

Re: Note to Chuck Crayne

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2005-11/msg00014.html>

- *From:* "a\b" <al@xxx>
 - *Date:* Wed, 02 Nov 2005 08:27:45 +0100
-

On Tue, 01 Nov 2005 14:19:16 -0500, Frank Kotler <fbkotler@xxxxxxxxxxxx> wrote:
>There must be "even worse" in common usage. As you know, I like "open
>sauce" software, so if I felt like plowing through the C source, I could
>find a real gem for us to optimize. When I click on the little
>hieroglyph to "sort by thread" instead of "sort by date", it takes – by
>wall clock – approximately a minute and fifteen seconds to complete.
>What kind of a sort algo does *that*???

>
>"Optimizing strlen" is not going to solve this type of problem, but
>getting rid of the "it doesn't matter" attitude might...

>
>Best,
>Frank

>
>P.S. I saved your posts on "qsort" – haven't played with 'em yet...

don't know how many errors there are but `qsort_m()` should order general arrays if it is known the order function. The idea is build an array of pointers and order that instead of ordering the original array.

```
; b=cmp, _qsortr1_m(U left, U right)
; 0Ra, 4@left, 8@right
_qsortr1_m:
%define @left esp+4
%define @right esp+8
mov esi, [@left]
mov edi, [@right]
cmp esi, edi
jae .ck
lea eax, [esi+edi]
lea ebp, [esi+4]
shr eax, 1
mov ecx, esi
and eax, 0xFFFFF0
and ecx, 0x3
or eax, ecx
; array=..4, ..8, ..12 ecc
```

Re: Note to Chuck Crayne

```
; array=..1, ..5, ..9
; array=..2, ..6, ..10
mov edx, [esi]
mov ecx, [eax]
mov [eax], edx
mov [esi], ecx
push ecx
jmp short .c2
..ck:
jmp short .cf
..c0:
push dword[ebp]
call ebx
add esp, 4
cmp eax, 0
jge .c1
add esi, 4
mov edx, [esi]
mov ecx, [ebp]
mov [esi], ecx
mov [ebp], edx
..c1:
add ebp, 4
..c2:
cmp ebp, edi
jbe .c0
pop eax
mov edx, [@left]
mov ecx, [esi]
mov [edx], ecx
mov [esi], eax
; r=@left, j=@right, i=last
lea eax, [esi-4]
push eax
push edx
call _qsortr1_m
add esp, 4
pop eax
mov edi, [@right]
add eax, 8
push edi
push eax
call _qsortr1_m
add esp, 8
..cf:
%undef @left
%undef @right
ret

; b=cmp, _qsortr_m(int left, int right)
; ORa, 4@left, 8@right
```

Re: Note to Chuck Crayne

```
_qsortr_m:
%define @left esp+4
%define @right esp+8
mov esi, [@left]
mov edi, [@right]
mov ebp, edi
sub ebp, esi ;/*
jg .hh
jmp .cf
..hh:
push dword[edi]
push dword[esi]
call ebx
add esp, 8
cmp eax, 0
jle .h0
mov edx, [esi]
mov ecx, [edi]
mov [esi], ecx
mov [edi], edx
..h0:
cmp ebp, 4
je .ck
push ebp
lea eax, [esi+edi] ; array=..4, ..8, ..12 ecc
mov ecx, esi
shr eax, 1
and ecx, 0x3
and eax, 0xFFFFF0 ; array=..1, ..5, ..9
or eax, ecx
mov ebp, eax
push dword[ebp]
push dword[esi]
call ebx
add esp, 8 ; array=..2, ..6, ..10
cmp eax, 0
jle .h1
mov edx, [esi] ;per allineamento a primo elemento
mov ecx, [ebp]
mov [esi], ecx
mov [ebp], edx
..h1:
push dword[edi]
push dword[ebp]
call ebx
add esp, 8
cmp eax, 0
jle .h2
mov edx, [ebp]
mov ecx, [edi]
mov [ebp], ecx
```

```
mov [edi], edx
..h2:
pop eax
cmp eax, 8
je .ck
sub edi, 4
mov edx, [ebp]
mov ecx, [edi]
mov [ebp], ecx
mov [edi], edx
mov ebp, edi
push edx
jmp short .c1
..ck:
jmp short .cf
..c0:
mov edx, [esi]
mov ecx, [edi]
mov [esi], ecx
mov [edi], edx
..c1:
add esi, 4
push dword[esi]
call ebx
add esp, 4
cmp eax, 0
jl .c1
..c2:
sub edi, 4
push dword[edi]
call ebx
add esp, 4
cmp eax, 0
jg .c2
cmp edi, esi
jg .c0 ;/*
pop eax
mov ecx, [esi]
mov [ebp], ecx
mov [esi], eax
; r=@left, j=@right, i=last
lea eax, [esi-4]
mov edx, [@left]
push eax
push edx
call _qsortr_m
add esp, 4
pop eax
mov edi, [@right]
add eax, 8
push edi
```

Re: Note to Chuck Crayne

```
push eax
call _qsortr_m
add esp, 8
..cf:
%undef @left
%undef @right
ret
```

```
; int _qsorti_m<(int* ris, int n, int (*cmp)(int , int))
; serve per ordinare array di ogg di dimensione 4char
; per tali oggetti utilizzare questa funzione e non qsort_m
; s= 0k, 4j, 8i, 12b, 16Ra, 20@ris, 24@n, 28@cmp
```

```
_qsorti_m:
push ebx
push esi
push edi
push ebp
%define @ris esp+20
%define @n esp+24
%define @cmp esp+28
mov eax, [@ris]
mov edx, [@n]
mov ebx, [@cmp]
cmp edx, 2
jl .c1
cmp eax, 0
je .ce
cmp ebx, 0
je .ce
dec edx
lea esi, [eax+4*edx]
push esi
push eax
call _qsortr_m
add esp, 8
..c1:
mov eax, 1
jmp short .cf
..ce:
xor eax, eax
..cf:
%undef @ris
%undef @n
%undef @cmp
pop ebp
pop edi
pop esi
pop ebx
ret
```

Re: Note to Chuck Crayne

```
; int _qsortu_m<(unsigned* ris, int n, int (*cmp)(U , U))  
; serve per ordinare array di ogg di dimensione 4char  
; per tali oggetti utilizzare questa funzione e non qsort_m  
; s= 0k, 4j, 8i, 12b, 16Ra, 20@ris, 24@n, 28@cmp
```

```
_qsortu_m:  
push ebx  
push esi  
push edi  
push ebp  
%define @ris esp+20  
%define @n esp+24  
%define @cmp esp+28  
mov eax, [@ris]  
mov edx, [@n]  
mov ebx, [@cmp]  
cmp edx, 2  
jl .c1  
cmp eax, 0  
je .ce  
cmp ebx, 0  
je .ce  
dec edx  
lea esi, [eax+4*edx]  
push esi  
push eax  
call _qsortr1_m  
add esp, 8  
..c1:  
mov eax, 1  
jmp short .cf  
..ce:  
xor eax, eax  
..cf:  
%undef @ris  
%undef @n  
%undef @cmp  
pop ebp  
pop edi  
pop esi  
pop ebx  
ret
```

```
; int _qsort_m<(char* res, char** ppres, char* orig,  
; int n, int size, int (*cmp)(char*, char*) )  
; it order the "array of objects" pointed by 'orig' pointer  
; of dimension 'size' and with number 'n'; it for compare use  
; the function 'cmp'  
; 'res' has to be a pointer to reserved memory to an array
```

Re: Note to Chuck Crayne

```
; of size and number of object at last as 'orig'  
; in 'res' at end of computation there is the ordered array.  
; 'ppres' is an array of pointers (dword) that has the same  
; numbers of elements at last as 'orig' and at the end of  
; computation these pointer have the array ordered  
; the idea is order the pointers and so the objects  
; s = 0k, 4j, 8i, 12b, 16Ra,  
; 20@res, 24@ppres, 28@orig, 32@n, 36@size, 40@cmp
```

```
_qsort_m:  
push ebx  
push esi  
push edi  
push ebp  
%define @res esp+20  
%define @ppres esp+24  
%define @orig esp+28  
%define @n esp+32  
%define @size esp+36  
%define @cmp esp+40  
mov ecx, [@size]  
mov edi, [@ppres]  
mov eax, [@n]  
mov edx, [@orig]  
cmp ecx, 0  
jle .ce  
cmp eax, 0  
jle .ce  
cmp edi, 0  
je .ce  
cmp edx, 0  
je .ce  
cmp dword[@cmp], 0  
je .ce  
jmp short .c1  
..ce:  
xor eax, eax  
jmp .cf  
..c1:  
mov [edi], edx  
add edi, 4  
add edx, ecx  
dec eax  
jnz .c1  
mov eax, [@ppres]  
mov edx, [@n]  
mov ebx, [@cmp]  
dec edx  
lea esi, [eax+4*edx]  
push esi  
push eax  
call _qsortr_m
```

```
add esp, 8
mov esi, [@ppres]
mov edi, [@res]
mov ecx, [@n]
mov ebp, [@size]
test ebp, 0x1
jnz .u1
test ebp, 0x2
jnz .u2
shr ebp, 2
..c2:
mov eax, [esi]
mov ebx, ebp
..c3:
mov edx, [eax]
mov [edi], edx
add eax, 4
add edi, 4
dec ebx
jnz .c3
add esi, 4
dec ecx
jnz .c2

jmp short .cz
..u2:
shr ebp, 1
..c4:
mov eax, [esi]
mov ebx, ebp
..c5:
mov dx, [eax]
mov [edi], dx
add eax, 2
add edi, 2
dec ebx
jnz .c5
add esi, 4
dec ecx
jnz .c4

jmp short .cz
..u1:
mov eax, [esi]
mov ebx, ebp
..c6:
mov dl, [eax]
mov [edi], dl
inc eax
inc edi
dec ebx
```

```
jnz .c6
add esi, 4
dec ecx
jnz .u1

..cz:
mov eax, 1
..cf:
%undef @res
%undef @ppres
%undef @orig
%undef @n
%undef @size
%undef @cmp
pop ebp
pop edi
pop esi
pop ebx
ret

////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define R return
#define F for
#define P printf
#define U unsigned
#define W while

void qsortu_m(char* b, int n, int (*cmp)(U, U));
int qsort_m(char* res, char* ppres, char* orig,
int n, int size, int (*cmp)(char*, char* ) );

void
qsortu_c(char* b, int l, int r, int (*cmp)(U, U))
{ int i, last;
  U *v, tmp;
  //////////////////////////////////
  if(l>=r) R;
  v=( U *) b;
  tmp=v[l]; i=(l+r)/2; v[l]=v[i]; v[i]=tmp;
  last=l;
  F( i=l+1; i<=r; ++i )
  if((*cmp)(v[i], v[l]) < 0)
  { ++last; tmp=v[last]; v[last]=v[i]; v[i]=tmp; }
  tmp=v[l]; v[l]=v[last]; v[last]=tmp;
  qsortu_c(b, l, last-1, cmp);
  qsortu_c(b, last+1, r, cmp);
}
```

```
void print_array(U* b, int len)
{int i;
//////////
if(len<=0||b==0) R;
F(i=0; i<len; ++i)
P("%u ", b[i]);
P("\n");
}
```

```
void print_arrayc(char* b, int len)
{int i;
//////////
if(len<=0||b==0) R;
F(i=0; i<len; ++i)
P("%u ", ((U)b[i]&0xff));
P("\n");
}
```

```
void random_array(U* b, int len)
{static int led=0;
int i;
//////////
if(len<=0||b==0) R;
if(led==0){ led=1; srand((U)time(0));}
F(i=0; i<len; ++i) b[i]=rand();
}
```

```
void random_array1(U* b, int len)
{static int led=0;
int i;
//////////
if(len<=0||b==0) R;
if(led==0){ led=1; srand((U)time(0));}
F(i=0; i<len; ++i)
{b[i]=rand();
b[i]<=16;
b[i]=rand();
}
}
```

```
int cmpu(U a, U b)
{ if(a<b) R -1;
else if(a>b) R 1;
R 0;
}
```

```
int cmpx(char* aa, char* bb)
{U *a, *b;
```

```
a=aa; b=bb;
if(*a<*b) R -1;
else if(*a>*b) R 1;
R 0;
}
```

```
int cmpxc(char* aa, char* bb)
{unsigned char *a, *b;
a=aa; b=bb;
if(*a<*b) R -1;
else if(*a>*b) R 1;
R 0;
}
```

```
int main(void)
{unsigned a[800], b[800], *c[800], i, h, j;
char ac[50], bc[50], *cc[50];
time_t ti, tf;
////////////////////////////////////

/*
random_array(a, 500); print_array(a, 51);
//qsortu_c(a, 0, 10, cmpu);
P("a=%p b=%p c=%p\n", (void*)a, (void*)b, (void*)c);
qsortu_m(a, 50, cmpu);
print_array(a, 51);
*/

random_array(a, 50); print_array(a, 51);
qsort_m( b, &c, a, 50, 4, cmpx );
print_array(b, 51);

/*
random_array(a, 50); print_array(a, 51);
qsort( &a, 50, 4, (int (*)(const void*, const void*))cmpx );
print_array(b, 51);
*/

/*
F(j=0; j<10000; ++j)
{h=rand()%500; ++h;
random_array(a, h);
F(i=0; i<h; ++i) b[i]=a[i];
qsortu_c(a, 0, h-1, cmpu);
qsortu_m(b, h, cmpu);
F(i=0; i<h; ++i)
if(b[i]!=a[i]) {P("ERRORE\n"); goto fine;}
}
```

```
fine;;
P("j==%u\n", j);
*/

/*
F(j=0; j<10000; ++j)
{h=rand()%500; ++h;
random_array(a, h);
qsort_m( b, &c, a, h, 4, cmpx );
qsortu_c( a, 0, h-1, cmpu);
F(i=0; i<h; ++i)
if(b[i]!=a[i])
{P("ERRORE h=%u\na=", (U)h); print_array(a, h);
P("b="); print_array(b, h);
goto fine;
}
}
}
fine;;
P("j==%u\n", j);
*/

F(j=0; j<10000; ++j)
{h=rand()%49; ++h;
random_array1(ac, h/4);
qsort_m( bc, &cc, ac, h, 1, cmpxc );
qsort( &ac, h, 1, (int (*)(const void*, const void*))cmpxc );

F(i=0; i<h; ++i)
if(bc[i]!=ac[i])
{P("ERRORE h=%u\na=", (U)h); print_arrayc(ac, h+1);
P("b="); print_arrayc(bc, h+1);
goto fine;
}
}
}
fine;;
P("j==%u\nac=", j);print_arrayc(ac, h);
P("bc="); print_arrayc(bc, h);
random_array1(ac, 50/4);
P("ac_random="); print_arrayc(ac, 50);

R 0;
}

.
```

• **References:**

- ◆ **Note to Chuck Crayne**
- ◇ From: hutch---

Re: Note to Chuck Crayne

- ◆ **Re: Note to Chuck Crayne**
 - ◇ From: Frank Kotler
- ◆ **Re: Note to Chuck Crayne**
 - ◇ From: hutch--
- ◆ **Re: Note to Chuck Crayne**
 - ◇ From: Frank Kotler

- Prev by Date: **SciTE for HLA?**
- Next by Date: **Re: Note to Chuck Crayne**
- Previous by thread: **Re: One more "strlen" – was: Note to Chuck Crayne**
- Next by thread: **Re: Note to Chuck Crayne**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**