central exercise: the art of doing homework

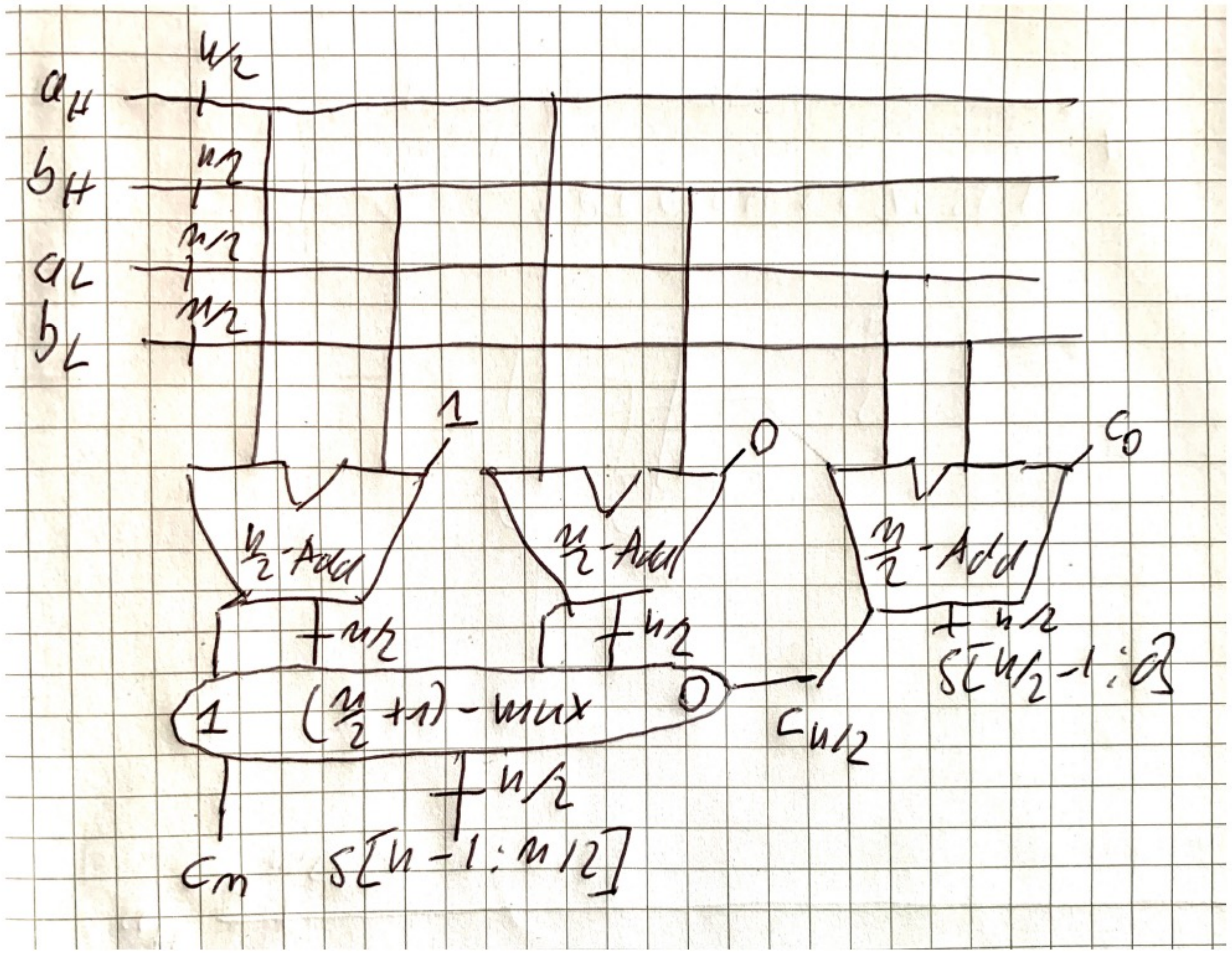
hint: i) homeworks are open book ii) (our) slide sets are manuals; even precise ones :)

exercise 2

1.1: look up in slide set 1

divide and conquer

- divide problem into smaller subproblems
- solve subproblems
- get solution of original problem from solution of subproblems
- usually applied recursively



- I2CA exercise 6
- conditional sum adder

Goal: multiply with asymptotically much less than O(n^2) gates

using subtraction

$$\begin{aligned} a_H &= a[n-1:n/2] \\ a_L &= a[n/2-1:0] \\ b_H &= b[n-1:n/2] \\ b_L &= b[n/2-1:0] \end{aligned}$$

	n-1 n/2	n/2-1 0
a	a_H	a_L
b	b_H	b_L

$$\begin{aligned} \langle a\rangle\cdot\langle b\rangle &= (\langle a_H\rangle\cdot 2^{n/2}+\langle a_L\rangle)\cdot(\langle b_H\rangle\cdot 2^{n/2}+\langle b_L\rangle) \\ &= A\cdot 2^n + B\cdot 2^{n/2}+C \\ A &= \langle a_H\rangle\cdot\langle b_H\rangle \\ C &= \langle a_L\rangle\cdot\langle b_L\rangle \\ B &= \langle a_H\rangle\cdot\langle b_L\rangle+\langle a_L\rangle\cdot\langle b_H\rangle \\ &= (\langle a_H\rangle+\langle a_L\rangle)\cdot(\langle b_H\rangle+\langle b_L\rangle)-A-C \\ \langle a_H\rangle+\langle a_L\rangle\,,\langle b_H\rangle+\langle b_L\rangle &\in B_{n/2+1} \end{aligned}$$

c(n) = cost of n-multiplier constructed here

$$c(1)=1 \quad , \quad c(n)=2\cdot c(n/2)+c(n/2+1)+O(n)$$

adders, subtractors

$$d,e\in\mathbb{B}^{n+1}$$

(n+1)-multiplier from n-multiplier

1.2: look up in slide set 1 here: 4 terms are added

$$c(n+1)=c(n)+O(n)$$

$$\boxed{c(n)=3\cdot c(n/2)+r\cdot n}$$

$$\begin{aligned} \langle d[n:0]\rangle\cdot\langle e[n:0]\rangle &= (d_n\cdot 2^n+\langle d[n-1:0]\rangle)\cdot(e_n\cdot 2^n+\langle e[n-1:0]\rangle) \\ &= d_n\cdot e_n\cdot 2^{2n}+d_n\cdot\langle e[n-1:0]\rangle\cdot 2^n+e_n\cdot\langle d[n-1:0]\rangle\cdot 2^n+ \\ &\quad \langle d[n-1:0]\rangle\cdot\langle e[n-1:0]\rangle \\ &= d_n\cdot e_n\cdot 2^{2n}+\langle d_n\wedge e[n-1:0]\rangle\cdot 2^n+\langle e_n\wedge d[n-1:0]\rangle\cdot 2^n+ \\ &\quad \langle d[n-1:0]\rangle\cdot\langle e[n-1:0]\rangle \end{aligned}$$

Goal: multiply with asymptotically much less than $O(n^2)$ gates

using subtraction

$$
\begin{aligned}
a_H &= a[n-1:n/2]\\
a_L &= a[n/2-1:0]\\
b_H &= b[n-1:n/2]\\
b_L &= b[n/2-1:0]
\end{aligned}
$$

	n-1 ... n/2	n/2-1 ... 0
a	a_H	a_L
b	b_H	b_L

1.3: look up in slide set 1 here: use module from 1.2 (please)

$$
\begin{aligned}
\langle a\rangle\cdot\langle b\rangle &= (\langle a_H\rangle\cdot 2^{n/2}+\langle a_L\rangle)\cdot(\langle b_H\rangle\cdot 2^{n/2}+\langle b_L\rangle)\\
&= A\cdot 2^n + B\cdot 2^{n/2}+C\\
A &= \langle a_H\rangle\cdot\langle b_H\rangle\\
C &= \langle a_L\rangle\cdot\langle b_L\rangle\\
B &= \langle a_H\rangle\cdot\langle b_L\rangle+\langle a_L\rangle\cdot\langle b_H\rangle\\
&= (\langle a_H\rangle+\langle a_L\rangle)\cdot(\langle b_H\rangle+\langle b_L\rangle)-A-C\\
\langle a_H\rangle+\langle a_L\rangle\ ,\langle b_H\rangle+\langle b_L\rangle &\in B_{n/2+1}
\end{aligned}
$$

c(n) = cost of n-multiplier constructed here

$$c(1)=1 \quad , \quad c(n)=2\cdot c(n/2)+c(n/2+1)+O(n)$$

adders, subtractors

$$d,e\in\mathbb{B}^{n+1}$$

(n+1)-multiplier from n-multiplier

$$c(n+1)=c(n)+O(n)$$

$$
\begin{aligned}
\langle d[n:0]\rangle\cdot\langle e[n:0]\rangle &= (d_n\cdot 2^n+\langle d[n-1:0]\rangle)\cdot(e_n\cdot 2^n+\langle e[n-1:0]\rangle)\\
&= d_n\cdot e_n\cdot 2^{2n}+d_n\cdot\langle e[n-1:0]\rangle\cdot 2^n+e_n\cdot\langle d[n-1:0]\rangle\cdot 2^n+\\
&\quad\ \langle d[n-1:0]\rangle\cdot\langle e[n-1:0]\rangle\\
&= d_n\cdot e_n\cdot 2^{2n}+\langle d_n\wedge e[n-1:0]\rangle\cdot 2^n+\langle e_n\wedge d[n-1:0]\rangle\cdot 2^n+\\
&\quad\ \langle d[n-1:0]\rangle\cdot\langle e[n-1:0]\rangle
\end{aligned}
$$

$$\boxed{c(n)=3\cdot c(n/2)+r\cdot n}$$

exercise 2.2

when translating pseudo code

- try to produce efficient code (name of lecture)
- use prefabricated libraries only if you understand their implementation
  - otherwise nasty surprises are possible
- otherwise better use C0 subset of Java
  - there you know the translation into ISA from the first half of I2OS

translating pseudo code to JAVA

when translating pseudo code

- try to produce efficient code (name of lecture)
- use prefabricated libraries only if you understand their implementation
  - otherwise nasty surprises are possible
- otherwise better use C0 subset of Java
  - there you know the translation into ISA from the first half of I2OS

correspondences (which you may have noticed):

- declarations:
  - C0 record type declarations
  - JAVA class declaration
- components
  - C0 record components
  - JAVA attributes
- functions
  - C0 functions
  - JAVA static functions
- dereferencing
  - Co p*.x
  - C0 syntactic sugar for this $p \to x$
  - JAVA p.x

exercise 2

exercise 2.1:

- copy from Helmut's slides!
- very hard to beat

new counter part of exercise 2.2 in sheet 3

- assume n is a power of 2
- use loops and arrays

new counter part of exercise 2.3 in sheet 3

- yes, with auxiliary arrays easily reducible to 2.3
- no auxiliary arrays
- maintain ist of references to remaining lists
    - initially n lists of length 1
    - round x: $n/2^x$ lists of length 2^x

exercises 3 and 4

apparently hard to get right

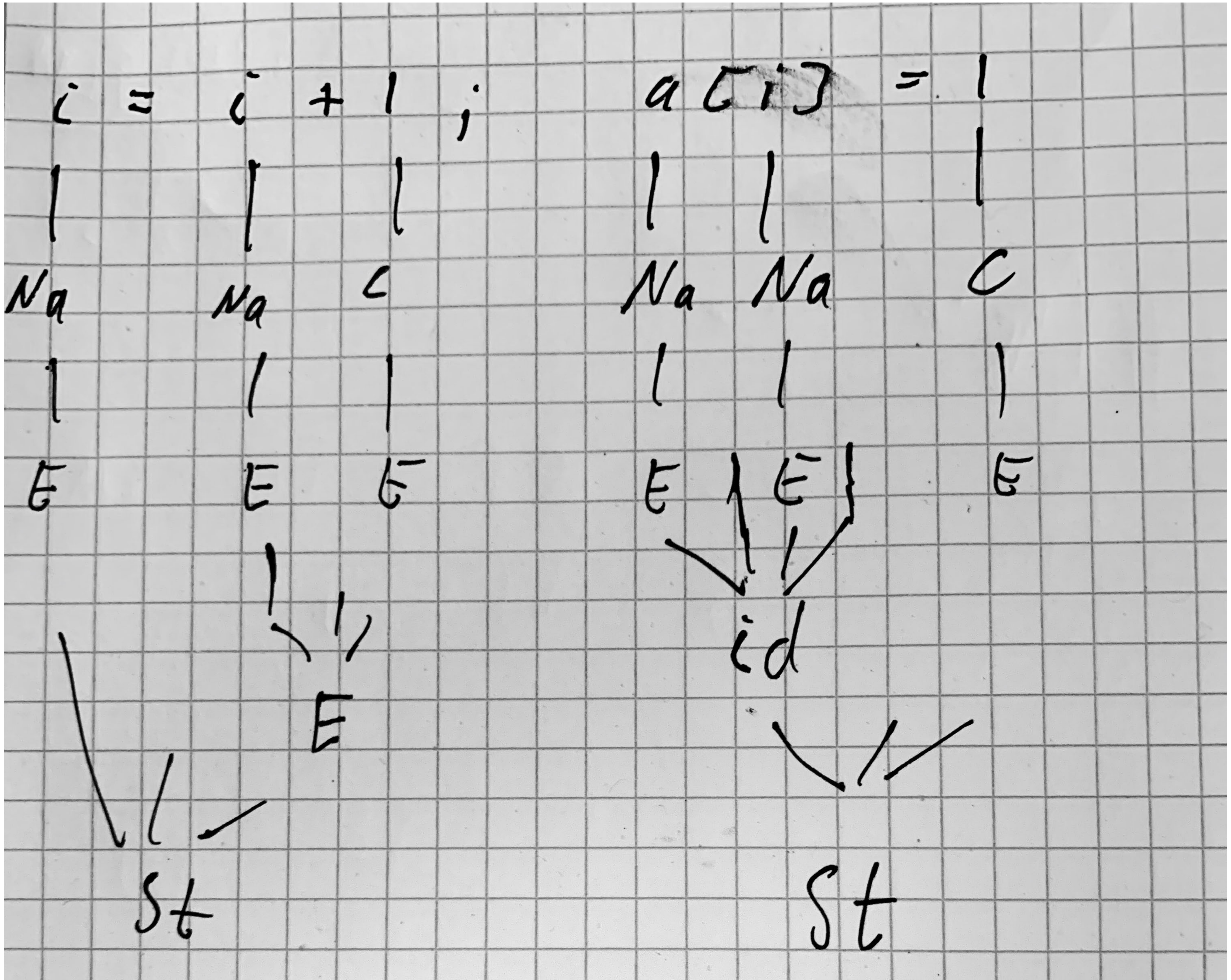
- but only if you refuse to look up expression translation and statement translation in I2OS slides
- just draw derivation trees, then look up generated code

exercises 3 and 4

apparently hard to get right

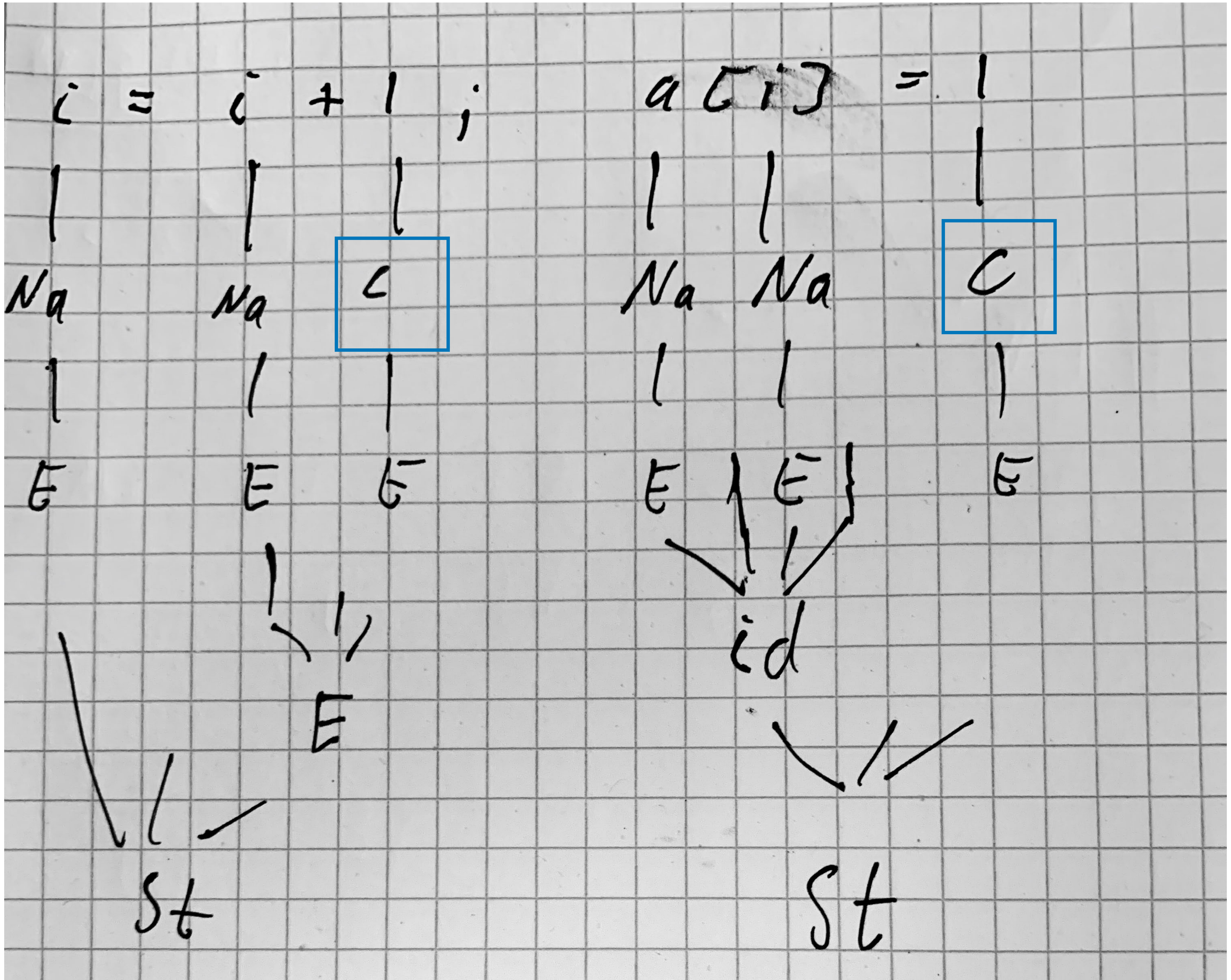
- but only if you refuse to look up expression translation and statement translation in I2OS slides
- just draw derivation trees, then look up generated code
- you would notice that the compiler is criminally non optimizing (so more lectures needed):
    - translating i = i+1 the base address for i is generated separately for the left and right side
- multiplication with 4 only for the displacement in arrays; the stack base and the arrays base addresses are a byte addresses

exercises 3.1



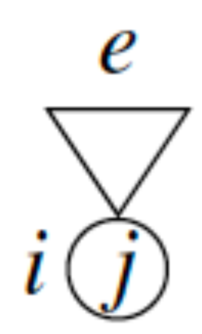
exercises 3.1

constants: I2OS 21.23



let's do the exercises

constants



interesting case:
decimal to binary conversion

$e = d[m-1:0]$ or $e = d[m-1:0]u$, $d_i \in [0:9]$

Horner rule

```
addi $23 $0 10 // gpr(23) = 10
addi $j $0 d[m-1] // gpr(j) = d[m-1]
mul(j,j,23) // gpr(j) = 10*gpr(j) , macro
addi $j $j d[m-2] // gpr(j) = gpr(j) + d[m-2]
mul(j,j,23) //gpr(j) = 10*gpr(i)
...
addi $j $j d[m-2] // gpr(j) = gpr(j) + d[1]
mul(j,j,23) //gpr(j) = 10*gpr(j)
addi $j $j d[0] // gpr(j) = gpr(j) + d[0]
```

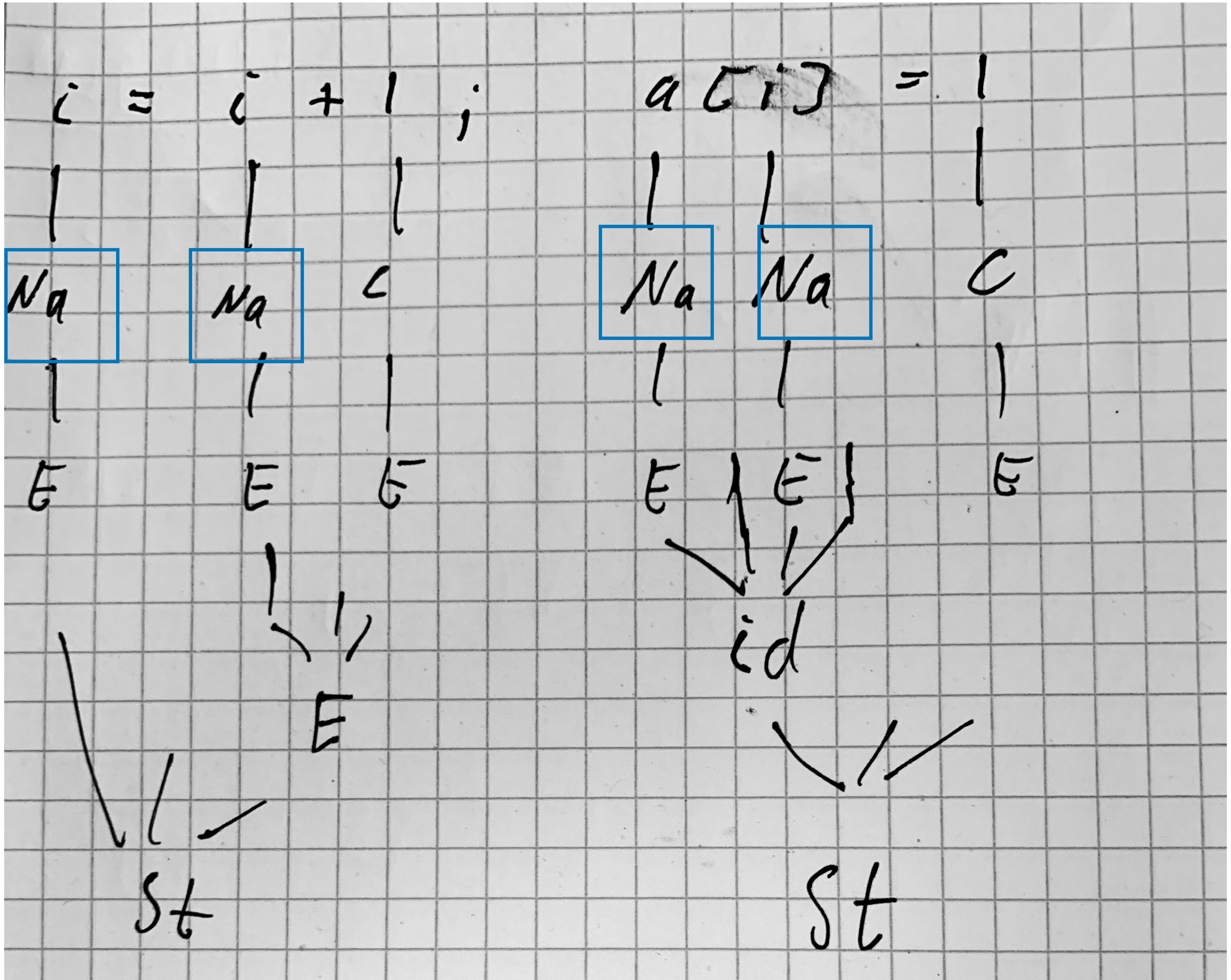
from hardware lab

mult. with 10 without macro, faster

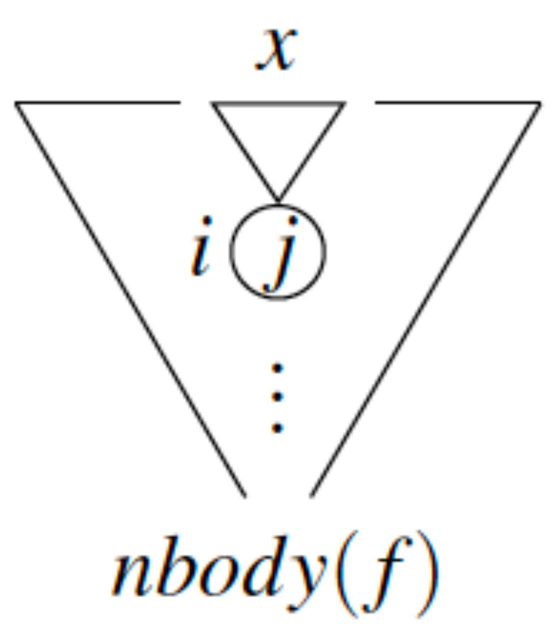
```
 add $j $j $j // gpr(j) = 2*gpr(j)old
add $23 $j $j // gpr(23) = 4*gpr(j)old
add $23 $j $j // gpr(23) = 8*gpr(j)old
add $j $j $23 // gpr(j) = 2*gpr(j)old + 8*gpr(j)old
```

exercises 3.1

global variable names: I2OS 21.28



variable name



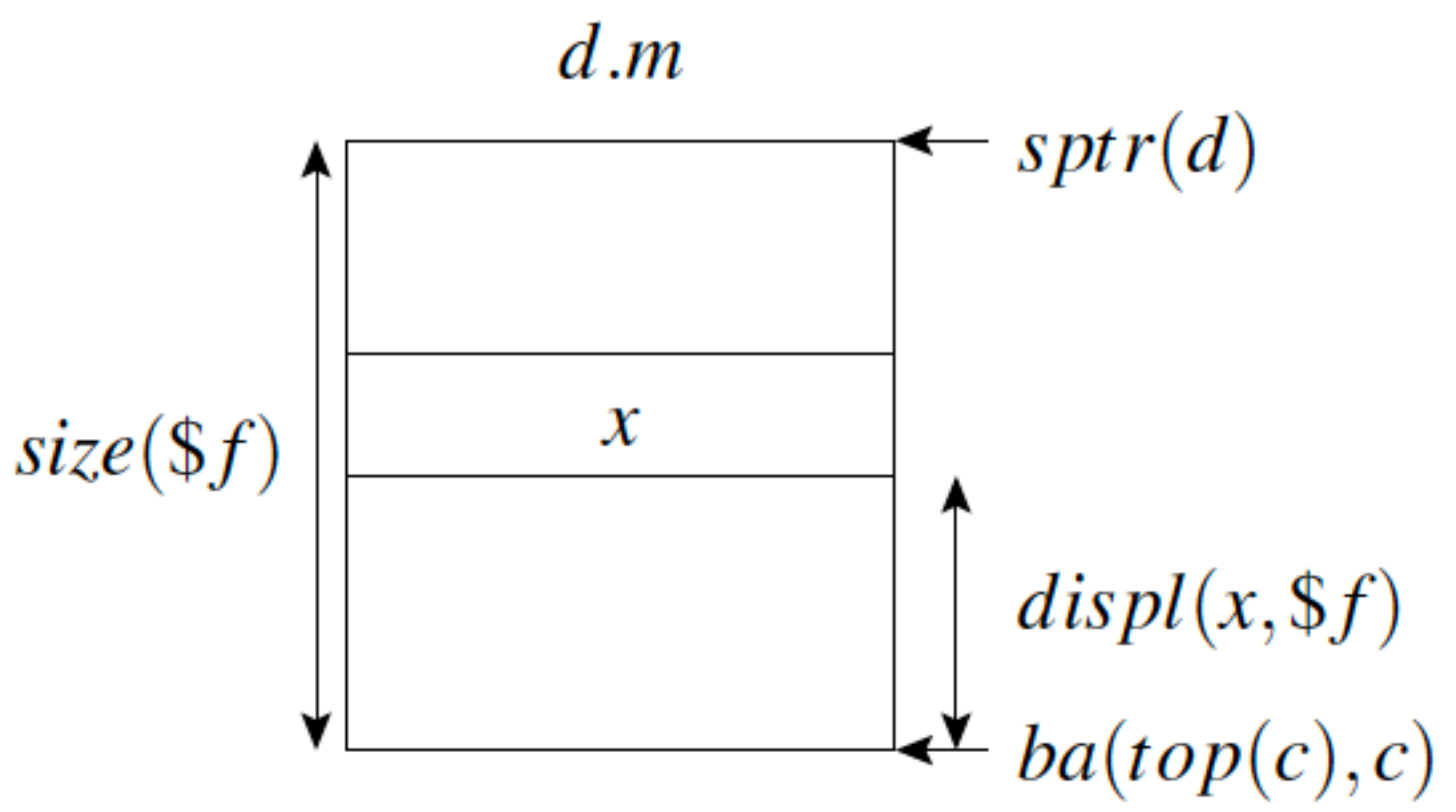
$$x \in VN \cup ft(f).VN.$$

global or local

local

$$lv(x,c) = top(c).x.$$

$$sptr(d) = d.gpr(spt_5)$$



$$ba(top(c),c) = sptr(d) -_{32} size(\$f)_{32}$$

$$ba(top(c).x,c) = ba(top(c),c) +_{32} displ(x,\$f)_{32}.$$

```
addi j spt displ(x,$f)-size($f)
```

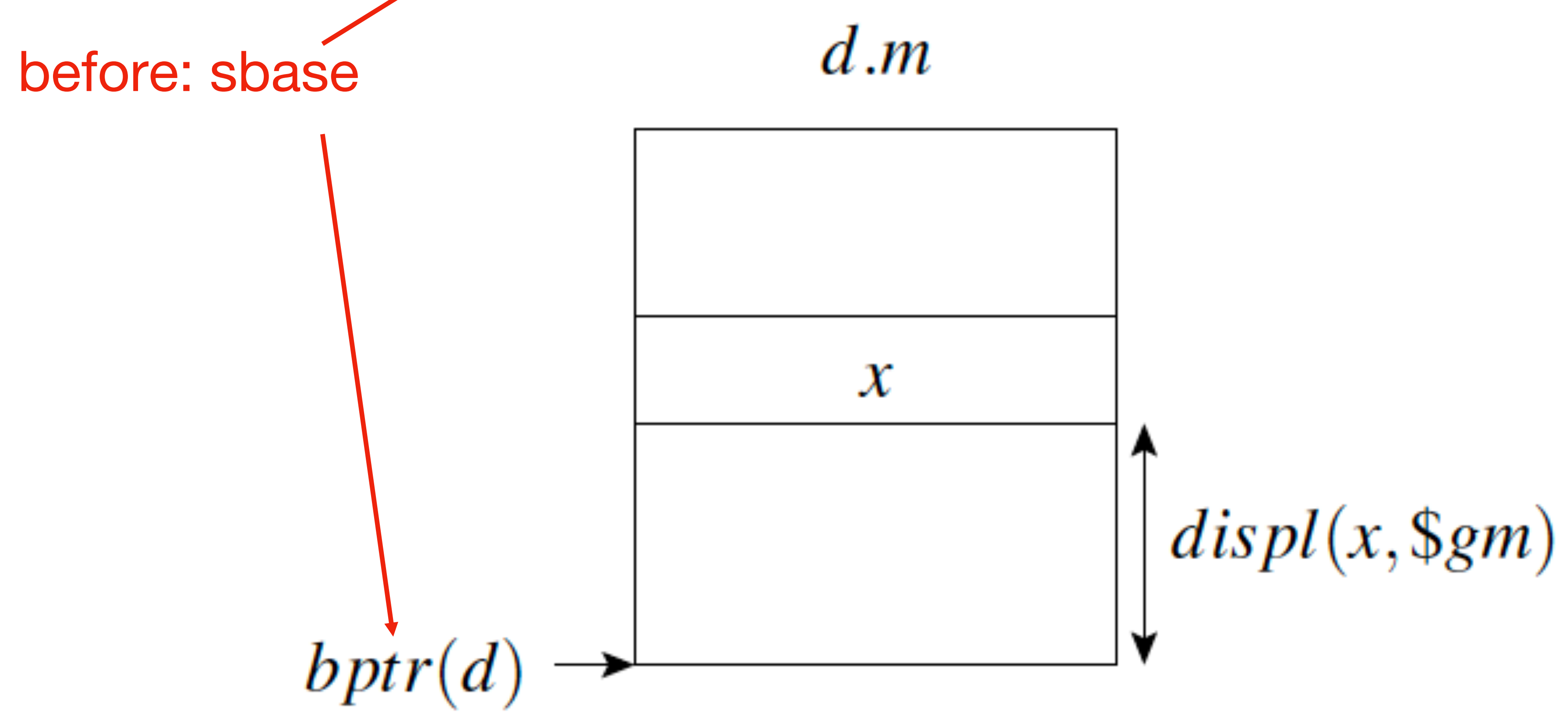
if R(i) = 1: dereference

```
deref(j)
```

global

$$lv(x,c) = gm.x.$$

$$bptr(d) = d.gpr(bpt_5)$$

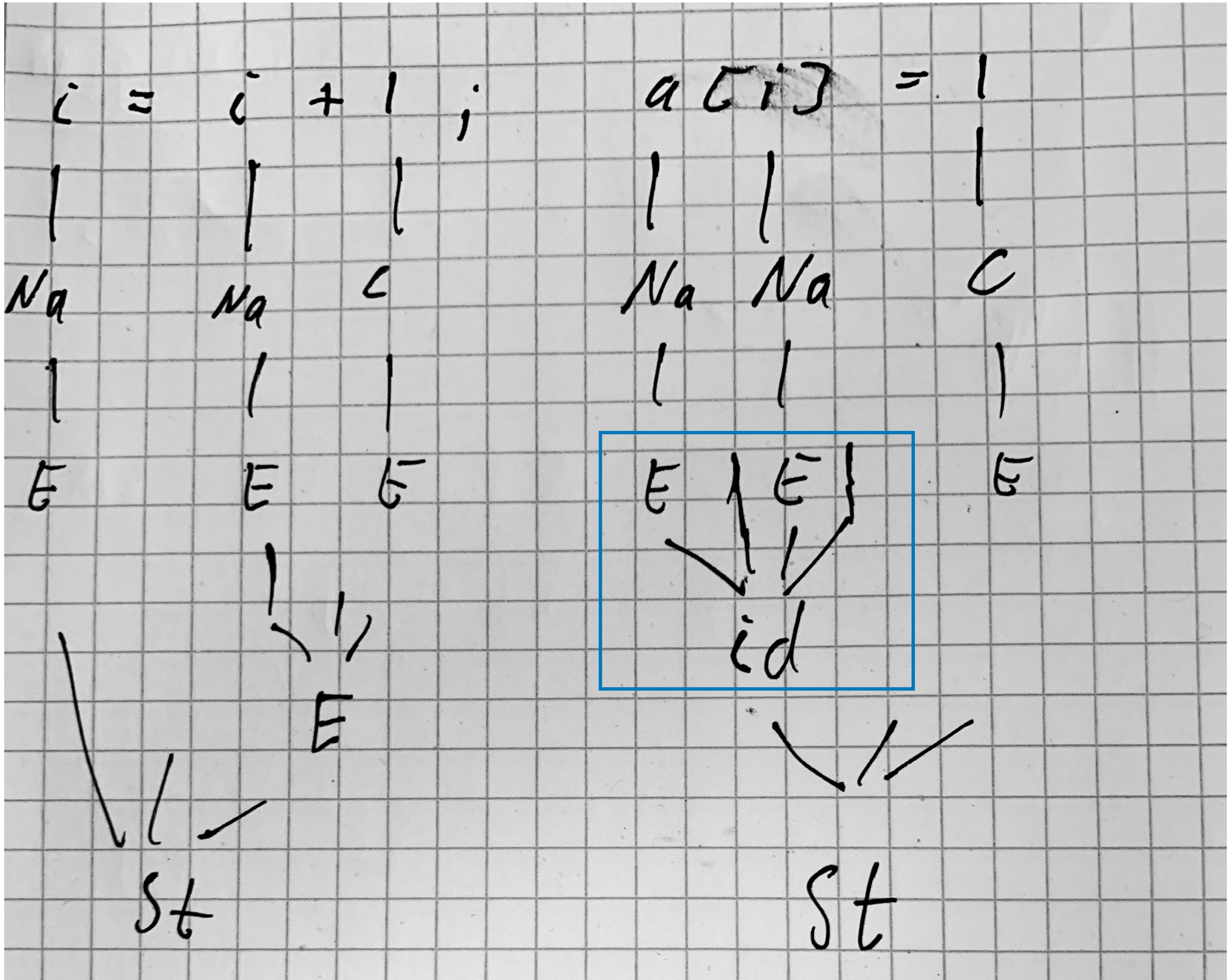


$$ba(gm,c) = bptr(d).$$

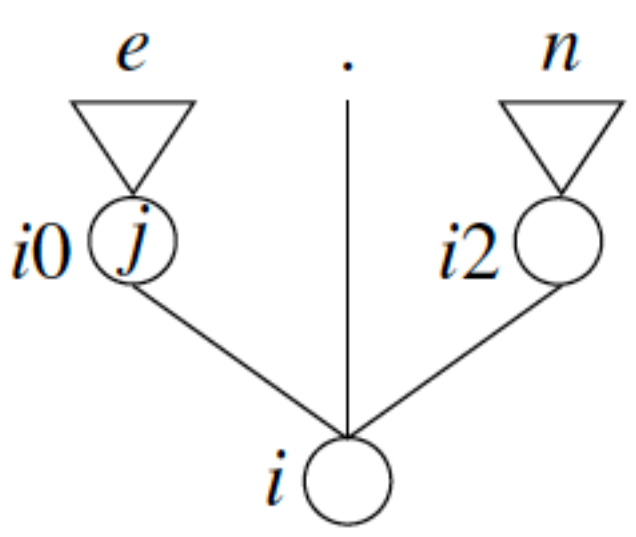
```
addi j bpt displ(x,$gm)
```

exercises 3.1

array element: I2OS 21.33



struct component



$$t = etype(e,f)$$

$$= \{t_1\ n_1; \ldots ; t_s\ n_s\}$$

$R(i0) = 0.$

$d^{k-1}.gpr(j_5) = ba(lv(e,c),c).$ IH

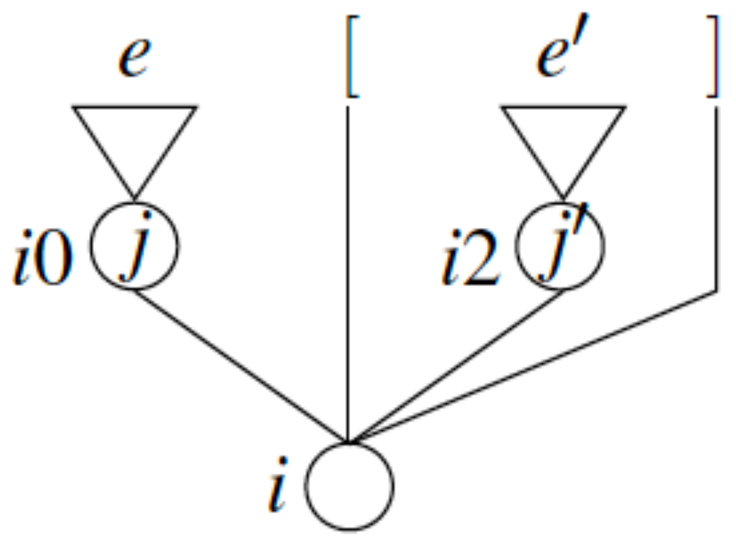
$$ba(lv(e.n,c),c) = ba(lv(e,c).n,c) = ba(lv(e,c),c) +_{32} displ(n,t)_{32}.$$

```
addi j j displ(n,t)
```

R(i)=1:

```
deref(j)
```

array element



$$etype(e,f) = t[n]$$

$$etype(e',f) \in \{int, uint\}$$

$$R(i0) = 0,$$
$$R(i2) = 1.$$

IH:

$$d^{k-1}.gpr(j_5) = ba(lv(e,c),c) \wedge d^{k-1}.gpr(j'_5) = enc(va(e',c), etype(e',f)).$$

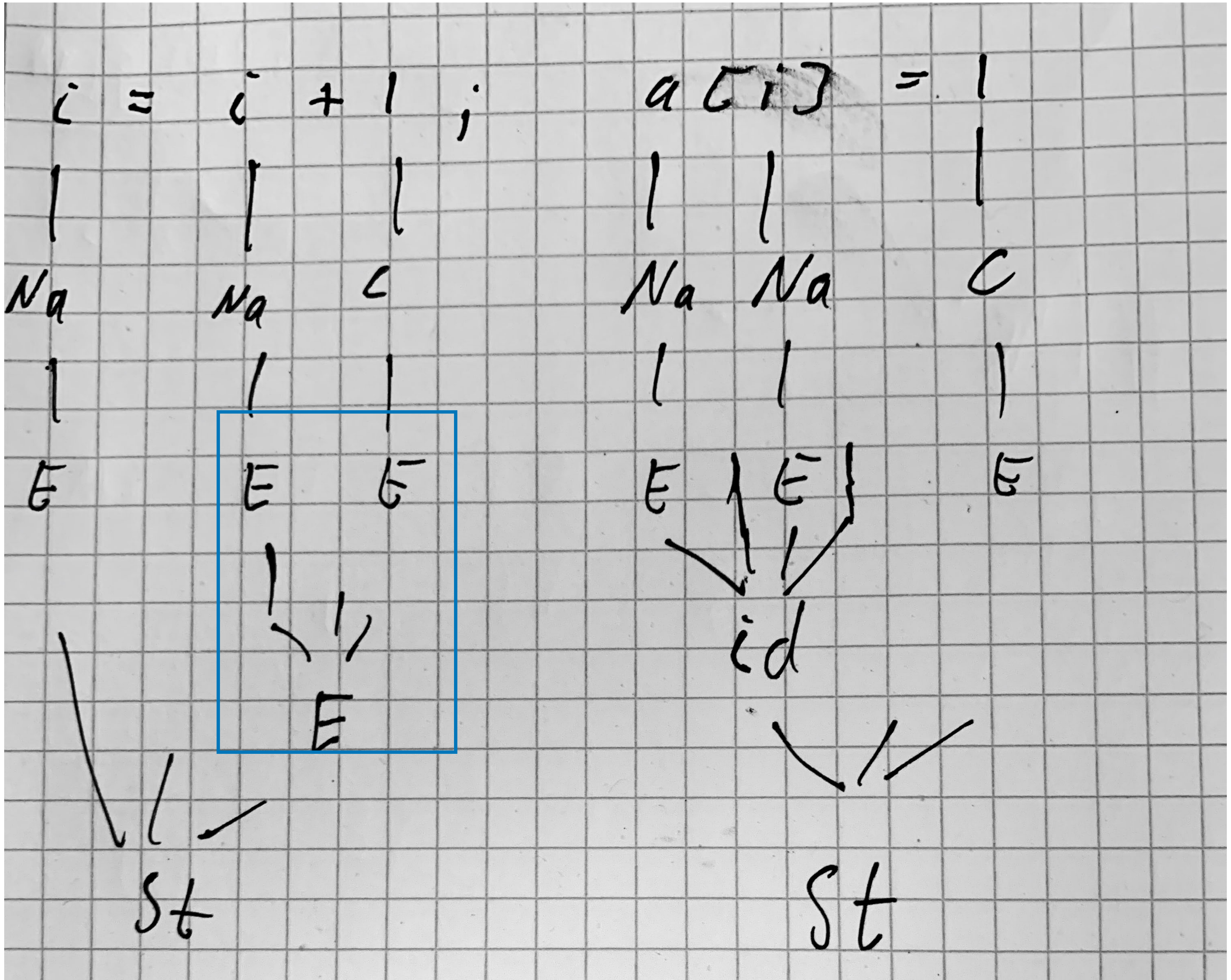
expr. eval. ba-computation

$$ba(lv(e[e'],c),c) = ba(lv(e,c)[va(e',c)],c) = ba(lv(e,c),c) +_{32} (va(e',c) \cdot size(t))_{32}$$

```
gpr(23)  = enc(size(t),uint)
mul(j',j',23)
add  j j j'
```

exercises 3.1

binary operator: I2OS 21.44



binary arithmetic operators

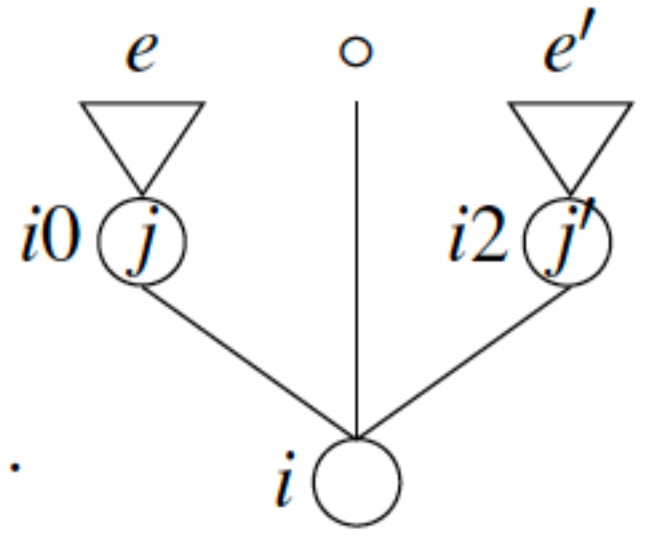
$$\frac{\text{If } \circ = + \text{ and } t = int}{\texttt{add j j j'}}$$

$$\frac{\text{If } \circ = + \text{ and } t = uint}{\texttt{addu j j j'}}$$

$$\frac{\text{If } \circ = - \text{ and } t = int}{\texttt{sub j j j'}}$$

$$\frac{\text{If } \circ = - \text{ and } t = uint}{\texttt{subu j j j'}}$$

$t \in \{int, uint\}$



$$etype(e \circ e', f) = etype(e, f) = etype(e', f) = t.$$

slide pebble j

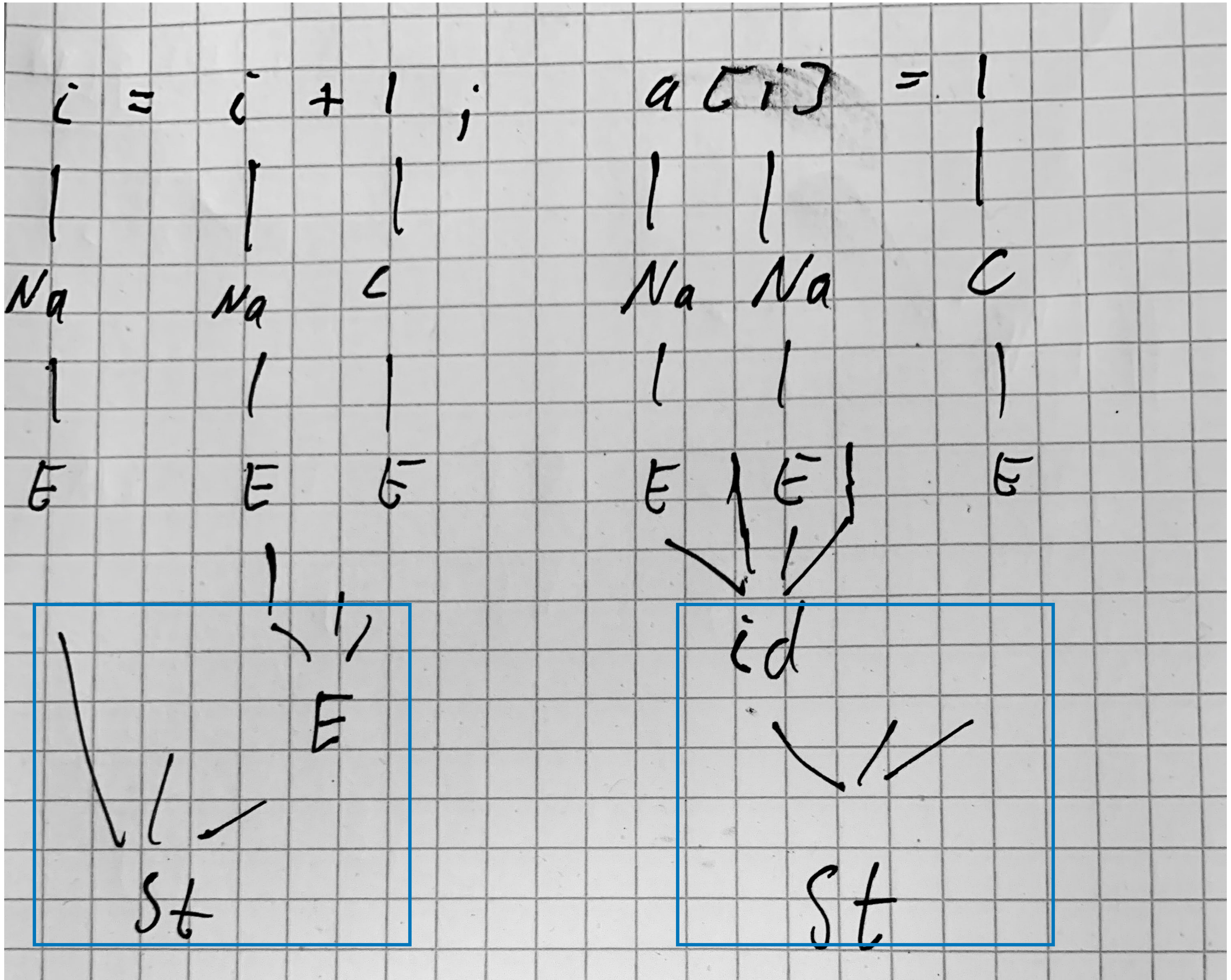
$$pebble(i0, j, k-1) \wedge pebble(i2, j', k-1) \wedge pebble(i, j, k).$$

$$R(i) = R(i0) = R(i2) = 1$$

IH: $d^{k-1}.gpr(j_5) = enc(va(e,c),t) \wedge d^{k-1}.gpr(j'_5) = enc(va(e',c),t)$

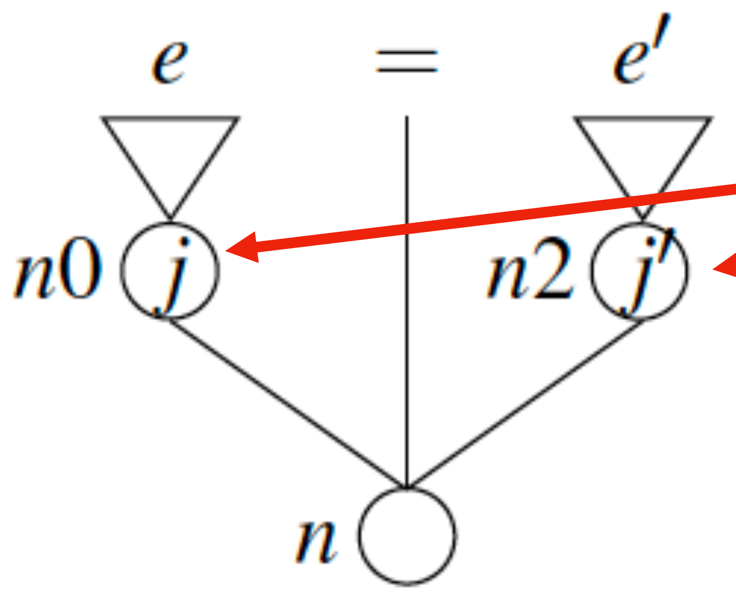
exercises 3.1

assignments: I2OS 22.15



code generation

assignments



problem:

pebble strategy for n2
might use pebble j

this overwrites register j

$$R(n0) = 0,$$
$$R(n1) = 1.$$

solution:

- J set of pebbles
- code(n,J) generated without pebbles in J
- for generation of code(n) remove J from free list

```
code(n0)
```

```
code(n2,{j})
```

$$d'.gpr(j_5) = ba(lv(e,c),c),$$

$$d'.gpr(j_5') = \begin{cases} ba(va(e',c),c), & pointer(t), \\ enc(va(e',c),t), & t \in ET. \end{cases}$$

```
sw    j' j 0
```

then $consis'(c',d'') \wedge d''.pc = end(n) +_{32} 1_{32}.$

exercises 3 and 4

apparently hard to get right

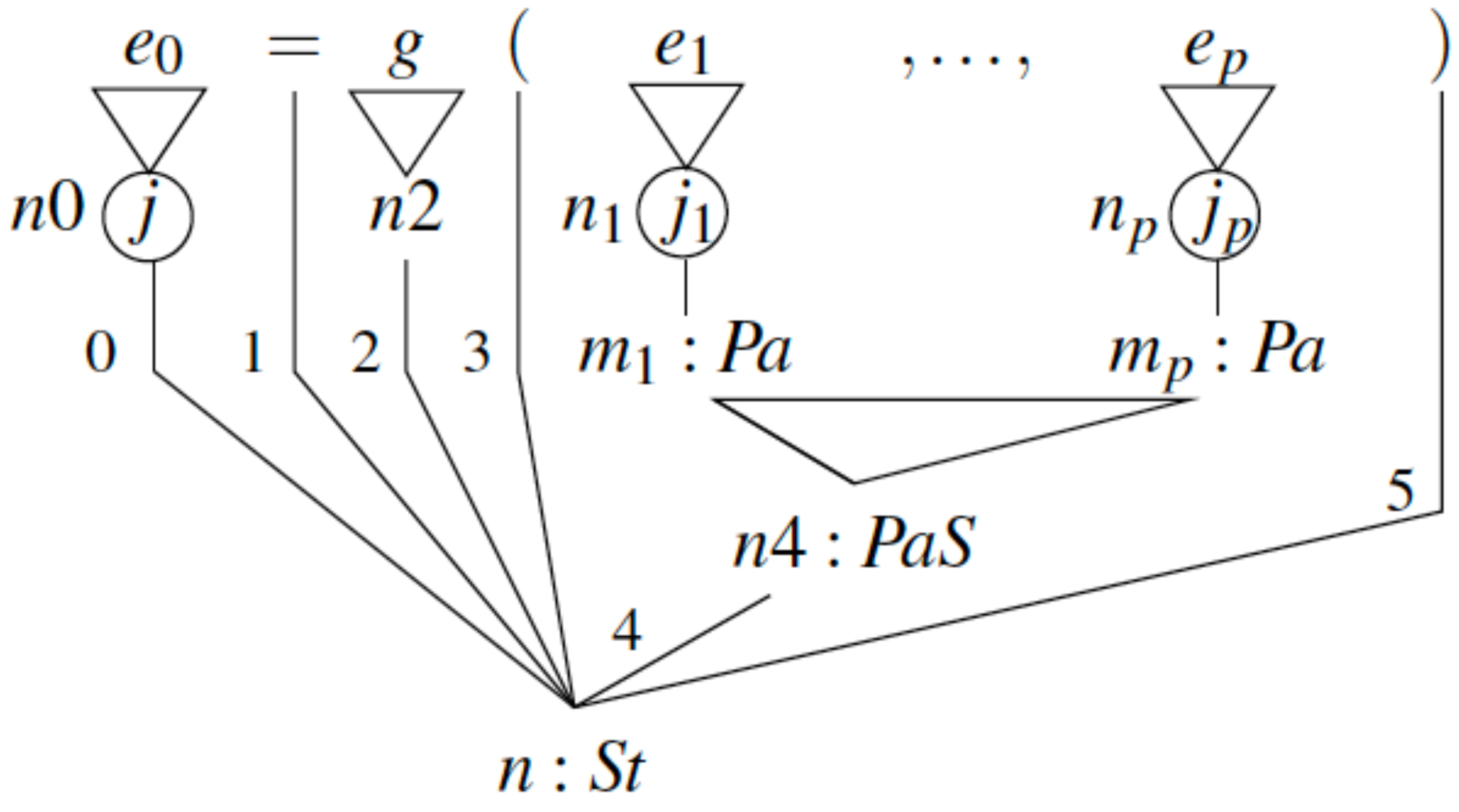
- but only if you refuse to look up expression translation and statement translation in I2OS slides
- just draw derivation trees, then look up generated code
- you would notice that the compiler is criminally non optimizing (so more lectures needed):
    - translating i = i+1 the base address for i is generated separately for the left and right side
- multiplication with 4 only for the displacement in arrays; the stack base and the arrays base addresses are a byte addresses

exercises 5

look up at slide set 22, slides 36 to 44

code generation

function call



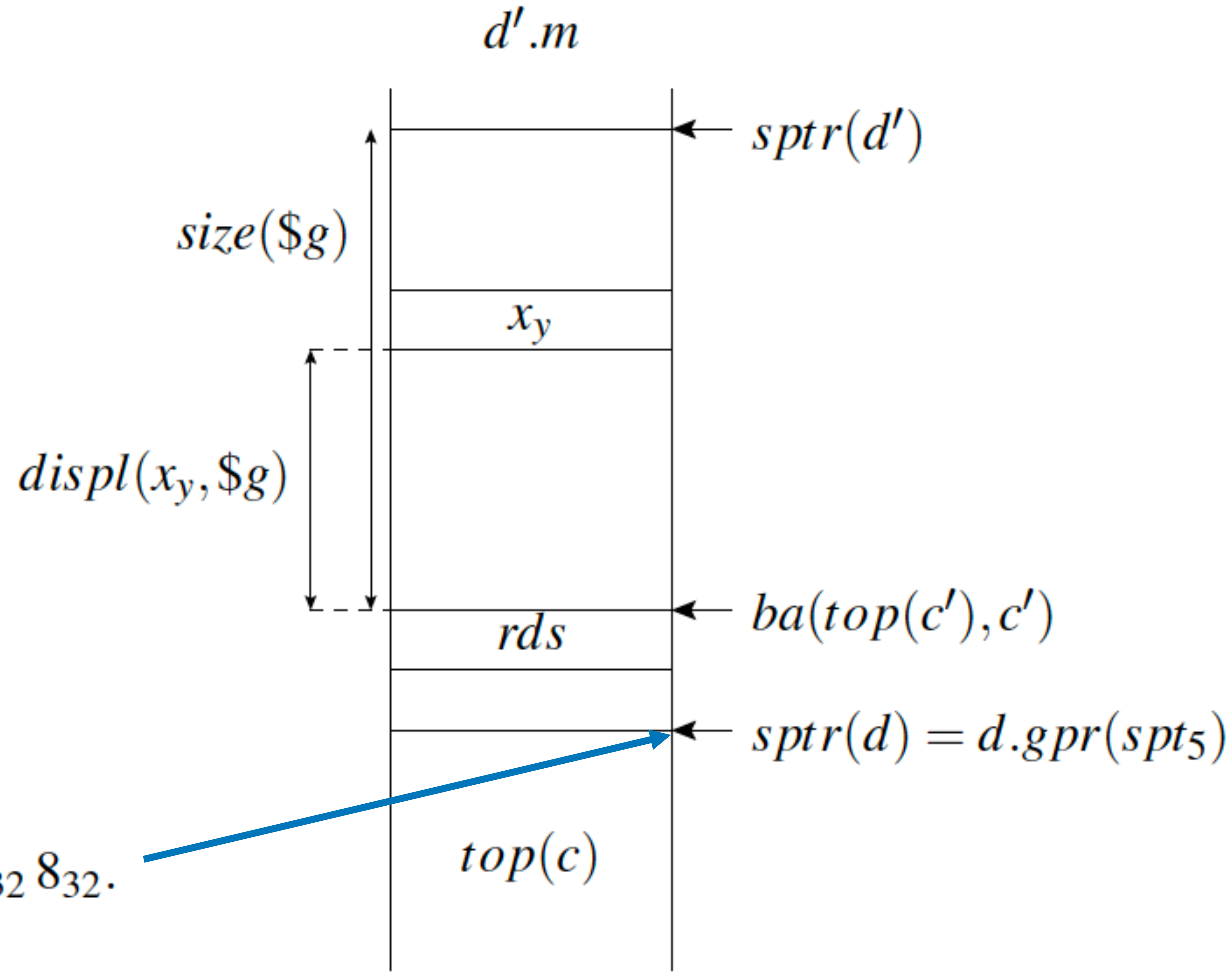
$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

$$R(n0) = 0$$

$$R(n_y) = 1.$$

$$d.gpr(spt_5) = ba(top(c'), c') -_{32} 8_{32}.$$

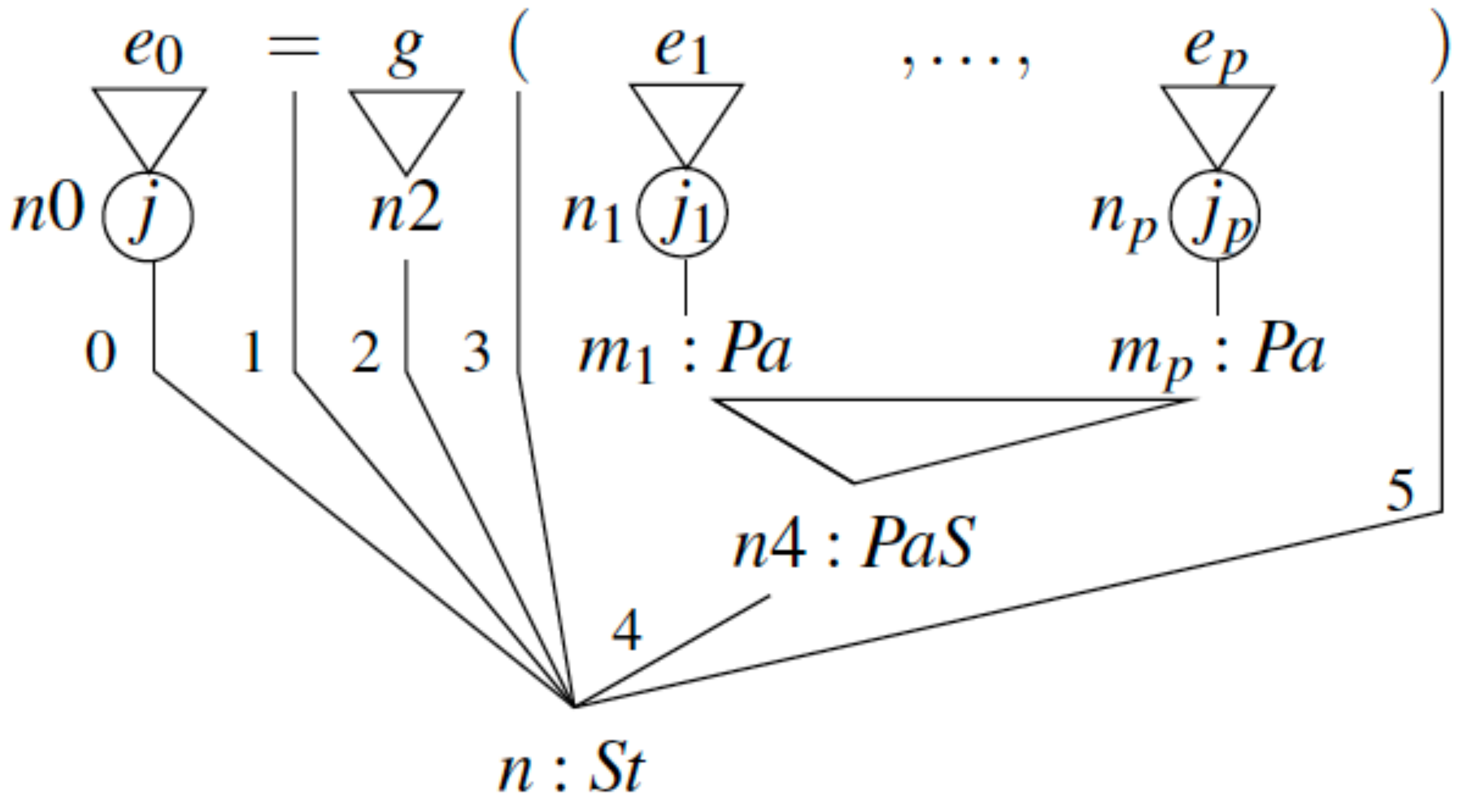


test space on stack

```
addi 1 spt size($g) +8 // gpr(1) = sptr + size($g) +8
                       //        = 'new stack pointer'
subi 1 1 smax          // gpr(1) = 'new stack pointer'
                       //        - smax
blez 1 4               // if 'new stack pointer'<=smax
                       // skip error handling
gpr(1) = x             // macro
sysc                   // system call
```

code generation

function call



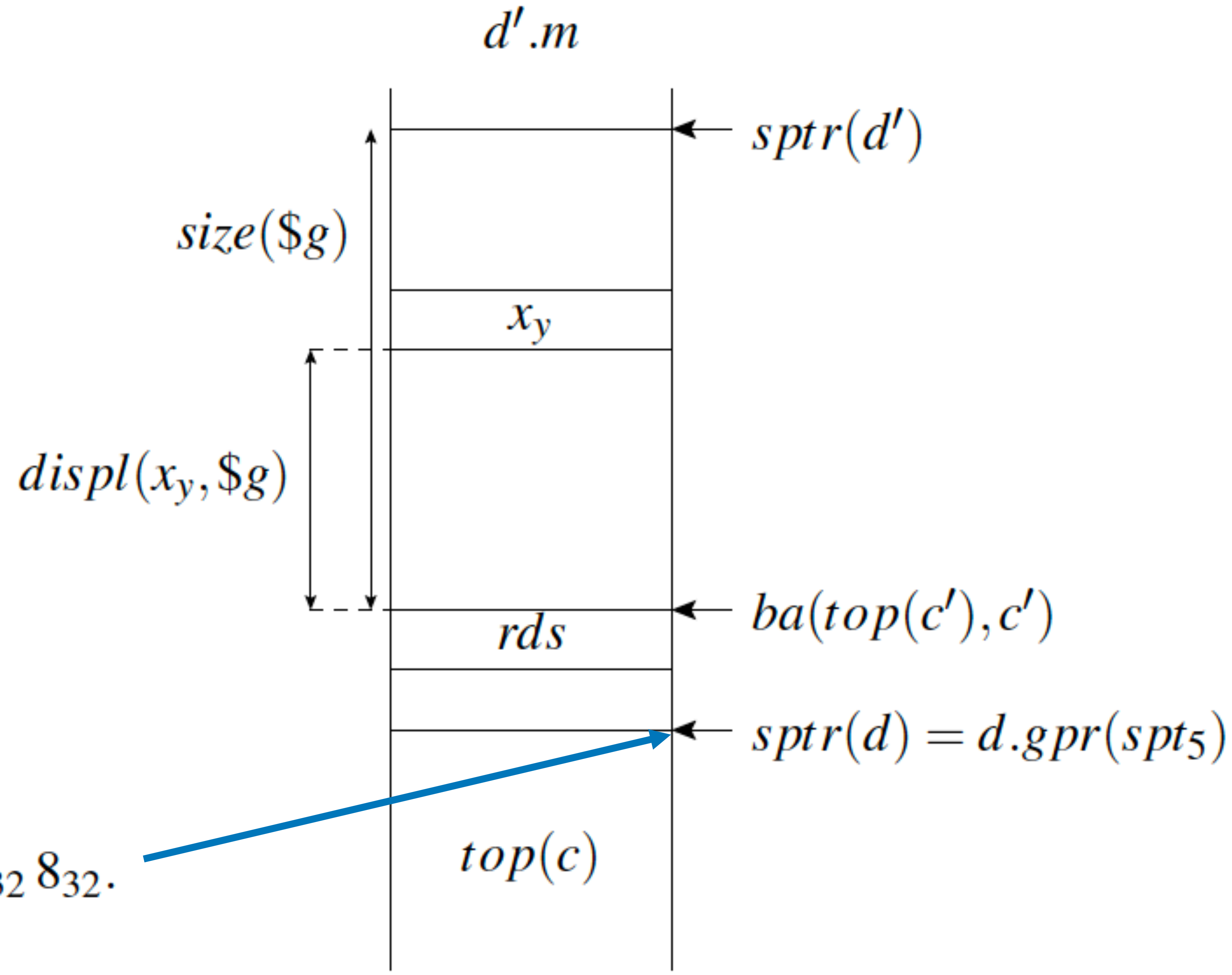
$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

$$R(n0) = 0$$

$$R(n_y) = 1.$$

$$d.gpr(spt_5) = ba(top(c'), c') -_{32} 8_{32}.$$



test space on stack

```
addi 1 spt size($g) +8 // gpr(1) = sptr + size($g) +8
                       //        = 'new stack pointer'
subi 1 1 smax          // gpr(1) = 'new stack pointer'
                       //        - smax
blez 1 4               // if 'new stack pointer'<=smax
                       // skip error handling
gpr(1) = x             // macro
sysc                   // system call
```

increase stack pointer

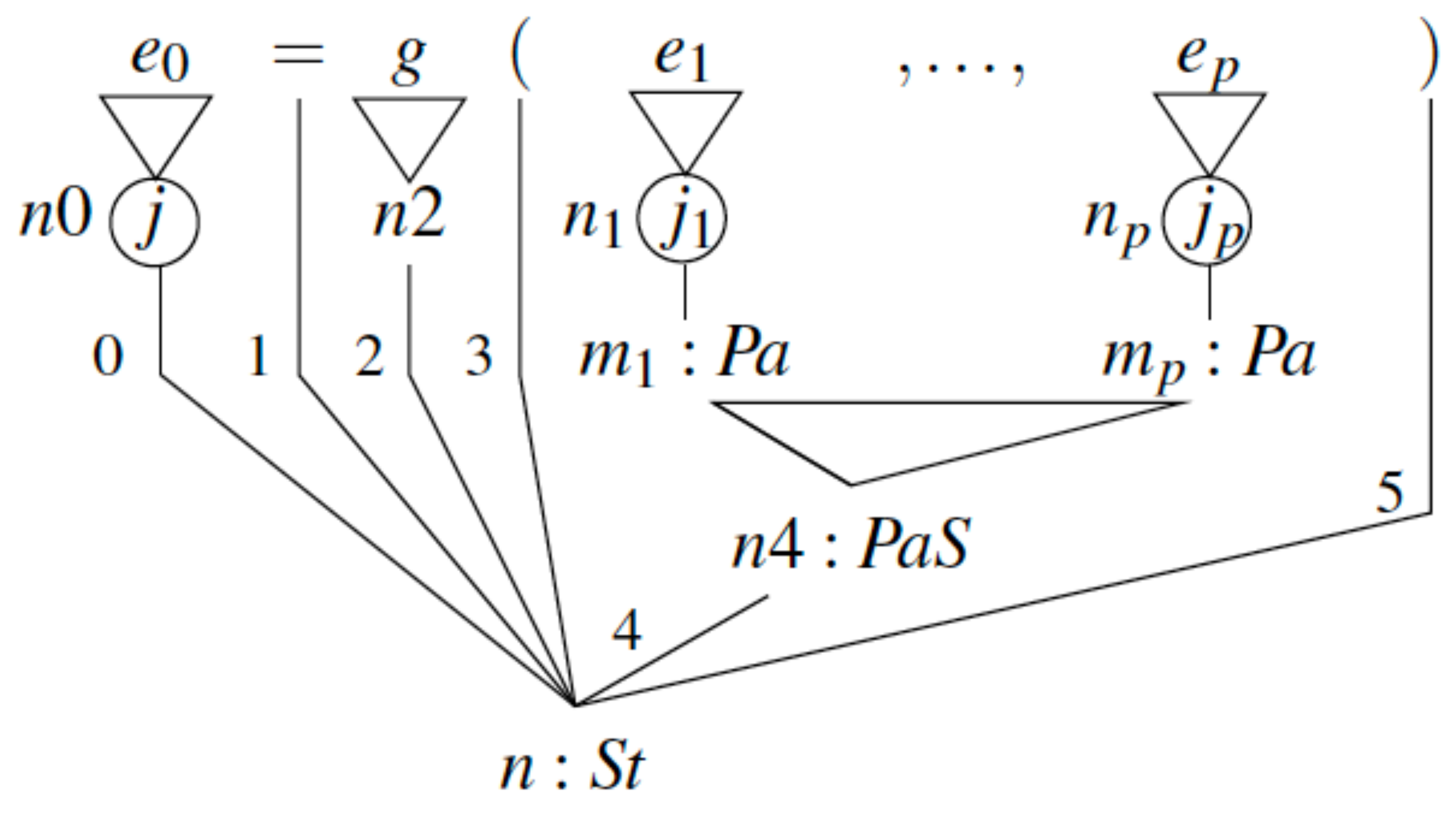
```
addi spt spt size($g)+8
```

then

$spt\text{-}consis(c', d')$

code generation

function call



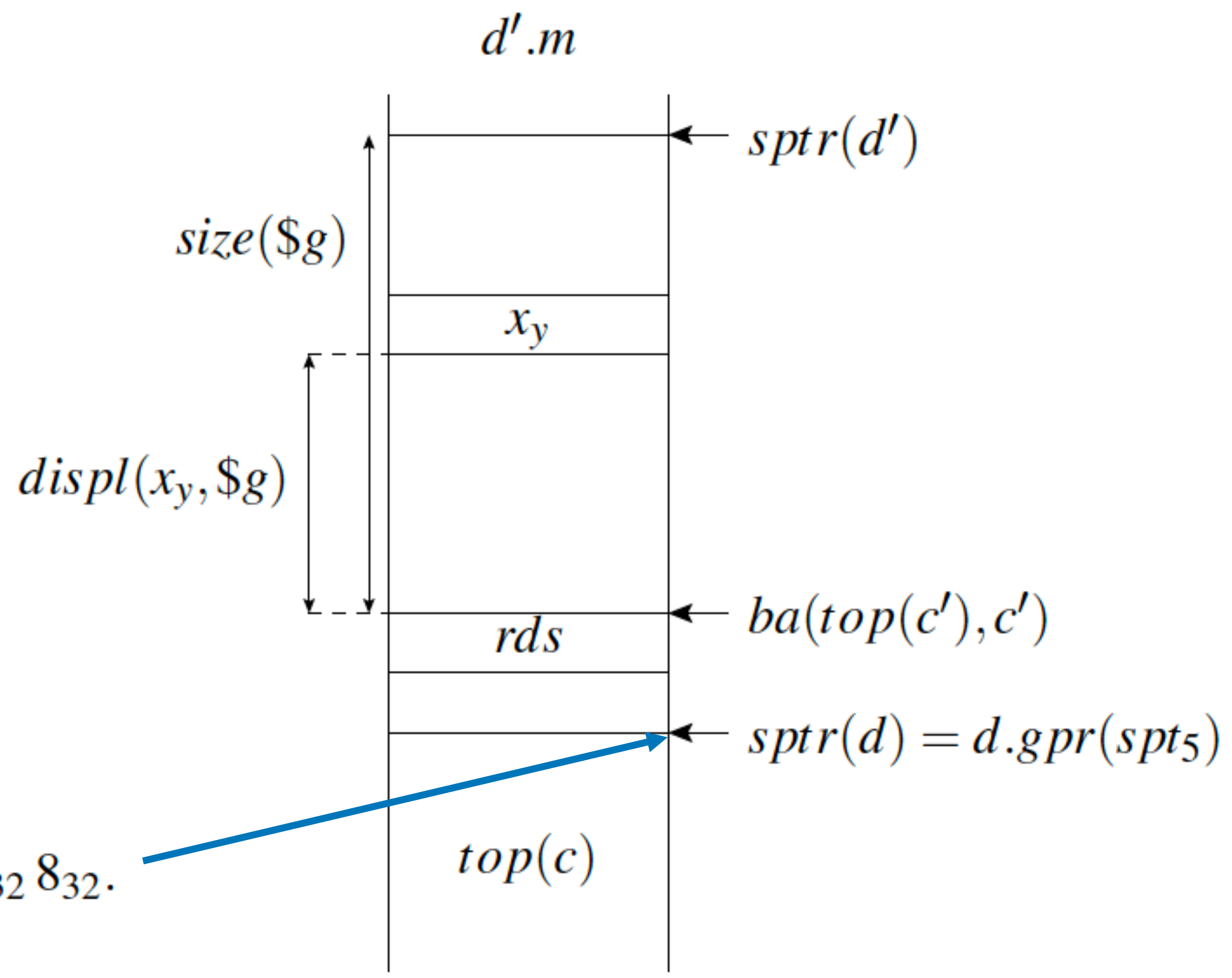
$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

$$R(n0) = 0$$

$$R(n_y) = 1.$$

$$d.gpr(spt_5) = ba(top(c'), c') -_{32} 8_{32}.$$



test space on stack

```
addi 1 spt size($g) +8 // gpr(1) = sptr + size($g) +8
                       //        = 'new stack pointer'
subi 1 1 smax          // gpr(1) = 'new stack pointer'
                       //        - smax
blez 1 4               // if 'new stack pointer'<=smax
                       // skip error handling
gpr(1) = x             // macro
sysc                   // system call
```

increase stack pointer

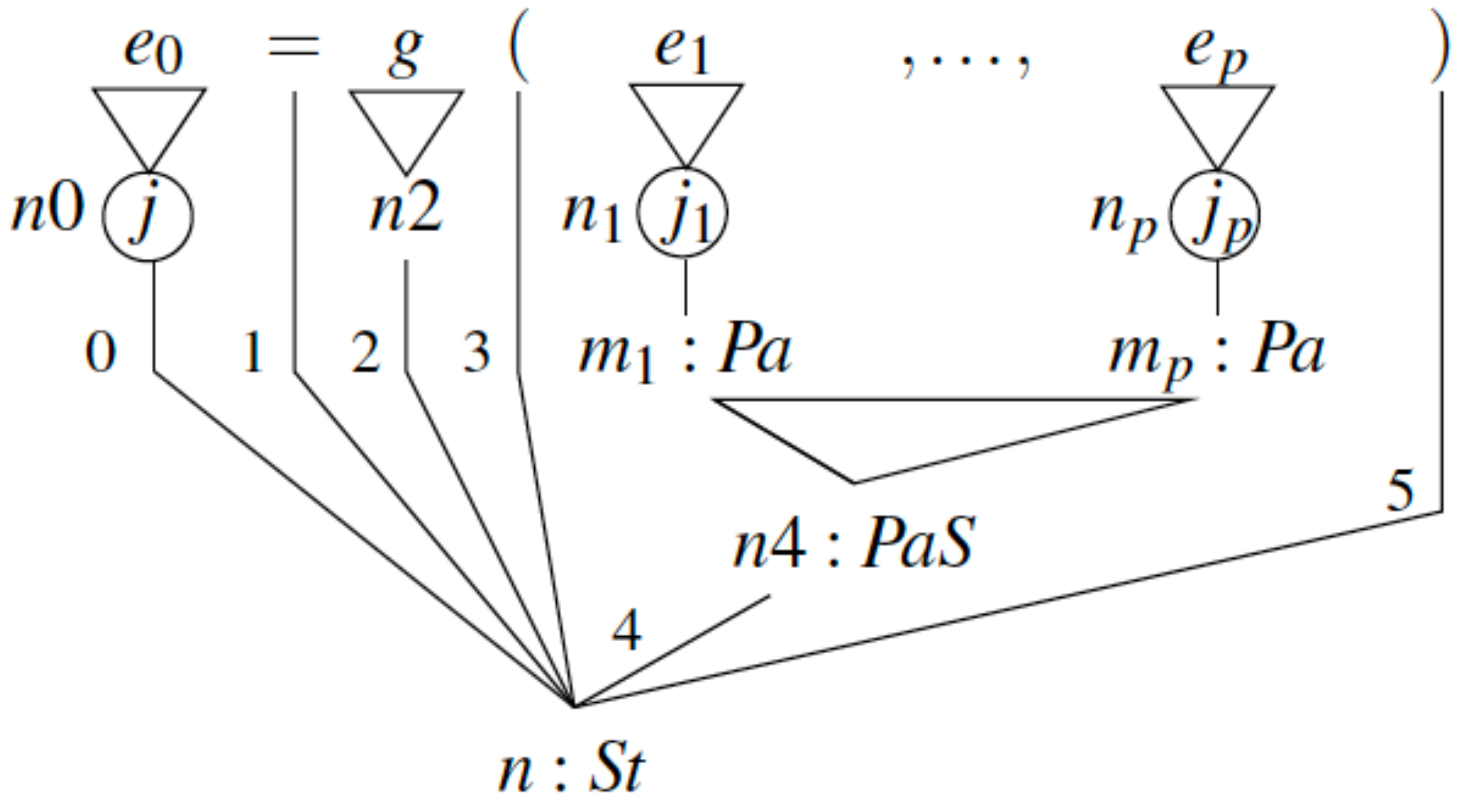
```
addi spt spt size($g)+8
```

then

$spt\text{-}consis(c', d')$

code generation

function call



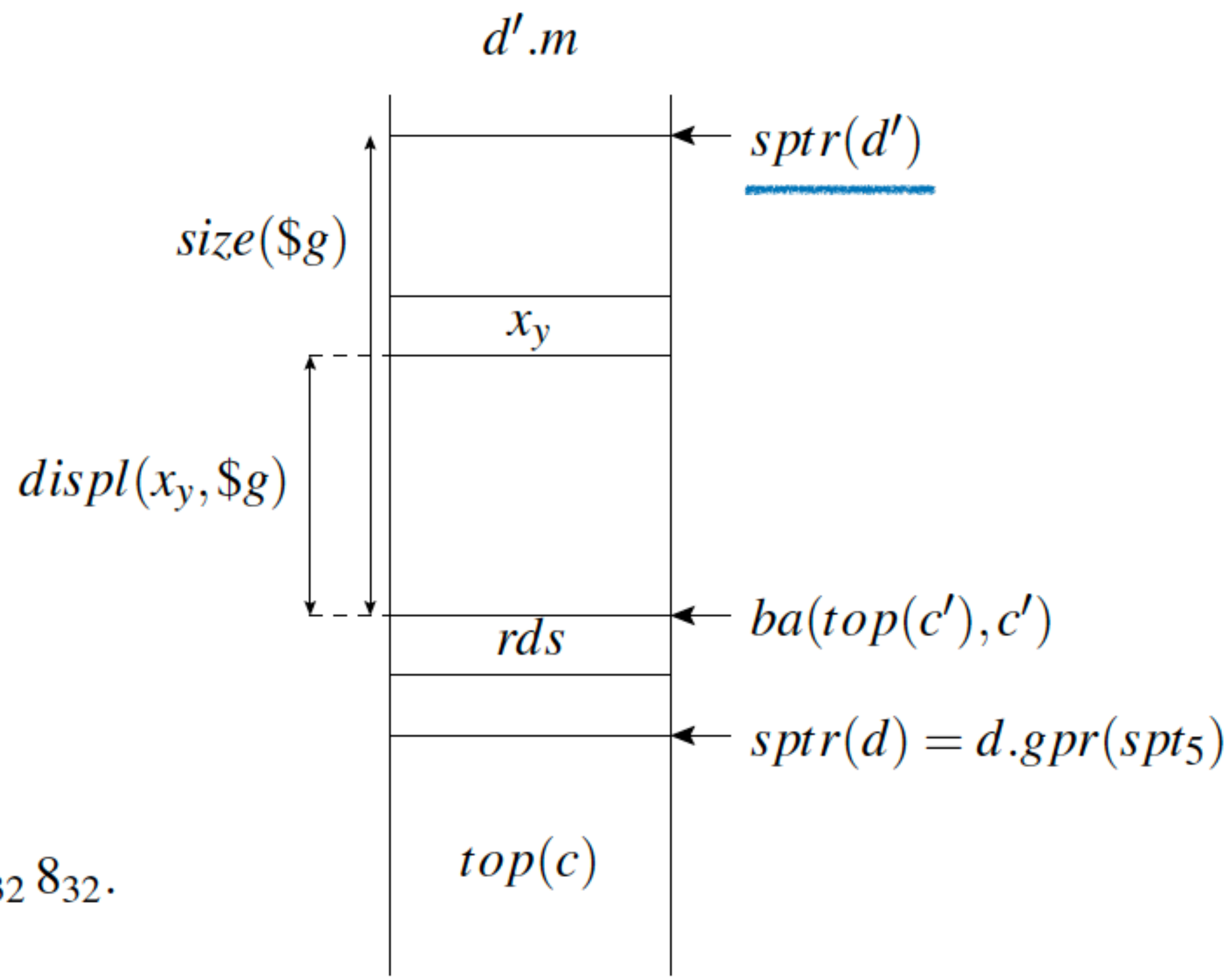
$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

$$R(n0) = 0$$

$$R(n_y) = 1.$$

$$d.gpr(spt_5) = ba(top(c'), c') -_{32} 8_{32}.$$



return destination

```
code(n0)
```

then

$$d_0.gpr(j_5) = ba(lv(e,c),c).$$

increase stack pointer

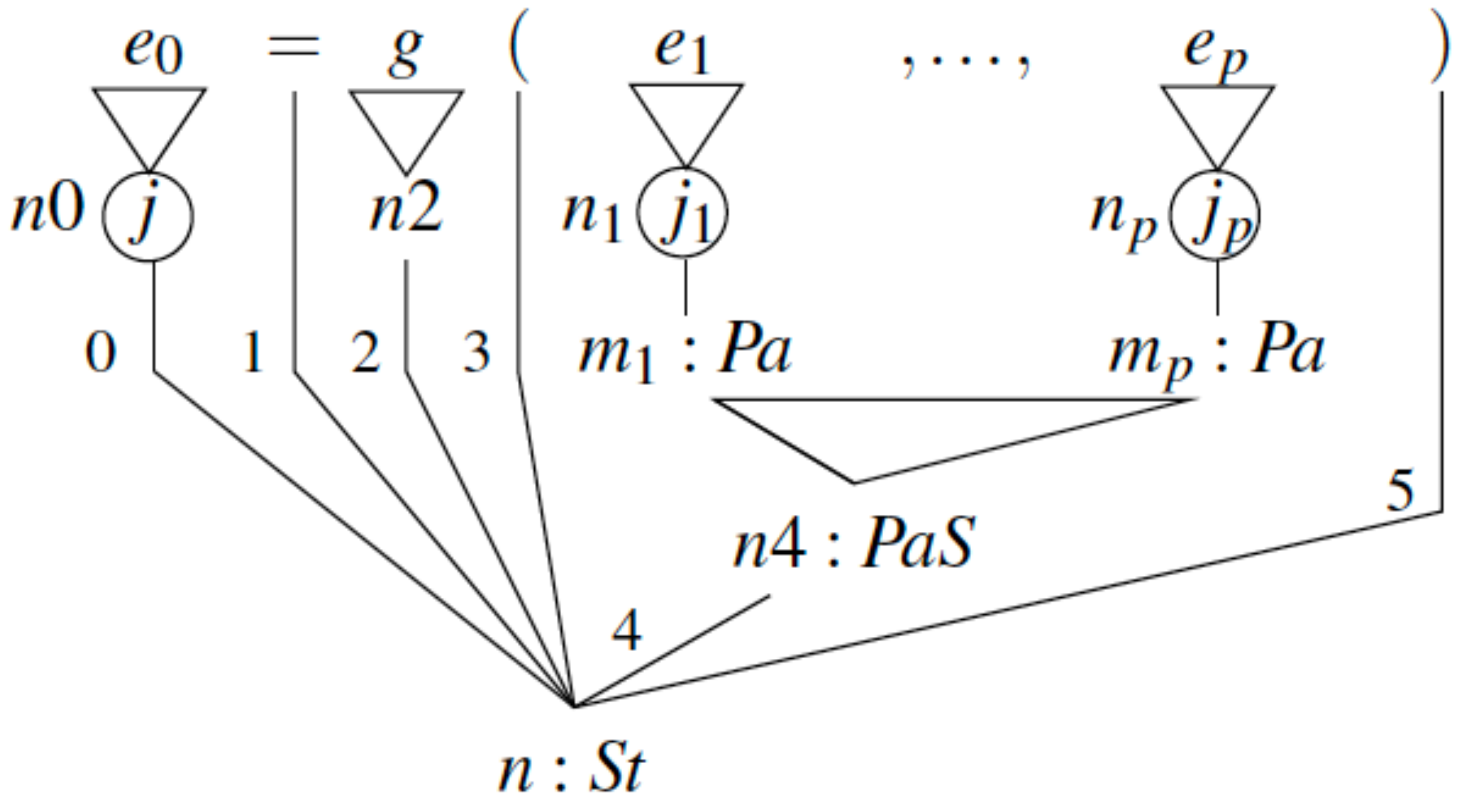
```
addi spt spt size($g)+8
```

then

$$spt\text{-}consis(c', d')$$

code generation

function call



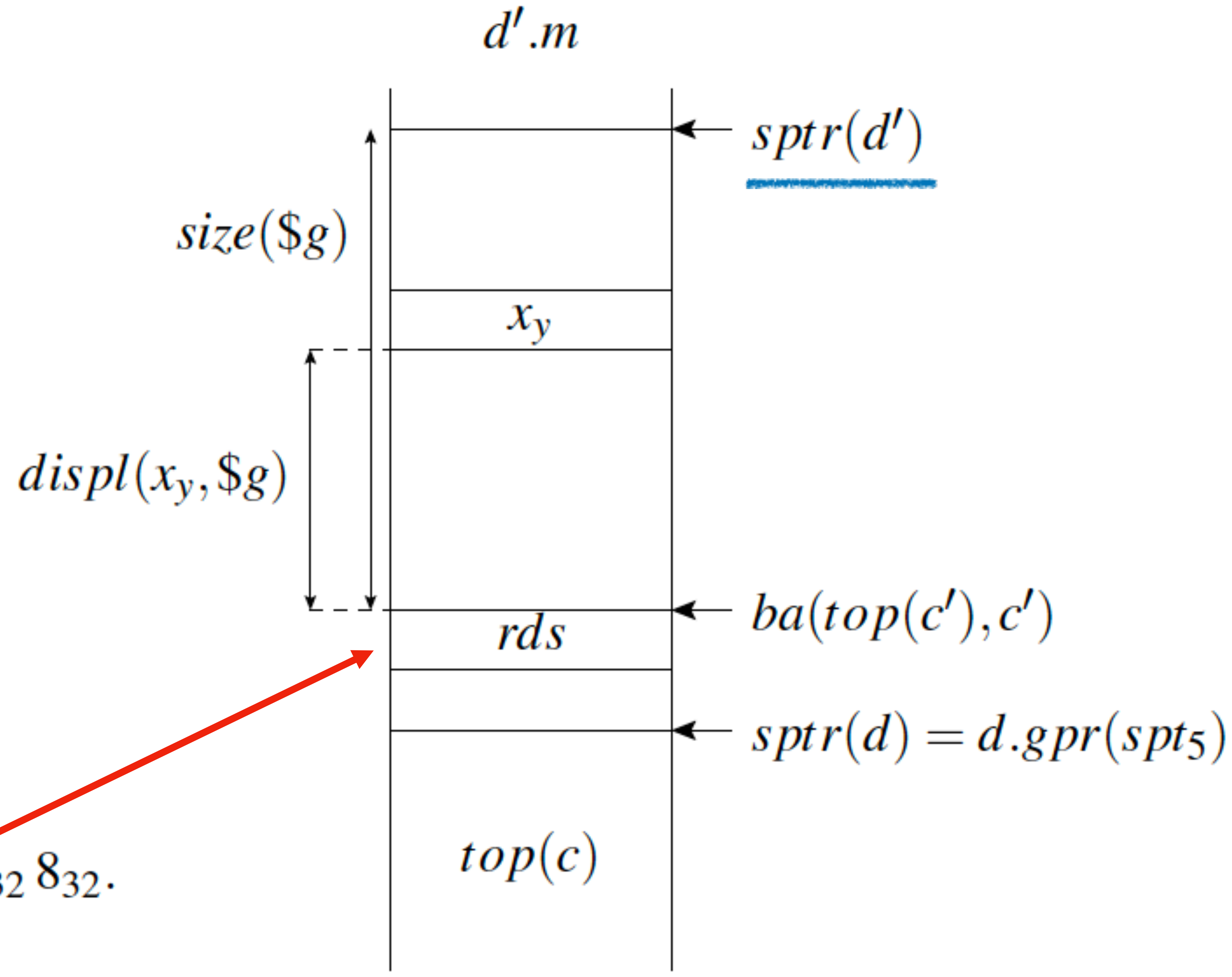
$n_y = m_y 0.$

$m_y = se(n4, y)$

$R(n0) = 0$

$R(n_y) = 1.$

$d.gpr(spt_5) = ba(top(c'), c') -_{32} 8_{32}.$



return destination

```
code(n0)
```

then

$d_0.gpr(j_5) = ba(lv(e,c),c).$

```
sw    j spt -(size($g)+4)
```

$rds\text{-}consis(c', d').$

increase stack pointer

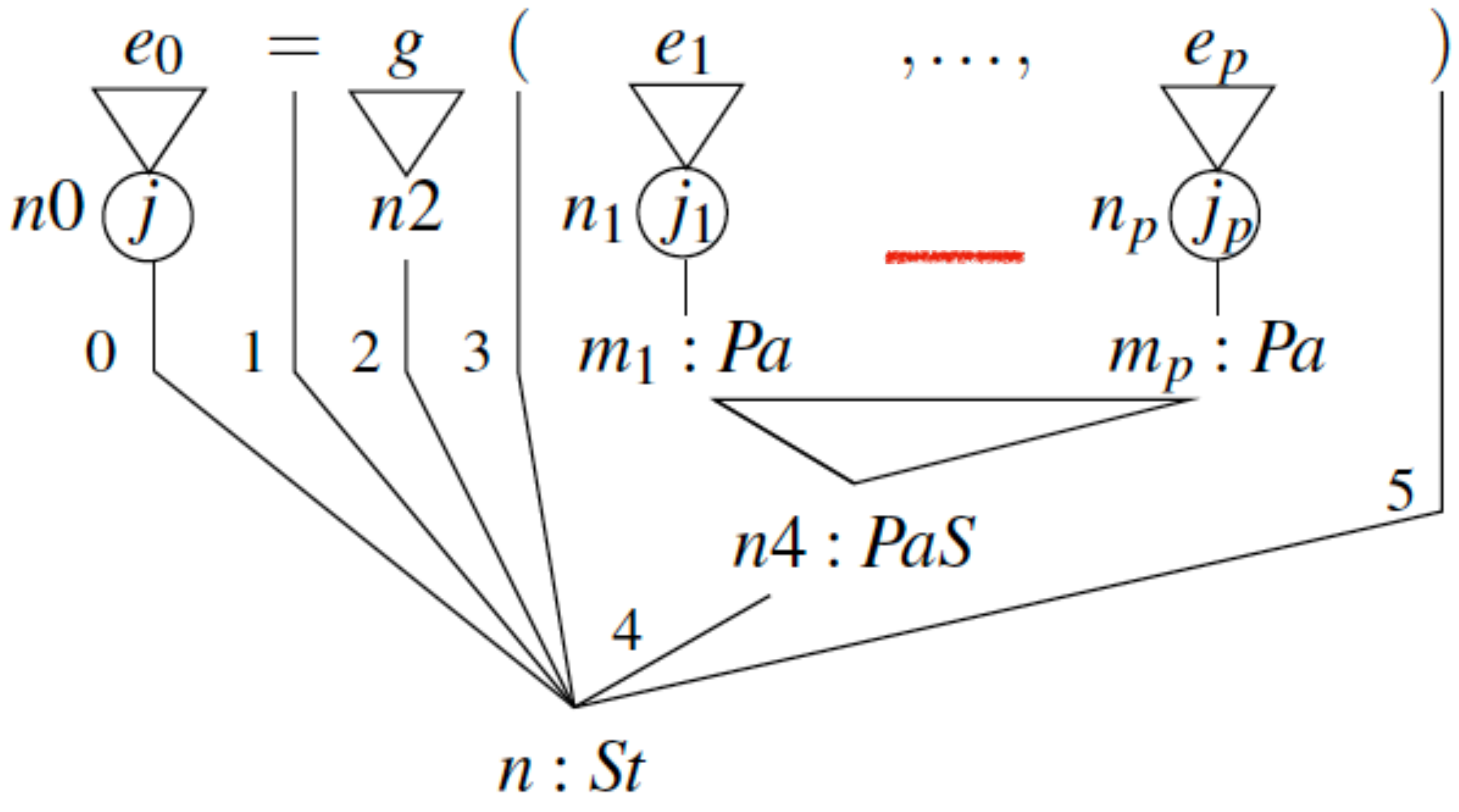
```
addi  spt  spt  size($g)+8
```

then

$spt\text{-}consis(c', d')$

code generation

function call



parameter passing

$$n_y = m_y 0.$$

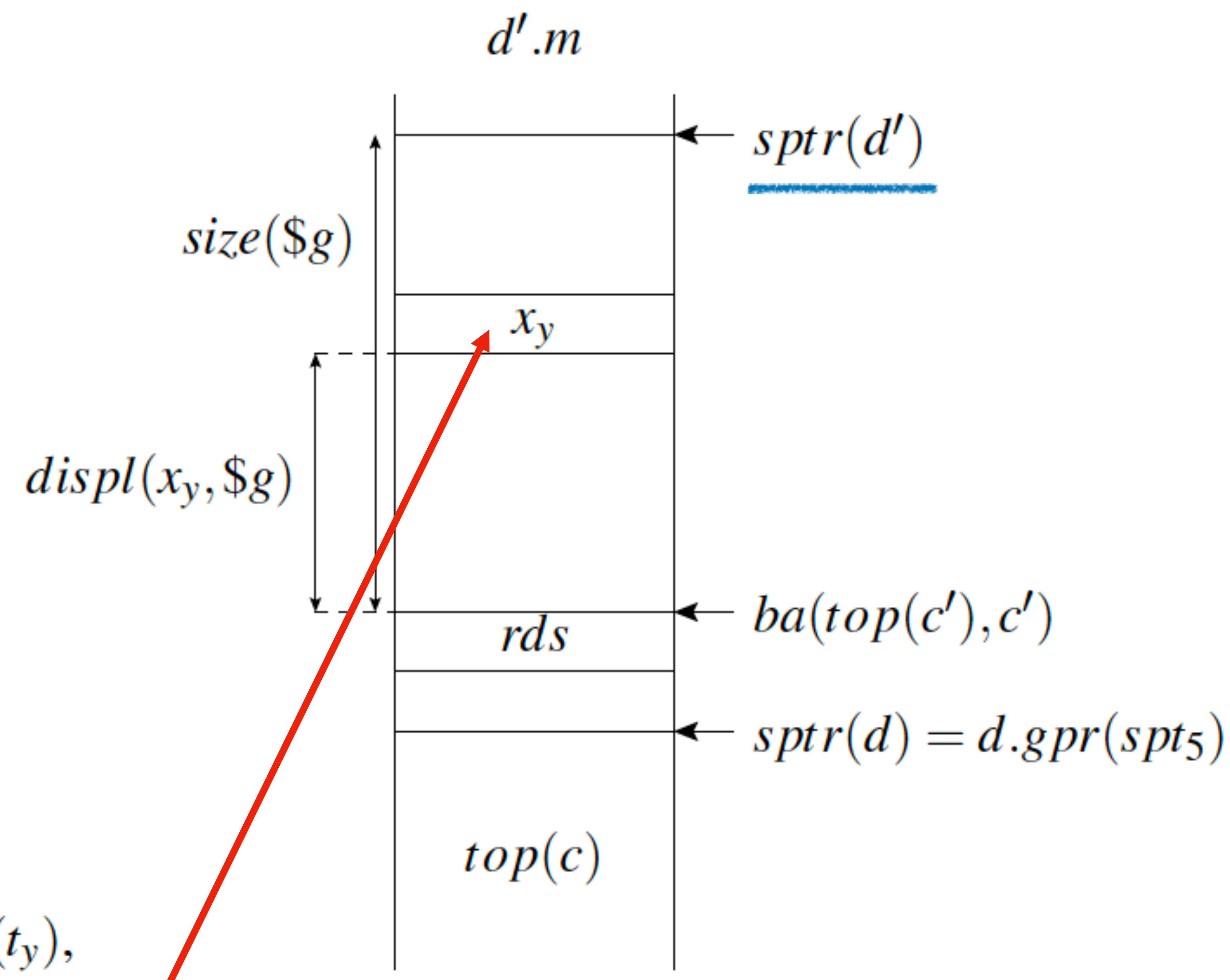
$$m_y = se(n4, y)$$

```
code(ny)
```

then

$$d_y.gpr(j_{y_5}) = \begin{cases} ba(va(e_y,c),c), & pointer(t_y), \\ enc(va(e_y,c),t_y), & t_y \in ET. \end{cases}$$

```
sw    jy spt -size($g)+displ(xy,$g)
```



increase stack pointer

```
addi spt spt size($g)+8
```

then

$$spt\text{-}consis(c', d')$$

return destination

```
code(n0)
```

then

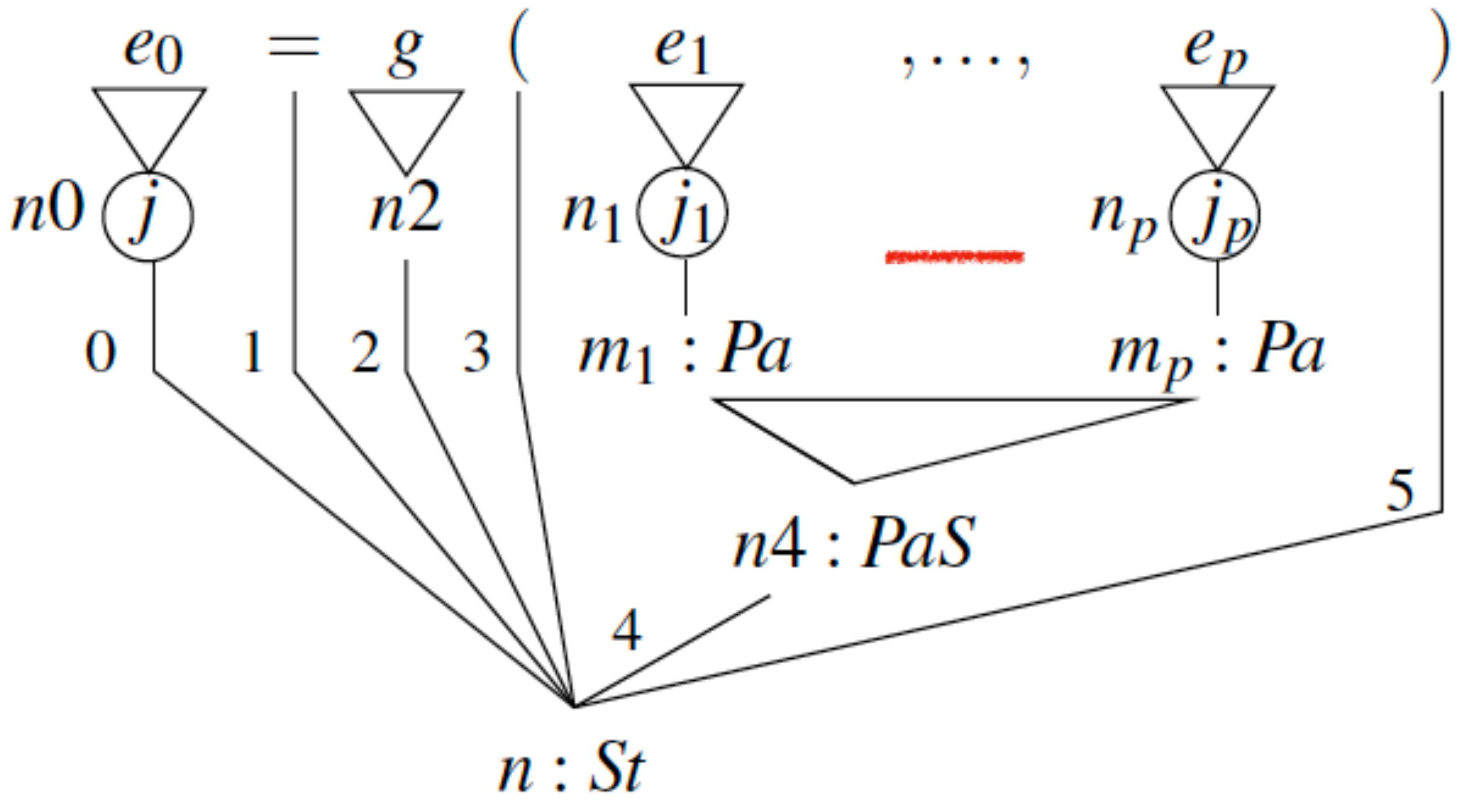
$$d_0.gpr(j_5) = ba(lv(e,c),c).$$

```
sw   j spt -(size($g)+4)
```

$$rds\text{-}consis(c', d').$$

code generation

function call



return destination

```
code(n0)
```

then

$$d_0.gpr(j_5) = ba(lv(e,c),c).$$

```
sw   j spt -(size($g)+4)
```

$$rds\text{-}consis(c',d').$$

$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

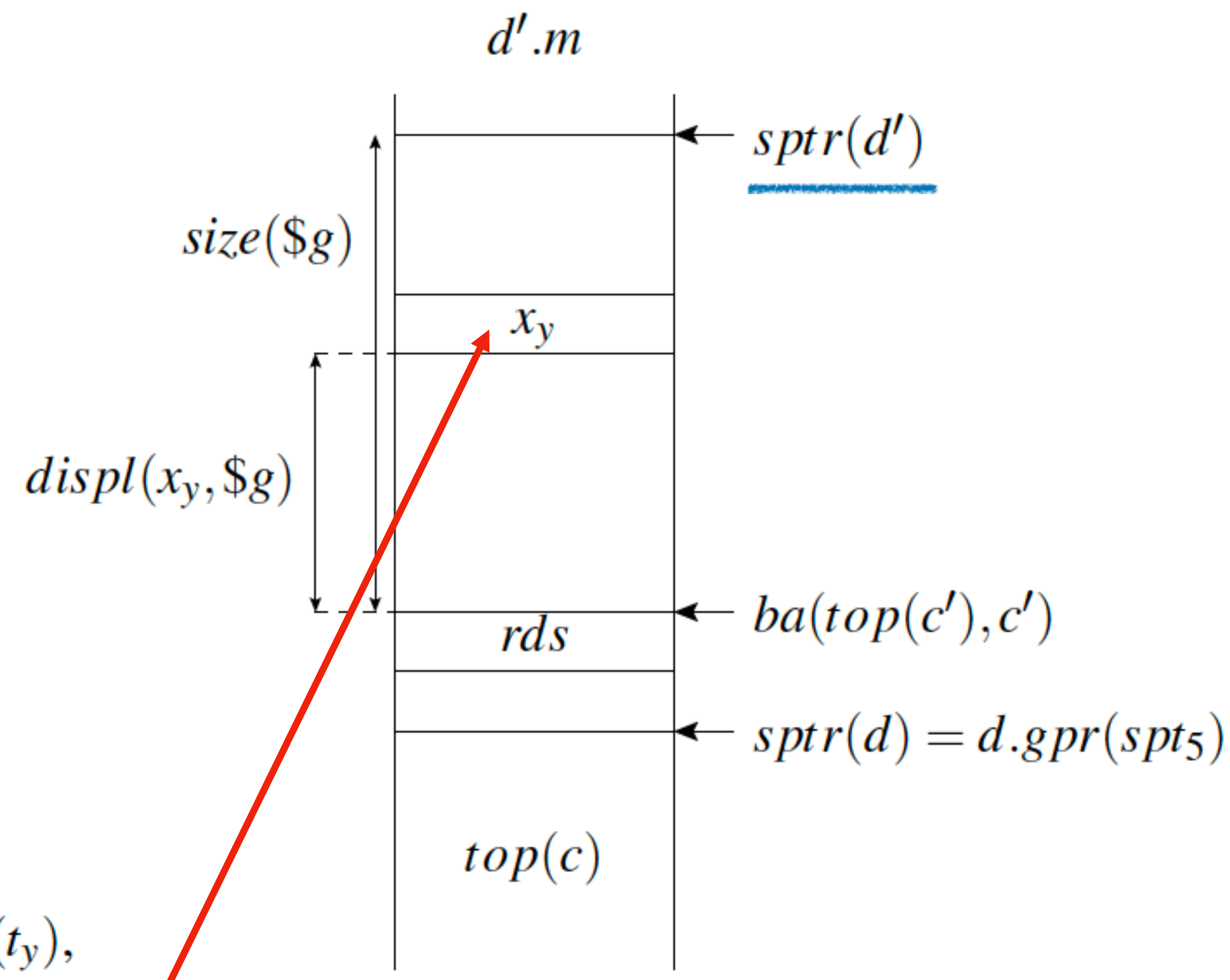
parameter passing

```
code(ny)
```

then

$$d_y.gpr(j_{y_5}) = \begin{cases} ba(va(e_y,c),c), & pointer(t_y), \\ enc(va(e_y,c),t_y), & t_y \in ET. \end{cases}$$

```
sw   jy spt -size($g)+displ(xy,$g)
```



initialize local variables

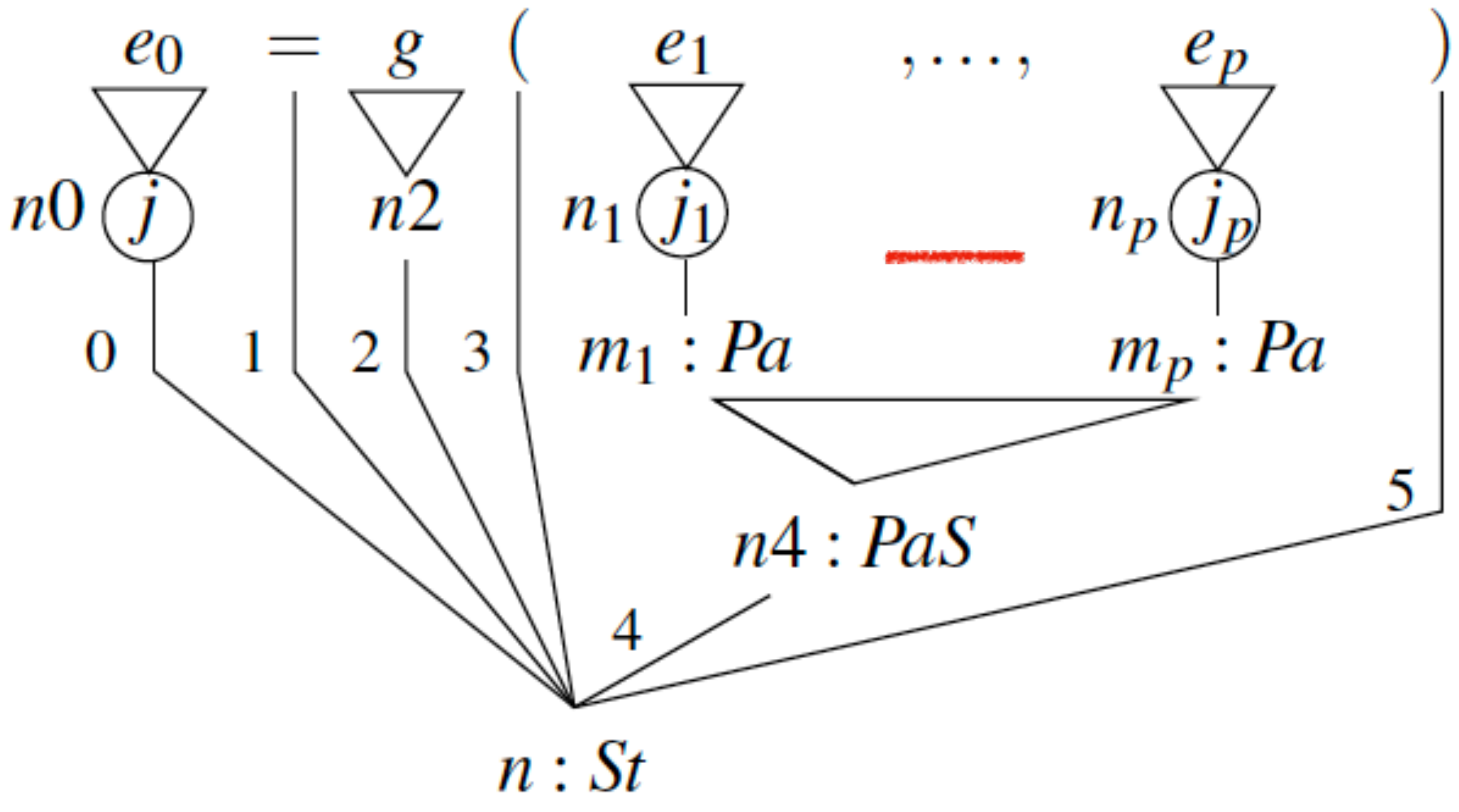
$$displ(x_{p+1}, \$g).$$ first local variable

$$z = (\sum_{y>p} size(t_y))/4.$$ size of all local variables

```
add    1 spt -size($g)+displ(xp+1,$g)
addi   2 0 z
zero(1,2)
```

code generation

function call



$$n_y = m_y 0.$$

$$m_y = se(n4, y)$$

parameter passing

```
code(ny)
```

then

$$d_y.gpr(j_{y_5}) = \begin{cases} ba(va(e_y,c),c), & pointer(t_y), \\ enc(va(e_y,c),t_y), & t_y \in ET. \end{cases}$$

```
sw    jy  spt  -size($g)+displ(xy,$g)
```

then $e\text{-}consis(c', d')$ $p\text{-}consis(c', d')$.

return destination

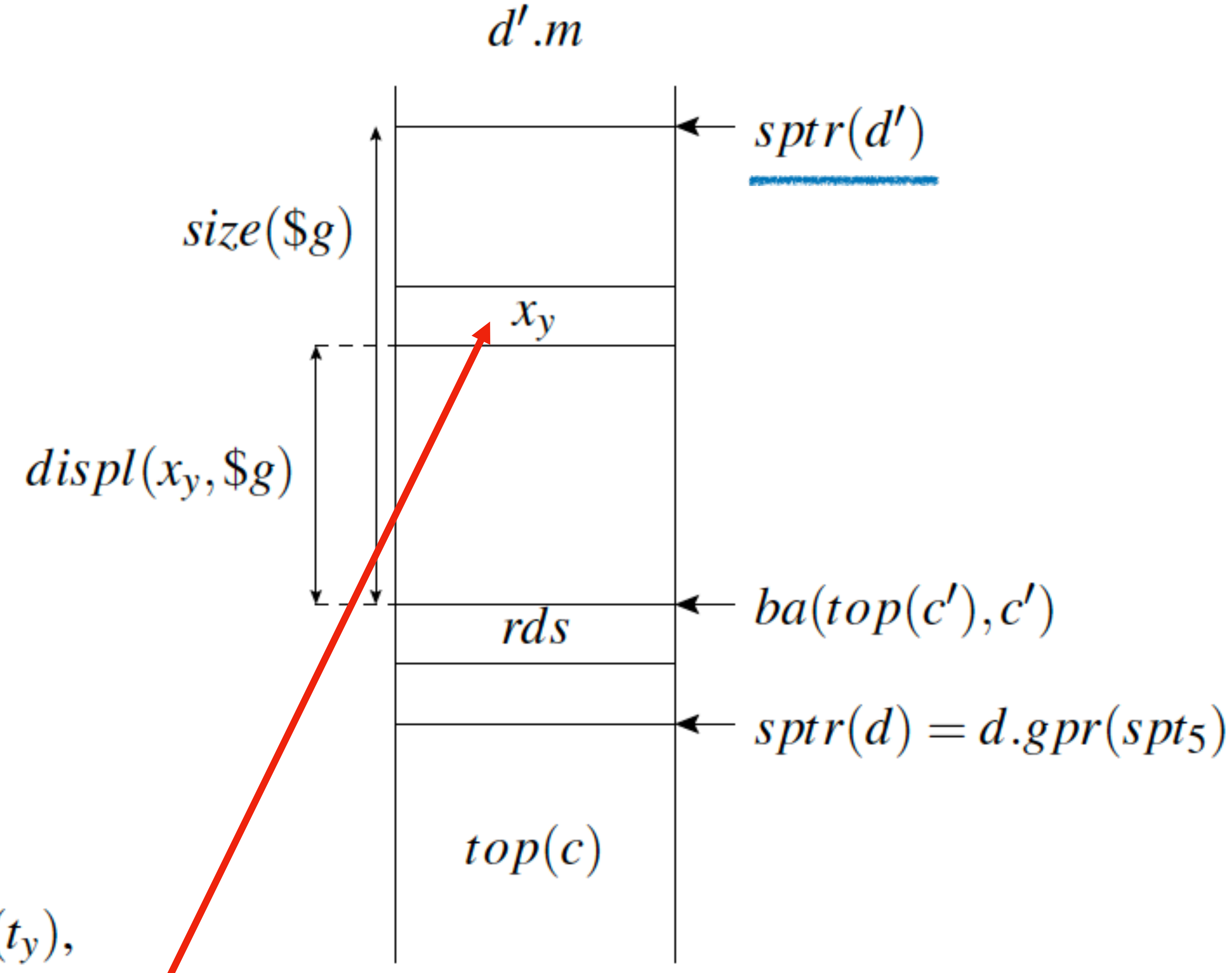
```
code(n0)
```

then

$$d_0.gpr(j_5) = ba(lv(e,c),c).$$

```
sw    j  spt  -(size($g)+4)
```

$$rds\text{-}consis(c', d').$$



initialize local variables

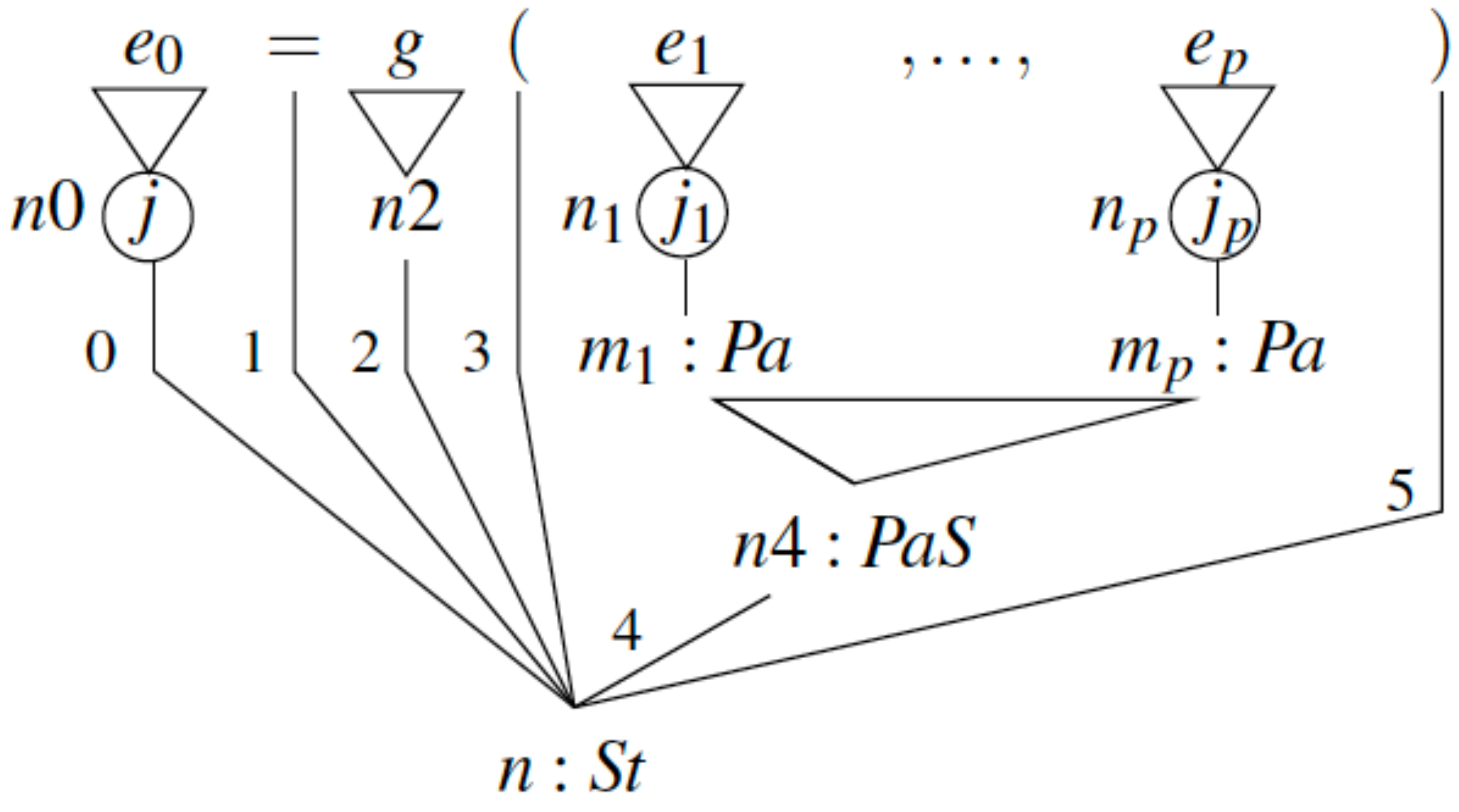
$displ(x_{p+1}, \$g)$. first local variable

$z = (\sum_{y>p} size(t_y))/4.$ size of all local variables

```
add    1  spt  -size($g)+displ(xp+1,$g)
addi   2  0  z
zero(1,2)
```

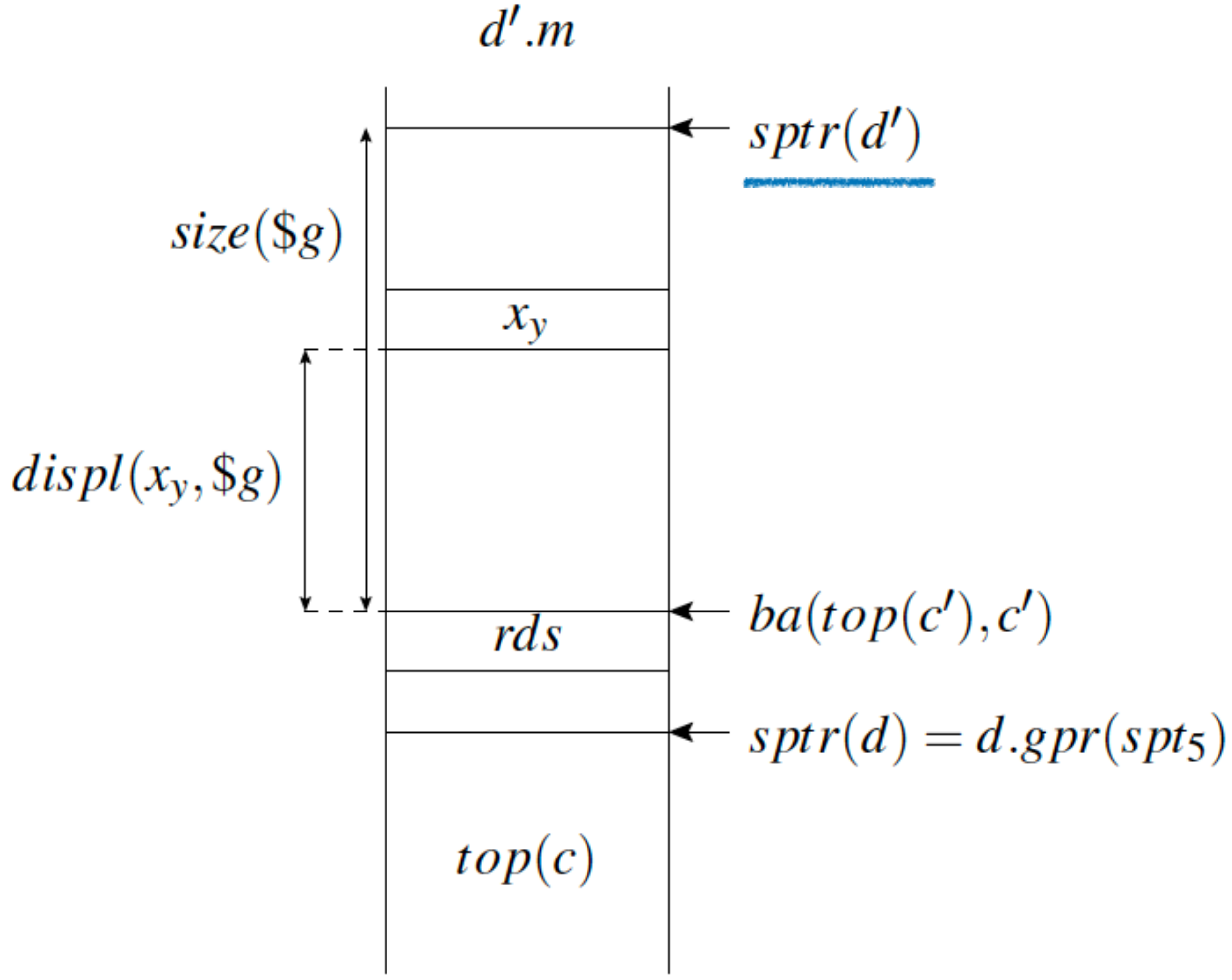
code generation

function call



jump and link

```
jal   0bstart(nbody(g))[27:2]
```



exercise 6.2

$$E(X) = E(a + b) = E(a) + E(b) = 3.5 + 3.5 = 7$$