



---

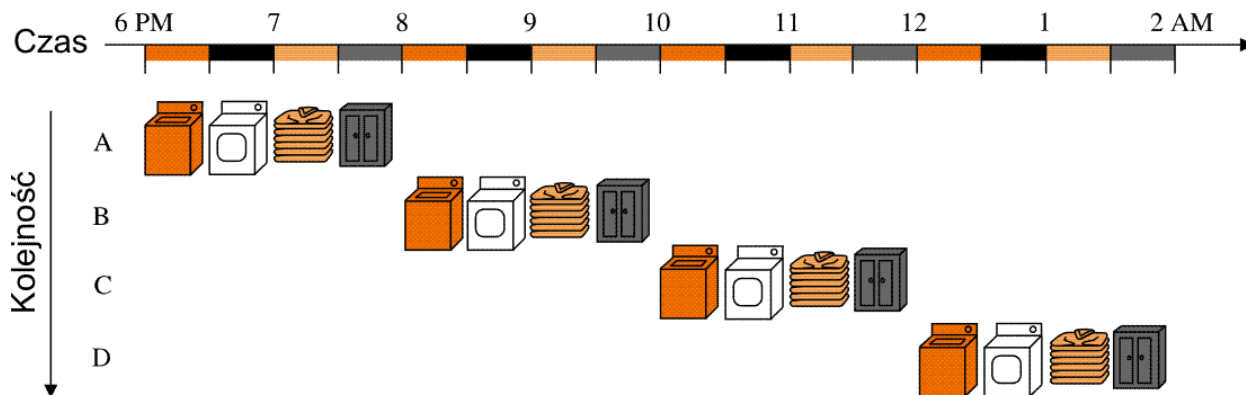
# *Architektura potokowa*

## *RISC*

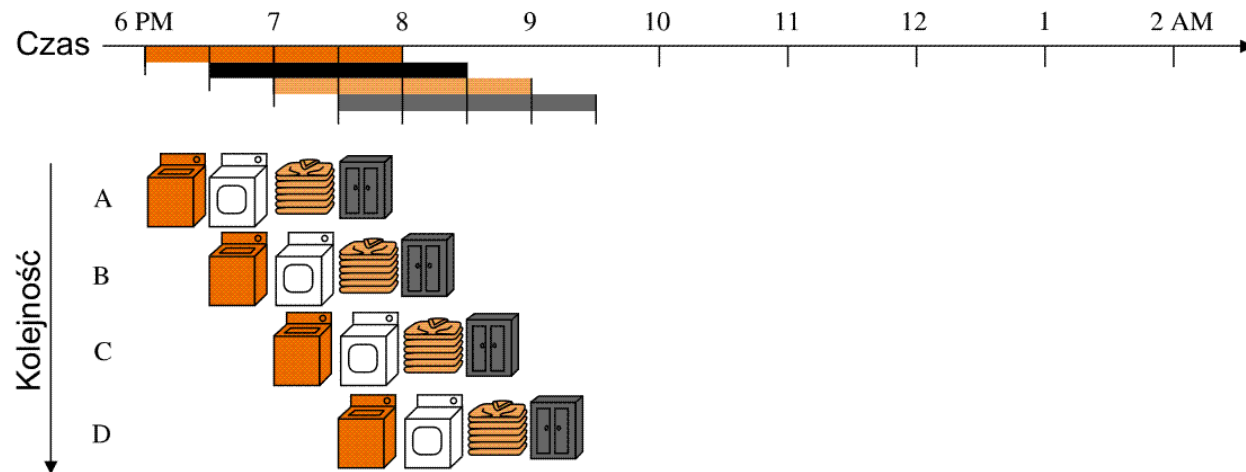
# Przetwarzanie szeregowe i potokowe

- Podział zadania na odrębne części i niezależny sprzęt
- Brak „nawrotów” podczas pracy

szeregowe

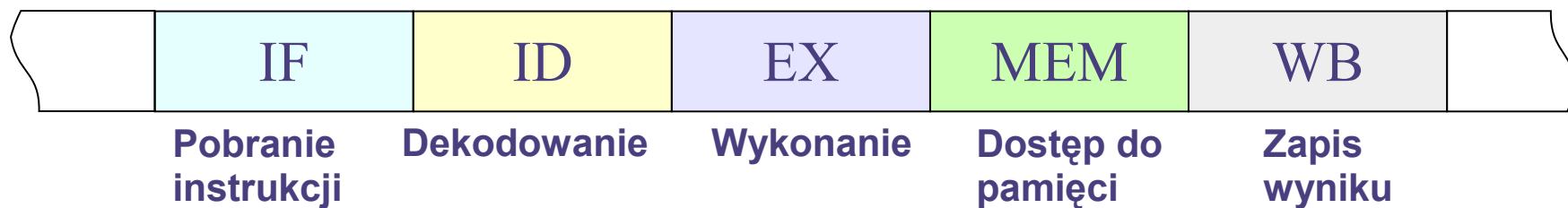


potokowe



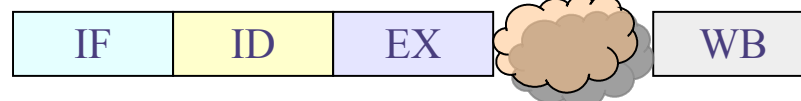
# Podział instrukcji na fazy wykonania

- Każda instrukcja musi mieć te same fazy wykonania:



- Jeśli dana instrukcja nie wykorzystuje wszystkich faz, procesor wykonuje puste cykle zegara dla tej instrukcji

Instrukcje operujące na rejestrach



Instrukcje zapisu rejestru do pamięci



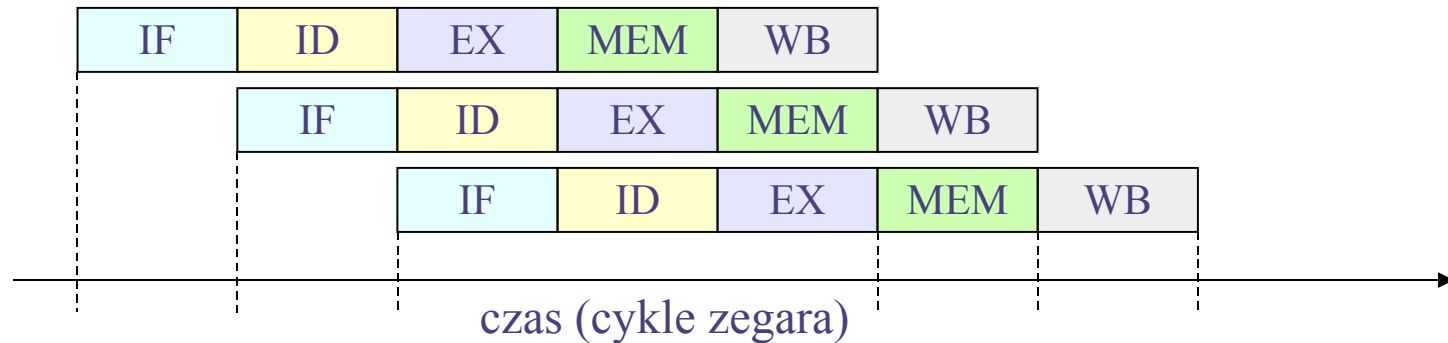
Instrukcje odczytu pamięci do rejestru



Instrukcje skoku

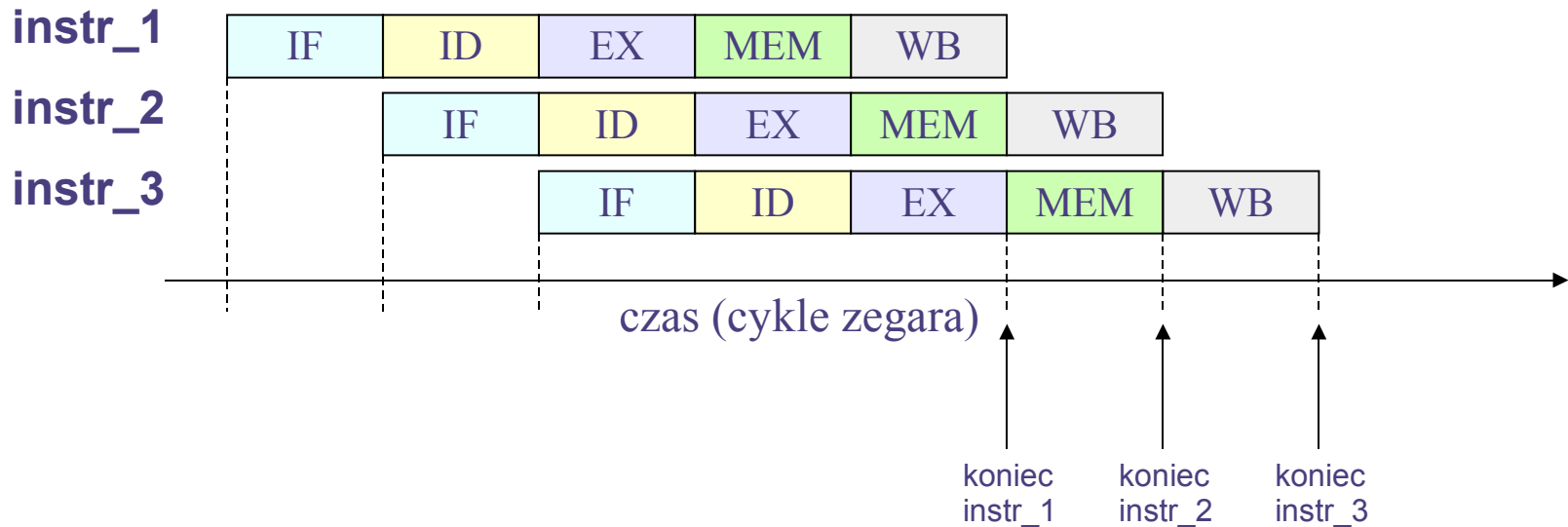


# Wykorzystanie zasobów sprzętowych



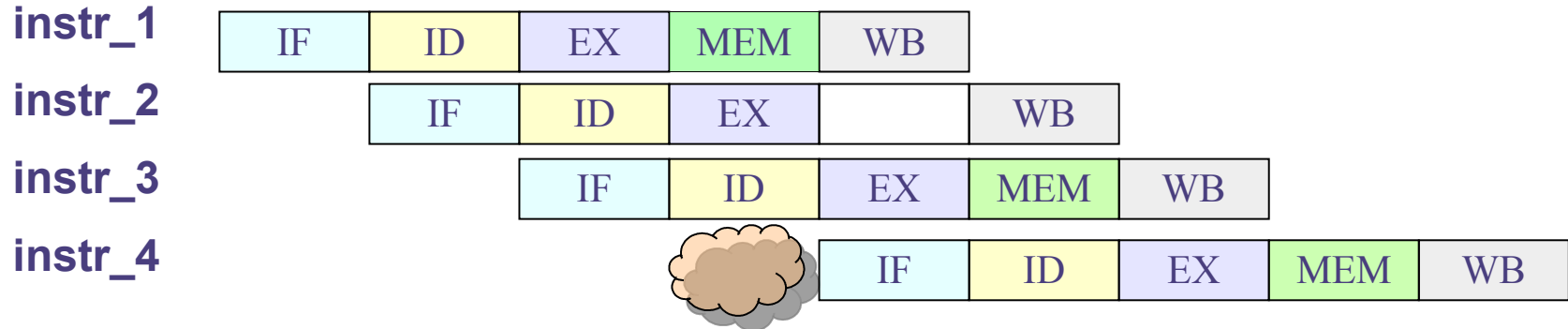
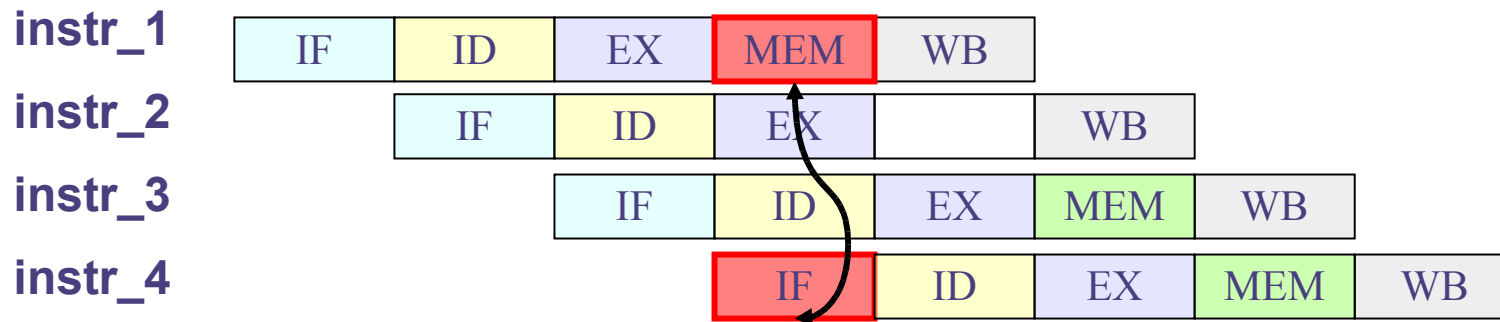
- ▶ Każda faza instrukcji wykorzystuje inne zasoby procesora
  - IF – zapis do rejestru IR i zwiększanie PC,
  - ID – spekulatywne odczytywanie rejestrów potrzebnych do operacji
  - EX – wykorzystanie ALU i zapis wyniku do rejestru buforowego
  - MEM – odczyt lub zapis do pamięci (do lub z rejestrów buforowych)
  - WB – zapis wyniku z rejestru buforowego do rejestru procesora
- ▶ Wszystkie zasoby procesora mogą być wykorzystane równocześnie przez różne instrukcje
- ▶ W danym cyklu, procesor wykonuje jednocześnie kilka instrukcji, ale każda znajduje się w innym stopniu zaawansowania.

# Wydajność przetwarzania potokowego



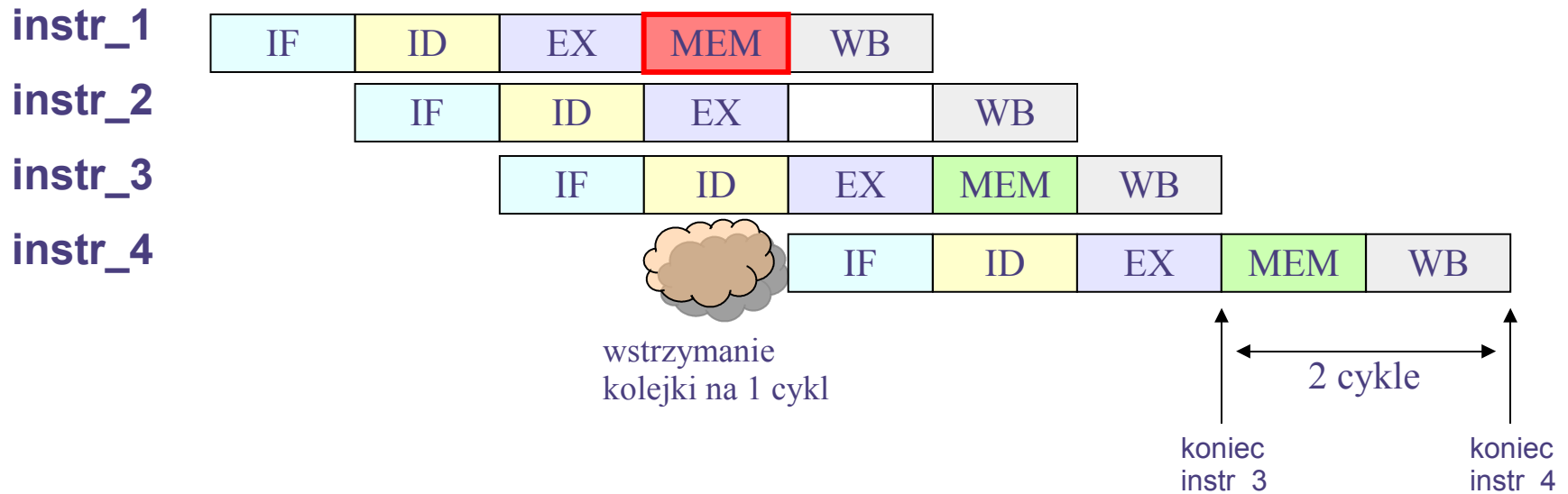
- Co 1 cykl zegara procesor kończy wykonywanie instrukcji, choć każda instrukcja nadal wykonuje się kilka cykli.
- Przyspieszenie w stosunku do architektury tradycyjnej jest proporcjonalne do liczby faz instrukcji (np. na rysunku powyżej 5 x szybciej)

# Konflikt zasobów (*Structural Hazard*)



- Konflikt zasobów występuje gdy, dwie instrukcje wymagają dostępu do tych samych zasobów procesora  
 (np. dostęp do pamięci: faza IF – odczyt instrukcji, faza MEM – odczyt lub zapis danych)

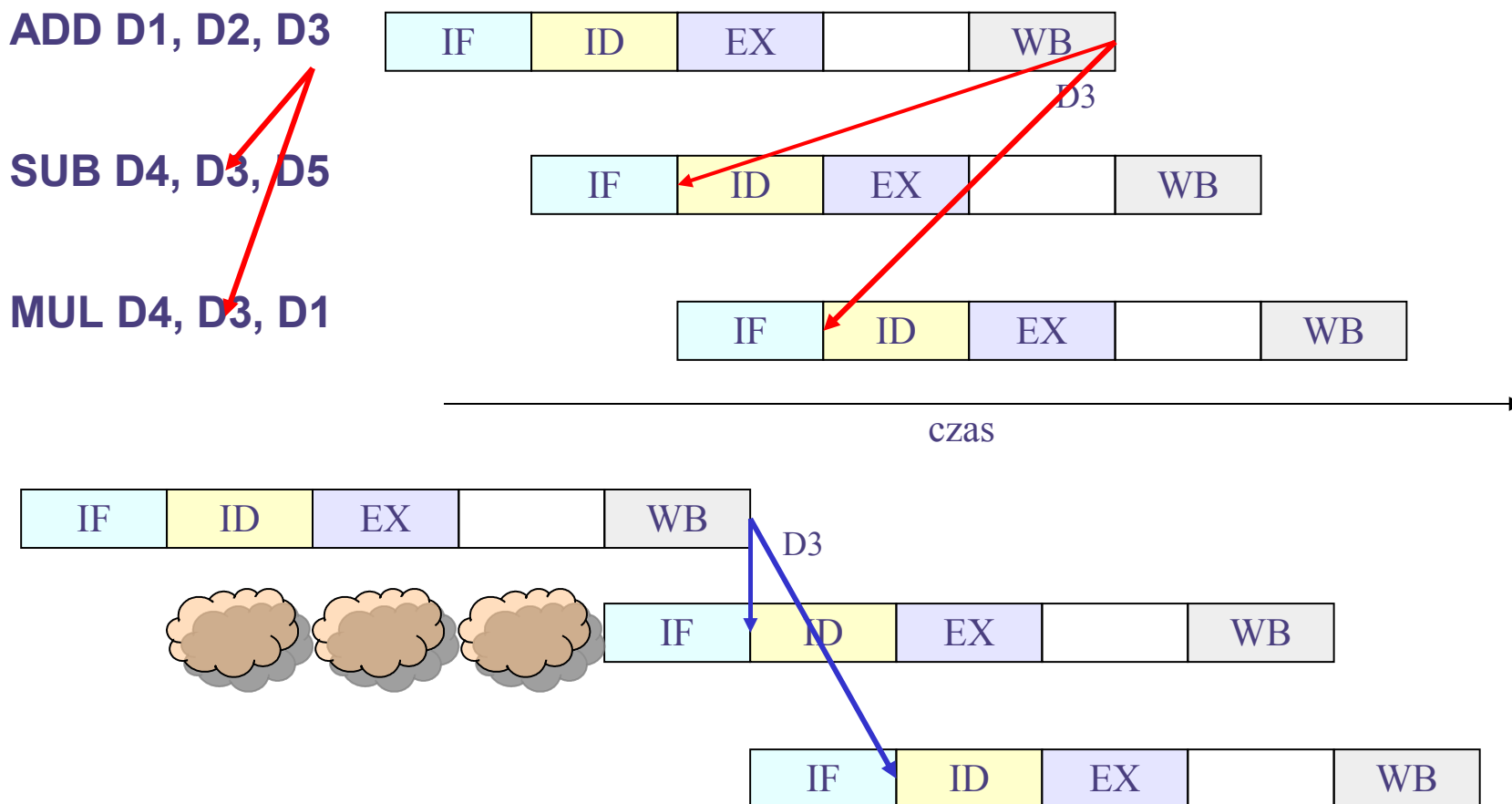
# Konflikt zasobów



- Rozwiązanie konfliktu zasobów polega na:
  - ◆ podwojeniu zasobów  
(np. oddzielna pamięć programu (faza IF) i danych (faza MEM))
  - ◆ wstrzymaniu kolejki  
(rozwiązanie najprostsze, ale powoduje spadek szybkość przetwarzania)

# Konflikt danych (*Data Hazard*)

Jeśli dwie kolejne instrukcje wykonują operacje na tych samych rejestrach, może zaistnieć sytuacja gdy jedna instrukcja potrzebuje wartości rejestru, który jest wynikiem obliczeń poprzedniej instrukcji i nie jest jeszcze dostępny.



Wstrzymywanie kolejki jest złym rozwiązaniem, gdyż konflikty danych zdarzają się bardzo często

# Konflikt danych - *Forwarding*

ADD D1, D2, D3



SUB D4, D3, D5



MUL D4, D3, D1



- ▶ *Forwarding* polega na przesyłaniu wyników działania z faz późniejszych do wcześniejszych, bez czekania na ich ostateczny zapis
- ▶ *Forwarding* realizowany jest poprzez sprzętowe porównywanie numerów rejestrów biorących udział w poszczególnych fazach instrukcji.

# Optymalizacja kodu

- Konflikty danych mogą być eliminowane przez optymalizację kodu (*optimising compilers*)

Procedura zamieniająca sąsiadujące elementy w tablicy:

$X[k]$  i  $X[k+1]$

Rejestr R3 zawiera adres elementu  $X[k]$ .

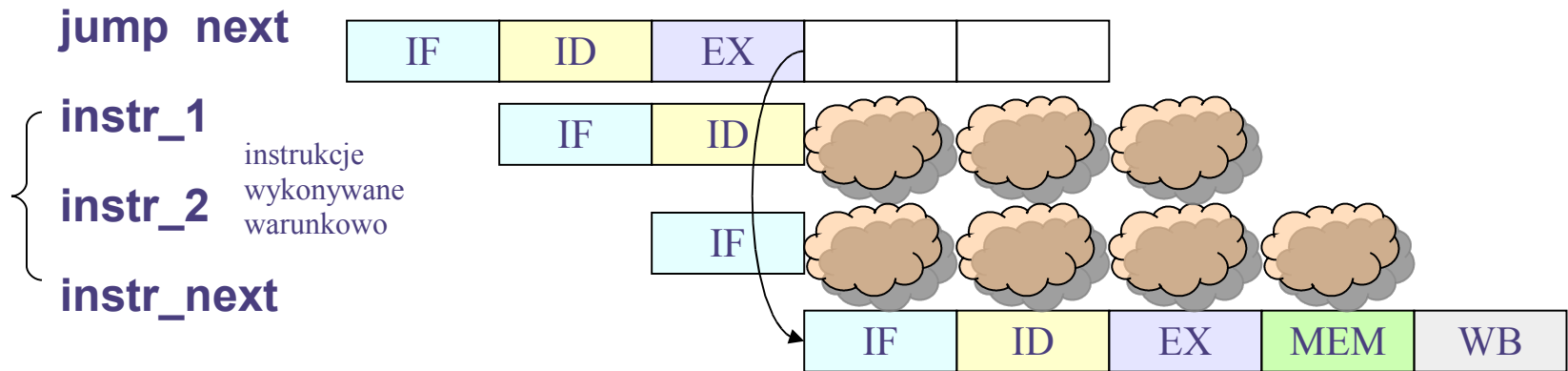
Przed optymalizacją:

```
lw R1,0(R3) * R1 = X[k]
lw R2,4(R3) * R2 = X[k+1]
sw R2,0(R3) * X[k] = R2
sw R1,4(R3) * X[k+1] = R1
```

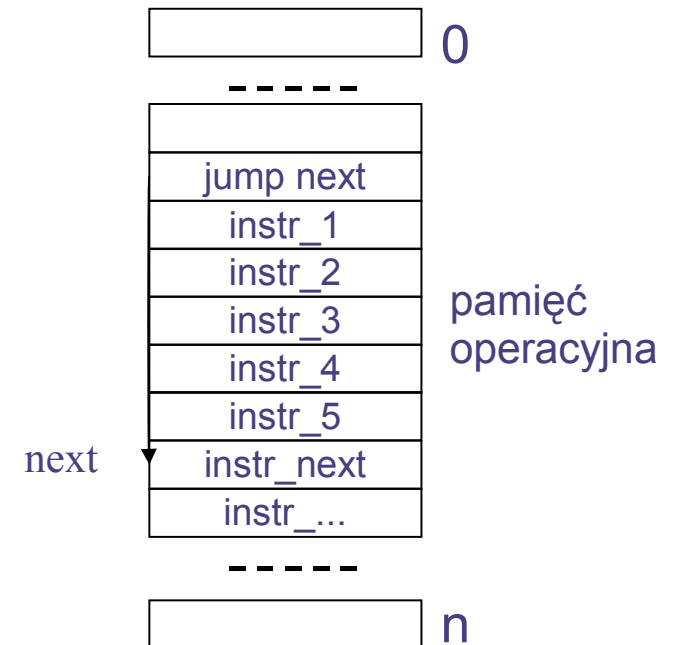
Po optymalizacji:

```
lw R1,0(R3) * R1 = X[k]
lw R2,4(R3) * R2 = X[k+1]
sw R1,4(R3) * X[k+1] = R1
sw R2,0(R3) * X[k] = R2
```

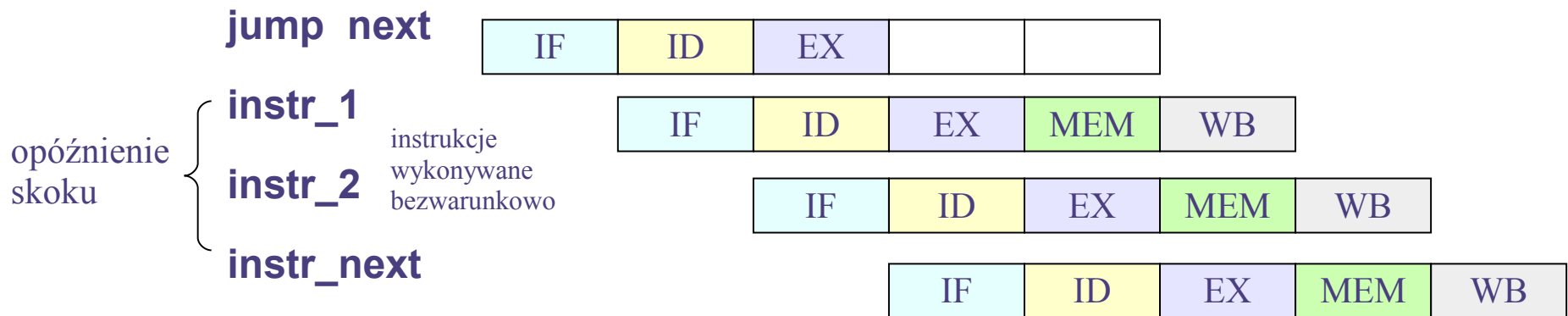
# Konflikt sterowania (*Control Hazard*)



- Każdy skok w programie powoduje zakłócenie kolejki instrukcji, występujących kolejno w pamięci.
- Przy skokach warunkowych, poznanie następnej instrukcji do wykonania jest możliwe dopiero po pewnej liczbie cykli zegara – w tym czasie procesor może rozpocząć wykonywanie kolejnych instrukcji i unieważnić je (lub nie) w zależności od tego czy skok rzeczywiście się wykona czy nie.



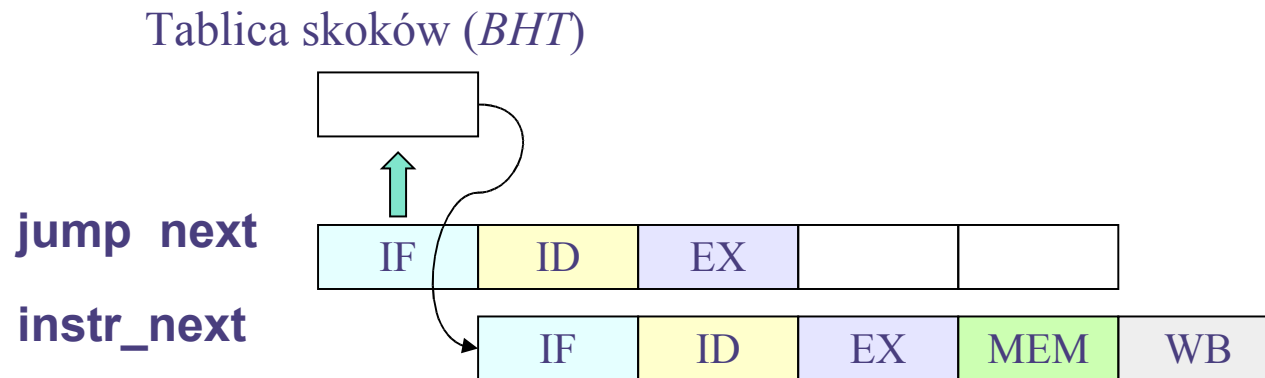
# Opóźnienie skoku (*Delayed Branch*)



Opóźniony skok (*delayed branch*) – instrukcje bezpośrednio po skoku (opóźnienie) są zawsze wykonywane, niezależnie czy skok się wykona czy nie.

Program powinien być tak skonstruowany, aby były to instrukcje niezależne od skoku, albo w najgorszym przypadku instrukcje puste NOP (no operation)

# Tablica skoków (*Branch History Table*)



Tablica skoków (*BHT*) –

zapamiętywane są ostatnio wykonywane instrukcje skoków i adresy, do których skoki zostały wykonane.

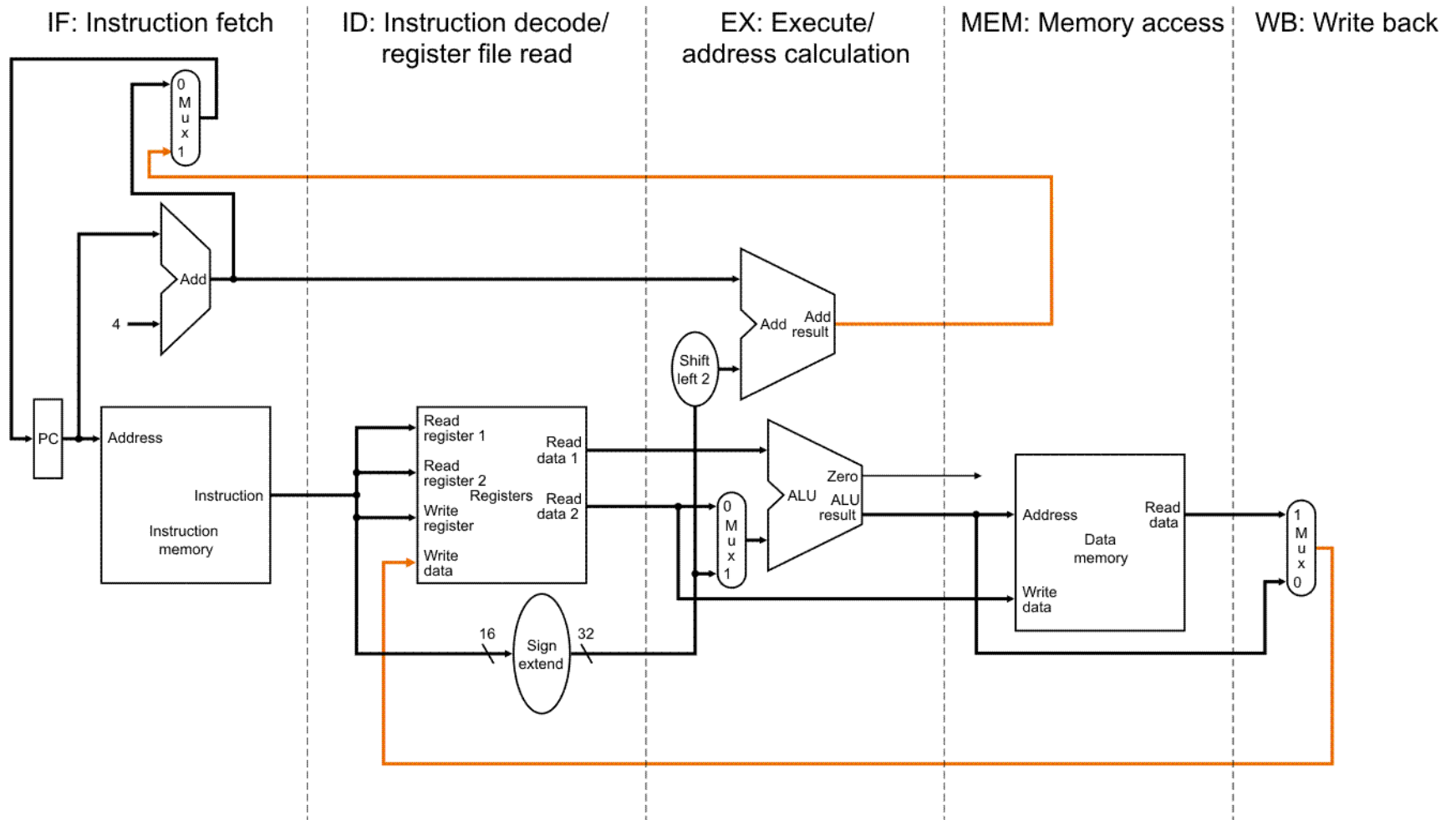
- ◆ Podczas pobieranie instrukcji skoku, adres następnej instrukcji jest odczytywany z tablicy skoków (o ile wcześniej już wystąpił).
- ◆ Metoda pozwala zredukować nietrafione instrukcje. do przypadków zakończenia lub rozpoczęcia pętli w programach, czyli sytuacji stosunkowo rzadkich.

# Cechy przetwarzania potokowego

- Idea: wykonywanie 1 instrukcji w 1 cyklu zegara ( $CPI = 1$ )
- Czas wykonywania pojedynczej instrukcji w przetwarzaniu potokowym nie zmniejsza się i jest kilkakrotnie dłuższy od cyklu zegara, ale procesor efektywnie kończy jedną instrukcję w każdym cyklu
- Konflikty zasobów, danych i sterowania powodują, że rzeczywista wydajność jest mniejsza niż 1instr/1cykl
- Podczas przetwarzania potokowego procesor wykonuje kilka instrukcji jednocześnie, ale w różnych fazach wykonania.
- Podczas przetwarzania potokowego w każdym cyklu zegara wykorzystywane są wszystkie zasoby procesora.
- Szczególne znaczenia ma optymalizacja kodu programu, w celu minimalizacji konfliktów.

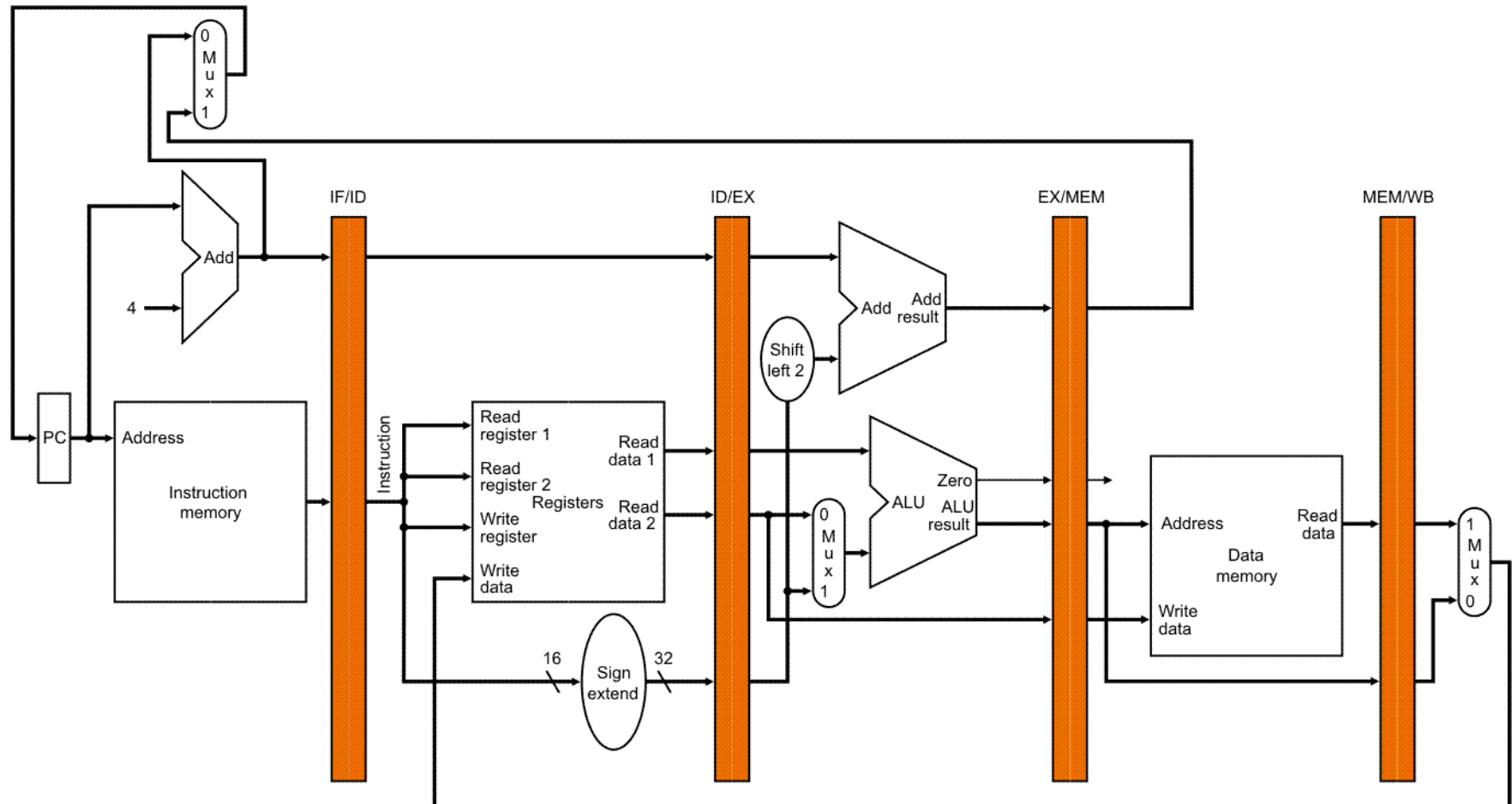
# Przeptyw danych w arch. *Single-Cycle*

- Podział na odrębne fazy i odrębny sprzęt








# Architektura potokowa

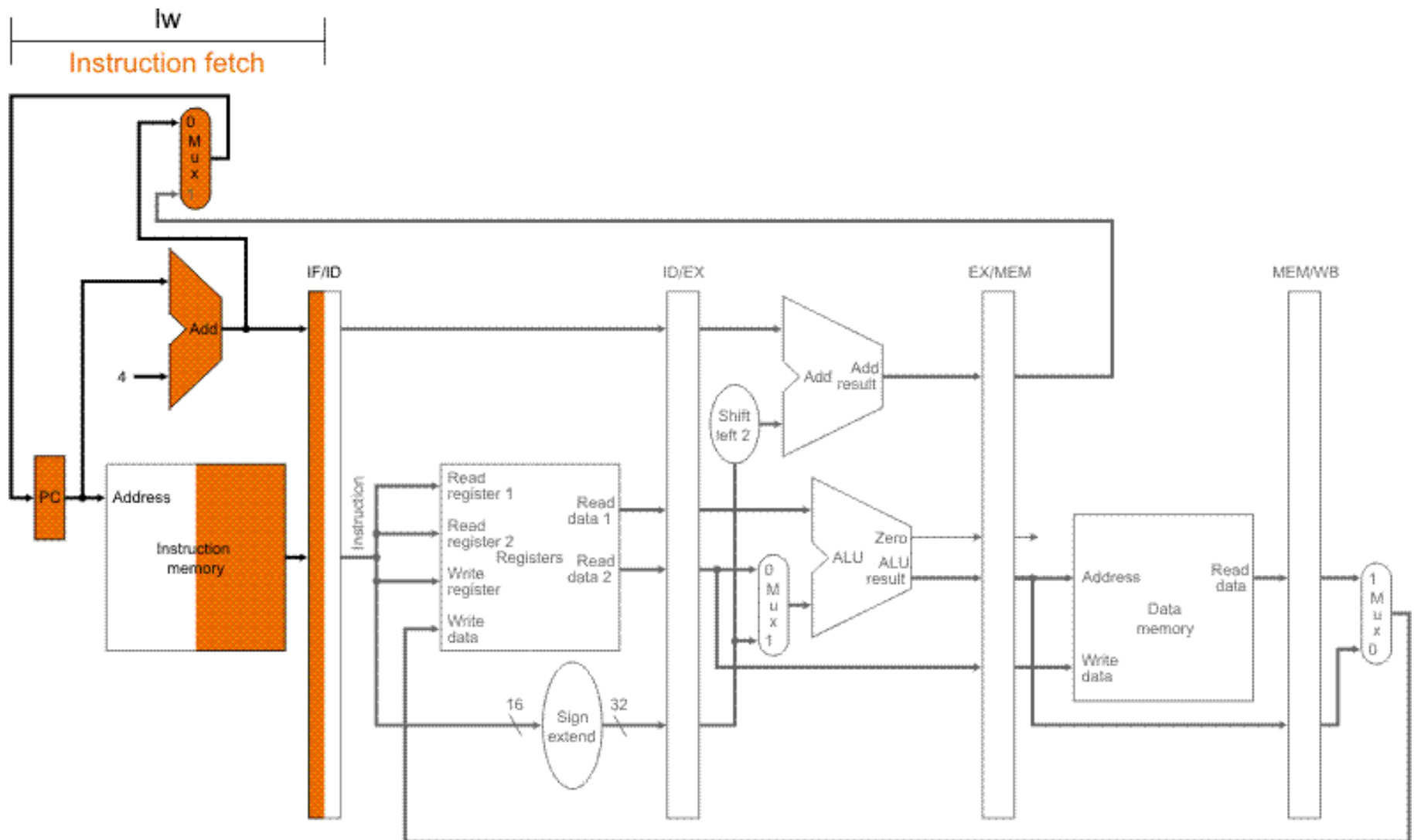
- Rejestry pośrednie do synchronizacji przepływu danych



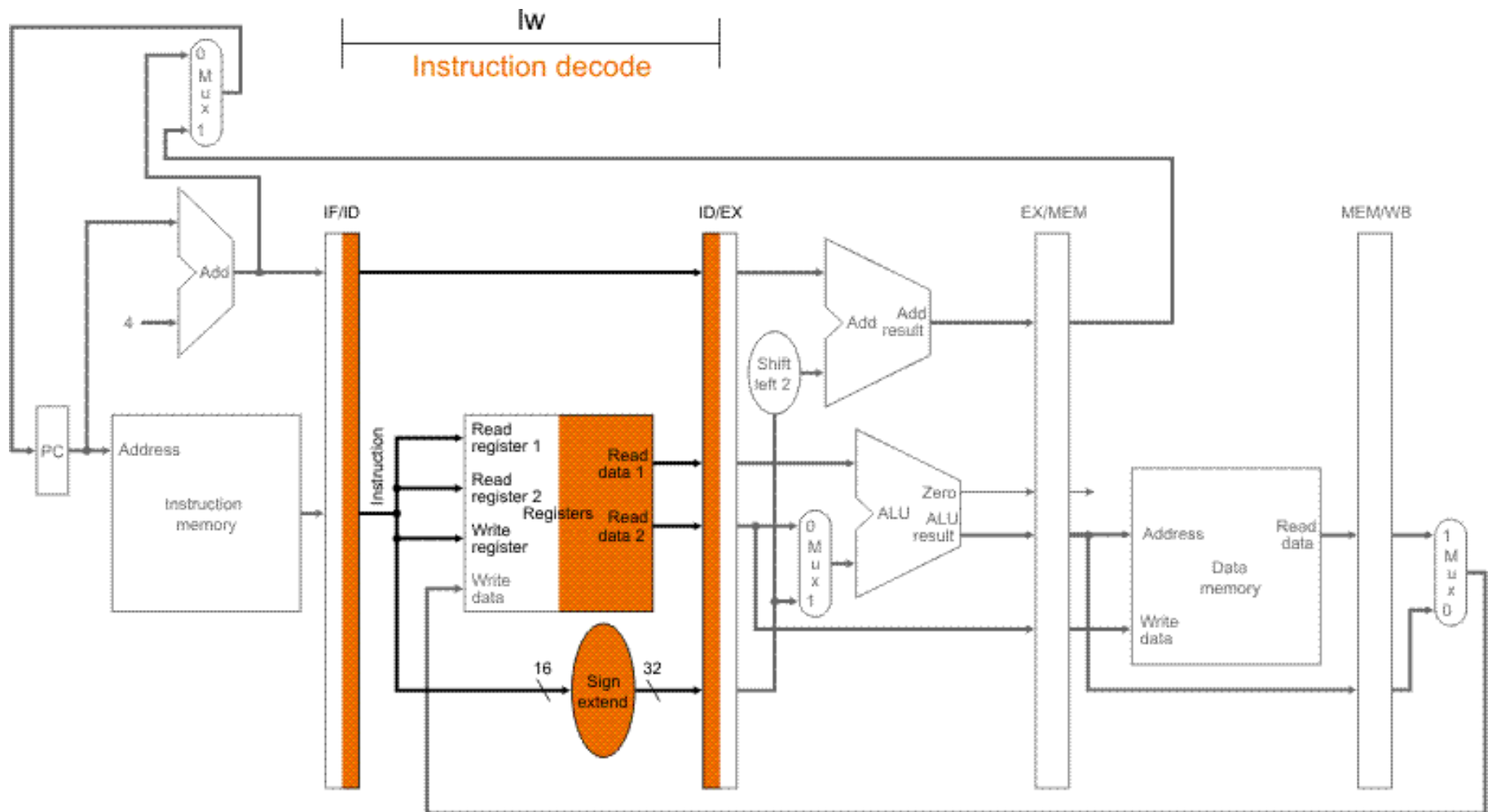
# Cechy architektury potokowej

- 
 Komplikacja sprzętu – powielanie sumatorów i bloków pamięci w celu uniknięcia konfliktów zasobów.
- 
 Wprowadzenie „długich” rejestrów pośrednich przekazujących wszystkie informacje dotyczące wykonywanej instrukcji do następnej fazy.
- 
 Instrukcje wykorzystują po kolei wszystkie zasoby procesora, ale nie mogą korzystać z zasobów spoza aktualnej fazy wykonania. Nie jest np. możliwe wielokrotne wykorzystanie ALU do realizacji bardziej złożonego trybu adresowania.
- 
 Wzrost wydajności odbywa się za cenę uproszczenia zestawu instrukcji.
- 
 Konsekwencją uproszczenia instrukcji jest łatwiejsza realizacja sterowania.

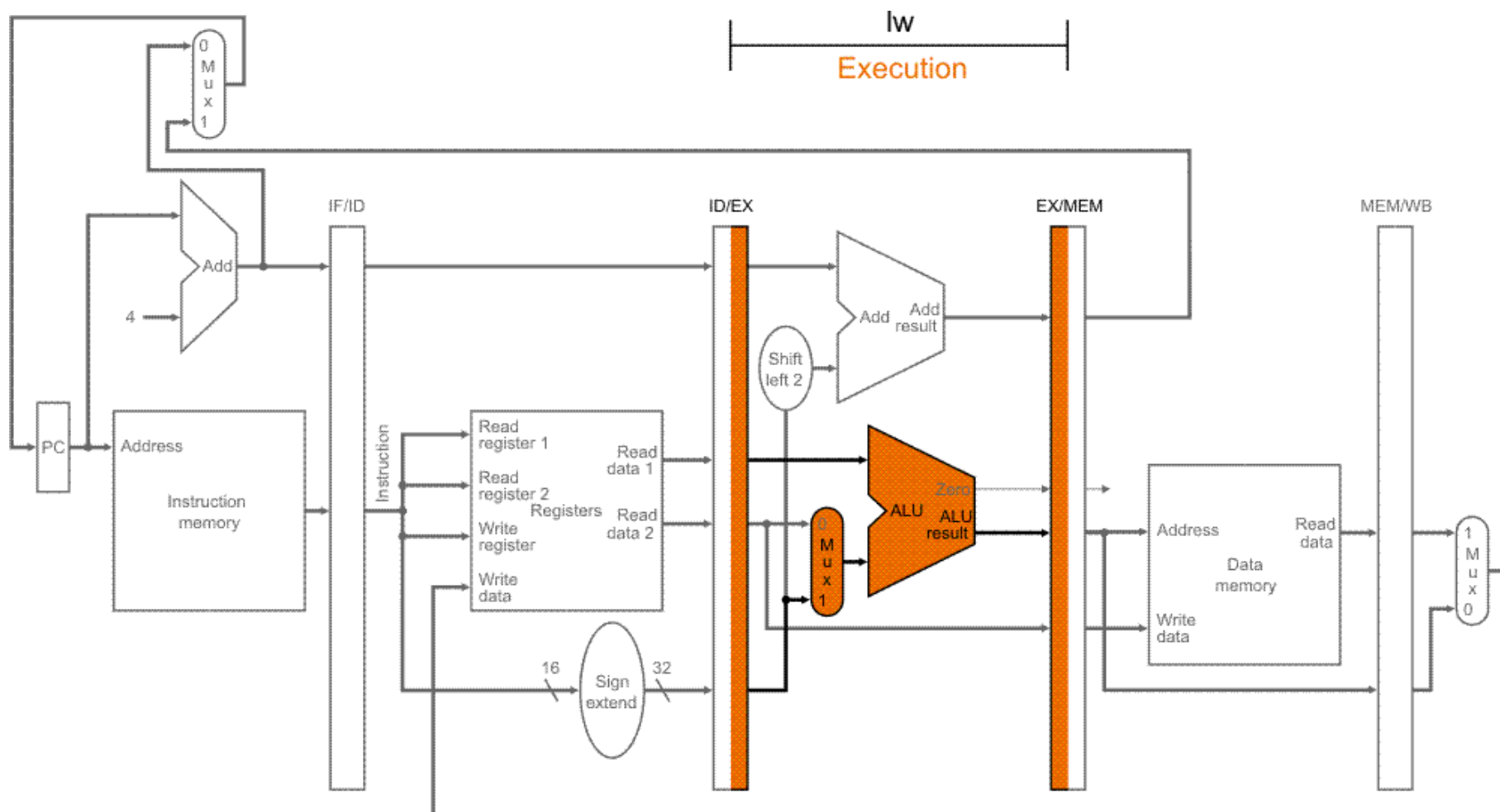
# Load Word (LW) – demonstration (1)



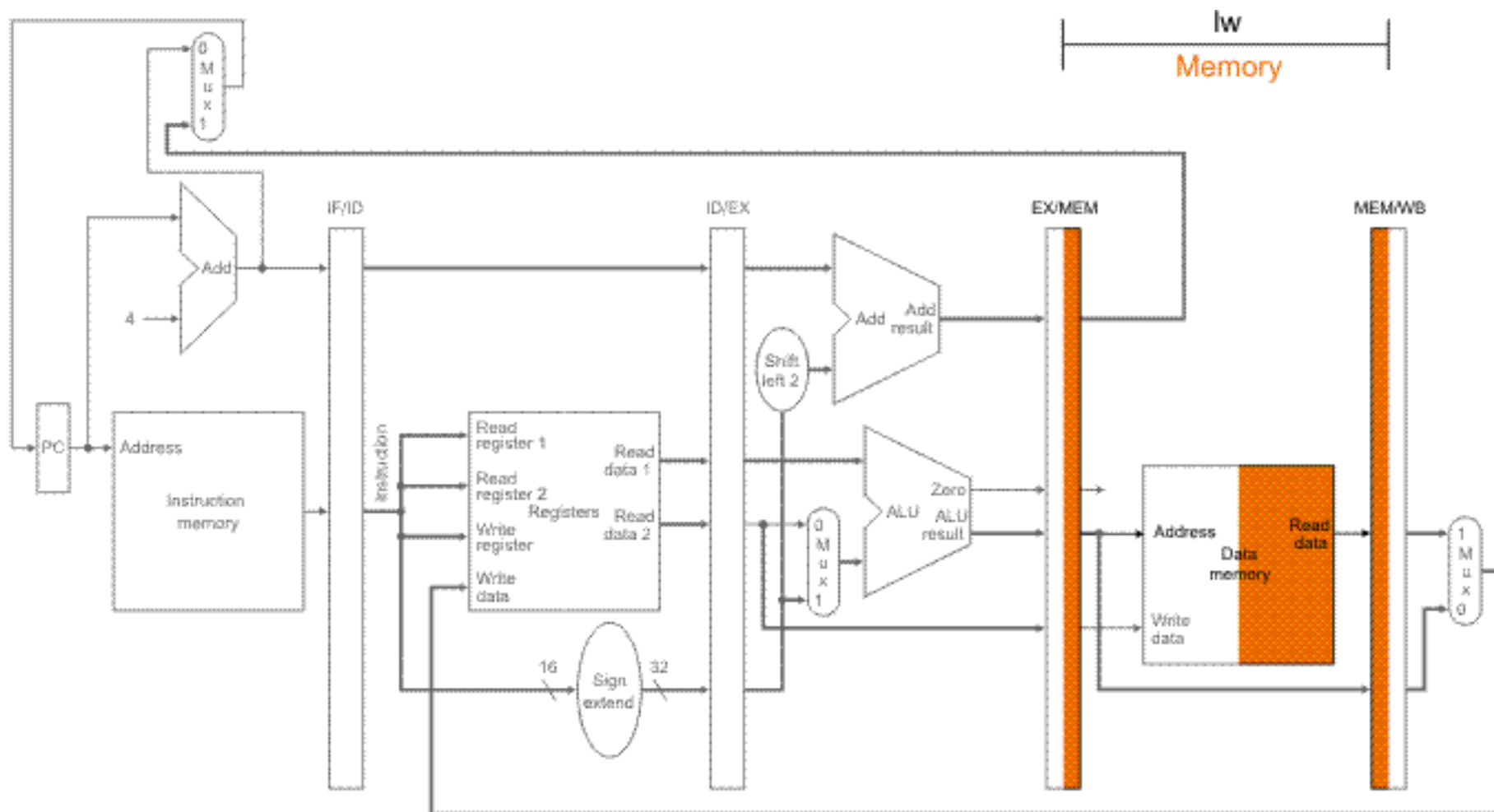
# LW (2)



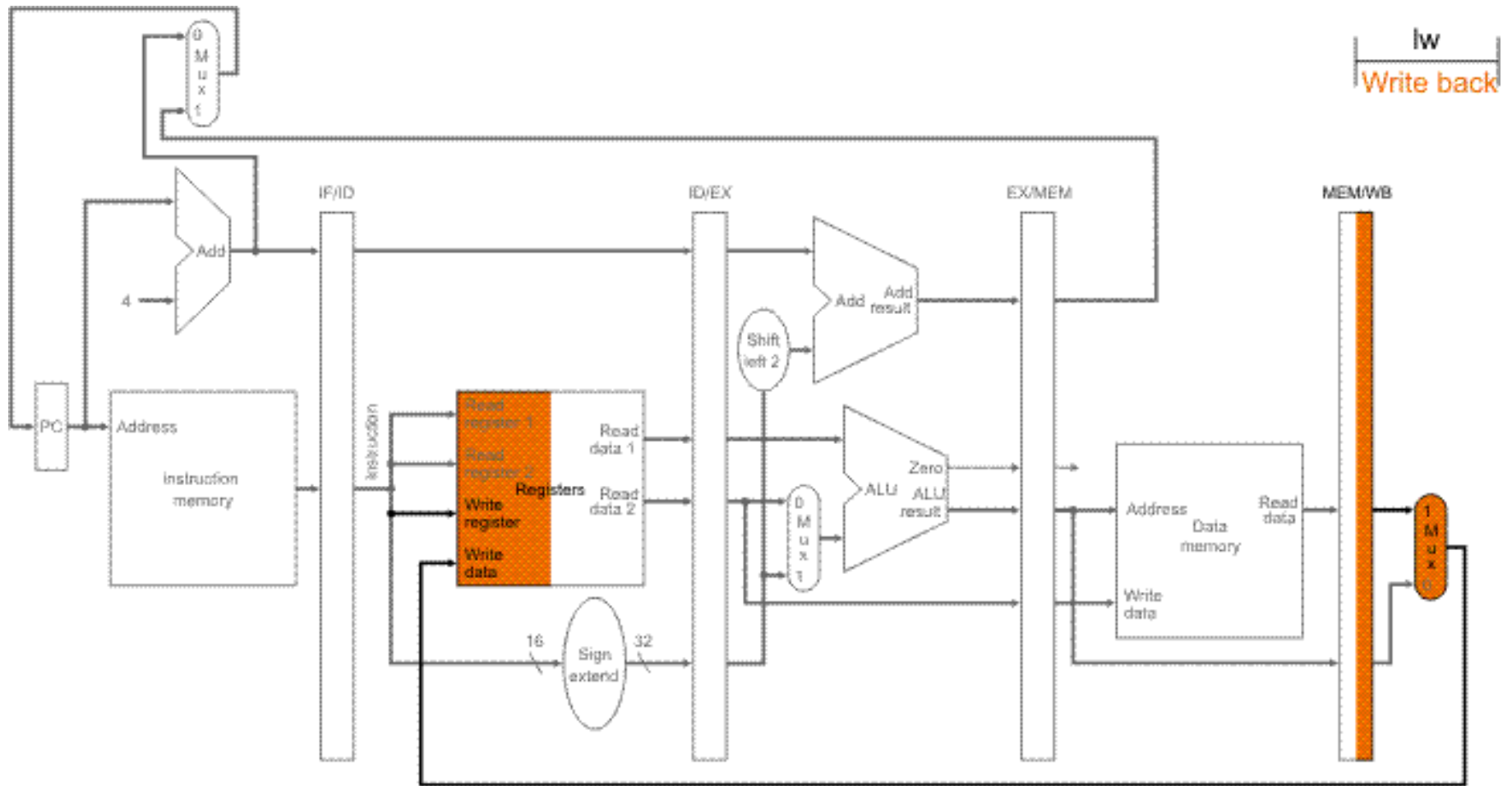
# LW (3)



# LW (4)



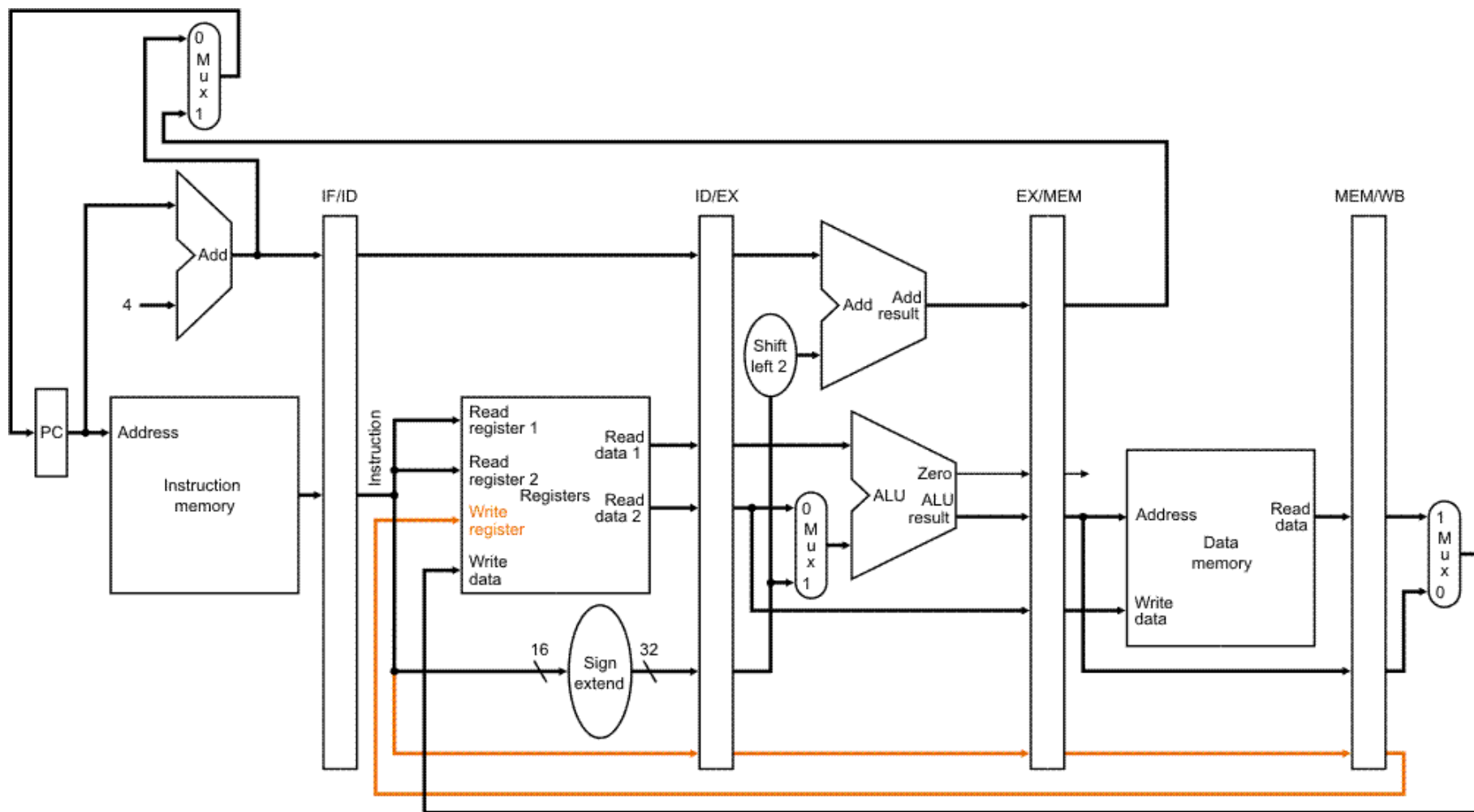
# LW (5)



lw  
Write back

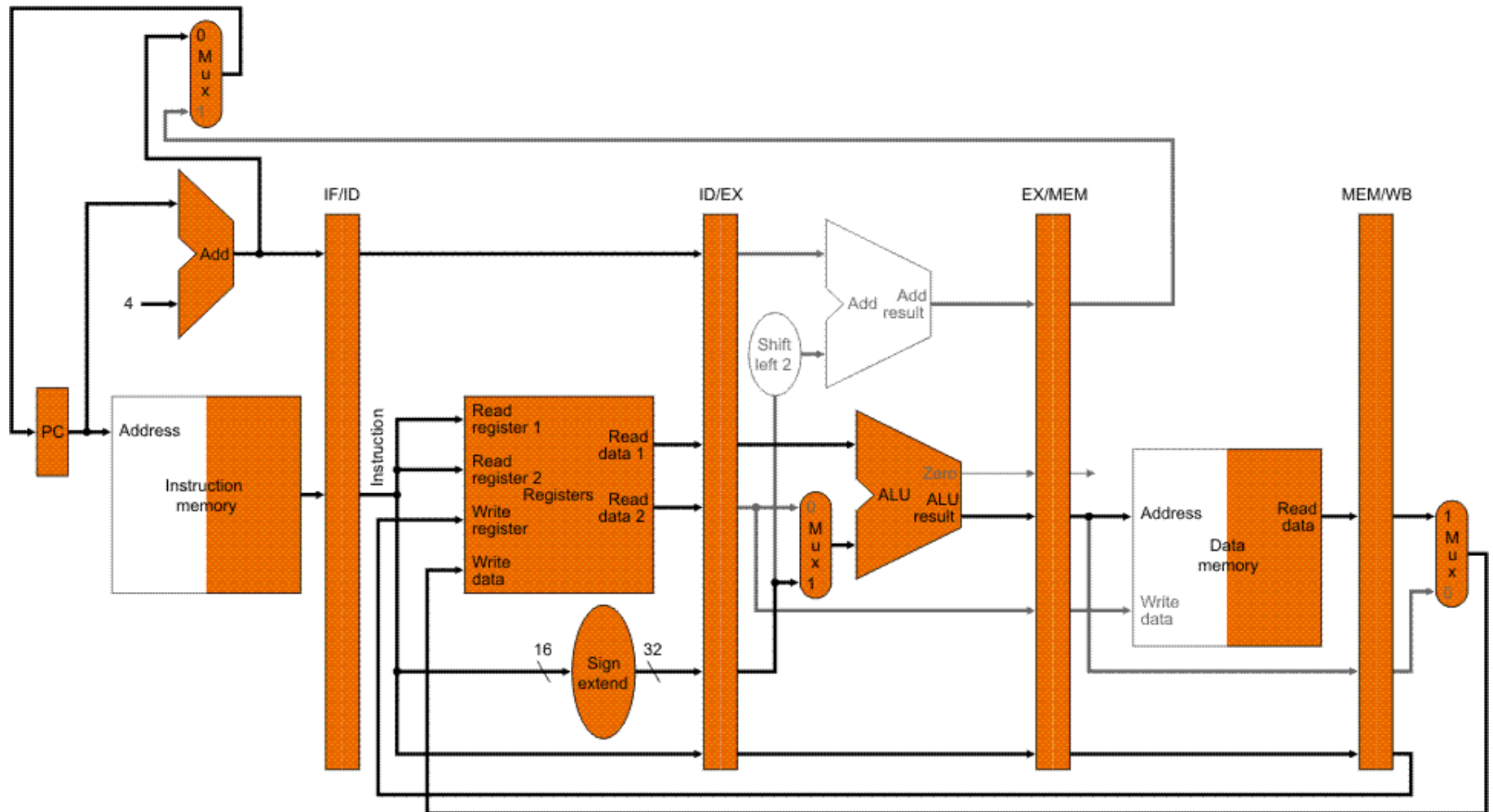
# Korekcja zapisu do rejestru (LW, R-type)

- Instrukcja „niesie” przez potok numer rejestru do modyfikacji w ostatniej fazie

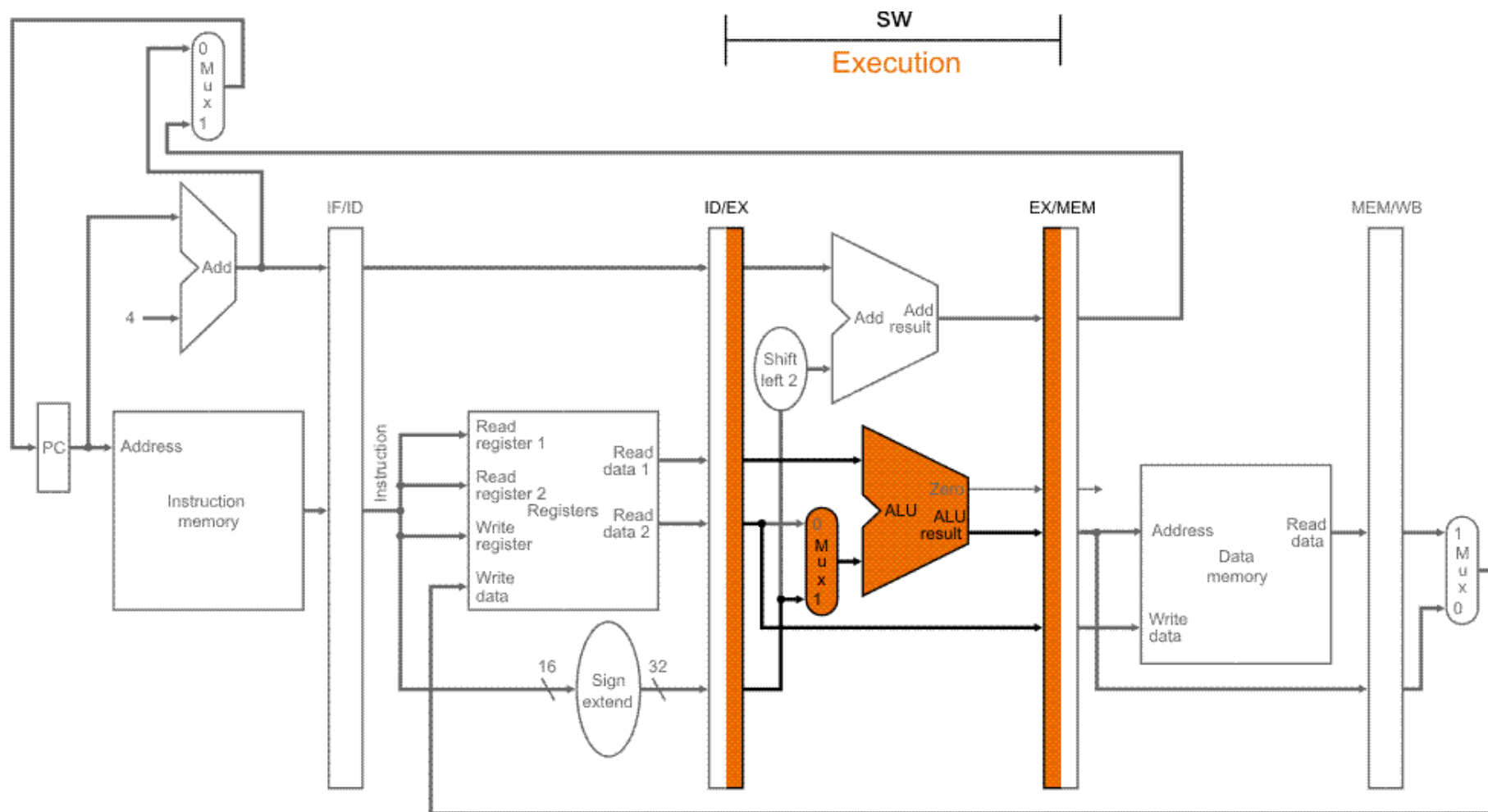


# LW – wykorzystanie zasobów

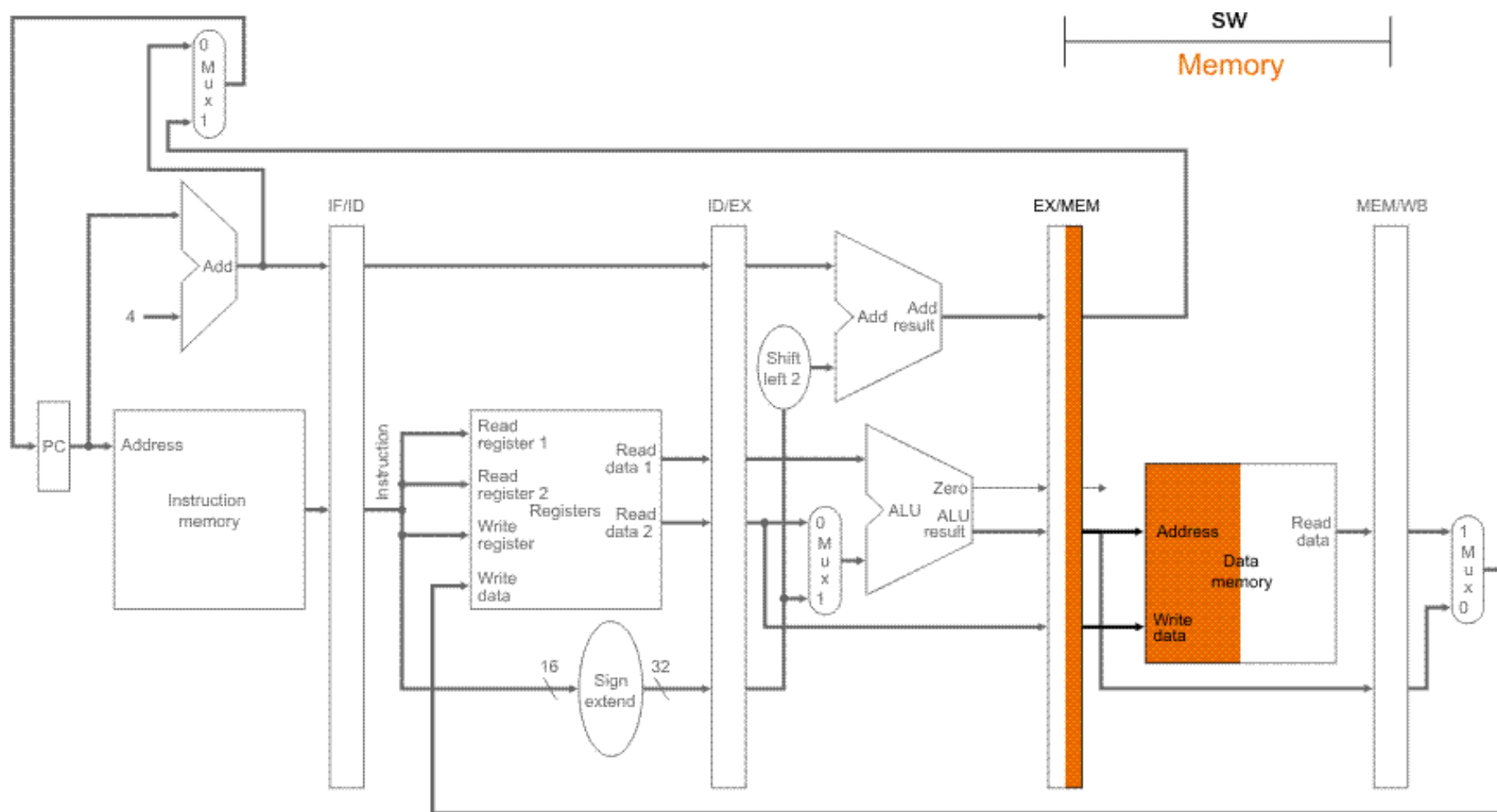
- Wykorzystanie zasobów jest rozłożone w czasie



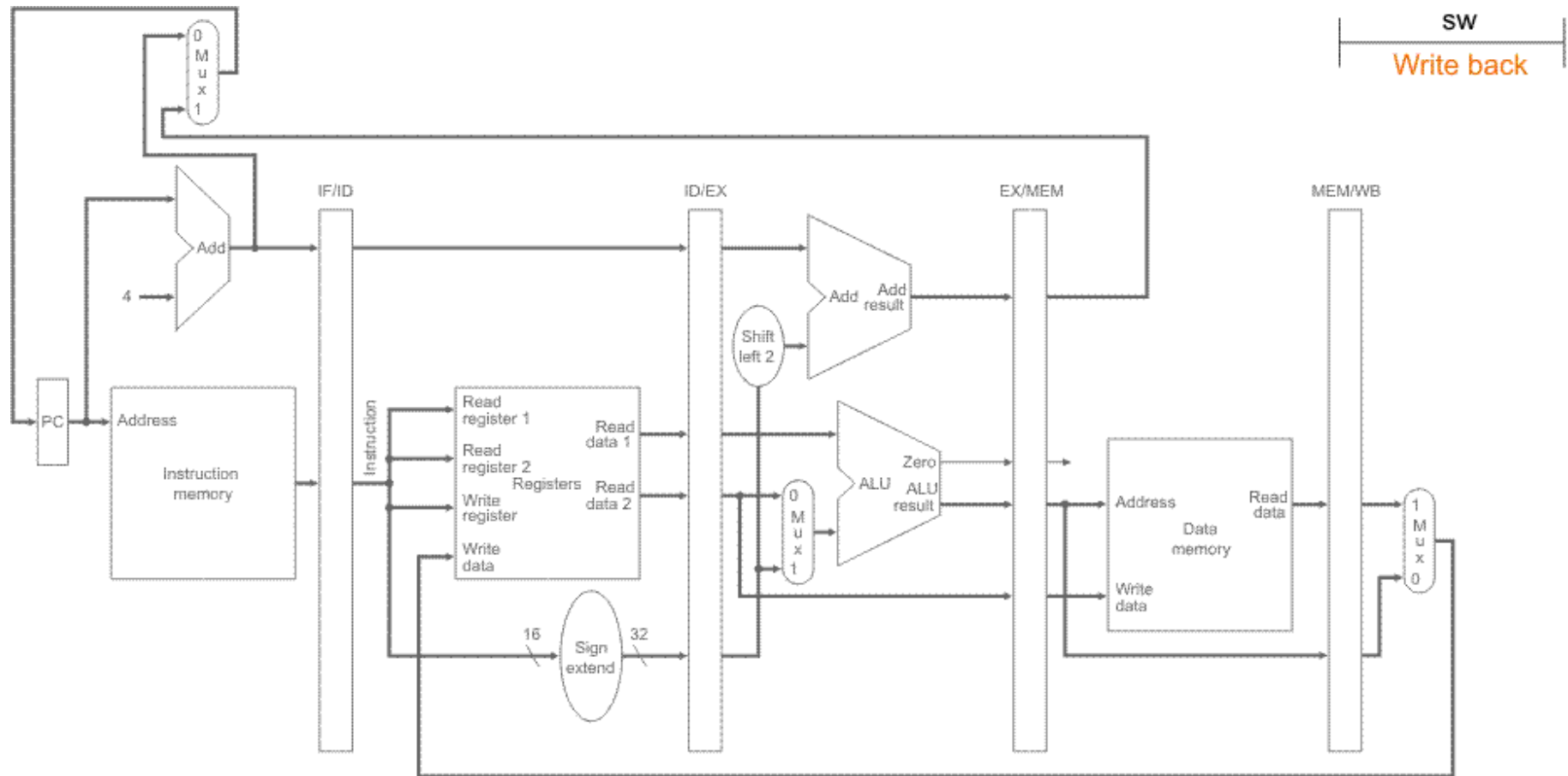
# SW (3)



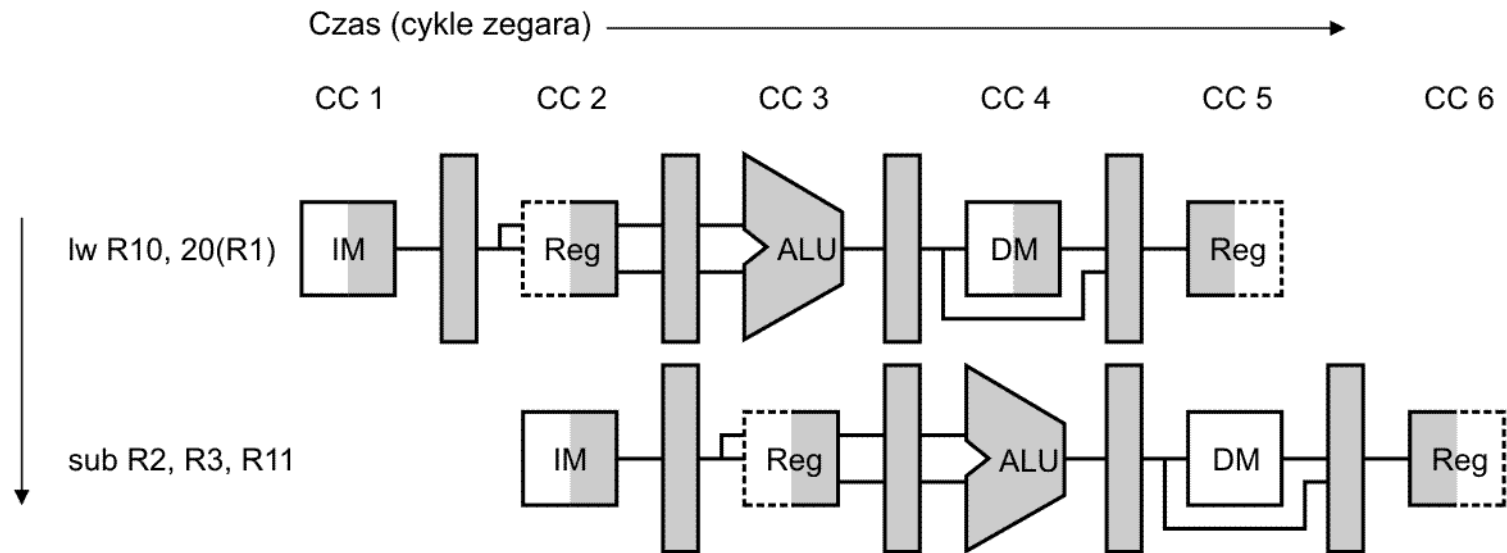
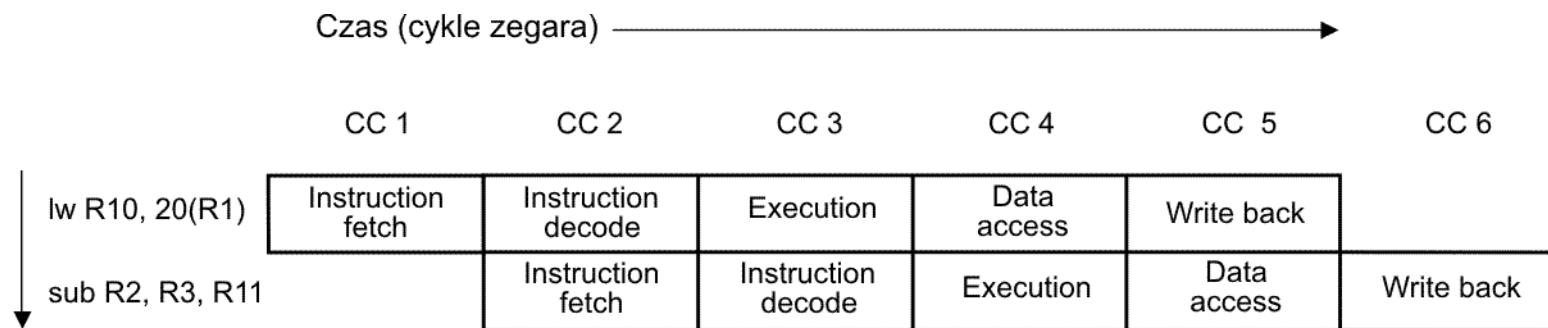
# SW (4)



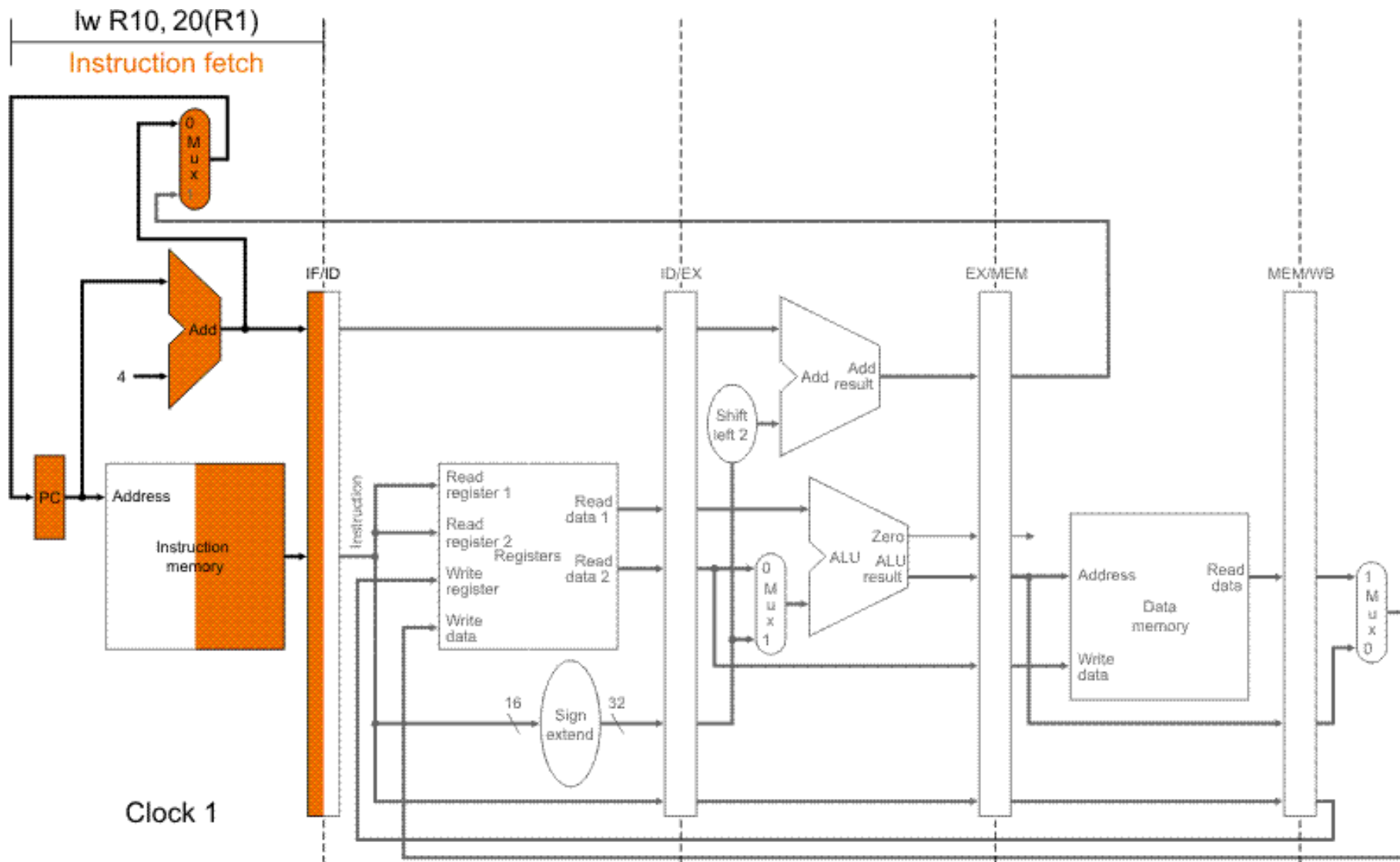
# SW (5) – „pusty cykl”



# Dwie instrukcje – demonstracja

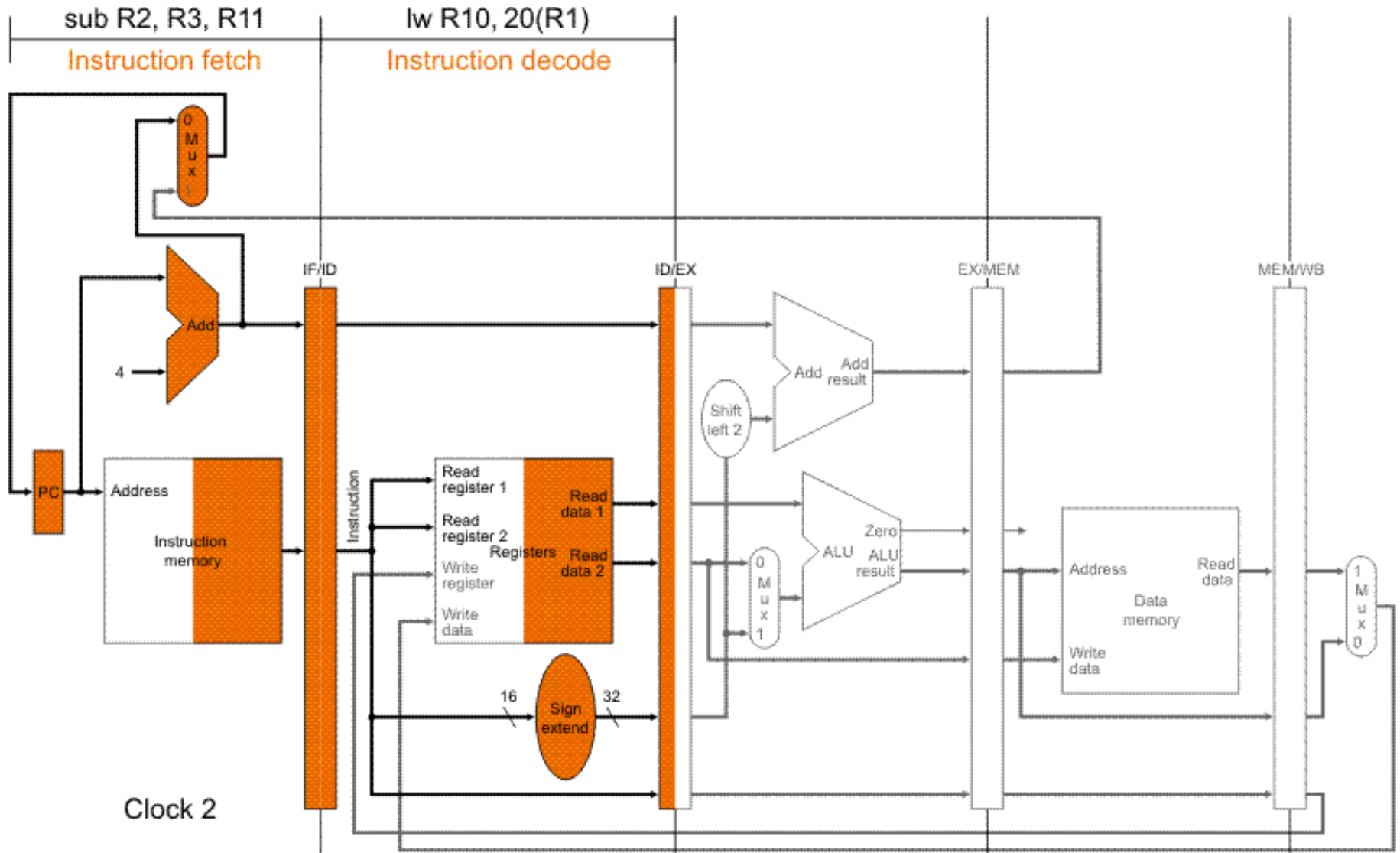


# LW / SUB (1)

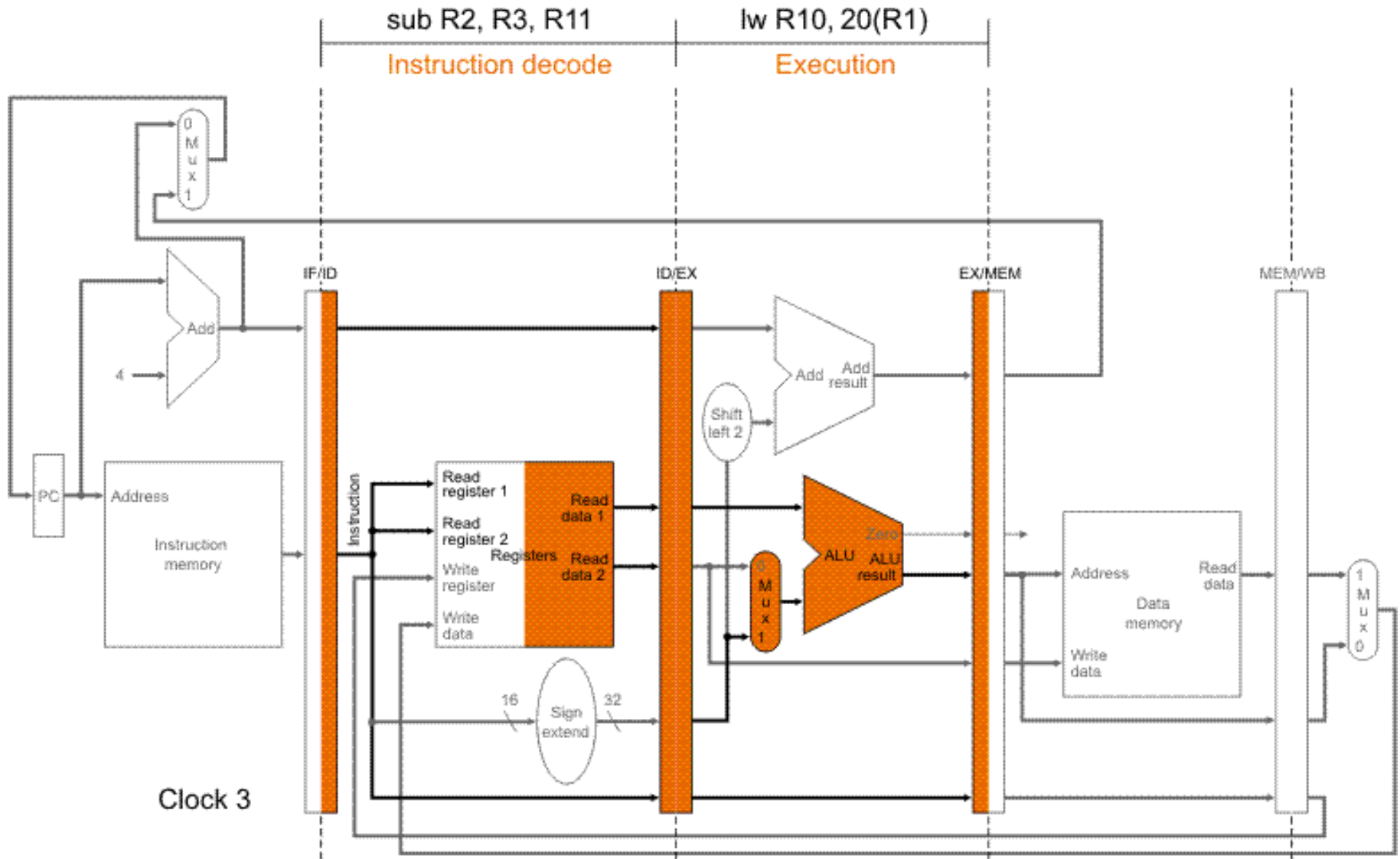


Clock 1

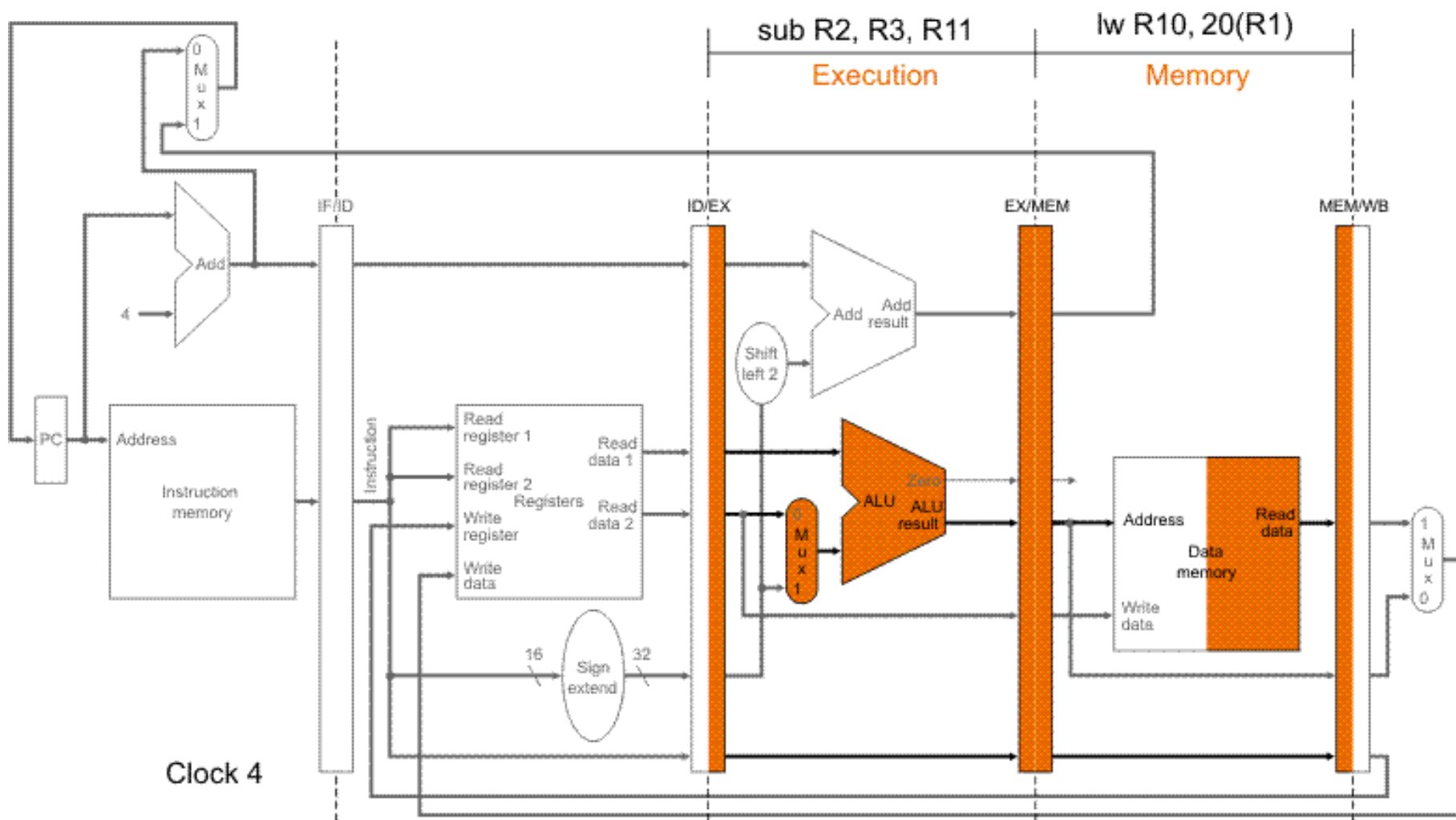
# LW / SUB (2)



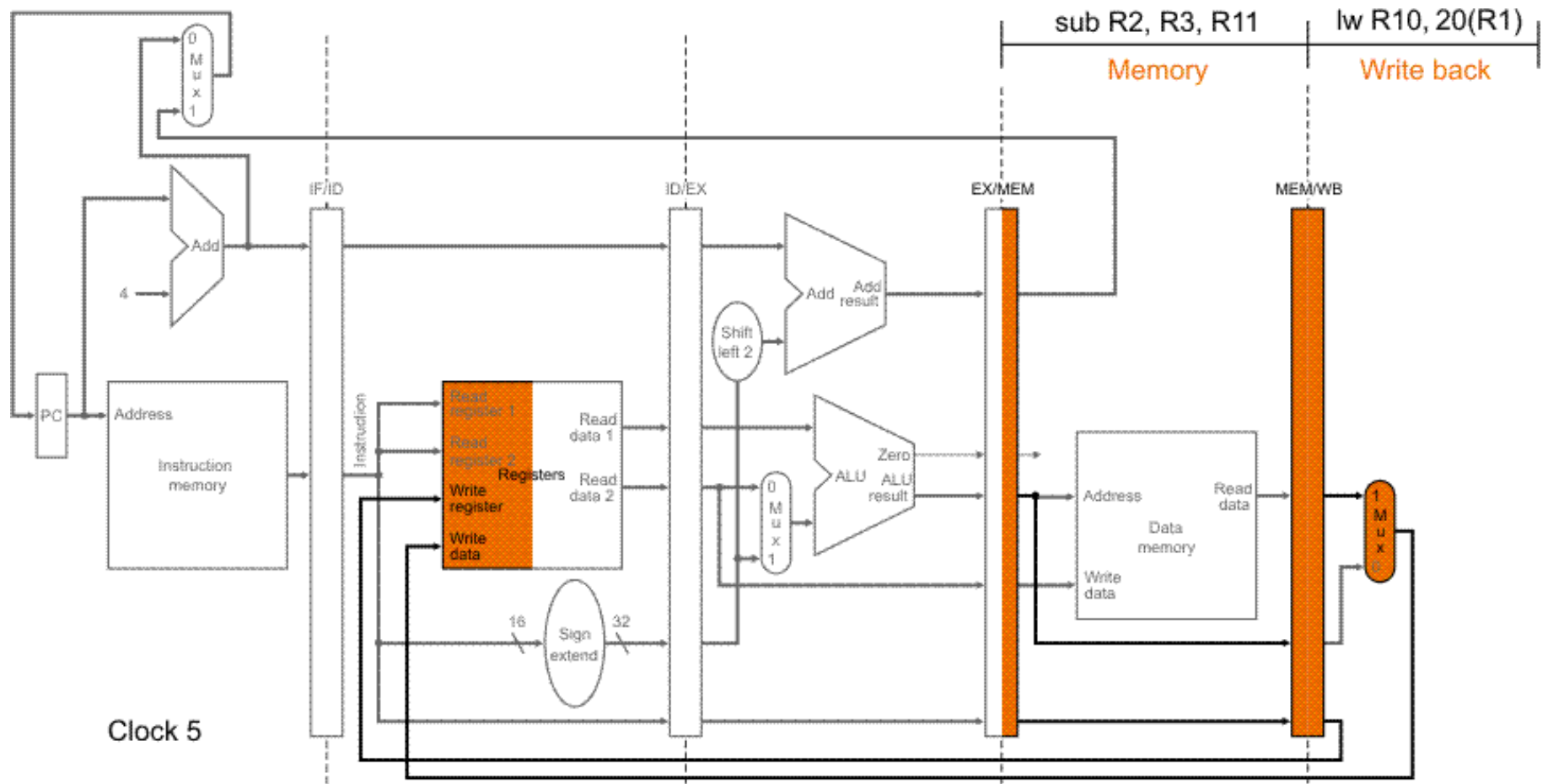
# LW / SUB (3)



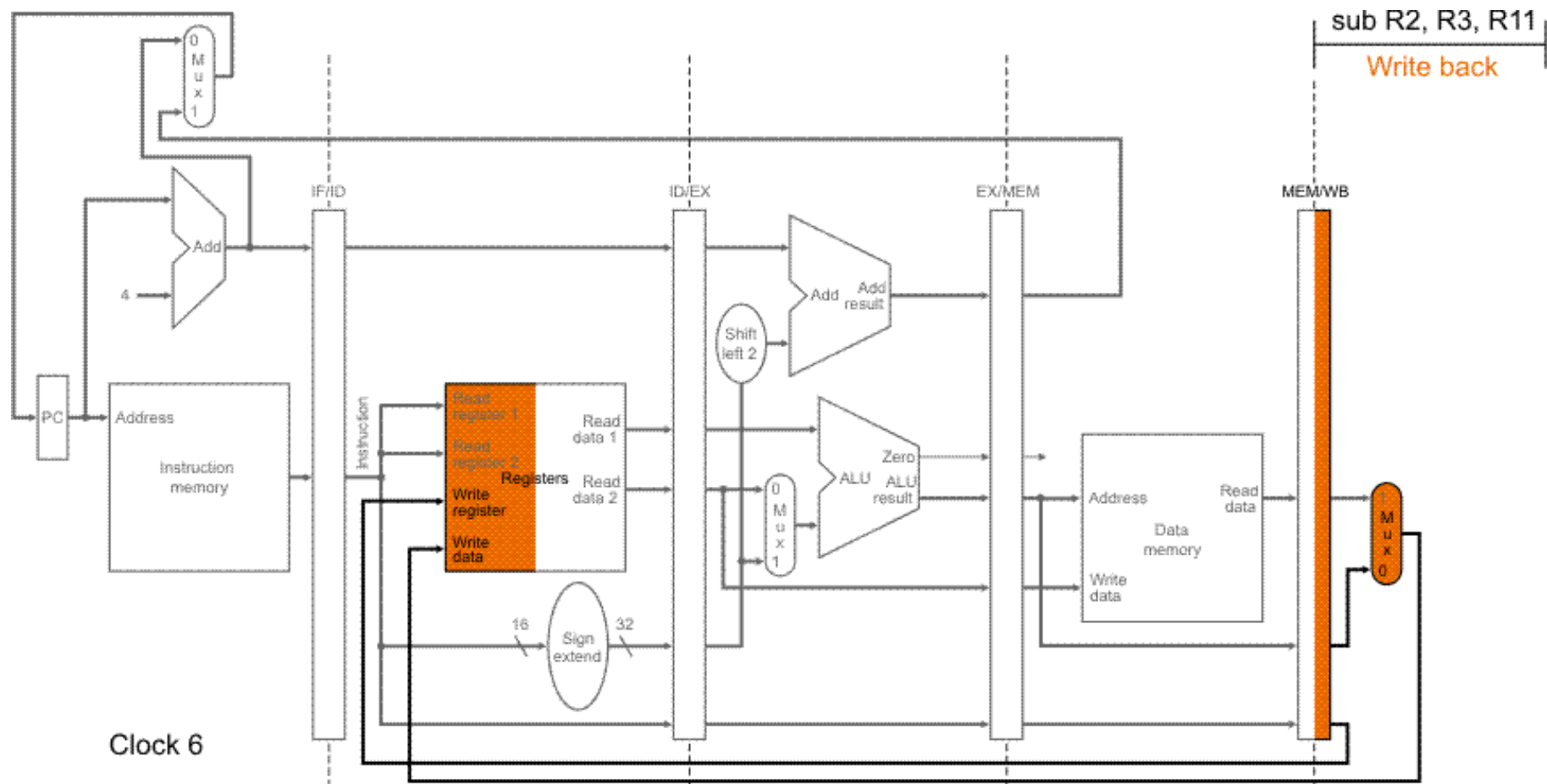
# LW / SUB (4)



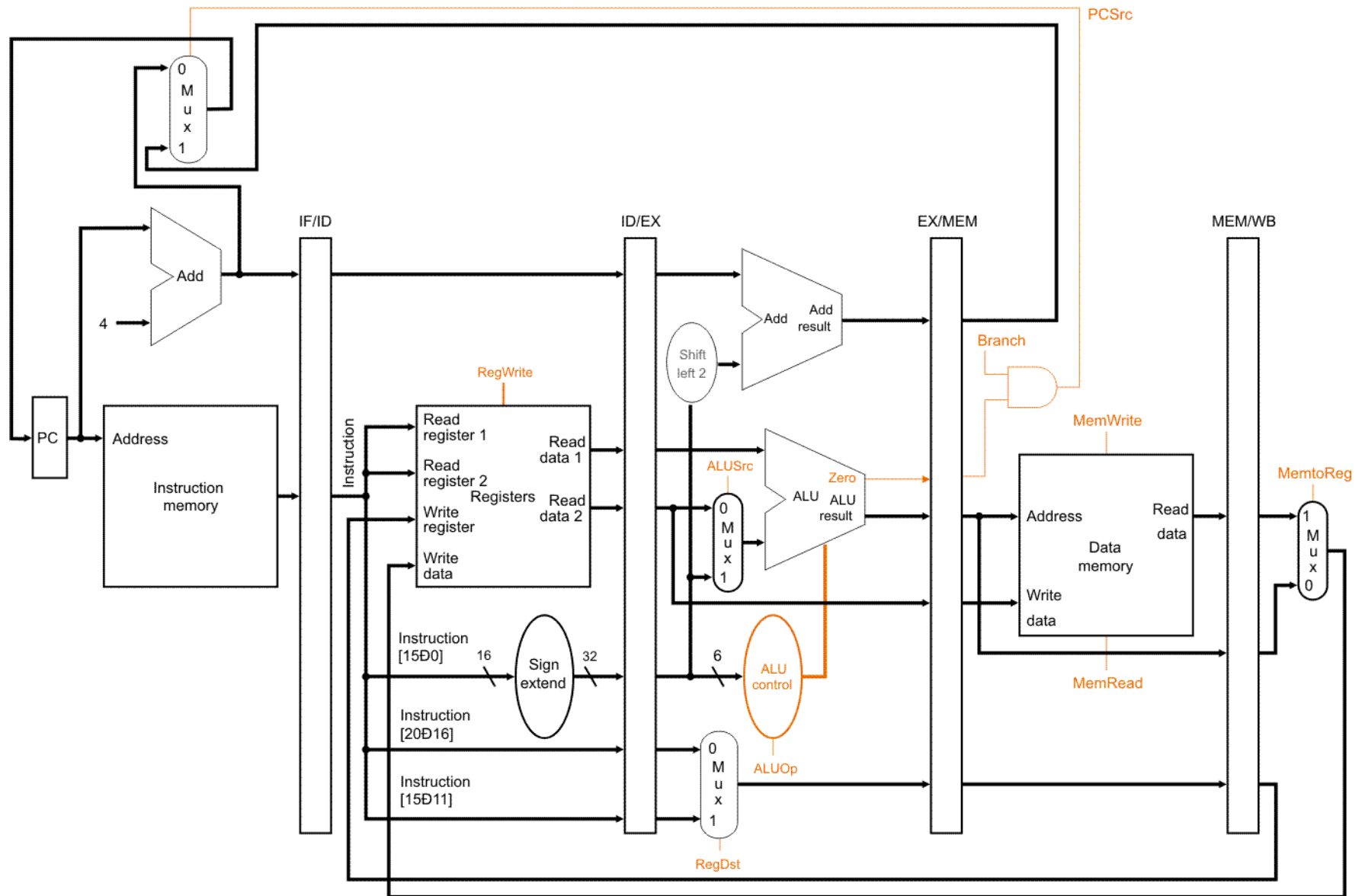
# LW / SUB (5)



# LW / SUB (6)



# Sygnaly sterujące w arch. potokowej

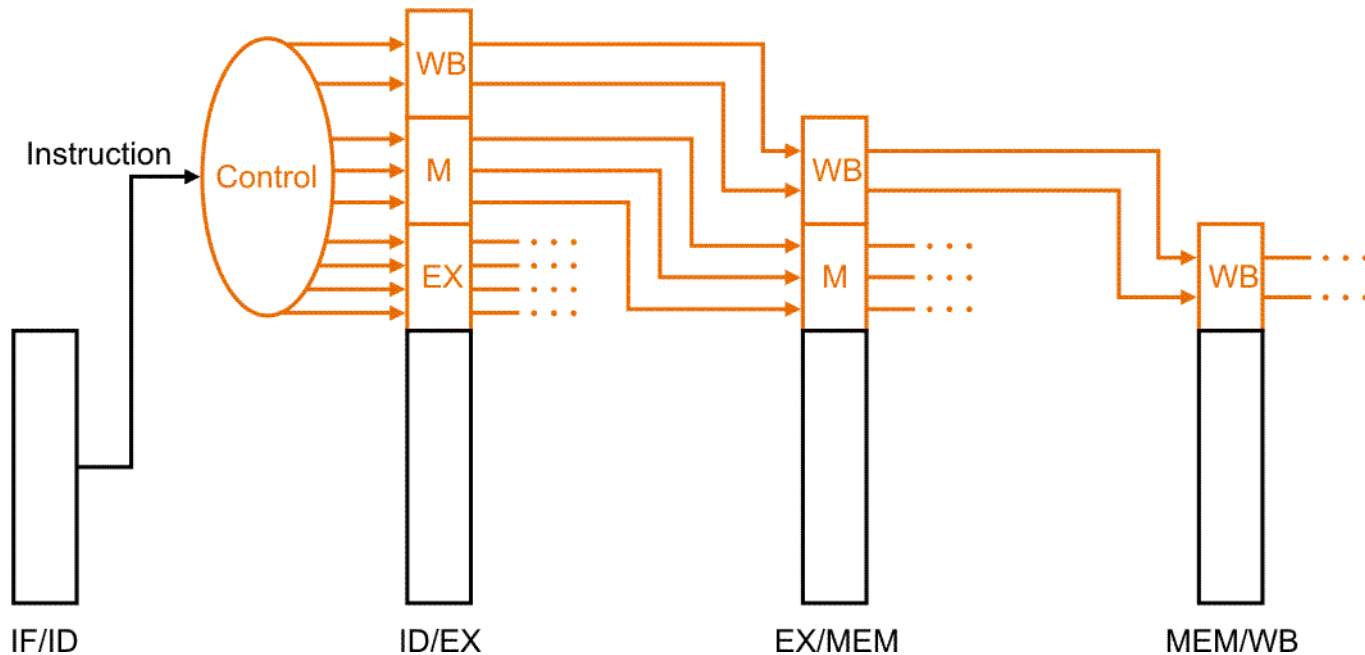


# Sterowanie w architekturze potokowej

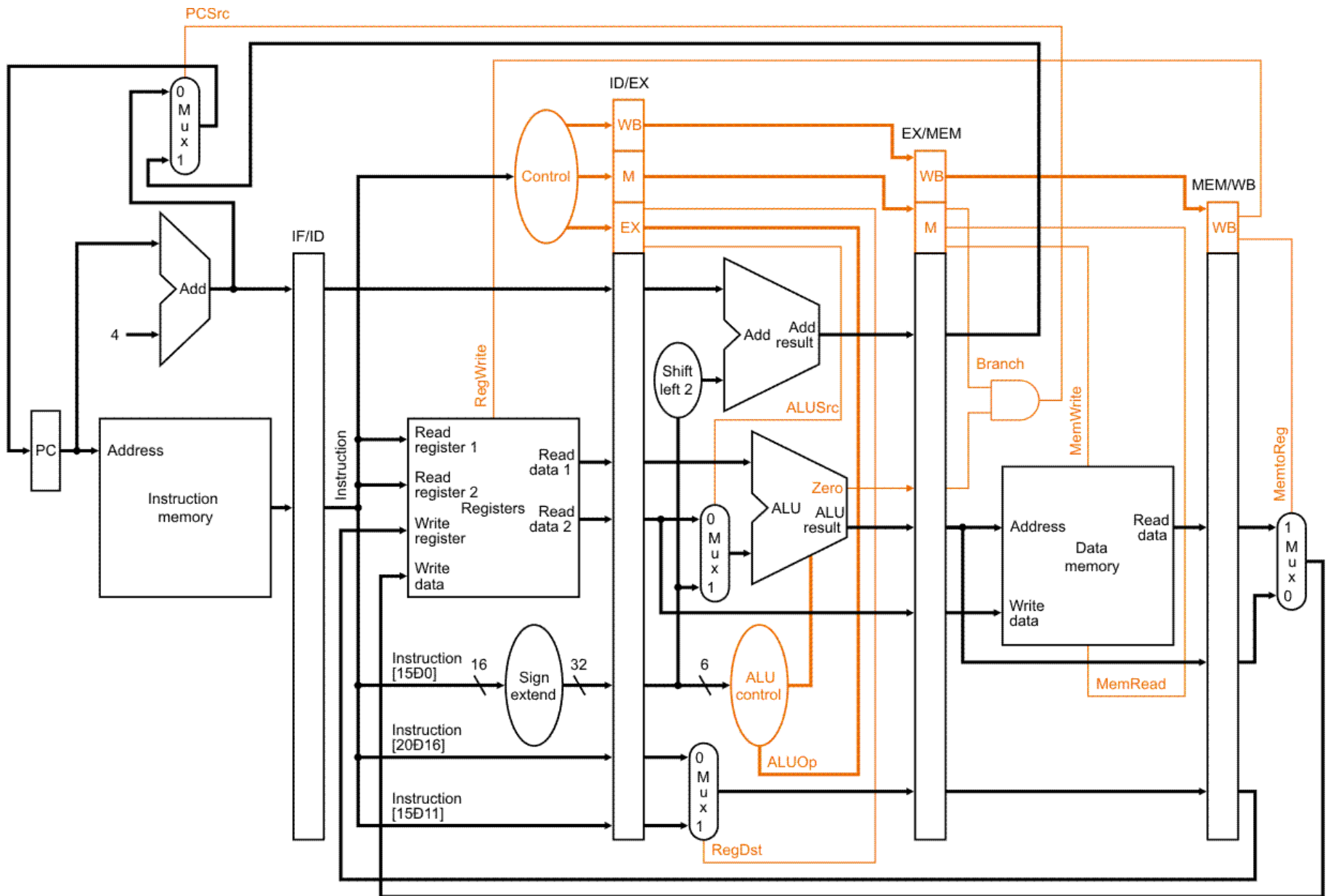
- Sterowanie jest proste, gdyż instrukcje mogą korzystać z zasobów tylko jeden raz (brak nawrotów)
- Sygnały sterujące można opisać tabelą prawdy, a jednostka sterująca jest układem kombinacyjnym
- Do sterowania wykonaniem instrukcji wystarczy jeden zestaw sygnałów (jak w *Single-Cycle*), ale rozłożony na kilka cyklic zegara (jak w *Multi-Cycle*)
- Instrukcja musi „nieść” ze sobą przez potok sygnały sterujące przeznaczone dla późniejszych faz wykonania
- Tylko proste instrukcji (jak w arch. *Single-Cycle*)

# Dystrybucja sygnałów sterujących

	EX			MEM			WB	
	RegDst	ALUOp	ALUSrc	Branch	Mem Read	Mem Write	Reg Write	MemTo Reg
R-type	<b>1</b>	<b>10</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
lw	<b>0</b>	<b>00</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>
sw	<b>X</b>	<b>00</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>X</b>
beq	<b>X</b>	<b>01</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>X</b>



# Najprostsza architektura potokowa



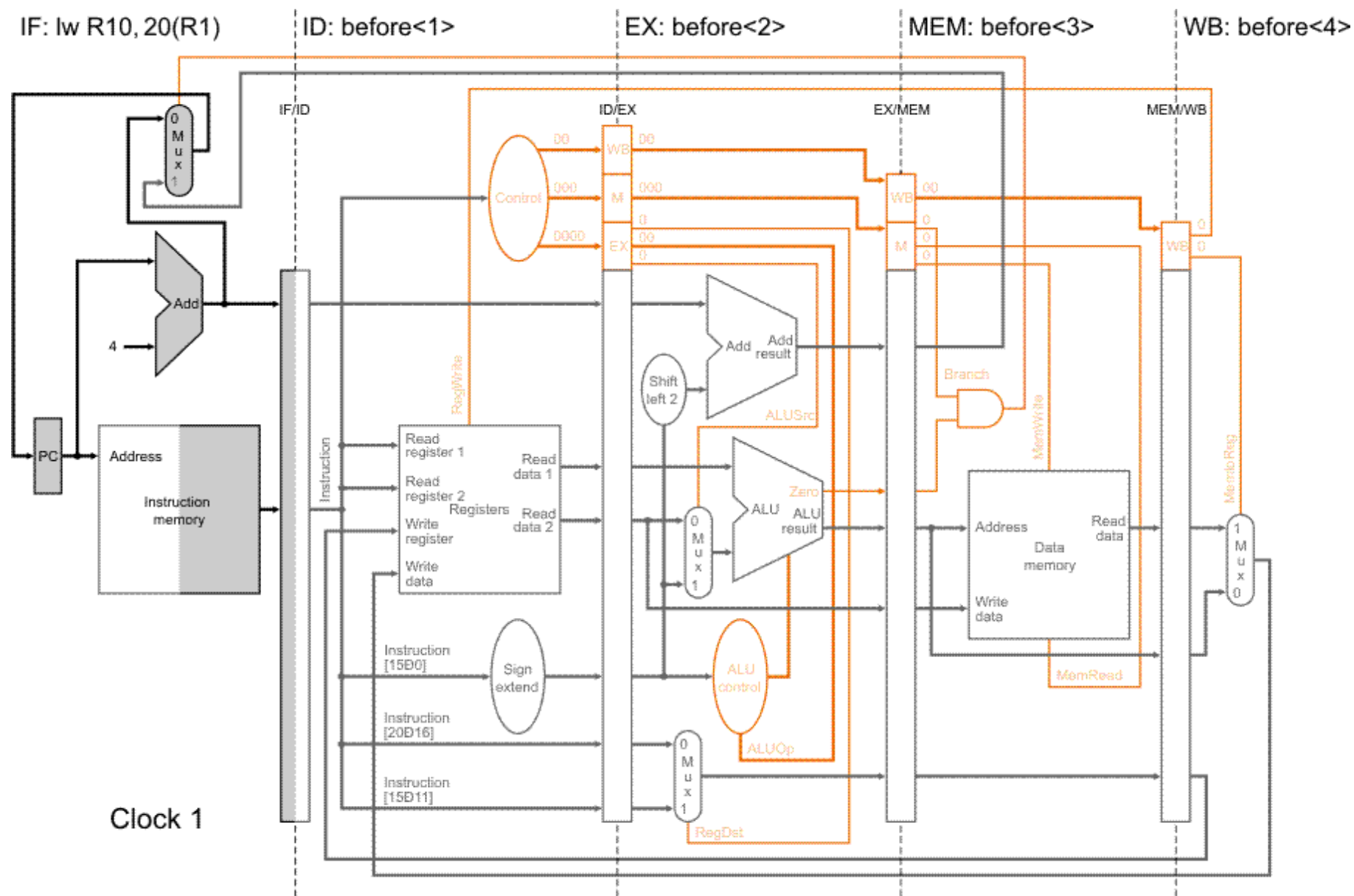
# Potok instrukcji – demonstracja

- Założenia
  - brak konfliktów zasobów, danych i sterowania

```
lw      R10 , 20 (R1)
sub     R2 , R3 , R11
and     R4 , R5 , R12
or      R6 , R7 , R13
add     R8 , R9 , R14
```



# Potok (1)

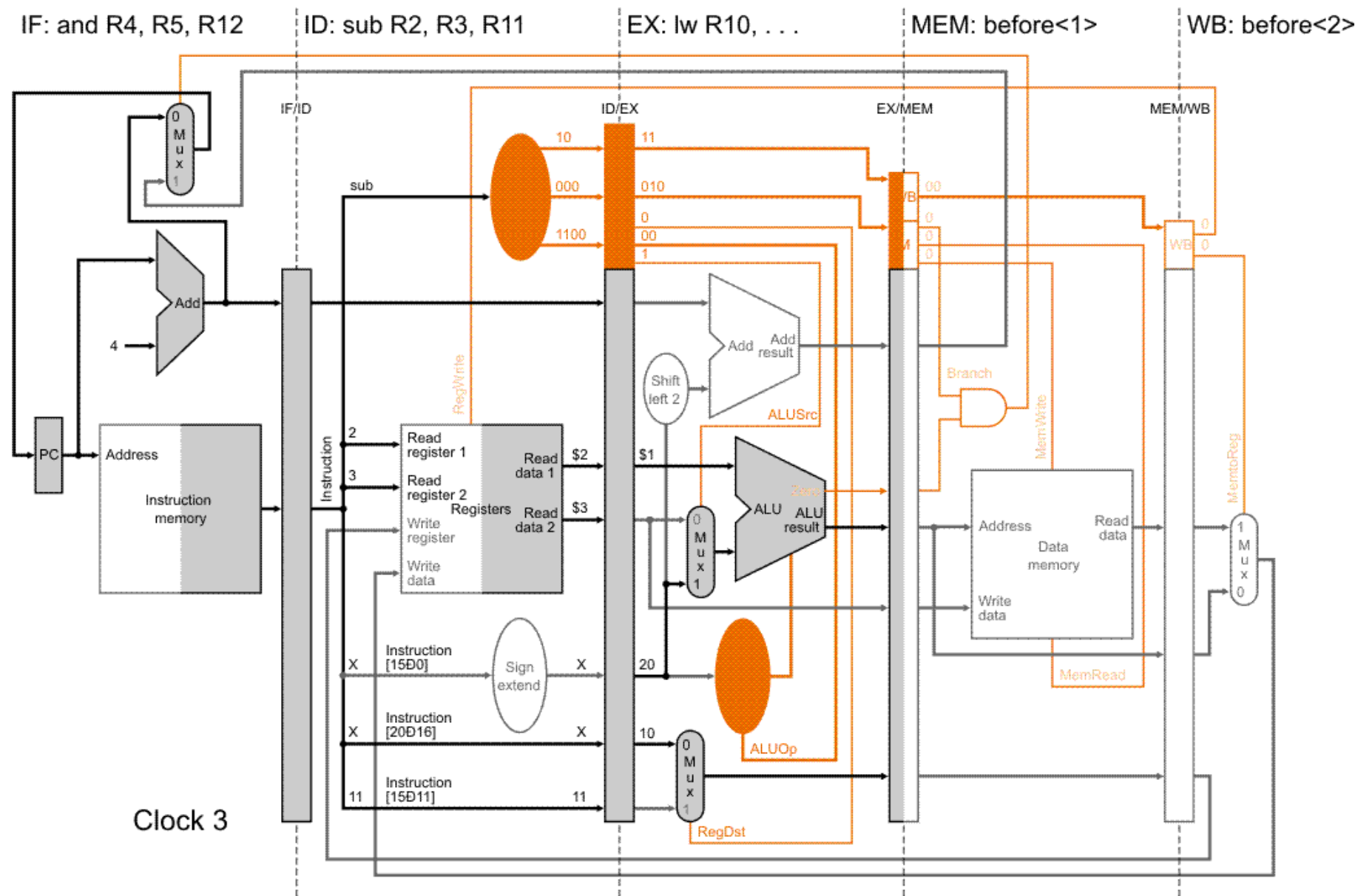


Clock 1





# Potok (3)

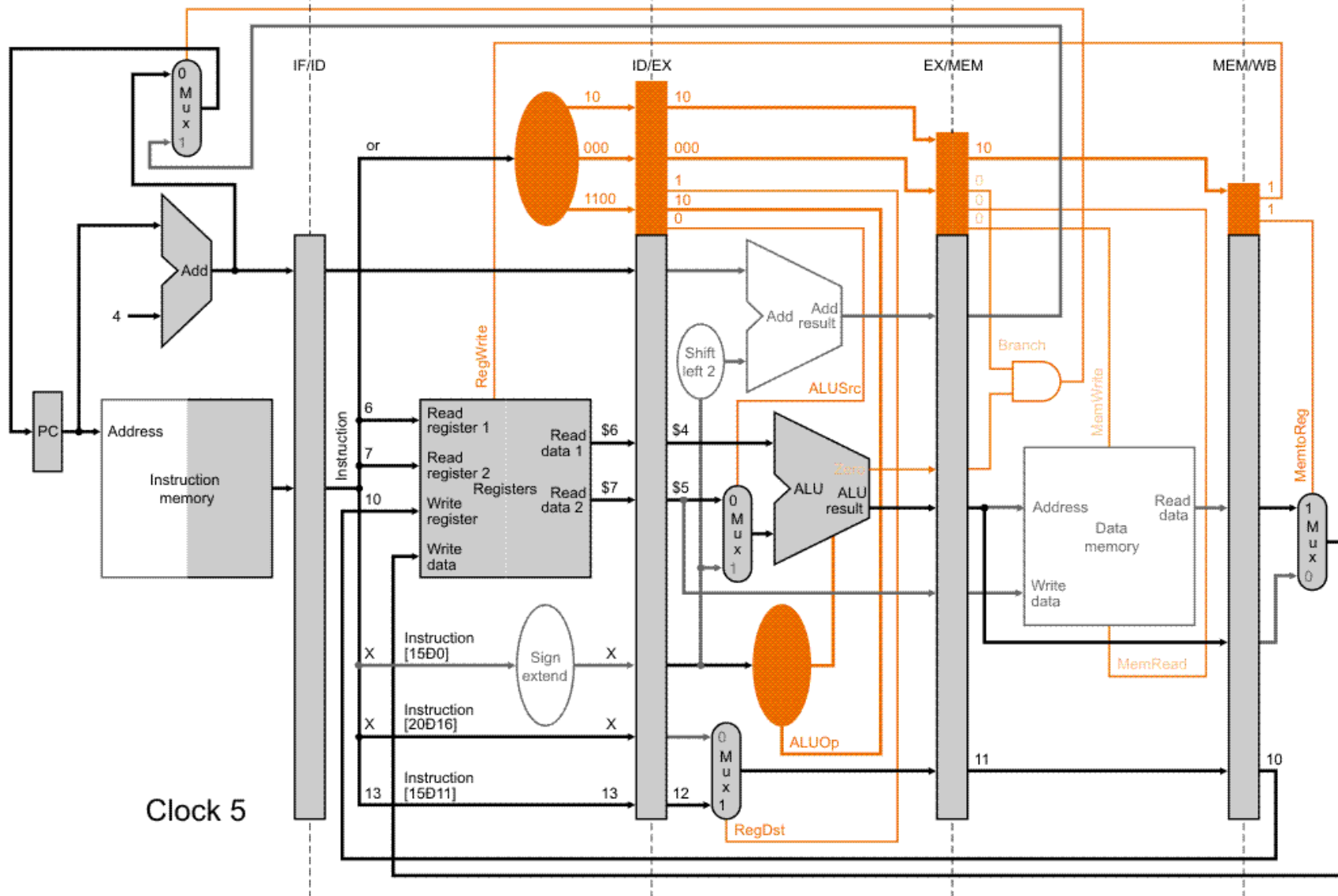






# Potok (5)

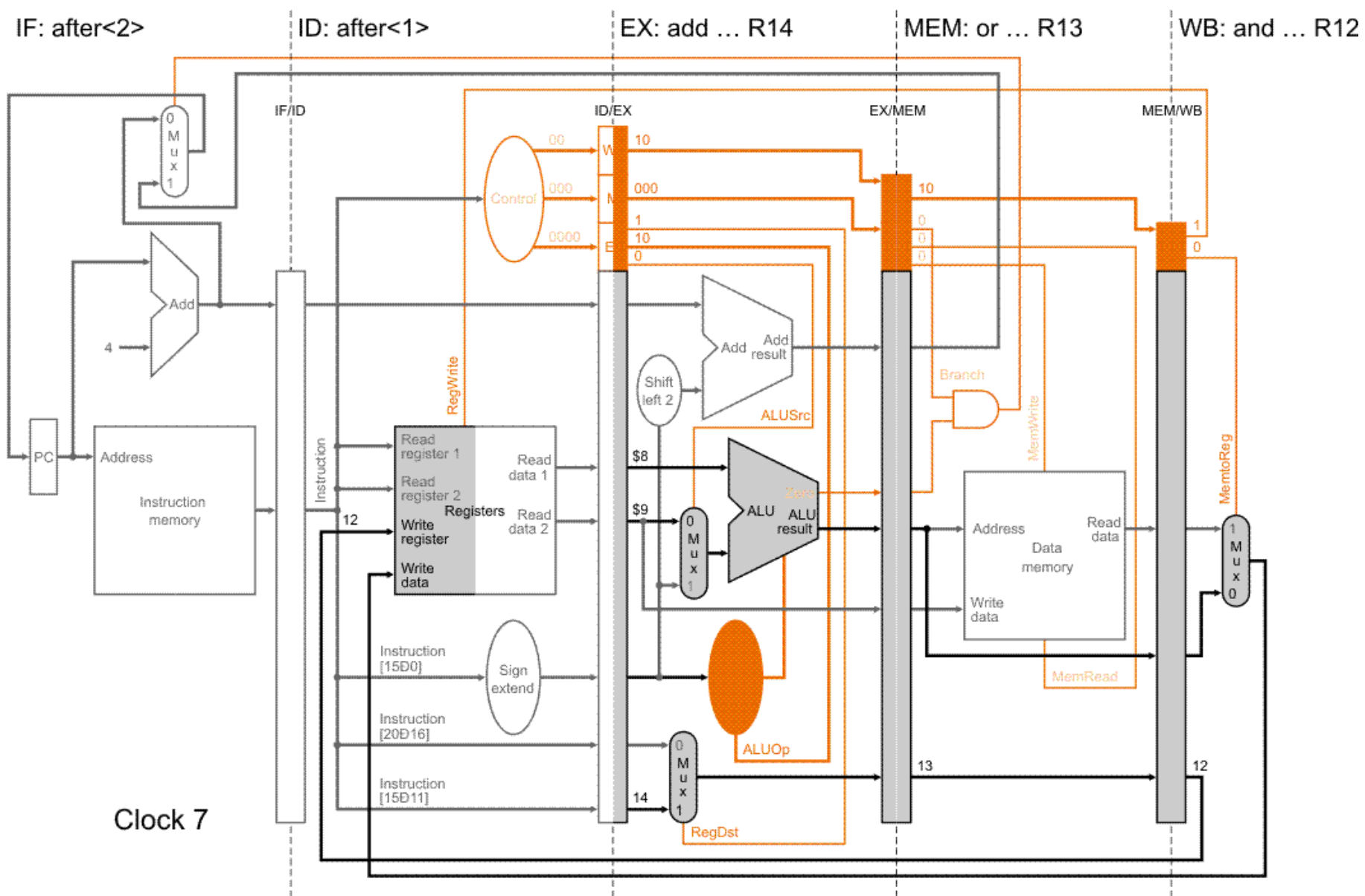
IF: add R8, R9, R14      ID: or R6, R7, R13      EX: and ... R12      MEM: sub ... R11      WB: lw R10, ...



Clock 5



# Potok (7)





# Potok (9)

