

程序 14-9 linux/include/string.h

```

1 #ifndef STRING_H
2 #define STRING_H
3
4 #ifndef NULL
5 #define NULL ((void *) 0)
6 #endif
7
8 #ifndef SIZE_T
9 #define SIZE_T
10 typedef unsigned int size_t;
11 #endif
12
13 extern char * strerror(int errno);
14
15 /*
16  * This string-include defines all string functions as inline
17  * functions. Use gcc. It also assumes ds=es=data space, this should be
18  * normal. Most of the string-functions are rather heavily hand-optimized,
19  * see especially strtok, strstr, str[c]spn. They should work, but are not
20  * very easy to understand. Everything is done entirely within the register
21  * set, making the functions fast and clean. String instructions have been
22  * used through-out, making for "slightly" unclear code :-)
23  *
24  * (C) 1991 Linus Torvalds
25  */
26
27  * 这个字符串头文件以内嵌函数的形式定义了所有字符串操作函数。使用 gcc 时，同时
28  * 假定了 ds=es=数据空间，这应该是常规的。绝大多数字符串函数都是经手工进行大量
29  * 优化的，尤其是函数 strtok、strstr、str[c]spn。它们应该能正常工作，但却不是那
30  * 么容易理解。所有的操作基本上都是使用寄存器集来完成的，这使得函数即快又整洁。
31  * 所有地方都使用了字符串指令，这又使得代码“稍微”难以理解☺
32  *
33  * (C) 1991 Linus Torvalds
34  */
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

    // %0 - esi(src), %1 - edi(dest), %2 - ecx(count)。
38 extern inline char * strncpy(char * dest, const char * src, int count)
39 {
40     __asm__( "cld\n" // 清方向位。
41             "1:|tdecl %2\n|t" // 寄存器 ecx-- (count--)。
42             "js 2f\n|t" // 如果 count<0 则向前跳转到标号 2, 结束。
43             "lodsbl\n|t" // 取 ds:[esi]处 1 字节→al, 并且 esi++。
44             "stosbl\n|t" // 存储该字节→es:[edi], 并且 edi++。
45             "testb %%al, %%al\n|t" // 该字节是 0?
46             "jne 1b\n|t" // 不是, 则向前跳转到标号 1 处继续拷贝。
47             "rep\n|t" // 否则, 在目的串中存放剩余个数的空字符。
48             "stosb\n"
49             "2:"
50             :: "S" (src), "D" (dest), "c" (count): "si", "di", "ax", "cx");
51 return dest; // 返回目的字符串指针。
52 }
53
54 // 将源字符串拷贝到目的字符串的末尾处。
55 // 参数: dest - 目的字符串指针, src - 源字符串指针。
56 // %0 - esi(src), %1 - edi(dest), %2 - eax(0), %3 - ecx(-1)。
57 extern inline char * strcat(char * dest, const char * src)
58 {
59     __asm__( "cld\n|t" // 清方向位。
60             "repne\n|t" // 比较 al 与 es:[edi]字节, 并更新 edi++,
61             "scasb\n|t" // 直到找到目的串中是 0 的字节, 此时 edi 已指向后 1 字节。
62             "decl %l\n|t" // 让 es:[edi]指向 0 值字节。
63             "1:|tlodsb\n|t" // 取源字符串字节 ds:[esi]→al, 并 esi++。
64             "stosb\n|t" // 将该字节存到 es:[edi], 并 edi++。
65             "testb %%al, %%al\n|t" // 该字节是 0?
66             "jne 1b" // 不是, 则向后跳转到标号 1 处继续拷贝, 否则结束。
67             :: "S" (src), "D" (dest), "a" (0), "c" (0xffffffff): "si", "di", "ax", "cx");
68 return dest; // 返回目的字符串指针。
69 }
70
71 // 将源字符串的 count 个字节复制到目的字符串的末尾处, 最后添一空字符。
72 // 参数: dest - 目的字符串, src - 源字符串, count - 欲复制的字节数。
73 // %0 - esi(src), %1 - edi(dest), %2 - eax(0), %3 - ecx(-1), %4 - (count)。
74 extern inline char * strncat(char * dest, const char * src, int count)
75 {
76     __asm__( "cld\n|t" // 清方向位。
77             "repne\n|t" // 比较 al 与 es:[edi]字节, edi++。
78             "scasb\n|t" // 直到找到目的串的末端 0 值字节。
79             "decl %l\n|t" // edi 指向该 0 值字节。
80             "movl %4, %3\n" // 欲复制字节数→ecx。
81             "1:|tdecl %3\n|t" // ecx-- (从 0 开始计数)。
82             "js 2f\n|t" // ecx < 0 ?, 是则向前跳转到标号 2 处。
83             "lodsbl\n|t" // 否则取 ds:[esi]处的字节→al, esi++。
84             "stosbl\n|t" // 存储到 es:[edi]处, edi++。
85             "testb %%al, %%al\n|t" // 该字节值为 0?
86             "jne 1b\n" // 不是则向后跳转到标号 1 处, 继续复制。
87             "2:|txorl %2, %2\n|t" // 将 al 清零。
88             "stosb" // 存到 es:[edi]处。
89             :: "S" (src), "D" (dest), "a" (0), "c" (0xffffffff), "g" (count)

```

```

84         : "si", "di", "ax", "cx");
85 return dest; // 返回目的字符串指针。
86 }
87
88 // 将一个字符串与另一个字符串进行比较。
89 // 参数: cs - 字符串 1, ct - 字符串 2。
90 // %0 - eax(__res)返回值, %1 - edi(cs)字符串 1 指针, %2 - esi(ct)字符串 2 指针。
91 // 返回: 如果串 1 > 串 2, 则返回 1; 串 1 = 串 2, 则返回 0; 串 1 < 串 2, 则返回-1。
92 // 第 90 行定义了一个局部寄存器变量。该变量将被保存在 eax 寄存器中, 以便于高效访问和操作。
93 // 这种定义变量的方法主要用于内嵌汇编程序中。详细说明参见 gcc 手册“指定寄存器中的变量”。
88 extern inline int strcmp(const char * cs, const char * ct)
89 {
90 register int __res __asm__ ("ax"); // __res 是寄存器变量(eax)。
91 __asm__ ("cld\n" // 清方向位。
92 "1:|t lodsb\n|t" // 取字符串 2 的字节 ds:[esi]→al, 并且 esi++。
93 "scasb\n|t" // al 与字符串 1 的字节 es:[edi]作比较, 并且 edi++。
94 "jne 2f\n|t" // 如果不相等, 则向前跳转到标号 2。
95 "testb %%al, %%al\n|t" // 该字节是 0 值字节吗(字符串结尾)?
96 "jne 1b\n|t" // 不是, 则向后跳转到标号 1, 继续比较。
97 "xorl %%eax, %%eax\n|t" // 是, 则返回值 eax 清零,
98 "jmp 3f\n" // 向前跳转到标号 3, 结束。
99 "2:|tmovl $1, %%eax\n|t" // eax 中置 1。
100 "jl 3f\n|t" // 若前面比较中串 2 字符<串 1 字符, 则返回正值结束。
101 "negl %%eax\n" // 否则 eax = -eax, 返回负值, 结束。
102 "3:"
103 : "=a" (__res): "D" (cs), "S" (ct): "si", "di");
104 return __res; // 返回比较结果。
105 }
106
107 // 字符串 1 与字符串 2 的前 count 个字符进行比较。
108 // 参数: cs - 字符串 1, ct - 字符串 2, count - 比较的字符数。
109 // %0 - eax(__res)返回值, %1 - edi(cs)串 1 指针, %2 - esi(ct)串 2 指针, %3 - ecx(count)。
110 // 返回: 如果串 1 > 串 2, 则返回 1; 串 1 = 串 2, 则返回 0; 串 1 < 串 2, 则返回-1。
107 extern inline int strncmp(const char * cs, const char * ct, int count)
108 {
109 register int __res __asm__ ("ax"); // __res 是寄存器变量(eax)。
110 __asm__ ("cld\n" // 清方向位。
111 "1:|t decl %3\n|t" // count--。
112 "js 2f\n|t" // 如果 count<0, 则向前跳转到标号 2。
113 "lodsbl\n|t" // 取串 2 的字符 ds:[esi]→al, 并且 esi++。
114 "scasbl\n|t" // 比较 al 与串 1 的字符 es:[edi], 并且 edi++。
115 "jne 3f\n|t" // 如果不相等, 则向前跳转到标号 3。
116 "testb %%al, %%al\n|t" // 该字符是 NULL 字符吗?
117 "jne 1b\n" // 不是, 则向后跳转到标号 1, 继续比较。
118 "2:|txorl %%eax, %%eax\n|t" // 是 NULL 字符, 则 eax 清零(返回值)。
119 "jmp 4f\n" // 向前跳转到标号 4, 结束。
120 "3:|tmovl $1, %%eax\n|t" // eax 中置 1。
121 "jl 4f\n|t" // 如果前面比较中串 2 字符<串 1 字符, 则返回 1 结束。
122 "negl %%eax\n" // 否则 eax = -eax, 返回负值, 结束。
123 "4:"
124 : "=a" (__res): "D" (cs), "S" (ct), "c" (count): "si", "di", "cx");
125 return __res; // 返回比较结果。
126 }

```

127

```
//// 在字符串中寻找第一个匹配的字符。  
// 参数: s - 字符串, c - 欲寻找的字符。  
// %0 - eax(__res), %1 - esi(字符串指针 s), %2 - eax(字符 c)。  
// 返回: 返回字符串中第一次出现匹配字符的指针。若没有找到匹配的字符, 则返回空指针。
```

128 extern inline char * [strchr](#)(const char * s, char c)

129 {

130 register char * __res __asm__("ax"); // __res 是寄存器变量(eax)。

131 __asm__("cld\n\t" // 清方向位。

132 "movb %%al, %%ah\n\t" // 将欲比较字符移到 ah。

133 "1:\t lodsb\n\t" // 取字符串中字符 ds:[esi]→al, 并且 esi++。

134 "cmpb %%ah, %%al\n\t" // 字符串中字符 al 与指定字符 ah 相比较。

135 "je 2f\n\t" // 若相等, 则向前跳转到标号 2 处。

136 "testb %%al, %%al\n\t" // al 中字符是 NULL 字符吗? (字符串结尾?)

137 "jne 1b\n\t" // 若不是, 则向后跳转到标号 1, 继续比较。

138 "movl \$1, %1\n\t" // 是, 则说明没有找到匹配字符, esi 置 1。

139 "2:\tmovl %1, %0\n\t" // 将指向匹配字符后一个字节处的指针值放入 eax

140 "decl %0" // 将指针调整为指向匹配的字符。

141 : "=a" (__res): "S" (s), "0" (c): "si");

142 return __res; // 返回指针。

143 }

144

```
//// 寻找字符串中指定字符最后一次出现的地方。(反向搜索字符串)
```

```
// 参数: s - 字符串, c - 欲寻找的字符。
```

```
// %0 - edx(__res), %1 - edx(0), %2 - esi(字符串指针 s), %3 - eax(字符 c)。
```

```
// 返回: 返回字符串中最后一次出现匹配字符的指针。若没有找到匹配的字符, 则返回空指针。
```

145 extern inline char * [strrchr](#)(const char * s, char c)

146 {

147 register char * __res __asm__("dx"); // __res 是寄存器变量(edx)。

148 __asm__("cld\n\t" // 清方向位。

149 "movb %%al, %%ah\n\t" // 将欲寻找的字符移到 ah。

150 "1:\t lodsb\n\t" // 取字符串中字符 ds:[esi]→al, 并且 esi++。

151 "cmpb %%ah, %%al\n\t" // 字符串中字符 al 与指定字符 ah 作比较。

152 "jne 2f\n\t" // 若不相等, 则向前跳转到标号 2 处。

153 "movl %%esi, %0\n\t" // 将字符指针保存到 edx 中。

154 "decl %0\n\t" // 指针后退一位, 指向字符串中匹配字符处。

155 "2:\ttestb %%al, %%al\n\t" // 比较的字符是 0 吗(到字符串尾)?

156 "jne 1b" // 不是则向后跳转到标号 1 处, 继续比较。

157 : "=d" (__res): "0" (0), "S" (s), "a" (c): "ax", "si");

158 return __res; // 返回指针。

159 }

160

```
//// 在字符串 1 中寻找第 1 个字符序列, 该字符序列中的任何字符都包含在字符串 2 中。
```

```
// 参数: cs - 字符串 1 指针, ct - 字符串 2 指针。
```

```
// %0 - esi(__res), %1 - eax(0), %2 - ecx(-1), %3 - esi(串 1 指针 cs), %4 - (串 2 指针 ct)。
```

```
// 返回字符串 1 中包含字符串 2 中任何字符的首个字符序列的长度值。
```

161 extern inline int [strspn](#)(const char * cs, const char * ct)

162 {

163 register char * __res __asm__("si"); // __res 是寄存器变量(esi)。

164 __asm__("cld\n\t" // 清方向位。

165 "movl %4, %%edi\n\t" // 首先计算串 2 的长度。串 2 指针放入 edi 中。

166 "repne\n\t" // 比较 al(0)与串 2 中的字符(es:[edi]), 并 edi++。

167 "scasb\n\t" // 如果不相等就继续比较(ecx 逐步递减)。

```

168     "notl %%ecx\n\t"           // ecx 中每位取反。
169     "decl %%ecx\n\t"           // ecx--, 得串 2 的长度值。
170     "movl %%ecx, %%edx\n\t"     // 将串 2 的长度值暂放入 edx 中。
171     "1:\t lodsb\n\t"           // 取串 1 字符 ds:[esi]→al, 并且 esi++。
172     "testb %al, %al\n\t"       // 该字符等于 0 值吗 (串 1 结尾)?
173     "je 2f\n\t"                // 如果是, 则向前跳转到标号 2 处。
174     "movl %4, %%edi\n\t"       // 取串 2 头指针放入 edi 中。
175     "movl %%edx, %%ecx\n\t"     // 再将串 2 的长度值放入 ecx 中。
176     "repne\n\t"                // 比较 al 与串 2 中字符 es:[edi], 并且 edi++。
177     "scasb\n\t"                // 如果不相等就继续比较。
178     "je 1b\n\t"                // 如果相等, 则向后跳转到标号 1 处。
179     "2:\t decl %0"             // esi--, 指向最后一个包含在串 2 中的字符。
180     : "=S" (__res): "a" (0), "c" (0xffffffff), "0" (cs), "g" (ct)
181     : "ax", "cx", "dx", "di");
182 return __res-cs;                // 返回字符序列的长度值。
183 }
184
185 // 寻找字符串 1 中不包含字符串 2 中任何字符的首个字符序列。
186 // 参数: cs - 字符串 1 指针, ct - 字符串 2 指针。
187 // %0 - esi(__res), %1 - eax(0), %2 - ecx(-1), %3 - esi(串 1 指针 cs), %4 - (串 2 指针 ct)。
188 // 返回字符串 1 中不包含字符串 2 中任何字符的首个字符序列的长度值。
189 extern inline int strcspn(const char * cs, const char * ct)
190 {
191     register char * __res __asm__("si"); // __res 是寄存器变量(esi)。
192     __asm__("cld\n\t"                // 清方向位。
193           "movl %4, %%edi\n\t"       // 首先计算串 2 的长度。串 2 指针放入 edi 中。
194           "repne\n\t"                // 比较 al(0)与串 2 中的字符(es:[edi]), 并 edi++。
195           "scasb\n\t"                // 如果不相等就继续比较(ecx 逐步递减)。
196           "notl %%ecx\n\t"           // ecx 中每位取反。
197           "decl %%ecx\n\t"           // ecx--, 得串 2 的长度值。
198           "movl %%ecx, %%edx\n\t"     // 将串 2 的长度值暂放入 edx 中。
199           "1:\t lodsb\n\t"           // 取串 1 字符 ds:[esi]→al, 并且 esi++。
200           "testb %al, %al\n\t"       // 该字符等于 0 值吗 (串 1 结尾)?
201           "je 2f\n\t"                // 如果是, 则向前跳转到标号 2 处。
202           "movl %4, %%edi\n\t"       // 取串 2 头指针放入 edi 中。
203           "movl %%edx, %%ecx\n\t"     // 再将串 2 的长度值放入 ecx 中。
204           "repne\n\t"                // 比较 al 与串 2 中字符 es:[edi], 并且 edi++。
205           "scasb\n\t"                // 如果不相等就继续比较。
206           "jne 1b\n\t"               // 如果不相等, 则向后跳转到标号 1 处。
207           "2:\t decl %0"             // esi--, 指向最后一个包含在串 2 中的字符。
208           : "=S" (__res): "a" (0), "c" (0xffffffff), "0" (cs), "g" (ct)
209           : "ax", "cx", "dx", "di");
210 return __res-cs;                // 返回字符序列的长度值。
211 }
212
213 // 在字符串 1 中寻找首个包含在字符串 2 中的任何字符。
214 // 参数: cs - 字符串 1 的指针, ct - 字符串 2 的指针。
215 // %0 -esi(__res), %1 -eax(0), %2 -ecx(0xffffffff), %3 -esi(串 1 指针 cs), %4 -(串 2 指针 ct)。
216 // 返回字符串 1 中首个包含字符串 2 中字符的指针。
217 extern inline char * strpbrk(const char * cs, const char * ct)
218 {
219     register char * __res __asm__("si"); // __res 是寄存器变量(esi)。
220     __asm__("cld\n\t"                // 清方向位。

```

```

213     "movl %4, %%edi|n|t" // 首先计算串 2 的长度。串 2 指针放入 edi 中。
214     "repne|n|t" // 比较 al(0)与串 2 中的字符(es:[edi]), 并 edi++。
215     "scasb|n|t" // 如果不相等就继续比较(ecx 逐步递减)。
216     "notl %%ecx|n|t" // ecx 中每位取反。
217     "decl %%ecx|n|t" // ecx--, 得串 2 的长度值。
218     "movl %%ecx, %%edx|n" // 将串 2 的长度值暂放入 edx 中。
219     "1:|t|lods|n|t" // 取串 1 字符 ds:[esi]→al, 并且 esi++。
220     "testb %%al, %%al|n|t" // 该字符等于 0 值吗(串 1 结尾)?
221     "je 2f|n|t" // 如果是, 则向前跳转到标号 2 处。
222     "movl %4, %%edi|n|t" // 取串 2 头指针放入 edi 中。
223     "movl %%edx, %%ecx|n|t" // 再将串 2 的长度值放入 ecx 中。
224     "repne|n|t" // 比较 al 与串 2 中字符 es:[edi], 并且 edi++。
225     "scasb|n|t" // 如果不相等就继续比较。
226     "jne 1b|n|t" // 如果不相等, 则向后跳转到标号 1 处。
227     "decl %0|n|t" // esi--, 指向一个包含在串 2 中的字符。
228     "jmp 3f|n" // 向前跳转到标号 3 处。
229     "2:|txorl %0, %0|n" // 没有找到符合条件的, 将返回值为 NULL。
230     "3:"
231     : "=S" (__res): "a" (0), "c" (0xffffffff), "0" (cs), "g" (ct)
232     : "ax", "cx", "dx", "di");
233 return __res; // 返回指针值。
234 }
235
//// 在字符串 1 中寻找首个匹配整个字符串 2 的字符串。
// 参数: cs - 字符串 1 的指针, ct - 字符串 2 的指针。
// %0 -eax(__res), %1 -eax(0), %2 -ecx(0xffffffff), %3 -esi(串 1 指针 cs), %4 -(串 2 指针 ct)。
// 返回: 返回字符串 1 中首个匹配字符串 2 的字符串指针。
236 extern inline char * strstr(const char * cs, const char * ct)
237 {
238 register char * __res __asm__("ax"); // __res 是寄存器变量(eax)。
239 __asm__("cld|n|t" \ // 清方向位。
240     "movl %4, %%edi|n|t" // 首先计算串 2 的长度。串 2 指针放入 edi 中。
241     "repne|n|t" // 比较 al(0)与串 2 中的字符(es:[edi]), 并 edi++。
242     "scasb|n|t" // 如果不相等就继续比较(ecx 逐步递减)。
243     "notl %%ecx|n|t" // ecx 中每位取反。
244     "decl %%ecx|n|t" /* NOTE! This also sets Z if searchstring='' */
// 注意! 如果搜索串为空, 将设置 Z 标志 */ // 得串 2 的长度值。
245     "movl %%ecx, %%edx|n" // 将串 2 的长度值暂放入 edx 中。
246     "1:|tmovl %4, %%edi|n|t" // 取串 2 头指针放入 edi 中。
247     "movl %%esi, %%eax|n|t" // 将串 1 的指针复制到 eax 中。
248     "movl %%edx, %%ecx|n|t" // 再将串 2 的长度值放入 ecx 中。
249     "repe|n|t" // 比较串 1 和串 2 字符(ds:[esi], es:[edi]), esi++, edi++。
250     "cmpsb|n|t" // 若对应字符相等就一直比较下去。
251     "je 2f|n|t" /* also works for empty string, see above */
// 对空串同样有效, 见上面 */ // 若全相等, 则转到标号 2。
252     "xchgl %%eax, %%esi|n|t" // 串 1 头指针→esi, 比较结果的串 1 指针→eax。
253     "incl %%esi|n|t" // 串 1 头指针指向下一个字符。
254     "cmpb $0, -1(%%eax)|n|t" // 串 1 指针(eax-1)所指字节是 0 吗?
255     "jne 1b|n|t" // 不是则转到标号 1, 继续从串 1 的第 2 个字符开始比较。
256     "xorl %%eax, %%eax|n|t" // 清 eax, 表示没有找到匹配。
257     "2:"
258     : "=a" (__res): "0" (0), "c" (0xffffffff), "S" (cs), "g" (ct)
259     : "cx", "dx", "di", "si");

```

```

260 return __res;          // 返回比较结果。
261 }
262
263 // 计算字符串长度。
264 // 参数: s - 字符串。
265 // %0 - ecx(__res), %1 - edi(字符串指针 s), %2 - eax(0), %3 - ecx(0xffffffff)。
266 // 返回: 返回字符串的长度。
267 extern inline int strlen(const char * s)
268 {
269     register int __res __asm__( "cx" );    // __res 是寄存器变量(ecx)。
270     __asm__( "cld\n\t"                    // 清方向位。
271             "repne\n\t"                  // al(0)与字符串中字符 es:[edi]比较,
272             "scasb\n\t"                  // 若不相等就一直比较。
273             "notl %0\n\t"                // ecx取反。
274             "decl %0"                     // ecx--, 得字符串得长度值。
275             : "=c" (__res): "D" (s), "a" (0), "0" (0xffffffff): "di" );
276     return __res;          // 返回字符串长度值。
277 }
278
279 extern char * strtok;    // 用于临时存放指向下面被分析字符串 1(s)的指针。
280
281 // 利用字符串 2 中的字符将字符串 1 分割成标记(token)序列。
282 // 将串 1 看作是包含零个或多个单词(token)的序列, 并由分割符字符串 2 中的一个或多个字符
283 // 分开。第一次调用 strtok()时, 将返回指向字符串 1 中第 1 个 token 首字符的指针, 并在返
284 // 回 token 时将一 null 字符写到分割符处。后续使用 null 作为字符串 1 的调用, 将用这种方
285 // 法继续扫描字符串 1, 直到没有 token 为止。在不同的调用过程中, 分割符串 2 可以不同。
286 // 参数: s - 待处理的字符串 1, ct - 包含各个分割符的字符串 2。
287 // 汇编输出: %0 - ebx(__res), %1 - esi(__strtok);
288 // 汇编输入: %2 - ebx(__strtok), %3 - esi(字符串 1 指针 s), %4 - (字符串 2 指针 ct)。
289 // 返回: 返回字符串 s 中第 1 个 token, 如果没有找到 token, 则返回一个 null 指针。
290 // 后续使用字符串 s 指针为 null 的调用, 将在原字符串 s 中搜索下一个 token。
291 extern inline char * strtok(char * s, const char * ct)
292 {
293     register char * __res __asm__( "si" );
294     __asm__( "testl %1, %1\n\t"           // 首先测试 esi(字符串 1 指针 s)是否是 NULL。
295             "jne 1f\n\t"                 // 如果不是, 则表明是首次调用本函数, 跳转标号 1。
296             "testl %0, %0\n\t"           // 若是 NULL, 表示此次是后续调用, 测 ebx(__strtok)。
297             "je 8f\n\t"                  // 如果 ebx 指针是 NULL, 则不能处理, 跳转结束。
298             "movl %0, %1\n\t"            // 将 ebx 指针复制到 esi。
299             "1:|txorl %0, %0\n\t"        // 清 ebx 指针。
300             "movl $-1, %%ecx\n\t"        // 置 ecx = 0xffffffff。
301             "xorl %%eax, %%eax\n\t"      // 清零 eax。
302             "cld\n\t"                    // 清方向位。
303             "movl %4, %%edi\n\t"         // 下面求字符串 2 的长度。edi 指向字符串 2。
304             "repne\n\t"                  // 将 al(0)与 es:[edi]比较, 并且 edi++。
305             "scasb\n\t"                  // 直到找到字符串 2 的结束 null 字符, 或计数 ecx==0。
306             "notl %%ecx\n\t"             // 将 ecx 取反,
307             "decl %%ecx\n\t"             // ecx--, 得到字符串 2 的长度值。
308             "je 7f\n\t"                  /* empty delimiter-string */
309             /* 分割符字符串空 */ // 若串 2 长度为 0, 则转标号 7。
310             "movl %%ecx, %%edx\n\t"      // 将串 2 长度暂存入 edx。
311             "2:|tlodsb\n\t"              // 取串 1 的字符 ds:[esi]→al, 并且 esi++。
312             "testb %%al, %%al\n\t"       // 该字符为 0 值吗(串 1 结束)?

```

```

298     "je 7f|n|t" // 如果是，则跳转标号 7。
299     "movl %4, %%edi|n|t" // edi 再次指向串 2 首。
300     "movl %%edx, %%ecx|n|t" // 取串 2 的长度值置入计数器 ecx。
301     "repne|n|t" // 将 a1 中串 1 的字符与串 2 中所有字符比较，
302     "scasb|n|t" // 判断该字符是否为分割符。
303     "je 2b|n|t" // 若能在串 2 中找到相同字符（分割符），则跳转标号 2。
304     "decl %1|n|t" // 若不是分割符，则串 1 指针 esi 指向此时的该字符。
305     "cmpb $0, (%1)|n|t" // 该字符是 NULL 字符吗？
306     "je 7f|n|t" // 若是，则跳转标号 7 处。
307     "movl %1, %0|n" // 将该字符的指针 esi 存放在 ebx。
308     "3:|t|odsb|n|t" // 取串 1 下一个字符 ds:[esi]→a1，并且 esi++。
309     "testb %%a1, %%a1|n|t" // 该字符是 NULL 字符吗？
310     "je 5f|n|t" // 若是，表示串 1 结束，跳转到标号 5。
311     "movl %4, %%edi|n|t" // edi 再次指向串 2 首。
312     "movl %%edx, %%ecx|n|t" // 串 2 长度值置入计数器 ecx。
313     "repne|n|t" // 将 a1 中串 1 的字符与串 2 中每个字符比较，
314     "scasb|n|t" // 测试 a1 字符是否是分割符。
315     "jne 3b|n|t" // 若不是分割符则跳转标号 3，检测串 1 中下一个字符。
316     "decl %1|n|t" // 若是分割符，则 esi--，指向该分割符字符。
317     "cmpb $0, (%1)|n|t" // 该分割符是 NULL 字符吗？
318     "je 5f|n|t" // 若是，则跳转到标号 5。
319     "movb $0, (%1)|n|t" // 若不是，则将该分割符用 NULL 字符替换掉。
320     "incl %1|n|t" // esi 指向串 1 中下一个字符，也即剩余串首。
321     "jmp 6f|n" // 跳转标号 6 处。
322     "5:|t|xorl %1, %1|n" // esi 清零。
323     "6:|t|cmpb $0, (%0)|n|t" // ebx 指针指向 NULL 字符吗？
324     "jne 7f|n|t" // 若不是，则跳转标号 7。
325     "xorl %0, %0|n" // 若是，则让 ebx=NULL。
326     "7:|t|testl %0, %0|n|t" // ebx 指针为 NULL 吗？
327     "jne 8f|n|t" // 若不是则跳转 8，结束汇编代码。
328     "movl %0, %1|n" // 将 esi 置为 NULL。
329     "8:"
330     : "=b" (__res), "=S" (__strtok)
331     : "0" (__strtok), "1" (s), "g" (ct)
332     : "ax", "cx", "dx", "di");
333 return __res; // 返回指向新 token 的指针。
334 }
335
336     //// 内存块复制。从源地址 src 处开始复制 n 个字节到目的地址 dest 处。
337     // 参数：dest - 复制的目的地址，src - 复制的源地址，n - 复制字节数。
338     // %0 - ecx(n)，%1 - esi(src)，%2 - edi(dest)。
339 extern inline void * memcpy(void * dest, const void * src, int n)
340 {
341     __asm__ ("cld|n|t" // 清方向位。
342             "rep|n|t" // 重复执行复制 ecx 个字节，
343             "movsb" // 从 ds:[esi]到 es:[edi]，esi++，edi++。
344             :: "c" (n), "S" (src), "D" (dest)
345             : "cx", "si", "di");
346 return dest; // 返回目的地址。
347 }
348
349     //// 内存块移动。同内存块复制，但考虑移动的方向。
350     // 参数：dest - 复制的目的地址，src - 复制的源地址，n - 复制字节数。

```



```

// 若 dest<src 则: %0 - ecx(n), %1 - esi(src), %2 - edi(dest)。
// 否则: %0 - ecx(n), %1 - esi(src+n-1), %2 - edi(dest+n-1)。
// 这样操作是为了防止在复制时错误地重叠覆盖。
346 extern inline void * memmove(void * dest, const void * src, int n)
347 {
348     if (dest<src)
349         __asm__ ("cld\n\t"                // 清方向位。
350                 "rep\n\t"                // 从 ds:[esi]到 es:[edi], 并且 esi++, edi++,
351                 "movsb"                  // 重复执行复制 ecx 字节。
352                 :: "c" (n), "S" (src), "D" (dest)
353                 : "cx", "si", "di");
354     else
355         __asm__ ("std\n\t"                // 置方向位, 从末端开始复制。
356                 "rep\n\t"                // 从 ds:[esi]到 es:[edi], 并且 esi--, edi--,
357                 "movsb"                  // 复制 ecx 个字节。
358                 :: "c" (n), "S" (src+n-1), "D" (dest+n-1)
359                 : "cx", "si", "di");
360     return dest;
361 }
362
//// 比较 n 个字节的内存块 (两个字符串), 即使遇上 NULL 字节也不停止比较。
// 参数: cs - 内存块 1 地址, ct - 内存块 2 地址, count - 比较的字节数。
// %0 - eax(__res), %1 - eax(0), %2 - edi(内存块 1), %3 - esi(内存块 2), %4 - ecx(count)。
// 返回: 若块 1>块 2 返回 1; 块 1<块 2, 返回-1; 块 1==块 2, 则返回 0。
363 extern inline int memcmp(const void * cs, const void * ct, int count)
364 {
365     register int __res __asm__ ("ax"); // __res 是寄存器变量。
366     __asm__ ("cld\n\t"                // 清方向位。
367             "repe\n\t"                // 如果相等则重复,
368             "cmpsb\n\t"               // 比较 ds:[esi]与 es:[edi]的内容, 并且 esi++, edi++。
369             "je 1f\n\t"                // 如果都相同, 则跳转到标号 1, 返回 0(eax)值
370             "movl $1, %%eax\n\t"      // 否则 eax 置 1,
371             "jl 1f\n\t"                // 若内存块 2 内容的值<内存块 1, 则跳转标号 1。
372             "negl %%eax\n\t"          // 否则 eax = -eax。
373             "1:"
374             : "=a" (__res): "0" (0), "D" (cs), "S" (ct), "c" (count)
375             : "si", "di", "cx");
376     return __res; // 返回比较结果。
377 }
378
//// 在 n 字节大小的内存块 (字符串) 中寻找指定字符。
// 参数: cs - 指定内存块地址, c - 指定的字符, count - 内存块长度。
// %0 - edi(__res), %1 - eax(字符 c), %2 - edi(内存块地址 cs), %3 - ecx(字节数 count)。
// 返回第一个匹配字符的指针, 如果没有找到, 则返回 NULL 字符。
379 extern inline void * memchr(const void * cs, char c, int count)
380 {
381     register void * __res __asm__ ("di"); // __res 是寄存器变量。
382     if (!count) // 如果内存块长度==0, 则返回 NULL, 没有找到。
383         return NULL;
384     __asm__ ("cld\n\t"                // 清方向位。
385             "repne\n\t"               // 如果不相等则重复执行下面语句,
386             "scasb\n\t"               // a1 中字符与 es:[edi]字符作比较, 并且 edi++,
387             "je 1f\n\t"                // 如果相等则向前跳转到标号 1 处。

```

```
388     "movl $1,%0\n"           // 否则 edi 中置 1。
389     "1:\tdecl %0"           // 让 edi 指向找到的字符 (或是 NULL)。
390     :"=D" (__res): "a" (c), "D" (cs), "c" (count)
391     :"cx");
392 return __res;                // 返回字符指针。
393 }
394
395     //// 用字符填写指定长度内存块。
396     // 用字符 c 填写 s 指向的内存区域, 共填 count 字节。
397     // %0 - eax(字符 c), %1 - edi(内存地址), %2 - ecx(字节数 count)。
398 extern inline void * memset(void * s, char c, int count)
399 {
400     __asm__ ("cld\n\t"           // 清方向位。
401             "rep\n\t"           // 重复 ecx 指定的次数, 执行
402             "stosb"           // 将 al 中字符存入 es:[edi]中, 并且 edi++。
403             :"a" (c), "D" (s), "c" (count)
404             :"cx", "di");
405 return s;
406 }
```
