Example

What TinyJ VM code would the TinyJ compiler generate for the declaration `static int x, y = 10;` in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

Solution

We use the static memory allocation rule on p. 3 and Code Generation Rules 1 and 2 on p. 9 of:
https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf

x is allocated data memory address **0**.
y is allocated data memory address **1**.

[*Blue* items are on EXPRSTACK *after* execution of each instruction.]

The generated instructions are:

```
0: PUSHSTATADDR 1      Pushes pointer to y.
1: PUSHNUM 10          Pushes 10.
2: SAVETOADDR          Pops 10.
                       Pops ptr to y.
                       Stores 10 into y's location.
```

Example

What is the 1st VM instruction generated by the TinyJ compiler for each of the methods main, f, and g in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

Solution

From Code Generation Rule 4, the instruction is:

INITSTKFRM <no. of stackframe locations given to local vars declared in the method's body>

- main's stackframe has **no locations for local vars** declared in main's body--there are no such vars ***in this example***!

From the stack-dynamic memory allocation rules on p. 3 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf:

- The 3 local variables declared in f's body are given 3 different locations in each stackframe of f.
- The only local variable declared in g's body is given 1 location in each stackframe of g.

Hence main's code begins with: 3: **INITSTKFRM 0**
f's code begins with: ?: **INITSTKFRM 3**
g's code begins with: ?: **INITSTKFRM 1**

Code memory address is ***not known*** until main and f have been translated.

Code memory address is ***not known*** until main has been translated.

Example

What TinyJ VM code would the TinyJ compiler generate for the 1st two statements of main method in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

The two statements are: **System.out.print("Enter num: ");**
x = input.nextInt();

Solution

The static memory allocation rules on p. 3 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf imply:

- x is allocated data memory address **0**.
- y is allocated data memory address **1**.
- The 11 characters of "Enter num: " are allocated data memory addresses **2, 3, …, 11, 12**.

[*Blue* items are on EXPRSTACK *after* execution of each instruction.]

Hence the generated instructions are:

```
4: WRITESTRING 2 12   Writes  "Enter num: "  to the screen.
5: PUSHSTATADDR 0     Pushes pointer to x.
6: READINT            Reads an int from kbd; pushes its value.
7: SAVETOADDR         Pops the int; pops the pointer to x;
                      stores the int into x's location.
```

What TinyJ VM code does the TinyJ compiler generate for this method?

```
static void g (int d, int e)
{
  int z;
  y = d / e;
}
```

Solution

The static and stack-dynamic memory allocation rules imply:

- y is given data memory address 1.
- z is given the location with offset +1 in a stackframe of g.
- e is given the location with offset -2 in a stackframe of g.
- d is given the location with offset -3 in a stackframe of g.

Generated Code: [*Blue* items are on EXPRSTACK *after execution* of each instruction.]

```
INITSTKFRM 1    From Code Generation Rule 4.

PUSHSTATADDR 1  Pushes ptr to y.
PUSHLOCADDR -3  Pushes ptr to d.
LOADFROMADDR    Replaces ptr to d with d's value on top of stack.
PUSHLOCADDR -2  Pushes ptr to e.
LOADFROMADDR    Replaces ptr to e with e's value on top of stack.
DIV             Pops e's value; pops d's value; pushes (d/e)'s value.
SAVETOADDR      Pops (d/e)'s value;
                pops ptr to y;
                stores (d/e)'s value into y's location.
RETURN 2        From Code Generation Rule 6, as g has 2 parameters.
```

What TinyJ VM code does the TinyJ compiler generate for the statement `return y - a % u;` in method `f` in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

Solution

The static and stack-dynamic memory allocation rules imply:

- y is given the data memory address 1.
- a is given the location with offset -4 in a stackframe of f.
- u is given the location with offset +3 in a stackframe of f.

In view of Code Generation Rule 7, the generated code is:

```
PUSHSTATADDR 1  Pushes ptr to y.
LOADFROMADDR    Replaces ptr to y with y's value on top of stack.
PUSHLOCADDR -4  Pushes ptr to a.
LOADFROMADDR    Replaces ptr to a with a's value on top of stack.
PUSHLOCADDR +3  Pushes ptr to u.
LOADFROMADDR    Replaces ptr to u with u's value on top of stack.
MOD             Pops u's and a's values; pushes (a%u)'s value.
SUB             Pops (a%u)'s and y's values; pushes (y-a%u)'s value.
RETURN 3        From Code Generation Rule 7, as f has 3 parameters.
```

Code Generation Rule 4 implies that the first TinyJ VM instruction generated for the method f in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf is **INITSTKFRM 3**. Hand-translation of main shows that this instruction is placed in code memory at address 34.

What code does the compiler generate so f(21,22,23)'s value will be on top of EXPRSTACK when the next instruction in main's code is executed?

Solution

In view of Code Generation Rule 8, the generated code is:

```
PUSHNUM 21          Pushes 21.
PASSPARAM           Pops 21.
                    Stores 21 in 1st param's loc in f's stackframe.
PUSHNUM 22          Pushes 22.
PASSPARAM           Pops 22.
                    Stores 22 in 2nd param's loc in f's stackframe.
PUSHNUM 23          Pushes 23.
PASSPARAM           Pops 23.
                    Stores 23 in 3rd param's loc in f's stackframe.
CALLSTATMETHOD 34   Next instr. to be executed will be: 34 INITSTKFRM 3
                    f's execution will leave f(21,22,23)'s value on stack.
```

What TinyJ VM code does the TinyJ compiler generate for the statement `System.out.println(y + f(21,22,23));` in `main` in the program on p. 7 of

https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

Solution

Recall that y is given the data memory address 1.

Generated Code:

```
PUSHSTATADDR 1    Pushes ptr to y.
LOADFROMADDR      Replaces ptr to y with y's value on top of stack.
PUSHNUM 21
PASSPARAM
PUSHNUM 22
PASSPARAM
PUSHNUM 23
PASSPARAM
CALLSTATMETHOD 34
ADD               Pops value returned by f(21,22,23); pops y's value;
                  pushes (y+f(21,22,23))'s value.
WRITEINT          Pops (y+f(21,22,23))'s value; writes it to the screen.
WRITELNOP         Writes a newline to the screen.
```

SEE EARLIER SLIDE.

Execution of method f puts the value returned by f(21,22,23) on top of stack.

What TinyJ VM code does the TinyJ compiler generate for the statement `f(17,y,x-y);` in `main` in the program on p. 7 of https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf ?

Solution

Recall: x is given data mem. addr. **0**; y is given data mem. addr. **1**.
f's code begins with **34 INITSTKFRM 3**

Generated Code:

```
PUSHNUM 17       Pushes 17.
PASSPARAM        Pops 17 & stores it in 1st param's loc in f's stackfrm.
PUSHSTATADDR 1   Pushes ptr to y.
LOADFROMADDR     Replaces ptr to y with y's value on top of stack.
PASSPARAM        Pops y's value & stores it in 2nd param's loc in f's stackfrm.
PUSHSTATADDR 0   Pushes ptr to x.
LOADFROMADDR     Replaces ptr to x with x's value on top of stack.
PUSHSTATADDR 1   Pushes ptr to y.
LOADFROMADDR     Replaces ptr to y with y's value on top of stack.
SUB              Pops y's and x's values; pushes (x-y)'s value.
PASSPARAM        Pops (x-y)'s value & stores it in 3rd param's loc in f's stackfrm.
CALLSTATMETHOD 34 Next instr. to be executed will be: 34 INITSTKFRM 3
                    f's execution will leave f(17,y,x-y)'s value on stack.
DISCARDVALUE     Pops f(17,y,x-y)'s value as per Code Generation Rule 9.
```

Hand-translation of the program on p. 7 of
https://euclid.cs.qc.cuny.edu/316/Memory-allocation-VM-instruction-set-and-hints-for-asn-2.pdf
shows that the ***first*** instruction of method g is placed in code memory at address 60.

What code is generated for the statement g(c,b+u); *in method* f?

Solution

- b is given the location with offset -3 in f's stackframe.
- c is given the location with offset -2 in f's stackframe.
- u is given the location with offset +3 in f's stackframe.

Generated Code:

```
PUSHLOCADDR -2  Pushes ptr to c.
LOADFROMADDR    Replaces ptr to c with c's value on top of stack.
PASSPARAM       Pops c's value & stores it in 1st param's loc in f's stackfrm.
PUSHLOCADDR -3  Pushes ptr to b.
LOADFROMADDR    Replaces ptr to b with b's value on top of stack.
PUSHLOCADDR +3  Pushes ptr to u.
LOADFROMADDR    Replaces ptr to u with u's value on top of stack.
ADD             Pops u's and b's values; pushes (b+u)'s value.
PASSPARAM       Pops (b+u)'s value & stores it in 2nd param's loc in f's stackfrm.
CALLSTATMETHOD 60 Next instr. to be executed will be g's 1st instr.
NOP             Does nothing. See Code Generation Rule 9.
```

EXECUTION OF VARIOUS TINYJ VM INSTRUCTIONS

BEFORE execution of: WRITESTRING 3 9

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

EXPRSTACK

address	DATA MEMORY
0	
1	
2	
3	'T'
4	'h'
5	'e'
6	' '
7	'C'
8	'a'
9	't'
10	
11	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
	WRITESTRING 3 9

AFTER execution of: WRITESTRING 3 9

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

address	DATA MEMORY
0	
1	
2	
3	'T'
4	'h'
5	'e'
6	' '
7	'C'
8	'a'
9	't'
10	
11	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
	WRITESTRING 3 9

EXPRSTACK

NOTE: In this example, execution of
WRITESTRING 3 9
writes the string
The Cat
to the screen.

BEFORE execution of: PUSHNUM 23

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

address	DATA MEMORY

address	HEAP (Part of Data Memory)

address	CODE MEMORY
	PUSHNUM 23

EXPRSTACK
???
⋮
???

AFTER execution of: PUSHNUM 23

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

address	DATA MEMORY

address	HEAP (Part of Data Memory)

address	CODE MEMORY
	PUSHNUM 23

EXPRSTACK
???
⋮
???
23

BEFORE execution of: PUSHSTATADDR 17

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

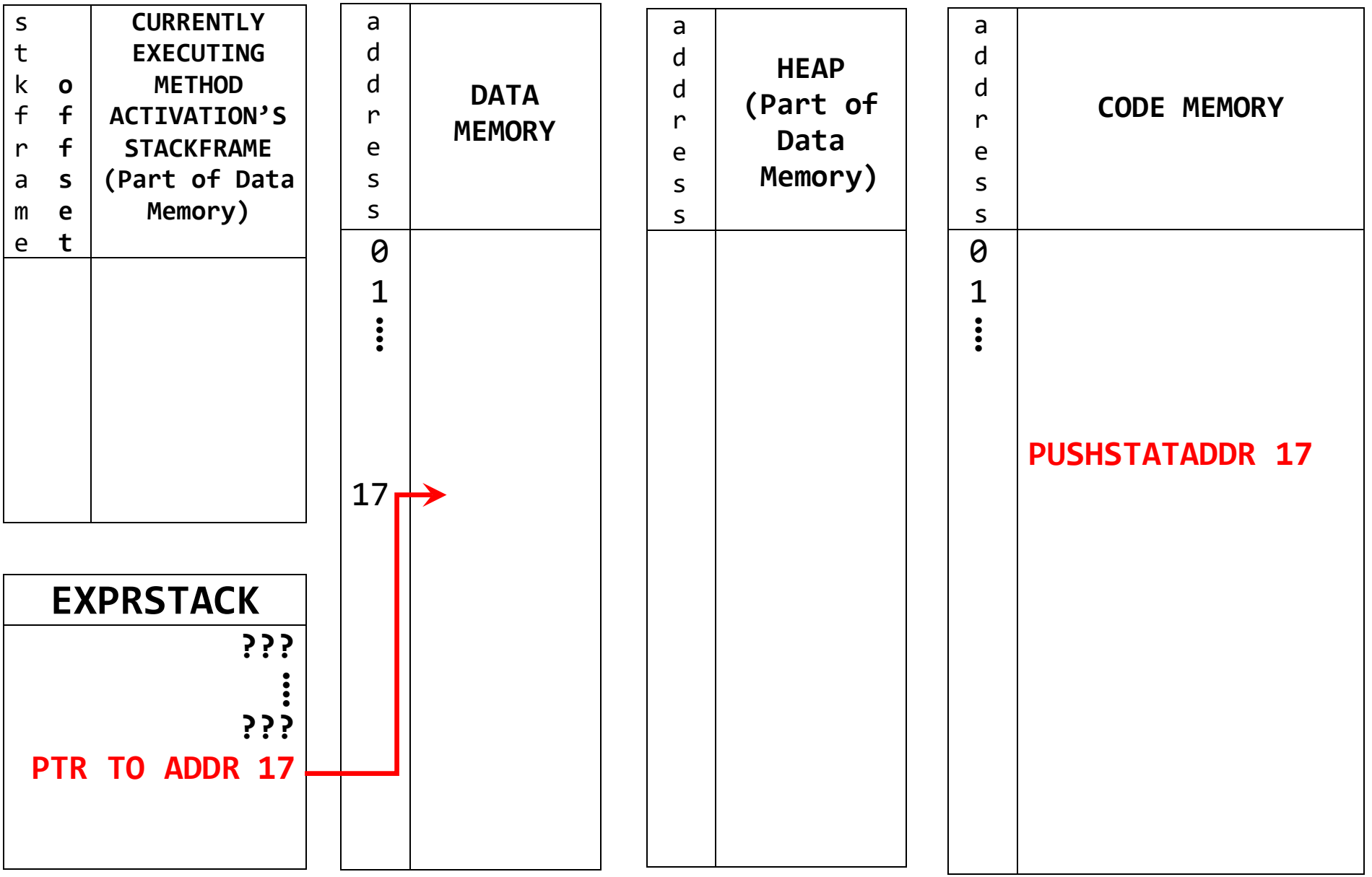
address	DATA MEMORY
0	
1	
⋮	
17	

address	HEAP (Part of Data Memory)

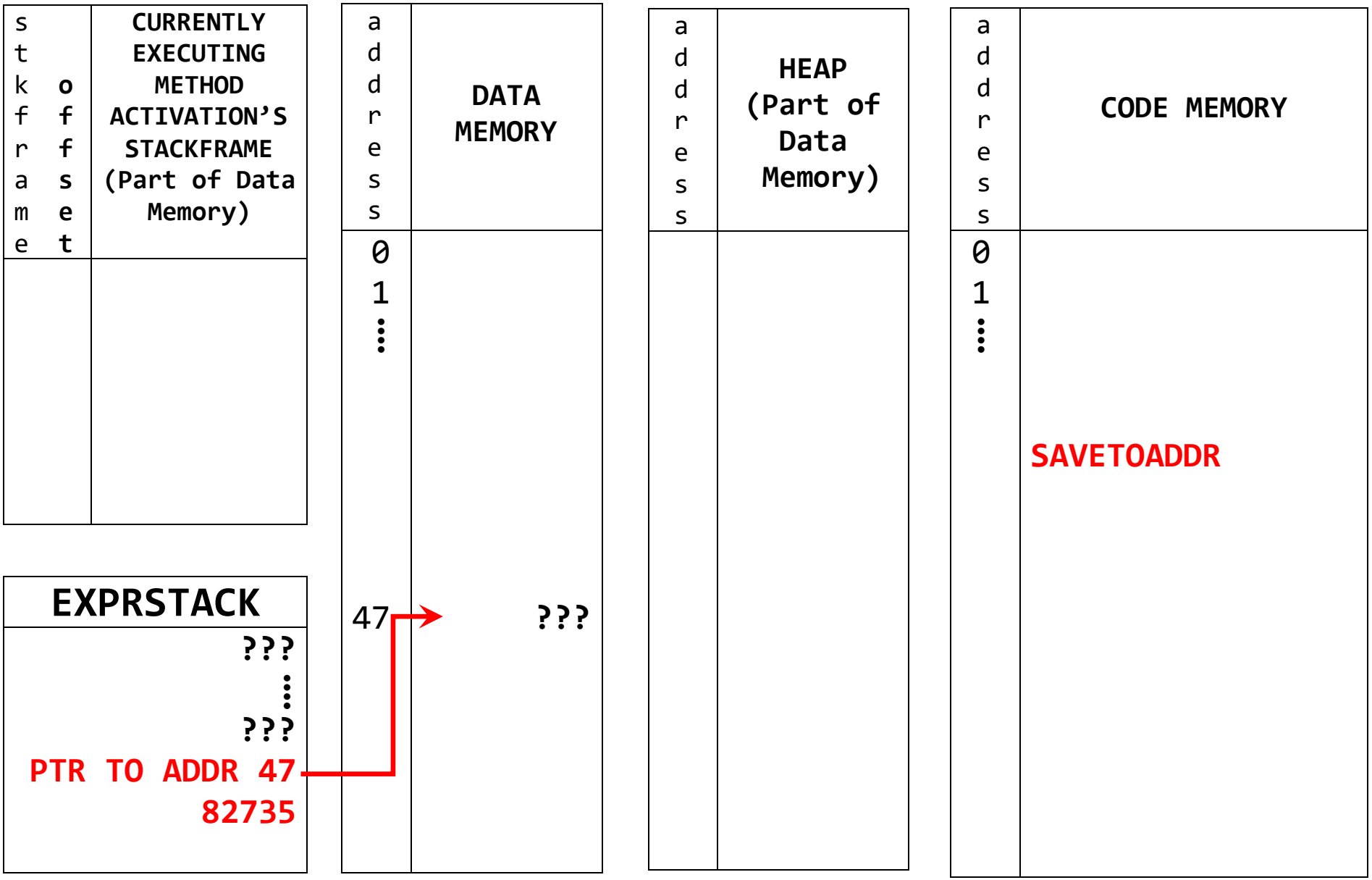
address	CODE MEMORY
0	
1	
⋮	
	PUSHSTATADDR 17

EXPRSTACK
???
⋮
???

AFTER *execution of*: **PUSHSTATADDR 17**



BEFORE execution of **SAVETOADDR**



AFTER *execution of* **SAVETOADDR**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

EXPRSTACK
???
⋮
???

address	DATA MEMORY
0	
1	
⋮	
47	82735

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	SAVETOADDR

BEFORE execution of READINT

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

EXPRSTACK
???
⋮
???

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	READINT

AFTER *execution of* **READINT**

stkframeoffset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	READINT

EXPRSTACK
???
⋮
???
int entered on kbd

BEFORE execution of: PUSHLOCADDR 3

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

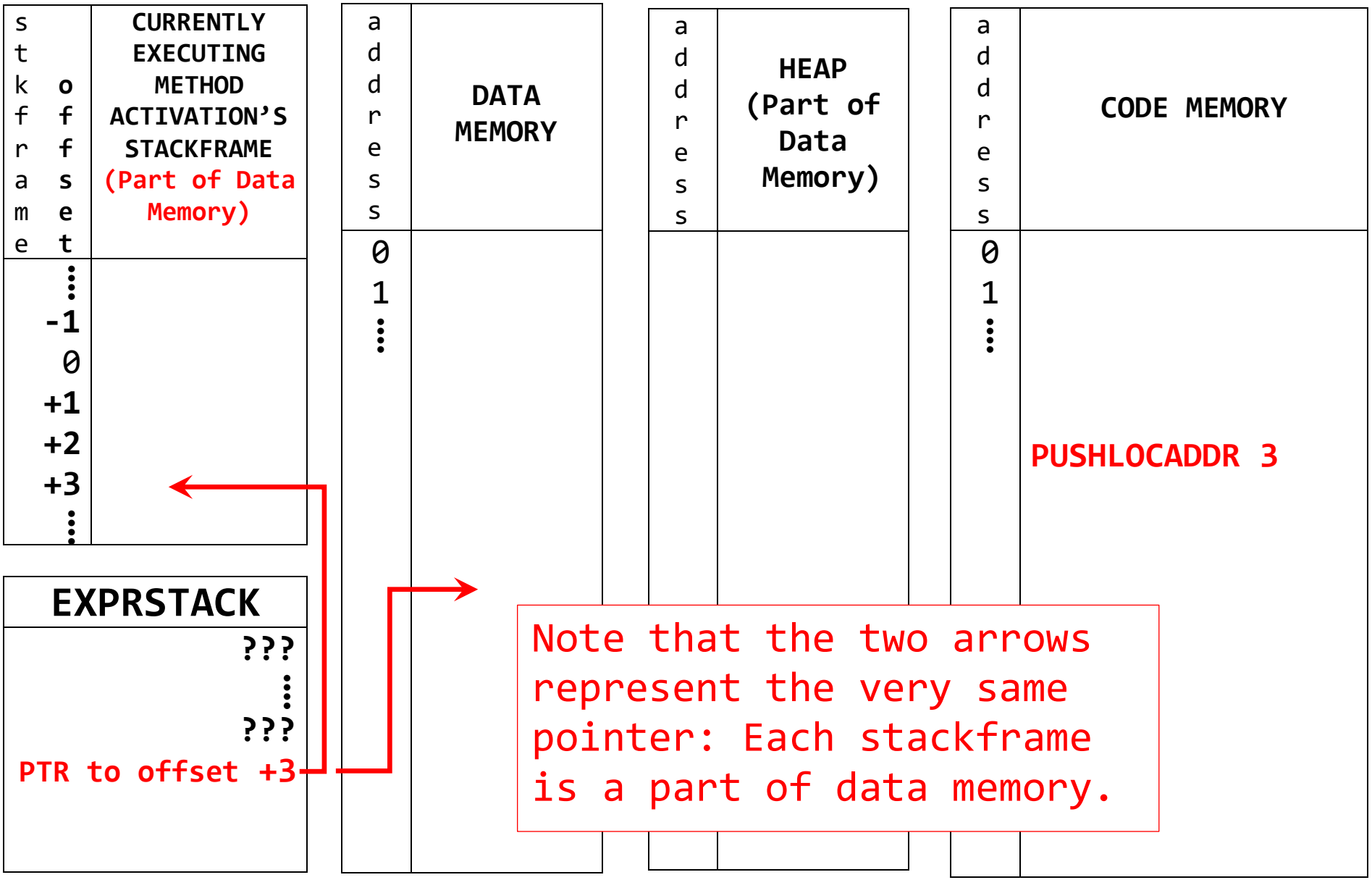
EXPRSTACK
???
⋮
???

address	DATA MEMORY
0	
1	
⋮	

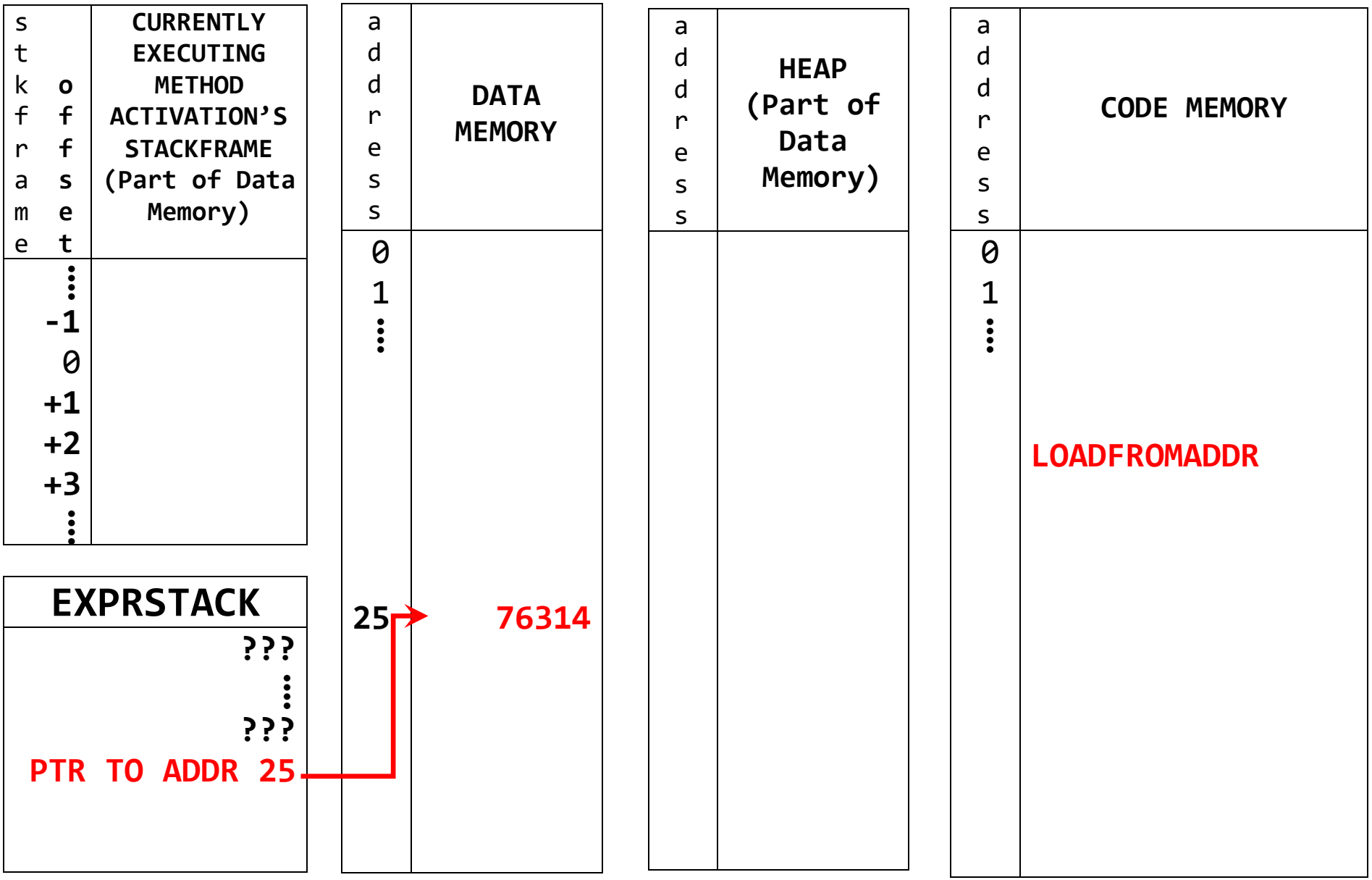
address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	PUSHLOCADDR 3

AFTER *execution of*: **PUSHLOCADDR 3**



BEFORE execution of: LOADFROMADDR



AFTER execution of: LOADFROMADDR

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
76314

address	DATA MEMORY
0	
1	
⋮	
25	76314

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	LOADFROMADDR

BEFORE execution of: SUB

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
3
12

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	SUB

AFTER *execution of*: **SUB**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
-9

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	SUB

BEFORE execution of: AND

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
1
0

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	AND

AFTER execution of: **AND**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
0

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	AND

BEFORE execution of: LE ("Less than or Equal to")

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
7
12

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	LE

AFTER *execution of*: LE (“Less than or Equal to”)

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION’S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
1

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	LE

BEFORE execution of: CHANGESIGN

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
-12

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	CHANGESIGN

AFTER execution of: CHANGESIGN

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
12

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	CHANGESIGN

BEFORE *execution of*: **NOT**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
1

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	NOT

AFTER *execution of*: **NOT**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
0

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	NOT

BEFORE execution of: WRITEINT

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
8276

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	WRITEINT

AFTER *execution of*: **WRITEINT**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	WRITEINT

NOTE: The popped integer (8276 in this example) is written to the screen!

BEFORE execution of: DISCARDVALUE

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

EXPRSTACK
???
⋮
???
7263

address	DATA MEMORY
0	
1	
⋮	

address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	DISCARDVALUE

AFTER *execution of*: **DISCARDVALUE**

stkframe offset	CURRENTLY EXECUTING METHOD ACTIVATION'S STACKFRAME (Part of Data Memory)
⋮	
-1	
0	
+1	
+2	
+3	
⋮	

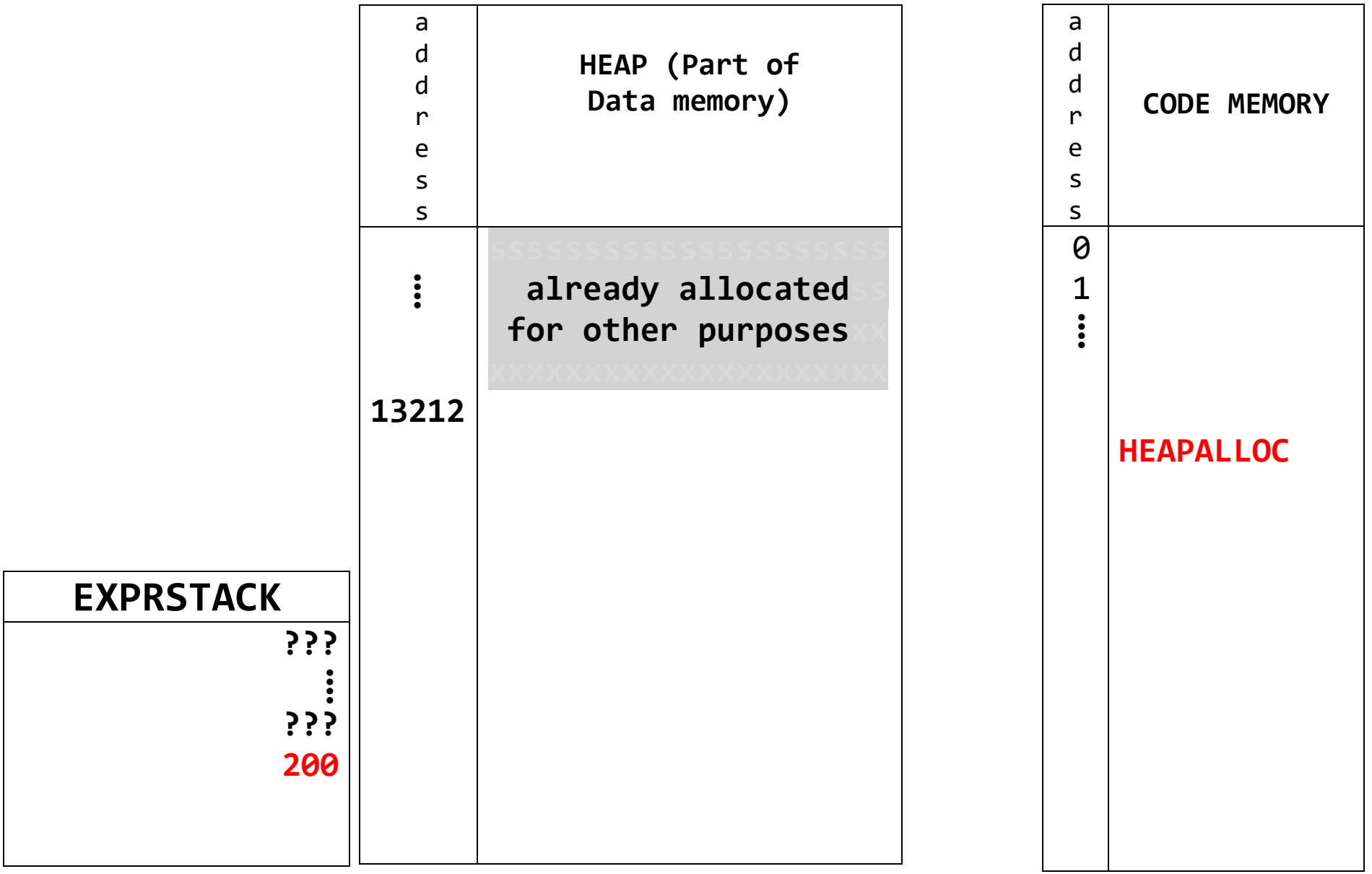
EXPRSTACK
???
⋮
???

address	DATA MEMORY
0	
1	
⋮	

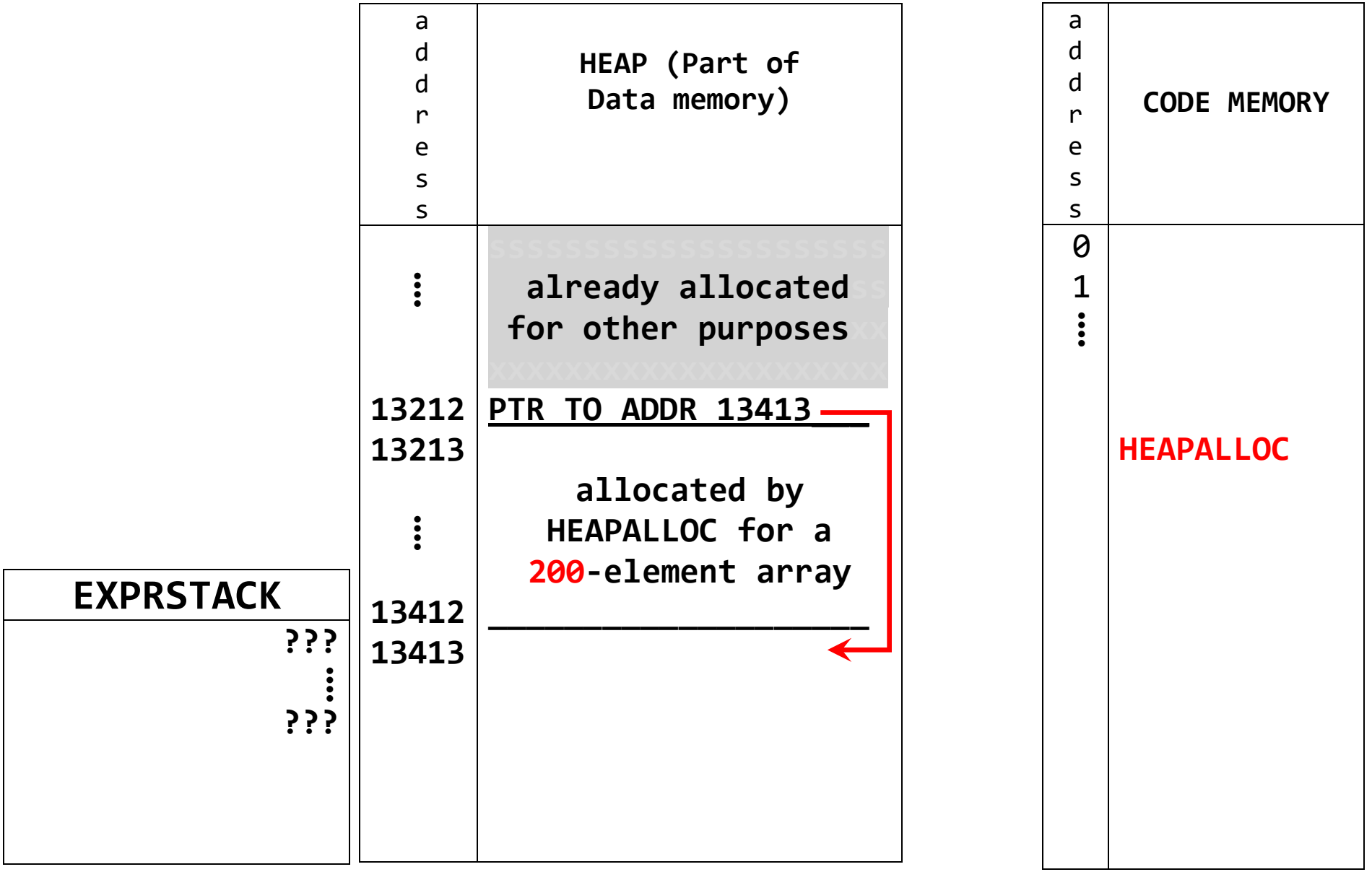
address	HEAP (Part of Data Memory)

address	CODE MEMORY
0	
1	
⋮	
	DISCARDVALUE

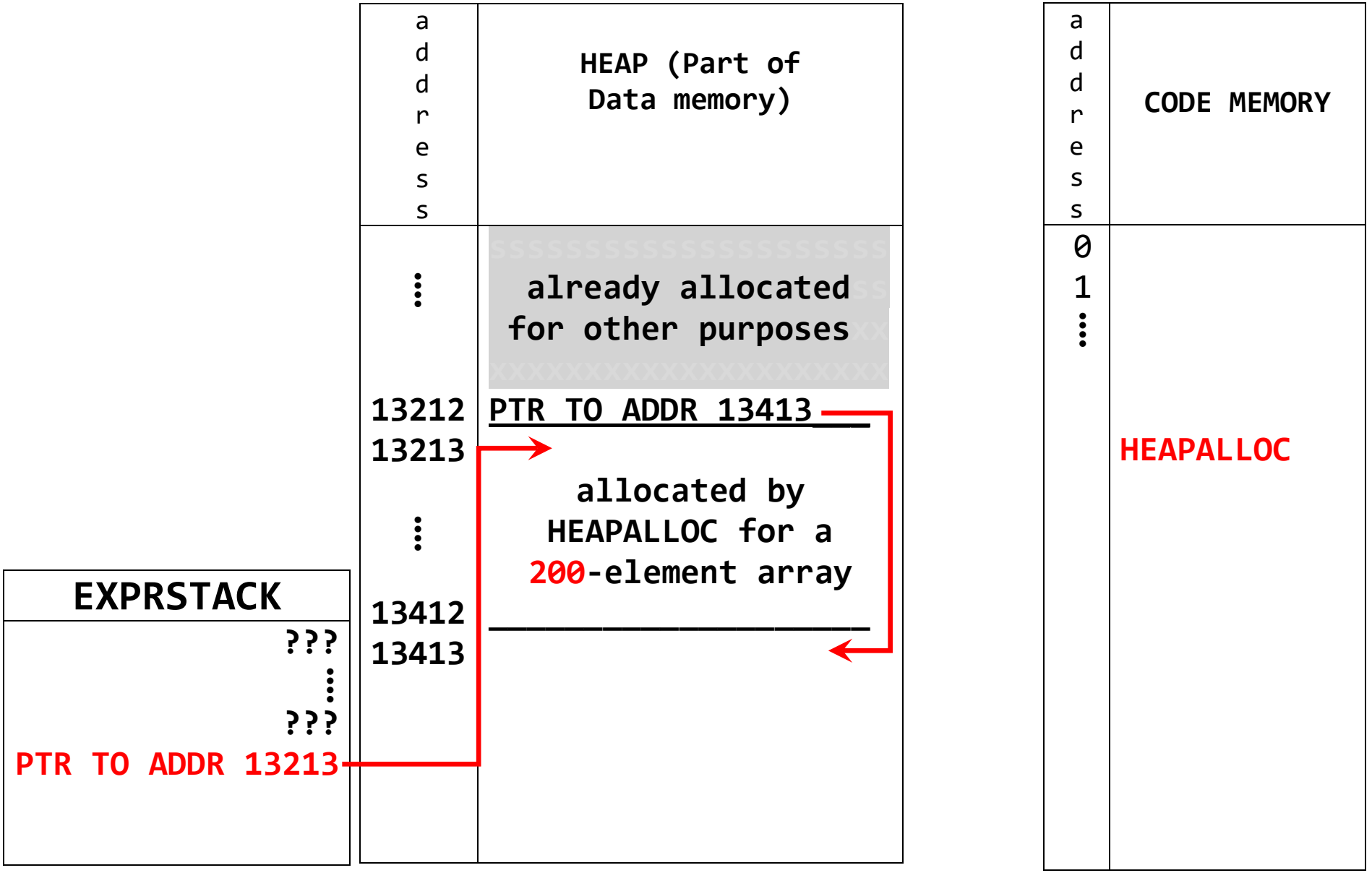
BEFORE execution of: HEAPALLOC



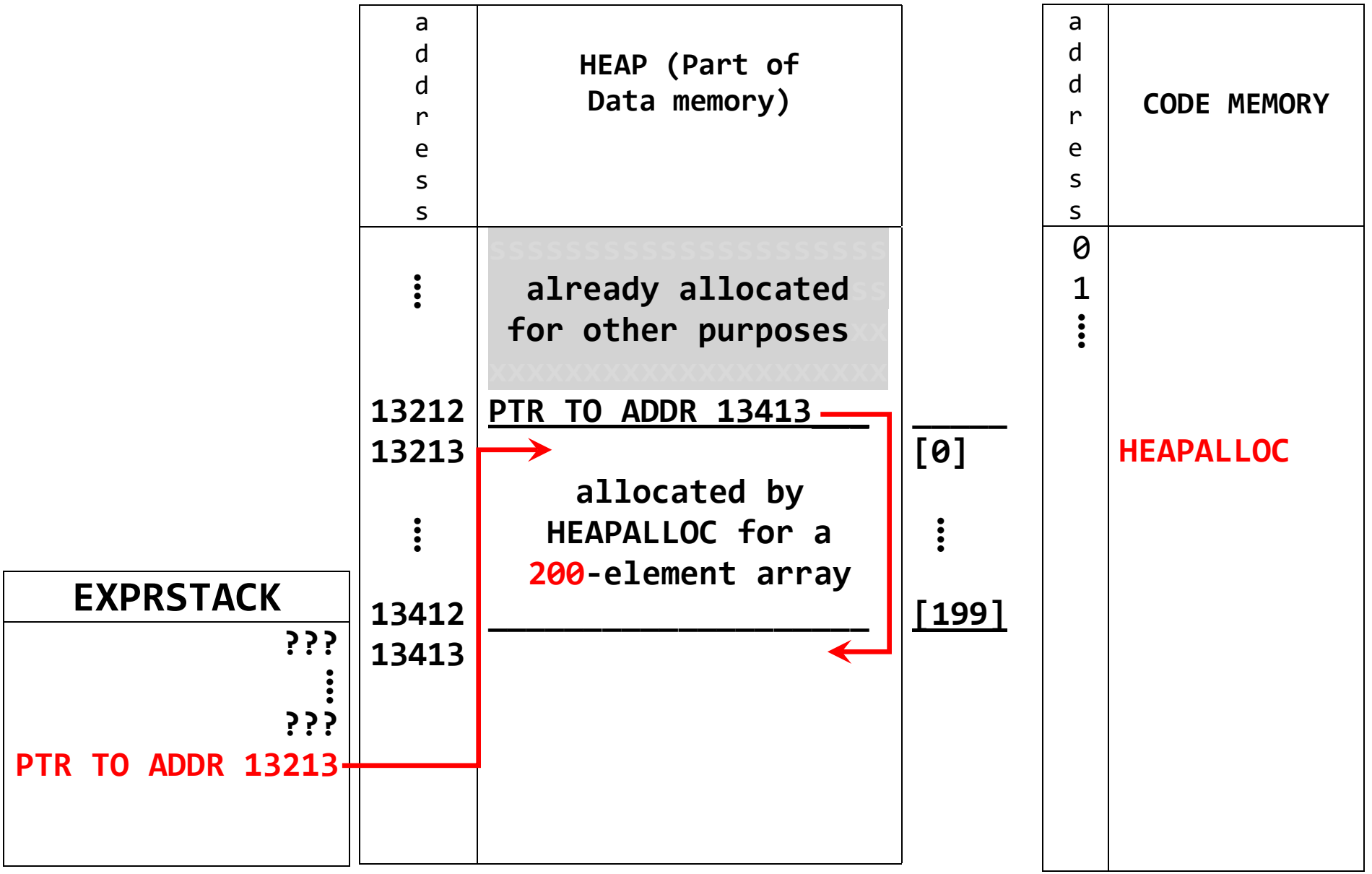
DURING *execution of*: **HEAPALLOC**



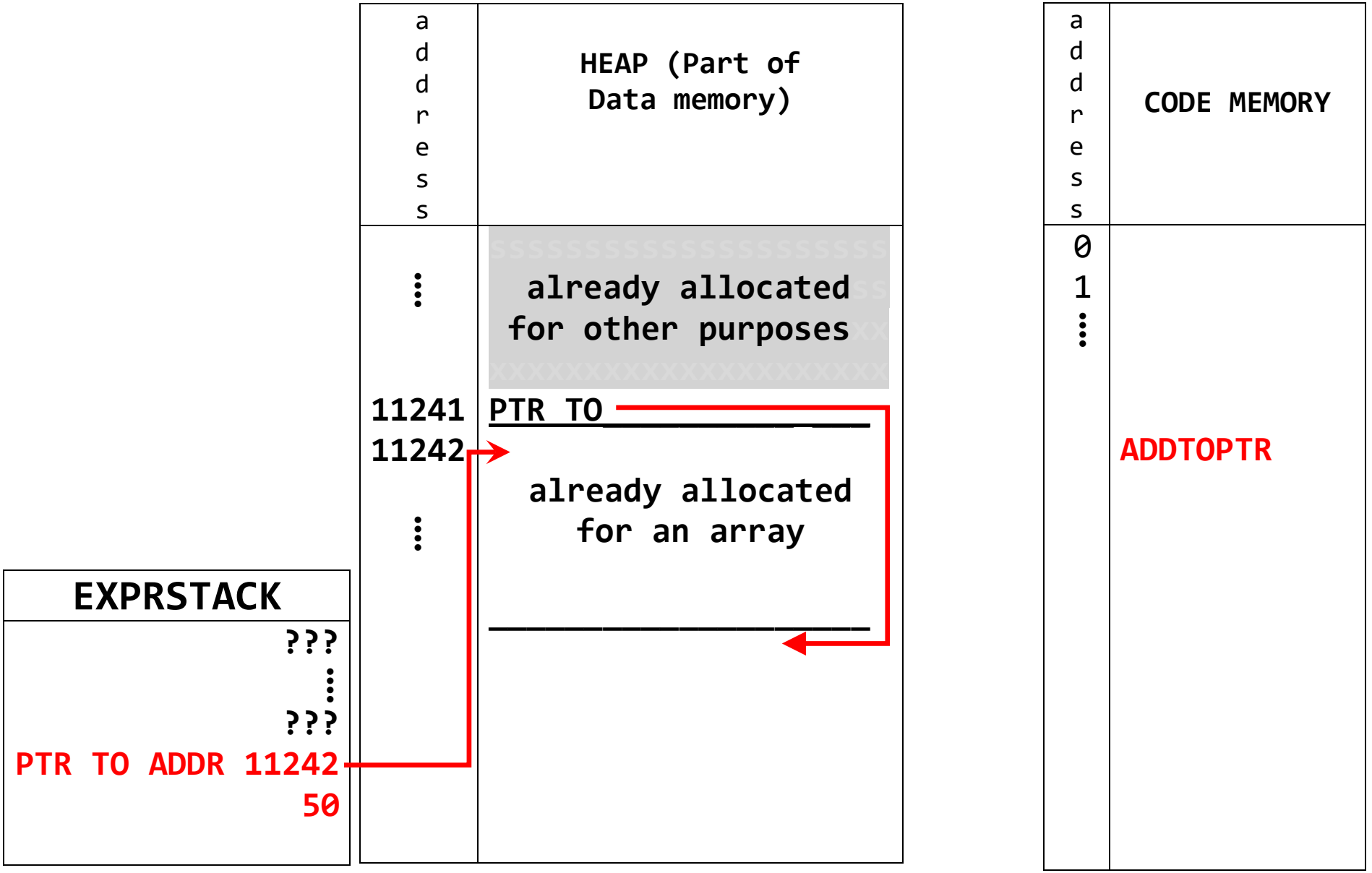
AFTER execution of: HEAPALLOC



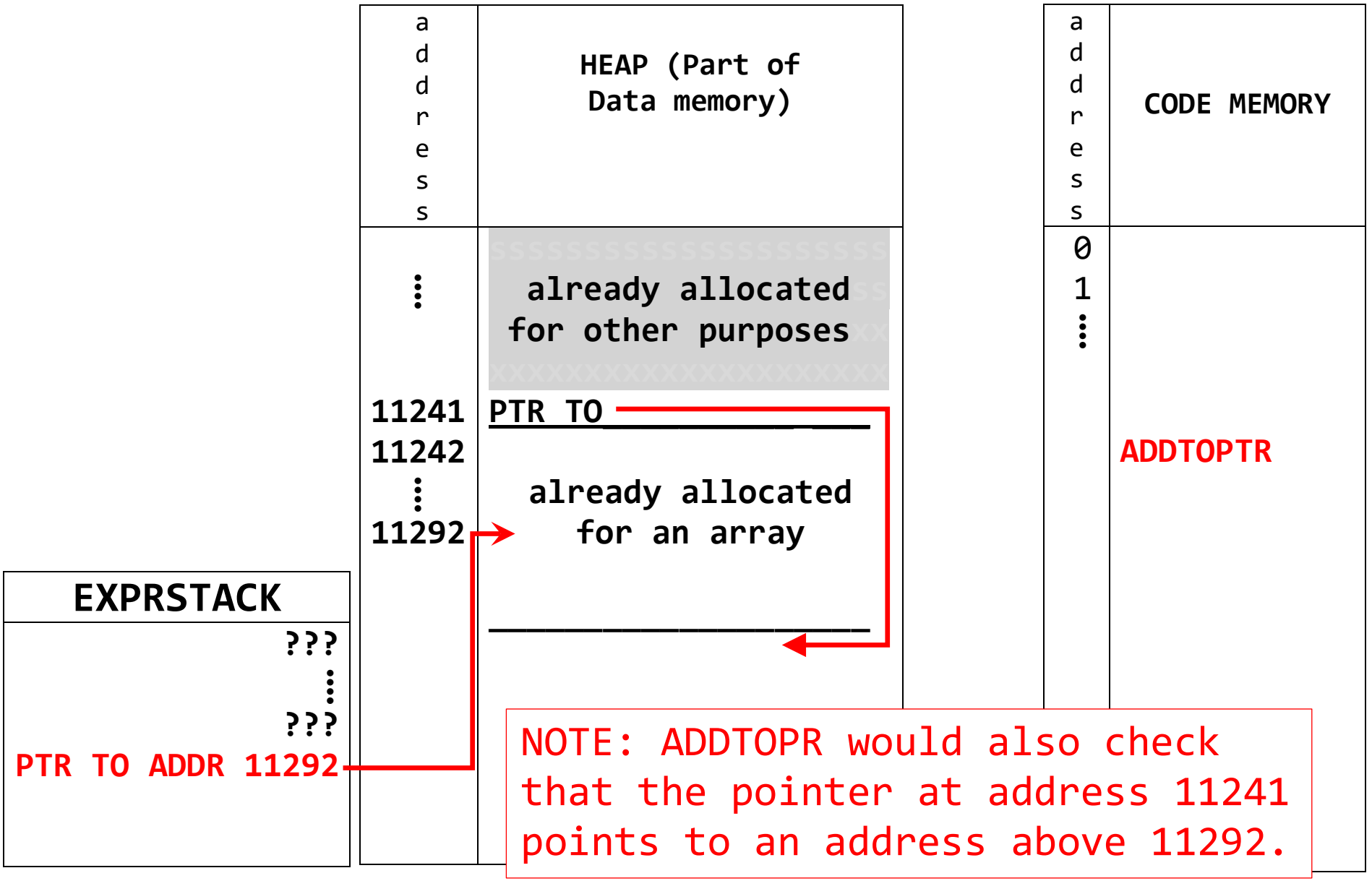
AFTER *execution of*: HEAPALLOC



BEFORE execution of: ADDTOPTR



AFTER execution of: ADDTOPTR



BEFORE *execution of*: **JUMP 87**

PC | 34

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

address	CODE MEMORY
0	
1	
⋮	
33	JUMP 87
⋮	
87	

AFTER *execution of*: **JUMP 87**

PC | 87

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

address	CODE MEMORY
0	
1	
⋮	
33	JUMP 87
⋮	
87	

BEFORE execution of: **JUMPONFALSE 77** (Example 1)

PC | 52

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

EXPRSTACK

???
⋮
???
1

address	CODE MEMORY
0	
1	
⋮	
51	JUMPONFALSE 77
⋮	
77	

AFTER *execution of*: **JUMPONFALSE 77** (Example 1)

PC | 52

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

EXPRSTACK
???
⋮
???

address	CODE MEMORY
0	
1	
⋮	
51	JUMPONFALSE 77
⋮	
77	

BEFORE execution of: **JUMPONFALSE 77** (Example 2)

PC | 52

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

EXPRSTACK
???
⋮
???
0

address	CODE MEMORY
0	
1	
⋮	
51	**JUMPONFALSE 77**
⋮	
77	

AFTER *execution of*: **JUMPONFALSE 77** (Example 2)

PC **77**

NOTE: The PC register is the program counter; it is used to store the code memory address of the next instruction to be executed.

EXPRSTACK

???
⋮
???

address	CODE MEMORY
0	
1	
⋮	
51	**JUMPONFALSE 77**
⋮	
77	