How to be a Real Person

James Forcier & Ross Delinger

October 11 2015

The PCjr

- Intel 8088 processor
  - Limited subset of x86 instructions
  - 16-bit real mode only
- 128KB RAM (upgraded!)
- 3.5” floppy drive

8088 Instruction Set

- Basic arithmetic, flow control, etc.
- Simplistic stack management
- Not many general purpose registers

The Project

- Bootloader and kernel
- Simple Lisp/Forth interpreter
- Coroutines
- Possibly network stack?

Challenges

- Ancient fucking hardware
- Documentation
- Booting
- Function calling convention
- Working with high level languages

Ancient Fucking Hardware

- Floppy disks/drives
- Interfacing with the machine

Documentation

- Everything out there assumes real mode is just a stepping stone to protected mode
- IBM didn't release any documentation publicly
- PCjr community are only interested in running DOS software, apparently

Booting

- More documentation issues
- OSDev Wiki turned out to be accurate
- MBR needs magic numbers

MBR Code

```
[BITS 16] ; 8088 is 16-bit
[ORG 0x7C00] ; At boot, we get put here

boot:
    ; Initialize some boot stuff

times 510-($-$$) db 0
db 0x55
db 0xAA
```

Booting

- Need to load more OS from floppy

Booting

```
reset_drive:
    mov ax, 0 ; function selector
    mov dl, 0 ; drive 0
    int 0x13 ; drive operation interrupt
    jc reset_drive ;if failure, try again
read_drive:
    mov ax, 0x07E0 ; load to immediately after MBR
    mov es, ax
    mov bx, 0 ; load to beginning of ES

    mov ah, 2 ; load to ES:BX
    mov al, 2 ; load 1 sector
    mov ch, 0 ; cylinder 0
    mov cl, 2 ; sector 2
    mov dh, 0 ; head 0
    mov dl, 0 ; drive 0
    int 0x13 ; drive operation interrupt
    jc read_drive ; if failure, try again
```

Function Calling Convention

- Eventually we (might) want to use a higher-level language
- How do arguments get passed into functions?
- We could use registers, but...

Function Calling Convention

- Eventually we (might) want to use a higher-level language
- How do arguments get passed into functions?
- We could use registers, but...
- There are only a few of them

Function Calling Convention

- Eventually we (might) want to use a higher-level language
- How do arguments get passed into functions?
- We could use registers, but...
- There are only a few of them
- Instead, we use something called `cdecl`

cdecl

Calling

1. Caller pushes arguments onto stack in right-to-left order
2. Return address is pushed onto stack by `call`
3. Caller's base pointer is saved onto stack by callee
4. Callee subtracts from stack pointer to create space for local variables
5. Callee saves `bx` to the stack

Returning

1. Callee restores `bx` from the stack
2. Callee restores stack pointer from base pointer
3. Callee restores base pointer from stack
4. Callee stores return value in ax
5. Callee returns
6. Caller adds to stack pointer to clean up stack

cdecl implementation

Caller

```
; Calling print_char(character, color)
; character is in ax, color is in bx
push bx
push ax
call print_char
; result is now in ax
add sp, 2 ; clean up stack
```

cdecl implementation

Entering callee

```
print_char:
    push bp ; save bp to stack
    mov bp, sp ; sp is our new bp
    sub sp, 4 ; make space for 2 local variables

    ; Because of local variables, args start at [bp + 4]

    mov [bp - 2], bx ; save bx to local variable
    mov bx, [bp + 4] ; load character into bx
    mov ax, [bp + 6] ; load color into ax

    ; ...
```

cdecl implementation

Exiting callee

```
; Assuming result is already in ax

mov bx, [bp - 2] ; restore bx from local vars
mov sp, bp ; restore sp from bp
pop bp ; restore bp from stack
ret
```

Working with High-Level Languages

- `cdecl` is needed to interface with C
- Basic memory management for heap needed for most languages
  - Except stack-based languages such as Forth, Brainfuck, etc.
- Need to write wrapper functions around BIOS interrupts for I/O

Next steps

- Forth/Lisp support (partially implemented)
- Coroutines
- Network stack over serial