ROZPROSZONE SYSTEMY KOMPUTEROWE

prof. dr hab. Ludwik Czaja

UV - 2021/2022 semestr zimowy

piątki, 10:10-12:05

Teams/Platon (on-line)

08.10.2021
15.10.2021
22.10.2021
29.10.2021
05.11.2021
12.11.2021
19.11.2021
26.11.2021
03.12.2021
10.12.2021
17.12.2021
14.01.2022

Problemy przedstawiane na kolejnych wykładach:

- Struktura i działanie komputera o wewnętrznym sterowaniu: główne części składowe i cykl wykonania rozkazu, przykładowa lista rozkazów, przykłady wykonania prostych programów w przykładowych rozkazach – symulacja animowana

- Pojęcie procesu. Mechanizmy synchronizacji i komunikacji międzyprocesowej. Scentralizowane systemy wieloprocesowe symulowane (wieloprogramowe, podział czasu, przerwania) i rzeczywiste (z wieloma fizycznymi procesorami). Przykłady działania takich systemów (m.in. synchronicznych: wektorowych, macierzowych) – symulacja animowana. Zjawiska patologiczne w systemach wieloprocesowych: zakleszczenie, zagłodzenie, niesprawiedliwy przydział zasobów

- Od systemów scentralizowanych symulujących pewne funkcje systemów rozproszonych do systemów fizycznie rozproszonych. Przegląd różnych rodzajów systemów relacji między nimi. Pewna klasyfikacja – tzw. taksonomia Flynn'a

- Systemy rozproszone: definicje (nieformalne), cele, cechy

- Współbieżne wykonywanie programów
  - transakcje współbieżne
  - zakleszczenia przy synchronizacji. Graf oczekiwania
  - zagłodzenia
  - wzajemne wykluczanie w systemach rozproszonych: z serwerem nadzorczym oraz algorytm pierścienia z żetonem

- Czas, koordynacja, wykluczanie bez nadzorcy, zagadnienia:
  - czas fizyczny i synchronizacja zegarów: metoda Cristiana, metoda Berkeley – „demon czasu”, metoda NTP (Network Time Protocol)
  - czas logiczny
  - wzajemne wykluczanie bez zewnętrznych usług dla procesów

- Komunikacja międzyprocesowa
  - podstawowe zagadnienia komunikacji
  - zadania protokołów komunikacyjnych – przykłady
  - wysyłanie i odbiór, rodzaje transmitowania komunikatów
  - warstwowe struktury protokołów

- Zdalne wywoływanie procedur (RPC)
  - motywacje, problemy, ograniczenia
  - przykład działania mechanizmu RPC

- Awarie w systemie rozproszonym
  - twierdzenia o szeregach liczbowych
  - reakcja na sytuacje awaryjne i ich szanse
  - problemy uzgodnień (problem „dwóch armii”, problem „bizantyjskich generałów”)
  - wybór nowego koordynatora po awarii: algorytm tyrana, pierścieniowy algorytm elekcji

- Rozproszona pamięć dzielona
  - motywacje, problemy, cechy korzystne i niekorzystne
  - przeplotowy model działania systemu
  - współbieżność operacji dostępu do pamięci DSM
  - zdarzenia inicjacji i zakończeń czytania i pisania
  - formalne definicje spójności ścisłej i sekwencyjnej, inne rodzaje spójności pamięci

Literatura

1. G. Coulouris, J. Dollimore, T. Kindberg: *Systemy rozproszone*, WNT 1999.
2. L. Czaja: *Zasady systemów rozproszonych z ilustracjami działania*, Vistula 2014
3. L. Czaja: *Introduction to Distributed Systems. Principles and Features*, Springer-Verlag, 2018
4. L. Czaja: *Cause-Effect Structures*, An Algebra of Nets with Examples of Applications, Springer-Verlag 2019
5. A.S. Tanenbaum: *Rozproszone systemy operacyjne, WNT 1997*
6. M. Ben-Ari: *Podstawy programowania współbieżnego i rozproszonego*,

Wykład 1

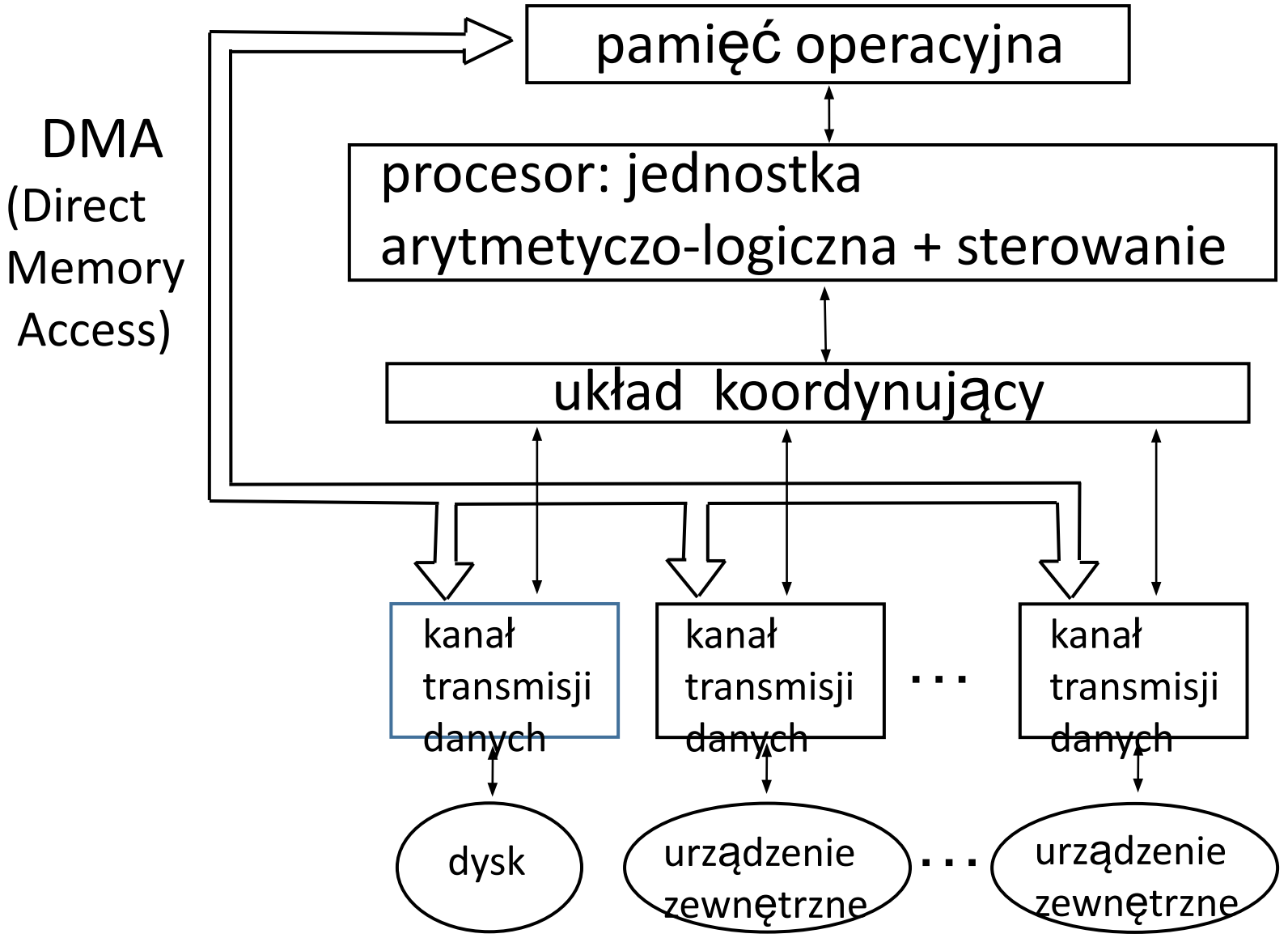
1. Zasada wewnętrznego (von Neumann) vs. zewnętrznego sterowania, uniwersalność – programowalność.

2. Schemat funkcjonalny komputera z wewnętrznym sterowaniem z przykładową listą rozkazów, przykład działania - symulacja (program w języku maszyny).

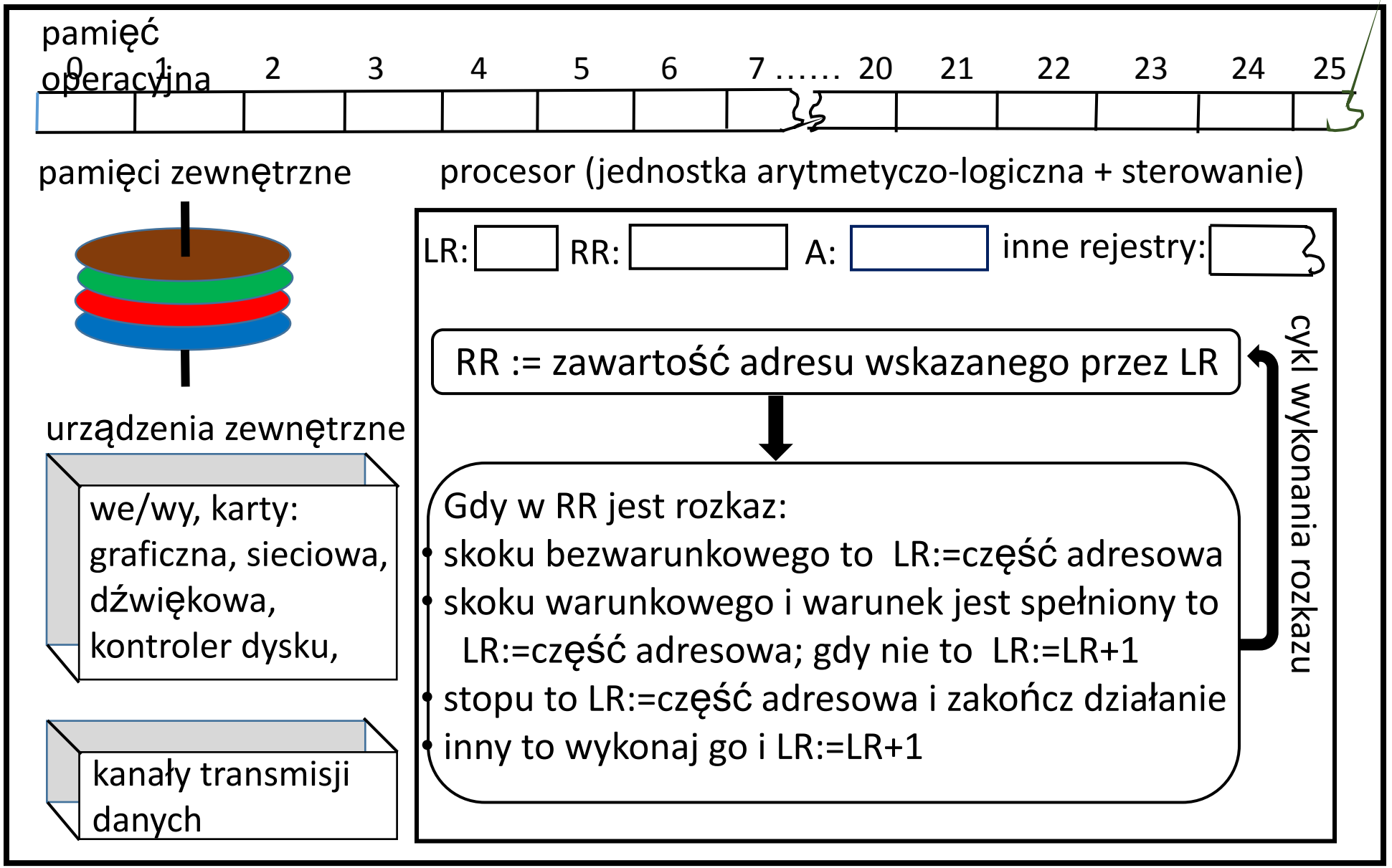
3. Wieloprogramowość z jednym procesorem.

4. Systemy wieloprocesorowe ze wspólną pamięcią. Przykład działania systemu 2-procesorowego (symulacja), zyski i zagrożenia.

Schemat połączeń układów komputera 1-procesorowego



Schemat funkcjonalny komputera z wewnętrznym sterowaniem
LR – licznik rozkazów, RR –rejestr rozkazów, A - akumulator



Przykładowa lista rozkazów (fragment)

ope-racja	adres	znaczenie
UA	*n*	**U**mieść w **A**kumulatorze zawartość adresu *n* i idź do następnego rozkazu w programie
PA	*n*	**P**amiętaj zawartość **A**kumulatora pod adresem *n* i idź do następnego rozkazu w programie
DO	*n*	**DO**daj do zawartości Akumulatora zawartość adresu *n*, wynik umieść w akumulatorze i idź do następnego rozkazu w programie
OD	*n*	**OD**ejmij od zawartości Akumulatora zawartość adresu *n*, wynik umieść w akumulatorze i idź do następnego rozkazu w programie
MN	*n*	**MN**óż zawartość Akumulatora przez zawartość adresu *n*, wynik umieść w akumulatorze i idź do następnego rozkazu w programie
DZ	*n*	**DZ**iel zawartość Akumulatora przez zawartość adresu *n*, wynik umieść w akumulatorze i idź do następnego rozkazu w programie; gdy zawartość adresu *n* jest zerem – sygnalizuj błąd
SK	*n*	idź do rozkazu pod adresem *n* w programie (**SK**ok bezwarunkowy)
SZ	*n*	gdy w akumulatorze jest 0 to idź do rozkazu pod adresem *n* w programie, a gdy nie to idź do następnego rozkazu („**S**kocz przy **Z**erze” - skok warunkowy)
SΩ	*n*	gdy w akumulatorze jest **Ω** to idź do rozkazu pod adresem *n* w programie, a gdy nie to idź do następnego rozkazu („**S**kocz przy **Ω**” - skok warunkowy)
ST	*n*	idź do rozkazu pod adresem *n* w programie i zatrzymaj działanie procesora

- algorytm: metoda postępowania, np. obliczeń
- program: algorytm zapisany w jakimś języku
- proces: (nieformalnie) wykonywanie się programu

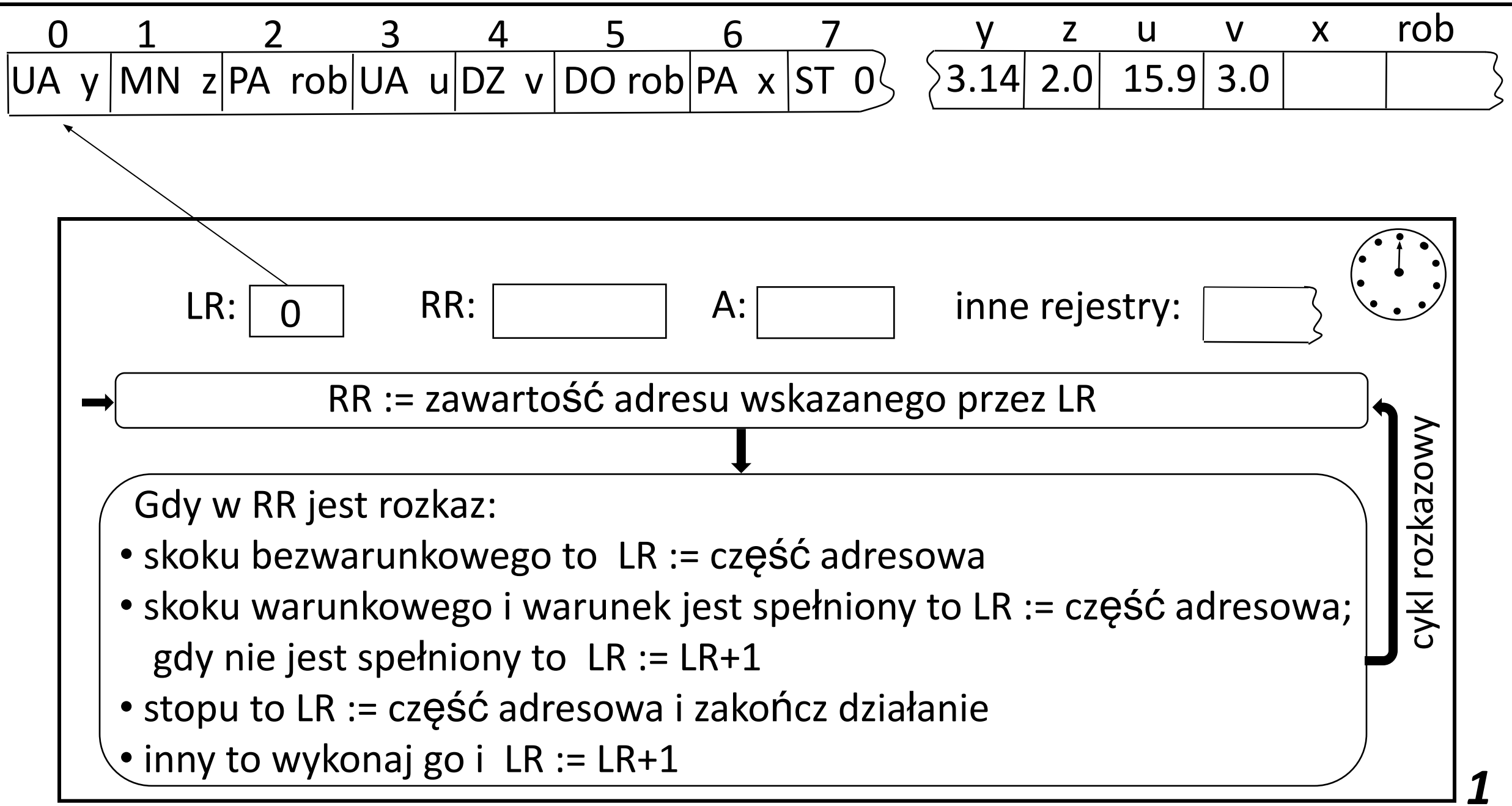
Program wykonania $x := y*z + u/v$ zapisany rozkazami z powyższej listy:

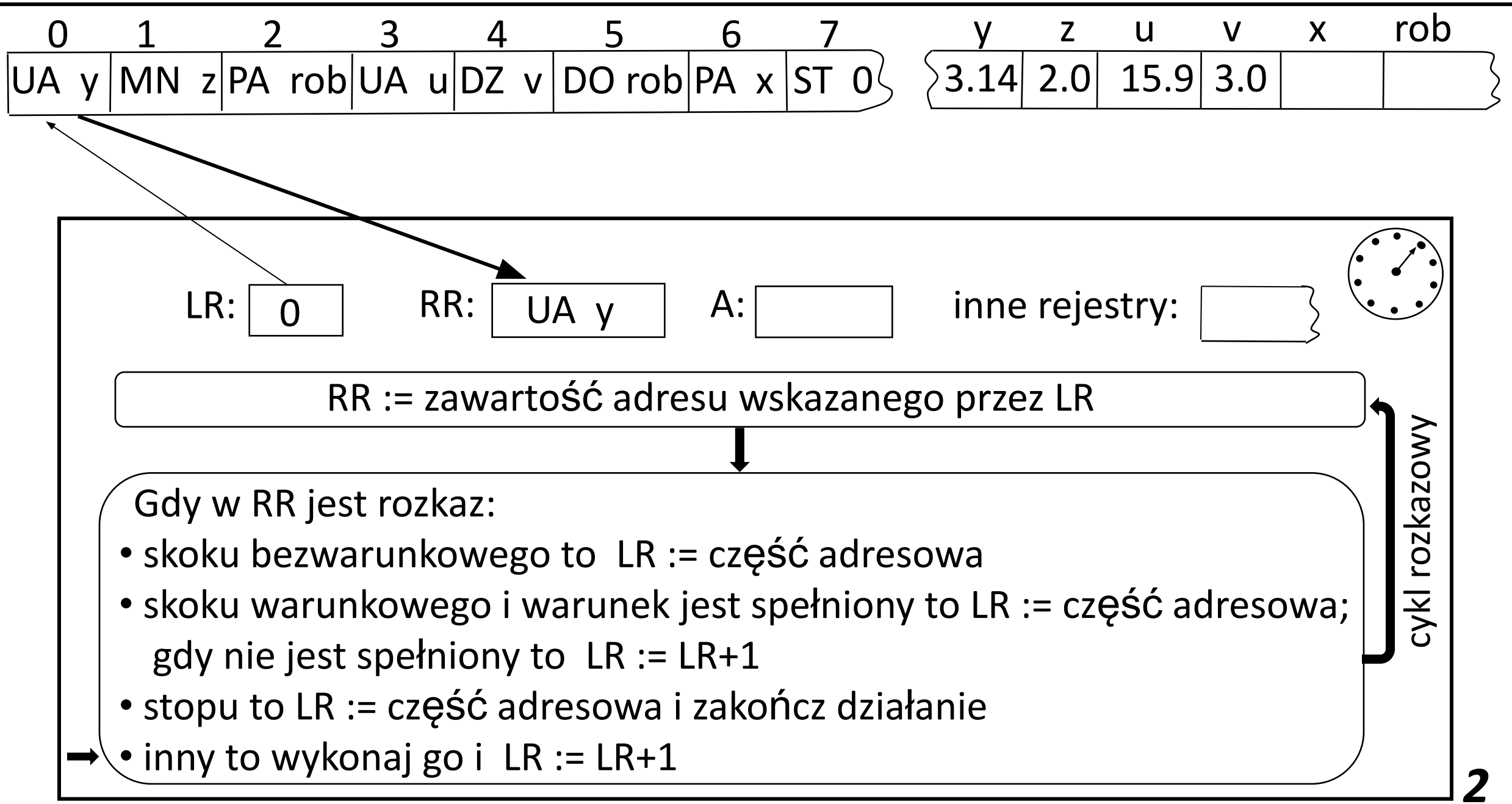
język typu assembler

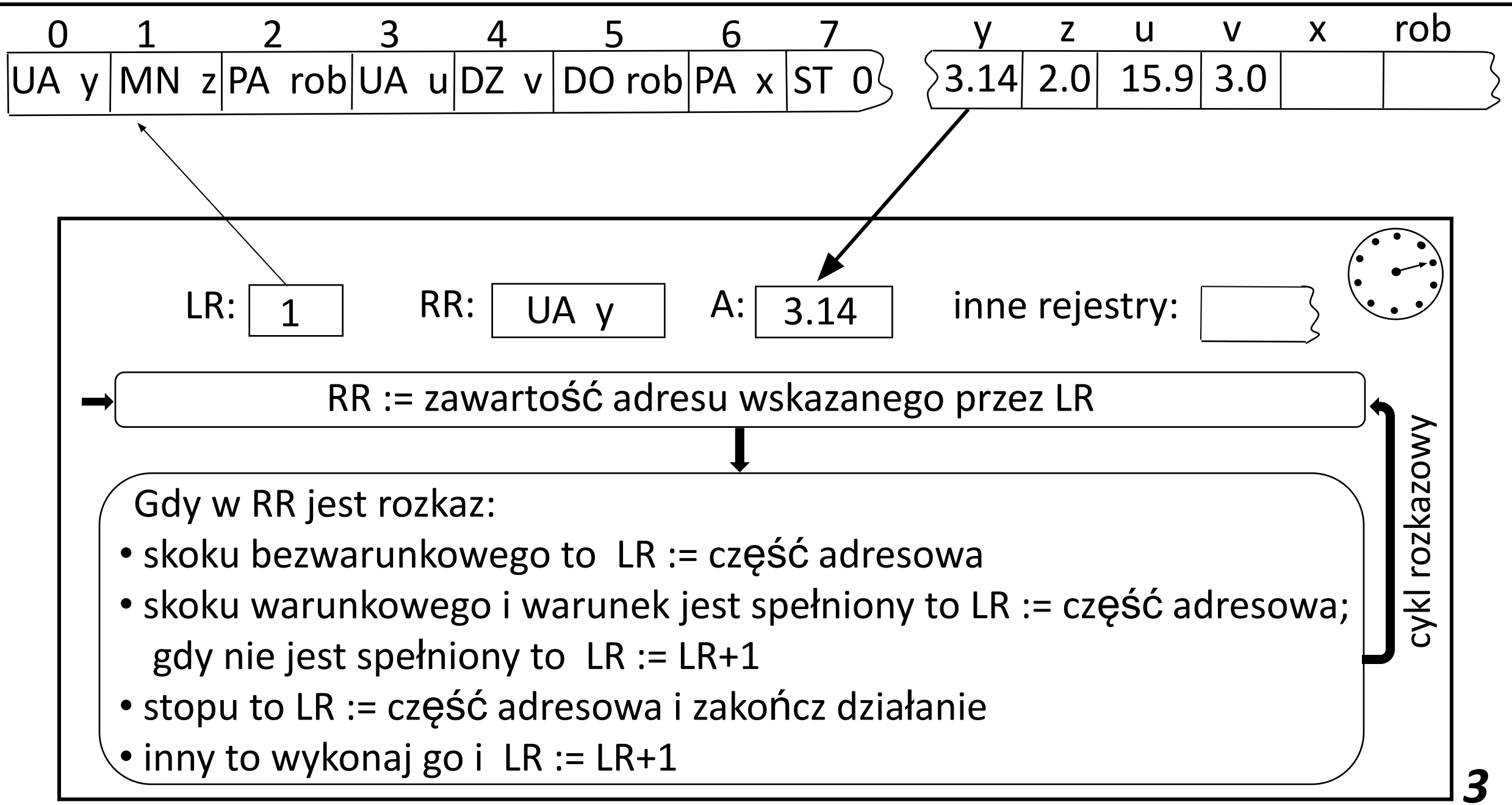
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0

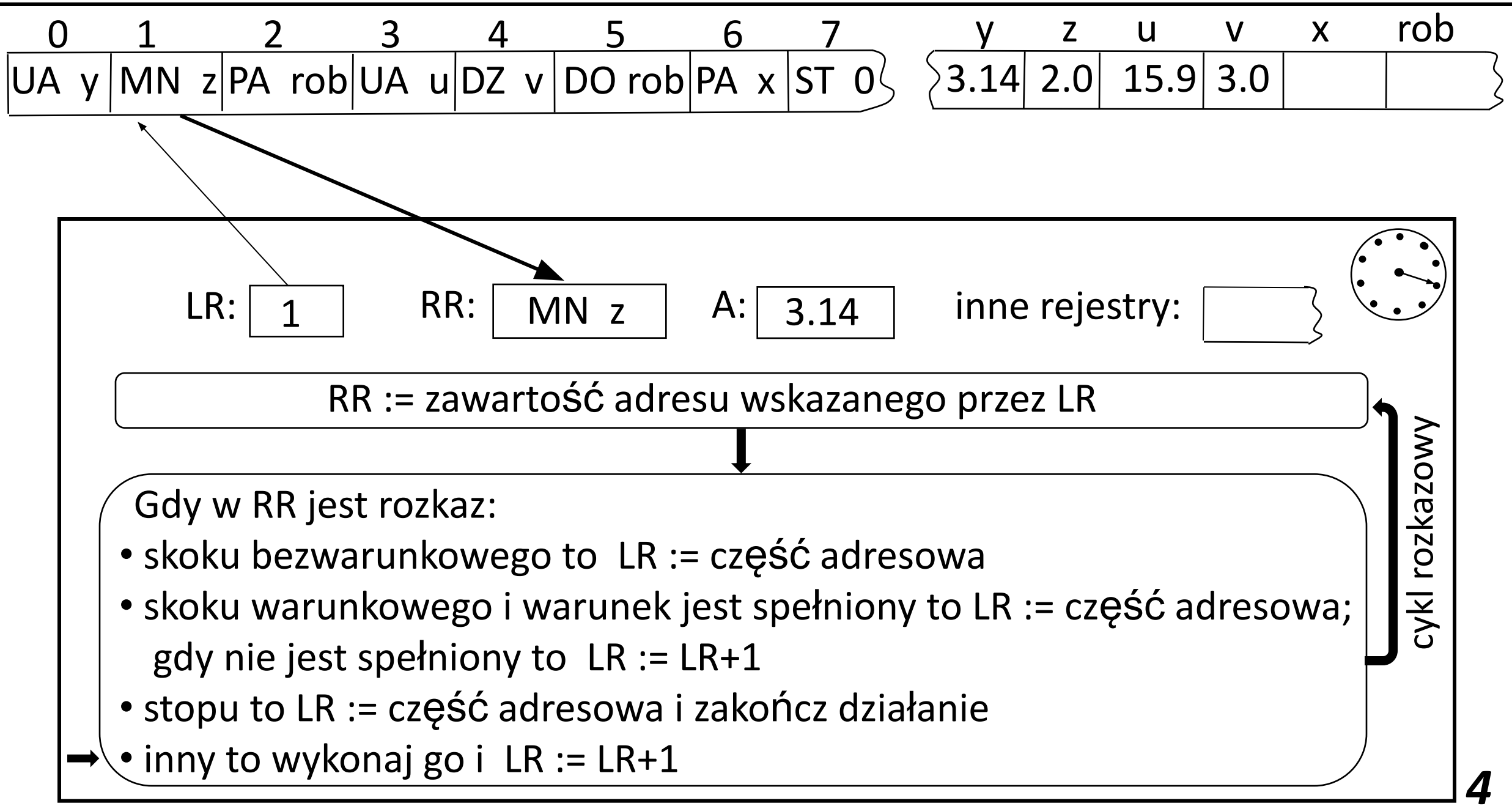
Symulacja wykonania programu na schemacie funkcjonalnym komputera 1-procesorowego z wewnętrznym sterowaniem

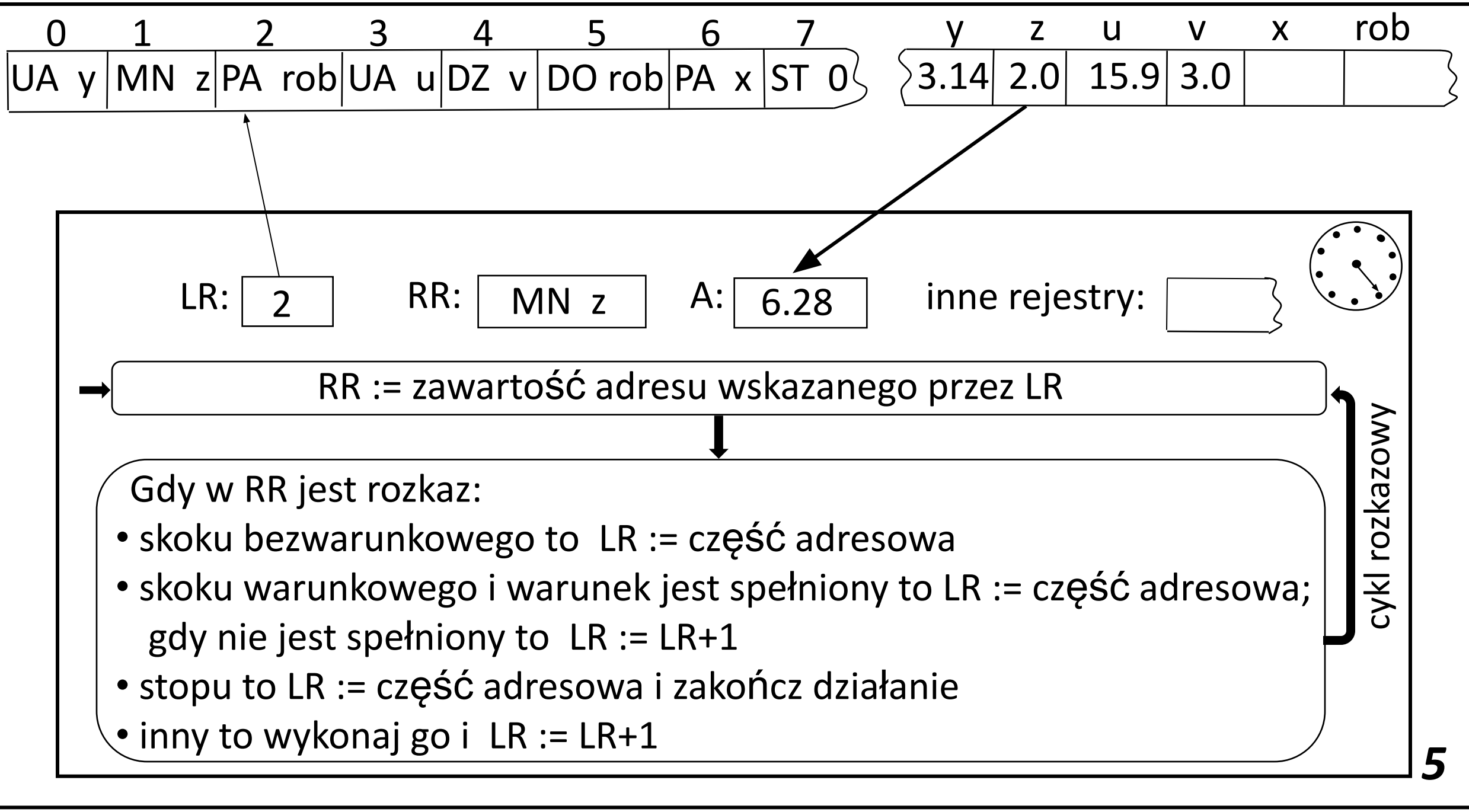
LR – licznik rozkazów RR – rejestr rozkazów A - akumulator

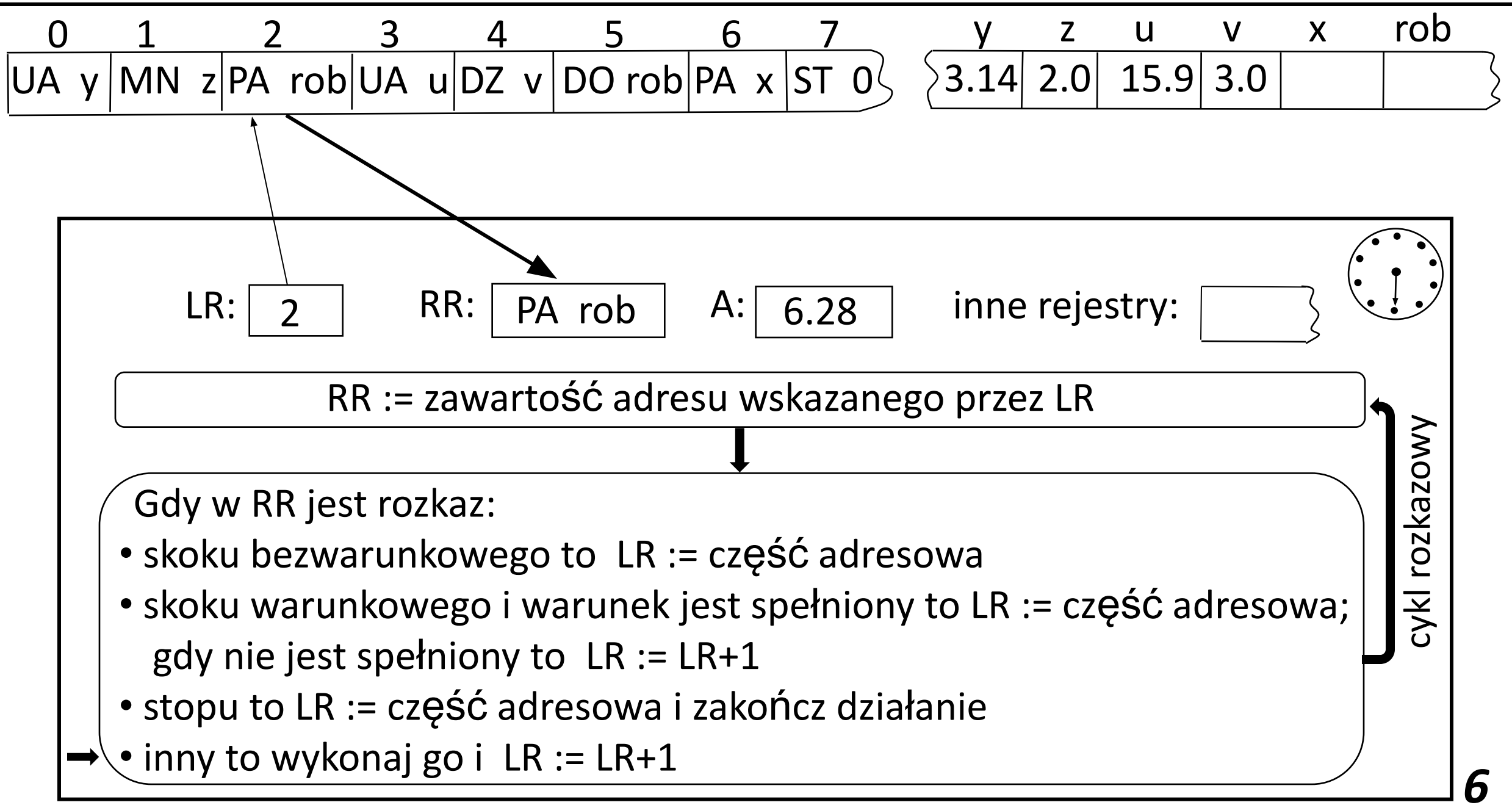

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 0
RR:
A:
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy
1

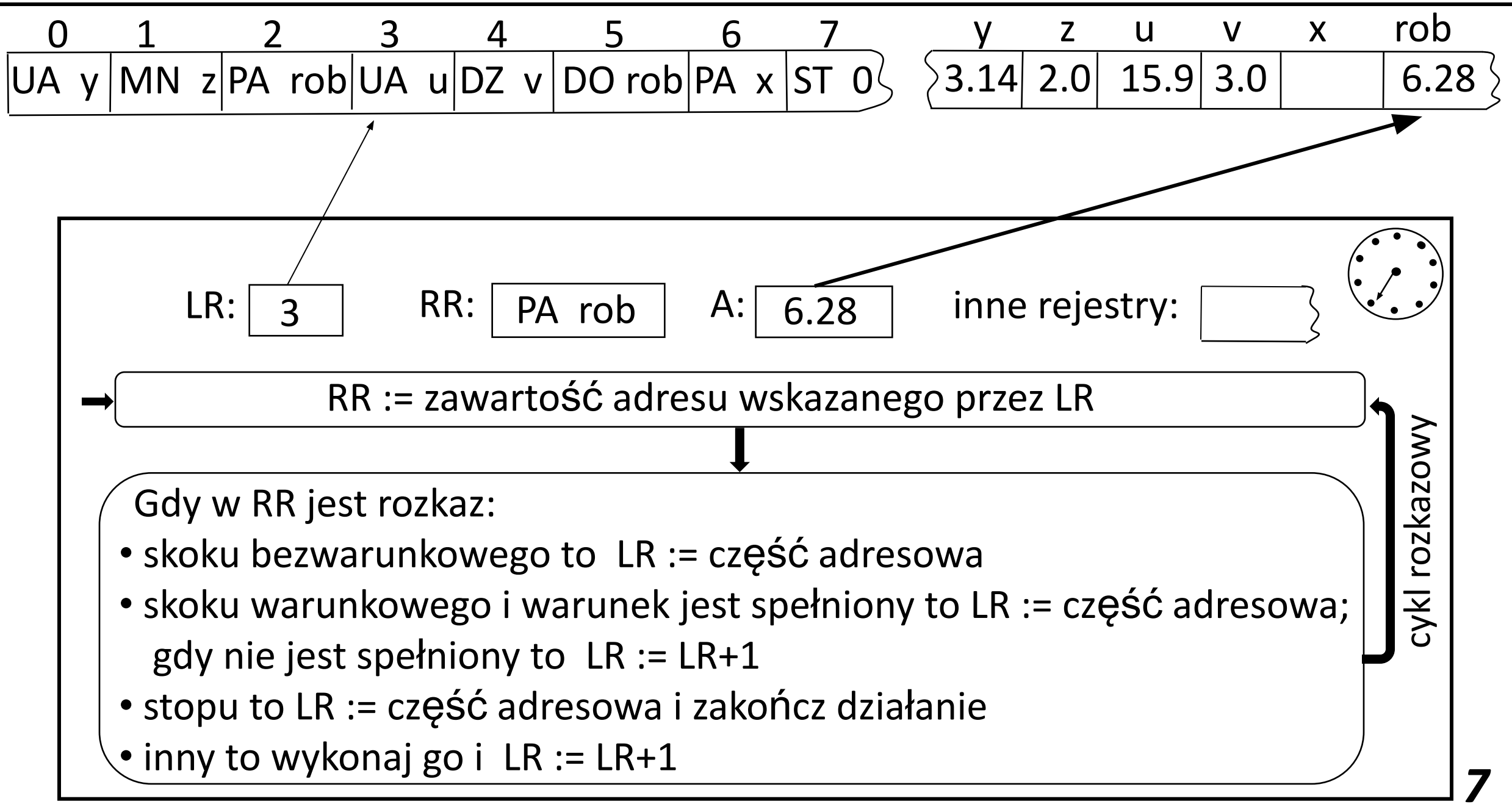

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 0
RR: UA y
A:
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy
2

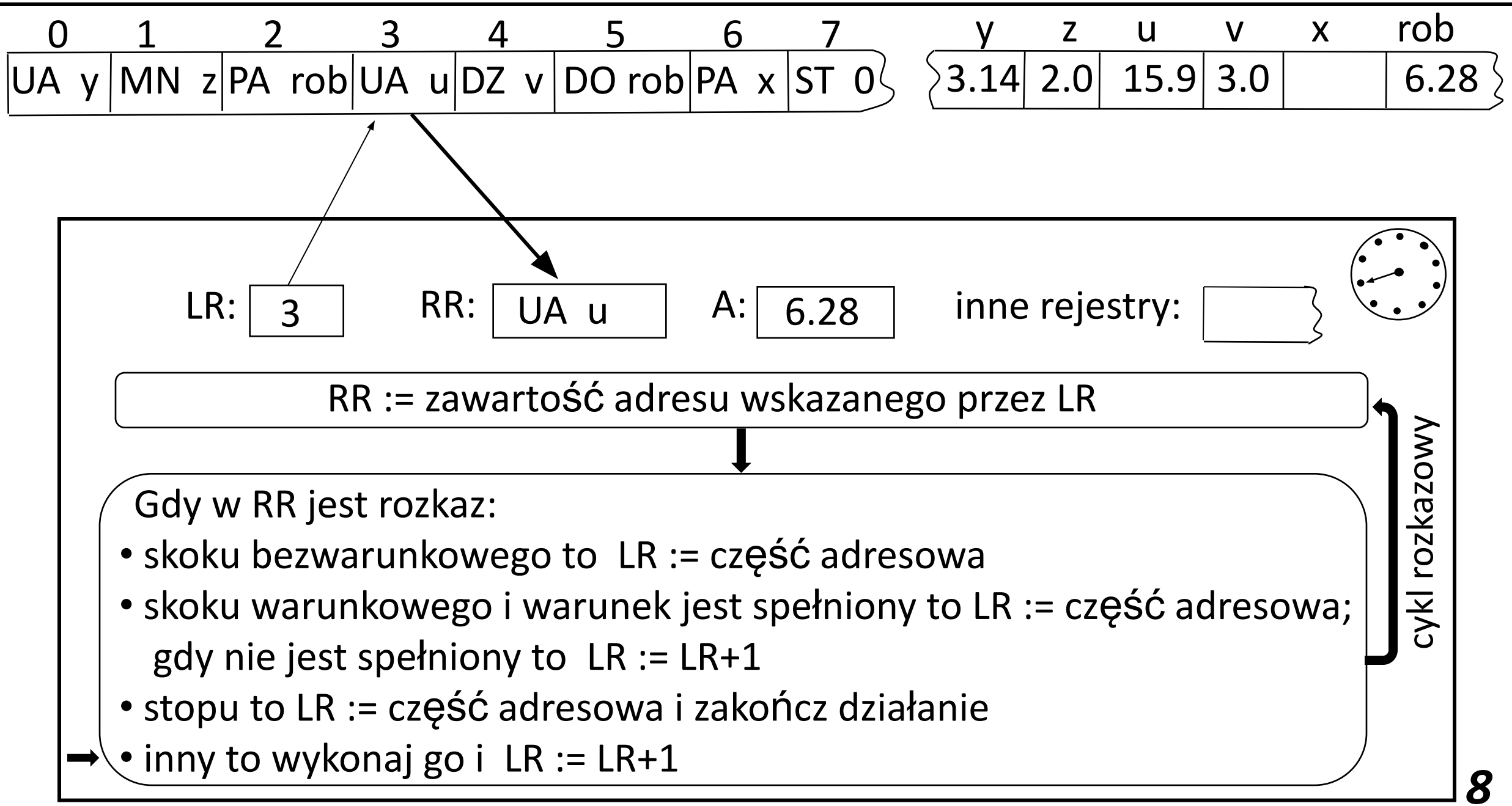

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 1
RR: UA y
A: 3.14
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

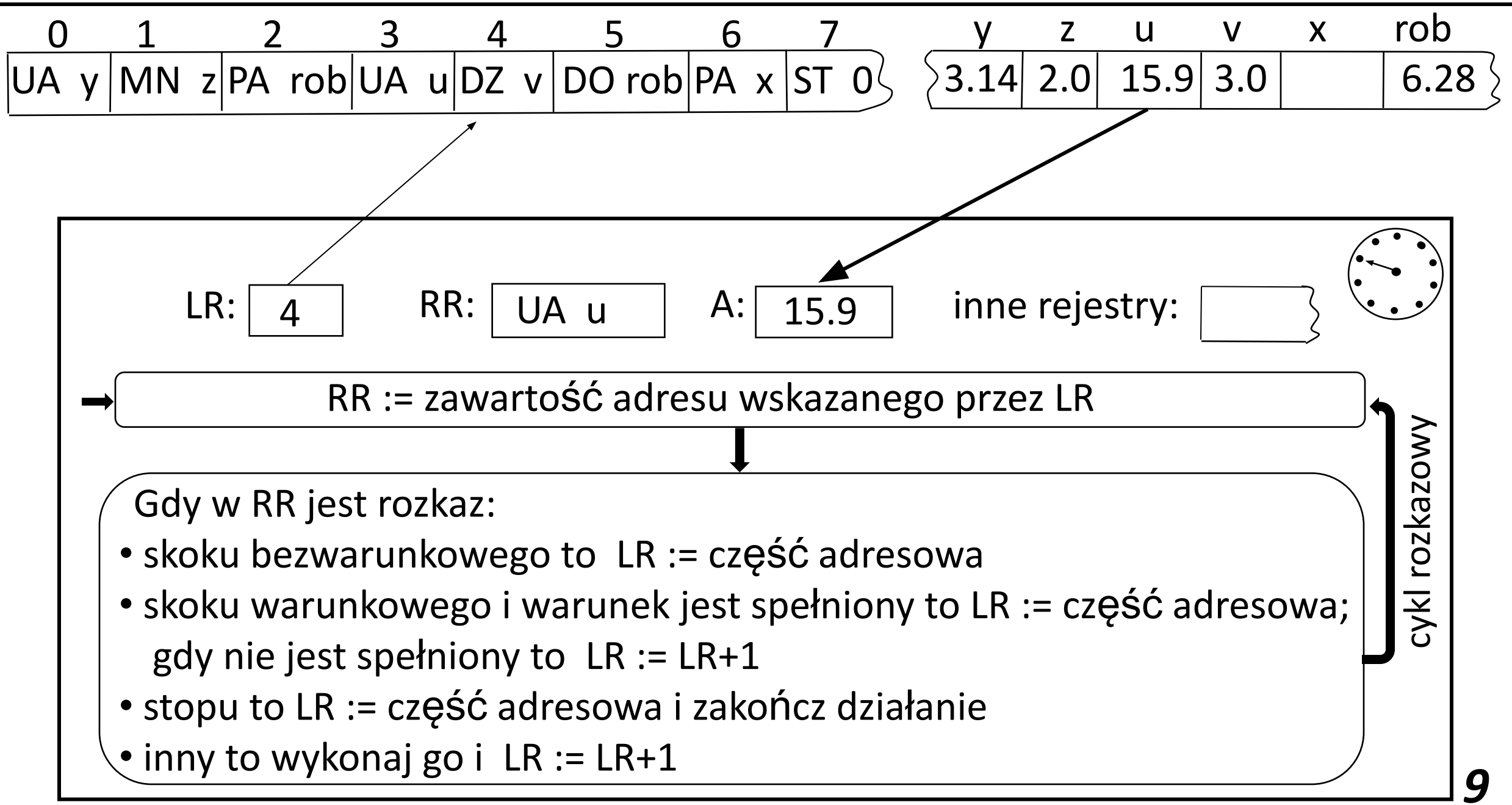

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 1
RR: MN z
A: 3.14
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

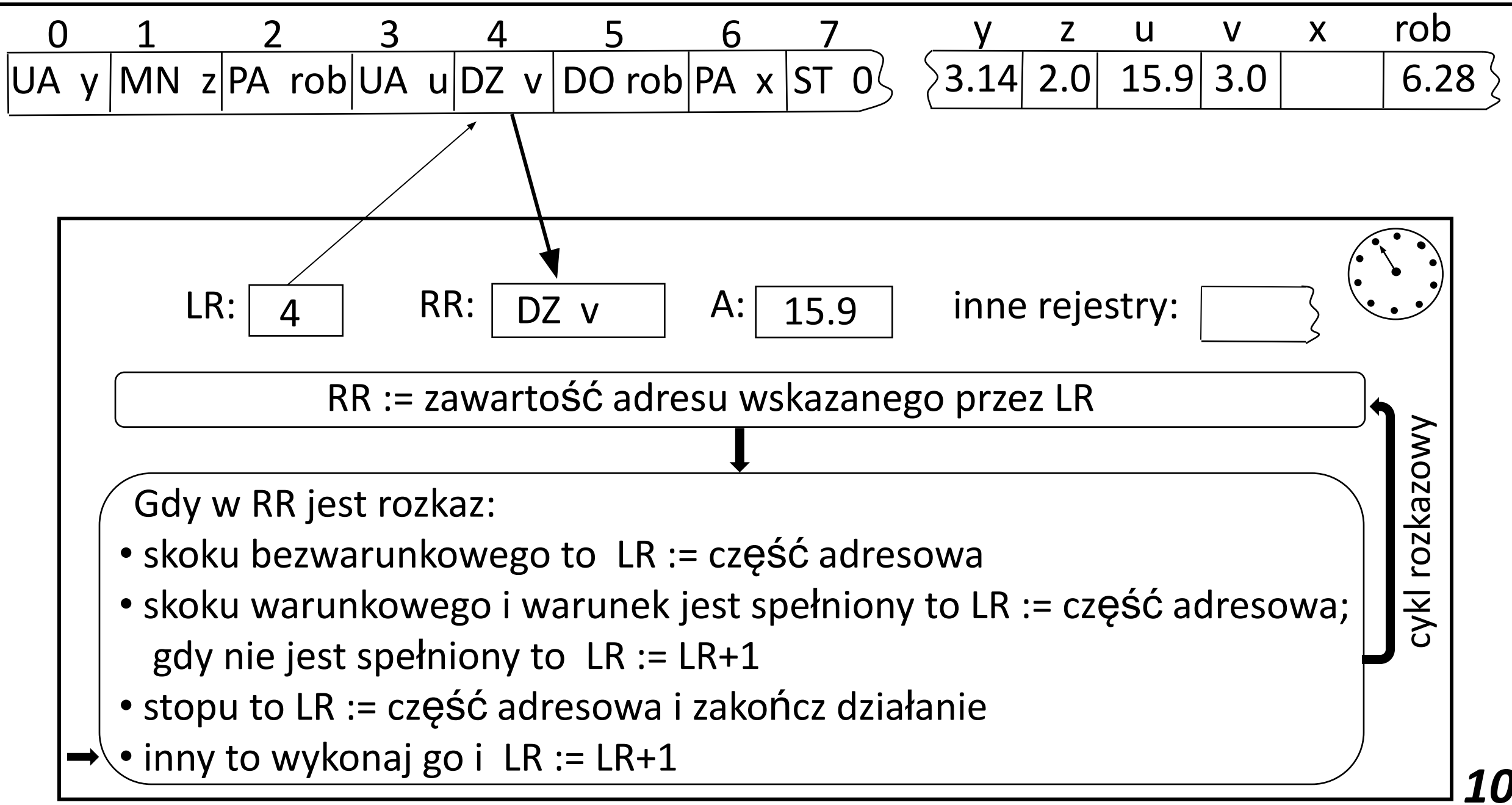

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 2
RR: MN z
A: 6.28
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

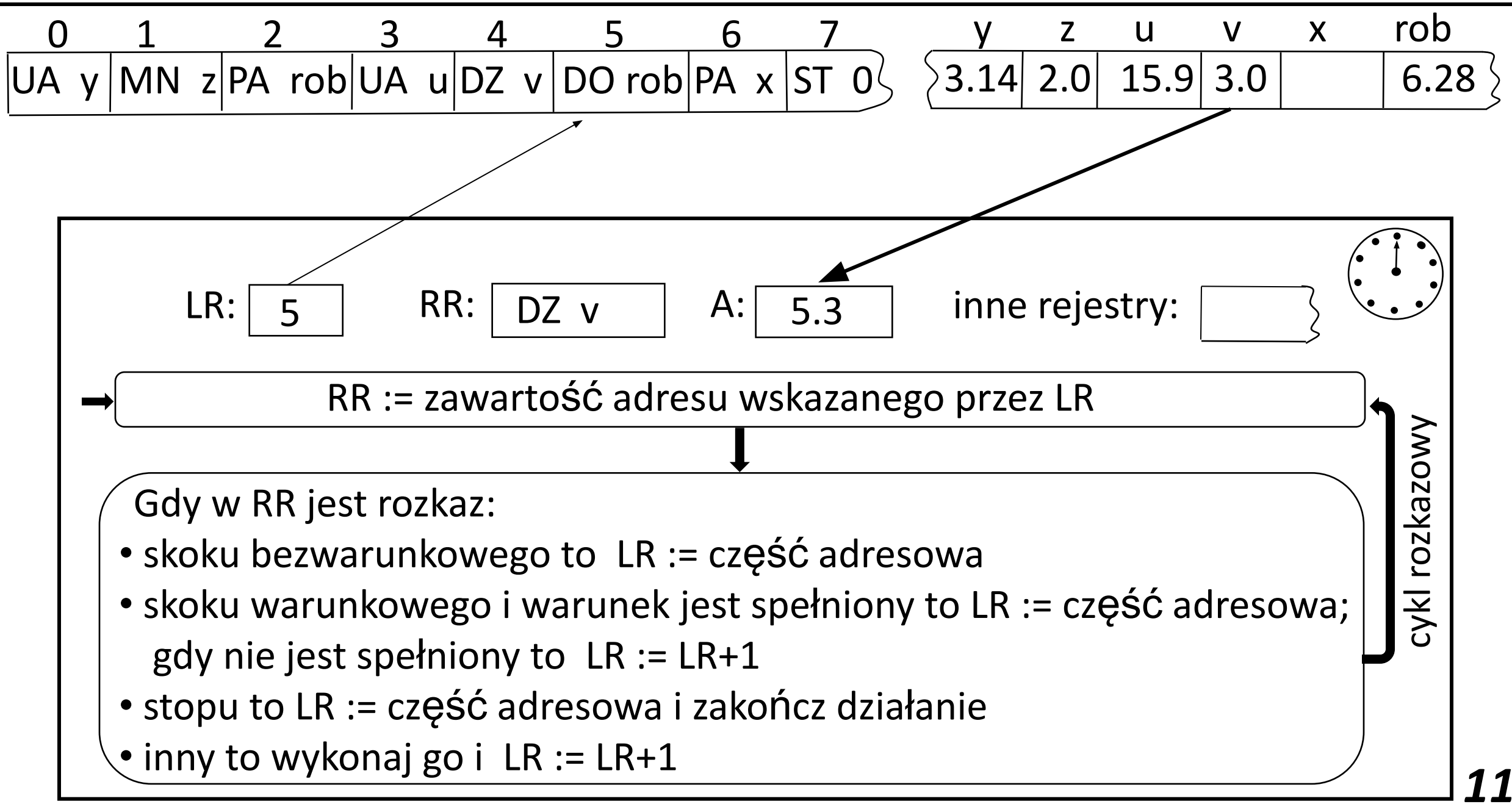

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
LR: 2
RR: PA rob
A: 6.28
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

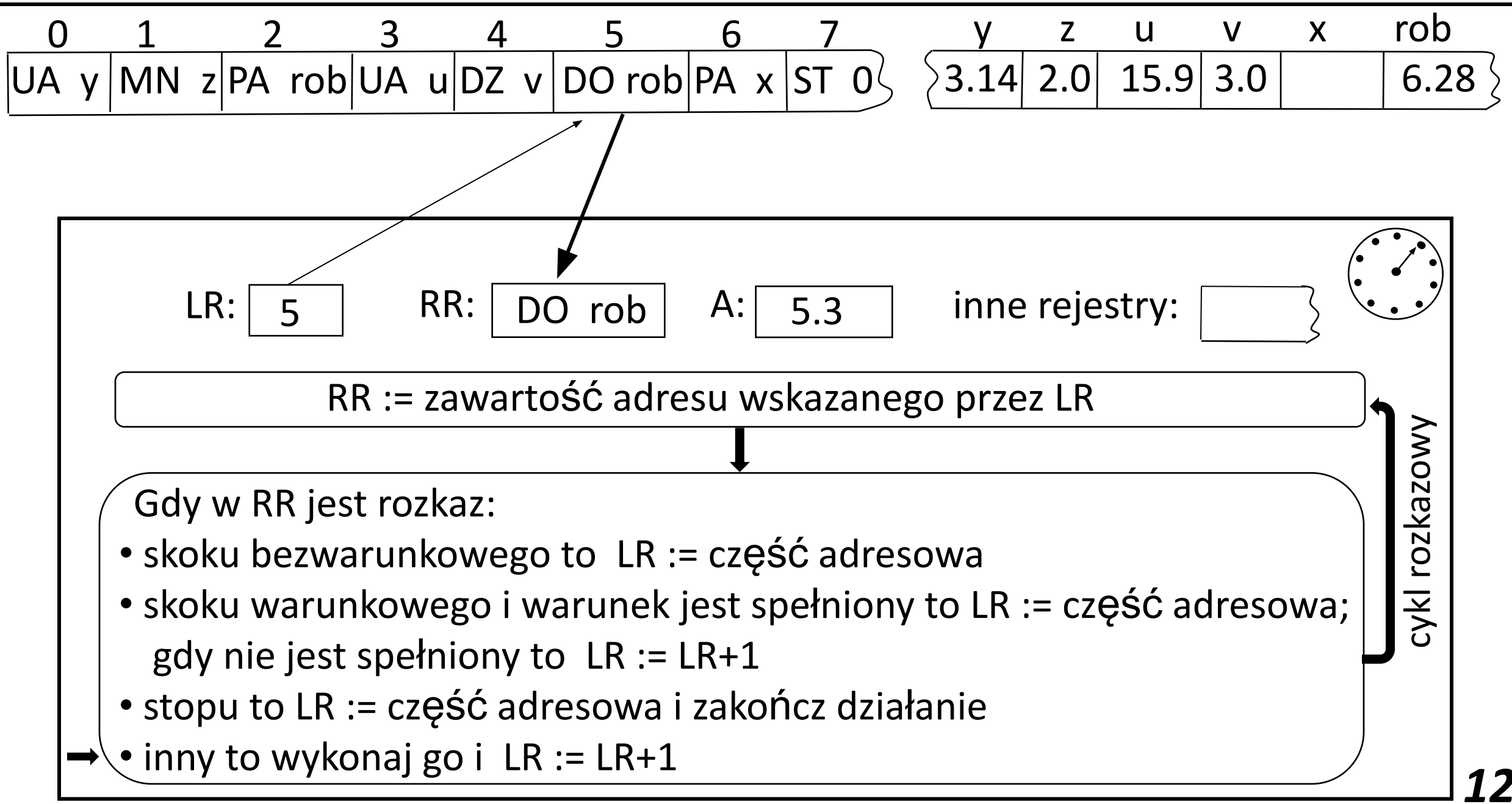

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
6.28
LR: 3
RR: PA rob
A: 6.28
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

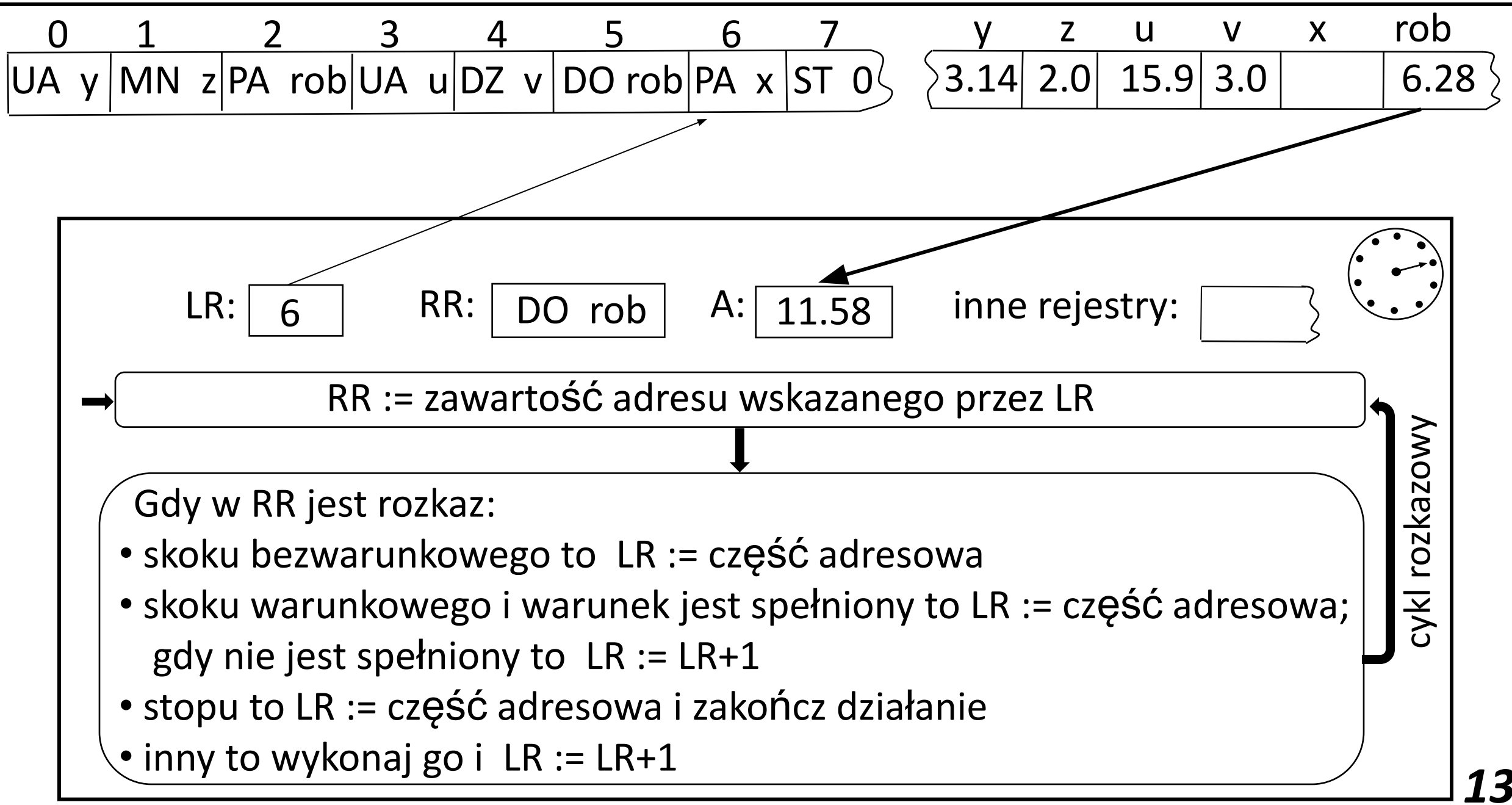

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14 2.0 15.9 3.0 6.28
LR: 3
RR: UA u
A: 6.28
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

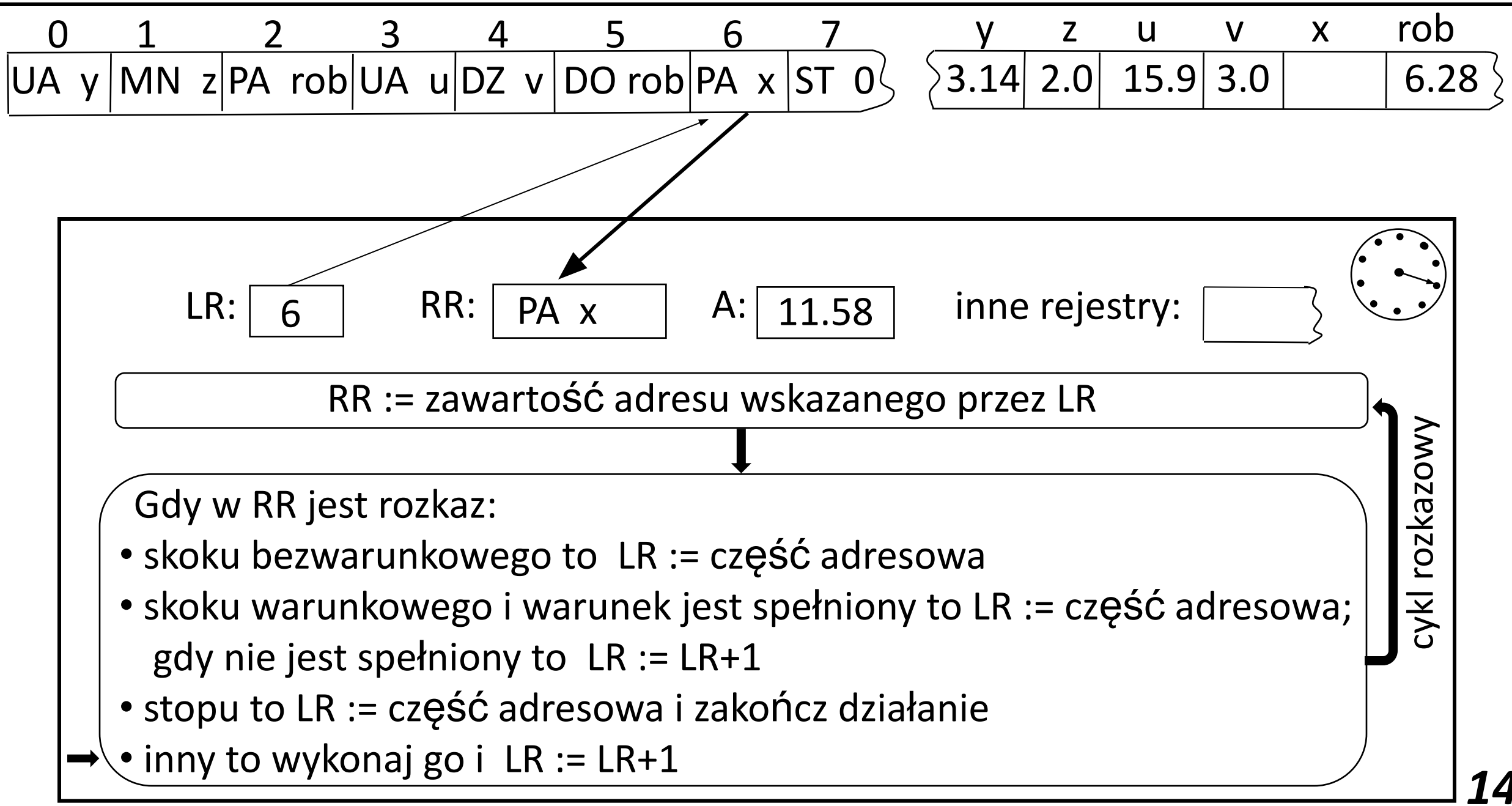

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
6.28
LR: 4
RR: UA u
A: 15.9
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

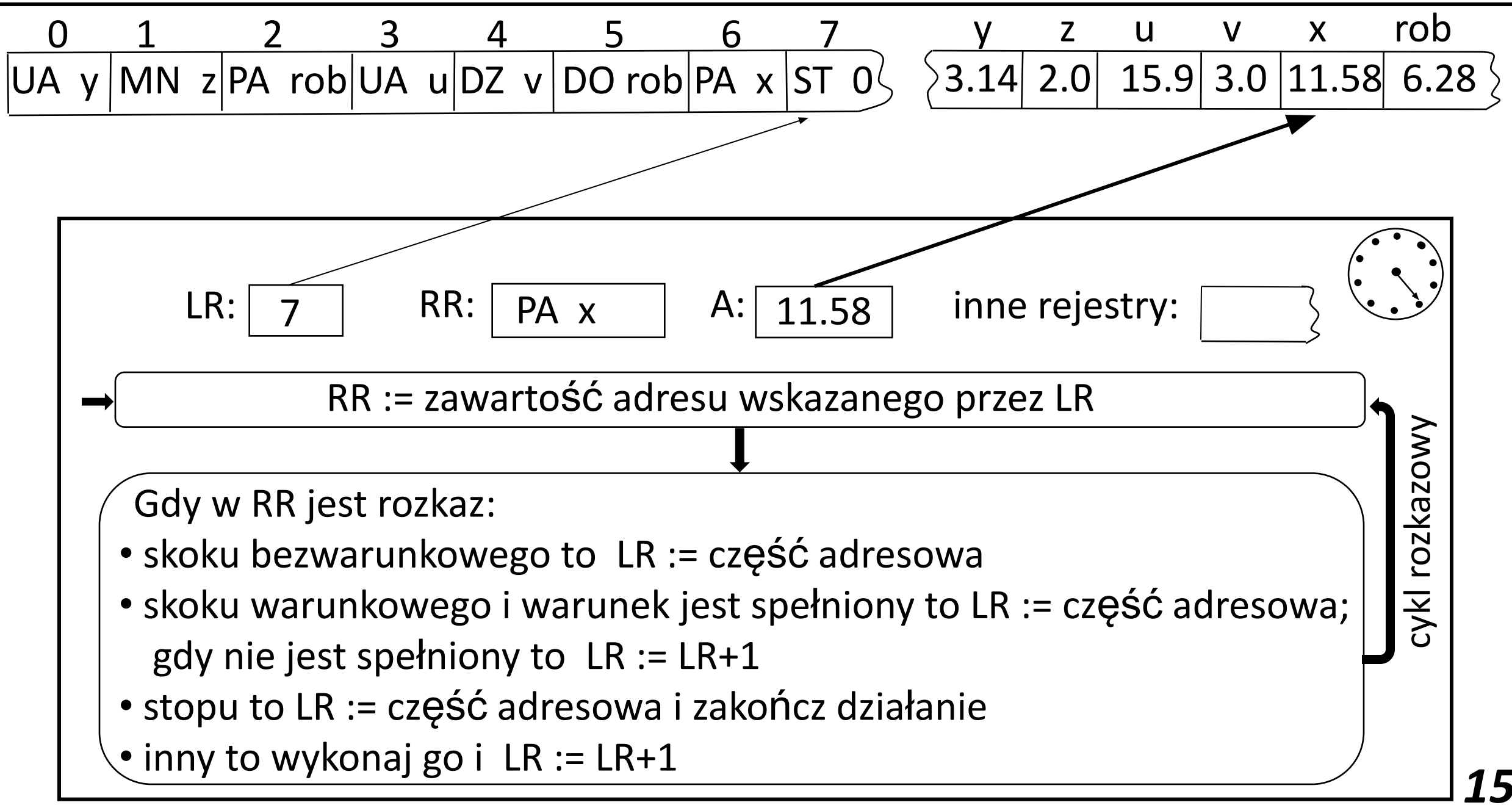

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14 2.0 15.9 3.0 6.28
LR: 4
RR: DZ v
A: 15.9
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

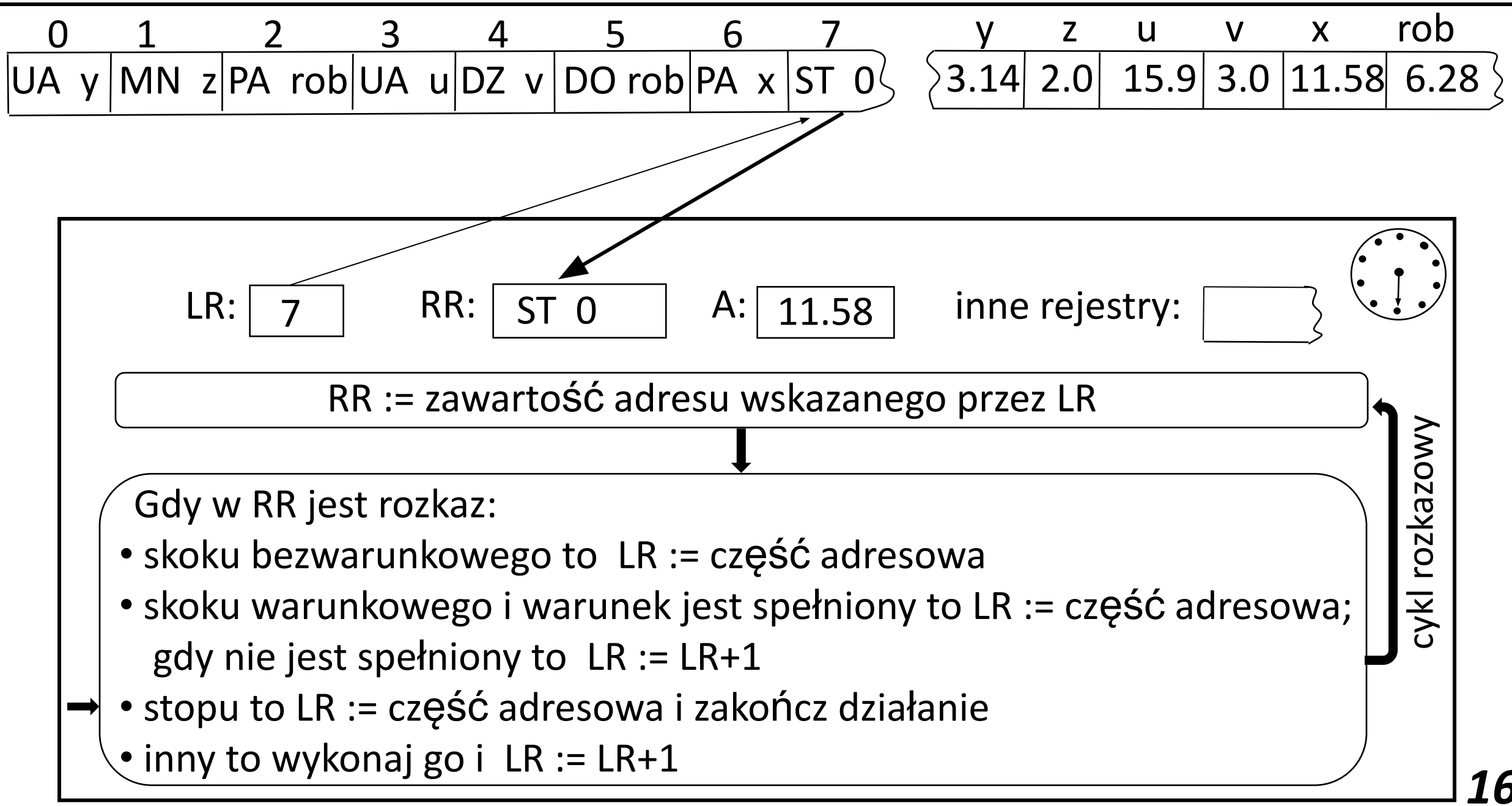

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
6.28
LR: 5
RR: DZ v
A: 5.3
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

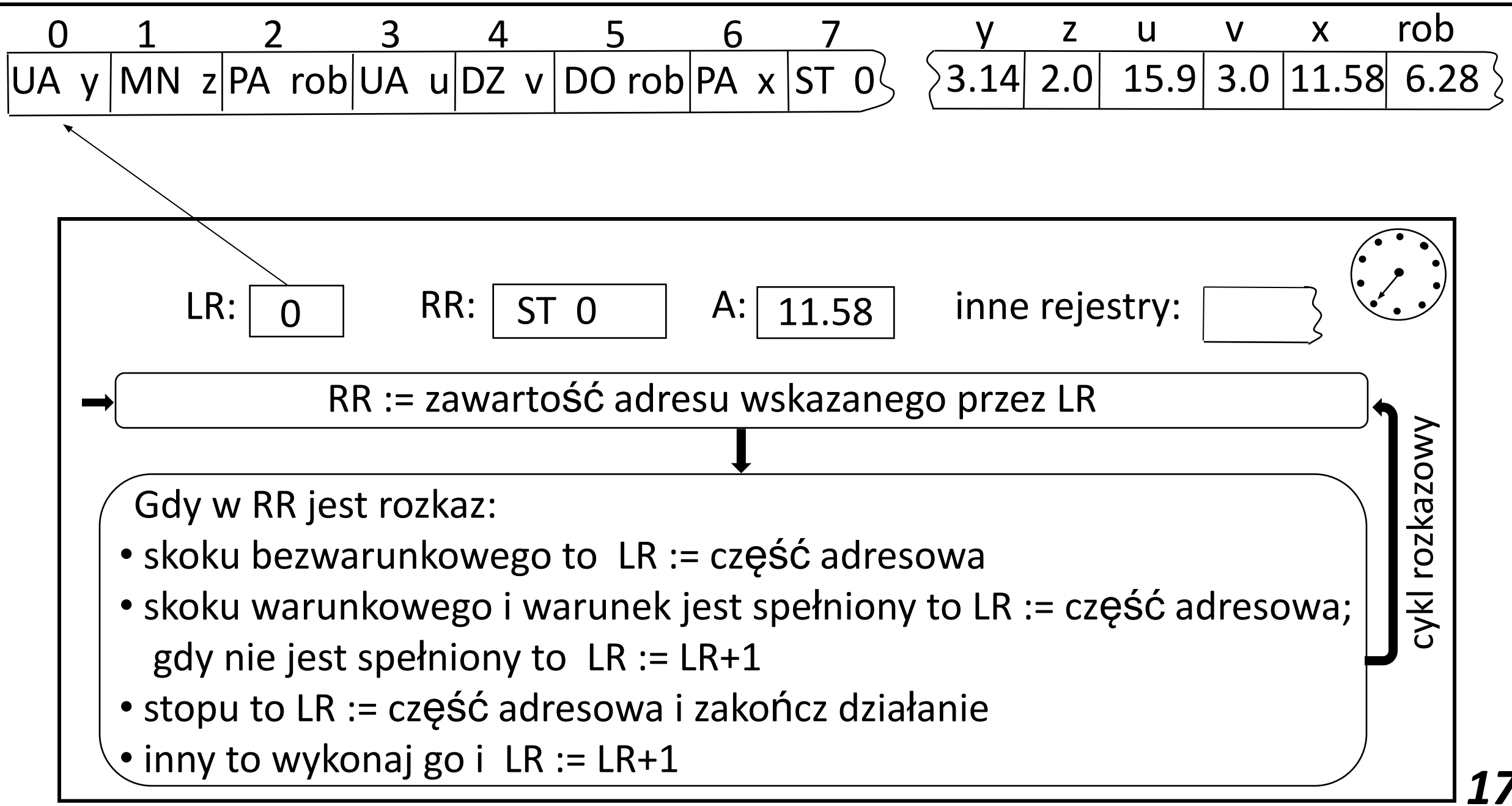

0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
6.28
LR: 5
RR: DO rob
A: 5.3
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy


0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14
2.0
15.9
3.0
6.28
LR: 6
RR: DO rob
A: 11.58
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy


0 1 2 3 4 5 6 7
UA y
MN z
PA rob
UA u
DZ v
DO rob
PA x
ST 0
y z u v x rob
3.14 2.0 15.9 3.0 6.28
LR: 6
RR: PA x
A: 11.58
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy


0 1 2 3 4 5 6 7
UA y | MN z | PA rob | UA u | DZ v | DO rob | PA x | ST 0
y z u v x rob
3.14 | 2.0 | 15.9 | 3.0 | 11.58 | 6.28
LR: 7
RR: PA x
A: 11.58
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy


0 1 2 3 4 5 6 7
UA y | MN z | PA rob | UA u | DZ v | DO rob | PA x | ST 0
y z u v x rob
3.14 | 2.0 | 15.9 | 3.0 | 11.58 | 6.28
LR: 7
RR: ST 0
A: 11.58
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy


0 1 2 3 4 5 6 7
UA y | MN z | PA rob | UA u | DZ v | DO rob | PA x | ST 0
y z u v x rob
3.14 | 2.0 | 15.9 | 3.0 | 11.58 | 6.28
LR: 0
RR: ST 0
A: 11.58
inne rejestry:
RR := zawartość adresu wskazanego przez LR
Gdy w RR jest rozkaz:
• skoku bezwarunkowego to LR := część adresowa
• skoku warunkowego i warunek jest spełniony to LR := część adresowa;
gdy nie jest spełniony to LR := LR+1
• stopu to LR := część adresowa i zakończ działanie
• inny to wykonaj go i LR := LR+1
cykl rozkazowy

Animowana symulacja działania komputera wieloprogramowego (z podziałem czasu).

Komputer ma rozkazy przerwania (wstrzymywania i wznawiania procesów) i obsługi kolejki

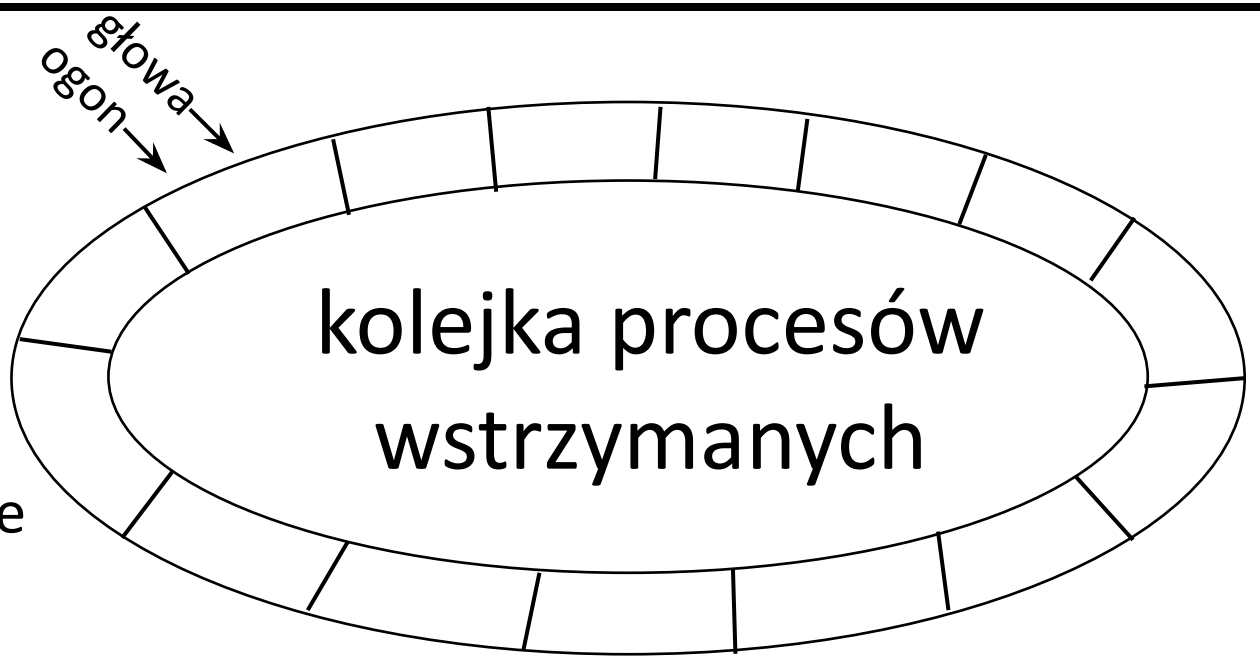
Stan programów:

przed działaniem

działa

wstrzymany

zakończył działanie



pamię

ć

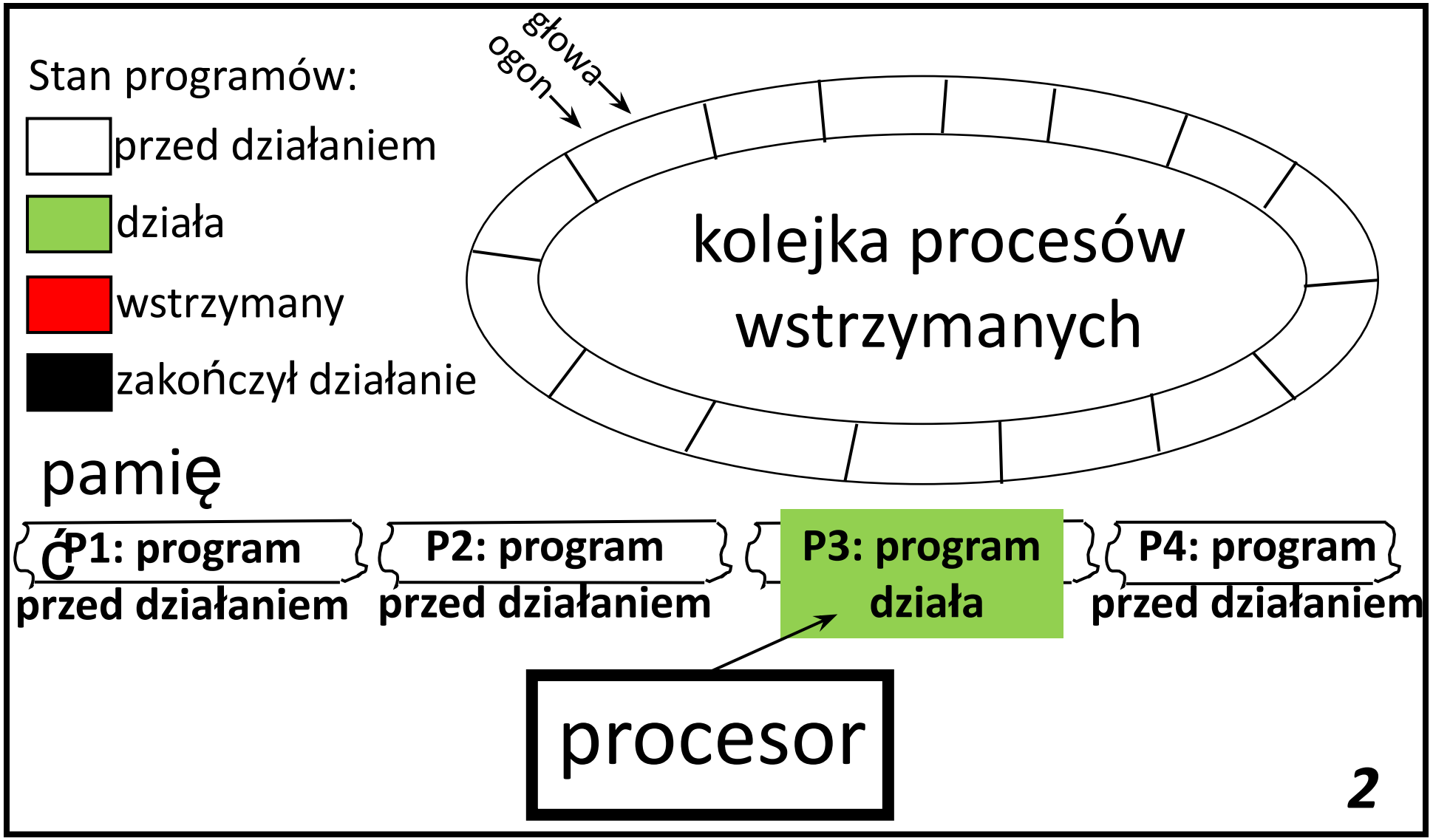
P1: program przed działaniem

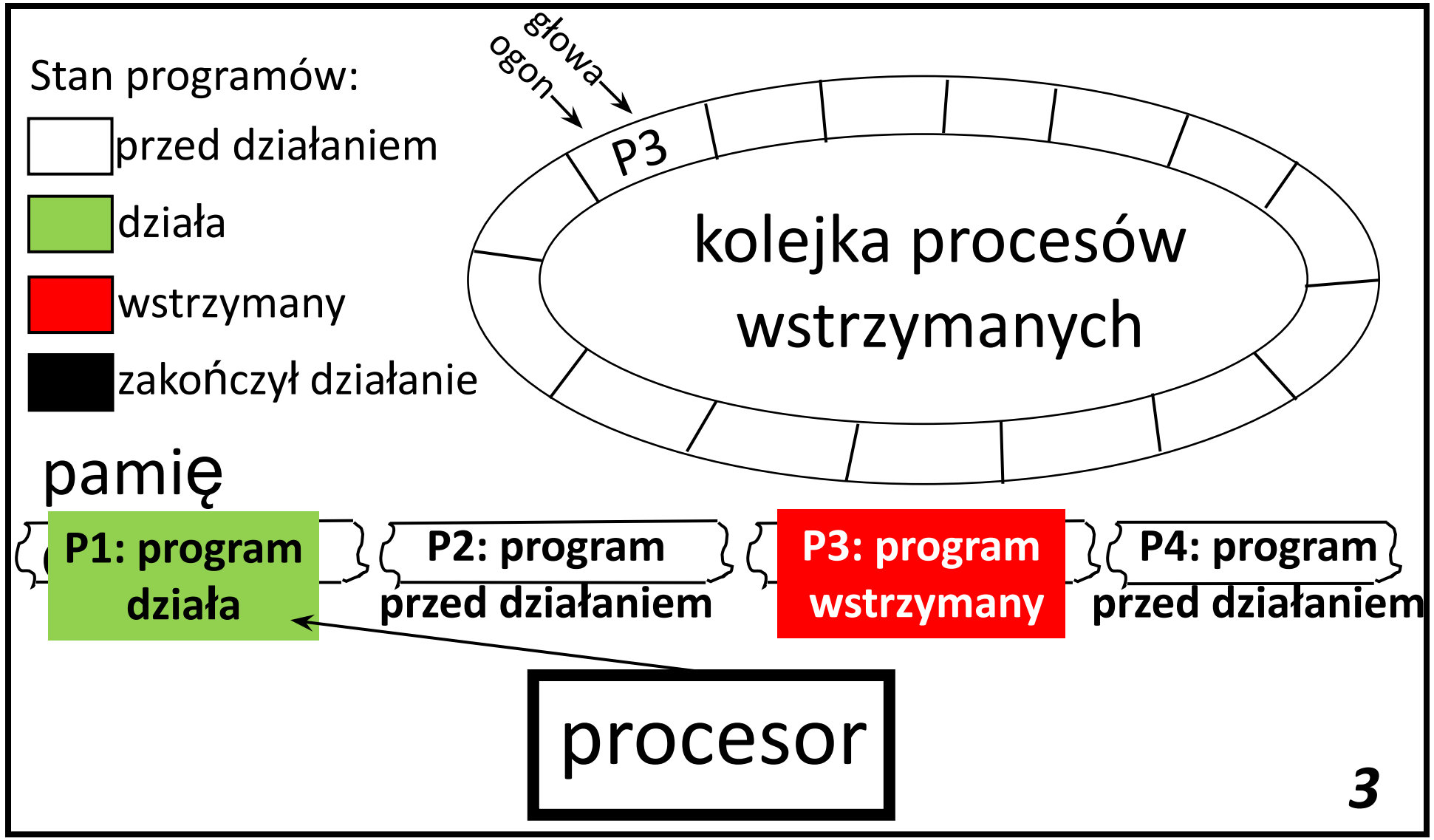
P2: program przed działaniem

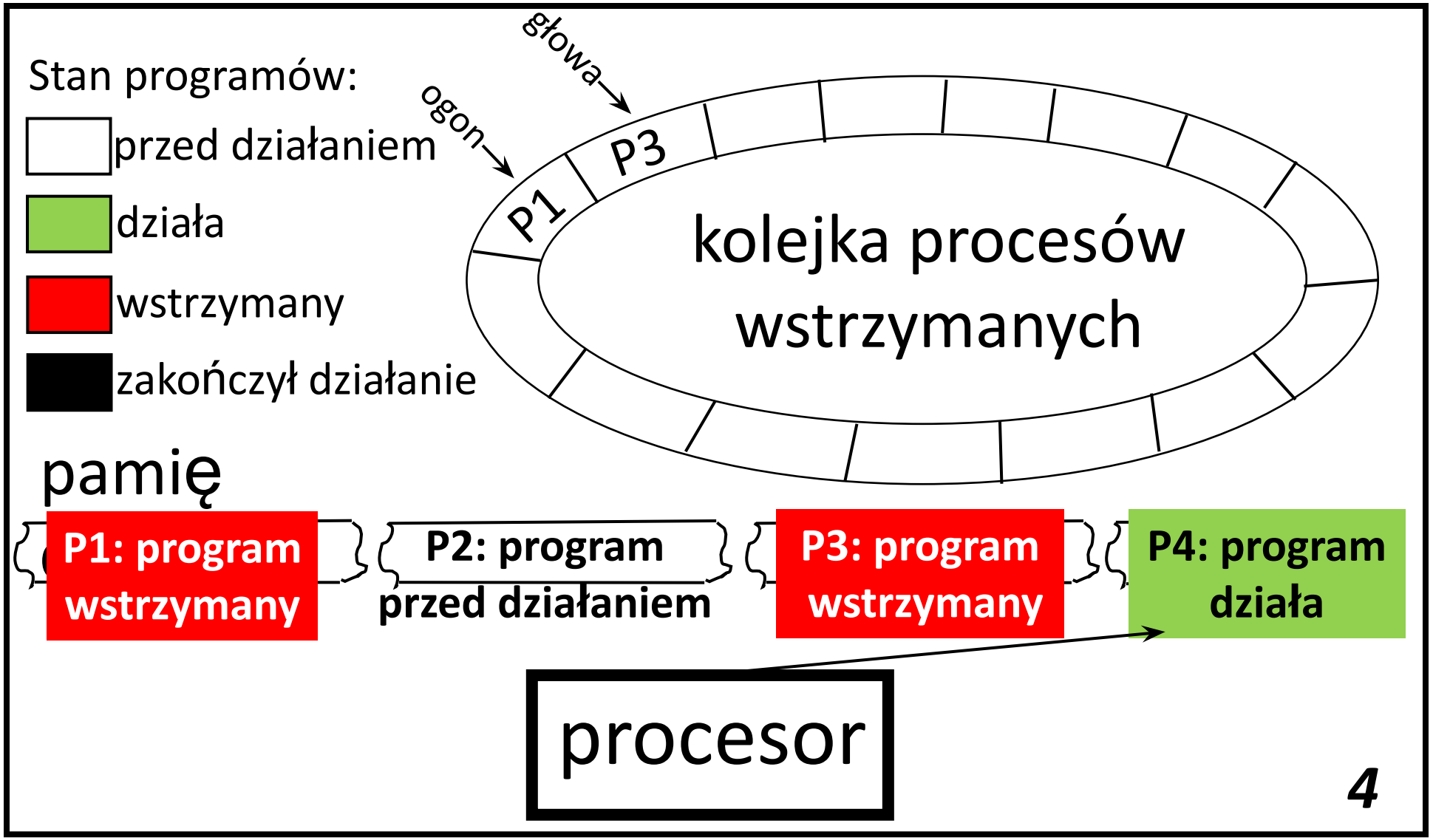
P3: program przed działaniem

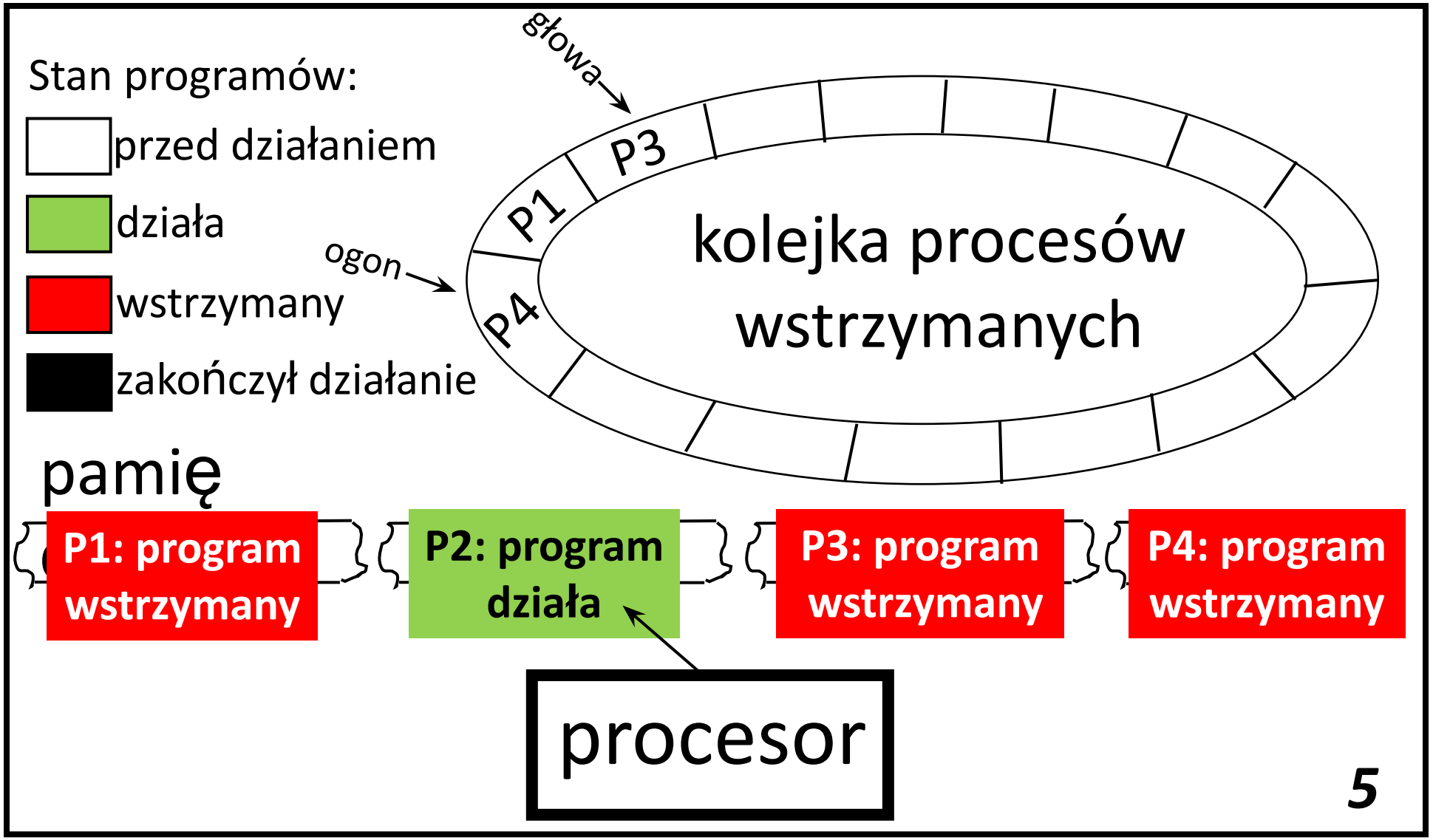
P4: program przed działaniem

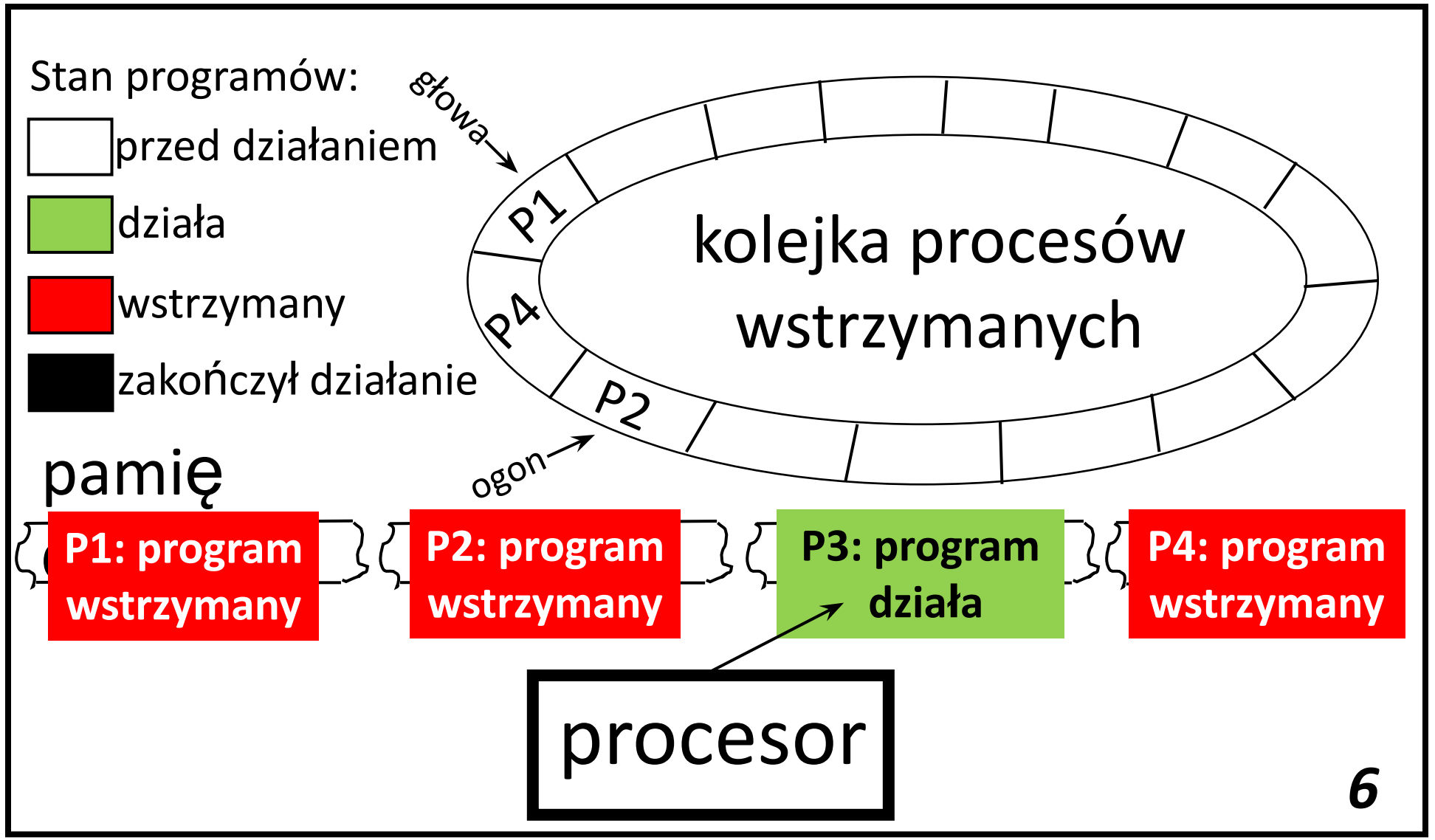
procesor

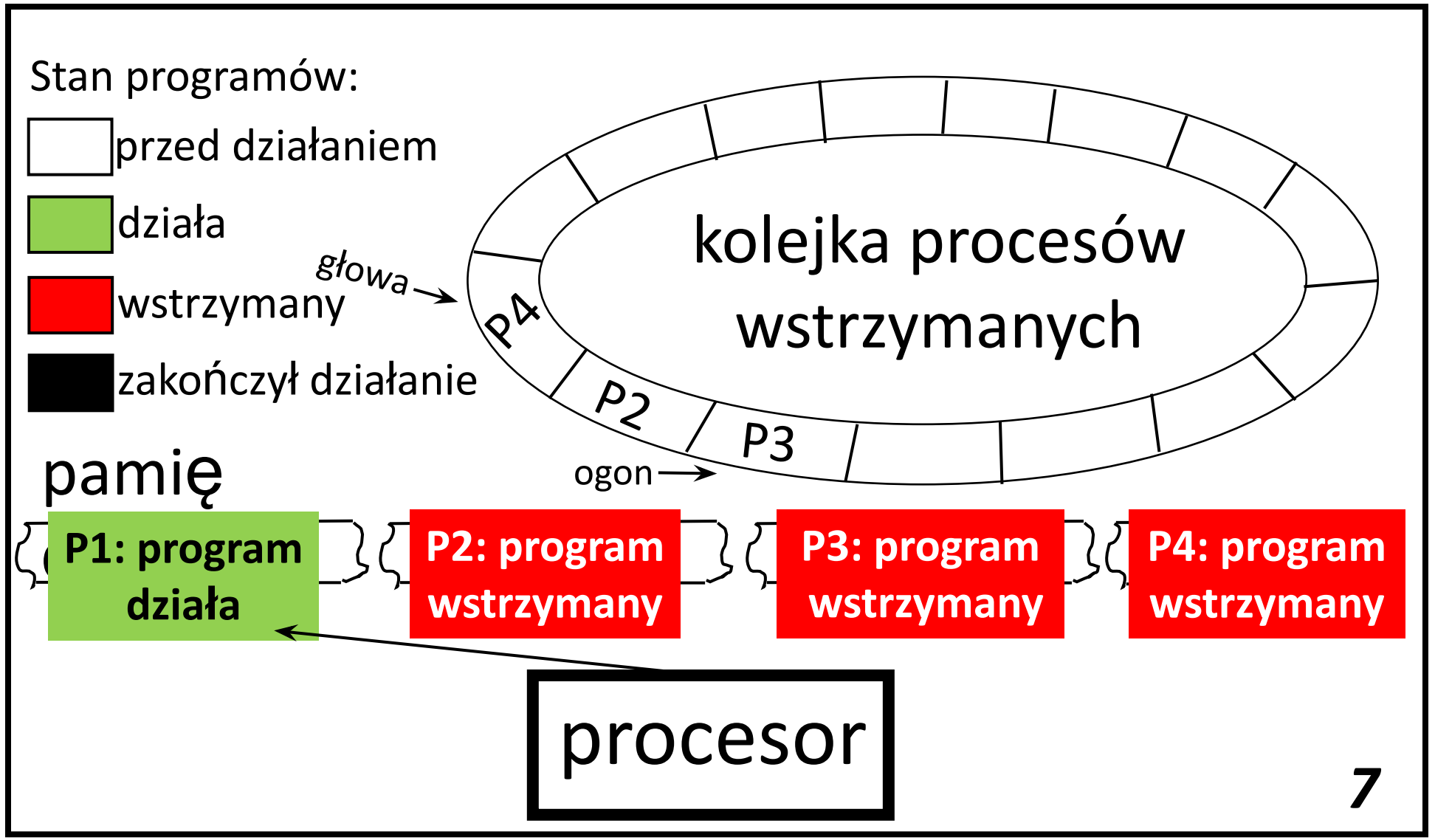

Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
ogon
kolejka procesów wstrzymanych
pamię
P1: program przed działaniem
P2: program przed działaniem
P3: program działa
P4: program przed działaniem
procesor

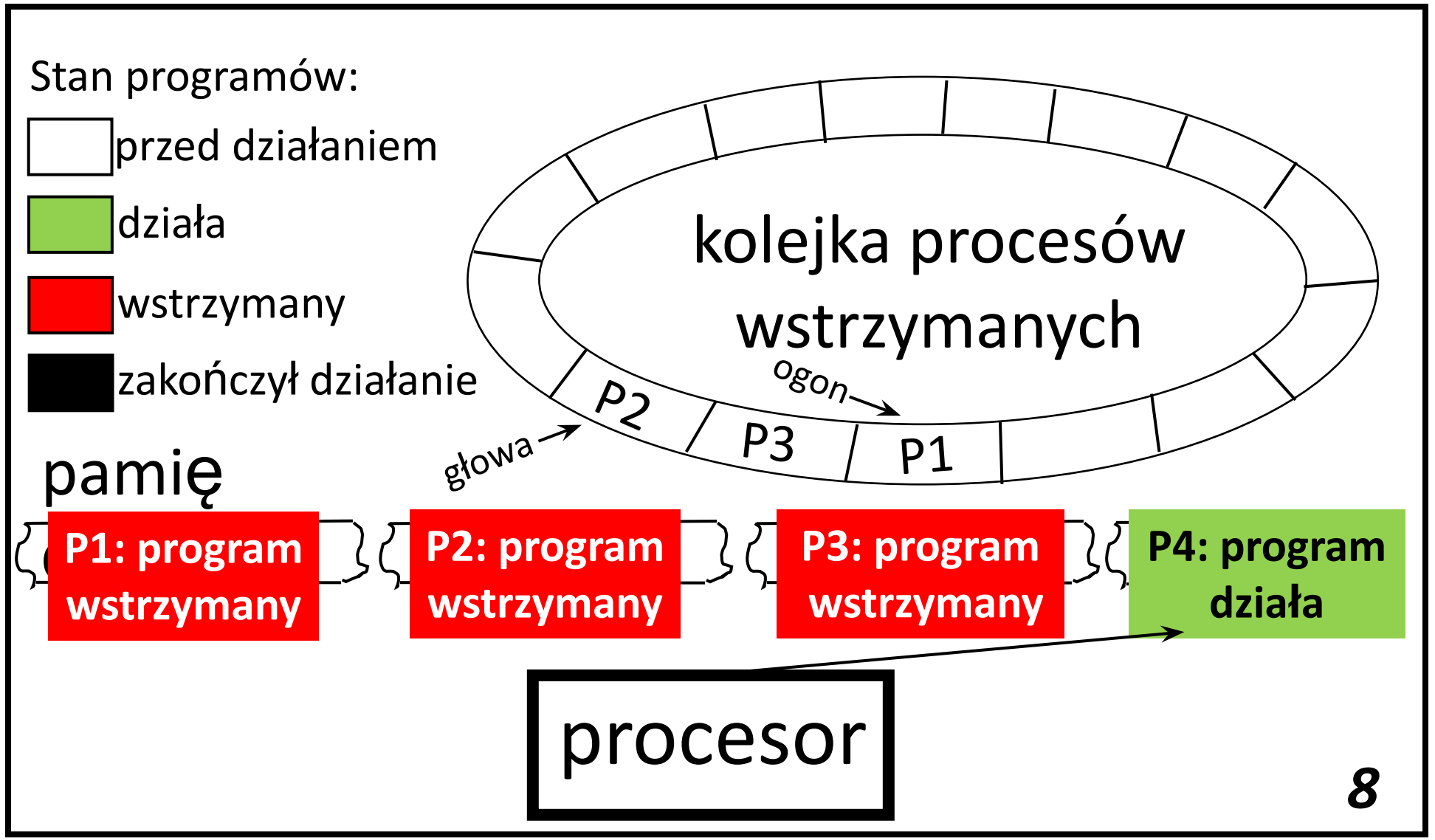

Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
ogon
P3
kolejka procesów
wstrzymanych
pamię
P1: program
działa
P2: program
przed działaniem
P3: program
wstrzymany
P4: program
przed działaniem
procesor
3

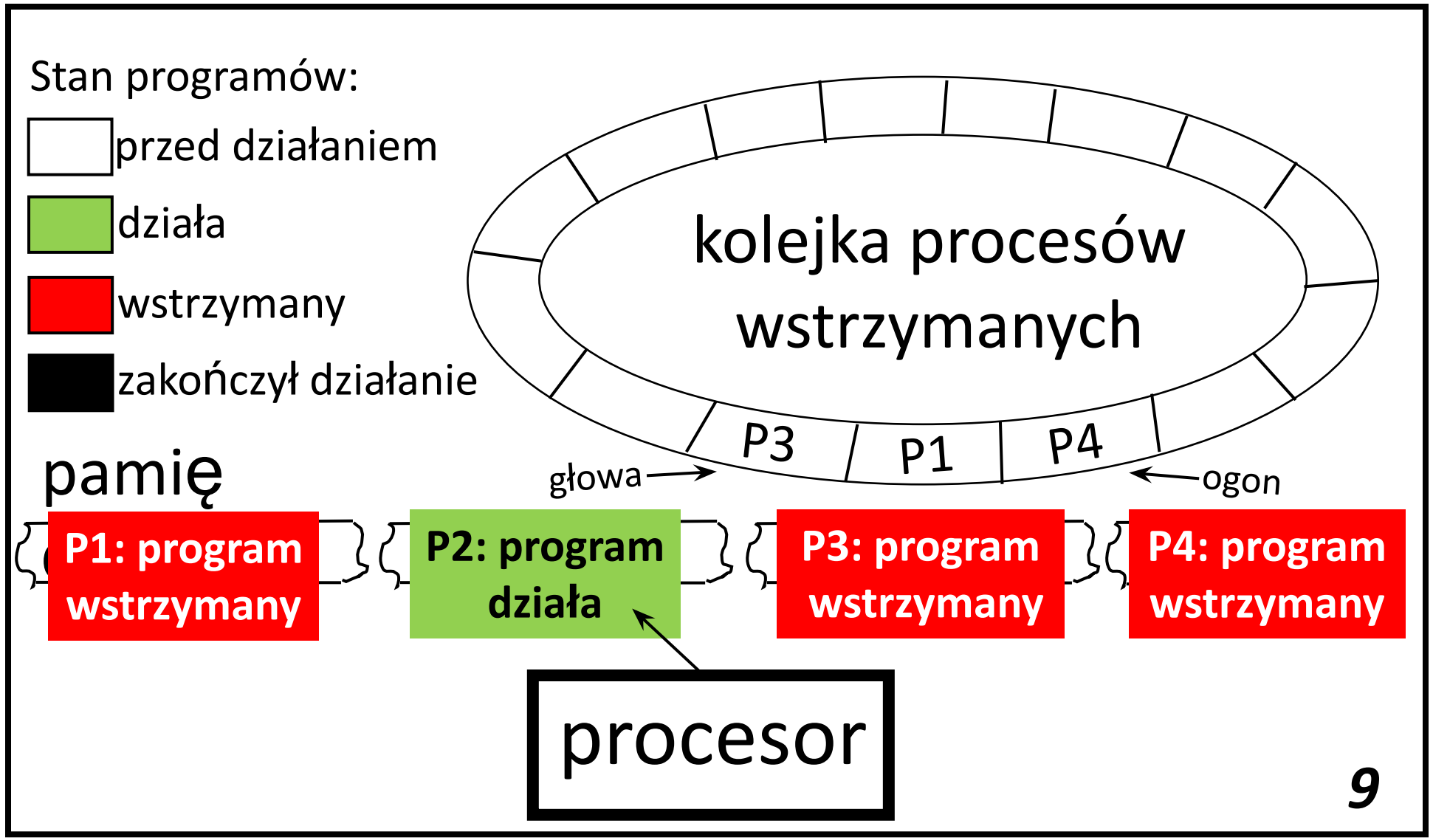

Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
ogon
P3
P1
kolejka procesów wstrzymanych
pamię
P1: program wstrzymany
P2: program przed działaniem
P3: program wstrzymany
P4: program działa
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
ogon
P3
P1
P4
kolejka procesów wstrzymanych
pamię
P1: program wstrzymany
P2: program działa
P3: program wstrzymany
P4: program wstrzymany
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
ogon
P1
P2
P4
kolejka procesów wstrzymanych
pamię
P1: program wstrzymany
P2: program wstrzymany
P3: program działa
P4: program wstrzymany
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
głowa
kolejka procesów wstrzymanych
P4
P2
P3
ogon
pamię
P1: program działa
P2: program wstrzymany
P3: program wstrzymany
P4: program wstrzymany
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
kolejka procesów wstrzymanych
ogon
głowa
P2
P3
P1
pamię
P1: program wstrzymany
P2: program wstrzymany
P3: program wstrzymany
P4: program działa
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
kolejka procesów wstrzymanych
P3
P1
P4
głowa
ogon
pamię
P1: program wstrzymany
P2: program działa
P3: program wstrzymany
P4: program wstrzymany
procesor

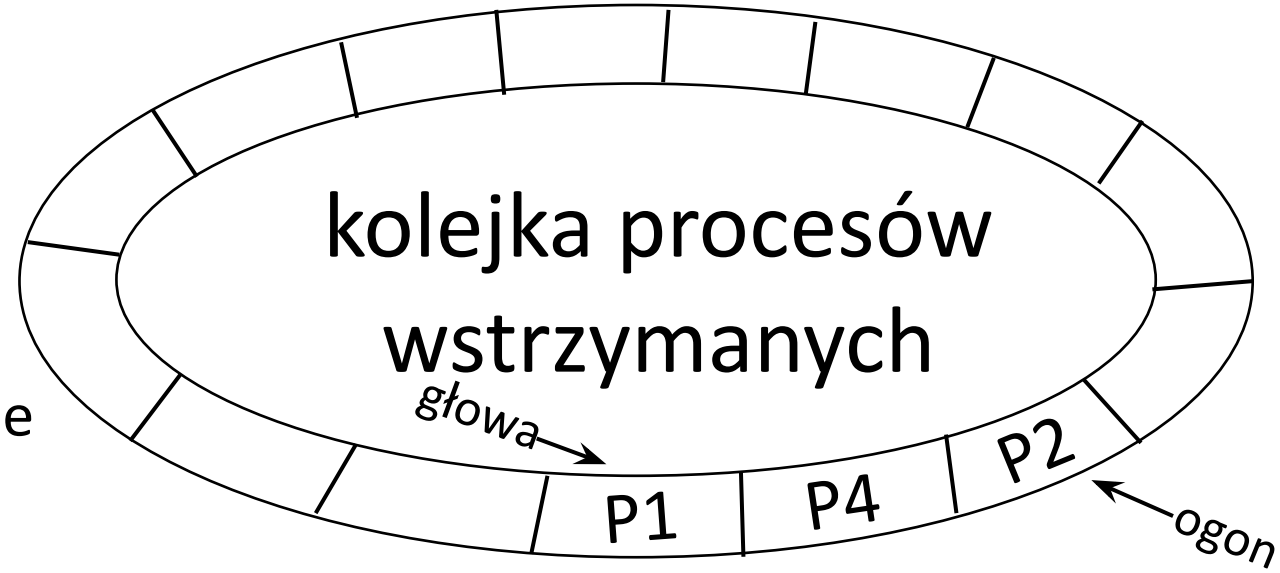
Stan programów:

przed działaniem

działa

wstrzymany

zakończył działanie



pamię

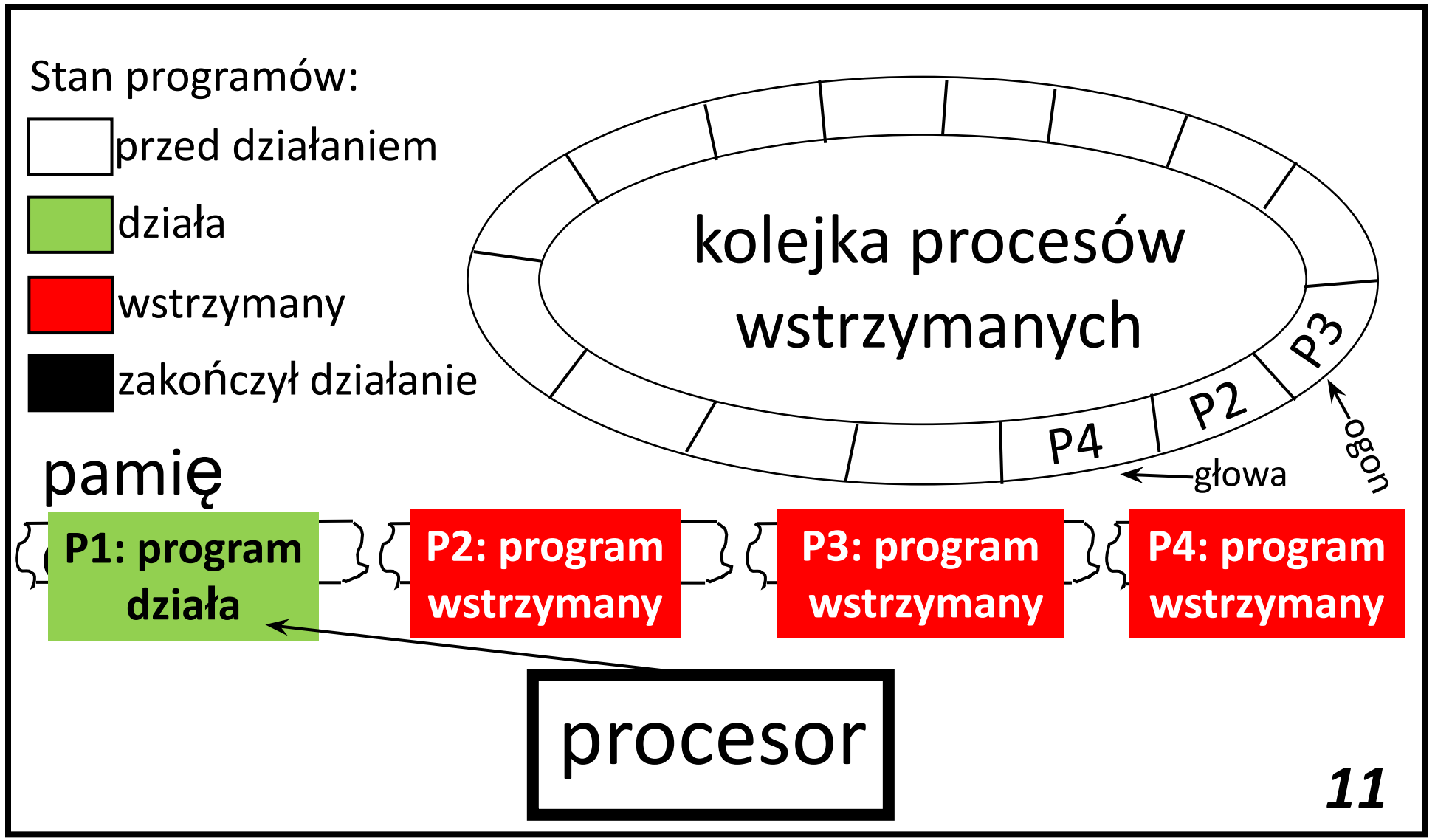
P1: program wstrzymany

P2: program wstrzymany

P3: program działa

P4: program wstrzymany

procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
kolejka procesów wstrzymanych
P3
P2
P4
ogon
głowa
pamię
P1: program działa
P2: program wstrzymany
P3: program wstrzymany
P4: program wstrzymany
procesor

Stan programów:

przed działaniem

działa

wstrzymany

zakończył działanie

pamię

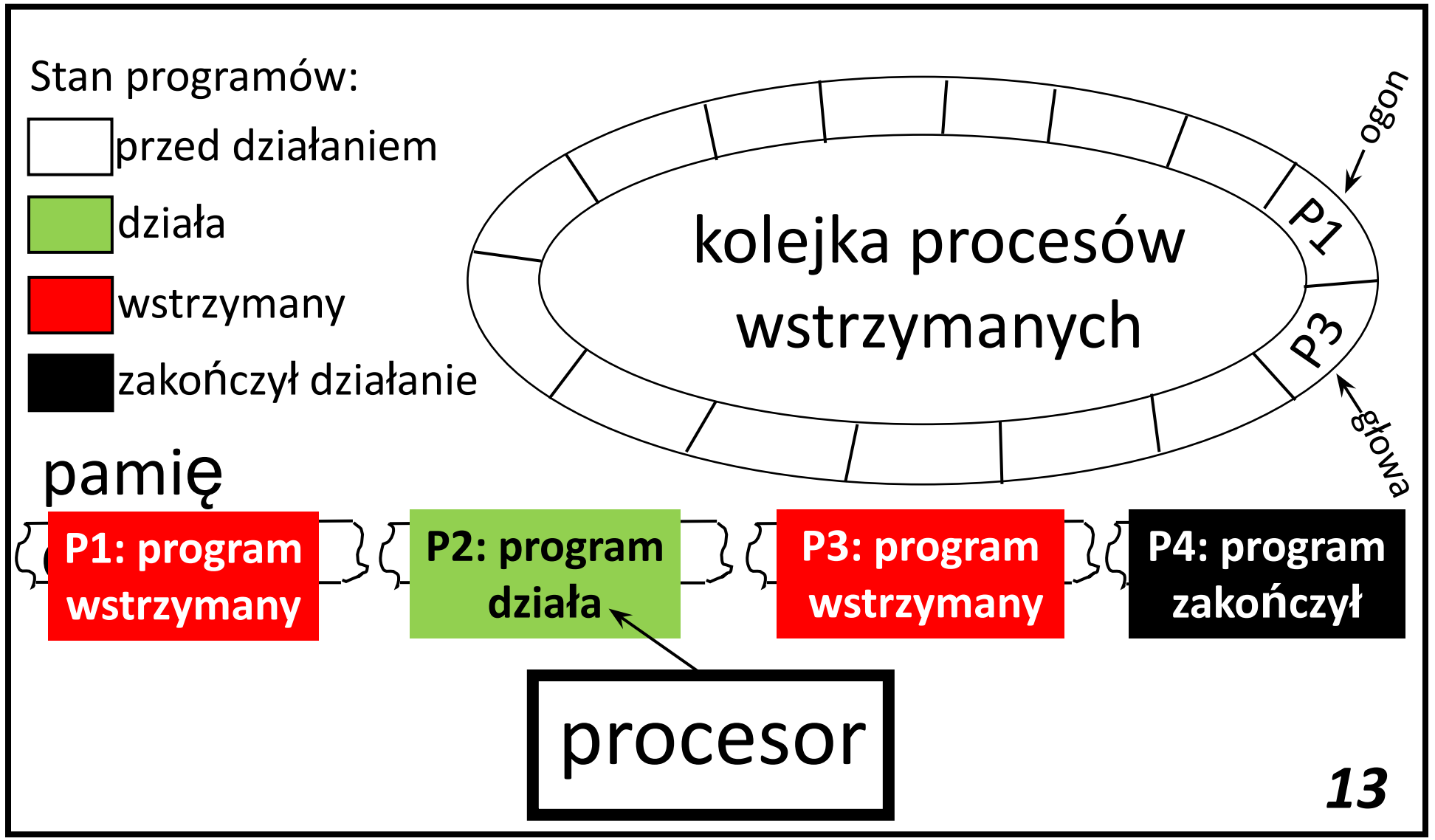
P1: program wstrzymany

P2: program wstrzymany

P3: program wstrzymany

P4: program działa

procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
kolejka procesów wstrzymanych
ogon
głowa
P1
P3
pamię
P1: program wstrzymany
P2: program działa
P3: program wstrzymany
P4: program zakończył
procesor

Stan programów:

przed działaniem

działa

wstrzymany

zakończył działanie

pamię

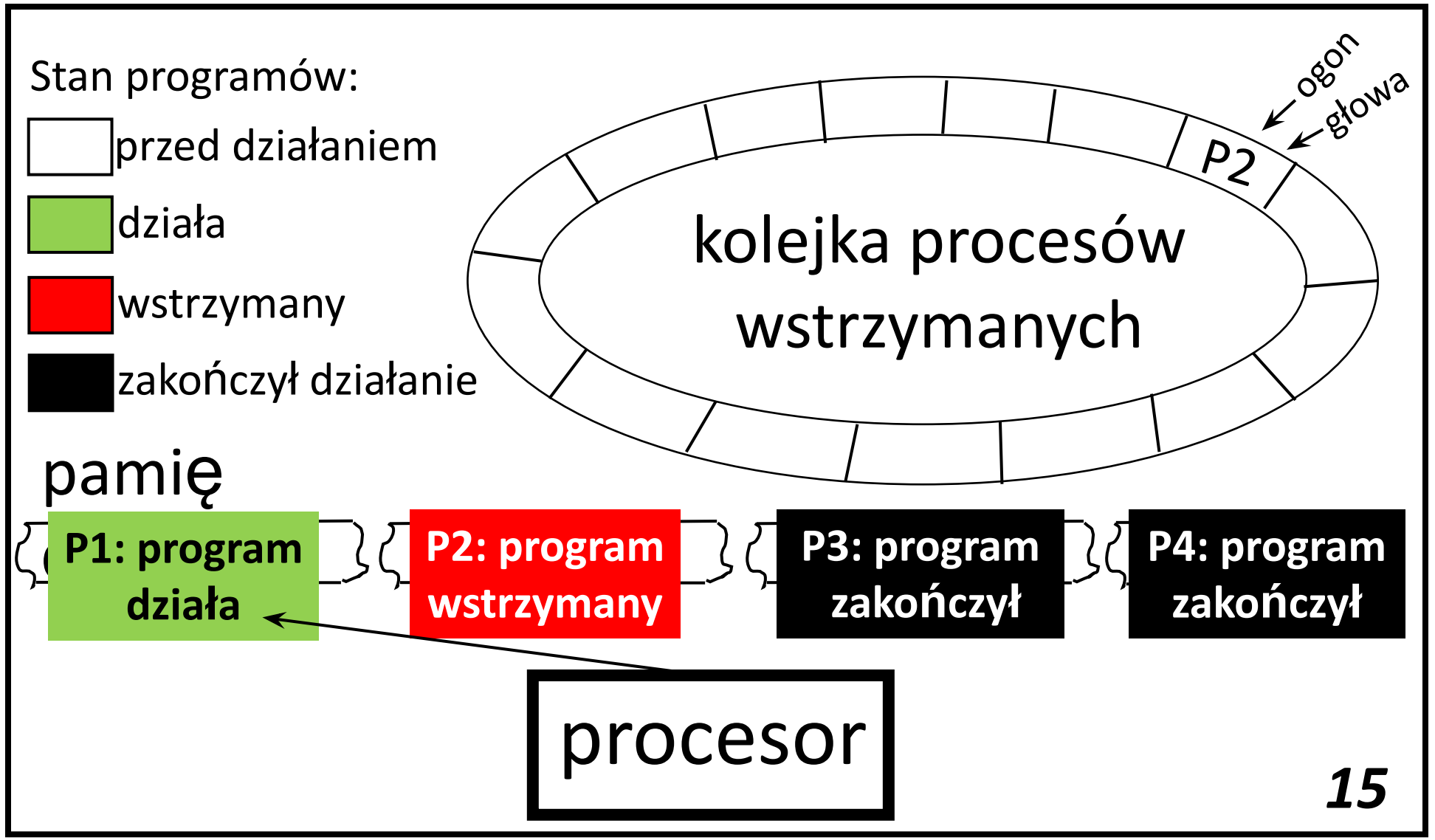
P1: program wstrzymany

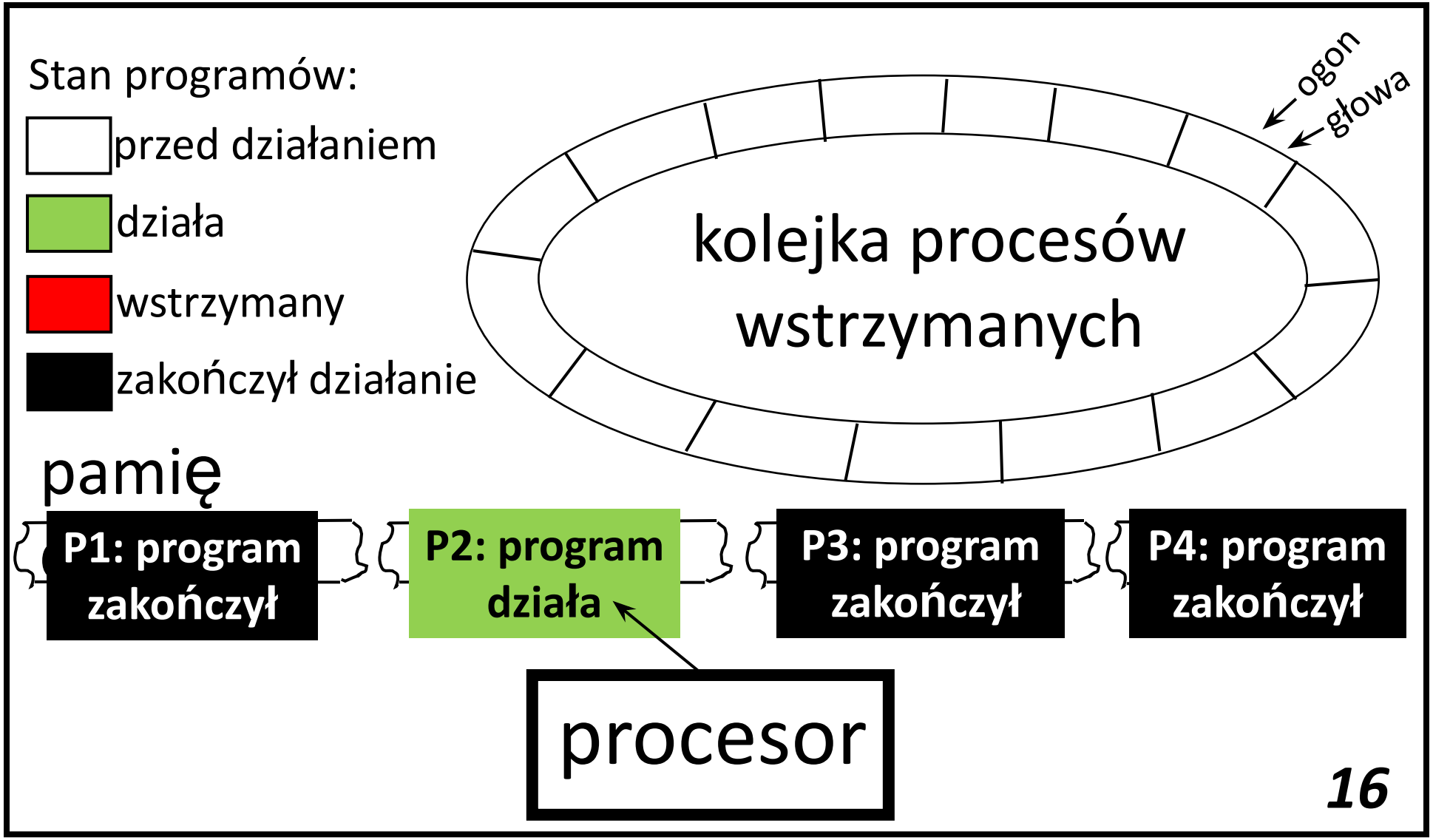
P2: program wstrzymany

P3: program działa

P4: program zakończył

procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
ogon
głowa
P2
kolejka procesów wstrzymanych
pamię
P1: program działa
P2: program wstrzymany
P3: program zakończył
P4: program zakończył
procesor


Stan programów:
przed działaniem
działa
wstrzymany
zakończył działanie
ogon
głowa
kolejka procesów wstrzymanych
pamię
P1: program zakończył
P2: program działa
P3: program zakończył
P4: program zakończył
procesor

Stan programów:

przed działaniem

działa

wstrzymany

zakończył działanie

pamię

P1: program zakończył

P2: program zakończył

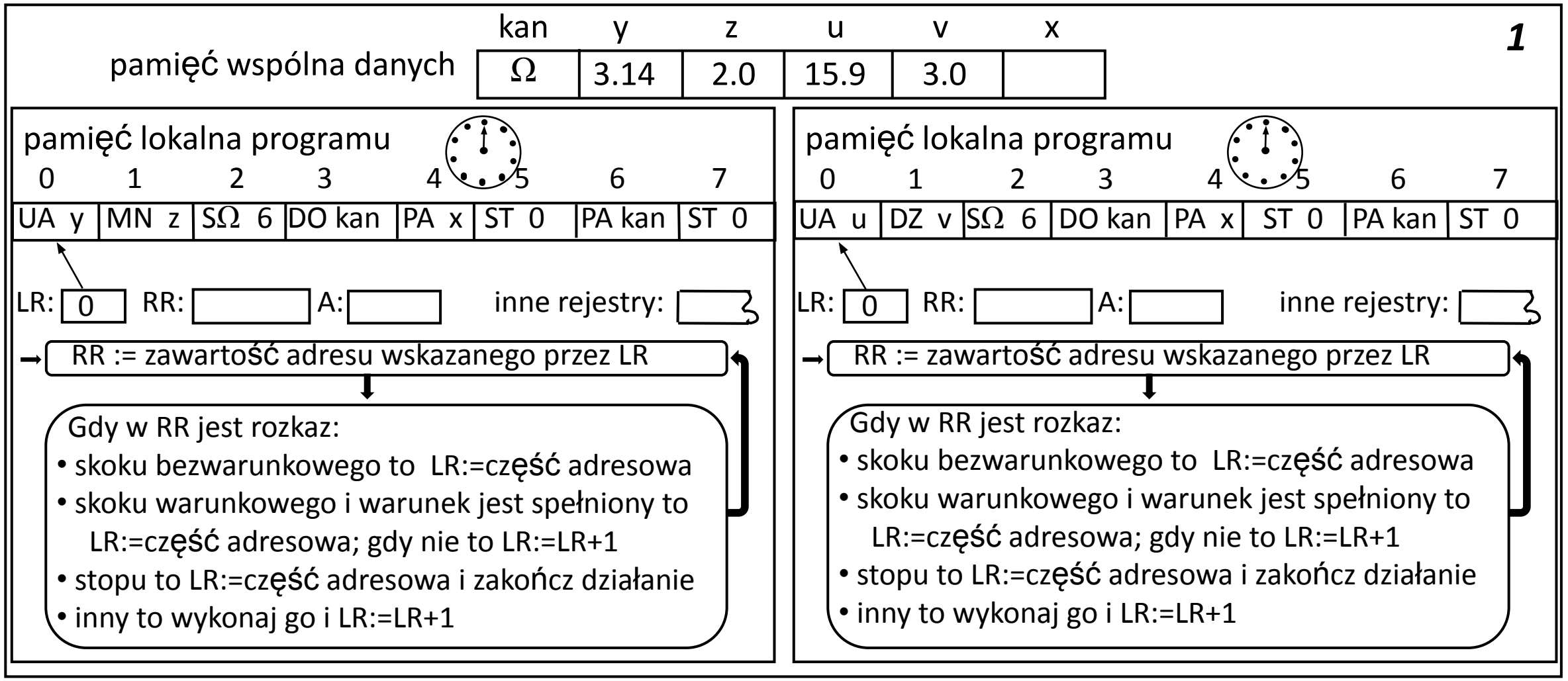
P3: program zakończył

P4: program zakończył

procesor

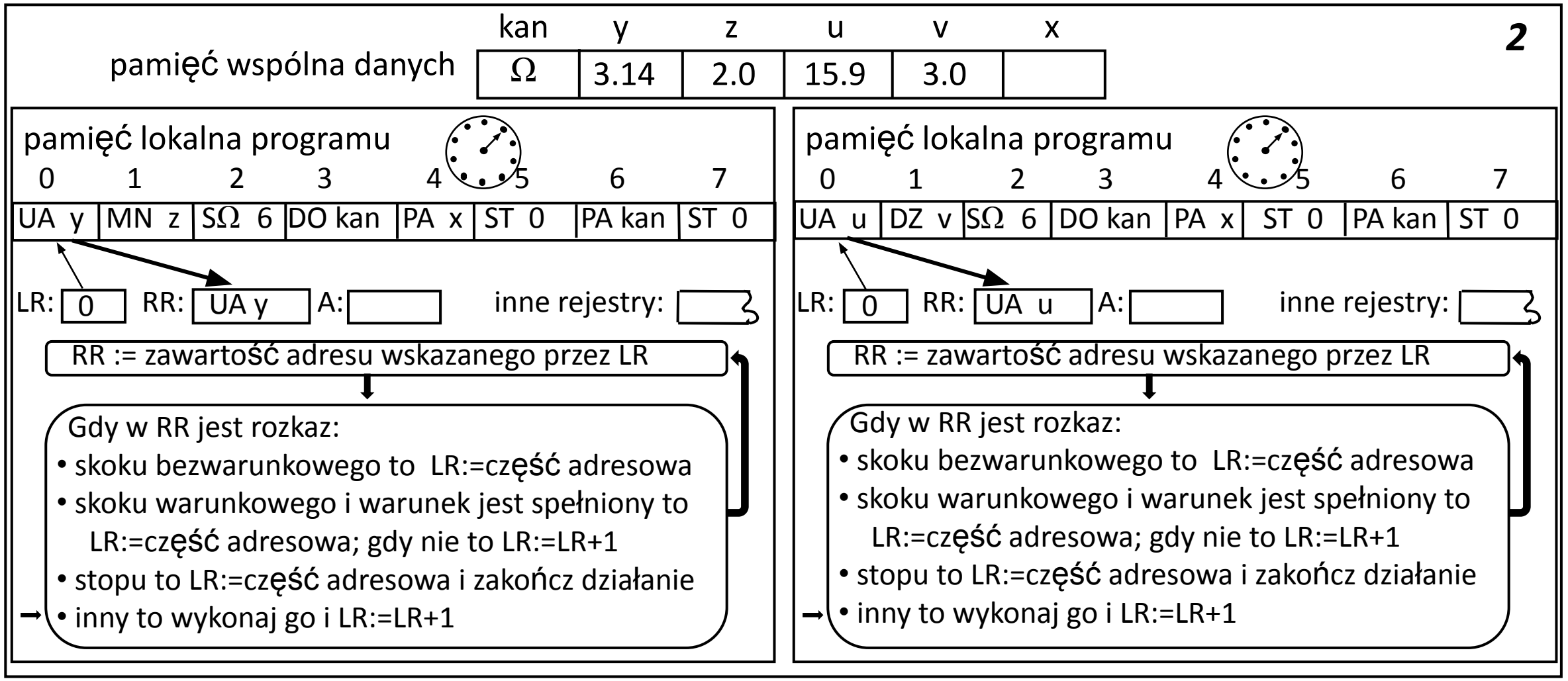
Współbieżne wykonywanie programów w języku maszyny

Jeżeli dopuszczamy istnienie wielu procesów w tym samym czasie, realizowanych w systemie jednoprocesorowym z podziałem czasu lub fizycznie wieloprocesorowym, to napotykamy problem konfliktu między procesami korzystającymi ze wspólnych zasobów. Przykład:
Animowana symulacja procesu wykonywania niepoprawnie zaprogramowanej instrukcji $x := y*z + u/v$
w systemie dwukomputerowym ze wspólną pamięcią danych. Komputery komunikują się przez wspólny zasób - komórkę *kan* (kanał komunikacyjny): komputer 1 posyła do niej wartość $y*z$, a komputer 2 - wartość u/v. Konflikt obydwu transmisji danych następuje gdy komputery dokonują ich w tej samej chwili, lub w chwilach bardzo bliskich. Nie wiadomo wtedy jaka wartość znajdzie się w kanale: mamy tu do czynienia z niepożądanym niedeterminizmem.



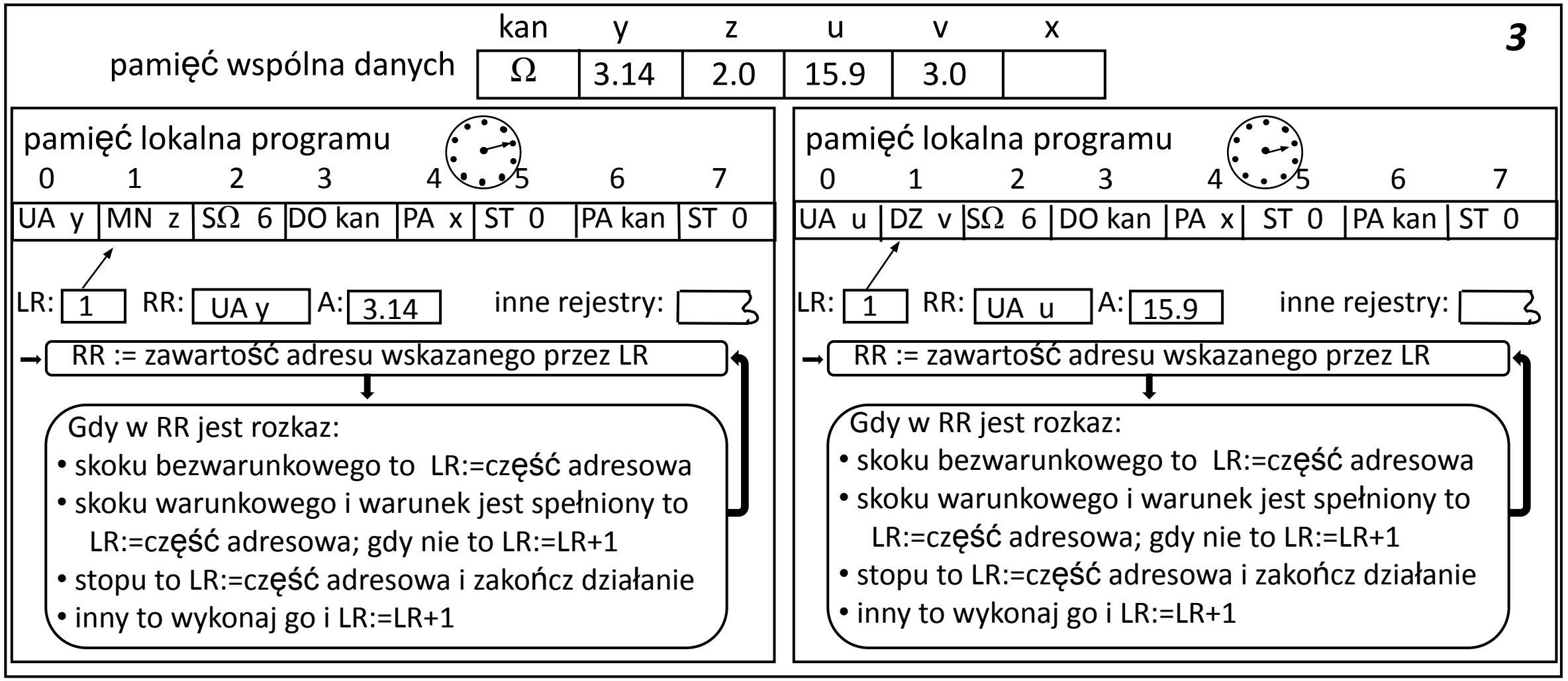
komputer 1

komputer 2



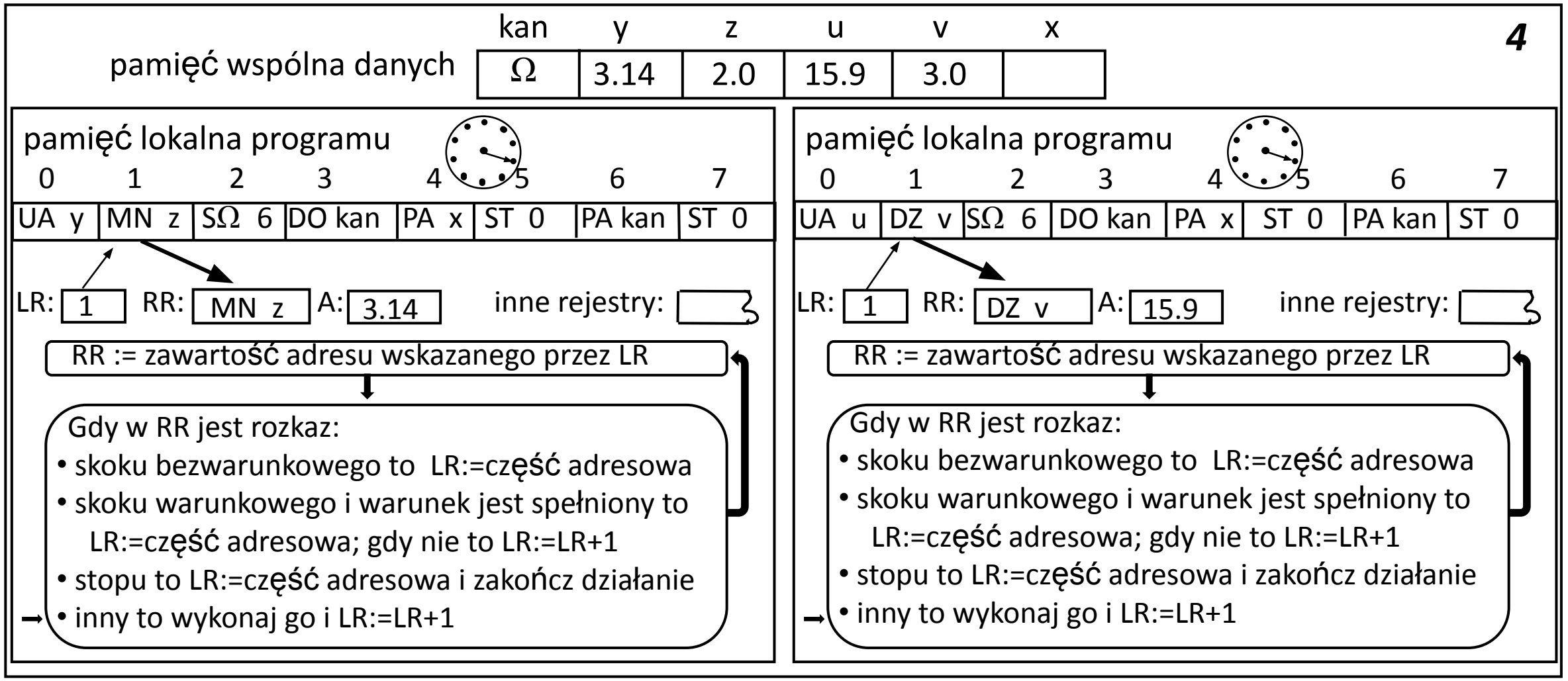
komputer 1

komputer 2



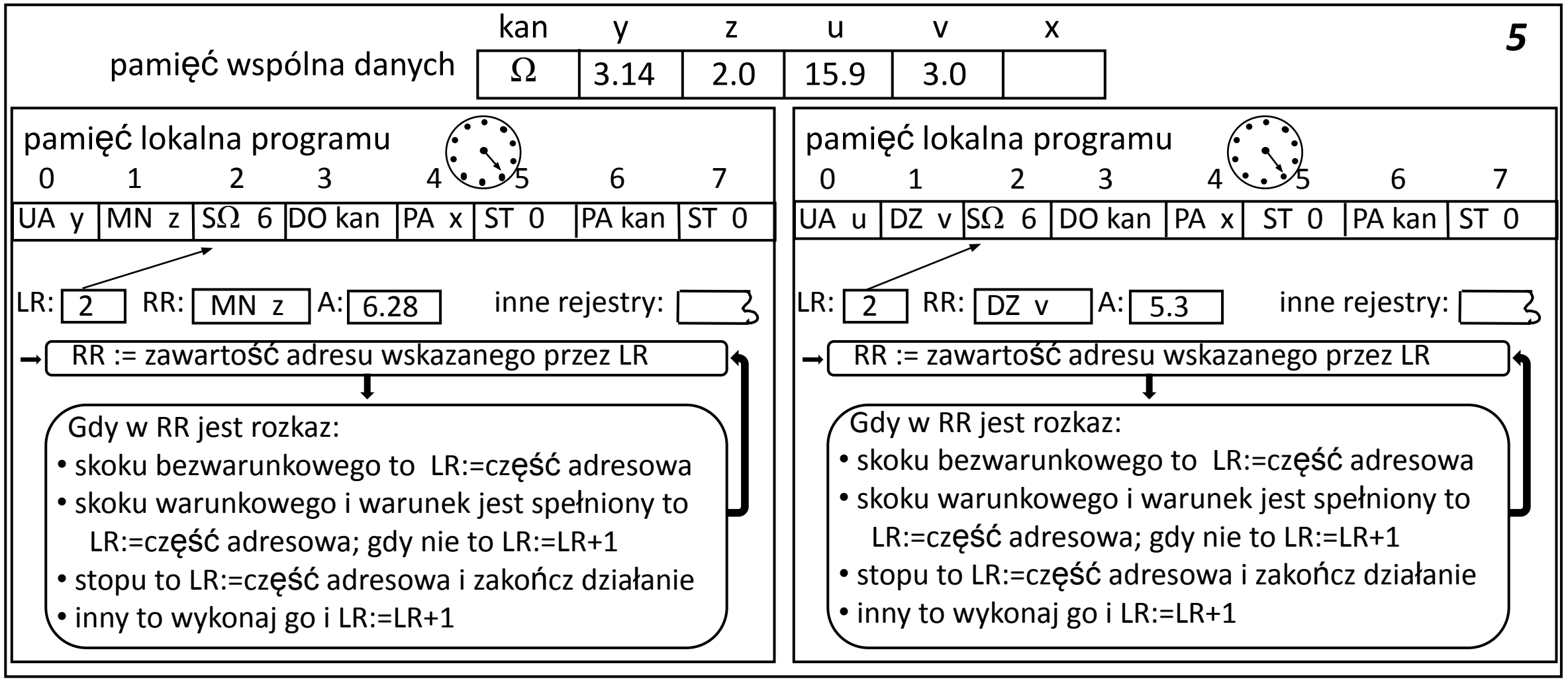
komputer 1

komputer 2



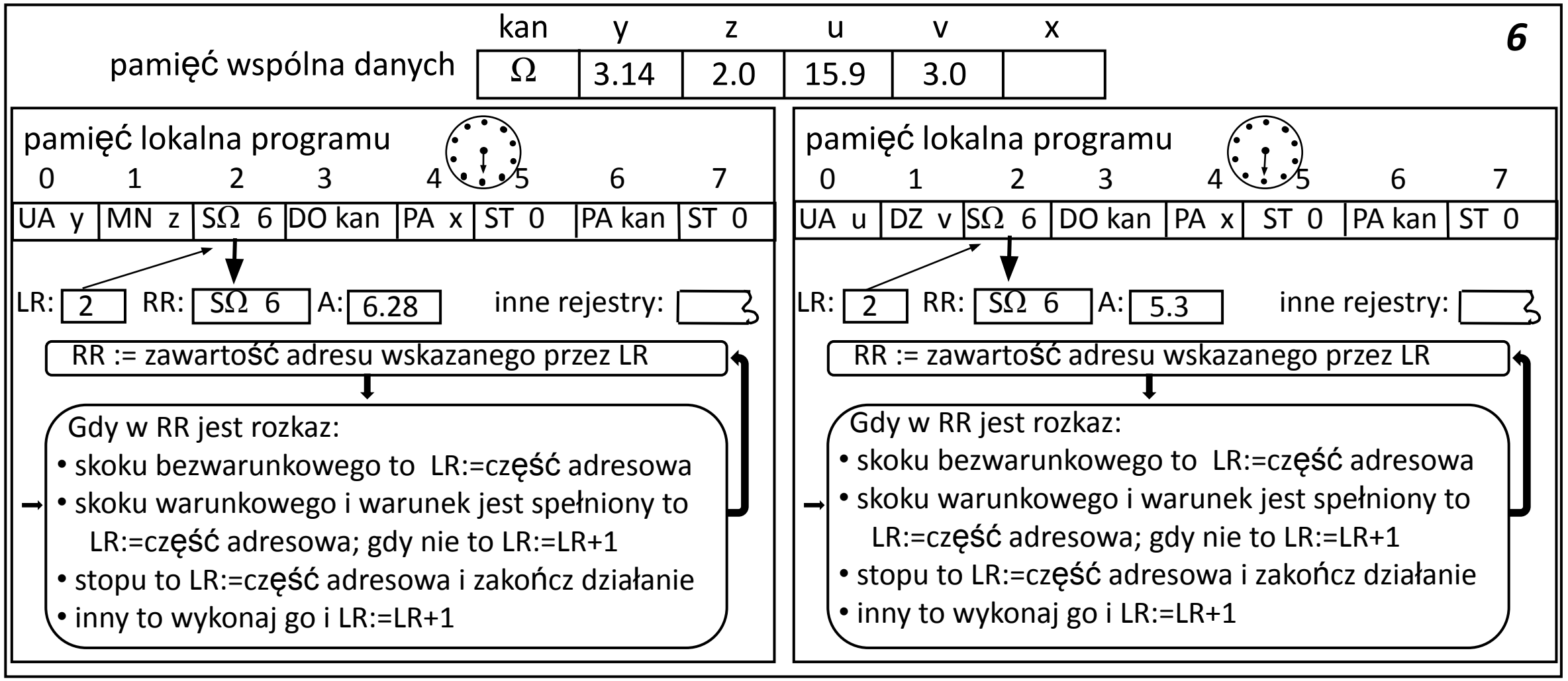
komputer 1

komputer 2



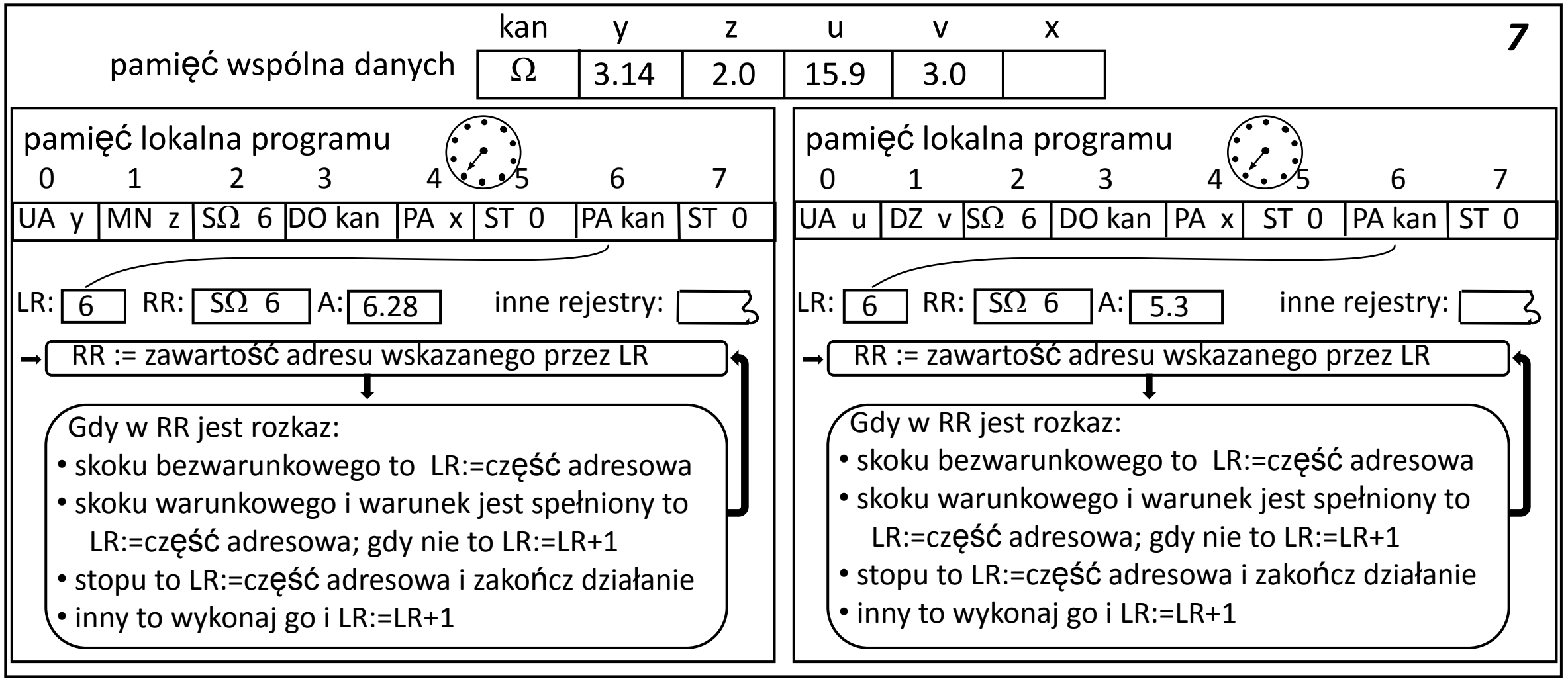
komputer 1

komputer 2



komputer 1

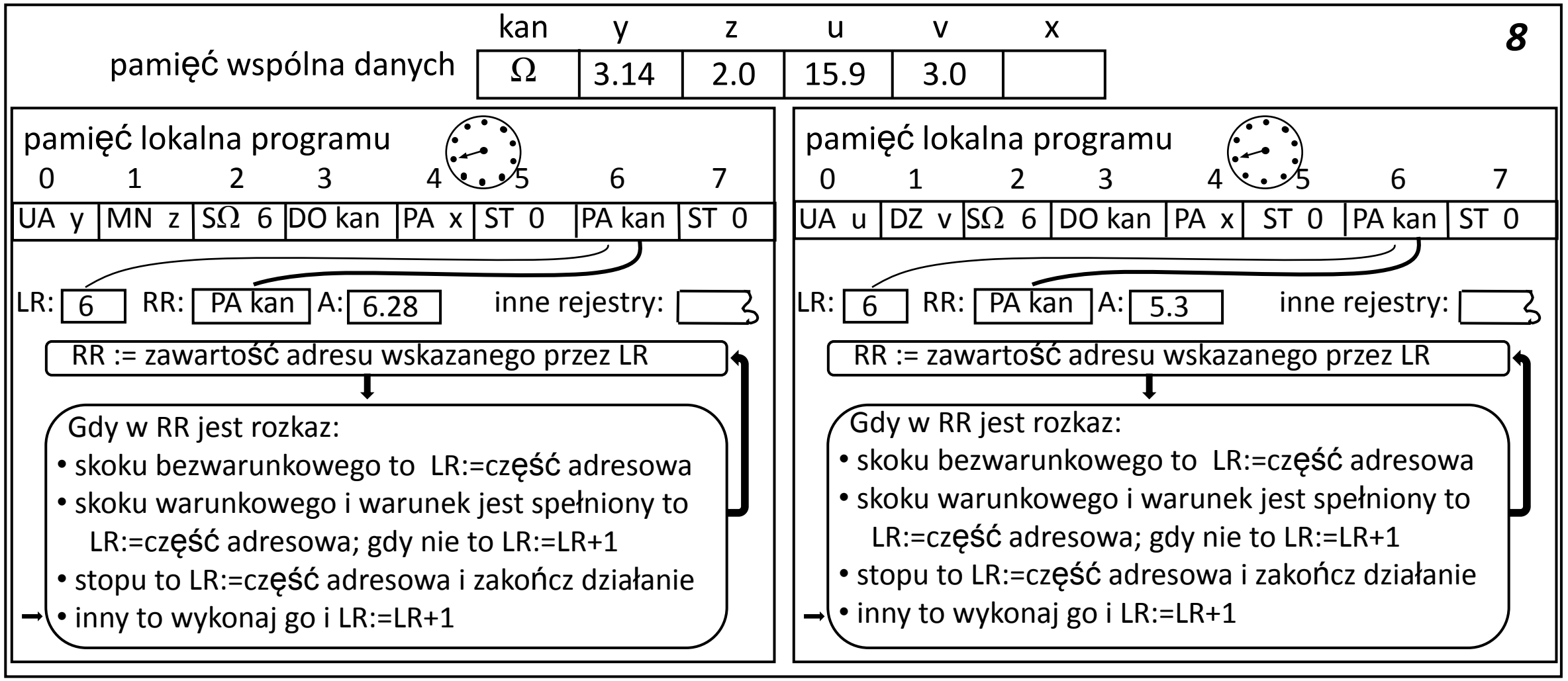
komputer 2



komputer 1

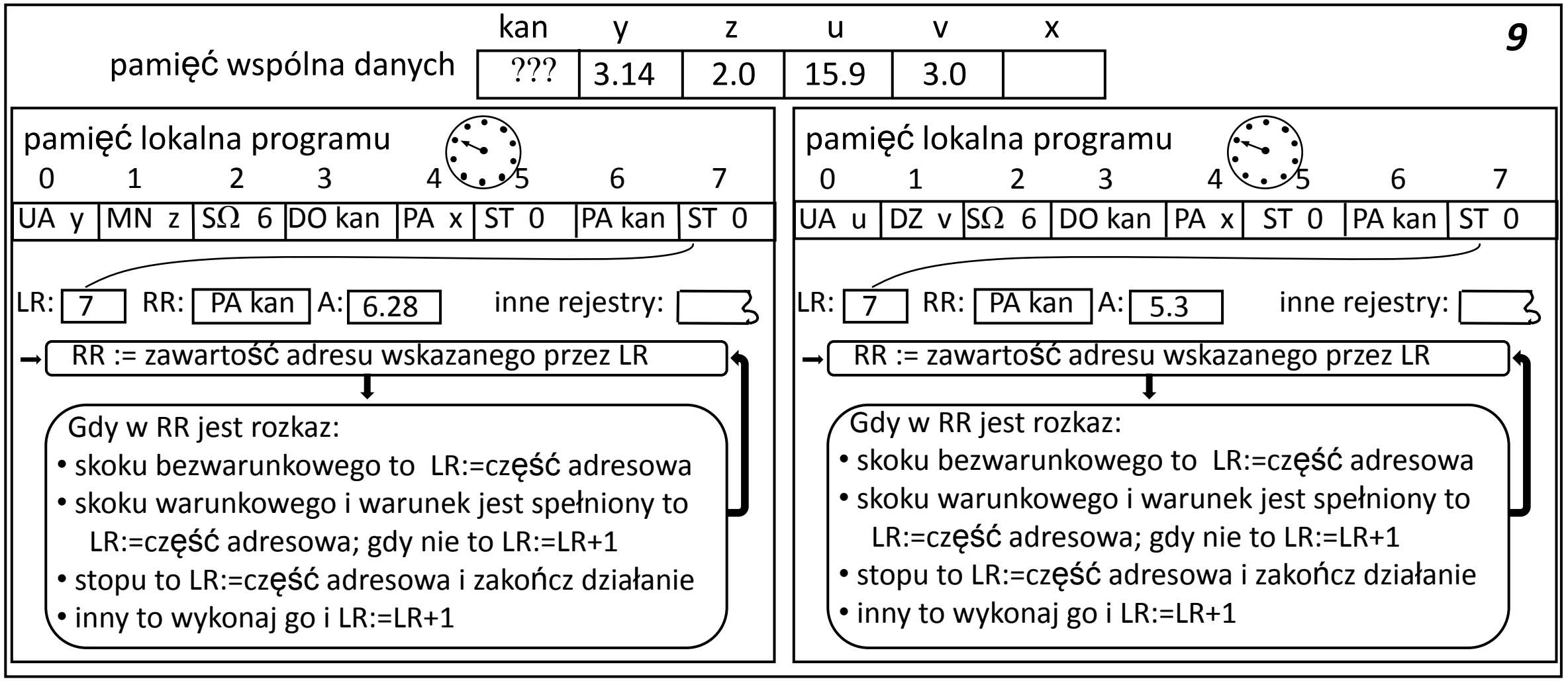
BŁĄD! Obydwa procesory jednocześnie widziały w kanale symbol Ω

komputer 2



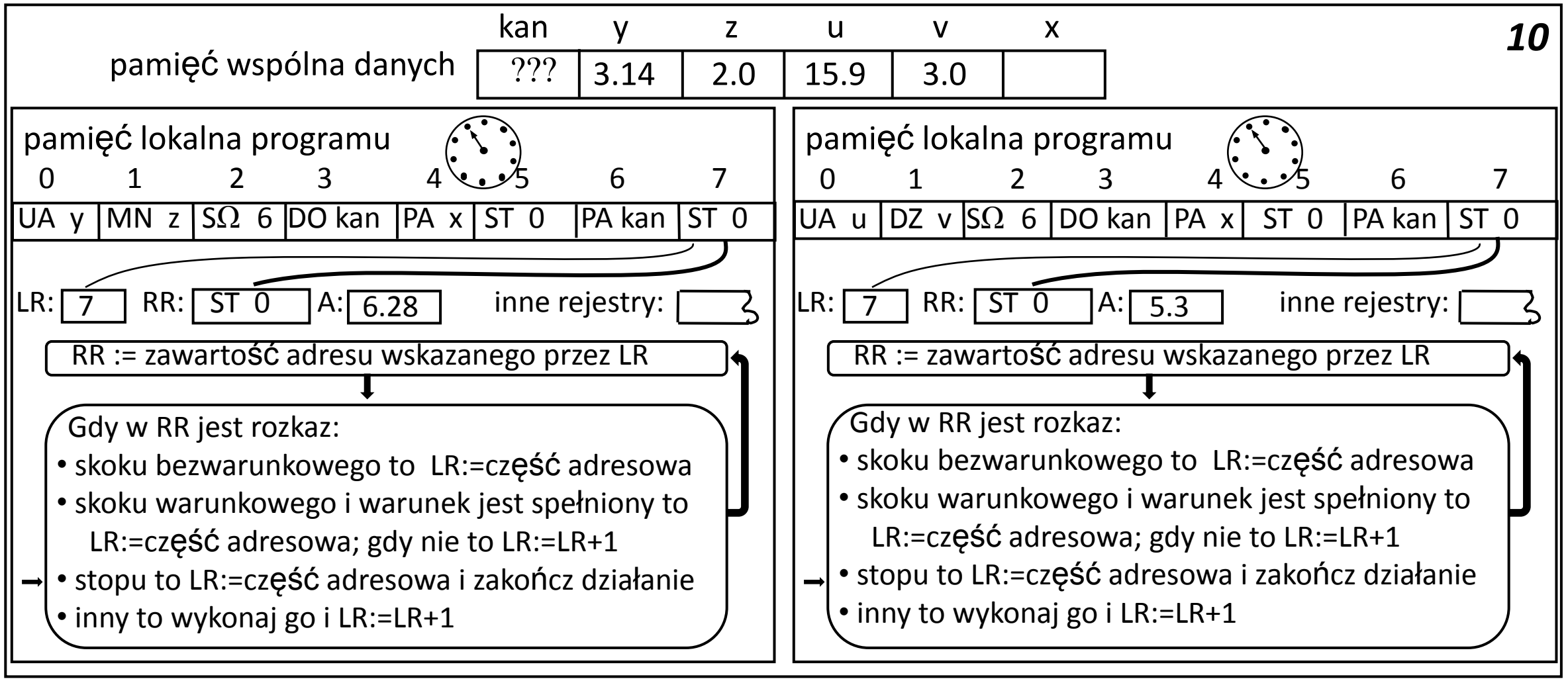
komputer 1

komputer 2



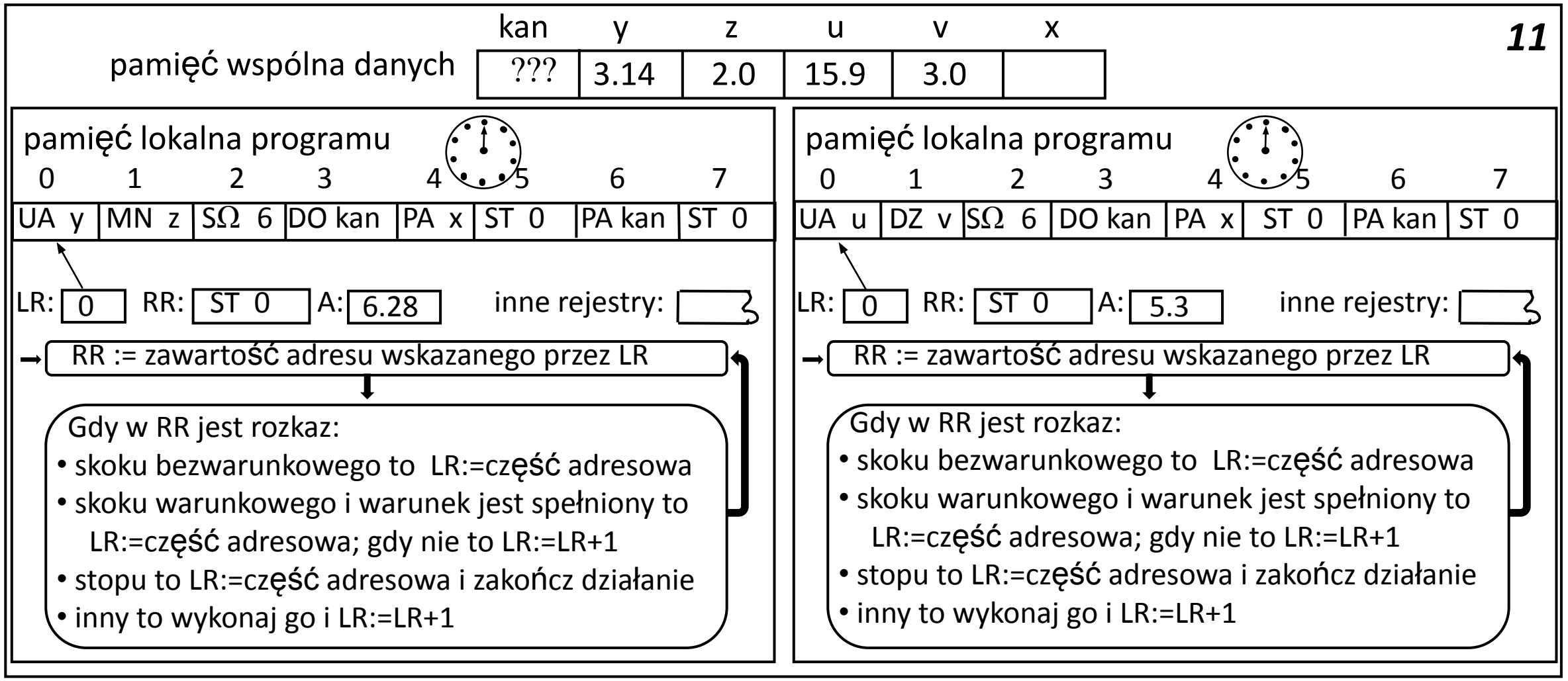
komputer 1

komputer 2



komputer 1

komputer 2



komputer 1

komputer 2

Po wykonaniu instrukcji $x := y*z+u/v$ zmienna x nie otrzymała

wartości wyrażenia $y*z+u/v$

(zawartość komórki x jest taka jak przed wykonaniem!)

Jak zaprogramować poprawnie? Trzeba synchronizować procesy!
Można zastosować mechanizm semaforów (tu binarnych) – gdy komputery (procesy) mają dostęp do wspólnej pamięci. Taki system nie jest więc prawdziwie rozproszonym!

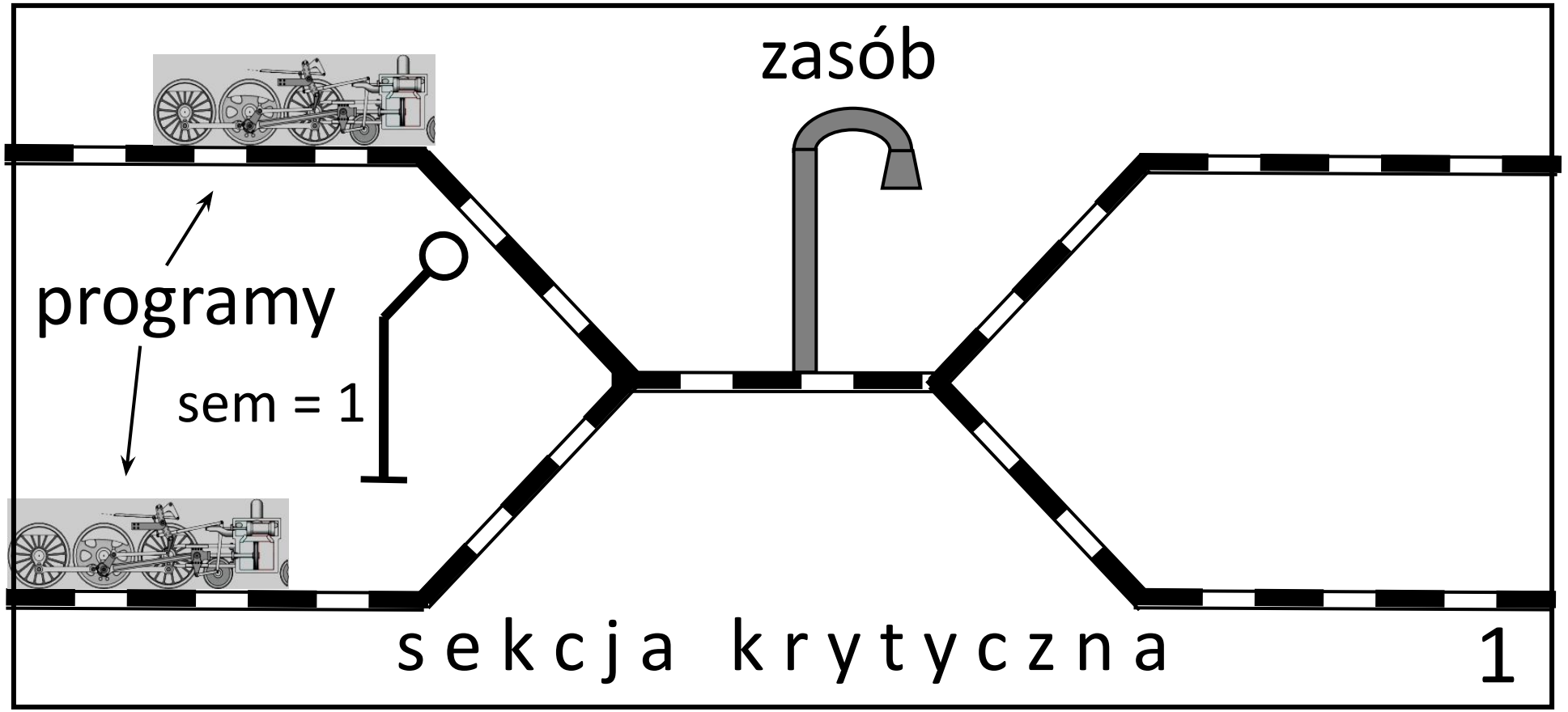
Semafor (E.W. Dijkstra) binarny jest zmienną przyjmującą wartości 0 i 1, wspólną dla programów współdzielących chroniony zasób. Dopuszczalne są na niej dwie operacje nazywane zwykle P i V, o następującym znaczeniu, gdzie zmienna *sem* jest semaforem:

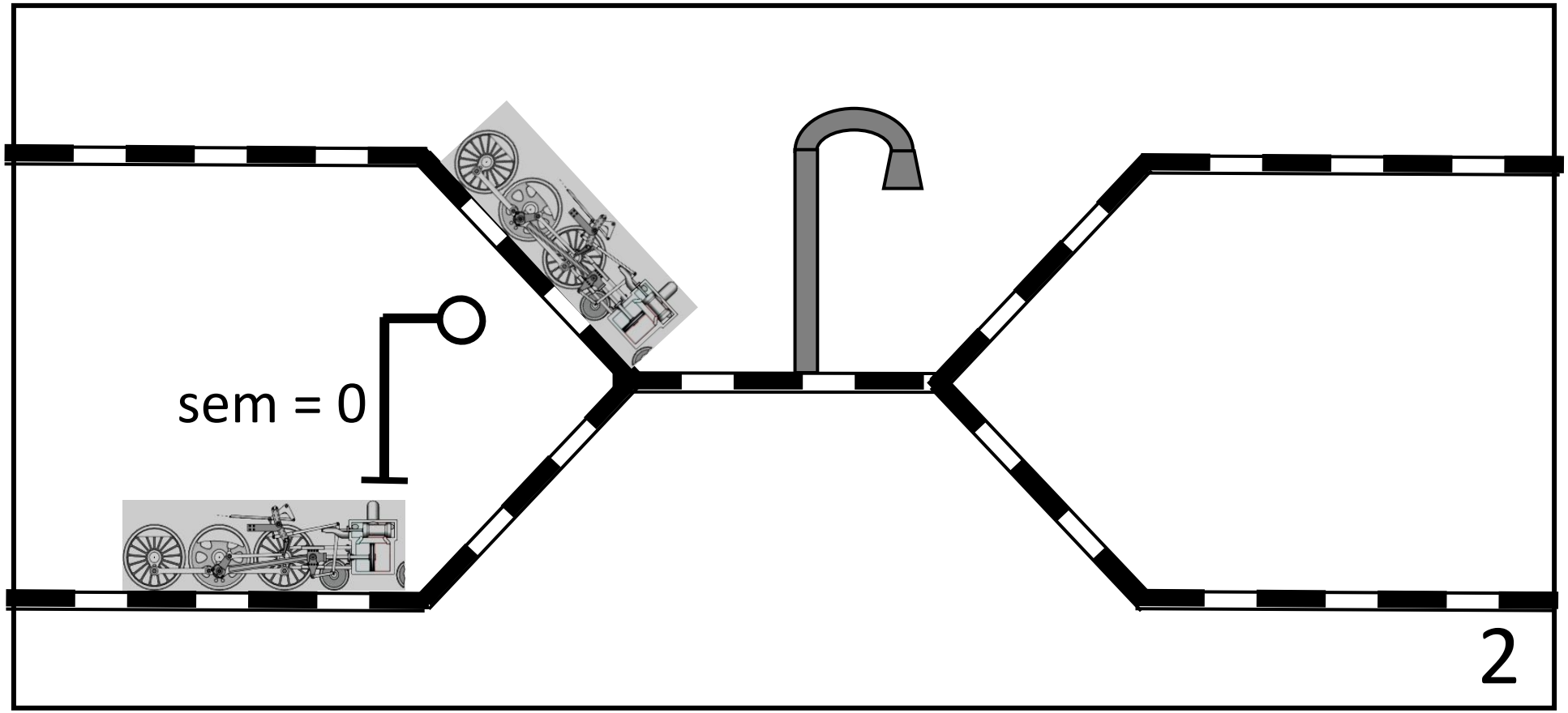
P(*sem*): gdy $sem > 0$ to $sem := sem - 1$, gdy $sem = 0$ to wstrzymaj proces.

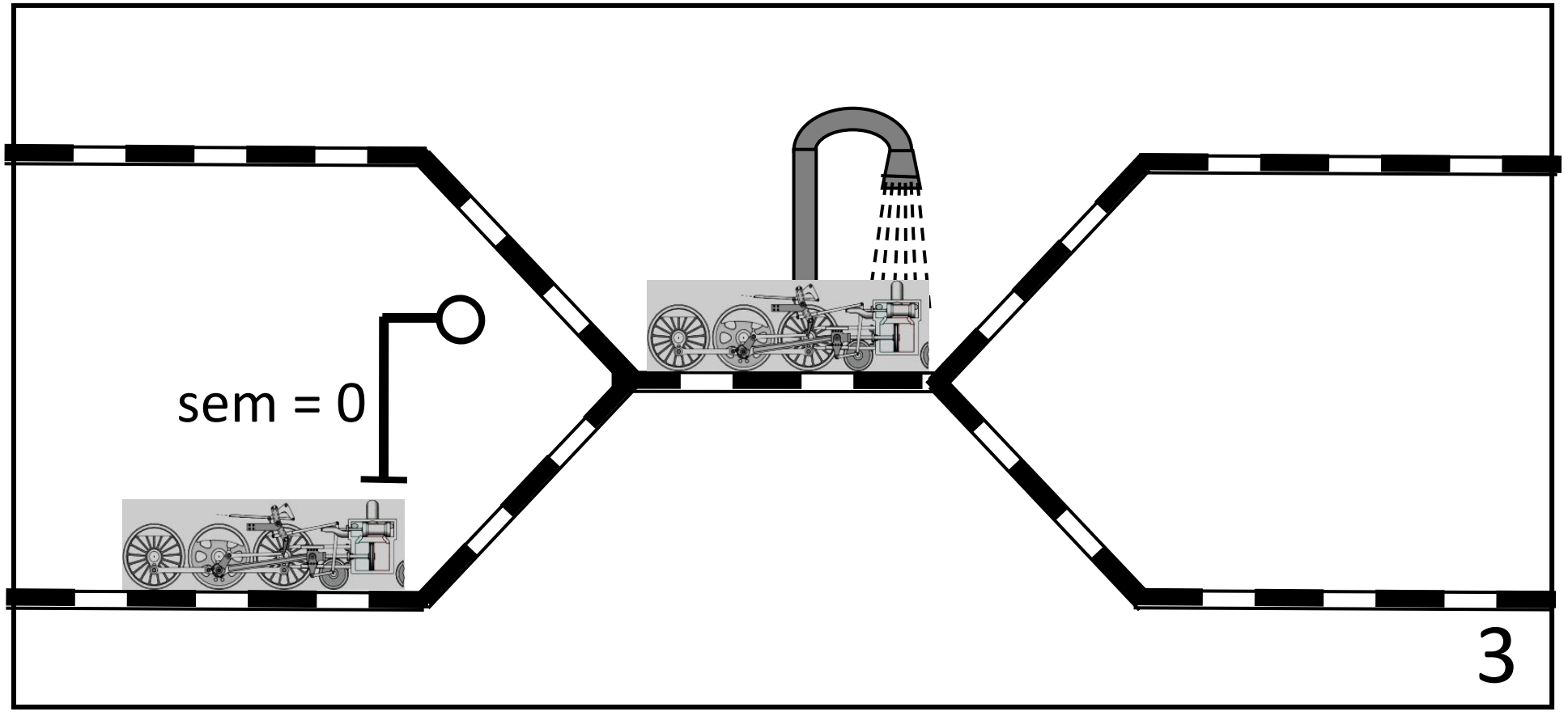
V(*sem*): $sem := 1$; wznów któryś z wstrzymanych procesów (np. wstrzymywany najdłużej).

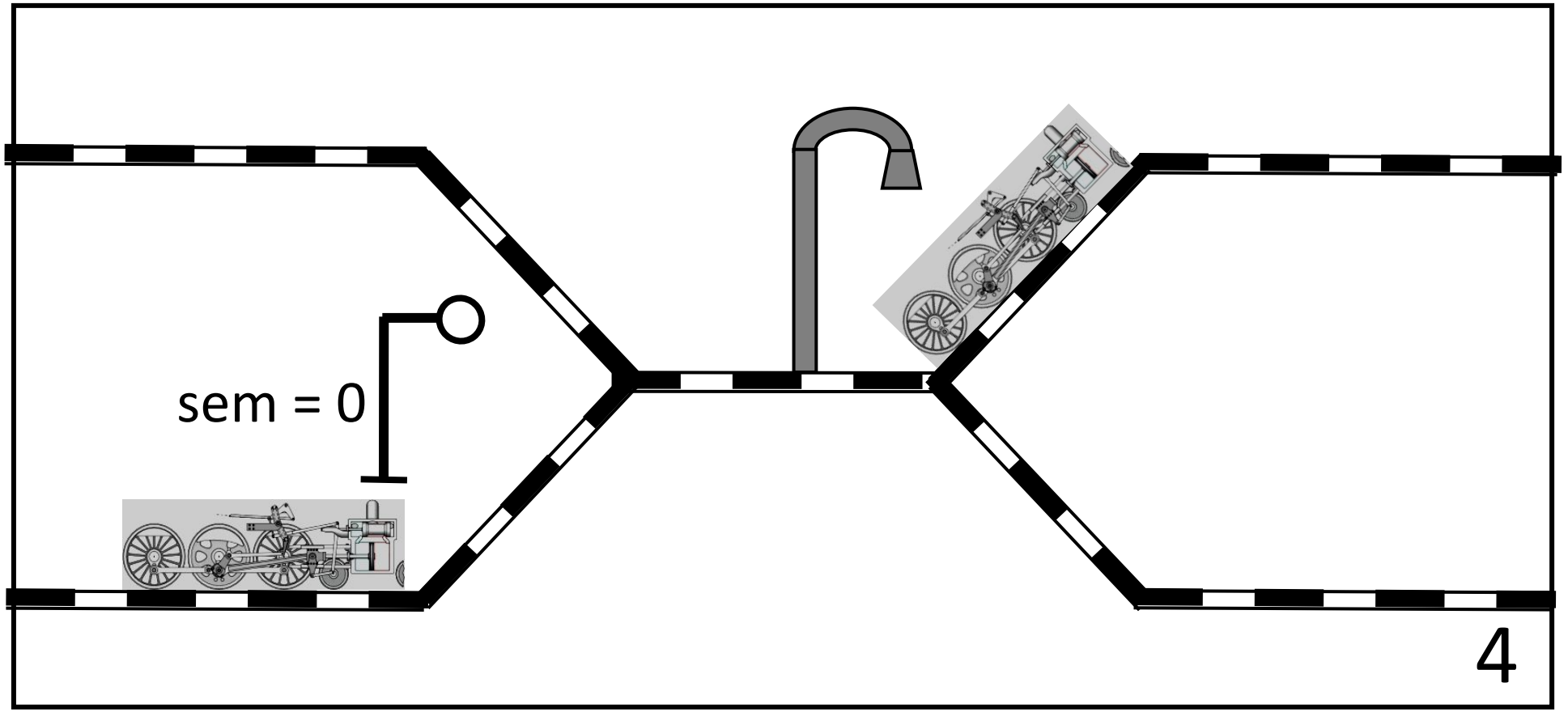
Fragment programu użytkownika między operacjami P(*sem*) i odpowiadajacym V(*sem*) to ***sekcja krytyczna*** chroniona semaforem *sem*.

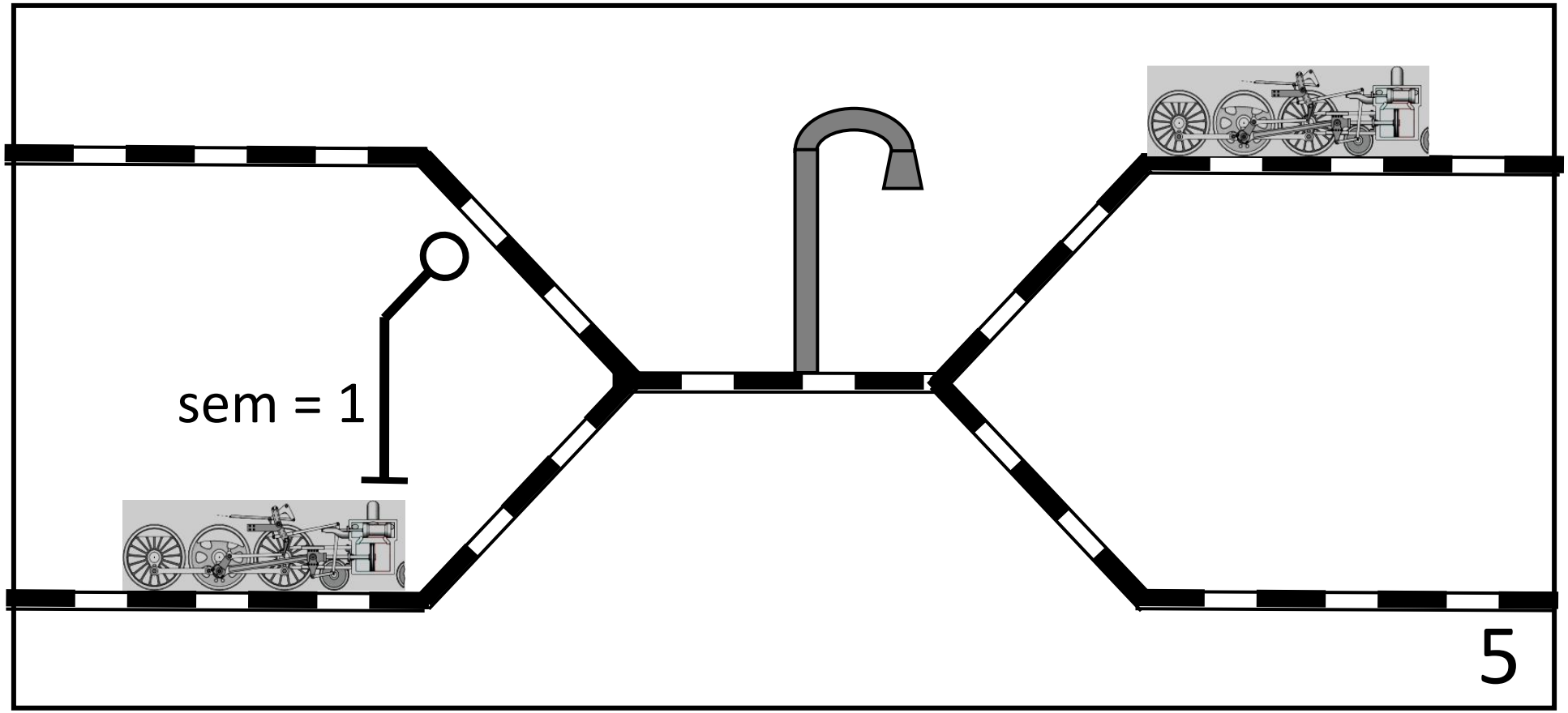
Operacje P, V (oznaczane niekiedy *wait*, *signal*) są niepodzielne (atomowe): gdy jakiś program wykonuje którąś, to inny nie może w tym czasie wykonać żadnej. Same są też sekcjami krytycznymi ale wykonywanymi na niższym poziomie niż programy użytkownika

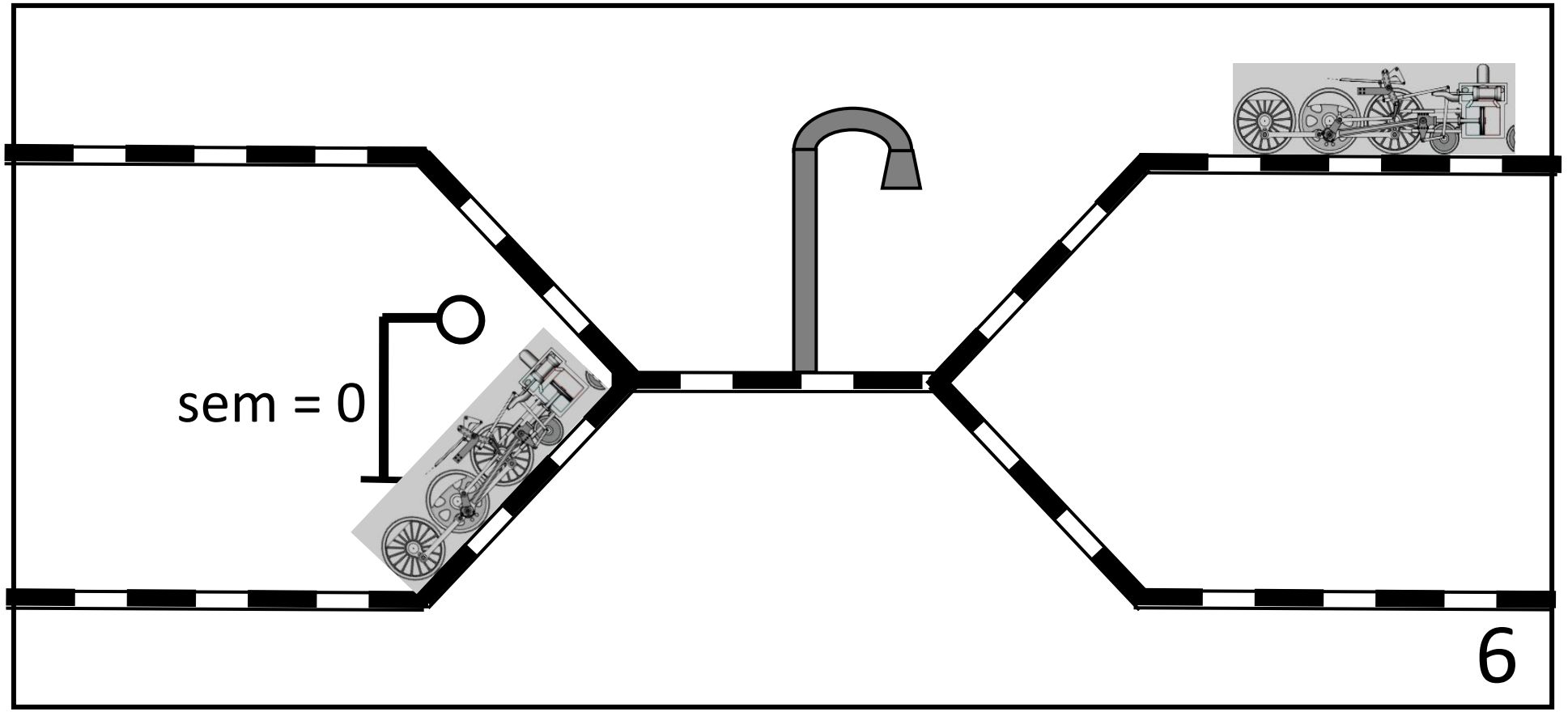

zasób
programy
sem = 1
s e k c j a k r y t y c z n a
1

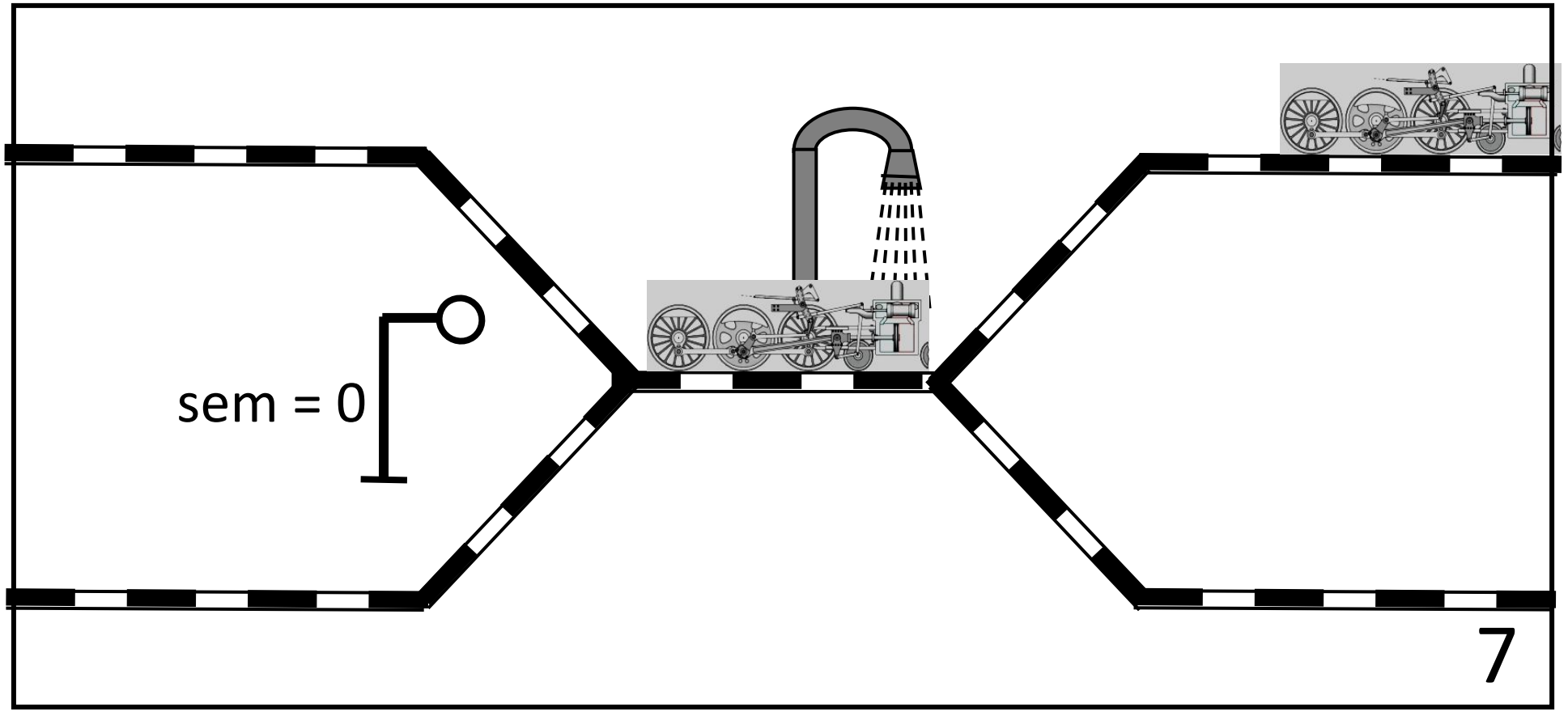

sem = 0
2

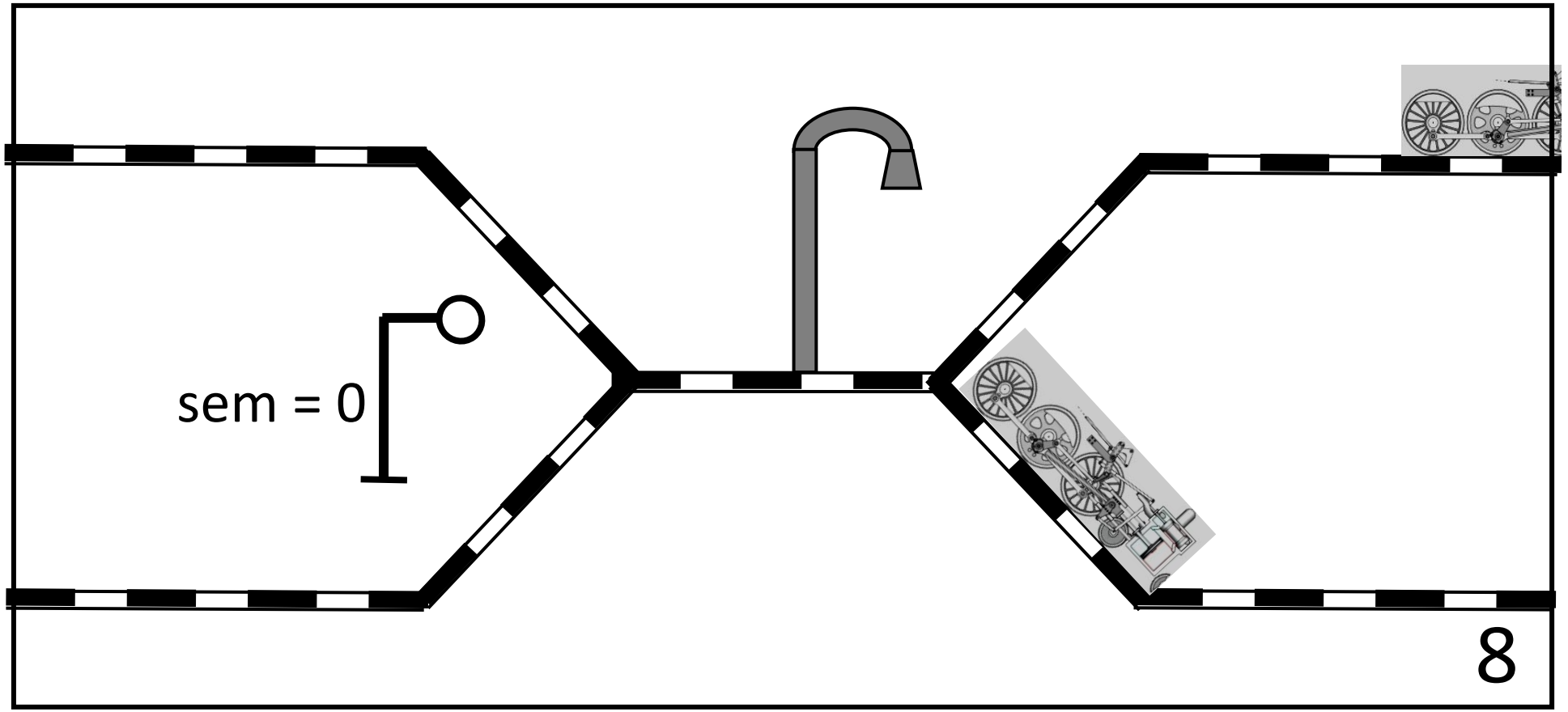

sem = 0
3

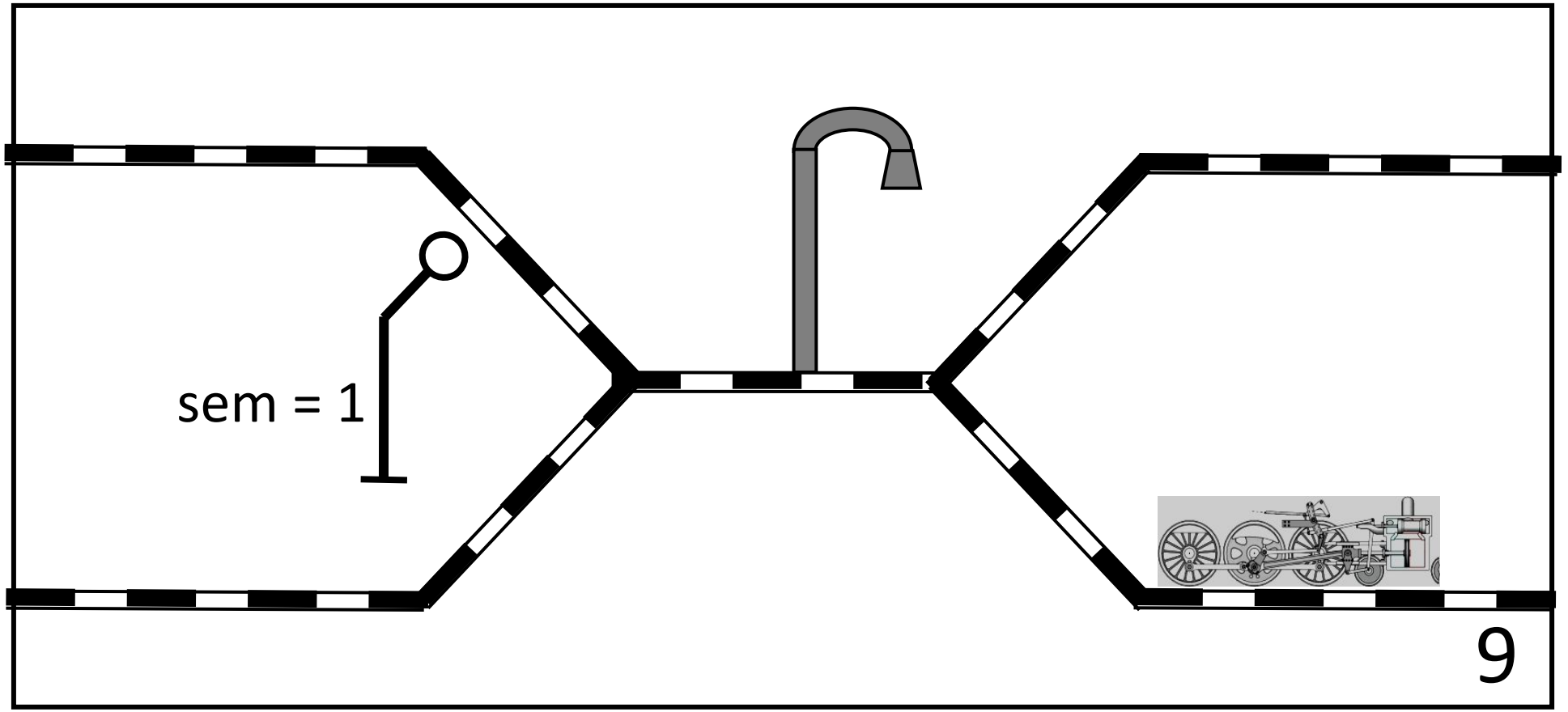

sem = 0
4

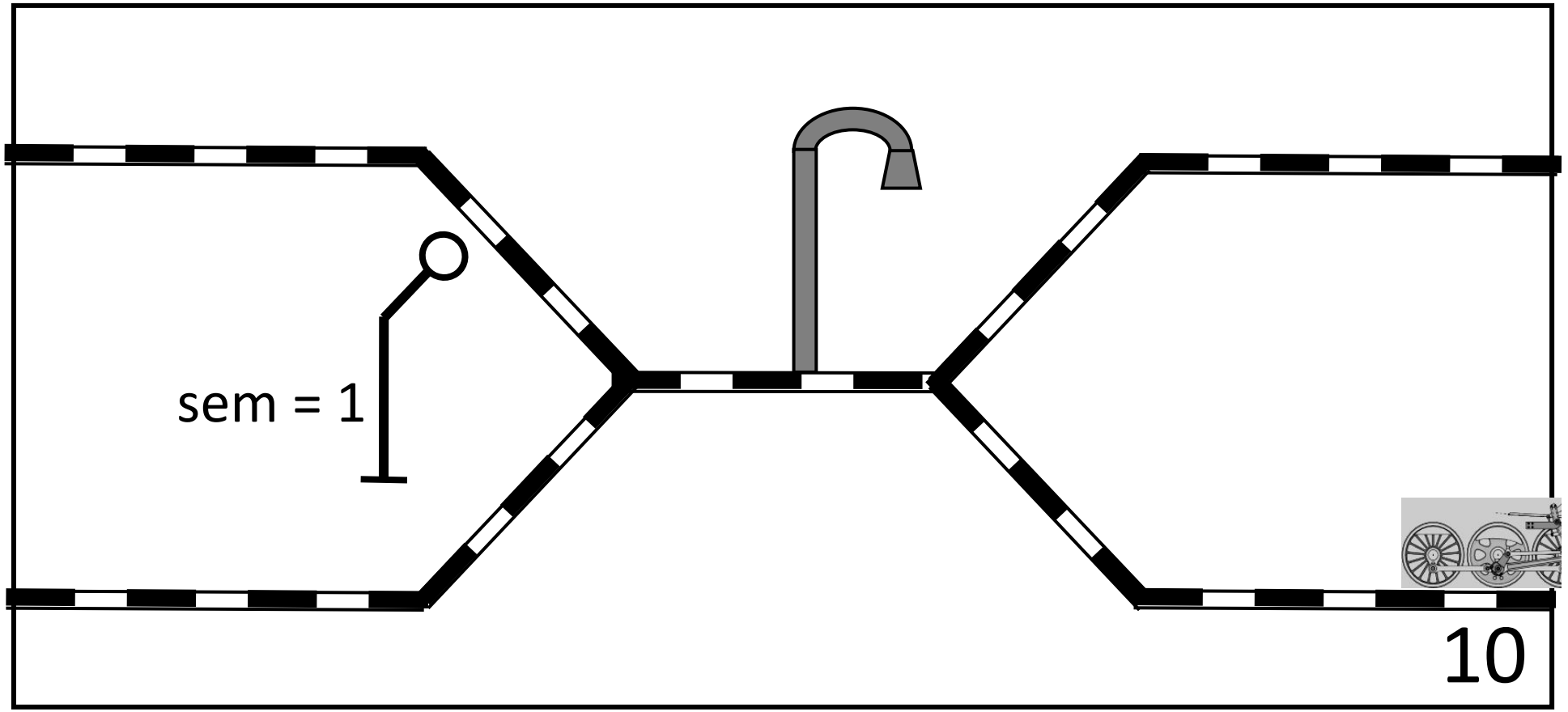

sem = 1
5


sem = 0
6


sem = 0
7


sem = 0
8


sem = 1
9

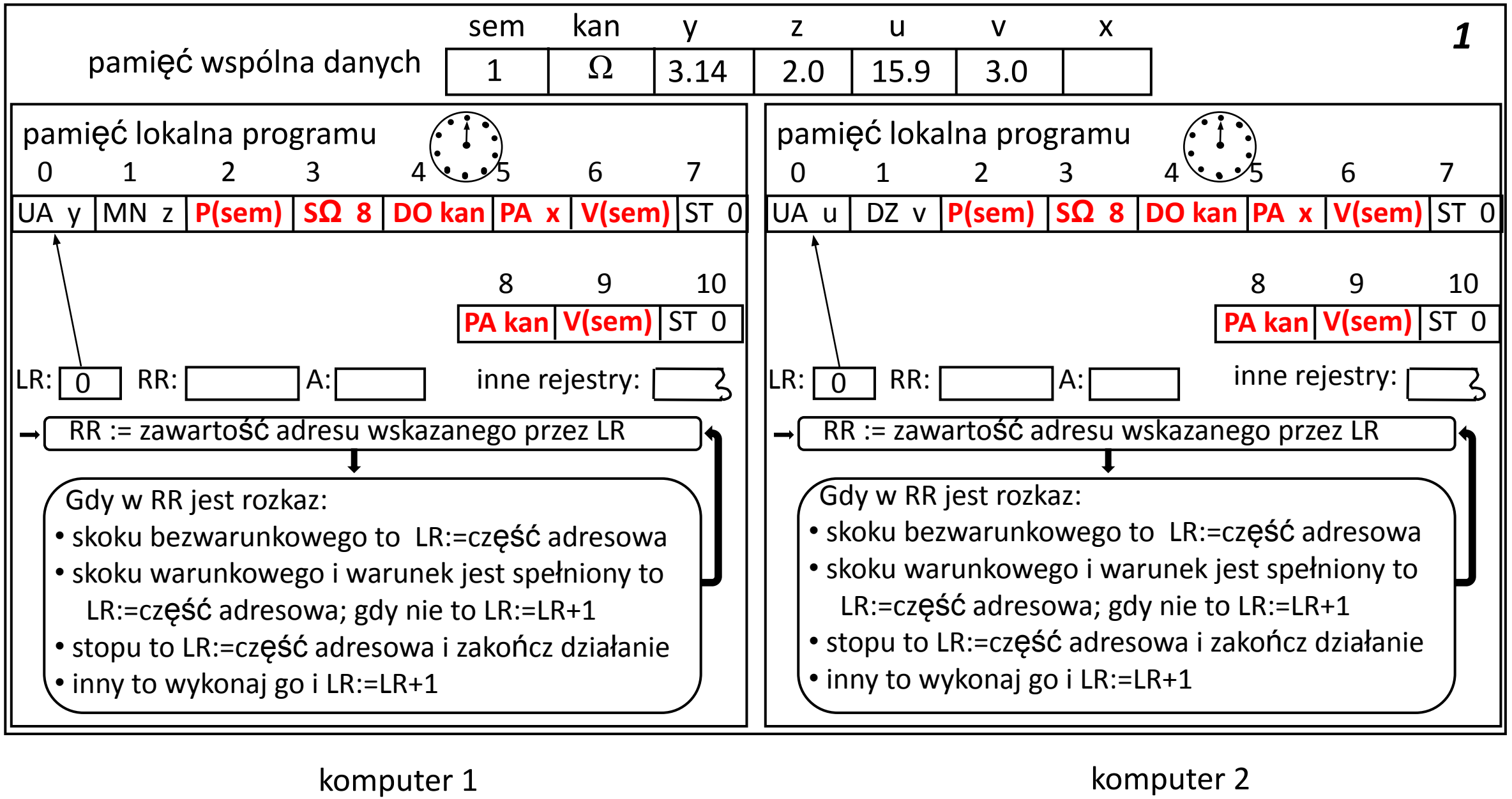

sem = 1
10

Symulacja wykonania programu na schemacie funkcjonalnym systemu 2-procesorowego z dostępem do wspólnej pamięci danych dla obydwu procesorów i z jednakową prędkością ich zegarów.
Synchronizacja semaforowa – wzajemne wykluczanie.
Sekcja krytyczna: rozkazy **czerwone**

Rozwiązanie poprawne!

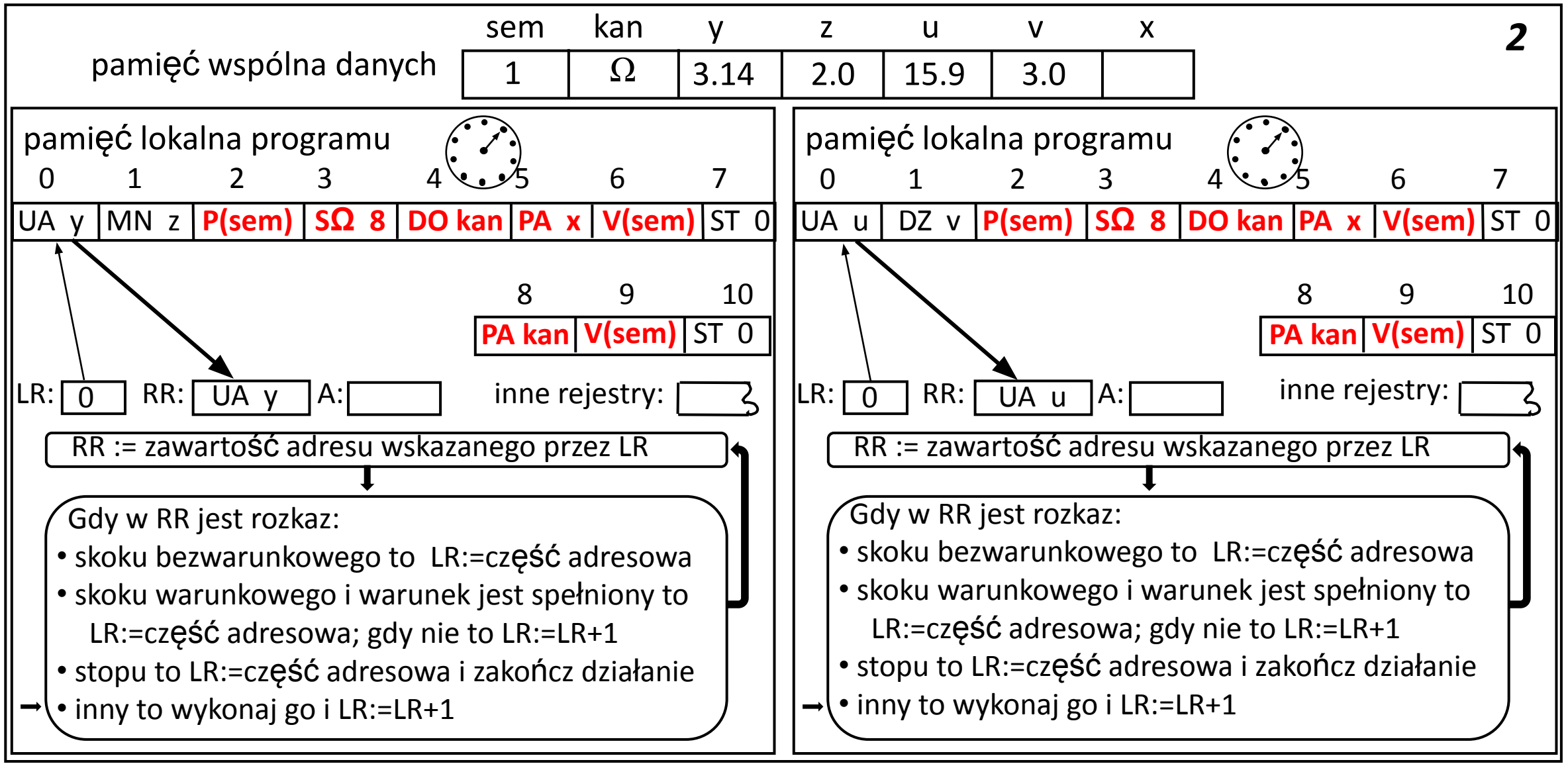
LR – licznik rozkazów RR – rejestr rozkazów A – akumulator

Przykład działania: obliczenie $x := y*z + u/v$



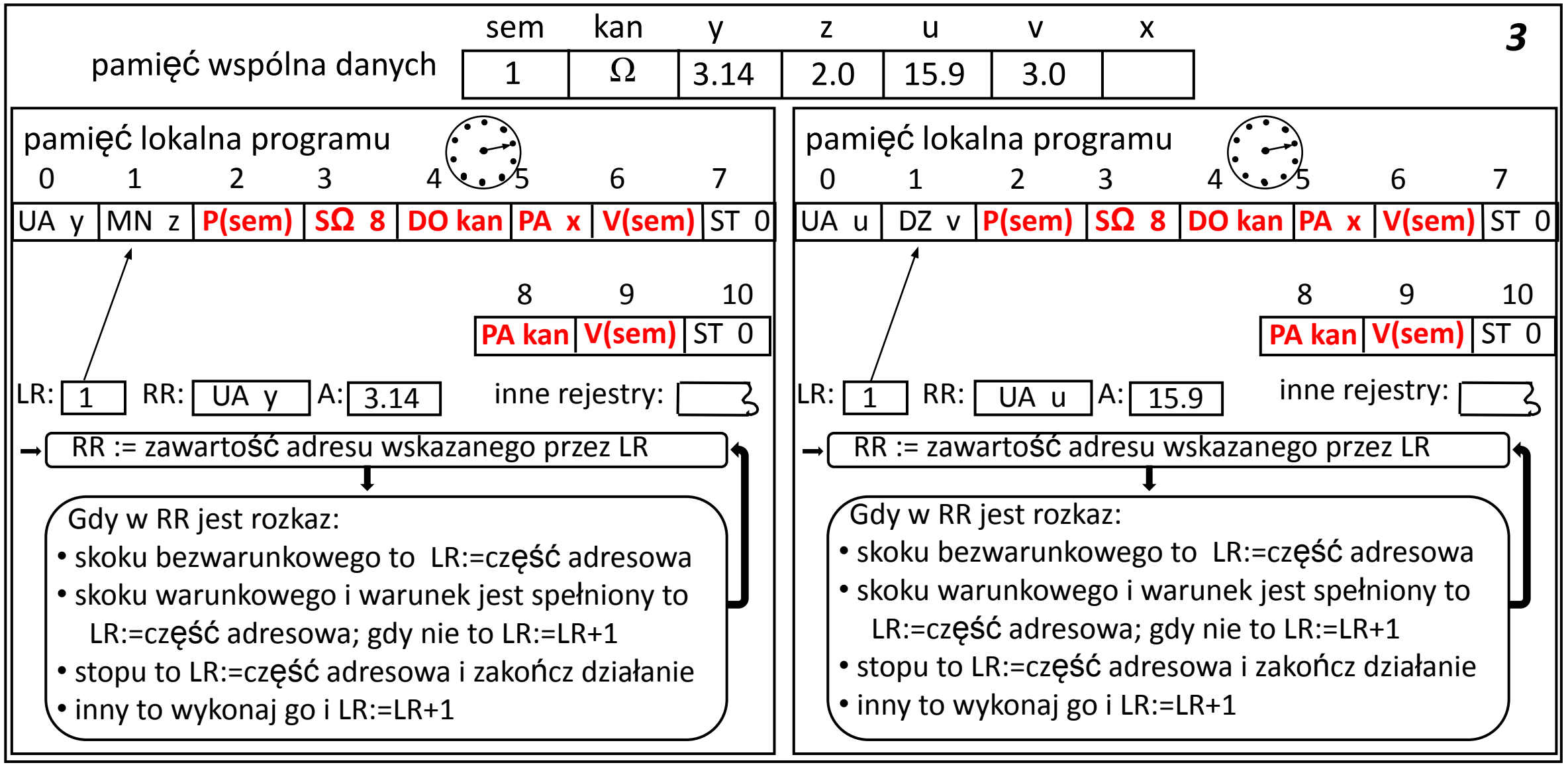
komputer 1

komputer 2



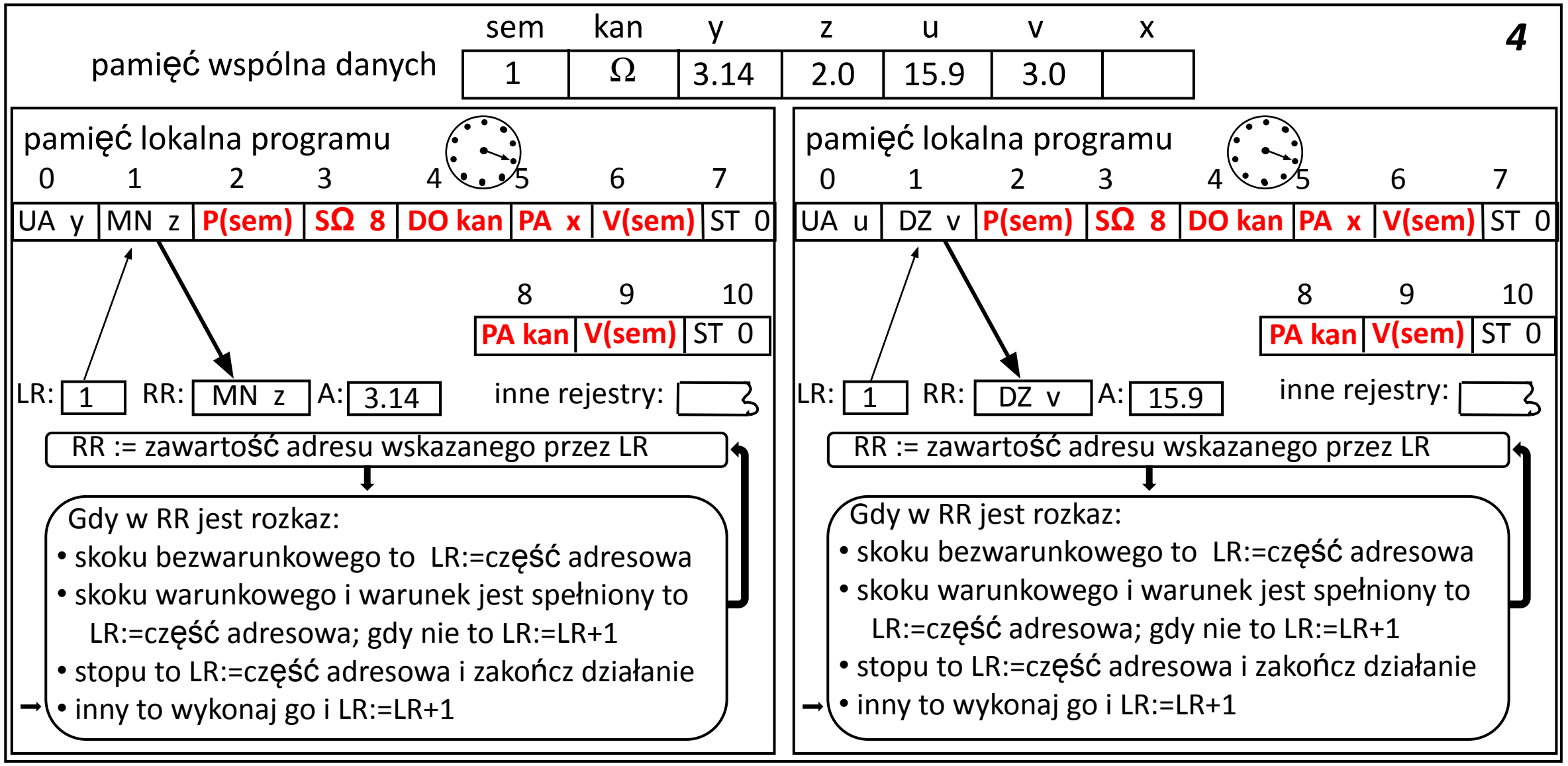
komputer 1

komputer 2



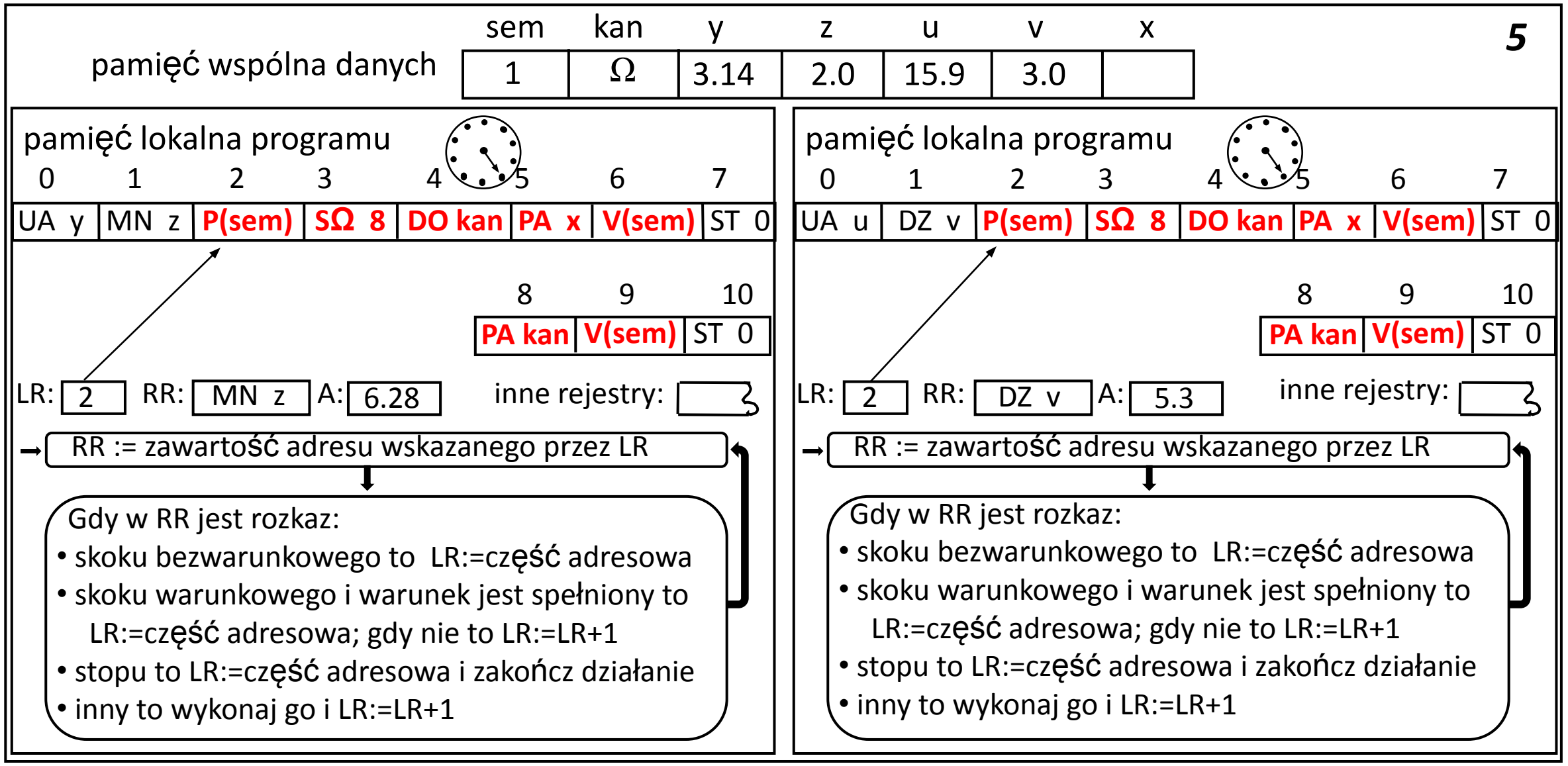
komputer 1

komputer 2



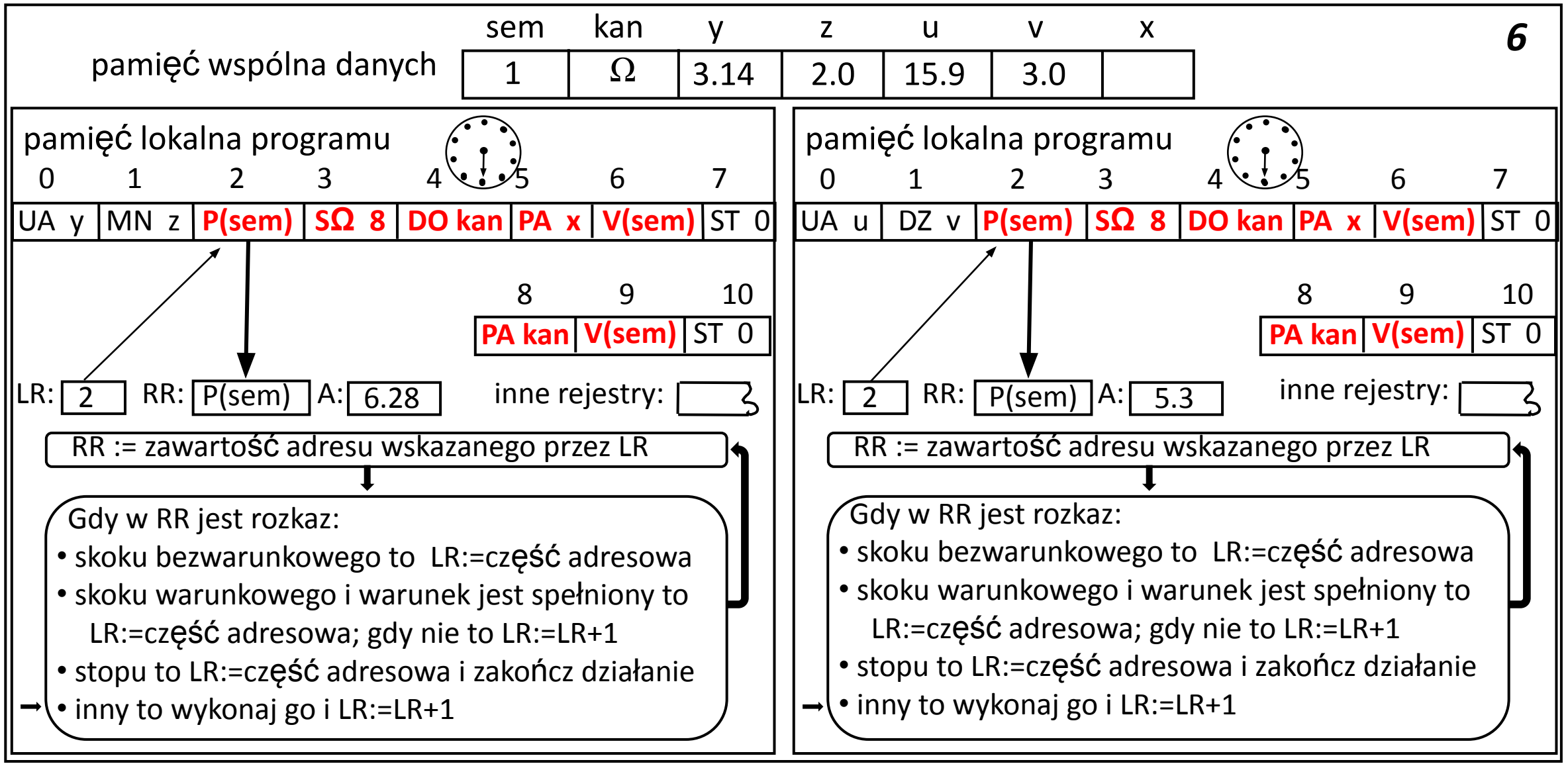
komputer 1

komputer 2



komputer 1

komputer 2



komputer 1

komputer 2

zmienia komputer 1

pamięć wspólna danych

sem	kan	y	z	u	v	x
0	Ω	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8		DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 3 RR: P(sem) A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8		DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

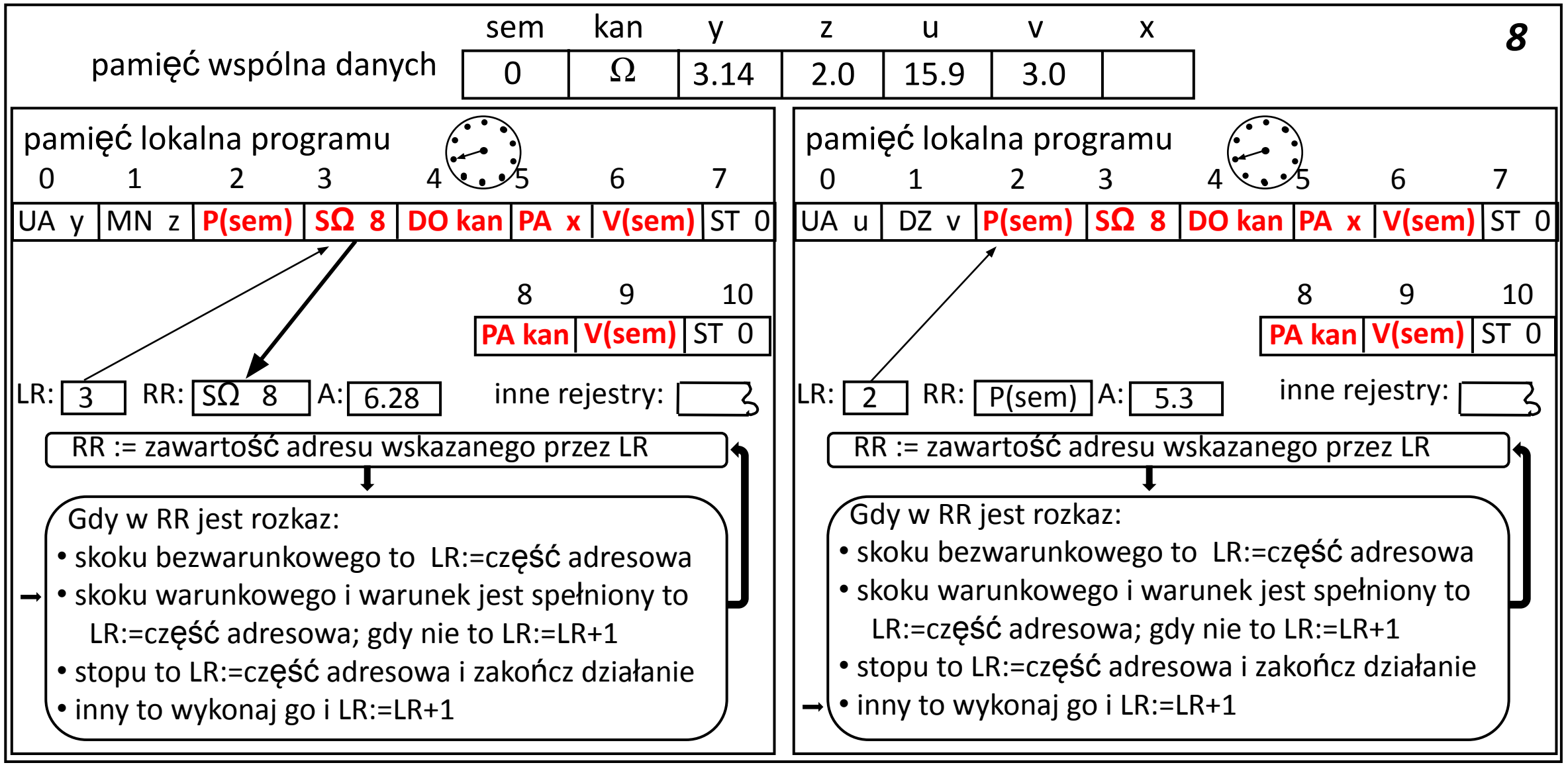
LR: 2 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

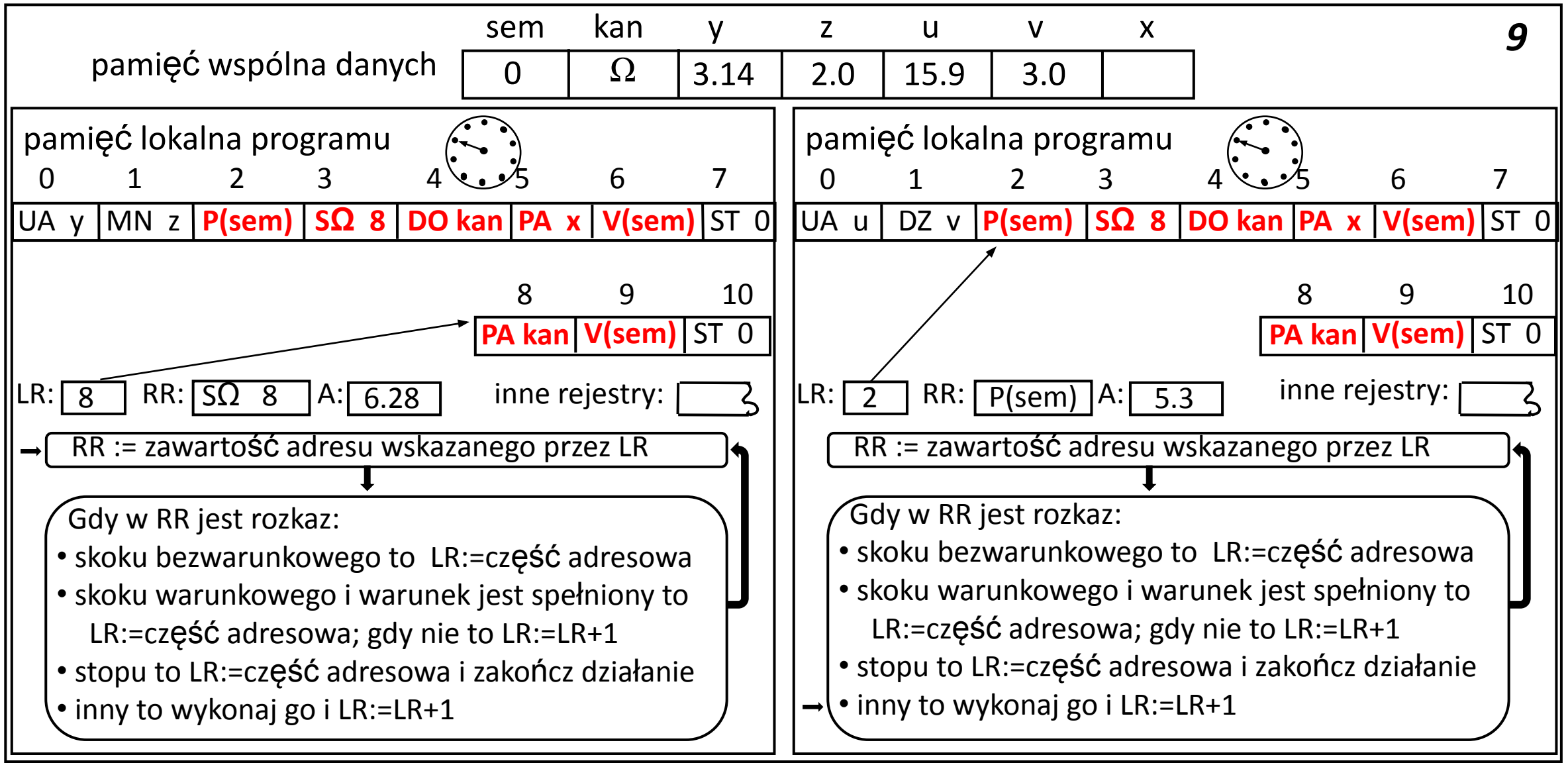
komputer 1

komputer 2
(wstrzymany)



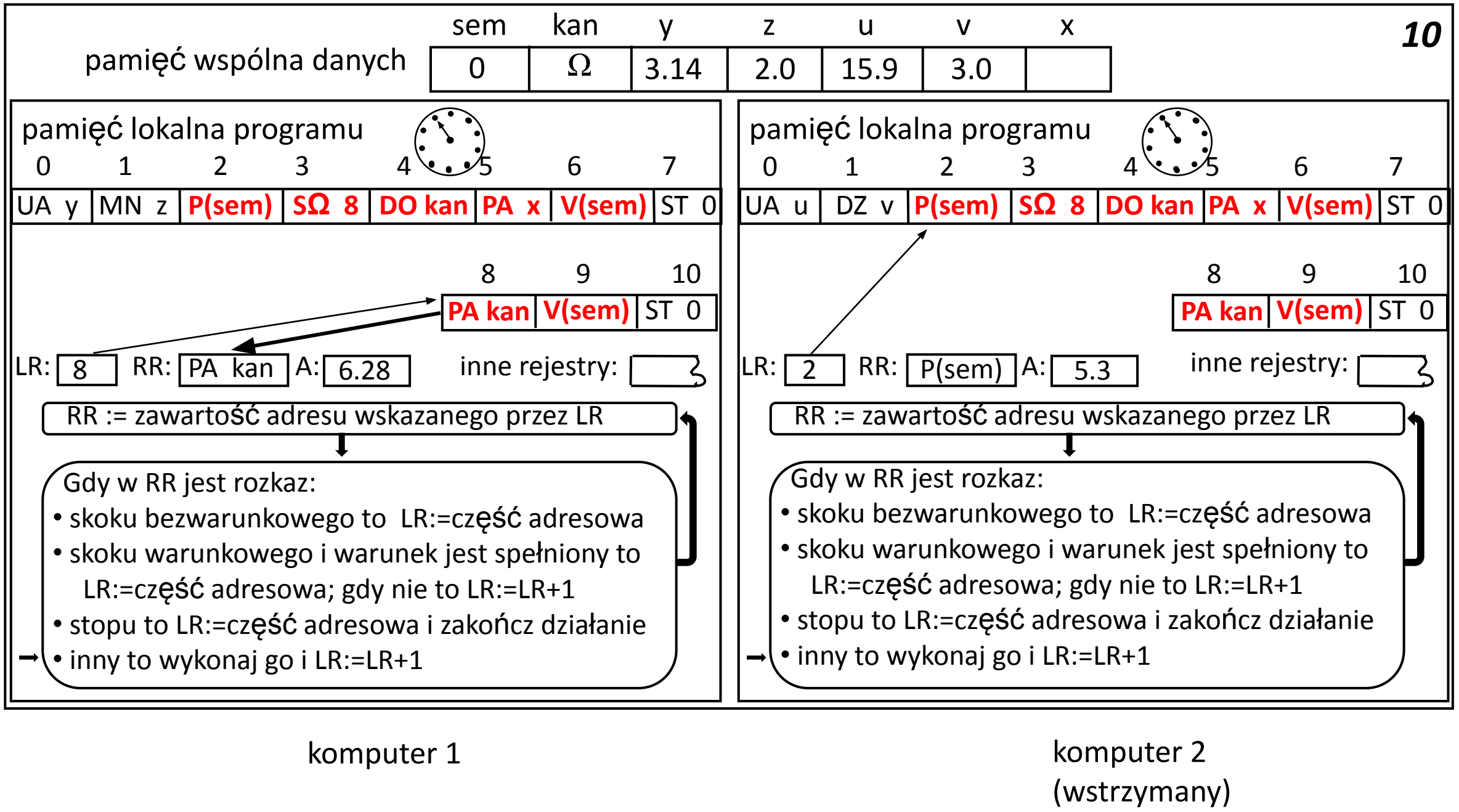
komputer 1

komputer 2
(wstrzymany)



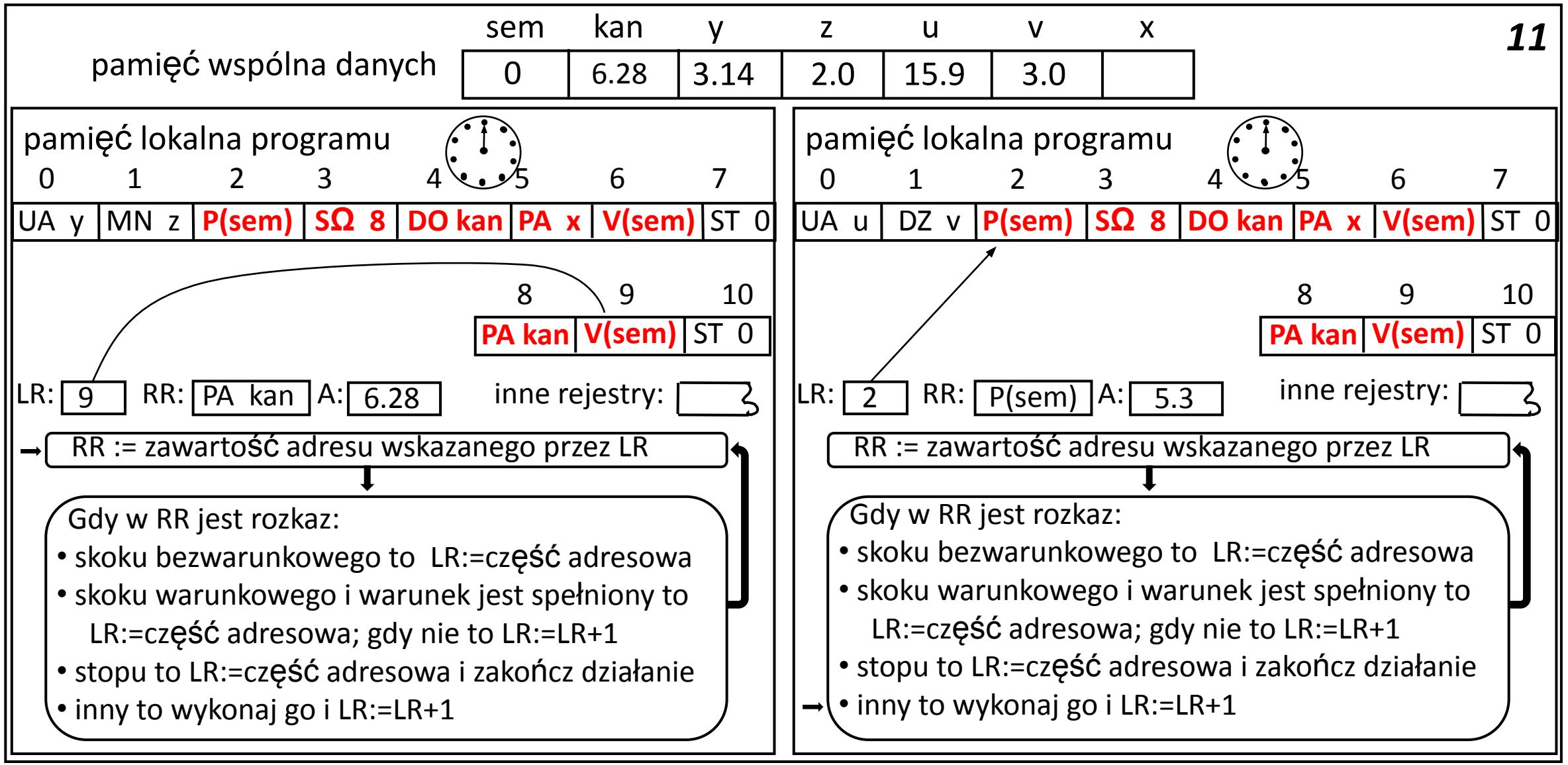
komputer 1

komputer 2
(wstrzymany)



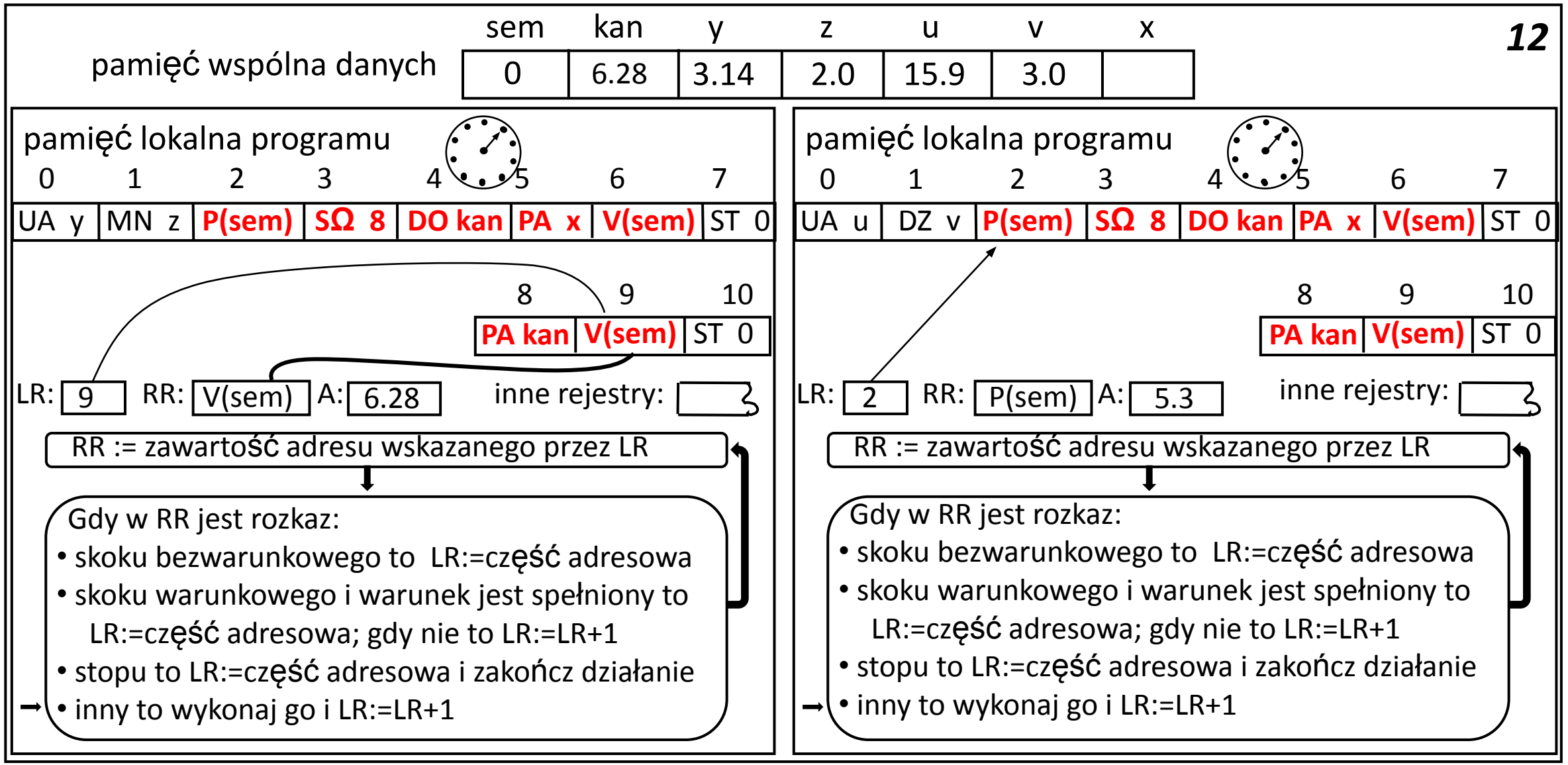
komputer 1

komputer 2
(wstrzymany)



komputer 1

komputer 2
(wstrzymany)



komputer 1

komputer 2
(wstrzymany)

zmienia komputer 1

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 10 RR: V(sem) A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

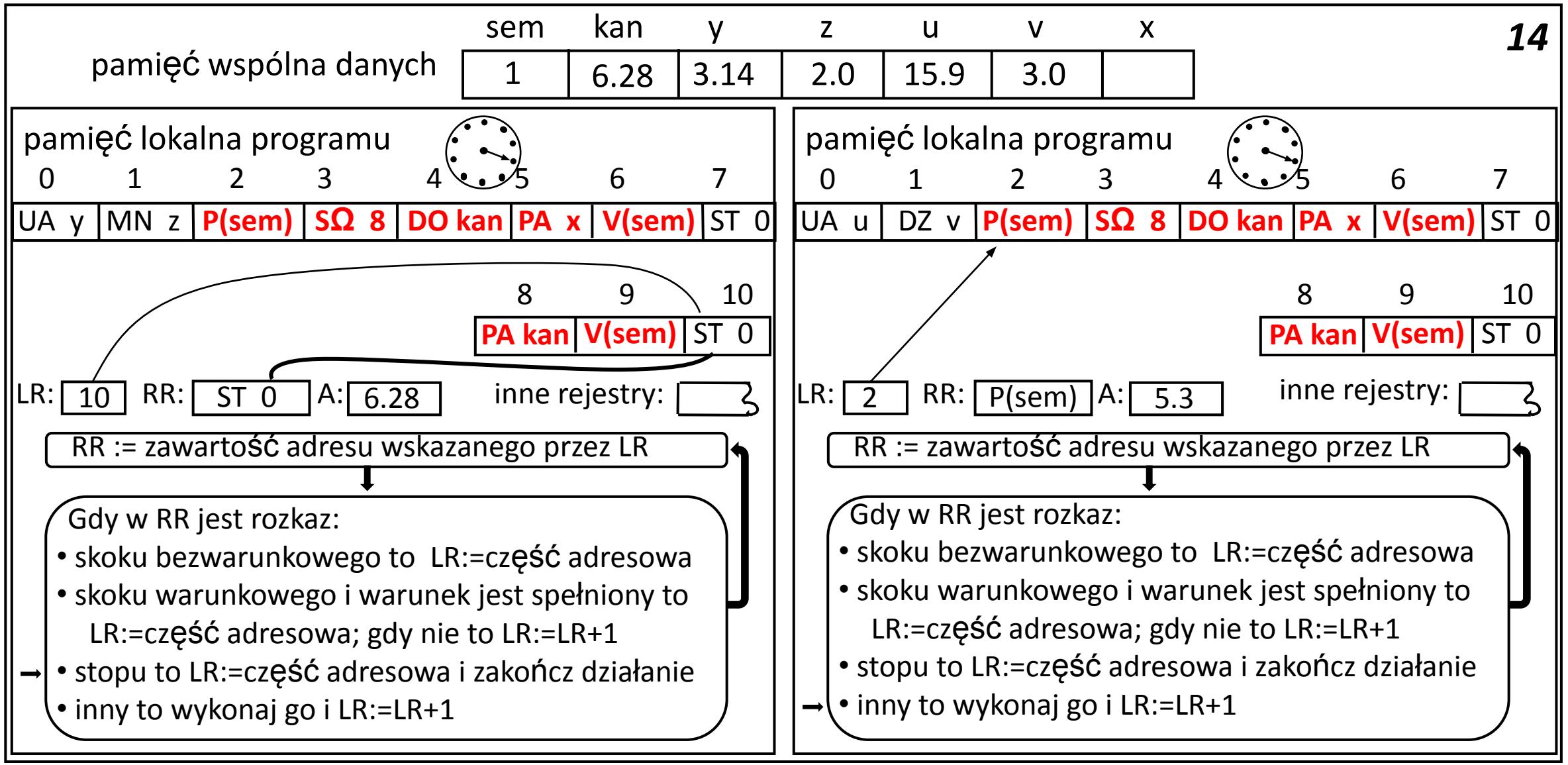
LR: 2 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 1

komputer 2
(wstrzymany)



komputer 1

komputer 2
(wstrzymany)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
0	6.28	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

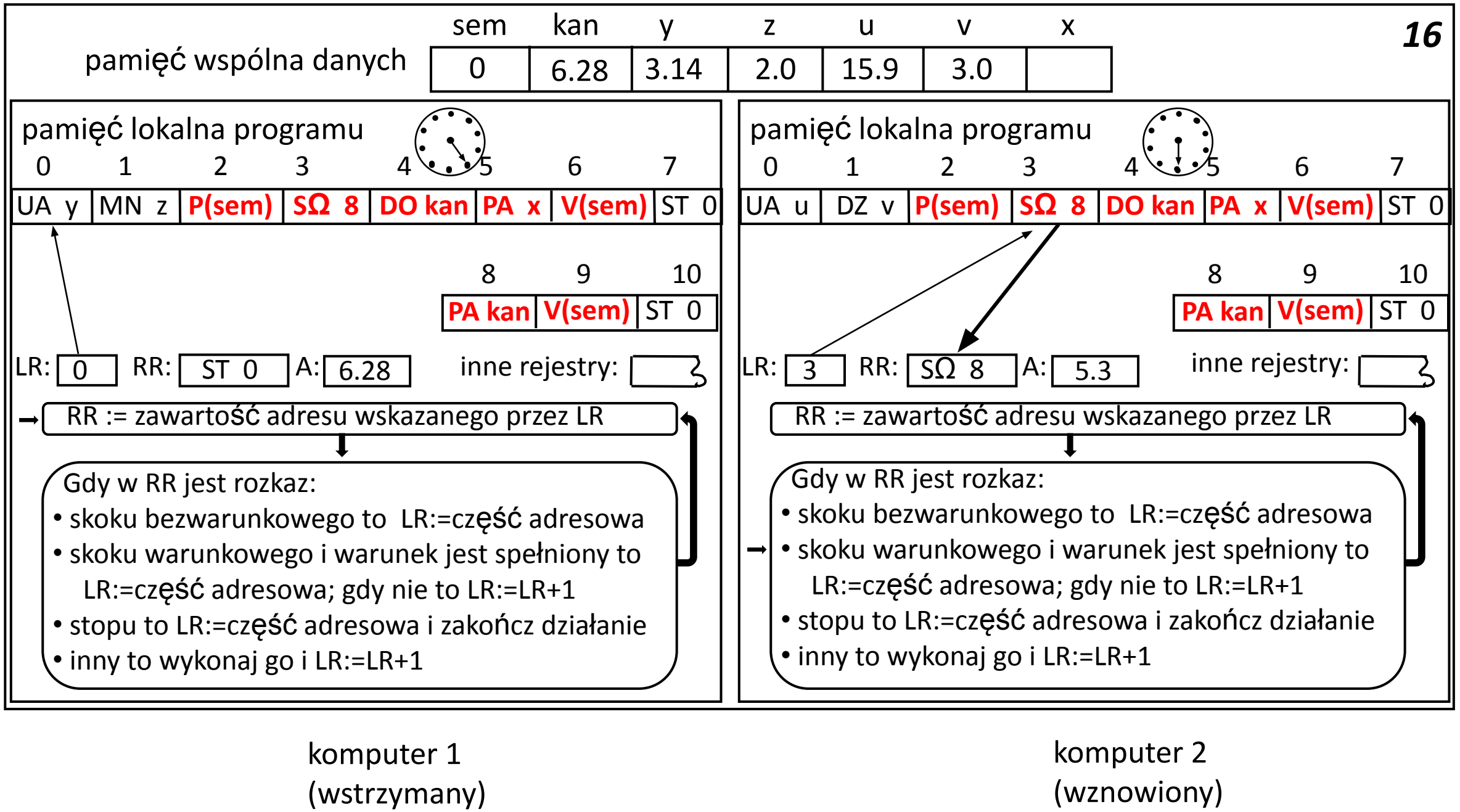
LR: 3 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

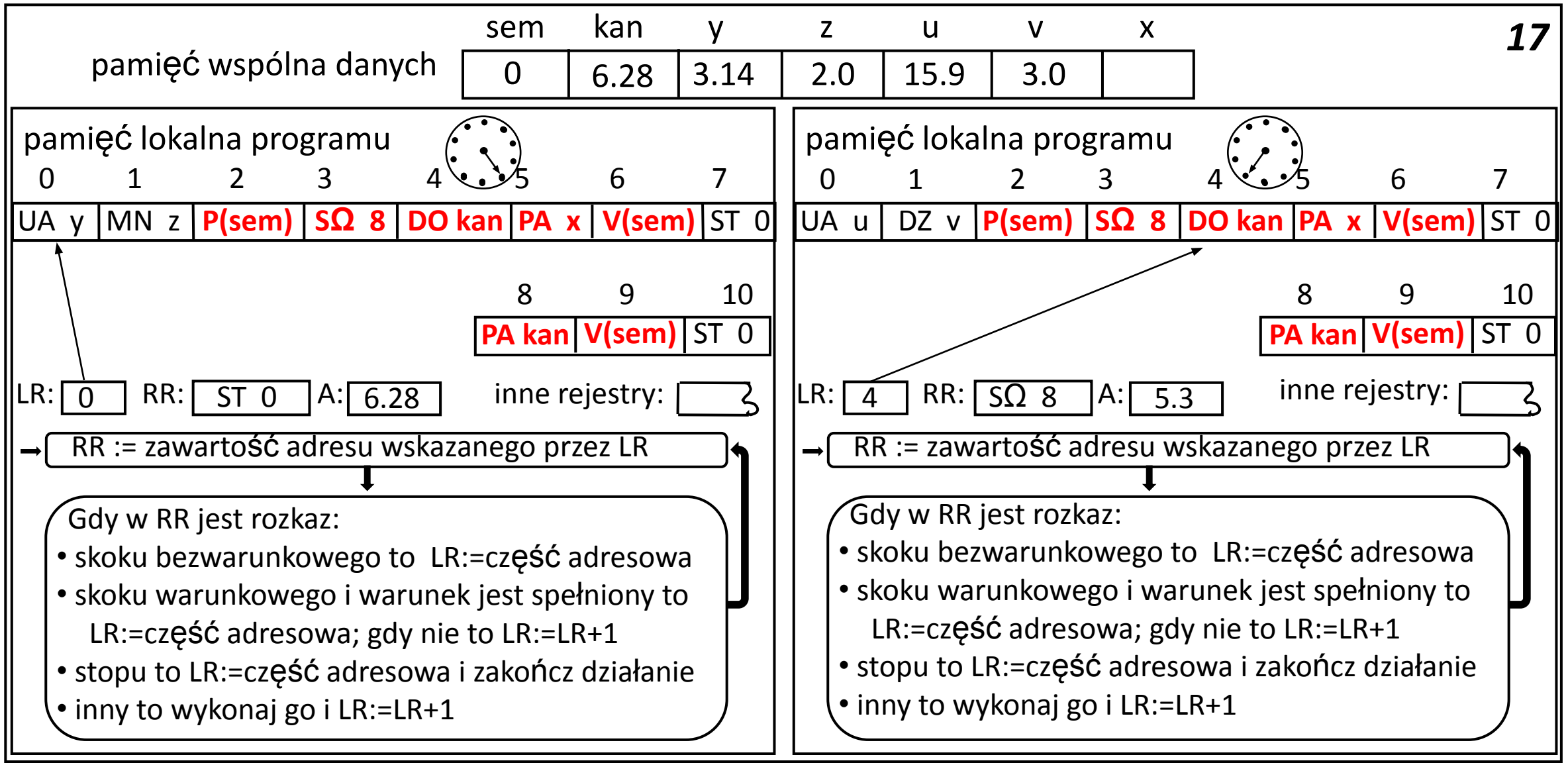
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



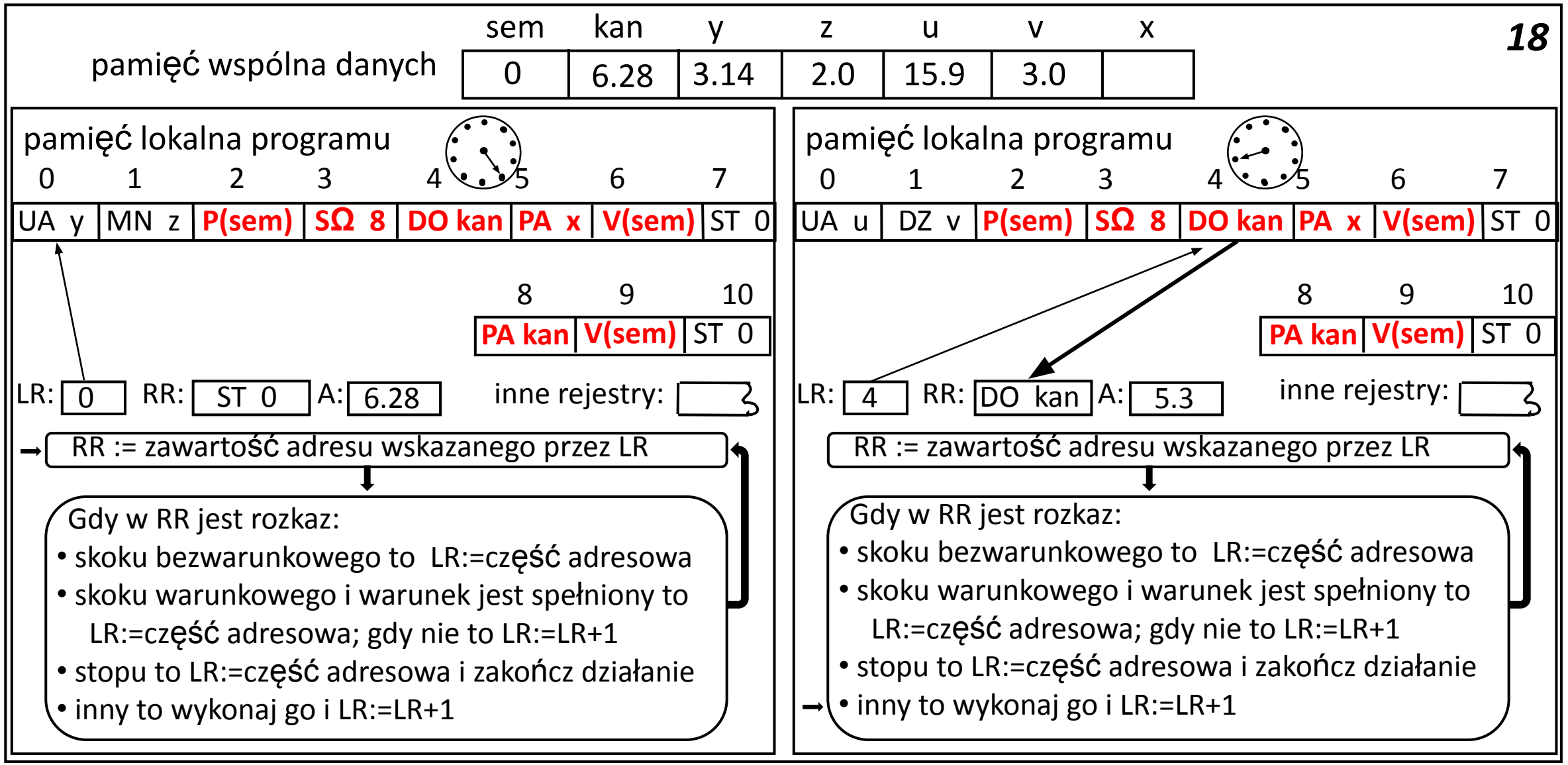
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



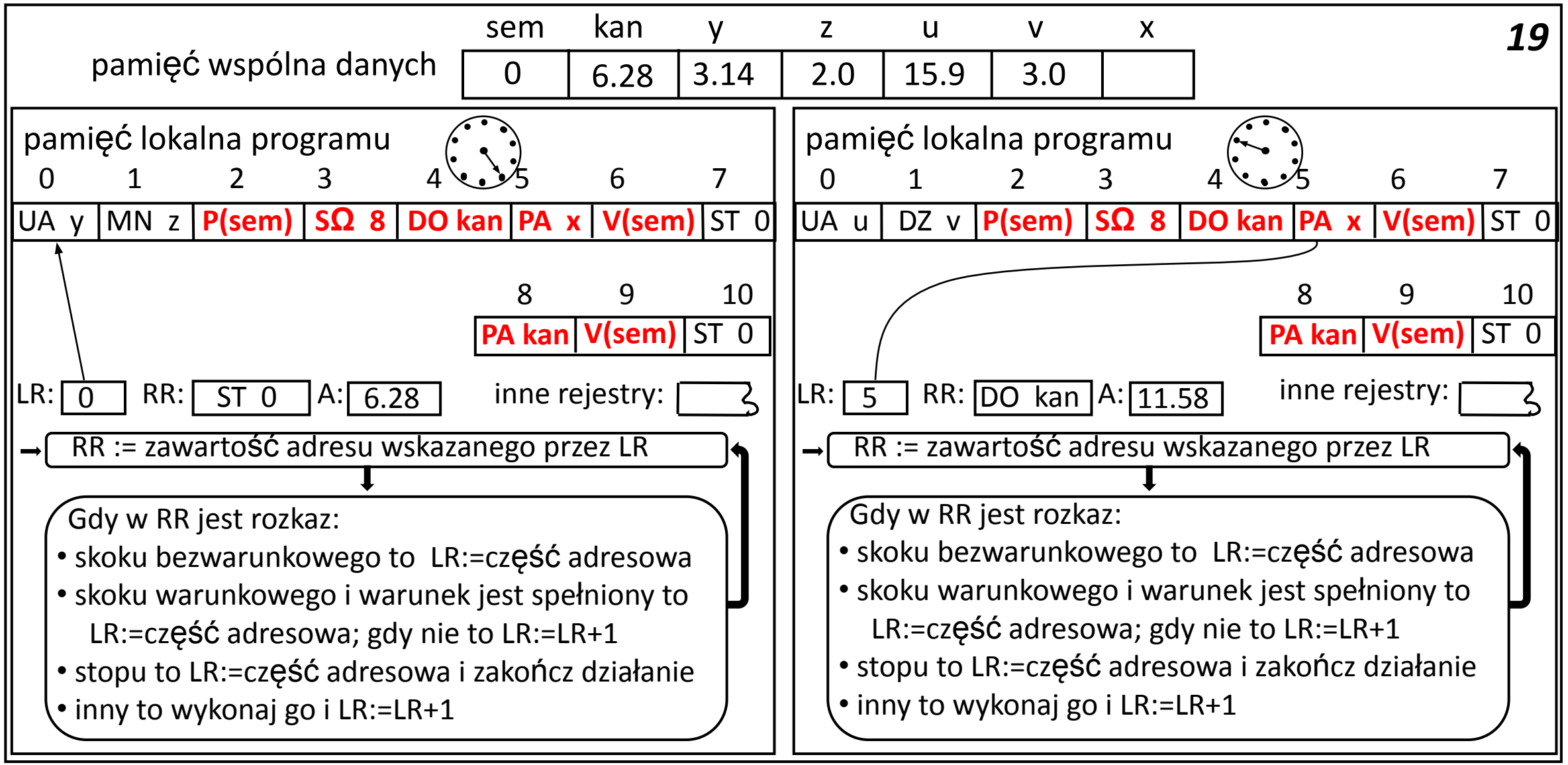
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



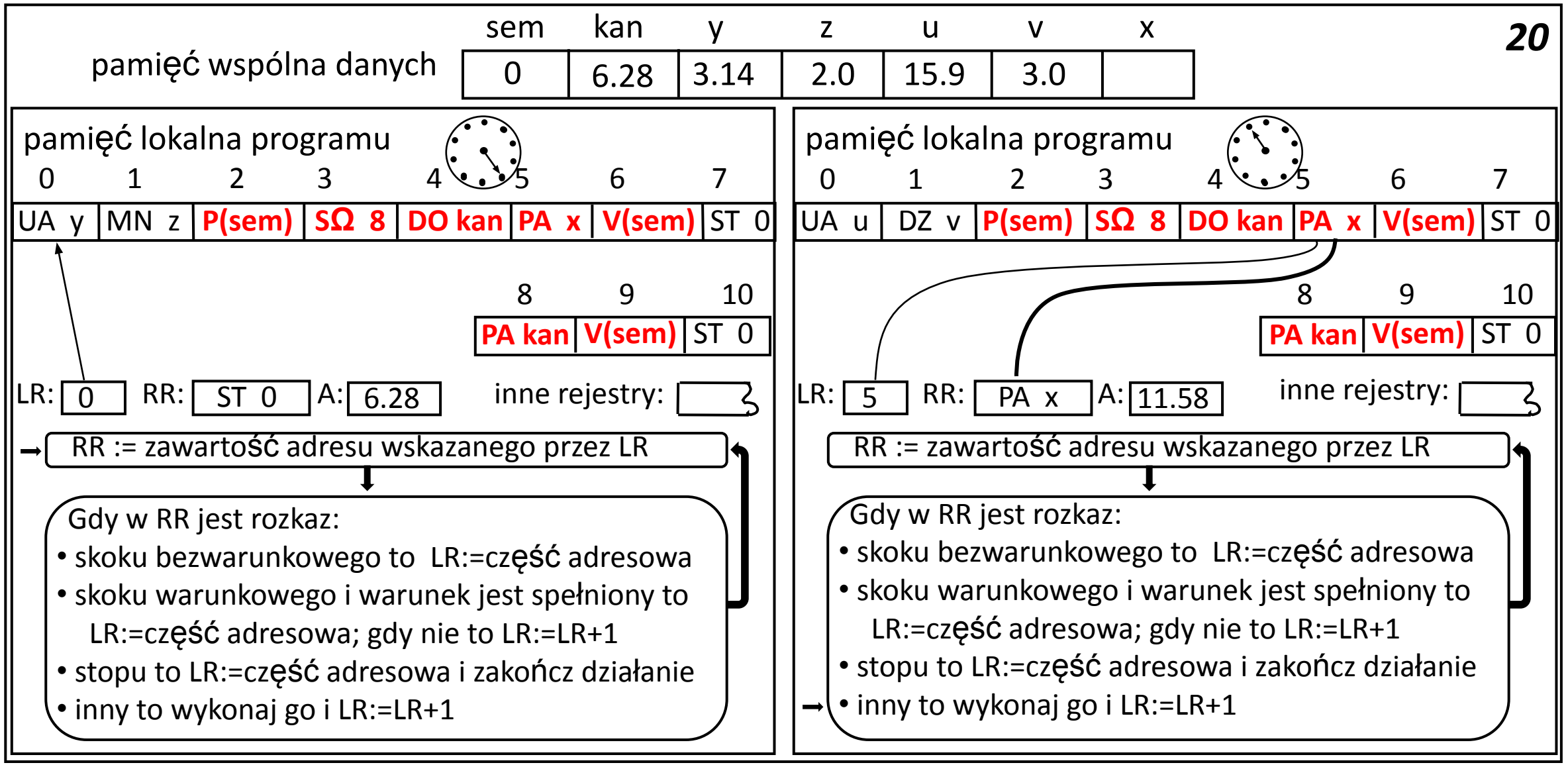
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



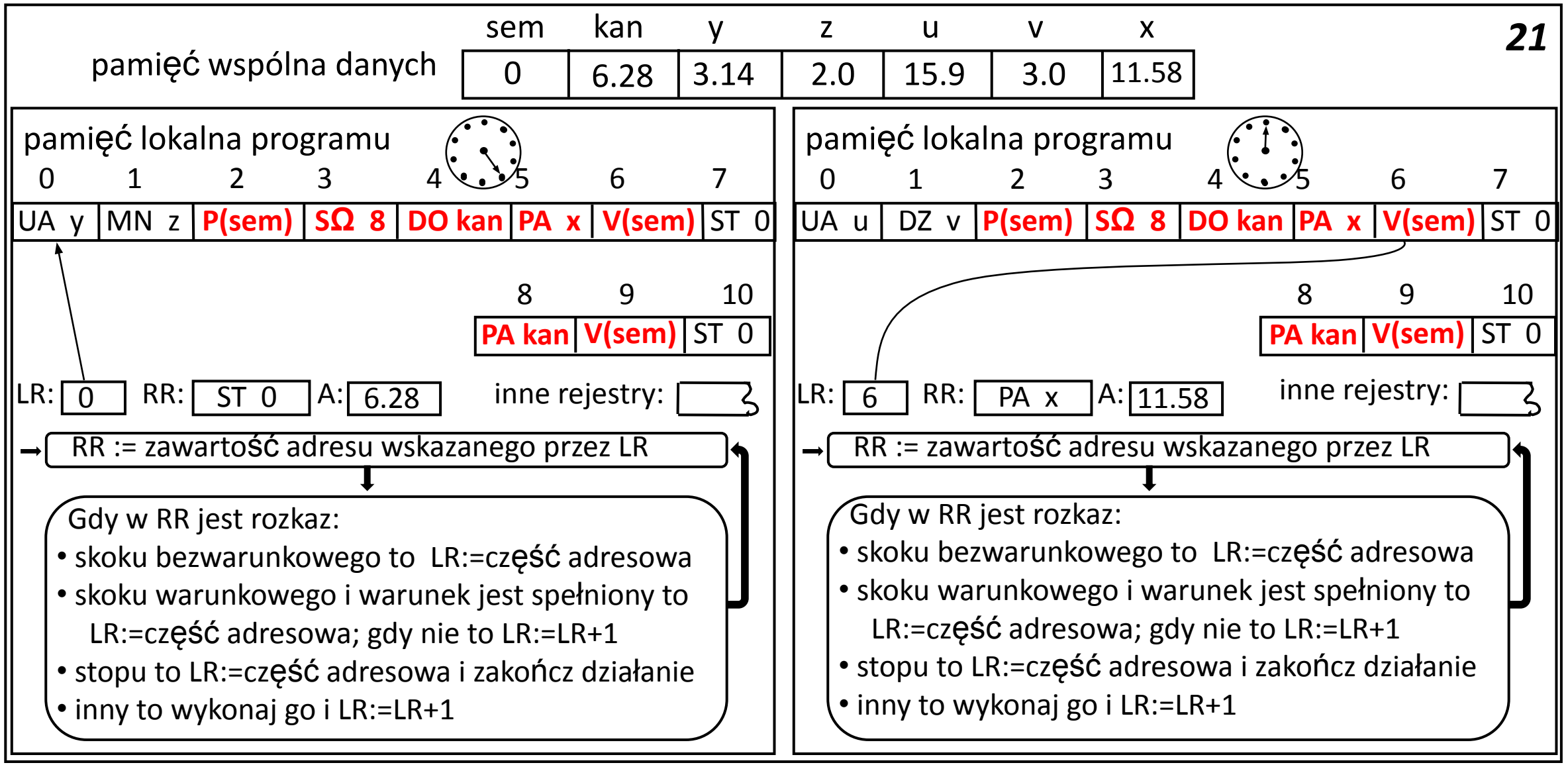
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



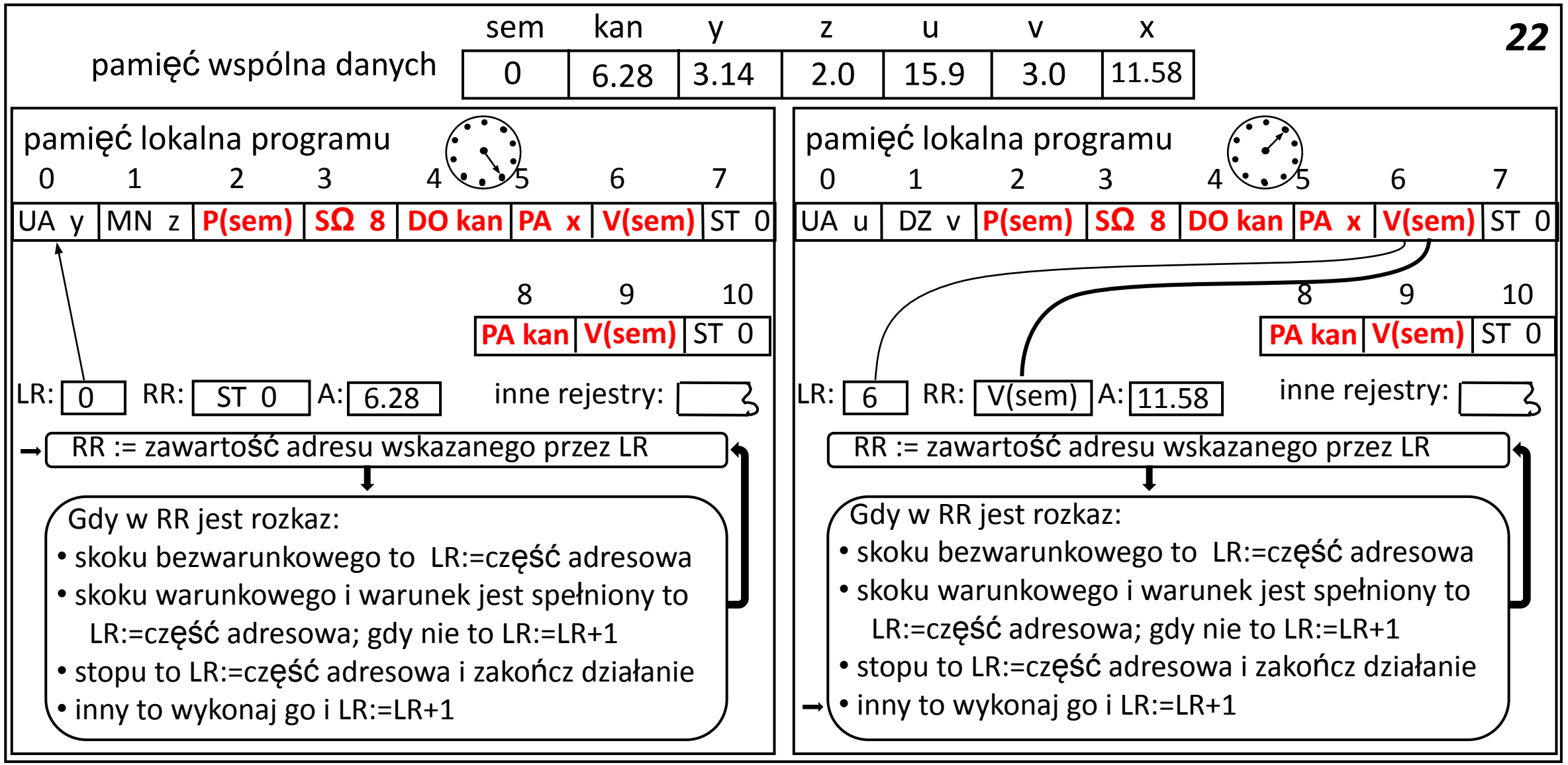
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wstrzymany)

komputer 2
(wznowiony)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	11.58

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

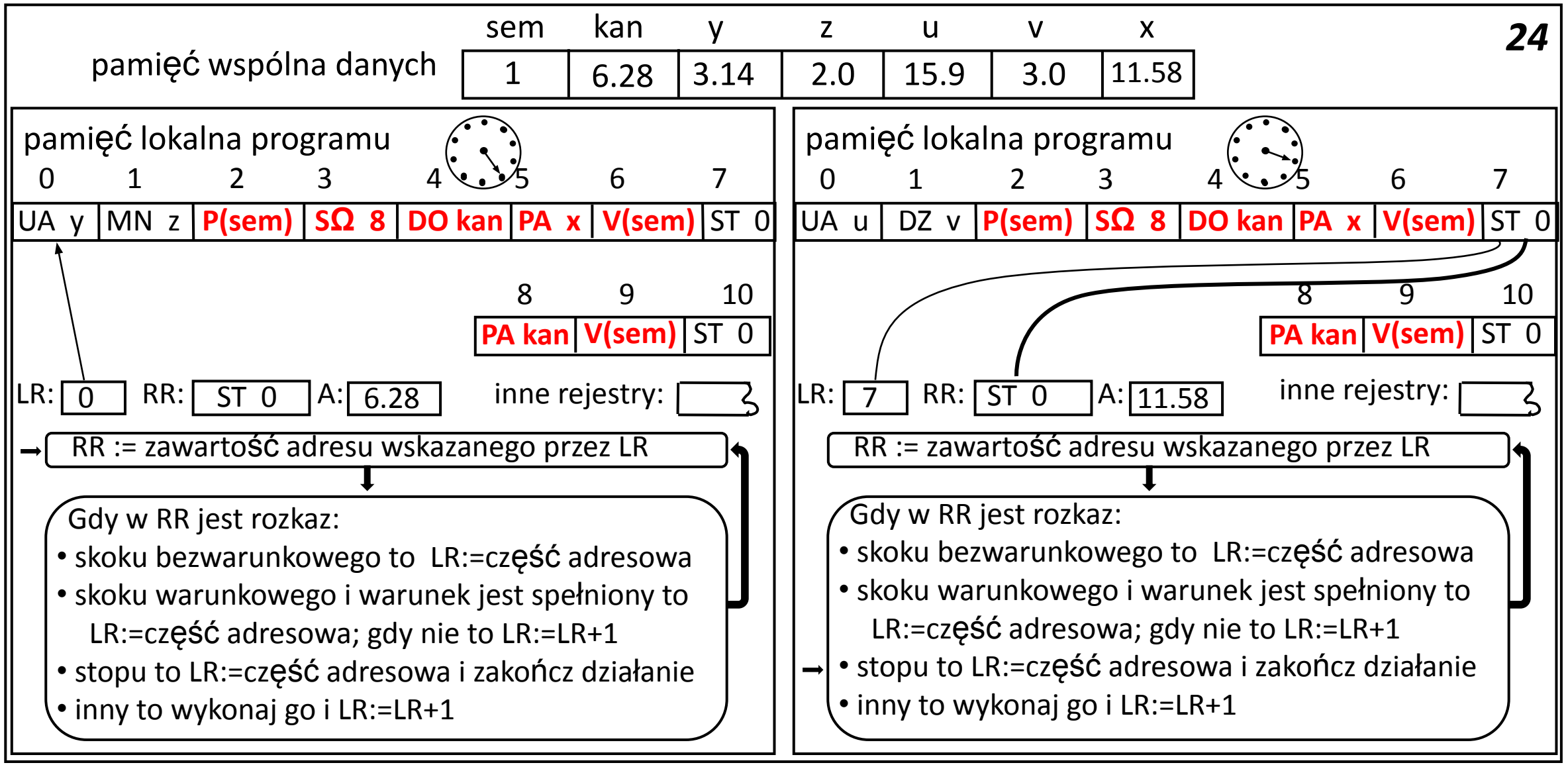
LR: 7 RR: V(sem) A: 11.58 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

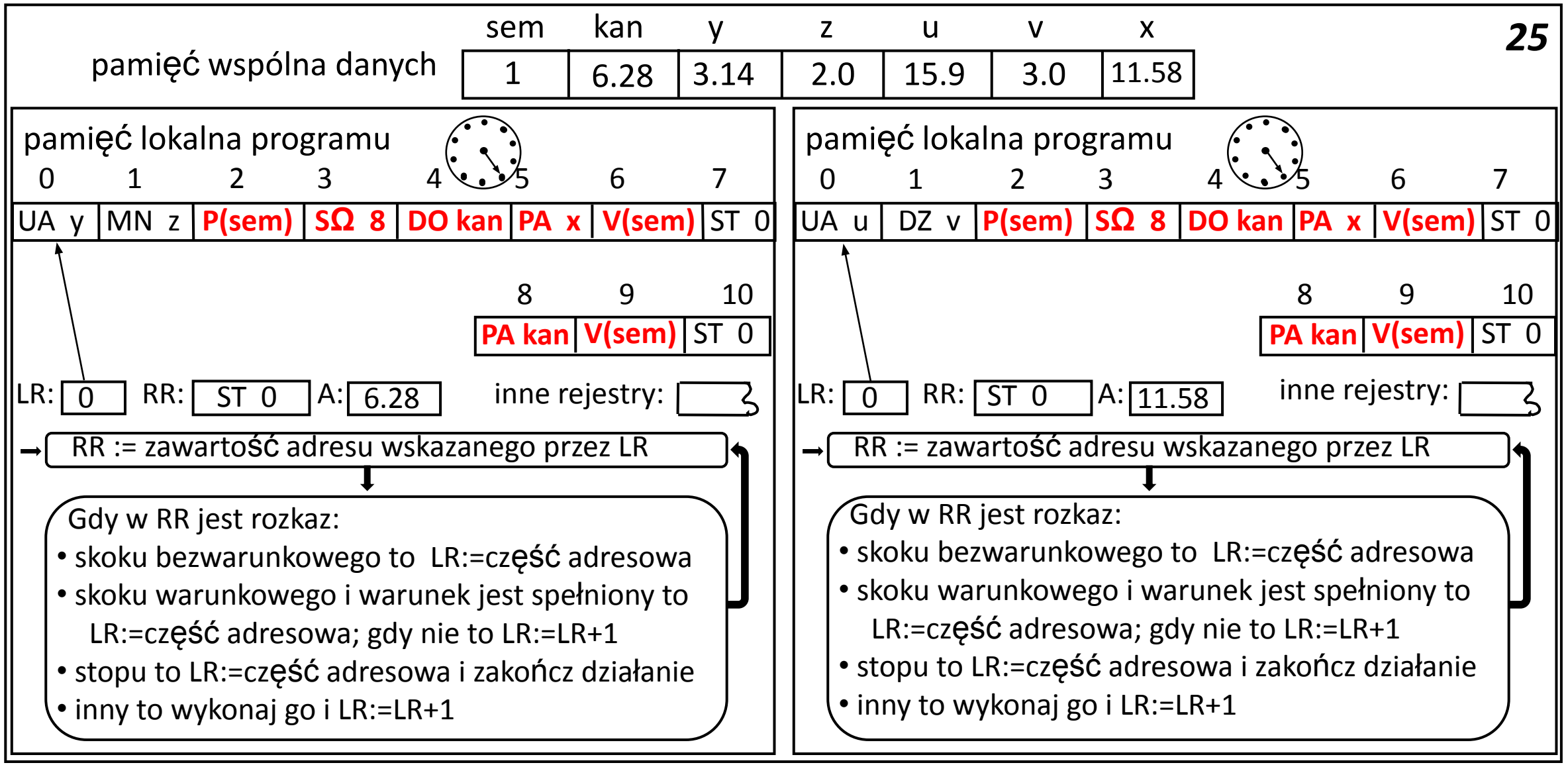
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wznowiony)

komputer 2
(wznowiony)



komputer 1
(wznowiony)

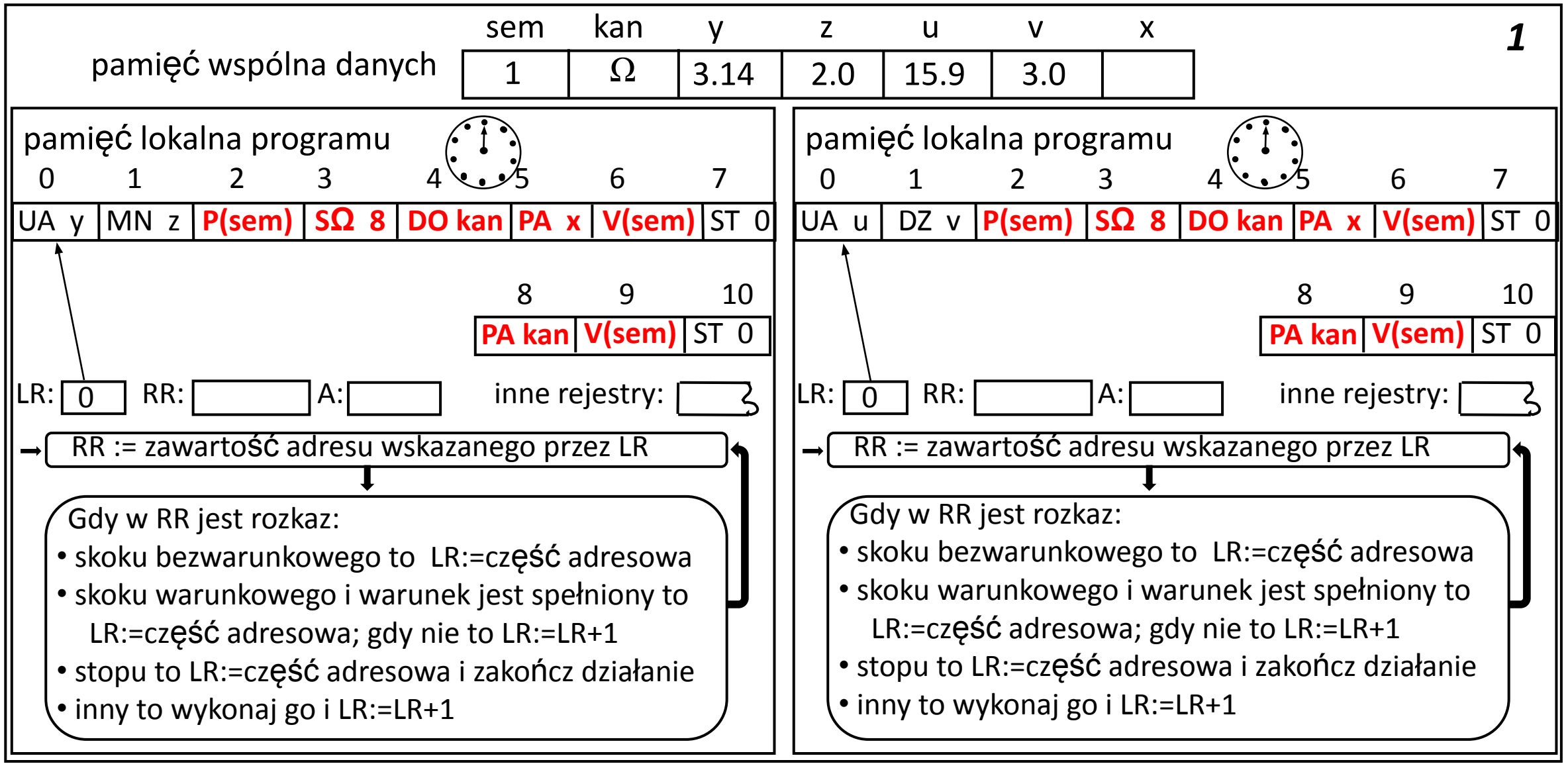
komputer 2
(wznowiony)

Symulacja wykonania programu na schemacie funkcjonalnym systemu 2-procesorowego z dostępem do wspólnej pamięci danych dla obydwu procesorów i z różną prędkością ich zegarów.
Synchronizacja semaforowa – wzajemne wykluczanie.
Sekcja krytyczna: rozkazy **czerwone**

Rozwiązanie poprawne!

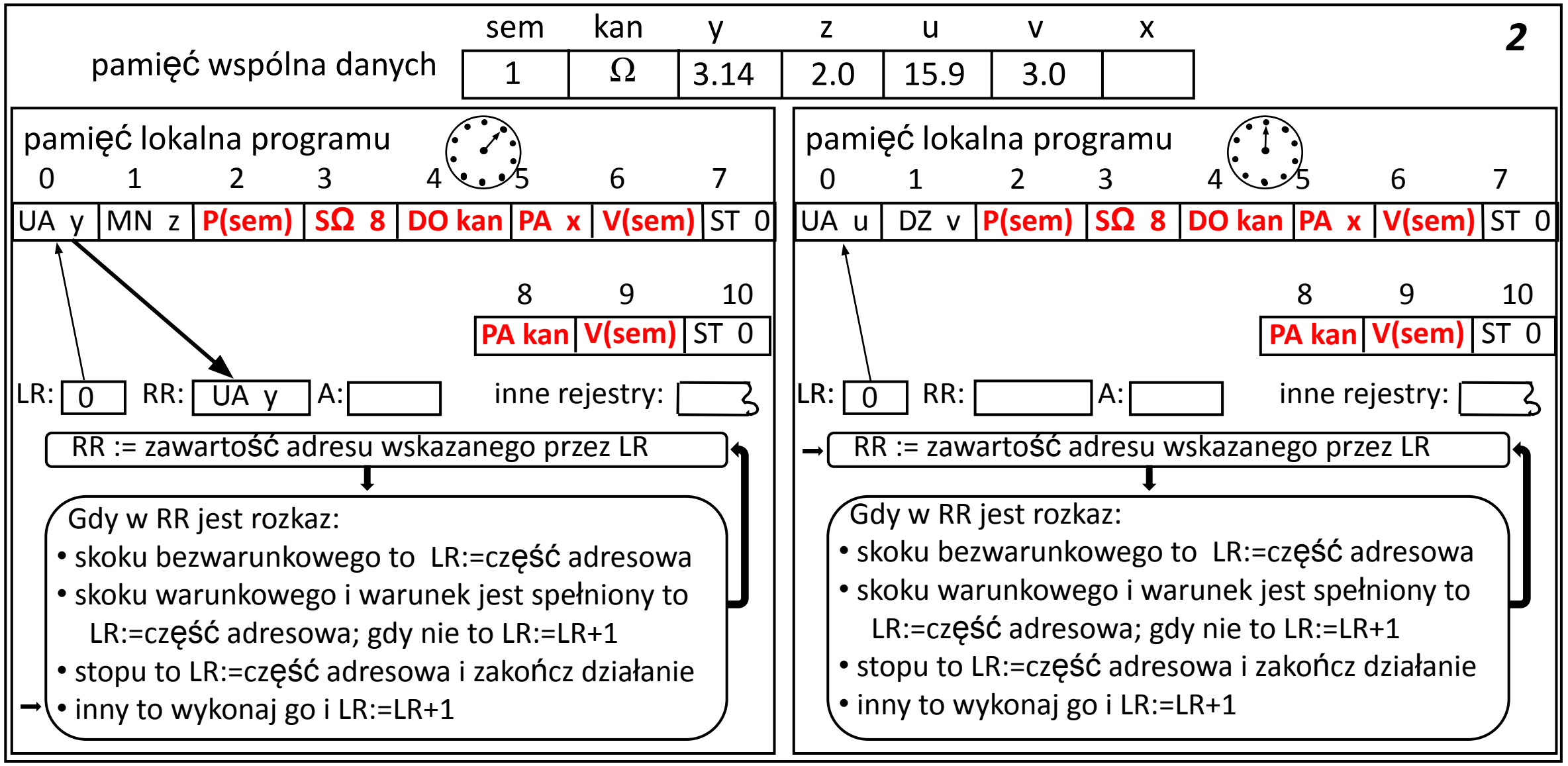
LR – licznik rozkazów RR – rejestr rozkazów A – akumulator

Przykład działania: obliczenie $x := y*z + u/v$



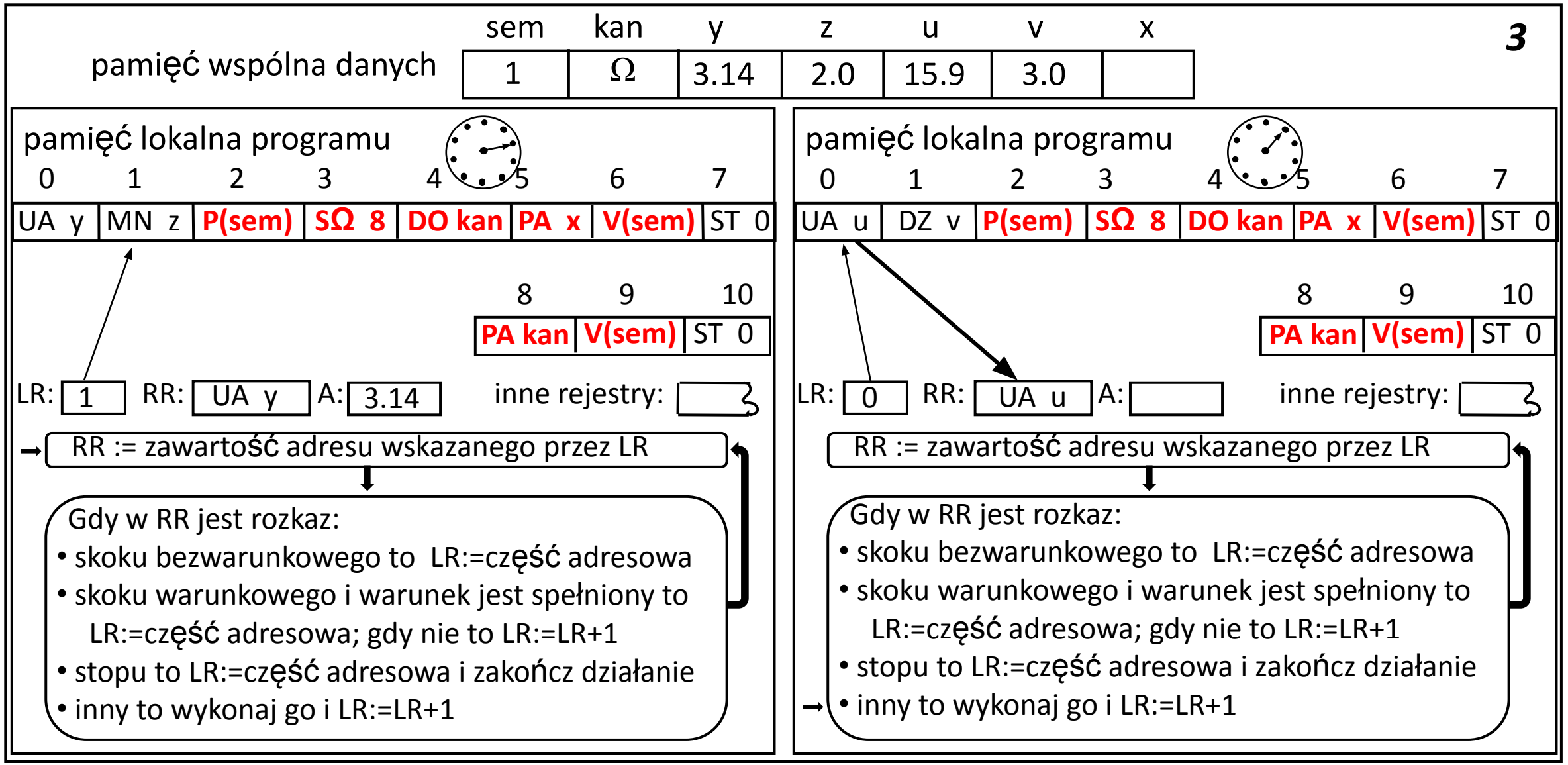
komputer 1

komputer 2



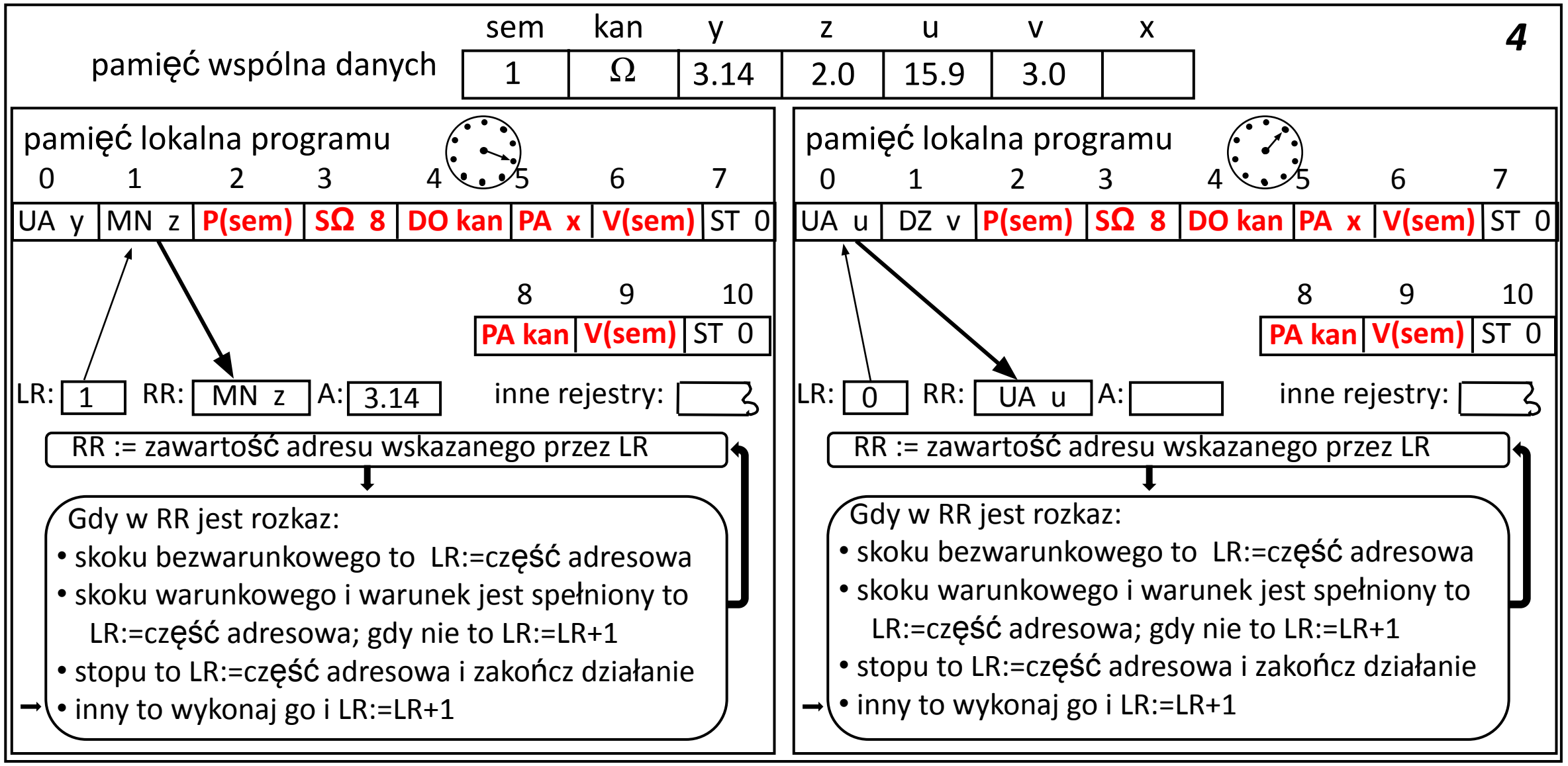
komputer 1

komputer 2



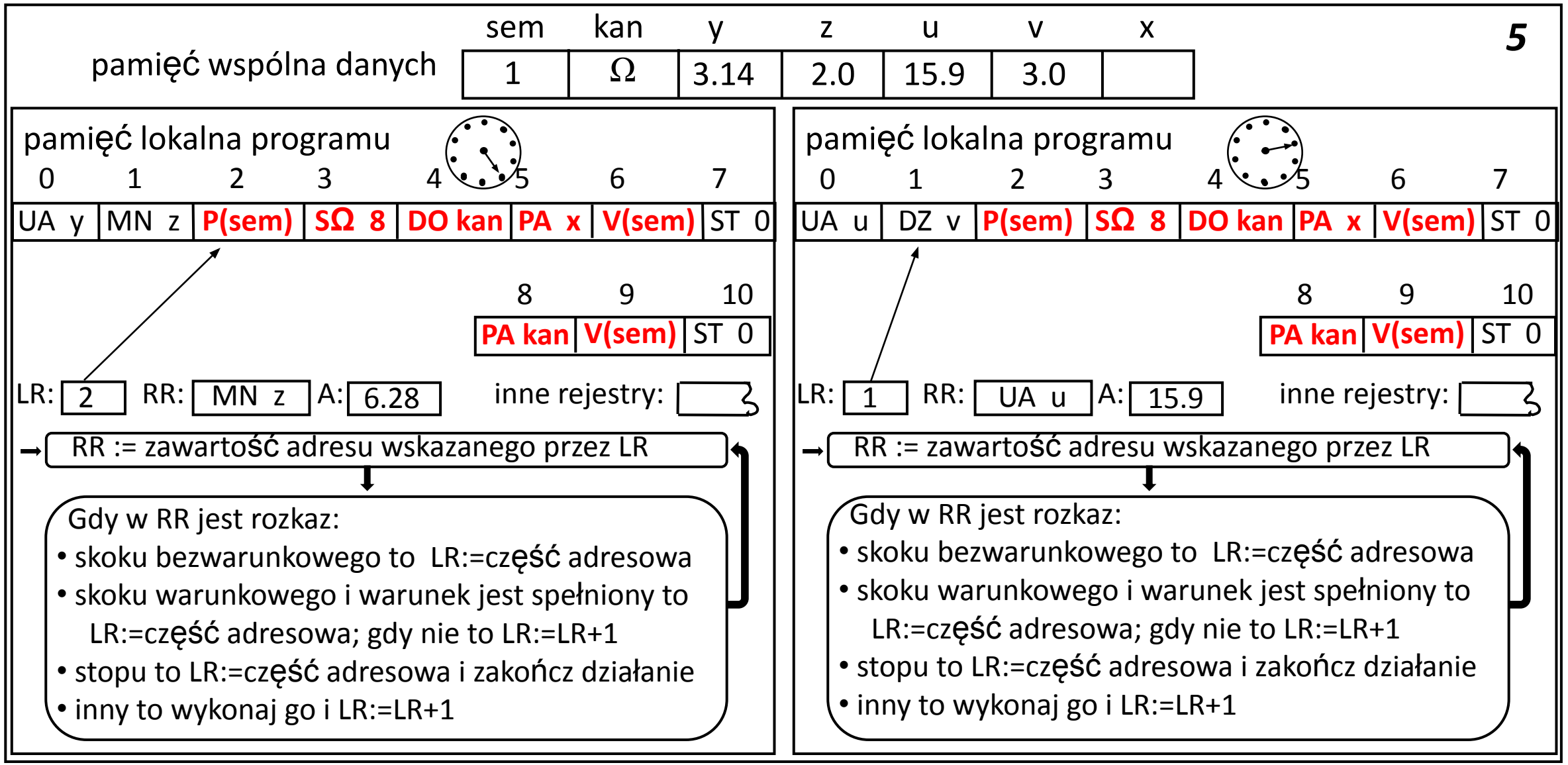
komputer 1

komputer 2



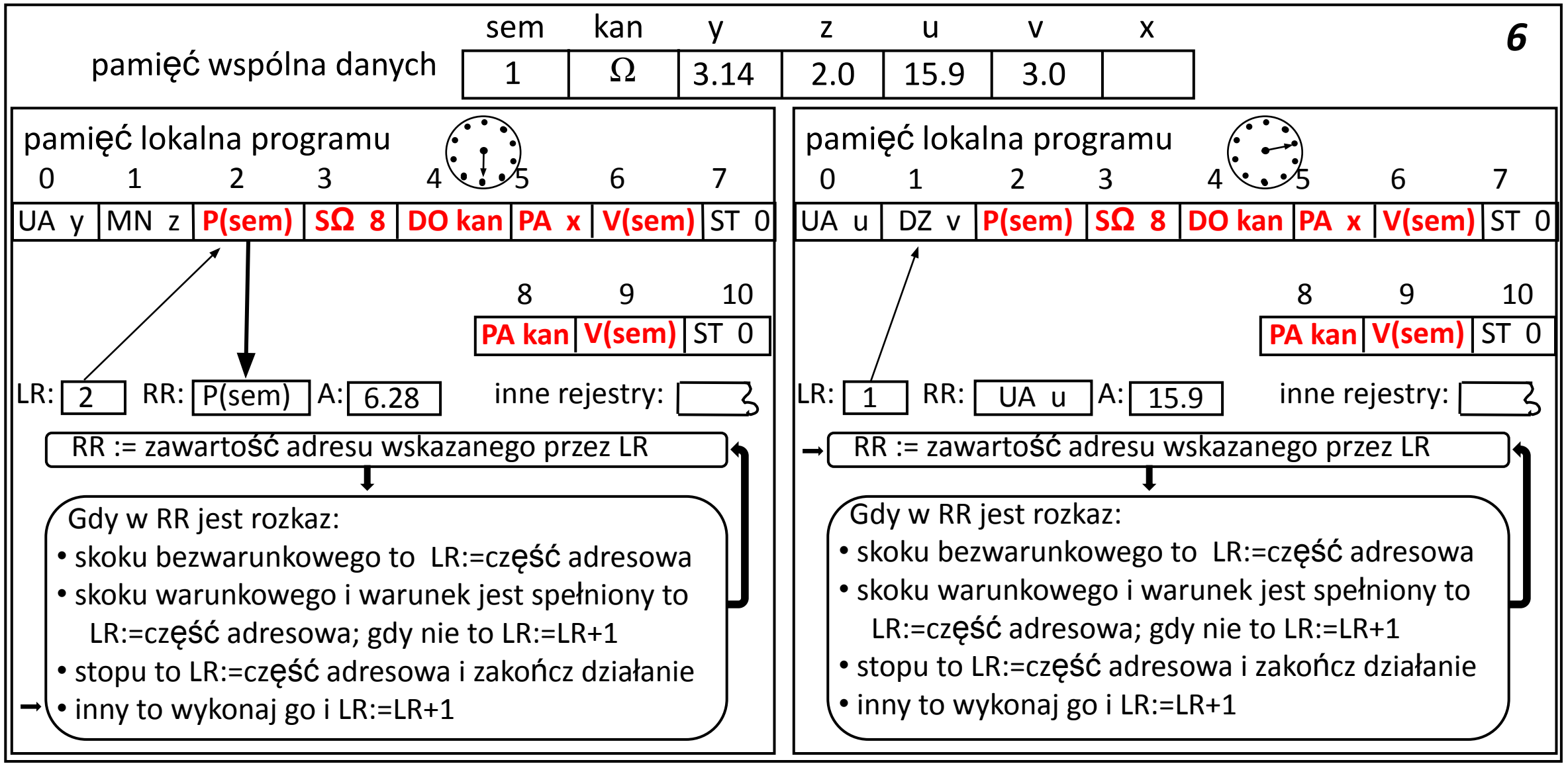
komputer 1

komputer 2



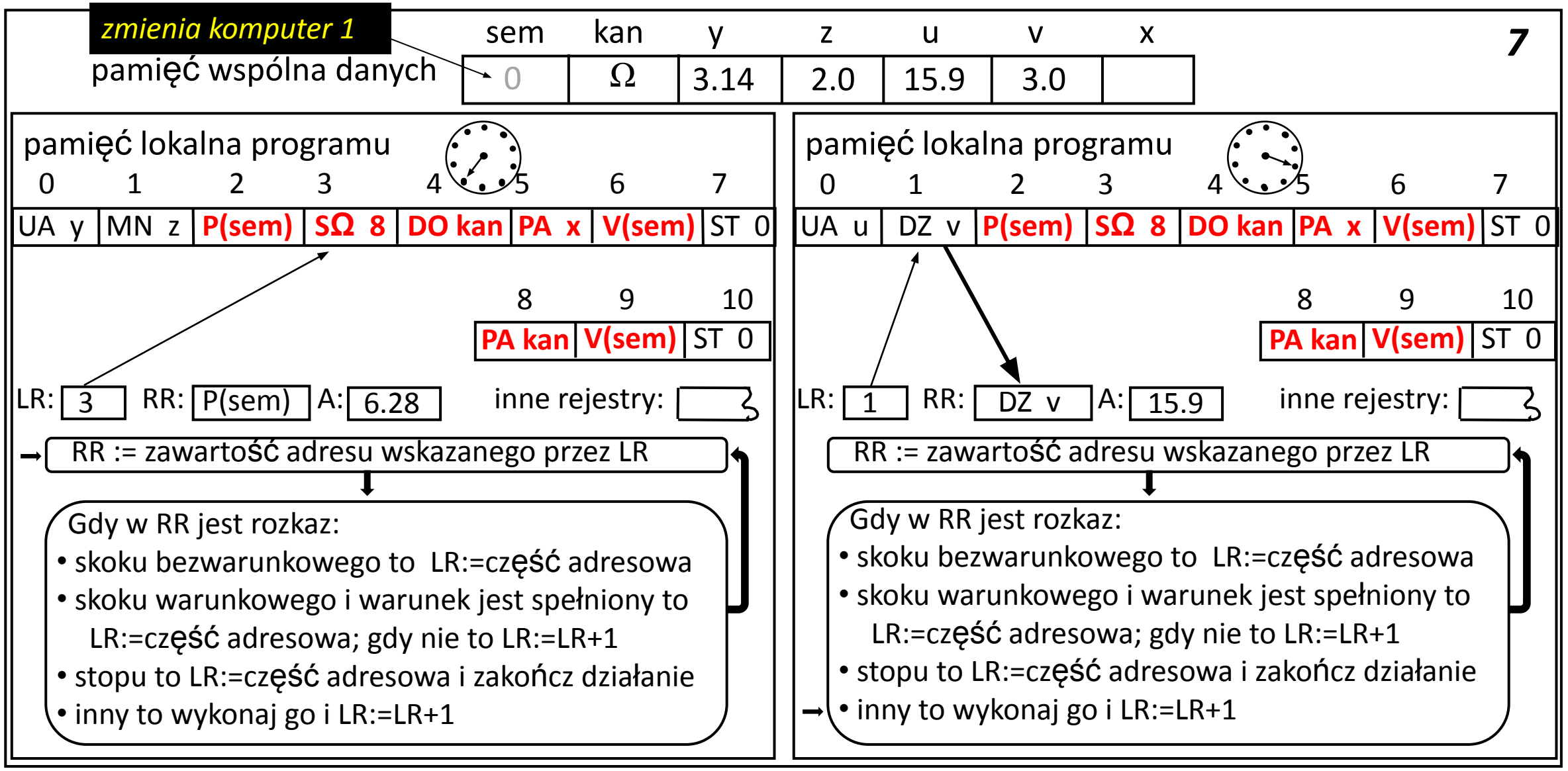
komputer 1

komputer 2



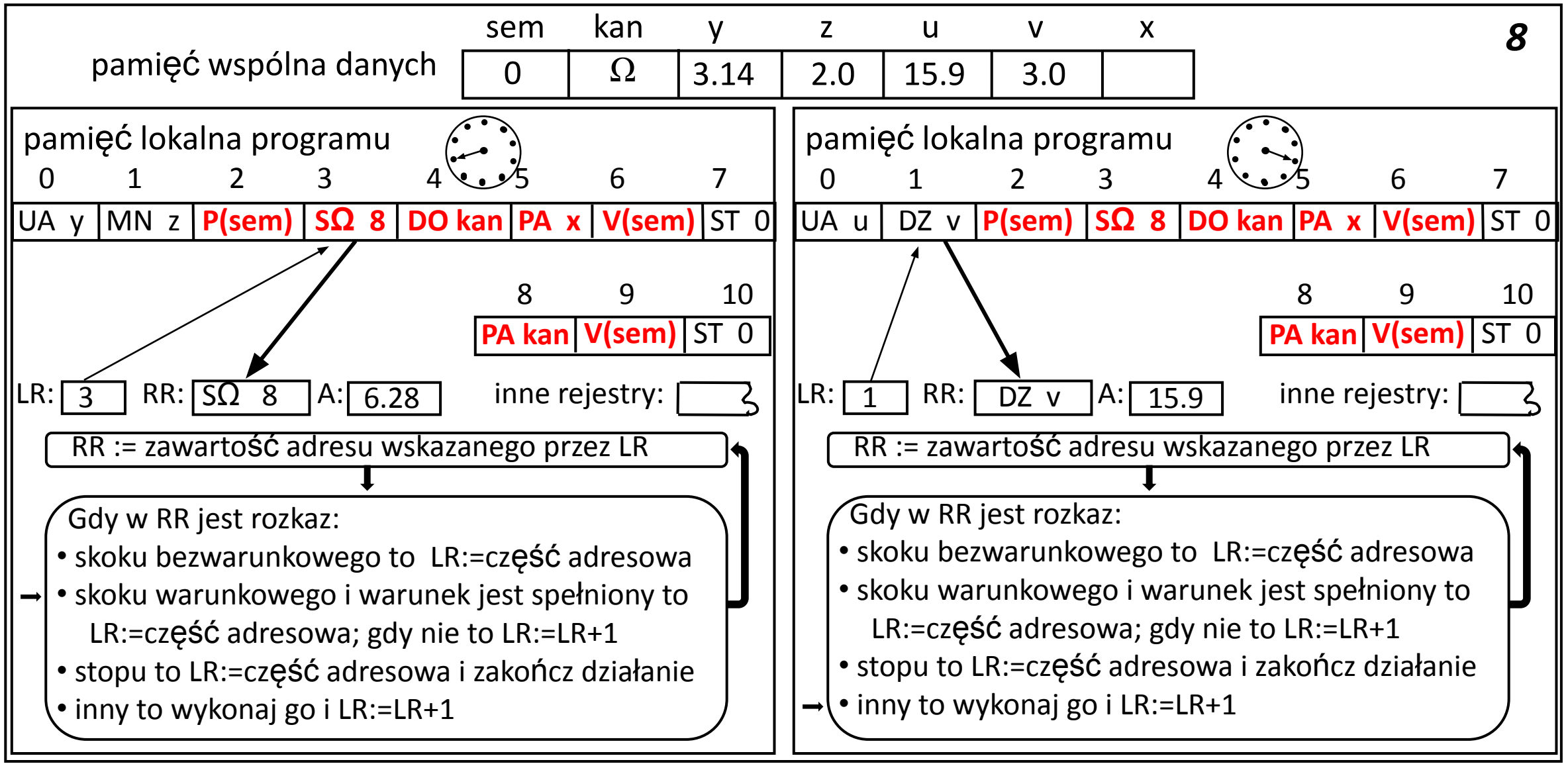
komputer 1

komputer 2



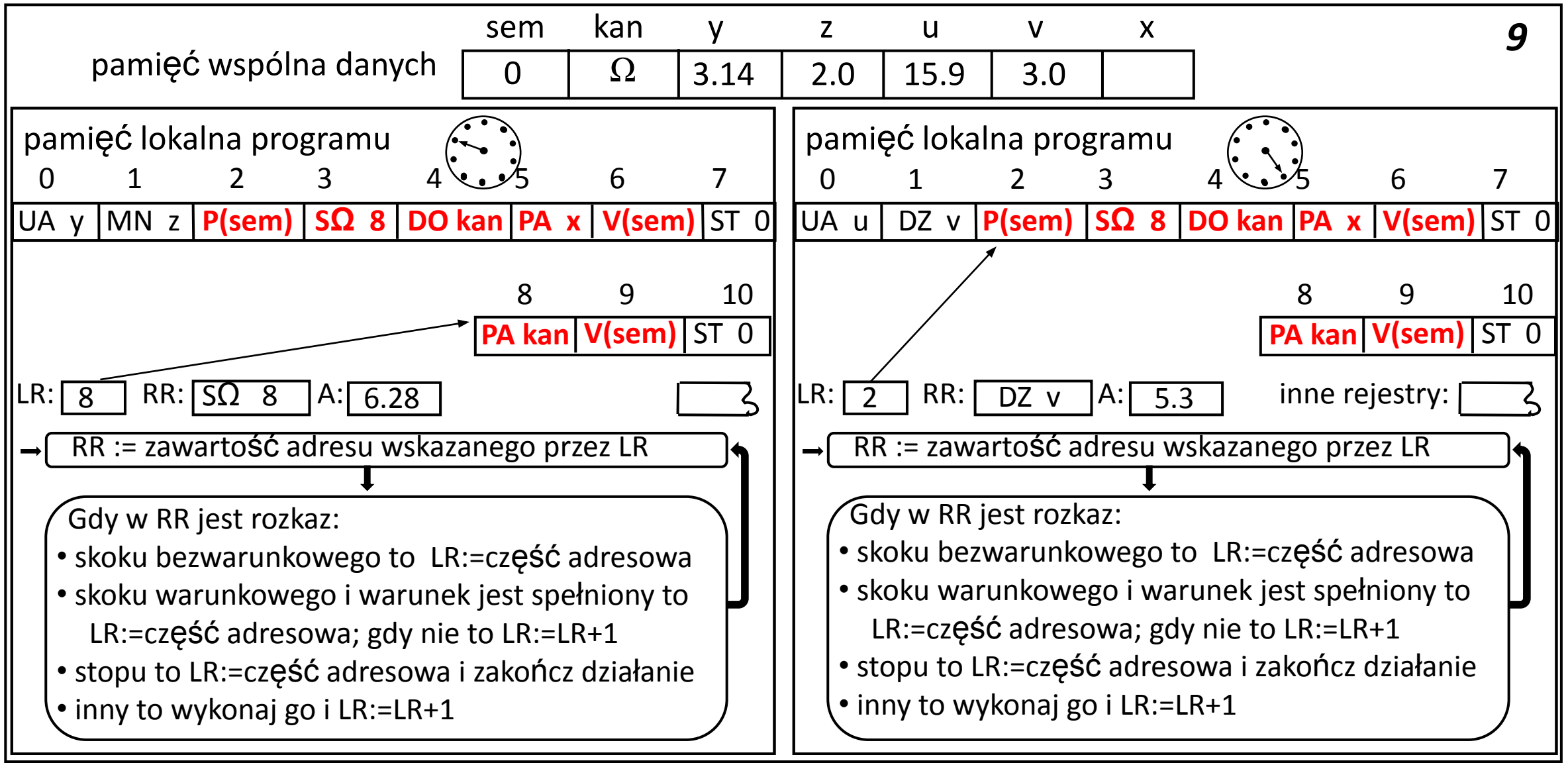
komputer 1

komputer 2



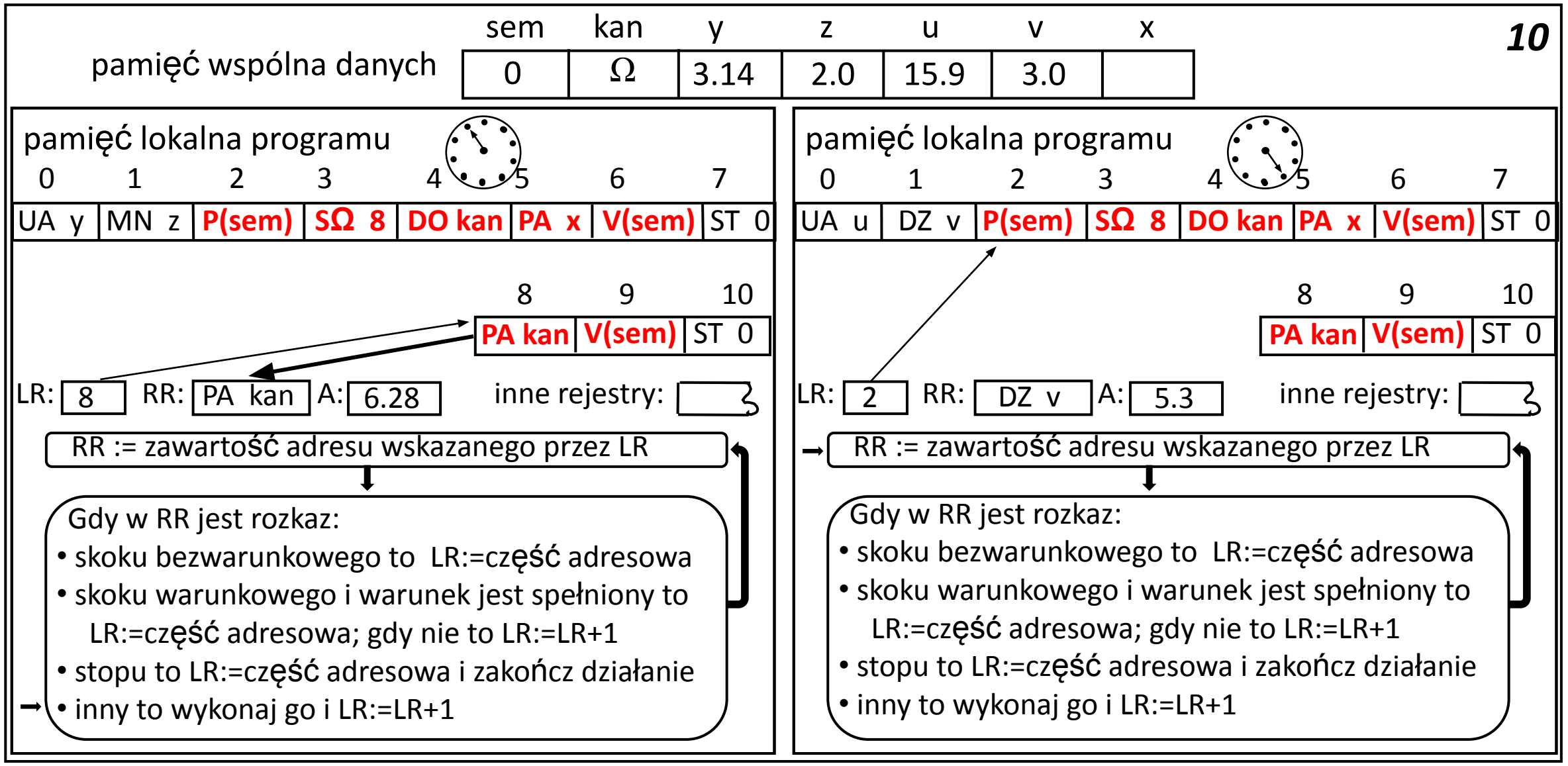
komputer 1

komputer 2



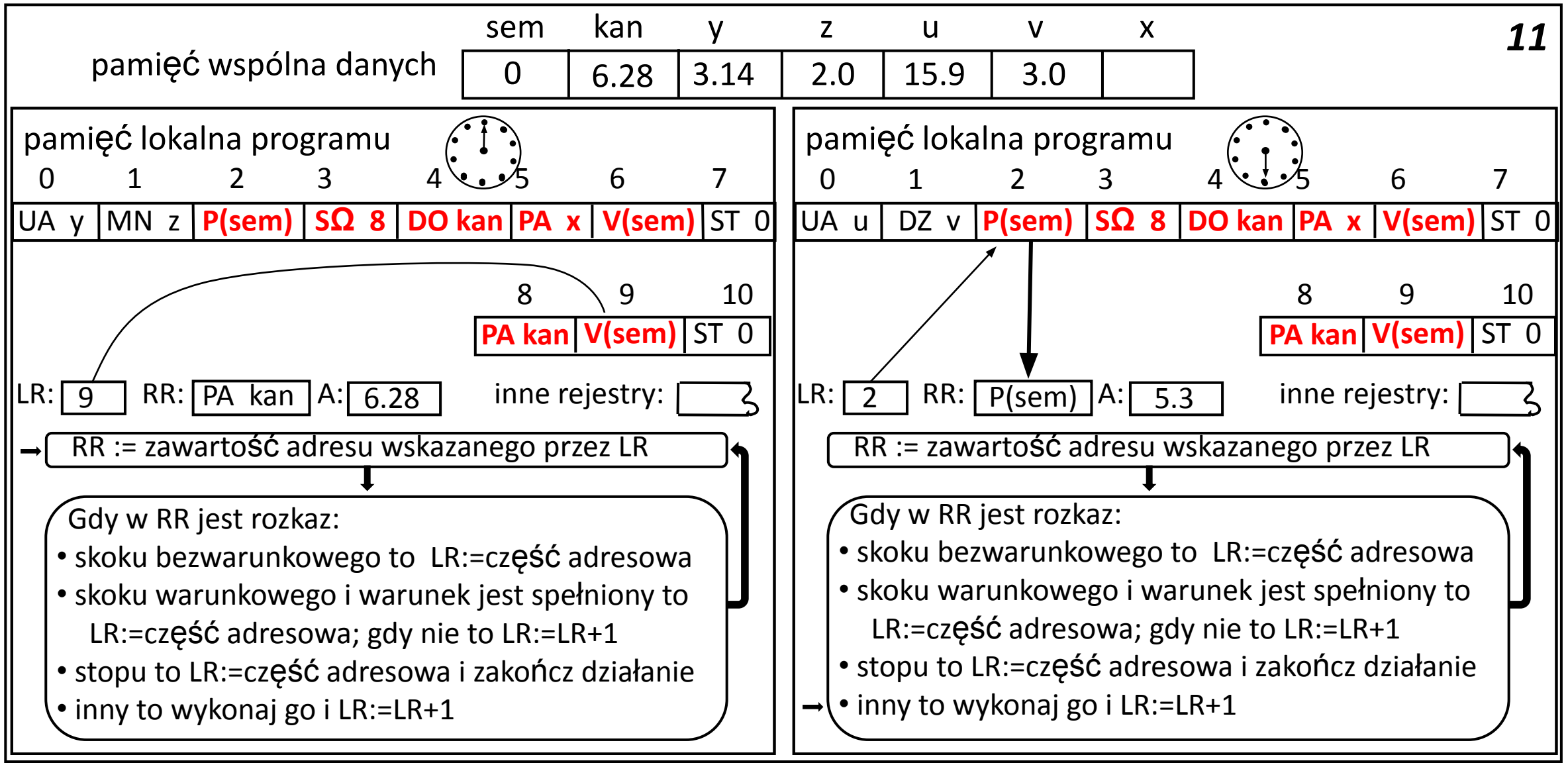
komputer 1

komputer 2



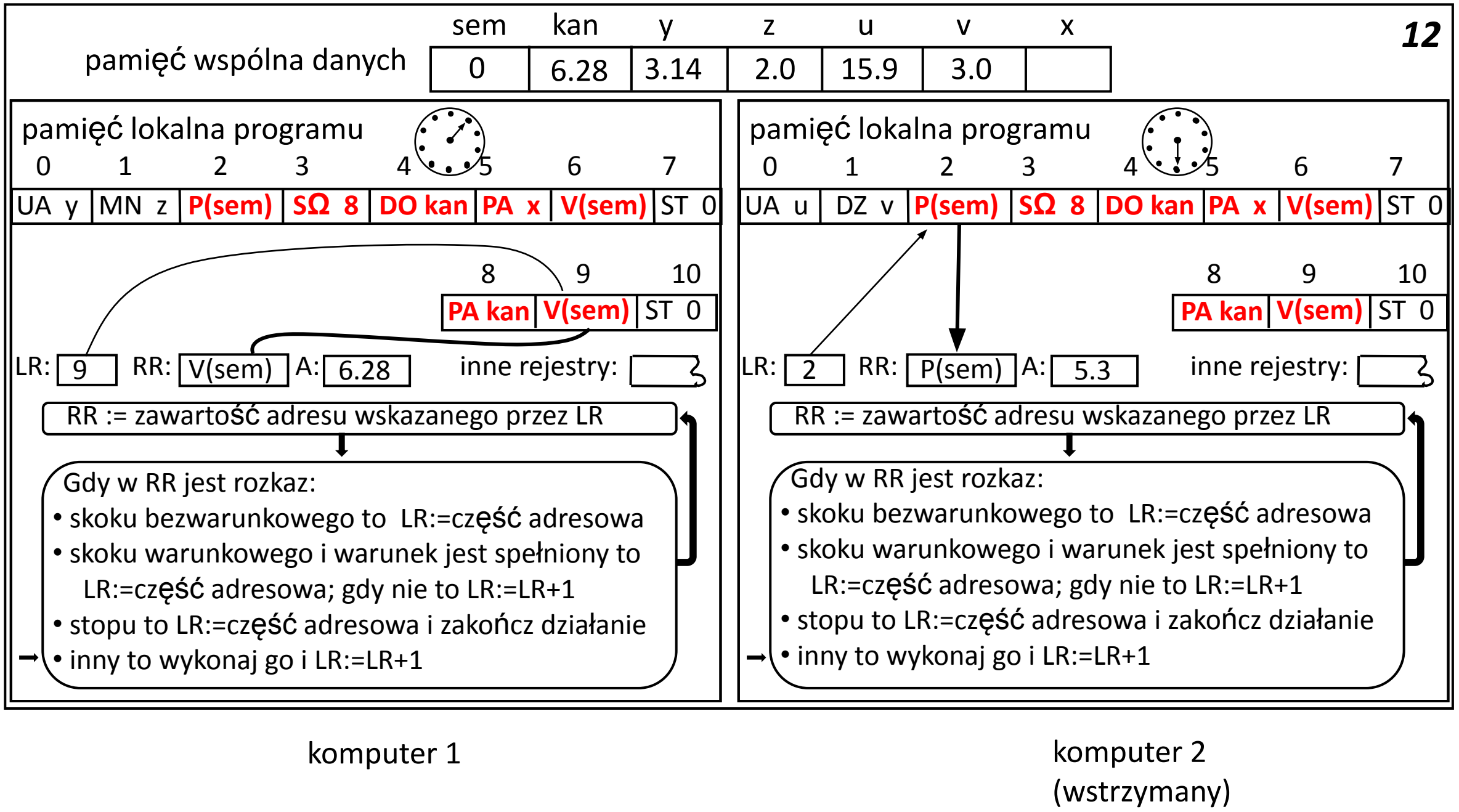
komputer 1

komputer 2



komputer 1

komputer 2
(wstrzymany)



komputer 1

komputer 2
(wstrzymany)

zmienia komputer 1

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 10 RR: V(sem) A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

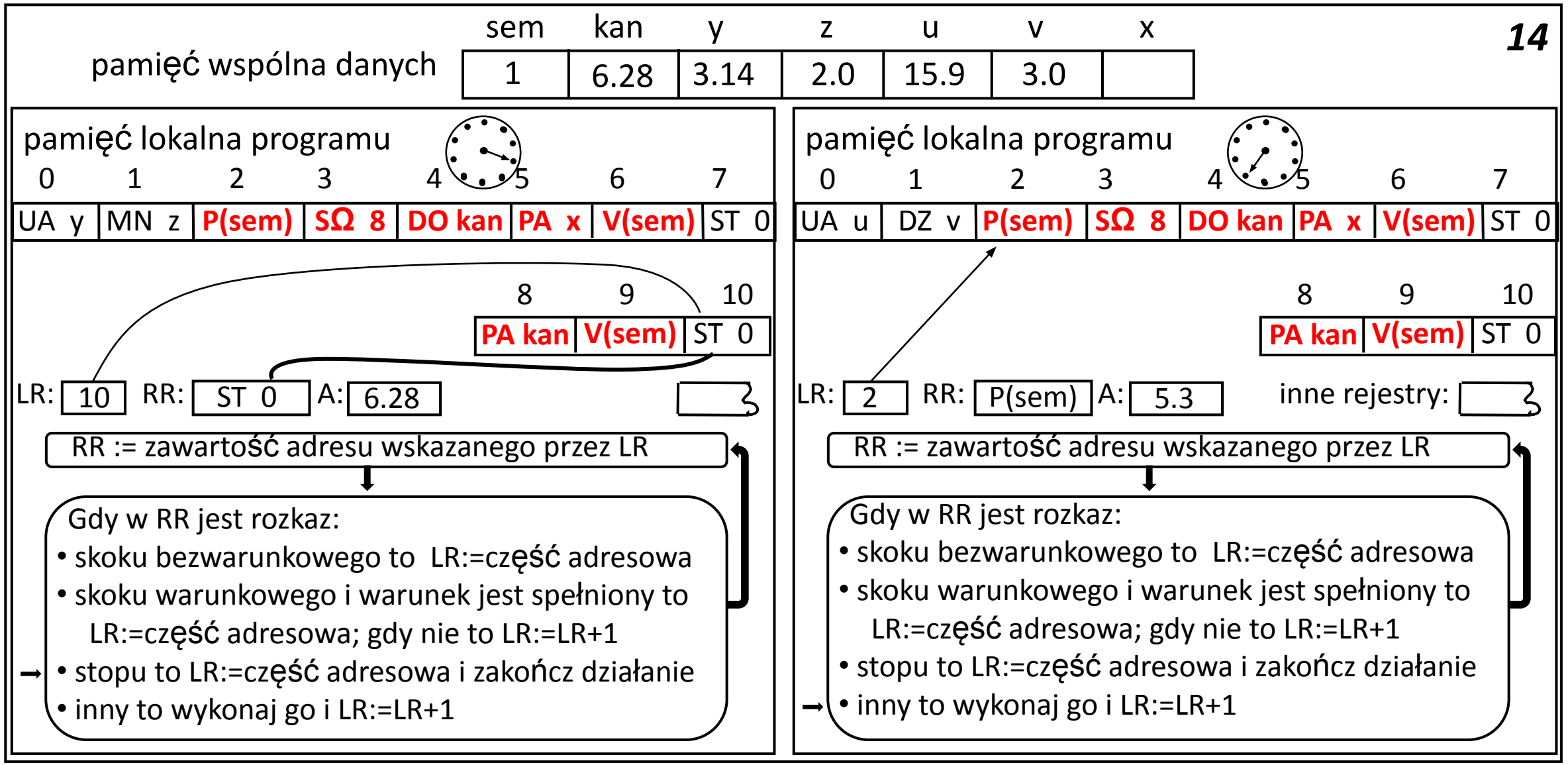
LR: 2 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 1

komputer 2
(wstrzymany)



komputer 1

komputer 2
(wstrzymany)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
0	6.28	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 3 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 1
(wstrzymany)

komputer 2
(wznowiony)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
0	6.28	3.14	2.0	15.9	3.0	

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

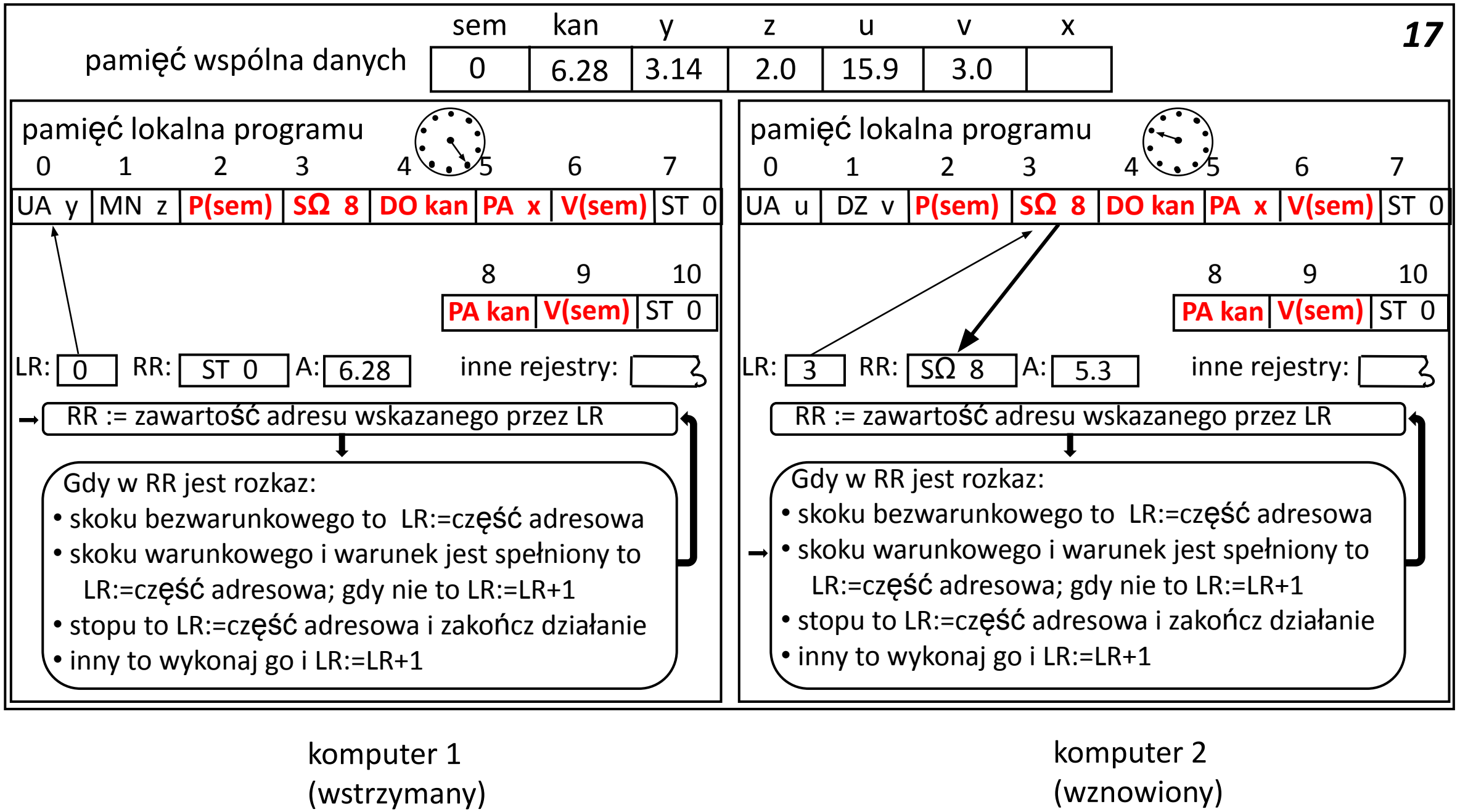
LR: 3 RR: P(sem) A: 5.3 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

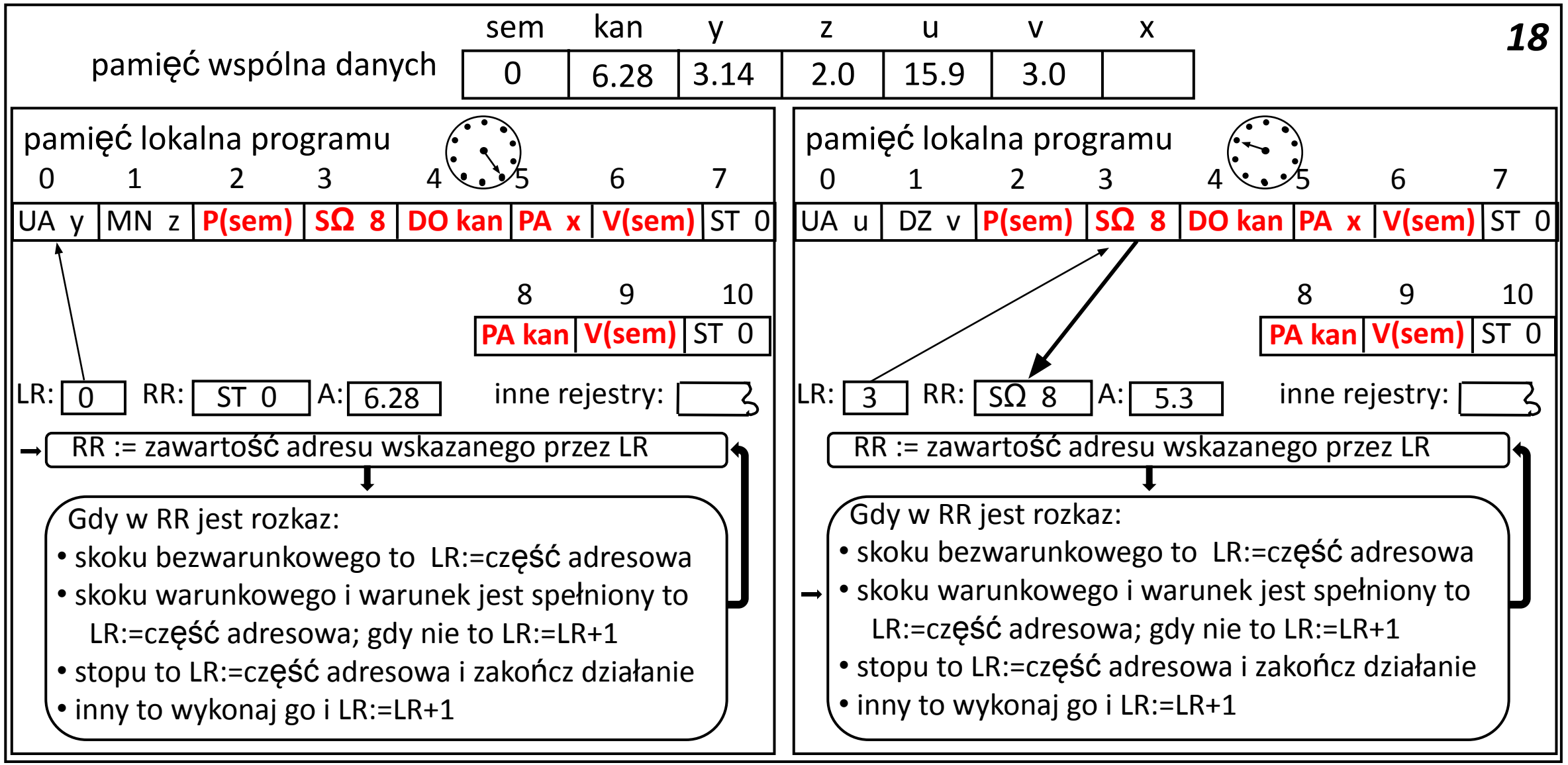
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



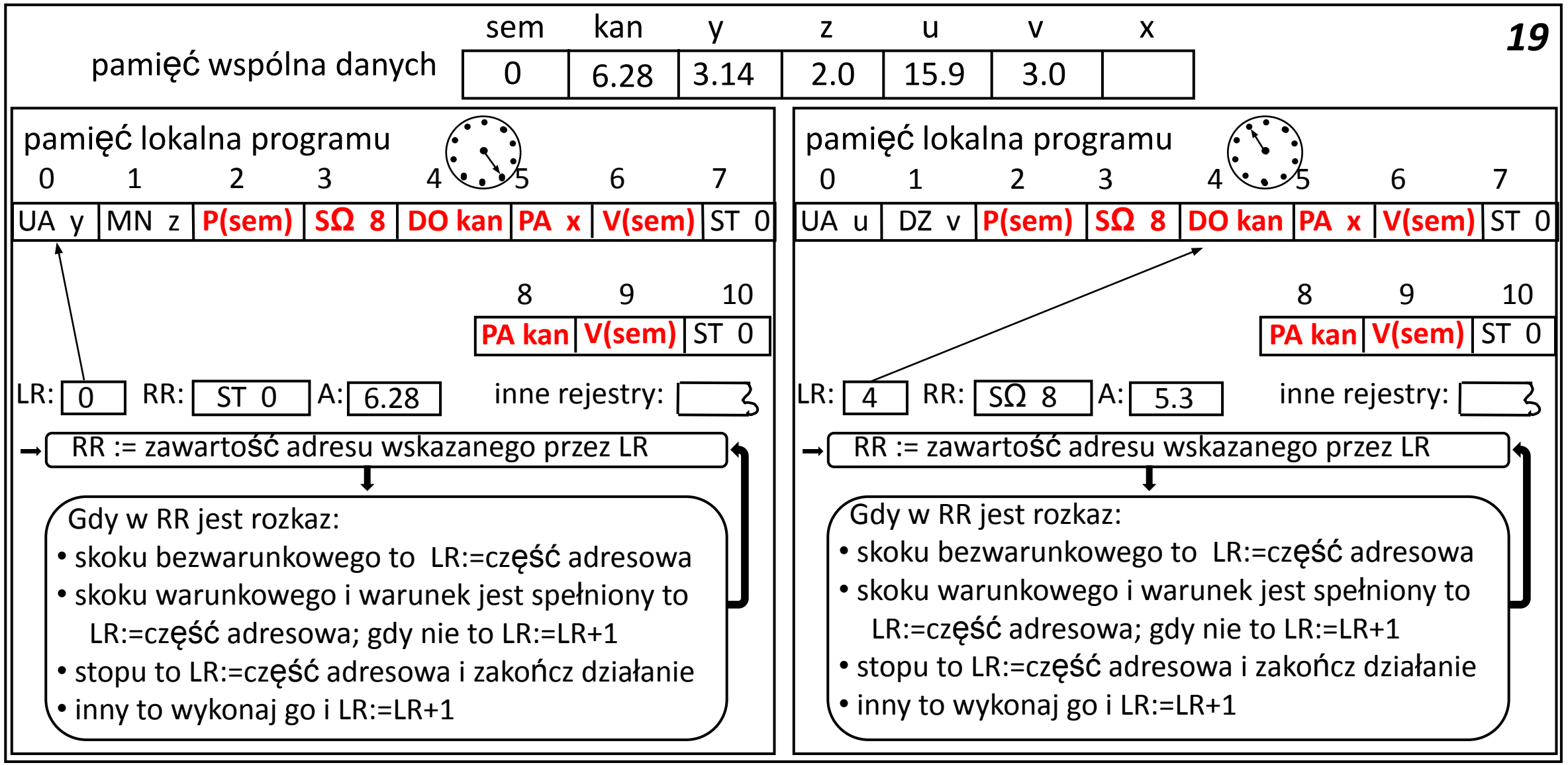
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



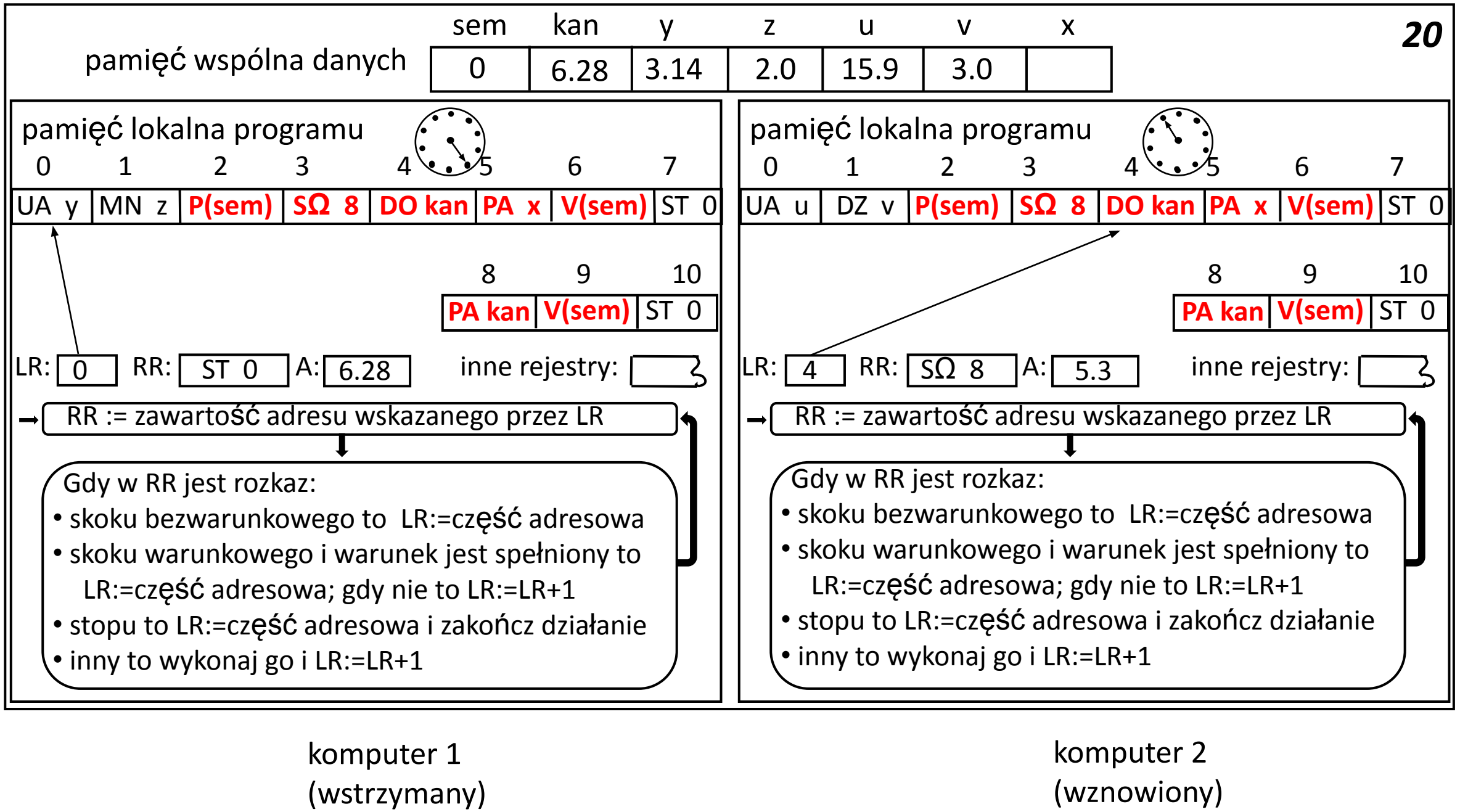
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



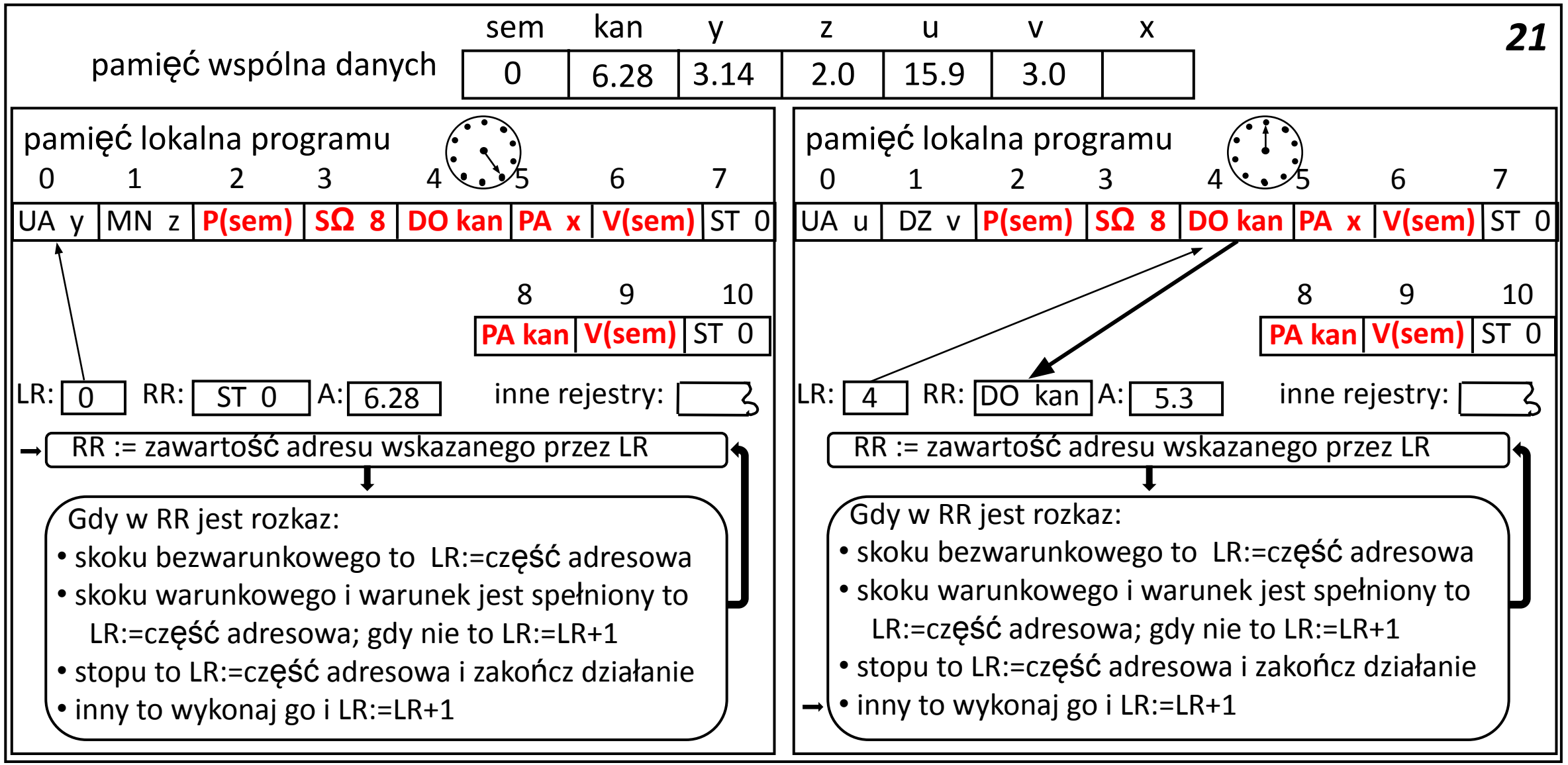
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



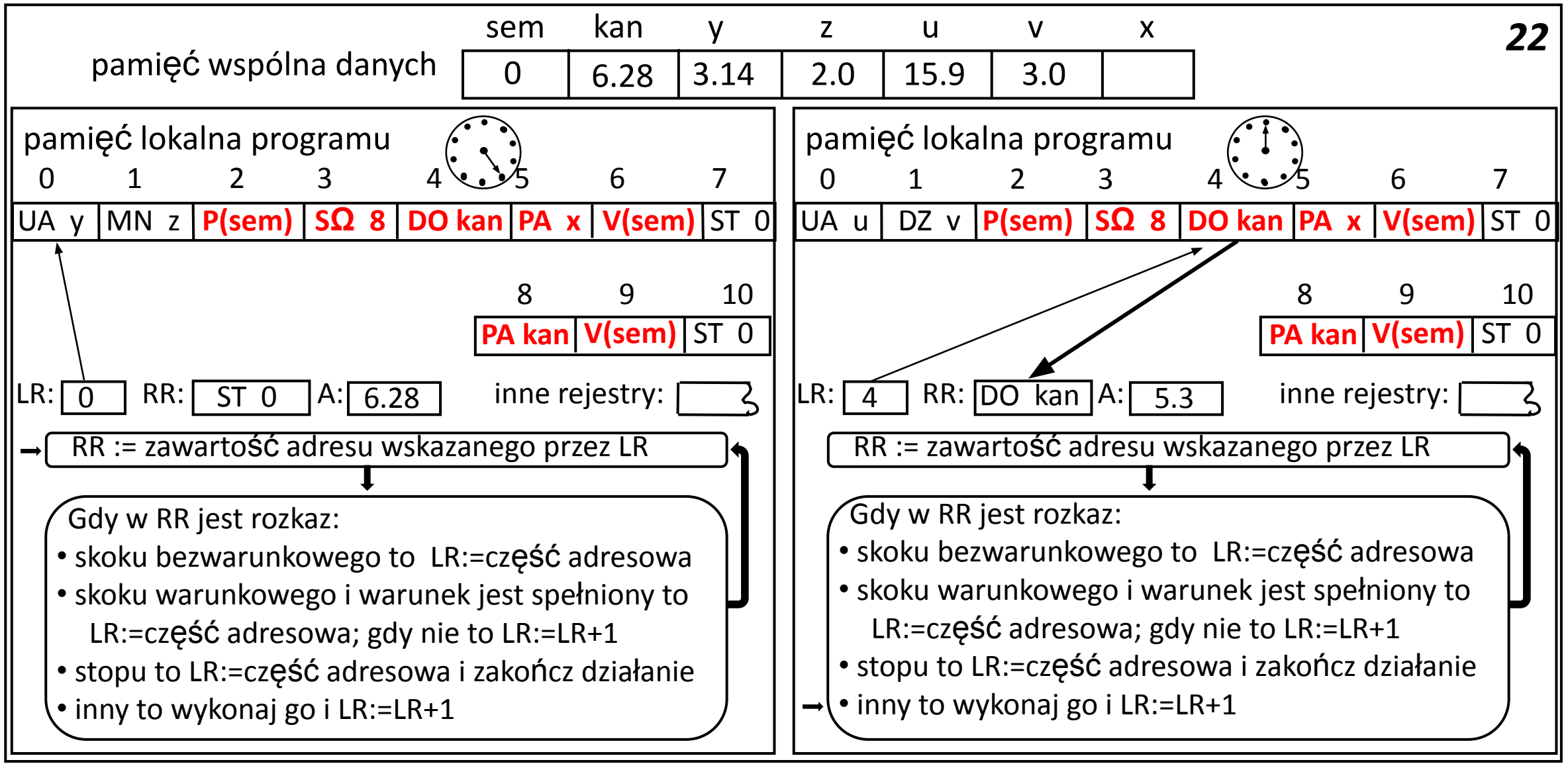
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



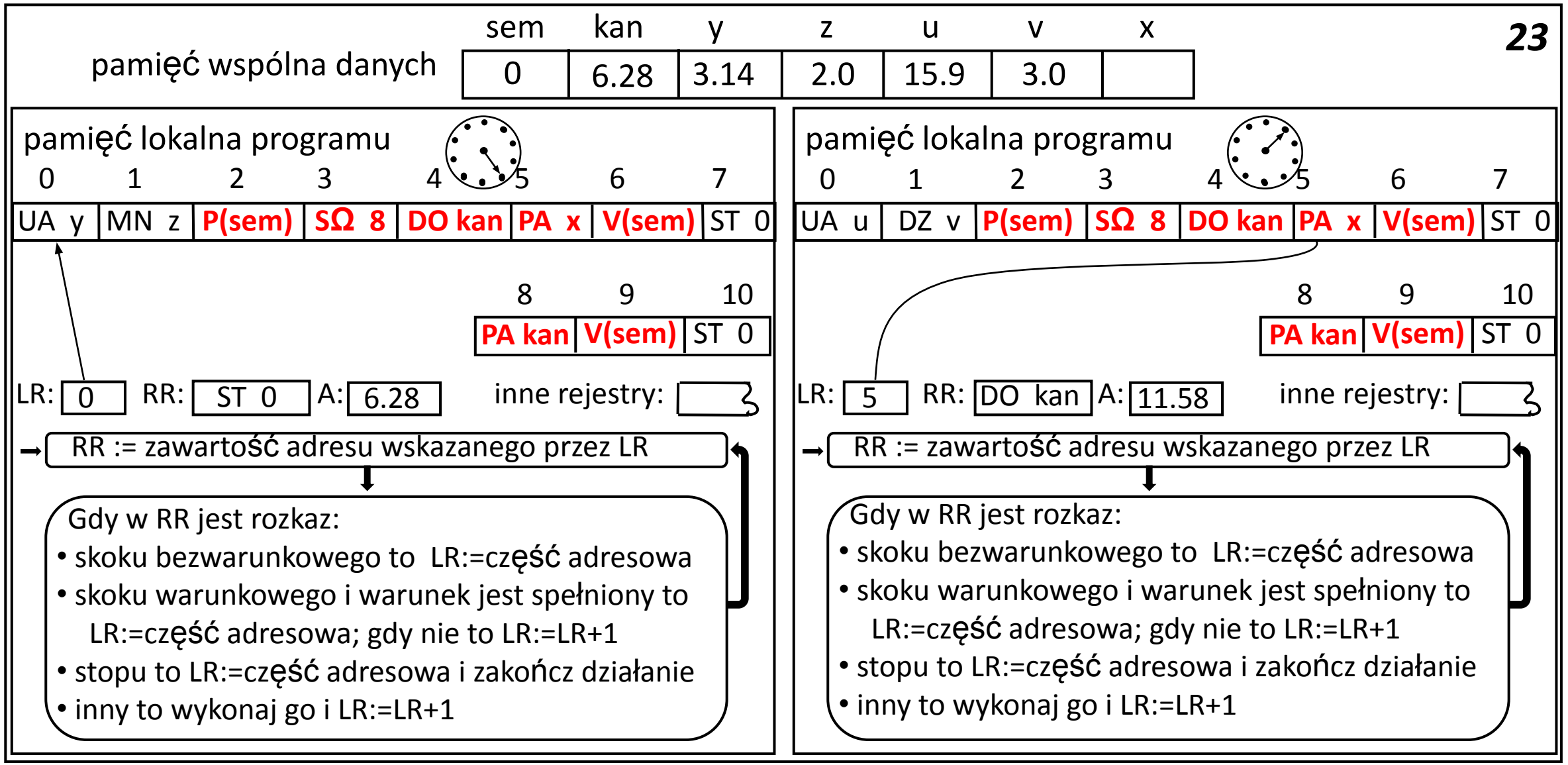
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



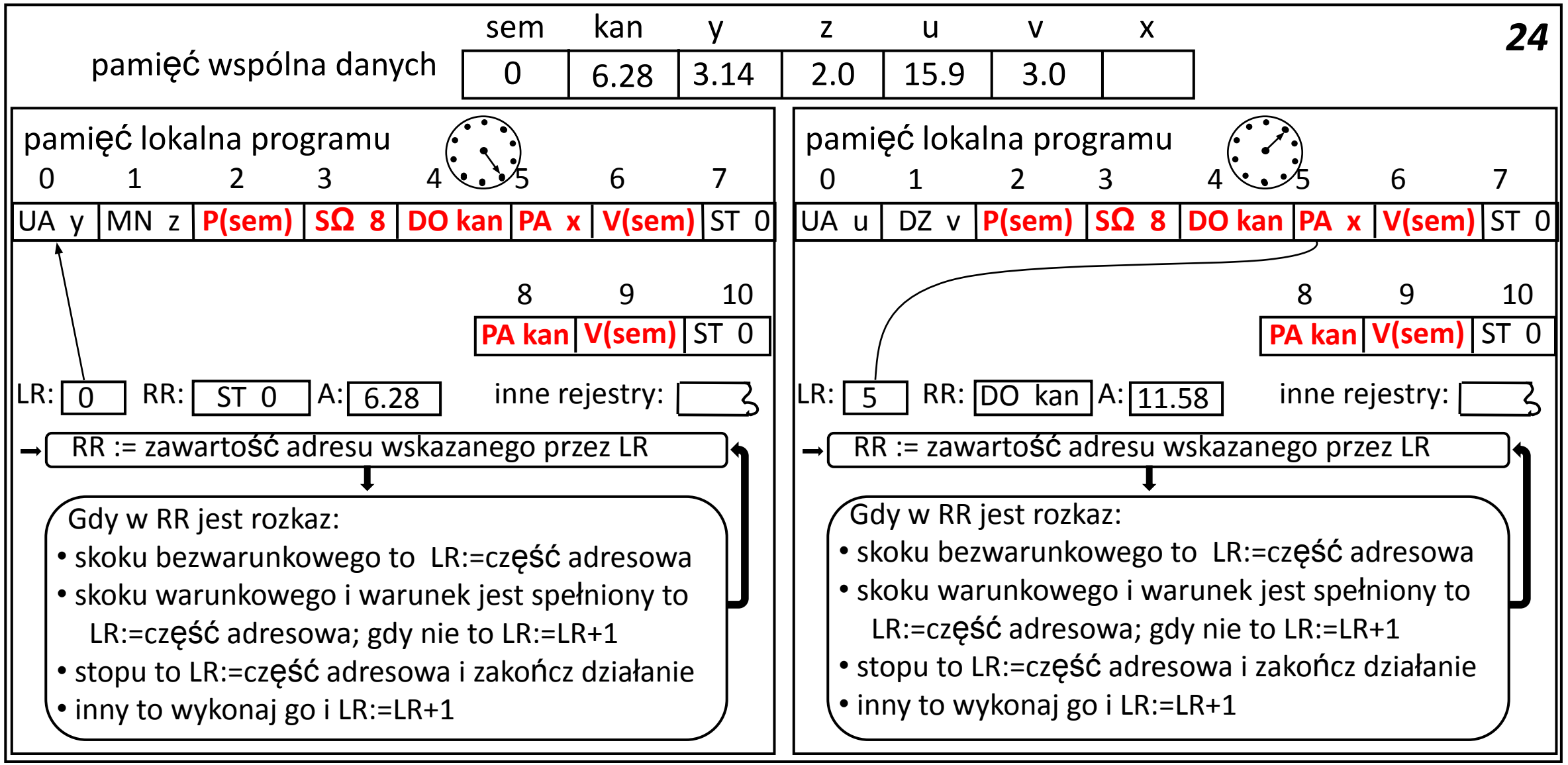
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



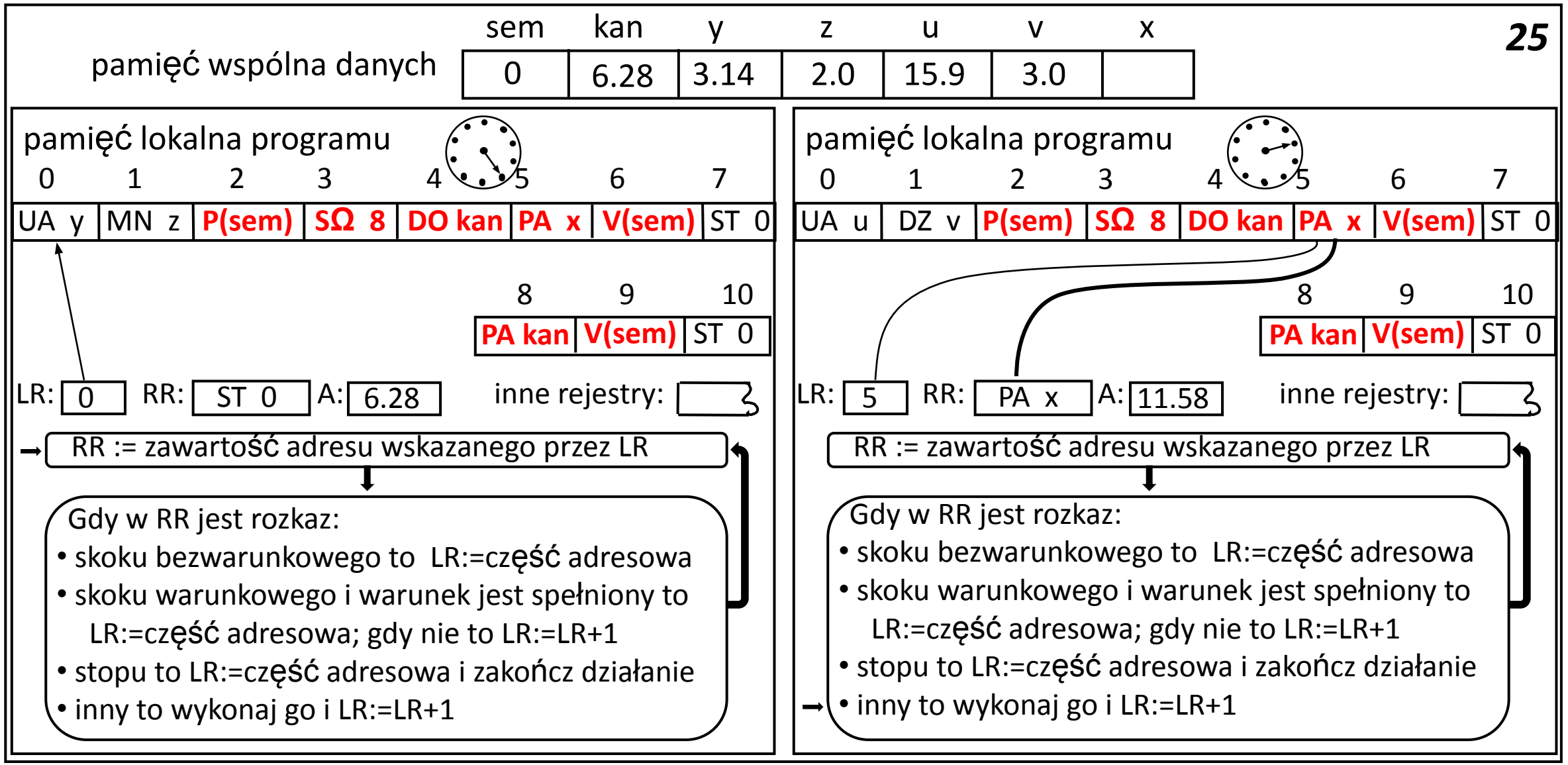
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



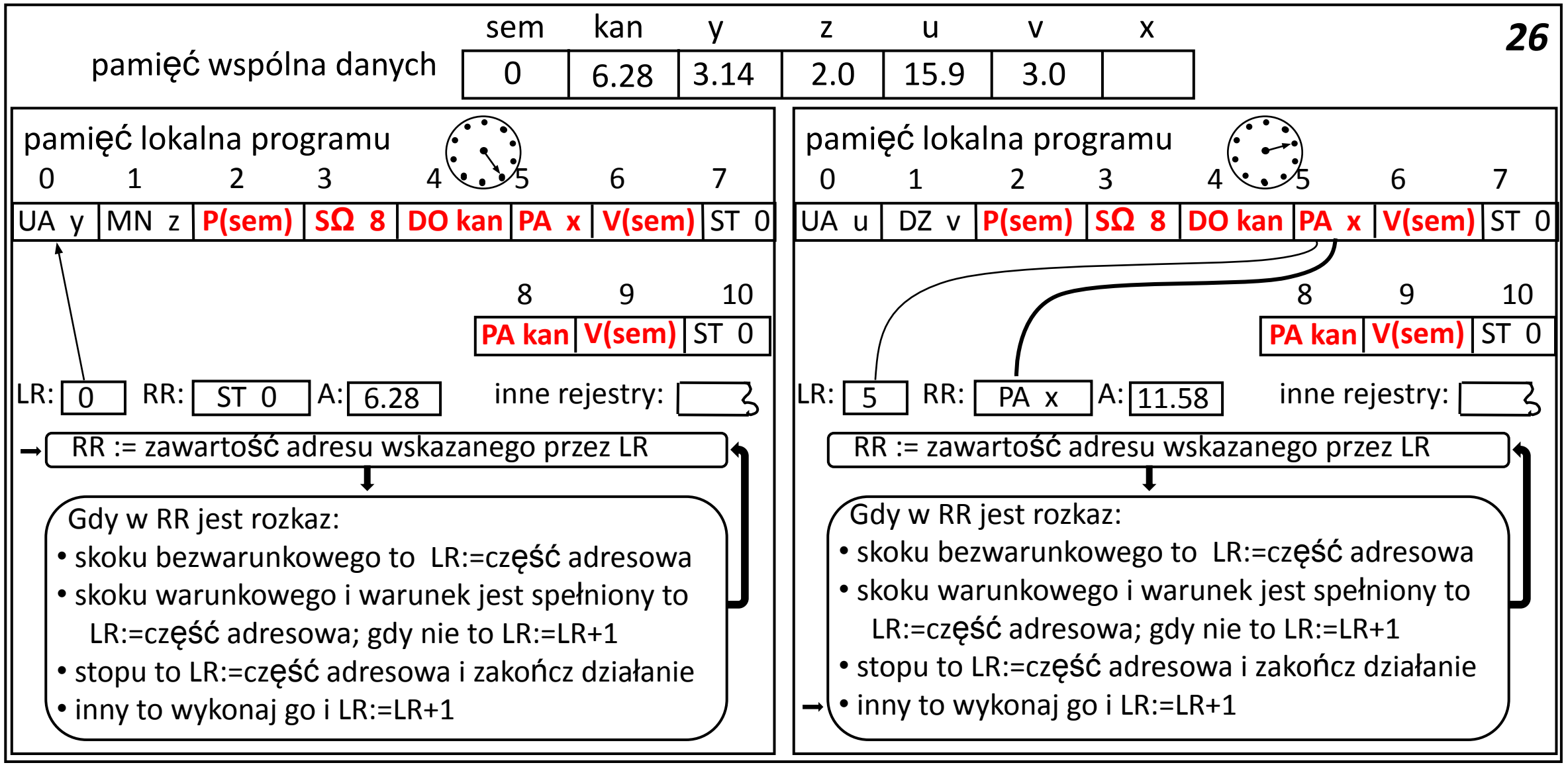
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



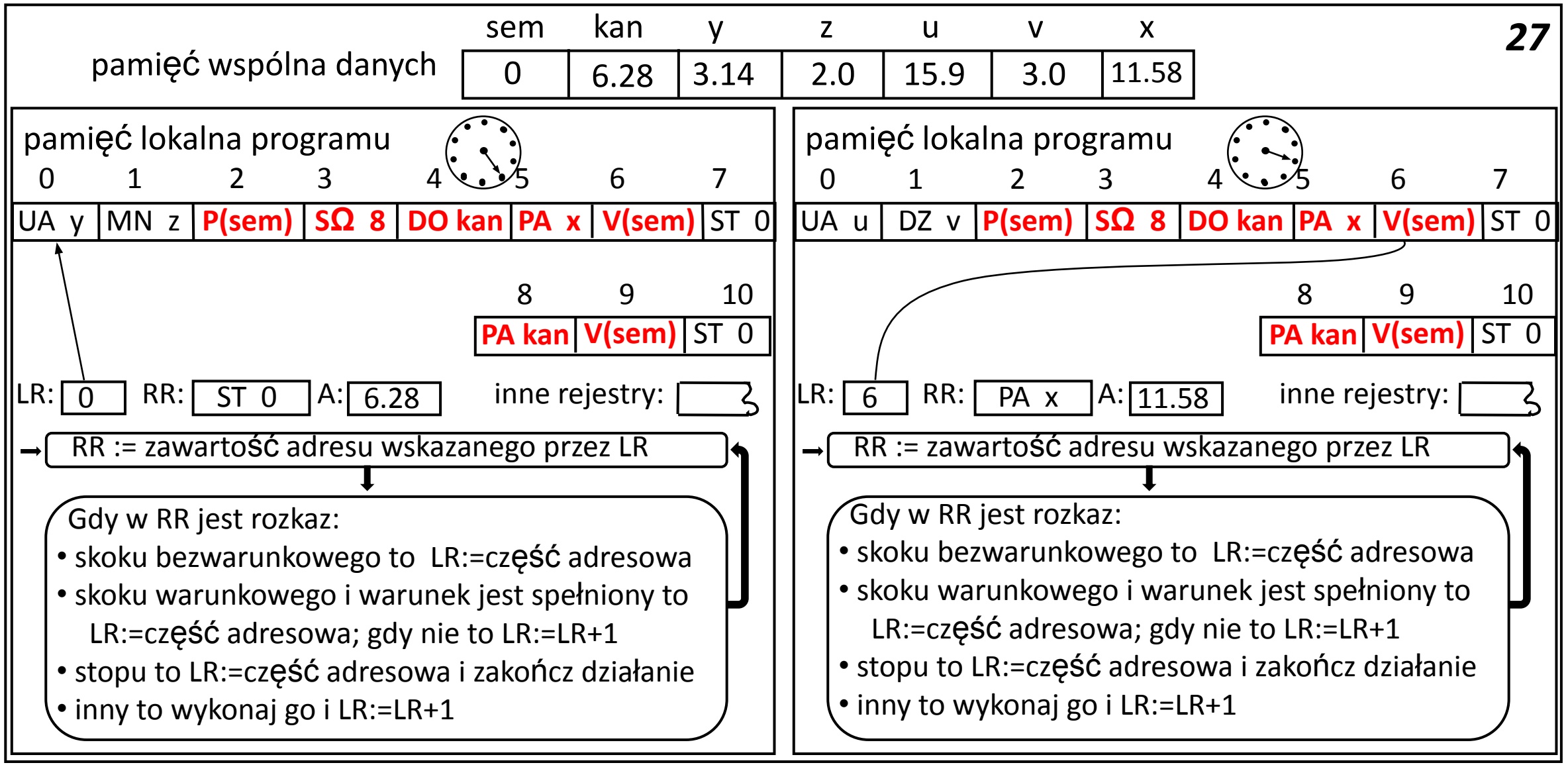
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



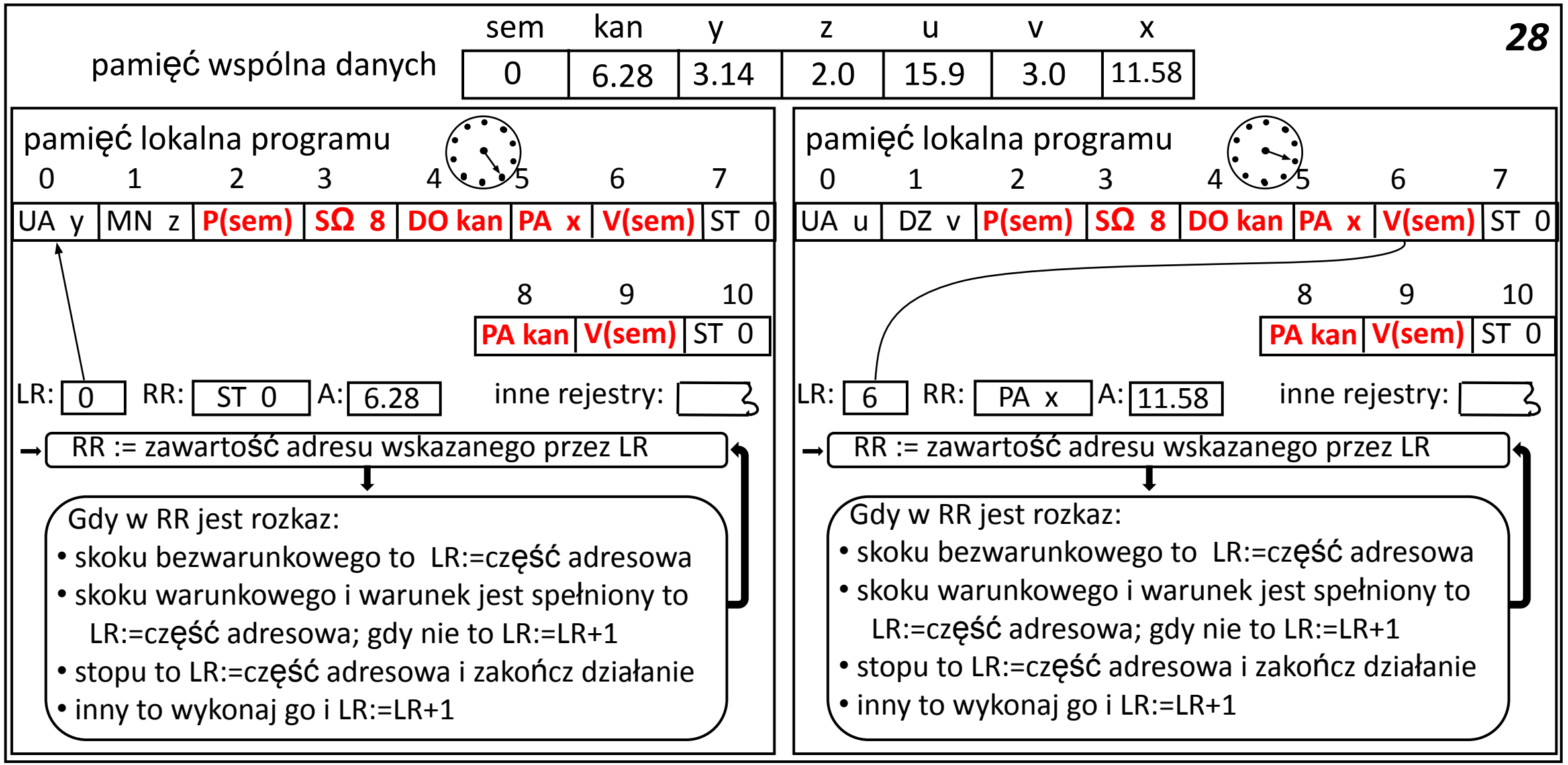
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



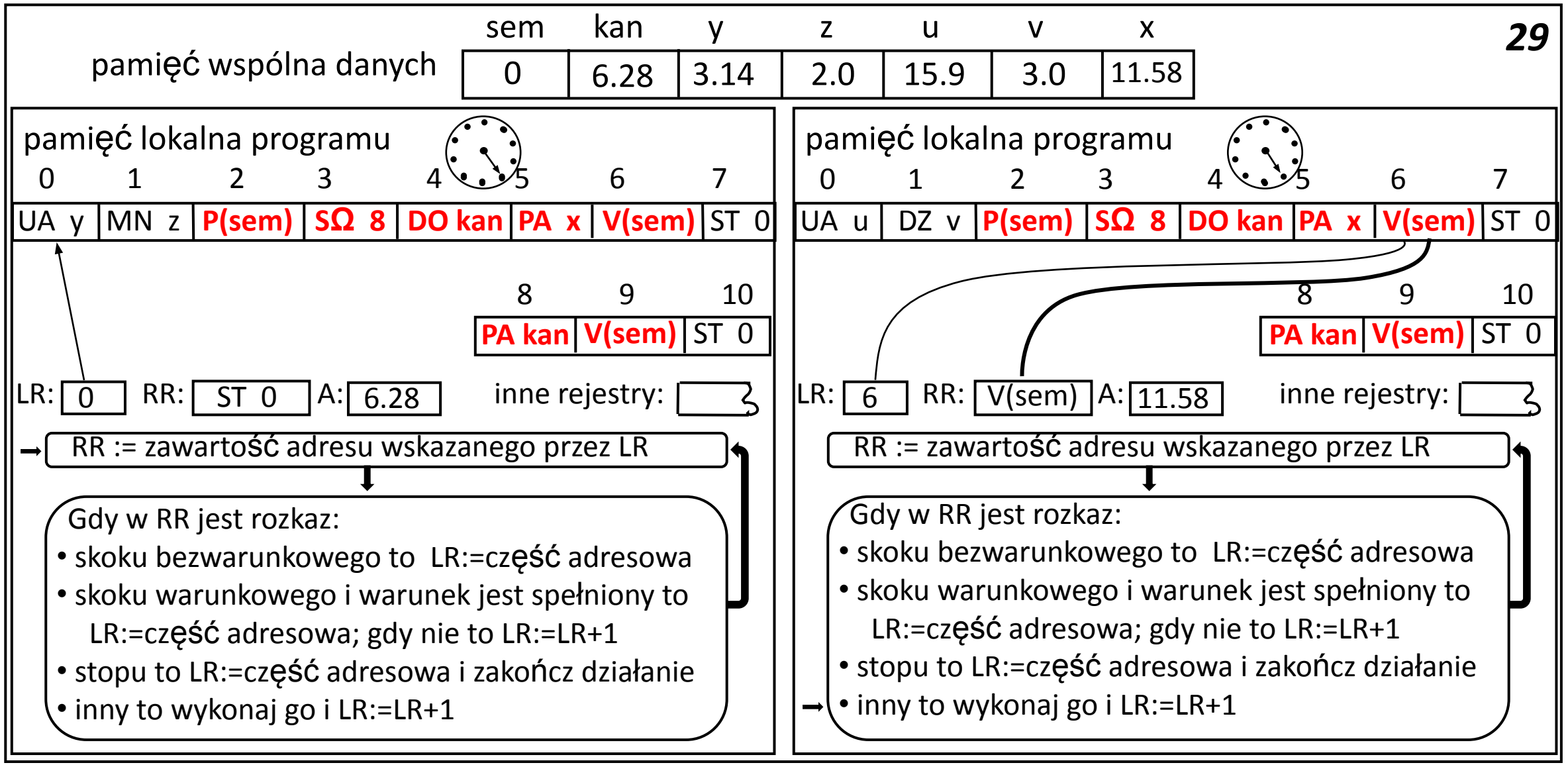
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



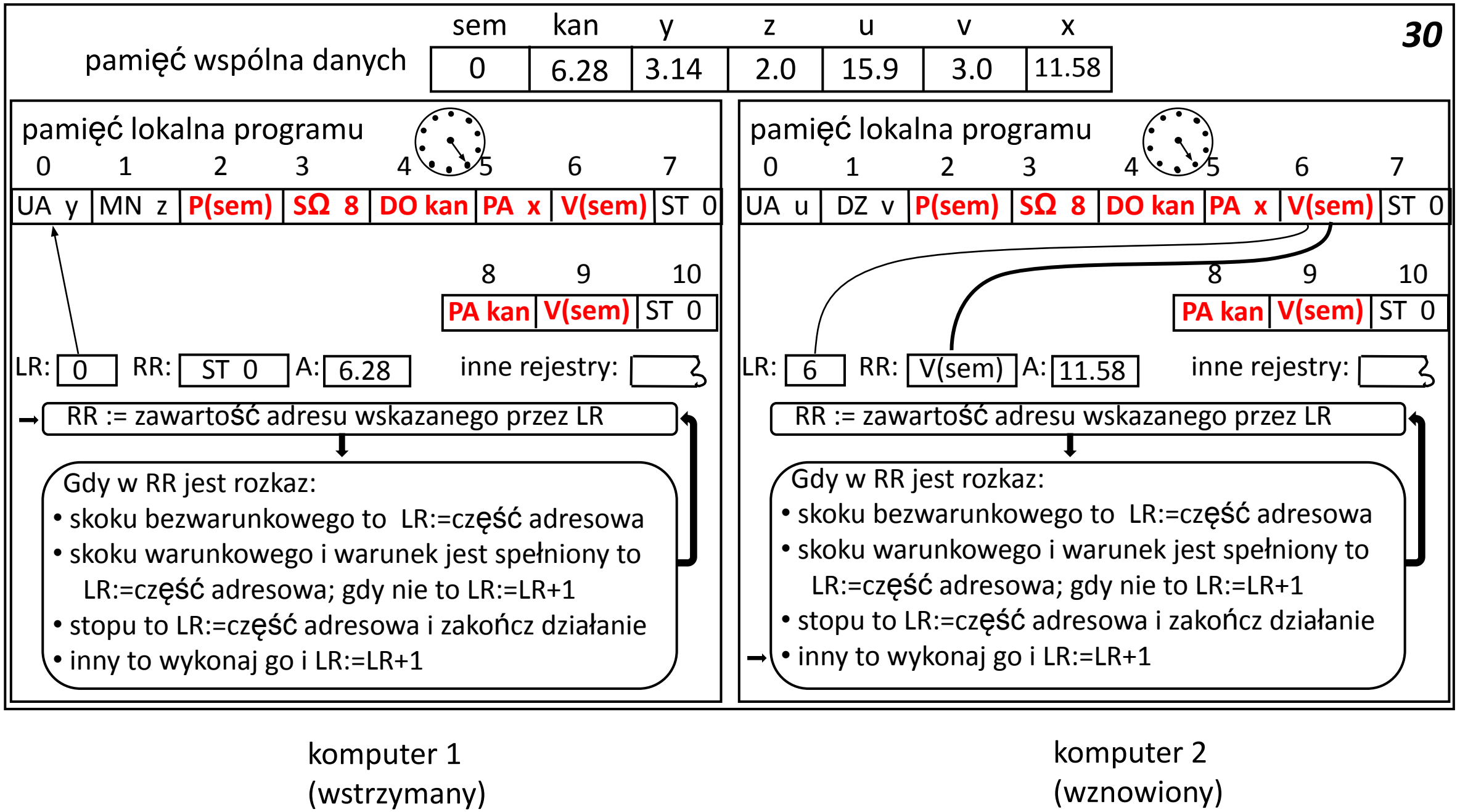
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wstrzymany)

komputer 2
(wznowiony)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	11.58

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 7 RR: V(sem) A: 11.58 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 1
(wstrzymany)

komputer 2
(wznowiony)

zmienia komputer 2

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	11.58

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

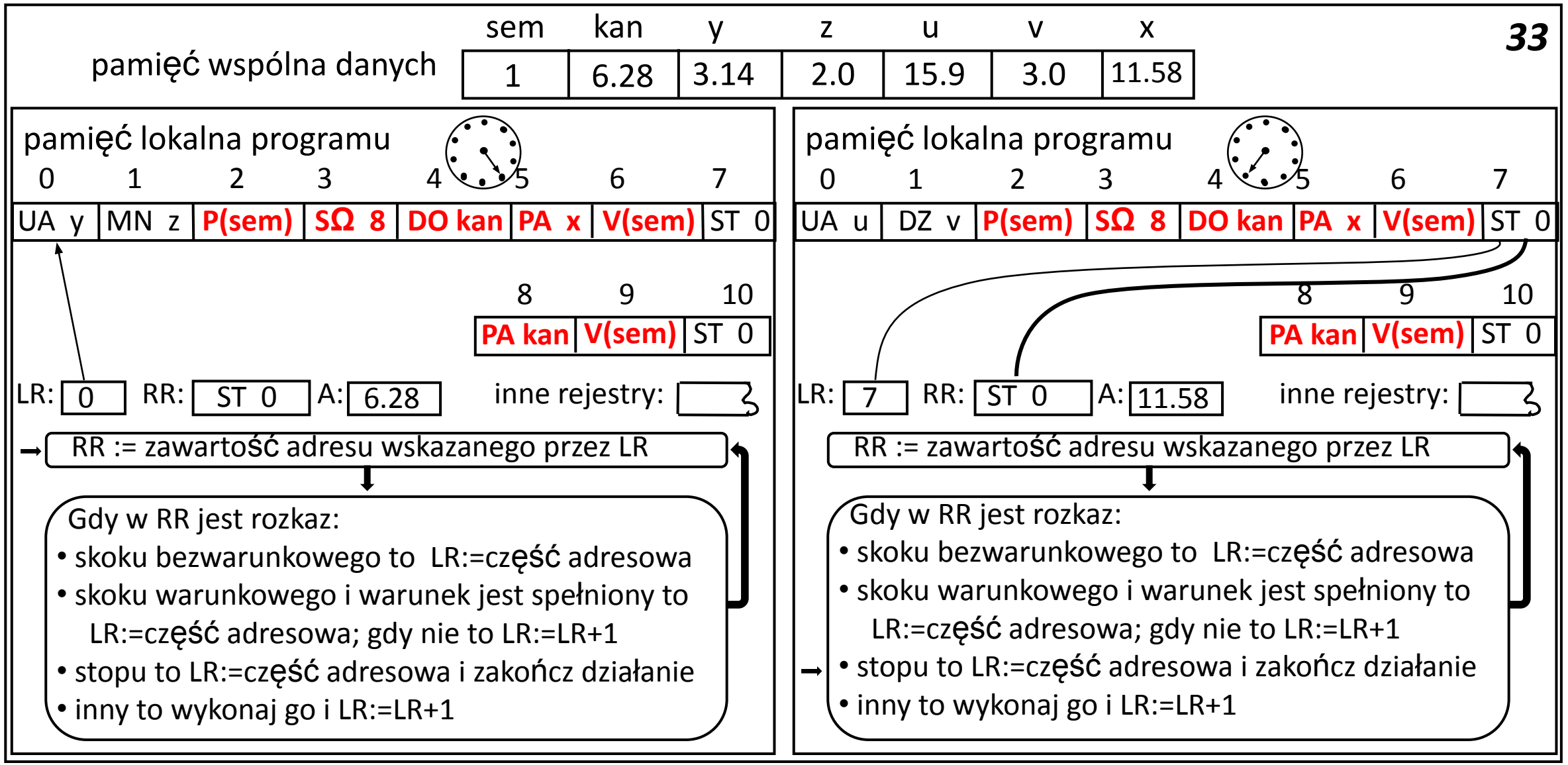
LR: 7 RR: V(sem) A: 11.58 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

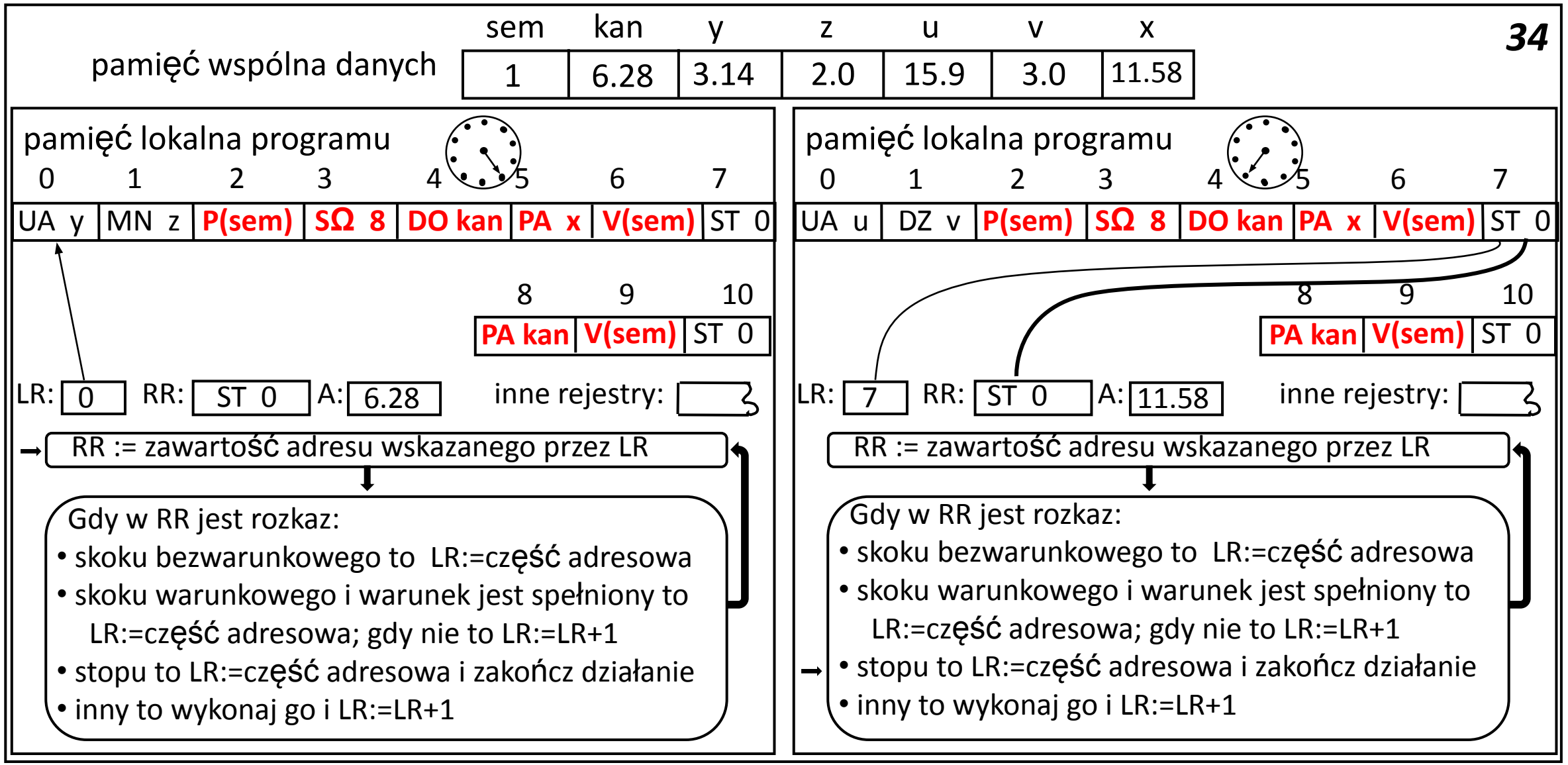
komputer 1
(wstrzymany)

komputer 2
(wznowiony)



komputer 1
(wznowiony)

komputer 2
(wznowiony)



komputer 1
(wznowiony)

komputer 2
(wznowiony)

pamięć wspólna danych

sem	kan	y	z	u	v	x
1	6.28	3.14	2.0	15.9	3.0	11.58

pamięć lokalna programu

0	1	2	3	4	5	6	7
UA y	MN z	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

8	9	10
PA kan	V(sem)	ST 0

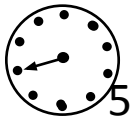
LR: 0 RR: ST 0 A: 6.28 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 1
(wznowiony)

pamięć lokalna programu



0	1	2	3	4	5	6	7
UA u	DZ v	P(sem)	SΩ 8	DO kan	PA x	V(sem)	ST 0

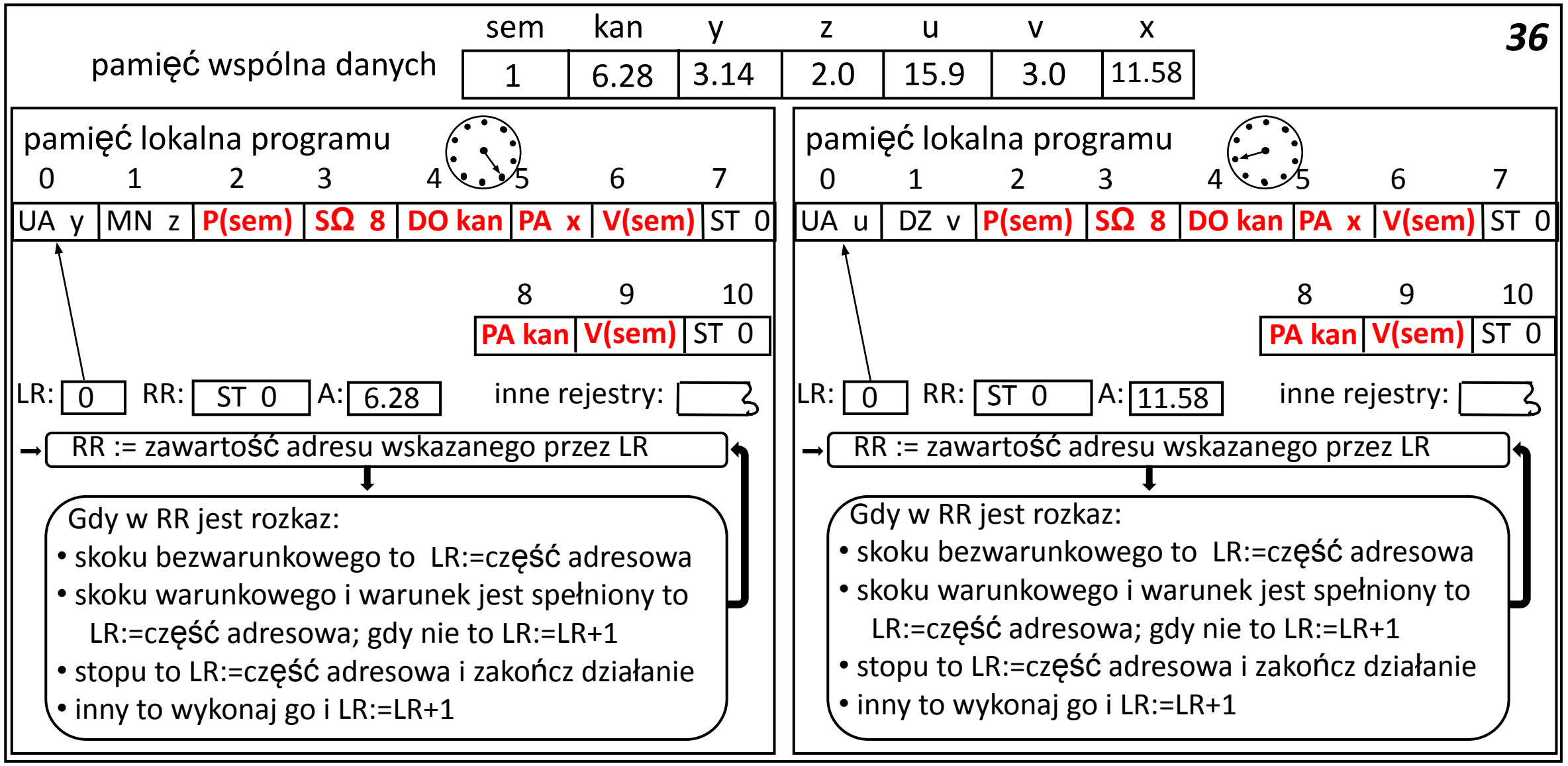
8	9	10
PA kan	V(sem)	ST 0

LR: 0 RR: ST 0 A: 11.58 inne rejestry:

RR := zawartość adresu wskazanego przez LR

Gdy w RR jest rozkaz:
- skoku bezwarunkowego to LR:=część adresowa
- skoku warunkowego i warunek jest spełniony to LR:=część adresowa; gdy nie to LR:=LR+1
- stopu to LR:=część adresowa i zakończ działanie
- inny to wykonaj go i LR:=LR+1

komputer 2
(wznowiony)



komputer 1
(wznowiony)

komputer 2
(wznowiony)