórki 0; liczbę n wpisać do komórki 9, wynik zostaje w kom. 10.

Adresy

Prosty adres komendy

Adresy

Prosty adres komendy:

liczba całkowita oznaczająca

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

Adresy

Prosty 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

Prosty 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.

Modyfikowany adres komendy:

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

Adresy

Prosty 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.

Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k, należy tą 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

Prosty adres komendy:

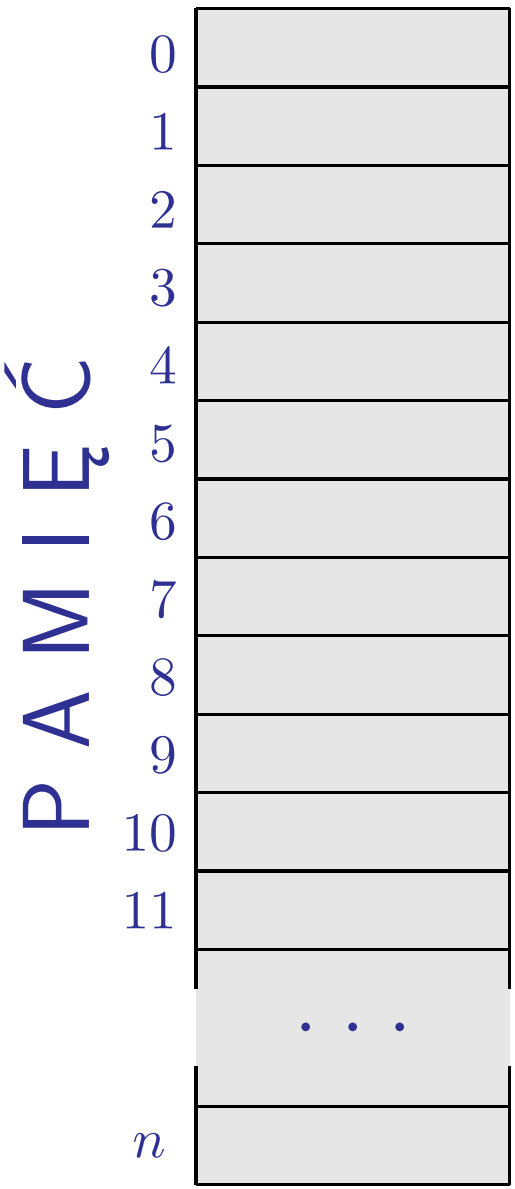
liczba całkowita oznaczająca

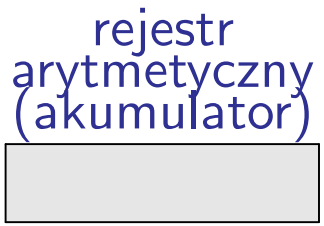
- albo samą liczbę,
  np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
  np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k, należy tą 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

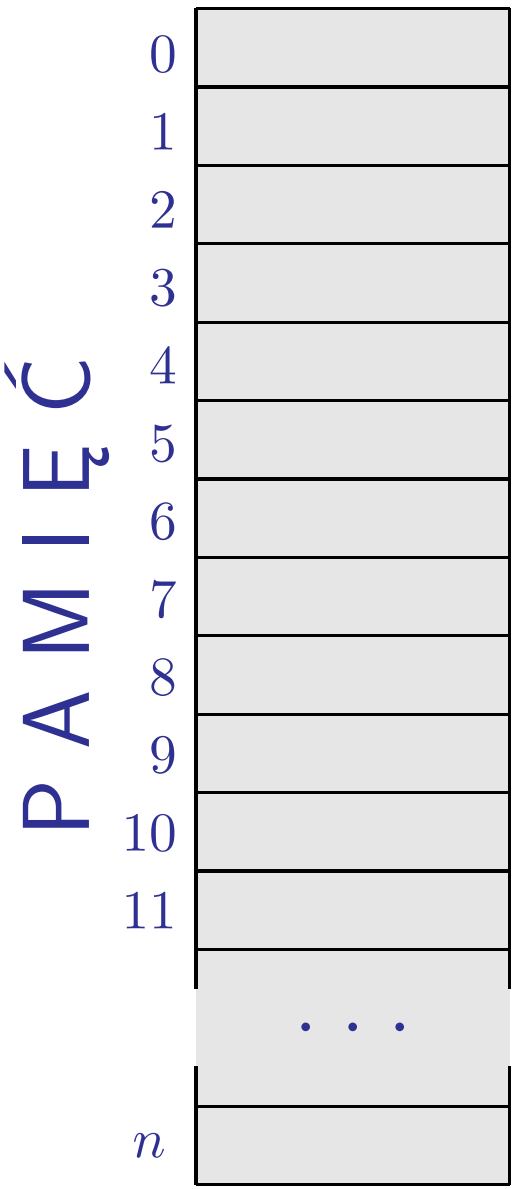


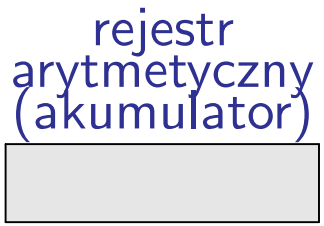


PRZYKŁAD:

Zerowanie n komórek

Adresy



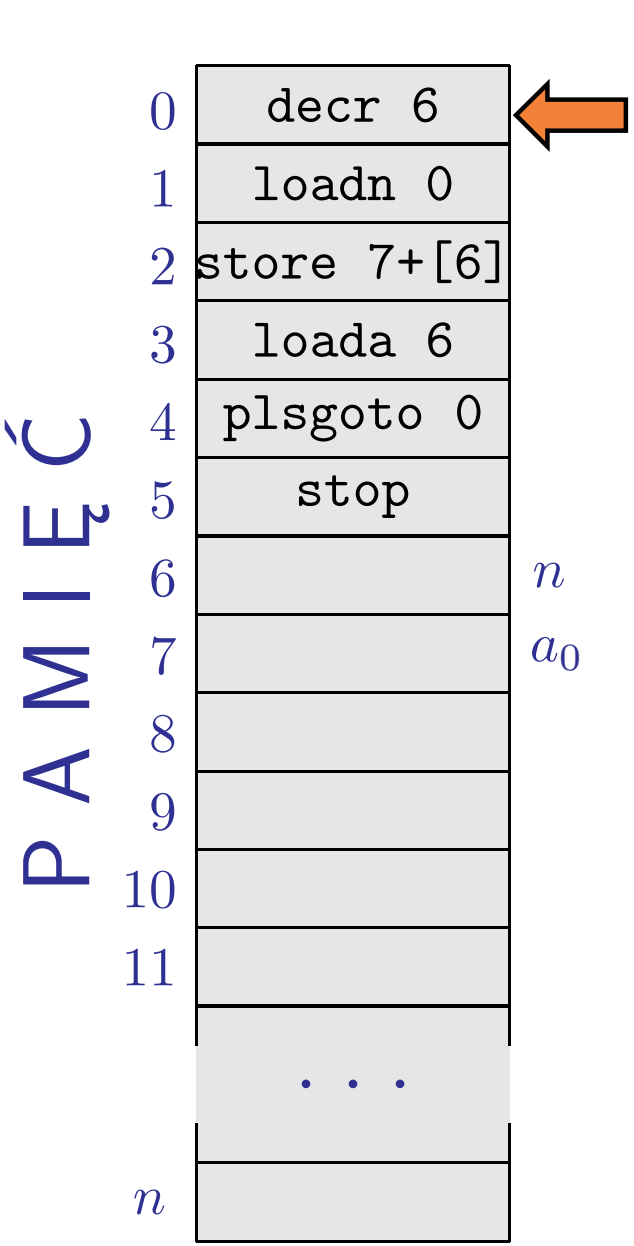


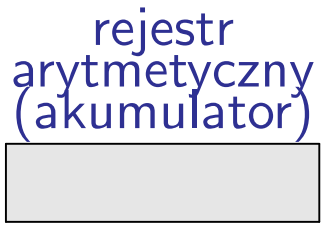
PRZYKŁAD:

Zerowanie n komórek:

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

Adresy





PRZYKŁAD:

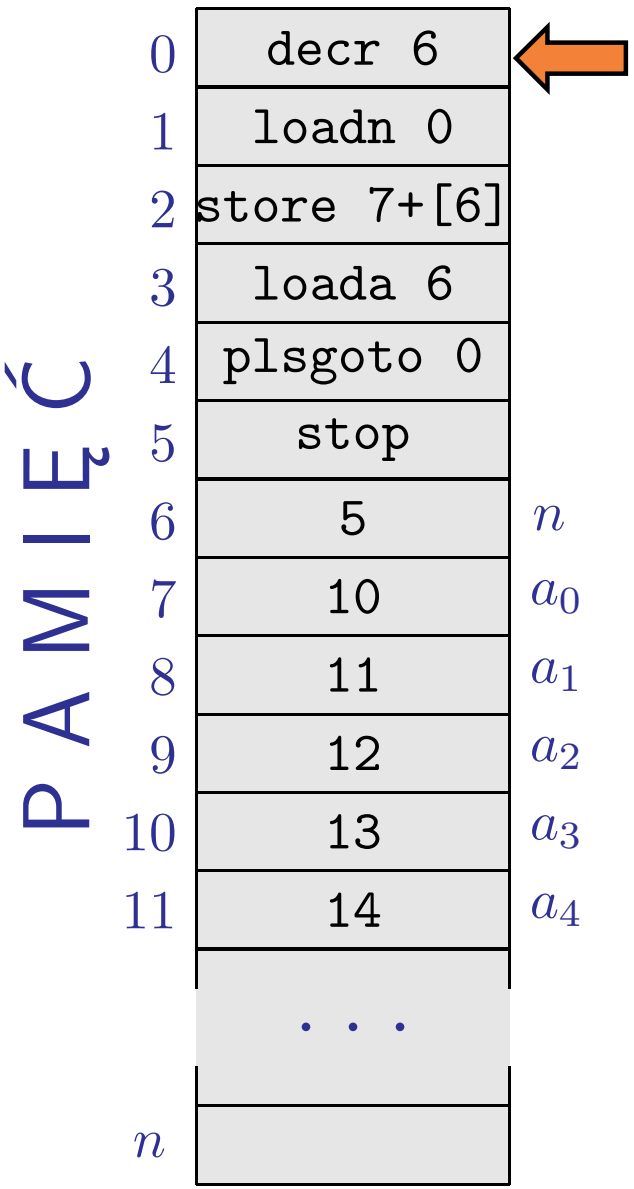
Zerowanie n komórek:

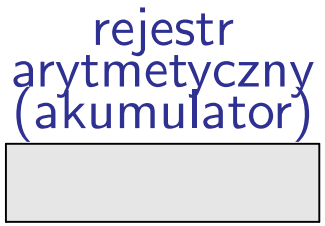
```
do
  n--;
  a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

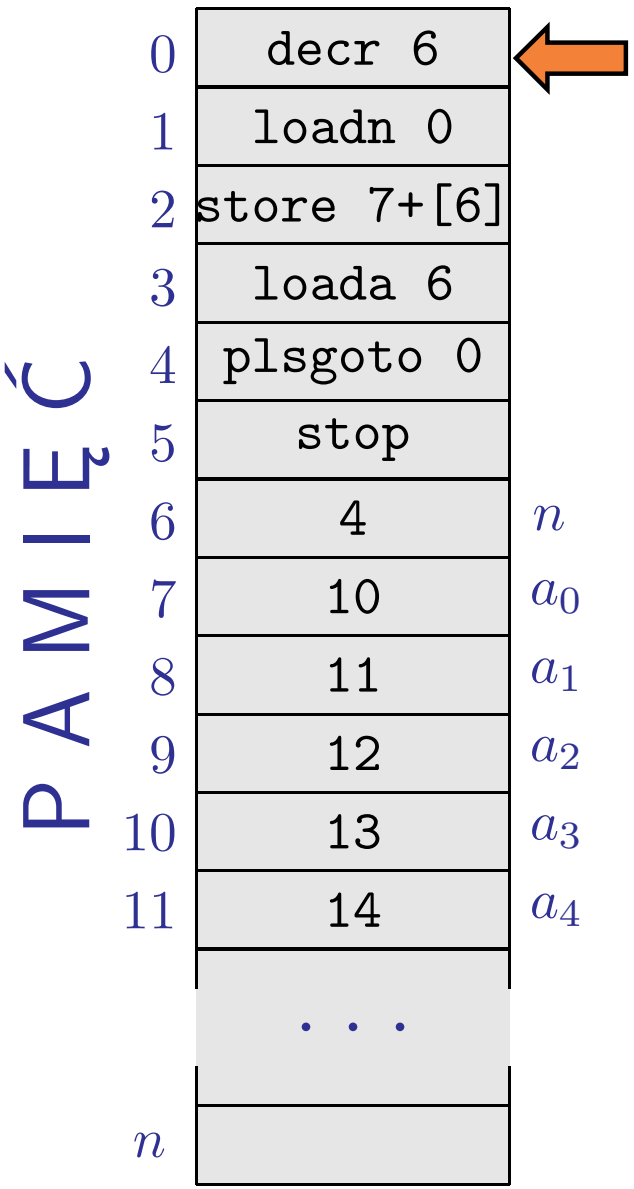
Zerowanie n komórek:

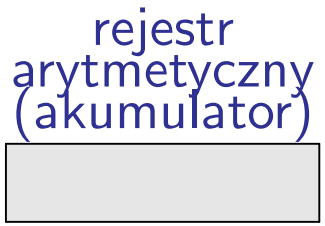
```
do
   n--;
   a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

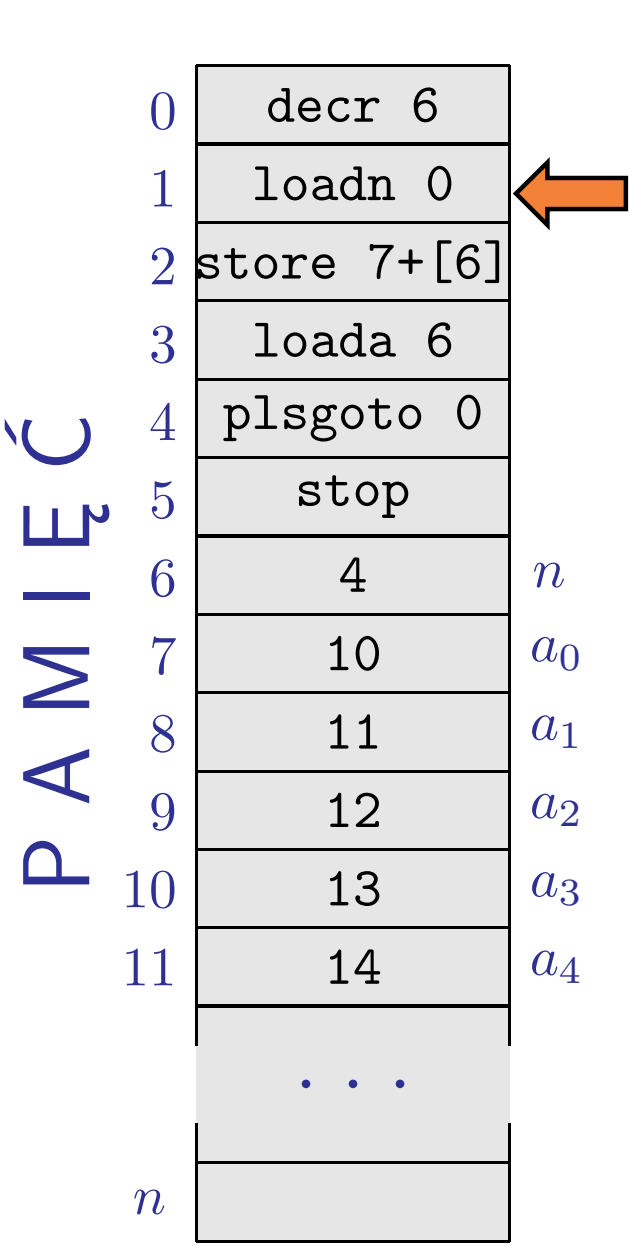
Zerowanie n komórek:

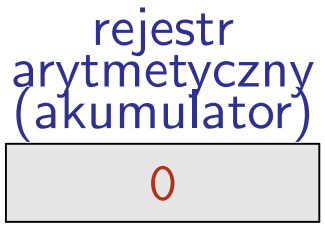
```
do
   n--;
   a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

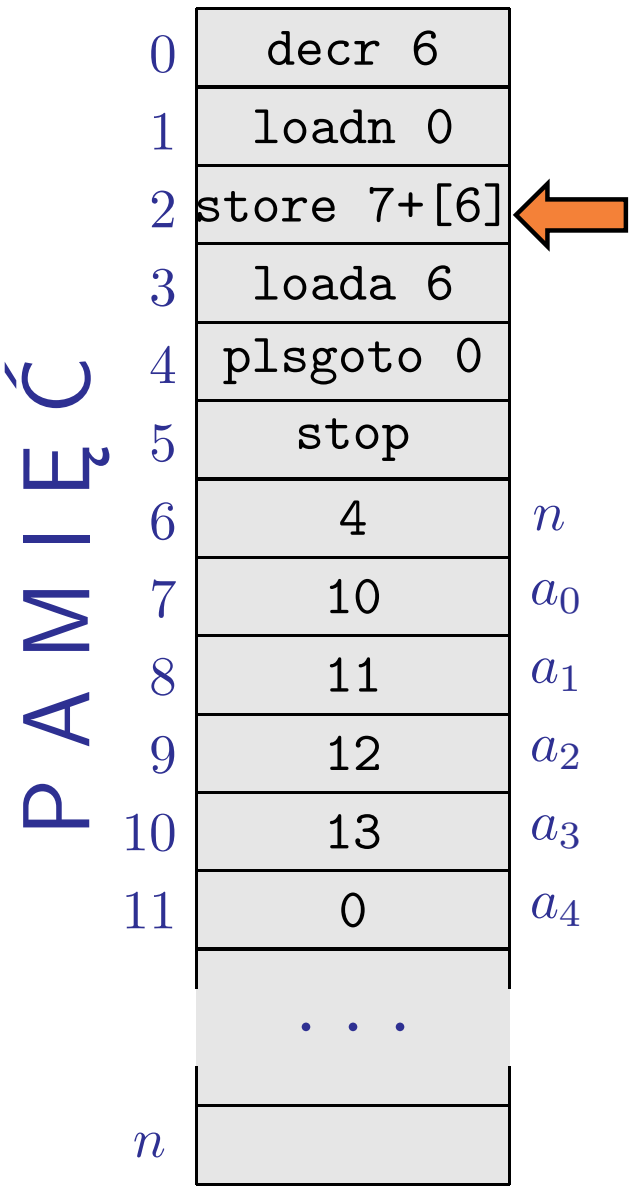
Zerowanie n komórek:

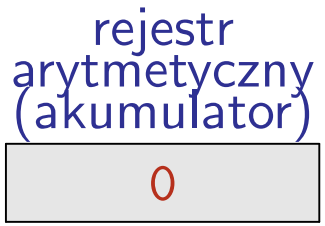
```
do
  n--;
  a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

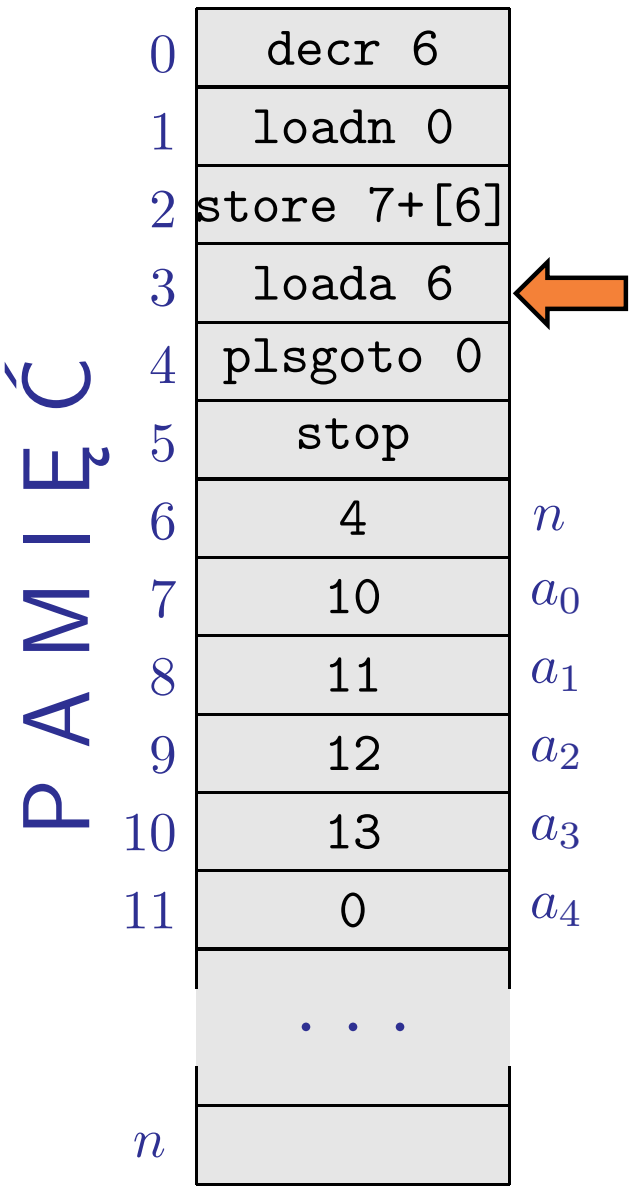
Zerowanie n komórek:

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

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy



rejestr arytmetyczny (akumulator)

4

PRZYKŁAD:

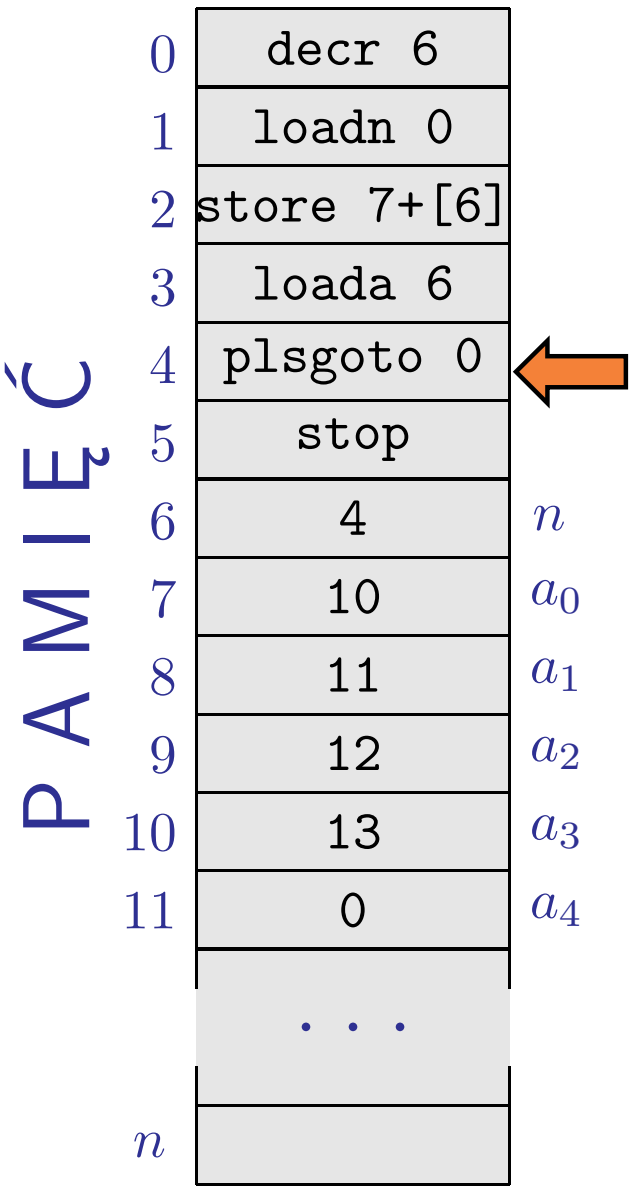
Zerowanie n komórek:

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

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy



rejestr arytmetyczny (akumulator)

4

PRZYKŁAD:

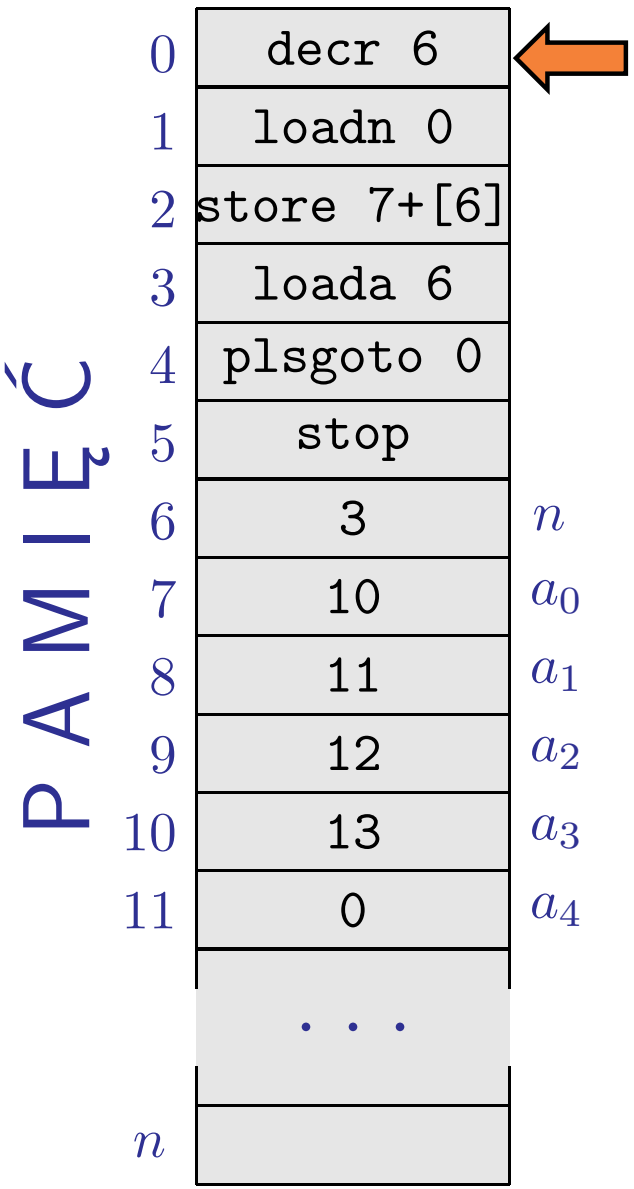
Zerowanie n komórek:

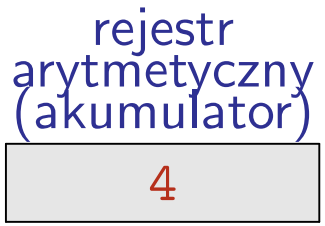
```
do
   n--;
   a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

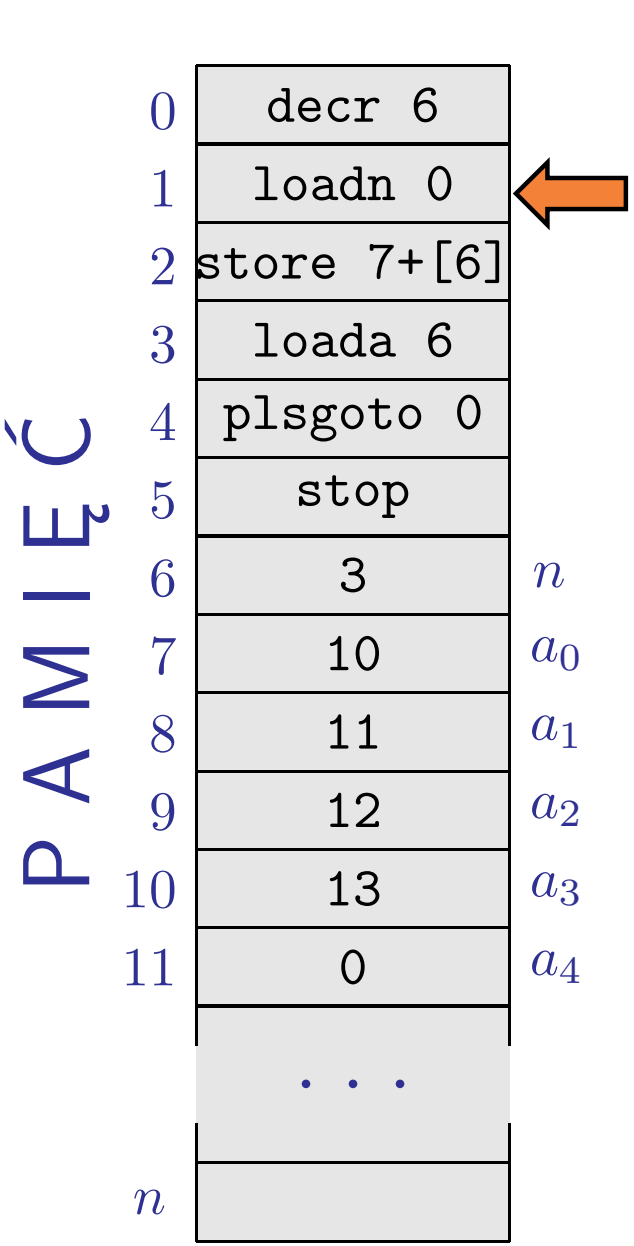
Zerowanie n komórek:

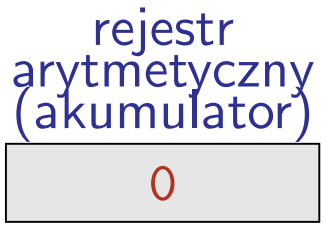
```
do
   n--;
   a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

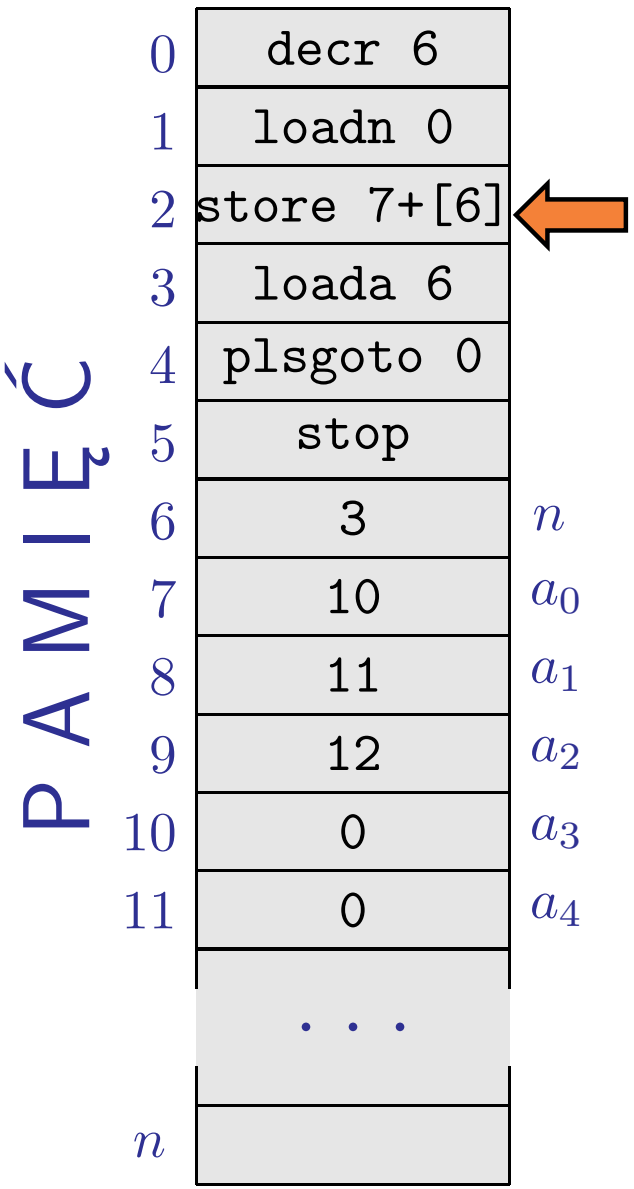
Zerowanie n komórek:

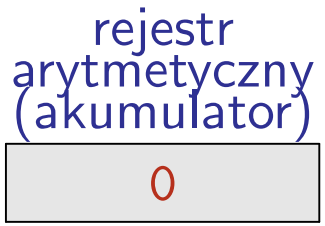
```
do
   n--;
   a[n]=0;
while (n>0);
```

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 0.

Adresy





PRZYKŁAD:

Zerowanie n komórek:

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

(wielkość n obszaru zerowanego zapisana jest w komórce 6; sam obszar zaczyna się w komórce 7).

Program należy uruchomić od komórki 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 drobniutkich kroczków — bardzo żmudne.
- Błędy trudne do znalezienia.

Jak programować w języku niskiego poziomu?

- Dużo drobniutkich 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 drobniutkich 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 drobniutkich 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 drobniutkich 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 drobniutkich 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)

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

```
          p = 1;

          i = 0;

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

          p = p*x;

          i++;
          goto pętla
koniec :
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

```
          p = 1;
          i = 0;

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

          p = p*x;

          i++;
          goto pętla
koniec :
```

```
loadn 1
store p
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

```
          p = 1;                          loadn 1
                                          store p
-------------------------------------------------------
          i = 0;                          loadn 0
                                          store i
-------------------------------------------------------

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



           p = p*x;

           i++;
           goto pętla
 koniec :
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
loadn 1
store p
loadn 0
store i
loada i          # pętla
sub n
plsgoto koniec
zergoto koniec
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
loadn 1
store p
loadn 0
store i
loada i          # pętla
sub n
plsgoto koniec
zergoto koniec
loada p
mul x
store p
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
loadn 1
store p
loadn 0
store i
loada i          # pętla
sub n
plsgoto koniec
zergoto koniec
loada p
mul x
store p
incr i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

```
          p = 1;                          loadn 1
                                          store p
-------------------------------------------------------
          i = 0;                          loadn 0
                                          store i
-------------------------------------------------------
                                          loada i          # pętla
  pętla :  if (i-n > 0) goto koniec;      sub n
           if (i-n == 0) goto koniec;     plsgoto koniec
                                          zergoto koniec
-------------------------------------------------------
                                          loada p
          p = p*x;                        mul x
                                          store p
-------------------------------------------------------
          i++;                            incr i
-------------------------------------------------------
          goto pętla                      goto pętla
-------------------------------------------------------
 koniec :
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

```
          p = 1;                          loadn 1
                                          store p
-------------------------------------------------------
          i = 0;                          loadn 0
                                          store i
-------------------------------------------------------
                                          loada i          # pętla
  pętla :  if (i-n > 0) goto koniec;      sub n
           if (i-n == 0) goto koniec;     plsgoto koniec
                                          zergoto koniec
-------------------------------------------------------
                                          loada p
          p = p*x;                        mul x
                                          store p
-------------------------------------------------------
          i++;                            incr i
-------------------------------------------------------
          goto pętla                      goto pętla
-------------------------------------------------------
 koniec :                                 stop             # koniec
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

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

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17         # pętla
 5: sub n
 6: plsgoto koniec
 7: zergoto koniec
 8: loada 16
 9: mul x
10: store 16
11: incr 17
12: goto pętla
13: stop             # koniec
14:                  # x
15:                  # n
16:                  # p
17:                  # i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17         # pętla
 5: sub 15
 6: plsgoto koniec
 7: zergoto koniec
 8: loada 16
 9: mul x
10: store 16
11: incr 17
12: goto pętla
13: stop             # koniec
14:                  # x
15:                  # n
16:                  # p
17:                  # i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17          # pętla
 5: sub 15
 6: plsgoto 13
 7: zergoto 13
 8: loada 16
 9: mul x
10: store 16
11: incr 17
12: goto pętla
13: stop              # koniec
14:                   # x
15:                   # n
16:                   # p
17:                   # i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17         # pętla
 5: sub 15
 6: plsgoto 13
 7: zergoto 13
 8: loada 16
 9: mul 14
10: store 16
11: incr 17
12: goto pętla
13: stop             # koniec
14:                  # x
15:                  # n
16:                  # p
17:                  # i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17          # pętla
 5: sub 15
 6: plsgoto 13
 7: zergoto 13
 8: loada 16
 9: mul 14
10: store 16
11: incr 17
12: goto 4
13: stop              # koniec
14:                   # x
15:                   # n
16:                   # p
17:                   # i
```

Jak programować w języku niskiego poziomu?

Przykład: (potęgowanie)

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

```
 0: loadn 1
 1: store 16
 2: loadn 0
 3: store 17
 4: loada 17          # pętla
 5: sub 15
 6: plsgoto 13
 7: zergoto 13
 8: loada 16
 9: mul 14
10: store 16
11: incr 17
12: goto 4
13: stop              # koniec
14: 3                 # x
15: 10                # n
16: 0                 # p
17: 0                 # i
END
```