

ECE390

Computer Engineering II

Lecture 17



Dr. Zbigniew Kalbarczyk
University of Illinois at Urbana- Champaign

Lecture outline



- Protected Mode Library (PModeLib)
- Example code using MMX
- PMode Interrupts
- Mouse

Protected Mode Library (PModeLib)

- Uses 32-bit C calling convention
- A set of macros is provided
 - implement C-calling convention
 - simplify specifying parameters to PMode functions

invoke ;call a function

procendproc ;facilitate writing PMode functions

Implementing Functions Using proc endproc Macros (1)

; Diff with Register Inputs

```
GLOBAL _main
SECTION .text ;=====
_main
    mov  eax, 5
    mov  ebx, 3
    call Diff ;result in EAX
    ret

; Purpose: Subtracts ebx from eax
; Input:  eax, number to be subtracted from
;        ebx, amount to subtract
; Output: eax
Diff
    sub  eax, ebx
    ret
```

Implementing Functions Using *proc endproc* Macros (2)

; *Diff* with C Calling Convention (but without *proc*)

```
GLOBAL _main
SECTION .text ;=====
_main
    ;parameters pushed in reverse order
    push dword 3
    push dword 5
    call _Diff
    add esp, 8 ;remove parameters (still on stack)

    ret

; int Diff (int a, b)
; Purpose: Subtracts b from a
; Input: a, number to be subtracted from
;       b, amount to subtract
; Output: returns a - b
```

```
_Diff
    push ebp
    mov  ebp, esp
    mov  eax, [ebp+8]
    sub  eax, [ebp+12]
    pop  ebp
    ret  ;return to caller, with result in eax
```

Implementing Functions Using *proc endproc* Macros (3)

; *Diff* Using *proc*

```
%include "lib291.inc" ;defines proc, endproc, invoke
GLOBAL _main
SECTION .text ;=====
_main
    invoke _Diff, dword 5, dword 3
    ret ; result in eax

; int Diff (int a, b)
; Purpose: Subtracts b from a
; Input: a, number to be subtracted from
;       b, amount to subtract
; Output: returns a - b
```

```
proc _Diff
.a arg 4 ;first parameters, dword (4 bytes)
.b arg 4 ;second parameters, dword (4 bytes)

    mov  eax, [ebp+.a]
    sub  eax, [ebp+.b]
    ret  ;return to caller, with result in eax

endproc
_Diff_arglen equ 8 ;sum of all parameter sizes
```

Implementing Functions Using *proc endproc* Macros (4)

; Pointer Parameters

```

#include "lib291.inc" ;defines proc, endproc, invoke
GLOBAL _main
SECTION .data
val1      dw 5
val2      dw 3
SECTION .bss
result    resw 1
SECTION .text ;=====
_main
    invoke _Diff, dword result, dword val1, dword val2
    ret

; int Diff (short *r, short *a, short *b)
; Purpose: Subtracts b from a
; Input: a, number to be subtracted from
;       b, amount to subtract
; Output: r, result a - b

```

```

proc _Diff
.r  arg 4
.a  arg 4
.b  arg 4
    push esi
    push edi
    mov esi, [ebp+.a]
    mov cx, [esi] ;get "a" value
    mov esi, [ebp+.b]
    sub cx, [esi] ;get "b" value

    mov edi, [ebp+.r]
    mov [edi], cx

    pop edi
    pop esi
    ret ;return to caller, with result in eax
endproc
_Diff_arglen equ 12 ;sum of all parameter sizes

```

Z. Kalbarczyk

ECE390

7

Image Dissolve Using Alpha Blending

- MMX instructions speed up image composition
- A flower will dissolve into a swan
- Alpha determines the intensity of the flower
- The full intensity, the flower's 8-bit alpha value is FFh, or 255
- The equation below calculates each pixel:

Result_pixel = Flower_pixel * (alpha/255) + Swan_pixel * [1-(alpha/255)]

For alpha 230, the resulting pixel is 90% flower and 10% swan



* 230/255 +



* 1 - 230/255 =

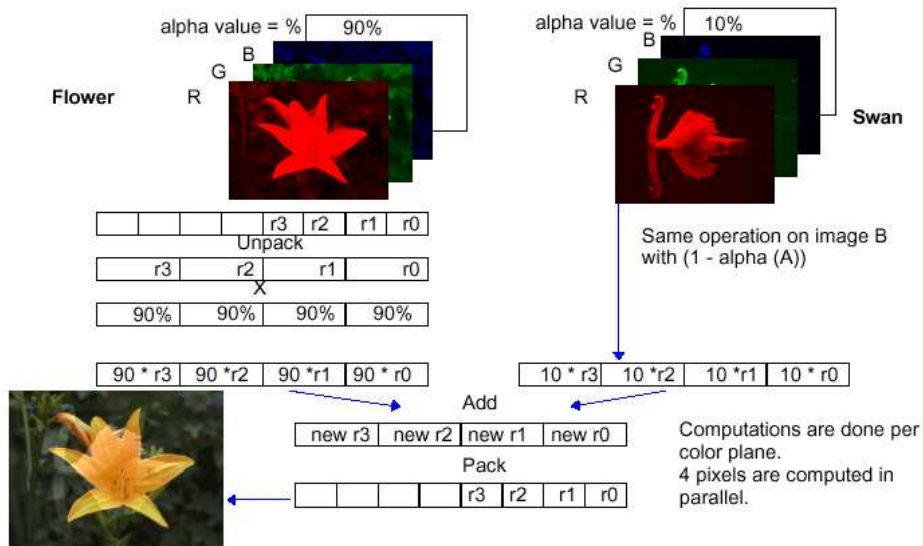


Z. Kalbarczyk

ECE390

8

Image Dissolve Using Alpha Blending



Z. Kalbarczyk

ECE390

9

Alpha Blending

```

proc _AlphaBlit
    .Foreground      arg 4
    .Background     arg 4
    .Alpha          arg 4
    .Width          arg 4
    .Height         arg 4

    xor     esi, esi
    xor     edi, edi
    mov     edx, [ebp + .Foreground]
    mov     ebx, [ebp + .Background]

    movd   mm0, [edx + esi*4] ;Load RedBox pixel (F)
    movd   mm1, [ebx + esi*4] ;Load BlueBox pixel (B)

    pxor   mm2, mm2
    punpcklbw mm0, mm2 ;unpack RedBox pixel
    punpcklbw mm1, mm2 ;unpack BlueBox pixel

    mov     eax, [ebp + .Alpha] ; alpha
    movd   mm2, eax
    pxor   mm3, mm3
    punpcklbw mm2, mm3 ; unpack alpha

    psubw  mm0, mm1 ; F-B
    pmullw mm0, mm2 ; ((F-B)*alpha)
    psrlw  mm0, 8 ; divide by 256
    paddw  mm0, mm1 ; ((F-B) * alpha)/256 + B
    packuswb mm0, mm0 ; pack pixel attributes

    movd   [edx + esi*4], mm0 ; write pixel to mem.

    inc     esi
    cmp     esi, [ebp + .Width]
    jb     .ldrawblit

    xor     esi, esi
    mov     eax, [ebp + .Width]
    shl     eax, 2
    add     edx, eax
    add     ebx, eax
    inc     edi
    cmp     edi, [ebp + .Height]
    jb     .ldrawblit

    ret

endproc

```

Z. Kalbarczyk

ECE390

_AlphaBlit_arglen EQU 20

10

Alpha Compose Algorithm Using MMX Instructions

- Load source pixel from memory into an MMX register
- Load destination pixel from memory into an MMX register
- For each pair of source/destination pixels:
 1. Unpack (bytes to words) the source pixel into an MMX register
 2. Copy out the source alpha byte into the four words of another MMX register
 3. Multiply the two previously mentioned MMX registers
 4. Shift each word right to divide by 256 (**alpha*A**)
 5. Unpack (bytes to words) the destination pixel into an MMX register
 6. Add the results of previous two steps (**alpha*A + B**)
 7. Multiply the destination pixel by the source alpha
 8. Shift each word right to divide by 256 (**alpha * B**)
 9. Subtract value of step 8 from the value of step 6 (**alpha*A + B - alpha *B**)
- Pack (words to bytes) the alpha-composed pixel back together with proper saturation in the correct order
- Write the composed pixel back to memory

Protected Mode Interrupt Handling

- *Instal_Int()* and *Remove_Int()* PModeLib functions help to install a standard interrupt handler in protected mode, e.g., timer or keyboard
 - Normal subroutine, which should end with a *ret* instruction
 - Should return a value in *eax* to indicate whether the interrupt should be chained (1) to the old handler or not (0)
- Important to lock the memory areas an interrupt handler will access
 - any variables it uses and the interrupt handler code itself

Mouse

- The mouse is controlled with INT 33h
 - A *mouse driver* must be installed to enable mouse
 - Installed at system boot time
- There are about 50 mouse functions available via INT 33
 - a function number should be specified in AX before invoking INT 33
- Define a callback function to be invoked on presses or releases of mouse buttons
- Use INT 33 function 0Ch to install the callback function

Mouse (Install)

```
_InstallMouse
    invoke _LockArea, ds, dword _mouseStatus, dword 1
    invoke _LockArea, cs, dword _MouseCallback, dword _MouseCallback_end - _MouseCallback

    invoke _Get_RMCB, dword _mouseSeg, dword _mouseOff, dword _MouseCallback, dword 1
    test   eax, eax
    jnz   .MouseDone

    mov   dword [DPMI_EAX], 0Ch
    mov   dword [DPMI_ECX], LEFT_BUTTON | RIGHT_BUTTON
    xor   edx, edx
    mov   dx, [_mouseOff]
    mov   [DPMI_EDX], edx
    mov   ax, [_mouseSeg]
    mov   [DPMI_ES], ax
    mov   bx, 33h
    call  DPMI_Int
    xor   eax, eax

.MouseDone
    ret
Z. Kalbarczyk
```

Mouse (Callback)

```
proc _MouseCallback
    .DPMIRegsPtr    arg    4

    test byte [_mouseState], LEFT_BUTTON | RIGHT_BUTTON
    jnz .done        ; if we haven't handled the last change, exit

    mov ebx, [ebp + .DPMIRegsPtr]

    mov ax, [es:ebx + DPMI_ECX_off]
    mov [_mouseX], ax                ;mouse position (X)
    mov ax, [es:ebx + DPMI_EDX_off]
    mov [_mouseY], ax                ;mouse position (Y)
    mov bl, [es:ebx + DPMI_EAX_off]
    ;mov bl, [es:ebx + DPMI_EBX_off]

    mov [_mouseStatus], bl           ; store the state info
.done
    ret
endproc
_MouseCallback_end
_MouseCallback_arglen EQU 4
Z. Kalbarczyk ECE390
```