opcode (3)

0 alpha – letter

1 cmp – letter / BLANK is_blank is_cmp

OR –

2 jmp – (11) address is_eq is_ne

jne –

jmp –

3 draw – letter / BLANK is_blank

4 move – signed_amt

5 stop – is_halt

1 1 unconditional

0 0 do nothing

(arrows mean instruction has been converted into parameter of previous instructions)

Feb 6th

bits

instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
alpha	0	0	0	rsvd					letter							
cmp	0	0	1	OR	rsvd			to	letter							
jmp	0	1	0	eq	ne	address (11) (unsigned)										
draw	0	1	1	rsvd				to	letter							
move	1	0	0	signed amount												
stop	1	0	1	h	rsvd											

alpha = enables letters in our alphabet (also boolean table)
takes letter and sets bool in table to true

move = left (or) right (or) left/right <values>
add amount to head of tape

* cont. in 2 pages

```
#include <stdint.h>
uint_16t word = *0x1347;
```

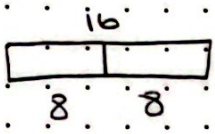
unsigned 16 bit integer data type

(one way to access)

word & 0x00FF

starting position

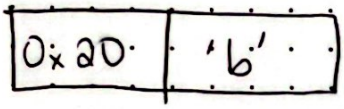
~(~0U << 8) << 0



((uint_8t*) &word)[0]

```
0000 ... 0000
1111 ... 1111
1111 ... 00000000
0000 ... 00111111
```

treating address as array of bytes



our machine

* little endian = least significant address at smallest memory location

big endian = most significant byte of a word at smallest memory location and least significant byte at the largest

terms to study

- bitmask
- endian

~(~0U << k) << Y

'k' 1s in a row

starting at bit Y

alphabet

equals

head

RAM (2k words) 4kB

Program Counter (PC) (address of next instruction)

Instruction Register (IR) (address of current instruction)

Fetch - grabs next instruction from memory.

↳ IR = *PC
PC++

Decode

Execute

* cont from opcode table

stop = check halt bit. If '1', we will do whatever we decide a halt (success) does, reset alpha, equals, etc.

*cmp = determine if 'OR' or not. check blank (b) bit, If we are not 'OR'-ing, set equal flag.

truth table on next page

jmp = check 'eq' and 'ne' bits. Set PC to address accordingly

draw = if drawing blank (b) set is-blank to true

* cmp

OR	ㅂ
0	0
0	1
1	0
1	1

EQ = *H == letter

EQ = *H == ㅂ

EQ |= *H == letter

EQ |= *H == ㅂ => EQ = EQ || (*H == ㅂ)

```
if (alphabet[letter])
  // good
```