```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/adi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .adi

        .text
.adi:
        pop     bx
        cmp     cx,#2
        jne     1f
        pop     cx
        add     ax,cx
        jmp     (bx)
1:
        cmp     cx,#4
        jne     9f
        pop     dx
        pop     cx
        add     ax,cx
        pop     cx
        adc     dx,cx
        push    dx
        jmp     (bx)
9:
.extern .trpilin
        push    bx
        jmp     .trpilin
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/and.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .and

        ! #bytes in cx
        ! save di; it might be a register variable

        .text
.and:
        pop     bx              ! return address
        mov     dx,di
        mov     di,sp
        add     di,cx
        sar     cx,#1
1:
        pop     ax
        and     ax,(di)
        stos
        loop    1b
        mov     di,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/blm.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .blm
.text

        ! cx: count in words
.blm:
        mov     bx,sp
        mov     ax,si
        mov     dx,di
        mov     di,2(bx)
        mov     si,4(bx)
        rep
        mov
        mov     si,ax
        mov     di,dx
        ret     4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/cii.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .cii

.text
.cii:
        pop     bx              ! return address
                                ! pop     cx, dest. size
                                ! pop     dx, src. size
                                ! ax is first word of source
        cmp     dx,#1
        jne     2f
        cbw
        mov     dx,#2
2:
        cmp     dx,cx
        je      8f
        cmp     dx,#2
        je      1f
        cmp     dx,#4
        jne     9f
        cmp     cx,#2
        jne     9f
        pop     dx
8:
        jmp     (bx)
1:
        cmp     cx,#4
        jne     9f
        cwd
        push    dx
        jmp     (bx)
```

```
9:
        push    ax              ! push low source
EILLINS = 18
.extern .fat
        mov     ax,#EILLINS
        push    ax
        jmp     .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/cmi4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .cmi4

.text
.cmi4:
        pop     bx              ! return address
        pop     cx
        pop     dx
        pop     ax
        push    si
        mov     si,sp
        xchg    bx,2(si)
        pop     si
        cmp     bx,dx
        jg      1f
        jl      2f
        cmp     ax,cx
        ja      1f
        je      3f
2:
        mov     ax,#-1
        ret
3:
        xor     ax,ax
        ret
1:
        mov     ax,#1
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/cms.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .cms

        ! #bytes in cx
        .text
.cms:
        pop     bx              ! return address
        mov     dx,sp
        push    si
        push    di
        mov     si,dx
```

```
        add     dx,cx
        mov     di,dx
        add     dx,cx
        sar     cx,#1
        repe
        cmp
        je      1f
        inc     cx
1:
        pop     di
        pop     si
        mov     sp,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/cmu4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .cmu4

.text
.cmu4:
        pop     bx              ! return address
        pop     cx
        pop     dx
        pop     ax
        push    si
        mov     si,sp
        xchg    bx,2(si)
        pop     si
        cmp     bx,dx
        ja      1f
        jb      2f
        cmp     ax,cx
        ja      1f
        je      3f
2:
        mov     ax,#-1
        ret
3:
        xor     ax,ax
        ret
1:
        mov     ax,#1
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/com.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .com

        ! #bytes in cx
        .text
.com:
        mov     bx,sp
        inc     bx
        inc     bx
        sar     cx,#1
1:
        not     (bx)
        inc     bx
        inc     bx
        loop    1b
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/csa2.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .csa2

.text
.csa2:
                                ! bx, descriptor address
                                ! ax, index
        mov     dx,(bx)         ! default
        sub     ax,2(bx)
        cmp     ax,4(bx)
        ja      1f
        sal     ax,#1
        add     bx,ax
        mov     bx,6(bx)
        test    bx,bx
        jnz     2f
1:
        mov     bx,dx
        test    bx,bx
        jnz     2f
ECASE = 20
.extern .fat
        mov     ax,#ECASE
        push    ax
        jmp     .fat
2:
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/csa4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.sect .text; .sect .rom; .sect .data; .sect .bss
.define .csa4

.sect .text
.csa4:
                                ! bx, descriptor address
                                ! ax, dx: index
        mov     cx,(bx)         ! default
        sub     ax,2(bx)
                                ! ignore high order word; if non-zero, the
                                ! case descriptor would not fit anyway
        cmp     ax,6(bx)
        ja      1f
2:
        sal     ax,#1
        add     bx,ax
        mov     bx,10(bx)
        test    bx,bx
        jnz     2f
1:
        mov     bx,cx
        test    bx,bx
        jnz     2f
ECASE = 20
.extern .fat
        mov     ax,#ECASE
        push    ax
        jmp     .fat
2:
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/csb2.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .csb2

.text
.csb2:
                                !bx, descriptor address
                                !ax,  index
        mov     dx,(bx)
        mov     cx,2(bx)
1:
        add     bx,#4
        dec     cx
        jl      4f
        cmp     ax,(bx)
        jne     1b
        mov     bx,2(bx)
```

```
2:
        test    bx,bx
        jnz     3f
ECASE = 20
.extern .fat
        mov     ax,#ECASE
        push    ax
        jmp     .fat
3:
        jmp     (bx)
4:
        mov     bx,dx
        jmp     2b
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/csb4.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.sect .text; .sect .rom; .sect .data; .sect .bss
.define .csb4

.sect .text
.csb4:
                                !bx: descriptor address
                                !ax, dx:  index
        push    (bx)            ! default
        mov     cx,2(bx)        ! count (ignore high order word, the descriptor
                                ! would not fit anyway)
1:
        add     bx,#6
        dec     cx
        jl      4f
        cmp     ax,(bx)
        jne     1b
        cmp     dx,2(bx)
        jne     1b
        pop     bx
        mov     bx,4(bx)
2:
        test    bx,bx
        jnz     3f
ECASE = 20
.extern .fat
        mov     ax,#ECASE
        push    ax
        jmp     .fat
3:
        jmp     (bx)
4:
        pop     bx
        jmp     2b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/cuu.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .ciu
.define .cui
.define .cuu

.text
.ciu:
.cui:
.cuu:
        pop     bx              ! return address
                                ! pop     cx, dest. size
                                ! pop     dx, source size
                                ! ax is low word of source
        cmp     dx,cx
        je      8f
        cmp     dx,#2
        je      1f
        cmp     dx,#4
        jne     9f
        cmp     cx,#2
        jne     9f
        pop     dx
8:
        jmp     (bx)
1:
        cmp     cx,#4
        jne     9f
        xor     dx,dx
        push    dx
        jmp     (bx)
9:
        push    ax              ! to help debugging ?
EILLINS = 18
.extern .fat
        mov     ax,#EILLINS
        push    ax
        jmp     .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/dup.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .dup

        ! #bytes in cx
        .text
.dup:
        pop     bx              ! return address
        mov     ax,si
        mov     dx,di
        mov     si,sp
        sub     sp,cx
```

```
        mov     di,sp
        sar     cx,#1
        rep
        mov
        mov     si,ax
        mov     di,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/em/dvi4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .dvi4

yl=6
yh=8
xl=10
xh=12

.text
.dvi4:
        push    si
        push    di
        mov     si,sp           ! copy of sp
        mov     bx,yl(si)
        mov     ax,yh(si)
        cwd
        mov     di,dx
        cmp     dx,ax
        jne     7f
        and     dx,dx
        jge     1f
        neg     bx
        je      7f
1:
        xor     dx,dx
        mov     cx,xl(si)
        mov     ax,xh(si)
        and     ax,ax
        jge     2f
        neg     ax
        neg     cx
        sbb     ax,dx
        not     di
2:
        div     bx
        xchg    ax,cx
        div     bx              ! cx = high abs(result), ax=low abs(result)
9:
        and     di,di
        jge     1f
        neg     cx
        neg     ax
        sbb     cx,#0
1:
                        ! cx is high order result
                        ! ax is low order result
```

```
        mov     dx,cx
        pop     di
        pop     si
        ret     8       ! result in ax/dx

7:
        push    dx              ! sign of y
        mov     di,ax
        xor     bx,bx
        and     di,di
        jge     1f
        neg     di
        neg     yl(si)
        sbb     di,bx
1:
        mov     ax,xl(si)
        mov     dx,xh(si)
        and     dx,dx
        jge     1f
        neg     dx
        neg     ax
        sbb     dx,bx
        not     -2(si)
1:
        mov     cx,#16
1:
        shl     ax,#1
        rcl     dx,#1
        rcl     bx,#1
        cmp     di,bx
        ja      3f
        jb      2f
        cmp     yl(si),dx
        jbe     2f
3:
        loop    1b
        jmp     1f
2:
        sub     dx,yl(si)
        sbb     bx,di
        inc     ax
        loop    1b
1:
        pop     di              ! di=sign of result,ax= result
        jmp     9b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/dvu4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .dvu4

yl=6
yh=8
xl=10
xh=12

.text
.dvu4:
        push    si
        push    di
        mov     si,sp           ! copy of sp
        mov     bx,yl(si)
        mov     ax,yh(si)
        or      ax,ax
        jne     7f
        xor     dx,dx
        mov     cx,xl(si)
        mov     ax,xh(si)
        div     bx
        xchg    ax,cx
        div     bx
9:
                        ! cx is high order result
                        ! ax is low order result
        mov     dx,cx
        pop     di
        pop     si
        ret     8       ! result in ax/dx

7:
        mov     di,ax
        xor     bx,bx
        mov     ax,xl(si)
        mov     dx,xh(si)
        mov     cx,#16
1:
        shl     ax,#1
        rcl     dx,#1
        rcl     bx,#1
        cmp     di,bx
        ja      3f
        jb      2f
        cmp     yl(si),dx
        jbe     2f
3:
        loop    1b
        jmp     9b
2:
        sub     dx,yl(si)
        sbb     bx,di
        inc     ax
        loop    1b
        jmp     9b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/error.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .error
.define .Xtrp

        ! ax is trap number
        ! all registers must be saved
        ! because return is possible
        ! May only be called with error no's <16
.text
.error:
        push bp
        push si
        push di
        push dx
        push cx
        push bx
        push ax
        mov  cx,ax
        mov  bx,#1
        sal  bx,cl
.extern .ignmask
.extern .trp
        test bx,.ignmask
        jne  2f
        call    .trp
2:
        pop  ax
        pop  bx
        pop  cx
        pop  dx
        pop  di
        pop  si
        pop  bp
        ret

.Xtrp:
        cmp     ax,#16
        jge     1f
        call    .error
        ret
1:
        jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/exg.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .exg

        ! #bytes in cx
.text
.exg:
        push    di
        mov     sp,di
        add     di,#4
        mov     bx,di
        add     bx,cx
        sar     cx,#1
1:
        mov     ax,(bx)
        xchg    ax,(di)
        mov     (bx),ax
        add     di,#2
        add     bx,#2
        loop    1b
2:
        pop     di
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/fat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .fat
.text

.fat:
.extern .trp
.extern .stop
        call    .trp
        call    .stop
        ! no return
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/fp8087.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .adf4, .adf8, .sbf4, .sbf8, .mlf4, .mlf8, .dvf4, .dvf8
.define .ngf4, .ngf8, .fif4, .fif8, .fef4, .fef8
.define .cif4, .cif8, .cuf4, .cuf8, .cfi, .cfu, .cff4, .cff8
.define .cmf4, .cmf8
.sect .text; .sect .rom; .sect .data; .sect .bss

!       $Header: fp8087.s,v 1.2 89/11/28 12:51:35 ceriel Exp $

!       Implement interface to floating point package for Intel 8087

```

```
        .sect .rom
one:
        .data2  1
two:
        .data2  2
bigmin:
        .data2  0, -32768

        .sect .text
.adf4:
        mov     bx,sp
        wait
        flds    2(bx)
        wait
        fadds   6(bx)
        wait
        fstps   6(bx)
        wait
        ret
.adf8:
        mov     bx,sp
        wait
        fldd    2(bx)
        wait
        faddd   10(bx)
        wait
        fstpd   10(bx)
        wait
        ret

.sbf4:
        mov     bx,sp
        wait
        flds    6(bx)
        wait
        fsubs   2(bx)
        wait
        fstps   6(bx)
        wait
        ret

.sbf8:
        mov     bx,sp
        wait
        fldd    10(bx)
        wait
        fsubd   2(bx)
        wait
        fstpd   10(bx)
        wait
        ret

.mlf4:
        mov     bx,sp
        wait
        flds    2(bx)
        wait
        fmuls   6(bx)
        wait
        fstps   6(bx)
```

```
        wait
        ret
.mlf8:
        mov     bx,sp
        wait
        fldd    2(bx)
        wait
        fmuld   10(bx)
        wait
        fstpd   10(bx)
        wait
        ret

.dvf4:
        mov     bx,sp
        wait
        flds    6(bx)
        wait
        fdivs   2(bx)
        wait
        fstps   6(bx)
        wait
        ret

.dvf8:
        mov     bx,sp
        wait
        fldd    10(bx)
        wait
        fdivd   2(bx)
        wait
        fstpd   10(bx)
        wait
        ret

.ngf4:
        mov     bx,sp
        wait
        flds    2(bx)
        wait
        fchs
        wait
        fstps   2(bx)
        wait
        ret

.ngf8:
        mov     bx,sp
        wait
        fldd    2(bx)
        wait
        fchs
        wait
        fstpd   2(bx)
        wait
        ret

.fif4:
        mov     bx,sp
        push    bx              ! make room for FP status word
```

```
        wait
        flds    4(bx)
        wait
        fmuls   8(bx)           ! multiply
        wait
        fld     st              ! copy result
        wait
        ftst                    ! test sign; handle negative separately
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf                    ! result of test in condition codes
        jb      1f
        frndint                 ! this one rounds (?)
        wait
        fcom    st(1)           ! compare with original; if <=, then OK
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        jbe     2f
        fisubs  one             ! else subtract 1
        wait
        jmp     2f
1:                              ! here, negative case
        frndint                 ! this one rounds (?)
        wait
        fcom    st(1)           ! compare with original; if >=, then OK
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        jae     2f
        fiadds  one             ! else add 1
        wait
2:
        fsub    st(1),st        ! subtract integer part
        wait
        mov     bx,2(bx)
        fstps   (bx)
        wait
        fstps   4(bx)
        wait
        pop     bx
        ret

.fif8:
        mov     bx,sp
        push    bx              ! make room for FP status word
        wait
        fldd    4(bx)
        wait
        fmuld   12(bx)          ! multiply
        wait
        fld     st              ! and copy result
        wait
        ftst                    ! test sign; handle negative separately
```

```
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf                    ! result of test in condition codes
        jb      1f
        frndint                 ! this one rounds (?)
        wait
        fcom    st(1)           ! compare with original; if <=, then OK
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        jbe     2f
        fisubs  one             ! else subtract 1
        wait
        jmp     2f
1:                              ! here, negative case
        frndint                 ! this one rounds (?)
        wait
        fcom    st(1)           ! compare with original; if >=, then OK
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        jae     2f
        fiadds  one             ! else add 1
        wait
2:
        fsub    st(1),st        ! subtract integer part
        mov     bx,2(bx)
        fstpd   (bx)
        wait
        fstpd   8(bx)
        wait
        pop     bx
        ret

.fef4:
                                ! this could be simpler, if only the
                                ! fxtract instruction was emulated properly
        mov     bx,sp
        mov     ax,6(bx)
        and     ax,#077600
        je      1f              ! zero exponent
        mov     cx,#7
        shr     ax,cl
        sub     ax,#126
        mov     cx,ax           ! exponent in cx
        mov     ax,6(bx)
        and     ax,#0100177
        or      ax,#0037400     ! load -1 exponent
        mov     dx,4(bx)
        mov     bx,2(bx)
        mov     4(bx),ax
        mov     2(bx),dx
        mov     (bx),cx
        ret
```

```
1:                              ! we get here on zero exp
        mov     ax,6(bx)
        and     ax,#0177
        or      ax,4(bx)
        jne     1f              ! zero result
        xor     ax,ax
        mov     bx,2(bx)
        mov     (bx),ax
        mov     2(bx),ax
        mov     4(bx),ax
        ret
1:                              ! otherwise unnormalized number
        mov     cx,6(bx)
        and     cx,#0100177
        mov     dx,cx
        and     cx,#0x8000
        mov     ax,#-125
2:
        test    dx,#0x80
        jne     1f
        dec     ax
        shl     4(bx),#1
        rcl     dx,#1
        or      dx,cx
        jmp     2b
1:
        mov     cx,4(bx)
        mov     bx,2(bx)
        mov     (bx),ax
        mov     2(bx),cx
        and     dx,#0100177
        or      dx,#0037400     ! load -1 exponent
        mov     4(bx),dx
        ret

.fef8:
                                ! this could be simpler, if only the
                                ! fxtract instruction was emulated properly
        mov     bx,sp
        mov     ax,10(bx)
        and     ax,#077760
        je      1f              ! zero exponent
        mov     cx,#4
        shr     ax,cl
        sub     ax,#1022
        mov     cx,ax           ! exponent in cx
        mov     ax,10(bx)
        and     ax,#0100017
        or      ax,#0037740     ! load -1 exponent
        push    8(bx)
        push    6(bx)
        push    4(bx)
        mov     bx,2(bx)
        pop     2(bx)
        pop     4(bx)
        pop     6(bx)
        mov     8(bx),ax
        mov     (bx),cx
        ret
1:                              ! we get here on zero exp
```

```
        mov     ax,10(bx)
        and     ax,#017
        or      ax,8(bx)
        or      ax,6(bx)
        or      ax,4(bx)
        jne     1f              ! zero result
        xor     ax,ax
        mov     bx,2(bx)
        mov     (bx),ax
        mov     2(bx),ax
        mov     4(bx),ax
        mov     6(bx),ax
        mov     8(bx),ax
        ret
1:                              ! otherwise unnormalized number
        mov     cx,10(bx)
        and     cx,#0100017
        mov     dx,cx
        and     cx,#0x8000
        mov     ax,#-1021
2:
        test    dx,#0x10
        jne     1f
        dec     ax
        shl     4(bx),#1
        rcl     6(bx),#1
        rcl     8(bx),#1
        rcl     dx,#1
        or      dx,cx
        jmp     2b
1:
        and     dx,#0100017
        or      dx,#0037740     ! load -1 exponent
        mov     cx,8(bx)
        push    6(bx)
        push    4(bx)
        mov     bx,2(bx)
        mov     (bx),ax
        mov     8(bx),dx
        mov     6(bx),cx
        pop     2(bx)
        pop     4(bx)
        ret

.cif4:
        mov     bx,sp
        cmp     2(bx),#2
        jne     1f
        wait
        filds   4(bx)
        wait
        fstps   2(bx)
        wait
        ret
1:
        wait
        fildl   4(bx)
        wait
        fstps   4(bx)
        wait
```

```
        ret

.cif8:
        mov     bx,sp
        cmp     2(bx),#2
        jne     1f
        wait
        filds   4(bx)
        wait
        fstpd   2(bx)
        wait
        ret
1:
        wait
        fildl   4(bx)
        wait
        fstpd   2(bx)
        wait
        ret

.cuf4:
        mov     bx,sp
        cmp     2(bx),#2
        jne     1f
        mov     ax,4(bx)
        mov     2(bx),ax
        mov     4(bx),#0
        wait
        fildl   2(bx)
        wait
        fstps   2(bx)
        wait
        ret
1:
        wait
        fildl   4(bx)
        wait
        cmp     6(bx),#0
        jge     1f
2:
        wait
        fisubl  bigmin
        wait
        fisubl  bigmin
1:
        wait
        fstps   4(bx)
        wait
        ret

.cuf8:
        mov     bx,sp
        cmp     2(bx),#2
        jne     1f
        mov     6(bx),#0
1:
        wait
        fildl   4(bx)
        wait
        cmp     6(bx),#0
```

```
        jge     1f
2:
        wait
        fisubl  bigmin
        wait
        fisubl  bigmin
1:
        wait
        fstpd   2(bx)
        wait
        ret

.cfi:
        mov     bx,sp
        push    bx
        wait
        fstcw   -2(bx)
        wait
        mov     dx,-2(bx)
        or      -2(bx),#0xc00   ! truncating mode
        wait
        fldcw   -2(bx)
        pop     ax
        cmp     4(bx),#4
        jne     2f
                                ! loc 4 loc ? cfi
        wait
        flds    6(bx)
        wait
        fistpl  6(bx)
        wait
        cmp     2(bx),#2
        jne     1f
        mov     ax,6(bx)
1:
        mov     4(bx),dx
        wait
        fldcw   4(bx)
        wait
        ret
2:
                                ! loc 8 loc ? cfi
        wait
        fldd    6(bx)
        wait
        fistpl  10(bx)
        wait
        cmp     2(bx),#2
        jne     1b
        mov     ax,10(bx)
        jmp     1b

.cfu:
        mov     bx,sp
        push    bx
        wait
        fstcw   -2(bx)
        wait
        mov     dx,-2(bx)
        and     -2(bx),#0xf3ff
```

```
        or      -2(bx),#0x400   ! to -infinity
        wait
        fldcw   -2(bx)
        wait
        pop     ax
        cmp     4(bx),#4
        jne     2f
                                ! loc 4 loc ? cfu
        flds    6(bx)
        wait
        fabs                    ! ???
        wait
        fiaddl  bigmin
        fistpl  6(bx)
        wait
        mov     ax,8(bx)
        sub     ax,bigmin+2
        mov     8(bx),ax
        cmp     2(bx),#2
        jne     1f
        mov     ax,6(bx)
1:
        mov     4(bx),dx
        wait
        fldcw   4(bx)
        wait
        ret
2:
        wait
                                ! loc 8 loc ? cfu
        fldd    6(bx)
        wait
        fabs                    ! ???
        wait
        fiaddl  bigmin
        fistpl  10(bx)
        wait
        mov     ax,12(bx)
        sub     ax,bigmin+2
        mov     12(bx),ax
        cmp     2(bx),#2
        jne     1b
        mov     ax,10(bx)
        jmp     1b

.cff4:
        mov     bx,sp
        wait
        fldd    2(bx)
        wait
        fstcw   2(bx)
        wait
        mov     dx,2(bx)
        and     2(bx),#0xf3ff   ! set to rounding mode
        wait
        fldcw   2(bx)
        wait
        fstps   6(bx)
        mov     2(bx),dx
        wait
```

```
        fldcw   2(bx)
        wait
        ret

.cff8:
        mov     bx,sp
        wait
        flds    2(bx)
        wait
        fstpd   2(bx)
        wait
        ret

.cmf4:
        mov     bx,sp
        push    bx              ! room for 8087 status word
        xor     cx,cx
        wait
        flds    6(bx)
        wait
        flds    2(bx)
        wait
        fcompp                  ! compare and pop operands
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        je      1f
        jb      2f
        dec     cx
        jmp     1f
2:
        inc     cx
1:
        mov     ax,cx
        pop     bx
        ret


.cmf8:
        mov     bx,sp
        push    bx              ! room for 8087 status word
        xor     cx,cx
        wait
        fldd    10(bx)
        wait
        fldd    2(bx)
        wait
        fcompp                  ! compare and pop operands
        wait
        fstsw   -2(bx)
        wait
        mov     ax,-2(bx)
        sahf
        je      1f
        jb      2f
        dec     cx
        jmp     1f
2:
```

```
        inc     cx
1:
        mov     ax,cx
        pop     bx
        ret
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/em/gto.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .gto
.text

.gto:
        mov     bp,4(bx)
        mov     sp,2(bx)
        jmp     @(bx)
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/em/iaar.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .iaar
.text

.iaar:
        pop     cx
        pop     dx
        cmp     dx,#2
.extern .unknown
        jne     .unknown
        pop     bx      ! descriptor address
        pop     ax      ! index
        sub     ax,(bx)
        mul     4(bx)
        pop     bx      ! array base
        add     bx,ax
        push    cx
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/ilar.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .ilar
.text

.ilar:
        pop     cx
        pop     dx
.extern .unknown
        cmp     dx,#2
        jne     .unknown
        pop     bx      ! descriptor address
        pop     ax      ! index
        push    cx
.extern .lar2
        jmp    .lar2
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/inn.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .inn
.text

        ! #bytes in cx
        ! bit # in ax
.inn:
        xor     dx,dx
        mov     bx,#8
        div     bx
        mov     bx,sp
        add     bx,#2
        add     bx,ax
        cmp     ax,cx
        jae     1f
        movb    al,(bx)
        mov     bx,dx
        testb   al,bits(bx)
        jz      1f
        mov     ax,#1
        jmp     2f
1:
        xor     ax,ax
2:
        pop     bx
        add     sp,cx
        ! ax is result
        jmp     (bx)

        .data
bits:
        .data1 1,2,4,8,16,32,64,128
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/ior.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .ior
.text
        ! #bytes in cx
.ior:
        pop     bx              ! return address
        mov     dx,di
        mov     di,sp
        add     di,cx
        sar     cx,#1
1:
        pop     ax
        or      ax,(di)
        stos
        loop    1b
        mov     di,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/isar.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .isar
.text
.isar:
        pop     cx
        pop     ax
        cmp     ax,#2
.extern .unknown
        jne     .unknown
        pop     bx      ! descriptor address
        pop     ax      ! index
        push    cx
.extern .sar2
        jmp    .sar2
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/lar2.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .lar2
.text
.lar2:
                                ! bx, descriptor address
                                ! ax, index
        pop     cx
        pop     dx              ! base address
        push    cx
        push    si
```

```
        mov     si,dx
        sub     ax,(bx)
        mov     cx,4(bx)
        imul    cx
        add     si,ax
        sar     cx,#1
        jnb     1f
        xorb    ah,ah
        lodsb
        pop     si
        pop     bx
        push    ax
        jmp     (bx)
1:
        pop     dx                      ! saved si
        mov     ax,4(bx)
        pop     bx                      ! return address
        sub     sp,ax
        mov     ax,di                   ! save di
        mov     di,sp
        rep
        mov
        mov     di,ax
        mov     si,dx
        jmp     (bx)
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                              src/lib/i86/em/lfr6.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .lfr6
.text
.extern .retarea

.lfr6:
        pop     bx
        push    .retarea+4
        push    .retarea+2
        push    .retarea
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/lfr8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .lfr8
.text
.extern .retarea

.lfr8:
        pop     bx
        push    .retarea+6
        push    .retarea+4
        push    .retarea+2
        push    .retarea
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/loi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .loi
.define .los
.text

        ! #bytes in cx
        ! address in bx
        ! save si/di. they might be register variables
.los:
        mov     dx,si
        mov     si,bx
        pop     bx
        mov     ax,cx
        sar     cx,#1
        jnb     1f
        xorb    ah,ah
        lodsb
        mov     si,dx
        push    ax
        jmp     (bx)
1:
        sub     sp,ax
        jmp     1f

.loi:
        ! only called with size > 4
        mov     dx,si
        mov     si,bx
        pop     bx
        sub     sp,cx
        sar     cx,#1
1:
        mov     ax,di
        mov     di,sp
        rep
        mov
```

```
        mov     si,dx
        mov     di,ax
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/mli4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .mli4
.text

yl=2
yh=4
        ! x * y
        ! xl in ax
        ! xh in dx

.mli4:
        mov     bx,sp
        push    dx
        mov     cx,ax
        mul     yh(bx)           ! xl*yh
        pop     dx
        push    ax
        mov     ax,dx
        mul     yl(bx)          ! xh * yl
        pop     dx
        add     dx,ax           ! xh*yl+xl*yh
        mov     ax,cx
        mov     cx,dx
        mul     yl(bx)           ! xl*yl
        add     dx,cx
        ret     4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/mon.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .mon
.text

.mon:
.extern .stop
        call    .stop
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/nop.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .nop
.text
.nop:
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/rck.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .rck
.text

        ! descriptor address in bx
        ! value in ax, must be left there
.rck:
        cmp     ax,(bx)
        jl      2f
        cmp     ax,2(bx)
        jg      2f
        ret
2:
        push    ax
ERANGE = 1
.extern .error
        mov     ax,#ERANGE
        call    .error
        pop     ax
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/ret6.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .ret6
.text
.extern .retarea

.ret6:
        pop     bx
        pop     .retarea
        pop     .retarea+2
        pop     .retarea+4
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/ret8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .ret8
.text
.extern .retarea

.ret8:
        pop     bx
        pop     .retarea
        pop     .retarea+2
        pop     .retarea+4
        pop     .retarea+6
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/retarea.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .retarea

.bss
.retarea:
        .zerow 8/2
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/return.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .sdret, .dsret, .sret, .dret, .cret
.text

.dsret:
        pop     di
.sret:
        pop     si
.cret:
        mov     sp,bp
        pop     bp
        ret

.sdret:
        pop     si
.dret:
        pop     di
        jmp     .cret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/rmi4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .rmi4
.text

yl=6
yh=8
xl=10
xh=12

.rmi4:
        push    si
        push    di
        mov     si,sp           ! copy of sp
        mov     bx,yl(si)
        mov     ax,yh(si)
        cwd
        cmp     dx,ax
        jne     7f
        and     dx,dx
        jge     1f
        neg     bx
        je      7f
1:
        xor     dx,dx
        mov     cx,xl(si)
        mov     ax,xh(si)
        and     ax,ax
        jge     2f
        neg     ax
        neg     cx
        sbb     ax,dx
2:
        div     bx
        xchg    ax,cx
        div     bx              ! dx= result(low), 0=result(high)
        xor     bx,bx
9:
        cmp     xh(si),#0
        jge     1f
        neg     bx
        neg     dx
        sbb     bx,#0
1:
                        ! bx is high order result
                        ! dx is low order result
        mov     ax,dx
        mov     dx,bx   ! result in ax/dx
        pop     di
        pop     si
        ret     8

7:
        mov     di,ax
        xor     bx,bx
        and     di,di
        jge     1f
```

```
        neg     di
        neg     yl(si)
        sbb     di,bx
1:
        mov     ax,xl(si)
        mov     dx,xh(si)
        and     dx,dx
        jge     1f
        neg     dx
        neg     ax
        sbb     dx,bx
1:
        mov     cx,#16
1:
        shl     ax,#1
        rcl     dx,#1
        rcl     bx,#1
        cmp     di,bx
        ja      3f
        jb      2f
        cmp     yl(si),dx
        jbe     2f
3:
        loop    1b
                ! dx=result(low), bx=result(high)
        jmp     9b
2:
        sub     dx,yl(si)
        sbb     bx,di
        inc     ax
        loop    1b
1:
                ! dx=result(low), bx=result(high)
        jmp     9b
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/rmu4.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .rmu4
.text

yl=6
yh=8
xl=10
xh=12

.rmu4:
        push    si
        push    di
        mov     si,sp           ! copy of sp
        mov     bx,yl(si)
        mov     ax,yh(si)
        or      ax,ax
        jne     7f
1:
        xor     dx,dx
        mov     cx,xl(si)
        mov     ax,xh(si)
```

```
2:
        div     bx
        xchg    ax,cx
        div     bx
        xor     bx,bx
9:
                        ! bx is high order result
                        ! dx is low order result
        mov     ax,dx
        mov     dx,bx
        pop     di
        pop     si
        ret     8       ! result in ax/dx

7:
        mov     di,ax
        xor     bx,bx
        mov     ax,xl(si)
        mov     dx,xh(si)
        mov     cx,#16
1:
        shl     ax,#1
        rcl     dx,#1
        rcl     bx,#1
        cmp     di,bx
        ja      3f
        jb      2f
        cmp     yl(si),dx
        jbe     2f
3:
        loop    1b
                ! dx=result(low), bx=result(high)
        jmp     9b
2:
        sub     dx,yl(si)
        sbb     bx,di
        inc     ax
        loop    1b
1:
                ! dx=result(low), bx=result(high)
        jmp     9b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/em/sar2.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .sar2
.text

.sar2:
                                ! bx, descriptor address
                                ! ax, index
        pop     cx
        pop     dx              ! base address
        push    cx
        xchg    di,dx           ! di = base address, dx is saved di
```

```
        sub     ax,(bx)
        mov     cx,4(bx)
        push    dx
        imul    cx
        pop     dx
        add     di,ax
        sar     cx,#1
        jnb     1f
        pop     bx
        pop     ax
        stosb
        mov     di,dx
        jmp     (bx)
1:
        pop     bx
        mov     ax,si
        mov     si,sp
        rep
        mov
        mov     sp,si
        mov     si,ax
        mov     di,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/sbi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .sbi
.text

        ! #bytes in cx , top of stack in ax
.sbi:
        pop     bx              ! return subress
        cmp     cx,#2
        jne     1f
        pop     cx
        sub     ax,cx
        neg     ax
        jmp     (bx)
1:
        cmp     cx,#4
        jne     9f
        pop     dx
        pop     cx
        sub     cx,ax
        mov     ax,cx
        pop     cx
        sbb     cx,dx
        push    cx
        jmp     (bx)
9:
.extern .trpilin
        push    bx
        jmp     .trpilin
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                          src/lib/i86/em/set.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .set
.text

        ! #bytes in cx
        ! bit # in ax
.set:
        pop     bx              ! return address
        xor     dx,dx
!ifdef create set
        sub     sp,cx
        push    bx
        push    di
        mov     bx,sp
        xor     di,di
        sar     cx,#1
1:
        mov     4(bx)(di),dx
        inc     di
        inc     di
        loop    1b
!endif
        mov     bx,#8
        div     bx
        cmp     ax,di
        jae     2f
        mov     di,dx
        movb    dl,bits(di)
        mov     di,sp
        add     di,ax
        orb     4(di),dl
        pop     di
        ret
2:
ESET = 2
.extern .error
        pop     di
        mov     ax,#ESET
        call    .error
        ret

        .data
bits:
        .data1   1,2,4,8,16,32,64,128
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/stb.c
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

/*
 * (c) copyright 1987 by the Vrije Universiteit, Amsterdam, The Netherlands.
 * See the copyright notice in the ACK home directory, in the file "Copyright".
 */
/* $Header: stb.c,v 1.1 89/02/07 11:01:46 ceriel Exp $ */
/* library routine for copying structs */

#include <ansi.h>

_PROTOTYPE(int __stb, (int n, char *f, char *t ));
__stb(n, f, t)
register int n;
register char *f, *t;
{
        if (n > 0)
                do
                        *t++ = *f++;
                while (--n);
}
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/sti.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define .sti
.define .sts
.text

        ! #bytes in cx
        ! address in bx
        ! save di/si. they might be register variables
.sts:
        mov     dx,di           ! save di
        mov     di,bx
        pop     bx              ! return address
        sar     cx,#1
        jnb     1f
        pop     ax
        stosb
        mov     di,dx
        jmp     (bx)
.sti:
        ! only called with count > 4
        mov     dx,di
        mov     di,bx
        pop     bx
        sar     cx,#1
1:
        mov     ax,si
        mov     si,sp
        rep
        mov
        mov     sp,si
        mov     di,dx
```

```
        mov     si,ax
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/stop.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .stop
.text
.stop:
        call    __exit
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/strhp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .strhp
.text
.extern .reghp, .limhp, .fat
EHEAP = 17

.strhp:
        pop     bx
        pop     ax
        mov     .reghp,ax
        cmp     ax,.limhp
        jb      1f
        add     ax,#01000
        and     ax,#0177000
        mov     .limhp,ax
        cmp     ax,sp
        jae     2f
1:
        jmp     (bx)
2:
        mov     ax,#EHEAP
        jmp     .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/trp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .trpdivz
.define .trpilin
.define .trpcase
.define .trprang
.define .trpset
.define .trpnofp
.define .trpheap
.define .trp

.bss
```

```
.M: .zerow 24/2

.text
.extern .trpdivz
.extern .trpilin
.extern .trpcase
.extern .trprang
.extern .trpset
.extern .trpnofp
.extern .trpheap
.extern .trp

.trpdivz:
mov ax,#6
mov dx,#.Mdivz
jmp .Trp
.trpilin:
mov ax,#18
mov dx,#.Milin
jmp .Trp
.trpcase:
mov ax,#20
mov dx,#.Mcase
jmp .Trp
.trprang:
mov ax,#1
mov dx,#.Mrang
jmp .Trp
.trpset:
mov ax,#2
mov dx,#.Mset
jmp .Trp
.trpnofp:
mov ax,#18
mov dx,#.Mnofp
jmp .Trp
.trpheap:
mov ax,#17
mov dx,#.Mheap
jmp .Trp

.Trp:
xor     bx,bx
.extern .trppc
xchg    bx,.trppc
test    bx,bx
jz      2f
push    ax
call    (bx)
pop     ax
ret
2:
mov bx,#22
push bx
push dx
mov ax,#2
push ax
call .Write
call __exit
```

```
.trp:
mov dx,ax
cmp dx,#21
jae 1f
sal dx,#1
mov bx,#.Mtable
add bx,dx
mov bx,(bx)
test bx,bx
jz 1f
mov dx,bx
jmp 2f
1:
mov bx,#.Mtrp+14
mov cx,#6
mov dx,ax
1:
and dx,#7
add dx,'0'
movb (bx),dl
dec bx
sar dx,#1
sar dx,#1
sar dx,#1
loop 1b
mov dx,#.Mtrp
2:
jmp .Trp

.Write:
push bp
mov bp,sp
mov .M+2,#4
mov bx,4(bp)
mov .M+4,bx
mov bx,8(bp)
mov .M+6,bx
mov bx,6(bp)
mov .M+10,bx
mov ax,#.M
push ax
mov ax,#1
push ax

mov ax,#1
mov bx,#.M
mov cx,#3
int 32
mov sp,bp
pop bp
ret


.data
.Mtable:
        .data2 0,       .Mrang, .Mset,  0,      0,      0,      .Mdivz, 0
        .data2 0,       0,      0,      0,      0,      0,      0,      0
        .data2 0,       .Mheap, .Milin, .Milin, .Mcase

.Mdivz: .asciz "Error: Division by 0 \n"
```

```
.Milin: .asciz "Illegal EM instruct'n\n"
.Mcase: .asciz "Err in EM case instr \n"
.Mrang: .asciz "Variable out of range\n"
.Mset:  .asciz "Err in EM set instr  \n"
.Mnofp: .asciz "Floating pt not impl.\n"
.Mheap: .asciz "Heap overflow        \n"

.Mtrp:  .asciz "EM trap 0000000 octal\n"
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/unknown.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .unknown
.text
.extern .fat
EILLINS = 18

.unknown:
        mov  ax,#EILLINS
        push ax
        jmp  .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/vars.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .reghp, .limhp, .ignmask, .trppc
.extern .reghp, .limhp, .ignmask, .trppc

.data
.reghp:
        .data2   endbss
.limhp:
        .data2   endbss
.ignmask:
        .data2   1336
.trppc:
        .data2   0
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/em/xor.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
.define .xor
.text

        ! #bytes in cx
.xor:
        pop     bx              ! return address
        mov     dx,di
        mov     di,sp
        add     di,cx
        sar     cx,#1
```

```
1:
        pop     ax
        xor     ax,(di)
        stos
        loop    1b
        mov     di,dx
        jmp     (bx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/misc/alloca.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
#
!       alloca() - allocate space on the stack          Author: Kees J. Bot
!                                                               26 Jan 1994

#if __ACK__     /* BCC can't do alloca(), register saving is wrong. */

.text
.define _alloca
_alloca:
        pop     cx              ! Return address
        pop     ax              ! Bytes to allocate
        add     ax, #2*2+1      ! Add space for two saved register variables
        andb    al, #0xFE       ! Align
        mov     bx, sp          ! Keep current sp
        sub     sp, ax          ! Lower stack
        mov     ax, sp          ! Return value
        push    2(bx)           ! Push what is probably the saved si
        push    (bx)            ! Saved di
                                ! Now ACK can still do:
                                !       pop di; pop si; mov sp, bp; pop bp; ret
        push    ax              ! Dummy argument
        jmp     (cx)
#endif
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/misc/getprocessor.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       getprocessor() - determine processor type       Author: Kees J. Bot
!                                                               26 Jan 1994

.text

        o32 = 0x66              ! 32 bit operand size prefix

! int getprocessor(void);
!       Return 86, 186, 286, 386, 486, 586, ...

.define _getprocessor

_getprocessor:
        push    bp
        mov     bp, sp
```

```
        push    sp              ! see if pushed sp == sp
        pop     ax
        cmp     ax, sp
        jz      new_processor
        mov     cx, #0x0120     ! see if shifts are mod 32
        shlb    ch, cl          ! zero tells if 86
        mov     ax, #86
        jz      got_processor
        mov     ax, #186
        jmp     got_processor

new_processor:                  ! see if high bits are set in saved GDT
        sub     sp, #6          ! space for GDT ptr
        .data1  0x0F            ! Prefix for 286 instruction: sgdt -6(bp)
        add     -6(bp), ax      ! save 3 word GDT ptr (This is NOT an add!)
        cmpb    -1(bp), #0      ! top byte of GDT ptr is zero only for 386
        mov     ax, #286
        jnz     got_processor

! 386, 486, 586
        and     sp, #0xFFFC     ! Align stack to avoid AC fault (needed?)
        mov     cx, #0x0004     ! Try to flip the AC bit introduced on the 486
        call    flip
        mov     ax, #386        ! 386 if it didn't react to "flipping"
        jz      got_processor
        mov     cx, #0x0020     ! Try to flip the ID bit introduced on the 586
        call    flip
        mov     ax, #486        ! 486 if it didn't react
        jz      got_processor
        .data1  o32
        pushf
        .data1  o32
        pusha                   ! Save the world
        .data1  o32
        xor     ax, ax
        inc     ax              ! eax = 1
        .data1  0x0F, 0xA2      ! CPUID instruction tells the processor type
        andb    ah, #0x0F       ! Extract the family (5, 6, ...)
        movb    al, ah
        movb    ah, #100
        mulb    ah              ! 500, 600, ...
        add     ax, #86         ! 586, 686, ...
        mov     bx, sp
        mov     7*4(bx), ax     ! Pass ax through
        .data1  o32
        popa
        .data1  o32
        popf

got_processor:
        mov     sp, bp
        pop     bp
        ret

flip:
        push    bx              ! Save bx and realign stack to multiple of 4
        .data1  o32             ! About to operate on a 32 bit object
        pushf                   ! Push eflags
        pop     ax
        pop     dx              ! dx:ax = eflags
```

```
        mov     bx, dx          ! Save original eflags (high word only)
        xor     dx, cx          ! Flip the bit to test
        push    dx
        push    ax              ! Push modified eflags value
        .data1  o32
        popf                    ! Load modified eflags register
        .data1  o32
        pushf
        pop     ax
        pop     dx              ! Get it again
        push    bx
        push    ax
        .data1  o32
        popf                    ! Restore original eflags register
        xor     dx, bx          ! See if the bit changed
        test    dx, cx
        pop     bx              ! Restore bx
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/misc/iolib.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
! Some I/O related routines like:
!       unsigned char inb(int port);
!       unsigned short inw(int port);
!       void outb(int port, int value);
!       void outw(int port, int value);
!       void rep_inb(int port, unsigned char *buf, size_t count);
!       void rep_inw(int port, unsigned short *buf, size_t count);
!       void rep_outb(int port, unsigned char *buf, size_t count);
!       void rep_outw(int port, unsigned short *buf, size_t count);
!       void intr_enable(void);
!       void intr_disable(void);

.text
.define _inb
_inb:
        push    bp
        mov     bp, sp
        mov     dx, 4(bp)               ! port
        inb     dx                      ! read 1 byte
        xorb    ah, ah
        pop     bp
        ret

.define _inw
_inw:
        push    bp
        mov     bp, sp
        mov     dx, 4(bp)               ! port
        in      dx                      ! read 1 word
        pop     bp
        ret

.define _outb
_outb:
        push    bp
```

```
        mov     bp, sp
        mov     dx, 4(bp)               ! port
        mov     ax, 4+2(bp)             ! value
        outb    dx                      ! output 1 byte
        pop     bp
        ret

.define _outw
_outw:
        push    bp
        mov     bp, sp
        mov     dx, 4(bp)               ! port
        mov     ax, 4+2(bp)             ! value
        out     dx                      ! output 1 word
        pop     bp
        ret

.define _rep_inb
_rep_inb:
        push    bp
        mov     bp, sp
        push    di
        mov     dx, 4(bp)               ! port
        mov     di, 6(bp)               ! buf
        mov     cx, 8(bp)               ! byte count
    rep inb     dx                      ! input many bytes
        pop     di
        pop     bp
        ret

.define _rep_inw
_rep_inw:
        push    bp
        mov     bp, sp
        push    di
        mov     dx, 4(bp)               ! port
        mov     di, 6(bp)               ! buf
        mov     cx, 8(bp)               ! byte count
        shr     cx, #1                  ! word count
    rep in      dx                      ! input many words
        pop     di
        pop     bp
        ret

.define _rep_outb
_rep_outb:
        push    bp
        mov     bp, sp
        push    si
        mov     dx, 4(bp)               ! port
        mov     si, 6(bp)               ! buf
        mov     cx, 8(bp)               ! byte count
    rep outb    dx                      ! output many bytes
        pop     si
        pop     bp
        ret

.define _rep_outw
_rep_outw:
        push    bp
```

```
        mov     bp, sp
        push    si
        mov     dx, 4(bp)               ! port
        mov     si, 6(bp)               ! buf
        mov     cx, 8(bp)               ! byte count
        shr     cx, #1                  ! word count
    rep out     dx                      ! output many words
        pop     si
        pop     bp
        ret

.define _intr_disable
_intr_disable:
        push    bp
        mov     bp, sp
        cli
        pop     bp
        ret

.define _intr_enable
_intr_enable:
        push    bp
        mov     bp, sp
        sti
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/rts/__sigreturn.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
! This routine is the low-level code for returning from signals.
! It calls __sigreturn, which is the normal "system call" routine.
! Both ___sigreturn and __sigreturn are needed.
.sect .text; .sect .rom; .sect .data; .sect .bss
.sect .text
.define ___sigreturn
.extern __sigreturn
___sigreturn:
        add sp, #8
        jmp __sigreturn
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/rts/_sendrec.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define __send, __receive, __sendrec

! See ../h/com.h for C definitions
SEND = 1
RECEIVE = 2
BOTH = 3
SYSVEC = 32

!*========================================================================*
!                            _send and _receive                           *
!*========================================================================*
! _send(), _receive(), _sendrec() all save bp, but destroy ax, bx, and cx.
.extern __send, __receive, __sendrec
__send: mov cx,*SEND            ! _send(dest, ptr)
        jmp L0

__receive:
        mov cx,*RECEIVE         ! _receive(src, ptr)
        jmp L0

__sendrec:
        mov cx,*BOTH            ! _sendrec(srcdest, ptr)
        jmp L0

  L0:   push bp                 ! save bp
        mov bp,sp               ! can't index off sp
        mov ax,4(bp)            ! ax = dest-src
        mov bx,6(bp)            ! bx = message pointer
        int SYSVEC              ! trap to the kernel
        pop bp                  ! restore bp
        ret                     ! return

```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/rts/brksize.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.define __brksize
.data
.extern endbss, __brksize
__brksize: .data2 endbss
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/rts/ncrtso.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
! This is the C run-time start-off routine.  It's job is to take the
! arguments as put on the stack by EXEC, and to parse them and set them up the
! way _main expects them.
! It also initializes _environ when this variable isn't defined by the
! programmer.  The detection of whether _environ belong to us is rather
! simplistic.  We simply check for some magic value, but there is no other
! way.

.extern _main, _exit, crtso, __penvp
.extern begtext, begdata, begbss, endtext, enddata, endbss
.text
begtext:
crtso:          mov     bx,sp
                mov     cx,(bx)
                add     bx,*2
                mov     ax,cx
                inc     ax
                shl     ax,#1
                add     ax,bx
                mov     __penvp,ax      ! save envp in __envp

                ! Test whether address of environ < address of end.
                ! This is done for separate I&D systems.
                mov     dx,#_environ
                cmp     dx,#__end
                jae     1f
                cmp     _environ,#21331         ! is it our _environ?
                jne     1f
                mov     _environ,ax
1:
                push    ax      ! push environ
                push    bx      ! push argv
                push    cx      ! push argc
                xor     bp,bp   ! clear bp for traceback of core files
                call    _main
                add     sp,*6
                push    ax      ! push exit status
                call    _exit

.data
begdata:
__penvp:        .data2 0
.bss
begbss:
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/rts/setjmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.sect .text; .sect .data; .sect .bss
.sect .bss
.comm _gtobuf, 12
.extern ___setjmp
.extern _sigprocmask
.extern __longjerr
.extern __sigjmp
.extern _printf, _exit
.extern .blm, .sti, .cret
.sect .text
!
! Warning:  this code depends on the C language declaration of
! jmp_buf in <setjmp.h>, and on the definitions of the flags
! SC_ONSTACK, SC_SIGCONTEXT, and SC_NOREGLOCALS in <sys/sigcontext.h>.
SC_SIGCONTEXT=2
SC_NOREGLOCALS=4
!
! ___setjmp is called with two arguments.  The third argument is pushed
! by compiler generated code transparently to the caller.
!
! ___setjmp(jmp_buf, savemask, retaddr)
!
___setjmp:
        push    bp
        mov     bp,sp

        push    0(bp)           ! frame pointer
        lea     ax, 4(bp)
        push    ax              ! stack pointer
        push    8(bp)           ! program counter

        xor     ax, ax
        push    ax              ! signal mask high
        push    ax              ! signal mask low
        mov     ax,#SC_NOREGLOCALS      ! flags (4 is SC_NOREGLOCALS)
        push    ax

        mov     ax, 6(bp)       ! get the savemask arg
        cmp     ax, #0
        je      nosigstate      ! don't save signal state

        or      -12(bp), #SC_SIGCONTEXT         ! flags |= SC_SIGCONTEXT

        lea     ax, -10(bp)
        push    ax
        push    #0
        push    #0
        call    _sigprocmask    ! fill in signal mask
        add     sp, #6

nosigstate:
        mov     bx, 4(bp)       ! jmp_buf
        mov     cx, #12         ! sizeof(jmp_buf)
        call    .sti            ! sp = src addr, bx = dest addr, cx = count
        xor     ax, ax
```

```
        jmp     .cret

.extern __longjmp
__longjmp:
        push    bp
        mov     bp, sp

! Copy the jmp_buf to the gtobuf.
        push    4(bp)                   ! src address
        mov     ax, #_gtobuf
        push    ax                      ! dst address
        mov     cx, #6                  ! word count
        call    .blm                    ! block move

! Check that this jmp_buf has no saved registers.
        mov     ax, _gtobuf             ! get flags
        test    ax, #4
        je      __longjerr

! Set up the value to return in ax.
        mov     ax, 6(bp)               ! value to return
        or      ax, ax
        jne     nonzero
        mov     ax, #1
nonzero:
        mov     bx, #_gtobuf
        mov     bp, 10(bx)
        mov     sp, 8(bx)
        jmp     @6(bx)

.extern _longjmp
_longjmp:
        push    bp
        mov     bp, sp

! Check that this is a jmp_buf with no saved regs and with signal context info.
        mov     bx, 4(bp)               ! pointer to jmp_buf
        mov     ax, 0(bx)               ! get the flags
        test    ax, #4                  ! check for no saved registers
        je      __longjerr
        test    ax, #2                  ! check for signal context
        je      __longjerr

! Compute the value to return
        mov     ax, 6(bp)               ! proposed value to return
        or      ax, ax
        jne     nonzero1
        mov     ax, #1
nonzero1:

! Call _sigjmp to restore the old signal context.
        push    ax
        push    4(bp)
        call    __sigjmp
        add     sp, #4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/_memmove.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       _memmove()                                      Author: Kees J. Bot
!                                                                 27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void *_memmove(void *s1, const void *s2, size_t n)
!       Copy a chunk of memory.  Handle overlap.
!
.sect .text
.define __memmove, __memcpy
__memmove:
        push    bp
        mov     bp, sp
        push    si
        push    di
        mov     di, 4(bp)       ! String s1
        mov     si, 6(bp)       ! String s2
        mov     cx, 8(bp)       ! Length
        mov     ax, di
        sub     ax, si
        cmp     ax, cx
        jb      downwards       ! if (s2 - s1) < n then copy downwards
__memcpy:
        cld                     ! Clear direction bit: upwards
        cmp     cx, #16
        jb      upbyte          ! Don't bother being smart with short arrays
        mov     ax, si
        or      ax, di
        testb   al, #1
        jnz     upbyte          ! Bit 0 set, use byte copy
upword: shr     cx, #1
    rep movs                    ! Copy words
        adc     cx, cx          ! One more byte?
upbyte:
    rep movsb                   ! Copy bytes
done:   mov     ax, 4(bp)       ! Absolutely noone cares about this value
        pop     di
        pop     si
        pop     bp
        ret

! Handle bad overlap by copying downwards, don't bother to do word copies.
downwards:
        std                     ! Set direction bit: downwards
        add     si, cx
        dec     si
        add     di, cx
        dec     di
    rep movsb                   ! Copy bytes
        cld
        jmp     done
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/_strncat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       _strncat()                                      Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *_strncat(char *s1, const char *s2, size_t dx)
!       Append string s2 to s1.
!
.sect .text
.define __strncat
__strncat:
        push    bp
        mov     bp, sp
        push    si
        push    di
        mov     di, 4(bp)       ! String s1
        mov     cx, #-1
        xorb    al, al          ! Null byte
        cld
  repne scasb                   ! Look for the zero byte in s1
        dec     di              ! Back one up (and clear 'Z' flag)
        push    di              ! Save end of s1
        mov     di, 6(bp)       ! di = string s2
        mov     cx, dx          ! Maximum count
  repne scasb                   ! Look for the end of s2
        jne     no0
        inc     cx              ! Exclude null byte
no0:    sub     dx, cx          ! Number of bytes in s2
        mov     cx, dx
        mov     si, 6(bp)       ! si = string s2
        pop     di              ! di = end of string s1
    rep movsb                   ! Copy bytes
        stosb                   ! Add a terminating null
        mov     ax, 4(bp)       ! Return s1
        pop     di
        pop     si
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/_strncmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strncmp()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! int strncmp(const char *s1, const char *s2, size_t cx)
!       Compare two strings.
!
.sect .text
.define __strncmp
__strncmp:
```

```
        push    bp
        mov     bp, sp
        push    si
        push    di
        xor     ax, ax          ! Prepare return value
        test    cx, cx          ! Max length is zero?
        je      equal
        mov     si, 4(bp)       ! si = string s1
        mov     di, 6(bp)       ! di = string s2
        cld
compare:
        cmpsb                   ! Compare two bytes
        jne     unequal
        cmpb    -1(si), #0      ! End of string?
        je      equal
        dec     cx              ! Length limit reached?
        jne     compare
        jmp     equal
unequal:
        ja      after
        sub     ax, #2          ! if (s1 < s2) ax -= 2;
after:  inc     ax              ! ax++, now it's -1 or 1
equal:  pop     di
        pop     si
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/_strncpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       _strncpy()                                      Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *_strncpy(char *s1, const char *s2, size_t cx)
!       Copy string s2 to s1.
!
.sect .text
.define __strncpy
__strncpy:
        mov     di, 6(bp)       ! di = string s2
        xorb    al, al          ! Look for a zero byte
        mov     dx, cx          ! Save maximum count
        cld
  repne scasb                   ! Look for end of s2
        sub     dx, cx          ! Number of bytes in s2 including null
        xchg    cx, dx
        mov     si, 6(bp)       ! si = string s2
        mov     di, 4(bp)       ! di = string s1
    rep movsb                   ! Copy bytes
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/_strnlen.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       _strnlen()                                      Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! size_t _strnlen(const char *s, size_t cx)
!       Return the length of a string.
!
.sect .text
.define __strnlen
__strnlen:
        push    bp
        mov     bp, sp
        push    di
        mov     di, 4(bp)       ! di = string
        xorb    al, al          ! Look for a zero byte
        mov     dx, cx          ! Save maximum count
        cmpb    cl, #1          ! 'Z' bit must be clear if cx = 0
        cld
  repne scasb                   ! Look for zero
        jne     no0
        inc     cx              ! Don't count zero byte
no0:    mov     ax, dx
        sub     ax, cx          ! Compute bytes scanned
        pop     di
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/bcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       bcmp()                                          Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! int bcmp(const void *s1, const void *s2, size_t n)
!       Compare two chunks of memory.
!       This is a BSD routine that escaped from the kernel.  Don't use.
!       (Alas it is not without some use, it reports the number of bytes
!       after the bytes that are equal.  So it can't be simply replaced.)
!
.sect .text
.define _bcmp
_bcmp:
        push    bp
        mov     bp, sp
        push    8(bp)
        push    6(bp)
        push    4(bp)
        call    _memcmp         ! Let memcmp do the work
        mov     sp, bp
```

```
        test    ax, ax
        jz      equal
        sub     dx, 4(bp)       ! Memcmp was nice enough to leave "si" in dx
        dec     dx              ! Number of bytes that are equal
        mov     ax, 8(bp)
        sub     ax, dx          ! Number of bytes that are unequal
equal:  pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/bcopy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       bcopy()                                         Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void bcopy(const void *s1, void *s2, size_t n)
!       Copy a chunk of memory.  Handle overlap.
!       This is a BSD routine that escaped from the kernel.  Don't use.
!
.sect .text
.define _bcopy
.extern __memmove
_bcopy:
        pop     cx
        pop     ax
        pop     dx              ! Pop return address and arguments
        push    ax
        push    dx              ! Arguments reversed
        push    cx
        jmp     __memmove       ! Call the proper routine
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/bzero.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       bzero()                                         Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void bzero(void *s, size_t n)
!       Set a chunk of memory to zero.
!       This is a BSD routine that escaped from the kernel.  Don't use.
!
.sect .text
.define _bzero
_bzero:
        push    bp
        mov     bp, sp
        push    6(bp)           ! Size
        xor     ax, ax
        push    ax              ! Zero
        push    4(bp)           ! String
        call    _memset         ! Call the proper routine
        mov     sp, bp
        pop     bp
```

```
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/string/index.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       index()                                         Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *index(const char *s, int c)
!       Look for a character in a string.  Has suffered from a hostile
!       takeover by strchr().
!
.sect .text
.define _index
_index:
        jmp     _strchr
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/string/memchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       memchr()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void *memchr(const void *s, int c, size_t n)
!       Look for a character in a chunk of memory.
!
.sect .text
.define _memchr
_memchr:
        push    bp
        mov     bp, sp
        push    di
        mov     di, 4(bp)       ! di = string
        movb    al, 6(bp)       ! The character to look for
        mov     cx, 8(bp)       ! Length
        cmpb    cl, #1          ! 'Z' bit must be clear if cx = 0
        cld
  repne scasb
        jne     failure
        lea     ax, -1(di)      ! Found
        pop     di
        pop     bp
        ret
failure:xor     ax, ax
        pop     di
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/memcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       memcmp()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! int memcmp(const void *s1, const void *s2, size_t n)
!       Compare two chunks of memory.
!
.sect .text
.define _memcmp
_memcmp:
        cld
        push    bp
        mov     bp, sp
        push    si
        push    di
        xor     ax, ax          ! Prepare return value
        mov     si, 4(bp)       ! String s1
        mov     di, 6(bp)       ! String s2
        mov     cx, 8(bp)       ! Length
        cmp     cx, #16
        jb      cbyte           ! Don't bother being smart with short arrays
        mov     dx, si
        or      dx, di
        andb    dl, #1
        jnz     cbyte           ! Bit 0 set, use byte compare
cword:  sar     cx, #1
        adcb    dl, dl          ! Save carry
   repe cmps                    ! Compare words
        mov     cx, #2          ! Recompare the last word
        sub     si, cx
        sub     di, cx
        addb    cl, dl          ! One more byte?
cbyte:  test    cx, cx          ! Set 'Z' flag if cx = 0
last:
   repe cmpsb                   ! Look for the first differing byte
        je      equal
        ja      after
        sub     ax, #2          ! if (s1 < s2) ax -= 2;
after:  inc     ax              ! ax++, now it's -1 or 1
equal:  mov     dx, si          ! For bcmp() to play with
        pop     di
        pop     si
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/memcpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       memcpy()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void *memcpy(void *s1, const void *s2, size_t n)
!       Copy a chunk of memory.
!       This routine need not handle overlap, so it does not handle overlap.
!       One could simply call __memmove, the cost of the overlap check is
!       negligible, but you are dealing with a programmer who believes that
!       if anything can go wrong, it should go wrong.
!
.sect .text
.define _memcpy
_memcpy:
        push    bp
        mov     bp, sp
        push    si
        push    di
        mov     di, 4(bp)       ! String s1
        mov     si, 6(bp)       ! String s2
        mov     cx, 8(bp)       ! Length
        ! No overlap check here
        jmp     __memcpy        ! Call the part of __memmove that copies up
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/memmove.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       memmove()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void *memmove(void *s1, const void *s2, size_t n)
!       Copy a chunk of memory.  Handle overlap.
!
.sect .text
.define _memmove
_memmove:
        jmp     __memmove       ! Call common code
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/string/memset.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       memset()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! void *memset(void *s, int c, size_t n)
!       Set a chunk of memory to the same byte value.
!
.sect .text
.define _memset
_memset:
        push    bp
        mov     bp, sp
        push    di
        mov     di, 4(bp)       ! The string
        movb    al, 6(bp)       ! The fill byte
        mov     cx, 8(bp)       ! Length
        cld
        cmp     cx, #16
        jb      sbyte           ! Don't bother being smart with short arrays
        test    di, #1
        jnz     sbyte           ! Bit 0 set, use byte store
sword:  movb    ah, al          ! One byte to two bytes
        sar     cx, #1
    rep stos                    ! Store words
        adc     cx, cx          ! One more byte?
sbyte:
    rep stosb                   ! Store bytes
done:   mov     ax, 4(bp)       ! Return some value you have no need for
        pop     di
        pop     bp
        ret
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i86/string/rindex.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       rindex()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *rindex(const char *s, int c)
!       Look for the last occurrence a character in a string.  Has suffered
!       from a hostile takeover by strrchr().
!
.sect .text
.define _rindex
_rindex:
        jmp     _strrchr
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strcat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strcat()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *strcat(char *s1, const char *s2)
!       Append string s2 to s1.
!
.sect .text
.define _strcat
_strcat:
        mov     dx, #-1         ! Unlimited length
        jmp     __strncat       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strchr()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *strchr(const char *s, int c)
!       Look for a character in a string.
!
.sect .text
.define _strchr
_strchr:
        push    bp
        mov     bp, sp
        push    di
        cld
        mov     di, 4(bp)       ! di = string
        mov     dx, #16         ! Look at small chunks of the string
next:   shl     dx, #1          ! Chunks become bigger each time
        mov     cx, dx
        xorb    al, al          ! Look for the zero at the end
  repne scasb
        pushf                   ! Remember the flags
        sub     cx, dx
        neg     cx              ! Some or all of the chunk
        sub     di, cx          ! Step back
        movb    al, 6(bp)       ! The character to look for
  repne scasb
        je      found
        popf                    ! Did we find the end of string earlier?
        jne     next            ! No, try again
        xor     ax, ax          ! Return NULL
        pop     di
        pop     bp
        ret
found:  pop     ax              ! Get rid of those flags
        lea     ax, -1(di)      ! Address of byte found
```

```
        pop     di
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       strcmp()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! int strcmp(const char *s1, const char *s2)
!       Compare two strings.
!
.sect .text
.define _strcmp
_strcmp:
        mov     cx, #-1         ! Unlimited length
        jmp     __strncmp       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strcpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
!       strcpy()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *strcpy(char *s1, const char *s2)
!       Copy string s2 to s1.
!
.sect .text
.define _strcpy
_strcpy:
        push    bp
        mov     bp, sp
        push    si
        push    di
        mov     cx, #-1         ! Unlimited length
        call    __strncpy       ! Common code
        mov     ax, 4(bp)       ! Return s1
        pop     di
        pop     si
        pop     bp
        ret
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strlen.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strlen()                                        Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! size_t strlen(const char *s)
!       Return the length of a string.
!
.sect .text
.define _strlen
_strlen:
        mov     cx, #-1         ! Unlimited length
        jmp     __strnlen       ! Common code
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strncat.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strncat()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! size_t strncat(char *s1, const char *s2, size_t n)
!       Append string s2 to s1.
!
.sect .text
.define _strncat
_strncat:
        mov     bx, sp
        mov     dx, 6(bx)       ! Maximum length
        jmp     __strncat       ! Common code
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strncmp.s
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strncmp()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! int strncmp(const char *s1, const char *s2, size_t n)
!       Compare two strings.
!
.sect .text
.define _strncmp
_strncmp:
        mov     bx, sp
        mov     cx, 6(bx)       ! Maximum length
        jmp     __strncmp       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strncpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strncpy()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *strncpy(char *s1, const char *s2, size_t n)
!       Copy string s2 to s1.
!
.sect .text
.define _strncpy
_strncpy:
        push    bp
        mov     bp, sp
        push    si
        push    di
        mov     cx, 8(bp)       ! Maximum length
        call    __strncpy       ! Common code
        mov     cx, dx          ! Number of bytes not copied
    rep stosb                   ! strncpy always copies n bytes by null padding
        mov     ax, 4(bp)       ! Return s1
        pop     di
        pop     si
        pop     bp
        ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strnlen.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strnlen()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! size_t strnlen(const char *s, size_t n)
!       Return the length of a string.
!
.sect .text
.define _strnlen
_strnlen:
        mov     bx, sp
        mov     cx, 4(bx)       ! Maximum length
        jmp     __strnlen       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i86/string/strrchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!       strrchr()                                       Author: Kees J. Bot
!                                                               27 Jan 1994
.sect .text; .sect .rom; .sect .data; .sect .bss

! char *strrchr(const char *s, int c)
!       Look for the last occurrence a character in a string.
!
.sect .text
.define _strrchr
_strrchr:
        push    bp
        mov     bp, sp
        push    di
        mov     di, 4(bp)       ! di = string
        mov     cx, #-1
        xorb    al, al
        cld
  repne scasb                   ! Look for the end of the string
        not     cx              ! -1 - cx = Length of the string + null
        dec     di              ! Put di back on the zero byte
        movb    al, 6(bp)       ! The character to look for
        std                     ! Downwards search
  repne scasb
        cld                     ! Direction bit back to default
        jne     failure
        lea     ax, 1(di)       ! Found it
        pop     di
        pop     bp
        ret
failure:xor     ax, ax          ! Not there
        pop     di
        pop     bp
        ret
```