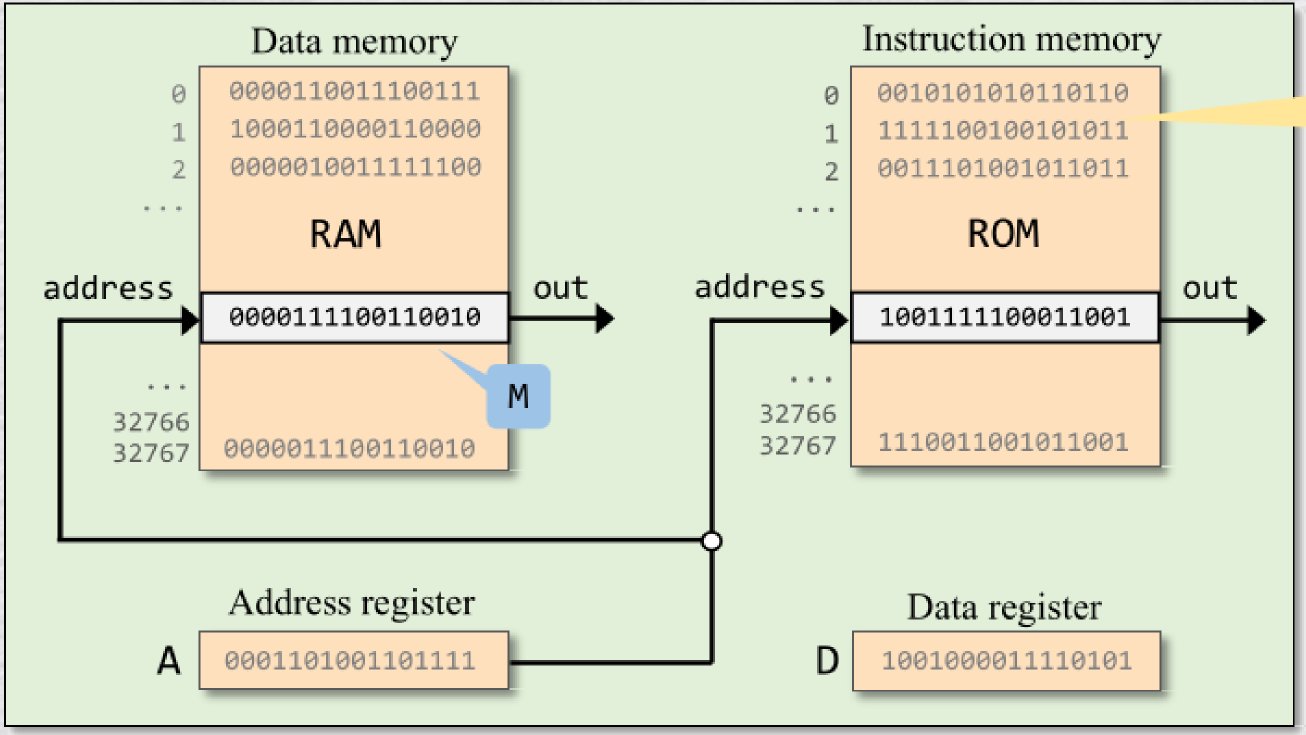
Computer Organization

HACK Binary Structure

Hack Computer – Overview



Loaded with a sequence of 16-bit Hack instructions

(Conceptual, partial view of the Hack computer architecture)

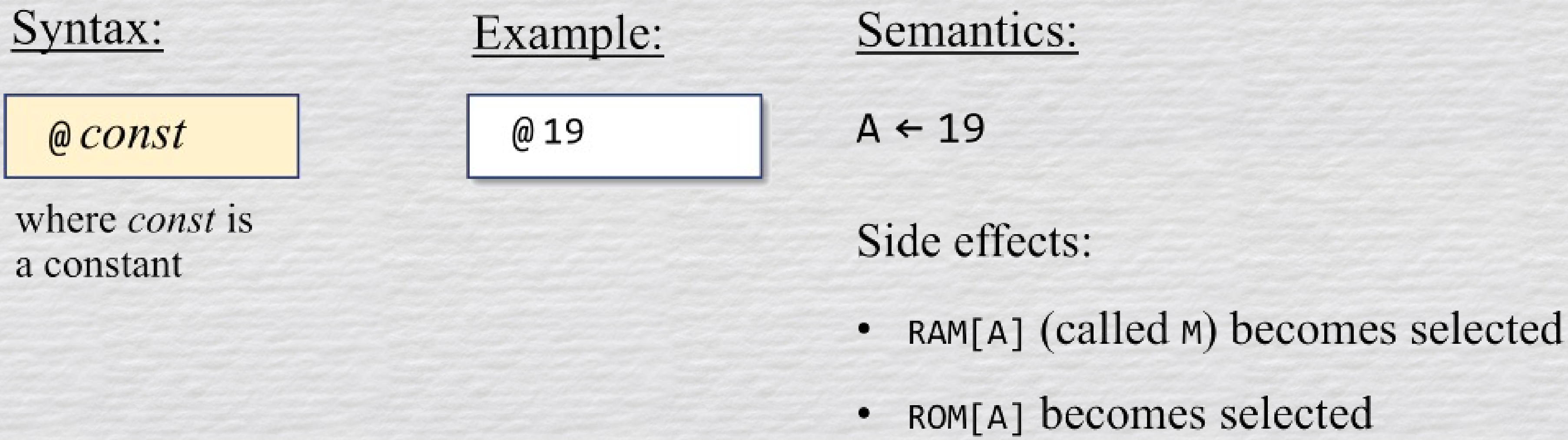
Hack Instructions

- As a reminder, the HACK assembly language is split into two separate types of instructions – *A* and *C*
    - *A* instructions are simply **assignment** instructions while *C* instructions are **computation** instructions tied to our ALU

- Both of these instruction types translate to 2 distinct binary formats with the main identifier being the MSB

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
```

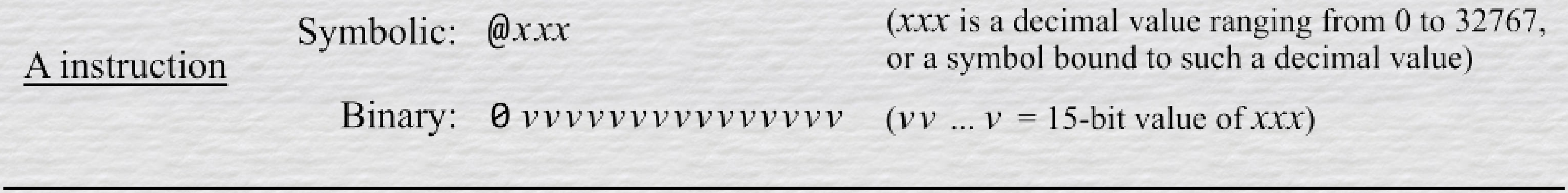
Semantics:

A ← 19

Side effects:

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

A Instructions – Binary



A instruction

Symbolic: @*xxx* (*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)

Binary: 0 *vvvvvvvvvvvvvvv* (*vv* ... *v* = 15-bit value of *xxx*)

Example:

Symbolic:

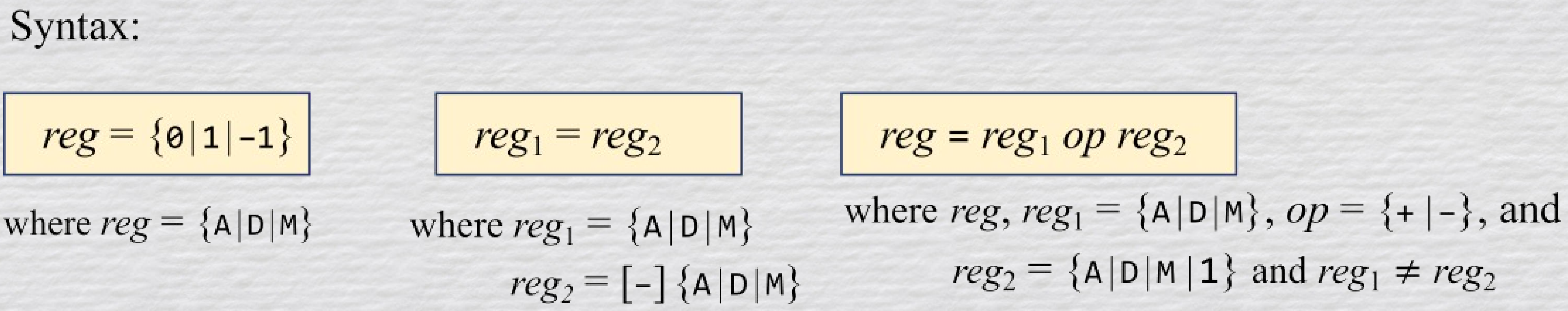
```
@6
```

Binary:

```
0000000000000110
```

C Instructions

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



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 – Components

<u>Syntax:</u> *dest* = *comp* ; *jump*   both *dest* and *jump* are optional

where:

comp =

```
0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A, A-D, D&A, D|A
            M,      !M,     -M,      M+1,      M-1, D+M, D-M, M-D, D&M, D|M
```

dest = `null, M, D, DM, A, AM, AD, ADM`   `M` stands for `RAM[A]`

jump = `null, JGT, JEQ, JGE, JLT, JNE, JLE, JMP`

C Instructions – Binary

C instruction

Symbolic: *dest* = *comp* ; *jump*

Binary: 111*accccccdddjjj*

(*comp* is mandatory.
If *dest* is empty, the = is omitted;
If *jump* is empty, the ; is omitted)

Example:

Symbolic:

```
D;JLE
```

Binary:

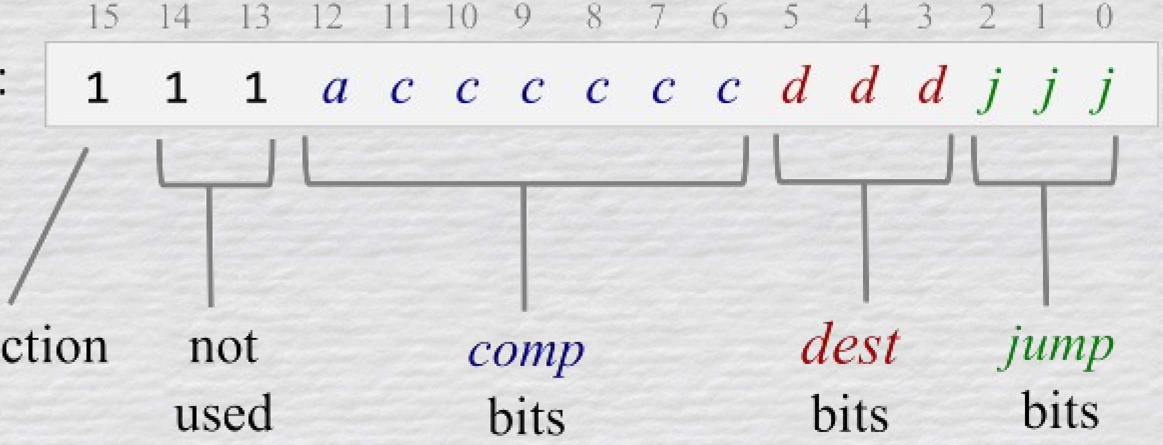
```
1110001100000110
```

C Instructions – Breakdown

Symbolic syntax: *dest* = *comp* ; *jump*

comp is mandatory.
If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary syntax:	1	1	1	*a*	*c*	*c*	*c*	*c*	*c*	*c*	*d*	*d*	*d*	*j*	*j*	*j*



C instruction op-code

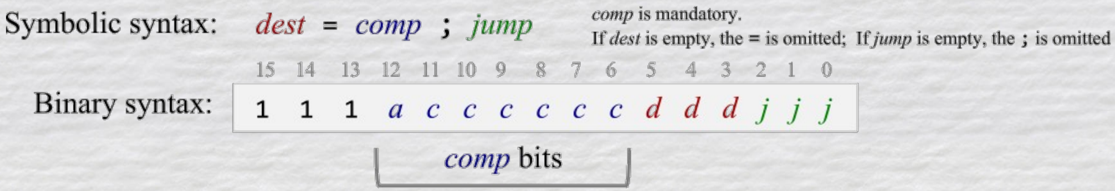
not used

comp bits

dest bits

jump bits

C Instructions – Computation



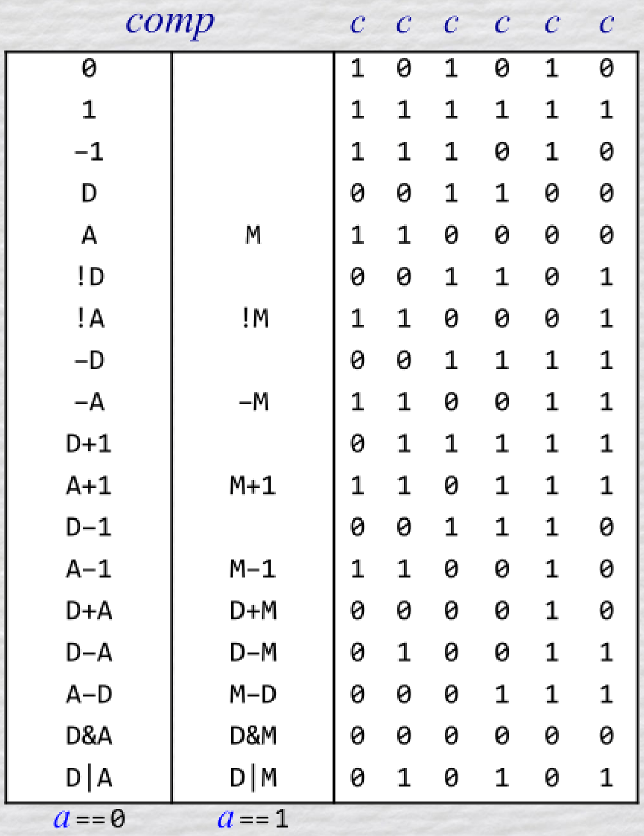
Symbolic syntax: *dest* = *comp* ; *jump*

comp is mandatory.
If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary syntax:	1	1	1	*a*	*c*	*c*	*c*	*c*	*c*	*c*	*d*	*d*	*d*	*j*	*j*	*j*

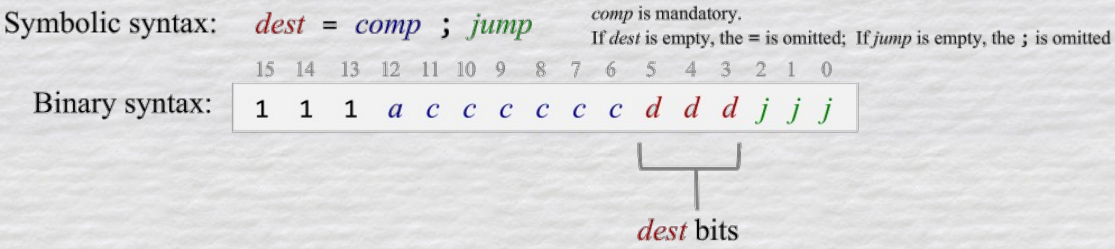
comp bits

- The computation bits consist of 7 bits
  - The first bit determines if the A or M register is being used
  - The remaining 6 bits are the inputs to the ALU select pins



comp		*c*	*c*	*c*	*c*	*c*	*c*
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a==0	*a*==1						

C Instructions – Destination



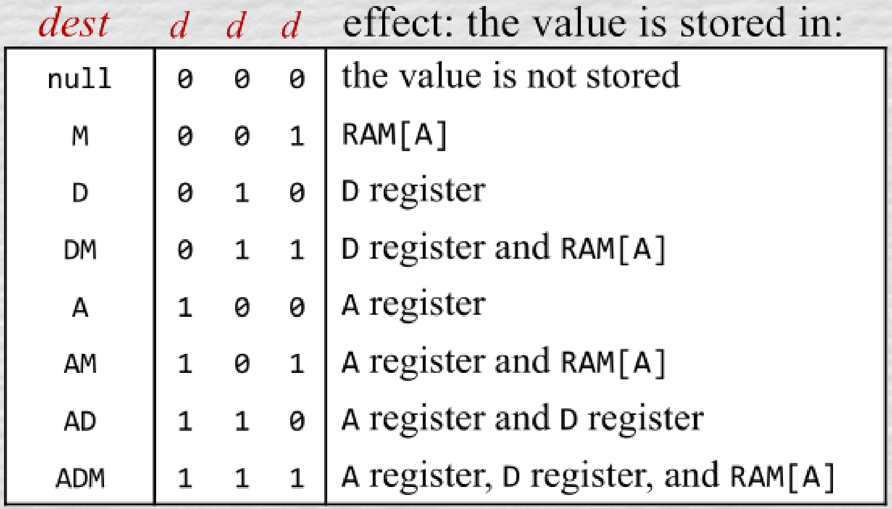
Symbolic syntax: *dest* = *comp* ; *jump*

comp is mandatory.
If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted

Binary syntax:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	*a*	*c*	*c*	*c*	*c*	*c*	*c*	*d*	*d*	*d*	*j*	*j*	*j*

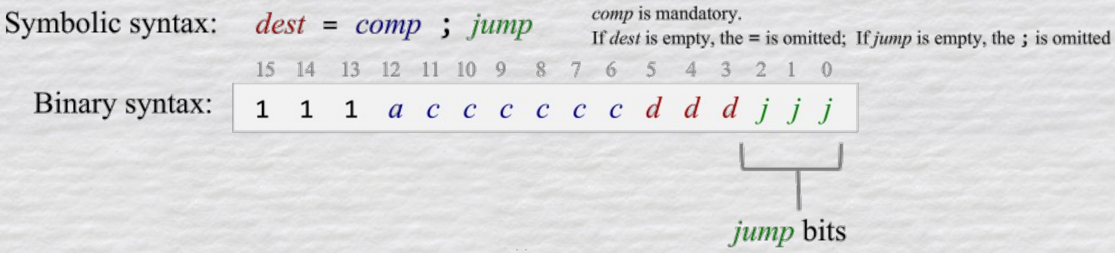
dest bits



dest	*d*	*d*	*d*	effect: the value is stored in:
null	0	0	0	the value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
DM	0	1	1	D register and RAM[A]
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
ADM	1	1	1	A register, D register, and RAM[A]

- Bits used to determine what registers the computation result will be stored in
  - These bits are optional as not every C Instruction is used to store data
- Column Alignment: **A | D | M**

C Instructions – Jump



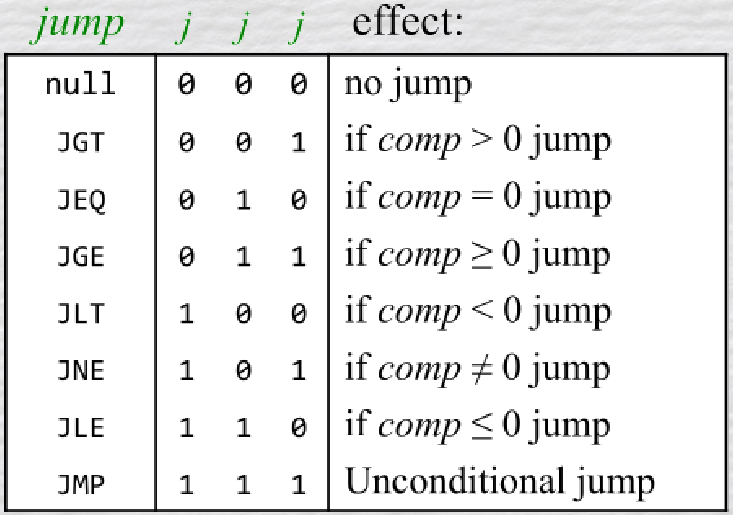
Symbolic syntax: *dest* = *comp* ; *jump*

comp is mandatory.
If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted

Binary syntax:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	*a*	*c*	*c*	*c*	*c*	*c*	*c*	*d*	*d*	*d*	*j*	*j*	*j*

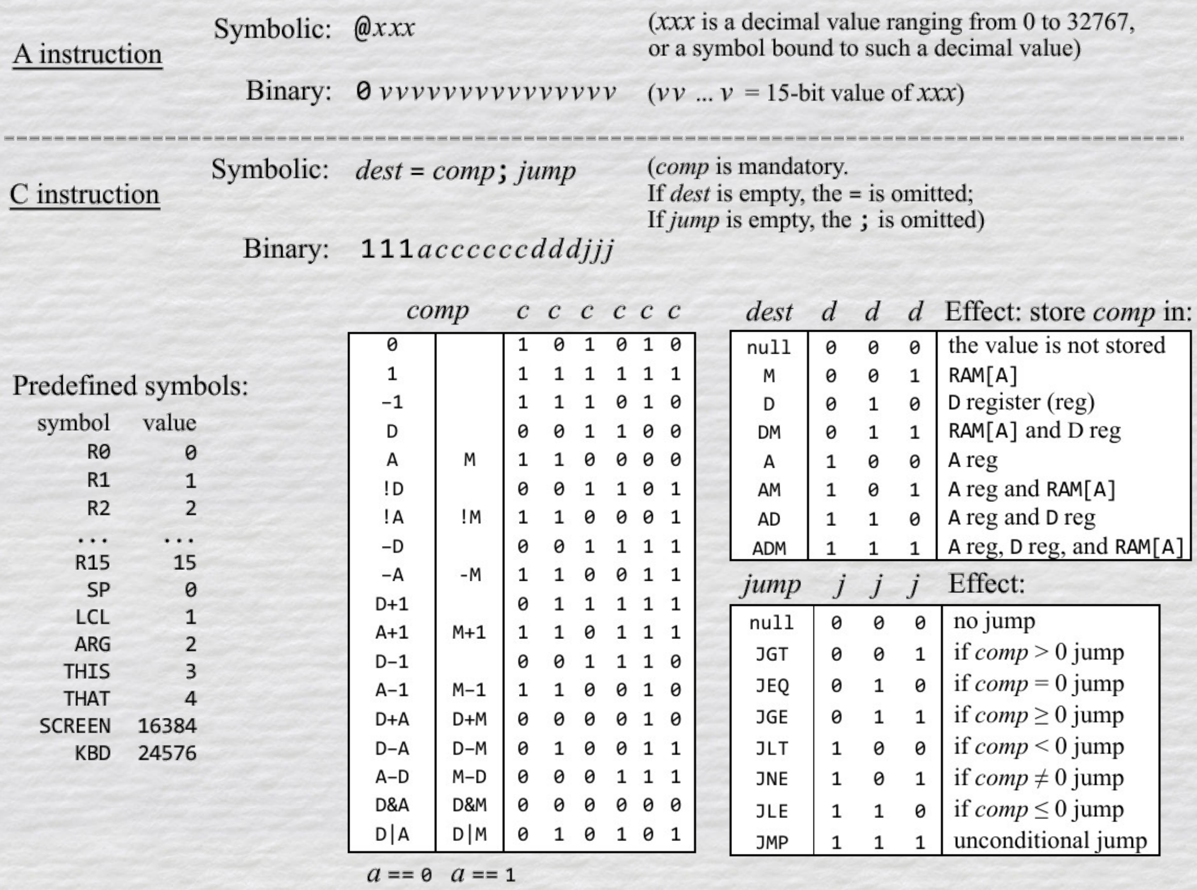
jump bits



jump	*j*	*j*	*j*	effect:
null	0	0	0	no jump
JGT	0	0	1	if *comp* > 0 jump
JEQ	0	1	0	if *comp* $= 0$ jump
JGE	0	1	1	if *comp* ≥ 0 jump
JLT	1	0	0	if *comp* < 0 jump
JNE	1	0	1	if *comp* $\neq 0$ jump
JLE	1	1	0	if *comp* ≤ 0 jump
JMP	1	1	1	Unconditional jump

- Bits used to determine if the Program Counter should load a new value
  - These bits are optional as not every C Instruction is used to jump
- Column Alignment: **Lt | Eq | Gt**

Instructions - Overview



A instruction

Symbolic: @*xxx* (*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)

Binary: 0 *vvvvvvvvvvvvvvv* (*vv* ... *v* = 15-bit value of *xxx*)

C instruction

Symbolic: *dest* = *comp* ; *jump* (*comp* is mandatory. If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted)

Binary: 111*accccccdddjjj*

Predefined symbols:

symbol	value
R0	0
R1	1
R2	2
...	...
R15	15
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
SCREEN	16384
KBD	24576

comp		*c*	*c*	*c*	*c*	*c*	*c*
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a == 0	*a* == 1						

dest	*d*	*d*	*d*	Effect: store *comp* in:
null	0	0	0	the value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register (reg)
DM	0	1	1	RAM[A] and D reg
A	1	0	0	A reg
AM	1	0	1	A reg and RAM[A]
AD	1	1	0	A reg and D reg
ADM	1	1	1	A reg, D reg, and RAM[A]

jump	*j*	*j*	*j*	Effect:
null	0	0	0	no jump
JGT	0	0	1	if *comp* > 0 jump
JEQ	0	1	0	if *comp* = 0 jump
JGE	0	1	1	if *comp* ≥ 0 jump
JLT	1	0	0	if *comp* < 0 jump
JNE	1	0	1	if *comp* ≠ 0 jump
JLE	1	1	0	if *comp* ≤ 0 jump
JMP	1	1	1	unconditional jump