

## ЛР № 3. Робота з симулятором машини Ноймана. Дослідження виконання асемблерної програми симулятора.

**Мета:** опанувати роботу асемблера на симуляторі машини Ноймана, зрозуміти і дослідити принцип виконання програми машиною Ноймана.

**Завдання:** скласти програму на асемблері, перетворити її у машинні коди, запустити симулятор, увести до нього коди машинних, проаналізувати і пояснити отримані результати, скласти звіт з виконання лабораторних досліджень та захистити його.

### Теоретичні відомості

В ході виконання даної лабораторної роботи розглядається симулятор та асемблер 32 розрядного CISC комп'ютера. Даний комп'ютер містить 8 регістрів (0 вий регістр завжди зберігає 0) та 65536 комірок пам'яті по 32 розряди. Набір машинних інструкцій складається з 8 команд (таблиця 3.1).

Таблиця 3.1. Множина інструкцій

№ пп	Код інструкції	2ко ве	СУТНІСТЬ ІНСТРУКЦІЙ МАШИНИ
Інструкції R-типу			
1	add	000	Додає вміст регістру regA до вмісту regB, та зберігає в destReg
2	nand	001	Виконує логічне побітове І-НЕ вмісту regA з вмістом regB, та зберігає в destReg
I-тип			
3	lw	010	Завантажує regB з пам'яті. Адреса пам'яті формується додаванням зміщення до вмісту regA.
4	sw	011	Зберігає вміст регістру regB в пам'ять. Адреса пам'яті формується додаванням зміщення до вмісту regA.
5	beq	100	Якщо вміст регістрів regA та regB однаковий, виконується перехід на адресу програмний лічильник(ПЛ) + 1+зміщення, в ПЛ зберігається адреса поточної тобто beq інструкції.
J-тип			
6	jalr	101	Спочатку зберігає ПЛ+1 в regB, в ПЛ адреса поточної (jalr) інструкції. Виконує перехід на адресу, яка зберігається в regA. Якщо в якості regA regB задано один і той самий регістр, то спочатку в цей регістр запишеться ПЛ+1, а потім виконається перехід до ПЛ+1.
O-тип			
7	halt	110	Збільшує значення ПЛ на 1, потім припиняє виконання,

			стимулятор має повідомляти, що виконано зупинку.
8	poor	111	Нічого не виконується

Формат лінійки асемблерного коду наступний (<пробіл> означає послідовність табуляцій і/або пробілів):

*мітка <пробіл> інструкція<пробіл>поле№1<пробіл> поле№2<пробіл> поле№3<пробіл>коментар*

Крайнє ліве поле лінійки асемблерного коду – поле мітки. Коректна мітка має складатися максимуму з 6 символів, символами можуть бути літери або цифри, але починатися з букви. Поле мітки є необов’язковим, проте пробіл після даного поля є обов’язковим. Після не обов’язкової мітки іде обов’язковий пробіл. Далі іде поле назви інструкції, в якому може бути ім’я будь якої асемблерної інструкції зазначені вище в таблиці. Після пробілів ідуть відповідні поля. Всі поля можуть зберігати або десяткові значення або мітки. Кількість полів залежить від інструкції, поля які не використовуються ігноруються (подібно до коментарів).

Інструкції r-типу (add, nand) потребують наявності 3 полів: поле№1 – regA, поле№2 regB, поле№3 destReg.

Інструкції i-типу (lw,sw,beq) вимагають 3 полів: поле№1 – regA, поле№2 regB, поле№3 – числове значення зміщення чи символічна адреса. Числове значення може бути як додатнім так і від’ємним. Символьні адреси описані нижче.

Інструкція J-типу (jalr) вимагає 2 полів: поле№1 – regA, поле№2 regB

Інструкція O-типу (poor, halt) не вимагає жодного.

Символьні адреси посилаються на відповідні мітки. Для інструкцій lw та sw асемблер має згенерувати зміщення, яке дорівнює адресі мітки. Вона може використовуватися з 0 регістром, тоді буде посилання на мітку, або може використовуватися з не нульовим базовим регістром у якості індексу масиву, який починається з мітки. Для інструкції beq, асемблер має перетворити мітку в числове зміщення куди має відбуватися перехід. Після останнього поля має йти пробіл за яким може розміщуватися коментар. Коментар закінчується з кінцем лінії асемблерної програми.

Крім інструкцій асемблерна програма може містити директиву - .fill (зверніть увагу на точку попереду). Директива . fill повідомляє компілятору про те, що він має зберегти число за адресою відповідно де дана інструкція знаходиться. Директива .fill використовує одне поле, в якому може бути як число так і символічна адреса. Наприклад «.fill 32» означає зберегти число 32 за адресою де дана інструкція знаходиться. (Оскільки в нас кожен рядок програми відповідає адресі починаючи з 0, то відповідно адреса буде дорівнювати номеру рядка - 1). Директива . fill з символічною адресою збереже адресу даної мітки. В прикладі нижче ".fill start" збереже значення 2, тому що мітка start знаходиться за адресою 2.

Приклад асемблерної програми.

Нехай наявна асемблерна програма prog1.as (її потрібно набирати в notepad, не допускаючи вільних рядків перед початком тесту, максимально наближуючись до формату поданого нижче тексту; спочатку отриманий файл з тестом програми має розширення .txt, яке треба замінити на .as) містить наступний текст:

```

        lw      0      1      five  load reg1 with 5 (symbolic address)
        lw      1      2      3     load reg2 with -1 (numeric address)
start   add     1      2      1     decrement reg1
        beq     0      1      2     goto end of program when reg1==0
        beq     0      0      start go back to the beginning of the loop

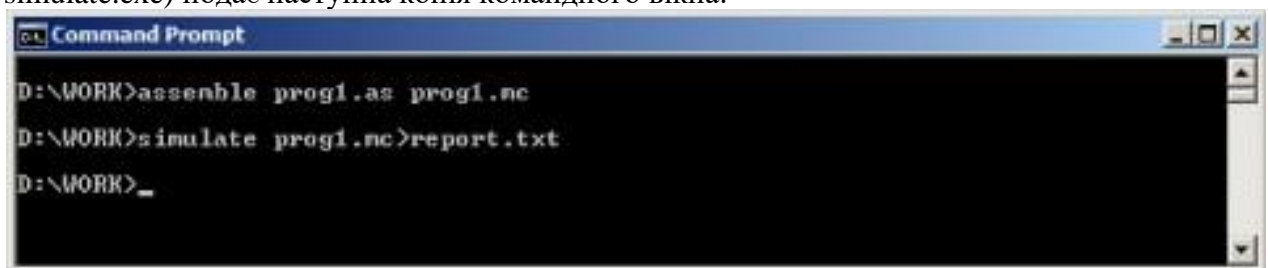
```

	noop	
done	halt	end of program
five	.fill 5	
neg1	.fill -1	
stAddr	.fill start	will contain the address of start (2)

Отриманий асемблюванням відповідний асемблерній програмі файл з вісім розрядним гексадецимальним машинним кодом цієї програми має назву prog1.mc та наступне наповнення:

```
8454151
9043971
655361
16842754
16842749
29360128
25165824
5
-1
2
```

Послідовність виконання асемблювання та симуляції (за допомогою програм assemble.exe та simulate.exe) подає наступна копія командного вікна:



```

Command Prompt
D:\WORK>assemble prog1.as prog1.mc
D:\WORK>simulate prog1.mc>report.txt
D:\WORK>_

```

Отримуємо наступний результат у файл report.txt

```
memory[0]=8454151
memory[1]=9043971
memory[2]=655361
memory[3]=16842754
memory[4]=16842749
memory[5]=29360128
memory[6]=25165824
memory[7]=5
memory[8]=-1
memory[9]=2
```

```
@@@
state:
pc 0
```

```
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
```

```

mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 0
reg[ 2 ] 0
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 1
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 5
reg[ 2 ] 0
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 2
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 5
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 3
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 4
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 4
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 4
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 2
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 4
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 3
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 3
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 4
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 3
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 2
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 3
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 3
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 2
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 4
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 2
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 2
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 2
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 3
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 1
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 4
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 1
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 2
memory:
mem[ 0 ] 8454151

```

```

mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 1
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 3
memory:
mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2
registers:
reg[ 0 ] 0
reg[ 1 ] 0
reg[ 2 ] -1
reg[ 3 ] 0
reg[ 4 ] 0
reg[ 5 ] 0
reg[ 6 ] 0
reg[ 7 ] 0
end state

```

```

@@@
state:
pc 6
memory:
mem[ 0 ] 8454151

```

mem[ 1 ] 9043971		reg[ 4 ] 0
mem[ 2 ] 655361		reg[ 5 ] 0
mem[ 3 ] 16842754		reg[ 6 ] 0
mem[ 4 ] 16842749		reg[ 7 ] 0
mem[ 5 ] 29360128	<b>end state</b>	
mem[ 6 ] 25165824		
mem[ 7 ] 5		
mem[ 8 ] -1		
mem[ 9 ] 2		
registers:		
reg[ 0 ] 0		
reg[ 1 ] 0		
reg[ 2 ] -1		
reg[ 3 ] 0		
reg[ 4 ] 0		
reg[ 5 ] 0		
reg[ 6 ] 0		
reg[ 7 ] 0		
end state		

**machine halted**  
**total of 17 instructions executed**  
**final state of machine:**

@ @ @

**state:**

**pc 7**

**memory:**

```

mem[ 0 ] 8454151
mem[ 1 ] 9043971
mem[ 2 ] 655361
mem[ 3 ] 16842754
mem[ 4 ] 16842749
mem[ 5 ] 29360128
mem[ 6 ] 25165824
mem[ 7 ] 5
mem[ 8 ] -1
mem[ 9 ] 2

```

**registers:**

```

reg[ 0 ] 0
reg[ 1 ] 0
reg[ 2 ] -1
reg[ 3 ] 0

```



У виводі показано значення лічильника команд, стан всіх регістрів та задіяної пам'яті для кожного кроку виконання програми. В кінці дається статистика та кінцевий стан машини.

### ***Хід виконання роботи:***

1. Створити тестовий документ у notepad.
2. Написати асемблерний код програми згідно індивідуального завдання.
3. Зберегти текстовий файл з розширенням \*.as.
4. Перетворити асемблерний код у машинний. Для цього у консольному вікні запустити програму *assemble.exe*. Наприклад:

***assemble prog1.as prog1.mc***

***prog1.as*** – ім'я вхідного файлу з асемблерним кодом

***prog1.mc*** – ім'я вихідного файлу з машинним кодом

*Примітка. Файл програми та асемблерного коду мають бути в одному каталозі, в інакшому випадку необхідно крім імені файлу вказувати шлях до нього.*

5. Запустити симулятор з отриманим у п.4 машинним кодом. Наприклад:

***simulate.exe prog1.mc>result.txt***

***result.txt*** – буде зберігати покроковий вивід станів машини в ході виконання машинних інструкцій.

6. Проаналізувати хід виконання машинних інструкцій, перевірити правильність результатів.
7. Скласти звіт по результатам виконання програми.

### ***Завдання***

Скласти програму на асемблерній мові симулятора, яка обчислює вираз згідно варіанта. Персональні варіанти завдань знайдете у таблиці з варіантами.