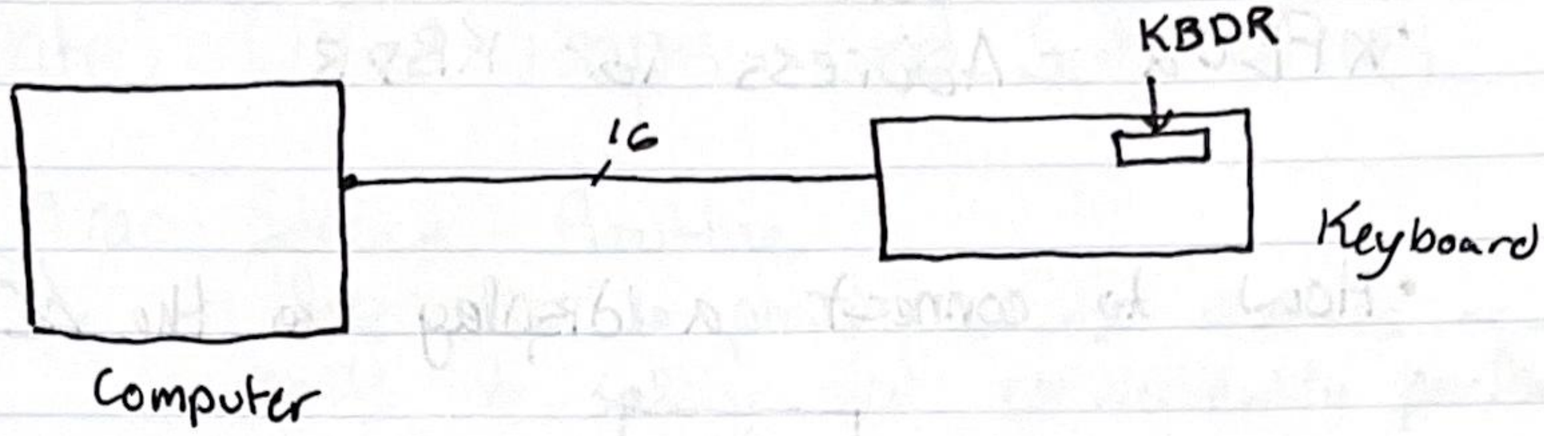
Lecture

Jan 17, 2023



- Nowadays, we use the same computer instruction as I/O devices

- All I/O devices are controlled by instructions within a computer's ISA
  - In modern devices, we do not use seperate instructions

- LC3 Memory: Memory mapped device registers

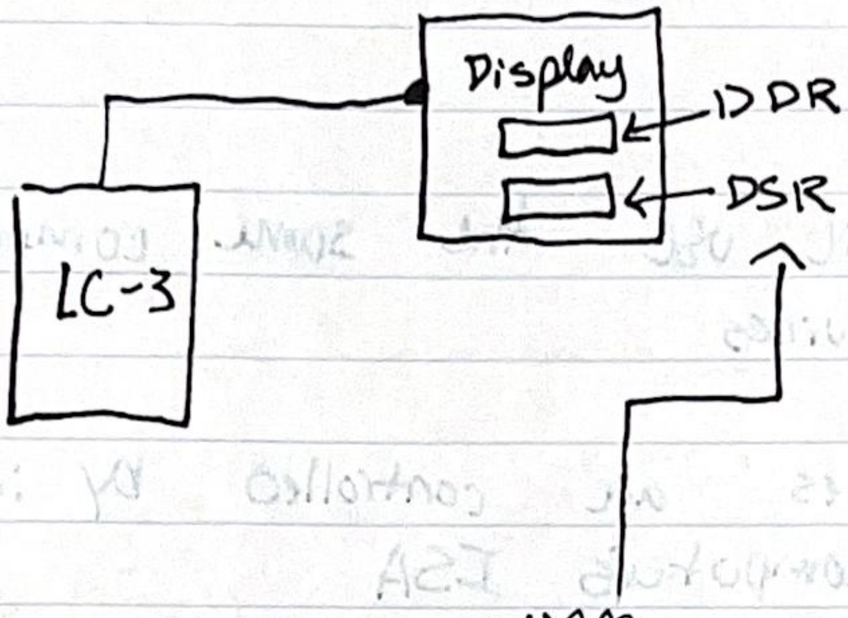
  x3000 - xFDEE is stored for user

  xFE00 - xFFFF reserved for I/O devices

- Handshaking using KBDR and KBSR
  - When char is typed:
    1. Ascii code placed in KBDR[0:7]
    2. KBSR[15] set to zero automatically

- xFE00 - Address for KBSR
- xFE02 - Address for KBDR

- How to connect a display to the LC-3



- When monitor is ready to display new char:
  DSR[15] set to 1

- When new char is written

Uses MSB to determine if device is ready for next instruction

A "1" means it is ready

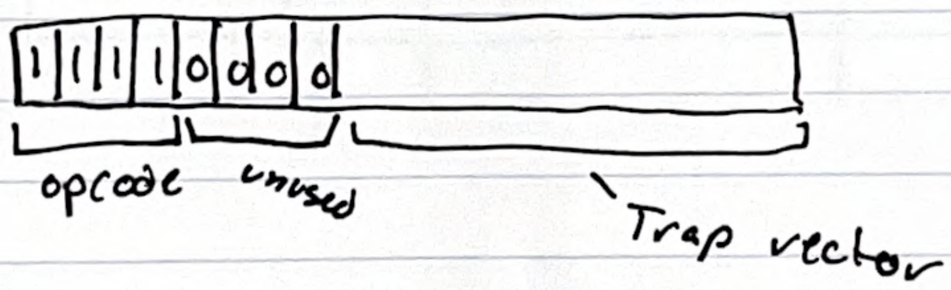
Lecture 2 — Jan 19, 2023

Repeated Code: Traps and Subroutines

Trap Service Routine

- Desirable to provide service routines or system calls to safely and conveniently perform low-level, <u>privileged</u> operations
  - User program invokes system call
  - Operating system code performs operation
  - Returns control to user program

TRAP x20 → Indirect Reference (8 bits)



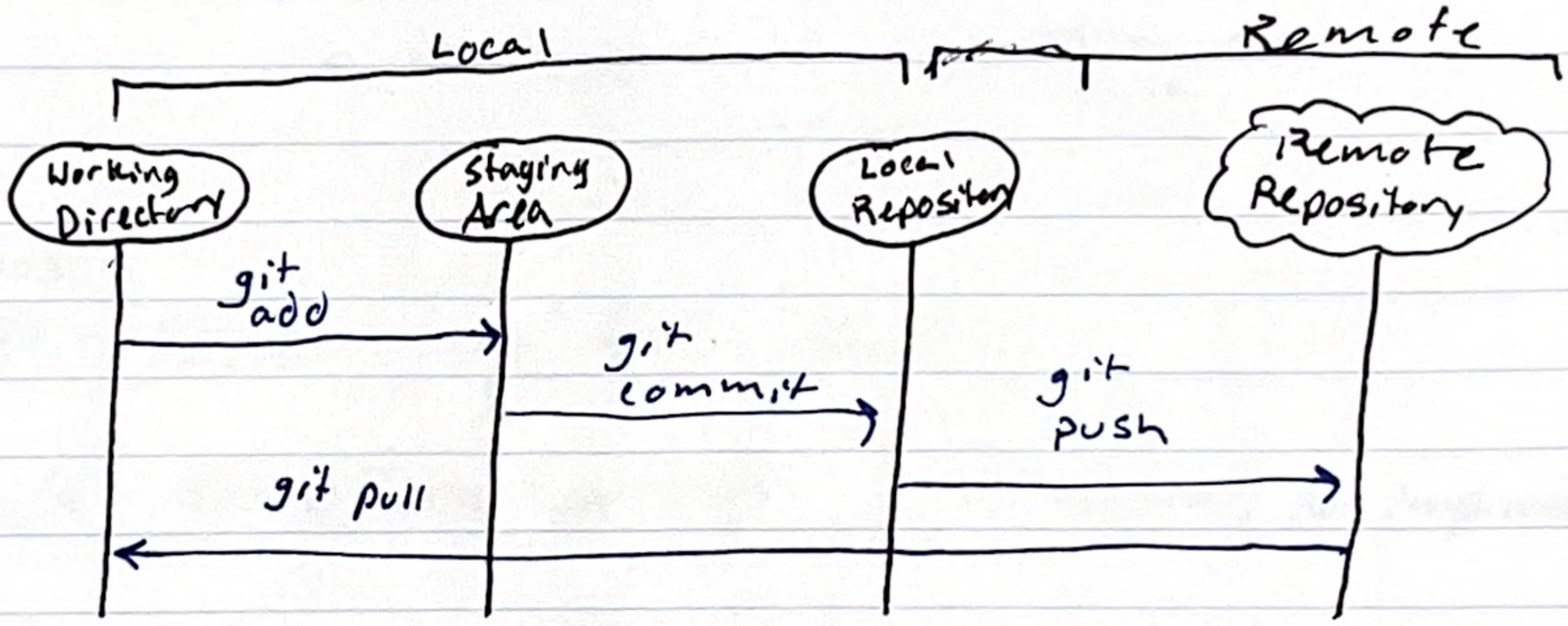
Lab 1

Jan 20, 2023

Recap: 1. Write programs using the LC-3 that use memory

What is git?

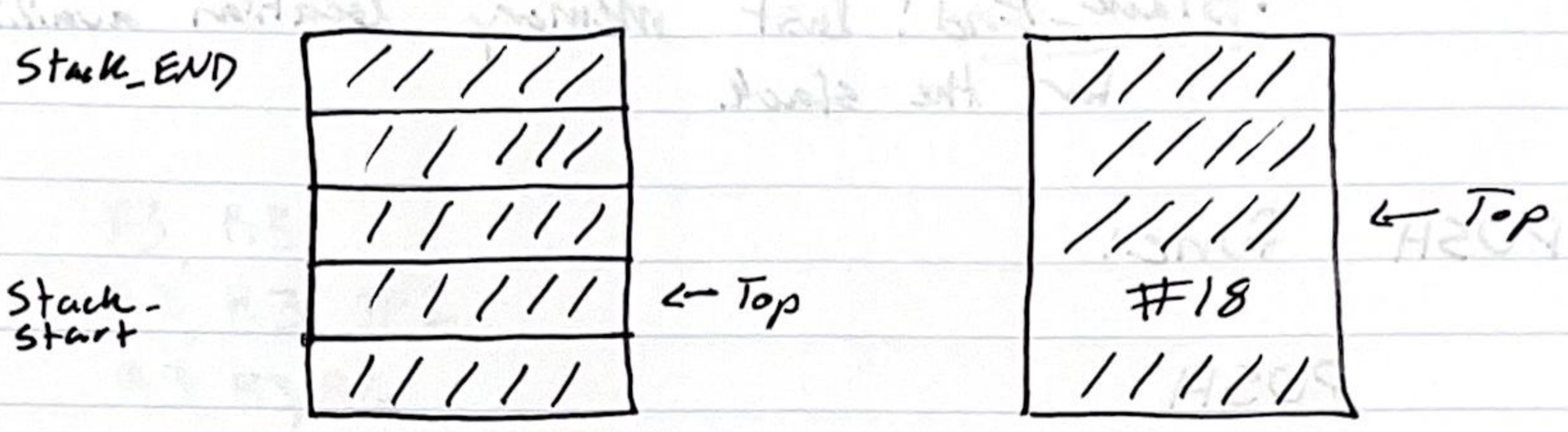
- Version control
- Widely used
- We will use it to distribute MPs



Lecture 3

Jan 24, 2023

R0 contains the data we want to stack



Definitions:

POP - Remove item from stack

Stack - Abstract Data Type (ADT). It is behavior, not implimentation

TOS - always points to next avaible location

PUSH: Add item to the stack

Must check for <u>overflows</u>:
- Check <u>before</u> adding data to stack
- Return status code to R5 (0: success 1: overflow)

Must check for <u>underflow</u>:
- Check before removing data from stack
- Return status code to R5 (0: success 1: underflow)

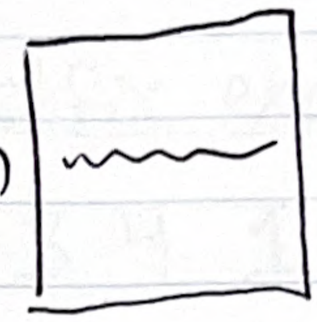
We label two memory locations:

- Stack_Start: first memory location available for the stack
- Stack_End: last memory location available for the stack

PUSH func:

```
PUSH
;
;prepare registers/callee save
    ST R3, PUSH_Save R3
    ...
```

```
NOT   R3  R3
ADD   R3, R3, #2        17
ADD   R3, R3, R6        18  Stack_END
```




```
NOT   R3  R3
ADD   R3, R3, #1
ADD   R3, R3, R6
```

Lecture 4

Jan 26, 2023

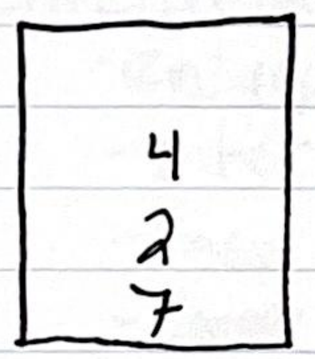
prefix expression	postfix expression
3*(4+1)	3 4 1 + * =
3/(3+4)	3 3 4 + / =

MP2 - Part 1: Postfix Expression & Stack

Numeber = 1 push

opperater = 2 pop > calculate > 1 push
(1 pop)

Ex



7 2 4 + - =

pop

add them

then push result back

then pop

subtract

Lecture 5 Jan 31, 2023

C implimentation

- Developed in 1970s to simplify coding

- Give symbolic names to values
  - Don't need to know when register or memory location

- Provides expressiveness
  - use symbols to convey meaning
  - simple expressions for common control patterns

- Provides abstraction of underlying hardware

Ways of translating high-level languages

- Interpretation
  - ~~Interpretator~~
  - interpreter: program that executes program statements
  - Easy to debug, make changes
  - longer to execute

- Compiler

Compiler

- Source Code Analysis
-
-

Variable declarations:
- must have a type (int, char, float, etc)
- used as names for data items

Must include <stdio.h> to use I/O functions

```
printf("%d\n", counter);
```

↑ decimal int ↑ line feed ↖ variable used

%d : decimal int
%x : hexadecimal int
%c : ASCII character
%f : floating point number

Lecture 6

Expressions

a | b	binary logical or
a || b	boolean
a & b	logical and
a && b	boolean
! (a + b)	boolean
a % b	logical
b / a	logical
a == b	boolean
a = b	logical (set a to b)
a = b = 5	logical (set a and b to 5)

++a + b--

↑ updated After assignment

↑ assigned during assignment

scanf("%d", &startpoint);

↑ points to address at startpoint

↑ must use ampersand if variable is being modified

```
if(x<=10) {
    y= x*x +5;
    z= y-2;}
```

skips all if false

<=>

```
if(x<=10)
    y= x*x +5;
z= y-2;
```

skips this if false

Lecture 8

- C-Compiler makes a symbol table

- Symbol table contains:
  1. name
  2. type
  3. location
  4. scope

- C passes arrays by refference
  - the address of the array (address of the first element) is written to the function's activation record

ex:

```
int main(){
    int
    int rnd_number[3];

    roll_a_dice(
```

```
void roll_a_dice(int* one, int* two, int* three

    *one = (rand()%6)+1;
    *two = (rand()%6)+1;
    *three = (rand()%6)+1;
```

points to memory location of 'one'

ex:

```
char temp[100] = "abcdef";
temp[3] = '\0';
printf("%s", temp);
```

} prints "abc"

```
Char temp[100] = "abcdef"
printf("%s", &temp[2]);
```

} prints "cdef"

Lecture

A text stream is a sequence of ASCII char

ex:

- Char printed to the monitor by a single program
- char entered by user during a single program
- Char in a single file

Redirect all error messages to stderr

Redirect all regular outputs to stdout

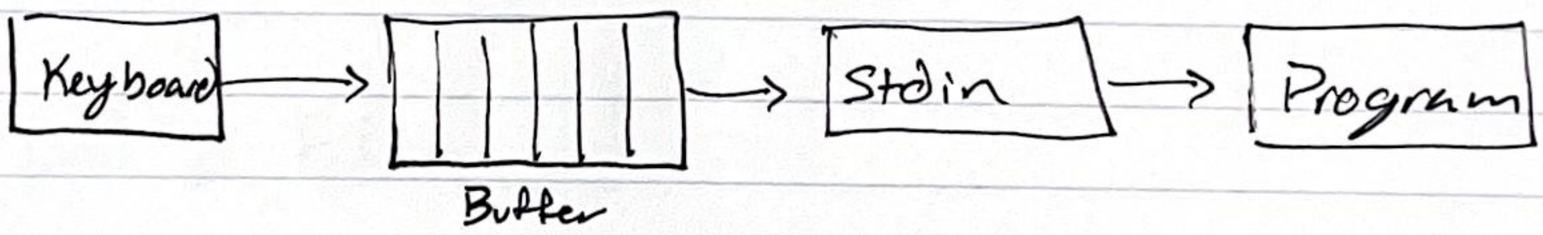
- Ex:

```
fprintf(stdout, "Normal output1\n");

fprintf(stderr, "Warning\n");
```

Input Buffering:

The buffer is released when the user presses Enter



Each file is associated with a stream.

- input stream
- output stream

To read or write a file, we declare a file pointer

```
FILE *infile
```

Ex:

```
FILE *myfile;

myfile = fopen("test.txt", "w");
if (myfile == NULL) {
    printf("Cannot open file for write.\n");
```

Creating I/O streams

- fopen: open/create a file for I/O
- fclose: close a file for I/O

- I/O one char at a time
  - fgetc: Reads ASCII from stream
  - fputc: Writes " to "
  - getchar: Reads Ascii from file
  - putchar: Writes " to "

Read/Write a file steps:

.
.
.

FILE*

write (could be read too)
↓
Ex: myfile = fopen("file_name.txt", "w")

EOF = End of File

Defining a struct

- To allocate memory for a struct, we declare a variable using this data type

struct flightType plane;

```
struct flightType {

    int lat;
    int long;
    bool flying; };
```

plane.lat (will give you the int lat for plane.

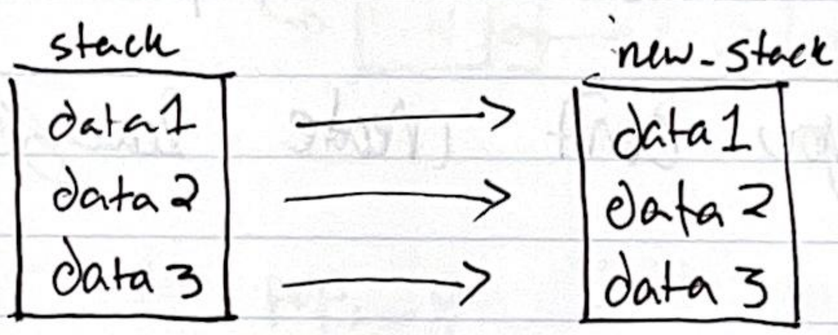
Pass structs by argument:

ex:

```
void print_flightName(Flight *plane)
```

What does this do?

instead of



making another stack with the data being referenced,

simply look at the same data/stack by passing as a pointer

%s is automatically passed by reference - no ampersand neccessary

malloc function allows you to assign more bytes of memory in the heap

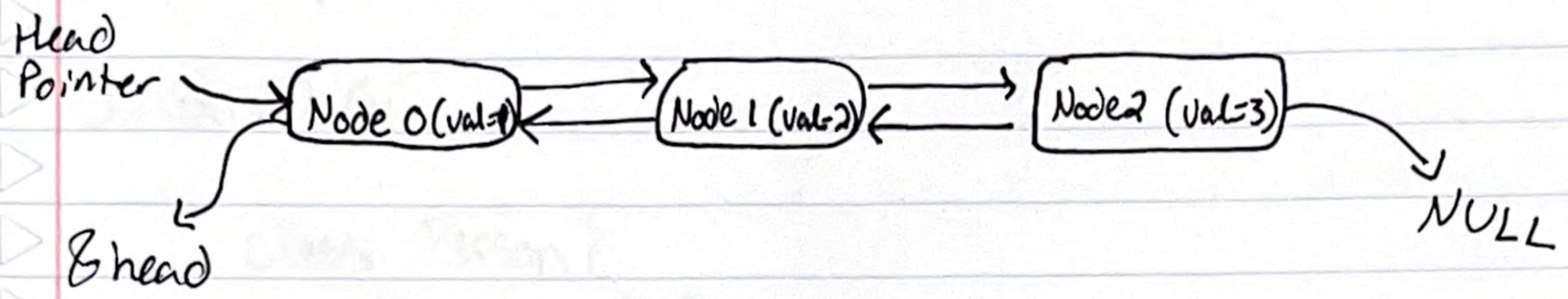
free function frees the memory space pointed by ptr

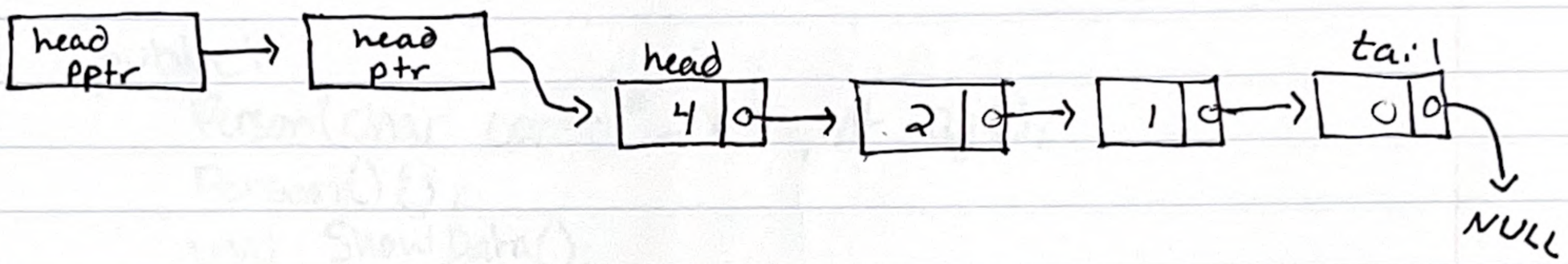
void free(void *ptr)

ptr should point any Heap memory location

Use valgrind to check memory leakage

Make sure you don't create leakage





if you want to pop, go left to right until

*headptr = NULL

Polymorphism

Constructor

```
class Person{
    Char name[20];
    int. age;

public:
    Person(char const *_name, int age);
    Person(){};
    void ShowData();
};


Person:: Person(char
```

Default Constructor

Destructor:

- A member funetion that destructs ~~the~~ an object
- Called automatically when the object goes out of scope
-

Operater overloading

- "redefining" built in operators (+, -, *, %, /, etc)
- Overloaded operaters are functions with special names:

operater followed by operater symbols

ex:

p.operator+(10);