ЛР № 8. Конкурентне виконання машинних інструкцій

Мета: Ознайомитись з принципами паралельного виконання інструкцій у процесорах типу RISC на прикладі архітектури MIPS64. Навчитись виявляти і долати залежності між інструкціями (WAR, WAW, RAW), оцінювати ефективність конвеєризації та оптимізувати код з урахуванням апаратного рівня.

Завдання:

За допомогою архітектурного симулятора **WinMIPS64**:

- Проаналізуйте паралельне (конвеєрне) виконання інструкцій створеної вами програми.
- Побудуйте два варіанти фрагмента коду:
  - **Неоптимізований** — із наявністю залежностей (RAW, WAR, WAW), затримками та конфліктами ресурсів.
  - **Оптимізований** — з мінімізованими затримками, використанням технік перепризначення регістрів (register renaming), перестановок інструкцій або іншими методами усунення конфліктів.
- Проведіть порівняльний аналіз за такими метриками (див. вікно *Statistics*):
  - Кількість тактів симуляції (simulation cycles)
  - Кількість виконаних інструкцій
  - Середня кількість тактів на інструкцію (CPI)
  - Кількість затримок (stalls)
  - Інші показники (load/store, conditional branches тощо)

 На основі результатів сформуйте **аналітичний звіт**, у якому:

- Опишіть обидва варіанти програми
- Розкрийте причини виникнення конфліктів та способи їх усунення
- Прокоментуйте ефективність оптимізованого варіанта з погляду апаратного прискорення

Теоретичні відомості

Процесор фірми MIPS, що з певними наближеннями симулюється програмою WinMIPS, містить процесор фіксованої коми (центральний процесорний вузол, CPU) разом із певними копроцесорами, що виконують допоміжні або службові функції, наприклад обробку в форматі рухомої коми, або ж керування пам'яттю і вводом-виводом див. наступний рисунок). Наш симулятор симулює центральний процесорний вузол і лише один з двох копроцесорів, а саме, копроцесор 1, що є вузлом обробки форматів рухомої коми. Копроцесор обслуговує виключні ситуації, що виникають під час виконання програми (наприклад, ділення на нуль, переповнення тощо), переривання, що дозволяють апаратним засобам реагувати на асинхронні події, та систему віртуальної пам'яті. (Рис 1).

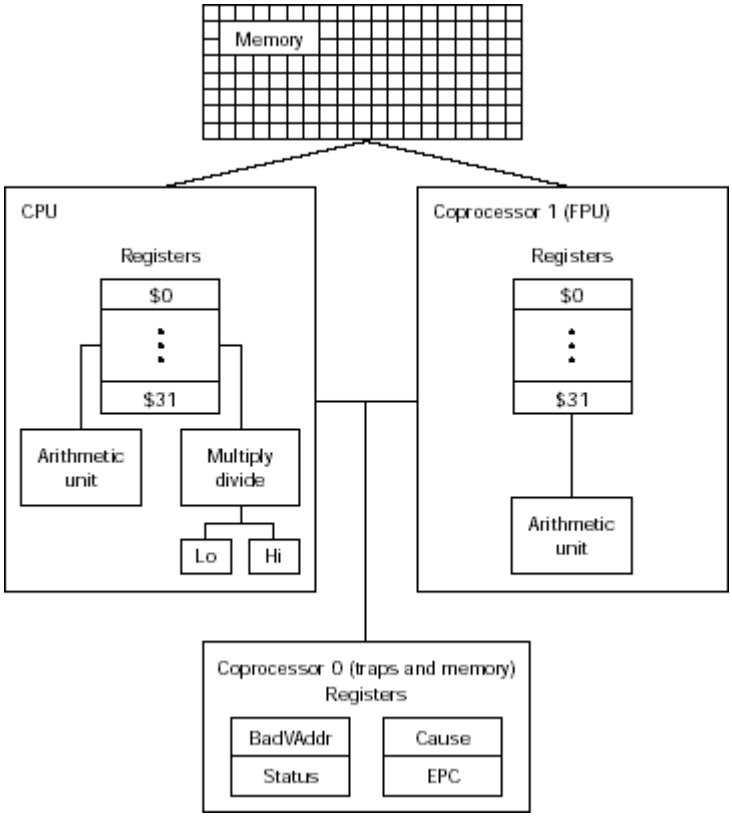

Рис. 1. Процесор MIPS R2000, CPU і FPU.

де:
Cause – причина переривання або виключення,
EPC – допоміжний лічильник інснструкцій, що зберігає адресу перерваної програми,
BadVAddr – некоректне значення адреси комірки пам'яті,

Status – стан,
trap – пастка (для запуску системниxx програми, що опрацьовують ситуації виключення і переривання),
Lo – молодша частина, Hi – старша частина (результату множення або ділення).

В подальшому викладенні для ознайомлення з функціонуванням симулятора використовується асемблерна програма, яку містить файл FACT.S. Програма обраховує факторіал цілого числа, а саме ціле числе треба уводити з клавіатури після запуску процесу симулювання. Разом із цією асемблерною програмою використовується допоміжна асемблерна програма вводу, що містить файл INPUT.S. Зауважимо, що розширення s в позначенні файлів відповідає асемблерним програмам.

Запуск симулятора

Симулятор використовує операційну систему Windows.

Симулятор WinMIPS64 стартує за правилами Windows. Основне вікно симулятора містить шість дочірних вікон і ще статусну лінію під ними. Дочірні вікна отримали назви Pipeline, Code, Data, Registers, Statistics і ще вікно часових діаграм (Clock Cycles Diagram).

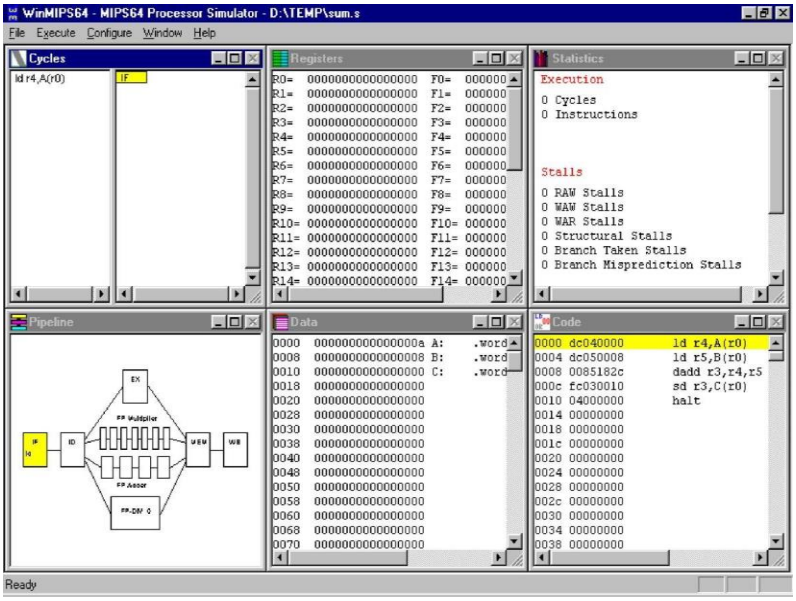

Рис. 2 – Основне і шість дочірних вікон симулятора WinMIPS64 (завантажено програму sum.s)

Далі подамо назви і пояснимо призначення дочірніх вікон симулятора (Таблиця 1).

Таблиця 1 – Дочірні вікна симулятора WinMIPS64

Pipeline window (вікно конвеєра інструкцій, five pipeline stages : 1. IF – instruction fetch from instruction memory, 2. ID – instruction decoding/operand fetch, 3. EX – execute, 4. MEM – to/from data memory, 5. WB – write back to registers file)	Вікно містить “схематичне” подання п’ятисходинкового конвеєра інструкцій 64-розрядного процесора MIPS64 разом з апаратними секціями виконання операцій рухомої коми, а саме, додавання/віднімання (addition/subtraction) множення (multiplication) і ділення (division). Рухоме ділення не конвеєризоване.Це вікно показує, на якій сходинці конвеєра знаходиться та чи інша інструкція. Вікно може збільшуватися.
Code window (вікно коду)	Вікно виконує триколонкове подання пам’яті інструкцій, а саме, (зліва направо): адреса байта, 32-бітову машинну інструкцію, асемблерну інструкцію. Подвійний лівий щиголь мишею на певній інструкції встановлює/знімає точку зупинки
Data window (вікно даних)	Це вікно показує вмістиме пам’яті даних,. Ясно, що адресування комірок виконується побайтне, проте у вікні вмістиме подане 64-бітовими пакетами, тобто так, як це вмістиме сприймає 64- розрядний процесор. Аби відредагувати певні дані, треба зробити на них подвійний лівий щиголь мешею (double-left-click). Аби побачити і відрегувати дані з рухомою комою, треба виконати подвійний правий щиголь (double-right-click).

Register window (регістрове вікно)	Вікно подає вмістиме регістрів. Коли вмістиме регістрів подають сірим кольором, тоді вмістиме цих регістрів змінюється під дією програми, що симулюється. Коли вмістиме регістру подається кольором, тоді цей колір відповідає кольору сходинки конвеєра, де знаходится відповідна інструкція за умови, щоз цієї сходинки є можливим так зване випередження (*forwarding)*. Це вікно дозволяє інтерактивну зміну вмістимого будь-якого регістру, тобто, залежно від контексту, зміну коду 64-бітового цілого або рухомого числа, що в поточний момент містить регістр. Останнє можливе лише за умови, що обраний регістр не знаходиться в процесі програмної зміни вмістимого і його вмістиме не має не використовуватися для випередження даними. Аби змінити вмістиме регістра на ньому треба зробити подвійний лівий щиголь (double-left- click on the register). З'явиться модальне спливаюче вікно. Треба натиснути OK, аби записати до регістру 64-бітове гексадецимальне 0x0000000000000777.
Clock Cycle diagram (вікно часової діаграми)	Вікно містить часову поведінку конвеєра, що знаходиться під дією поточної програми, що симулюється. Воно фіксує історію кожної інструкції. Коли певна інструкція спричиняє пригальмування конвеєра, тоді її символьне подання в лівій частині циклового вінка міняє колір з чорного на синій. Інструкції, що споживають результат пригальмованої інструкції змінюють колірність на сіру.
Statistics (вікно статистики) Саме це вікно є найважливішим, адже воно накопичує результати виконання кожної лабораторної роботи з дослідження ефективності RISC архітектури.	Вікно подає накопичені і поточні статистики програми, що симулюється, а саме, число циклів (тактових інтервалів) на симуляцію (number of simulation cycles), число виконаних за ці цикли інструкцій, середнє число циклів на одну інструкцію (average Cycles Per Instruction, тобто CPI), типи затримок (stalls) і числа виконаних особливих інструкцій, а саме, умовних переходів (conditional branches) і інструкцій завантаження / збереження (Load/Store).
Status Line (стан симулятора)	Статусна лінія на низу основного вікна симулятора нормально видає повідомлення "Ready",

Зауважимо, що числові результати обчислень за програмним кодом для нас є другорядними в порівнянні з аналізом обчислених симулятором статистик просимульованої програми.

Аби привести симулятор до початкового (стартового) стану перед симулюванням програми спочатку скидають симулятор через пункт меню *File* щиголем миші по *Reset MIPS64.* Потім, як треба, конфігурують симулятор (тобто віртуальну апаратну частину) через вибирання числених опцій функціонування цієї апаратури. Можна змінювати структуру і час виконання рухомих операцій на конвеєрі, місткість пам'яті коду/даних. Аби зробити це чи побачити стандартні призначення архітектури викликати наступне вікно.

Методика виконання лабораторної роботи

Ми дозволимо випускати на виконання інструкції рухомої коми зі сходинки ID тоді, коли це стає можливим. Така інструкція може продовжити виконання у власному конвеєрі виконання операції рухомої коми або призупинитися через неготовність її операндів. Обрана нами стратегія дозволяє продемонструвати переваги невпорядкованого завершення виконання (out-of- order completion), але вона також може спричинити небезпеку WAR конвеєрного виконання. Тут може допомогти техніка переназв регістрів (register renaming), яку треба досконально розуміти. Наприклад, таке може статися в наступному фрагменті коду:

```
.text
add.d f7,f7,f3add.d f7,f7,f4
mul.d f4,f5,f6          ; WAR через спільний регістр f4
```

Коли випустити mul.d, тоді ця інструкція (за певних умов) «пережене» другу інструкцію add.d і першої запише до f4. Отже, mul.d мусимо затримати на ID. Structural hazards arise at the MEM stage bottleneck, as instructions attempt to exit more than one of the execute stage piplelines at the same time. Просте правило запобігання небезпек звучить так: довгі інструкції виконують першими.

Нехай маємо фрагмент коду:

```
;************************************
;*** winMIPS64 //hazard3.s//          *****
;*** (c) 2003 CA226, DCU              *****
;************************************
.text
div.d f7,f9,f10mul.d f2,f4,f3 sub.d f7,f7,f4 ld      r1,78(r0)add.d f4,f5,f6 halt
```

Результати його синтаксичного контролю.

```
Pass 1 completed with 0 errors
;************************************
;*** winMIPS64 //hazard3.s//                          *****
;*** (c) 2005 CA226, DCU                              *****
;************************************
00000000             .text
00000000 462a49c3        div.d f7,f9,f1000000004 46232082     mul.d f2,f4,f3
00000008 462439c1        sub.d f7,f7,f4
0000000c dc01004e        ld  r1,78(r0)
00000010 46262900        add.d f4,f5,f6    ; WAR on f4
00000014 04000000        halt

Pass 2 completed with 0 errorsCode Symbol Table
Data Symbol Table
```

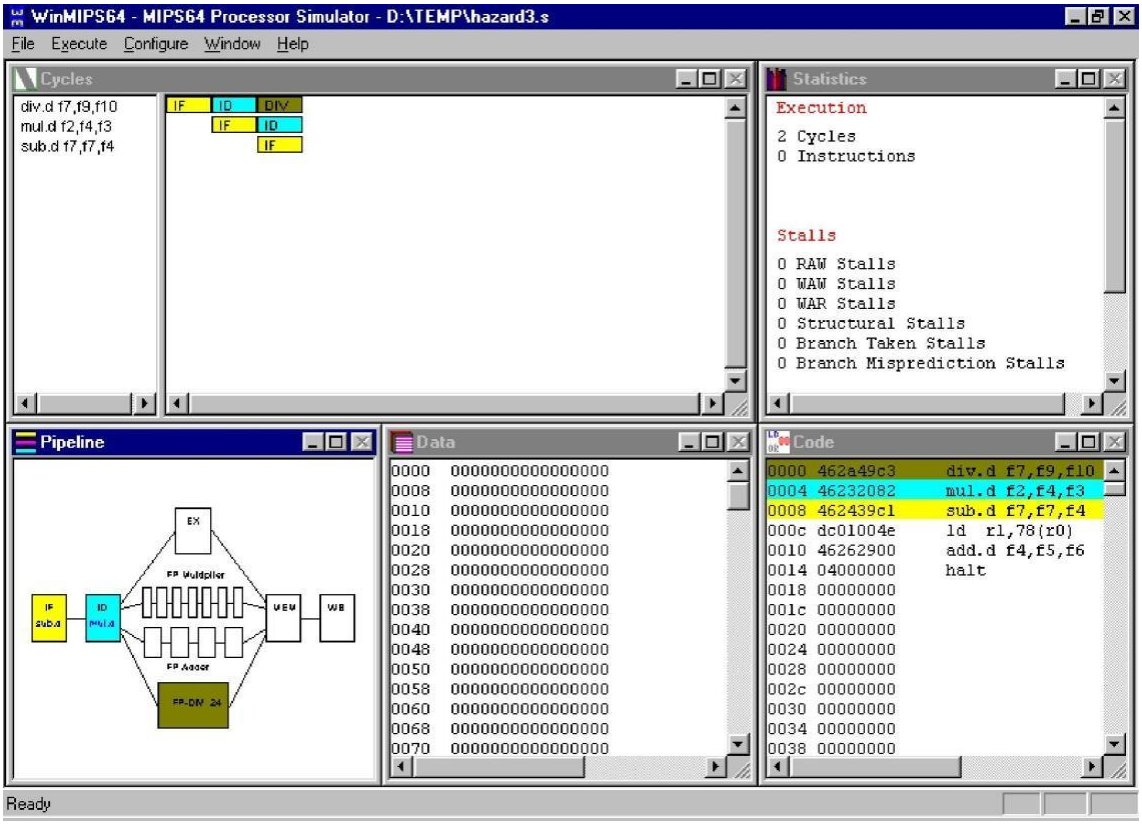

Рис. 3. Неконвеєрний повільний виконавчий вузол ділення з рухомою комою розпочав роботу першим

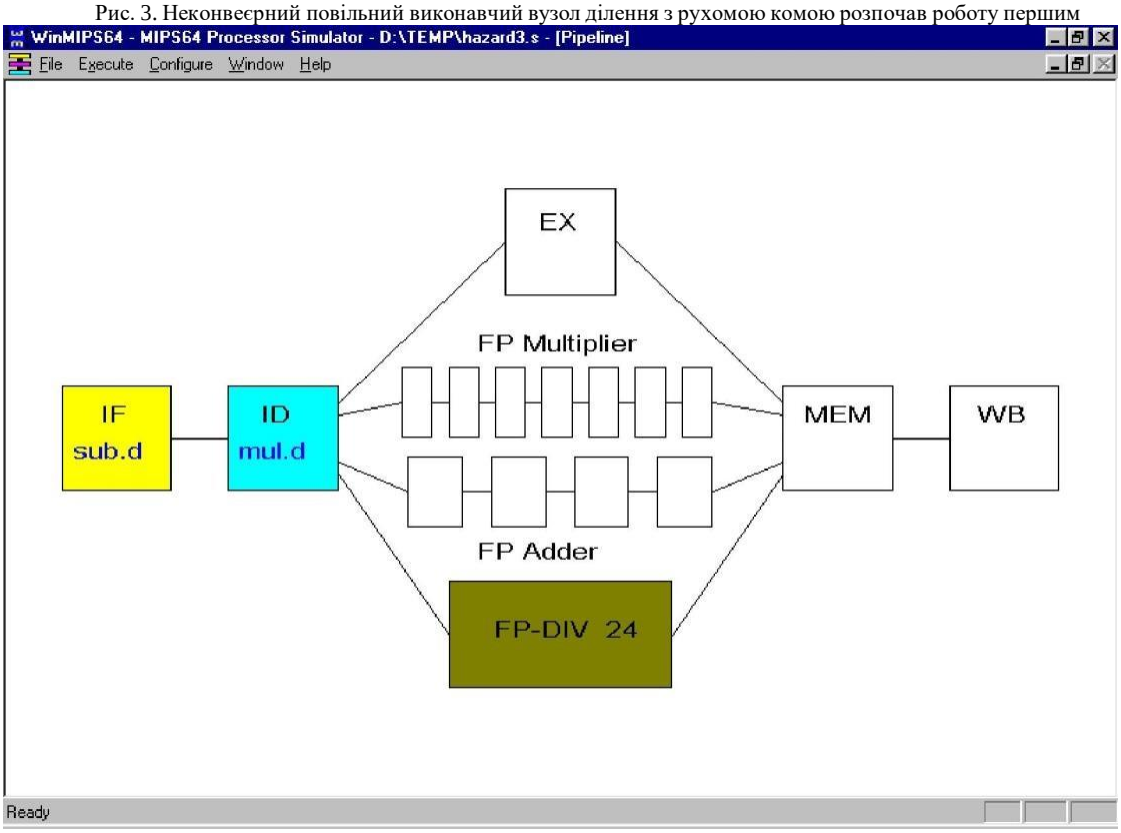

Рис. 4. Неконвеєрний виконавчий вузол ділення рухомої коми (24 такти затримки) завантажено

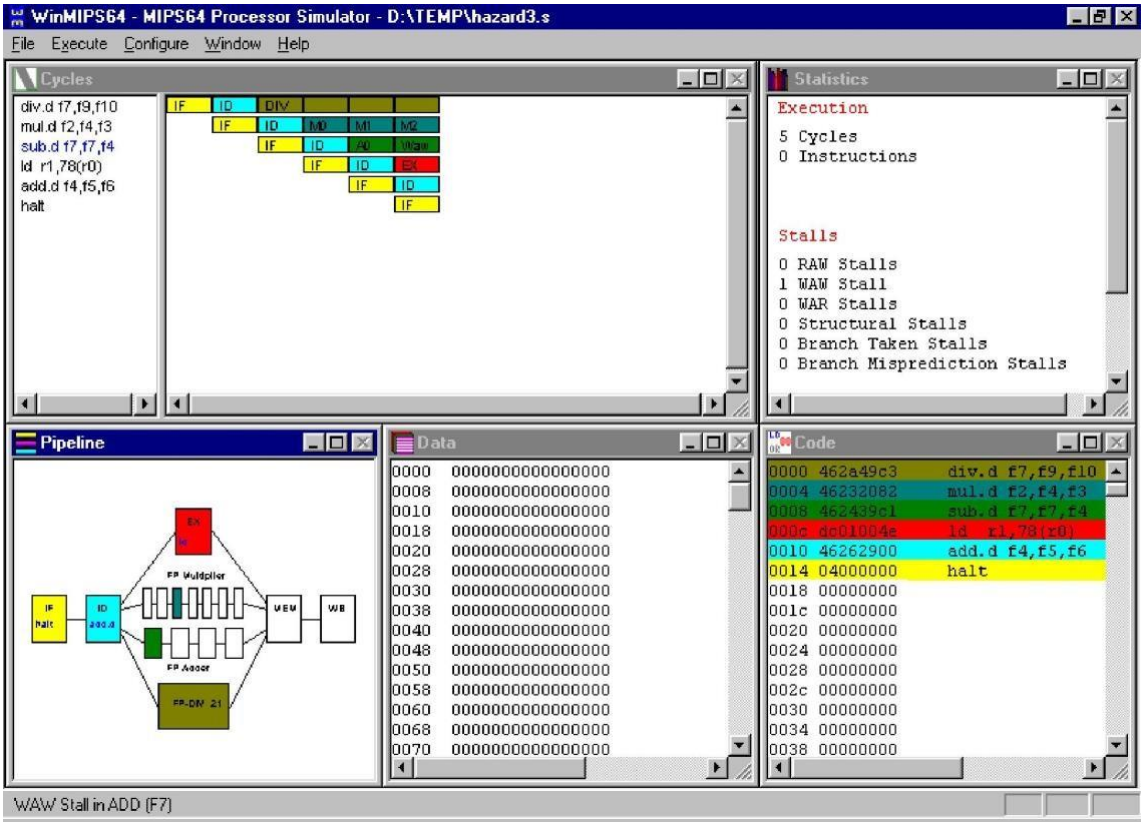

Рис. 5. Паралельно виконуються чотири інструкції (ясно, що з невпорядкованим завершенням)

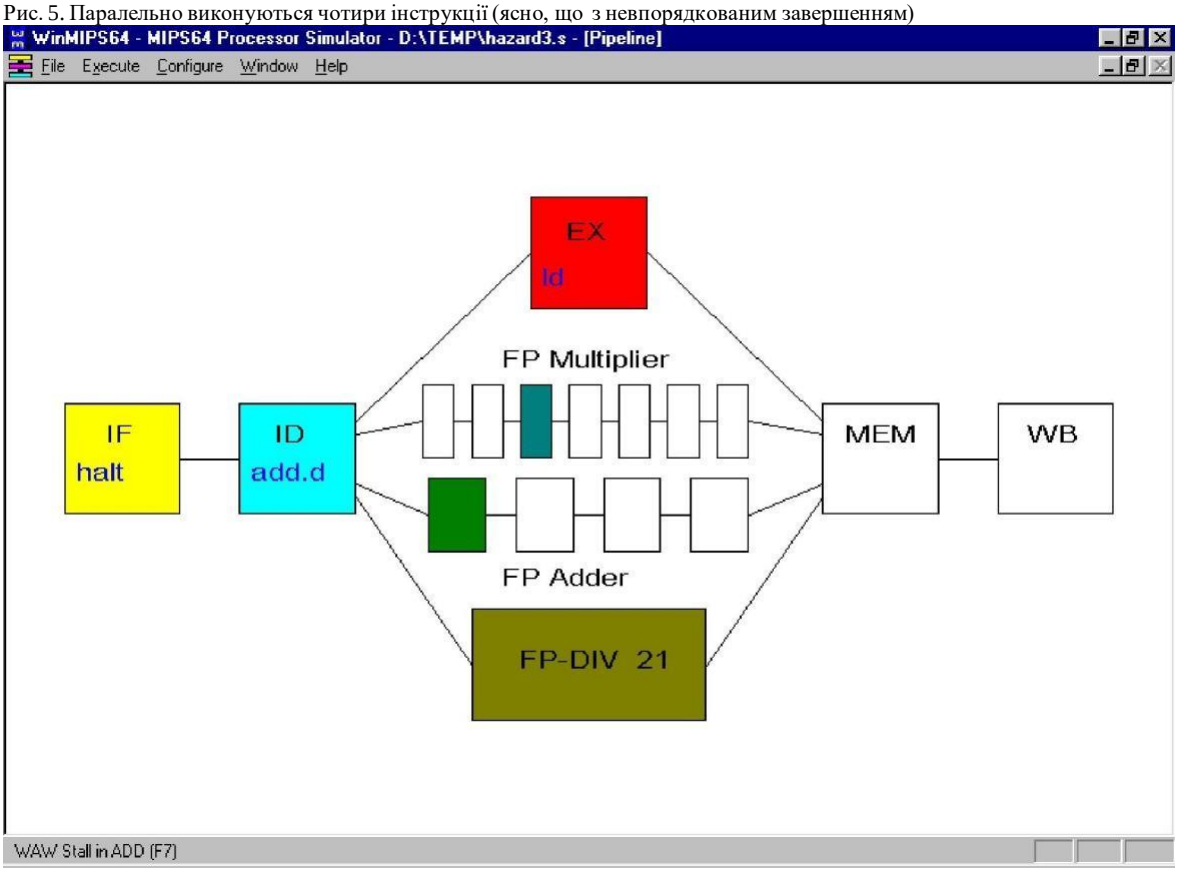

Рис. 6. Паралельне виконання продовжується, сходинки MEM жодна інструкція ще не досягла

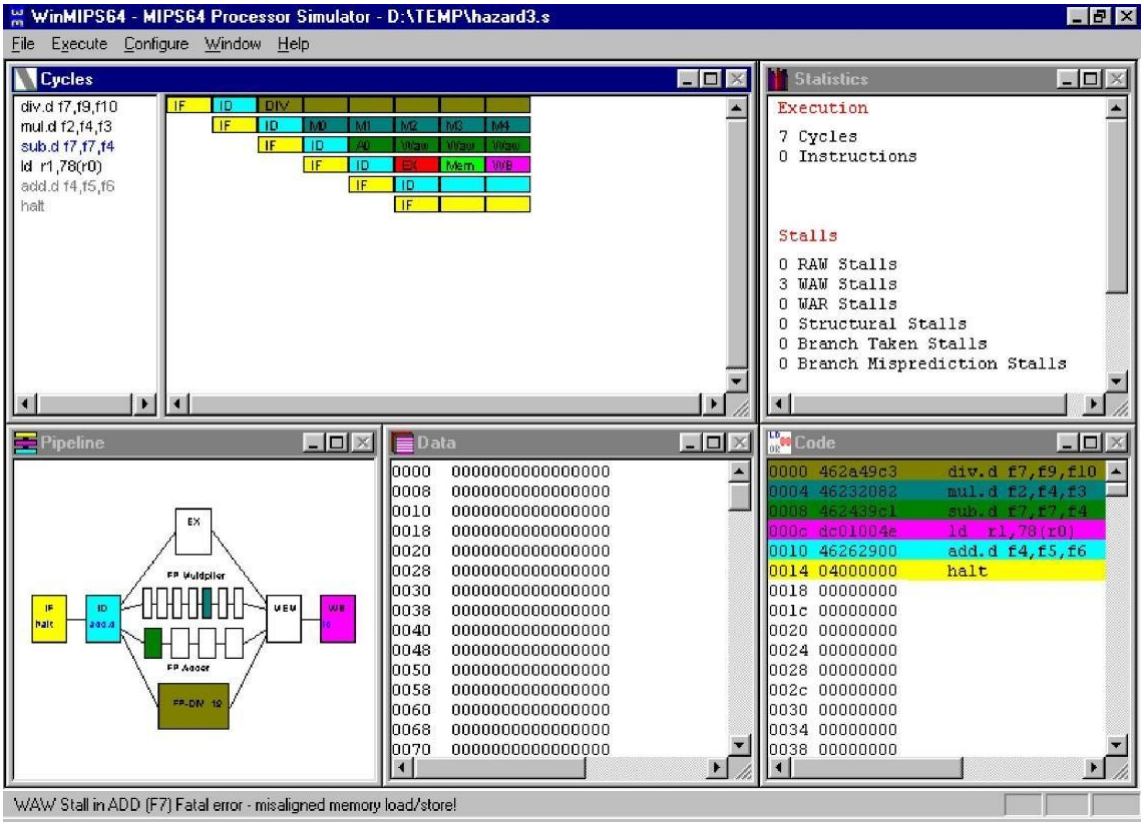

Рис.7. Перша інструкція ділення ще виконується, а третя від кінця ld майже завершилася

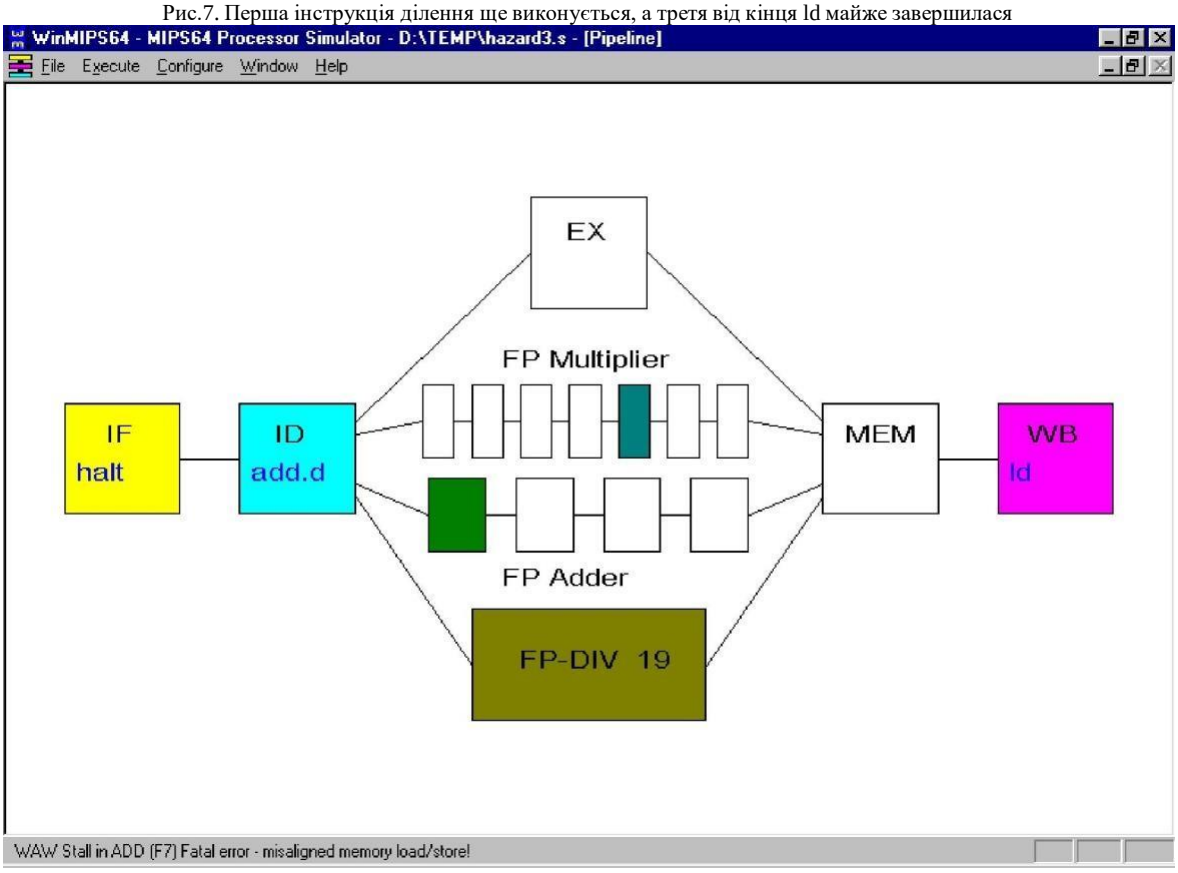

Рис. 8. Невпорядковане (кажуть, хаотичне) завершення виконання інструкцій потоку

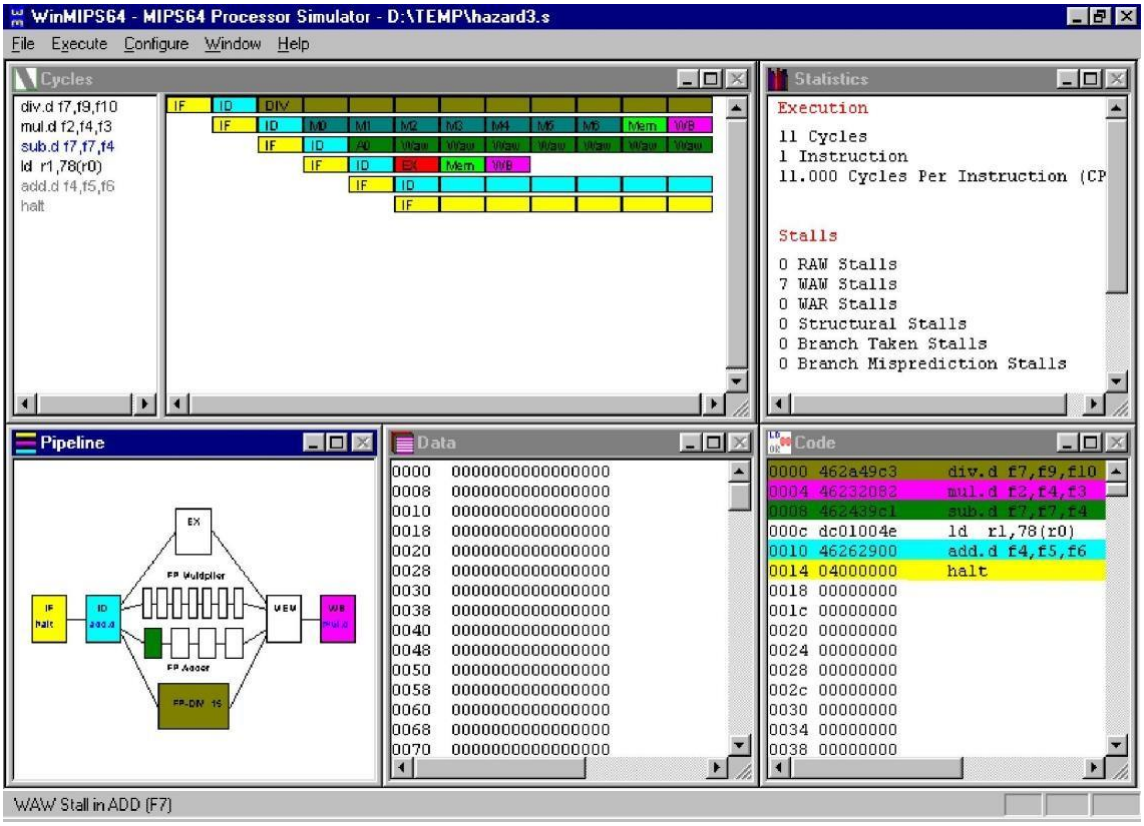

Рис. 9. sub.d має ту саму мету, що і div.d. Утворилася ситуація WAW, тому пригальмовані add.d та halt

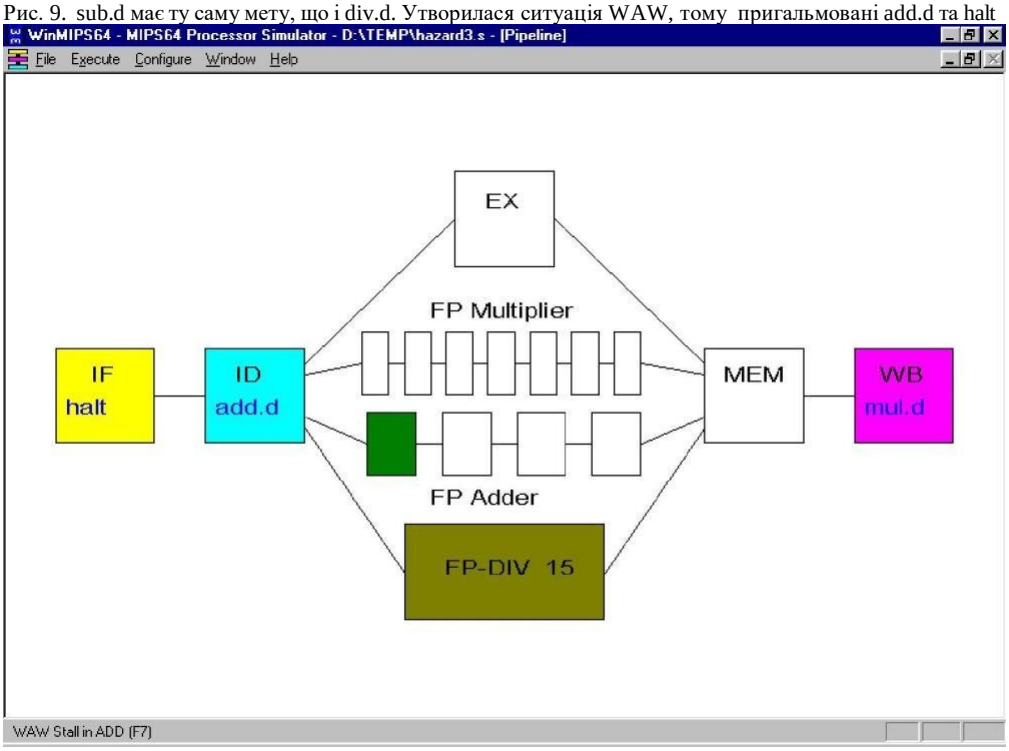

Рис. 10. Пригальмована на першій з чотирьох виконавчих сходинок sub.d *гальмує наступнуза неюadd.d*

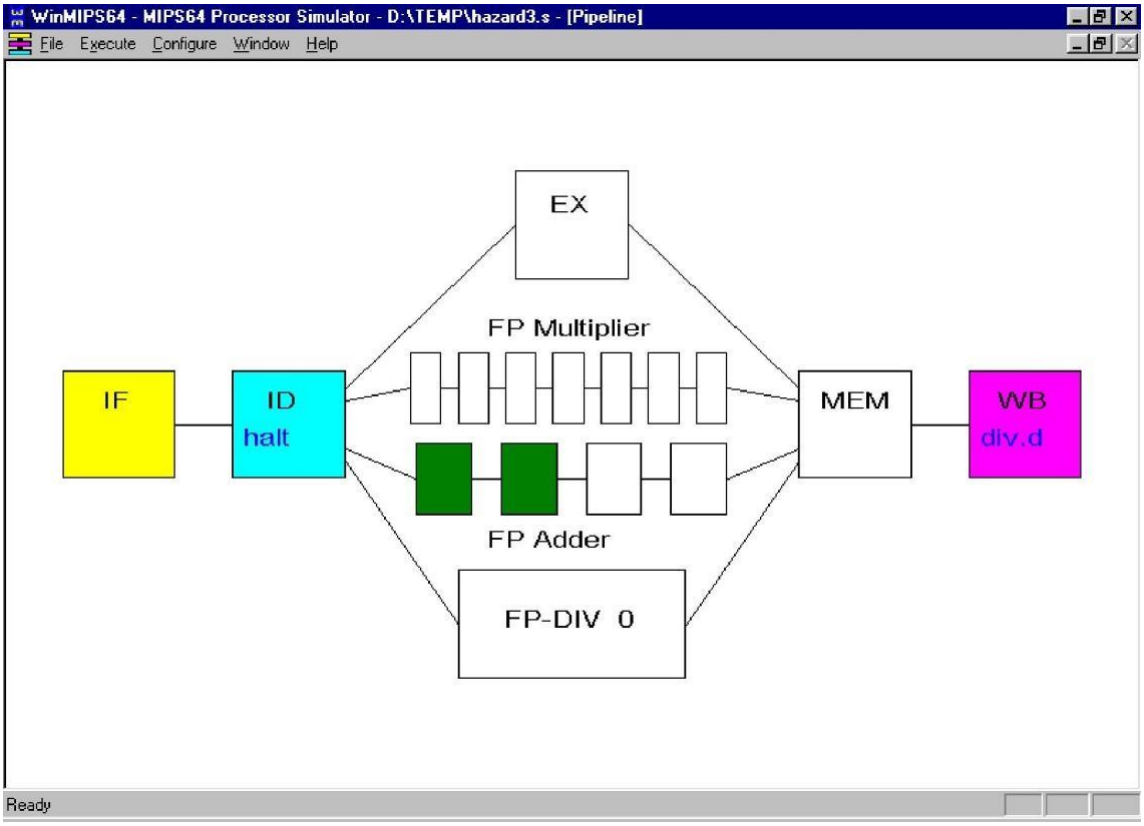

Рис. 11 Завершення div.d розблокувало конвеєр. На виконанні знаходяться віднімання і додавання

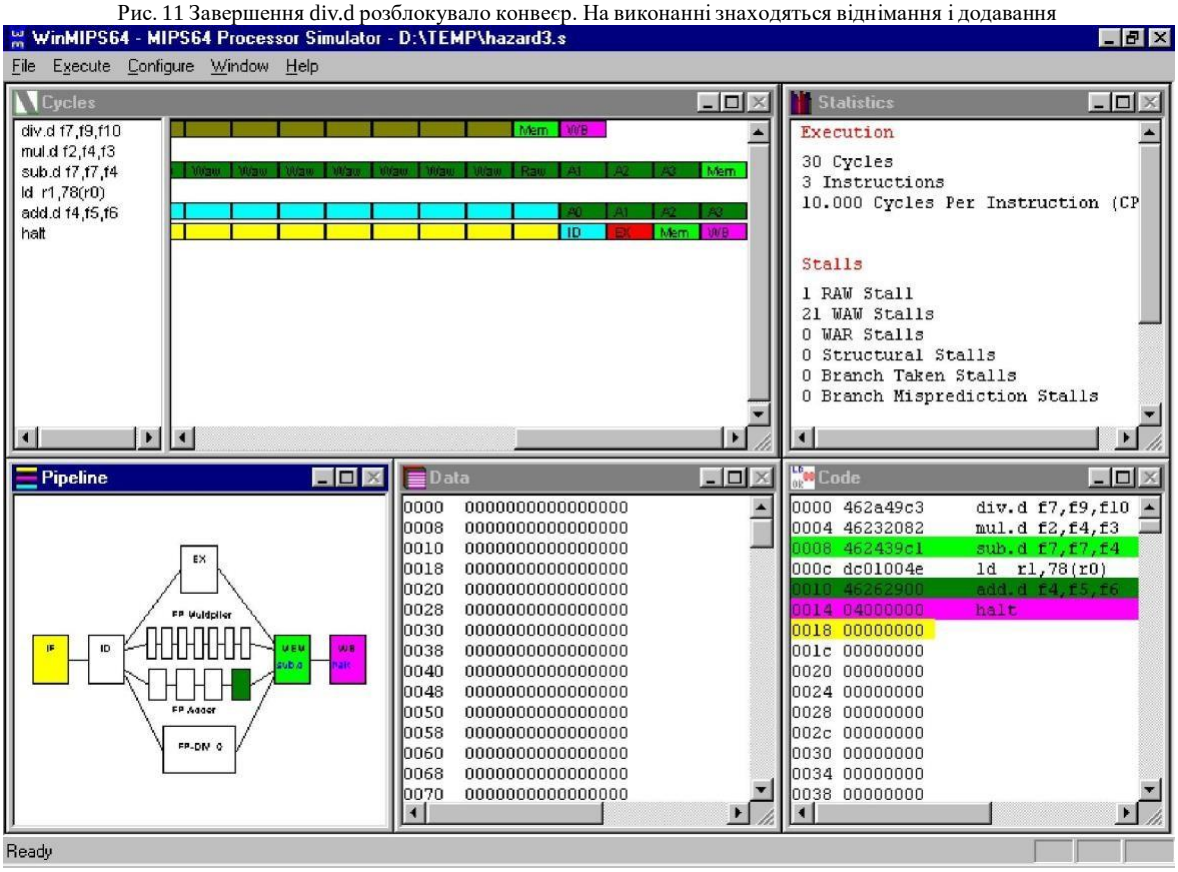

Рис. 12 Хаотично завершуються спочатку halt, потім рухомі віднімання та додавання

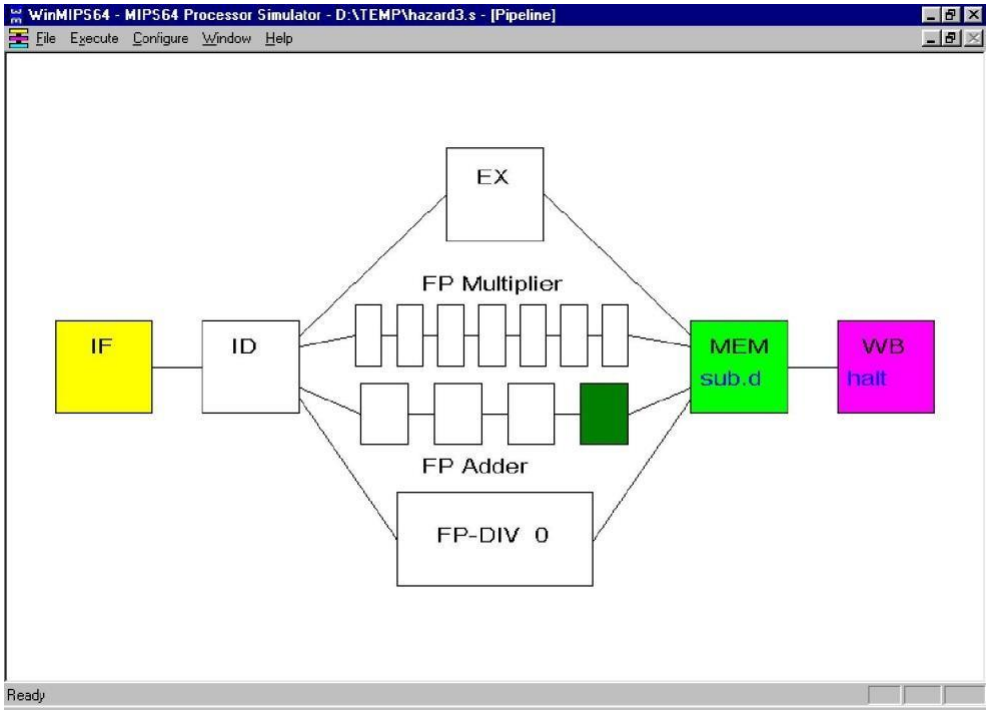

Рис. 13 Хаотичне завершення виконання інструкцій потоку (передтеча методу динамічного виконання)

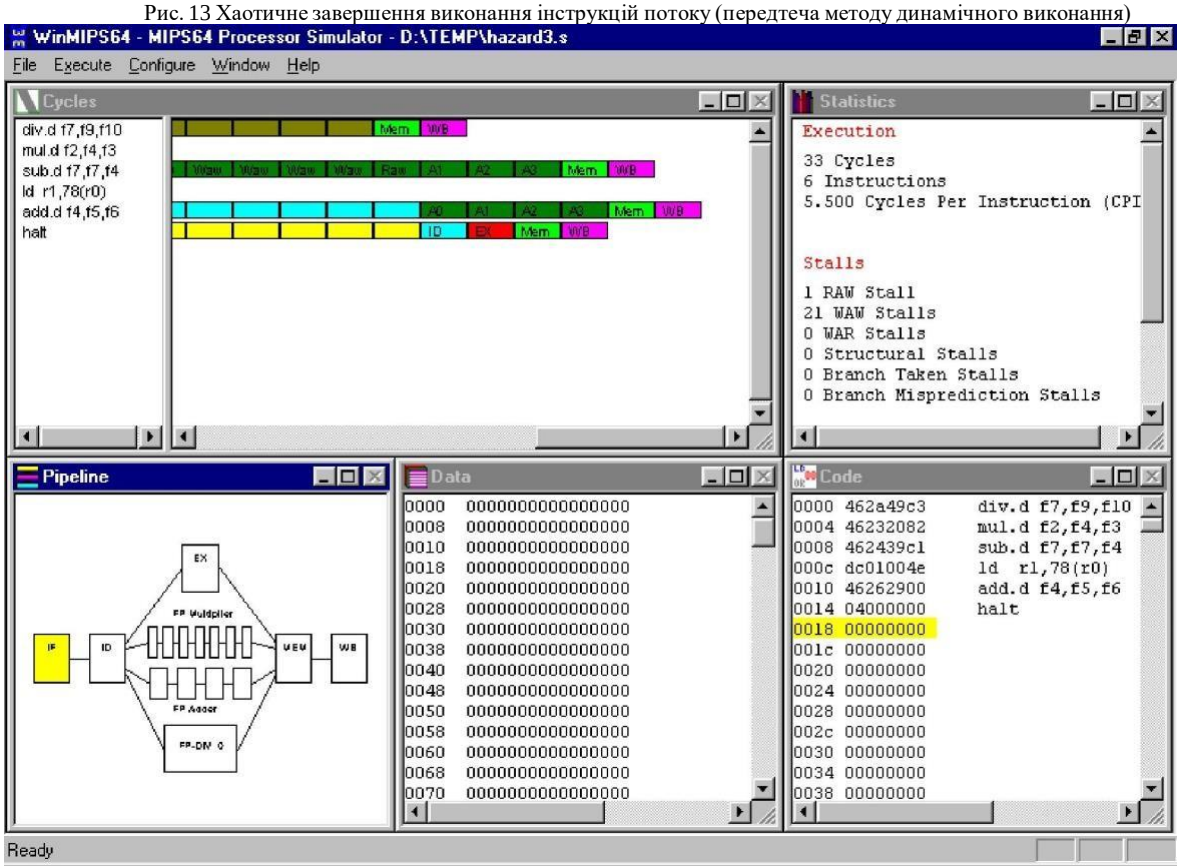

Рис. 14 Програму виконано, отримано програмні статистики

Варіанти завдань

Персональні варіанти завдань наведено у відповідній таблиці згідно з розподілом варіантів між студентами.

Для виконання лабораторної роботи необхідно:

1. **Реалізувати два варіанти одного і того ж обчислювального фрагмента**:
    - **Неоптимізований код** — з типовими проблемами для конвеєрного виконання:
        - наявність залежностей типу RAW, WAR, WAW;
        - конфлікти при доступі до регістрів або пам’яті;
        - необґрунтоване дублювання інструкцій або їх нераціональне розташування.
    - **Оптимізований код** — з урахуванням:
        - усунення або зменшення затримок (hazards);
        - перестановки інструкцій (reordering);
        - техніки переназви регістрів (register renaming);
        - скорочення кількості циклів без зміни логіки обчислень.
2. **Типи завдань можуть включати** (залежно від варіанту):
    - обчислення арифметичних виразів з кількома операндами;
    - виконання простих циклів;
    - обробку масивів (сума, добуток, пошук максимуму тощо);
    - обчислення математичних функцій (наприклад, факторіал, степінь, середнє арифметичне);
    - імітацію завантаження/збереження даних з пам’яті.

Важливо: Обидва варіанти коду повинні давати однаковий результат, проте мати різну ефективність з точки зору конвеєрного виконання (CPI, кількість затримок, тощо).