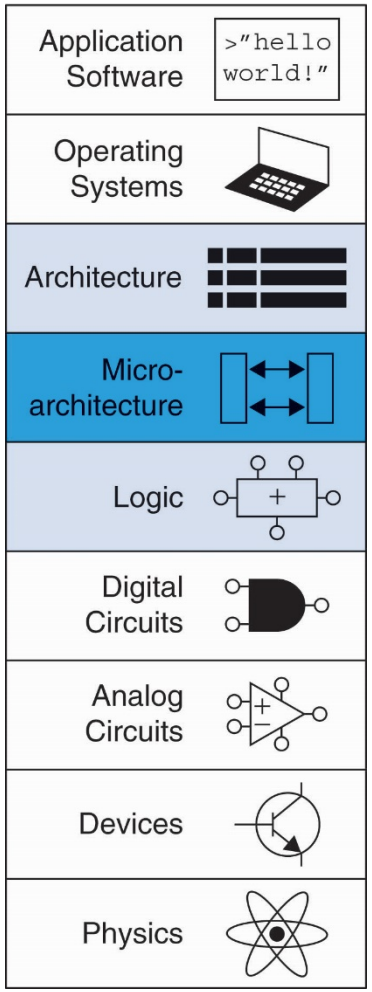
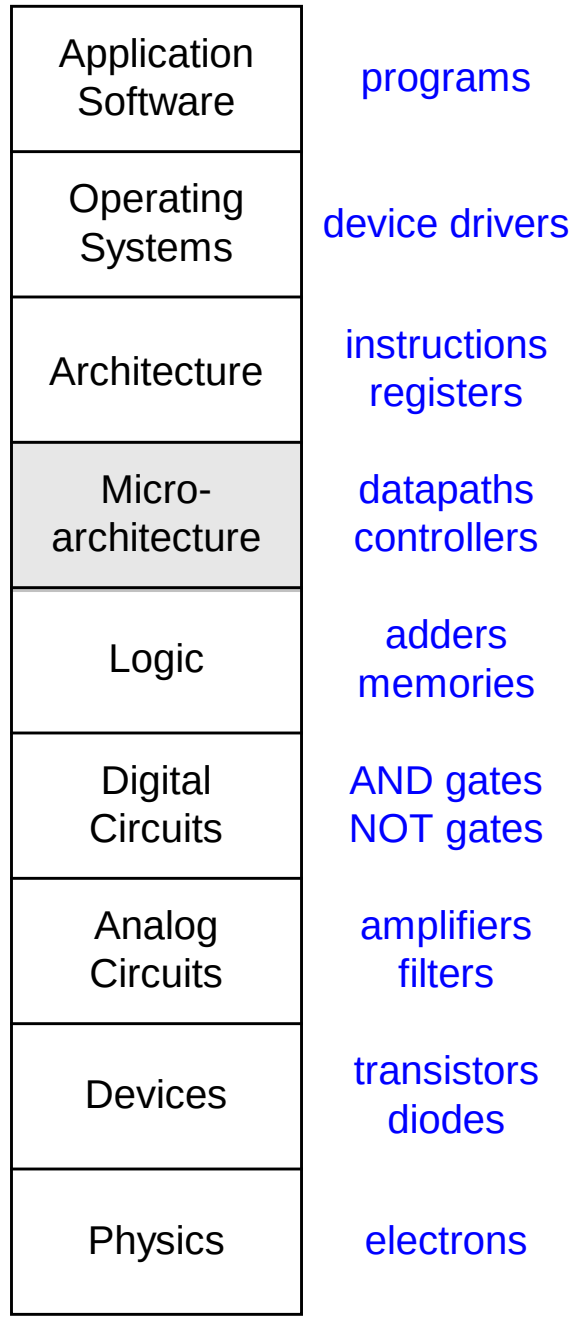
Роділ 7. Теми:

- **Вступ**
- **Аналіз продуктивності**
- **Однотактний процесор**
- **Багатотактний процесор**
- **Конвеєрний процесор**
- **Винятки**
- **Покращення мікроархітектури**



Вступ

- **Мікроархітектура:** апаратна реалізація архітектури у вигляді схеми
- Процесор:
  - **Тракт даних:** функціональні блоки обробки і передачі даних (арифметико-логічний пристрій, регістровий файл, мультиплексори і т.д.)
  - **Пристрій керування:** формує керуючі сигнали для функціональних блоків



Мікроархітектура

- Декілька апаратних реалізацій однієї і тієї ж архітектури:
  - **Однотактна реалізація:** кожна інструкція виконується за один такт
  - **Багатотактна реалізація:** кожна інструкція розбивається на декілька кроків і виконується за декілька кроків
  - **Конвеєрна реалізація:** кожна інструкція розбивається на декілька кроків і декілька інструкцій виконуються одночасно

Продуктивність процесора

- Час виконання програми

  Execution Time = (#instructions)(cycles/instruction)(seconds/cycle)

  Час виконання = (#інструкції)(такти/інструкція)(секунди/такт)

- Визначення:
  - CPI: Кількість тактів на виконання інструкції (Cycles/instruction)
  - Період тактової частоти: секунди/такт
  - IPC: Кількість інструкцій виконаних за такт (instructions/cycle - IPC=1/CPI)
- Необхідно забезпечити наступні обмеження:
  - Вартість
  - Площа на кристалі
  - Енергоспоживання
  - Продуктивність

MIPS процесор

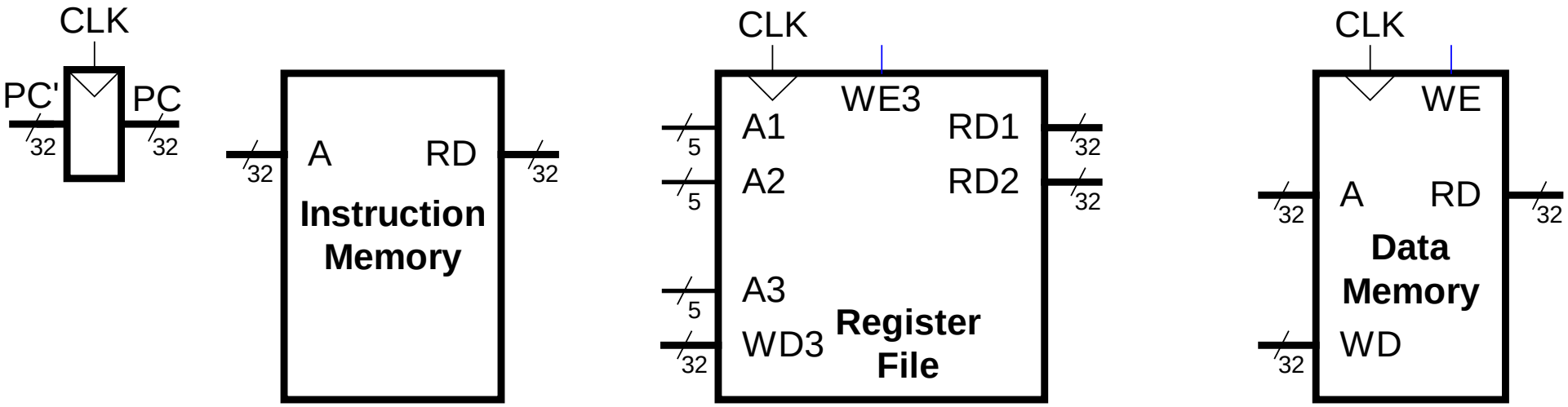
- Розглядується наступна підмножина інструкцій MIPS:
  - Інструкції R-типу: and, or, add, sub, slt
  - Інструкції роботи з пам’яттю: lw, sw
  - Інструкції переходів: beq, j

Архітектурний стан

Визначається:

- Значенням лічильника команд (program counter, PC)
- Вмістом 32-х регістрів загального призначення
- Вмістом пам’яті

Елементи, які зберігають стан MIPS

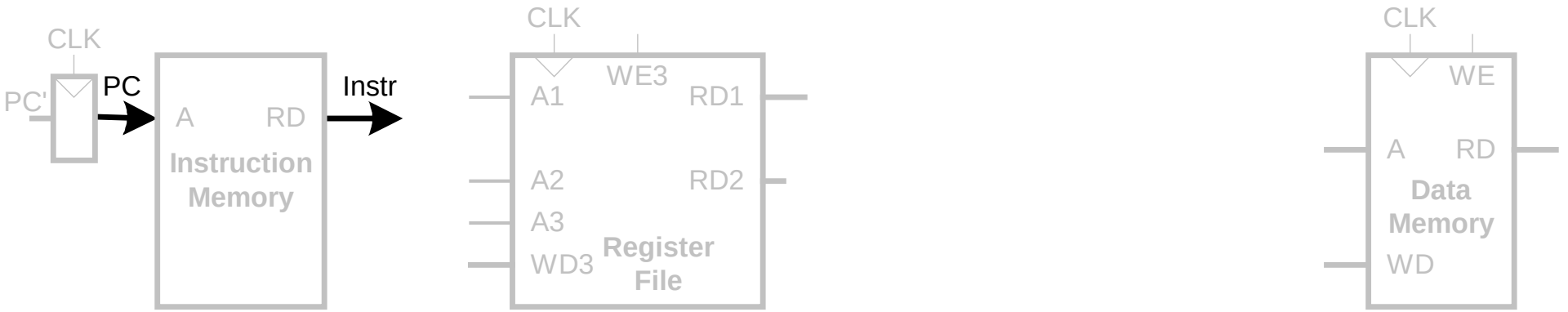


Однотактний MIPS процесор

- тракт даних
- пристрій керування

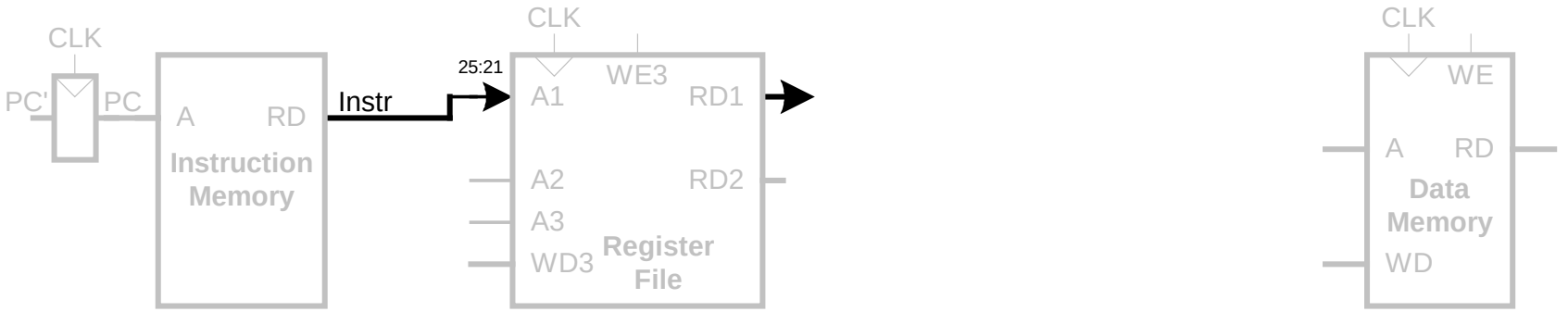
Однотактний тракт даних: вибірка *lw*

Крок 1: Вибірка (зчитування) інструкції *lw* з пам’яті інструкцій



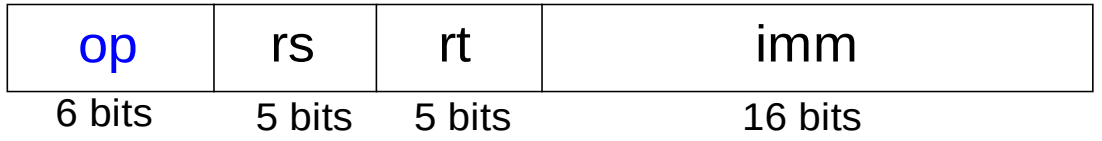
Однотактний тракт даних: читання регістрів

Крок 2: для інструкції *lw* зчитування операндів-джерела з регістрового файлу



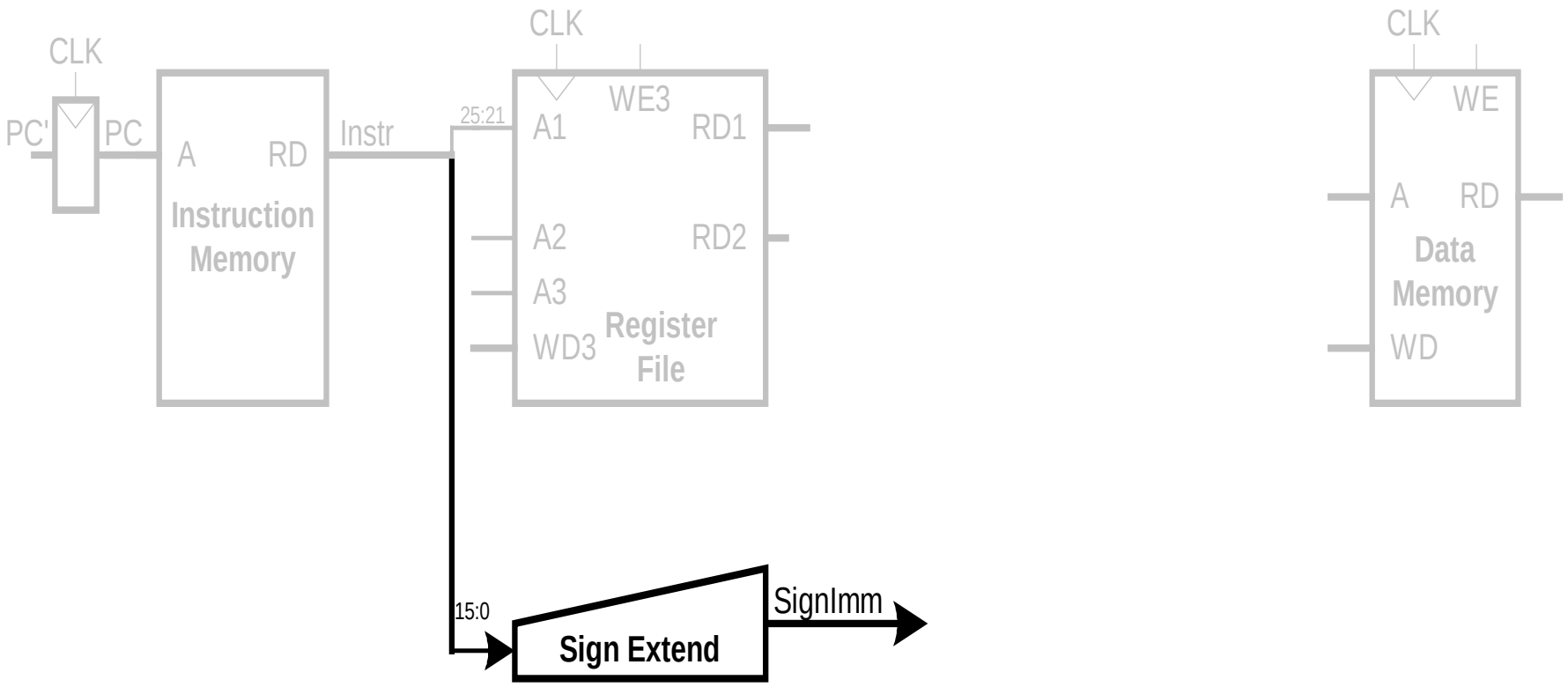

```
lw rt, imm(rs)
```

I-Type



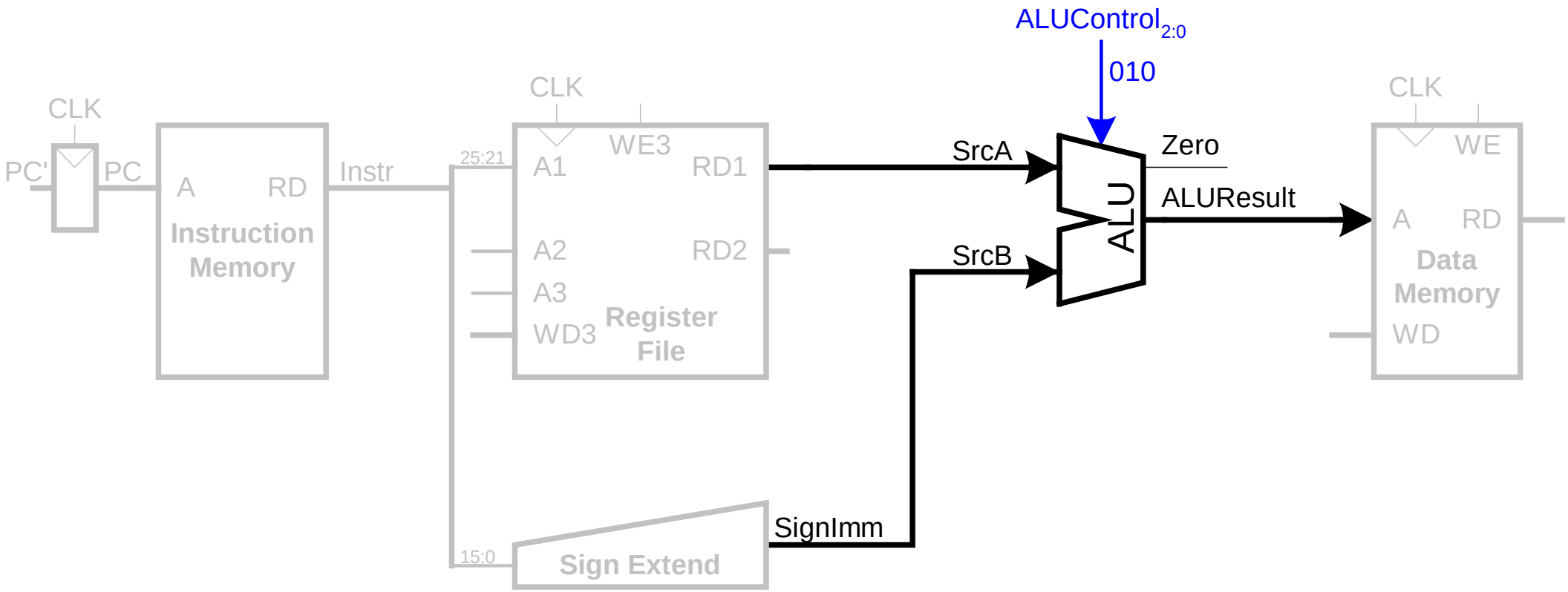
Однотактний тракт даних: розширення константи

Крок 3: розширення 16-бітової константи до 32-х розрядів бітом знаку



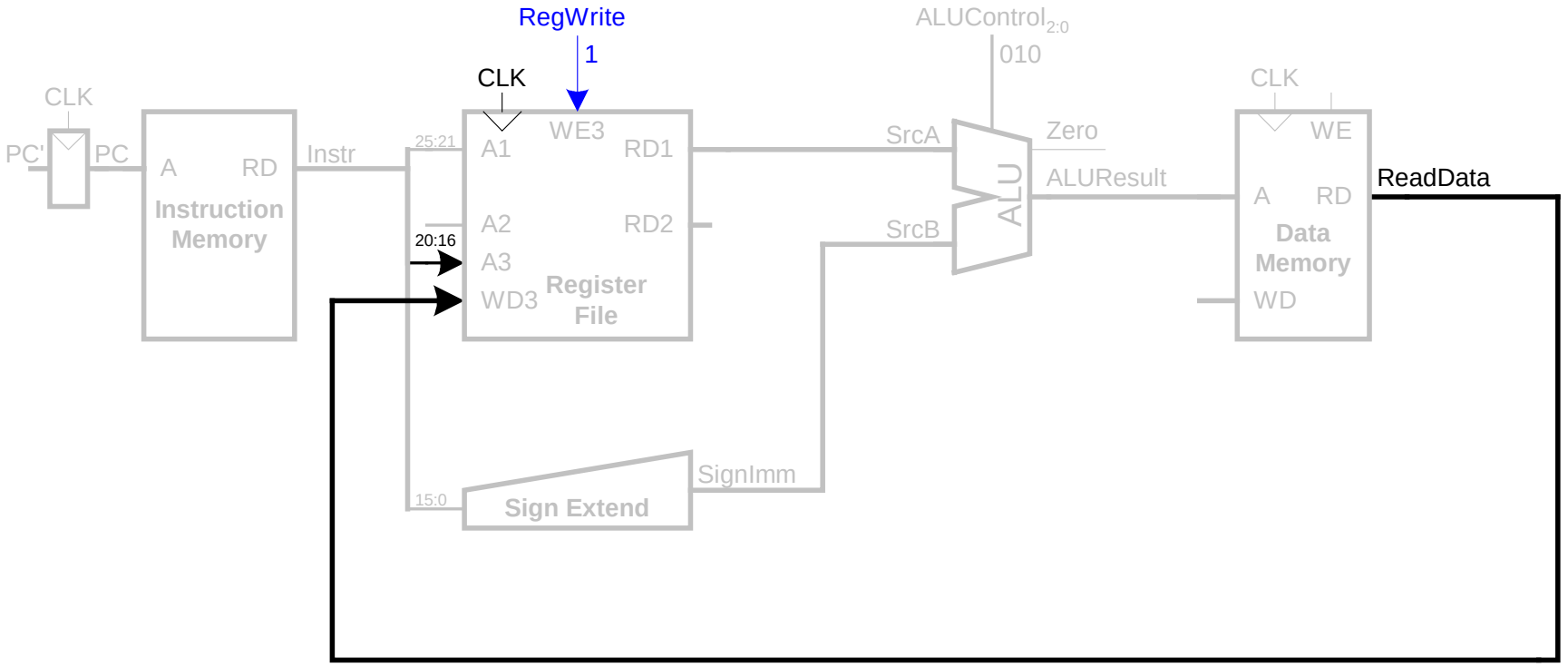
Однотактний тракт даних: обчислення адреси

Крок 4: Обчислення адреси комірки в пам’яті даних



Однотактний тракт даних: зчитування з пам’яті

- **Крок 5:** зчитування даних з пам’яті даних і записування їх в регістр, номер якого зберігається в коді інструкції



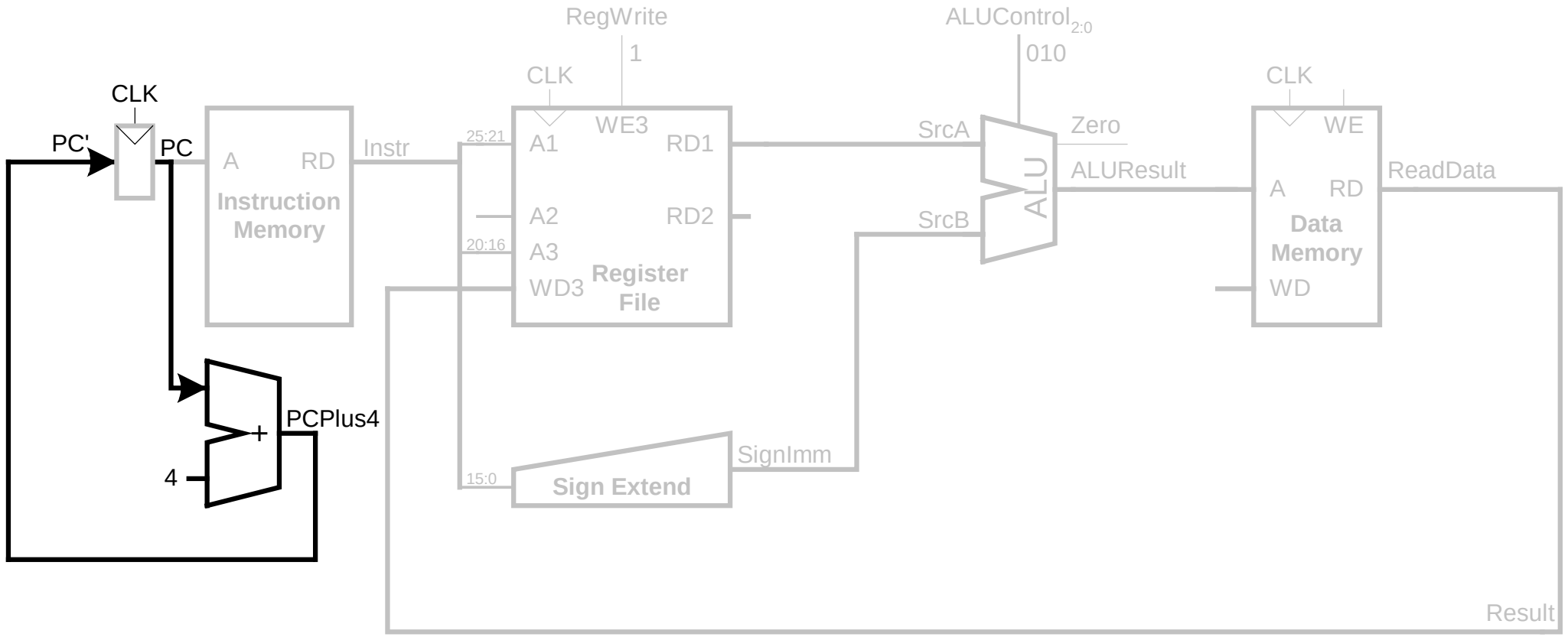
`lw rt, imm(rs)`

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

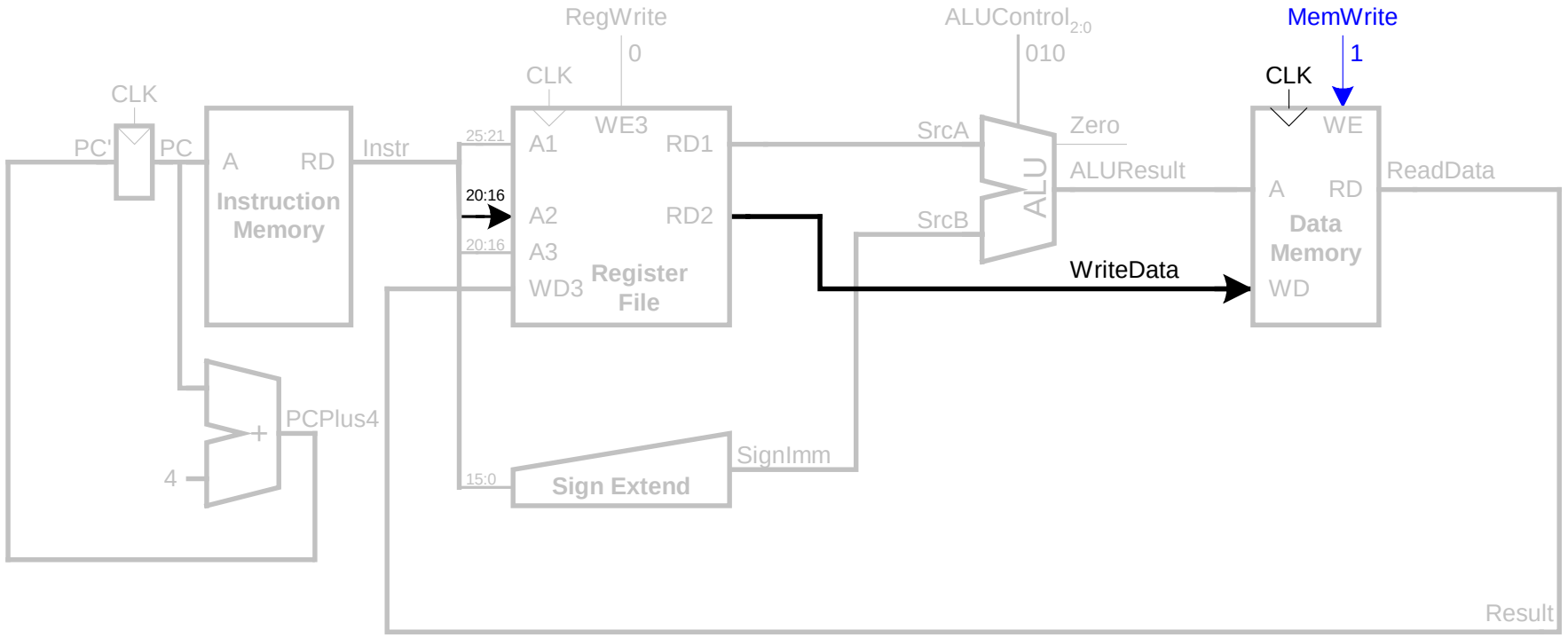
Однотактний тракт даних: збільшення лічильника команд PC

Крок 6: Обчислення адреси наступної інструкції



Однотактний тракт даних: інструкція *sw*

Запис вмісту регіста *rt* в пам’ять



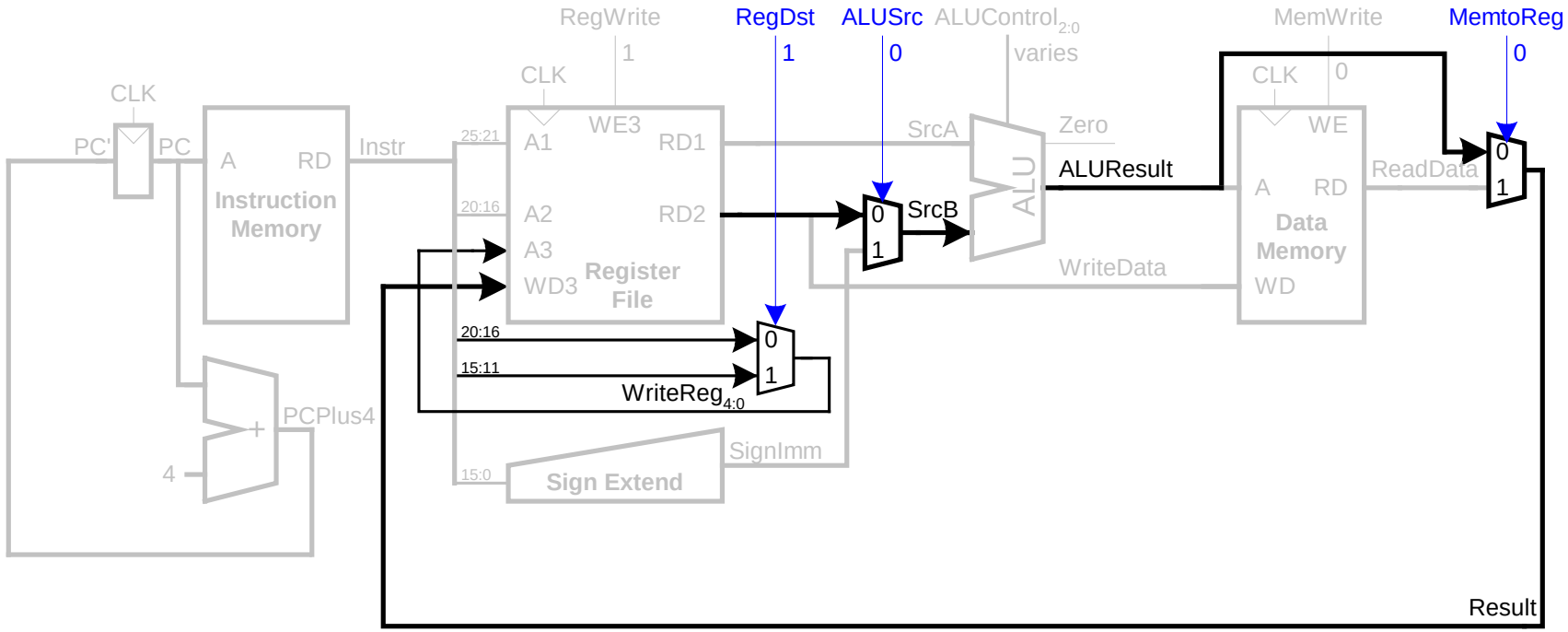
`sw rt, imm(rs)`

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Однотактний тракт даних: інструкції R-типу

- Зчитування операндів з регістрів *rs* і *rt*
- Записування *ALUResult* в регістр з номером з поля *rd* інструкції (для інструкцій I-типу результат записується в регістр з номером *rt*)

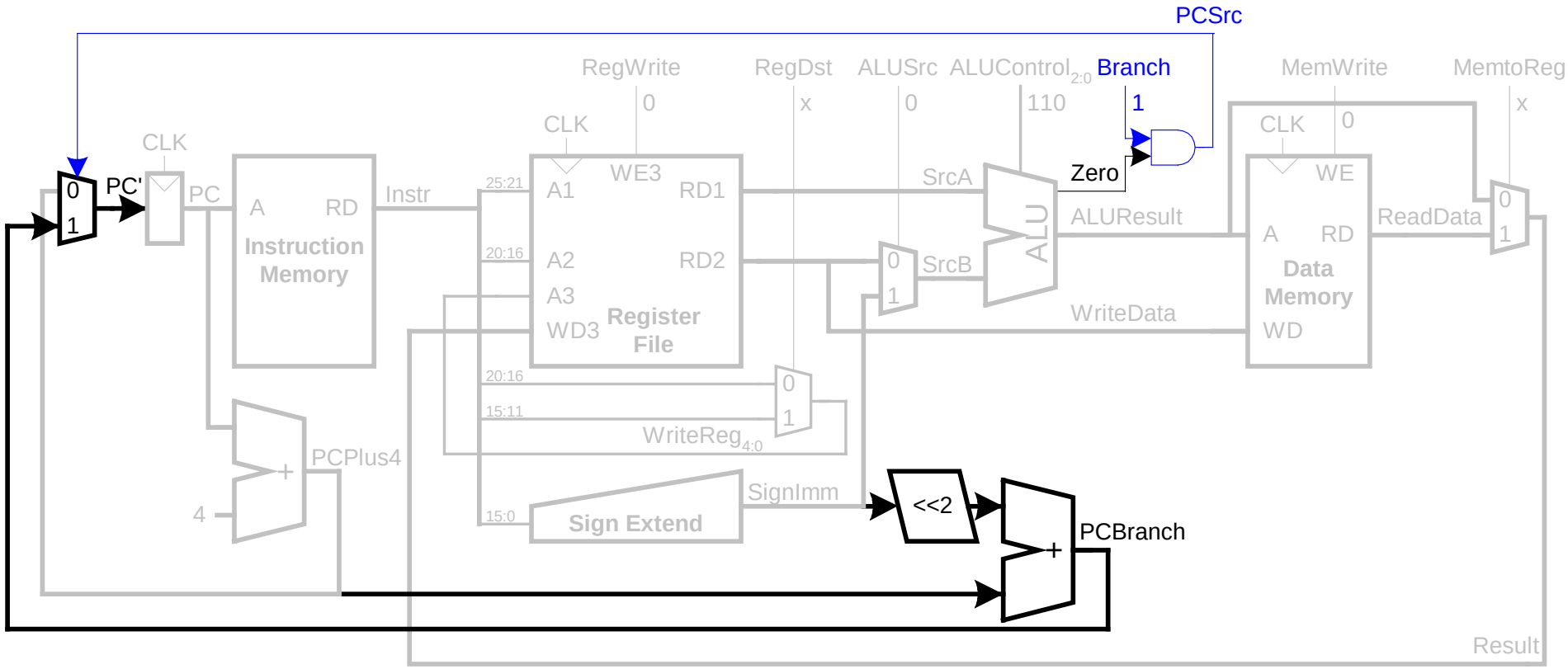


R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Однотактний тракт даних: інструкція *beq*

- Перевірка на рівність регістрів *rs* і *rt*
- Розрахунок адреси для умовного переходу: BTA = (sign-extended immediate << 2) + (PC+4)



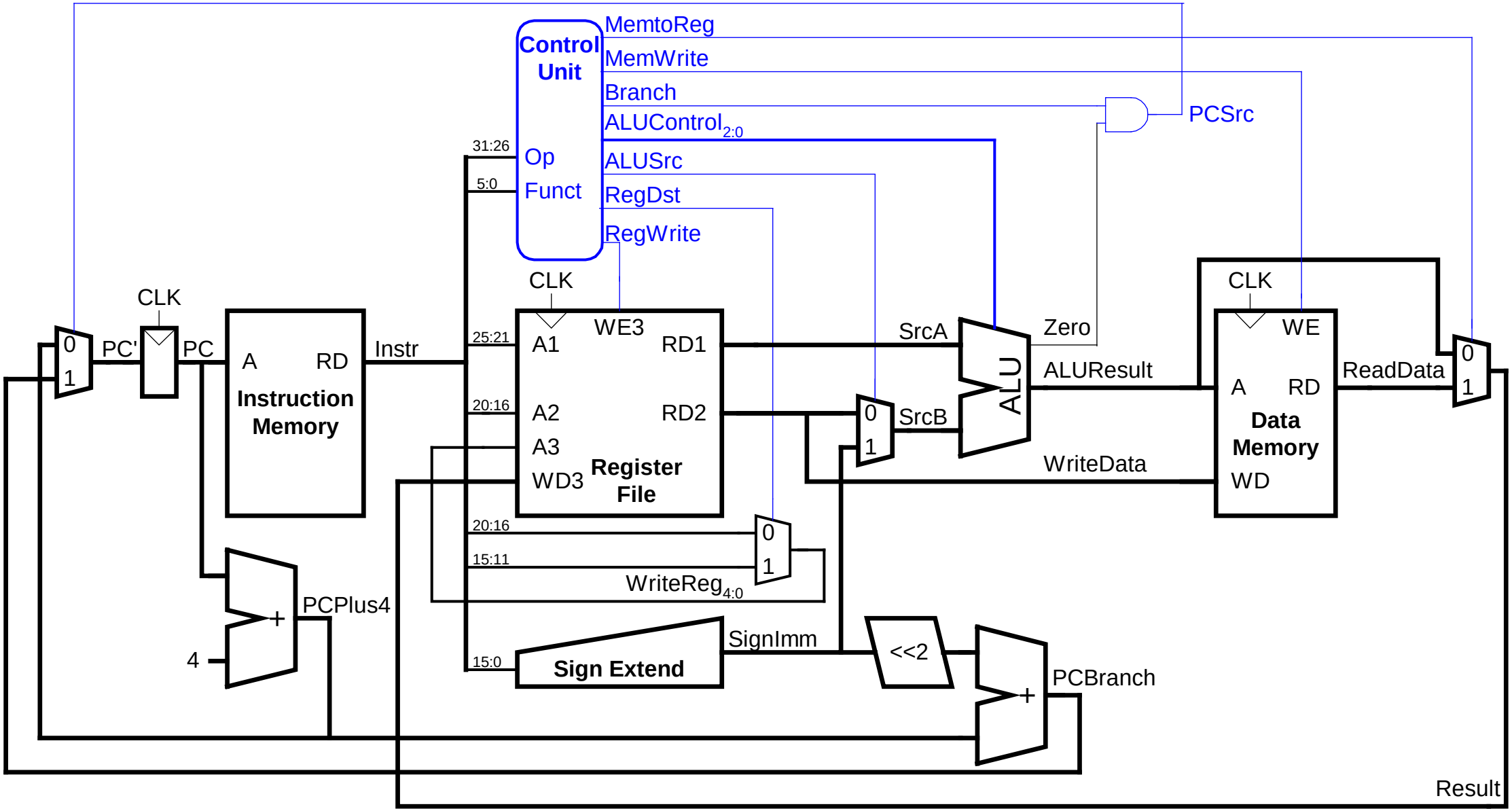
Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

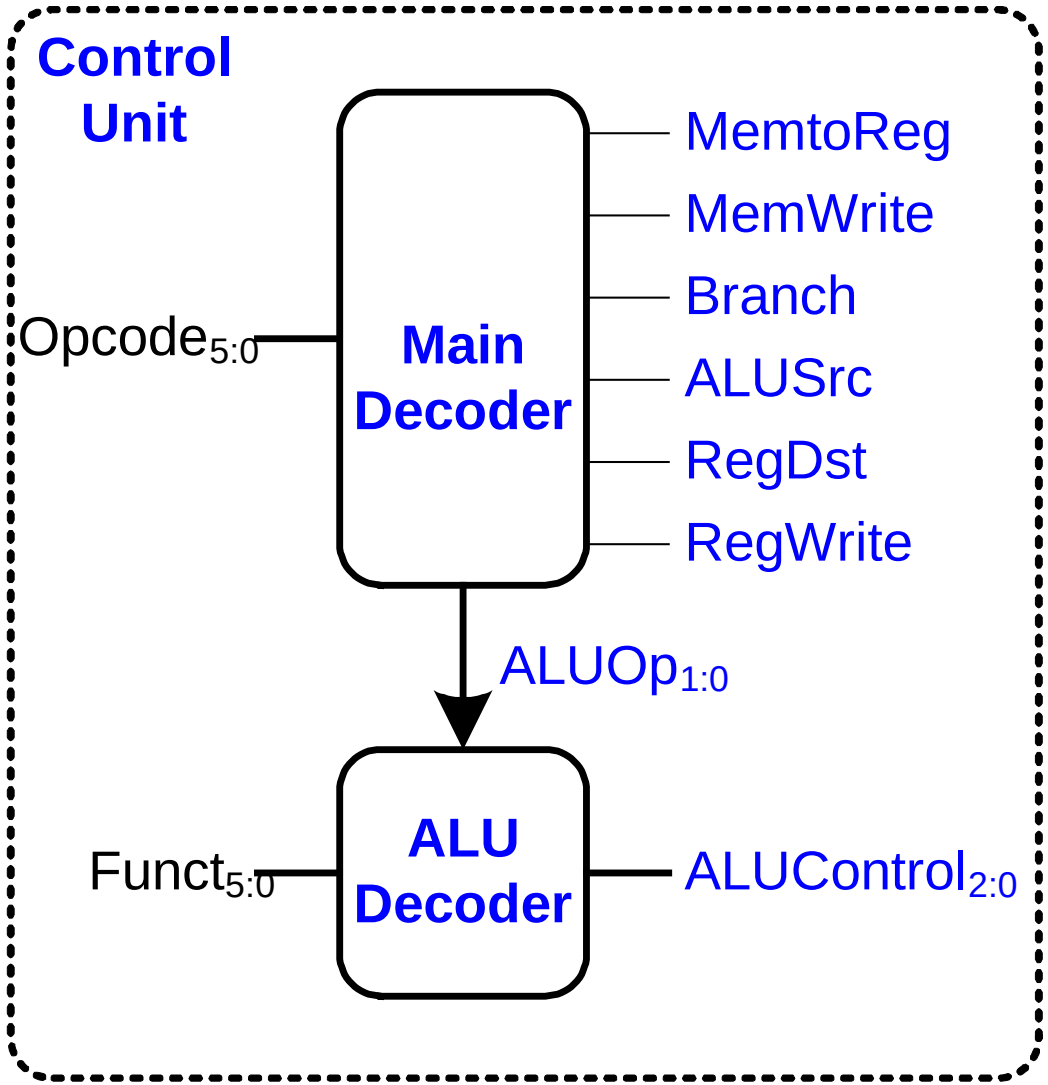
Field Values

op	rs	rt	imm		
4	8	0	3		
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

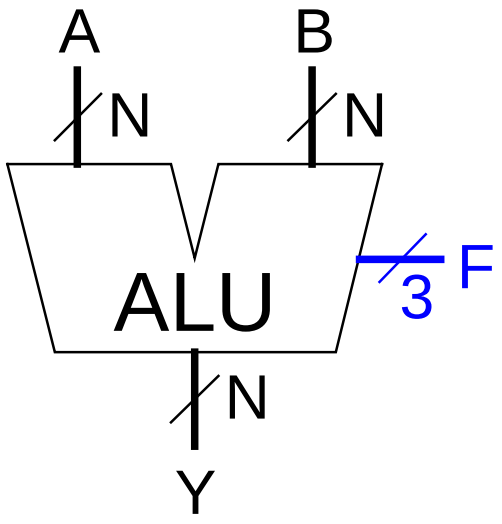
Однотактний процесор



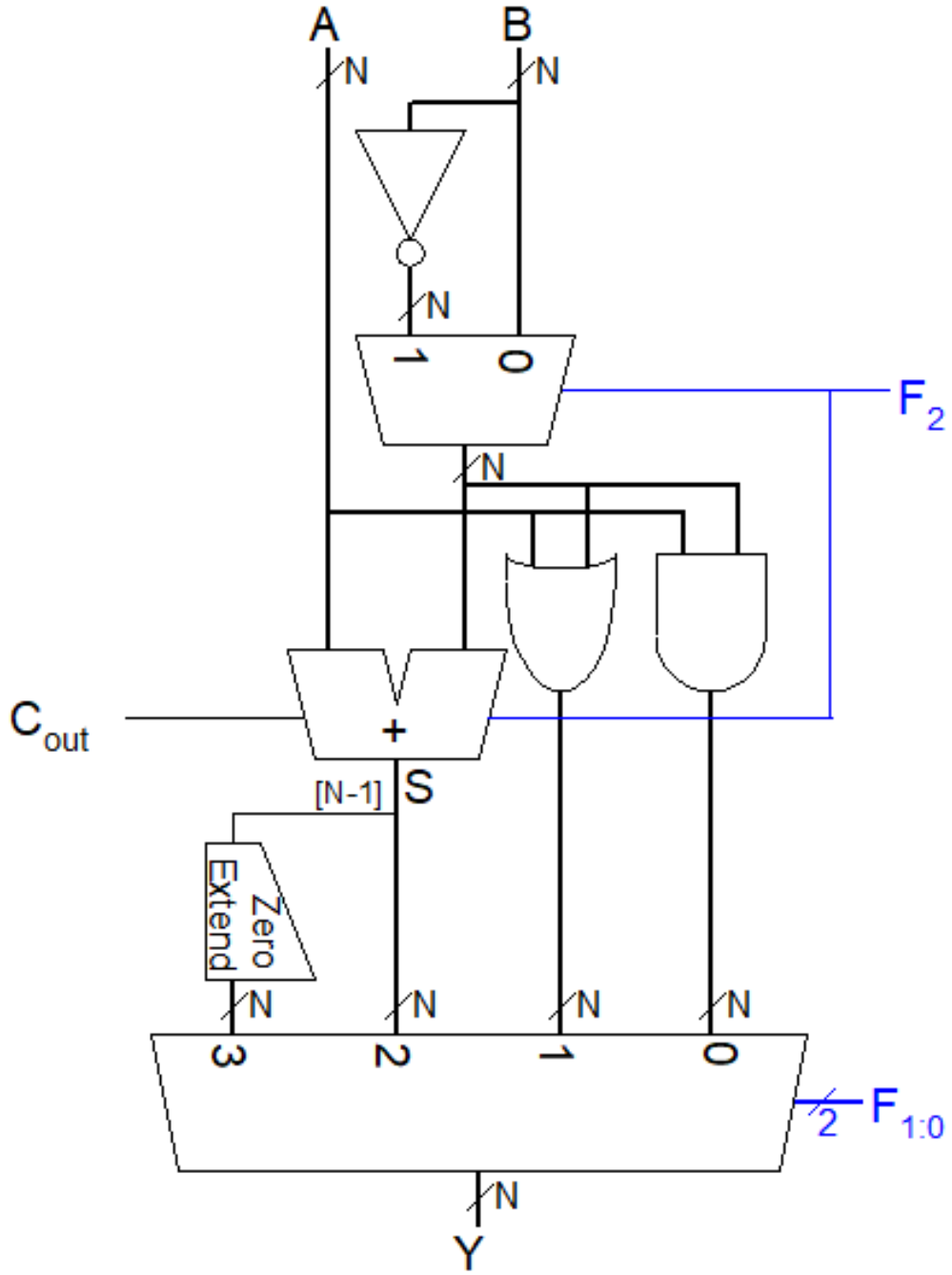
Керування однотактним процесором



Принцип роботи АЛП



$F_{2:0}$	Функція
000	A & B
001	A | B
010	A + B
011	Не вик.
100	A & ~B
101	A | ~B
110	A - B
111	SLT



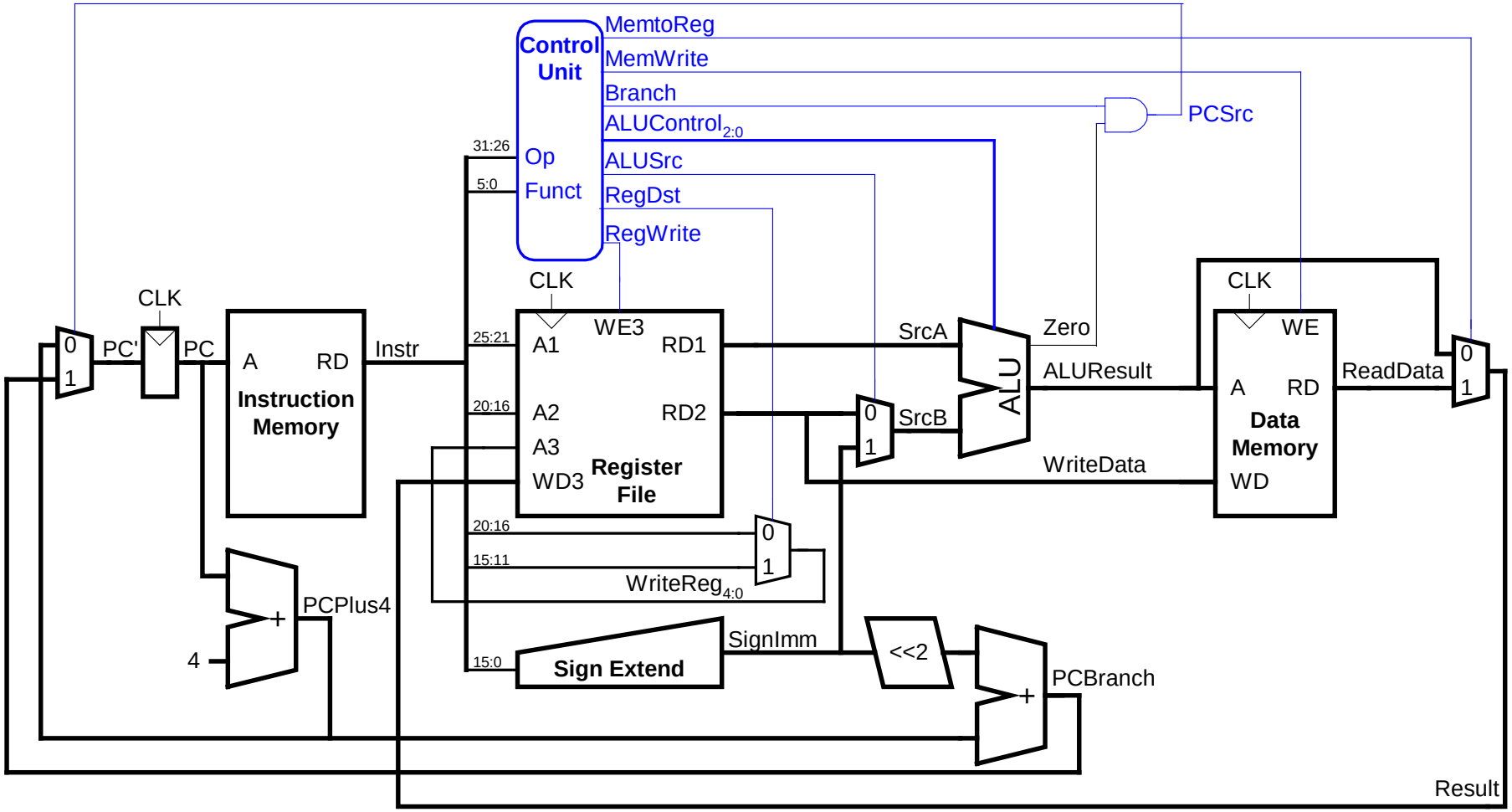
Керуючий пристрій: дешифратор АЛП

$ALUOp_{1:0}$	**Дія**
00	Додавання
01	Віднімання
10	Визначається полем Funct
11	Не використовується

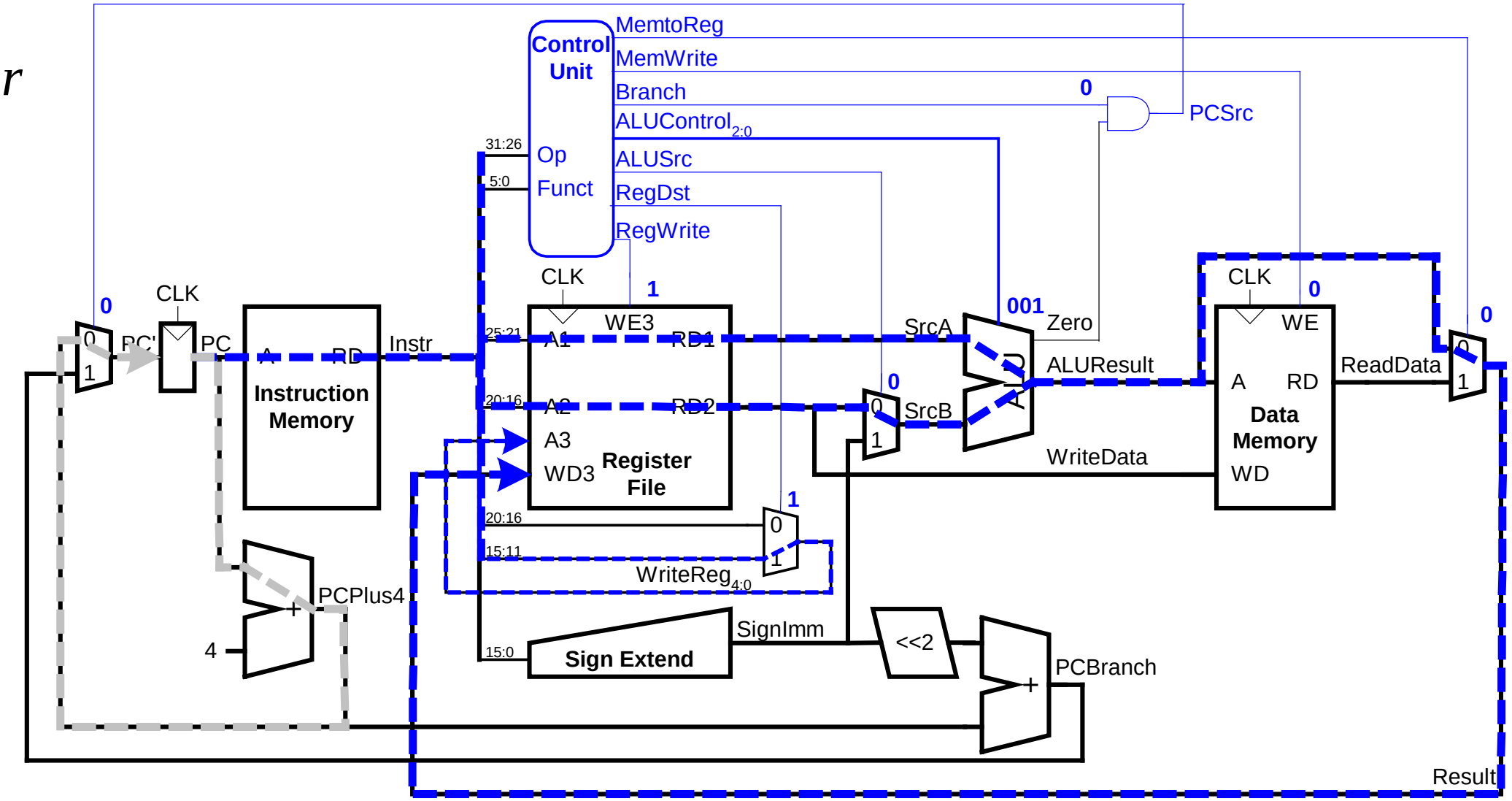
$ALUOp_{1:0}$	**Funct**	**$ALUControl_{2:0}$**
00	X	010 (Додавання)
X1	X	110 (Віднімання)
1X	100000 (`add`)	010 (Додавання)
1X	100010 (`sub`)	110 (Віднімання)
1X	100100 (`and`)	000 (I)
1X	100101 (`or`)	001 (АБО)
1X	101010 (`slt`)	111 (SLT)

Керуючий пристрій: основний дешифратор

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	0	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01



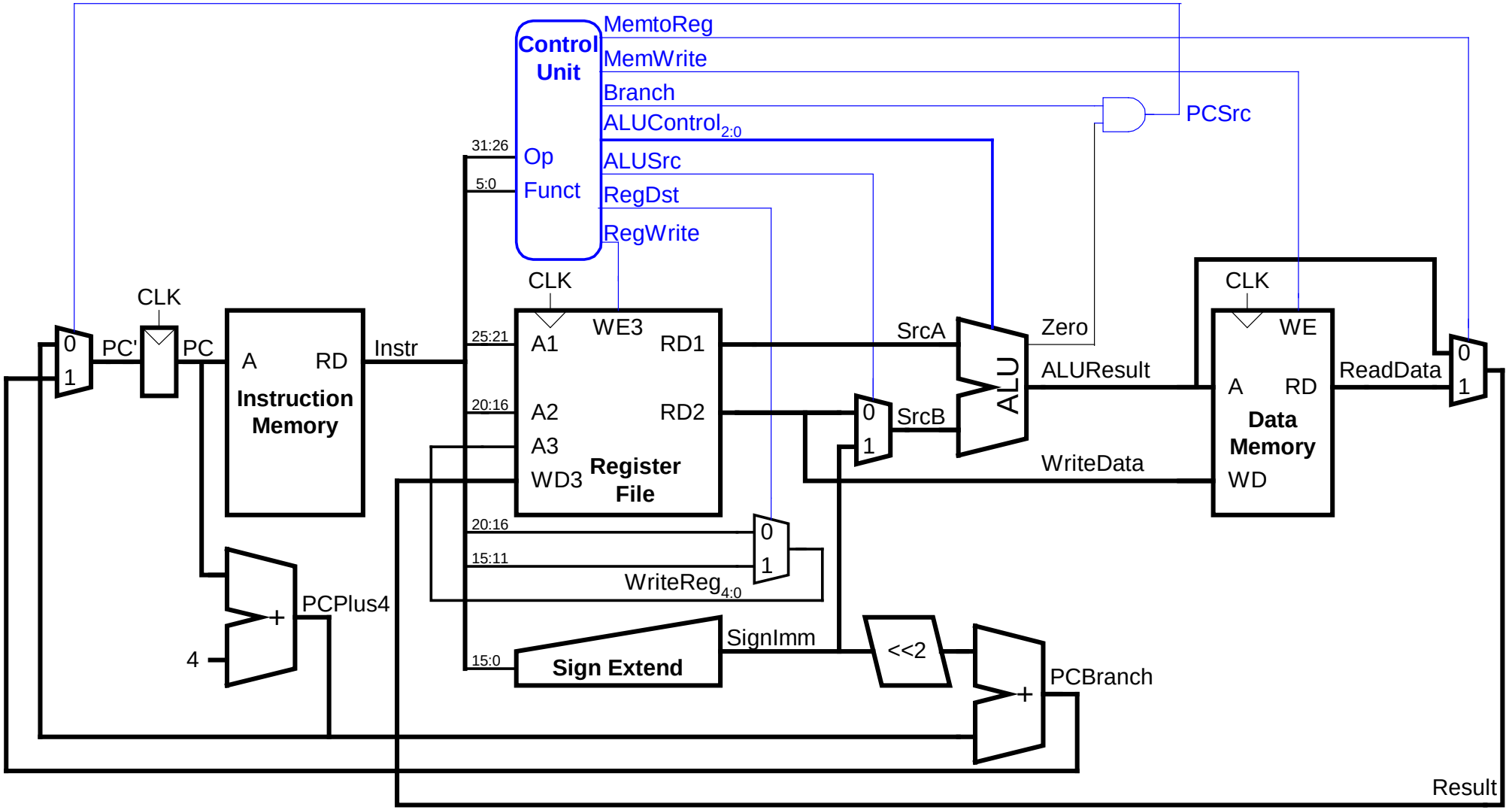
Однотактний тракт даних: *or*



R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Добавлення інструкції *addi*

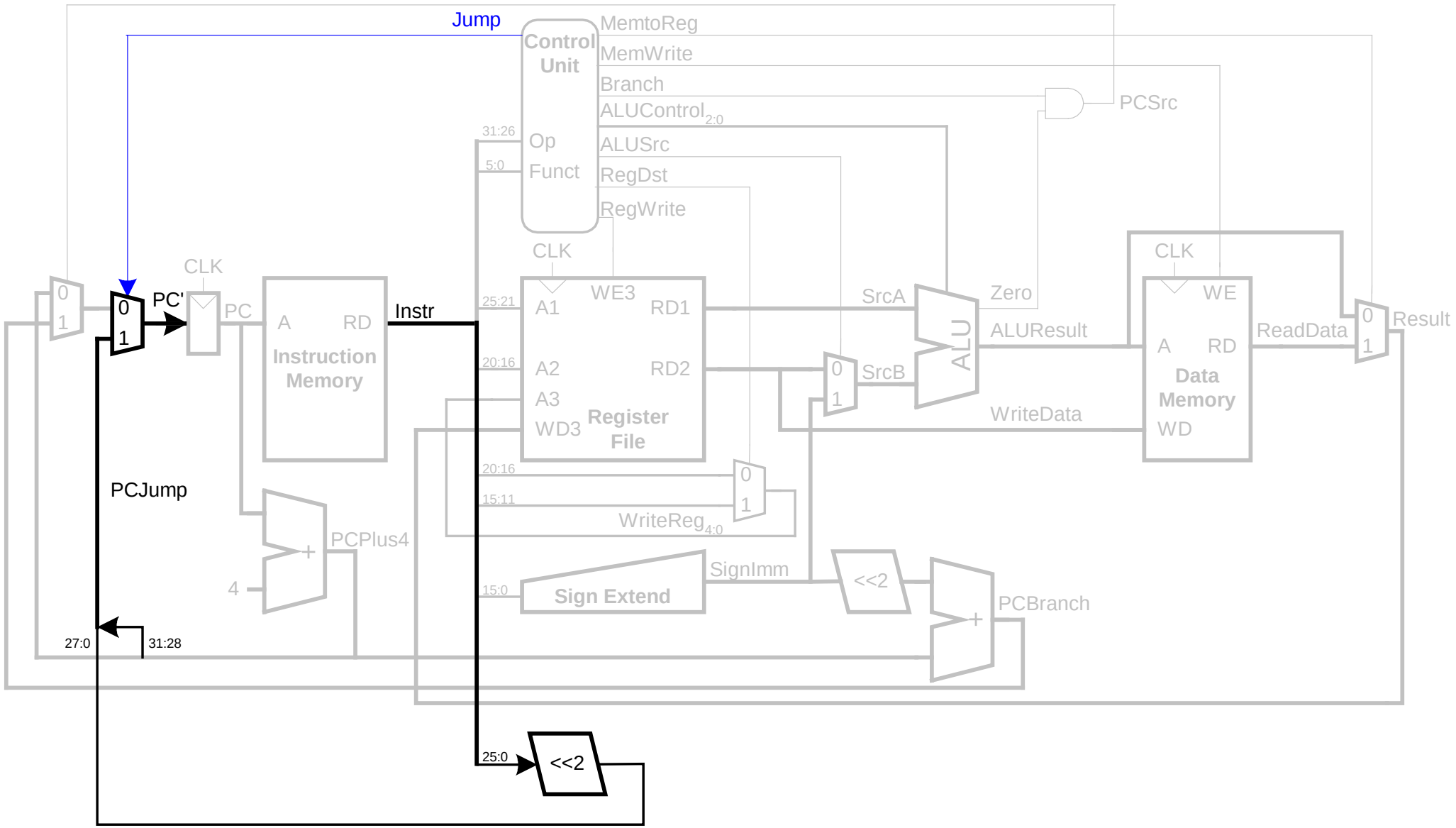


Необхідно сформувати керуючі сигнали, а тракт даних міняти не потрібно

Керуючий пристрій *addi*

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	**001000**	**1**	**0**	**1**	**0**	**0**	**0**	**00**

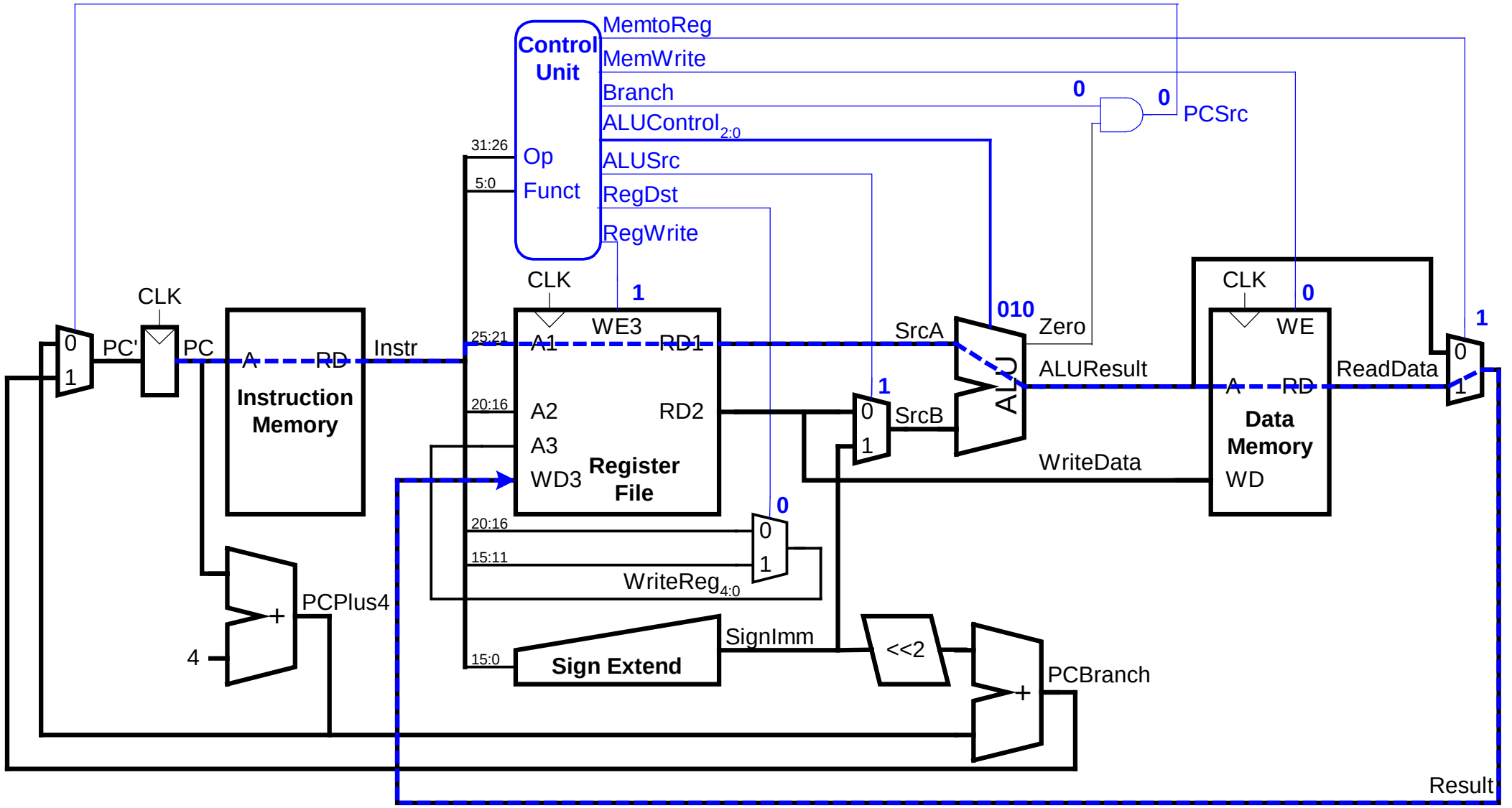
Добавлення функціоналу *j*



Керуючий пристрій *j*

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	**0**
`lw`	100011	1	0	1	0	0	1	00	**0**
`sw`	101011	0	X	1	0	1	X	00	**0**
`beq`	000100	0	X	0	1	0	X	01	**0**
`j`	**000100**	**0**	**X**	**X**	**X**	**0**	**X**	**XX**	**1**

Продуктивність однотактного процесора



CPI=1. T_C визначається ланцюгом з найбільшою затримкою (*lw*)

Продуктивність однотактного процесора

- Затримка самого довгого ланцюга комбінаційної логіки:
  $T_c = t_{pcq\_PC} + t_{\text{mem}} + \max(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$
- Звичайно на тривалість періоду більше всього впливають:
  - пам’ять, АЛП, регістровий файл
  - $T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$

Розрахунок продуктивності однотактного процесора

Параметр	**Позна-чення**	**Затрим-ка (пс)**
Час запису у регістр	t_{pcq_PC}	30
Час передвстанов-лення регістру	t_{setup}	20
Затримка мультиплексору	t_{mux}	25
Затримка АЛП	t_{ALU}	200
Затримка зчитування з пам’яті	t_{mem}	250
Затримка зчитування з регістрового файлу	t_{RFread}	150
Час передвстанов-лення регістрового файлу	$t_{RFsetup}$	20

Максимальна затримка:

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ пс}$$
$$= 925 \text{ пс}$$

Час виконання $N=100\times10^9$ інструкцій:

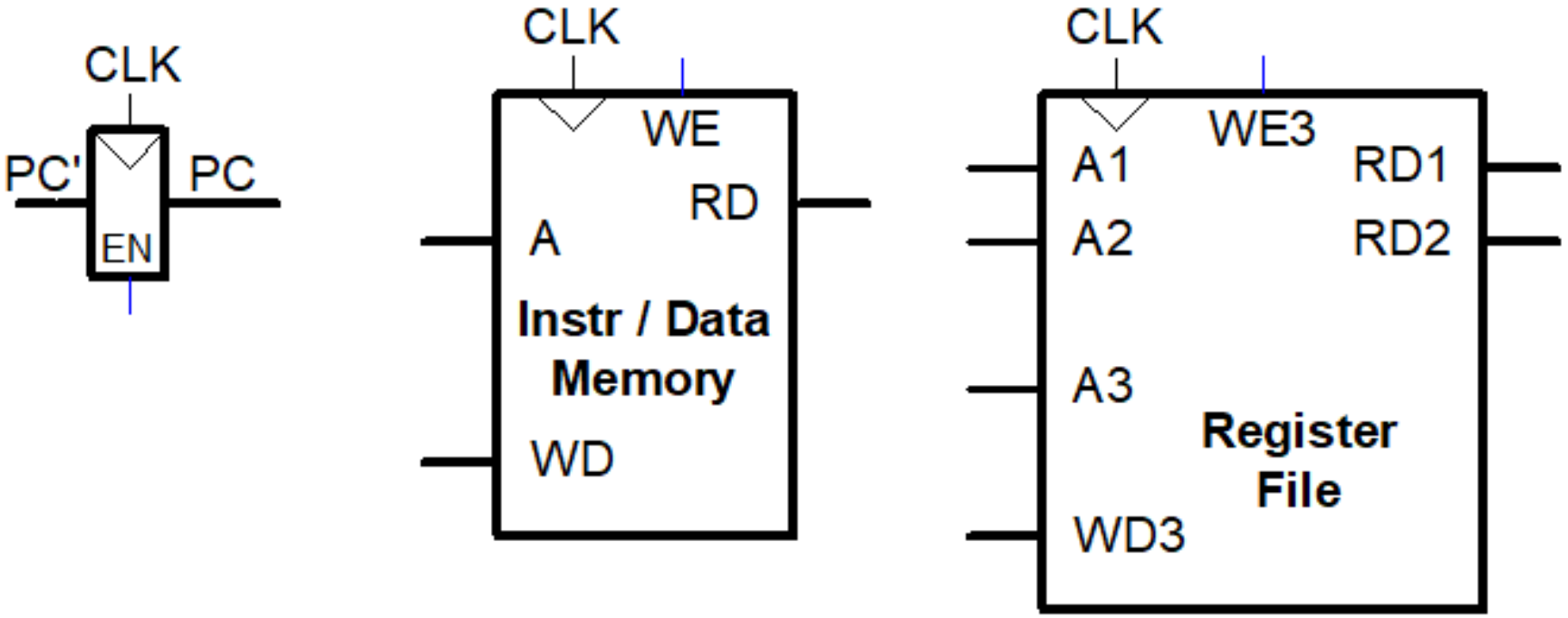
$$t = N \times CPI \times T_c = (100\times10^9)(1)(925\times10^{-12})$$
$$= \mathbf{92,5} \text{ сек}$$

Багатотактний MIPS процесор

- **Однотактний:**
  + Простий
  - Період тактової частоти обмежений інструкцією із самим довгим ланцюгом комбінаційної логіки (*lw*)
  - Декілька суматорів і дві окремі пам’яті
- **Багатотактний:**
  + Вища тактова частота
  + Прості інструкції виконуються швидше (за менше тактів)
  + Повторне використання апаратних ресурсів в різних тактах
  - Значно ускладнюється пристрій керування
- **Етапи розроблення: тракт даних і пристрій керування**

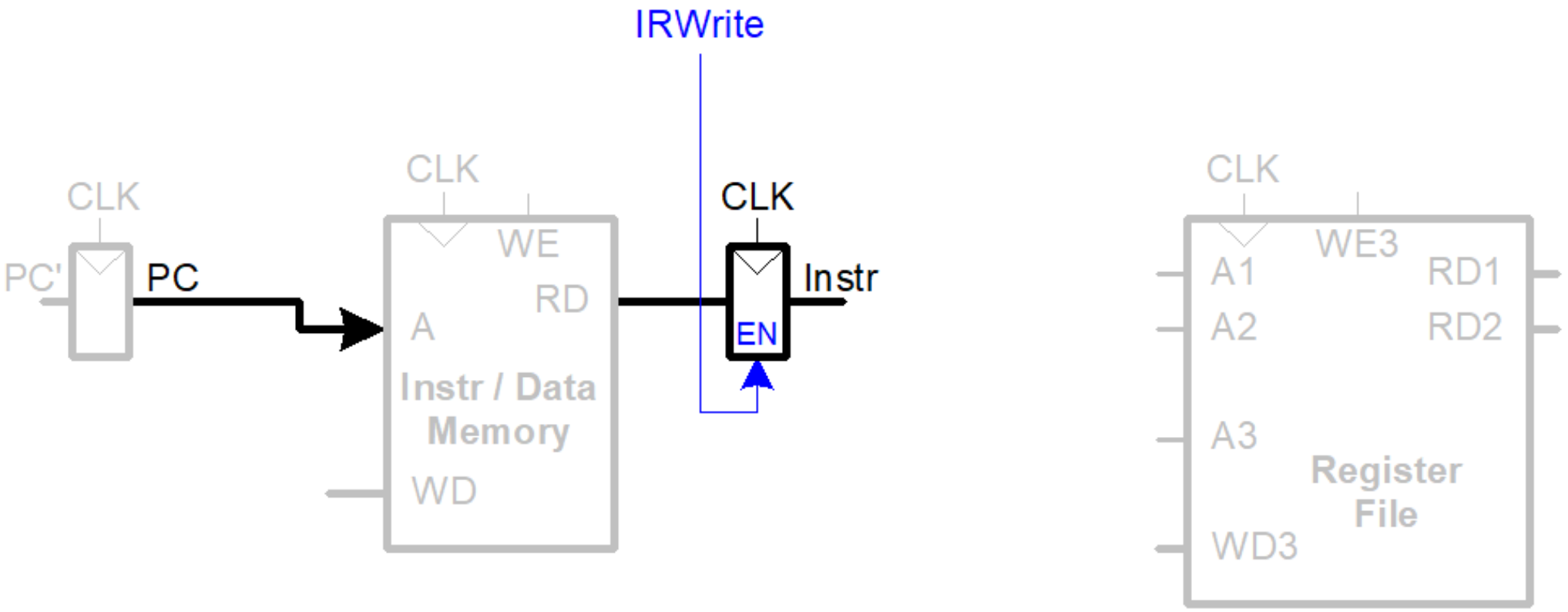
Елементи, зберігаючі стан багатотактного процесора

- Замість окремої пам’яті для інструкцій і даних буде використовуватися одна загальна пам’ять



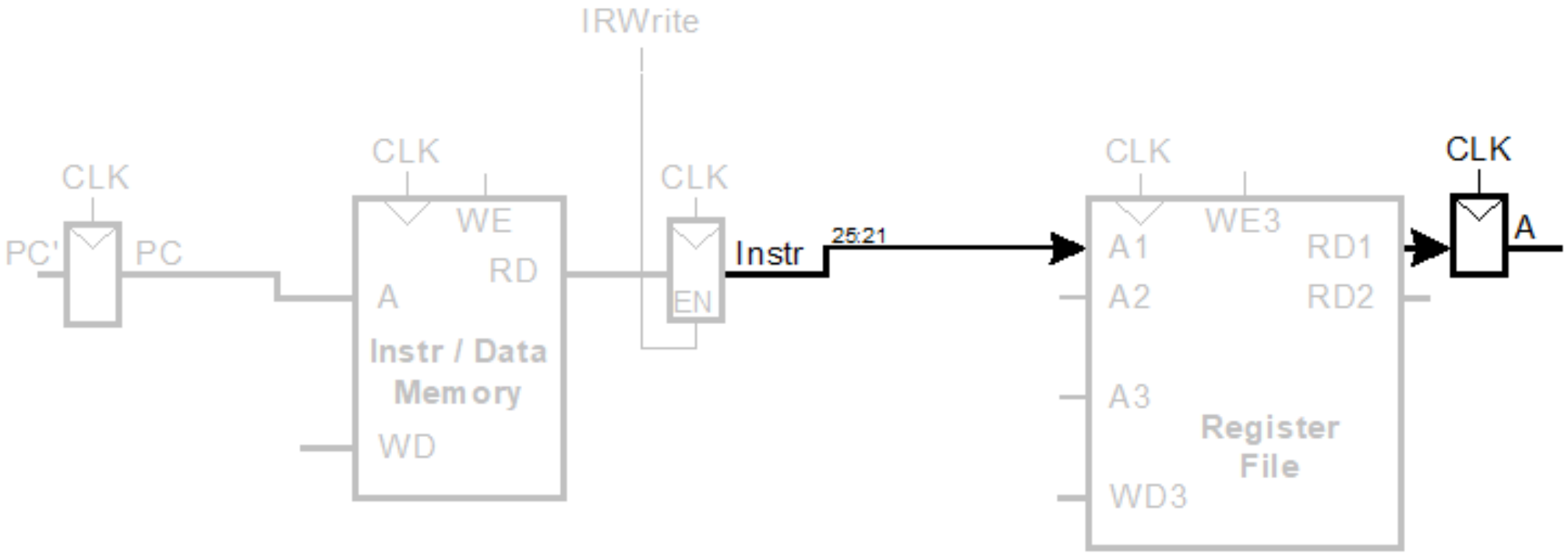
Багатотактний тракт даних: вибірка інструкції

Крок 1: Вибірка інструкції



Багатотактний тракт даних: читання регістрів

Крок 2а: зчитування операндів-джерел із регістрового файлу (на прикладі інструкції *lw*)



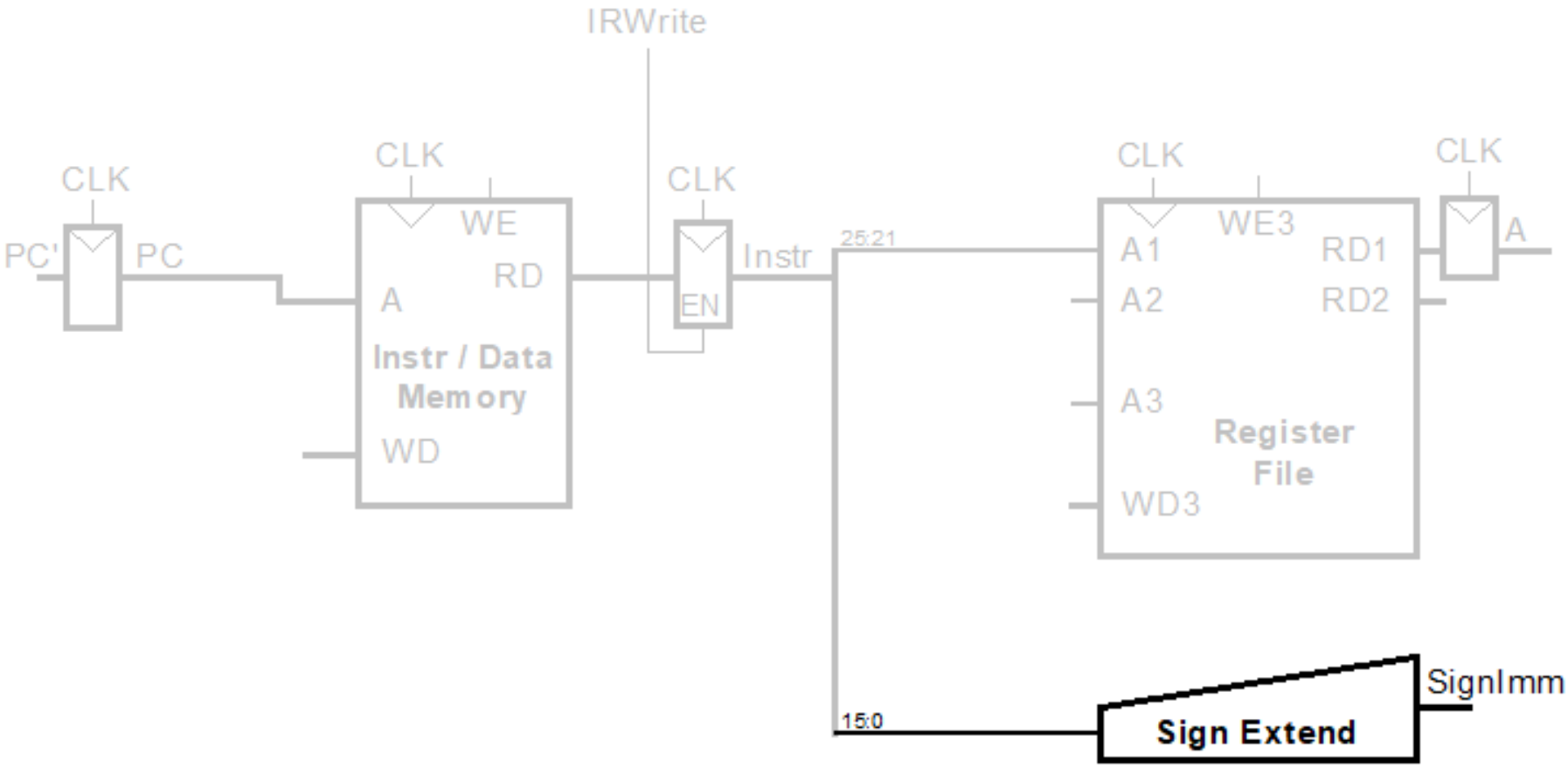

```
lw rt,imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

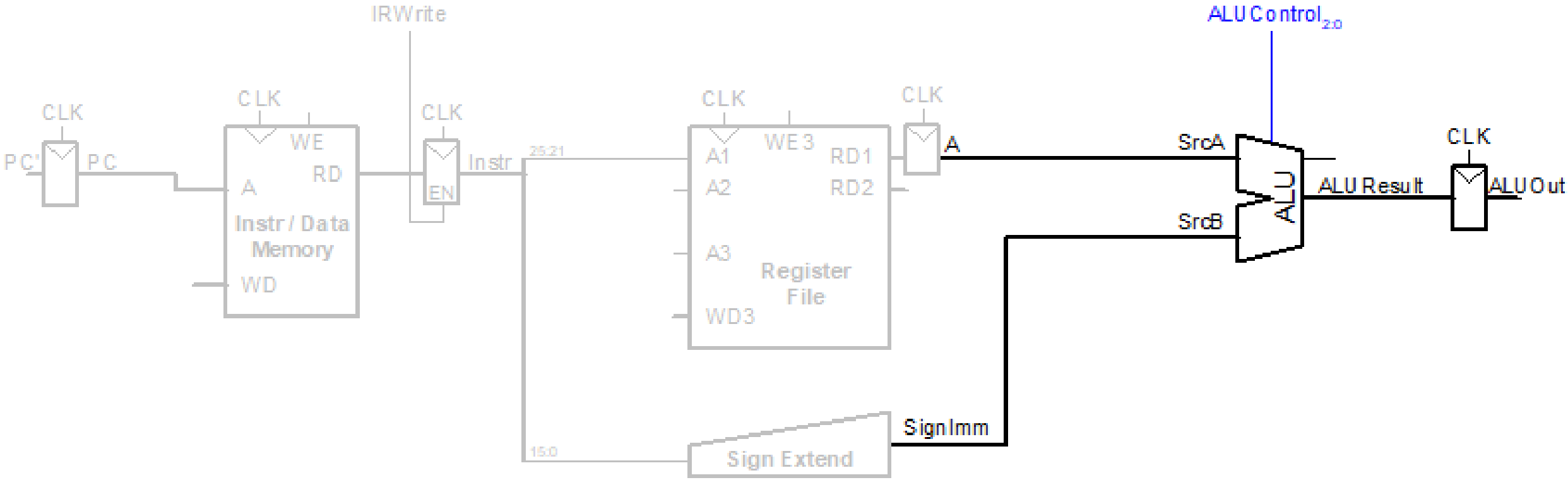
Багатотактний тракт даних: розширення константи

Крок 2b: розширення 16-бітової константи до 32-х розрядів бітом знаку



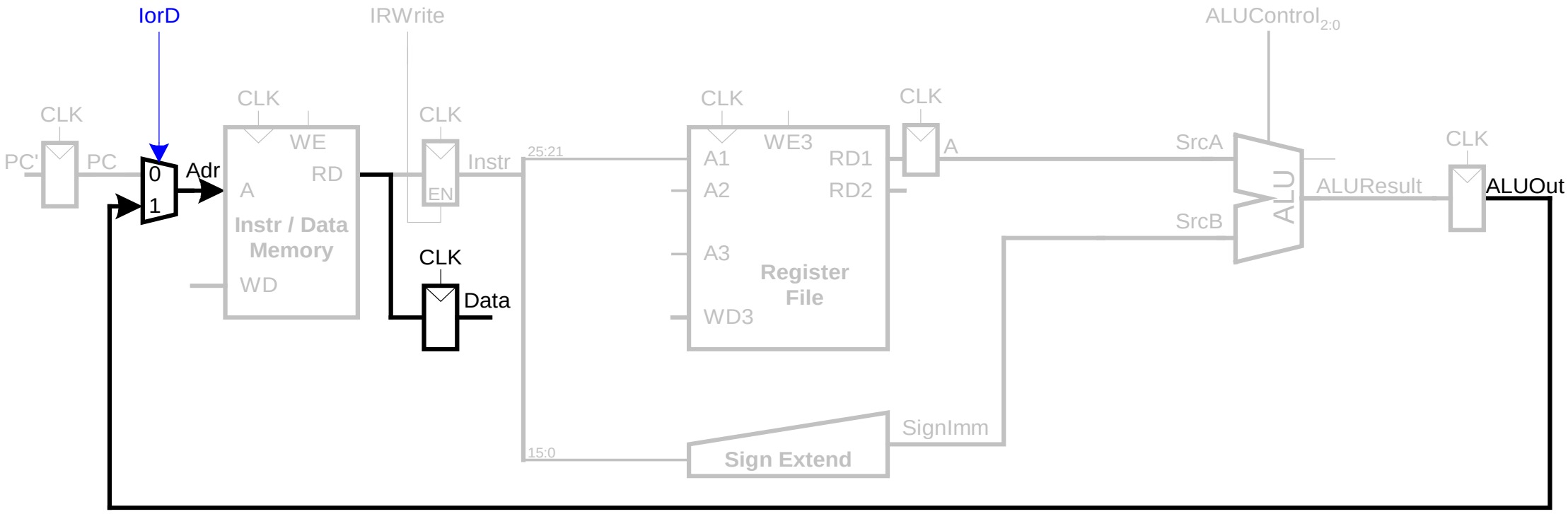
Багатотактний тракт даних: обчислення адреси

Крок 3: Обчислення адреси комірки в пам’яті



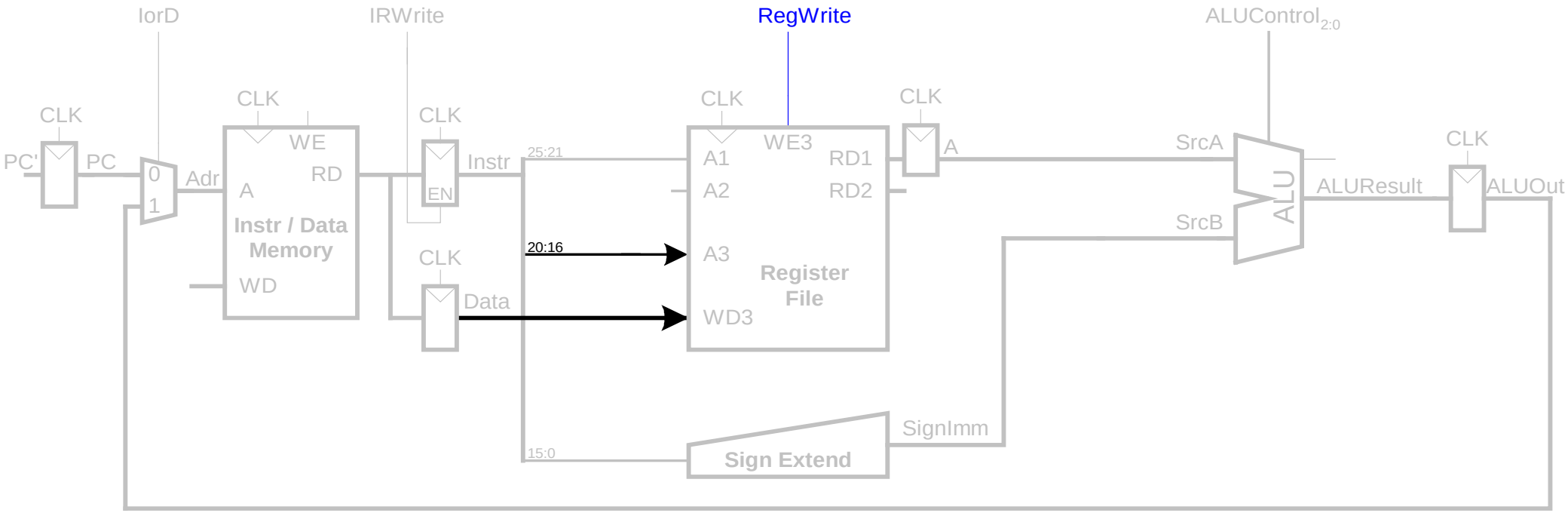
Багатотактний тракт даних: зчитування з пам’яті

Крок 4: зчитування даних з пам’яті



Багатотактний тракт даних: записування у регістр

Крок 5: записування зчитаного з пам’яті 32-бітового числа в регістр загального призначення, номер якого зберігається в полі *rt* інструкції



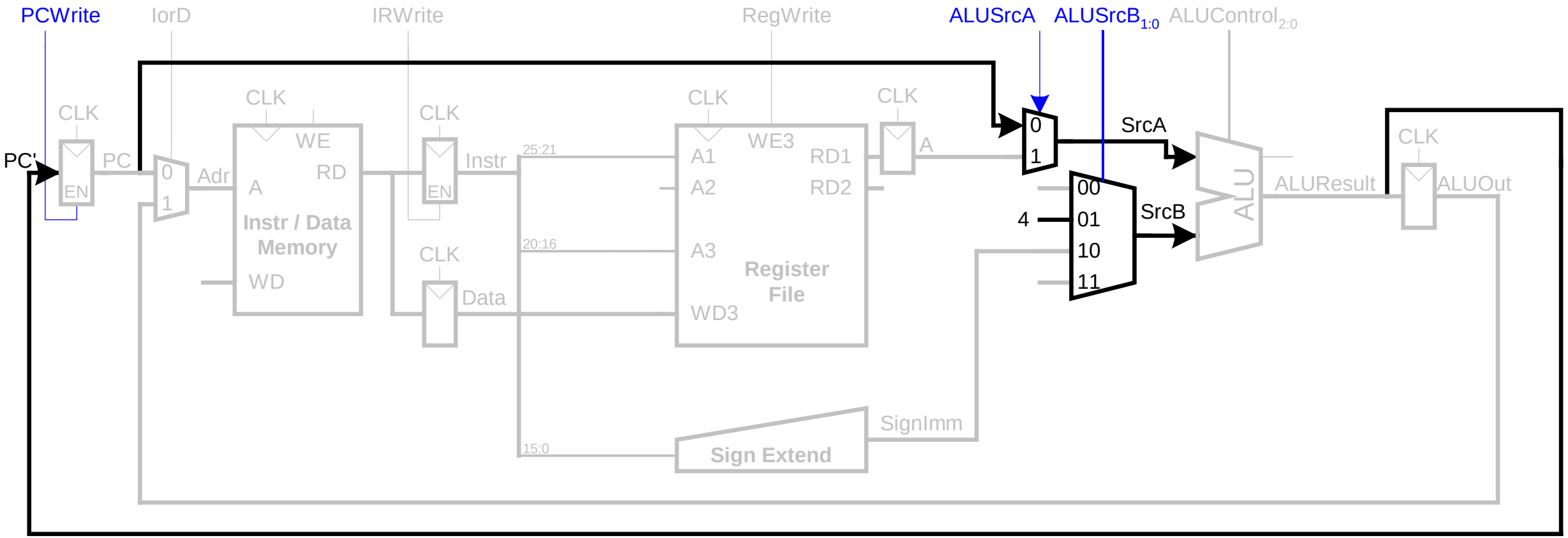
I-Type

`lw rt,imm(rs)`

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

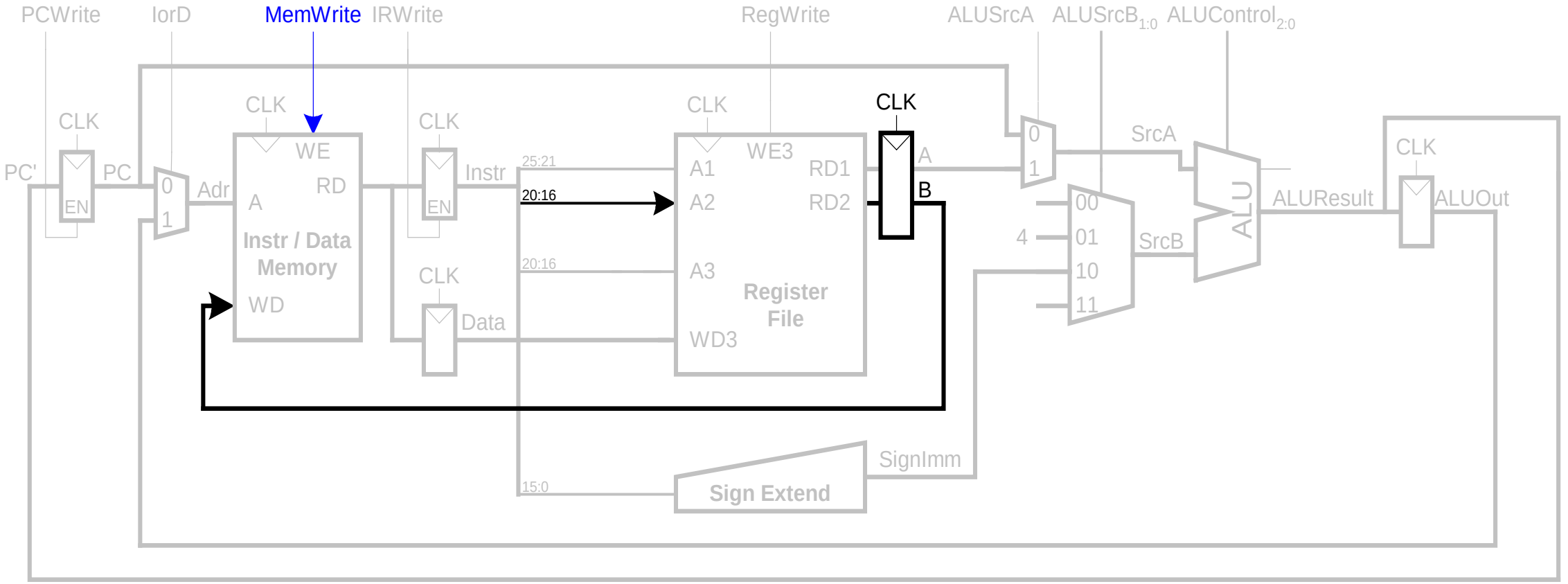
Багатотактний тракт даних: збільшення програмного лічильника PC

Крок 6: обчислення адреси наступної інструкції і записування її в PC



Багатотактний тракт даних: інструкція *sw*

Записування вмісту регістра *rt* в пам'ять



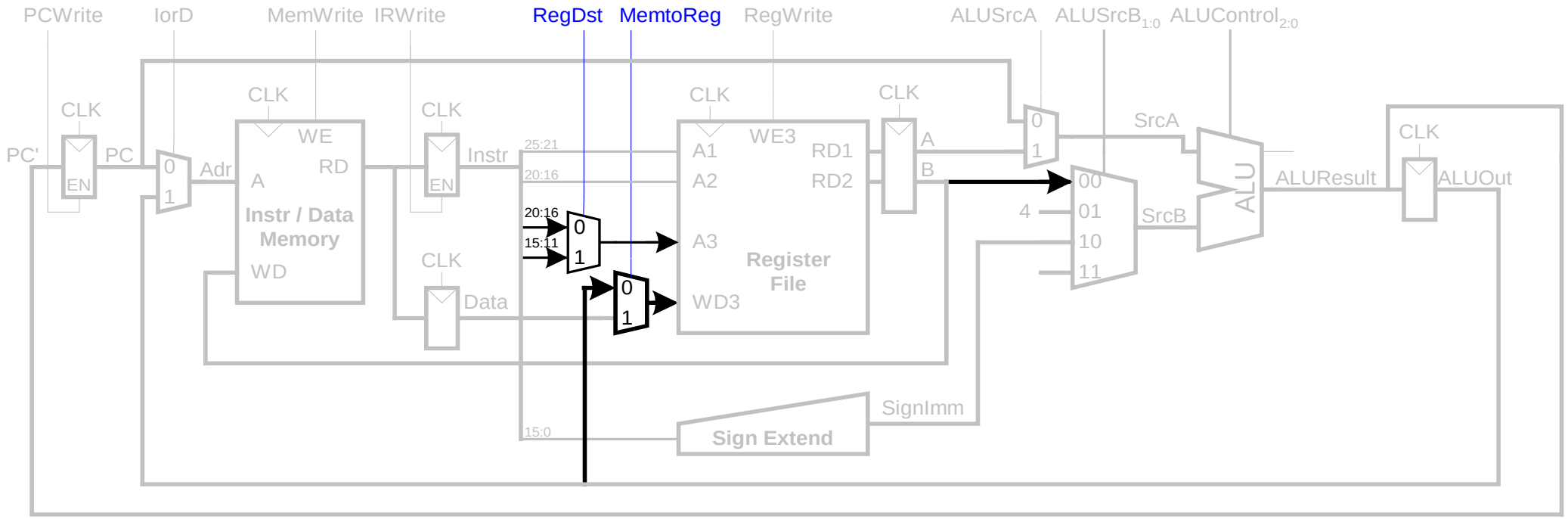
`sw rt,imm(rs)`

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Багатотактний тракт даних: інструкції R-типу

- Зчитування операндів з регістрів *rs* і *rt*
- Записування *ALUResult* в регістр з номером із поля *rd* інструкції (для інструкцій I-типу результат записується в регістр з номером *rt*)

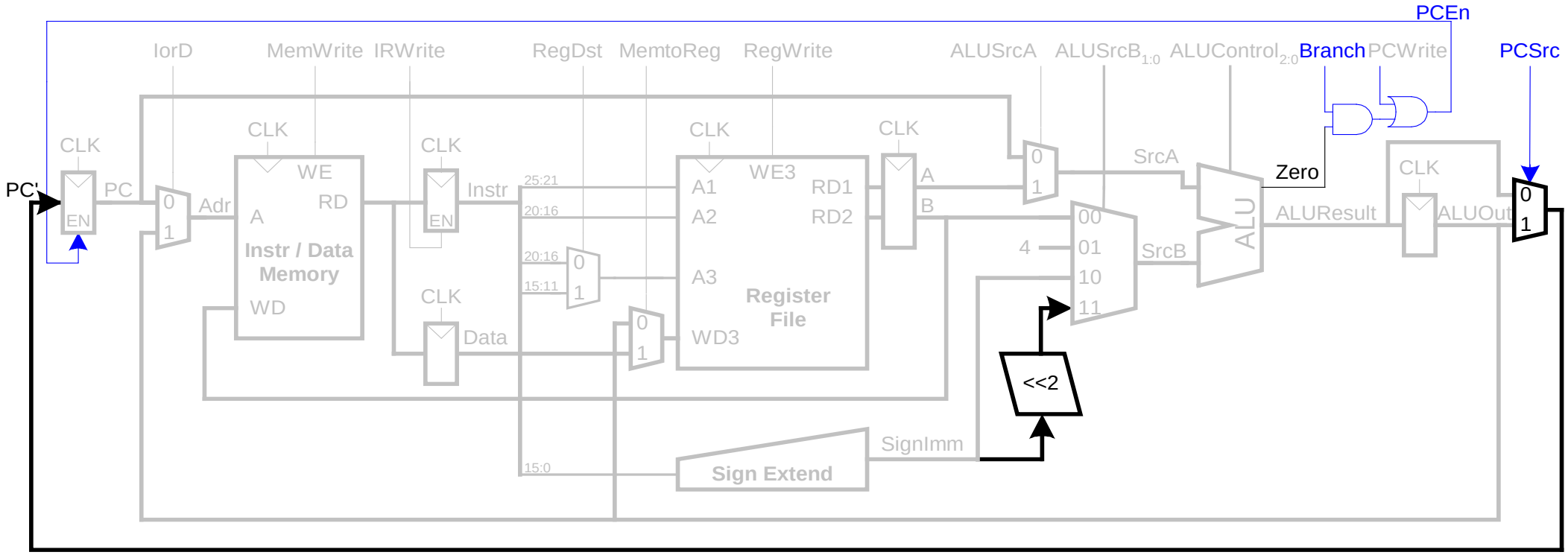


R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Багатотактний тракт даних: інструкція *beq*

- *rs == rt?*
- BTA = (sign-extended immediate << 2) + (PC+4)



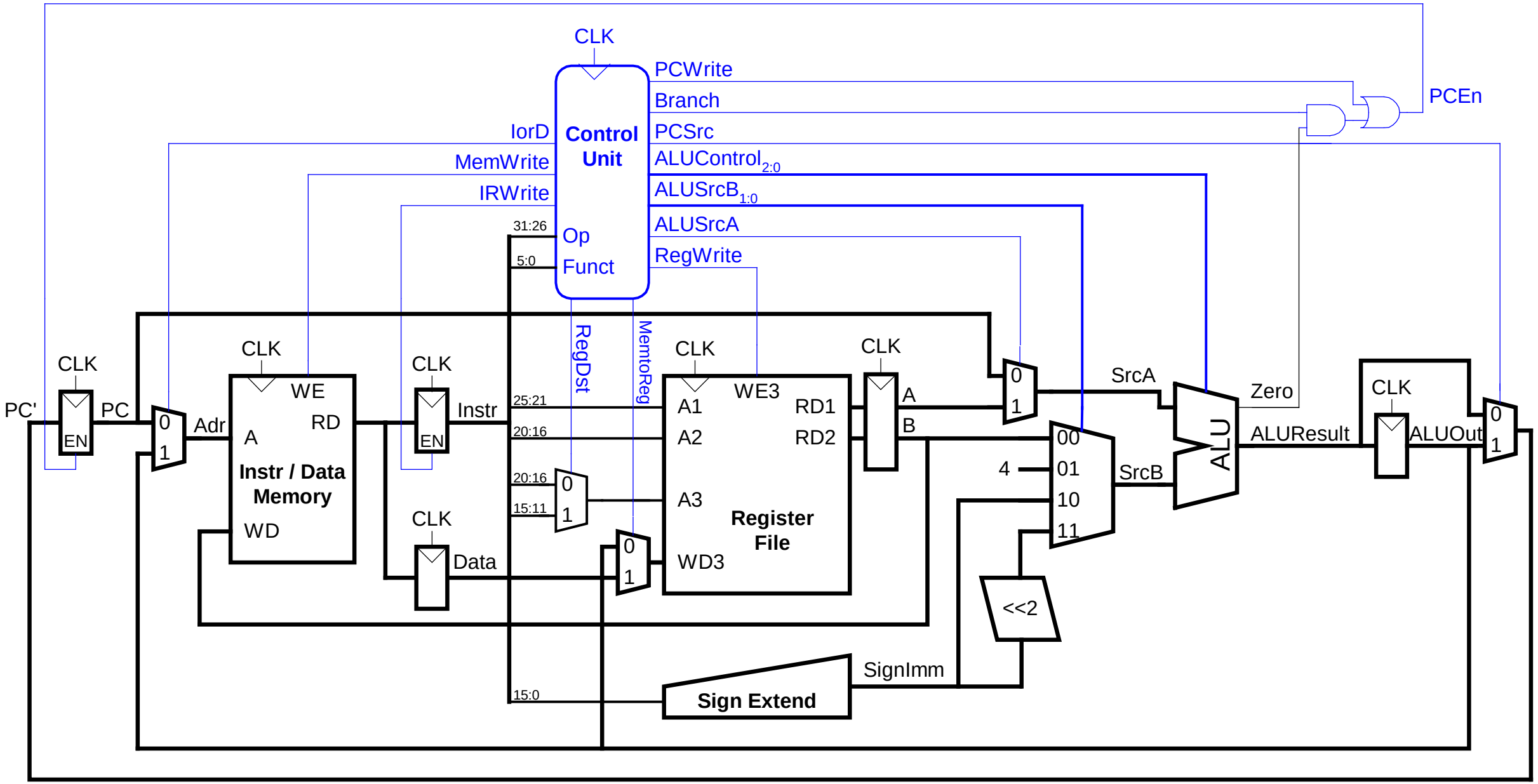
Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

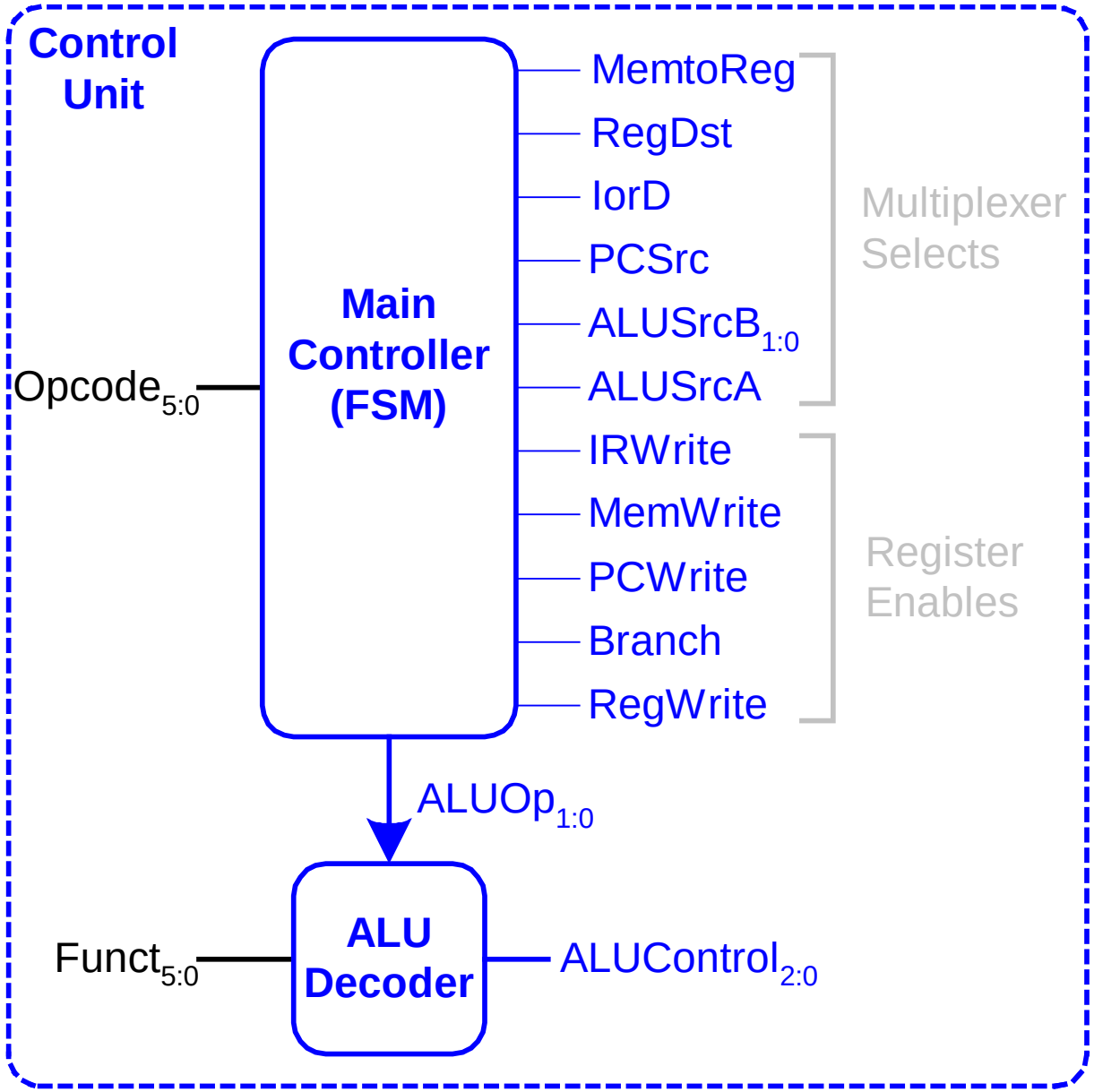
Field Values

op	rs	rt	imm		
4	8	0	3		
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Багатотактний процесор

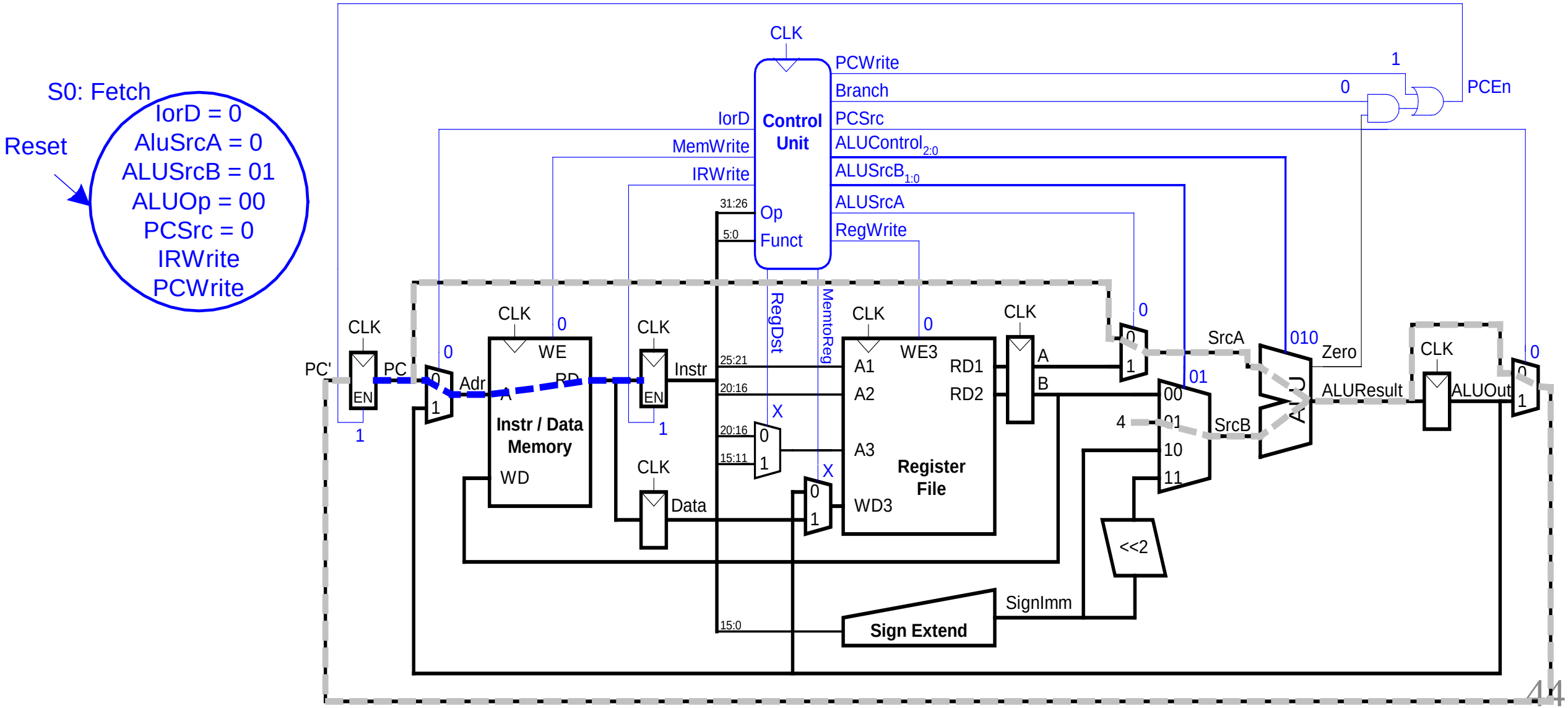


Багатотактний пристрій керування

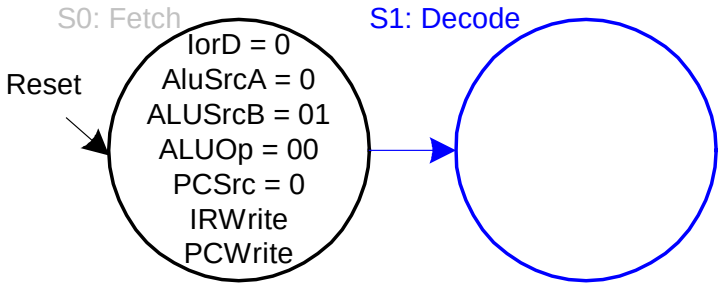


Основний керуючий автомат: вибірка

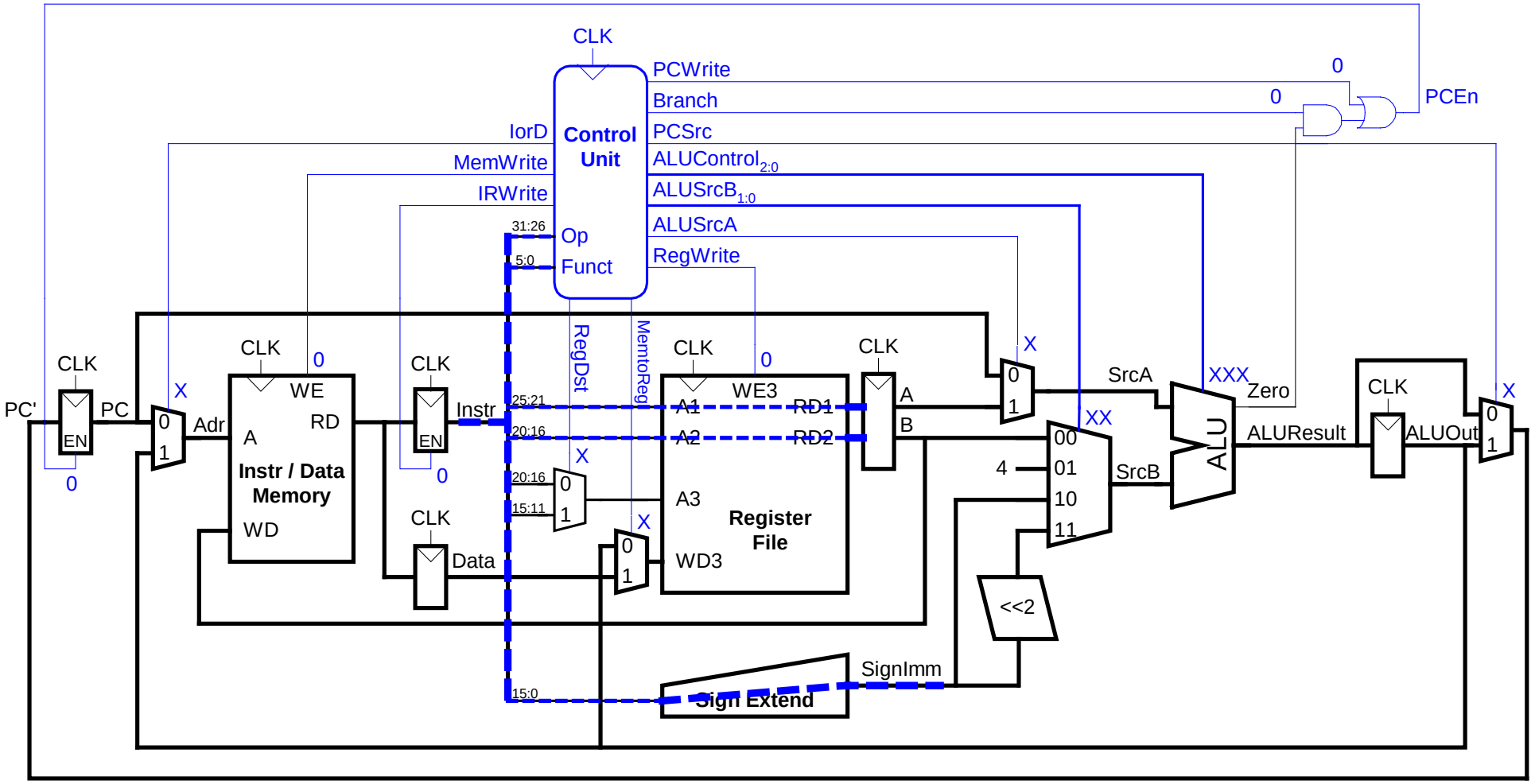
- Сигнали дозволу записування будуть показані тільки якщо вони не дорівнюють нулю
- Одночасно із зчитуванням інструкції за допомогою АЛП збільщується на 4 вміст PC



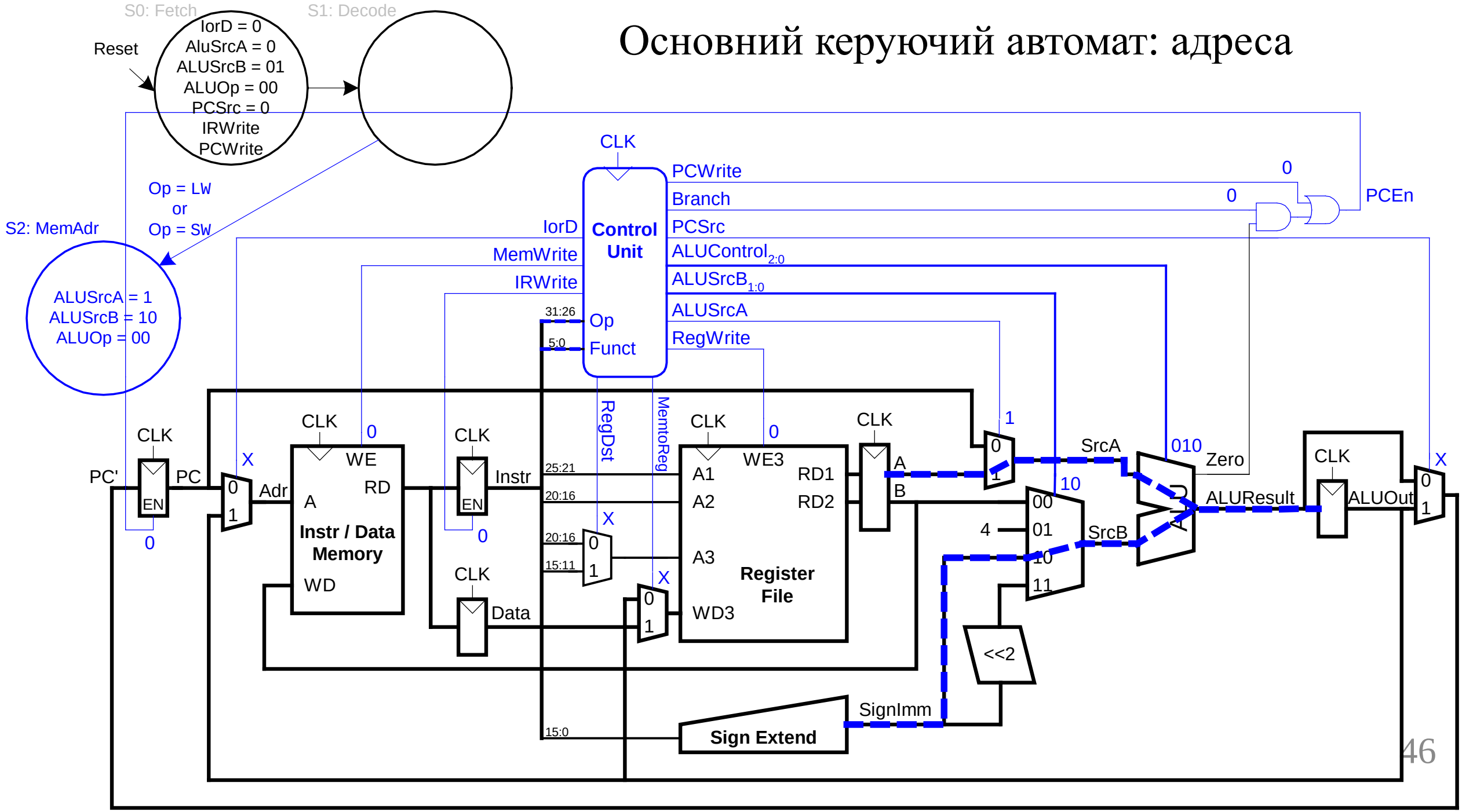
Основний керуючий автомат: декодування



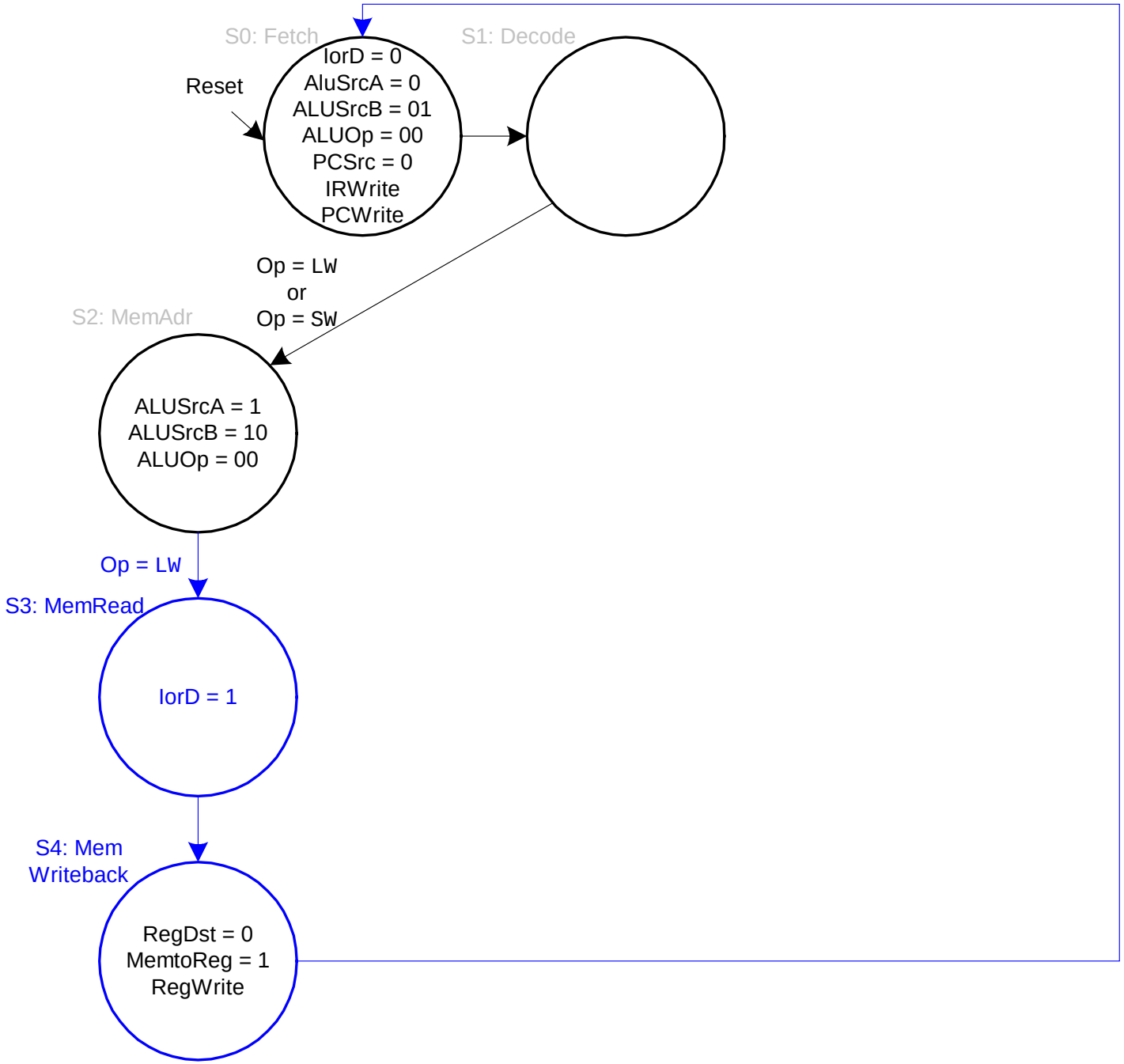
- Вказуються тільки ті керуючі сигнали, які мають зміст на конкретному етапі виконання команди
- На цьому етапі відбувається зчитування з регістрового файлу, розширення константи і декодування операції



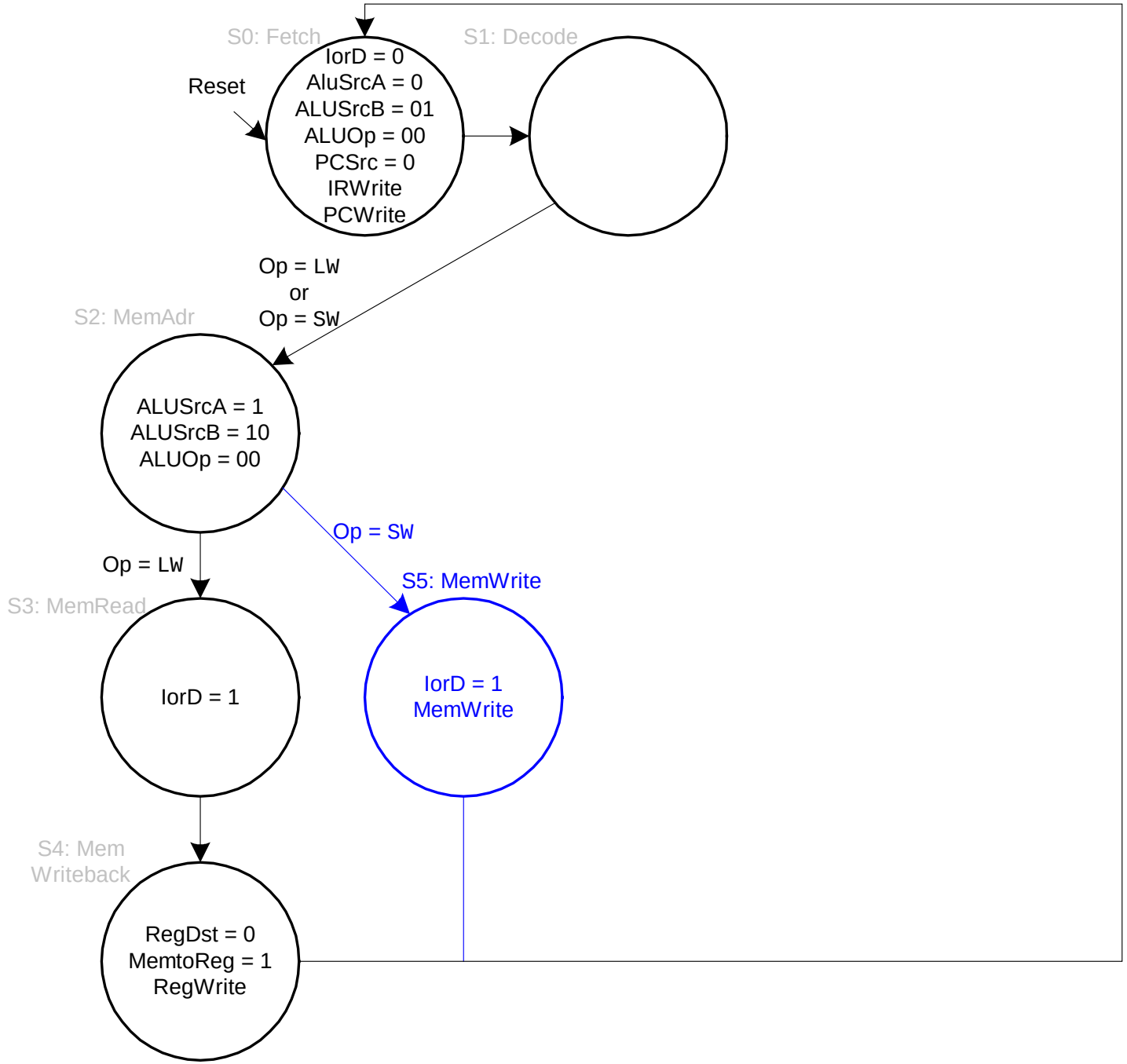
Основний керуючий автомат: адреса



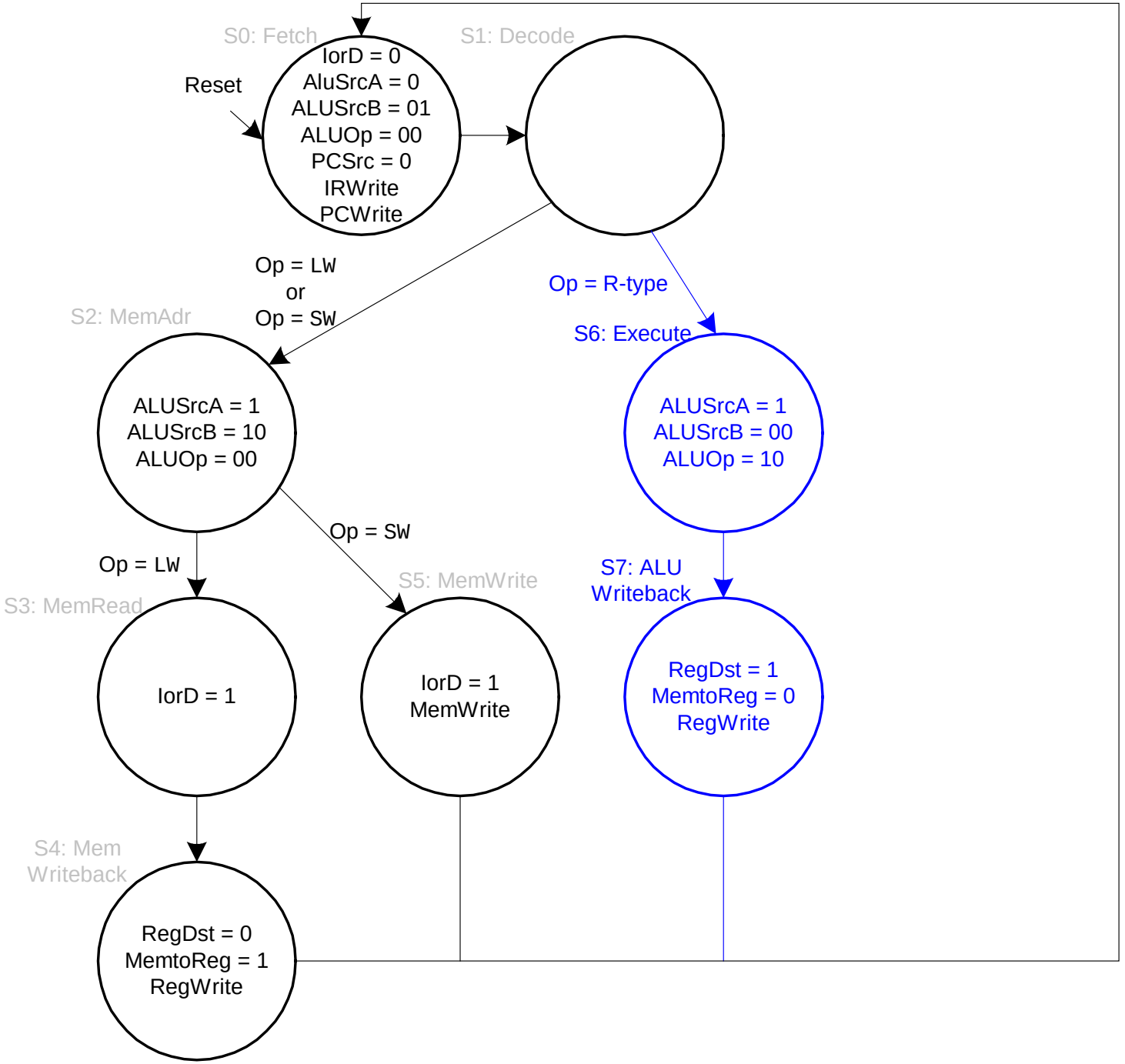
Основний керуючий автомат: *lw*



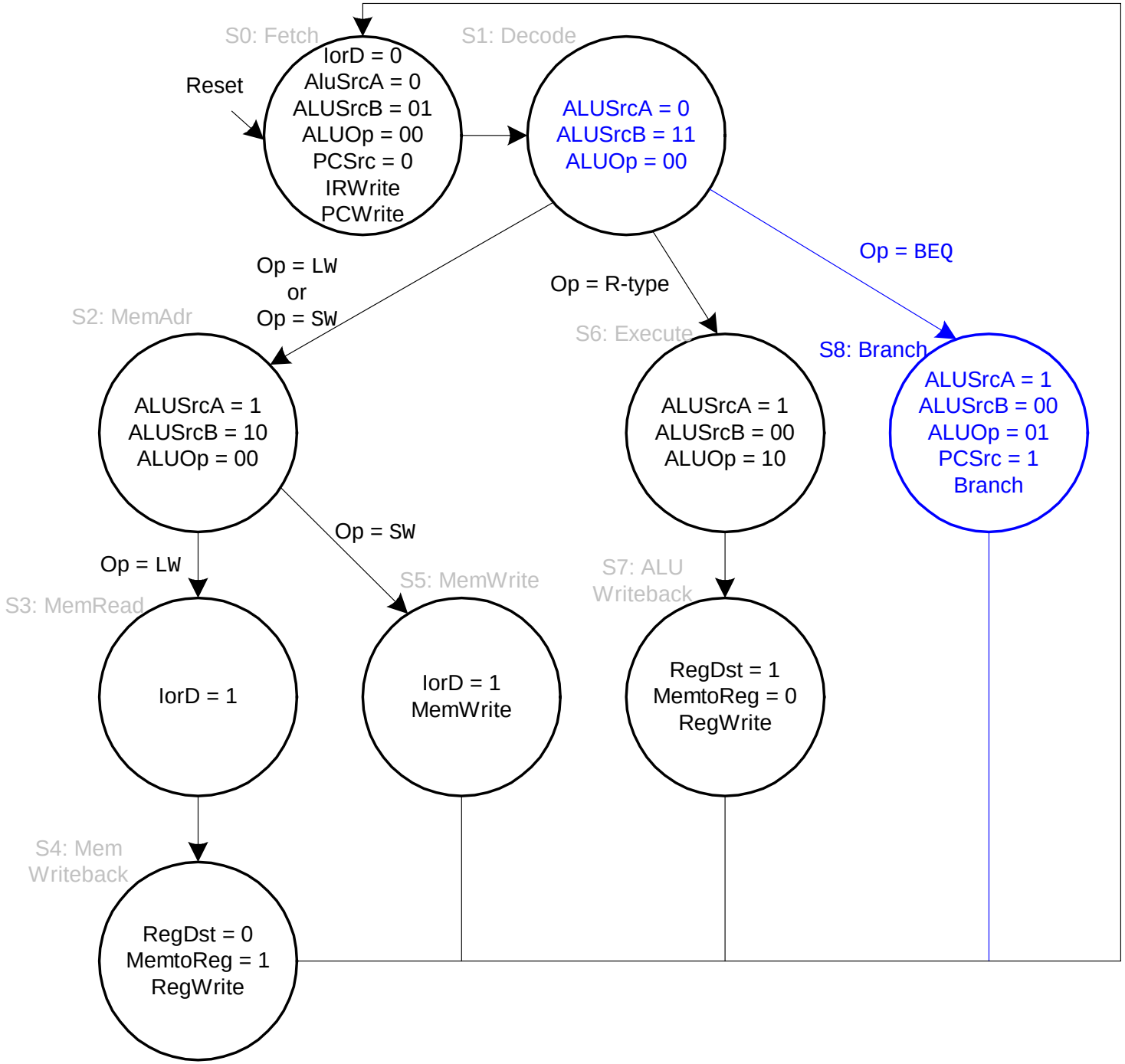
Основний керуючий автомат: *sw*



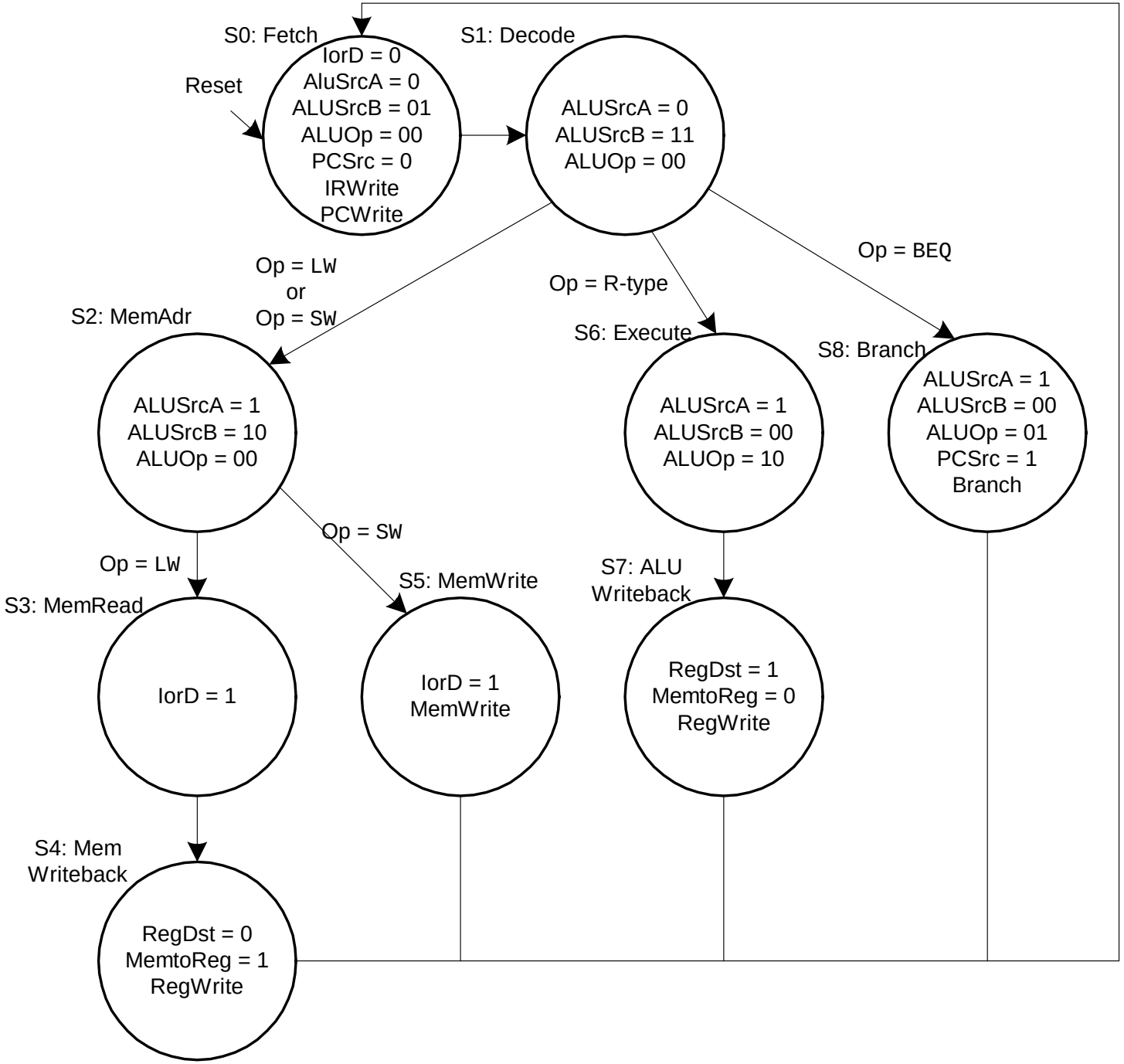
Основний керуючий автомат: R-тип



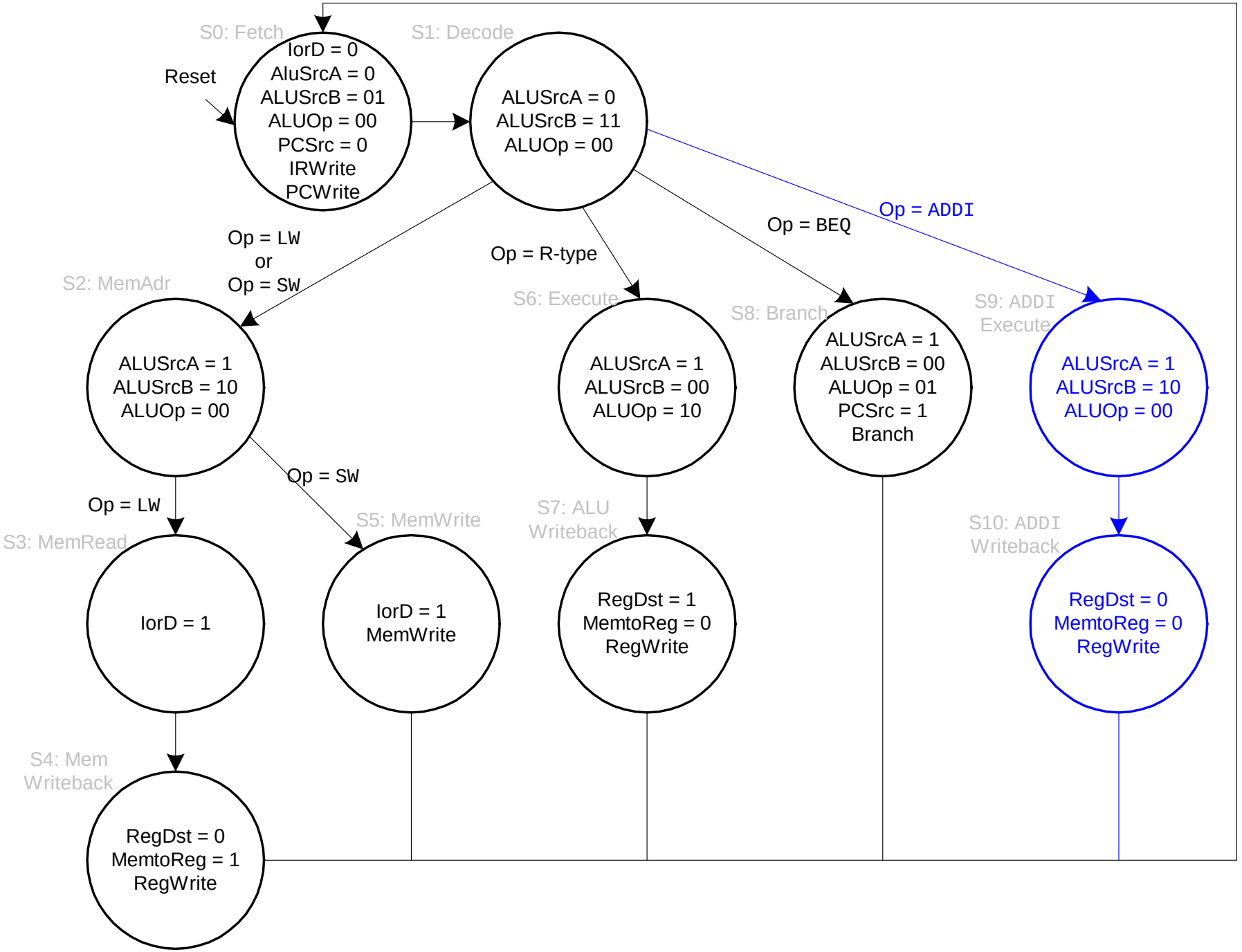
Основний керуючий автомат: *beq*



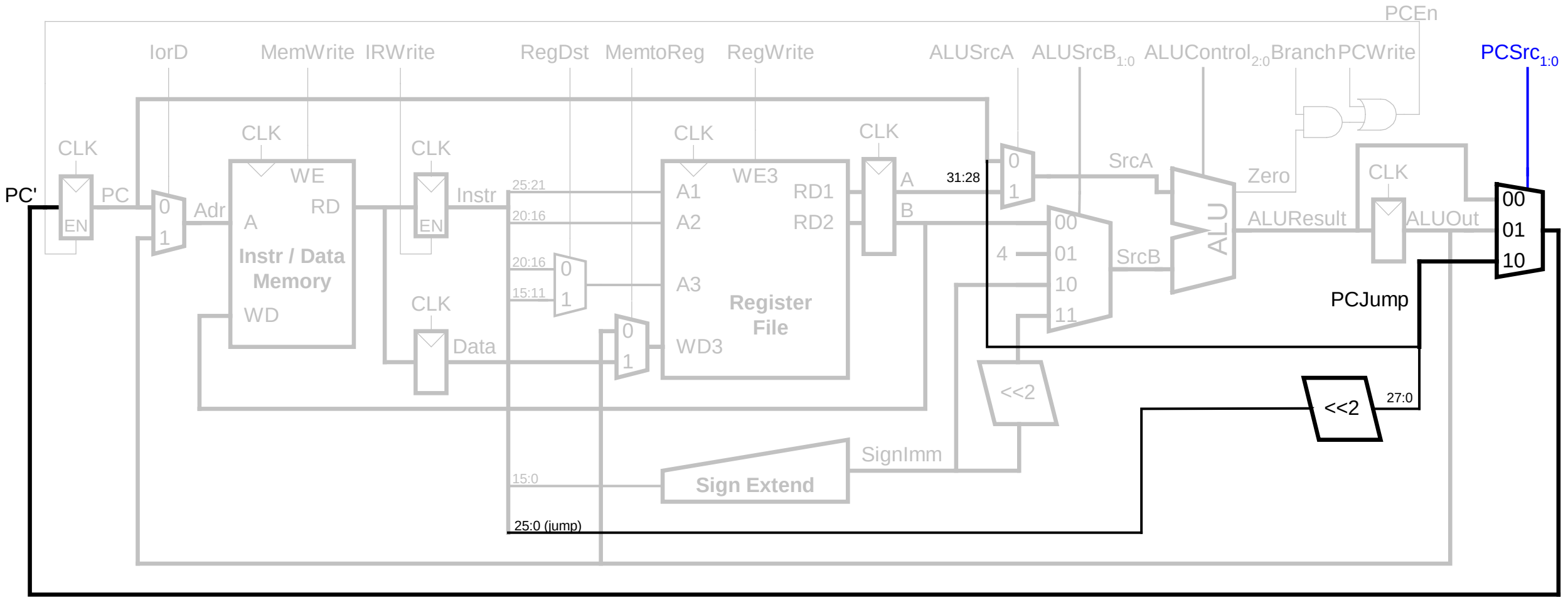
Основний керуючий автомат



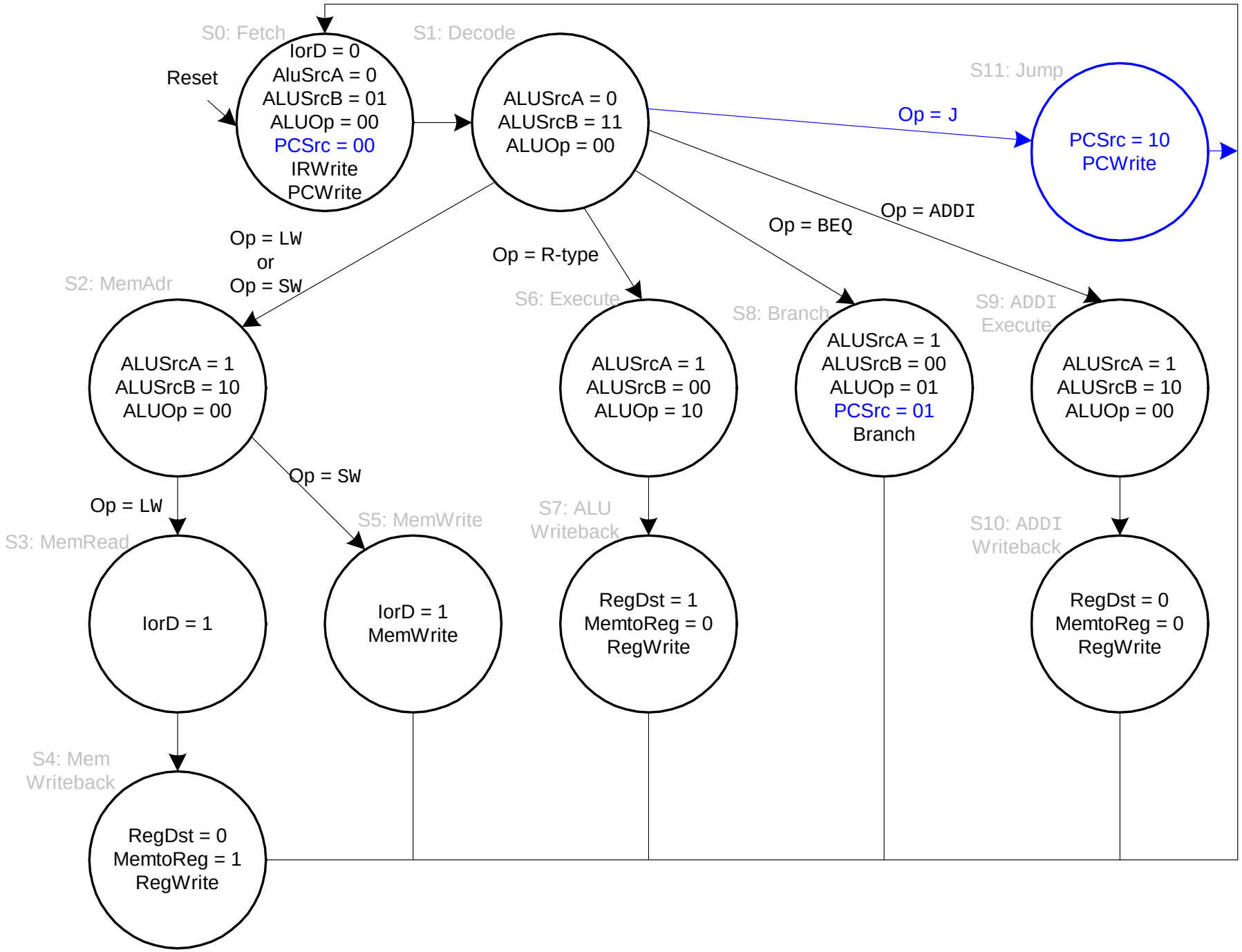
Основний керуючий автомат: добавлення інструкції *addi*



Добавлення функціоналу *j*



Основний керуючий автомат: *j*



Продуктивність багатотактного процесора

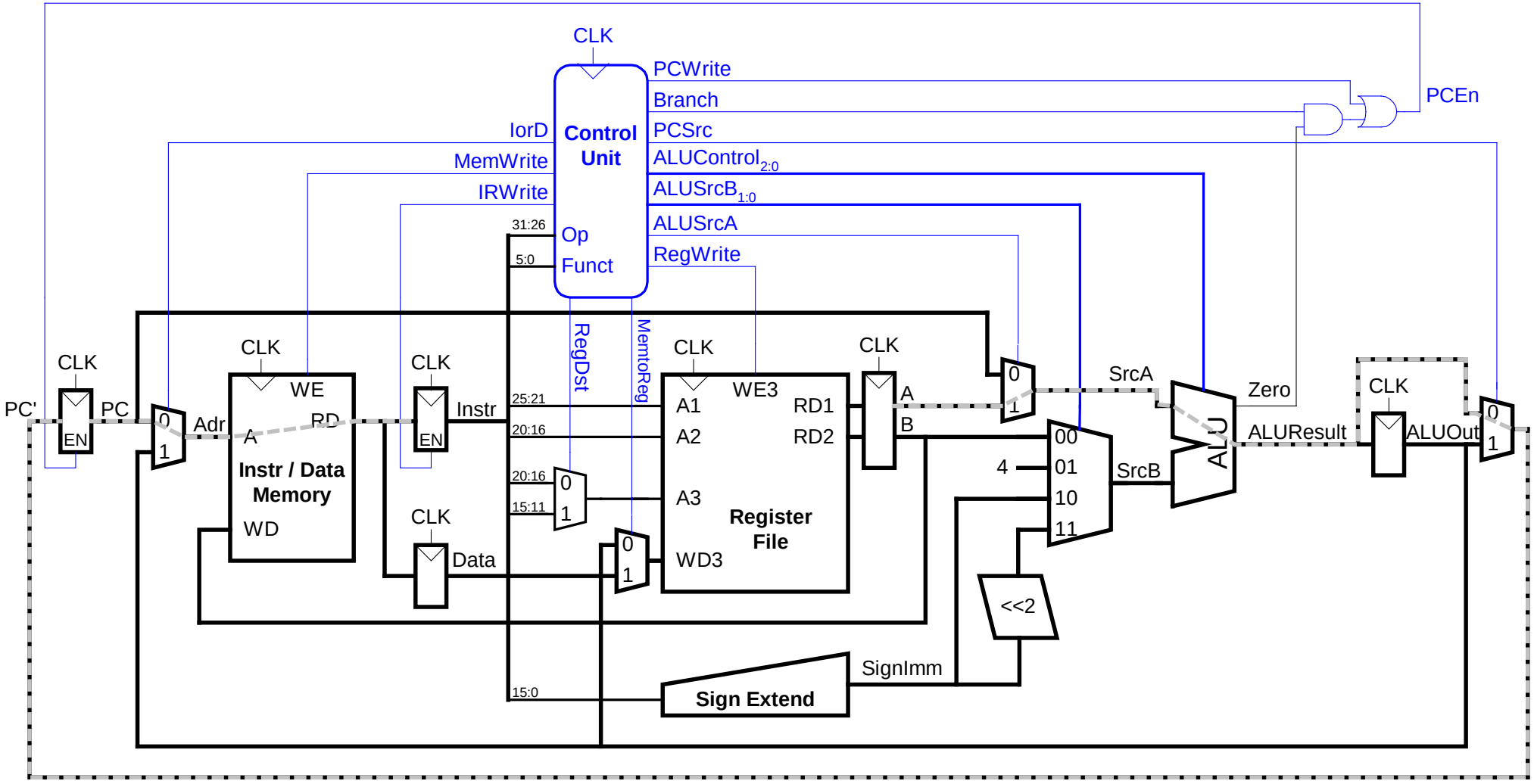
- Інструкції виконуються за різну кількість тактів:
  - 3 такти: beq, j
  - 4 такти: R-тип, sw, addi
  - 5 тактів: lw
- CPI буде середнім значенням
- Тестовий набір SPECINT2000 містить:
  - 25% інструкцій lw
  - 10% інструкцій sw
  - 11% умовних переходів
  - 2% безумовних переходів
  - 52% інструкці R-типу

Середній CPI = (0.11 + 0.2)(3) + (0.52 + 0.10)(4) + (0.25)(5) **= 4.12**

Продуктивність багатотактного процесора

Затримка самого довгого ланцюга комбінаційної логіки багатотактного процесора:

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$



Розрахунок продуктивності багатотактного процесора

Параметр	**Позначення**	**Затримка (пс)**
Час запису у регістр	t_{pcq_PC}	30
Час передвстановлення регістру	t_{setup}	20
Затримка мультиплексору	t_{mux}	25
Затримка АЛП	t_{ALU}	200
Затримка зчитування з пам’яті	t_{mem}	250
Затримка зчитування з регістрового файлу	t_{RFread}	150
Час передвстановлення регістрового файлу	$t_{RFsetup}$	20

CPI=4.12

$$\boldsymbol{T_c} = t_{pcq\_PC} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$
$$= t_{pcq\_PC} + t_{mux} + t_{mem} + t_{setup}$$
$$= [30 + 25 + 250 + 20] \text{ пс} = \mathbf{325 \text{ пс}}$$

Час виконання $N=100\times10^9$ інструкцій

$$t = N \times CPI \times \boldsymbol{T_c} = (100\times10^9)(4.12)(325\times10^{-12})$$
$$= \mathbf{133.9 \text{ сек}}$$

Це більше як для однотактного процесора (92.5 сек). Причини:

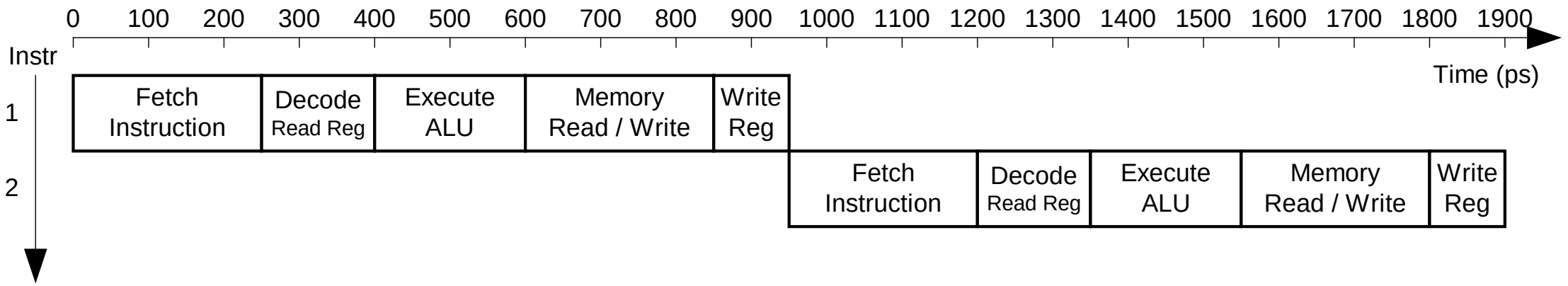
- у різних команд різна тривалість виконання;
- додаткові затримки на кожному кроці ($t_{pcq} + t_{setup}$= 50 пс)

Конвеєрний MIPS процесор

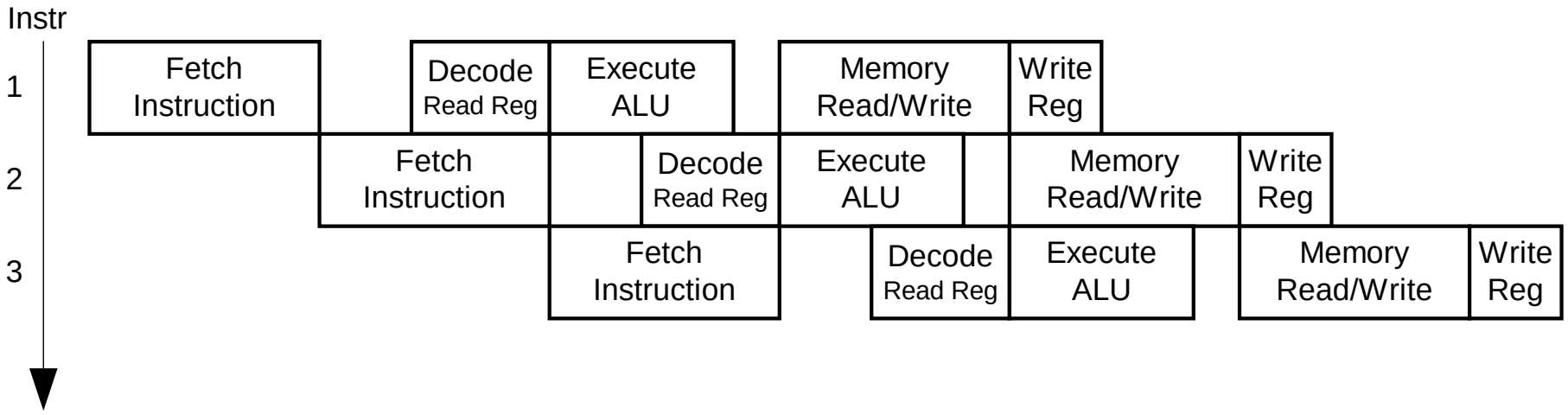
- Часовий паралелізм
- Давайте розділимо однотактний процесор на 5 стадій:
  - Вибірка
  - Декодування
  - Виконання
  - Доступ до пам’яті
  - Записування результатів
- Давайте добавимо регістри між стадіями конвеєра

Однотактний/конвеєрний процесор

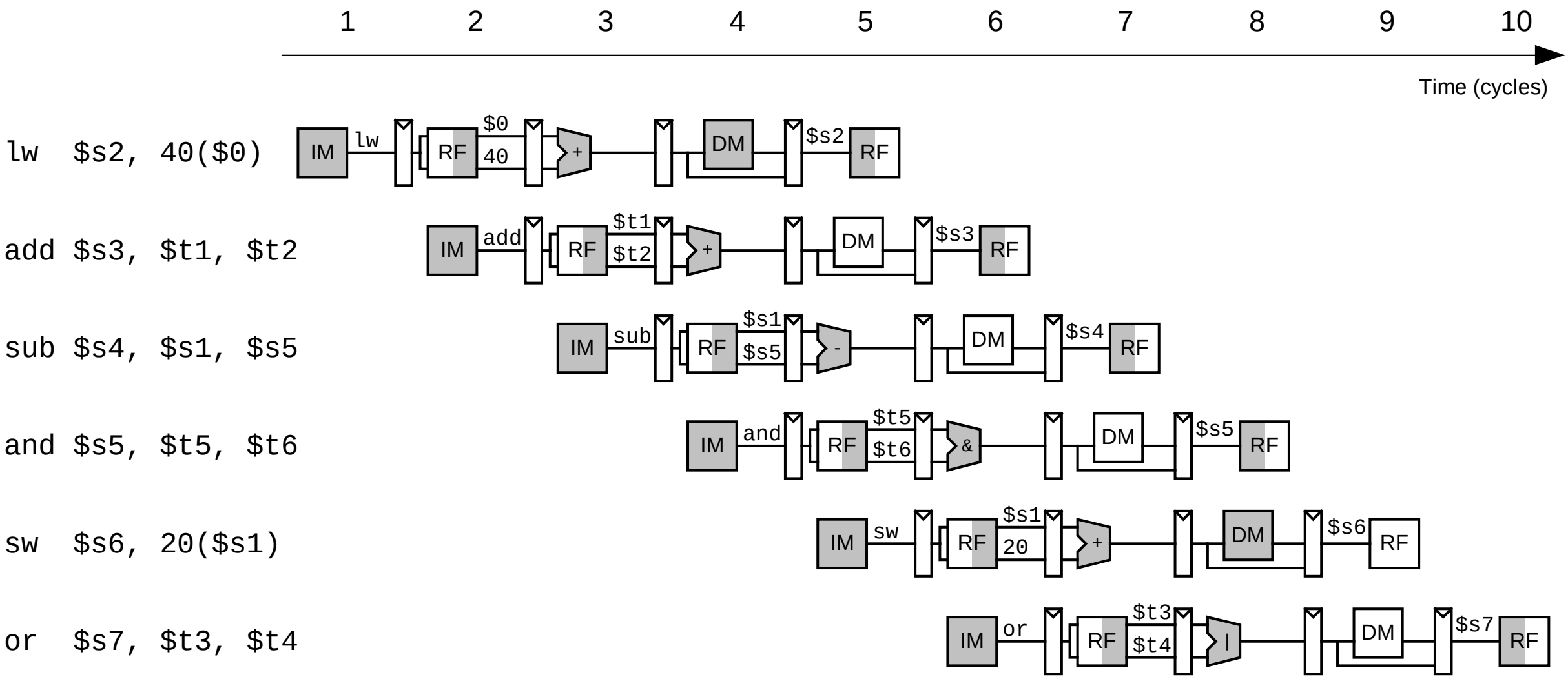
Single-Cycle



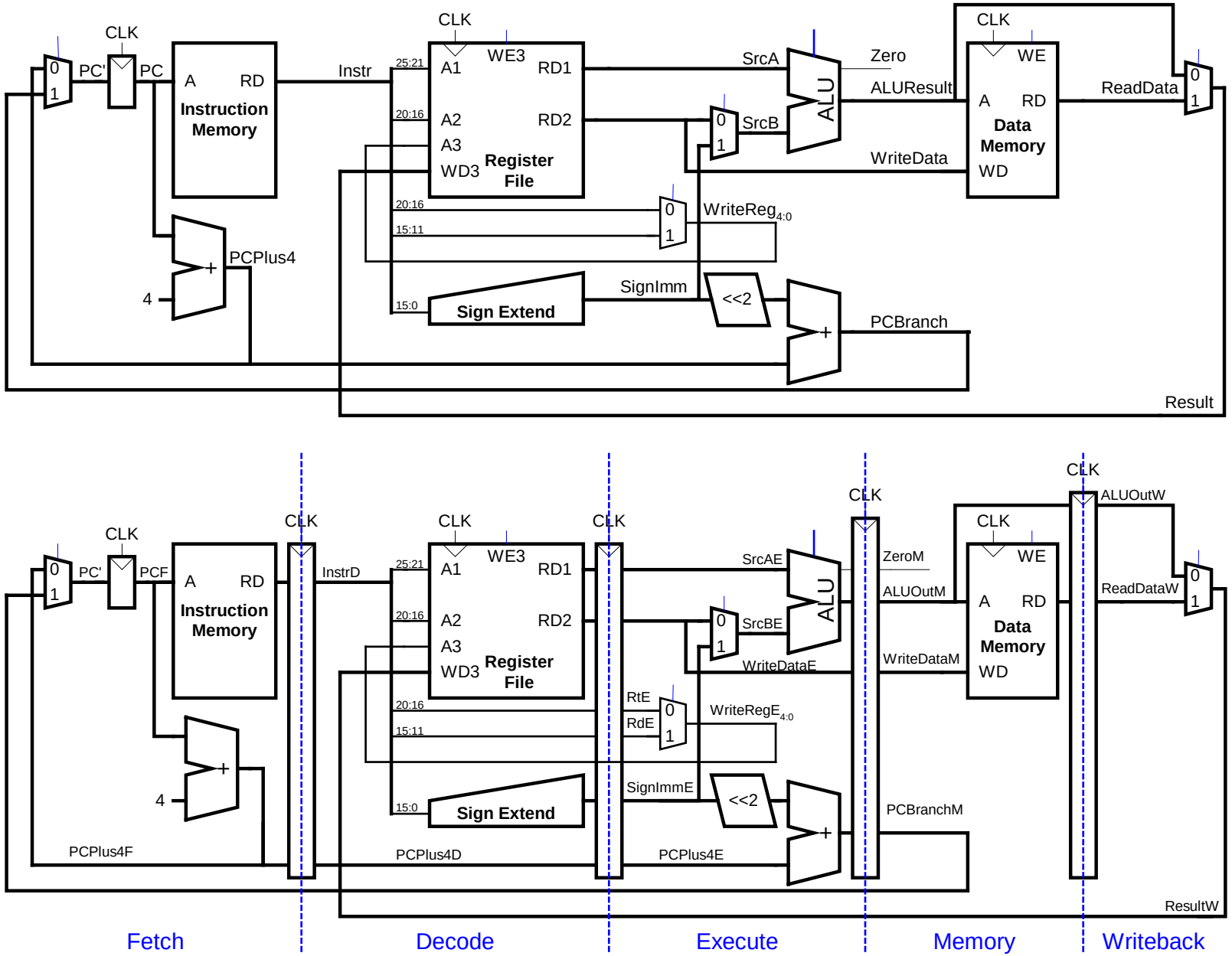
Pipelined



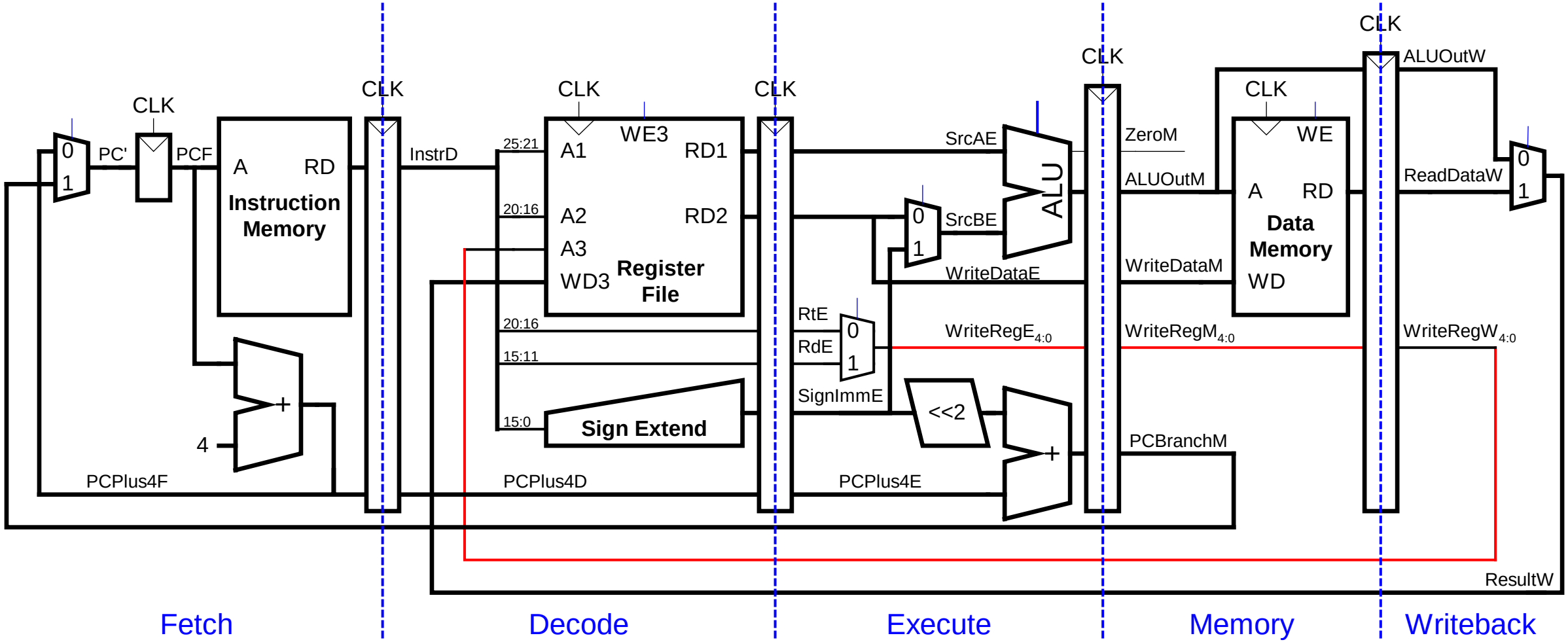
Абстрактне подання конвеєра



Однотактний і конвеєрний тракт даних

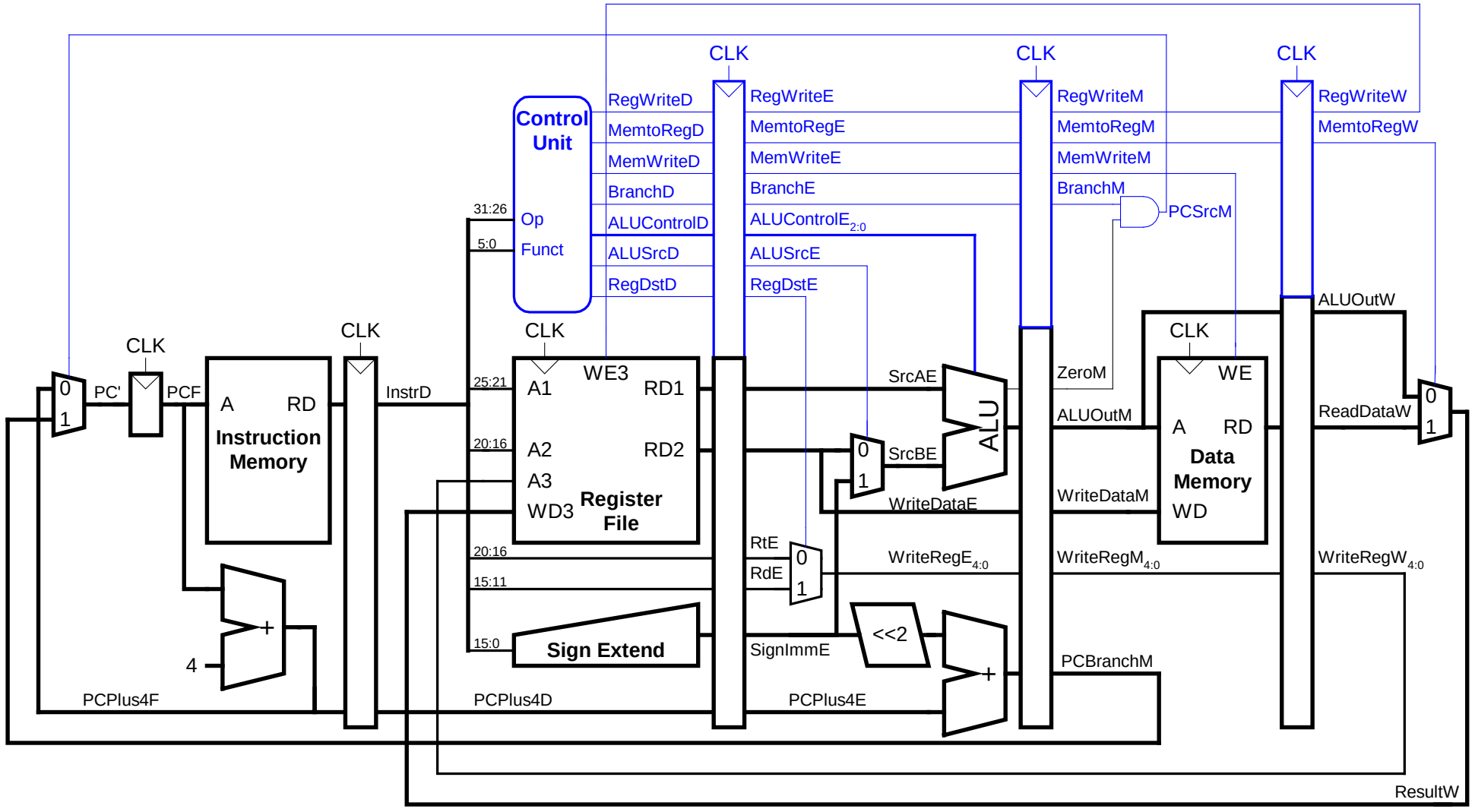


Виправлений конвеєрний тракт даних



Тепер *WriteRegW* і *ResultW* подаються на входи регістрового файлу в стадії Writeback одночасно.

Керування конвеєрним процесором

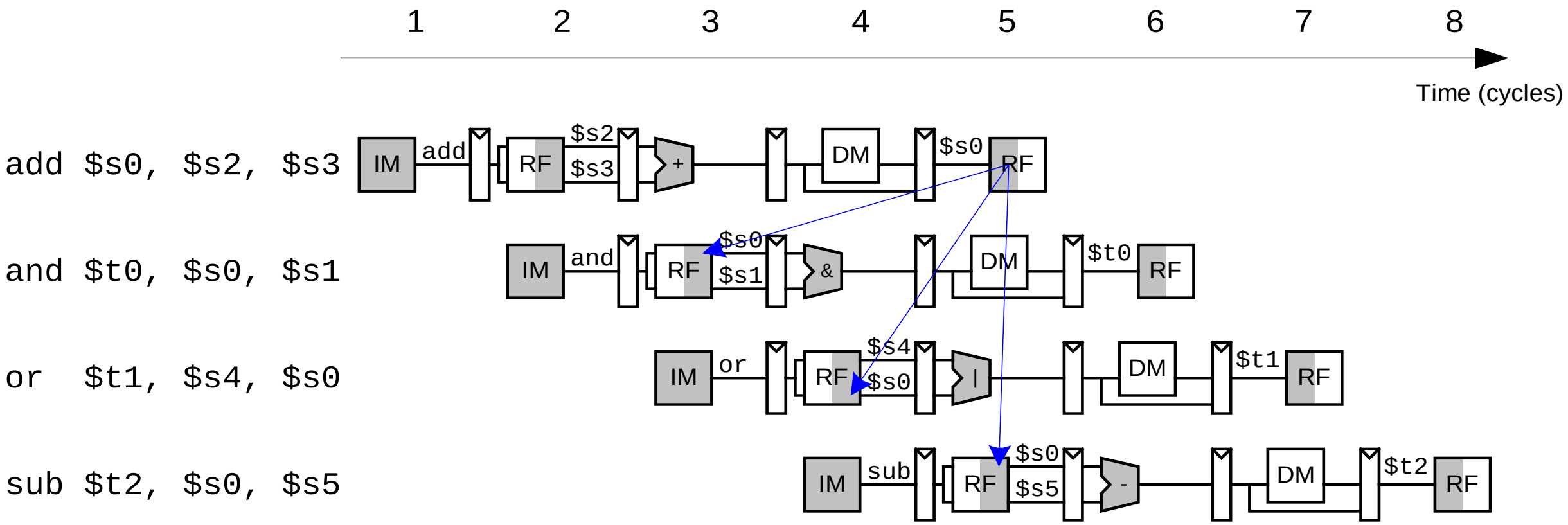


- Той же пристрій керування, що і в однотактному процесорі
- Сигнали керування доходять до відповідної стадії із затримкою (сигнали керування також конвеєризуються)

Конфлікти конвеєра

- В конвеєрі виконується декілька інструкцій одночасно
- Конфлікти виникають коли одна інструкція залежить від результату іншої, ще не завершеної інструкції
- Типи конфліктів:
  - **Конфлікти даних:** результат інструкції ще не записаний в регістр, а наступна інструкція уже пробує зчитати цей регістр
  - **Конфлікти керування:** процесор вибирає з пам’яті наступну інструкцію до того, як стало ясно, яку саме інструкцію потрібно вибрати (виникають через умовні переходи)

Конфлікт даних

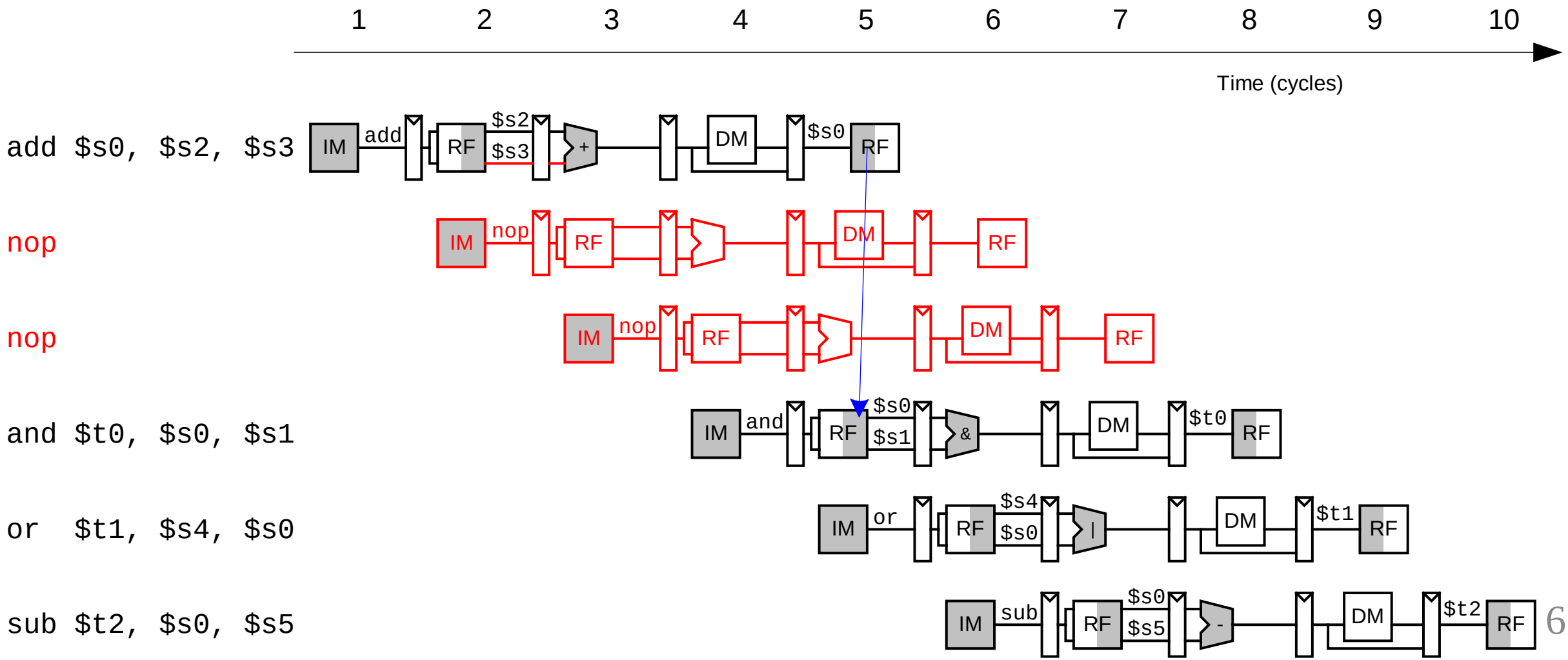


Розв’язування конфлікту даних

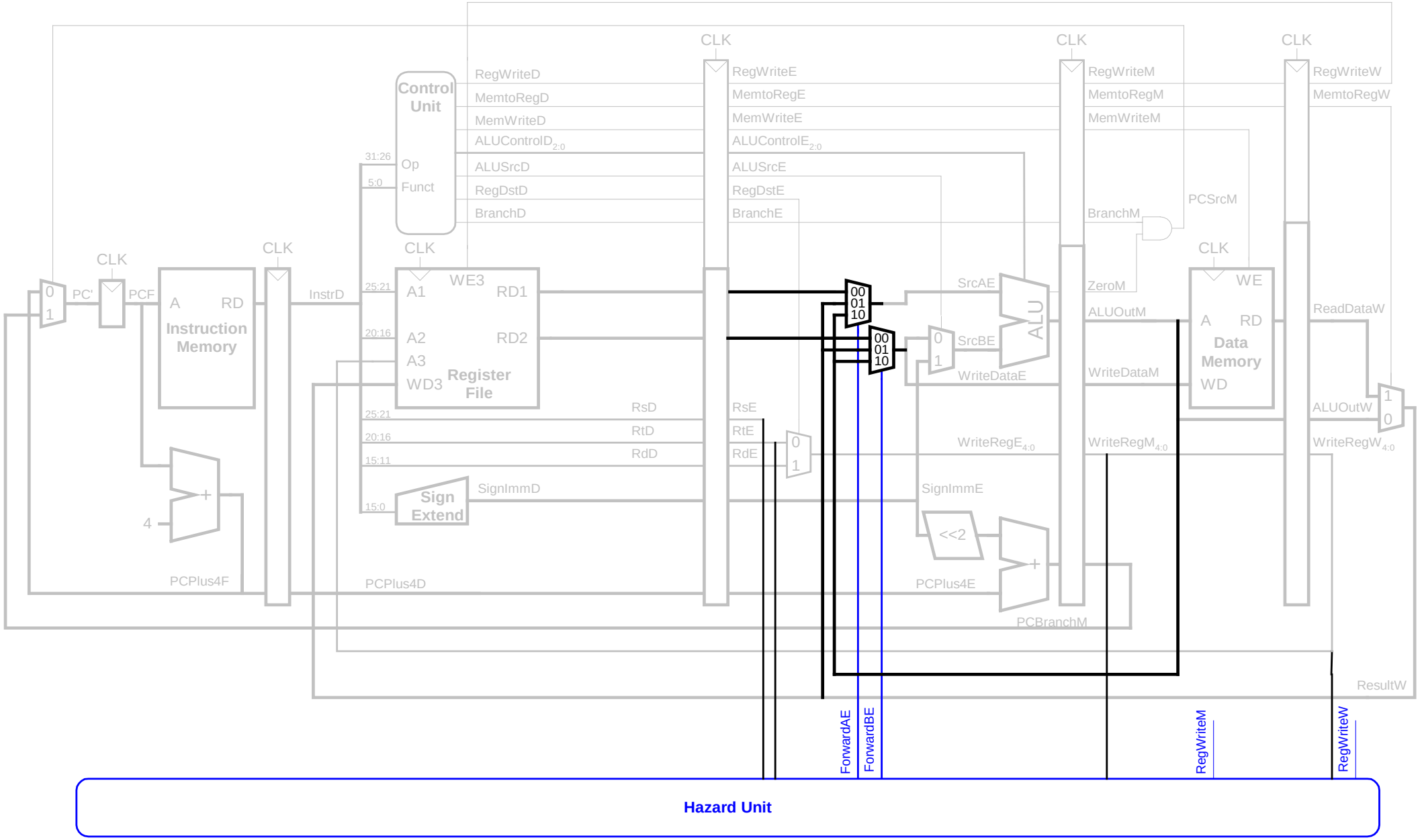
- Можна вставляти порожні інструкції (nop) в код програми перед компіляцією або під час компіляції
- Під час виконання програми реалізувати апаратну передачу даних з одного етапу конвеєра на інший не очікуючи завершення інструкції
- Під час виконання програми зупиняти (stall) деякі етапи конвеєра до того часу, поки проблемна інструкція не запише в регістровий файл результат, від якого залежать інструкції на зупинених етапах

Усунення конфліктів на рівні компілятора

Вставити в код достатньо порожніх інструкцій (nop), якими будуть заповнюватися стадії конвеєра, поки необхідний результат не буде записаний у регістр



Передача даних між стадіями (Forwarding, Bypass)



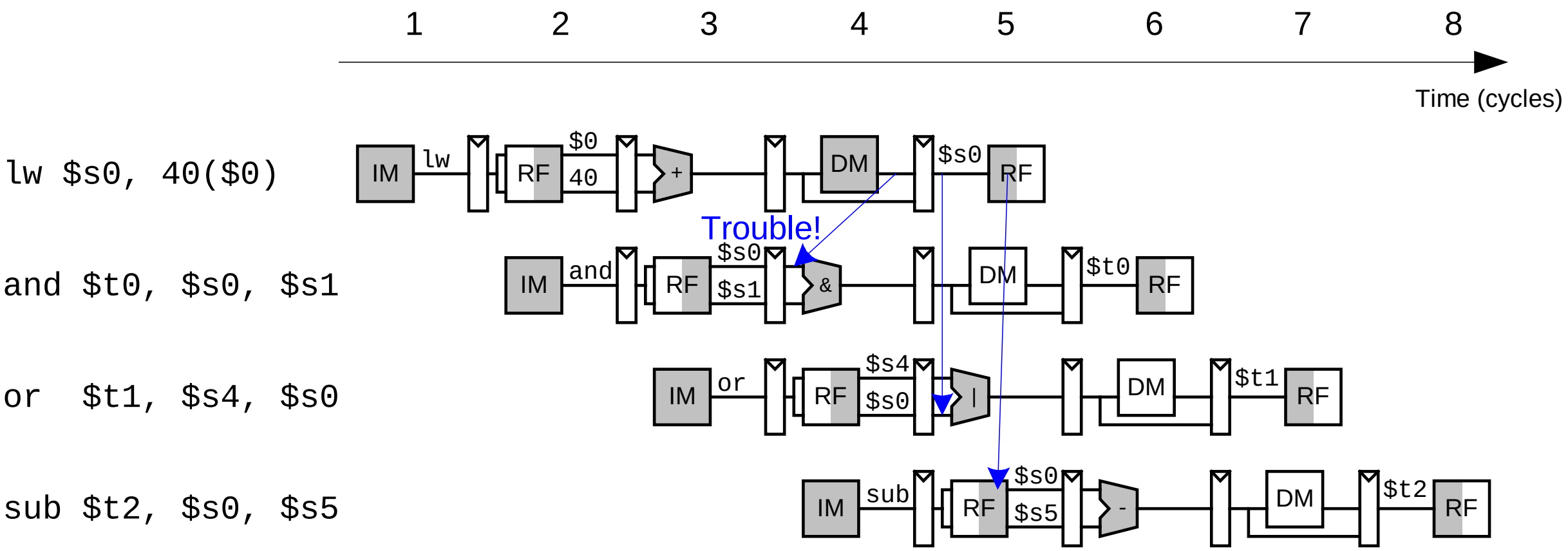
Передача даних між стадіями (Forwarding, Bypass)

- Можна передавати необхідні дані на етапі Виконання з етапів:
  - Доступу до пам’яті або
  - Запису результатів в регістровий файл
- Керуюча логіка для *ForwardAE*:

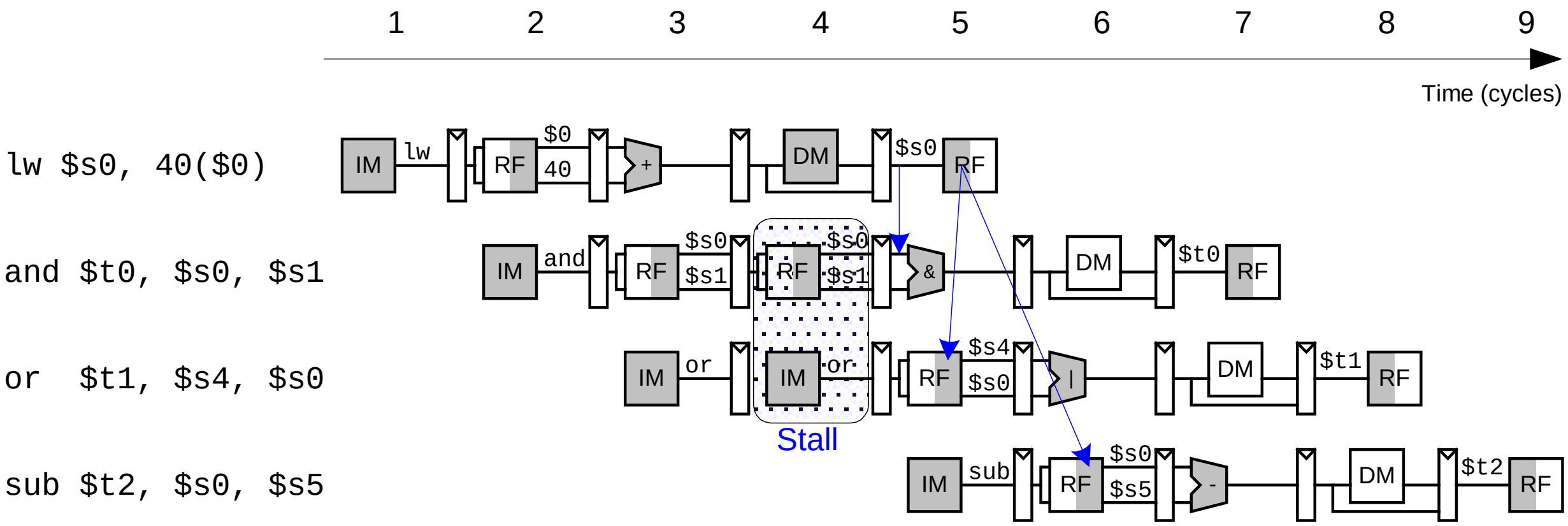
```
if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
     then  ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
     then  ForwardAE = 01
else       ForwardAE = 00
```

Керуюча логіка для ***ForwardBE*** подібна, але потрібно замінити ***rsE*** на ***rtE***

Зупинка конвеєра



Зупинка конвеєра

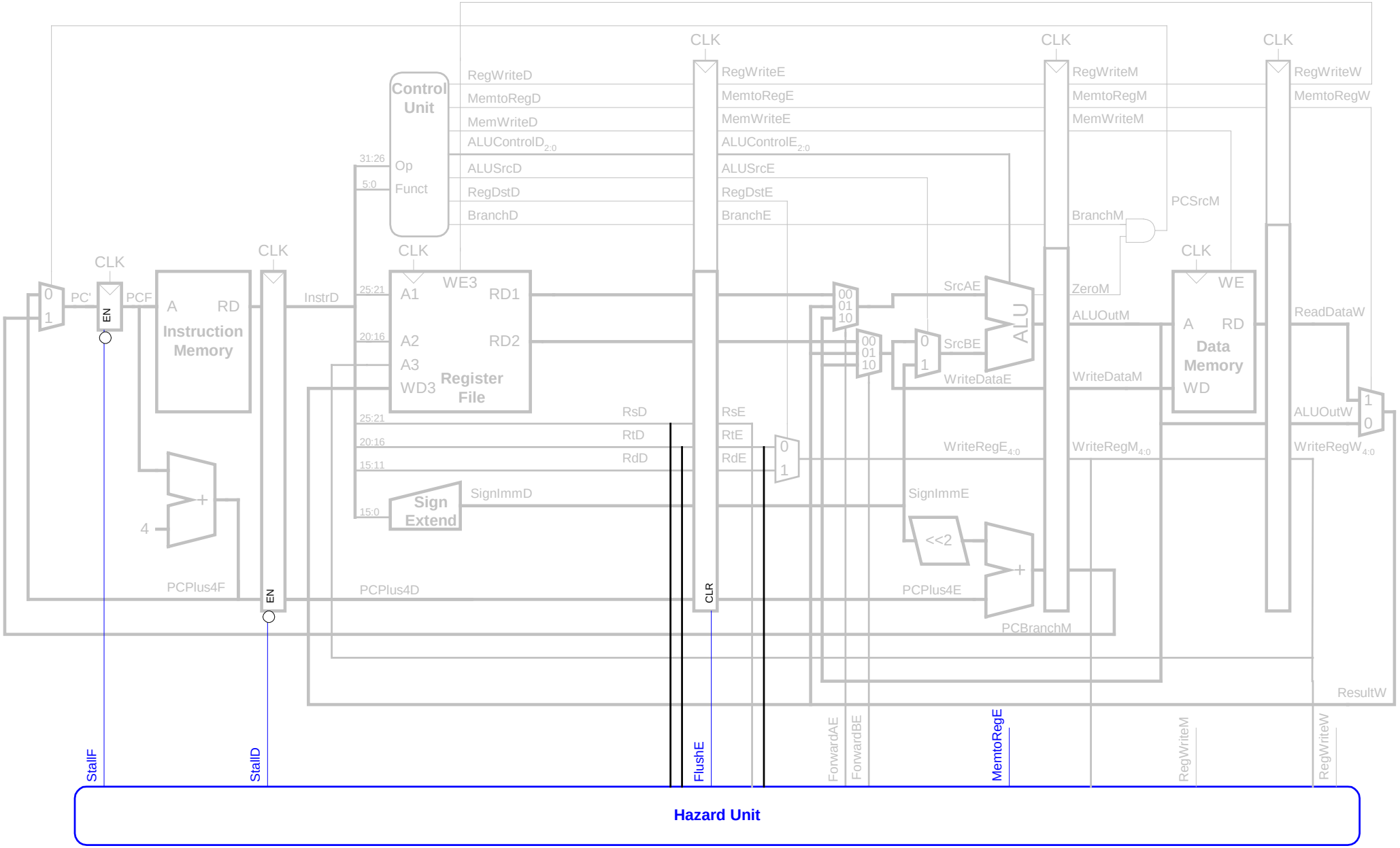


Логіка керування зупинкою (для інструкції *lw*)

lwstall* = ((*rsD*==*rtE*) OR (*rtD*==*rtE*)) AND *MemtoRegE

StallF* = *StallD* = *FlushE* = *lwstall

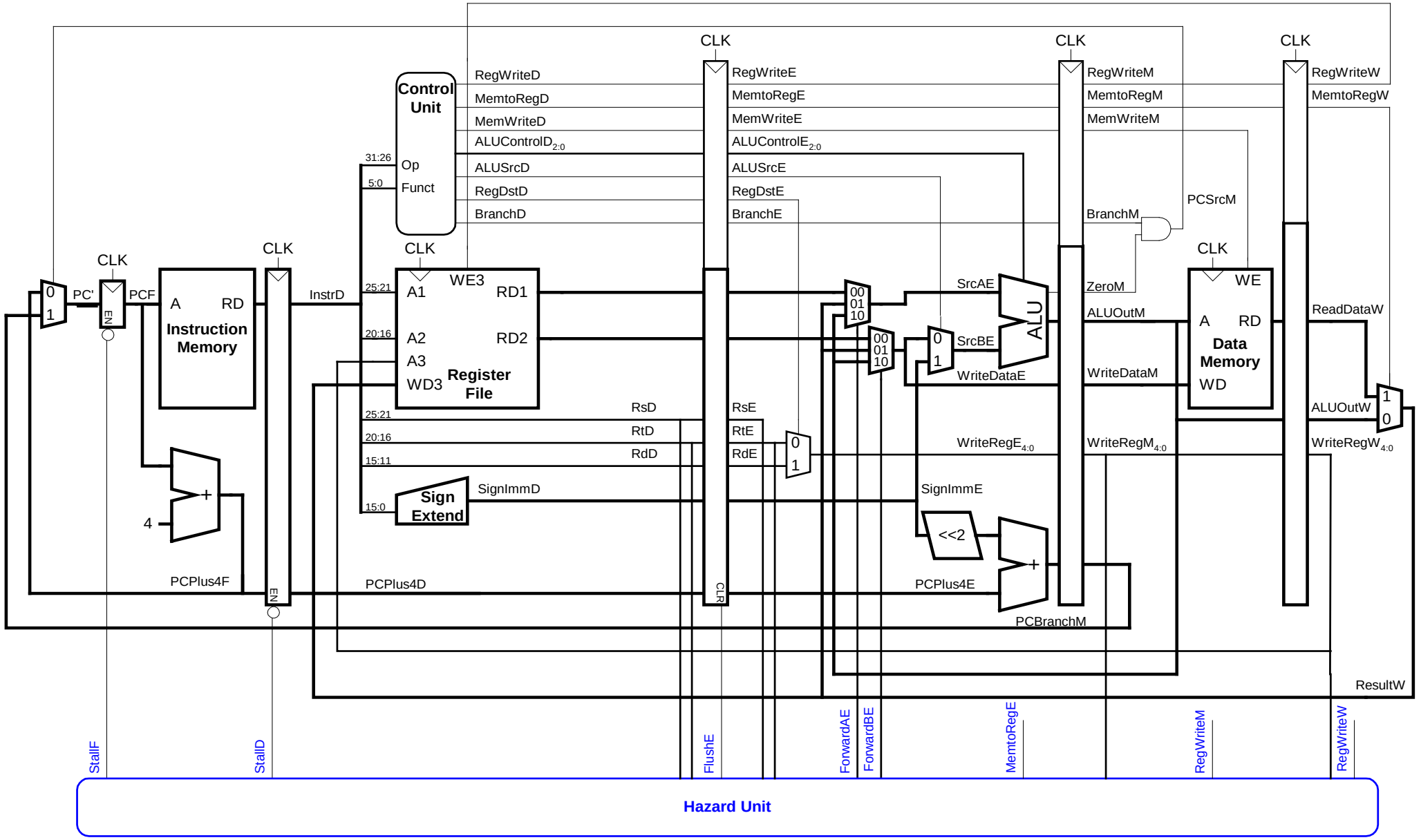
Зупинка конвеєра



Конфлікти керування

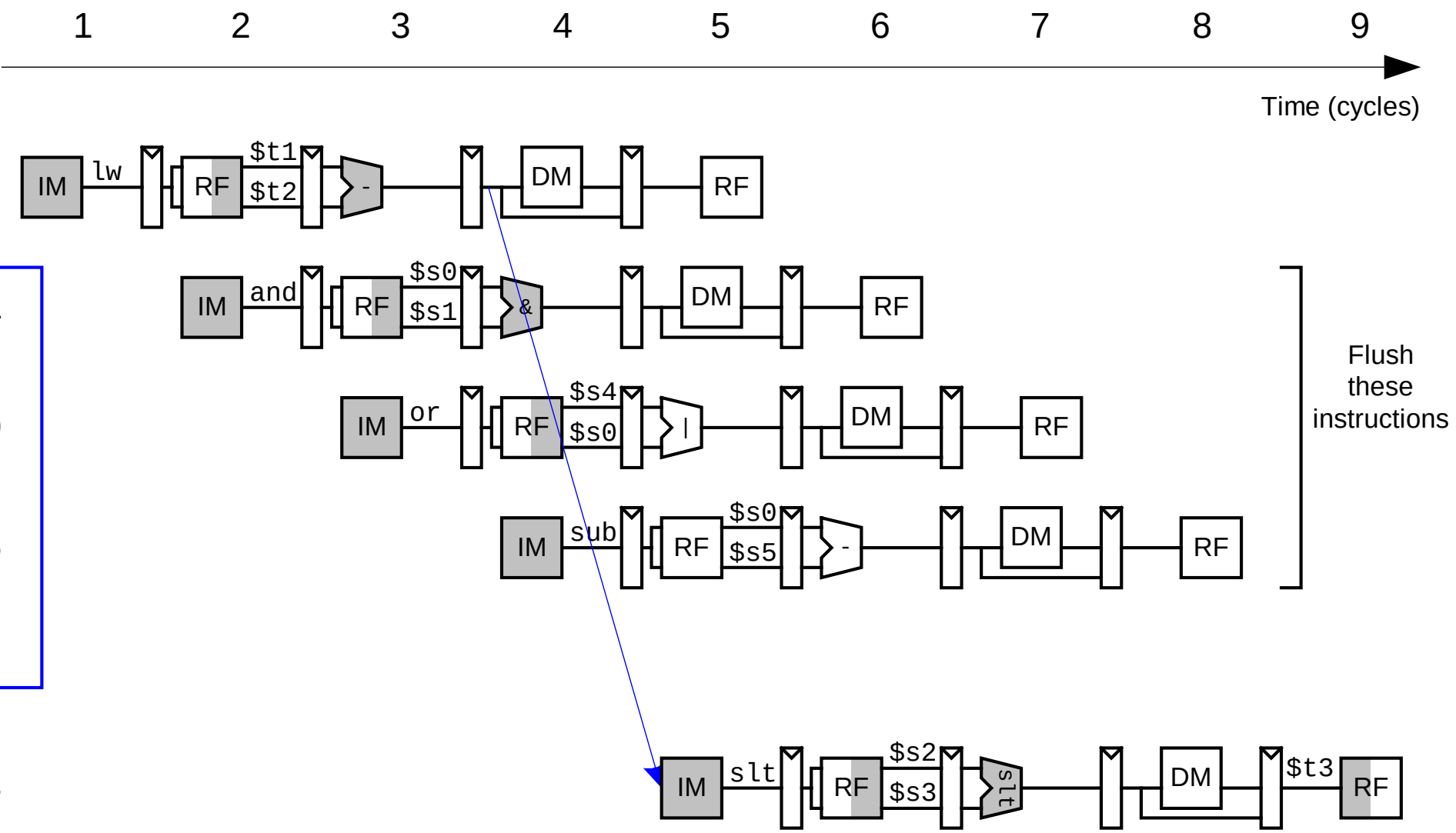
- **`beq`:**
  - Буде виконаний умовний перехід або ні, стане відомо тільки на 4-й стадії конвеєра
  - Поки це не стане відомо, інструкції слідуючі за інструкцією умовного переход продовжують попадати в конвеєр
  - У випадку необхідності умовного переходу ці інструкції (які йдуть після *beq*) не повинні бути виконані і їх потрібно вилучити з конвеєра
- **Ціна неправильного передбачення результату умовного переходу**
  - Кількість інструкцій, які необхідно вилучити з конвейера, якщо перехід все таки відбудеться
  - Цю кількість можна зменшити, перевіряючи умову переходу на більш ранніх стадіях конвеєра

Конвеєр для розв'язування конфліктів керування

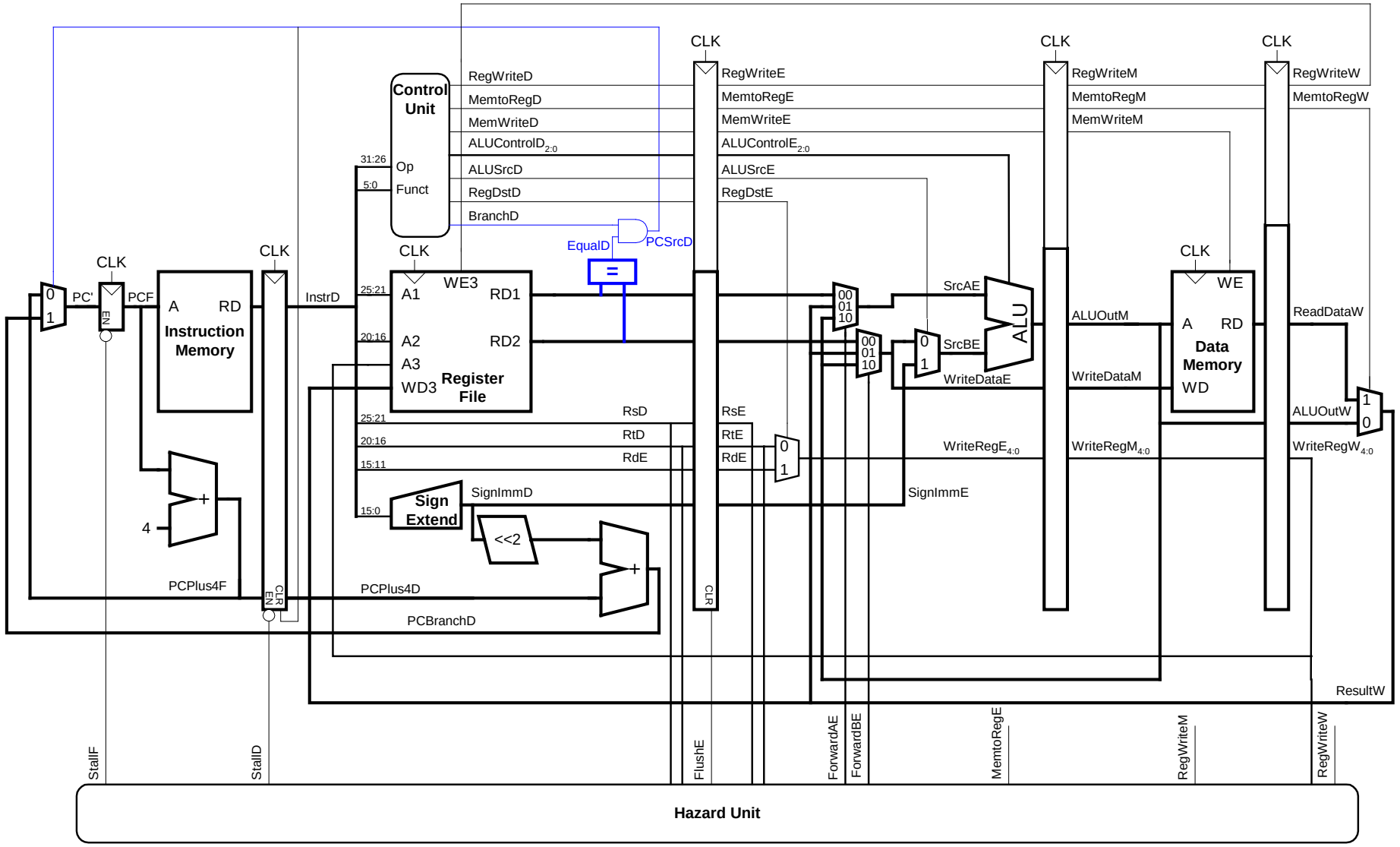


Конфлікти керування

```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

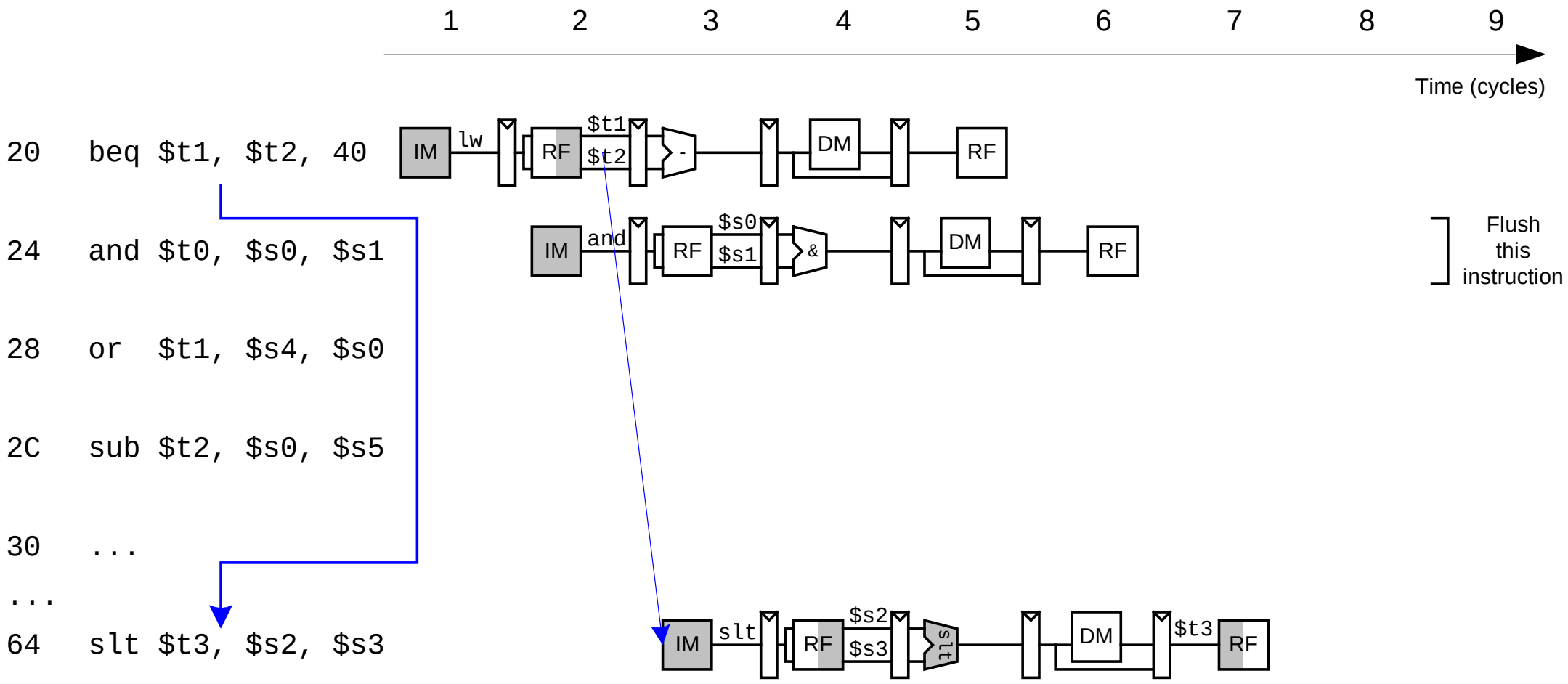


Рання перевірка умови переходу



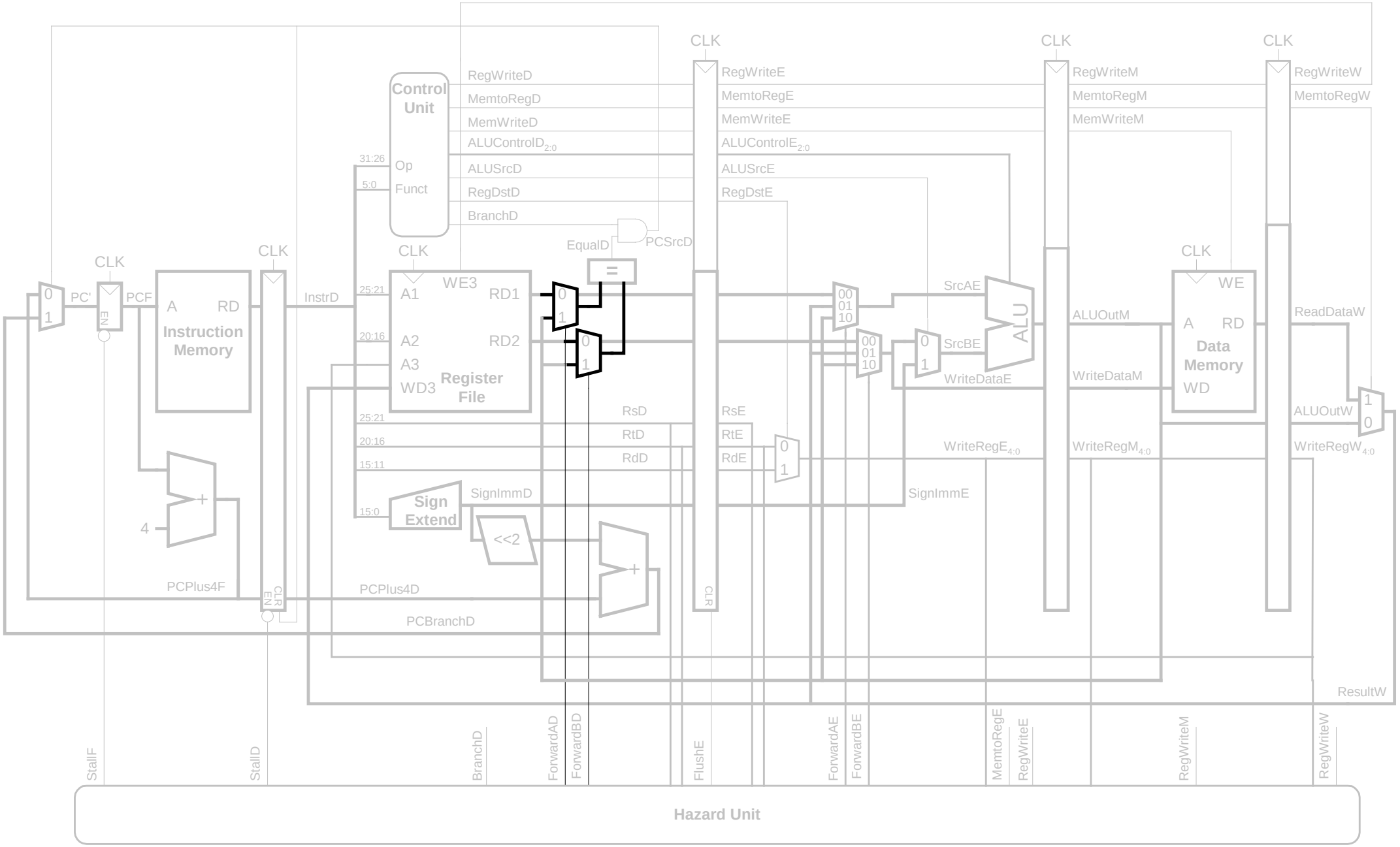
Спричиняє новий конфлікт даних на стадії Декодування

Рання перевірка умови переходу



Інструкцію завантажену в конвеєр після *beq* не обов'язково вилучати у випадку виконання переходу. Можна ввести умову, що інструкція слідуюча за переходом (умовним або безумовним) виконається завжди. Таке припущення називається branch delay slot.

Усунення конфліктів керування і даних



Логіка усунення конфліктів

- **Логіка керування передачею даних між стадіями конвеєра (Forwarding logic):**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

- **Логіка зупинки конвеєера (Stalling logic):**

```
branchstall = BranchD AND RegWriteE AND
                    (WriteRegE == rsD OR WriteRegE == rtD)
                OR
              BranchD AND MemtoRegM AND
                    (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = lwstall OR branchstall
```

Передбачення переходів

- Можна попробувати оцінити на скільки ймовірне виконання умовного переходу і використати найбільш ймовірний результат
  - Наприклад, в циклах найбільш ймовірне виконання умовних переходів назад (переходи на початок ітерації циклу більш всього виконаються, ніж не виконаються)
  - Для покращення передбачення переходів можна використати результати попередніх передбачень (наприклад, якщо три попередніх рази був перехід назад, то швидше всього це цикл і в наступний раз умовний перехід назад також відбудеться)
- Добре передбачення зменшує кількість скидань стадій конвеєра

Оцінка продуктивності конвеєрного процесора

- Тестовий набір SPECINT2000 містить :
  - 25% інструкцій *lw*
  - 10% інструкцій *sw*
  - 11% умовних переходів
  - 2% безумовних переходів
  - 52% інструкцій R-типу
- Припустимо:
  - 40% зчитувань із пам’яті використовується наступною інструкцією
  - 25% переходів передбачаються невірно
  - Всі переходи вилучають наступну інструкцію з конвеєра
- **Який середній CPI?**
  - Для *lw*/*beq* CPI = 1 якщо не має зупинку, CPI = 2 у випадку зупинки
  - $CPI_{lw} = 1(0.6) + 2(0.4) = 1.4$
  - $CPI_{beq} = 1(0.75) + 2(0.25) = 1.25$

Середній CPI = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1) **= 1.15**

Оцінка продуктивності конвеєрного процесора

- Затримка самого довгого ланцюга комбінаційної логіки конвеєрного процесора:

$$T_c = \max \{$$
$$t_{pcq} + t_{\text{mem}} + t_{\text{setup}}$$
$$2(t_{RFread} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}})$$
$$t_{pcq} + t_{\text{mux}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{setup}}$$
$$t_{pcq} + t_{\text{memwrite}} + t_{\text{setup}}$$
$$2(t_{pcq} + t_{\text{mux}} + t_{\text{RFwrite}}) \}$$

Оцінка продуктивності конвеєрного процесора

Параметр	**Позначення**	**Затримка (пс)**
Час запису у регістр	t_{pcq_PC}	30
Час передвстановлення регістру	t_{setup}	20
Затримка мультиплексору	t_{mux}	25
Затримка АЛП	t_{ALU}	200
Затримка зчитування з пам’яті	t_{mem}	250
Затримка зчитування з регістрового файлу	t_{RFread}	150
Час передвстановлення регістрового файлу	$t_{RFsetup}$	20
Затримка компаратору	t_{eq}	40
Затримка елементу I	t_{AND}	15
Затримка записування у пам’ять	$T_{memwrite}$	220
Затримка записування у регістровий файл	$t_{RFwrite}$	100 ps

$T_c = 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup}) = 2[150 + 25 + 40 + 15 + 25 + 20]$ ps **= 550 пс**

$t = N \times CPI \times T_c = (100 \times 10^9)(1.15)(550 \times 10^{-12})$ **= 63 сек**

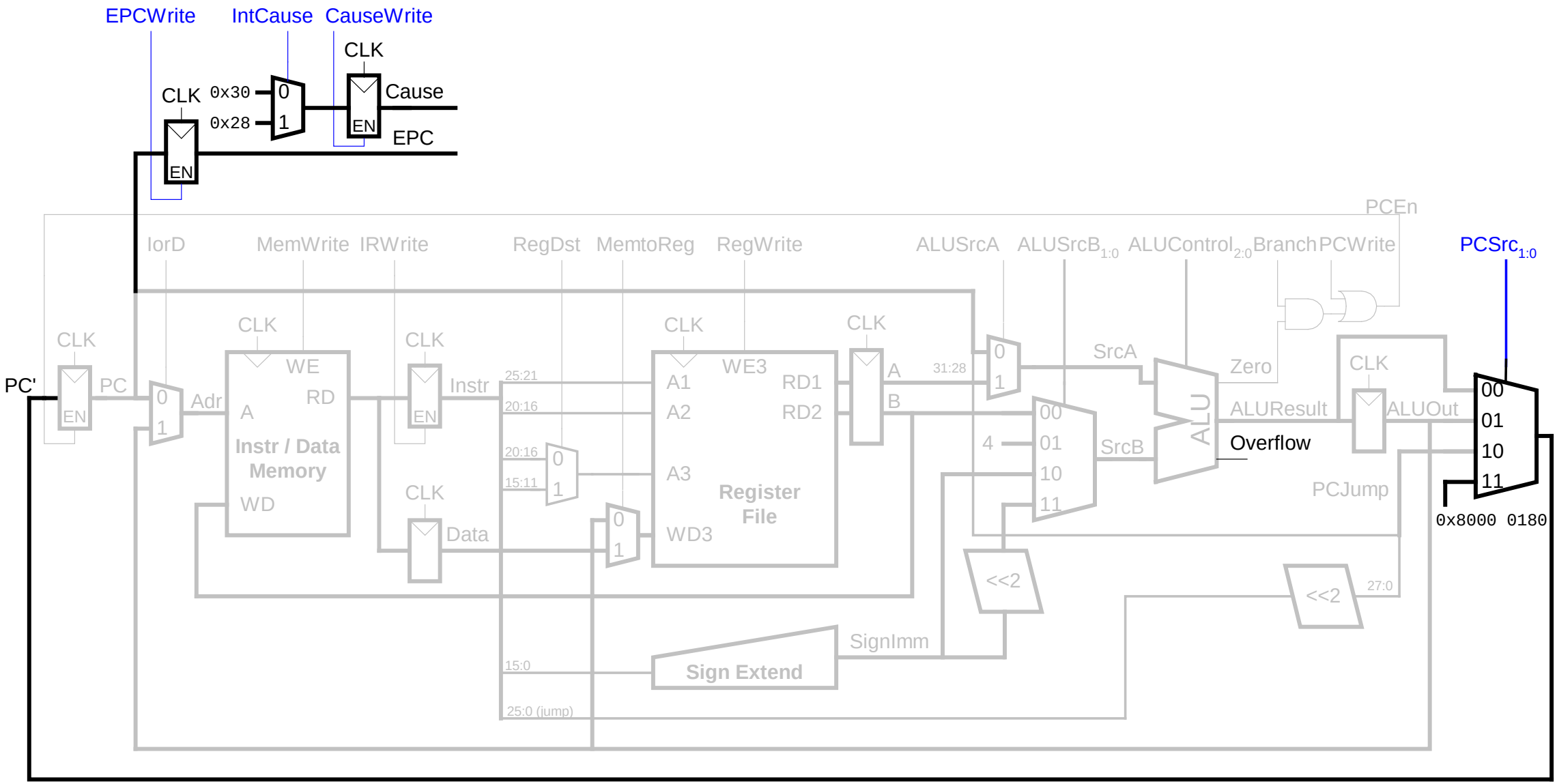
Порівняння продуктивності

Мікроархітектура	Час виконання (секунди)	Приріст продуктивності
Однотактна	92.5	1
Багатотактна	133	0.70
Конвеєрна	63	1.47

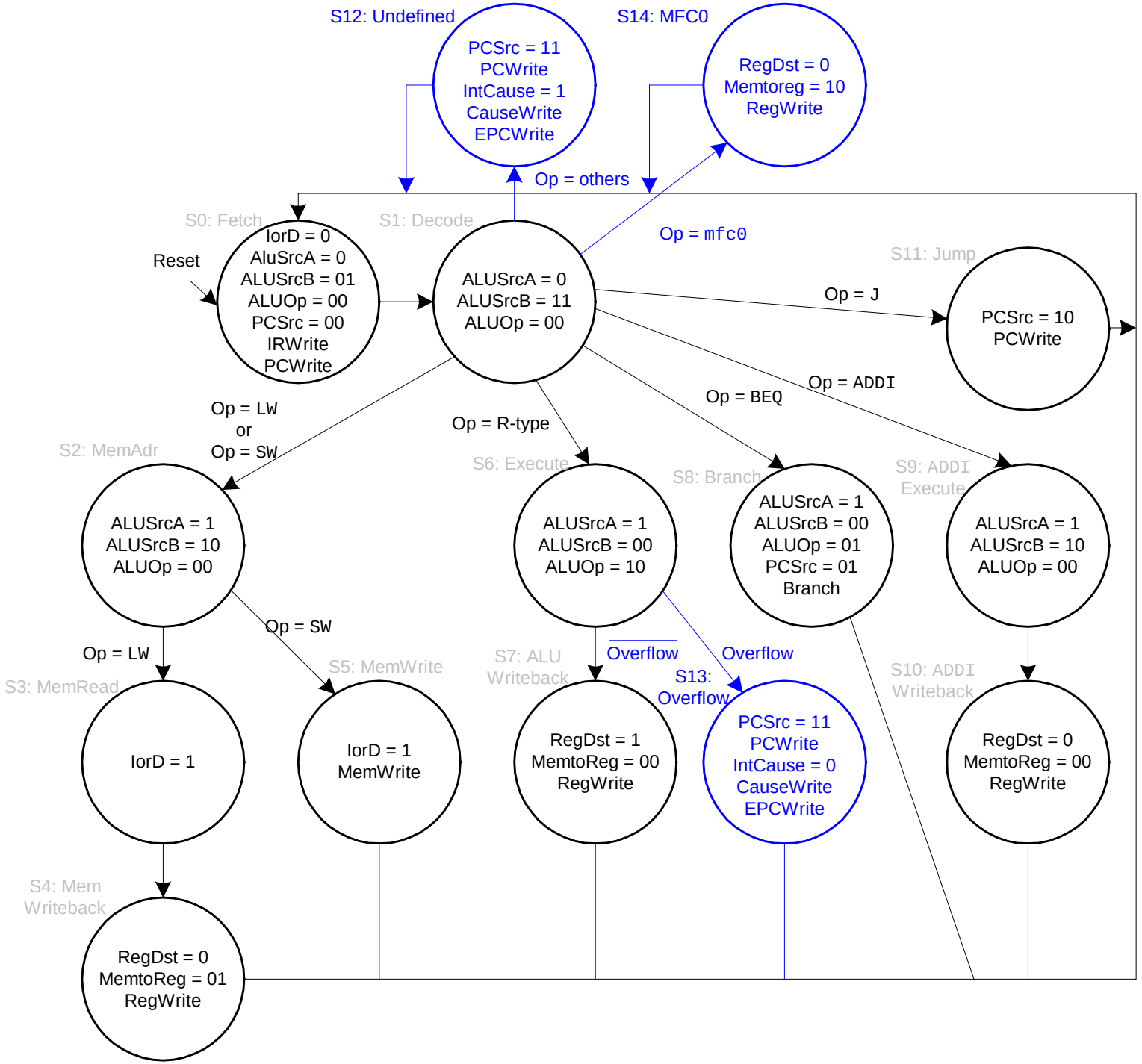
Добавлення оброблення винятків у MIPS процесор (останніх двох типів)

Винятки	Код причини
Апаратне переривання	0x00000000
Системний виклик	0x00000020
Точка зупинки / Ділення на 0	0x00000024
Невідома інструкція	0x00000028
Арифметичне переповнення	0x00000030

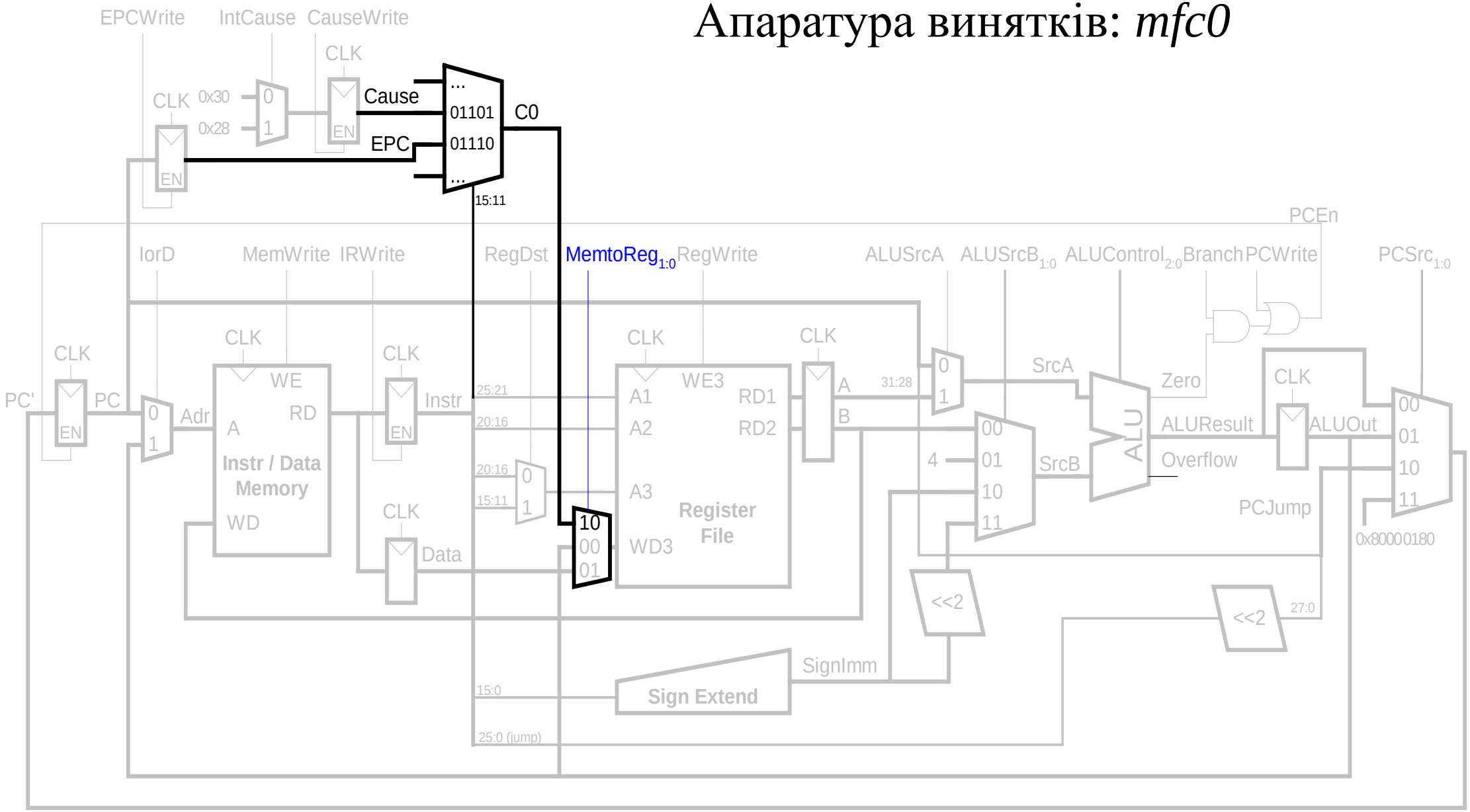
Апаратура винятків: EPC і Cause



Винятки у керуючому автоматі



Апаратура винятків: *mfc0*



Покращення мікроархітектури

- Довгі конвеєри (10-20 стадій)

  Кількість стадій обмежується:
  - конфліктами конвеєра
  - енергоспоживанням
  - вартістю
  - збільшенням затримки тактового сигналу
- Динамічне передбачення переходів
- Суперскалярні процесори
- Процесори з позачерговим виконанням інструкцій
- Перейменування регістрів
- SIMD
- Багатопоточність
- Багатопроцесорність

Передбачення переходів

- У ідеального конвеєрного процесора: CPI = 1
- Неправильне передбачення переходів збільшує CPI
- **Статичне передбачення переходів:**
  - Перевіряється напрямок переходу (вперед або назад)
  - Якщо перехід назад, вважається, що він буде виконаний
  - Інакше, вважається, що перехід не буде виконаний
- **Динамічне передбачення переходів:**
  - Процесор містить таблицю з останніми декількома сотнями (або тисячами) інструкцій умовного переходу. Цю таблицю ще називають буфером цільових адрес галужень (branch target buffer). Вона містить адреси переходів і інформацію про те, чи був виконаний перехід.

Приклад передбачення переходів

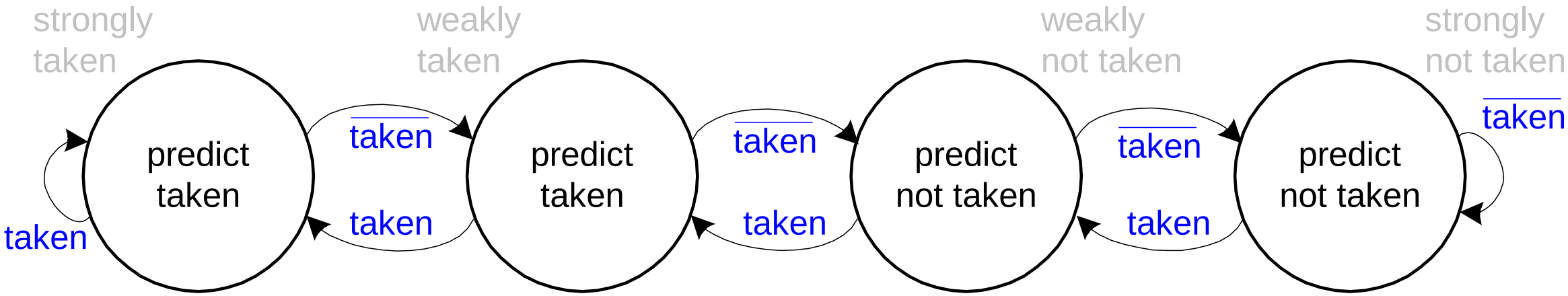
```
  add  $s1, $0, $0      # sum = 0
  add  $s0, $0, $0      # i   = 0
  addi $t0, $0, 10      # $t0 = 10
for:
  beq  $s0, $t0, done   # if i == 10, branch
  add  $s1, $s1, $s0    # sum = sum + i
  addi $s0, $s0, 1      # increment i
  j    for
done:
```

Однобітовий динамічний передбачувач переходів

- Запам’ятовує, чи був перехід виконаний в попередній раз, і передбачає, що в наступний раз відбудеться те ж саме
- Помиляється двічі: для першого і останнього умовних переходів циклу

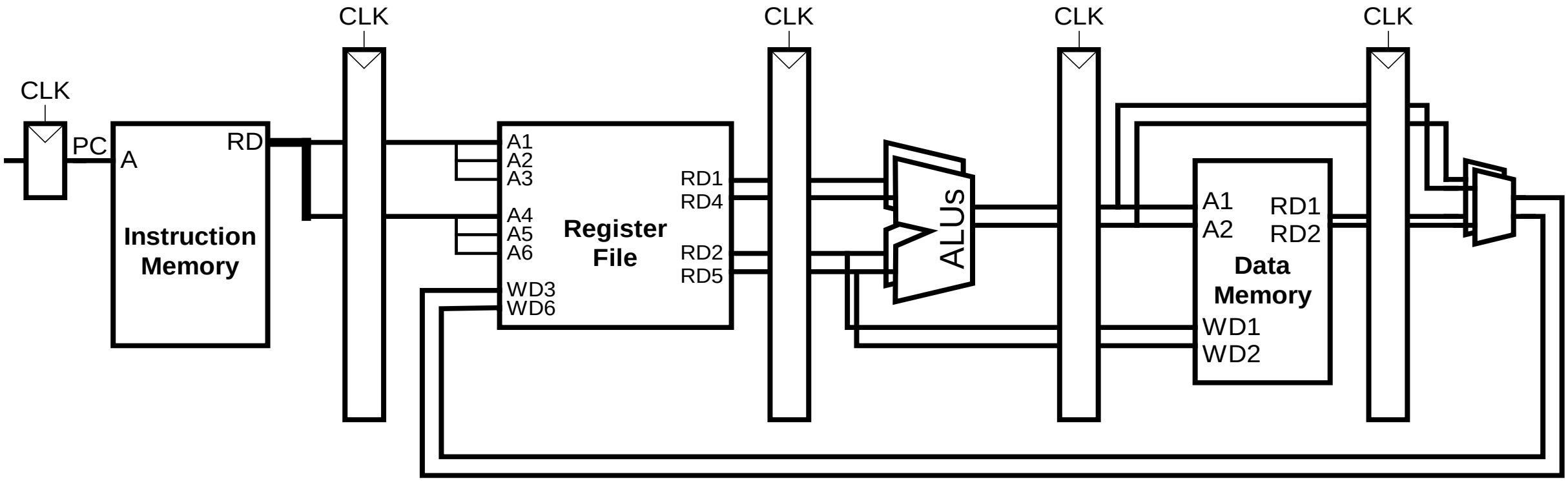
  (на першій і останній ітерації)

Двобітовий динамічний передбачувач переходів



Дає невірне передбачення тільки для останнього умовного переходу циклу (на останній ітерації)

Суперскалярний процесор



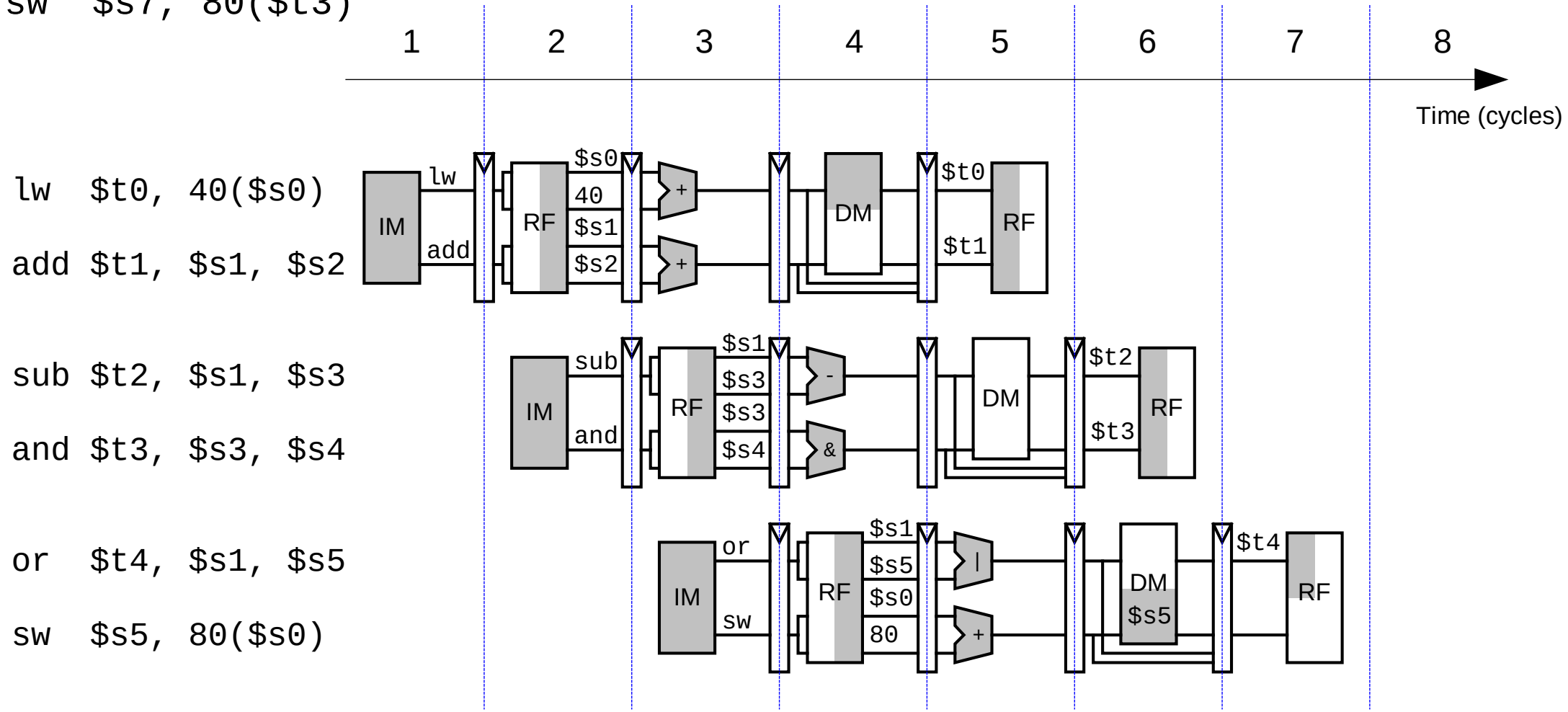
- Дозволяє одночасно зчитувати і виконувати декілька інструкцій за рахунок дублювання функціональних блоків
- Так ніби в процесорі одночасно функціонує декілька конвеєрів
- Залежності між інструкціями значно ускладнює їх одночасне виконання

Приклад суперскалярності

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Ідеальний IPC: 2

Реальний IPC: 2

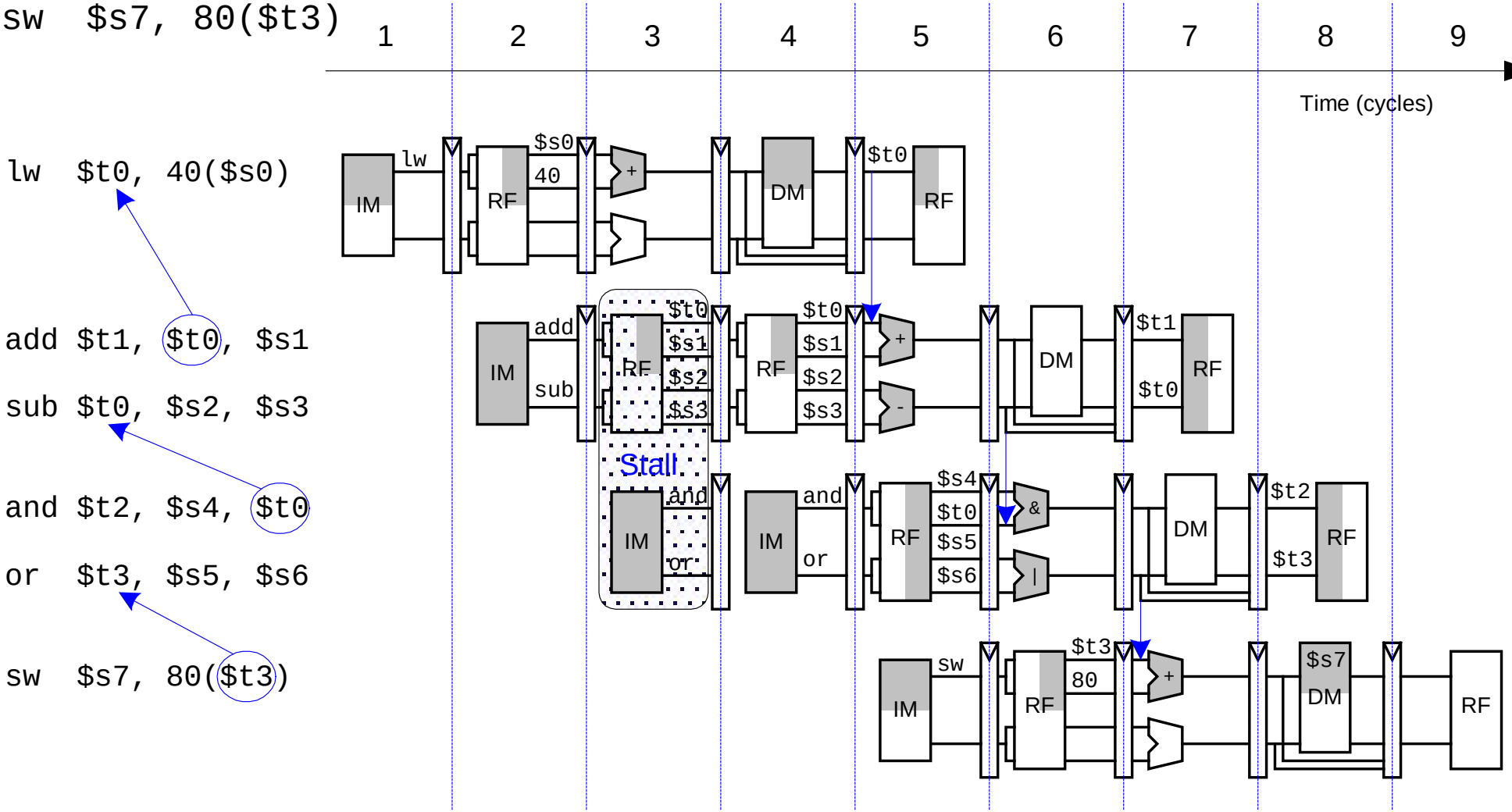


Суперскалярність із залежностями

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Ідеальний IPC: 2

Реальний IPC: 6/5 = 1.17



Позачергове виконання інструкцій

- Процесор заздалегідь передивляється наперед велику кількість інструкцій, знаходить незалежні одна від одної інструкції і запускає їх на одночасне виконання
- Інструкції можуть виконуватися не за тим порядком, в якому вони розміщені у програмі
- Процесор слідкує за тим, щоб позачергове виконання не нарушало алгоритм роботи програми
- **Залежності:**
  - **RAW** (read after write, читання після запису): попередня інструкція записує, наступна зчитує регістр
  - **WAR** (write after read, запис після читання): попередня інструкція зчитує регістр, наступна інструкція записує в цей регістр
  - **WAW** (write after write, запис після запису): інструкція пробує писати в регістр після того, як у нього вже записала наступна по ходу програми інструкція

Позачергове виконання інструкцій

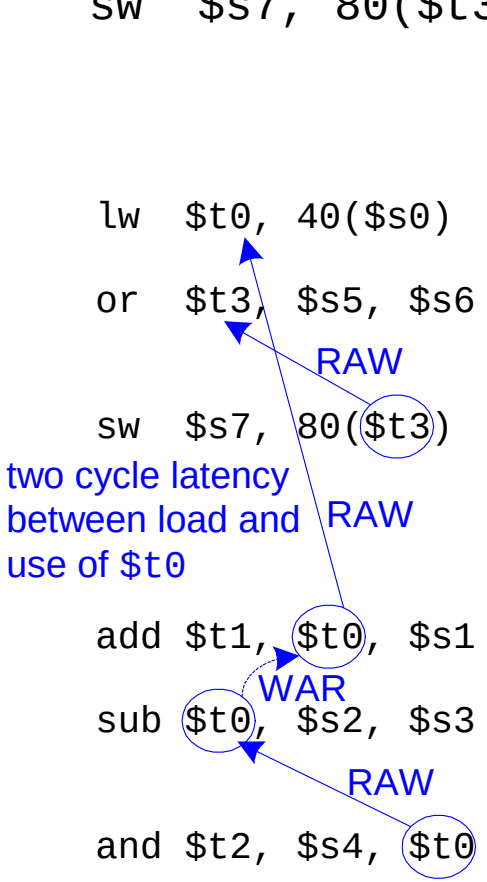
- **Паралелізм на рівні інструкцій (Instruction level parallelism, ILP):** число інструкцій, які можуть виконуватися одночасно (звичайно < 3)
- **Таблиця готовності (Scoreboard):** таблиця, яка зберігає інформацію про:
  - інструкції, очікуючі виконання
  - доступні функціональні блоки (АЛП, порти пам’яті і т.д.)
  - залежності між інструкціями

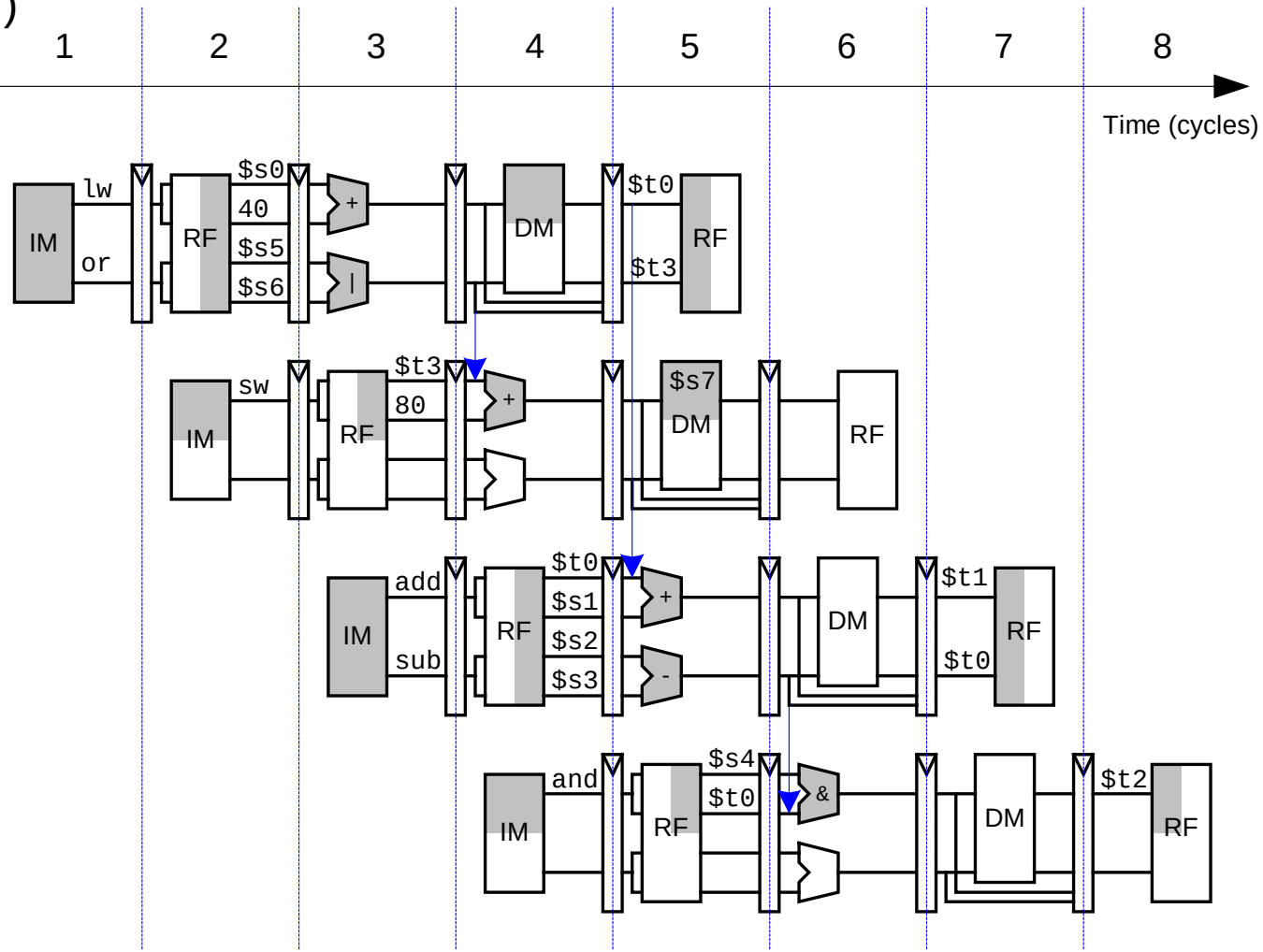
Приклад позачергового виконання

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Ідеальний IPC: 2

Реальный IPC: 6/4 = 1.5



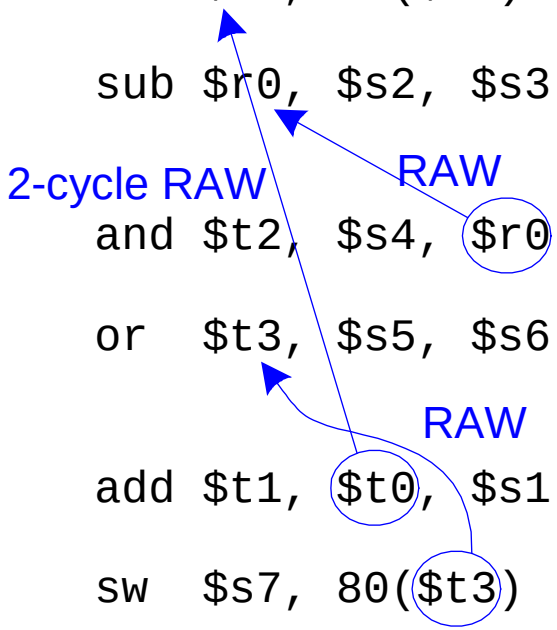


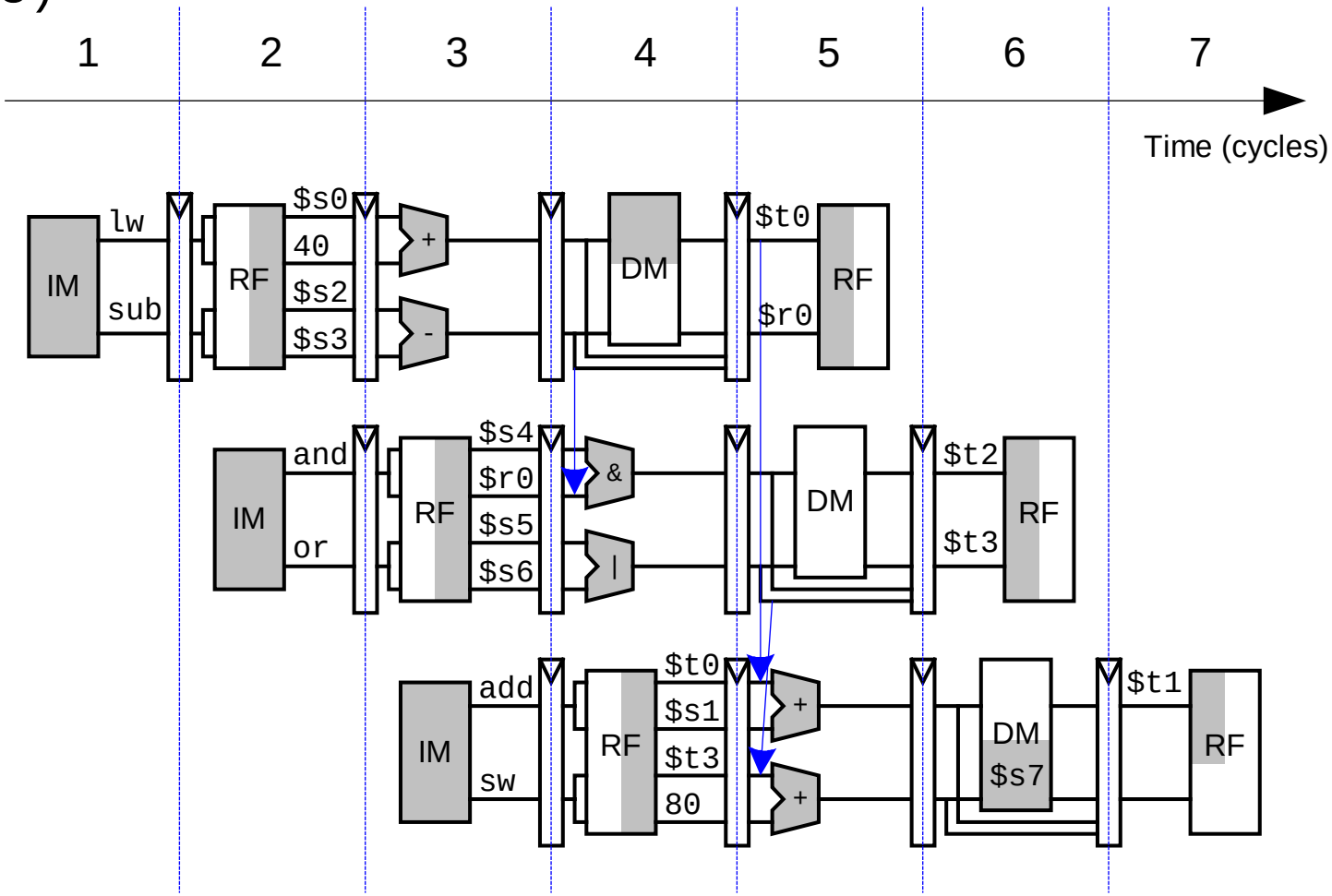
Перейменування регістрів

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Ідеальний IPC: 2

Реальний IPC: 6/3 = 2

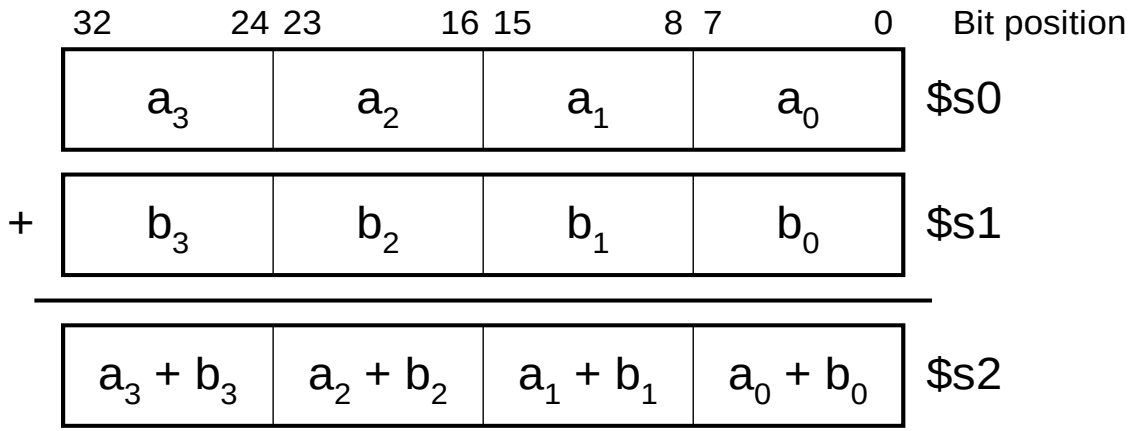




SIMD

- Одиночний потік команд, множинний потік даних (Single Instruction Multiple Data, SIMD)
  - Одна інструкція обробляє множину блоків даних одночасно (наприклад, паралельно додає декілька пар чисел)
  - Часто використовується у комп'ютерній графіці
  - Виконується арифметична операція над декількома невеликими незалежними блоками даних (запакована арифметика)
- Наприклад, в 32-розрядному суматорі можна одночасно додавати 4-ри пари 8-бітових операндів

```
padd8 $s2, $s0, $s1
```



Багатопоточність і мультипроцесорність

- **Багатопоточність**
  - Наприклад, в текстовому редакторі один потік може відповідати за обробку введення з клавіатури символів і набирання тексту, інший потік “одночасно” виконувати перевірку правопису, третій потік може при цьому виводити текст на друк
- **Мультипроцесорність**
  - Декілька окремих процесорів всередині одного кристалу

Потоки: визначення

- **Процес:** програма, яка виконується на комп’ютері
  - Декілька процесів можуть виконуватися одночасно, наприклад: веб пошук, прослуховування музики, написання статті в текстовому редакторі
- **Потік:** частина процесу (програми)
  - Процес може містити декілька потоків, наприклад текстовий редактор може містити потоки для набору тексту, перевірки орфографії, друку

Потоки у звичайному процесорі

- У кожний момент часу виконується один потік
- Коли виконання потоку блокується (наприклад, потік очікує дані з повільної внутрішньої пам’яті):
  - архітектурний стан потоку зберігається
  - архітектурний стан наступного потоку завантажується в процесор і потік запускається на виконання
  - Така процедура називається **перемиканням контексту**
- До тих пір, поки процесор перемикається між потоками достатньо швидко, користувачу здається, що всі потоки виконуються одночасно.

Багатопоточність

- У багатопоточного процесора є декілька копій архітектурного стану
- Декілька потоків можуть бути **активними** одночасно:
  - коли виконання одного потоку блокується, сразу ж запускається виконання іншого потоку на наявних функціональних блоках
  - Якщо один потік не використовує всі функціональні блоки процесора, їх використовує інший потік
- Багатопоточність не впливає на паралелізм на рівні інструкцій (ILP) окремого потоку, але збільшує загальну продуктивність обчислень

Intel називає таку технологію “hyperthreading”

Багатопроцесорність

- Багатопроцесорна система (multiprocessor system), або просто мультипроцесор, складається з декількох процесорів і апаратури для з’єднання їх між собою
- Типи:
  - **Гомогенна (симетрична) багатопроцесорність**: декілька однакових процесорів підключені до загальної шини пам’яті
  - **Гетерогенна (асимметрична) багатопроцесорність:** різні типи процесорних ядер використовуються для задач різних типів (наприклад, в мобільному телефоні для обчислень використовується звичайний процесор, а для оброблення аудіо/відео – спеціалізоване DSP ядро)
  - **Кластери:** кожне ядро має свою власну пам’ять