Київський національний університет
імені Тараса Шевченка

Звіт

до лабораторної роботи №2
на тему:

«Імітаційна модель процесора»

Студента другого курсу
Групи К-22
Факультету комп'ютерних
наук та кібернетики
Коваля Максима

Київ

2020

Мета

Метою даної лабораторної роботи є розробка програмної моделі процесора та реалізування його імітаційної (тобто комп’ютерної) моделі.

Основні принципи виконання роботи

Для виконання даної лабораторної роботи було обрано мову програмування Java (SE 8). Також було використано основні принципи ООП, що можна побачити у Додатку.

Згідно з варіантом мені було запропоновано розробити імітаційну модель 1-адресного 18-бітного процесора з обов’язковою операцією інвертування, тобто зміною значення з 0 на 1, чи з 1 на 0 лише парних або непарних бітів в регістрі в залежності від значення в другому операнді команди.

Окрім обов'язкової команди інвертування (inv), занесення значення в акумулятор (mov) та перенесення значення, що зберігається у акумуляторі, в регістр (save) мною було реалізовано наступний перелік команд: додавання (add), віднімання (sub), остача від ділення (mod), логічне множення (and) та логічне додавання (or).

Також до програмної реалізації було додано наступні регістри: регістр команди (зберігає команду для виконання та її операнди), 4 регістри даних (зберігають бінарне представлення числа), регістр стану (містить знак останнього результату та ознаку переповнення), регістр лічильника команд (зазначає, яка команда за рахунком виконується) та регістр лічильника тактів (зазначає, який такт за рахунком виконується)

Щодо самої роботи програми, то файл з заданими інструкціями обробляється в реальному часу (зчитування інструкції - її виконання). Кожна команда виконується у два такти: перший - занесення команди у регістр команди, другий - виконання відповідної команди . Слід зазначити, що відрахунок тактів

регулюється користувачем: кожен такт починає своє виконання лише після натиску клавіші Enter. Для регістру стасусу було використано наступну логіку: у випадку переповнення на момент внесення команди та операнду в регістр команди, дана команда просто пропускається; у випадку переповнення на момент виконання відповідної операції це відзначається у регістрі стану, а у самому результаті відкидаються біти, котрі сприяють переповненню.

Також слід зазначити, що дотримання принципів ООП дозволяють легко модифікувати програму. Для того, щоб змінити бітність процесора, робота якого імітується, необхідно лише змінити значення однієї змінної, після чого програма підлаштується під бажаний розмір. Також доволі легко добавити нові команди: для цього необхідно виконати наслідування класу ValueCommand (для команд, котрі виконують в акумуляторі) чи класу OperandCommad (для команд, котрі працюють з іншими регістрами). Аналогічно для добавлення регістрів нових типів потрібно віднаслідуватися від відповідного класу (залежить, від потреб регістру).

Результати роботи програми

Для демонстрації роботи програми було сформовано файл з наступними інструкціями:

```
mov 10
save R1
mov 10
inv 1
save R2
```

При запуску програми з заданим параметром, котрий містить шлях до файлу, котрий був описаний раніше, отримали наступні результати:

```
Command = mov 10
R1 = 00 0000 0000 0000 0000         IR = mov | 00 0000 0000 0000 1010
R2 = 00 0000 0000 0000 0000         PC = 1
R3 = 00 0000 0000 0000 0000         TC = 1
R4 = 00 0000 0000 0000 0000         PS = 0 | 0
A  = 00 0000 0000 0000 0000

Command = mov 10
R1 = 00 0000 0000 0000 0000         IR = mov | 00 0000 0000 0000 1010
R2 = 00 0000 0000 0000 0000         PC = 1
R3 = 00 0000 0000 0000 0000         TC = 2
R4 = 00 0000 0000 0000 0000         PS = 0 | 0
A  = 00 0000 0000 0000 1010

Command = save R1
R1 = 00 0000 0000 0000 0000         IR = save | R1
R2 = 00 0000 0000 0000 0000         PC = 2
R3 = 00 0000 0000 0000 0000         TC = 1
R4 = 00 0000 0000 0000 0000         PS = 0 | 0
A  = 00 0000 0000 0000 1010

Command = save R1
R1 = 00 0000 0000 0000 1010         IR = save | R1
R2 = 00 0000 0000 0000 0000         PC = 2
R3 = 00 0000 0000 0000 0000         TC = 2
R4 = 00 0000 0000 0000 0000         PS = 0 | 0
A  = 00 0000 0000 0000 1010
```

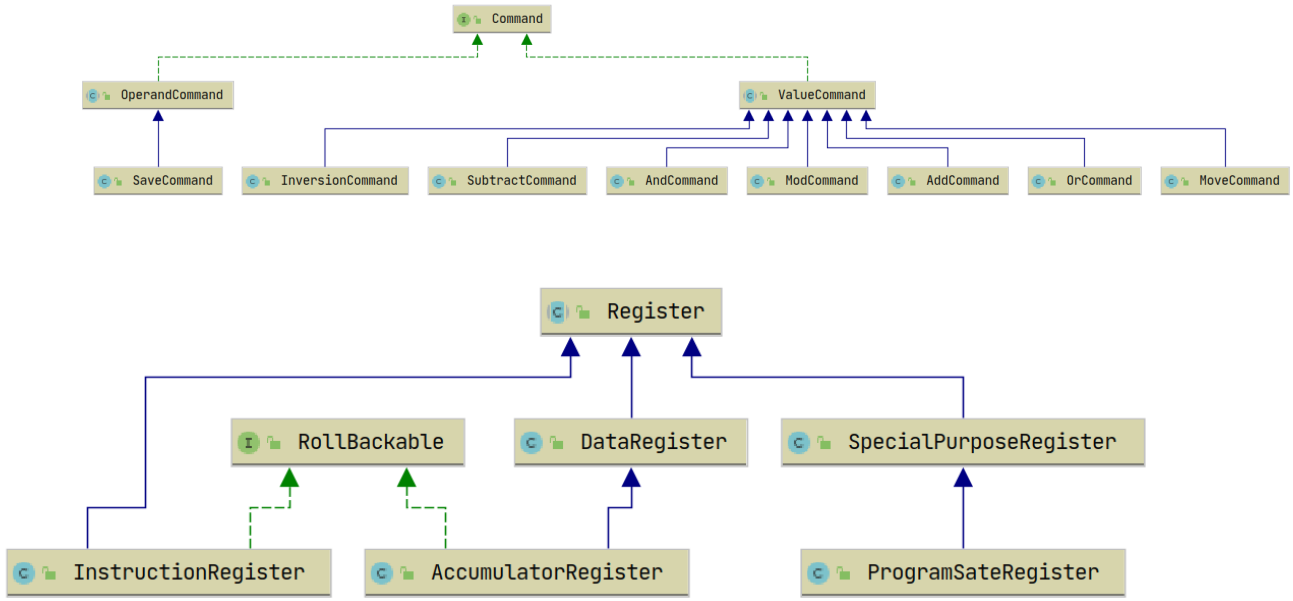
```
Command = mov 10
R1 = 00 0000 0000 0000 1010          IR = mov | 00 0000 0000 0000 1010
R2 = 00 0000 0000 0000 0000          PC = 3
R3 = 00 0000 0000 0000 0000          TC = 1
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 00 0000 0000 0000 1010

Command = mov 10
R1 = 00 0000 0000 0000 1010          IR = mov | 00 0000 0000 0000 1010
R2 = 00 0000 0000 0000 0000          PC = 3
R3 = 00 0000 0000 0000 0000          TC = 2
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 00 0000 0000 0000 1010

Command = inv 1
R1 = 00 0000 0000 0000 1010          IR = inv | 00 0000 0000 0000 0001
R2 = 00 0000 0000 0000 0000          PC = 4
R3 = 00 0000 0000 0000 0000          TC = 1
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 00 0000 0000 0000 1010

Command = inv 1
R1 = 00 0000 0000 0000 1010          IR = inv | 00 0000 0000 0000 0001
R2 = 00 0000 0000 0000 0000          PC = 4
R3 = 00 0000 0000 0000 0000          TC = 2
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 01 0101 0101 0101 1111

Command = save R2
R1 = 00 0000 0000 0000 1010          IR = save | R2
R2 = 00 0000 0000 0000 0000          PC = 5
R3 = 00 0000 0000 0000 0000          TC = 1
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 01 0101 0101 0101 1111

Command = save R2
R1 = 00 0000 0000 0000 1010          IR = save | R2
R2 = 01 0101 0101 0101 1111          PC = 5
R3 = 00 0000 0000 0000 0000          TC = 2
R4 = 00 0000 0000 0000 0000          PS = 0 | 0
A  = 01 0101 0101 0101 1111
```

Висновок

Під час виконання даної лабораторної роботи були виконані всі вимоги до її виконання, а саме: було реалізовано 1-адресний 18-бітний процесор з командами інвертування, занесення значення в акумулятор та зберігання значення, що зберігається у акумуляторі, в регістр. Також було реалізовані додаткові команди, такі як: додавання, віднімання, остача від ділення, логічне множення та логічне додавання. Також було реалізована логіка регістрів команди, даних, стану, лічильника команд та лічильника тактів. Разом з реалізацією регістру стану було реалізовано логіку при переповненні перед чи після виконання команди.

Додаток

Програма має наступну структуру (UML Diagram):



Посилання на репозиторій GitHub, на котрому розміщено вихідні коди програми - https://github.com/maxym-ko/cyb_java