Лабораторна робота "Імітаційна модель процесора"

1. Теоретичні відомості

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 по-іншому.

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*.

2. Постановка задачі

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- чи 3-адресна, або кількість операндів);
2) бітність процесора (магістралі даних);
3) обов'язкова для реалізації команда процесора.;

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/стек/ОП, інші задаються варіантом);
3) для операндів/регістрів представлення побітно, можливо, для деяких варіантів із побайтним групуванням бітів. Оперативна пам'ять має представлятися у 16-річному форматі;
4) фіксація у *регістрі стану* як мінімум знаку результату виконання команди;
5) потактове виконання команд (наприклад, 1-й такт – занесення команди у регстр команди, 2-й такт - інтерпретація операндів, 3-й такт – виконання операції і занесення результату).

3. Рекомендації щодо виконання роботи

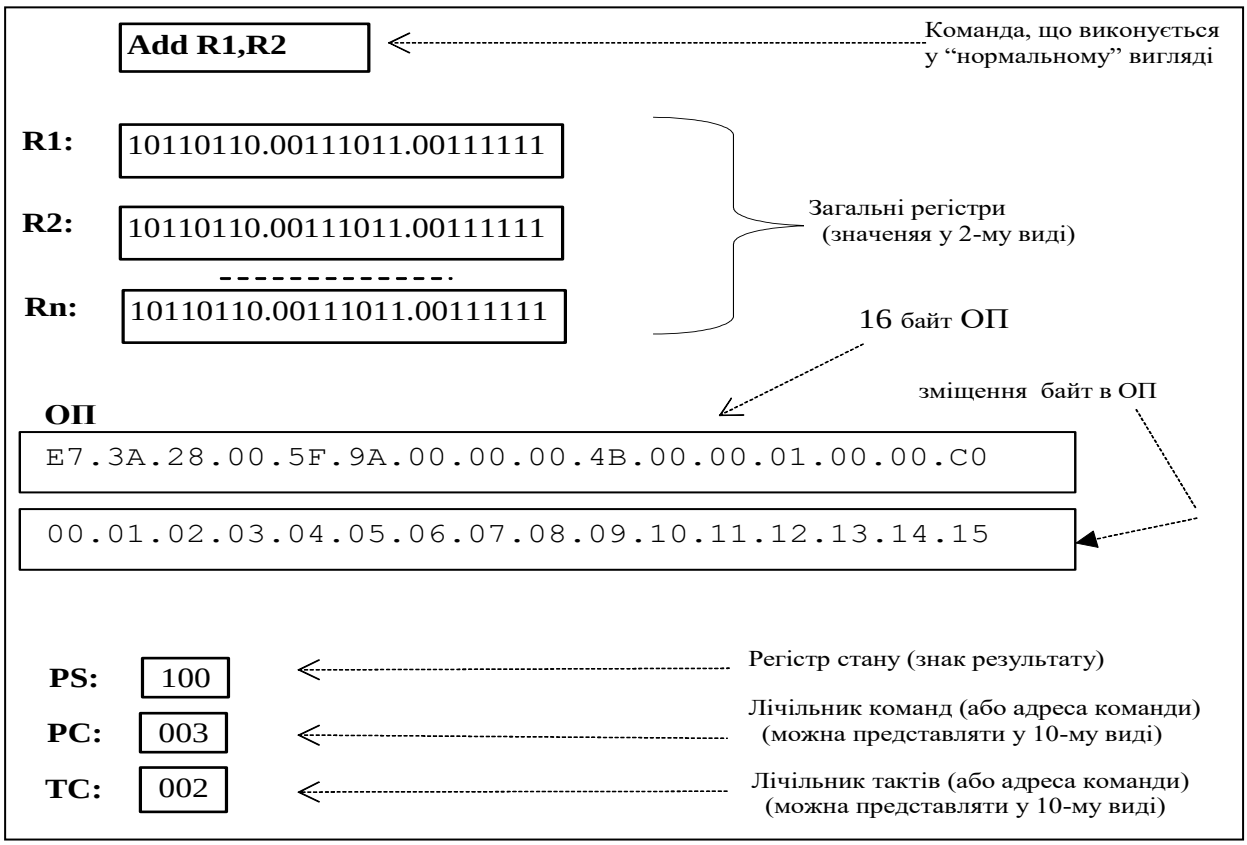
1. Щодо вибору кількості тактів для команд - мінімально достатньо двох: 1-й) занесення поточної команди у регістр команди; 2-й) виконання команди.

2. Щодо реалізації ОП - можна обмежитися роботою лише з 10-20 байтами (але вони мають бути відображеними на екрані).

3. Для відрахунку тактів найзручніше брати нажимання певної чи будь-якої клавіші клавіатури. Доцільно завжди мати можливість вийти із стану інтерпретації програми імітуємим процесором за допомогою, наприклад, *Esc*. Не варто реалізовувати програмної затримки для кожного такта у порівнянні із "тактуванням" за допомогою миші чи клавіатури, бо вибраний вами темп може перешкоджати аналізу виконуваних дій.

4. Файл інтерпретуємої програми має бути заданим у програмі або один раз у командному рядку запуску імітаційної моделі. Тобто програма не повинна пропонувати у діалозі вказати файл команд для виконання імітуємим процесором.

5. На екрані (достатньо в режимі скролінгу тексту і без рамок для даних у регістрах) для кожного такту повинні бути представлені такі структурні елементи процесору із даними в них:



Якщо має бути одно адресна безстекова модель команд, то ще потрібний регістр, який називають ***акумулятор***. В командах він явно операндом не задається, хоча може використовуватись як реальний операнд. Типово для команд із акумулятором-операндом є зберігати результат команди саме в акумуляторі.

Приклад інтерпретуємої програми (ланцюжка команд):

- Нехай процесор має чотири загальних регістри: R1, R2, R3, R4 довжиною 8 біт. Спочатку у регістрах будь-яке значення.
- Для представлення даних використовується додатковий код.
- Всі інші регістри – як на схемі вище. Власні імена регістрів виконавець може змінювати і вибирати сам.
- Регістр стану із одного біта – знакового біта із значенням *0* для '+' та *1* для '-'.
- Програма у прикладі складається із 4 таких інструкцій (команд) процесора:

```
Load R2,-5
Load R1,3
Add  R1,R2
Add  R2,R3
```

- Розшифровка виконання програми наведена у наступній таблиці (фактично те, що видно на екрані, знаходиться у 3-й колонці, коментар тактів, взятий у *(...)* на екрані не потрібний).

Команда	Коментар	Демонстрація дій процесора
Load R2,-5	Завантаження у *R2* константи *–5*	*(Подаємо ззовні 1-й такт – це 1-й такт 1-ї команди)* **Команда** = Load R2,-5 **R1** = 0101 0011 **Ins** = Load | R2 | 1111 1011 **R2** = 0111 0011 **PC** = 1 **R3** = 1100 0111 **TC** = 1 **R4** = 0111 1011 **PS** = 0 *(Подаємо ззовні 2-й такт – це 2-й такт 1-ї команди)* **Команда** = Load R2,-5 **R1** = 0101 0011 **Ins** = Load | R2 | 1111 1011 **R2** = 1111 1011 **PC** = 1 **R3** = 1100 0111 **TC** = 2 **R4** = 0111 1011 **PS** = 1
Load R1,3	Завантаження у *R1* константи *3*	*(Подаємо ззовні 3-й такт – це 1-й такт 2-ї команди)* **Команда** = Load R1, 3 **R1** = 0101 0011 **Ins** = Load | R1 | 0000 0011

Команда	Коментар	Демонстрація дій процесора
		R2 = 1111 1011 P**C** = 2 **R3** = 1100 0111 **TC** = 1 **R4** = 0111 1011 **PS** = 1 *(Подаємо ззовні 4-й такт – це 2-й такт 2-ї команди)* **Команда** = Load R1, 3 **R1** = 0000 0011 **Ins** = Load | R1 | 0000 0011 **R2** = 1111 1011 P**C** = 2 **R3** = 1100 0111 **TC** = 2 **R4** = 0111 1011 **PS** = 0
Add R1,R2	Скласти *R1* та *R2*, результат у 1-му регістрі. Регістр *R2* отримав значення -2_{10}.	*(Подаємо ззовні 5-й такт – це 1-й такт 3-ї команди)* **Команда** = Add R1, R2 **R1** = 0000 0011 **Ins** = Add | R1 | R2 **R2** = 1111 1011 P**C** = 3 **R3** = 1100 0111 **TC** = 1 **R4** = 0111 1011 **PS** = 0 *(Подаємо ззовні 6-й такт – це 2-й такт 3-ї команди)* **Команда** = Add R1, R2 **R1** = 1111 1110 **Ins** = Add | R1 | R2 **R2** = 1111 1011 P**C** = 3 **R3** = 1100 0111 **TC** = 2 **R4** = 0111 1011 **PS** = 0
Add R2,R3	Скласти *R2* та *R3*, результат у 2-му регістрі. У регістрі *R3* значення довільне, а саме -57_{10}, на момент запуску програми. Регістр *R2* отримав значення -62_{10}.	*(Подаємо ззовні 7-й такт – це 1-й такт 4-ї команди)* **Команда** = Add R2, R3 **R1** = 1111 1110 **Ins** = Add | R2 | R3 **R2** = 1111 1011 P**C** = 4 **R3** = 1100 0111 **TC** = 1 **R4** = 0111 1011 **PS** = 0 *(Подаємо ззовні 8-й такт – це 2-й такт 4-ї команди)* **Команда** = Add R2, R3 **R1** = 1111 1110 **Ins** = Add | R2 | R3 **R2** = 1100 0010 P**C** = 4 **R3** = 1100 0111 **TC** = 2 **R4** = 0111 1011 **PS** = 1

4. Варіанти роботи

Варіанти роботи

Варіант має три складових. із своїми підваріантами.

***1-а складова*. Адресність процесора** (це стосується індивідуальних команд варіанту):

	Адресність	Коментар
1.	1-адреса	1-й операнд завжди в акумуляторі, результат команди заноситься в акумулятор
2.	2-адресна	Результат розміщується у 1-му операнді
3.	3-адресна	Результат розміщується у 1-му операнді
4.	Стекова (безадресна)	Результат заноситься у верхівку стека. Можуть бути також декілька регістрів.

***2-а складова*. Бітність регістрів/стеку та операндів команд**:

	Бітність	Коментар
1.	12-бітні	
2.	14-бітні	
3.	15-бітні	
4.	18-бітні	

5.	20-бітні	
6.	22-бітні	
7.	24-бітні	
8.	26-бітні	
9.	28-бітні	
10.	30-бітні	

***3-а складова*. Команди процесора**:

Кожний варіант має спільні команди(у) (мнемоніку команд виберіть самостійно):

- занесення в регістр (акумулятор, стек) даного, яке може бути чи літералом у команді, чи адресою в пам'яті;
- занесення в пам'ять (із регістру/стеку чи літералу);
- **Індивідуальні варіанти**:

1.	Цілочисельне додавання (у доповнюйочому коді).
2.	Цілочисельне віднімання (у доповнюйочому коді).
3.	Логічний зсув вліво та вправо.
4.	Арифметичний зсув вліво та вправо.
5.	Циклічний зсув вліво та вправо.
6.	Побітове додавання по модулю 2.
7.	X mod Y (остача від ділення X на Y).
8.	X and Y (логічне множення).
9.	X or Y (логічне додавання).
10.	Доповнення до числа у доповнюючому коді за умови що змінюваний операнд не менший за значення у: • вказаному регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів.
11.	Інверсія лише парних/непарних бітів в залежності від параметра (значення вибрати самостійно), представленого у: • регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів.
12.	Інверсія у суміжних парах бітів за умови рівності старшого біта пари значенню 2-го операнда команди, представленого у: • регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів.
13.	Кількість одиничних/нульових бітів в залежності від значення *1* чи *0* 2-го операнда команди, представленого у: • команді безпосередньо чи регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів.
14.	Операнд з даними складається із байтів-символів із цифр та латиниці. Команда перетворює набір символів у 1-му операнді у верхній/нижній регістр в залежності від значення 2-го операнда команди (його значення вибрати самостійно), представленого у: • команді безпосередньо чи регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів. Біти регістра/стека відображати із побайтовою розбивкою.
15.	Перестановка значень пари бітів у 1-му операнді. Номери бітів задаються 2-м операндом у вигляді *AABB*, де *AA* та *BB* є номерами бітів (з ведучими нулями для доповнення до двозначного числа). 2-й операнд представляється у: • команді безпосередньо чи регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів.
16.	Кожний байт 1-го операнда окремо складається по модулю 2 з молодшим байтом 2-го операнда, представленого у:

	• команді безпосередньо чи регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів. Біти регістра/стека відображати із побайтовою розбивкою.
17.	Кожний байт операнда команди розглядається як цифра від *0* до *9* (без знаку). Команда із двох операндів будує нове значення, в яке заноситься найбільша цифра з кожного одноіменного байту операндів. Необхідно врахувати: • біти регістра/стека відображати із побайтовою розбивкою; • у варіантах з неповним байтом (18-, 20-, 22-бітні операнди) такий байт доповнюємо ведучими нулями; • передбачити формат команди із занесенням числа в операнд у літеральній формі (з цифрамии 0..9 для відповідних байтів).
18.	Кожний байт 1-го операнда команди розглядається як цифра від *0* до *9* (без знаку). Команда повертає суму цих байт-цифр, взятих по модулю числа, заданого 2-м операндом, представленого у: • команді безпосередньо чи регістрі для безстекової реалізації; • верхівці стека в стековій реалізації розміщення операндів. Необхідно врахувати: • біти регістра/стека відображати із побайтовою розбивкою; • у варіантах з неповним байтом (18-, 20-, 22-бітні операнди) такий байт доповнюємо ведучими нулями; • передбачити формат команди із занесенням числа в операнд у літеральній формі (з цифрамии 0..9 для відповідних байтів).
19.	Логічний зсув вліво та вправо групи (не менше 2-х) регістрів (або верхніх елементів стеку). Біти регістрів (елементів стеку) при зсуві розглядаються як єдиний бітовий рядок. Самостійно передбачити варіант задання операнда, який визначає кількість початкових регістрів (елементів стеку) для зсувів.
20.	Циклічний зсув вліво та вправо групи (не менше 2-х) регістрів (або верхніх елементів стеку). Біти регістрів (елементів стеку) при зсуві розглядаються як єдиний бітовий рядок. Самостійно передбачити варіант задання операнда, який визначає кількість початкових регістрів (елементів стеку) для зсувів.

Власне номер варіанту, умовно позначимо як *A.B.C*, складається із трьох чисел:

- перша цифра/число *A* є підваріантом 1-ї складової;
- друга цифра/число *B* є підваріантом 2-ї складової;
- третя цифра/число *C* є підваріантом 3-ї складової.

Наприклад, номер вашого варіанту ***2.4.7***, тоді його цифри будуть означати:

- ***2*** – у 1-й складовій у вас буде 2-й підваріант (тобто *"процесор 2-адресний"*);
- ***4*** – у 2-й складовій у вас буде 4-й підваріант (тобто *"18-бітні регістри/операнди"*)
- ***7*** – у 3-й складовій у вас буде 7-й підваріант (тобто потрібно, крім загальних команд, також реалізувати команду *"X mod Y"*).