Traps and Pointers

CIS 2400 Recitation 7

Recitation Outline

- Interacting with OS Code
  - TRAP/RTI
  - Directives
  - LC4 I/O
  - System calls
- Intro to C
  - Pointers
  - Strings
  - Output parameters

Interacting with OS Code

Invoking OS Code

TRAP UIMM8

- Stores PC + 1 into R7
  - Stores the address we were at so we can return to it later
- Sets PC to 0x8000 + UIMM8
  - Sends the User to execute instructions in the OS potion of memory.
- Sets PSR[15] to 1
  - Sets the privilege bit, marking that it is safe to access and execute OS code

Note that we are limited to TRAP to only 256 different addresses 0x8000 to 0x80FF

- OS Code is much bigger than 256 addresses

OS Layout

To have some control over where the user enters the OS, they are limited to 256 possible locations.

- These 256 locations are populated with JMP instructions that will send the user to a specified OS function
- Each of the 256 locations could send the user to a different OS system call, maximizing our use of the UIMM8 stored in TRAP

```
;;; OS Code
.OS
.CODE
.ADDR x8000;
JMP TRAP_GETC   ; x00
JMP TRAP_PUTC   ; x01
JMP TRAP_DRAW_H_LINE ; x02
JMP TRAP_DRAW_V_LINE ; x03
```

Returning From OS Code

RTI

- PC = R7
  - Sets the PC to be value we stored last time when we called TRAP
- PSR[15] = 0
  - Sets Privilege bit to 0, we are returning to User code

ASM Directives

Used in the assembler, not stored in memory

Prefixed with “.”

- [!!] .UCONST: Associates a label with a constant unsigned value
- .CONST: Associates a label with a constant signed value
- .BLKW: Reserve *UIMM16* words of memory from the current address
- .STRINGZ "String": Expands to a .FILL for each character in "String"
- .FALIGN: Pad current memory address to next multiple of 16

Pseudo-Instructions

Useful abstraction to bundle different instructions together / auto-fill arguments

- LC: Load Constant
  - Sets a register to have a constant value like those set with .UCONST
  - Assembles to a CONST and HICONST pair at runtime
- LEA: Load Effective Address
  - Stores address of <Label> in Rd
  - Assembles to a CONST and HICONST pair at runtime
- RET: Return to R7
  - Assembles to JMPR R7

I/O Registers

There are separate “registers” that are used for I/O

- Not the same as R0, R1, PC, or the PSR
- Instead, the register is a dedicated location in Memory
  - Usually address is setup in ASM with .UCONST

```
OS_ADSR_ADDR  .UCONST xFE04  ; display status register
OS_ADDR_ADDR  .UCONST xFE06  ; display data register
```

To read/write to the register:

- LC the address
- Use LDR or STR to read/write

I/O Registers

There are two registers paired for I/O operations

```
OS_ADSR_ADDR   .UCONST xFE04   ; display status register
OS_ADDR_ADDR   .UCONST xFE06   ; display data register
```

- Status Register
  - The MSB of the status register designates if I/O is ready
- Data Register
  - Where we read/write to depending on the register.

Example System Call (PUTC)

```
TRAP_PUTC
    LC R4, OS_ADSR_ADDR
    LDR R1, R4, #0
    BRzp TRAP_PUTC ; Loop while the MSB is zero

    LC R4, OS_ADDR_ADDR
    STR R0, R4, #0 ; Write out the character

    RTI
```

System Call Practice 1

Implement the system call TRAP_DRAW_H_LINE.

This will draw a horizontal line on the video display in between two given columns in a specified row.

Inputs:

- R0 - row to draw on
- R1 - column address 1
- R2 - column address 2
- R3 - color to draw with

```
TRAP_DRAW_PIXEL
  ; fill (R0, R1) with R3
  ; compute (R0, R1) address
  LEA R4, OS_VIDEO_MEM
  LC R5, OS_VIDEO_NUM_COLS
  MUL R5, R0, R5
  ADD R5, R5, R1
  ADD R4, R4, R5

  ; draw the color
  STR R3, R4, #0
```

System Call Practice 1 – Sample Solution

```
TRAP_DRAW_H_LINE
     CMP R1, R2 ; figure out whether R1 or R2 is larger
     BRnz NO_SWAP
     ADD R4, R1, #0 ; swap R1 and R2 using R4
     ADD R1, R2, #0
     ADD R2, R4, #0 ; R1 <= R2
NO_SWAP
     LEA R4, OS_VIDEO_MEM
     LC R5, OS_VIDEO_NUM_COLS
     MUL R5, R0, R5 ; compute (row * NUM_COLS)
     ADD R5, R5, R1 ; compute (row * NUM_COLS) + col
     ADD R4, R4, R5 ; add offset to the start of video mem
DRAW_LOOP
     STR R3, R4, #0 ; fill in the pixel
     ADD R4, R4, #1 ; update pixel address (increment col)
     ADD R1, R1, #1 ; update R1
     CMP R1, R2 ; test whether R1 <= R2
     BRnz DRAW_LOOP
     RTI
```

System Call Practice 2

Implement the system call TRAP_GETL.

This will read in a line of input from the terminal, until it reads the new line character \n. *\n should not be included in the output.*

R0 is setup to be a pointer to where the result should be stored. You may assume there is enough space to store the string.

Hints:

- Implementation for TRAP_GETC (gets a single character), which returns a character if there is one to read.
- Your code should wait until a character is ready.
- Newline character has an ASCII value of 10

```
TRAP_GETC
    LC R4, OS_KBSR_ADDR
    LDR R0, R4, #0
    BRzp GETC_END
    LC R4, OS_KBDR_ADDR
    LDR R1, R4, #0
GETC_END
    RTI
```

System Call Practice 2 – Sample Solution

```
TRAP_GETL
    LC R4, OS_KBSR_ADDR
    LDR R4, R4, #0
    BRzp TRAP_GETL ; try again if not ready
    LC R4, OS_KBDR_ADDR
    LDR R1, R4, #0 ; read character
    CMPI R1, #10 ; compare to \n (ASCII 10)
    BRz GETL_END
    STR R1, R0, #0
    ADD R0, R0, #1
    JMP TRAP_GETL
GETL_END
    RTI
```

Memory in C

Pointers

- Pointers are another primitive data type
- An integer can hold an index into an array
- If memory is a giant array of bytes, then a pointer just holds an index into that array

```
type *name;
```

```
int index;
```

3

```
int *ptr;
```

0x20...

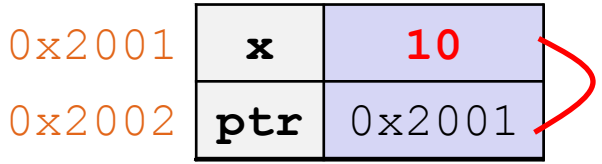
Pointer Syntax

& "Address of"

***** "value at"

```
int x;
int *ptr;

ptr = &x;
x = 5;
*ptr = 10;
```



Note the two different uses of *

→ Declaring a pointer

→ Dereferencing a pointer

Pointer Practice

What does this program print?

→ 36, 42, 5

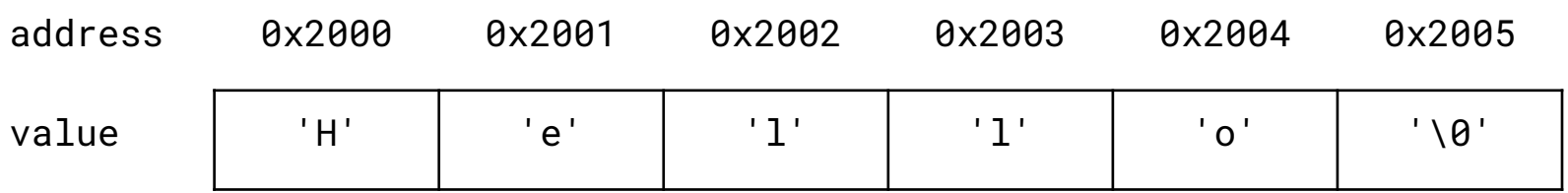
```
void bar(int *x, int *y, int *z) {
  z = x;
  *x = 6;
  *z = *x * *z;
  y = *x;
}

int main(int argc, char *argv[]) {
  int x = 16, y = 42, z = 5;
  bar(&x, &y, &z);
  printf("%d, %d, %d\n", x, y, z);
  return 0;
}
```

Strings

- A string is an array of characters that has a null terminator character at the end '\0'
- When allocating space for a string, remember to save space for the null terminator!

```
char str[] = "Hello";
```

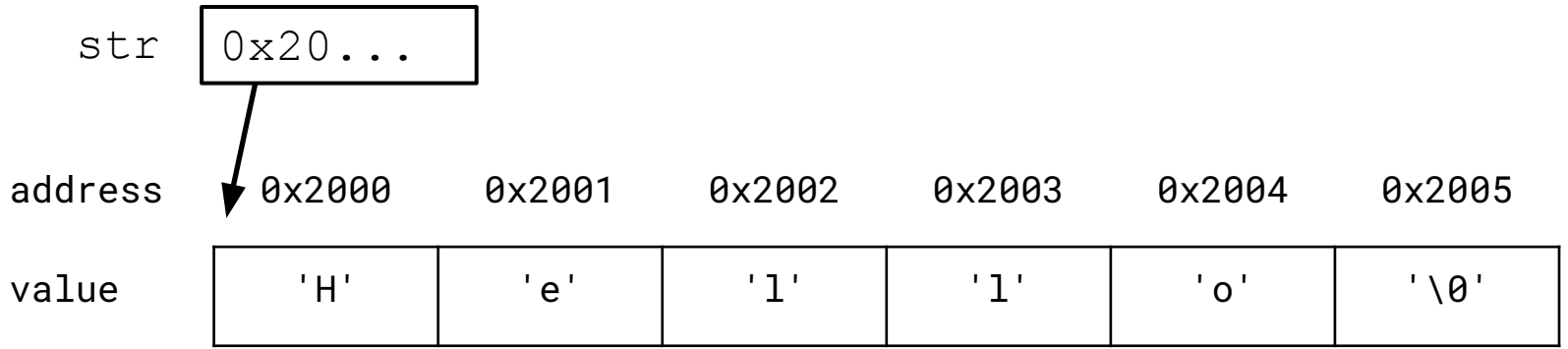


Strings as char*

You can also use a pointer for a string.

C will allocate the characters somewhere else in memory and the pointer will point to the first character in the string

```
char *str = "Hello";
```



Strings Practice

Complete the implementation of the function strcpy

```
char* strcpy(char *dest, char *src);
```

Takes in a destination and a source, copying the string of the source into the destination. Assume that there is enough space to hold the copied string in **dest** returns the address passed in as **dest**.

Strings Practice – Sample Solution

```
char* strcpy(char *dest, char *src) {
    char* dest_res = dest;
    while (*src != '\0') { // loop until we hit \0
        *dest = *src; // copy over content at src to dest
        ++dest;
        ++src;
    }
    *dest = '\0'; // don't forget to null-terminate!
    return dest_res;
}
```

Output Parameters

In the following function, will the user get 5 as output? If not, how would you rewrite the function for the user to get 5?

```
void get_five(int out) {
  ret = 5;
}

int main() {
  int x;
  get_five(x);
  printf("%d\n", x);
}
```

Output Parameters

Will the user get 5 as output?

No! You need to use a pointer so that the function can access the integer owned by the caller

```
void get_five(int *out) {
  *ret = 5;
}

int main() {
  int x;
  get_five(&x);
  printf("%d\n", x);
}
```

Output Parameters Practice

Write a function called **product_and_sum()** that take an array, array length, as input parameters, has two integer output parameters, and returns void.

The function should calculate the sum of all values in the array and the product of all values in the array, and then return the sum and product through output parameter.

After you have written the function, write a main() function that setups an array, calls the function, and prints the output.

Output Parameters Practice – Sample Solution

```
void product_and_sum(int* arr, int len, int* sum, int* prod) {
    *sum = 0;
    *prod = 1;
    for (int i = 0; i < len; ++i) {
        *sum += *arr;
        *prod *= *arr;
        ++arr;
    }
}

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int sum, prod;
    product_and_sum(arr, 5, &sum, &prod);
}
```

That's all we have for today!

Reminders:

- TA-lead recitations will take place on
  - Tuesdays 6:30-8:00pm in Moore 100A
  - Wednesday 12:00-1:30pm in Moore 100C
- Due dates
  - Check-in06 4:59pm Wednesday 11/2
  - HW06 11:59 pm on Friday 11/4
  - Mid-semester survey 11:59pm on Wednesday 11/9