# 14-io-x86-notes

## Recursion; I/O

# Agenda

- 0. Re-O
- 1. Recursion
- 2. I/O
- 3. Basic I/O Devices
- 4. Processor-I/O interaction

*Reading:*

- Chapter 3, up to and including 3.6.2

# 0. Re-O

The layers, from the human down to the physics

The ISA

Using the stack

Caller-save, callee-save

# 1. Recursion

What if we wanted a subroutine to call itself?

E.g.

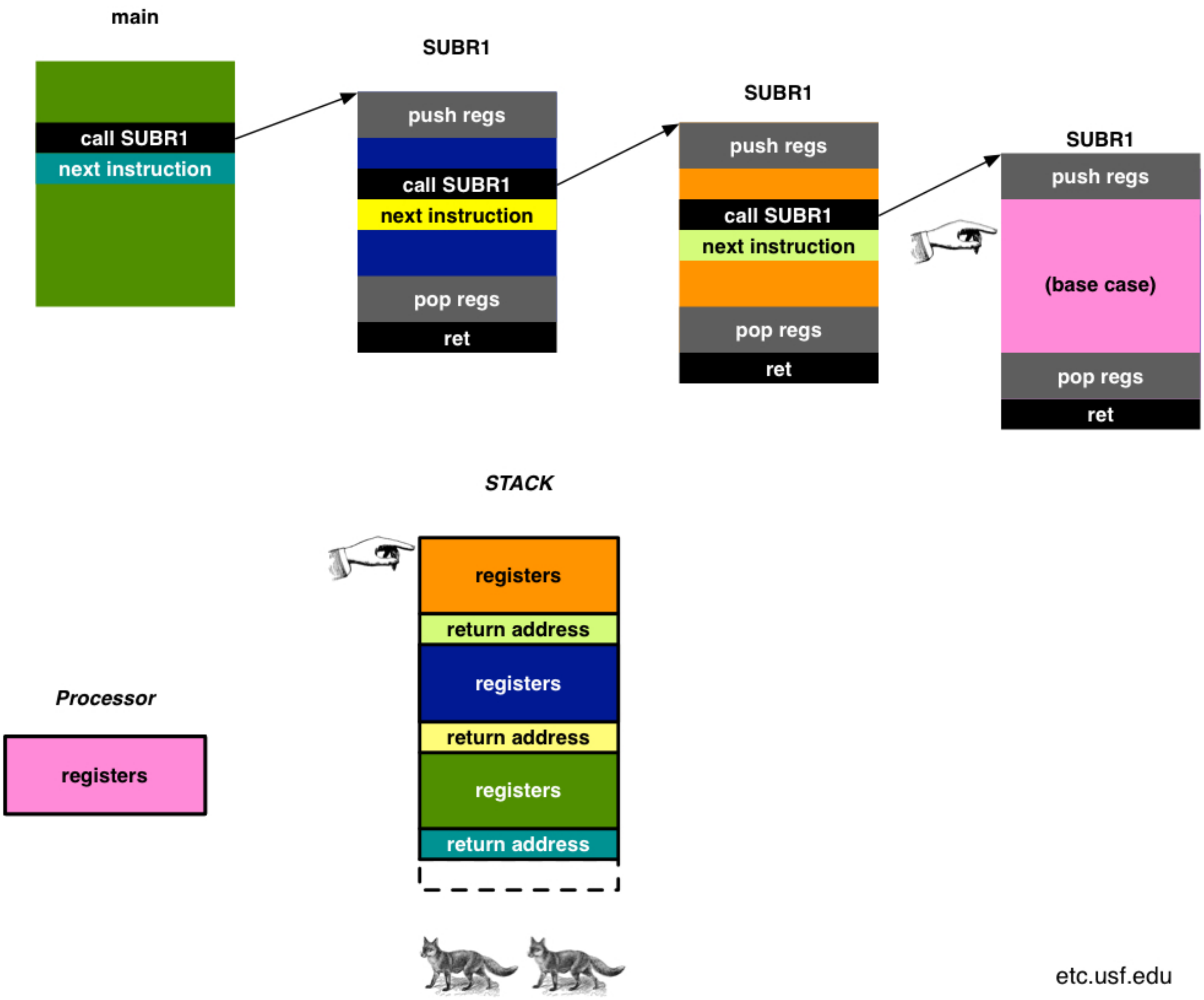
```
Precondition:  x >= 0.

if x = 0       FUN(x) = 0
if x > 0       FUN(x) = x + FUN(x-1)
```

Think through the pieces..

- Set up a stack
- Establish a convention for where FUN will look for its argument
- Establish a convention for where it will leave its argument
- Do a test!
- And make sure that anything FUN cares about before it recurses will still be there afterwards

[recurse.ys (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/13-stack/recurse.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/13-stack/recurse.zip)



etc.usf.edu

## 2. I/O

Since we aspire to do "hello world"....

Examples:

- keyboard input
- monitor output
- D/A converter
- Voltage sensor

What's involved....

- with computation
- with hw?

# 3. Basic I/O Devices

Generic model of a device

data register

status register

(but sometimes, the "ready" bit for output devices has a different meaning)

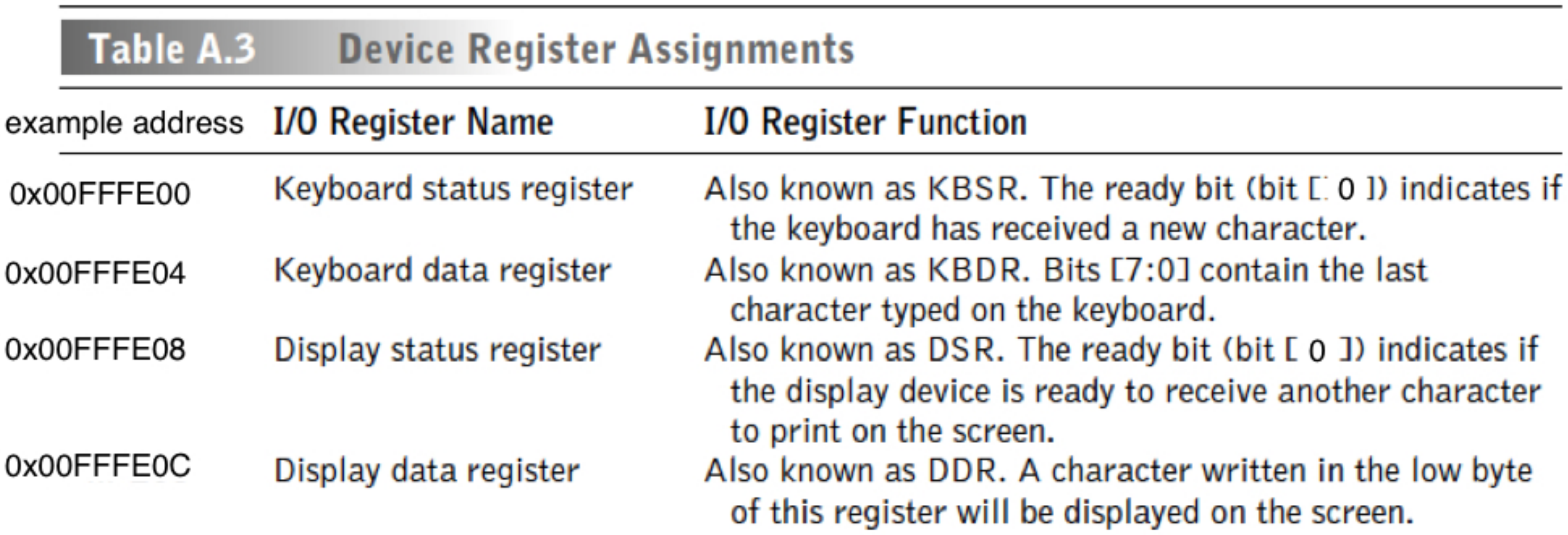


**Table A.3 Device Register Assignments**

| example address | I/O Register Name | I/O Register Function |
|---|---|---|
| 0x00FFFE00 | Keyboard status register | Also known as KBSR. The ready bit (bit [ 0 ]) indicates if the keyboard has received a new character. |
| 0x00FFFE04 | Keyboard data register | Also known as KBDR. Bits [7:0] contain the last character typed on the keyboard. |
| 0x00FFFE08 | Display status register | Also known as DSR. The ready bit (bit [ 0 ]) indicates if the display device is ready to receive another character to print on the screen. |
| 0x00FFFE0C | Display data register | Also known as DDR. A character written in the low byte of this register will be displayed on the screen. |

(adapted from Patt & Patel for the Y86)

LogiSim has some fun devices baked-in:

[io.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/io.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/io.zip)

ASCII (see the course resources page)

# 4. Processor-I/O interaction

Memory-mapping vs special instructions

**How would you wire devices to a system so that they listened/spoke at the right addresses?**

## Asynchronous I/O

Handshakes

Polling

Input:

```
start:          irmovl pKBSR, %eax # read KBSR until it's 1
                mrmovl (%eax), %eax
KBNotReady:     mrmovl (%eax), %ebx
                addl %ebx,%ebx
                je KBNotReady # jmps if zero

                # got a character---get it into %ecx
                irmovl pKBDR, %eax
                mrmovl (%eax), %eax
                mrmovl (%eax), %ecx

                # go back
                jmp start

# these named locations store the addresses     of the registers
pKBSR: .long 0x00FFFE00
pKBDR: .long 0x00FFFE04
```

in.ys (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/in.zip)

Note the annoyance of "go to this memory address, get the data there, use THAT an address, and get the data THERE into the register"

```
        irmovl pKBDR, %eax
        mrmovl (%eax), %eax
```

With the assembler, we could also do it this way:

```
KBNotReady:     mrmovl 0x00FFFE00, %ebx
```

---

Output:

```
start:
                irmovl $0x41, %ecx      # put ascii A into %ecx

                # read DSR until it's 1
                irmovl pDSR, %eax
                mrmovl (%eax),%eax

DNotReady:      mrmovl (%eax), %ebx
                addl %ebx,%ebx
                je DNotReady # jmps if zero

                # write the char!
                irmovl pDDR, %eax
                mrmovl (%eax), %eax
                rmmovl %ecx, (%eax)

                halt

# these named locations store the addresses of the registers
pDSR:  .long 0x00FFFE08
pDDR:  .long 0x00FFFE0C
```

[out.ys (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/out.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/out.zip)

---

Together (and as subroutines!)

```
echo:           call GETC
                call PUTC
                jmp echo
```

echo.ys (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/14-io-x86/echo.zip)

(My classroom simulator includes memory-mapped I/O. This is what I'll demo in class today.)

---

**(Caveat: we'll talk about more advanced methods later in the term)**

---