O88: AN OPTIMIZER

for the

C-WARE/DESMET
C COMPILER

Version 3.0

KEY SOFTWARE PRODUCTS

440 Ninth Avenue, Menlo Park, CA 94025
(415) 364-9847

November 1, 1987

TABLE OF CONTENTS

INTRODUCTION

O88 is an optimizer that will reduce both the size and execution time of programs you compile using versions 3.03 and earlier of the C-Ware/DeSmet C compiler, and supports both the small and large memory models. O88 requires PC/MS-DOS 2.0 or later and a minimum of 128k of memory beyond that required for the operating system. No special hardware is required.

O88 can be run as an automatic part of compilation by using the companion driver program CC.EXE, or as a stand-alone program. Optimization options may be controlled by command-line arguments and/or via environment variables, including the ability to turn optimization on and off.

In general, O88 replaces instruction sequences with shorter or faster sequences. It can eliminate unnecessary register loads, dead (unreachable) code, and redundant code sequences. In addition, O88 can compress the data space by folding both string constants and floating-point data.

O88 can optimize for time or space, take advantage of the 80188 instruction set extensions, collapse jump chains, replace floating-point library calls by in-line 8087 instructions, and expand simple library functions in-line.

Since most C functions are always called with a fixed number of arguments (unlike printf), O88 provides an option to convert the C function call model to the more efficient PASCAL model which uses the 8088 **RET n** instruction to clean the stack instead of following every CALL by an **ADD SP,n** instruction.

As an aid to debugging, O88 can insert a call to a user-defined function at the entry point of every function. One example of such a function is provided on the distribution disk as the file HEAPCHEK.C which can be used to intercept runaway stack growth.

In stand-alone mode, O88 provides an option to add comments to its assembly-language output file where code has been deleted, inserted, or otherwise modified.

In normal 8088 mode, O88 will typically eliminate up to about 13% of the instructions and simplify up to about 12% of those that remain. In 80188 mode, O88 will use the 80188 instruction superset to eliminate as much as 20% of the instructions. Depending on which set of options are selected, the final EXE file of typical programs can be reduced in size by 10-20%.

The .O files produced by the PC/MS-DOS version of the C88 compiler are identical to the .O files produced by the CP/M-86 version of the C88 compiler. Therefore, although there is no CP/M-86 version of the optimizer, you *can* compile your .C files using PC/MS-DOS with the optimizer enabled, then transfer the optimized .O files to your CP/M-86 machine and bind them there into a .CMD file.

INSTALLATION

Files on the Distribution Disk

The distribution disk contains the following files:

O88.EXE	The optimizer, version 3.0
CC.EXE	The driver program
WHETSTNE.C	A floating-point benchmark
ACKERMAN.C	A recursion benchmark
DAMPSTNE.C	A general-purpose benchmark
DRYSTONE.C	A fixed-point benchmark
FIBONACI.C	A fixed-point benchmark
HEAPCHEK.C	A function for use with the +H option
MACROS.H	Some #defines to replace library functions

The MACROS.H File

The MACROS.H file contains #defines that can be used to replace the following library functions by in-line code:

isdigit()	**isupper()**	**islower()**	**isascii()**
iscntrl()	**isprint()**	**isalpha()**	**isalnum()**
ispunct()	**tolower()**	**toupper()**	

If you #include this file in your .C source files, the run-time overhead of the library function call will be eliminated at the expense of a slightly larger code space.

This overhead can be significant if the function is called within a tight loop: Entry to the function requires execution of a **PUSH** instruction to pass the argument, followed by a **CALL** instruction; leaving the function requires execution of a **RET** instruction followed by an **ADD SP,n** or **MOV BP,SP** instruction to remove the argument from the stack. Together, these four instructions incur a minimum of 10 extra memory cycles of execution time per function call.

Making a Backup Copy

As with any software package, you should first make a backup copy of the O88 distribution diskette, using the DOS COPY command. With the O88 distribution diskette in drive A: and a freshly formatted diskette in drive B:, enter the following:

```
A>COPY A:*.* B:
```

Updating an Old Version of O88

If (*and only if*) you have previously installed any version of O88 prior to version 3.0, you must first de-install it. Do so by deleting the files O88.EXE and ASM88.EXE, and then rename the file ASM88OLD.EXE to ASM88.EXE (it's original name). For example, if you have installed an earlier version of O88 on your harddisk into a subdirectory called 'C88' where you also keep C88, then enter the following sequence of commands:

```
C>PROMPT $P$G
C:\>CD \C88
C:\C88>DEL O88.EXE
C:\C88>DEL ASM88.EXE
C:\C88>REN ASM88OLD.EXE ASM88.EXE
```

Installing Version 3.0 of O88

Installing version 3.0 of O88 is simply a matter of copying all of the files from the distribution disk onto your compiler disk or hard disk subdirectory where C88 is kept. If your compiler is kept in a harddisk subdirectory called C88, then insert the O88 distribution diskette into drive A: and enter the following commands:

```
C>PROMPT $P$G
C:\>CD \C88
C:\C88>COPY A:*.*/B
```

Testing O88

Enter the following command to verify that O88 has been sucessfully installed and is operating correctly:

```
C:\C88>CC DRYSTONE /+FJPRT
```

You should see the following on your display:

```
C88 Compiler    V2.6    (c) Mark DeSmet, 1982,83,84,85
End of C88     10% Utilization
O88 Optimizer   V3.0   (c) Key Software Products, 1986,87
+FJOPRT:          8%,8% code      0 data          0 calls
ASM88   8088 Assembler    V1.5    (c) Mark DeSmet, 1982-86
end of ASM88        05A4 code   1559 data   4% utilization
C:\C88>
```

Now enter the following two commands to BIND and execute the test program:

C:\C88>BIND DRYSTONE
C:\C88>DRYSTONE

After 10-20 seconds, you should receive a benchmark result on your screen like the following:

Elapsed time for 10000 passes = 6.27
This machine benchmarks at 1666 dhrystones/second

O88's Performance Report

From above, we see that O88 reports its achievements at the end of optimization using the folloowing format :

+FJOPRT: 8%,8% code 0 data 0 calls

The '+FJOPRT' at the far left indicates which of O88's options were enabled during optimization. The two percentages to the left of the word 'code' are represent the reduction and simplification , respectively, of the total number of assembly language instructions. The number to the left of the word 'data' is the number of bytes of constants eliminated by constant folding, and the number to the left of the word 'calls' is the number of library function calls that were replaced by in-line code.

It is important to know that BIND outputs EXE files in increments of 512 bytes; thus, the (apparent) size of small EXE files may not change after optimization. This does *not* mean that the file has not been improved!

Benchmarks

Included on the distribution diskette is the C language source code of five benchmark programs for your enjoyment:

WHETSTNE.C	Floating point benchmark	15.22 secs
DAMPSTNE.C	Mixed: fixed, float, and I/O	9.72 secs
DRYSTONE.C	Fixed point benchmark	6.05 secs
FIBONACI.C	Iteration benchmark	13.44 secs
ACKERMAN.C	Recursion benchmark	39.06 secs

DAMPSTNE includes disk I/O as part of its benchmark, and was thus executed on a ram disk to eliminate any dependence on physical disk characteristics.

These programs were compiled and executed using C88 version 2.6 under DOS 3.1 on a Compaq Portable II (8 Mhz 80286 with 1 wait state and an 80287 installed). The times shown are the average (out of 10 trials) execution times in seconds on that machine *without* optimization, and will give you some idea of how long to expect the benchmark programs to run.

Once you are familiar with all of O88's options, try recompiling these programs with O88 enabled, and vary the options to see if you can do better than the times given above.

THE DRIVER PROGRAM

CC is a program that integrates execution of C88, O88, and ASM88. If optimization is enabled CC wil: (1) run C88 to generate an assembly language file (.A), (2) run O88 to optimize it, and (3) run ASM88 to assemble it. If optimization is disabled, CC will simply run C88 to compile your program without running O88.

The command line arguments of CC are identical to those of C88, except that there's a provision to have CC pass some of its command line arguments to O88. In particular, anything on the command line after a forward slash (/) is used as command line arguments to O88, and everything before the slash is passed to C88. For example, the CC command,

CC MYPROG TD /+PT -17 SINC\PROG.STK

passes the arguments 'MYPROG TD' to C88 and the arguments '+PT -17 SINC\PROG.STK' to O88. O88's use of these arguments is explained in a later chapter.

If the 'A' option of C88 is used on the CC command line to request that C88 generate an assembly language (.A) file instead of an object (.O) file, then CC will not execute either O88 or ASM88. If you wish to obtain an optimized version of this .A file, you must then run O88 in the stand-alone mode to optimize it.

Using CC with a 'MAKE' Program

CC fixes a minor problem with C88 and ASM88. These programs always create a .O file, even if there were errors during the compilation or assembly. This can be a nuisance if you are using a Unix-like MAKE program which looks at the time and date stamps on files to determine what needs to be recompiled.

Since the .O file will be newer than the .C file, MAKE won't think a recompile is necessary unless you update the .C file. This becomes a problem if you use ^C (or control-break) to abort a compilation. The same dilema occurs if your compilation gets through the compiler but fails to assemble properly (it *is* possible).

CC examines the exit code of C88, O88, and ASM88 to determine if each operation was successful. If not (exit code not zero), CC deletes the corresponding .O file if one is present. This allows MAKE programs to operate correctly, since MAKE will be forced to recompile.

O88's Use of C88's '-C' Option, and D88

The -C options of C88 and BIND are used to generate .CHK files for use with D88. When C88 is given the -C option, it inserts **LINE** statements in the assembly language output file to mark the beginning of the code generated for one line of C code.

Unfortunately, O88 is incompatible with D88. That's because the optimizer rearranges the code in a way that D88 doesn't understand. If you need to use D88, turn off the optimizer!

However, the **LINE** statements generated by C88's -C option help O88 to understand the structure of the code, and can produce additional optimization. Since the optimized code is incompatible with D88

anyway, O88 will eliminate the **LINE** statements from its output.

Thus the driver program (CC.EXE) always enables the -C option of C88 so that O88 can benefit from the additional information. However, since the **LINE** statements are removed from by O88, the driver program does not enable the -C option of ASM88, and so no .CHK file is generated.

Note, however, that if you prepare a .A file using C88's -A option, and then choose to optimize it using the stand-alone mode of O88, you will get better optimization if you use the -C option when you run C88.

STAND-ALONE OPERATION

Sometimes it can be useful to execute O88 directly, without using CC. For example, stand-alone operation of O88 can be used to generate an optimized assembly language file with comments added by O88 where optimizations were applied.

The command line of O88 may be used to specify the input file name, to direct the optimized output to a file or to the display, and to select the optimization options.

Specifying Input and Output Files

O88 uses 'standard in' (the keyboard) and 'standard out' (the display) for its input and output, respectively. Typical use of O88 therefore requires that I/O redirection be specified on the command line to process files. For example, the O88 command,

O88 <SLOW.A >FAST.A

may be used to optimize the input file named 'SLOW.A', generating an optimized output file named 'FAST.A'. If the output redirection is omitted, as in

O88 <SLOW.A

the output will be sent to the display instead of a file.

If the input redirection is omitted (no source file to optimize), O88 will display all the current option settings. The keyboard ***cannot*** be used to supply unoptimized assembly language input to O88.

Specifying Options on the Command Line

Every option of O88 has an associated letter or digit. An option may be *enabled* on the command line of O88 by using its character preceded by a plus ('+') sign, or *disabled* by preceding the character by a minus ('-') sign. Those options which use a letter may be specified by either an upper or lower case letter.

For example, the O88 command

O88 <SLOW.A >FAST.A +P +T -1 -7

enables options P and T and disables options 1 and 7.

More than one option may be enabled or disabled at the same time by concatenating their associated letters or digits and preceding the group of characters by a single plus or minus sign:

O88 <SLOW.A >FAST.A +PT -17

These options and others are discussed in detail in a later chapter.

The +S Option and its Companion File

One of the new options (+S) provided with version 3.0 of O88 requires the use of an associated text file. The name of this file must be specified by a string beginning with the letter 'S' (*without a plus sign*), followed immediately by the corresponding filespec:

O88 <SLOW.A >FAST.A +S SINC\PROG.STK

The command line above enables the +S option, and uses the S directive to specify a file called PROG.STK (located in the INC subdirectory of the current directory) to be used with the +S option.

HINT

The file associated with the +S option is specified separately so that the option may be disabled and then later re-enabled (via environment variables) without changing the CC or O88 command line.

Multiple Passes & Temporary Files

O88 makes multiple passes over the assembly language, requiring the use of several temporary files. You can specify that these files be placed on your RAM disk for faster optimization by using the letter 'T' followed by the drive letter and a colon:

O88 <SLOW.A >FAST.A TD:

NOTE: *No* space is allowed between the 'T' and the 'D:'; this use of 'T' is *not* preceded by a '+' or '-' sign.

The command line above specifies that these temporary files be placed on drive D (presumably the RAM disk). In reality, everything after the 'T' is used as a filename *prefix* (the colon is required); thus you can place these files in a subdirectory in the following manner:

O88 <SLOW.A >FAST.A TC:\SCRATCH

HINT

If the driver program (CC.EXE) is used to compile and optimize your program, and you specify the ramdisk option to C88 (e.g., -TD), CC.EXE will automatically prepare and use an equivalent option for O88, and thus there is no need to also specify that option for O88 on the CC command line.

ENVIRONMENT VARIABLES

O88 provides two environment variables (O88 and O88.1) to control the operation of the optimizer. With the exception of I/O redirection, anything that appears on the O88 command line can be specified by one of the environment variables instead.

The two environment variables are O88 and O88.1, with O88 taking precedence. Options specified by command line arguments have even higher precedence.

For example, the default values of certain options may be set before compiling or optimizing by using the DOS SET command:

```
A>SET O88.1=-17 +PT
```

These options may then be omitted from the command line arguments.

HINT

It is often convenient to use environment variable O88.1 to establish the operating modes for the optimizer, and then use environment variable O88 to enable and disable the optimizer without changing the modes.

I.e., to disable optimization, use the DOS command:

```
A>SET O88=-O
```

and then to re-enable optimization, use the DOS command:

```
A>SET O88=+O
```

OPTIMIZATION OPTIONS

O88 supports a set of options to control optimization. Each option has a default setting, but these may be overridden by either command line arguments or environment variables.

The entire set of options and their defaults are described in this section. You can get a list of what defaults are in effect at any time by running 'O88' with no command line arguments.

The 'A' Option (default is -A)

+A ALLOCATE maximum buffers (may overlay COMMAND.COM)

- A Do NOT allocate maximum buffers (preserve COMMAND.COM)

At the very top of memory, DOS keeps a copy of the transient portion of COMMAND.COM. This area is where COMMAND.COM keeps the data it needs to implement recall of the last command line using function key F3. If a program writes over this area, the data there will be re-initialized by COMMAND.COM when the program terminates, thus temporarily defeating the use of F3.

O88 uses up to 128K of DOS memory for I/O buffers. If these buffers overlay the very top of memory they will affect F3. This usually doesn't happen unless you have less than 256k of available memory for O88 to run in.

If you enable this option (+A), it means that you are willing to sacrifice the use of F3 in favor of using that extra memory for buffer space. Otherwise, O88 will preserve it.

The 'B' Option (default set by C88 option, else defaults to -B)

+B Optimize code for the BIG (large) memory model

- B Do NOT optimize for the big (large) memory model

To simplify processing, O88 replaces every **CALL** and **LCALL** instruction mnemonic by **_CALL** and replaces every **RET** and **LRET** by **_RET** so that the internal pattern matching can be independent of whether the code was compiled for the small or large memory model. In order that the code still assemble properly, O88 equates these two symbols at the beginning of the output file according to whether this option is enabled or disabled.

For example, if this option is disabled (-B), O88 inserts:

```
_CALL EQU CALL
_RET EQU RET
```

And if this option is enabled (+B), O88 inserts:

```
_CALL EQU LCALL
_RET EQU LRET
```

Recent versions of ASM88 provide pseudo-op's for conditional assembly based on large versus small memory model use, and this feature could have been used by O88 to automatically select the appropriate set of equates rather than requiring that +/-B be specified. However, older versions of ASM88 don't support conditional assembly. This is why is this option is necessary in order to let O88 know which model is in use.

When the driver program (CC.EXE) is used, it automatically sets this O88 option based on the absence or presence of the corresponding C88 option specified on the CC command line. Thus the only time you really need to specify +/-B yourself is when you use O88 in the stand-alone mode.

The 'C' Option (default is -C)

+C Add COMMENTS where optimizations are made

- C Do NOT add optimization comments to the output

Enabling this option (+C) causes the assembly language output to be appropriately commented where O88 has made changes. (An example of these comments can be seen in the discussion of the next option.) The +C option is always disabled by the driver program (CC.EXE), and is thus only useful when O88 is run in stand-alone mode.

The 'E' Option (default is -E)

+E EXPAND non-f.p. library function calls in-line

- E Do NOT expand (non-f.p.) library function calls

Enable this option if you want calls to the following library functions expanded into in-line code:

_showds()	**_showcs()**	**_showsp()**
_inb()	**_inw()**	**_outb()**
_outw()	**_peek()**	**_poke()**

Shown below is an example of how O88 would replace a call to the library function _peek() by in-line code when this option is enabled (+E). The +C option has been used here to add comments to the file produced by O88 so that you can see how it performs its job. Instructions inserted by O88 are indicated by a comment of '{O88}' appended to the end of the line. Note that even the instructions output by O88 are candidates for further optimization.

```
;       Instructions combined --> PUSH WORD seg_
;       Instructions combined --> PUSH WORD off_
;         Instruction replaced --> _CALL _peek_
;       Instructions combined --> POP DI ; {O88}
 MOV DI,WORD off_ ; {O88}
;       Instructions combined --> POP ES ; {O88}
 MOV ES,WORD seg_ ; {O88}
 MOV AL,BYTE ES:[DI] ; {O88}
 XOR AH,AH ; {O88}
;        Unnecessary, deleted --> ADD SP,4
```

NOTE: This option has no effect on expansion of calls to the floating-point library (see option '7') or the in-line expansion of functions _move() or _lmove() (see option 'M').

IMPORTANT: This option replaces the above library function calls by in-line code. For these functions, it is imperative that your C source code always use the proper number of arguments. The functions that O88 replaces by in-line code, and the corresponding number of arguments that are expected are as follows:

Function	Arguments	Type
_inb()	1	unsigned
_inw()	1	unsigned
_outb()	2	char, unsigned
_outw()	2	unsigned, unsigned
_peek()	2	unsigned, unsigned
_poke()	3	char, unsigned, unsigned
_showcs()	none	
_showds()	none	
_showsp()	none	

The 'F' Option (default is -F)

+F FOLD duplicate constants in data segment

- F Do NOT fold duplicate constants in dseg

When enabled, this option eliminates duplicate compiler-generated string and floating-point constants:

```
__6 DB 'Hello',0
_F1 DW 08667H,0F01BH,21F9H,4009H
;    Duplicate string or real -->__7 DB 'Hello',0
__7 EQU __6 ; {O88}
;    Duplicate string or real -->_F2 DW 08667H, ...
_F2 EQU _F1 ; {O88}
```

This option affects all floating-point constants; however, only those string constants that are not specified as the initial values of static character arrays are affected. For example, the string in:

```
static char a[] = "ignored by +F" ;
```

is ignored by +F whereas the strings in the following examples may be replaced by a pointer to a duplicate copy if one exists:

```
char *ptr = "processed by +F" ;
strcpy(bfr, "processed by +F") ;
```

With constant folding enabled (+F), the two lines of code above would effectively be replaced by:

```
char *ptr = "processed by +F" ;
strcpy(bfr, ptr) ;
```

You should *not* enabled this option if you have code that modifies such strings! For example, the following should preclude use of +F:

```
char *ptr = "Hello" ;
strcpy(ptr, "Bye") ;
printf("Hello") ;
```

With constant folding enabled (+F), you should expect that the printf above will print "**Bye**", not "**Hello**"! If you have code like this and must modify such strings, simply change the declarations to arrays instead of a pointers:

```
static char ptr[] = "Hello" ;
strcpy(ptr, "Bye") ;
printf("Hello") ;
```

The 'H' Option (default is -H)

+H Add CALL _HELP_ at entrance of every function

- H Do NOT add CALL _HELP_ at function entrance

This option causes a **_CALL _HELP_** to be inserted at the entrance of every function, placed immediately after the **PUSH BP**/**MOV BP,SP** function prologue:

```
My_Function_: PUSH BP
 MOV BP,SP
 _CALL _HELP_ ; {O88}
 SUB SP,100  <-- (Allocate automatic variables, if any)
   ...
   ...
   ...
```

The implementation of function _HELP() is left to the user, with the only restriction being that it must not have any arguments, nor return any value.

One possible use is as a debugging aid to trap runaway recursion (unbounded stack growth) that would otherwise overrun the heap in the small memory model. Source code for such a function is provided on the distribution disk as the file HEAPCHEK.C. This file is heavily commented, and should be used as a model for writing other _HELP() functions.

The 'I' Option (default is +I)

+I INITIALIZE the 8087/80287 with FINIT at main()

- I Do NOT initialize the 8087/80287 with FINIT

NOTE: This option uses the letter 'eye' (for init), not the digit 'one'.

This option is relevant only when the '7' option is enabled (+7).

The C-Ware/DeSmet 8087 library (CSTDIO7.S) checks to see if the 8087 is initialized before every 8087 **FLD** instruction within those library routines that push values onto the internal stack of the 8087. O88 replaces these (and other) library routines by in-line 8087 instructions, and thus eliminates the initialization.

If 8087 optimization is enabled (+7), O88 will normally (+I) insert an 8087 **FINIT** instruction at the beginning of function main( ) to initialize the 8087:

```
main_: PUSH BP
 MOV BP,SP
 FINIT ; {O88}
 ...
 ...
 ...
```

If this option is disabled (-I), then the user should supply the initialization by creating a special assembly-language routine and binding it in.

The 'J' Option (default is -J)

+J Collapse JMP chains (Remove JMPs to JMPs)

- J Do NOT collapse JMP chains (JMPs to JMPs)

The compiler sometimes generates code containing a **JMP**, to a **JMP**, to a **JMP**, etc. This option replaces the **JMP** destination labels to avoid execution of more than one **JMP** in sequence:

```
_L4:
 JMP _L2
 ...
 ...
 ...
;    Jump destination replaced --> JMP _L4
 JMP _L2 ; {O88}
```

The 'M' Option (default is -M)

+M Replace CALL _MOVE/_LMOVE by inline REP MOVS

- M Do NOT replace CALL _MOVE/_LMOVE by REP MOVS

When enabled, +M expands calls to the library functions _move() and _lmove() into inline **REP MOVS** code:

```
;       Instructions combined --> MOV AX,OFFSET to_bfr_
;       Instructions combined --> PUSH AX
;       Instructions combined --> PUSH WORD fm_ptr_
;       Instructions combined --> MOV AX,127
;       Instructions combined --> PUSH AX
;         Instruction replaced --> _CALL _move_
;       Instructions combined --> POP CX ; {O88}
;         Instruction replaced --> MOV CX,127 ; {O88}
 MOV CX,63 ; {O88}
;       Instructions combined --> POP SI ; {O88}
 MOV SI,WORD fm_ ptr_; {O88}
;       Instructions combined --> POP DI ; {O88}
 MOV DI,OFFSET to_bfr_ ; {O88}
 MOV AX,DS ; {O88}
 MOV ES,AX ; {O88}
 CLD ; {O88}
;        Instruction replaced --> REP MOVSB ; {O88}
 REP MOVSW ; {O88}
 MOVSB ; {O88}
;        Unnecessary, deleted --> ADD SP,6
```

This option is controlled separately from other inline expansions (+/-E and +/-7) of library functions because there is a special requirement for this replacement: The library functions _move() and _lmove() allow the source and destination to overlap; the replacement by an inline **REP MOVS** sequence does *not*!

The replacement code always performs the copy starting with the lowest memory address and working towards the highest. This can give unexpected results if the source and destination overlap, and the destination begins at a higher memory location than the source. For example, the following code would be corrupted by enabling (+M) this option, because it would propagate the value of **bfr[0]** throughout the array:

```
char bfr[100] ;
_move(99, &bfr[0], &bfr[1]) ;
```

This problem is avoided by the library functions because they compare the source and destination addresses and reverse the direction of the copy if necessary.

HINT

This option is very useful if you have functions that pass or return structures (or unions), since the compiler generates a **CALL _MOVE_** to do the copy, and these transfers never involve an overlapping source and destination.

The 'O' Option (default is +O)

+O OPTIMIZE! - the code will be modified

- O Do NOT optimize - code will NOT be modified

NOTE: This option uses the letter 'oh' (for optimize), not the digit 'zero'.

This option is normally used with an environment variable to enable or disable optimization during compilation with the CC command. It has no effect when O88 is used without CC.

HINT

Use environment variable O88.1 to set the desired options, and then set the environment variable O88 to + or - O to turn optimization on or off without modifying the other options.

The 'P' Option (default is -P)

+P Assume that POINTERS never point to themselves

- P Do NOT assume that pointers never point to themselves

When O88 examines the sequence of instructions in the input file, it tries to remember what has been loaded into the CPU registers so that it can replace memory references with register references whenever possible. However, O88 is forced to forget what it knows about the content of the registers whenever it encounters an instuction that uses register SI, DI, or BX to select and modify a memory location. This is necessary because the optimizer has no idea where the register points and thus can't tell if the modified memory location's content had been in any of the CPU registers. This applies to the content of all registers except those that contain constants.

Enabling this option (+P) causes the optimizer to preserve its knowledge about the contents of that register which is used as a pointer, and can be useful for C code fragments like:

```
structure_pointer->member1 = 126 ;
structure_pointer->member2 = -34 ;
                .
                .
structure_pointer->member9 = 936 ;
```

Each line of the C code shown above loads 'structure_pointer' into an index register before every reference. The '+P' option allows the optimizer to eliminate the redundant reloading of this register.

Note that similar (but less obvious) code is generated whenever the name of an array is passed as an argument to a function. Inside the function, the name of the array is treated as a pointer to the base address of the array, and the subscript is converted to an offset from the base.

Of course, enabling this option (+P) requires that the register never point to the memory location from which its contents was obtained! Such a situation rarely occurs in compiled code since it requires that a pointer be loaded with its own address. (Beware of empty circular linked lists!)

The 'R' Option (default is -R)

+R Remove REDUNDANT code (O88 runs very slowly!)

- R Do NOT remove redundant code (O88 runs faster)

When enabled, this option looks for duplicate code fragments and replaces subsequent copies by a **JMP** to the first:

```
_O88_1: ; {O88}
 INC WORD x_
 MOV AX,WORD [BP-2]
 MOV SP,BP
 POP BP
 RET
 ...
 ...
 ...
;       Redundant, eliminated --> INC WORD x_
;       Redundant, eliminated --> MOV AX,WORD [BP-2]
;       Redundant, eliminated --> MOV SP,BP
;       Redundant, eliminated --> POP BP
;       Redundant, eliminated --> RET
 JMP _O88_1 ; {O88}
```

Such code fragments necessarily end with either a **RET** or **JMP** instruction; this implies that maximum optimization occurs when C source code is arranged so that (to the extent possible) the last few lines of each function (in the same file) are identical.

For example, the following code would benefit from +R optimization:

```
int x ;

fun1()
        {
        int a ;
        .
        .
        x = a + 1 ;
        return a ;
        }

fun2()
        {
        int b ;
        .
        .
```

```
x = b + 1 ;
return b ;
}
```

Note that references to 'a' and 'b' above generate identical assembly language code since they share the same relative positions in the set of automatic storage class (temporary) variables of each function.

NOTE: This is a space (not time) optimization. Moreover, the optimization process is quite time consuming, and grows exponentially with the file size. Try it with small files first to get an idea of how long it runs before using this option with large files.

The 'S' Option (default is -S)

+S Clean STACK before function return

- S Do NOT clean stack at function return

Since C allows functions to be called with a variable number of arguments, it is necessary that the calling function remove any arguments from the stack that it passed to the called function, rather than letting the function remove them. This is why C88 uses an **ADD SP,n** or **MOV SP,BP** after **CALL**'s when arguments are passed.

PASCAL does not allow functions to be called with a variable number of arguments, and thus called functions can clean the stack themselves using the **RET n** instruction. This means less code is needed since the C function model requires an **ADD SP,n** or **MOV SP,BP** each time the function is called, rather than a single **RET n** at the end of the function.

What becomes interesting is that most C programmers rarely create functions with a variable number of arguments! If we can specify which functions never use a variable number of arguments, the optimizer can change the code generated for these functions so that they clean the stack within the called function (like PASCAL) instead of after the **CALL**.

To use this option, you must first create a text file containing one line for each of your C functions (see exceptions below). Each line must contain exactly two items: Starting in column 1 is the name of the function. Capitalization must exactly match that used in the C source code, and the function name must include the trailing underscore ('_') that the compiler appends to the end. The second item is seperated from the first by one or more spaces or tabs, and is the number of bytes pushed onto the stack in order to pass the function arguments.

The number of bytes per argument can be determined from the following:

char	2 bytes (always passed as int)
int	2 bytes
unsigned	2 bytes
pointer	2 bytes (4 for large case)
array	2 bytes (4 for large case)
long	4 bytes
double	8 bytes
float	8 bytes (always passed as double)
struct	sizeof(struct)

For functions that return a structure (not just a pointer to a structure), add 2 bytes (4 for large case) to the number of bytes pushed on the stack to pass the parameters.

EXCEPTIONS!

(1) Functions that don't have arguments and don't return a structure cannot benefit from this optimization, and so should not be included. (The number of bytes to be cleaned from the stack is zero, and C88 will not generate an **ADD SP,n** after the **CALL**.)

(2) Library functions must not be included since although the optimizer can change their **CALL**, it cannot change their **RET**.

(3) If you use a pointer to call a function, the corresponding **CALL** instruction will not reference the function by name. Consequently, there is no way that O88 can recognize the **CALL** in order to remove the **ADD SP,n** that follows it. Thus any function which is called indirectly through a pointer must not be included.

The effect of using this option is illustrated in the following example:

```
 PUSH WORD [BP+4]
;   Function now cleans stack --> _CALL My_Function_
;   Function now cleans stack --> ADD SP,2
 _CALL My_Function_ ; {O88}
...
...
My_Function_:
...
...
 POP BP
;   Function now cleans stack --> RET
 RET 2 ; {O88}
```

In general, it is *not* a good idea to use the **RET n** instruction in your own assembly language functions, even if you include them in this file. One day you may try to optimize with the 'S' option disabled, and then the **CALL**'s won't match the **RET**'s:

For example, writing assembly language code like the following is *not* recommended:

```
; void fun(arg)
; int arg ;

        cseg    fun_
        public  fun_
fun_:   push    bp
        mov     bp,sp
        ...
        mov     sp,bp
        pop     bp
        ret     2; discard the argument
```

On the other hand, the #asm option of C88 may be used, and then this function may be included in the file and optimized with the +S option:

```
void fun(arg)
int arg ;
        {
#asm
```

```
        ...
#end
        }
```

NOTE: If the +S option is enabled, you need to tell O88 the filename that contains the function names and the number of bytes pushed as arguments.

I.e., This optimization is *only* enabled when you:

(1) Enable this optimization using '+S', and
(2) Specify a file using 'S<filespec>'.

The reason *both* 'S<filespec>' and '+S' are required is to allow you to set-up a makefile in which the 'S<filespec>' is specified on the CC.EXE command lines and still be able to turn this option on and off (via environment variables) without editing the makefile.

WARNING!

If your program is built from several seperately compiled source files, and there are functions in one file that are called by functions in other files, then it is imperative that either (1) all files are optimized with +S (enabled), or (2) all files are optimized with -S (disabled). Failure to do so will corrupt the stack during execution and your program will definitely crash!

This also suggests that a single file should be created that contains the function names and number of bytes pushed, and that this file be used with the +S option when optimizing all source code modules of the program.

The 'T' Option (default is -T)

+T Optimize for TIME (as opposed to space)

- T Do NOT optimize for time (optimize for space)

When enabled (+T), this option enables four optimizations:

(1) Multiplication by a constant is replaced by an equivalent but faster sequence of **SHL**, **ADD**, and **SUB** instructions:

```
MOV AX,WORD subscript_
;   Strength reduction applied --> MOV SI,14
;   Strength reduction applied --> IMUL SI
MOV DX,AX ; {O88}
SHL AX,1 ; {O88}
SHL AX,1 ; {O88}
SHL AX,1 ; {O88}
SUB AX,DX ; {O88}
SHL AX,1 ; {O88}
```

(The 80188/80286 multiply is approximately four times faster than that of the 8088. Thus if 80188 mode is enabled, the optimizer decides to expand or not depending on a calculated estimate of the number of clock cycles required with and without expansion.)

(2) Division by a power of two is replaced by an equivalent but faster shift sequence:

```
;   Strength reduction applied --> XOR DX,DX
;   Strength reduction applied --> MOV CX,256
;   Strength reduction applied --> DIV CX
;       Instructions combined --> MOV CL,8 ; {O88}
;       Instructions combined --> SHR AX,CL ; {O88}
MOV AL,AH ; {O88}
MOV AH,0 ; {O88}
```

(3) Remaindering an integer by a power of two is replaced by an equivalent but faster sequence of masking instructions:

```
;   Strength reduction applied --> XOR DX,DX
;   Strength reduction applied --> MOV CX,256
;   Strength reduction applied --> DIV CX
;       Instructions combined --> MOV DX,AX ; {O88}
;       Instructions combined --> AND DX,255 ; {O88}
;       Instructions combined --> MOV WORD i_,DX
;      Size reduction applied --> AND AX,255 ; {O88}
;   Strength reduction applied --> AND AH,0 ; {O88}
;   Strength reduction applied --> MOV AH,0 ; {O88}
XOR AH,AH ; {O88}
```

(4) Static word operands in the data segment are forced to align on even addresses by preceding them with the **EVEN** directive of ASM88. This benefits CPU's with a 16-bit (or wider) data bus (i.e., 8086, 80186, 80286, 80386, V20, and V30), and has no effect on those with an 8-bit bus.

These optimizations may cause some expansion of both the code and data segments, but the elimination of the multiplications and divisions will significantly reduce execution time on 8088's and 8086's.

Many multiplications and divisions generated by the compiler are not always obvious! Multiplication by a constant is generated by the compiler to subscript arrays whose members are larger than four bytes. Division by a constant is generated when your source code computes the difference between two pointers, (e.g., a pointer and the base of an array) where the constant reflects the size of the objects pointed to.

HINT

> Since division can only be replaced by shifting when the divisor is a power of two, you might want to add some dummy members to your structures (those that are used in an array of structures) so that the number of bytes occupied by each structure in the array is a power of 2. Not only will this benefit computing the difference between two structure pointers, but it has the added advantage that when you subscript these arrays, the implied multiplication is replaced by a simple multi-bit shift (with no additions or subtractions).

Note: If time optimization is disabled (-T), then these optimizations will still be used if they cause no increase in code space.

The '1' Option (default is -1)

+1 Replace with 80188/186 instructions where possible

- 1 Do NOT use 80188/186 instructions - only 8088/8086

NOTE: This option uses the digit 'one' (for 80188), not the letter 'el'.

When this option is enabled, the optimizer generates 80188 instructions as constants (ASM88 doesn't recognize the 80188 instruction superset) inserted into the code segment using **DW** and **DB** directives:

```
;       Instructions combined --> PUSH BP
;       Instructions combined --> MOV BP,SP
 DB 200,0,0,0 ;  ENTER 0,0
 ...
 ...
;        Instruction absorbed --> MOV CL,4
;        Instruction replaced --> SHL AX,CL
 DB 193,224
 DB 4 ;  SHL AX,4
 ...
 ...
;       Instructions combined --> MOV AX,OFFSET _L3
;       Instructions combined --> PUSH AX
 DB 104
 DW _L3 ;  PUSH OFFSET _L3
 ...
 ...
;       Instructions combined --> MOV SP,BP
;       Instructions combined --> POP BP
 DB 201 ;  LEAVE
```

INCOMPATIBILITY WITH OLD VERSIONS OF ASM88

Early versions of ASM88 (compiler versions prior to version 2.4) will reject the **DW** reference to **_L3**. Consequently, the +1 option cannot be used with these older versions. Contact C-Ware corporation for information on how to upgrade to the latest version of C88.

OLDER VERSIONS OF O88 ARE DIFFERENT

Unlike earlier versions of O88, version 3.0 does not set the default value of this option according to whether or not an 80188-compatible processor is installed.

The '7' Option (default is -7)

+7 Replace f.p. CALLs by in-line 8087/80287 code

- 7 Do NOT replace floating-point library CALLs

When enabled, this option replaces calls to the floating-point library by inline 8087 instructions:

```
;       Unnecessary, deleted --> MOV SI,OFFSET b_
;         Instruction replaced --> _CALL _FLOADD
;         Instruction replaced --> FLD QWORD [SI] ; {O88}
 FLD QWORD b_ ; {O88}
;       Unnecessary, deleted --> MOV SI,OFFSET _F1
;         Instruction replaced --> CALL _FLOADD
;         Instruction replaced --> FLD QWORD [SI] ; {O88}
;         Instruction replaced --> FLD QWORD _F1 ; {O88}
 FLD1 ; {O88}
;         Instruction replaced --> _CALL _FADD
 FADD ; {O88}
;       Unnecessary, deleted --> MOV SI,OFFSET a_
;         Instruction replaced --> _CALL _FSTORED
;         Instruction replaced --> FSTP QWORD [SI] ; {O88}
 FSTP QWORD a_ ; {O88}
 FWAIT ; {O88}
```

IMPORTANT!

Your C-Ware/DeSmet compiler is supplied with *two* versions of the function library, one for use with an 8087 floating-point chip (CSTDIO7.S/BCSTDIO7.S), and another (CSTDIO.S/BCSTDIO.S) which implements the floating point library entirely in 8088 code. If you compile with the 8087 option enabled (+7), you *must* use the 8087 version of the library, CSTDIO7.S. Failure to do so will cause your program to crash!

OLDER VERSIONS OF O88 ARE DIFFERENT

Unlike earlier versions of O88, version 3.0 does not set the default value of this option according to whether or not an 8087 coprocessor is installed.

O88 AND THE ASSEMBLER

The algorithms used in O88 assume that its input is the output of the compiler's code-generator (GEN.EXE), and thus O88 takes advantage of the expected characteristics of the code. Do *not* try to use O88 with assembly language programs *you* have created! Doing so will probably cause O88 to generate bad code.

If you *do* optimize your own assembly language code, you will very likely get a message such as:

Reg [SI] contents undefined --> MOV AX,WORD [SI]

on the console. This means that O88 has detected code that is referencing a register (SI in this example) which O88 thinks has not been loaded with any data (probably as the result of trying to optimize assembly language code *not* produced by GEN).

If this message appears, you can be sure that the optimized output file is invalid. In that case, use O88 in stand-alone mode with the +C option enabled (comments added) and search the output file for the context in which the message occurs. (The same error message will appear in the output file.) You might then be able to edit the output file by hand to try to correct the error.

On the bright side, GEN outputs assembly language code in uppercase only, and O88 ignores any code written in lowercase. Thus if you use the in-line #asm option of the compiler in programs that go through the optimizer, be sure to use only lowercase assembly language mnemonics so that O88 will not attempt to modify the code.

For best results when using the #asm directive, don't depend on C statements to leave either the registers or flags in any particular state. Better yet, don't mix assembly language with C code in the same function.