EC325 Microprocessors System Calls etc

Yasser F. O. Mohammad

REMINDER 1: Logical Operation

- AND destination, source
- OR destination, source
- XOR destination, source
- NOT destination

REMINDER 2: Shift and Rotation Instructions

	Left	Right
Logical	SHL	SHR
Arithmetic	SAL	SAR

- S* destination, count
  - Count can be immediate or CL (mod 32)
- SHL, SAL are identical
- SAR sign extends the shifted bit
- SHR zero extends the shifted bit
- The bit that goes out is written to CF

Who can you call?*

- System
  - Low Level
  - Depends on OS
  - Fastest
  - e.g. using INT 21H in DOS
- Runtime Libraries (e.g. C)
  - Higher Level
  - OS independent (usually)
  - Slightly slower
  - e.g. calling *printf* from your assembly code
- Runtime libraries use system calls to achieve their goals

Why Performing System/Library Calls? *

- Input / Output
- Interacting with Files
- System Services (e.g. Dynamic Memory Allocation)

System calls in DOS*

- INT 21H
  - Service number is put in AH
  - Parameters are passed using registers
  - Pointers to large parameters are passed in registers
  - Return value can be read from registers (AL)
  - Supports only 16-bit code

  - Find them all in DOS Programmer's Reference

Example DOS calls*

- **Function 1- Character input with echo**

- **Action:** Reads a character from the standard input device and echoes it to the standard output device. If no character is ready it waits until one is available. I/O can be re-directed, but prevents detection of OEF.
- **On entry:** AH = 01h
- **Returns:** AL = 8 bit data input

Example DOS calls*

- **Function 2 - Character output**
- **Action:** Outputs a character to the standard output device. I/O can be re-directed, but prevents detection of 'disc full'.
- **On entry:**

  AH = 02h
  DL = 8 bit data (usually ASCII character)
- **Returns:** Nothing

Example DOS calls*

- **Function 09- Output character string**
- **Action:** Writes a string to the display.
- **On entry:**
  AH = 09h
  DS:DX = segment:offset of string
- **Returns:** Nothing
- **Notes:** The string must be terminated by the $ character (24h), which is not transmitted. Any ASCII codes can be embedded within the string.

Example DOS calls*

- **Function oAh - Buffered input**
- **Action:** Reads a string from the current input device up to and including an ASCII carriage return (oDh), placing the received data in a user-defined buffer

- **On entry:**
  AH = oAh
  DS:DX = segment:offset of string buffer

- **Returns:** Nothing

- **Notes:** The first byte of the buffer specifies the maximum number of characters it can hold (1 to 255). This value must be supplied by the user. The second byte of the buffer is set by DOS to the number of characters actually read, excluding the terminating RETURN. If the buffer fills to one less than its maximum size the bell is sounded and subsequent input is ignored.

  If a CTRL-C is detected an INT 23h is executed. Normal DOS keyboard editing is supported during input

Example DOS calls*

- **Function oBh - Get input status**
- **Action:** Checks whether a character is available from the standard input device. Input can be redirected

- **On entry:**
  AH = oBh

- **Returns:**
  AL = o if no character available
  AL = oFFh if character available

- **Notes:** Notes: if an input character is waiting this function continues to return a true flag until the character is read by a call to function 1, 6, 7, 8 or oAh.

Example DOS calls*

- **Function 4Ch - Terminate program with return code**
- **Action:** Terminates execution of a program with return to COMMAND.COM or a calling routine, passing back a return code.
- **On entry:** AH = 4Ch
  AL = Return code (Error level)
- **Returns:** Nothing
- **Notes:** This is the approved method of terminating program execution. Conventionally a return code of zero indicates success, any other value failure. Standard DOS return codes are:

  0: Successful operation
  1: CTRL-BREAK termination
  2: Critical error terrnination
  3: Terminated and stayed resident

  Return code values can be used at the discretion of the programmer (avoiding codes 1 to 3).

Example DOS program*

```
.model small
.Data
str2in    DB 20,0
str2out   DB 20 DUP(?)
.Code
_start:
    mov ah, 0ah                    ; read a string to str2in from standard input
    mov dx, offset str2in
    int   21h

    mov bx, 0                      ; get the real number of input characters to bx
    mov bl, byte ptr [strin+1]
    add  bx, dx                    ; add number of read characters to the string pointer
    mov byte ptr[bx+2],'$'         ; add a '$' after the string to prepare for function 09h

    mov  ah, 09h                   ; write the string to standard output
    mov  dx, offset str2out
    int    21h

   mov ah, 4ch
   int   21

END
```

LINUX system calls*

- INT 80H
  - System call value is place in EAX
  - Parameters are passed in registered if < 6
    - EBX (first), ECX (second), EDX (third), ESI (fourth), EDI (fifth)
  - Parameters are passed in the stack if >5

- Notice that MASM cannot be used with LINUX
  - Usually NASM is used and it has completely different syntax

WINDOWS system calls

- CALL function
  - Uses *stdcall* convention for parameter passing
  - 16-bit and 32-bit versions are available (64-bit recently)
  - Usually done using INVOKE
  - The ones we will used are defined in Kernel32 library

What is a calling convention? *

- An agreement about:
  - How parameters are passed from caller to callee?
  - Who is responsible of saving registers used?
  - Who is responsible of cleaning parameters from stack (if needed) upon return?
  - What is the order of the parameters in the stack (or register file)?
  - How to name procedures?

Most Important calling conventions*

- C
- THISCALL
- STDCALL
- FASTCALL

C (*cdecl* in C, *C* in assembly) *

- Default for C functions and global C++ functions
- Arguments are pushed on the stack in reverse order.
- The caller pops arguments after return.
- Primitive data types, except floating point values, are returned in EAX or EAX:EDX depending on the size.
- Float and double are returned in fpo, i.e. the first floating point register.

LINUX and WINDOWS compilers differ in how to handle returning class objects

ThisCall
(*thiscall* in assembly) *

- Used with C++ member functions that take fixed number of parameters. Functions taking variable number of parameters always uses C calling convention with *this* pointer passed as first in stack
- *this* pointer is passed in ECX
- The **callee** pops arguments after return.
- Primitive data types, except floating point values, are returned in EAX or EAX:EDX depending on the size.
- Float and double are returned in fpo, i.e. the first floating point register.

LINUX and WINDOWS compilers differ in how to handle returning class objects

STDCALL (*stddecl* in C, *stdcall* in assembly) *

- Used for ALL Win32 API functions
- Arguments are pushed on the stack in reverse order.
- The **callee** pops arguments after return.
- Primitive data types, except floating point values, are returned in EAX or EAX:EDX depending on the size.
- Float and double are returned in fpo, i.e. the first floating point register.

LINUX and WINDOWS compilers differ in how to handle returning class objects

FASTCALL

(*fastdecl* in C, *fastcall* in assembly) *

- Similar to stdcall but with first two parameters passed in registers ECX and EDX
- Rarely used

How to call a windows system function?

1. Declare it using PROTO
2. Call it using INVOKE
3. When linking add the needed library (kernel32.lib) to the linker command line

Declaring using PROTO

- label PROTO [distance] [langtype] [,[parameter]:tag]
  - Distance:
    - NEAR32
    - NEAR
    - FAR
  - Langtype
    - C (An underscore is automatically added to function name)
    - STDCALL
    - FASTCALL
  - TAG= data type

Examples

```
ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

GetStdHandle PROTO NEAR32 stdcall,
    nStdHandle:DWORD

WriteFile PROTO NEAR32 stdcall,
    hFile:DWORD, lpBuffer:NEAR32, nNumberOfCharsToWrite:DWORD,
    lpNumberOfBytesWritten:NEAR32, lpOverlapped:NEAR32
```

Calling using INVOKE

- INVOKE label [, parameter]

```
INVOKE  ExitProcess, 0
```

```
INVOKE GetStdHandle,
  STD_OUTPUT


INVOKE WriteFile,
  hStdOut,
  strAddr,
  strLength,
  NEAR32 PTR written,
  0
```

CASE STUDY: Console I/O using Kernel32

- Input/output from all devices can be handled as file I/O:
  - Keyboard/display
  - Serial port
  - MODEM
- To access a file you need a HANDLE to it
- With the handle you can READ, WRITE, and CLOSE files (among other things)

How to do console I/O

- Get a handle to the console using GetStdHandle:
  - STDIN for input (-10)
  - STDOUT for output (-11)
- Read using ReadFile
- Write using WriteFile

Writing a string

```
.386
.MODEL FLAT

ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

GetStdHandle PROTO NEAR32 stdcall,
    nStdHandle:DWORD

WriteFile PROTO NEAR32 stdcall,
    hFile:DWORD, lpBuffer:NEAR32, nNumberOfCharsToWrite:DWORD,
    lpNumberOfBytesWritten:NEAR32, lpOverlapped:NEAR32

STD_OUTPUT EQU -11

cr      EQU     0dh     ; carriage return character
Lf      EQU     0ah     ; line feed

.STACK
.DATA

OldProg BYTE    "Old programmers never die.", cr, lf
        BYTE    "They just lose their byte.", cr, lf
msgLng  DWORD   56   ; number of characters in above message
written DWORD   ?
hStdOut DWORD   ?

.CODE
_start:
        INVOKE GetStdHandle,    ; get handle for console output
          STD_OUTPUT
        mov    hStdOut, eax

        INVOKE WriteFile,
          hStdOut,              ; file handle for screen
          NEAR32 PTR OldProg,   ; address of string
          msgLng,               ; length of string
          NEAR32 PTR written,   ; bytes written
          0                     ; overlapped mode

        INVOKE  ExitProcess, 0  ; exit with return code 0

PUBLIC _start
END
```

Full example

- Reading a string from the keyboard, converting it to lower case and putting it back to the screen

- SEE the text book page 423

Full example 2

- Input output procedures in IO.ASM (as written by Detmar)

- See text book page 425

File I/O

- CreateFile
  - Creates or opens a file
  - Has two versions (as many Win32 functions)
    - CreateFileA: strings are using ASCII
    - CreateFileW: strings are using UNICODE
- CloseHandle
  - Closes a file opened by CreateFile

Examples

- Reading a file and displaying it to console
- Getting text from user and storing it into a file

- Text book pages 429, 433

How to get help

- MSDN is the primary source of information about WIn32 functions
- MASM32 has built in libraries and include files having all the prototypes of standard Win32 functions and ALL standard C functions as well.
- Look at IO.ASM, IO.H to get some idea about the details of using these prototypes

You can use *printf* in assembly