Macros in Pentium Assembly Language

October 25

CSC201 Section 002

Fall, 2000

Equates

```
PI          equ   3.1415926
MAXRECS     equ   200
RECSIZE     equ   5
move        equ   mov
la          equ   lea
…
float1      dd    PI
recarray    db    MAXRECS*RECSIZE dup (?)
…
move        EAX, EBX
```


Copyright 2000, Douglas Reeves

Equates (cont.)

- Assigns a label to a constant value or expression
- Easier to understand, easier to change, prevents inconsistencies
- Type of value
  - An integer constant or expression evaluates to a numeric type
  - Anything else (including floating-point value) evaluates to a string type

Copyright 2000, Douglas Reeves

Macros (Without Parameters)

```
GET_ID      macro
        mov   byte ptr [EBX+3], AL
        get_ch
        mov   byte ptr [EBX+2], AL
        get_ch
        mov   byte ptr [EBX+1], AL
        get_ch
        mov   byte ptr [EBX], AL
endm
```

```
....
GET_ID
....
GET_ID
....
GET_ID
```


Copyright 2000, Douglas Reeves

Macros (Without Parameters)

- Same benefits/uses as Equates
- Like "#define" in C/C++
- A macro definition appears once; a macro call can appear many times
- Syntax of a macro definition

```
<macro_name> macro <parameters>
<body of macro>
endm
```


Copyright 2000, Douglas Reeves

Macros vs. Subroutines (Procedure Calls)

- Macros are expanded at *assembly time*
  - the macro call is replaced by the "body" of the macro
  - increases program size
- Subroutines are called at *program execution time*
  - increases program execution time (due to overhead)

Copyright 2000, Douglas Reeves

Subroutine Calls

```
…
call    get_ID
….
call    get_ID
….
call    get_ID

get_ID proc
        mov    byte ptr [EBX+3], AL
        get_ch
        mov    byte ptr [EBX+2], AL
        get_ch
        mov    byte ptr [EBX+1], AL
        get_ch
        mov    byte ptr [EBX], AL
        ret
get_ID endp
```


Copyright 2000, Douglas Reeves

Macros With Parameters

```
SWAPDW      macro       v1:req, v2:req
        push    dword ptr [v1]
        push    dword ptr [v2]
        pop     dword ptr [v1]
        pop     dword ptr [v2]
endm
```

```
num1  dd      100
num2  dd      200
num3  dd      300
…
        SWAPDW      num1, num2
…
        lea     EBX, num2
        lea     EDX, num3
        SWAPDW      EBX, EDX
```


Copyright 2000, Douglas Reeves

More Advanced Macros (With Parameters)

- Allows macro to expand to a different instruction sequence for each call
- Parameters to a macro can be register names, variable names, or constants
- The order, number, and type of parameters in the macro call should agree with the macro definition
  - We will assume all parameters are "required"
  - Indicated by ":req" parameter qualifier in macro definition

Copyright 2000, Douglas Reeves

Protecting Register Values in Macro Calls

- (Deferred until we discuss procedures)

Copyright 2000, Douglas Reeves

Macros Used for Data Definition

```
BUFDEFN     macro   bufName:req, bufSz:req
        bufName     dd      bufSz  dup (?)
endm
```

```
BUFDEFN     buffer1, 200
BUFDEFN     buffer2, 400
BUFDEFN     buffer3, 100
```


Copyright 2000, Douglas Reeves

Labels in Macros

```
ZERONEG     macro          mvar:req
            cmp    dword ptr mvar, 0
            jge    dontzero
            mov    dword ptr mvar, 0
dontzero:
endm
```

```
...
            ZERONEG      var1
...
            ZERONEG      var2
...
```


Copyright 2000, Douglas Reeves

Labels and the "Local" Directive

- Labels in macros cause problems; expansion can create duplicate labels!
- Solution: declare the label as "local" to the macro
- Each time the macro is called, the assembler will generate a different label

```
ZERONEG     macro          mvar:req
            local   dontzero
            cmp     dword ptr mvar, 0
            jge     dontzero
            mov     dword ptr mvar, 0
dontzero:
endm
```


Copyright 2000, Douglas Reeves

Nested Macros

```
SORT2 macro       v1:req, v2:req
      local   dontswap
      mov     EAX, v2
      cmp     v1, EAX
      jle     dontswap
      SWAPDW      v1, v2
dontswap:
endm
```

```
…
      SORT2       dword1, dword2
…
      SORT2       dword3, dword2
…
```

- A macro definition may contain a call to another macro

Copyright 2000, Douglas Reeves

Conditional Assembly

```
SASM   equ    1
…
COPYVAR  macro     mvar1:req, mvar2:req
ifdef   SASM
        move    mvar1, dword ptr [mvar2]
else
        mov     EAX, dword ptr [mvar2]
        mov     mvar1, EAX
endif
endm
```

- Define a variable using EQU
- Common uses: debugging code (only generated when debug flag is set), code for a specific machine (i.e., 80286), etc.

Copyright 2000, Douglas Reeves

Macro Operators

```
BUFDEFN2    macro  bufName:req, bufSz:req
buffer&bufName    db      bufSz dup (?)
endm
```

```
...
BUFDEFN2    Small, 20      ; define bufferSmall
...
BUFDEFN2    Large,2000     ; define bufferLarge
...
```

- The "substitute" operator (&)
  - "&parameter_name" means replace parameter_name with value of the parameter passed to the call
  - useful when "parameter_name" is embedded in a string

Copyright 2000, Douglas Reeves

Macro Operators (cont.)

```
STRDEFN     macro  strName:req, strVal:req
strName     db     strVal, 0
endm
```

```
…
STRDEFN     msg1, “Error occurred.”
STRDEFN     msg2, “Update performed successfully”
```

- The "literal-text" operator (<>)
  - "<string of some sort>" means pass the entire string (including blanks, punctuation, etc.) as a single parameter

Copyright 2000, Douglas Reeves

The IRP Directive

```
irp     regName, <EAX, EBX, ECX, EDX>
        inc     regName
Endm

;       same as....
;       inc     EAX
;       inc     EBX
;       inc     ECX
;       inc     EDX
```


Copyright 2000, Douglas Reeves

IRP (cont.)

- A way to expand the macro multiple times from a single call
- Each expansion differs in some fixed way
- Syntax:

```
irp     param_name, <paramlist...>

        <body using param_name>

endm
```

- Expand the macro once for each occurrence of a parameter value in the list

Copyright 2000, Douglas Reeves

Nesting IRP Inside Macro Definitions

```
INCREG macro        registers:req
  irp    regName, <registers>
         inc      regName
  endm
endm
```

```
...
         INCREG        <EAX, EDX>
....
         INCREG        <ECX, EDX>
...
```


Copyright 2000, Douglas Reeves

Drawback of Macros

- Makes symbolic debugging more difficult!

Copyright 2000, Douglas Reeves