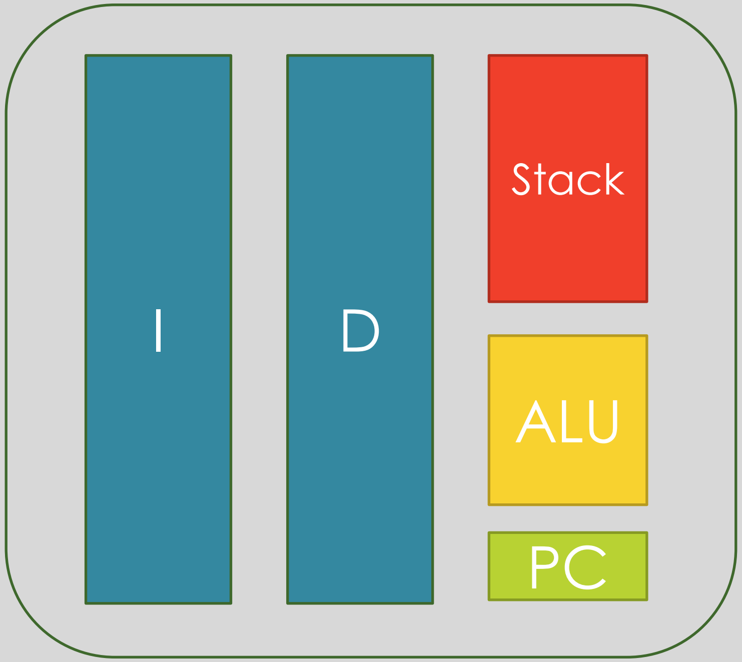
ABSTRACT STACK MACHINE

CMPT 379 Lecture 6a

Lecture Overview

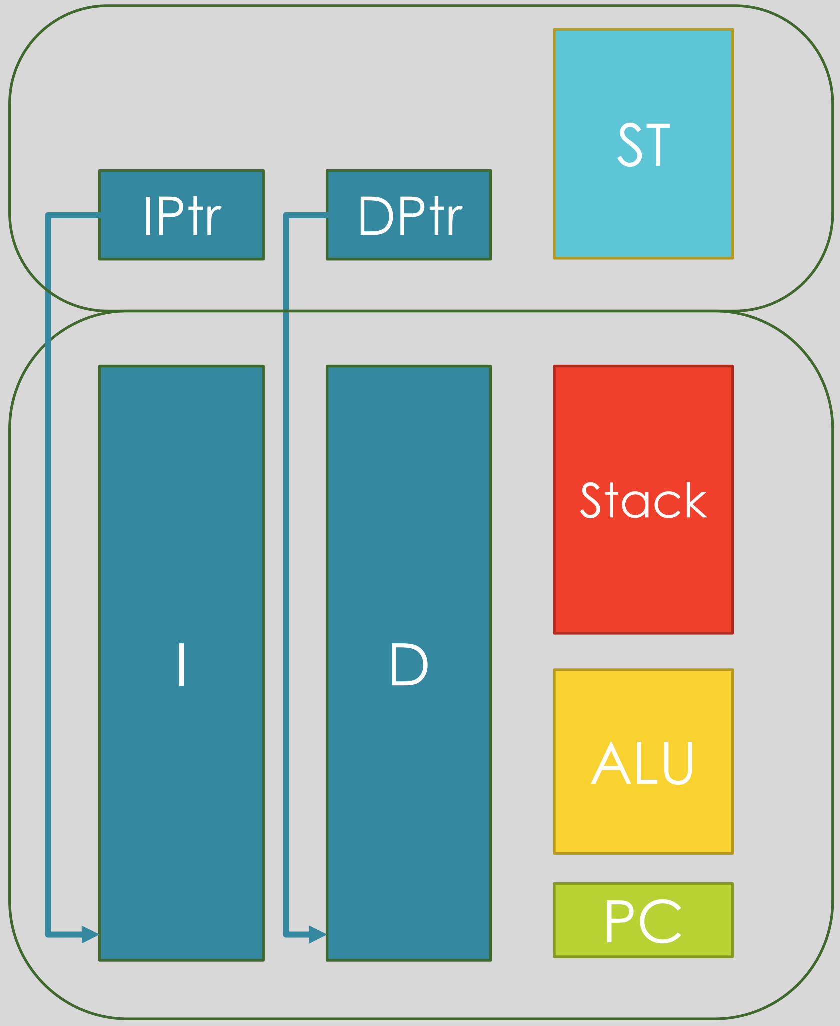
- Abstract Stack Machine
- ASM Loader
- Load and Execute example
- Commenting Guidelines
- ASMOpcode
- ASM Code examples

The Abstract Stack Machine



- **I:** Array of memory for instructions [0..N] instructions
- **D:** Array of memory for data [0..Memtop-1] bytes
- **Stack:** Array of memory accessed as stack. Takes the place of registers. ("Accumulator Stack")
- **ALU:** Arithmetic-Logic Unit. The processor.
- **PC:** Program Counter. Contains address of the next instruction to be executed.

The ASM Loader



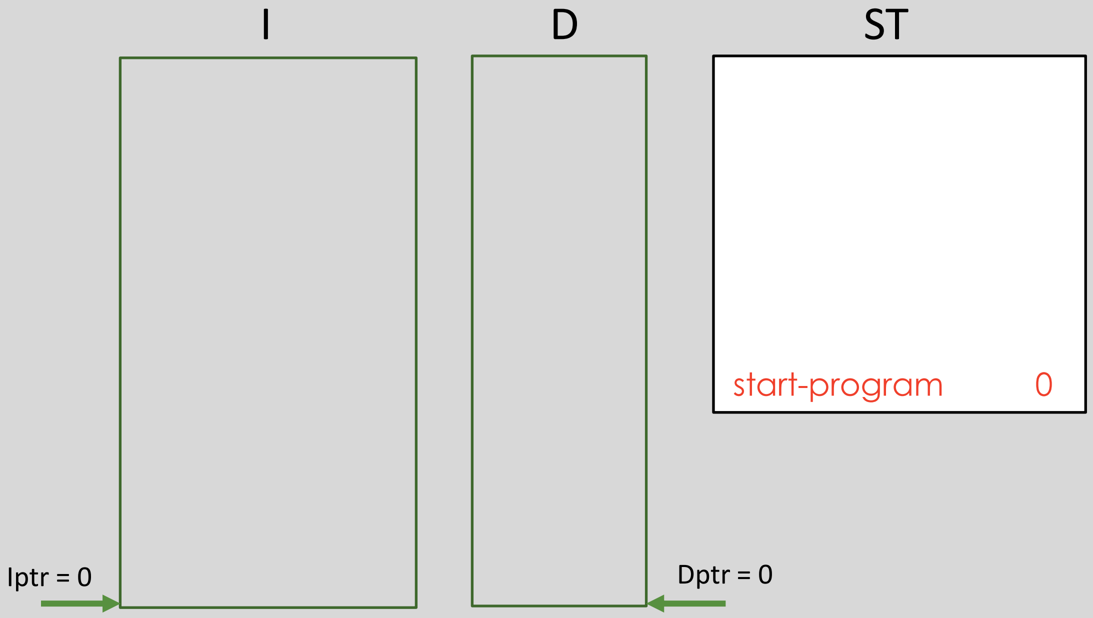
- **IPtr:** Pointer to next instruction to fill
- **DPtr:** Pointer to next data location to fill
- **ST:** Symbol table with (name, location) entries

ASM Loader Operation

- The loader starts by setting an empty symbol table, empty ASM Stack, IPtr = 0, DPtr = 0, and PC = 0.
- Then it reads an input ASM file, one line at a time.
- If the line contains an **instruction**, then it puts that instruction at location IPtr in the instruction store, and increments IPtr.
- If the line contains a **Label <name>** directive, then it installs *name* into the symbol table with the integer IPtr.
- If the line contains a **DLabel <name>** directive, then it installs *name* into the symbol table with the integer DPtr.
- If the line contains any other directive (a data directive), then it fills in the data array starting at DPtr according to the directive. It then increments DPtr by the number of bytes stored.
- Then it goes through the instruction store and the data array looking for symbol operands, and replaces them with the value found in the symbol table for them.
- Finally, it starts the ASM.

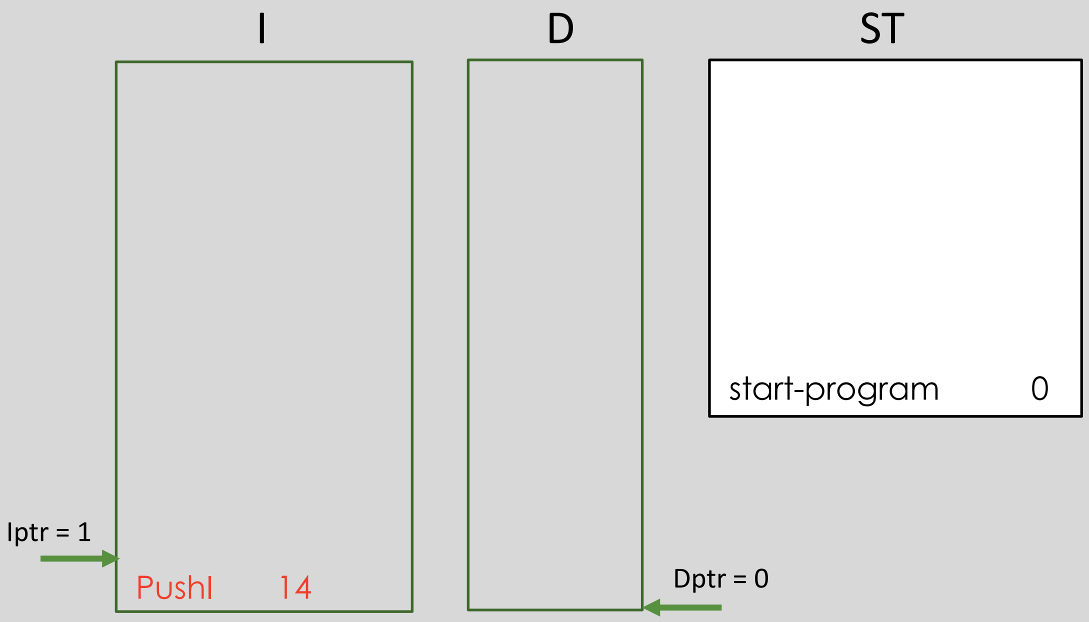
Example

```
Label      start-program
PushI      14
PushI      2
Add
PushD      storage-for-x
Exch
StoreI
DLabel     storage-for-u
DataF      5.1
DLabel     storage-for-x
DataI      7
PushI      0
Label      loop-start-1
DataD      storage-for-u
PushD      storage-for-x
LoadI
Duplicate
PushI      1
Subtract
PushD      storage-for-x
Exch
StoreI
Add
Jump       loop-start-1
```



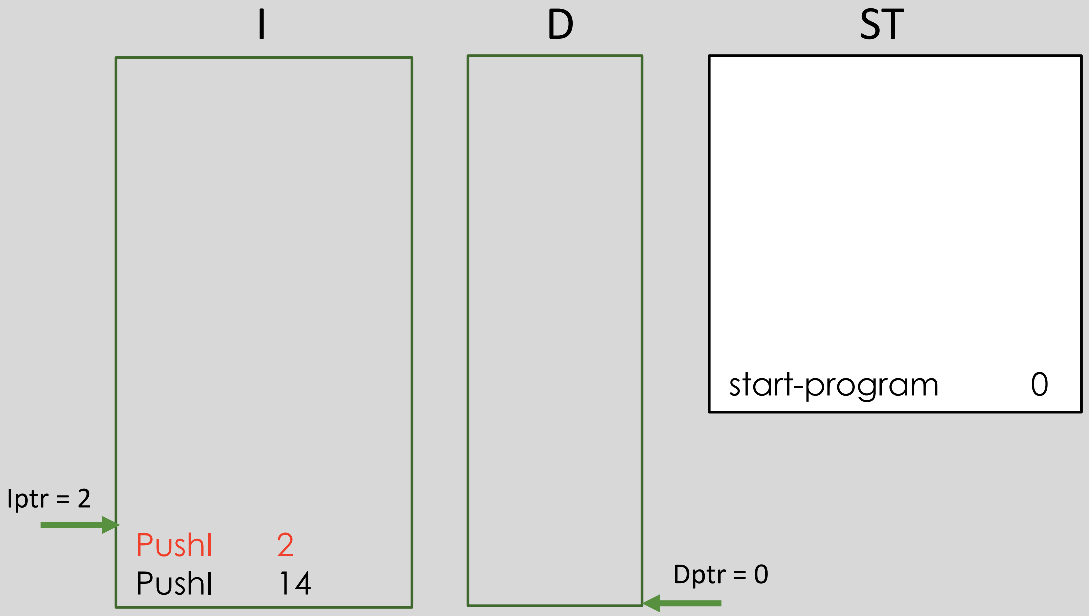
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



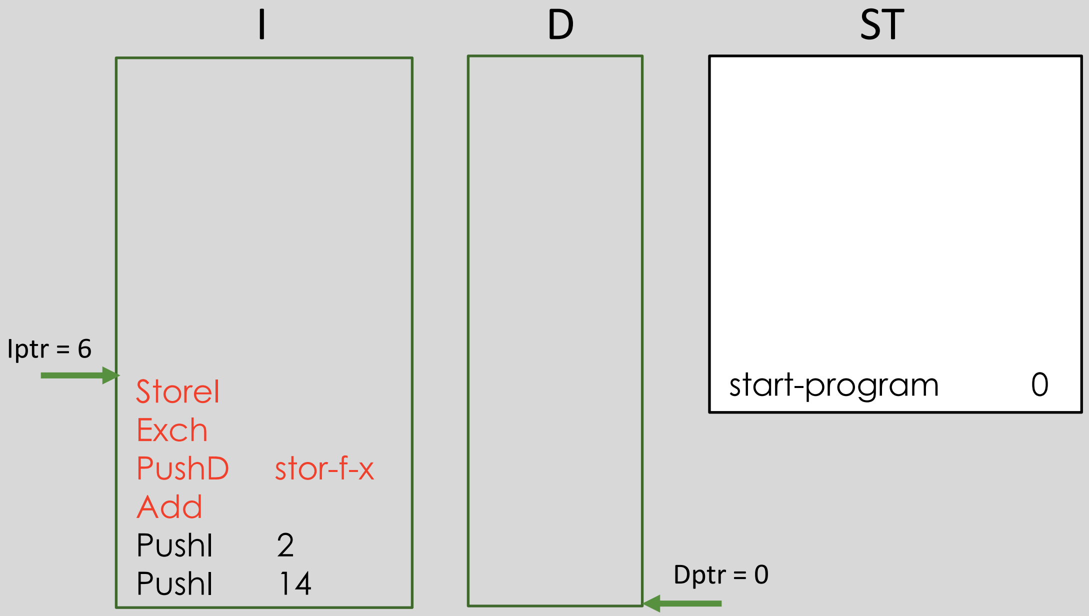
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



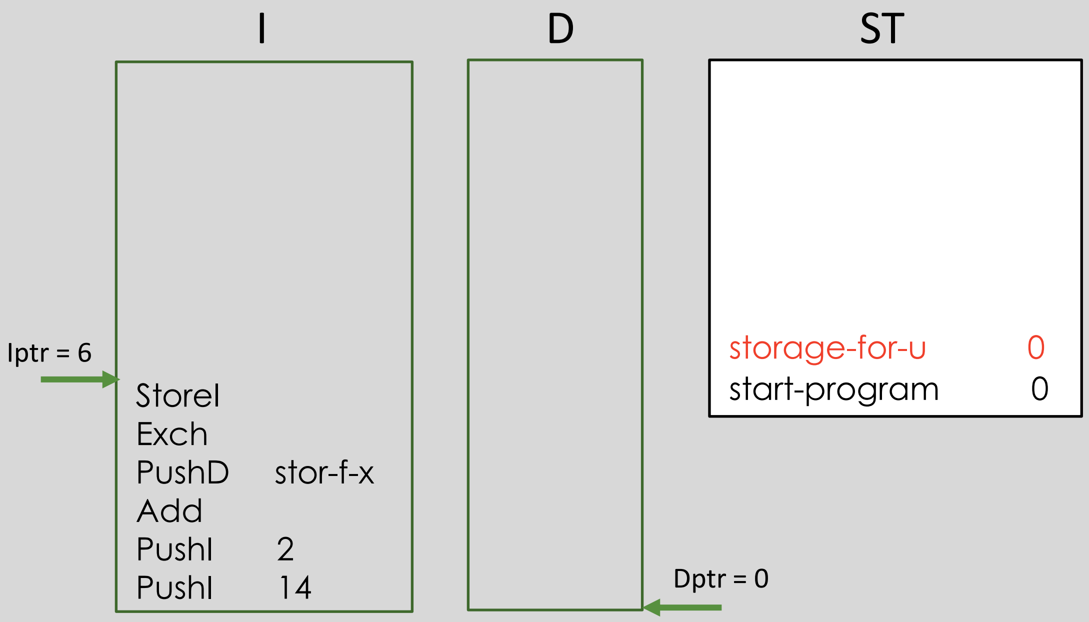
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



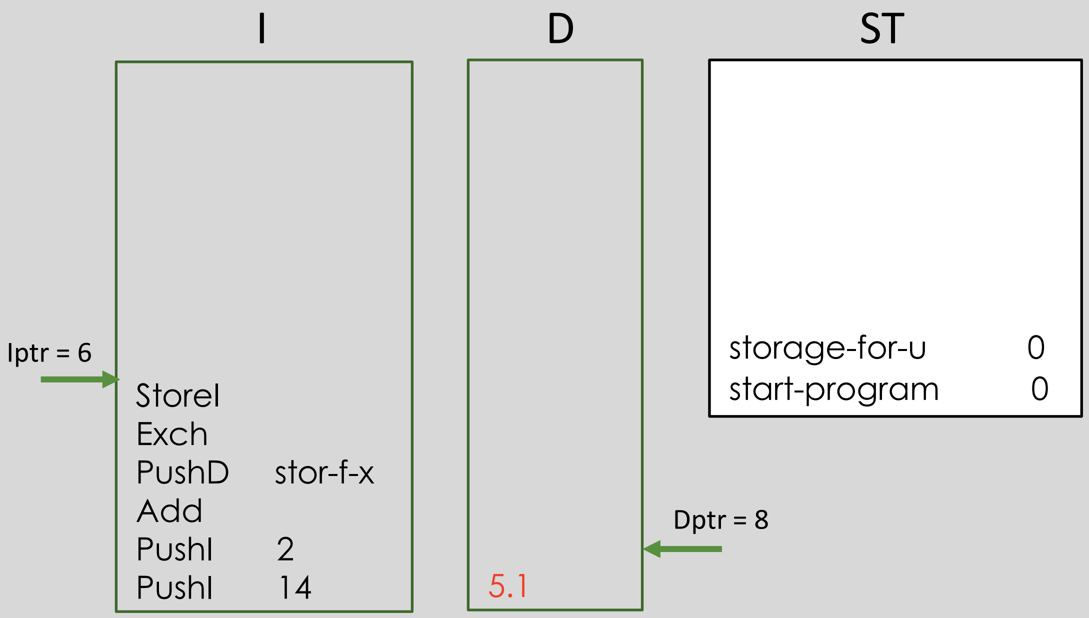
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



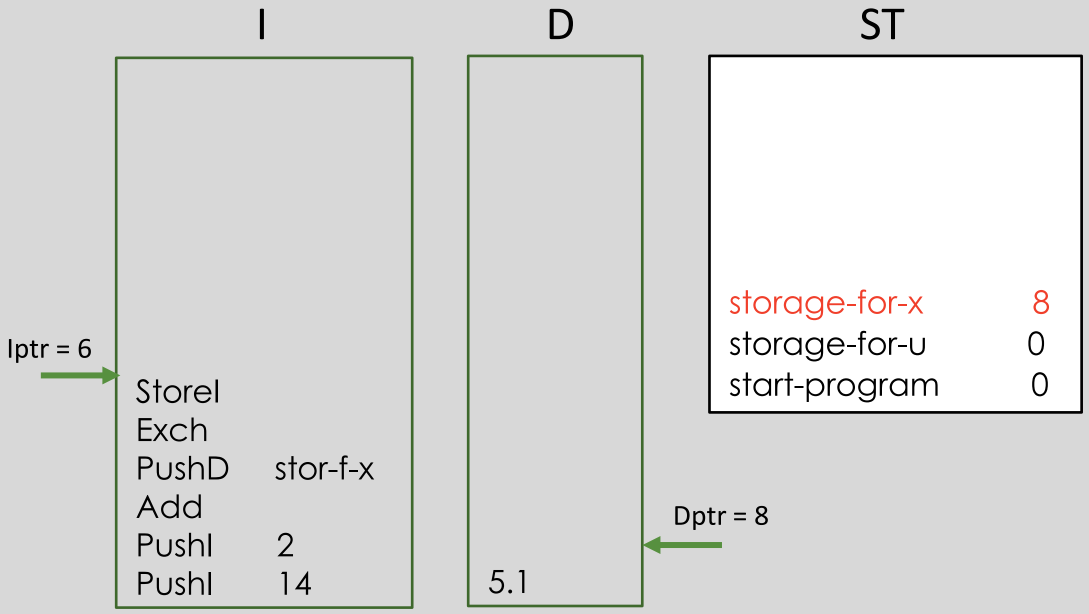
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



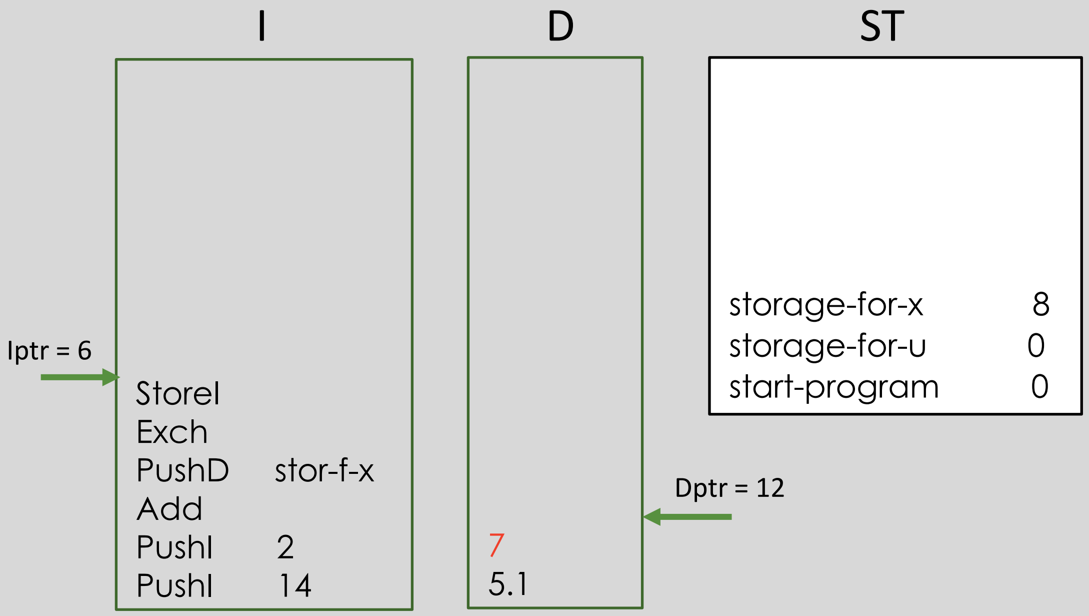
Example

```
Label      start-program
PushI      14
PushI      2
Add
PushD      storage-for-x
Exch
StoreI
DLabel     storage-for-u
DataF      5.1
DLabel     storage-for-x
DataI      7
PushI      0
Label      loop-start-1
DataD      storage-for-u
PushD      storage-for-x
LoadI
Duplicate
PushI      1
Subtract
PushD      storage-for-x
Exch
StoreI
Add
Jump       loop-start-1
```



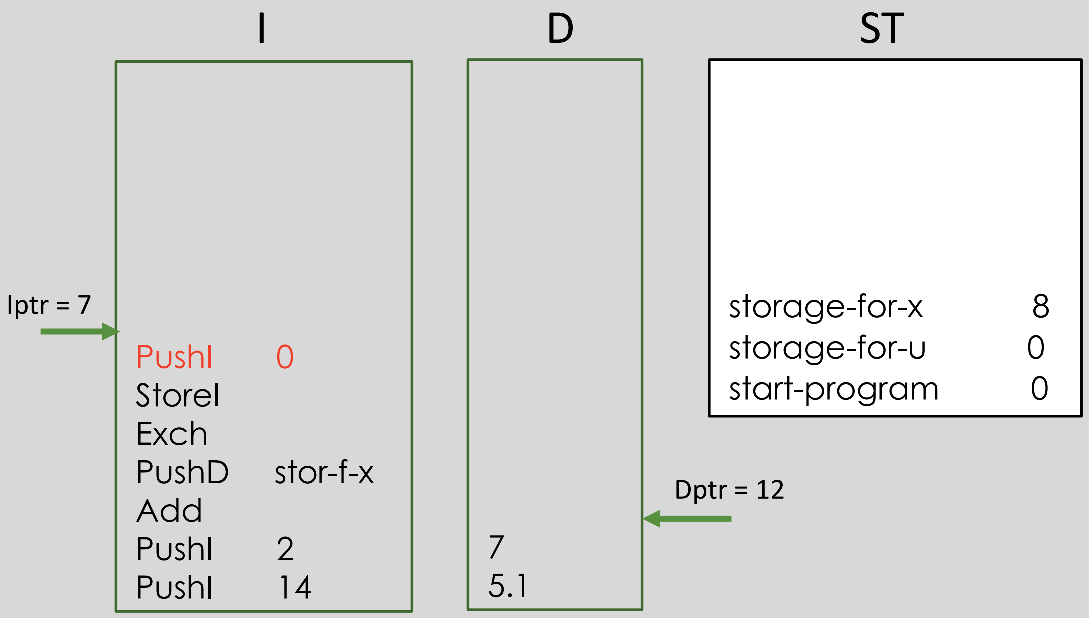
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



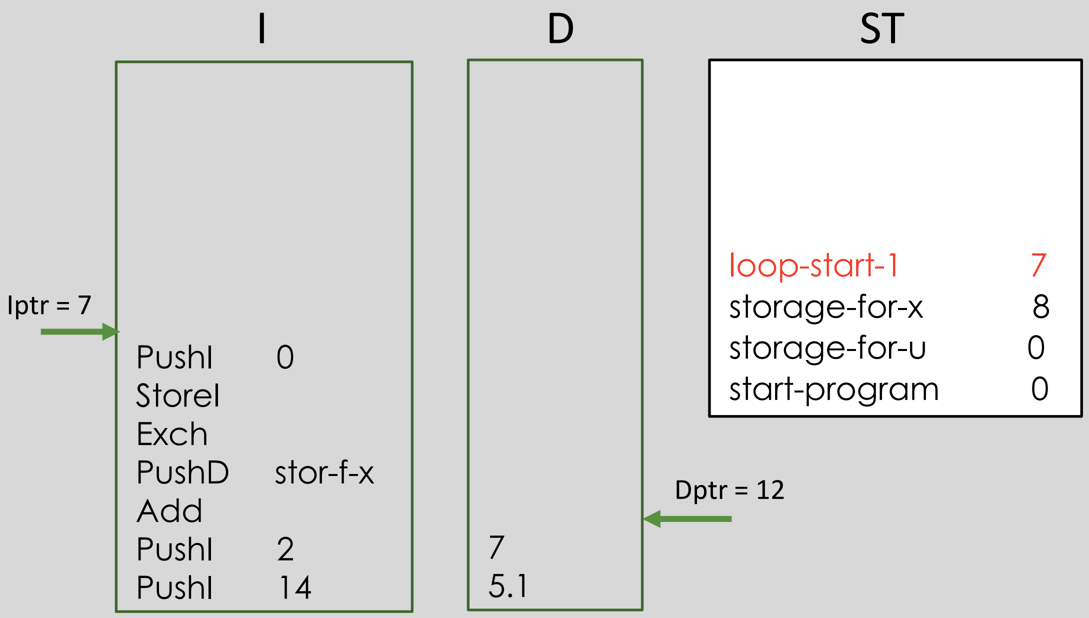
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



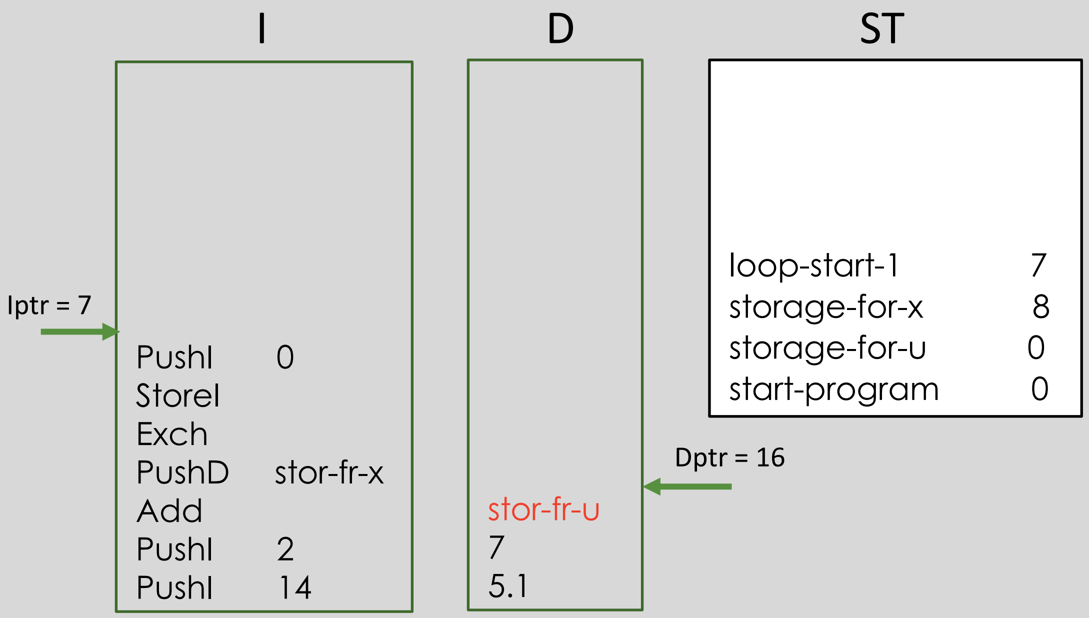
```
PushI    0
StoreI
Exch
PushD    stor-f-x
Add
PushI    2
PushI    14
```

```
7
5.1
```

loop-start-1	7
storage-for-x	8
storage-for-u	0
start-program	0

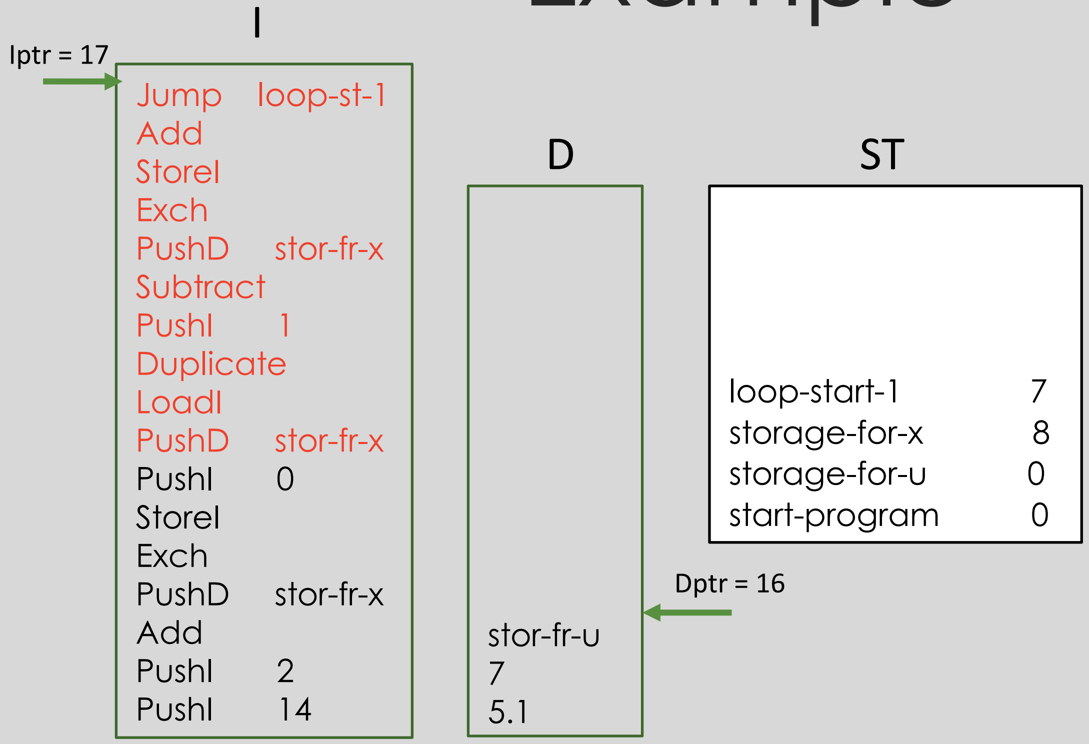
Example

```
Label      start-program
PushI      14
PushI      2
Add
PushD      storage-for-x
Exch
StoreI
DLabel     storage-for-u
DataF      5.1
DLabel     storage-for-x
DataI      7
PushI      0
Label      loop-start-1
DataD      storage-for-u
PushD      storage-for-x
LoadI
Duplicate
PushI      1
Subtract
PushD      storage-for-x
Exch
StoreI
Add
Jump       loop-start-1
```



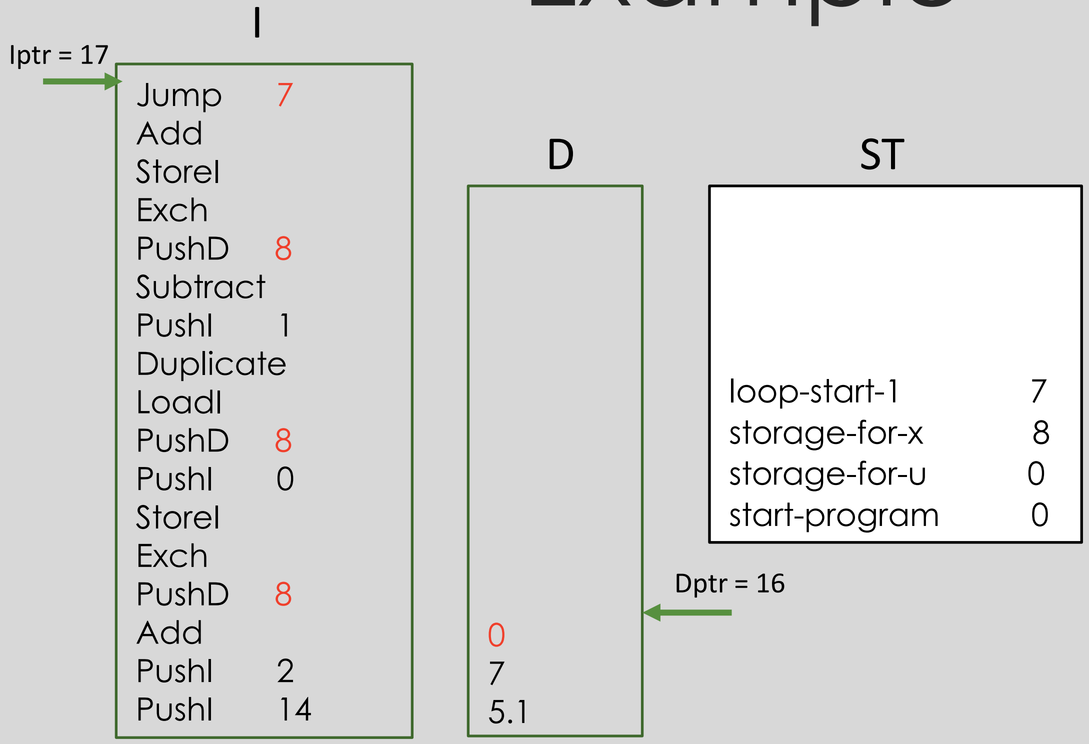
Example

```
Label      start-program
PushI      14
PushI      2
Add
PushD      storage-for-x
Exch
StoreI
DLabel     storage-for-u
DataF      5.1
DLabel     storage-for-x
DataI      7
PushI      0
Label      loop-start-1
DataD      storage-for-u
PushD      storage-for-x
LoadI
Duplicate
PushI      1
Subtract
PushD      storage-for-x
Exch
StoreI
Add
Jump       loop-start-1
```



Example

```
Label      start-program
PushI      14
PushI      2
Add
PushD      storage-for-x
Exch
StoreI
DLabel     storage-for-u
DataF      5.1
DLabel     storage-for-x
DataI      7
PushI      0
Label      loop-start-1
DataD      storage-for-u
PushD      storage-for-x
LoadI
Duplicate
PushI      1
Subtract
PushD      storage-for-x
Exch
StoreI
Add
Jump       loop-start-1
```



I:

```
Jump       7
Add
StoreI
Exch
PushD      8
Subtract
PushI      1
Duplicate
LoadI
PushD      8
PushI      0
StoreI
Exch
PushD      8
Add
PushI      2
PushI      14
```

D:

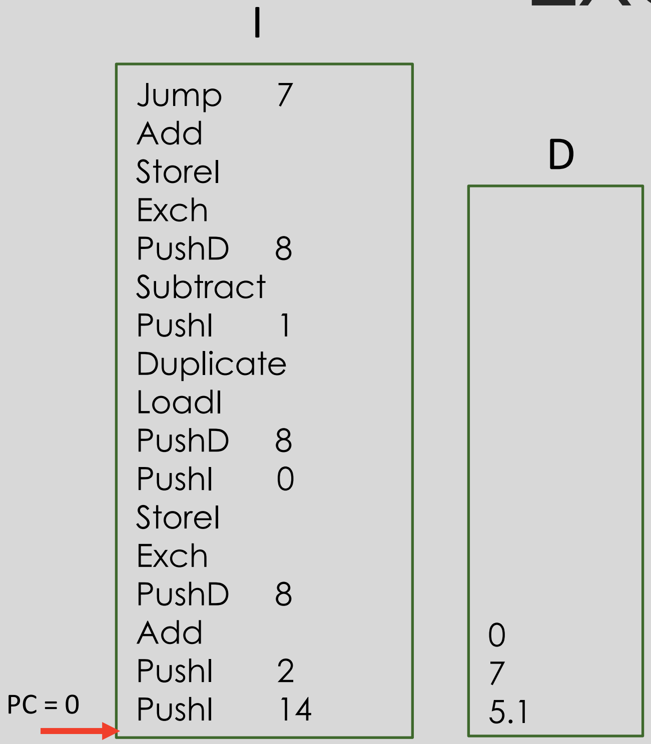
```
0
7
5.1
```

ST:

loop-start-1	7
storage-for-x	8
storage-for-u	0
start-program	0

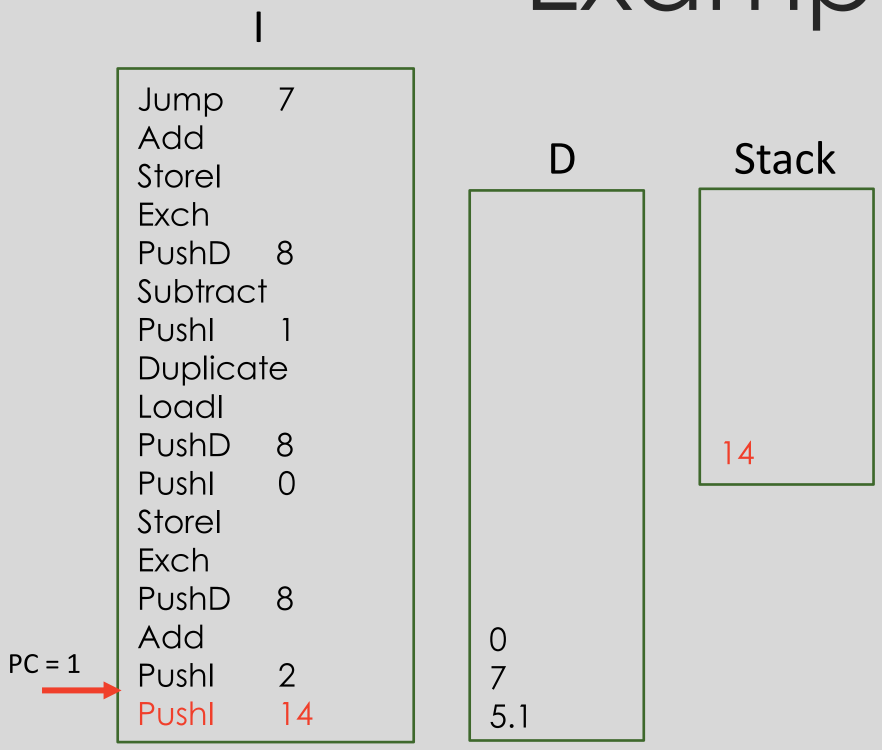
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```

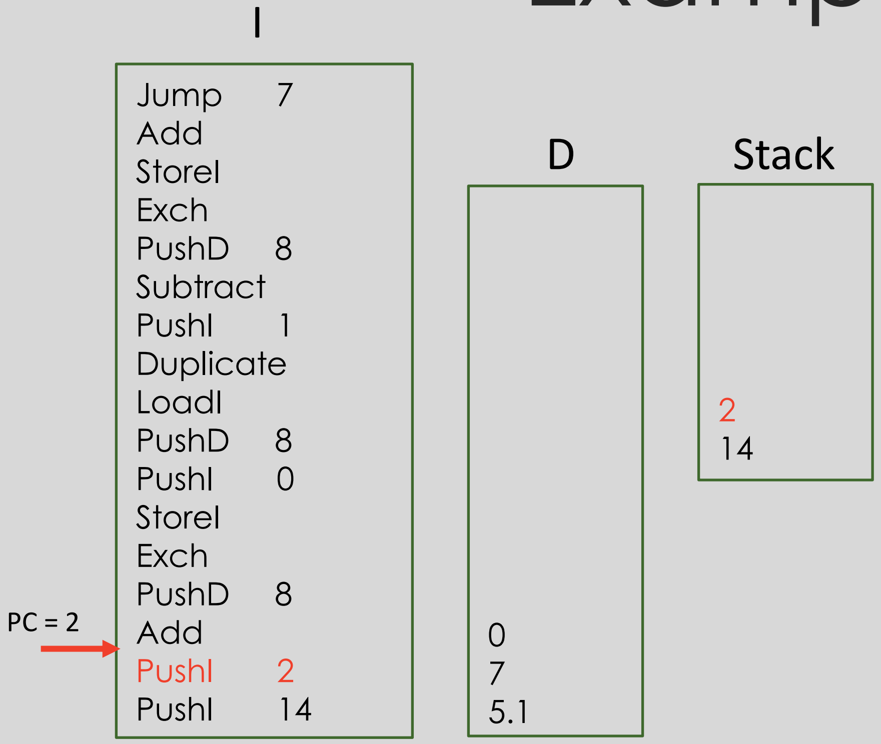


```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

```
0
7
5.1
```

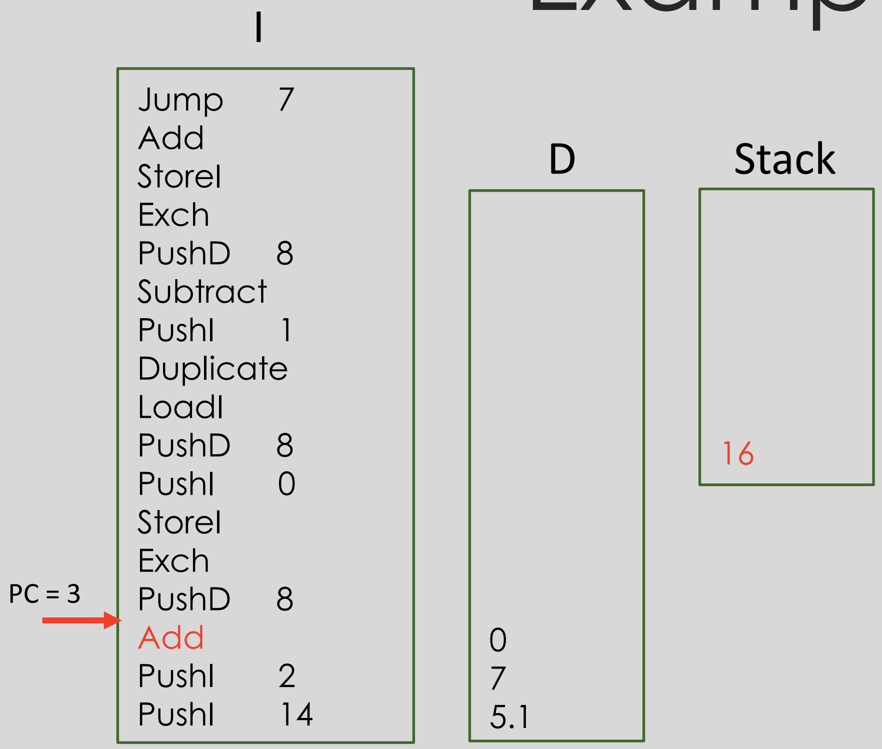
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



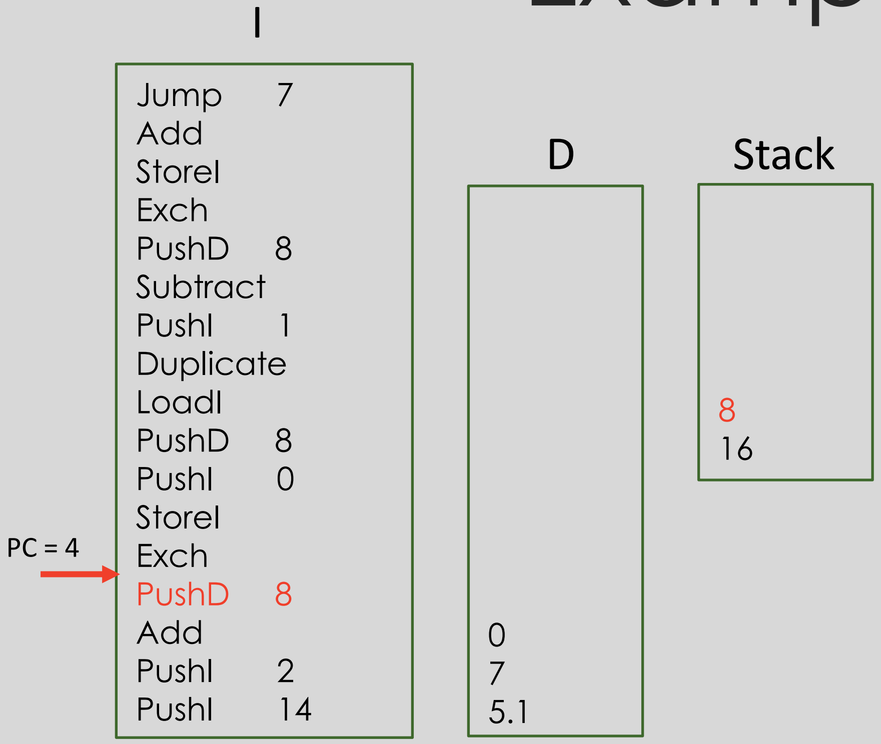
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



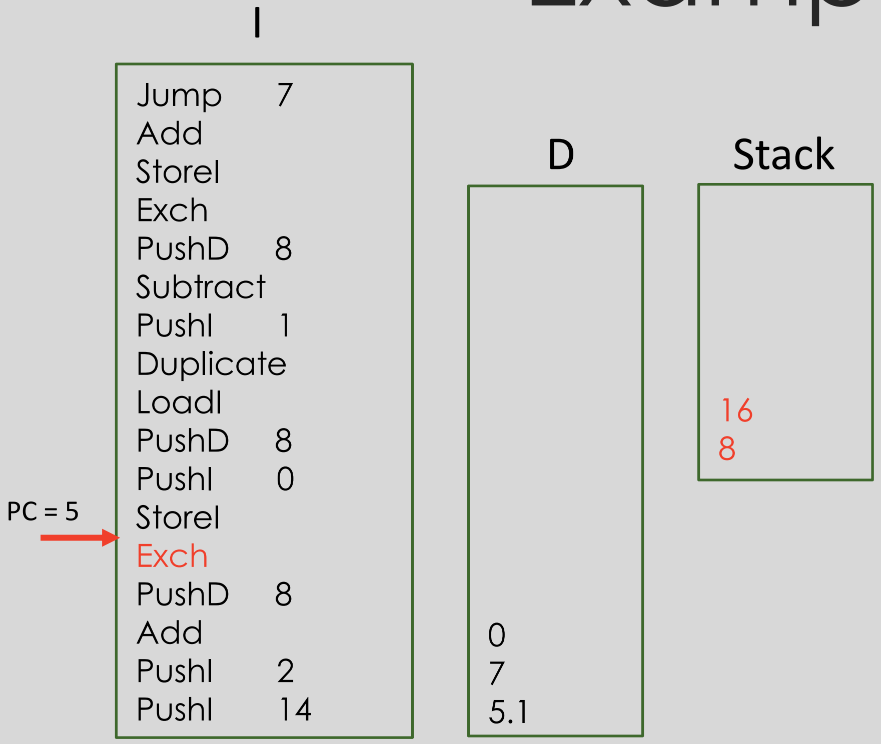
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



Example

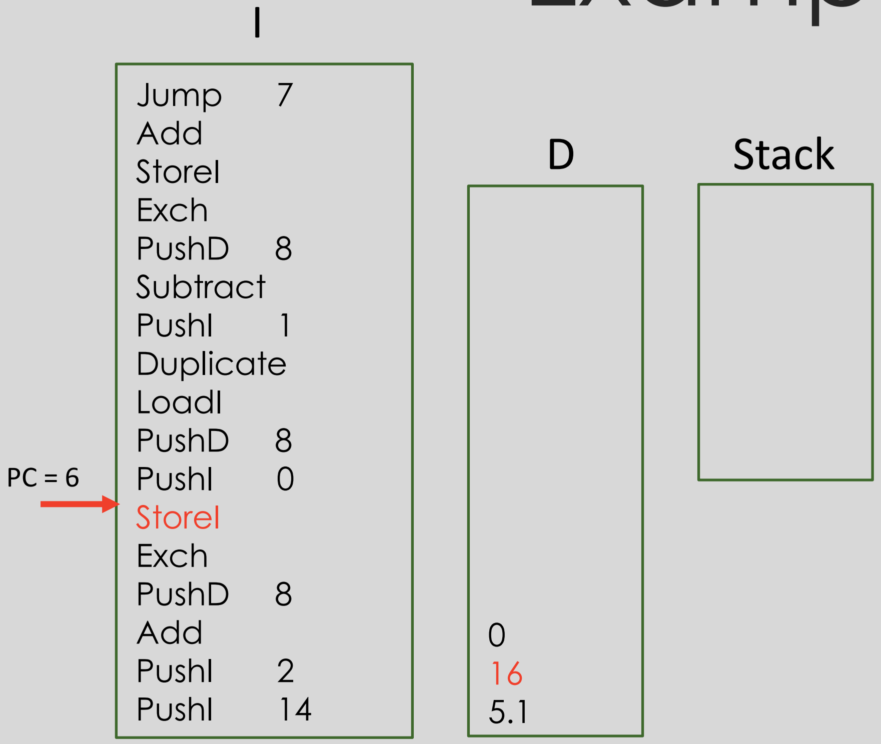
```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

Example

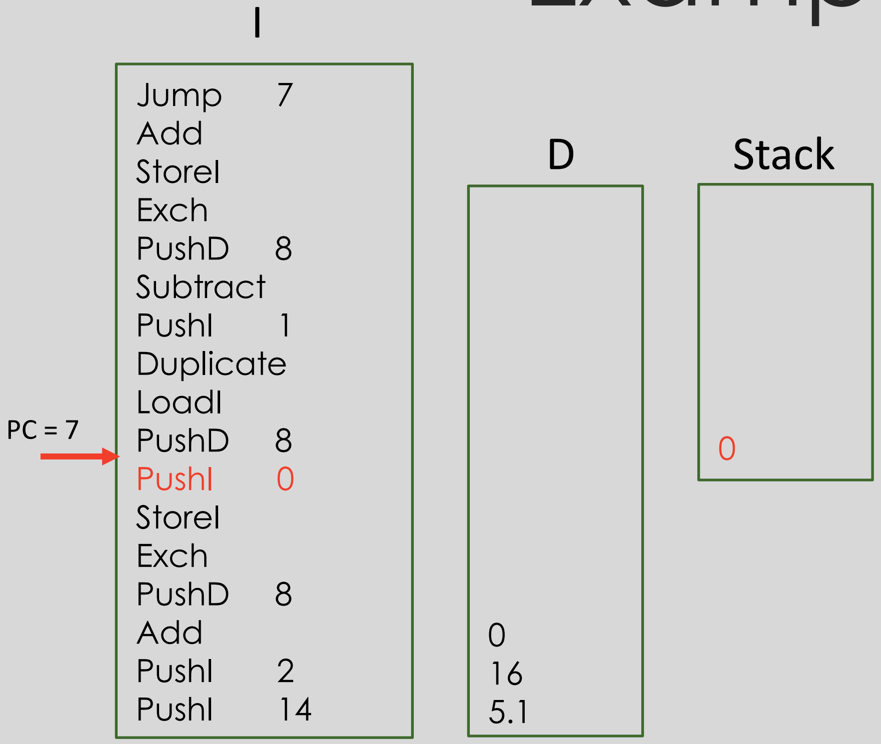
```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

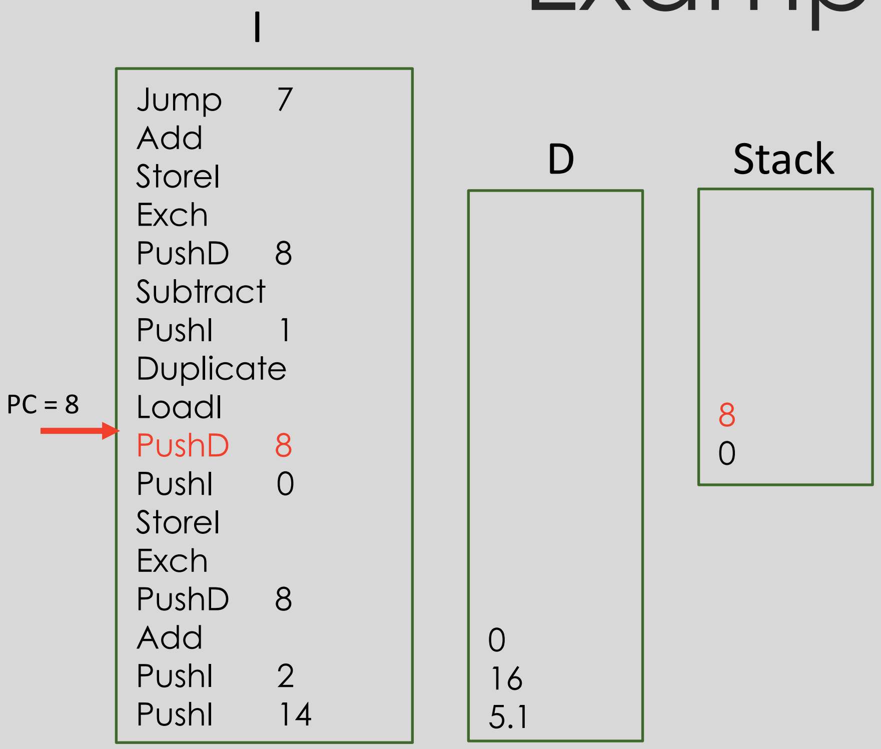
Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



Example

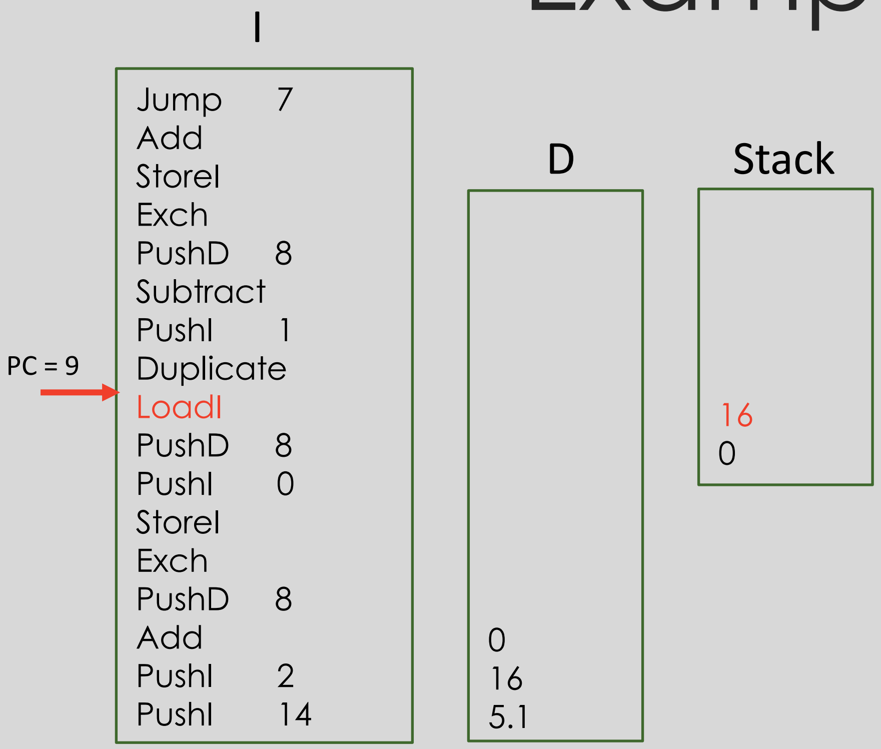
```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



```
Jump       7
Add
StoreI
Exch
PushD      8
Subtract
PushI      1
Duplicate
LoadI
PushD      8
PushI      0
StoreI
Exch
PushD      8
Add
PushI      2
PushI      14
```

Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



I:

```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

D:

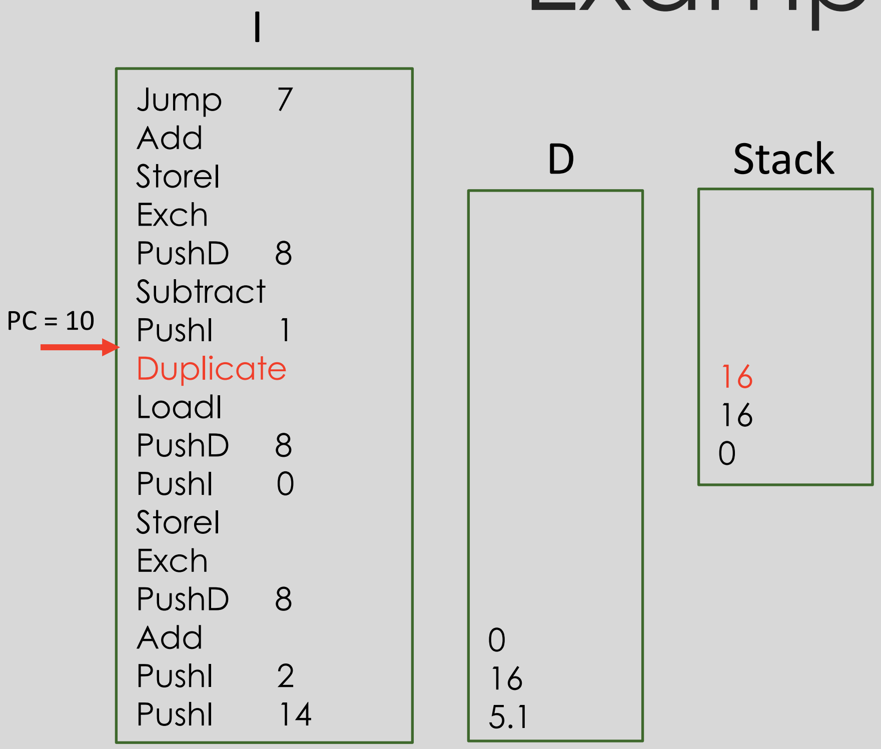
```
0
16
5.1
```

Stack:

```
16
0
```

Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



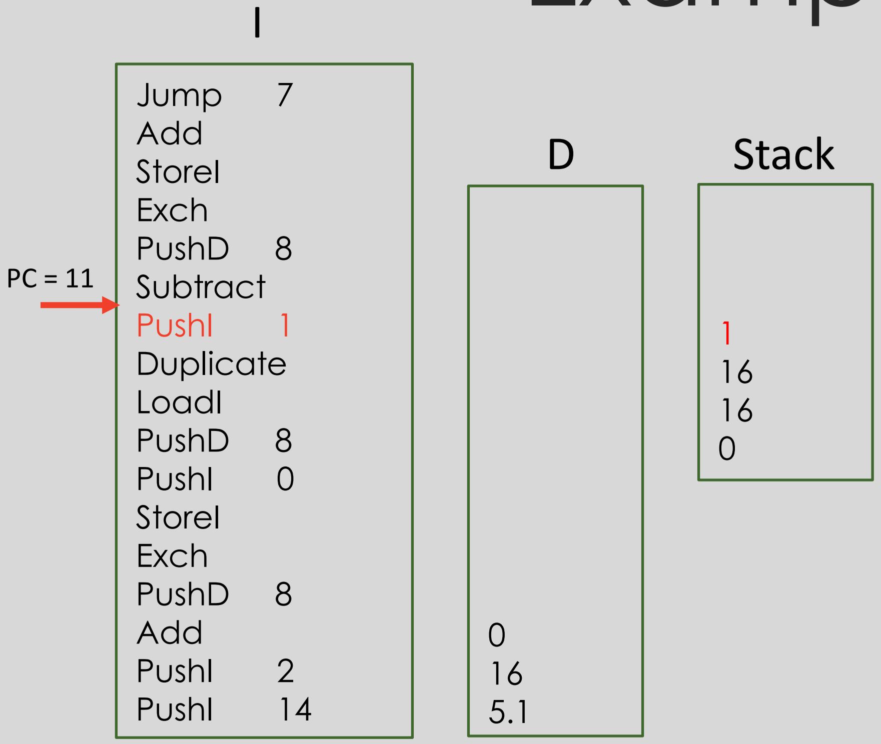
I:
```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

D:
```
0
16
5.1
```

Stack:
```
16
16
0
```

Example

```
Label       start-program
PushI       14
PushI       2
Add
PushD       storage-for-x
Exch
StoreI
DLabel      storage-for-u
DataF       5.1
DLabel      storage-for-x
DataI       7
PushI       0
Label       loop-start-1
DataD       storage-for-u
PushD       storage-for-x
LoadI
Duplicate
PushI       1
Subtract
PushD       storage-for-x
Exch
StoreI
Add
Jump        loop-start-1
```



```
Jump      7
Add
StoreI
Exch
PushD     8
Subtract
PushI     1
Duplicate
LoadI
PushD     8
PushI     0
StoreI
Exch
PushD     8
Add
PushI     2
PushI     14
```

```
0
16
5.1
```

```
1
16
16
0
```

Commenting ASM

Use square brackets [ ] to denote stack contents. The bottom of the stack corresponds to the left, the top to the right. For instance

[4 1.2 7]

is a stack with 4 on the bottom, 1.2 as the second element, and 7 on top.

Use ellipsis … to denote "other stuff on the stack that isn't important. This should always be on the left.

[… 1.2 7]

is a stack with 7 on top, and 1.2 right below that.

When placing variables on the stack, use the variable name.

[… x deltaX]

is a stack with deltaX on top, and x below that.

Commenting ASM

An endline comment should document what the stack is *after* whatever operation is on the line.

```
PushI    1     // [… 1]
PushI    41    // [… 1 41]
Add            // [… 42]
```

If you must document the stack before, use a '->' after it and also document the stack afterwards.

```
PushI    20     // [… x] -> [… x 20]
```

Commenting Java that writes ASM

Also use these conventions in java:

```
code.add(PushI, 1);          // [… 1]
code.add(PushI, 41);         // [… 1 41]
code.add(Add);               // [… 42]
```

ASMOpcode Overview

- Integer arithmetic instructions
- Floating-point arithmetic instructions
- Boolean logical instructions
- Bitwise logical instructions
- Type conversions
- Stack manipulations, loads and stores
- Control flow
- Data initialization directives

```
public enum ASMOpcode {
        // For the following arithmetic instructions, the one or two operands involved
        // (top element(s) of accumulator stack) must be integer.
        // If not, the machine halts.  The result is an int.

        Add,                    // [... a b] -> [... a+b]
        Subtract,               // [... a b] -> [... a-b]
        Negate,                 // [... a]   -> [... -a]
        Multiply,               // [... a b] -> [... a*b]
        Divide,                 // [... a b] -> [... a/b]
        Remainder,              // [... a b] -> [... a%b]

        // the following are for floating-point; they generate an error if an operand
        // is integer.  The result is floating-point.

        FAdd,                   // [... a b] -> [... a+b]
        FSubtract,              // [... a b] -> [... a-b]
        FNegate,                // [... a]   -> [... -a]
        FMultiply,              // [... a b] -> [... a*b]
        FDivide,                // [... a b] -> [... a/b]
    // There is no FRemainder.
```

```
 // the following are boolean operations; the top two (or one for BNegate)
 // elements of the accumulator must be integers.
 // Each integer is treated as boolean TRUE if it is nonzero,
// and FALSE if it is zero.
// The result is an integer: 0 if FALSE, something nonzero if TRUE

        And,      // [... a b] -> [... (a AND b)]
        Or,       // [... a b] -> [... (a OR b)]
        Nand,     // [... a b] -> [... (a NAND b)]
        Nor,      // [... a b] -> [... (a NOR b)]
        Xor,      // [... a b] -> [... (a XOR b)]
        BEqual,   // [... a b] -> [... (a NXOR b)]
        BNegate,  // [... a] -> [... (NOT a)]

 // the following are bitwise operations; the top two (or one for BTNegate)
 // elements of the accumulator must be integers.

        BTAnd,    // [... a b] -> [... (a AND b)]
        BTOr,     // [... a b] -> [... (a OR b)]
        BTNand,   // [... a b] -> [... (a NAND b)]
        BTNor,    // [... a b] -> [... (a NOR b)]
        BTXor,    // [... a b] -> [... (a XOR b)]
        BTEqual,  // [... a b] -> [...(a NXOR b)]
        BTNegate, // [... a] -> [... (NOT a)]
```

```
  // Type conversions.
ConvertF,       // Convert the top to floating
ConvertI,       // Convert the top to int.

// Accumulator stack manipulation
Duplicate,      // [... a] -> [... a a]
Exchange,       // [... a b] -> [... b a]
Pop,            // [... a b] -> [... a]
PushI,          // [... a] -> [... a i]
PushD,          // pushes the location
                   labelled with this string.
PushF,          // [... a] -> [... a f]
PushPC,         // [... a] -> [... a v]
                   (where v is the (already incremented
                    to next instruction) value of PC)
PopPC,          // [... a b] -> [... a]
                   and the PC is set to b
LoadC,  // load a byte [... a] -> [... MEM(a)]
LoadI,  // load an int [... a] -> [... IMEM(a..a+3)]
LoadF,  // load a float [... a] -> [... FMEM(a..a+7)]
```

```
StoreC,  // store a byte
         // [... a b] -> [...]
         // MEM(a) <- (b & 0xff)
StoreI,  // store an int
         // [... a b] -> [...]
         // IMEM(a..a+3) <- b
StoreF,  // store a float
         // [... a b] -> [...]
         // FMEM(a..a+7) <- b

Memtop,  // pushes the size s of the data
         // memory.  This is an invalid address. [… ] -> [… s]


// Control flow
Label,                // labels this place in the
                      // instruction store
Jump,                 // branches to label.
JumpFalse,            // Pops. Jump if value = 0
JumpTrue,             // Pops. Jump if value != 0
JumpNeg,              // Pops. Jump if value < 0
JumpPos,              // Pops. Jump if value > 0
JumpFNeg,             // Pops. Jump if value < 0.0
JumpFPos,             // Pops. Jump if value > 0.0
JumpFZero,            // Pops. Jump if value = 0.0
```

```
Call,     // Jumps to location, and pushes
          // return instruction location.
JumpV,    // [... addr] -> [...]
          // Branches to addr.
CallV,    // [... addr] -> [...]
          // Branches to addr, and pushes
          // return instruction location.

Return,   // another name for PopPC
Halt,     // stops the machine.

// Data initialization directives (low memory; done once before program starts)

DLabel,    // labels the location of the
           // next encountered data
DataC,     // stores the low 8 bits in
           // the next available location.
DataI,     // stores int in the next 4
           // available memory locations.
DataF,     // stores float in the next 8
           // available memory locations.
DataS,     // stores a string in the next
           // available memory locations
DataZ,     // zero in the next n available
           // memory locations.
DataD,     // stores a label value in the
           // next 4 available memory locations.
```

```
PStack,   // Nondestructively prints a copy of the
          // current ASM accumulator stack.  For
          // debugging purposes.

Printf,   // Does a C-style printf, with args taken
          // from the top of the stack
          // (Top of stack = first arg, etc.)

Nop;      // No operation; guaranteed to be the last
          // opcode in this list.
```

Memory usage for stack manipulation

- It is impossible to do some stack manipulations without also using data memory. For instance, [… a b] -> […a a b]. To accomplish this operation, use a temporary location in memory (that you permanently allocate):

```
DLabel    stack-temp
DataI     0
PushD     stack-temp          // [… a b] -> [… a b &temp]
Exch                          // [… a &temp b]
StoreI                        // [… a]
Duplicate                     // [… a a]
PushD     stack-temp          // [… a a &temp]
LoadI                         // [… a a b]
```

- Note that the temporary is **live** only from the StoreI to the LoadI. (A variable or memory location is *live* when it holds a value that *will be* used later.) This means that if you do this operation more than once, you can use the same temporary both times (Just do the DLabel and DataI once).

Memory usage for stack manipulation

That code only works for integer b, of course. How would you change it to work for a floating b? A byte-sized b?

Exercises:

1. Write code to do [… a b c] -> [… c a b].
2. Write code to do [… a b c] -> [… c b a].
3. Write code to do [… a b] -> [… b a b].
4. Write code to do [… a b c] -> [… a a b c].

You can test your code using the *PStack* instruction.

Loop with decrementing counter in memory

```
DLabel    counter
DataI     0
…
PushD     counter
PushI     10
StoreI                                        // counter = 10

Label  loop-start-17
PushD  counter                                // [… &counter]
LoadI                                         // [… counter]
JumpFalse            loopExit-17              // […]

…                                             // loop body

PushD  counter                                // [… &counter]
Duplicate                                     // [… &counter &counter]
LoadI                                         // [… &counter counter]
PushI 1                                       // [… &counter counter 1]
Subtract                                      // [… &counter counter-1 ]
StoreI                                        // […]
Jump  loop-start-17
Label loop-exit-17
…
```

Loop with decrementing counter on stack

```
                          // [...]
PushI      10             // [... counter]


Label  loop-start-17
duplicate                                       // [... counter counter]
JumpFalse               loopExit-17             // [... counter]

...                               // loop body [... counter] -> [... counter]
                                  // (cannot affect stack)

PushI 1                           // [... counter 1]
Subtract                          // [... counter-1]
Jump  loop-start-17

Label loop-exit-17                // arrive with [... counter]
Pop                               // [...]
...
```