Звіт по лабораторній роботі №6

З архітектури обчислювальних систем

Студента групи К-22

Ламзіна Олега

Мій варіант:

1. Адресність процесора – **2-адресна**

2. Бітність регістру – **28-бітні**

3. Команди: **№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.

Для реалізації я використав мову програмування Go. Регістри зберігаю в доповнюючомі коді в змінній типу int64. Молодші 27 бітів виділяється для зберігання інформації про число, а також старший біт int64 використовую для зберігання знаку. Тобто всього корисних бітів буде 28, що власне і потрібно. Таким чином діапазон значень над якими можемо працювати: **[-2^27, 2^27 - 1].**

Структура **Processor** та відповідний конструктор. Тут ми зберігаємо інформацію, яка необхідна з умов лабораторної роботи а також значення регістрів **R1, R2** та значення усіх змінних – **map[string]int64**.

```
type Processor struct {
    IR          string
    R1, R2      int64
    PS, PC, TC  int32
    Variables   map[string]int64
}


func NewProcessor() Processor {
    var proc Processor
    proc.Variables          = make(map[string]int64)
    proc.Variables["R1"]    = 0
    proc.Variables["R2"]    = 0
    return proc
}
```

Структура **Command** та конструктор:

```
type Command struct {
    Name, Operand1, Operand2 string
}


func NewCommand(name, operand1, operand2 string) Command{
    var command Command
    command.Name        = name
    command.Operand1    = operand1
    command.Operand2    = operand2
    return command
}
```

Цих структур нам буде достатньо.

Далі створюємо об‘єкт процессор та парсимо вхідний файл, після чого отримуємо набір операцій. Крім того створюємо map функцій які будуть викликатися для обробки операції по ключу назви команди(щоб не робити величезний switch оператор). Доступні команди: додавання, віднімання, множення, ділення і т.д.

```
13  func main(){
14      proc := processor.NewProcessor()
15      commands := command.GetCommands()
16      processing := map[string] func (*processor.Processor, *command.Command){
17          "add"   : functions.Add,
18          "sub"   : functions.Sub,
19          "mul"   : functions.Mul,
20          "div"   : functions.Div,
21          "bswp"  : functions.BitsSwap,
22          "copy"  : functions.Copy,
23      }
24
```

Далі по черзі опрацьовуємо всі вхідні операції. Watch() – ф-я що виводить на екран стан процесора. Виводиться лише після натискання клавіші юзером.

```
26      var ch []byte = make([]byte, 2)
27      for i, command := range commands {
28          os.Stdin.Read(ch)
29
30          proc.IR = strings.Join([]string{command.Name, command.Operand1, command.Operand2}, " ")
31          proc.PC = int32(i + 1)
32          proc.TC = 1
33          proc.Watch()
34
35
36          if fn, ok := processing[command.Name]; ok {
37              fn(&proc, &command)
38          } else {
39              panic("Command not found!")
40          }
41
42
43          os.Stdin.Read(ch)
44
45          proc.PS = functions.Signum(proc.Variables[command.Operand1])
46          proc.TC = 2
47          proc.Watch()
48      }
```

Лістинг ф-ї **Watch**() & **BitRepresent**() – виводить побітово **int64**.

```go
func BitRepresent(R int64) string {
    fmt.Println("R =", R)
    var last, pow int64 = 0, 134217728 // 0, 2 ** 27
    var s string

    if R < 0 {
        last = 1
    }


    x, y := (last > 0), (R & (1 << 27) > 0)
    if (x || y) && !(x && y) {
        R = R ^ (1 << 27)
    }

    for i := 27; i>=0; i-- {
        if R & pow > 0 {
            s += "1"
        } else {
            s += "0"
        }

        if i % 4 == 0 {
            s += " "
        }

        pow /= 2
    }

    return s
}


func (s Processor) Watch() {
    fmt.Println("IR: ", s.IR)
    fmt.Println("R1: ", BitRepresent(s.Variables["R1"]))
    fmt.Println("R2: ", BitRepresent(s.Variables["R2"]))
    fmt.Println("PS: ", s.PS)
    fmt.Println("PC: ", s.PC)
    fmt.Println("TC: ", s.TC)
    fmt.Println()
}
```

Парсинг файлу:

```go
func GetCommands() []Command{
    var commands []Command
    input, err := ioutil.ReadFile("input")

    if err != nil{
        panic("#Error: problems with input file.")
    }

    for _, line := range strings.Split(string(input), "\n"){
        line = strings.Replace(line, "\r", "", -1)
        if arr := strings.Split(line, " "); len(arr) == 3 {
            commands = append(commands, NewCommand(arr[0], arr[1], arr[2]))
        } else {
            panic("#Error: line not contain 3 operands")
        }
    }

    return commands
}
```

Парсинг операції – отримання назви змінної куди записувати та лівого, правого значення операнда:

```go
func GetOperands(proc *processor.Processor, command *command.Command) (write_to string, left, right int64) {
    write_to = command.Operand1

    if _, ok := proc.Variables[write_to]; !ok {
        proc.Variables[write_to] = 0
    }

    left = proc.Variables[write_to]

    if value, err := strconv.Atoi(command.Operand2); err != nil {
        if val, ok := proc.Variables[command.Operand2]; !ok {
            panic("Item doesn't exist.")
        } else {
            right = val
        }
    } else {
        right = int64(value)
    }

    return
}
```

Тепер ф-я обробки операції, що вказано в моєму варіанті – свопінг двох бітів:

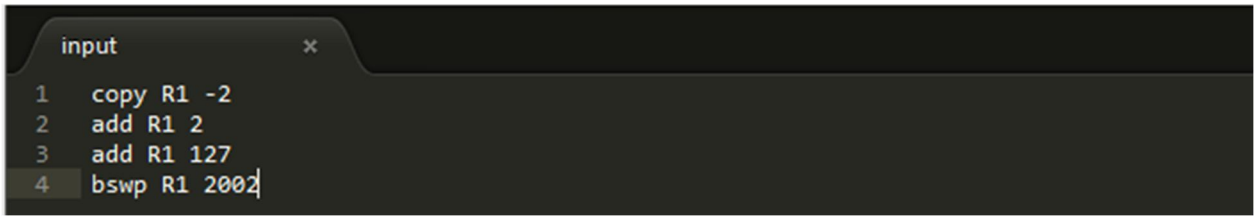
```go
func BitsSwap(proc *processor.Processor, command *command.Command) {
    if len(command.Operand2) != 4 {
        panic("Invalid operand!")
    }

    write_to, value_left, _ := GetOperands(proc, command)

    n := uint((int(command.Operand2[0]) - 48) * 10 + (int(command.Operand2[1]) - 48))
    m := uint((int(command.Operand2[2]) - 48) * 10 + (int(command.Operand2[3]) - 48))

    var nbit, mbit bool = value_left & (1 << n) > 0, value_left & (1 << m) > 0
    if n > 27 || m > 27 {
        panic("Invalid operand!")
    } else if nbit != mbit {
        value_left = value_left ^ (1 << n)
        value_left = value_left ^ (1 << m)
    }

    proc.Variables[write_to] = value_left
}
```

Інші ф-ї:

```go
func Add(proc *processor.Processor, command *command.Command) {
    write_to, value_left, value_right := GetOperands(proc, command)
    proc.Variables[write_to] = value_left + value_right
}


func Sub(proc *processor.Processor, command *command.Command) {
    write_to, value_left, value_right := GetOperands(proc, command)
    proc.Variables[write_to] = value_left - value_right
}


func Mul(proc *processor.Processor, command *command.Command) {
    write_to, value_left, value_right := GetOperands(proc, command)
    proc.Variables[write_to] = value_left * value_right
}


func Div(proc *processor.Processor, command *command.Command) {
    write_to, value_left, value_right := GetOperands(proc, command)
    if value_right == 0 {
        panic("Division by zero!")
    }
    proc.Variables[write_to] = value_left / value_right
}
```

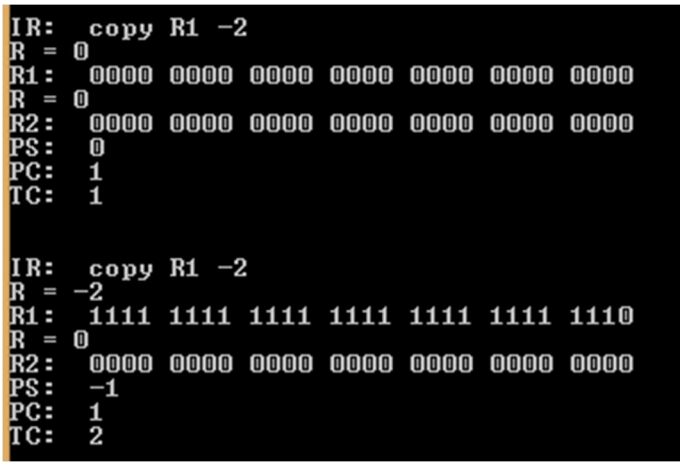
Скріншоти роботи програми:

Вхідний файл:

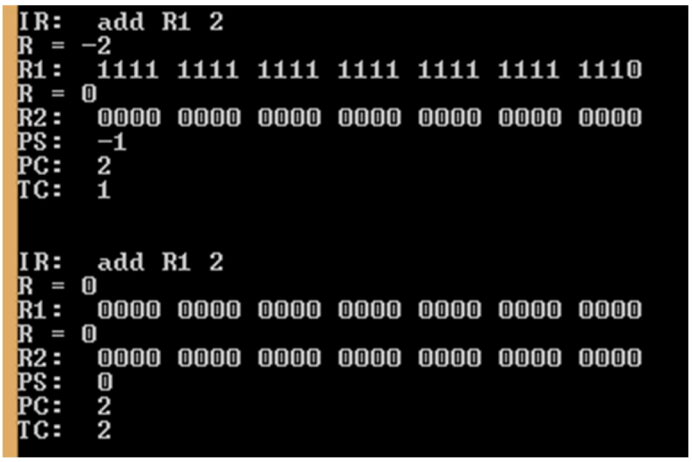


Запуск: **go run main.go**

1 операція, 1 та 2 такти // R – значення регістру, а потім його побітове представлення.



2 операція, 1 та 2 такти



3 операція, 1 та 2 такти

```
IR:  add R1 127
R = 0
R1:  0000 0000 0000 0000 0000 0000 0000
R = 0
R2:  0000 0000 0000 0000 0000 0000 0000
PS:  0
PC:  3
TC:  1


IR:  add R1 127
R = 127
R1:  0000 0000 0000 0000 0000 0111 1111
R = 0
R2:  0000 0000 0000 0000 0000 0000 0000
PS:  1
PC:  3
TC:  2
```

4 операція, 1 та 2 такти

```
IR:  bswp R1 2002
R = 127
R1:  0000 0000 0000 0000 0000 0111 1111
R = 0
R2:  0000 0000 0000 0000 0000 0000 0000
PS:  1
PC:  4
TC:  1


IR:  bswp R1 2002
R = 1048699
R1:  0000 0001 0000 0000 0000 0111 1011
R = 0
R2:  0000 0000 0000 0000 0000 0000 0000
PS:  1
PC:  4
TC:  2
```

Програма іспішно виконала усі операції.