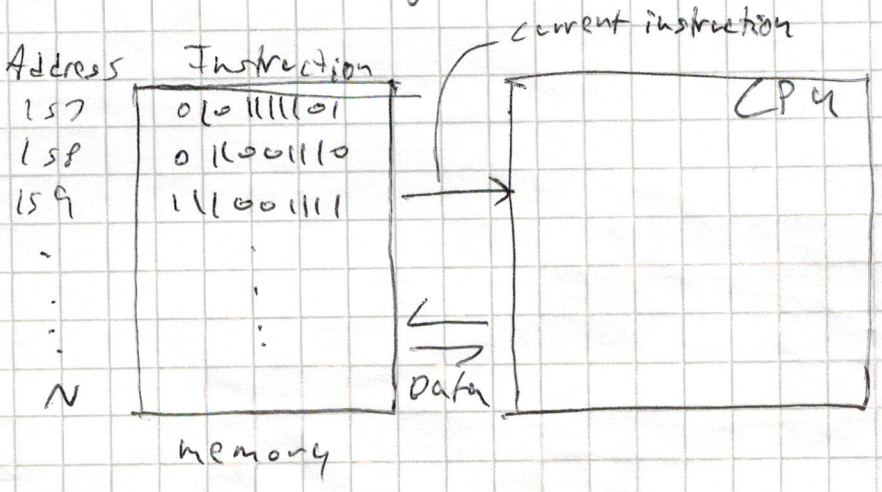
May 14, 2020

Unit 4.1 - Machine Languages: Overview

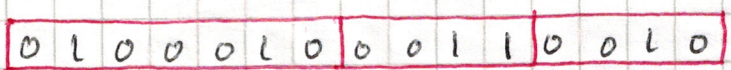
Universality: Same hardware can run many different programs (software programs). Theory of a Universal Turing Machine first proposed by Alan Turing. Put into practice by Von Neumann.



- the binary sequences of instructions is machine language
- Compilers translate high level code into machine language
- Architecture also requires addresses for instructions, & a program counter to select instructions / data

Can use mnemonics to code an instruction into a binary #.

e.g. Instruction:



⇒ Assembly Language

↳
- The collection of mnemonic "symbols" is known as assembly language
- syntactic sugar for machine language, but allows humans to code in machine language if desired
- An "assembler" converts assembly lang into machine lang

May 19, 2020

Unit 4.2 - Machine Languages: Elements

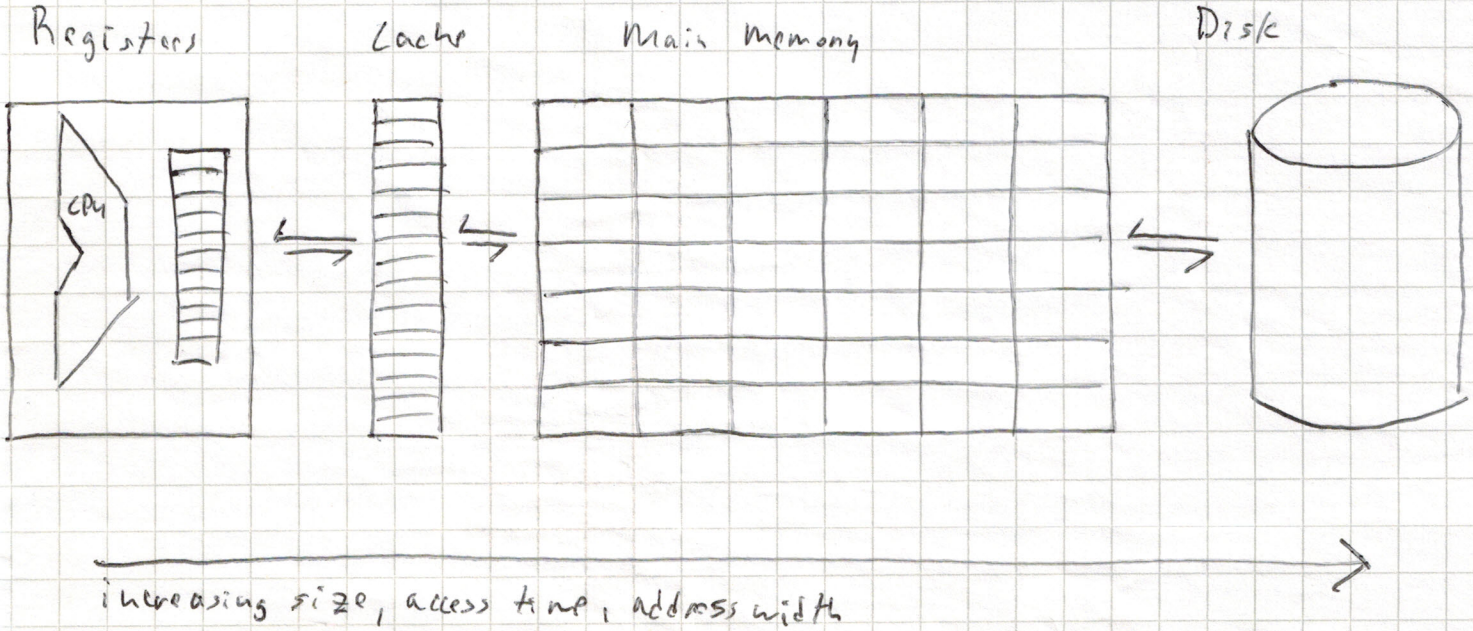
- Machine language is the most important interface in computer science
- specification of the Hardware / Software Interface:
  ↳ supported operations? What do they operate on? How is program controlled?
- Usually in close correspondence to actual Hardware Architecture
- Cost - Performance Trade off
  ↳ silicon area, time to complete instruction

Machine Operations - usually correspond to what is implemented in hardware

- Arithmetic: Add, subtract, ...
- Logical: and, or, ...
- Flow control: "goto instruction X", "if C then goto instruction Y"

Differences between machine languages:
- richness of set of operations (divisions? bulk copy?)
- Data types (width, floating point...)

Issue: How to specify what data to operate on (which address?)
↳ - Accessing memory location is expensive; must supply long address, accessing memory contents takes time
  - solution: memory hierarchy



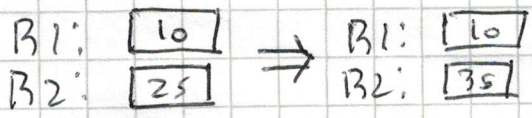
May 14, 2020

Unit 4.2 - Machine Language: Elements (continued)

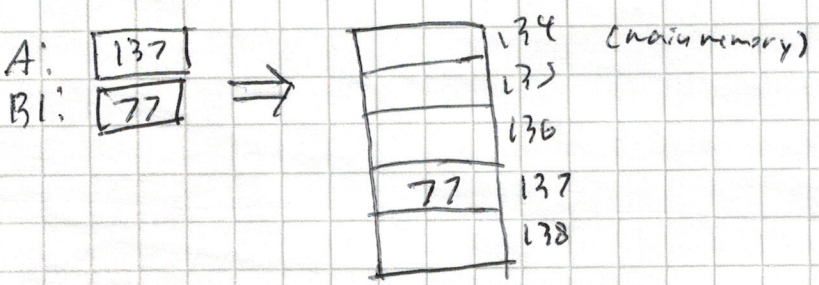
Registers

- CPUs usually contain a few, easily accessed "registers"
- Their number and function are a central part of the machine language

Data Registers
e.g. ADD B1, B2



Address Registers
e.g. Store B1, @A



Addressing Modes

Register
↳ ADD B1, B2 // B2 ← B2 + B1

Direct
↳ ADD B1, m[200] // Mem[200] ← Mem[200] + B1

Indirect
↳ ADD B1, @A // Mem[A] ← Mem[A] + B1

Immediate
ADD 73, B1 // B1 ← B1 + 73

Input/Output

- many types of I/O devices (mouse, keyboard, screen)
- CPU needs some kind of protocol to talk to each → software drivers know these protocols
- One general method uses "memory mapping":
  - ↳ e.g. memory location X holds direction of last movement of mouse
  - ↳ e.g. memory location Y tells printer which paper to use

May 14, 2020

Unit 9.2 – Machine Language: Elements (continued)

Flow Control

- Usually CPU executes machine ~~language~~ instructions in sequence
- Sometimes we want "jump" unconditionally to another location. e.g. executing loops

```
101  Load R1, 0
102  Add 1, R1
103  ...
...  // Do something with R1's value
     ...
156  Jump 102
```

symbolic abstraction:

```
     Load R1, 0
Loop: Add R1, 0
     ...
     // Do something
     ...
     Jump Loop
```

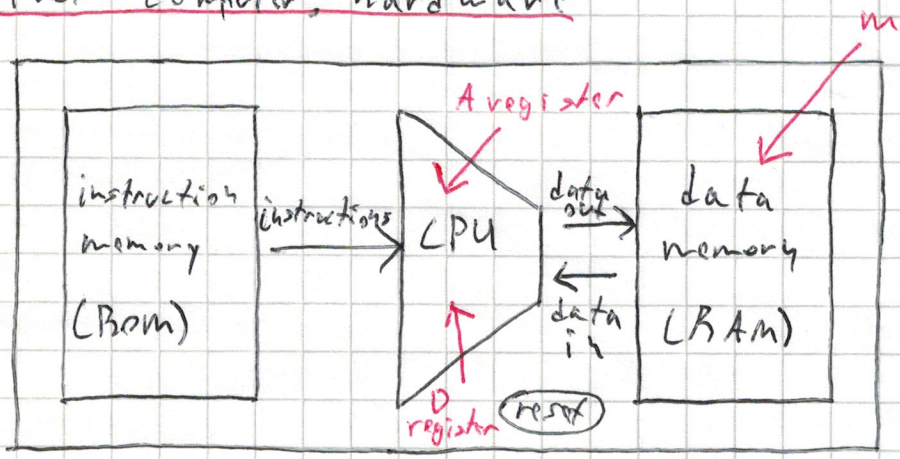
- Sometimes we want jump only if condition is not

```
     JGT R1, 0, Cont    // Jump if R1 > 0 (Jump greater than)
     subtract R1, 0, R1 // R1 ← (0 - R1)
     ...
Cont: // Do something with
      positive R1
```

May 19, 2020

Unit 4.3 - The Hack Computer & Machine Language

Hack Computer: hardware



16 bit machine consisting of:
- Data memory (RAM): a sequence of 16-bit registers
- Instruction memory (ROM): sequence of 16-bit registers
- Central Processing Unit (CPU): perform 16-bit instructions
- Instruction bus / data bus / address bus

Hack Computer: software

- A Hack program = sequence of instructions written in the Hack Machine Lang
  - ↳ 16-bit A-instructions / 16-bit B-instructions

Hack computer: control

- ROM is loaded with a Hack program
- the reset button is pushed
- Program starts running

Hack Computer: registers - hack machine lang recognizes 3 registers:

- D holds a 16-bit value
- A holds a 16-bit value
- M represents the 16-bit RAM register addressed by A

The A-instruction

syntax: @value

- value: either:
  - non-negative decimal constant or
  - symbol referring to such a constant
- semantics:
  - Sets A register to value
  - side effect: RAM[A] becomes the selected RAM register (M register)

e.g @21
↳ set A register to 21
↳ RAM[21] becomes selected RAM register

Usage: set RAM[100] to -1 ⟹ @100, M=-1 (A=100 ⟹ RAM[100] = -1)

May 15, 2020

Unit 4.3 - The Hack Computer & Machine Language (Continued)

The C-instruction

Syntax: dest = comp ; jump

where:

comp is one of several computation directives [0, 1, -1, D, A, M, !D, ... A-D, D&A etc.]

dest = null, M, D, MD, A, AM, AD, AMD (M = RAM[A])

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

semantics:

- compute the value of comp
- store result in dest
- if the Boolean expression (comp jump 0) is true, jumps to execute instruction in ROM[A]

e.g. set D to -1

```
D = -1
```

e.g. set RAM[300] to value of D register minus 1

```
@300
M = D-1
```

e.g. if (D-1 == 0) jump to execute instruction stored in ROM[56]

```
@56     (set A = 56)
D-1; JEQ (if D-1 == 0) go to 56
```

e.g. unconditional jump

```
0; JMP
```

May 13, 2020

Unit 4.4 - Hack Language Specification

2 ways to express the same semantics:

Symbolic:

@17
D+1;JLE

translate →

Binary:

0000000000010001
1110001111110010

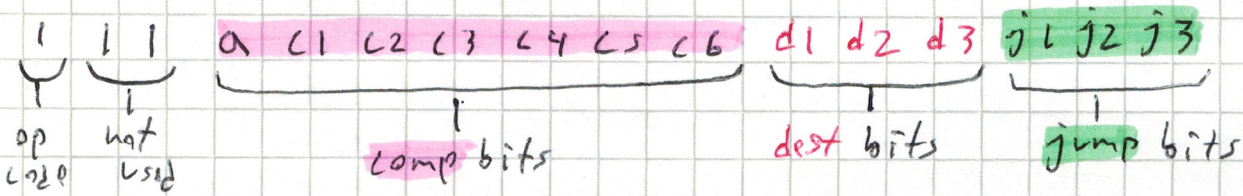
execute →

A-instruction binary syntax: 0 value → where value is 15-bit binary number (or symbol)
op-code
Value ≤ 32767 ($2^{15}-1$)

e.g. 0000000000010101

C-instruction dest = comp ; jump

binary syntax:

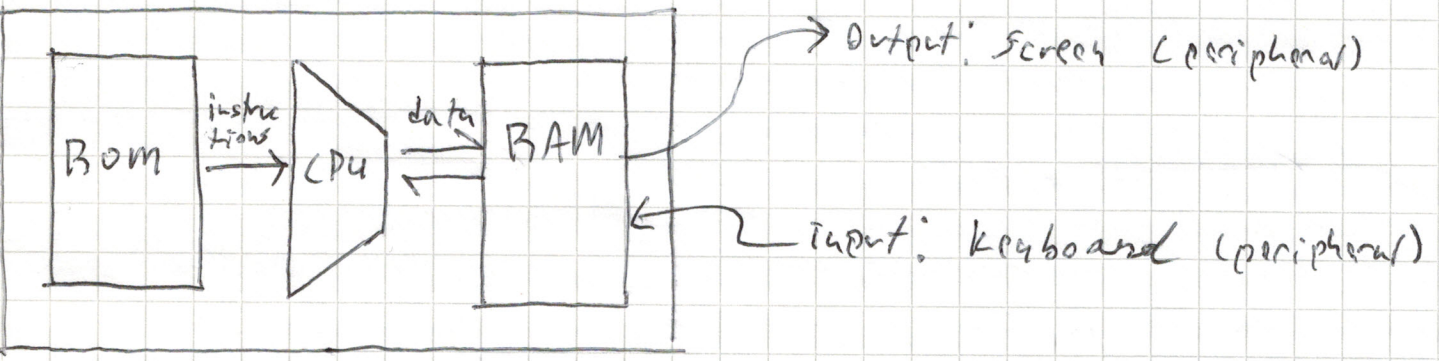


- can build a table to map comp bit combinations to different computations
- same for destinations & jump bits
- The mappings create the language specification

May 15, 2020

Unit 4.5 - Input/Output

Extending the Hack computer platform:

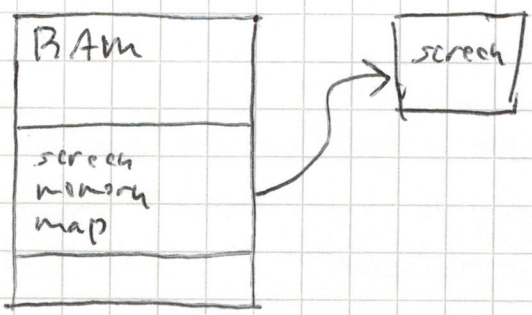


I/O devices used for:
- Getting data from users
- displaying data to users

High-level Approach: (Part 2 of course)
- sophisticated software libraries enabling text, graphics, animation, audio, video, etc to control I/O

Low-level Approach (Part 1)
- use bits to control I/O

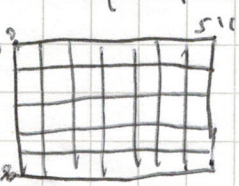
Output



Screen memory map

- A designated memory area, dedicated to manage a display unit
- The physical display is continuously refreshed from the memory map, many times per second
- output is affected by writing code that manipulates the screen memory map

- Display unit for Hack computer is a table: 256 rows x 512 columns



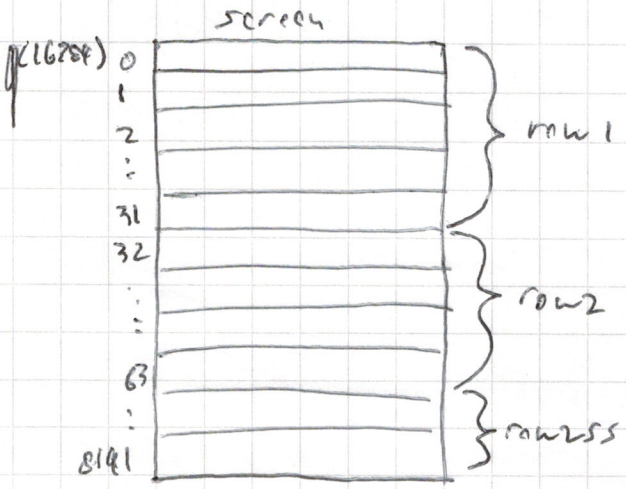
- each intersection is a pixel, pixel may be on/off

Unit 4.5 - Input/Output (continued)

Controlling the Hack Display using screen memory map:

- the memory map is 8192 16-bit registers (words) in the RAM
- individual bits (not words) control output of each pixel on display

proof: $256 \times 512 = 131,072$ px $\Rightarrow \frac{131,072}{16} = 8192$



- each row is 32 registers (32×16 = 512)
- To set pixel(row,col) on/off:

1. i = 32*row + col/16 (integer division, drop remainder)

   or word = screen[32*row + col/16]

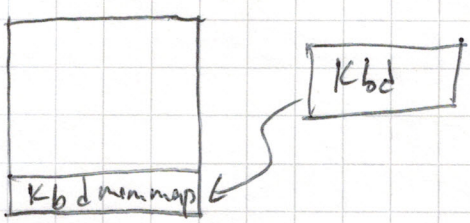
   ↪ word = RAM[16384 + 32*row + col/16]

   (because screen is a chip inside RAM, begins at register 16384)

2. Set the (col%16)th bit of word to 0 or 1

3. commit word to RAM

Input



Keyboard memory map

- A single 16-bit register called "Keyboard"
- When a key is pressed, the key's scan code appears in the keyboard memory map
  e.g. K = 75, 4 = 52, space = 32
- can create a Hack character set of key-code mappings
- when no key is pressed, the resulting code is 0

To check which key is pressed:

- probe contents of keyboard chip → RAM[24576]

May 17, 2020

Unit 4.6 - Hack Programming (Part 1)

The Hack programming language involves:
- Working with registers & memory
- Branching
- Variables
- Iterations
- Pointers
- Input/output

Working with Registers and Memory - "bread & butter of low level program

review: D = data register, A = address/data register,
M = currently selected memory register, M = RAM[A]

Example: add two numbers

```
// Program: Add2.asm
// Computes: RAM[2] = RAM[0] + RAM[1]
// Usage: put values in RAM[0], RAM[1]

0  @0
1  D=M      // D = RAM[0]

2  @1
3  D=D+M    // D = D + RAM[1]   (D = RAM[0] + RAM[1])

4  @2
5  M=D      // RAM[2] = D
```

memory (Rom)

0	@0	→ 0000000....
1	D=M	1111....
2	@1	0000.....1
3	D=D+M	1111.....
4	@2	0000.....10
5	M=D	111.....1000
:		
32767		

- instructions in Hack ~~Assembly~~ asmbly Lang. Program have implicit line numbers
- white space/comments are ignored
- symbols translate into binary
- Best Practice:
  - to terminate a program safely, end it with an infinite loop.

```
6  @6
7  0;JMP
```

→ else will go on to rest of Rom (insecure)

see "napslide" ←

May 17, 2020

Unit 4.6 - Hack Programming (Part 1) (Continued)

Built-in Symbols

The Hack assembly language features built-in symbols:

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576

- sequence of 15 labels that act as "virtual registers" (R0 - R15)

instead of:	better style:
// RAM[5] = 15	// RAM[5] = 15
@15	@15
D = A	D = A
@5	@R5
M = D	M = D

because more clear that you require address of RAM[5].

note: Hack is case sensitive. R5 ≠ r5

other symbols used in P2 II of nand2tetris:

Symbol	Value
SP	0
LCL	1
ARG	2
THIS	3
THAT	4

May 10, 2020

Unit 4.7 - Hack Programming, Part 2

Branching

Example 1

```
// Program: Sign.asm
// Computes: if R0 > 0
//               R1 = 1
//           else
//               R1 = 0
0  @R0
1  D=M

2  @8
3  D;JGT

4  @R1
5  M=0
6  @10
7  0;JMP

8  @R1
9  M=1

10 @10
11 0;JMP
```

Example 2 - Using symbolic references

```
0  @R0
1  D=M

2  @POSITIVE      // using label
3  D;JGT

4  @R1
5  M=0
6  @END
7  0;JMP

(POSITIVE)        // declare label
8  @R1
9  M=1

(END)
10 @END
11 0;JMP
```

* @LABEL translates to @n, where n is the instruction following the (LABEL) declaration

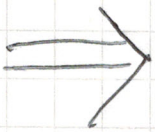
May 10, 2020

Unit 4.7 - Hack Programming, Part 2 (Continued)

Variables

Example

```
// Program: Flip.asm
// Flips values of
// RAM[0] and RAM[1]

// temp = R1
// R1 = R0
// R0 = temp

   @R1
   D=M
   @temp
   M=D

   @R0
   D=M
   @R1
   M=D

   @temp
   D=M
   @R0
   M=D

(END)
   @END
   0;JMP
```



Memory (ROM)

```
0  @1
1  D=M
2  @16
3  M=D
4  @0
5  D=M
6  @1
7  M=D
8  @16
9  D=M
10 @0
11 M=D
12 @12
13 0;JMP
14
15
...
```

32767

Contract:

- reference to symbol with no corresponding label declaration is treated as reference to a variable
- variables allocated to the RAM from address 16 onward

*@temp: "find some available memory register (say register n), and use it to represent the variable temp. So, from now on, each occurrence of @temp in program will translate to @n.

Benefits:

- easier to read & debug
- can create symbolic program that can be placed anywhere in memory

May 18, 2020

Unit 4.7 - Hack Programming, Part 2 (continued)

Iterative Processing - Example

Pseudo Code

```
// Computes RAM[1] = 1+2+... +RAM[0]

   n = R0
   i = 1
   sum = 0
LOOP:
   if i > n, goto STOP
   sum = sum + i
   i = i+1
   goto LOOP
STOP:
   R1 = sum
```

Assembly code

```
// Program: Sum1toN.asm
// Computes RAM[1] = 1+2+... +n
// Usage: put a number (n) in RAM[0]

   @R0
   D=M
   @n
   M=D
   @i
   M=1
   @sum
   M=0
(LOOP)
   @i
   D=M
   @n
   D=D-M
   @STOP
   D;JGT

   @sum
   D=M
   @i
   D=D+M
   @sum
   M=D          → sum = sum + i
   @i
   M=M+1        → i = i + 1
   @LOOP
   0;JMP

(STOP)
   @sum
   D=M
   @R1
   M=D

(END)
   @END
   0;JMP
```

Best Practices:

1. Design program in pseudocode
2. Write in assembly lang
3. Test program (on paper) using variable-value trace table

May 10, 2020

Unit 4.8 - Hack Programming, Part 3

Pointers - example

```
// for (i=0; i<n; i++){
     arr[i] = -1
   }
// suppose that arr = 100 and n = 10
//
// arr = 100
@100
D=A
@arr
M=D

// n = 10
@10
D=A
@n
M=D

// initialize i=0
@i
M=0

(LOOP)
  // if (i==n) goto END
  @i
  D=M
  @n
  D=D-M
  @END
  D; JEQ

// RAM[arr+i] = -1
@arr
D=M
@i
A=D+M
M=-1

  // i++
  @i
  M=M+1

  @LOOP
  0; JMP

(END)
  @END
  0; JMP
```

RAM

	0	
	...	
arr:	16	100
n:	17	10
i:	18	0
	100	
	101	
	102	

3 iterations ⟹

RAM

0	
...	
16	100
17	10
18	2
...	
100	-1
101	-1
102	-1

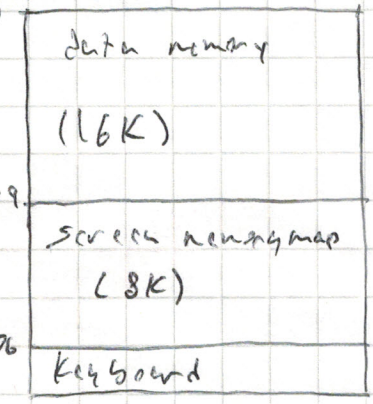
- Variables that store memory addresses, like arr & i, are called pointers
- Hack Pointer Logic: whenever we have to access memory using a pointer, we need an instruction like A=M
- Typical pointer semantics: "set the address register to the contents of some memory register"

May 18, 2020

Unit 4.8 - Hack Programming, Part 3 (continued)

Input/Output

review: Hack RAM



SCREEN: base address of screen memory map
KBD: address of keyboard memory map

Handling Input

To check which key is currently pressed:
- read contents of RAM[24,576] (addr KBD)
- if register contains 0, no key pressed
- otherwise, register contains scan code of currently pressed key

---

Example - draw rectangle of width 16 & length n on screen

Pseudo code

```
// for (i=0; i<n; i++) {
//     draw 16 black pixels at
//     beginning of row i
// }

addr = SCREEN
n = RAM[0]
i = 0

LOOP:
   if i>n goto END
   RAM[addr] = -1 // 1111111111111111
   // advance to next row
   addr = addr +32
   i = i +1
   goto LOOP

END:
   goto END
```

screen: