# S3 (1) - Masina cu stiva si acumulator

Tuesday, January 10, 2023 11:37 PM

## ① MAȘINA CU STIVĂ ȘI ACUMULATOR

Calcul 1+2

Mașina cu stivă și acumulator:

```
1 acc <- 1
2 push acc
3
4 acc <- 2
5 acc <- acc + varf(stiva)
6
7 pop
```

MIPS:

```
1 li $a0 1
2 sw $a0 0($sp)
3 addiu $sp $sp -4
4 li $a0 2
5 lw $t1 4($sp)
6 add $a0 $a0 $t1
7 addiu $sp $sp 4
```

① $(7+5) * (3+2)$

acc ← 7
push acc
acc ← 5
acc ← acc(5) + vf(7)
pop
push acc
(7+5)

acc ← 3
push acc
acc ← 2
acc ← acc(2) + vf(3)
pop

acc ← acc(5+3) * vf(7+5)
pop

a) Câte push-uri / câte pop-uri?
R: 3 push, 3 pop (= nr. operații)

b) Câte op. de înmulțire? (acc ← nul)
R: 4 (= nr. operanzi)

② $1 + (2 * 3) + 4$ => codul pt. mașina cu stivă și acc.

acc ← 1 ; push acc
acc ← 2 ; push acc
acc ← 3
acc ← acc(3) * vf(2)
pop 2
push acc

acc ← 4
acc ← acc(4) + vf(2*3)
pop '(2*3)'
acc ← acc(4+(2*3)) + vf → 1
pop 1

(3) $E_1 = ((3-2) * 6) + 5$

$E_2 = 5 + (6 * (3-2))$

a) nr. push/pop ?

b) nr. instr.

c) cine folosește mai multă stivă

a) $\#pop(E_1) = \#pop(E_2) = \#push(E_1) = \#push(E_2) = 3$

b) nr instrucțiuni?

(1.1) de câte operații avem nevoie?

$m_1$ = nr. push = 3

$m_1$ = nr. pop = 3

$m_3$ = nr. înălțări = 4

$m_4$ = nr. op. aritmetice = 3

(1.2) de câte instrucțiuni este nevoie pt fiecare operație?

① push - { sw $a0 0($sp)
addiu $sp $sp -4 } ⇒ push → 2

② pop - addiu $sp $sp 4 ⇒ pop → 1

③ înălțări - acc ← ... ⇒ inc → 1

④ op aritm - { lw $t1, 4($sp) ⇒ vf(stivă) ⇒ op → 2
acc ← acc + ... }

⇒ nr instr = $3*2 + 3*1 + 4*1 + 3*2 = 19$ instr. MIPS

push pop inc. op.aritm

c) cine folosește mai multă stivă?

$E_1 = ((3-2) * 6) + 5$

$E_2 = 5 + (6 * (3-2))$

Generalizare → n literali întregi

$(((1+2)+3)+\dots)+n$ ①  vs  $1+(2+(3+(\dots+n))$ ②

① push 1
pop 1
push '(1+2)'
pop '(1+2)'
push '((1+2)+3)'
pop ...

② push 1
push 2
push 3
⋮
push n-1
pop n-1
pop
⋮

⇒ ambele variante folosesc (n-1) push-uri/pop

→ a doua variantă încarcă totul pe stivă până să facă prima operație $((n-1)+n)$, iar apoi scoate totul odată cu efectuarea restului de operații

⇒ ② fol. mai multă stivă

④ $((82+19)-(12/4)*9)+5*(13+4)$.

a) #push, #pop ?
b) # instr. ?

a) #push = #pop = nr. operații = 7
b) # instr =
#încărcări = 8 = nr. operanzi
# op_aritm = 7 = nr. operații
⇒ #instr = 7*2 + 7*1 + 8*1 + 7*2 = 43

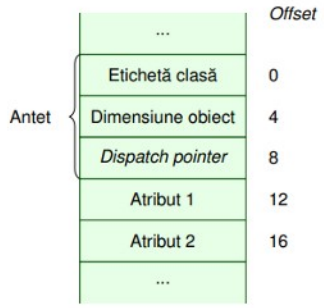
push pop inc aritm

# S3(2) - Organizarea obiectelor in memorie

Tuesday, January 10, 2023 11:42 PM

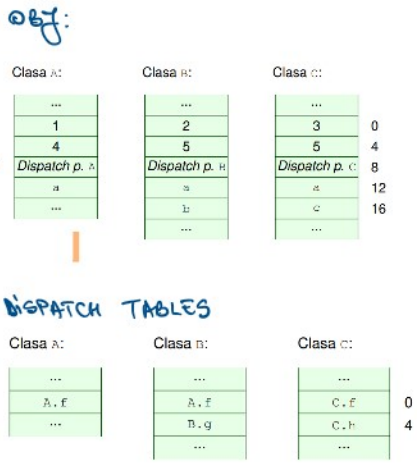
## 1 ORGANIZAREA OBIECTELOR ÎN MEMORIE

### Reprezentarea obiectelor în memorie I




```
1  class A {
2      a : Int;
3      f() : Int { 0 };
4  };
```

```
6  class B inherits A {
7      b : Int;
8      g() : Int { 0 };
9  };
10
11 class C inherits A {
12     c : Int;
13     f() : Int { 0 };
14     h() : Int { 0 };
15 };
```




## EXERCIȚII

1

```
class A {
    a : Int;

    f() : String { "" };
};

class B inherits A {
    b : String;

    g(c : Int) : Object {
        let c : Bool <- true in c
    };
};
```

A:
- tag 0
- dim 4
- A-dispTable → Object.copy, Object.abort, Object.type_name, A.f
- a

B:
- tag 1
- dim 5
- B-dispTable → A.f, B.g
- a
- b

(2)

```
class A {
    a : Int;

    f() : String { "" };

    g(c : Int) : Object {
        let c : Bool <- true in c
    };
};

class B inherits A {
    b : String;

    g(c : Int) : Object {
        let c : Bool <- false in c
    };
};
```

A: tag 0
dim 4
A_dispTable → (*) A.f, A.g
a

B: tag 1
dim 5
b_dispTable → (*) A.f, B.g
a
b

(3)

```
class Main {
   main():Object {
    (new Bar).foo()
   };
};
class Foo inherits IO{
   a:Int;
   foo():Object{self};
};
class Bar inherits Foo {
   b:Int;
   foo():Object {
     let i:Int<-0 in
      while i < 10 loop{
       (new Bar)@Foo.foo();
       i <- i+1;
      } pool
   };
};
```

Main: tag 0
dim 3
Main_dispTab → (*) (Object), Main.main

Foo: tag 1
dim 4
Foo_dispTab → (*) Object (x3), (*) IO (x4), Foo.foo()
a

Bar: tag 2
dim 5
Bar_dispTab → (*) Object (x3), (*) IO (x4), Bar.foo()
a
b

a) Câți pointeri în memorie există nevoie pt. a pointa la implementarea lui foo()?
b) — " — la 'a'? (la Int-uri care sunt atr. 'a' ai unor obiecte)

R: a) sunt numai 2 implementări pt. foo() => doar 2 zone de cod unde ar putea pointa: Foo.foo, Bar.foo

b) (new Bar)@Foo.foo() × 10

↳ 10 obiecte noi => 10 'a'-uri noi

+1 (care era deja în Bar) } => 11 pointeri la Int-uri care se referă la atr. 'a' al obiectelor

④

```
class A {
   x ← Int;
   a() : Int { .. };
};
class B inherits A {
   y ← Int;
   z ← Int;
};
class C inherits B {
   c() : Int { .. };
};
```

A: tag 0
dim 4
A_dispTab → (*)
x — A.a

B: tag 1
dim 6
B_dispTab (*)
x — A.a
y
z

C: tag 2
dim 6
C_dispTab (*)
x — A.a
y — C.c

# S3(3) - Recunoastere cod MIPS (simplu)

Tuesday, January 10, 2023 11:45 PM

**RECUNOAȘTERE / COMPLETARE COD MIPS**

OBS Nă am lucrat cu 2 tipuri de cod generat (pentru cod CoLang și pt. cod Cool) (orientat obiect)

- Convențiile de organizare sunt puțin dificile.

Exercițiile următoare urmăresc cod care nu este orientat obiect.

La examen, cel mai probabil va fi cod Cool => orientat obiect (vedeți exemplul rezolvat)

## Generarea de cod pentru apelurile de funcție

```
1  cgen(f(e1, ..., en)) =
2      sw $fp 0($sp)
3      addiu $sp $sp -4
4      cgen(en)
5      sw $a0 0($sp)
6      addiu $sp $sp -4
7      ...
8      cgen(e1)
9      sw $a0 0($sp)
10     addiu $sp $sp -4
11     jal f
```

Depunerea parametrilor în ordine inversă.

## Generarea de cod pentru definițiile de funcție

```
1  cgen(f(x1, ..., xn) { e }) =
2      move $fp $sp
3      sw $ra 0($sp)
4      addiu $sp $sp -4
5      cgen(e)
6      lw $ra 4($sp)
7      addiu $sp $sp offset
8      lw $fp 0($sp)
9      jr $ra
```

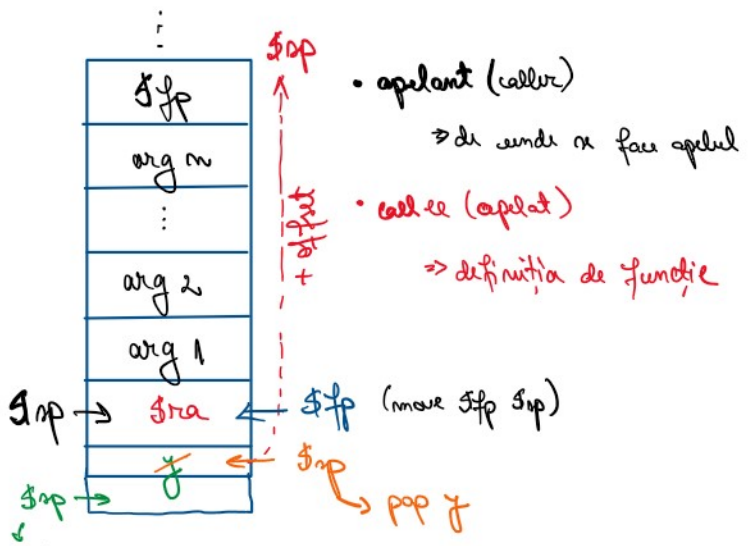
*offset* $= 4n+8$: $n$ parametri, `$ra`, `$fp`

**1** cod generat:

```
foo_entry:
    move $fp $sp                 ┐ prolog
    sw $ra 0($sp)                │
    addiu sp $sp -4              ┘
    lw $a0 8($fp)        = y
    sw $a0 0($sp)        ┐ push y
    addiu $sp $sp -4     ┘
    lw $a0 4($fp)        -> x
    lw $t1 4($sp)        ┐ pop y
    addiu $sp $sp 4      ┘
    beq $a0 $t1 true1
false1:                  x  y
    lw $a0 12($fp)       -> z
    b endif1
true1:
    li $a0 17
endif1:
    lw $ra 4($sp)        ┐ epilog
    addiu $sp $sp #1     │
    lw $fp 0($sp)        │
    jr $ra               ┘
```

secvența:

def foo(x,y,z) = if y = #2 then 17 else #3




- apelant (caller)
  - => de unde se face apelul
- callee (apelat)
  - => definiția de funcție

offset = 4(m+2) => 4(3+2) = 4*5 = 20

=> #1 = 20 , #2 = x , #3 = z

2)

```
f:
        move    $fp $sp
        sw      $ra 0($sp)
        addiu   $sp $sp -4
        sw      $fp 0($sp)
        addiu   $sp $sp -4
        lw      $a0 4($fp)
        sw      $a0 0($sp)
        addiu   $sp $sp -4
        lw      $a0 8($fp)
        sw      $a0 0($sp)
        addiu   $sp $sp -4
        jal     g
        lw      $ra 4($sp)
        addiu   $sp $sp 16
        lw      $fp 0($sp)
        jr      $ra
```

prolog

body

epilog

prolog apel

param 1 (f)

param 2 (g)

param 2 (f)

param 1 (g)

apel g

=> la apel, param. se imarcă invers => primul param f => ultimul param g (pt. că se imarcă primul)

=> 16 = 4(m+2) => 16 = m+2 => m = 2 param pt. f

f(x, y) = {g(y, x)}

# S3(4) - Locatii temporare

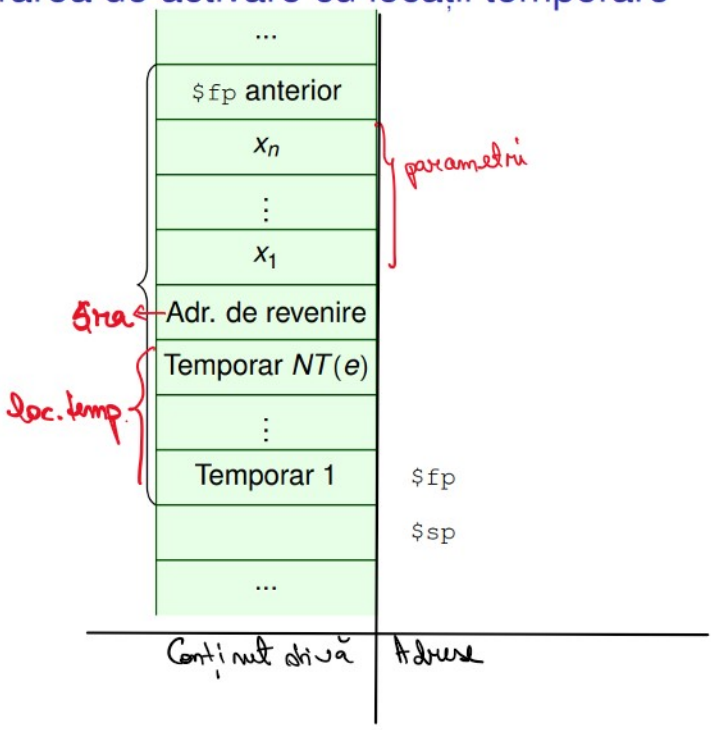
Tuesday, January 10, 2023 11:47 PM

LOCAȚII TEMPORARE + DIMENSIUNE ÎNREGISTRARE DE ACTIVARE (STACK FRAME)

## Stabilirea anticipată a locațiilor temporare II

$$
\begin{aligned}
NT(1) &= 0 \\
NT(x) &= 0 \\
NT(e_1 + e_2) &= \max(NT(e_1), 1 + NT(e_2)) \\
NT(\textit{if } e_1 = e_2 \textit{ then } e_3 \textit{ else } e_4) &= \max(NT(e_1), 1 + NT(e_2), \\
&\qquad NT(e_3), NT(e_4)) \\
NT(f(e_1, \ldots, e_n)) &= \max(NT(e_1), \ldots, NT(e_n))
\end{aligned}
$$

## Înregistrarea de activare cu locații temporare



## EXERCIȚIU:

① 

```
def f(x, y, z, w):
  if x = (y + z)
    then 5
  else if x = (if y = z then f(y, z, x, 1) else x)
    then x
  else x * (y + (z - x))
```

DIMENSIUNE:

2 reg (sfp, sra)

4 param

3 loc. temporare

⇒ (2+4+3) * 4 = 9 * 4 = 36 octeți

Dimensiune stack frame?

⇒ Calculăm NT:

- if ⇒ $NT = \max(NT(l_1), 1 + NT(l_2), NT(l_3), NT(l_4)) \Rightarrow 3$ (0, 2, 0, 3)

  $NT(l_1) = NT(x) = 0$

  $NT(l_2) = NT(y+z) = \max(NT(y), 1 + NT(z)) = \max(0, 1+0) = 1$

  $NT(l_3) = NT(5) = 0$

  $NT(l_4) = NT(\text{if} \dots)$

- if ⇒ $NT = \max(NT(l_1'), 1 + NT(l_2'), NT(l_3'), NT(l_4'))$ (0, 1, 0)

  $NT(l_1') = NT(x) = 0$

  $NT(l_2') = NT(\text{if} \dots) = \max(NT(l_1''), 1 + NT(l_2''), NT(l_3''), NT(l_4'')) = 1$ (0, 1, 0, 0)

  - if $NT(l_1'') = NT(y) = 0$

    $NT(l_2'') = NT(z) = 0$

    $NT(l_3'') = NT(f(y, z, x, 1)) = \max(NT(y), NT(z), NT(x), NT(1)) = 0$ (0, 0, 0, 0)

    $NT(l_4'') = NT(x) = 0$

  $NT(l_3') = NT(x) = 0$

  $NT(l_4') = NT(x * (y + (z - x)))$

  $= \max(NT(x), 1 + NT(y + (z - x)))$

  $= \max(0, 1 + \max(NT(y), 1 + NT(z - x)))$

  $= \max(0, 1 + \max(0, 1 + \max(NT(z), 1 + NT(x))))$

  $= \max(0, 1 + \max(0, 1 + \max(0, 1)))$

  $= \max(0, 1 + \max(0, 1 + 1))$

  $= \max(0, 1 + 2) = 3$

(2)

```
1  f(x, y, z) {                                  l3
2      if x = y + 1 then {g(x, y + 1, z)}
3          l1      l2    else {h(x + 1, y + 1)} }
                                    l4
```

$l_1 \to 0$

$l_2 \to 1$

$l_3 \to g(x^{0}, y+1^{1}, z^{0}) = 1$

$l_4 \to h(x+1^{1}, y+1^{1}) = 1$

$NT(l) = \max(NT(l_1), 1 + NT(l_2), NT(l_3), NT(l_4)) = 2$

$\downarrow 0 \quad \downarrow 2 \quad \downarrow 1 \quad \downarrow 1$

(3)

$E_1 = l_1 + l_2 + l_3 + l_4 + l_5$

$E_2 = l_1 + (l_2 + (l_3 + (l_4 + l_5)))$

$NT(E_1) \;?\; NT(E_2)$

(<) dacă sunt literali $\Rightarrow E_1 \to \max 1$

$E_2 \to \max 4$

(expr. se oval prima dată la $l_4 + l_5$ $\Rightarrow$ până acolo se fac loc. temp. pt restul lit. $\Rightarrow l_1 \dots l_4 = 4$)

(=) $E_1 \to (l_1 + l_2) \to \max(NT(l_1), 1 + NT(l_2))$ *

$l_1 + l_2 + l_3 \to \max(*, 1 + NT(l_3))$ **

$l_1 + l_2 + l_3 + l_4 \to \max(**, 1 + NT(l_4))$ ***

$l_1 + l_2 + l_3 + l_4 + l_5 \to \max(***, 1 + NT(l_5))$

$\to \max(NT(l_1), 1 + NT(l_2), 1 + NT(l_3), 1 + NT(l_4), 1 + NT(l_5))$

$E_2 \to (l_4 + l_5) \to \max(NT(l_4), 1 + NT(l_5))$

$l_3 + (l_4 + l_5) \to \max(NT(l_3), 1 + *)$ **

$l_2 + (l_3 + (l_4 + l_5)) \to \max(NT(l_2), 1 + **)$ ***

$l_1 + (l_2 + (l_3 + (l_4 + l_5))) \to \max(NT(l_1), 1 + ***)$

$\to \max(NT(l_1), 1 + NT(l_2), 2 + NT(l_3), 3 + NT(l_4), 4 + NT(l_5))$

$\Rightarrow$ atunci când $NT(l_1) = \max$

* $\Rightarrow \max(NT(l_3), 1 + \max(NT(l_4), 1 + NT(l_5))) =$ (um pas calculat pt $E_2$)

$\max(NT(l_3), \max(1 + NT(l_4), 2 + NT(l_5))) =$

$\max(NT(l_3), 1 + NT(l_4), 2 + NT(l_5)) \dots$

(>) niciodată

# S3 (6) - Cod Cool - MIPS

Tuesday, January 10, 2023 1:30 PM

## ① DEFAULT DISPATCH TABLES

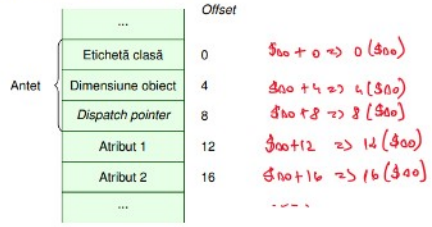
```
Object_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
IO_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   IO.out_string
    .word   IO.out_int
    .word   IO.in_string
    .word   IO.in_int
```

```
Int_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
String_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   String.length
    .word   String.concat
    .word   String.substr
```

```
Bool_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
Main_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   Main.main
    .globl  heap_start
```

## ② ORGANIZARE OBIECT => $a0 self object

Reprezentarea obiectelor în memorie I




## ③ INREGISTRAREA DE ACTIVARE

Înregistrarea de activare

| Conținut | Adresă |
|---|---|
| Parametru $n$ | |
| ⋮ | ⋮ |
| Parametru 2 | `$fp + 16` |
| Parametru 1 | `$fp + 12` |
| `$fp` | |
| `$s0` | |
| `$ra` | `$fp` |
| | `$sp` |

=> param 1 => $fp + 12 12($fp)
param 2 => $fp + 16 16($fp)
...

Înregistrarea de activare cu variabile LET

| Conținut | Adresă |
|---|---|
| Parametru $n$ | |
| ⋮ | ⋮ |
| Parametru 2 | `$fp + 16` |
| Parametru 1 | `$fp + 12` |
| `$fp` | |
| `$s0` | |
| `$ra` | `$fp` |
| Variabilă `let` 1 | `$fp - 4` |
| Variabilă `let` 2 | `$fp - 8` |
| ⋮ | ⋮ |
| Variabilă `let` $m$ | |
| | `$sp` |

let Var1 => $fp - 4 -4($fp)
let Var2 => $fp - 8 -8($fp)
...

# EXERCIȚII

①

```
1  class A {
2      content : String <- "abc";
3      f(str : String) : String {{
4          content <- str.concat(content);
5      }};
6  };
```

```
lw $a0 4($s0)
sw $a0 0($sp)
addiu $sp $sp -4
lw $a0 12($fp)
<verificare dispatch on void>
lw $t1 8($a0)
lw $t1 16($t1)
jalr $t1
```

push content

=> get dispatch object (str)

=> dispTab

=> method offset (concat)

str = primul param al lui f => $fp+1

concat = offset 16 in String_dispTab

content = primul atribut al clasei => $a0+12

```
    la      $a0 str_const7
    sw      $a0 0($sp)
    addiu   $sp $sp -4
    lw      $a0 12($fp)
<verificare dispatch on void>
    la      $t1 A_dispTab
    lw      $t1 12($t1)
    jalr    $t1
```

OBS -> la apelul de funcție

① se pun param. în ordine inversă pe stivă =>

```
cgen (param n)
sw $a0 0($sp)
addiu $sp $sp -4
```

push

② se încarcă obiectul pe care se face dispatch (ex: e.f())

- self => move $a0 $s0
- obiect static (ex: "abc") => la $a0 str_const1

```
str_const1:
    .word   2
    .word   5
    .word   String_dispTab
    .word   int_const2
    .asciiz "abc"
    .align  2
```

"abc", const(...)

- id -> atribut de clasă => <offset> ($s0)
  - param de funcție => <offset> ($fp)
  - variabilă id => -<offset> ($fp)
- ...
- rezultatul evaluării unei expresii

③ verificare dispatch on void

④ încărcare adresă dispatch_Table în $t1

① statică: e@A.f() => la $t1, A_dispTab

② dinamică: e.f() => lw $t1, 8($a0) => dispTab este mereu la offset 8 față de $s0/$a0

⑤ încărcare offset metodă (din dispTab)

lw $t1, <offset>($t1)

Nu uitați că V dispTab conține metodele lui Object + alte clase moștenite și albia apoi metodele proprii

(2)

```
class A inherits IO {};

class B inherits A{
    f() : A {{
        self@A.out_string("def");
    }};
};
```

```
    la      $a0 str_const6 => "def"
    sw      $a0 0($sp)      } push "def"
    addiu   $sp $sp -4      }
    move    $a0 $s0  -> dispatch obj = self
<verificare dispatch on void>
    la      $t1 A_dispTab  => static dispatch
    lw      $t1 12($t1)    => method offset out_string in A_dispTab
    jalr    $t1                out_string
```

```
A_dispTab:
    .word  0 Object.abort
    .word  4 Object.type_name
    .word  8 Object.copy
    .word 12 IO.out_string
    .word 16 IO.out_int
    .word 20 IO.in_string
    .word 24 IO.in_int
```

(3)

```
class A {
    v1 : Int;
    v2 : C;
    f(i : Int, c : C) : SELF_TYPE {
        {
            v1 <- i;
            v2 <- c;
            self;
        }
    };
};

class B inherits A {
    index : Int;
    set(i : Int) : SELF_TYPE {{
        index <- i;
        self;
    }};
};

class C { };

class Main inherits IO {
main() : Object {
        {
            let b : B <- new B.f(0, new C)
            in b.set(0);

            new B.set(0);
        }
    };
};
```

Se propune următoarea secv. MIPS pt. apelul B.f(0, new C)
Este corectă? Dacă nu, atunci care este secv. cod care îi corespunde?

```
    la      $a0 int_const0  } push 0
    sw      $a0 0($sp)      }
    addiu   $sp $sp -4      }
    la      $a0 B_protObj   } new B
    jal     Object.copy     }
    jal     B_init          }
<verificare dispatch on void>
    lw      $t1 8($a0)    # dispatch table
    lw      $t1 16($t1)   # method offset
    jalr    $t1           ↳ set
```

```
B_dispTab:
    .word  0 Object.abort
    .word  4 Object.type_name
    .word  8 Object.copy
    .word 12 A.f
    .word 16 B.set
```

=> new B.set(0)

# S3(7) - Exemplu

Tuesday, January 10, 2023 12:00 PM

EXEMPLU EXAMEN

```
1  class Tree inherits IO {  -- empty tree
2      isEmpty() : Bool { true };
3
4      insert(k : Int) : Tree {
5
6          new NETree.init(k, self, self);
7      };
8
9      sum() : Int { 0 };
10 };
11
12 class NETree inherits Tree {  -- non-empty tree
13     key : Int;
14     left : Tree;
15     right : Tree;
16
17     init(k : Int, l : Tree, r : Tree) : Tree {{
18         key <- k;
19         left <- l;
20         right <- r;
21         self;
22     }};
23
24     isEmpty() : Bool { false };
25
26     insert(k : Int) : Tree {
27         if k <= key then new SELF_TYPE.init(key,
28                                             left.insert(k),
29                                             right)
30                     else new SELF_TYPE.init(key,
31                                             left,
32                                             right.insert(k)) fi
33         (* P1 *)
34     };
35
36     sum() : Int { key + left.sum() + right.sum() };
37 };
38
39 class Main {
40     main() : Object {
41         let tree : Tree <-
42             new Tree.insert(2).insert(1).insert(3).insert(4)
43         (* P2 *)
44         in tree.out_int(tree.sum())
45         (* P3 *)
46     };
47 };
48
```

① Organizare directă în memorie

Tree: flag 0
dim 3
Tree_dispTab → Object. ... ×3
IO. ... ×4
Tree.isEmpty
Tree.insert
Tree.sum

NETree: flag 1
dim 6
NETree_dispTab → Object. ... ×3
IO. ... ×4
NETree.isEmpty
NETree.insert
NETree.sum
NETree.init
key
left
right

Înregistrarea de activare

| Conținut | Adresă |
|---|---|
| Parametru $n$ | |
| ⋮ | ⋮ |
| Parametru 2 | `$fp + 16` |
| Parametru 1 | `$fp + 12` |
| `$fp` | |
| `$s0` | |
| `$ra` | `$fp` |
| | `$sp` |

Înregistrarea de activare cu variabile LET

| Conținut | Adresă |
|---|---|
| Parametru $n$ | |
| ⋮ | ⋮ |
| Parametru 2 | `$fp + 16` |
| Parametru 1 | `$fp + 12` |
| `$fp` | |
| `$s0` | |
| `$ra` | `$fp` |
| Variabilă `let` 1 | `$fp - 4` |
| Variabilă `let` 2 | `$fp - 8` |
| ⋮ | ⋮ |
| Variabilă `let` $m$ | |
| | `$sp` |

(2) Dimensiunea minimă a înreg. de activare pt. metoda sum din NETree?

(1) registrii : $fp, $so, $ra => 3

(2) parametri : 0

(3) locații temporare : 1

key + left.sum() + right.sum <=> ((key + left.sum()) + right.sum())

key + left.sum() => NT = max(NT(key), 1 + NT(left.sum()))
= max(0, 1+0) = 1

key + left.sum() + right.sum() => NT = max(1, 1+NT(right.sum()))
= max(1, 1+0) = 1

=> DIMENSIUNE : 4 cuvinte

(3) Completați spațiile libere : (codul MIPS de mai jos este propus pentru key + left.sum() din metoda sum() a clasei NETree

```
l1 -> lw      $a0 12($s0)   => key
      sw      $a0 0($sp)    } push key
      addiu   $sp $sp -4
      lw      $a0 16($s0)   => $a0 -> left
      <verificare dispatch on void>
l2 -> lw      $t1 8($a0)    => dispTab
      lw      $t1 36($t1)   => method offset (sum)
      jalr    $t1
      jal     Object.copy
      lw      $t1 4($sp)
      addiu   $sp $sp 4
      lw      $t1 12($t1)
      lw      $t2 12($a0)
      add     $t1 $t1 $t2
      sw      $t1 12($a0)
```

```
aritmetic(op, e1, e2) ::= <<
<e1>
    sw      $a0 0($sp)      } push Oe1
    addiu   $sp $sp -4
<e2>
    jal     Object.copy     -> $a0 = copy Oe2
    lw      $t1 4($sp)      } pop Oe1
    addiu   $sp $sp 4
    lw      $t1 12($t1)     -> get value from Oe1
    lw      $t2 12($a0)     -> get value from Oe2
    <op>    $t1 $t1 $t2
    sw      $t1 12($a0)     # int slot
>>                          => store the value at offset 12
                               in the new obj ($a0)
```

Oe1 = de care rezultă în urma evaluării lui e1

=> poate fi { Int(...)
Bool(...), dacă op este "="

Int : 0 tag 0
4 dim 4
8 Int - dispTab
12 0

Key + left.sum()

(1) Key = primul atrib => $a0 + 12

(2) left = al doilea atrib => $a0 + 16

(3) sum() = offset 36 în dispTab

(4) Semantica operațională

new SELF_TYPE din învod (NETree) => Tree.init() => ... new SELF_TYPE

$$T_0 = \begin{cases} X & \text{if } T = \text{SELF\_TYPE and } so = X(\ldots) \\ T & \text{otherwise} \end{cases}$$ => T0 = NETree

$class(T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n)$ => class(NETree) = (key: Int, left: Tree, right: Tree)

$l_i = newloc(S_1)$, for $i = 1 \ldots n$ and each $l_i$ is distinct => l1, l2, l3 = new loc(S)

$v_1 = T_0(a_1 = l_1, \ldots, a_n = l_n)$ => v1 = NETree(key = l1, left = l2, right = l3)

$S_2 = S_1[D_{T_1}/l_1, \ldots, D_{T_n}/l_n]$ => S2 = S1[Int(0)/l1, void/l2, void/l3]

$$\frac{\cancel{v_1, S_2, [a_1 : l_1, \ldots, a_n : l_n] \vdash \{a_1 \leftarrow e_1; \ldots; a_n \leftarrow e_n;\} \mapsto v_2, S_3}}{so, S_1, E \vdash \text{new } T \mapsto v_1, S_3}$$ => nu avem expr. de inițializare pt. atribute

↳ SELF_TYPE

so = NETree(key = lkey, left = ll, right = lr)

E = {key : lkey, left : ll, right : lr, k : lk}

S = {lkey -> Int(2), ll -> void, lr -> void, lk -> Int(1)}