CHEAT SHEET

HACK PROGRAMMING

1. Memory Elements : D, A, M (RAM[A]) for HACK asm
   (D, A: on-chip)

2. For any operation (ALU), data needs to be in CPU
   ⇒ otherwise, copy from RAM → CPU
   M D

3. Always start with a Pseudocode comprising D, A, M
   ⇒ Use R0, ..., R15, SCREEN, KBD
   for accessing RAM[0], ..., RAM[15], RAM[16,384], RAM[24576]
   ⇒ Use (LABEL) if code has any Branching

example:

```
if (D < 0)
    // jump to TRUE
    ___
    ___

else
    // FALSE
    ___
    ___

// Remaining code
___
___

//END with ∞ loop
```

```
0  @TRUE
1  D; JLT
   // FALSE
2  ___
3  ___
4  @REMAINING
5  0; JMP
(TRUE)
6      ___
7      ___
(REMAINING)
8  ___
9  ___
(END)
10     @END
11     0; JMP
```

Note: (LABEL) declaration is not assigned any address on ROM (≡ comment)

⇒ LABEL is stored in Symbol Table with value as "ROM" address of HACK asm instruction "after" declaration

⇒ example: @TRUE // @6 i.e. A=6

@REMAINING // @8 i.e. A=8

refer above @END // @10 i.e. A=10

⇒ User-defined variables are mapped based on order of "first" appearance in asm code

example: sum = 0;

```
@sum
M=0
```

i = 2;

```
@i
M=2
```

sin -1, 0, 1 valid constants as per HACK "comp"

```
@2
D=A
@i
M=D
```

sum = i

in HACK, any operation involving RAM registers has to go through CPU

```
@i
D=M
@sum
M=D
```

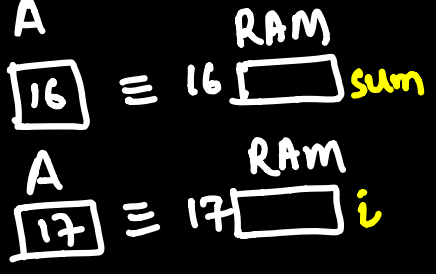
CPU ← RAM

RAM ←

Note: User defined variables are stored in data memory starting RAM[16] onwards

example: Refer above

```
@sum    // @16  i.e.
@i      // @17  i.e.
```



everytime @sum @i are now used in rest of the code, A-register on CPU will store 16 17 respectively

4 {
- Keywords (in-built symbols)
- Labels
- User-defined Variables

are for Programmer's convenience to write HACK asm code

However,

what gets stored in ROM (instruction memory) is the corresponding "numeric values" they are mapped to in Symbol Table

Example: refer above

```
@R0
@sum
@END
```

ROM view
@0
@16
@10

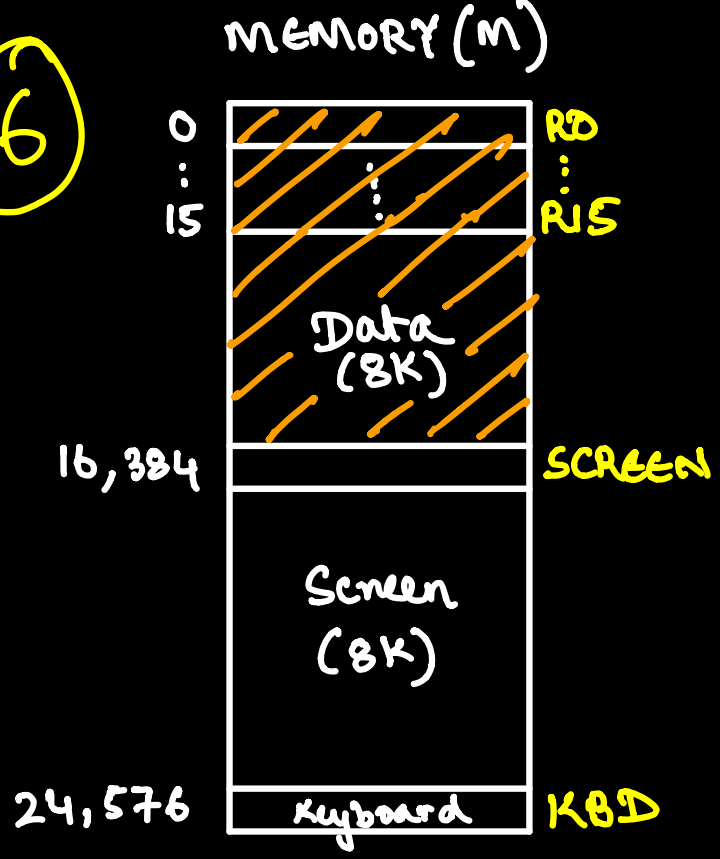
(5) A - register can have 3 interpretations based on the HACK asm statement which immediately follows

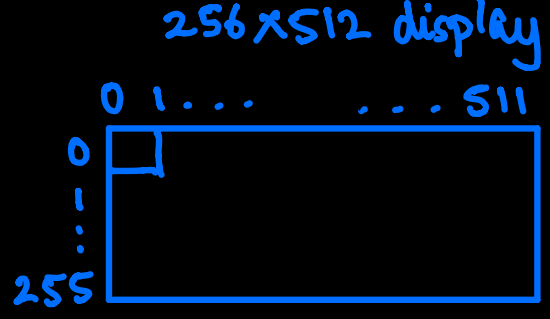
example: @20
D = A // used as constant

example: @20
D = M // used as RAM[20]

example: @20
D ; JLT // used as ROM[20]
any jump

(6)





(r, c) ; r : 0 → 255
c : 0 → 511

$\Rightarrow$ 256 x 512 pixels

$\equiv 2^8 \times 2^9 = 2^{17}$ pixels

$\equiv 2^{17}$ bits (1 pixel $\equiv$ B/W $\equiv$ 2 values $\equiv$ 1 bit)

$\equiv \frac{2^{17}}{16}$ Registers (16 bits $\equiv$ 1 Register)

$\equiv 2^{13}$ Registers

$\equiv 2^3 \times 2^{10}$ Registers

$\equiv$ 8K Registers $(1K = 2^{10})$

$\equiv$ RAM8K chip to implement Screen Memory

$\Rightarrow$ Each row in Display,

$\equiv$ 512 pixels

$\equiv$ 512 bits

$\equiv \frac{512}{16} = \frac{2^9}{2^4} =$ 32 Registers

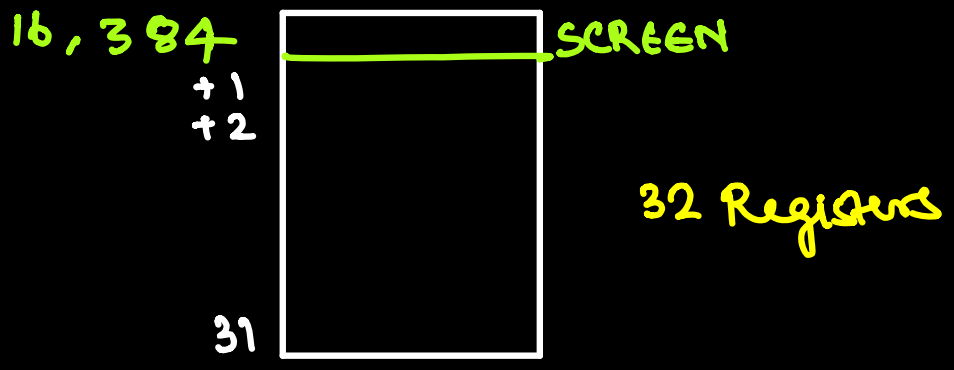
i.e. 256 rows each requires 32 Registers

$\equiv$ 256 x 32 Registers

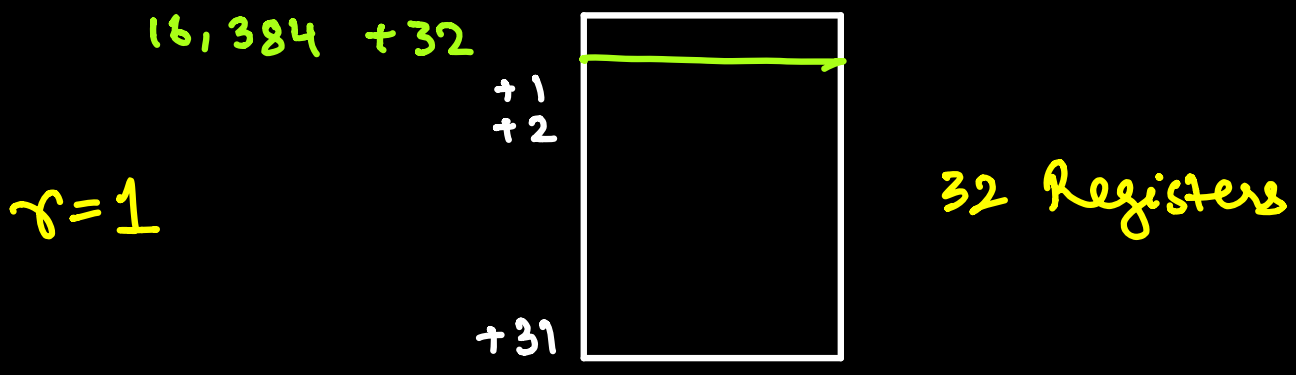
$\equiv 2^8 \times 2^5$

$\equiv 2^{13}$ = 8K Registers (again verified)

$\Rightarrow (0,0) \rightarrow (0,511)$: RAM[16,384] or SCREEN onwards



$\Rightarrow (1,0) \rightarrow (1,511)$: RAM[16,384+32] onwards



$\Rightarrow (r,0) \rightarrow (r,511) \equiv$ RAM[16,384+(32*r)] as starting register

⇒ For any 'c' in a given 'r'

$\left\lfloor \frac{c}{16} \right\rfloor$ refers to the register index within the 32 Registers assigned to given 'r'

Example: $(0, \overset{c}{0}) \rightarrow (0, \overset{c}{15})$ pixels map to register index 0 in RAM[16,384 + 0], ..., RAM[16,384 + 31]

Example: $(0, \overset{c}{16}) \rightarrow (0, \overset{c}{31})$ pixels map to register index 1 in RAM[16,384], RAM[16,384+1], ..., RAM[16,384+31]

Example: $(1, \overset{c}{16}) \rightarrow (1, \overset{c}{31})$ pixels map to register index 1 in RAM[16,384 + 32*1], RAM[16,384+32*1 +1], ..., RAM[16,384+32*1 + 31]

So, for: (r,c) px mapped to

RAM[16,384 + (32*r) + $\left\lfloor \frac{16}{c} \right\rfloor$]

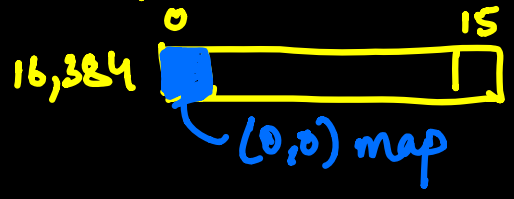
⇒ Remainder of $\frac{c}{16}$ i.e. $(c \% 16)$ specifies which cell position (left to right) corresponds to (r, c) pixel

Example: (0,0) → (0,15)

→ r=0 ≡ RAM[16,384 + 32*0]
≡ RAM[16,384]

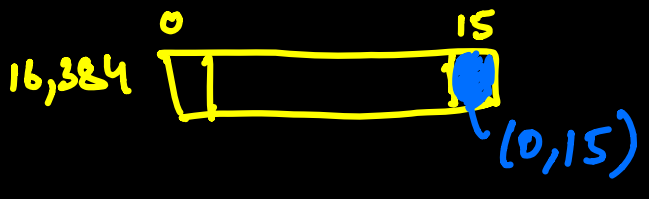
→ c=0 ≡ RAM[16,384 + $\lfloor \frac{0}{16} \rfloor$]
≡ RAM[16,384]

→ c % 16 = 0 % 16 = 0



→ c=15 ≡ RAM[16,384 + $\lfloor \frac{15}{16} \rfloor$]
≡ RAM[16,384]

→ c % 16 = 15 % 16 = 15



(7) Summary of (r,c) px mapping onto Screen Memory

$$(r, c) \equiv RAM\left[16384 + (32 * r) + \left\lfloor \frac{c}{16} \right\rfloor\right]$$

with cell position $(c \% 16)$ from Left to Right
(0) (15)

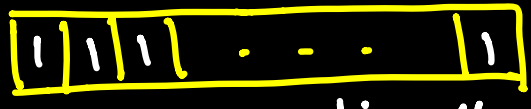
(8) Pixel Coloring

Black (1) or White (0) at single pixel

Example $M = 0 \equiv$ 

16-bit

all corresponding 16 pixels on screen display colored WHITE

example. $M = -1 \equiv$ 

in 2's complement 16-bit

all corresponding 16 pixels on screen display colored BLACK

example:

```
@3
D=A
@SCREEN
M=D
```

≡ storing 3 into RAM[16,384]

RAM[16,384]

will color (0,14) and (0,15) pixels as BLACK

(0,0) → (0,13) will be WHITE