Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Звіт
з лабораторної роботи № 2
на тему:

«Імітаційна модель процесора»

Варіант 1, 4, 10

Студента другого курсу
групи К-23(2)
Флакей Романа Руслановича
Факультету комп’ютерних наук
та кібернетики

Київ – 2022

1. Постановка задачі

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).

Рекомендації щодо виконання роботи

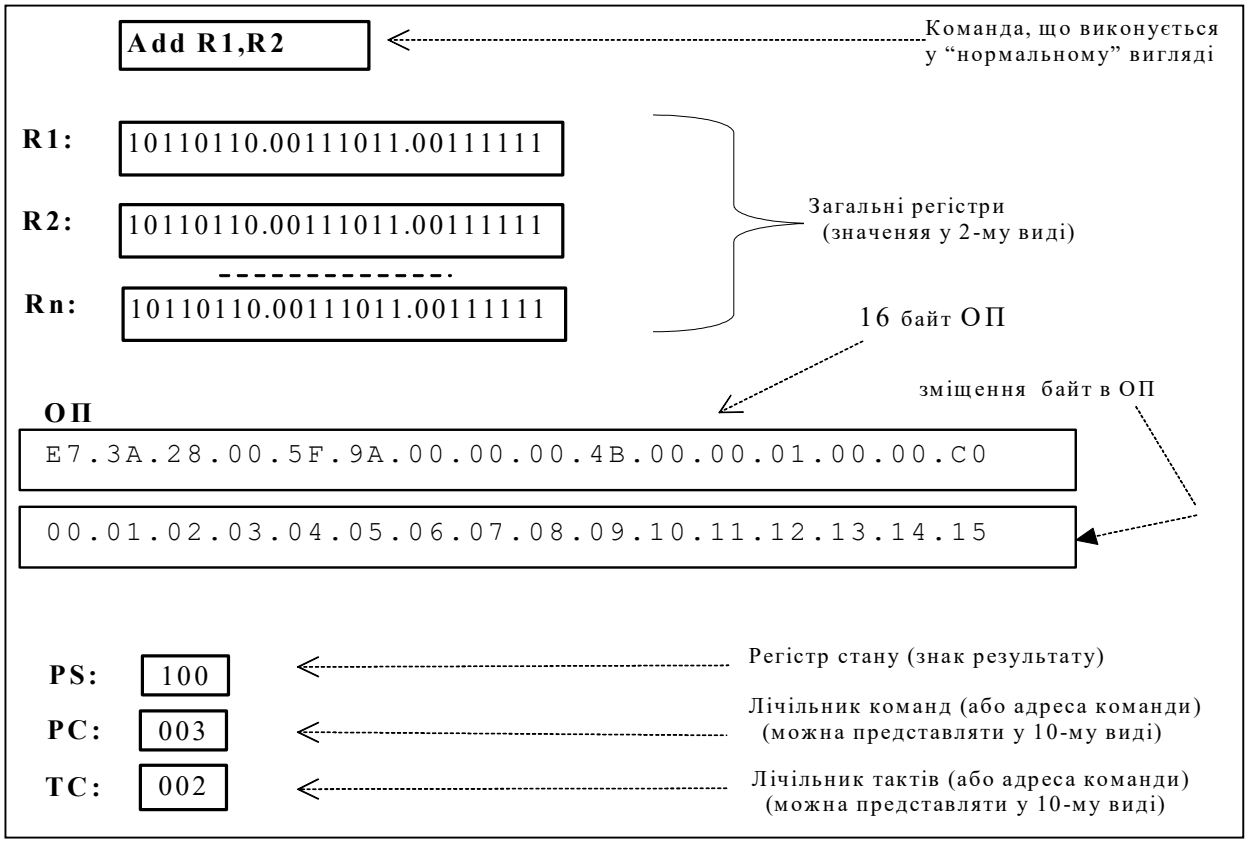
1. Щодо вибору кількості тактів для команд - мінімально достатньо двох: 1-й) занесення поточної команди у регістр команди; 2-й) виконання команди.

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

Команда	Коментар	Демонстрація дій процесора
		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 **R2** = 1111 1011 **PC** = 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 **PC** = 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 **PC** = 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 **PC** = 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 **PC** = 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 **PC** = 4 **R3** = 1100 0111 **TC** = 2 **R4** = 0111 1011 **PS** = 1

Задача варіанту

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

4.	18-бітні	

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

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.

2. Реалізація

Регістри і оперативна пам’ять

По умові у нас 1-адресний процесор, з 18-бітністю. Загалом в моделі використовуються такі регістри

```
R1: 110011111011101001
Memory:
    00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
    00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19
PS: 0            INS:
PC: 1
TC: 1
```

- **R1 – Регістер даних**
- **PS – Регістер стану**
- **PC – Лічильний команд**
- **TC – Лічильний тактів**
- **CMD – Регістер команди**
- **INS – Регістер адреси**

Крім цього було реалізована оперативна пам’ять, яку процесор може цілком використовувати під час складних обчислень. Її розмір у демонстрованій імітаційній моделі складає 20 комірок по 18 біт із можливістю швидкого збільшення цієї кількості до будь-якого іншого значення(через наслідування класу, параметром _RAM_SIZE). Існує можливість здійснювати запис та читання даних будь-якої комірки серед зазначених. Вміст оперативної пам’яті виводиться в термінал у шістнадцятковому вигляді.

Команди

Так як процесор одноадресний, то відповідно для даних маємо лише один регістер R1.

Відповідна команда для вводу в нього даних (value – цілочислове значення)

SET value – записує в регістр R1 бітовий вигляд поданого числа

DUMP address – вивантажує значення з R1 в оперативно пам’ять по вказаному адресі

LOAD address – завантажує в R1 даны з вказаної комірки оперативної пам’яті

COPY_MEM address1 > address2 – Копіює значення ячейки з address1 в address2

COMP address – (завдання з варіанту) виконує Доповнення до числа у доповнюючому коді у разі виконання умови. Для виконання бере дані по вказаному адресу з оперативної пам’яті

3. Код до лабараторної

main.py

```
"""
Flakey Roman K-23
var. 1 4 10
1 - 1-адреса | 1-й операнд завжди в акумуляторі, результат команди заноситься в акумулятор
4 - 18-бітні
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.
"""

import random;
import sys;
import os;


TACT_WORK = True;




class Processor:
    _BIT_LENGTH = 18;
    _RAM_SIZE = 16;

    @property
    def BIT_LENGTH(self):
        return self.__class__._BIT_LENGTH

    @property
    def RAM_SIZE(self):
        return self.__class__._RAM_SIZE


    @staticmethod
    def to_hex(value):
        return hex(value)[2:].zfill(2)


    @staticmethod
    def from_hex(value):
        return

    def to_binary(self, value):
        val = abs(value)
```

```
        res = [0]*self.BIT_LENGTH
        for i in range(self.BIT_LENGTH-1, -1, -1):
            res[i] = val % 2;
            val //= 2;

        if value >= 0:
            return res

        else:
            return [int(not x) for x in res]

    @staticmethod
    def bin_add(*bin_nums: str) :
        return bin(sum(int(x, 2) for x in bin_nums))[2:]


    def tact(self):
        if TACT_WORK:
            input("Press Enter to next tact...")


    def _prepare(self, command):
        self._CMD = command;
        self._Ins = [];
        self._PC += 1;
        self._TC = 1;


    def _command_parser(self):
        cmd = self._CMD.strip()

        if cmd.startswith("SET "):
            value = int(cmd[4:])
            self._Ins = ["SET_IN_R1", value]

        elif cmd.startswith("DUMP "):
            value = int(cmd[5:])
            self._Ins = ["DUMP_IN_MEM", value]

        elif cmd.startswith("LOAD "):
            value = int(cmd[5:])
            self._Ins = ["LOAD_FROM_MEM", value]

        elif cmd.startswith("COPY_MEM "):
            args = cmd[9:]
            addr1, addr2 = map(int, args.split(">", 1))
            self._Ins = ["COPY_MEM", addr1, addr2]

        elif cmd.startswith("COMP "):
            value = int(cmd[5:])
            self._Ins = ["COMPLEMENT", value]
```

```
        self._TC += 1



    def _executer(self):
        cmd = self._Ins[0]

        if cmd == "SET_IN_R1":
            self.set_in_register(self._Ins[1])

        elif cmd == "DUMP_IN_MEM":
            self.dump_in_memory(self._Ins[1])

        elif cmd == "LOAD_FROM_MEM":
            self.load_from_memory(self._Ins[1])

        elif cmd == "COPY_MEM":
            self.copy_memory(self._Ins[1], self._Ins[2])

        elif cmd == "COMPLEMENT":
            self.complement_operation(self._Ins[1])

        self._TC += 1


    def main_command_procedure(self, command):
        if command.strip() == "" or command.strip().startswith("#"):
            return;

        self._prepare(command);
        self.show();
        self.tact()

        self._command_parser();
        self.show();
        self.tact()

        self._executer();
        self.show();
        self.tact()


    def show(self):
        print("\nCommand:", self._CMD)
        print("\nR1:", "".join(map(str,self._R1)))
        print("Memory:",
            ".".join(map(self.to_hex, self._RAM.values())),
            ".".join(str(k).zfill(2) for k in self._RAM.keys()), sep="\n    ")

        print("PS:", self._PS, " "*10, "INS:", " | ".join(map(str, self._Ins)))
        print("PC:", self._PC)
        print("TC:", self._TC)
```

```
    def set_in_register(self, value):
        self._R1 = self.to_binary(value);


    def dump_in_memory(self, address):
        self._RAM[address] = int(
            "".join(map(str, self._R1)),
            base=2);


    def load_from_memory(self, address):
        self._R1 = self.to_binary(self._RAM[address]);


    def copy_memory(self, address1, address2):
        self._RAM[address2] = self._RAM[address1]


    def complement_operation(self, address):
        A = int("".join(map(str, self._R1)))
        B = self._RAM[address]

        if A>=B:
            D = "".join([str(int(not x)) for x in self._R1])

            sum = self.bin_add(D, D).zfill(self.BIT_LENGTH)[-self.BIT_LENGTH:]
            self._R1 = [str(sum)[i] for i in range(self.BIT_LENGTH)]


    def __init__(self):
        self._R1 = random.choices(range(2),k=self.BIT_LENGTH)

        self._RAM = {k:0 for k in range(self.RAM_SIZE)}

        self._CMD = ""
        self._Ins = [];
        self._PS = 0;
        self._PC = 0;
        self._TC = 0;


class WorkProcessor(Processor):
    _BIT_LENGTH = 18;
    _RAM_SIZE = 20;



def main():
    global TACT_WORK
    program_file = None;

    if len(sys.argv) > 1:
        if os.path.isfile(sys.argv[1]):
            program_file = sys.argv[1]
```

```
        if "-no-tact" in sys.argv:
            print("Tact stoping is off.")
            TACT_WORK = False

    if program_file is None:
        print("File not given. Used default program-file..")
        program_file = "program.txt"

    print("Load program file..")
    with open(program_file) as f:
        program_code = f.read().splitlines()
        f.close()

    print("Loaded program code:", *program_code, sep="\n ")

    print("Create simulation processor..")
    processor = WorkProcessor()
    print("Start program code on created simulation processor..")

    for line in program_code:
        processor.main_command_procedure(line)

    print("\nProgram finished.")



main()
```

Dockerfile.docker.file

```
# syntax=docker/dockerfile:1

FROM python:3.10-slim-buster

COPY . ./app
WORKDIR /app

CMD ["python3", "main.py", "-no-tact"]
```

4. Приклади використання

program.txt

- SET -5
- SET 12

- DUMP 0
- SET 15

- DUMP 1
- COMP 0
- COPY_MEM 0>2
- LOAD 2

result.txt

python main.py -no-tact

Tact stoping is off.

File not given. Used default program-file..

Load program file..

Loaded program code:

SET -5

SET 12

DUMP 0

SET 15

DUMP 1

COMP 0

COPY_MEM 0>2

LOAD 2

Create simulation processor..

Start program code on created simulation processor..

Command: SET -5

R1: 101001100100001111

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 1

TC: 1

Command: SET -5

R1: 101001100100001111

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: SET_IN_R1 | -5

PC: 1

TC: 2

Command: SET -5

R1: 1111111111111111010

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: SET_IN_R1 | -5

PC: 1

TC: 3

Command: SET 12

R1: 1111111111111111010

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS:

PC: 2

TC: 1

Command: SET 12

R1: 1111111111111111010

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: SET_IN_R1 | 12

PC: 2

TC: 2

Command: SET 12

R1: 0000000000000001100

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: SET_IN_R1 | 12

PC: 2

TC: 3

Command: DUMP 0

R1: 0000000000000001100

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS:

PC: 3

TC: 1

Command: DUMP 0

R1: 0000000000000001100

Memory:

00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: DUMP_IN_MEM | 0

PC: 3

TC: 2

Command: DUMP 0

R1: 0000000000000001100

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: DUMP_IN_MEM | 0

PC: 3

TC: 3

Command: SET 15

R1: 0000000000000001100

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 4

TC: 1

Command: SET 15

R1: 0000000000000001100

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: SET_IN_R1 | 15

PC: 4

TC: 2

Command: SET 15

R1: 0000000000000001111

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: SET_IN_R1 | 15

PC: 4

TC: 3

Command: DUMP 1

R1: 0000000000000001111

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 5

TC: 1

Command: DUMP 1

R1: 0000000000000001111

Memory:

0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: DUMP_IN_MEM | 1

PC: 5

TC: 2

Command: DUMP 1

R1: 0000000000000001111

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: DUMP_IN_MEM | 1

PC: 5

TC: 3

Command: COMP 0

R1: 0000000000000001111

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 6

TC: 1

Command: COMP 0

R1: 000000000000001111

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: COMPLEMENT | 0

PC: 6

TC: 2

Command: COMP 0

R1: 111111111111100000

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: COMPLEMENT | 0

PC: 6

TC: 3

Command: COPY_MEM 0>2

R1: 111111111111100000

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 7

TC: 1

Command: COPY_MEM 0>2

R1: 111111111111100000

Memory:

0c.0f.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: COPY_MEM | 0 | 2

PC: 7

TC: 2

Command: COPY_MEM 0>2

R1: 111111111111100000

Memory:

0c.0f.0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: COPY_MEM | 0 | 2

PC: 7

TC: 3

Command: LOAD 2

R1: 111111111111100000

Memory:

0c.0f.0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS:

PC: 8

TC: 1

Command: LOAD 2

R1: 111111111111100000

Memory:

0c.0f.0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17.18.19

PS: 0 INS: LOAD_FROM_MEM | 2

PC: 8

TC: 2

Command: LOAD 2

R1: 000000000000001100

Memory:

0c.0f.0c.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.
00.00

00.01.02.03.04.05.06.07.08.09.10.11.12.13.14.15.16.17
.18.19

PS: 0 INS: LOAD_FROM_MEM | 2

PC: 8

TC: 3

Program finished.