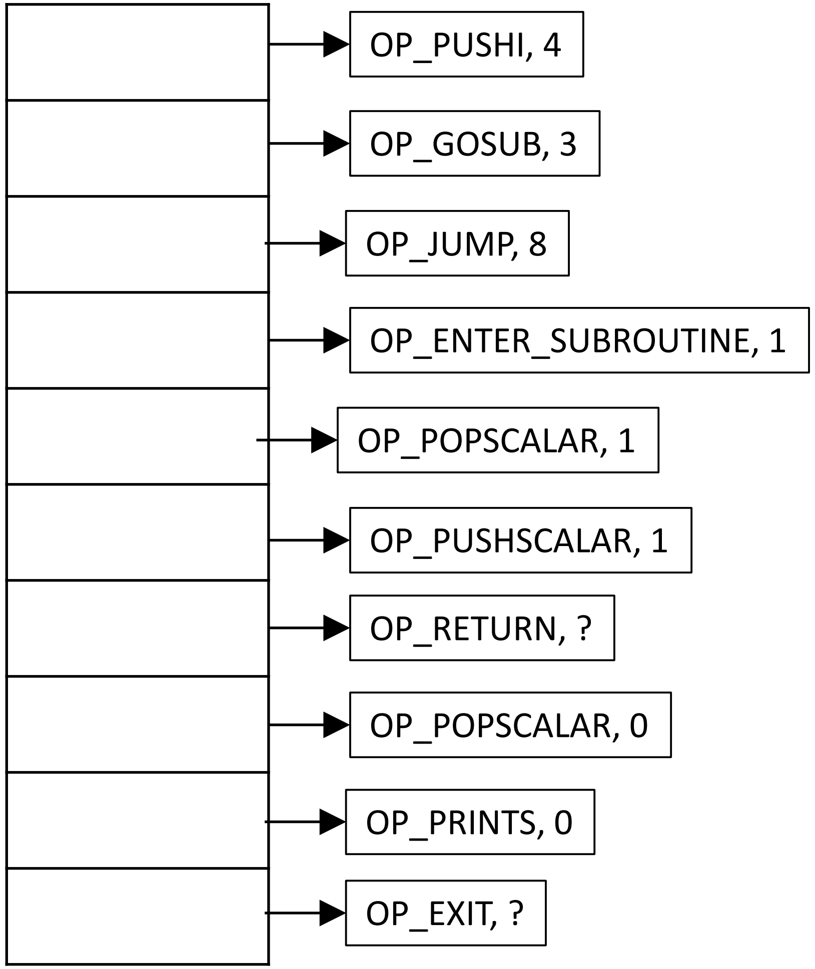
```
start
declscal A
pushi 4
gosub L1
jump L2
gosublabel L1
declscal A
popscal A
pushscal A
return
label L2
popscal A
prints exit_pgm
exit 0
end
```

This is the program at the end of the parse, and the statements that have been written to the output file to be executed by the VM

instruction buffer:



OP_PUSHI, 4

OP_GOSUB, 3

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

OP_POPSCALAR, 0

OP_PRINTS, 0

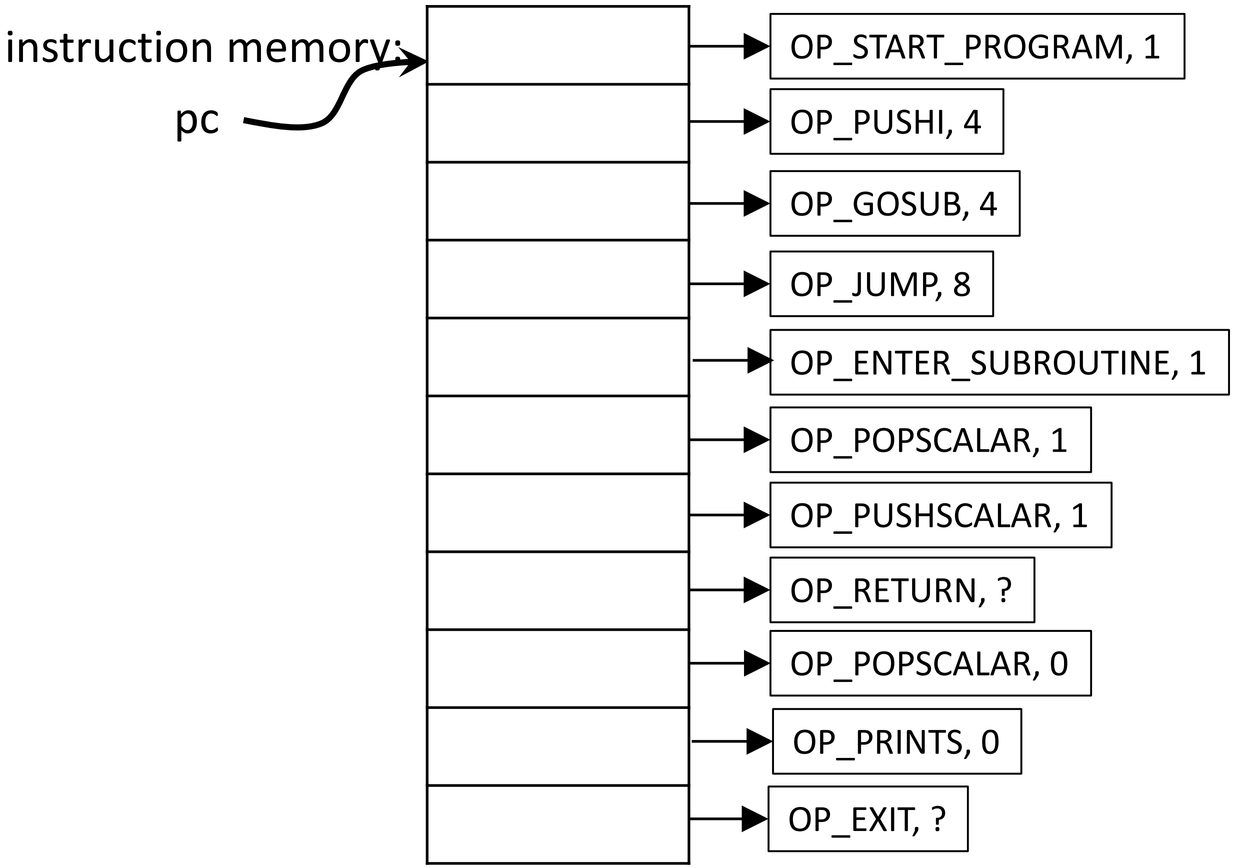
OP_EXIT, ?

String buffer **exit_pgm**

runtime stack

These are the data structures after reading in the program

data memory



instruction memory:

pc

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

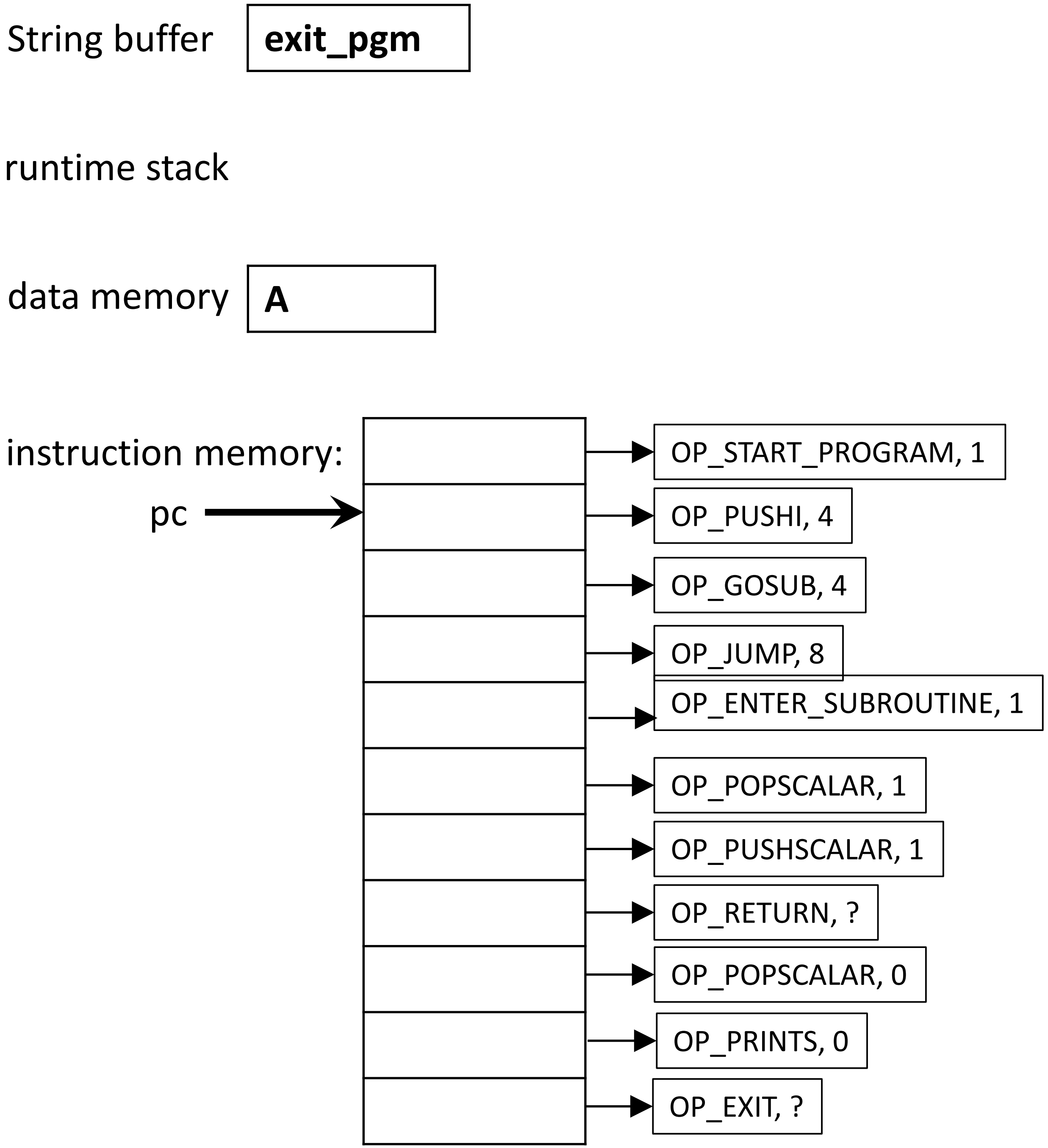
OP_PUSHSCALAR, 1

OP_RETURN, ?

OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?



String buffer **exit_pgm**

runtime stack

data memory **A**

instruction memory:

pc

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

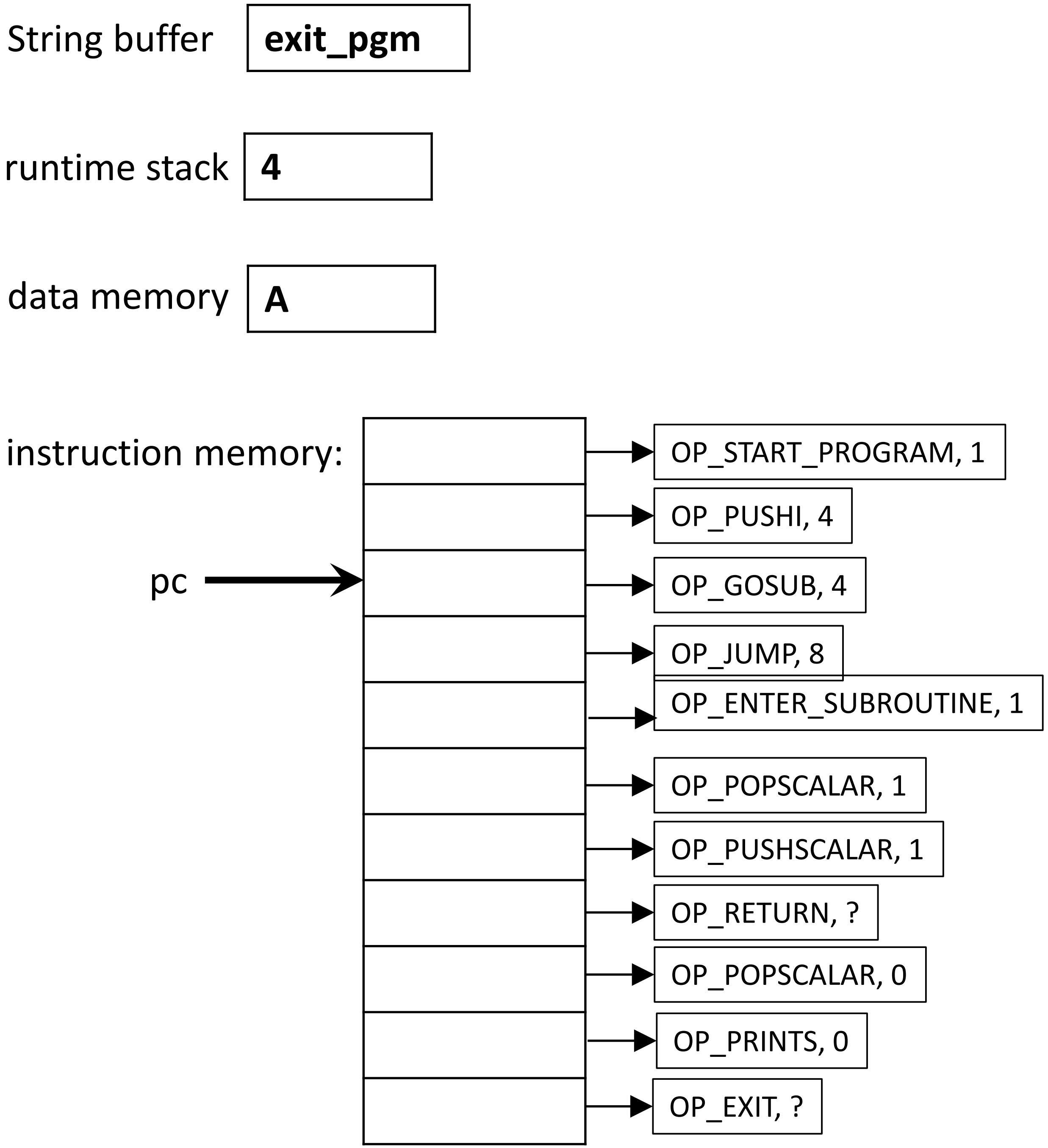
OP_RETURN, ?

OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

- After executing *OP_START_PROGRAM 1* the data memory contains a stack frame for the outer scope.
  - The stack frame has one element, to contain the outer scope variable A.
- After executing the OP_START_PROGRAM instruction the pc points to the next statement.



String buffer **exit_pgm**

runtime stack **4**

data memory **A**

instruction memory:

pc

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

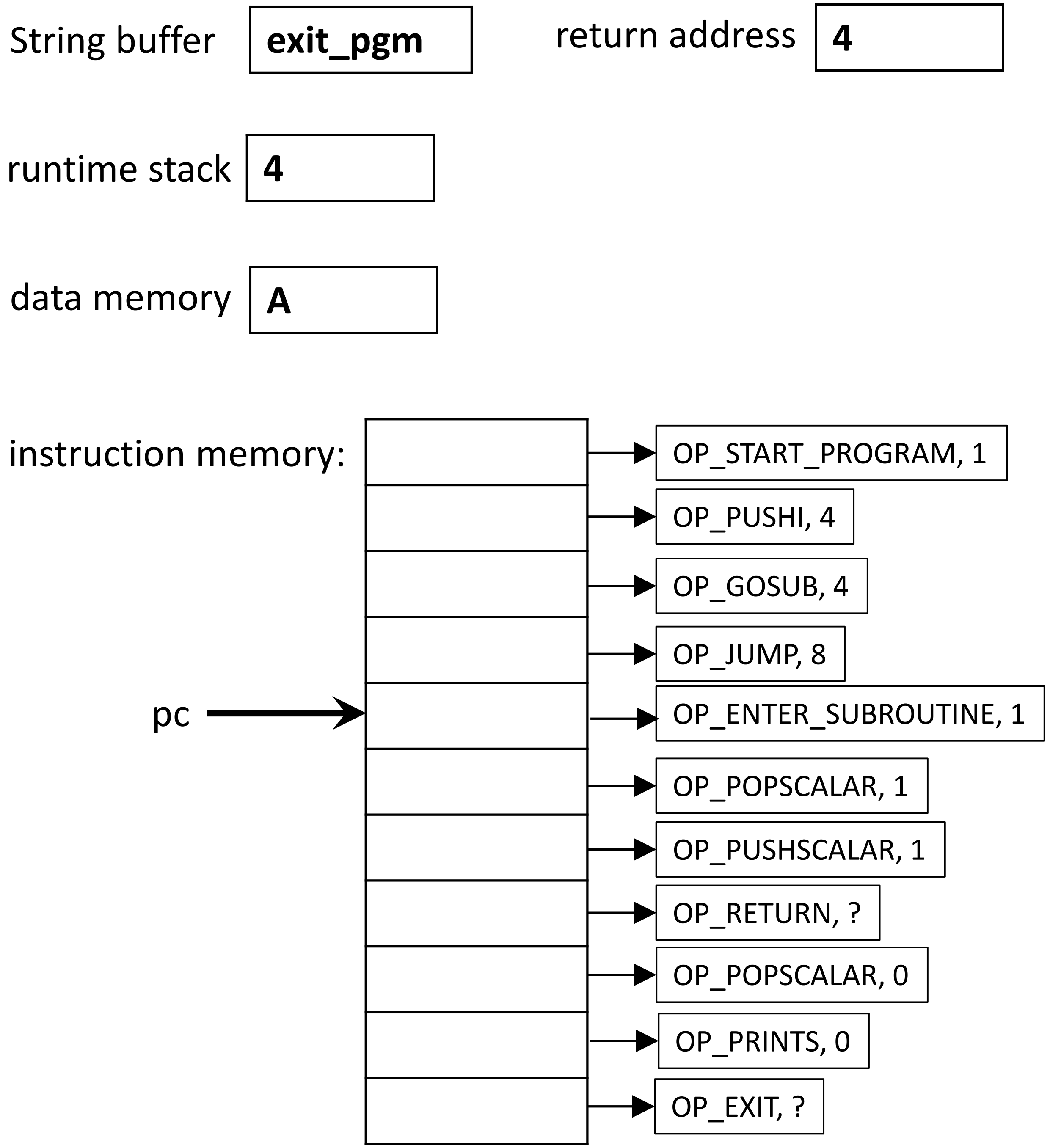
OP_RETURN, ?

OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

- After executing *OP_PUSHI 4* instruction
    - the runtime stack contains a 4
    - the pc points to the next statement, the GOSUB statement.



String buffer **exit_pgm**

return address **4**

runtime stack **4**

data memory **A**

instruction memory:

pc

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

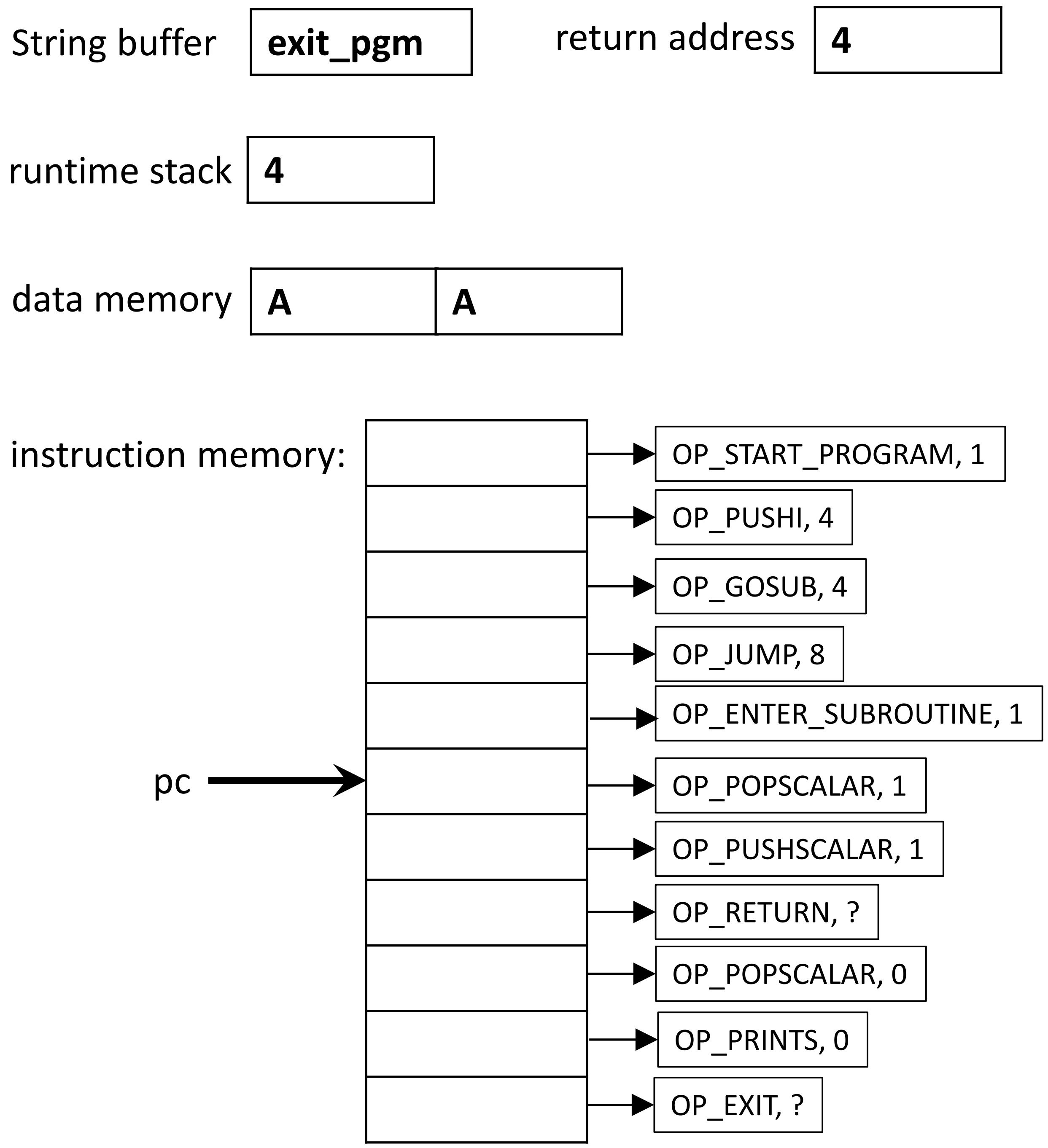
OP_RETURN, ?

OP_POPSCALAR, 0

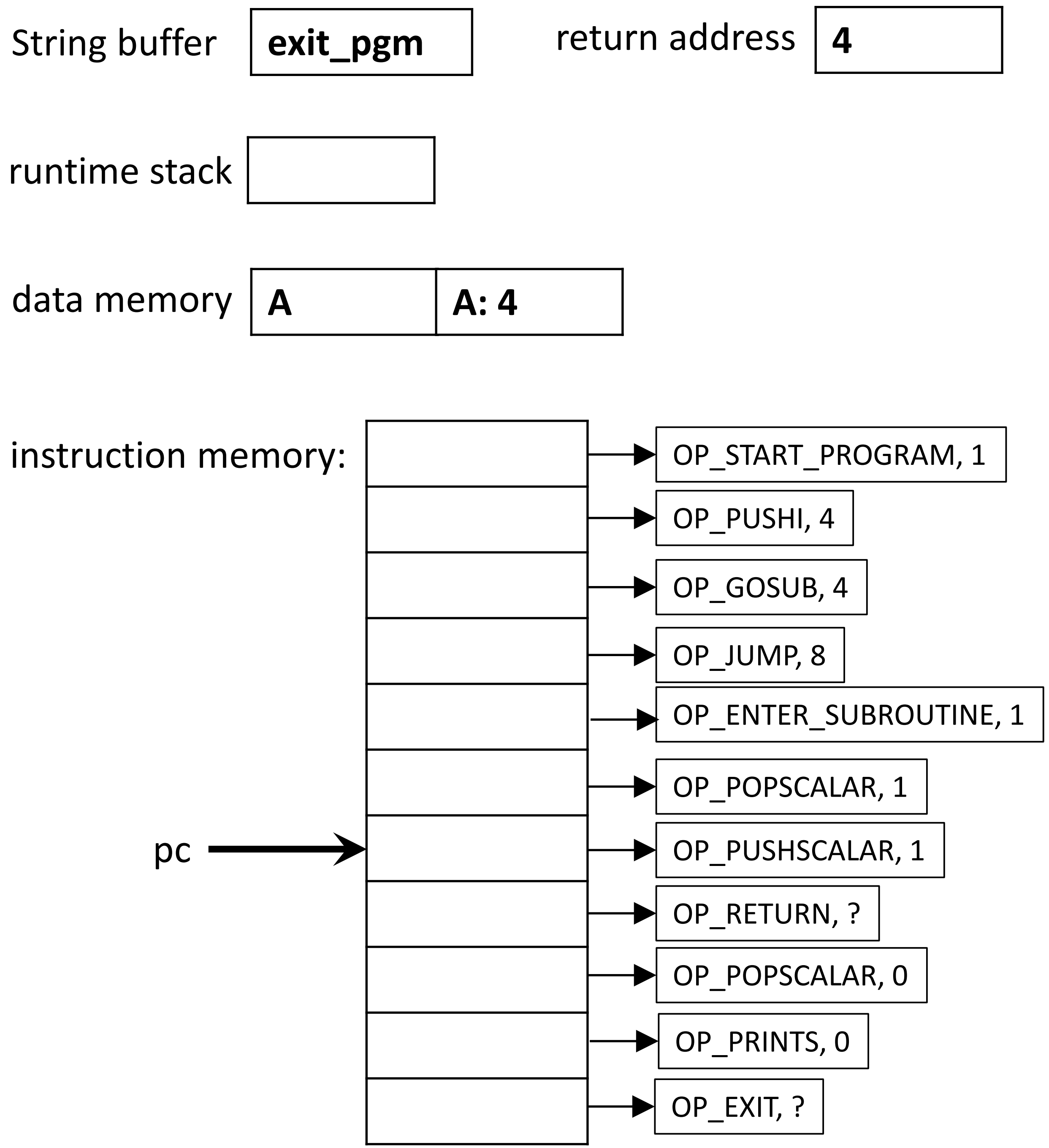
OP_PRINTS, 0

OP_EXIT, ?

- After executing OP_GOSUB statement the pc points to the start of the subroutine.
- The return address from the subroutine being entered should be the statement after the GOSUB, i.e., the OP_JUMP.
  - Since we allow recursive programs, return addresses should be kept in a return address stack
  - The top address will be the return address for the currently executing subroutine.
- After executing the GOSUB the pc points to the start of the subroutine.



- After executing the OP_ENTER_SUBROUTINE instruction
  - A new stack frame for the subroutine is created in the data memory to hold the A declared in the subroutine
  - The pc points to the OP_POPSCALAR instruction.



String buffer **exit_pgm**

return address **4**

runtime stack

data memory **A** **A: 4**

instruction memory:

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

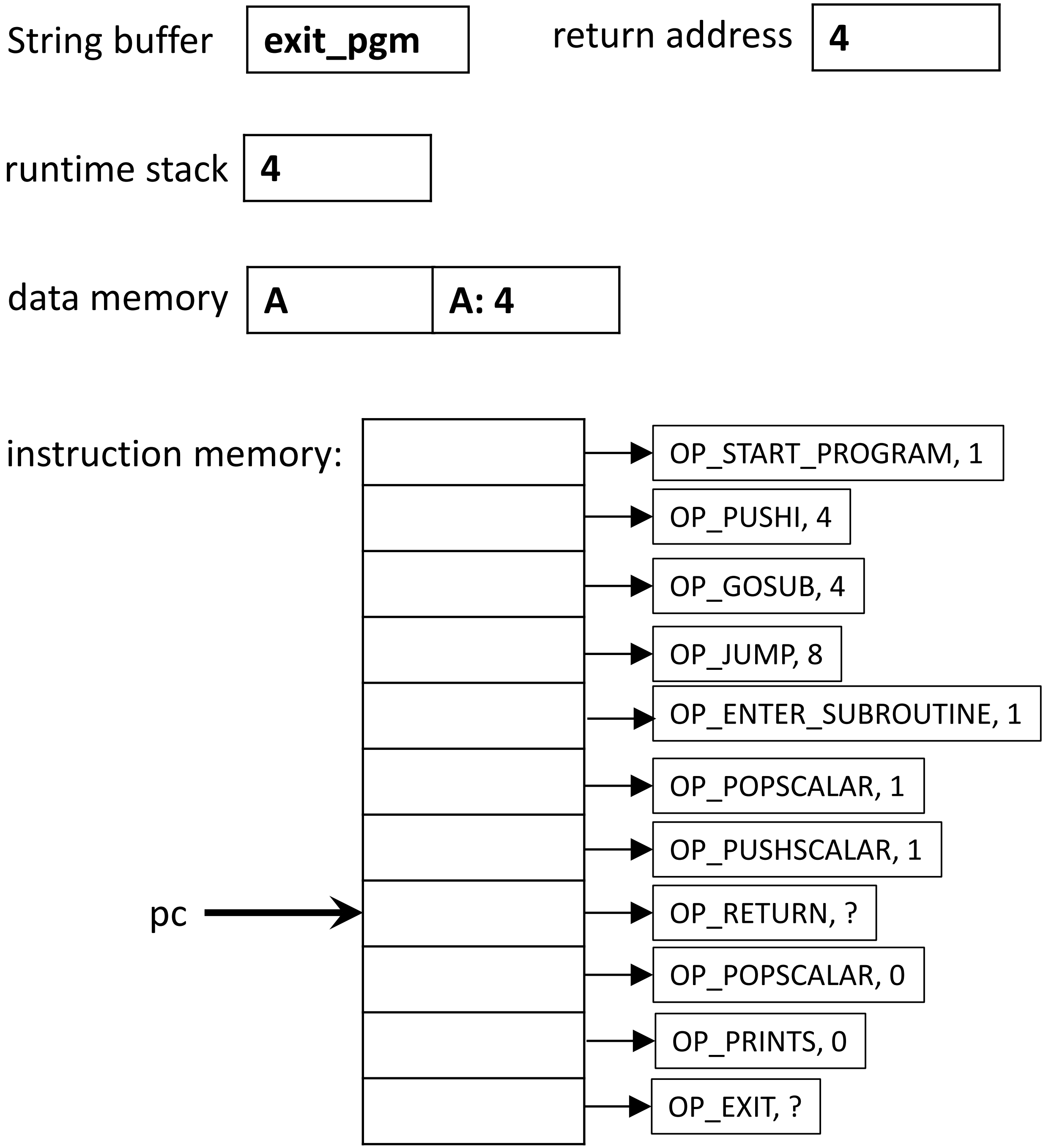
OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

pc

- After executing the OP_POPSCALAR
  - the value at the top of the runtime stack is placed in the subroutine’s A, the variable at position 1 in the data memory
  - The runtime stack is now empty
  - The pc points to the OP_PUSHSCALAR instruction



String buffer: **exit_pgm**

return address: **4**

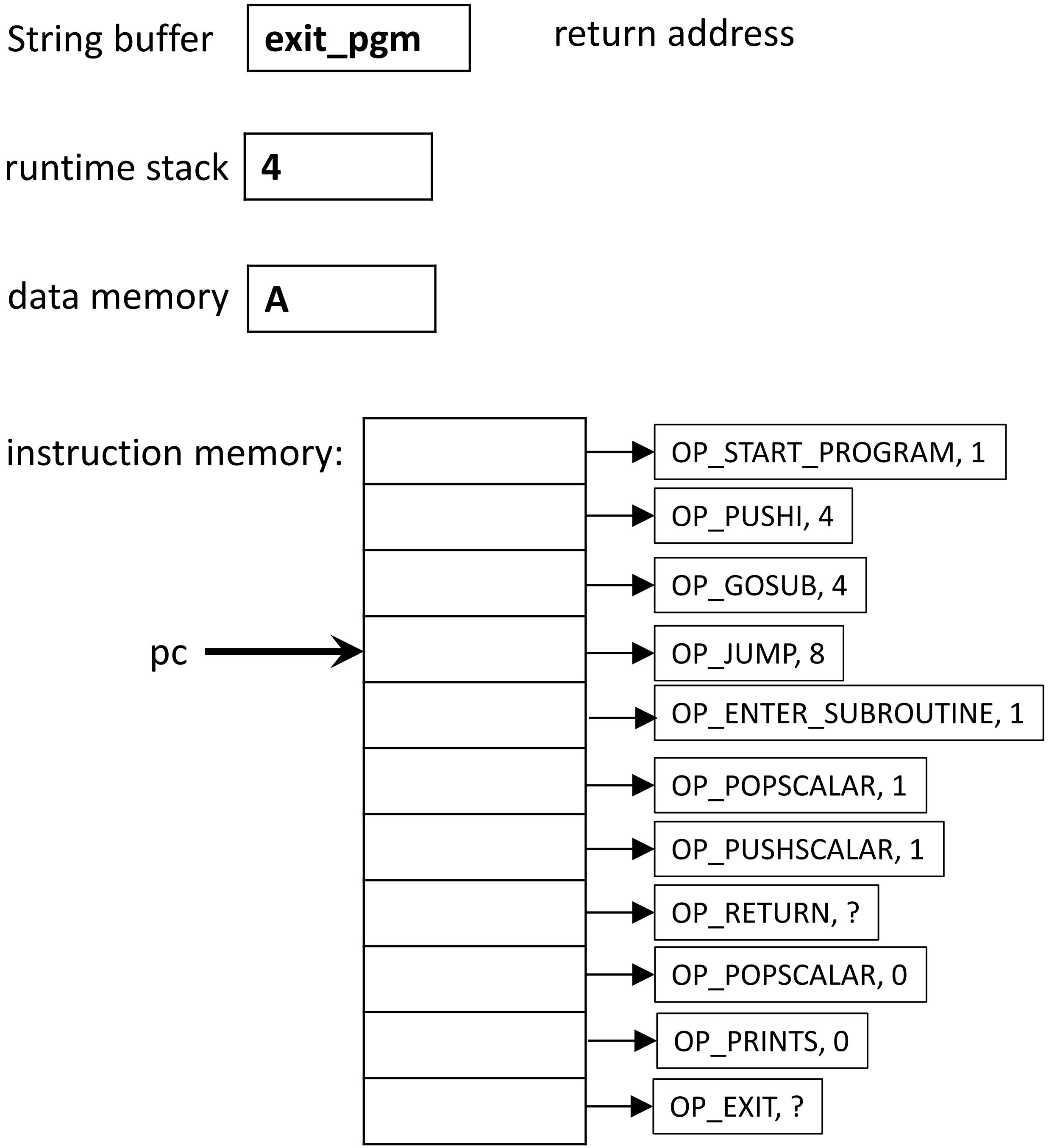
runtime stack: **4**

data memory: **A** | **A: 4**

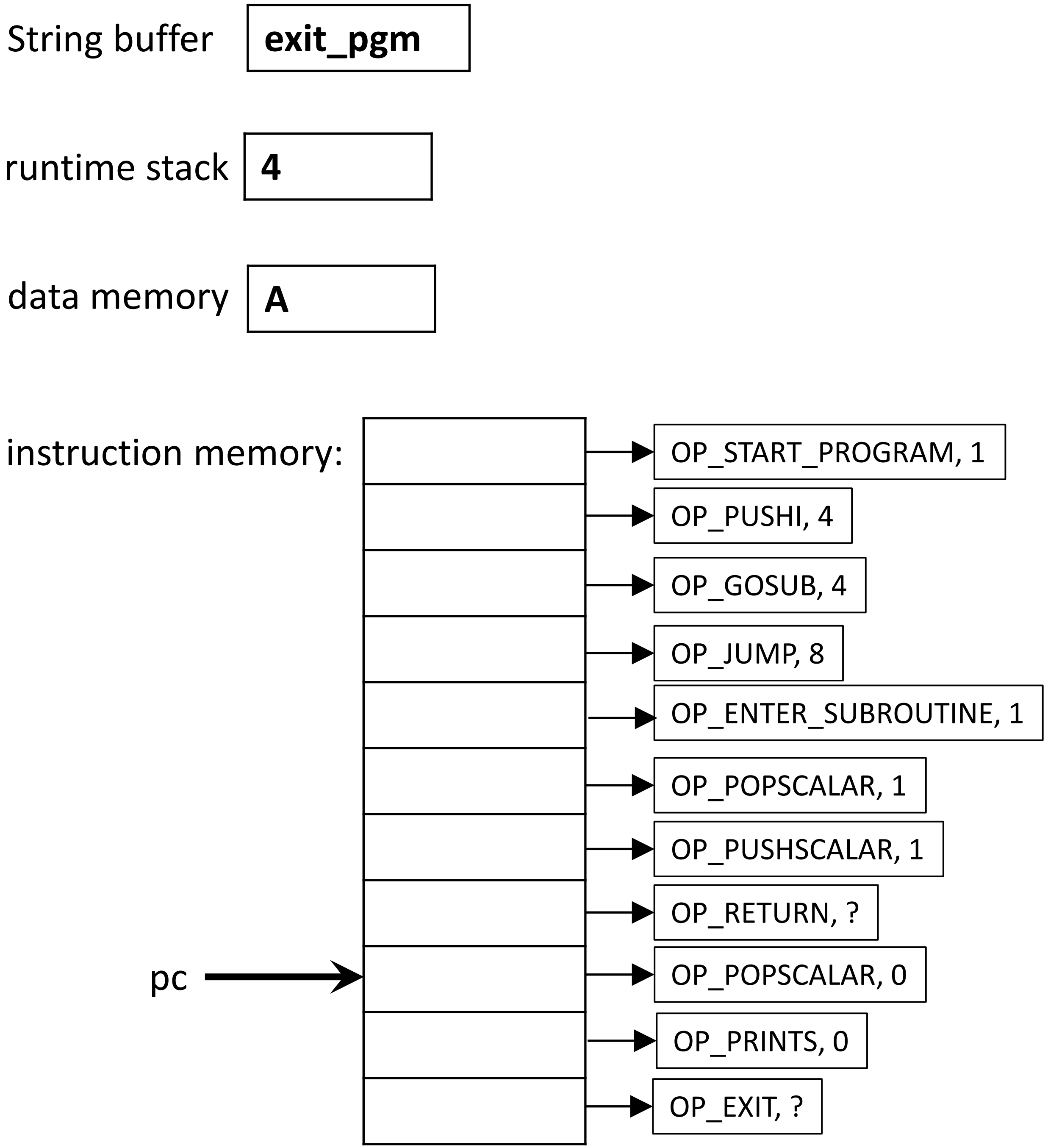
instruction memory:

- OP_START_PROGRAM, 1
- OP_PUSHI, 4
- OP_GOSUB, 4
- OP_JUMP, 8
- OP_ENTER_SUBROUTINE, 1
- OP_POPSCALAR, 1
- OP_PUSHSCALAR, 1
- OP_RETURN, ?
- OP_POPSCALAR, 0
- OP_PRINTS, 0
- OP_EXIT, ?

- After executing the OP_PUSHSCALAR
  - the value in the variable at position 1 of the data memory (the subroutine’s A) is pushed onto the stack.
  - The pc points to the OP_RETURN instruction



- The OP_RETURN statement
  - Sets the PC to the return address (position 4 in the instruction memory)
  - Pops the return address of the return address stack
  - Pops the stack frame for the subroutine from the data memory.



String buffer **exit_pgm**

runtime stack **4**

data memory **A**

instruction memory:

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

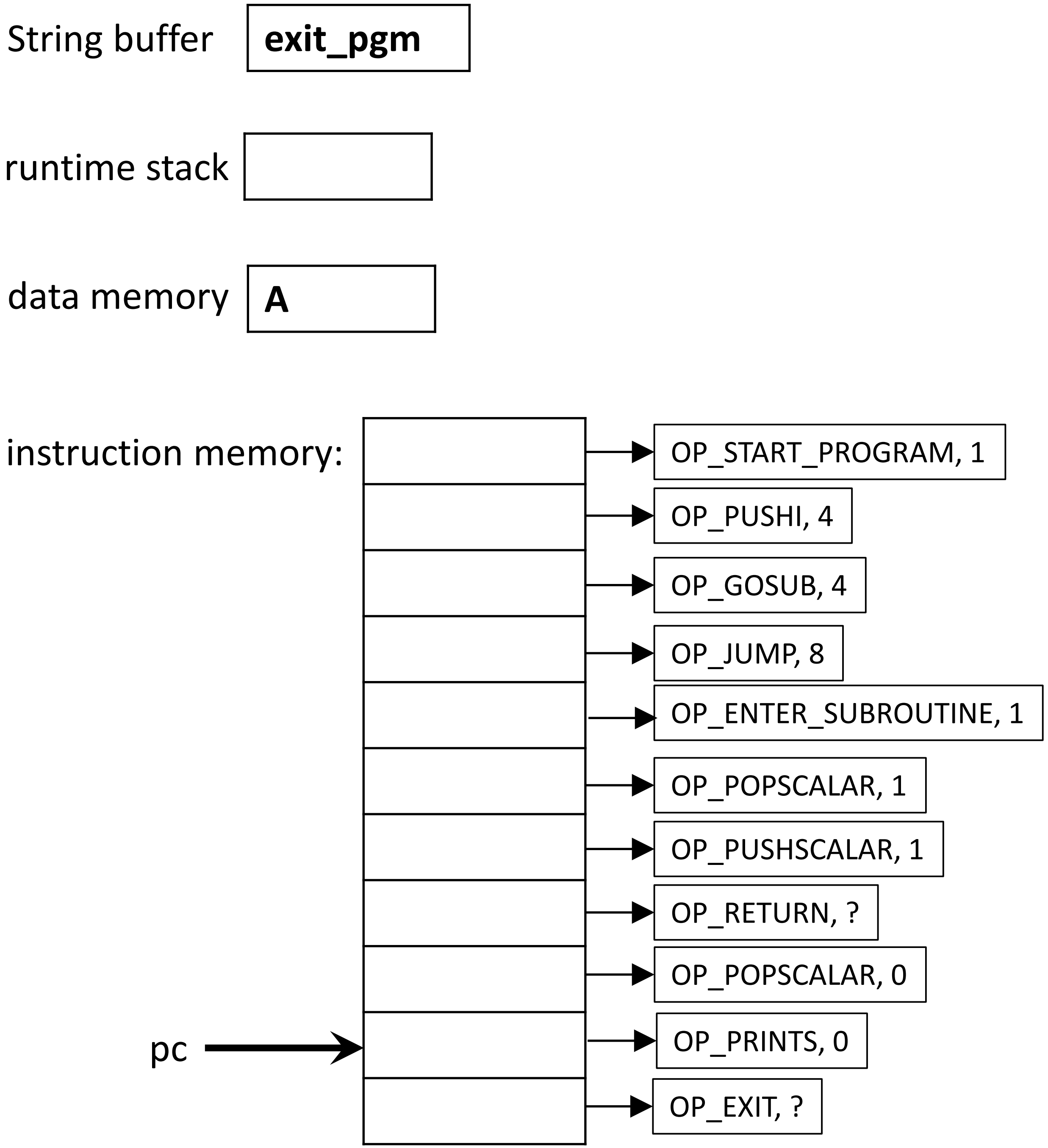
OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

pc

- The OP_JUMP statement sets the PC to the value of its operand
- This causes control to pass to the OP_POPSCALAR, 0 instruction in position 8 of the instruction memory.



String buffer **exit_pgm**

runtime stack

data memory **A**

instruction memory:

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

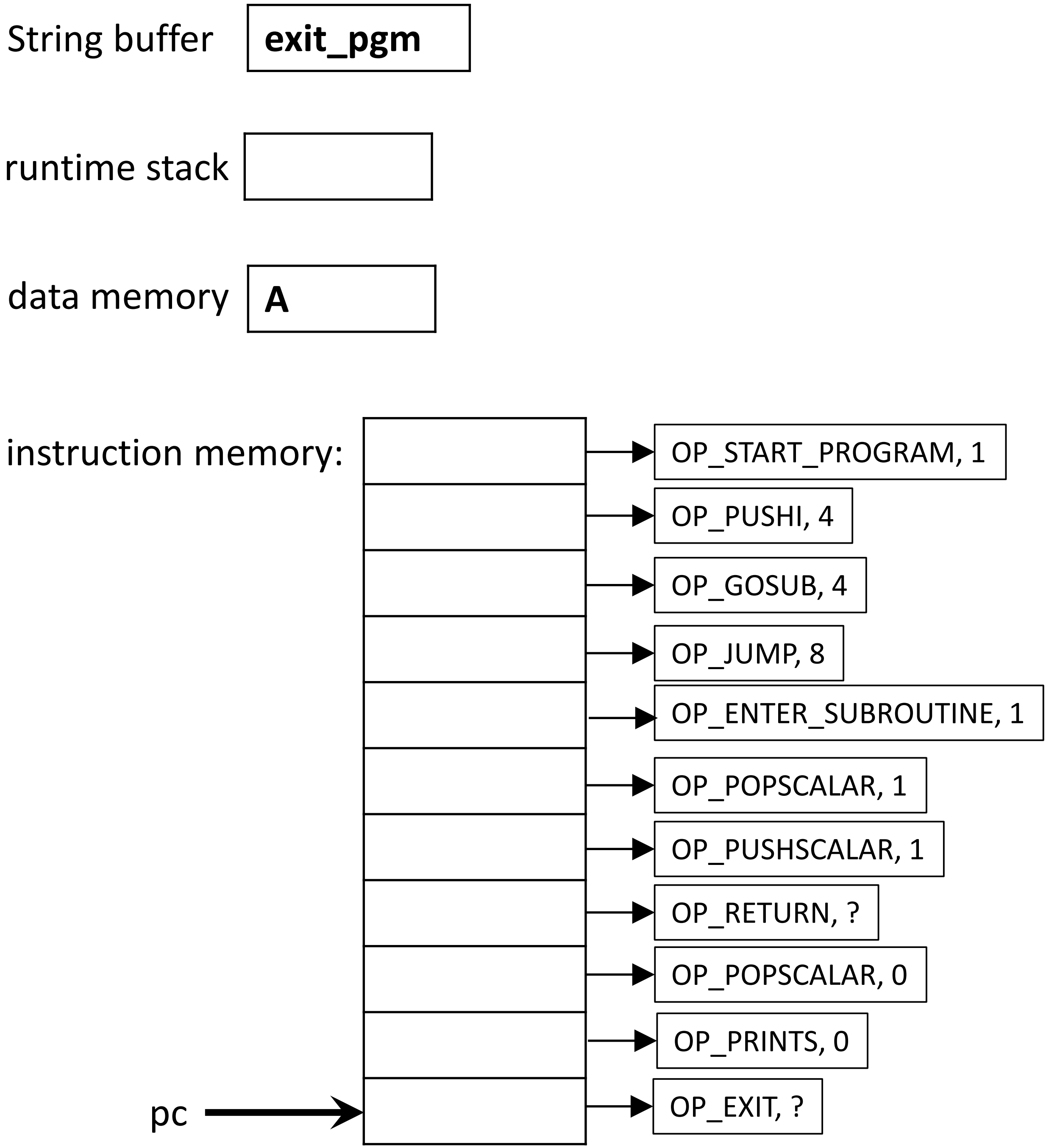
OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

pc

- The OP_POPSCALAR
    - places the value at the top of the stack into data memory location 0 (as specified by its operand).
    - That value is removed from the stack
    - The pc is set to the address of the next instruction.



String buffer **exit_pgm**

runtime stack

data memory **A**

instruction memory:

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

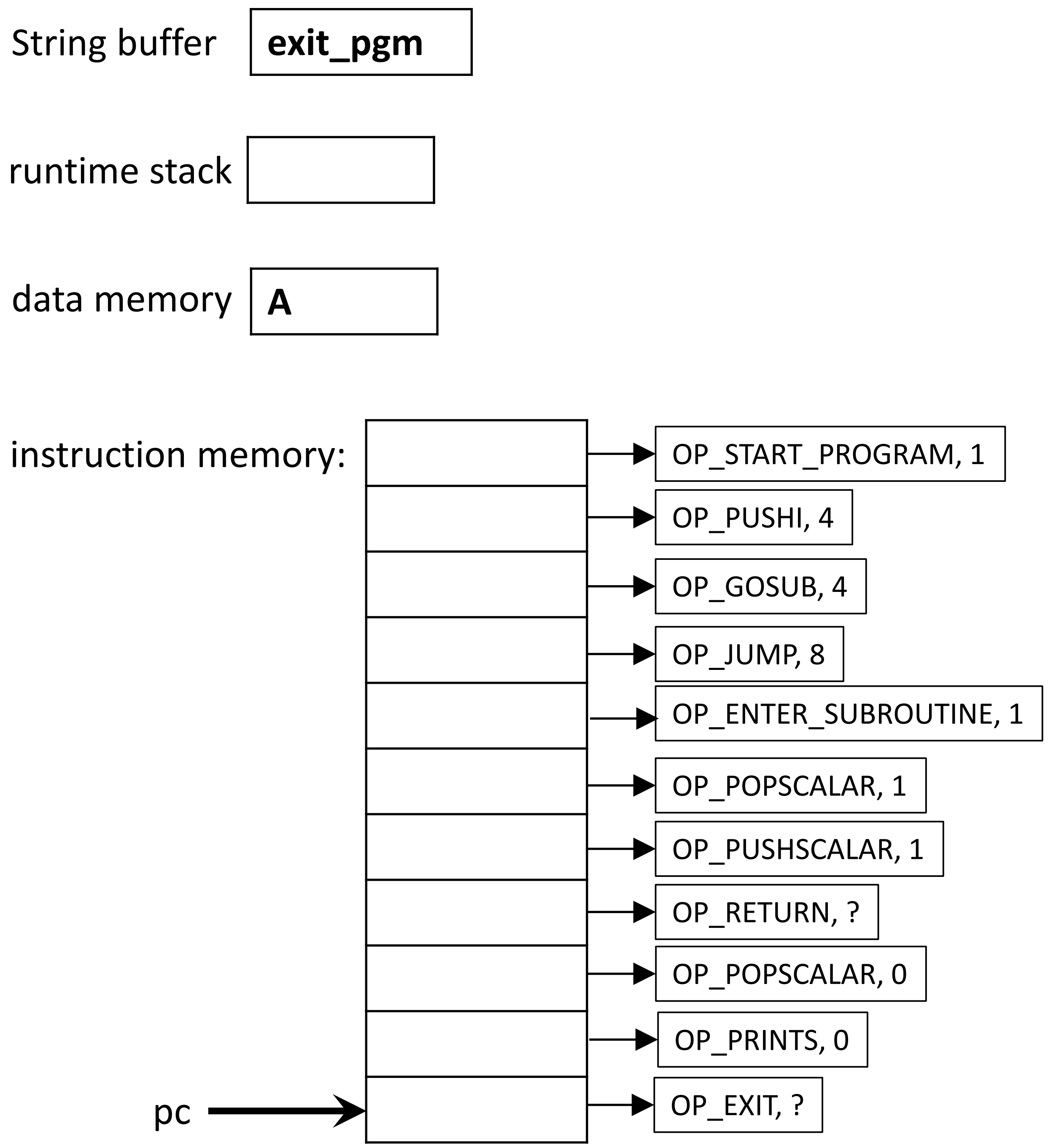
OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

pc

- The OP_PRINTS statement
  - prints the string at position 0 of the string buffer, as specified by the OP_PRINTS operand.
  - The string *exit_pgm* is printed
  - The pc is incremented.



String buffer **exit_pgm**

runtime stack

data memory **A**

instruction memory:

OP_START_PROGRAM, 1

OP_PUSHI, 4

OP_GOSUB, 4

OP_JUMP, 8

OP_ENTER_SUBROUTINE, 1

OP_POPSCALAR, 1

OP_PUSHSCALAR, 1

OP_RETURN, ?

OP_POPSCALAR, 0

OP_PRINTS, 0

OP_EXIT, ?

pc

The OP_EXIT exits the program.

I print “terminated normally” or something similar in my version just so I know that it finished executing ok. This is not required.