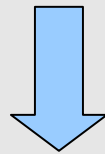


dco is a compiler post-processor
asm->asm

```
cc -c -O3 foo.c
```

```
.c.o:  
$(CC) $(CFLAGS) -c $<
```



```
cc -S -O3 foo.c  
dco foo.s -o ofoo.s  
mv ofoo.s foo.s  
cc -c foo.s  
rm foo.s
```

```
.c.o:  
$(CC) $(CFLAGS) -S $<  
dco $*.s -o $*.s $(DOPTS)  
$(CC) $(CFLAGS) -c $*.s  
rm $*.s
```

<http://www.dalsoft.com>



x86

AlphaPowered™

SHARC

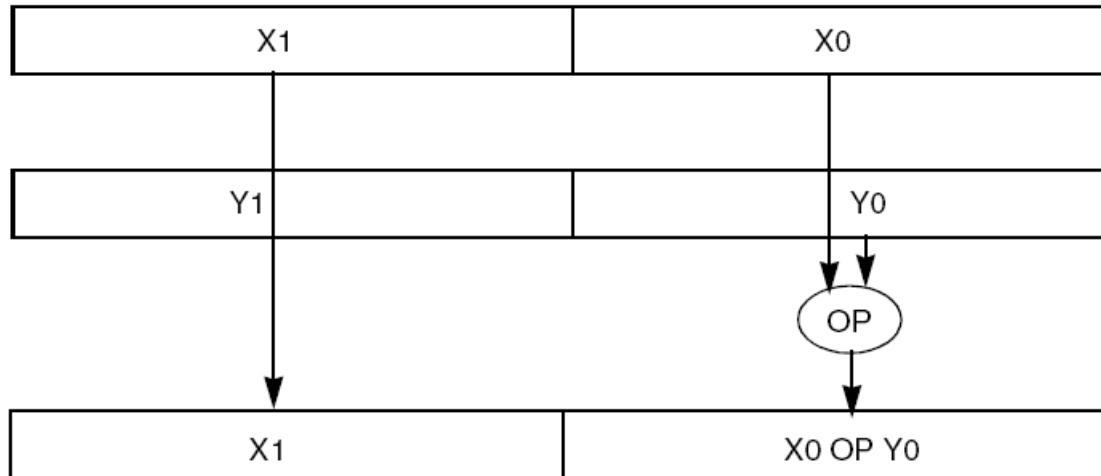


STAR CORE
BRIGHTER DSP TECHNOLOGY!

- allows selective optimization
- optimizes optimized code
- takes full advantage of the options and features provided by the target processor

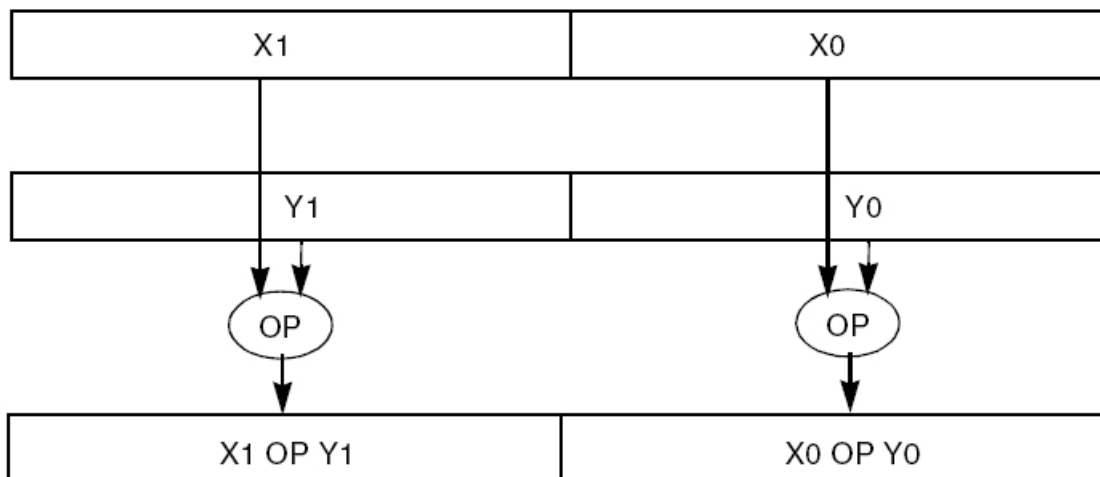
x86 offers the 'Single Instruction Multiple Data' (SIMD) instructions. The part of dco utilizing these instructions is called

SIMDinator (pronounced *seem-d-ney-ter*)



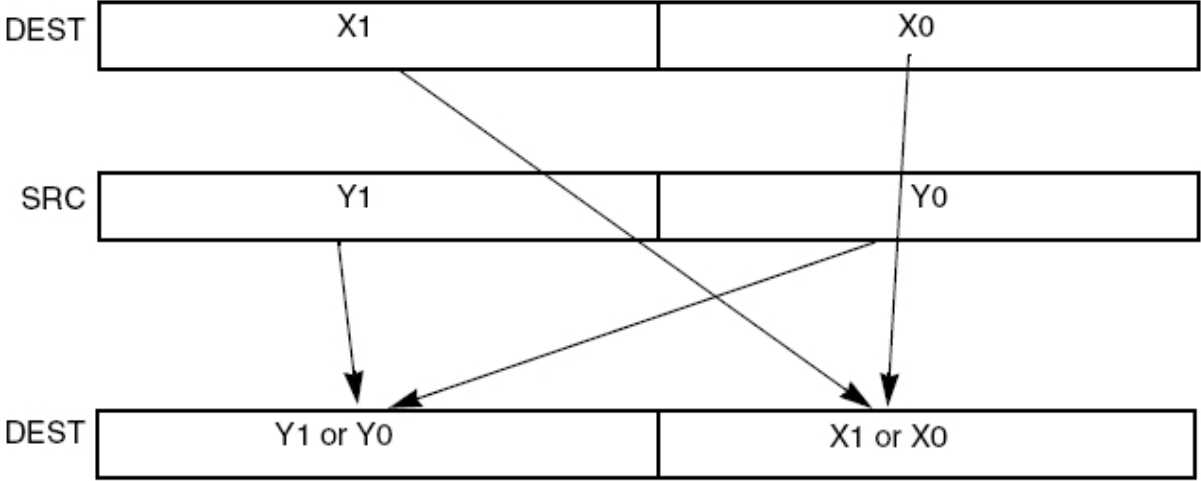
Scalar Double-Precision Floating-Point Operations

addsd
mulsd



Packed Double-Precision Floating-Point Operations

addpd
mulpd



SHUFPD Instruction Packed Shuffle Operation


shufpd
unpcklpd

http://www.dalsoft.com

x86 Optimizer - Mozilla Firefox


File Edit View History Bookmarks Tools Help

http://www.dalsoft.com/ Google



the tools to do the job

x86 optimizer



Home Benchmarks Documentation FAQ Contact Us

- general
- users manual
- SIMDinator
- Code Samples
- Compilers Comparison

This site is devoted to the x86 assembly code optimizer (**dco**) developed by DaLSoft for other architectures.

The x86 assembly code optimizer (**dco**) is a code for the IA-32 architecture which include SSE and SSE2 extensions.

The **dco** is used to optimize code generated by a compiler. The programmer uses a compiler (C/C++, FORTRAN etc.) to translate his code into x86 assembly code. This code is used as an input to **dco**. The output, generated by **dco**, is a highly optimized x86 assembly code that is logically identical to the original; **dco** rearranges the existing code, utilizing options and features that are supported by the x86 architecture. To create a final object file the generated code should be assembled.

Note that **dco** does not require preprocessing or any other involvement from the user. It is fully automated and it is possible to incorporate **dco** into makefiles or other product generation tools.

[About Us](#)

Copyright. ©2005-2007 DaLSoft All rights reserved. [Privacy Policy, Legal Notice](#)

http://www.dalsoft.com/documentation_simdinator.html

AdBlock

SIMDinator makes code faster

Kernel#	gcc 4.1.2	gcc+dco	-np	gcc+dco/gcc	-np/gcc	-np/gcc+dco
1	4.96	3.32	4	33.06%	19.35%	-20.48%
2	2.38	2.32	2.32	2.52%	2.52%	0.00%
3	5.93	3.55	3.84	40.13%	35.24%	-8.17%
4	4.66	4.12	3.8	11.59%	18.45%	7.77%
5	5.2	2.07	2.07	60.19%	60.19%	0.00%
6	4.53	3.9	3.63	13.91%	19.87%	6.92%
7	4.87	3.12	3.96	35.93%	18.69%	-26.92%
8	5	5.4	3.88	-8.00%	22.40%	28.15%
9	4.6	3.95	4.23	14.13%	8.04%	-7.09%
10	4.94	3.87	3.38	21.66%	31.58%	12.66%
11	5.78	1.52	1.52	73.70%	73.70%	0.00%
12	5.18	4.98	4.39	3.86%	15.25%	11.85%
13	4.57	5.56	4.58	-21.66%	-0.22%	17.63%
14	4.71	4.71	4.26	0.00%	9.55%	9.55%
15	3.72	3.72	3.72	0.00%	0.00%	0.00%
16	5.61	5.31	5.29	5.35%	5.70%	0.38%
17	5.01	4.99	4.99	0.40%	0.40%	0.00%
18	4.7	3.96	3.95	15.74%	15.96%	0.25%
19	5.81	4.1	4.1	29.43%	29.43%	0.00%
20	4.53	4.43	4.43	2.21%	2.21%	0.00%
21	4.88	4.61	4.61	5.53%	5.53%	0.00%
22	4.88	6.21	4.86	-27.25%	0.41%	21.74%
23	4.17	4.09	4.09	1.92%	1.92%	0.00%
24	4.85	0.77	0.77	84.12%	84.12%	0.00%
Geometric Mean	4.75	3.64	3.53	23.37%	25.68%	3.02%

```
for ( k=0 ; k<n ; k++ )
{
    x[k] = u[k] + r*( z[k] + r*y[k] ) +
          t*( u[k+3] + r*( u[k+2] + r*u[k+1] ) +
              t*( u[k+6] + q*( u[k+5] + q*u[k+4] ) ) );
}
```

36%
faster

dco generated code

```
.L1012:
    leal as1+32032(,%edx,8),%eax
    unpkplpd %xmm3,%xmm3
    unpkplpd %xmm4,%xmm4
    unpkplpd %xmm5,%xmm5
    ___dcox86_wl_0_ :
    movsd -31992(%eax),%xmm2
    addl $2,%edx
    movhpd -32000(%eax),%xmm2
    addl $16,%eax
    cmpl %ecx,%edx
    movsd -32000(%eax),%xmm7
    movhpd -32008(%eax),%xmm7
    movsd -32032(%eax),%xmm6
    movhpd -32040(%eax),%xmm6
    movsd -8(%eax),%xmm1
    movhpd -16(%eax),%xmm1
    mulpd %xmm4,%xmm2
    movsd 8000(%eax),%xmm0
    movhpd 7992(%eax),%xmm0
    addpd %xmm7,%xmm2
    movsd -32024(%eax),%xmm7
    movhpd -32032(%eax),%xmm7
    mulpd %xmm3,%xmm6
    mulpd %xmm4,%xmm2
    mulpd %xmm3,%xmm1
    addpd %xmm7,%xmm6
    movsd -31992(%eax),%xmm7
    movhpd -32000(%eax),%xmm7
    addpd %xmm0,%xmm1
    movsd -32040(%eax),%xmm0
    movhpd -32048(%eax),%xmm0
    mulpd %xmm3,%xmm6
    mulpd %xmm3,%xmm1
    addpd %xmm7,%xmm2
    movsd -32016(%eax),%xmm7
    movhpd -32024(%eax),%xmm7
    addpd %xmm0,%xmm1
    mulpd %xmm5,%xmm2
    addpd %xmm7,%xmm6
    addpd %xmm2,%xmm6
    mulpd %xmm5,%xmm6
    addpd %xmm6,%xmm1
    movhpd %xmm1,-8024(%eax)
    movsd %xmm1,-8016(%eax)
    jne ___dcox86_wl_0_
```

compiler generated code

```
.L1012:
    leal 1(%edx), %esi
    movapd %xmm3, %xmm6
    mulsd as1+32032(,%edx,8), %xmm6
    addsd as1+40040(,%edx,8), %xmm6
    mulsd %xmm3, %xmm6
    addsd as1(,%edx,8), %xmm6
    movapd %xmm3, %xmm1
    mulsd as1(,%esi,8), %xmm1
    addsd as1+16(,%edx,8), %xmm1
    mulsd %xmm3, %xmm1
    addsd as1+24(,%edx,8), %xmm1
    movapd %xmm4, %xmm0
    mulsd as1+24(,%esi,8), %xmm0
    addsd as1+32(,%esi,8), %xmm0
    mulsd %xmm4, %xmm0
    addsd as1+40(,%esi,8), %xmm0
    mulsd %xmm5, %xmm0
    addsd %xmm0, %xmm1
    mulsd %xmm5, %xmm1
    addsd %xmm1, %xmm6
    movsd %xmm6, as1+24016(,%esi,8)
    addl $2, %edx
    movapd %xmm3, %xmm6
    mulsd as1+32032(,%esi,8), %xmm6
    addsd as1+40040(,%esi,8), %xmm6
    mulsd %xmm3, %xmm6
    addsd as1(,%esi,8), %xmm6
    movapd %xmm3, %xmm7
    mulsd as1(,%edx,8), %xmm7
    addsd as1+16(,%esi,8), %xmm7
    mulsd %xmm3, %xmm7
    addsd as1+24(,%esi,8), %xmm7
    movapd %xmm4, %xmm2
    mulsd as1+24(,%edx,8), %xmm2
    addsd as1+32(,%edx,8), %xmm2
    mulsd %xmm4, %xmm2
    addsd as1+40(,%edx,8), %xmm2
    mulsd %xmm5, %xmm2
    addsd %xmm2, %xmm7
    mulsd %xmm5, %xmm7
    addsd %xmm7, %xmm6
    movsd %xmm6, as1+24016(,%edx,8)
    cmpl %ecx, %edx
    jne .L1012
```


SIMDinator makes code slower

Kernel#	gcc 4.1.2	gcc+dco	-np	gcc+dco/gcc	-np/gcc	-np/gcc+dco
1	4.96	3.32	4	33.06%	19.35%	-20.48%
2	2.38	2.32	2.32	2.52%	2.52%	0.00%
3	5.93	3.55	3.84	40.13%	35.24%	-8.17%
4	4.66	4.12	3.8	11.59%	18.45%	7.77%
5	5.2	2.07	2.07	60.19%	60.19%	0.00%
6	4.53	3.9	3.63	13.91%	19.87%	6.92%
7	4.87	3.12	3.96	35.93%	18.69%	-26.92%
8	5	5.4	3.88	-8.00%	22.40%	28.15%
9	4.6	3.95	4.23	14.13%	8.04%	-7.09%
10	4.94	3.87	3.38	21.66%	31.58%	12.66%
11	5.78	1.52	1.52	73.70%	73.70%	0.00%
12	5.18	4.98	4.39	3.86%	15.25%	11.85%
13	4.57	5.56	4.58	-21.66%	-0.22%	17.63%
14	4.71	4.71	4.26	0.00%	9.55%	9.55%
15	3.72	3.72	3.72	0.00%	0.00%	0.00%
16	5.61	5.31	5.29	5.35%	5.70%	0.38%
17	5.01	4.99	4.99	0.40%	0.40%	0.00%
18	4.7	3.96	3.95	15.74%	15.96%	0.25%
19	5.81	4.1	4.1	29.43%	29.43%	0.00%
20	4.53	4.43	4.43	2.21%	2.21%	0.00%
21	4.88	4.61	4.61	5.53%	5.53%	0.00%
22	4.88	6.21	4.86	-27.25%	0.41%	21.74%
23	4.17	4.09	4.09	1.92%	1.92%	0.00%
24	4.85	0.77	0.77	84.12%	84.12%	0.00%
Geometric Mean	4.75	3.64	3.53	23.37%	25.68%	3.02%

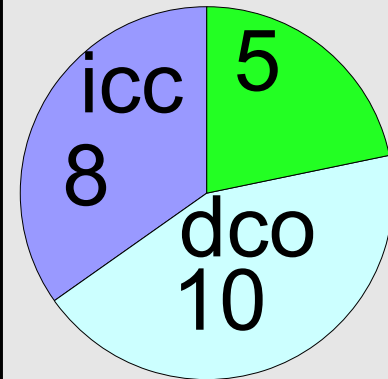
SIMDinator prevents better code generation

Kernel#	gcc 4.1.2	gcc+dco	-np	gcc+dco/gcc	-np/gcc	-np/gcc+dco
1	4.96	3.32	4	33.06%	19.35%	-20.48%
2	2.38	2.32	2.32	2.52%	2.52%	0.00%
3	5.93	3.55	3.84	40.13%	35.24%	-8.17%
4	4.66	4.12	3.8	11.59%	18.45%	7.77%
5	5.2	2.07	2.07	60.19%	60.19%	0.00%
6	4.53	3.9	3.63	13.91%	19.87%	6.92%
7	4.87	3.12	3.96	35.93%	18.69%	-26.92%
8	5	5.4	3.88	-8.00%	22.40%	28.15%
9	4.6	3.95	4.23	14.13%	8.04%	-7.09%
10	4.94	3.87	3.38	21.66%	31.58%	12.66%
11	5.78	1.52	1.52	73.70%	73.70%	0.00%
12	5.18	4.98	4.39	3.86%	15.25%	11.85%
13	4.57	5.56	4.58	-21.66%	-0.22%	17.63%
14	4.71	4.71	4.26	0.00%	9.55%	9.55%
15	3.72	3.72	3.72	0.00%	0.00%	0.00%
16	5.61	5.31	5.29	5.35%	5.70%	0.38%
17	5.01	4.99	4.99	0.40%	0.40%	0.00%
18	4.7	3.96	3.95	15.74%	15.96%	0.25%
19	5.81	4.1	4.1	29.43%	29.43%	0.00%
20	4.53	4.43	4.43	2.21%	2.21%	0.00%
21	4.88	4.61	4.61	5.53%	5.53%	0.00%
22	4.88	6.21	4.86	-27.25%	0.41%	21.74%
23	4.17	4.09	4.09	1.92%	1.92%	0.00%
24	4.85	0.77	0.77	84.12%	84.12%	0.00%
Geometric Mean	4.75	3.64	3.53	23.37%	25.68%	3.02%

faster	29.00%
slower	12.50%
prevents	25.00%

Kernel#	gcc 4.1.2	gcc+dco	icc	gcc+dco/gcc	icc/gcc	icc/gcc+dco
1	4.96	3.32	2.96	33.06%	40.32%	10.84%
2	2.38	2.32	2.28	2.52%	4.20%	1.72%
3	5.93	3.55	2.54	40.13%	57.17%	28.45%
4	4.66	3.8	4.63	18.45%	0.64%	-21.84%
5	5.2	2.07	2.38	60.19%	54.23%	-14.98%
6	4.53	3.63	3.87	19.87%	14.57%	-6.61%
7	4.87	3.12	2.57	35.93%	47.23%	17.63%
8	5	3.88	3.25	22.40%	35.00%	16.24%
9	4.6	3.95	4.97	14.13%	-8.04%	-25.82%
10	4.94	3.38	4.32	31.58%	12.55%	-27.81%
11	5.78	1.52	1.65	73.70%	71.45%	-8.55%
12	5.18	4.39	4.13	15.25%	20.27%	5.92%
13	4.57	4.58	4.61	-0.22%	-0.88%	-0.66%
14	4.71	4.26	2.3	9.55%	51.17%	46.01%
15	3.72	3.72	3.67	0.00%	1.34%	1.34%
16	5.61	5.29	5.66	5.70%	-0.89%	-6.99%
17	5.01	4.99	4.86	0.40%	2.99%	2.61%
18	4.7	3.95	3.45	15.96%	26.60%	12.66%
19	5.81	4.1	6.77	29.43%	-16.52%	-65.12%
20	4.53	4.43	4.38	2.21%	3.31%	1.13%
21	4.88	4.61	1.05	5.53%	78.48%	77.22%
23	4.17	4.09	4.67	1.92%	-11.99%	-14.18%
24	4.85	0.77	1.66	84.12%	65.77%	-115.58%
Geometric Mean	4.74	3.4	3.29	28.27%	30.56%	3.19%

Number of cases



Compiler generated information to aid optimization:

- aliasing
- failed register allocations
- alignment

<http://www.dalsoft.com>

Thank you!