



Today's Agenda

- Review of Hack Assembly Programs
- Pointers and Arrays
- Input / Output Instructions
- Debugging





Pointers and Arrays



Pointers and Arrays: Example

```
// for (i=0; i<n; i++)  
//     arr[i] = -1;
```

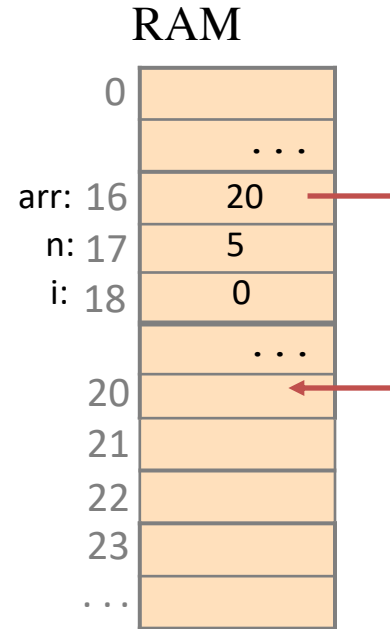
Observations:

- Variables that store memory addresses like **arr** in this example are called pointers
- Abstraction of arrays exist only in high level languages. In machine language there is no abstraction of arrays. Rather array is a segment of memory of which we know the base address of this segment and the length of the array that programmer has declared
- Arrays are implemented as a block of memory registers and in order to access these memory registers one after the other, we need a variable that holds the current address
- There is nothing special about pointer variables, except that their values are interpreted as addresses



Pointers and Arrays: Example

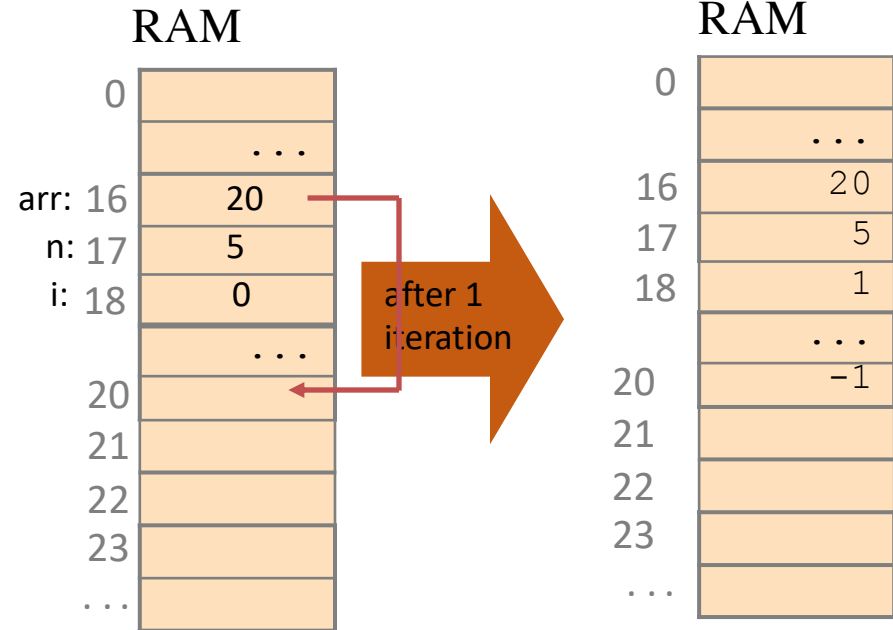
```
// for (i=0; i<n; i++)  
//     arr[i] = -1;  
// Let us initialize arr=20, n=5, i=0  
  
// arr = 20  
@20  
D=A  
@arr  
M=D  
  
// n = 5  
@5  
D=A  
@n  
M=D  
  
// i = 0  
@i  
M=0  
  
// Loop code continues on next slide...  
      (LOOP)
```





Pointers and Arrays: Example

```
// Code continues from previous slide
(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
```

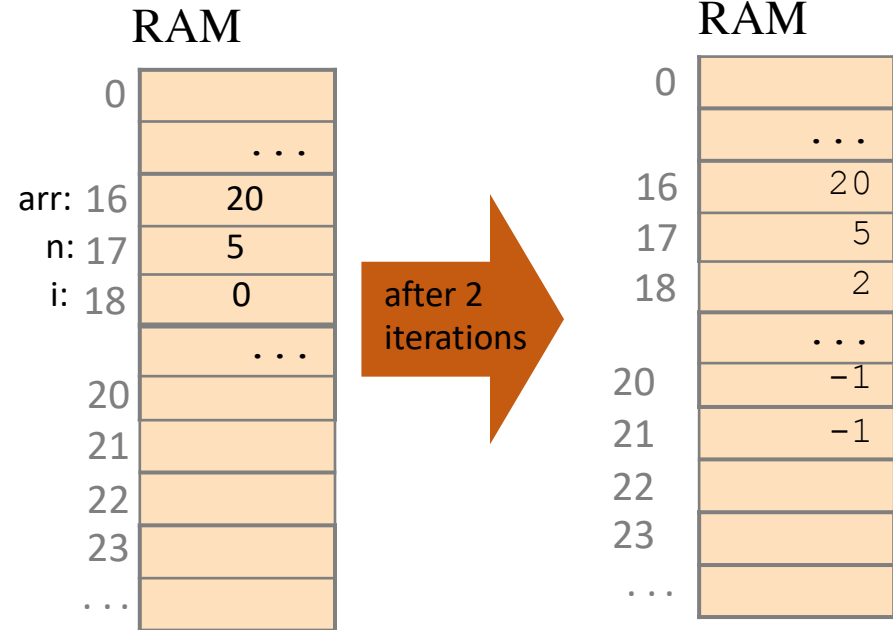


- Pointers in Hack: Whenever we have to access memory using a pointer, we need an instruction like $A=expression$
- Typical Pointer Semantics: Set the address register to the contents of some memory register



Pointers and Arrays: Example

```
// Code continues from previous slide
(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
```

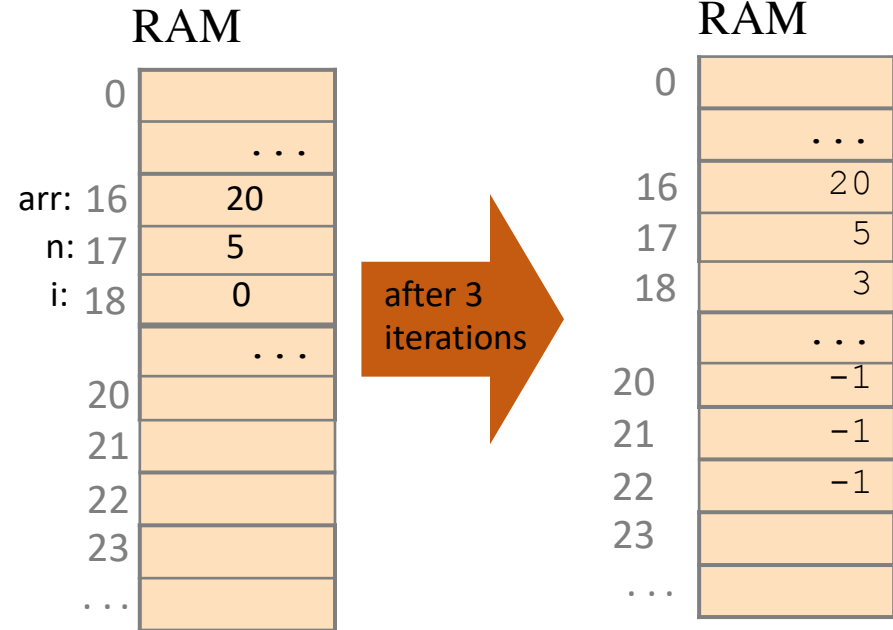


- Pointers in Hack: Whenever we have to access memory using a pointer, we need an instruction like $A=expression$
- Typical Pointer Semantics: Set the address register to the contents of some memory register



Pointers and Arrays: Example

```
// Code continues from previous slide
(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
```



- Pointers in Hack: Whenever we have to access memory using a pointer, we need an instruction like $A = \text{expression}$
- Typical Pointer Semantics: Set the address register to the contents of some memory register



Manipulating Arrays using Pointers





Input & Output



I/O Devices: Screen And Keyboard

Simulated screen: 256 columns by 512 rows, black & white memory-mapped device. The pixels are continuously refreshed from respective bits in an 8K memory-map, located at RAM[16384] - RAM[24575].

Simulated keyboard:
One click on this button causes the CPU emulator to intercept all the keys subsequently pressed on the real computer's keyboard; another click disengages the real keyboard from the emulator.

ROM	Asm
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	

RAM	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

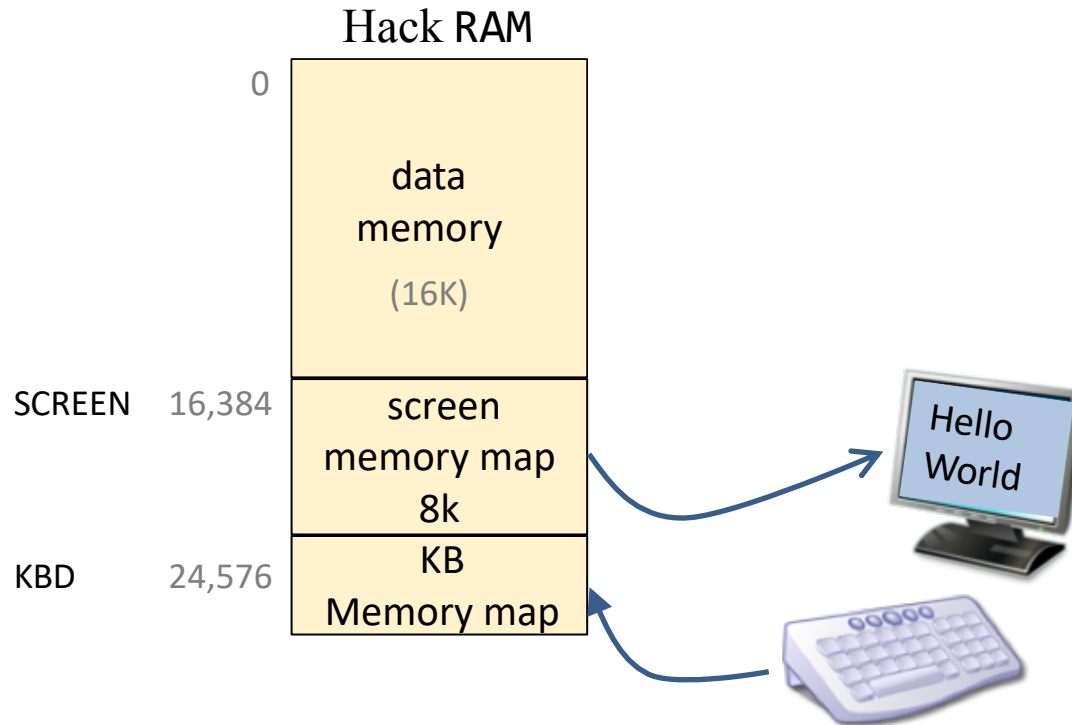
D: 0

ALU
D Input: 0
M/A Input: 0
ALU output: 0

Script restarted



Memory Map of Screen and Keyboard



Hack language convention:

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



Screen Memory Map

Memory Map Screen (chip)

(16384)

A sequence of 8K x 16 bit words
8192 words
131072 bits

row 0 16 x 32 = 512

row 1

row 255



Black & White Display Unit

- A matrix of 256 rows x 512 columns
- 131072 pixels



To set pixel (row,col) on/off

`word = Screen[32*row + col/16]`

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

Set $(col\%16)^{th}$ bit of word to 0 or 1

`RAM[i] = word`



Handling The Keyboard

Hack RAM

24576

0000 000001001011



Scan-code of ' k ' = 75

To check which key is currently pressed:

- Read the contents of RAM[24576] (address KBD)
- If the register contains 0, no key is pressed
- Otherwise, the register contains the scan code of the currently pressed key



Drawing a Rectangle on The Screen

File View Run Help

Slow Fast Animate: Program flow View: Scr... Format: D...

ROM	RAM
0 @0	0 50
1 D=M	1 0
2 @16	2 0
3 M=D	3 0
4 @17	4 0
5 M=0	5 0
6 @16384	6 0
7 D=A	7 0
8 @18	8 0
9 M=D	9 0
10 @17	10 0
11 D=M	11 0
12 @16	12 0
13 D=D-M	13 0
14 @27	14 0
15 D;JGT	15 0
16 @18	16 50
17 A=M	17 51
18 M=-1	18 18016
19 @17	19 0
20 M=M+1	20 0
21 @32	21 0
22 D=A	22 0
23 @18	23 0
24 M=D+M	24 0
25 @10	25 0
26 0;JMP	26 0
27 @27	27 0
28 0;JMP	28 0

Code

RAM

50 pixels long

16 pixels wide

Screen

Task: draw a filled rectangle at the upper left corner of the screen, 16 pixels wide and RAM[0] pixels long

PC 27 A 27



Drawing a Rectangle on The Screen

```
/* Program: Rectangle.asm
```

```
Draws a filled rectangle at the screen's  
top left corner, with width of 16 pixels  
and height of RAM[0] pixels.
```

```
Usage: put a non-negative number  
(rectangle's height) in RAM[0] */
```

```
@R0  
D=M  
@n  
M=D // n = RAM[0]  
@i  
M=0 // i = 0  
@SCREEN  
D=A  
@addr  
M=D // addr = 16384 (screen's base  
address)
```

```
(LOOP)
```

```
// ...
```

```
//...
```

```
(LOOP)
```

```
@i
```

```
D=M
```

```
@n
```

```
D=D-M
```

```
@END
```

```
D;JGT // if i>n goto END
```

```
@addr
```

```
A=M
```

```
M=-1 //RAM[addr]=1111111111111111
```

```
@i
```

```
M=M+1 // i = i + 1
```

```
@32
```

```
D=A
```

```
@addr
```

```
M=D+M // addr = addr + 32
```

```
@LOOP
```

```
0;JMP // goto LOOP
```

```
(END)
```

```
@END // program's end
```

```
0;JMP // infinite loop
```



Drawing a Rectangle on The Screen





An Interactive Program



Fill: A Simple Interactive Program

CPU Emulator (2.5) - /Users/arif/Documents/01 Arif-CS223-COAL/LectureSlides-Video Sessions/Lecture Codes/21/Fill.asm

File View Run Help

Slow Fast Animate: No animation View: Scr... Format: D...

ROM	
14	@20
15	0; JMP
16	@1
17	M=0
18	@20
19	0; JMP
20	@1
21	D=M
22	@0
23	A=M
24	M=D
25	@0
26	D=M+1
27	@24576
28	D=A-D
29	@0
30	M=M+1
31	A=M
32	@20
33	D; JGT
34	@0
35	0; JMP
36	
37	
38	
39	
40	
41	
42	

RAM	
0	18432
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	20
17	5
18	5
19	0
20	-1
21	-1
22	-1
23	-1
24	-1
25	0
26	0
27	0
28	0

PC 20 A 20

D 6144

ALU
D Input : 6144
M/A Input : 20

Running...

Select No animation

Listen to the keyboard

No key is pressed so all pixels of screen are white



Fill: A Simple Interactive Program

CPU Emulator (2.5) - /Users/arif/Documents/01 Arif-CS223-COAL/LectureSlides-Video Sessions/Lecture Codes/21/Fill.asm

File View Run Help

Slow Fast Animate: No animation View: Scr... Format: D...

ROM	
14	@20
15	0;JMP
16	@1
17	M=0
18	@20
19	0;JMP
20	@1
21	D=M
22	@0
23	A=M
24	M=D
25	@0
26	D=M+1
27	@24576
28	D=A-D
29	@0
30	M=M+1
31	A=M
32	@20
33	D;JGT
34	@0
35	0;JMP
36	
37	
38	
39	
40	
41	
42	

RAM		
0	18432	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	
16	20	
17	5	
18	5	
19	0	
20	-1	
21	-1	
22	-1	
23	-1	
24	0	
25	0	
26	0	
27	0	
28	0	

PC 20 A 20

D 6144

ALU
D Input : 6144
M/A Input : 20

Running...

When any key is pressed all pixels of screen becomes black



Fill: A Simple Interactive Program





Debugging



Breakpoints: A Powerful Debugging Tool

The CPU emulator continuously keeps track of:

- **A:** value of the A register
- **D:** value of the D register
- **PC:** value of the Program Counter
- **RAM[i]:** value of any RAM location
- **time:** number of elapsed machine cycles

Breakpoints:

- A breakpoint is a pair <variable, value> where variable is one of {A, D, PC, RAM[i], time} and i is between 0 and 32K.
- Breakpoints can be declared either interactively, or via script commands.
- For each declared breakpoint, when the variable reaches the value, the emulator pauses the program's execution with a proper message.



Breakpoints Declaration

CPU Emulator - D:\hack\instructor\Examples\rect\rect.bin

File View Run Help

Slow Fast Animate: Program flow View: None Format: Decimal

1. Open the breakpoints panel

2. Previously-declared breakpoints

3. Add, delete, or update breakpoints

Variable Name	Value
A	2
RAM[20]	5
Time	12

ALU

D Input : 0

M/A Input : 0

ALU output : 0

D 0

PC 0

A



Breakpoints Declaration

CPU Emulator - D:\hack\instructor\Examples\rect\rect.bin

File View Run Help

Slow Fast Animate: Program flow View: None Format: Decimal

ROM Asm

0	@0
1	D=M
2	@23
3	D; JLE
4	@16
5	M=D
6	@16384
7	D=A
8	@17
9	M=D
10	@17
11	A=M
12	M=-1
13	@17
14	D=M
15	@32
16	D=D+A
17	@17
18	M=D
19	@16
20	MD=M-1
21	@10
22	D; JGT
23	@23
24	0; JMP
25	
26	
27	
28	

RAM

0	50
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	

PC 0

A

Breakpoint Panel

Variable Name	Value
A	2
RAM[20]	5
Time	12

Breakpoint Variables

Name: RAM[21] RAM[]

Value: 200

ALU output: 0



Breakpoints Usage

The screenshot shows the CPU Emulator interface with the following components:

- ROM Asm:** A list of assembly instructions from address 0 to 28. Instruction 1 is highlighted in yellow.
- RAM:** A memory dump showing addresses 0 to 28. Address 20 contains the value 5, and address 21 contains the value 200.
- Breakpoint Panel:** A window titled "Breakpoint Panel" containing a table of variables and their values.

Variable Name	Value
A	2
RAM[20]	5
Time	12
RAM[21]	200

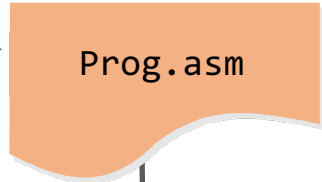
Callouts and instructions:

1. New breakpoint (points to the Breakpoint Panel)
2. Run the program (points to the ROM Asm window)
3. When the A register will be 2, or RAM[20] will be 5, or 12 time units (cycles) will elapse, or RAM[21] will be 200, the emulator will pause the program's execution with an appropriate message.

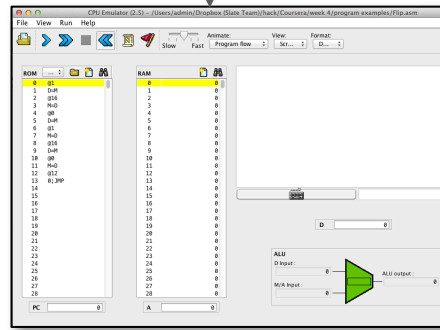
A powerful debugging tool!



Program Development Process



Write / edit the program using a text editor



Load the program into the CPU Emulator, and run it

Find and fix the errors



Yes



No



Best Practice

Well-written low-level code is

- Short
- Efficient
- Elegant
- Self-describing

Technical tips

- Use symbolic variables and labels
- Use sensible variable and label names
- Variables: lower-case
- Labels: upper-case
- Use indentation
- Start with pseudo code



Things To Do

- Download all the assembly program from the course bitbucket repository

<https://bitbucket.org/arifpucit/coal/>

make changes to them and execute them in the CPU Emulator

- Run the programs, one instruction at a time, do the working in your head or on a piece of paper, while executing the programs one instruction at a time



Coming to office hours does NOT mean you are academically week!