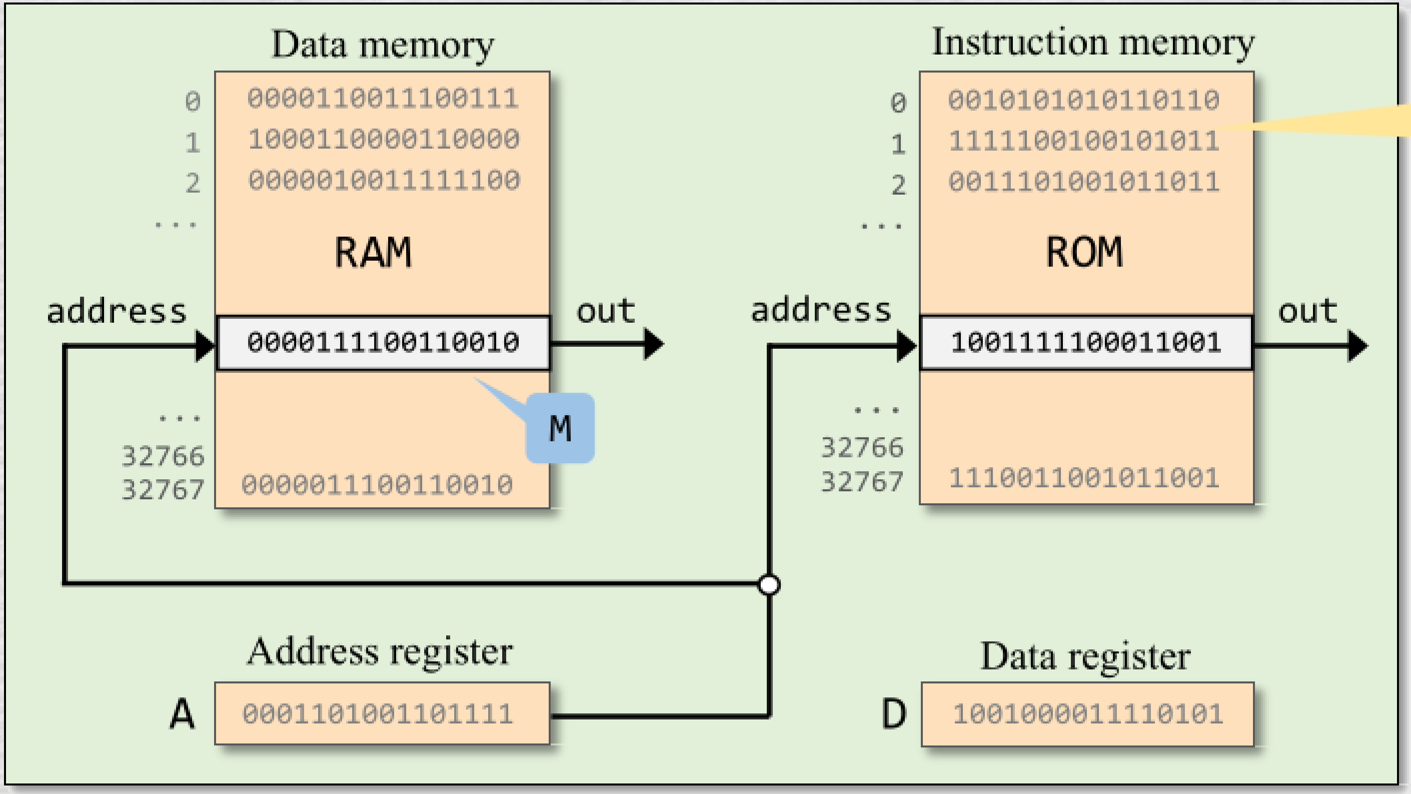
Computer Organization

Basic HACK Instructions

Hack Computer – Overview



Loaded with a sequence of 16-bit Hack instructions

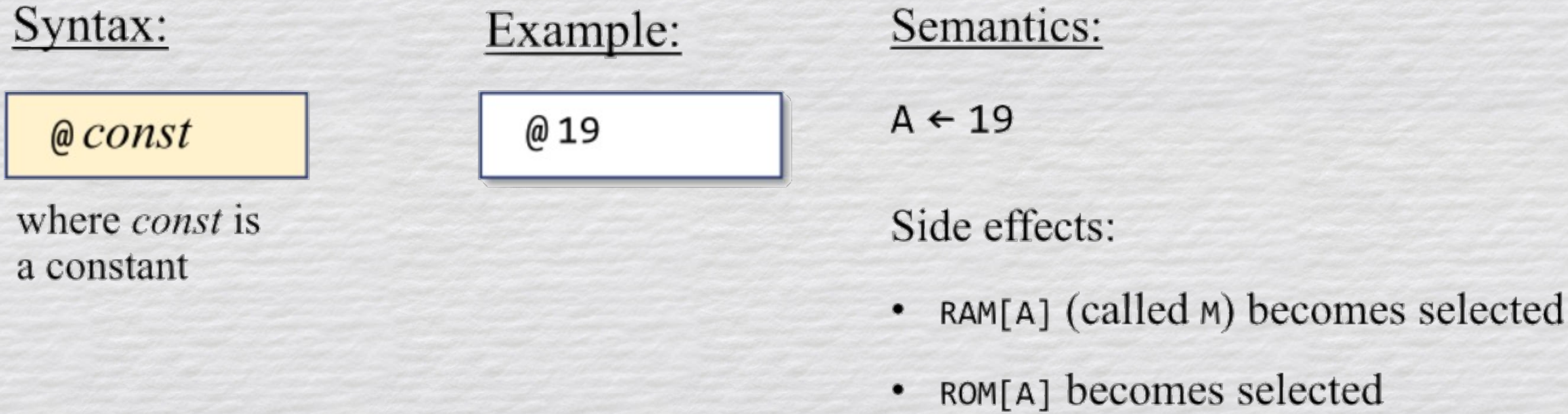
(Conceptual, partial view of the Hack computer architecture)

Assembly

- A programming language that consists of mnemonic codes for corresponding machine language instructions
  - Essentially, a collection of instructions that control the handling of data for a machine
- The HACK assembly language is split into two separate types of instructions – A and C

A Instructions

- *A* instructions are essentially **assignment** instructions
  - They are used to assign data to various registers



Syntax:

`@`*const*

where *const* is a constant

Example:

`@19`

Semantics:

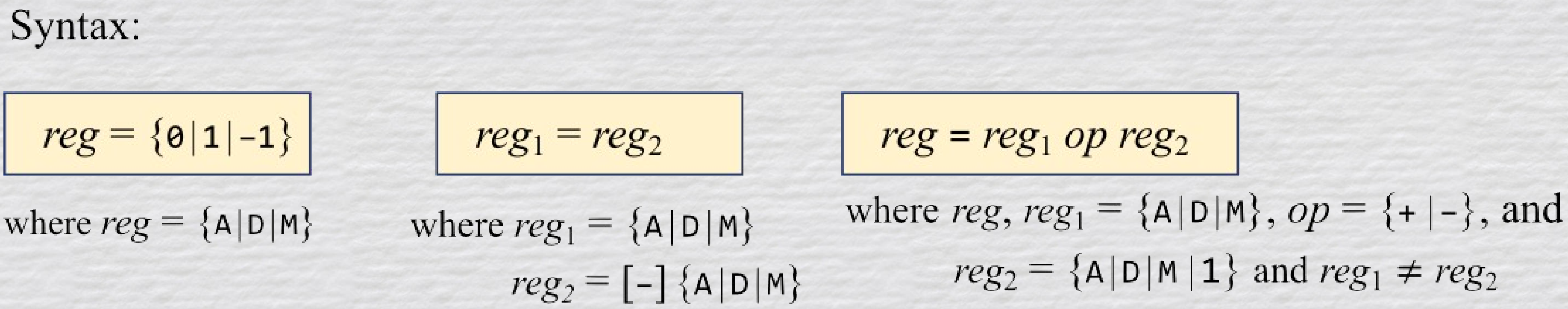
A ← 19

Side effects:

- `RAM[A]` (called `M`) becomes selected
- `ROM[A]` becomes selected

C Instructions

- C instructions actually act on the data we have stored in our system by performing various **calculations**
  - These are a fair bit more complicated by design



Syntax:

$\boxed{reg = \{\texttt{0}|\texttt{1}|\texttt{-1}\}}$

where $reg = \{\texttt{A}|\texttt{D}|\texttt{M}\}$

$\boxed{reg_1 = reg_2}$

where $reg_1 = \{\texttt{A}|\texttt{D}|\texttt{M}\}$

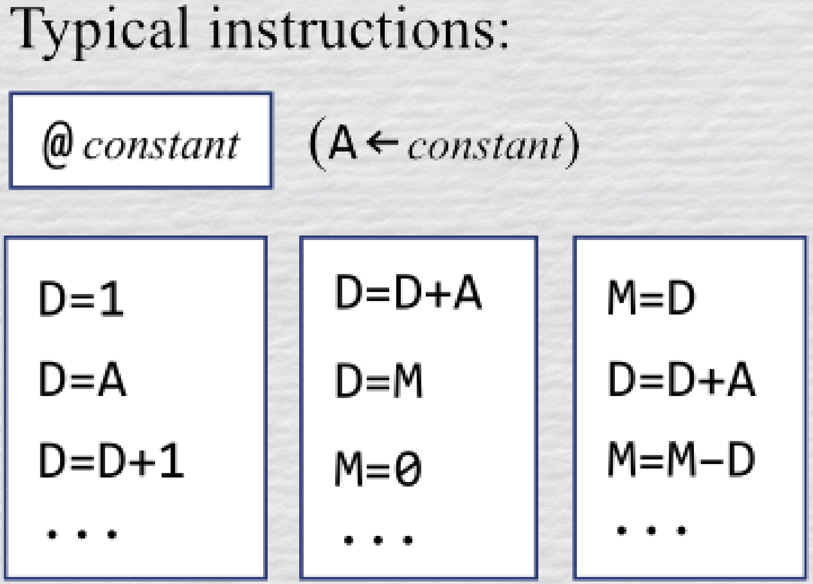
$reg_2 = [\texttt{-}]\ \{\texttt{A}|\texttt{D}|\texttt{M}\}$

$\boxed{reg = reg_1\ op\ reg_2}$

where reg, $reg_1 = \{\texttt{A}|\texttt{D}|\texttt{M}\}$, $op = \{\texttt{+}\,|\,\texttt{-}\}$, and

$reg_2 = \{\texttt{A}|\texttt{D}|\texttt{M}\,|\,\texttt{1}\}$ and $reg_1 \neq reg_2$

C Instructions – cont.



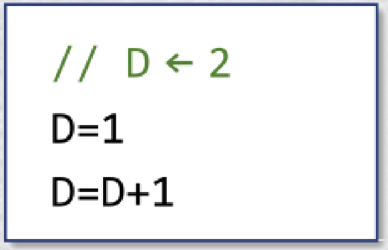
Typical instructions:

@*constant*   (A ← *constant*)

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```



```
// D ← 2
D=1
D=D+1
```

C Instructions – cont.

Typical instructions:

```
@constant
```

(A ← *constant*)

```
D=1
D=A
D=D+1
...
```

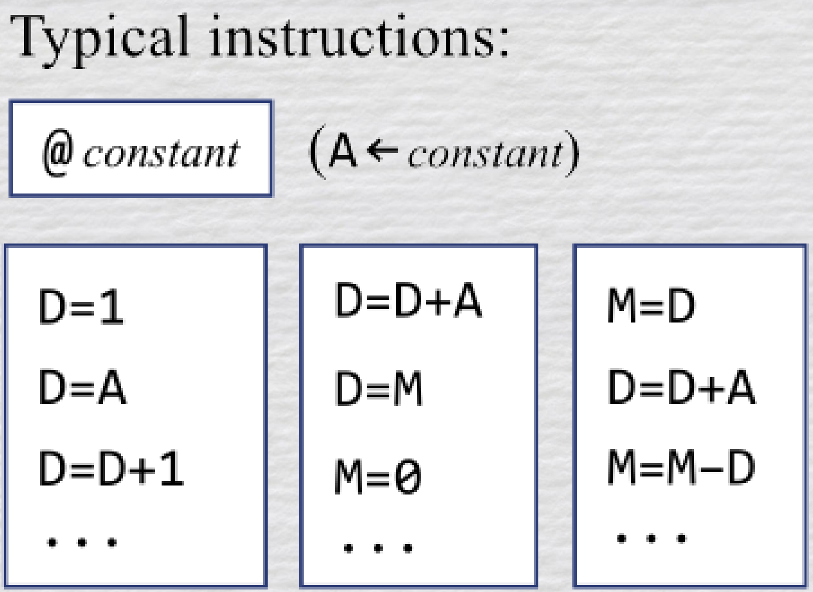
```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

```
// D ← 2
D=1
D=D+1
```

```
// D ← 1954
@1954
D=A
```

C Instructions – cont.



Typical instructions:

@constant (A ← *constant*)

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

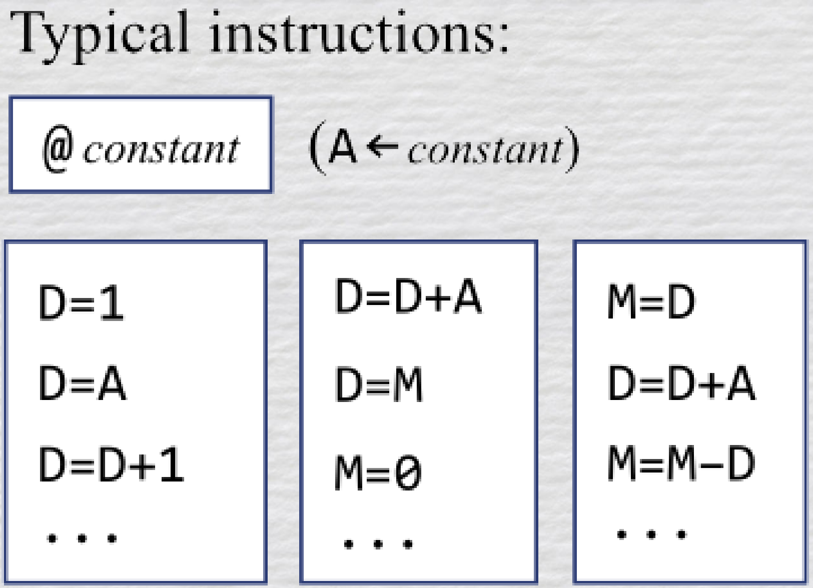
```
M=D
D=D+A
M=M-D
...
```

```
// D ← 2
D=1
D=D+1
```

```
// D ← 1954
@1954
D=A
```

```
// D ← D + 23
@23
D=D+A
```

C Instructions – cont.



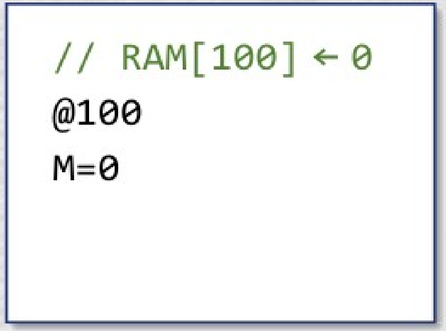
Typical instructions:

```
@constant     (A←constant)
```

```
D=1
D=A
D=D+1
...
```

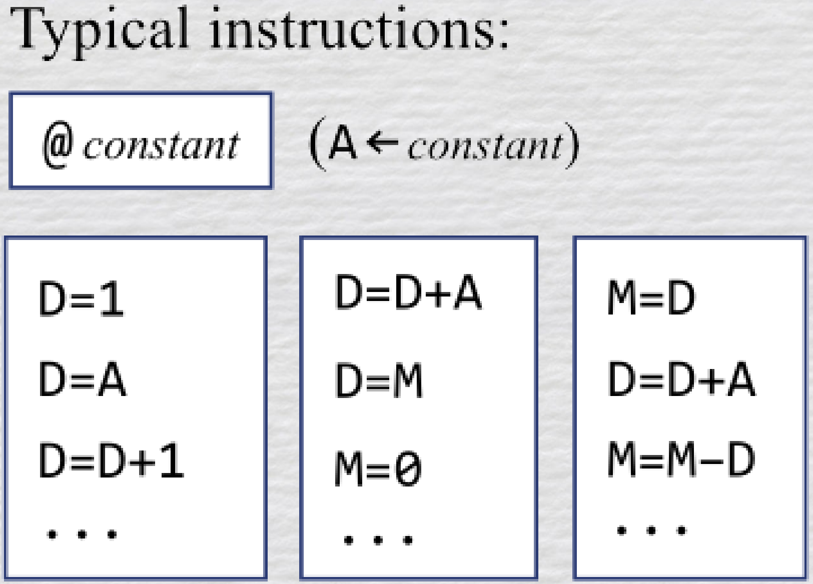
```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```



```
// RAM[100] ← 0
@100
M=0
```

C Instructions – cont.



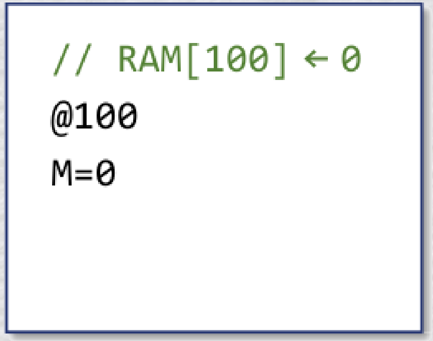
Typical instructions:

@*constant* (A←*constant*)

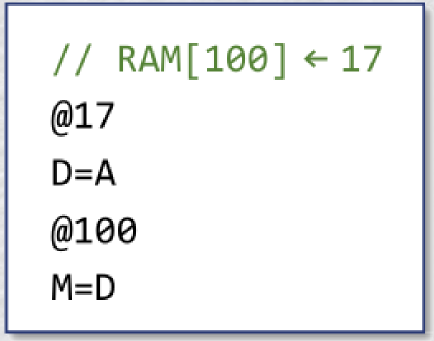
```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

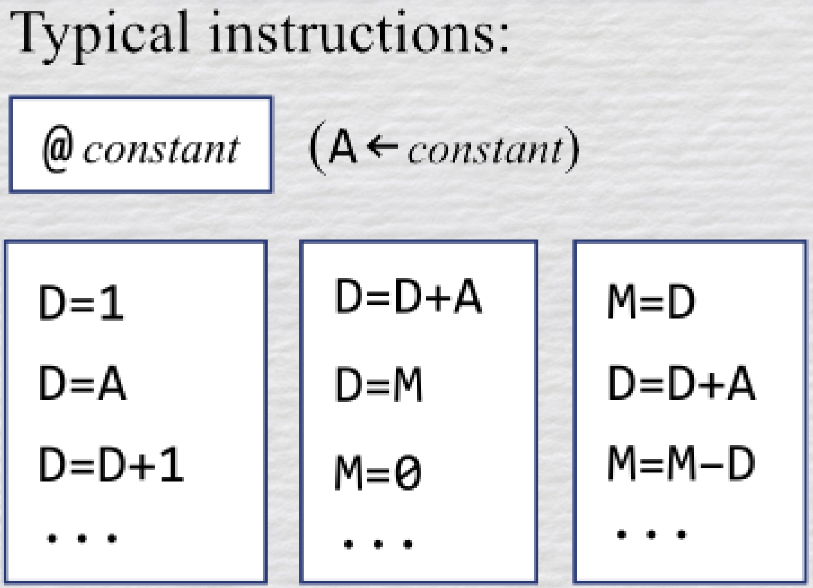


```
// RAM[100] ← 0
@100
M=0
```



```
// RAM[100] ← 17
@17
D=A
@100
M=D
```

C Instructions – cont.



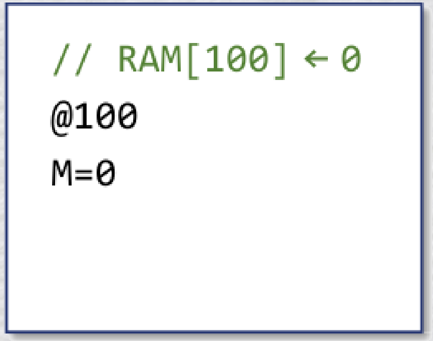
Typical instructions:

@ constant (A ← *constant*)

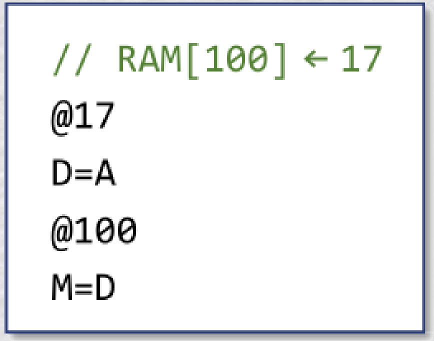
```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

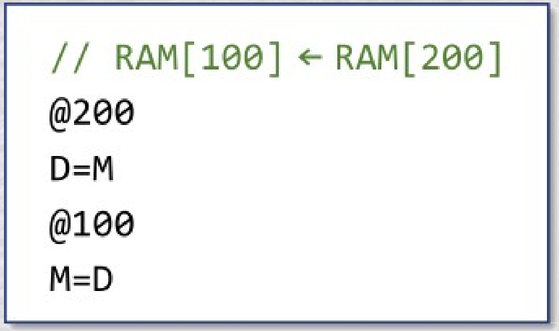
```
M=D
D=D+A
M=M-D
...
```



```
// RAM[100] ← 0
@100
M=0
```



```
// RAM[100] ← 17
@17
D=A
@100
M=D
```



```
// RAM[100] ← RAM[200]
@200
D=M
@100
M=D
```

C Instructions – cont.

Typical instructions:

```
@constant        (A ← constant)
```

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

```
// RAM[3] ← RAM[3] - 15
```

?

C Instructions – cont.

Typical instructions:

@*constant* (A←*constant*)

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

```
// RAM[3] ← RAM[3] – 15
```

?

```
// RAM[3] ← RAM[4] + 1
```

?

C Instructions – cont.

Typical instructions:

```
@constant
```

(A ← *constant*)

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

```
// RAM[3] ← RAM[3] - 15
@15
D=A
@3
M=M-D
```

```
// RAM[3] ← RAM[4] + 1
@4
D=M+1
@3
M=D
```

C Instructions – cont.

Typical instructions:

`@`*constant*   (A ← *constant*)

```
D=1
D=A
D=D+1
...
```

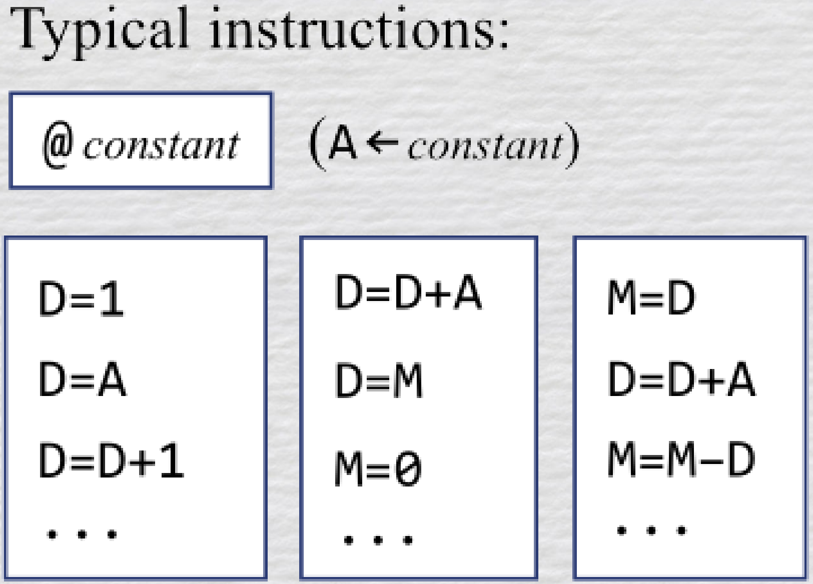
```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```

```
// Computes: RAM[2] = RAM[0] + RAM[1] + 17

?
```

C Instructions – cont.



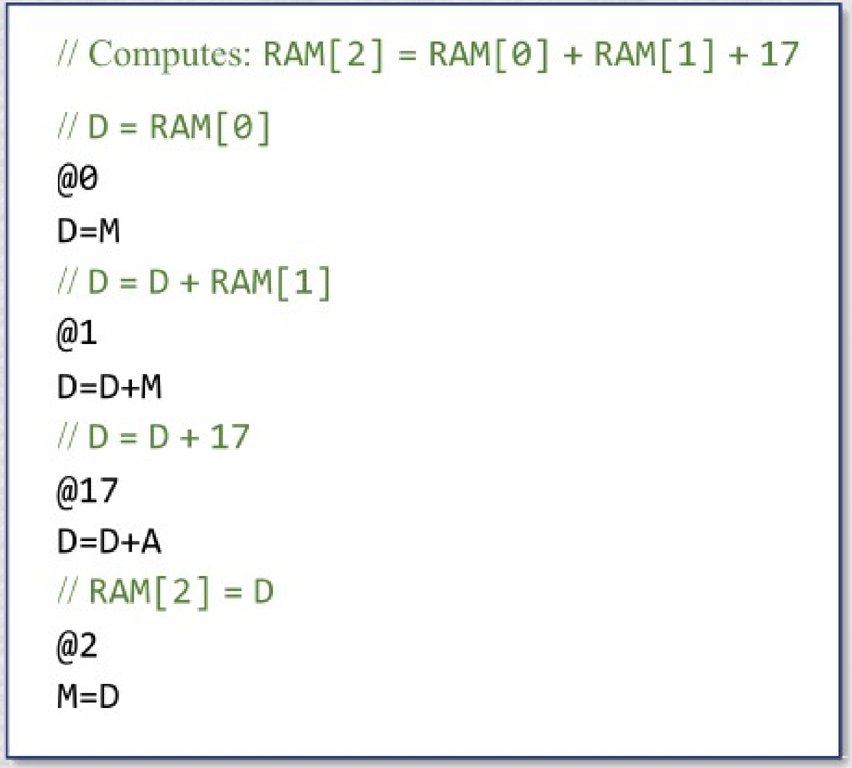
Typical instructions:

@*constant* (A ← *constant*)

```
D=1
D=A
D=D+1
...
```

```
D=D+A
D=M
M=0
...
```

```
M=D
D=D+A
M=M-D
...
```



```
// Computes: RAM[2] = RAM[0] + RAM[1] + 17

// D = RAM[0]
@0
D=M
// D = D + RAM[1]
@1
D=D+M
// D = D + 17
@17
D=D+A
// RAM[2] = D
@2
M=D
```