ЛР № 2. Дослідження макроалгоритмів та мікроалгоритмів виконання машинних інструкцій.

Мета: зрозуміти і дослідити макроалгоритм та мікроалгоритм виконання кожної машинної інструкції машини Ноймана.

Завдання: покроковим режимом протестувати виконання кожної машинної інструкції, проаналізувати і пояснити отримані результати, потактовим режимом протестувати поокреме виконання кожної машинної інструкції, проаналізувати і пояснити отримані результати, скласти звіт з виконання лабораторних досліджень та захистит.

Методика виконання лабораторної роботи

В покроковому (по-інструкційному, покомандному) режимі виконання програма може містити навіть одну машинну інструкцію. Її виконують одноразовим натисканням влавіші S. Перезапустимо симулятор та уведемо до нульової комірки пам'яті машинну інструкцію обчислення модуля різниці, тобто, дослідимо алгоритм виконання машинної інструкції віднімання. При цьому спланує використання пам/яті, до якої треба увести код машинної інструкції, коди операндів і виділити місце для запису результату. Наприклад, операнди розташуємо в комірках з адресами 13 і 14, а результат – в комірці з адресою 15. Код інструкції запишемо до комірки з нульовою адресою. Нагадаємо, що в симуляторі покроковий режим має назву Крок.

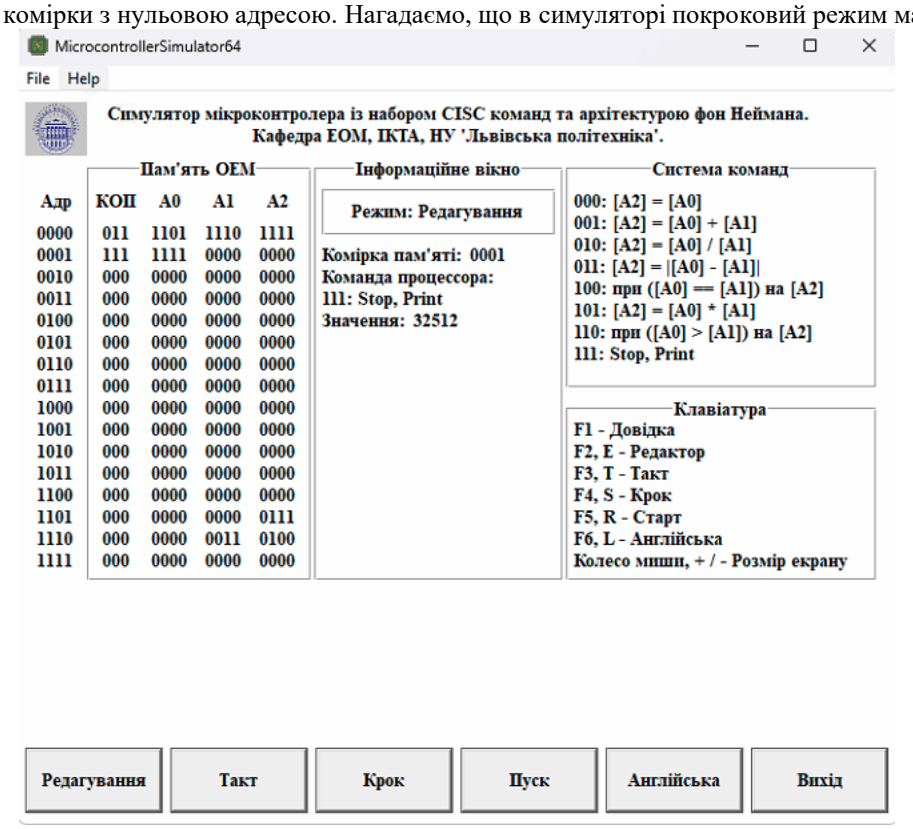

Рис. 2.1 – Стан симулятора з інструкцією віднімання [- A1 A2 A3 чи – 13_{10} 14_{10} 15_{10}]. Отже, наказали відняти від вмістимого комірки 13 вмістиме комірки 14, а модуль результату записати до комірки 15. Іншими словами, наказали обчислити модуль mod (7 – 52) = ?.

Після одноразового натискання на клавішу S отримуємо наступний стан симулятора:

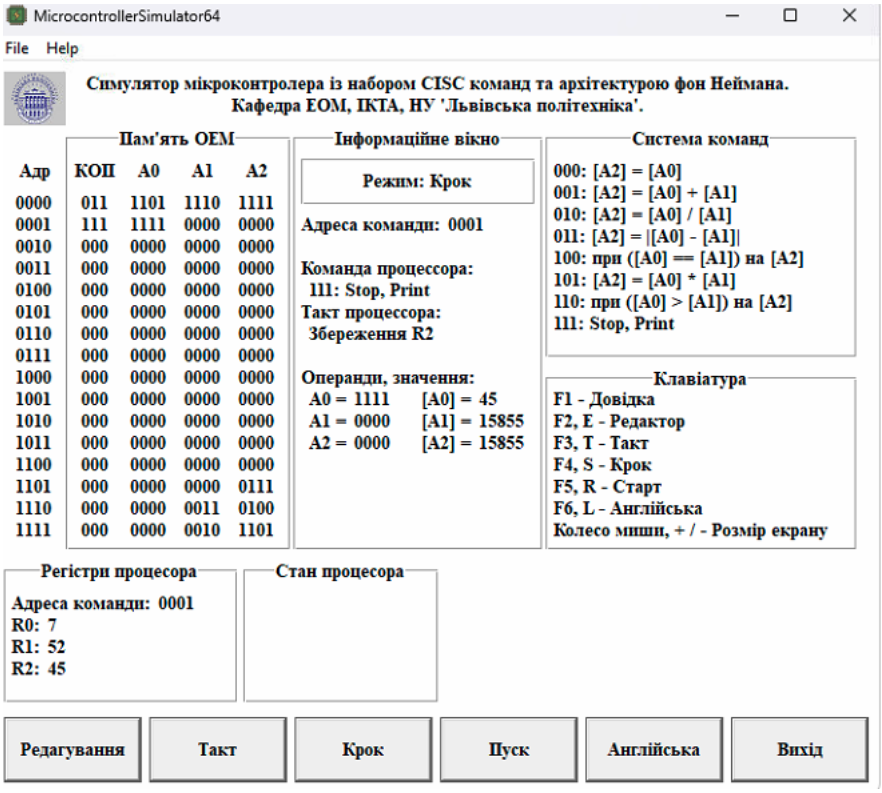

Рис. 2.2 – Стан симулятора з інструкцією віднімання [- A1 A2 A3 чи – 13_{10} 14_{10} 15_{10}]. Отже, наказали відняти від вмістимого комірки 13 вмістиме комірки 14, а модуль результату записати до комірки 15. Коректно обчислили mod (7 – 52) = 45.

Дослідження алгоритму виконання інструкції віднімання завершено.

В покроковому (інші назви: поінструкційний, покомандний режим) режимі виконання окремої машинної інструкції досліджують мікроалгоритм виконання цієї інструкції. Якщо прийняти до уваги, що в класі універсальних машин алгоритми програмують, програми завантажують до комп'ютерів і ними виконують, то зараз вийде наступне.

1. Програми складають з машинних інструкцій. Кожна інструкція має свій алгоритм виконання і цей алгоритм має точно уявляти системний програміст, що пише програму машинним кодом (як не дивно, але таке трапляється і трапляється не так зрідка). Алгоритми виконання кожної окремої машинної інструкції, що сприймає програміст, називають макроалгоритмами.
2. Для комп'ютерного інженера кожний макроалгоритм, особливий для кожної машинної інструкції, теж потрібно «запрограмувати», тобто, в певній формі записати «програму», що реалізує макроалгоритм виконання тої чи іншої машинної інструкції. Це подібно до програмування, але є іншим.
3. Кажимо, що *програміст програмує* задачу, а комп'ютерний інженер (*мікропрограміст) мікропрограмує* макроалгоритм виконання машинної інструкції. Виконання однієї такої мікропрограми є для програміста еквівалентним виконанню однієї машинної інструкції. Якщо програміст розуміє, як поокремі машинні інструкціями складають цілістну програму, то мікропрограміст цим не цікавиться. Задача мікропрограміста - надати в розпорядження програміста ефективний за певними критеріями набір машинних інструкцій (кажуть, множину машинних інструкцій).

Наприклад, множина з вісьми машинних інструкцій симулятора Кроха і його дуже обмежена пам'ять в 16 15-розрядних комірок дозволяє створити для нього компілятор мови Паскаль.

Отже, технології програмування комп'ютерів і їхнього проектування (створення мікрокодів для кожної машинної інструкції) є досить спорідненими.

4. Коли програми складені з машинних інструкцій (команд), тоді кожну машинну інструкцію складають з мікрокоманд, що всі разом утворюють мікропрограму виконання певної машинної інструкції команди).

В цій лабораторній роботі ми маємо дослідити мікроалгоритми і мікропрограми, що реалізують мікро-алгоритми, виконання кожної мащинної інструкції машини Ноймана. Далі як приклад дослідимо мікроалгоритм виконання мащинної інструкції пересилання з бінарним кодом операції 000_2. Машинна інструкція пересилання пересилає вмістиме комірки пам'яті з адресою A1 до комірки пам'яті з адресою A3. Значення і вмістиме адреси A2 ні на що не впливає. Викликаємо симулятор, пишемо до нульової комірки пам'яті машинну інструкцію пересилання та операнд до комірки з адресою A2. Під час виклику симулятора вся пам'ять онулюється. Тому спочатку за адресою призначення маємо всі нулі.

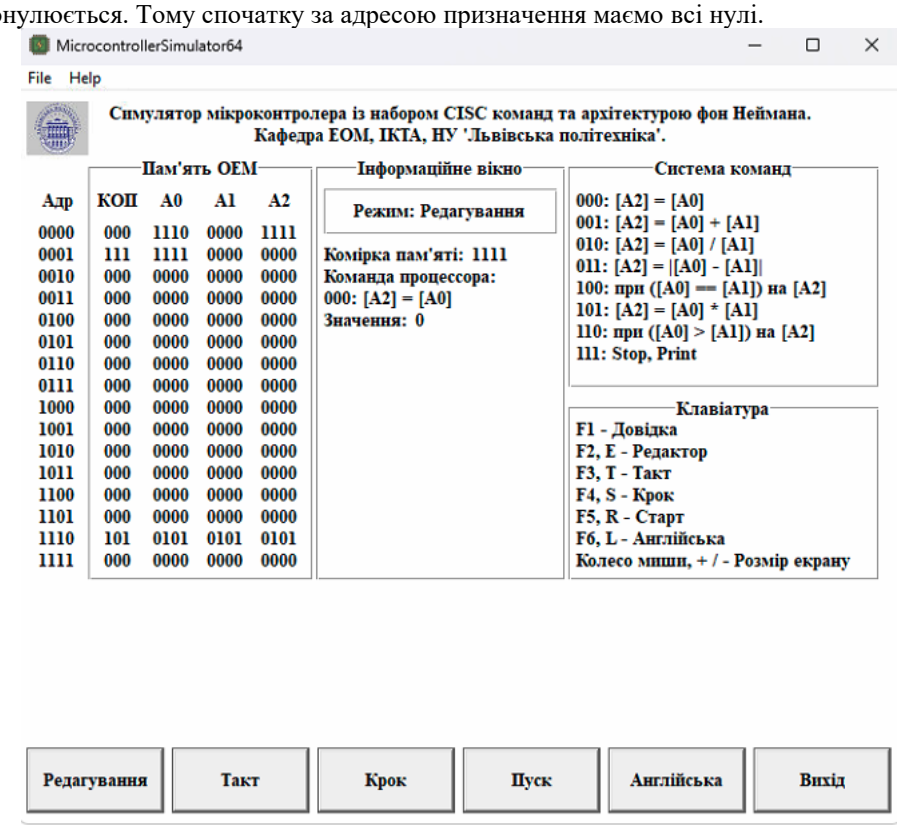

Рис. 2.3 – Машина Ноймана з машинною інструкцією пересилання в нульовій комірці пам'яті та операндом на пересилання в 14 комірці. Наповнення комірки призначення (адреса 15) до виконання пересилання є нульовим. Лічильник інструкцій онулений. Значить першою виконуватиметься інструкція, яку містить нульова комірка пам'яті.

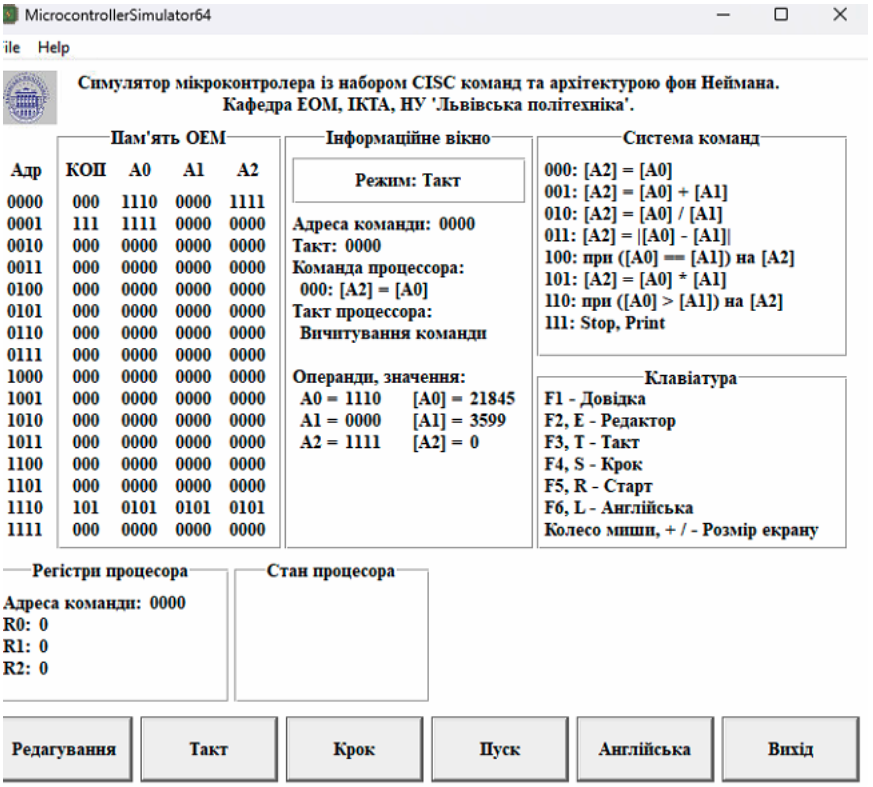

Рис. 2.4 – Стан машини Ноймана після натиснення клавіші R вибору режиму з подальшим вибором режиму ТАКТ. Режим ТАКТ дозволяє покрокове виконання але не в межах виконання програми, а в межах виконання однієї машинної інструкції. Отже, «потактово» ми бачимо мікрокроки, послідовне здійснення яких спричиняє виконання певної машинної інструкції. Іншими словами, коли виконують послідовність машинних інструкцій, тоді фактично виконують послідовність мікропрограм. Віртаульно в комп'ютері виконуються машинні програми, а реально виконуються мікропрограми тих машинних інструкцій, що записав машинними кодами програміст (як не він. То компілятор замість нього). В інформаційному вікні зазначено, що першою мікродією в мікропрограмі виконання машинної інструкції пересилання є «*вибірка команди з комірки 0000 в РК*». Тобто, першим мікрокроком бінарний код, що містить комірка пам'яті з адресою нуль (0000_2) копіюється до регістру інструкцій (РК або R0). Якщо в межах пам'яті не можна зрозуміти чим є бінарний код: інструкцією або числом, то код завантажений з пам'яті до регістру машинної інструкції (РК – регістр команди, IR – instruction register) завжди буде кодом машинної інструкції. Отже, маємо першу мікрокоманду мікропрограми виконання машинної інструкції пересилання:

`МК1: memory(0000) → IR.`

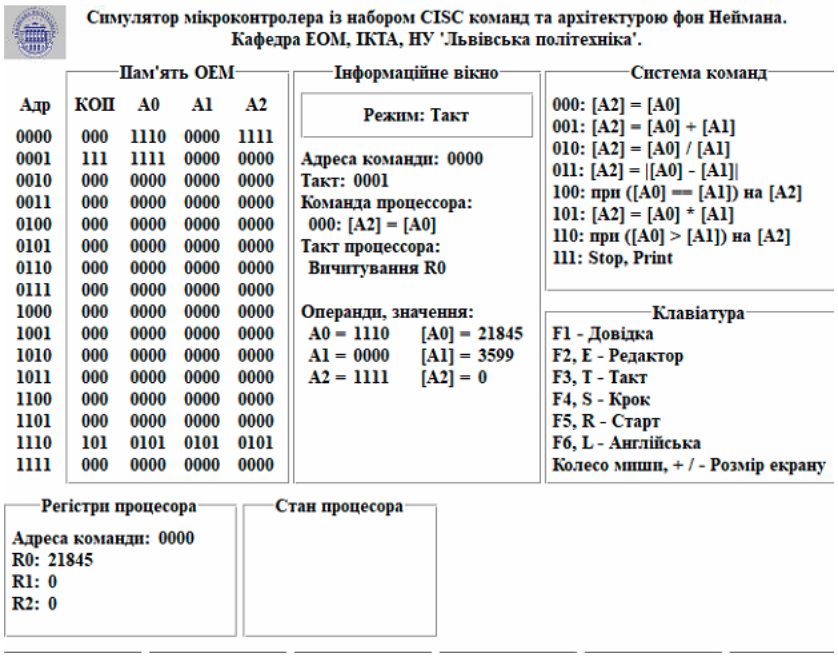

Рис. 2.5 – Натиснемо клавішу T (такт) і отримаємо цей стан машини Ноймана, коли виконалася друга мікродія мікропрограми виконання машинної інструкції пересидання. Цією мікродією вмістиме значення Такт (лічильник машинної інструкції, PC – program counter) інкрементовано (збільшено на 1), аби отримати адресу, з якої треба вибирати

наступної машинної інструкції. З якої складається програма комп’ютера. В нас уся програма – це одна інструкція в комірці з нульовою адресою. Хоч і нема, а коли була. Тоді другу інструкцію розташували за першою адресою. Саме її і обчислив лічильник номера інструкції. Отже, маємо другу мікрокоманду мікропрограми виконання машинної інструкції пересилання:

МК2: PC + 1 → PC.

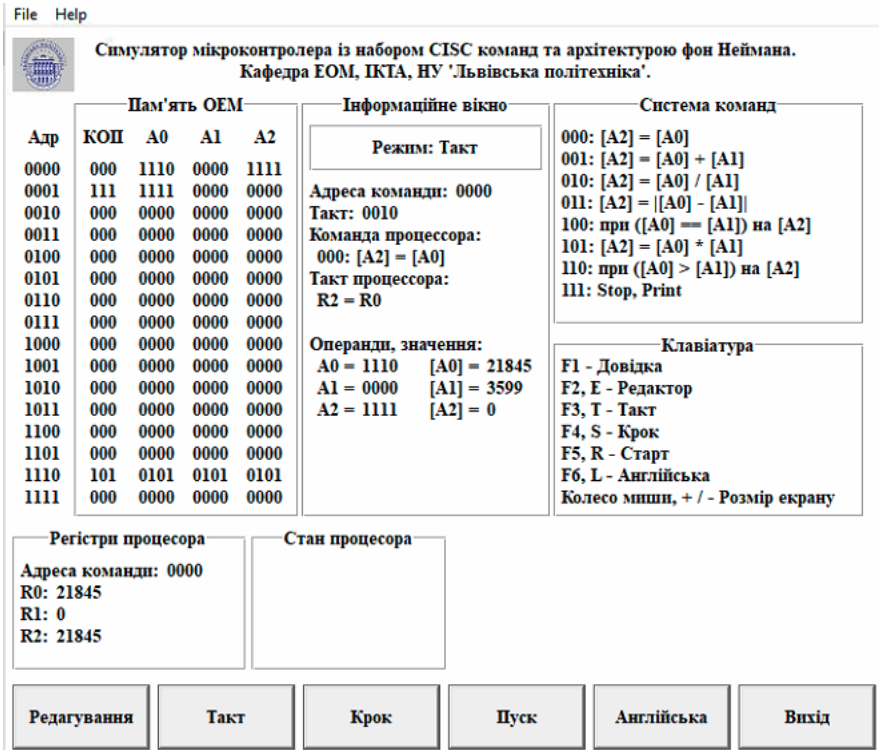

Рис. 2.6 – Ще раз натиснемо клавішу T (такт) і отримаємо цей стан машини Ноймана. Третьою мікрокомандою мікропрограми бінарний код операнда переслано з джерельної комірки пам’яті з адресою 14 (1110_2) до суматора СМ (ще має назву акумулятора, ACC або accumulator) компьютера. Отже, маємо третю мікрокоманду мікропрограми виконання машинної інструкції пересилання:

МК3: memory(1110) → ACC.

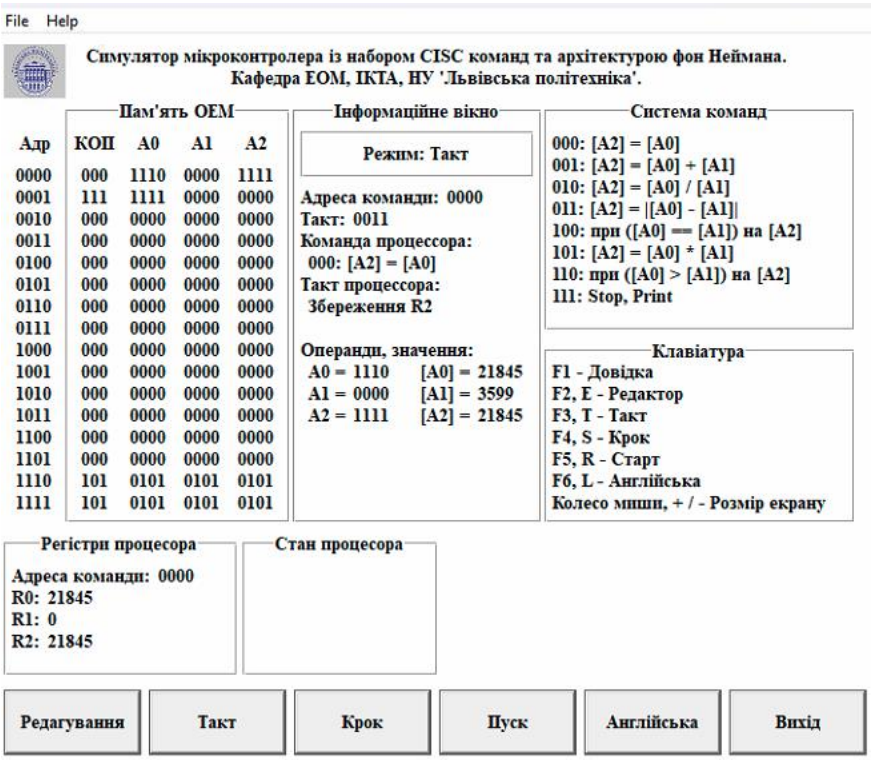

Рис. 2.7 – Ще раз натиснемо клавішу T (такт) і отримаємо цей стан машини Ноймана. Четвертою мікрокомандою мікропрограми бінарний код операнда переслано з акумулятора до цільової комірки пам’яті з адресою 15 (1111_2). Виконання інструкції пересилання завершено так само, як завершено виконання її мікропрограми. Отже, маємо четверту і завершальну мікрокоманду мікропрограми виконання машинної інструкції пересилання:

МК4: ACC → memory(1111).

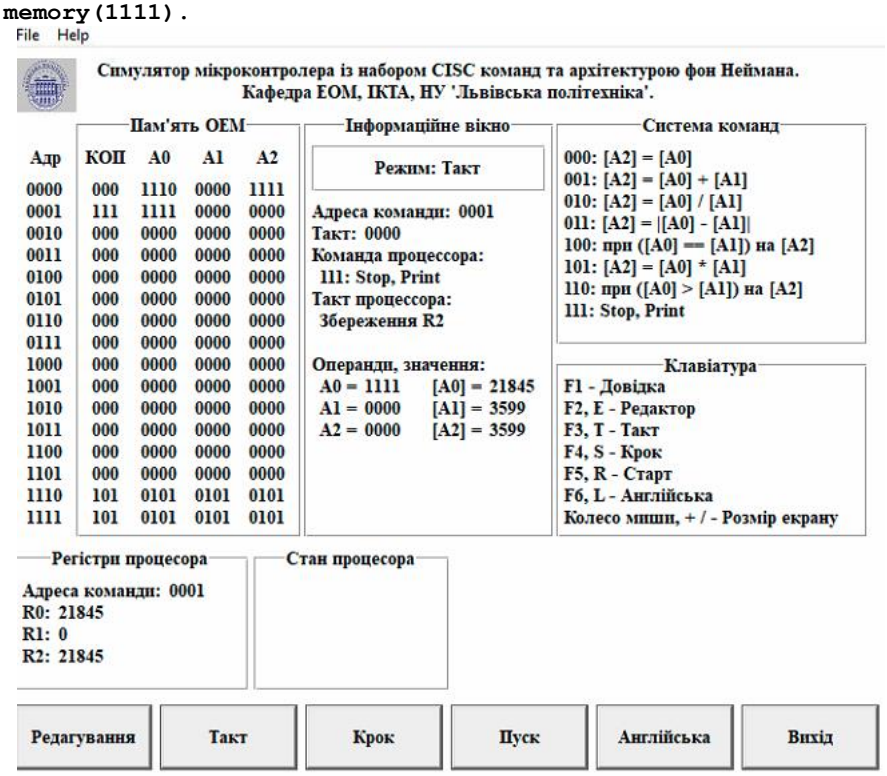

Рис. 2.8 – Якщо по завершенню мікропрограми ще раз натиснути клавішу T (такт), тоді отримаємо цей стан машини Ноймана з виконаною першою мікрокомандою мікропрограми виконання машинної інструкції, яку містить перша комірка пам'яті. Нехай там маємо нульове сміття (ми це не писали, машина Ноймана сприймає це сміття за бінарний код машинної інструкції і починає його виконувати. Проте виконання наступної машинної інструкції є іншою історією. Що нас зараз не цікавить.

В результаті ми отримали наступну мікропрограму виконання машинної інструкції пересилання:

```
МК1: memory(0000) → IR.
МК2: PC + 1  → PC.
МК3: memory(1110) → ACC.
МК4: ACC → memory(1111).
```

Цим дослідження мікропрограми виконання машинної інструкції пересилання вмістимого однієї комірки пам'яті до іншої завершено.

Про пристрій керування процесора

Процесор комп'ютера складають з двох автоматів:

1. Операційного автомата (його складові і стан відбиває вікно симулятора,
2. Керуючого автомата (він у вікні симулятора поданий фрагментарно, а саме, регістром машинних інструкцій РК (IR або R0) та лічильником машинних інструкцій Такт (PC).

Всі мікропрограми, а їх ми маємо вісім за числом різних машинних інструкцій комп'ютера, вміщує керуючий автомат (пристрій керування). Щодо мікропрограм, то важливим є наступне:

1. Пристрій керування, що використовує техніку мікропрограмного виконання машинних інструкцій, називають мікропрограмним.
2. Мікропрограмний пристрій керування може містити додаткову, так звану мікропрограмну пам'ять, аби зберігати мікрокоди мікропрограм (тобто, бінарно кодовані мікропрограми). Кажуть мікропрограмний пристрій керування з мікропрограмною пам'яттю побудований на принципі *гнучкої логіки* (адже можна за потреби без зміни схеми керування міняти лише коди мікропрограм, перепрограмовувати мікропрограмну пам'ять). Пам'ять мікропрограм оригінально називають control memory. Ця мікропрограмна пам'ять існує окремо від основної пам'яті (main memory) і ніяк з нею не пов'язана. Мікропрограмна пам'ять функціонує швидко, на частоті процесора, тому її місткість обмежена і не перевищує рівня кілобайтів або десятків кілобайтів (до 32 КБ).
3. Мікрокоди, що за своєю сутністю збігаютє постійною пам'яттю (в штатному використанні процесора мікорокоди не змінюють), можна імплементувати в пристрої керування ще як комбінаційну схему. В цьому випадку кажуть, що мікропрограмний пристрій керування реалізований на *жорсткій логіці* (або просто, що це є апаратний, а не мікропрограмний пристрій керування). Математичеими моделями апаратного пристрою керування є автомати Уілкса-Стринджера, Мура, Мілі.
4. В будь-якому варіанті, треба розрізняти основну і мікропрограмну пам'ять. Основна пам'ять знаходиться поза межами процесора та йому не належить. Мікропрограмна пам'ять знаходиться в процесорі, точніше, в його пристрої керування і вона належить процесору.

5. Аби виконати машинну інструкцію (команду) потрібно виконати відповідну інструкції мікропрограму. На початку виконання нової машинної інструкції пристрій керування безумовно (але спираючись на вмістиме лічильника інструкцій) вибирає на регістр інструкції (РК, IR) бінарний інструкції. При цьому визначаються три праві біти кода операції. На основі наповнення цих трьох бітів вибирають і виконують відповідну коду операції мікропрограму.
6. Об'єднану граф-схему мікроалгоритму виконання всіх машинних інструкцій подано рис. 4.7.

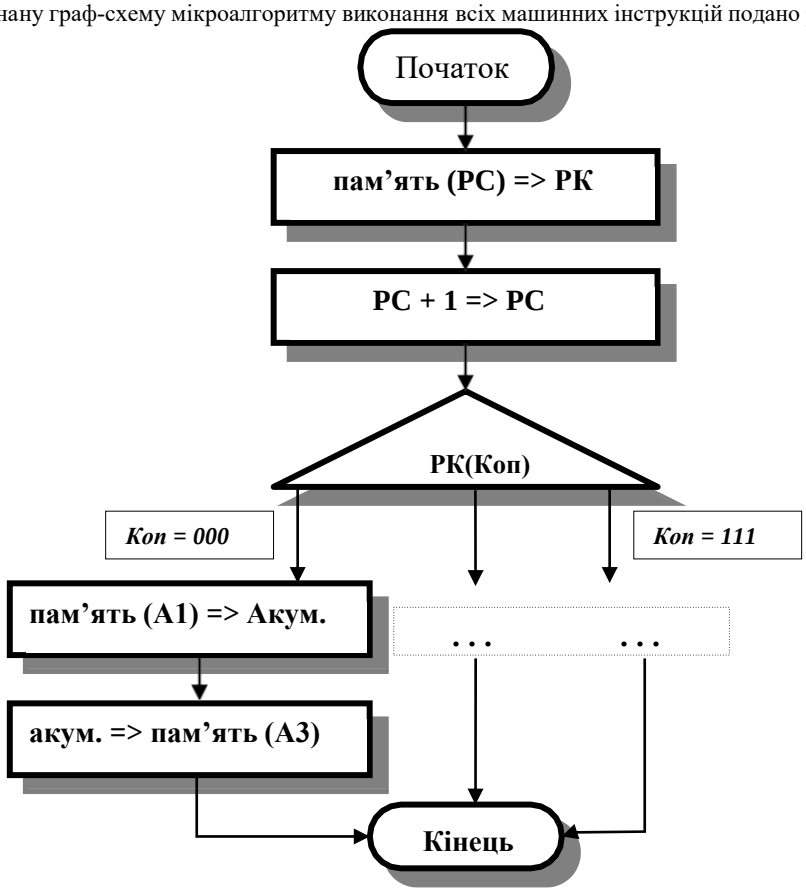

Рис. 2.9 – Об'єднана граф-схема (ОГС) мікропрограмного керування. ОГС містить вершини чотирьох типів, а саме:

- початкову з одним виходом,
- кінцеву з одним входом,
- операторні з одним входолм та одним виходом,
- умовну з одним входом та багатьма (класично лише з двома) виходами,

Після кінцевої вершини слідує початкова вершина аби забезпечити керуванням не одну ізольовану інструкцію, а тривалий потік цих інструкцій (Король помер. Хай живе король!). Один прохід через ОГС викликає виконання чергової машинної інструкції. В нас умовна вершина повинна мати вісім виходів (по числу різних машинних інструкцій) і, відповідно, повний граф містить вісім паралельних гілок. Але ми детально розписали лише першу гілку, яка відповідає керуванню, потрібному для виконання машинної інструкції пересилання, що має код операції Коп = 000. Важливо, що дві перші операторні вершини не аналізують Коп і виконуються безумовно для будь-якого типу машинної інструкції. Далі кожній інструкції потрібні свої мікродії. Тому в ОГС на третьому місці стоїть умовна вершина, що опитує Коп і, відповідно, направляє подальше керування до однієї з вісьми паралельних гілок.

Студент повиннен дослідити поведінку комп'ютера і скласти повну ОГС мікропрограмного керування.

Зауважимо, що сучасні засоби автоматизованого керування комп'ютерних апаратних засобів (САПР) після уведення до них ОГС в графічній формі автоматично (без участі людини) синтезують функційну і принципову схеми, на основі яких також автоматично імплементує цей граф в «залізі».

Варіанти завдань на лабораторну роботу №2

Створити програму у машинних кодах для обчислення виразу згідно наведених варіантів. Результат виконання має виводитися у вікно результату. Для змінних та констант визначити відповідні комірки пам'яті. Персональні варіанти завдань знайдете у табличці з варіантами.

Додаток 01. Задачі і програми-відповіді для симулятора комп'ютера

Подані нижче програми, як і окремі фрагменти цих програм, можна використовувати під час виконання лабораторних робіт, аби отримати індивідуальні завдання на симуляцію програм комп'ютера «Кроха».

Задача № 01. Скласти програму обчислення периметру прямокутника за наданими довжина сторін.
Алгоритм. Позначимо сторони прямокутника як a та b, а результат - P. Обчислення виконуємо за алгоритмом:
P=a+b;
P=P+P.
Саме цьому алгоритму відповідає ефективна (за критерієм числа використаних комірок пам'яті) програму.

Програма

```
Адреса     Інструкція    Дія інструкції            Комментар
000   001 110 111 101   (6)+(7) ==> (5)           a+b ==> P
001   001 101 101 101   (5)+(5) ==> (5)           P+P ==> P
010   111 110 111 101   стоп; вивід (6),(7),(5)   вивести на екран a,b,P
011   не використовується
100   не використовується
101         P                                     результат
110         b                                      операнд
111         a                                      операнд
```

Задача № 02. Скласти програму обчислення половини повноїповерхні паралеліпіпеда по довжинах ребер.
Алгоритм. Позначимо ребра паралеліпіпеда як a, b и c, а шуканий результат - s. Обислення проведемо за формулою:
s = ab+bc+ac = b(a+c)+ac,
Що зменшить кількість арифметичних дій, тобто число машинних інструкцій в програмі. Надану формулу для зручності складання програми перепишемо двома частинами:
r = b(a+c); s = ac+r.

Програма

```
Адреса     Інструкція    Дія інструкції              Коментар
000   001 101 111 000   (5)+(7) ==> (0)          a+c ==> r  [ r ]
001   101 110 000 000   (6)*(0) ==> (0)          b*r ==> r  [ s ]
010   101 101 111 001   (5)*(7) ==> (1)          a*c ==> s
011   001 000 001 001   (0)+(1) ==> (1)          s+r ==> s
100   111 001 001 001   стоп; вивід (1),(1),(1)  виведення результату s
101         a                                    операнд
110         b                                    операнд
111         c                                    операнд
```

Особливістю задачі є нестача комірок пам'яті для виконання обчислень. Через це вимушені поміщати робочу змінну r і результат s на місце тих інструкцій програми, що вже відпрацювали. Неможна казати, що це зручно (адже перед кожним перезапуском програми перші інструкції треба поновлювати). Проте іншого способу програмування для вісьми комірок пам'яті немає.

Задача № 03. Скласти програму знаходження більшого з двох чисел, що зберігаються в комірках пам'яті.
Алгоритм. Позначимо вихідні числа a та b, а результат max.

Програма

```
Адреса     Інструкція         Дія інструкції                Коментар
000   110 110 111 011   если (6)>(7), перейти на 3      порівняти a та b
001   000 111 000 101   (7) ==> (5)                     a ==> max
010   111 111 110 101   стоп; вивід (7),(6),(5)        виведення a,b,max
011   000 110 000 101   (6) ==> (5)                     b ==> max
100   111 111 110 101   стоп; вивід (7),(6),(5)        виведення a,b,max
101        max                                          результат
110         b                                           операнд
111         a                                           операнд
```

Зауваження. Замість інструкції «стоп» за адресою 010 можна виконати безумовний перехід на інструкцію 100. На перший погляд інструкції безумовного перходу у «Крохи» нема. Але проаналізуємо інструкцію
100 000 000 100.
Розтлумачемо інструкцію. Коли (000_2)=(000_2), тоді виконати перехід на (100_2), *безумовно*, адже вмістиме нульової комірки пам'яті завжди дорівнює собі. Так і отримують бузумовний перехід.

Задача № 04. Написати програму ділення двох чисел з врахуванням можливості діелення на нуль (в цьому випадку відобразити на екрані всі нулі).
Алгоритм. Позначимо вихідні числа як a та b, а результат – як d.

Програма

Адреса	Інструкція	Дія інструкції	Коментар
000	100 101 111 011	если (5)=(7), перейти на 3	порівняти b з 0
001	010 110 101 100	(6)/(5) ==> (4)	a/b ==> d
010	111 110 101 100	стоп; вивід (6),(5),(4)	виведення a, b, d
011	111 111 111 111	стоп; вивід (7),(7),(7)	виведення 0, 0, 0
100	d		результат
101	b		операнд
110	a		операнд
111	0		константа 0

Задача № 05. Надані два числа a та b. Написати програму, що більше з них ділить на менше.
Алгоритм. Позначимо результат як d.

Програма

Адреса	Інструкція	Дія інструкції	Комментар
00	110 110 111 011	если (6)>(7), перейти на 3	порівняти a та b
001	010 111 110 101	(7)/(6) ==> (5)	a/b ==> d
010	111 111 110 101	стоп; вивід (7),(6),(5)	виведення a, b, d
011	010 110 111 101	(6)/(7) ==> (5)	b/a ==> d
100	111 111 110 101	стоп; вивід (7),(6),(5)	виведення a, b, d
101	d		результат
110	b		операнд
111	a		операнд

Задача № 06. Написати програму обчислення **n!**

Алгоритм. Позначимо як k робочу змінну, що є поточним множником для факторіалу і змінюється в межах від 1 до n. Початкове значення для k змушені задавати перед кожним запуском «вручну», адже для інструкції пересилання константи 1 з комірки 7 до комірки 4 вже не вистачає пам'яті. Те саме можна зауважити і про початкове значення n!, яке перед запуском природньо встановити рівним одиниці. Також треба до комірки 6 занести значення n+1, що є верхньою границею циклу (цикл виконуватиметься доти, доки k < n+1, тому завершиться після множення на n).

Програма

Адреса	Інструкція	Дія інструкції	Коментар
000	101 101 100 101	(5)*(4) ==> (5)	n! * k ==> n!
001	001 100 111 100	(4)+(7) ==> (4)	k+1 ==> k
010	110 110 100 000	коли (6)>(4), тоді перейти на 0	k < n+1 ?
011	111 101 101 101	стоп; виведення (5),(5),(5)	виведення n!
100	k [задати 1]		робоча комірка
101	n! [задати 1]		результат
110	n+1		константа
111	1		константа 1

При роботі з програмою корисно звернути увагу на ефект переповнення, який для швидкозростаючого виразу типу факториал досягають досить швидко. Дійсно, максимальне допустиме число для 12-разрядного варіанту симулятора дорівнює 4095 (а для 15 розрядного варіанту потрібно разрахувати самому). Отже, вже спроба обчислити 7!=1*2*3*4*5*6*7=5040 змусить симулятор надати некоректну відповідь. Потрібно на власні очі побачити, як симулятор реагує на переповнення під час виконання цієї програми.

Задача № 07. Написати програму обчислення виразу **1+2+3+4+...+n**

Задача схожа на попередню. Але зауважимо, що початкове значення суми (на відміну від початкового значення факторіалу) потрібно задавати нулем.

Задача № 07. Написати програму обчислення виразу
X*2*2*...*2.
Розв'язування задачі полягає в подвоєнні значення X n разів, що зручно здійснити в спосіб циклічного складання комірки з « самою собою».

Програма

```
Адреса    Інструкція       Дія інструкції              Коментар
000   001 101 101 101   (5)+(5) ==> (5)            X + X ==> X
001   011 100 110 100   (4)-(6) ==> (4)            n-1 ==> n
010   110 100 111 000   коли (4)>(7),перейти на 0  n > 0 ?
011   111 101 101 101   стоп; вивід (5),(5),(5)    виведення X
100         n                                       операнд
101         X                                       результат
110         1                                       константа 1
111         0                                       константа 0
```

Задача № 09. Обчислити вираз **y=1*2*4*8*...*n**
Алгоритм. Ясно, що для обчислення у програма повинна підсумовувати послідовні степені числа 2. Позначимо чергову степінь як s.
Зауважимо, що верхня границя циклу в комірці 7 перевстановлюється програмою. Перед пуском уводимо до цієї комірки потрібне конкретне значення n. Проте симулятор це значення «не влаштовує»: йому потрібно, аби верхняя границя дорівнювала 2*n, лише за цієї умови останній врахований в циклі множення співмножник дорівнюватиме n. Через це першою машинною інструкцією симулятор подвоює вмістиме сьомої комірки.

Програма

```
Адреса     Інструкція      Дія інструкції            Коментар
000   001 111 111 111   (7)+(7) ==> (7)              n+n ==> n
001   101 101 110 110   (5)*(6) ==> (6)              s * y ==> y
010   001 101 101 101   (5)+(5) ==> (5)              s+s ==> s
011   110 111 101 001   коли (7)>(5), перейти на 1   s < 2*n ?
100   111 110 110 110   стоп; виведення (6),(6),(6)  виведення y
101    s  [задать 1]                                  робоча комірка
110    y  [задать 1]                                  результат
111       n, 2*n                                      границя циклу
```

Задача № 11.. Задача про самомодифікуючу програму (заборонена техніка поліморфних вірусів). Надано стан пам'яті симулятора перед пуском (див. нижче). Знайти результат виконання програми.
Коментар. Задача має і теоретичний аспект. Задача ілюструєте те, що:

- комп'ютер може сам собі формувати програму,
- вмістиме тої самої комірки пам'яті в різні часові моменти може трактуватися порізному, а саме, інколи як код числа, а інколи як код машинної інструкції.

Початковий стан пам'яті

```
Адрес     Команда          Расшифровка            Комментарий
000   001 111 110 001   (7)+(6) ==> (1)           формуємо (1)
01   не имеет значения
010  не имеет значения
011   001 001 010 011   сума комірок (6) та (7) надає
100   110 000 000 000   команду стоп для (3)
101   001 011 100 011   (3)+(4) ==> (3) для (2)
110   000 101 000 000   сум комірок (6) та (7) надає
111   000 000 000 010   команду (5) ==> (2) для (1)
```

Кінцевий (модифікований) стан пам'яті

```
Адреса     Інструкція     дія інструкції            Коментар
000   001 111 110 001   (7)+(6) ==> (1)            формируем (1)
001   000 101 000 010   (5) ==> (2)                формируем (2)
010   001 011 100 011   (3)+(4) ==> (3)            формируем (3)
011   111 001 010 011   стоп; вывод (1), (2), (3)
100   110 000 000 000                             константы со-
101   001 011 100 011                             храняются без
110   000 101 000 000                             изменения кро-
111   000 000 000 010                             ме ячейки (3)
```

В результаті виконання програми на екран симулятора виводяться вмістимі комірок (1), (2) і (3).

Задача № 12. Скласти циклічну програму, що записує одиницю до комірок (4)-(6).

Програма

```
Адреса    Інструкція      Алгоритм інструкції         Коментар
000   000 111 000 100   (7) ==> (4)                  1 ==> ячейку
001   001 111 000 000   (7)+(0) ==> (0)              модиф. адрес
010   110 111 110 000   если (7)>(6), перейти на 0    (6) < 1 ?
011   111 100 101 110   стоп; вивід (4),(5),(6)      вывод ячеек
100  не має значення                                 [сюди
101  не має значення                                  пишемо
110     [задати  0]                                   одиниці]
111   000 000 000 001                                константа 1
```

Для перевірки завершення циклу використовуємо наступне. Перед запуском до комірки (6) заносять 0, тому під час перевіркм умови і інструкції (2) перехід спрацьовує, адже 1>0. Так продовжується до запису до комірки (6) одиниці. Після цього умова припиняє виконуватися і цикл припиняється.

Подані і цьому додатку задачі і програмні розв'язки до них склав доцент Пермського педагогічного університету Е.А.Єрьомін. Файл є додатком до симулятора «Кроха», автором якої є Е.А.Єрьомін.

Література до першої частини лабораторних робіт

1. Еремин Е.А. Популярные лекции об устройстве компьютера. СПб.: BHV-Петербург, 2003, 272 с.
2. Абель П. Язык Ассемблера для IBM PC и программирования. М.: Высш. шк., 1992, 447 с.
3. Еремин Е.А. Моделирование работы ЭВМ с помощью программы Microsoft Excel. Информатика, 2006, N 21, с.37-40; N 22, с.42-45.
4. Гейн А.Г., Житомирский В.Г., Линецкий Е.В. и др. "Основы информатики и вычислительной техники" (1989).