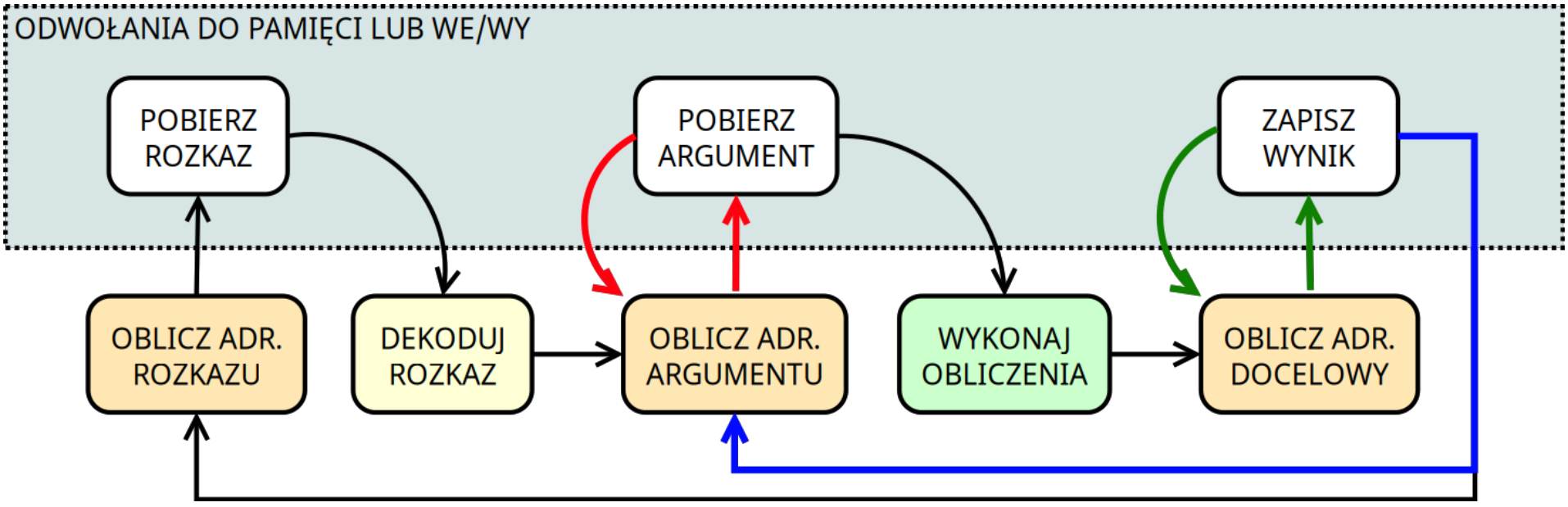
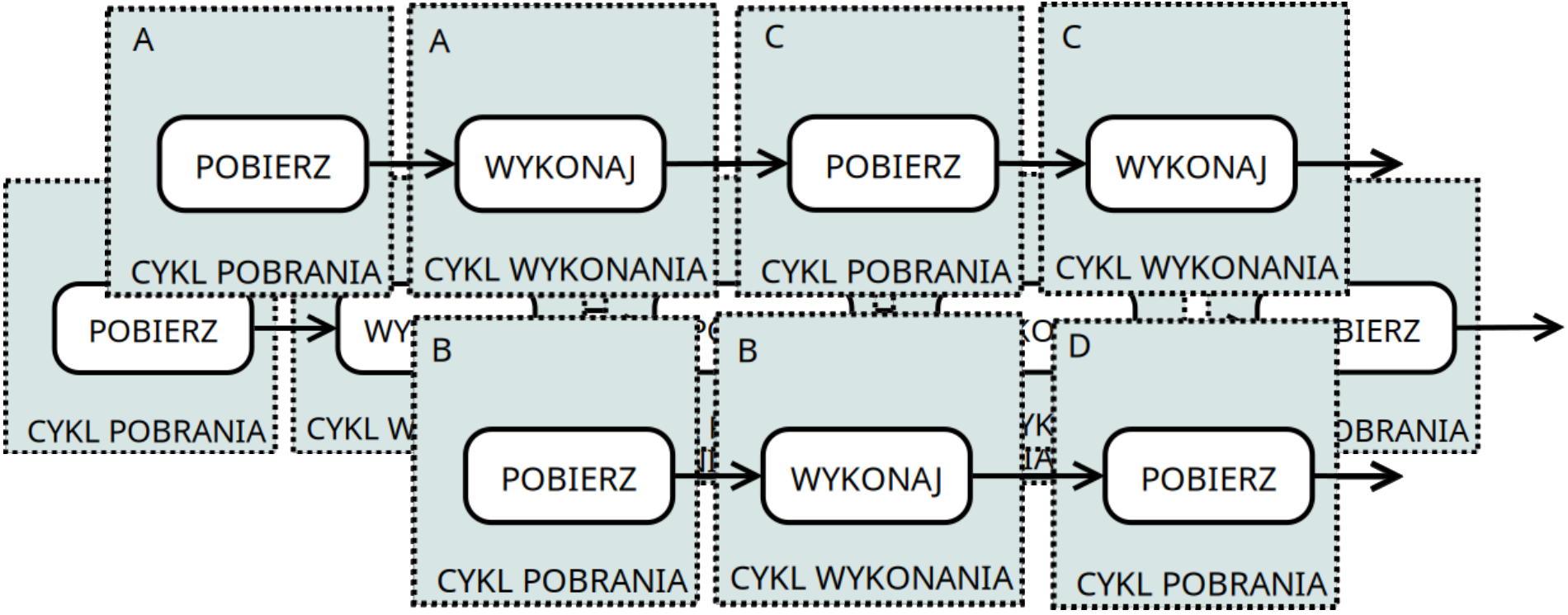
Przetwarzanie potokowe


ODWOŁANIA DO PAMIĘCI LUB WE/WY
POBIERZ ROZKAZ
POBIERZ ARGUMENT
ZAPISZ WYNIK
OBLICZ ADR. ROZKAZU
DEKODUJ ROZKAZ
OBLICZ ADR. ARGUMENTU
WYKONAJ OBLICZENIA
OBLICZ ADR. DOCELOWY

Od liniowego do potokowego



- O ile udało się przyspieszyć?
- ALE: czas pobrania < czas wykonania
  - tracimy czas po „POBIERZ” (czekamy)
  - dzielimy „WYKONAJ” na kilka pomniejszych
  - dodajemy bufory (zatrzaski) między blokami
- A co ze skokami?
  - zgadujemy
  - ignorujemy

Fazy wykonania rozkazu (*)

- FI – pobierz instrukcję (MEM)
- DI – zdekoduj instrukcję
- CO – wylicz operandy (ALU)
- FO – pobierz operandy (MEM)
- EI – wykonaj rozkaz (ALU)
- WO – zapisz wynik (MEM)

„Czysty” potok

	1	2	3	4	5	6	7	8	9	10	11	12	13
R1	FI	DI	CO	FO	EI	WO							
R2		FI	DI	CO	FO	EI	WO						
R3			FI	DI	CO	FO	EI	WO					
R4				FI	DI	CO	FO	EI	WO				
R5					FI	DI	CO	FO	EI	WO			
R6						FI	DI	CO	FO	EI	WO		
R7							FI	DI	CO	FO	EI	WO	
R8								FI	DI	CO	FO	EI	WO

Zawsze jest „ale”:

- Każdy rozkaz to wszystkie etapy (?)
- Brak konfliktów dostępu do pamięci
- Brak skoków warunkowych
- Brak przerwań
- Brak zależności danych między rozkazami

Potok ze skokiem warunkowym

	1	2	3	4	5	6	7	8	9	10	11	12	13
R1	FI	DI	CO	FO	EI	WO							
R2		FI	DI	CO	FO	EI	WO						
JNZ loop R3			FI	DI	CO	FO	EI	WO					
R4				FI	DI	CO	FO	EI	WO				
R5					FI	DI	CO	FO	EI	WO			
R6						FI	DI	CO	FO	EI	WO		
R7							FI	DI	CO	FO	EI	WO	
loop: R8								FI	DI	CO	FO	EI	WO

Efektywność potoku

- k – liczba etapów w potoku (głębokość)
- t – maksymalne opóźnienie etapu
- n – liczba wykonywanych rozkazów

czas wykonania w potoku: $T = [k + (n-1)]t$

przyspieszenie: $S = nk/[k+(n-1)]$

Inny przykład (opóźnienia)

- Zegar: 4GHz → 0,5ns
- Narzut potoku: 0,1 ns

Operacja	l. taktów	częstotliwość
ALU	4	40%
BRA	4	20%
MEM	5	40%

Wydajność bez potoku: [4*0,4+4*0,2+5*0,4]*0,5 ns = 4,4*0,5 ns = 2,2 ns/instr.
Wydajność z potokiem: 0,5 + 0,1 = 0,6 ns/instr.

przyspieszenie: S = 2,2/0,6 = 3,7

Zagrożenia dla potoku

- Zagrożenie strukturalne (dostęp do zasobu)
  - zasób: pamięć, ALU, FPU...
- Zagrożenie danych
  - konflikt rozkazów sięgających tego samego miejsca
- Zagrożenie sterowania
  - skoki warunkowe

Zagrożenie strukturalne

	1	2	3	4	5
R1	**FI**	**DI**	**CO**	**FO**	**EI**
R2		**FI**	**DI**	**CO**	**FO**
R3			**FI**	**DI**	**CO**
R4				**FI**	**DI**
R5					**FI**

W tej samej chwili (t=4) pobierany jest operand dla R1 oraz instrukcja dla R4

	1	2	3	4	5
R1	**FI**	**DI**	**CO**	**FO**	**EI**
R2		**FI**	**DI**	**CO**	**FO**
R3			**FI**	**DI**	**CO**
R4				**stall**	**FI**
R5					

Operacja pobrania instrukcji dla R4 została opóźniona o 1 takt (co koliduje z ...)

Zagrożenie danych (1)

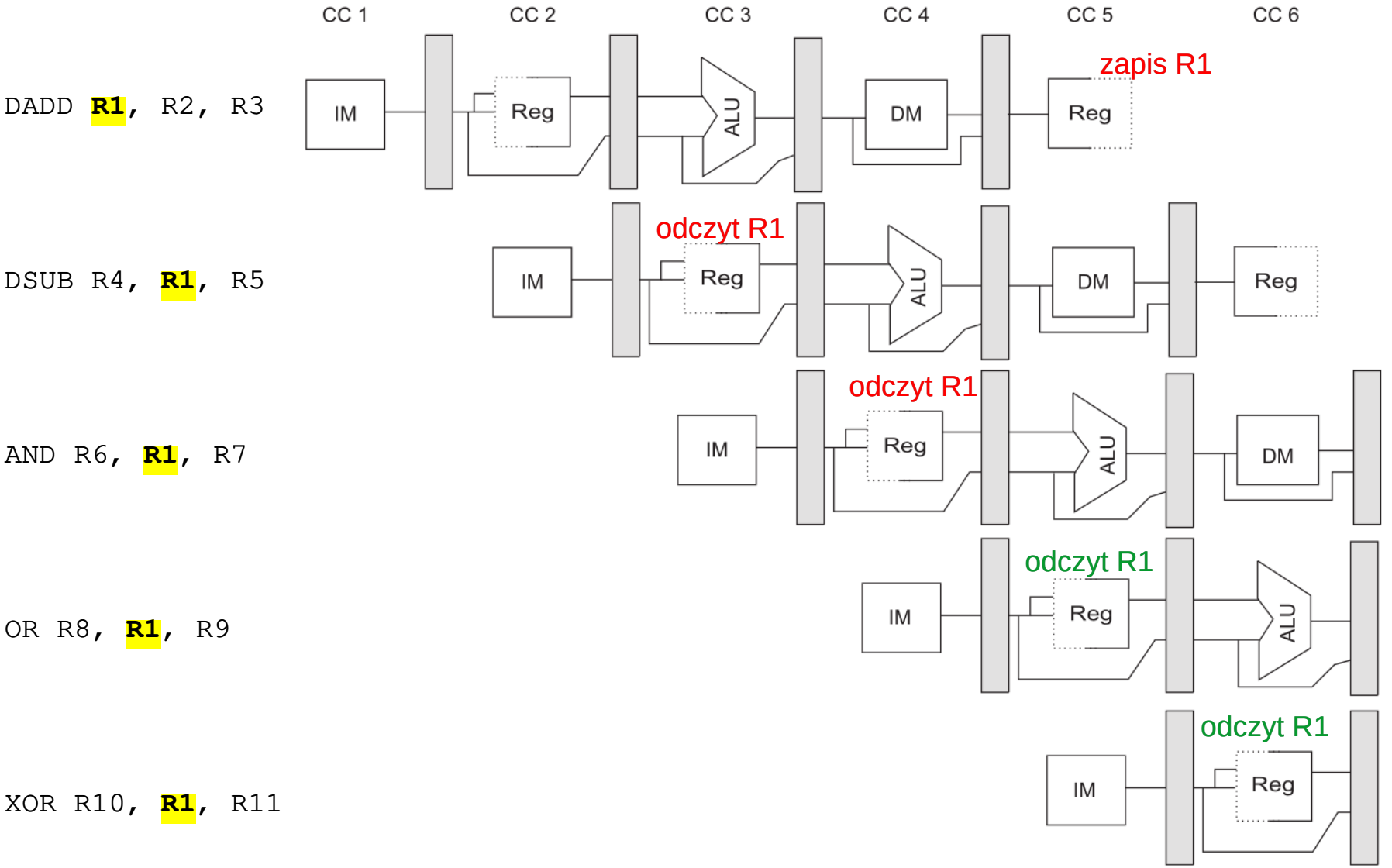
```
ADD EAX, EBX
SUB ECX, EAX
```

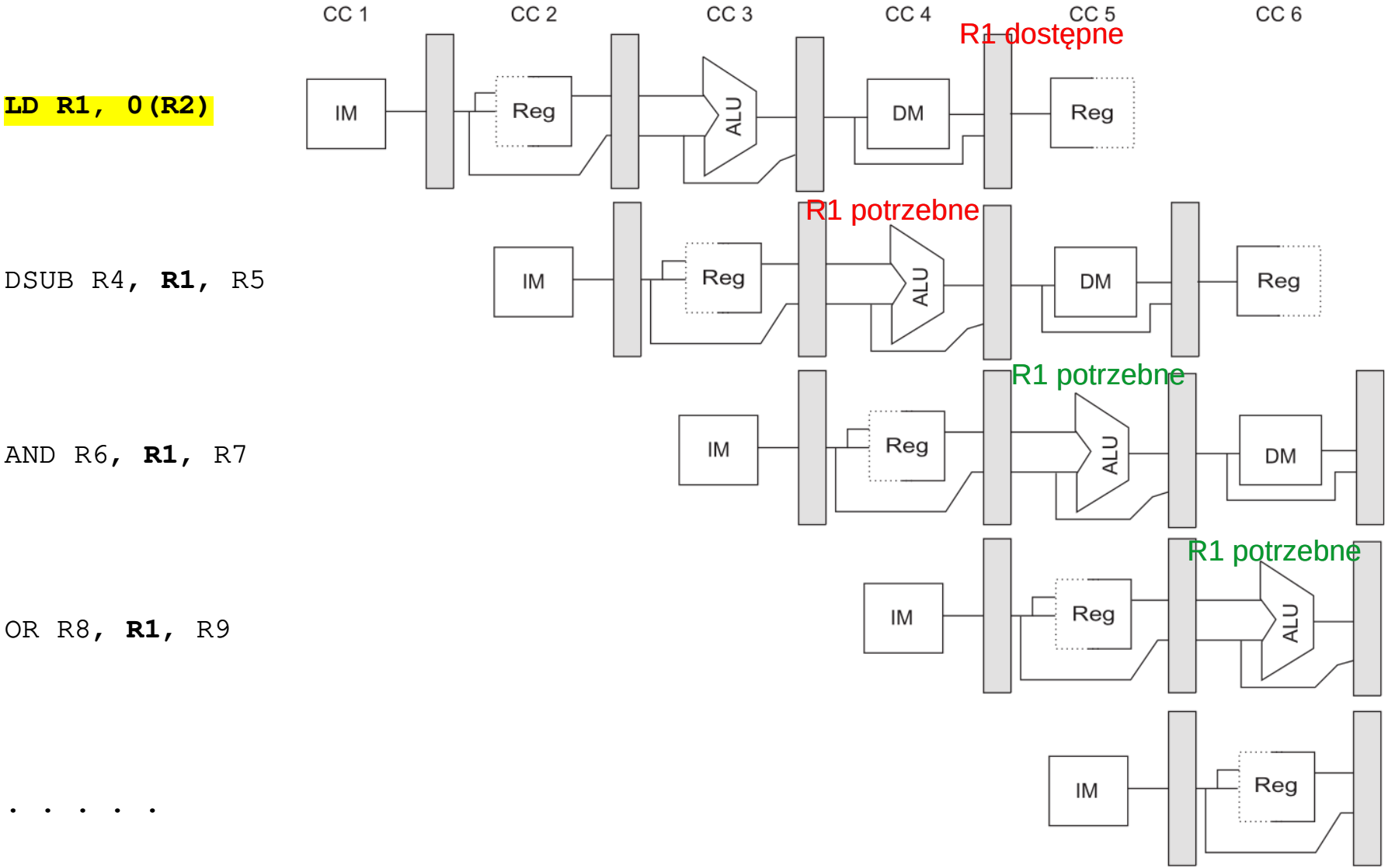
	1	2	3	4	5	6	7
R1	**FI**	**DI**	**FO**	**EI**	**WO**		
R2		**FI**	**DI**	**stall**	**stall**	**FO**	**EI**
R3			**FI**			**DI**	**CO**

Pobranie poprawnego argumentu dla `SUB` (rejestr **EAX**) jest możliwe dopiero po wykonaniu WO przez rozkaz `ADD`.

Zagrożenie danych (2)

- RAW – read after write
  - problem: odczyt przed końcem zapisu
  - „prawdziwe” zagrożenie
- WAR – write after read
  - problem: zapis przed końcem odczytu
- WAW – write after write
  - problem: kto zapisze ostatni...


CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
DADD R1, R2, R3
DSUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
IM
Reg
ALU
DM
Reg
zapis R1
odczyt R1
odczyt R1
odczyt R1
odczyt R1


CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
LD R1, 0(R2)
IM
Reg
ALU
DM
Reg
R1 dostępne
DSUB R4, R1, R5
IM
Reg
ALU
DM
Reg
R1 potrzebne
AND R6, R1, R7
IM
Reg
ALU
DM
R1 potrzebne
OR R8, R1, R9
IM
Reg
ALU
R1 potrzebne
. . . . .
IM
Reg

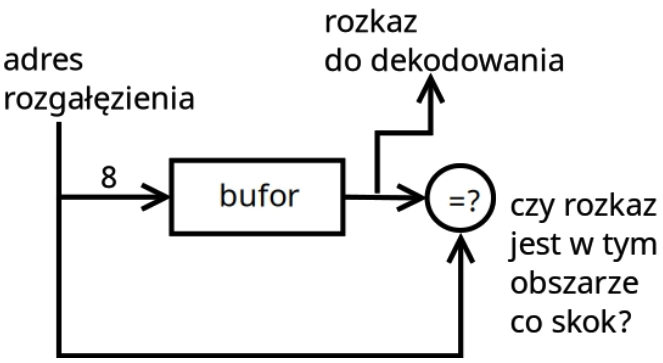
Zagrożenia sterowania

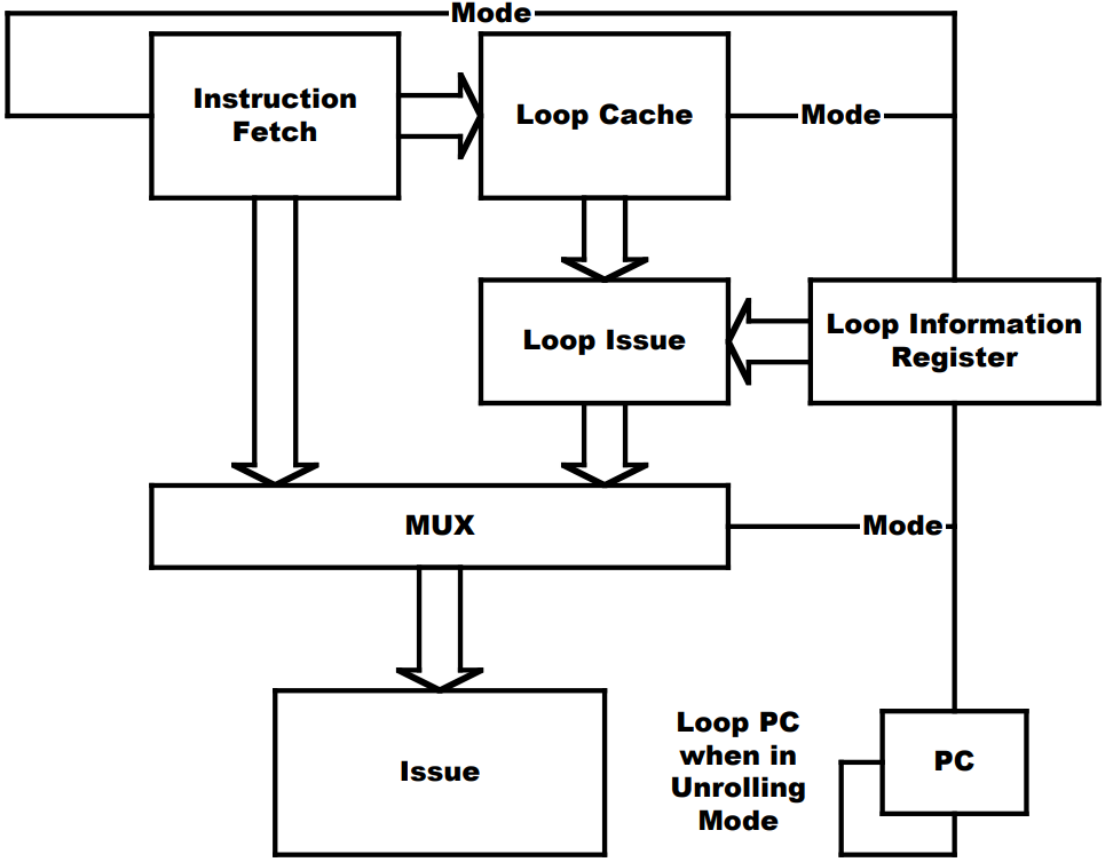
- Wykonanie rozkazu poza kolejnością
- Wykonanie rozkazu nieprzeznaczonego do wykonania
- Obsługa rozgałęzień
  - łagodzenie skutków błędnej decyzji
  - predykcja skoków

Problem skoku w potoku

- Wiele strumieni
  - alternatywne potoki dla skoku wykonanego i nie
- Pobieranie z wyprzedzeniem
  - oprócz celu skoku pobierany jest też kolejny rozkaz
- Bufor pętli
- Przewidywanie rozgałęzień

Bufor pętli





Hardware Loop Buffering Scott DiPasquale, Khaled Elmeleegy, C.J. Ganier, Erik Swanson

```
for(i=999;i>=0;i=i-1)
    x[i] = x[i] + s;
```

Rola kompilatora (1)

```
Loop: fld     f0, 0(x1)
      fadd.d  f4, f0, f2
      fsd     f4, 0(x1)
      addi    x1, x1, -8
      bne     x1, x2, Loop
```

```
Loop: fld     f0, 0(x1)
      // stall
      fadd.d  f4, f0, f2
      // stall
      // stall
      fsd     f4, 0(x1)
      addi    x1, x1, -8
      // stall
      bne     x1, x2, Loop
```

```
Loop: fld     f0, 0(x1)
      addi    x1, x1, -8
      fadd.d  f4, f0, f2
      // stall
      // stall
      fsd     f4, 8(x1)
      bne     x1, x2, Loop
```

Rola kompilatora (2)

```
L: fld    f0, 0(x1)
   fadd.d f4, f0, f2
   fsd    f4, 0(x1)
   addi   x1, x1, -8
   bne    x1, x2, L
```

```
L: fld    f0,  0(x1)
   fadd.d f4,  f0, f2
   fsd    f4,  0(x1)
   fld    f6, -8(x1)
   fadd.d f8,  f6, f2
   fsd    f8, -8(x1)
   fld    f0, -16(x1)
   fadd.d f12, f0, f2
   fsd    f12,-16(x1)
   fld    f14,-24(x1)
   fadd.d f16, f14, f2
   fsd    f16,-24(x1)
   addi   x1,  x1, -32
   bne    x1,  x2, L
```

```
L: fld    f0,  0(x1)
   fld    f6, -8(x1)
   fld    f0, -16(x1)
   fld    f14,-24(x1)
   fadd.d f4,  f0, f2
   fadd.d f8,  f6, f2
   fadd.d f12, f0, f2
   fadd.d f16, f14, f2
   fsd    f4,  0(x1)
   fsd    f8, -8(x1)
   fsd    f12,-16(x1)
   fsd    f16,-24(x1)
   addi   x1,  x1, -32
   bne    x1,  x2, L
```

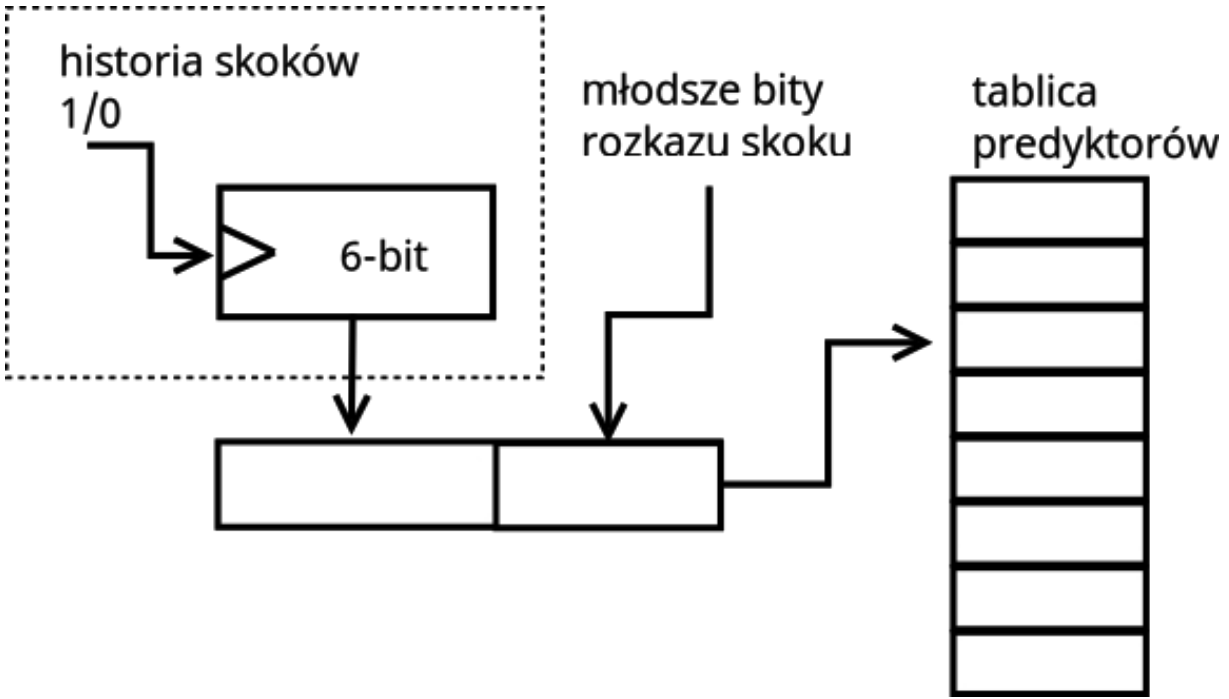
Przewidywanie rozgałęzień

- Nigdy nie nastąpi
- Zawsze nastąpi
- To zależy, jaka operacja…
- Co było krok wcześniej
- Większa historia

Predyktory dynamiczne

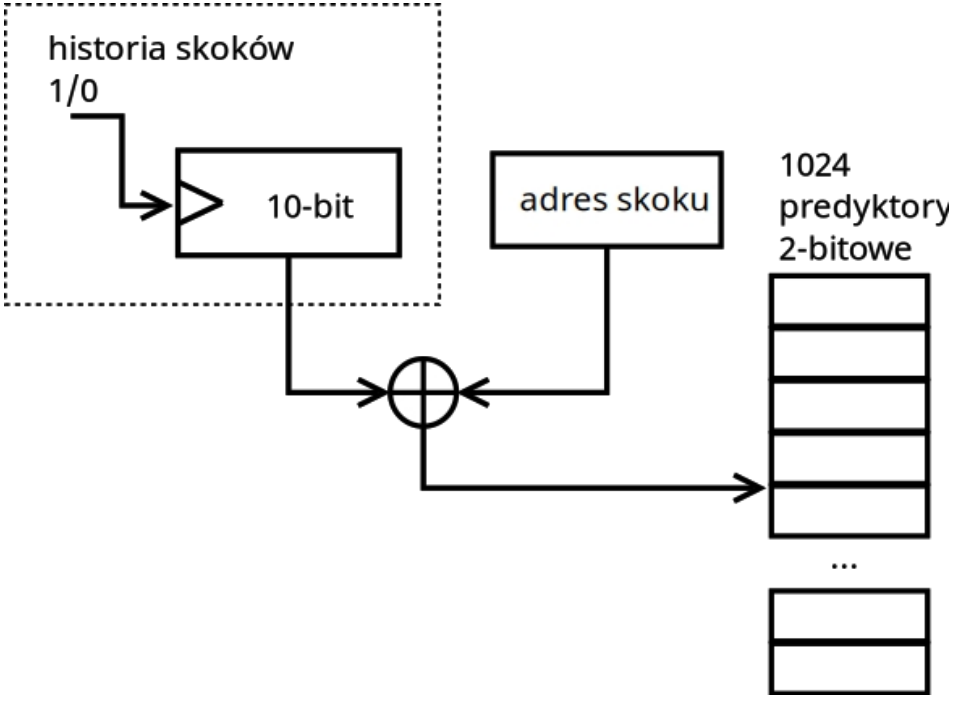
- Podstawowy, jednobitowy
  - liczy się tylko historia ostatniego skoku
  - prosty problem: pętla
- Dwubitowy
  - nie tak łatwo mnie zmylić…
  - podstawa dla innych predyktorów

Predyktor korelujący




```
if(a == 2)
    a = 0;
if(b == 2)
    b = 0;
if(a != b)
    ...
```

Predyktor gshare McFarlinga



Predyktor turniejowy