Київський національний університет

імені Т.Шевченка

**Звіт**

до лабораторної роботи №3

на тему:

«*Імітація роботи процесора****»***

***Студента другого курсу***

***Групи К-27***

***Факультету комп’ютерних наук***

***та кібернетики***

***Приходька Ярослава Андрійовича***

***Київ***

*2020*

**Мета**

Необхідно розробити програмну модель процесора та реалізувати його імітаційну (тобто комп’ютерну) модель.

Виконавцю буде запропоновано індивідуальний варіант, в якому буде визначена конкретна:

1)    адресність процесора (1-, 2-, 3-адресна або стекова);

2)    бітність процесора (магістралі даних);

3)    обов’язкова для реалізації команда процесора (згідно індивідуального варіанта).

Має бути реалізовано:

1)    розміщення інтерпретуємої програми у текстовому файлі (наприклад, один рядок=одна команда);

2)    мінімум 2 команди (одна з них - занесення значення у регістр, інші задаються варіантом);

3)    для операндів/регістрів представлення побітно, можливо, для деяких варіантів із побайтним групуванням бітів;

4)    фіксація у регістрі стану  як мінімум знаку ре­зуль­та­ту виконання команди;

5)    потактове виконання команд (наприклад, 1-й такт – занесення команди у регістр команди, 2-й такт -  виконання операції і занесення результату).

**Основні принципи виконання роботи**

Структура процесора

Зпрощено будь-який процесор може бути представлений як пристрій, що виконує фіксовану кількість операцій над фіксованим форматом даних.

Операції, будучи обмеженими заданими форматами, називаються командами. Пос­лі­дов­ність команд утворює програму в кодах конкретного процесора (або об’єктна програма).

Команда, як послідовність деяких дій над даними, виконується по тактам (мікропрограма команди). І, у залежності від формату операндів, кількість тактів може бути різна. Процесор працює з командами та даними, де дані є не просто масивами бітових рядків, а операндами команд. Тобто доступ процесора до пам’яті виконується у вигляді звернення (зчитування/запис) до операндів. По суті, нічого іншого процесор не робить. Команда має вигляд:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Код команди |  | 1-й операнд |  | 2-й операнд | ........ | N-й операнд |

З максимальною кількістю операндів для команд пов’язується поняття адресності процесора. Типові реалізації: 1‑ад­рес­ні, 2-адресні та 2.5-адресні (коли третій операнд неявний, або один чи декілька операндів задають діапазон фактично задіяних операндів). Найчастіше результат команди заноситься за місцем першого операнда. Таким чином, процесор вирізняє у оперативній пам’яті (ОП)  коди і дані, які за формою співпадають між собою (це бітові рядки заданих форматів). Формат операндів закладається у формат команди. Зафіксовану множину команд, їх формати та формати даних відносять до програмної  моделі процесора.

Абсолютно повноцінно може виконуватися програма користувача, коли всі операнди є полями в ОП. Але типово у програмах ланцюжки команд готують проміжні результати для наступних за ними команд, тобто частина даних постійно потрібна на протязі деякого сегменту команд, Тому якщо такі дані зберігати в ОП, щоб зразу їх викликати для наступного виразу, недоцільно, бо швидкість роботи процесора перевищує швидкості всіх інших пристроїв, включно із ОП, яка є звичайним пристроєм на магістралі системи. З метою оптимізації переміщення даних процесори завжди мають свою власну невелику пам’ять у вигляді регістрів. Останні, як правило, спеціалізують під конкретні формати даних: 32-бітні цілі, 32/64-бітні з плаваючою точкою тощо, і кількісно їх – від десятків до декількох сотень. Також регістрам присвоюється власне ім’я чи номер. Відповідно, компілятори оптимізують код генеруємих програм для максимального використання регістрів. Серед команд є обов’язково команди переміщення даних, зокрема, і занесення даних з ОП у регістри та обміну кодами між регістрами. В одноадресних процесорах для виконання бінарних операцій вводиться, як правило, регістр акумулятор для неявного представлення одного з операндів, він же і приймає результат.

Розглянемо основний склад структур представлення внутрішніх даних будь-якого універсального процесора (без реалізації конвейєра команд).

1)       Регістри даних: цілих та адрес, даних з плаваючою точкою. Цілочисельні регістри часто іменують регістрами загального використання.

2)      Регістр команди, яка в даний момент інтерпретується. Містить спочатку саму команду для подальшого розбору, зокрема виділення операндів та приведення їх до потрібного для операції робочого вигляду, бо на рівні команди операнди можуть представлятися:

явно у команді (літералом);

номером/ім’ям регістра;

 адресою у ОП;

 косвеною адресою, тобто адресою поля ОП чи регістра, в якому зберігається адреса значення операнда.

3)       Регістр стану, в якому фіксується бітовими комбінаціями, зокрема:

·         знак попередньої арифметичної операції (мінус, нуль чи плюс);

·         виникнення переповнення;

·         втрата значимості (коли у числі з плаваючою точкою при ненульовому порядку мантиса стала нульовою);

·         превілевійовані режими, деякі аварійні ситуації;

·         ключ захисту пам’яті (маска пам’яті) тощо.

4)       Регістр адреси поточної чи наступної команди. Може містити у лабораторній поточний номер команди.

Представлення цілочисельної інформації в процесорах та пам’яті

Для представлення цілих зі знаком є багато кодів, проте найчастіше використовується прямий та доповнюючий коди. У прямому коді серед N бітів числа виділяється один знаковий біт (0- плюс, 1 –мінус), а інші біти представляють число 2-й системі числення за модулем:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| біт N-1  біт знаку | біт N-2 числа  (старший) | біт N-3 числа | ………… | біт 1 числа | біт 0 числа  (молодший) |

Наприклад, у форматі із 8 бітів число –5 матиме у прямому коді вигляд 10000101, а +5 таке: 00000101.

Доповнюючий код був задуманий для того, щоб команду додавання та віднімання реалізувати на апаратному рівні як одну команду, проте на рівні програмної моделі процесора залишається і команда додавання, і віднімання. Побітовий формат числа аналогічний наведеному вище, а біти числа кодуються дещо поіншому.

Додатнє число у доповнюючому коді співпадає з формою прямого коду. Від’ємне ж число будується так:

|  |  |  |
| --- | --- | --- |
| Крок | Дія | Коментар |
| 1 | Представляємо число у прямому коді і як додатнє | Представимо, наприклад, –5 у доповнюючому коді (знаковий біт – зліва):         00000101 |
| 2 | Інвертуємо всі біти | 11111010 |
| 3 | До отриманого числа додаємо 1 | 11111010  +      00000001 |
| 4 | Отримуємо число у доповнюючому коді | 11111011 |

Наприклад, число –1 у доповнюючому коді у всіх бітах формату буде мати 1. Тепер, коли всі операнди пред­став­ле­ні у доповнюючому коді, ми можемо виконувати лише операцію додавання. Знакові біти приймають участь у операції і прий­мають біти переносу від молодших розрядів. Для прикладу складемо (+5)+(-5), маючи до послуг вже отримане представлення у форматі з 8 біт:

    00000101

+

    11111011

 1’00000000

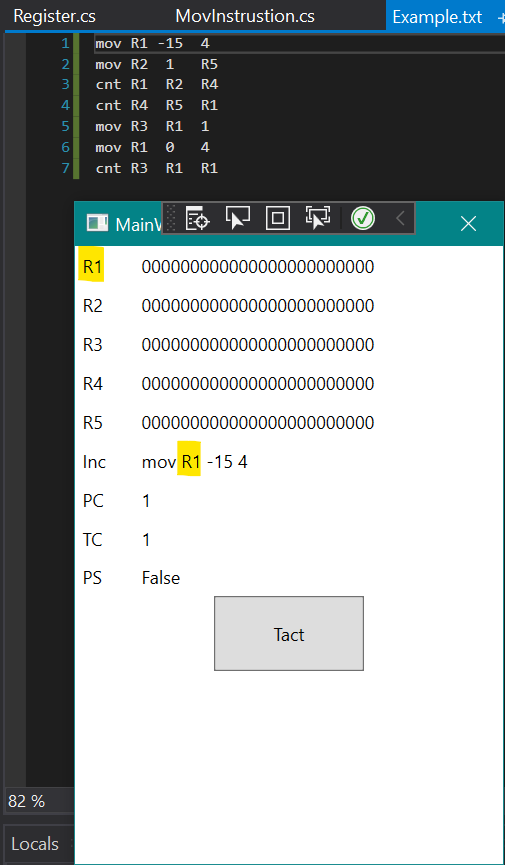
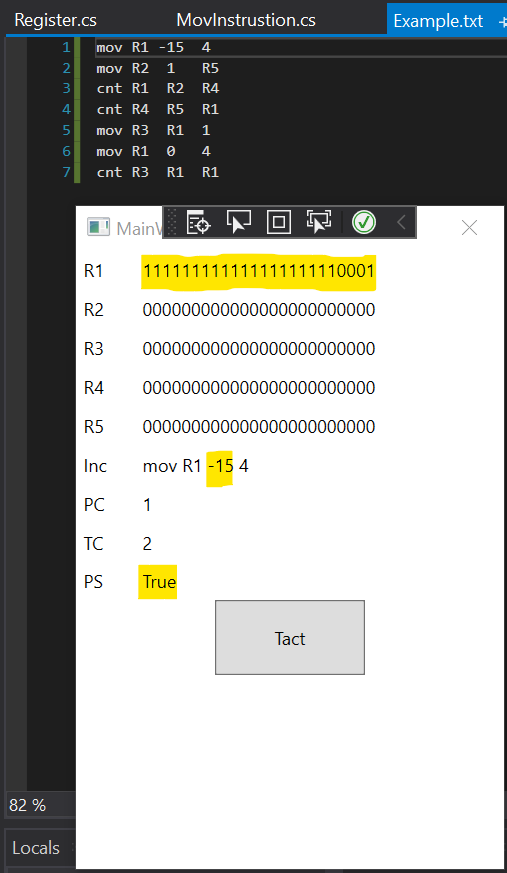
де апострофом відокремлений біт переносу в результаті переповнення у знаковому біті, тобто це вже мав бути 9-й біт, але для 8-бітного формату він втрачається, - отже в результаті отримуємо 00000000.

**Лістинг програми**

Також буде прикладено відео роботи програми.

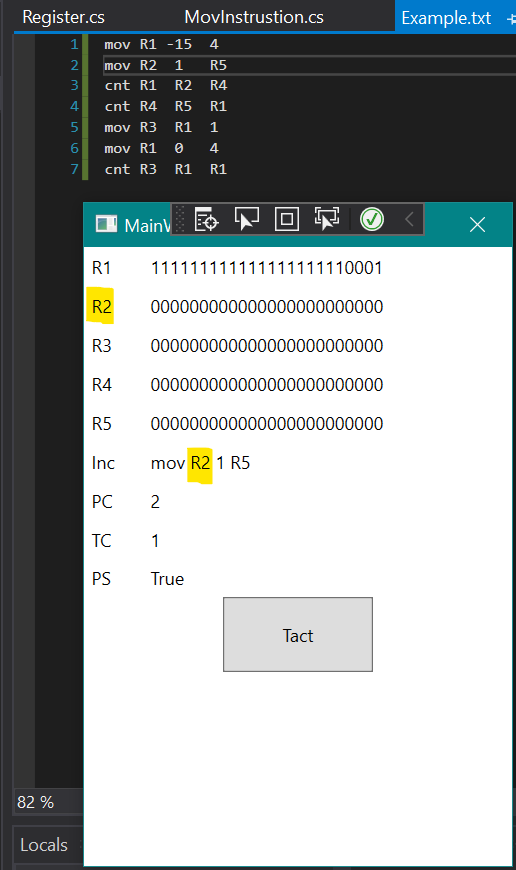
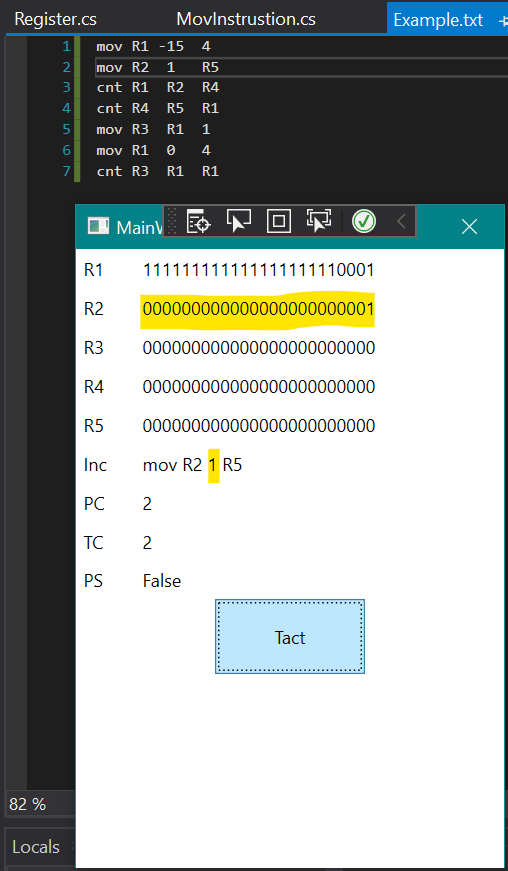
Команди описані у висновку.

Інструкція 1:

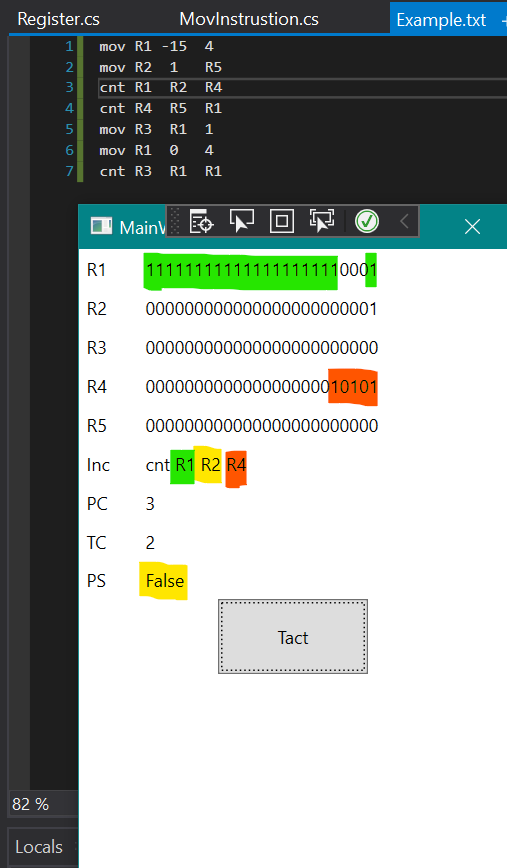
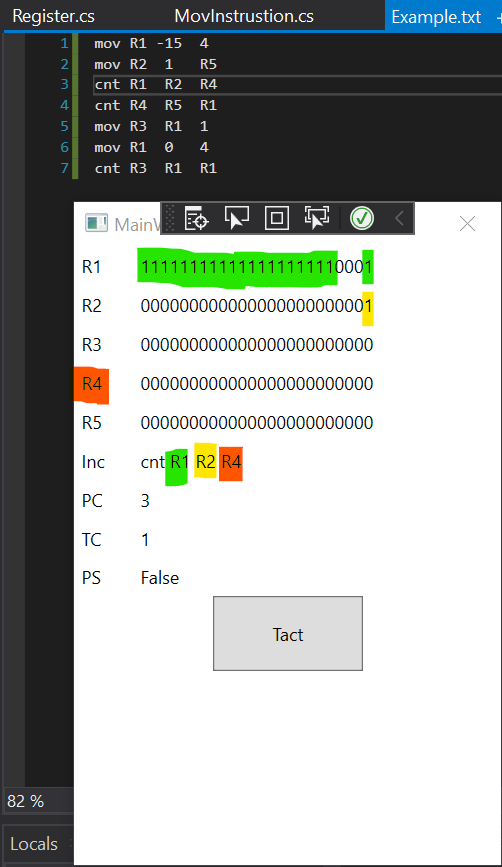
****

1 Такт 2 Такт

Інструкція 2:

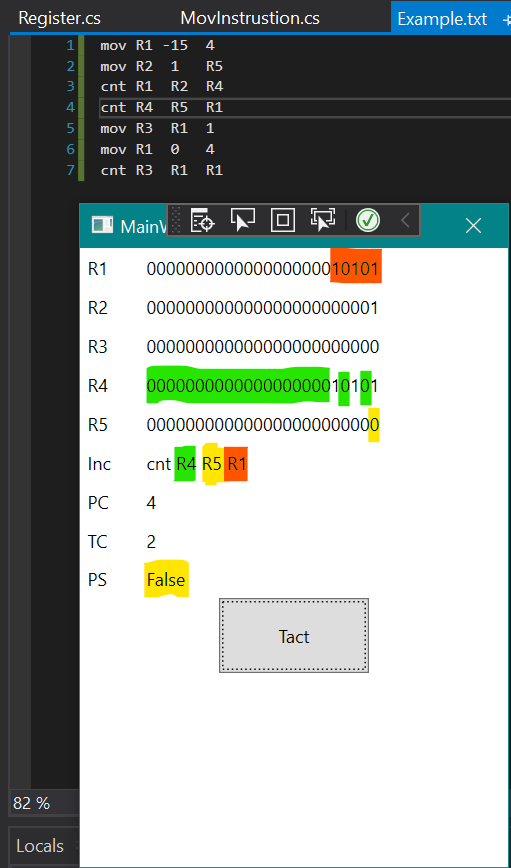
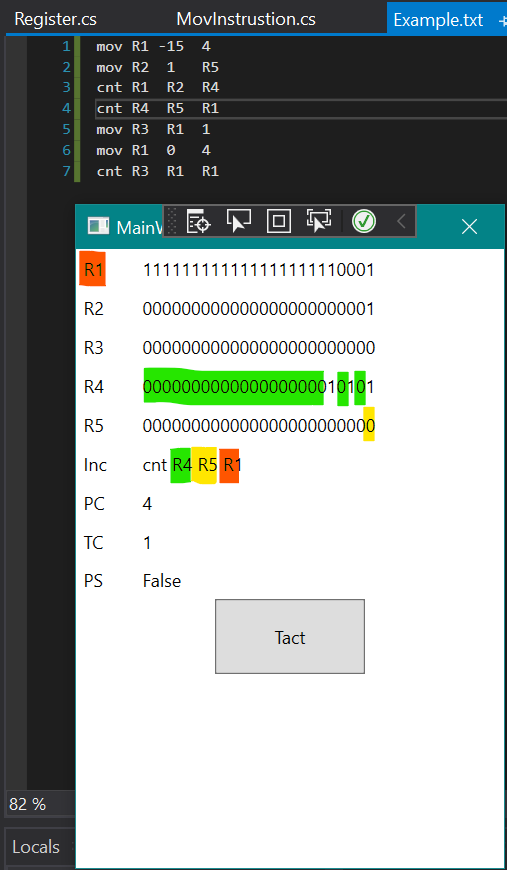
****

1 Такт 2 Такт

**** Інструкція 3:

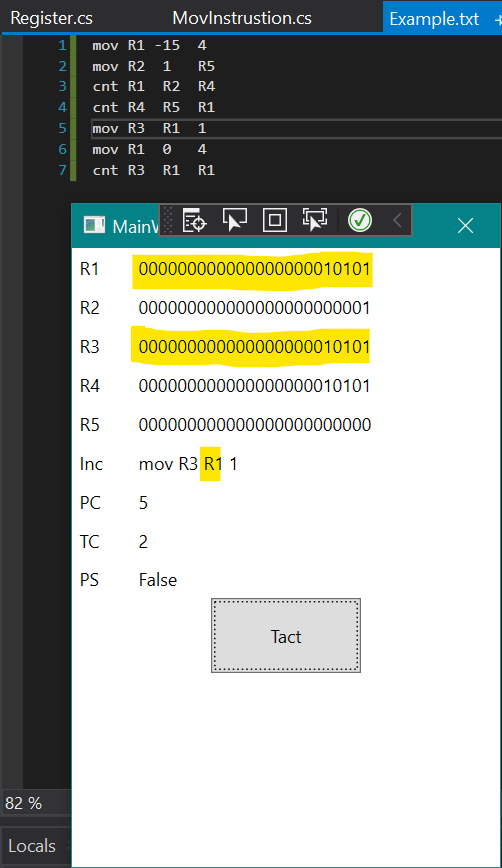
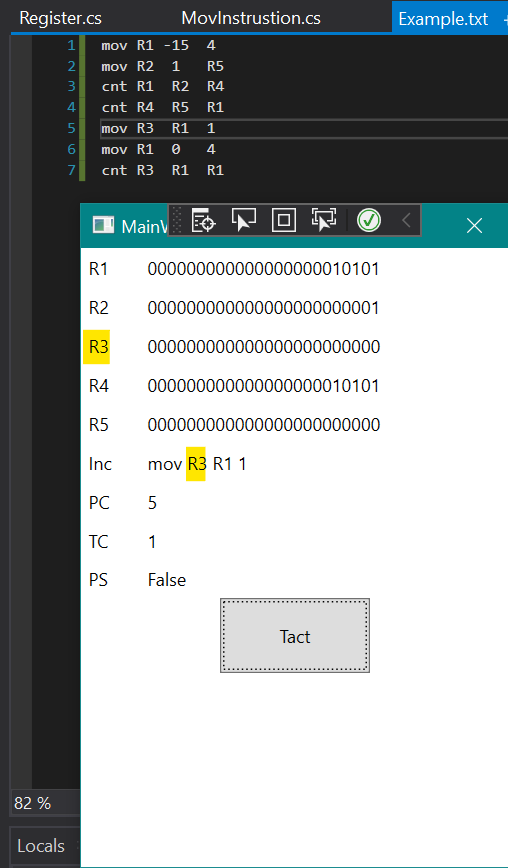
Такт 1 Такт 2

Інструкція 4:



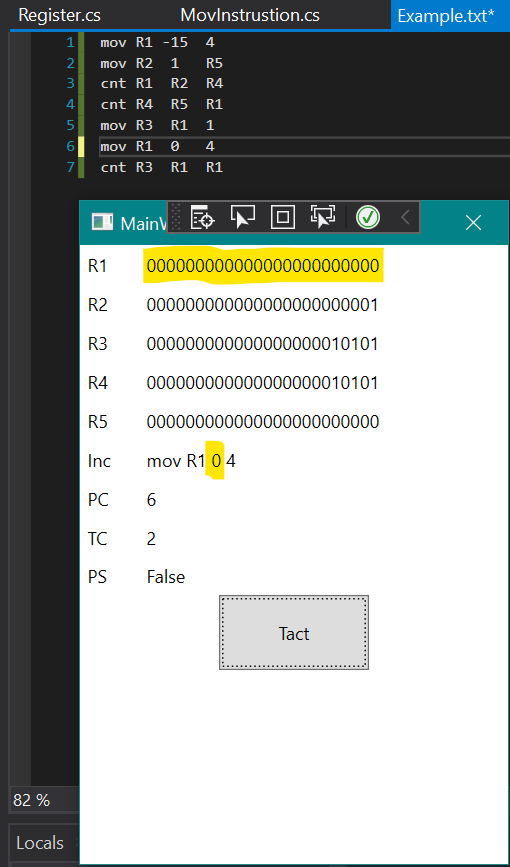
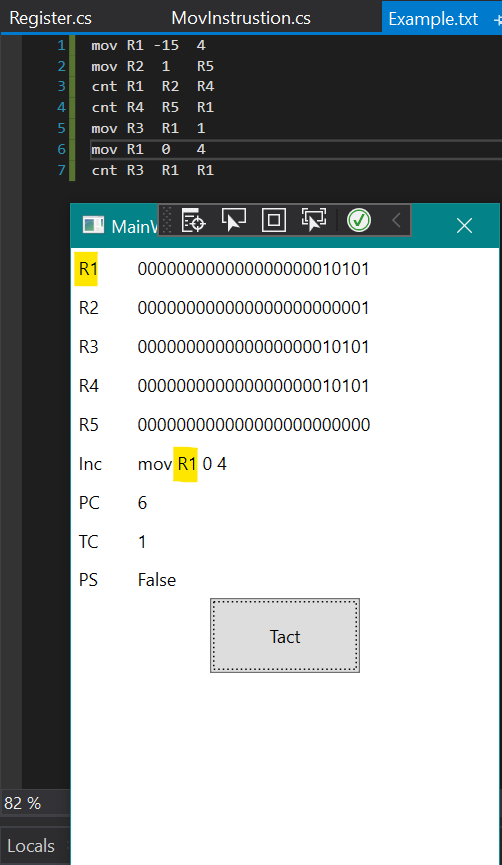
Такт 1 Такт 2

Інструкція 5:



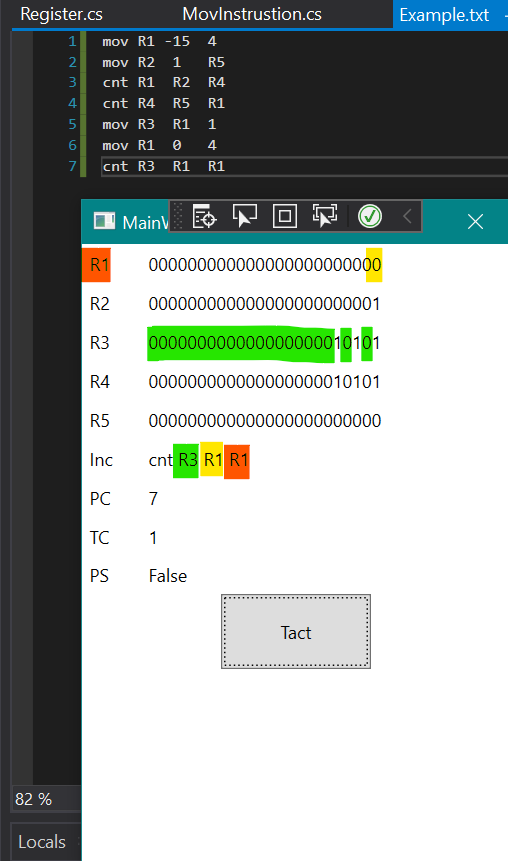
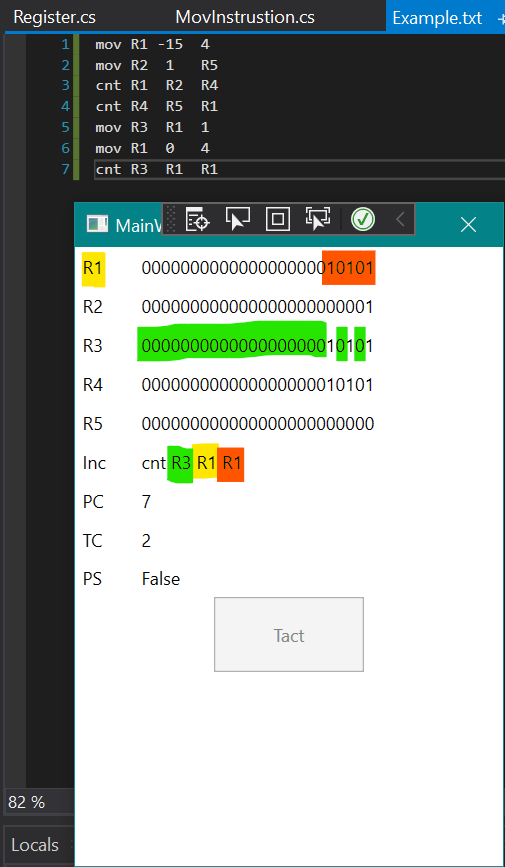
Такт 1 Такт 2

Інструкція 6:



Такт 1 Такт 2

Інструкція 7:

Такт 1 Такт

**Висновок**

В даній лабораторній роботі, я на практиці засвоїв принципи роботи спрощеної моделі імітації роботи процесора, бело реалізовано триадресну структуру інструкцій, в моделі було 5 регістрів процесора, кожен по 24 біти, команди mov та cnt.

cnt – рахує кількість 0 чи 1 в бітовому представленні даних, що зберігаються в першому аргументі, в залежності від значення другого, та зберігає в третій.

mov – зберігає значення другого аргументу в перший, третій аргумент ніяк не використовується і потрібен лише для того щоб відповідати вимозі на 3-адресність.

Також довелося враховувати особливості бітового представлення чисел в пам’яті комп’ютера. Але в цілому робота виявилась не дуже складною та в певному розумінні цікавою з точки зору нового погляду на роботу процесора, що використовується практично в кожному сучасному пристрої.

**Додаток. Код програми**

<https://github.com/pryhodkin/ProcessorImitator>