++
src/lib/i386/em/byte_order.h
++

```
00000    #define CHAR_UNSIGNED   0
00001    #define MSB_AT_LOW_ADDRESS      0
00002    #define MSW_AT_LOW_ADDRESS      0
00003    #define FL_MSB_AT_LOW_ADDRESS   0
00004    #define FL_MSW_AT_LOW_ADDRESS   0
00005    #define FL_MSL_AT_LOW_ADDRESS   0
```

++
src/lib/i386/em/em_adf4.s
++

```
00100    .sect .text; .sect .rom; .sect .data; .sect .bss
00101    .define .adf4
00102
00103            .sect .text
00104    .adf4:
00105            mov     bx,sp
00106            flds    4(bx)
00107            fadds   8(bx)
00108            fstps   8(bx)
00109            wait
00110            ret
```

++
src/lib/i386/em/em_adf8.s
++

```
00200    .sect .text; .sect .rom; .sect .data; .sect .bss
00201    .define .adf8
00202
00203            .sect .text
00204    .adf8:
00205            mov     bx,sp
00206            fldd    4(bx)
00207            faddd   12(bx)
00208            fstpd   12(bx)
00209            wait
00210            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_adi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00300    .sect .text; .sect .rom; .sect .data; .sect .bss
00301    .define .adi
00302
00303            ! #bytes in ecx , top of stack in eax
00304            .sect .text
00305    .adi:
00306            pop     ebx              ! return address
00307            cmp     ecx,4
00308            jne     9f
00309            pop     ecx
00310            add     eax,ecx
00311            jmp     ebx
00312    9:
00313    .extern EODDZ
00314    .extern .trp
00315            mov     eax,EODDZ
00316            push    ebx
00317            jmp     .trp



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_and.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00400    .sect .text; .sect .rom; .sect .data; .sect .bss
00401    .define .and
00402
00403            ! #bytes in ecx
00404            ! save edi; it might be a register variable
00405
00406            .sect .text
00407    .and:
00408            pop     ebx             ! return address
00409            mov     edx,edi
00410            mov     edi,esp
00411            add     edi,ecx
00412            sar     ecx,2
00413    1:
00414            pop     eax
00415            and     eax,(edi)
00416            stos
00417            loop    1b
00418            mov     edi,edx
00419            jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_blm.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00500    .sect .text; .sect .rom; .sect .data; .sect .bss
00501    .sect .text
00502    .define .blm
00503
00504            ! ecx: count in words
00505    .blm:
00506            mov     ebx,esp
00507            mov     eax,esi
00508            mov     edx,edi
00509            mov     edi,4(ebx)
00510            mov     esi,8(ebx)
00511            rep     movs
00512            mov     esi,eax
00513            mov     edi,edx
00514            ret     8
00515
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cff4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00600    .sect .text; .sect .rom; .sect .data; .sect .bss
00601    .define .cff4
00602
00603            .sect .text
00604    .cff4:
00605            mov     bx,sp
00606            fldd    4(bx)
00607            fstcw   4(bx)
00608            wait
00609            mov     dx,4(bx)
00610            and     4(bx),0xf3ff    ! set to rounding mode
00611            wait
00612            fldcw   4(bx)
00613            fstps   8(bx)
00614            mov     4(bx),dx
00615            wait
00616            fldcw   4(bx)
00617            wait
00618            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cff8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00700    .sect .text; .sect .rom; .sect .data; .sect .bss
00701    .define .cff8
00702
00703            .sect .text
00704    .cff8:
00705            mov     bx,sp
00706            flds    4(bx)
00707            fstpd   4(bx)
00708            wait
00709            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cfi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00800    .sect .text; .sect .rom; .sect .data; .sect .bss
00801    .define .cfi
00802
00803            .sect .text
00804    .cfi:
00805            mov     bx,sp
00806            fstcw   4(bx)
00807            wait
00808            mov     dx,4(bx)
00809            or      4(bx),0xc00     ! truncating mode
00810            wait
00811            fldcw   4(bx)
00812            cmp     8(bx),4
00813            jne     2f
00814                                    ! loc 4 loc ? cfi
00815            flds    12(bx)
00816            fistpl  12(bx)
00817    1:
00818            mov     4(bx),dx
00819            wait
00820            fldcw   4(bx)
00821            ret
00822    2:
00823                                    ! loc 8 loc ? cfi
00824            fldd    12(bx)
00825            fistpl  16(bx)
00826            jmp     1b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cfu.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00900    .sect .text; .sect .rom; .sect .data; .sect .bss
00901    .define .cfu
00902
00903            .sect .text
00904    .cfu:
00905            mov     bx,sp
00906            fstcw   4(bx)
00907            wait
00908            mov     dx,4(bx)
00909            or      4(bx),0xc00     ! truncating mode
00910            wait
00911            fldcw   4(bx)
00912            cmp     8(bx),4
00913            jne     2f
00914                                    ! loc 4 loc ? cfu
00915            flds    12(bx)
00916            fabs                    ! ???
00917            fiaddl  (bigmin)
00918            fistpl  12(bx)
00919            wait
00920            mov     ax,12(bx)
00921            sub     ax,(bigmin)
00922            mov     12(bx),ax
00923    1:
00924            mov     4(bx),dx
00925            wait
00926            fldcw   4(bx)
00927            ret
00928    2:
00929                                    ! loc 8 loc ? cfu
00930            fldd    12(bx)
00931            fabs                    ! ???
00932            fiaddl  (bigmin)
00933            fistpl  16(bx)
00934            mov     ax,16(bx)
00935            sub     ax,(bigmin)
00936            mov     16(bx),ax
00937            jmp     1b


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cif4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01000    .sect .text; .sect .rom; .sect .data; .sect .bss
01001    .define .cif4
01002
01003            .sect .text
01004    .cif4:
01005            mov     bx,sp
01006            fildl   8(bx)
01007            fstps   8(bx)
01008            wait
01009            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cif8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01100    .sect .text; .sect .rom; .sect .data; .sect .bss
01101    .define .cif8
01102
01103            .sect .text
01104    .cif8:
01105            mov     bx,sp
01106            fildl   8(bx)
01107            fstpd   4(bx)
01108            wait
01109            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cii.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01200    .sect .text; .sect .rom; .sect .data; .sect .bss
01201    .define .cii
01202
01203    .sect .text
01204    .cii:
01205            pop     ebx              ! return address
01206                                     ! pop     ecx, dest. size
01207                                     ! pop     edx, src. size
01208                                     ! eax is source
01209            cmp     edx,1
01210            jne     2f
01211            movsxb  eax,al
01212            mov     edx,4
01213            jmp     1f
01214    2:
01215            cmp     edx,2
01216            jne     1f
01217            cwde                    ! convert from 2 to 4 bytes
01218            mov     edx,4
01219    1:
01220            cmp     edx,ecx
01221            jne     9f
01222            cmp     edx,4
01223            jne     9f
01224            jmp     ebx
01225    9:
01226    .extern EILLINS
01227    .extern .fat
01228            mov     eax,EILLINS
01229            push    eax
01230            jmp     .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cmf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01300    .sect .text; .sect .rom; .sect .data; .sect .bss
01301    .define .cmf4
01302
01303            .sect .text
01304    .cmf4:
01305            mov     bx,sp
01306            xor     cx,cx
01307            flds    8(bx)
01308            flds    4(bx)
01309            fcompp                  ! compare and pop operands
01310            fstsw   ax
01311            wait
01312            sahf
01313            je      1f
01314            jb      2f
01315            dec     cx
01316            jmp     1f
01317    2:
01318            inc     cx
01319    1:
01320            mov     ax,cx
01321            ret



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cmf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01400    .sect .text; .sect .rom; .sect .data; .sect .bss
01401    .define .cmf8
01402
01403            .sect .text
01404    .cmf8:
01405            mov     bx,sp
01406            xor     cx,cx
01407            fldd    12(bx)
01408            fldd    4(bx)
01409            fcompp                  ! compare and pop operands
01410            fstsw   ax
01411            wait
01412            sahf
01413            je      1f
01414            jb      2f
01415            dec     cx
01416            jmp     1f
01417    2:
01418            inc     cx
01419    1:
01420            mov     ax,cx
01421            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cms.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01500    .sect .text; .sect .rom; .sect .data; .sect .bss
01501    .define .cms
01502
01503            ! #bytes in ecx
01504            .sect .text
01505    .cms:
01506            pop     ebx             ! return address
01507            mov     edx,esp
01508            push    esi
01509            push    edi
01510            mov     esi,edx
01511            add     edx,ecx
01512            mov     edi,edx
01513            add     edx,ecx
01514            sar     ecx,2
01515            repe cmps
01516            je      1f
01517            inc     ecx
01518    1:
01519            pop     edi
01520            pop     esi
01521            mov     esp,edx
01522            jmp     ebx


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_com.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01600    .sect .text; .sect .rom; .sect .data; .sect .bss
01601    .define .com
01602
01603            ! #bytes in ecx
01604            .sect .text
01605    .com:
01606            mov     ebx,esp
01607            add     ebx,4
01608            sar     ecx,2
01609    1:
01610            not     (ebx)
01611            add     ebx,4
01612            loop    1b
01613            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_csa4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01700    .sect .text; .sect .rom; .sect .data; .sect .bss
01701    .define .csa4
01702
01703    .sect .text
01704    .csa4:
01705                                    ! ebx, descriptor address
01706                                    ! eax, index
01707            mov     edx,(ebx)         ! default
01708            sub     eax,4(ebx)
01709            cmp     eax,8(ebx)
01710            ja      1f
01711            sal     eax,2
01712            add     ebx,eax
01713            mov     ebx,12(ebx)
01714            test    ebx,ebx
01715            jnz     2f
01716    1:
01717            mov     ebx,edx
01718            test    ebx,ebx
01719            jnz     2f
01720    .extern ECASE
01721    .extern .fat
01722            mov     eax,ECASE
01723            push    eax
01724            jmp     .fat
01725    2:
01726            jmp     ebx



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_csb4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01800    .sect .text; .sect .rom; .sect .data; .sect .bss
01801    .define .csb4
01802
01803    .sect .text
01804    .csb4:
01805                                    !ebx, descriptor address
01806                                    !eax,  index
01807            mov     edx,(ebx)
01808            mov     ecx,4(ebx)
01809    1:
01810            add     ebx,8
01811            dec     ecx
01812            jl      4f
01813            cmp     eax,(ebx)
01814            jne     1b
01815            mov     ebx,4(ebx)
01816    2:
01817            test    ebx,ebx
01818            jnz     3f
01819    .extern ECASE
```

```
01820    .extern .fat
01821            mov     eax,ECASE
01822            push    eax
01823            jmp     .fat
01824    3:
01825            jmp     ebx
01826    4:
01827            mov     ebx,edx
01828            jmp     2b
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cuf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

01900    .sect .text; .sect .rom; .sect .data; .sect .bss
01901    .define .cuf4
01902
01903            .sect .text
01904    .cuf4:
01905            mov     bx,sp
01906            fildl   8(bx)
01907            cmp     8(bx),0
01908            jge     1f
01909            fisubl  (bigmin)
01910            fisubl  (bigmin)
01911    1:
01912            fstps   8(bx)
01913            wait
01914            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cuf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02000    .sect .text; .sect .rom; .sect .data; .sect .bss
02001    .define .cuf8
02002
02003            .sect .text
02004    .cuf8:
02005            mov     bx,sp
02006            fildl   8(bx)
02007            cmp     8(bx),0
02008            jge     1f
02009            fisubl  (bigmin)
02010            fisubl  (bigmin)
02011    1:
02012            fstpd   4(bx)
02013            wait
02014            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_cuu.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02100    .sect .text; .sect .rom; .sect .data; .sect .bss
02101    .define .ciu
02102    .define .cui
02103    .define .cuu
02104
02105    .sect .text
02106    .ciu:
02107    .cui:
02108    .cuu:
02109            pop     ebx              ! return address
02110                                    ! pop     ecx, dest. size
02111                                    ! pop     edx, source size
02112                                    ! eax is source
02113            cmp     edx,ecx
02114            jne     8f
02115            jmp     ebx
02116    8:
02117    .extern EILLINS
02118    .extern .fat
02119            mov     eax,EILLINS
02120            push    eax
02121            jmp     .fat
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_dup.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02200    .sect .text; .sect .rom; .sect .data; .sect .bss
02201    .define .dup
02202
02203            ! #bytes in ecx
02204            .sect .text
02205    .dup:
02206            pop     ebx             ! return address
02207            mov     eax,esi
02208            mov     edx,edi
02209            mov     esi,esp
02210            sub     esp,ecx
02211            mov     edi,esp
02212            sar     ecx,2
02213            rep movs
02214            mov     esi,eax
02215            mov     edi,edx
02216            jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_dvf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02300    .sect .text; .sect .rom; .sect .data; .sect .bss
02301    .define .dvf4
02302
02303            .sect .text
02304    .dvf4:
02305            mov     bx,sp
02306            flds    8(bx)
02307            fdivs   4(bx)
02308            fstps   8(bx)
02309            wait
02310            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_dvf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02400    .sect .text; .sect .rom; .sect .data; .sect .bss
02401    .define .dvf8
02402
02403            .sect .text
02404    .dvf8:
02405            mov     bx,sp
02406            fldd    12(bx)
02407            fdivd   4(bx)
02408            fstpd   12(bx)
02409            wait
02410            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_dvi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02500    .sect .text; .sect .rom; .sect .data; .sect .bss
02501    .define .dvi
02502
02503            ! #bytes in eax
02504            .sect .text
02505    .dvi:
02506            pop     ebx             ! return address
02507            cmp     eax,4
02508            jne     1f
02509            pop     eax
02510            cwd
02511            pop     ecx
02512            idiv    ecx
02513            push    eax
02514            jmp     ebx
```

```
02515    1:
02516    .extern EODDZ
02517    .extern .trp
02518            mov     eax,EODDZ
02519            push    ebx
02520            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_dvu.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02600    .sect .text; .sect .rom; .sect .data; .sect .bss
02601    .define .dvu
02602
02603            ! #bytes in eax
02604            .sect .text
02605    .dvu:
02606            pop     ebx             ! return address
02607            cmp     eax,4
02608            jne     1f
02609            pop     eax
02610            xor     edx,edx
02611            pop     ecx
02612            div     ecx
02613            push    eax
02614            jmp     ebx
02615    1:
02616    .extern EODDZ
02617    .extern .trp
02618            mov     eax,EODDZ
02619            push    ebx
02620            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_error.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02700    .sect .text; .sect .rom; .sect .data; .sect .bss
02701    .define .error
02702    .define .Xtrp
02703
02704            ! eax is trap number
02705            ! all registers must be saved
02706            ! because return is possible
02707            ! May only be called with error no's <16
02708    .sect .text
02709    .error:
02710            mov  ecx,eax
02711            mov  ebx,1
02712            sal  ebx,cl
02713    .extern .ignmask
02714    .extern .trp
```

```
02715           test ebx,(.ignmask)
02716           jne  2f
02717           call    .trp
02718   2:
02719           ret
02720
02721   .Xtrp:
02722           pusha
02723           cmp     eax,16
02724           jge     1f
02725           call    .error
02726           popa
02727           ret
02728   1:
02729           call    .trp
02730           popa
02731           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_exg.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02800   .sect .text; .sect .rom; .sect .data; .sect .bss
02801   .define .exg
02802
02803           ! #bytes in ecx
02804   .sect .text
02805   .exg:
02806           push    edi
02807           mov     edi,esp
02808           add     edi,8
02809           mov     ebx,edi
02810           add     ebx,ecx
02811           sar     ecx,2
02812   1:
02813           mov     eax,(ebx)
02814           xchg    eax,(edi)
02815           mov     (ebx),eax
02816           add     edi,4
02817           add     ebx,4
02818           loop    1b
02819   2:
02820           pop     edi
02821           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

02900    .sect .text; .sect .rom; .sect .data; .sect .bss
02901    .sect .text
02902    .define .fat
02903
02904    .fat:
02905    .extern .trp
02906    .extern .stop
02907            call    .trp
02908            call    .stop
02909            ! no return
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fef4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03000    .sect .text; .sect .rom; .sect .data; .sect .bss
03001    .define .fef4
03002
03003            .sect .text
03004    .fef4:
03005                                    ! this could be simpler, if only the
03006                                    ! fxtract instruction was emulated properly
03007            mov     bx,sp
03008            mov     ax,8(bx)
03009            and     ax,0x7f800000
03010            je      1f              ! zero exponent
03011            shr     ax,23
03012            sub     ax,126
03013            mov     cx,ax           ! exponent in cx
03014            mov     ax,8(bx)
03015            and     ax,0x807fffff
03016            or      ax,0x3f000000   ! load -1 exponent
03017            mov     bx,4(bx)
03018            mov     4(bx),ax
03019            mov     (bx),cx
03020            ret
03021    1:                              ! we get here on zero exp
03022            mov     ax,8(bx)
03023            and     ax,0x007fffff
03024            jne     1f              ! zero result
03025            mov     bx,4(bx)
03026            mov     (bx),ax
03027            mov     4(bx),ax
03028            ret
03029    1:                              ! otherwise unnormalized number
03030            mov     cx,8(bx)
03031            and     cx,0x807fffff
03032            mov     dx,cx
03033            and     cx,0x80000000
03034            mov     ax,-125
```

```
03035   2:
03036           test    dx,0x800000
03037           jne     1f
03038           dec     ax
03039           shl     dx,1
03040           or      dx,cx
03041           jmp     2b
03042   1:
03043           mov     bx,4(bx)
03044           mov     (bx),ax
03045           and     dx,0x807fffff
03046           or      dx,0x3f000000   ! load -1 exponent
03047           mov     4(bx),dx
03048           ret


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fef8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03100   .sect .text; .sect .rom; .sect .data; .sect .bss
03101   .define .fef8
03102
03103           .sect .text
03104   .fef8:
03105                                   ! this could be simpler, if only the
03106                                   ! fxtract instruction was emulated properly
03107           mov     bx,sp
03108           mov     ax,12(bx)
03109           and     ax,0x7ff00000
03110           je      1f              ! zero exponent
03111           shr     ax,20
03112           sub     ax,1022
03113           mov     cx,ax           ! exponent in cx
03114           mov     ax,12(bx)
03115           and     ax,0x800fffff
03116           or      ax,0x3fe00000   ! load -1 exponent
03117           mov     dx,8(bx)
03118           mov     bx,4(bx)
03119           mov     4(bx),dx
03120           mov     8(bx),ax
03121           mov     (bx),cx
03122           ret
03123   1:                              ! we get here on zero exp
03124           mov     ax,12(bx)
03125           and     ax,0xfffff
03126           or      ax,8(bx)
03127           jne     1f              ! zero result
03128           mov     bx,4(bx)
03129           mov     (bx),ax
03130           mov     4(bx),ax
03131           mov     8(bx),ax
03132           ret
03133   1:                              ! otherwise unnormalized number
03134           mov     cx,12(bx)
03135           and     cx,0x800fffff
03136           mov     dx,cx
03137           and     cx,0x80000000
03138           mov     ax,-1021
03139   2:
```

```
03140           test    dx,0x100000
03141           jne     1f
03142           dec     ax
03143           shl     8(bx),1
03144           rcl     dx,1
03145           or      dx,cx
03146           jmp     2b
03147   1:
03148           and     dx,0x800fffff
03149           or      dx,0x3fe00000   ! load -1 exponent
03150           mov     cx,8(bx)
03151           mov     bx,4(bx)
03152           mov     (bx),ax
03153           mov     8(bx),dx
03154           mov     4(bx),cx
03155           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fif4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03200   .sect .text; .sect .rom; .sect .data; .sect .bss
03201   .define .fif4
03202
03203           .sect .text
03204   .fif4:
03205           mov     bx,sp
03206           flds    8(bx)
03207           fmuls   12(bx)          ! multiply
03208           fld     st              ! copy result
03209           ftst                    ! test sign; handle negative separately
03210           fstsw   ax
03211           wait
03212           sahf                    ! result of test in condition codes
03213           jb      1f
03214           frndint                 ! this one rounds (?)
03215           fcom    st(1)           ! compare with original; if <=, then OK
03216           fstsw   ax
03217           wait
03218           sahf
03219           jbe     2f
03220           fisubs  (one)           ! else subtract 1
03221           jmp     2f
03222   1:                              ! here, negative case
03223           frndint                 ! this one rounds (?)
03224           fcom    st(1)           ! compare with original; if >=, then OK
03225           fstsw   ax
03226           wait
03227           sahf
03228           jae     2f
03229           fiadds  (one)           ! else add 1
03230   2:
03231           fsub    st(1),st        ! subtract integer part
03232           mov     bx,4(bx)
03233           fstps   (bx)
03234           fstps   4(bx)
```

```
03235           wait
03236           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fif8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
03300   .sect .text; .sect .rom; .sect .data; .sect .bss
03301   .define .fif8
03302
03303           .sect .text
03304   .fif8:
03305           mov     bx,sp
03306           fldd    8(bx)
03307           fmuld   16(bx)          ! multiply
03308           fld     st              ! and copy result
03309           ftst                    ! test sign; handle negative separately
03310           fstsw   ax
03311           wait
03312           sahf                    ! result of test in condition codes
03313           jb      1f
03314           frndint                 ! this one rounds (?)
03315           fcom    st(1)           ! compare with original; if <=, then OK
03316           fstsw   ax
03317           wait
03318           sahf
03319           jbe     2f
03320           fisubs  (one)           ! else subtract 1
03321           jmp     2f
03322   1:                              ! here, negative case
03323           frndint                 ! this one rounds (?)
03324           fcom    st(1)           ! compare with original; if >=, then OK
03325           fstsw   ax
03326           wait
03327           sahf
03328           jae     2f
03329           fiadds  (one)           ! else add 1
03330   2:
03331           fsub    st(1),st        ! subtract integer part
03332           mov     bx,4(bx)
03333           fstpd   (bx)
03334           fstpd   8(bx)
03335           wait
03336           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_fp8087.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03400    .sect .text; .sect .rom; .sect .data; .sect .bss
03401    .define one, bigmin
03402
03403            .sect .rom
03404    one:
03405            .data2  1
03406    two:
03407            .data2  2
03408    bigmin:
03409            .data4  -2147483648
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_gto.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03500    .sect .text; .sect .rom; .sect .data; .sect .bss
03501    .sect .text
03502    .define .gto
03503
03504    .gto:
03505            mov     ebp,8(ebx)
03506            mov     esp,4(ebx)
03507            jmp     (ebx)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_iaar.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03600    .sect .text; .sect .rom; .sect .data; .sect .bss
03601    .sect .text
03602    .define .iaar
03603
03604    .iaar:
03605            pop     ecx
03606            pop     edx
03607            cmp     edx,4
03608    .extern .unknown
03609            jne     .unknown
03610            pop     ebx     ! descriptor address
03611            pop     eax     ! index
03612            sub     eax,(ebx)
03613            mul     8(ebx)
03614            pop     ebx     ! array base
03615            add     ebx,eax
03616            push    ecx
03617            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ilar.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03700    .sect .text; .sect .rom; .sect .data; .sect .bss
03701    .sect .text
03702    .define .ilar
03703
03704    .ilar:
03705            pop     ecx
03706            pop     edx
03707    .extern .unknown
03708            cmp     edx,4
03709            jne     .unknown
03710            pop     ebx      ! descriptor address
03711            pop     eax      ! index
03712            push    ecx
03713    .extern .lar4
03714            jmp    .lar4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_inn.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03800    .sect .text; .sect .rom; .sect .data; .sect .bss
03801    .sect .text
03802    .define .inn
03803
03804            ! #bytes in ecx
03805            ! bit # in eax
03806    .inn:
03807            xor     edx,edx
03808            mov     ebx,8
03809            div     ebx
03810            mov     ebx,esp
03811            add     ebx,4
03812            add     ebx,eax
03813            cmp     eax,ecx
03814            jae     1f
03815            movb    al,(ebx)
03816            mov     ebx,edx
03817            testb   al,bits(ebx)
03818            jz      1f
03819            mov     eax,1
03820            jmp     2f
03821    1:
03822            xor     eax,eax
03823    2:
03824            pop     ebx
03825            add     esp,ecx
03826            ! eax is result
03827            jmp     ebx
03828
03829            .sect .rom
```

```
03830    bits:
03831            .data1 1,2,4,8,16,32,64,128
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ior.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

03900    .sect .text; .sect .rom; .sect .data; .sect .bss
03901    .sect .text
03902    .define .ior
03903
03904            ! #bytes in ecx
03905    .ior:
03906            pop     ebx             ! return address
03907            mov     edx,edi
03908            mov     edi,esp
03909            add     edi,ecx
03910            sar     ecx,2
03911    1:
03912            pop     eax
03913            or      eax,(edi)
03914            stos
03915            loop    1b
03916            mov     edi,edx
03917            jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_isar.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04000    .sect .text; .sect .rom; .sect .data; .sect .bss
04001    .sect .text
04002    .define .isar
04003
04004    .isar:
04005            pop     ecx
04006            pop     eax
04007            cmp     eax,4
04008    .extern .unknown
04009            jne     .unknown
04010            pop     ebx      ! descriptor address
04011            pop     eax      ! index
04012            push    ecx
04013    .extern .sar4
04014            jmp    .sar4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_lar4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04100    .sect .text; .sect .rom; .sect .data; .sect .bss
04101    .sect .text
04102    .define .lar4
04103
04104    .lar4:
04105                                    ! ebx, descriptor address
04106                                    ! eax, index
04107            sub     eax,(ebx)
04108            mov     ecx,8(ebx)
04109            imul    ecx
04110            pop     ebx
04111            pop     edx             ! base address
04112            add     edx,eax
04113            sar     ecx,1
04114            jnb     1f
04115            xor     eax,eax
04116            movb    al,(edx)
04117            push    eax
04118            jmp     ebx
04119    1:
04120            sar     ecx,1
04121            jnb     1f
04122            xor     eax,eax
04123            o16 mov ax,(edx)
04124            push    eax
04125            jmp     ebx
04126    1:
04127            xchg    edx,esi         ! saved esi
04128            mov     eax,ecx
04129            sal     eax,2
04130            sub     esp,eax
04131            mov     eax,edi         ! save edi
04132            mov     edi,esp
04133            rep movs
04134            mov     edi,eax
04135            mov     esi,edx
04136            jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_loi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04200    .sect .text; .sect .rom; .sect .data; .sect .bss
04201    .sect .text
04202    .define .loi
04203    .define .los
04204
04205            ! #bytes in ecx
04206            ! address in ebx
04207            ! save esi/edi. they might be register variables
04208    .los:
04209            pop     edx
```

```
04210           mov     eax,ecx
04211           sar     ecx,1
04212           jnb     1f
04213           movsxb  eax,(ebx)
04214           push    eax
04215           jmp     edx
04216   1:
04217           sar     ecx,1
04218           jnb     1f
04219           movsx   eax,(ebx)
04220           push    eax
04221           jmp     edx
04222   1:
04223           push    edx
04224           mov     edx,esi
04225           mov     esi,ebx
04226           pop     ebx
04227           sub     esp,eax
04228           jmp     1f
04229
04230   .loi:
04231           ! only called with size >= 4
04232           mov     edx,esi
04233           mov     esi,ebx
04234           pop     ebx
04235           sub     esp,ecx
04236           sar     ecx,2
04237   1:
04238           mov     eax,edi
04239           mov     edi,esp
04240           rep movs
04241           mov     esi,edx
04242           mov     edi,eax
04243           jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_mlf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04300   .sect .text; .sect .rom; .sect .data; .sect .bss
04301   .define .mlf4
04302
04303           .sect .text
04304   .mlf4:
04305           mov     bx,sp
04306           flds    4(bx)
04307           fmuls   8(bx)
04308           fstps   8(bx)
04309           wait
04310           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_mlf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04400    .sect .text; .sect .rom; .sect .data; .sect .bss
04401    .define .mlf8
04402
04403            .sect .text
04404    .mlf8:
04405            mov     bx,sp
04406            fldd    4(bx)
04407            fmuld   12(bx)
04408            fstpd   12(bx)
04409            wait
04410            ret




++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_mli.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04500    .sect .text; .sect .rom; .sect .data; .sect .bss
04501    .sect .text
04502    .define .mli
04503
04504            ! #bytes in eax
04505    .mli:
04506            pop     ebx             ! return address
04507            cmp     eax,4
04508            jne     1f
04509            pop     eax
04510            pop     ecx
04511            mul     ecx
04512            push    eax
04513            jmp     ebx
04514    1:
04515    .extern EODDZ
04516    .extern .trp
04517            mov     eax,EODDZ
04518            push    ebx
04519            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_mon.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04600    .sect .text; .sect .rom; .sect .data; .sect .bss
04601    .sect .text
04602    .define .mon
04603
04604    .mon:
04605    .extern .stop
04606            call    .stop



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ngf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04700    .sect .text; .sect .rom; .sect .data; .sect .bss
04701    .define .ngf4
04702
04703            .sect .text
04704    .ngf4:
04705            mov     bx,sp
04706            flds    4(bx)
04707            fchs
04708            fstps   4(bx)
04709            wait
04710            ret



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ngf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04800    .sect .text; .sect .rom; .sect .data; .sect .bss
04801    .define .ngf8
04802
04803            .sect .text
04804    .ngf8:
04805            mov     bx,sp
04806            fldd    4(bx)
04807            fchs
04808            fstpd   4(bx)
04809            wait
04810            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ngi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

04900    .sect .text; .sect .rom; .sect .data; .sect .bss
04901    .sect .text
04902    .define .ngi
04903
04904            ! #bytes in eax
04905    .ngi:
04906            pop     ebx             ! return address
04907            cmp     eax,4
04908            jne     1f
04909            pop     ecx
04910            neg     ecx
04911            push    ecx
04912            jmp     ebx
04913    1:
04914    .extern EODDZ
04915    .extern .trp
04916            mov     eax,EODDZ
04917            push    ebx
04918            jmp     .trp


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_nop.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05000    .sect .text; .sect .rom; .sect .data; .sect .bss
05001    .sect .text
05002    .define .nop
05003    .extern printd, printc, hol0
05004
05005    .nop:
05006            mov     eax,(hol0)
05007            call    printd
05008            movb    al,'\n'
05009            jmp     printc




++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_print.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05100    .sect .text; .sect .rom; .sect .data; .sect .bss
05101    .sect .text
05102    .define printc,printd,prints
05103
05104            ! argument in eax
05105            ! uses ebx
05106    prints:
05107            xchg    eax,ebx
05108    1:
05109            movb    al,(ebx)
```

```
05110           inc     ebx
05111           testb   al,al
05112           jz      2f
05113           call    printc
05114           jmp     1b
05115   2:
05116           ret
05117
05118           ! argument in eax
05119           ! uses ecx and edx
05120   printd:
05121           xor     edx,edx
05122           mov     ecx,10
05123           div     ecx
05124           test    eax,eax
05125           jz      1f
05126           push    edx
05127           call    printd
05128           pop     edx
05129   1:
05130           xchg    eax,edx
05131           addb    al,'0'
05132
05133           ! argument in eax
05134   printc:
05135           push    eax
05136           mov     ebx,esp
05137           mov     eax,1
05138           push    eax
05139           push    ebx
05140           push    eax
05141           call    __write
05142           pop     ebx
05143           pop     ebx
05144           pop     ebx
05145           pop     ebx
05146           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_rck.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05200   .sect .text; .sect .rom; .sect .data; .sect .bss
05201   .sect .text
05202   .define .rck
05203
05204           ! descriptor address in ebx
05205           ! value in eax, must be left there
05206   .rck:
05207           cmp     eax,(ebx)
05208           jl      2f
05209           cmp     eax,4(ebx)
05210           jg      2f
05211           ret
05212   2:
05213           push    eax
05214   .extern ERANGE
```

```
05215   .extern .error
05216           mov     eax,ERANGE
05217           call    .error
05218           pop     eax
05219           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_rmi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05300   .sect .text; .sect .rom; .sect .data; .sect .bss
05301   .sect .text
05302   .define .rmi
05303
05304           ! #bytes in eax
05305   .rmi:
05306           pop     ebx             ! return address
05307           cmp     eax,4
05308           jne     1f
05309           pop     eax
05310           cwd
05311           pop     ecx
05312           idiv    ecx
05313           push    edx
05314           jmp     ebx
05315   1:
05316   .extern EODDZ
05317   .extern .trp
05318           mov     eax,EODDZ
05319           push    ebx
05320           jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_rmu.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05400   .sect .text; .sect .rom; .sect .data; .sect .bss
05401   .sect .text
05402   .define .rmu
05403
05404           ! #bytes in eax
05405   .rmu:
05406           pop     ebx             ! return address
05407           cmp     eax,4
05408           jne     1f
05409           pop     eax
05410           xor     edx,edx
05411           pop     ecx
05412           idiv    ecx
05413           push    edx
05414           jmp     ebx
```

```
05415    1:
05416    .extern EODDZ
05417    .extern .trp
05418            mov     eax,EODDZ
05419            push    ebx
05420            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_rol.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05500    .sect .text; .sect .rom; .sect .data; .sect .bss
05501    .sect .text
05502    .define .rol
05503
05504            ! #bytes in eax
05505    .rol:
05506            pop     edx             ! return address
05507            cmp     eax,4
05508            jne     1f
05509            pop     eax
05510            pop     ecx
05511            rol     eax,cl
05512            push    eax
05513            jmp     edx
05514    1:
05515    .extern EODDZ
05516    .extern .trp
05517            mov     eax,EODDZ
05518            push    edx
05519            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_ror.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05600    .sect .text; .sect .rom; .sect .data; .sect .bss
05601    .sect .text
05602    .define .ror
05603
05604            ! #bytes in eax
05605    .ror:
05606            pop     edx             ! return address
05607            cmp     eax,4
05608            jne     1f
05609            pop     eax
05610            pop     ecx
05611            ror     eax,cl
05612            push    eax
05613            jmp     edx
05614    1:
```

```
05615   .extern EODDZ
05616   .extern .trp
05617           mov     eax,EODDZ
05618           push    edx
05619           jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sar4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05700   .sect .text; .sect .rom; .sect .data; .sect .bss
05701   .sect .text
05702   .define .sar4
05703
05704   .sar4:
05705                                   ! ebx, descriptor address
05706                                   ! eax, index
05707           sub     eax,(ebx)
05708           mov     ecx,8(ebx)
05709           imul    ecx
05710           pop     ebx
05711           pop     edx             ! base address
05712           add     edx,eax
05713           sar     ecx,1
05714           jnb     1f
05715           pop     eax
05716           movb    (edx),al
05717           jmp     ebx
05718   1:
05719           sar     ecx,1
05720           jnb     1f
05721           pop     eax
05722           o16 mov (edx),ax
05723           jmp     ebx
05724   1:
05725           xchg    edi,edx         ! edi = base address, edx is saved edi
05726           mov     eax,esi
05727           mov     esi,esp
05728           rep movs
05729           mov     esp,esi
05730           mov     esi,eax
05731           mov     edi,edx
05732           jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sbf4.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05800   .sect .text; .sect .rom; .sect .data; .sect .bss
05801   .define .sbf4
05802
05803           .sect .text
05804   .sbf4:
05805           mov     bx,sp
05806           flds    8(bx)
05807           fsubs   4(bx)
05808           fstps   8(bx)
05809           wait
05810           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sbf8.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

05900   .sect .text; .sect .rom; .sect .data; .sect .bss
05901   .define .sbf8
05902
05903           .sect .text
05904   .sbf8:
05905           mov     bx,sp
05906           fldd    12(bx)
05907           fsubd   4(bx)
05908           fstpd   12(bx)
05909           wait
05910           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sbi.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06000   .sect .text; .sect .rom; .sect .data; .sect .bss
06001   .sect .text
06002   .define .sbi
06003
06004           ! #bytes in ecx , top of stack in eax
06005   .sbi:
06006           pop     ebx             ! return subress
06007           cmp     ecx,4
06008           jne     1f
06009           pop     ecx
06010           sub     eax,ecx
06011           neg     eax
06012           jmp     ebx
06013   1:
06014   .extern EODDZ
```

```
06015    .extern .trp
06016            mov     eax,EODDZ
06017            push    ebx
06018            jmp     .trp


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                         src/lib/i386/em/em_set.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06100    .sect .text; .sect .rom; .sect .data; .sect .bss
06101    .sect .text
06102    .define .set
06103
06104            ! #bytes in ecx
06105            ! bit # in eax
06106    .set:
06107            pop     ebx             ! return address
06108            xor     edx,edx
06109    !ifdef create set
06110            sub     esp,ecx
06111            push    ebx
06112            push    edi
06113            mov     ebx,esp
06114            xor     edi,edi
06115            sar     ecx,2
06116    1:
06117            mov     8(ebx)(edi),edx
06118            add     edi,4
06119            loop    1b
06120    !endif
06121            mov     ebx,8
06122            div     ebx
06123            cmp     eax,edi
06124            jae     2f
06125            mov     edi,edx
06126            movb    dl,bits(edi)
06127            mov     edi,esp
06128            add     edi,eax
06129            orb     8(edi),dl
06130            pop     edi
06131            ret
06132    2:
06133    .extern ESET
06134    .extern .trp
06135            pop     edi
06136            mov     eax,ESET
06137            jmp     .trp
06138
06139            .sect .rom
06140    bits:
06141            .data1   1,2,4,8,16,32,64,128
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sli.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06200    .sect .text; .sect .rom; .sect .data; .sect .bss
06201    .sect .text
06202    .define .sli
06203
06204            ! #bytes in eax
06205    .sli:
06206            pop     edx             ! return address
06207            cmp     eax,4
06208            jne     1f
06209            pop     eax
06210            pop     ecx
06211            sal     eax,cl
06212            push    eax
06213            jmp     edx
06214    1:
06215    .extern EODDZ
06216    .extern .trp
06217            mov     eax,EODDZ
06218            push    edx
06219            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sri.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06300    .sect .text; .sect .rom; .sect .data; .sect .bss
06301    .sect .text
06302    .define .sri
06303
06304            ! #bytes in eax
06305    .sri:
06306            pop     edx             ! return address
06307            cmp     eax,4
06308            jne     1f
06309            pop     eax
06310            pop     ecx
06311            sar     eax,cl
06312            push    eax
06313            jmp     edx
06314    1:
06315    .extern EODDZ
06316    .extern .trp
06317            mov     eax,EODDZ
06318            push    edx
06319            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_sti.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06400    .sect .text; .sect .rom; .sect .data; .sect .bss
06401    .sect .text
06402    .define .sti
06403    .define .sts
06404
06405            ! #bytes in ecx
06406            ! address in ebx
06407            ! save edi/esi. they might be register variables
06408    .sts:
06409            pop     edx
06410            sar     ecx,1
06411            jnb     1f
06412            pop     eax
06413            movb    (ebx),al
06414            jmp     edx
06415    1:
06416            sar     ecx,1
06417            jnb     1f
06418            pop     eax
06419            o16 mov (ebx),ax
06420            jmp     edx
06421    1:
06422            push    edx
06423            mov     edx,edi
06424            mov     edi,ebx
06425            pop     ebx
06426            jmp     1f
06427    .sti:
06428            ! only called with count >> 4
06429            mov     edx,edi
06430            mov     edi,ebx
06431            pop     ebx
06432            sar     ecx,2
06433    1:
06434            mov     eax,esi
06435            mov     esi,esp
06436            rep movs
06437            mov     esp,esi
06438            mov     edi,edx
06439            mov     esi,eax
06440            jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_stop.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06500    .sect .text; .sect .rom; .sect .data; .sect .bss
06501    .sect .text
06502    .define .stop
06503    .stop:
06504            jmp     ___exit
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_strhp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06600    .sect .text; .sect .rom; .sect .data; .sect .bss
06601    .sect .text
06602    .define .strhp
06603    .extern .reghp, .limhp, EHEAP, .trp
06604
06605    .strhp:
06606            pop     ebx
06607            pop     eax
06608            mov     (.reghp),eax
06609            cmp     eax,(.limhp)
06610            jb      1f
06611            add     eax,02000
06612            and     eax,~0777
06613            push    ebx
06614            push    eax
06615            call    __brk
06616            pop     ecx
06617            pop     ebx
06618            cmp     eax,-1
06619            je      2f
06620    1:
06621            mov     (.limhp),ecx
06622            jmp     ebx
06623    2:
06624            mov     eax,EHEAP
06625            push    ebx
06626            jmp     .trp
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_trp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06700    .sect .text; .sect .rom; .sect .data; .sect .bss
06701    .sect .text
06702    .define .trp
06703    .extern .trppc, .stop
06704
06705                    ! eax is trap number
06706    .trp:
06707            xor     ebx,ebx
06708            xchg    ebx,(.trppc)
06709            test    ebx,ebx
06710            jz      2f
06711            push    eax
06712            call    ebx
06713            pop     eax
06714            ret
06715    2:
06716            push    eax
06717            call    .stop



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_unknown.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06800    .sect .text; .sect .rom; .sect .data; .sect .bss
06801    .sect .text
06802    .define .unknown
06803    .extern EILLINS, .fat
06804
06805    .unknown:
06806            mov  eax,EILLINS
06807            push eax
06808            jmp  .fat


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/em/em_xor.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

06900    .sect .text; .sect .rom; .sect .data; .sect .bss
06901    .sect .text
06902    .define .xor
06903
06904            ! #bytes in ecx
06905    .xor:
06906            pop     ebx             ! return address
06907            mov     edx,edi
06908            mov     edi,esp
06909            add     edi,ecx
06910            sar     ecx,2
06911    1:
06912            pop     eax
06913            xor     eax,(edi)
06914            stos
```

```
06915           loop    1b
06916           mov     edi,edx
06917           jmp     ebx
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/head/em_abs.h
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07000   /* $Header: em_abs.h,v 1.3 87/03/10 08:58:45 ceriel Exp $ */
07001   /*
07002    * (c) copyright 1987 by the Vrije Universiteit, Amsterdam, The Netherlands.
07003    * See the copyright notice in the ACK home directory, in the file "Copyright".
07004    */
07005   #define LINO_AD         0
07006   #define FILN_AD         4
07007
07008   #define LINO            (*(int    *)(_hol0()+LINO_AD))
07009   #define FILN            (*(char  **)(_hol0()+FILN_AD))
07010
07011   #define EARRAY          0
07012   #define ERANGE          1
07013   #define ESET            2
07014   #define EIOVFL          3
07015   #define EFOVFL          4
07016   #define EFUNFL          5
07017   #define EIDIVZ          6
07018   #define EFDIVZ          7
07019   #define EIUND           8
07020   #define EFUND           9
07021   #define ECONV           10
07022
07023   #define ESTACK          16
07024   #define EHEAP           17
07025   #define EILLINS         18
07026   #define EODDZ           19
07027   #define ECASE           20
07028   #define EMEMFLT         21
07029   #define EBADPTR         22
07030   #define EBADPC          23
07031   #define EBADLAE         24
07032   #define EBADMON         25
07033   #define EBADLIN         26
07034   #define EBADGTO         27
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/head/em_head.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07100    #
07101    .sect .text; .sect .rom; .sect .data; .sect .bss
07102    .define ERANGE,ESET,EHEAP,ECASE,EILLINS,EIDIVZ,EODDZ
07103    .define .trppc, .ignmask
07104
07105    ERANGE          = 1
07106    ESET            = 2
07107    EIDIVZ          = 6
07108    EHEAP           = 17
07109    EILLINS         = 18
07110    EODDZ           = 19
07111    ECASE           = 20
07112
07113    #include <em_abs.h>
07114
07115    .sect .data
07116    .trppc:
07117            .data4  0
07118    .ignmask:
07119            .data4  EIOVFL | EIUND | ECONV | EFOVFL | EFUNFL
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/misc/alloca.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07200    #
07201    !       alloca() - allocate space on the stack          Author: Kees J. Bot
07202    !                                                               2 Dec 1993
07203    .sect .text; .sect .rom; .sect .data; .sect .bss
07204
07205    .sect .text
07206            .align  16
07207    .define _alloca
07208    _alloca:
07209    #if __ACK__
07210            pop     ecx             ! Return address
07211            pop     eax             ! Bytes to allocate
07212            add     eax, 2*4+3      ! Add space for two saved register variables
07213            andb    al, 0xFC        ! Align
07214            mov     ebx, esp        ! Keep current esp
07215            sub     esp, eax        ! Lower stack
07216            mov     eax, esp        ! Return value
07217            push    4(ebx)          ! Push what is probably the saved esi
07218            push    (ebx)           ! Saved edi
07219                                    ! Now ACK can still do:
07220                                    !       pop edi; pop esi; leave; ret
07221            push    eax             ! Dummy argument
07222            jmp     ecx
07223    #else
07224            pop     ecx             ! Return address
```

```
07225           pop     eax             ! Bytes to allocate
07226           add     eax, 3
07227           andb    al, 0xFC        ! Align
07228           sub     esp, eax        ! Lower stack
07229           mov     eax, esp        ! Return value
07230           push    eax             ! Dummy argument
07231           jmp     ecx
07232   #endif
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/misc/getprocessor.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
07300   !       getprocessor() - determine processor type       Author: Kees J. Bot
07301   !                                                               26 Jan 1994
07302
07303   .sect .text; .sect .rom; .sect .data; .sect .bss
07304   .sect .text
07305
07306   ! int getprocessor(void);
07307   !       Return 386, 486, 586, ...
07308
07309   .define _getprocessor
07310
07311   _getprocessor:
07312           push    ebp
07313           mov     ebp, esp
07314           and     esp, 0xFFFFFFFC ! Align stack to avoid AC fault
07315           mov     ecx, 0x00040000 ! Try to flip the AC bit introduced on the 486
07316           call    flip
07317           mov     eax, 386        ! 386 if it didn't react to "flipping"
07318           jz      gotprocessor
07319           mov     ecx, 0x00200000 ! Try to flip the ID bit introduced on the 586
07320           call    flip
07321           mov     eax, 486        ! 486 if it didn't react
07322           jz      gotprocessor
07323           pushf
07324           pusha                   ! Save the world
07325           mov     eax, 1
07326           .data1  0x0F, 0xA2      ! CPUID instruction tells the processor type
07327           andb    ah, 0x0F        ! Extract the family (5, 6, ...)
07328           movzxb  eax, ah
07329           imul    eax, 100        ! 500, 600, ...
07330           add     eax, 86         ! 586, 686, ...
07331           mov     7*4(esp), eax   ! Pass eax through
07332           popa
07333           popf
07334   gotprocessor:
07335           leave
07336           ret
07337
07338   flip:
07339           pushf                   ! Push eflags
07340           pop     eax             ! eax = eflags
07341           mov     edx, eax        ! Save original eflags
07342           xor     eax, ecx        ! Flip the bit to test
07343           push    eax             ! Push modified eflags value
07344           popf                    ! Load modified eflags register
```

```
07345           pushf
07346           pop     eax             ! Get it again
07347           push    edx
07348           popf                    ! Restore original eflags register
07349           xor     eax, edx        ! See if the bit changed
07350           test    eax, ecx
07351           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/misc/iolib.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
07400   ! Some I/O related routines like:
07401   !       unsigned char inb(int port);
07402   !       unsigned short inw(int port);
07403   !       void outb(int port, int value);
07404   !       void outw(int port, int value);
07405   !       void rep_inb(int port, unsigned char *buf, size_t count);
07406   !       void rep_inw(int port, unsigned short *buf, size_t count);
07407   !       void rep_outb(int port, unsigned char *buf, size_t count);
07408   !       void rep_outw(int port, unsigned short *buf, size_t count);
07409   !       void intr_enable(void);
07410   !       void intr_disable(void);
07411
07412   .sect .text; .sect .rom; .sect .data; .sect .bss
07413
07414   .sect .text
07415   .define _inb
07416   _inb:
07417           push    ebp
07418           mov     ebp, esp
07419           mov     edx, 8(ebp)             ! port
07420           xor     eax, eax
07421           inb     dx                      ! read 1 byte
07422           pop     ebp
07423           ret
07424
07425   .define _inw
07426   _inw:
07427           push    ebp
07428           mov     ebp, esp
07429           mov     edx, 8(ebp)             ! port
07430           xor     eax, eax
07431       o16 in      dx                      ! read 1 word
07432           pop     ebp
07433           ret
07434
07435   .define _outb
07436   _outb:
07437           push    ebp
07438           mov     ebp, esp
07439           mov     edx, 8(ebp)             ! port
07440           mov     eax, 8+4(ebp)           ! value
07441           outb    dx                      ! output 1 byte
07442           pop     ebp
07443           ret
07444
```

```
07445   .define _outw
07446   _outw:
07447           push    ebp
07448           mov     ebp, esp
07449           mov     edx, 8(ebp)             ! port
07450           mov     eax, 8+4(ebp)           ! value
07451       o16 out     dx                      ! output 1 word
07452           pop     ebp
07453           ret
07454
07455   .define _rep_inb
07456   _rep_inb:
07457           push    ebp
07458           mov     ebp, esp
07459           push    edi
07460           mov     edx, 8(ebp)             ! port
07461           mov     edi, 12(ebp)            ! buf
07462           mov     ecx, 16(ebp)            ! byte count
07463           rep
07464           inb     dx                      ! input many bytes
07465           pop     edi
07466           pop     ebp
07467           ret
07468
07469   .define _rep_inw
07470   _rep_inw:
07471           push    ebp
07472           mov     ebp, esp
07473           push    edi
07474           mov     edx, 8(ebp)             ! port
07475           mov     edi, 12(ebp)            ! buf
07476           mov     ecx, 16(ebp)            ! byte count
07477           shr     ecx, 1                  ! word count
07478           rep
07479       o16 in      dx                      ! input many words
07480           pop     edi
07481           pop     ebp
07482           ret
07483
07484   .define _rep_outb
07485   _rep_outb:
07486           push    ebp
07487           mov     ebp, esp
07488           push    esi
07489           mov     edx, 8(ebp)             ! port
07490           mov     esi, 12(ebp)            ! buf
07491           mov     ecx, 16(ebp)            ! byte count
07492           rep
07493           outb    dx                      ! output many bytes
07494           pop     esi
07495           pop     ebp
07496           ret
07497
07498   .define _rep_outw
07499   _rep_outw:
07500           push    ebp
07501           mov     ebp, esp
07502           push    esi
07503           mov     edx, 8(ebp)             ! port
07504           mov     esi, 12(ebp)            ! buf
```

```
07505           mov     ecx, 16(ebp)            ! byte count
07506           shr     ecx, 1                  ! word count
07507           rep
07508       o16 out     dx                      ! output many words
07509           pop     esi
07510           pop     ebp
07511           ret
07512
07513   .define _intr_disable
07514   _intr_disable:
07515           push    ebp
07516           mov     ebp, esp
07517           cli
07518           pop     ebp
07519           ret
07520
07521   .define _intr_enable
07522   _intr_enable:
07523           push    ebp
07524           mov     ebp, esp
07525           sti
07526           pop     ebp
07527           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/rts/__sigreturn.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07600   ! This routine is the low-level code for returning from signals.
07601   ! It calls __sigreturn, which is the normal "system call" routine.
07602   ! Both ___sigreturn and __sigreturn are needed.
07603   .sect .text; .sect .rom; .sect .data; .sect .bss
07604   .sect .text
07605   .define ___sigreturn
07606   .extern __sigreturn
07607   ___sigreturn:
07608           add esp, 16
07609           jmp __sigreturn
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/rts/_sendrec.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07700   .sect .text; .sect .rom; .sect .data; .sect .bss
07701   .define __send, __receive, __sendrec
07702
07703   ! See ../h/com.h for C definitions
07704   SEND = 1
07705   RECEIVE = 2
07706   BOTH = 3
07707   SYSVEC = 33
07708
07709   SRCDEST = 8
```

```
07710    MESSAGE = 12
07711
07712    !*===========================================================================*
07713    !                           _send and _receive                              *
07714    !*===========================================================================*
07715    ! _send(), _receive(), _sendrec() all save ebp, but destroy eax and ecx.
07716    .define __send, __receive, __sendrec
07717    .sect .text
07718    __send:
07719            push    ebp
07720            mov     ebp, esp
07721            push    ebx
07722            mov     eax, SRCDEST(ebp)       ! eax = dest-src
07723            mov     ebx, MESSAGE(ebp)       ! ebx = message pointer
07724            mov     ecx, SEND               ! _send(dest, ptr)
07725            int     SYSVEC                  ! trap to the kernel
07726            pop     ebx
07727            pop     ebp
07728            ret
07729
07730    __receive:
07731            push    ebp
07732            mov     ebp, esp
07733            push    ebx
07734            mov     eax, SRCDEST(ebp)       ! eax = dest-src
07735            mov     ebx, MESSAGE(ebp)       ! ebx = message pointer
07736            mov     ecx, RECEIVE            ! _receive(src, ptr)
07737            int     SYSVEC                  ! trap to the kernel
07738            pop     ebx
07739            pop     ebp
07740            ret
07741
07742    __sendrec:
07743            push    ebp
07744            mov     ebp, esp
07745            push    ebx
07746            mov     eax, SRCDEST(ebp)       ! eax = dest-src
07747            mov     ebx, MESSAGE(ebp)       ! ebx = message pointer
07748            mov     ecx, BOTH               ! _sendrec(srcdest, ptr)
07749            int     SYSVEC                  ! trap to the kernel
07750            pop     ebx
07751            pop     ebp
07752            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/rts/brksize.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07800    .sect .text; .sect .rom; .sect .data; .sect .bss
07801    .define __brksize
07802    .sect .data
07803    .extern endbss, __brksize
07804    __brksize: .data4 endbss
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/rts/crtso.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

07900    ! This is the C run-time start-off routine.  It's job is to take the
07901    ! arguments as put on the stack by EXEC, and to parse them and set them up the
07902    ! way _main expects them.
07903    ! It also initializes _environ when this variable isn't defined by the
07904    ! programmer.  The detection of whether _environ belong to us is rather
07905    ! simplistic.  We simply check for some magic value, but there is no other
07906    ! way.
07907
07908    .sect .text; .sect .rom; .sect .data; .sect .bss
07909
07910    .define begtext, begdata, begbss
07911    .sect .text
07912    begtext:
07913    .sect .rom
07914    begrom:
07915    .sect .data
07916    begdata:
07917    .sect .bss
07918    begbss:
07919
07920    .define crtso, ___main, __penvp, __fpu_present
07921    .extern _main, _exit
07922    .sect .text
07923    crtso:
07924            xor     ebp, ebp                ! clear for backtrace of core files
07925            mov     eax, (esp)              ! argc
07926            lea     edx, 4(esp)             ! argv
07927            lea     ecx, 8(esp)(eax*4)      ! envp
07928
07929            mov     (__penvp), ecx          ! save envp in __envp
07930
07931            ! Test whether address of environ < address of end.
07932            ! This is done for separate I&D systems.
07933            mov     ebx, _environ
07934            cmp     ebx, __end
07935            jae     0f
07936            cmp     (_environ), 0x53535353  ! is it our _environ?
07937            jne     0f
07938            mov     (_environ), ecx
07939    0:
07940            push    ecx                     ! push environ
07941            push    edx                     ! push argv
07942            push    eax                     ! push argc
07943
07944            ! Test the EM bit of the MSW to determine if an FPU is present and
07945            ! set __fpu_present if one is found.
07946            smsw    ax
07947            testb   al, 0x4                 ! EM bit in MSW
07948            setz    al                      ! True if not set
07949            movb    (__fpu_present), al
07950
07951            call    _main                   ! main(argc, argv, envp)
07952
07953            push    eax                     ! push exit status
07954            call    _exit
```

```
07955
07956           hlt                             ! force a trap if exit fails
07957
07958   ___main:                                ! for GCC
07959           ret
07960
07961   .sect .rom
07962           .data4  0                       ! Common I&D: *NULL == 0
07963   .sect .bss
07964           .comm   __penvp, 4              ! Environment vector
07965           .comm   __fpu_present, 4        ! FPU present flag
07966
07967   .extern endtext                         ! Force loading of end labels.
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/rts/setjmp.e
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
08000   #
08001    mes 2,_EM_WSIZE,_EM_PSIZE
08002   ;
08003   ; layout of a setjmp buffer:
08004   ;
08005   ;  -----------------
08006   ; |      flag       |           (!0 when blocked signals saved (POSIX))
08007   ;  -----------------
08008   ; | signal mask/set |           (for Berkeley 4.[2-] / POSIX)
08009   ;  -----------------
08010   ; |                 |
08011   ; |  GTO descriptor |
08012   ; |   (SP, LB, PC)  |
08013   ; |                 |
08014   ;  -----------------
08015   ;
08016   ; setjmp saves the signalmask, PC, SP, and LB of caller, and creates a
08017   ; GTO descriptor from this.
08018   ; The big problem here is how to get the return address, i.e. the PC of
08019   ; the caller; This problem is solved by the front-end, which must pass
08020   ; it as an extra parameter to setjmp.
08021
08022   ; a GTO descriptor must be in the global data area
08023   gtobuf
08024    bss 3*_EM_PSIZE,0,0
08025
08026    inp $fill_ret_area
08027    exp $__setjmp
08028    pro $__setjmp,0
08029   #if     defined(_POSIX_SOURCE)
08030   ; save mask of currently blocked signals.
08031   ; longjmp must restore this mask
08032    lol _EM_PSIZE                  ; the flag integer at offset _EM_PSIZE
08033    lal 0
08034    loi _EM_PSIZE
08035    stf 3*_EM_PSIZE+_EM_LSIZE
08036    lol _EM_PSIZE                  ; the flag integer at offset _EM_PSIZE
08037    zeq *1
08038    lal 0
08039    loi _EM_PSIZE
```

```
08040     adp 3*_EM_PSIZE
08041     cal $__newsigset
08042     asp _EM_PSIZE
08043    1
08044    #elif   defined(__BSD4_2)
08045     loc 0
08046     cal $sigblock
08047     asp _EM_WSIZE
08048     lfr _EM_WSIZE
08049     lal 0
08050     loi _EM_PSIZE
08051     stf 3*_EM_PSIZE
08052    #endif
08053    ; create GTO descriptor for longjmp
08054     lxl 0
08055     dch            ; Local Base of caller
08056     lxa 0          ; Stackpointer of caller
08057     lal _EM_PSIZE+_EM_WSIZE
08058     loi _EM_PSIZE  ; Return address of caller
08059     lal 0
08060     loi _EM_PSIZE  ; address of jmpbuf
08061     sti 3*_EM_PSIZE        ; LB, SP, and PC stored in jmpbuf
08062     loc 0
08063     ret _EM_WSIZE  ; setjmp must return 0
08064     end 0
08065
08066     pro $fill_ret_area,0
08067    ; put argument in function result area
08068     lol 0
08069     ret _EM_WSIZE
08070     end 0
08071
08072     exp $longjmp
08073     pro $longjmp,?
08074    #if     defined(_POSIX_SOURCE)
08075    ; restore blocked mask
08076     lal 0
08077     loi _EM_PSIZE
08078     lof 3*_EM_PSIZE+_EM_LSIZE
08079     zeq *2
08080     lal 0
08081     loi _EM_PSIZE
08082     adp 3*_EM_PSIZE
08083     cal $__oldsigset
08084     asp _EM_PSIZE
08085    2
08086    #elif   defined(__BSD4_2)
08087    ; restore signal mask
08088     lal 0
08089     loi _EM_PSIZE
08090     lof 3*_EM_PSIZE
08091     cal $_sigsetmask
08092     asp _EM_WSIZE
08093     lfr _EM_WSIZE
08094     asp _EM_WSIZE
08095    #endif
08096     lal 0
08097     loi _EM_PSIZE  ; address of jmpbuf
08098     lae gtobuf
08099     blm 3*_EM_PSIZE        ; fill GTO descriptor from jmpbuf
```

```
08100     lol _EM_PSIZE  ; second parameter of longjmp: the return value
08101     dup _EM_WSIZE
08102     zne *3
08103    ; of course, longjmp may not return 0!
08104     inc
08105    3
08106    ; put return value in function result area
08107     cal $fill_ret_area
08108     asp _EM_WSIZE
08109     gto gtobuf     ; there we go ...
08110    ; ASP and GTO do not damage function result area
08111     end 0
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/_memmove.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08200    !       _memmove()                                      Author: Kees J. Bot
08201    !                                                               2 Jan 1994
08202    .sect .text; .sect .rom; .sect .data; .sect .bss
08203
08204    ! void *_memmove(void *s1, const void *s2, size_t n)
08205    !       Copy a chunk of memory.  Handle overlap.
08206    !
08207    .sect .text
08208    .define __memmove, __memcpy
08209            .align  16
08210    __memmove:
08211            push    ebp
08212            mov     ebp, esp
08213            push    esi
08214            push    edi
08215            mov     edi, 8(ebp)     ! String s1
08216            mov     esi, 12(ebp)    ! String s2
08217            mov     ecx, 16(ebp)    ! Length
08218            mov     eax, edi
08219            sub     eax, esi
08220            cmp     eax, ecx
08221            jb      downwards       ! if (s2 - s1) < n then copy downwards
08222    __memcpy:
08223            cld                     ! Clear direction bit: upwards
08224            cmp     ecx, 16
08225            jb      upbyte          ! Don't bother being smart with short arrays
08226            mov     eax, esi
08227            or      eax, edi
08228            testb   al, 1
08229            jnz     upbyte          ! Bit 0 set, use byte copy
08230            testb   al, 2
08231            jnz     upword          ! Bit 1 set, use word copy
08232    uplword:shrd    eax, ecx, 2     ! Save low 2 bits of ecx in eax
08233            shr     ecx, 2
08234            rep
08235            movs                    ! Copy longwords.
08236            shld    ecx, eax, 2     ! Restore excess count
08237    upword: shr     ecx, 1
08238            rep
08239        o16 movs                    ! Copy words
```

```
08240           adc     ecx, ecx        ! One more byte?
08241   upbyte: rep
08242           movsb                   ! Copy bytes
08243   done:   mov     eax, 8(ebp)     ! Absolutely noone cares about this value
08244           pop     edi
08245           pop     esi
08246           pop     ebp
08247           ret
08248
08249   ! Handle bad overlap by copying downwards, don't bother to do word copies.
08250   downwards:
08251           std                     ! Set direction bit: downwards
08252           lea     esi, -1(esi)(ecx*1)
08253           lea     edi, -1(edi)(ecx*1)
08254           rep
08255           movsb                   ! Copy bytes
08256           cld
08257           jmp     done
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/_strncat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08300   !       _strncat()                                      Author: Kees J. Bot
08301   !                                                               1 Jan 1994
08302   .sect .text; .sect .rom; .sect .data; .sect .bss
08303
08304   ! char *_strncat(char *s1, const char *s2, size_t edx)
08305   !       Append string s2 to s1.
08306   !
08307   .sect .text
08308   .define __strncat
08309           .align  16
08310   __strncat:
08311           push    ebp
08312           mov     ebp, esp
08313           push    esi
08314           push    edi
08315           mov     edi, 8(ebp)     ! String s1
08316           mov     ecx, -1
08317           xorb    al, al          ! Null byte
08318           cld
08319           repne
08320           scasb                   ! Look for the zero byte in s1
08321           dec     edi             ! Back one up (and clear 'Z' flag)
08322           push    edi             ! Save end of s1
08323           mov     edi, 12(ebp)    ! edi = string s2
08324           mov     ecx, edx        ! Maximum count
08325           repne
08326           scasb                   ! Look for the end of s2
08327           jne     no0
08328           inc     ecx             ! Exclude null byte
08329   no0:    sub     edx, ecx        ! Number of bytes in s2
08330           mov     ecx, edx
08331           mov     esi, 12(ebp)    ! esi = string s2
08332           pop     edi             ! edi = end of string s1
08333           rep
08334           movsb                   ! Copy bytes
```

```
08335           stosb                   ! Add a terminating null
08336           mov     eax, 8(ebp)     ! Return s1
08337           pop     edi
08338           pop     esi
08339           pop     ebp
08340           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/_strncmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08400   !       strncmp()                                       Author: Kees J. Bot
08401   !                                                               1 Jan 1994
08402   .sect .text; .sect .rom; .sect .data; .sect .bss
08403
08404   ! int strncmp(const char *s1, const char *s2, size_t ecx)
08405   !       Compare two strings.
08406   !
08407   .sect .text
08408   .define __strncmp
08409           .align  16
08410   __strncmp:
08411           push    ebp
08412           mov     ebp, esp
08413           push    esi
08414           push    edi
08415           test    ecx, ecx        ! Max length is zero?
08416           je      done
08417           mov     esi, 8(ebp)     ! esi = string s1
08418           mov     edi, 12(ebp)    ! edi = string s2
08419           cld
08420   compare:
08421           cmpsb                   ! Compare two bytes
08422           jne     done
08423           cmpb    -1(esi), 0      ! End of string?
08424           je      done
08425           dec     ecx             ! Length limit reached?
08426           jne     compare
08427   done:   seta    al              ! al = (s1 > s2)
08428           setb    ah              ! ah = (s1 < s2)
08429           subb    al, ah
08430           movsxb  eax, al         ! eax = (s1 > s2) - (s1 < s2), i.e. -1, 0, 1
08431           pop     edi
08432           pop     esi
08433           pop     ebp
08434           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/_strncpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08500   !       _strncpy()                                      Author: Kees J. Bot
08501   !                                                               1 Jan 1994
08502   .sect .text; .sect .rom; .sect .data; .sect .bss
08503
08504   ! char *_strncpy(char *s1, const char *s2, size_t ecx)
08505   !       Copy string s2 to s1.
08506   !
08507   .sect .text
08508   .define __strncpy
08509           .align  16
08510   __strncpy:
08511           mov     edi, 12(ebp)    ! edi = string s2
08512           xorb    al, al          ! Look for a zero byte
08513           mov     edx, ecx        ! Save maximum count
08514           cld
08515           repne
08516           scasb                   ! Look for end of s2
08517           sub     edx, ecx        ! Number of bytes in s2 including null
08518           xchg    ecx, edx
08519           mov     esi, 12(ebp)    ! esi = string s2
08520           mov     edi, 8(ebp)     ! edi = string s1
08521           rep
08522           movsb                   ! Copy bytes
08523           ret


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/_strnlen.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08600   !       _strnlen()                                      Author: Kees J. Bot
08601   !                                                               1 Jan 1994
08602   .sect .text; .sect .rom; .sect .data; .sect .bss
08603
08604   ! size_t _strnlen(const char *s, size_t ecx)
08605   !       Return the length of a string.
08606   !
08607   .sect .text
08608   .define __strnlen
08609           .align  16
08610   __strnlen:
08611           push    ebp
08612           mov     ebp, esp
08613           push    edi
08614           mov     edi, 8(ebp)     ! edi = string
08615           xorb    al, al          ! Look for a zero byte
08616           mov     edx, ecx        ! Save maximum count
08617           cmpb    cl, 1           ! 'Z' bit must be clear if ecx = 0
08618           cld
08619           repne
08620           scasb                   ! Look for zero
08621           jne     no0
08622           inc     ecx             ! Don't count zero byte
08623   no0:    mov     eax, edx
08624           sub     eax, ecx        ! Compute bytes scanned
```

```
08625           pop     edi
08626           pop     ebp
08627           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/bcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08700   !       bcmp()                                          Author: Kees J. Bot
08701   !                                                               2 Jan 1994
08702   .sect .text; .sect .rom; .sect .data; .sect .bss
08703
08704   ! int bcmp(const void *s1, const void *s2, size_t n)
08705   !       Compare two chunks of memory.
08706   !       This is a BSD routine that escaped from the kernel.  Don't use.
08707   !       (Alas it is not without some use, it reports the number of bytes
08708   !       after the bytes that are equal.  So it can't be simply replaced.)
08709   !
08710   .sect .text
08711   .define _bcmp
08712           .align  16
08713   _bcmp:
08714           push    ebp
08715           mov     ebp, esp
08716           push    16(ebp)
08717           push    12(ebp)
08718           push    8(ebp)
08719           call    _memcmp         ! Let memcmp do the work
08720           test    eax, eax
08721           jz      equal
08722           sub     edx, 8(ebp)     ! Memcmp was nice enough to leave "esi" in edx
08723           dec     edx             ! Number of bytes that are equal
08724           mov     eax, 16(ebp)
08725           sub     eax, edx        ! Number of bytes that are unequal
08726   equal:  leave
08727           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/bcopy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08800   !       bcopy()                                         Author: Kees J. Bot
08801   !                                                               2 Jan 1994
08802   .sect .text; .sect .rom; .sect .data; .sect .bss
08803
08804   ! void bcopy(const void *s1, void *s2, size_t n)
08805   !       Copy a chunk of memory.  Handle overlap.
08806   !       This is a BSD routine that escaped from the kernel.  Don't use.
08807   !
08808   .sect .text
08809   .define _bcopy
08810           .align  16
08811   _bcopy:
08812           mov     eax, 4(esp)     ! Exchange string arguments
08813           xchg    eax, 8(esp)
08814           mov     4(esp), eax
```

```
08815           jmp     __memmove       ! Call the proper routine
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/bzero.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

08900   !       bzero()                                         Author: Kees J. Bot
08901   !                                                               2 Jan 1994
08902   .sect .text; .sect .rom; .sect .data; .sect .bss
08903
08904   ! void bzero(void *s, size_t n)
08905   !       Set a chunk of memory to zero.
08906   !       This is a BSD routine that escaped from the kernel.  Don't use.
08907   !
08908   .sect .text
08909   .define _bzero
08910           .align  16
08911   _bzero:
08912           push    ebp
08913           mov     ebp, esp
08914           push    12(ebp)         ! Size
08915           push    0               ! Zero
08916           push    8(ebp)          ! String
08917           call    _memset         ! Call the proper routine
08918           leave
08919           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/index.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09000   !       index()                                         Author: Kees J. Bot
09001   !                                                               2 Jan 1994
09002   .sect .text; .sect .rom; .sect .data; .sect .bss
09003
09004   ! char *index(const char *s, int c)
09005   !       Look for a character in a string.  Has suffered from a hostile
09006   !       takeover by strchr().
09007   !
09008   .sect .text
09009   .define _index
09010           .align  16
09011   _index:
09012           jmp     _strchr
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/memchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09100    !       memchr()                                        Author: Kees J. Bot
09101    !                                                               2 Jan 1994
09102    .sect .text; .sect .rom; .sect .data; .sect .bss
09103
09104    ! void *memchr(const void *s, int c, size_t n)
09105    !       Look for a character in a chunk of memory.
09106    !
09107    .sect .text
09108    .define _memchr
09109            .align  16
09110    _memchr:
09111            push    ebp
09112            mov     ebp, esp
09113            push    edi
09114            mov     edi, 8(ebp)     ! edi = string
09115            movb    al, 12(ebp)     ! The character to look for
09116            mov     ecx, 16(ebp)    ! Length
09117            cmpb    cl, 1           ! 'Z' bit must be clear if ecx = 0
09118            cld
09119            repne
09120            scasb
09121            jne     failure
09122            lea     eax, -1(edi)    ! Found
09123            pop     edi
09124            pop     ebp
09125            ret
09126    failure:xor     eax, eax
09127            pop     edi
09128            pop     ebp
09129            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/memcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09200    !       memcmp()                                        Author: Kees J. Bot
09201    !                                                               2 Jan 1994
09202    .sect .text; .sect .rom; .sect .data; .sect .bss
09203
09204    ! int memcmp(const void *s1, const void *s2, size_t n)
09205    !       Compare two chunks of memory.
09206    !
09207    .sect .text
09208    .define _memcmp
09209            .align  16
09210    _memcmp:
09211            cld
09212            push    ebp
09213            mov     ebp, esp
09214            push    esi
```

```
09215           push    edi
09216           mov     esi, 8(ebp)     ! String s1
09217           mov     edi, 12(ebp)    ! String s2
09218           mov     ecx, 16(ebp)    ! Length
09219           cmp     ecx, 16
09220           jb      cbyte           ! Don't bother being smart with short arrays
09221           mov     eax, esi
09222           or      eax, edi
09223           testb   al, 1
09224           jnz     cbyte           ! Bit 0 set, use byte compare
09225           testb   al, 2
09226           jnz     cword           ! Bit 1 set, use word compare
09227   clword: shrd    eax, ecx, 2     ! Save low two bits of ecx in eax
09228           shr     ecx, 2
09229           repe
09230           cmps                    ! Compare longwords
09231           sub     esi, 4
09232           sub     edi, 4
09233           inc     ecx             ! Recompare the last longword
09234           shld    ecx, eax, 2     ! And any excess bytes
09235           jmp     last
09236   cword:  shrd    eax, ecx, 1     ! Save low bit of ecx in eax
09237           shr     ecx, 1
09238           repe
09239       o16 cmps                    ! Compare words
09240           sub     esi, 2
09241           sub     edi, 2
09242           inc     ecx             ! Recompare the last word
09243           shld    ecx, eax, 1     ! And one more byte?
09244   cbyte:  test    ecx, ecx        ! Set 'Z' flag if ecx = 0
09245   last:   repe
09246           cmpsb                   ! Look for the first differing byte
09247           seta    al              ! al = (s1 > s2)
09248           setb    ah              ! ah = (s1 < s2)
09249           subb    al, ah
09250           movsxb  eax, al         ! eax = (s1 > s2) - (s1 < s2), i.e. -1, 0, 1
09251           mov     edx, esi        ! For bcmp() to play with
09252           pop     edi
09253           pop     esi
09254           pop     ebp
09255           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/memcpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09300   !       memcpy()                                        Author: Kees J. Bot
09301   !                                                               2 Jan 1994
09302   .sect .text; .sect .rom; .sect .data; .sect .bss
09303
09304   ! void *memcpy(void *s1, const void *s2, size_t n)
09305   !       Copy a chunk of memory.
09306   !       This routine need not handle overlap, so it does not handle overlap.
09307   !       One could simply call __memmove, the cost of the overlap check is
09308   !       negligible, but you are dealing with a programmer who believes that
09309   !       if anything can go wrong, it should go wrong.
```

```
09310   !
09311   .sect .text
09312   .define _memcpy
09313           .align  16
09314   _memcpy:
09315           push    ebp
09316           mov     ebp, esp
09317           push    esi
09318           push    edi
09319           mov     edi, 8(ebp)     ! String s1
09320           mov     esi, 12(ebp)    ! String s2
09321           mov     ecx, 16(ebp)    ! Length
09322           ! No overlap check here
09323           jmp     __memcpy        ! Call the part of __memmove that copies up


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/memmove.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09400   !       memmove()                                       Author: Kees J. Bot
09401   !                                                               2 Jan 1994
09402   .sect .text; .sect .rom; .sect .data; .sect .bss
09403
09404   ! void *memmove(void *s1, const void *s2, size_t n)
09405   !       Copy a chunk of memory.  Handle overlap.
09406   !
09407   .sect .text
09408   .define _memmove
09409           .align  16
09410   _memmove:
09411           jmp     __memmove       ! Call common code



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/memset.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09500   !       memset()                                        Author: Kees J. Bot
09501   !                                                               2 Jan 1994
09502   .sect .text; .sect .rom; .sect .data; .sect .bss
09503
09504   ! void *memset(void *s, int c, size_t n)
09505   !       Set a chunk of memory to the same byte value.
09506   !
09507   .sect .text
09508   .define _memset
09509           .align  16
09510   _memset:
09511           push    ebp
09512           mov     ebp, esp
09513           push    edi
09514           mov     edi, 8(ebp)     ! The string
09515           movzxb  eax, 12(ebp)    ! The fill byte
09516           mov     ecx, 16(ebp)    ! Length
09517           cld
09518           cmp     ecx, 16
09519           jb      sbyte           ! Don't bother being smart with short arrays
```

```
09520            test    edi, 1
09521            jnz     sbyte           ! Bit 0 set, use byte store
09522            test    edi, 2
09523            jnz     sword           ! Bit 1 set, use word store
09524    slword: movb    ah, al
09525            mov     edx, eax
09526            sal     edx, 16
09527            or      eax, edx        ! One byte to four bytes
09528            shrd    edx, ecx, 2     ! Save low two bits of ecx in edx
09529            shr     ecx, 2
09530            rep
09531            stos                    ! Store longwords.
09532            shld    ecx, edx, 2     ! Restore low two bits
09533    sword:  movb    ah, al          ! One byte to two bytes
09534            shr     ecx, 1
09535            rep
09536        o16 stos                    ! Store words
09537            adc     ecx, ecx        ! One more byte?
09538    sbyte:  rep
09539            stosb                   ! Store bytes
09540    done:   mov     eax, 8(ebp)     ! Return some value you have no need for
09541            pop     edi
09542            pop     ebp
09543            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/rindex.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09600    !       rindex()                                        Author: Kees J. Bot
09601    !                                                               2 Jan 1994
09602    .sect .text; .sect .rom; .sect .data; .sect .bss
09603
09604    ! char *rindex(const char *s, int c)
09605    !       Look for the last occurrence a character in a string.  Has suffered
09606    !       from a hostile takeover by strrchr().
09607    !
09608    .sect .text
09609    .define _rindex
09610            .align  16
09611    _rindex:
09612            jmp     _strrchr
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strcat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09700    !       strcat()                                        Author: Kees J. Bot
09701    !                                                               1 Jan 1994
09702    .sect .text; .sect .rom; .sect .data; .sect .bss
09703
09704    ! char *strcat(char *s1, const char *s2)
09705    !       Append string s2 to s1.
09706    !
09707    .sect .text
09708    .define _strcat
09709            .align  16
```

```
09710   _strcat:
09711           mov     edx, -1         ! Unlimited length
09712           jmp     __strncat       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
09800   !       strchr()                                        Author: Kees J. Bot
09801   !                                                               1 Jan 1994
09802   .sect .text; .sect .rom; .sect .data; .sect .bss
09803
09804   ! char *strchr(const char *s, int c)
09805   !       Look for a character in a string.
09806   !
09807   .sect .text
09808   .define _strchr
09809           .align  16
09810   _strchr:
09811           push    ebp
09812           mov     ebp, esp
09813           push    edi
09814           cld
09815           mov     edi, 8(ebp)     ! edi = string
09816           mov     edx, 16         ! Look at small chunks of the string
09817   next:   shl     edx, 1          ! Chunks become bigger each time
09818           mov     ecx, edx
09819           xorb    al, al          ! Look for the zero at the end
09820           repne
09821           scasb
09822           pushf                   ! Remember the flags
09823           sub     ecx, edx
09824           neg     ecx             ! Some or all of the chunk
09825           sub     edi, ecx        ! Step back
09826           movb    al, 12(ebp)     ! The character to look for
09827           repne
09828           scasb
09829           je      found
09830           popf                    ! Did we find the end of string earlier?
09831           jne     next            ! No, try again
09832           xor     eax, eax        ! Return NULL
09833           pop     edi
09834           pop     ebp
09835           ret
09836   found:  pop     eax             ! Get rid of those flags
09837           lea     eax, -1(edi)    ! Address of byte found
09838           pop     edi
09839           pop     ebp
09840           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strcmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

09900    !       strcmp()                                        Author: Kees J. Bot
09901    !                                                               1 Jan 1994
09902    .sect .text; .sect .rom; .sect .data; .sect .bss
09903
09904    ! int strcmp(const char *s1, const char *s2)
09905    !       Compare two strings.
09906    !
09907    .sect .text
09908    .define _strcmp
09909            .align  16
09910    _strcmp:
09911            mov     ecx, -1         ! Unlimited length
09912            jmp     __strncmp       ! Common code



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strcpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10000    !       strcpy()                                        Author: Kees J. Bot
10001    !                                                               1 Jan 1994
10002    .sect .text; .sect .rom; .sect .data; .sect .bss
10003
10004    ! char *strcpy(char *s1, const char *s2)
10005    !       Copy string s2 to s1.
10006    !
10007    .sect .text
10008    .define _strcpy
10009            .align  16
10010    _strcpy:
10011            push    ebp
10012            mov     ebp, esp
10013            push    esi
10014            push    edi
10015            mov     ecx, -1         ! Unlimited length
10016            call    __strncpy       ! Common code
10017            mov     eax, 8(ebp)     ! Return s1
10018            pop     edi
10019            pop     esi
10020            pop     ebp
10021            ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strlen.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10100    !       strlen()                                        Author: Kees J. Bot
10101    !                                                               1 Jan 1994
10102    .sect .text; .sect .rom; .sect .data; .sect .bss
10103
10104    ! size_t strlen(const char *s)
10105    !       Return the length of a string.
10106    !
10107    .sect .text
10108    .define _strlen
10109            .align  16
10110    _strlen:
10111            mov     ecx, -1         ! Unlimited length
10112            jmp     __strnlen       ! Common code



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strncat.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10200    !       strncat()                                       Author: Kees J. Bot
10201    !                                                               1 Jan 1994
10202    .sect .text; .sect .rom; .sect .data; .sect .bss
10203
10204    ! size_t strncat(char *s1, const char *s2, size_t n)
10205    !       Append string s2 to s1.
10206    !
10207    .sect .text
10208    .define _strncat
10209            .align  16
10210    _strncat:
10211            mov     edx, 12(esp)    ! Maximum length
10212            jmp     __strncat       ! Common code



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strncmp.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10300    !       strncmp()                                       Author: Kees J. Bot
10301    !                                                               1 Jan 1994
10302    .sect .text; .sect .rom; .sect .data; .sect .bss
10303
10304    ! int strncmp(const char *s1, const char *s2, size_t n)
10305    !       Compare two strings.
10306    !
10307    .sect .text
10308    .define _strncmp
10309            .align  16
10310    _strncmp:
10311            mov     ecx, 12(esp)    ! Maximum length
10312            jmp     __strncmp       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strncpy.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10400   !       strncpy()                                       Author: Kees J. Bot
10401   !                                                               1 Jan 1994
10402   .sect .text; .sect .rom; .sect .data; .sect .bss
10403
10404   ! char *strncpy(char *s1, const char *s2, size_t n)
10405   !       Copy string s2 to s1.
10406   !
10407   .sect .text
10408   .define _strncpy
10409           .align  16
10410   _strncpy:
10411           push    ebp
10412           mov     ebp, esp
10413           push    esi
10414           push    edi
10415           mov     ecx, 16(ebp)    ! Maximum length
10416           call    __strncpy       ! Common code
10417           mov     ecx, edx        ! Number of bytes not copied
10418           rep
10419           stosb                   ! strncpy always copies n bytes by null padding
10420           mov     eax, 8(ebp)     ! Return s1
10421           pop     edi
10422           pop     esi
10423           pop     ebp
10424           ret
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strnlen.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10500   !       strnlen()                                       Author: Kees J. Bot
10501   !                                                               1 Jan 1994
10502   .sect .text; .sect .rom; .sect .data; .sect .bss
10503
10504   ! size_t strnlen(const char *s, size_t n)
10505   !       Return the length of a string.
10506   !
10507   .sect .text
10508   .define _strnlen
10509           .align  16
10510   _strnlen:
10511           mov     ecx, 8(esp)     ! Maximum length
10512           jmp     __strnlen       ! Common code
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        src/lib/i386/string/strrchr.s
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

10600   !       strrchr()                                       Author: Kees J. Bot
10601   !                                                               2 Jan 1994
10602   .sect .text; .sect .rom; .sect .data; .sect .bss
10603
10604   ! char *strrchr(const char *s, int c)
10605   !       Look for the last occurrence a character in a string.
10606   !
10607   .sect .text
10608   .define _strrchr
10609           .align  16
10610   _strrchr:
10611           push    ebp
10612           mov     ebp, esp
10613           push    edi
10614           mov     edi, 8(ebp)     ! edi = string
10615           mov     ecx, -1
10616           xorb    al, al
10617           cld
10618           repne
10619           scasb                   ! Look for the end of the string
10620           not     ecx             ! -1 - ecx = Length of the string + null
10621           dec     edi             ! Put edi back on the zero byte
10622           movb    al, 12(ebp)     ! The character to look for
10623           std                     ! Downwards search
10624           repne
10625           scasb
10626           cld                     ! Direction bit back to default
10627           jne     failure
10628           lea     eax, 1(edi)     ! Found it
10629           pop     edi
10630           pop     ebp
10631           ret
10632   failure:xor     eax, eax        ! Not there
10633           pop     edi
10634           pop     ebp
10635           ret
```