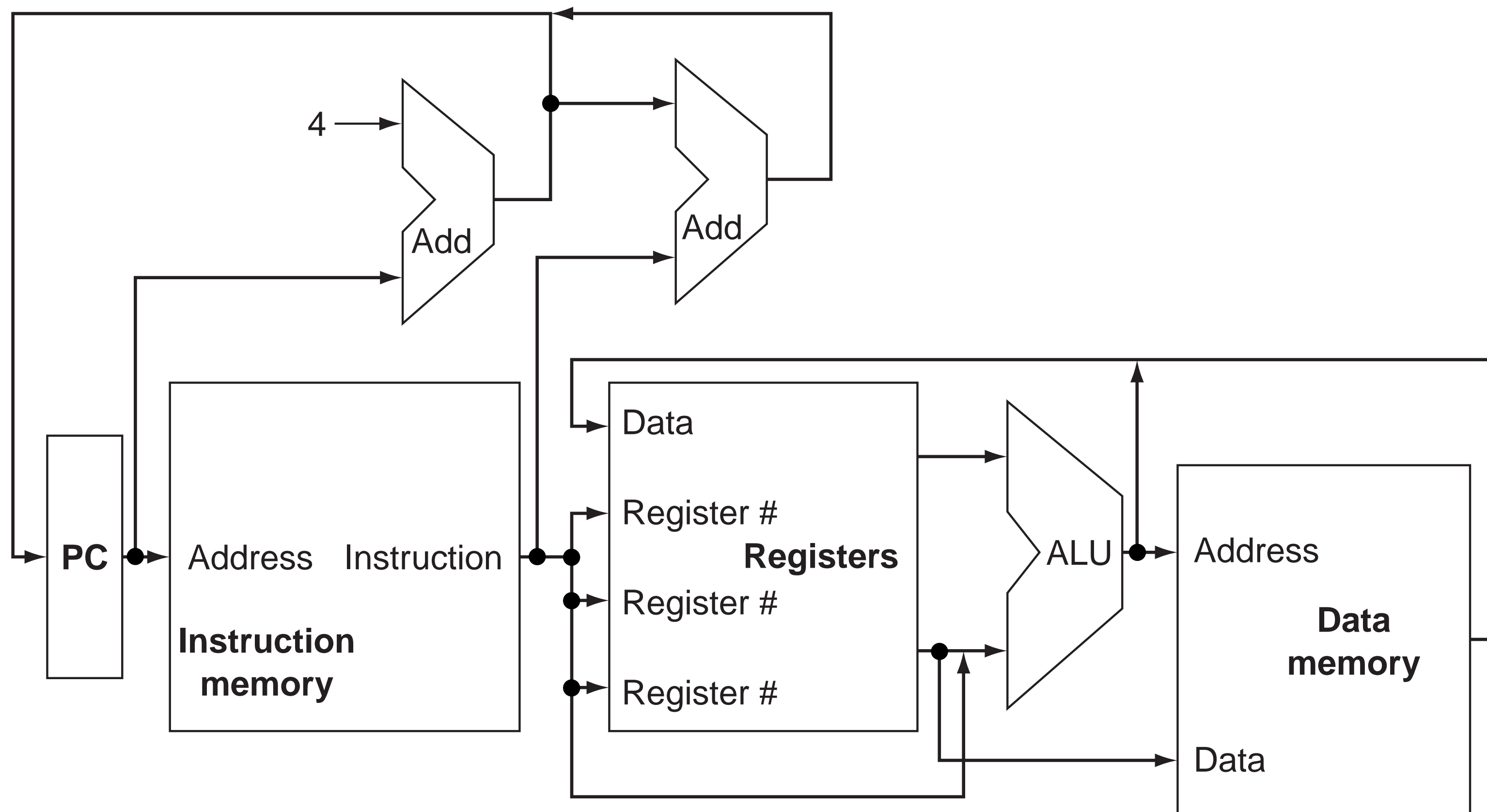
Przetwarzanie potokowe

(pipelining)

Przypomnienie - implementacja jednocyklowa



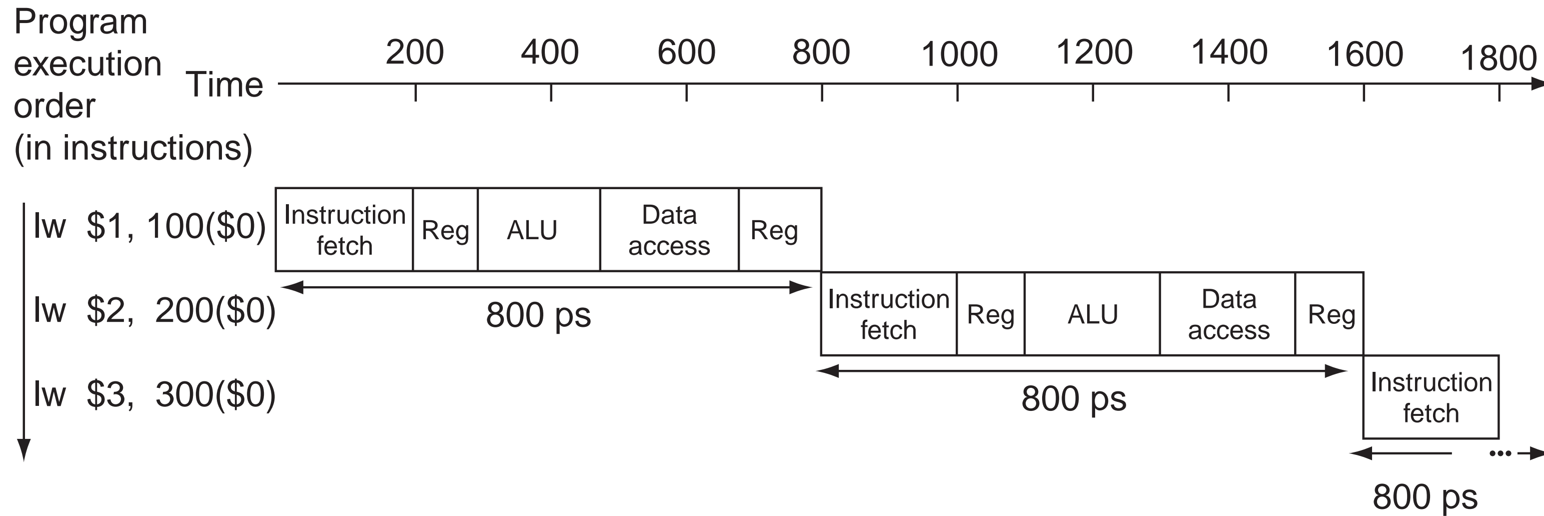
Wstęp

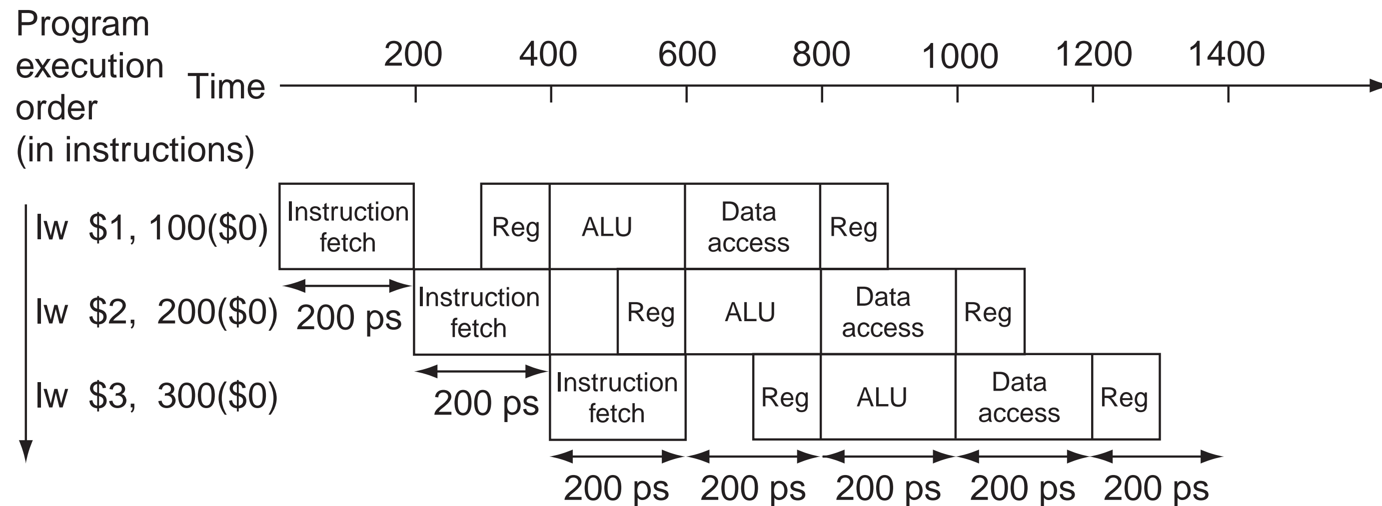
- W implementacjach prezentowanych tydzień temu w jednym momencie wykonuje się tylko jednen rozkaz.
- Jego wykonanie przebiega w kilku fazach.
- W każdej fazie używana jest tylko część układów logicznych.
- Pomysł: w momencie gdy pierwszy rozkaz rozpoczyna drugą fazę, drugi zaczyna pierwszą; gdy pierwszy zaczyna trzecią, drugi zaczyna drugą, a trzeci pierwszą, itd.
- W efekcie w jednym momencie wykonuje się tyle rozkazów ile jest faz. Każdy wykonywany rozkaz znajduje się w innej fazie.
- Taką technikę nazywamy *przetwarzaniem potokowym* (ang. *pipelining*).
- W przetwarzaniu potokowym nie przyspieszamy wykonania pojedynczego rozkazu (wręcz przeciwnie!).
- Zysk dzięki urównolegleniu wykonywanie operacji.

MIPS - fazy wykonania rozkazu

- Wykonanie rozkazu podzielimy na 5 faz:
  - Pobranie rozkazu z pamięci (IF, instruction fetch) - 200ps
  - Dekodowanie rozkazu i odczyt rejestrów (ID, instruction decode) - 100 ps
  - Wykonanie operacji typu R lub wyliczenie adresu (EX, execute) - 200 ps
  - Operacja na pamięci (MEM, memory) - 200 ps
  - Zapis do rejestru (WB, write back) - 100 ps

Porównanie





Projektowanie listy rozkazów

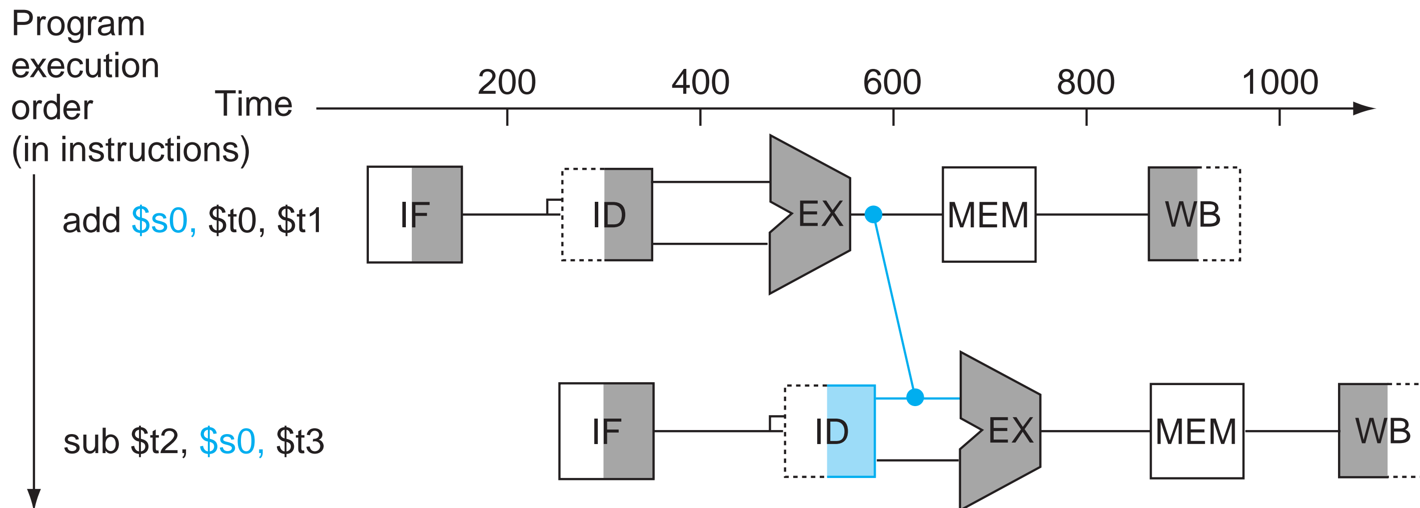
- Lista rozkazów MIPS została zaprojektowana z myślą o przetwarzaniu potokowym.
- Stała długość rozkazów ułatwia ich pobieranie.
- Mała liczba formatów rozkazów. Rejestry źródłowe w tych samych miejscach - rejestry można odczytywać równolegle z dekodowaniem rozkazu.
- Operacje na pamięci wykonują tylko rozkazy `lw`, `sw` (i ich warianty).

Hazardy

- Hazardy strukturalne
  - Dwa rozkazy chcą w tej samej chwili skorzystać z tego samego zasobu (np. ALU, pamięć, rejestry)
  - W zasadzie problem nie występuje w przypadku MIPS.
  - Pojawiłby się, gdybyśmy nie rozdzieli pamięci.
- Hazardy danych.
  - Rozkaz potrzebuje rejestru, który zapisuje poprzedni rozkaz.
  - Np. `add $s0, $t0, $t1` `add $t2, $s0, $t3`
- Hazardy sterowania.
  - W przypadku rozkazu skoku nie wiadomo, który rozkaz należy pobrać jako następny.
  - Szczególnie trudne w przypadku skoków warunkowych.

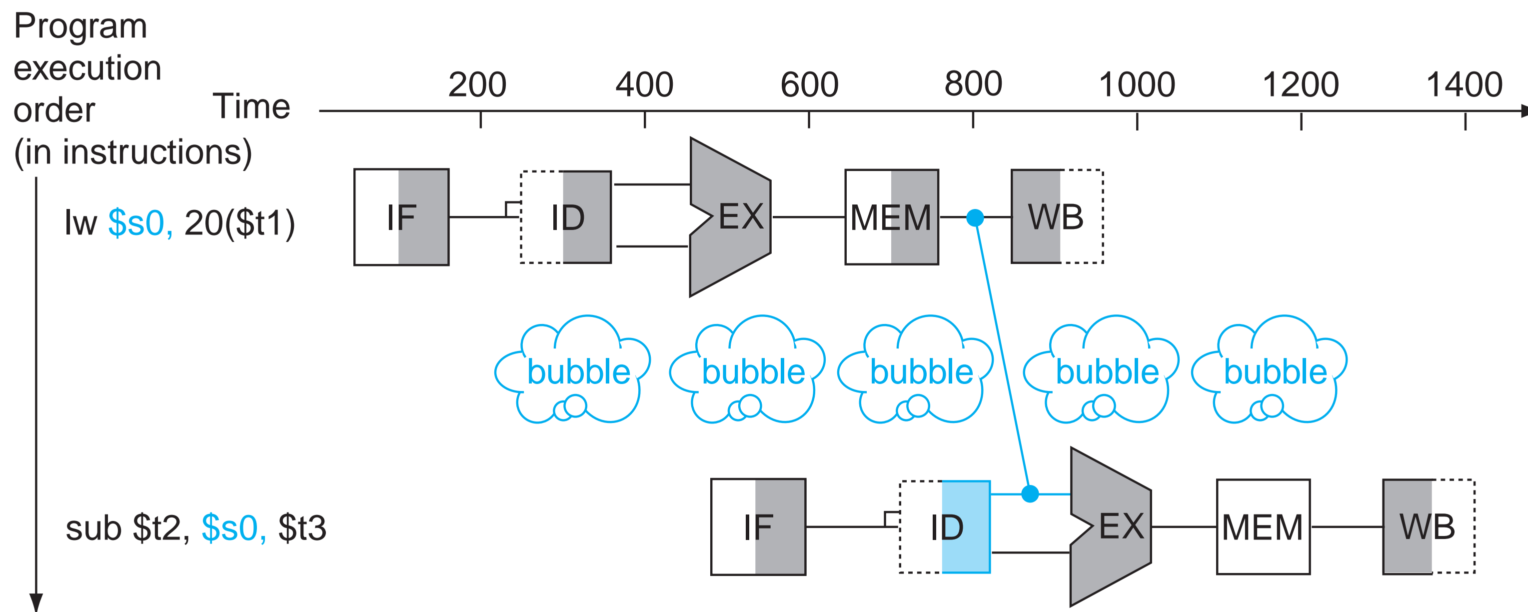
Hazardy danych

- Np. `add $s0, $t0, $t1`   `add $t2, $s0, $t3`
- `$s0` jest zapisywany dopiero w piątej fazie pierwszego rozkazu
- Drugi rozkaz potrzebuje go już dwa cykle wcześniej!
- Rozwiązanie kiepskie: opóźnić (jakoś) wykonanie drugiego rozkazu o dwa cykle (dodać dwa „bąbelki”).
- Rozwiązanie programowe: reorganizacja kodu (np. przez kompilator) - trudne.
- Rozwiązanie sprzętowe: *forwarding* (*bypassing*)



Hazardy danych - cd.

- Forwarding nie eliminuje wszystkich hazardów danych.



Hazardy danych - reorganizacja kodu

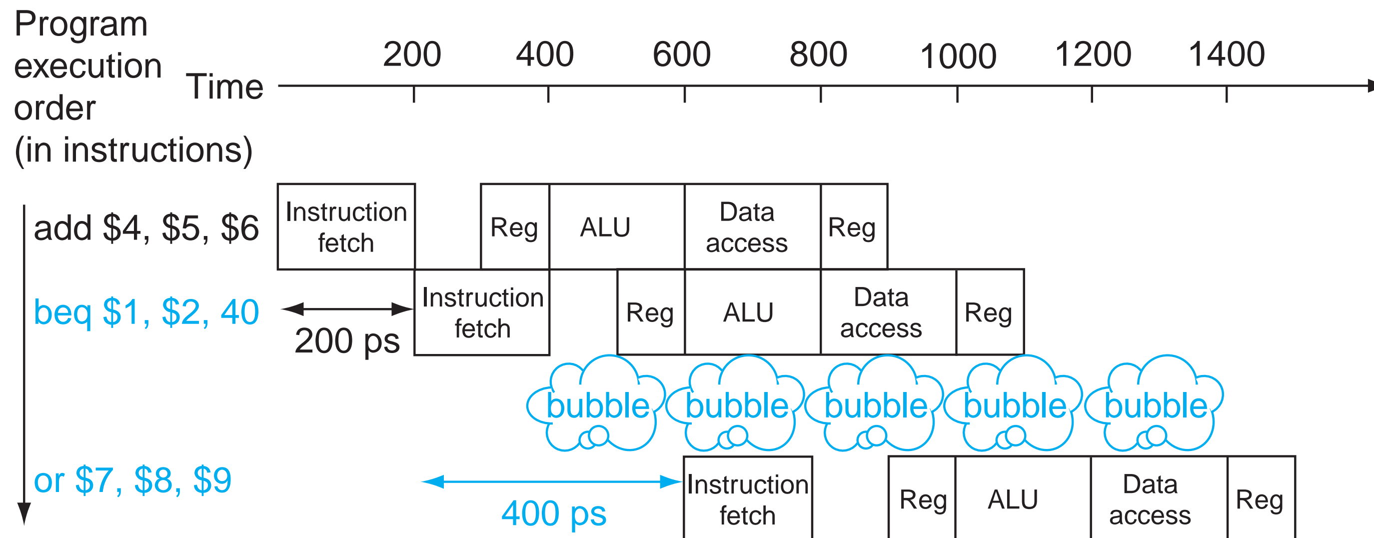
```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

- Z lewej strony 2 hazardy (rozkazy `add`) – pozostałe eliminowane przez forwarding.
- Z prawej – brak hazardów.

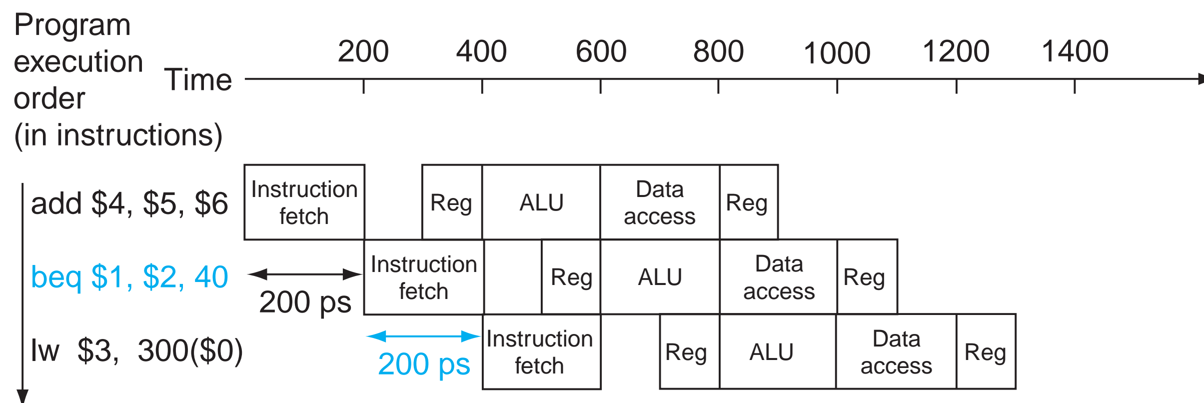
Hazardy sterowania

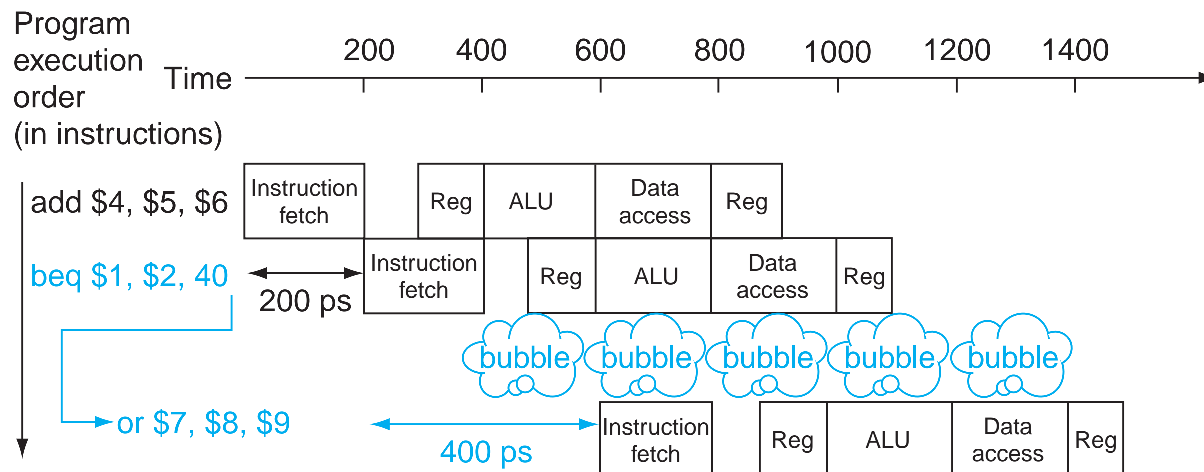
- Załóżmy, że potrafimy w drugiej fazie wykonywania `beq` porównać rejestry, wyliczyć adres i ustawić nowy PC
- Nawet wtedy musimy opóźnić potok:



Hazardy sterowania - przewidywanie skoków

- Jedno z rozwiązań: przewidywanie skoków
- Np. zakładamy, że skoki się nie wykonują



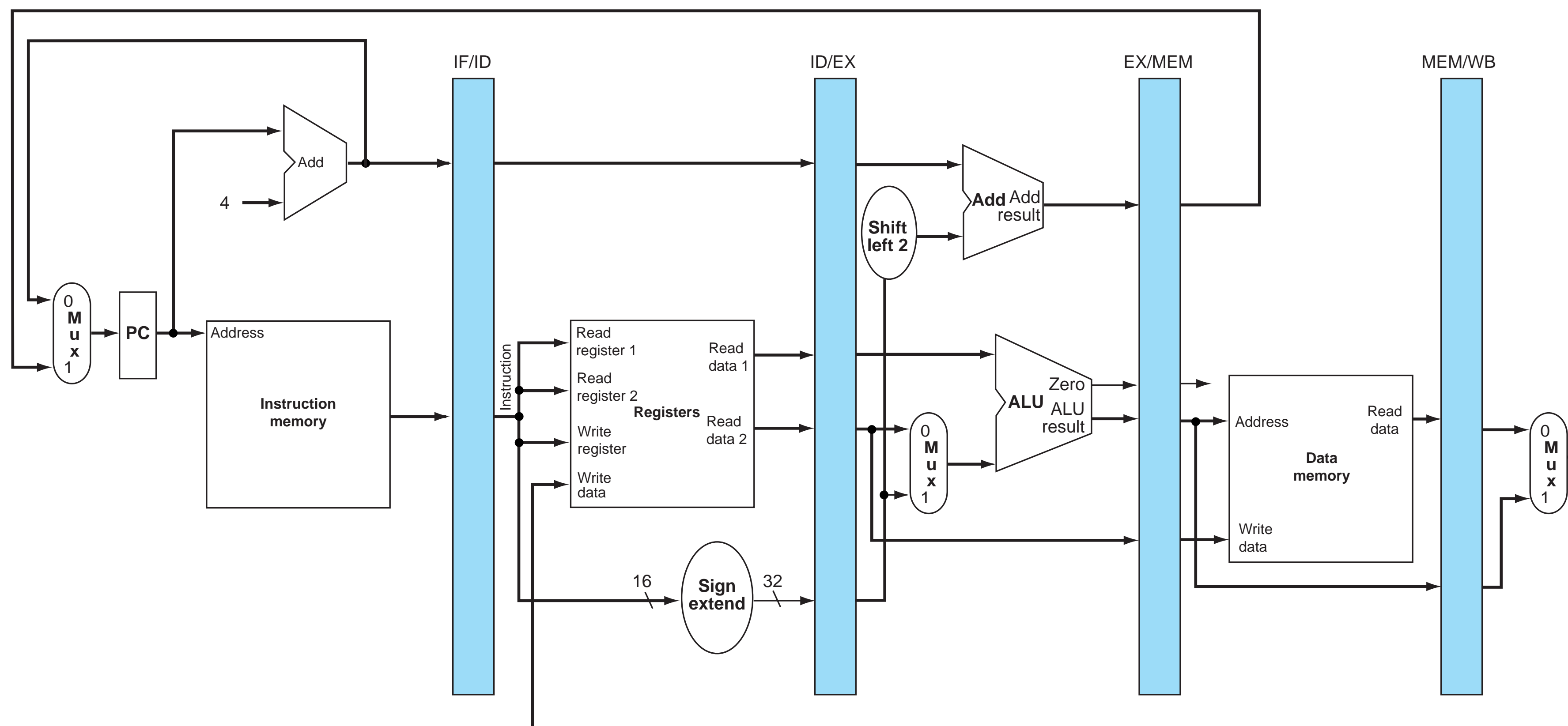


Hazardy sterowania - przewidywanie skoków

- Można zakładać, że niektóre skoki się wykonują (np. te skaczące wstecz), a inne nie.
- Przewidywanie może być też dynamiczne – jeśli podczas poprzedniego przejścia przez rozkaz skok się wykonał, to teraz też zakładamy, że się wykona. Można prowadzić historię kilku ostatnich wykonań. Potrzebne oczywiście dodatkowe układy logiczne.
- Dobrze zorganizowane przewidywanie dynamiczne daje ponad 90% skuteczność!
- Inne rozwiązanie: skoki opóźnione (stosowane naprawdę w MIPS, ale niewidoczne dla programisty asemblerowego).

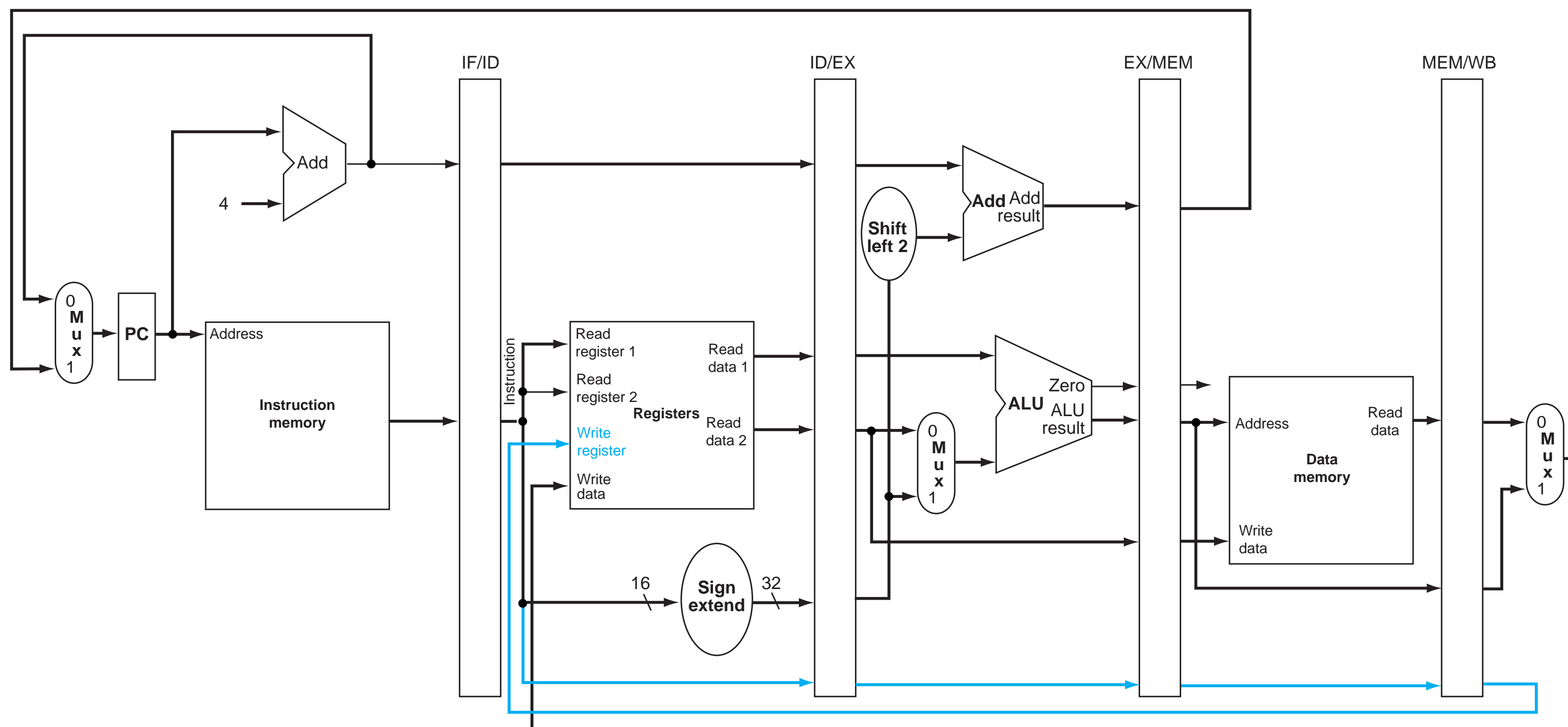
Model procesora potokowego

- Rozważamy listę rozkazów taką jak tydzień temu, ale bez `j`.
- Nowość - rejestry rozdzielające poszczególne fazy.
- Na początek nie przejmujemy się hazardami.

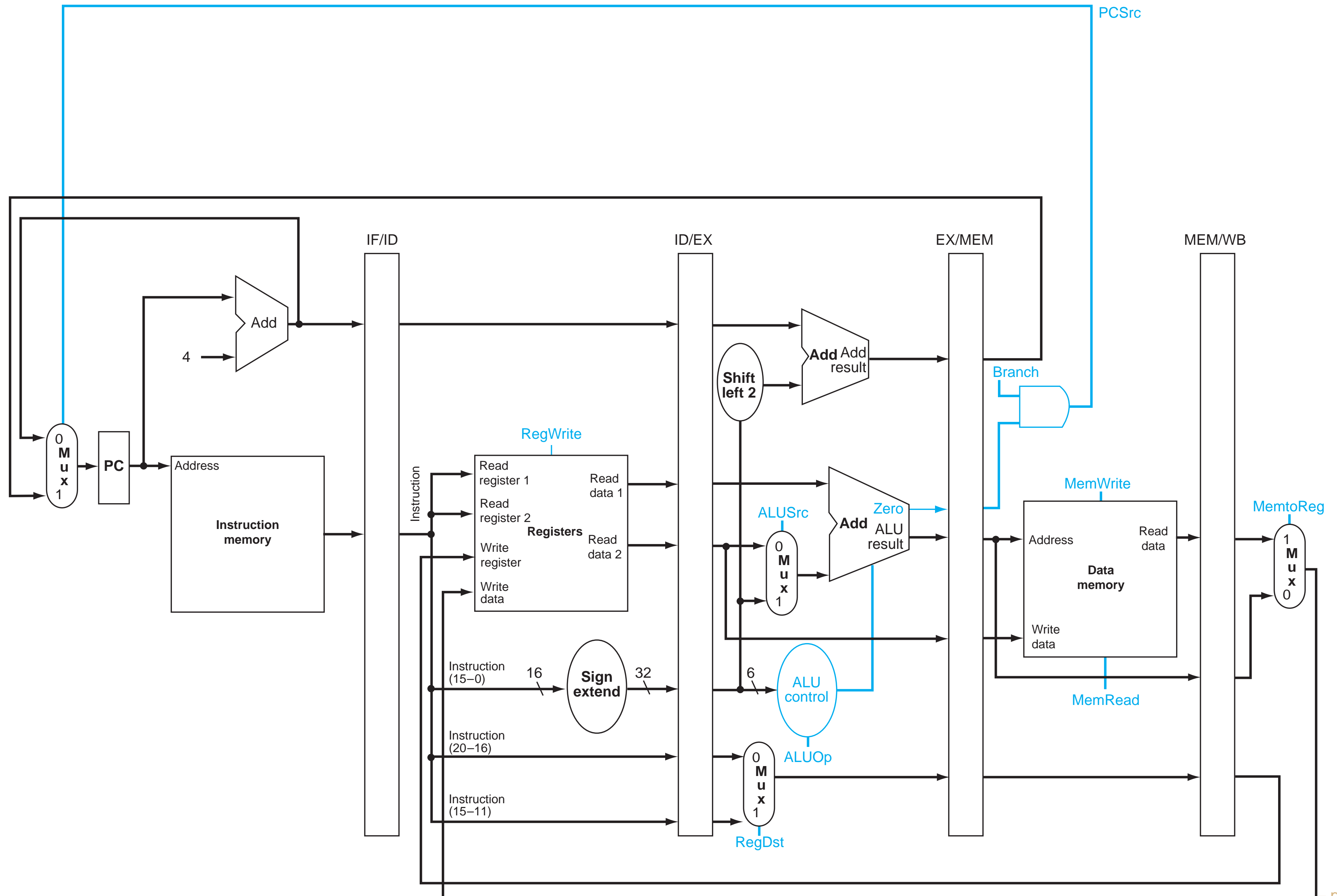


Model procesora potokowego – poprawka

- Skąd bierzemy numer rejestru do zapisania (dla `lw`)?

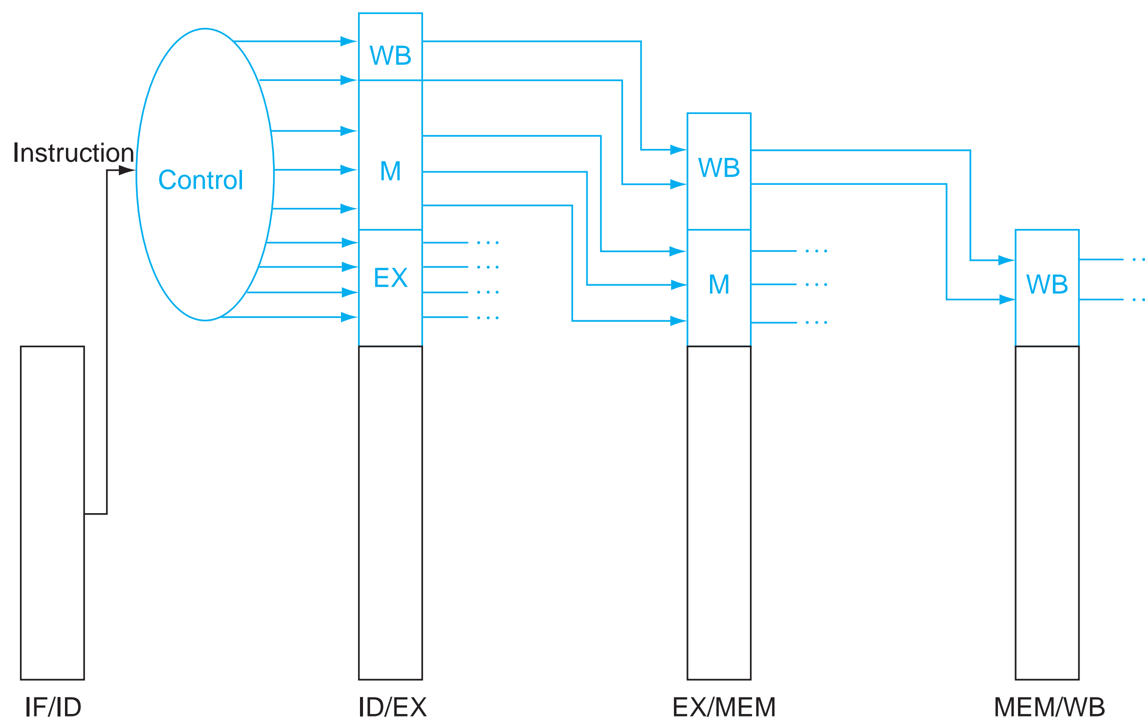


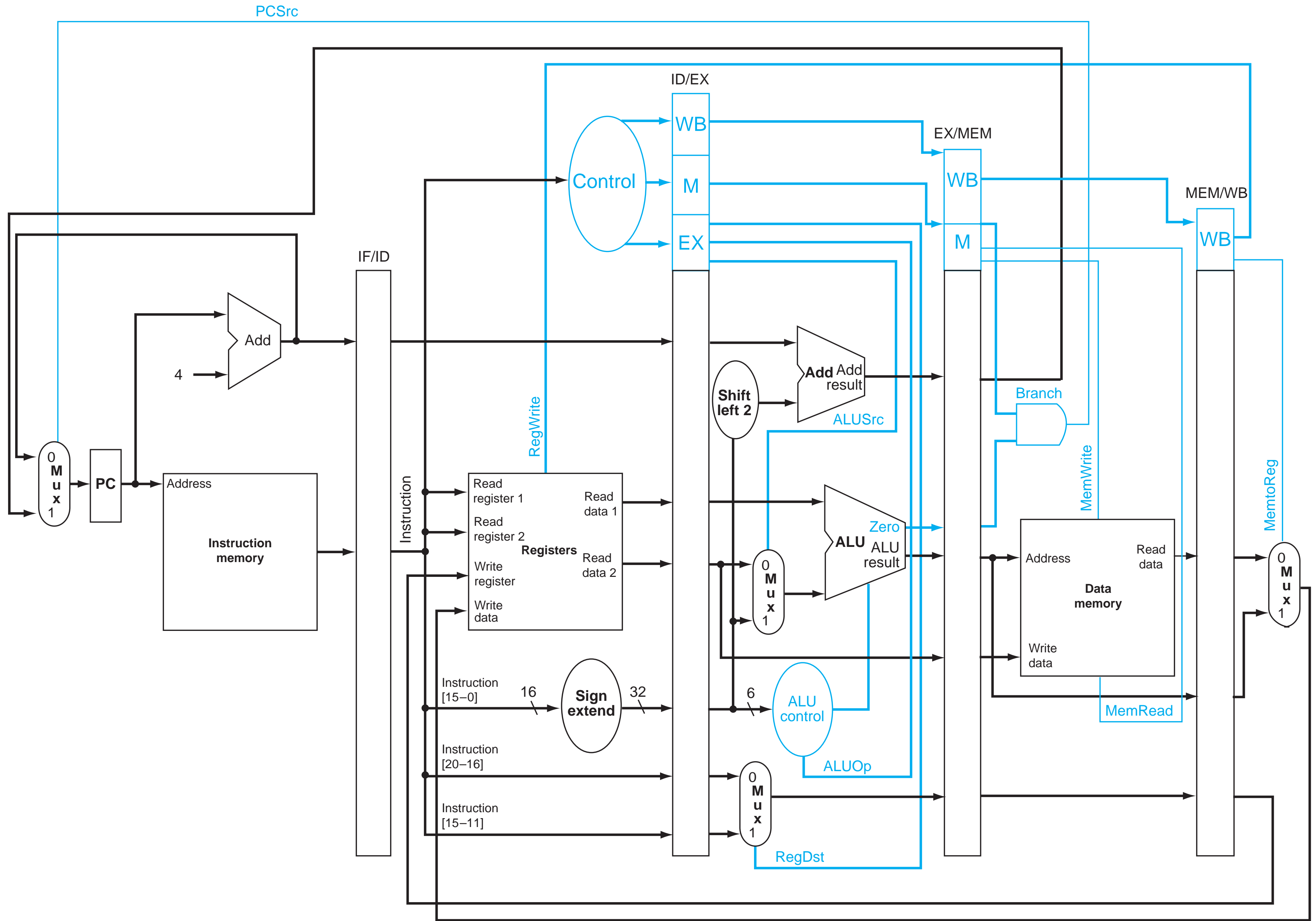
Model procesora potokowego - sygnały sterujące



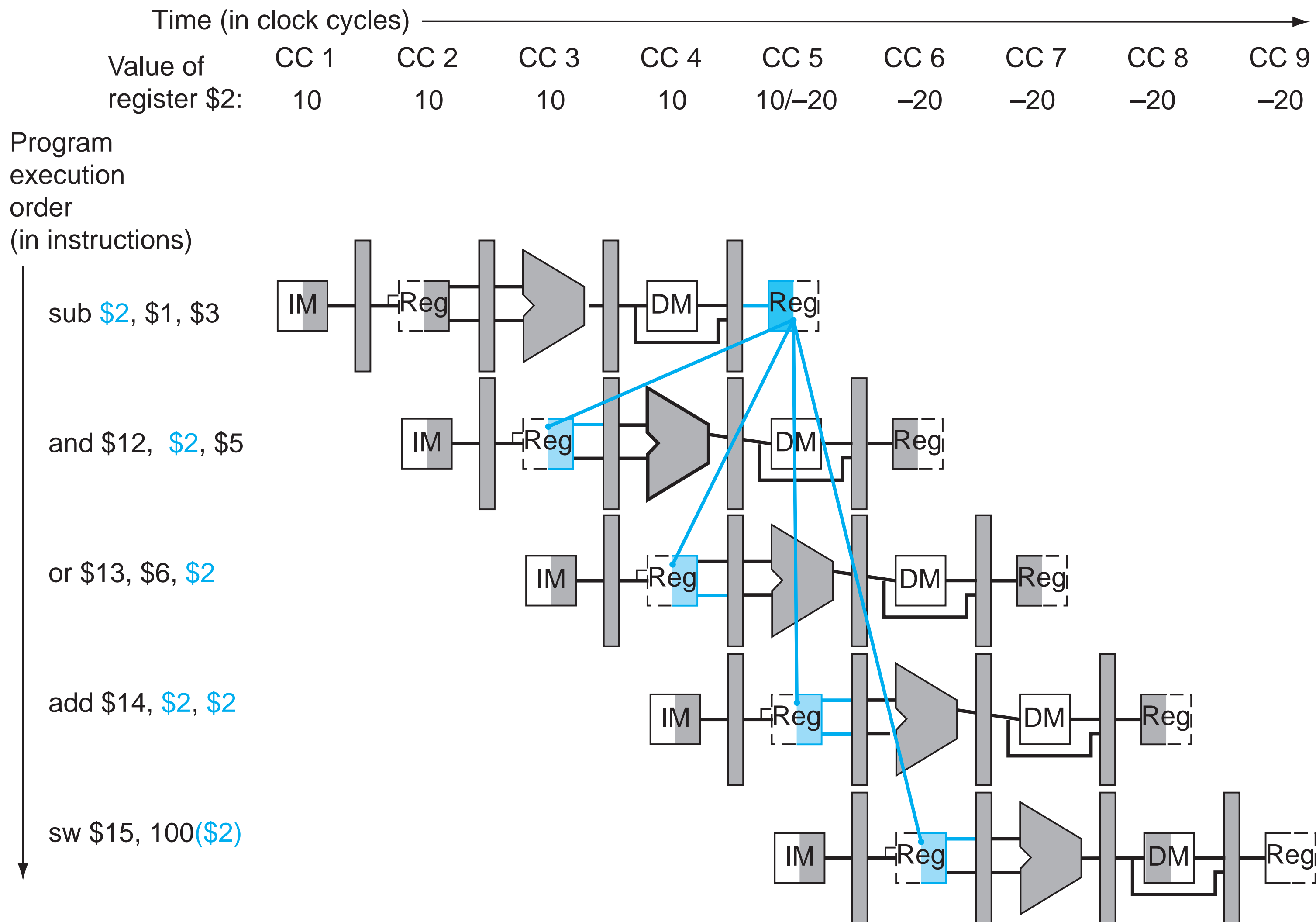
Model procesora potokowego – sterowanie

- Sygnały sterujące podobne jak w rozwiązaniu jednocyklowym
- Brak specjalnych sygnałów dla dwóch pierwszych faz (niezależne od rozkazu)



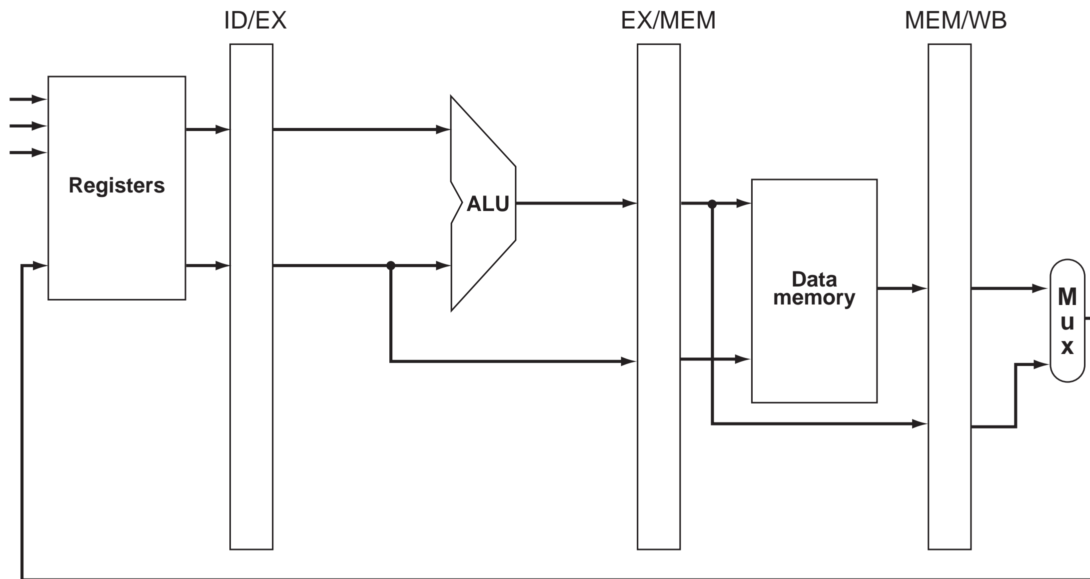

PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Control
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Instruction [15–0]
16
Sign extend
32
Instruction [20–16]
Instruction [15–11]
Shift left 2
Add
Add result
ALUSrc
ALU
Zero
ALU result
6
ALU control
ALUOp
RegDst
Branch
MemWrite
Address
Read data
Data memory
Write data
MemRead
MemtoReg
0
M
u
x
1

Hazardy danych i forwarding

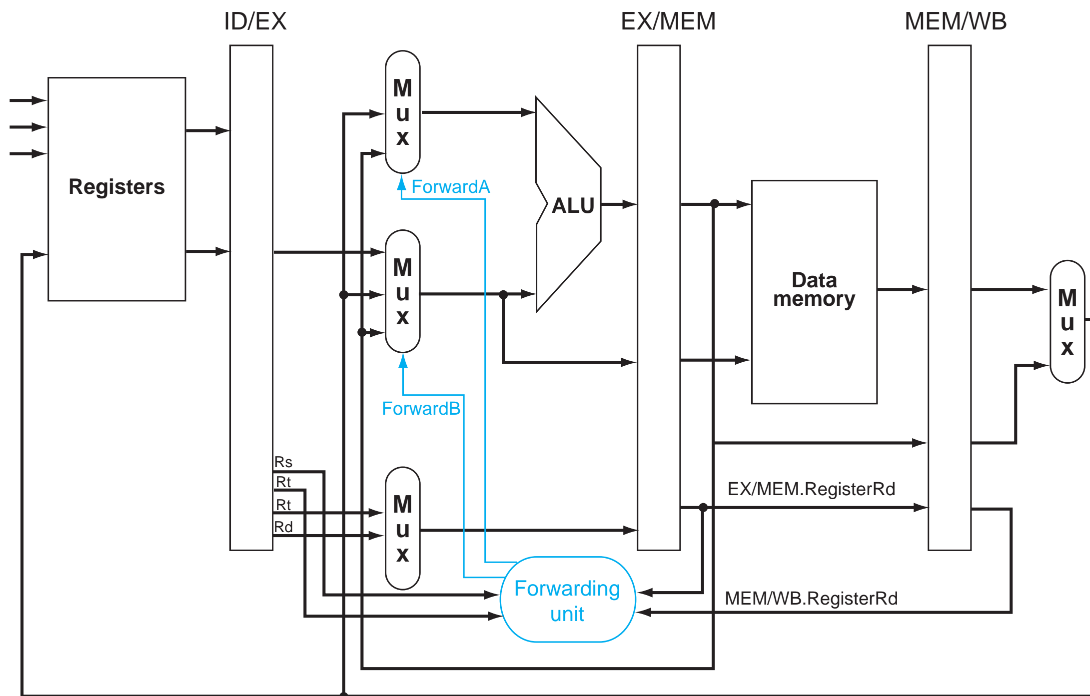


Forwarding

- Rejestry rozdzielające fazy potoku przechowują m.in. numery rejestów źródłowych (rs, rt) oraz przeznaczenia (rd) biorących udział w operacji.
- Cztery sytuacje:
  - EX/MEM.RegRd = ID/EX.RegRs
  - EX/MEM.RegRd = ID/EX.RegRt
  - MEM/WB.RegRd = ID/EX.RegRs
  - MEM/WB.RegRd = ID/EX.RegRt

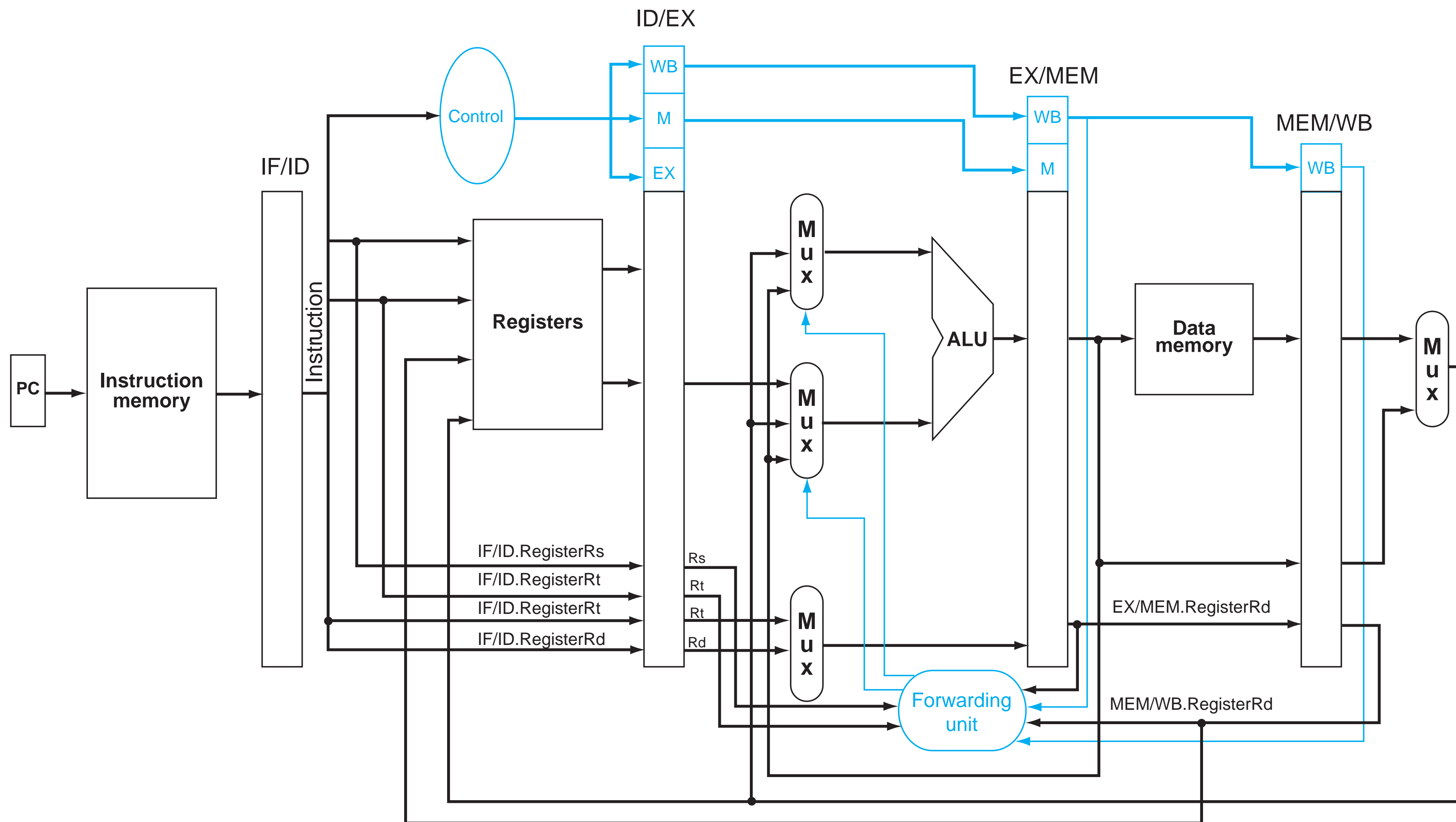


a. No forwarding



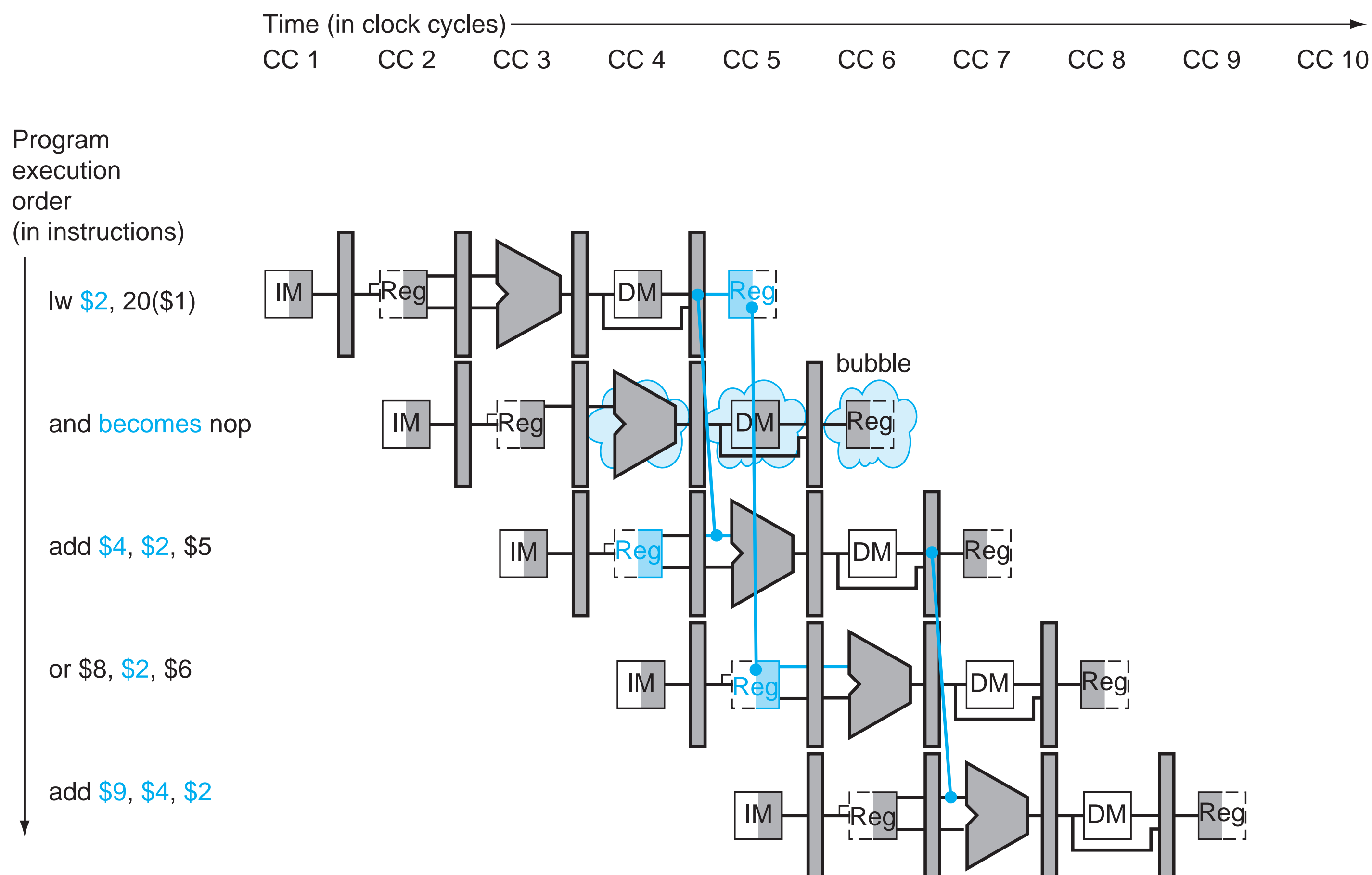
b. With forwarding

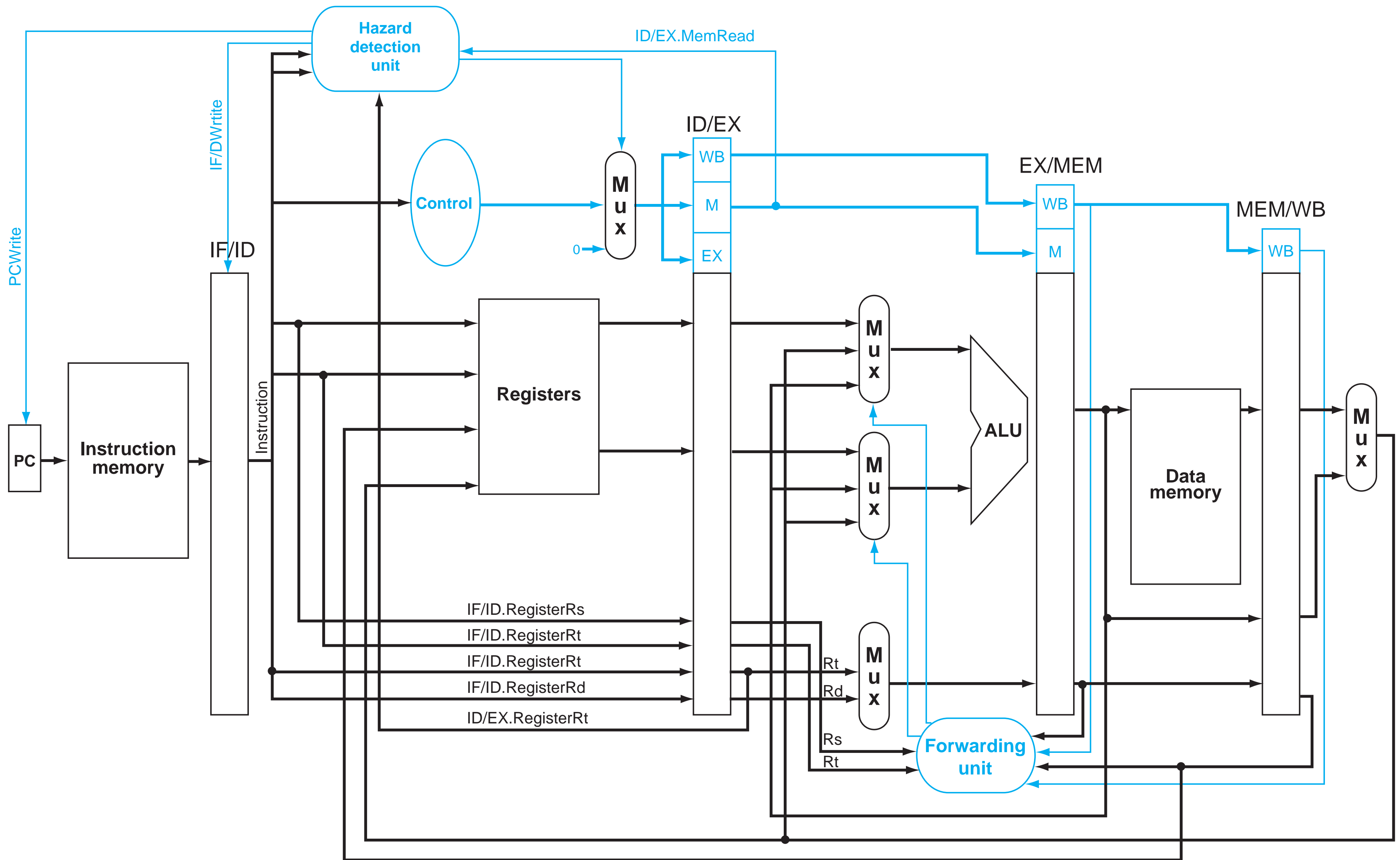
Sterowanie dla forwardingu



Hazardy nieusuwalne

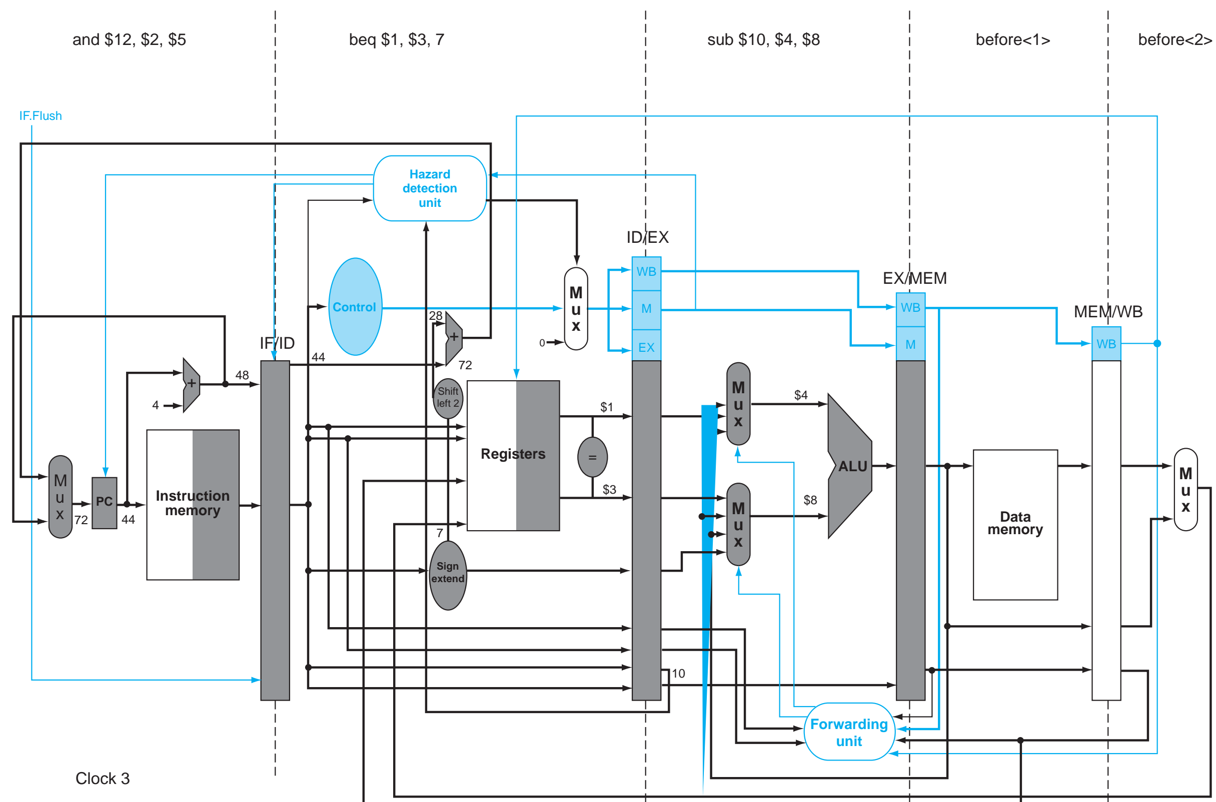
- `lw $2, 20($1)        add $4, $2, $5`
- Dana jest potrzebna na jeden cykl przed odczytem z pamięci
- Musimy stracić jeden cykl - dodajemy pojedynczy „bąbelek”



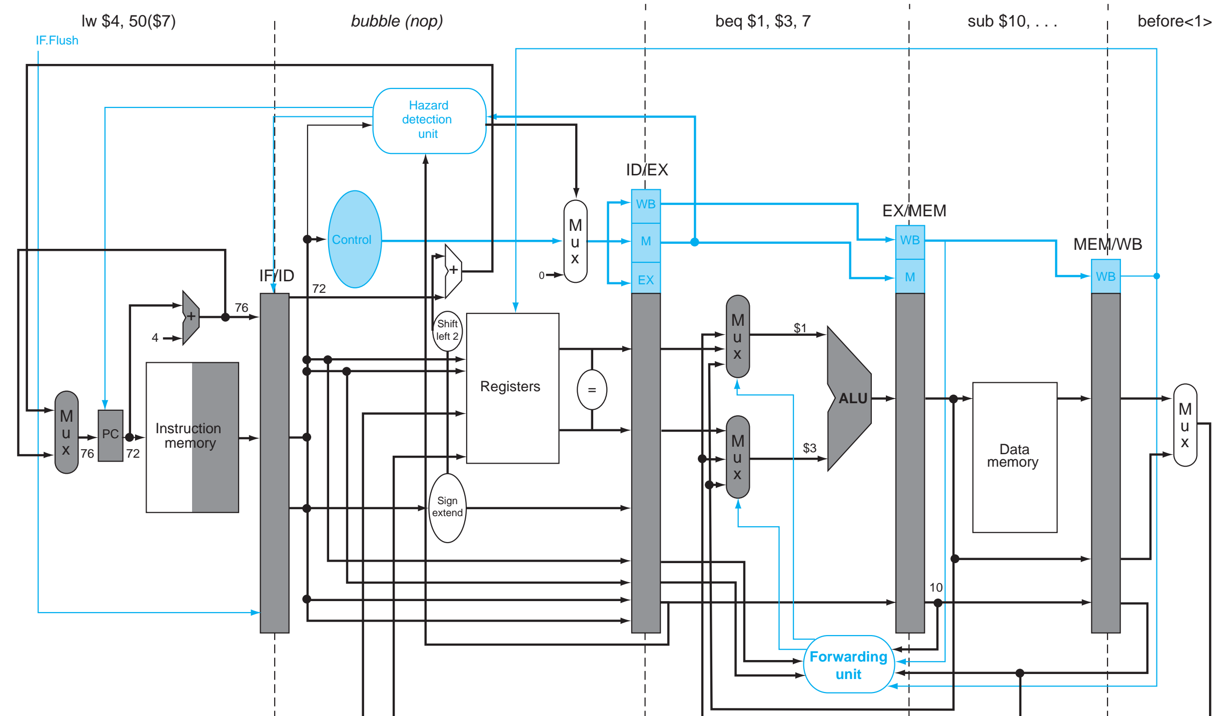

Hazard detection unit
ID/EX.MemRead
IF/DWrite
PCWrite
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
M
u
x
0
PC
Instruction memory
Instruction
Registers
ALU
Data memory
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit

Hazardy sterowania

- Przesuwamy wyliczanie adresu skoku oraz sprawdzanie, czy skok się wykona do drugiej fazy.
- Zakładamy, że skok się nie wykona – pobieramy do potoku kolejny rozkaz.
- Jeśli skok się jednak wykonał musimy wstawić jeden „bąbelek” oraz wyczyścić rejestr IF/ID



and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>
IF.Flush
Hazard detection unit
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
Mux
Registers
Instruction memory
Data memory
ALU
Forwarding unit
Sign extend
Shift left 2
PC
$1
$3
$4
$8
Clock 3



lw $4, 50($7)
bubble (nop)
beq $1, $3, 7
sub $10, . . .
before<1>
IF.Flush
Hazard detection unit
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
Mux
Registers
Instruction memory
Data memory
ALU
Forwarding unit
Sign extend
Shift left 2
PC
$1
$3

Krok dalej: superskalarność

- Nowoczesne procesory mają zazwyczaj po kilka równoległych potoków. Takie procesory nazywamy *superskalarnymi*.
- Mają wobec tego po kilka jednostek wykonawczych (np. kilka kopii ALU)
- Układy zarządzające przydziałem rozkazów do poszczególnych potoków oraz wykrywające/usuwające hazardy robią się bardzo skomplikowane.

VLIW

- Inny pomysł – zrzucić problem rozdzielania rozkazów do potoków na kompilator
- Very Large Instruction Word: długa „paczka rozkazów" składająca się z kilku prostych rozkazów
- Kompilator zapewnia, że rozkazy z jednej paczki mogą być wykonane równolegle (czasem oczywiście nie uda się wypełnić całej paczki i trzeba zostawić wolne pola).
- Przykładem są procesory Itanium (architektura IA-64), chociaż intel klasyfikuje je jako EPIC (*Explicitly Parallel Instruction Computers*).

Potoki w procesorach Intela

- potok pojawia się w modelu 486
- Pentium jest już superskalarny: dwa 5-etapowe potoki (wczytywanie, dekodowanie, generowanie adresu, wykonanie, zapisanie); za zapełnienie obydwu potoków odpowiadał kompilator.
- Pentium II miało już 12 etapów potoku, Pentium III - 14, a Pentium IV - 24.
- dla odmiany 64-bitowy procesor Itanium ma tylko 10 etapów potoku;

Itanium 2

- Dużo rejestrów.
- Potrafi wykonywać jednocześnie do 6 rozkazów
- Rozkazy 128-bitowe (3 sloty po 41 bitów + 5 bitów dodatkowych)
- Potoki 8-etapowe
- Jednostki wykonawcze:
  - 6 jednostek ALU całkowitoliczbowych, 1 przesuwająca
  - 2 * 2 jednostki zmiennopozycyjne (dod/mnoż , inne)
  - 6 jednostek wektorowych dla rozkazów multimedialnych
  - 3 jednostki skoków.
- Itanium 2 taktowany zegarem 1,5GH jest szybszy od procesorów superskalarnych taktowanych dwukrotnie szybszym zegarem!