Materiały do wykładu

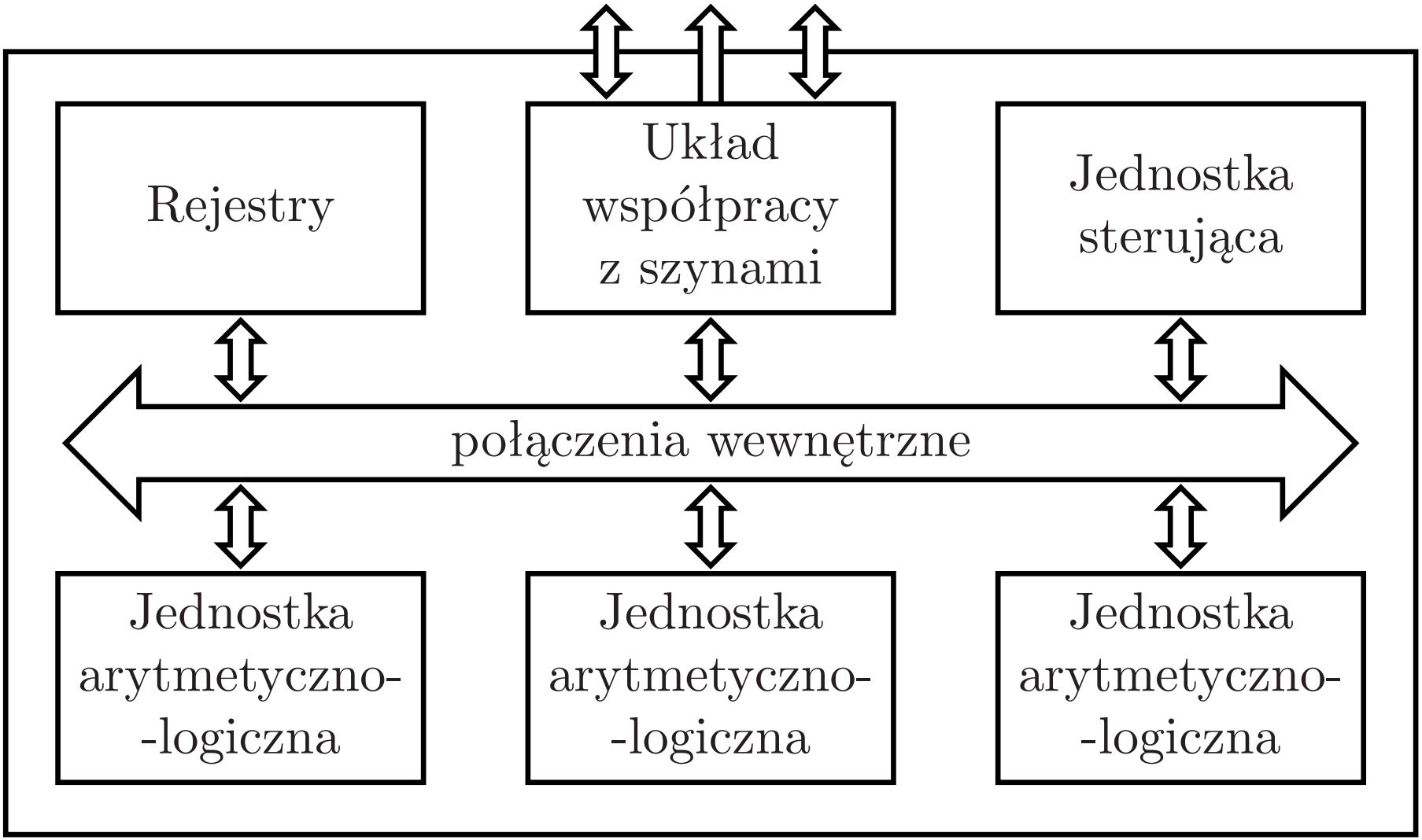
4. Mikroprocesor

Marcin Peczarski

Instytut Informatyki
Uniwersytet Warszawski

12 marca 2013

Architektura mikroprocesora



Kod maszynowy a asembler

- **kod maszynowy** – binarny sposób opisu rozkazów procesora
  - Każda architektura ma swój unikalny kod maszynowy.
  - W obrębie tej samej architektury kod maszynowy może być nieco różny dla poszczególnych modeli procesorów.
- **asembler** – język programowania niskiego poziomu
  - Jedno polecenie odpowiada zwykle jednemu rozkazowi maszynowemu.
  - Każda architektura ma swój unikalny asembler.
- **asembler** – program tłumaczący asembler na kod maszynowy
- **GNU Assembler** – nazwa własna konkretnego asemblera

Dlaczego Z80?

- Jeden z najpopularniejszych mikroprocesorów 8-bitowych
- Nadal produkowany i używany
- Bezpośredni wpływ na najpopularniejszą obecnie architekturę x86
- Kompromisy projektowe, kompatybilność z 8080
- Wystarczająco prosty

Z80 – rejestry

A	F
B	C
D	E
H	L

A’	F’
B’	C’
D’	E’
H’	L’

PC
SP
IX
IY

I
IFF
R

Z80 – przesyłanie danych 8-bitowych (1)

LD d, s

0 1	d	s

d, s	argument
000	B
001	C
010	D
011	E
100	H
101	L
110	(HL)
111	A

▷ Nie ma rozkazu LD (HL), (HL); kod 01110110 to rozkaz HALT!

Z80 – przesyłanie danych 8-bitowych (2)

LD d, n

0 0	d	1 1 0
n		

d	argument
000	B
001	C
010	D
011	E
100	H
101	L
110	(HL)
111	A

Z80 – przesyłanie danych 8-bitowych (3)

LD (p), A

LD A, (p)

0 0 0	p	d	0 1 0

LD (aa), A

LD A, (aa)

0 0 1 1	d	0 1 0
aal		
aah		

p	para rejestrów
0	BC
1	DE

d	kierunek
0	z akumulatora do pamięci
1	z pamięci do akumulatora

Z80 – rejestry indeksowe (1)

LD r, (HL)

0 1	r	1 1 0

LD r, (i + d)

1 1	i	1	1	1	0	1	
0 1	r			1	1	0	
d							

r	rejestr
000	B
001	C
010	D
011	E
100	H
101	L
111	A

i	rejestr indeksowy
0	IX
1	IY

Z80 – rejestry indeksowe (2)

LD (HL), n

0 0	1 1 0	1 1 0
n		

LD (i + d), n

1 1	i	1 1 1 0 1	
0 0	1 1 0	1 1 0	
d			
n			

i	rejestr indeksowy
0	IX
1	IY

Z80 – rejestry primowane

EX AF, AF'

0	0	0	0	1	0	0	0

EXX

1	1	0	1	1	0	0	1

Z80 – rejestr znaczników

S	Z	–	AC	–	P/V	N	CY

- S – najstarszy bit wyniku operacji, wynik ujemny
- Z – wynik operacji zerowy
- AC – pomocnicze przeniesienie z pozycji 3, arytmetyka BCD
- P/V – znacznik parzystości dla operacji logicznych i nadmiaru (ang. *overflow*) dla operacji arytmetycznych
- N – znacznik zerowany przy wykonywaniu dodawania i ustawiany przy wykonywaniu odejmowania, arytmetyka BCD
- CY – znacznik przeniesienia (pożyczki przy odejmowaniu)

Z80 – 8-bitowe operacje arytmetyczno-logiczne

1 0	op	s

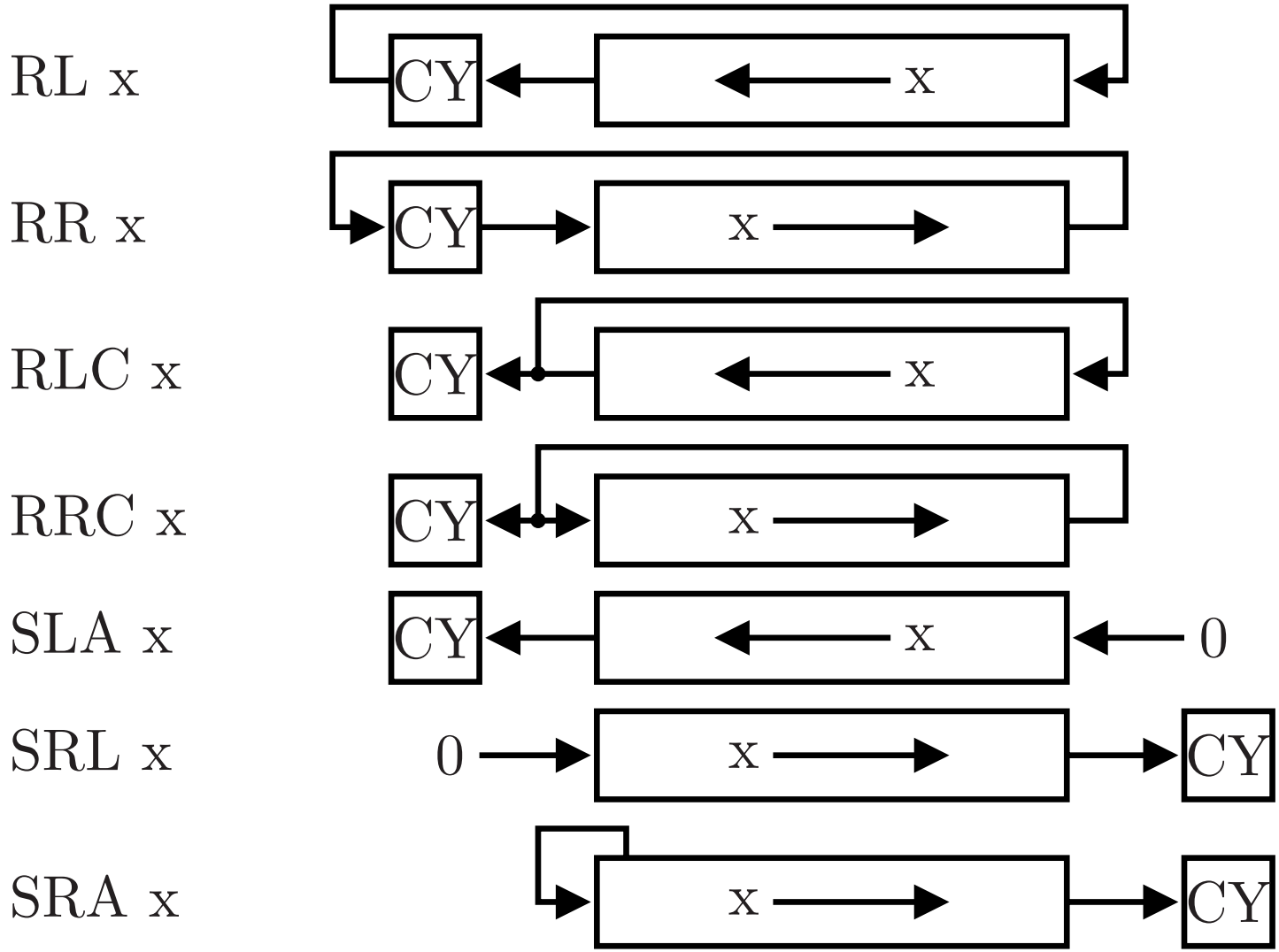
1 1	op	1 1 0
n		

s	argument
000	B
001	C
010	D
011	E
100	H
101	L
110	(HL)
111	A

op	operacja
000	ADD A, y
001	ADC A, y
010	SUB A, y
011	SBC A, y
100	AND A, y
101	OR A, y
110	XOR A, y
111	CP A, y

$y \in \{s, n\}$

Z80 – rotacje i przesunięcia bitów



$x \in \{A, B, C, D, E, H, L, (HL), (IX + d), (IY + d)\}$

Z80 – operacje na adresach (1)

- Ładowanie
  LD p, nn
  LD p, (aa)
  LD (aa), p
  gdzie p $\in$ {BC, DE, HL, SP, IX, IY}

- Zamiana
  EX DE, HL

- Zwiększenie i zmniejszenie o jeden
  INC p
  DEC p
  gdzie p $\in$ {BC, DE, HL, SP, IX, IY}

Z80 – operacje na adresach (2)

- Arytmetyka 16-bitowa
  ADD HL, p
  ADC HL, p
  SBC HL, p
  gdzie p $\in$ {BC, DE, HL, SP}

- Dodawanie i odejmowanie wartości w rejestrach indeksowych
  ADD IX, p
  gdzie p $\in$ {BC, DE, SP, IX}
  ADD IY, p
  gdzie p $\in$ {BC, DE, SP, IY}

Z80 – stos (1)

- Zajmuje pewien obszar w pamięci.
- Rejestr SP wskazuje wierzchołek stosu – ostatni zajęty bajt.
- Stos rośnie w dół – w kierunku mniejszych adresów.

Z80 – stos (2)

- PUSH p
  - Odkłada na stos najpierw starszy, potem młodszy bajt argumentu (architektura *little-endian*).
  - Zmniejsza SP o 2.
  - p $\in$ {AF, BC, DE, HL, IX, IY}
- POP p
  - Zdejmuje ze stosu najpierw młodszy, potem starszy bajt i umieszcza je w argumencie.
  - Zwiększa SP o 2.
  - p $\in$ {AF, BC, DE, HL, IX, IY}

Z80 – stos (3)

- Inicjowanie wskaźnika stosu
  LD SP, nn
  LD SP, (aa)
  LD SP, p
  gdzie p $\in$ {HL, IX, IY}

- Manipulowanie wierzchołkiem stosu
  INC SP
  DEC SP
  EX (SP), p
  gdzie p $\in$ {HL, IX, IY}

Z80 – skoki i podprogramy

JP aa

1 1 0 0 0 0 1 1
aal
aah

CALL aa

1 1 0 0 1 1 0 1
aal
aah

RET

1 1 0 0 1 0 0 1

- JP wykonuje skok pod podany adres (wpisuje aa do PC).
- CALL odkłada na stos adres następnego rozkazu (adres powrotu) i wykonuje skok pod podany adres.
- RET zdejmuje ze stosu adres powrotu i wykonuje skok pod ten adres.

Z80 – rozkazy warunkowe

JP w, aa

1 1	w	0 1 0
aal		
aah		

CALL w, aa

1 1	w	1 0 0
aal		
aah		

RET w

1 1	w	0 0 0

w	skrót	warunek skoku
000	NZ	Z = 0
001	Z	Z = 1
010	NC	CY = 0
011	C	CY = 1
100	PO	P/V = 0
101	PE	P/V = 1
110	P	S = 0
111	M	S = 1

Z80 – skoki względne

JR d

0 0 0 1 1 0 0 0
d

JR v, d

0 0 1	v	0 0 0
d		

v	skrót	warunek skoku
00	NZ	Z = 0
01	Z	Z = 1
10	NC	CY = 0
11	C	CY = 1

▷ Skok względny dodaje do PC wartość d traktowaną jako liczba ze znakiem.

Z80 – przykład procedury


```
MUL:     PUSH AF        ; Nie modyfikuj rejestrów roboczych.
         PUSH BC        ; w B i C argumenty
         LD   HL, 0     ; w HL wynik
         LD   A, B
         LD   B, H      ; wyzerowanie B
MULTST:  CP   A, 0
         JR   Z, MULEND
         SRL  A
         JR   NC, MUL2BC
         ADD  HL, BC
MUL2BC:  SLA  C         ; pomnożenie BC
         RL   B         ;   przez 2
         JR   MULTST
MULEND:  POP  BC        ; przywrócenie wartości
         POP  AF        ;   rejestrom roboczym
         RET
```

Z80 – przekazywanie argumentów przez stos


```
        LD H, (x)   ; drugi (ostatni) argument
        LD L, 5     ; pierwszy (przedostatni) argument
        PUSH HL     ; argumenty na stosie od ostatniego
        CALL MUL
        INC SP      ; czyszczenie stosu przez
        INC SP      ;   procedurę wołającą
        LD (y), HL  ; zapisanie wyniku

MUL:    PUSH IX
        LD IX, 3
        ADD IX, SP
        ...         ; (IX + 1) - pierwszy argument
        ...         ; (IX + 2) - drugi argument
        POP IX
        RET
```

Z80 – przekazywanie argumentów przez stos


```
        LD H, 5     ; pierwszy argument
        LD L, (x)   ; drugi argument
        PUSH HL     ; argumenty na stosie od pierwszego
        CALL MUL
        LD (y), HL  ; zapisanie wyniku

MUL:    LD IX, 4
        ADD IX, SP
        ...         ; (IX - 1) - pierwszy argument
        ...         ; (IX - 2) - drugi argument
        POP IY      ; zdjęcie adresu powrotu
        LD SP, IX   ; stos czyści procedura wołana
        JP (IY)     ; powrót z procedury
```

Z80 – cykl pracy

cykl rozkazowy	ADD A, E			
cykle procesora	pobranie			
cykle zegara	T_1	T_2	T_3	T_4

cykl rozkazowy	INC (HL)									
cykle procesora	pobranie				odczyt			zapis		
cykle zegara	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_1	T_2	T_3

▷ Istotne są tylko dostępy do pamięci!

Z80 – przerwania

- Służą do sygnalizowania procesorowi, że jakieś urządzenie wymaga obsługi, np.:
  - wciśnięto klawisz,
  - odebrano pakiet danych.
- Niemaskowalne
  - zawsze aktywne
- Maskowalne
  - włączane rozkazem EI – ustawienie przerzutnika przerwań IFF
  - wyłączane rozkazem DI – wyzerowanie IFF
  - trzy tryby, ustawiane odpowiednio rozkazami: IM0, IM1, IM2
  - Tryby 0 i 1 są odziedziczone po 8080, omówimy tylko tryb 2.

Z80 – zgłaszanie przerwań

- Urządzenie chcąc zgłasić przerwanie, podaje stan niski na wejście $\overline{\text{NMI}}$ lub $\overline{\text{INT}}$.

- Procesor testuje stan tych wejść w ostatnim takcie zegara każdego cyklu rozkazowego.

- Jeśli wejście $\overline{\text{NMI}}$ ma stan niski, zamiast wykonywać kolejny rozkaz, procesor przechodzi do obsługi przerwania niemaskowalnego.

- Jeśli wejście $\overline{\text{INT}}$ ma stan niski i przerwanie maskowalne jest włączone, zamiast wykonywać kolejny rozkaz, procesor przechodzi do obsługi przerwania maskowalnego.

Z80 – obsługa przerwania niemaskowalnego

- Zapamiętywany jest aktualny stan IFF.
- Przerwania maskowalne są wyłączane – IFF jest zerowany.
- Wykonywany jest skok do podprogramu obsługi rozpoczynającego się od adresu $(66)_{16}$.
- Zakończenie obsługi polega na wykonaniu rozkazu RETN.
- Rozkaz RETN zdejmuje ze stosu adres powrotu i wpisuje go do PC oraz przywraca poprzednią wartość IFF.

Z80 – obsługa przerwania maskowalnego (1)

- Procesor generuje zmodyfikowany cykl pobrania rozkazu.
- W tym cyklu wystawia sygnał potwierdzenia przyjęcia przerwania.
- Urządzenie zgłaszające przerwanie po rozpoznaniu potwierdzenia wystawia na szynę danych numer przerwania b.
- Wartość PC jest odkładana na stosie.
- Przerwanie maskowalne jest wyłączane.
- Z tablicy przerwań pobierany jest adres obsługi przerwania.
- Tablica przerwań jest wskazywana przez rejestr przerwań I.

Z80 – obsługa przerwania maskowalnego (2)

- Adres początku procedury obługi przerwania numer b znajduje się pod adresami $2^8 \cdot \mathrm{I} + b$ (młodszy bajt), $2^8 \cdot \mathrm{I} + b + 1$ (starszy bajt).
- Do PC wpisywany jest adres początku procedury obługi przerwania.
- Przed zakończeniem obsługi zwykle włącza się ponownie przyjmowanie przerwań rozkazem EI.
- Zakończenie obsługi polega zwykle na wykonaniu rozkazu RETI.
- Rozkaz RETI zdejmuje ze stosu adres powrotu i wpisuje go do PC.
- Do manipulowania rejestrem przerwań służą rozkazy
  LD I, A
  LD A, I

Z80 – szkielet procedury obsługi przerwania


```
INT:    EX AF, AF'
        EXX
        ...
        EXX
        EX AF, AF'
        EI
        RETI
```

Z80 – wejście-wyjście

- Komunikacja z urządzeniami, np.:
  - odczytanie kodu wciśniętego klawisza,
  - odczytanie odebranego pakietu danych,
  - sterowanie wyświetlaczem.

- Oddzielna przestrzeń adresowa wejścia-wyjścia

- Rozkazy
  IN A, (n)
  OUT (n), A
  IN r, (C)
  OUT (C), r
  gdzie r $\in$ {A, B, C, D, E, H, L}

Z80 – pozostałe rozkazy (1)

- Arytmetyczno-logiczne
  INC x
  DEC x
  BIT b, x
  SET b, x
  RES b, x
  RLD
  RRD
  CPL
  NEG
  CCF
  SCF
  DAA
  gdzie b $\in \{0, 1, \ldots, 7\}$,
  x $\in$ {A, B, C, D, E, H, L, (HL), (IX + d), (IY + d)}

Z80 – pozostałe rozkazy (2)

- Skoki
  DJNZ d
  JP (p)
  gdzie d $\in \{-128, -127, \ldots, 127\}$, p $\in$ {HL, IX, IY}

- Operacje na ciągach bajtów w pamięci
  LDI
  LDD
  LDIR
  LDDR
  CPI
  CPD
  CPIR
  CPDR

Z80 – pozostałe rozkazy (3)

- Przesłania blokowe z wejścia i na wyjście
  INI
  IND
  INIR
  INDR
  OUTI
  OUTD
  OUTIR
  OUTDR

- Przerwania programowe
  RST a
  gdzie a $\in \{0, 8, 16, \ldots, 56\}$

Z80 – pozostałe rozkazy (4)

- Inne
  NOP
  HALT
  LD R, A
  LD A, R

- Nieudokumentowane
  EX DE, IX
  EX DE, IY
  SLL x / SL1 x
  ...

Małe podsumowanie (1)

- Z punku widzenia programisty wszystkie mikroprocesory działają podobnie do opisanego.

- Poszczególne architektury różnią się m.in.:
  - porządkiem bajtów,
  - liczbą i przeznaczeniem rejestrów,
  - listą oferowanych instrukcji (rozkazów),
  - semantyką operacji dwuargumentowych,
  - sposobem realizacji sterowania programem,
  - trybami adresowania,
  - systemem przerwań.

Małe podsumowanie (2)

- Dalsze różnice:
  - rozdzielne lub wspólne przestrzenie adresowe: danych i programu (architektury typu Princeton lub Harward), wejścia-wyjścia,
  - mechanizmy ochrony pamięci (brak, segmentacja, stronicowanie).
- Żeby działać szybciej, współczesne mikroprocesory mają zupełnie inną organizację (budowę) niż opisany.

Rejestry (1)

- danych
- adresowe
- ogólnego przeznaczenia
- specjalizowane
- stanu, znaczników
- wskaźnik stosu
- licznik programu
- zmiennopozycyjne
- wektorowe
- segmentowe
- zarządzania pamięcią
- debugowania

Rejestry (2)

- architektura bezrejestrowa
- minimalny zestaw rejestrów
- mały zestaw rejestrów specjalizowanych
- mały zestaw rejestrów uniwersalnych
- duży zestaw rejestrów uniwersalnych
- bufor wierzchołka stosu
- stosowy zestaw rejestrów

Lista instrukcji (rozkazów)

- Aplikacyjne
  - przesyłanie danych
  - operacje arytmetyczne
  - operacje bitowe
  - operacje na blokach danych
  - instrukcje sterujące (skok bezwarunkowy i warunkowy, warunkowe wykonanie instrukcji, wołanie procedury, powrót z procedury)
- Systemowe (tryb uprzywilejowany lub nadzorcy)
  - zarządzanie systemowymi strukturami danych
  - zmiana trybu pracy procesora

Operacje dwuargumentowe

Przykładowa składnia	Możliwa semantyka
`add r1, r2`	`r1 := r1 + r2`
`add r1, r2`	`r2 := r1 + r2`
`add r1, r2, r3`	`r1 := r2 + r3`
`add r1, r2, r3`	`r3 := r1 + r2`

Sterowanie wykonaniem programu

- Jednoetapowe (bez znaczników)

```
bz s1, label                    ; branch if s1 = 0
jecxz label                     ; jump if ecx = 0
```

- Dwuetapowe ze znacznikami

```
CP A, 0                         ; set flags
JR Z, MULEND                    ; jump if Z = 1
cmp eax, 0                      ; set flags
jz mulend                       ; jump if zf = 1
```

- Dwuetapowe z predykatami

```
cmp.eq p1, p2 = r1, r2          ; if r1 = r2
(p1) add r3 = 1, r3             ; then r3 := r3 + 1
(p2) add r3 = 3, r3             ; else r3 := r3 + 3
```

Tryby adresowania argumentów (1)

Nazwa	Z80	IA-32
natychmiastowy	`LD B, 7`	`mov ebx, 7`
bezpośredni	`LD A, (1000)`	`mov eax, [1000]`
rejestrowy (bezpośredni)	`ADD A, B`	`add eax, ebx`
(rejestrowy) pośredni	`ADD A, (HL)`	`add eax, [ebx]`
indeksowy	`LD A, (IX+8)`	`mov eax, [ebx+4*ecx+8]`

stosowy	`PUSH AF`
	`POP AF`
	`EX (SP), HL`
		`push eax`
		`pop eax`
		`xchg [esp+8*ecx], eax`

względny	`JR Z, MULEND`	`jz mulend`

Tryby adresowania argumentów (2)

Nazwa	Oznaczenia	Z80	IA-32
rejestrowy pośredni z preinkrementacją	`[+r1]` `(+r1)`		
rejestrowy pośredni z postinkrementacją	`[r1+]` `(r1+)` `@r1+`	`LDI` `POP AF`	`movs` `pop eax`
rejestrowy pośredni z predekrementacją	`[-r1]` `(-r1)`	`DJNZ d` `PUSH AF`	`loop d` `push eax`
rejestrowy pośredni z postdekrementacją	`[r1-]` `(r1-)` `@r1-`	`LDD`	`movs`

System przerwań

- Przerwania sprzętowe
  - maskowalne
  - niemaskowalne
- Przerwania programowe
- Praca krokowa – debugowanie
- Wyjątki – ponawianie instrukcji, która spowodowała wyjątek
- Element tablicy przerwań
  - adres procedury obsługi
  - instrukcja skoku do procedury obsługi
  - deskryptor procedury obsługi

Pomiar wydajności

- MIPS – Milion Instructions Per Second
- DMIPS – Dhrystone MIPS
- MWIPS – Whetstone MIPS
- FLOPS – Floating Point Operations Per Second
  - MFLOPS $= 10^6 \cdot$ FLOPS
  - GFLOPS $= 10^9 \cdot$ FLOPS
  - TFLOPS $= 10^{12} \cdot$ FLOPS
  - PFLOPS $= 10^{15} \cdot$ FLOPS

Metody zwiększania wydajności (1)

- Zwiększanie częstotliwości taktowania
  - ograniczenia konstrukcyjne i technologiczne
  - wydzielanie ciepła
  - czas propagacji

- Inne pomysły
  - pobieranie instrukcji na zakładkę
  - kolejka (bufor) instrukcji
  - przetwarzanie potokowe
  - zrównoleglanie wykonywania instrukcji

Metody zwiększania wydajności (2)

- Nienadążanie szybkość pracy pamięci za wzrostem szybkości pracy procesorów
  - zwiększanie szerokości szyny danych
  - wielopoziomowe pamięci podręczne, początkowo jako zewnętrzne, a obecnie w jednym układzie scalonym z procesorem
  - pamięci umożliwiające pobieranie danych co jeden cykl zegara, a nawet dwa razy w jednym cyklu zegara

Przetwarzanie potokowe

I_1				I_2			I_3			I_4			
T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3	T_4

I_1					I_2					I_3					I_4				
F	D	E	M	W	F	D	E	M	W	F	D	E	M	W	F	D	E	M	W

F	D	E	M	W															
	F	D	E	M	W														
		F	D	E	M	W													
			F	D	E	M	W												

F – fetch

D – decode

E – execute

M – memory access

W – write back

Potokowanie rozkazów

- Rozkazy procesora takiego jak Z80 źle się potokuje:
  - kod rozkazu zajmuje od 1 do 4 bajtów, co wymaga do 4 dostępów do pamięci;
  - rozkaz może wykonywać skomplikowane operacje na argumentach w pamięci, co może wymagać kilku kolejnych odwołań.

- Listę rozkazów należy przeprojektować tak, aby wszystkie rozkazy:
  - miały kod tego samego rozmiaru,
  - wykonywały podobną liczbę operacji,
  - potrzebowały co najwyżej jednego dodatkowego dostępu do pamięci (oprócz wczytania kodu rozkazu).

Klasy rozkazów

- Ograniczamy repertuar rozkazów do następujących klas.
- Odczyt z pamięci do rejestru (ang. *load*), np.:
  `r1 := (r2 + offset)`
- Zapis z rejestru do pamięci (ang. *store*), np.:
  `(r2 + offset) := r1`
- Arytmetyczno-logiczne tylko na rejestrach, np.:
  `r1 := r2 + r3`
- Rozgałęzienia jednoetapowe, np.:
  `branch to label if r1 = r2`

Podział rozkazów na etapy

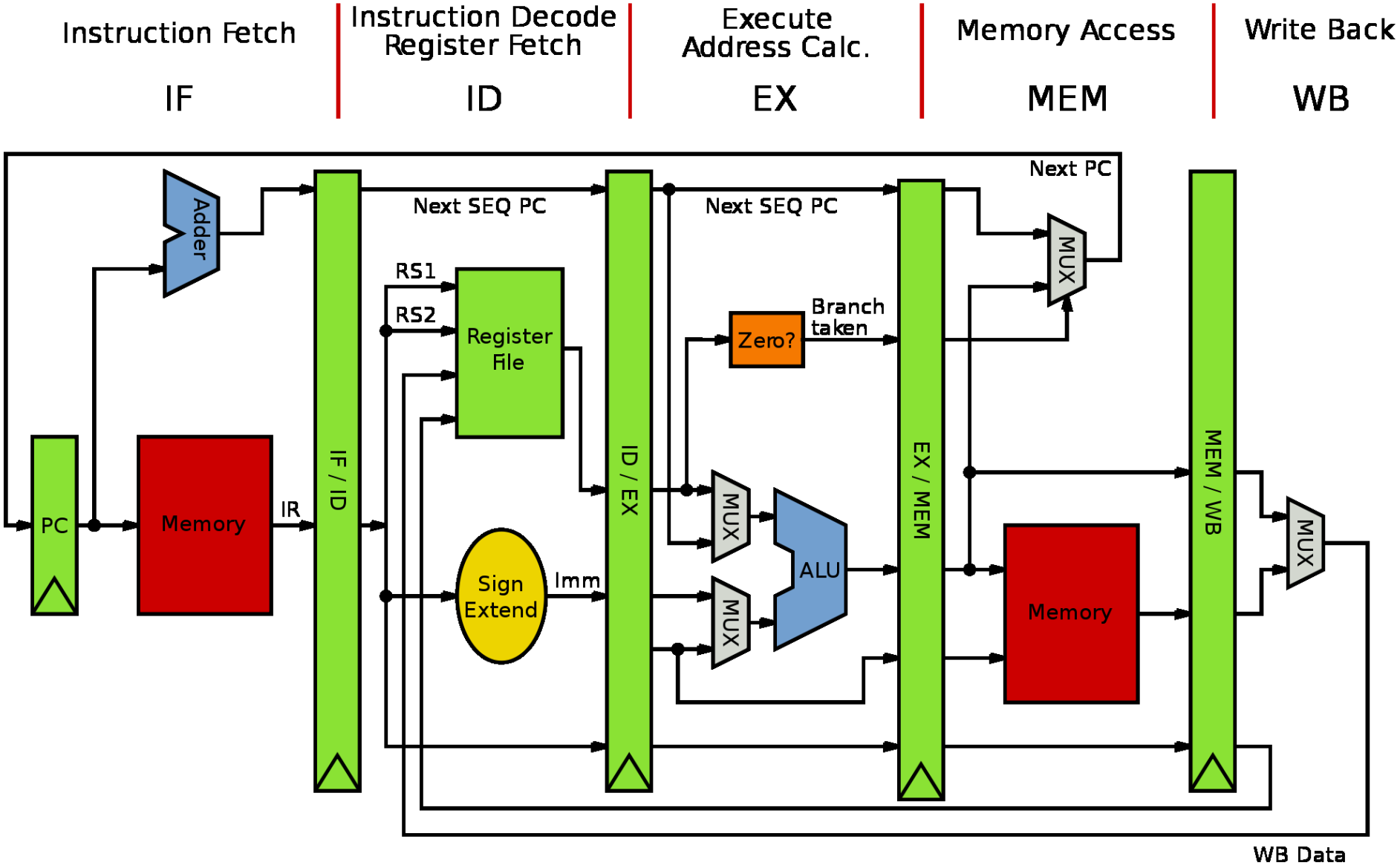
- Rozważamy przykładowy potok piecioetapowy.
- F – pobranie kodu rozkazu z pamięci
- D – zdekodowanie rozkazu i pobranie argumentów z rejestrów
- E – wykonanie operacji arytmetyczno-logicznych, obliczenie adresu
- M – odczytanie danych z pamięci lub zapisanie danych do pamięci
- W – zapisanie wyniku do rejestru

Przykładowe czasy wykonania

klasa instrukcji	F	D	E	M	W	razem
odczyt z pam.	4 ns	2 ns	3 ns	4 ns	2 ns	15 ns
zapis do pam.	4 ns	2 ns	3 ns	4 ns		13 ns
arytm.-logiczne	4 ns	2 ns	4 ns		2 ns	12 ns
rozgałęzienia	4 ns	2 ns	3 ns			9 ns

- ▷ Rozkaz musi przebywać w jednym etapie potoku 4 ns.
- ▷ Wykonanie jednego rozkazu potokowo trwa 20 ns, czyli dłużej niż wykonanie dowolnego rozkazu niepotokowanego.
- ▷ Wykonanie n rozkazów potokowo trwa $4(n + 4)$ ns.

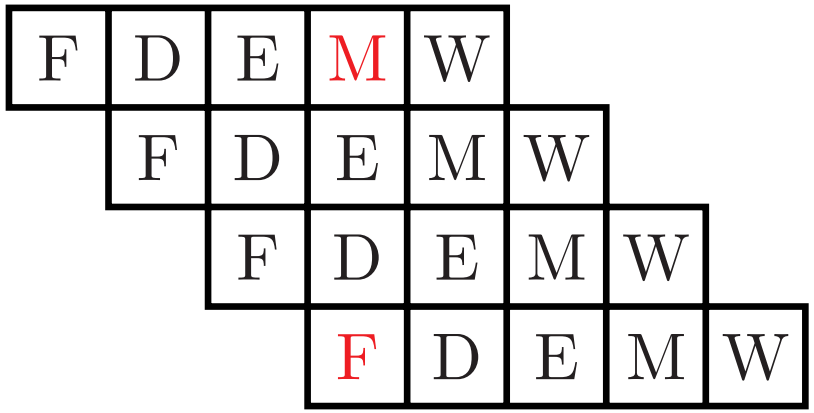
Budowa potoku



Zależności zasobów, strukturalne

- Potok może zostać wstrzymany, gdy rozkazy będące na różnych etapach wykonania potrzebują dostępu do tego samego sprzętu.

- Przykładowo rozkaz chce wykonać odczyt danych z pamięci, a w tym samym czasie trzeba pobrać kod kolejnego rozkazu.



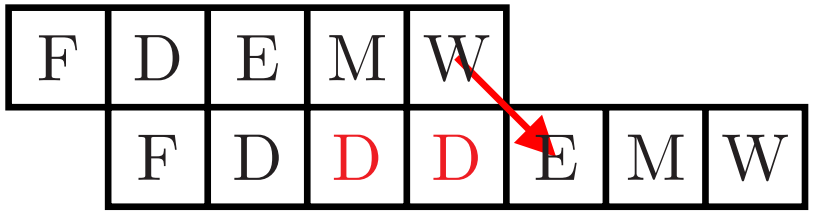
- Zależności strukturalne usuwa się przez dołożenie sprzętu.

- Dlatego m.in. współczesne mikroprocesory mają oddzielne pamięci podręczne pierwszego poziomu dla kodu i danych.

Zależności danych typu *define-use*

- Potok może zostać wstrzymany, gdy rozkaz potrzebuje wyniku poprzedniego rozkazu.

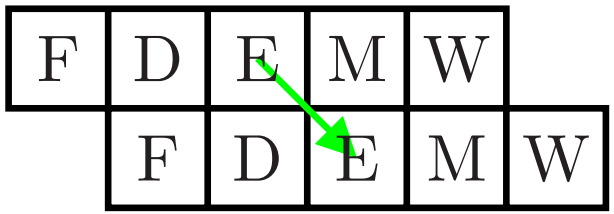
```
r1 := r2 + r3
r4 := r1 - r5
```



- Takie zależności usuwa się przez data forwarding lub data bypassing.

- Wynik obliczony na etapie E pierwszego rozkazu przesyła się bezpośrednio do etapu E drugiego rozkazu, zanim zostanie zapisany do rejestru wynikowego.

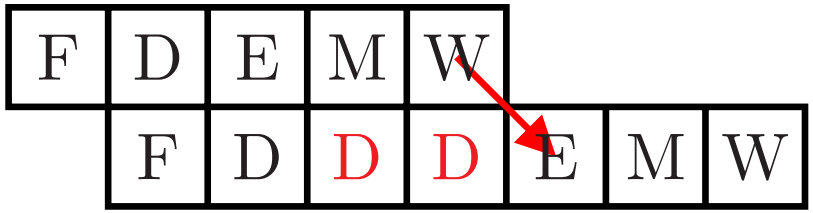
```
r1 := r2 + r3
r4 := r1 - r5
```



Zależności danych typu *load-use*

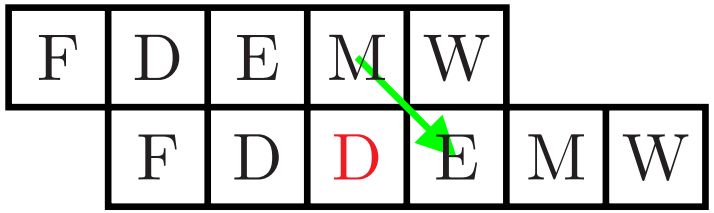
- Potok może zostać wstrzymany, gdy rozkaz potrzebuje danych ładowanych z pamięci przez poprzedni rozkaz.

```
r1 := (r2 + 4)
r4 := r1 - r5
```



- Data forwarding nie zawsze jest w stanie usunąć taką zależność.

```
r1 := (r2 + 4)
r4 := r1 - r5
```

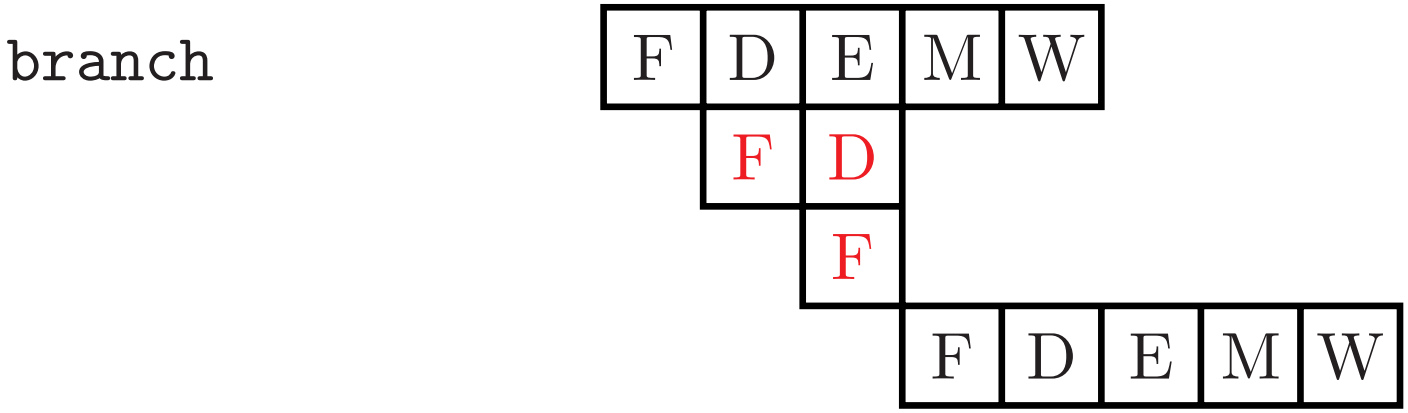


- Dodatkowe opóźnienie wprowadzane przez rozkaz, gdy następny rozkaz potrzebuje jego wyniku, nazywa się latency.

- Kompilator może usuwać latency, zmieniając kolejność rozkazów.

Zależności sterowania

▷ Potok musi być opróżniony, gdy rozkaz warunkowy wykona skok.

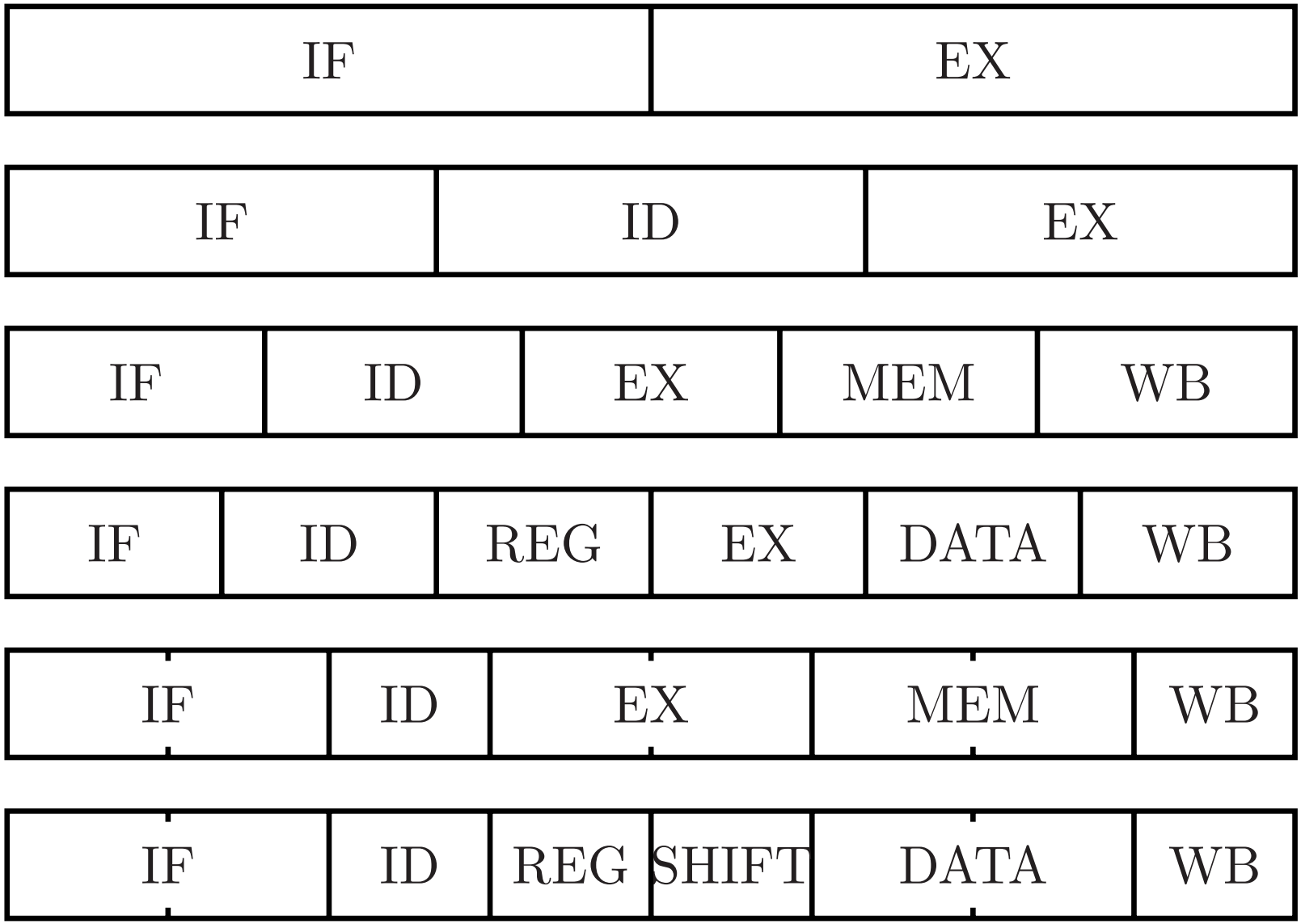


▷ W rozważanym przykładzie dodatkowe opóźnienie (latency) wynosi dwa takty.

Rozwiązywanie zależności sterowania

- Równoległe przetwarzanie obu gałęzi programu
- Predykcja skoków (przewidywanie rozgałęzień)
  - zawsze następuje skok
  - nigdy nie następuje skok
  - decyduje kod instrukcji (kompilator)
  - jak przy ostatnim wykonaniu
  - tablica historii skoków
- Opóźnione rozgałęzianie

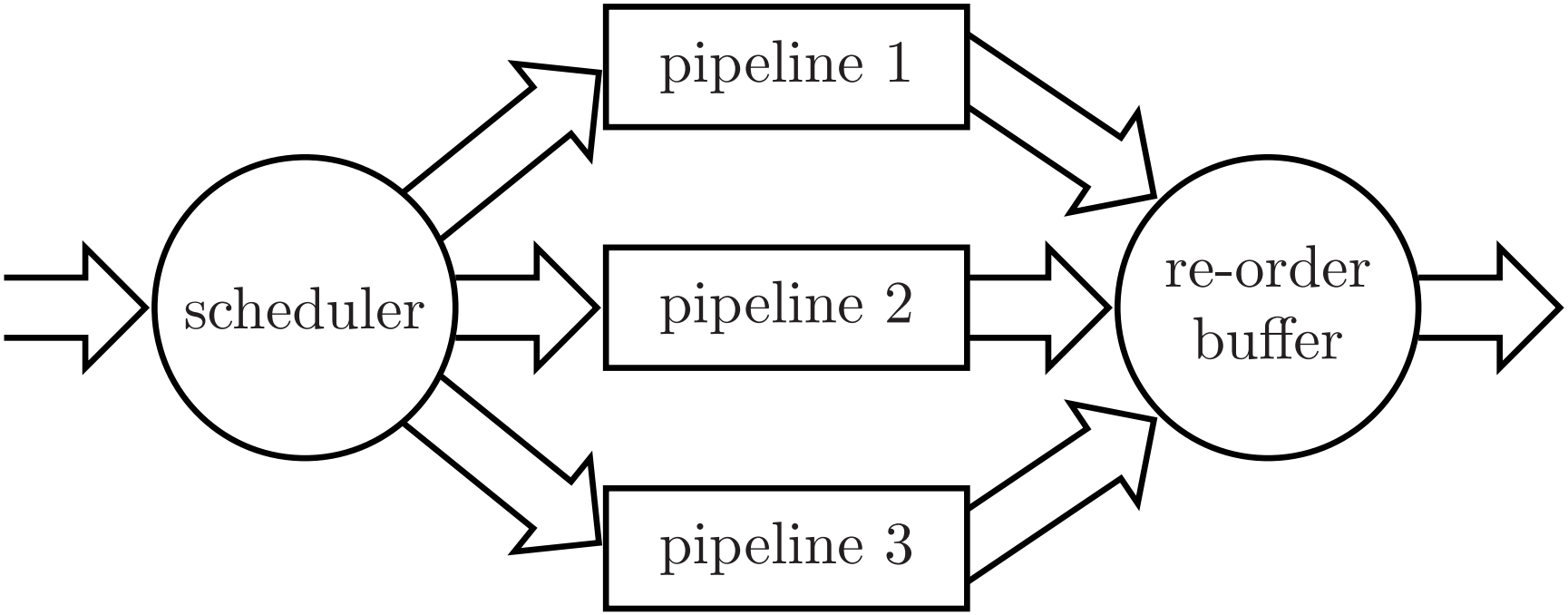
Przykładowe architektury potoków



Metody zrównoleglania

Procesor	Zrównoleglenie
superskalarny	w trakcie wykonywania
wektorowy SIMD – Single Instruction Multiple Data	algorytm
VLIW – Very Long Instruction Word EPIC – Explicite Parallel Instruction Computing	kompilator

Przetwarzanie wielopotokowe



▷ Wydawanie i wykonywanie rozkazów w zmienionej kolejności (ang. *out of order issue and execution*)

Zależności danych przy wielopotokowości

- RAW – odczyt po zapisie

  load-use
  ```
  mov eax, [ecx+4]
  add ebx, eax
  ```

  define-use
  ```
  add eax, ecx
  add ebx, eax
  ```

- WAR – zapis po odczycie
  ```
  add eax, ebx
  mov ebx, 4
  ```

- WAW – zapis po zapisie
  ```
  add eax, ebx
  and ecx, edx
  ```

- Zależności RAW rozwiązuje się jak poprzednio przez data forwarding lub data bypassing.
- Zależności WAR i WAW są zależnościami fałszywymi i rozwiązuje się je przez przemianowanie rejestrów.

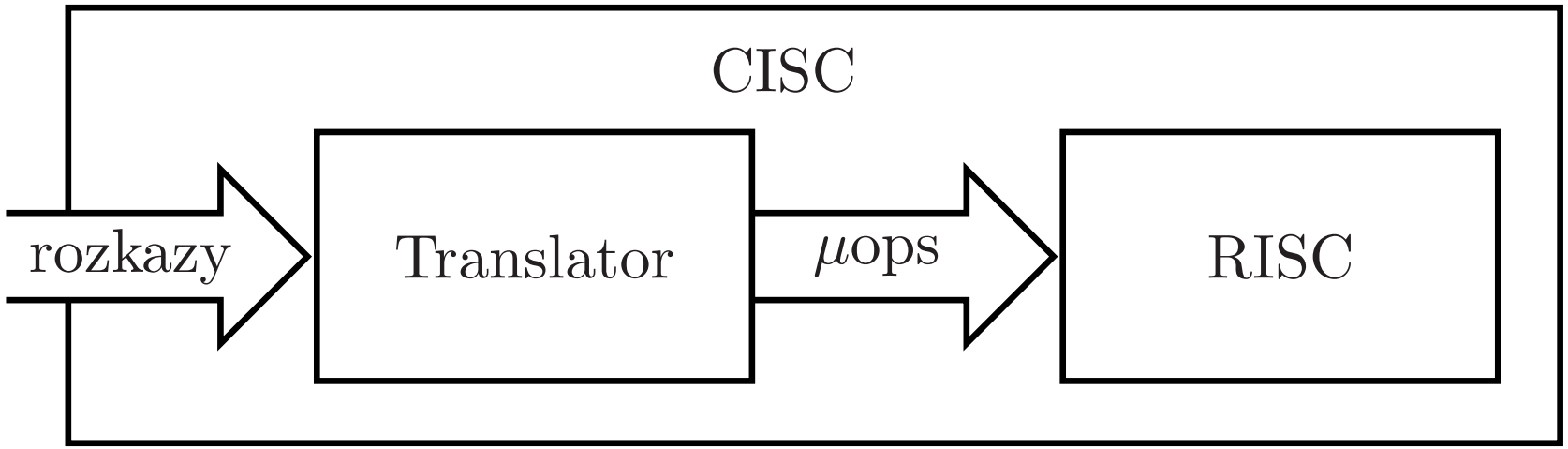
Architektury RISC i CISC (1)

RISC – Reduced Instruction Set Computer	CISC – Complex Instruction Set Computer
Ograniczony, prosty zbiór instrukcji	Skomplikowane instrukcje, instrukcje wspierające języki wysokiego poziomu
Duża liczba uniwersalnych rejestrów	Mała liczba rejestrów lub rejestry specjalizowane
Instrukcje arytmetyczno-logiczne wykonywane tylko na rejestrach	Instrukcje arytmetyczno-logiczne pobierające argumenty z pamięci lub umieszczające wynik w pamięci

Architektury RISC i CISC (2)

RISC – Reduced Instruction Set Computer	CISC – Complex Instruction Set Computer
Kody instrukcji o stałej długości, typowo 4 bajty	Kody instrukcji o zmiennej długości, typowo od jednego do kilkunastu bajtów
Stałe rozmieszczenie pól w kodach instrukcji, ułatwiające dekodowanie	Prefiksowanie instrukcji, utrudniające dekodowanie
Mała liczba trybów adresowania	Duża liczba trybów adresowania
Dozwolone tylko adresowanie wyrównane	Dozwolone adresowanie niewyrównane
Mniejsza gęstość kodu	Większa gęstość kodu

Architektury RISC i CISC (3)



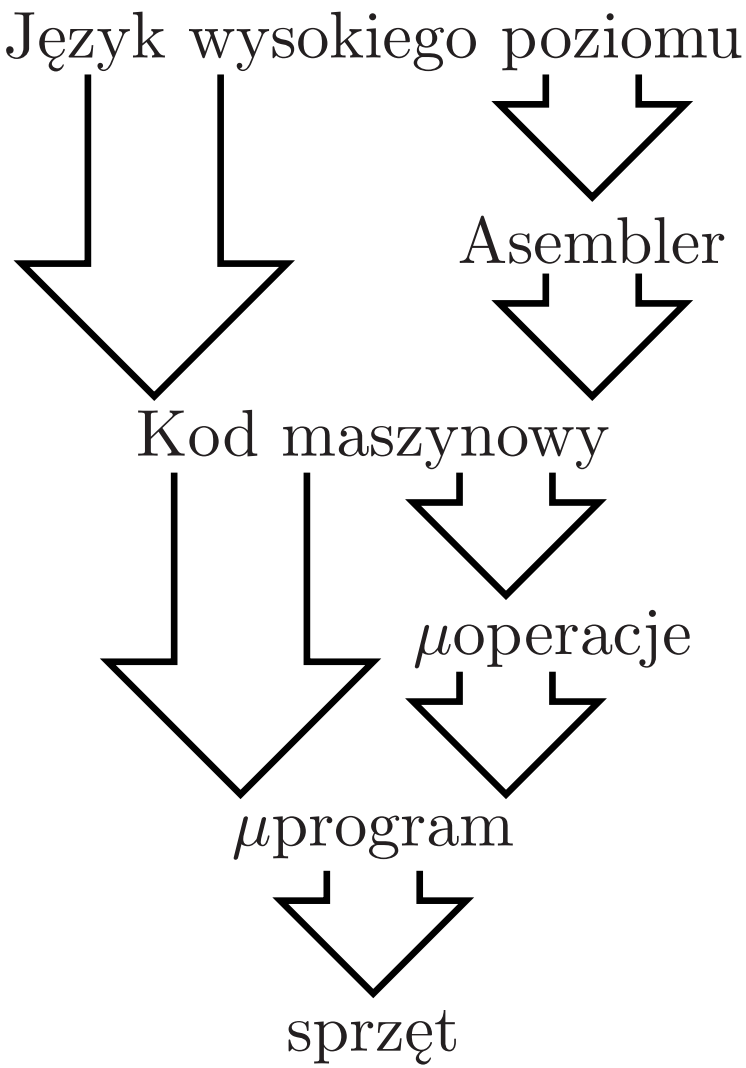
Geneza mikroprocesora

- Zbyt czasochłonne projektowanie od podstaw z bramek logicznych nowego modelu np. kalkulatora lub sterownika windy
- Podczas projektowania łatwo popełnić trudny do usunięcia błąd
- Układ uniwersalny – mikroprocesor
- Nowa funkcjonalność przez wymianę programu

Geneza mikroprogramowania

- Zbyt czasochłonne projektowanie od podstaw z bramek logicznych nowej jednostki sterującej (mikro)procesora
- Podczas projektowania łatwo popełnić trudny do usunięcia błąd
- Uniwersalna jednostka sterująca
- Nowa funkcjonalność przez wymianę mikroprogramu

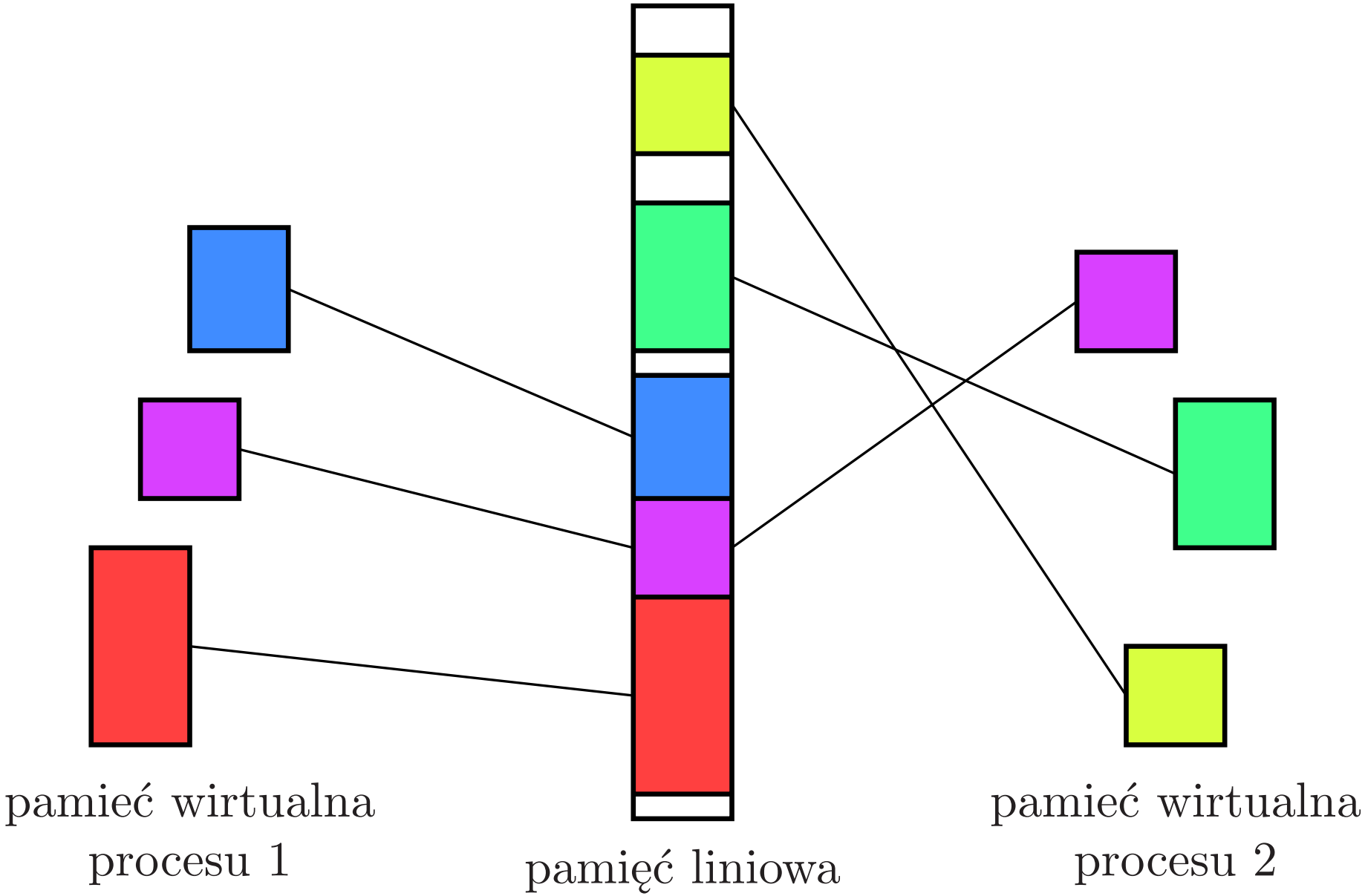
Mikroprogramowanie



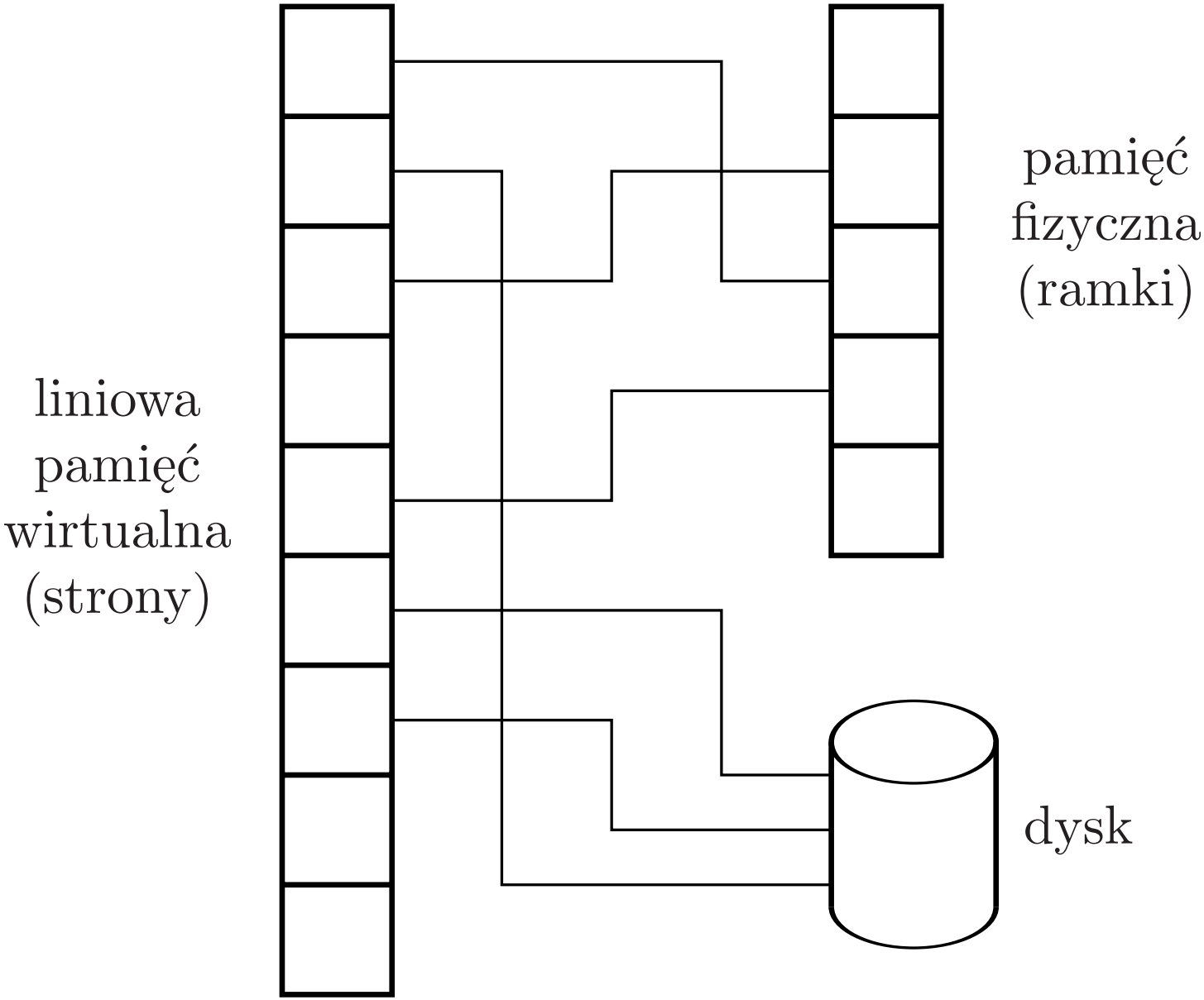
Sprzętowe wsparcie dla systemów operacyjnych

- Translacja adresów
  - segmentacja
  - stronicowanie, prosta i odwrotna tablica stron
- Poziomy ochrony
  - wewnętrzny, nadzorcy, uprzywilejowany
  - zewnętrzny, aplikacji
- Wywoływanie usług systemu operacyjnego
  - przerwania
  - specjalna instrukcja (syscall)

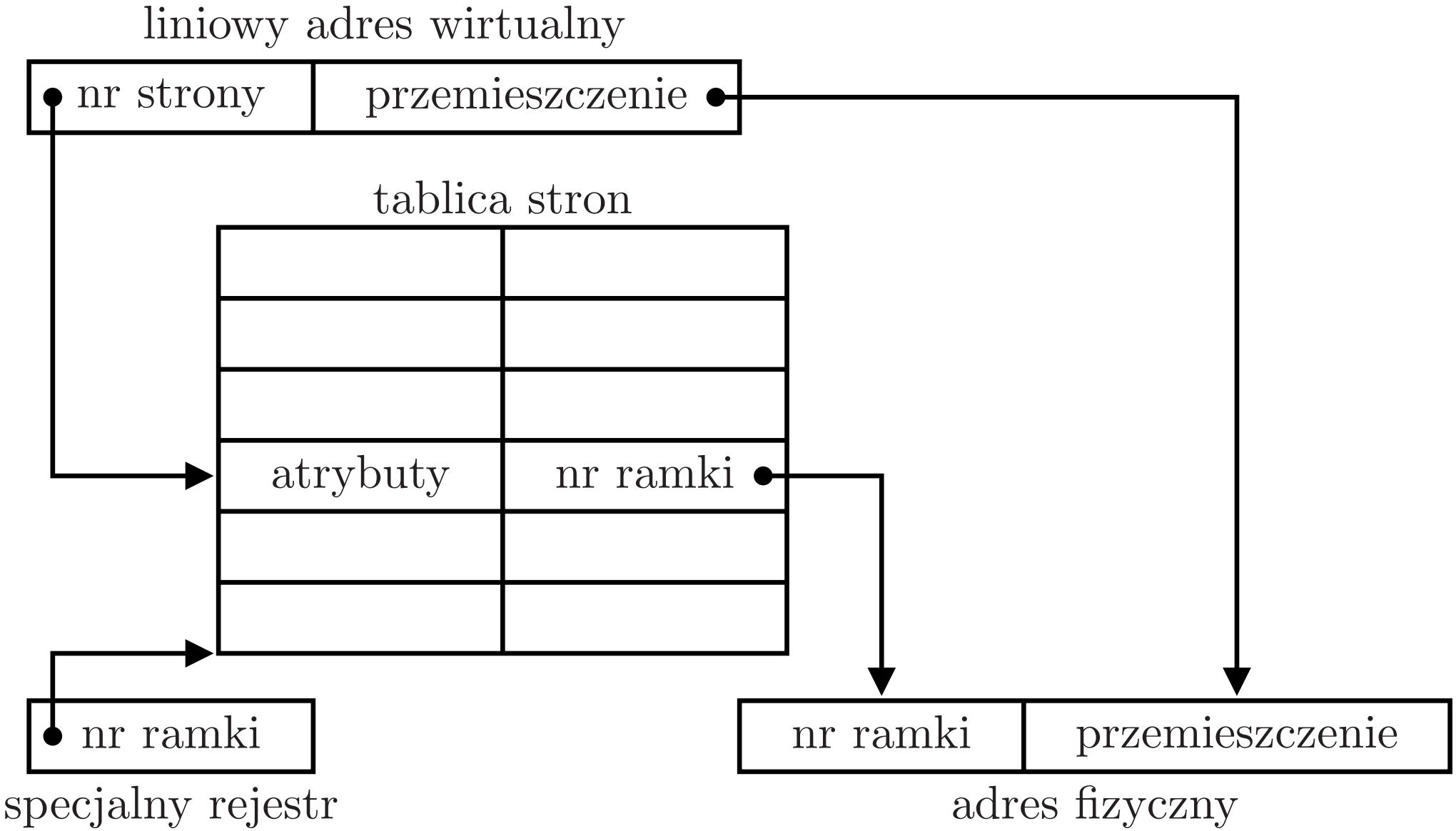
Segmentacja



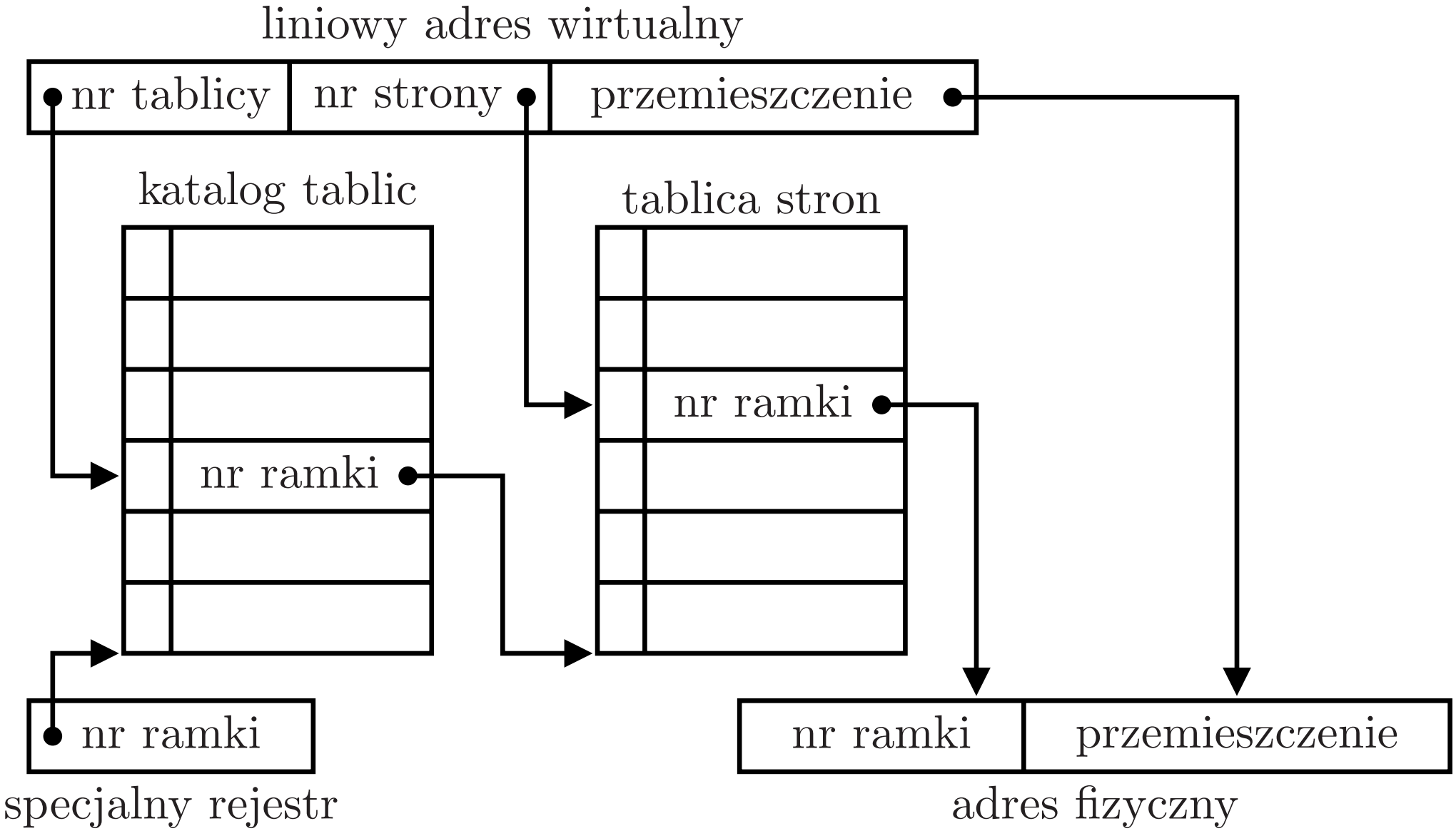
Stronicowanie



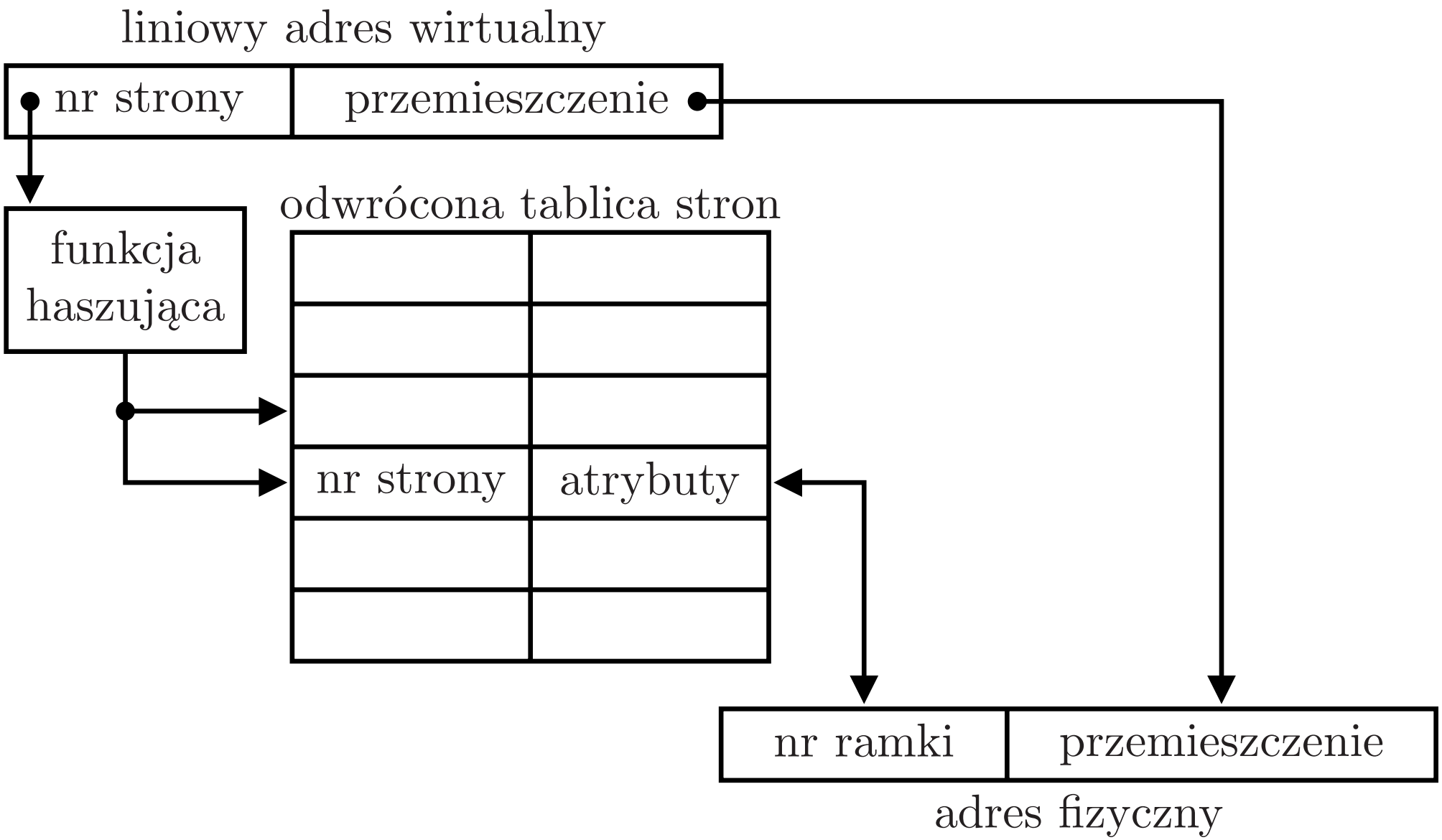
Tablica stron



Stronicowanie wielopoziomowe



Odwrócona tablica stron



TLB

- Każde odwołanie do pamięci wymaga zajrzenia do tablicy stron, która sama jest przechowywana w pamięci.
- Taki mechanizm jest nieefektywny.
- Przy odwoływaniu się do pamięci obowiązuje zasada lokalności.
- Rozwiązaniem jest bufor translacji adresów (ang. *TLB – Translation Lookaside Buffer*).

DMA – Direct Memory Access

- Odciąża procesor przy przesyłaniu dużych bloków danych.
- Dane przesyłane są bezpośrednio między sterownikiem urządzenia a pamięcią operacyjną.
- Przesyłanie obsługuje sterownik DMA – specjalizowany układ podłączony do szyny systemowej.
- Fazy transmisji:
  - rozkazów – procesor inicjuje kanał DMA i zapisuje parametry transmisji,
  - danych – wykonywane jest przesyłanie,
  - statusu – sterownik DMA zgłasza przerwanie, procesor odczytuje status i zwalnia kanał DMA.

Mikroprocesory są wszędzie