ЛР № 7. Дослідження виконання циклів на конвеєрі інструкцій

Мета: Опанувати техніку конвеєрного виконання RISC інструкцій.

Завдання: Засобами архітектурного симулятора WinMIPS64 машини з 64-розрядною RISC архітектурою MIPS64 дослідити конвеєрне виконання фрагментів машинних програм, що містять цикли. Виявити наявні залежності (небезпеки) даних і керування, оптимізувати програмний код та дослідити дію запропонованої оптимізації. За результатами проведених лабораторних досліджень оформити звіт та захистити його.

Методика виконання лабораторної роботи

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).

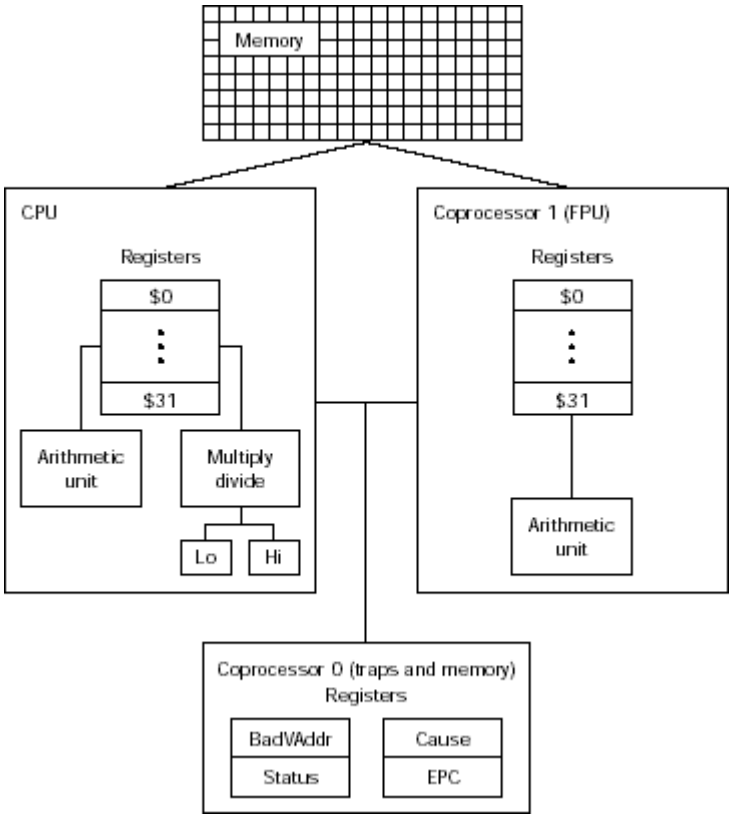

Рис. 7.1. Процесор MIPS R2000, CPU і FPU.

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).

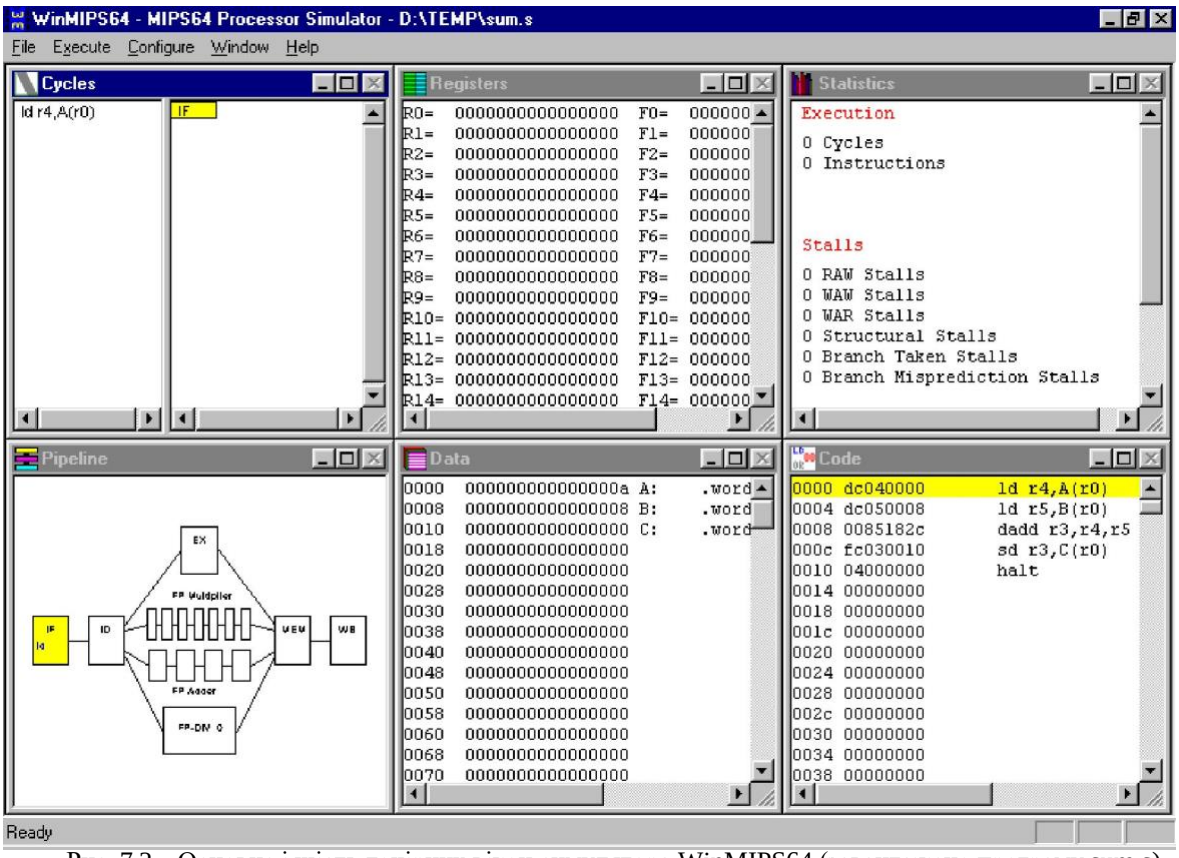

Рис. 7.2 – Основне і шість дочірних вікон симулятора WinMIPS64 (завантажено програму sum.s)

Далі подамо назви і пояснимо призначення дочірніх вікон симулятора (Таблиця 7.1).

Таблиця 7.1 – 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", проте протягом симулювання подає

	різну корисну користувачеві інформацію щодо поточного стану процесу симулювання.

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 численихопції функціонування цієї апаратури. Можна змінювати струкуктуру і час виконання рухомих операцій на конвеєрі, місткість пам'яті коду/даних. Аби зробити це чи побачити стандартні призначення архітектури викликати наступне вікно.

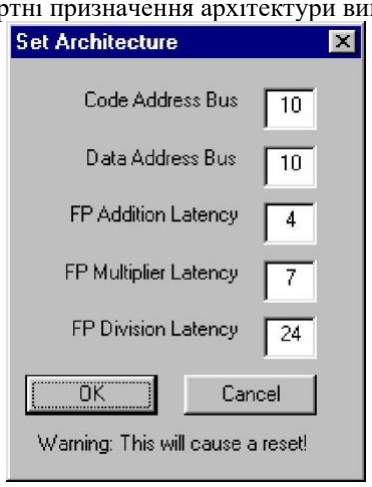

Рис. 7.3 – Модальне вікно конфігурування апаратної архітектури: місткість пам'яті коду і пам'яті даних, затримки виконання рухомих (FP) операцій

Latency – *прихована затримка виконання*, ніби "інкубаційний період хвороби". В останньому вікні зафіксовано розрядності (і відповідно, місткість) пам'яті даних і пам'яті програм. В нас вони дорівнюють 2^{10} = 1024 = 1K 32-х розрядних машинних інструкцій і 2^{10} = 1024 = 1K байтів даних. (Приховані) затримки виконання операцій рухомої коми (FP)складають 4 цикли (тактові інтервали) для рухомого додавання/віднімання (4 сходинки числового конвеєра додавання/віднімання), 7 циклів для рухомого множення (7 сходинок ще одного, паралельного до конвеєра додавання/віднімання, числового конвеєра множення) і 24 цикли для неконвеєрного пристрою ділення кодів з рухомою комою. Всі операції для фіксованої коми виконуються за один цикл, тобто, не мають латентних (прихованих) затримок. Будь-які зміни архітектури спричинюють скид симулятора! Достатися до цього модального вікна можна через меню *Configure*/*Architecture* (читати так: "клікнути" на пункті меню *Configure*; щиголь відкриє ще одне меню, де треба щигольнути пункт *Architecture*).

Ще три опції можна вибрати через пункт меню *Configuration*, а саме, *Multi-Step (декілька кроківодним натиском), Enable Forwarding (дозволити випередження)* і *Enable Delay Slot (дозволити слот затримки переходу),*

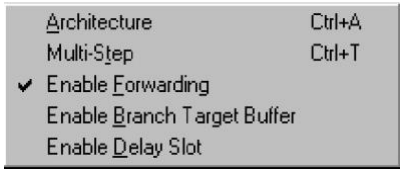

Рис.7.4 Опції: пункт меню головного вікна "Конфігурування"(дозволене лише випередження) тобто, завжди виконуватиметься інструкція за інструкцією умовного переходу, незалежно відтого, зроблено (taken) перехід чи не зроблено (not taken). Коли опцію обрано, поруч з неюз'являється сіра "гава".

Завантаження програми

Користуйтеся стандартним редактором NotePad, аби створити програмний файл, наприклад, sum.s. Ця програма містить код для MIPS64, що вираховує суму двох цілих чисел A і B.

Числові коди спочатку вибрати з комірок пам'яті за адресами A і B до регістрів, потім додати на регістрах і, нарешті, записати отриману суму до комірки пам'яті за адресою C. Далі йде текст асемблерної програми.

```
;*** winMIPS64 //sum.s//   C=A+B    *****
;*** (c) 2003 CA226, DCU                  *****

      .data
A:    .word 10
B:    .word 8
C:    .word 0
      .text
main:
      ld  r4,A(r0) ld
            r5,B(r0)
      dadd r3,r4,r5sd
            r3,C(r0) halt
```

Зауважимо, що невеличка за розміром утиліта asm.exe дозволяє ще до симулювання перевірити синтаксис програми, що мають симулювати. Аби перевірити синтаксис потрібно виконати команду операційної системи:

C:\winmips64> asm sum.s

Результат побачимо на дисплеї. Коли виконатиC:\winmips64> asm sum.s > report.txt

тоді результат вміщуватиме не екран, а файл report.txt, вмістиме якого для програми sum.s заразі подамо.

```
Pass 1 completed with 0 errors
                  ;*************************************
                  ;*** winMIPS64 //sum.s//        ***
                  ;*** (c) 2003 CA226, DCU                 ***
                  ;*************************************

      00000000                .data
      00000000 000000000a A:        ! 10
      00000008 0000000008 B:        ! 8
      00000010 0000000000 C:        ! 0

00000000                      .text
00000000                main:
00000000 dc040000             ld r4,A(r0)
00000004 dc050008             ld r5,B(r0)
00000008 0085182c             dadd r3,r4,r5
0000000c fc030010             sd r3,C(r0)
00000010 04000000             halt

Pass 2 completed with 0 errorsCode
      Symbol Table
main = 00000000Data Symbol Table
                  A = 00000000
                  B = 00000008
                  C = 00000010
```

До старту симуляції, потрібно завантажити до пам'яті симулятора синтаксично перевірену програму. Це виконують через меню *File*/*Open*. По завершенню можна побачити оновлене вмістиме вікон коду і даних (останнє, коли програма містить дані або розміщує результати в пам'яті. Нагорі вікна симулятора з'являється рядок зі шляхом до програми, в нас – до програмиsum.s.

Аби завантажити програмний файл до симулятора WinMIPS64 потрібно виконати наступне:Click on sum.s.

Click the *Open* button

Тепер програма завантажена до пам'яті і симулятор готовий до функціонування.

Коди спостерігають у вікні кодів. Першозначення і зміни вмістимого комірок і регістрів спостерігають через вікна даних і регістрів.

Симулювання

В будь-якому стані симулятора потрібно натиснути *F10*, аби розпочати новий цикл симулювання. Зауважимо, що на старті перша кольорова лінія в вікні Code з адресою 0x0000 (0000) забарвлена жовтим. Сходинка конвеєра IF у вікні Pipeline також забарвлена жовтим і містить асемблернумнемоніку першої інструкції програми. Поглянемо до кодового вікна Code І знайдемо тампершу інструкцію ld r4,A(r0). У вікні Data можна знайти програмну змінну A.

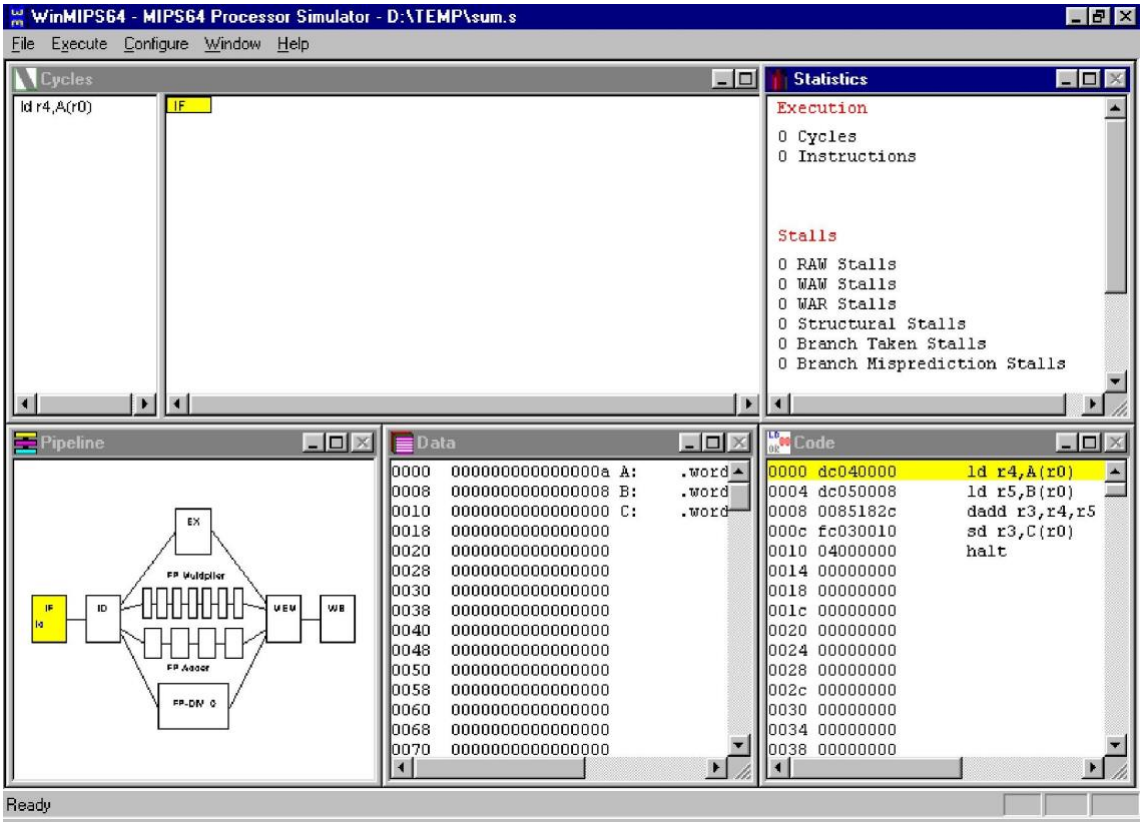

Рис. 7.5 Головне вікно симулятора з завантаженою програмою sum.s, стан – до початку виконання

Clock 1

Натиснути *Execute/Single Cycle* (або натиснути *F7*). Симулятор виконає перший цикл (тактовий інтервал) програми. Увага! Не першу інструкцію, а лише перший цикл, що для п'яти сходинкового конвеєра відповідає виконанню 1/5 першої інструкції в найкращому випадку, коли нема вимушених затримок конвеєра. Зауважимо, що абревіатура MIPS відповідає в українському перекладі "мікропроцесору, що немає затримок конвеєра", а не "мільону операцій на секунду". У вікні Code колірність першої інструкції змінюється на блакитну і вже друга інструкція забарвлена жовтим. Таке забарвлення вказує на сходинку конвеєра, де знаходиться відповідно забарвлена інструкція:

жовтий для IF (вибирання інструкції з пам'яті інструкцій, точніше з кеша інструкцій), блакитний для ID (декодування інструкції і вибирання операндів безпосередніх і зрегістрового файла),

червоний для EX (виконання інструкції, 1 такт для фіксованих кодів і більше тактів для рухомих кодів),

зелений для MEM (запис/читання до/з комірок пам'яті даних), пурпурний для WB (запис результату до регістру регістрового файлу).

Коли подивитися на сходинку конвеєра IF у вікні Pipeline , можна побачити, що друга інструкція програми ld r5, B(r0) з'явилася на цій, в той час, як перша інструкція ld r4,A(r0) пересунулася на другу сходинку ID конвеєра. Конвеєр розпочав функціонування.

Clock 2

Наступний натиск на *F7* змінює забарвленість в вікні Code через уведення червоного коліру для третьої сходинки EX конвеєра. До конвеєра увійшла інструкція dadd r3,r4,r5.

Clock 3

Третій натиск на *F7* азнову змінює колірність вікна Code, іуводячи зелену забарвленість для четвертої сходинки MEM конвеєра. До конвеєру входить інструкція sd r3,C(r0). У вікні Cycle бачимо історію виконання кожної інструкції програми, тобто сходинку, на якій знаходиться кожна інструкція перед надсиланням чергового тактового імпульса.

Clock 4

Знову натискаємо *F7*. Кожна сходинка конвеєра отримує нову інструкцію. Регістр r4 має завантажуватися з комірки пам'яті даних. Проте оновлене вмістиме регістра можна отримати

завчасно (кажуть – з випередженням в часі, прямо з виходу пам'яті даних, тобто, з виходу сходинки MEM (for *forwarding* from the MEM stage)). Отже, прямо з виходу пам'яті забираємо новий, ще не записаний до регістру призначення операнд т(це і є випередження), а далі паралельно виконуємо додавання цього нового операнду і в той самий час пишемо цей операнд до регістру регістрового файлу. Добре, що в конвеєрі водночас знаходиться черга з п'ятьох інструкцій, тобто, всі ці інструкції є «під рукою».

Можна бачити, що r4 забарвлюється зеленим (колір сходинки MEM) у вікні Registers. Зауважимо, що остання інструкція програми halt (симулятор не має навіть примітивної програми монітора, що вже казати про операційну систему; отже, програміст мусить вручну гальмувати виконання програми саме такою інструкцією) вже уведена до конвеєра.

Clock 5

Натискаємо *F7*. Тут відбуваються цікаві події. The value destined for r5 becomes available for forwarding. However the value for r5 was not available in time for the dadd r3,r4,r5 instruction to execute in EX. So it remains in EX, stalled. The status line reads "RAW stall in EX (R5)", indicating where the stall occurred, and which register's unavailability was responsible for it.

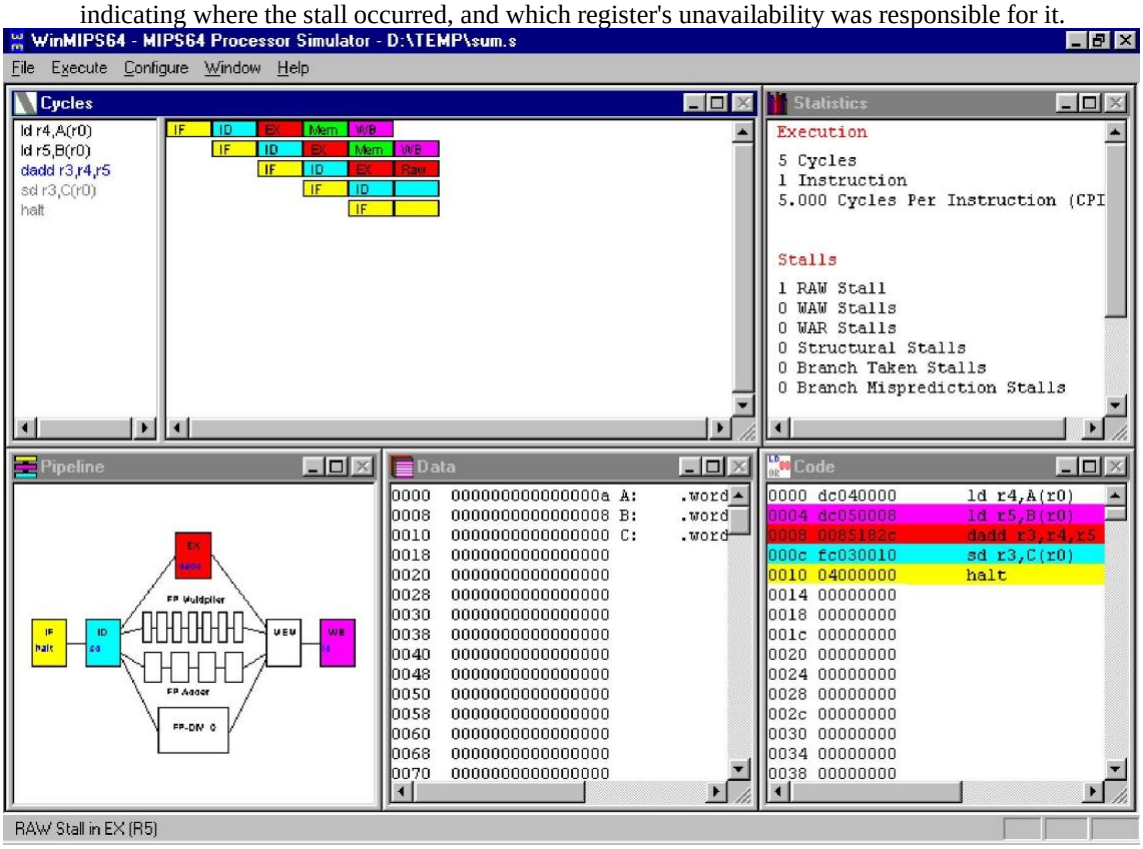

Рис. 7.6. Такт виконання програми. Статус: відбулося пригальмування RAW (read after write) Фігури в вікнах Clock Cycle Diagram і Pipeline чітко вказують на те, що інструкція dadd"застрягла" в сходинці EX, і що всі інструкції за нею також неспроможні пересуватисяконвеєром далі. У вікні Clock Cycle Diagram інструкцію dadd підсвічено блакитним, а інструкції,що розташовані за нею "посіріли".

Clock 6

Натискаємо *F7*. Інструкція dadd r3,r4,r5 instruction нарешті виконується, а щойно отримана сума, призначена для запису до r3, паралельно стає досяжною для виконання випередження (зі сходинки execute). Отримано значення 0x12, що є сумою 10+8 = 18 в десятковій. Це і є відповідь програми, а нас більше цікавить не ця відповідь, а часові витрати на виконання програми (вимірюємо числом тактових імпульсів) і скорочення цього часу різними методами. Щойно зауважене і є предметом курсу з архітектури комп'ютера.

Clock 7

Натискаємо F7. Інструкція halt, вже уведена до конвеєра, спричинює ефект "закриття" конвеєра. По ній вже жодна інша інструкція не уводиться до конвеєра, а сам конвеєр поступово спорожнюється, він сходинки IF до сходинки WB.

Clock 8

Натискаємо *F7*. Перевіряємо пам'ять даних Data , де фіксуємо, що змінна C набула значення 0x12. Інструкція sd r3,C(r0) записала це значення до пам'яті на сходинці MEM конвеєра, використовуючі випереджені дані з виходу IF, які вже нормально, без випередження записуються до регістра r3.

Clock 9

Натискаємо F7.

Clock 10

Натискаємо F7. Програма фінішує.

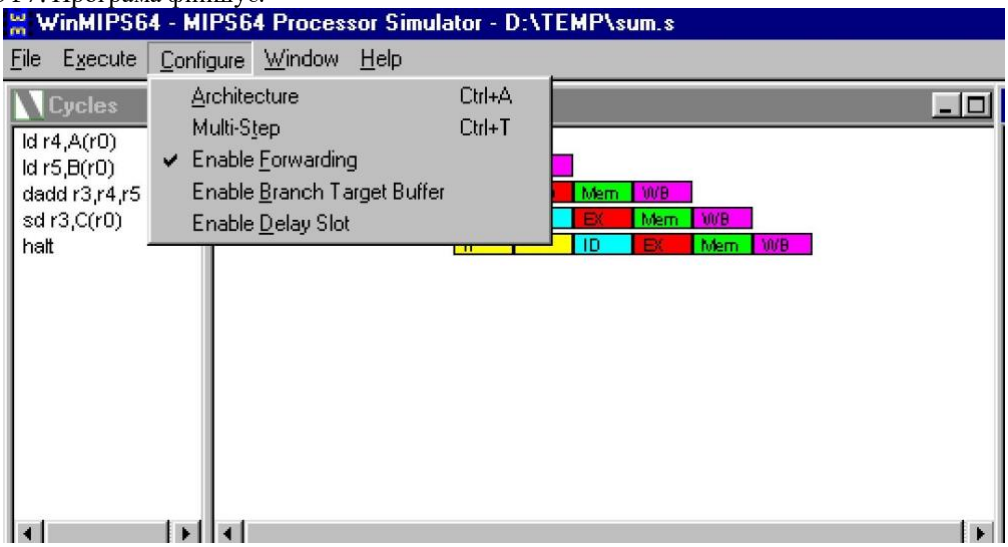

Рис. 7.7. Дозвіл випередження ще до виконання програми sum.s

Зараз проаналізуємо вмістиме вікна статистики (Statistics) і зауважимо, що зафіксовано 2 loads та 1 store. Це є "тяжкі" інструкції. Ще ми мали 1 пригальмування RAW. Витратили 10 тактових циклів на виконання п'ятьох інструкцій. Значить, отримали середнє число тактових імпульсів CPI=2 на одну інструкцію. Це в два рази гірше від ідеалу і тут є де вдосконалюватися. Проте, отримане вже є добрим результатом.

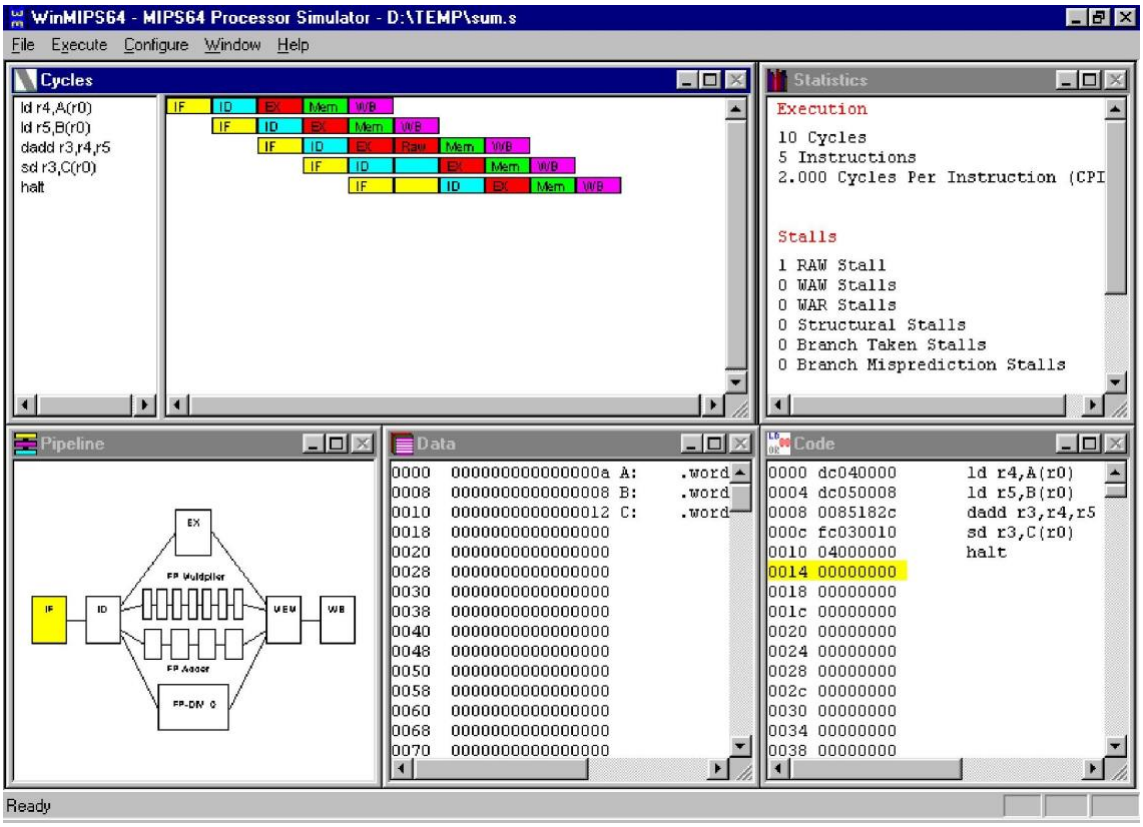

Рис. 7.8. Головне вікно симулятора по завершенню виконання програми sum.s

Вмістиме вікна статистики негайно відгукується на будь-які зміни в архітектурі. Дослідимо негативний вплив на продуктивність заборони випередження (що є наслідком спрощення апаратної частини, а дешеве не буває оптимальним). Який час нам знадобиться без використання випередження? Аби дізнатися це активуємо *Configure*. Для заборони випередження (forwarding) треба "клікнути" на *Enable Forwarding* (гава дозволу має зникнути).

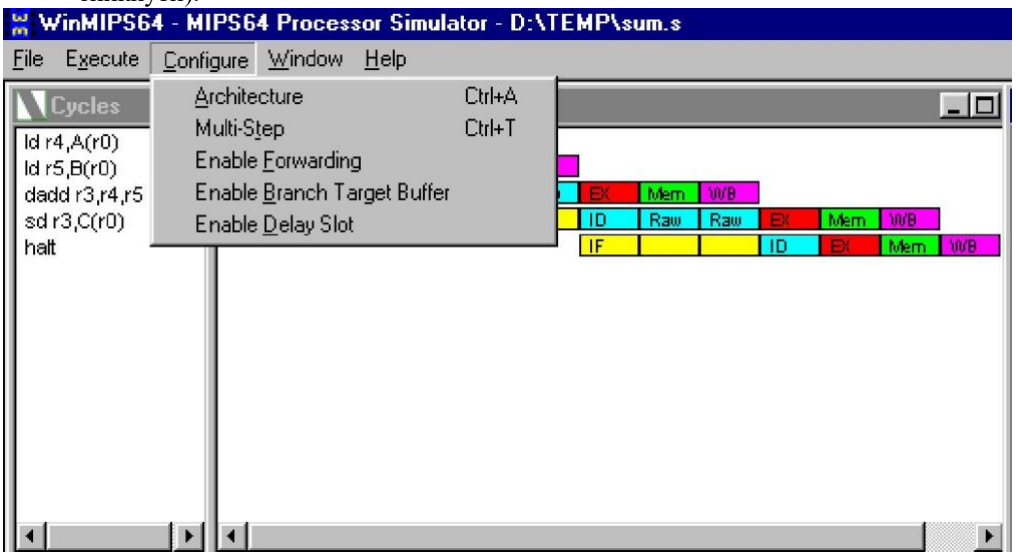

Рис 7.9. Заборона випередження

По виконанню нашої програми, але вже без випередження,з проаналізуємо вмістиме вікна статистики (Statistics). Зауважимо, що маємо 2 loads та 1 store і аж 4 пригальмування класу RAW(додатково втрачено 4-1=3 такти). Отже програма "з'їла" 10+3=13 тактових циклів на виконанняп'ятьох інструкцій, а середнє число тактів на виконання інструкції погіршилося і становить CPI=2.600. Отже, продуктивність спрощеної версії процесора є нижчою.

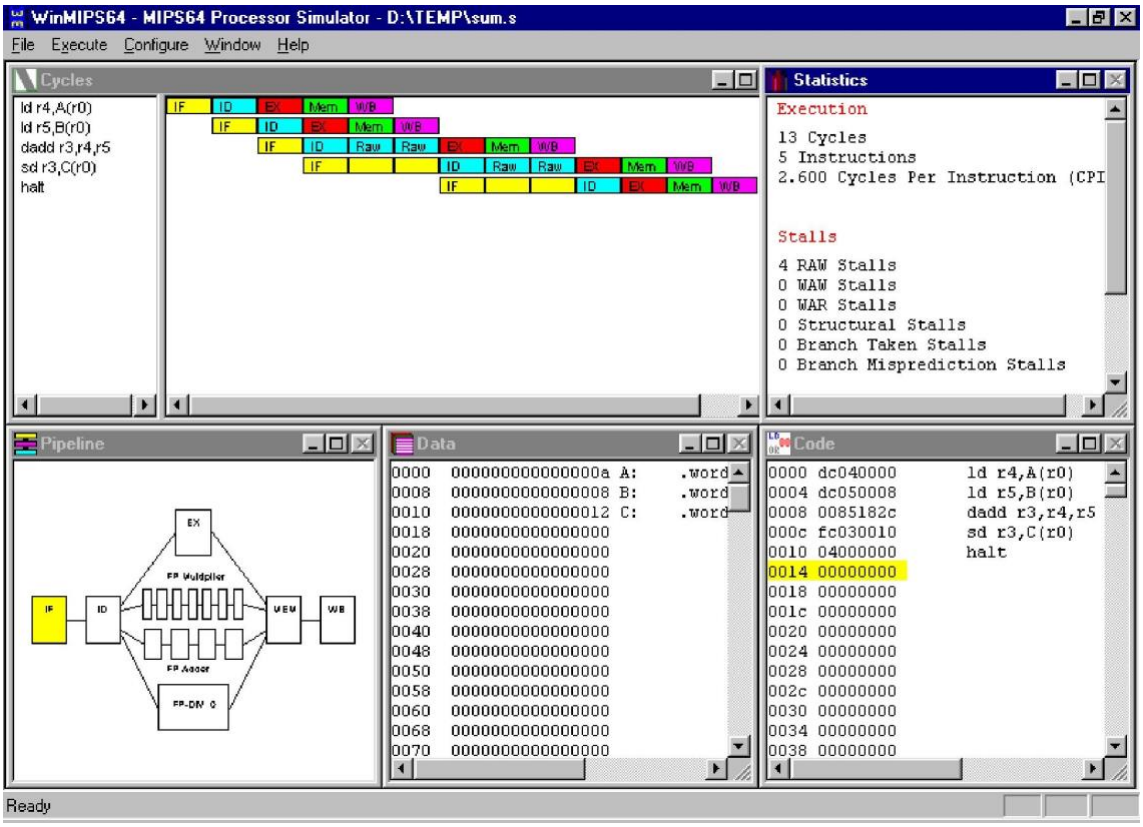

Рис 7.10. Головне вікно симулятора по завершенню виконання програми sum.s без використання випередження

Repeat the cycle-by-cycle program execution, re-examine the Statistics window and compare the results. Note that there are more stalls as instructions are held up in ID waiting for a register, and hence waiting for an earlier instruction to complete WB. The advantages of *forwarding* should be obvious.

3.2. Інші режими симулювання

Активуємо *File/Reset MIPS64*. Коли натиснути *File/Full Reset*, тоді ми спорожнимо пам'ять, але можемо повторно завантажити ту чи іншу програму. Натиском на *File/Reload* чи на *F10* можна рестартувати симуляцію.

За один крок можна виконувати більше від одного такту. Аби зробити це, викликають *Execute/Multi cycle...* Число виконаних одним натиском тактів змінюють через *Configure/Multi- step.* Повне виконання програми спричинює натиск на *F4* або ж на *Execute/Run to.*

Можна скористуватися точками зупинки. Спочатку натиснемо на *F10*. Аби встановити/зняти точки зупинки, зробимо лівий подвійний щиголь мишею, наприклад на інструкції dadd r3,r4,r5.Далі натиснемо на *F4*. Програма розпочне виконуватися в автоматичному режимі і загальмується. Коли розпочнеться фаза (сходинка) IF позначеної інструкції подвійного додавання. Ще один подвійний щиголь на тій самій інструкції знімає точку зупинки.

Приклад завдання з циклом

Нехай симулюється наступна програма, записана мовою асемблер. Програма знаходить суму десяти чисел (від 1 до 10 з кроком 1, відповідь є 0x37 = 55 десяткове). Ясно, що в пам'яті програмно резервують (.word) з ініціалізацією 10 комірок з доданками, а також (без ініціалізації,
.space) місце для результату (суми).

```
; Program: loop.s
; Sum of 10 integer values
        .data
values: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ;      64-bit integers
```

```
result: .space 8
        .text
MAIN:
         daddui R1,R0,10 ;  R1 <- 10
         dadd   R2,R0,R0 ;  R2 <- 0 POINTER REG
         dadd   R3,R0,R0 ;  R3 <- 0 RESULT REG
LOOP:
         ld     R4,values(R2) ;GET A VALUE IN R4
         dadd   R3,R3,R4 ;     R3 <- R3 + R4
         daddi  R2,R2,8 ;      R2 <- R2 + 8 POINTER INCREMENT
         daddi  R1,R1,-1 ;     R1 <- R1 - 1 DECREMENT COUNTER
         bnez   R1,LOOP
         nop
         sd     R3,result(R0) ; Result in R3
         HALT                 ; the end
```

Будемо користуватися процесором, де задіяні апаратні механізми випередження даними і прогнозування напрямку умовного переходу за допомогою буфера цільових адрес переходу (branch target buffer, BTB).

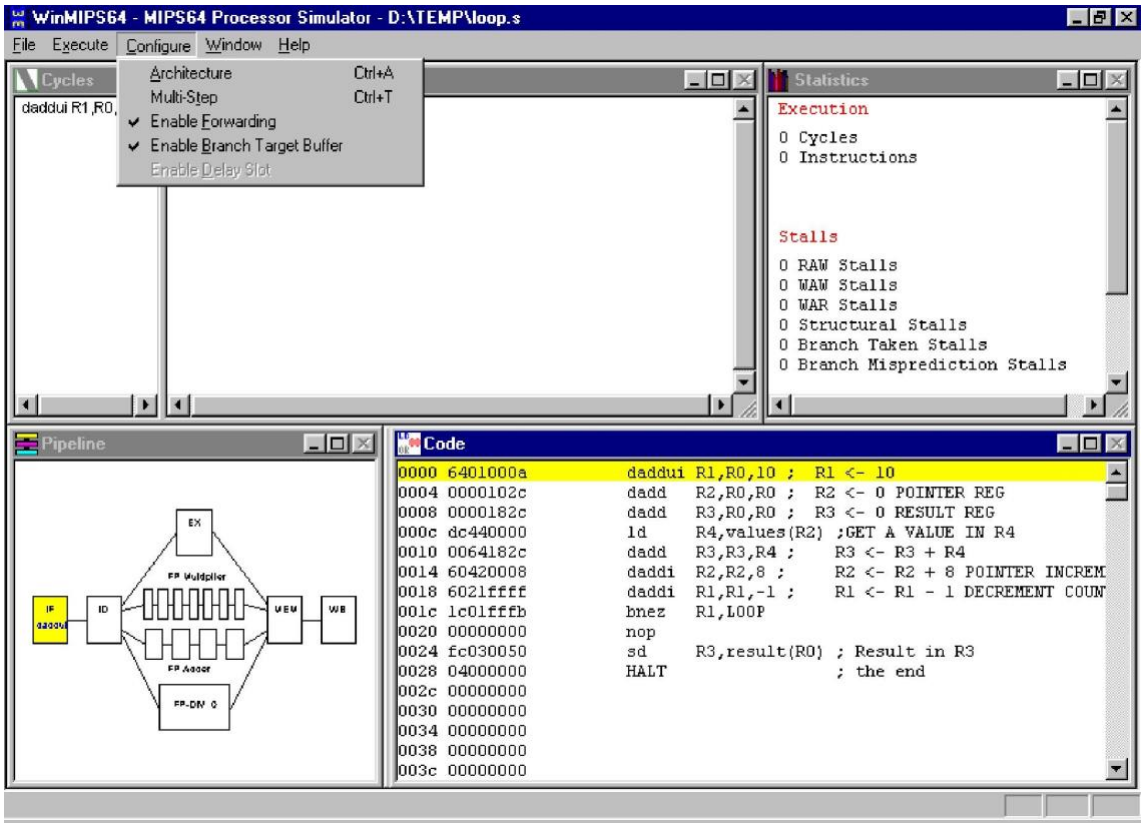

Рис. 7.11 – Основне вікно симулятора зі завантаженою програмою (одне дочірнє вікно не видно). Дозволено використання апаратури випередження (forwarding) і апартури передбачення напрямку умовного переходу (branch target buffer)

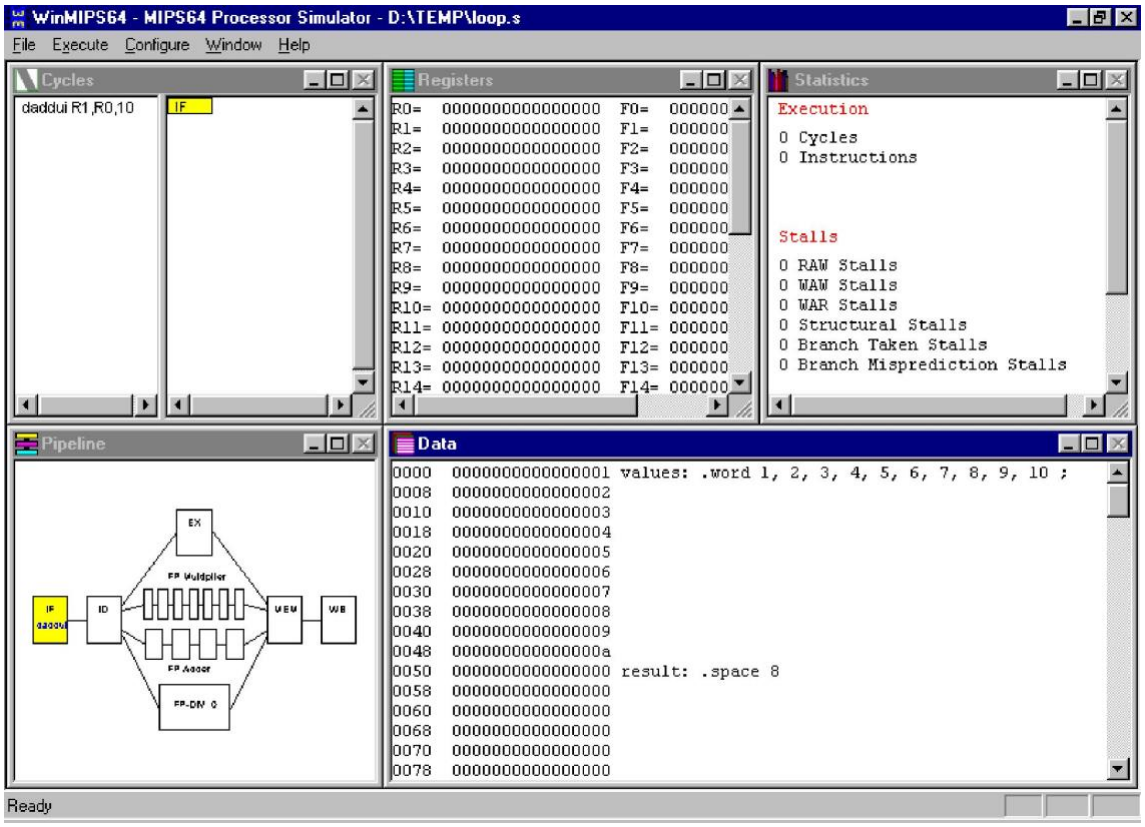

Рис. 7.12 Основне вікно симулятора, чітко видно вмістиме пам’яті до старту виконання програми

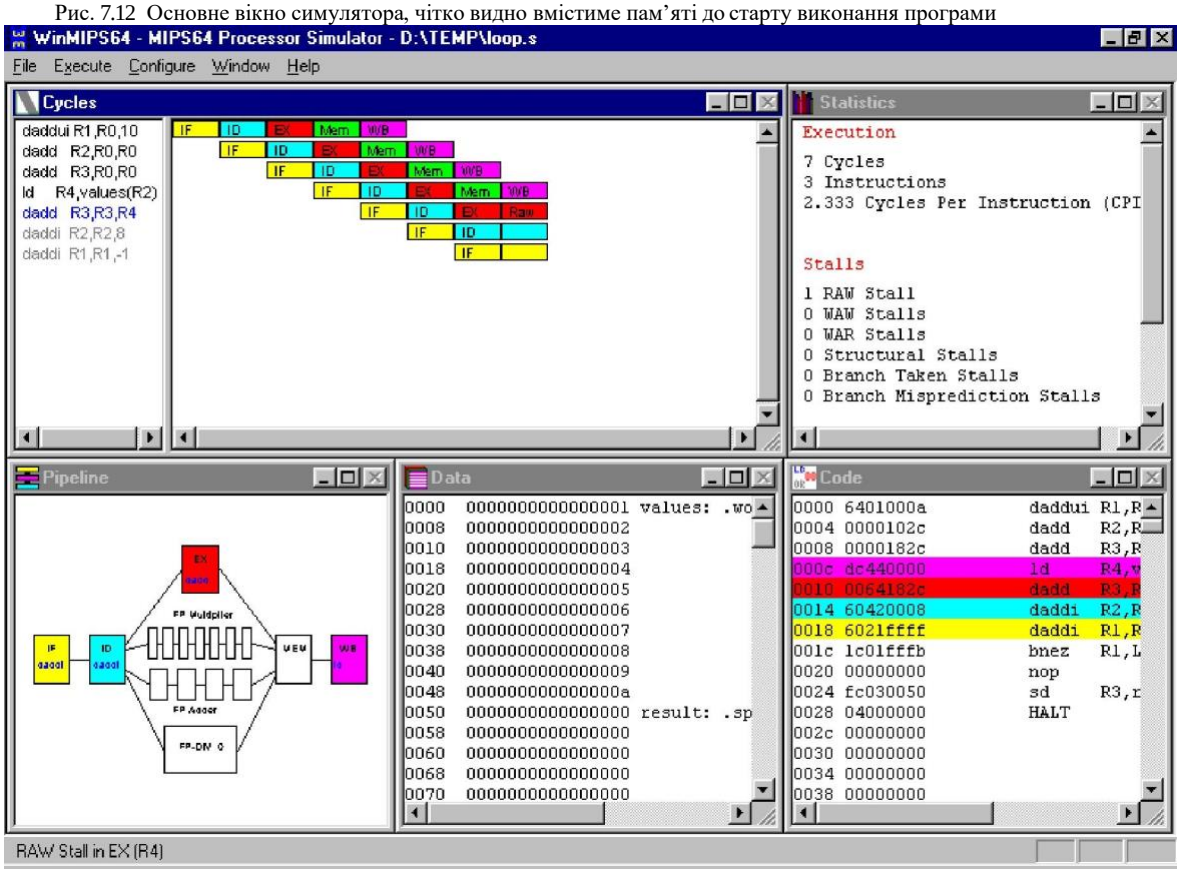

Рис. 7.13. Основне вікно симулятора, виконано 7 циклів симулювання. Є перше RAW (read after write)

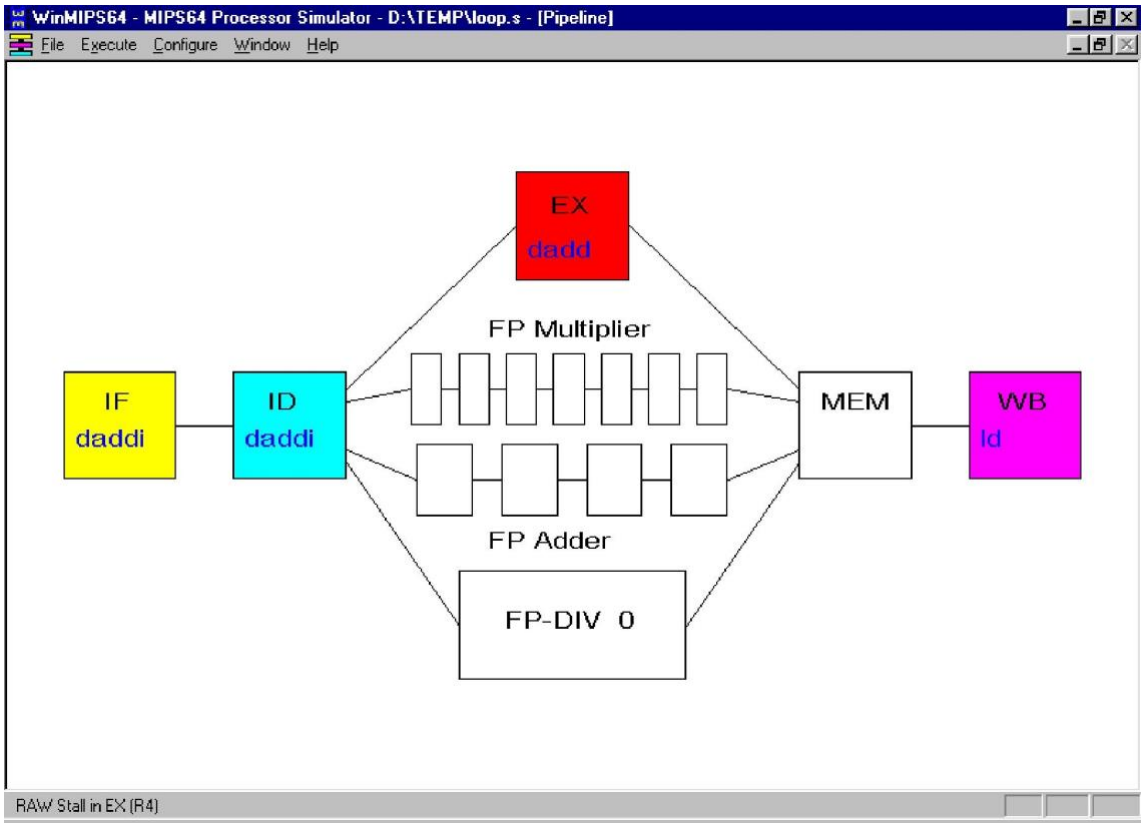

Рис. 7.14 Дочірнє вікно “конвеєр” симулятора, перше RAW пригальмування, 7 тактів симулювання

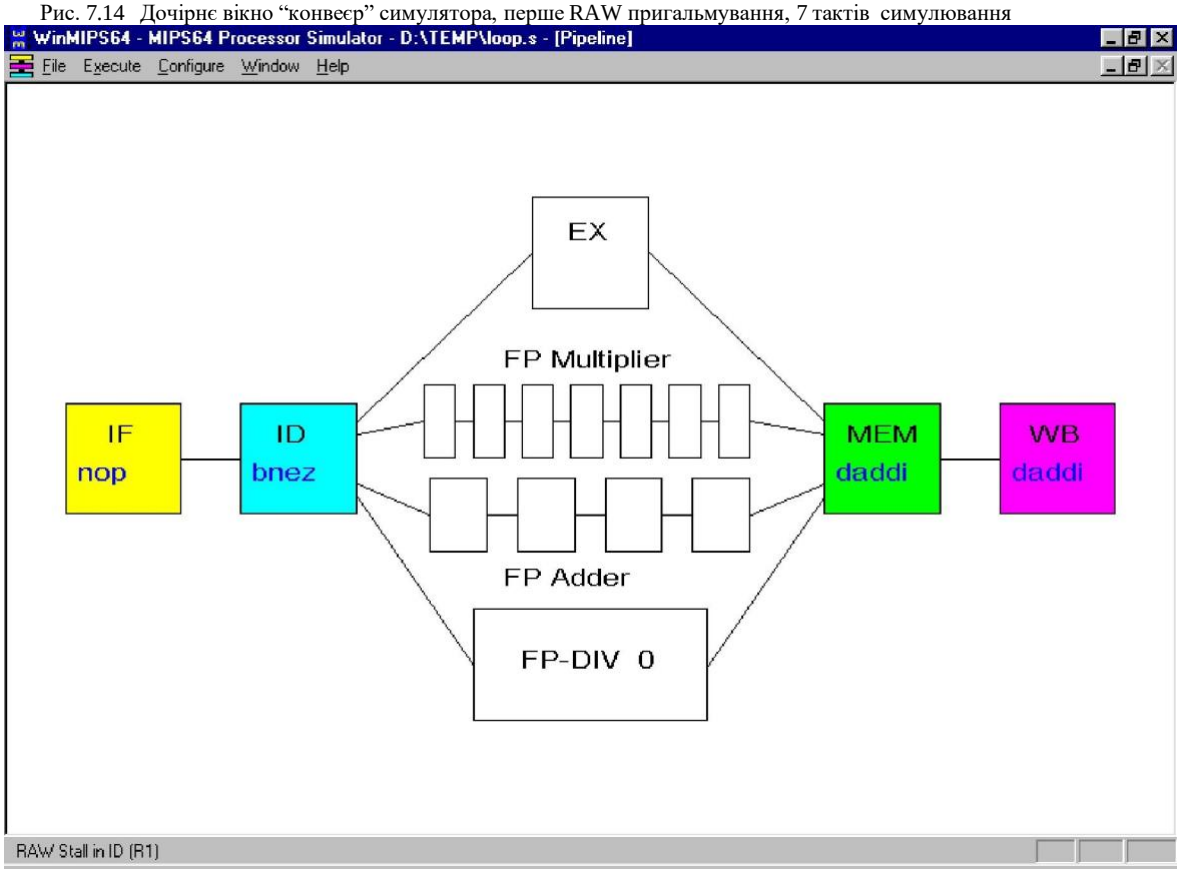

Рис. 7.15. Друге RAW пригальмування. Виконано 10 тактів

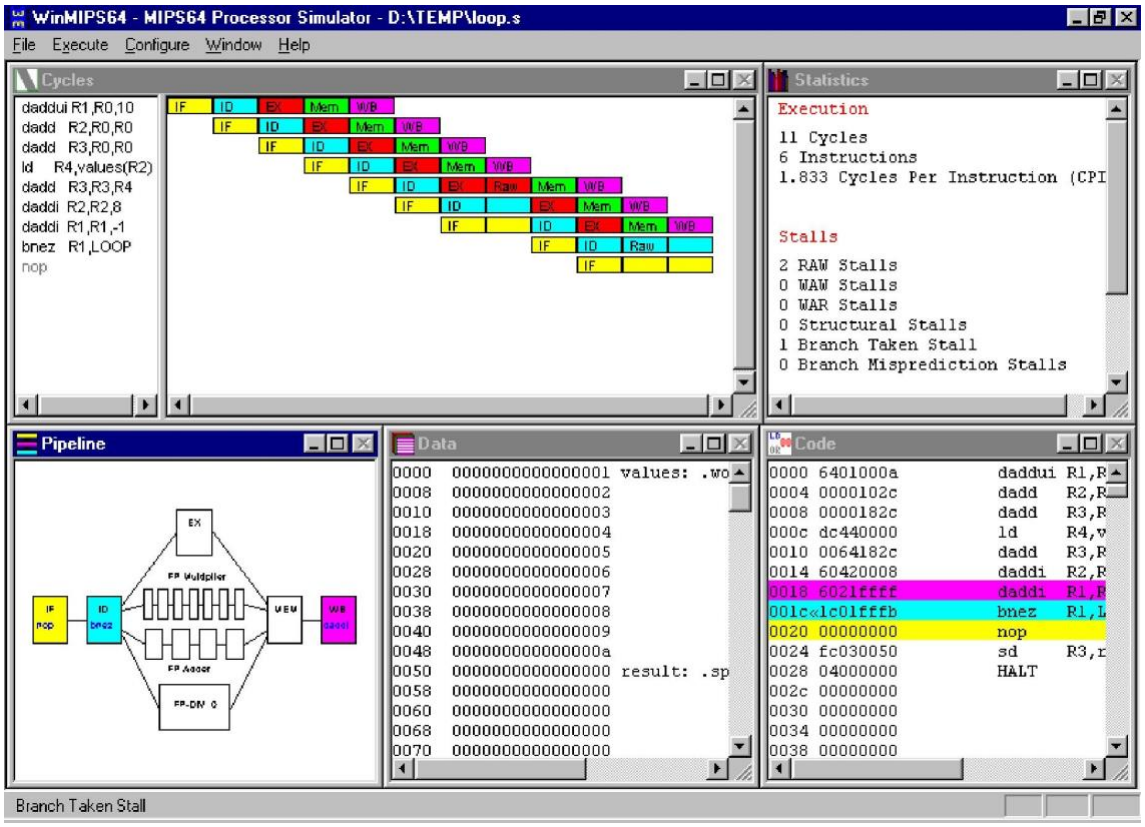

Рис. 7.16. Виконано 11 тактів симулювання, сталося перше пригальмування за рахуноквиконаного умовного переходу (branch taken stall)

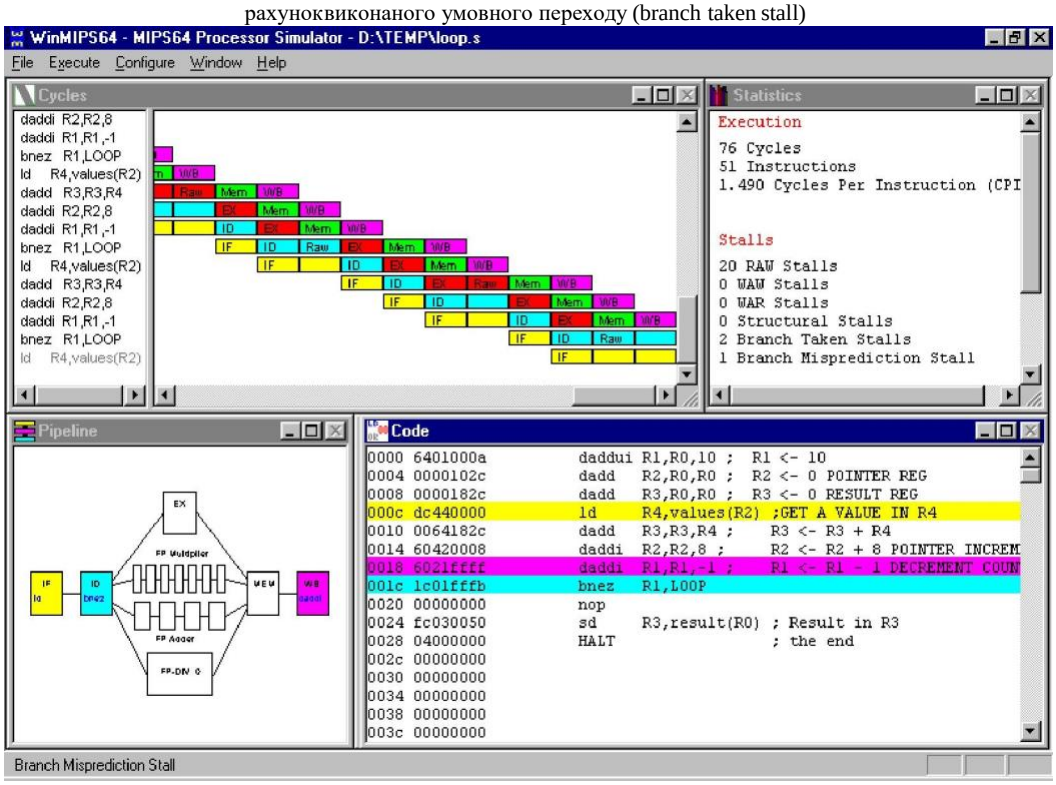

Рис.7.17 Стан виникнення першої помилки в передбаченні напрямку умовного переходу(Branch misprediction stall)

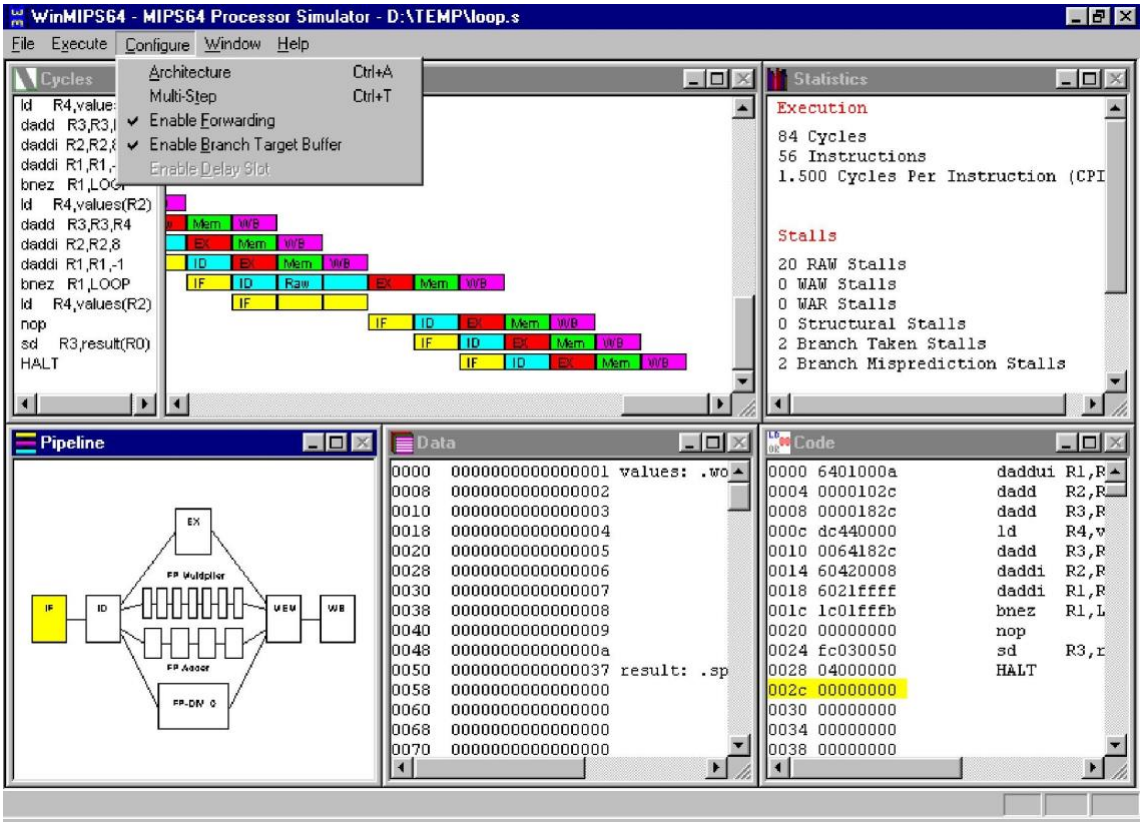

Рис.7.18 Головне вікно симулятора по завершенню симулювання програми ЦИКЛ

CPI – це (середнє) число тактових інтервалів (cycles per instruction), що припало на виконання кожної інструкції програми. В ідеальному випадку для нашого п’ятисходинкового конвеєра маємо CPI=1, але залежності поміж інсирукціями через дані (data hazards, RAW, WAR, WAW) збільшують CPI, тобто часові витрати на виконання програми.

По виконанню програми з оптимізованою апаратурою втрати конвеєра інструкцій склали 20 RAW-пригальмувань, 2 пригальмування під час виконання виконаних взятих, виконаних умовних переходів (Branch taken) і ще 2 пригальмування через помилками передбачення напрямку умовного переходу апаратними засобами (тут використовується буфер цільових адресепереходів - branch target buffer). Зрозуміло, що при використанні неоптимізованої апаратури час виконання програми зросте. Симулюванням треба подати відповідь та питання – коли, чому, наскільки?

Варіанти завдань

Персональні варіанти завдань знайдете у табличці з варіантами.